STCmicro™
宏 晶 科 技

STC8H 系列单片机
技术参考手册

资料更新日期：**2018/12/08**

目录

1 概述 1
2 特性及价格 2
2.1 STC8H1K16 系列特性及价格 2
2.2 STC8H1K08 系列特性及价格 4
3 管脚及说明 6
3.1 管脚图 6
3.1.1 STC8H1K16 系列管脚图 6
3.2 管脚说明 7
3.2.1 STC8H1K16 系列管脚说明 7
3.3 功能脚切换 11
3.4 范例程序 13
3.4.1 串口 1 切换 13
3.4.2 串口 2 切换 14
3.4.3 串口 3 切换 15
3.4.4 串口 4 切换 15
3.4.5 SPI切换 16
3.4.6 I2C切换 16
3.4.7 比较器输出切换 17
3.4.8 主时钟输出切换 18
4 封装尺寸图 19
4.1 LQFP32 封装尺寸图（9mm*9mm） 19
4.2 STC8H系列单片机命名规则 20
5 ISP下载及典型应用线路图 21
5.1 STC8F系列ISP下载应用线路图 21
5.1.1 使用RS-232 转换器下载 21
5.1.2 使用PL2303-SA下载 22
5.1.3 使用PL2303-GL下载 23
5.1.4 使用U8-Mini工具下载 24
5.1.5 使用U8W工具下载 25
5.1.6 USB直接ISP下载 26
5.2 STC8A系列ISP下载应用线路图 27
5.2.1 使用RS-232 转换下载（使用高精度ADC） 27
5.2.2 使用RS-232 转换下载（ADC一般应用） 28
5.2.3 使用PL2303-SA下载 29
5.2.4 使用PL2303-GL下载 30
5.2.5 使用U8-Mini工具下载 31
5.2.6 使用U8W工具下载 32
5.2.7 USB直接ISP下载 33
6 时钟、复位与电源管理 34
6.1 系统时钟控制 34

6.2 STC8H系列内部IRC频率调整 37
6.3 系统复位 40
6.4 系统电源管理 42
6.5 范例程序 43
6.5.1 选择系统时钟源 43
6.5.2 主时钟分频输出 44
6.5.3 看门狗定时器应用 45
6.5.4 软复位实现自定义下载 46
6.5.5 低压检测 47
6.5.6 省电模式 48
6.5.7 使用INT0/INT1/INT2/INT3/INT4 中断唤醒MCU 49
6.5.8 使用T0/T1/T2/T3/T4 中断唤醒MCU 51
6.5.9 使用RxD/RxD2/RxD3/RxD4 中断唤醒MCU 55
6.5.10 使用LVD中断唤醒MCU 57
6.5.11 CMP中断唤醒MCU 58
6.5.12 使用LVD功能检测工作电压（电池电压） 59
7 存储器 64
7.1 程序存储器 64
7.2 数据存储器 65
7.2.1 内部RAM 65
7.2.2 内部扩展RAM 66
7.3 存储器中的特殊参数 67
7.3.1 读取Bandgap电压值 (从ROM中读取) 68
7.3.2 读取Bandgap电压值 (从RAM中读取) 70
7.3.3 读取全球唯一ID号 (从ROM中读取) 72
7.3.4 读取全球唯一ID号 (从RAM中读取) 75
7.3.5 读取 32K掉电唤醒定时器的频率 (从ROM中读取) 77
7.3.6 读取 32K掉电唤醒定时器的频率 (从RAM中读取) 79
8 特殊功能寄存器 82
8.1 STC8H1K16 系列 82
8.2 特殊功能寄存器列表 83
9 I/O口 88
9.1 I/O口相关寄存器 88
9.2 配置I/O口 92
9.3 I/O的结构图 93
9.3.1 准双向口（弱上拉） 93
9.3.2 推挽输出 93
9.3.3 高阻输入 93
9.3.4 开漏输出 94
9.4 范例程序 95
9.4.1 端口模式设置 95
9.4.2 双向口读写操作 96
10 指令系统 98
11 中断系统 102

11.1 STC8H系列中断源 102
11.2 STC8H中断结构图 104
11.3 STC8H系列中断列表 105
11.4 中断相关寄存器 107
11.4.1 中断使能寄存器（中断允许位） 107
11.4.2 中断请求寄存器（中断标志位） 111
11.4.3 中断优先级寄存器 113
11.5 范例程序 116
11.5.1 INT0 中断（上升沿和下降沿） 116
11.5.2 INT0 中断（下降沿） 117
11.5.3 INT1 中断（上升沿和下降沿） 117
11.5.4 INT1 中断（下降沿） 118
11.5.5 INT2 中断（下降沿） 119
11.5.6 INT3 中断（下降沿） 120
11.5.7 INT4 中断（下降沿） 121
11.5.8 定时器 0 中断 122
11.5.9 定时器 1 中断 123
11.5.10 定时器 2 中断 124
11.5.11 定时器 3 中断 125
11.5.12 定时器 4 中断 126
11.5.13 UART1 中断 127
11.5.14 UART2 中断 129
11.5.15 UART3 中断 131
11.5.16 UART4 中断 132
11.5.17 ADC中断 134
11.5.18 LVD中断 135
11.5.19 CMP中断 136
11.5.20 I2C中断 137
12 定时器/计数器 140
12.1 定时器的相关寄存器 140
12.2 定时器 0/1 142
12.3 定时器 2 145
12.4 定时器 3/4 146
12.5 掉电唤醒定时器 148
12.6 范例程序 149
12.6.1 定时器 0（模式 0－16 位自动重载） 149
12.6.2 定时器 0（模式 1－16 位不自动重载） 150
12.6.3 定时器 0（模式 2－8 位自动重载） 151
12.6.4 定时器 0（模式 3－16 位自动重载不可屏蔽中断） 152
12.6.5 定时器 0（外部计数－扩展T0 为外部下降沿中断） 153
12.6.6 定时器 0（测量脉宽－INT0 高电平宽度） 154
12.6.7 定时器 0（时钟分频输出） 155
12.6.8 定时器 1（模式 0－16 位自动重载） 155
12.6.9 定时器 1（模式 1－16 位不自动重载） 156

12.6.10 定时器 1（模式 2－8 位自动重载） 157
12.6.11 定时器 1（外部计数－扩展T1 为外部下降沿中断） 158
12.6.12 定时器 1（测量脉宽－INT1 高电平宽度） 159
12.6.13 定时器 1（时钟分频输出） 161
12.6.14 定时器 1（模式 0）做串口 1 波特率发生器 161
12.6.15 定时器 1（模式 2）做串口 1 波特率发生器 164
12.6.16 定时器 2（16 位自动重载） 167
12.6.17 定时器 2（外部计数－扩展T2 为外部下降沿中断） 168
12.6.18 定时器 2（时钟分频输出） 170
12.6.19 定时器 2 做串口 1 波特率发生器 170
12.6.20 定时器 2 做串口 2 波特率发生器 173
12.6.21 定时器 2 做串口 3 波特率发生器 176
12.6.22 定时器 2 做串口 4 波特率发生器 179
12.6.23 定时器 3（16 位自动重载） 182
12.6.24 定时器 3（外部计数－扩展T3 为外部下降沿中断） 184
12.6.25 定时器 3（时钟分频输出） 185
12.6.26 定时器 3 做串口 3 波特率发生器 186
12.6.27 定时器 4（16 位自动重载） 189
12.6.28 定时器 4（外部计数－扩展T4 为外部下降沿中断） 190
12.6.29 定时器 4（时钟分频输出） 191
12.6.30 定时器 4 做串口 4 波特率发生器 192
13 串口通信 196
13.1 串口相关寄存器 196
13.2 串口 1 197
13.2.1 串口 1 模式 0 198
13.2.2 串口 1 模式 1 199
13.2.3 串口 1 模式 2 202
13.2.4 串口 1 模式 3 202
13.2.5 自动地址识别 203
13.3 串口 2 205
13.3.1 串口 2 模式 0 205
13.3.2 串口 2 模式 1 206
13.4 串口 3 208
13.4.1 串口 3 模式 0 208
13.4.2 串口 3 模式 1 209
13.5 串口 4 211
13.5.1 串口 4 模式 0 211
13.5.2 串口 4 模式 1 212
13.6 串口注意事项 214
13.7 范例程序 215
13.7.1 串口 1 使用定时器 2 做波特率发生器 215
13.7.2 串口 1 使用定时器 1（模式 0）做波特率发生器 217
13.7.3 串口 1 使用定时器 1（模式 2）做波特率发生器 220
13.7.4 串口 2 使用定时器 2 做波特率发生器 223

13.7.5 串口 3 使用定时器 2 做波特率发生器 226
13.7.6 串口 3 使用定时器 3 做波特率发生器 229
13.7.7 串口 4 使用定时器 2 做波特率发生器 232
13.7.8 串口 4 使用定时器 4 做波特率发生器 235
14 比较器，掉电检测，内部固定比较电压 239
14.1 比较器内部结构图 239
14.2 比较器相关的寄存器 240
14.3 范例程序 242
14.3.1 比较器的使用（中断方式） 242
14.3.2 比较器的使用（查询方式） 243
14.3.3 比较器作外部掉电检测 245
14.3.4 比较器检测工作电压（电池电压） 247
15 IAP/EEPROM 251
15.1 EEPROM相关的寄存器 251
15.2 EEPROM大小及地址 253
15.3 范例程序 255
15.3.1 EEPROM基本操作 255
15.3.2 使用MOVC读取EEPROM 257
15.3.3 使用串口送出EEPROM数据 260
16 ADC模数转换 264
16.1 ADC相关的寄存器 264
16.2 范例程序 266
16.2.1 ADC基本操作（查询方式） 266
16.2.2 ADC基本操作（中断方式） 267
16.2.3 格式化ADC转换结果 268
16.2.4 利用ADC第 16 通道测量外部电压或电池电压 270
17 同步串行外设接口SPI 273
17.1 SPI相关的寄存器 273
17.2 SPI通信方式 275
17.2.1 单主单从 275
17.2.2 互为主从 275
17.2.3 单主多从 276
17.3 配置SPI 277
17.4 数据模式 279
17.5 范例程序 280
17.5.1 SPI单主单从系统主机程序（中断方式） 280
17.5.2 SPI单主单从系统从机程序（中断方式） 281
17.5.3 SPI单主单从系统主机程序（查询方式） 282
17.5.4 SPI单主单从系统从机程序（查询方式） 284
17.5.5 SPI互为主从系统程序（中断方式） 285
17.5.6 SPI互为主从系统程序（查询方式） 287
18 I^2C总线 289
18.1 I^2C相关的寄存器 289
18.2 I^2C主机模式 290

18.3 I²C从机模式 293
18.4 范例程序 296
18.4.1 I²C主机模式访问AT24C256（中断方式） 296
18.4.2 I²C主机模式访问AT24C256（查询方式） 301
18.4.3 I²C主机模式访问PCF8563 306
18.4.4 I²C从机模式（中断方式） 310
18.4.5 I²C从机模式（查询方式） 314
18.4.6 测试I²C从机模式代码的主机代码 317
19 高级PWM 322
19.1 简介 322
19.2 主要特性 322
19.3 时基单元 323
19.3.1 读写 16 位计数器 324
19.3.2 16 位PWM1_ARR寄存器的写操作 324
19.3.3 预分频器 324
19.3.4 向上计数模式 324
19.3.5 向下计数模式 326
19.3.6 中间对齐模式（向上/向下计数） 328
19.3.7 重复计数器 330
19.4 时钟/触发控制器 331
19.4.1 预分频时钟（CK_PSC） 331
19.4.2 内部时钟源（fMASTER） 332
19.4.3 外部时钟源模式 1 332
19.4.4 外部时钟源模式 2 333
19.4.5 触发同步 334
19.4.6 与PWM2 同步 337
19.5 捕获/比较通道 341
19.5.1 16 位PWM1_CCRi寄存器的写流程 343
19.5.2 输入模块 343
19.5.3 输入捕获模式 344
19.5.4 输出模块 346
19.5.5 强制输出模式 347
19.5.6 输出比较模式 347
19.5.7 PWM模式 348
19.5.8 使用刹车功能 354
19.5.9 在外部事件发生时清除OCiREF信号 356
19.5.10 编码器接口模式 357
19.6 中断 359
19.7 PWM1 寄存器描述 359
19.7.1 控制寄存器 1（PWM1_CR1） 359
19.7.2 控制寄存器 2（PWM1_CR2） 361
19.7.3 从模式控制寄存器(PWM1_SMCR) 362
19.7.4 外部触发寄存器(PWM1_ETR) 362
19.7.5 中断使能寄存器(PWM1_IER) 364

19.7.6 状态寄存器 1(PWM1_SR1) 364
19.7.7 状态寄存器 2(PWM1_SR2) 365
19.7.8 事件产生寄存器（PWM1_EGR） 366
19.7.9 捕获/比较模式寄存器 1（PWM1_CCMR1） 366
19.7.10 捕获/比较模式寄存器 2（PWM1_CCMR2） 369
19.7.11 捕获/比较模式寄存器 3（PWM1_CCMR3） 370
19.7.12 捕获/比较模式寄存器 4（PWM1_CCMR4） 371
19.7.13 捕获/比较使能寄存器 1（PWM1_CCER1） 372
19.7.14 捕获/比较使能寄存器 2（PWM1_CCER2） 373
19.7.15 计数器高 8 位（PWM1_CNTRH） 373
19.7.16 计数器低 8 位（PWM1_CNTRL） 374
19.7.17 预分频器高 8 位（PWM1_PSCRH） 374
19.7.18 预分频器低 8 位（PWM1_PSCRL） 374
19.7.19 自动重装载寄存器高 8 位（PWM1_ARRH） 374
19.7.20 自动重装载寄存器低 8 位（PWM1_ARRL） 374
19.7.21 重复计数器寄存器（PWM1_RCR） 375
19.7.22 捕获/比较寄存器 1 高 8 位（PWM1_CCR1H） 375
19.7.23 捕获/比较寄存器 1 低 8 位（PWM1_CCR1L） 375
19.7.24 捕获/比较寄存器 2 高 8 位（PWM1_CCR2H） 375
19.7.25 捕获/比较寄存器 2 低 8 位（PWM1_CCR2L） 376
19.7.26 捕获/比较寄存器 3 高 8 位（PWM1_CCR3H） 376
19.7.27 捕获/比较寄存器 3 低 8 位（PWM1_CCR3L） 376
19.7.28 捕获/比较寄存器 4 高 8 位（PWM1_CCR4H） 377
19.7.29 捕获/比较寄存器 4 低 8 位（PWM1_CCR4L） 377
19.7.30 刹车寄存器（PWM1_BKR） 377
19.7.31 死区寄存器（PWM1_DTR） 378
19.7.32 输出空闲状态寄存器（PWM1_OISR） 379
20 增强型双数据指针 380
20.1 范例程序 382
20.1.1 示例代码 1 382
20.1.2 示例代码 2 382
附录A 应用注意事项 384
附录B STC仿真器使用指南 385
附录C STC-USB驱动程序安装说明 390
附录D 电气特性 453
附录E 更新记录 455

1 概述

STC8H 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，STC8H 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，STC8H 系列单片机仅需 147 个时钟，而传统 8051 则需要 1944 个时钟。STC8H 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℃)，-1.8%～+0.8%温飘(-40℃~+85℃) ，-1.0%～+0.5%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，若温度范围为-45℃~+125℃时，最高频率须控制在 30MHz 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 24MHz 高精度 IRC 时钟（可适当调高或调低）、内部 32KHz 的低速 IRC、外部 4M~33M 晶振或外部时钟信号。用户代码中可自由选择时钟源，时钟源选定后可再经过 8-bit 的分频器分频后再将时钟信号提供给 CPU 和各个外设（如定时器、串口、SPI 等）。

MCU 提供两种低功耗模式：IDLE 模式和 STOP 模式。IDLE 模式下，MCU 停止给 CPU 提供时钟，CPU 无时钟，CPU 停止执行指令，但所有的外设仍处于工作状态，此时功耗约为 1.3mA（6MHz 工作频率）。STOP 模式即为主时钟停振模式，即传统的掉电模式/停电模式/停机模式，此时 CPU 和全部外设都停止工作，功耗可降低到 0.1uA 以下。

MCU 提供了丰富的数字外设（4 个串口、5 个定时器、2 组高级 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。

STC8H 系列单片机内部集成了增强型的双数据指针。通过程序控制, 可实现数据指针自动递增或递减功能以及两组数据指针的自动切换功能。

产品线	UART	定时器	ADC	增强型 PWM	高级 PWM	RTC	PCA	比较器	SPI	I2C	备注
STC8H1K16	●	●	●		●			●	●	●	

2 特性及价格

2.1 STC8H1K16 系列特性及价格

➢ 选型价格（不需要外部晶振、不需要外部复位，10 位 ADC，12 通道）

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	大容量扩展 SRAM 字节	强大的双 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	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	15 位增强型 PWM（带死区控制）	PCA/CCP/PWM（可当外部中断并可掉电唤醒）	掉电唤醒专用定时器	12 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（24MHz 可调）	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 USB 直接下载	本身就可在线仿真	封装		2018 年新品供货信息
																											LQFP32	QFN32	
STC8H1K16	1.7-5.5	16K	1K	2	12K	29	4	有	有	5	2	-	-	有	10 位	有	有	有	4 级	有	是	有	是	是	是	是	¥1.55	¥1.55	12 月送样
STC8H1K24	1.7-5.5	24K	1K	2	4K	29	4	有	有	5	2	-	-	有	10 位	有	有	有	4 级	有	是	有	是	是	是	是	¥1.7	¥1.7	
STC8H1K28	1.7-5.5	28K	1K	2	IAP	29	4	有	有	5	2	-	-	有	10 位	有	有	有	4 级	有	是	有	是	是	是	是	¥1.7	¥1.7	

➢ 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
- ✓ 21 个中断源，4 级中断优先级
- ✓ 支持在线仿真

➢ 工作电压

- ✓ 1.7V～5.5V
- ✓ 内建 LDO

➢ 工作温度

- ✓ -40℃～85℃

➢ Flash 存储器

- ✓ 最大 28K 字节 FLASH 空间，用于存储用户代码
- ✓ 支持用户配置 EEPROM 大小，512 字节单页擦除，擦写次数可达 10 万次以上
- ✓ 支持在系统编程方式（ISP）更新用户应用程序，无需专用编程器
- ✓ 支持单芯片仿真，无需专用仿真器，理论断点个数无限制

➢ SRAM

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

➢ 时钟控制

- ✓ 内部 24MHz 高精度 IRC（ISP 编程时可进行上下调整）
  - ⌖ 误差±0.3%（常温下 25℃）
  - ⌖ -1.8%～+0.8%温漂（全温度范围，-40℃～85℃）

- -1.0%～+0.5%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
  - 复位脚复位（高电平复位），出厂时 P5.4 默认为 IO 口，ISP 下载时可将 P5.4 管脚设置为复位脚
  - 看门狗溢出复位
  - 低压检测复位，提供 4 级低压检测电压：2.2V、2.4V、2.7V、3.0V
- ✓ 软件复位
  - 软件方式写复位触发寄存器

➢ **中断**

- ✓ 提供 21 个中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、串口 1、串口 2、串口 3、串口 4、ADC 模数转换、LVD 低压检测、SPI、I^2C、比较器、PWM1、PWM2
- ✓ 提供 4 级中断优先级

➢ **数字外设**

- ✓ 5 个 16 位定时器：定时器 0、定时器 1、定时器 2、定时器 3、定时器 4，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
- ✓ 4 个高速串口：串口 1、串口 2、串口 3、串口 4，波特率时钟源最快可为 FOSC/4
- ✓ 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
- ✓ 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

➢ **模拟外设**

- ✓ 超高速 ADC，支持 **10 位高精度** 12 通道（通道 0～通道 11）的模数转换
- ✓ **ADC 的通道 15 用于测试内部参考电压（芯片在出厂时，内部参考电压调整为 1.236V，误差±1%）**
- ✓ 比较器，一组比较器附近

➢ **GPIO**

- ✓ 最多可达 29 个 GPIO：P0.0~P0.3、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P5.4
- ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式
- ✓ **注意：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 口时不需先设置 IO 口模式**

➢ **封装**

- ✓ LQFP32

2.2 STC8H1K08 系列特性及价格

➢ 选型价格（不需要外部晶振、不需要外部复位，10 位 ADC，9 通道）

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	大容量扩展 SRAM 字节	强大的双 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	I²C	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	15 位增强型 PWM（带死区控制）	PCA/CCP/PWM（可当外部中断并可掉电唤醒）	掉电唤醒专用定时器	9 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（24MHz 可调）	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 USB 直接下载	本身就可在线仿真	封装			2018 年新品供货信息
																											TSSOP20	QFN20	SOP16	
STC8H1K08	1.7-5.5	8K	1K	2	4K	17	2	有	有	3	2	-	-	有	10 位	有	有	有	4 级	有	是	有	是	是	是	是	¥1.2	¥1.2	¥1.15	12 月送样
STC8H1K12	1.7-5.5	12K	1K	2	IAP	17	2	有	有	3	2	-	-	有	10 位	有	有	有	4 级	有	是	有	是	是	是	是	¥1.3	¥1.3	¥1.25	

➢ 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
- ✓ 17 个中断源，4 级中断优先级
- ✓ 支持在线仿真

➢ 工作电压

- ✓ 1.7V～5.5V
- ✓ 内建 LDO

➢ 工作温度

- ✓ -40℃～85℃

➢ Flash 存储器

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

➢ SRAM

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

➢ 时钟控制

- ✓ 内部 24MHz 高精度 IRC（ISP 编程时可进行上下调整）
  - ⌖ 误差±0.3%（常温下 25℃）
  - ⌖ -1.8%～+0.8%温漂（全温度范围，-40℃～85℃）
  - ⌖ -1.0%～+0.5%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
    - ⌖ 复位脚复位（高电平复位），出厂时 P5.4 默认为 IO 口，ISP 下载时可将 P5.4 管脚设置为复位脚
    - ⌖ 看门狗溢出复位
    - ⌖ 低压检测复位，提供 4 级低压检测电压：2.2V、2.4V、2.7V、3.0V
  - ✓ 软件复位
    - ⌖ 软件方式写复位触发寄存器

- **中断**
  - ✓ 提供 17 个中断源：INT0、INT1、INT2、INT3、INT4、定时器 0、定时器 1、定时器 2、串口 1、串口 2、ADC 模数转换、LVD 低压检测、SPI、I^2C、比较器、PWM1、PWM2
  - ✓ 提供 4 级中断优先级

- **数字外设**
  - ✓ 3 个 16 位定时器：定时器 0、定时器 1、定时器 2，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
  - ✓ 2 个高速串口：串口 1、串口 2，波特率时钟源最快可为 FOSC/4
  - ✓ 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
  - ✓ 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

- **模拟外设**
  - ✓ 超高速 ADC，支持 **10 位高精度** 9 通道（通道 0～通道 1，通道 8～通道 14）的模数转换
  - ✓ **ADC 的通道 15 用于测试内部参考电压（芯片在出厂时，内部参考电压调整为 1.236V，误差±1%）**
  - ✓ 比较器，一组比较器附近

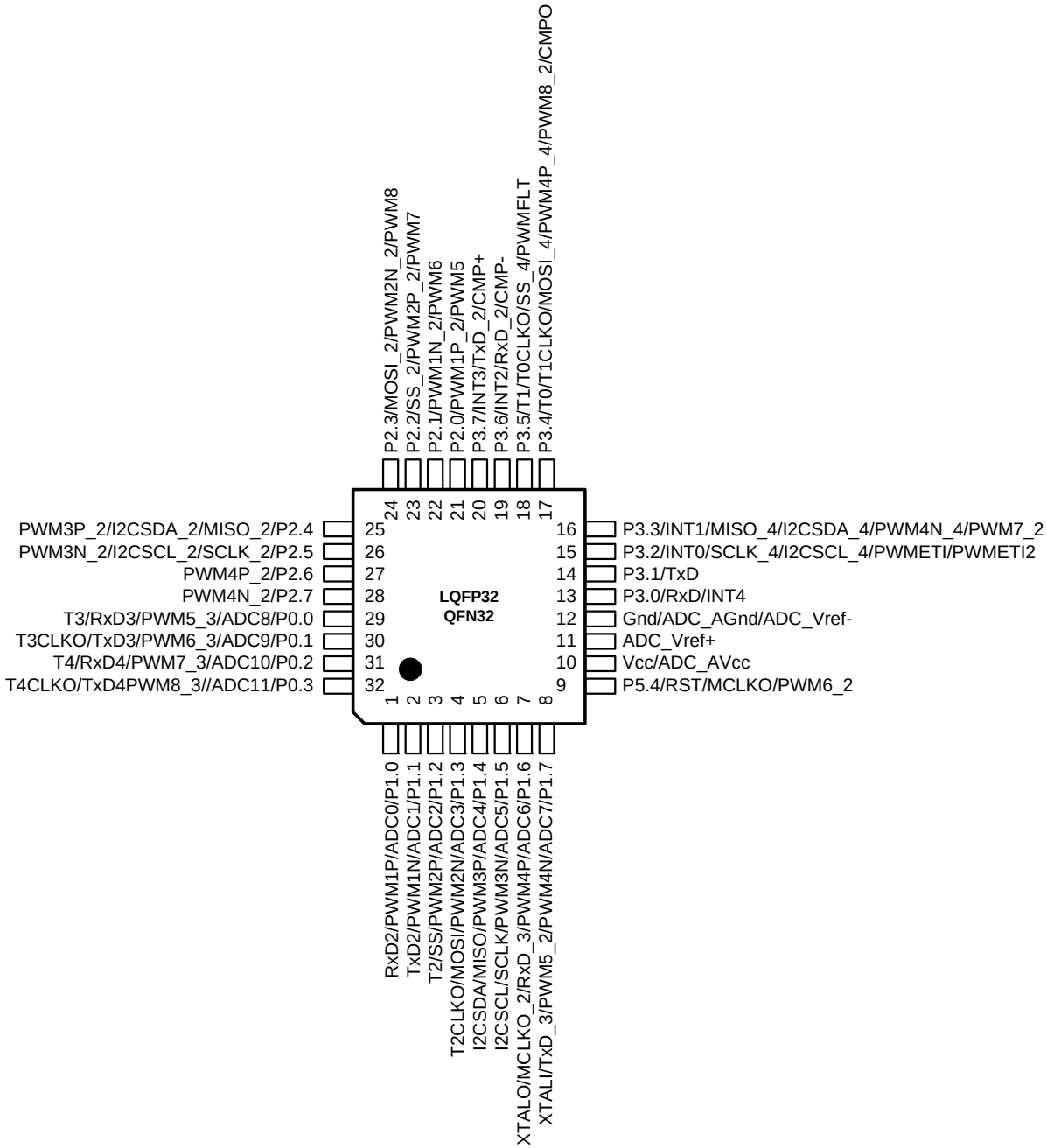
- **GPIO**
  - ✓ 最多可达 17 个 GPIO：P1.0~P1.7、P3.0~P3.7、P5.4
  - ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏输出模式、高阻输入模式
  - ✓ **注意：除 P3.0 和 P3.1 外，其余所有 IO 口上电后的状态均为高阻输入状态，用户在使用 IO 口时不需先设置 IO 口模式**

- **封装**
  - ✓ TSSOP20、QFN20、SOP16

3 管脚及说明

3.1 管脚图

3.1.1 STC8H1K16 系列管脚图



3.2 管脚说明

3.2.1 STC8H1K16 系列管脚说明

编号		名称	类型	说明
LQFP32				
1		P1.0	I/O	标准 IO 口
		RxD2	O	串口 2 的接收脚
		ADC0	I	ADC 模拟输入通道 0
		PWM1P	I/O	PWM1 的捕获输入和脉冲输出正极
2		P1.1	I/O	标准 IO 口
		TxD2	O	串口 2 的发送脚
		ADC1	I	ADC 模拟输入通道 1
		PWM1N	I/O	PWM1 的捕获输入和脉冲输出负极
3		P1.2	I/O	标准 IO 口
		ADC2	I	ADC 模拟输入通道 2
		SS	I/O	SPI 从机选择
		T2	I	定时器 2 外部时钟输入
		PWM2P	I/O	PWM2 捕获输入和脉冲输出正极
4		P1.3	I/O	标准 IO 口
		ADC3	I	ADC 模拟输入通道 3
		MOSI	I/O	SPI 主机输出从机输入
		T2CLKO	O	定时器 2 时钟分频输出
		PWM2N	I/O	PWM2 的捕获输入和脉冲输出负极
5		P1.4	I/O	标准 IO 口
		ADC4	I	ADC 模拟输入通道 4
		MISO	I/O	SPI 主机输入从机输出
		SDA	I/O	I2C 接口的数据线
		PWM3P	I/O	PWM3 捕获输入和脉冲输出正极
6		P1.5	I/O	标准 IO 口
		ADC5	I	ADC 模拟输入通道 5
		SCLK	I/O	SPI 的时钟脚
		SCL	I/O	I2C 的时钟线
		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				
7		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
		PWM4P	I/O	PWM4 捕获输入和脉冲输出正极
		MCLKO_2	O	主时钟分频输出
		XTALO	O	外部晶振的输出脚
8		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
		PWM4N	I/O	PWM4 的捕获输入和脉冲输出负极
		PWM5_2	I/O	PWM5 捕获输入和脉冲输出正极
		XTALI	I	外部晶振/外部时钟的输入脚
9		P5.4	I/O	标准 IO 口
		RST	I	复位引脚
		MCLKO	O	主时钟分频输出
		PWM6_2	I/O	PWM6 捕获输入和脉冲输出正极
10		VCC	VCC	电源脚
		AVCC	VCC	ADC 电源
11		VREF+	I	ADC 的参考电压脚
12		GND	GND	地线
		AGND	GND	ADC 地线
		VREF-	I	ADC 的参考电压地线
13		P3.0	I/O	标准 IO 口
		RxD	I	串口 1 的接收脚
		INT4	I	外部中断 4
14		P3.1	I/O	标准 IO 口
		TxD	O	串口 1 的发送脚
15		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
16		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
		PWM4N_4	I/O	PWM4 的捕获输入和脉冲输出负极
		PWM7_2	I/O	PWM7 捕获输入和脉冲输出正极

编号		名称	类型	说明
LQFP32				
17		P3.4	I/O	标准 IO 口
		T0	I	定时器 0 外部时钟输入
		T1CLKO	O	定时器 1 时钟分频输出
		MOSI_4	I/O	SPI 主机输出从机输入
		PWM4P_4	I/O	PWM4 捕获输入和脉冲输出正极
		PWM8_2	I/O	PWM8 捕获输入和脉冲输出正极
		CMPO	O	比较器输出
18		P3.5	I/O	标准 IO 口
		T1	I	定时器 1 外部时钟输入
		T0CLKO	O	定时器 0 时钟分频输出
		SS_4	I/O	SPI 从机选择
		PWMFLT	I	PWM 的外部异常检测脚
19		P3.6	I/O	标准 IO 口
		INT2	I	外部中断 2
		RxD_2	I	串口 1 的接收脚
		CMP-	I	比较器负极输入
20		P3.7	I/O	标准 IO 口
		INT3	I	外部中断 3
		TxD_2	O	串口 1 的发送脚
		CMP+	I	比较器正极输入
21		P2.0	I/O	标准 IO 口
		PWM1P_2	I/O	PWM1 的捕获输入和脉冲输出正极
		PWM5	I/O	PWM5 捕获输入和脉冲输出正极
22		P2.1	I/O	标准 IO 口
		PWM1N_2	I/O	PWM1 的捕获输入和脉冲输出负极
		PWM6	I/O	PWM6 捕获输入和脉冲输出正极
23		P2.2	I/O	标准 IO 口
		PWM2P_2	I/O	PWM2 的捕获输入和脉冲输出正极
		PWM7	I/O	PWM7 捕获输入和脉冲输出正极
		SS_2	I/O	SPI 从机选择
24		P2.3	I/O	标准 IO 口
		PWM2N_2	I/O	PWM2 的捕获输入和脉冲输出负极
		PWM8	I/O	PWM8 捕获输入和脉冲输出正极
		MOSI_2	I/O	SPI 主机输出从机输入

编号		名称	类型	说明
LQFP32				
25		P2.4	I/O	标准 IO 口
		PWM3P_2	I/O	PWM3 的捕获输入和脉冲输出正极
		MISO_2	I/O	SPI 主机输入从机输出
		SDA_2	I/O	I2C 的数据线
26		P2.5	I/O	标准 IO 口
		PWM3N_2	I/O	PWM3 的捕获输入和脉冲输出负极
		SCLK_2	I/O	SPI 的时钟脚
		SCL_2	I/O	I2C 的时钟线
27		P2.6	I/O	标准 IO 口
		PWM4P_2	I/O	PWM4 的捕获输入和脉冲输出正极
28		P2.7	I/O	标准 IO 口
		PWM4N_2	I/O	PWM4 的捕获输入和脉冲输出负极
29		P0.0	I/O	标准 IO 口
		ADC8	I	ADC 模拟输入通道 8
		RxD3	I	串口 3 的接收脚
		T3	I	定时器 3 外部时钟输入
		PWM5_3	I/O	PWM5 捕获输入和脉冲输出正极
30		P0.1	I/O	标准 IO 口
		ADC9	I	ADC 模拟输入通道 9
		TxD3	O	串口 3 的发送脚
		T3CLKO	O	定时器 3 时钟分频输出
		PWM6_3	I/O	PWM6 捕获输入和脉冲输出正极
31		P0.2	I/O	标准 IO 口
		ADC10	I	ADC 模拟输入通道 10
		RxD4	I	串口 4 的接收脚
		T4	I	定时器 4 外部时钟输入
		PWM7_3	I/O	PWM7 捕获输入和脉冲输出正极
32		P0.3	I/O	标准 IO 口
		ADC11	I	ADC 模拟输入通道 11
		TxD4	O	串口 4 的发送脚
		T4CLKO	O	定时器 4 时钟分频输出
		PWM8_3	I/O	PWM8 捕获输入和脉冲输出正极

3.3 功能脚切换

STC8H 系列单片机的特殊外设串口 1、串口 2、串口 3、串口 4、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。

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]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
MCLKOCR	主时钟输出控制寄存器	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
PWM1PS	PWM1 切换寄存器	FEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]		0000,0000
PWM2PS	PWM2 切换寄存器	FEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]		0000,0000

外设端口切换控制寄存器 1

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

SPI_S[1:0]：SPI 功能脚选择位

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

外设端口切换控制寄存器 2

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

I2C_S[1:0]：I²C 功能脚选择位

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

CMPO_S：比较器输出脚选择位

CMPO_S	CMPO
0	P3.4

1	P4.1

S4_S：串口 4 功能脚选择位

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

S3_S：串口 3 功能脚选择位

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

S2_S：串口 2 功能脚选择位

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

时钟选择寄存器

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

MCLKO_S：主时钟输出脚选择位

MCLKO_S	MCLKO
0	P5.4
1	P1.6

高级 PWM 选择寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM1PS	FEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]	
PWM2PS	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	保留	保留
11	保留	保留

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	保留	保留
11	保留	保留

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	保留	保留
11	保留	保留

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	保留	保留
11	P3.4	P3.3

C5PS[1:0]：高级 PWM 通道 5 输出脚选择位

C5PS[1:0]	PWM5
00	P2.0
01	P1.7
10	P0.0
11	保留

C6PS[1:0]：高级 PWM 通道 6 输出脚选择位

C6PS[1:0]	PWM6
00	P2.1
01	P5.4
10	P0.1
11	保留

C7PS[1:0]：高级 PWM 通道 7 输出脚选择位

C7PS[1:0]	PWM7
00	P2.2
01	P3.3
10	P0.2
11	保留

C8PS[1:0]：高级 PWM 通道 8 输出脚选择位

C8PS[1:0]	PWM8
00	P2.3
01	P3.4
10	P0.3
11	保留

3.4 范例程序

3.4.1 串口 1 切换

汇编代码

```
P_SW1   DATA    0A2H

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

        MOV     P_SW1,#00H              ;RXD/P3.0, TXD/P3.1
;       MOV     P_SW1,#40H              ;RXD_2/P3.6, TXD_2/P3.7
```

```
;       MOV     P_SW1,#80H              ;RXD_3/P1.6, TXD_3/P1.7
;       MOV     P_SW1,#0C0H             ;RXD_4/P4.3, TXD_4/P4.4

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW1 = 0xa2;

void main()
{
        P_SW1 = 0x00;           //RXD/P3.0, TXD/P3.1
//      P_SW1 = 0x40;           //RXD_2/P3.6, TXD_2/P3.7
//      P_SW1 = 0x80;           //RXD_3/P1.6, TXD_3/P1.7
//      P_SW1 = 0xc0;           //RXD_4/P4.3, TXD_4/P4.4

        while (1);
}
```

3.4.2 串口 2 切换

汇编代码

```
P_SW2   DATA    0BAH

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

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

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW2 = 0xba;

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

        while (1);
}
```

3.4.3 串口 3 切换

汇编代码

```
P_SW2   DATA    0BAH

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

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

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW2 = 0xba;

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

        while (1);
}
```

3.4.4 串口 4 切换

汇编代码

```
P_SW2   DATA    0BAH

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

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

        SJMP    $

        END
```

C 语言代码

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

```
sfr P_SW2 = 0xba;

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

        while (1);
}
```

3.4.5 SPI切换

汇编代码

```
P_SW1   DATA    0A2H

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

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

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW1 = 0xa2;

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

        while (1);
}
```

3.4.6 I2C切换

汇编代码

```
P_SW2   DATA    0BAH

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
```

```
MAIN:
        MOV     SP, #3FH

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

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW2 = 0xba;

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

        while (1);
}
```

3.4.7 比较器输出切换

汇编代码

```
P_SW2   DATA    0BAH

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

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

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

sfr P_SW2 = 0xba;

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

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

3.4.8 主时钟输出切换

汇编代码

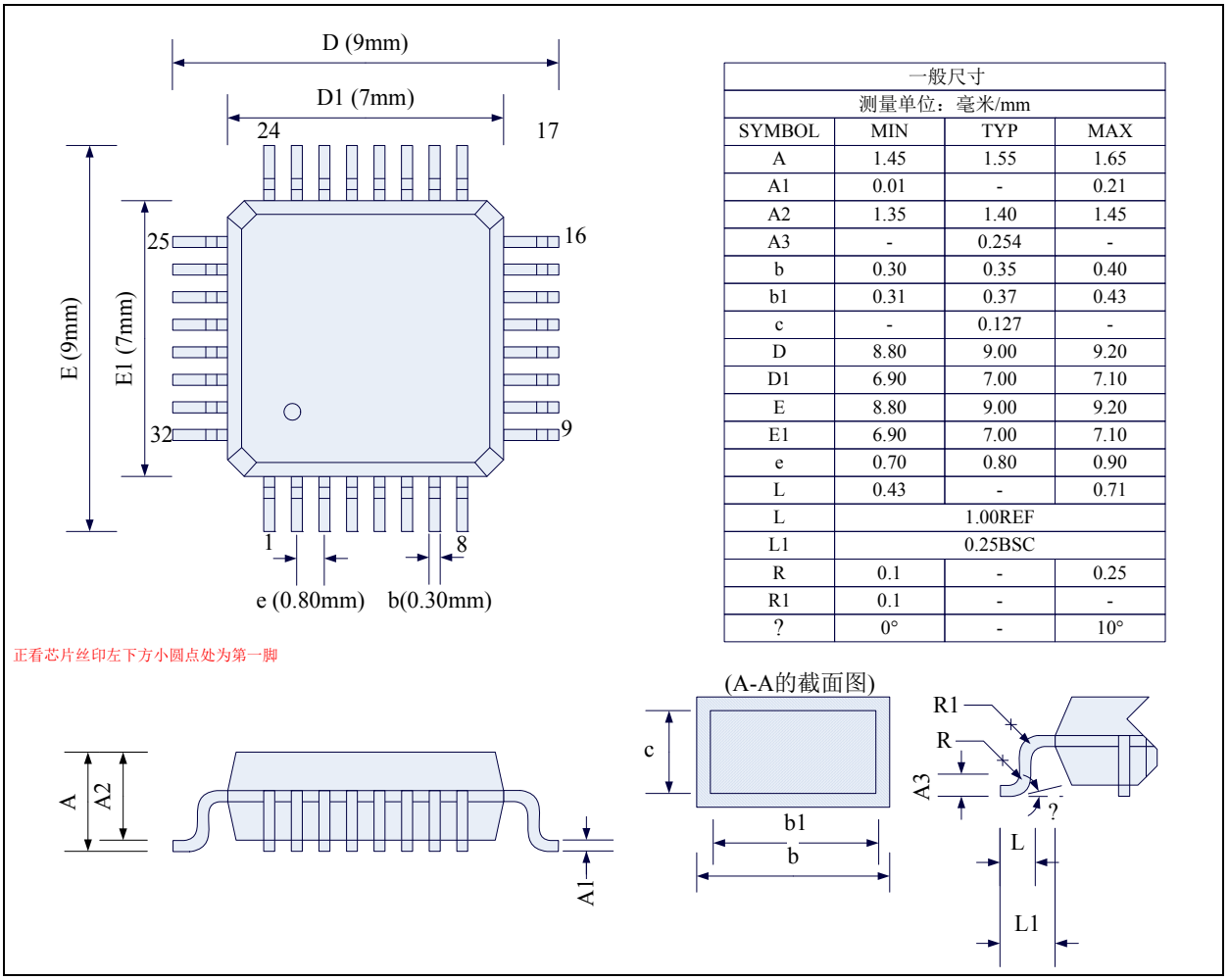
```
P_SW2   DATA    0BAH
CLKOCR  EQU     0FE05H

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

        MOV     P_SW2,#80H
        MOV     A,#04H                  ;IRC24M/4 output via MCLKO/P5.4
;       MOV     A,#84H                  ;IRC24M/4 output via MCLKO_2/P1.6
        MOV     DPTR,#CLKOCR
        MOVX    @DPTR,A
        MOV     P_SW2,#00H

        SJMP    $

        END
```

C 语言代码

```
#include "reg51.h"

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

sfr P_SW2 = 0xba;

void main()
{
        P_SW2 = 0x80;
        CLKOCR = 0x04;              //IRC24M/4 output via MCLKO/P5.4
//      CLKOCR = 0x84;              //IRC24M/4 output via MCLKO_2/P1.6
        P_SW2 = 0x00;

        while (1);
}
```

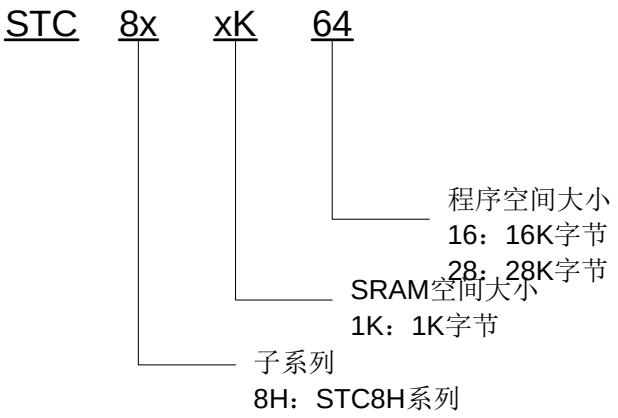
4 封装尺寸图

4.1 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	-
b	0.30	0.35	0.40
b1	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°

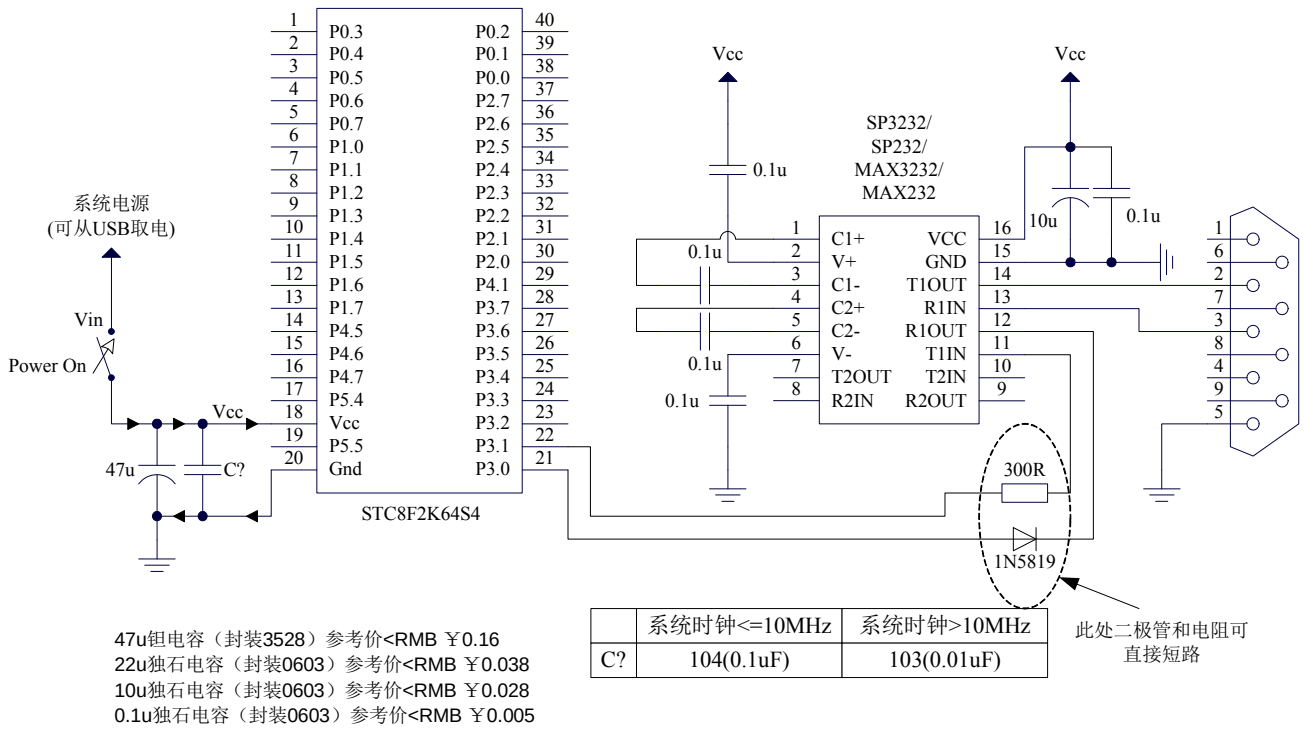
4.2 STC8H 系列单片机命名规则



5 ISP下载及典型应用线路图

5.1 STC8F系列ISP下载应用线路图

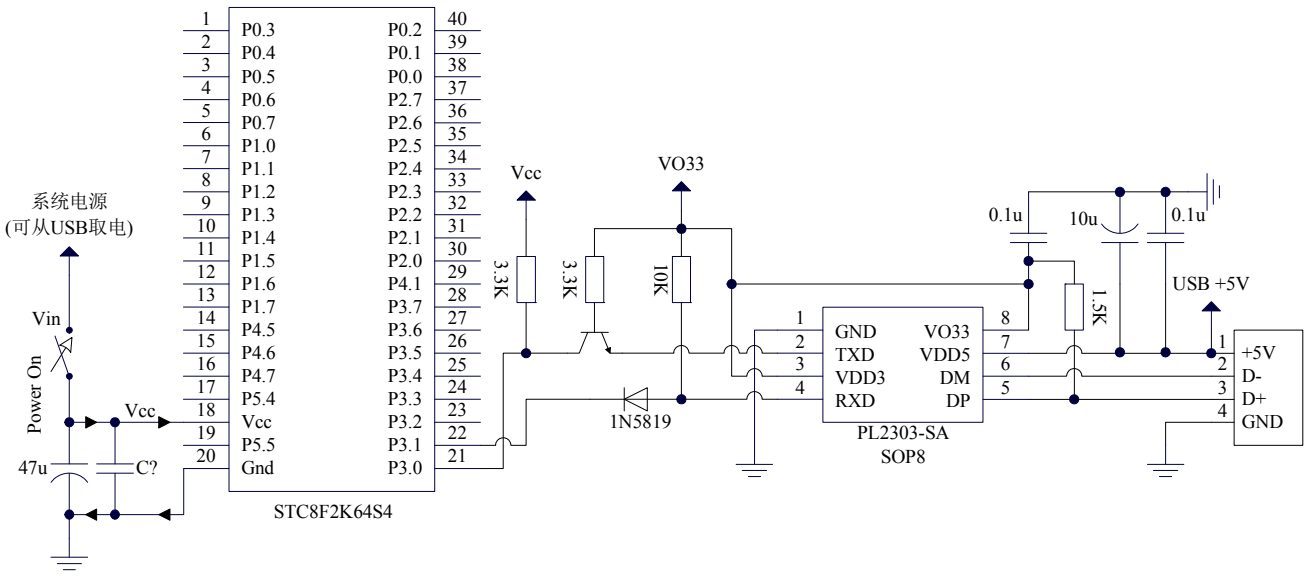
5.1.1 使用RS-232 转换器下载



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

47u钽电容（封装3528）参考价<RMB ￥0.16
22u独石电容（封装0603）参考价<RMB ￥0.038
10u独石电容（封装0603）参考价<RMB ￥0.028
0.1u独石电容（封装0603）参考价<RMB ￥0.005

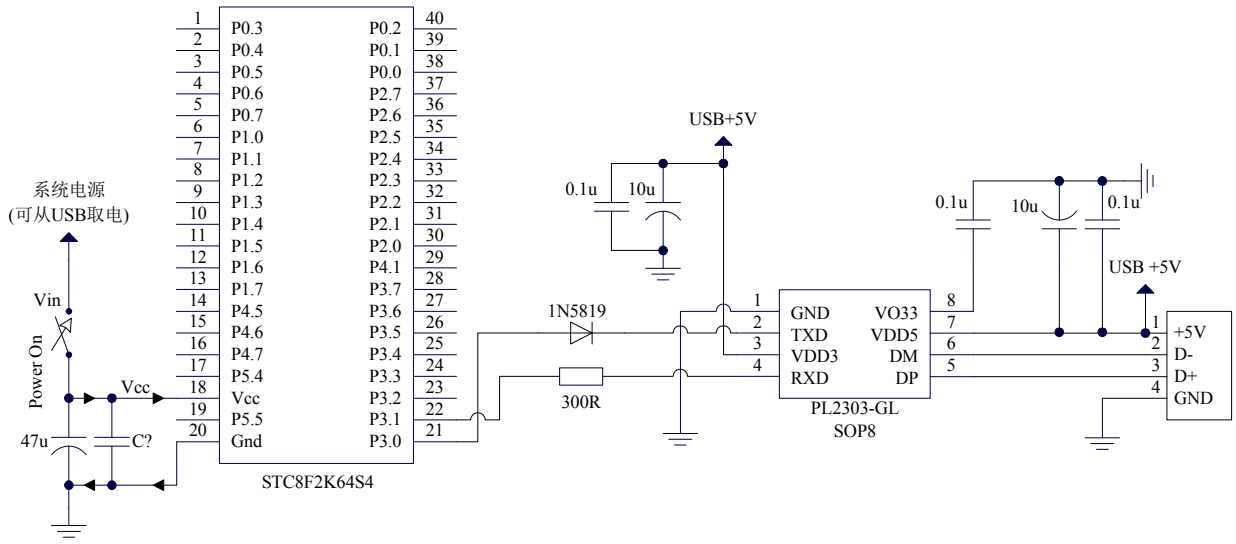
5.1.2 使用PL2303-SA下载



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

47u钽电容（封装3528）参考价<RMB ￥0.16
22u独石电容（封装0603）参考价<RMB ￥0.038
10u独石电容（封装0603）参考价<RMB ￥0.028
0.1u独石电容（封装0603）参考价<RMB ￥0.005

5.1.3 使用PL2303-GL下载

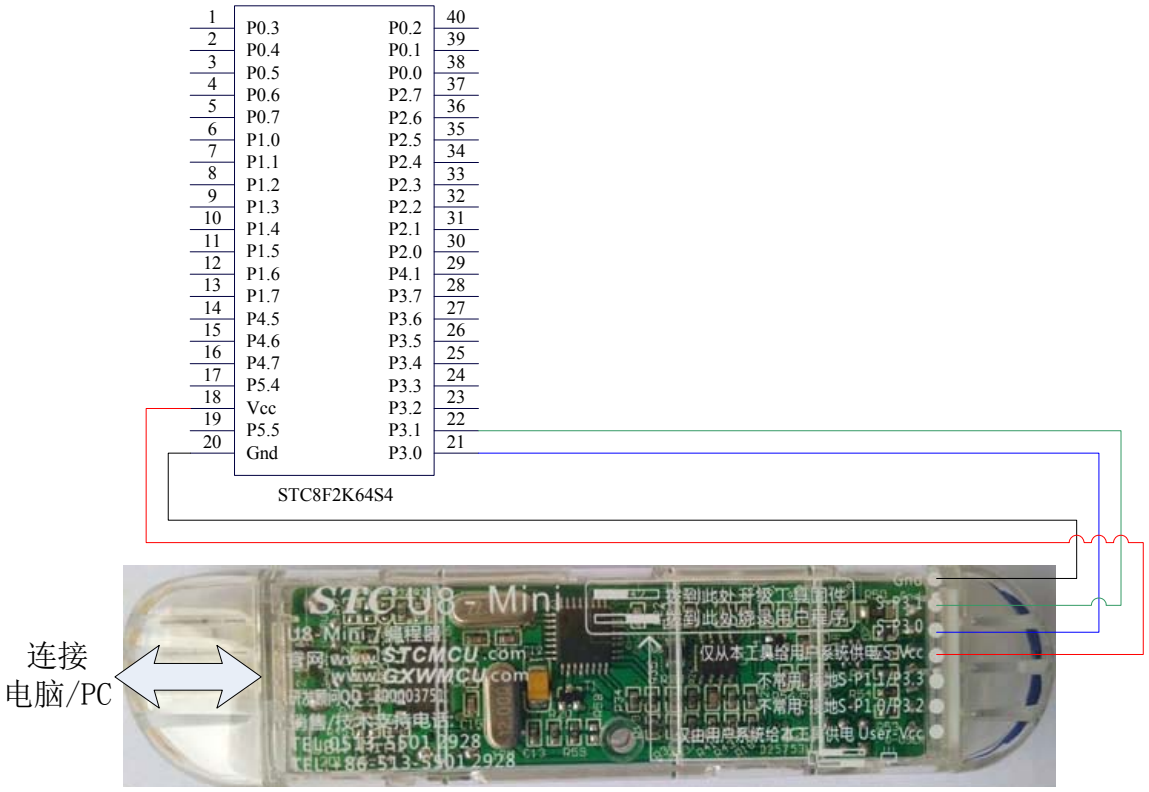


	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

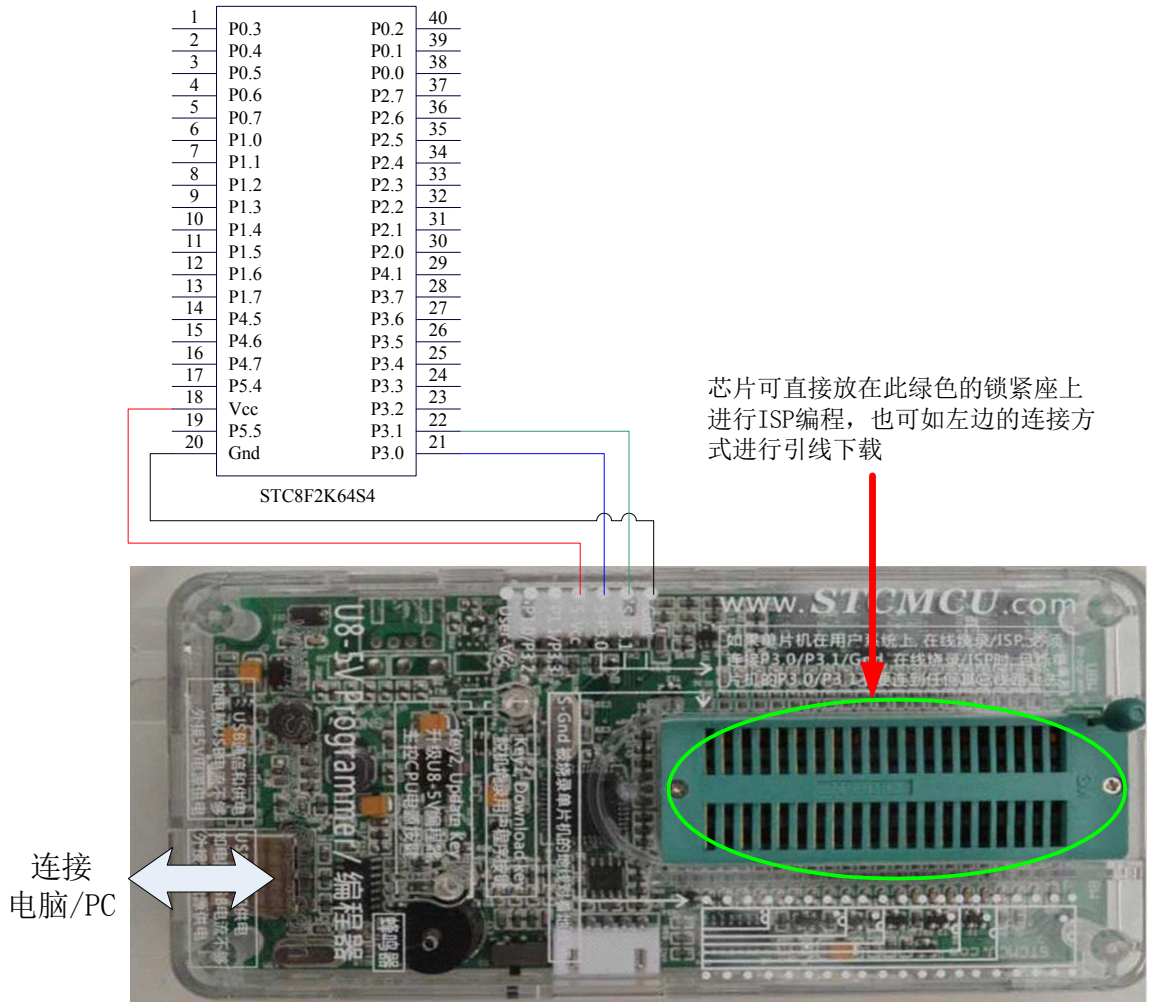
47u钽电容（封装3528）参考价<RMB ￥0.16
22u独石电容（封装0603）参考价<RMB ￥0.038
10u独石电容（封装0603）参考价<RMB ￥0.028
0.1u独石电容（封装0603）参考价<RMB ￥0.005

注：PL2303-GL 型号目前正处于试产阶段，预计到 2018 年 6 月中旬才会正式进入量产，届时使用 PL2303-GL 进行开发产品时，必须到 Prolific 官网下载最新驱动进行安装。

5.1.4 使用U8-Mini工具下载

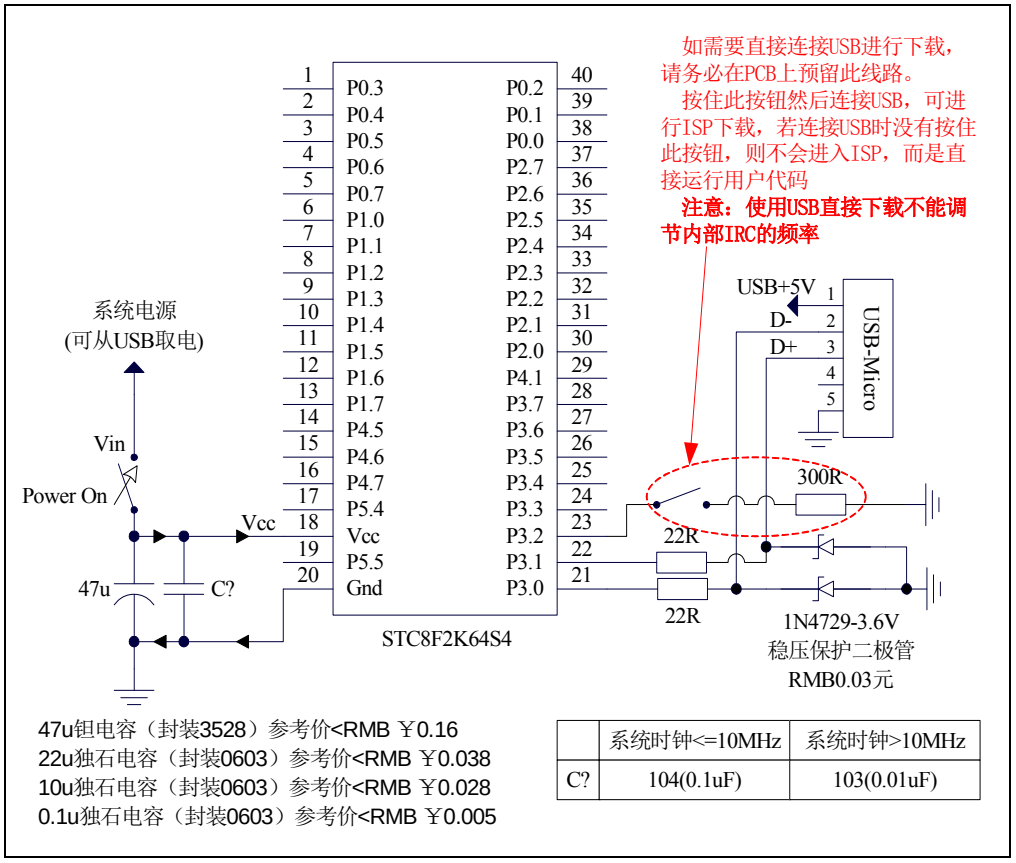


5.1.5 使用U8W工具下载



5.1.6 USB直接ISP下载

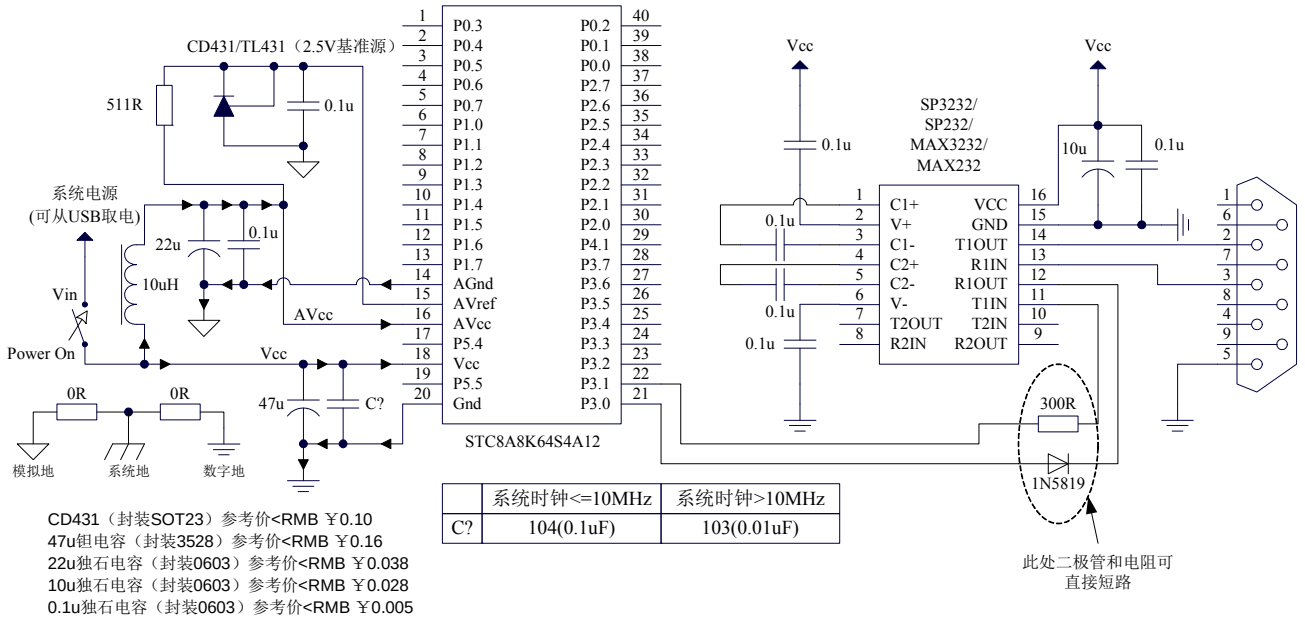
注：使用 USB 下载时需要将 P3.2 接 GND 才可进行正常下载



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

5.2 STC8A系列ISP下载应用线路图

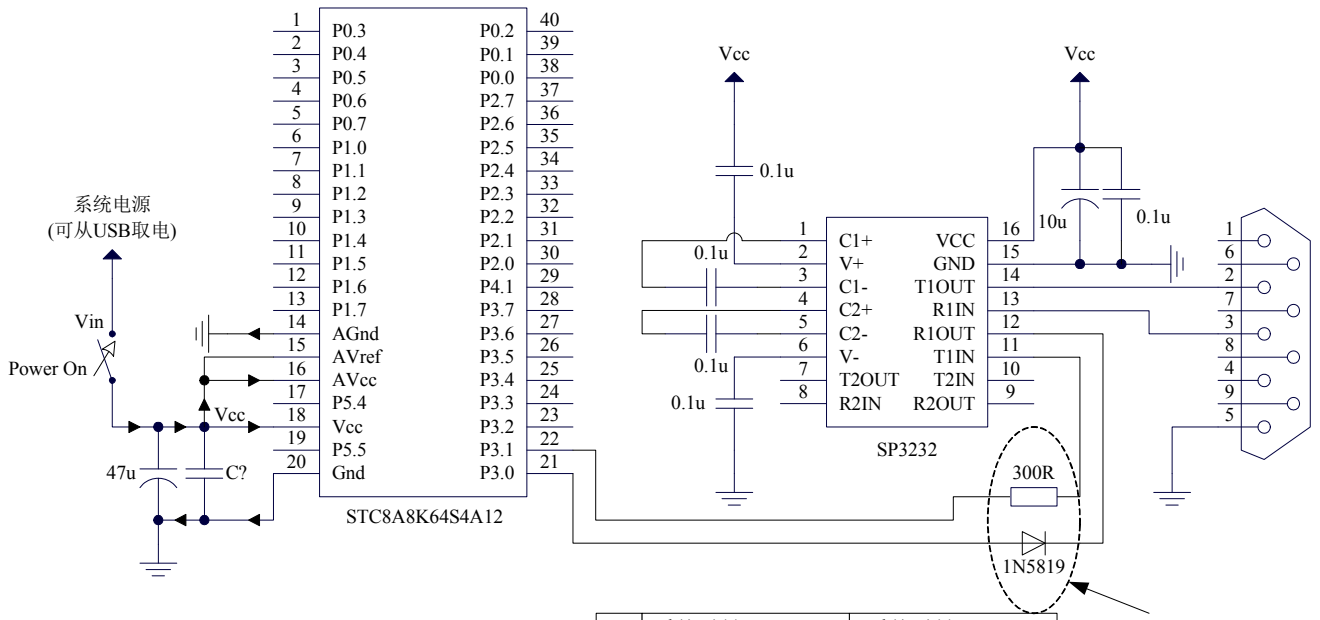
5.2.1 使用RS-232 转换下载（使用高精度ADC）



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

CD431（封装SOT23）参考价<RMB ￥0.10
47u钽电容（封装3528）参考价<RMB ￥0.16
22u独石电容（封装0603）参考价<RMB ￥0.038
10u独石电容（封装0603）参考价<RMB ￥0.028
0.1u独石电容（封装0603）参考价<RMB ￥0.005

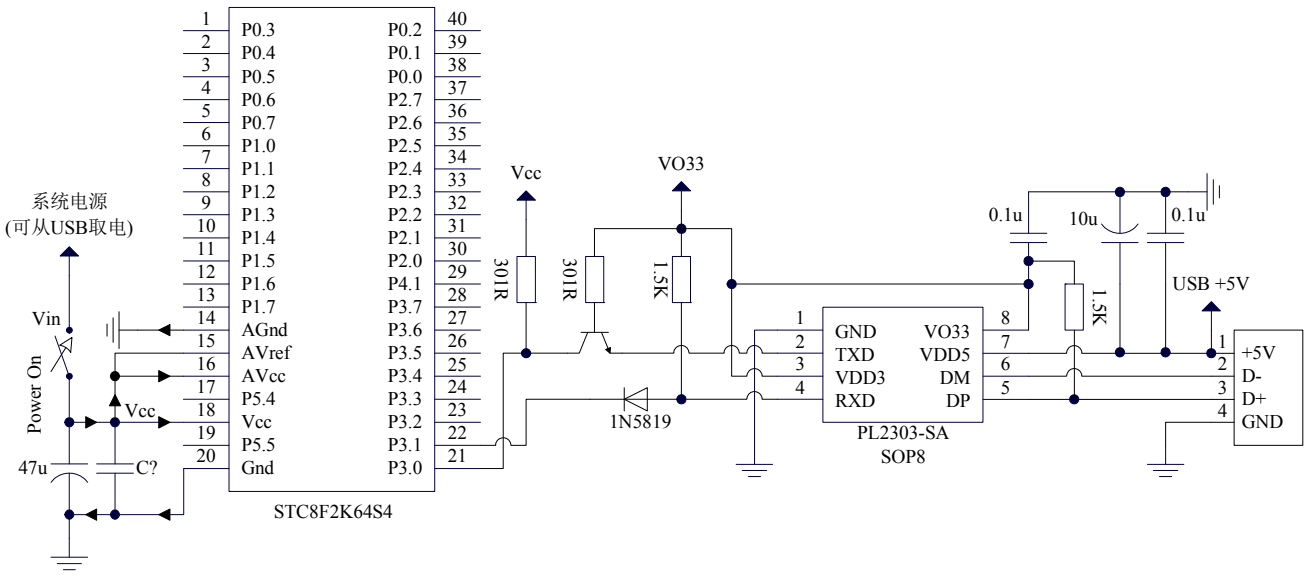
5.2.2 使用RS-232 转换下载（ADC一般应用）



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

47u钽电容（封装3528）参考价<RMB ￥0.16
22u独石电容（封装0603）参考价<RMB ￥0.038
10u独石电容（封装0603）参考价<RMB ￥0.028
0.1u独石电容（封装0603）参考价<RMB ￥0.005

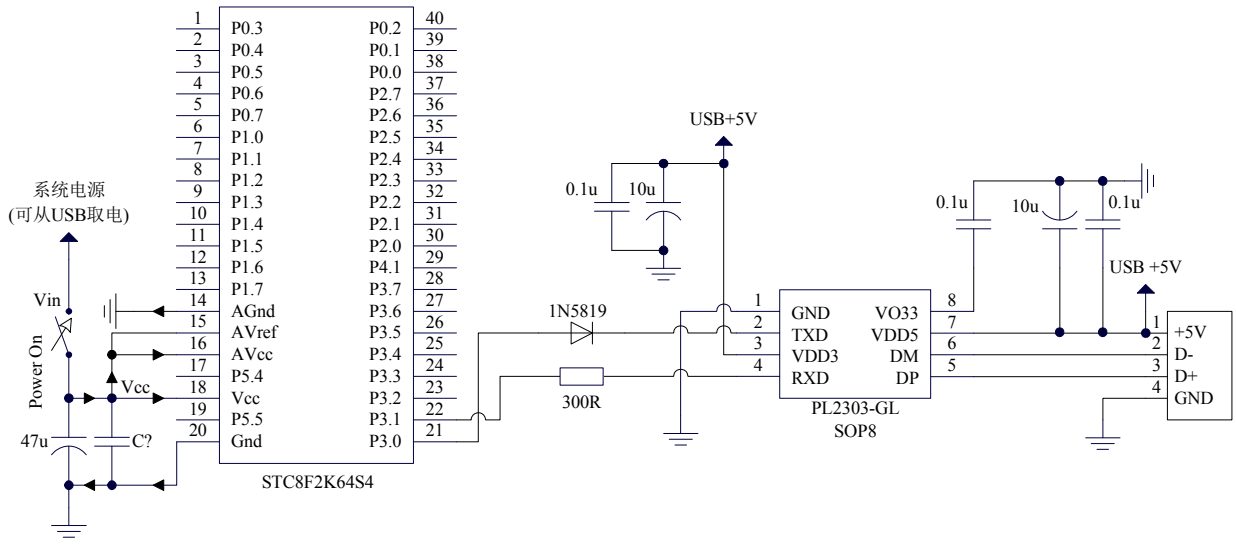
5.2.3 使用PL2303-SA下载



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

47u钽电容（封装3528）参考价<RMB ￥0.16
22u独石电容（封装0603）参考价<RMB ￥0.038
10u独石电容（封装0603）参考价<RMB ￥0.028
0.1u独石电容（封装0603）参考价<RMB ￥0.005

5.2.4 使用PL2303-GL下载

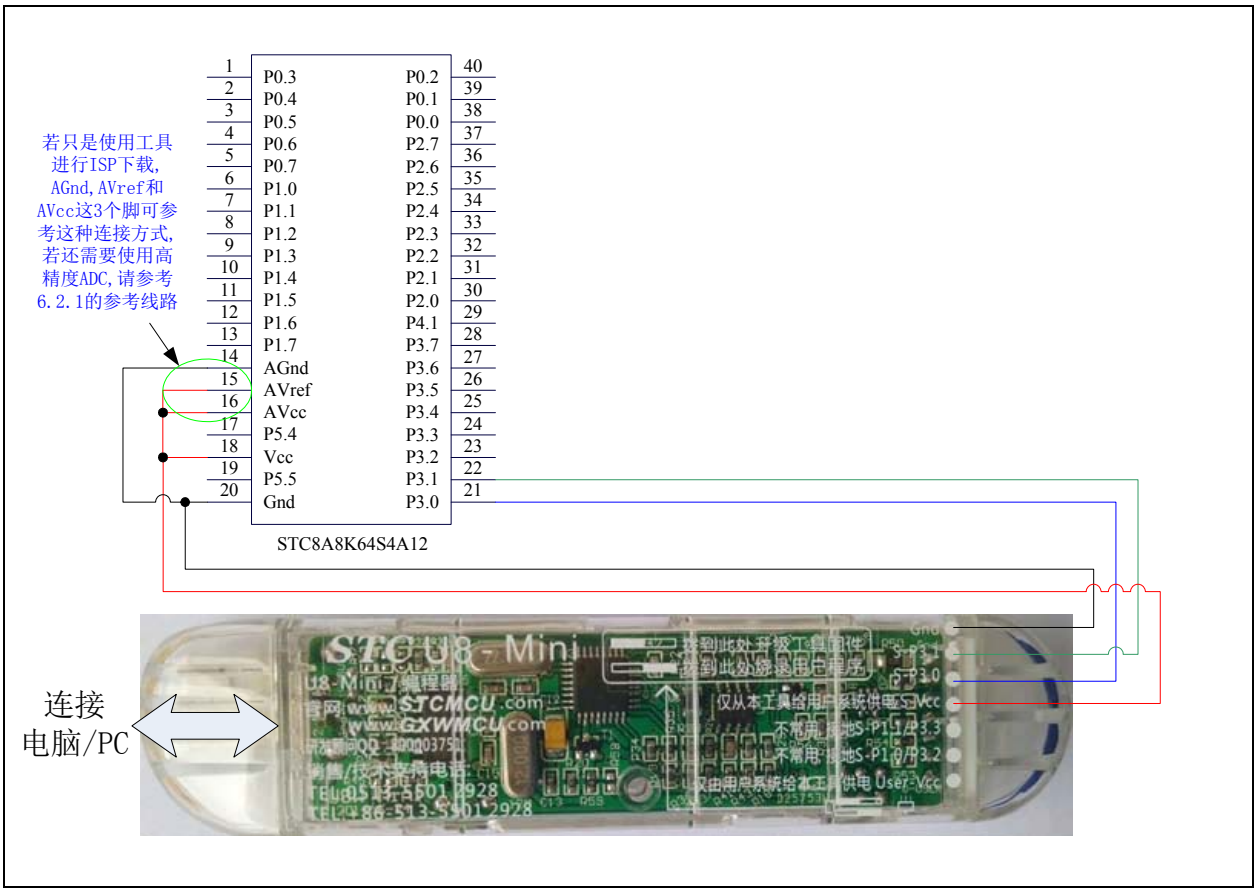


	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

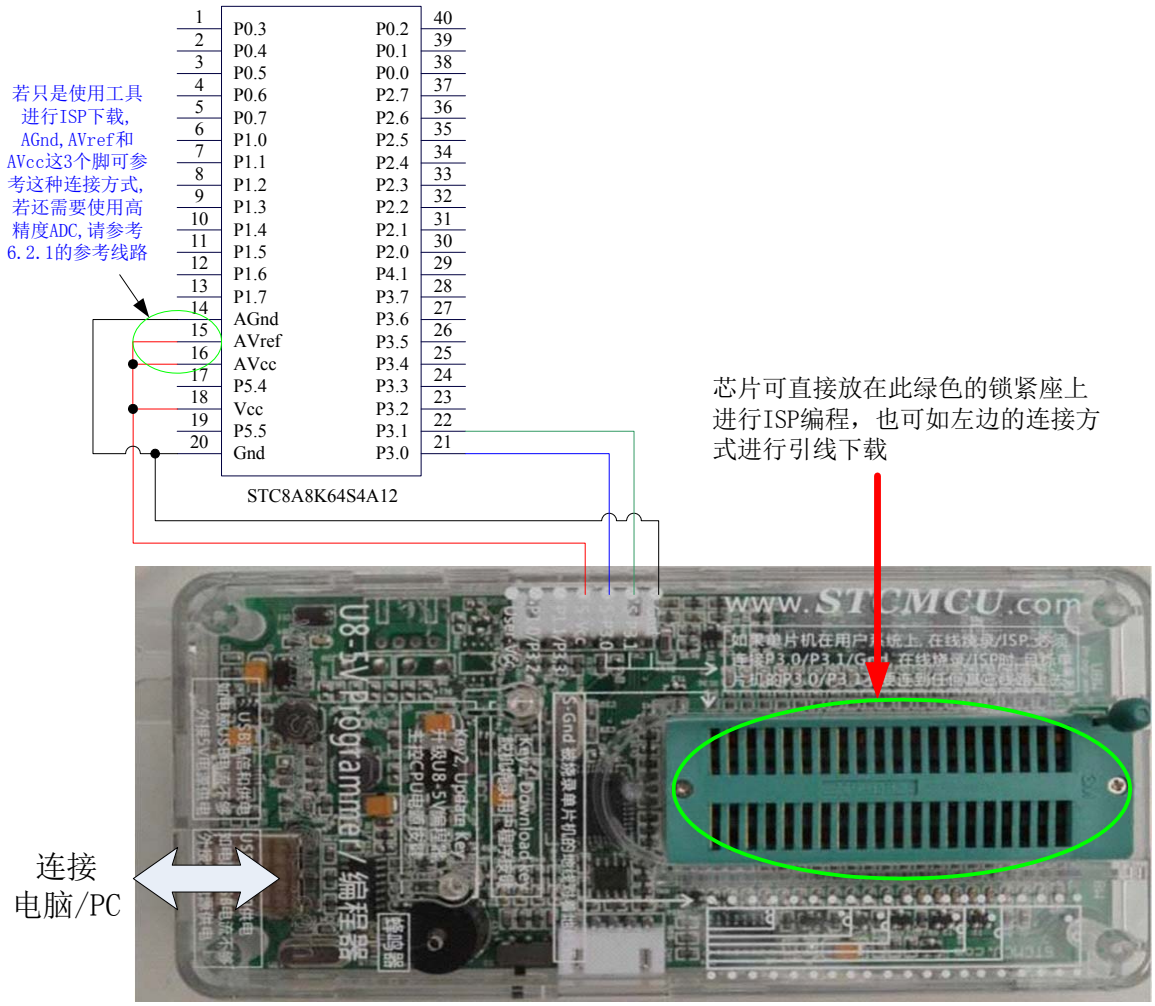
47u钽电容（封装3528）参考价<RMB ￥0.16
22u独石电容（封装0603）参考价<RMB ￥0.038
10u独石电容（封装0603）参考价<RMB ￥0.028
0.1u独石电容（封装0603）参考价<RMB ￥0.005

注：PL2303-GL 型号目前正处于试产阶段，预计到 2018 年 6 月中旬才会正式进入量产，届时使用 PL2303-GL 进行开发产品时，必须到 Prolific 官网下载最新驱动进行安装。

5.2.5 使用U8-Mini工具下载

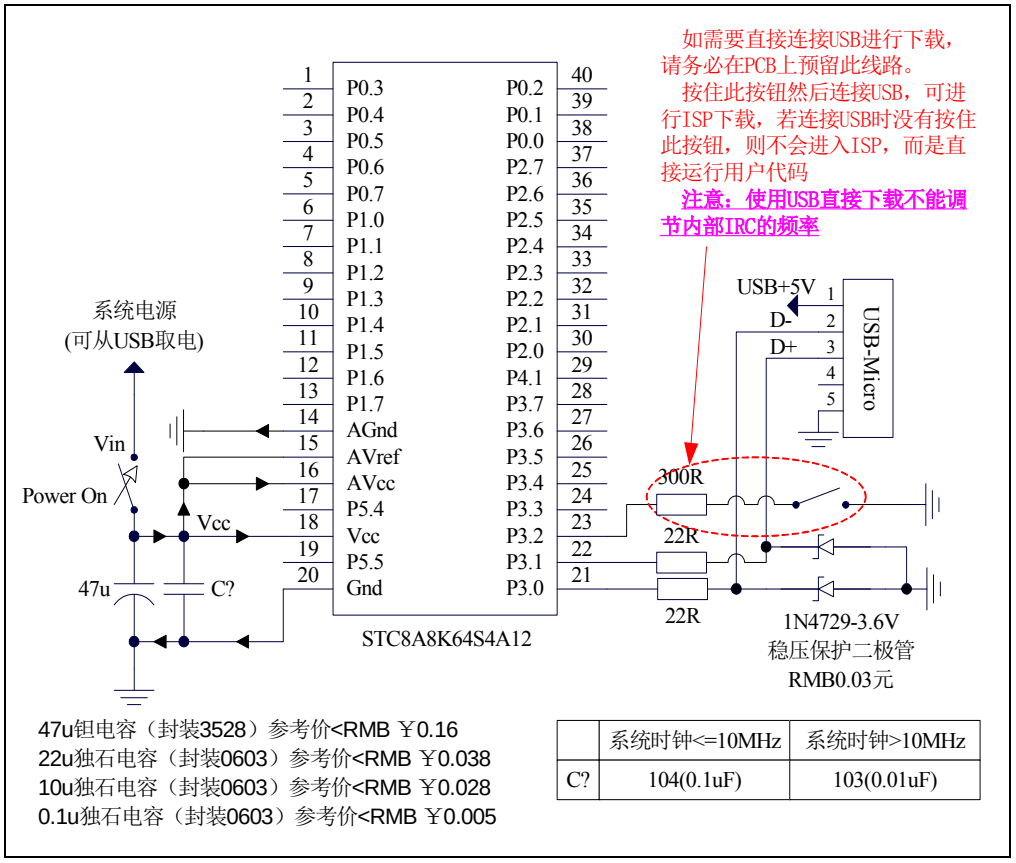


5.2.6 使用U8W工具下载



5.2.7 USB 直接 ISP 下载

注：使用 USB 下载时需要将 P3.2 接 GND 才可进行正常下载



47u钽电容（封装3528）参考价<RMB ￥0.16
22u独石电容（封装0603）参考价<RMB ￥0.038
10u独石电容（封装0603）参考价<RMB ￥0.028
0.1u独石电容（封装0603）参考价<RMB ￥0.005

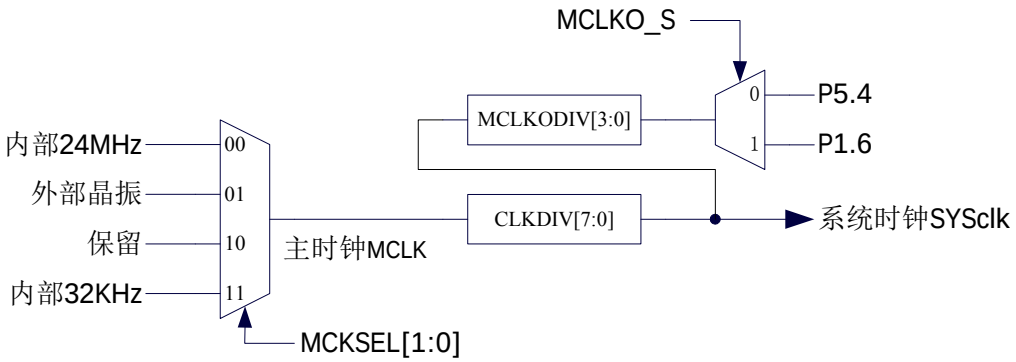
	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

6 时钟、复位与电源管理

6.1 系统时钟控制

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

单片机进入掉电模式后，时钟控制器将会关闭所有的时钟源



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	
CKSEL	时钟选择寄存器	FE00H	-	-	-	-	-	-	MCKSEL[1:0]		xxxx,xx00
CLKDIV	时钟分频寄存器	FE01H									0000,0100
IRC24MCR	内部 24M 振荡器控制寄存器	FE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	内部 32K 振荡器控制寄存器	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	主时钟输出控制寄存器	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000

CKSEL（系统时钟选择寄存器）

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

MCKSEL[1:0]：主时钟源选择

MCKSEL[1:0]	主时钟源
00	内部 24MHz 高精度 IRC
01	外部晶振
10	保留
11	内部 32KHz 低速 IRC

CLKDIV（时钟分频寄存器）

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

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

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

IRC24MCR（内部 24M 高精度 IRC 控制寄存器）

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

ENIRC24M：内部 24M 高精度 IRC 使能位

0：关闭内部 24M 高精度 IRC

1：使能内部 24M 高精度 IRC

IRC24MST：内部 24M 高精度 IRC 频率稳定标志位。（只读位）

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

XOSCCR（外部振荡器控制寄存器）

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

ENXOSC：外部晶体振荡器使能位

0：关闭外部晶体振荡器

1：使能外部晶体振荡器

XITYPE：外部时钟源类型

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

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

XOSCST：外部晶体振荡器频率稳定标志位。（只读位）

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

IRC32KCR（内部 32KHz 低速 IRC 控制寄存器）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0

IRC32KCR	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST

ENIRC32K：内部 32K 低速 IRC 使能位

0：关闭内部 32K 低速 IRC

1：使能内部 32K 低速 IRC

IRC32KST：内部 32K 低速 IRC 频率稳定标志位。（只读位）

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

MCLKOCR（主时钟输出控制寄存器）

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

MCLKODIV[6:0]：主时钟输出分频系数

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

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

MCLKO_S：系统时钟输出管脚选择

0：系统时钟分频输出到 P5.4 口

1：系统时钟分频输出到 P1.6 口

6.2 STC8H系列内部IRC频率调整

STC8H 系列单片机内部均集成有一颗高精度内部 IRC 振荡器。在用户使用 ISP 下载软件进行下载时，ISP 下载软件会根据用户所选择/设置的频率自动进行调整，一般频率值可调整到±0.3%以下，调整后的频率在全温度范围内（－40℃～85℃）的温漂可达-1.8%～0.8%。

STC8H 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。

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]								0000,0100

IRC 频段选择寄存器

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

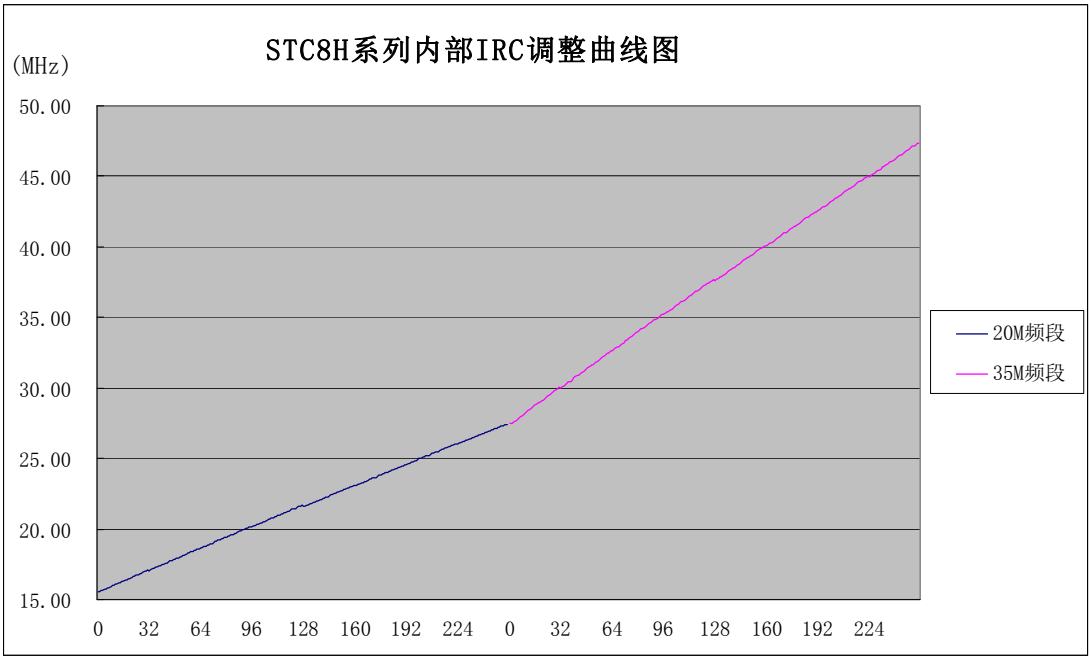
1：选择 35MHz 频段

内部 IRC 频率调整寄存器

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

IRTRIM[7:0]：内部高精度 IRC 频率调整寄存器

IRTRIM 可对 IRC 频率进行 256 个等级的调整，每个等级所调整的频率值在整体上呈线性分布，局部会有波动。宏观上，每一级所调整的频率约为 0.24%，即 IRTRIM 为（n+1）时的频率比 IRTRIM 为（n）时的频率约快 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。



内部 IRC 频率微调寄存器

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%

CLKDIV（时钟分频寄存器）

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

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

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

STC8H 系列内部的两个频段的可调范围分别为 15.5MHz～27MHz 和 27.5MHz～47MHz，所以两个频段直接存在调节盲区（如 27MHz~27.5MHz 之间的频率）。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。

6.3 系统复位

STC8 系列单片机的复位分为硬件复位和软件复位两种。

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

- 上电复位
- 低压复位
- 复位脚复位
- 看门狗复位

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

- 写 IAP_CONTR 的 SWRST 所触发的复位

相关寄存器

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

WDT_CONTR（看门狗控制寄存器）

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

WDT_FLAG：看门狗溢出标志

看门狗发生溢出时，硬件自动将此位置 1，需要软件清零。

EN_WDT：看门狗使能位

0：对单片机无影响

1：启动看门狗定时器

CLR_WDT：看门狗定时器清零

0：对单片机无影响

1：清零看门狗定时器，硬件自动将此位复位

IDL_WDT：IDLE 模式时的看门狗控制位

0：IDLE 模式时看门狗停止计数

1：IDLE 模式时看门狗继续计数

WDT_PS[2:0]：看门狗定时器时钟分频系数

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

看门狗溢出时间计算公式如下：

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

IAP_CONTR（IAP 控制寄存器）

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

SWBS：软件复位启动选择

0：软件复位后从用户程序区开始执行代码。用户数据区的数据保持不变。

1：软件复位后从系统 ISP 区开始执行代码。用户数据区的数据会被初始化。

SWRST：软件复位触发位

0：对单片机无影响

1：触发软件复位

RSTCFG（复位配置寄存器）

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

ENLVR：低压复位控制位

0：禁止低压复位。当系统检测到低压事件时，会产生低压中断

1：使能低压复位。当系统检测到低压事件时，自动复位

P54RST：RST 管脚功能选择

0：RST 管脚用作普通 I/O 口（P54）

1：RST 管脚用作复位脚

LVDS[1:0]：低压检测门槛电压设置

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

6.4 系统电源管理

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

PCON（电源控制寄存器）

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

LVDF：低压检测标志位。当系统检测到低压事件时，硬件自动将此位置 1，并向 CPU 提出中断请求。此位需要用户软件清零。

POF：上电标志位。当硬件自动将此位置 1。

PD：掉电模式控制位

0：无影响

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

IDL：IDLE（空闲）模式控制位

0：无影响

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

VOCTRL（电压控制寄存器）

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

SCC：静态电流控制位

0：选择内部静态保持电流控制线路，静态电流一般为 1.5uA 左右。

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

[B1:B0]：内部测试位，必须写入 0

6.5 范例程序

6.5.1 选择系统时钟源

汇编代码

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

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

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

;           MOV     P_SW2,#80H
;           MOV     A,#80H              ;启动内部 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,#CKSEL
;           MOVX    @DPTR,A
;           MOV     P_SW2,#00H

            JMP     $

            END
```

C 语言代码

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

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

sfr        P_SW2        =    0xba;

void main()
{
    P_SW2 = 0x80;
    CKSEL = 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
    CKSEL = 0x01;                   //选择外部晶振
    P_SW2 = 0x00;
*/

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

6.5.2 主时钟分频输出

汇编代码

```
P_SW2       DATA    0BAH
MCLKOCR     EQU     0FE05H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

            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 P1.6 口
            MOV     DPTR,#MCLKOCR
```

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

                JMP     $

                END
```

C 语言代码

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

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

sfr         P_SW2       =   0xba;

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

    while (1);
}
```

6.5.3 看门狗定时器应用

汇编代码

```
;测试工作频率为 11.0592MHz

WDT_CONTR   DATA    0C1H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

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

            END
```

C 语言代码

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

```
//测试工作频率为11.0592MHz

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

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

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

6.5.4 软复位实现自定义下载

汇编代码

```
;测试工作频率为11.0592MHz

IAP_CONTR       DATA        0C7H

                ORG         0000H
                LJMP        MAIN

                ORG         0100H
MAIN:
                MOV         SP,#3FH

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

                END
```

C 语言代码

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

//测试工作频率为11.0592MHz

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

void main()
{
        P32 = 1;                        //测试端口
        P33 = 1;                        //测试端口
```

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

6.5.5 低压检测

汇编代码

```
RSTCFG      DATA    0FFH
ENLVR       EQU     40H             ;RSTCFG.6
LVD2V2      EQU     00H             ;LVD@2.2V
LVD2V4      EQU     01H             ;LVD@2.4V
LVD2V7      EQU     02H             ;LVD@2.7V
LVD3V0      EQU     03H             ;LVD@3.0V
ELVD        BIT     IE.6
LVDF        EQU     20H             ;PCON.5

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF  ;清中断标志
            CPL     P3.2            ;测试端口
            RETI
MAIN:
            MOV     SP,#3FH

            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
```

C 语言代码

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

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

```
sbit        P32             =   P3^2;

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

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

    while (1);
}
```

6.5.6 省电模式

汇编代码

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

            ORG     0000H
            LJMP    MAIN
            ORG     0003H
            LJMP    INT0ISR

            ORG     0100H
INT0ISR:
            CPL     P3.4            ;测试端口
            RETI
MAIN:
            MOV     SP,#3FH

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

            END
```

C 语言代码

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

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

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

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

    while (1);
}
```

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

汇编代码

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

            ORG     0000H
            LJMP    MAIN

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

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

```
INT2ISR:
                CPL     P1.0                    ;测试端口
                RETI
INT3ISR:
                CPL     P1.0                    ;测试端口
                RETI
INT4ISR:
                CPL     P1.0                    ;测试端口
                RETI

MAIN:
                MOV     SP,#3FH

                CLR     IT0                     ;使能 INT0 上升沿和下降沿中断
;               SETB    IT0                     ;使能 INT0 下降沿中断
                SETB    EX0                     ;使能 INT0 中断

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

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

                SETB    EA

                MOV     PCON,#02H               ;MCU 进入掉电模式
                NOP                             ;掉电唤醒后立即进入中断服务程序
                NOP
LOOP:
                CPL     P1.1
                JMP     LOOP

                END
```

C 语言代码

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

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

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

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

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

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

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

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

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

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

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

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

6.5.8 使用T0/T1/T2/T3/T4 中断唤醒MCU

汇编代码

```
;测试工作频率为 11.0592MHz

T2L     DATA    0D7H
T2H     DATA    0D6H
T3L     DATA    0D5H
T3H     DATA    0D4H
T4L     DATA    0D3H
T4H     DATA    0D2H
T4T3M   DATA    0D1H
AUXR    DATA    8EH
IE2     DATA    0AFH
ET2     EQU     04H
ET3     EQU     20H
ET4     EQU     40H
```

```
AUXINTIF    DATA    0EFH
T2IF        EQU     01H
T3IF        EQU     02H
T4IF        EQU     04H

            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
            ANL     AUXINTIF,#NOT T2IF  ;清中断标志
            RETI
TM3ISR:
            CPL     P1.0                ;测试端口
            ANL     AUXINTIF,#NOT T3IF  ;清中断标志
            RETI
TM4ISR:
            CPL     P1.0                ;测试端口
            ANL     AUXINTIF,#NOT T4IF  ;清中断标志
            RETI

MAIN:
            MOV     SP,#3FH

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

            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
LOOP:
                CPL     P1.1
                JMP     LOOP

                END
```

C 语言代码

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

//测试工作频率为11.0592MHz

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

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

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

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

void TM2_Isr() interrupt 12
{
```

```
	P10 = !P10;				//测试端口
	AUXINTIF &= ~T2IF;		//清中断标志
}

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

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

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

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

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

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

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

	EA = 1;

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

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

6.5.9 使用RxD/RxD2/RxD3/RxD4 中断唤醒MCU

汇编代码

```
;测试工作频率为11.0592MHz

IE2         DATA    0AFH
ES2         EQU     01H
ES3         EQU     08H
ES4         EQU     10H

P_SW1       DATA    0A2H
P_SW2       DATA    0BAH


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

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

MAIN:
            MOV     SP,#3FH

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

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

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

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

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

```
                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
LOOP:
                CPL     P1.1
                JMP     LOOP

                END
```

C 语言代码

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

//测试工作频率为11.0592MHz

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

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

sbit    P11     =   P1^1;

void UART1_Isr() interrupt 4
{
}

void UART2_Isr() interrupt 8
{
}

void UART3_Isr() interrupt 17
{
}

void UART4_Isr() interrupt 18
{
}

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

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

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

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

    ES = 1;                         //使能串口中断
```

```
    IE2 = ES2;                      //使能串口中断
    IE2 |= ES3;                     //使能串口中断
    IE2 |= ES4;                     //使能串口中断
    EA = 1;

    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_();

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

6.5.10 使用LVD中断唤醒MCU

汇编代码

```
RSTCFG      DATA    0FFH
ENLVR       EQU     40H             ;RSTCFG.6
LVD2V2      EQU     00H             ;LVD@2.2V
LVD2V4      EQU     01H             ;LVD@2.4V
LVD2V7      EQU     02H             ;LVD@2.7V
LVD3V0      EQU     03H             ;LVD@3.0V
ELVD        BIT     IE.6
LVDF        EQU     20H             ;PCON.5

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF  ;清中断标志
            CPL     P1.0            ;测试端口
            RETI

MAIN:
            MOV     SP,#3FH

            ANL     PCON,#NOT LVDF  ;上电需要清中断标志
            MOV     RSTCFG,# LVD3V0 ;设置 LVD 电压为 3.0V
            SETB    ELVD            ;使能 LVD 中断
            SETB    EA

            MOV     PCON,#02H       ;MCU 进入掉电模式
            NOP                     ;掉电唤醒后立即进入中断服务程序
            NOP
LOOP:
            CPL     P1.1
            JMP     LOOP

            END
```

C 语言代码

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

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

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

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

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

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

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

6.5.11 CMP中断唤醒MCU

汇编代码

```
CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR

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

MAIN:
            MOV     SP,#3FH
```

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

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

LOOP:
                CPL     P1.1
                JMP     LOOP

                END
```

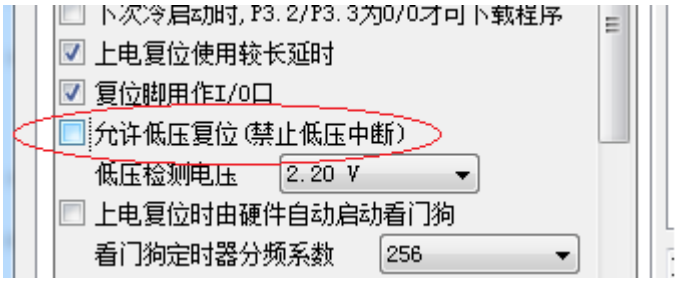
C 语言代码

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

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

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

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

void main()
{
    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_();

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

6.5.12 使用LVD功能检测工作电压（电池电压）

若需要使用 LVD 功能检测电池电压，则在 ISP 下载时需要将低压复位功能去掉，如下图 “允许低压复

位（禁止低压中断）”的硬件选项的勾选项需要去掉



汇编代码

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

            ORG     0000H
            JMP     MAIN

            ORG     0100H
MAIN:
            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
                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
```

C 语言代码

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

#define FOSC            24000000UL
#define T1MS            (65536 - FOSC/4/100)

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

#define LVDF            0x20        //PCON.5

void delay()
{
    int i;

    for (i=0; i<100; i++)
    {
```

```
            _nop_();
            _nop_();
            _nop_();
            _nop_();
        }
}

void main()
{
        unsigned char power;

        PCON &= ~LVDF;
        RSTCFG = LVD3V0;

        while (1)
        {
            power = 0x0f;

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

                RSTCFG = LVD2V7;
                delay();
                PCON &= ~LVDF;
                delay();
                if (PCON & LVDF)
                {
                    power >>= 1;

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

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

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

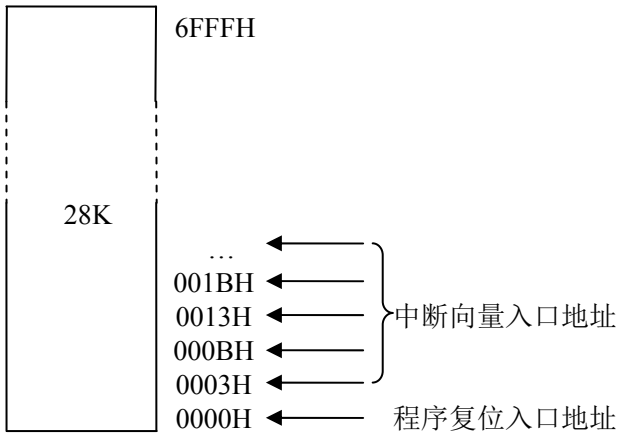
7 存储器

STC8H 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。

STC8H 系列单片机内部集成了大容量的数据存储器，STC8H1K16 系列单片机内部有 1024+256 字节的数据存储器。STC8H 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。STC8H 系列单片内部集成了 28K 字节的 Flash 程序存储器。



单片机复位后，程序计数器(PC)的内容为 0000H，从 0000H 单元开始执行程序。另外中断服务程序的入口地址(又称中断向量)也位于程序存储器单元。在程序存储器中，每个中断都有一个固定的入口地址，当中断发生并得到响应后，单片机就会自动跳转到相应的中断入口地址去执行程序。外部中断 0（INT0）的中断服务程序的入口地址是 0003H，定时器/计数器 0（TIMER0）中断服务程序的入口地址是 000BH，外部中断 1（INT1)的中断服务程序的入口地址是 0013H,定时器/计数器 1(TIMER1)的中断服务程序的入口地址是 001BH 等。更多的中断服务程序的入口地址(中断向量)请参考中断介绍章节。

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

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

7.2 数据存储器

STC8H 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。

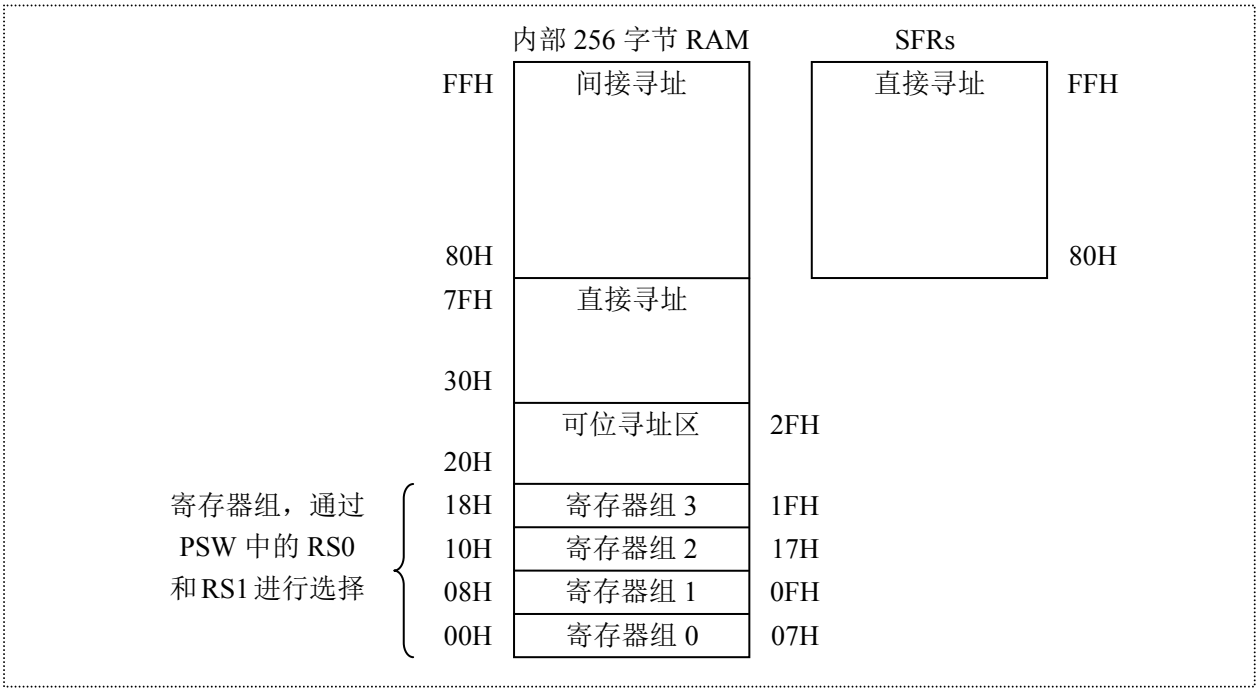
单片机系列	内部直接访问 RAM（DATA）	内部直接访问 RAM（IDATA）	内部扩展 RAM（XDATA）
STC8H1K16 系列	128 字节	128 字节	1024 字节

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

7.2.1 内部RAM

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

内部 RAM 的结构如下图所示：



低 128 字节 RAM 也称通用 RAM 区。通用 RAM 区又可分为工作寄存器组区，可位寻址区，用户 RAM 区和堆栈区。工作寄存器组区地址从 00H~1FH 共 32 字节单元，分为 4 组，每一组称为一个寄存器组，每组包含 8 个 8 位的工作寄存器，编号均为 R0 ~ R7，但属于不同的物理空间。通过使用工作寄存器组，可以提高运算速度。R0~R7 是常用的寄存器，提供 4 组是因为 1 组往往不够用。程序状态字 PSW 寄存器中的 RS1 和 RS0 组合决定当前使用的工作寄存器组，见下面 PSW 寄存器的介绍。

PSW（程序状态寄存器）

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

RS1，RS0：工作寄存器选择位

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

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

内部 RAM 中的 30H~FFH 单元是用户 RAM 和堆栈区。一个 8 位的堆栈指针(SP)，用于指向堆栈区。单片机复位后，堆栈指针 SP 为 07H，指向了工作寄存器组 0 中的 R7，因此，用户初始化程序都应对 SP 设置初值，一般设置在 80H 以后的单元为宜。

堆栈指针是一个 8 位专用寄存器。它指示出堆栈顶部在内部 RAM 块中的位置。系统复位后，SP 初始化位 07H，使得堆栈事实上由 08H 单元开始，考虑 08H~1FH 单元分别属于工作寄存器组 1~3，若在程序设计中用到这些区，则最好把 SP 值改变为 80H 或更大的值为宜。STC8 系列单片机的堆栈是向上生长的，即将数据压入堆栈后，SP 内容增大。

7.2.2 内部扩展RAM

STC8H 系列单片机片内除了集成 256 字节的内部 RAM 外，还集成了内部的扩展 RAM。访问内部扩展 RAM 的方法和传统 8051 单片机访问外部扩展 RAM 的方法相同，但是不影响 P0 口(数据总线和高八位地址总线)、P2 口(低八位地址总线)、以及 RD、WR 和 ALE 等端口上的信号。

在汇编语言中，内部扩展 RAM 通过 MOVX 指令访问，

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

在 C 语言中，可使用 xdata/pdata 声明存储类型即可。如：

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

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

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

AUXR（辅助寄存器）

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

EXTRAM：扩展 RAM 访问控制

0：访问内部扩展 RAM。

1：内部扩展 RAM 被禁用。

7.3 存储器中的特殊参数

STC8H 系列单片机内部的数据存储器和程序存储器中保存有与芯片相关的一些特殊参数，包括：全球唯一 ID 号、32K 掉电唤醒定时器的频率、内部 Bandgap 电压值以及 IRC 参数。

这些参数在程序存储器（ROM）中的存放地址分别如下：

参数名称	保存地址		参数说明
	STC8K1K16	STC8H1K28	
全球唯一 ID 号	3FF9H~3FFFH	6FF9H~6FFFH	7 字节
Bandgap 电压值	3FF7H~3FF8H	6FF7H~6FF8H	电压单位为毫伏
32K 掉电唤醒定时器的频率	3FF5H~3FF6H	6FF5H~6FF6H	单位 Hz
22.1184MHz 的 IRC 参数	3FF4H	6FF4H	—
24MHz 的 IRC 参数	3FF3H	6FF3H	—

这些参数在数据存储器（RAM）中的存放地址分别如下：

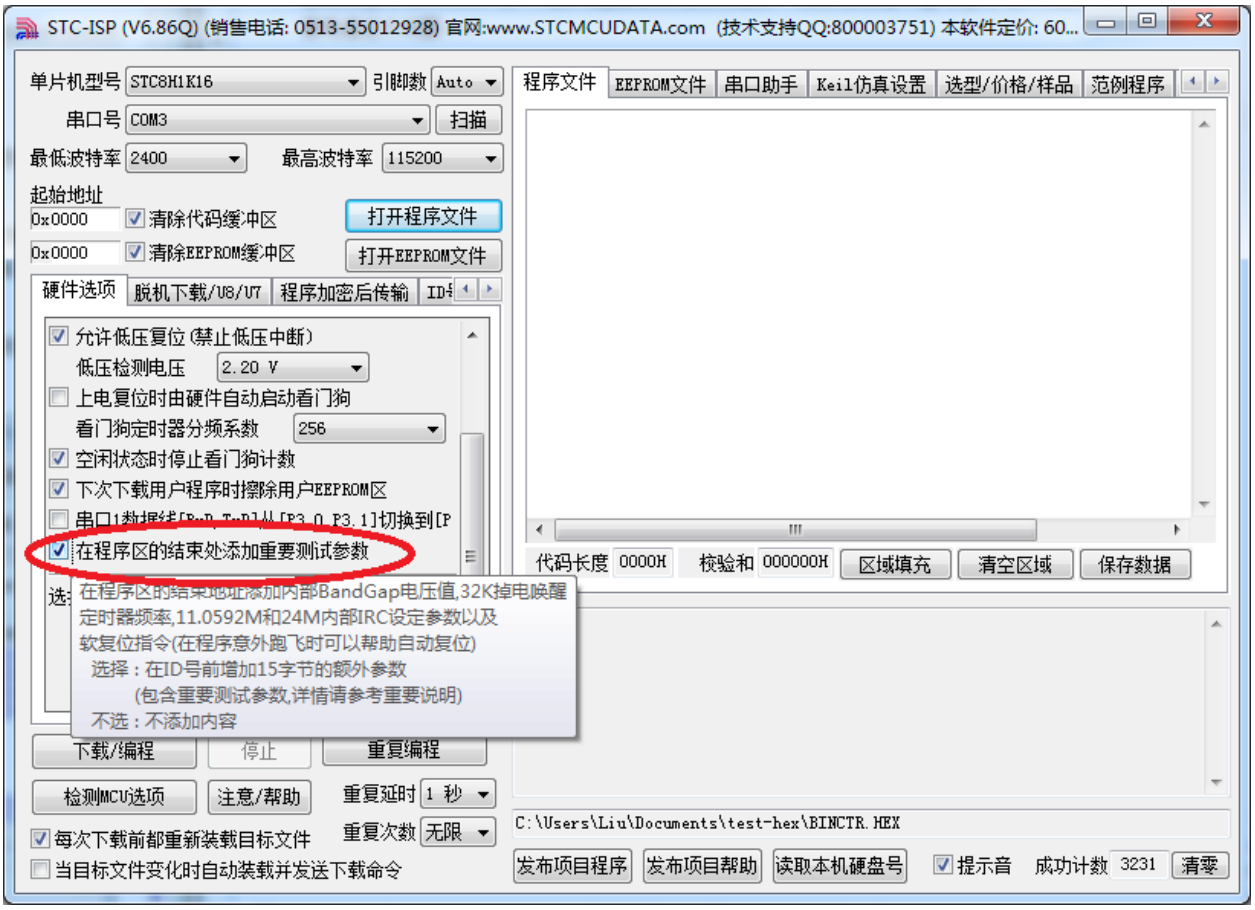
参数名称	保存地址	参数说明
Bandgap 电压值	idata：0EFH~0F0H	电压单位为毫伏，高字节在前
全球唯一 ID 号	idata：0F1H~0F7H	7 字节
32K 掉电唤醒定时器的频率	idata：0F8H~0F9H	单位 Hz，高字节在前
22.1184MHz 的 IRC 参数	idata：0FAH	—
24MHz 的 IRC 参数	idata：0FBH	—

特别说明

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 STC8H1K28 这个型号的 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 Bandgap 电压值、32K 掉电唤醒定时器的频率以及 IRC 参数都是没有的，需要在 ISP 下载时选择如下图所示的选项才可用。



7.3.1 读取Bandgap电压值 (从ROM中读取)

汇编代码

```
AUXR    DATA    8EH
BGV     EQU     03FF7H          ;STC8H1K16
;BGV    EQU     06FF7H          ;STC8H1K28

BUSY    BIT     20H.0

        ORG     0000H
        LJMP    MAIN
        ORG     0023H
        LJMP    UART_ISR
```

```
                ORG     0100H

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

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     SBUF,A
                RET

MAIN:
                MOV     SP,#3FH

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#BGV
                CLR     A
                MOVC    A,@A+DPTR               ;读取 Bandgap 电压的高字节
                LCALL   UART_SEND
                MOV     A,#1
                MOVC    A,@A+DPTR               ;读取 Bandgap 电压的低字节
                LCALL   UART_SEND

LOOP:
                JMP     LOOP

                END
```

C 语言代码

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

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

sfr         AUXR        =   0x8e;
```

```
bit         busy;
int         *BGV;

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

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

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

void main()
{
    BGV = (int code *)0x3ff7;           // STC8H1K16
//  BGV = (int code *)0x6ff7;           // STC8H1K28
    UartInit();
    ES = 1;
    EA = 1;
    UartSend(*BGV >> 8);                //读取 Bandgap 电压的高字节
    UartSend(*BGV);                     //读取 Bandgap 电压的低字节

    while (1);
}
```

7.3.2 读取Bandgap电压值 (从RAM中读取)

汇编代码

```
AUXR        DATA    8EH
BGV         DATA    0EFH

BUSY        BIT     20H.0

            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,#3FH

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     R0,#BGV
                MOV     A,@R0           ;读取 Bandgap 电压的高字节
                LCALL   UART_SEND
                INC     R0
                MOV     A,@R0           ;读取 Bandgap 电压的低字节
                LCALL   UART_SEND

LOOP:
                JMP     LOOP

                END
```

C 语言代码

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

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

```
sfr         AUXR        =   0x8e;

bit         busy;
int         *BGV;

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

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

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

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

    while (1);
}
```

7.3.3 读取全球唯一ID号 (从ROM中读取)

汇编代码

```
AUXR        DATA        8EH
ID          EQU         03FF9H              ; STC8H1K16
;ID         EQU         06FF9H              ; STC8H1K28

BUSY        BIT         20H.0
```

```
                ORG     0000H
                LJMP    MAIN
                ORG     0023H
                LJMP    UART_ISR

                ORG     0100H

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

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     TH1,#0FFH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     SBUF,A
                RET

MAIN:
                MOV     SP,#3FH

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

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

LOOP:
                JMP     LOOP

                END
```

C 语言代码

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

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

sfr       AUXR        =     0x8e;

bit       busy;
char      *ID;

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

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

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

void main()
{
    char i;

    ID = (char code *)0x3ff9;           // STC8H1K16
//  ID = (char code *)0x6ff9;           // STC8H1K28
    UartInit();
    ES = 1;
    EA = 1;

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

    while (1);
}
```

7.3.4 读取全球唯一ID号 (从RAM中读取)

汇编代码

```
AUXR        DATA    8EH
ID          DATA    0F1H

BUSY        BIT     20H.0

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#00H
            MOV     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,#3FH

            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
```

C 语言代码

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

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

sfr        AUXR        =   0x8e;

bit        busy;
char       *ID;

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

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

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

void main()
{
    char i;

    ID = (char idata *)0xf1;
    UartInit();
    ES = 1;
    EA = 1;

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

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

7.3.5 读取 32K掉电唤醒定时器的频率 (从ROM中读取)

汇编代码

```
AUXR        DATA    8EH
F32K        EQU     03FF5H              ; STC8H1K16
;F32K       EQU     06FF5H              ; STC8H1K28

BUSY        BIT     20H.0

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#00H
            MOV     TL1,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     TH1,#0FFH
            SETB    TR1
            MOV     AUXR,#40H
            CLR     BUSY
            RET

UART_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     SBUF,A
            RET

MAIN:
            MOV     SP,#3FH

            LCALL   UART_INIT
            SETB    ES
            SETB    EA

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

```
                CLR     A
                MOVC    A,@A+DPTR               ;读取 32K 频率的低字节
                LCALL   UART_SEND

LOOP:
                JMP     LOOP

                END
```

C 语言代码

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

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

sfr         AUXR        =   0x8e;

bit         busy;
int         *F32K;

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

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

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

void main()
{
    F32K = (int code *)0x3ff5;          // STC8H1K16
//  F32K = (int code *)0x6ff5;          // STC8H1K28
    UartInit();
    ES = 1;
```

```
    EA = 1;

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

    while (1);
}
```

7.3.6 读取 32K掉电唤醒定时器的频率 (从RAM中读取)

汇编代码

```
AUXR        DATA    8EH
F32K        DATA    0F8H

BUSY        BIT     20H.0

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#00H
            MOV     TL1,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     TH1,#0FFH
            SETB    TR1
            MOV     AUXR,#40H
            CLR     BUSY
            RET

UART_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     SBUF,A
            RET

MAIN:
            MOV     SP,#3FH

            LCALL   UART_INIT
            SETB    ES
            SETB    EA

            MOV     R0,#F32K
            MOV     A,@R0               ;读取 32K 频率的高字节
```

```
                LCALL   UART_SEND
                INC     R0
                MOV     A,@R0                   ;读取 32K 频率的低字节
                LCALL   UART_SEND

LOOP:
                JMP     LOOP

                END
```

C 语言代码

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

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

sfr         AUXR        =   0x8e;

bit         busy;
int         *F32K;

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

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

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

void main()
{
    F32K = (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);
}
```

8 特殊功能寄存器

8.1 STC8H1K16 系列

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
F8H								RSTCFG
F0H	B					IAP_TPS		
E8H								AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H							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	VOCTRL	ADC_CONTR	ADC_RES	ADC_RESL	
B0H	P3	P3M1	P3M0			IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH	S3CON	S3BUF	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	S4CON	S4BUF		PCON

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FEF8H	PWM2_CCR2L	PWM2_CCR3H	PWM2_CCR3L	PWM2_CCR4H	PWM2_CCR4L	PWM2_BKR	PWM2_DTR	PWM2_OISR
FEF0H	PWM2_PSCRH	PWM2_PSCRL	PWM2_ARRH	PWM2_ARRL	PWM2_RCR	PWM2_CCR1H	PWM2_CCR1L	PWM2_CCR2H
FEE8H	PWM2_CCMR1	PWM2_CCMR2	PWM2_CCMR3	PWM2_CCMR4	PWM2_CCER1	PWM2_CCER2	PWM2_CNTRH	PWM2_CNTRL
FEE0H	PWM2_CR1	PWM2_CR2	PWM2_SMCR	PWM2_ETR	PWM2_IER	PWM2_SR1	PWM2_SR2	PWM2_EGR
FED8H	PWM1_CCR2L	PWM1_CCR3H	PWM1_CCR3L	PWM1_CCR4H	PWM1_CCR4L	PWM1_BKR	PWM1_DTR	PWM1_OISR
FED0H	PWM1_PSCRH	PWM1_PSCRL	PWM1_ARRH	PWM1_ARRL	PWM1_RCR	PWM1_CCR1H	PWM1_CCR1L	PWM1_CCR2H
FEC8H	PWM1_CCMR1	PWM1_CCMR2	PWM1_CCMR3	PWM1_CCMR4	PWM1_CCER1	PWM1_CCER2	PWM1_CNTRH	PWM1_CNTRL
FEC0H	PWM1_CR1	PWM1_CR2	PWM1_SMCR	PWM1_ETR	PWM1_IER	PWM1_SR1	PWM1_SR2	PWM1_EGR
FEB0H	PWM1_ETRPS	PWM1_ENO	PWM1_PS	PWM1_IOAUX	PWM2_ETRPS	PWM2_ENO	PWM2_PS	PWM2_IOAUX
FEA8H	ADCTIM							
FEA0H			TM2PS	TM3PS	TM4PS			
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H	P0IE	P1IE						
FE28H	P0DR	P1DR	P2DR	P3DR		P5DR		
FE20H	P0SR	P1SR	P2SR	P3SR		P5SR		
FE18H	P0NCS	P1NCS	P2NCS	P3NCS		P5NCS		
FE10H	P0PU	P1PU	P2PU	P3PU		P5PU		
FE00H	CKSEL	CLKDIV	IRC24MCR	XOSCCR	IRC32KCR	MCLKOCR		

8.2 特殊功能寄存器列表

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	P0 端口	80H									1111,1111
SP	堆栈指针	81H									0000,0111
DPL	数据指针（低字节）	82H									0000,0000
DPH	数据指针（高字节）	83H									0000,0000
S4CON	串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	串口 4 数据寄存器	85H									0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	定时器 0 低 8 为寄存器	8AH									0000,0000
TL1	定时器 1 低 8 为寄存器	8BH									0000,0000
TH0	定时器 0 高 8 为寄存器	8CH									0000,0000
TH1	定时器 1 高 8 为寄存器	8DH									0000,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	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									1111,1111
P1M1	P1 口配置寄存器 1	91H									0000,0000
P1M0	P1 口配置寄存器 0	92H									0000,0000
P0M1	P0 口配置寄存器 1	93H									0000,0000
P0M0	P0 口配置寄存器 0	94H									0000,0000
P2M1	P2 口配置寄存器 1	95H									0000,0000
P2M0	P2 口配置寄存器 0	96H									0000,0000
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	串口 1 数据寄存器	99H									0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S2BUF	串口 2 数据寄存器	9BH									0000,0000
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									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
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	串口 3 数据寄存器	ADH									0000,0000
TA	DPTR 时序控制寄存器	AEH									0000,0000
IE2	中断允许寄存器 2	AFH	-	ET4	ET3	ES4	ES3	ET2	ESPI	ES2	x000,0000

P3	P3 端口	B0H								1111,1111
P3M1	P3 口配置寄存器 1	B1H								n000,0000
P3M0	P3 口配置寄存器 0	B2H								n000,0000
IP2	中断优先级控制寄存器 2	B5H	-	PI2C	PCMP	PX4	PPWM2	PPWM1	PSPI	PS2	x000,0000
IP2H	高中断优先级控制寄存器 2	B6H	-	PI2CH	PCMPH	PX4H	PPWMF2	PPWMH1	PSPIH	PS2H	x000,0000
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]		CMPO_S	S4_S	S3_S	S2_S	0x00,0000
VOCTRL	电压控制寄存器	BBH	SCC	-	-	-	-	-	0	0	0xxx,xx00
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]				000x,0000
ADC_RES	ADC 转换结果高位寄存器	BDH									0000,0000
ADC_RESL	ADC 转换结果低位寄存器	BEH									0000,0000
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	-	-	-		-	-	-	-	xxx1,xxxx
P5M1	P5 口配置寄存器 1	C9H	-	-	-		-	-	-	-	xxx1,xxxx
P5M0	P5 口配置寄存器 0	CAH	-	-	-		-	-	-	-	xxx0,xxxx
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI 控制寄存器	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100
SPDAT	SPI 数据寄存器	CFH									0000,0000
PSW	程序状态字寄存器	D0H	CY	AC	F0	RS1	RS0	OV	-	P	0000,00x0
T4T3M	定时器 4/3 控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
T4H	定时器 4 高字节	D2H									0000,0000
T4L	定时器 4 低字节	D3H									0000,0000
T3H	定时器 3 高字节	D4H									0000,0000
T3L	定时器 3 低字节	D5H									0000,0000
T2H	定时器 2 高字节	D6H									0000,0000
T2L	定时器 2 低字节	D7H									0000,0000
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
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
IAP_TPS	IAP 等待时间控制寄存器	F5H	-	-	IAPTPS[5:0]						xx00,0000

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	
CKSEL	时钟选择寄存器	FE00H	-	-	-	-	-	-	MCKSEL[1:0]		xxxx,xx00
CLKDIV	时钟分频寄存器	FE01H									0000,0100
IRC24MCR	内部 24M 振荡器控制寄存器	FE02H	ENIRC24M	-	-	-	-	-	-	IRC24MST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	内部 32K 振荡器控制寄存器	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	主时钟输出控制寄存器	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
P0PU	P0 口上拉电阻控制寄存器	FE10H	-	-	-	-					xxxx,0000
P1PU	P1 口上拉电阻控制寄存器	FE11H									0000,0000
P2PU	P2 口上拉电阻控制寄存器	FE12H									0000,0000
P3PU	P3 口上拉电阻控制寄存器	FE13H									0000,0000
P5PU	P5 口上拉电阻控制寄存器	FE15H	-	-	-		-	-	-	-	xxx0,xxxx
P0NCS	P0 口施密特触发控制寄存器	FE18H	-	-	-	-					xxxx,0000
P1NCS	P1 口施密特触发控制寄存器	FE19H									0000,0000
P2NCS	P2 口施密特触发控制寄存器	FE1AH									0000,0000
P3NCS	P3 口施密特触发控制寄存器	FE1BH									0000,0000
P5NCS	P5 口施密特触发控制寄存器	FE1DH	-	-	-		-	-	-	-	xxx0,xxxx
P0SR	P0 口电平转换速率寄存器	FE20H	-	-	-	-					xxxx,0000
P1SR	P1 口电平转换速率寄存器	FE21H									0000,0000
P2SR	P2 口电平转换速率寄存器	FE22H									0000,0000
P3SR	P3 口电平转换速率寄存器	FE23H									0000,0000
P5SR	P5 口电平转换速率寄存器	FE25H	-	-	-		-	-	-	-	xxx0,xxxx
P0DR	P0 口驱动电流控制寄存器	FE28H	-	-	-	-					xxxx,1111
P1DR	P1 口驱动电流控制寄存器	FE29H									1111,1111
P2DR	P2 口驱动电流控制寄存器	FE2AH									1111,1111
P3DR	P3 口驱动电流控制寄存器	FE2BH									1111,1111
P5DR	P5 口驱动电流控制寄存器	FE2DH	-	-	-		-	-	-	-	xxx1,xxxx
P0IE	P0 口输入使能控制寄存器	FE30H	-	-	-	-					xxxx,1111
P1IE	P1 口输入使能控制寄存器	FE31H									1111,1111
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	SLADR[6:0]							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
PWM1_ETRPS	PWM1 的 ETR 选择寄存器	FEB0H						BRK1PS	ETR1PS[1:0]	xxxx,x000
PWM1_ENO	PWM1 输出使能控制	FEB1H	ENO4N	ENO4P	ENO3N	ENO3P	ENO2N	ENO2P	ENO1N	ENO1P	0000,0000
PWM1_PS	PWM1 输出脚选择寄存器	FEB2H	C4PS[1:0]		C3PS[1:0]		C2PS[1:0]		C1PS[1:0]		0000,0000
PWM1_IOAUX	PWM1 辅助寄存器	FEB3H	AUX4N	AUX4P	AUX3N	AUX3P	AUX2N	AUX2P	AUX1N	AUX1P	0000,0000
PWM2_ETRPS	PWM2 的 ETR 选择寄存器	FEB4H						BRK2PS	ETR2PS[1:0]		xxxx,x000
PWM2_ENO	PWM2 输出使能控制	FEB5H	-	ENO8P	-	ENO7P	-	ENO6P	-	ENO5P	x0x0,x0x0
PWM2_PS	PWM2 输出脚选择寄存器	FEB6H	C8PS[1:0]		C7PS[1:0]		C6PS[1:0]		C5PS[1:0]		0000,0000
PWM2_IOAUX	PWM2 辅助寄存器	FEB7H	-	AUX8P	-	AUX7P	-	AUX6P	-	AUX5P	x0x0,x0x0
PWM1_CR1	PWM1 控制寄存器 1	FEC0H	ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN	0000,0000
PWM1_CR2	PWM1 控制寄存器 2	FEC1H	-	MMS[2:0]			-	COMS	-	CCPC	x000,x0x0
PWM1_SMCR	PWM1 从模式控制寄存器	FEC2H	MSM	TS[2:0]			-	SMS[2:0]			0000,x000
PWM1_ETR	PWM1 外部触发寄存器	FEC3H	ETP	ECE	ETPS[1:0]		ETF[3:0]				0000,0000
PWM1_IER	PWM1 中断使能寄存器	FEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE	0000,0000
PWM1_SR1	PWM1 状态寄存器 1	FEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF	0000,0000
PWM1_SR2	PWM1 状态寄存器 2	FEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-	xxx0,000x
PWM1_EGR	PWM1 事件发生寄存器	FEC7H	BG	TG	COMG	CC4G	CC3G	CC2G	CC1G	UG	0000,0000
PWM1_CCMR1	PWM1 捕获模式寄存器 1	FEC8H	OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]		0000,0000
	PWM1 比较模式寄存器 1		IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]		0000,0000
PWM1_CCMR2	PWM1 捕获模式寄存器 2	FEC9H	OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]		0000,0000
	PWM1 比较模式寄存器 2		IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]		0000,0000
PWM1_CCMR3	PWM1 捕获模式寄存器 3	FECAH	OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]		0000,0000
	PWM1 比较模式寄存器 3		IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]		0000,0000
PWM1_CCMR4	PWM1 捕获模式寄存器 4	FECBH	OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]		0000,0000
	PWM1 比较模式寄存器 4		IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]		0000,0000
PWM1_CCER1	PWM1 捕获比较使能寄存器 1	FECCH	CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E	0000,0000
PWM1_CCER2	PWM1 捕获比较使能寄存器 2	FECDH	CC4NP	CC4NE	CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E	0000,0000
PWM1_CNTRH	PWM1 计数器高字节	FECEH	CNT[15:8]								0000,0000
PWM1_CNTRL	PWM1 计数器低字节	FECFH	CNT[7:0]								0000,0000
PWM1_PSCRH	PWM1 预分频高字节	FED0H	PSC[15:8]								0000,0000
PWM1_PSCRL	PWM1 预分频低字节	FED1H	PSC[7:0]								0000,0000
PWM1_ARRH	PWM1 自动重装寄存器高字节	FED2H	ARR[15:8]								0000,0000
PWM1_ARRL	PWM1 自动重装寄存器低字节	FED3H	ARR[7:0]								0000,0000
PWM1_RCR	PWM1 重复计数器寄存器	FED4H	REP[7:0]								0000,0000
PWM1_CCR1H	PWM1 比较捕获寄存器 1 高位	FED5H	CCR1[15:8]								0000,0000
PWM1_CCR1L	PWM1 比较捕获寄存器 1 低位	FED6H	CCR1[7:0]								0000,0000
PWM1_CCR2H	PWM1 比较捕获寄存器 2 高位	FED7H	CCR2[15:8]								0000,0000
PWM1_CCR2L	PWM1 比较捕获寄存器 2 低位	FED8H	CCR2[7:0]								0000,0000
PWM1_CCR3H	PWM1 比较捕获寄存器 3 高位	FED9H	CCR3[15:8]								0000,0000
PWM1_CCR3L	PWM1 比较捕获寄存器 3 低位	FEDAH	CCR3[7:0]								0000,0000
PWM1_CCR4H	PWM1 比较捕获寄存器 4 高位	FEDBH	CCR4[15:8]								0000,0000

PWM1_CCR4L	PWM1 比较捕获寄存器 4 低位	FEDCH	CCR4[7:0]								0000,0000
PWM1_BKR	PWM1 刹车寄存器	FEDDH	MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		0000,000x
PWM1_DTR	PWM1 死区控制寄存器	FEDEH	DTG[7:0]								0000,0000
PWM1_OISR	PWM1 输出空闲状态寄存器	FEDFH	OIS4N	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1	0000,0000
PWM2_CR1	PWM2 控制寄存器 1	FEEH	ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN	0000,0000
PWM2_CR2	PWM2 控制寄存器 2	FEE1H	-	MMS[2:0]			-	COMS	-	CCPC	x000,x0x0
PWM2_SMCR	PWM2 从模式控制寄存器	FEE2H	MSM	TS[2:0]			-	SMS[2:0]			0000,x000
PWM2_ETR	PWM2 外部触发寄存器	FEE3H	ETP	ECE	ETPS[1:0]		ETF[3:0]				0000,0000
PWM2_IER	PWM2 中断使能寄存器	FEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE	0000,0000
PWM2_SR1	PWM2 状态寄存器 1	FEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF	0000,0000
PWM2_SR2	PWM2 状态寄存器 2	FEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-	xxx0,000x
PWM2_EGR	PWM2 事件发生寄存器	FEE7H	BG	TG	COMG	CC8G	CC7G	CC6G	CC5G	UG	0000,0000
PWM2_CCMR1	PWM2 捕获模式寄存器 1	FEE8H	OC5CE	OC5M[2:0]			OC5PE	OC5FE	CC5S[1:0]		0000,0000
	PWM2 比较模式寄存器 1		IC5F[3:0]				IC5PSC[1:0]		CC5S[1:0]		0000,0000
PWM2_CCMR2	PWM2 捕获模式寄存器 2	FEE9H	OC6CE	OC6M[2:0]			OC6PE	OC6FE	CC6S[1:0]		0000,0000
	PWM2 比较模式寄存器 2		IC6F[3:0]				IC6PSC[1:0]		CC6S[1:0]		0000,0000
PWM2_CCMR3	PWM2 捕获模式寄存器 3	FEEAH	OC7CE	OC7M[2:0]			OC7PE	OC7FE	CC7S[1:0]		0000,0000
	PWM2 比较模式寄存器 3		IC7F[3:0]				IC7PSC[1:0]		CC7S[1:0]		0000,0000
PWM2_CCMR4	PWM2 捕获模式寄存器 4	FEEBH	OC8CE	OC8M[2:0]			OC8PE	OC8FE	CC8S[1:0]		0000,0000
	PWM2 比较模式寄存器 4		IC8F[3:0]				IC8PSC[1:0]		CC8S[1:0]		0000,0000
PWM2_CCER1	PWM2 捕获比较使能寄存器 1	FEECH	-	-	CC6P	CC6E	-	-	CC5P	CC5E	xx00,xx00
PWM2_CCER2	PWM2 捕获比较使能寄存器 2	FEEDH	-	-	CC8P	CC8E	-	-	CC7P	CC7E	xx00,xx00
PWM2_CNTRH	PWM2 计数器高字节	FEEEH	CNT[15:8]								0000,0000
PWM2_CNTRL	PWM2 计数器低字节	FEEFH	CNT[7:0]								0000,0000
PWM2_PSCRH	PWM2 预分频高字节	FEF0H	PSC[15:8]								0000,0000
PWM2_PSCRL	PWM2 预分频低字节	FEF1H	PSC[7:0]								0000,0000
PWM2_ARRH	PWM2 自动重装寄存器高字节	FEF2H	ARR[15:8]								0000,0000
PWM2_ARRL	PWM2 自动重装寄存器低字节	FEF3H	ARR[7:0]								0000,0000
PWM2_RCR	PWM2 重复计数器寄存器	FEF4H	REP[7:0]								0000,0000
PWM2_CCR1H	PWM2 比较捕获寄存器 1 高位	FEF5H	CCR1[15:8]								0000,0000
PWM2_CCR1L	PWM2 比较捕获寄存器 1 低位	FEF6H	CCR1[7:0]								0000,0000
PWM2_CCR2H	PWM2 比较捕获寄存器 2 高位	FEF7H	CCR2[15:8]								0000,0000
PWM2_CCR2L	PWM2 比较捕获寄存器 2 低位	FEF8H	CCR2[7:0]								0000,0000
PWM2_CCR3H	PWM2 比较捕获寄存器 3 高位	FEF9H	CCR3[15:8]								0000,0000
PWM2_CCR3L	PWM2 比较捕获寄存器 3 低位	FEFAH	CCR3[7:0]								0000,0000
PWM2_CCR4H	PWM2 比较捕获寄存器 4 高位	FEFBH	CCR4[15:8]								0000,0000
PWM2_CCR4L	PWM2 比较捕获寄存器 4 低位	FEFCH	CCR4[7:0]								0000,0000
PWM2_BKR	PWM2 刹车寄存器	FEFDH	MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK[1:0]		0000,000x
PWM2_DTR	PWM2 死区控制寄存器	FEFEH	DTG[7:0]								0000,0000
PWM2_OISR	PWM2 输出空闲状态寄存器	FEFFH	-	OIS8	-	OIS7	-	OIS6	-	OIS5	x0x0,x0x0

9 I/O口

STC8H 系列单片机最多有 29 个 I/O 口。所有的 I/O 口均有 4 种工作模式：准双向口/弱上拉（标准 8051 输出口模式）、推挽输出/强上拉、高阻输入（电流既不能流入也不能流出）、开漏输出。可使用软件对 I/O 口的工作模式进行容易配置。

注意：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

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	-	-	-	-					xxxx,1111
P1	P1 端口	90H									1111,1111
P2	P2 端口	A0H									1111,1111
P3	P3 端口	B0H									1111,1111
P5	P5 端口	C8H	-	-	-		-	-	-	-	xx1x,xxxx
P0M1	P0 口配置寄存器 1	93H	-	-	-	-					xxxx,1111
P0M0	P0 口配置寄存器 0	94H	-	-	-	-					xxxx,0000
P1M1	P1 口配置寄存器 1	91H									1111,1111
P1M0	P1 口配置寄存器 0	92H									0000,0000
P2M1	P2 口配置寄存器 1	95H									1111,1111
P2M0	P2 口配置寄存器 0	96H									0000,0000
P3M1	P3 口配置寄存器 1	B1H									0111,1111
P3M0	P3 口配置寄存器 0	B2H									0000,0000
P5M1	P5 口配置寄存器 1	C9H	-	-	-		-	-	-	-	xx1x,xxxx
P5M0	P5 口配置寄存器 0	CAH	-	-	-		-	-	-	-	xx0x,xxxx

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	-	-	-	-					xxxx,0000
P1PU	P1 口上拉电阻控制寄存器	FE11H									0000,0000
P2PU	P2 口上拉电阻控制寄存器	FE12H									0000,0000
P3PU	P3 口上拉电阻控制寄存器	FE13H									0000,0000
P5PU	P5 口上拉电阻控制寄存器	FE15H	-	-	-		-	-	-	-	xxx0,xxxx
P0NCS	P0 口施密特触发控制寄存器	FE18H	-	-	-	-					xxxx,0000
P1NCS	P1 口施密特触发控制寄存器	FE19H									0000,0000
P2NCS	P2 口施密特触发控制寄存器	FE1AH									0000,0000
P3NCS	P3 口施密特触发控制寄存器	FE1BH									0000,0000
P5NCS	P5 口施密特触发控制寄存器	FE1DH	-	-	-		-	-	-	-	xxx0,xxxx
P0SR	P0 口电平转换速率寄存器	FE20H	-	-	-	-					xxxx,0000
P1SR	P1 口电平转换速率寄存器	FE21H									0000,0000
P2SR	P2 口电平转换速率寄存器	FE22H									0000,0000
P3SR	P3 口电平转换速率寄存器	FE23H									0000,0000

P5SR	P5 口电平转换速率寄存器	FE25H	-	-	-		-	-	-	-	xxx0,xxxx
P0DR	P0 口驱动电流控制寄存器	FE28H	-	-	-	-					xxxx,1111
P1DR	P1 口驱动电流控制寄存器	FE29H									1111,1111
P2DR	P2 口驱动电流控制寄存器	FE2AH									1111,1111
P3DR	P3 口驱动电流控制寄存器	FE2BH									1111,1111
P5DR	P5 口驱动电流控制寄存器	FE2DH	-	-	-		-	-	-	-	xxx1,xxxx
P0IE	P0 口输入使能控制寄存器	FE30H	-	-	-	-					xxxx,1111
P1IE	P1 口输入使能控制寄存器	FE31H									1111,1111

端口数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0	80H	-	-	-	-	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	-	-	-	P5.4	-	-	-	-

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

端口模式配置寄存器

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

配置端口的模式

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

端口上拉电阻控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	FE10H	-	-	-	-				
P1PU	FE11H								
P2PU	FE12H								
P3PU	FE13H								
P5PU	FE15H	-	-	-		-	-	-	-

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

0：禁止端口内部的 3.7K 上拉电阻（实测为 4.2K 左右）

1：使能端口内部的 3.7K 上拉电阻（实测为 4.2K 左右）

端口施密特触发控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0NCS	FE18H	-	-	-	-				
P1NCS	FE19H								
P2NCS	FE1AH								
P3NCS	FE1BH								
P5NCS	FE1DH	-	-	-		-	-	-	-

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。

端口电平转换速度控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0SR	FE20H	-	-	-	-				
P1SR	FE21H								
P2SR	FE22H								
P3SR	FE23H								
P5SR	FE25H	-	-	-		-	-	-	-

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

端口驱动电流控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0DR	FE28H	-	-	-	-				
P1DR	FE29H								
P2DR	FE2AH								
P3DR	FE2BH								
P5DR	FE2DH	-	-	-		-	-	-	-

控制端口的驱动能力

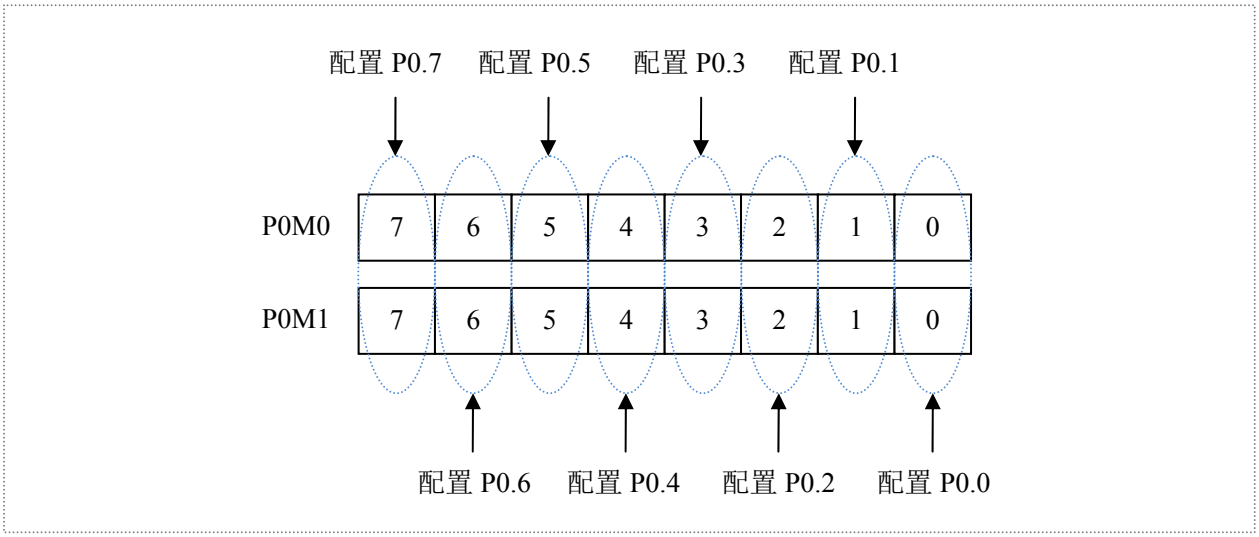
0：一般驱动能力

1：增强驱动能力

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 90mA，即从 VCC 流入的电流建议不要超过 90mA，从 GND 流出电流建议不要超过 90mA，整体流入/流出电流建议都不要超过 90mA。

9.3 I/O的结构图

9.3.1 准双向口（弱上拉）

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

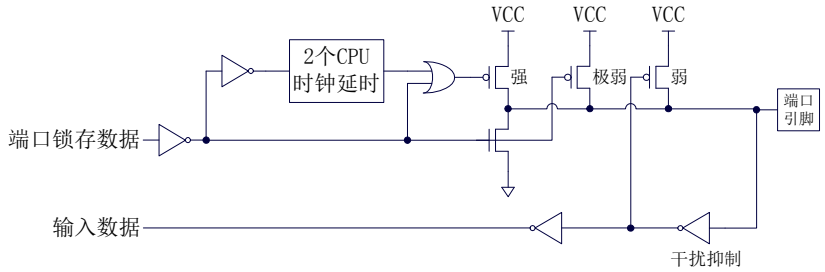
在 3 个上拉晶体管中，有 1 个上拉晶体管称为“弱上拉”，当端口寄存器为 1 且引脚本身也为 1 时打开。此上拉提供基本驱动电流使准双向口输出为 1。如果一个引脚输出为 1 而由外部装置下拉到低时，弱上拉关闭而“极弱上拉”维持开状态，为了把这个引脚强拉为低，外部装置必须有足够的灌电流能力使引脚上的电压降到门槛电压以下。对于 5V 单片机，“弱上拉”晶体管的电流约 250uA；对于 3.3V 单片机，“弱上拉”晶体管的电流约 150uA。

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

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

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

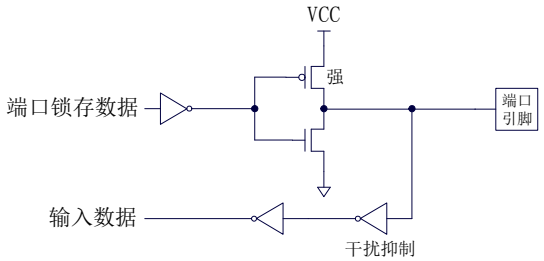
准双向口（弱上拉）输出如下图所示：



9.3.2 推挽输出

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

强推挽引脚配置如下图所示：

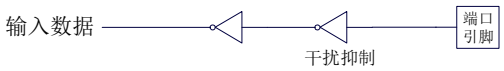


9.3.3 高阻输入

电流既不能流入也不能流出

输入口带有一个施密特触发输入以及一个干扰抑制电路

高阻输入引脚配置如下图所示：



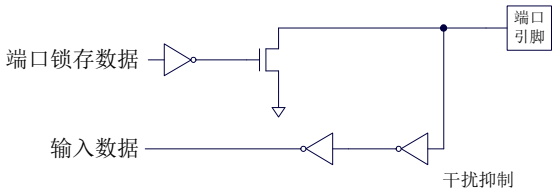
9.3.4 开漏输出

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

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

开漏端口带有一个施密特触发输入以及一个干扰抑制电路。

输出端口配置如下图所示：



9.4 范例程序

9.4.1 端口模式设置

汇编代码

```
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,#3FH

            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
```

C 语言代码

```
#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);
}
```

9.4.2 双向口读写操作

汇编代码

```
P0M0        DATA    094H
P0M1        DATA    093H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

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

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

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

            JMP     $

            END
```

C 语言代码

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

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

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

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

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

    while (1);
}
```

10 指令系统

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

ORL	direct,A	直接地址单元与累加器相或	2	1
ORL	direct,#data	直接地址单元与立即数相或	3	1
XRL	A,Rn	累加器与寄存器相异或	1	1
XRL	A,direct	累加器与直接地址单元相异或	2	1
XRL	A,@Ri	累加器与间接地址单元相异或	1	1
XRL	A,#data	累加器与立即数相异或	2	1
XRL	direct,A	直接地址单元与累加器相异或	2	1
XRL	direct,#data	直接地址单元与立即数相异或	3	1
CLR	A	累加器清0	1	1
CPL	A	累加器取反	1	1
RL	A	累加器循环左移	1	1
RLC	A	累加器带进位循环左移	1	1
RR	A	累加器循环右移	1	1
RRC	A	累加器带进位循环右移	1	1
SWAP	A	累加器高低半字节交换	1	1
CLR	C	清零进位位	1	1
CLR	bit	清0直接地址位	2	1
SETB	C	置1进位位	1	1
SETB	bit	置1直接地址位	2	1
CPL	C	进位位求反	1	1
CPL	bit	直接地址位求反	2	1
ANL	C,bit	进位位和直接地址位相与	2	1
ANL	C,/bit	进位位和直接地址位的反码相与	2	1
ORL	C,bit	进位位和直接地址位相或	2	1
ORL	C,/bit	进位位和直接地址位的反码相或	2	1
MOV	C,bit	直接地址位送入进位位	2	1
MOV	bit,C	进位位送入直接地址位	2	1
MOV	A,Rn	寄存器内容送入累加器	1	1
MOV	A,direct	直接地址单元中的数据送入累加器	2	1
MOV	A,@Ri	间接地址中的数据送入累加器	1	1
MOV	A,#data	立即数送入累加器	2	1
MOV	Rn,A	累加器内容送入寄存器	1	1
MOV	Rn,direct	直接地址单元中的数据送入寄存器	2	1
MOV	Rn,#data	立即数送入寄存器	2	1
MOV	direct,A	累加器内容送入直接地址单元	2	1
MOV	direct,Rn	寄存器内容送入直接地址单元	2	1
MOV	direct,direct	直接地址单元中的数据送入另一个直接地址单元	3	1

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

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

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

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

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

11 中断系统

中断系统是为使 CPU 具有对外界紧急事件的实时处理能力而设置的。

当中央处理机 CPU 正在处理某件事的时候外界发生了紧急事件请求，要求 CPU 暂停当前的工作，转而去处理这个紧急事件，处理完以后，再回到原来被中断的地方，继续原来的工作，这样的过程称为中断。实现这种功能的部件称为中断系统，请示 CPU 中断的请求源称为中断源。微型机的中断系统一般允许多个中断源，当几个中断源同时向 CPU 请求中断，要求为它服务的时候，这就存在 CPU 优先响应哪一个中断源请求的问题。通常根据中断源的轻重缓急排队，优先处理最紧急事件的中断请求源，即规定每一个中断源有一个优先级别。CPU 总是先响应优先级别最高的中断请求。

当 CPU 正在处理一个中断源请求的时候（执行相应的中断服务程序），发生了另外一个优先级比它还高的中断源请求。如果 CPU 能够暂停对原来中断源的服务程序，转而去处理优先级更高的中断请求源，处理完以后，再回到原低级中断服务程序，这样的过程称为中断嵌套。这样的中断系统称为多级中断系统，没有中断嵌套功能的中断系统称为单级中断系统。

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

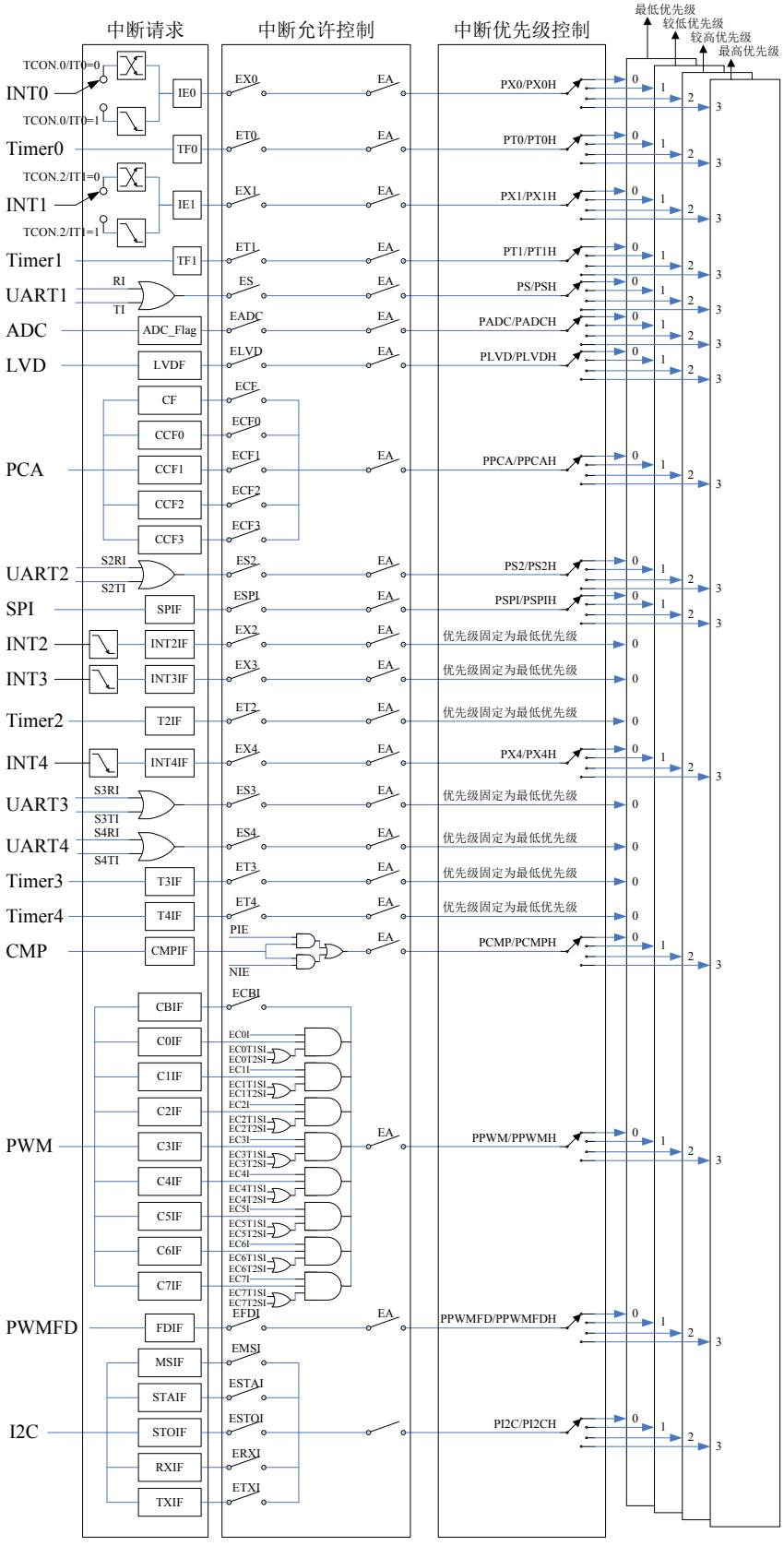
11.1 STC8H系列中断源

下表中 √ 表示对应的系列有相应的中断源

中断源	STC8H1K16系列
外部中断 0 中断（INT0）	√
定时器 0 中断（Timer0）	√
外部中断 1 中断（INT1）	√
定时器 1 中断（Timer1）	√
串口 1 中断（UART1）	√
模数转换中断（ADC）	√
低压检测中断（LVD）	√
捕获中断（CCP/PCA/PWM）	√
串口 2 中断（UART2）	√
串行外设接口中断（SPI）	√
外部中断 2 中断（INT2）	√
外部中断 3 中断（INT3）	√
定时器 2 中断（Timer2）	√
外部中断 4 中断（INT4）	√
串口 3 中断（UART3）	√
串口 4 中断（UART4）	√

定时器 3 中断（Timer3）	√
定时器 4 中断（Timer4）	√
比较器中断（CMP）	√
增强型 PWM 中断	
PWM 异常检测中断（PWMFD）	
I2C 总线中断	√
PWM1	√
PWM2	√

11.2 STC8H中断结构图



11.3 STC8H系列中断列表

中断源	中断向量	次序	优先级设置	优先级	中断请求位	中断允许位
INT0	0003H	0	PX0PX0H	0/1/2/3	IE0	EX0
Timer0	000BH	1	PT0,PT0H	0/1/2/3	TF0	ET0
INT1	0013H	2	PX1,PX1H	0/1/2/3	IE1	EX1
Timer1	001BH	3	PT1,PT1H	0/1/2/3	TF1	ET1
UART1	0023H	4	PS,PSH	0/1/2/3	RI || TI	ES
ADC	002BH	5	PADC,PADCH	0/1/2/3	ADC_FLAG	EADC
LVD	0033H	6	PLVD,PLVDH	0/1/2/3	LVDF	ELVD
PCA	003BH	7	PPCA,PPCAH	0/1/2/3	CF	ECF
					CCF0	ECCF0
					CCF1	ECCF1
					CCF2	ECCF2
					CCF3	ECCF3
UART2	0043H	8	PS2,PS2H	0/1/2/3	S2RI || S2TI	ES2
SPI	004BH	9	PSPI,PSPIH	0/1/2/3	SPIF	ESPI
INT2	0053H	10		0	INT2IF	EX2
INT3	005BH	11		0	INT3IF	EX3
Timer2	0063H	12		0	T2IF	ET2
INT4	0083H	16	PX4,PX4H	0/1/2/3	INT4IF	EX4
UART3	008BH	17	~~PS3,PS3H~~	~~0/1/2/3~~	S3RI || S3TI	ES3
UART4	0093H	18	~~PS4,PS4H~~	~~0/1/2/3~~	S4RI || S4TI	ES4
Timer3	009BH	19		0	T3IF	ET3
Timer4	00A3H	20		0	T4IF	ET4
CMP	00ABH	21	PCMP,PCMPH	0/1/2/3	CMPIF	PIE|NIE
I2C	00C3H	24	PI2C,PI2CH	0/1/2/3	MSIF	EMSI
					STAIF	ESTAI
					RXIF	ERXI
					TXIF	ETXI
					STOIF	ESTOI
PWM1	00D3H	26	PPWM1,PPWM1H	0/1/2/3	PWM1_SR	PWM1_IER
PWM2	00DDH	27	PPWM2,PPWM2H	0/1/2/3	PWM2_SR	PWM2_IER

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    UART3_Routine(void)     interrupt  17;
void    UART4_Routine(void)     interrupt  18;
void    TM3_Routine(void)       interrupt  19;
void    TM4_Routine(void)       interrupt  20;
void    CMP_Routine(void)       interrupt  21;
void    I2C_Routine(void)       interrupt  24;
void    PWM1_Routine(void)      interrupt  26;
void    PWM2_Routine(void)      interrupt  27;
```

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	ES4	ES3	ET2	ESPI	ES2	x000,0000
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
IP	中断优先级控制寄存器	B8H	-	PLVD	PADC	PS	PT1	PX1	PT0	PX0	0000,0000
IPH	高中断优先级控制寄存器	B7H	-	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H	0000,0000
IP2	中断优先级控制寄存器 2	B5H	-	PI2C	PCMP	PX4	PPWM2	PPWM1	PSPI	PS2	x000,0000
IP2H	高中断优先级控制寄存器 2	B6H	-	PI2CH	PCMPH	PX4H	PPWM2H	PPWM1H	PSPIH	PS2H	x000,0000
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
AUXINTIF	扩展外部中断标志寄存器	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S4CON	串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]				000x,0000
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
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
PWM1_IER	PWM1 中断使能寄存器	FEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE	0000,0000
PWM1_SR1	PWM1 状态寄存器 1	FEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF	0000,0000
PWM1_SR2	PWM1 状态寄存器 2	FEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-	xxx0,000x
PWM2_IER	PWM2 中断使能寄存器	FEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE	0000,0000
PWM2_SR1	PWM2 状态寄存器 1	FEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF	0000,0000
PWM2_SR2	PWM2 状态寄存器 2	FEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-	xxx0,000x

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	ES4	ES3	ET2	ESPI	ES2

ET4：定时/计数器 T4 的溢出中断允许位。

0：禁止 T4 中断

1：允许 T4 中断

ET3：定时/计数器 T3 的溢出中断允许位。

0：禁止 T3 中断

1：允许 T3 中断

ES4：串行口 4 中断允许位。

0：禁止串行口 4 中断

1：允许串行口 4 中断

ES3：串行口 3 中断允许位。

0：禁止串行口 3 中断

1：允许串行口 3 中断

ET2：定时/计数器 T2 的溢出中断允许位。

0：禁止 T2 中断

1：允许 T3 中断

ESPI：SPI 中断允许位。

0：禁止 SPI 中断

1：允许 SPI 中断

ES2：串行口 2 中断允许位。

0：禁止串行口 2 中断

1：允许串行口 2 中断

INTCLKO（外部中断与时钟输出控制寄存器）

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

EX4：外部中断 4 中断允许位。

0：禁止 INT4 中断

1：允许 INT4 中断

EX3：外部中断 3 中断允许位。

0：禁止 INT3 中断

1：允许 INT3 中断

EX2：外部中断 2 中断允许位。

0：禁止 INT2 中断

1：允许 INT2 中断

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 I^2C 从机发送数据完成事件中断

1：允许 I^2C 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

PWM1 中断使能寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM1_IER	FEC4H	BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE

BIE：PWM1刹车中断允许位。

0：禁止 PWM1 刹车中断

1：允许 PWM1 刹车中断

TIE：PWM1触发中断允许位。

0：禁止 PWM1 触发中断

1：允许 PWM1 触发中断

COMIE：PWM1比较中断允许位。

0：禁止 PWM1 比较中断

1：允许 PWM1 比较中断

CC4IE：PWM1捕获比较通道4中断允许位。

0：禁止 PWM1 捕获比较通道 4 中断

1：允许 PWM1 捕获比较通道 4 中断

CC3IE：PWM1捕获比较通道3中断允许位。

0：禁止 PWM1 捕获比较通道 3 中断

1：允许 PWM1 捕获比较通道 3 中断

CC2IE：PWM1捕获比较通道2中断允许位。

0：禁止 PWM1 捕获比较通道 2 中断

1：允许 PWM1 捕获比较通道 2 中断

CC1IE：PWM1捕获比较通道1中断允许位。

0：禁止 PWM1 捕获比较通道 1 中断

1：允许 PWM1 捕获比较通道 1 中断

UIE：PWM1更新中断允许位。

0：禁止 PWM1 更新中断

1：允许 PWM1 更新中断

PWM2 中断使能寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM2_IER	FEE4H	BIE	TIE	COMIE	CC8IE	CC7IE	CC6IE	CC5IE	UIE

BIE：PWM2刹车中断允许位。

0：禁止 PWM2 刹车中断

1：允许 PWM2 刹车中断

TIE：PWM2触发中断允许位。

0：禁止 PWM2 触发中断

1：允许 PWM2 触发中断

COMIE：PWM2比较中断允许位。

0：禁止 PWM2 比较中断

1：允许 PWM2 比较中断

CC8IE：PWM2捕获比较通道8中断允许位。

0：禁止 PWM2 捕获比较通道 8 中断

1：允许 PWM2 捕获比较通道 8 中断

CC7IE：PWM2捕获比较通道7中断允许位。

0：禁止 PWM2 捕获比较通道 7 中断

1：允许 PWM2 捕获比较通道 7 中断

CC6IE：PWM2捕获比较通道6中断允许位。

0：禁止 PWM2 捕获比较通道 6 中断

1：允许 PWM2 捕获比较通道 6 中断

CC5IE：PWM2捕获比较通道5中断允许位。

0：禁止 PWM2 捕获比较通道 5 中断

1：允许 PWM2 捕获比较通道 5 中断

UIE：PWM2更新中断允许位。

0：禁止 PWM2 更新中断

1：允许 PWM2 更新中断

11.4.2 中断请求寄存器（中断标志位）

定时器控制寄存器

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

TF1：定时器1溢出中断标志。中断服务程序中，硬件自动清零。

TF0：定时器0溢出中断标志。中断服务程序中，硬件自动清零。

IE1：外部中断1中断请求标志。中断服务程序中，硬件自动清零。

IE0：外部中断0中断请求标志。中断服务程序中，硬件自动清零。

中断标志辅助寄存器

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

INT4IF：外部中断4中断请求标志。需要软件清零。

INT3IF：外部中断3中断请求标志。需要软件清零。

INT2IF：外部中断2中断请求标志。需要软件清零。

T4IF：定时器4溢出中断标志。需要软件清零。

T3IF：定时器3溢出中断标志。需要软件清零。

T2IF：定时器2溢出中断标志。需要软件清零。

串口控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0

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

TI：串口1发送完成中断请求标志。需要软件清零。

RI：串口1接收完成中断请求标志。需要软件清零。

S2TI：串口2发送完成中断请求标志。需要软件清零。

S2RI：串口2接收完成中断请求标志。需要软件清零。

S3TI：串口3发送完成中断请求标志。需要软件清零。

S3RI：串口3接收完成中断请求标志。需要软件清零。

S4TI：串口4发送完成中断请求标志。需要软件清零。

S4RI：串口4接收完成中断请求标志。需要软件清零。

电源管理寄存器

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

LVDF：低压检测中断请求标志。需要软件清零。

ADC 控制寄存器

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

ADC_FLAG：ADC转换完成中断请求标志。需要软件清零。

SPI 状态寄存器

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

SPIF：SPI数据传输完成中断请求标志。需要软件清零。

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。

PWM1 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM1_SR1	FEC5H	BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF
PWM1_SR2	FEC6H	-	-	-	CC4OF	CC3OF	CC2OF	CC1OF	-

BIF：PWM1刹车中断请求标志。需要软件清零。

TIF：PWM1触发中断请求标志。需要软件清零。

COMIF：PWM1比较中断请求标志。需要软件清零。

CC4IF：PWM1通道4发生捕获比较中断请求标志。需要软件清零。

CC3IF：PWM1通道3发生捕获比较中断请求标志。需要软件清零。

CC2IF：PWM1通道2发生捕获比较中断请求标志。需要软件清零。

CC1IF：PWM1通道1发生捕获比较中断请求标志。需要软件清零。

TIF：PWM1更新中断请求标志。需要软件清零。

CC4OF：PWM1通道4发生重复捕获中断请求标志。需要软件清零。

CC3OF：PWM1通道3发生重复捕获中断请求标志。需要软件清零。

CC2OF：PWM1通道2发生重复捕获中断请求标志。需要软件清零。

CC1OF：PWM1通道1发生重复捕获中断请求标志。需要软件清零。

PWM2 状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PWM2_SR1	FEE5H	BIF	TIF	COMIF	CC8IF	CC7IF	CC6IF	CC5IF	UIF
PWM2_SR2	FEE6H	-	-	-	CC8OF	CC7OF	CC6OF	CC5OF	-

BIF：PWM2刹车中断请求标志。需要软件清零。

TIF：PWM2触发中断请求标志。需要软件清零。

COMIF：PWM2比较中断请求标志。需要软件清零。

CC8IF：PWM2通道8发生捕获比较中断请求标志。需要软件清零。

CC7IF：PWM2通道7发生捕获比较中断请求标志。需要软件清零。

CC6IF：PWM2通道6发生捕获比较中断请求标志。需要软件清零。

CC5IF：PWM2通道5发生捕获比较中断请求标志。需要软件清零。

TIF：PWM2更新中断请求标志。需要软件清零。

CC8OF：PWM2通道8发生重复捕获中断请求标志。需要软件清零。

CC7OF：PWM2通道7发生重复捕获中断请求标志。需要软件清零。

CC6OF：PWM2通道6发生重复捕获中断请求标志。需要软件清零。

CC5OF：PWM2通道5发生重复捕获中断请求标志。需要软件清零。

11.4.3 中断优先级寄存器

中断优先级控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IP	B8H	-	PLVD	PADC	PS	PT1	PX1	PT0	PX0

IPH	B7H	-	PLVDH	PADCH	PSH	PT1H	PX1H	PT0H	PX0H
IP2	B5H	-	PI2C	PCMP	PX4	PPWM2	PPWM1	PSPI	PS2
IP2H	B6H	-	PI2CH	PCMPH	PX4H	PPWM2H	PPWM1H	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）
11：SPI 中断优先级为 3 级（最高级）

PPWM1H,PPWM1：高级PWM1中断优先级控制位

00：高级 PWM1 中断优先级为 0 级（最低级）
01：高级 PWM1 中断优先级为 1 级（较低级）
10：高级 PWM1 中断优先级为 2 级（较高级）
11：高级 PWM1 中断优先级为 3 级（最高级）

PPWM2H,PPWM2：高级PWM2中断优先级控制位

00：高级 PWM2 中断优先级为 0 级（最低级）
01：高级 PWM2 中断优先级为 1 级（较低级）
10：高级 PWM2 中断优先级为 2 级（较高级）
11：高级 PWM2 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）

11.5 范例程序

11.5.1 INT0 中断（上升沿和下降沿）

汇编代码

```
                ORG     0000H
                LJMP    MAIN
                ORG     0003H
                LJMP    INT0ISR

                ORG     0100H
INT0ISR:
                JB      INT0,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,#3FH

                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
```

C 语言代码

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

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

void INT0_Isr() interrupt 0
{
    if (INT0)                       //判断上升沿和下降沿
    {
        P10 = !P10;                 //测试端口
    }
    else
    {
        P11 = !P11;                 //测试端口
    }
}

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

    while (1);
}
```

11.5.2 INT0 中断（下降沿）

汇编代码

```
                ORG         0000H
                LJMP        MAIN
                ORG         0003H
                LJMP        INT0ISR

                ORG         0100H
INT0ISR:
                CPL         P1.0                ;测试端口
                RETI

MAIN:
                MOV         SP,#3FH

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

                END
```

C 语言代码

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

sbit        P10             =   P1^0;

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

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

    while (1);
}
```

11.5.3 INT1 中断（上升沿和下降沿）

汇编代码

```
                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,#3FH

                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
```

C 语言代码

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

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

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

void main()
{
    IT1 = 0;                    //使能 INT1 上升沿和下降沿中断
    EX1 = 1;                    //使能 INT1 中断
    EA = 1;

    while (1);
}
```

11.5.4 INT1 中断（下降沿）

汇编代码

```
                ORG     0000H
                LJMP    MAIN
                ORG     0013H
                LJMP    INT1ISR

                ORG     0100H
INT1ISR:
                CPL     P1.0                    ;测试端口
                RETI
```

```
MAIN:
        MOV     SP,#3FH

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

        END
```

C 语言代码

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

sbit    P10     =   P1^0;

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

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

    while (1);
}
```

11.5.5 INT2 中断（下降沿）

汇编代码

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

            ORG     0000H
            LJMP    MAIN
            ORG     0053H
            LJMP    INT2ISR

            ORG     0100H
INT2ISR:
            CPL     P1.0            ;测试端口
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     INTCLKO,#EX2    ;使能 INT2 中断
            SETB    EA
            JMP     $
```

```
        END
```

C 语言代码

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

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

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

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

    while (1);
}
```

11.5.6 INT3 中断（下降沿）

汇编代码

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

            ORG     0000H
            LJMP    MAIN
            ORG     005BH
            LJMP    INT3ISR

            ORG     0100H
INT3ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     INTCLKO,#EX3        ;使能 INT3 中断
            SETB    EA
            JMP     $

            END
```

C 语言代码

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

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

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

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

    while (1);
}
```

11.5.7 INT4 中断（下降沿）

汇编代码

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

            ORG     0000H
            LJMP    MAIN
            ORG     0083H
            LJMP    INT4ISR

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

MAIN:
            MOV     SP,#3FH

            MOV     INTCLKO,#EX4        ;使能 INT4 中断
            SETB    EA
            JMP     $

            END
```

C 语言代码

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

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

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

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

    while (1);
}
```

11.5.8 定时器 0 中断

汇编代码

```
;测试工作频率为 11.0592MHz

        ORG     0000H
        LJMP    MAIN
        ORG     000BH
        LJMP    TM0ISR

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

MAIN:
        MOV     SP,#3FH

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

        JMP     $

        END
```

C 语言代码

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

//测试工作频率为 11.0592MHz

sbit    P10     =   P1^0;

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

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

    while (1);
}
```

11.5.9 定时器 1 中断

汇编代码

```
;测试工作频率为11.0592MHz

        ORG     0000H
        LJMP    MAIN
        ORG     001BH
        LJMP    TM1ISR

        ORG     0100H
TM1ISR:
        CPL     P1.0                ;测试端口
        RETI

MAIN:
        MOV     SP,#3FH

        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
```

C 语言代码

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

//测试工作频率为11.0592MHz

sbit    P10     =   P1^0;

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

void main()
{
```

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

    while (1);
}
```

11.5.10 定时器 2 中断

汇编代码

```
;测试工作频率为11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     0063H
            LJMP    TM2ISR

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

MAIN:
            MOV     SP,#3FH

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

            JMP     $

            END
```

C 语言代码

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

//测试工作频率为11.0592MHz

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

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

sbit        P10             =   P1^0;

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

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

    while (1);
}
```

11.5.11 定时器 3 中断

汇编代码

```
;测试工作频率为11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     009BH
            LJMP    TM3ISR

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

MAIN:
            MOV     SP,#3FH

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

            JMP     $
```

```
        END
```

C 语言代码

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

//测试工作频率为11.0592MHz

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

sbit    P10     =   P1^0;

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

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

    while (1);
}
```

11.5.12 定时器 4 中断

汇编代码

```
;测试工作频率为11.0592MHz

T4L         DATA    0D3H
T4H         DATA    0D2H
T4T3M       DATA    0D1H
IE2         DATA    0AFH
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T4IF        EQU     04H

            ORG     0000H
            LJMP    MAIN
            ORG     00A3H
            LJMP    TM4ISR

            ORG     0100H
TM4ISR:
```

```
            CPL     P1.0                ;测试端口
            ANL     AUXINTIF,#NOT T4IF  ;清中断标志
            RETI

MAIN:
            MOV     SP,#3FH

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

            JMP     $

            END
```

C 语言代码

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

//测试工作频率为11.0592MHz

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

sbit    P10     =   P1^0;

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

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

    while (1);
}
```

11.5.13 UART1 中断

汇编代码

```
;测试工作频率为11.0592MHz

T2L         DATA    0D7H
```

```
T2H             DATA    0D6H
AUXR            DATA    8EH

                ORG     0000H
                LJMP    MAIN
                ORG     0023H
                LJMP    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,#3FH

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

                JMP     $

                END
```

C 语言代码

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

//测试工作频率为11.0592MHz

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

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

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

```
            P11 = !P11;                 //测试端口
        }
}

void main()
{
        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);
}
```

11.5.14 UART2 中断

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

                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,#3FH

                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
```

C 语言代码

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

//测试工作频率为11.0592MHz

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

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

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

void main()
{
    S2CON = 0x10;
    T2L = 0xe8;                     //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;                    //启动定时器
    IE2 = ES2;                      //使能串口中断
    EA = 1;
    S2BUF = 0x5a;                   //发送测试数据

    while (1);
}
```

11.5.15 UART3 中断

汇编代码

```
;测试工作频率为11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH
ES3         EQU     08H

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3ISR

            ORG     0100H
UART3ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     A,S3CON
            JNB     ACC.1,CHECKRI
            ANL     S3CON,#NOT 02H      ;清中断标志
            CPL     P1.0                ;测试端口
CHECKRI:
            MOV     A,S3CON
            JNB     ACC.0,ISREXIT
            ANL     S3CON,#NOT 01H      ;清中断标志
            CPL     P1.1                ;测试端口
ISREXIT:
            POP     PSW
            POP     ACC
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     S3CON,#10H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H           ;启动定时器
            MOV     IE2,#ES3            ;使能串口中断
            SETB    EA
            MOV     S3BUF,#5AH          ;发送测试数据

            JMP     $

            END
```

C 语言代码

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

//测试工作频率为11.0592MHz
```

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

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

void UART3_Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;             //清中断标志
        P10 = !P10;                 //测试端口
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;             //清中断标志
        P11 = !P11;                 //测试端口
    }
}

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

    while (1);
}
```

11.5.16 UART4 中断

汇编代码

```
;测试工作频率为11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4ISR

            ORG     0100H
UART4ISR:
```

```
                PUSH    ACC
                PUSH    PSW
                MOV     A,S4CON
                JNB     ACC.1,CHECKRI
                ANL     S4CON,#NOT 02H          ;清中断标志
                CPL     P1.0                    ;测试端口
CHECKRI:
                MOV     A,S4CON
                JNB     ACC.0,ISREXIT
                ANL     S4CON,#NOT 01H          ;清中断标志
                CPL     P1.1                    ;测试端口
ISREXIT:
                POP     PSW
                POP     ACC
                RETI

MAIN:
                MOV     SP,#3FH

                MOV     S4CON,#10H
                MOV     T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     T2H,#0FFH
                MOV     AUXR,#14H               ;启动定时器
                MOV     IE2,#ES4                ;使能串口中断
                SETB    EA
                MOV     S4BUF,#5AH              ;发送测试数据

                JMP     $

                END
```

C 语言代码

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

//测试工作频率为11.0592MHz

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

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

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

```
            P11 = !P11;                 //测试端口
        }
}

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

        while (1);
}
```

11.5.17 ADC中断

汇编代码

```
ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH
EADC        BIT     IE.5

            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,#3FH

            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
```

C 语言代码

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

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

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

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

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

    while (1);
}
```

11.5.18 LVD中断

汇编代码

```
RSTCFG      DATA    0FFH
ENLVR       EQU     40H             ;RSTCFG.6
LVD2V2      EQU     00H             ;LVD@2.2V
LVD2V4      EQU     01H             ;LVD@2.4V
LVD2V7      EQU     02H             ;LVD@2.7V
LVD3V0      EQU     03H             ;LVD@3.0V
ELVD        BIT     IE.6
LVDF        EQU     20H             ;PCON.5

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF  ;清中断标志
            CPL     P1.0            ;测试端口
            RETI

MAIN:
            MOV     SP,#3FH

            ANL     PCON,#NOT LVDF  ;上电需要清中断标志
            MOV     RSTCFG,# LVD3V0 ;设置 LVD 电压为 3.0V
            SETB    ELVD            ;使能 LVD 中断
            SETB    EA
            JMP     $

            END
```

C 语言代码

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

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

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

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

    while (1);
}
```

11.5.19 CMP中断

汇编代码

```
CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR

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

MAIN:
            MOV     SP,#3FH

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

            JMP     $
```

```
        END
```

C 语言代码

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

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

sbit    P10         =   P1^0;

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

void main()
{
    CMPCR2 = 0x00;
    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);
}
```

11.5.20 I2C中断

汇编代码

```
P_SW2       DATA    0BAH

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

            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,#3FH

                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
```

C 语言代码

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

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sbit    P10         =   P1^0;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;
    if (I2CMSST & 0x40)
    {
```

```
            I2CMSST &= ~0x40;               //清中断标志
            P10 = !P10;                     //测试端口
        }
        _pop_(P_SW2);
}

void main()
{
        P_SW2 = 0x80;
        I2CCFG = 0xc0;                      //使能 I2C 主机模式
        I2CMSCR = 0x80;                     //使能 I2C 中断;
        P_SW2 = 0x00;
        EA = 1;

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

        while (1);
}
```

12 定时器/计数器

STC8H 系列单片机内部设置了 5 个 16 位定时器/计数器。5 个 16 位定时器 T0、T1、T2、T3 和 T4 都具有计数方式和定时方式两种工作方式。对定时器/计数器 T0 和 T1，用它们在特殊功能寄存器 TMOD 中相对应的控制位 C/T 来选择 T0 或 T1 为定时器还是计数器。对定时器/计数器 T2，用特殊功能寄存器 AUXR 中的控制位 T2_C/T 来选择 T2 为定时器还是计数器。对定时器/计数器 T3，用特殊功能寄存器 T4T3M 中的控制位 T3_C/T 来选择 T3 为定时器还是计数器。对定时器/计数器 T4，用特殊功能寄存器 T4T3M 中的控制位 T4_C/T 来选择 T4 为定时器还是计数器。定时器/计数器的核心部件是一个加法计数器，其本质是对脉冲进行计数。只是计数脉冲来源不同：如果计数脉冲来自系统时钟，则为定时方式，此时定时器/计数器每 12 个时钟或者每 1 个时钟得到一个计数脉冲，计数值加 1；如果计数脉冲来自单片机外部引脚，则为计数方式，每来一个脉冲加 1。

当定时器/计数器 T0、T1 及 T2 工作在定时模式时，特殊功能寄存器 AUXR 中的 T0x12、T1x12 和 T2x12 分别决定是系统时钟/12 还是系统时钟/1（不分频）后让 T0、T1 和 T2 进行计数。当定时器/计数器 T3 和 T4 工作在定时模式时，特殊功能寄存器 T4T3M 中的 T3x12 和 T4x12 分别决定是系统时钟/12 还是系统时钟/1（不分频）后让 T3 和 T4 进行计数。当定时器/计数器工作在计数模式时，对外部脉冲计数不分频。

定时器/计数器 0 有 4 种工作模式：模式 0（16 位自动重装载模式），模式 1（16 位不可重装载模式），模式 2（8 位自动重装模式），模式 3（不可屏蔽中断的 16 位自动重装载模式）。定时器/计数器 1 除模式 3 外，其他工作模式与定时器/计数器 0 相同。T1 在模式 3 时无效，停止计数。定时器 T2 的工作模式固定为 16 位自动重装载模式。T2 可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。定时器 3、定时器 4 与定时器 T2 一样，它们的工作模式固定为 16 位自动重装载模式。T3/T4 可以当定时器使用，也可以当串口的波特率发生器和可编程时钟输出。

12.1 定时器的相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	定时器 0 低 8 为寄存器	8AH									0000,0000
TL1	定时器 1 低 8 为寄存器	8BH									0000,0000
TH0	定时器 0 高 8 为寄存器	8CH									0000,0000
TH1	定时器 1 高 8 为寄存器	8DH									0000,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	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

12.2 定时器 0/1

定时器 0/1 控制寄存器

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

TF1：T1溢出中断标志。T1被允许计数以后，从初值开始加1计数。当产生溢出时由硬件将TF1位置“1”，并向CPU请求中断，一直保持到CPU响应中断时，才由硬件清“0”（也可由查询软件清“0”）。

TR1：定时器T1的运行控制位。该位由软件置位和清零。当GATE（TMOD.7）=0，TR1=1时就允许T1开始计数，TR1=0时禁止T1计数。当GATE（TMOD.7）=1，TR1=1且INT1输入高电平时，才允许T1计数。

TF0：T0溢出中断标志。T0被允许计数以后，从初值开始加1计数，当产生溢出时，由硬件置“1”TF0，向CPU请求中断，一直保持CPU响应该中断时，才由硬件清0（也可由查询软件清0）。

TR0：定时器T0的运行控制位。该位由软件置位和清零。当GATE（TMOD.3）=0，TR0=1时就允许T0开始计数，TR0=0时禁止T0计数。当GATE（TMOD.3）=1，TR0=1且INT0输入高电平时，才允许T0计数，TR0=0时禁止T0计数。

IE1：外部中断1请求源（INT1/P3.3）标志。IE1=1，外部中断向CPU请求中断，当CPU响应该中断时由硬件清“0”IE1。

IT1：外部中断源1触发控制位。IT1=0，上升沿或下降沿均可触发外部中断1。IT1=1，外部中断1程控为下降沿触发方式。

IE0：外部中断0请求源（INT0/P3.2）标志。IE0=1外部中断0向CPU请求中断，当CPU响应外部中断时，由硬件清“0”IE0（边沿触发方式）。

IT0：外部中断源0触发控制位。IT0=0，上升沿或下降沿均可触发外部中断0。IT0=1，外部中断0程控为下降沿触发方式。

定时器 0/1 模式寄存器

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

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

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

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

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

T1_M1/T1_M0：定时器定时器/计数器1模式选择

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	16位自动重载模式 与模式0相同，产生不可屏蔽中断

定时器 0 计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL0	8AH								
TH0	8CH								

当定时器/计数器0工作在16位模式（模式0、模式1、模式3）时，TL0和TH0组合成为一个16位寄存器，TL0为低字节，TH0为高字节。若为8位模式（模式2）时，TL0和TH0为两个独立的8位寄存器。

定时器 1 计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL1	8BH								
TH1	8DH								

当定时器/计数器1工作在16位模式（模式0、模式1）时，TL1和TH1组合成为一个16位寄存器，TL1为低字节，TH1为高字节。若为8位模式（模式2）时，TL1和TH1为两个独立的8位寄存器。

辅助寄存器 1（AUXR）

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

T0x12：定时器0速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

T1x12：定时器1速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

中断与时钟输出控制寄存器（INTCLKO）

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

T0CLKO：定时器0时钟输出控制

0：关闭时钟输出

1：使能 P3.5 口的是定时器 0 时钟输出功能

当定时器 0 计数发生溢出时，P3.5 口的电平自动发生翻转。

T1CLKO：定时器1时钟输出控制

0：关闭时钟输出

1：使能 P3.4 口的是定时器 1 时钟输出功能

当定时器 1 计数发生溢出时，P3.4 口的电平自动发生翻转。

12.3 定时器 2

辅助寄存器 1（AUXR）

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

TR2：定时器2的运行控制位

0：定时器 2 停止计数

1：定时器 2 开始计数

T2_C/T：控制定时器0用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T2/P1.2外部脉冲进行计数）。

T2x12：定时器2速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

中断与时钟输出控制寄存器（INTCLKO）

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

T2CLKO：定时器2时钟输出控制

0：关闭时钟输出

1：使能 P1.3 口的是定时器 2 时钟输出功能

当定时器 2 计数发生溢出时，P1.3 口的电平自动发生翻转。

定时器 2 计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T2L	D7H								
T2H	D6H								

定时器/计数器2的工作模式固定为16位重载模式，T2L和T2H组合成为一个16位寄存器，T2L为低字节，T2H为高字节。当[T2H,T2L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T2H,T2L]中。

12.4 定时器 3/4

定时器 4/3 控制寄存器（T4T3M）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

TR4：定时器4的运行控制位

0：定时器 4 停止计数

1：定时器 4 开始计数

T4_C/T：控制定时器4用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T4/P0.6外部脉冲进行计数）。

T4x12：定时器4速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

T4CLKO：定时器4时钟输出控制

0：关闭时钟输出

1：使能 P0.7 口的是定时器 4 时钟输出功能

当定时器 4 计数发生溢出时，P0.7 口的电平自动发生翻转。

TR3：定时器3的运行控制位

0：定时器 3 停止计数

1：定时器 3 开始计数

T3_C/T：控制定时器3用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T3/P0.4外部脉冲进行计数）。

T3x12：定时器3速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

T3CLKO：定时器3时钟输出控制

0：关闭时钟输出

1：使能 P0.5 口的是定时器 3 时钟输出功能

当定时器 3 计数发生溢出时，P0.5 口的电平自动发生翻转。

定时器 3 计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T3L	D5H								
T3H	D4H								

定时器/计数器3的工作模式固定为16位重载模式，T3L和T3H组合成为一个16位寄存器，T3L为低字节，T3H为高字节。当[T3H,T3L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T3H,T3L]中。

定时器 4 计数寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4L	D3H								
T4H	D2H								

定时器/计数器4的工作模式固定为16位重载模式，T4L和T4H组合成为一个16位寄存器，T4L为低字节，T4H为高字节。当[T4H,T4L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T4H,T4L]中。

注意事项：

1. **定时器 0 从 P3.5 高速输出、定时器 1 从 P3.4 高速输出、定时器 2 从 P1.3 高速输出、定时器 3 从 P0.5 高速输出、定时器 4 从 P0.7 高速输出，这些 IO 设置为准双向口或推挽输出，脉冲输出正常，均为推挽输出，并且再直接操作 IO 将不影响输出波形。但是如果将 IO 设置为开漏输出（允许内部上拉电阻或外接上拉电阻），则如果对应的 IO 输出高电平，波形无输出（IO 为高阻），而对应的 IO 输出低电平，波形有推挽输出。**

12.5 掉电唤醒定时器

内部掉电唤醒定时器是一个 15 位的计数器（由{WKTCH[6:0],WKTCL[7:0]}组成 15 位）。用于唤醒处于掉电模式的 MCU。

掉电唤醒定时器计数寄存器

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

WKTEN：掉电唤醒定时器的使能控制位

0：停用掉电唤醒定时器

1：启用掉电唤醒定时器

如果 STC8 系列单片机内置掉电唤醒专用定时器被允许（通过软件将 WKTCH 寄存器中的 WKTEN 位置 1），当 MCU 进入掉电模式/停机模式后，掉电唤醒专用定时器开始计数，当计数值与用户所设置的值相等时，掉电唤醒专用定时器将 MCU 唤醒。MCU 唤醒后，程序从上次设置单片机进入掉电模式语句的下一条语句开始往下执行。掉电唤醒之后，可以通过读 WKTCH 和 WKTCL 中的内容获取单片机在掉电模式中的睡眠时间。

这里请注意：用户在寄存器{WKTCH[6:0],WKTCL[7:0]}中写入的值必须比实际计数值少 1。如用户需计数 10 次，则将 9 写入寄存器{WKTCH[6:0],WKTCL[7:0]}中。同样，如果用户需计数 32768 次，则应对{WKTCH[6:0],WKTCL[7:0]}写入 7FFFH（即 32767）。

内部掉电唤醒定时器有自己的内部时钟，其中掉电唤醒定时器计数一次的时间就是由该时钟决定的。内部掉电唤醒定时器的时钟频率约为 32KHz，当然误差较大。用户可以通过读 RAM 区 F8H 和 F9H 的内容（F8H 存放频率的高字节，F9H 存放低字节）来获取内部掉电唤醒专用定时器出厂时所记录的时钟频率。

掉电唤醒专用定时器计数时间的计算公式如下所示：（F_{wt} 为我们从 RAM 区 F8H 和 F9H 获取到的内部掉电唤醒专用定时器的时钟频率）

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

假设 F_{wt}=32KHz，则有：

{WKTCH[6:0],WKTCL[7:0]}	掉电唤醒专用定时器计数时间
0	10^6÷32K×16×(1+0) ≈ 0.5 毫秒
9	10^6÷32K×16×(1+9) ≈ 5 毫秒
99	10^6÷32K×16×(1+99) ≈ 50 毫秒
999	10^6÷32K×16×(1+999) ≈ 0.5 秒
4095	10^6÷32K×16×(1+4095) ≈ 2 秒
32767	10^6÷32K×16×(1+32767) ≈ 16 秒

12.6 范例程序

12.6.1 定时器 0（模式 0－16 位自动重载）

汇编代码

```
;测试工作频率为 11.0592MHz

        ORG     0000H
        LJMP    MAIN
        ORG     000BH
        LJMP    TM0ISR

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

MAIN:
        MOV     SP,#3FH

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

        JMP     $

        END
```

C 语言代码

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

//测试工作频率为 11.0592MHz

sbit    P10     =   P1^0;

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

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

    while (1);
}
```

12.6.2 定时器 0（模式 1－16 位不自动重载）

汇编代码

```
;测试工作频率为11.0592MHz

                ORG     0000H
                LJMP    MAIN
                ORG     000BH
                LJMP    TM0ISR

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

MAIN:
                MOV     SP,#3FH

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

                JMP     $

                END
```

C 语言代码

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

//测试工作频率为11.0592MHz

sbit    P10     =   P1^0;

void TM0_Isr() interrupt 1
{
    TL0 = 0x66;                 //重设定时参数
    TH0 = 0xfc;
    P10 = !P10;                 //测试端口
}

void main()
{
    TMOD = 0x01;                //模式1
    TL0 = 0x66;                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                    //启动定时器
    ET0 = 1;                    //使能定时器中断
    EA = 1;

    while (1);
}
```

12.6.3 定时器 0（模式 2—8 位自动重载）

汇编代码

```
;测试工作频率为11.0592MHz

        ORG     0000H
        LJMP    MAIN
        ORG     000BH
        LJMP    TM0ISR

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

MAIN:
        MOV     SP,#3FH

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

        JMP     $

        END
```

C 语言代码

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

//测试工作频率为11.0592MHz

sbit    P10     =   P1^0;

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

void main()
{
    TMOD = 0x02;                    //模式2
    TL0 = 0xf4;                     //256-11.0592M/12/76K
    TH0 = 0xf4;
    TR0 = 1;                        //启动定时器
    ET0 = 1;                        //使能定时器中断
    EA = 1;

    while (1);
}
```

12.6.4 定时器 0（模式 3－16 位自动重载不可屏蔽中断）

汇编代码

```
;测试工作频率为11.0592MHz

                ORG     0000H
                LJMP    MAIN
                ORG     000BH
                LJMP    TM0ISR

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

MAIN:
                MOV     SP,#3FH

                MOV     TMOD,#03H               ;模式3
                MOV     TL0,#66H                ;65536-11.0592M/12/1000
                MOV     TH0,#0FCH
                SETB    TR0                     ;启动定时器
                SETB    ET0                     ;使能定时器中断
;               SETB    EA                      ;不受EA控制

                JMP     $

                END
```

C 语言代码

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

//测试工作频率为11.0592MHz

sbit        P10             =   P1^0;

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

void main()
{
    TMOD = 0x03;                    //模式3
    TL0 = 0x66;                     //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                        //启动定时器
    ET0 = 1;                        //使能定时器中断
//  EA = 1;                         //不受EA控制

    while (1);
}
```

12.6.5 定时器 0（外部计数－扩展T0 为外部下降沿中断）

汇编代码

```
;测试工作频率为11.0592MHz

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

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

MAIN:
            MOV     SP,#3FH

            MOV     TMOD,#04H           ;外部计数模式
            MOV     TL0,#0FFH
            MOV     TH0,#0FFH
            SETB    TR0                 ;启动定时器
            SETB    ET0                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

C 语言代码

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

//测试工作频率为11.0592MHz

sbit    P10     =   P1^0;

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

void main()
{
    TMOD = 0x04;                //外部计数模式
    TL0 = 0xff;
    TH0 = 0xff;
    TR0 = 1;                    //启动定时器
    ET0 = 1;                    //使能定时器中断
    EA = 1;

    while (1);
}
```

12.6.6 定时器 0（测量脉宽－INT0 高电平宽度）

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA    8EH

            ORG     0000H
            LJMP    MAIN
            ORG     0003H
            LJMP    INT0ISR

            ORG     0100H
INT0ISR:
            MOV     P0,TL0              ;TL0 为测量值低字节
            MOV     P1,TH0              ;TH0 为测量值低高字节
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     AUXR,#80H           ;1T 模式
            MOV     TMOD,#08H           ;使能 GATE,INT0 为 1 时使能计时
            MOV     TL0,#00H
            MOV     TH0,#00H
            JB      INT0,$              ;等待 INT0 为低
            SETB    TR0                 ;启动定时器
            SETB    IT0                 ;使能 INT0 下降沿中断
            SETB    EX0
            SETB    EA

            JMP     $

            END
```

C 语言代码

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

//测试工作频率为 11.0592MHz

sfr     AUXR    =   0x8e;

void INT0_Isr() interrupt 0
{
    P0 = TL0;                   //TL0 为测量值低字节（会有约 10 个时钟的误差）
    P1 = TH0;                   //TH0 为测量值低高字节
}

void main()
{
    AUXR = 0x80;                //1T 模式
    TMOD = 0x08;                //使能 GATE,INT0 为 1 时使能计时
    TL0 = 0x00;
    TH0 = 0x00;
    while (INT0);               //等待 INT0 为低
    TR0 = 1;                    //启动定时器
```

```
    IT0 = 1;                        //使能 INT0 下降沿中断
    EX0 = 1;
    EA = 1;

    while (1);
}
```

12.6.7 定时器 0（时钟分频输出）

汇编代码

```
;测试工作频率为11.0592MHz

INTCLKO     DATA    8FH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

            MOV     TMOD,#00H           ;模式0
            MOV     TL0,#66H            ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0                 ;启动定时器
            MOV     INTCLKO,#01H        ;使能时钟输出

            JMP     $

            END
```

C 语言代码

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

//测试工作频率为11.0592MHz

sfr     INTCLKO     =   0x8f;

void main()
{
    TMOD = 0x00;                    //模式0
    TL0 = 0x66;                     //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                        //启动定时器
    INTCLKO = 0x01;                 //使能时钟输出

    while (1);
}
```

12.6.8 定时器 1（模式 0－16 位自动重载）

汇编代码

```
;测试工作频率为11.0592MHz
```

```
                ORG     0000H
                LJMP    MAIN
                ORG     001BH
                LJMP    TM1ISR

                ORG     0100H
TM1ISR:
                CPL     P1.0                    ;测试端口
                RETI

MAIN:
                MOV     SP,#3FH

                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
```

C 语言代码

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

//测试工作频率为 11.0592MHz

sbit    P10     =   P1^0;

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

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

    while (1);
}
```

12.6.9 定时器 1（模式 1－16 位不自动重载）

汇编代码

```
;测试工作频率为 11.0592MHz

                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,#3FH

            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
```

C 语言代码

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

//测试工作频率为 11.0592MHz

sbit        P10             =   P1^0;

void TM1_Isr() interrupt 3
{
    TL1 = 0x66;                 //重设定时参数
    TH1 = 0xfc;
    P10 = !P10;                 //测试端口
}

void main()
{
    TMOD = 0x10;                //模式 1
    TL1 = 0x66;                 //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                    //启动定时器
    ET1 = 1;                    //使能定时器中断
    EA = 1;

    while (1);
}
```

12.6.10 定时器 1（模式 2—8 位自动重载）

汇编代码

```
;测试工作频率为 11.0592MHz
```

```
                ORG     0000H
                LJMP    MAIN
                ORG     001BH
                LJMP    TM1ISR

                ORG     0100H
TM1ISR:
                CPL     P1.0                    ;测试端口
                RETI

MAIN:
                MOV     SP,#3FH

                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
```

C 语言代码

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

//测试工作频率为11.0592MHz

sbit    P10     =   P1^0;

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

void main()
{
    TMOD = 0x20;                //模式2
    TL1 = 0xf4;                 //256-11.0592M/12/76K
    TH1 = 0xf4;
    TR1 = 1;                    //启动定时器
    ET1 = 1;                    //使能定时器中断
    EA = 1;

    while (1);
}
```

12.6.11 定时器 1（外部计数—扩展T1 为外部下降沿中断）

汇编代码

```
;测试工作频率为11.0592MHz

                ORG     0000H
                LJMP    MAIN
```

```
                ORG         001BH
                LJMP        TM1ISR

                ORG         0100H
TM1ISR:
                CPL         P1.0                ;测试端口
                RETI

MAIN:
                MOV         SP,#3FH

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

                JMP         $

                END
```

C 语言代码

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

//测试工作频率为11.0592MHz

sbit        P10             =   P1^0;

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

void main()
{
    TMOD = 0x40;                //外部计数模式
    TL1 = 0xff;
    TH1 = 0xff;
    TR1 = 1;                    //启动定时器
    ET1 = 1;                    //使能定时器中断
    EA = 1;

    while (1);
}
```

12.6.12 定时器 1（测量脉宽－INT1 高电平宽度）

汇编代码

```
;测试工作频率为11.0592MHz

AUXR            DATA        8EH

                ORG         0000H
                LJMP        MAIN
```

```
                ORG     0013H
                LJMP    INT1ISR

                ORG     0100H
INT1ISR:
                MOV     P0,TL1                  ;TL1 为测量值低字节
                MOV     P1,TH1                  ;TH1 为测量值低高字节
                RETI

MAIN:
                MOV     SP,#3FH

                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
```

C 语言代码

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

//测试工作频率为 11.0592MHz

sfr     AUXR    =   0x8e;

void INT1_Isr() interrupt 2
{
    P0 = TL1;                   //TL1 为测量值低字节（会有约 11 个时钟的误差）
    P1 = TH1;                   //TH1 为测量值低高字节
}

void main()
{
    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);
}
```

12.6.13 定时器 1（时钟分频输出）

汇编代码

```
;测试工作频率为 11.0592MHz

INTCLKO     DATA    8FH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

            MOV     TMOD,#00H           ;模式 0
            MOV     TL1,#66H            ;65536-11.0592M/12/1000
            MOV     TH1,#0FCH
            SETB    TR1                 ;启动定时器
            MOV     INTCLKO,#02H        ;使能时钟输出

            JMP     $

            END
```

C 语言代码

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

//测试工作频率为 11.0592MHz

sfr     INTCLKO     =   0x8f;

void main()
{
    TMOD = 0x00;                    //模式 0
    TL1 = 0x66;                     //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                        //启动定时器
    INTCLKO = 0x02;                 //使能时钟输出

    while (1);
}
```

12.6.14 定时器 1（模式 0）做串口 1 波特率发生器

汇编代码

```
AUXR        DATA    8EH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

            ORG     0000H
            LJMP    MAIN
```

```
                ORG     0023H
                LJMP    UART_ISR

                ORG     0100H

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

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

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#00H
                MOV     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,#3FH

                LCALL   UART_INIT
```

```
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

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

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

                END
```

C 语言代码

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

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

sfr         AUXR        =   0x8e;

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

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
```

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

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

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

void main()
{
    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

12.6.15 定时器 1（模式 2）做串口 1 波特率发生器

汇编代码

```
AUXR        DATA    8EH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
```

```
            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,#3FH

            LCALL   UART_INIT
            SETB    ES
            SETB    EA

            MOV     DPTR,#STRING
            LCALL   UART_SENDSTR
```

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

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

                END
```

C 语言代码

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

#define    FOSC            11059200UL
#define    BRT             (256 - FOSC / 115200 / 32)

sfr        AUXR        =   0x8e;

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

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x20;
    TL1 = BRT;
    TH1 = BRT;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
```

```
}

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

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

void main()
{
	UartInit();
	ES = 1;
	EA = 1;
	UartSendStr("Uart Test !\r\n");

	while (1)
	{
		if (rptr != wptr)
		{
			UartSend(buffer[rptr++]);
			rptr &= 0x0f;
		}
	}
}
```

12.6.16 定时器 2（16 位自动重载）

汇编代码

```
;测试工作频率为11.0592MHz

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

		ORG	0000H
		LJMP	MAIN
		ORG	0063H
		LJMP	TM2ISR

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

```
MAIN:
        MOV     SP,#3FH

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

        JMP     $

        END
```

C 语言代码

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

//测试工作频率为11.0592MHz

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2         0x04
sfr     AUXINTIF=   0xef;
#define T2IF        0x01

sbit    P10     =   P1^0;

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

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

    while (1);
}
```

12.6.17 定时器 2（外部计数—扩展T2 为外部下降沿中断）

汇编代码

```
;测试工作频率为11.0592MHz

T2L     DATA    0D7H
T2H     DATA    0D6H
AUXR    DATA    8EH
IE2     DATA    0AFH
ET2     EQU     04H
```

```
AUXINTIF    DATA    0EFH
T2IF        EQU     01H

            ORG     0000H
            LJMP    MAIN
            ORG     0063H
            LJMP    TM2ISR

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

MAIN:
            MOV     SP,#3FH

            MOV     T2L,#0FFH
            MOV     T2H,#0FFH
            MOV     AUXR,#18H           ;设置外部计数模式并启动定时器
            MOV     IE2,#ET2            ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

C 语言代码

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

//测试工作频率为11.0592MHz

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2         0x04
sfr     AUXINTIF    =   0xef;
#define T2IF        0x01

sbit    P10     =   P1^0;

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

void main()
{
    T2L = 0xff;
    T2H = 0xff;
    AUXR = 0x18;                //设置外部计数模式并启动定时器
    IE2 = ET2;                  //使能定时器中断
    EA = 1;

    while (1);
```

```
}
```

12.6.18 定时器 2（时钟分频输出）

汇编代码

```
;测试工作频率为11.0592MHz

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

                ORG     0000H
                LJMP    MAIN

                ORG     0100H
MAIN:
                MOV     SP,#3FH

                MOV     T2L,#66H                ;65536-11.0592M/12/1000
                MOV     T2H,#0FCH
                MOV     AUXR,#10H               ;启动定时器
                MOV     INTCLKO,#04H            ;使能时钟输出

                JMP     $

                END
```

C 语言代码

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

//测试工作频率为11.0592MHz

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     INTCLKO =   0x8f;

void main()
{
    T2L = 0x66;                 //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                //启动定时器
    INTCLKO = 0x04;             //使能时钟输出

    while (1);
}
```

12.6.19 定时器 2 做串口 1 波特率发生器

汇编代码

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

```
BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

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

UART_INIT:
            MOV     SCON,#50H
            MOV     T2L,#0E8H           ;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,#3FH

                LCALL       UART_INIT
                SETB        ES
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART_SENDSTR

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

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

                END
```

C 语言代码

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

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

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

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

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
```

```
    }
}

void UartInit()
{
    SCON = 0x50;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

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

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

void main()
{
    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

12.6.20 定时器 2 做串口 2 波特率发生器

汇编代码

```
AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S2CON       DATA    9AH
S2BUF       DATA    9BH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
```

```
RPTR        DATA    22H
BUFFER      DATA    23H             ;16 bytes

            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,#3FH

                LCALL   UART2_INIT
                MOV     IE2,#01H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART2_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART2_SEND
                INC     RPTR
                JMP     LOOP

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

                END
```

C 语言代码

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

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

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

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

void Uart2Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
```

```
            wptr &= 0x0f;
        }
}

void Uart2Init()
{
        S2CON = 0x10;
        T2L = BRT;
        T2H = BRT >> 8;
        AUXR = 0x14;
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}

void Uart2Send(char dat)
{
        while (busy);
        busy = 1;
        S2BUF = dat;
}

void Uart2SendStr(char *p)
{
        while (*p)
        {
            Uart2Send(*p++);
        }
}

void main()
{
        Uart2Init();
        IE2 = 0x01;
        EA = 1;
        Uart2SendStr("Uart Test !\r\n");

        while (1)
        {
            if (rptr != wptr)
            {
                Uart2Send(buffer[rptr++]);
                rptr &= 0x0f;
            }
        }
}
```

12.6.21 定时器 2 做串口 3 波特率发生器

汇编代码

```
AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

BUSY        BIT     20H.0
```

```
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR

            ORG     0100H

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

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

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

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

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

```
MAIN:
                MOV     SP,#3FH

                LCALL   UART3_INIT
                MOV     IE2,#08H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART3_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART3_SEND
                INC     RPTR
                JMP     LOOP

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

                END
```

C 语言代码

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

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

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

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

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
```

```
            buffer[wptr++] = S3BUF;
            wptr &= 0x0f;
        }
}

void Uart3Init()
{
        S3CON = 0x10;
        T2L = BRT;
        T2H = BRT >> 8;
        AUXR = 0x14;
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}

void Uart3Send(char dat)
{
        while (busy);
        busy = 1;
        S3BUF = dat;
}

void Uart3SendStr(char *p)
{
        while (*p)
        {
            Uart3Send(*p++);
        }
}

void main()
{
        Uart3Init();
        IE2 = 0x08;
        EA = 1;
        Uart3SendStr("Uart Test !\r\n");

        while (1)
        {
            if (rptr != wptr)
            {
                Uart3Send(buffer[rptr++]);
                rptr &= 0x0f;
            }
        }
}
```

12.6.22 定时器 2 做串口 4 波特率发生器

汇编代码

```
AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S4CON       DATA    84H
S4BUF       DATA    085H
IE2         DATA    0AFH
```

```
BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

            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,#3FH

                LCALL       UART4_INIT
                MOV         IE2,#10H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART4_SENDSTR

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

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

                END
```

C 语言代码

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

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

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

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

void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
```

```
            S4CON &= ~0x01;
            buffer[wptr++] = S4BUF;
            wptr &= 0x0f;
        }
}

void Uart4Init()
{
        S4CON = 0x10;
        T2L = BRT;
        T2H = BRT >> 8;
        AUXR = 0x14;
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}

void Uart4Send(char dat)
{
        while (busy);
        busy = 1;
        S4BUF = dat;
}

void Uart4SendStr(char *p)
{
        while (*p)
        {
            Uart4Send(*p++);
        }
}

void main()
{
        Uart4Init();
        IE2 = 0x10;
        EA = 1;
        Uart4SendStr("Uart Test !\r\n");

        while (1)
        {
            if (rptr != wptr)
            {
                Uart4Send(buffer[rptr++]);
                rptr &= 0x0f;
            }
        }
}
```

12.6.23 定时器 3（16 位自动重载）

汇编代码

```
;测试工作频率为11.0592MHz

T3L             DATA    0D5H
T3H             DATA    0D4H
T4T3M           DATA    0D1H
IE2             DATA    0AFH
```

```
ET3             EQU         20H
AUXINTIF        DATA        0EFH
T3IF            EQU         02H

                ORG         0000H
                LJMP        MAIN
                ORG         009BH
                LJMP        TM3ISR

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

MAIN:
                MOV         SP,#3FH

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

                JMP         $

                END
```

C 语言代码

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

//测试工作频率为11.0592MHz

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

sbit        P10             =   P1^0;

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

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

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

12.6.24 定时器 3（外部计数—扩展T3 为外部下降沿中断）

汇编代码

```
;测试工作频率为11.0592MHz

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

                ORG     0000H
                LJMP    MAIN
                ORG     009BH
                LJMP    TM3ISR

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

MAIN:
                MOV     SP,#3FH

                MOV     T3L,#0FFH
                MOV     T3H,#0FFH
                MOV     T4T3M,#0CH          ;设置外部计数模式并启动定时器
                MOV     IE2,#ET3            ;使能定时器中断
                SETB    EA

                JMP     $

                END
```

C 语言代码

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

//测试工作频率为11.0592MHz

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

sbit    P10         =   P1^0;
```

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

void main()
{
    T3L = 0xff;
    T3H = 0xff;
    T4T3M = 0x0c;               //设置外部计数模式并启动定时器
    IE2 = ET3;                  //使能定时器中断
    EA = 1;

    while (1);
}
```

12.6.25 定时器 3（时钟分频输出）

汇编代码

```
;测试工作频率为11.0592MHz

T3L         DATA    0D5H
T3H         DATA    0D4H
T4T3M       DATA    0D1H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

            MOV     T3L,#66H            ;65536-11.0592M/12/1000
            MOV     T3H,#0FCH
            MOV     T4T3M,#09H          ;使能时钟输出并启动定时器

            JMP     $

            END
```

C 语言代码

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

//测试工作频率为11.0592MHz

sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T4T3M   =   0xd1;

void main()
{
    T3L = 0x66;                 //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x09;               //使能时钟输出并启动定时器
```

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

12.6.26 定时器 3 做串口 3 波特率发生器

汇编代码

```
T4T3M       DATA    0D1H
T3H         DATA    0D4H
T3L         DATA    0D5H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR

            ORG     0100H

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

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

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

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

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

MAIN:
                MOV     SP,#3FH

                LCALL   UART3_INIT
                MOV     IE2,#08H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART3_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART3_SEND
                INC     RPTR
                JMP     LOOP

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

                END
```

C 语言代码

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

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

sfr       T4T3M     =     0xd1;
sfr       T3H       =     0xd4;
sfr       T3L       =     0xd5;
sfr       S3CON     =     0xac;
sfr       S3BUF     =     0xad;
sfr       IE2       =     0xaf;
```

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

void Uart3Isr() interrupt 17
{
	if (S3CON & 0x02)
	{
		S3CON &= ~0x02;
		busy = 0;
	}
	if (S3CON & 0x01)
	{
		S3CON &= ~0x01;
		buffer[wptr++] = S3BUF;
		wptr &= 0x0f;
	}
}

void Uart3Init()
{
	S3CON = 0x50;
	T3L = BRT;
	T3H = BRT >> 8;
	T4T3M = 0x0a;
	wptr = 0x00;
	rptr = 0x00;
	busy = 0;
}

void Uart3Send(char dat)
{
	while (busy);
	busy = 1;
	S3BUF = dat;
}

void Uart3SendStr(char *p)
{
	while (*p)
	{
		Uart3Send(*p++);
	}
}

void main()
{
	Uart3Init();
	IE2 = 0x08;
	EA = 1;
	Uart3SendStr("Uart Test !\r\n");

	while (1)
	{
		if (rptr != wptr)
		{
			Uart3Send(buffer[rptr++]);
```

```
                rptr &= 0x0f;
            }
        }
}
```

12.6.27 定时器 4（16 位自动重载）

汇编代码

```
;测试工作频率为 11.0592MHz

T4L         DATA    0D3H
T4H         DATA    0D2H
T4T3M       DATA    0D1H
IE2         DATA    0AFH
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T4IF        EQU     04H

            ORG     0000H
            LJMP    MAIN
            ORG     00A3H
            LJMP    TM4ISR

            ORG     0100H
TM4ISR:
            CPL     P1.0                ;测试端口
            ANL     AUXINTIF,#NOT T4IF  ;清中断标志
            RETI

MAIN:
            MOV     SP,#3FH

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

            JMP     $

            END
```

C 语言代码

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

//测试工作频率为 11.0592MHz

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

```
sbit        P10             =   P1^0;

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

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

    while (1);
}
```

12.6.28 定时器 4（外部计数—扩展T4 为外部下降沿中断）

汇编代码

```
;测试工作频率为 11.0592MHz

T4L         DATA    0D3H
T4H         DATA    0D2H
T4T3M       DATA    0D1H
IE2         DATA    0AFH
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T4IF        EQU     04H

            ORG     0000H
            LJMP    MAIN
            ORG     00A3H
            LJMP    TM4ISR

            ORG     0100H
TM4ISR:
            CPL     P1.0                ;测试端口
            ANL     AUXINTIF,#NOT T4IF  ;清中断标志
            RETI

MAIN:
            MOV     SP,#3FH

            MOV     T4L,#0FFH
            MOV     T4H,#0FFH
            MOV     T4T3M,#0C0H         ;设置外部计数模式并启动定时器
            MOV     IE2,#ET4            ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

C 语言代码

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

//测试工作频率为11.0592MHz

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

sbit        P10             =   P1^0;

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

void main()
{
    T4L = 0xff;
    T4H = 0xff;
    T4T3M = 0xc0;                   //设置外部计数模式并启动定时器
    IE2 = ET4;                      //使能定时器中断
    EA = 1;

    while (1);
}
```

12.6.29 定时器 4（时钟分频输出）

汇编代码

```
;测试工作频率为11.0592MHz

T4L             DATA    0D3H
T4H             DATA    0D2H
T4T3M           DATA    0D1H

                ORG     0000H
                LJMP    MAIN

                ORG     0100H
MAIN:
                MOV     SP,#3FH

                MOV     T4L,#66H            ;65536-11.0592M/12/1000
                MOV     T4H,#0FCH
                MOV     T4T3M,#90H          ;使能时钟输出并启动定时器

                JMP     $

                END
```

C 语言代码

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

//测试工作频率为11.0592MHz

sfr         T4L         =   0xd3;
sfr         T4H         =   0xd2;
sfr         T4T3M       =   0xd1;

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

    while (1);
}
```

12.6.30 定时器 4 做串口 4 波特率发生器

汇编代码

```
T4T3M       DATA    0D1H
T4H         DATA    0D2H
T4L         DATA    0D3H
S4CON       DATA    84H
S4BUF       DATA    085H
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

            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,#3FH

                LCALL   UART4_INIT
                MOV     IE2,#10H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART4_SENDSTR

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

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

```
END
```

C 语言代码

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

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

sfr       T4T3M     =     0xd1;
sfr       T4H       =     0xd2;
sfr       T4L       =     0xd3;
sfr       S4CON     =     0x84;
sfr       S4BUF     =     0x85;
sfr       IE2       =     0xaf;

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

void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
```

```
            Uart4Send(*p++);
        }
}

void main()
{
        Uart4Init();
        IE2 = 0x10;
        EA = 1;
        Uart4SendStr("Uart Test !\r\n");

        while (1)
        {
            if (rptr != wptr)
            {
                Uart4Send(buffer[rptr++]);
                rptr &= 0x0f;
            }
        }
}
```

13 串口通信

STC8 系列单片机具有 4 个全双工异步串行通信接口(串口 1、串口 2、串口 3 和串口 4)。每个串行口由 2 个数据缓冲器、一个移位寄存器、一个串行控制寄存器和一个波特率发生器等组成。每个串行口的数据缓冲器由 2 个互相独立的接收、发送缓冲器构成，可以同时发送和接收数据。

STC8 系列单片机的串口 1 有 4 种工作方式，其中两种方式的波特率是可变的，另两种是固定的，以供不同应用场合选用。串口 2/串口 3/串口 4 都只有两种工作方式，这两种方式的波特率都是可变的。用户可用软件设置不同的波特率和选择不同的工作方式。主机可通过查询或中断方式对接收/发送进行程序处理，使用十分灵活。

串口 1、串口 2、串口 3、串口 4 的通讯口均可以通过功能管脚的切换功能切换到多组端口，从而可以将一个通讯口分时复用为多个通讯口。

13.1 串口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	串口 1 数据寄存器	99H									0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S2BUF	串口 2 数据寄存器	9BH									0000,0000
S3CON	串口 3 控制寄存器	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	0000,0000
S3BUF	串口 3 数据寄存器	ADH									0000,0000
S4CON	串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	串口 4 数据寄存器	85H									0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
SADDR	串口 1 从机地址寄存器	A9H									0000,0000
SADEN	串口 1 从机地址屏蔽寄存器	B9H									0000,0000

13.2 串口 1

串口 1 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

SM0/FE：当PCON寄存器中的SMOD0位为1时，该位为帧错误检测标志位。当UART在接收过程中检测到一个无效停止位时，通过UART接收器将该位置1，必须由软件清零。当PCON寄存器中的SMOD0位为0时，该位和SM1一起指定串口1的通信工作模式，如下表所示：

SM0	SM1	串口1工作模式	功能说明
0	0	模式0	同步移位串行方式
0	1	模式1	可变波特率8位数据方式
1	0	模式2	固定波特率9位数据方式
1	1	模式3	可变波特率9位数据方式

SM2：允许模式 2 或模式 3 多机通信控制位。当串口 1 使用模式 2 或模式 3 时，如果 SM2 位为 1 且 REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 RB8）来筛选地址帧，若 RB8=1，说明该帧是地址帧，地址信息可以进入 SBUF，并使 RI 为 1，进而在中断服务程序中再进行地址号比较；若 RB8=0，说明该帧不是地址帧，应丢掉且保持 RI=0。在模式 2 或模式 3 中，如果 SM2 位为 0 且 REN 位为 1，接收收机处于地址帧筛选被禁止状态，不论收到的 RB8 为 0 或 1，均可使接收到的信息进入 SBUF，并使 RI=1，此时 RB8 通常为校验位。模式 1 和模式 0 为非多机通信方式，在这两种方式时，SM2 应设置为 0。

REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

TB8：当串口 1 使用模式 2 或模式 3 时，TB8 为要发送的第 9 位数据，按需要由软件置位或清 0。在模式 0 和模式 1 中，该位不用。

RB8：当串口 1 使用模式 2 或模式 3 时，RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 和模式 1 中，该位不用。

TI：串口 1 发送中断请求标志位。在模式 0 中，当串口发送数据第 8 位结束时，由硬件自动将 TI 置 1，向主机请求中断，响应中断后 TI 必须用软件清零。在其他模式中，则在停止位开始发送时由硬件自动将 TI 置 1，向 CPU 发请求中断，响应中断后 TI 必须用软件清零。

RI：串口 1 接收中断请求标志位。在模式 0 中，当串口接收第 8 位数据结束时，由硬件自动将 RI 置 1，向主机请求中断，响应中断后 RI 必须用软件清零。在其他模式中，串行接收到停止位的中间时刻由硬件自动将 RI 置 1，向 CPU 发中断申请，响应中断后 RI 必须由软件清零。

串口 1 数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

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

电源管理寄存器

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

SMOD：串口 1 波特率控制位

0：串口 1 的各个模式的波特率都不加倍

1：串口 1 模式 1、模式 2、模式 3 的波特率加倍

SMOD0：帧错误检测控制位

0：无帧错检测功能

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

辅助寄存器 1

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

UART_M0x6：串口 1 模式 0 的通讯速度控制

0：串口 1 模式 0 的波特率不加倍，固定为 Fosc/12

1：串口 1 模式 0 的波特率 6 倍速，即固定为 Fosc/12*6 ＝ Fosc/2

S1ST2：串口 1 波特率发射器选择位

0：选择定时器 1 作为波特率发射器

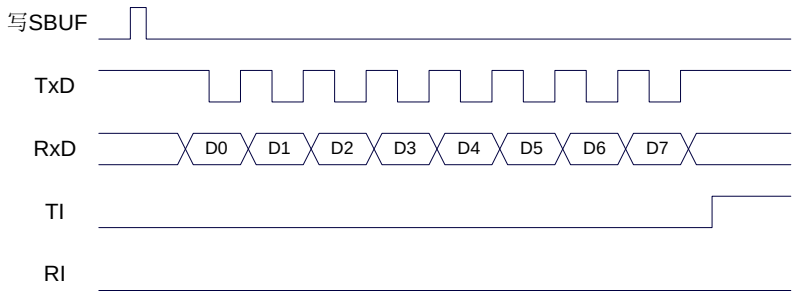
1：选择定时器 2 作为波特率发射器

13.2.1 串口 1 模式 0

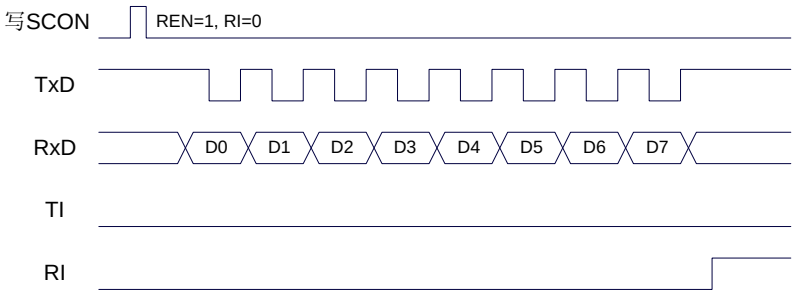
当串口 1 选择工作模式为模式 0 时，串行通信接口工作在同步移位寄存器模式，当串行口模式 0 的通信速度设置位 UART_M0x6 为 0 时，其波特率固定为系统时钟时钟的 12 分频（SYSclk/12）；当设置 UART_M0x6 为 1 时，其波特率固定为系统时钟频率的 2 分频（SYSclk/2）。RxD 为串行通讯的数据口，TxD 为同步移位脉冲输出脚，发送、接收的是 8 位数据，低位在先。

模式 0 的发送过程：当主机执行将数据写入发送缓冲器 SBUF 指令时启动发送，串行口即将 8 位数据以 SYSclk/12 或 SYSclk/2（由 UART_M0x6 确定是 12 分频还是 2 分频）的波特率从 RxD 管脚输出(从低位到高位)，发送完中断标志 TI 置 1，TxD 管脚输出同步移位脉冲信号。当写信号有效后，相隔一个时钟，发送控制端 SEND 有效(高电平)，允许 RxD 发送数据，同时允许 TxD 输出同步移位脉冲。一帧(8 位)数据发送完毕时，各控制端均恢复原状态，只有 TI 保持高电平，呈中断申请状态。在再次发送数据前，必须用软件将 TI 清 0。

模式 0 的接收过程：首先将接收中断请求标志 RI 清零并置位允许接收控制位 REN 时启动模式 0 接收过程。启动接收过程后，RxD 为串行数据输入端，TxD 为同步脉冲输出端。串行接收的波特率为 SYSclk/12 或 SYSclk/2（由 UART_M0x6 确定是 12 分频还是 2 分频）。当接收完成一帧数据（8 位）后，控制信号复位，中断标志 RI 被置 1，呈中断申请状态。当再次接收时，必须通过软件将 RI 清 0



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 $= \frac{\text{SYSclk}}{12}$
1	波特率 $= \frac{\text{SYSclk}}{2}$

13.2.2 串口 1 模式 1

当软件设置 SCON 的 SM0、SM1 为“01”时，串行口 1 则以模式 1 进行工作。此模式为 8 位 UART 格式，一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，即可根据需要进行设置波特率。TxD 为数据发送口，RxD 为数据接收口，串行口全双工接受/发送。

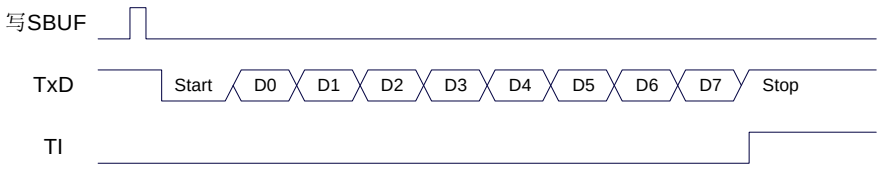
模式 1 的发送过程：串行通信模式发送时，数据由串行发送端 TxD 输出。当主机执行一条写 SBUF 的指令就启动串行通信的发送，写“SBUF”信号还把“1”装入发送移位寄存器的第 9 位，并通知 TX 控制单元开始发送。移位寄存器将数据不断右移送 TxD 端口发送，在数据的左边不断移入“0”作补充。当数据的最高位移到移位寄存器的输出位置，紧跟其后的是第 9 位“1”，在它的左边各位全为“0”，这个状态条件，使 TX 控制单元作最后一次移位输出，然后使允许发送信号“SEND”失效，完成一帧信息的发送，并置位中断请求位 TI，即 TI=1，向主机请求中断处理。

模式 1 的接收过程：当软件置位接收允许标志位 REN，即 REN=1 时，接收器便对 RxD 端口的信号进行检测，当检测到 RxD 端口发送从“1”→“0”的下降沿跳变时就启动接收器准备接收数据，并立即复位波特率发生器的接收计数器，将 1FFH 装入移位寄存器。接收的数据从接收移位寄存器的右边移入，已装入的 1FFH 向左边移出，当起始位"0"移到移位寄存器的最左边时，使 RX 控制器作最后一次移位，完成一帧的接收。若同时满足以下两个条件：

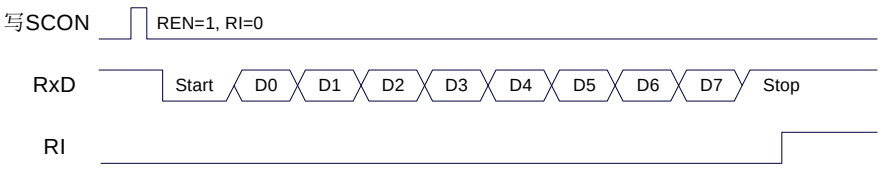
• RI=0；

• SM2=0 或接收到的停止位为 1。

则接收到的数据有效，实现装载入 SBUF，停止位进入 RB8，RI 标志位被置 1，向主机请求中断，若上述两条件不能同时满足，则接收到的数据作废并丢失，无论条件满足与否，接收器重又检测 RxD 端口上的"1"→"0"的跳变，继续下一帧的接收。接收有效，在响应中断后，RI 标志位必须由软件清 0。通常情况下，串行通信工作于模式 1 时，SM2 设置为"0"。



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 = $65536 - \frac{\text{SYSclk}}{4\times\text{波特率}}$
	12T	定时器2重载值 = $65536 - \frac{\text{SYSclk}}{12\times4\times\text{波特率}}$
定时器1模式0	1T	定时器1重载值 = $65536 - \frac{\text{SYSclk}}{4\times\text{波特率}}$
	12T	定时器1重载值 = $65536 - \frac{\text{SYSclk}}{12\times4\times\text{波特率}}$
定时器1模式2	1T	定时器1重载值 = $256 - \frac{2^{\text{SMOD}}\times\text{SYSclk}}{32\times\text{波特率}}$
	12T	定时器1重载值 = $256 - \frac{2^{\text{SMOD}}\times\text{SYSclk}}{12\times32\times\text{波特率}}$

下面为常用频率与常用波特率所对应定时器的重载值

频率 (MHz)	波特率	定时器 2		定时器 1 模式 0		定时器 1 模式 2			
						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

13.2.3 串口 1 模式 2

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

模式 2 的波特率固定为系统时钟的 64 分频或 32 分频（取决于 PCON 中 SMOD 的值）

串口 1 模式 2 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

SMOD	波特率计算公式
0	波特率 $= \frac{\text{SYSclk}}{64}$
1	波特率 $= \frac{\text{SYSclk}}{32}$

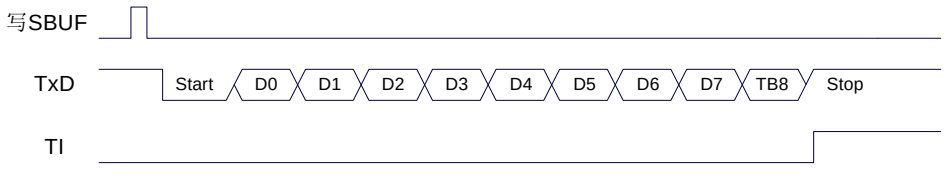
模式 2 和模式 1 相比，除波特率发生源略有不同，发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收/发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

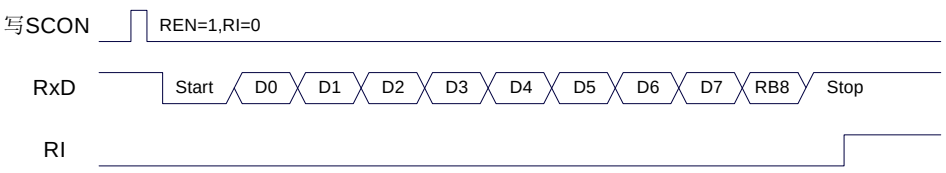
- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，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）

13.2.4 串口 1 模式 3

当 SM0、SM1 两位为 11 时，串行口 1 工作在模式 3。串行通信模式 3 为 9 位数据异步通信 UART

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

模式 3 和模式 1 相比，除发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收‘发送操作过程及时序也基本相同。

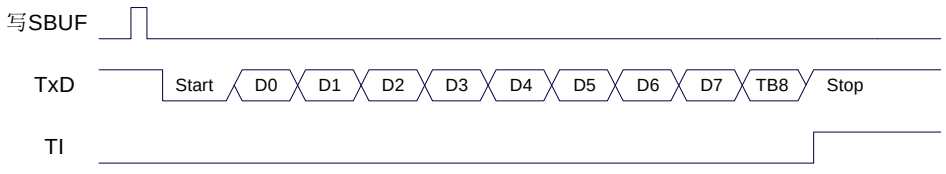
当接收器接收完一帧信息后必须同时满足下列条件：

• RI=0

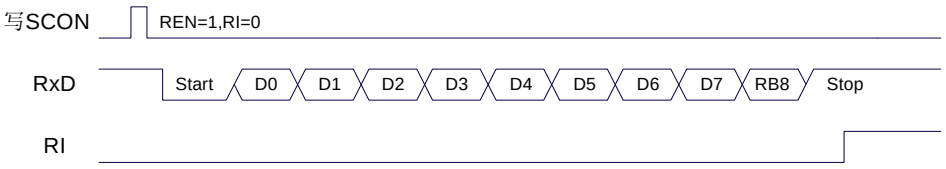
• SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 RxD 输入端口的跳变信息，接收下一帧的输入信息。在模式 3 中，接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信协议的约定，为多机通信提供了方便。



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。

13.2.5 自动地址识别

串口 1 从机地址控制寄存器

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

SADDR：从机地址寄存器

SADEN：从机地址屏蔽位寄存器

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

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

要使用串口的自动地址识别功能，首先需要将参与通讯的 MCU 的串口通讯模式设置为模式 2 或者模式 3（通常都选择波特率可变的模式 3，因为模式 2 的波特率是固定的，不便于调节），并开启从机的 SCON 的 SM2 位。对于串口模式 2 或者模式 3 的 9 位数据位中，第 9 位数据（存放在 RB8 中）为地址/数据的标志位，当第 9 位数据为 1 时，表示前面的 8 位数据（存放在 SBUF 中）为地址信息。当 SM2 被设置为 1 时，从机 MCU 会自动过滤掉非地址数据（第 9 位为 0 的数据），而对 SBUF 中的地址数据（第 9 位为 1 的数据）自动与 SADDR 和 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。

13.3 串口 2

串口 2 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2SM0：指定串口2的通信工作模式，如下表所示：

S2SM0	串口2工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S2SM2：允许串口 2 在模式 1 时允许多机通信控制位。在模式 1 时，如果 S2SM2 位为 1 且 S2REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 S2RB8）来筛选地址帧：若 S2RB8=1，说明该帧是地址帧，地址信息可以进入 S2BUF，并使 S2RI 为 1，进而在中断服务程序中再进行地址号比较；若 S2RB8=0，说明该帧不是地址帧，应丢掉且保持 S2RI=0。在模式 1 中，如果 S2SM2 位为 0 且 S2REN 位为 1，接收收机处于地址帧筛选被禁止状态。不论收到的 S2RB8 为 0 或 1，均可使接收到的信息进入 S2BUF，并使 S2RI=1，此时 S2RB8 通常为校验位。模式 0 为非多机通信方式，在这种方式时，要设置 S2SM2 应为 0。

S2REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

S2TB8：当串口 2 使用模式 1 时，S2TB8 为要发送的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位，按需要由软件置位或清 0。在模式 0 中，该位不用。

S2RB8：当串口 2 使用模式 1 时，S2RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 中，该位不用。

S2TI：串口 2 发送中断请求标志位。在停止位开始发送时由硬件自动将 S2TI 置 1，向 CPU 发请求中断，响应中断后 S2TI 必须用软件清零。

S2RI：串口 2 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S2RI 置 1，向 CPU 发中断申请，响应中断后 S2RI 必须由软件清零。

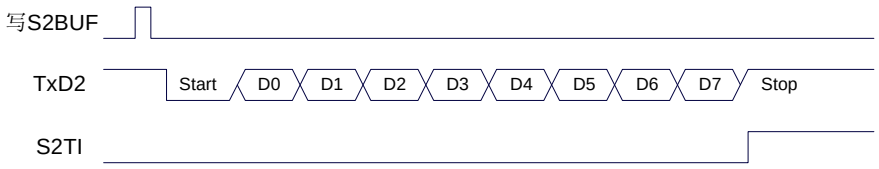
串口 2 数据寄存器

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

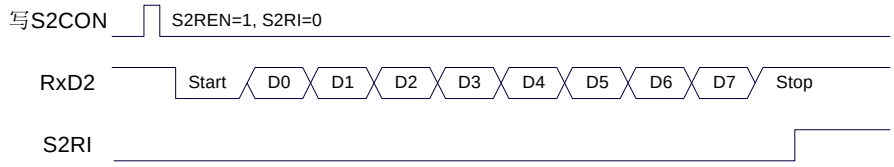
S2BUF：串口 1 数据接收/发送缓冲区。S2BUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 S2BUF 进行读操作，实际是读取串口接收缓冲区，对 S2BUF 进行写操作则是触发串口开始发送数据。

13.3.1 串口 2 模式 0

串行口 2 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD2 为数据发送口，RxD2 为数据接收口，串行口全双工接受/发送。



发送数据（串口2模式0）



接收数据（串口2模式0）

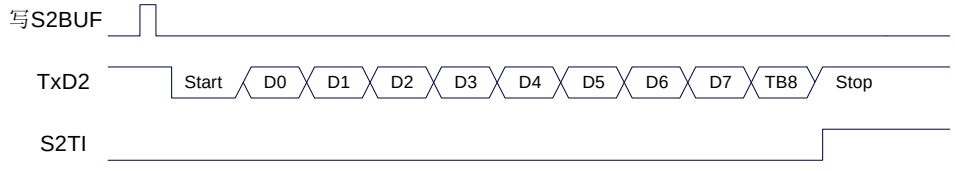
串口 2 的波特率是可变的，其波特率由定时器 2 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

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）

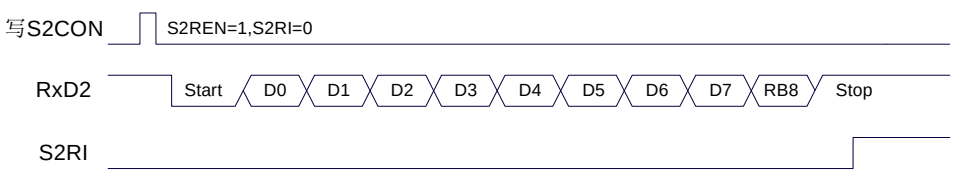
选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536 - \frac{\text{SYSclk}}{4\times\text{波特率}}$
	12T	定时器2重载值 = $65536 - \frac{\text{SYSclk}}{12\times4\times\text{波特率}}$

13.3.2 串口 2 模式 1

串行口 2 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD2 为数据发送口，RxD2 为数据接收口，串行口全双工接受/发送。



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。

13.4 串口 3

串口 3 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3SM0：指定串口3的通信工作模式，如下表所示：

S3SM0	串口3工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S3ST3：选择串口 3 的波特率发生器

0：选择定时器 2 为串口 3 的波特率发生器

1：选择定时器 3 为串口 3 的波特率发生器

S3SM2：允许串口 3 在模式 1 时允许多机通信控制位。在模式 1 时，如果 S3SM2 位为 1 且 S3REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 S3RB8）来筛选地址帧：若 S3RB8=1，说明该帧是地址帧，地址信息可以进入 S3BUF，并使 S3RI 为 1，进而在中断服务程序中再进行地址号比较；若 S3RB8=0，说明该帧不是地址帧，应丢掉且保持 S3RI=0。在模式 1 中，如果 S3SM2 位为 0 且 S3REN 位为 1，接收收机处于地址帧筛选被禁止状态。不论收到的 S3RB8 为 0 或 1，均可使接收到的信息进入 S3BUF，并使 S3RI=1，此时 S3RB8 通常为校验位。模式 0 为非多机通信方式，在这种方式时，要设置 S3SM2 应为 0。

S3REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

S3TB8：当串口 3 使用模式 1 时，S3TB8 为要发送的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位，按需要由软件置位或清 0。在模式 0 中，该位不用。

S3RB8：当串口 3 使用模式 1 时，S3RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 中，该位不用。

S3TI：串口 3 发送中断请求标志位。在停止位开始发送时由硬件自动将 S3TI 置 1，向 CPU 发请求中断，响应中断后 S3TI 必须用软件清零。

S3RI：串口 3 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S3RI 置 1，向 CPU 发中断申请，响应中断后 S3RI 必须由软件清零。

串口 3 数据寄存器

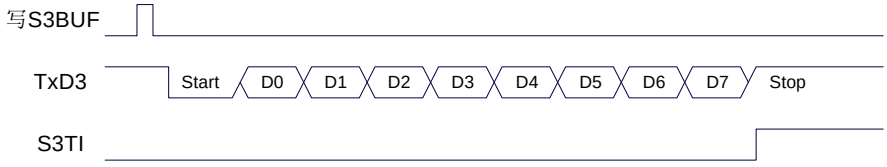
符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3BUF	ADH								

S3BUF：串口 1 数据接收/发送缓冲区。S3BUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 S3BUF 进行读操作，实际是读取串口接收缓冲区，对 S3BUF 进行写操作则是触发串口开始发送数据。

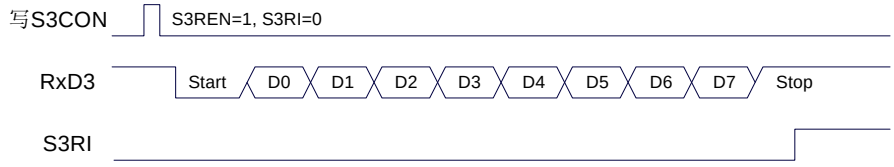
13.4.1 串口 3 模式 0

串行口 3 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD3 为数据

发送口，RxD3 为数据接收口，串行口全双工接受/发送。



发送数据（串口3模式0）



接收数据（串口3模式0）

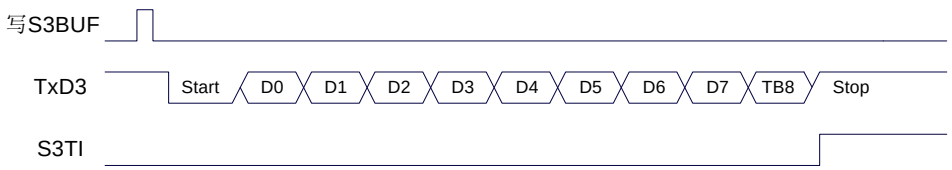
串口 3 的波特率是可变的，其波特率可由定时器 2 或定时器 3 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 3 模式 0 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

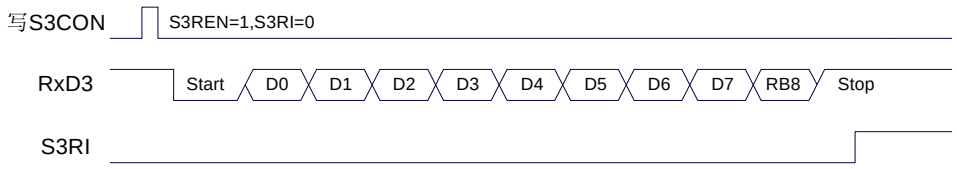
选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536-\frac{SYSclk}{4\times波特率}$
	12T	定时器2重载值 = $65536-\frac{SYSclk}{12\times4\times波特率}$
定时器3	1T	定时器3重载值 = $65536-\frac{SYSclk}{4\times波特率}$
	12T	定时器3重载值 = $65536-\frac{SYSclk}{12\times4\times波特率}$

13.4.2 串口 3 模式 1

串行口 3 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD3 为数据发送口，RxD3 为数据接收口，串行口全双工接受/发送。



发送数据（串口3模式1）



接收数据（串口3模式1）

串口 3 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

13.5 串口 4

串口 4 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4SM0：指定串口4的通信工作模式，如下表所示：

S4SM0	串口4工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S4ST4：选择串口 4 的波特率发生器

0：选择定时器 2 为串口 4 的波特率发生器

1：选择定时器 4 为串口 4 的波特率发生器

S4SM2：允许串口 4 在模式 1 时允许多机通信控制位。在模式 1 时，如果 S4SM2 位为 1 且 S4REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 S4RB8）来筛选地址帧：若 S4RB8=1，说明该帧是地址帧，地址信息可以进入 S4BUF，并使 S4RI 为 1，进而在中断服务程序中再进行地址号比较；若 S4RB8=0，说明该帧不是地址帧，应丢掉且保持 S4RI=0。在模式 1 中，如果 S4SM2 位为 0 且 S4REN 位为 1，接收收机处于地址帧筛选被禁止状态。不论收到的 S4RB8 为 0 或 1，均可使接收到的信息进入 S4BUF，并使 S4RI=1，此时 S4RB8 通常为校验位。模式 0 为非多机通信方式，在这种方式时，要设置 S4SM2 应为 0。

S4REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

S4TB8：当串口 4 使用模式 1 时，S4TB8 为要发送的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位，按需要由软件置位或清 0。在模式 0 中，该位不用。

S4RB8：当串口 4 使用模式 1 时，S4RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 中，该位不用。

S4TI：串口 4 发送中断请求标志位。在停止位开始发送时由硬件自动将 S4TI 置 1，向 CPU 发请求中断，响应中断后 S4TI 必须用软件清零。

S4RI：串口 4 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S4RI 置 1，向 CPU 发中断申请，响应中断后 S4RI 必须由软件清零。

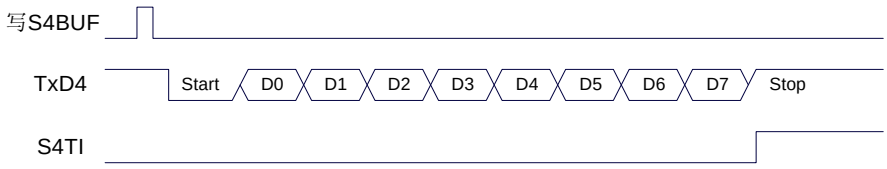
串口 4 数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S4BUF	85H								

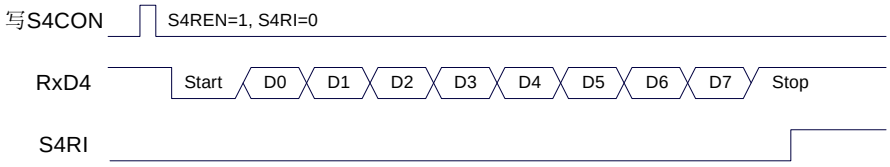
S4BUF：串口 1 数据接收/发送缓冲区。S4BUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 S4BUF 进行读操作，实际是读取串口接收缓冲区，对 S4BUF 进行写操作则是触发串口开始发送数据。

13.5.1 串口 4 模式 0

串行口 4 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD4 为数据发送口，RxD4 为数据接收口，串行口全双工接受/发送。



发送数据（串口4模式0）



接收数据（串口4模式0）

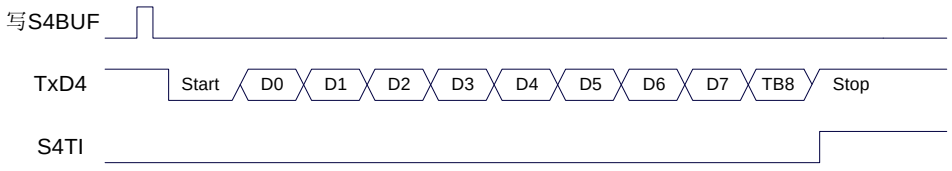
串口 4 的波特率是可变的，其波特率可由定时器 2 或定时器 4 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 4 模式 0 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

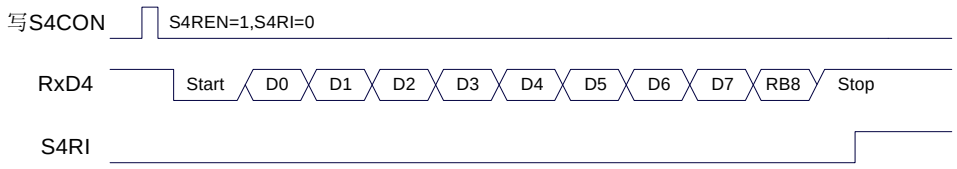
选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536-\frac{SYSclk}{4\times波特率}$
	12T	定时器2重载值 = $65536-\frac{SYSclk}{12\times4\times波特率}$
定时器4	1T	定时器4重载值 = $65536-\frac{SYSclk}{4\times波特率}$
	12T	定时器4重载值 = $65536-\frac{SYSclk}{12\times4\times波特率}$

13.5.2 串口 4 模式 1

串行口 4 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD4 为数据发送口，RxD4 为数据接收口，串行口全双工接受/发送。



发送数据（串口4模式1）



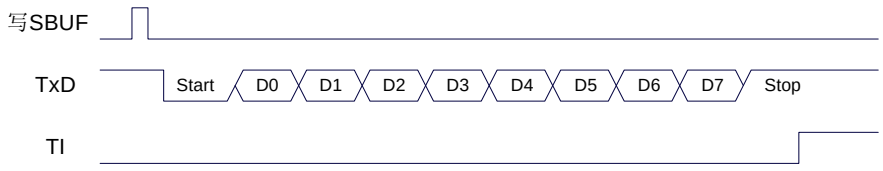
接收数据（串口4模式1）

串口 4 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

13.6 串口注意事项

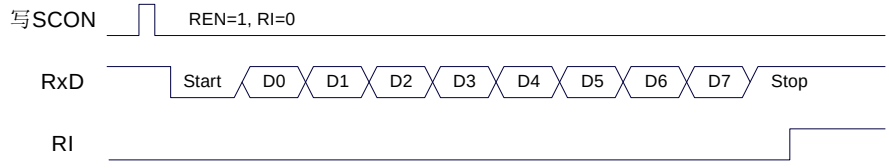
关于串口中断请求有如下问题需要注意：（串口 1、串口 2、串口 3、串口 4 均类似，下面以串口 1 为例进行说明）

8 位数据模式时，发送完成整个停止位后产生 TI 中断请求，如下图所示：



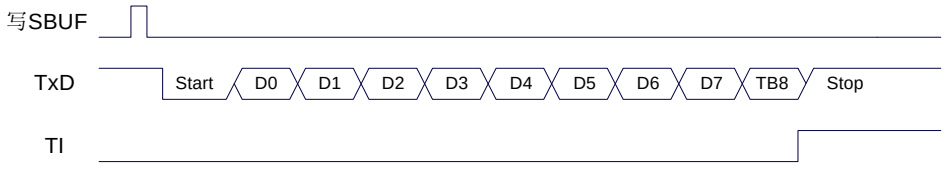
发送数据（8位数据）

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：



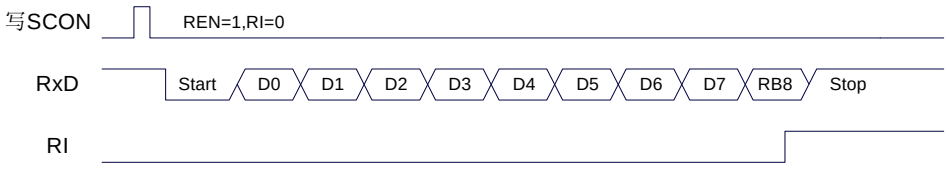
接收数据（8位数据）

9 位数据模式时，发送完成整个第 9 位数据位后产生 TI 中断请求，如下图所示：



发送数据（9位数据）

9 位数据模式时，接收完成一半个第 9 位数据位后产生 RI 中断请求，如下图所示：



接收数据（9位数据）

13.7 范例程序

13.7.1 串口 1 使用定时器 2 做波特率发生器

汇编代码

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

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

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

UART_INIT:
            MOV     SCON,#50H
            MOV     T2L,#0E8H           ;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,#3FH

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

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

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

                END
```

C 语言代码

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

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

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

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

void UartIsr() interrupt 4
{
    if (TI)
```

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

void UartInit()
{
    SCON = 0x50;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

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

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

void main()
{
    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

13.7.2 串口 1 使用定时器 1（模式 0）做波特率发生器

汇编代码

```
AUXR        DATA    8EH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

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

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#00H
            MOV     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,#3FH

                LCALL       UART_INIT
                SETB        ES
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART_SENDSTR

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

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

                END
```

C 语言代码

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

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

sfr         AUXR        =       0x8e;

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

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
```

```
            wptr &= 0x0f;
        }
}

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

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

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

void main()
{
        UartInit();
        ES = 1;
        EA = 1;
        UartSendStr("Uart Test !\r\n");

        while (1)
        {
            if (rptr != wptr)
            {
                UartSend(buffer[rptr++]);
                rptr &= 0x0f;
            }
        }
}
```

13.7.3 串口 1 使用定时器 1（模式 2）做波特率发生器

汇编代码

```
AUXR        DATA    8EH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes
```

```
                ORG     0000H
                LJMP    MAIN
                ORG     0023H
                LJMP    UART_ISR

                ORG     0100H

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

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

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#20H
                MOV     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,#3FH

                LCALL       UART_INIT
                SETB        ES
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART_SENDSTR

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

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

                END
```

C 语言代码

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

#define     FOSC                11059200UL
#define     BRT                 (256 - FOSC / 115200 / 32)

sfr         AUXR        =       0x8e;

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

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
```

```
    SCON = 0x50;
    TMOD = 0x20;
    TL1 = BRT;
    TH1 = BRT;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

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

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

void main()
{
    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

13.7.4 串口 2 使用定时器 2 做波特率发生器

汇编代码

```
AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S2CON       DATA    9AH
S2BUF       DATA    9BH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes
```

```
                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,#3FH
```

```
                LCALL   UART2_INIT
                MOV     IE2,#01H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART2_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART2_SEND
                INC     RPTR
                JMP     LOOP

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

                END
```

C 语言代码

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

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

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

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

void Uart2Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}
```

```
void Uart2Init()
{
	S2CON = 0x10;
	T2L = BRT;
	T2H = BRT >> 8;
	AUXR = 0x14;
	wptr = 0x00;
	rptr = 0x00;
	busy = 0;
}

void Uart2Send(char dat)
{
	while (busy);
	busy = 1;
	S2BUF = dat;
}

void Uart2SendStr(char *p)
{
	while (*p)
	{
		Uart2Send(*p++);
	}
}

void main()
{
	Uart2Init();
	IE2 = 0x01;
	EA = 1;
	Uart2SendStr("Uart Test !\r\n");

	while (1)
	{
		if (rptr != wptr)
		{
			Uart2Send(buffer[rptr++]);
			rptr &= 0x0f;
		}
	}
}
```

13.7.5 串口 3 使用定时器 2 做波特率发生器

汇编代码

```
AUXR		DATA	8EH
T2H		DATA	0D6H
T2L		DATA	0D7H
S3CON		DATA	0ACH
S3BUF		DATA	0ADH
IE2		DATA	0AFH

BUSY		BIT	20H.0
WPTR		DATA	21H
RPTR		DATA	22H
BUFFER		DATA	23H			;16 bytes
```

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

                ORG     0100H

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

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

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

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

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

MAIN:
                MOV     SP,#3FH
```

```
                LCALL   UART3_INIT
                MOV     IE2,#08H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART3_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART3_SEND
                INC     RPTR
                JMP     LOOP

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

                END
```

C 语言代码

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

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

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

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

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
```

```
}

void Uart3Init()
{
	S3CON = 0x10;
	T2L = BRT;
	T2H = BRT >> 8;
	AUXR = 0x14;
	wptr = 0x00;
	rptr = 0x00;
	busy = 0;
}

void Uart3Send(char dat)
{
	while (busy);
	busy = 1;
	S3BUF = dat;
}

void Uart3SendStr(char *p)
{
	while (*p)
	{
		Uart3Send(*p++);
	}
}

void main()
{
	Uart3Init();
	IE2 = 0x08;
	EA = 1;
	Uart3SendStr("Uart Test !\r\n");

	while (1)
	{
		if (rptr != wptr)
		{
			Uart3Send(buffer[rptr++]);
			rptr &= 0x0f;
		}
	}
}
```

13.7.6 串口 3 使用定时器 3 做波特率发生器

汇编代码

```
T4T3M	DATA	0D1H
T4H	DATA	0D2H
T4L	DATA	0D3H
T3H	DATA	0D4H
T3L	DATA	0D5H
S3CON	DATA	0ACH
S3BUF	DATA	0ADH
IE2	DATA	0AFH

BUSY	BIT	20H.0
```

```
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR

            ORG     0100H

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

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

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

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

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

```
MAIN:
                MOV     SP,#3FH

                LCALL   UART3_INIT
                MOV     IE2,#08H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART3_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART3_SEND
                INC     RPTR
                JMP     LOOP

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

                END
```

C 语言代码

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

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

sfr     T4T3M   =   0xd1;
sfr     T4H     =   0xd2;
sfr     T4L     =   0xd3;
sfr     T3H     =   0xd4;
sfr     T3L     =   0xd5;
sfr     S3CON   =   0xac;
sfr     S3BUF   =   0xad;
sfr     IE2     =   0xaf;

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

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
```

```
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x50;
    T3L = BRT;
    T3H = BRT >> 8;
    T4T3M = 0x0a;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3Send(*p++);
    }
}

void main()
{
    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

13.7.7 串口 4 使用定时器 2 做波特率发生器

汇编代码

```
AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H
S4CON   DATA    84H
S4BUF   DATA    085H
```

```
IE2             DATA    0AFH

BUSY            BIT     20H.0
WPTR            DATA    21H
RPTR            DATA    22H
BUFFER          DATA    23H                     ;16 bytes

                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,#3FH

                LCALL       UART4_INIT
                MOV         IE2,#10H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART4_SENDSTR

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

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

                END
```

C 语言代码

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

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

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

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

void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
```

```
        if (S4CON & 0x01)
        {
            S4CON &= ~0x01;
            buffer[wptr++] = S4BUF;
            wptr &= 0x0f;
        }
}

void Uart4Init()
{
        S4CON = 0x10;
        T2L = BRT;
        T2H = BRT >> 8;
        AUXR = 0x14;
        wptr = 0x00;
        rptr = 0x00;
        busy = 0;
}

void Uart4Send(char dat)
{
        while (busy);
        busy = 1;
        S4BUF = dat;
}

void Uart4SendStr(char *p)
{
        while (*p)
        {
            Uart4Send(*p++);
        }
}

void main()
{
        Uart4Init();
        IE2 = 0x10;
        EA = 1;
        Uart4SendStr("Uart Test !\r\n");

        while (1)
        {
            if (rptr != wptr)
            {
                Uart4Send(buffer[rptr++]);
                rptr &= 0x0f;
            }
        }
}
```

13.7.8 串口 4 使用定时器 4 做波特率发生器

汇编代码

```
T4T3M       DATA    0D1H
T4H         DATA    0D2H
T4L         DATA    0D3H
T3H         DATA    0D4H
```

```
T3L             DATA    0D5H
S4CON           DATA    84H
S4BUF           DATA    085H
IE2             DATA    0AFH

BUSY            BIT     20H.0
WPTR            DATA    21H
RPTR            DATA    22H
BUFFER          DATA    23H                 ;16 bytes

                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,#3FH

                LCALL       UART4_INIT
                MOV         IE2,#10H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART4_SENDSTR

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

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

                END
```

C 语言代码

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

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

sfr        T4T3M       =    0xd1;
sfr        T4H         =    0xd2;
sfr        T4L         =    0xd3;
sfr        T3H         =    0xd4;
sfr        T3L         =    0xd5;
sfr        S4CON       =    0x84;
sfr        S4BUF       =    0x85;
sfr        IE2         =    0xaf;

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

void Uart4Isr() interrupt 18
{
```

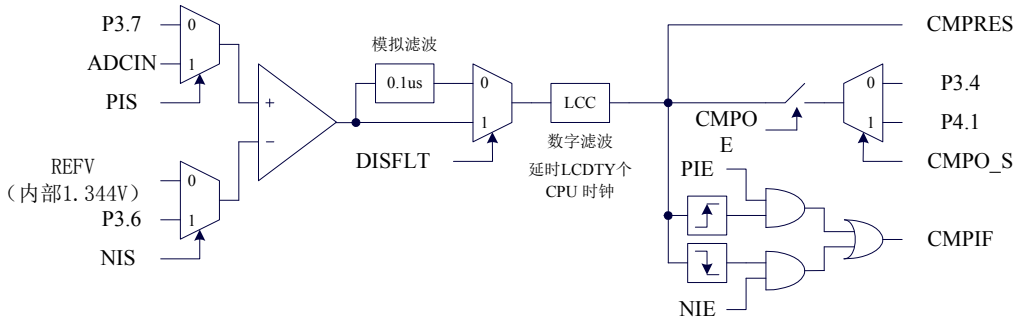
```
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4Send(*p++);
    }
}

void main()
{
    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4Send(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

14 比较器，掉电检测，内部固定比较电压

STC8 系列单片机内部集成了一个比较器。比较器的正极可以是 P3.7 端口或者 ADC 的模拟输入通道，而负极可以 P3.6 端口或者是内部 BandGap 经过 OP 后的 REFV 电压（内部固定比较电压）。

比较器内部有可程序控制的两级滤波：模拟滤波和数字滤波。模拟滤波可以过滤掉比较输入信号中的毛刺信号，数字滤波可以等待输入信号更加稳定后再进行比较。比较结果可直接通过读取内部寄存器位获得，也可将比较器结果正向或反向输出到外部端口。将比较结果输出到外部端口可用作外部事件的触发信号和反馈信号，可扩大比较的应用范围。

14.1 比较器内部结构图



比较器内部结构

14.2 比较器相关的寄存器

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

比较器控制寄存器 1

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

CMPEN：比较器模块使能位

0：关闭比较功能

1：使能比较功能

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

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

PIE：比较器上升沿中断使能位。

0：禁止比较器上升沿中断。

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

NIE：比较器下降沿中断使能位。

0：禁止比较器下降沿中断。

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

PIS：比较器的正极选择位

0：选择外部端口 P3.7 为比较器正极输入源。

1：通过 ADC_CONTR 中的 ADC_CHS 位选择 ADC 的模拟输入端作为比较器正极输入源。

NIS：比较器的负极选择位

0：选择内部 BandGap 经过 OP 后的电压 REFV 作为比较器负极输入源（REFV 的电压值为 1.344V，由于制造误差，实际电压值可能在 1.34V～1.35V 之间）。

1：选择外部端口 P3.6 为比较器负极输入源。

CMPOE：比较器结果输出控制位

0：禁止比较器结果输出

1：使能比较器结果输出。比较器结果输出到 P3.4 或者 P4.1（由 P_SW2 中的 CMPO_S 进行设定）

CMPRES：比较器的比较结果。此位为只读。

0：表示 CMP+的电平低于 CMP-的电平

1：表示 CMP+的电平高于 CMP-的电平

CMPRES 是经过数字滤波后的输出信号，而不是比较器的直接输出结果。

比较器控制寄存器 2

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR2	E7H	INVCMPO	DISFLT	LCDTY[5:0]					

INVCMPO：比较器结果输出控制

0：比较器结果正向输出。若 CMPRES 为 0，则 P3.4/P4.1 输出低电平，反之输出高电平。

1：比较器结果反向输出。若 CMPRES 为 0，则 P3.4/P4.1 输出高电平，反之输出低电平。

DISFLT：模拟滤波功能控制

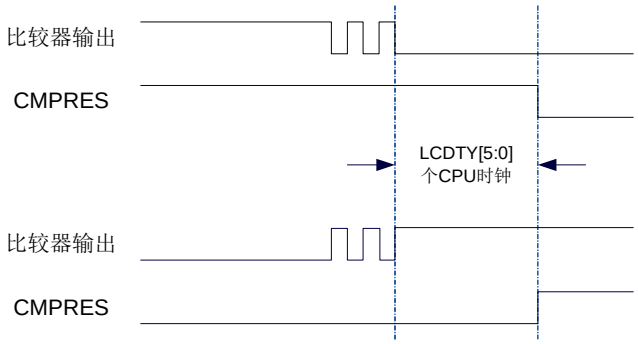
0：使能 0.1us 模拟滤波功能

1：关闭 0.1us 模拟滤波功能，可略微提高比较器的比较速度。

LCDTY[5:0]：数字滤波功能控制

数字滤波功能即为数字信号去抖动功能。当比较结果发生上升沿或者下降沿变化时，比较器侦测变化后的信号必须维持 LCDTY 所设置的 CPU 时钟数不发生变化，才认为数据变化是有效的；否则将视同信号无变化。

若 LCDTY 设置为 0 时表示关闭数字滤波功能。



14.3 范例程序

14.3.1 比较器的使用（中断方式）

汇编代码

```
CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

            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,#3FH

            MOV     CMPCR2,#00H
            ANL     CMPCR2,#NOT 80H         ;比较器正向输出
;           ORL     CMPCR2,#80H             ;比较器反向输出
            ANL     CMPCR2,#NOT 40H         ;禁止 0.1us 滤波
;           ORL     CMPCR2,#40H             ;使能 0.1us 滤波
;           ANL     CMPCR2,#NOT 3FH         ;比较器结果直接输出
            ORL     CMPCR2,#10H             ;比较器结果经过 16 个去抖时钟后输出
            MOV     CMPCR1,#00H
            ORL     CMPCR1,#30H             ;使能比较器边沿中断
;           ANL     CMPCR1,#NOT 20H         ;禁止比较器上升沿中断
;           ORL     CMPCR1,#20H             ;使能比较器上升沿中断
;           ANL     CMPCR1,#NOT 10H         ;禁止比较器下降沿中断
;           ORL     CMPCR1,#10H             ;使能比较器下降沿中断
            ANL     CMPCR1,#NOT 08H         ;P3.7 为 CMP+输入脚
;           ORL     CMPCR1,#08H             ;ADC 输入脚为 CMP+输入教
;           ANL     CMPCR1,#NOT 04H         ;内部参考电压为 CMP-输入脚
            ORL     CMPCR1,#04H             ;P3.6 为 CMP-输入脚
;           ANL     CMPCR1,#NOT 02H         ;禁止比较器输出
            ORL     CMPCR1,#02H             ;使能比较器输出
            ORL     CMPCR1,#80H             ;使能比较器模块
            SETB    EA

            JMP     $

            END
```

C 语言代码

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

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

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

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

void main()
{
    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80;                    //比较器正向输出
//  CMPCR2 |= 0x80;                     //比较器反向输出
    CMPCR2 &= ~0x40;                    //禁止 0.1us 滤波
//  CMPCR2 |= 0x40;                     //使能 0.1us 滤波
//  CMPCR2 &= ~0x3f;                    //比较器结果直接输出
    CMPCR2 |= 0x10;                     //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPCR1 |= 0x30;                     //使能比较器边沿中断
//  CMPCR1 &= ~0x20;                    //禁止比较器上升沿中断
//  CMPCR1 |= 0x20;                     //使能比较器上升沿中断
//  CMPCR1 &= ~0x10;                    //禁止比较器下降沿中断
//  CMPCR1 |= 0x10;                     //使能比较器下降沿中断
    CMPCR1 &= ~0x08;                    //P3.7 为 CMP+输入脚
//  CMPCR1 |= 0x08;                     //ADC 输入脚为 CMP+输入教
//  CMPCR1 &= ~0x04;                    //内部参考电压为 CMP-输入脚
    CMPCR1 |= 0x04;                     //P3.6 为 CMP-输入脚
//  CMPCR1 &= ~0x02;                    //禁止比较器输出
    CMPCR1 |= 0x02;                     //使能比较器输出
    CMPCR1 |= 0x80;                     //使能比较器模块

    EA = 1;

    while (1);
}
```

14.3.2 比较器的使用（查询方式）

汇编代码

```
CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H
```

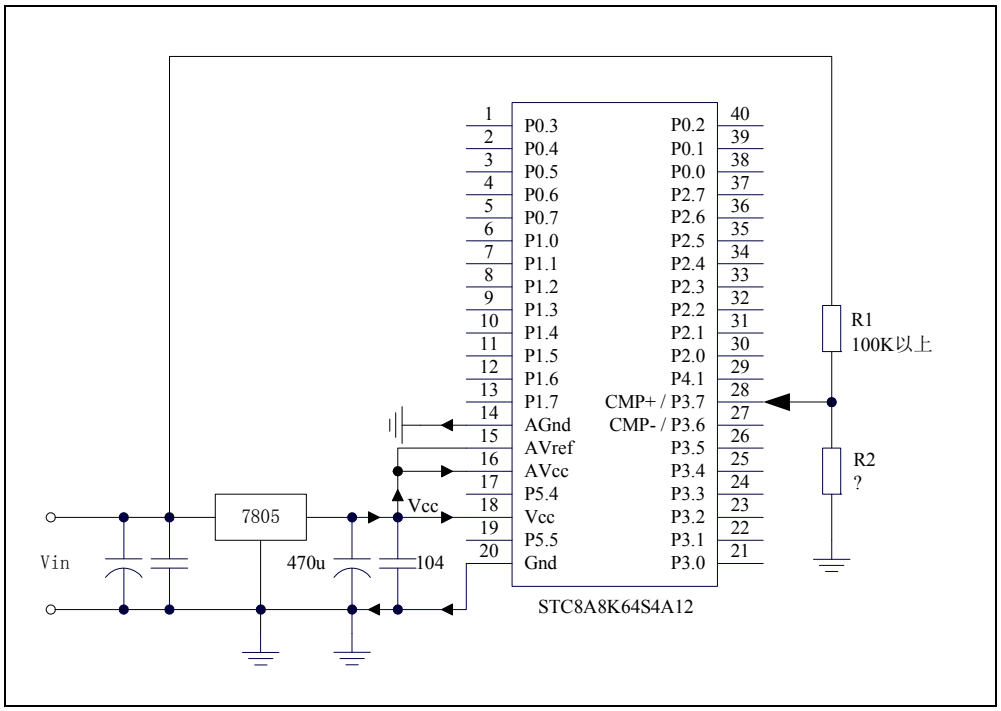
```
                ORG     0000H
                LJMP    MAIN

                ORG     0100H
MAIN:
                MOV     SP,#3FH

                MOV     CMPCR2,#00H
                ANL     CMPCR2,#NOT 80H         ;比较器正向输出
;               ORL     CMPCR2,#80H             ;比较器反向输出
                ANL     CMPCR2,#NOT 40H         ;禁止 0.1us 滤波
;               ORL     CMPCR2,#40H             ;使能 0.1us 滤波
;               ANL     CMPCR2,#NOT 3FH         ;比较器结果直接输出
                ORL     CMPCR2,#10H             ;比较器结果经过 16 个去抖时钟后输出
                MOV     CMPCR1,#00H
                ORL     CMPCR1,#30H             ;使能比较器边沿中断
;               ANL     CMPCR1,#NOT 20H         ;禁止比较器上升沿中断
;               ORL     CMPCR1,#20H             ;使能比较器上升沿中断
;               ANL     CMPCR1,#NOT 10H         ;禁止比较器下降沿中断
;               ORL     CMPCR1,#10H             ;使能比较器下降沿中断
                ANL     CMPCR1,#NOT 08H         ;P3.7 为CMP+输入脚
;               ORL     CMPCR1,#08H             ;ADC 输入脚为CMP+输入教
;               ANL     CMPCR1,#NOT 04H         ;内部参考电压为CMP-输入脚
                ORL     CMPCR1,#04H             ;P3.6 为CMP-输入脚
;               ANL     CMPCR1,#NOT 02H         ;禁止比较器输出
                ORL     CMPCR1,#02H             ;使能比较器输出
                ORL     CMPCR1,#80H             ;使能比较器模块

LOOP:
                MOV     A,CMPCR1
                MOV     C,ACC.0
                MOV     P1.0,C                  ;读取比较器比较结果
                JMP     LOOP

                END
```

C 语言代码

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

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

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

void main()
{
    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80;                //比较器正向输出
//  CMPCR2 |= 0x80;                 //比较器反向输出
    CMPCR2 &= ~0x40;                //禁止 0.1us 滤波
//  CMPCR2 |= 0x40;                 //使能 0.1us 滤波
//  CMPCR2 &= ~0x3f;                //比较器结果直接输出
    CMPCR2 |= 0x10;                 //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPCR1 |= 0x30;                 //使能比较器边沿中断
//  CMPCR1 &= ~0x20;                //禁止比较器上升沿中断
```

```
//      CMPCR1 |= 0x20;                 //使能比较器上升沿中断
//      CMPCR1 &= ~0x10;                //禁止比较器下降沿中断
//      CMPCR1 |= 0x10;                 //使能比较器下降沿中断
        CMPCR1 &= ~0x08;                //P3.7 为CMP+输入脚
//      CMPCR1 |= 0x08;                 //ADC 输入脚为CMP+输入教
//      CMPCR1 &= ~0x04;                //内部参考电压为CMP-输入脚
        CMPCR1 |= 0x04;                 //P3.6 为CMP-输入脚
//      CMPCR1 &= ~0x02;                //禁止比较器输出
        CMPCR1 |= 0x02;                 //使能比较器输出
        CMPCR1 |= 0x80;                 //使能比较器模块

        while (1)
        {
            P10 = CMPCR1 & 0x01;        //读取比较器比较结果
        }
}
```

14.3.3 比较器作外部掉电检测



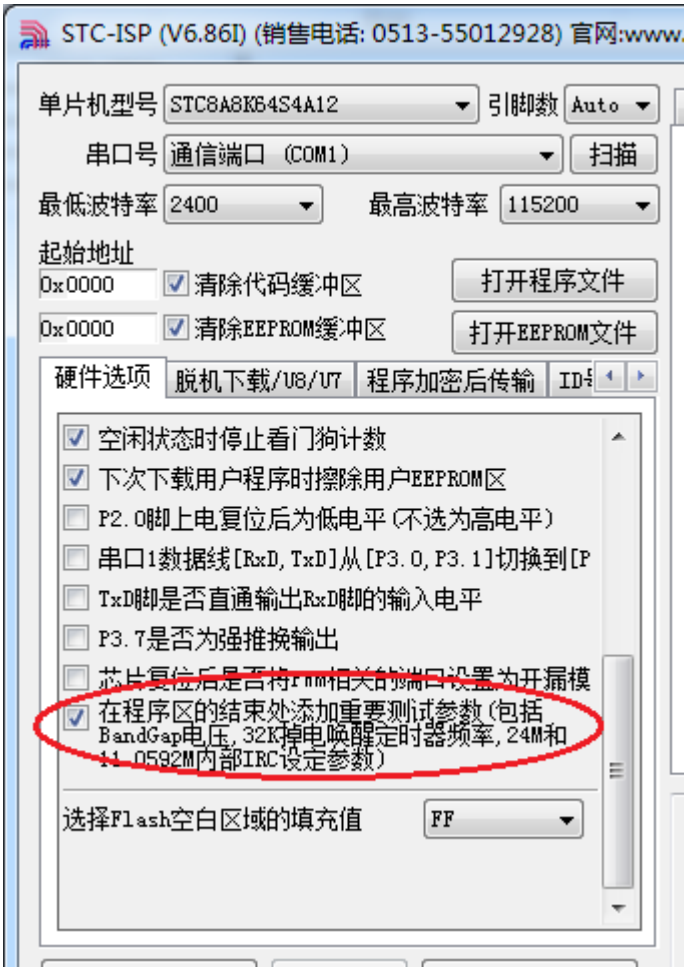
上图中电阻 R1 和 R2 对稳压块 7805 的前端电压进行分压，分压后的电压作为比较器 CMP+的外部输入与内部参考电压（约 1.344V 附近）进行比较。

一般当交流电在 220V 时，稳压块 7805 前端的直流电压为 11V，但当交流电压降到 160V 时，稳压块 7805 前端的直留电压为 8.5V。当稳压块 7805 前端的直留电压低于或等于 8.5V 时，该前端输入的直留电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+，CMP+端输入电压低于内部参考电压，此时可产生比较器中断，这样在掉电检测时就有充足的时间将数据保存到 EEPROM 中。当稳压块 7805 前端的直留电压高高于 8.5V 时，该前端输入的直流电压被电阻 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参考电压，此时 CPU 可继续正常工作。

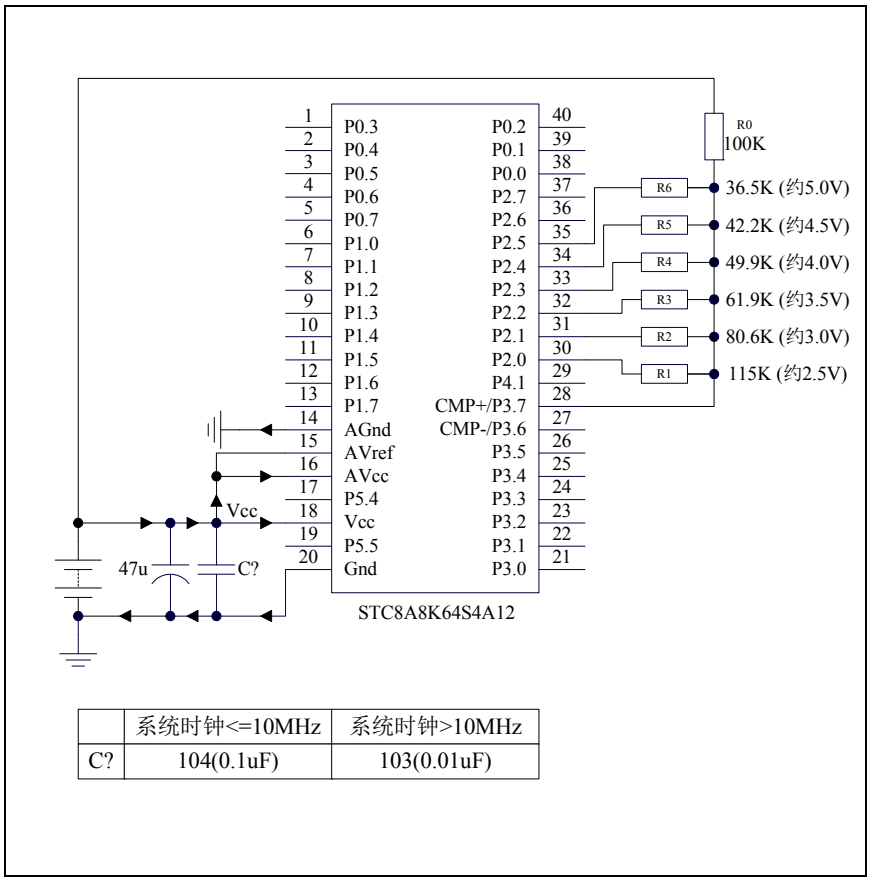
内部参考电压即为内部 BandGap 经过 OP 后的电压 REFV，REFV 的电压值约在 1.344V 附近，由于制造误差，实际电压值可能在 1.34V～1.35V 之间。具体的数值要通过读取内部参考电压在内部 RAM 区或者 ROM 区所占用的地址的值获得。对于 STC8 系列，内部参考电压值在 RAM 和 ROM 中的存储地址如下表所示：

单片机型号	RAM 中存储的地址（高字节在前）	ROM 中存储的地址（高字节在前）
STC8H1K16	0EFH-0F0H	3FF7H-3FF8H
STC8H1K28	0EFH-0F0H	6FF7H-6FF8H

注：若需要从 ROM 中读取参考电压值，则需要在进行 ISP 下载时勾选下列选项



14.3.4 比较器检测工作电压（电池电压）



	系统时钟<=10MHz	系统时钟>10MHz
C?	104(0.1uF)	103(0.01uF)

上图中，利用电阻分压的原理可以近似的测量出 MCU 的工作电压（选通的通道，MCU 的 IO 口输出低电平，端口电压值接近 GND，未选通的通道，MCU 的 IO 口输出开漏模式的高，不影响其他通道）。

比较器的负端选择内部参考电压（约 1.344V），正端选择通过电阻分压后输入到 CMP+管脚的电压值。

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

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

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

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

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

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

汇编代码

```
CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H
P2M0        DATA    96H
```

```
P2M1        DATA    95H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

            MOV     P2M0,#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     ;内部参考电压为 CMP-输入脚
            ANL     CMPCR1,#NOT 02H     ;禁止比较器输出
            ORL     CMPCR1,#80H         ;使能比较器模块
LOOP:
            MOV     R0,#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
```

C 语言代码

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

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

sfr     P2M0    =   0x96;
sfr     P2M1    =   0x95;

void delay ()
{
    char i;

    for (i=0; i<20; i++);
}

void main()
{
    unsigned char v;

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

    CMPCR2 = 0x10;                  //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPCR1 &= ~0x08;                //P3.6 为CMP+输入脚
    CMPCR1 &= ~0x04;                //内部参考电压为CMP-输入脚
    CMPCR1 &= ~0x02;                //禁止比较器输出
    CMPCR1 |= 0x80;                 //使能比较器模块

    while (1)
    {
        v = 0x00;                   //电压<2.5V
        P2 = 0xfe;                  //P2.0 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x01;                   //电压>2.5V
        P2 = 0xfd;                  //P2.1 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x03;                   //电压>3.0V
        P2 = 0xfb;                  //P2.2 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x07;                   //电压>3.5V
        P2 = 0xf7;                  //P2.3 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x0f;                   //电压>4.0V
        P2 = 0xef;                  //P2.4 输出 0
        delay();
```

```
            if (!(CMPCR1 & 0x01)) goto ShowVol;
            v = 0x1f;                       //电压>4.5V
            P2 = 0xdf;                      //P2.5 輸出 0
            delay();
            if (!(CMPCR1 & 0x01)) goto ShowVol;
            v = 0x3f;                       //电压>5.0V
ShowVol:
            P2 = 0xff;
            P0 = ~v;
        }
}
```

15 IAP/EEPROM

STC8H 系列单片机内部集成了大容量的 EEPROM。利用 ISP/IAP 技术可将内部 Data Flash 当 EEPROM，擦写次数在 10 万次以上。EEPROM 可分为若干个扇区，每个扇区包含 512 字节。使用时，建议同一次修改的数据放在同一个扇区，不是同一次修改的数据放在不同的扇区，不一定要用满。数据存储器的擦除操作是按扇区进行的。

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

15.1 EEPROM相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IAP_DATA	IAP 数据寄存器	C2H									1111,1111
IAP_ADDRH	IAP 高地址寄存器	C3H									0000,0000
IAP_ADDRL	IAP 低地址寄存器	C4H									0000,0000
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	-	CMD[1:0]		xxxx,xx00
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx
IAP_TPS	IAP 等待时间控制寄存器	F5H	-	-	IAPTPS[5:0]						xx00,0000

EEPROM 数据寄存器（IAP_DATA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_DATA	C2H								

在进行 EEPROM 的读操作时，命令执行完成后读出的 EEPROM 数据保存在 IAP_DATA 寄存器中。在进行 EEPROM 的写操作时，在执行写命令前，必须将待写入的数据存放在 IAP_DATA 寄存器中，再发送写命令。擦除 EEPROM 命令与 IAP_DATA 寄存器无关。

EEPROM 地址寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_ADDRH	C3H								
IAP_ADDRL	C4H								

EEPROM 进行读、写、擦除操作的目标地址寄存器。IAP_ADDRH 保存地址的高字节，IAP_ADDRL 保存地址的低字节

EEPROM 命令寄存器（IAP_CMD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	-	-	-	-	-	-	CMD[1:0]	

CMD[1:0]：发送EEPROM操作命令

00：空操作

01：读 EEPROM 命令。读取目标地址所在的 1 字节。

10：写 EEPROM 命令。写目标地址所在的 1 字节。

11：擦除 EEPROM。擦除目标地址所在的 1 页（1 扇区/512 字节）。

EEPROM 触发寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TRIG	C6H								

设置完成 EEPROM 读、写、擦除的命令寄存器、地址寄存器、数据寄存器以及控制寄存器后，需要向触发寄存器 IAP_TRIG 依次写入 5AH、A5H（顺序不能交换）两个触发命令来触发相应的读、写、擦除操作。操作完成后，EEPROM 地址寄存器 IAP_ADDRH、IAP_ADDRL 和 EEPROM 命令寄存器 IAP_CMD 的内容不变。如果接下来要对下一个地址的数据进行操作，需手动更新地址寄存器 IAP_ADDRH 和寄存器 IAP_ADDRL 的值。

注意：每次 EEPROM 操作时，都要对 IAP_TRIG 先写入 5AH，再写入 A5H，相应的命令才会生效。写完触发命令后，CPU 会处于 IDLE 等待状态，直到相应的 IAP 操作执行完成后 CPU 才会从 IDLE 状态返回正常状态继续执行 CPU 指令。

EEPROM 控制寄存器

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

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

EEPROM 擦除等待时间控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TPS	F5H	-	-	IAPTPS[5:0]					

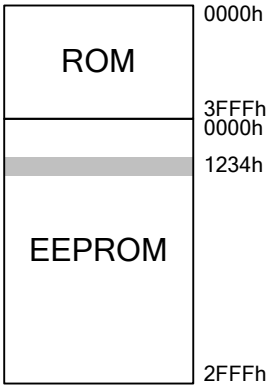
需要根据工作频率进行设置

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

15.2 EEPROM大小及地址

STC8H 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。

STC8H 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 STC8H1K16 这个型号为例，对目标地址进行详细说明：



STC8H1K16

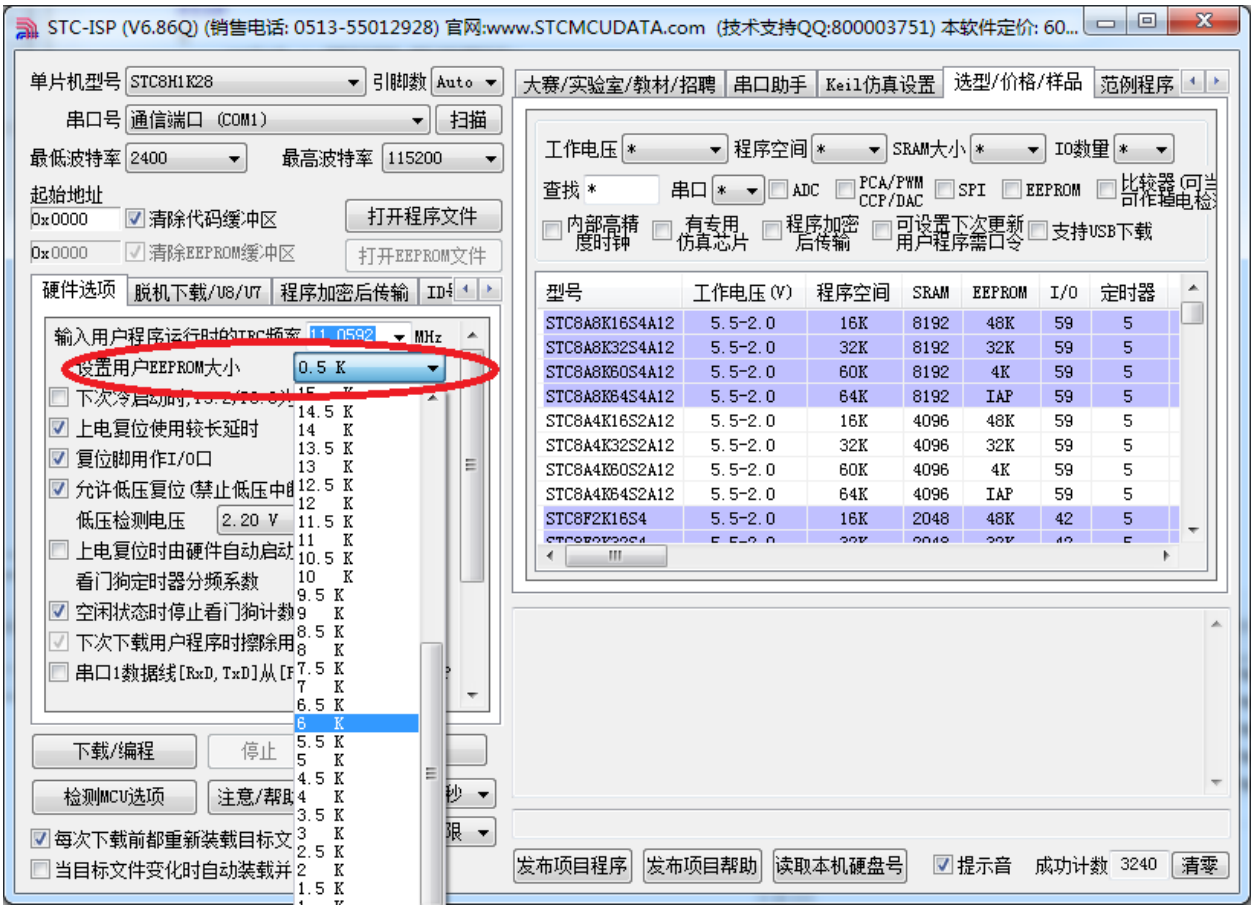
STC8H1K16 的程序空间为 16K 字节（0000h~3FFFh），EEPROM 空间为 12K（0000h~2FFFh）。当需要对 EEPROM 物理地址 1234h 的单元进行读、写、擦除时，若使用 IAP 方式进行访问时，设置的目标地址为 1234h，即 IAP_ADDRH 设置 12h，IAP_ADDRL 设置 34h，然后设置相应的触发命令即可对 1234h 单元进行正确操作了。但若是使用 MOVC 方式读取 EEPROM 的 1234h 单元，则必须在 1234h 的基础上还有加上 ROM 空间的大小 4000h，即必须将 DPTR 设置为 5234h，然后才能使用 MOVC 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
STC8H1K16	12K	24	0000h	2FFFh	4000h	6FFFh
STC8H1K28	用户自定义[1]					

[1]：STC8H1K28 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：STC8H1K28 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。

15.3 范例程序

15.3.1 EEPROM基本操作

汇编代码

```
;测试工作频率为11.0592MHz

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

            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             ;设置擦除等待参数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             ;设置擦除等待参数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             ;设置擦除等待参数 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,#3FH

                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
```

C 语言代码

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

//测试工作频率为 11.0592MHz

sfr     IAP_DATA    =   0xC2;
sfr     IAP_ADDRH   =   0xC3;
sfr     IAP_ADDRL   =   0xC4;
sfr     IAP_CMD     =   0xC5;
sfr     IAP_TRIG    =   0xC6;
sfr     IAP_CONTR   =   0xC7;
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;               //设置擦除等待参数 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;               //设置擦除等待参数 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;               //设置擦除等待参数 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()
{
        IapErase(0x0400);
        P0 = IapRead(0x0400);       //P0=0xff
        IapProgram(0x0400, 0x12);
        P1 = IapRead(0x0400);       //P1=0x12

        while (1);
}
```

15.3.2 使用MOVC读取EEPROM

汇编代码

```
;测试工作频率为 11.0592MHz

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H
IAP_ADDRL   DATA    0C4H
IAP_CMD     DATA    0C5H
```

```
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H
IAP_TPS     DATA    0F5H

IAP_OFFSET  EQU     4000H               ;STC8H1K16

            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         ;设置擦除等待参数 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         ;设置擦除等待参数 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,#3FH

            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
```

C 语言代码

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

//测试工作频率为 11.0592MHz

sfr     IAP_DATA    =   0xC2;
sfr     IAP_ADDRH   =   0xC3;
sfr     IAP_ADDRL   =   0xC4;
sfr     IAP_CMD     =   0xC5;
sfr     IAP_TRIG    =   0xC6;
sfr     IAP_CONTR   =   0xC7;
sfr     IAP_TPS     =   0xF5;

#define IAP_OFFSET      0x4000H         //STC8H1K16

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;               //设置擦除等待参数 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;               //设置擦除等待参数 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()
{
        IapErase(0x0400);
        P0 = IapRead(0x0400);       //P0=0xff
        IapProgram(0x0400, 0x12);
        P1 = IapRead(0x0400);       //P1=0x12

        while (1);
}
```

15.3.3 使用串口送出EEPROM数据

汇编代码

```
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

            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             ;设置擦除等待参数 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             ;设置擦除等待参数 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             ;设置擦除等待参数 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,#3FH

        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
```

C 语言代码

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

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

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

sfr     IAP_DATA    =   0xC2;
sfr     IAP_ADDRH   =   0xC3;
sfr     IAP_ADDRL   =   0xC4;
sfr     IAP_CMD     =   0xC5;
sfr     IAP_TRIG    =   0xC6;
sfr     IAP_CONTR   =   0xC7;
sfr     IAP_TPS     =   0xF5;

void UartInit()
{
        SCON = 0x5a;
        T2L = BRT;
        T2H = BRT >> 8;
        AUXR = 0x15;
}

void UartSend(char dat)
{
        while (!TI);
        TI = 0;
        SBUF = dat;
}

void IapIdle()
{
        IAP_CONTR = 0;              //关闭 IAP 功能
        IAP_CMD = 0;                //清除命令寄存器
        IAP_TRIG = 0;               //清除触发寄存器
        IAP_ADDRH = 0x80;           //将地址设置到非 IAP 区域
        IAP_ADDRL = 0;
}

char IapRead(int addr)
{
        char dat;

        IAP_CONTR = 0x80;           //使能 IAP
        IAP_TPS = 12;               //设置擦除等待参数 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;                   //设置擦除等待参数 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;                   //设置擦除等待参数 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()
{
        UartInit();
        IapErase(0x0400);
        UartSend(IapRead(0x0400));
        IapProgram(0x0400, 0x12);
        UartSend(IapRead(0x0400));

        while (1);
}
```

16 ADC模数转换

STC8H 系列单片机内部集成了一个 10 位 12 通道的高速 A/D 转换器（**注：第 16 通道只能用于检测内部参考电压，参考电压值出厂时校准为 1.236V，由于制造误差，实际电压值可能在 1.23V～1.24V 之间**）。ADC 的时钟频率为系统频率 2 分频再经过用户设置的分频系数进行再次分频（ADC 的时钟频率范围为 SYSclk/2/1～SYSclk/2/16）。每固定 16 个 ADC 时钟可完成一次 A/D 转换。

ADC 转换结果的数据格式有两种：左对齐和右对齐。可方便用户程序进行读取和引用。

16.1 ADC相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
ADC_CONTR	ADC 控制寄存器	BCH	ADC_POWER	ADC_START	ADC_FLAG	-	ADC_CHS[3:0]				000x,0000
ADC_RES	ADC 转换结果高位寄存器	BDH									0000,0000
ADC_RESL	ADC 转换结果低位寄存器	BEH									0000,0000
ADCCFG	ADC 配置寄存器	DEH	-	-	RESFMT	-	SPEED[3:0]				xx0x,0000

ADC 控制寄存器

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

ADC_POWER：ADC 电源控制位

0：关闭 ADC 电源

1：打开 ADC 电源。

建议进入空闲模式和掉电模式前将 ADC 电源关闭，以降低功耗

ADC_START：ADC 转换启动控制位。写入 1 后开始 ADC 转换，转换完成后硬件自动将此位清零。

0：无影响。即使 ADC 已经开始转换工作，写 0 也不会停止 A/D 转换。

1：开始 ADC 转换，转换完成后硬件自动将此位清零。

ADC_FLAG：ADC 转换结束标志位。当 ADC 完成一次转换后，硬件会自动将此位置 1，并向 CPU 提出中断请求。此标志位必须软件清零。

ADC_CHS[3:0]：ADC 模拟通道选择位

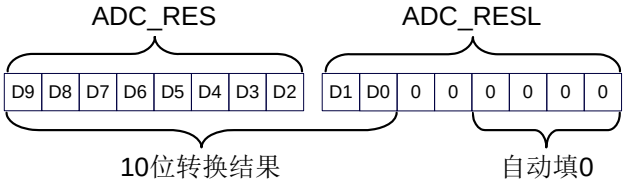
ADC_CHS[3:0]	ADC 通道	ADC_CHS[3:0]	ADC 通道
0000	P1.0	1000	P0.0
0001	P1.1	1001	P0.1
0010	P1.2	1010	P0.2
0011	P1.3	1011	P0.3
0100	P1.4	1100	-
0101	P1.5	1101	-
0110	P1.6	1110	-
0111	P1.7	1111	测试内部 1.236V

ADC 配置寄存器

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

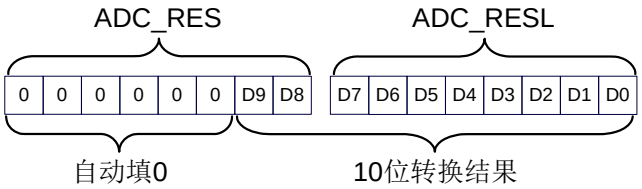
RESFMT：ADC 转换结果格式控制位

0：转换结果左对齐。ADC_RES 保存结果的高 8 位，ADC_RESL 保存结果的低 2 位。格式如下：



RESFMT=0

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：



RESFMT=1

SPEED[3:0]：ADC 时钟控制（F_{ADC}＝SYSclk/2/16/SPEED）

SPEED[3:0]	ADC 转换时间（CPU 时钟数）	SPEED[3:0]	ADC 转换时间（CPU 时钟数）
0000	32	1000	288
0001	64	1001	320
0010	96	1010	352
0011	128	1011	384
0100	160	1100	416
0101	192	1101	448
0110	224	1110	480
0111	256	1111	512

ADC 转换结果寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ADC_RES	BDH								
ADC_RESL	BEH								

当 A/D 转换完成后，10 为的转换结果会自动保存到 ADC_RES 和 ADC_RESL 中。保存结果的数据格式请参考 ADC_CFG 寄存器中的 RESFMT 设置。

16.2 范例程序

16.2.1 ADC基本操作（查询方式）

汇编代码

```
;测试工作频率为11.0592MHz

ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH

P1M0        DATA    092H
P1M1        DATA    091H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

            MOV     P1M0,#00H           ;设置P1.0为ADC口
            MOV     P1M1,#01H
            MOV     ADCCFG,#0FH         ;设置ADC时钟为系统时钟/2/16/16
            MOV     ADC_CONTR,#80H      ;使能ADC模块
LOOP:
            ORL     ADC_CONTR,#40H      ;启动AD转换
            NOP
            NOP
            MOV     A,ADC_CONTR         ;查询ADC完成标志
            JNB     ACC.5,$-2
            ANL     ADC_CONTR,#NOT 20H  ;清完成标志
            MOV     P2,ADC_RES          ;读取ADC结果

            SJMP    LOOP

            END
```

C 语言代码

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

//测试工作频率为11.0592MHz

sfr     ADC_CONTR   =   0xbc;
sfr     ADC_RES     =   0xbd;
sfr     ADC_RESL    =   0xbe;
sfr     ADCCFG      =   0xde;

sfr     P1M0        =   0x92;
sfr     P1M1        =   0x91;

void main()
{
    P1M0 = 0x00;                    //设置P1.0为ADC口
```

```
    P1M1 = 0x01;
    ADCCFG = 0x0f;                  //设置 ADC 时钟为系统时钟/2/16/16
    ADC_CONTR = 0x80;               //使能 ADC 模块

    while (1)
    {
        ADC_CONTR |= 0x40;          //启动 AD 转换
        _nop_();
        _nop_();
        while (!(ADC_CONTR & 0x20)); //查询 ADC 完成标志
        ADC_CONTR &= ~0x20;         //清完成标志
        P2 = ADC_RES;               //读取 ADC 结果
    }
}
```

16.2.2 ADC基本操作（中断方式）

汇编代码

```
;测试工作频率为 11.0592MHz

ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH

EADC        BIT     IE.5

P1M0        DATA    092H
P1M1        DATA    091H

            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,#3FH

            MOV     P1M0,#00H               ;设置 P1.0 为 ADC 口
            MOV     P1M1,#01H
            MOV     ADCCFG,#0FH             ;设置 ADC 时钟为系统时钟/2/16/16
            MOV     ADC_CONTR,#80H          ;使能 ADC 模块
            SETB    EADC                    ;使能 ADC 中断
            SETB    EA
            ORL     ADC_CONTR,#40H          ;启动 AD 转换

            SJMP    $

            END
```

C 语言代码

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

//测试工作频率为11.0592MHz

sfr         ADC_CONTR   =   0xbc;
sfr         ADC_RES     =   0xbd;
sfr         ADC_RESL    =   0xbe;
sfr         ADCCFG      =   0xde;

sbit        EADC        =   IE^5;

sfr         P1M0        =   0x92;
sfr         P1M1        =   0x91;

void ADC_Isr() interrupt 5
{
    ADC_CONTR &= ~0x20;             //清中断标志
    P2 = ADC_RES;                   //读取 ADC 结果
    ADC_CONTR |= 0x40;              //继续 AD 转换
}

void main()
{
    P1M0 = 0x00;                    //设置 P1.0 为 ADC 口
    P1M1 = 0x01;
    ADCCFG = 0x0f;                  //设置 ADC 时钟为系统时钟/2/16/16
    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);
}
```

16.2.3 格式化ADC转换结果

汇编代码

```
;测试工作频率为11.0592MHz

ADC_CONTR   DATA    0BCH
ADC_RES     DATA    0BDH
ADC_RESL    DATA    0BEH
ADCCFG      DATA    0DEH

P1M0        DATA    092H
P1M1        DATA    091H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

            MOV     P1M0,#00H           ;设置 P1.0 为 ADC 口
```

```
                MOV     P1M1,#01H
                MOV     ADCCFG,#0FH             ;设置 ADC 时钟为系统时钟/2/16/16
                MOV     ADC_CONTR,#80H          ;使能 ADC 模块

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

                MOV     ADCCFG,#00H             ;设置结果左对齐
                MOV     A,ADC_RES               ;A 存储 ADC 的 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
```

C 语言代码

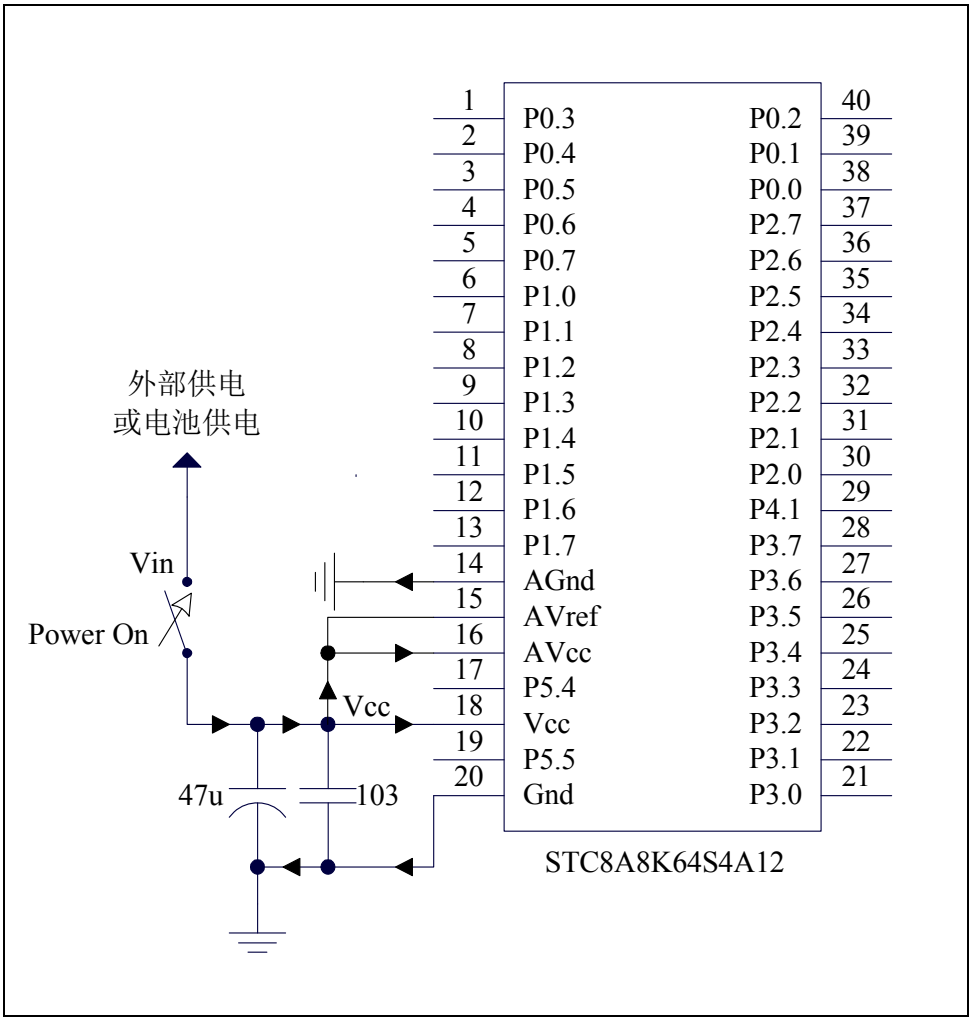
```
#include "reg51.h"
#include "intrins.h"

//测试工作频率为 11.0592MHz

sfr     ADC_CONTR   =   0xbc;
sfr     ADC_RES     =   0xbd;
sfr     ADC_RESL    =   0xbe;
sfr     ADCCFG      =   0xde;

sfr     P1M0        =   0x92;
sfr     P1M1        =   0x91;

void main()
{
    P1M0 = 0x00;                    //设置 P1.0 为 ADC 口
    P1M1 = 0x01;
    ADCCFG = 0x0f;                  //设置 ADC 时钟为系统时钟/2/16/16
    ADC_CONTR = 0x80;               //使能 ADC 模块
    ADC_CONTR |= 0x40;              //启动 AD 转换
    _nop_();
    _nop_();
    while (!(ADC_CONTR & 0x20));    //查询 ADC 完成标志
    ADC_CONTR &= ~0x20;             //清完成标志

    ADCCFG = 0x00;                  //设置结果左对齐
    ACC = ADC_RES;                  //A 存储 ADC 的 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);
```

```
}
```

16.2.4 利用ADC第 16 通道测量外部电压或电池电压

STC8 系列 ADC 的第 16 通道是用来测试内部 BandGap 参考电压的，由于内部 BandGap 参考电压很稳定，约为 1.35V，且不会随芯片的工作电压的改变而变化，所以可以通过测量内部 BandGap 参考电压，然后通过 ADC 的值便可反推出外部电压或外部电池电压。

下图为参考线路图:



C 语言代码

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

//测试工作频率为11.0592MHz

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

sfr        AUXR        =    0x8e;

sfr        ADC_CONTR   =    0xbc;
```

```
sfr         ADC_RES     =   0xbd;
sfr         ADC_RESL    =   0xbe;
sfr         ADCCFG      =   0xde;

sfr         P1M0        =   0x92;
sfr         P1M1        =   0x91;

int         *BGV;                   //内部 Bandgap 电压值存放在 idata 中
                                    //idata 的 EFH 地址存放高字节
                                    //idata 的 F0H 地址存放低字节
                                    //电压单位为毫伏(mV)
bit         busy;

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

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

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

void ADCInit()
{
    ADCCFG = 0x2f;              //设置 ADC 时钟为系统时钟/2/16/16
    ADC_CONTR = 0x8f;           //使能 ADC 模块,并选择第 16 通道
}

int   ADCRead()
{
    int res;

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

```
    res = (ADC_RES << 8) | ADC_RESL;  //读取 ADC 结果

    return res;
}

void main()
{
    int res;
    int vcc;
    int i;

    BGV = (int idata *)0xfe;
    ADCInit();                        //ADC 初始化
    UartInit();                       //串口初始化

    ES = 1;
    EA = 1;

    ADCRead();
    ADCRead();                        //前两个数据丢弃
    res = 0;
    for (i=0; i<8; i++)
    {
        res += ADCRead();             //读取 8 次数据
    }
    res >>= 3;                        //取平均值

    vcc = (int)(4095L * *BGV / res);  //计算 VREF 管脚电压,即电池电压
                                      //注意,此电压的单位为毫伏(mV)
    UartSend(vcc >> 8);               //输出电压值到串口
    UartSend(vcc);

    while (1);
}
```

17 同步串行外设接口SPI

STC8 系列单片机内部集成了一种高速串行通信接口——SPI 接口。SPI 是一种全双工的高速同步通信总线。STC8 系列集成的 SPI 接口提供了两种操作模式：主模式和从模式。

17.1 SPI相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SPSTAT	SPI 状态寄存器	CDH	SPIF	WCOL	-	-	-	-	-	-	00xx,xxxx
SPCTL	SPI 控制寄存器	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]		0000,0100
SPDAT	SPI 数据寄存器	CFH									0000,0000

SPI 状态寄存器

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

SPIF：SPI 中断标志位。

当发送/接收完成 1 字节的数据后，硬件自动将此位置 1，并向 CPU 提出中断请求。当 SSIG 位被设置为 0 时，由于 SS 管脚电平的变化而使得设备的主/从模式发生改变时，此标志位也会被硬件自动置 1，以标志设备模式发生变化。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

WCOL：SPI 写冲突标志位。

当 SPI 在进行数据传输的过程中写 SPDAT 寄存器时，硬件将此位置 1。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

SPI 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]	

SSIG：SS 引脚功能控制位

0：SS 引脚确定器件是主机还是从机

1：忽略 SS 引脚功能，使用 MSTR 确定器件是主机还是从机

SPEN：SPI 使能控制位

0：关闭 SPI 功能

1：使能 SPI 功能

DORD：SPI 数据位发送/接收的顺序

0：先发送/接收数据的高位（MSB）

1：先发送/接收数据的低位（LSB）

MSTR：器件主/从模式选择位

设置主机模式：

若 SSIG＝0，则 SS 管脚必须为高电平且设置 MSTR 为 1

若 SSIG＝1，则只需要设置 MSTR 为 1（忽略 SS 管脚的电平）

设置从机模式：

若 SSIG＝0，则 SS 管脚必须为低电平（与 MSTR 位无关）

若 SSIG＝1，则只需要设置 MSTR 为 0（忽略 SS 管脚的电平）

CPOL：SPI 时钟极性控制

0：SCLK 空闲时为低电平，SCLK 的前时钟沿为上升沿，后时钟沿为下降沿

1：SCLK 空闲时为高电平，SCLK 的前时钟沿为下降沿，后时钟沿为上升沿

CPHA：SPI 时钟相位控制

0：数据 SS 管脚为低电平驱动第一位数据并在 SCLK 的后时钟沿改变数据，前时钟沿采样数据（必须 SSIG＝0）

1：数据在 SCLK 的前时钟沿驱动，后时钟沿采样

SPR[1:0]：SPI 时钟频率选择

SPR[1:0]	SCLK 频率
00	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	SYSclk/32

SPI 数据寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH								

SPI 发送/接收数据缓冲器。

17.2 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）

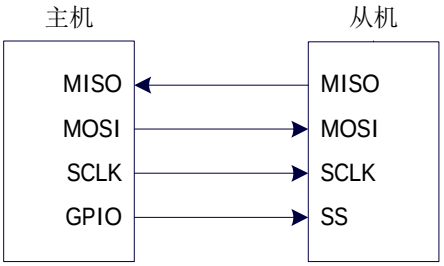
17.2.1 单主单从

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

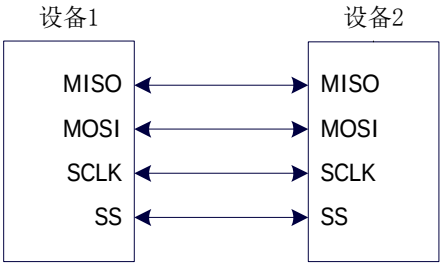
17.2.2 互为主从

两个设备相连，主机和从机不固定。

设置方法 1：两个设备初始化时都设置为 SSIG 设置为 0，MSTR 设置为 1，且将 SS 脚设置为双向口模式输出高电平。此时两个设备都是不忽略 SS 的主机模式。当其中一个设备需要启动传输时，可将自己的 SS 脚设置为输出模式并输出低电平，拉低对方的 SS 脚，这样另一个设备就被强行设置为从机模式了。

设置方法 2：两个设备初始化时都将自己设置成忽略 SS 的从机模式，即将 SSIG 设置为 1，MSTR 设置为 0。当其中一个设备需要启动传输时，先检测 SS 管脚的电平，如果时候高电平，就将自己设置成忽略 SS 的主模式，即可进行数据传输了。

互为主从连接配置图如下所示：



互为主从配置

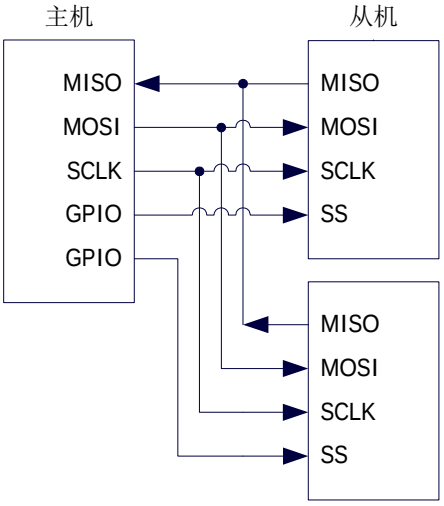
17.2.3 单主多从

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

17.3 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。

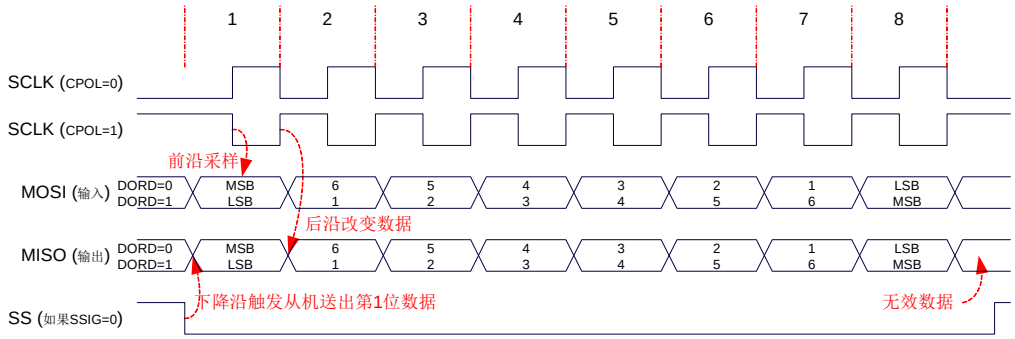
当对主机或从机进行写冲突检测时，主机发生写冲突的情况是很罕见的，因为主机拥有数据传输的完全控制权。但从机有可能发生写冲突，因为当主机启动传输时，从机无法进行控制。

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。

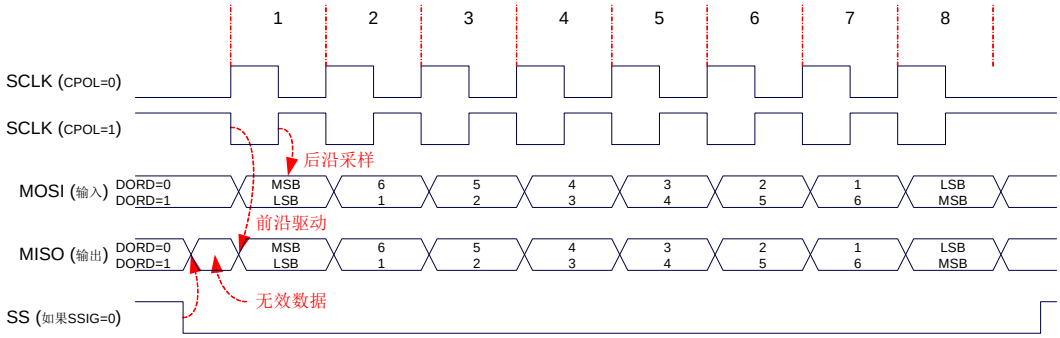
WCOL 可通过软件向其写入“1”清零。

17.4 数据模式

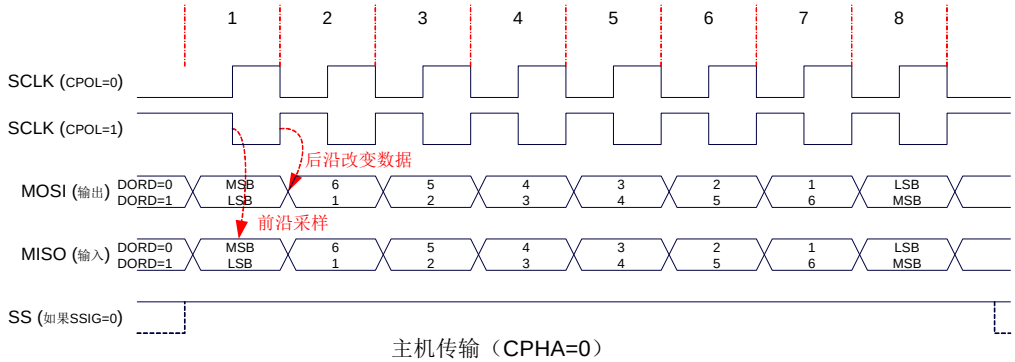
SPI 的时钟相位控制位 CPHA 可以让用户设定数据采样和改变时的时钟沿。时钟极性位 CPOL 可以让用户设定时钟极性。下面图例显示了不同时钟相位、极性设置下 SPI 通讯时序。



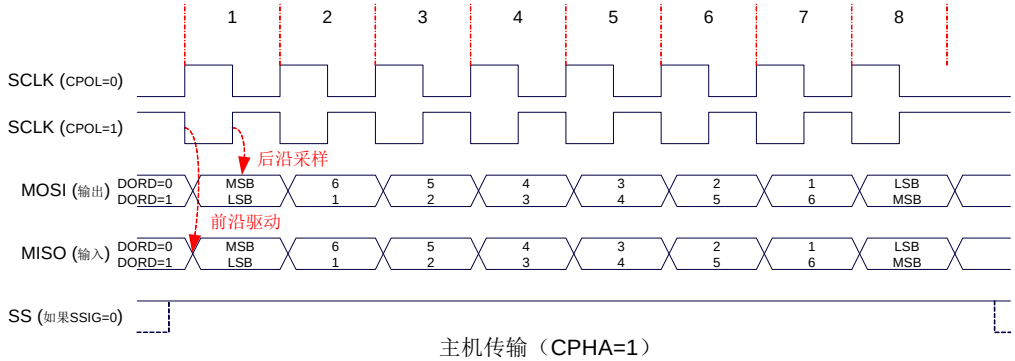
从机传输（CPHA=0）



从机传输（CPHA=1）



主机传输（CPHA=0）



主机传输（CPHA=1）

17.5 范例程序

17.5.1 SPI单主单从系统主机程序（中断方式）

汇编代码

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

BUSY        BIT     20H.0
SS          BIT     P1.0
LED         BIT     P1.1

            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,#3FH

            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
```

C 语言代码

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

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
sfr     SPDAT       =   0xcf;
```

```
sfr         IE2             =   0xaf;
#define     ESPI                0x02

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

bit         busy;

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

void main()
{
    LED = 1;
    SS = 1;
    busy = 0;

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

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

17.5.2 SPI单主单从系统从机程序（中断方式）

汇编代码

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

LED         BIT     P1.1

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H        ;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,#3FH

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

        JMP     $

        END
```

C 语言代码

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

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI        0x02

sbit    LED     =   P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;              //清中断标志
    SPDAT = SPDAT;              //将接收到的数据回传给主机
    LED = !LED;                 //测试端口
}

void main()
{
    SPCTL = 0x40;               //使能 SPI 从机模式
    SPSTAT = 0xc0;              //清中断标志
    IE2 = ESPI;                 //使能 SPI 中断
    EA = 1;

    while (1);
}
```

17.5.3 SPI单主单从系统主机程序（查询方式）

汇编代码

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

SS          BIT     P1.0
LED         BIT     P1.1

            ORG     0000H
            LJMP    MAIN
```

```
            ORG     0100H
MAIN:
            MOV     SP,#3FH

            SETB    LED
            SETB    SS

            MOV     SPCTL,#50H          ;使能 SPI 主机模式
            MOV     SPSTAT,#0C0H        ;清中断标志

LOOP:
            CLR     SS                  ;拉低从机 SS 管脚
            MOV     SPDAT,#5AH          ;发送测试数据
            MOV     A,SPSTAT            ;查询完成标志
            JNB     ACC.7,$-2
            MOV     SPSTAT,#0C0H        ;清中断标志
            SETB    SS
            CPL     LED
            JMP     LOOP

            END
```

C 语言代码

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

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
sfr     SPDAT       =   0xcf;
sfr     IE2         =   0xaf;
#define ESPI            0x02

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

void main()
{
    LED = 1;
    SS = 1;

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

    while (1)
    {
        SS = 0;                     //拉低从机 SS 管脚
        SPDAT = 0x5a;               //发送测试数据
        while (!(SPSTAT & 0x80));   //查询完成标志
        SPSTAT = 0xc0;              //清中断标志
        SS = 1;                     //拉高从机的 SS 管脚
        LED = !LED;                 //测试端口
    }
}
```

17.5.4 SPI单主单从系统从机程序（查询方式）

汇编代码

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

LED         BIT     P1.1

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

            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
```

C 语言代码

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

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
sfr     SPDAT       =   0xcf;
sfr     IE2         =   0xaf;
#define ESPI            0x02

sbit    LED         =   P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //清中断标志
}

void main()
{
    SPCTL = 0x40;                   //使能 SPI 从机模式
    SPSTAT = 0xc0;                  //清中断标志

    while (1)
    {
        while (!(SPSTAT & 0x80));   //查询完成标志
        SPSTAT = 0xc0;              //清中断标志
```

```
            SPDAT = SPDAT;              //将接收到的数据回传给主机
            LED = !LED;                 //测试端口
        }
}
```

17.5.5 SPI互为主从系统程序（中断方式）

汇编代码

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

SS          BIT     P1.0
LED         BIT     P1.1
KEY         BIT     P0.0

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            PUSH    ACC
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     A,SPCTL
            JB      ACC.4,MASTER
SLAVE:
            MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
            JMP     ISREXIT
MASTER:
            SETB    SS                  ;拉高从机的SS 管脚
            MOV     SPCTL,#40H          ;重新设置为从机待机
ISREXIT:
            CPL     LED
            POP     ACC
            RETI

MAIN:
            MOV     SP,#3FH

            SETB    SS
            SETB    LED
            SETB    KEY

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

LOOP:
            JB      KEY,LOOP            ;等待按键触发
            MOV     SPCTL,#50H          ;使能SPI 主机模式
            CLR     SS                  ;拉低从机SS 管脚
            MOV     SPDAT,#5AH          ;发送测试数据
            JNB     KEY,$               ;等待按键释放
```

```
        JMP     LOOP

        END
```

C 语言代码

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

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI        0x02

sbit    SS      =   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()
{
    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
        }
    }
}
```

17.5.6 SPI互为主从系统程序（查询方式）

汇编代码

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

SS          BIT     P1.0
LED         BIT     P1.1
KEY         BIT     P0.0

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP,#3FH

            SETB    SS
            SETB    LED
            SETB    KEY

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

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
```

C 语言代码

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

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
```

```
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sbit        SS          =   P1^0;
sbit        LED         =   P1^1;
sbit        KEY         =   P0^0;

void main()
{
    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;                   //使能 SPI 从机模式进行待机
    SPSTAT = 0xc0;                  //清中断标志

    while (1)
    {
        if (!KEY)                   //等待按键触发
        {
            SPCTL = 0x50;           //使能 SPI 主机模式
            SS = 0;                 //拉低从机 SS 管脚
            SPDAT = 0x5a;           //发送测试数据
            while (!KEY);           //等待按键释放
        }
        if (SPSTAT & 0x80)
        {
            SPSTAT = 0xc0;          //清中断标志
            if (SPCTL & 0x10)
            {                       //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
        }
    }
}
```

18 I²C总线

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

与标准 I²C 协议相比较，忽略了如下两种机制：

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

STC8 系列的 I²C 总线提供了两种操作模式：主机模式（SCL 为输出口，发送同步时钟信号）和从机模式（SCL 为输入口，接收同步时钟信号）

18.1 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[6:1]						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	SLADR[6:0]							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

18.2 I²C主机模式

I²C 配置寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CCFG	FE80H	ENI2C	MSSL	MSSPEED[6:1]					

ENI2C：I²C 功能使能控制位

0：禁止 I²C 功能

1：允许 I²C 功能

MSSL：I²C 工作模式选择位

0：从机模式

1：主机模式

MSSPEED[6:1]：I²C 总线速度（等待时钟数）控制

MSSPEED[6:1]	对应的时钟数
0	1
1	3
2	5
…	…
x	2x+1
…	…
62	125
63	127

只有当 I²C 模块工作在主机模式时，MSSPEED 参数设置的等待参数才有效。此等待参数主要用于主机模式的以下几个信号：

T_{SSTA}：起始信号的建立时间（Setup Time of START）

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

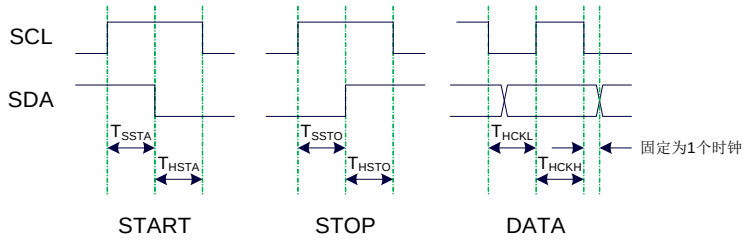
T_{SSTO}：停止信号的建立时间（Setup Time of STOP）

T_{HSTO}：停止信号的保持时间（Hold Time of STOP）

T_{HCKL}：时钟信号的低电平保持时间（Hold Time of SCL Low）

注意：

- 由于需要配合时钟同步机制，对于时钟信号的高电平保持时间（T_{HCKH}）至少为时钟信号的低电平保持时间（T_{HCKL}）的 1 倍长，而 T_{HCKH} 确切的长度取决于 SCL 端口的上拉速度。
- SDA 在 SCL 下降沿后的数据保持时间固定为 1 个时钟



I²C 主机控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-	-	MSCMD[2:0]		

EMSI：主机模式中断使能控制位

0：关闭主机模式的中断

1：允许主机模式的中断

MSCMD[3:0]：主机命令

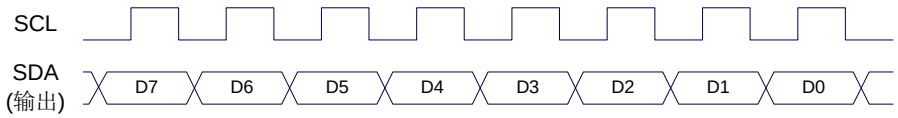
0000：待机，无动作。

0001：起始命令。

发送 START 信号。如果当前 I²C 控制器处于空闲状态，即 MSBUSY（I2CMSST.7）为 0 时，写此命令会使控制器进入忙状态，硬件自动将 MSBUSY 状态位置 1，并开始发送 START 信号；**若当前 I²C 控制器处于忙状态，写此命令可触发发送 START 信号**。发送 START 信号的波形如下图所示：

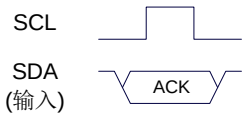
0010：发送数据命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 8 个时钟，并将 I2CTXD 寄存器里面数据按位送到 SDA 管脚上（先发送高位数据）。发送数据的波形如下图所示：



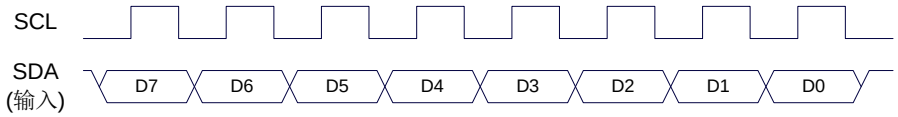
0011：接收 ACK 命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 1 个时钟，并将从 SDA 端口上读取的数据保存到 MSACKI（I2CMSST.1）。接收 ACK 的波形如下图所示：



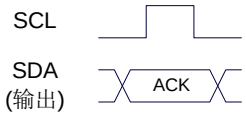
0100：接收数据命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 8 个时钟，并将从 SDA 端口上读取的数据依次左移到 I2CRXD 寄存器（先接收高位数据）。接收数据的波形如下图所示：



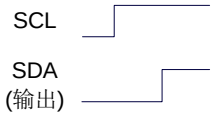
0101：发送 ACK 命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 1 个时钟，并将 MSACKO（I2CMSST.0）中的数据发送到 SDA 端口。发送 ACK 的波形如下图所示：



0110：停止命令。

发送 STOP 信号。写此命令后，I²C 总线控制器开始发送 STOP 信号。信号发送完成后，硬件自动将 MSBUSY 状态位清零。STOP 信号的波形如下图所示：



0111：保留。

1000：保留。

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。

I^2C 主机辅助控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSAUX	FE88H	-	-	-	-	-	-	-	WDTA

WDTA：主机模式时 I^2C 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，I^2C 控制器会自动触发“1010”命令，即自动发送数据并接收 ACK 信号。

I^2C 主机状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO

MSBUSY：主机模式时 I^2C 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 I^2C 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 I^2C 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。

18.3 I²C从机模式

I²C 从机控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLCR	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

ESTAI：从机模式时接收到 START 信号中断允许位

0：禁止从机模式时接收到 START 信号时发生中断

1：使能从机模式时接收到 START 信号时发生中断

ERXI：从机模式时接收到 1 字节数据后中断允许位

0：禁止从机模式时接收到数据后发生中断

1：使能从机模式时接收到 1 字节数据后发生中断

ETXI：从机模式时发送完成 1 字节数据后中断允许位

0：禁止从机模式时发送完成数据后发生中断

1：使能从机模式时发送完成 1 字节数据后发生中断

ESTOI：从机模式时接收到 STOP 信号中断允许位

0：禁止从机模式时接收到 STOP 信号时发生中断

1：使能从机模式时接收到 STOP 信号时发生中断

SLRST：复位从机模式

I²C 从机状态寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLST	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	-	SLACKI	SLACKO

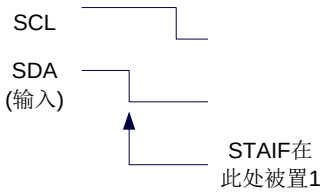
SLBUSY：从机模式时 I²C 控制器状态位（只读位）

0：控制器处于空闲状态

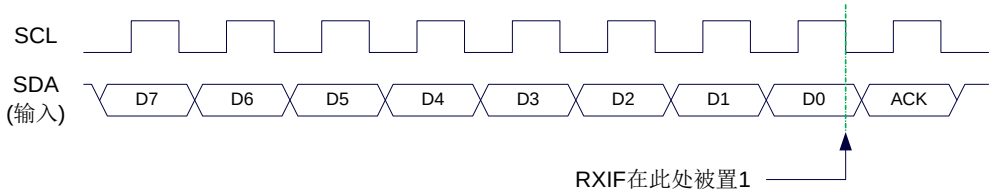
1：控制器处于忙碌状态

当 I²C 控制器处于从机模式时，在空闲状态下，接收到主机发送 START 信号后，控制器会继续检测之后的设备地址数据，若设备地址与当前 I2CSLADR 寄存器中所设置的从机地址像匹配时，控制器便进入到忙碌状态，忙碌状态会一直维持到成功接收到主机发送 STOP 信号，之后状态会再次恢复到空闲状态。

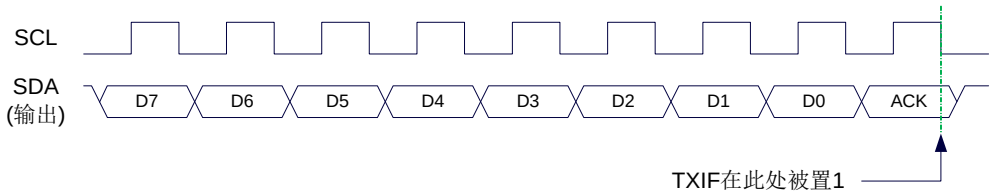
STAIF：从机模式时接收到 START 信号后的中断请求位。从机模式的 I²C 控制器接收到 START 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STAIF 位必须用软件清零。STAIF 被置 1 的时间点如下图所示：



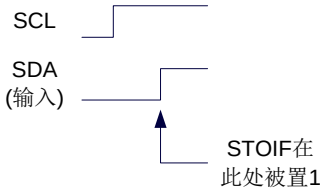
RXIF：从机模式时接收到 1 字节的数据后的中断请求位。从机模式的 I²C 控制器接收到 1 字节的数据后，在第 8 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 RXIF 位必须用软件清零。RXIF 被置 1 的时间点如下图所示：



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

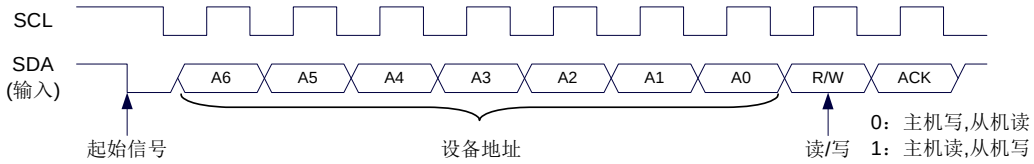


STOIF：从机模式时接收到 STOP 信号后的中断请求位。从机模式的 I²C 控制器接收到 STOP 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STOIF 位必须用软件清零。STOIF 被置 1 的时间点如下图所示：



SLACKI：从机模式时，接收到的 ACK 数据。

SLACKO：从机模式时，准备将要发送出去的 ACK 信号。



I²C 从机地址寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLADR	FE85H	SLADR[6:0]							MA

SLADR[6:0]：从机设备地址

当 I²C 控制器处于从机模式时，控制器在接收到 START 信号后，会继续检测接下来主机发送出的设备地址数据以及读/写信号。当主机发送出的设备地址与 SLADR[6:0]中所设置的从机设备地址相匹配时，控制器才会向 CPU 发出中断求，请求 CPU 处理 I²C 事件；否则若设备地址不匹配，I²C 控制器继续继续监控，等待下一个起始信号，对下一个设备地址继续匹配。

MA：从机设备地址匹配控制

0：设备地址必须与 SLADR[6:0]继续匹配

1：忽略 SLADR 中的设置，匹配所有的设备地址

I^2C 数据寄存器

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

18.4 范例程序

18.4.1 I²C主机模式访问AT24C256（中断方式）

汇编代码

```
P_SW2       DATA    0BAH

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

SDA         BIT     P1.4
SCL         BIT     P1.5

BUSY        BIT     20H.0

            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,#3FH
        MOV     P_SW2,#80H
```

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

CALL    START                   ;发送起始命令
MOV     A,#0A0H
CALL    SENDDATA                ;发送设备地址+写命令
CALL    RECVACK
MOV     A,#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
```

C 语言代码

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

```
sfr         P_SW2           =   0xba;

#define     I2CCFG              (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR             (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST             (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR             (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST             (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR            (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD              (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD              (*(unsigned char volatile xdata *)0xfe87)

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

bit         busy;

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

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

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

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

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

void SendACK()
```

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

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

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

void Delay()
{
    int i;

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

void main()
{
    P_SW2 = 0x80;

    I2CCFG = 0xe0;                  //使能 I2C 主机模式
    I2CMSST = 0x00;
    EA = 1;

    Start();                        //发送起始命令
    SendData(0xa0);                 //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                 //发送存储地址高字节
    RecvACK();
    SendData(0x00);                 //发送存储地址低字节
    RecvACK();
    SendData(0x12);                 //写测试数据 1
    RecvACK();
    SendData(0x78);                 //写测试数据 2
    RecvACK();
    Stop();                         //发送停止命令

    Delay();                        //等待设备写数据

    Start();                        //发送起始命令
```

```
    SendData(0xa0);                 //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                 //发送存储地址高字节
    RecvACK();
    SendData(0x00);                 //发送存储地址低字节
    RecvACK();
    Start();                        //发送起始命令
    SendData(0xa1);                 //发送设备地址+读命令
    RecvACK();
    P0 = RecvData();                //读取数据 1
    SendACK();
    P2 = RecvData();                //读取数据 2
    SendNAK();
    Stop();                         //发送停止命令

    P_SW2 = 0x00;

    while (1);
}
```

18.4.2 I²C主机模式访问AT24C256（查询方式）

汇编代码

```
P_SW2       DATA    0BAH

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

SDA         BIT     P1.4
SCL         BIT     P1.5

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
START:
            MOV     A,#00000001B        ;发送 START 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDDATA:
            MOV     DPTR,#I2CTXD        ;写数据到数据缓冲区
            MOVX    @DPTR,A
            MOV     A,#00000010B        ;发送 SEND 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVACK:
            MOV     A,#00000011B        ;发送读 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
```

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

DELAY:
        MOV     R0,#0
        MOV     R1,#0
DELAY1:
        NOP
        NOP
        NOP
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

MAIN:
        MOV     SP,#3FH
        MOV     P_SW2,#80H

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

```
        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
```

C 语言代码

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

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
```

```
#define     I2CSLADR        (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD          (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD          (*(unsigned char volatile xdata *)0xfe87)

sbit        SDA         =   P1^4;
sbit        SCL         =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                 //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                   //写数据到数据缓冲区
    I2CMSCR = 0x02;                 //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                 //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                 //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                 //设置 ACK 信号
    I2CMSCR = 0x05;                 //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                 //设置 NAK 信号
    I2CMSCR = 0x05;                 //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                 //发送 STOP 命令
    Wait();
}
```

```
void Delay()
{
	int i;

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

void main()
{
	P_SW2 = 0x80;

	I2CCFG = 0xe0;			//使能 I2C 主机模式
	I2CMSST = 0x00;

	Start();			//发送起始命令
	SendData(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);
}
```

18.4.3 I²C主机模式访问PCF8563

汇编代码

```
P_SW2       DATA    0BAH

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

SDA         BIT     P1.4
SCL         BIT     P1.5

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
START:
            MOV     A,#00000001B        ;发送 START 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDDATA:
            MOV     DPTR,#I2CTXD        ;写数据到数据缓冲区
            MOVX    @DPTR,A
            MOV     A,#00000010B        ;发送 SEND 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVACK:
            MOV     A,#00000011B        ;发送读 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVDATA:
            MOV     A,#00000100B        ;发送 RECV 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            CALL    WAIT
            MOV     DPTR,#I2CRXD        ;从数据缓冲区读取数据
            MOVX    A,@DPTR
            RET
SENDACK:
            MOV     A,#00000000B        ;设置 ACK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDNAK:
            MOV     A,#00000001B        ;设置 NAK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
```

```
                MOV     A,#00000101B        ;发送 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
STOP:
                MOV     A,#00000110B        ;发送 STOP 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
WAIT:
                MOV     DPTR,#I2CMSST       ;清中断标志
                MOVX    A,@DPTR
                JNB     ACC.6,WAIT
                ANL     A,#NOT 40H
                MOVX    @DPTR,A
                RET

DELAY:
                MOV     R0,#0
                MOV     R1,#0
DELAY1:
                NOP
                NOP
                NOP
                NOP
                DJNZ    R1,DELAY1
                DJNZ    R0,DELAY1
                RET

MAIN:
                MOV     SP,#3FH
                MOV     P_SW2,#80H

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

                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
```

C 语言代码

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

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sbit    SDA     =   P1^4;
sbit    SCL     =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                 //发送 START 命令
    Wait();
}

void SendData(char dat)
```

```
{
    I2CTXD = dat;                   //写数据到数据缓冲区
    I2CMSCR = 0x02;                 //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                 //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                 //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                 //设置 ACK 信号
    I2CMSCR = 0x05;                 //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                 //设置 NAK 信号
    I2CMSCR = 0x05;                 //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                 //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;

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

void main()
{
    P_SW2 = 0x80;

    I2CCFG = 0xe0;                  //使能 I2C 主机模式
    I2CMSST = 0x00;
```

```
    Start();                    //发送起始命令
    SendData(0xa2);             //发送设备地址+写命令
    RecvACK();
    SendData(0x02);             //发送存储地址
    RecvACK();
    SendData(0x00);             //设置秒值
    RecvACK();
    SendData(0x00);             //设置分钟值
    RecvACK();
    SendData(0x12);             //设置小时值
    RecvACK();
    Stop();                     //发送停止命令

    while (1)
    {
        Start();                //发送起始命令
        SendData(0xa2);         //发送设备地址+写命令
        RecvACK();
        SendData(0x02);         //发送存储地址
        RecvACK();
        Start();                //发送起始命令
        SendData(0xa3);         //发送设备地址+读命令
        RecvACK();
        P0 = RecvData();        //读取秒值
        SendACK();
        P2 = RecvData();        //读取分钟值
        SendACK();
        P3 = RecvData();        //读取小时值
        SendNAK();
        Stop();                 //发送停止命令

        Delay();
    }
}
```

18.4.4 I²C从机模式（中断方式）

汇编代码

```
P_SW2       DATA    0BAH

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

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

ADDR        DATA    21H

            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
                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     P_SW2,#80H

            MOV     A,#10000001B        ;使能 I2C 从机模式
            MOV     DPTR,#I2CCFG
            MOVX    @DPTR,A
            MOV     A,#01011010B        ;设置从机设备地址为 5A
            MOV     DPTR,#I2CSLADR
            MOVX    @DPTR,A
            MOV     A,#00000000B
            MOV     DPTR,#I2CSLST
            MOVX    @DPTR,A
            MOV     A,#01111000B        ;使能从机模式中断
            MOV     DPTR,#I2CSLCR
            MOVX    @DPTR,A

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

            SETB    EA

            SJMP    $

            END
```

C 语言代码

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

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sbit    SDA         =   P1^4;
sbit    SCL         =   P1^5;

bit                 isda;           //设备地址标志
```

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

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

    if (I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40;           //处理 START 事件
    }
    else if (I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20;           //处理 RECV 事件
        if (isda)
        {
            isda = 0;               //处理 RECV 事件（RECV DEVICE ADDR）
        }
        else if (isma)
        {
            isma = 0;               //处理 RECV 事件（RECV MEMORY ADDR）
            addr = I2CRXD;
            I2CTXD = buffer[addr];
        }
        else
        {
            buffer[addr++] = I2CRXD; //处理 RECV 事件（RECV DATA）
        }
    }
    else if (I2CSLST & 0x10)
    {
        I2CSLST &= ~0x10;           //处理 SEND 事件
        if (I2CSLST & 0x02)
        {
            I2CTXD = 0xff;          //接收到 NAK 则停止读取数据
        }
        else
        {
            I2CTXD = buffer[++addr]; //接收到 ACK 则继续读取数据
        }
    }
    else if (I2CSLST & 0x08)
    {
        I2CSLST &= ~0x08;           //处理 STOP 事件
        isda = 1;
        isma = 1;
    }

    _pop_(P_SW2);
}

void main()
{
    P_SW2 = 0x80;

    I2CCFG = 0x81;                  //使能 I2C 从机模式
    I2CSLADR = 0x5a;                //设置从机设备地址为 5A
```

```
    I2CSLST = 0x00;
    I2CSLCR = 0x78;                 //使能从机模式中断
    EA = 1;

    isda = 1;                       //用户变量初始化
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1);
}
```

18.4.5 I²C从机模式（查询方式）

汇编代码

```
P_SW2       DATA    0BAH

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

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

ADDR        DATA    21H

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     P_SW2,#80H

            MOV     A,#10000001B    ;使能 I2C 从机模式
            MOV     DPTR,#I2CCFG
            MOVX    @DPTR,A
            MOV     A,#01011010B    ;设置从机设备地址为 5A
            MOV     DPTR,#I2CSLADR
            MOVX    @DPTR,A
            MOV     A,#00000000B
            MOV     DPTR,#I2CSLST
            MOVX    @DPTR,A
            MOV     A,#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
                JMP     LOOP
RXIF:
                ANL     A,#NOT 20H              ;处理 RECV 事件
                MOVX    @DPTR,A
                MOV     DPTR,#I2CRXD
                MOVX    A,@DPTR
                JBC     ISDA,RXDA
                JBC     ISMA,RXMA
                MOV     R0,ADDR                 ;处理 RECV 事件（RECV DATA）
                MOVX    @R0,A
                INC     ADDR
                JMP     LOOP
RXDA:
                JMP     LOOP                    ;处理 RECV 事件（RECV DEVICE ADDR）
RXMA:
                MOV     ADDR,A                  ;处理 RECV 事件（RECV MEMORY ADDR）
                MOV     R0,A
                MOVX    A,@R0
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     LOOP
TXIF:
                ANL     A,#NOT 10H              ;处理 SEND 事件
                MOVX    @DPTR,A
                JB      ACC.1,RXNAK
                INC     ADDR
                MOV     R0,ADDR
                MOVX    A,@R0
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     LOOP
RXNAK:
                MOVX    A,#0FFH
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     LOOP
STOPIF:
                ANL     A,#NOT 08H              ;处理 STOP 事件
                MOVX    @DPTR,A
                SETB    ISDA
                SETB    ISMA
                JMP     LOOP

                END
```

C 语言代码

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

sfr         P_SW2           =   0xba;

#define     I2CCFG              (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR             (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST             (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR             (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST             (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR            (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD              (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD              (*(unsigned char volatile xdata *)0xfe87)

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

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

void main()
{
    P_SW2 = 0x80;

    I2CCFG = 0x81;                      //使能 I2C 从机模式
    I2CSLADR = 0x5a;                    //设置从机设备地址为 5A
    I2CSLST = 0x00;
    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
        }
        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;
        }
    }
}
```

18.4.6 测试I²C从机模式代码的主机代码

汇编代码

```
P_SW2       DATA    0BAH

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

SDA         BIT     P1.4
SCL         BIT     P1.5

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
START:
            MOV     A,#00000001B        ;发送 START 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDDATA:
            MOV     DPTR,#I2CTXD        ;写数据到数据缓冲区
            MOVX    @DPTR,A
            MOV     A,#00000010B        ;发送 SEND 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
```

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

DELAY:
        MOV     R0,#0
        MOV     R1,#0
DELAY1:
        NOP
        NOP
        NOP
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

MAIN:
        MOV     SP,#3FH
        MOV     P_SW2,#80H

        MOV     A,#11100000B        ;设置 I2C 模块为主机模式
```

```
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A

        CALL    START               ;发送起始命令
        MOV     A,#5AH              ;从机地址为5A
        CALL    SENDDATA            ;发送设备地址+写命令
        CALL    RECVACK
        MOV     A,#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+写命令
        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+读命令
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA            ;读取数据1
        MOV     P0,A
        CALL    SENDACK
        CALL    RECVDATA            ;读取数据2
        MOV     P2,A
        CALL    SENDNAK
        CALL    STOP                ;发送停止命令

        JMP     $

        END
```

C 语言代码

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

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
```

```
#define     I2CTXD          (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD          (*(unsigned char volatile xdata *)0xfe87)

sbit        SDA         =   P1^4;
sbit        SCL         =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                 //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                   //写数据到数据缓冲区
    I2CMSCR = 0x02;                 //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                 //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                 //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                 //设置 ACK 信号
    I2CMSCR = 0x05;                 //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                 //设置 NAK 信号
    I2CMSCR = 0x05;                 //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                 //发送 STOP 命令
    Wait();
}

void Delay()
```

```
{
    int i;

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

void main()
{
    P_SW2 = 0x80;

    I2CCFG = 0xe0;                  //使能 I2C 主机模式
    I2CMSST = 0x00;

    Start();                        //发送起始命令
    SendData(0x5a);                 //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                 //发送存储地址
    RecvACK();
    SendData(0x12);                 //写测试数据 1
    RecvACK();
    SendData(0x78);                 //写测试数据 2
    RecvACK();
    Stop();                         //发送停止命令

    Start();                        //发送起始命令
    SendData(0x5a);                 //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                 //发送存储地址高字节
    RecvACK();
    Start();                        //发送起始命令
    SendData(0x5b);                 //发送设备地址+读命令
    RecvACK();
    P0 = RecvData();                //读取数据 1
    SendACK();
    P2 = RecvData();                //读取数据 2
    SendNAK();
    Stop();                         //发送停止命令

    P_SW2 = 0x00;

    while (1);
}
```

19 高级PWM

STC8H 系列的单片机内部集成了 PWM1 和 PWM2 两组高级 PWM。PWM1 有 4 个通道，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；PWM2 有 4 个通道，每个通道也可独立实现 PWM 输出、捕获和比较功能，但 PWM2 不能输出带死区的互补对称 PWM。

PWM2 与 PWM1 唯一的区别是 PWM1 可输出带死区的互补对称 PWM，而 PWM2 则只能输出单端的 PWM，其他功能完全相同。下面关于高级 PWM 的介绍只以 PWM1 为例进行说明。

19.1 简介

PWM1 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。

PWM1 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 PWM2 或者外部信号（外部时钟，复位信号，触发和使能信号）同步

PWM1 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。

PWM1 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。

19.2 主要特性

PWM1 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（ETR）
- 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）

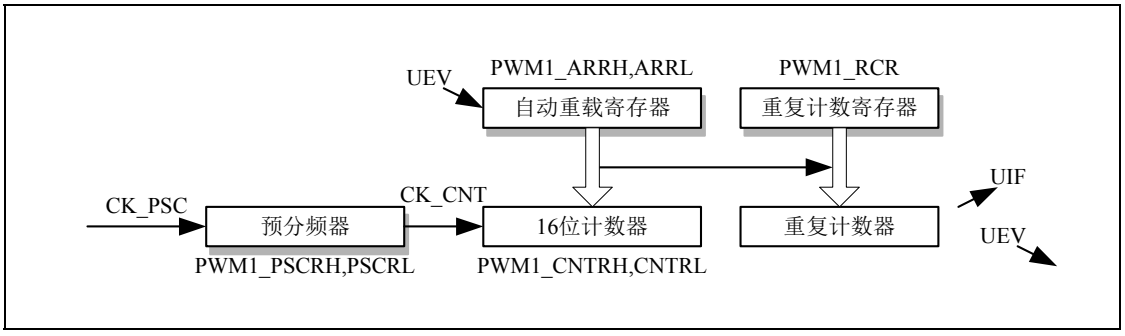
- 输入捕获
- 输出比较
- 刹车信号输入

19.3 时基单元

PWM1 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

PWM1 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（PWM1_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（PWM1_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 PWM1_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（PWM1_ARR）将写入影子寄存器中，并且 PWM1_PSCR 寄存器中的值将写入预分频器中。

置位 PWM1_CR1 寄存器的 UDIS 位将禁止更新事件（UEV）。

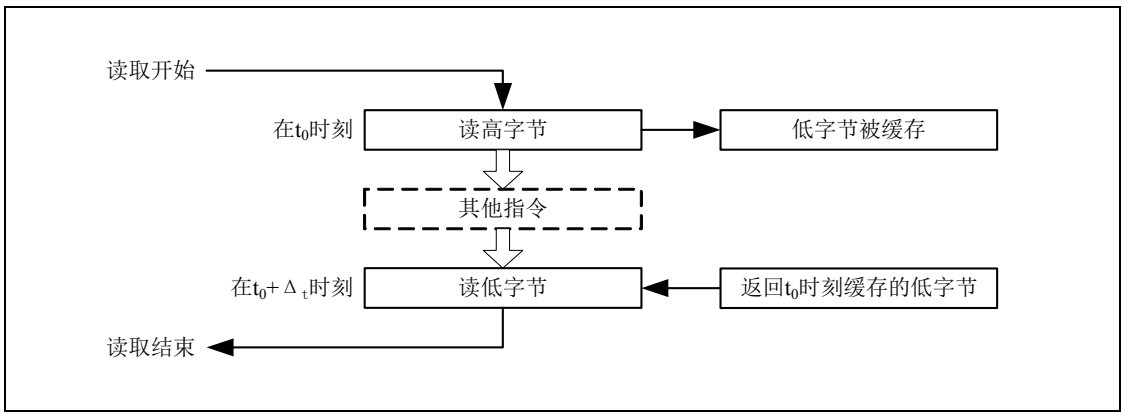
预分频器的输出 CK_CNT 驱动计数器，而 CK_CNT 仅在 IM1_CR1 寄存器的计数器使能位（CEN）被置位时才有效。

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。

19.3.1 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 PWM1_CNTRH 和 PWM1_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。



19.3.2 16 位PWM1_ARR寄存器的写操作

预装载寄存器中的值将写入 16 位的 PWM1_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。

19.3.3 预分频器

预分频器的实现：

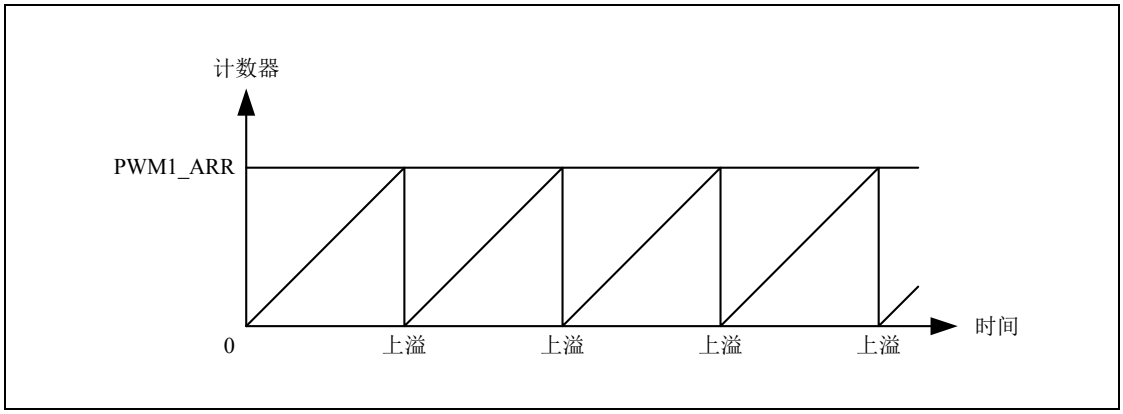
PWM1 的预分频器基于一个由 16 位寄存器（PWM1_PSCR）控制的 16 位计数器。由于这个控制寄存器带有缓冲器，因此它能够在运行时被改变。预分频器可以将计数器的时钟频率按 1 到 65536 之间的任意值分频。预分频器的值由预装载寄存器写入，保存了当前使用值的影子寄存器在低字节写入时被载入。由于需两次单独的写操作来写 16 位寄存器，因此必须保证高字节先写入。新的预分频器的值在下一次更新事件到来时被采用。对 PWM1_PSCR 寄存器的读操作通过预装载寄存器完成。

计数器的频率计算公式：$f_{CK\_CNT} = f_{CK\_PSC} / (PSCR[15:0] + 1)$

19.3.4 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（PWM1_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 PWM1_CR1 寄存器的 UDIS 位是 0，将会产生一个更新事件（UEV）。

向上计数模式的计数器



通过软件方式或者通过使用触发控制器置位 PWM1_EGR 寄存器的 UG 位同样也可以产生一个更新事件。

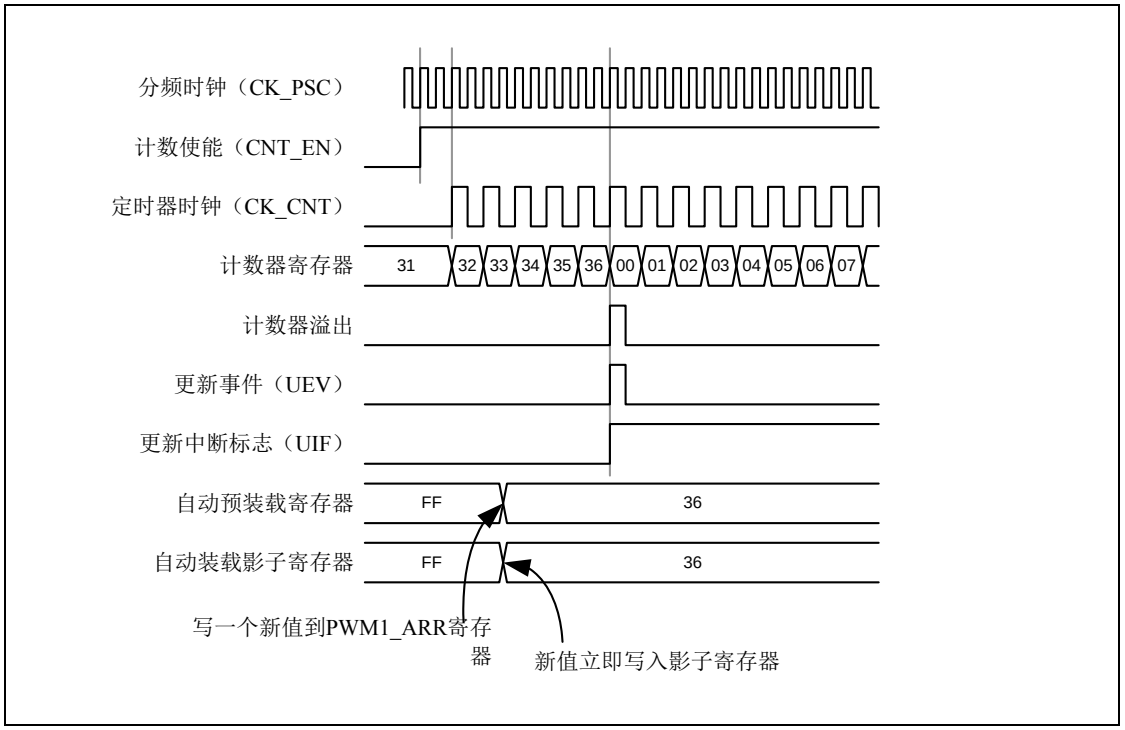
使用软件置位 PWM1_CR1 寄存器的 UDIS 位，可以禁止更新事件，这样可以避免在更新预装载寄存器时更新影子寄存器。在 UDIS 位被清除之前，将不产生更新事件。但是在应该产生更新事件时，计数器仍会被清 0，同时预分频器的计数也被清 0（但预分频器的数值不变）。此外，如果设置了 PWM1_CR1 寄存器中的 URS 位（选择更新请求），设置 UG 位将产生一个更新事件 UEV，但硬件不设置 UIF 标志（即不产生中断请求）。这是为了避免在捕获模式下清除计数器时，同时产生更新和捕获中断。

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（PWM1_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（PWM1_ARR）。
- 预分频器的缓存器被置入预装载寄存器的值（PWM1_PSC 寄存器的内容）。

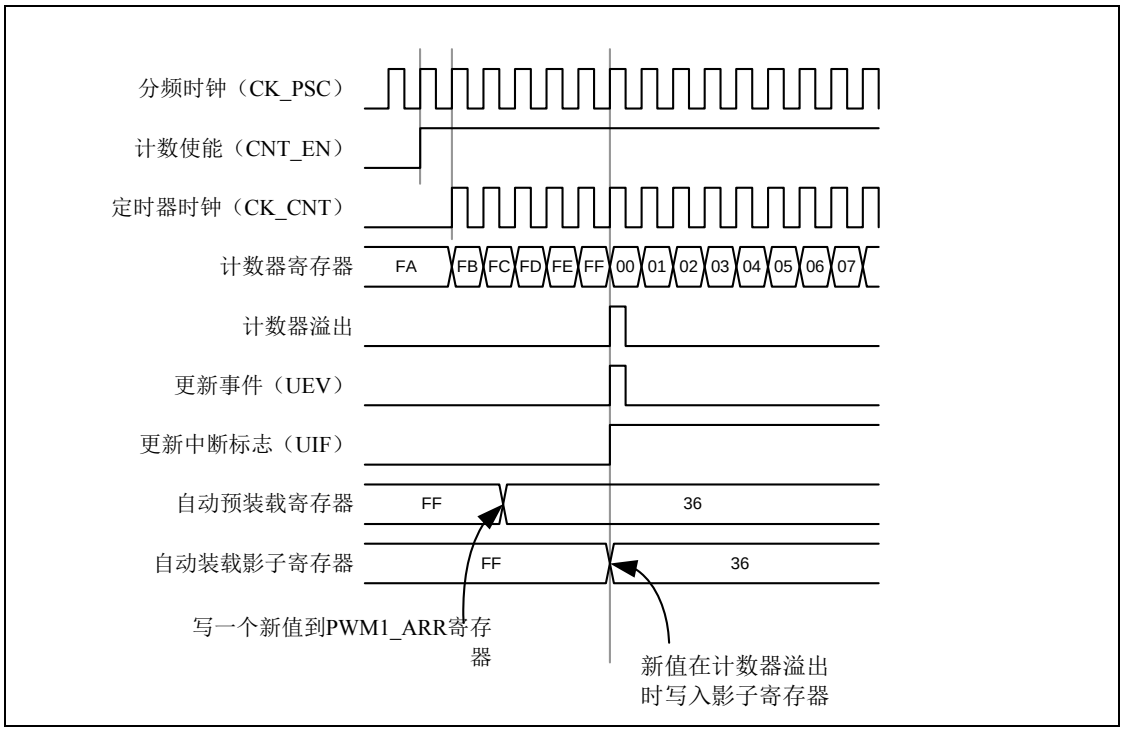
下图给出一些例子，说明当 PWM1_ARR=0x36 时，计数器在不同时钟频率下的动作。图中预分频为 2，因此计数器的时钟（CK_CNT）频率是预分频时钟（CK_PSC）频率的一半。图中禁止了自动装载功能（ARPE=0），所以在计数器达到 0x36 时，计数器溢出，影子寄存器立刻被更新，同时产生一个更新事件。

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。

ARPE=1(PWM1_ARR 预装载) 预分频为 1 时的计数器更新：

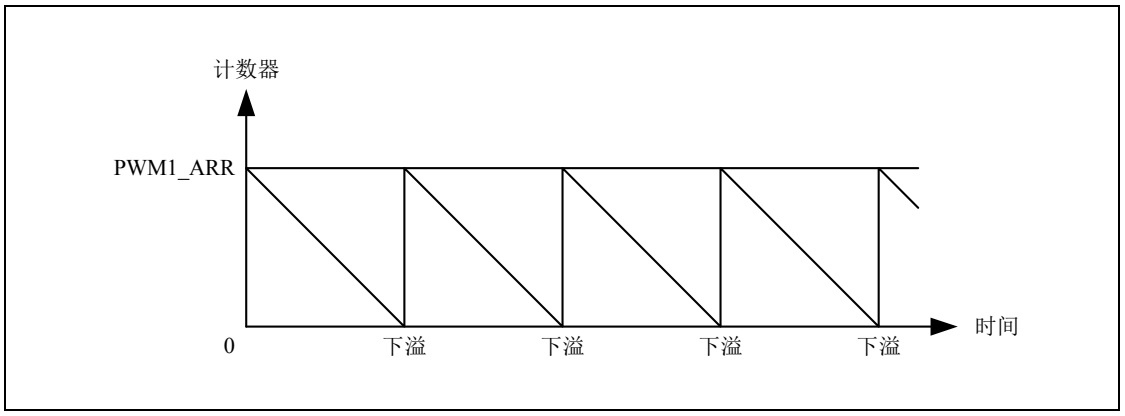


19.3.5 向下计数模式

在向下模式中，计数器从自动装载的值（PWM1_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 PWM1_CR1 寄存器的 UDIS 位被清除，还会产生一个更新事件（UEV）。

向下计数模式的计数器



通过软件方式或者通过使用触发控制器置位 PWM1_EGR 寄存器的 UG 位同样也可以产生一个更新事件。

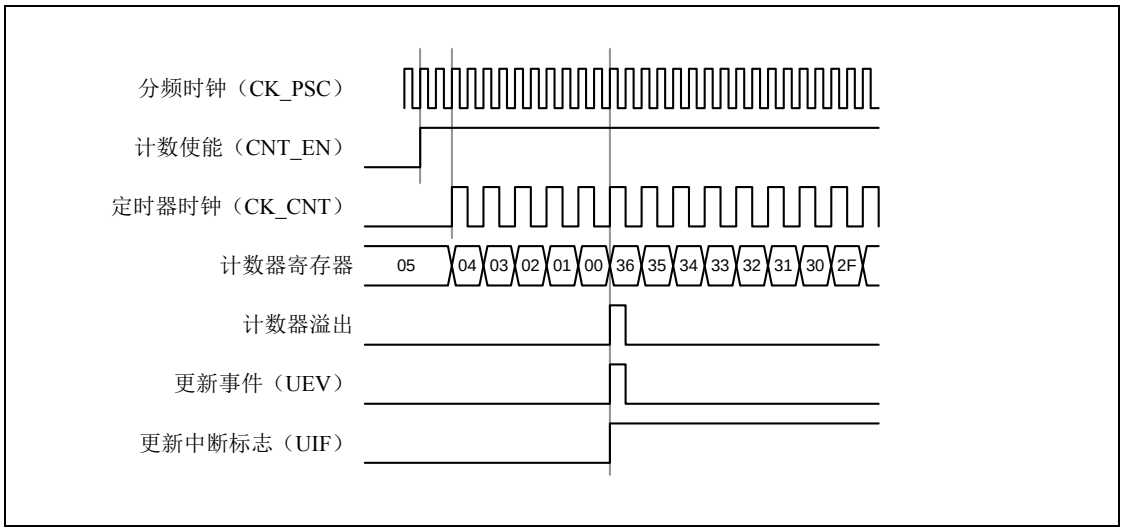
置位 PWM1_CR1 寄存器的 UDIS 位可以禁止 UEV 事件。这样可以避免在更新预装载寄存器时更新影子寄存器。因此 UDIS 位清除之前不会产生更新事件。然而，计数器仍会从当前自动加载值重新开始计数，并且预分频器的计数器重新从 0 开始（但预分频器不能被修改）。此外，如果设置了 PWM1_CR1 寄存器中的 URS 位（选择更新请求），设置 UG 位将产生一个更新事件 UEV 但不设置 UIF 标志（因此不产生中断），这是为了避免在发生捕获事件并清除计数器时，同时产生更新和捕获中断。

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（PWM1_SR 寄存器的 UIF 位）：

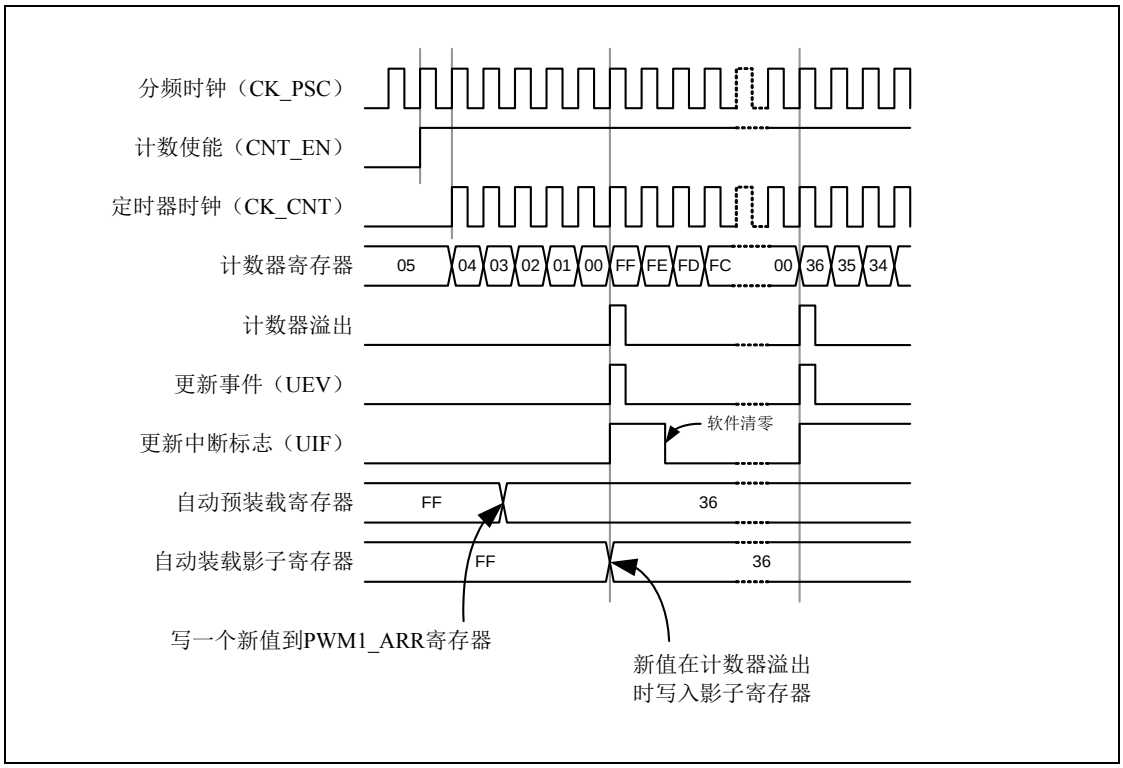
- 自动装载影子寄存器被重新置入预装载寄存器的值（PWM1_ARR）。
- 预分频器的缓存器被置入预装载寄存器的值（PWM1_PSC 寄存器的内容）。

以下是一些当 PWM1_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。

ARPE=0（ARR 不预装载），预分频为 2 时的计数器更新：



ARPE=1（ARR 预装载），预分频为 1 时的计数器更新

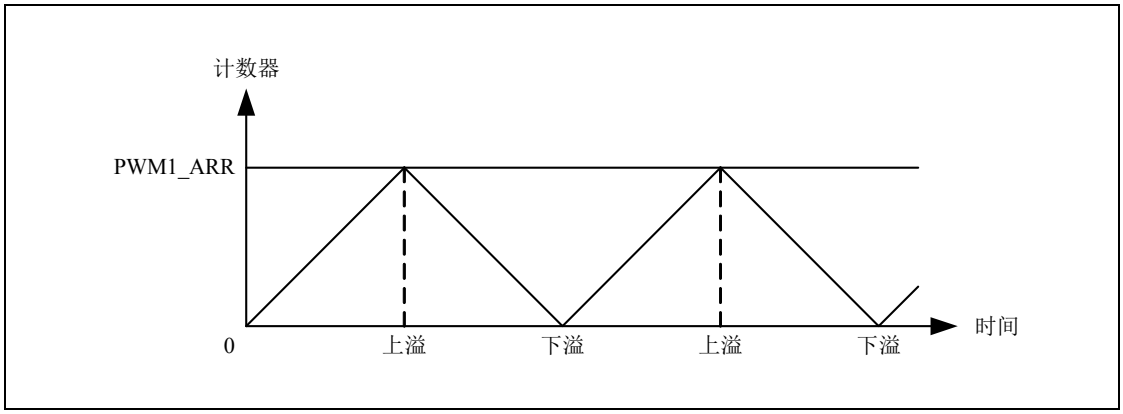


19.3.6 中间对齐模式（向上/向下计数）

在中央对齐模式，计数器从 0 开始计数到自动加载的值（PWM1_ARR 寄存器）-1，产生一个计数器上溢事件，然后向下计数到 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 PWM1_CR1 中的 DIR 方向位。它由硬件更新并指示当前的计数方向。

中央对齐模式的计数器



如果定时器带有重复计数器，在重复了指定次数（PWM1_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 PWM1_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 PWM1_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 PWM1_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。

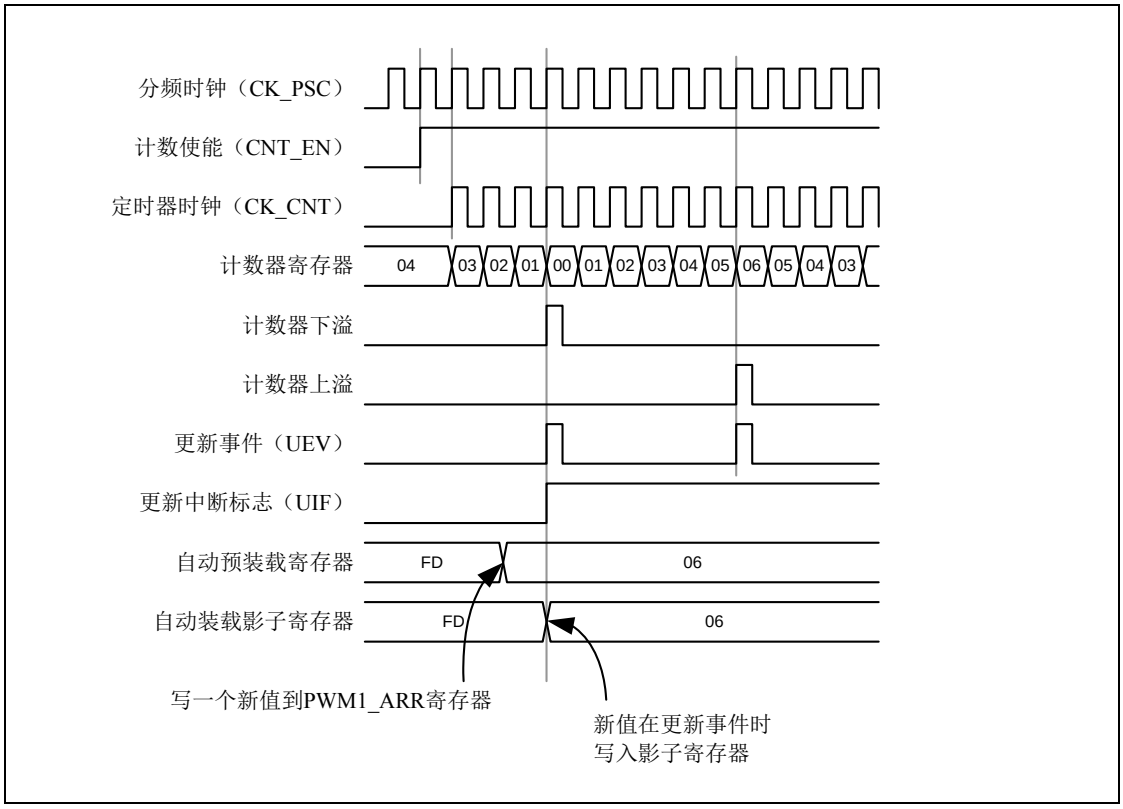
当发生更新事件时，所有的寄存器都被更新，硬件依据 URS 位更新标志位（PWM1_SR 寄存器中的 UIF 位）：

- 预分频器的缓存器被加载为预装载（PWM1_PSC 寄存器）的值。
- 当前的自动加载寄存器被更新为预装载值（PWM1_ARR 寄存器中的内容）。

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，PWM1_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 PWM1_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（PWM1_CNT>PWM1_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 PWM1_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 PWM1_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。

19.3.7 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（PWM1_ARR 自动重载入寄存器，PWM1_PSC 预装载寄存器，还有在比较模式下的捕获/比较寄存器 PWM1_CCRx），N 是 PWM1_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：

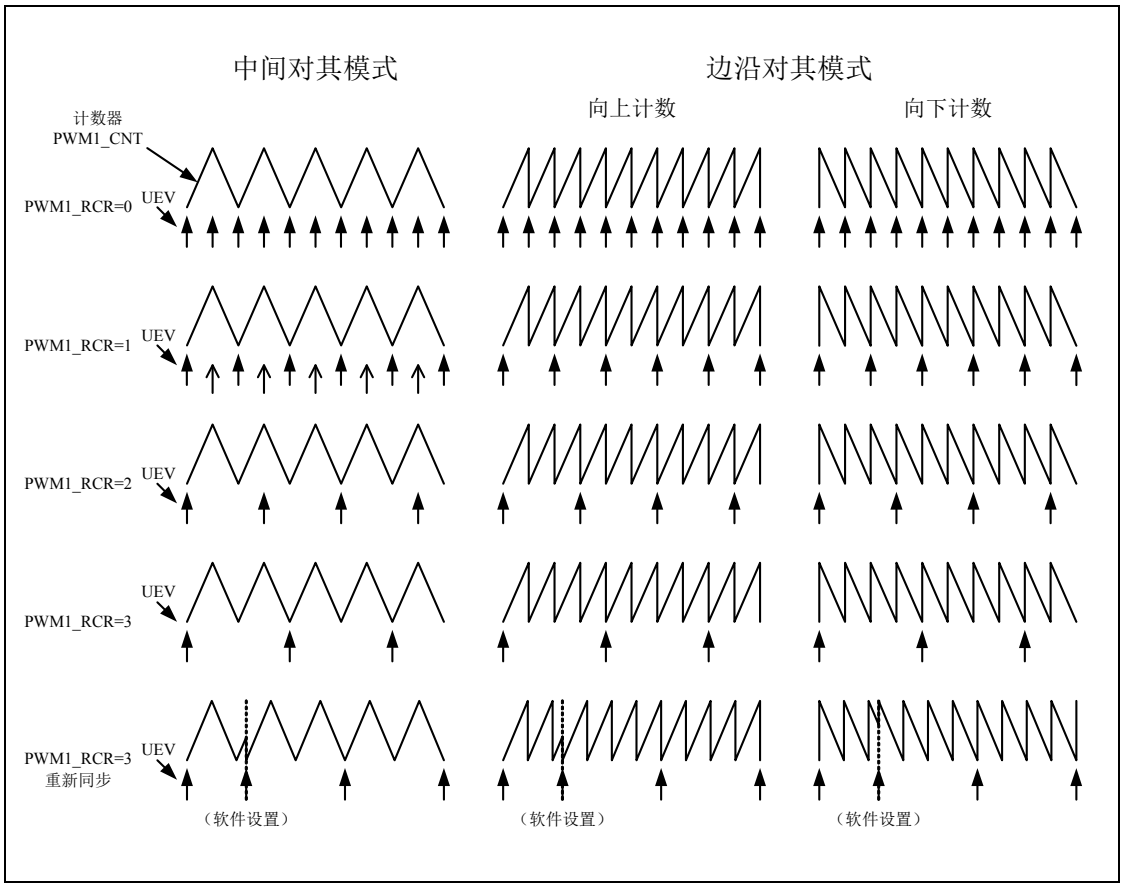
- 向上计数模式下每次计数器向上溢出时
- 向下计数模式下每次计数器向下溢出时

- 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*tCK_PSC。

重复计数器是自动加载的，重复速率由 PWM1_RCR 寄存器的值定义。当更新事件由软件产生（通过设置 PWM1_EGR 中的 UG 位）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 PWM1_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 PWM1_RCR 的寄存器设置



19.4 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，

19.4.1 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：

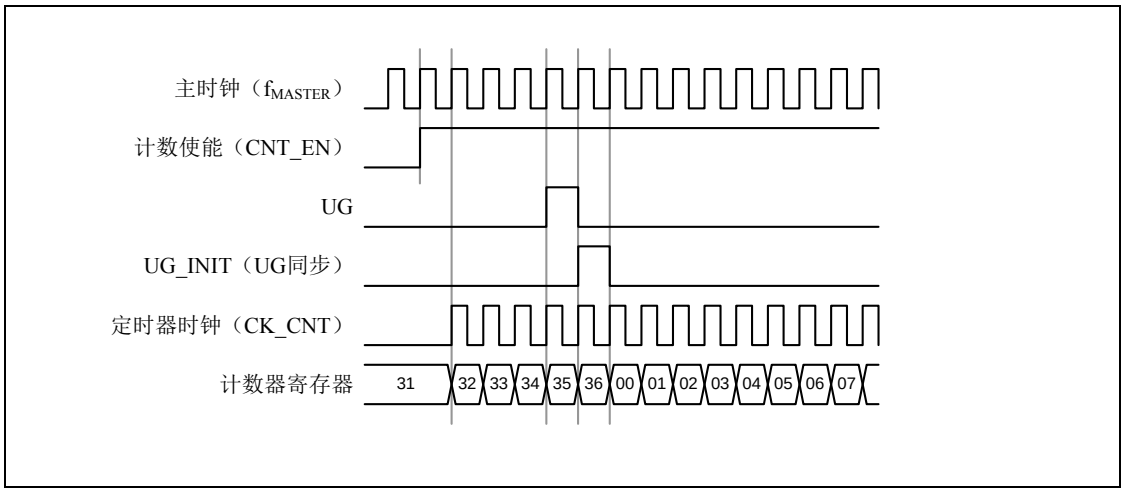
- 内部时钟（fMASTER）
- 外部时钟模式 1：外部时钟输入（TIx）
- 外部时钟模式 2：外部触发输入 ETR
- 内部触发输入（ITRx）：使用一个定时器做为另一个定时器的预分频时钟。

19.4.2 内部时钟源（fMASTER）

如果同时禁止了时钟/触发模式控制器和外部触发输入（PWM1_SMCR 寄存器的 SMS=000，PWM1_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。

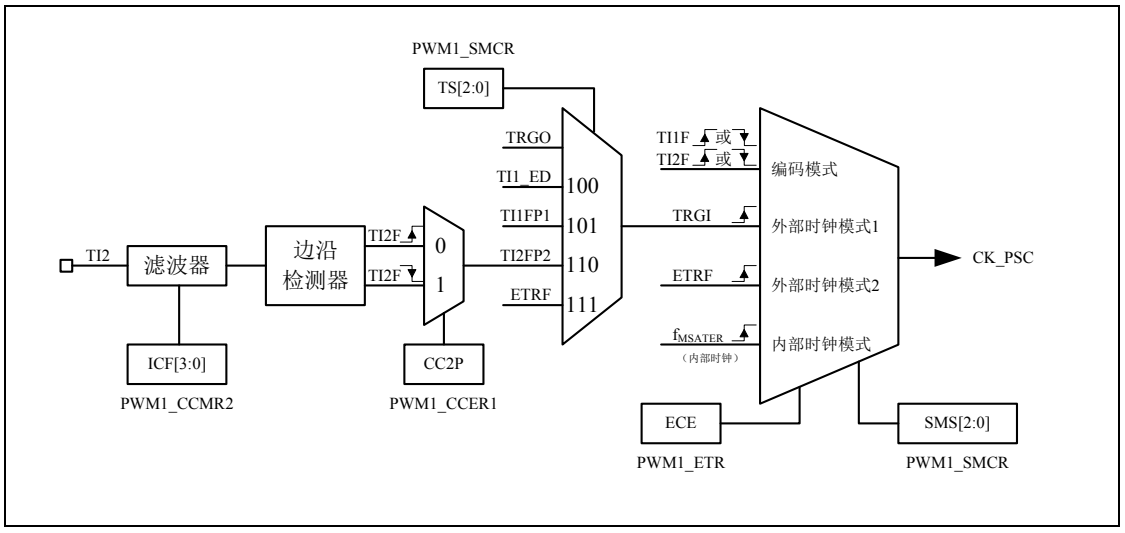
普通模式下的控制电路，f_{MASTER} 分频因子为 1



19.4.3 外部时钟源模式 1

当 PWM1_SMCR 寄存器的 SMS=111 时，此模式被选中。计数器可以在选定输入端的每个上升沿或下降沿计数。

TI2 外部时钟连接例子



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：

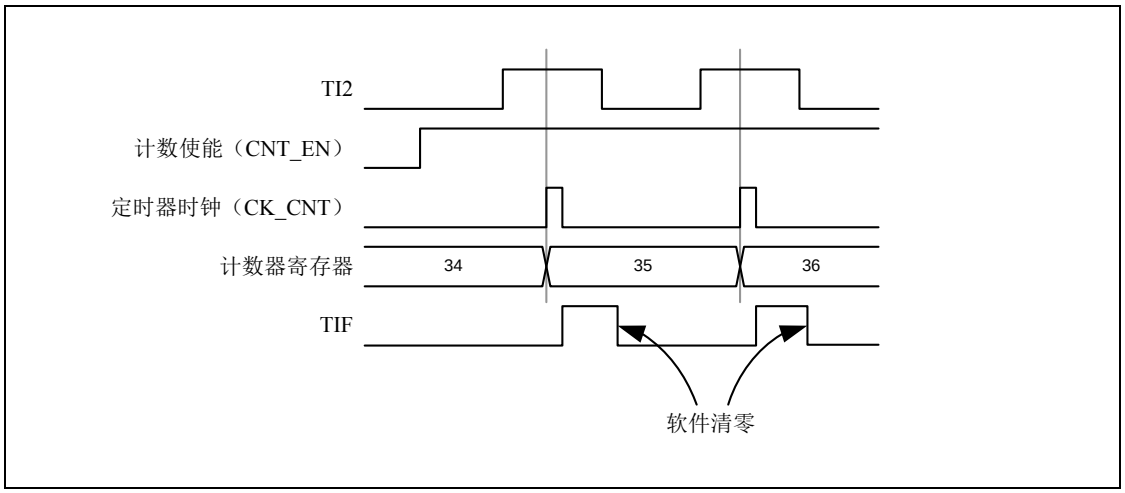
1. 配置 PWM1_CCMR2 寄存器的 CC2S=01，使用通道 2 检测 TI2 输入的上升沿

2. 配置 PWM1_CCMR2 寄存器的 IC2F[3:0]位，选择输入滤波器带宽（如果不需要滤波器，保持 IC2F=0000）注：捕获预分频器不用作触发，所以不需要对它进行配置，同样也不需要配置 TI2S 位，他们仅用来选择输入捕获源。
3. 配置 PWM1_CCER1 寄存器的 CC2P=0，选定上升沿极性
4. 配置 PWM1_SMCR 寄存器的 SMS=111，配置计数器使用外部时钟模式 1
5. 配置 PWM1_SMCR 寄存器的 TS=110，选定 TI2 作为输入源
6. 设置 PWM1_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（PWM1_SR1 寄存器的 TIF 位）被置 1，如果使能了中断（在 PWM1_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。

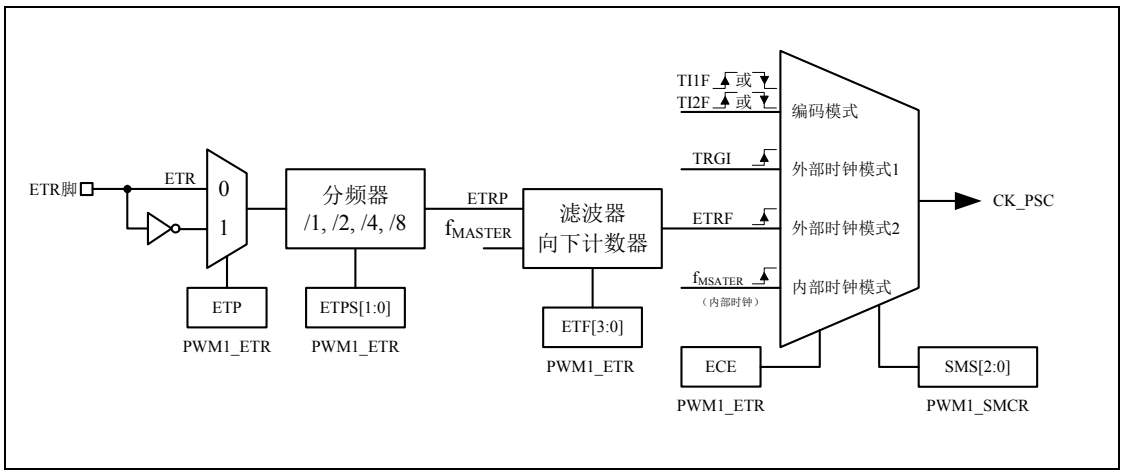
外部时钟模式 1 下的控制电路



19.4.4 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 PWM1_ETR 寄存器的 ECE 位写 1，即可选定此模式。

外部触发输入的总体框图：

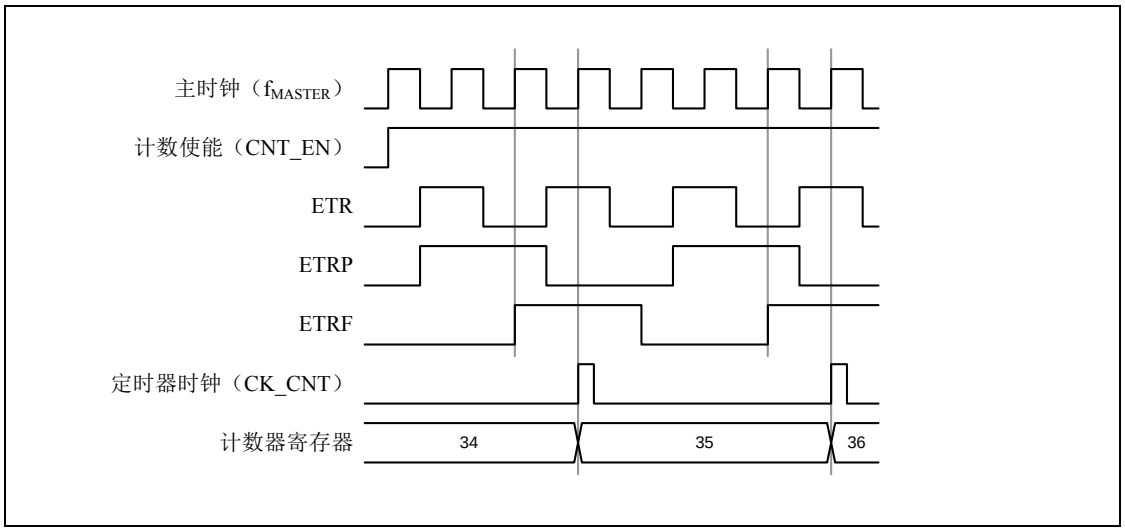


例如，要配置计数器在 ETR 信号的每 2 个上升沿时向上计数一次，需使用下列步骤：

1. 本例中不需要滤波器，配置 PWM1_ETR 寄存器的 ETF[3:0]=0000
2. 设置预分频器，配置 PWM1_ETR 寄存器的 ETPS[1:0]=01
3. 选择 ETR 的上升沿检测，配置 PWM1_ETR 寄存器的 ETP=0
4. 开启外部时钟模式 2，配置 PWM1_ETR 寄存器中的 ECE=1
5. 启动计数器，写 PWM1_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



19.4.5 触发同步

计数器允许四种触发输入：

- ETR
- TI1
- TI2
- TRGO

PWM1 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。

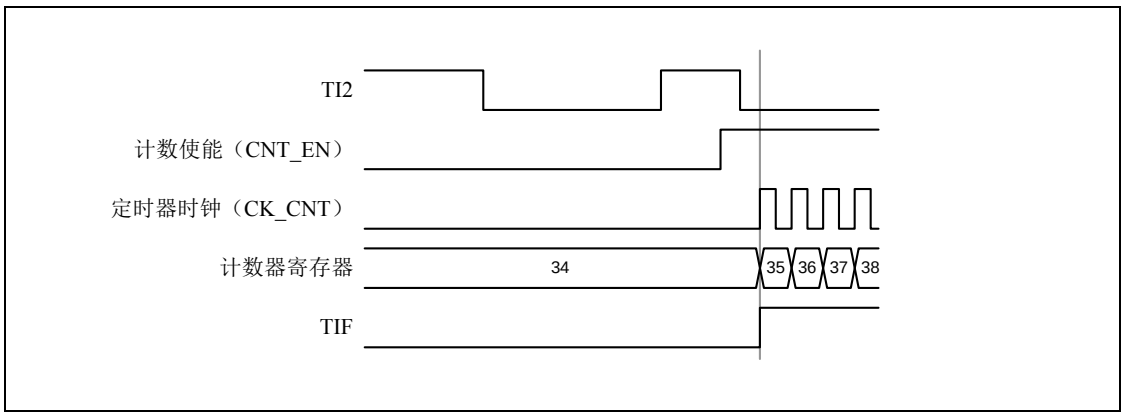
在下面的例子中，计数器在 TI2 输入的上升沿开始向上计数：

1. 配置通道 2 检测 TI2 的上升沿。配置输入滤波器带宽（本例中，不需要任何滤波器，保持 IC2F=0000）。触发操作中不使用捕获预分频器，不需要配置。CC2S 位仅用于选择输入捕获源，也不需要配置。配置 PWM1_CCER1 寄存器的 CC2P＝0，选择上升沿做为触发条件。

2. 配置 PWM1_SMCR 寄存器的 SMS=110，选择计数器为触发模式。配置 PWM1_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 PWM1_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（PWM1_ARR，PWM1_CCRx）都会被更新。

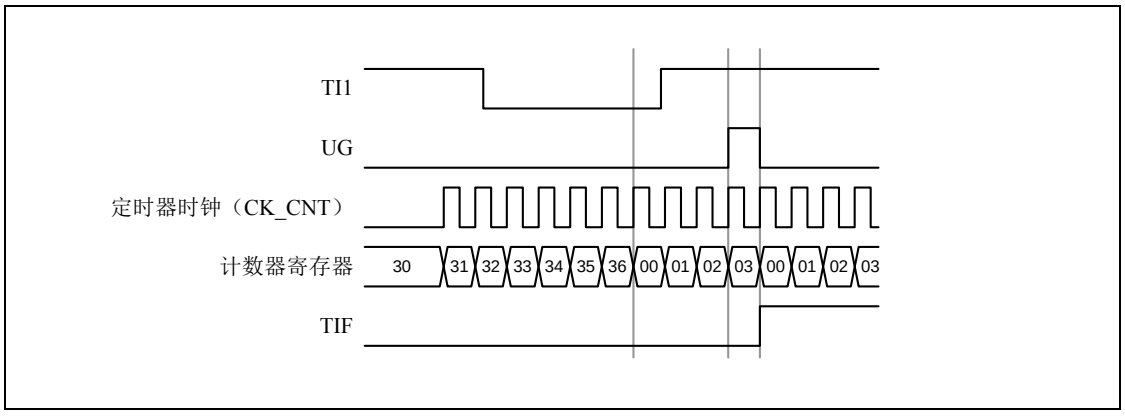
在以下的例子中，TI1 输入端的上升沿导致向上计数器被清零：

1. 配置通道 1 用于检测 TI1 的上升沿。配置输入滤波器的带宽（在本例中，不需要任何滤波器，因此保持 IC1F=0000）。触发操作中不使用捕获预分频器，所以不需要配置。CC1S 位仅用于选择输入捕获源，也不需要配置。配置 PWM1_CCER1 寄存器的 CC1P=0 来选择极性（只检测上升沿）。
2. 配置 PWM1_SMCR 寄存器的 SMS=100，选择定时器为复位触发模式。配置 PWM1_SMCR 寄存器的 TS=101，选择 TI1 作为输入源。
3. 配置 PWM1_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（PWM1_SR1 寄存器的 TIF 位）被置位，如果使能了中断（PWM1_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 PWM1_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

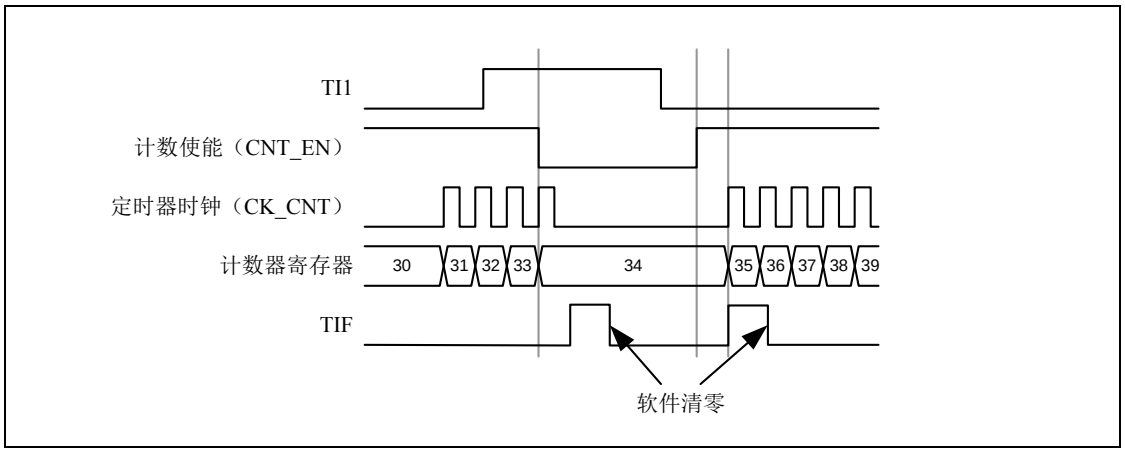
计数器由选中的输入端信号的电平使能。

在如下的例子中，计数器只在 TI1 为低时向上计数：

1. 配置通道 1 用于检测 TI1 上的低电平。配置输入滤波器带宽（本例中，不需要滤波，所以保持 IC1F=0000）。触发操作中不使用捕获预分频器，所以不需要配置。CC1S 位用于选择输入捕获源，也不需要配置。配置 PWM1_CCER1 寄存器的 CC1P=1 来确定极性（只检测低电平）。
2. 配置 PWM1_SMCR 寄存器的 SMS=101，选择定时器为门控触发模式，配置 PWM1_SMCR 寄存器中 TS=101，选择 TI1 作为输入源。
3. 配置 PWM1_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 PWM1_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：

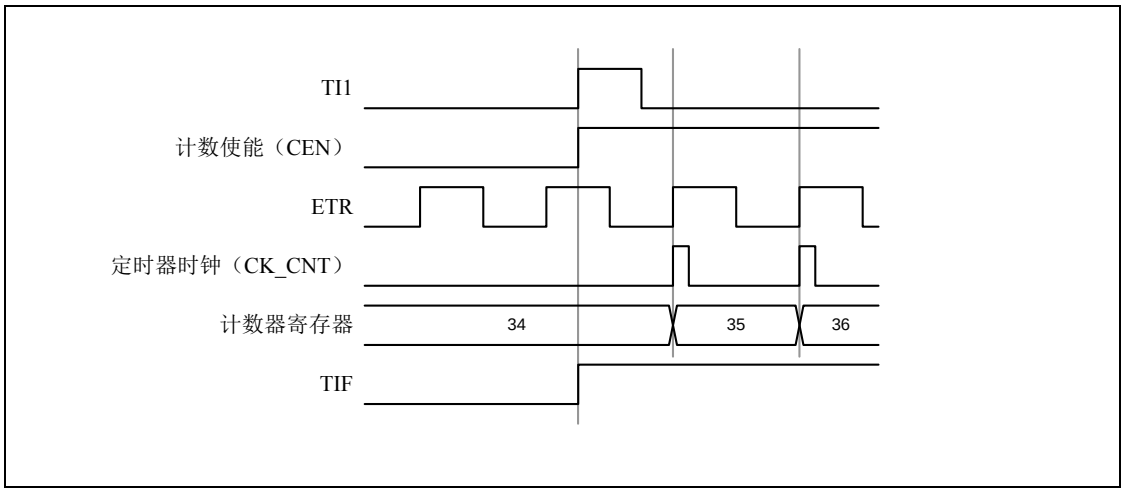
1. 通过 PWM1_ETR 寄存器配置外部触发输入电路。在这个例子中，由于不使用滤波，因此 ETF＝0000。配置 ETPS=00 禁止预分频，配置 ETP=0 监测 ETR 信号的上升沿，配置 ECE=1 使能外部时钟模式 2。
2. 使用通道 1 监测 TI1 的上升沿。配置输入滤波（由于本例不使用滤波，因此配置 IC1F=0000）。由于触发操作不使用预分频，所以不配置预分频器，CC1S 位仅用于选择输入捕获源，因此也不需要配置。配置 PWM1_CCER1 寄存器的 CC1P=0 来选择上升沿触发。
3. 配置 PWM1_SMCR 寄存器的 SMS=110 来选择定时器为触发模式。配置 PWM1_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



19.4.6 与PWM2 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。

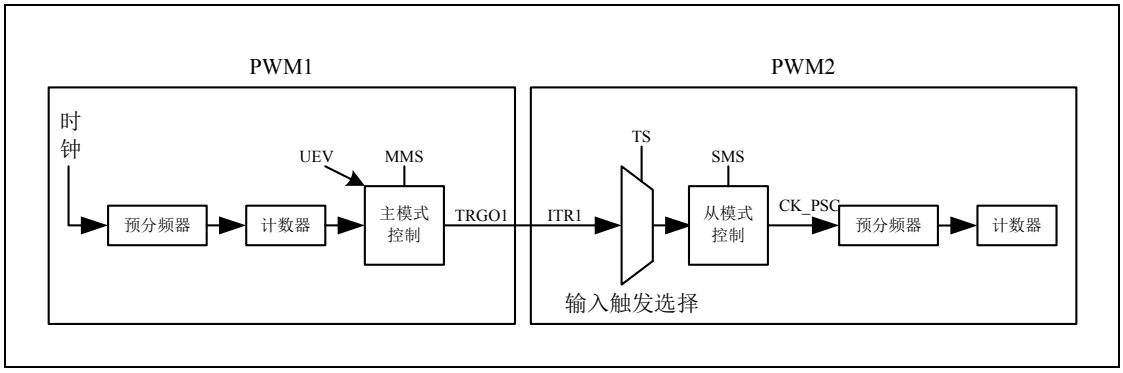
使用一个定时器作为另一个定时器的预分频时钟

例如，用户可以配置定时器 A 作为定时器 B 的预分频时钟，需进行如下配置：

1. 配置定时器 A 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 PWM1_CR2 寄存器的 MMS=010，使每个更新事件时 TRGO1 能输出一个上升沿。
2. 定时器 A 输出的 TRGO1 信号链接到定时器 B。定时器 B 需要配置成触发从模式，使用 ITR1 作为输入触发信号。以上操作可以通过配置 PWM1_SMCR 寄存器的 TS=001 实现。
3. 配置 PWM1_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。此操作将使定时器 A 输出的周期性触发信号（由定时器 A 的溢出产生）的上升沿驱动定时器 B 的时钟。
4. 最后，置位两个定时器的 CEN 位（PWM1_CR1 寄存器中），使能两个定时器。

注意：如果选择 OCi 作为定时器 A 输出的触发信号（MMS=1xx），它的上升沿将驱动定时器 B 的时钟。

主/触发从模式的定时器例子



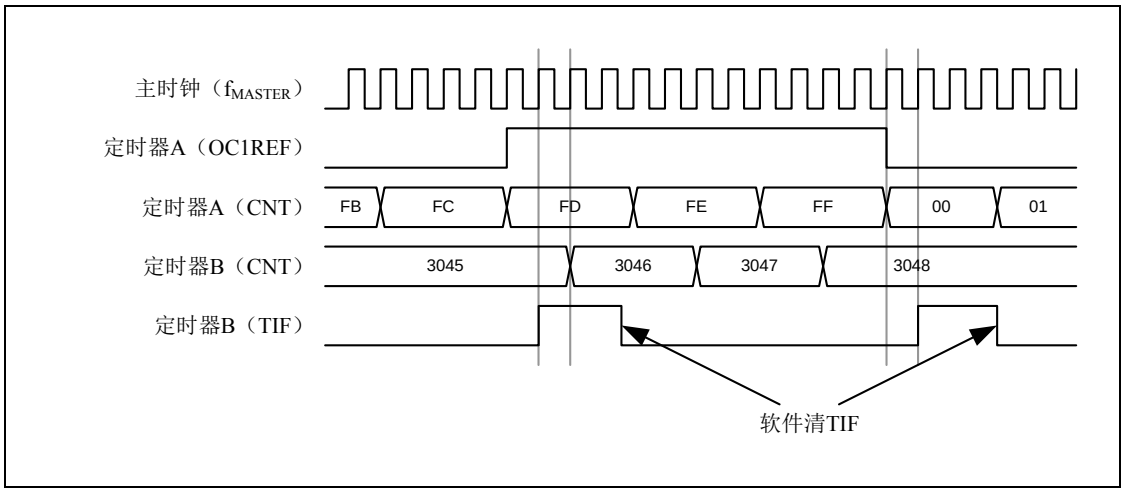
使用一个定时器使能另一个定时器

在本例中，我们用定时器 A 的比较输出使能定时器 B。定时器 B 仅在定时器 A 的 OC1REF 信号为高时按照自己的驱动时钟计数。两个定时器都使用 4 分频的 f_{MASTER} 为时钟（$f_{CK\_CNT} = f_{MASTER}/4$）。

1. 配置定时器 A 为主模式，将比较输出信号（OC1REF）作为触发信号输出。（配置 PWM1_CR2 寄存器的 MMS=100）。
2. 配置定时器 A 的 OC1REF 信号的波形（PWM1_CCMR1 寄存器）。
3. 配置定时器 B 把定时器 A 的输出作为自己的触发输入信号（配置 PWM1_SMCR 寄存器的 TS=001）。
4. 配置定时器 B 为门控触发模式（配置 PWM1_SMCR 寄存器的 SMS=101）。
5. 置位 CEN 位（PWM1_CR1 寄存器），使能定时器 B。
6. 置位 CEN 位（PWM1_CR1 寄存器），使能定时器 A。

注意：两个计数器的时钟并不同步，但仅影响定时器 B 的使能信号。

定时器 A 的输出门控触发定时器 B

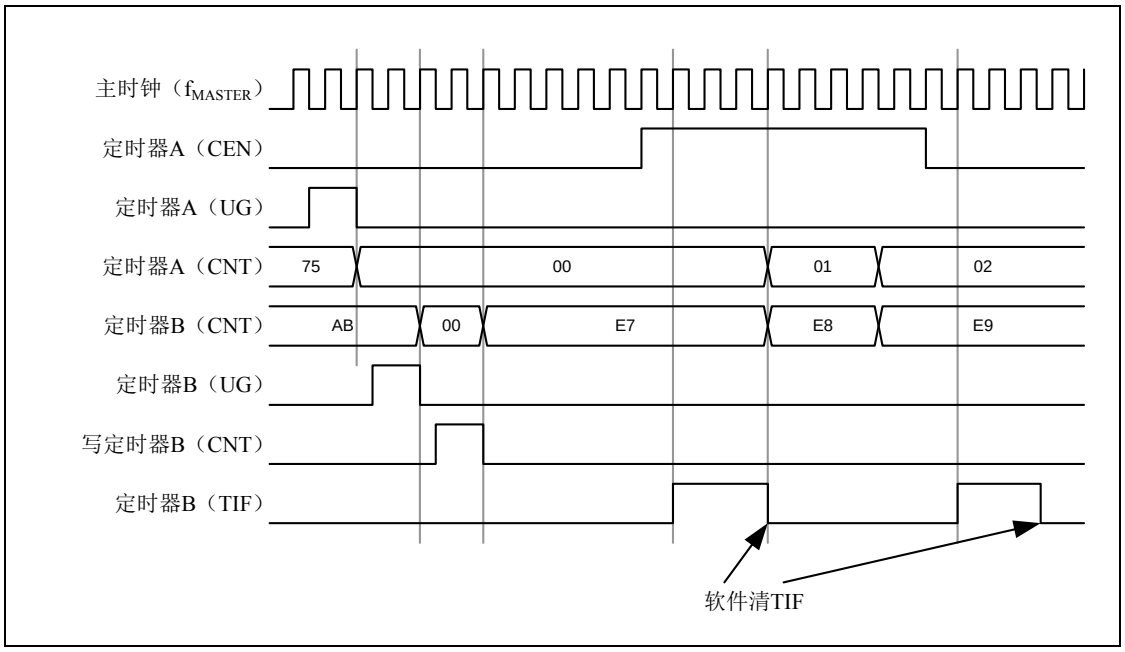


上图中，定时器 B 的计数器和预分频器都没有在启动前初始化，所以都是从现有值开始计数的。如果在启动定时器 A 之前复位两个定时器，用户就可以写入期望的数值到定时器的计数器，使之从指定值开始计数。对定时器的复位操作可以通过软件写 PWM1_EGR 寄存器的 UG 位实现。

在下面这个例子中，我们使定时器 A 和定时器 B 同步。定时器 A 为主模式并从 0 启动计数。定时

器 B 为触发从模式，并从 0xE7 启动计数。两个定时器采用相同的分频系数。当清除 PWM1_CR1 寄存器的 CEN 位时，定时器 A 被禁止，同时定时器 B 停止计数。

1. 配置定时器 A 为主模式，将比较输出信号（OC1REF）作为触发信号输出。（配置 PWM1_CR2 寄存器的 MMS=100）。
2. 配置定时器 A 的 OC1REF 信号的波形（PWM1_CCMR1 寄存器）。
3. 配置定时器 B 把定时器 A 的输出作为自己的触发输入信号（配置 PWM1_SMCR 寄存器的 TS=001）。
4. 配置定时器 B 为门控触发模式（配置 PWM1_SMCR 寄存器的 SMS=101）。
5. 通过对 UG 位（PWM1_EGR 寄存器）写 1，复位定时器 A。
6. 通过对 UG 位（PWM1_EGR 寄存器）写 1，复位定时器 B。
7. 将 0xE7 写入定时器 B 的计数器中（PWM1_CNTRL），初始化定时器 B。
8. 通过对 CEN 位（PWM1_CR1 寄存器）写 1，使能定时器 B。
9. 通过对 CEN 位（PWM1_CR1 寄存器）写 1，启动定时器 A。
10. 通过对 CEN 位（PWM1_CR1 寄存器）写 0，停止定时器 A。



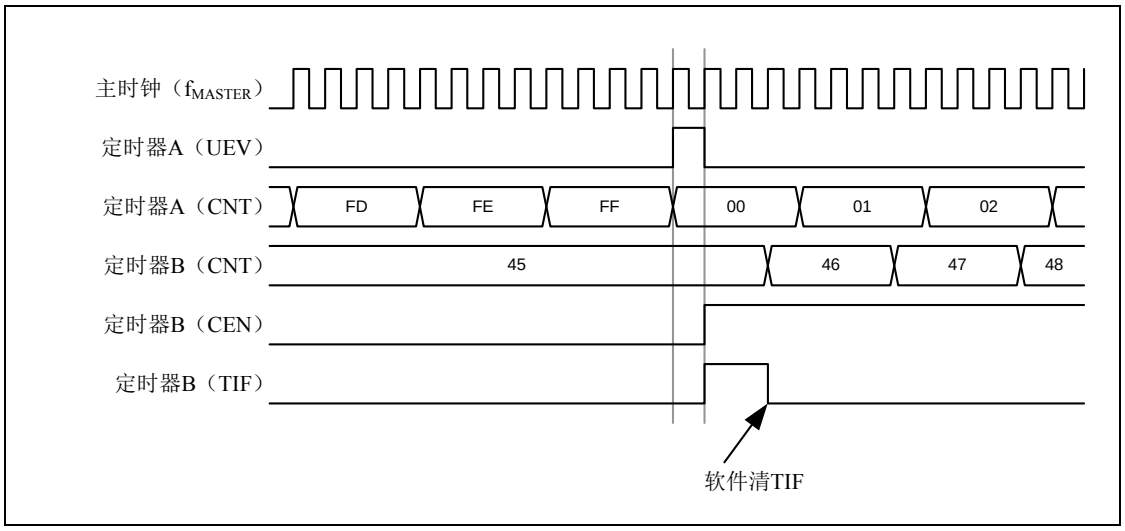
使用一个定时器启动另一个定时器

在本例中，我们用定时器 A 的更新事件来使能定时器 B。

定时器 B 在定时器 A 发生更新事件时按照定时器 B 自己的驱动时钟从它的现有值开始计数（可以是非 0 值）。定时器 B 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 PWM1_CR1 寄存器的 CEN 位写 0。两个定时器都使用 4 分频的 f_{MASTER} 作为驱动时钟（$f_{CK\_CNT} = f_{MASTER}/4$）。

1. 配置定时器 A 为主模式，输出更新信号（UEV）。（配置 PWM1_CR2 寄存器的 MMS=010）。
2. 配置定时器 A 的周期（PWM1_ARR 寄存器）。
3. 配置定时器 B 用定时器 A 的输出作为输入的触发信号（配置 PWM1_SMCR 寄存器的 TS=001）。
4. 配置定时器 B 为触发模式（配置 PWM1_SMCR 寄存器的 SMS=110）。
5. 置位 CEN 位（PWM1_CR1 寄存器）启动定时器 A。

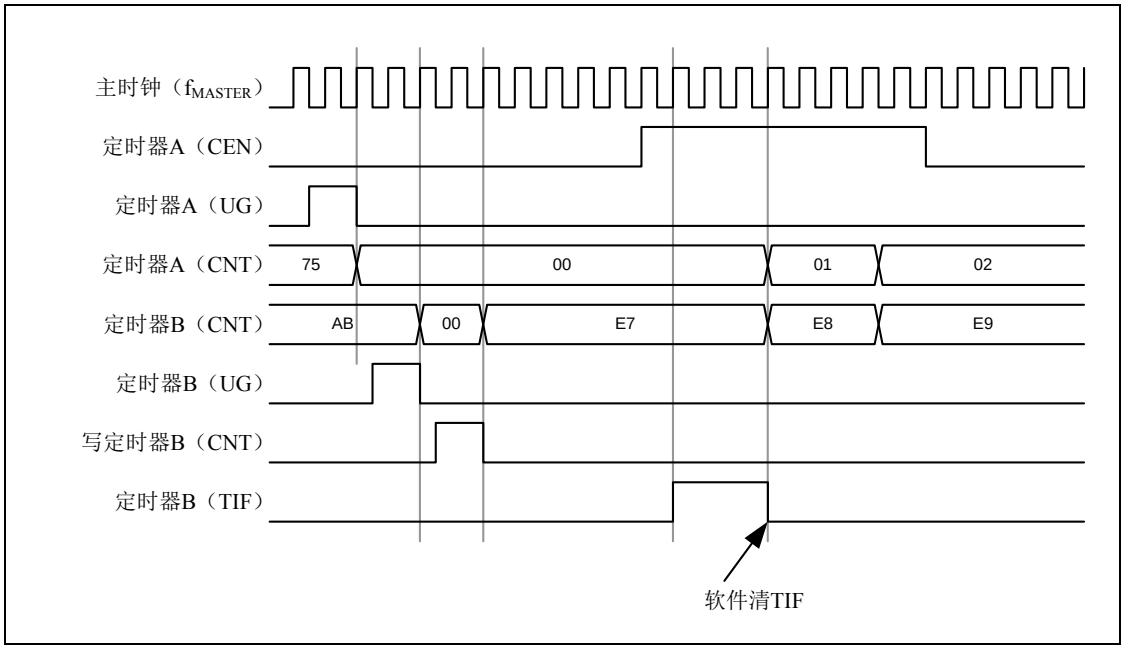
定时器 A 的更新事件（TIMERA-UEV）触发定时器 B



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。

下面的例子采用和前一个相同的配置，但用普通触发模式（配置 PWM1_SMCR 寄存器的 SMS=110）取代了门控触发模式。

定时器 A 的 CNT_EN 触发定时器 B



用外部信号同步的触发两个定时器

在本例中，使用 TI1 的上升沿使能定时器 A，并同时使能定时器 B。为了保持定时器的对齐，定时器 A 需要配置成主/从模式（对于 TI1 信号为从模式，对于定时器 B 为主模式）。

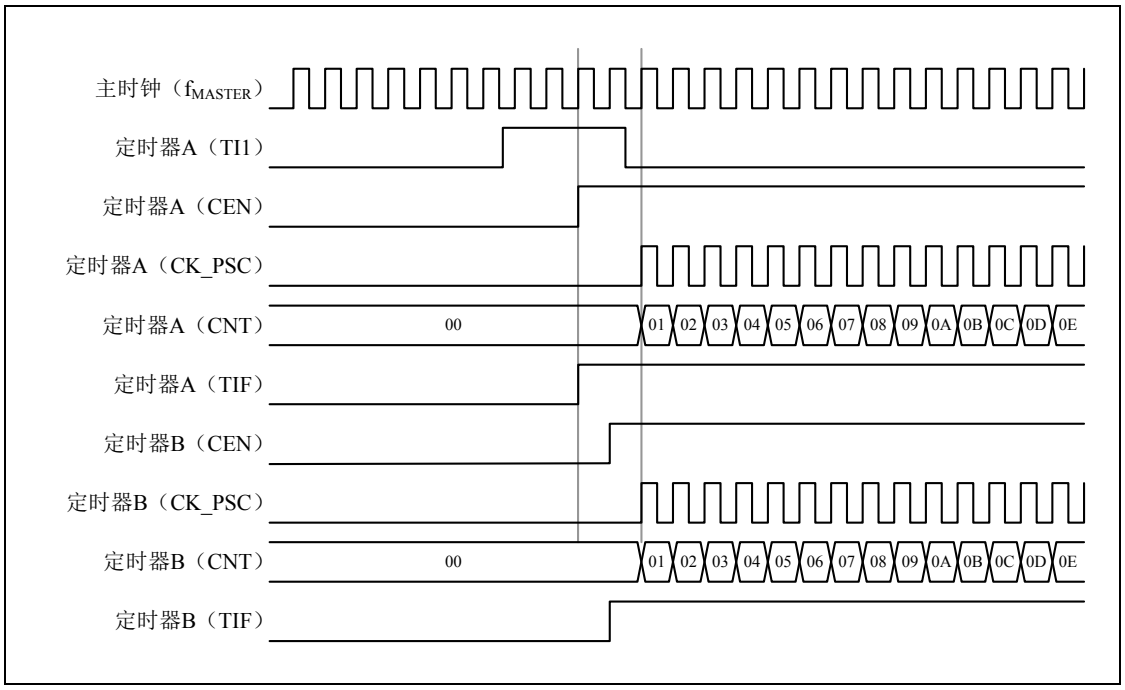
1. 配置定时器 A 为主模式，以输出使能信号作为定时器 B 的触发（配置 PWM1_CR2 寄存器的 MMS=001）。
2. 配置定时器 A 为从模式，把 TI1 信号作为输入的触发信号（配置 PWM1_SMCR 寄存器的 TS=100）。
3. 配置定时器 A 的触发模式（配置 PWM1_SMCR 寄存器的 SMS=110）。

4. 配置定时器 A 为主/从模式（配置 PWM1_SMCR 寄存器的 MSM=1）。
5. 配置定时器 B 以定时器 A 的输出为输入触发信号（配置 PWM1_SMCR 寄存器的 TS=001）。
6. 配置定时器 B 的触发模式（配置 PWM1_SMCR 寄存器的 SMS=110）。

当 TI1（定时器 A 的）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（PWM1_CNT）来插入一个偏移量，这样的话，在定时器 A 的 CK_PSC 信号和 CNT_EN 信号间会插入延时。

定时器 A 的 TI1 信号触发定时器 A 和定时器 B

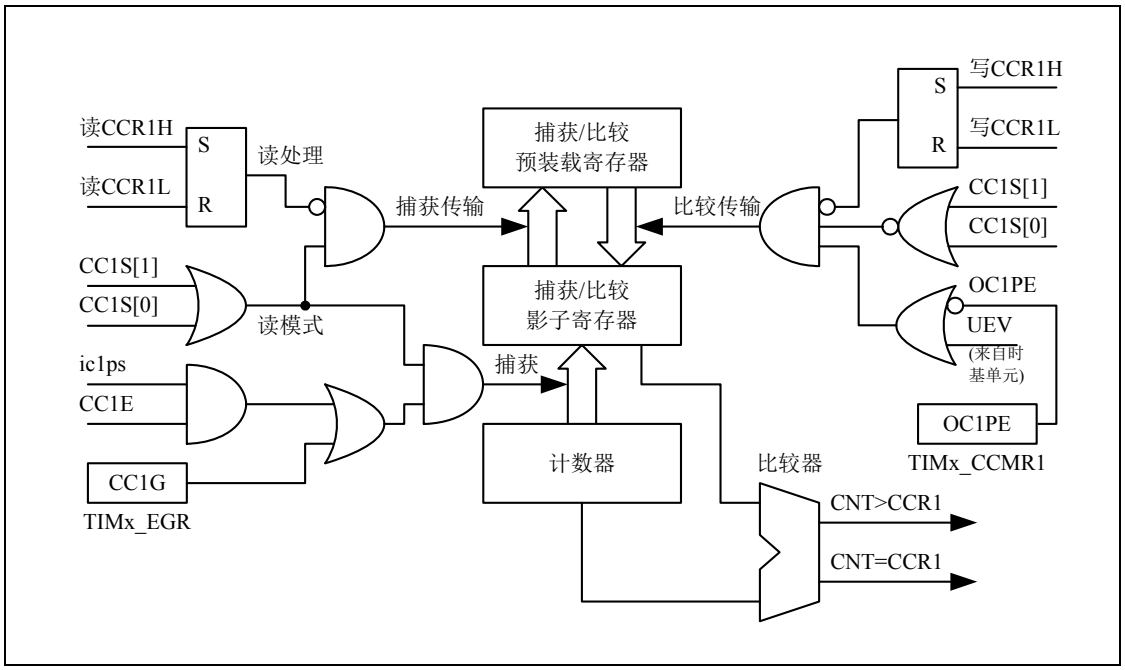


19.5 捕获/比较通道

定时器的 I/O 引脚（PWM1_CCi）可以用作输入捕获或者输出比较，这个功能可以通过配置捕获/比较通道模式寄存器（PWM1_CCMRi）的 CCiS 通道选择位来实现，此处的 i 代表通道数。

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

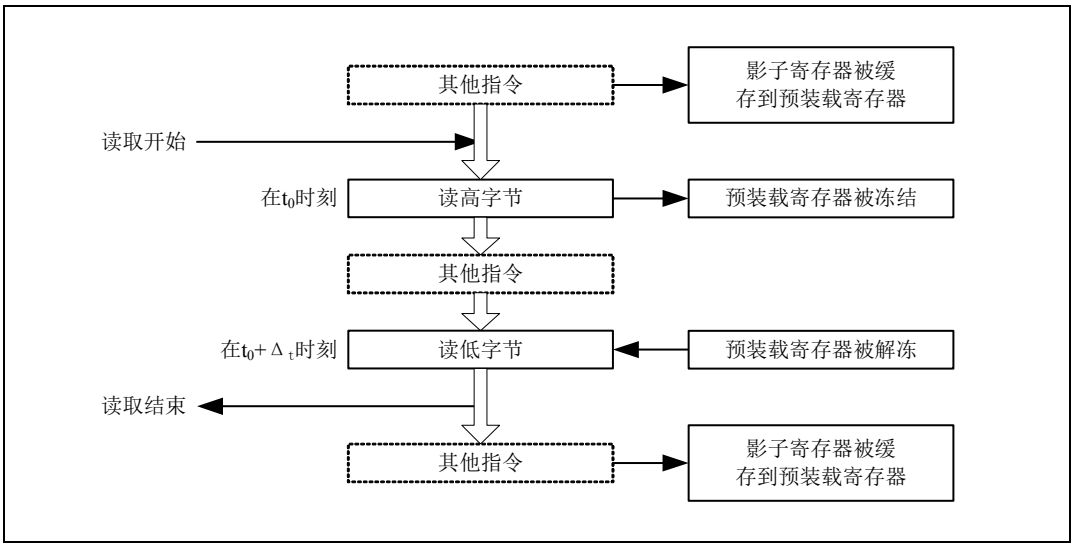


捕获/比较模块由一个预装载寄存器和一个影子寄存器组成。读写过程仅操作预装载寄存器。在捕获模式下，捕获发生在影子寄存器上，然后再复制到预装载寄存器中。在比较模式下，预装载寄存器的内容被复制到影子寄存器中，然后影子寄存器的内容和计数器进行比较。

当通道被配置成输出模式时（PWM1_CCMRi 寄存器的 CCiS=0），可以随时访问 PWM1_CCRi 寄存器。

当通道被配置成输入模式时，对 PWM1_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 PWM1_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。

PWM1 通道 1 的输入



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 PWM1_CCRiL 寄存器，返回计数器数值的低位（LS）。

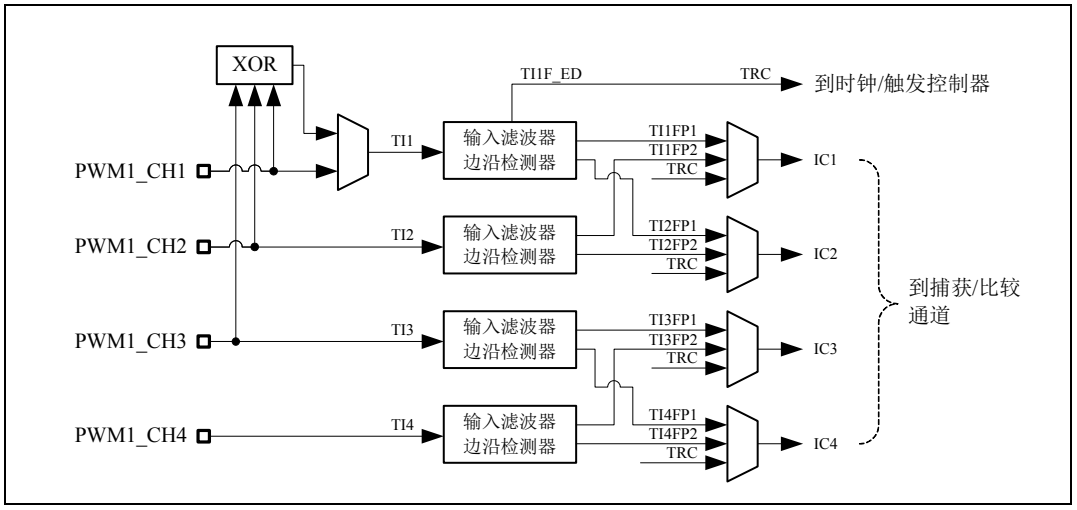
如果在读了低位（LS）数据以后再读高位（MS）数据，将不再返回同样的低位数据。

19.5.1 16 位PWM1_CCRi寄存器的写流程

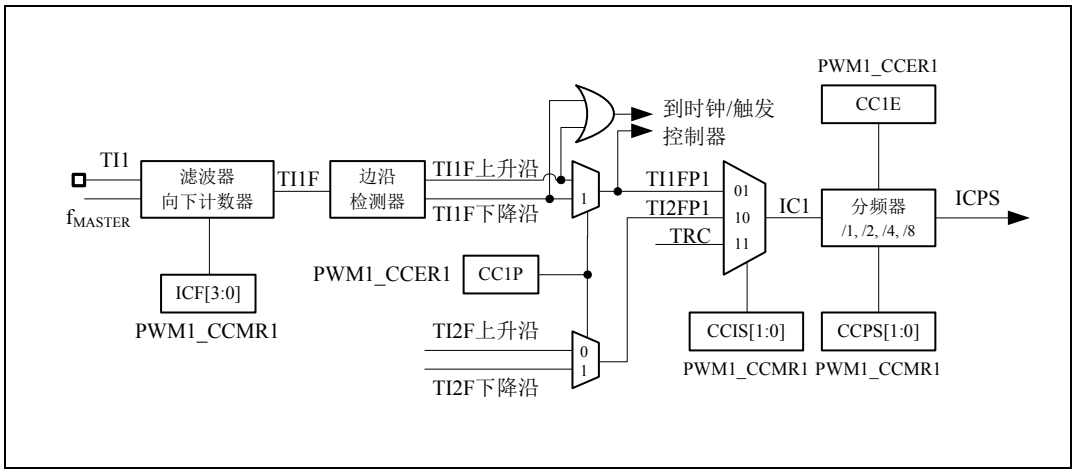
16 位 PWM1_CCRi 寄存器的写操作通过预装载寄存器完成。必需使用两条指令来完成整个流程，一条指令对应一个字节。必需先写高位字节（MS）。在写高位字节（MS）时，影子寄存器的更新被禁止直到低位字节（LS）的写操作完成。

19.5.2 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）。



19.5.3 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（PWM1_CCRx）中。当发生捕获事件时，相应的 CCiIF 标志（PWM1_SR 寄存器）被置 1。

如果 PWM1_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（PWM1_SR2 寄存器）被置 1。写 CCiIF=0 或读取存储在 PWM1_CCRiL 寄存器中的捕获数据都可清除 CCiIF。写 CCiOF=0 可清除 CCiOF。

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 PWM1_CCR1 寄存器中，步骤如下：

1. 选择有效输入端：例如 PWM1_CCR1 连接到 TI1 输入，所以写入 PWM1_CCR1 寄存器中的 CC1S=01，此时通道被配置为输入，并且 PWM1_CCR1 寄存器变为只读。
2. 根据输入信号 TIi 的特点，可通过配置 PWM1_CCMRi 寄存器中的 ICiF 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 TIMi_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 PWM1_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 PWM1_CCMR1 寄存器的 IC1PS=00）。
5. 设置 PWM1_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 PWM1_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 PWM1_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。

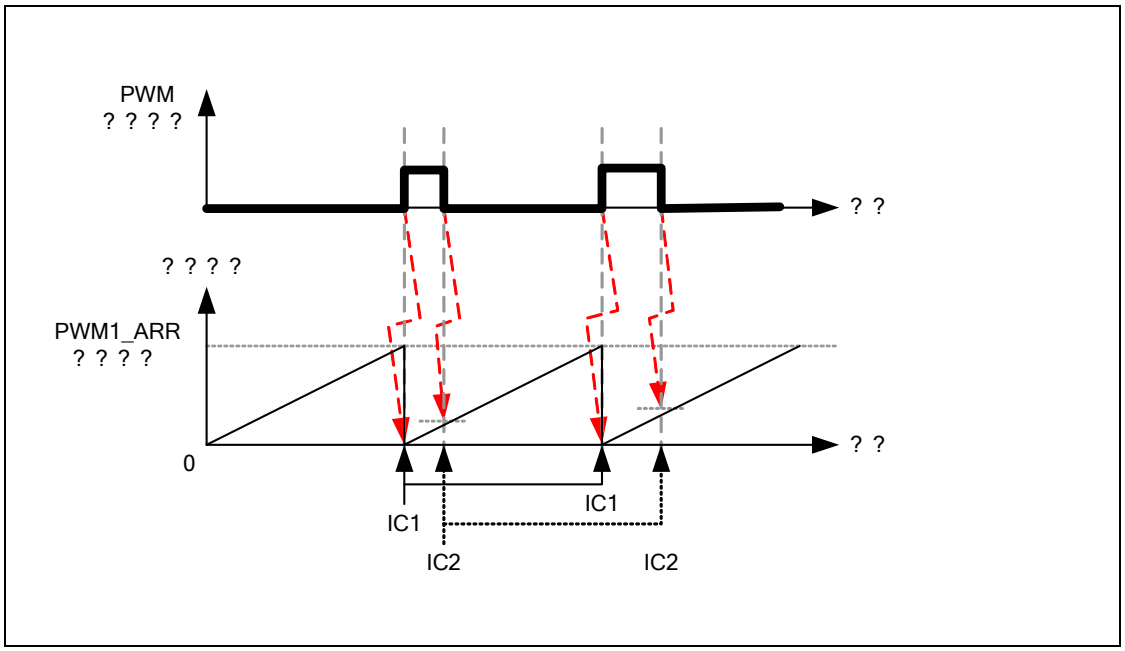
为了处理捕获溢出事件（CC1OF 位），建议在读出重复捕获标志之前读取数据，这是为了避免丢失在读出捕获溢出标志之后和读取数据之前可能产生的重复捕获信息。

注意：设置 PWM1_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：

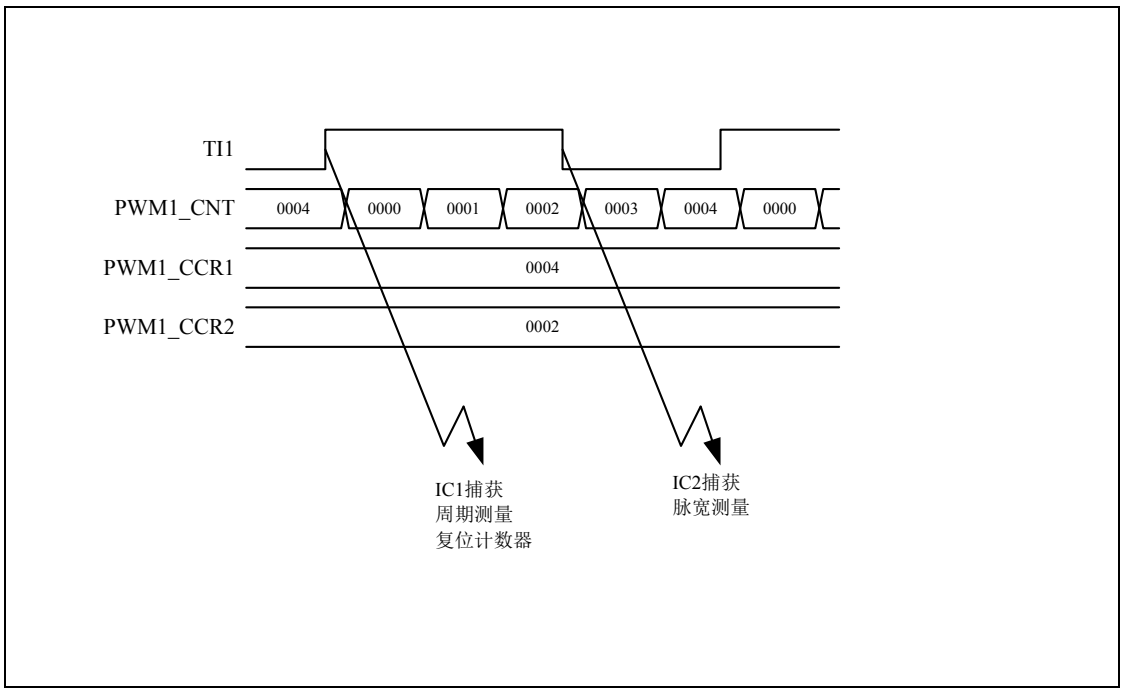
- 两个 ICi 信号被映射至同一个 TIi 输入。
- 这两个 ICi 信号的有效边沿的极性相反。
- 其中一个 TIiFP 信号被作为触发输入信号，而触发模式控制器被配置成复位触发模式。



例如，你可以用以下方式测量 TI1 上输入的 PWM 信号的周期（PWM1_CCR1 寄存器）和占空比（PWM1_CCR2 寄存器）。（具体取决于 f_{MASTER} 的频率和预分频器的值）

1. 选择 PWM1_CCR1 的有效输入：置 PWM1_CCMR1 寄存器的 CC1S=01（选中 TI1）。
2. 选择 TI1FP1 的有效极性（用来捕获数据到 PWM1_CCR1 中和清除计数器）：置 CC1P=0（上升沿 有效）。
3. 选择 PWM1_CCR2 的有效输入：置 PWM1_CCMR2 寄存器的 CC2S=10（选中 TI1FP2）。
4. 选择 TI1FP2 的有效极性（捕获数据到 PWM1_CCR2）：置 CC2P=1（下降沿有效）。
5. 选择有效的触发输入信号：置 PWM1_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 PWM1_SMCR 中的 SMS=100。
7. 使能捕获：置 PWM1_CCER1 寄存器中 CC1E=1，CC2E=1。

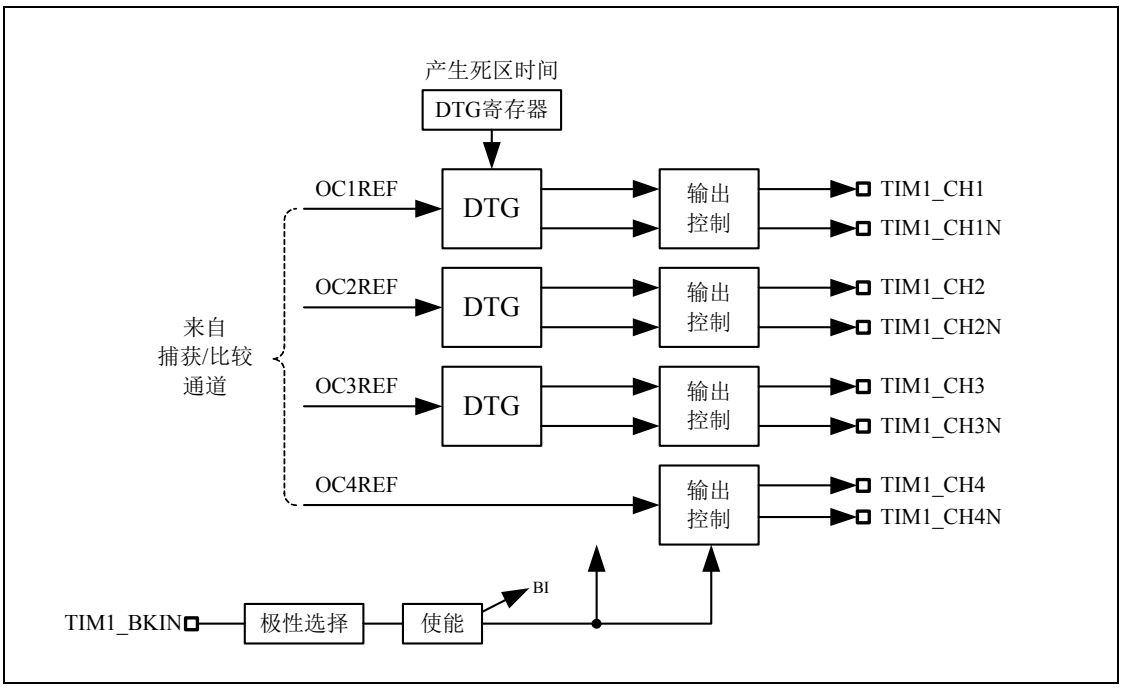
PWM 输入信号测量实例



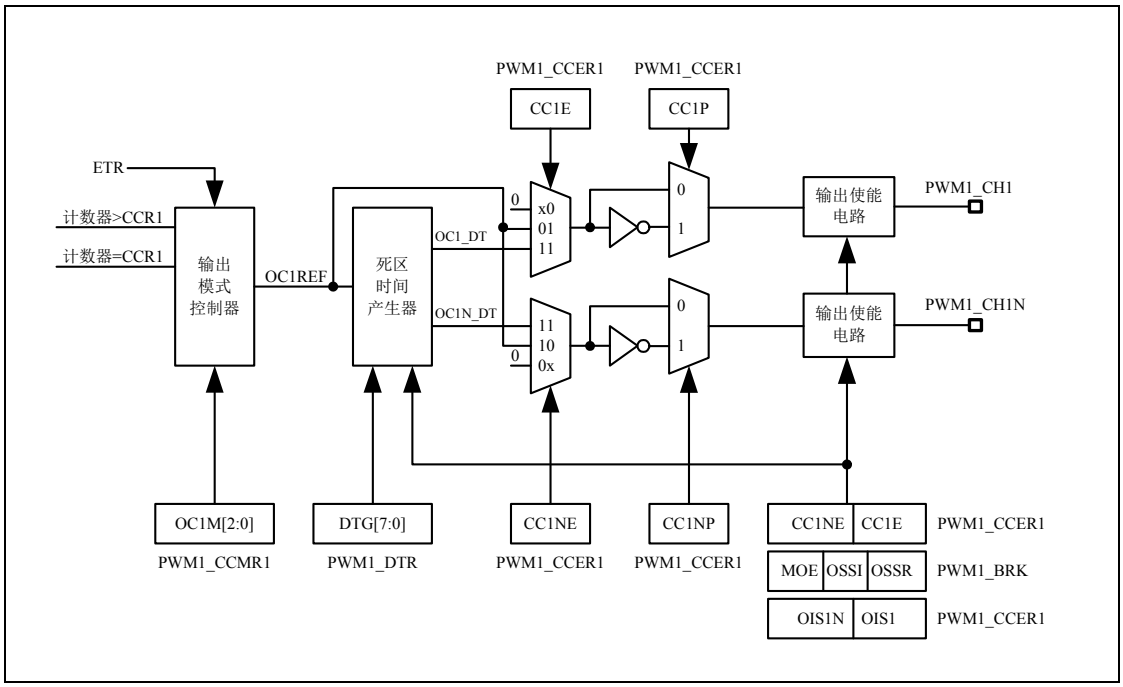
19.5.4 输出模块

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



详细的带互补输出的输出模块框图（通道 1）



19.5.5 强制输出模式

在输出模式（PWM1_CCMRi 寄存器中 CCiS=00）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 PWM1_CCMRi 寄存器中相应的 OCiM=101，即可强制输出比较信号为有效状态。这样 OCiREF 被强制为高电平（OCiREF 始终为高电平有效），而 OCi 的输出是高还是低则取决于 CCiP 极性标志位。

例如：如果 CCiP=0（OCi 高电平有效），则 OCi 被强制为高电平。

置 PWM1_CCMRi 寄存器的 OCiM=100，可强制 OCiREF 信号为低。

该模式下，在 PWM1_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。

19.5.6 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（PWM1_SR1 寄存器中的 CCiIF 位）。
- 若设置了相应的中断使能位（PWM1_IER 寄存器中的 CCiIE 位），则产生一个中断。

PWM1_CCMRi 寄存器的 OCiM 位用于选择输出比较模式，而 PWM1_CCMRi 寄存器的 CCiP 位用

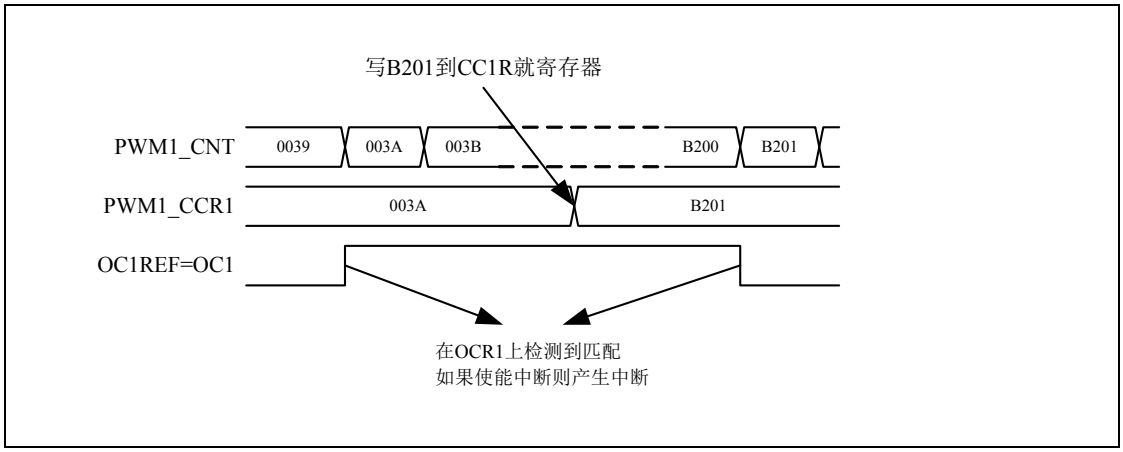
于选择有效和无效的电平极性。PWM1_CCMRi 寄存器的 OCiPE 位用于选择 PWM1_CCRi 寄存器是否需要使用预装载寄存器。在输出比较模式下，更新事件 UEV 对 OCiREF 和 OCi 输出没有影响。时间精度为计数器的一个计数周期。输出比较模式也能用来输出一个单脉冲。

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 PWM1_ARR 和 PWM1_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 PWM1_CR1 寄存器的 CEN 位来启动计数器

PWM1_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 PWM1_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



19.5.7 PWM模式

脉冲宽度调制（PWM）模式可以产生一个由 PWM1_ARR 寄存器确定频率，由 PWM1_CCRi 寄存器确定占空比的信号。

在 PWM1_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 PWM1_CCMRi 寄存器的 OCiPE 位使能相应的预装载寄存器，也可以设置 PWM1_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 PWM1_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 PWM1_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 PWM1_CCERi 和 PWM1_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，PWM1_CNT 和 PWM1_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 PWM1_CCRi≤PWM1_CNT 或者 PWM1_CNT≤PWM1_CCRi。

根据 PWM1_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。

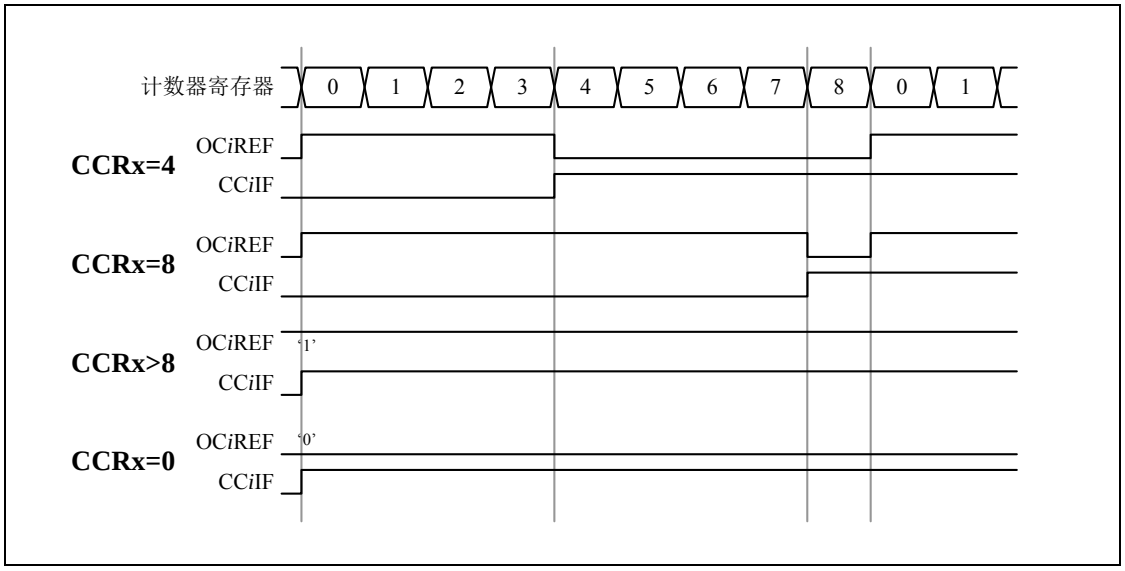
PWM 边沿对齐模式

向上计数配置

当 PWM1_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 PWM1_CNT<PWM1_CCRi 时，PWM 参考信号 OCiREF 为高，否则为低。如果 PWM1_CCRi 中的比较值大于自动重装载值（PWM1_ARR），则 OCiREF 保持为'1'。如果比较值为 0，则 OCiREF 保持为'0'。

边沿对齐，PWM 模式 1 的波形（ARR=8）



向下计数的配置

当 PWM1_CR1 寄存器的 DIR 位为 1 时，执行向下计数。

在 PWM 模式 1 时，当 PWM1_CNT>PWM1_CCRi 时参考信号 OCiREF 为低，否则为高。如果 PWM1_CCRi 中的比较值大于 PWM1_ARR 中的自动重装载值，则 OCiREF 保持为'1'。该模式下不能产生 0%的 PWM 波形。

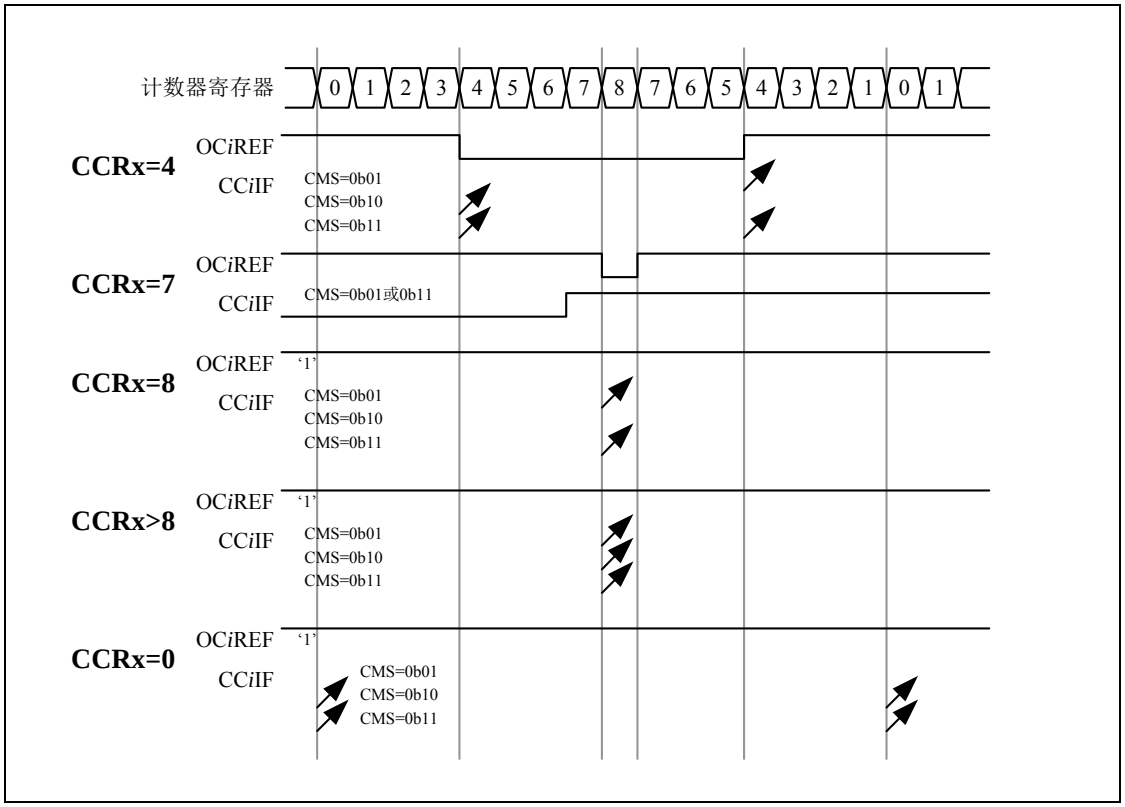
PWM 中央对齐模式

当 PWM1_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。PWM1_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：

- PWM1_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（APR=8）



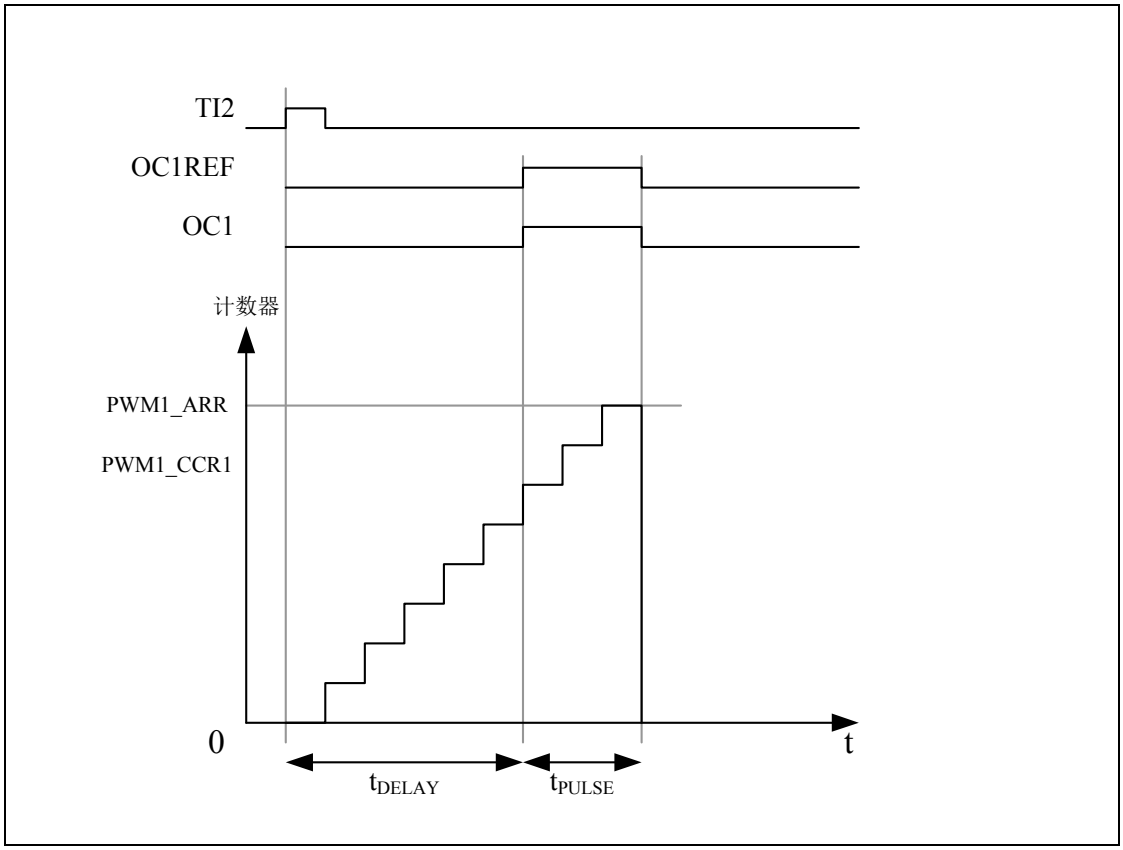
单脉冲模式

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 PWM1_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 tDELAY，在 OC1 上产生一个 tPULSE 宽度的正脉冲：（假定 IC2 作为触发 1 通道的触发源）

- 置 PWM1_CCMR2 寄存器的 CC2S=01，把 IC2 映射到 TI2。
- 置 PWM1_CCER1 寄存器的 CC2P=0，使 IC2 能够检测上升沿。
- 置 PWM1_SMCR 寄存器的 TS=110，使 IC2 作为时钟/触发控制器的触发源（TRGI）。
- 置 PWM1_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 PWM1_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（PWM1_ARR – PWM1_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 PWM1_CCMR1 寄存器的 OCiM=111，进入 PWM 模式 2，根据需要有选择的设置 PWM1_CCMR1 寄存器的 OC1PE=1，置位 PWM1_CR1 寄存器中的 ARPE，使能预装载寄存器，然后在 PWM1_CCR1 寄存器中填写比较值，在 PWM1_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，PWM1_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 PWM1_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

tDELAY。

如果要以最小延时输出波形，可以设置 PWM1_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 PWM1 和 PWM2 模式时起作用。

互补输出和死区插入

PWM1 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 PWM1_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：PWM1_CCERi 寄存器的 CCiE 和 CCiNE 位，PWM1_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。参考信号 OCiREF 可以产生 2 路输出 OCi 和 OCiN。

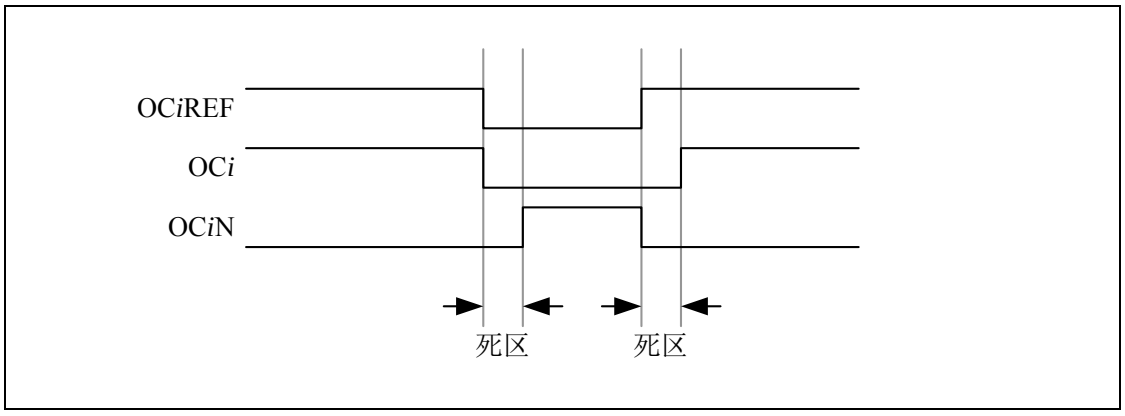
如果 OCi 和 OCiN 为高有效：

- OCi 输出信号与参考信号相同，只是它的上升沿相对于参考信号的上升沿有一个延迟。
- OCiN 输出信号与参考信号相反，只是它的上升沿相对于参考信号的下降沿有一个延迟。

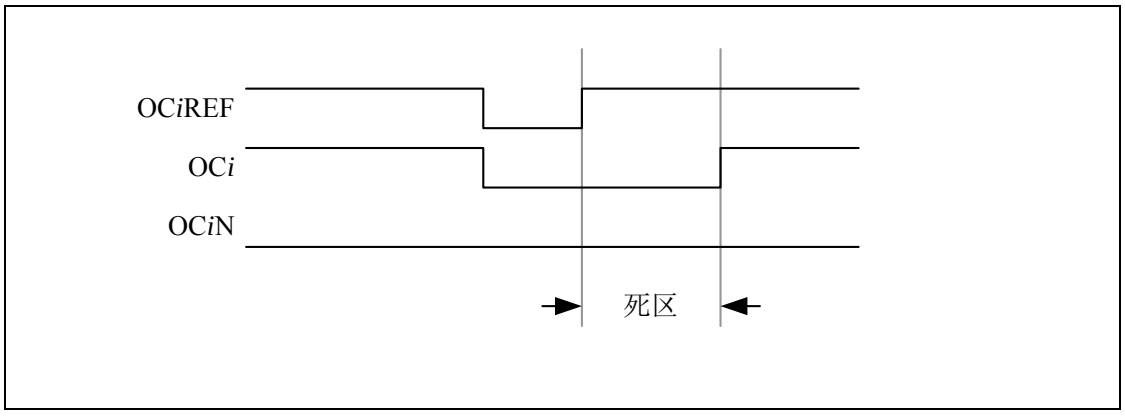
如果延迟大于当前有效的输出宽度（OCi 或者 OCiN），则不会产生相应的脉冲。

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）

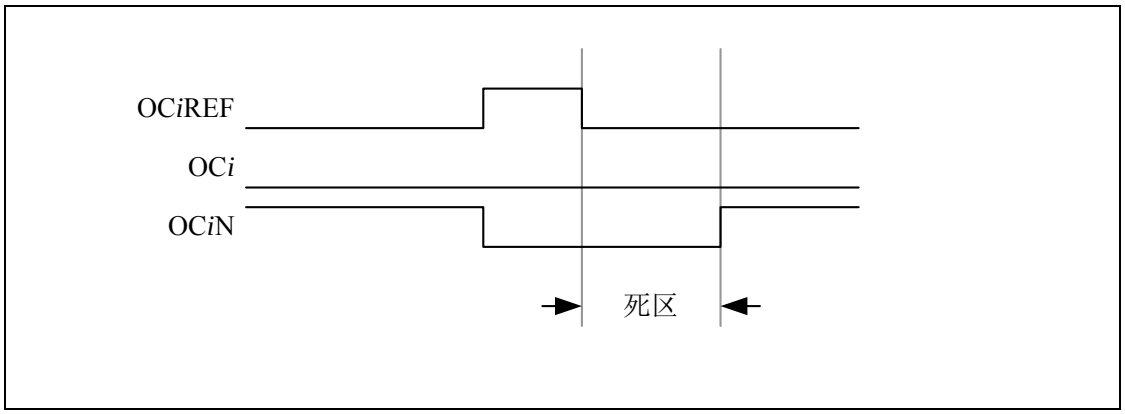
带死区插入的互补输出



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 PWM1_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 PWM1_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）。

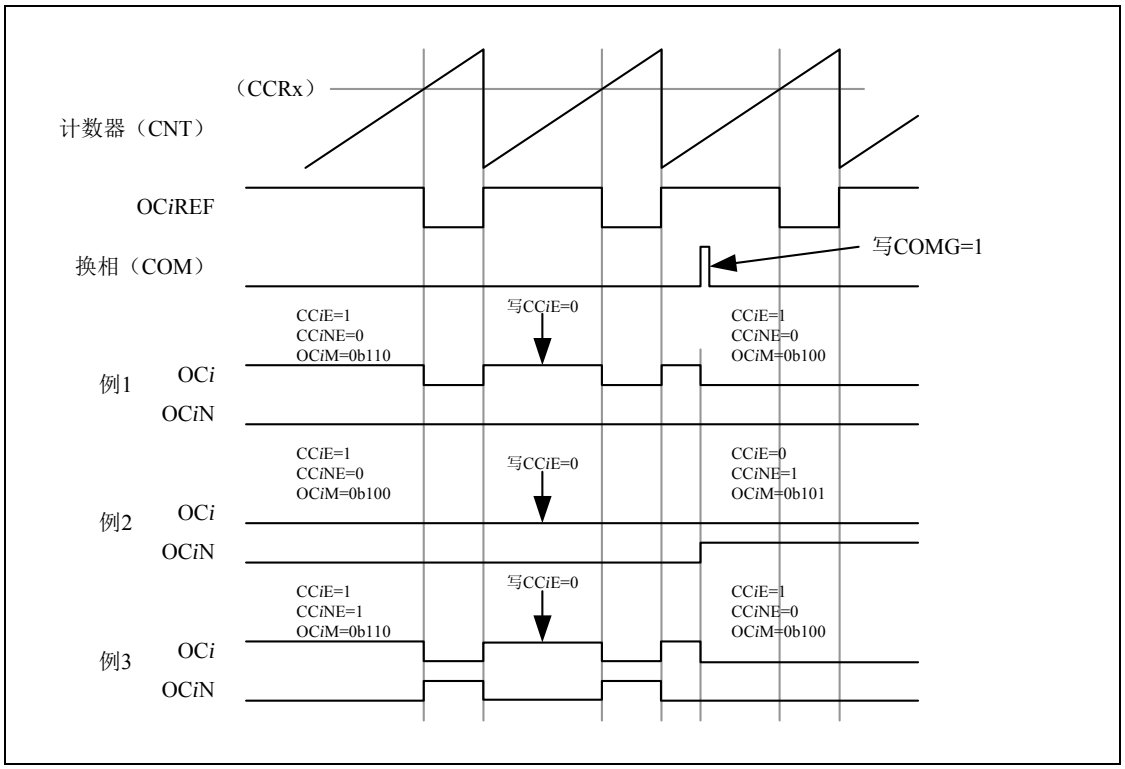
注：当只使能 OCiN（CCiE=0, CCiNE=1）时，它不会反相，而当 OCiREF 变高时立即有效。例如，如果 CCiNP=0，则 OCiN=OCiREF。另一方面，当 OCi 和 OCiN 都被使能时（CCiE=CCiNE=1），当 OCiREF 为高时 OCi 有效；而 OCiN 相反，当 OCiREF 低时 OCiN 变为有效。

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 PWM1_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）



19.5.8 使用刹车功能

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（PWM1_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 PWM1_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 PWM1_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：

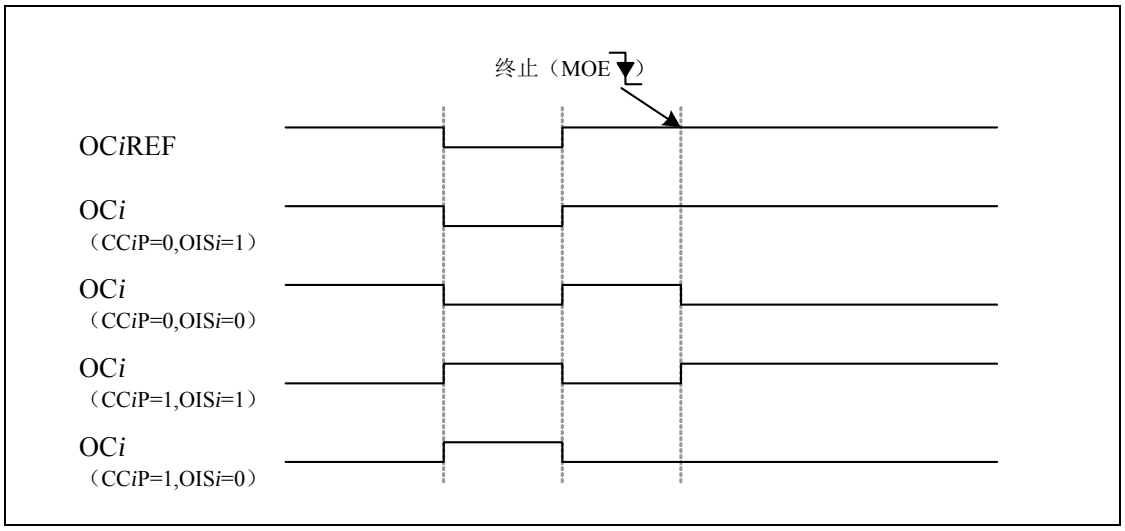
- MOE 位被异步地清除，将输出置于无效状态、空闲状态或者复位状态（由 OSSI 位选择）。这个特性在 MCU 的振荡器关闭时依然有效。
- 一旦 MOE=0，每一个输出通道输出由 PWM1_OISR 寄存器的 OISi 位设定的电平。如果 OSSI=0，则定时器不再控制输出使能信号，否则输出使能信号始终为高。
- 当使用互补输出时：
  - 输出首先被置于复位状态即无效的状态（取决于极性）。这是异步操作，即使定时器没有时钟时，此功能也有效。
  - 如果定时器的时钟依然存在，死区生成器将会重新生效，在死区之后根据 OISi 和 OISiN 位指示的电平驱动输出端口。即使在这种情况下，OCi 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 PWM1_IER 寄存器的 BIE 位，当刹车状态标志（PWM1_SR1 寄存器中的 BIF 位）为’1’时，则产生一个中断。
- 如果设置了 PWM1_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。

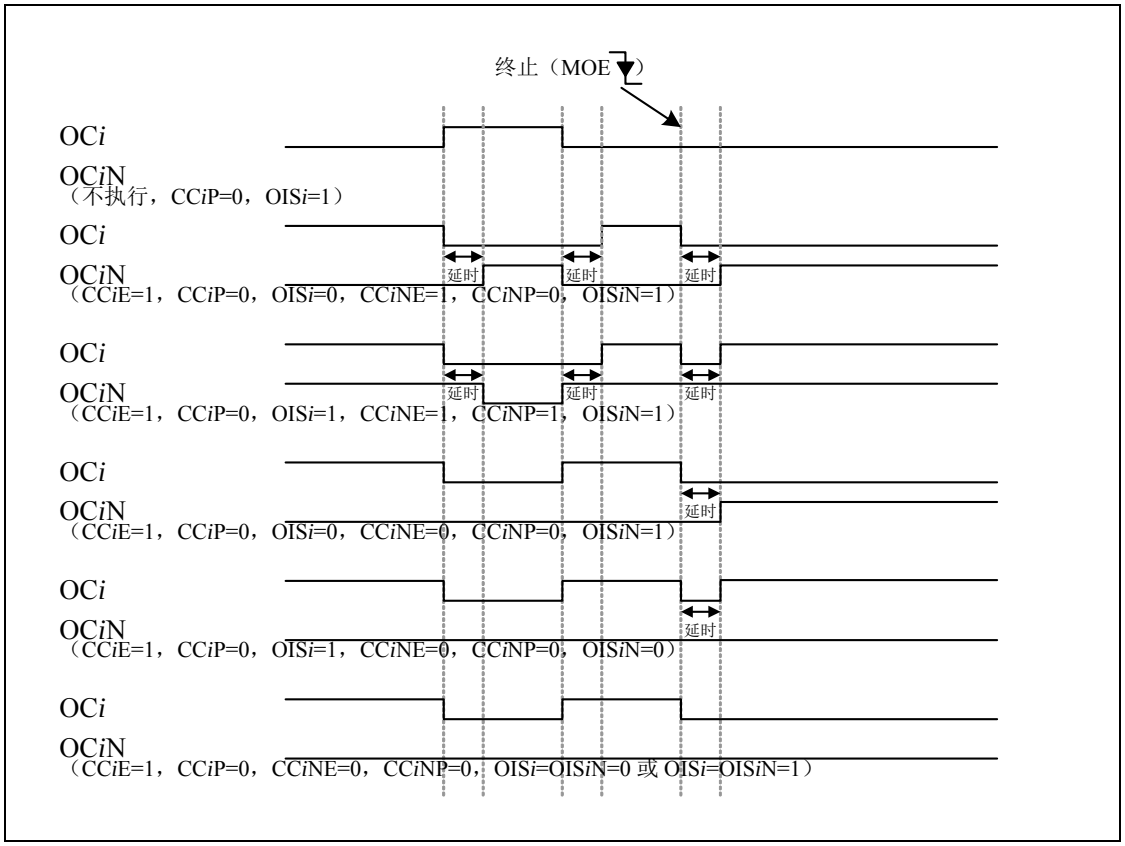
注：刹车输入为电平有效。所以，当刹车输入有效时，不能同时（自动地或者通过软件）设置 MOE。同时，状态标志 BIF 不能被清除。

刹车由 BRK 输入（BKIN）产生，它的有效极性是可编程的，且由 PWM1_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 PWM1_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（PWM1 互补输出）



19.5.9 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 PWM1_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。

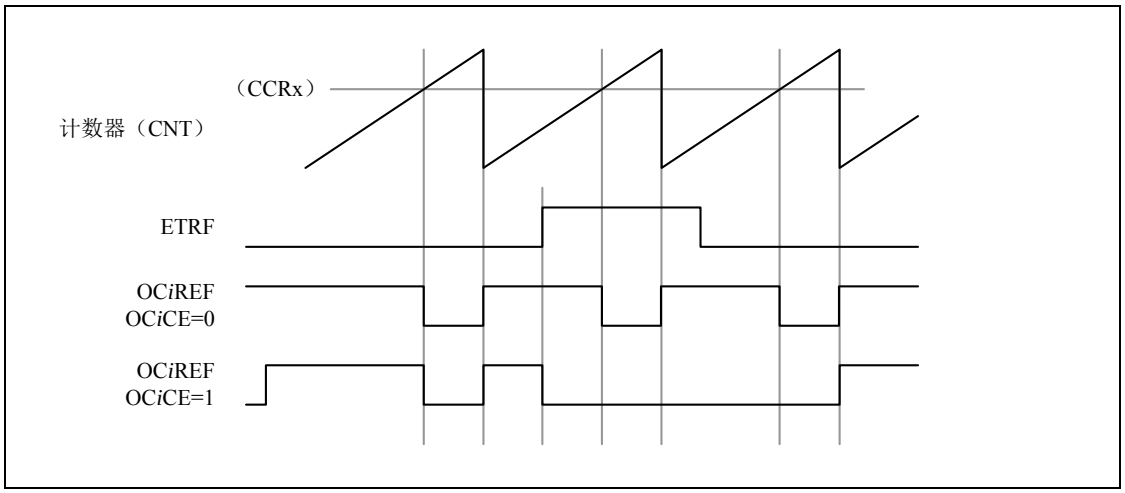
例如，OCiREF 信号可以联到一个比较器的输出，用于控制电流。这时，ETR 必须配置如下：

1. 外部触发预分频器必须处于关闭：PWM1_ETR 寄存器中的 ETPS[1:0]=00。
2. 必须禁止外部时钟模式 2：PWM1_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 PWM1 被置于 PWM 模式。

ETR 清除 PWM1 的 OCiREF



19.5.10 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 PWM1_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 PWM1_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（PWM1_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 PWM1_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 PWM1_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 PWM1_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

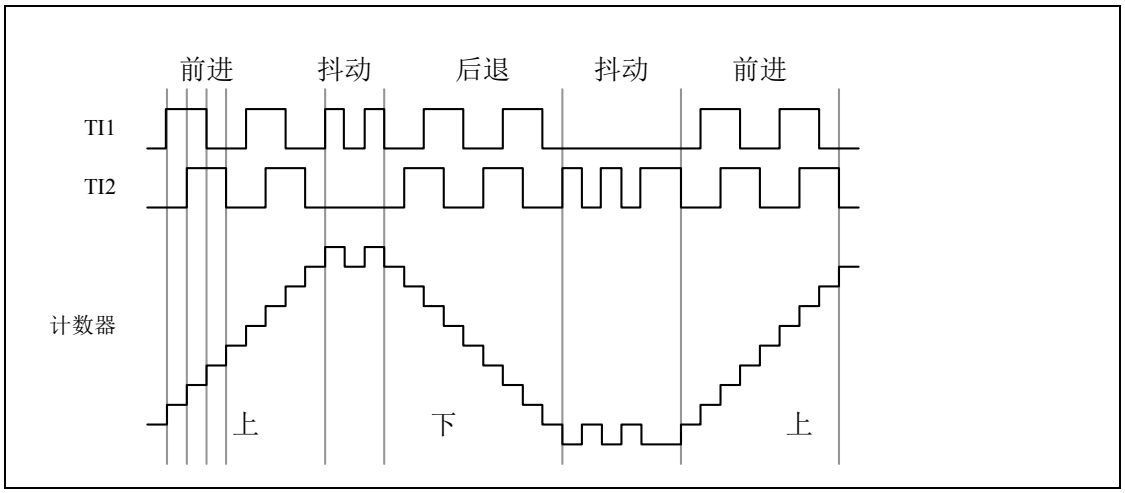
	低	向上计数	向下计数	不计数	不计数
仅在 TI2 计数	高	不计数	不计数	向上计数	向下计数
	低	不计数	不计数	向下计数	向上计数
在 TI1 和 TI2 上计数	高	向下计数	向上计数	向上计数	向下计数
	低	向上计数	向下计数	向下计数	向上计数

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：

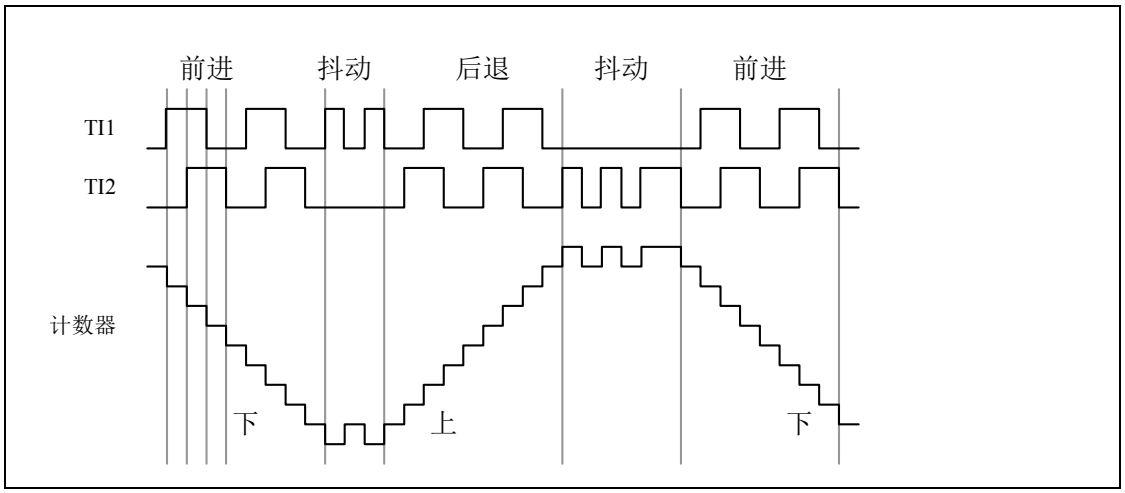
- CC1S=01（PWM1_CCMR1 寄存器，IC1FP1 映射到 TI1）
- CC2S=01（PWM1_CCMR2 寄存器，IC2FP2 映射到 TI2）
- CC1P=0（PWM1_CCER1 寄存器，IC1 不反相，IC1=TI1）
- CC2P=0（PWM1_CCER1 寄存器，IC2 不反相，IC2=TI2）
- SMS=011（PWM1_SMCR 寄存器，所有的输入均在上升沿和下降沿有效）.
- CEN=1（PWM1_CR1 寄存器，计数器使能）

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）。

19.6 中断

PWM1 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 PWM1_IER 寄存器中相应的中断使能位：即 BIE，TIE，COMIE，CCiIE，UIE 位。通过设置 PWM1_EGR 寄存器中的相应位，也可以用软件产生上述各个中断源。

19.7 PWM1 寄存器描述

19.7.1 控制寄存器 1（PWM1_CR1）

寄存器地址：FEC0H

7	6	5	4	3	2	1	0

ARPE	CMS[1:0]	DIR	OPM	URS	UDIS	CEN

位 7	ARPE：自动预装载允许位 0：PWM1_ARR 寄存器没有缓冲，它可以被直接写入； 1：PWM1_ARR 寄存器由预装载缓冲器缓冲。
位 6:5	CMS：选择中央对齐模式 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（PWM1_CCMRx 寄存器 中 CCiS=00）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（PWM1_CCMRx 寄存器 中 CCiS=00）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（PWM1_CCMRx 寄存器 中 CCiS=00）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（PWM1_SMCR 寄存器中的 SMS=001，010，011）必须 被禁止。
位 4	DIR：方向 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。
位 3	OPM：单脉冲模式 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（清除 CEN 位）时，计数器停止。
位 2	URS：更新请求源 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）
位 1	UDIS：禁止更新 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。
位 0	CEN：允许计数器 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。

19.7.2 控制寄存器 2（PWM1_CR2）

寄存器地址：FEC1H

7	6	5	4	3	2	1	0
TI1S	MMS[2:0]			保留	COMS	保留	CCPC

位 7	TI1S：TI1 选择 0：CC1 输入管脚连到 TI1（数字滤波器的输入）； 1：CC1、CC2 和 CC3 管脚经异或后连到 TI1。
位 6:4	MMS：主模式选择 该位用于选择在主模式下送到 ADC 或其它从定时器的同步信息（TRGO）。可能的组合如下： 000：复位 – PWM1_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）。其用于启动多个定时器或 ADC，以 便控制在一段时间内使能从定时器或 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（见 PWM1_SMCR 寄存器中 MSM 位的描述）， 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入（TRGO）。 011：比较脉冲（MATCH1） – 一旦发生一次捕获或一次比较成功，当 CC1IF 标志被置 1 时（即使它 已经为高），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）。
位 3	保留，必须保持为 0。
位 2	COMS：捕获/比较控制位的更新控制选择 0：当捕获/比较的控制位为预装载时（CCPC=1），只有在 COMG 位置 1 的时候这些控制位才被更 新； 1：当捕获/比较的控制位为预装载时（CCPC=1），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； 注：该位只对拥有互补输出的通道有效。
位 1	保留，被硬件设为 0。
位 0	CCPC：捕获/比较预装载控制位 0：CCIE，CCINE，CCiP，CCiNP 位（PWM1_CCERx 寄存器）和 OCIM 位（PWM1_CCMRx 寄存器）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（PWM1_EGR 寄存器）后被更新。 注：该位只对具有互补输出的通道起作用。

19.7.3 从模式控制寄存器(PWM1_SMCR)

寄存器地址：FEC2H

7	6	5	4	3	2	1	0
MSM	TS[2:0]			-	SMS[2:0]		

位 7	MSM：主/从模式 0：无作用； 1：触发输入（TRGI）上的事件被延迟了，以允许定时器 1 与它的从定时器间的完美同步（通过 TRGO）。
位 6:4	TS：触发选择 这 3 位选择用于选择同步计数器的触发输入。 000：内部触发 ITR0 连接到 TIM6 TRGO 001：保留 010：内部触发 ITR2 连接到 TIM5 TRGO 011：保留 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） 注：这些位只能在未用到（如 SMS=000）时被改变，以避免在改变时产生错误的边沿检测。
位 3	保留，始终读为 0。
位 2:0	SMS：时钟/触发/从模式选择 当选择了外部信号，触发信号（TRGI）的有效边沿与选中的外部输入极性相关（见输入控制寄存器 和控制寄存器的说明） 000：时钟/触发控制器禁止 – 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。

19.7.4 外部触发寄存器(PWM1_ETR)

寄存器地址：FEC3H

7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]			

位 7	ETP：外部触发极性，该位决定是 ETR 还是 ETR 用于触发操作。 0：ETR 不反相，即高电平或上升沿有效； 1：ETR 反相，即低电平或下降沿有效。
位 6	ECE：外部时钟使能 该位用于使能外部时钟模式 2。 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 位置 1 的效果与选择把 TRGI 连接到 ETRF 的外部时钟模式 1 相同（PWM1_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（PWM1_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。
位 5:4	ETPS：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： 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。
位 3:0	ETF：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。数字滤波器由一个事件计数器组成，它记录 到 N 个事件后会产生一个输出的跳变： 0000：无滤波器，以 f_{MASTER} 采样 0001：采样频率 $f_{SAMPLING}=f_{MASTER}$，N=2 0010：采样频率 $f_{SAMPLING}=f_{MASTER}$，N=4 0011：采样频率 $f_{SAMPLING}=f_{MASTER}$，N=8 0100：采样频率 $f_{SAMPLING}=f_{MASTER}/2$，N=6 0101：采样频率 $f_{SAMPLING}=f_{MASTER}/2$，N=8 0110：采样频率 $f_{SAMPLING}=f_{MASTER}/4$，N=6 0111：采样频率 $f_{SAMPLING}=f_{MASTER}/4$，N=8 1000：采样频率 $f_{SAMPLING}=f_{MASTER}/8$，N=6 1001：采样频率 $f_{SAMPLING}= f_{MASTER}/ 8$，N=8 1010：采样频率 $f_{SAMPLING}= f_{MASTER}/ 16$，N=5 1011：采样频率 $f_{SAMPLING}= f_{MASTER}/ 16$，N=6 1100：采样频率 $f_{SAMPLING}= f_{MASTER}/ 16$，N=8 1101：采样频率 $f_{SAMPLING}= f_{MASTER}/ 32$，N=5 1110：采样频率 $f_{SAMPLING}= f_{MASTER}/ 32$，N=6 1111：采样频率 $f_{SAMPLING}= f_{MASTER}/ 32$，N=8

ETF	延时时钟数	ETF	延时时钟数
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

19.7.5 中断使能寄存器(PWM1_IER)

寄存器地址：FEC4H

7	6	5	4	3	2	1	0
BIE	TIE	COMIE	CC4IE	CC3IE	CC2IE	CC1IE	UIE

位 7	BIE：允许刹车中断 0：禁止刹车中断； 1：允许刹车中断。
位 6	TIE：触发中断使能 0：禁止触发中断； 1：使能触发中断。
位 5	COMIE：允许 COM 中断 0：禁止 COM 中断； 1：允许 COM 中断。
位 4	CC4IE：允许捕获/比较 4 中断 0：禁止捕获/比较 4 中断； 1：允许捕获/比较 4 中断。
位 3	CC3IE：允许捕获/比较 3 中断 0：禁止捕获/比较 3 中断； 1：允许捕获/比较 3 中断。
位 2	CC2IE：允许捕获/比较 2 中断 0：禁止捕获/比较 2 中断； 1：允许捕获/比较 2 中断。
位 1	CC1IE：允许捕获/比较 1 中断 0：禁止捕获/比较 1 中断； 1：允许捕获/比较 1 中断。
位 0	UIE：允许更新中断 0：禁止更新中断； 1：允许更新中断。

19.7.6 状态寄存器 1(PWM1_SR1)

寄存器地址：FEC5H

7	6	5	4	3	2	1	0
BIF	TIF	COMIF	CC4IF	CC3IF	CC2IF	CC1IF	UIF

位 7	BIF：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。 0：无刹车事件产生； 1：刹车输入上检测到有效电平。

位 6	TIF：触发器中断标记 当发生触发事件（当从模式控制器处于除门控模式外的其它模式时，在 TRGI 输入端检测到有效 边沿，或门控模式下的任一边沿）时由硬件对该位置 1。它由软件清 0。 0：无触发器事件产生； 1：触发中断等待响应。
位 5	COMIF：COM 中断标记 一旦产生 COM 事件（当捕获/比较控制位：CCiE、CCiNE、OCiM 已被更新）该位由硬件置 1。它 由软件清 0。 0：无 COM 事件产生； 1：COM 中断等待响应。
位 4	CC4IF：捕获/比较 4 中断标记 参考 CC1IF 描述。
位 3	CC3IF：捕获/比较 3 中断标记 参考 CC1IF 描述。
位 2	CC2IF：捕获/比较 2 中断标记 参考 CC1IF 描述。
位 1	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（参考 PWM1_CR1 寄存器 的 CMS 位）。它由软件清 0。 0：无匹配发生； 1：PWM1_CNT 的值与 PWM1_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 PWM1_CCR1L 清 0。 0：无输入捕获产生； 1：计数器值已被捕获（拷贝）至 PWM1_CCR1（在 IC1 上检测到与所选极性相同的边沿）。
位 0	UIF：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。 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 PWM1_CR1 寄存器的 UDIS=0，当计数器上溢或下溢时； – 若 PWM1_CR1 寄存器的 UDIS=0、URS=0，当设置 PWM1_EGR 寄存器的 UG 位软件对计数器 CNT 重新初始化时； – 若 PWM1_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 （参考 0 从模式控制寄存器 PWM1_SMCR）。

19.7.7 状态寄存器 2(PWM1_SR2)

寄存器地址：FEC6H

7	6	5	4	3	2	1	0
-			CC4OF	CC3OF	CC2OF	CC1OF	-

位 7:5	保留，始终读为 0。
位 4	CC4OF：捕获/比较 4 重复捕获标记 参见 CC1OF 描述。
位 3	CC3OF：捕获/比较 3 重复捕获标记 参见 CC1OF 描述。
位 2	CC2OF：捕获/比较 2 重复捕获标记 参见 CC1OF 描述。
位 1	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 PWM1_CCR1 寄存器时，CC1IF 的状态已经为 1。
位 0	保留，始终读为 0。

19.7.8 事件产生寄存器（PWM1_EGR）

寄存器地址：FEC7H

7	6	5	4	3	2	1	0
BG	TG	COMG	CC4G	CC3G	CC2G	CC1G	UG

位 7	BG：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。 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。
位 6	TG：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。 0：无动作； 1：PWM1_SR 寄存器的 TIF=1，若开启对应的中断（TIE=1），则产生相应的中断。
位 5	COMG：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。 0：无动作； 1：当 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。
位 4	CC4G：产生捕获/比较 4 事件 参考 CC1G 描述。
位 3	CC3G：产生捕获/比较 3 事件 参考 CC1G 描述。
位 2	CC2G：产生捕获/比较 2 事件 参考 CC1G 描述。
位 1	CC1G：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 PWM1_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。
位 0	UG：产生更新事件 该位由软件置 1，由硬件自动清 0。 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 PWM1_ARR 的值。

19.7.9 捕获/比较模式寄存器 1（PWM1_CCMR1）

寄存器地址：FEC8H

通道可用于输入（捕获模式）或输出（比较模式），通道的方向由相应的 CC1S 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

7	6	5	4	3	2	1	0
OC1CE	OC1M[2:0]			OC1PE	OC1FE	CC1S[1:0]	

位 7	OC1CE：输出比较 1 清零使能 该位用于使能使用 PWM1_TRIG 引脚上的外部事件来清通道 1 的输出信号（OC1REF），参考 17.5.9 在外部事件发生时清除 OCREF 信号 0：OC1REF 不受 ETRF 输入（来自 PWM1_TRIG 引脚）的影响； 1：一旦检测到 ETRF 输入高电平，OC1REF=0。
位 6:4	OC1M[2:0]：输出比较 1 模式 该 3 位定义了输出参考信号 OC1REF 的动作，而 OC1REF 决定了 OC1 的值。OC1REF 是高电平 有效，而 OC1 的有效电平取决于 CC1P 位。 000：冻结。输出比较寄存器 PWM1_CCR1 与计数器 PWM1_CNT 间的比较对 OC1REF 不起作用； 001：匹配时设置通道 1 的输出为有效电平。当计数器 PWM1_CNT 的值与捕获/比较寄存器 1 （PWM1_CCR1）相同时，强制 OC1REF 为高。 010：匹配时设置通道 1 的输出为无效电平。当计数器 PWM1_CNT 的值与捕获/比较寄存器 1 （PWM1_CCR1）相同时，强制 OC1REF 为低。 011：翻转。当 PWM1_CCR1=PWM1_CNT 时，翻转 OC1REF 的电平。 100：强制为无效电平。强制 OC1REF 为低。 101：强制为有效电平。强制 OC1REF 为高。 110：PWM 模式 1－ 在向上计数时，一旦 PWM1_CNT<PWM1_CCR1 时通道 1 为有效电平，否则 为无效电平；在向下计数时，一旦 PWM1_CNT>PWM1_CCR1 时通道 1 为无效电平（OC1REF=0）， 否则为有效电平（OC1REF=1）。 111：PWM 模式 2－ 在向上计数时，一旦 PWM1_CNT<PWM1_CCR1 时通道 1 为无效电平，否则 为有效电平；在向下计数时，一旦 PWM1_CNT>PWM1_CCR1 时通道 1 为有效电平，否则为无效 电平。 注 1：一旦 LOCK 级别设为 3（PWM1_BKR 寄存器中的 LOCK 位）并且 CC1S=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，OC1REF 电平才改变。（参考 17.5.7PWM 模式） 注 3：在有互补输出的通道上，这些位是预装载的。如果 PWM1_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。
位 3	OC1PE：输出比较 1 预装载使能 0：禁止 PWM1_CCR1 寄存器的预装载功能，可随时写入 PWM1_CCR1 寄存器，并且新写入的数 值立即起作用。 1：开启 PWM1_CCR1 寄存器的预装载功能，读写操作仅对预装载寄存器操作，PWM1_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（PWM1_BKR 寄存器中的 LOCK 位）并且 CC1S=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（PWM1_CR1 寄存 器的 OPM=1），它不是必须的。
位 2	OC1FE：输出比较 1 快速使能 该位用于加快 CC 输出对触发输入事件的响应。 0：根据计数器与 CCR1 的值，CC1 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 CC1 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 PWM1 或 PWM2 模式时起作用。
位 1:0	CC1S[1:0]：捕获/比较 1 选择。 这 2 位定义通道的方向（输入/输出），及输入脚的选择： 00：CC1 通道被配置为输出； 01：CC1 通道被配置为输入，IC1 映射在 TI1FP1 上； 10：CC1 通道被配置为输入，IC1 映射在 TI2FP1 上； 11：CC1 通道被配置为输入，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 PWM1_SMCR 寄存器的 TS 位选择）。 注：CC1S 仅在通道关闭时（PWM1_CCER1 寄存器的 CC1E=0）才是可写的。

通道配置为捕获输入模式

7	6	5	4	3	2	1	0
IC1F[3:0]				IC1PSC[1:0]		CC1S[1:0]	

位 7:4	IC1F[3:0]：输入捕获 1 滤波器 这几位定义了 TI1 输入的采样频率及数字滤波器长度。数字滤波器由一个事件计数器组成，只有 发生了 N 个事件后输出的跳变才被认为有效。 0000：无滤波器，$f_{SAMPLING}=f_{MASTER}$ 0001：采样频率 $f_{SAMPLING} = f_{MASTER}$，N=2 0010：采样频率 $f_{SAMPLING} = f_{MASTER}$，N=4 0011：采样频率 $f_{SAMPLING} = f_{MASTER}$，N=8 0100：采样频率 $f_{SAMPLING} = f_{MASTER}$ /2，N=6 0101：采样频率 $f_{SAMPLING} = f_{MASTER}$ /2，N=8 0110：采样频率 $f_{SAMPLING} = f_{MASTER}$ /4，N=6 0111：采样频率 $f_{SAMPLING} = f_{MASTER}$ /4，N=8 1000：采样频率 $f_{SAMPLING} = f_{MASTER}$/8，N=6 1001：采样频率 $f_{SAMPLING} = f_{MASTER}$ /8，N=8 1010：采样频率 $f_{SAMPLING} = f_{MASTER}$ /16，N=5 1011：采样频率 $f_{SAMPLING} = f_{MASTER}$ /16，N=6 1100：采样频率 $f_{SAMPLING} = f_{MASTER}$ /16，N=8 1101：采样频率 $f_{SAMPLING} = f_{MASTER}$ /32，N=5 1110：采样频率 $f_{SAMPLING} = f_{MASTER}$ /32，N=6 1111：采样频率 $f_{SAMPLING} = f_{MASTER}$ /32，N=8 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（PWM1_CR2 寄存器）的值。
位 3:2	IC1PSC[1:0]：输入/捕获 1 预分频器 这 2 位定义了 CC1 输入（IC1）的预分频系数。一旦 CC1E=0（PWM1_CCER 寄存器中），则预分频器复位。 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。
位 1:0	CC1S[1:0]：捕获/比较 1 选择。这 2 位定义通道的方向（输入/输出），及输入脚的选择： 00：CC1 通道被配置为输出； 01：CC1 通道被配置为输入，IC1 映射在 TI1FP1 上；

	10：CC1 通道被配置为输入，IC1 映射在 TI2FP1 上； 11：CC1 通道被配置为输入，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 PWM1_SMCR 寄存器的 TS 位选择）。 注：CC1S 仅在通道关闭时（PWM1_CCER1 寄存器的 CC1E=0）才是可写的。

IC1F	延时时钟数	IC1F	延时时钟数
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

19.7.10 捕获/比较模式寄存器 2（PWM1_CCMR2）

寄存器地址：FEC9H

通道配置为比较输出模式

7	6	5	4	3	2	1	0
OC2CE	OC2M[2:0]			OC2PE	OC2FE	CC2S[1:0]	

位 7	OC2CE：输出比较 2 清零使能 该位用于使能使用 PWM1_TRIG 引脚上的外部事件来清通道 2 的输出信号（OC2REF），参考 17.5.9 在外部事件发生时清除 OCREF 信号 0：OC2REF 不受 ETRF 输入（来自 PWM1_TRIG 引脚）的影响； 1：一旦检测到 ETRF 输入高电平，OC2REF=0。
位 6:4	0C2M[2:0]：输出比较 2 模式
位 3	OC2PE：输出比较 2 预装载使能
位 2	OC2FE：输出比较 2 快速使能
位 1:0	CC2S[1:0]：捕获/比较 2 选择。 该位定义通道的方向（输入/输出），及输入脚的选择： 00：CC2 通道被配置为输出； 01：CC2 通道被配置为输入，IC2 映射在 TI2FP2 上； 10：CC2 通道被配置为输入，IC2 映射在 TI1FP2 上； 11：预留 注：CC2S 仅在通道关闭时（PWM1_CCER1 寄存器的 CC2E=0，CC2NE=0 且已被更新）才是可写 的。

通道配置为捕获输入模式

7	6	5	4	3	2	1	0
IC2F[3:0]				IC2PSC[1:0]		CC2S[1:0]	

位 7:4	IC2F[3:0]：输入捕获 2 滤波器
位 3:2	IC2PSC[1:0]：输入/捕获 2 预分频器

位 1:0	CC2S[1:0]：捕获/比较 2 选择。 这 2 位定义通道的方向（输入/输出），及输入脚的选择： 00：CC2 通道被配置为输出； 01：CC2 通道被配置为输入，IC2 映射在 TI2FP2 上； 10：CC2 通道被配置为输入，IC2 映射在 TI1FP2 上； 11：CC2 通道被配置为输入，IC2 映射在 TRC 上。此模式仅工作在内部触发器输入被选中时（由 PWM1_SMCR 寄存器的 TS 位选择）。 注：CC2S 仅在通道关闭时（PWM1_CCER1 寄存器的 CC2E=0，CC2NE=0 且已被更新）才是可写 的。

19.7.11 捕获/比较模式寄存器 3（PWM1_CCMR3）

寄存器地址：FECAH

通道配置为比较输出模式

7	6	5	4	3	2	1	0
OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]	

位 7	OC3CE：输出比较 3 清零使能 该位用于使能使用 PWM1_TRIG 引脚上的外部事件来清通道 3 的输出信号（OC3REF），参考 17.5.9 在外部事件发生时清除 OCREF 信号 0：OC3REF 不受 ETRF 输入（来自 PWM1_TRIG 引脚）的影响； 1：一旦检测到 ETRF 输入高电平，OC3REF=0。
位 6:4	0C3M[2:0]：输出比较 3 模式
位 3	OC3PE：输出比较 3 预装载使能
位 2	OC3FE：输出比较 3 快速使能
位 1:0	CC3S[1:0]：捕获/比较 3 选择。 该位定义通道的方向（输入/输出），及输入脚的选择： 00：CC3 通道被配置为输出； 01：CC3 通道被配置为输入，IC3 映射在 TI3FP3 上； 10：CC3 通道被配置为输入，IC3 映射在 TI4FP3 上； 11：预留 注：CC3S 仅在通道关闭时（PWM1_CCER2 寄存器的 CC3E=0，CC3NE=0 且已被更新）才是可写 的。

通道配置为捕获输入模式

7	6	5	4	3	2	1	0
IC3F[3:0]				IC3PSC[1:0]		CC3S[1:0]	

位 7:4	IC3F[3:0]：输入捕获 3 滤波器
位 3:2	IC3PSC[1:0]：输入/捕获 3 预分频器
位 1:0	CC3S[1:0]：捕获/比较 3 选择。 这 2 位定义通道的方向（输入/输出），及输入脚的选择： 00：CC3 通道被配置为输出； 01：CC3 通道被配置为输入，IC3 映射在 TI3FP3 上； 10：CC3 通道被配置为输入，IC3 映射在 TI4FP3 上； 11：预留

	注：CC3S 仅在通道关闭时（PWM1_CCER2 寄存器的 CC3E=0，CC3NE=0 且已被更新）才是可写 的。

19.7.12 捕获/比较模式寄存器 4（PWM1_CCMR4）

寄存器地址：FECBH

通道配置为比较输出模式

7	6	5	4	3	2	1	0
OC4CE	OC4M[2:0]			OC4PE	OC4FE	CC4S[1:0]	

位 7	OC4CE：输出比较 4 清零使能 该位用于使能使用 PWM1_TRIG 引脚上的外部事件来清通道 4 的输出信号（OC4REF），参考 17.5.9 在外部事件发生时清除 OCREF 信号 0：OC4REF 不受 ETRF 输入（来自 PWM1_TRIG 引脚）的影响； 1：一旦检测到 ETRF 输入高电平，OC4REF=0。
位 6:4	0C4M[2:0]：输出比较 4 模式
位 3	OC4PE：输出比较 4 预装载使能
位 2	OC4FE：输出比较 4 快速使能
位 1:0	CC4S[1:0]：捕获/比较 4 选择。 该位定义通道的方向（输入/输出），及输入脚的选择： 00：CC4 通道被配置为输出； 01：CC4 通道被配置为输入，IC4 映射在 TI4FP4 上； 10：CC4 通道被配置为输入，IC4 映射在 TI3FP4 上； 11：预留 注：CC4S 仅在通道关闭时（PWM1_CCER2 寄存器的 CC4E=0，CC4NE=0 且已被更新）才是可写 的。

通道配置为捕获输入模式

7	6	5	4	3	2	1	0
IC4F[3:0]				IC4PSC[1:0]		CC4S[1:0]	

位 7:4	IC4F[3:0]：输入捕获 4 滤波器
位 3:2	IC4PSC[1:0]：输入/捕获 4 预分频器
位 1:0	CC4S[1:0]：捕获/比较 4 选择。 这 2 位定义通道的方向（输入/输出），及输入脚的选择： 00：CC4 通道被配置为输出； 01：CC4 通道被配置为输入，IC4 映射在 TI4FP4 上； 10：CC4 通道被配置为输入，IC4 映射在 TI3FP4 上； 11：预留 注：CC4S 仅在通道关闭时（PWM1_CCER2 寄存器的 CC4E=0，CC4NE=0 且已被更新）才是可写 的。

19.7.13 捕获/比较使能寄存器 1（PWM1_CCER1）

寄存器地址：FECCH

7	6	5	4	3	2	1	0
CC2NP	CC2NE	CC2P	CC2E	CC1NP	CC1NE	CC1P	CC1E

位 7	CC2NP：输入捕获/比较 2 互补输出极性。参考 CC1NP 的描述。
位 6	CC2NE：输入捕获/比较 2 互补输出使能。参考 CC1NE 的描述。
位 5	CC2P：输入捕获/比较 2 输出极性。参考 CC1P 的描述。
位 4	CC2E：输入捕获/比较 2 输出使能。参考 CC1E 的描述。
位 3	CC1NP：输入捕获/比较 1 互补输出极性 0：OC1N 高电平有效； 1：OC1N 低电平有效。 注 1：一旦 LOCK 级别（PWM1_BKR 寄存器中的 LCCK 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（PWM1_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。
位 2	CC1NE：输入捕获/比较 1 互补输出使能 0：关闭－ OC1N 禁止输出，因此 OC1N 的输出电平依赖于 MOE、OSSI、OSSR、OIS1、OIS1N 和 CC1E 位的值。 1：开启－ OC1N 信号输出到对应的输出引脚，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（PWM1_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。
位 1	CC1P：输入捕获/比较 1 输出极性 CC1 通道配置为输出： 0：OC1 高电平有效 1：OC1 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.
位 0	CC1E 通道配置为输入（参考图 61）： 0：捕捉发生在 TI1F 的高电平或上升沿； 1：捕捉发生在 TI1F 的低电平或下降沿。 注 1：一旦 LOCK 级别（PWM1_BKR 寄存器中的 LCCK 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（PWM1_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。

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（与定时器断开）	输出禁止（与定时器断开）

		0	0	1	输出禁止（与定时器断开）	OCiREF + 极性， OCiN= OCiREF xor CCiNP
		0	1	0	OCiREF + 极性， OCi= OCiREF xor CCiP	输出禁止（与定时器断开）
		0	1	1	OCiREF + 极性 + 死区	OCiREF 反相 + 极性 + 死区
		1	0	0	输出禁止（与定时器断开）	输出禁止（与定时器断开）
		1	0	1	关闭状态（输出使能且为无效 电平）OCi=CCiP	OCiREF + 极性， OCiN= OCiREF xor CCiNP
		1	1	0	OCiREF + 极性， OCi= OCiREF xor CCiP	关闭状态（输出使能且为无效电 平） OCiN=CCiNP
		1	1	1	OCiREF + 极性 + 死区	OCiREF 反相 + 极性 + 死区
0	0	X	X	X	输出禁止（与定时器断开）	
	0					
	0					
	0					
	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。	
	1					
	1					
	1					

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。

19.7.14 捕获/比较使能寄存器 2（PWM1_CCER2）

寄存器地址：FECDH

7	6	5	4	3	2	1	0
-		CC4P	CC4E	CC3NP	CC3NE	CC3P	CC3E

位 7:6	保留。
位 5	CC4P：输入捕获/比较 4 输出极性。参考 CC1P 的描述。
位 4	CC4E：输入捕获/比较 4 输出使能。参考 CC1E 的描述。
位 3	CC3NP：输入捕获/比较 3 互补输出极性。参考 CC1NP 的描述。
位 2	CC3NE：输入捕获/比较 3 互补输出使能。参考 CC1NE 的描述。
位 1	CC3P：输入捕获/比较 3 输出极性。参考 CC1P 的描述。
位 0	CC3E：输入捕获/比较 3 输出使能。参考 CC1E 的描述。

19.7.15 计数器高 8 位（PWM1_CNTRH）

寄存器地址：FECEH

7	6	5	4	3	2	1	0
CNT[15:8]							

位 7:0	CNT[15:8]：计数器的高 8 位值

19.7.16 计数器低 8 位（PWM1_CNTRL）

寄存器地址：FECFH

7	6	5	4	3	2	1	0
CNT[7:0]							

位 7:0	CNT[7:0]：计数器的低 8 位值

19.7.17 预分频器高 8 位（PWM1_PSCRH）

寄存器地址：FED0H

7	6	5	4	3	2	1	0
PSC[15:8]							

位 7:0	PSC[15:8]：预分频器的高 8 位值预分频器用于对 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。

19.7.18 预分频器低 8 位（PWM1_PSCRL）

寄存器地址：FED1H

7	6	5	4	3	2	1	0
PSC[7:0]							

位 7:0	PSC[7:0]：预分频器的低 8 位值

19.7.19 自动重装载寄存器高 8 位（PWM1_ARRH）

寄存器地址：FED2H

7	6	5	4	3	2	1	0
ARR[15:8]							

位 7:0	ARR[15:8]：自动重装载高 8 位值 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空时，计数器不工作。

19.7.20 自动重装载寄存器低 8 位（PWM1_ARRL）

寄存器地址：FED3H

7	6	5	4	3	2	1	0
ARR[7:0]							

位 7:0	ARR[7:0]：自动重装载低 8 位值

19.7.21 重复计数器寄存器（PWM1_RCR）

寄存器地址：FED4H

7	6	5	4	3	2	1	0
REP[7:0]							

位 7:0	REP[7:0]：重复计数器值 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 PWM1_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；

19.7.22 捕获/比较寄存器 1 高 8 位（PWM1_CCR1H）

寄存器地址：FED5H

7	6	5	4	3	2	1	0
CCR1[15:8]							

位 7:0	CCR1[15:8]：捕获/比较 1 的高 8 位值 若 CC1 通道配置为输出（PWM1_CCMR1 的 CC1S 位）： CCR1 包含了装入当前捕获/比较 1 寄存器的值（预装载值）。如果在 PWM1_CCMR1 寄存器（OC1PE 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 1 寄存器中。当前捕获/比较寄存器的值同计数器 PWM1_CNT 的值相比较，并在 OC1 端口上产生输出信号。 若 CC1 通道配置为输入：CCR1 包含了上一次输入捕获 1 事件（IC1）发生时的计数器值（此时该寄存器为只读）。

19.7.23 捕获/比较寄存器 1 低 8 位（PWM1_CCR1L）

寄存器地址：FED6H

7	6	5	4	3	2	1	0
CCR1[7:0]							

位 7:0	CCR1[7:0]：捕获/比较 1 的低 8 位值

19.7.24 捕获/比较寄存器 2 高 8 位（PWM1_CCR2H）

寄存器地址：FED7H

7	6	5	4	3	2	1	0
CCR2[15:8]							

位 7:0	CCR2[15:8]：捕获/比较 2 的高 8 位值 若 CC2 通道配置为输出（PWM1_CCMR2 的 CC2S 位）： CCR2 包含了装入当前捕获/比较 2 寄存器的值（预装载值）。 如果在 PWM1_CCMR2 寄存器（OC2PE 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 1 寄存器中。 当前捕获/比较寄存器的值同计数器 PWM1_CNT 的值相比较，并在 OC2 端口上产生输出信号。若 CC2 通道配置为输入：CCR2 包含了由上一次输入捕获 2 事件（IC2）传输的计数器值（此时该寄存器为只读）。

19.7.25 捕获/比较寄存器 2 低 8 位（PWM1_CCR2L）

寄存器地址：FED8H

7	6	5	4	3	2	1	0
CCR2[7:0]							

位 7:0	CCR2[7:0]：捕获/比较 2 的低 8 位值

19.7.26 捕获/比较寄存器 3 高 8 位（PWM1_CCR3H）

寄存器地址：FED9H

7	6	5	4	3	2	1	0
CCR3[15:8]							

位 7:0	CCR3[15:8]：捕获/比较 3 的高 8 位值 若 CC3 通道配置为输出（PWM1_CCMR3 的 CC3S 位）： CCR3 包含了装入当前捕获/比较 3 寄存器的值（预装载值）。 如果在 PWM1_CCMR3 寄存器（OC3PE 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 1 寄存器中。当前捕获/比较寄存器的值同计数器 PWM1_CNT 的值相比较，并在 OC3 端口上产生输出信号。 若 CC3 通道配置为输入： CCR3 包含了由上一次输入捕获 3 事件（IC3）传输的计数器值（此时该寄存器为只读）。

19.7.27 捕获/比较寄存器 3 低 8 位（PWM1_CCR3L）

寄存器地址：FEDAH

7	6	5	4	3	2	1	0
CCR3[7:0]							

位 7:0	CCR3[7:0]：捕获/比较 3 的低 8 位值

19.7.28 捕获/比较寄存器 4 高 8 位（PWM1_CCR4H）

寄存器地址：FEDBH

7	6	5	4	3	2	1	0
CCR4[15:8]							

位 7:0	CCR4[15:8]：捕获/比较 4 的高 8 位值 若 CC4 通道配置为输出（PWM1_CCMR4 的 CC4S 位）： CCR4 包含了装入当前捕获/比较 4 寄存器的值（预装载值）。 如果在 PWM1_CCMR4 寄存器（OC4PE 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 1 寄存器中。 当前捕获/比较寄存器的值同计数器 PWM1_CNT 的值相比较，并在 OC4 端口上产生输出信号。若 CC4 通道配置为输入：CCR4 包含了由上一次输入捕获 4 事件（IC4）传输的计数器值（此时该寄存器为只读）。

19.7.29 捕获/比较寄存器 4 低 8 位（PWM1_CCR4L）

寄存器地址：FEDCH

7	6	5	4	3	2	1	0
CCR4[7:0]							

位 7:0	CCR4[7:0]：捕获/比较 4 的低 8 位值

19.7.30 刹车寄存器（PWM1_BKR）

寄存器地址：FEDDH

7	6	5	4	3	2	1	0
MOE	AOE	BKP	BKE	OSSR	OSSI	LOCK	

位 7	MOE: 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。 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（PWM1_CCERX 寄存器的 CCIE 位），则使能 OC 和 OCN 输出。
位 6	AOE: 自动输出使能 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（PWM1_BKR 寄存器中的 LOCK 位）设为 1，则该位不能被修改。
位 5	BKP: 刹车输入极性 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（PWM1_BKR 寄存器中的 LOCK 位）设为 1，则该位不能被修改。
位 4	BKE: 刹车功能使能

	0：禁止刹车输入（BRK）； 1：开启刹车输入（BRK）。 注：一旦 LOCK 级别（PWM1_BKR 寄存器中的 LOCK 位）设为 1，则该位不能被修改。
位 3	OSSR: 运行模式下“关闭状态”选择 该位用于当 MOE=1 且通道为互补输出时。参考 OC/OCN 使能的详细说明（参见 17.7.13）。 0：当定时器不工作时，禁止 OC/OCN 输出（OC/OCN 使能输出信号=0）； 1：当定时器不工作时，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（PWM1_BKR 寄存器中的 LOCK 位）设为 2，则该位不能被修改。
位 2	OSSI: 空闲模式下“关闭状态”选择 该位用于当 MOE=0 且通道设为输出时。参考 OC/OCN 使能的详细说明（参见 17.7.13）。 0：当定时器不工作时，禁止 OC/OCN 输出（OC/OCN 使能输出信号=0）； 1：当定时器不工作时，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（PWM1_BKR 寄存器中的 LOCK 位）设为 2，则该位不能被修改。
位 1:0	LOOK[1:0]: 锁定设置 该位为防止软件错误而提供写保护。 00：锁定关闭，寄存器无写保护； 01：锁定级别 1，不能写入 PWM1_BKR 寄存器的 BKE、BKP、AOE 位和 PWM1_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（一旦相关通道通过 CCIS 位设为输出，CC 极性位是 PWM1_CCERX 寄存器的 CCIP 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（一旦相关通道通过 CCIS 位设为输出，CC 控制位是 PWM1_CCMRx 寄存器的 OCIM/OCIPE 位）； 注：在系统复位后，只能写一次 LOCK 位，一旦写入 PWM1_BKR 寄存器，则其内容保持不变直 至复位。

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 PWM1_BKR 寄存器时必须对它们进行设置。

19.7.31 死区寄存器（PWM1_DTR）

寄存器地址：FEDEH

7	6	5	4	3	2	1	0
DTG[7:0]							

位 7:0	DTG[7:0]：死区发生器设置。这些位定义了插入互补输出之间的死区持续时间。假设 DT 表示其持续时间，tCK_PSC 为 PWM1 的时钟脉冲： DTG[7:5] = 0xx → DT = DTG[7:0] * tdtg，其中： tdtg=tCK_PSC.(f1) DTG[7:5] = 10x → DT= (64+DTG[5:0]) * tdtg，其中：tdtg = 2 * tCK_PSC.(f2) DTG[7:5] = 110 → DT= (32+DTG[4:0]) * tdtg，其中：tdtg =8 * tCK_PSC.(f3) DTG[7:5] = 111 → DT= (32+DTG[4:0]) * tdtg，其中：tdtg =16 * tCK_PSC.(f4)

	举列: 如果 tCK_PSC =125ns（8MHz）,可能的死区时间为： DTG[7:0] = 0~7Fh，0 到 15875ns，步长时间为 125ns（参考 f1） DTG[7:0] = 80h~BFh，16μs 到 31750ns，步长时间为 250ns（参考 f2） DTG[7:0] = C0h~DFh，32μs 到 63μs，步长时间为 1μs（参考 f3） DTG[7:0] = E0h~FFh，64μs 到 126μs，步长时间为 2μs（参考 f4） 注：一旦 LOCK 级别（PWM1_BKR 寄存器中的 LOCK 位）设为 1、2 或 3，则不能修改这些位。

19.7.32 输出空闲状态寄存器（PWM1_OISR）

寄存器地址：FEDFH

7	6	5	4	3	2	1	0
-	OIS4	OIS3N	OIS3	OIS2N	OIS2	OIS1N	OIS1

位 7	保留，始终读为 0。
位 6	OIS4：输出空闲状态 4（OC4 输出）。
位 5	OIS3N：输出空闲状态 3（OC3N 输出）。
位 4	OIS3：输出空闲状态 3（OC3 输出）。
位 3	OIS2N：输出空闲状态 2（OC2N 输出）。
位 2	OIS2：输出空闲状态 2（OC2 输出）。
位 1	OIS1N：输出空闲状态 1（OC1N 输出）。 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。 注：已经设置了 LOCK（PWM1_BKR 寄存器）级别 1、2 或 3 后，该位不能被修改。
位 0	OIS1：输出空闲状态 1（OC1 输出）。 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。 注：已经设置了 LOCK（PWM1_BKR 寄存器）级别 1、2 或 3 后，该位不能被修改。

20 增强型双数据指针

STC8 系列的单片机内部集成了两组 16 位的数据指针。通过程序控制, 可实现数据指针自动递增或递减功能以及两组数据指针的自动切换功能

相关的特殊功能寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DPL	数据指针（低字节）	82H									0000,0000
DPH	数据指针（高字节）	83H									0000,0000
DPL1	第二组数据指针（低字节）	E4H									0000,0000
DPH1	第二组数据指针（高字节）	E5H									0000,0000
DPS	DPTR 指针选择器	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
TA	DPTR 时序控制寄存器	AEH									0000,0000

第 1 组 16 位数据指针寄存器（DPTR0）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPL	82H								
DPH	83H								

DPL为低8位数据（低字节）

DPH为高8位数据（高字节）

DPL和DPH组合为第一组16位数据指针寄存器DPTR0

第 2 组 16 位数据指针寄存器（DPTR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPL1	E4H								
DPH1	E5H								

DPL1为低8位数据（低字节）

DPH1为高8位数据（高字节）

DPL1和DPH1组合为第二组16位数据指针寄存器DPTR1

数据指针控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPS	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL

ID1：控制DPTR1自动递增方式

0：DPTR1 自动递增

1：DPTR1 自动递减

ID0：控制DPTR0自动递增方式

0：DPTR0 自动递增

1：DPTR0 自动递减

TSL：DPTR0/DPTR1自动切换控制（自动对SEL进行取反）

0：关闭自动切换功能

1：使能自动切换功能

当 TSL 位被置 1 后，每当执行完成相关指令后，系统会自动将 SEL 位取反。

与 TSL 相关的指令包括如下指令：

```
MOV     DPTR,#data16
INC     DPTR
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
```

AU1/AU0：使能DPTR1/DPTR0使用ID1/ID0控制位进行自动递增/递减控制

0：关闭自动递增/递减功能

1：使能自动递增/递减功能

注意：在写保护模式下，AU0 和 AU1 位无法直接单独使能，若单独使能 AU1 位，则 AU0 位也会被自动使能，若单独使能 AU0，没有效果。若需要单独使能 AU1 或者 AU0，则必须使用 TA 寄存器触发 DPS 的保护机制（参考 TA 寄存器的说明）。另外，只有执行下面的 3 条指令后才会对 DPTR0/DPTR1 进行自动递增/递减操作。3 条相关指令如下：

```
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
```

SEL：选择DPTR0/DPTR1作为当前的目标DPTR

0：选择 DPTR0 作为目标 DPTR

1：选择 DPTR1 作为目标 DPTR

SEL 选择目标 DPTR 对下面指令有效：

```
MOV     DPTR,#data16
INC     DPTR
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
JMP     @A+DPTR
```

数据指针控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TA	AEH								

TA寄存器是对DPS寄存器中的AU1和AU0进行写保护的。由于程序无法对DPS中的AU1和AU0进行单独的写入，所以当需要单独使能AU1或者AU0时，必须使用TA寄存器进行触发。TA寄存器是只写寄存器。当需要对AU1或者AU0进行单独使能时，必须按照如下的步骤进行操作：

```
CLR     EA          ;关闭中断（必需）
MOV     TA,#0AAH    ;写入触发命令序列 1
                    ;此处不能有其他任何指令
MOV     TA,#55H     ;写入触发命令序列 2
                    ;此处不能有其他任何指令
MOV     DPS,#xxH    ;写保护暂时关闭，可向 DPS 中写入任何值
                    ;DSP 再次进行写保护状态
SETB    EA          ;打开中断（如有必要）
```

20.1 范例程序

20.1.1 示例代码 1

将程序空间 1000H～1003H 的 4 个字节数据反向复制到扩展 RAM 的 0100H～0103H 中，即

C:1000H -> X:0103H
C:1001H -> X:0102H
C:1002H -> X:0101H
C:1003H -> X:0100H

汇编代码

```
        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

        MOV     DPS,#00100000B      ;使能 TSL,并选择 DPTR0
        MOV     DPTR,#1000H         ;将 1000H 写入 DPTR0 中,执行完成后选择 DPTR1 为 DPTR
        MOV     DPTR,#0103H         ;将 0103H 写入 DPTR1 中
        MOV     DPS,#10111000B      ;设置 DPTR1 为递减模式,DPTR0 为递加模式,使能 TSL 以及
                                    ;AU0 和 AU1,并选择 DPTR0 为当前的 DPTR
        MOV     R7,#4               ;设置数据复制个数
COPY_NEXT:
        CLR     A                   ;
        MOVC    A,@A+DPTR           ;从 DPTR0 所指的程序空间读取数据,
                                    ;完成后 DPTR0 自动加 1 并将 DPTR1 设置为下一个目标 DPTR
        MOVX    @DPTR,A             ;将 ACC 的数据写入到 DPTR1 所指的 XDATA 中,
                                    ;完成后 DPTR1 自动减 1 并将 DPTR0 设置为下一个目标 DPTR
        DJNZ    R7,COPY_NEXT        ;

        SJMP    $

        END
```

20.1.2 示例代码 2

将扩展 RAM 的 0100H～0103H 中的数据依次发送到 P0 口

汇编代码

```
        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #3FH

        CLR     EA                  ;关闭中断
        MOV     TA,#0AAH            ;写入 DPS 写保护触发命令 1
        MOV     TA,#55H             ;写入 DPS 写保护触发命令 2
        MOV     DPS,#00001000B      ;DPTR0 递增,单独使能 AU0,并选择 DPTR0
        SETB    EA                  ;打开中断
        MOV     DPTR,#0100H         ;将 0100H 写入 DPTR0 中
```

```
        MOVX    A,@DPTR                 ;从 DPTR0 所指的 XRAM 读取数据,完成后 DPTR0 自动加 1
        MOV     P0,A                    ;数据输出到 P0 口
        MOVX    A,@DPTR                 ;从 DPTR0 所指的 XRAM 读取数据,完成后 DPTR0 自动加 1
        MOV     P0,A                    ;数据输出到 P0 口
        MOVX    A,@DPTR                 ;从 DPTR0 所指的 XRAM 读取数据,完成后 DPTR0 自动加 1
        MOV     P0,A                    ;数据输出到 P0 口
        MOVX    A,@DPTR                 ;从 DPTR0 所指的 XRAM 读取数据,完成后 DPTR0 自动加 1
        MOV     P0,A                    ;数据输出到 P0 口

        SJMP    $

        END
```

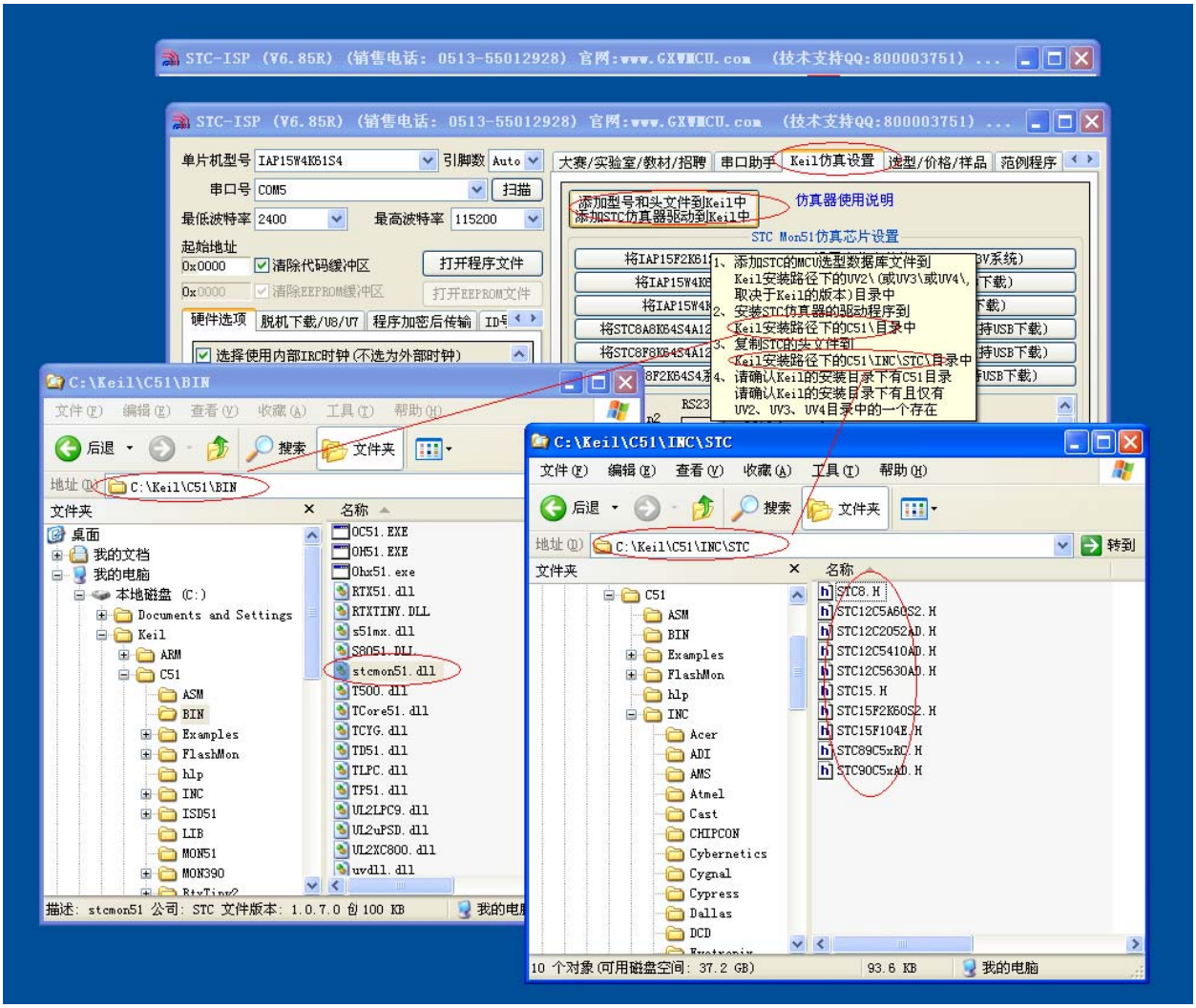
附录A　应用注意事项

关于 STC8H 系列 IO 口的注意事项

1. **STC8H 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. **STC8H 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**

附录B　STC仿真器使用指南

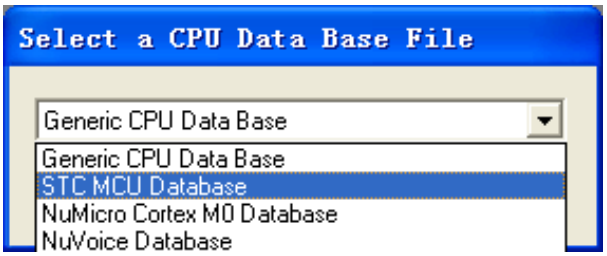
1、安装 Keil 版本的仿真驱动



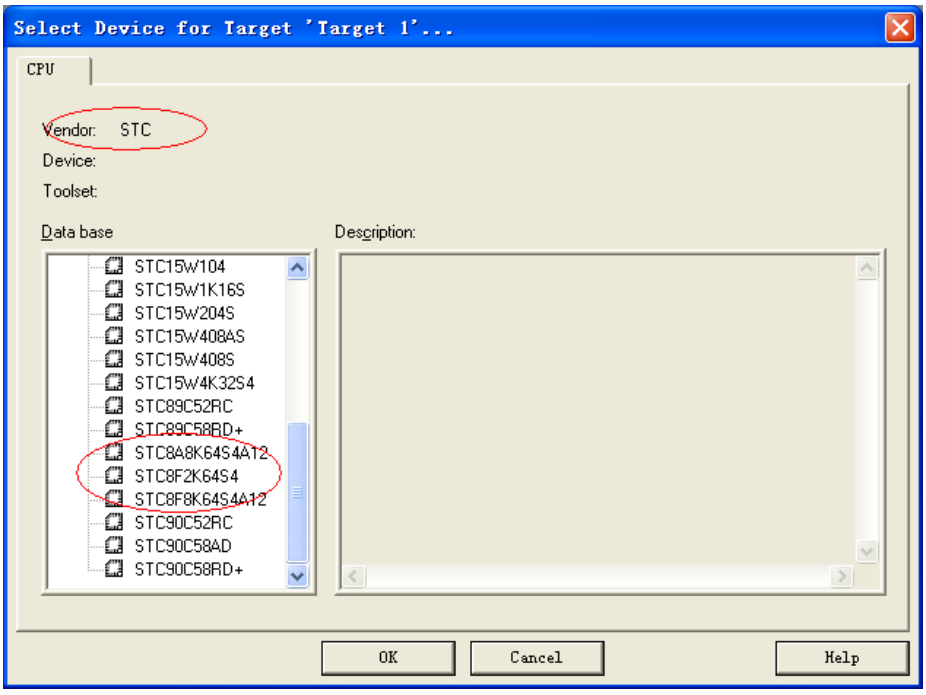
如上图，首先选择“Keil 仿真设置”页面，点击“添加 MCU 型号到 Keil 中”，在出现的如下的目录选择窗口中，定位到 Keil 的安装目录(一般可能为“C:\Keil\”)，“确定”后出现下图中右边所示的提示信息，表示安装成功。添加头文件的同时也会安装 STC 的 Monitor51 仿真驱动 STCMON51.DLL，驱动与头文件的的安装目录如上图所示。

2、在 Keil 中创建项目

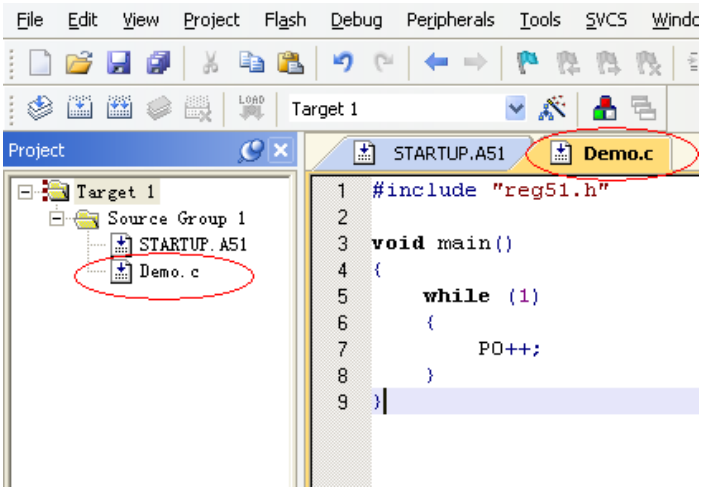
若第一步的驱动安装成功，则在 Keil 中新建项目时选择芯片型号时，便会有“STC MCU Database”的选择项，如下图



然后从列表中选择响应的 MCU 型号，我们在此选择“STC8A8K64S4A12”的型号，点击“确定”完成选择



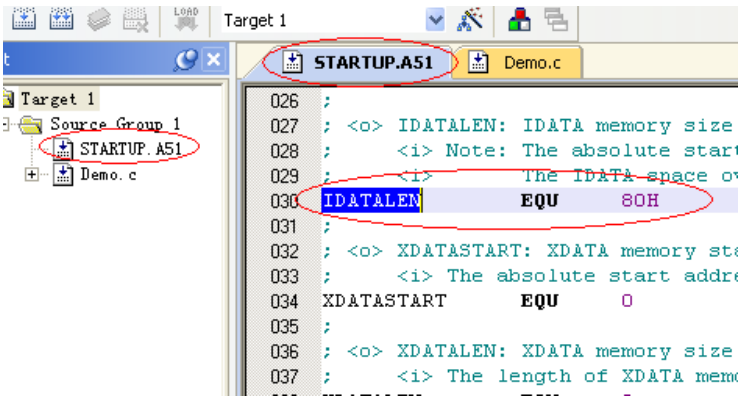
添加源代码文件到项目中，如下图：



保存项目，若编译无误，则可以进行下面的项目设置了

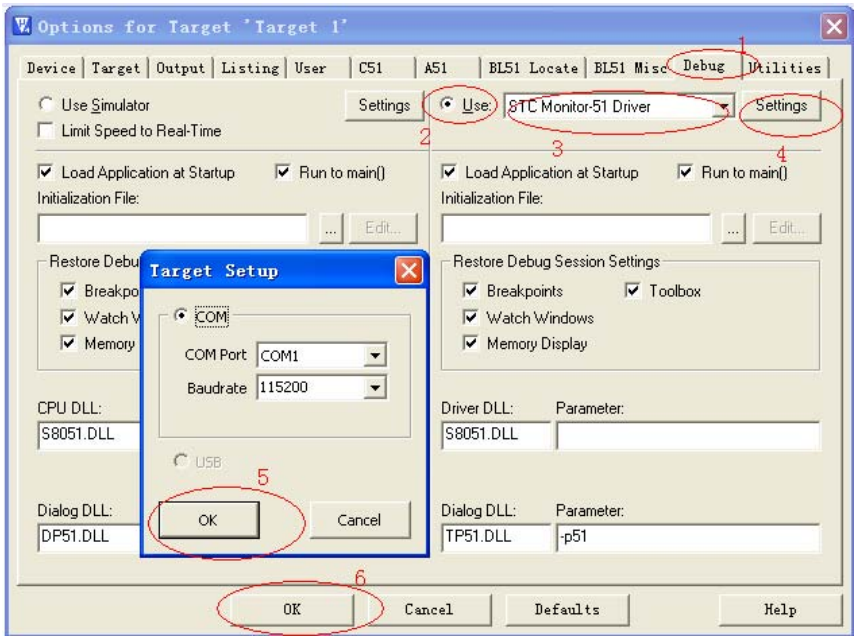
附加说明一点：

当创建的是 C 语言项目，且有将启动文件“STARTUP.A51”添加到项目中时，里面有一个命名为“IDATALEN”的宏定义，它是用来定义 IDATA 大小的一个宏，默认值是 128，即十六进制的 80H，同时它也是启动文件中需要初始化为 0 的 IDATA 的大小。所以当 IDATA 定义为 80H，那么 STARTUP.A51 里面的代码则会将 IDATA 的 00-7F 的 RAM 初始化为 0；同样若将 IDATA 定义为 0FFH，则会将 IDATA 的 00-FF 的 RAM 初始化为 0。



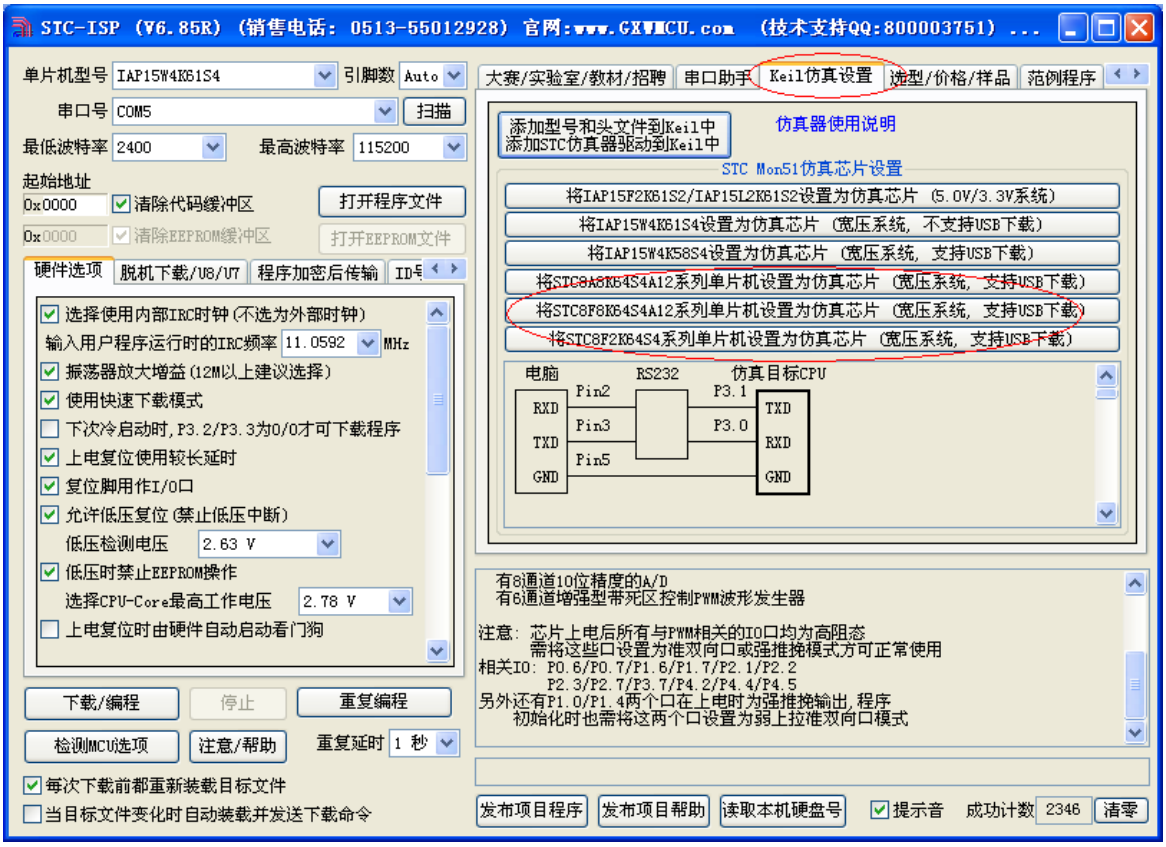
虽然 STC8 系列的单片机的 IDATA 大小为 256 字节（00-7F 的 DATA 和 80H-FFH 的 IDATA），但由于在 RAM 的最后 17 个字节有写入 ID 号以及相关的测试参数，若用户在程序中需要使用这一部分数据，则一定不要将 IDATALEN 定义为 256。

3、项目设置，选择 STC 仿真驱动



如上图，首先进入到项目的设置页面，选择“Debug”设置页，第 2 步选择右侧的硬件仿真“Use …”，第 3 步，在仿真驱动下拉列表中选择“STC Monitor-51 Driver”项，然后点击“Settings”按钮，进入下面的设置画面，对串口的端口号和波特率进行设置，波特率一般选择 115200。到此设置便完成了。

4、创建仿真芯片



准备一颗 STC8A 系列或者 STC8F 系列的芯片，并通过下载板连接到电脑的串口，然后如上图，选择正确的芯片型号，然后进入到“Keil 仿真设置”页面，点击相应型号的按钮，当程序下载完成后仿真器便制作完成了。

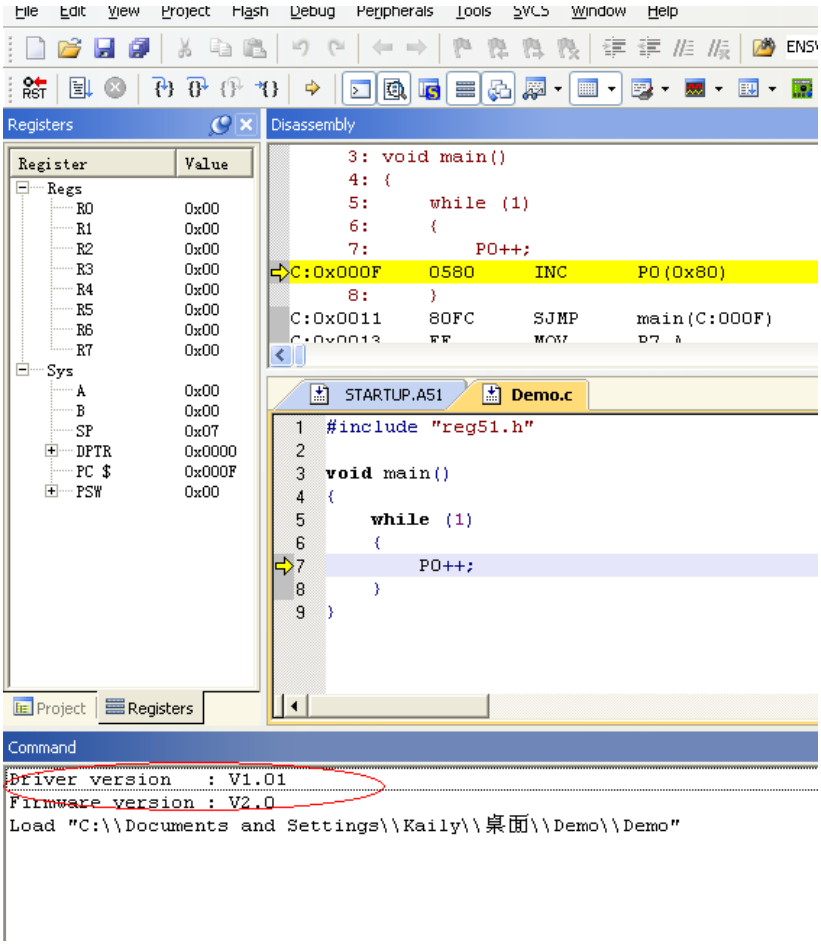
5、开始仿真

将制作完成的仿真芯片通过串口与电脑相连接。

将前面我们所创建的项目编译至没有错误后，按“Ctrl+F5”开始调试。

若硬件连接无误的话，将会进入到类似于下面的调试界面，并在命令输出窗口显示当前的仿真驱动版本号和当前仿真监控代码固件的版本号

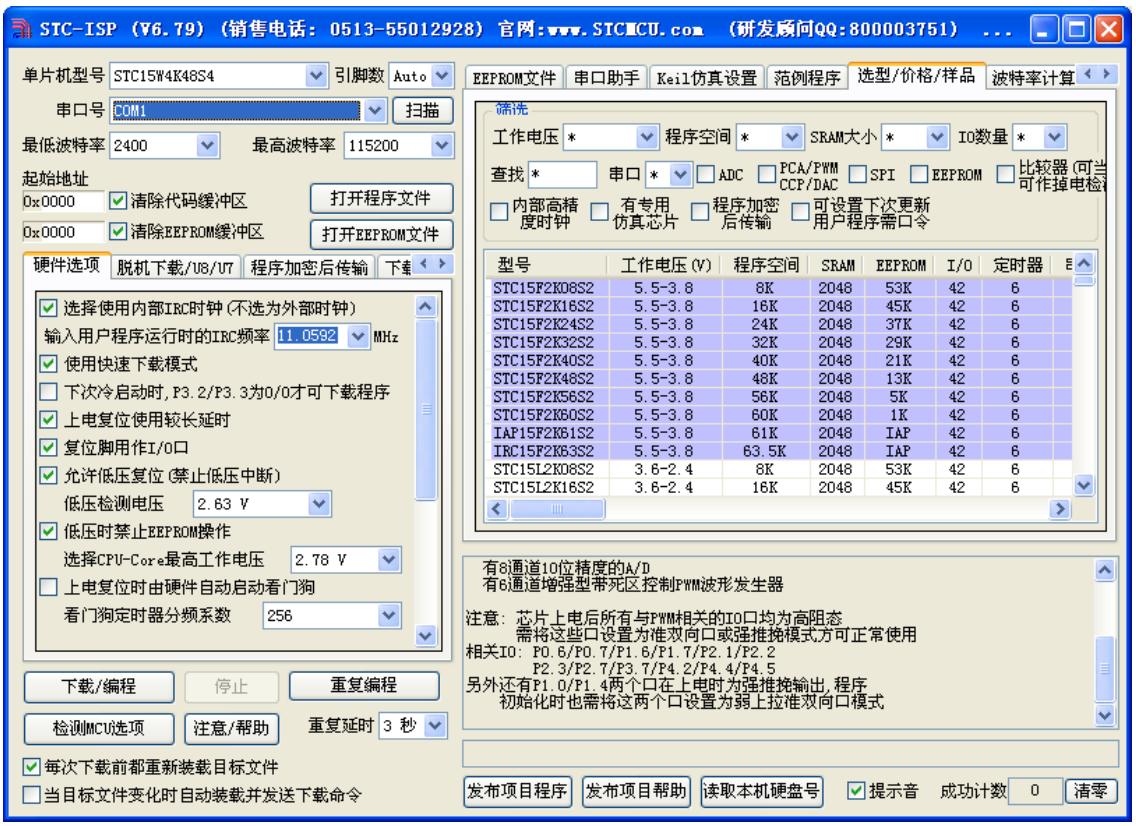
断点设置的个数目前最大允许 20 个（理论上可设置任意个，但是断点设置得过多会影响调试的速度）。



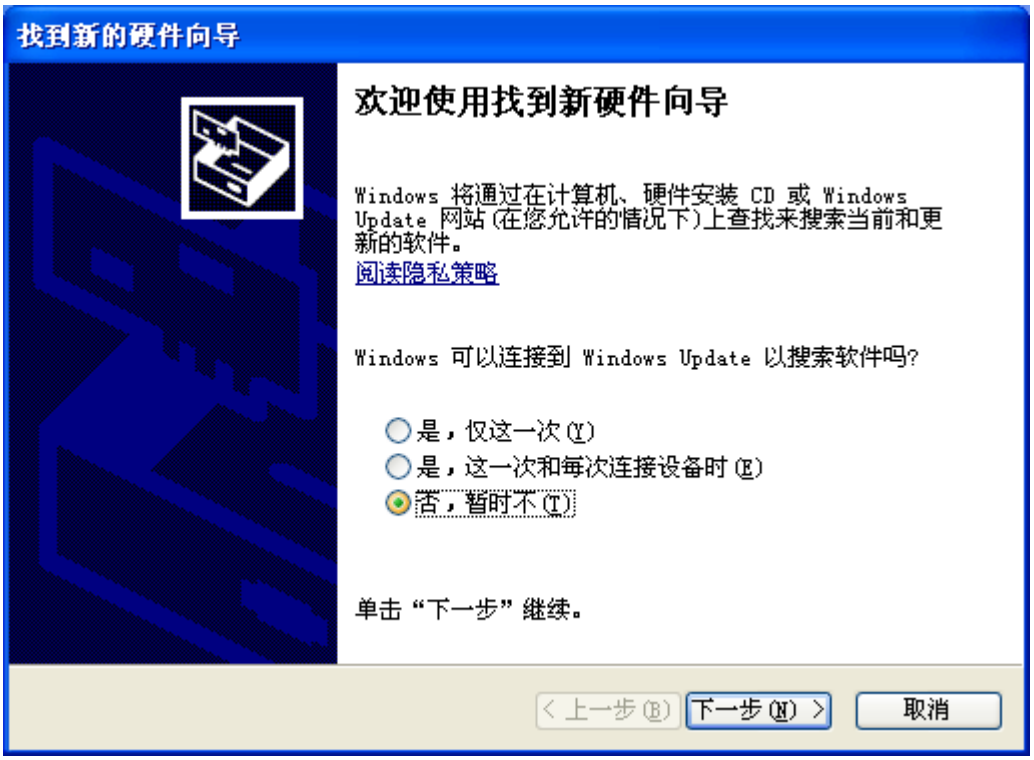
附录C　STC-USB驱动程序安装说明

Windows XP 安装方法

打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录

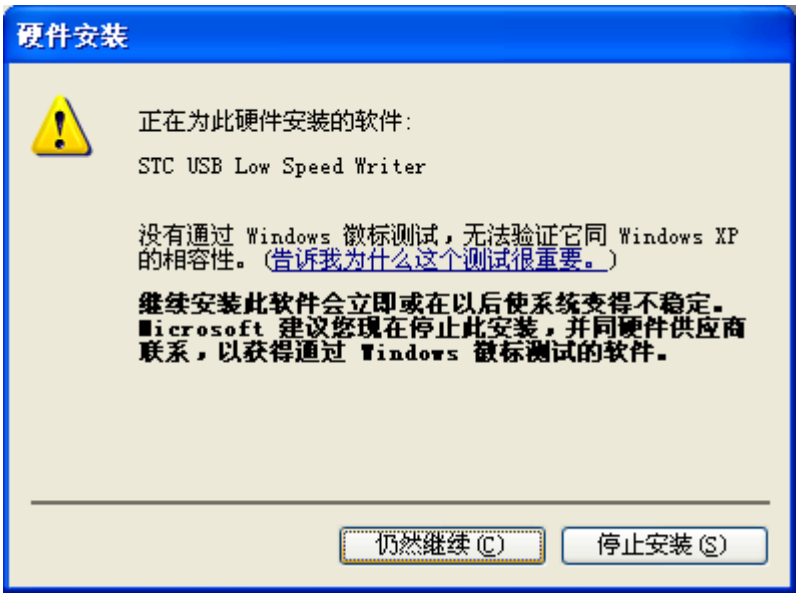


插入 USB 设备，系统找到设备后自动弹出如下对话框，选择其中的“否，暂时不”项



在下面的对话框中选择“自动安装软件(推荐)”项

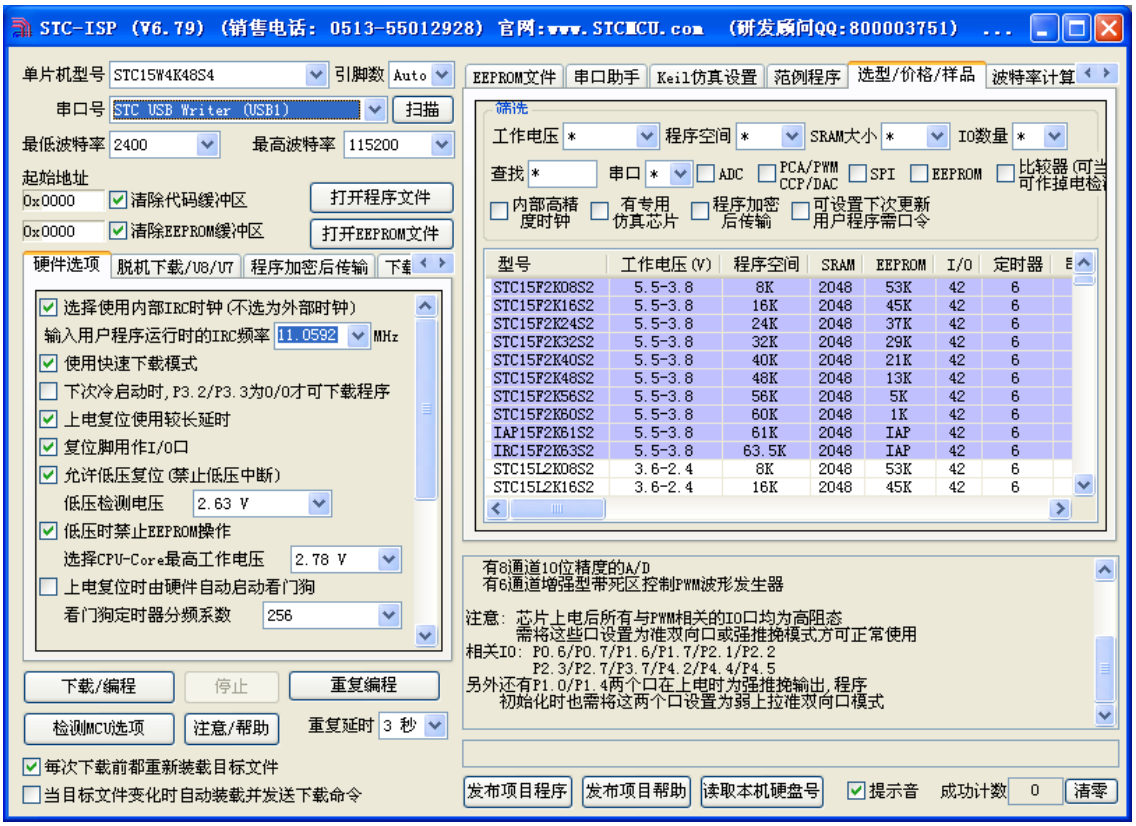
在弹出的下列对话框中，选择“仍然继续”按钮



接下系统会自动安装驱动，如下图

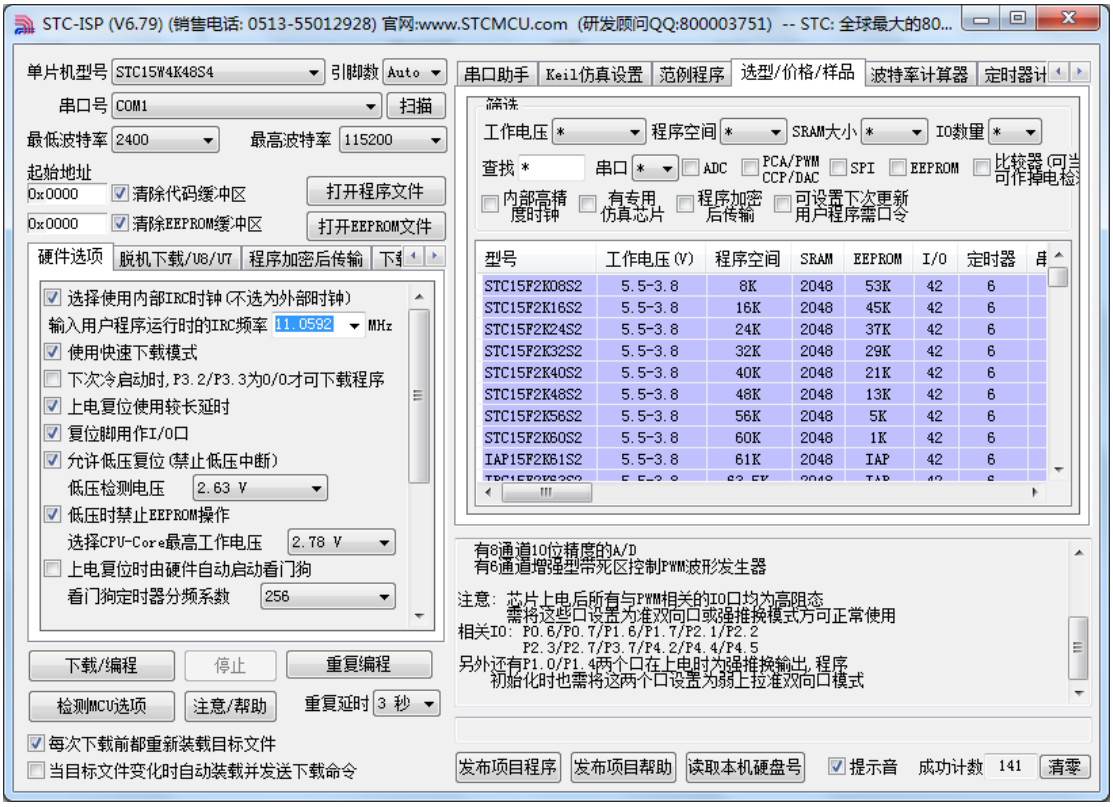
出现下面的的对话框表示驱动安装完成

此时，之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)” ，如下图：

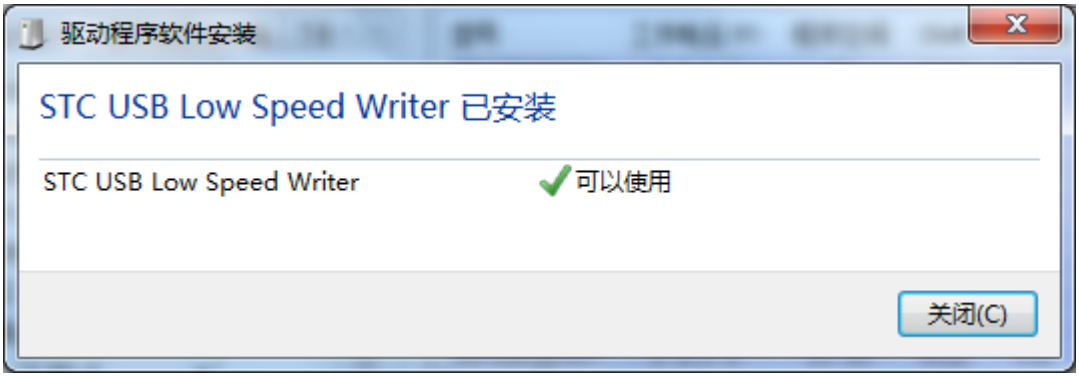


Windows 7（32 位）安装方法

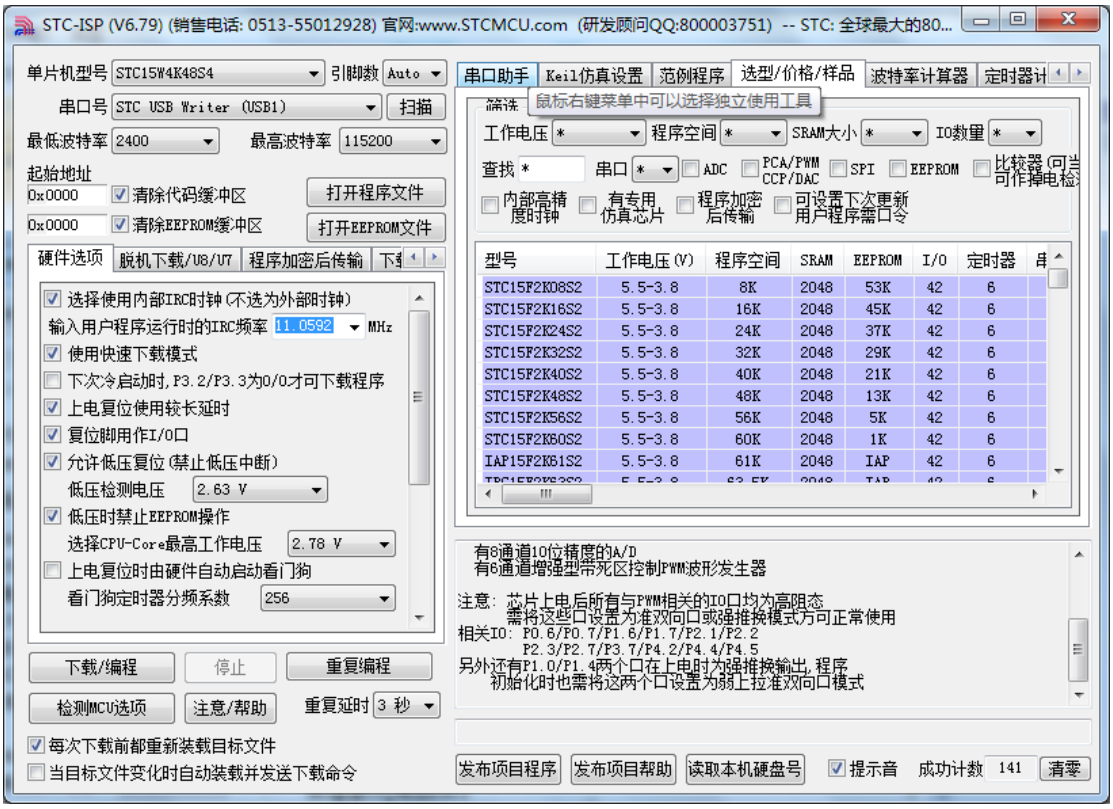
打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录



插入 USB 设备，系统找到设备后会自动安装驱动。安装完成后会有如下的提示框。



此时，之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：

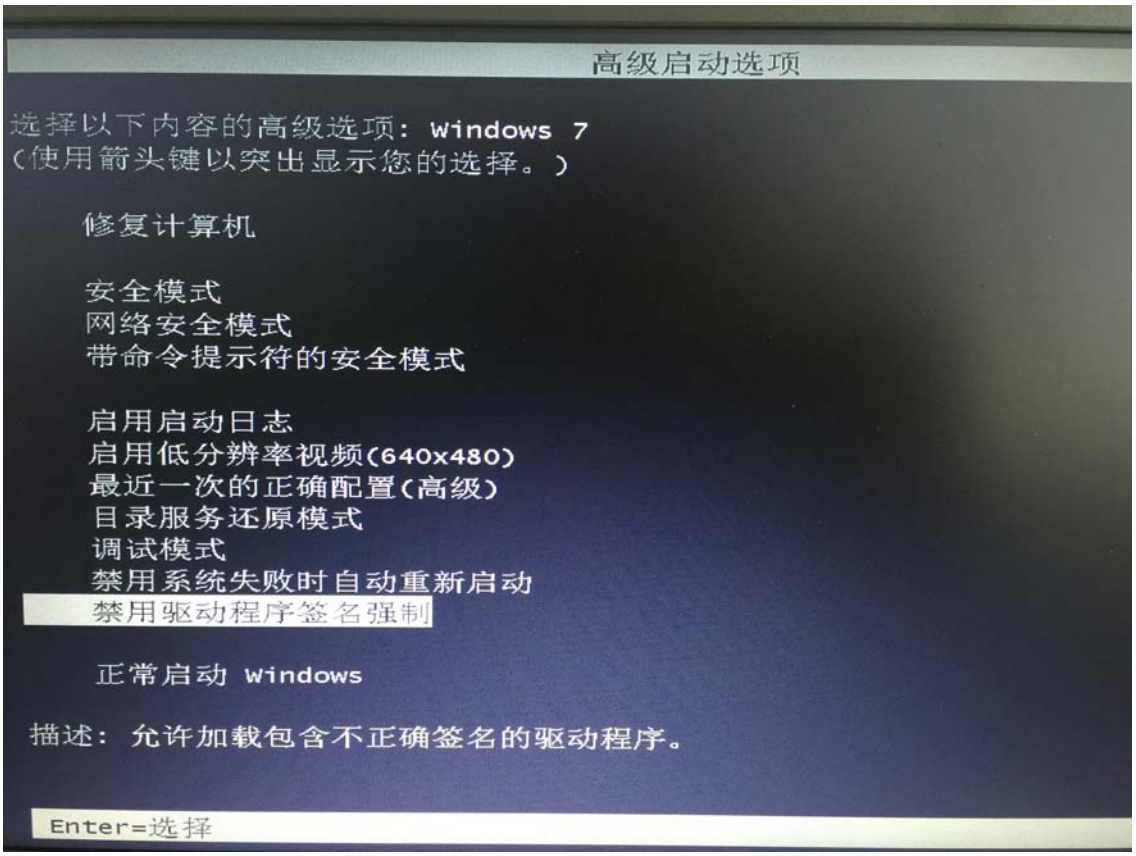


注：若 Windows 7 下，系统并没有自动安装驱动，则驱动的安装方法请参考 **Windows 8（32 位）的安装方法**

Windows 7（64 位）安装方法

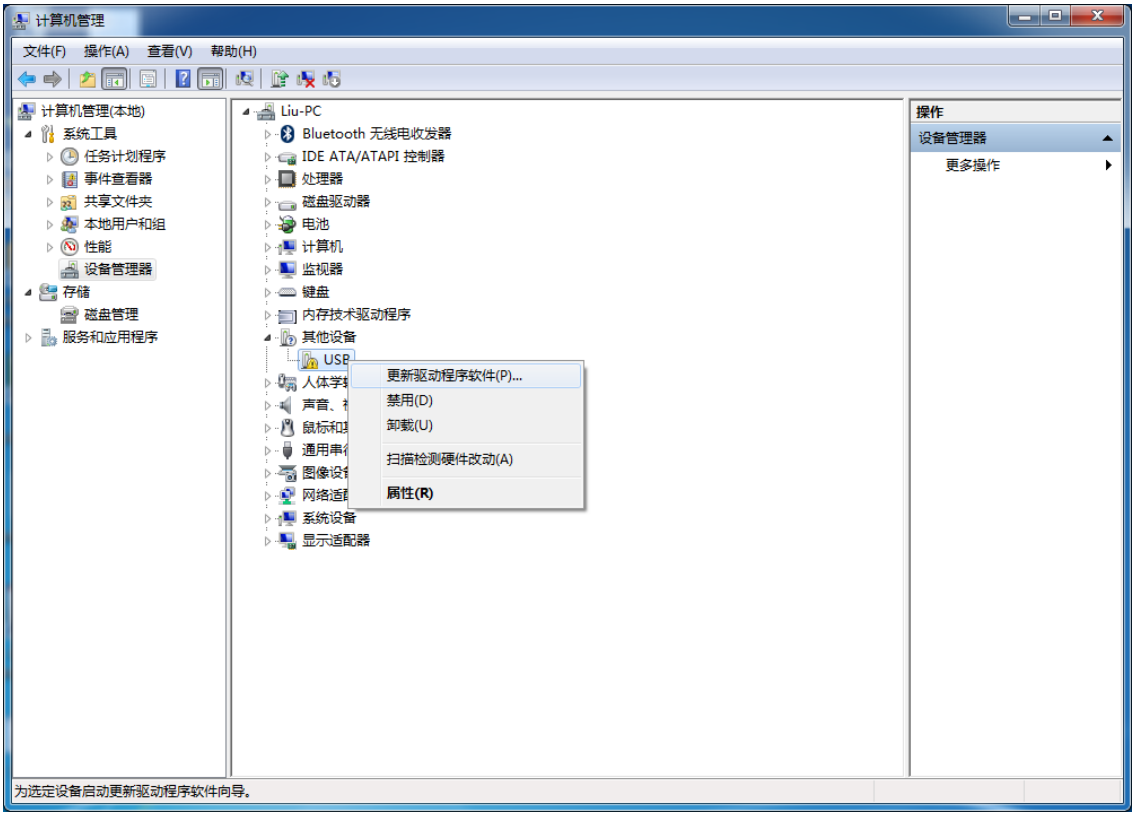
由于 Windows7 64 位操作系统在默认状态下，对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前，需要按照如下步骤，暂时跳过数字签名，即可顺利安装成功。

首先重启电脑，并一直按住 F8，直到出现下面启动画面

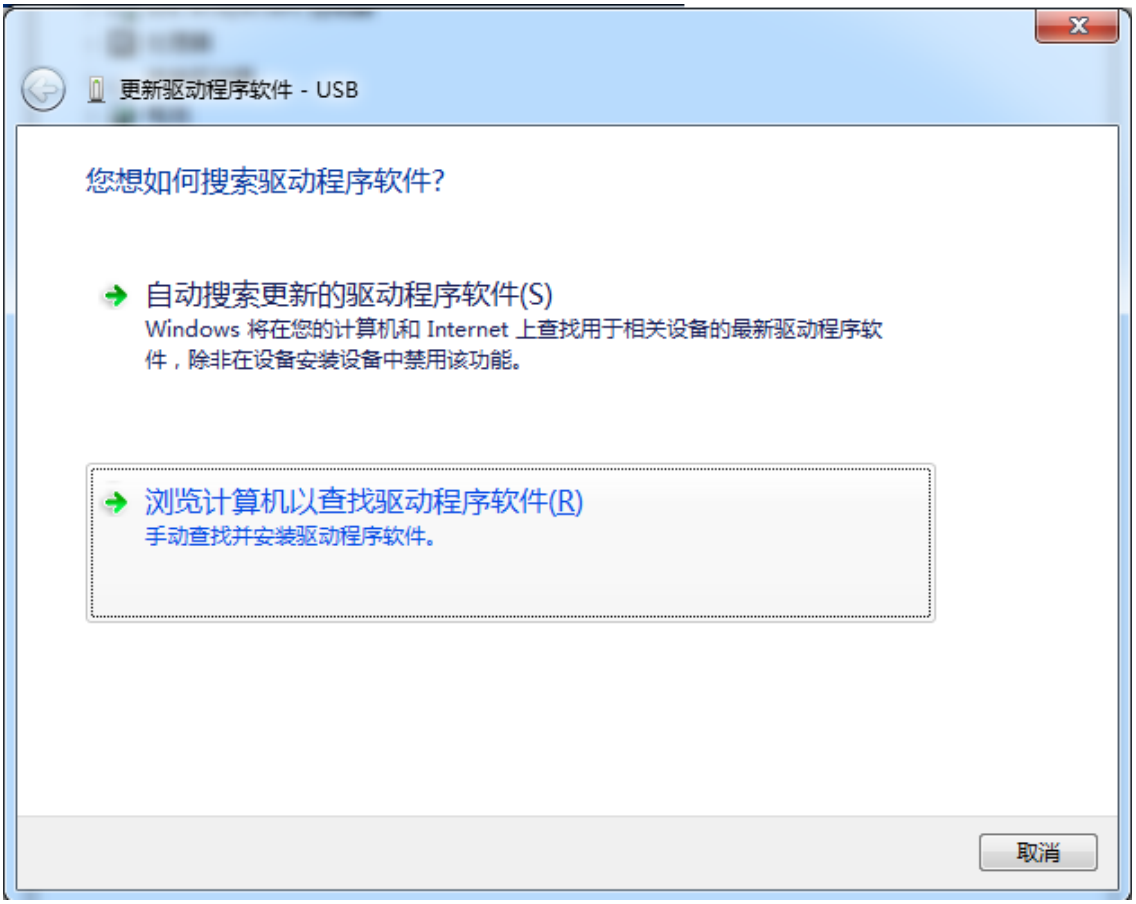


选择“禁用驱动程序签名强制”。启动后即可暂时关闭数字签名验证功能

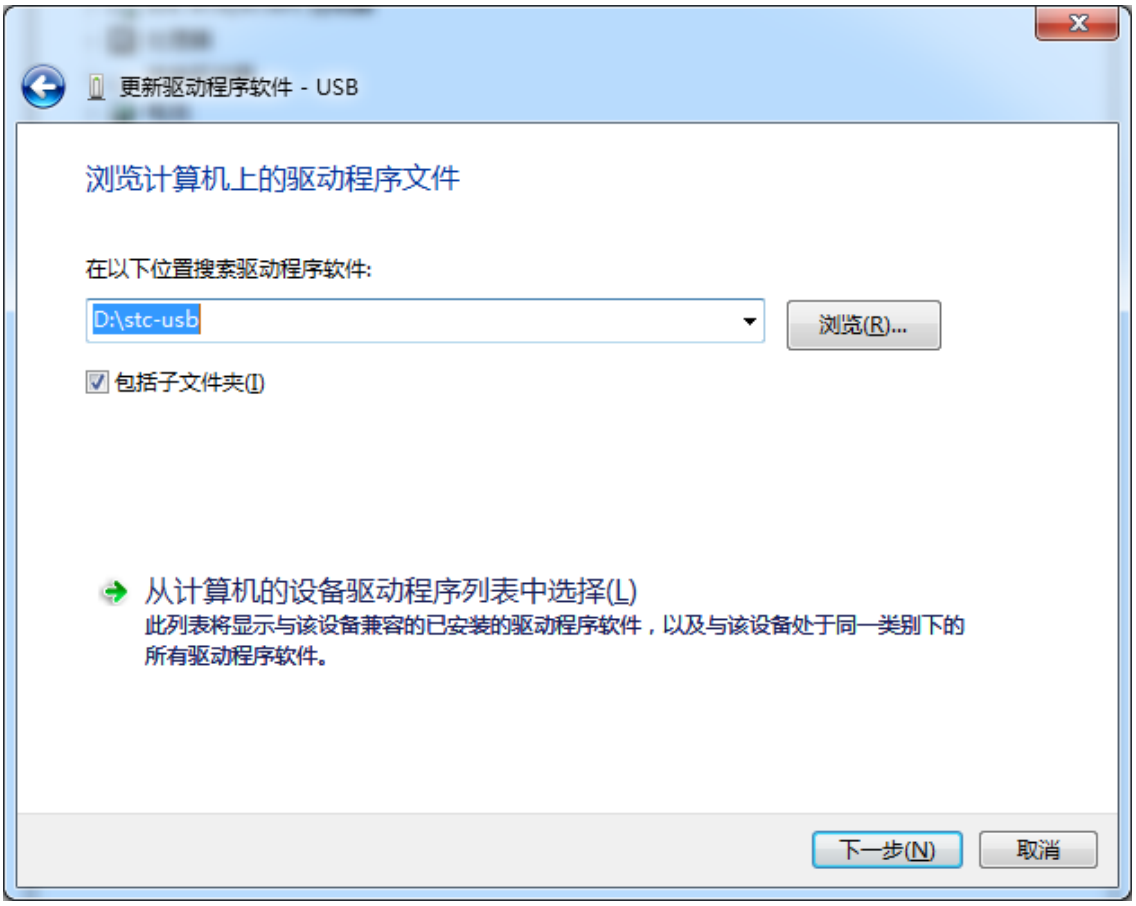
插入 USB 设备，并打开“设备管理器”。找到设备列表中带黄色感叹号的 USB 设备，在设备的右键菜单中，选择“更新驱动程序软件”



在下面的的对话框中选择“浏览计算机以查找驱动程序软件”

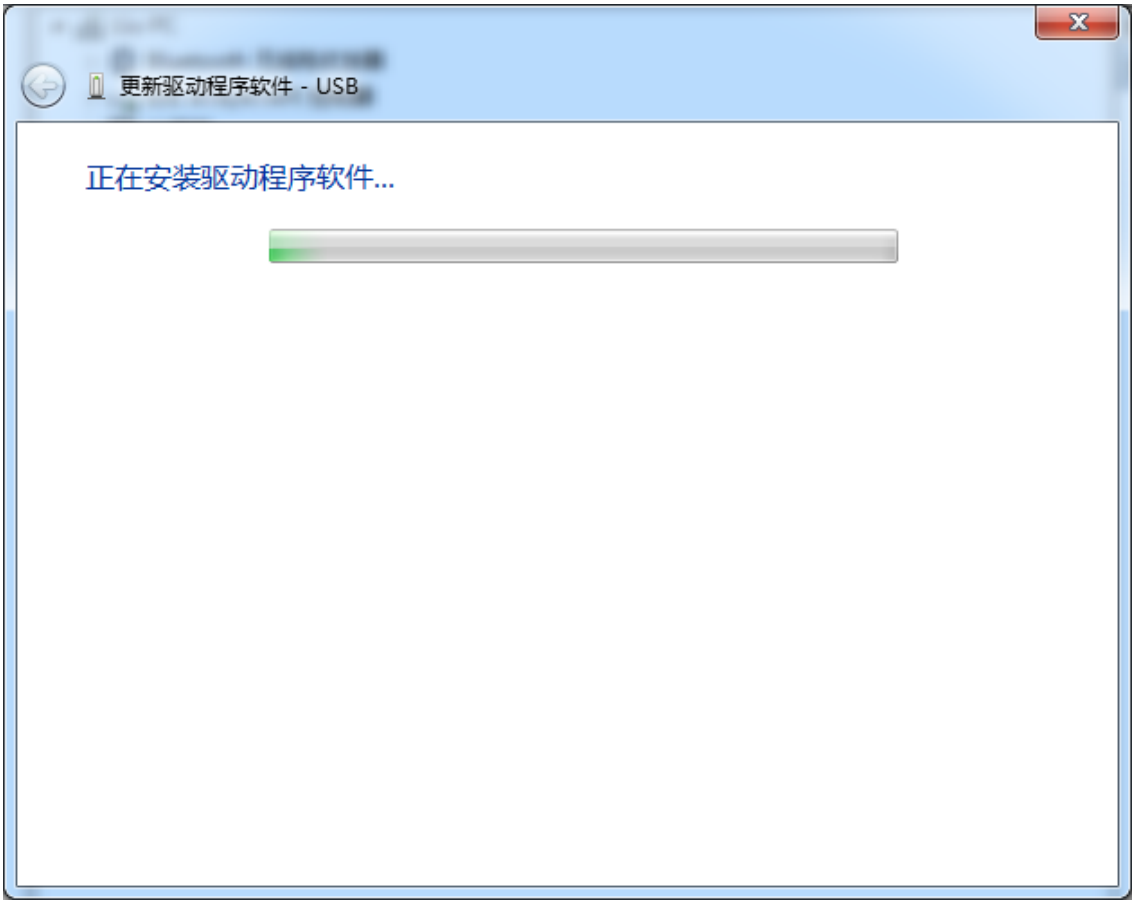


单击下面对话框中的“浏览”按钮，找到之前 STC-USB 驱动程序的存放目录（例如：之前的示例目录为 “ D:\STC-USB ”， 用 户 将 路 径 定 位 到 实 际 的 解 压 目 录 ）

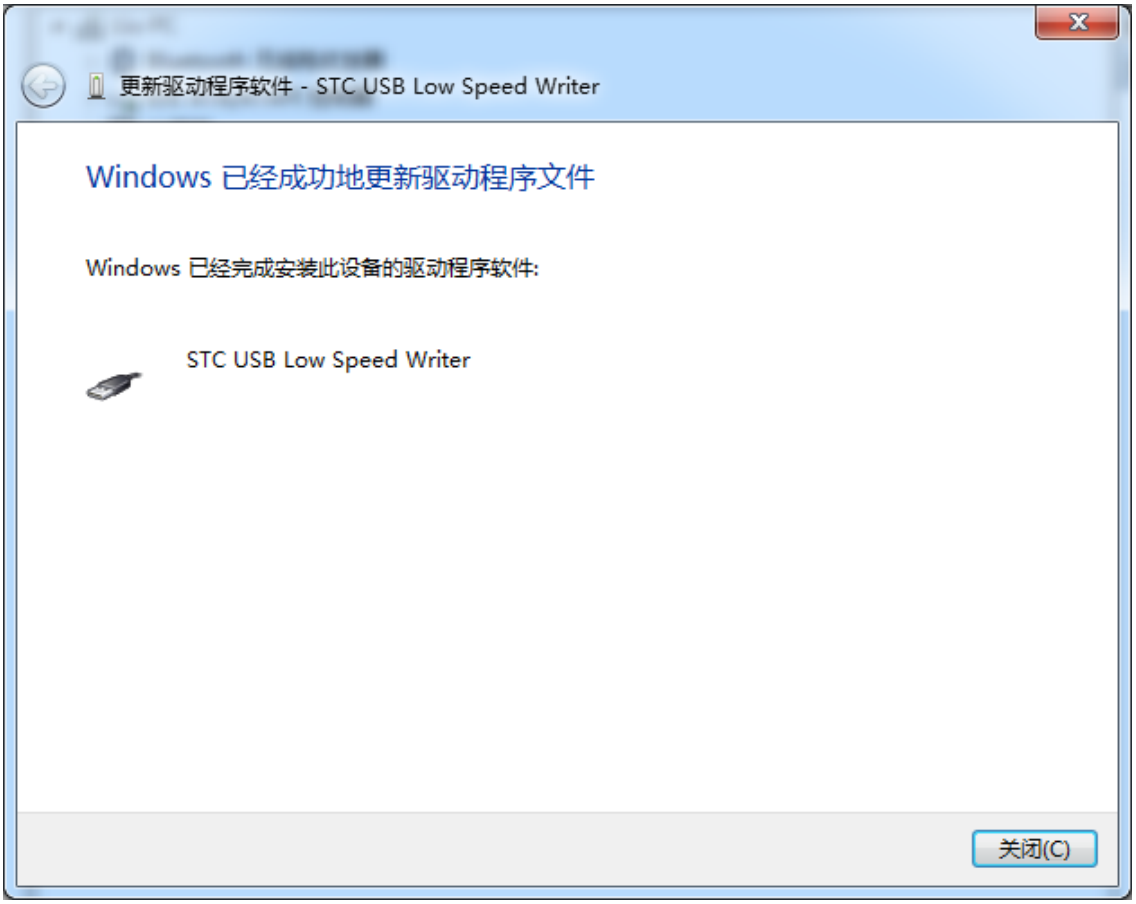


驱动程序开始安装时，会弹出如下对话框，选择“始终安装此驱动程序软件”

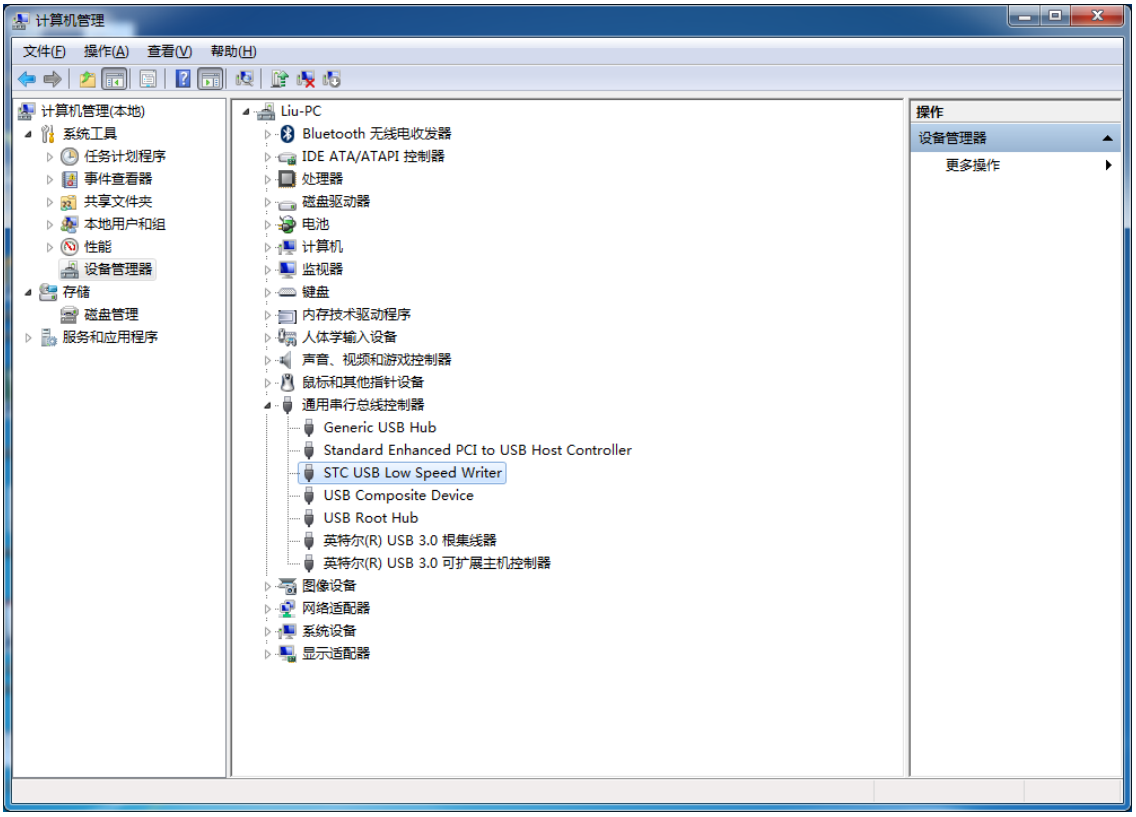
接下来，系统会自动安装驱动，如下图



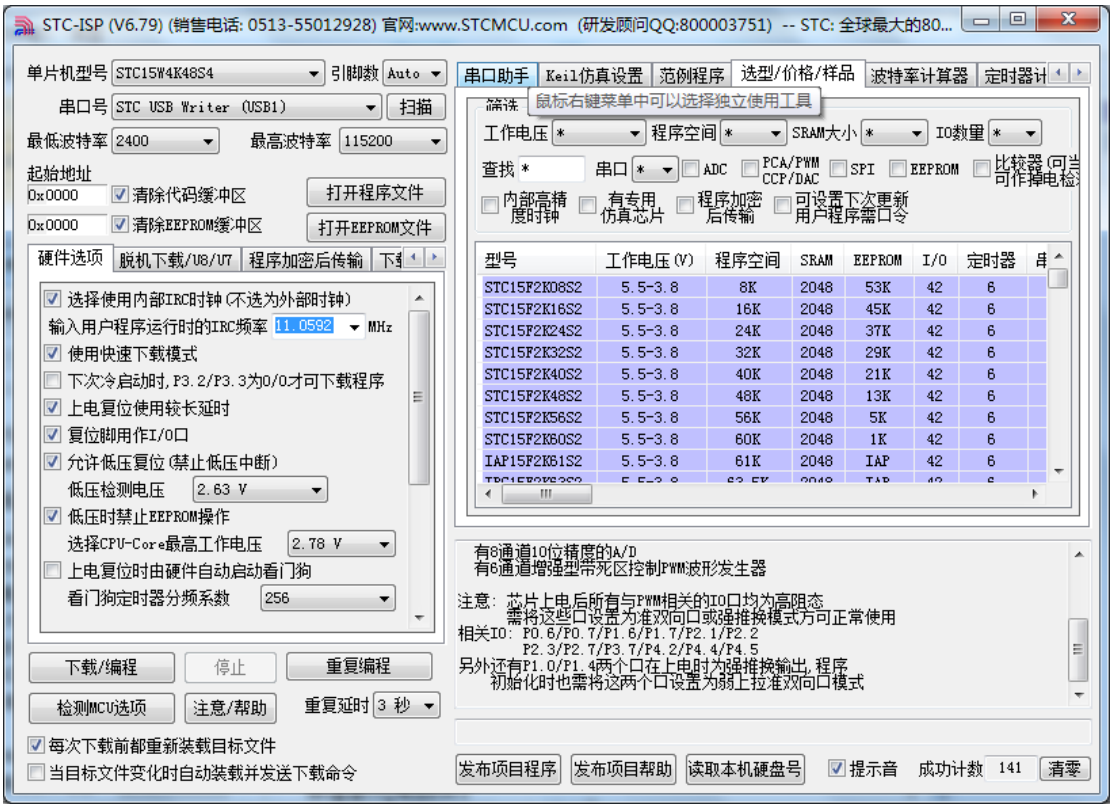
出现下面的的对话框表示驱动安装完成



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

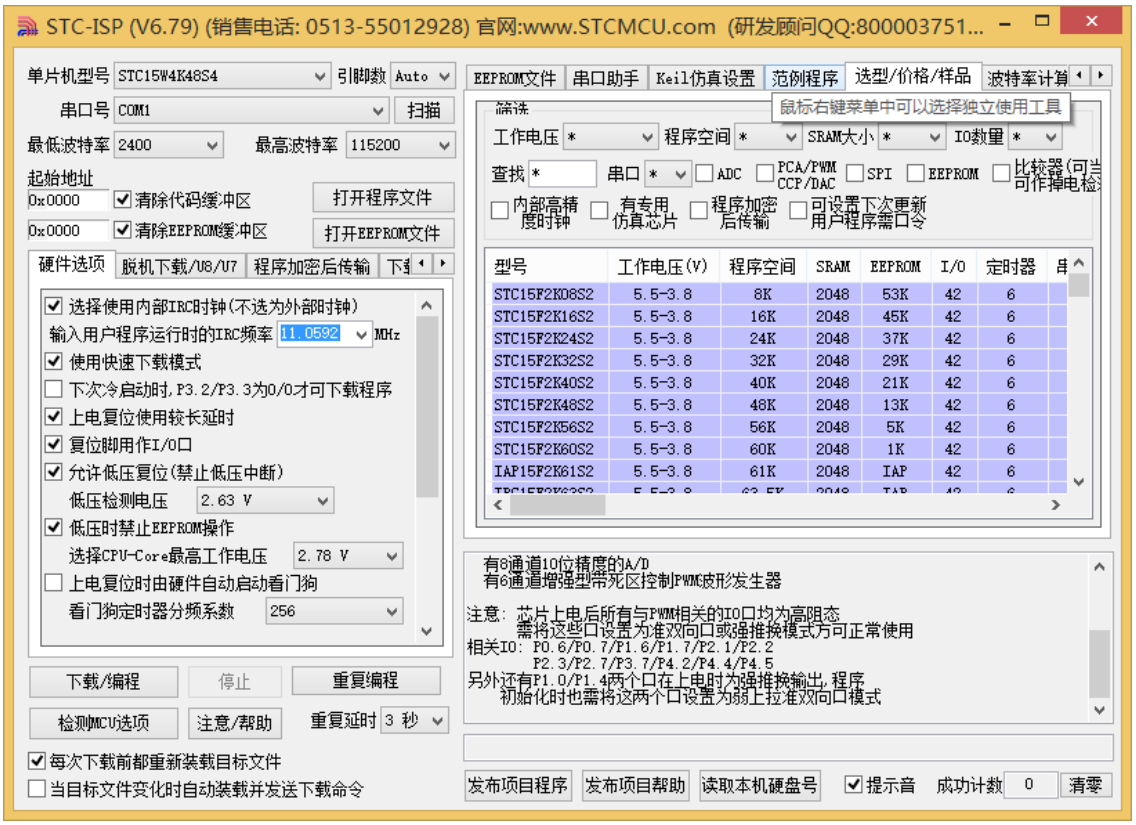


在之前打开的 STC-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：

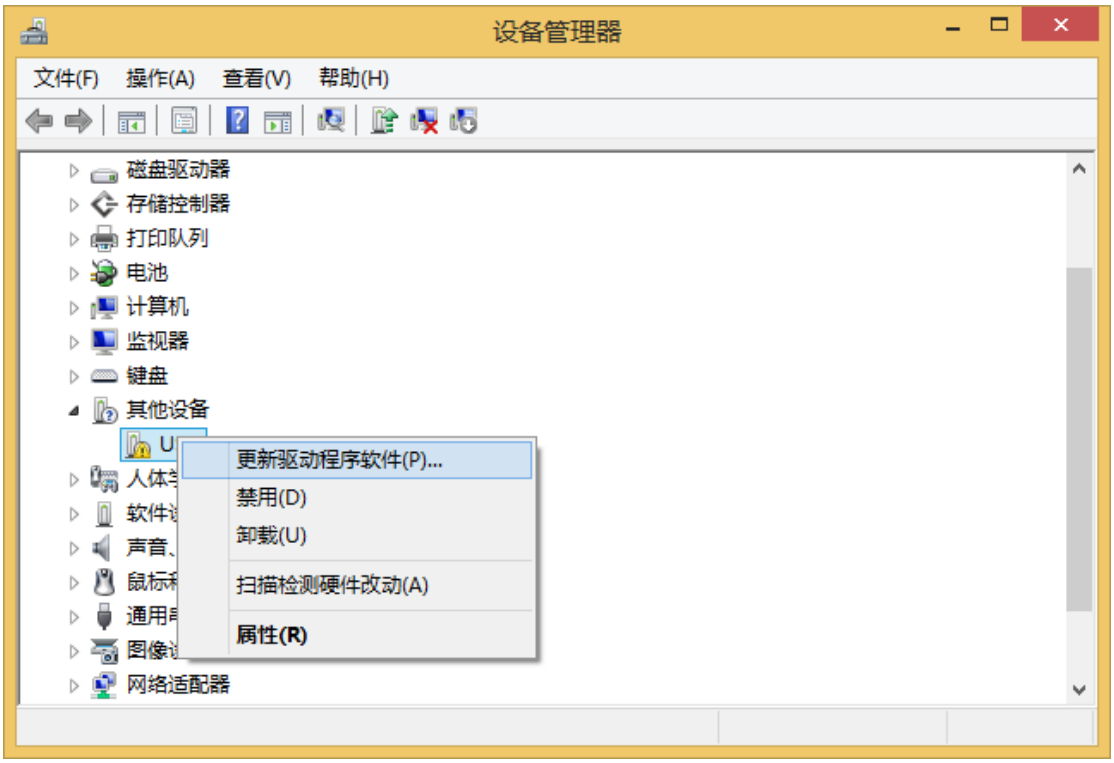


Windows 8（32 位）安装方法

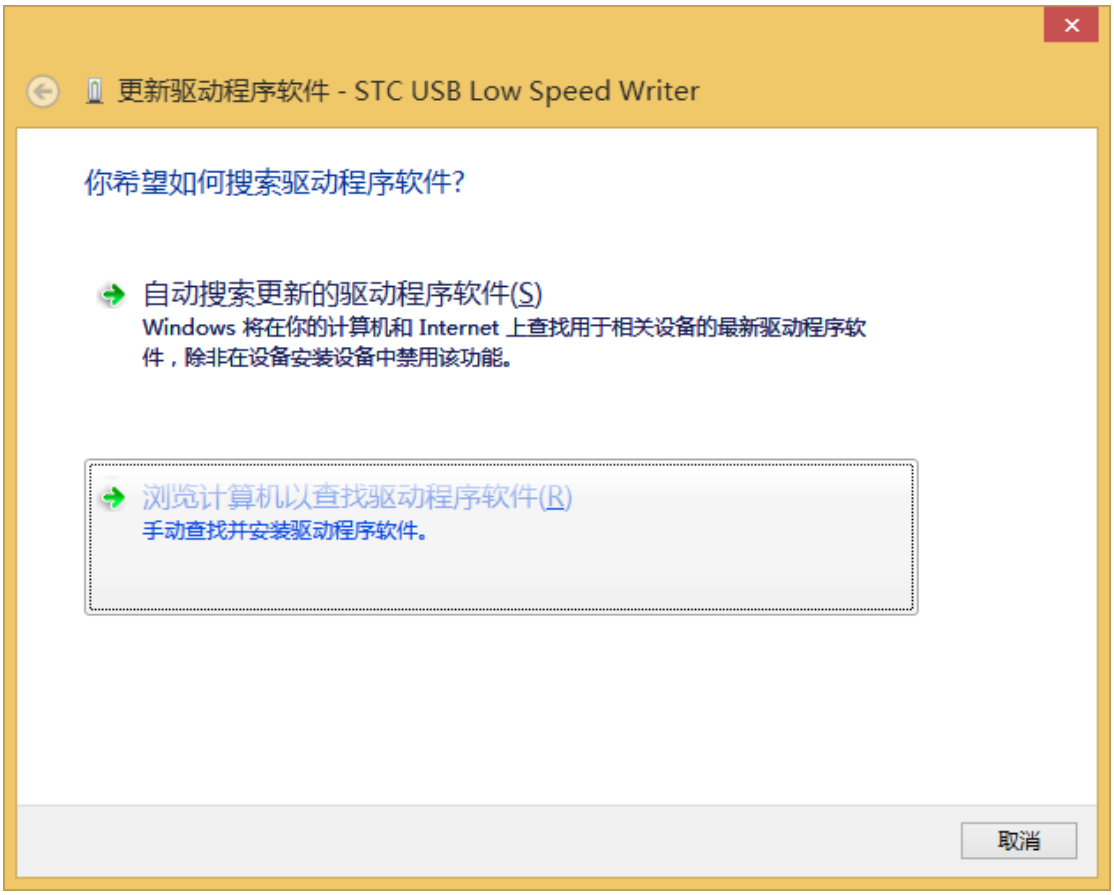
打开 V6.79 版（或者更新的版本）的 STC-ISP 下载软件（由于权限的原因，在 Windows 8 中下载软件不会将驱动文件复制到相关的系统目录，需要用户手动安装。首先从 STC 官方网站下载“stc-isp-15xx-v6.79.zip”（或更新版本），下载后解压到本地磁盘，则 STC-USB 的驱动文件也会被解压到当前解压目录中的“STC-USB Driver”中（例如将下载的压缩文件“stc-isp-15xx-v6.79.zip”解压到“F:\”，则 STC-USB 驱动程序在“F:\STC-USB Driver”目录中））



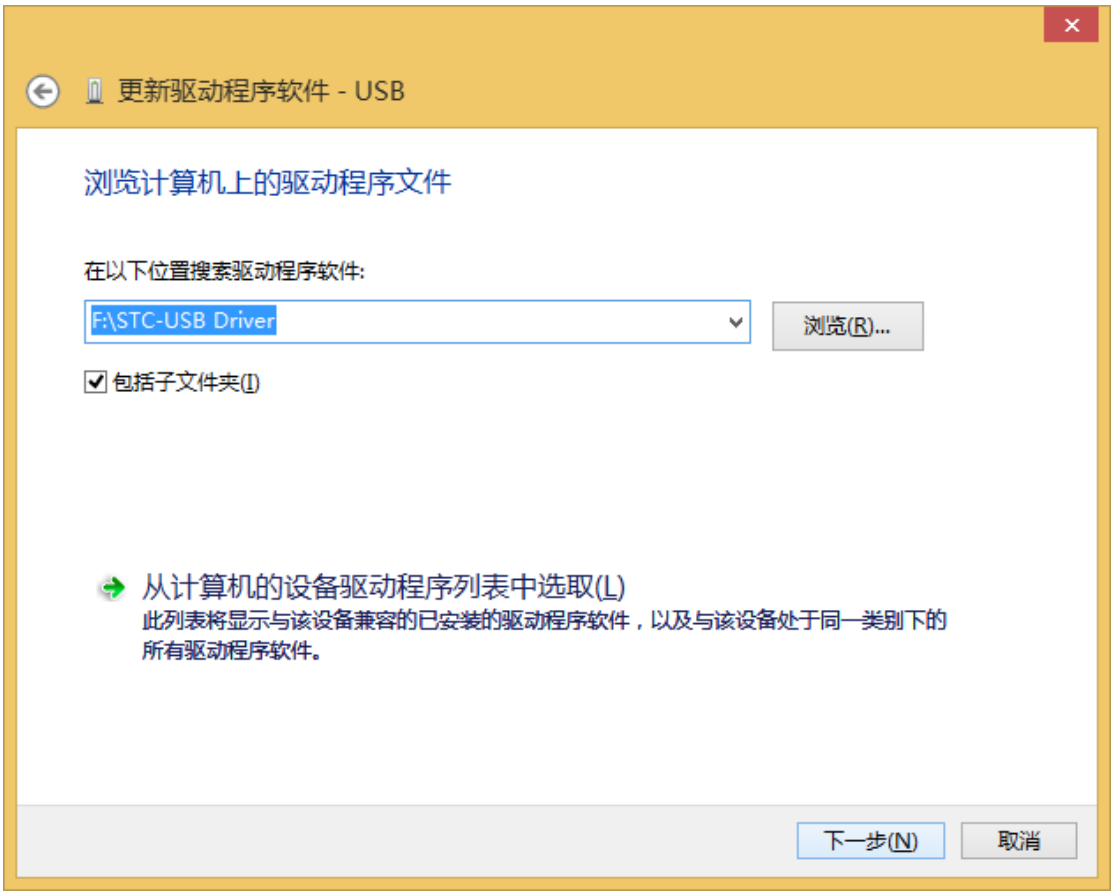
插入 USB 设备，并打开“设备管理器”。找到设备列表中带黄色感叹号的 USB 设备，在设备的右键菜单中，选择“更新驱动程序软件”



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”

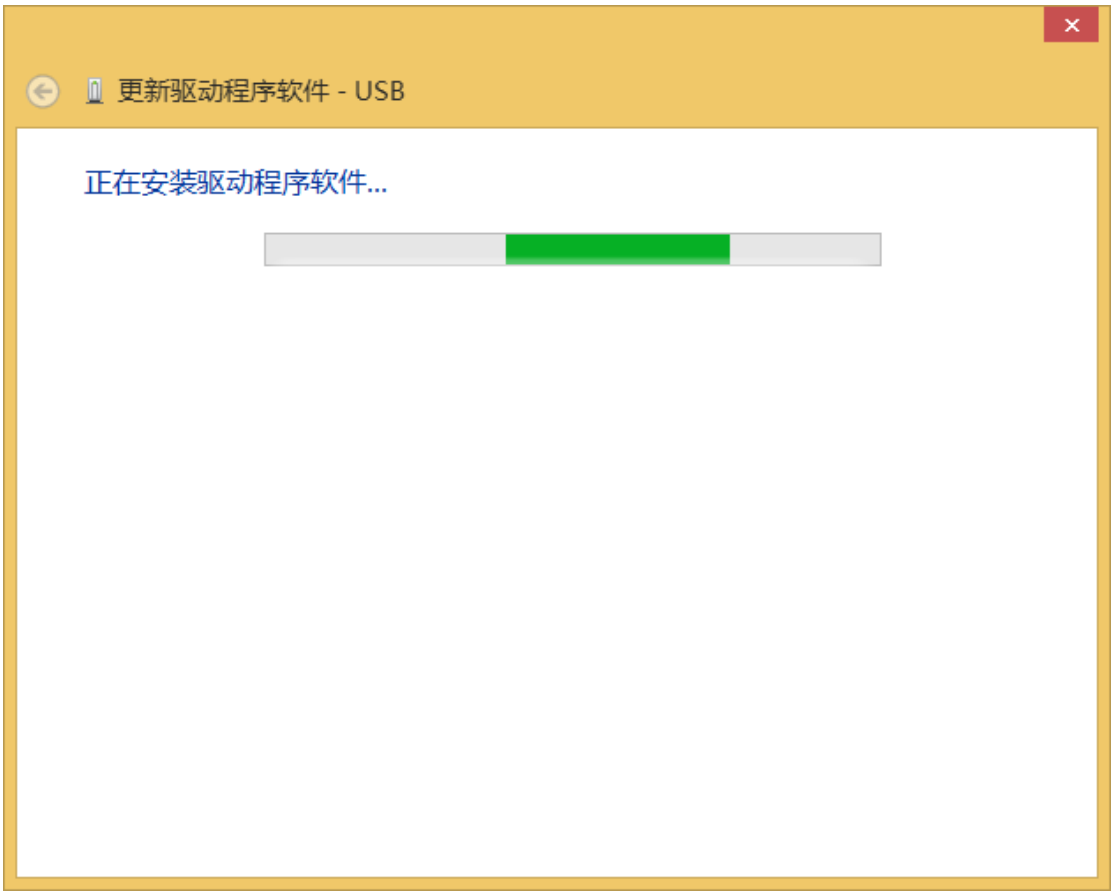


单击下面对话框中的“浏览”按钮，找到之前 STC-USB 驱动程序的存放目录（例如：之前的示例目录为“F:\STC-USB Driver”，用户将路径定位到实际的解压目录）

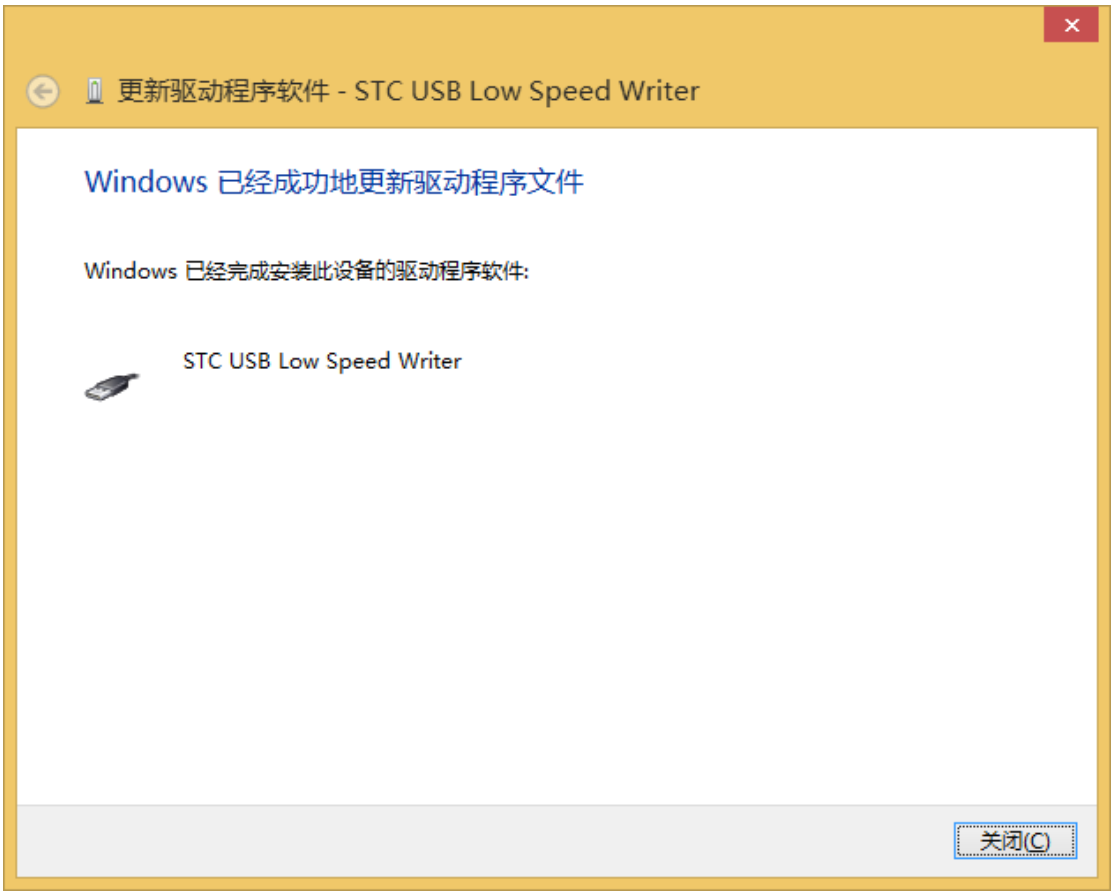


驱动程序开始安装时，会弹出如下对话框，选择“始终安装此驱动程序软件”

接下来，系统会自动安装驱动，如下图

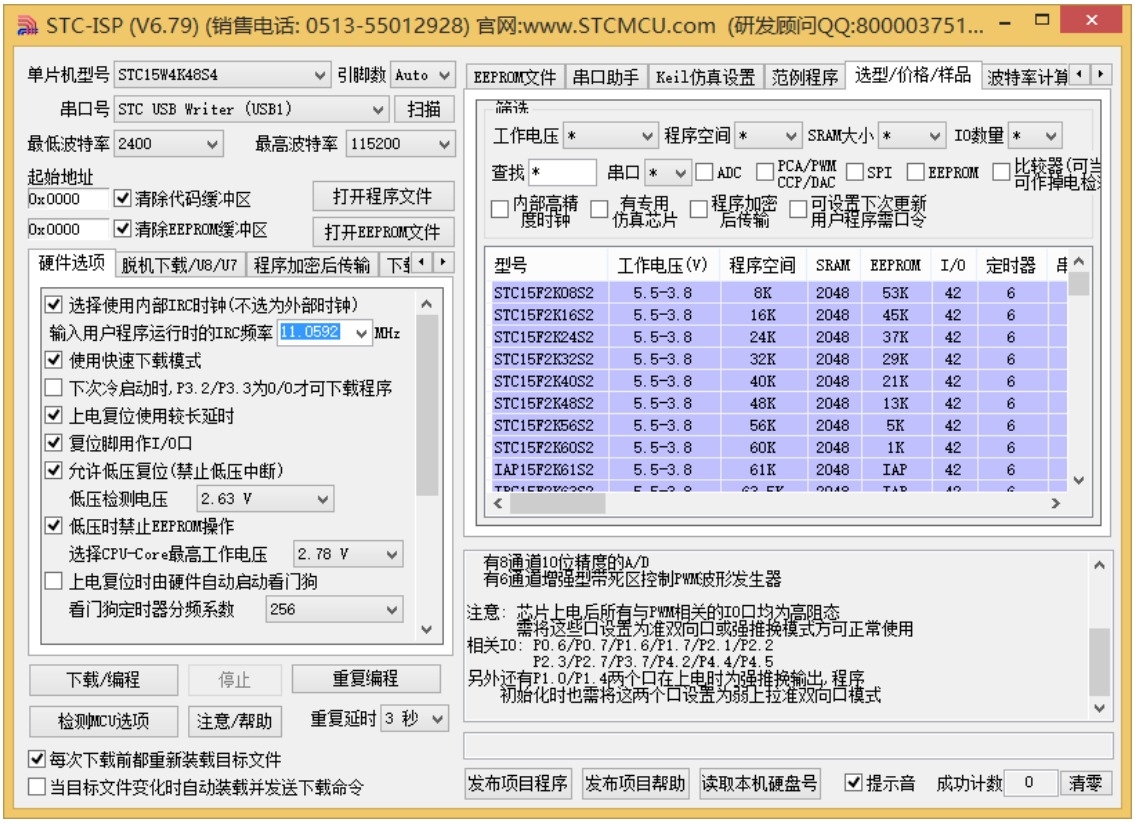


出现下面的的对话框表示驱动安装完成



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：

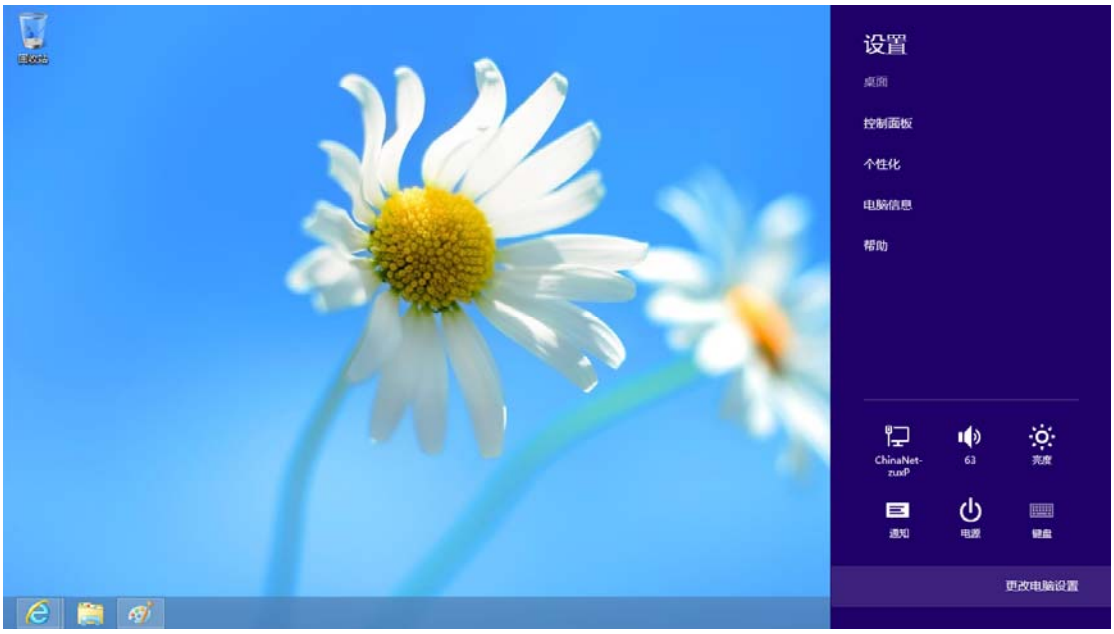


Windows 8（64 位）安装方法

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



在电脑设置中，选择“常规”属性页中“高级启动”项下面的“立即启动”按钮

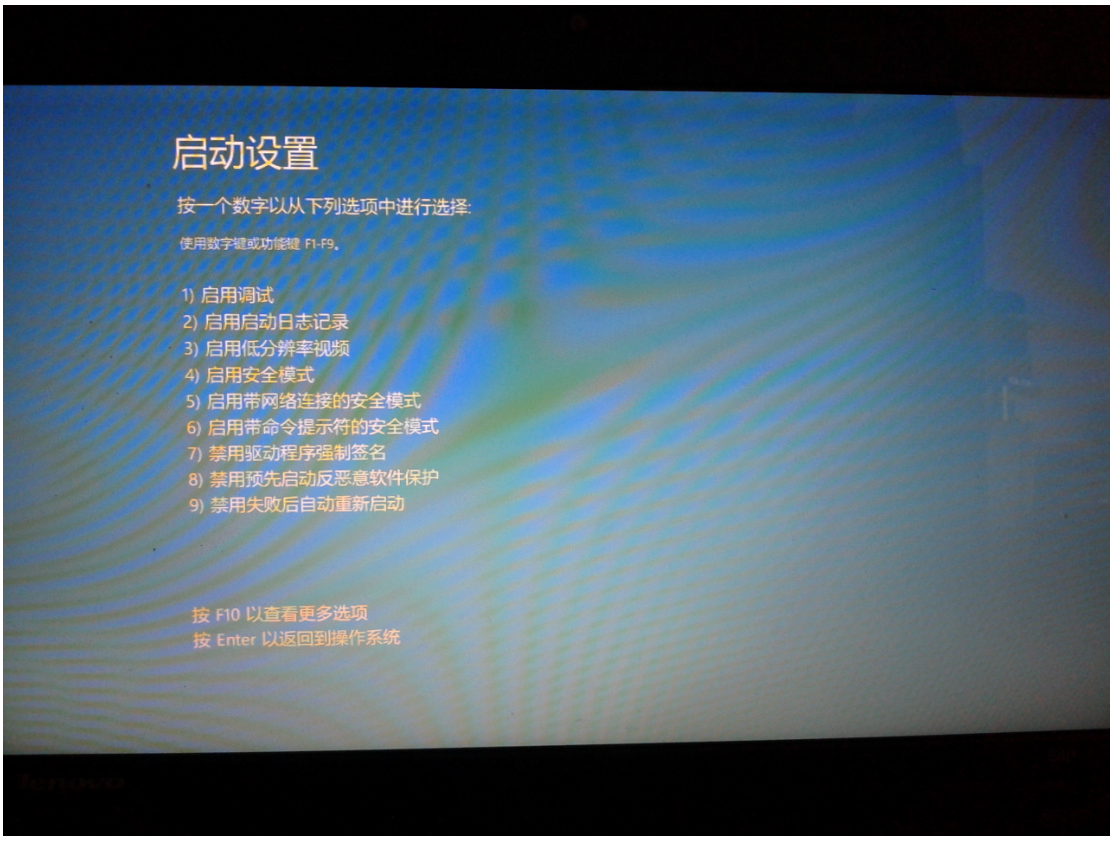
在下面的界面中，选择“疑难解答”项

然后选择“疑难解答”中的“高级选项”

在下面的“高级选项”界面中，选择“启动设置”

在下面的“启动设置”界面中，单击“重启”按钮对电脑进行重新启动

在电脑重新启动后会自动进入如下图所示的“启动设置”界面，按数字键“7”或者按功能键“F7”选择“禁用驱动程序强制签名”进行启动

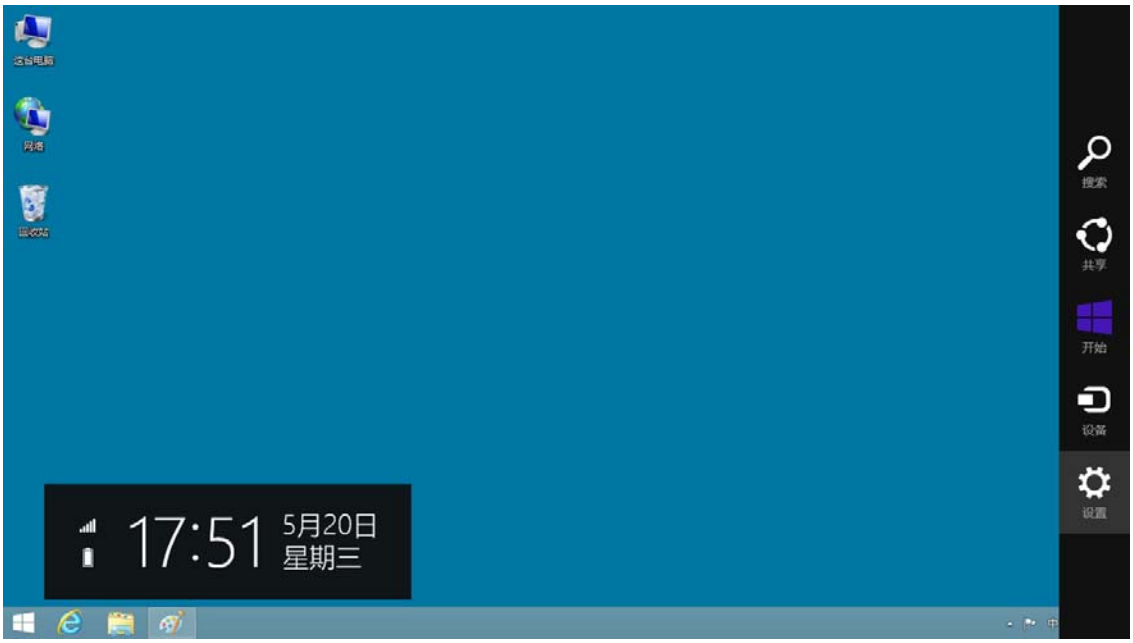


启动到 Windows 8 后，按照 **Windows 8（32 位）的安装方法**即可完成驱动的安装

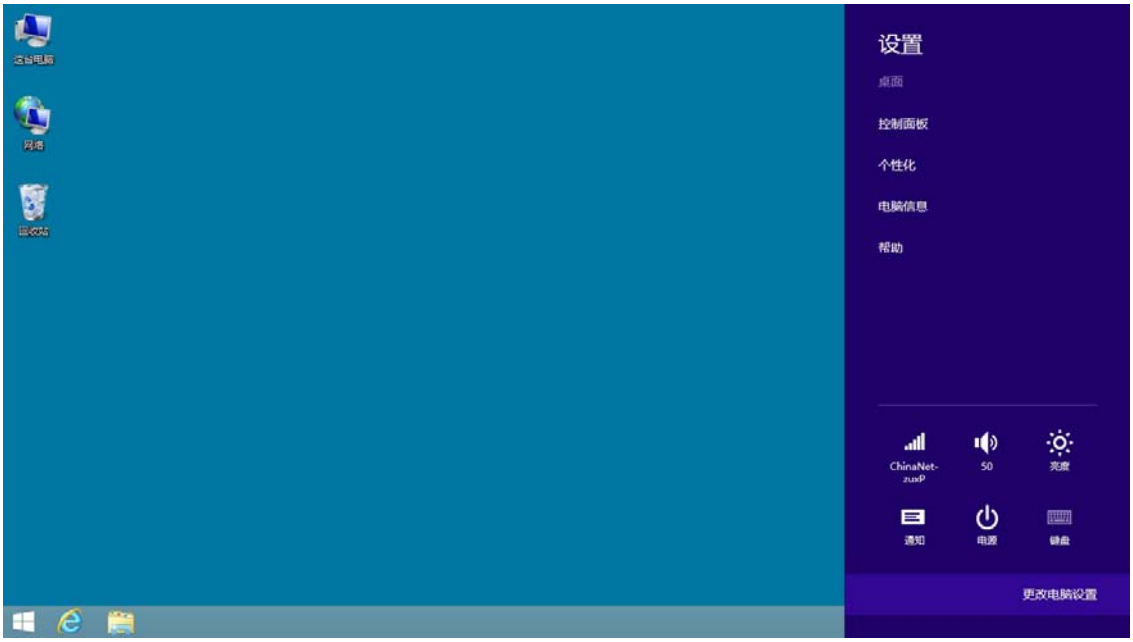
Windows 8.1（64 位）安装方法

Windows 8.1 与 Windows 8 进入高级启动菜单的方法不一样,在此专门进行说明。

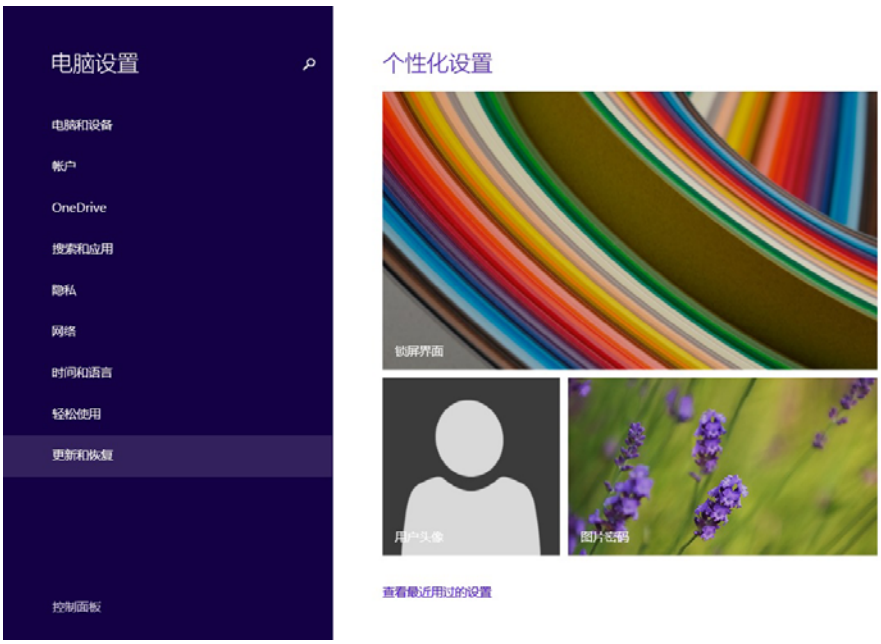
首先将鼠标移动到屏幕的右下角，选择其中的“设置”按钮



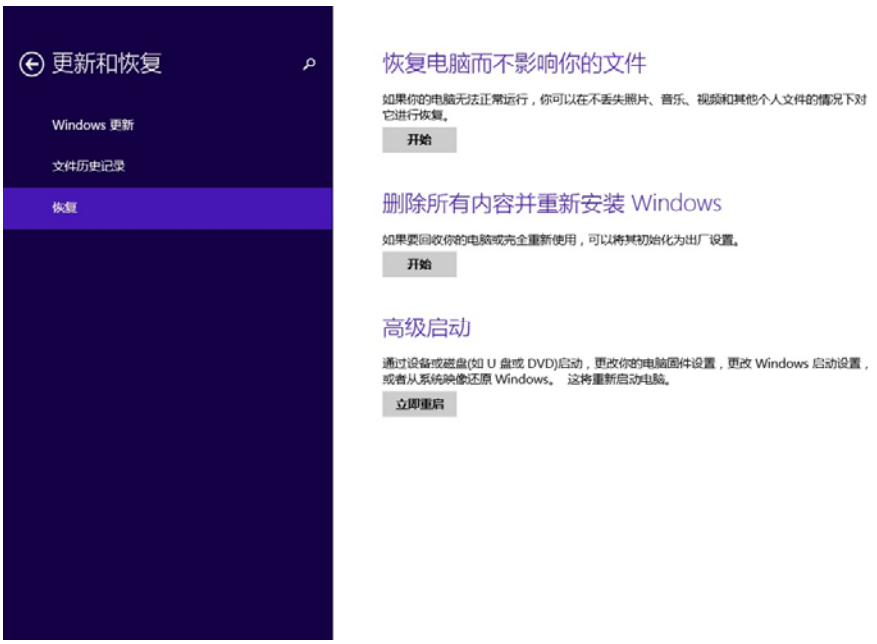
然后在设置界面中选择“更改电脑设置”项



在电脑设置中，选择“更新和恢复”（这里与 Windows 8 不一样，Windows 8 选择的是“常规”）



在更新和恢复页面中选择“恢复”属性页，单击“高级启动”项下面的“立即启动”按钮



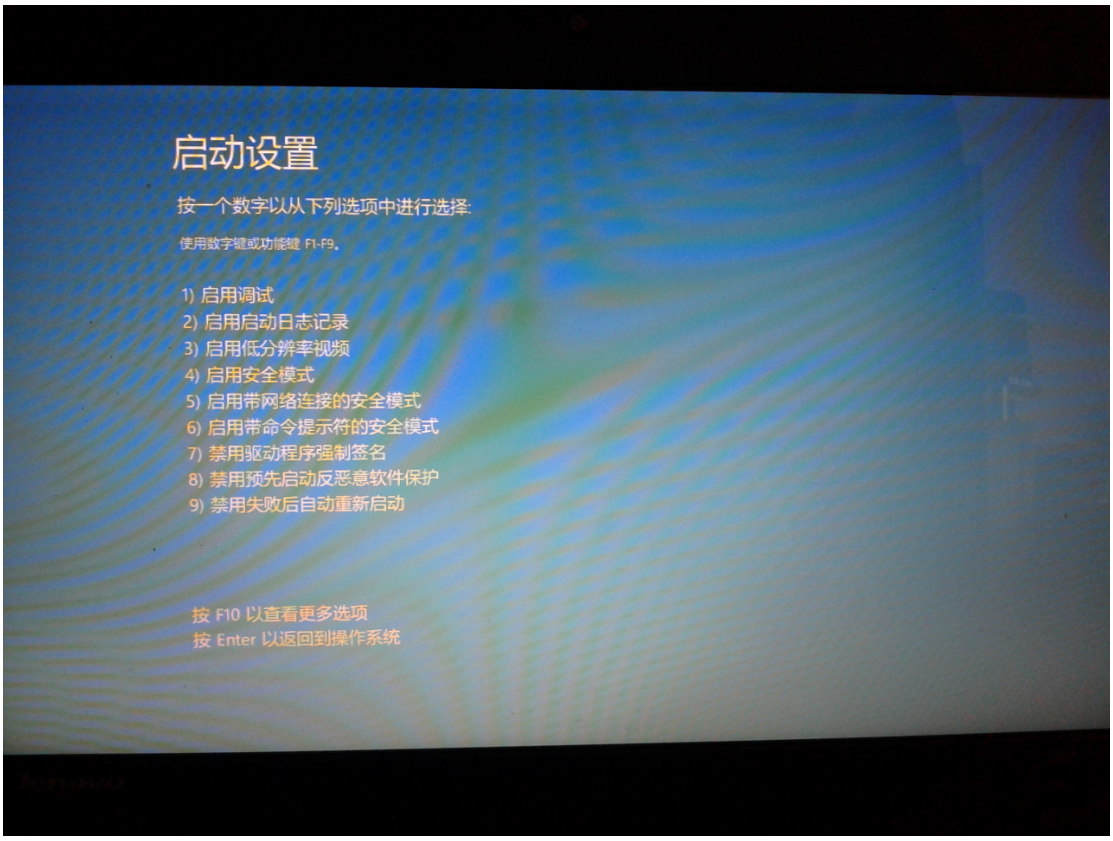
接下来的操作与 Window 8 的步骤相同
在下面的界面中，选择“疑难解答”项

然后选择“疑难解答”中的“高级选项”

在下面的“高级选项”界面中，选择“启动设置”

在下面的“启动设置”界面中，单击“重启”按钮对电脑进行重新启动

在电脑重新启动后会自动进入如下图所示的“启动设置”界面，按数字键“7”或者按功能键“F7”选择“禁用驱动程序强制签名”进行启动



启动到 Windows 8 后，按照 **Windows 8（32 位）的安装方法**即可完成驱动的安装

Windows10（64 位）安装方法

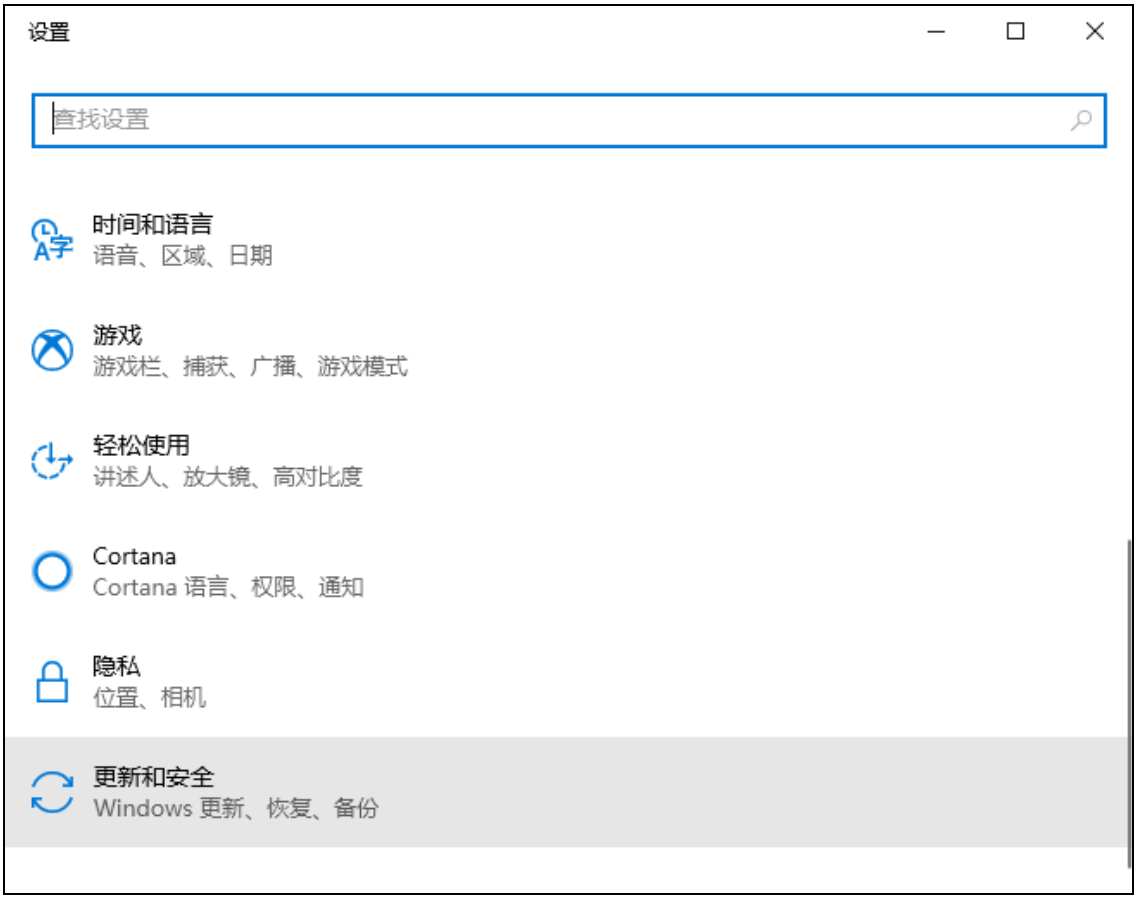
由于 Windows10 64 位操作系统在默认状态下，对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前，需要按照如下步骤，暂时跳过数字签名，即可顺利安装成功。

安装驱动前需要从 STC 官网下载的 STC-ISP 下载软件压缩包中将“STC-USB Driver”文件夹解压缩到硬盘中。将具有 USB 下载功能的芯片准备好，但先不要连接电脑

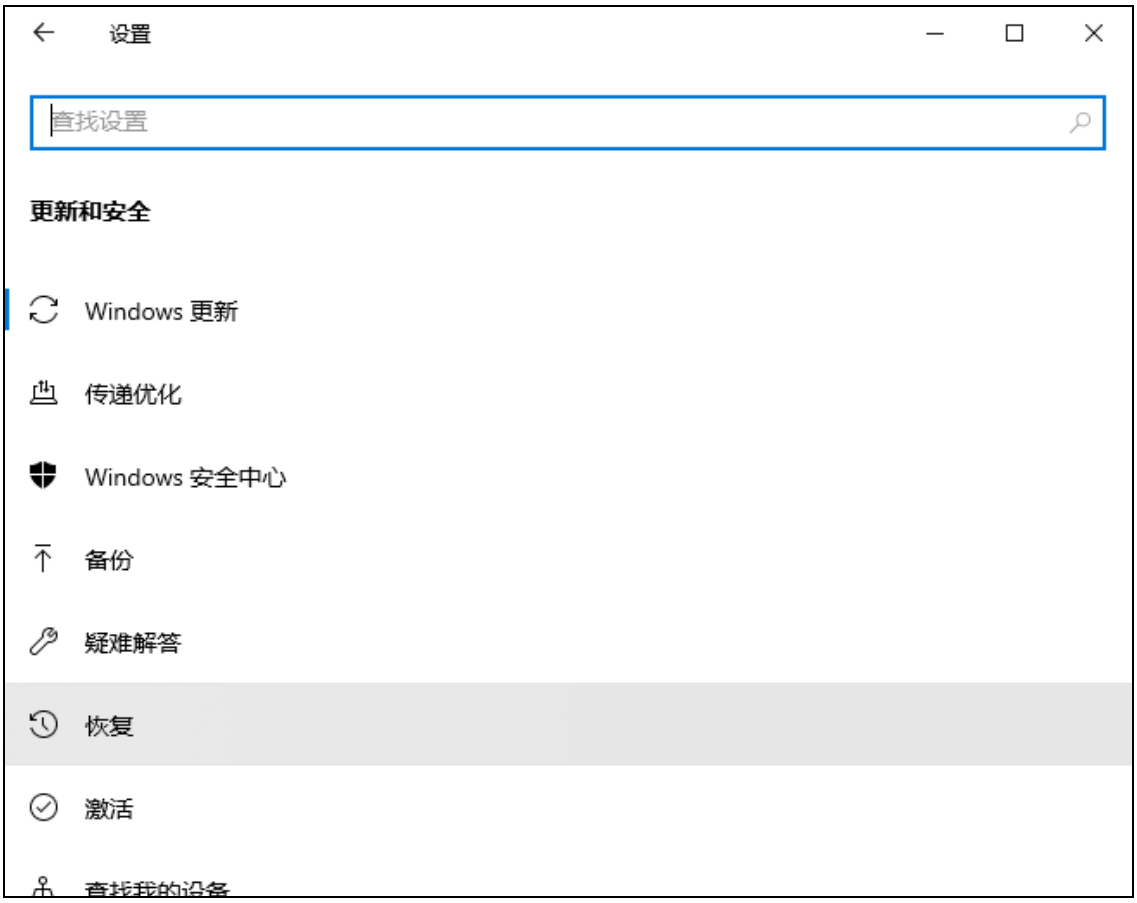
鼠标右键点击“开始”菜单，选择“设置”选项


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

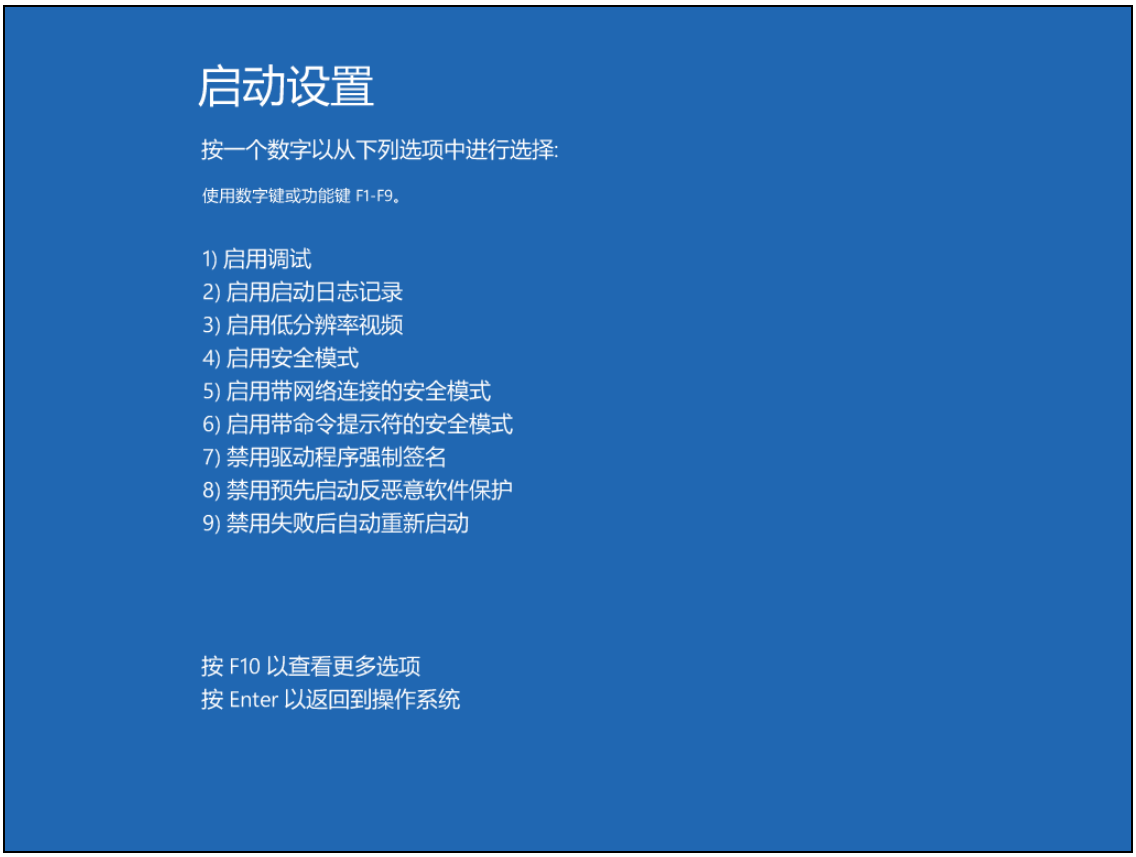
在疑难解答界面中选择“高级选项”

然后选择“查看更多恢复选项”

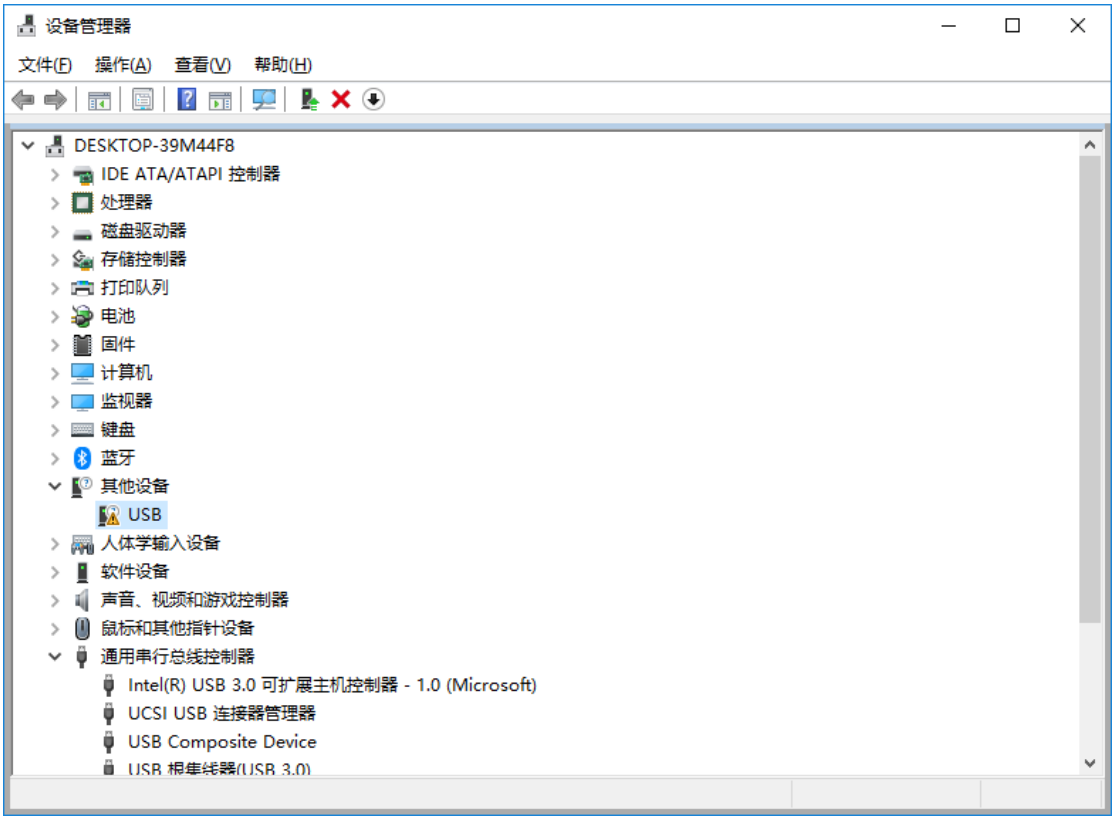
选择“启动设置”项

出现如下画面后，点击“重启”按钮重启电脑

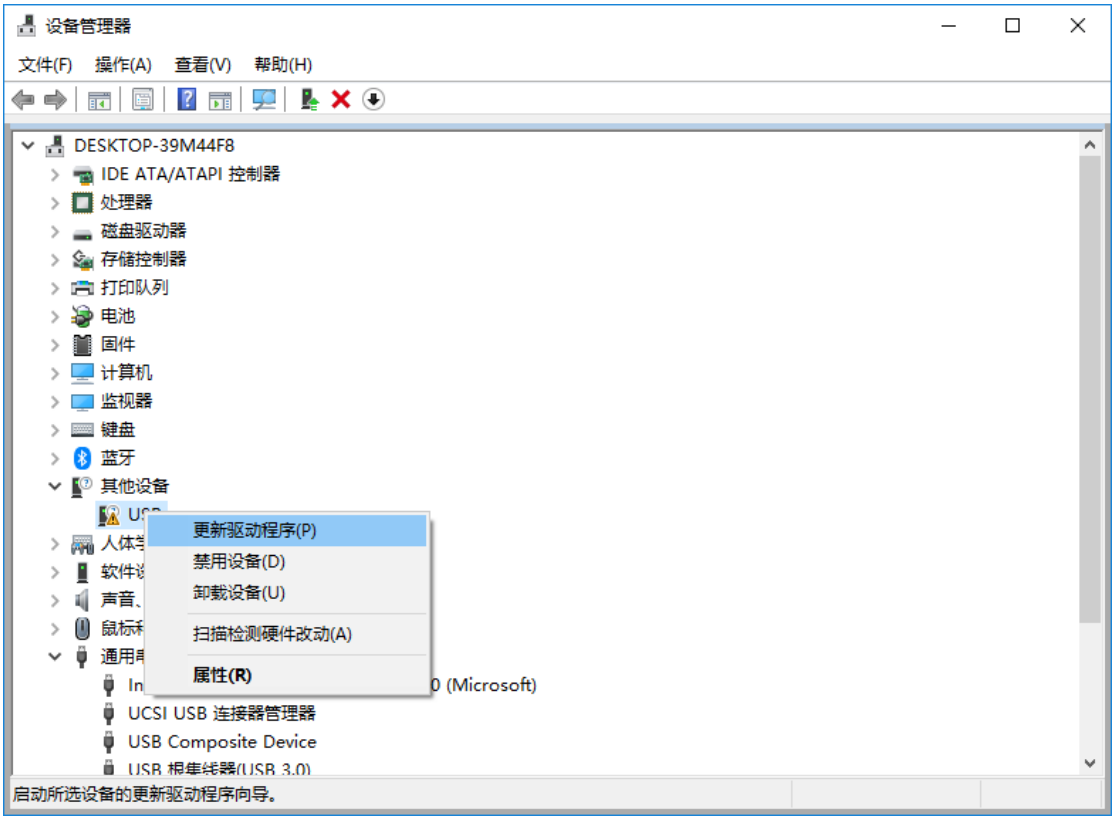
电脑重启后，会弹出“启动设置”界面，按“F7”按钮来选择“禁止驱动程序强制签名”项



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

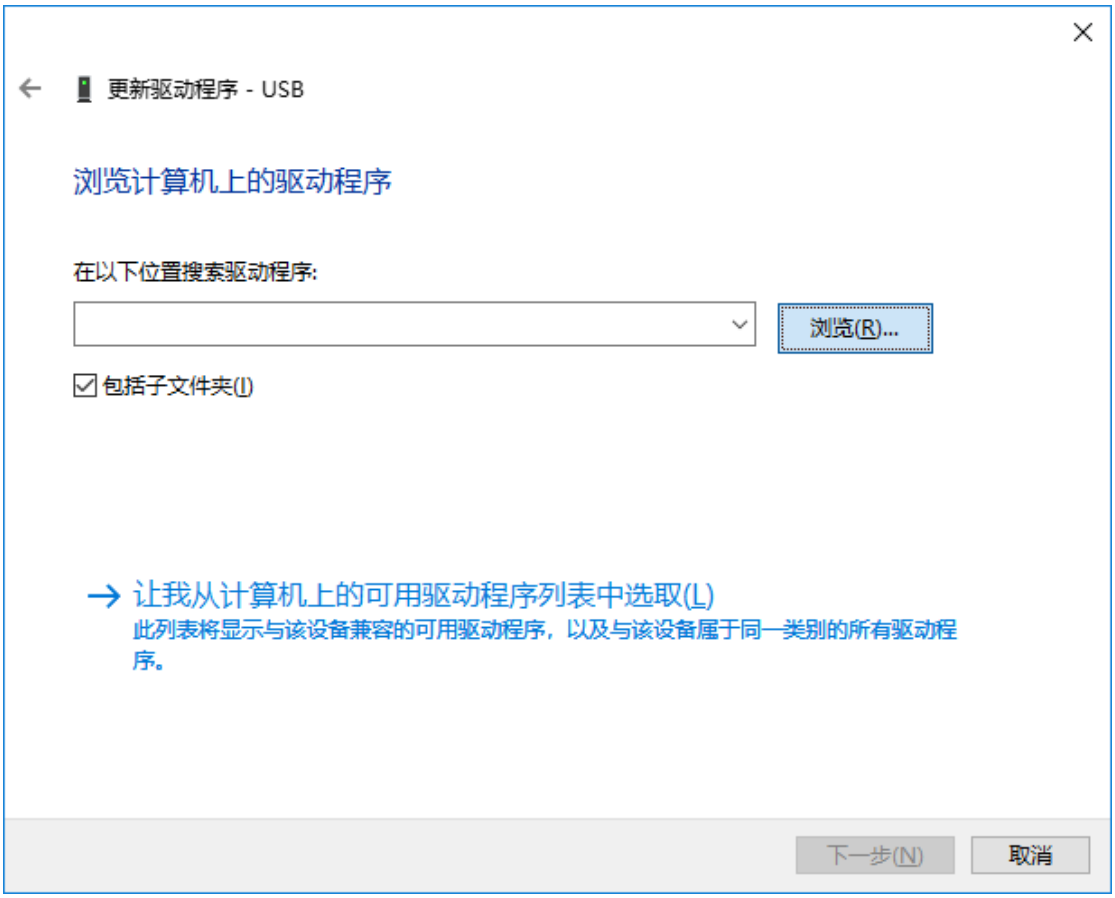


鼠标右键单击未知设备，选择右键菜单中的“更新驱动程序”

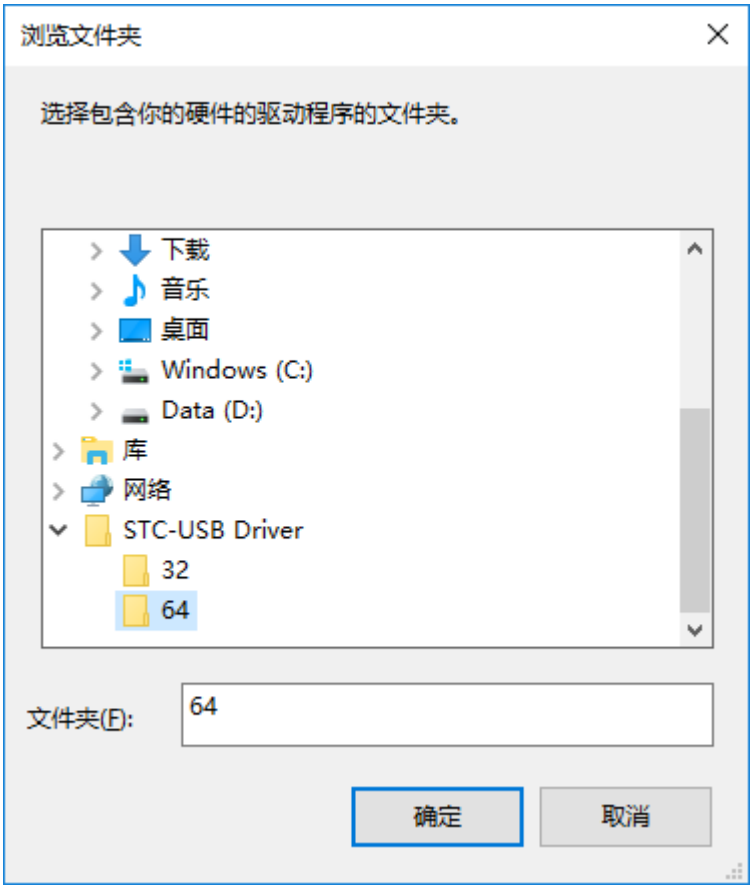


在弹出的驱动安装程序选择画面中，选择“浏览我的计算机以查找驱动程序软件”项

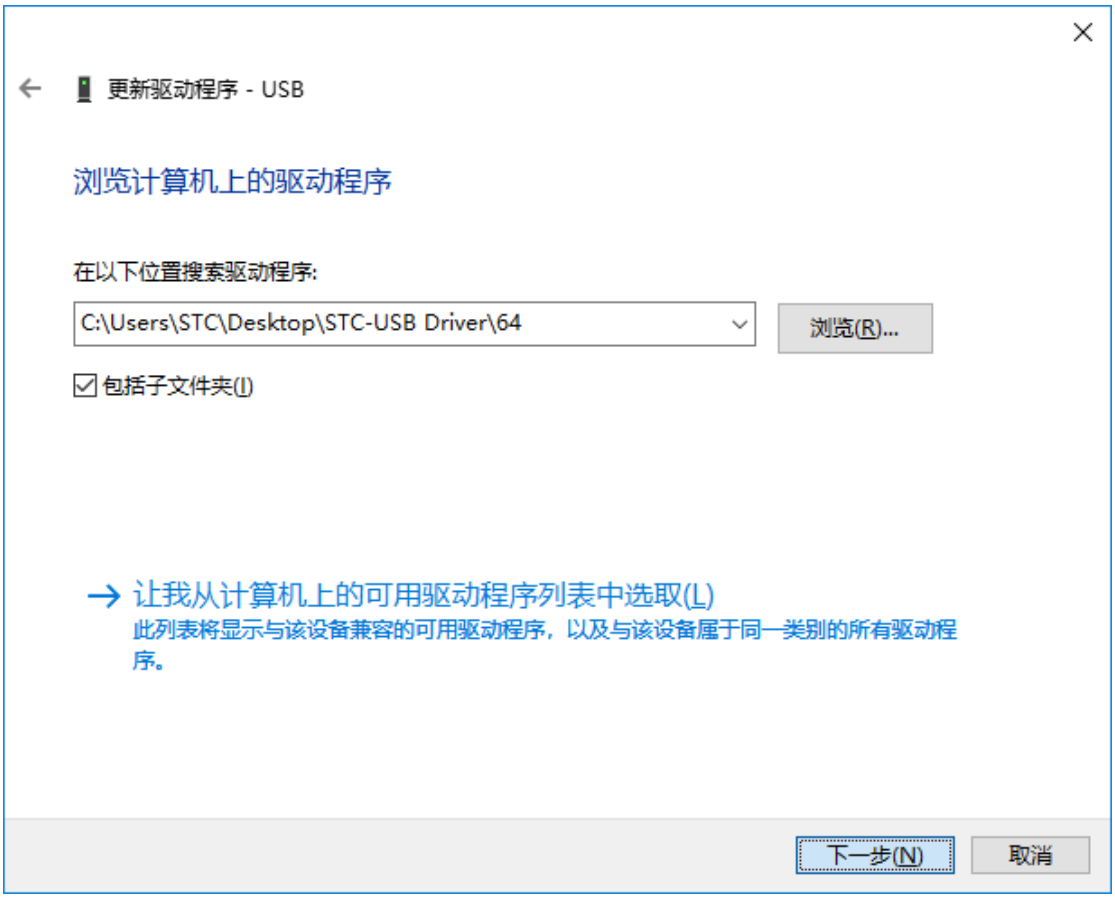
在如下界面中，点击“浏览”按钮



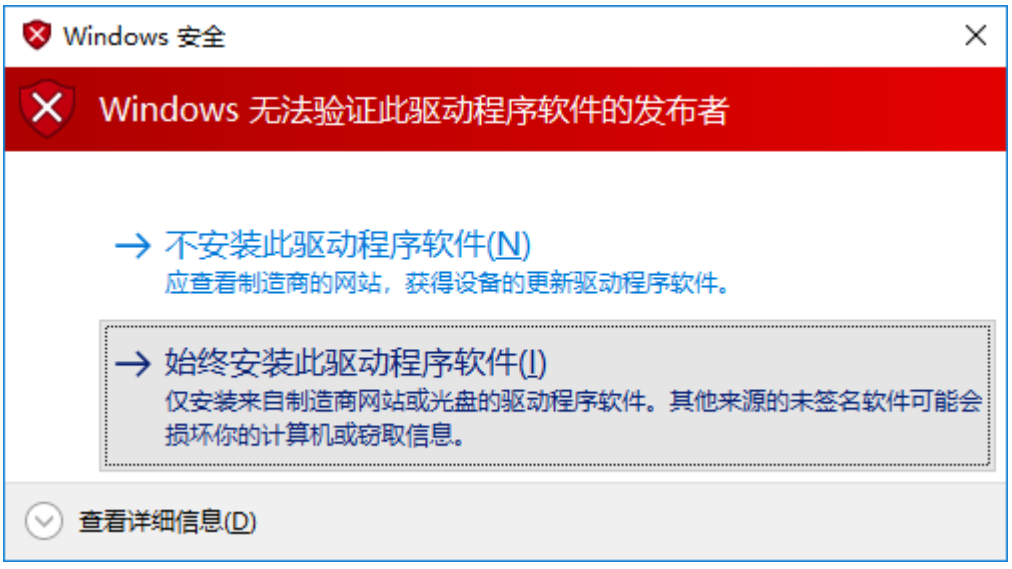
找到之前解压缩到硬盘中的“STC-USB Driver”目录，选择目录中的“64”目录，并确定



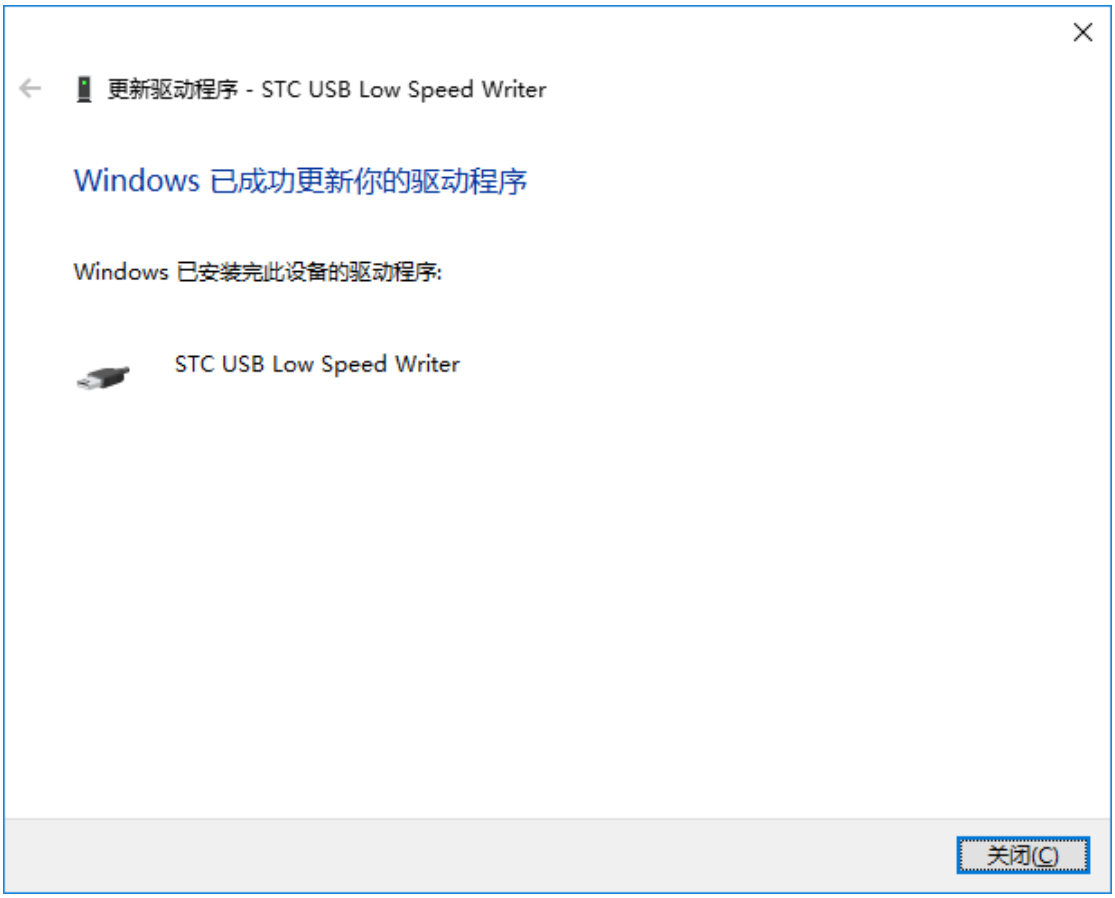
点击“下一步”开始安装驱动



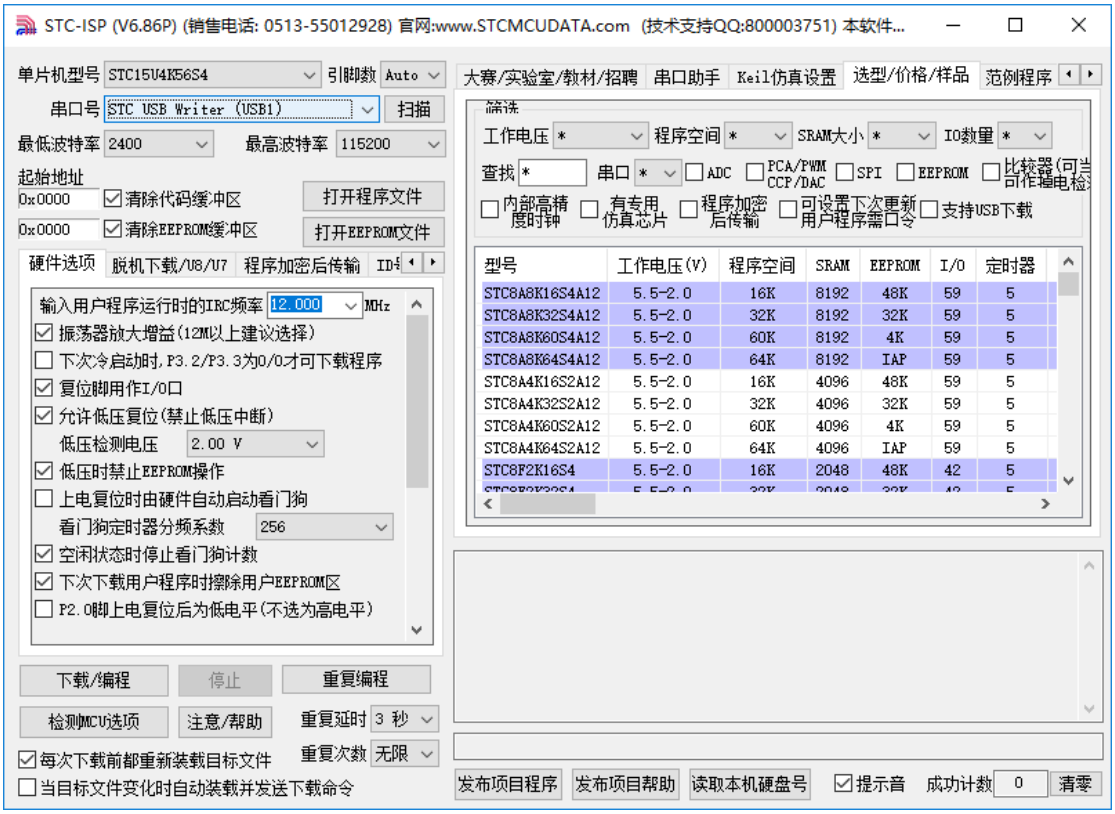
驱动安装的过程中，会弹出如下的警告画面，选择“始终安装此驱动程序软件”



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 电气特性

绝对最大额定值

参数	最小值	最大值	单位
存储温度	-55	+125	℃
工作温度	-40	+85	℃
工作电压	1.7	5.5	V
VDD 对地电压	-0.3	+5.5	V
IO 口对地电压	-0.3	VDD+0.3	V

直流特性（VSS=0V，VDD=5.0V，测试温度=25℃）（**STC8F2K 系列**）

标号	参数	范围				测试环境
		最小值	典型值	最大值	单位	
I_{PD}	掉电模式电流（SCC = 1）	-	0.08	-	uA	5.0V
	掉电模式电流（SCC = 0）	-	1.5	-	uA	5.0V
I_{WKT}	掉电唤醒定时器	-	5	-	uA	5.0V
I_{LVD}	低压检测模块	-	260	-	uA	5.0V
I_{IDL}	空闲模式电流（6MHz）	-	1.3	-	mA	5.0V
	空闲模式电流（11.0592MHz）		1.7		mA	5.0V
	空闲模式电流（20MHz）		2.3		mA	5.0V
	空闲模式电流（22.1184MHz）	-	2.5	-	mA	5.0V
	空闲模式电流（24MHz）	-	2.6	-	mA	5.0V
	空闲模式电流（内部 32KHz）	-	850	-	uA	5.0V
I_{NOR}	正常模式电流（6MHz）	-	2.7	-	mA	5.0V
	正常模式电流（11.0592MHz）	-	3.8	-	mA	5.0V
	正常模式电流（20MHz）	-	5.9	-	mA	5.0V
	正常模式电流（22.1184MHz）	-	6.3	-	mA	5.0V
	正常模式电流（24MHz）	-	6.5	-	mA	5.0V
	正常模式电流（内部 32KHz）	-	950	-	uA	5.0V
I_{CC}	普通工作模式电流	-	4	20	mA	5.0V
V_{IL1}	输入低电平	-	-	1.4	V	5.0V（打开施密特触发）
		-	-	1.5	V	5.0V（关闭施密特触发）
V_{IH1}	输入高电平（普通 I/O）	1.7	-	-	V	5.0V（打开施密特触发）
		1.6	-	-	V	5.0V（关闭施密特触发）
V_{IH2}	输入高电平（复位脚）	1.6	-	1.7	V	5.0V
I_{OL1}	输出低电平的灌电流	-	20	-	mA	5.0V，端口电压 0.45V
I_{OH1}	输出高电平电流（双向模式）	200	270	-	uA	5.0V
I_{OH2}	输出高电平电流（推挽模式）	-	20	-	mA	5.0V，端口电压 2.4V
I_{IL}	逻辑 0 输入电流	-	-	50	uA	5.0V，端口电压 0V
I_{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	5.0V，端口电压 2.0V
R_{PU}	IO 口上拉电阻	4.1	4.2	4.4	KΩ	5.0V

R_{PU}	IO 口上拉电阻	5.8	5.9	6.0	KΩ	3.3V

直流特性（VSS=0V，VDD=5.0V，测试温度=25℃）（**STC8A8K 系列**）

标号	参数	范围				测试环境
		最小值	典型值	最大值	单位	
I_{PD}	掉电模式电流（SCC = 1）	-	0.08	-	uA	5.0V
	掉电模式电流（SCC = 0）	-	1.6	-	uA	5.0V
I_{WKT}	掉电唤醒定时器	-	5	-	uA	5.0V
I_{LVD}	低压检测模块	-	260	-	uA	5.0V
I_{IDL}	空闲模式电流（6MHz）	-	1.5	-	mA	5.0V
	空闲模式电流（11.0592MHz）		1.9		mA	5.0V
	空闲模式电流（20MHz）		2.7		mA	5.0V
	空闲模式电流（22.1184MHz）	-	3.0	-	mA	5.0V
	空闲模式电流（24MHz）	-	3.2	-	mA	5.0V
	空闲模式电流（内部 32KHz）	-	890	-	uA	5.0V
I_{NOR}	正常模式电流（6MHz）	-	3.0	-	mA	5.0V
	正常模式电流（11.0592MHz）	-	4.3	-	mA	5.0V
	正常模式电流（20MHz）	-	6.8	-	mA	5.0V
	正常模式电流（22.1184MHz）	-	7.4	-	mA	5.0V
	正常模式电流（24MHz）	-	7.8	-	mA	5.0V
	正常模式电流（内部 32KHz）	-	950	-	uA	5.0V
I_{CC}	普通工作模式电流	-	4	20	mA	5.0V
V_{IL1}	输入低电平	-	-	1.4	V	5.0V（打开施密特触发）
		-	-	1.5	V	5.0V（关闭施密特触发）
V_{IH1}	输入高电平（普通 I/O）	2.2	-	-	V	5.0V（打开施密特触发）
		1.6	-	-	V	5.0V（关闭施密特触发）
V_{IH2}	输入高电平（复位脚）	2.2	-	-	V	5.0V
I_{OL1}	输出低电平的灌电流	-	20	-	mA	5.0V，端口电压 0.45V
I_{OH1}	输出高电平电流（双向模式）	200	270	-	uA	5.0V
I_{OH2}	输出高电平电流（推挽模式）	-	20	-	mA	5.0V，端口电压 2.4V
I_{IL}	逻辑 0 输入电流	-	-	50	uA	5.0V，端口电压 0V
I_{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	5.0V，端口电压 2.0V
R_{PU}	IO 口上拉电阻	4.1	4.2	4.4	KΩ	5.0V
R_{PU}	IO 口上拉电阻	5.8	5.9	6.0	KΩ	3.3V

内部 IRC 温漂特性（参考温度 25℃）

温度	范围
-40℃～85℃	−1.8%～+0.8%
-20℃～65℃	−1.0%～+0.5%

附录E　更新记录

- **2018/12/8**

1. 更新选型价格表中的芯片的参考价格

- **2018/12/6**

1. 创建 STC8H 系列单片机技术参考手册文档