STC 车规 32位 8051 AEC-Q100 Grade1
超强抗干扰，超级加密
车规 AEC-Q100 Grade1 MCU 设计公司，高稳定，低功耗
官网/技术论坛：www.STCAI.com 分销电话：0513-5501 2928, 8989 6509
深圳国芯人工智能有限公司
车规 AEC-Q100 Grade1 MCU 设计公司

STC8C 系列
单片机原理及应用

STC MCU

技术支持网站：www.STCAI.com
官方技术论坛：http://www.STCAIMCU.com
资料更新日期：**2025/3/4**
(本文档可直接添加备注和标记)

车规启航
扫码去微信小商城

目录

1 单片机基础概述 1
1.1 数制与编码 1
1.1.1 数制转换 1
1.1.2 原码、反码及补码 4
1.1.3 常用编码 5
1.2 几种常用的逻辑运算及其图形符号 5
1.3 STC8C 单片机性能概述 9
1.4 STC8C 单片机产品线 9
2 STC8C 系列各子系列选型简介、特性、价格、管脚图 10
2.1 STC8C2K64S4-36I-LQFP44/32,QFN32 系列 10
2.1.1 特性及价格（有 16 位硬件乘除法器 MDU16，准 16 位单片机） 10
2.1.2 STC8C2K64S4-36I-LQFP44 管脚图，ISP 下载/编程线路图 13
2.1.3 STC8C2K64S4-36I-LQFP32/QFN32 管脚图，ISP 下载/编程线路图 15
2.1.4 STC8C2K64S4-36I-PDIP40 管脚图，ISP 下载/编程线路图 17
2.1.5 管脚说明 19
2.1.6 USB-Link1D 对 STC8C 系列进行自动串口烧录、仿真+串口通讯 23
2.1.7 使用【一箭双雕之 USB 转双串口】进行串口烧录、仿真+串口通讯 24
2.1.8 USB 转双串口芯片全自动停电/上电烧录，串口仿真+串口通讯，5V 25
2.1.9 USB 转双串口芯片全自动烧录，串口仿真+串口通讯，3.3V 原理图 26
2.1.10 USB 转双串口芯片进行自动烧录/仿真+串口通讯，5V/3.3V 跳线选择 27
2.1.11 通用 USB 转串口芯片全自动停电/上电烧录，串口仿真，5V 原理图 28
2.1.12 通用 USB 转串口芯片全自动停电/上电烧录，串口仿真，3.3V 原理图 29
2.1.13 USB 转串口芯片全自动停电/上电烧录/仿真/通信，5V/3.3V 跳线选择 30
2.1.14 USB 转串口芯片进行烧录/串口仿真，手动停电/上电，5V/3.3V 原理图 31
2.1.15 USB 转串口芯片进行烧录，串口仿真，手动停电/上电，3.3V 原理图 32
2.1.16 USB-Link1D 支持 脱机下载 说明 33
2.1.17 USB-Link1D 支持 脱机下载，如何免烧录环节 34
2.1.18 USB-Writer1A 编程器/烧录器 支持 插在 锁紧座上 烧录 36
2.1.19 USB-Writer1A 支持 自动烧录机，通信协议和接口 37
2.2 STC8C2K64S2-36I-LQFP44/32,QFN32,TSSOP/20 系列 39
2.2.1 特性及价格（有 16 位硬件乘除法器 MDU16，准 16 位单片机） 39
2.2.2 STC8C2K64S4-36I-LQFP44 管脚图，ISP 下载/编程线路图 42
2.2.3 STC8C2K64S4-36I-LQFP32/QFN32 管脚图，ISP 下载/编程线路图 44
2.2.4 STC8C2K64S4-36I-PDIP40 管脚图，ISP 下载/编程线路图 46
2.2.5 STC8C2K64S4-36I-TSSOP20/SOP20 管脚图，ISP 下载/编程线路图 48
2.2.6 管脚说明 49
2.2.7 USB-Link1D 对 STC8C 系列进行自动串口烧录、仿真+串口通讯 53
2.2.8 使用【一箭双雕之 USB 转双串口】进行串口烧录、仿真+串口通讯 54
2.2.9 USB 转双串口芯片全自动停电/上电烧录，串口仿真+串口通讯，5V 55
2.2.10 USB 转双串口芯片全自动烧录，串口仿真+串口通讯，3.3V 原理图 56
2.2.11 USB 转双串口芯片进行自动烧录/仿真+串口通讯，5V/3.3V 跳线选择 57

2.2.12 通用 USB 转串口芯片全自动停电/上电烧录，串口仿真，5V 原理图 58
2.2.13 通用 USB 转串口芯片全自动停电/上电烧录，串口仿真，3.3V 原理图 59
2.2.14 USB 转串口芯片全自动停电/上电烧录/仿真/通信，5V/3.3V 跳线选择 60
2.2.15 USB 转串口芯片进行烧录/串口仿真，手动停电/上电，5V/3.3V 原理图 61
2.2.16 USB 转串口芯片进行烧录，串口仿真，手动停电/上电，3.3V 原理图 62
2.2.17 USB-Link1D 支持 脱机下载 说明 63
2.2.18 USB-Link1D 支持 脱机下载，如何免烧录环节 64
2.2.19 USB-Writer1A 编程器/烧录器 支持 插在 锁紧座上 烧录 66
2.2.20 USB-Writer1A 支持 自动烧录机，通信协议和接口 67
2.3 USB-Link1D 工具使用注意事项 69
2.3.1 工具接口说明 69
2.3.2 附送的各种强大的人性化配线图片及使用说明 70
2.3.3 USB-Link1D 实际应用 75
2.3.4 USB-Link1D 插上电脑并正常识别到后的显示 82
2.3.5 如电脑端新版本 ISP 软件提示 USB-Link1D 工具固件需升级 83
2.3.6 制作 USB-Link1D 主控芯片 84
2.4 ISP 下载相关硬件选项的说明 87
2.5 通用 USB 转双串口芯片：STC USB-2UART, TSSOP20/SOP16 88
2.5.1 通用 USB 转双串口芯片 STC USB-2UART-45I-TSSOP20，自动停电上电 88
2.5.2 通用 USB 转双串口芯片 STC USB-2UART-45I-TSSOP20，手动停电上电 89
2.5.3 通用 USB 转双串口芯片 STC USB-2UART-45I- SOP16，自动停电上电 90
2.5.4 通用 USB 转双串口芯片 STC USB-2UART-45I-SOP16，手动停电上电 91
3 功能脚切换 92
3.1 功能脚切换相关寄存器 92
3.1.1 外设端口切换控制寄存器 1（P_SW1），串口 1、SPI 切换 92
3.1.2 外设端口切换控制寄存器 2（P_SW2），串口 2、I2C、比较器输出切换 93
3.1.3 时钟选择寄存器（MCLKOCR） 93
3.2 范例程序 94
3.2.1 串口 1 切换 94
3.2.2 串口 2 切换 95
3.2.3 SPI 切换 97
3.2.4 I2C 切换 98
3.2.5 比较器输出切换 100
3.2.6 主时钟输出切换 101
4 封装尺寸图 104
4.1 LQFP32 封装尺寸图（9mm*9mm） 104
4.2 QFN32 封装尺寸图（4mm*4mm） 105
4.3 LQFP44 封装尺寸图（12mm*12mm） 106
4.4 STC8C 系列单片机命名规则 107
5 编译、仿真开发环境的建立与 ISP 下载 108
5.1 安装 Keil 108
5.1.1 安装 C51 编译环境 108
5.1.2 如何同时安装 Keil 的 C51、C251 和 MDK 111
5.2 添加型号和头文件到 Keil 112

5.3 STC 单片机程序中头文件的使用方法 114
5.4 新建项目与项目设置 116
5.4.1 设置项目路径和项目名称 116
5.4.2 选择目标单片机型号（STC8H8K64U） 117
5.4.3 添加源代码文件到项目 118
5.4.4 设置项目 1（设置“Memory Model”） 119
5.4.5 设置项目 3（“Code Rom Size”选择 Large） 121
5.4.6 设置项目 5（HEX 文件格式设置） 122
5.5 如何在 Keil C51 中对变量、表格数据、函数指定绝对地址 123
5.5.1 Keil C51 中，变量如何指定绝对地址 123
5.5.2 Keil C51 中，表格数据如何指定绝对地址 124
5.5.3 Keil C51 中，函数如何指定绝对地址 126
5.6 Keil 软件中获取帮助的简单方法 128
5.7 在 Keil 中建立多文件项目的方法 131
5.8 Keil 小技巧----在多文件项目的源文件中声明并使用外部变量和外部函数 134
5.9 扩展 Keil 对中断号数量的支持，中断号大于 31 编译出错的处理 136
5.9.1 如何下载 Keil 中断号拓展工具，及安装 136
5.9.2 如不使用扩展中断号工具，则需借用保留中断号进行中转 139
5.10 用户程序复位到系统区进行 USB 模式 ISP 下载的方法（不停电） 148
5.11 ISP 下载流程及典型应用线路图 151
5.11.1 ISP 下载流程图（串口下载模式） 151
5.11.2 ISP 下载流程图（硬件/软件模拟 USB+串口模式） 152
5.11.3 使用 USB-Link1D 工具下载，支持在线和脱机下载 153
5.11.4 软件模拟 USB 直接 ISP 下载，建议尝试，不支持仿真（5V 系统） 156
5.11.5 软件模拟 USB 直接 ISP 下载，建议尝试，不支持仿真（3.3V 系统） 158
5.11.6 使用一箭双雕之 USB 转串口工具下载 160
5.11.7 使用 USB 转双串口/TTL 下载（有外部晶振） 162
5.11.8 使用 USB 转双串口/TTL 下载（无外部晶振） 163
5.11.9 使用 USB 转双串口/TTL 下载（自动停电/上电） 165
5.11.10 使用 USB 转双串口/RS485 下载（5.0V） 166
5.11.11 使用 USB 转双串口/RS485 下载（3.3V） 166
5.11.12 使用 USB 转双串口/RS232 下载（5.0V） 168
5.11.13 使用 USB 转双串口/RS232 下载（3.3V） 168
5.11.14 使用 U8-Mini 工具下载，支持 ISP 在线和脱机下载，也可支持仿真 169
5.11.15 使用 U8W 工具下载，支持 ISP 在线和脱机下载，也可支持仿真 171
5.11.16 使用 RS-232 转换器下载，也可支持仿真 173
5.11.17 使用 PL2303-GL 下载，也可支持仿真 174
5.11.18 单片机电源控制参考电路 175
5.12 用 STC 一箭双雕之 USB 转双串口仿真 STC8 系列 MCU 176
5.13 AIapp-ISP 下载软件高级应用 184
5.13.1 发布项目程序 184
5.13.2 程序加密后传输（防烧录时串口分析出程序） 188
5.13.3 发布项目程序+程序加密后传输结合使用 192
5.13.4 用户自定义下载（实现不停电下载） 194

5.13.5 如何简单的控制下载次数，通过 ID 号来限制实际可以下载的 MCU 数量 198
6 时钟、复位、省电模式与系统电源管理，芯片上电工作过程 205
6.1 系统时钟控制 205
6.2 芯片上电工作过程： 206
6.3 相关寄存器 207
6.3.1 系统时钟选择寄存器（CLKSEL） 208
6.3.2 时钟分频寄存器（CLKDIV） 208
6.3.3 内部高速高精度 IRC 控制寄存器（HIRCCR） 208
6.3.4 外部振荡器控制寄存器（XOSCCR） 209
6.3.5 内部 32KHz 低速 IRC 控制寄存器（IRC32KCR） 209
6.3.6 主时钟输出控制寄存器（MCLKOCR） 210
6.3.7 内部高速 IRC 时钟恢复振荡到稳定需要等待的时钟数控制寄存器（IRCDB） 211
6.4 STC8C 系列内部 IRC 频率调整 212
6.4.1 IRC 频段选择寄存器（IRCBAND） 212
6.4.2 内部 IRC 频率调整寄存器（IRTRIM） 212
6.4.3 内部 IRC 频率微调寄存器（LIRTRIM） 213
6.4.4 时钟分频寄存器（CLKDIV） 213
6.4.5 分频出 3MHz 用户工作频率，并用户动态改变频率追频示例 214
6.5 系统复位 217
6.5.1 看门狗复位（WDT_CONTR） 218
6.5.2 软件复位（IAP_CONTR） 220
6.5.3 低压复位（RSTCFG） 220
6.5.4 低电平上电复位参考电路（一般不需要） 221
6.5.5 低电平按键手动复位参考电路 221
6.5.6 传统 8051 高电平上电复位参考电路 222
6.6 外部晶振及外部时钟电路 223
6.6.1 外部晶振输入电路 223
6.6.2 外部时钟输入电路（P1.6 为高阻输入模式，可当输入口使用） 223
6.7 时钟停振/省电模式与系统电源管理 224
6.7.1 电源控制寄存器（PCON） 224
6.7.2 内部高速 IRC 时钟恢复振荡到稳定需要等待的时钟数控制寄存器（IRCDB） 225
6.8 掉电唤醒定时器 226
6.8.1 掉电唤醒定时器计数寄存器（WKTCL，WKTCH） 226
6.9 范例程序 227
6.9.1 选择系统时钟源 227
6.9.2 主时钟分频输出 229
6.9.3 看门狗定时器应用 231
6.9.4 软复位实现自定义下载 233
6.9.5 低压检测 235
6.9.6 省电模式 237
6.9.7 使用 INT0/INT1/INT2/INT3/INT4 管脚中断唤醒省电模式 239
6.9.8 使用 T0/T1/T2/T3/T4 管脚中断唤醒 MCU 省电模式 242
6.9.9 使用 RxD/RxD2/RxD3/RxD4 管脚中断唤醒 MCU 省电模式 247
6.9.10 使用 I2C 的 SDA 脚唤醒 MCU 省电模式 250

6.9.11 使用掉电唤醒定时器唤醒省电模式 253
6.9.12 LVD 中断唤醒省电模式，建议配合使用掉电唤醒定时器 255
6.9.13 比较器中断唤醒省电模式，建议配合使用掉电唤醒定时器 257
6.9.14 使用 LVD 功能检测工作电压（电池电压） 260
7 存储器 265
7.1 程序存储器 265
7.2 数据存储器 266
7.2.1 内部 RAM 266
7.2.2 程序状态寄存器（PSW） 266
7.2.3 内部扩展 RAM，XRAM，XDATA 267
7.2.4 辅助寄存器（AUXR） 267
7.2.5 外部扩展 RAM，XRAM，XDATA 268
7.2.6 总线速度控制寄存器（BUS_SPEED） 268
7.2.7 8051 中可位寻址的数据存储器 269
7.3 存储器中的特殊参数，在 ISP 下载时可烧录进程序 FLASH 271
7.3.1 读取内部 1.19V 参考信号源-BGV 值 (从 Flash 程序存储器（ROM）中读取) 272
~~7.3.2 读取内部 1.19V 参考信号源-BGV 值 (从 RAM 中读取)~~ 275
7.3.3 读取全球唯一 ID 号 (从 Flash 程序存储器（ROM）中读取) 278
~~7.3.4 读取全球唯一 ID 号 (从 RAM 中读取)~~ 281
7.3.5 读取 32K 掉电唤醒定时器的频率 (从 Flash 程序存储器（ROM）中读取) 284
~~7.3.6 读取 32K 掉电唤醒定时器的频率 (从 RAM 中读取)~~ 287
7.3.7 用户自定义内部 IRC 频率 (从 Flash 程序存储器（ROM）中读取) 290
~~7.3.8 用户自定义内部 IRC 频率 (从 RAM 中读取)~~ 296
8 特殊功能寄存器 299
8.1 STC8C2K64S4-44Pin/32Pin 系列 299
8.2 STC8C2K64S2-44Pin/32Pin/20Pin 系列 300
8.3 特殊功能寄存器列表 301
9 I/O 口 306
9.1 I/O 口相关寄存器 306
9.1.1 端口数据寄存器（Px） 307
9.1.2 端口模式配置寄存器（PxM0，PxM1） 308
9.1.3 端口上拉电阻控制寄存器（PxPU） 308
9.1.4 端口施密特触发控制寄存器（PxNCS） 309
9.1.5 端口电平转换速度控制寄存器（PxSR） 309
9.1.6 端口驱动电流控制寄存器（PxDR） 309
9.1.7 端口数字信号输入使能控制寄存器（PxIE） 310
9.2 配置 I/O 口 311
9.3 I/O 的结构图 312
9.3.1 准双向口（弱上拉） 312
9.3.2 推挽输出 312
9.3.3 高阻输入 313
9.3.4 开漏模式 313
9.3.5 新增 4.1K 上拉电阻 314
9.3.6 如何设置 I/O 口对外输出速度 314

9.3.7 如何设置 I/O 口电流驱动能力 315
9.3.8 如何降低 I/O 口对外辐射 315
9.4 AIapp-ISP | I/O 口配置工具 316
9.4.1 普通配置模式 316
9.4.2 高级配置模式 317
9.5 范例程序 318
9.5.1 端口模式设置 318
9.5.2 双向口读写操作 319
9.6 一种典型三极管控制电路 322
9.7 典型发光二极管控制电路 322
9.8 混合电压供电系统 3V/5V 器件 I/O 口互连 323
9.9 如何让 I/O 口上电复位时为低电平 325
9.10 利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)的线路图 326
9.11 I/O 口直接驱动 LED 数码管应用线路图 327
9.12 用 STC 系列 MCU 的 I/O 口直接驱动段码 LCD 328
10 指令系统 347
10.1 寻址方式 347
10.1.1 立即寻址 347
10.1.2 直接寻址 347
10.1.3 间接寻址 347
10.1.4 寄存器寻址 347
10.1.5 相对寻址 347
10.1.6 变址寻址 348
10.1.7 位寻址 348
10.2 指令表 348
10.3 指令详解（中文） 351
10.4 指令详解（英文） 384
10.5 多级流水线内核的中断响应 418
11 中断系统 421
11.1 STC8C 系列中断源 421
11.2 STC8C 中断结构图 423
11.3 STC8C 系列中断列表 424
11.4 中断相关寄存器 426
11.4.1 中断使能寄存器（中断允许位） 427
11.4.2 中断请求寄存器（中断标志位） 430
11.4.3 中断优先级寄存器 432
11.5 范例程序 434
11.5.1 INT0 中断（上升沿和下降沿），可同时支持上升沿和下降沿 434
11.5.2 INT0 中断（下降沿） 436
11.5.3 INT1 中断（上升沿和下降沿），可同时支持上升沿和下降沿 437
11.5.4 INT1 中断（下降沿） 439
11.5.5 INT2 中断（下降沿），只支持下降沿中断 441
11.5.6 INT3 中断（下降沿），只支持下降沿中断 443
11.5.7 INT4 中断（下降沿），只支持下降沿中断 445

11.5.8 定时器 0 中断 447
11.5.9 定时器 1 中断 448
11.5.10 定时器 2 中断 450
11.5.11 定时器 3 中断 452
11.5.12 定时器 4 中断 455
11.5.13 UART1 中断 457
11.5.14 UART2 中断 459
11.5.15 UART3 中断 462
11.5.16 UART4 中断 464
11.5.17 LVD 中断 467
11.5.18 SPI 中断 469
11.5.19 比较器中断 471
11.5.20 I2C 中断 473
12 普通 I/O 口均可中断，不是传统外部中断 477
12.1 I/O 口中断相关寄存器 477
12.1.1 端口中断使能寄存器（PxINTE） 478
12.1.2 端口中断标志寄存器（PxINTF） 478
12.1.3 端口中断模式配置寄存器（PxIM0，PxIM1） 479
12.2 范例程序 480
12.2.1 P0 口下降沿中断 480
12.2.2 P1 口上升沿中断 483
12.2.3 P2 口低电平中断 487
12.2.4 P3 口高电平中断 491
13 定时器/计数器 496
13.1 定时器的相关寄存器 497
13.2 定时器 0/1 498
13.2.1 定时器 0/1 控制寄存器（TCON） 498
13.2.2 定时器 0/1 模式寄存器（TMOD） 499
13.2.3 定时器 0 模式 0（16 位自动重装载模式） 500
13.2.4 定时器 0 模式 1（16 位不可重装载模式） 501
13.2.5 定时器 0 模式 2（8 位自动重装载模式） 502
13.2.6 定时器 0 模式 3（不可屏蔽中断 16 位自动重装载，实时操作系统节拍器） 503
13.2.7 定时器 1 模式 0（16 位自动重装载模式） 504
13.2.8 定时器 1 模式 1（16 位不可重装载模式） 505
13.2.9 定时器 1 模式 2（8 位自动重装载模式） 506
13.2.10 定时器 0 计数寄存器（TL0，TH0） 506
13.2.11 定时器 1 计数寄存器（TL1，TH1） 506
13.2.12 辅助寄存器 1（AUXR） 507
13.2.13 中断与时钟输出控制寄存器（INTCLKO） 507
13.2.14 定时器 0 定时计算公式 508
13.2.15 定时器 1 定时计算公式 509
13.3 定时器 2（24 位定时器，8 位预分频+16 位定时） 510
13.3.1 辅助寄存器 1（AUXR） 510
13.3.2 中断与时钟输出控制寄存器（INTCLKO） 510

13.3.3 定时器 2 计数寄存器（T2L，T2H）.....510
13.3.4 定时器 2 的 8 位预分频寄存器（TM2PS）.....510
13.3.5 定时器 2 工作模式.....511
13.3.6 定时器 2 计算公式.....511
13.4 定时器 3/4（24 位定时器，8 位预分频+16 位定时）.....512
13.4.1 定时器 4/3 控制寄存器（T4T3M）.....512
13.4.2 定时器 3 计数寄存器（T3L，T3H）.....513
13.4.3 定时器 4 计数寄存器（T4L，T4H）.....513
13.4.4 定时器 3 的 8 位预分频寄存器（TM3PS）.....513
13.4.5 定时器 4 的 8 位预分频寄存器（TM4PS）.....513
13.4.6 定时器 3 工作模式.....514
13.4.7 定时器 4 工作模式.....515
13.4.8 定时器 3 计算公式.....516
13.4.9 定时器 4 计算公式.....516
13.5 AIapp-ISP | 定时器计算器工具.....517
13.6 范例程序.....518
13.6.1 定时器 0（模式 0－16 位自动重载），用作定时.....518
13.6.2 定时器 0（模式 1－16 位不自动重载），用作定时.....519
13.6.3 定时器 0（模式 2－8 位自动重载），用作定时.....521
13.6.4 定时器 0（模式 3－16 位自动重载不可屏蔽中断），用作定时.....523
13.6.5 定时器 0（外部计数－扩展 T0 为外部下降沿中断）.....525
13.6.6 定时器 0（测量脉宽－INT0 高电平宽度）.....527
13.6.7 定时器 0（模式 0），时钟分频输出.....529
13.6.8 定时器 1（模式 0－16 位自动重载），用作定时.....531
13.6.9 定时器 1（模式 1－16 位不自动重载），用作定时.....533
13.6.10 定时器 1（模式 2－8 位自动重载），用作定时.....534
13.6.11 定时器 1（外部计数－扩展 T1 为外部下降沿中断）.....536
13.6.12 定时器 1（测量脉宽－INT1 高电平宽度）.....538
13.6.13 定时器 1（模式 0），时钟分频输出.....540
13.6.14 定时器 1（模式 0）做串口 1 波特率发生器.....542
13.6.15 定时器 1（模式 2）做串口 1 波特率发生器.....546
13.6.16 定时器 2（16 位自动重载），用作定时.....550
13.6.17 定时器 2（外部计数－扩展 T2 为外部下降沿中断）.....552
13.6.18 定时器 2，时钟分频输出.....554
13.6.19 定时器 2 做串口 1 波特率发生器.....556
13.6.20 定时器 2 做串口 2 波特率发生器.....559
13.6.21 定时器 2 做串口 3 波特率发生器.....563
13.6.22 定时器 2 做串口 4 波特率发生器.....567
13.6.23 定时器 3（16 位自动重载），用作定时.....571
13.6.24 定时器 3（外部计数－扩展 T3 为外部下降沿中断）.....574
13.6.25 定时器 3，时钟分频输出.....576
13.6.26 定时器 3 做串口 3 波特率发生器.....578
13.6.27 定时器 4（16 位自动重载），用作定时.....582
13.6.28 定时器 4（外部计数－扩展 T4 为外部下降沿中断）.....585

13.6.29 定时器 4，时钟分频输出 587
13.6.30 定时器 4 做串口 4 波特率发生器 589
14 串口通信 594
14.1 串口相关寄存器 594
14.2 串口 1 595
14.2.1 串口 1 控制寄存器（SCON） 595
14.2.2 串口 1 数据寄存器（SBUF） 595
14.2.3 电源管理寄存器（PCON） 596
14.2.4 辅助寄存器 1（AUXR） 596
14.2.5 串口 1 模式 0，模式 0 波特率计算公式 597
14.2.6 串口 1 模式 1，模式 1 波特率计算公式 599
14.2.7 串口 1 模式 2，模式 2 波特率计算公式 601
14.2.8 串口 1 模式 3，模式 3 波特率计算公式 602
14.2.9 自动地址识别，从机地址控制寄存器（SADDR，SADEN） 603
14.3 串口 2 604
14.3.1 串口 2 控制寄存器（S2CON） 604
14.3.2 串口 2 数据寄存器（S2BUF） 604
14.3.3 串口 2 模式 0，模式 0 波特率计算公式 605
14.3.4 串口 2 模式 1，模式 1 波特率计算公式 606
14.4 串口 3 607
14.4.1 串口 3 控制寄存器（S3CON） 607
14.4.2 串口 3 数据寄存器（S3BUF） 607
14.4.3 串口 3 模式 0，模式 0 波特率计算公式 608
14.4.4 串口 3 模式 1，模式 1 波特率计算公式 609
14.5 串口 4 610
14.5.1 串口 4 控制寄存器（S4CON） 610
14.5.2 串口 4 数据寄存器（S4BUF） 610
14.5.3 串口 4 模式 0，模式 0 波特率计算公式 611
14.5.4 串口 4 模式 1，模式 1 波特率计算公式 612
14.6 串口注意事项 613
14.7 AIapp-ISP | 串口波特率计算器工具 614
14.8 AIapp-ISP | 串口助手/USB-CDC 615
14.9 范例程序 619
14.9.1 串口 1 使用定时器 2 做波特率发生器 619
14.9.2 串口 1 使用定时器 1（模式 0）做波特率发生器 622
14.9.3 串口 1 使用定时器 1（模式 2）做波特率发生器 626
14.9.4 串口 2 使用定时器 2 做波特率发生器 630
14.9.5 串口 3 使用定时器 2 做波特率发生器 634
14.9.6 串口 3 使用定时器 3 做波特率发生器 638
14.9.7 串口 4 使用定时器 2 做波特率发生器 642
14.9.8 串口 4 使用定时器 4 做波特率发生器 646
14.9.9 串口多机通讯 651
14.9.10 串口转 LIN 总线 652
15 比较器，掉电检测，内部 1.19V 参考信号源（BGV） 661

15.1 比较器内部结构图 661
15.2 比较器相关的寄存器 662
15.2.1 比较器控制寄存器 1（CMPCR1） 662
15.2.2 比较器控制寄存器 2（CMPCR2） 663
15.3 范例程序 664
15.3.1 比较器的使用（中断方式） 664
15.3.2 比较器的使用（查询方式） 666
15.3.3 比较器作外部掉电检测（掉电过程中应及时保存用户数据到 EEPROM 中） 669
15.3.4 比较器检测工作电压（电池电压） 670
16 IAP/EEPROM/DATA-FLASH 675
16.1 EEPROM 操作时间 675
16.2 EEPROM 相关的寄存器 676
16.2.1 EEPROM 数据寄存器（IAP_DATA） 676
16.2.2 EEPROM 地址寄存器（IAP_ADDR） 676
16.2.3 EEPROM 命令寄存器（IAP_CMD） 676
16.2.4 EEPROM 触发寄存器（IAP_TRIG） 677
16.2.5 EEPROM 控制寄存器（IAP_CONTR） 677
16.2.6 EEPROM 等待时间控制寄存器（IAP_TPS） 677
16.3 EEPROM 大小及地址 678
16.4 范例程序 681
16.4.1 EEPROM 基本操作 681
16.4.2 使用 MOVC 读取 EEPROM 684
16.4.3 使用串口送出 EEPROM 数据 688
17 同步串行外设接口 SPI 693
17.1 SPI 相关的寄存器 693
17.1.1 SPI 状态寄存器（SPSTAT） 693
17.1.2 SPI 控制寄存器（SPCTL），SPI 速度控制 694
17.1.3 SPI 数据寄存器（SPDAT） 694
17.2 SPI 通信方式 695
17.2.1 单主单从 695
17.2.2 互为主从 696
17.2.3 单主多从 697
17.3 配置 SPI 698
17.4 数据模式 700
17.5 范例程序 701
17.5.1 SPI 单主单从系统主机程序（中断方式） 701
17.5.2 SPI 单主单从系统从机程序（中断方式） 703
17.5.3 SPI 单主单从系统主机程序（查询方式） 705
17.5.4 SPI 单主单从系统从机程序（查询方式） 707
17.5.5 SPI 互为主从系统程序（中断方式） 709
17.5.6 SPI 互为主从系统程序（查询方式） 712
18 I^2C 总线 716
18.1 I^2C 相关的寄存器 716
18.2 I^2C 主机模式 717

18.2.1 I2C 配置寄存器（I2CCFG），总线速度控制 717
18.2.2 I2C 主机控制寄存器（I2CMSCR） 718
18.2.3 I2C 主机辅助控制寄存器（I2CMSAUX） 719
18.2.4 I2C 主机状态寄存器（I2CMSST） 720
18.3 I^2C 从机模式 721
18.3.1 I2C 从机控制寄存器（I2CSLCR） 721
18.3.2 I2C 从机状态寄存器（I2CSLST） 721
18.3.3 I2C 从机地址寄存器（I2CSLADR） 723
18.3.4 I2C 数据寄存器（I2CTXD，I2CRXD） 724
18.4 范例程序 725
18.4.1 I^2C 主机模式访问 AT24C256（中断方式） 725
18.4.2 I^2C 主机模式访问 AT24C256（查询方式） 731
18.4.3 I^2C 主机模式访问 PCF8563 736
18.4.4 I^2C 从机模式（中断方式） 742
18.4.5 I^2C 从机模式（查询方式） 747
18.4.6 测试 I^2C 从机模式代码的主机代码 751
19 增强型双数据指针 758
19.1 相关的特殊功能寄存器 758
19.1.1 第 1 组 16 位数据指针寄存器（DPTR0） 758
19.1.2 第 2 组 16 位数据指针寄存器（DPTR1） 758
19.1.3 数据指针控制寄存器（DPS） 759
19.1.4 数据指针控制寄存器（TA） 760
19.2 范例程序 761
19.2.1 示例代码 1 761
19.2.2 示例代码 2 762
20 MDU16 硬件 16 位乘除法器 764
20.1 相关的特殊功能寄存器 764
20.1.1 操作数 1 数据寄存器（MD0～MD3） 765
20.1.2 操作数 2 数据寄存器（MD4～MD5） 765
20.1.3 MDU 模式控制寄存器（ARCON），运算所需时钟数 766
20.1.4 MDU 操作控制寄存器（OPCON） 766
20.2 关于 MDU16 的网友应用杂谈（提供思路，仅供参考） 767
20.3 范例程序 769
附录 A STC8C 系列单片机取代 STC8F 系列的注意事项 771
附录 B 电气特性 772
B.1 绝对最大额定值 772
B.2 直流特性（3.3V） 773
B.3 直流特性（5.0V） 775
B.4 I/O 口驱动能力（驱动电流对应的 I/O 上的电压） 776
B.5 内部 IRC 温漂特性（参考温度 25°C） 776
B.6 低压复位门槛电压（测试温度 25°C） 777
附录 C 关于回流焊前是否要烘烤 778
附录 D 如何使用万用表检测芯片 I/O 口好坏 779
附录 E 大批量生产，如何省去专门的烧录人员，如何无烧录环节 780

附录 F 如何使用 AIapp-ISP 下载软件制作和编辑 EEPROM 文件 781
附录 G 编译器（汇编器）/仿真器使用指南 782
附录 H AIapp-ISP 下载软件高级应用 788
H.1 发布项目程序 788
H.2 程序加密后传输（防烧录时串口分析出程序） 792
H.3 发布项目程序+程序加密后传输结合使用 795
H.4 用户自定义下载（实现不停电下载） 796
附录 I 如何让传统的 8051 单片机学习板可仿真 800
附录 J STC-USB 驱动程序安装说明 802
附录 K USB 下载步骤演示 837
附录 L RS485 自动控制或 I/O 口控制线路图 840
附录 M STC 工具使用说明书 841
M.1 概述 841
M.2 系统可编程（ISP）流程说明 841
M.3 USB 型联机/脱机下载工具 U8W/U8W-Mini 842
M.3.1 安装 U8W/U8W-Mini 驱动程序 844
M.3.2 U8W 的功能介绍 847
M.3.3 U8W 的在线联机下载使用说明 848
M.3.4 U8W 的脱机下载使用说明 851
M.3.5 U8W-Mini 的功能介绍 858
M.3.6 U8W-Mini 的在线联机下载使用说明 859
M.3.7 U8W-Mini 的脱机下载使用说明 860
M.3.8 制作/更新 U8W/U8W-Mini 865
M.3.9 U8W/U8W-Mini 设置直通模式（可用于仿真） 867
M.3.10 U8W/U8W-Mini 的参考电路 867
M.4 STC 通用 USB 转串口工具 869
M.4.1 STC 通用 USB 转串口工具外观图 869
M.4.2 STC 通用 USB 转串口工具布局图 870
M.4.3 STC 通用 USB 转串口工具驱动安装 871
M.4.4 使用 STC 通用 USB 转串口工具下载程序到 MCU 872
M.4.5 使用 STC 通用 USB 转串口工具仿真用户代码 874
M.5 应用线路图 881
M.5.1 U8W 工具应用参考线路图 881
M.5.2 STC 通用 USB 转串口工具应用参考线路图 881
附录 N U8W 下载工具中 RS485 部分线路图 883
附录 O 运行用户程序时收到用户命令后自动启动 ISP 下载(不停电) 884
附录 P 使用 STC 的 IAP 系列单片机开发自己的 ISP 程序 885
附录 Q 用户程序复位到系统区进行 ISP 下载的方法（不停电） 897
附录 R 使用第三方 MCU 对 STC8C 系列单片机进行 ISP 下载范例程序 903
附录 S 使用第三方应用程序调用 STC 发布项目程序对单片机进行 ISP 下载 911
附录 T 在 Keil 中建立多文件项目的方法 914
附录 U 关于中断号大于 31 在 Keil 中编译出错的处理 917
附录 V 触摸按键的 PCB 设计指导 926
附录 W QFN/DFN 封装元器件焊接方法 928

附录 X 可以自己去生产的【USB 转双串口】资料 931
附录 Y 关于 Keil 软件中 0xFD 问题的说明.....932
附录 Z 单片机是否可以提供裸芯.....933
附录 AA 官方网址说明.....934
附录 BB STC8 系列命名花絮 935
附录 CC 更新记录.....936

STC MCU

1　单片机基础概述

——无微机原理的用户请从本章开始学习

这一章主要讲述的内容有：①在数字设备中进行算术运算的基本知识——数制和编码；②数字电路中一些常用逻辑运算及其图形符号。它们是学习单片机这门课程的基础。对于没有微机原理基础的用户和同学，请从这章开始学习。

1.1　数制与编码

数制是人们利用符号进行计数的科学方法。

数制有很多种，常用的数制有：二进制，十进制和十六进制。

进位计数制是把数划分为不同的位数，逐位累加，加到一定数量之后，再从零开始，同时向高位进位。进位计数制有三个要素：数码符号、进位规律和计数基数。下表是各常用数制的总体介绍。

常用的数制	表示符号	数码符号	进制规律	计数基数
二进制	B	0、1	逢二进一	2
十进制	D	0、1、2、3、4、5、6、7、8、9	逢十进一	10
十六进制	H	0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F	逢十六进一	16

我们日常生活中计数一般采用十进制。计算机中采用的是二进制，因为二进制具有运算简单，易实现且可靠，为逻辑设计提供了有利的途径、节省设备等优点。为区别于其它进制数，二进制数的书写通常在数的右下方注上基数 2，或加后面加 B 表示。二进制数中每一位仅有 0 和 1 两个可能的数码，所以计数基数为 2。二进制数的加法和乘法运算如下：

0 + 0 = 0　　0 + 1 = 1 + 0 = 1　　1 + 1 = 10

0 × 0 = 0　　0 × 1 = 1 × 0 =0　　1 × 1 = 1

由于二进制数在使用中位数太长,不容易记忆，为了便于描述，又常用十六进制作为二进制的缩写。十六进制通常在表示时用尾部标志 H 或下标 16 以示区别。

1.1.1　数制转换

现在我们来介绍这些常用数制之间的转换。

一：二进制 — 十进制转换

方法：将二进制数按权(如下式)展开，然后将各项的数值按十进制数相加，就得到相应的等值十进制数。

例如：N=(1101.101)B，那么 N 所对应的十进制数时多少呢?

按权展开 $N=1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}=8+4+0+1+0.5+0+0.125=(13.625)D$

二：十进制 — 二进制转换

方法：分两部分进行即整数部分和小数部分。

①整数部分转换(基数除法)：

★ 把我们要转换的数除以二进制的基数(二进制的基数为 2)，把余数作为二进制的最低位；

★ 把上一次得的商在除以二进制基数(即 2)，把余数作为二进制的次低位；

★ 继续上一步,直到最后的商为零,这时的余数就是二进制的最高位.

②小数部分转换(基数乘法)：

★ 把要转换数的小数部分乘以二进制的基数(二进制的基数为 2)，把得到的整数部分作为二进制小数部分的最高位；

★ 把上一步得的小数部分再乘以二进制的基数(即 2)，把整数部分作为二进制小数部分的次高位；

★ 继续上一步，直到小数部分变成零为止。或者达到预定的要求也可以。

STC MCU

例如：将$(213.8125)_{10}$化为二进制数可按如下进行：
先化整数部分：

2	213	余数=1=k_0
2	106	余数=0=k_1
2	53	余数=1=k_2
2	26	余数=0=k_3
2	13	余数=1=k_4
2	6	余数=0=k_5
2	3	余数=1=k_6
2	1	余数=1=k_7
	0	

于是整数部分$(213)_{10}=(11010101)_2$

再化小数部分：

0.8125 × 2 = 1.6250 ------ 整数部分=1=k_{-1}

0.6250 × 2 = 1.2500 ------ 整数部分=1=k_{-2}

0.2500 × 2 = 0.5000 ------ 整数部分=0=k_{-3}

0.5000 × 2 = 1.0000 ------ 整数部分=1=k_{-4}

于是小数部分$(0.8125)_{10}=(0.1101)_2$

综上所述，十进制数 $213.8125=(11010101.1101)_2=(11010101.1101)B$

STC MCU

三：二进制 — 十六进制转换

方法：二进制和十六进制之间满足 24 的关系，因此把要转换的二进制从低位到高位每 4 位一组，高位不足时在有效位前面添“0”，然后把每组二进制数转换成十六进制即可。

例如，将(010111011110.10110010)B 转换为十六进制数：

(0101 1101 1110 . 1011 0010)B
↓ ↓ ↓ ↓ ↓
= (5 D E B 2)H

于是，(010111011110.10110010)B=(5DE.B2)H

四：十六进制 — 二进制转换

方法：十六进制转换为二进制时，把上面二进制转换十六进制的过程逆过来，即转换时只需将十六进制的每一位用等值的 4 位二进制代替就行了。

例如：将(C1B.C6)H 转换为二进制数：

(C 1 B . C 6)H
↓ ↓ ↓ ↓ ↓
= (1100 0001 1011 1100 0110)B

于是，(C1B.C6)H=(110000011011.11000110)B

五：十六进制 — 十进制转换

方法：将十六进制数按权(如下式)展开，然后将各项的数值按十进制数相加，就得到相应的等值十进制数。

例如：N=(2A.7F)H，那么 N 所对应的十进制数时多少呢？
按权展开 $N=2\times16^{1}+10\times16^{0}+7\times16^{-1}+15\times16^{-2}=32+10+0.4375+0.05859375=(42.49609375)D$
于是，(2A.7F)H =(42.49609375)D

六：十进制 — 十六进制转换

方法：将十进制数转换为十六进制数时，可以先将十进制数转换为二进制数，然后再将得到的二进制数转换为等值的十六进制数。

1.1.2 原码、反码及补码

在生活中,数有正负之分,在计算机中是怎样表示数的正负符号呢？

在生活中表示数的时候一般都是把正数前面加一个“+”，负数前面加一个“-”，但是计算机是不认识这些的，通常在二进制数前面增加一位符号位。符号位为“0”表示“+”，符号位为“1”表示“-”。这种形式的二进制数称为原码。如果原码为正数，则原码的反码和补码都与原码相同。如果原码为负数，则将原码(除符号位外)按位取反，所得的新二进制数称为原码的反码，反码加 1 为其补码。

原码、反码、补码这三种形式的总结如下表所示：

	真值	原码	反码	补码

正数	+N	0N	0N	0N
负数	-N	1N	$(2^n-1)+N$	2^n+N

例 1：求+18 和-18 八位原码、反码、补码形式。

真值	原码	反码	补码
+18	00010010	00010010	00010010
-18	10010010	11101101	11101110

1.1.3 常用编码

指定某一组二进制数去代表某一指定的信息，就称为编码。

一：十进制编码

用二进制码表示的十进制数，称为十进制编码。它具有二进制的形式，还具有十进制的特点它可作为人们与数字系统的联系的一种间表示。十进制编码有很多种，最常用的一种是 BCD 码，又称 8421 码。

下面我们用表列出几种常见的十进制编码:

编码种类 / 十进制数	8421 码（BCD 码）	余 3 码	2421 码	5211 码	7321 码
0	0000	0011	0000	0000	0000
1	0001	0100	0001	0001	0001
2	0010	0101	0010	0100	0010
3	0011	0110	0011	0101	0011
4	0100	0111	0100	0111	0101
5	0101	1000	1011	1000	0110
6	0110	1001	1100	1001	0111
7	0111	1010	1101	1100	1000
8	1000	1011	1110	1101	1001
9	1001	1100	1111	1111	1010
权	8421		2421	5211	7321

十进制编码分为有权和无权编码。有权编码码是指每一位十进制数符均用一组四位二进制码来表示，而且二进制码的每一位都有固定权值。无权编码码是指二进制码中每一位都没有固 定的权值。上表中 8421 码(即 BCD 码)、2421 码、5211 码、7321 码都是有权编码，而余 3 码是无权编码。

二：奇偶校验码

在数据的存取、运算和传送过程中，难免会发生错误，把“1”错成“0”或把“0”错成“1”。奇偶校验码是一种能检验这种错误的代码。它分为两部分；信息位和奇偶校验位。有奇数个“1”称为奇校验，有偶数个“1”则称为偶校验。

1.2 几种常用的逻辑运算及其图形符号

逻辑代数中常用的运算有：与(AND)、或(OR)、非(NOT)、与非(NAND)、或非(NOR)、与或非(AND-NOR)、异或(EXCLUSIVE OR)、同或(EXCLUSIVE NOR)等。其中与(AND)、或(OR)、非(NOT)运算时三种最基本的运算。

一：与运算及与门

与运算：决定事件结果的全部条件同时具备时，事件才发生。

逻辑变量 A 和 B 进行与运算时可写成：Y=A • B

真值表		
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

与门：实行与逻辑运算的单元电路。

与门图形符号： 

二：或运算及或门

或运算：决定事件结果的各条件中只要有任何一个满足，事件就会发生。

逻辑变量 A 和 B 进行或运算时可写成：Y=A+B

真值表		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

或门：实行或逻辑运算的单元电路。

或门图形符号： 

三：非运算及非门

非运算：条件具备时，事件不会发生；条件不具备时，事件才会发生。

逻辑变量 A 进行非运算时可写成：Y=A′

真值表	
A	Y
0	1
1	0

非门：实行非逻辑运算的单元电路。

非门图形符号： 

四：与非运算及与非图形符号

与非运算：先进行与运算，然后将结果求反，最后得到的即为与非运算结果。

逻辑变量 A 和 B 进行与非运算时可写成：Y=(A • B)′

真值表		
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

与非图形符号： 

五：或非运算及或非图形符号

或非运算：先进行或运算，然后将结果求反，最后得到的即为或非运算结果。

逻辑变量 A 和 B 进行或非运算时可写成：Y=(A+B)′

真值表

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

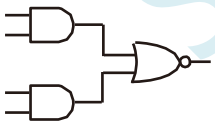
或非图形符号： 

六：与或非运算及与或非图形符号

与或非运算：在与或非逻辑运算中有 4 个逻辑变量 A、B、C、D。假设 A 和 B 为一组，C 和 D 为一 组，A、B 之间以及 C、D 之间都是与的关系，只要 A、B 或 C、D 任何一组同时为 1， 输出 Y 就是 0。只有当每一组输入都不全是 1 时，输出 Y 才是 1。

逻辑变量 A 和 B 进行或非运算时可写成：$Y=(A \cdot B+C \cdot D)'$

真值表				
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

与或非图形符号： 

七：异或运算及异或图形符号

异或运算：当 A、B 不同时，输出 Y 为 1；而当 A、B 相同时，输出 Y 为 0。 逻辑变量 A 和 B 进行异或运算时可写成：$Y = A \oplus B = (A \cdot B') + (A' \cdot B)$

真值表		
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

异或图形符号： 

八：同或运算及同或图形符号

同或运算：当 A、B 不同时，输出 Y 为 0；而当 A、B 相同时，输出 Y 为 1。 逻辑变量 A 和 B 进行同

或运算时可写成：Y = A⊙B = (A • B)+(A′ • B′)

真值表		
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

同或图形符号： 

STC MCU

1.3 STC8C 单片机性能概述

STC8C 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，STC8C 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，STC8C 系列单片机仅需 147 个时钟，而传统 8051 则需要 1944 个时钟。STC8C 系列单片机是 STC 生产的单时钟/机器周期(1T)的单片机，是宽电压/高速/高可靠/低功耗/强抗静电/较强抗干扰的新一代 8051 单片机，超级加密。指令代码完全兼容传统 8051。

MCU 内部集成高精度 R/C 时钟(±0.3%，常温下+25℃)，−1.38%～+1.42%温飘(-40℃~+85℃)，−0.88%～+1.05%温飘(-20℃~+65℃)。ISP 编程时 4MHz~35MHz 宽范围可设置（注意：温度范围为-40℃~+85℃时，最高频率须控制在 35MHz 以下），可彻底省掉外部昂贵的晶振和外部复位电路(内部已集成高可靠复位电路，ISP 编程时 4 级复位门槛电压可选)。

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

MCU 提供两种低功耗模式：IDLE 模式和 STOP 模式。IDLE 模式下，MCU 停止给 CPU 提供时钟，CPU 无时钟，CPU 停止执行指令，但所有的外设仍处于工作状态，此时功耗约为 1.0mA（6MHz 工作频率）。STOP 模式即为主时钟停振模式，即传统的掉电模式/停电模式/停机模式，此时 CPU 和全部外设都停止工作，功耗可降低到 0.6uA@Vcc=5.0V，0.4uA@Vcc=3.3V。

掉电模式可以使用 INT0(P3.2)、INT1(P3.3)、INT2(P3.6)、INT3(P3.7)、INT4(P3.0)、T0(P3.4)、T1(P3.5)、T2(P1.2)、T3(P0.4)、T4(P0.6)、RXD(P3.0/P3.6/P1.6/P4.3)、RXD2(P1.0/P4.0)、RXD3(P0.0)、RXD4(P0.2)、I2C_SDA(P1.4/P2.4/P3.3)、SPI_SS(P1.2/P2.2/P3.5)以及所有端口的 I/O 中断、比较器中断、低压检测中断、掉电唤醒定时器唤醒。

MCU 提供了丰富的数字外设（串口、定时器以及 I^2C、SPI）接口与模拟外设（比较器），可满足广大用户的设计需求。

STC8C 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。

1.4 STC8C 单片机产品线

产品线	I/O	UART	定时器	ADC	增强型 PWM	PCA	CMP	SPI	I2C	MDU16	I/O 中断
STC8C2K64S4 系列-44Pin/32Pin	42	4	5				●	●	●	●	●
STC8C2K64S2 系列-44Pin/32Pin/20Pin	42	2	5				●	●	●	●	●

2　STC8C 系列各子系列选型简介、特性、价格、管脚图

2.1　STC8C2K64S4-36I-LQFP44/32,QFN32 系列

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

➢ 选型价格（不需要外部晶振、不需要外部复位）

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	idata，内部传统 8051 RAM 字节	xdata，内部大容量扩展 SRAM 字节	强大的双 DPTR 可增可减	EEPROM 10 万次 字节	I/O 口最多数量	串口并可掉电唤醒	MDU16 硬件 16 位乘除法器	SPI	I^2C	定时器/计数器（T0-T4 外部管脚也可掉电唤醒）	15 位增强型 PWM 满足舞台灯光要求	PCA/CCP/PWM（可当外部中断并可掉电唤醒）	掉电唤醒专用定时器	15 路高速 ADC（全部 PWM 均可当 D/A 使用）	比较器（可当 1 路 A/D，可作外部掉电检测）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（36MHz 以下可调）追频	可对外输出时钟及复位	程序加密后传输（防拦截）	可设置下次更新程序需口令	支持 RS485 下载	支持软件 USB 直接下载	本身就可在线仿真	价格及封装			主力产品供货信息
																												LQFP44	LQFP32	QFN32<4*4mm>	
STC8C2K16S4	1.9-5.5	16K	256	2K	2	48K	42	4	有	有	有	5	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	¥3.7	¥3.5	¥3.5	现货
STC8C2K32S4	1.9-5.5	32K	256	2K	2	32K	42	4	有	有	有	5	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	¥3.7	¥3.5	¥3.5	
STC8C2K60S4	1.9-5.5	60K	256	2K	2	4K	42	4	有	有	有	5	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	¥3.7	¥3.5	¥3.5	
STC8C2K64S4	1.9-5.5	64K	256	2K	2	IAP	42	4	有	有	有	5	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	¥3.7	¥3.5	¥3.5	

注：以上的单价为 10K 及以上订货量的价格，量小则每片需增加 0.1 元人民币。



扫码去微信小商城

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

➢ **工作电压**
- ✓ 1.9V～5.5V
- ✓ 内建 LDO

➢ **工作温度**
- ✓ -40℃～85℃（超温度范围应用请参考电器特性章节说明）

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

➢ **SRAM**
- ✓ 128 字节内部直接访问 RAM（DATA）
- ✓ 128 字节内部间接访问 RAM（IDATA）
- ✓ 2048 字节内部扩展 RAM（内部 XDATA）

- **时钟控制**
  - ✓ 内部高精度、高稳定的高速 IRC（4MHz～38MHz，ISP 编程时可进行上下调整，还可以用户软件分频到较低的频率工作，如 100KHz）
    - ⌖ 误差±0.3%（常温下 25℃）
    - ⌖ -1.38%～+1.42%温漂（全温度范围，-40℃～85℃）
    - ⌖ -0.88%～+1.05%温漂（温度范围，-20℃～65℃）
  - ✓ 内部 32KHz 低速 IRC（为了低功耗，省去了温度补偿和电压补偿电路，误差较大）
  - ✓ 外部晶振（4MHz～38MHz）和外部时钟

（芯片上电工作过程：上电复位/复位脚复位/看门狗复位/低压检测复位时，芯片默认从 ISP 系统程序开始执行代码，此时固定使用内部 24MHz 的高速 IRC 时钟，当需要下载用户程序且下载完成后复位到用户程序区或者不需要下载直接复位到用户程序区时，默认会使用上次用户下载时所调节的高速 IRC 时钟，如果用户程序需要使用外部高速晶振、外部 32.768KHz 晶振或者内部 30KHz 低速 IRC，则需要用户软件先启动相应的时钟，然后通过设置 CLKSEL 寄存器进行切换）

- **复位**
  - ✓ 硬件复位
    - ⌖ 上电复位，实测电压值为 1.69V～1.82V。（**在芯片未使能低压复位功能时有效**）
      上电复位电压由一个上限电压和一个下限电压组成的电压范围，当工作电压从 5V/3.3V 向下掉到上电复位的下限门槛电压时，芯片处于复位状态；当电压从 0V 上升到上电复位的上限门槛电压时，芯片解除复位状态。
    - ⌖ 复位脚复位，出厂时 P5.4 默认为 I/O 口，ISP 下载时可将 P5.4 管脚设置为复位脚（**注意：当设置 P5.4 管脚为复位脚时，复位电平为低电平**）
    - ⌖ 看门狗溢出复位
    - ⌖ 低压检测复位，提供 4 级低压检测电压：2.0V（实测为 1.90V～2.04V）、2.4V（实测为 2.30V～2.50V）、2.7V（实测为 2.61V～2.82V）、3.0V（实测为 2.90V～3.13V）。
      每级低压检测电压都是由一个上限电压和一个下限电压组成的电压范围，当工作电压从 5V/3.3V 向下掉到低压检测的下限门槛电压时，低压检测生效；当电压从 0V 上升到低压检测的上限门槛电压时，低压检测生效。
  - ✓ 软件复位
    - ⌖ 软件方式写复位触发寄存器

- **中断**
  - ✓ 提供 18 个中断源：INT0（支持上升沿和下降沿中断）、INT1（支持上升沿和下降沿中断）、INT2（只支持下降沿中断）、INT3（只支持下降沿中断）、INT4（只支持下降沿中断）、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、串口 1、串口 2、串口 3、串口 4、LVD 低压检测、SPI、I2C、比较器。
  - ✓ 提供 4 级中断优先级
  - ✓ 时钟停振模式下可以唤醒的中断：INT0(P3.2)、INT1(P3.3)、INT2(P3.6)、INT3(P3.7)、INT4(P3.0)、T0(P3.4)、T1(P3.5)、T2(P1.2)、T3(P0.4)、T4(P0.6)、RXD(P3.0/P3.6/P1.6/P4.3)、RXD2(P1.0/P4.0)、RXD3(P0.0)、RXD4(P0.2)、I2C_SDA(P1.4/P2.4/P3.3)、SPI_SS(P1.2/P2.2/P3.5)以及所有端口的 I/O 中断、比较器中断、低压检测中断、掉电唤醒定时器唤醒

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

- ✓ **MDU16：**硬件 16 位乘除法器（支持 32 位除以 16 位、16 位除以 16 位、16 位乘 16 位、数据移位以及数据规格化等运算）
- ✓ **I/O 口中断：**所有的 I/O 均支持中断，每组 I/O 中断有独立的中断入口地址，所有的 I/O 中断可支持 4 种中断模式：高电平中断、低电平中断、上升沿中断、下降沿中断
  （注意：STC8C2K64S4-44Pin/32Pin 系列的 I/O 口中断可以进行掉电唤醒，但 I/O 口中断只有一级中断优先级）

➢ **模拟外设**

- ✓ 比较器，一组比较器（比较器的正端可选择 CMP+端口和所有的 ADC 输入端口，所以比较器可当作多路比较器进行分时复用）

➢ **GPIO**

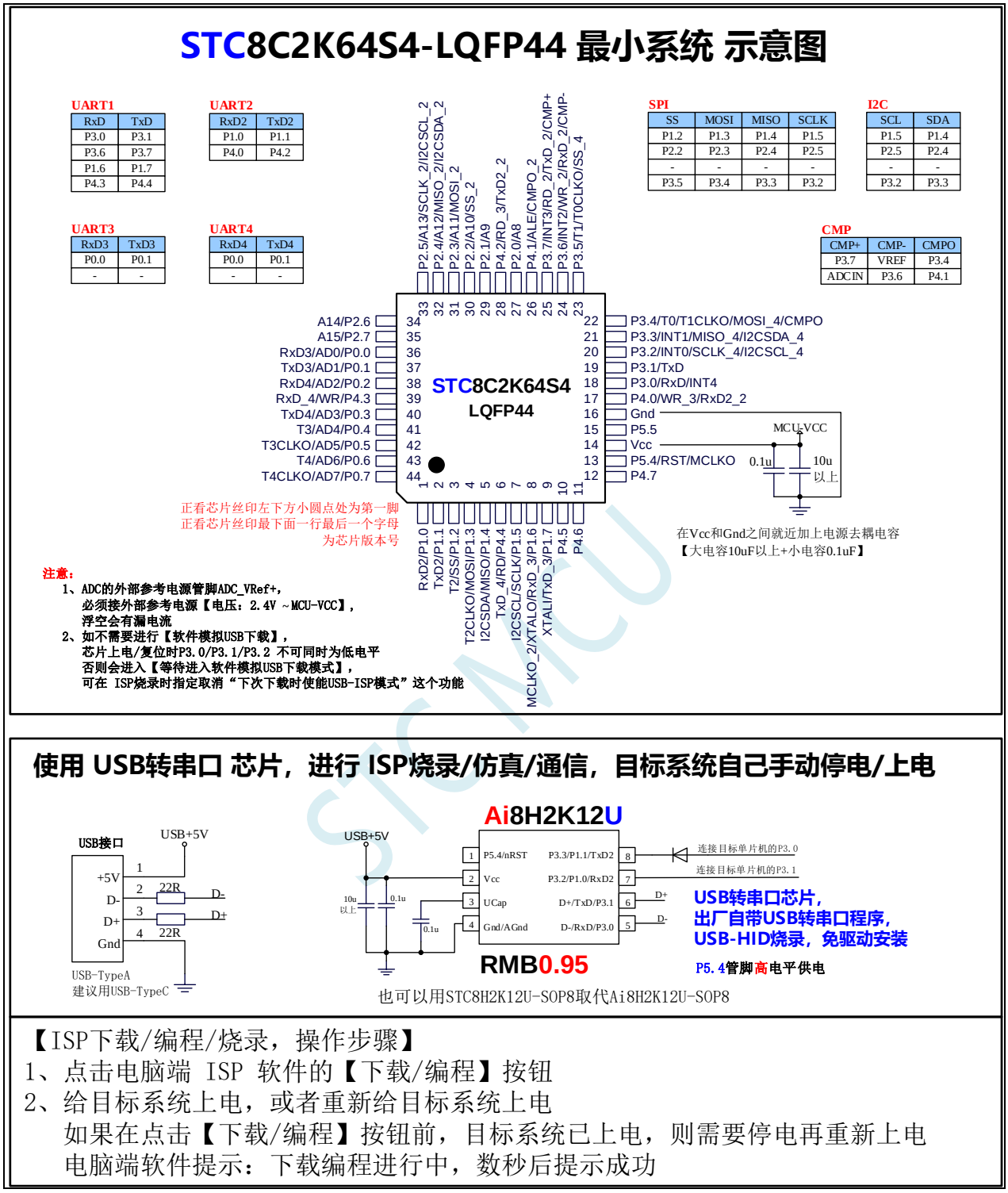
- ✓ 最多可达 45 个 GPIO：P0.0~P0.7、P1.0~P1.7、P2.0~P2.7、P3.0~P3.7、P4.0~P4.7、P5.0~P5.4
- ✓ 所有的 GPIO 均支持如下 4 种模式：准双向口模式、强推挽输出模式、开漏模式、高阻输入模式
- ✓ **除 P3.0 和 P3.1 外，其余所有 I/O 口上电后的状态均为高阻输入状态，用户在使用 I/O 口时必须先设置 I/O 口模式，另外每个 I/O 均可独立使能内部 4K 上拉电阻**

➢ **封装**

- ✓ LQFP44、LQFP32、QFN32<4mm*4mm>

STC MCU

2.1.2　STC8C2K64S4-36I-LQFP44 管脚图，ISP 下载/编程线路图



【ISP下载/编程/烧录，操作步骤】

1、点击电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新给目标系统上电

如果在点击【下载/编程】按钮前，目标系统已上电，则需要停电再重新上电

电脑端软件提示：下载编程进行中，数秒后提示成功

关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式

2、 除 P3.0 和 P3.1 外，其余所有 IO 口上电后的状态均为高阻输入状态，用户在使用 IO 口前必须先设置 IO 口模式

3、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能

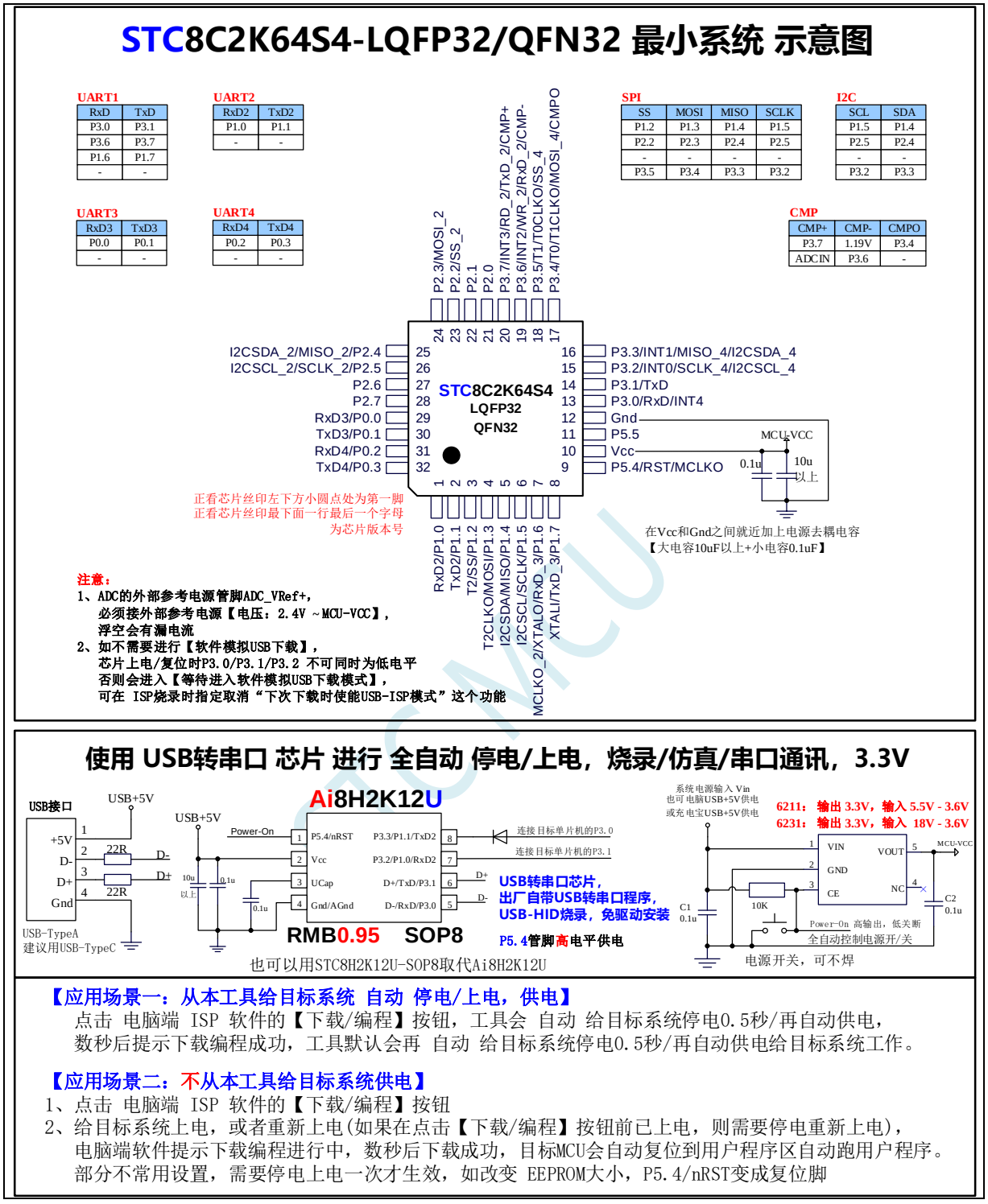
同时为低电平，否则会进入 USB 下载模式而无法运行用户代码

4、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间由高阻输入状态切换到双向口模式，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式

5、 当使用 P5.4 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P5.4 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P5.4 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

STC MCU

2.1.3　STC8C2K64S4-36I-LQFP32/QFN32 管脚图，ISP 下载/编程线路图



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。

部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式

2、 除 P3.0 和 P3.1 外，其余所有 IO 口上电后的状态均为高阻输入状态，用户在使用 IO 口前必须先设置 IO 口模式

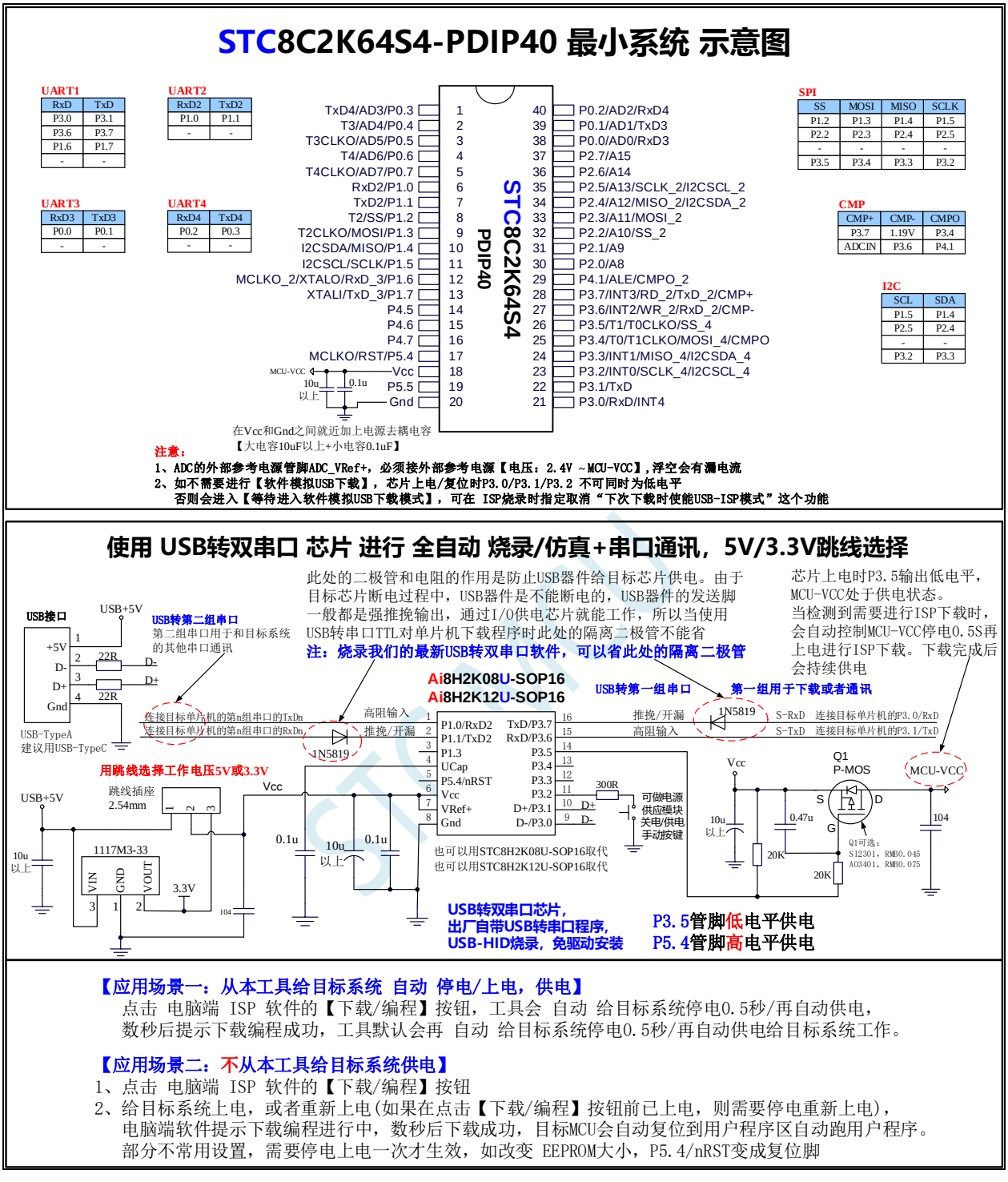
3、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码

4、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间由高阻输入状态切换到双向口模式，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式

5、 当使用 P5.4 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P5.4 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P5.4 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

STC MCU

2.1.4　STC8C2K64S4-36I-PDIP40 管脚图，ISP 下载/编程线路图



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式

2、 除 P3.0 和 P3.1 外，其余所有 IO 口上电后的状态均为高阻输入状态，用户在使用 IO 口前必须先设置 IO 口模式

3、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码

4、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间由高阻输入状态切换到双向口模式，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式

5、 当使用 P5.4 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P5.4 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P5.4 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

STC MCU

2.1.5 管脚说明

编号				名称	类型	说明
LQFP44	PDIP40	LQFP32 QFN32				
2	7	2		P1.1	I/O	标准 IO 口
				TxD2	O	串口 2 的发送脚
3	8	3		P1.2	I/O	标准 IO 口
				SS	I	SPI 的从机选择脚（主机为输出）
				T2	I	定时器 2 外部时钟输入
4	9	4		P1.3	I/O	标准 IO 口
				MOSI	I/O	SPI 主机输出从机输入
				T2CLKO	O	定时器 2 时钟分频输出
5	10	5		P1.4	I/O	标准 IO 口
				MISO	I/O	SPI 主机输入从机输出
				SDA	I/O	I2C 接口的数据线
6				P4.4	I/O	标准 IO 口
				RD	O	外部总线的读信号线
				TxD_4	O	串口 1 的发送脚
7	11	6		P1.5	I/O	标准 IO 口
				SCLK	I/O	SPI 的时钟脚
				SCL	I/O	I2C 的时钟线
8	12	7		P1.6	I/O	标准 IO 口
				RxD_3	I	串口 1 的接收脚
				XTALO	O	外部晶振的输出脚
				MCLKO_2	O	主时钟分频输出
9	13	8		P1.7	I/O	标准 IO 口
				TxD_3	O	串口 1 的发送脚
				XTALI	I	外部晶振/外部时钟的输入脚
10	14			P4.5	I/O	标准 IO 口
11	15			P4.6	I/O	标准 IO 口
12	16			P4.7	I/O	标准 IO 口
13	17	9		P5.4	I/O	标准 IO 口
				RST	I	复位引脚
				MCLKO	O	主时钟分频输出
14	18	10		Vcc	VCC	电源脚

编号				名称	类型	说明
LQFP44	PDIP40	LQFP32 QFN32				
15	19	11		P5.5	I/O	标准 IO 口
16	20	12		Gnd	GND	地线
17				P4.0	I/O	标准 IO 口
				WR_3	O	外部总线的写信号线
				RxD2_2	I	串口 2 的接收脚
18	21	13		P3.0	I/O	标准 IO 口
				RxD	I	串口 1 的接收脚
				INT4	I	外部中断 4
19	22	14		P3.1	I/O	标准 IO 口
				TxD	O	串口 1 的发送脚
20	23	15		P3.2	I/O	标准 IO 口
				INT0	I	外部中断 0
				SCL_4	I/O	I2C 的时钟线
				SCLK_4	I/O	SPI 的时钟脚
21	24	16		P3.3	I/O	标准 IO 口
				INT1	I	外部中断 1
				SDA_4	I/O	I2C 接口的数据线
				MISO_4	I/O	SPI 主机输入从机输出
22	25	17		P3.4	I/O	标准 IO 口
				T0	I	定时器 0 外部时钟输入
				T1CLKO	O	定时器 1 时钟分频输出
				MOSI_4	I/O	SPI 主机输出从机输入
				CMPO	O	比较器输出
23	26	18		P3.5	I/O	标准 IO 口
				T1	I	定时器 1 外部时钟输入
				T0CLKO	O	定时器 0 时钟分频输出
				SS_4	I	SPI 的从机选择脚（主机为输出）
24	27	19		P3.6	I/O	标准 IO 口
				INT2	I	外部中断 2
				WR_2	O	外部总线的写信号线
				RxD_2	I	串口 1 的接收脚
				CMP-	I	比较器负极输入

编号				名称	类型	说明
LQFP44	PDIP40	LQFP32 QFN32				
25	28	20		P3.7	I/O	标准 IO 口
				INT3	I	外部中断 3
				RD_2	O	外部总线的读信号线
				TxD_2	O	串口 1 的发送脚
				CMP+	I	比较器正极输入
26	29			P4.1	I/O	标准 IO 口
				ALE	O	地址锁存信号
				CMPO_2	O	比较器输出
27	30	21		P2.0	I/O	标准 IO 口
				A8	O	地址总线
				RSTSV	-	端口的初始电平可 ISP 下载时配置
28				P4.2	I/O	标准 IO 口
				RD_3	O	外部总线的读信号线
				TxD2_2	O	串口 2 的发送脚
29	31	22		P2.1	I/O	标准 IO 口
				A9	O	地址总线
30	32	23		P2.2	I/O	标准 IO 口
				A10	O	地址总线
				SS_2	I	SPI 的从机选择脚（主机为输出）
31	33	24		P2.3	I/O	标准 IO 口
				A11	O	地址总线
				MOSI_2	I/O	SPI 主机输出从机输入
32	34	25		P2.4	I/O	标准 IO 口
				A12	O	地址总线
				MISO_2	I/O	SPI 主机输入从机输出
				SDA_2	I/O	I2C 接口的数据线
33	35	26		P2.5	I/O	标准 IO 口
				A13	O	地址总线
				SCLK_2	I/O	SPI 的时钟脚
				SCL_2	I/O	I2C 的时钟线
34	36	27		P2.6	I/O	标准 IO 口
				A14	O	地址总线
35	37	28		P2.7	I/O	标准 IO 口
				A15	O	地址总线

编号				名称	类型	说明
LQFP44	PDIP40	LQFP32 QFN32				
36	38	29		P0.0	I/O	标准 IO 口
				AD0	I/O	地址/数据总线
				RxD3	I	串口 3 的接收脚
37	39	30		P0.1	I/O	标准 IO 口
				AD1	I/O	地址/数据总线
				TxD3	O	串口 3 的发送脚
38	40	31		P0.2	I/O	标准 IO 口
				AD2	I/O	地址/数据总线
				RxD4	I	串口 4 的接收脚
39				P4.3	I/O	标准 IO 口
				WR	O	外部总线的写信号线
				RxD_4	I	串口 1 的接收脚
40	1	32		P0.3	I/O	标准 IO 口
				AD3	I/O	地址/数据总线
				TxD4	O	串口 4 的发送脚
41	2			P0.4	I/O	标准 IO 口
				AD4	I/O	地址/数据总线
				T3	I	定时器 3 外部时钟输入
42	3			P0.5	I/O	标准 IO 口
				AD5	I/O	地址/数据总线
				T3CLKO	O	定时器 3 时钟分频输出
43	4			P0.6	I/O	标准 IO 口
				AD6	I/O	地址/数据总线
				T4	I	定时器 4 外部时钟输入
44	5			P0.7	I/O	标准 IO 口
				AD7	I/O	地址/数据总线
				T4CLKO	O	定时器 4 时钟分频输出
1	6	1		P1.0	I/O	标准 IO 口
				RxD2	I	串口 2 的接收脚

2.1.6 USB-Link1D 对 STC8C 系列进行自动串口烧录、仿真+串口通讯

使用【USB Link1D】对 STC8C2K64S4系列 进行全自动停电/上电串口烧录、仿真+串口通讯

【USB Link1D】工具：支持 全自动 停电 / 上电，在线下载 / 脱机下载，仿真



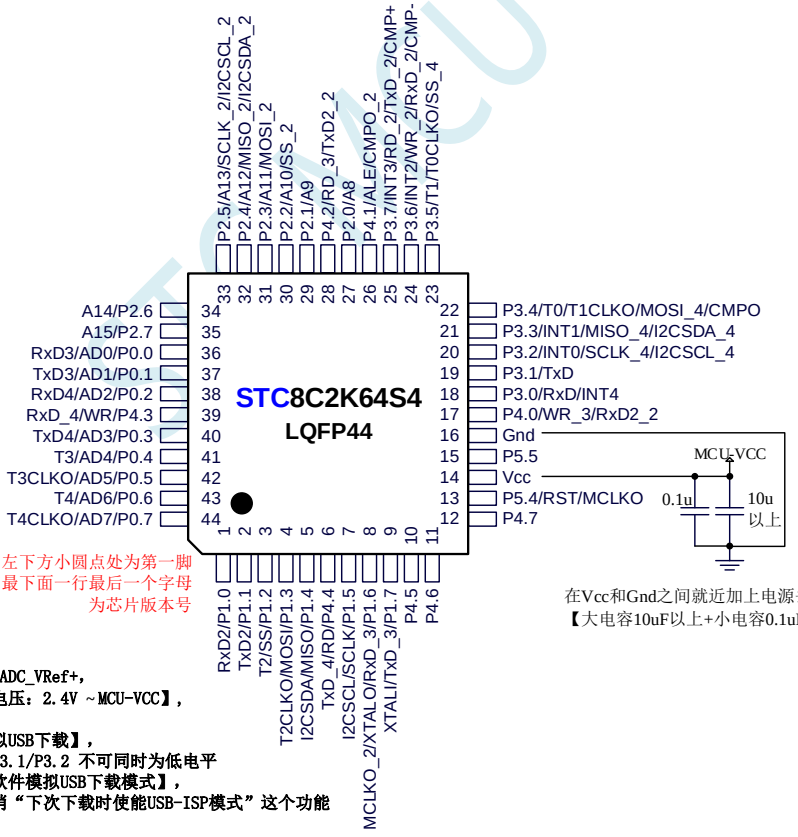
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

STC8C2K64S4-LQFP44 最小系统 示意图



正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号

在Vcc和Gnd之间就近加上电源去耦电容
【大电容10uF以上+小电容0.1uF】

注意：

1、ADC的外部参考电源管脚ADC_VRef+，必须接外部参考电源【电压：2.4V ~MCU-VCC】，浮空会有漏电流
2、如不需要进行【软件模拟USB下载】，芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平否则会进入【等待进入软件模拟USB下载模式】，可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

2.1.7　使用【一箭双雕之 USB 转双串口】进行串口烧录、仿真+串口通讯

使用【一箭双雕之USB转双串口】对 STC8C2K64S4系列 进行串口烧录、仿真+串口通讯

【一箭双雕之USB转双串口】：支持 全自动 停电 / 上电，在线下载，仿真



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

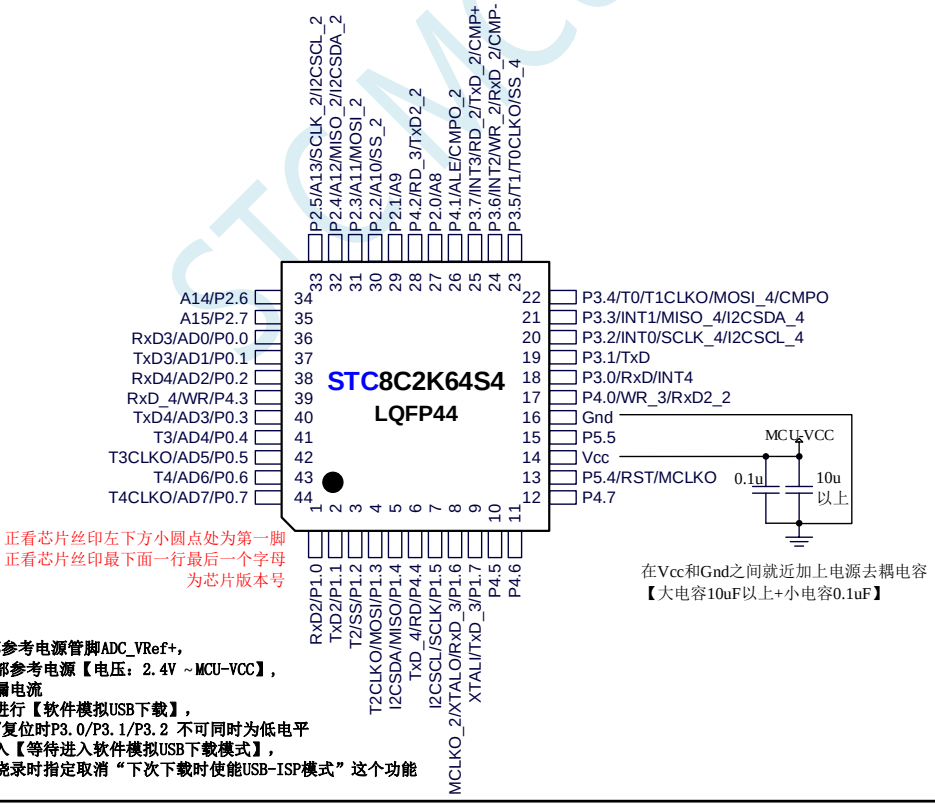
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区跑用户程序。部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

STC8C2K64S4-LQFP44 最小系统 示意图



正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号

在Vcc和Gnd之间就近加上电源去耦电容
【大电容10uF以上+小电容0.1uF】

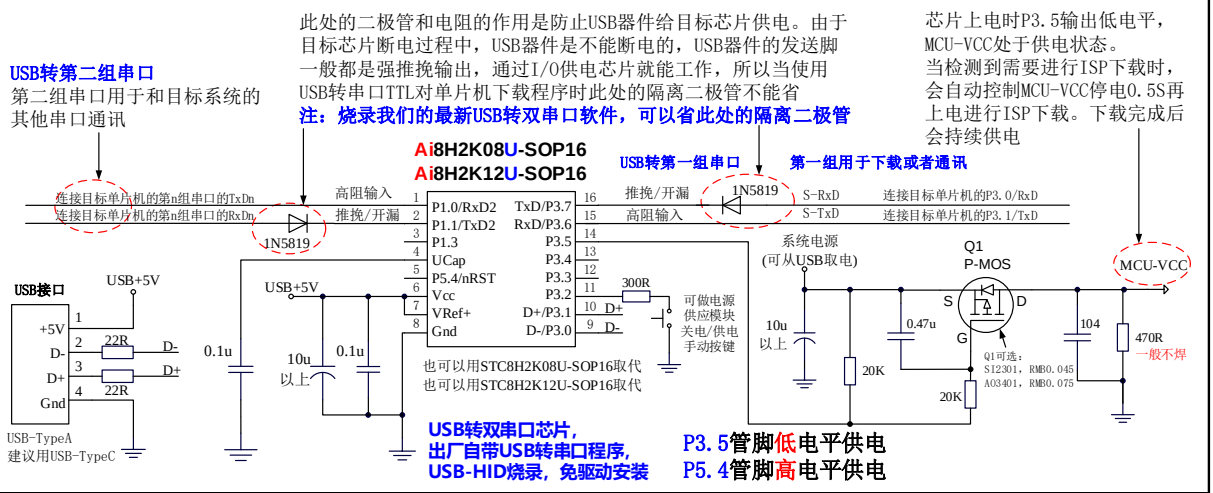
注意：

1、ADC的外部参考电源管脚ADC_VRef+，必须接外部参考电源【电压：2.4V ~MCU-VCC】，浮空会有漏电流

2、如不需要进行【软件模拟USB下载】，芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平否则会进入【等待进入软件模拟USB下载模式】，可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

2.1.8 USB 转双串口芯片全自动停电/上电烧录，串口仿真+串口通讯，5V

使用 USB转双串口 芯片 进行 全自动 停电/上电，烧录/仿真+串口通讯，5V



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

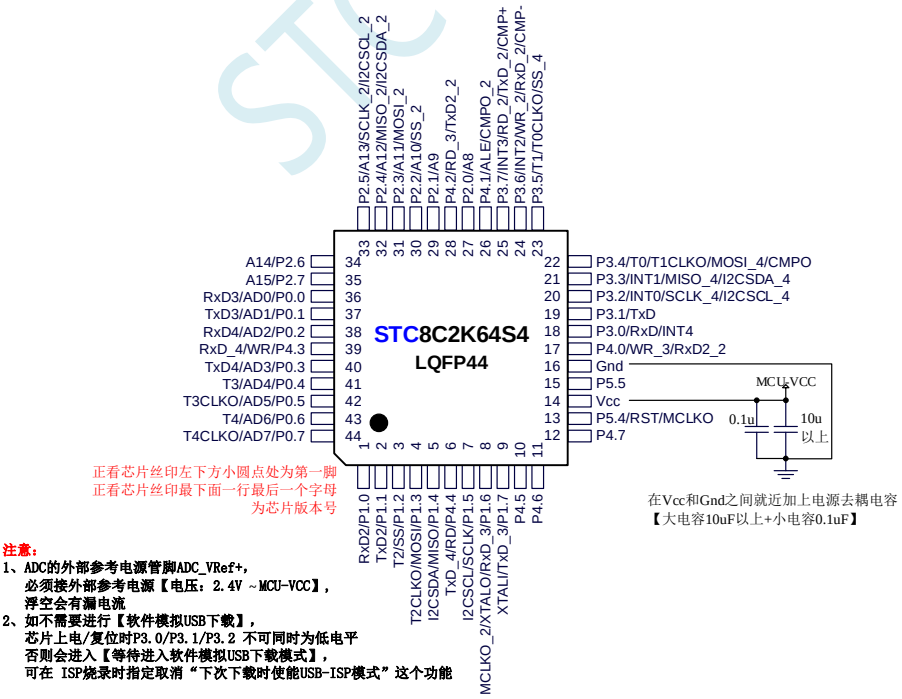
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。

部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

STC8C2K64S4-LQFP44 最小系统 示意图

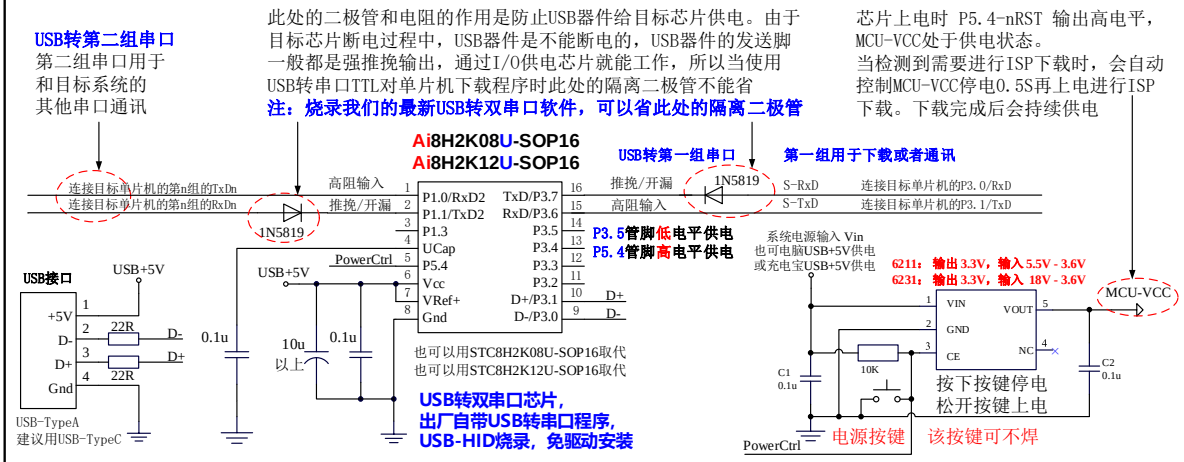


注意：

1、ADC的外部参考电源管脚ADC_VRef+，必须接外部参考电源【电压：2.4V ~MCU-VCC】，浮空会有漏电流

2、如不需要进行【软件模拟USB下载】，芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平否则会进入【等待进入软件模拟USB下载模式】，可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

2.1.9 USB 转双串口芯片全自动烧录，串口仿真+串口通讯，3.3V 原理图



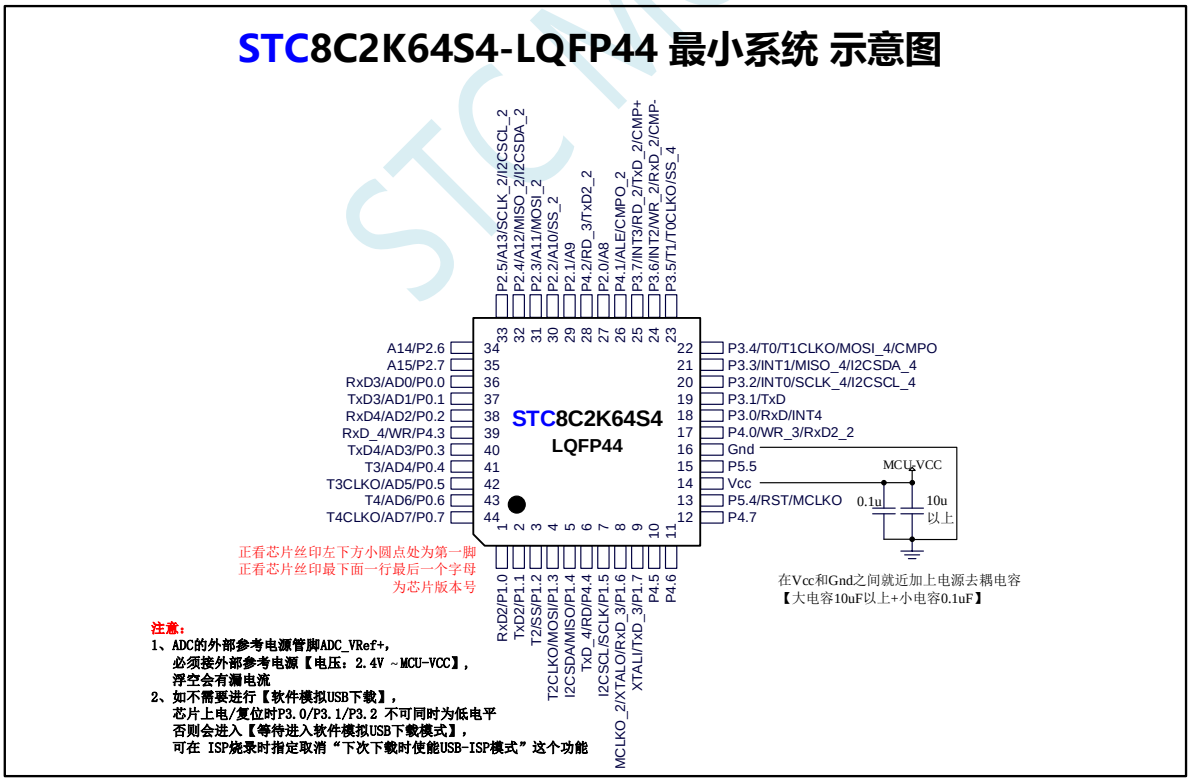
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，
数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

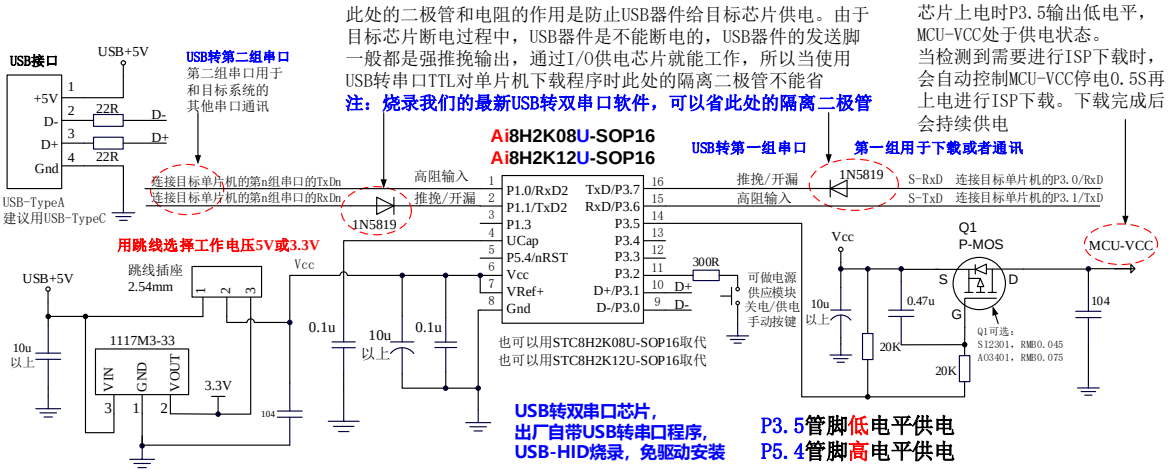
1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，
电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚



2.1.10 USB 转双串口芯片进行自动烧录/仿真+串口通讯，5V/3.3V 跳线选择

使用 USB转双串口 芯片 进行 全自动 烧录/仿真+串口通讯，5V/3.3V跳线选择



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

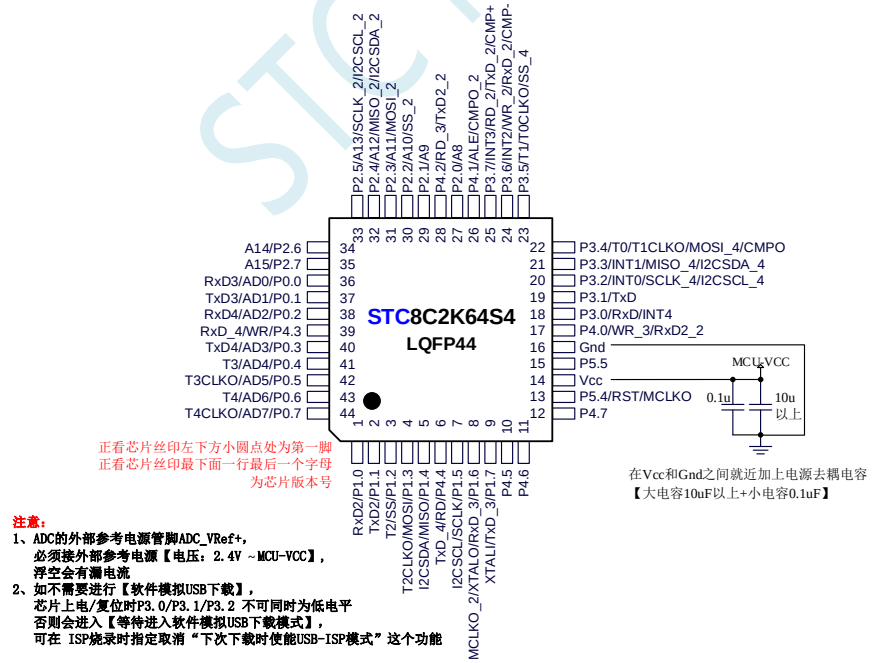
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

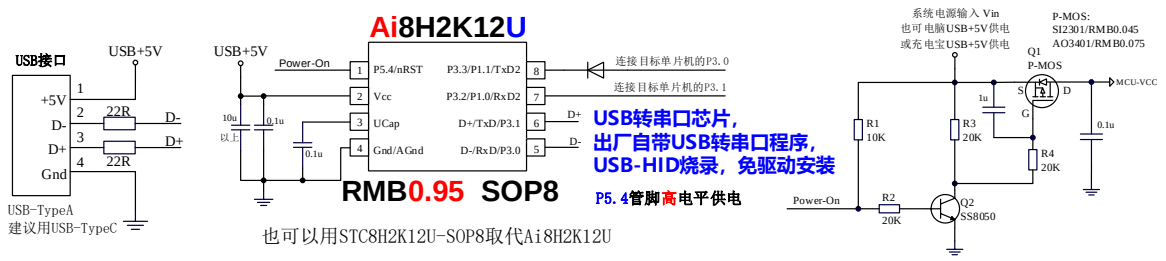
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

STC8C2K64S4-LQFP44 最小系统 示意图



2.1.11 通用 USB 转串口芯片全自动停电/上电烧录，串口仿真，5V 原理图

使用 USB转串口 芯片 进行 全自动 停电/上电，烧录/仿真/串口通讯，5V



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

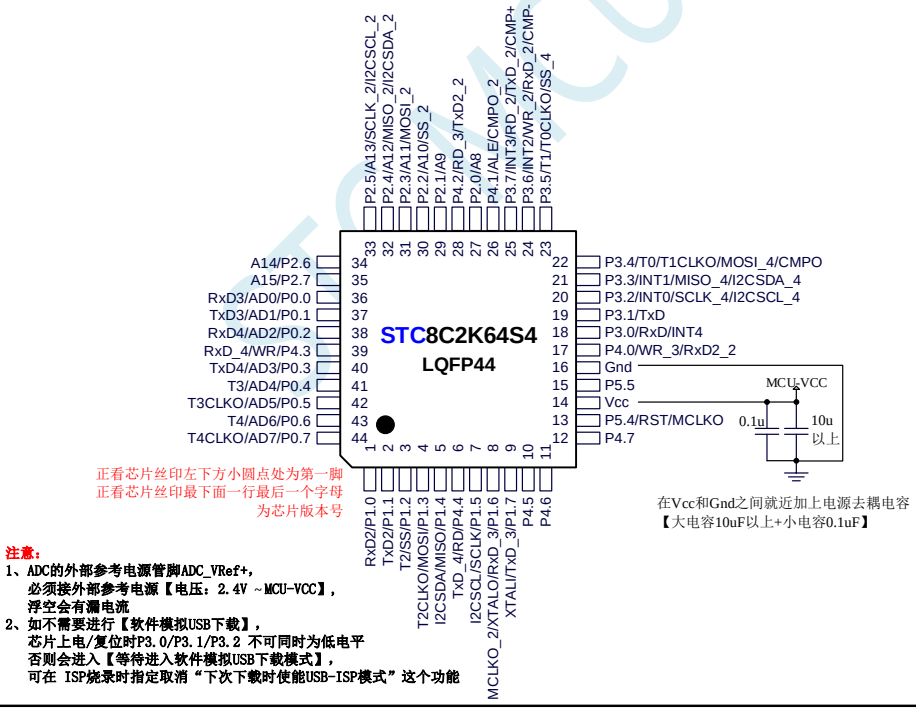
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。

部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

STC8C2K64S4-LQFP44 最小系统 示意图



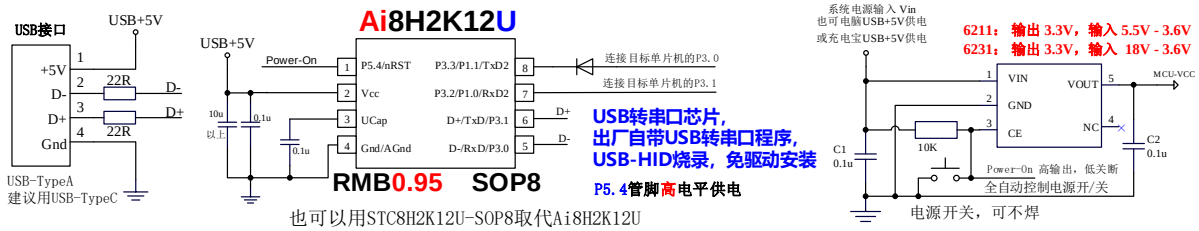
注意：

1、ADC的外部参考电源管脚ADC_VRef+，必须接外部参考电源【电压：2.4V ~MCU-VCC】，浮空会有漏电流

2、如不需要进行【软件模拟USB下载】，芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平否则会进入【等待进入软件模拟USB下载模式】，可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

2.1.12 通用 USB 转串口芯片全自动停电/上电烧录，串口仿真，3.3V 原理图

使用 USB转串口 芯片 进行 全自动 停电/上电，烧录/仿真/串口通讯，3.3V



也可以用STC8H2K12U-SOP8取代Ai8H2K12U

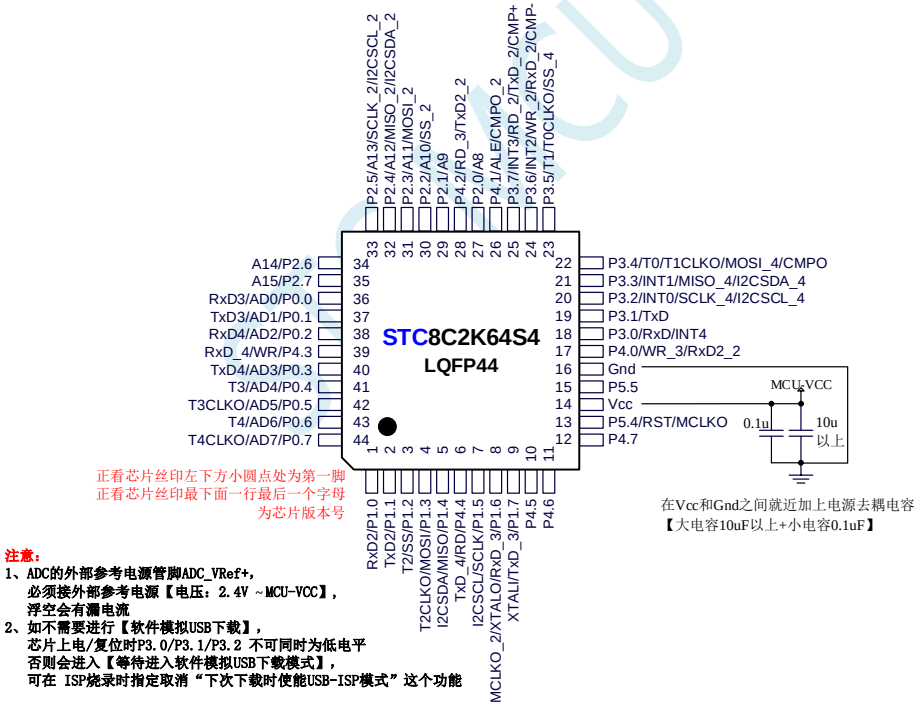
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，
数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，
电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

STC8C2K64S4-LQFP44 最小系统 示意图



正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母
为芯片版本号

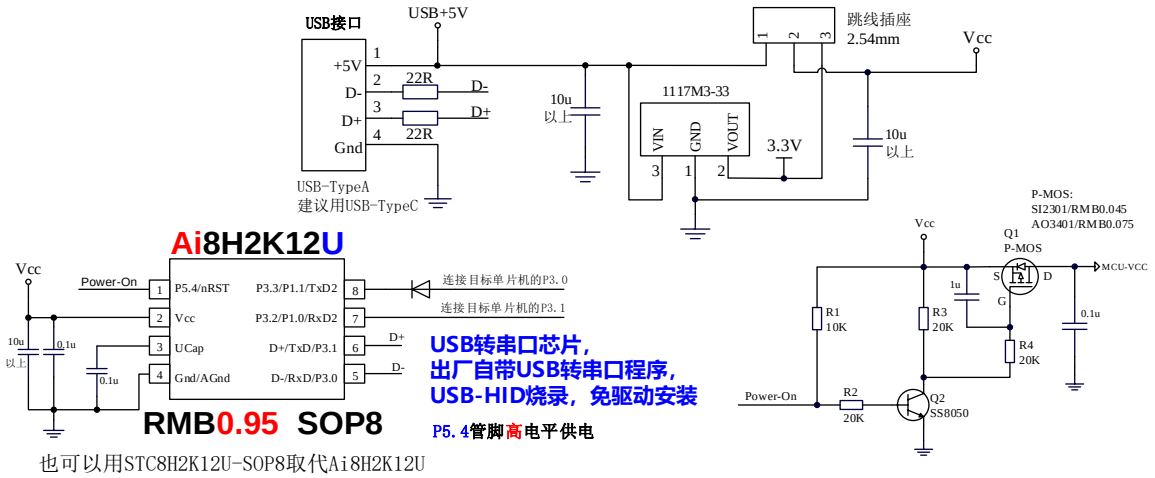
在Vcc和Gnd之间就近加上电源去耦电容
【大电容10uF以上+小电容0.1uF】

注意：
1、ADC的外部参考电源管脚ADC_VRef+，
必须接外部参考电源【电压：2.4V ~MCU-VCC】，
浮空会有漏电流
2、如不需要进行【软件模拟USB下载】，
芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平
否则会进入【等待进入软件模拟USB下载模式】，
可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

2.1.13　USB 转串口芯片全自动停电/上电烧录/仿真/通信，5V/3.3V 跳线选择

USB转串口 芯片 全自动 停电/上电，烧录/仿真/通信，5V/3.3V跳线选择

用跳线选择工作电压5V或3.3V



也可以用STC8H2K12U-SOP8取代Ai8H2K12U

【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

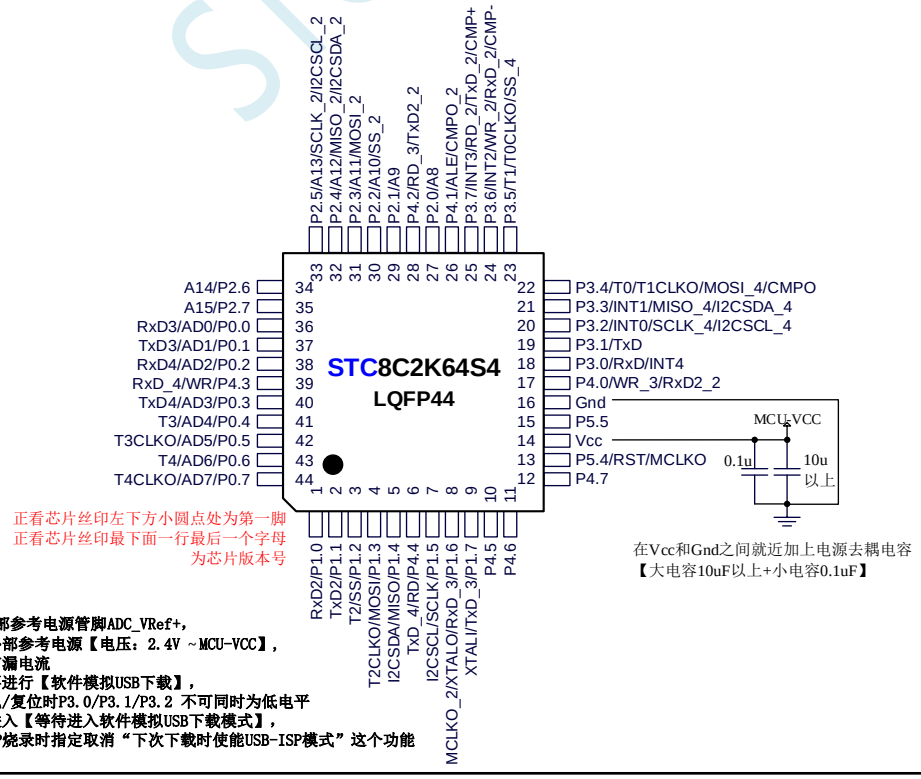
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

STC8C2K64S4-LQFP44 最小系统 示意图



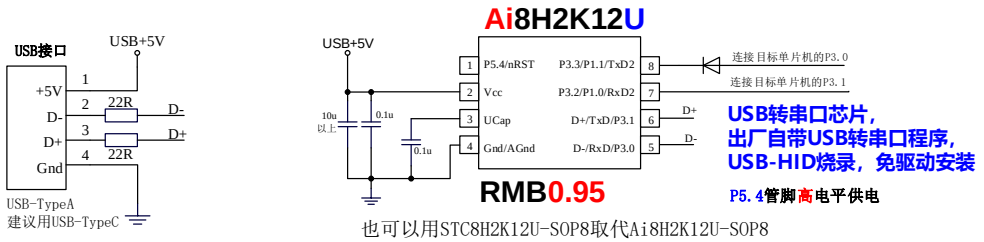
注意：

1、ADC的外部参考电源管脚ADC_VRef+，必须接外部参考电源【电压：2.4V ~MCU-VCC】，浮空会有漏电流

2、如不需要进行【软件模拟USB下载】，芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平否则会进入【等待进入软件模拟USB下载模式】，可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

2.1.14　USB 转串口芯片进行烧录/串口仿真，手动停电/上电，5V/3.3V 原理图

使用 USB转串口 芯片，进行 ISP烧录/仿真/通信，目标系统自己手动停电/上电



也可以用STC8H2K12U-SOP8取代Ai8H2K12U-SOP8

【ISP下载/编程/烧录，操作步骤】

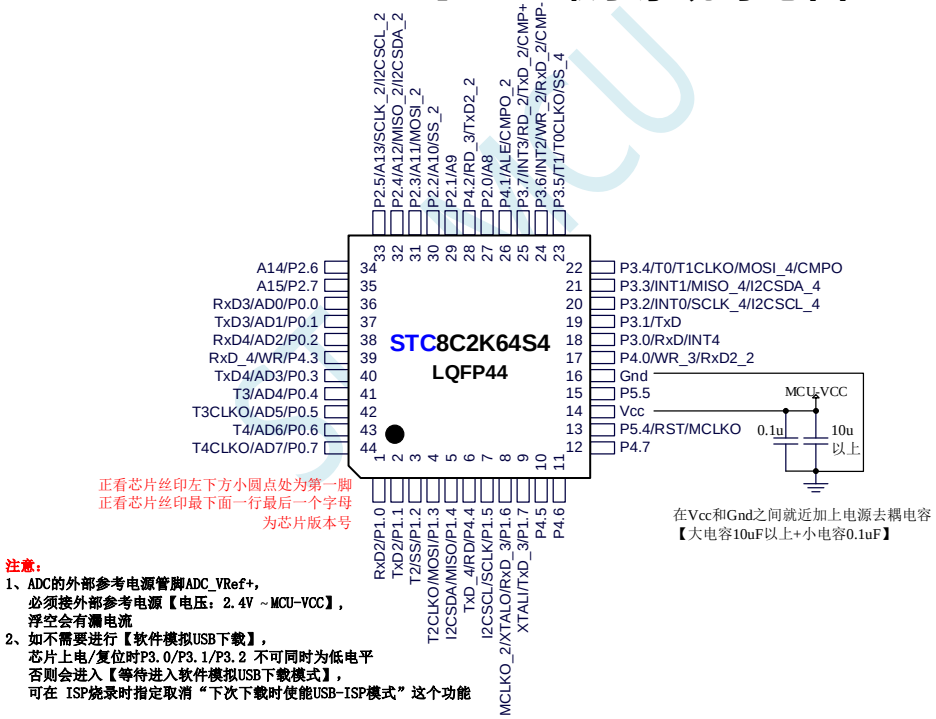
1、点击电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新给目标系统上电

如果在点击【下载/编程】按钮前，目标系统已上电，则需要停电再重新上电

电脑端软件提示：下载编程进行中，数秒后提示成功

STC8C2K64S4-LQFP44 最小系统 示意图



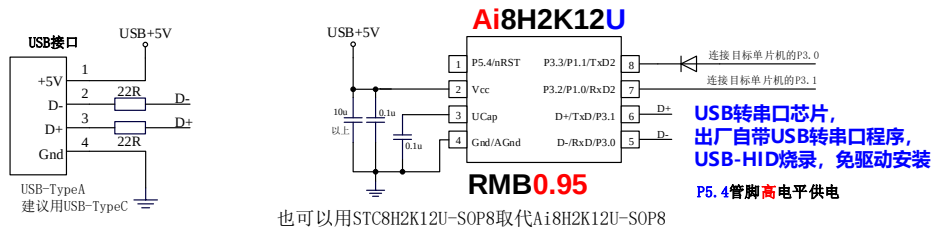
注意：

1、ADC的外部参考电源管脚ADC_VRef+，
必须接外部参考电源【电压：2.4V ~MCU-VCC】，
浮空会有漏电流

2、如不需要进行【软件模拟USB下载】，
芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平
否则会进入【等待进入软件模拟USB下载模式】，
可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

2.1.15 USB 转串口芯片进行烧录，串口仿真，手动停电/上电，3.3V 原理图

使用 USB转串口 芯片，进行 ISP烧录/仿真/通信，目标系统自己 手动 停电/上电



也可以用STC8H2K12U-SOP8取代Ai8H2K12U-SOP8

【ISP下载/编程/烧录，操作步骤】

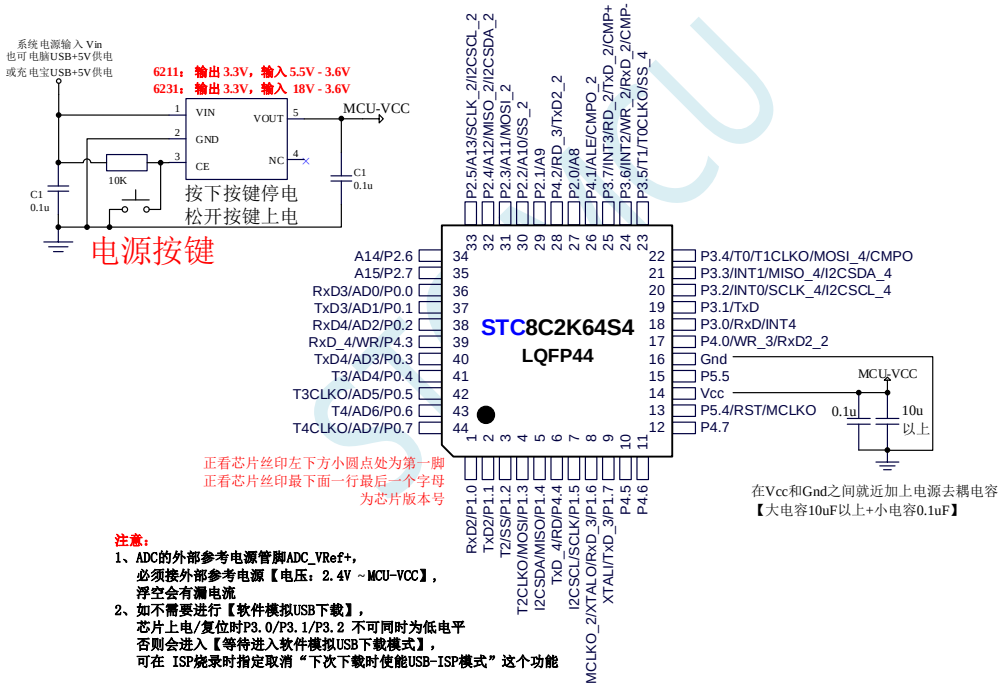
1、点击电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新给目标系统上电

如果在点击【下载/编程】按钮前，目标系统已上电，则需要停电再重新上电

电脑端软件提示：下载编程进行中，数秒后提示成功

STC8C2K64S4-LQFP44 最小系统 示意图



注意：

1、ADC的外部参考电源管脚ADC_VRef+，
必须接外部参考电源【电压：2.4V ~MCU-VCC】，
浮空会有漏电流

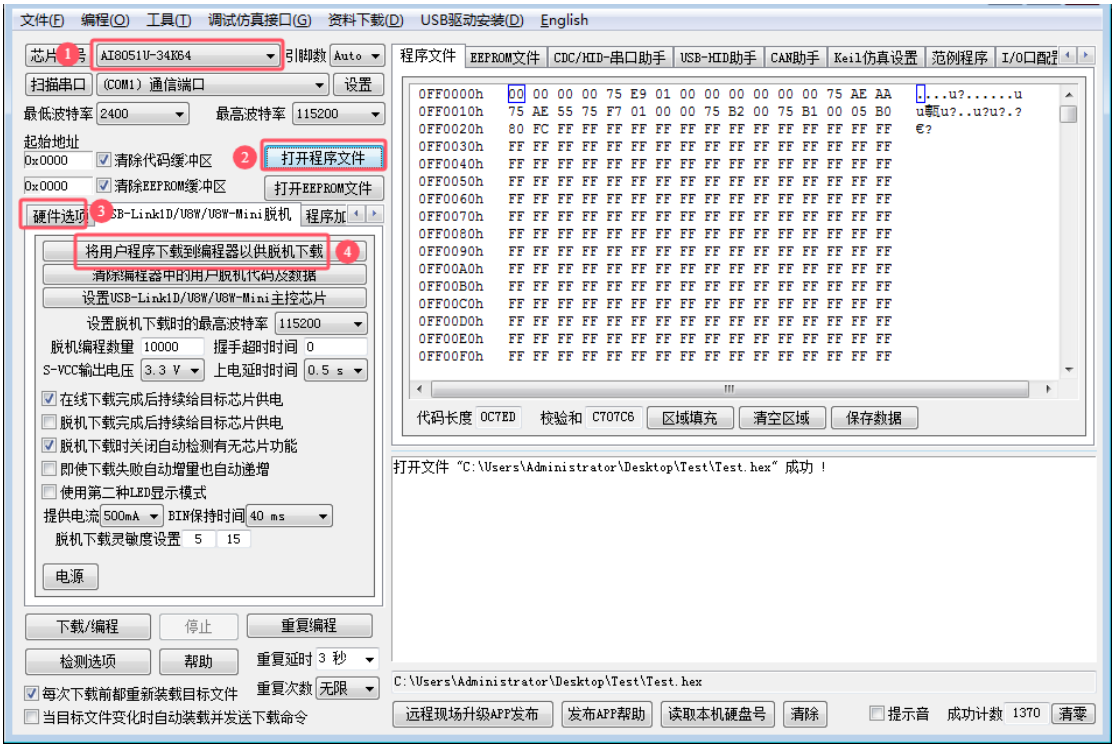
2、如不需要进行【软件模拟USB下载】，
芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平
否则会进入【等待进入软件模拟USB下载模式】，
可在 ISP烧录时指定取消 “下次下载时使能USB-ISP模式” 这个功能

2.1.16 USB-Link1D 支持 脱机下载 说明

脱机下载是指脱离电脑主机进行下载的一种方法，一般是使用下载控制芯片（又称脱机下载母片）进行控制。USB-Link1D 工具除了支持在线 ISP 下载，还支持脱机下载。USB-Link1D 工具使用外部的 22.1184MHz 晶振，可保证对目标芯片进行在线或脱机下载时，校准频率的精度。用户可将代码下载到 USB-Link1D 工具中，就可实现脱机下载。USB-Link1D 主控芯片如内部存储空间不够时，会将部分被下载的用户程序用加密的方式加密放部分到片外串行 Flash。

先将 USB-Link1D 工具使用 USB 线连接到电脑，然后按照下面的步骤进行脱机下载：

1、选择目标芯片的型号
2、打开需要下载的文件
3、设置硬件选项
4、进入“USB-Link1D 脱机”页面，点击“将用户程序下载到编程器以供脱机下载“按钮，即可将用户代码下载到 USB-Linik1D 工具中。下载完成后便使用 USB-Link1D 工具对目标芯片进行脱机下载了



2.1.17　USB-Link1D 支持 脱机下载，如何免烧录环节

大批量生产，如何省去专门的烧录人员，如何无烧录环节

大批量生产，你在将由 STC 的 MCU 作为主控芯片的控制板组装到设备里面之前，在你将 STC MCU 贴片到你的控制板完成之后，你必须测试你的控制板的好坏。不要说 100%，直通无问题，那是抬杠，不是搞生产，只要生产，就会虚焊，短路，部分元件贴错，部分元件采购错。

所以在贴片回来后，组装到外壳里面之前，你必须要测试，你的含有 STC MCU 控制板的好坏，好的去组装，坏的去维修抢救。

控制板的测试/不是烧录！

控制板的测试环节必须有，但烧录环节可以省！

大批量生产，必须要有方便控制板测试的测试架/下面接上我们的脱机烧录工具：

USB-Link1D / U8W-Mini / U8W，还要接上其他控制部分！

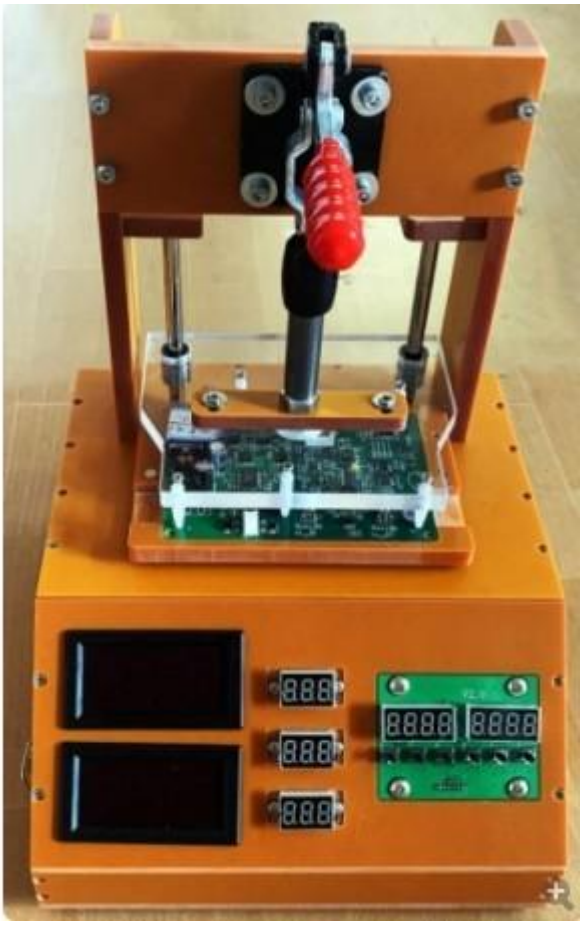
1、通过 USER-VCC、P3.0、P3.1、GND 连接，要工人每次都开电源

2、通过 S-VCC、P3.0、P3.1、GND 连接，不要你开电源，STC 的脱机工具给你自动供电

外面帮你做一个测试架的成本 500 元以下，就是有机玻璃，夹具，顶针。

1 个测试你控制板是否正常的工人管理 2 - 3 个 测试架



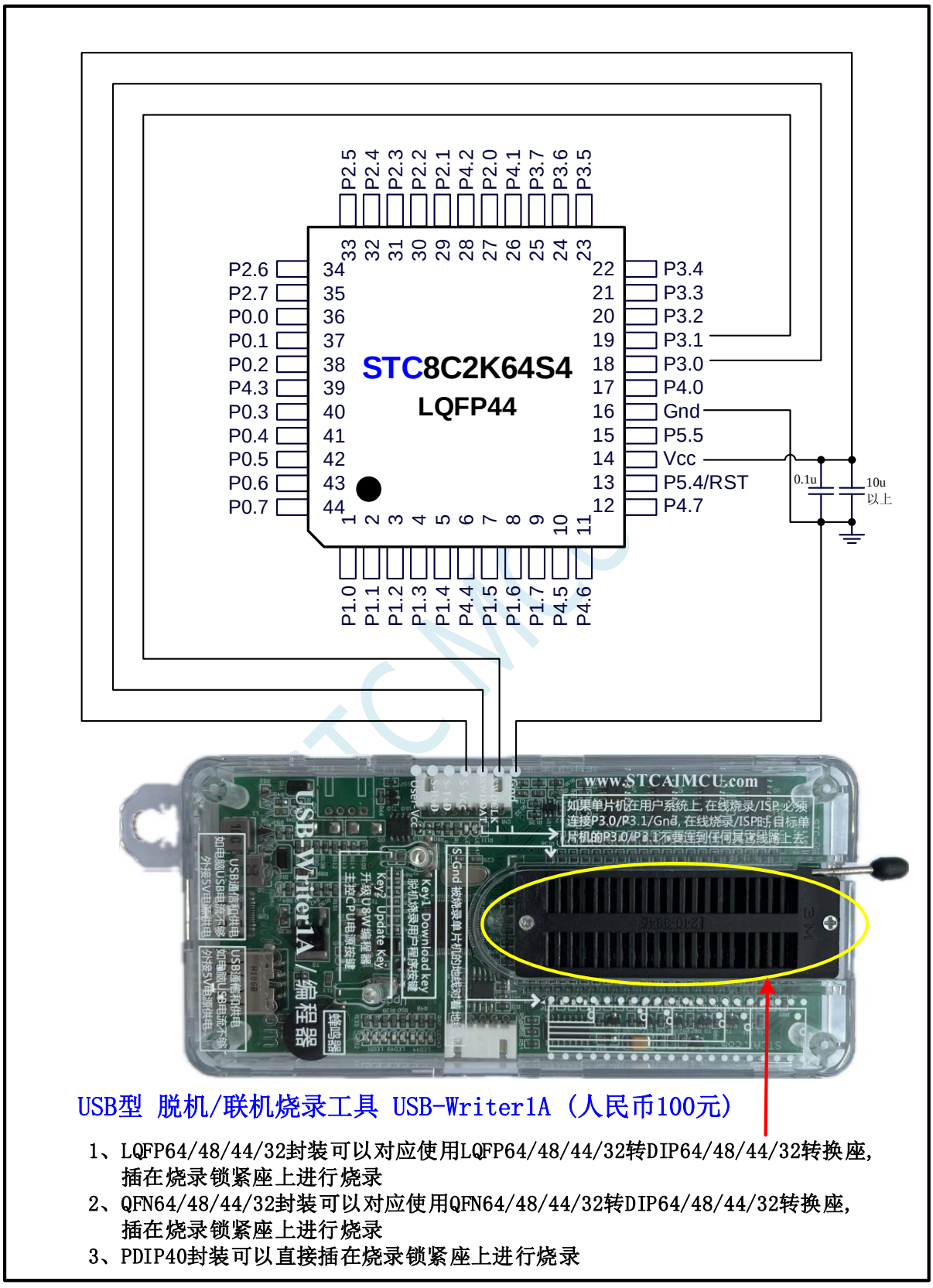


操作流程：

1、将你的 MCU 控制板 卡到测试架 1 上
2、将你的 MCU 控制板 卡到测试架 2 上，测试架 1 上的程序已烧录完成/感觉不到烧录时间
3、测试 测试架 1 上的 MCU 主控板功能是否正常，正常放到正常区，不正常，放到不正常区
4、给测试架 1 卡上新的未测试的无程序的控制板
5、测试 测试架 2 上的未测试控制板/程序不知何时早就不知不觉的烧好了，换新的未测试未烧录的控制板
6、循环步骤 3 到步骤 5
========不需要安排烧录人员

STC MCU

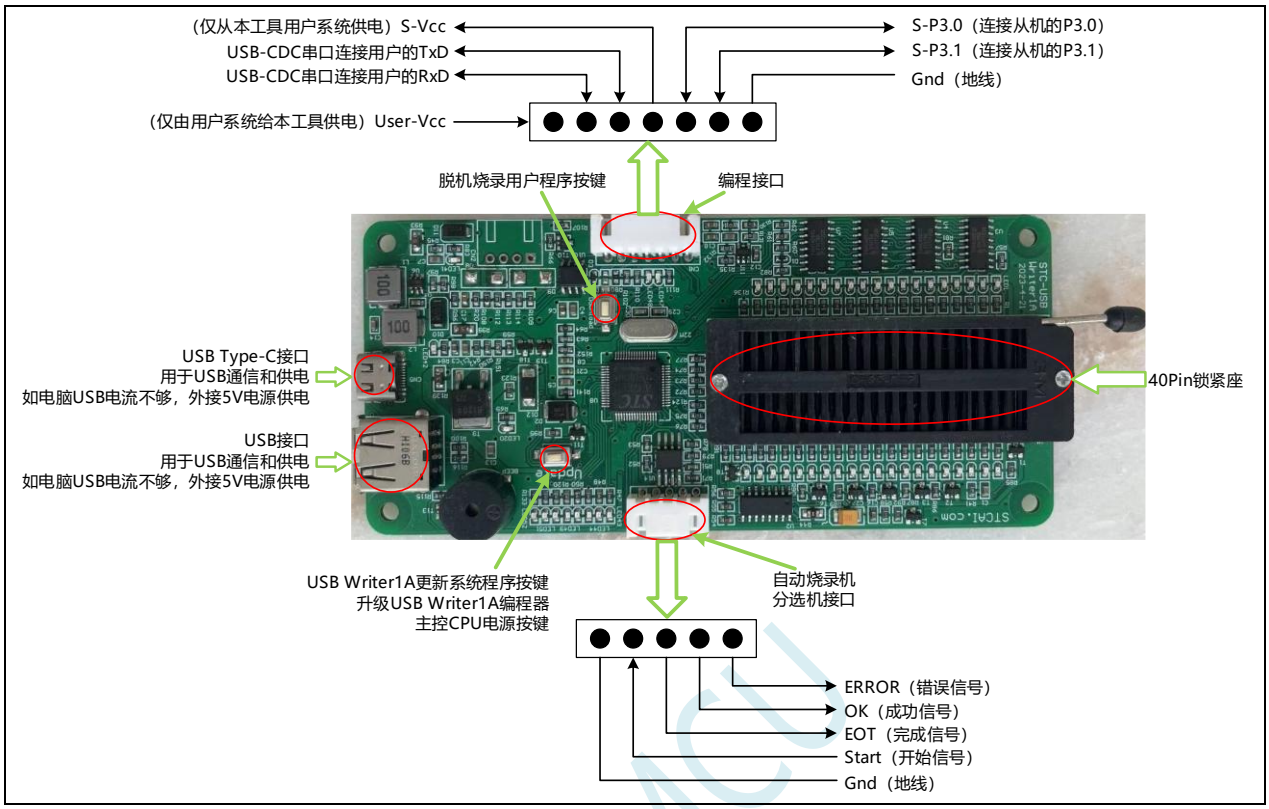
2.1.18 USB-Writer1A 编程器/烧录器 • 支持 • 插在 • 锁紧座上 • 烧录



USB型 脱机/联机烧录工具 USB-Writer1A（人民币100元）

1、LQFP64/48/44/32封装可以对应使用LQFP64/48/44/32转DIP64/48/44/32转换座，插在烧录锁紧座上进行烧录
2、QFN64/48/44/32封装可以对应使用QFN64/48/44/32转DIP64/48/44/32转换座，插在烧录锁紧座上进行烧录
3、PDIP40封装可以直接插在烧录锁紧座上进行烧录

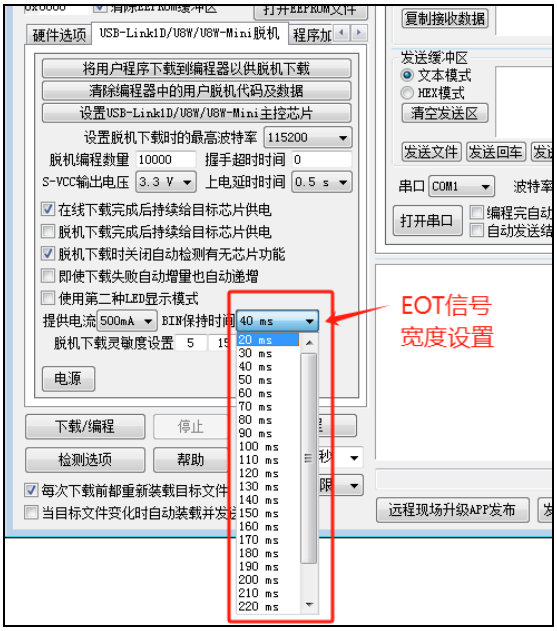
2.1.19 USB-Writer1A 支持 自动烧录机，通信协议和接口



自动烧录接口（分选机自动控制接口）协议：

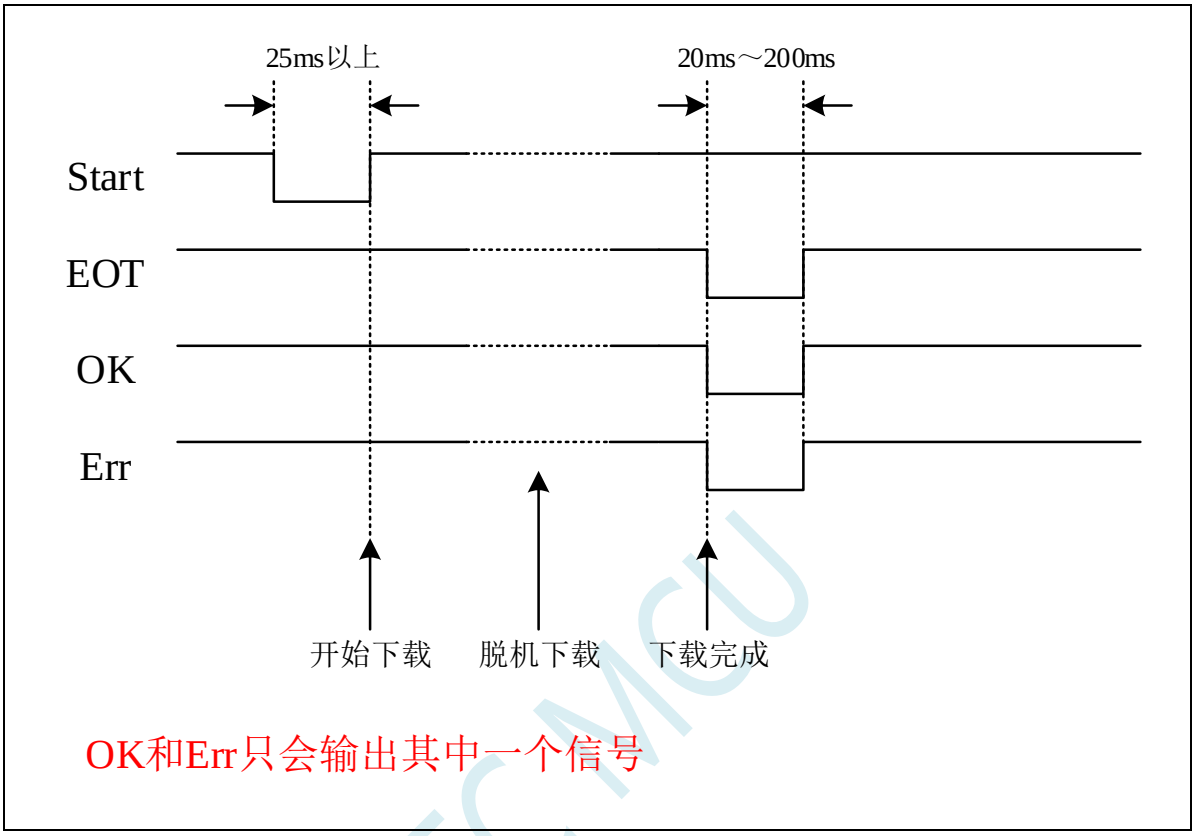
Start： 开始信号输入脚。从外部输入低电平信号触发开始脱机烧录，低电平必须维持 25 毫秒以上

EOT： 烧录完成信号输出脚。脱机烧录完成后，工具输出 20ms～250ms 的低电平 EOT 信号。电平宽度如下图所示的地方进行设置



OK： 良品信号输出脚。下载成功后工具从 OK 脚输出低电平信号，信号与 EOT 信号同步。

Err： 不良品信号输出脚。若下载失败，工具从 ERR 脚输出输出低电平信号，信号与 EOT 完成信号同步。



STCMCU

2.2 STC8C2K64S2-36I-LQFP44/32,QFN32,TSSOP/20 系列

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

➢ 选型价格（不需要外部晶振、不需要外部复位）

单片机型号	工作电压（V）	Flash 程序存储器 10 万次 字节	idata，内部传统 8051 RAM 字节	xdata，内部大容量扩展 SRAM 字节	强大的双 DPTR 可增可减	EEPROM 10 万次 字节	I/O 口最多数量	串口并可掉电唤醒	MDU16 硬件 16 位乘除法器	SPI	I²C	定时器计数器（T0-T4 外部管脚也可掉电唤醒）	15 位增强型 PWM 满足舞台灯光要求	PCA/CCP/PWM（可当外部中断并可掉电唤醒）	掉电唤醒专用定时器	15 路高速 ADC（全部 PWM 均可当 D/A 使用）	比较器（可当 1 路 A/D，可作外部掉电检测）	内部低压检测中断并可掉电唤醒	看门狗 复位定时器	内部高可靠复位（可选复位门槛电压）	内部高精准时钟（36MHz 以下可调）追频	可对外输出时钟及复位	程序加密后传输（防拦截）	可设置下次更新程序需口令	支持 RS485 下载	支持软件 USB 直接下载	本身就可在线仿真	价格及封装				主力产品供货信息
																												LQFP44	LQFP32	QFN32<4*4mm>	TSSOP20	
STC8C2K16S2	1.9-5.5	16K	256	2K	2	48K	42	2	有	有	有	5	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	√	√	√		现货
STC8C2K32S2	1.9-5.5	32K	256	2K	2	32K	42	2	有	有	有	5	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	¥2.2	¥1.99	¥1.99		
STC8C2K48S2	1.9-5.5	48K	256	2K	2	16K	42	2	有	有	有	5	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	¥2.2	¥1.99	¥1.99		
STC8C2K60S2	1.9-5.5	60K	256	2K	2	4K	42	2	有	有	有	5	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	√	√	√		
STC8C2K64S2	1.9-5.5	64K	256	2K	2	IAP	42	2	有	有	有	5	-	-	有	-	有	有	有	4 级	有	是	有	是	是	是	是	¥2.2	¥1.99	¥1.99	¥1.99	

注：以上的单价为 10K 及以上订货量的价格，量小则每片需增加 0.1 元人民币。

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

➢ **工作电压**
- ✓ 1.9V～5.5V
- ✓ 内建 LDO

➢ **工作温度**
- ✓ -40℃～85℃（超温度范围应用请参考电器特性章节说明）

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

➢ **SRAM**
- ✓ 128 字节内部直接访问 RAM（DATA）
- ✓ 128 字节内部间接访问 RAM（IDATA）
- ✓ 2048 字节内部扩展 RAM（内部 XDATA）

➢ **时钟控制**

车规启航
扫码去微信小商城

- ✓ 内部高精度、高稳定的高速 IRC（4MHz～38MHz，ISP 编程时可进行上下调整，还可以用户软件分频到较低的频率工作，如 100KHz）
  - ⌖ 误差±0.3%（常温下 25℃）
  - ⌖ -1.38%～+1.42%温漂（全温度范围，-40℃～85℃）
  - ⌖ -0.88%～+1.05%温漂（温度范围，-20℃～65℃）
- ✓ 内部 32KHz 低速 IRC（为了低功耗，省去了温度补偿和电压补偿电路，误差较大）
- ✓ 外部晶振（4MHz～38MHz）和外部时钟

（芯片上电工作过程：上电复位/复位脚复位/看门狗复位/低压检测复位时，芯片默认从 ISP 系统程序开始执行代码，此时固定使用内部 24MHz 的高速 IRC 时钟，当需要下载用户程序且下载完成后复位到用户程序区或者不需要下载直接复位到用户程序区时，默认会使用上次用户下载时所调节的高速 IRC 时钟，如果用户程序需要使用外部高速晶振、外部 32.768KHz 晶振或者内部 30KHz 低速 IRC，则需要用户软件先启动相应的时钟，然后通过设置 CLKSEL 寄存器进行切换）

- ➢ **复位**
  - ✓ 硬件复位
    - ⌖ 上电复位，实测电压值为 1.69V～1.82V。（**在芯片未使能低压复位功能时有效**）
      上电复位电压由一个上限电压和一个下限电压组成的电压范围，当工作电压从 5V/3.3V 向下掉到上电复位的下限门槛电压时，芯片处于复位状态；当电压从 0V 上升到上电复位的上限门槛电压时，芯片解除复位状态。
    - ⌖ 复位脚复位，出厂时 P5.4 默认为 I/O 口，ISP 下载时可将 P5.4 管脚设置为复位脚（**注意：当设置 P5.4 管脚为复位脚时，复位电平为低电平**）
    - ⌖ 看门狗溢出复位
    - ⌖ 低压检测复位，提供 4 级低压检测电压：2.0V（实测为 1.90V～2.04V）、2.4V（实测为 2.30V～2.50V）、2.7V（实测为 2.61V～2.82V）、3.0V（实测为 2.90V～3.13V）。
      每级低压检测电压都是由一个上限电压和一个下限电压组成的电压范围，当工作电压从 5V/3.3V 向下掉到低压检测的下限门槛电压时，低压检测生效；当电压从 0V 上升到低压检测的上限门槛电压时，低压检测生效。
  - ✓ 软件复位
    - ⌖ 软件方式写复位触发寄存器

- ➢ **中断**
  - ✓ 提供 16 个中断源：INT0（支持上升沿和下降沿中断）、INT1（支持上升沿和下降沿中断）、INT2（只支持下降沿中断）、INT3（只支持下降沿中断）、INT4（只支持下降沿中断）、定时器 0、定时器 1、定时器 2、定时器 3、定时器 4、串口 1、串口 2、LVD 低压检测、SPI、I2C、比较器。
  - ✓ 提供 4 级中断优先级
  - ✓ 时钟停振模式下可以唤醒的中断：INT0(P3.2)、INT1(P3.3)、INT2(P3.6)、INT3(P3.7)、INT4(P3.0)、T0(P3.4)、T1(P3.5) 、 T2(P1.2) 、 T3(P0.4) 、 T4(P0.6) 、 RXD(P3.0/P3.6/P1.6/P4.3) 、 RXD2(P1.0/P4.0) 、 I2C_SDA(P1.4/P2.4/P3.3)、SPI_SS(P1.2/P2.2/P3.5)以及所有端口的 I/O 中断、比较器中断、低压检测中断、掉电唤醒定时器唤醒

- ➢ **数字外设**
  - ✓ 5 个 16 位定时器：定时器 0、定时器 1、定时器 2、定时器 3、定时器 4，其中定时器 0 的模式 3 具有 NMI（不可屏蔽中断）功能，定时器 0 和定时器 1 的模式 0 为 16 位自动重载模式
  - ✓ 2 个高速串口：串口 1、串口 2，波特率时钟源最快可为 FOSC/4
  - ✓ SPI：支持主机模式和从机模式以及主机/从机自动切换
  - ✓ I^2C：支持主机模式和从机模式
  - ✓ **MDU16：硬件 16 位乘除法器（支持 32 位除以 16 位、16 位除以 16 位、16 位乘 16 位、数据移位以及数**

据规格化等运算）

- ✓ **I/O 口中断：所有的 I/O 均支持中断，每组 I/O 中断有独立的中断入口地址，所有的 I/O 中断可支持 4 种中断模式：高电平中断、低电平中断、上升沿中断、下降沿中断**

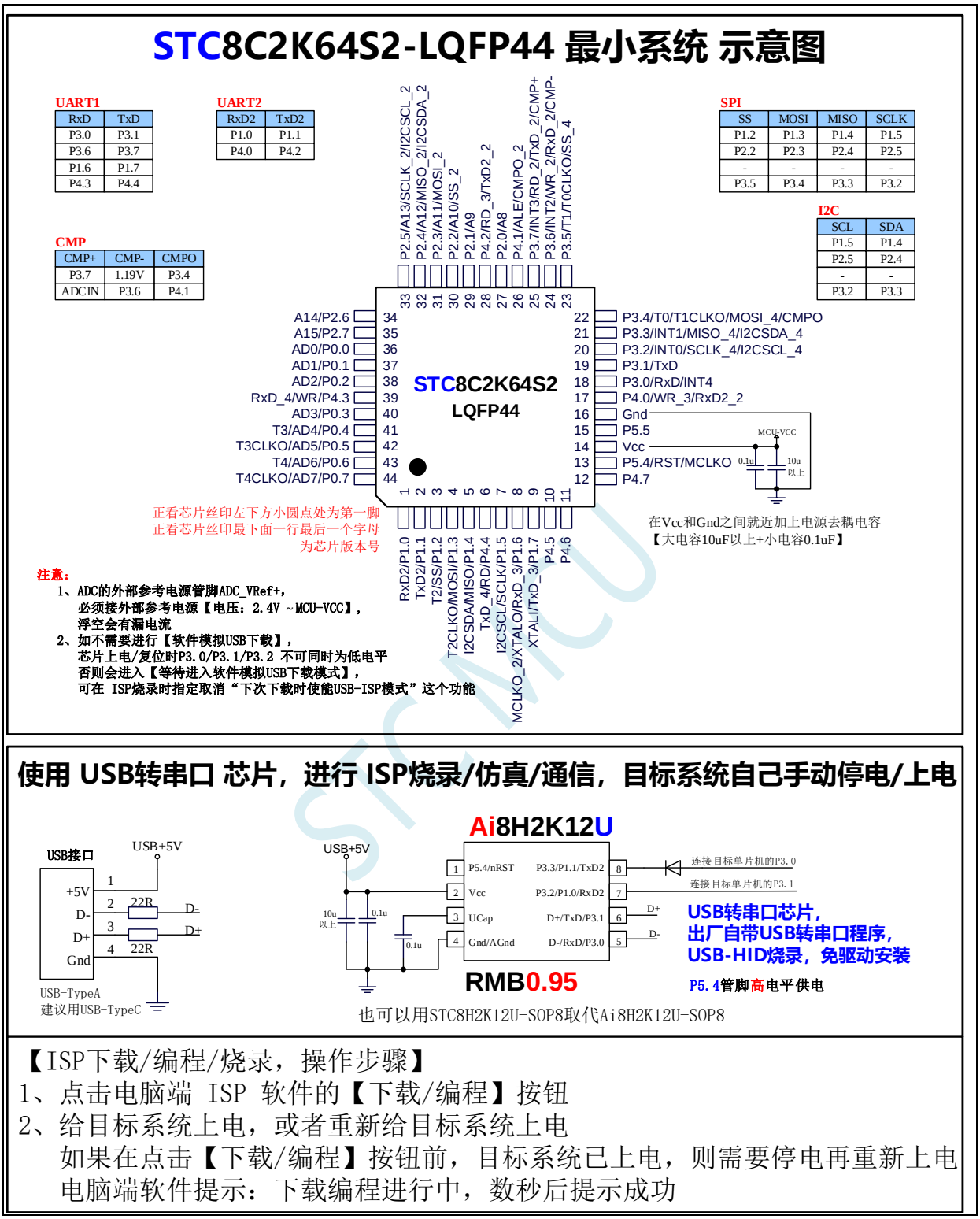
- ➢ **模拟外设**
  - ✓ 比较器，一组比较器（比较器的正端可选择 CMP+端口和所有的 ADC 输入端口，所以比较器可当作多路比较器进行分时复用）

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

- ➢ **封装**
  - ✓ LQFP44、LQFP32、QFN32<4mm*4mm>、TSSOP20

STC MCU

2.2.2　STC8C2K64S4-36I-LQFP44 管脚图，ISP 下载/编程线路图



【ISP下载/编程/烧录，操作步骤】

1、点击电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新给目标系统上电

如果在点击【下载/编程】按钮前，目标系统已上电，则需要停电再重新上电

电脑端软件提示：下载编程进行中，数秒后提示成功

关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式

2、 除 P3.0 和 P3.1 外，其余所有 IO 口上电后的状态均为高阻输入状态，用户在使用 IO 口前必须先设置 IO 口模式

3、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能

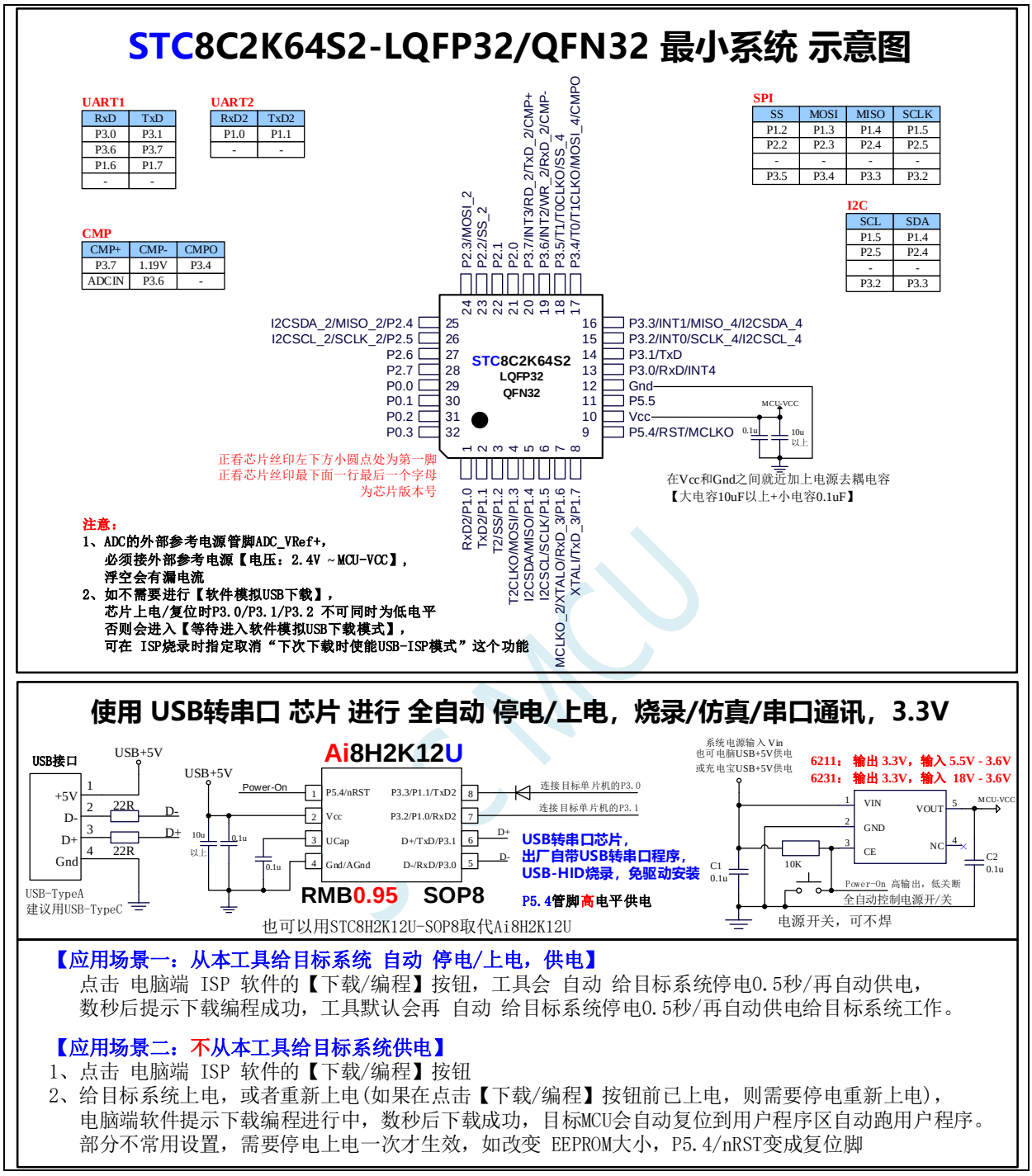
同时为低电平，否则会进入 USB 下载模式而无法运行用户代码

4、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间由高阻输入状态切换到双向口模式，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式

5、 当使用 P5.4 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P5.4 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P5.4 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

STC MCU

2.2.3 STC8C2K64S4-36I-LQFP32/QFN32 管脚图，ISP 下载/编程线路图



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式

2、 除 P3.0 和 P3.1 外，其余所有 IO 口上电后的状态均为高阻输入状态，用户在使用 IO 口前必须先设置 IO 口模式

3、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码

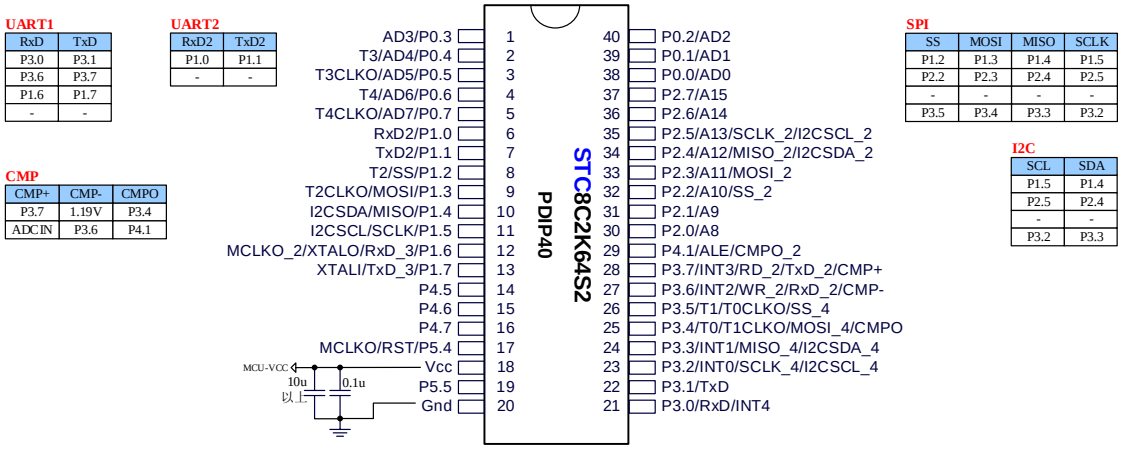
4、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间由高阻输入状态切换到双向口模式，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式

5、 当使用 P5.4 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P5.4 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P5.4 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

STC MCU

2.2.4 STC8C2K64S4-36I-PDIP40 管脚图，ISP 下载/编程线路图

STC8C2K64S2-PDIP40 最小系统 示意图



在Vcc和Gnd之间就近加上电源去耦电容【大电容10uF以上+小电容0.1uF】

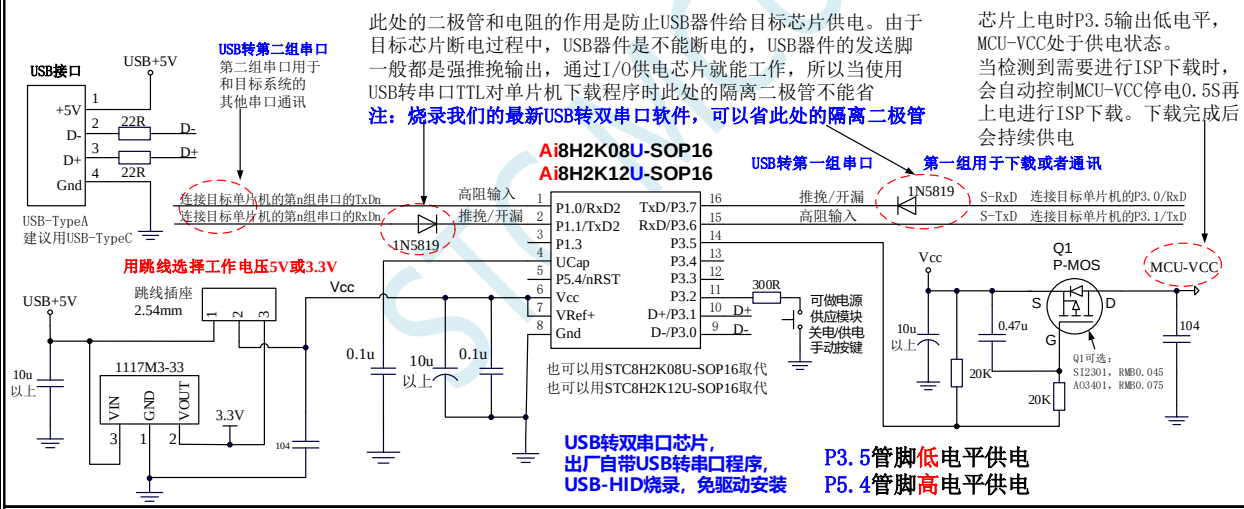
注意：

1、ADC的外部参考电源管脚ADC_VRef+，必须接外部参考电源【电压：2.4V ~MCU-VCC】,浮空会有漏电流

2、如不需要进行【软件模拟USB下载】，芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平

否则会进入【等待进入软件模拟USB下载模式】，可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

使用 USB转双串口 芯片 进行 全自动 烧录/仿真+串口通讯，5V/3.3V跳线选择



USB转双串口芯片，出厂自带USB转串口程序，USB-HID烧录，免驱动安装

P3.5管脚低电平供电

P5.4管脚高电平供电

【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，
数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，
电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

关于 I/O 的注意事项：

1、 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式

2、 除 P3.0 和 P3.1 外，其余所有 IO 口上电后的状态均为高阻输入状态，用户在使用 IO 口前必须先设置 IO 口模式

3、 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码

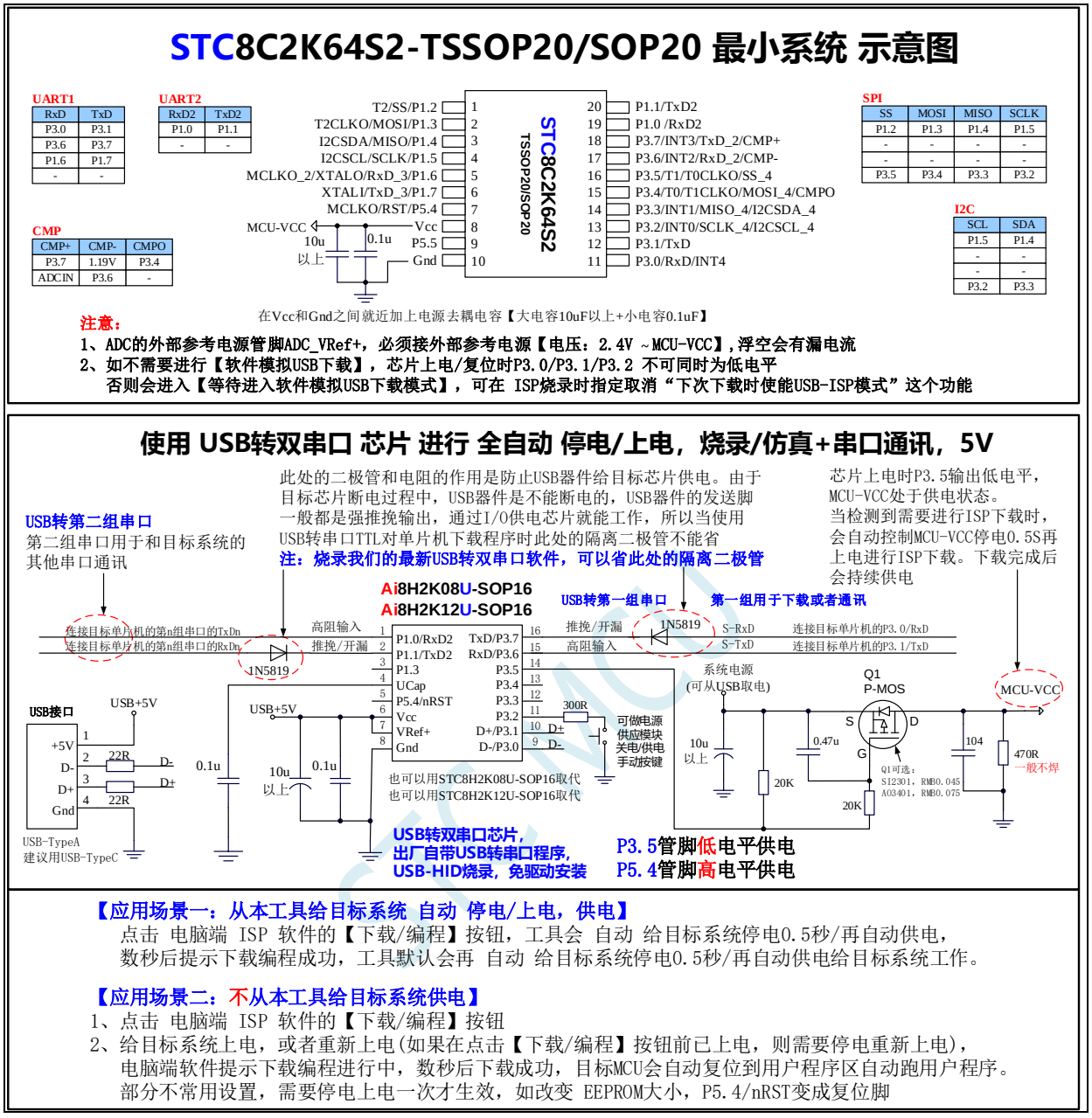
4、 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间由高阻输入状态切换

到双向口模式，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式

5、 当使用 P5.4 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P5.4 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P5.4 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

STC MCU

2.2.5　STC8C2K64S4-36I-TSSOP20/SOP20 管脚图，ISP 下载/编程线路图



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】
1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

关于 I/O 的注意事项：

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

2.2.6 管脚说明

编号				名称	类型	说明
LQFP44	PDIP40	LQFP32 QFN32	TSOP20			
2	7	2	20	P1.1	I/O	标准 IO 口
				TxD2	O	串口 2 的发送脚
3	8	3	1	P1.2	I/O	标准 IO 口
				SS	I	SPI 的从机选择脚（主机为输出）
				T2	I	定时器 2 外部时钟输入
4	9	4	2	P1.3	I/O	标准 IO 口
				MOSI	I/O	SPI 主机输出从机输入
				T2CLKO	O	定时器 2 时钟分频输出
5	10	5	3	P1.4	I/O	标准 IO 口
				MISO	I/O	SPI 主机输入从机输出
				SDA	I/O	I2C 接口的数据线
6				P4.4	I/O	标准 IO 口
				RD	O	外部总线的读信号线
				TxD_4	O	串口 1 的发送脚
7	11	6	4	P1.5	I/O	标准 IO 口
				SCLK	I/O	SPI 的时钟脚
				SCL	I/O	I2C 的时钟线
8	12	7	5	P1.6	I/O	标准 IO 口
				RxD_3	I	串口 1 的接收脚
				XTALO	O	外部晶振的输出脚
				MCLKO_2	O	主时钟分频输出
9	13	8	6	P1.7	I/O	标准 IO 口
				TxD_3	O	串口 1 的发送脚
				XTALI	I	外部晶振/外部时钟的输入脚
10	14			P4.5	I/O	标准 IO 口
11	15			P4.6	I/O	标准 IO 口
12	16			P4.7	I/O	标准 IO 口
13	17	9	7	P5.4	I/O	标准 IO 口
				RST	I	复位引脚
				MCLKO	O	主时钟分频输出
14	18	10	8	Vcc	VCC	电源脚

编号				名称	类型	说明
LQFP44	PDIP40	LQFP32 QFN32	TSSOP20			
15	19	11	9	P5.5	I/O	标准 IO 口
16	20	12	10	Gnd	GND	地线
17				P4.0	I/O	标准 IO 口
				WR_3	O	外部总线的写信号线
				RxD2_2	I	串口 2 的接收脚
18	21	13	11	P3.0	I/O	标准 IO 口
				RxD	I	串口 1 的接收脚
				INT4	I	外部中断 4
19	22	14	12	P3.1	I/O	标准 IO 口
				TxD	O	串口 1 的发送脚
20	23	15	13	P3.2	I/O	标准 IO 口
				INT0	I	外部中断 0
				SCL_4	I/O	I2C 的时钟线
				SCLK_4	I/O	SPI 的时钟脚
21	24	16	14	P3.3	I/O	标准 IO 口
				INT1	I	外部中断 1
				SDA_4	I/O	I2C 接口的数据线
				MISO_4	I/O	SPI 主机输入从机输出
22	25	17	15	P3.4	I/O	标准 IO 口
				T0	I	定时器 0 外部时钟输入
				T1CLKO	O	定时器 1 时钟分频输出
				MOSI_4	I/O	SPI 主机输出从机输入
				CMPO	O	比较器输出
23	26	18	16	P3.5	I/O	标准 IO 口
				T1	I	定时器 1 外部时钟输入
				T0CLKO	O	定时器 0 时钟分频输出
				SS_4	I	SPI 的从机选择脚（主机为输出）
24	27	19	17	P3.6	I/O	标准 IO 口
				INT2	I	外部中断 2
				WR_2	O	外部总线的写信号线
				RxD_2	I	串口 1 的接收脚
				CMP-	I	比较器负极输入

编号				名称	类型	说明
LQFP44	PDIP40	LQFP32 QFN32	TSSOP20			
25	28	20	18	P3.7	I/O	标准 IO 口
				INT3	I	外部中断 3
				RD_2	O	外部总线的读信号线
				TxD_2	O	串口 1 的发送脚
				CMP+	I	比较器正极输入
26	29			P4.1	I/O	标准 IO 口
				ALE	O	地址锁存信号
				CMPO_2	O	比较器输出
27	30	21		P2.0	I/O	标准 IO 口
				A8	O	地址总线
				RSTSV	-	端口的初始电平可 ISP 下载时配置
28				P4.2	I/O	标准 IO 口
				RD_3	O	外部总线的读信号线
				TxD2_2	O	串口 2 的发送脚
29	31	22		P2.1	I/O	标准 IO 口
				A9	O	地址总线
30	32	23		P2.2	I/O	标准 IO 口
				A10	O	地址总线
				SS_2	I	SPI 的从机选择脚（主机为输出）
31	33	24		P2.3	I/O	标准 IO 口
				A11	O	地址总线
				MOSI_2	I/O	SPI 主机输出从机输入
32	34	25		P2.4	I/O	标准 IO 口
				A12	O	地址总线
				MISO_2	I/O	SPI 主机输入从机输出
				SDA_2	I/O	I2C 接口的数据线
33	35	26		P2.5	I/O	标准 IO 口
				A13	O	地址总线
				SCLK_2	I/O	SPI 的时钟脚
				SCL_2	I/O	I2C 的时钟线
34	36	27		P2.6	I/O	标准 IO 口
				A14	O	地址总线
35	37	28		P2.7	I/O	标准 IO 口
				A15	O	地址总线

编号				名称	类型	说明
LQFP44	PDIP40	LQFP32 QFN32	TSSOP20			
36	38	29		P0.0	I/O	标准 IO 口
				AD0	I/O	地址/数据总线
				RxD3	I	串口 3 的接收脚
37	39	30		P0.1	I/O	标准 IO 口
				AD1	I/O	地址/数据总线
				TxD3	O	串口 3 的发送脚
38	40	31		P0.2	I/O	标准 IO 口
				AD2	I/O	地址/数据总线
				RxD4	I	串口 4 的接收脚
39				P4.3	I/O	标准 IO 口
				WR	O	外部总线的写信号线
				RxD_4	I	串口 1 的接收脚
40	1	32		P0.3	I/O	标准 IO 口
				AD3	I/O	地址/数据总线
				TxD4	O	串口 4 的发送脚
41	2			P0.4	I/O	标准 IO 口
				AD4	I/O	地址/数据总线
				T3	I	定时器 3 外部时钟输入
42	3			P0.5	I/O	标准 IO 口
				AD5	I/O	地址/数据总线
				T3CLKO	O	定时器 3 时钟分频输出
43	4			P0.6	I/O	标准 IO 口
				AD6	I/O	地址/数据总线
				T4	I	定时器 4 外部时钟输入
44	5			P0.7	I/O	标准 IO 口
				AD7	I/O	地址/数据总线
				T4CLKO	O	定时器 4 时钟分频输出
1	6	1	19	P1.0	I/O	标准 IO 口
				RxD2	I	串口 2 的接收脚

2.2.7　USB-Link1D 对 STC8C 系列进行自动串口烧录、仿真+串口通讯

使用【USB Link1D】对 STC8C2K64S2系列 进行全自动停电/上电串口烧录、仿真+串口通讯

【USB Link1D】工具：支持 全自动 停电 / 上电，在线下载 / 脱机下载，仿真



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

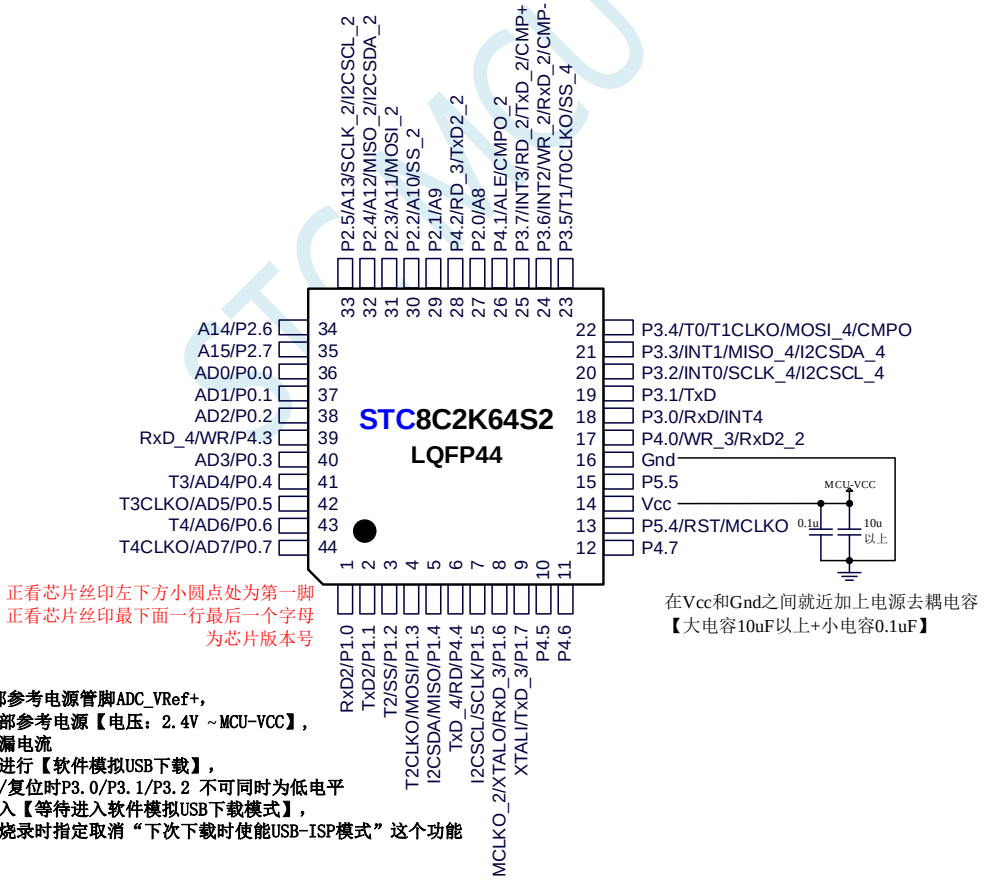
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

STC8C2K64S2-LQFP44 最小系统 示意图



正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号

在Vcc和Gnd之间就近加上电源去耦电容【大电容10uF以上+小电容0.1uF】

注意：

1、ADC的外部参考电源管脚ADC_VRef+，必须接外部参考电源【电压：2.4V ~MCU-VCC】，浮空会有漏电流

2、如不需要进行【软件模拟USB下载】，芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平否则会进入【等待进入软件模拟USB下载模式】，可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

2.2.8　使用【一箭双雕之 USB 转双串口】进行串口烧录、仿真+串口通讯

使用【一箭双雕之USB转双串口】对 STC8C2K64S2系列 进行串口烧录、仿真+串口通讯

【一箭双雕之USB转双串口】：支持 全自动 停电 / 上电，在线下载，仿真



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

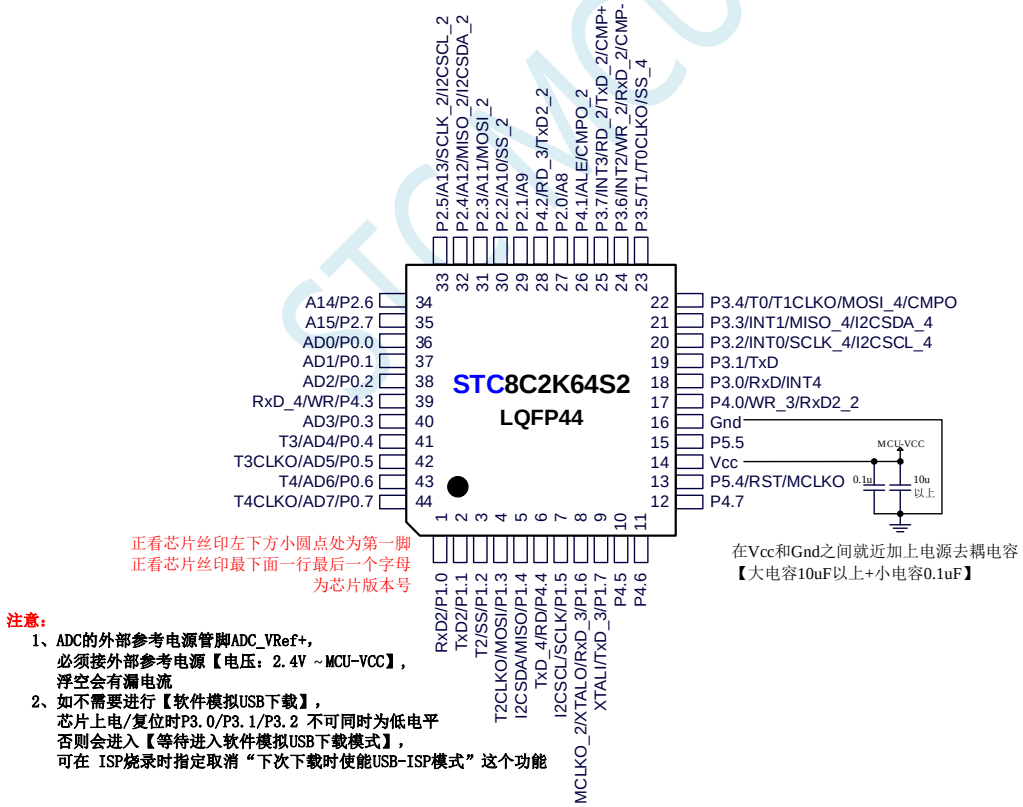
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区跑用户程序。

部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

STC8C2K64S2-LQFP44 最小系统 示意图



正看芯片丝印左下方小圆点处为第一脚
正看芯片丝印最下面一行最后一个字母为芯片版本号

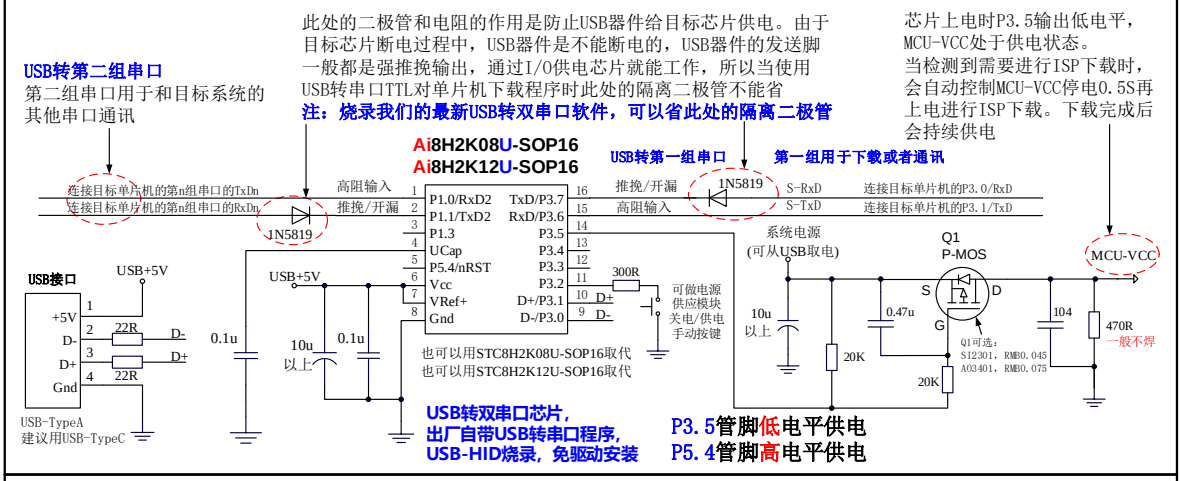
在Vcc和Gnd之间就近加上电源去耦电容
【大电容10uF以上+小电容0.1uF】

注意：

1、ADC的外部参考电源管脚ADC_VRef+，必须接外部参考电源【电压：2.4V ~MCU-VCC】，浮空会有漏电流

2、如不需要进行【软件模拟USB下载】，芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平否则会进入【等待进入软件模拟USB下载模式】，可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

2.2.9 USB 转双串口芯片全自动停电/上电烧录，串口仿真+串口通讯，5V



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

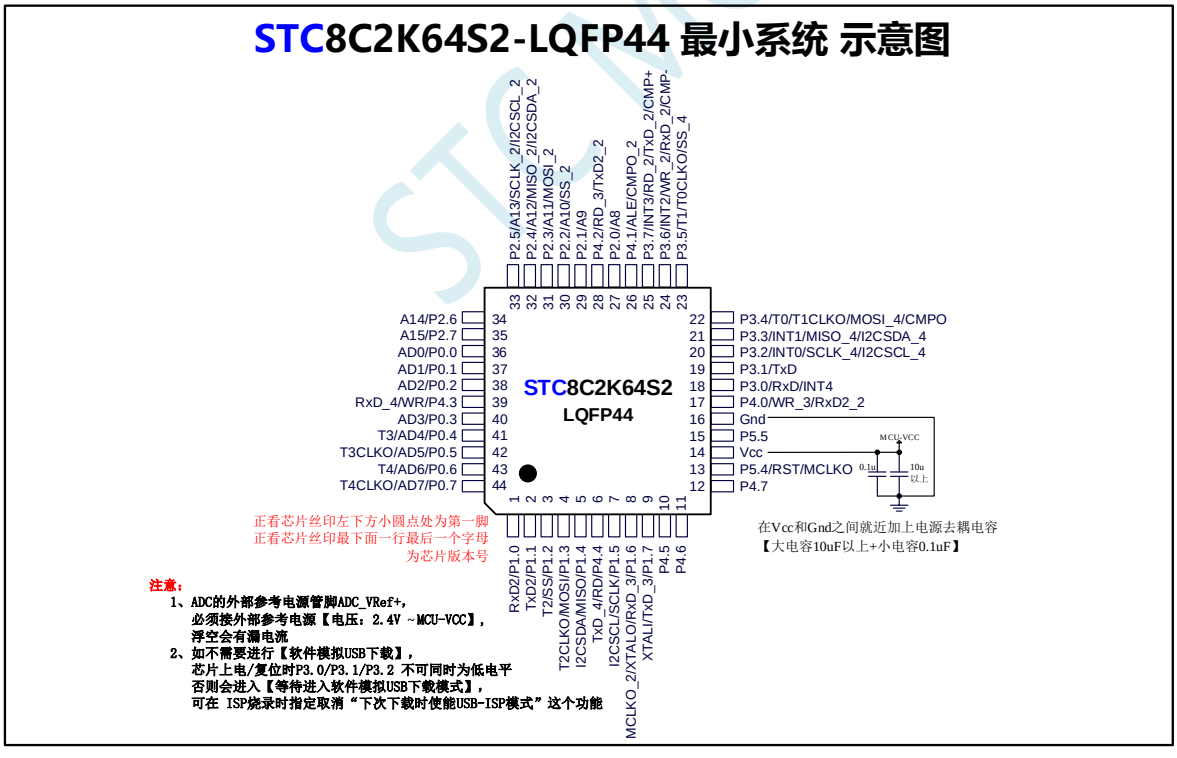
点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

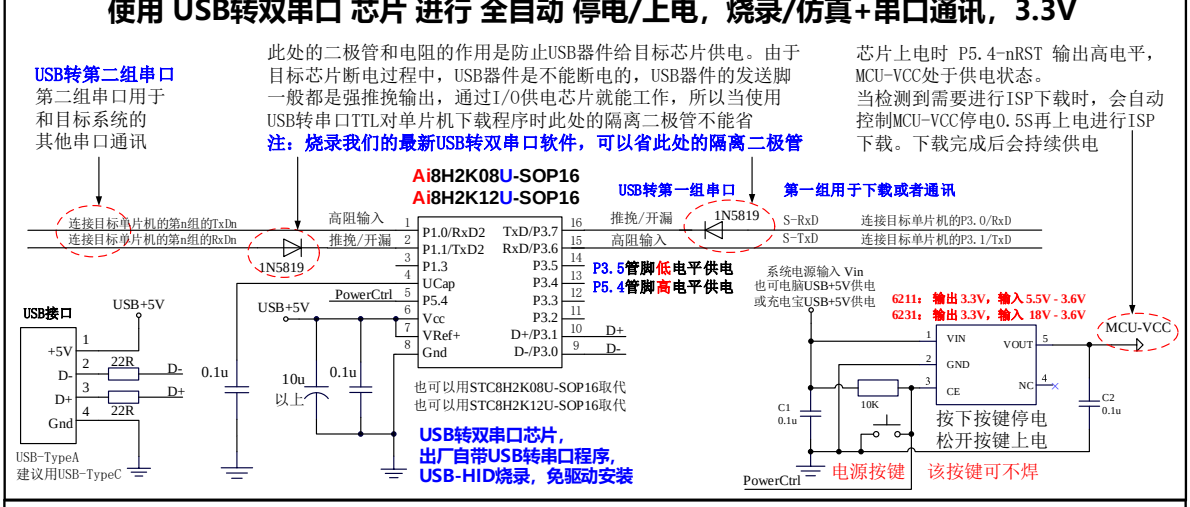
1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。

部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚



2.2.10 USB 转双串口芯片全自动烧录，串口仿真+串口通讯，3.3V 原理图

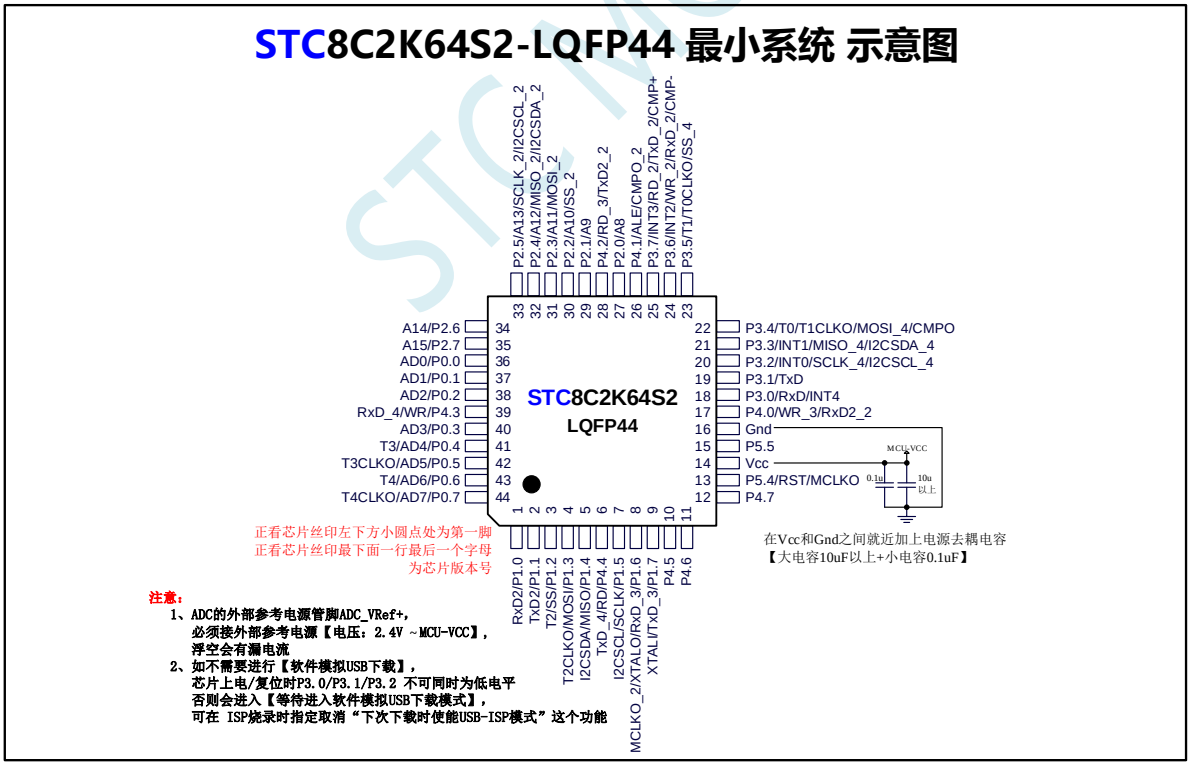


【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，
数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

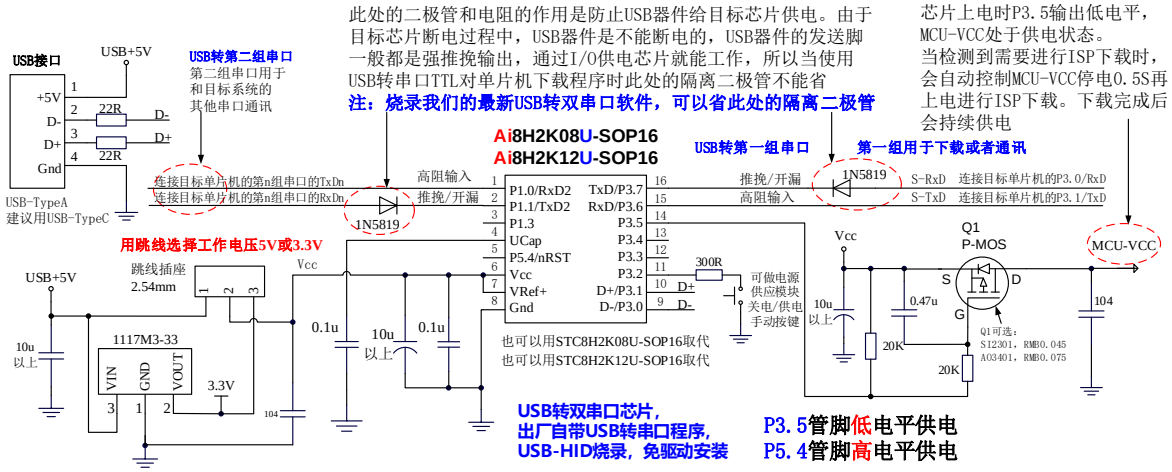
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，
电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚



2.2.11 USB 转双串口芯片进行自动烧录/仿真+串口通讯，5V/3.3V 跳线选择

使用 USB转双串口 芯片 进行 全自动 烧录/仿真+串口通讯，5V/3.3V跳线选择



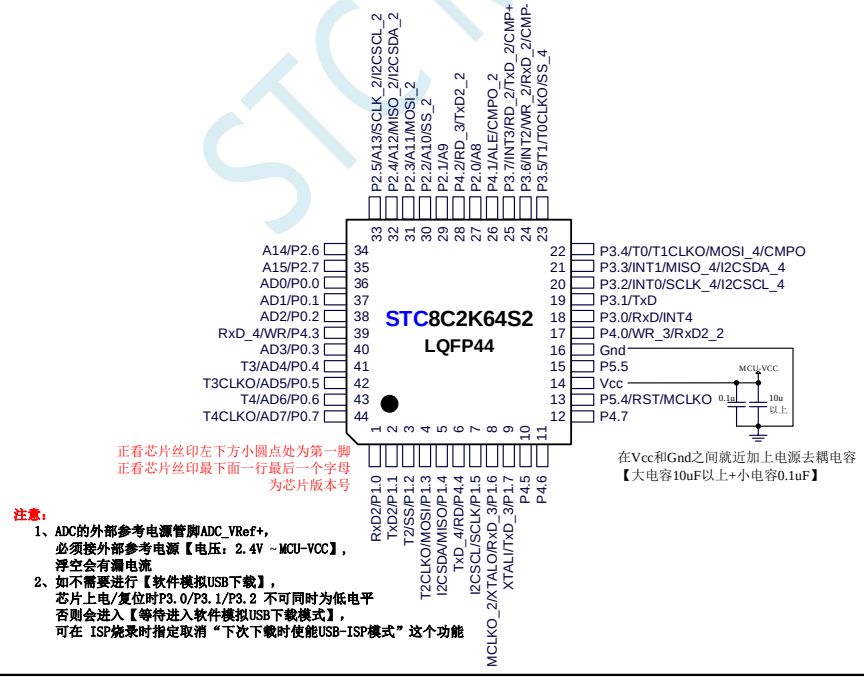
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，
数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，
电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

STC8C2K64S2-LQFP44 最小系统 示意图

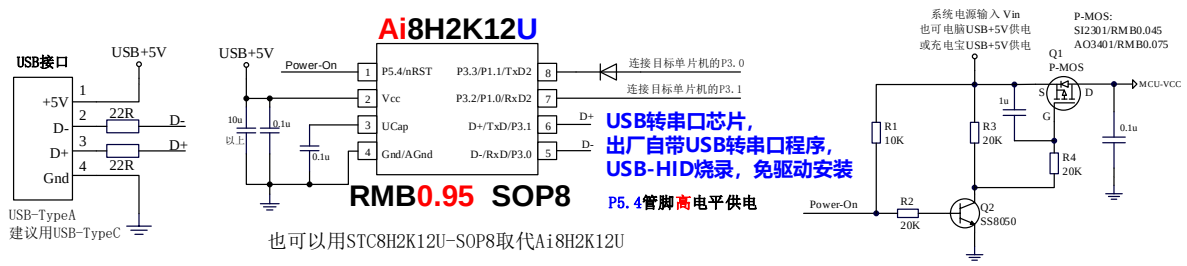


注意：

1、ADC的外部参考电源管脚ADC_VRef+，
必须接外部参考电源【电压：2.4V ~MCU-VCC】，
浮空会有漏电流
2、如不需要进行【软件模拟USB下载】，
芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平
否则会进入【等待进入软件模拟USB下载模式】，
可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

2.2.12 通用 USB 转串口芯片全自动停电/上电烧录，串口仿真，5V 原理图

使用 USB转串口 芯片 进行 全自动 停电/上电，烧录/仿真/串口通讯，5V



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

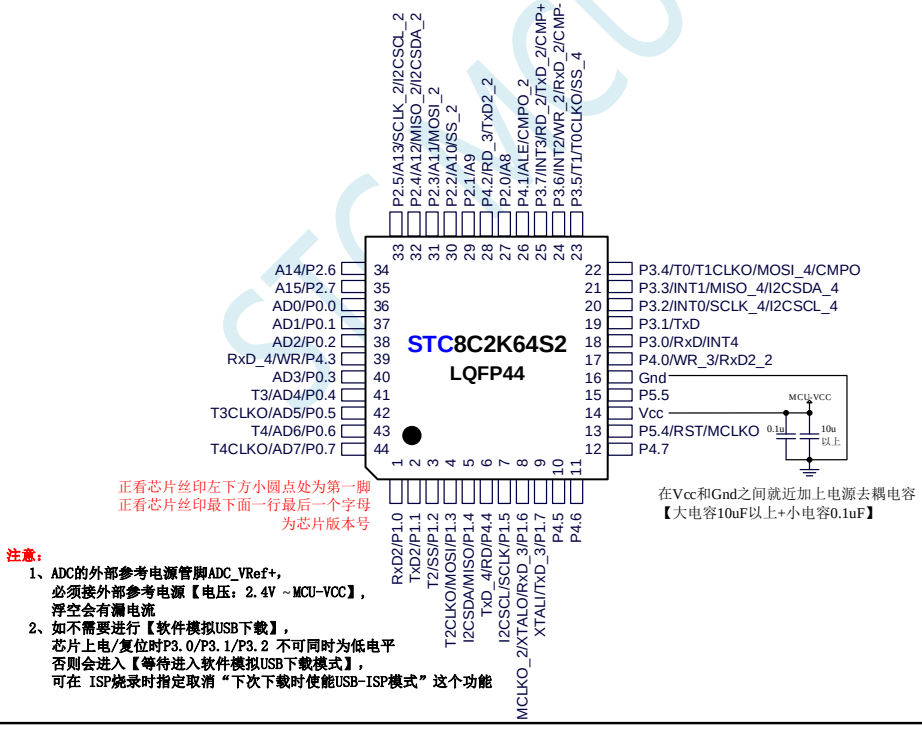
【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。

部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

STC8C2K64S2-LQFP44 最小系统 示意图



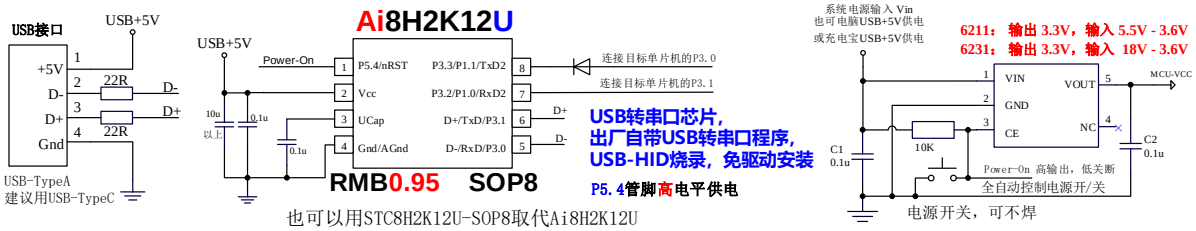
注意：

1、ADC的外部参考电源管脚ADC_VRef+，必须接外部参考电源【电压：2.4V ~MCU-VCC】，浮空会有漏电流

2、如不需要进行【软件模拟USB下载】，芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平否则会进入【等待进入软件模拟USB下载模式】，可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

2.2.13 通用 USB 转串口芯片全自动停电/上电烧录，串口仿真，3.3V 原理图

使用 USB转串口 芯片 进行 全自动 停电/上电，烧录/仿真/串口通讯，3.3V



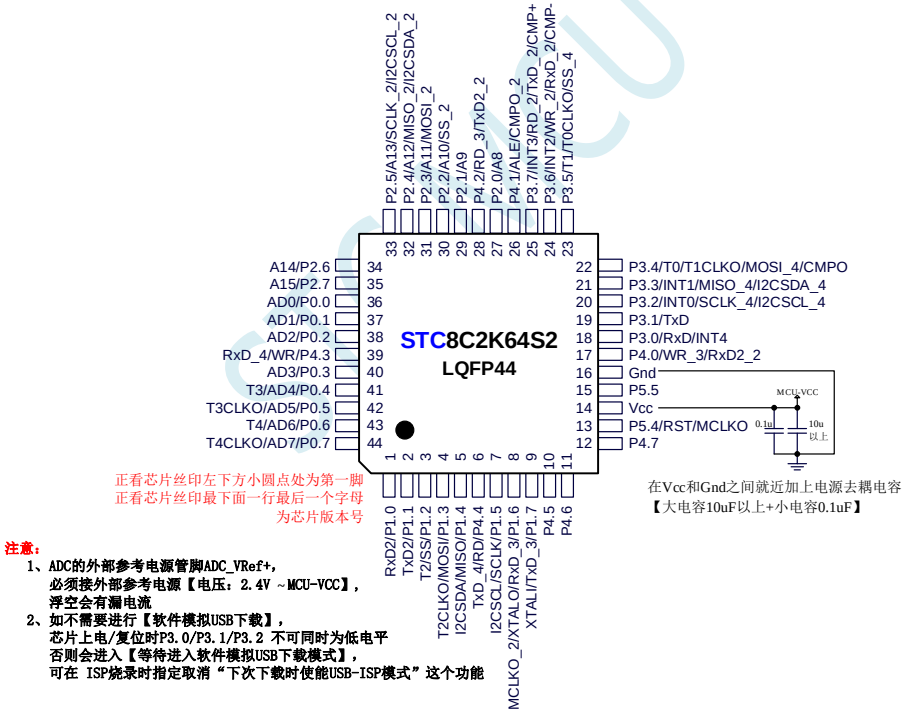
【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，
数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

1、点击 电脑端 ISP 软件的【下载/编程】按钮
2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，
电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。
部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

STC8C2K64S2-LQFP44 最小系统 示意图

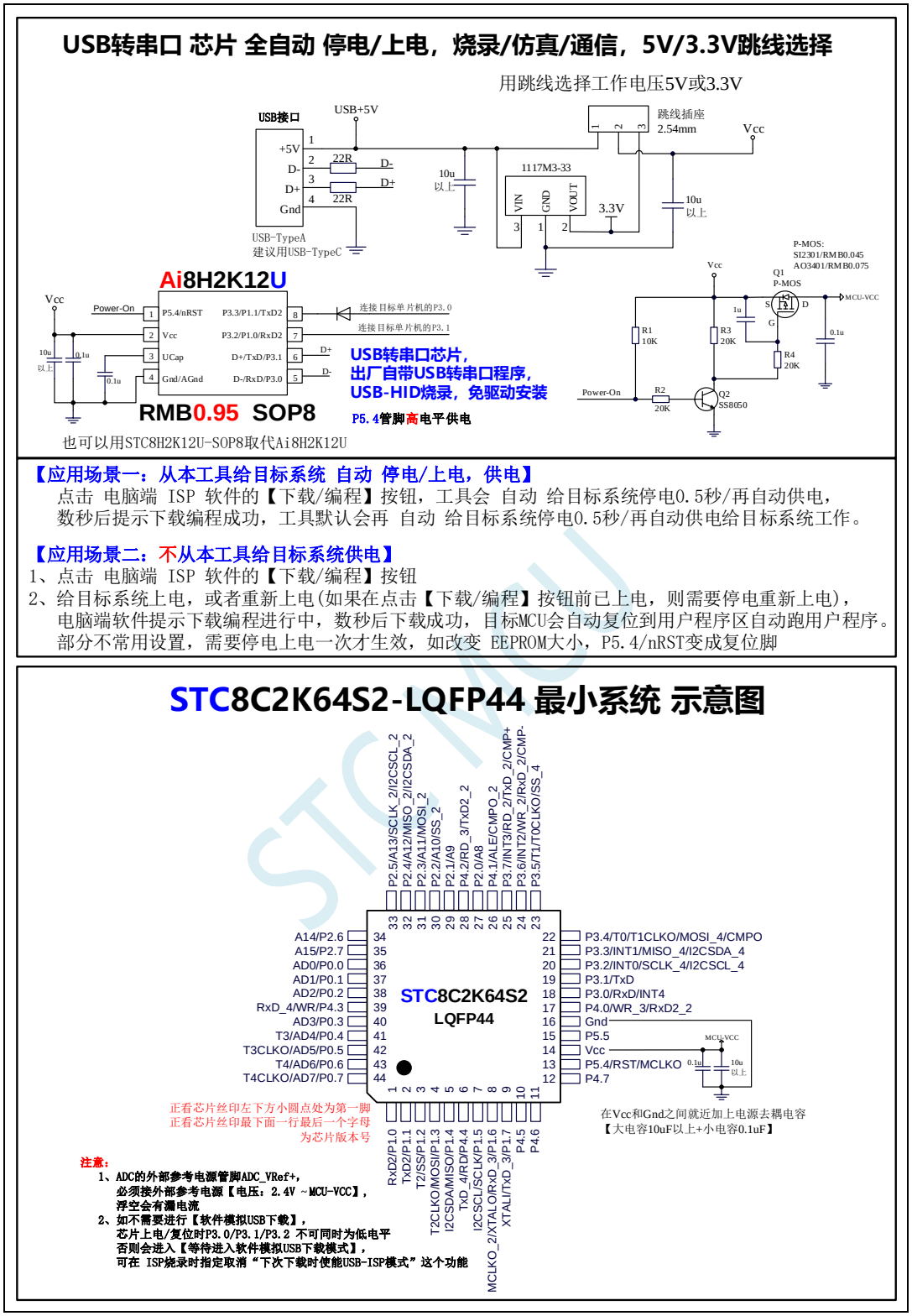


注意：

1、ADC的外部参考电源管脚ADC_VRef+，
必须接外部参考电源【电压：2.4V ~MCU-VCC】，
浮空会有漏电流

2、如不需要进行【软件模拟USB下载】，
芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平
否则会进入【等待进入软件模拟USB下载模式】，
可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

2.2.14 USB 转串口芯片全自动停电/上电烧录/仿真/通信，5V/3.3V 跳线选择



【应用场景一：从本工具给目标系统 自动 停电/上电，供电】

点击 电脑端 ISP 软件的【下载/编程】按钮，工具会 自动 给目标系统停电0.5秒/再自动供电，数秒后提示下载编程成功，工具默认会再 自动 给目标系统停电0.5秒/再自动供电给目标系统工作。

【应用场景二：不从本工具给目标系统供电】

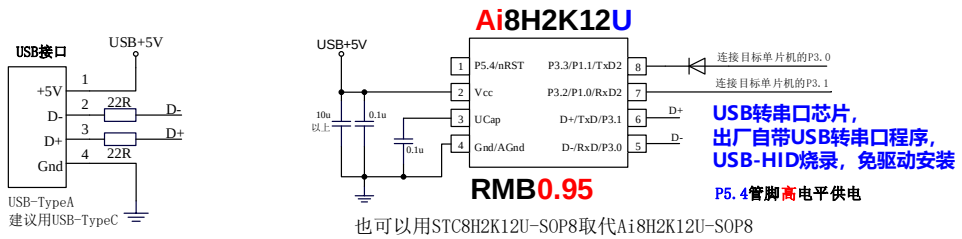
1、点击 电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新上电(如果在点击【下载/编程】按钮前已上电，则需要停电重新上电)，电脑端软件提示下载编程进行中，数秒后下载成功，目标MCU会自动复位到用户程序区自动跑用户程序。

部分不常用设置，需要停电上电一次才生效，如改变 EEPROM大小，P5.4/nRST变成复位脚

2.2.15 USB 转串口芯片进行烧录/串口仿真，手动停电/上电，5V/3.3V 原理图

使用 USB转串口 芯片，进行 ISP烧录/仿真/通信，目标系统自己手动停电/上电



也可以用STC8H2K12U-SOP8取代Ai8H2K12U-SOP8

【ISP下载/编程/烧录，操作步骤】

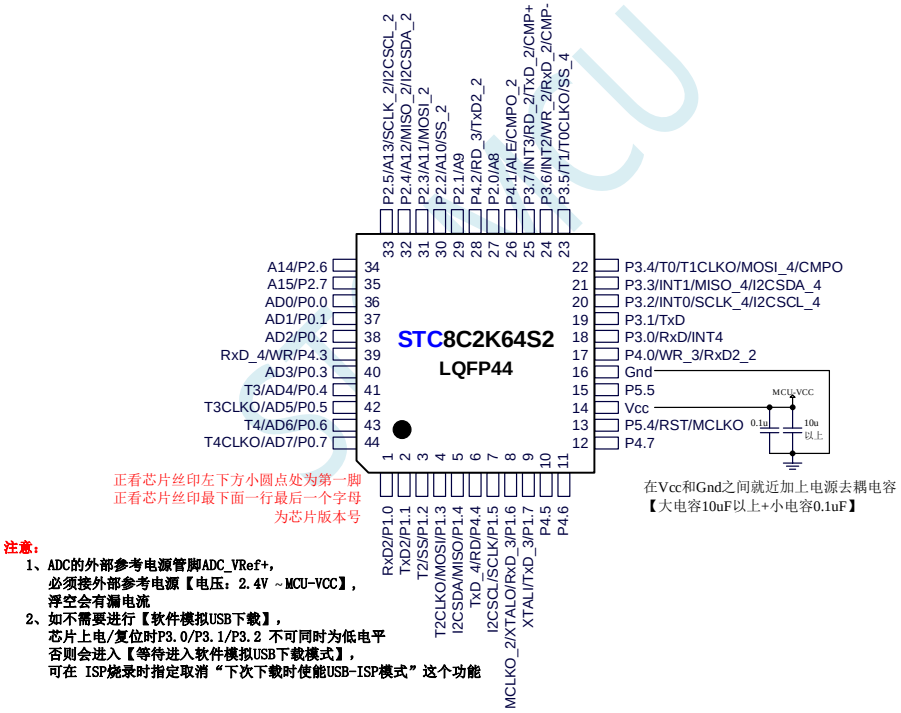
1、点击电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新给目标系统上电

如果在点击【下载/编程】按钮前，目标系统已上电，则需要停电再重新上电

电脑端软件提示：下载编程进行中，数秒后提示成功

STC8C2K64S2-LQFP44 最小系统 示意图



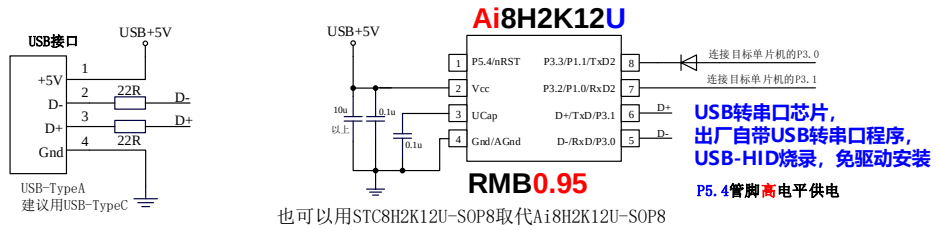
注意：

1、ADC的外部参考电源管脚ADC_VRef+，必须接外部参考电源【电压：2.4V ~MCU-VCC】，浮空会有漏电流

2、如不需要进行【软件模拟USB下载】，芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平否则会进入【等待进入软件模拟USB下载模式】，可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

2.2.16 USB 转串口芯片进行烧录，串口仿真，手动停电/上电，3.3V 原理图

使用 USB转串口 芯片，进行 ISP烧录/仿真/通信，目标系统自己 手动 停电/上电



也可以用STC8H2K12U-SOP8取代Ai8H2K12U-SOP8

【ISP下载/编程/烧录，操作步骤】

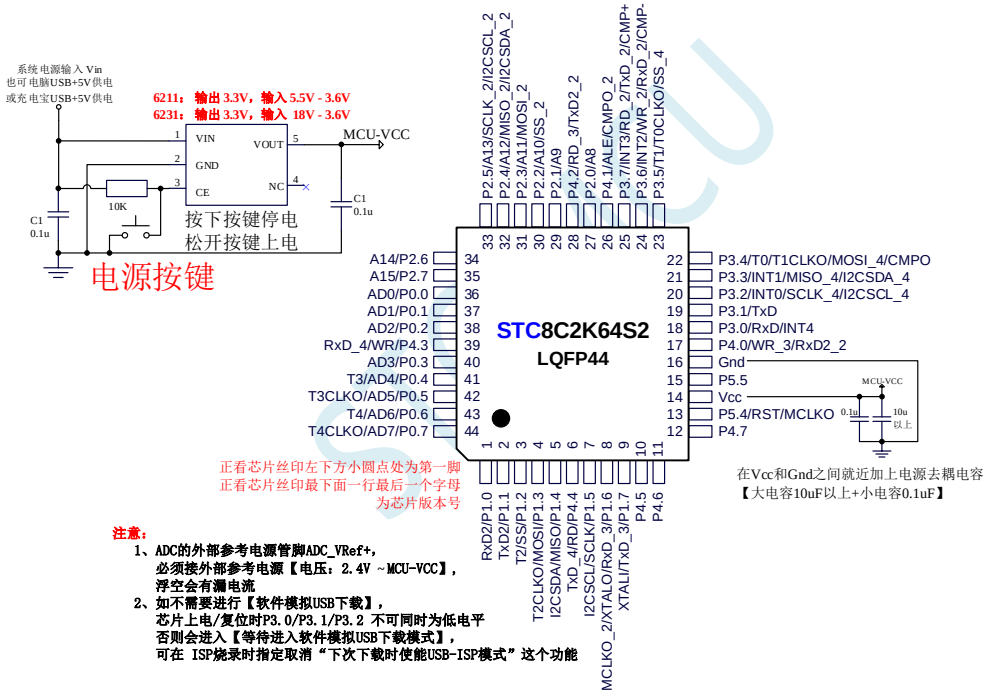
1、点击电脑端 ISP 软件的【下载/编程】按钮

2、给目标系统上电，或者重新给目标系统上电

如果在点击【下载/编程】按钮前，目标系统已上电，则需要停电再重新上电

电脑端软件提示：下载编程进行中，数秒后提示成功

STC8C2K64S2-LQFP44 最小系统 示意图



注意：

1、ADC的外部参考电源管脚ADC_VRef+，必须接外部参考电源【电压：2.4V ~MCU-VCC】，浮空会有漏电流

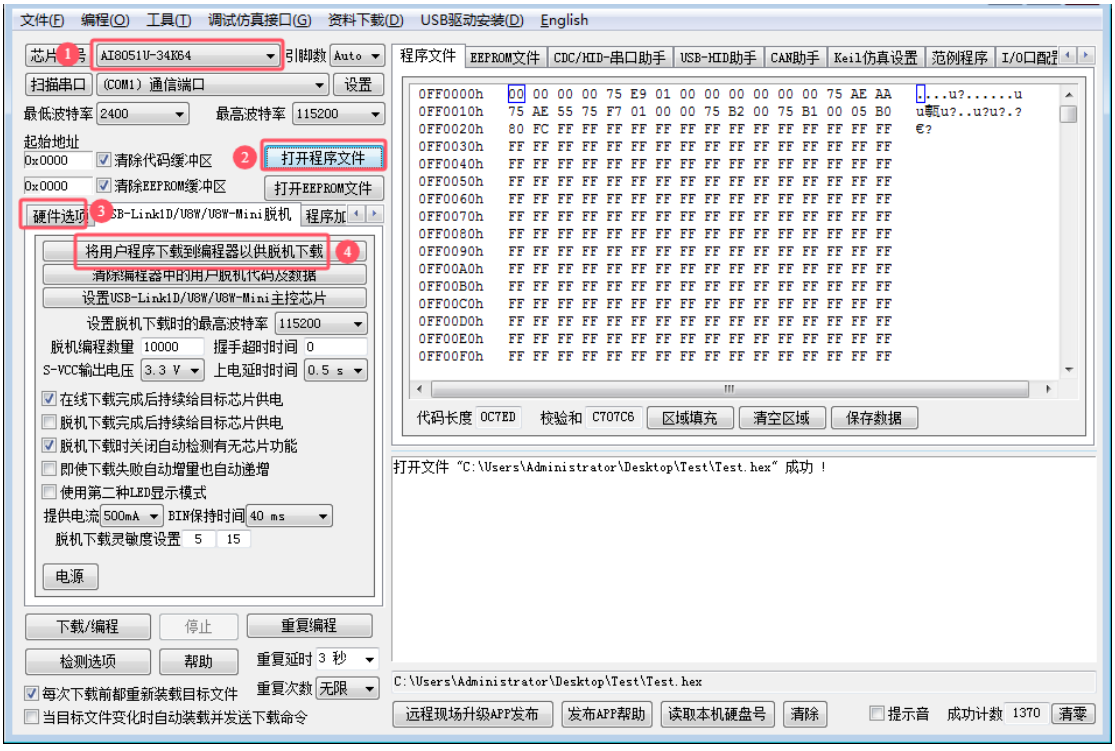
2、如不需要进行【软件模拟USB下载】，芯片上电/复位时P3.0/P3.1/P3.2 不可同时为低电平否则会进入【等待进入软件模拟USB下载模式】，可在 ISP烧录时指定取消“下次下载时使能USB-ISP模式”这个功能

2.2.17 USB-Link1D 支持 脱机下载 说明

脱机下载是指脱离电脑主机进行下载的一种方法，一般是使用下载控制芯片（又称脱机下载母片）进行控制。USB-Link1D 工具除了支持在线 ISP 下载，还支持脱机下载。USB-Link1D 工具使用外部的 22.1184MHz 晶振，可保证对目标芯片进行在线或脱机下载时，校准频率的精度。用户可将代码下载到 USB-Link1D 工具中，就可实现脱机下载。USB-Link1D 主控芯片如内部存储空间不够时，会将部分被下载的用户程序用加密的方式加密放部分到片外串行 Flash。

先将 USB-Link1D 工具使用 USB 线连接到电脑，然后按照下面的步骤进行脱机下载：

1、选择目标芯片的型号
2、打开需要下载的文件
3、设置硬件选项
4、进入“USB-Link1D 脱机”页面，点击“将用户程序下载到编程器以供脱机下载“按钮，即可将用户代码下载到 USB-Linik1D 工具中。下载完成后便使用 USB-Link1D 工具对目标芯片进行脱机下载了



2.2.18 USB-Link1D 支持 脱机下载，如何免烧录环节

大批量生产，如何省去专门的烧录人员，如何无烧录环节

大批量生产，你在将由 STC 的 MCU 作为主控芯片的控制板组装到设备里面之前，在你将 STC MCU 贴片到你的控制板完成之后，你必须测试你的控制板的好坏。不要说 100%，直通无问题，那是抬杠，不是搞生产，只要生产，就会虚焊，短路，部分元件贴错，部分元件采购错。

所以在贴片回来后，组装到外壳里面之前，你必须要测试，你的含有 STC MCU 控制板的好坏，好的去组装，坏的去维修抢救。

控制板的测试/不是烧录！

控制板的测试环节必须有，但烧录环节可以省！

大批量生产，必须要有方便控制板测试的测试架/下面接上我们的脱机烧录工具：

USB-Link1D / U8W-Mini / U8W，还要接上其他控制部分！

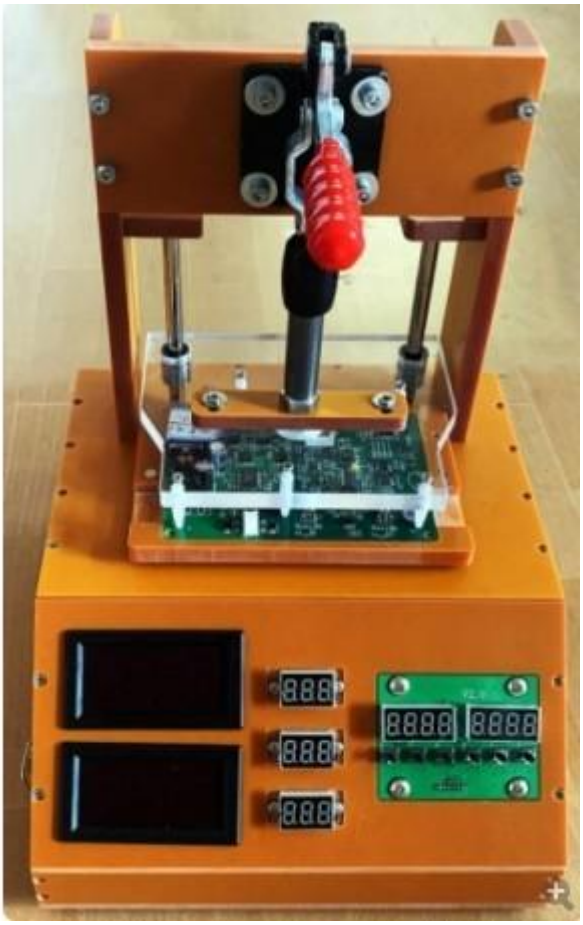
1、通过 USER-VCC、P3.0、P3.1、GND 连接，要工人每次都开电源

2、通过 S-VCC、P3.0、P3.1、GND 连接，不要你开电源，STC 的脱机工具给你自动供电

外面帮你做一个测试架的成本 500 元以下，就是有机玻璃，夹具，顶针。

1 个测试你控制板是否正常的工人管理 2 - 3 个 测试架



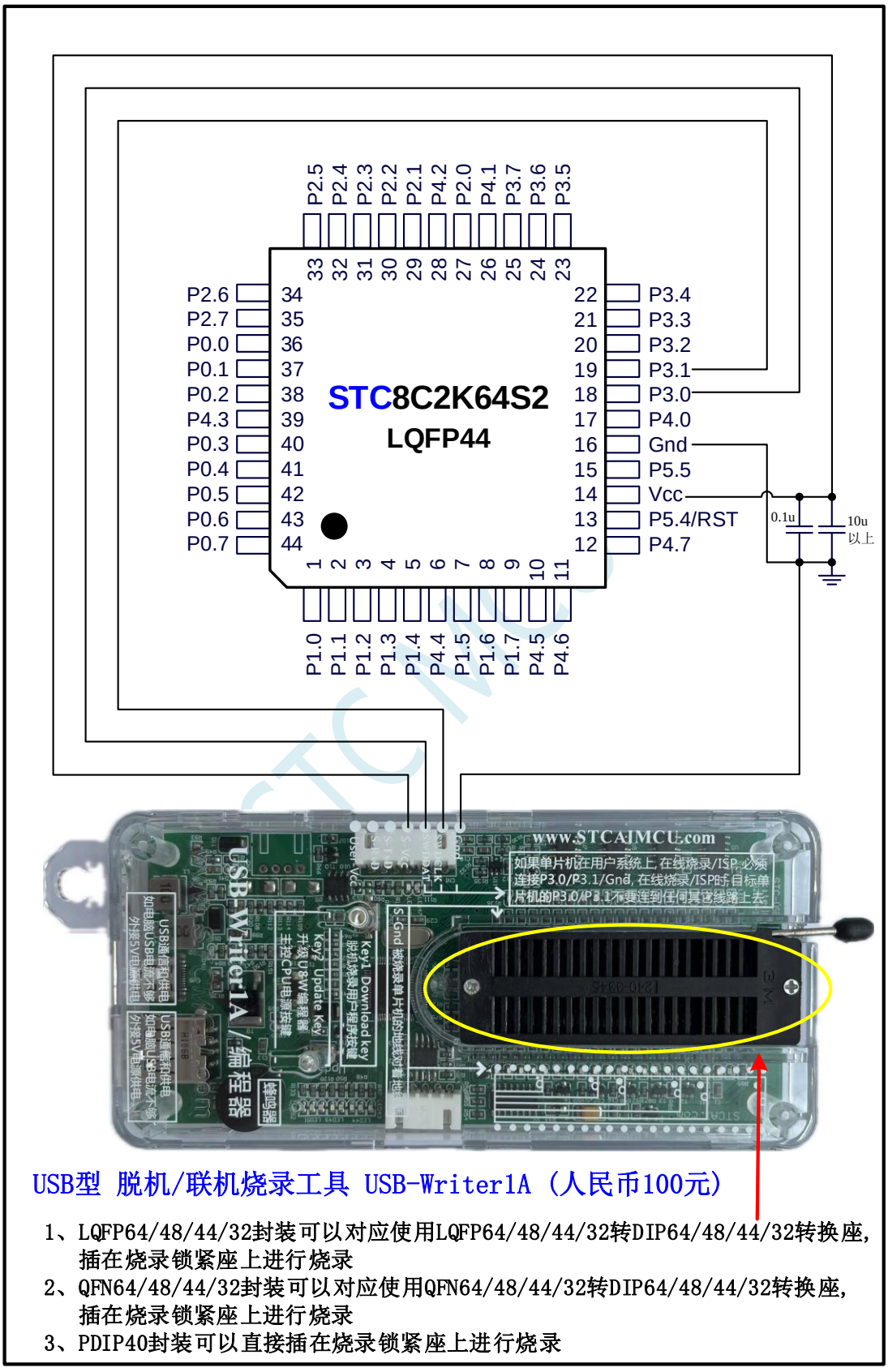


操作流程：

1、将你的 MCU 控制板 卡到测试架 1 上
2、将你的 MCU 控制板 卡到测试架 2 上，测试架 1 上的程序已烧录完成/感觉不到烧录时间
3、测试 测试架 1 上的 MCU 主控板功能是否正常，正常放到正常区，不正常，放到不正常区
4、给测试架 1 卡上新的未测试的无程序的控制板
5、测试 测试架 2 上的未测试控制板/程序不知何时早就不知不觉的烧好了，换新的未测试未烧录的控制板
6、循环步骤 3 到步骤 5
========不需要安排烧录人员

STC MCU

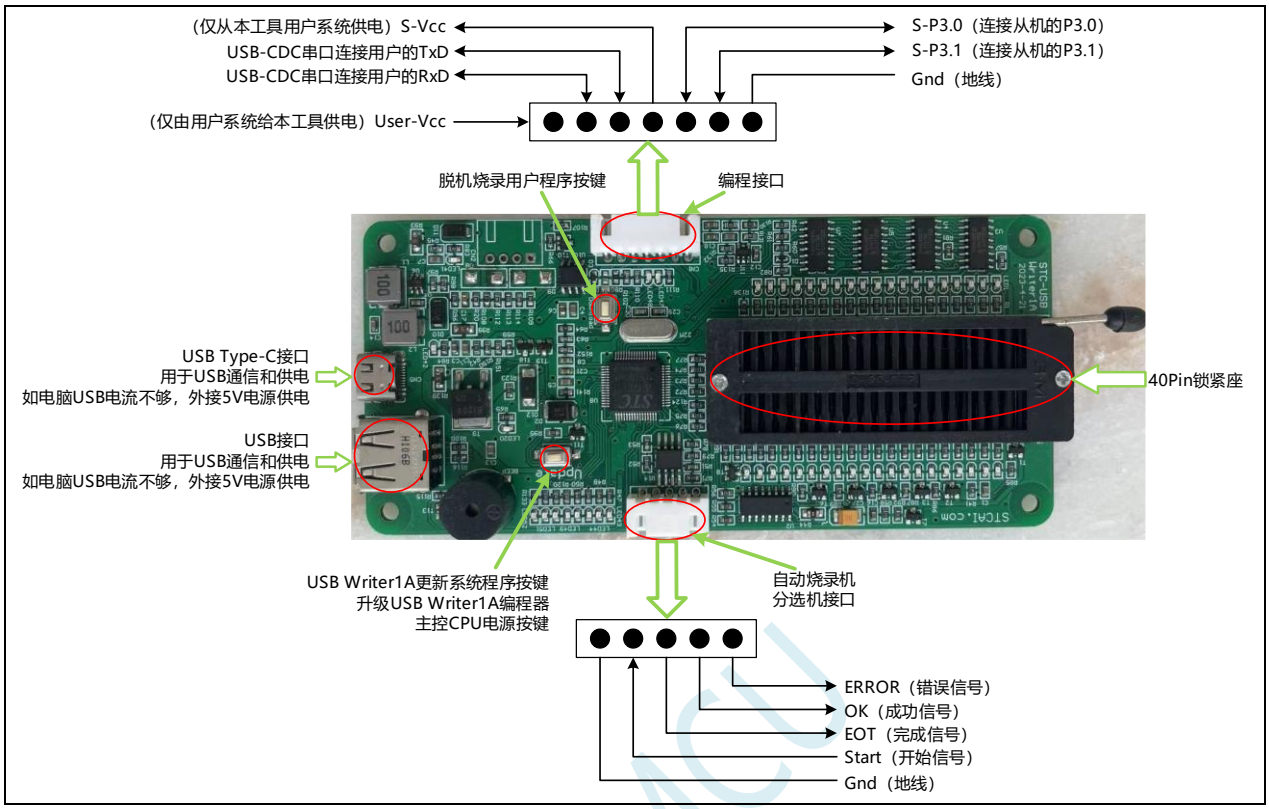
2.2.19 USB-Writer1A 编程器/烧录器 • 支持 • 插在 • 锁紧座上 • 烧录



USB型 脱机/联机烧录工具 USB-Writer1A（人民币100元）

1、LQFP64/48/44/32封装可以对应使用LQFP64/48/44/32转DIP64/48/44/32转换座，插在烧录锁紧座上进行烧录
2、QFN64/48/44/32封装可以对应使用QFN64/48/44/32转DIP64/48/44/32转换座，插在烧录锁紧座上进行烧录
3、PDIP40封装可以直接插在烧录锁紧座上进行烧录

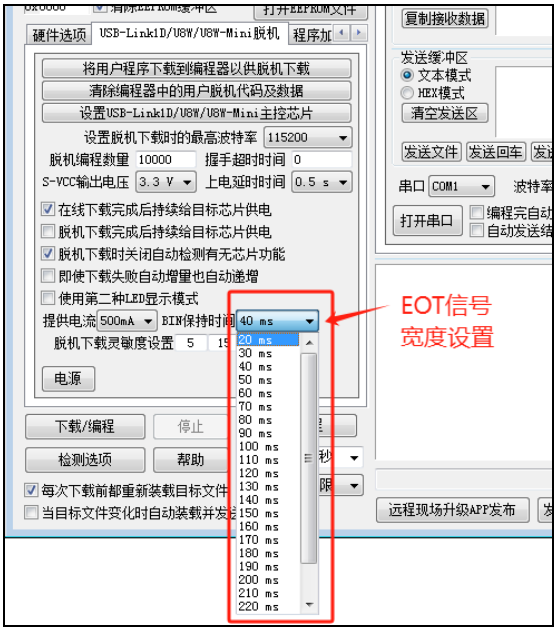
2.2.20 USB-Writer1A 支持 自动烧录机，通信协议和接口



自动烧录接口（分选机自动控制接口）协议：

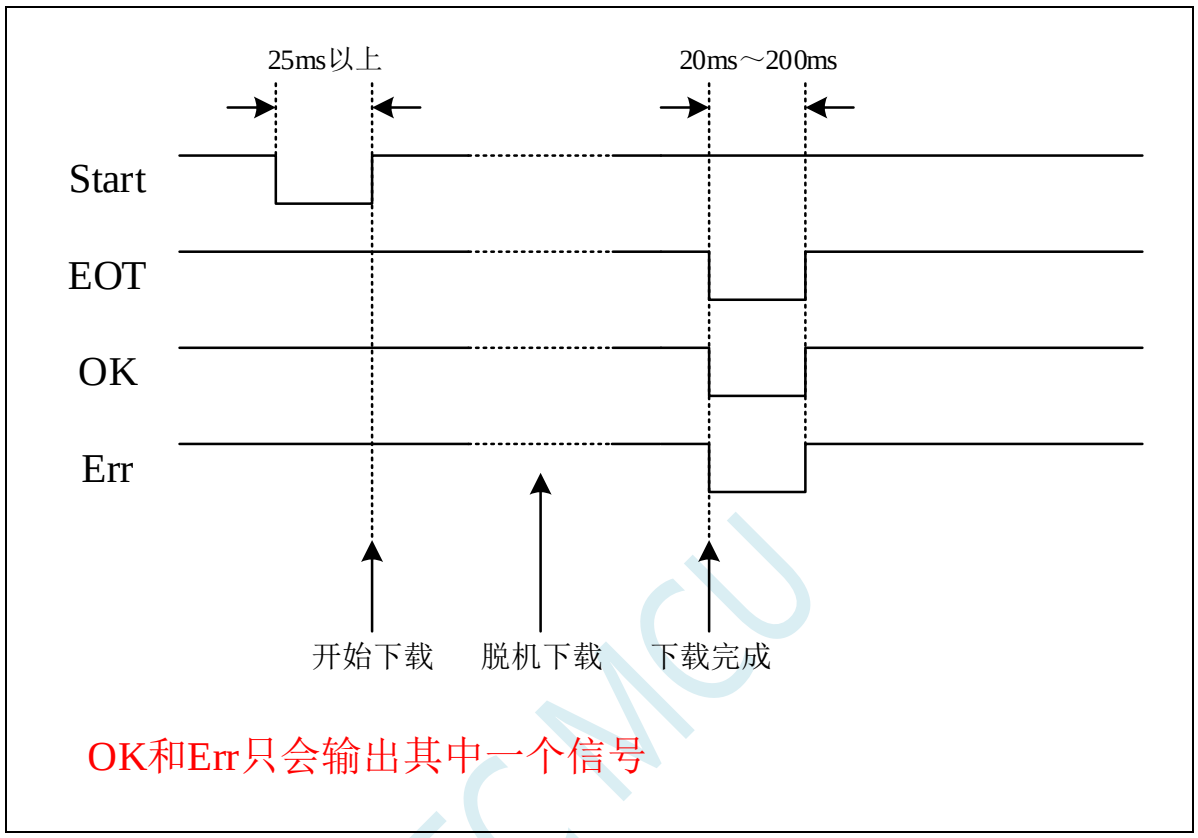
Start： 开始信号输入脚。从外部输入低电平信号触发开始脱机烧录，低电平必须维持 25 毫秒以上

EOT： 烧录完成信号输出脚。脱机烧录完成后，工具输出 20ms～250ms 的低电平 EOT 信号。电平宽度如下图所示的地方进行设置



OK： 良品信号输出脚。下载成功后工具从 OK 脚输出低电平信号，信号与 EOT 信号同步。

Err： 不良品信号输出脚。若下载失败，工具从 ERR 脚输出输出低电平信号，信号与 EOT 完成信号同步。



STCMCU

2.3 USB-Link1D 工具使用注意事项

2.3.1 工具接口说明

USB-Link1D 工具是 STC-USB Link1 的升级版、功能在 STC-USB Link1 的基础上增加了两个 STC-CDC 串口，可作为通用 USB 转串口使用。

工具 USB Link1D 的使用注意事项请参考附录章节



工具正面图



工具反面图

User-Vcc，外部给工具供电

S-RxD，第2组STC-CDC串口发送脚，接用户单片机串口的接收脚

S-TxD，第2组STC-CDC串口接收脚，接用户单片机串口的发送脚

S-Vcc，本工具给外部系统供电

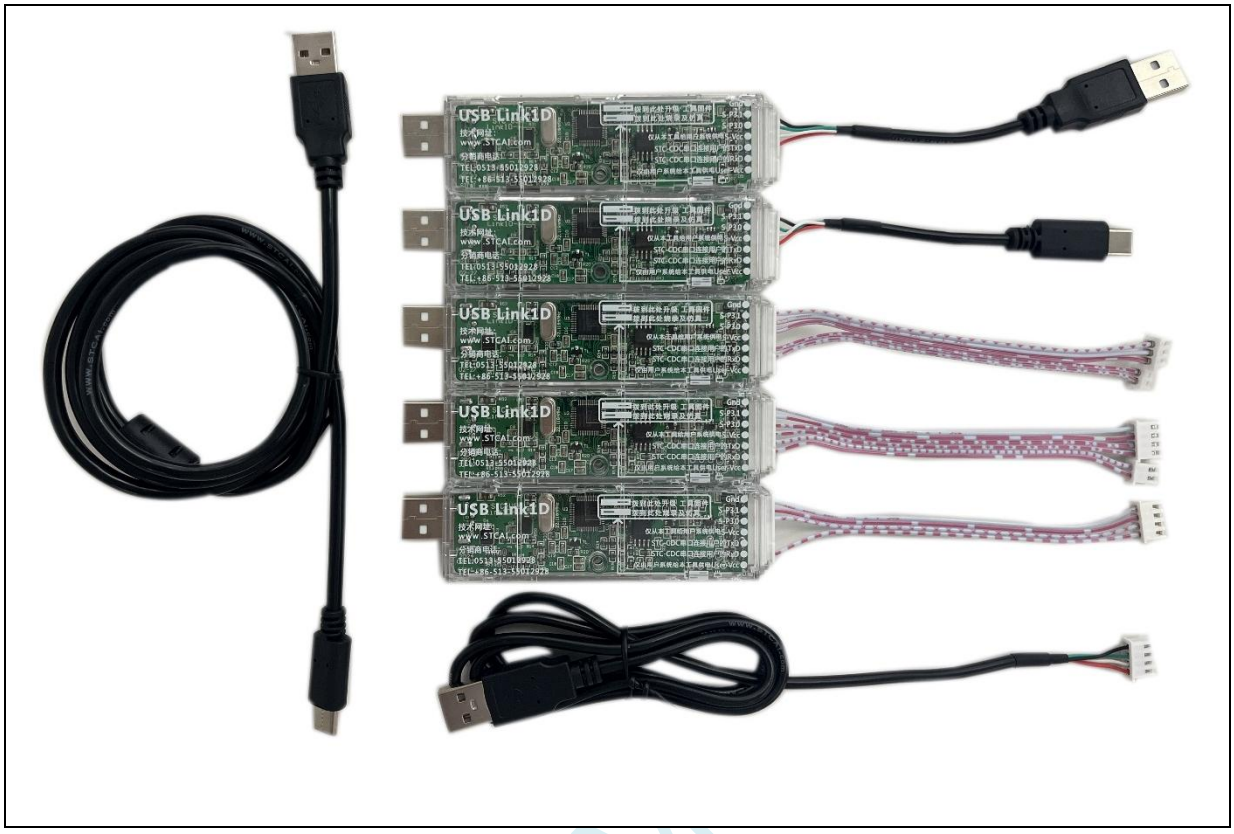
S-P3.0，Link1硬件仿真数据线、Link1进行ISP下载的串口发送脚、第1组STC-CDC串口发送脚

S-P3.1，Link1硬件仿真时钟线、Link1进行ISP下载的串口接收脚、第1组STC-CDC串口接收脚

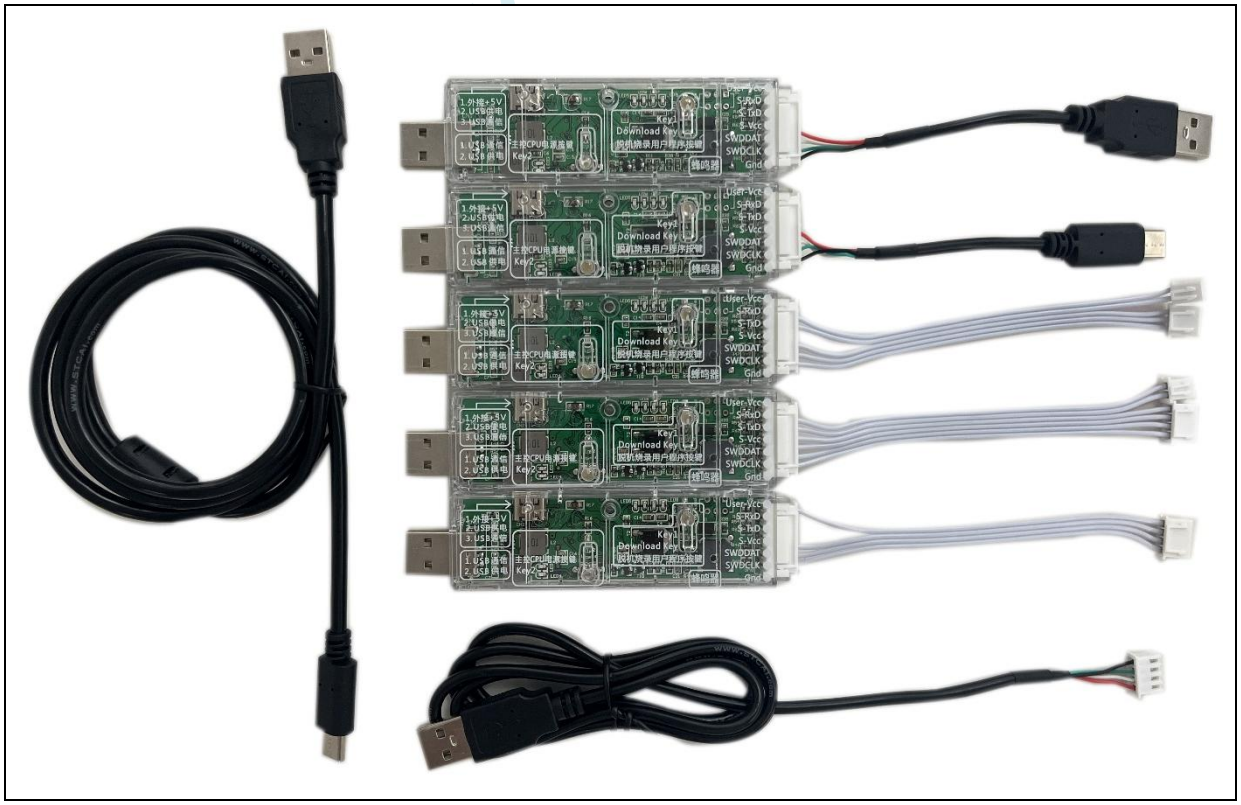
Gnd，地线

管脚编号	接口名称	接口功能
1	User-Vcc	仅由用户系统给本工具供电
2	S-RxD	第 2 组 STC-CDC 串口的发送脚，连接用户单片机串口的接收脚
3	S-TxD	第 2 组 STC-CDC 串口的接收脚，连接用户单片机串口的发送脚
4	S-Vcc	仅从本工具给用户系统供电
5	S-P3.0	使用 Link1D 进行 ISP 下载时的串口发送脚，连接目标单片机的 P3.0
		使用 Link1D 进行 SWD 硬件仿真时的数据脚，连接目标单片机的 SWDDAT
		第 1 组 STC-CDC 串口的发送脚，连接用户单片机串口的接收脚
6	S-P3.1	使用 Link1D 进行 ISP 下载时的串口接收脚，连接目标单片机的 P3.1
		使用 Link1D 进行 SWD 硬件仿真时的时钟脚，连接目标单片机的 SWDCLK
		第 1 组 STC-CDC 串口的接收脚，连接用户单片机串口的发送脚
7	Gnd	地线

2.3.2 附送的各种强大的人性化配线图片及使用说明



正面



反面

USB-Link1D 各种豪华配线的应用场景介绍



USB转双串口/TTL，连接线
不从本工具供电

连接目标用户系统的Gnd
连接目标用户系统的P3.1或串口发送脚
连接目标用户系统的P3.0或串口接收脚
不从本工具给目标系统供电，目标系统自己供电
连接目标用户系统的第n组串口的TxDn
连接目标用户系统的第n组串口的RxDn

这种连接方式为本工具和目标系统各自独立供电，ISP下载需要手动给目标系统停电再上电。
当目标系统的耗电大于300mA时，在线下载、脱机下载、在线仿真，可选这种接线方式。



USB转双串口/TTL，连接线
从本工具供电

连接目标用户系统的Gnd
连接目标用户系统的P3.1或串口发送脚
连接目标用户系统的P3.0或串口接收脚
从本工具给目标用户系统供电，300mA以内
连接目标用户系统的第n组串口的TxDn
连接目标用户系统的第n组串口的RxDn

这种连接方式为本工具给目标系统自动 停电/上电 供电，ISP下载不需要手动停电/上电。
当目标系统电流不超过300mA时，在线下载、脱机下载、在线仿真，可选择这种线方式。



USB转串口/TTL，脱机烧录连接线
从用户系统给本工具供电

连接目标用户系统的Gnd
连接目标用户系统的P3.1
连接目标用户系统的P3.0
目标用户系统给工具供电

这种连接方式为常用的脱机下载连接方式，从用户的目标系统给本脱机工具供电。
如果用户系统电流大于300mA，建议使用这种连接方式。用户系统上电之后自动脱机下载程序，下载成功之后，给用户系统停电，换下一个待烧录用户程序的新的用户系统，再上电，脱机下载程序，重复如上过程即为常见的【批量生产脱机烧录】。
特别注意：用户系统给本工具供电的电压不能超过5.3V



USB转串口/TTL，专门定制的TypeC接口线
但此处为TTL串口，不是USB

连接目标用户系统的Gnd
连接目标用户系统的P3.1或串口发送脚
连接目标用户系统的P3.0或串口接收脚
从本工具给目标用户系统供电，300mmA以内



USB转串口/TTL，专门定制的TypeA接口线
但此处为TTL串口，不是USB

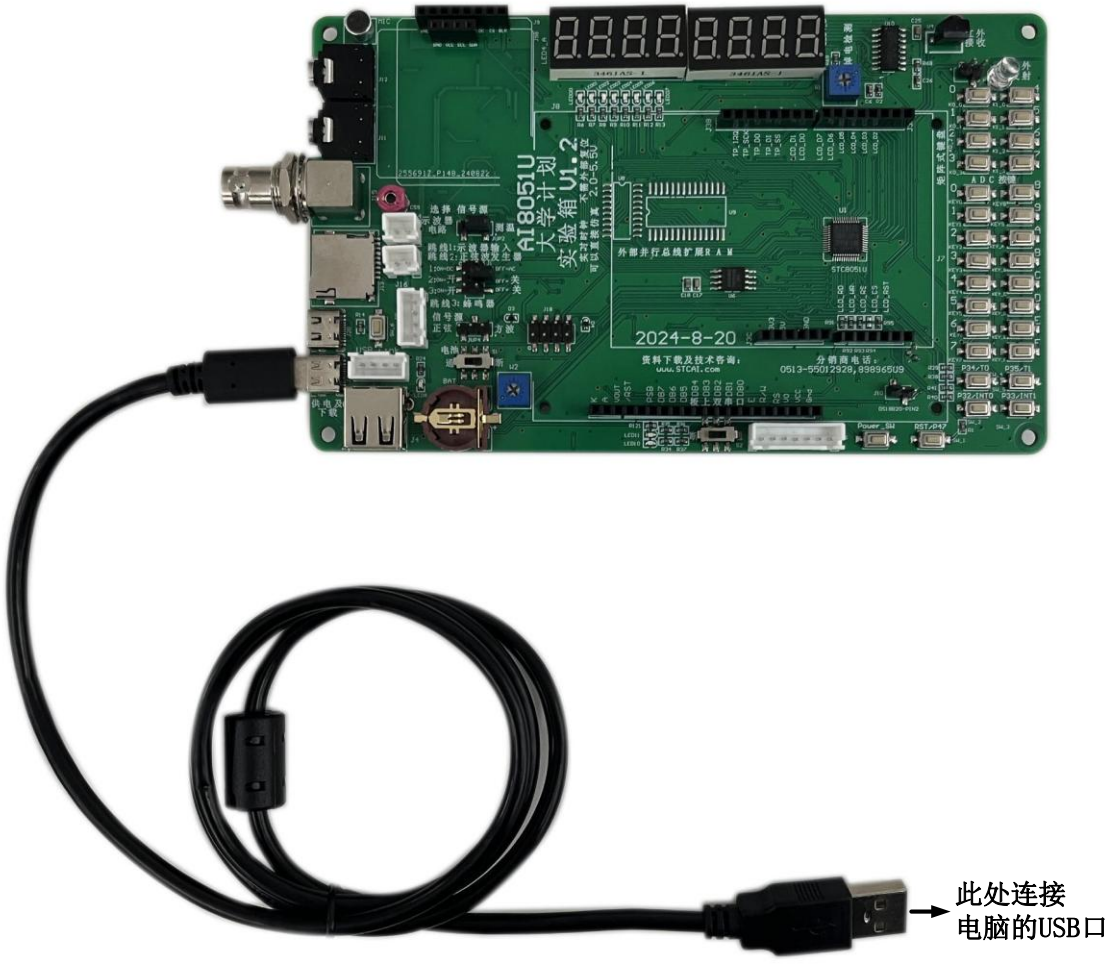
连接目标用户系统的Gnd
连接目标用户系统的P3.1或串口发送脚
连接目标用户系统的P3.0或串口接收脚
从本工具给目标用户系统供电，300mmA以内

USB直接下载连接示意图

使用 USB-TypeA 到 USB-TypeC 线



这种一端为 USB-TypeA ，另一端为 USB-TypeC 的标准线，STCAI.com 也会提供这种连接线，方便预留了 USB-TypeC 接口的用户系统，进行硬件USB直接下载。

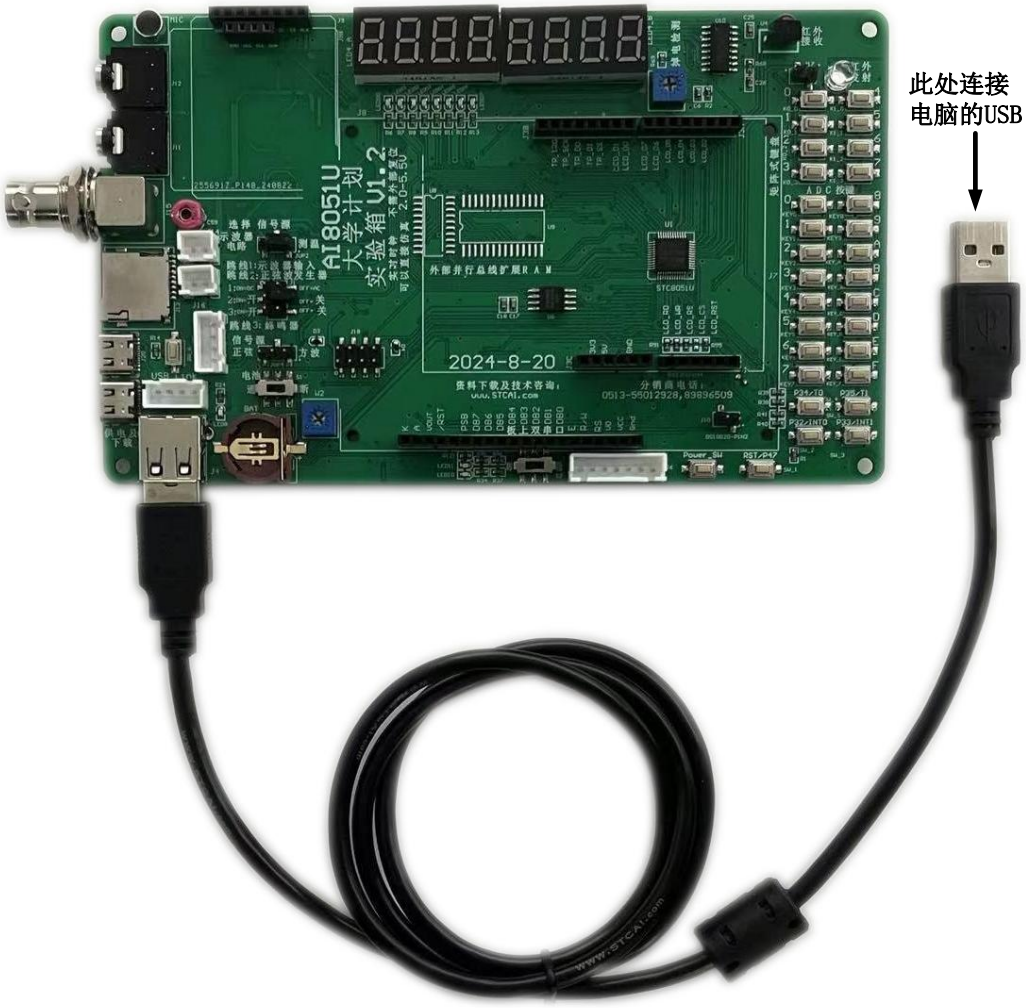


USB 直接下载连接示意图

使用 USB-TypeA 到 USB-TypeA 线



这种两端同为 USB-TypeA 接口的标准线，
方便预留了 USB-TypeA 接口的用户系统，进行硬件USB直接下载

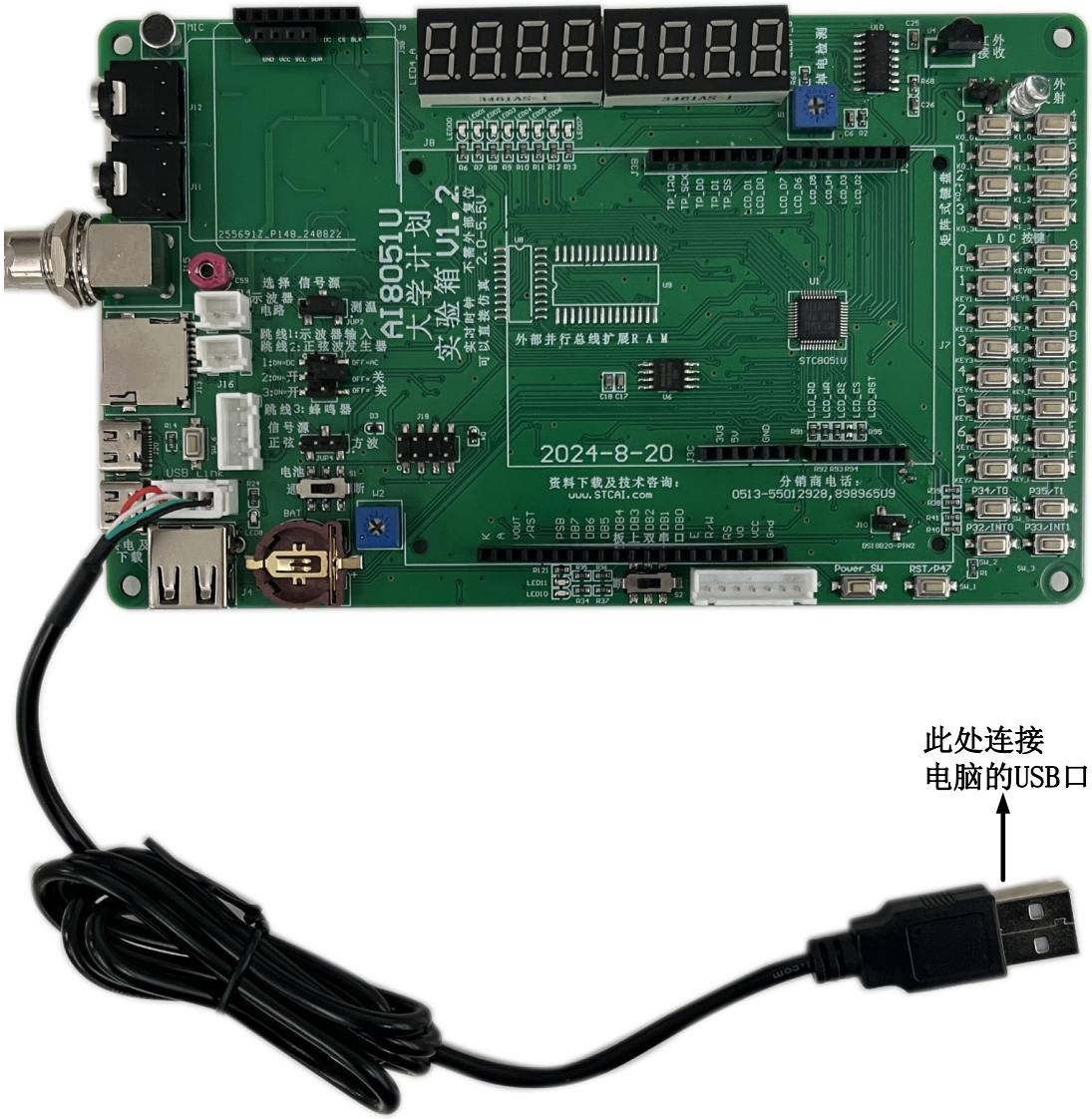


USB 直接下载连接示意图

使用 USB-TypeA 到流行的 2.54mm 间距通用插座



STCAI.com提供这种连接线，方便对只预留了2.54mm间距的通用插座目标系统芯片进行硬件USB直接下载



2.3.3 USB-Link1D 实际应用

1、使用 USB-Link1D 工具对 STC8C 系列单片机进行 SWD 硬件仿真

按照如下图所示的方式将工具的 S-Vcc、S-P3.0、S-P3.1、GND 分别与目标单片机的 M-Vcc、P3.0（SWDDAT）、P3.1（SWDCLK）、GND 相连接，然后参考前面章节中硬件仿真的步骤和设置即可进行 SWD 硬件仿真

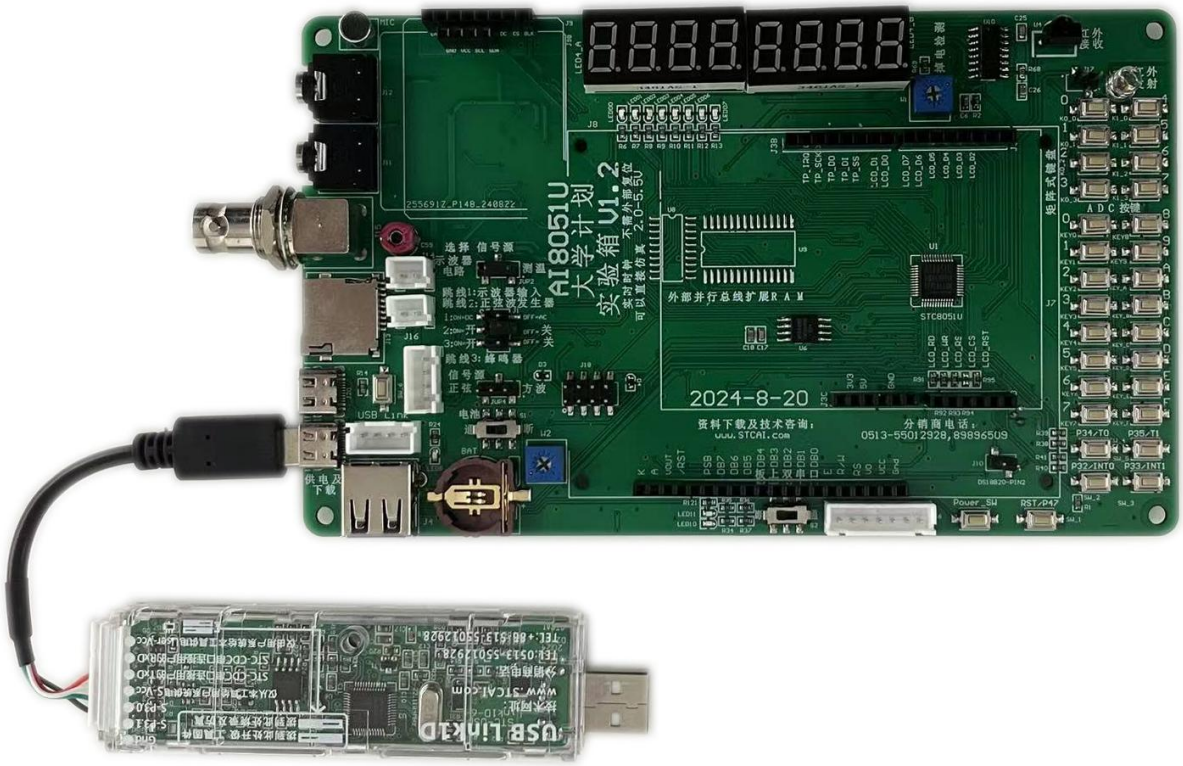


使用 USB-Link1D 的 USB转串口/TTL，用单排2.54mm间距插头连接目标芯片系统，串口下载连接示意图-反面



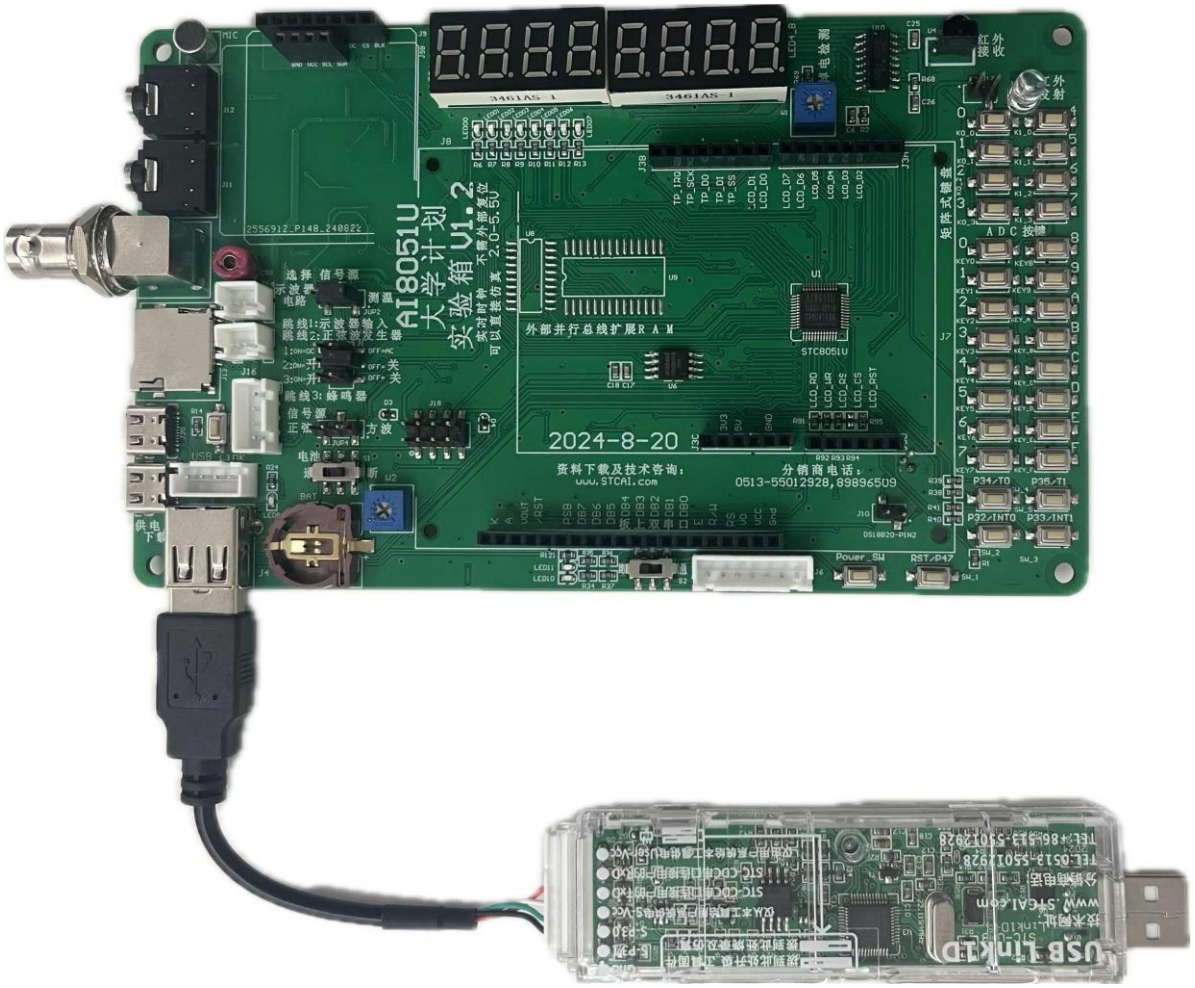
使用USB-Link1D的USB转串口对目标系统进行串口下载

注意：此处连接目标系统的 USB-TypeC接口 是 USB转串口/TTL，
对目标系统芯片 串口/TTL 下载连接示意图



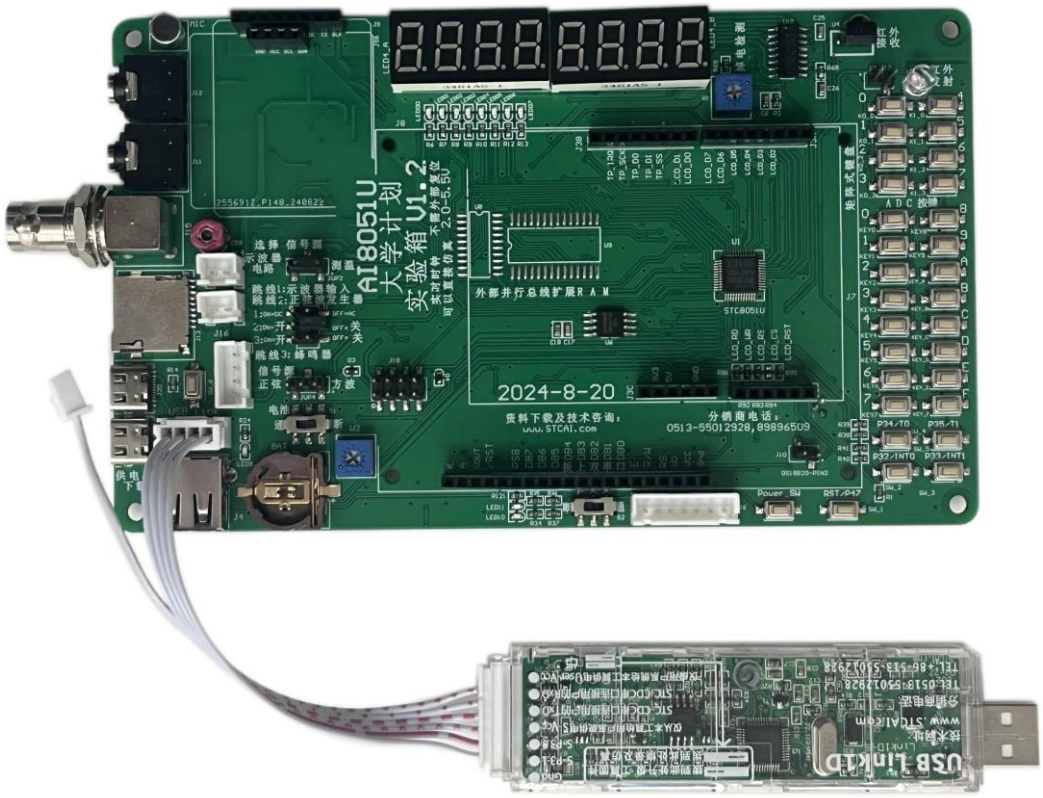
使用USB-Link1D的USB转串口对目标系统进行串口下载

注意：此处连接目标系统的 USB-TypeA接口 是 USB转串口/TTL,
对目标系统芯片 串口/TTL 下载连接示意图



使用USB-Link1D的USB转串口对目标系统进行串口下载

注意：此处连接目标系统的 普通四芯单排插座接口 是 USB转串口/TTL，
对目标系统芯片 串口/TTL 下载连接示意图

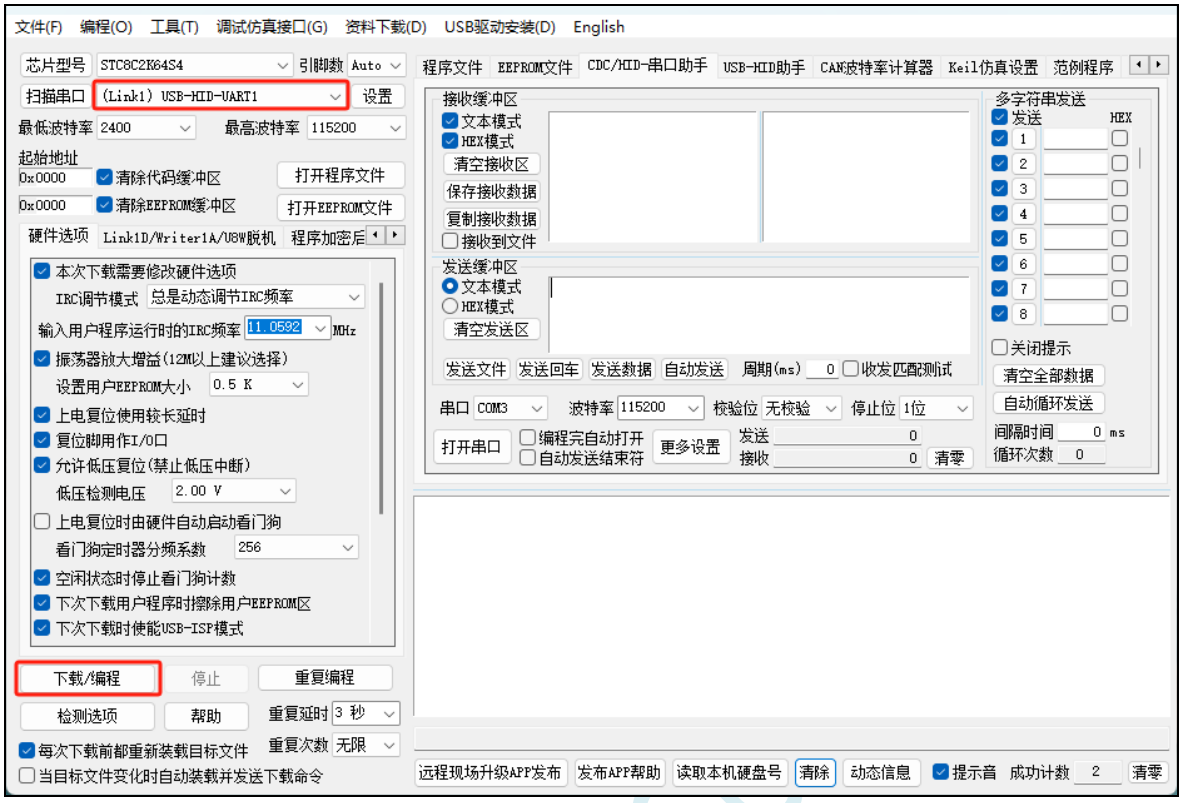


2、使用 USB-Link1D 工具对 STC15 和 STC8 系列进行串口仿真

工具的 S-Vcc、S-P3.0、S-P3.1、GND 分别与目标单片机的 M-Vcc、P3.0（RxD）、P3.1（TxD）、GND 相连接，然后 Keil 仿真设置中选择 STC-CDC1 所对应的串口号，然后参考 STC15/STC8 系列数据手册中的直接串口仿真章节中仿真的步骤和设置，即可进行串口仿真

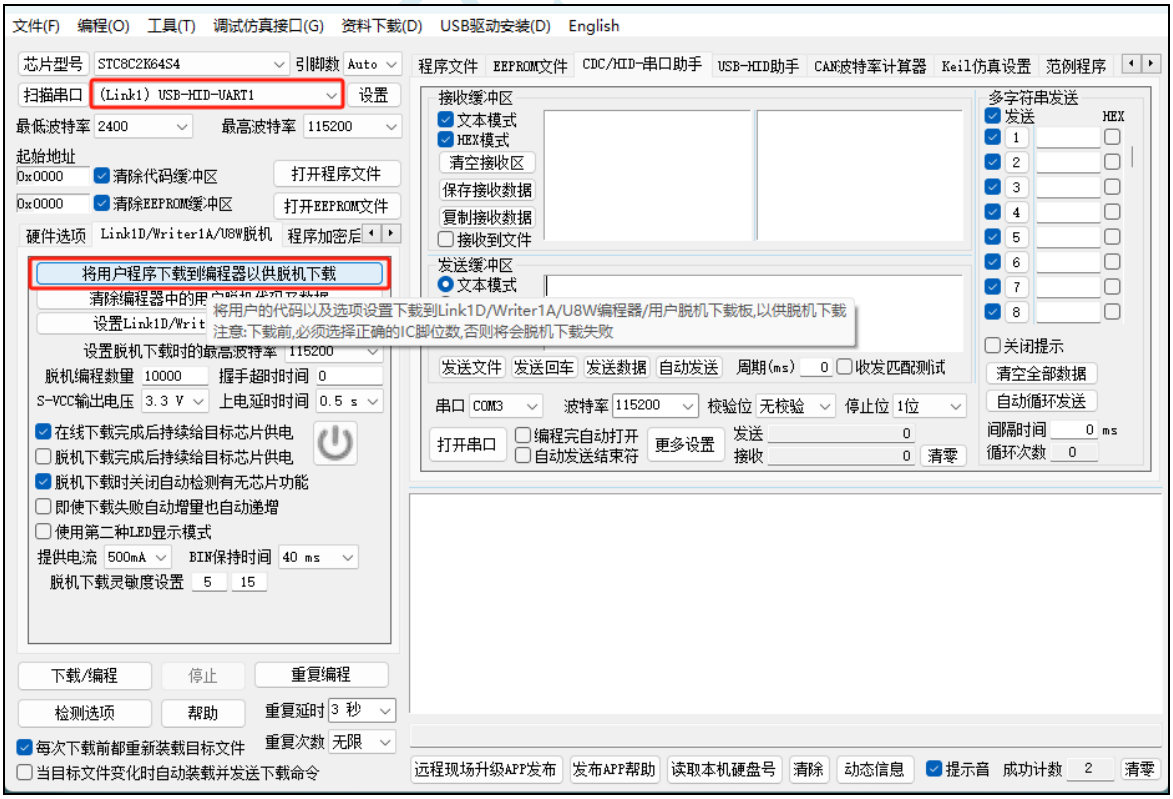
3、使用 USB-Link1D 工具对 STC 全系列单片机进行 ISP 在线下载

工具的 S-Vcc、S-P3.0、S-P3.1、GND 分别与目标单片机的 M-Vcc、P3.0（RxD）、P3.1（TxD）、GND 相连接，在 AIapp-ISP 下载软件中的串口号选择“(Link1) USB-HID-UART1”，打开程序文件以及设置相关硬件选项，然后点击“下载/编程”按钮即可进行 ISP 在线下载



4、使用 USB-Link1D 工具对 STC 全系列单片机进行 ISP 脱机下载

在 AIapp-ISP 下载软件中的串口号选择“(Link1) USB-HID-UART1”，打开程序文件以及设置相关硬件选项，后点击“Link1D/Writer1A/U8W 脱机”页面中的“将用户程序下载到编程器以供脱机下载”按钮，将用户代码和相关设置下载到 USB-Link1D 工具上的存储器中。



将工具的 S-Vcc、S-P3.0、S-P3.1、GND 分别与目标单片机的 M-Vcc、P3.0（RxD）、P3.1（TxD）、GND 相连接，然后按下工具上的“Key1”按键即可对目标芯片进行脱机下载（即不需要 PC 端的控

制，独立进行 ISP 下载）

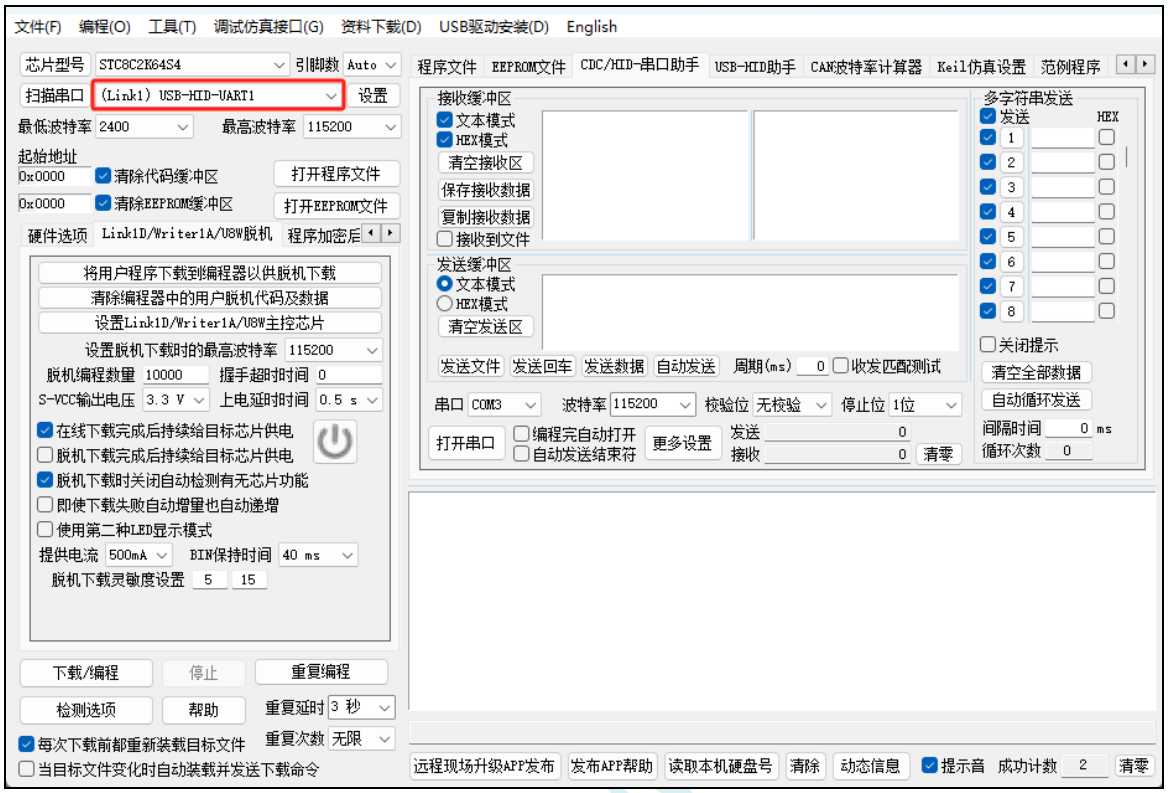
5、USB-Link1D 工具当作通用 USB 专串口工具使用

USB-Link1D 工具提供了两个 STC-CDC 串口，可作为通用 USB 专串口工具使用，由于第一个串口 CDC1 与硬件仿真、ISP 下载共用 S-P3.0 和 S-P3.1 端口，而第二个串口 CDC2 是独立串口，所以建议 S-P3.0 和 S-P3.1 作为仿真和 ISP 下载使用，当需要使用通用 USB 专串口工具时，使用 S-TxD 和 S-RxD 所对应的 CDC2。（注：在没有使用冲突的情况下，CDC1 和 CDC2 均可各自独立的当作通用 USB 专串口工具使用）

STC MCU

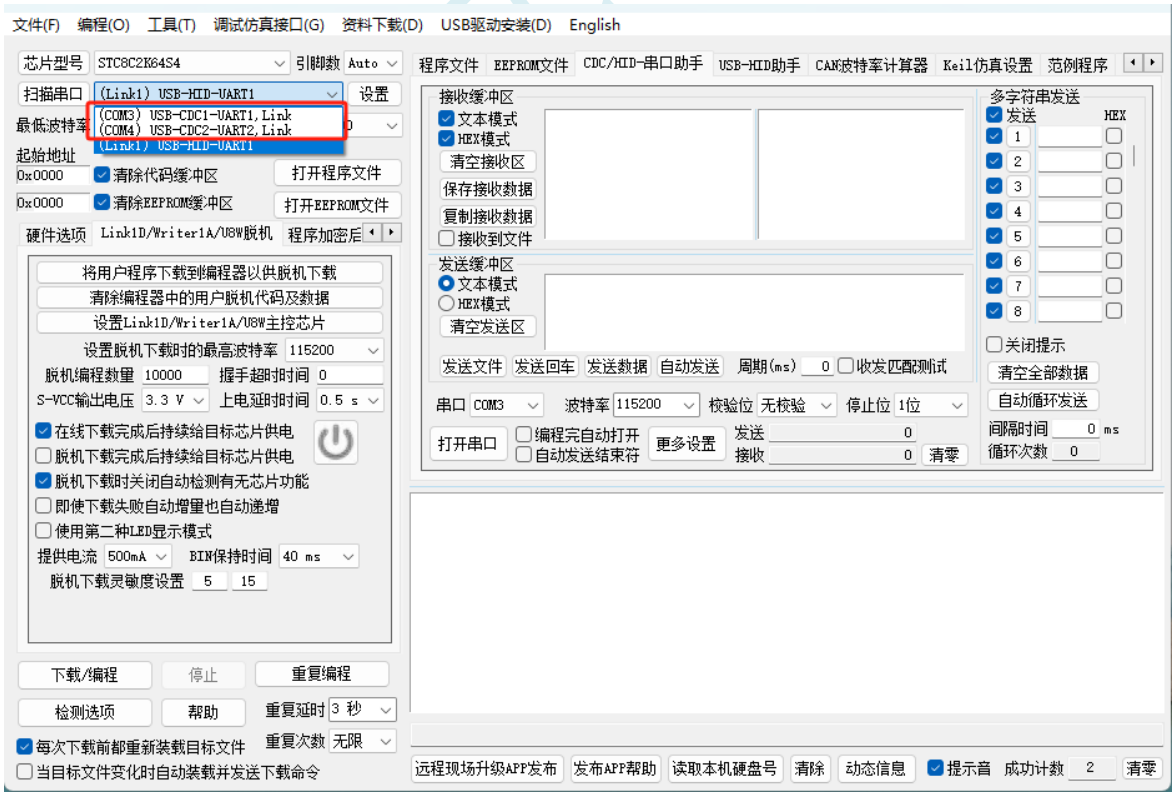
2.3.4 USB-Link1D 插上电脑并正常识别到后的显示

USB-Link1D 工具在出厂时，主控芯片内已烧录了 USB-Link1D 的控制程序。正常情况下，工具连接到电脑后，在 AIapp-ISP 下载软件中会立即识别出“(Link1) USB-HID-UART1”，如下图所示



正确识别后，即可使用 USB-Link1D 进行在线 ISP 下载或者脱机 ISP 下载。

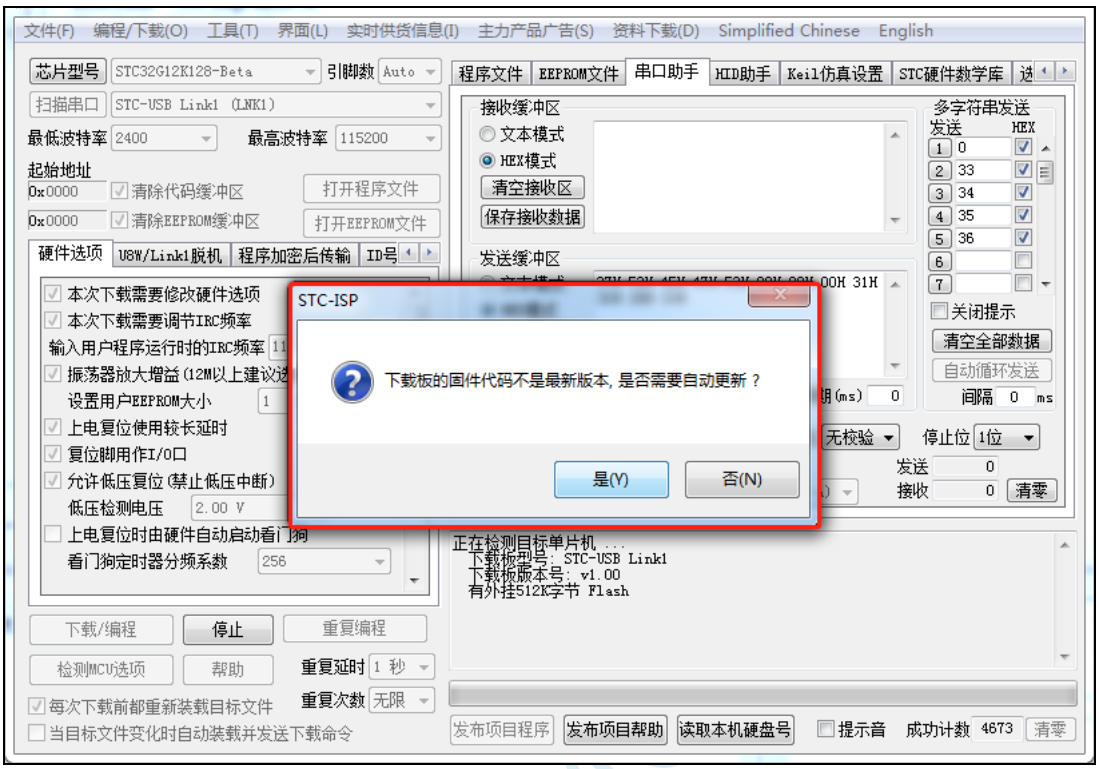
在驱动安装成功后，还会自动识别出两个 STC-CDC 串口，如下图所示：



可以当作通用 USB 转串口工具使用。

2.3.5　如电脑端新版本 ISP 软件提示 USB-Link1D 工具固件需升级

当使用工具进行 ISP 下载时，软件弹出如下画面，表示工具的固件需要升级

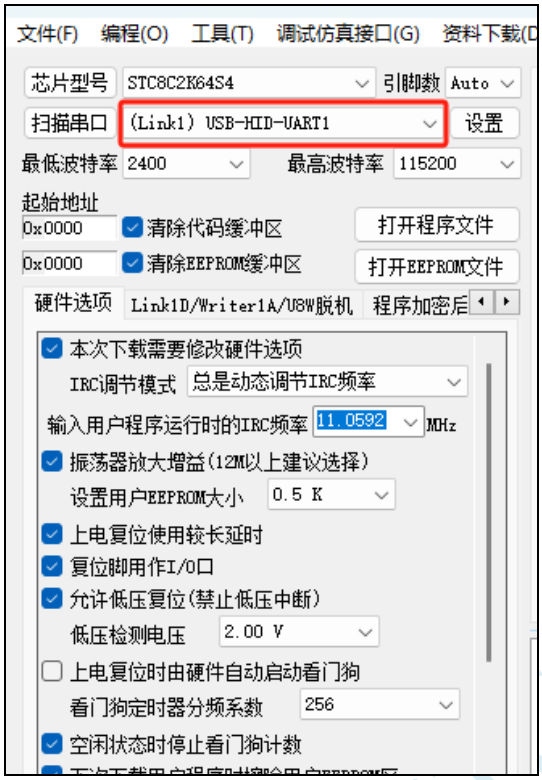


点击“是”按钮，工具便会自动开始升级。

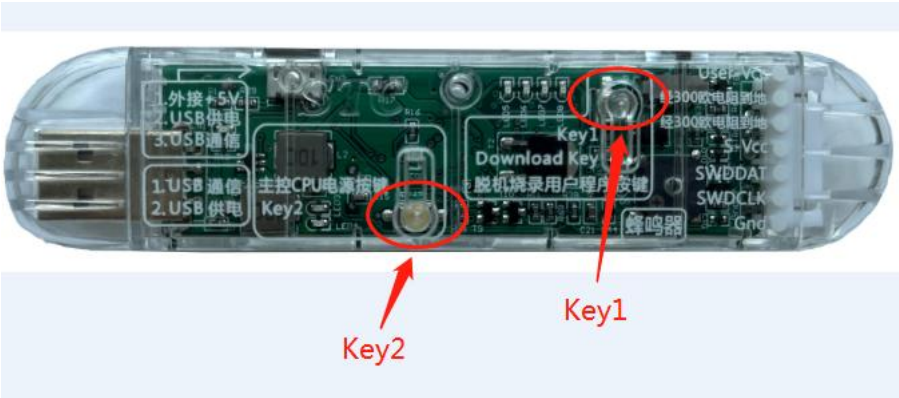
2.3.6　制作 USB-Link1D 主控芯片

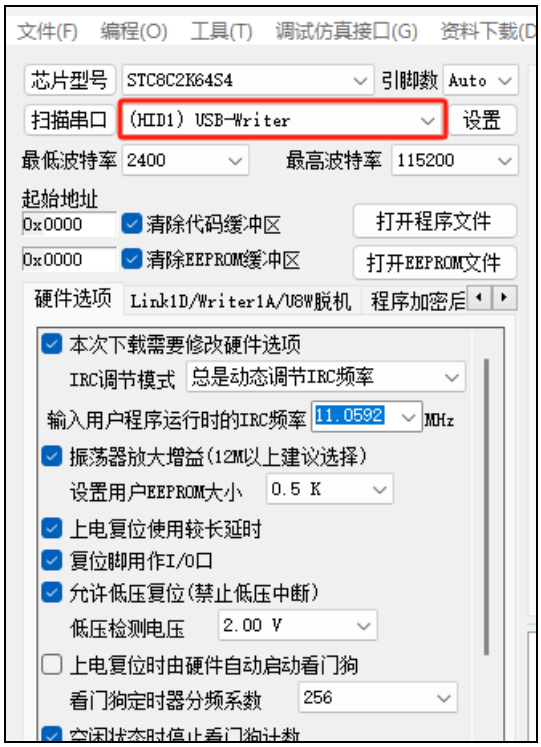
USB-Link1D 工具在出厂时，我公司的操作人员已经将主控芯片制作完成，所以客户拿到工具后不需要自己手动再次制作 USB-Link1D 工具的主控芯片。只有在客户将工具的主控芯片更换为一颗全新的芯片才需要进行下面的步骤。

1、将 USB-Link1D 工具插入电脑的 USB 口，如果 AIapp-ISP 下载软件的端口列表中没有显示出“(Link1) USB-HID-UART1”的设备（如下图），则说明工具的主控芯片为空片，需要继续接下来的步骤

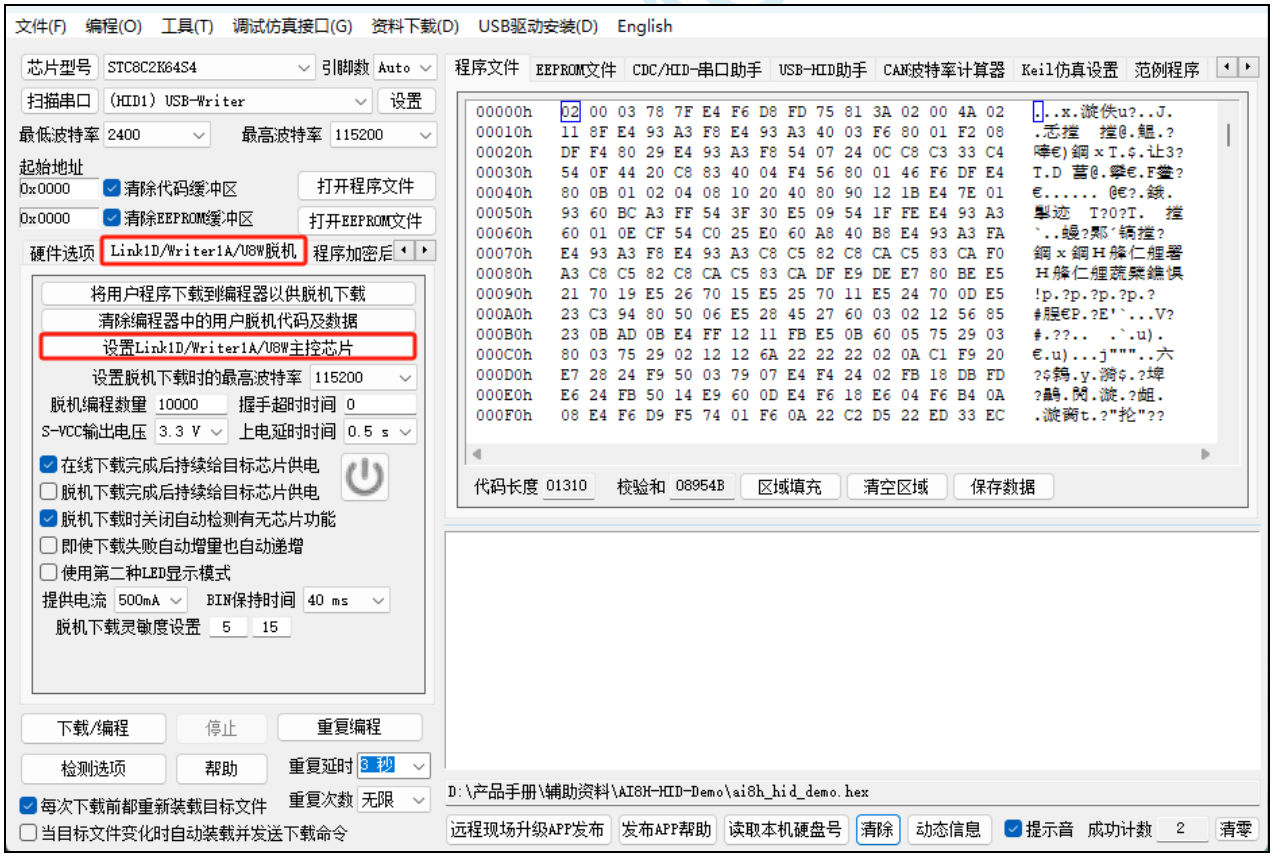


2、按住工具的“Key1”按键不要松开，再轻按一下“Key2”按键后松开 Key2 按键（此时需要保持 Key1 按键一直处于按下状态），等待 AIapp-ISP 下载软件的端口列表中显示出“(HID1) USB-Writer”的设备（如下图）时，再松开 Key1 按键

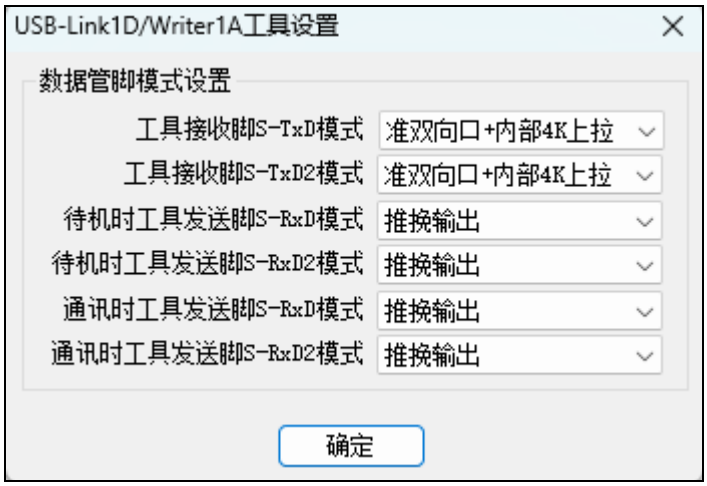




3、点击 AIapp-ISP 下载软件中“Link1D/Writer1A/U8W 脱机”页面的“设置 Link1D/Writer1A/U8W 主控芯片”按钮

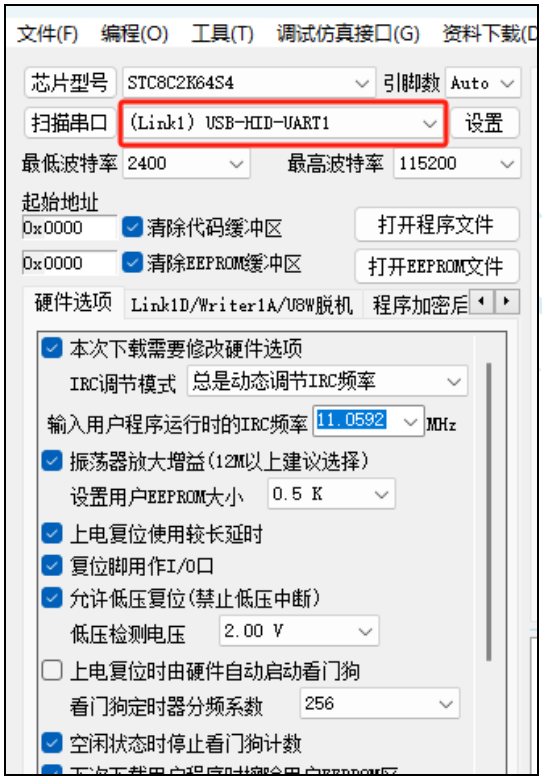


4、弹出的 Link1D 工具设置窗口中可以根据实际需要进行设置，若无特殊需要可保持默认选项



5、接下来软件会自动开始制作 USB-Link1D 工具的主控芯片。制作完成后，建议将工具的 USB 线重新插拔一次（特别是对一颗全新的芯片第一次制作主控芯片时必须重新插拔一次）。

当主控芯片制作完成并再次重新插入电脑的 USB 口时，AIapp-ISP 下载软件的端口列表中显示出“(Link1) USB-HID-UART1”的设备（如下图），则说明工具的主控芯片制作成功。



2.4 ISP 下载相关硬件选项的说明

硬件选项	选项何时生效
☑ 选择使用内部IRC时钟(不选为外部时钟)	需要重新上电才生效
输入用户程序运行时的IRC频率 11.0592 MHz	动态调整，立即生效
☑ 振荡器放大增益(12M以上建议选择)	需要重新上电才生效
☑ 使用快速下载模式	只与本次下载有关
设置用户EEPROM大小 0.5 K	需要重新上电才生效
☐ 下次冷启动时,P3.2/P3.3为0/0才可下载程序	下次下载时有效
☑ 上电复位使用较长延时	需要重新上电才生效
☑ 复位脚用作I/O口	需要重新上电才生效
☑ 允许低压复位(禁止低压中断)	需要重新上电才生效
低压检测电压 2.20 V	需要重新上电才生效
☑ 低压时禁止EEPROM操作	需要重新上电才生效
☐ 上电复位时由硬件自动启动看门狗 看门狗定时器分频系数 256 ☑ 空闲状态时停止看门狗计数	需要重新上电才生效
☑ 下次下载用户程序时擦除用户EEPROM区	下次下载时有效
☐ 在程序区的结束处添加重要测试参数	每次下载时一并写入

需要重新上电才生效：选项修改后，目标芯片需要断电一次（停电），重新再上电，新的设置才生效

动态调整，立即生效：本次 ISP 下载有效

只与本次下载有关：此选项只与本次 ISP 下载有关，不影响下一次下载

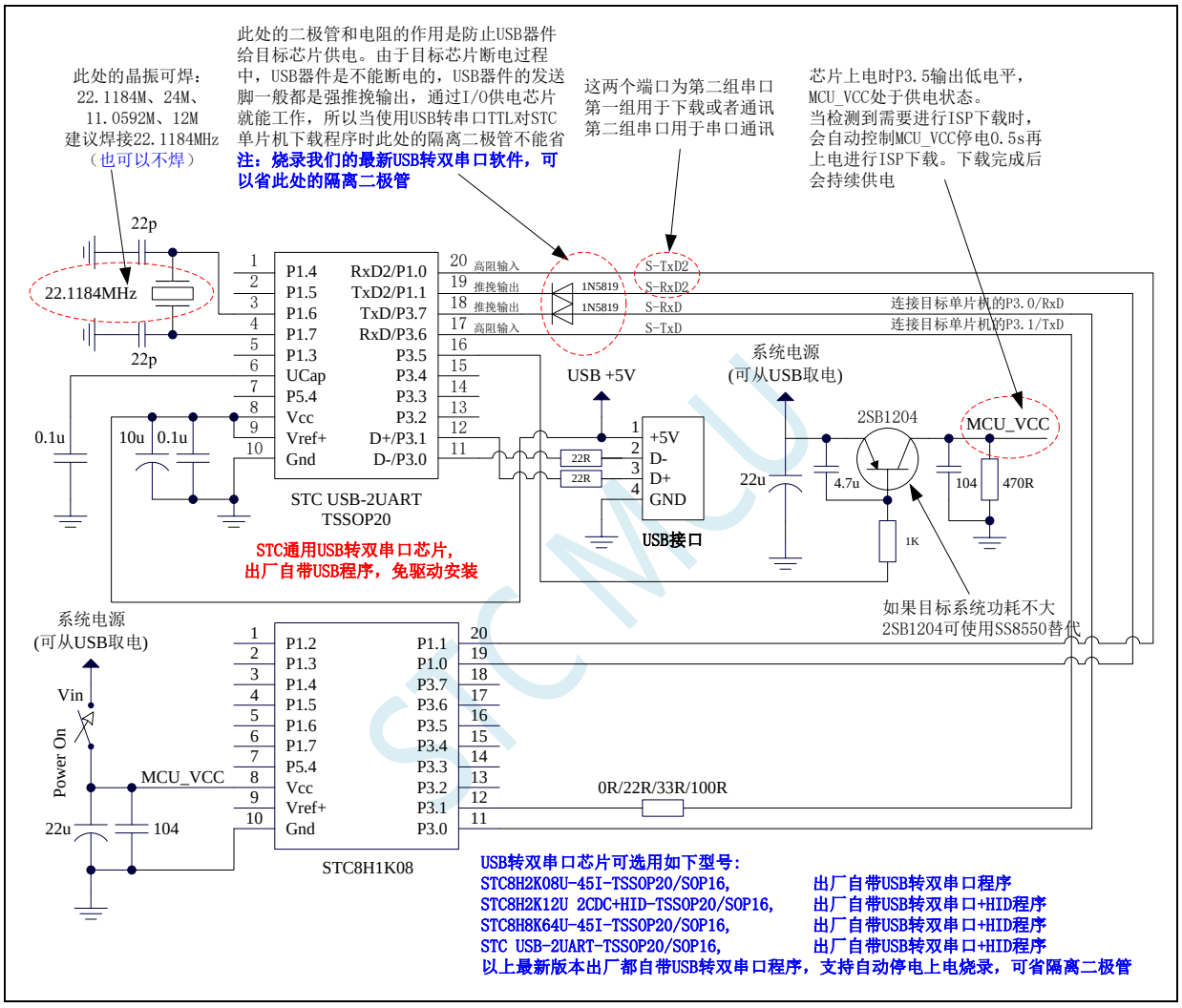
下次下载时有效：选项修改后，下次下载时才生效，修改对本次 ISP 下载无效

每次下载时一并写入：选择此选项后，在本次下载时将附加的数据一并写入，与下次下载无关

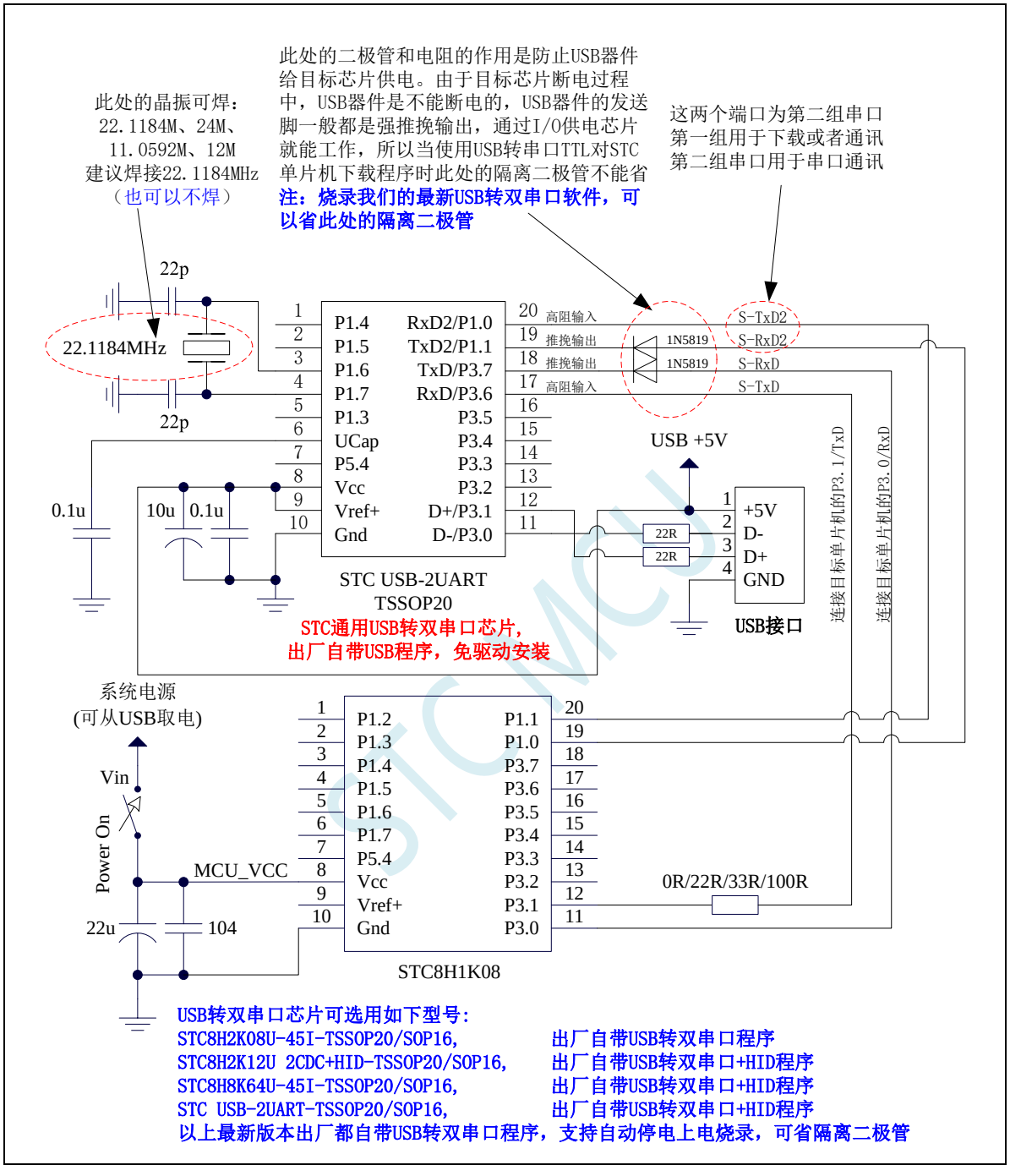
STC MCU

2.5 通用 USB 转双串口芯片：STC USB-2UART, TSSOP20/SOP16

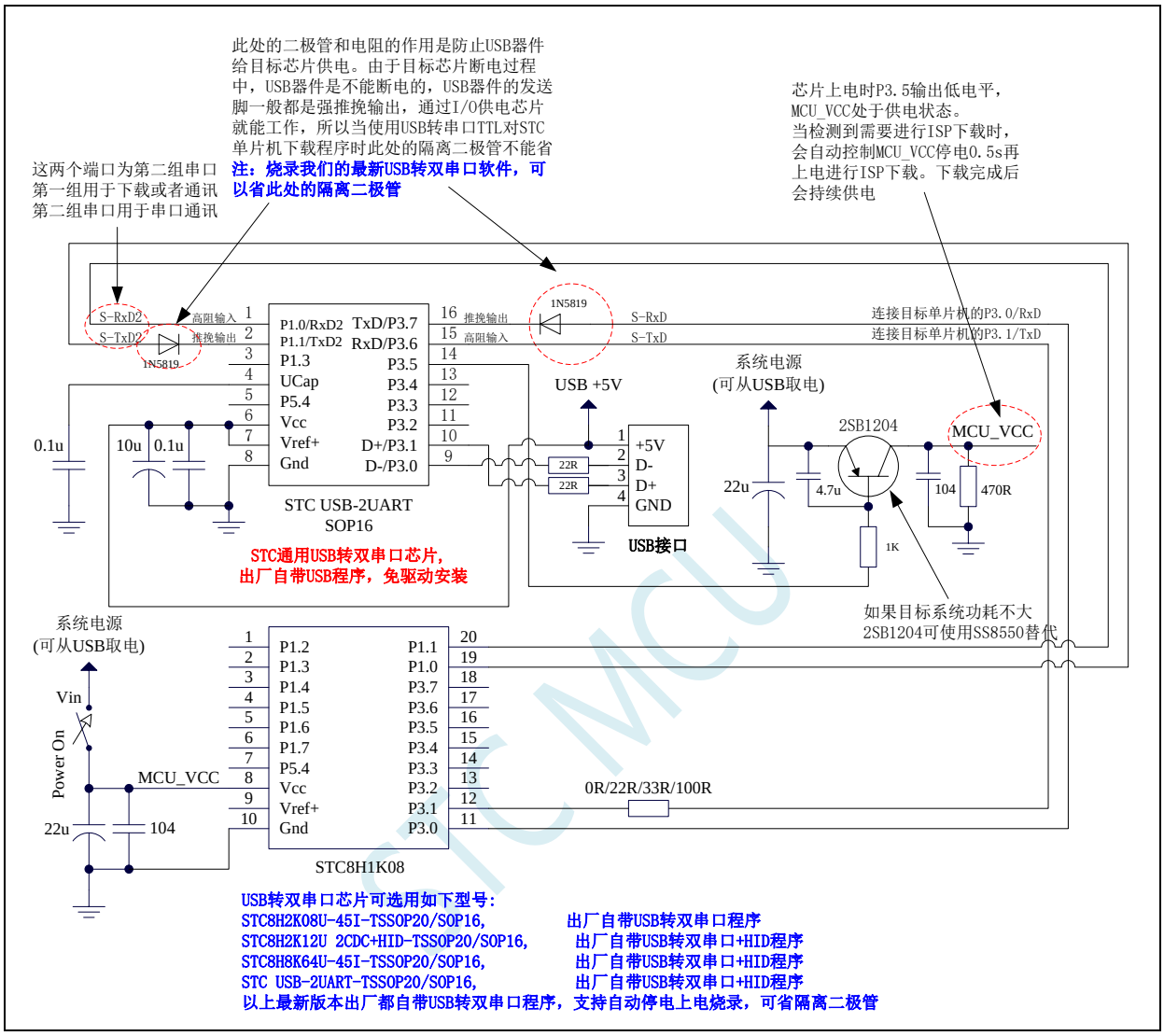
2.5.1 通用 USB 转双串口芯片 STC USB-2UART-45I-TSSOP20，自动停电上电



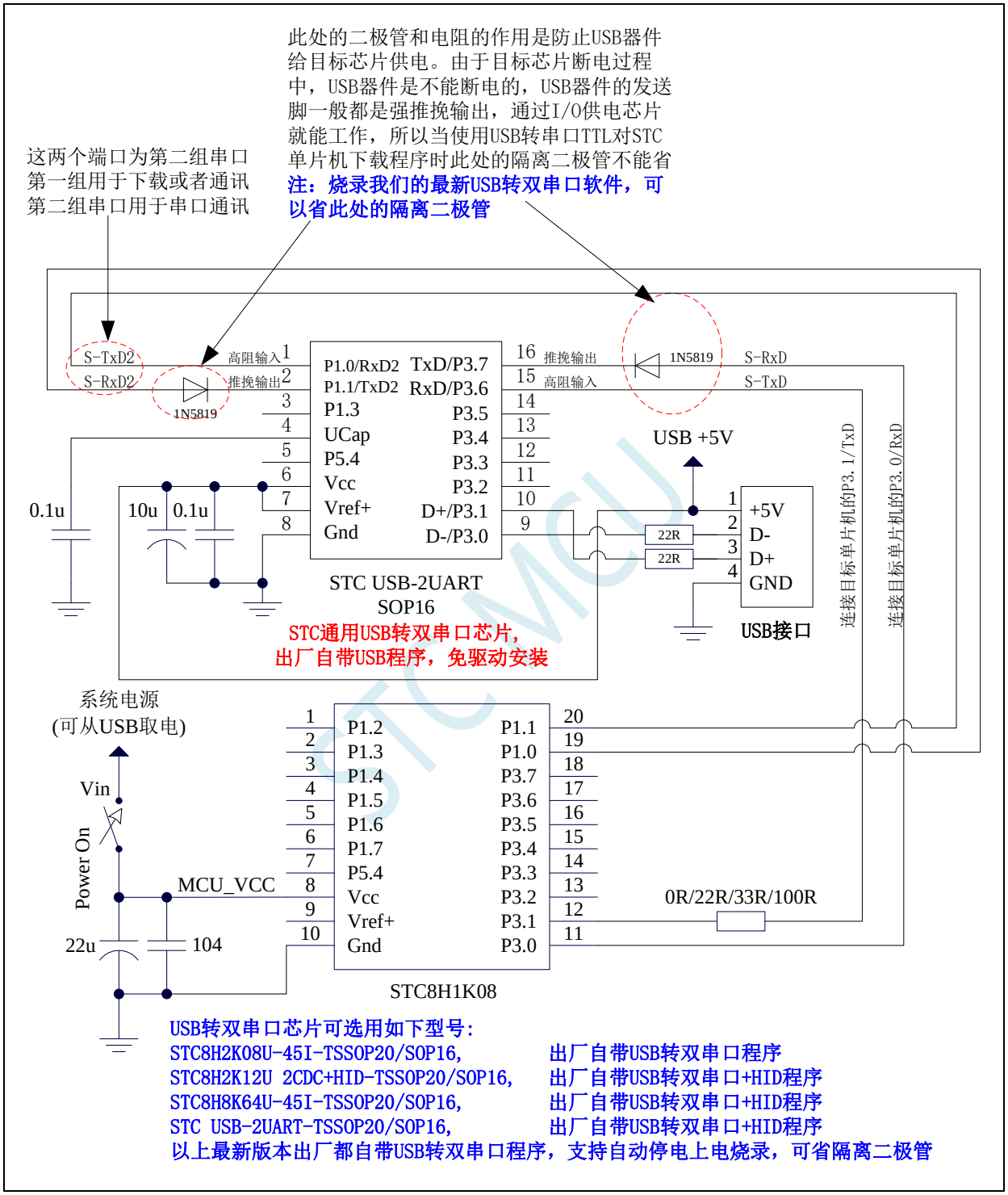
2.5.2 通用 USB 转双串口芯片 STC USB-2UART-45I-TSSOP20，手动停电上电



2.5.3 通用 USB 转双串口芯片 STC USB-2UART-45I- SOP16，自动停电上电



2.5.4 通用 USB 转双串口芯片 STC USB-2UART-45I-SOP16，手动停电上电



3 功能脚切换

STC8C 系列单片机的特殊外设串口、SPI、I²C 以及总线控制脚可以在多个 I/O 直接进行切换，以实现一个外设当作多个设备进行分时复用。

3.1 功能脚切换相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P_SW1	外设端口切换寄存器 1	A2H	S1_S[1:0]		-	-	SPI_S[1:0]		0	-	nnxx,000x
P_SW2	外设端口切换寄存器 2	BAH	EAXFR	-	I2C_S[1:0]		CMPO_S	-	-	S2_S	0x00,0xx0

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

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

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

S1_S[1:0]：串口 1 功能脚选择位

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

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	-	-	-	-
11	P3.5	P3.4	P3.3	P3.2

STC MCU

3.1.2 外设端口切换控制寄存器 2（P_SW2），串口 2、I2C、比较器输出切换

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

EAXFR：扩展 RAM 区特殊功能寄存器（XFR）访问控制寄存器

0：禁止访问 XFR

1：使能访问 XFR。

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

I2C_S[1:0]：I^2C 功能脚选择位

I2C_S[1:0]	SCL	SDA
00	P1.5	P1.4
01	P2.5	P2.4
10	-	-
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

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

3.1.3 时钟选择寄存器（MCLKOCR）

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

MCLKO_S：主时钟输出脚选择位

MCLKO_S	MCLKO
0	P5.4
1	P1.6

3.2 范例程序

3.2.1 串口 1 切换

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"

sfr         P_SW1       =   0xa2;

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

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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW1       DATA        0A2H

P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
```

STC MCU

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

            ORG     0000H
            LJMP    MAIN

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

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

            SJMP    $

            END
```

3.2.2 串口 2 切换

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"

sfr     P_SW2   =   0xba;

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

```
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

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

    P_SW2 = 0x00;                   //RXD2/P1.0, TXD2/P1.1
//  P_SW2 = 0x01;                   //RXD2_2/P4.0, TXD2_2/P4.2

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

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

            ORG     0000H
            LJMP    MAIN

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

STC MCU

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

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

        SJMP    $

        END
```

3.2.3 SPI 切换

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"

sfr     P_SW1       =   0xa2;

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

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

    P_SW1 = 0x00;                   //SS/P1.2, MOSI/P1.3, MISO/P1.4, SCLK/P1.5
//  P_SW1 = 0x04;                   //SS_2/P2.2, MOSI_2/P2.3, MISO_2/P2.4, SCLK_2/P2.5
//  P_SW1 = 0x0c;                   //SS_4/P3.5, MOSI_4/P3.4, MISO_4/P3.3, SCLK_4/P3.2

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW1       DATA    0A2H

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

            ORG     0000H
            LJMP    MAIN

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

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

            SJMP    $

            END
```

3.2.4 I2C 切换

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"

sfr     P_SW2       =   0xba;

sfr     P0M1        =   0x93;
```

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

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

        P_SW2 = 0x00;                   //SCL/P1.5, SDA/P1.4
//      P_SW2 = 0x10;                   //SCL_2/P2.5, SDA_2/P2.4
//      P_SW2 = 0x30;                   //SCL_4/P3.2, SDA_4/P3.3

        while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
```

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

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

        SJMP    $

        END
```

3.2.5 比较器输出切换

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"

sfr     P_SW2       =   0xba;

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

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

```
    P5M1 = 0x00;

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

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

            ORG     0000H
            LJMP    MAIN

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

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

            SJMP    $

            END
```

3.2.6 主时钟输出切换

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"

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

sfr       P_SW2     =   0xba;

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

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

    P_SW2 = 0x80;
    CLKOCR = 0x04;                  //HIRC/4 output via MCLKO/P5.4
//  CLKOCR = 0x84;                  //HIRC/4 output via MCLKO_2/P1.6
    P_SW2 = 0x00;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2     DATA    0BAH

CLKOCR    EQU     0FE05H

P0M1      DATA    093H
P0M0      DATA    094H
P1M1      DATA    091H
P1M0      DATA    092H
P2M1      DATA    095H
P2M0      DATA    096H
P3M1      DATA    0B1H
```

```
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

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

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

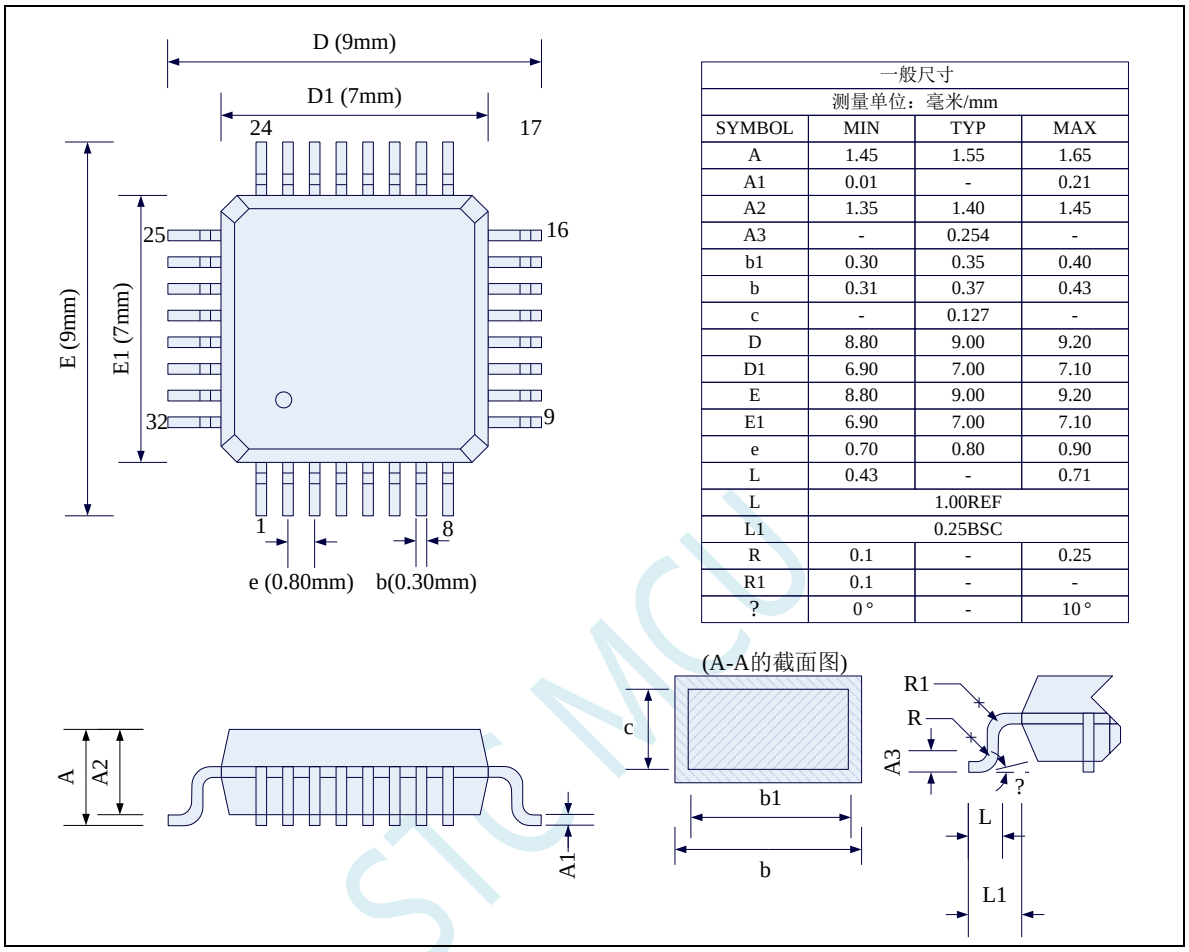
            SJMP    $

            END
```

STC MCU

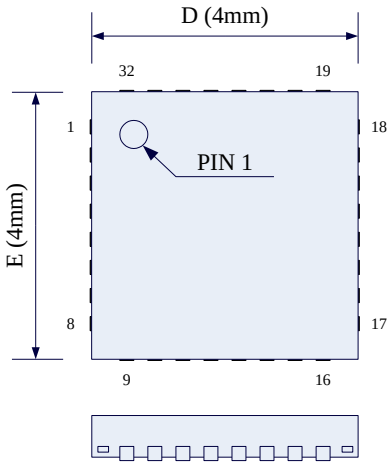
4 封装尺寸图

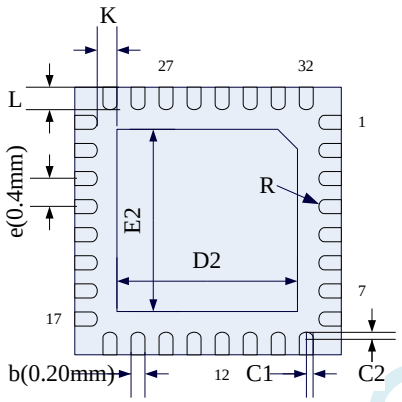
4.1 LQFP32 封装尺寸图（9mm*9mm）

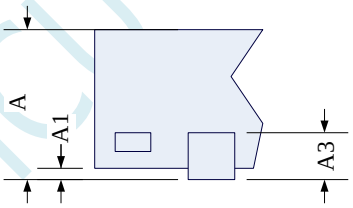


一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	1.45	1.55	1.65
A1	0.01	-	0.21
A2	1.35	1.40	1.45
A3	-	0.254	-
b1	0.30	0.35	0.40
b	0.31	0.37	0.43
c	-	0.127	-
D	8.80	9.00	9.20
D1	6.90	7.00	7.10
E	8.80	9.00	9.20
E1	6.90	7.00	7.10
e	0.70	0.80	0.90
L	0.43	-	0.71
L	1.00REF		
L1	0.25BSC		
R	0.1	-	0.25
R1	0.1	-	-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

4.2 QFN32 封装尺寸图（4mm*4mm）



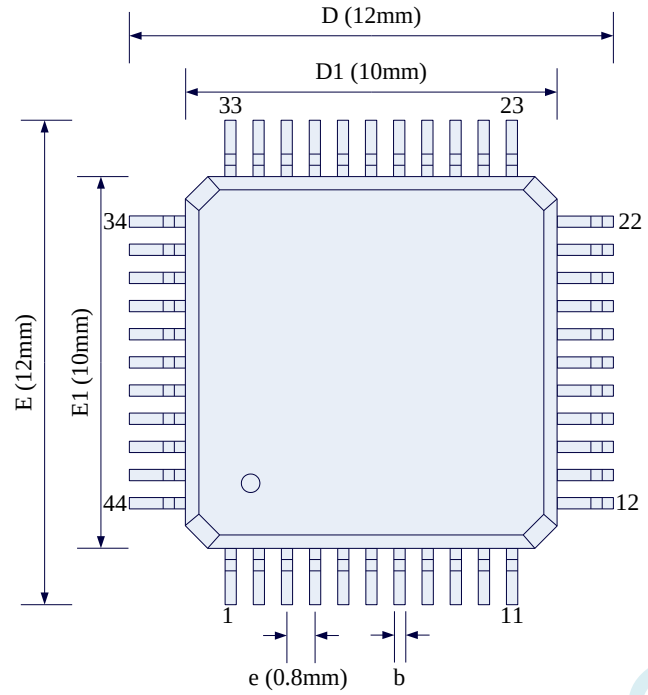




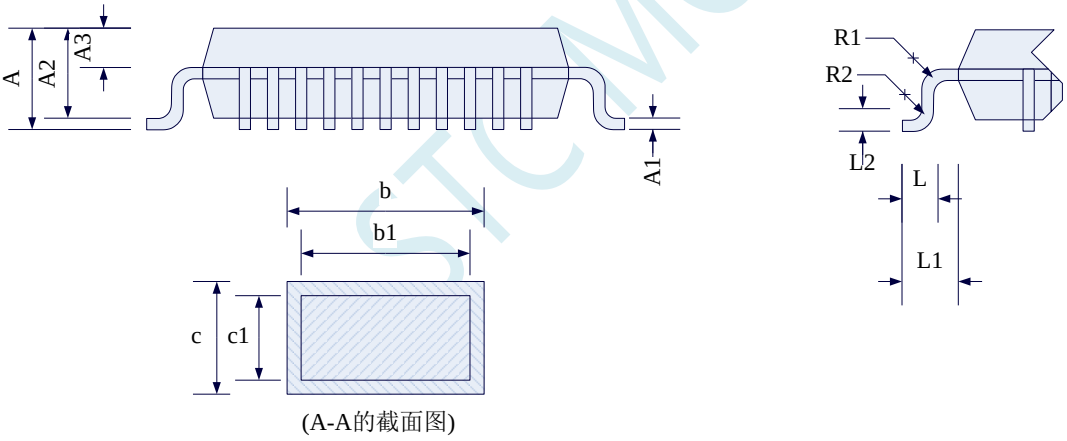
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	0.70	0.75	0.80
A1	0	0.02	0.05
A2	0.50	0.55	0.60
A3	-	0.20REF	-
b	0.15	0.20	0.25
D	3.90	4.00	4.10
E	3.90	4.00	4.10
D2	2.60	2.70	2.80
E2	2.60	2.70	2.80
e	0.30	0.40	0.50
L	0.35	0.40	0.45
K	0.25REF		
R	0.09	-	-
C1	-	0.16	-
C2	-	0.16	-

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

4.3 LQFP44 封装尺寸图（12mm*12mm）



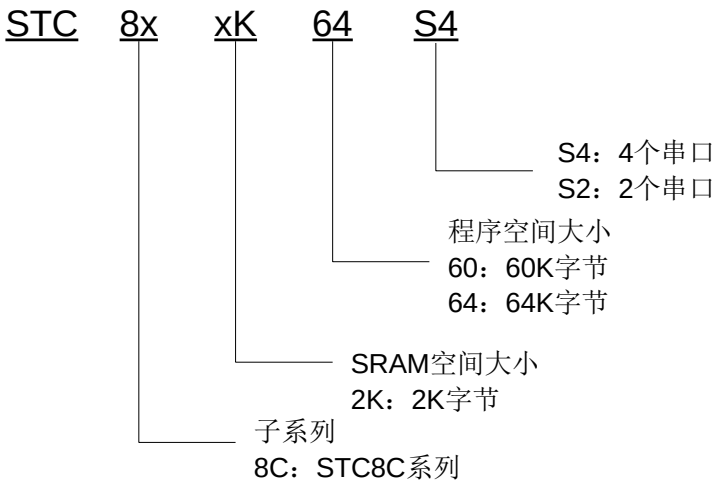
一般尺寸			
测量单位：毫米/mm			
SYMBOL	MIN	TYP	MAX
A	-	-	1.60
A1	0.05	-	0.15
A2	1.35	1.40	1.45
A3	0.59	0.64	0.69
b	0.33	-	0.42
b1	0.32	0.35	0.38
c	0.13	-	0.18
c1	0.12	0.13	0.14
D	11.95	12.00	12.05
D1	9.90	10.00	10.10
E	11.95	12.00	12.05
E1	9.90	10.00	10.10
e	0.70	0.80	0.90
L	0.45	0.60	0.75
L1	1.00REF		
L2	0.25BSC		
R1	0.08	-	-
R2	0.08	-	0.20
S	0.20	-	-



(A-A的截面图)

STCMCU

4.4 STC8C 系列单片机命名规则



STC MCU

5 编译、仿真开发环境的建立与 ISP 下载

5.1 安装 Keil

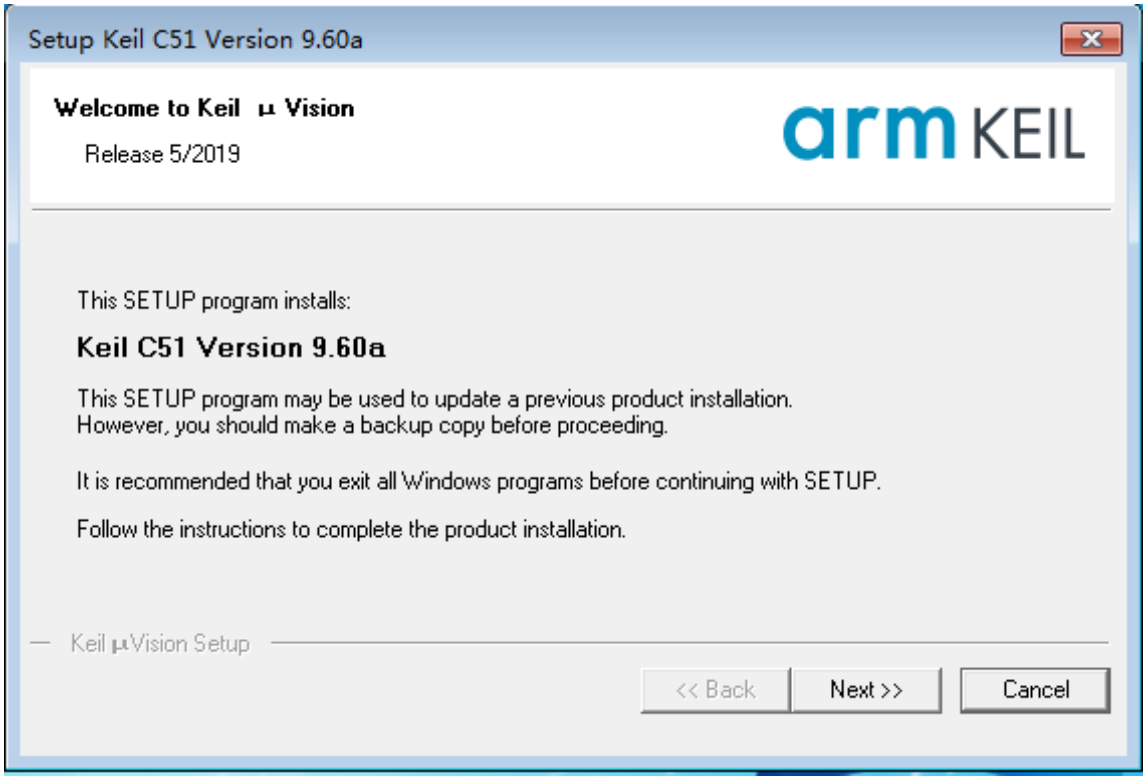
5.1.1 安装 C51 编译环境

首先登录 Keil 官网，下载最新版的 C51 安装包，下载链接如下：

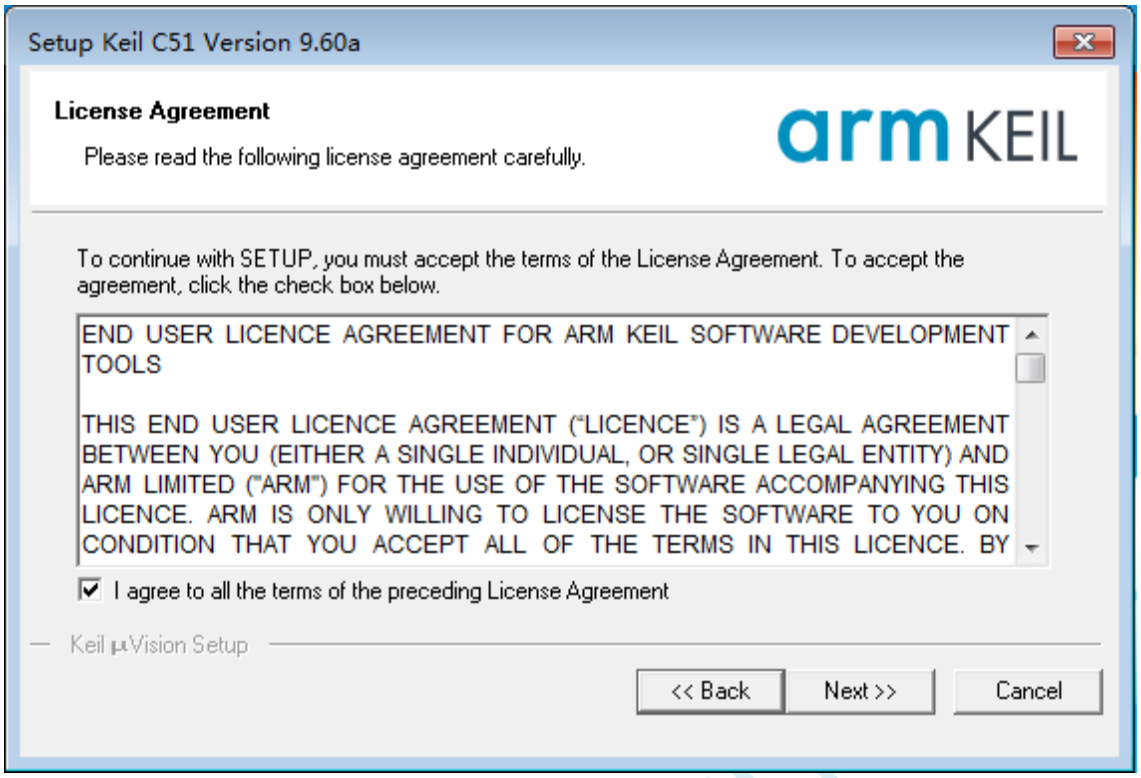
Keil Product Downloads

信息随便填写，点确定后进入下载页面进行下载。

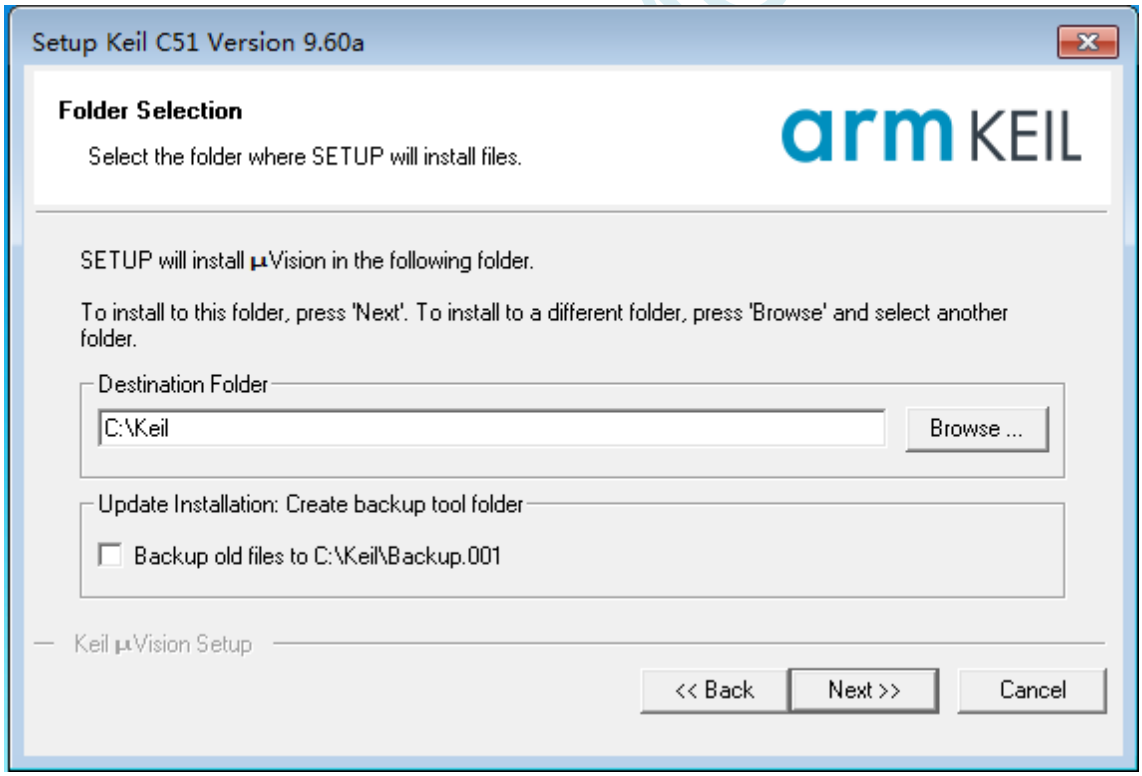
双击下载的安装包开始安装，点击“Next”：



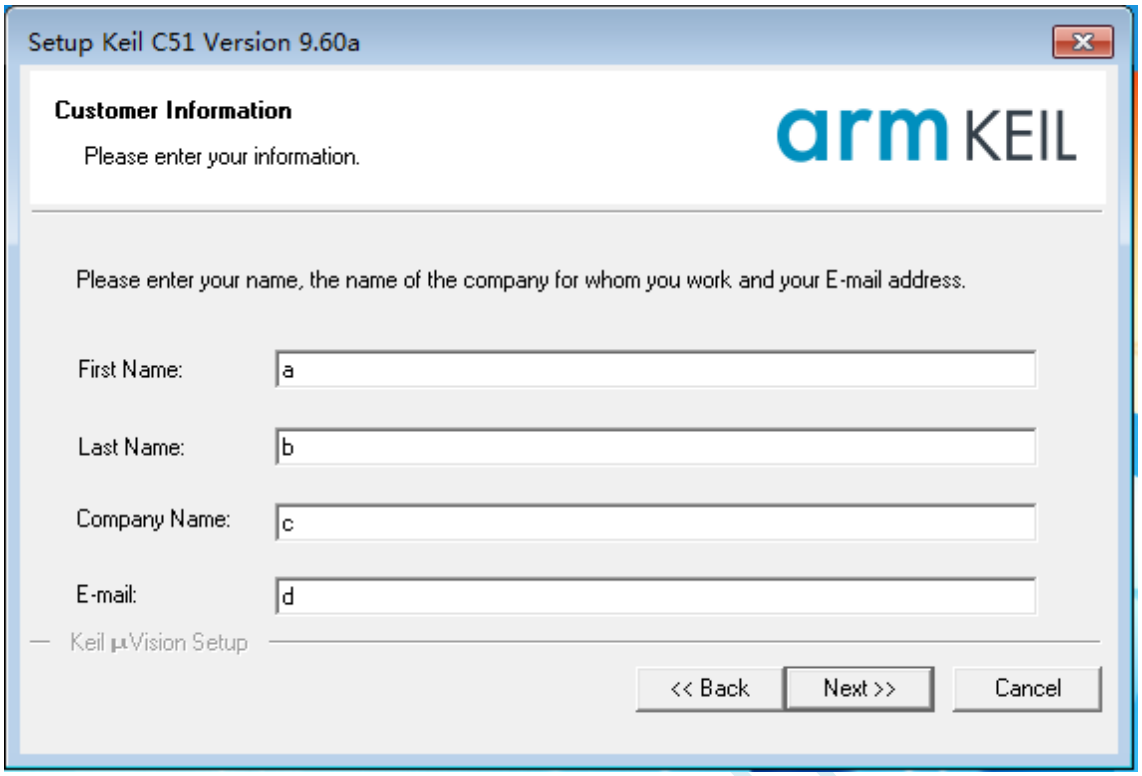
勾选“I agree to all the terms of the preceding License Agreement”,然后点击“Next”：



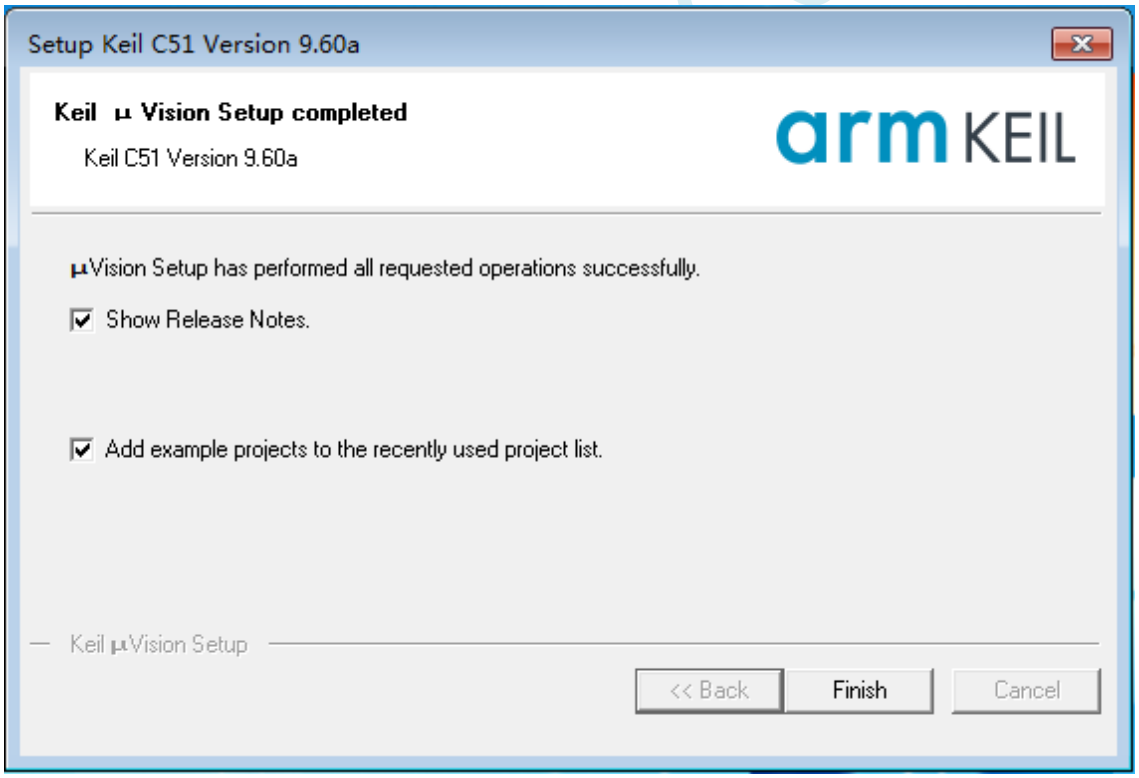
选择安装目录，然后点击“Next”：



填写个人信息，然后点击“Next”：

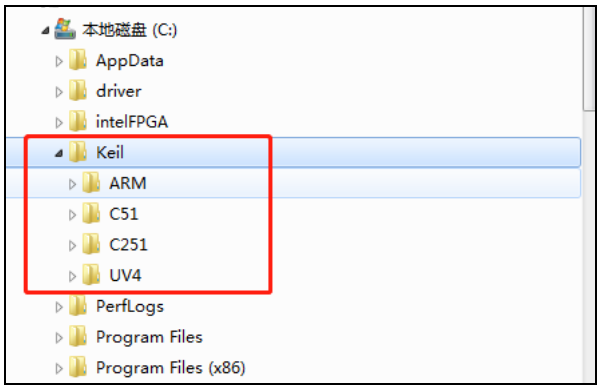


安装完成，点击“Finish”结束。

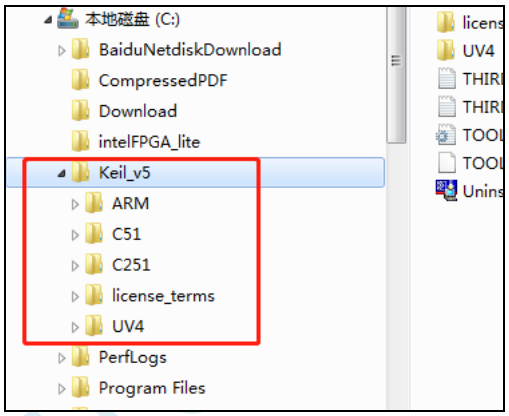


5.1.2 如何同时安装 Keil 的 C51、C251 和 MDK

旧版本的 Keil 软件的安装目录默认是 C:\Keil，C51、C251 和 MDK 分别会被安装在 C:\Keil 目录下的 C51、C251 和 ARM 目录中，如下图所示。



新版本的 Keil 软件的安装目录默认是 C:\Keil_v5，C51、C251 和 MDK 分别会被安装在 C:\Keil_v5 目录下的 C51、C251 和 ARM 目录中，如下图所示。

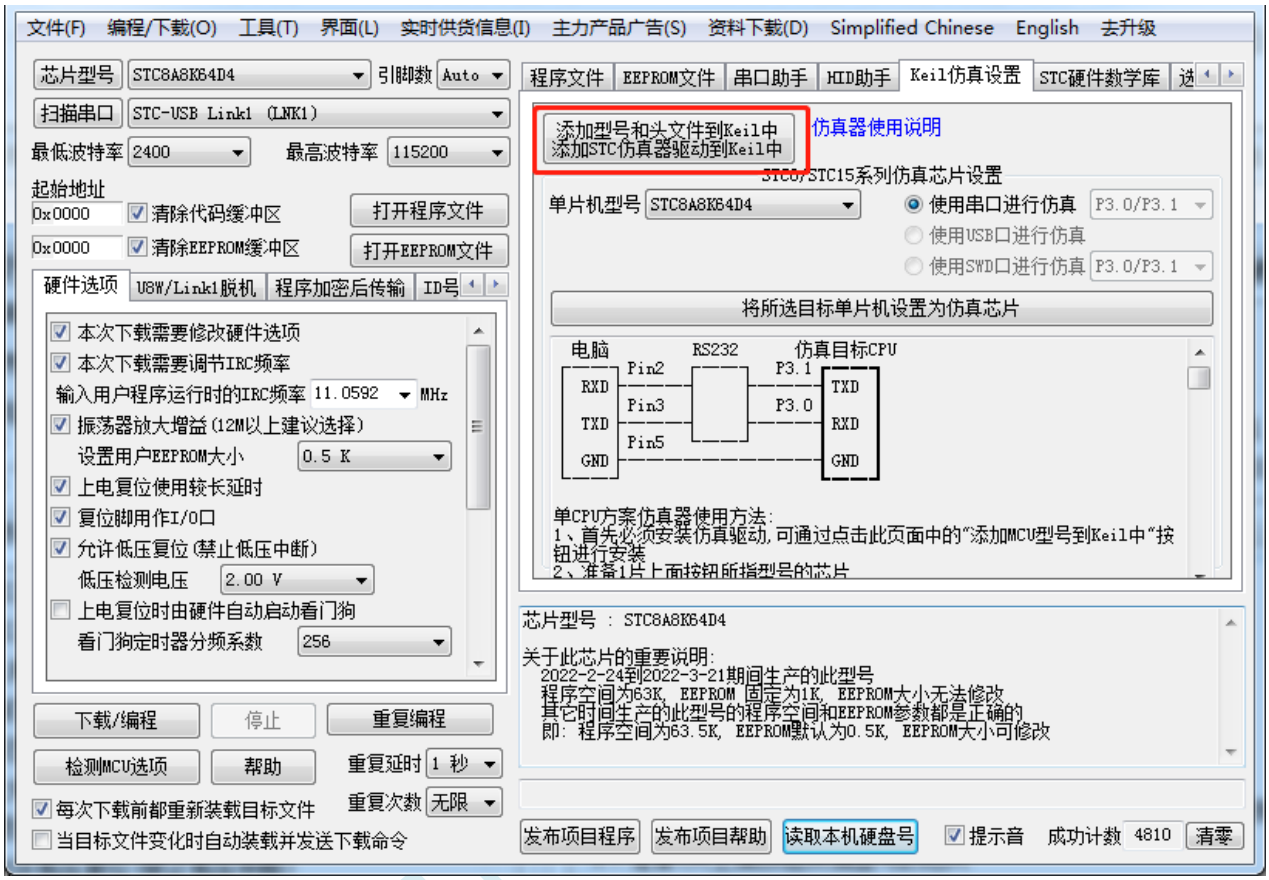


无论是新版本还是旧版本，C51、C251 和 MDK 是安装在不同的目录，并不会有冲突。软件的和谐也是 3 个软件分别进行的，之前已经安装完成并设置好的软件，并不会因为后续有安装新的软件而改变。所以安装时只需要按照默认方式安装即可，Keil 软件会自动处理好。

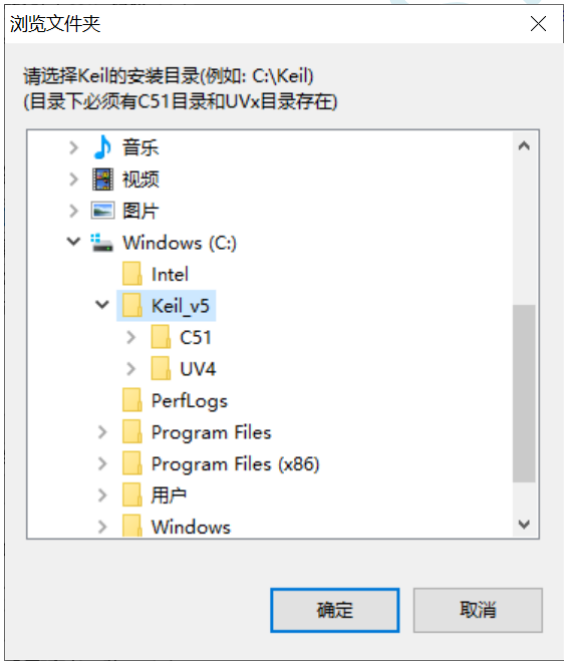
5.2 添加型号和头文件到 Keil

使用 Keil 之前需要先安装 STC 的仿真驱动。STC 的仿真驱动的安装步骤如下：

首先开 STC 的 ISP 下载软件，然后在软件右边功能区的“Keil 仿真设置”页面中点击“添加型号和头文件到 Keil 中 添加 STC 仿真器驱动到 Keil 中”按钮：



按下后会出现如下画面：



将目录定位到 Keil 软件的安装目录，然后确定。安装成功后会弹出如下的提示框：



即表示驱动正确安装了

头文件默认复制到 Keil 安装目录下的“C251\INC\STC”目录中
在 C 代码中使用“#include <STC32G.H>”或者“#include "STC32G.H"”进行包含均可正确使用

STC MCU

5.3 STC 单片机程序中头文件的使用方法

c 语言中 include 用法

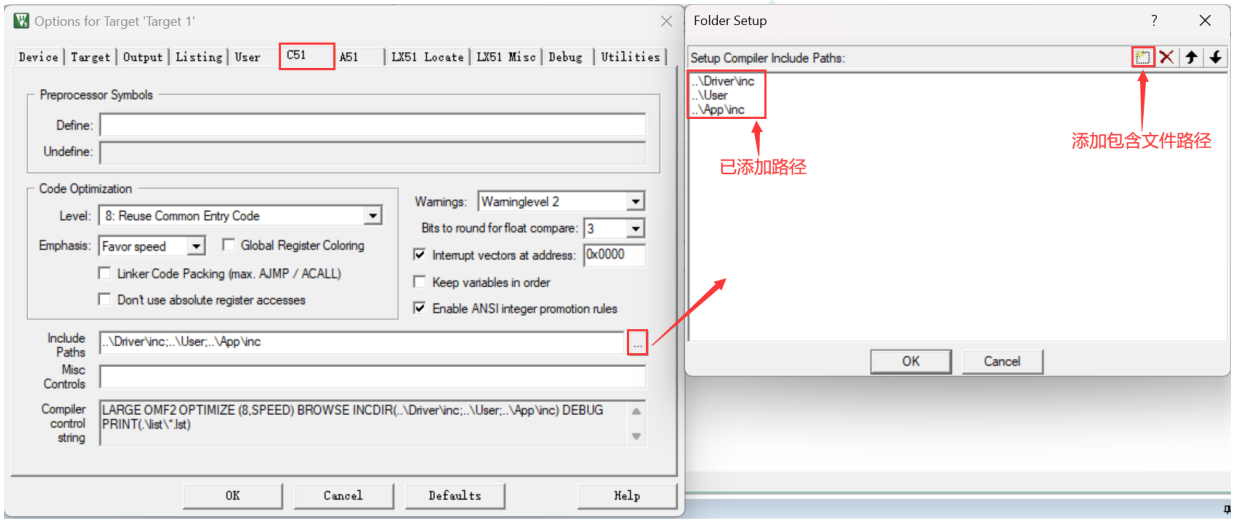
#include 命令是预处理命令的一种，预处理命令可以将别的源代码内容插入到所指定的位置。
有两种方式可以指定插入头文件：

#include <文件名.h>
#include "文件名.h"

使用尖括号< >和双引号" "的区别在于头文件的搜索路径不同：
使用尖括号< >，编译器会到系统路径下查找头文件；
使用双引号" "，编译器首先在当前目录下查找头文件，如果没有找到，再到系统路径下查找。

路径设置方式 1：

通过 keil 设置界面，添加包含文件的路径：



添加后，调用时直接使用 #include "文件名.h" 就可以将需要的文件包含进来，编译器会自动到以上路径下面寻找所包含的文件。
这种情况下，使用双引号" "包含头文件，编译器首先在当前目录下查找头文件，如果没有找到，编译器会到 keil 设置路径查找，还没有的话再到系统路径下查找。（注：系统路径是编译器安装位置存放头文件的目录）

路径设置方式 2：

在包含文件名前添加绝对路径，例如：

#include "E:\xxxx\xxxx\文件名.h"
#include "E:/xxxx/xxxx/文件名.h"

路径设置方式 3：

在包含文件名前添加相对路径，例如：

#include "..\comm\文件名.h"
#include "../comm/文件名.h"

其中 ".."是指上一级目录，以上路径是指包含文件在当前目录的上一级目录的 comm 目录下面。

汇编语言中 include 用法与 c 语言类似，将"#"换成"$"，用小括号()包含文件：

$include (../../comm/STC8H.INC)

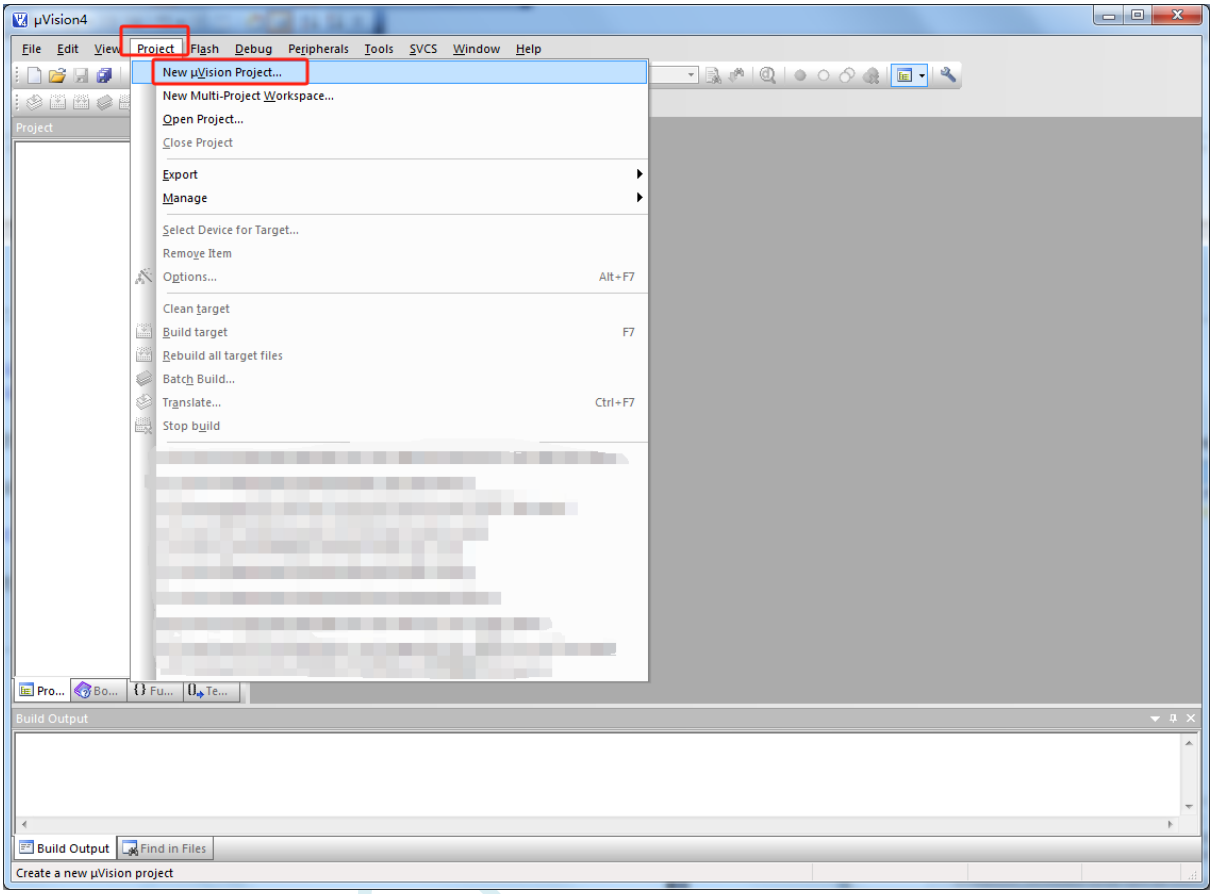
以上指令表示要包含的文件 STC8H.INC，在当前目录的上一级目录的上一级目录的 comm 目录下面。

STC MCU

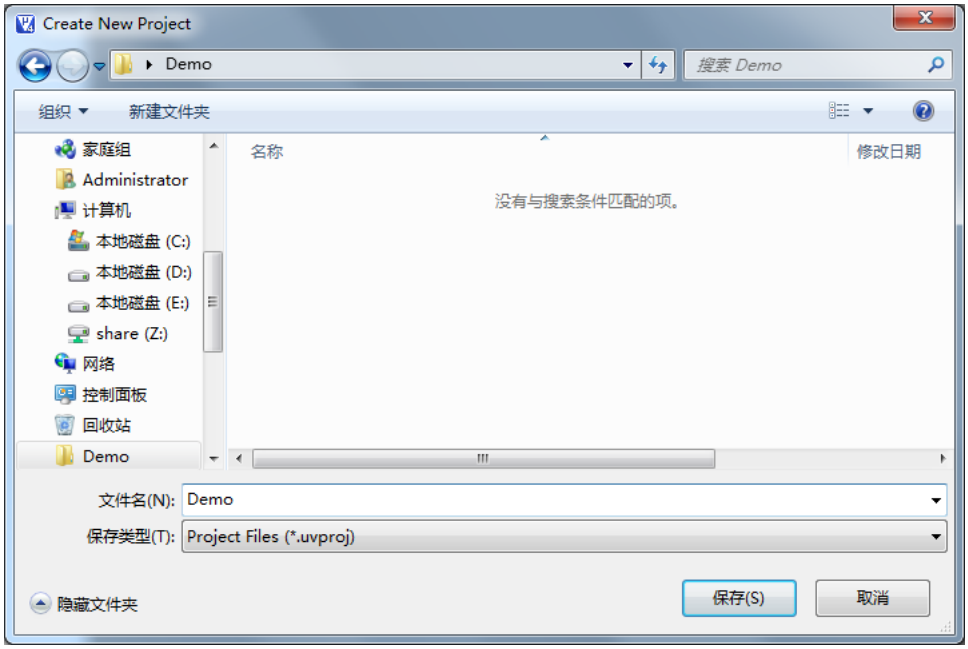
5.4 新建项目与项目设置

5.4.1 设置项目路径和项目名称

打开 Keil 软件，并点击“Project”菜单中的“New uVision Project ...”项

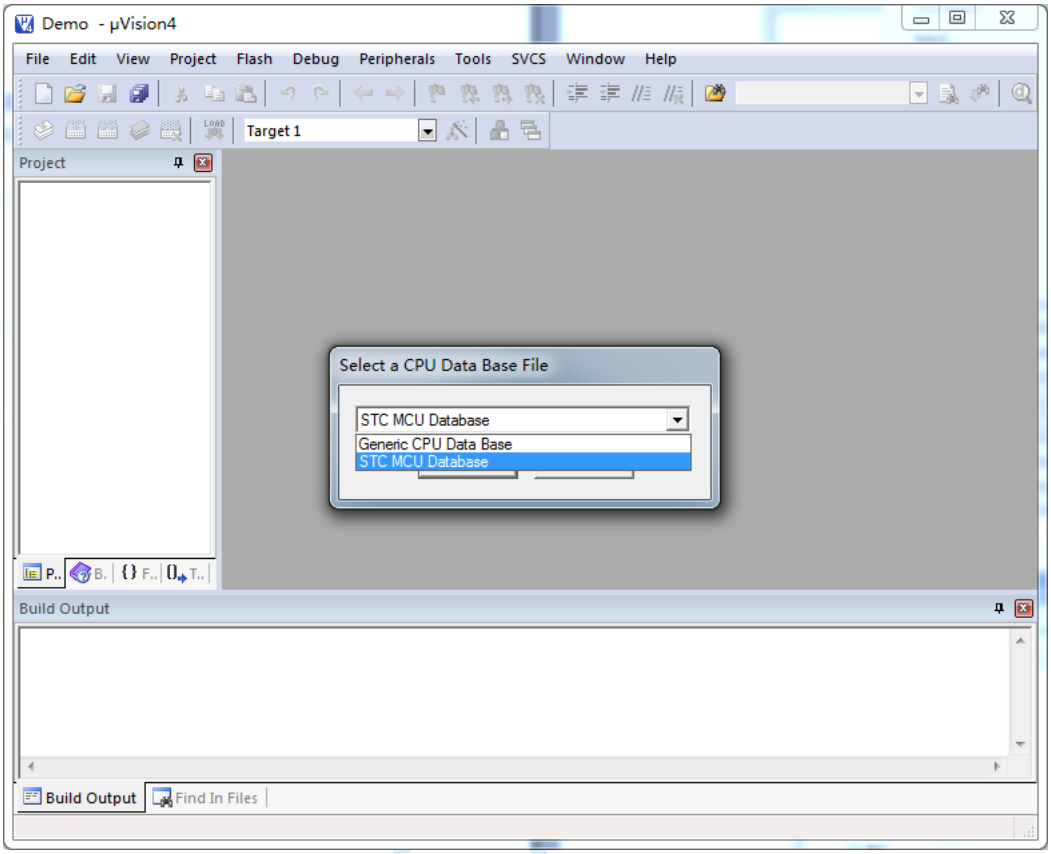


将目录定位在准备好的项目文件夹中，并输入项目名称（例如：Demo）

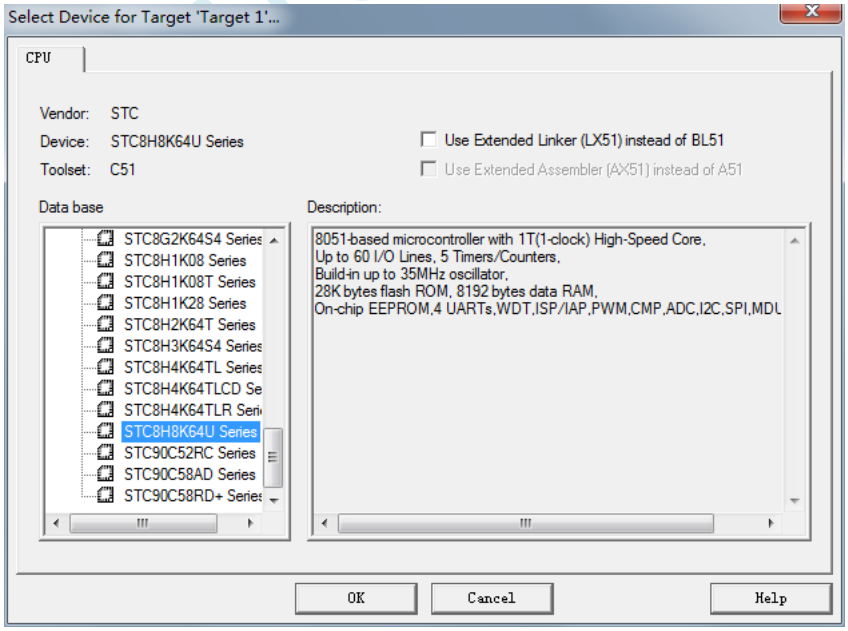


5.4.2　选择目标单片机型号（STC8H8K64U）

在弹出的“Select a CPU Data Base File”窗口中选择“STC MCU Database”

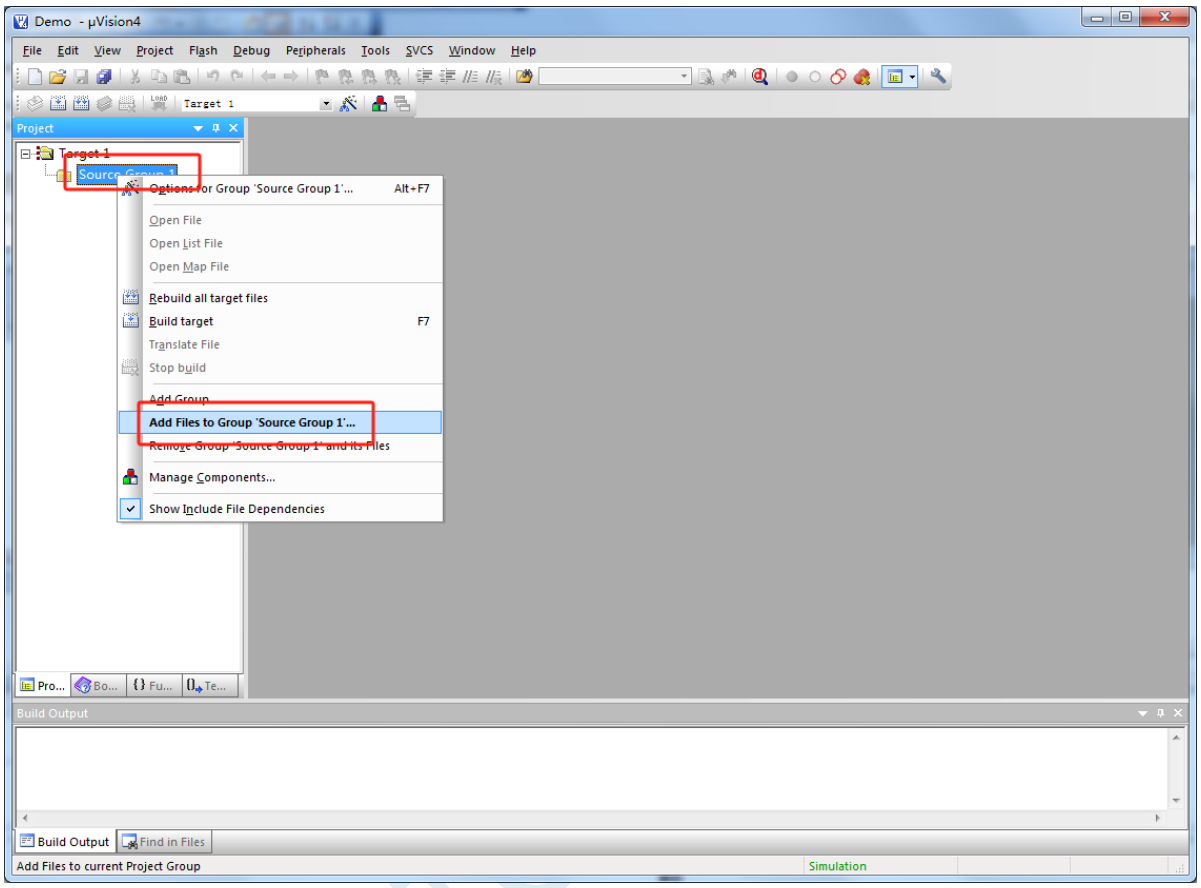


在“Select Device for Target ...”窗口中选择正确的目标单片机型号（例如：STC8H8K64U）

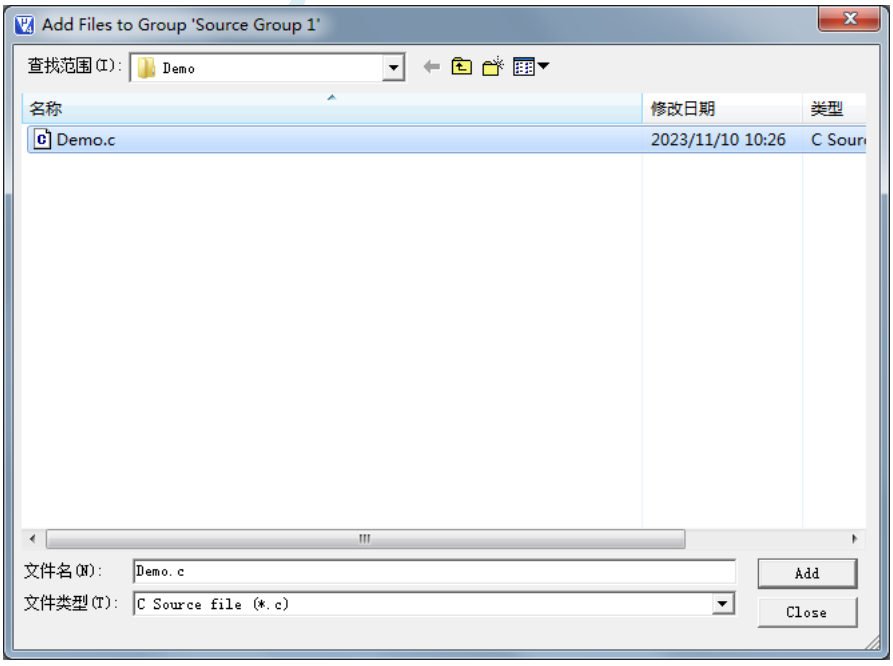


5.4.3　添加源代码文件到项目

如下图所示，在“Source Group 1”所在的图标点击鼠标右键，并选择右键菜单中的“Add Files to Group ' Source Group 1'...”

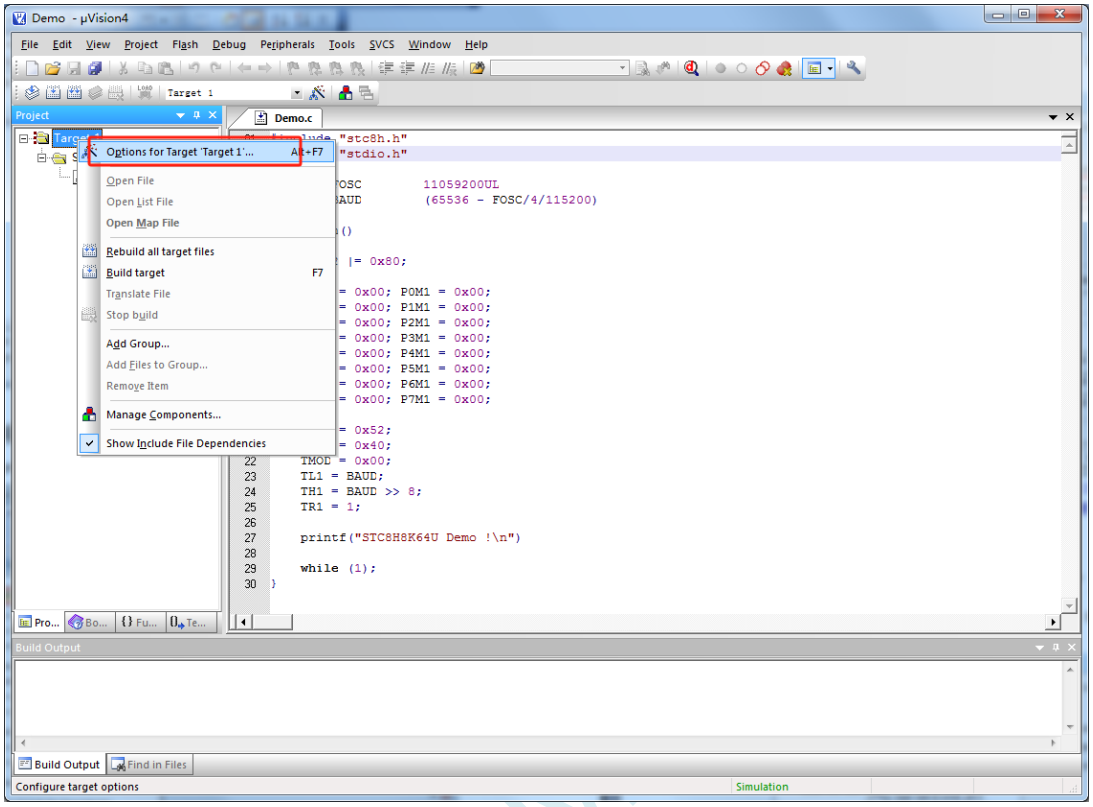


选择已编辑完成的代码文件加入到项目中



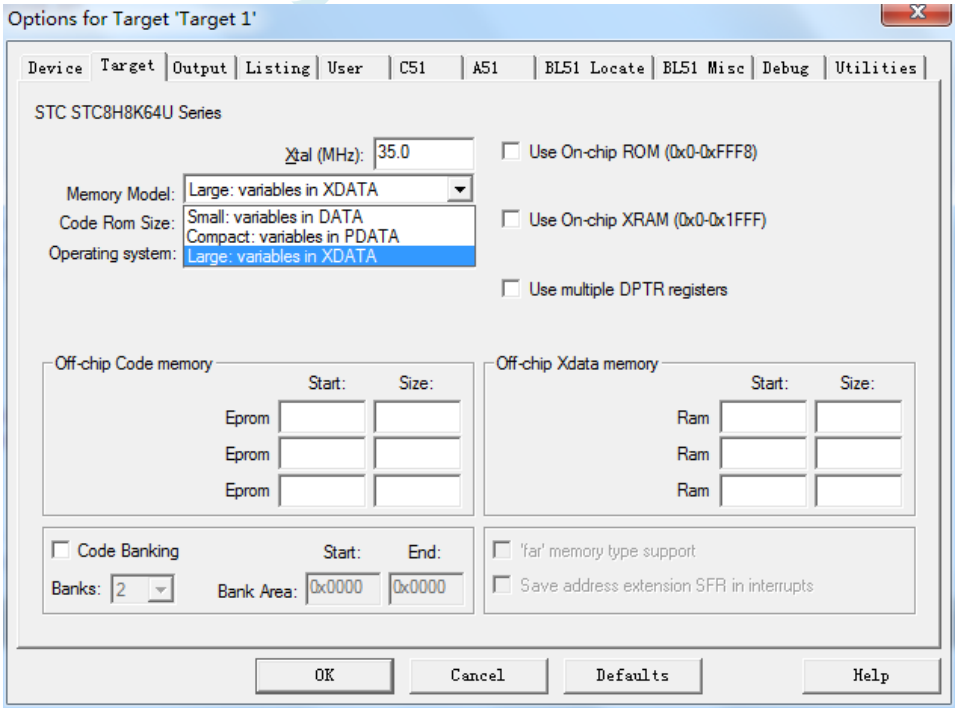
5.4.4　设置项目 1（设置“Memory Model”）

如下图所示，在“Target1”所在的图标点击鼠标右键，并选择“Options for Target ' Target 1'...”



弹出的“Options for Target ' Target 1'”窗口中选择“Target”选项页，在“Memory Model”的下拉选项中可选择“Small”模式或者“Large”模式。

在 Keil 软件中的“Memory Model”有如下 3 个选择



各种模式对比如下表：

Memory Model	默认变量类型 （数据存储器）	存储器大小	地址范围
Small 模式	data	128 字节	D:00 ~ D:7F
Compact 模式	pdata	256 字节	X:0000 ~ X:00FF
Large 模式	**xdata**	**64K 字节（理论值）**	**X:0000 ~ X:FFFF**

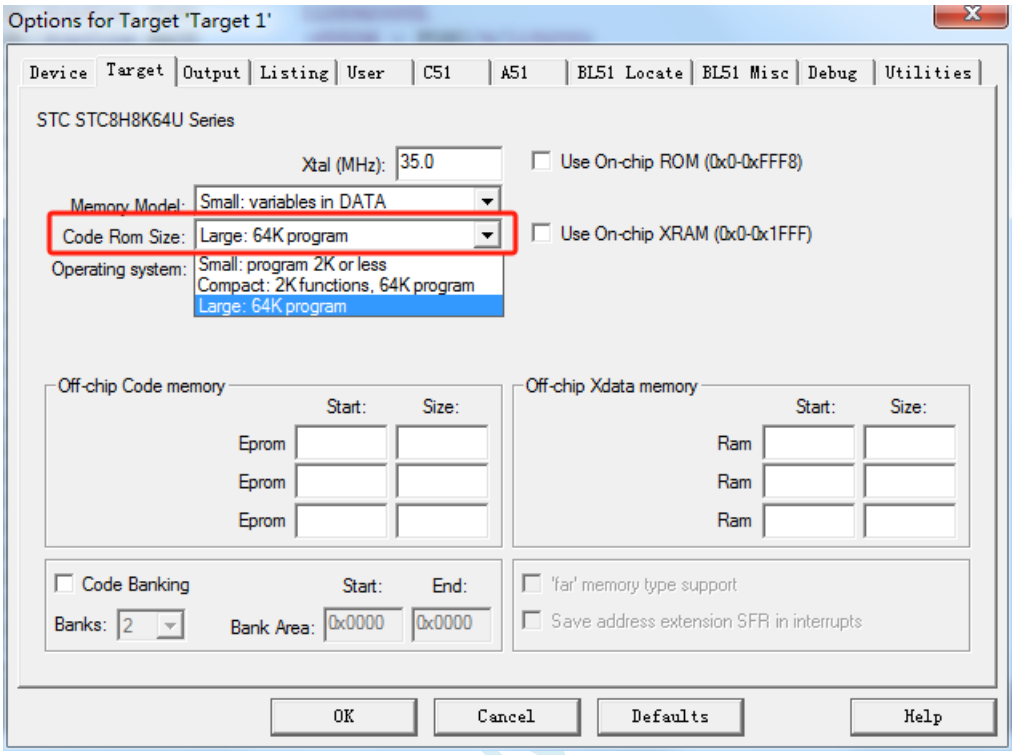
为了达到比较高的效率，一般建议选择“Small”模式，当编译器出现“error C249: 'DATA': SEGMENT TOO LARGE”错误时，则需要手动将部分比较大的数组通过“xdata”强制分配到 XDATA 区域（例如：char xdata buffer[256];）

STC MCU

5.4.5 设置项目 3（“Code Rom Size”选择 Large）

在“Code Rom Size”的下拉选项中选择“Large: ...”模式

8051 的代码大小模式，在 Keil 环境下有如下图所示的 3 种模式：

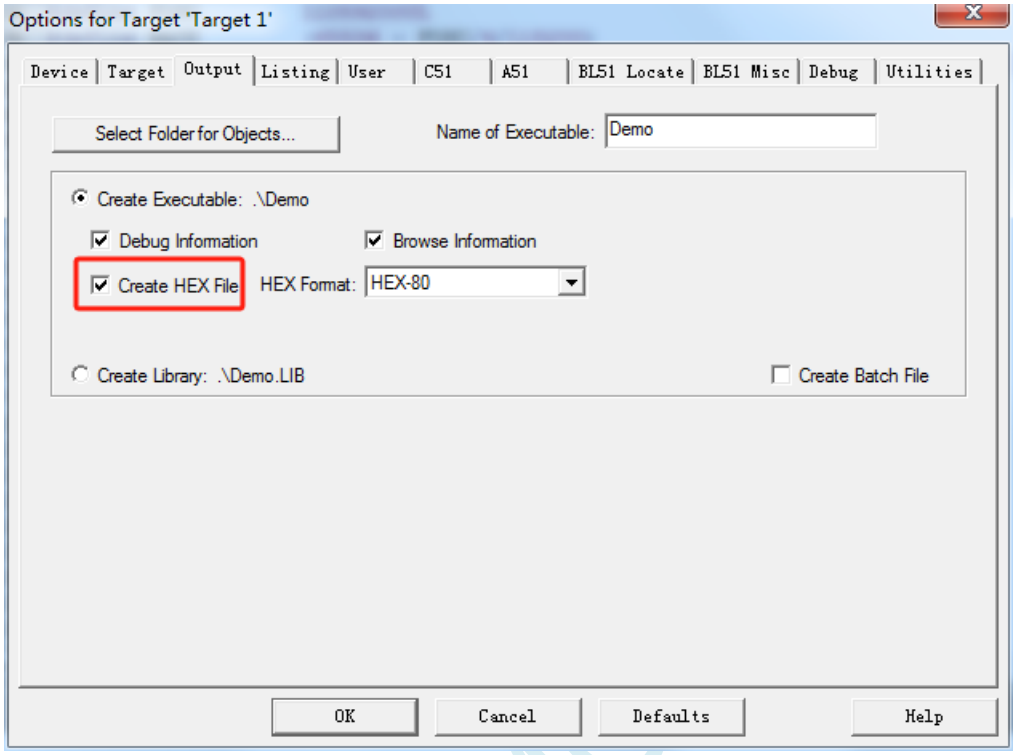


各种模式对比如下表：

Code Rom Size	跳转/调用指令	代码大小限制	
		单个函数/模块/文件的代码大小	总代码大小
Small 模式	AJMP/ACALL	2K	2K
Compact 模式	内部模块代码使用 AJMP/ACALL 外部模块代码使用 LJMP/LCALL	2K	64K
Large 模式	LCALL/LJMP	64K	64K

5.4.6 设置项目 5（HEX 文件格式设置）

“Options for Target ' Target 1'”窗口中选择“Output”选项页，勾选其中的“Create HEX File”选项。



完成上面的设置后，鼠标单击如下图所示的编译按钮，如果代码没有错误，即可生成 HEX 文件

5.5 如何在 Keil C51 中对变量、表格数据、函数指定绝对地址

5.5.1 Keil C51 中，变量如何指定绝对地址

语法如下：

数据类型 [存储类型] 变量名称 **_at_** 绝对地址;

在 data 区域指定绝对地址变量的范例：

```
int  data  var_data_abs  _at_  0x50;
```

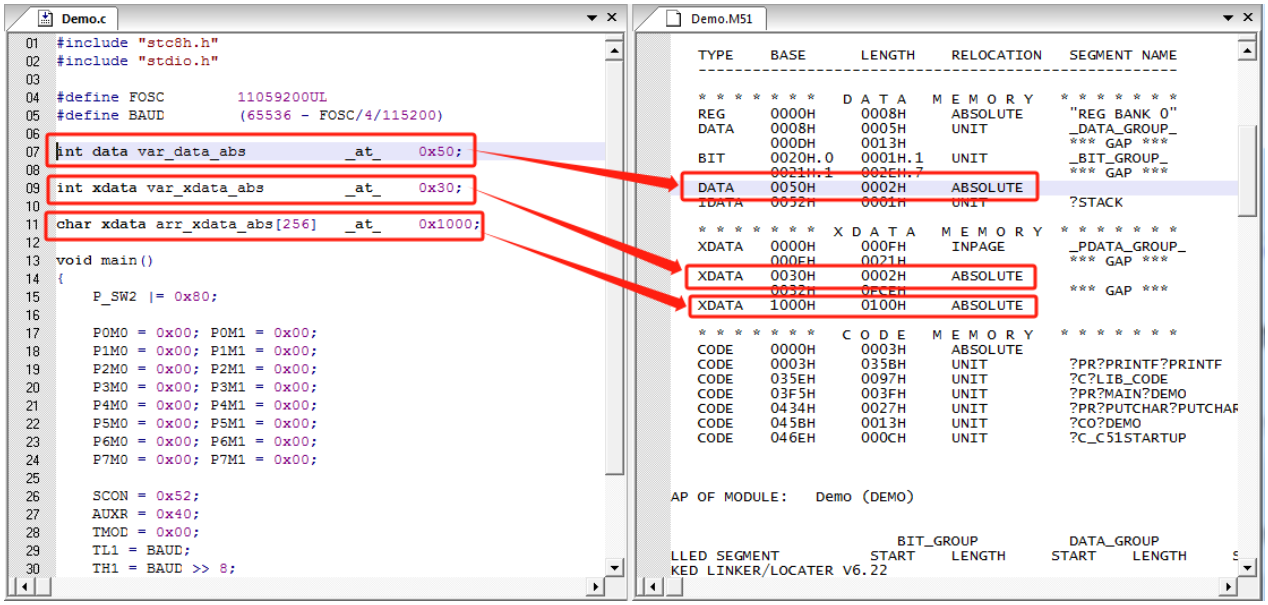
在 xdata 区域指定绝对地址变量的范例：

```
int  xdata  var_xdata_abs  _at_  0x30;
```

在 data 区域指定绝对地址变量的范例：

```
char  xdata  arr_xdata_abs[256]  _at_  0x1000;
```

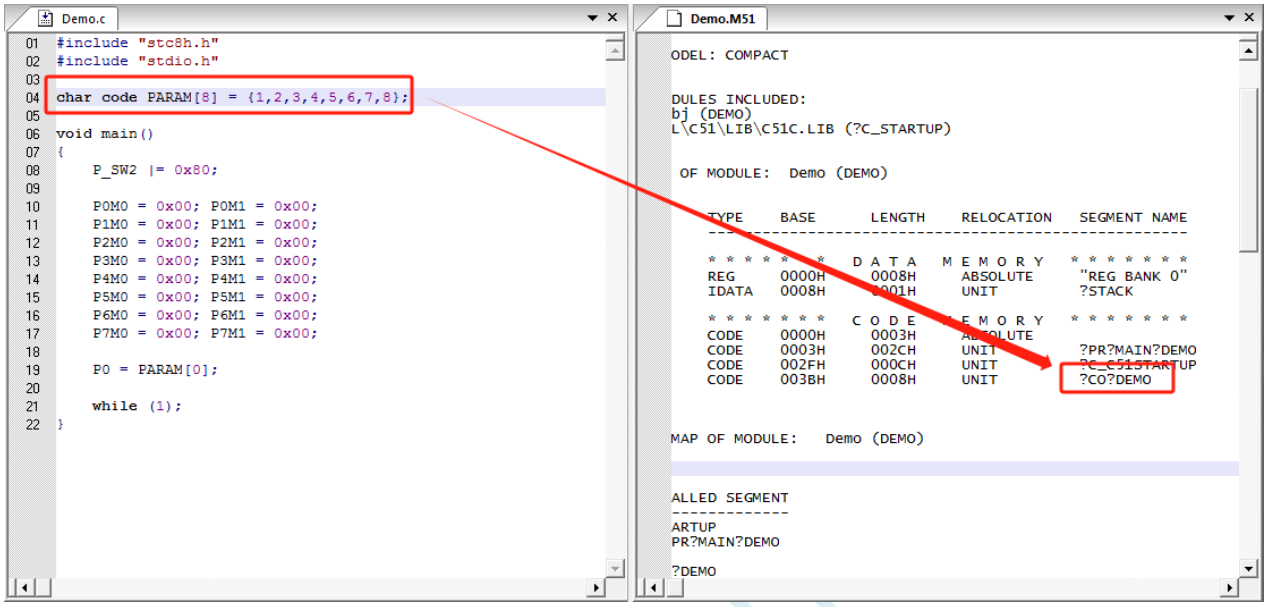
编译完成后地址分配如下图：



5.5.2 Keil C51 中，表格数据如何指定绝对地址

C51 中无法直接在程序中指定表格数据的绝对地址，需要通过如下方法实现

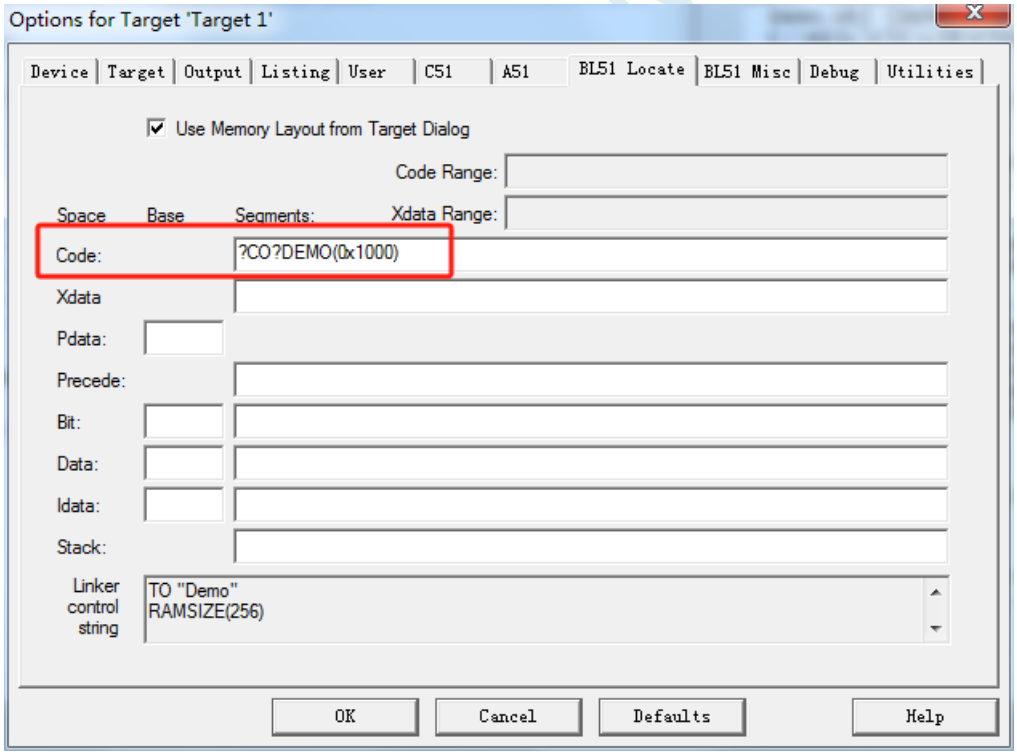
首先在程序中定义表格数据，编译成功后，获取表格的链接符号（如下图为“?CO?DEMO”）



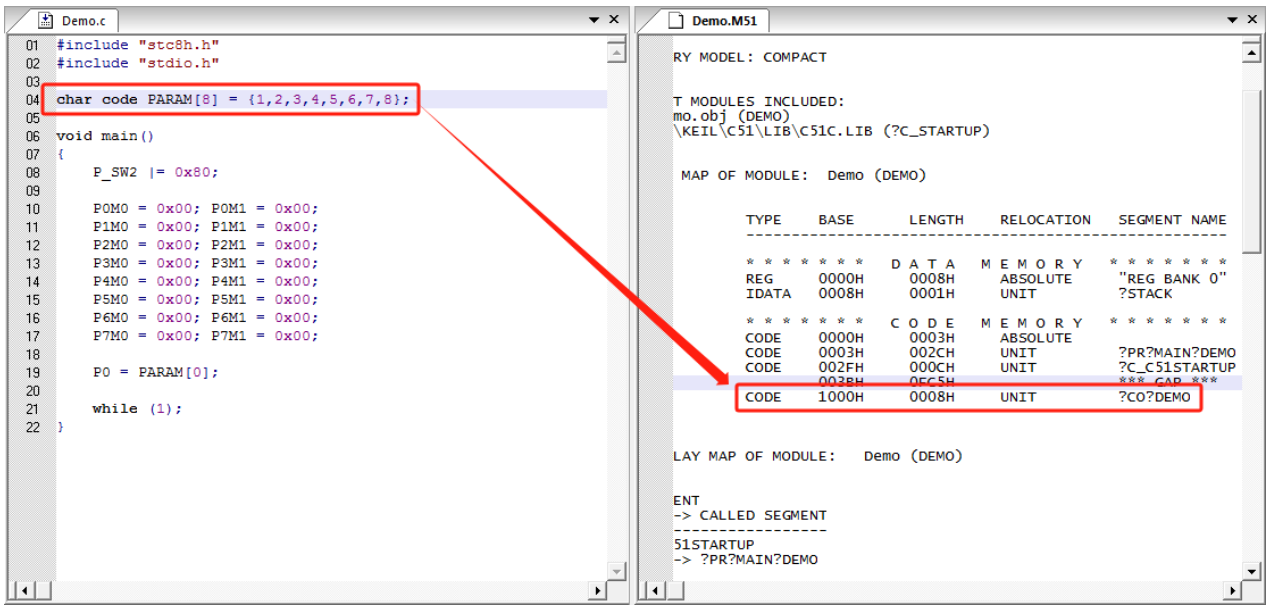
接下来在项目设置项中打开“BL51 Locate”设置页面

在 code 一栏中按照：链接名称 (链接地址)　的格式，输入绝对地址

如下图，将表格数据指定到代码区的 0x1000 的绝对地址



设置完成后，再次编译，表格数据即可被链接到指定的绝对地址，如下图：



Demo.c

```
#include "stc8h.h"
#include "stdio.h"

char code PARAM[8] = {1,2,3,4,5,6,7,8};

void main()
{
    P_SW2 |= 0x80;

    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;
    P4M0 = 0x00; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;
    P6M0 = 0x00; P6M1 = 0x00;
    P7M0 = 0x00; P7M1 = 0x00;

    P0 = PARAM[0];

    while (1);
}
```

Demo.M51

```
RY MODEL: COMPACT


T MODULES INCLUDED:
mo.obj (DEMO)
\KEIL\C51\LIB\C51C.LIB (?C_STARTUP)


 MAP OF MODULE:  Demo (DEMO)


        TYPE    BASE      LENGTH    RELOCATION   SEGMENT NAME
        -----------------------------------------------------

        * * * * * * *   D A T A   M E M O R Y   * * * * * * *
        REG     0000H     0008H     ABSOLUTE     "REG BANK 0"
        IDATA   0008H     0001H     UNIT         ?STACK

        * * * * * * *   C O D E   M E M O R Y   * * * * * * *
        CODE    0000H     0003H     ABSOLUTE
        CODE    0003H     002CH     UNIT         ?PR?MAIN?DEMO
        CODE    002FH     000CH     UNIT         ?C_C51STARTUP
                003BH     0FC5H                  *** GAP ***
        CODE    1000H     0008H     UNIT         ?CO?DEMO


LAY MAP OF MODULE:   Demo (DEMO)

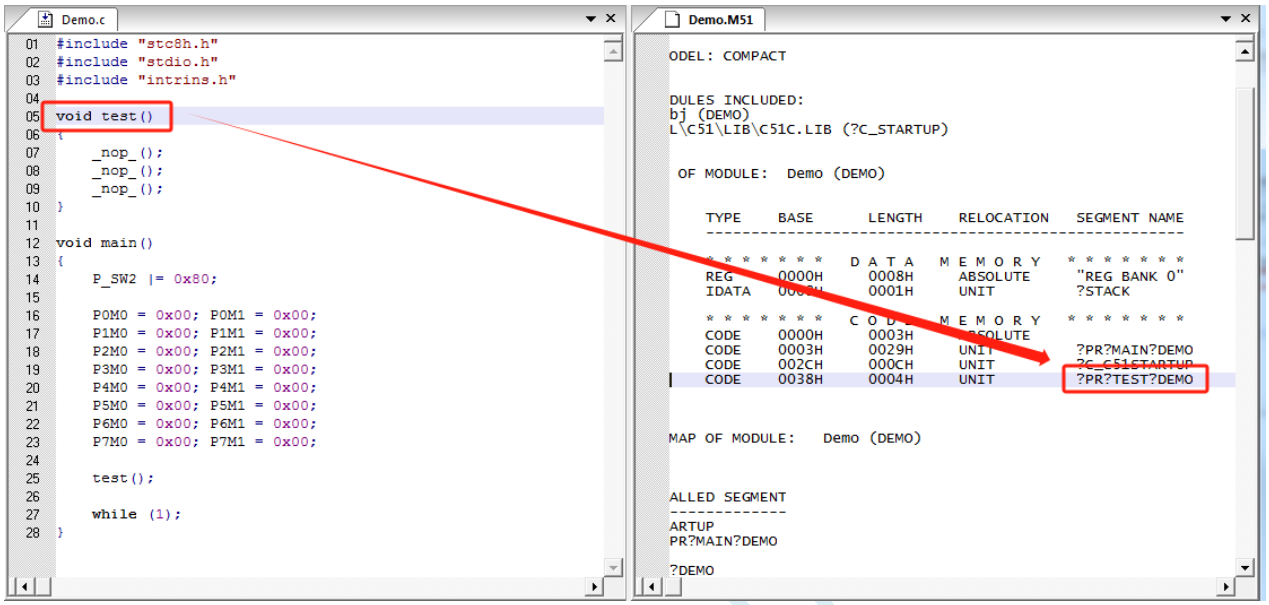

ENT
-> CALLED SEGMENT
-----------------
51STARTUP
-> ?PR?MAIN?DEMO
```

STC MCU

5.5.3 Keil C51 中，函数如何指定绝对地址

C51 中无法直接在程序中指定函数的绝对地址，需要通过如下方法实现

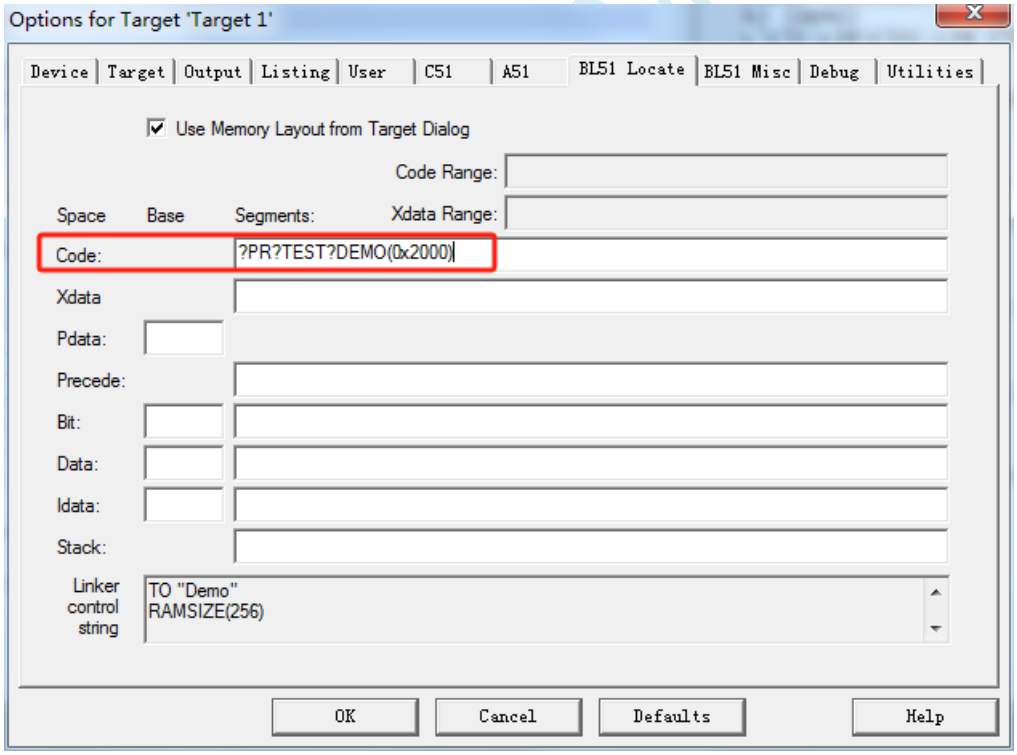
首先在程序编写完成函数代码，编译成功后，获取函数的链接符号（如下图为“?PR?TEST?DEMO”）



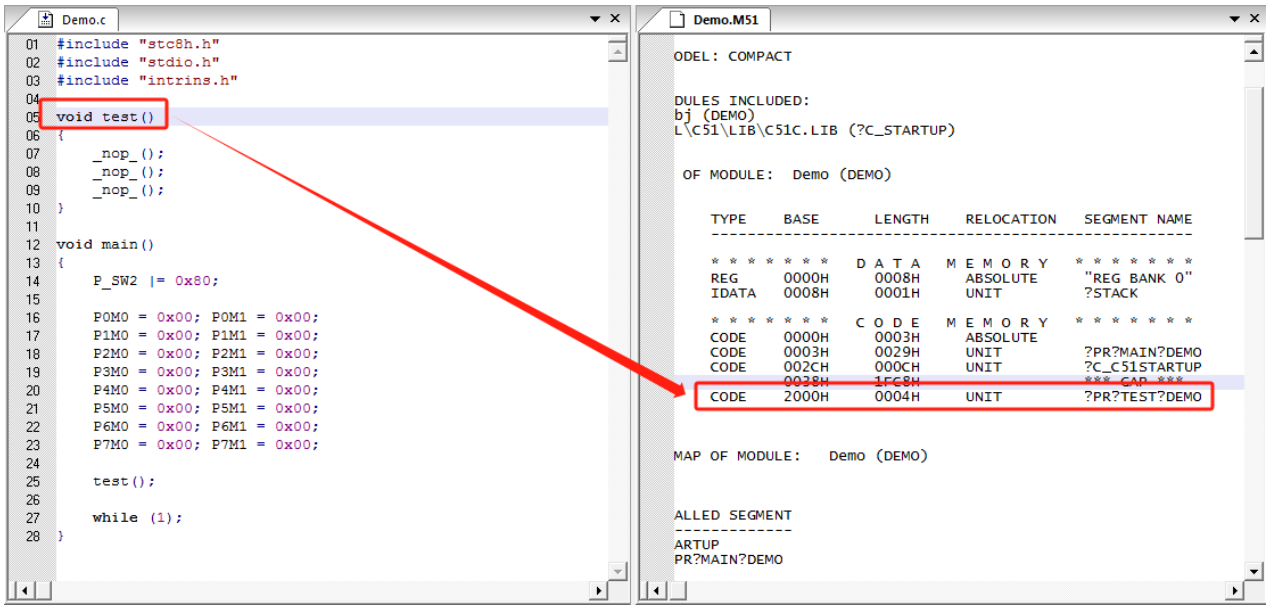
接下来在项目设置项中打开“BL51 Locate”设置页面

在 code 一栏中按照：链接名称 (链接地址)　的格式，输入绝对地址

如下图，将函数指定到代码区的 0x2000 的绝对地址



设置完成后，再次编译，函数即可被链接到指定的绝对地址，如下图：



Demo.c

```
#include "stc8h.h"
#include "stdio.h"
#include "intrins.h"

void test()
{
    _nop_();
    _nop_();
    _nop_();
}

void main()
{
    P_SW2 |= 0x80;

    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
    P3M0 = 0x00; P3M1 = 0x00;
    P4M0 = 0x00; P4M1 = 0x00;
    P5M0 = 0x00; P5M1 = 0x00;
    P6M0 = 0x00; P6M1 = 0x00;
    P7M0 = 0x00; P7M1 = 0x00;

    test();

    while (1);
}
```

Demo.M51

```
ODEL: COMPACT


DULES INCLUDED:
bj (DEMO)
L\C51\LIB\C51C.LIB (?C_STARTUP)


 OF MODULE:  Demo (DEMO)


    TYPE    BASE      LENGTH    RELOCATION   SEGMENT NAME
    -----------------------------------------------------

    * * * * * * *   D A T A   M E M O R Y   * * * * * * *
    REG     0000H     0008H     ABSOLUTE     "REG BANK 0"
    IDATA   0008H     0001H     UNIT         ?STACK

    * * * * * * *   C O D E   M E M O R Y   * * * * * * *
    CODE    0000H     0003H     ABSOLUTE
    CODE    0003H     0029H     UNIT         ?PR?MAIN?DEMO
    CODE    002CH     000CH     UNIT         ?C_C51STARTUP
            0038H     1FC8H                  *** GAP ***
    CODE    2000H     0004H     UNIT         ?PR?TEST?DEMO

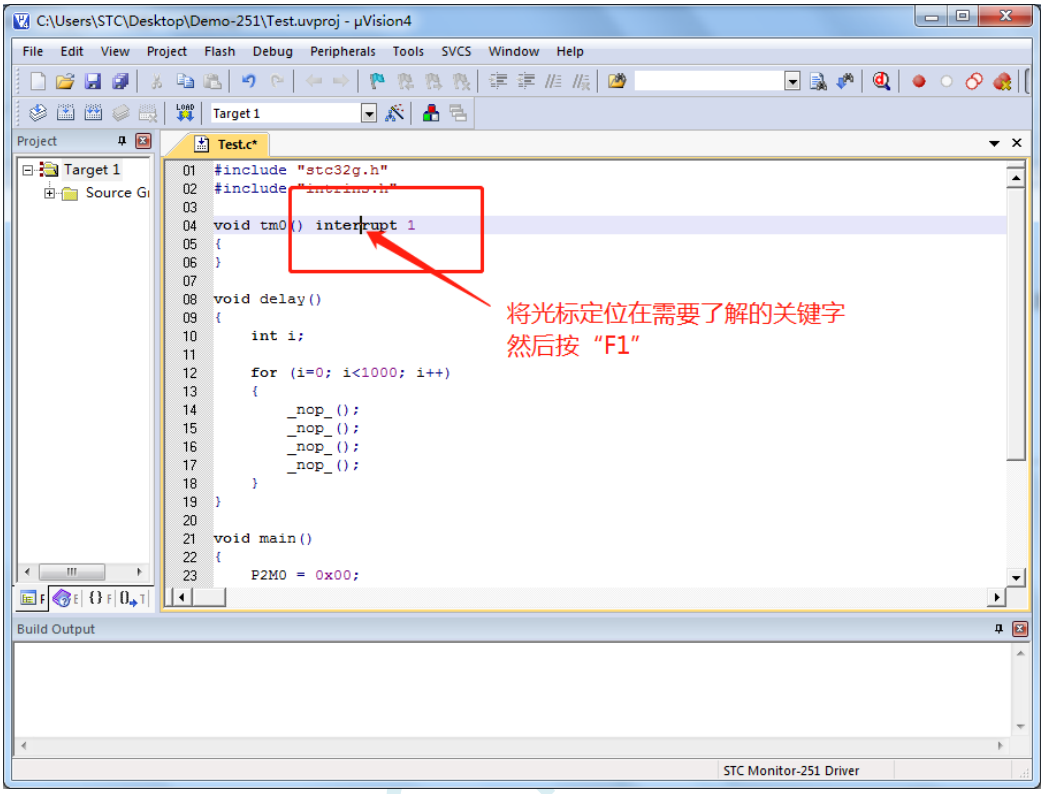

MAP OF MODULE:   Demo (DEMO)

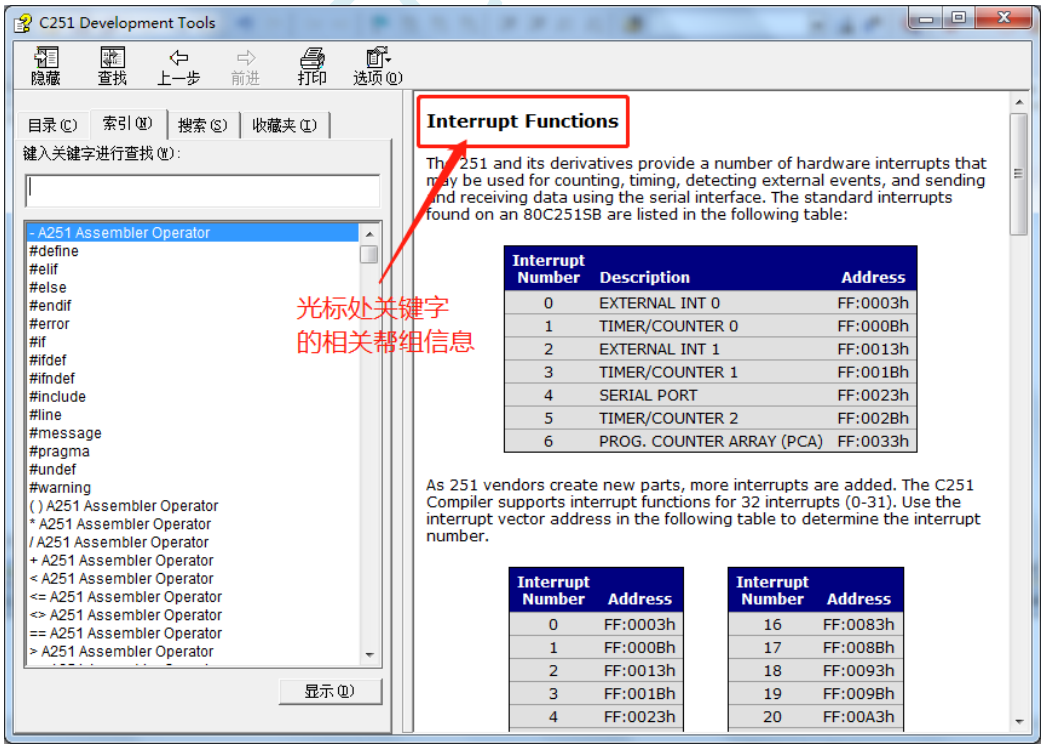

ALLED SEGMENT
-------------
ARTUP
PR?MAIN?DEMO
```

STC MCU

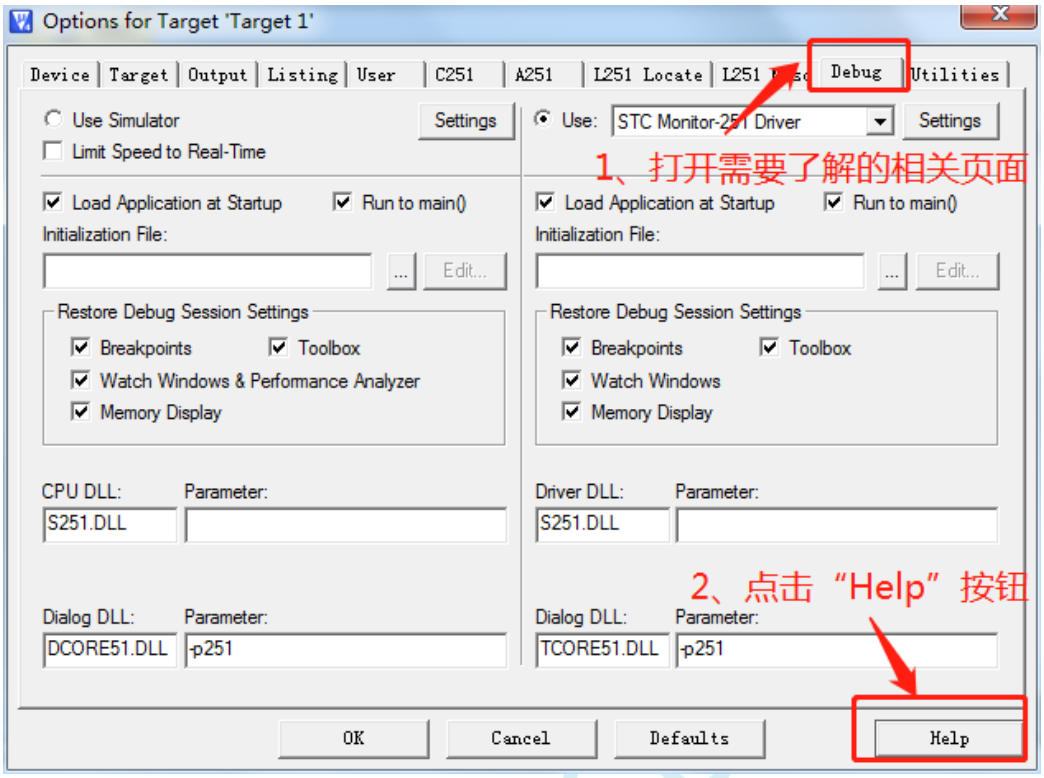
5.6 Keil 软件中获取帮助的简单方法

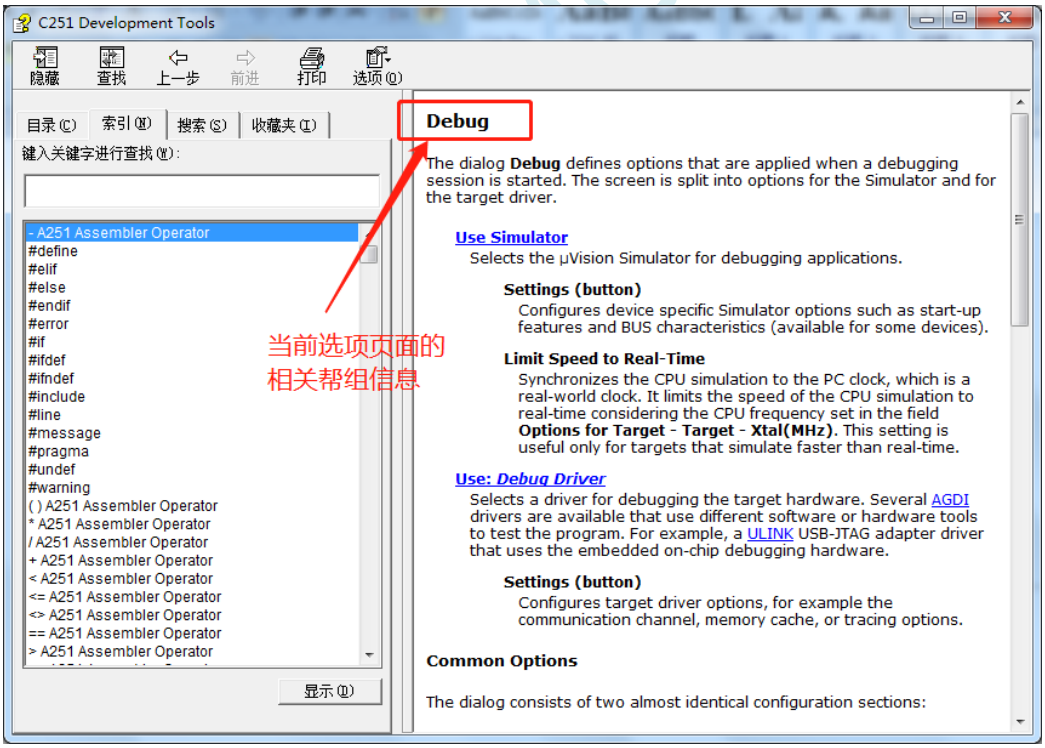
Keil 软件提供了很完整的帮助文件，对于一般的软件使用和编程问题，直接使用 Keil 软件的帮助基本都可以得到解决。如下图：



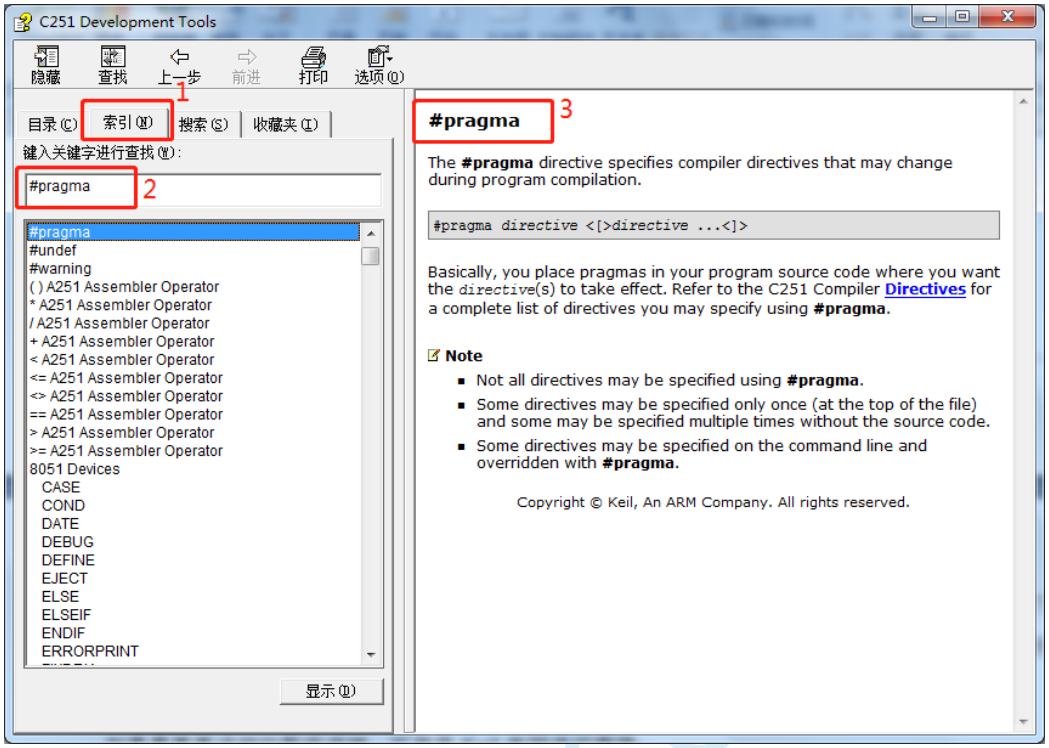


若需要了解项目设置中的相关设置的，可按下图所示的方法获取帮助





另外，也可在帮助窗口中直接输入想了解的内如。比如需要了解如何在程序中设置特殊的编译指示，可按下图所示，在搜索框输入“#pragma”即可

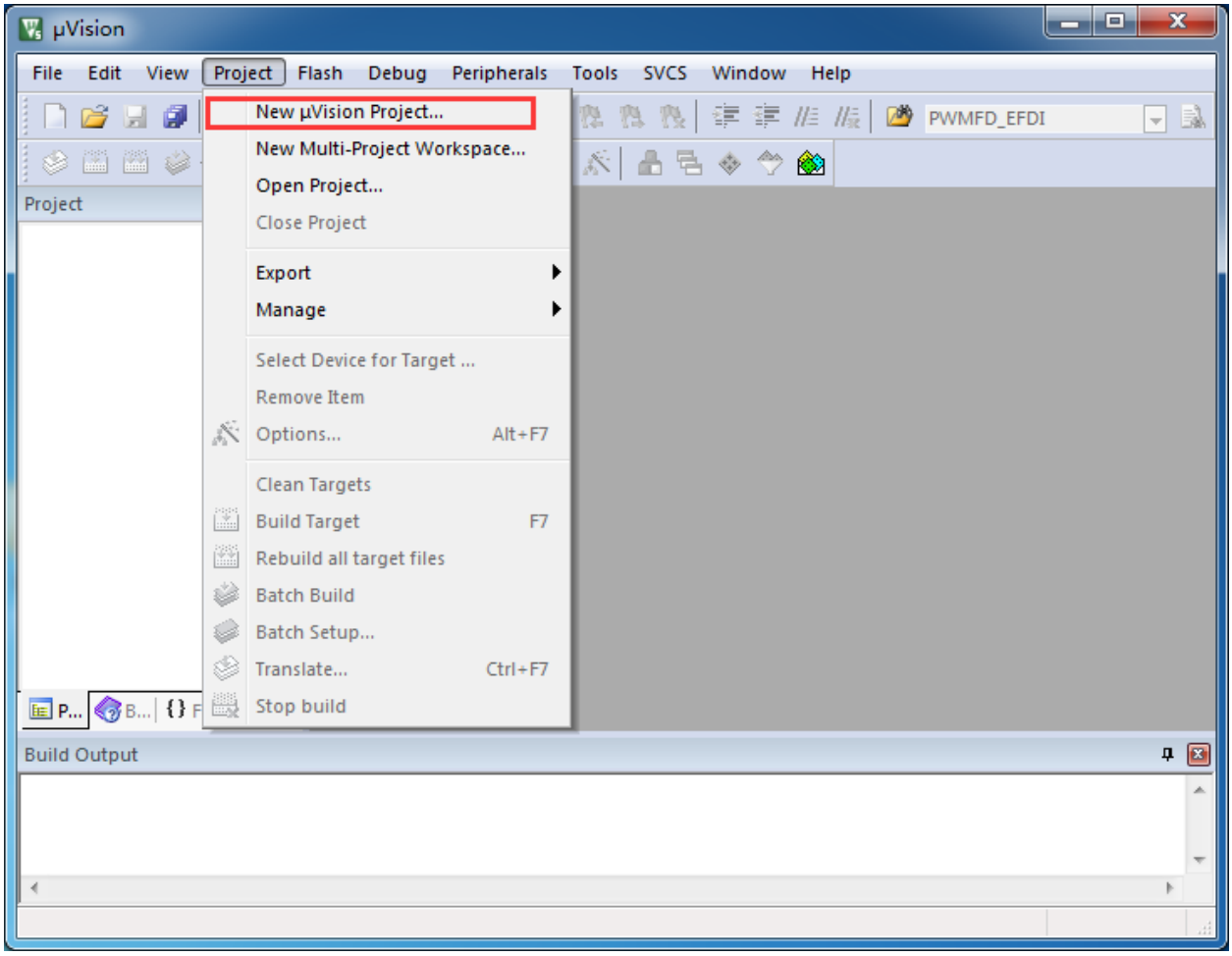


如果需要更详细的帮助详细，可登录 Keil 官网进行查询

5.7　在 Keil 中建立多文件项目的方法

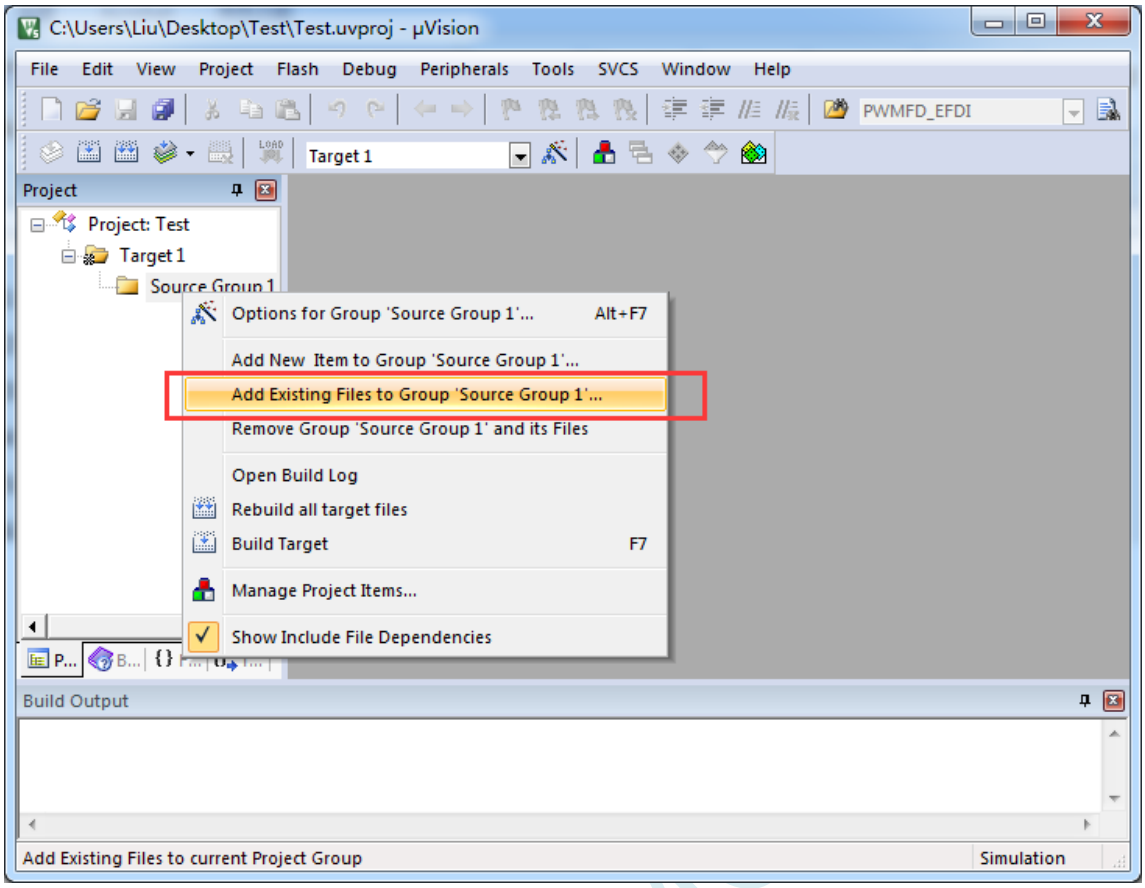
在 Keil 中，一般比较小的项目都只有一个源文件，但对于一些稍微复杂的项目往往需要多个源文件建立多文件项目的方法如下：

1、首先打开 Keil，在菜单“Project”中选择“New uVision Project ...”

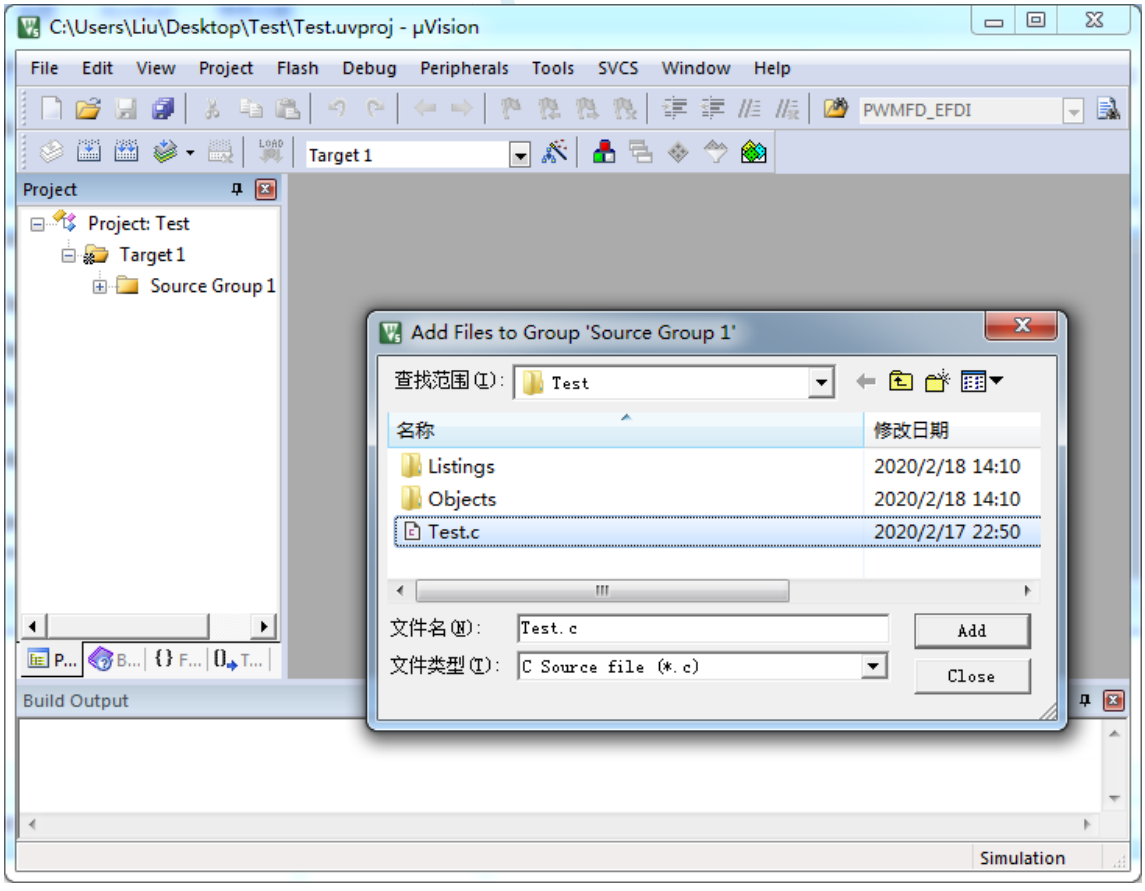


即可完成一个空项目的建立

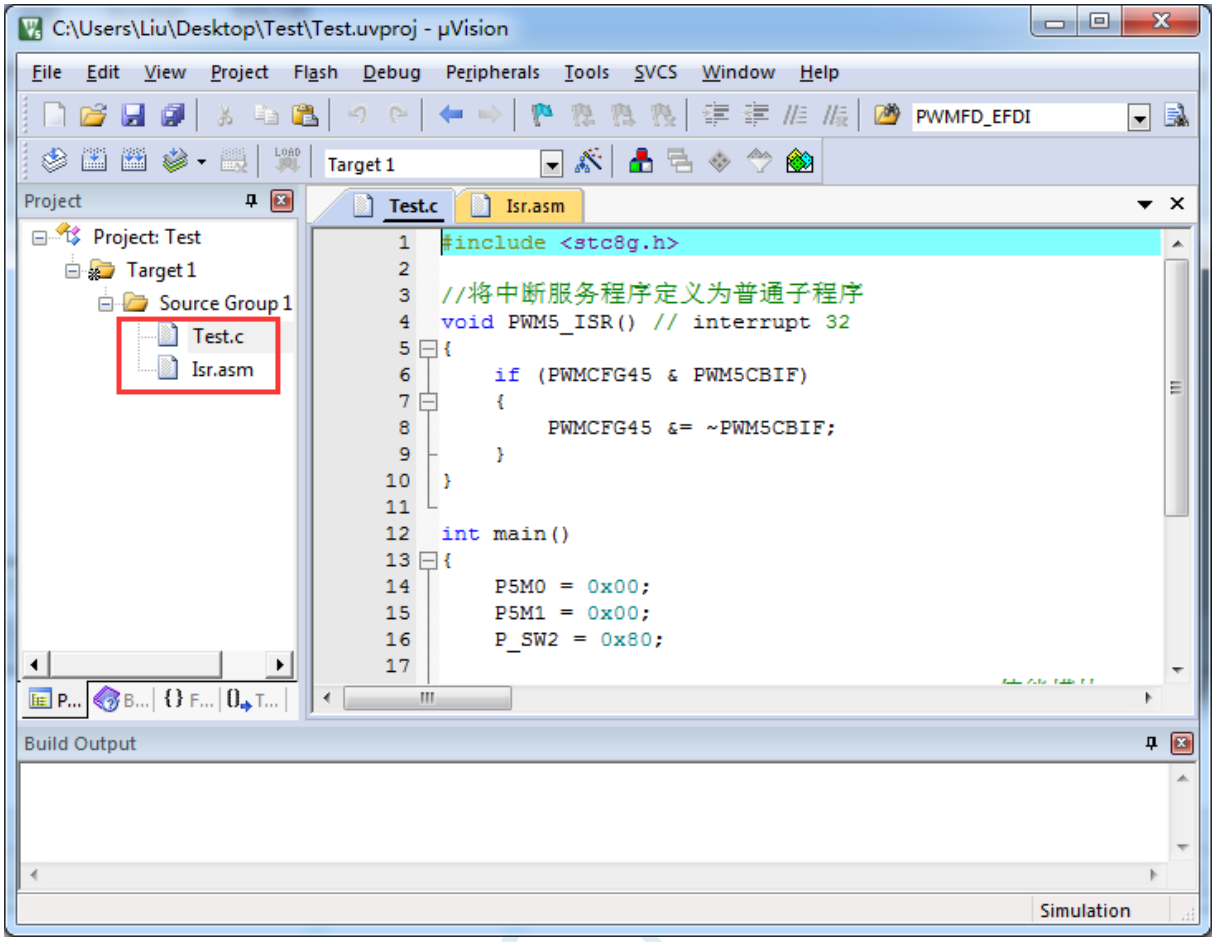
2、在空项目的项目树中，鼠标右键单击“Source Group 1”，并选择右键菜单中的“Add Existing Files to Group "Source Group 1" ...”



3、在弹出的文件对话框中，多次添加源文件



如下图所示即可完成多文件项目的建立



5.8 Keil 小技巧----在多文件项目的源文件中声明并使用外部变量和外部函数

在 Keil 环境下开发比较大的项目时，往往需要在项目中加入多个源文件。而不同源文件内，可能会有一些变量和函数是开放为其它源文件使用的。但默认情况下，在源文件内部定义的变量和函数只在本源文件可见(可使用)，如果需要跨源文件使用则需要遵循一定的规则进行定义和声明。

对于**需要跨源文件访问变量**：

1、变量必须定义为全局变量
所谓全局变量就是在函数外部定义，在函数内部定义的变量为局部变量
由于局部变量的作用域仅在定义的函数内部，且生命周期在函数返回时就结束了
所以**要使变量能跨源文件访问，变量必须定义为全局变量**

2、变量必须定义为非静态变量
使用 static 关键字定义的变量为静态变量，静态变量作用域仅为本源文件，外部源文件不可访问。
所以要使变量能跨源文件访问，变量必须定义为非静态变量，即定义时不要使用 static 关键字

```
13
14  //定义全局变量
15  /*****************************************************
16  非静态的全局变量可以使用extern声明后跨源文件使用
17  不同的源文件不可以定义相同名称的非静态全局变量
18  *****************************************************/
19  int g_iVar1;
20
```

3、不同的源文件中不可定义名称相同的全局变量
如果在不同源文件中定义名称相同的全局变量，编译器会如下错误提示

```
linking...
*** ERROR L104: MULTIPLE PUBLIC DEFINITIONS
```

4、在需要使用外部变量的源文件内使用 extern 声明变量

```
06
07  //使用extern声明全局变量,使变量可跨文件访问
08  extern int g_iVar1;
09
```

如果需要在多个源文件中使用同一个外部变量，可将上面的外部声明语句
放在.h 文件中，在需要使用这个变量的源文件中 include 一下这个头文件即可

同样的，若需要**跨源文件调用外部函数**：

1、函数必须定义为非静态函数
使用 static 关键字定义的函数为静态函数，静态函数作用域仅为本源文件，外部源文件不可访问。
所以要使函数能跨源文件访问，必须定义为非静态函数，即定义函数时不要使用 static 关键字。

```
32
33  //定义非静态函数
34  /*************************************************
35  非静态函数可以使用extern声明后跨源文件使用
36  不同的源文件不可以定义相同名称的非静态函数
37  *************************************************/
38  int func1(int var)
39  {
40      check();
41      return (var * factor);
42  }
43
```

2、不同的源文件中不可定义名称相同的非静态函数

如果在不同源文件中定义名称相同的非静态函数时，编译器会如下错误提示

```
linking...
*** ERROR L104: MULTIPLE PUBLIC DEFINITIONS
```

3、在需要调用外部函数的源文件内直接声明一下即可

对于外部函数声明，是否使用 extern 关键字，编译时都可通过

```
03
04  //声明外部函数,使函数可跨文件调用
05  int func1(int var);
06
```

也可以 extern int func1(int var);

如果需要在多个源文件中使用同一个外部函数，可将上面的外部声明语句放在.h 文件中，在需要调用这个函数的源文件中 include 一下这个头文件即可

5.9 扩展 Keil 对中断号数量的支持，中断号大于 31 编译出错的处理

注：Keil C51/C251 编译器只支持 31 以内的中断号（0~31），超过 31 编译会报错。有热心网友提供了一个简单的 Keil 中断号拓展工具，可将中断号拓展到 254。工具界面如下：

但对于目前现有的 Keil 版本，只能使用本章节的方法进行临时解决。

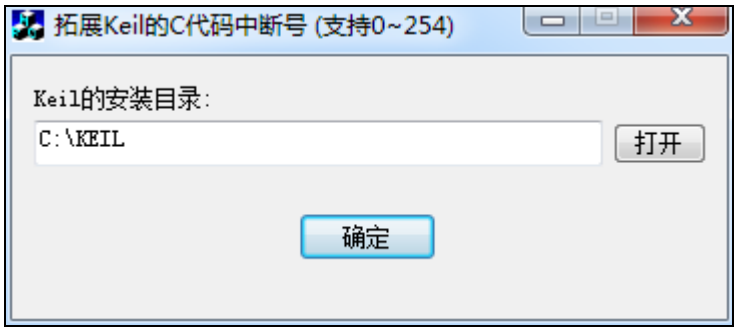
5.9.1 如何下载 Keil 中断号拓展工具，及安装

软件下载地址：https://www.STCAI.com/gjrj



下载完成后，直接解压缩，运行该绿色免安装工具。





点击“打开”按钮，定位到 Keil 的安装目录后，点击“确定”即可。

根据实际的 Keil 安装目录选择地址，如部分用户安装的目录是，C:\Keil_v5

由于 Keil 的版本在不断更新，而早期版本过多，有无法收集齐，这里列举一下已测试通过的 C51.EXE 版本和 C251.EXE 版本

已测试通过的 C51.EXE 版本：

V6.12.0.1
V8.8.0.1
V9.0.0.1
V9.1.0.1
V9.53.0.0
V9.54.0.0
V9.57.0.0
V9.59.0.0
V9.60.0.0

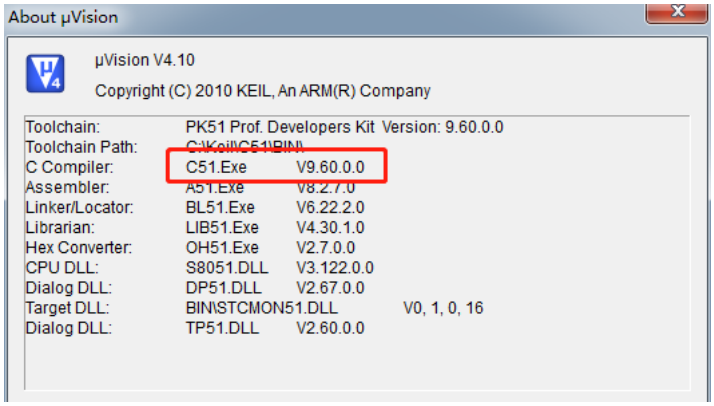
已测试通过的 C251.EXE 版本：

V5.57.0.0
V5.60.0.0

STC MCU

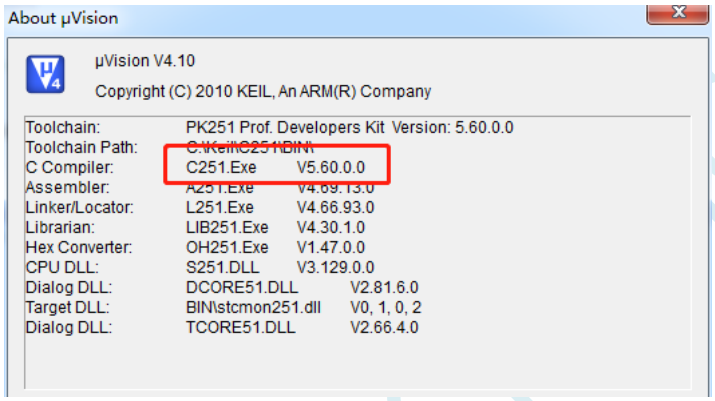
查看 C51.EXE 版本的方法：

在 keil 中打开一个基于 STC8 系列或者 STC15 系列单片机的项目，在 Keil 软件菜单项“Help”中打开“About uVision...”



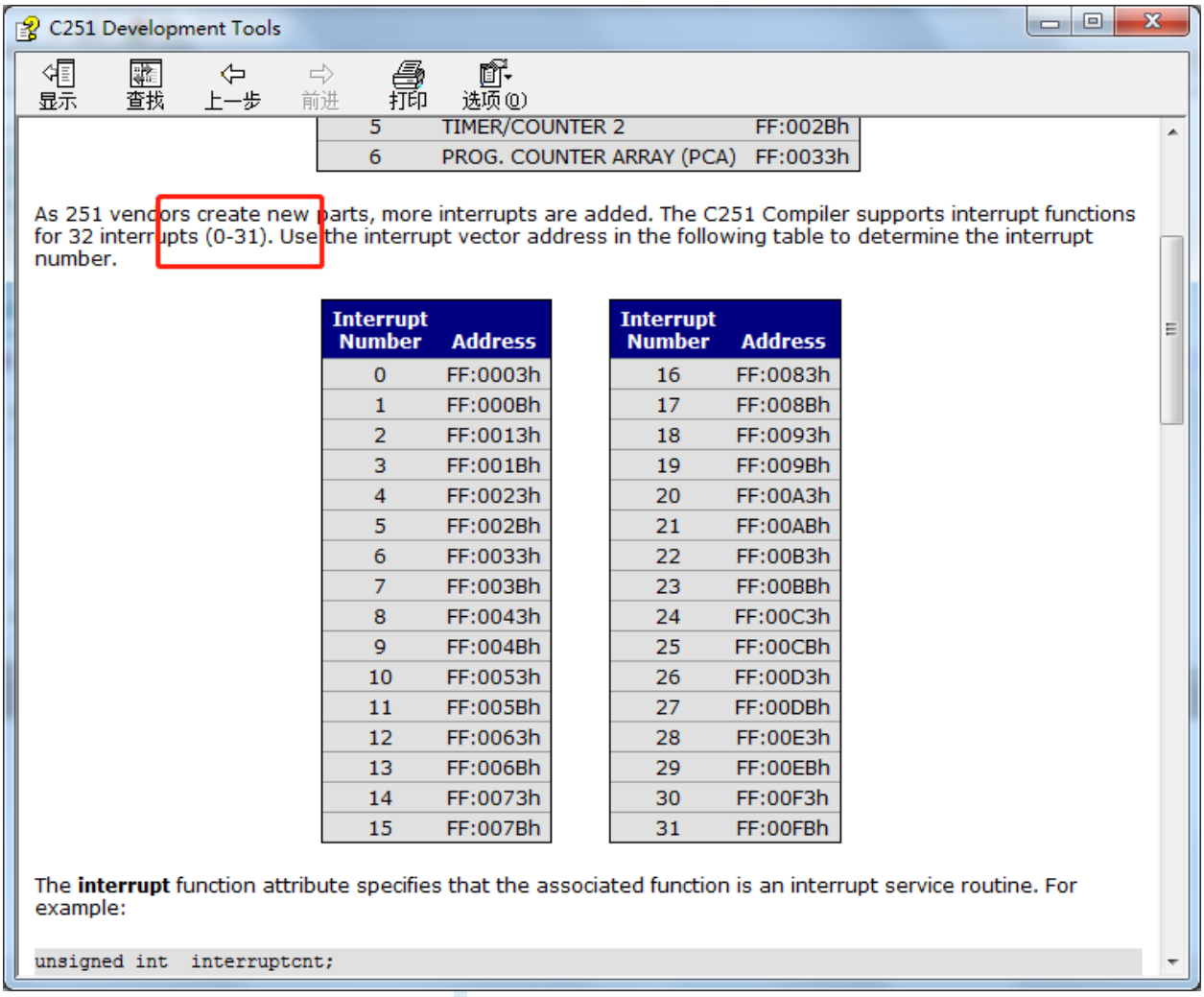
查看 C251.EXE 版本的方法：

在 keil 中打开一个基于 STC32G 系列单片机的项目，在 Keil 软件菜单项“Help”中打开“About uVision...”



5.9.2 如不使用扩展中断号工具，则需借用保留中断号进行中转

在 Keil 的 C251 编译环境下，中断号只支持 0~31，即中断向量必须小于 0100H。

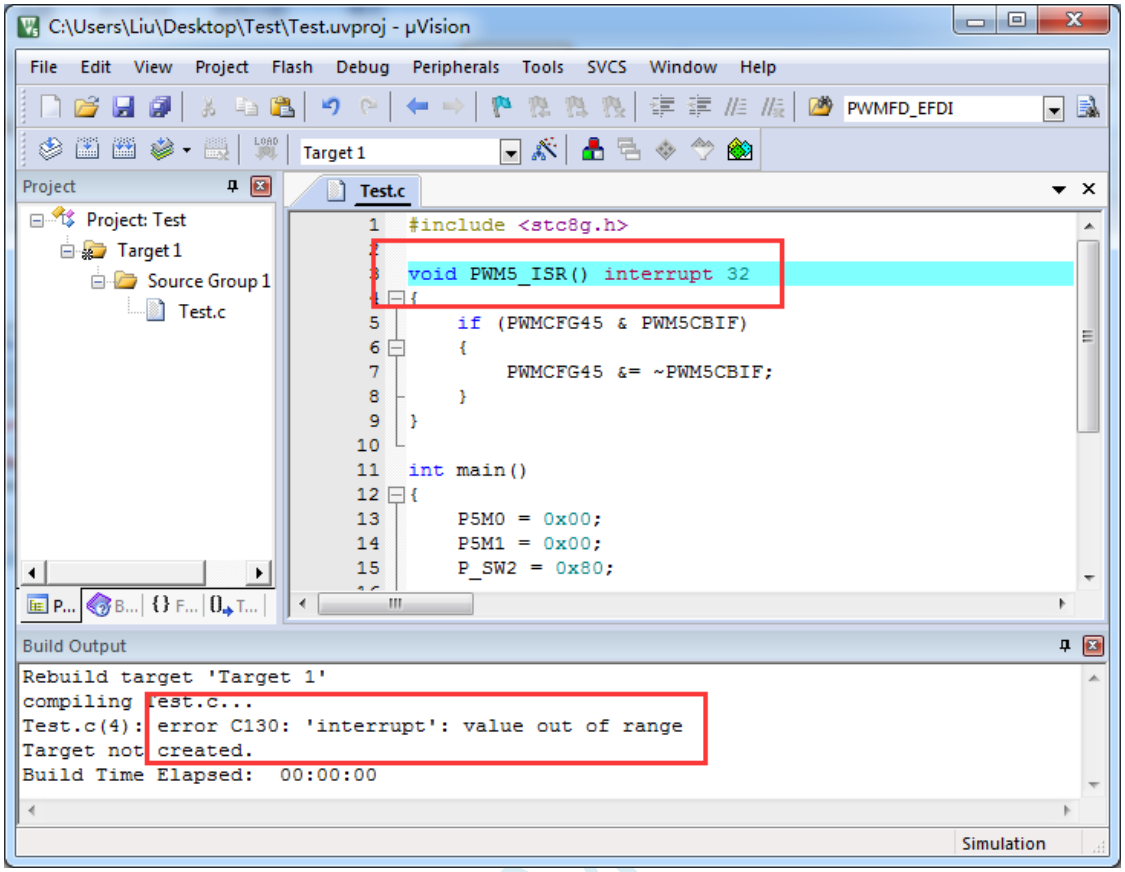


下表是 STC 目前所有系列的中断列表：

中断号	中断向量	中断类型
0	0003 H	INT0
1	000B H	定时器 0
2	0013 H	INT1
3	001B H	定时器 1
4	0023 H	串口 1
5	002B H	ADC
6	0033 H	LVD
8	0043 H	串口 2
9	004B H	SPI
10	0053 H	INT2
11	005B H	INT3
12	0063 H	定时器 2
13	**006B H**	
14	0073 H	系统内部中断
15	007B H	系统内部中断

中断号	中断向量	中断类型
16	0083 H	INT4
17	008B H	串口 3
18	0093 H	串口 4
19	009B H	定时器 3
20	00A3 H	定时器 4
21	00AB H	比较器
24	00C3 H	I2C
25	00CB H	USB
26	00D3 H	PWMA
27	00DB H	PWMB
28	00E3 H	CAN1
29	00EB H	CAN2
30	00F3 H	LIN
36	**0123 H**	**RTC**
37	**012B H**	**P0 口中断**
38	**0133 H**	**P1 口中断**
39	**013B H**	**P2 口中断**
40	**0143 H**	**P3 口中断**
41	**014B H**	**P4 口中断**
42	**0153 H**	**P5 口中断**
43	**015B H**	**P6 口中断**
44	**0163 H**	**P7 口中断**
45	**016B H**	**P8 口中断**
46	**0173 H**	**P9 口中断**
47	**017BH**	**M2M DMA 中断**
48	**0183H**	**ADC DMA 中断**
49	**018BH**	**SPI DMA 中断**
50	**0193H**	**UR1T DMA 中断**
51	**019BH**	**UR1R DMA 中断**
52	**01A3H**	**UR2T DMA 中断**
53	**01ABH**	**UR2R DMA 中断**
54	**01B3H**	**UR3T DMA 中断**
55	**01BBH**	**UR3R DMA 中断**
56	**01C3H**	**UR4T DMA 中断**
57	**01CBH**	**UR4R DMA 中断**
58	**01D3H**	**LCM DMA 中断**
59	**01DBH**	**LCM 中断**
60	**01E3H**	**I2CT DMA 中断**
61	**01EBH**	**I2CR DMA 中断**
62	**01F3H**	**I2S 中断**
63	**01FBH**	**I2ST DMA 中断**
64	**0203H**	**I2SR DMA 中断**

不难发现，RTC 中断开始，后面所有的中断服务程序，在 keil 中均会编译出错，如下图所示：



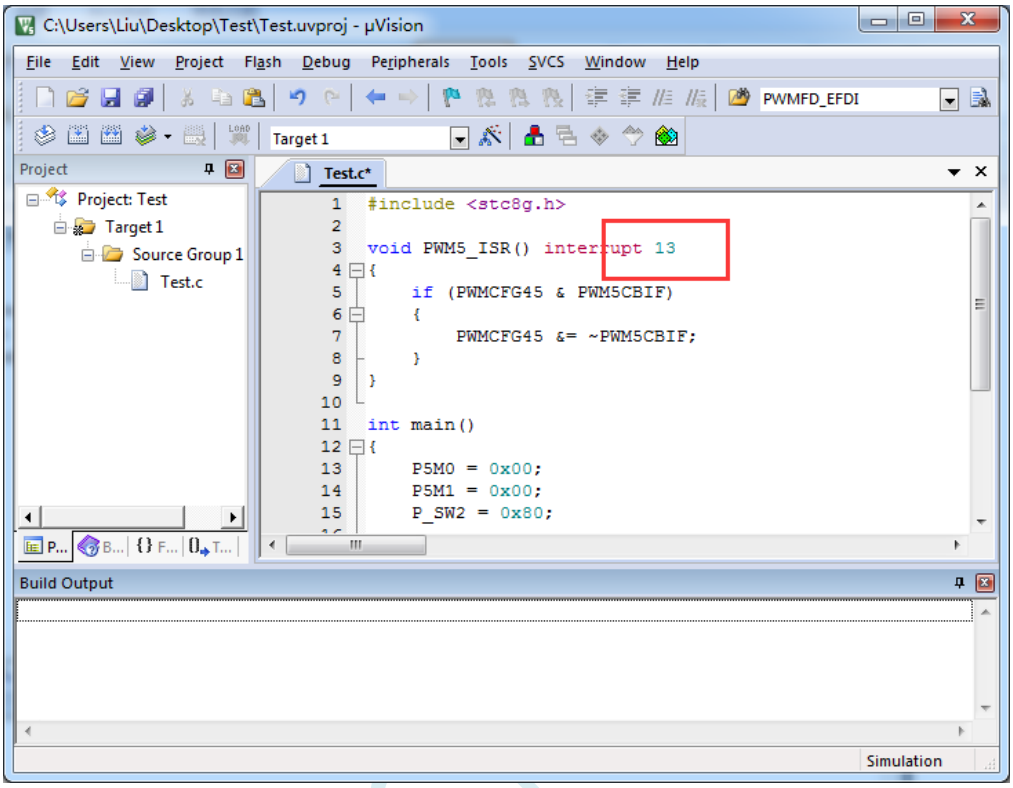
处理这种错误有如下三种方法：（均需要借助于汇编代码，优先推荐使用方法 1）

方法 1：借用 13 号中断向量

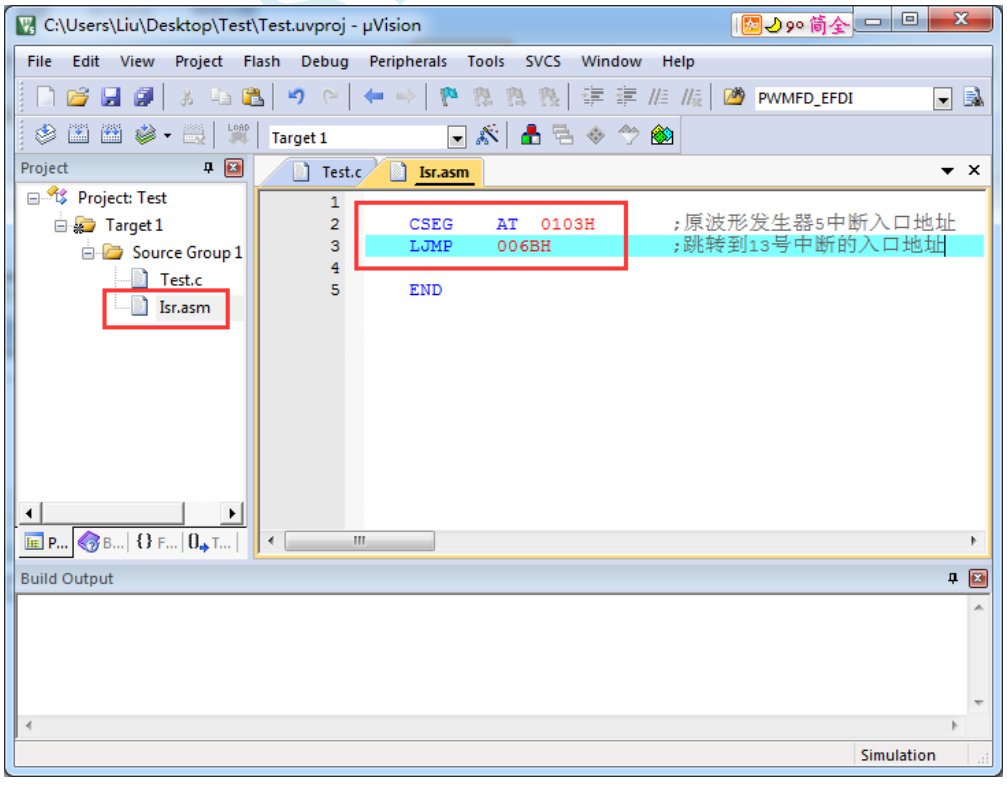
0~31 号中断中，第 13 号是保留中断号，我们可以借用此中断号

操作步骤如下：

1、将我们报错的中断号改为“13”，如下图：

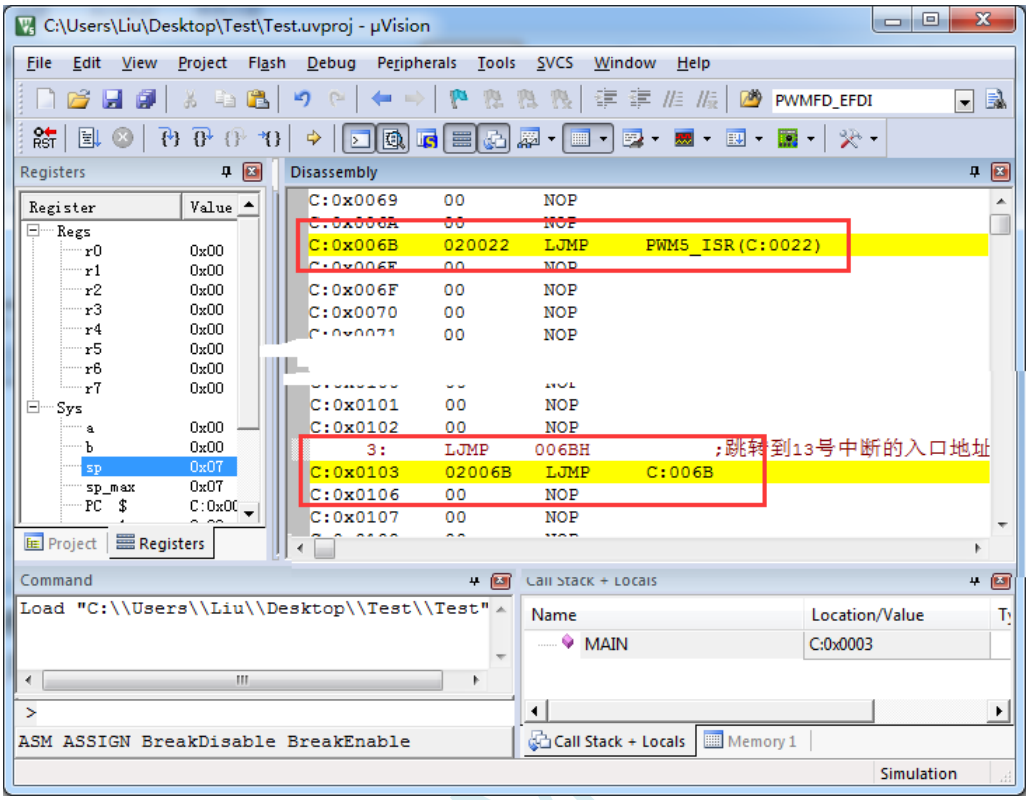


2、新建一个汇编语言文件，比如“isr.asm”，加入到项目，并在地址“0103H”的地方添加一条“LJMP 006BH”，如下图：

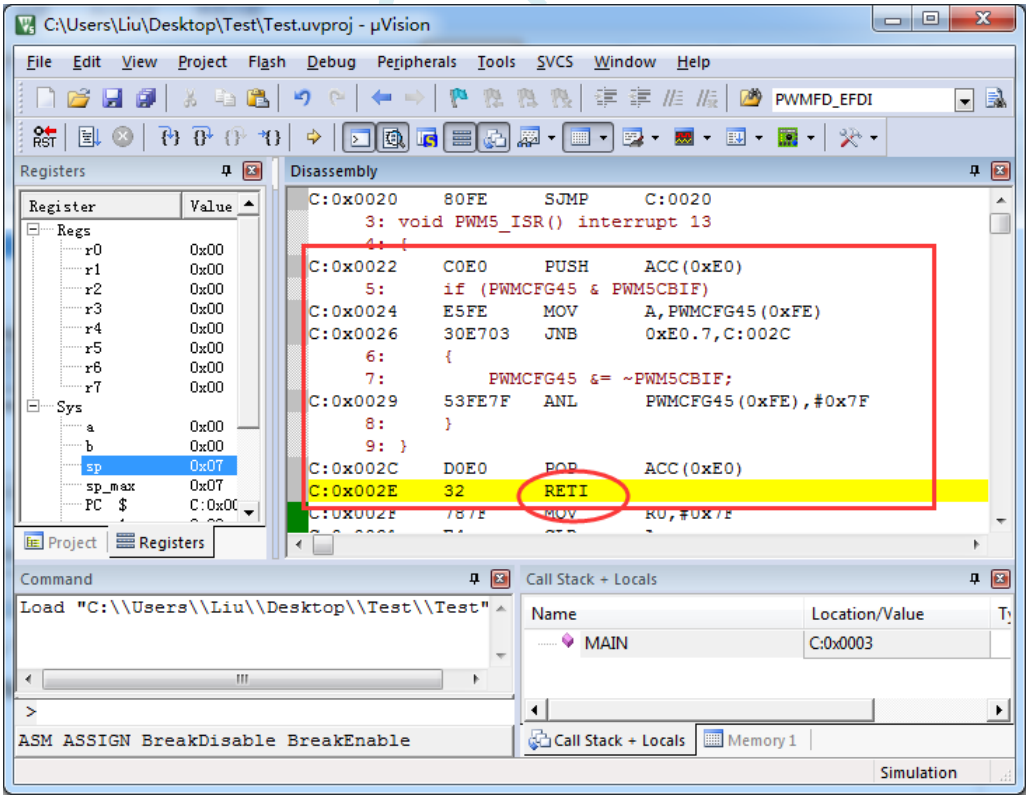


3、编译即可通过。

此时经过 Keil 的 C51 编译器编译后，在 006BH 处有一条“LJMP　PWM5_ISR”，在 0103H 处有一条“LJMP　006BH”，如下图：



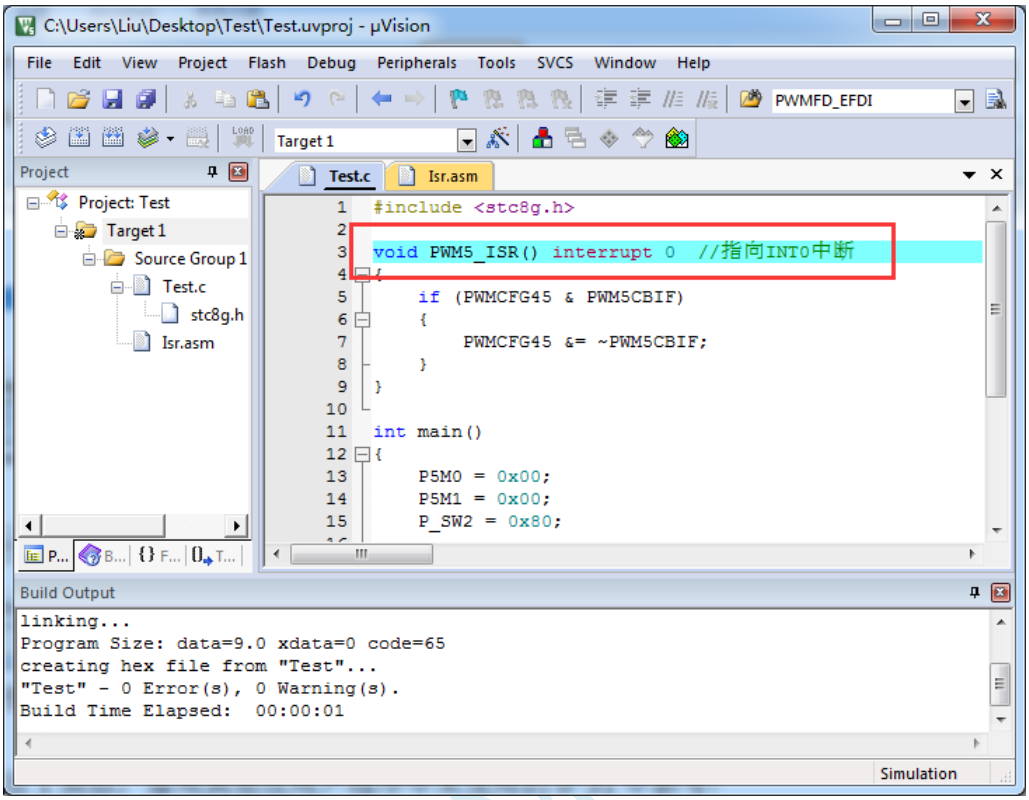
当发生 PWM5 中断时，硬件会自动跳转到 0103H 地址执行“LJMP　006BH”，然后在 006BH 处再执行“LJMP　PWM5_ISR”即可跳转到真正的中断服务程序，如下图：

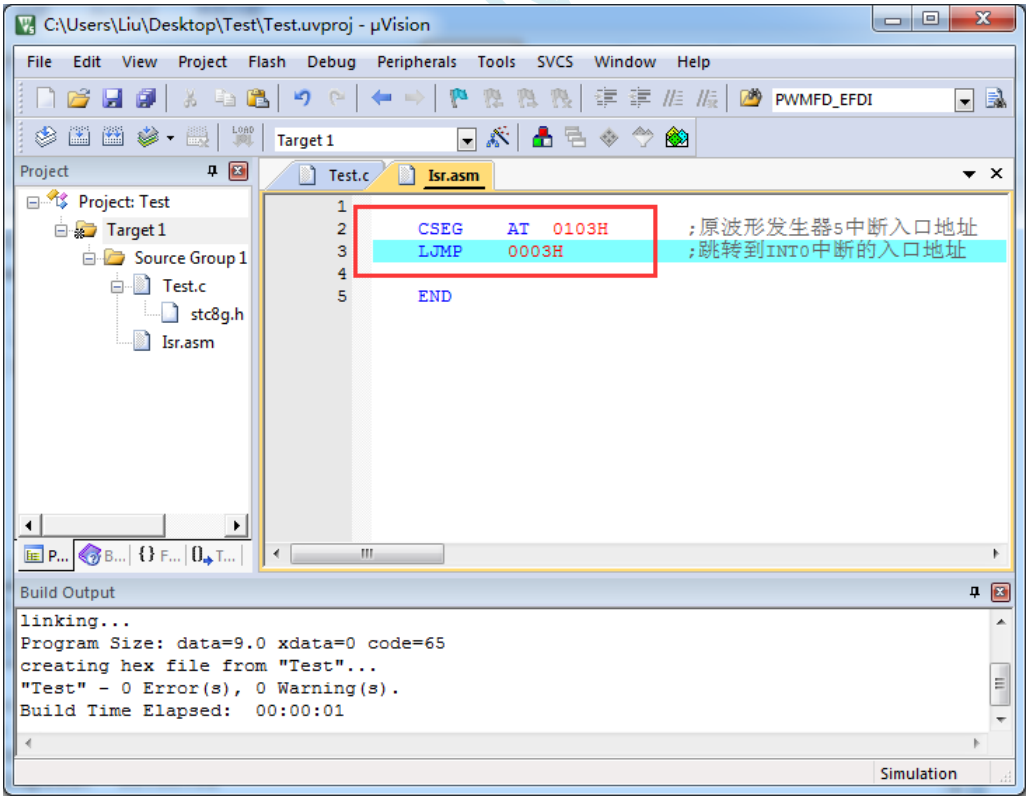


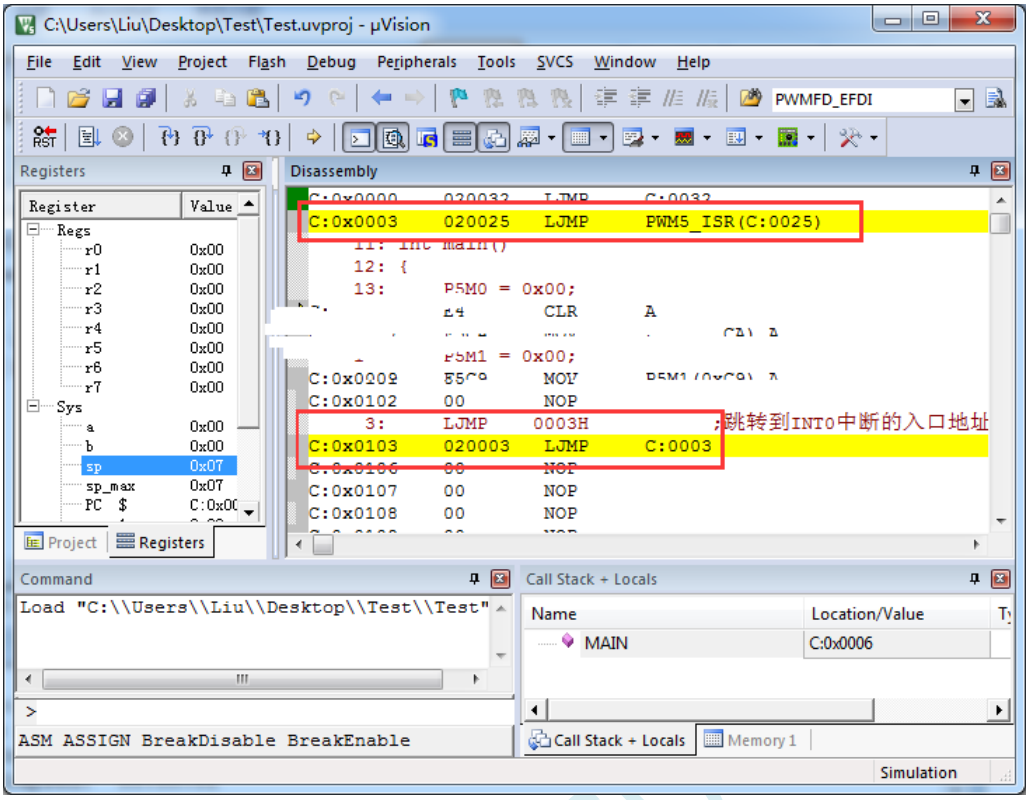
中断服务程序执行完成后，再通过 RETI 指令返回。整个中断响应过程只是多执行了一条 LJMP 语句而已。

方法 2：与方法 1 类似，借用用户程序中未使用的 0~31 的中断号

比如在用户的代码中，没有使用 INT0 中断，则可将上面的代码作类似与方法 1 的修改：





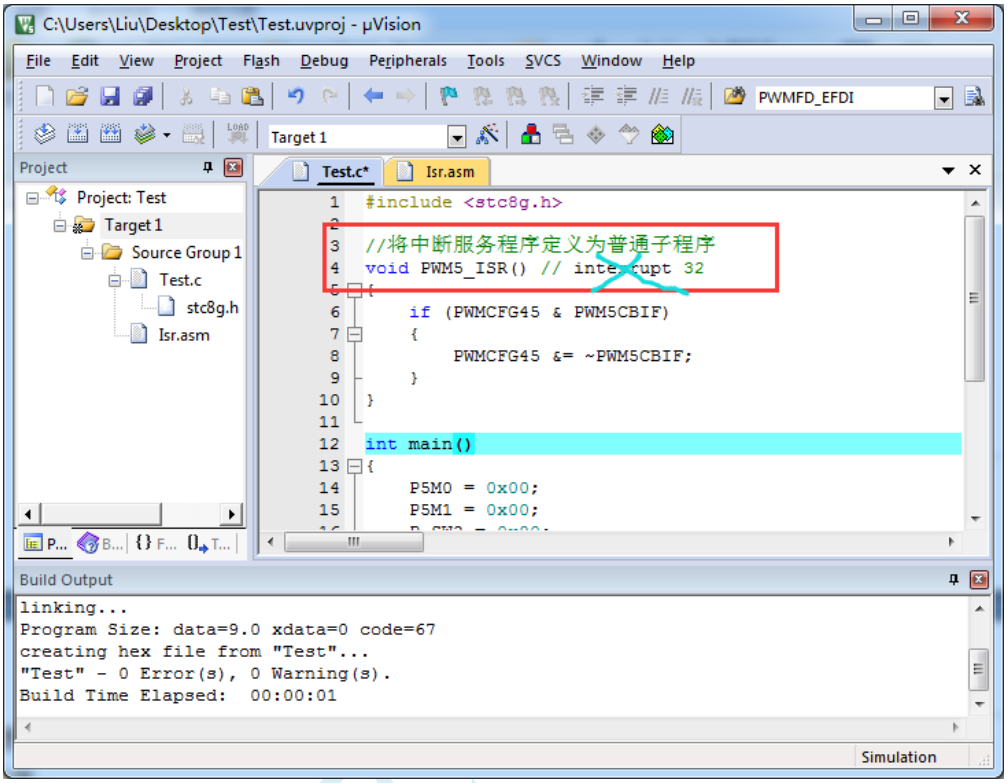


执行效果与方法 1 相同，此方法适用于需要重映射多个中断号大于 31 的情况。

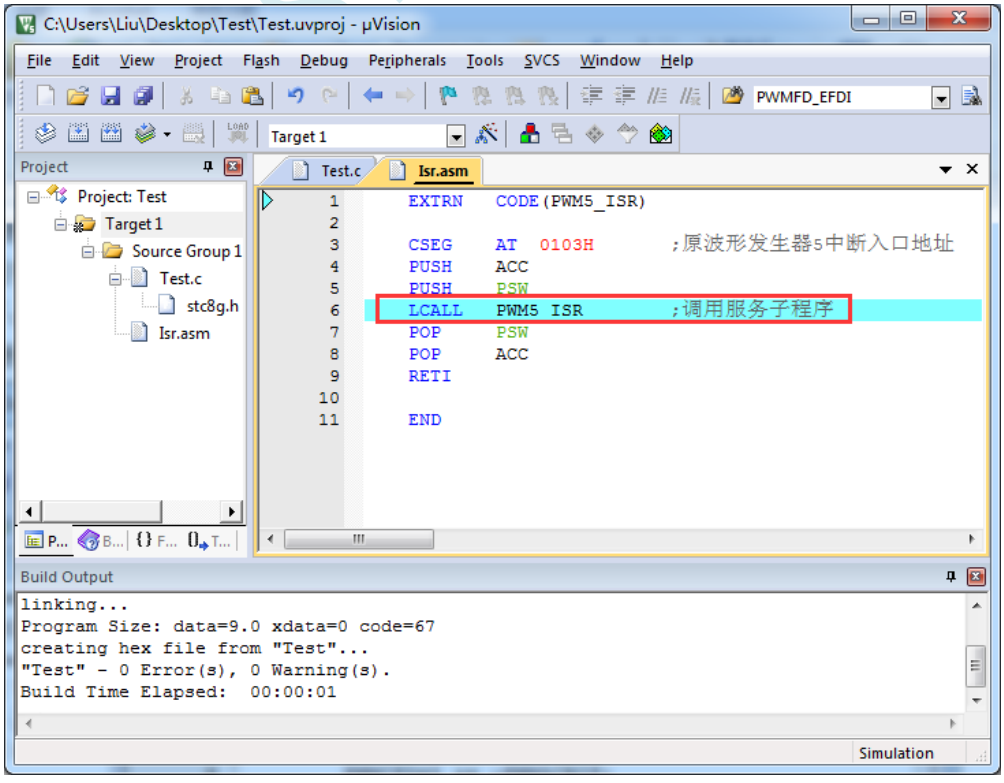
方法 3：将中断服务程序定义成子程序，然后在汇编代码中的中断入口地址中使用 LCALL 指令执行服务程序

操作步骤如下：

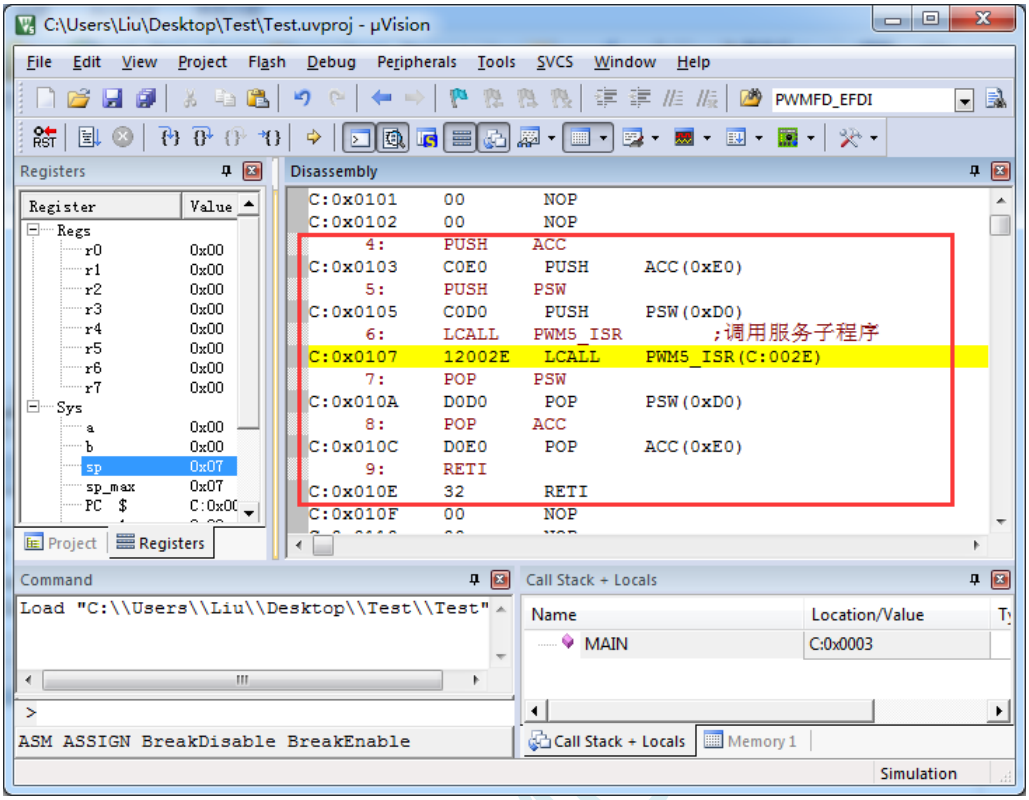
1、首先将中断服务程序去掉“interrupt”属性，定义成普通子程序



2、然后在汇编文件的 0103H 地址输入如下图所示的代码



3、编译通过后，即可发现在 0103H 地址的地方即为中断服务程序



此方法不需要重映射中断入口，不过这种方法有一个问题，在汇编文件中具体需要将哪些寄存器压入堆栈，需要用户查看 C 程序的反汇编代码来确定。一般包括 PSW、ACC、B、DPL、DPH 以及 R0~R7。除 PSW 必须压栈外，其他哪些寄存器在用户子程序中有使用，就必须将哪些寄存器压栈。

5.10 用户程序复位到系统区进行 USB 模式 ISP 下载的方法（不停电）

当项目处于开发阶段时，需要反复的下载用户代码到目标芯片中进行代码验证，使用 USB 模式对 STC 的单片机进行正常的 ISP 下载，需要先将 P3.2 口短路到 GND，然后对目标芯片进行重新上电，从而会使得项目在开发阶段烧录步骤比较繁琐。为此 STC 单片机增加了一个特殊功能寄存器 IAP_CONTR，当用户向此寄存器写入 0x60，即可实现软件复位到系统区，进而实现不停电就可进行 ISP 下载。

注：当用户程序软复位到系统区时，若 P3.0/D-和 P3.1/D+已经和电脑的 USB 口相连，则系统代码会自动进入 USB 下载模式等待 ISP 下载，此时不需要 P3.2 连接到地

下面介绍如下两种方法：

1、使用 P3.2 口的按键（非 USB 项目）

这里使用 P3.2 口的按键触发软复位和“P3.2 口短路到 GND，然后对目标芯片进行重新上电”的方法不一样。用户程序的主循环中，判断 P3.2 口电平状态，当检测到 P3.2 口电平为 0 时，触发软件复位到系统区即可进行 USB ISP 下载。P3.2 口的按键在释放状态时，用户程序从 P3.2 口读取的电平为 1，当需要复位到 ISP 进行 USB 下载时，只需手动按一下 P3.2 即可。

程序中判断 P3.2 电平的范例程序如下：

```
//测试工作频率为 11.0592MHz

//#include "stc8h.h"
#include "stc32g.h"                     //头文件见下载软件

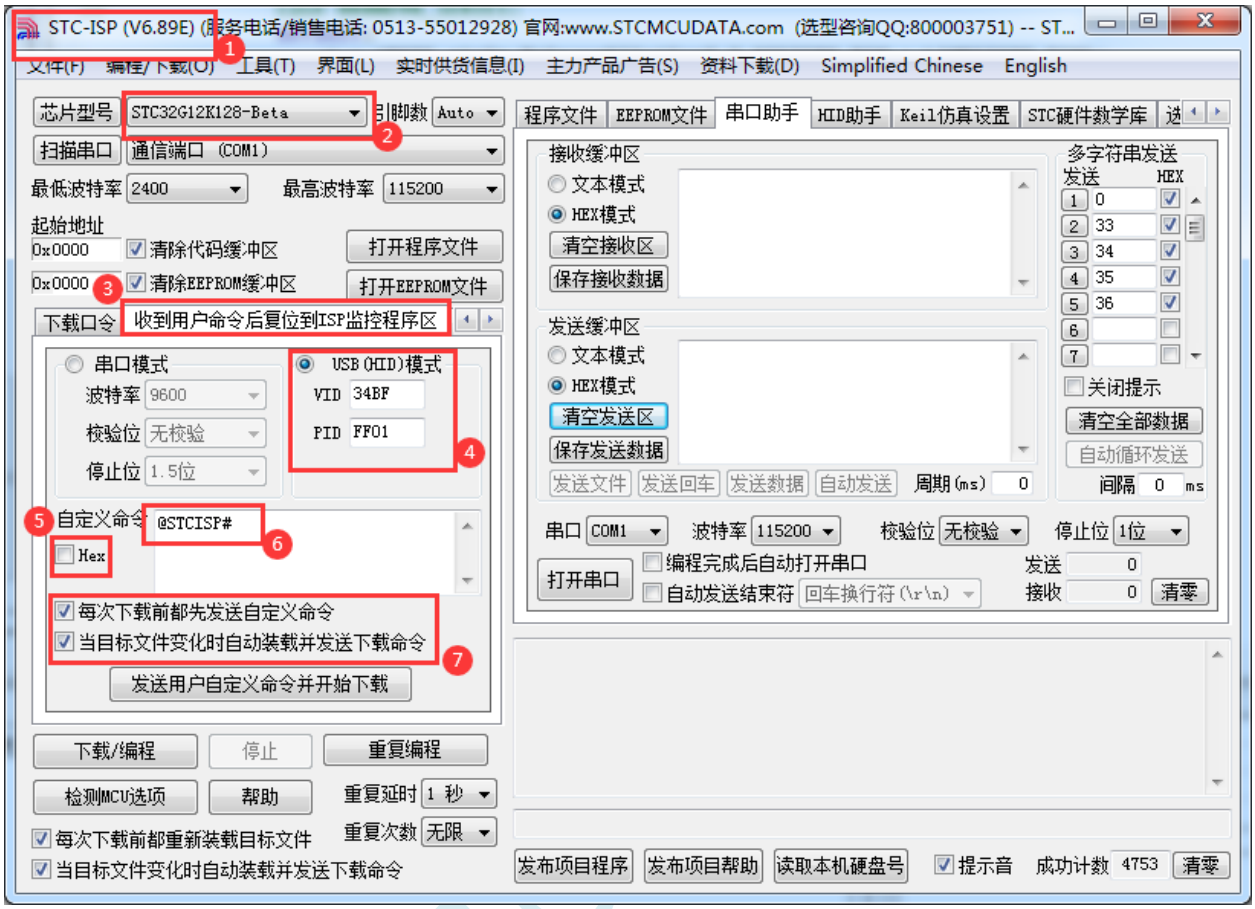
void main()
{
    EAXFR = 1;                          //使能访问 XFR
    CKCON = 0x00;                       //设置外部数据总线速度为最快
    WTST = 0x00;                        //设置程序代码等待参数，
                                        //赋值为 0 可将 CPU 执行程序的速度设置为最快

    P3M0 = 0x00;
    P3M1 = 0x00;
    P32 = 1;

    while (1)
    {
        if (!P32) IAP_CONTR = 0x60;     //当检测到 P3.2 的电平为低时
                                        //软件复位到系统区

        ...                             //用户代码
    }
}
```

2、使用 AIapp-ISP 下载软件发送的用户下载命令（USB 项目）



1、下载最新版本的 AIapp-ISP 下载软件

2、选择正确的单片机型号

3、打开“收到用户命令后复位到 ISP 监控程序区”选项页

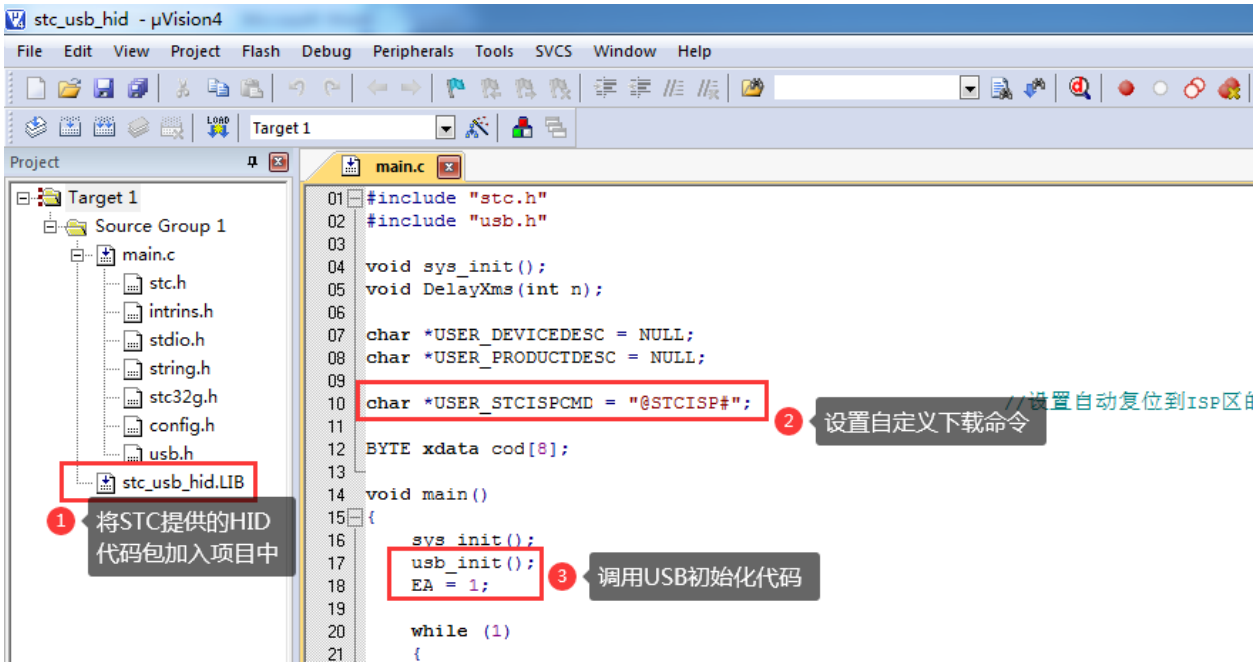
4、选择“USB(HID)模式”，并设置 USB 设备的 VID 和 PID，STC 提供的范例中的 VID 为“34BF”，PID 为“FF01”

5、选择 HEX 模式或者文本模式

6、设置自定义下载命令，需要和代码中的自定义命令相一致

7、选择上这两项，当目标代码重新编译后，AIapp-ISP 下载软件便会自动发送复位命令，并自动开始 USB 模式的 ISP 下载

注意：若需要使用此模式，则必须将 STC 提供的“stc_usb_hid.lib”代码库添加到项目中，并按照下图所示的方式设置自定义下载命令。

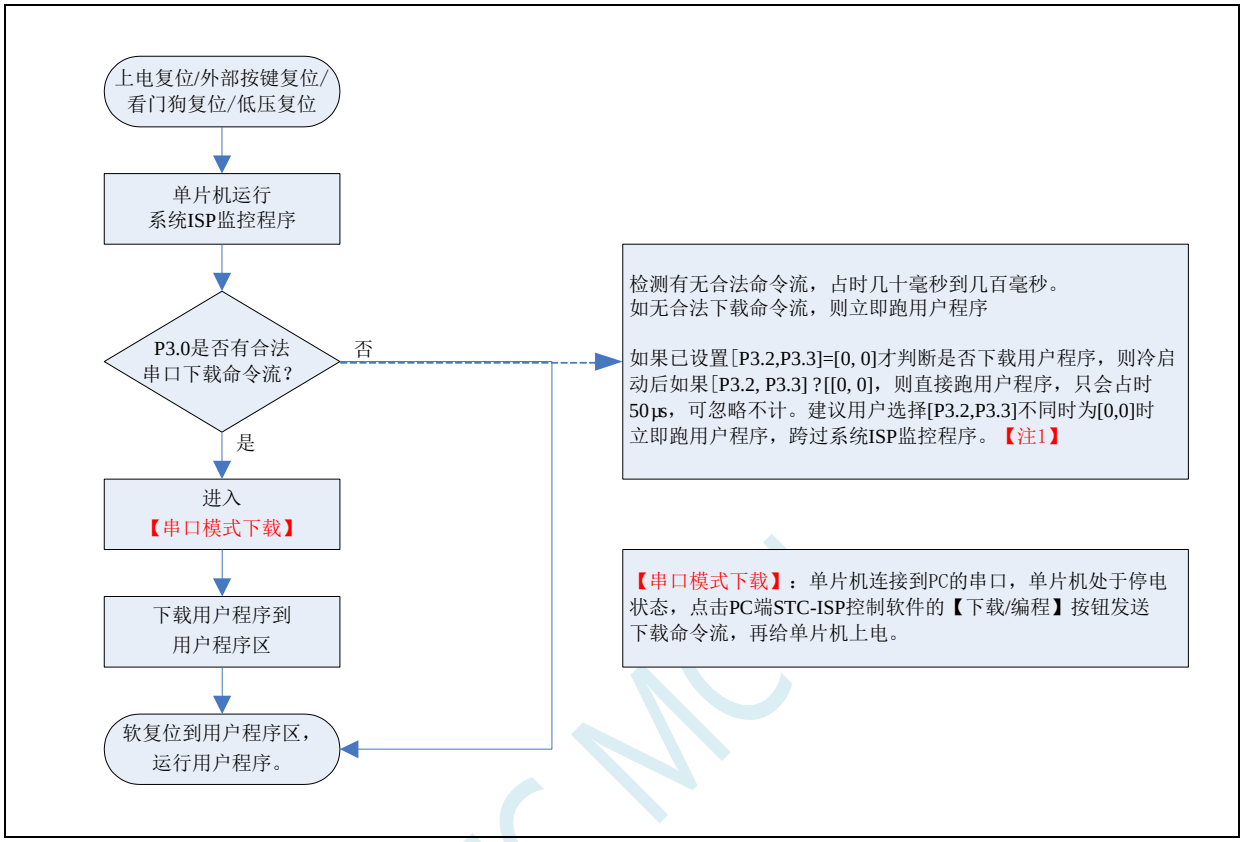


详细代码请参考官网上的“STC32G 实验箱演示程序”包中的“76-通过 USB HID 协议打印数据信息-可用于调试”

STC MCU

5.11 ISP 下载流程及典型应用线路图

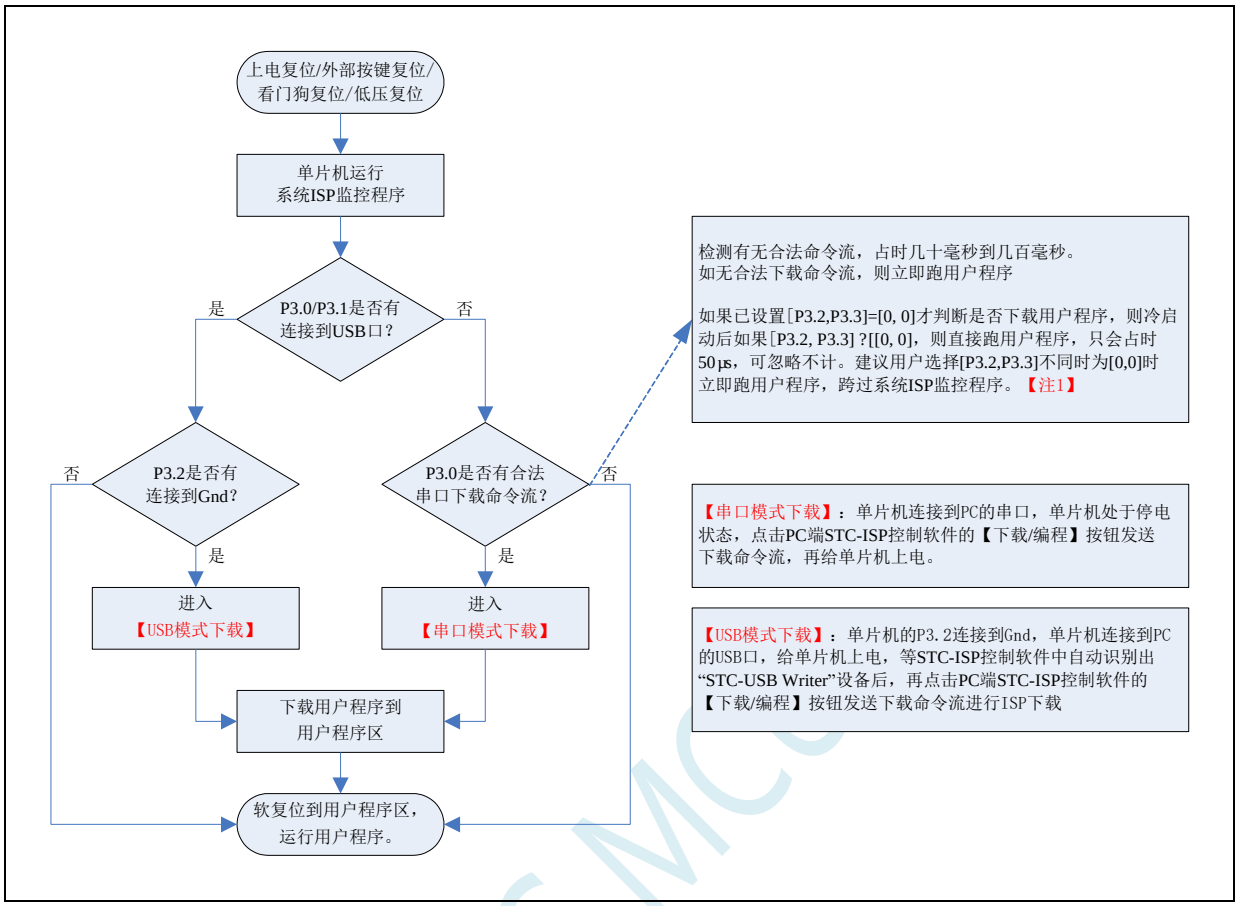
5.11.1 ISP 下载流程图（串口下载模式）



注意：因 [P3.0, P3.1] 作下载/仿真用(下载/仿真接口仅可用 [P3.0, P3.1])，故建议用户将串口 1 放在 P3.6/P3.7 或 P1.6/P1.7 ，若用户不想切换，坚持使用 P3.0/P3.1 工作或作为串口 1 进行通信，则务必在下载程序时，在软件上勾选“下次冷启动时，P3.2/P3.3 为 0/0 时才可以下载程序”。

【注 1】：STC15，STC8 系列及以后新出的芯片的烧录保护引脚为 P3.2/P3.3，之前早期芯片的烧录保护引脚为 P1.0/P1.1。

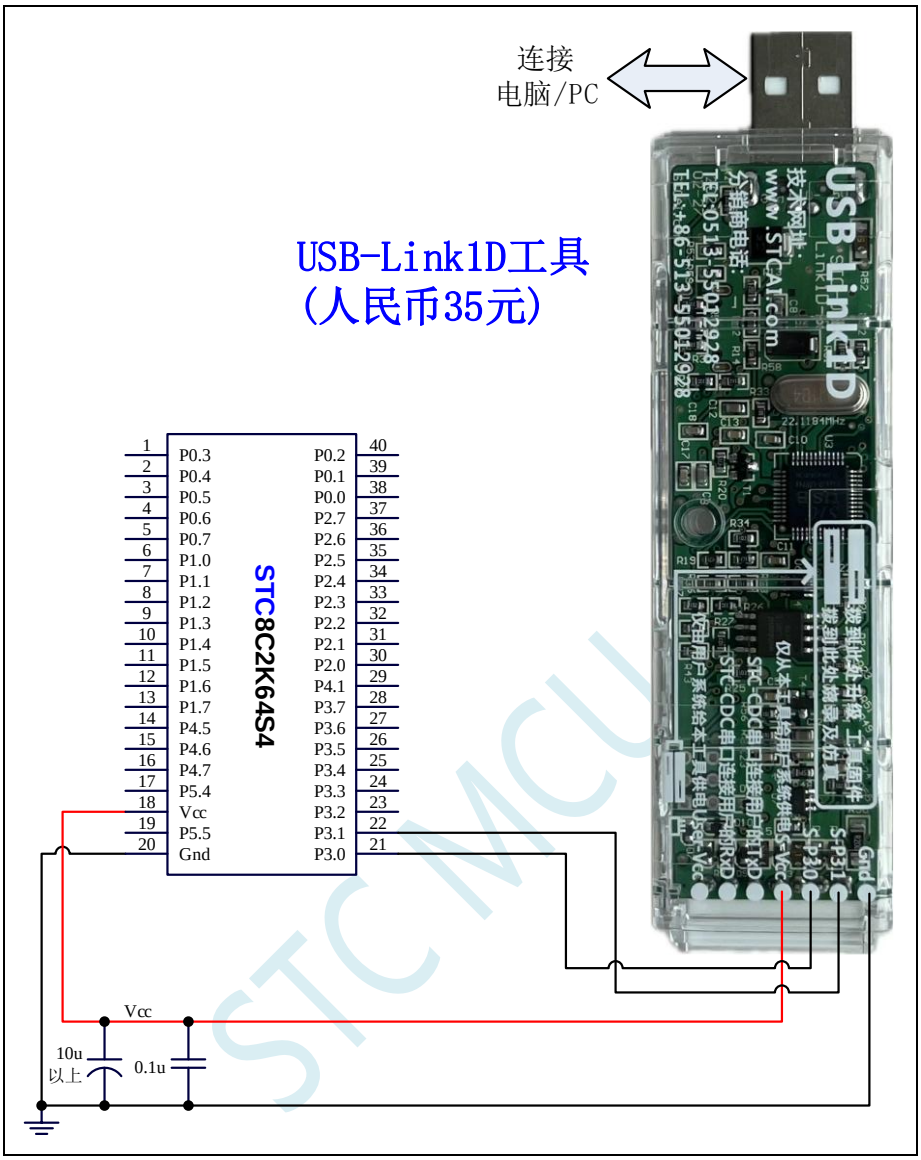
5.11.2 ISP 下载流程图（硬件/软件模拟 USB+串口模式）



注意：因 [P3.0, P3.1] 作下载/仿真用(下载/仿真接口仅可用 [P3.0, P3.1])，故建议用户将串口 1 放在 P3.6/P3.7 或 P1.6/P1.7 ，若用户不想切换，坚持使用 P3.0/P3.1 工作或作为串口 1 进行通信，则务必在下载程序时，在软件上勾选“下次冷启动时，P3.2/P3.3 为 0/0 时才可以下载程序”。

【注 1】：STC15，STC8 系列及以后新出的芯片的烧录保护引脚为 P3.2/P3.3，之前早期芯片的烧录保护引脚为 P1.0/P1.1。

5.11.3 使用 USB-Link1D 工具下载，支持在线和脱机下载



ISP 下载步骤：

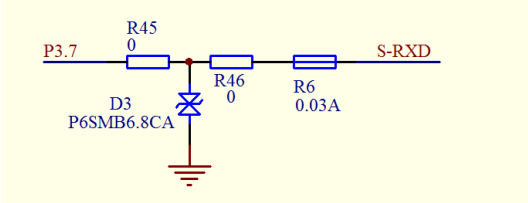
1、 按照如图所示的连接方式将 USB-Link1D 和目标芯片连接

2、 点击 AIapp-ISP 下载软件中的“下载/编程”按钮

3、 开始 ISP 下载

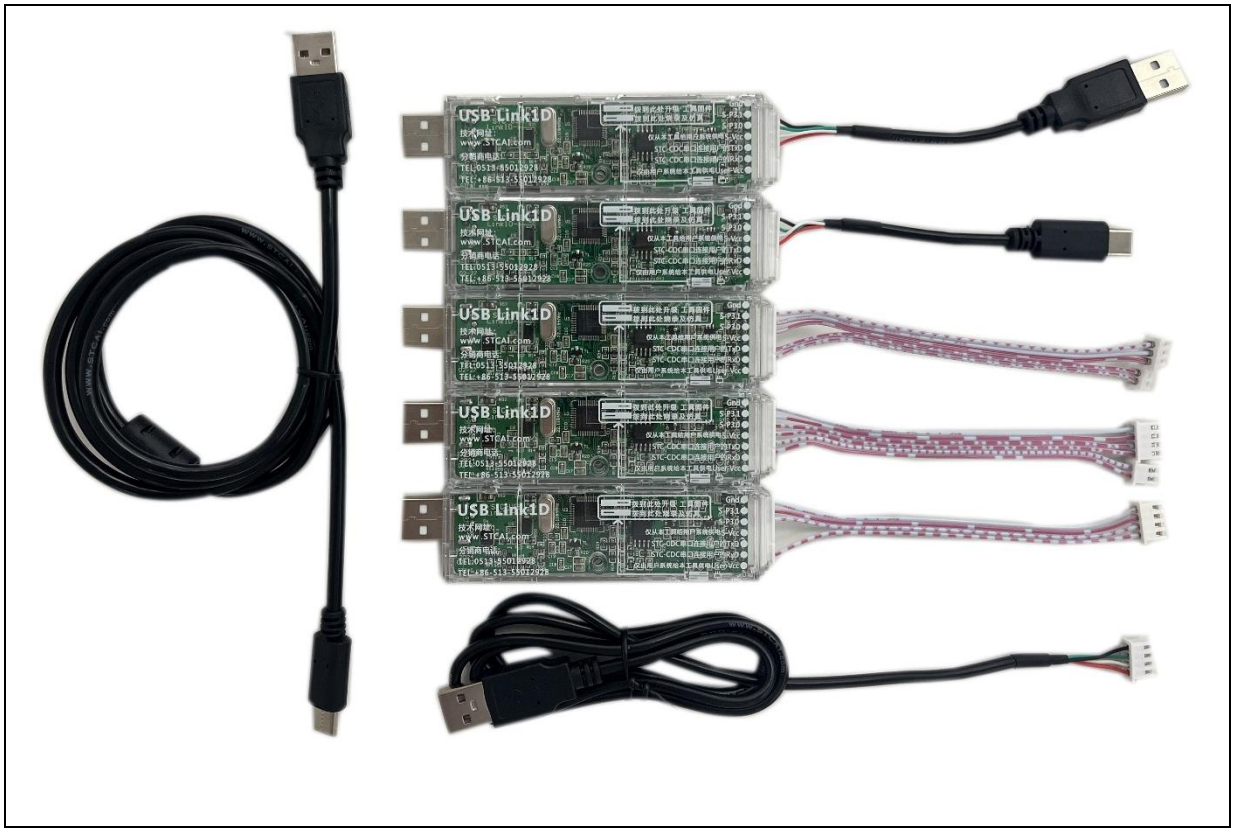
注意：若是使用 USB-Link1D 给目标系统供电，目标系统的总电流不能大于 200mA，否则会导致下载失败。

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





如果用户板上的串口接收脚 P3.0 口上有强上拉或者强下拉（比如处于接收状态的 RS485），此时使用 USB-Link1D 可能会无法下载，用户可将 USB-Link1D 工具上的 30mA 的保险丝 R6 用 0 欧姆电阻替换（实际测量 30mA 的保险丝的静态电阻值为 10～15 欧姆）



上面 **RMB35** 是配上面全部的线，是亏本补助大家的

USB-Link1D 工具接口说明

USB-Link1D 工具是 STC-USB Link1 的升级版、功能在 STC-USB Link1 的基础上增加了两个 STC-CDC 串口，可作为通用 USB 转串口使用。

工具 USB-Link1D 的使用注意事项请参考附录章节



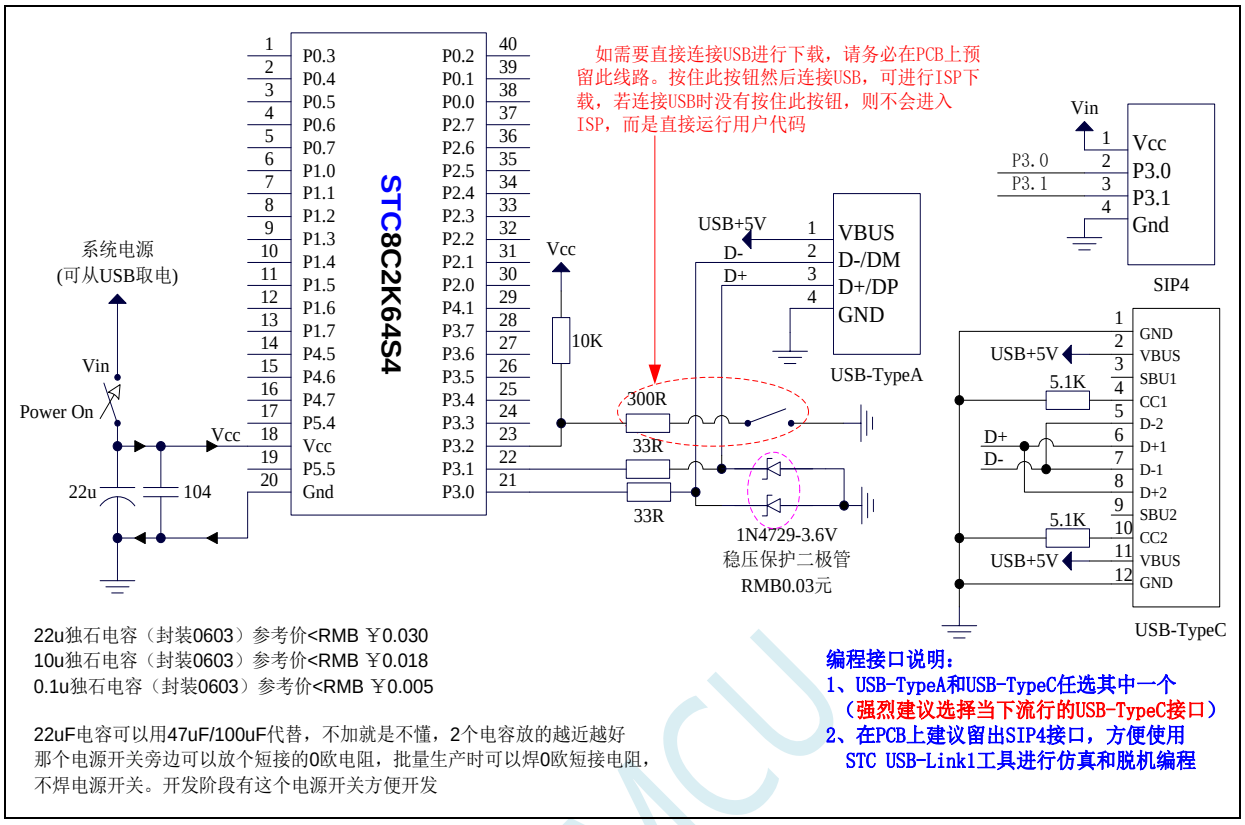
工具正面图



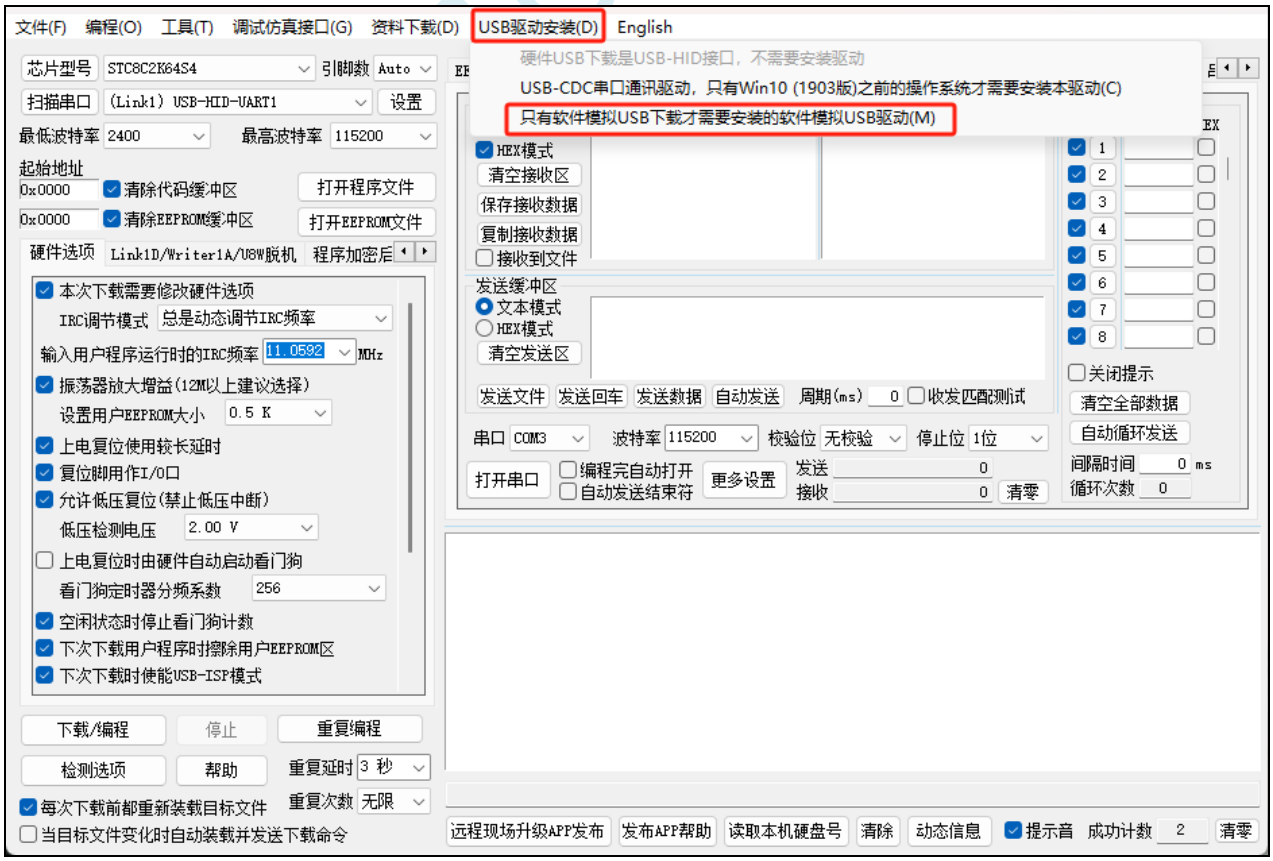
工具反面图

管脚编号	接口名称	接口功能
1	User-Vcc	仅由用户系统给本工具供电
2	S-RxD	第 2 组 STC-CDC 串口的发送脚，连接用户单片机串口的接收脚
3	S-TxD	第 2 组 STC-CDC 串口的接收脚，连接用户单片机串口的发送脚
4	S-Vcc	仅从本工具给用户系统供电
5	S-P3.0	使用 Link1D 进行 ISP 下载时的串口发送脚，连接目标单片机的 P3.0
		使用 Link1D 进行 SWD 硬件仿真时的数据脚，连接目标单片机的 SWDDAT
		第 1 组 STC-CDC 串口的发送脚，连接用户单片机串口的接收脚
6	S-P3.1	使用 Link1D 进行 ISP 下载时的串口接收脚，连接目标单片机的 P3.1
		使用 Link1D 进行 SWD 硬件仿真时的时钟脚，连接目标单片机的 SWDCLK
		第 1 组 STC-CDC 串口的接收脚，连接用户单片机串口的发送脚
7	Gnd	地线

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



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



ISP 下载步骤：

1、D-/P3.0, D+/P3.1 与 PC-USB 端口连接好

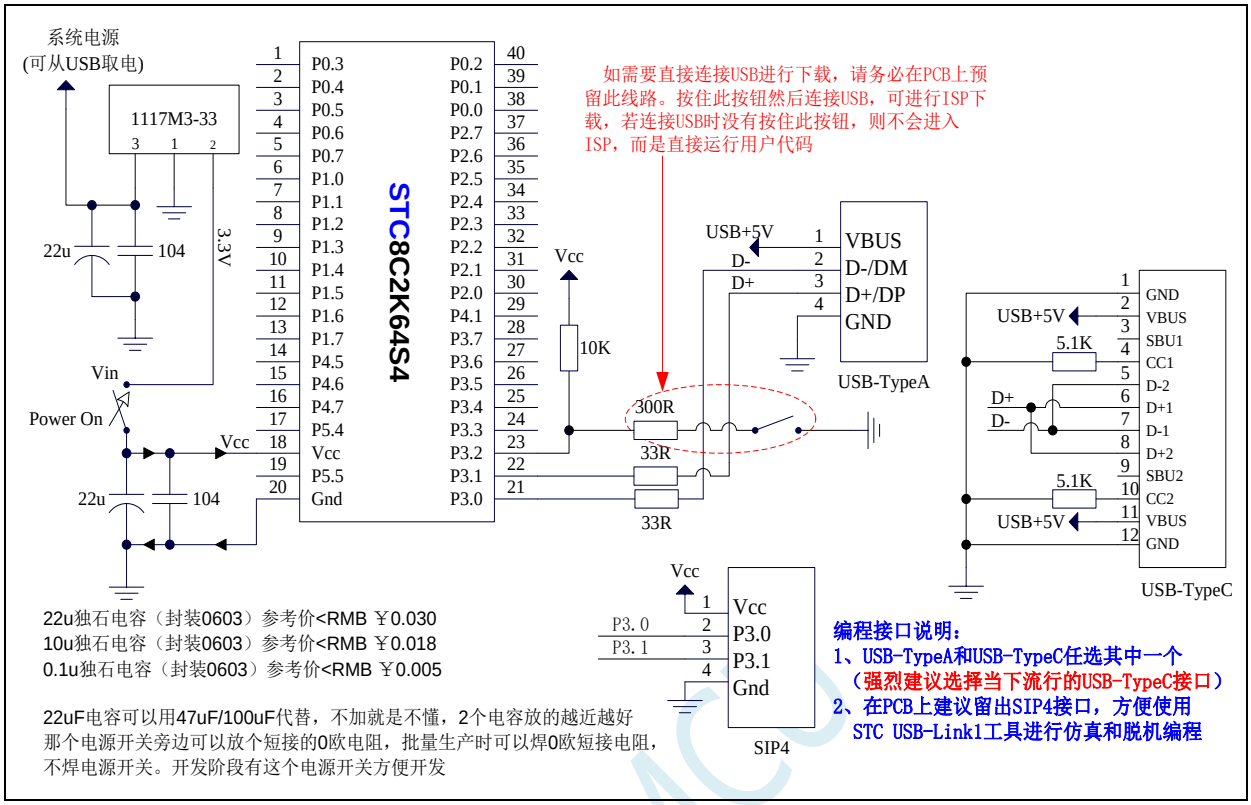
2、将 P3.2 与 GND 短接，实验箱板子上的 P3.2/INT0 按键按下

3、给目标芯片重新上电。若目标芯片已经停电，直接上电即可；若目标芯片处于通电状态，则需给目标芯片停电再上电（冷启动）。等待 AIapp-ISP 下载软件中自动识别出“（HID1）USB-Writer”识别出来后,就与 P3.2 状态无关了。

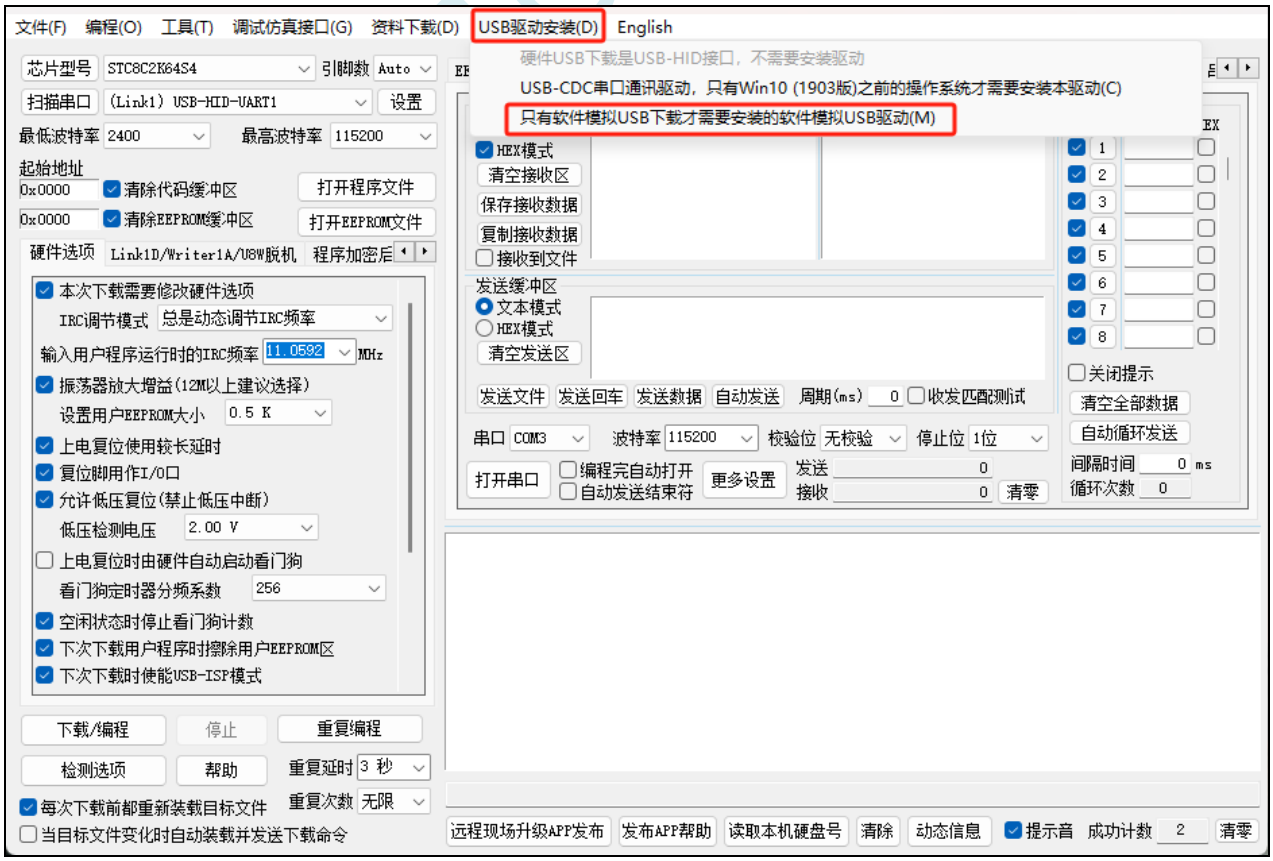
4、点击下载软件中的“下载/编程”按钮（注意：USB 下载与串口下载的操作顺序不同，**千万千万不要先点下载按钮，一定到等到电脑端识别出“（HID1）USB-Writer”设备后，才能点下载按钮开始下载**）

STC MCU

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



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



ISP 下载步骤：

1、D-/P3.0, D+/P3.1 与 PC-USB 端口连接好

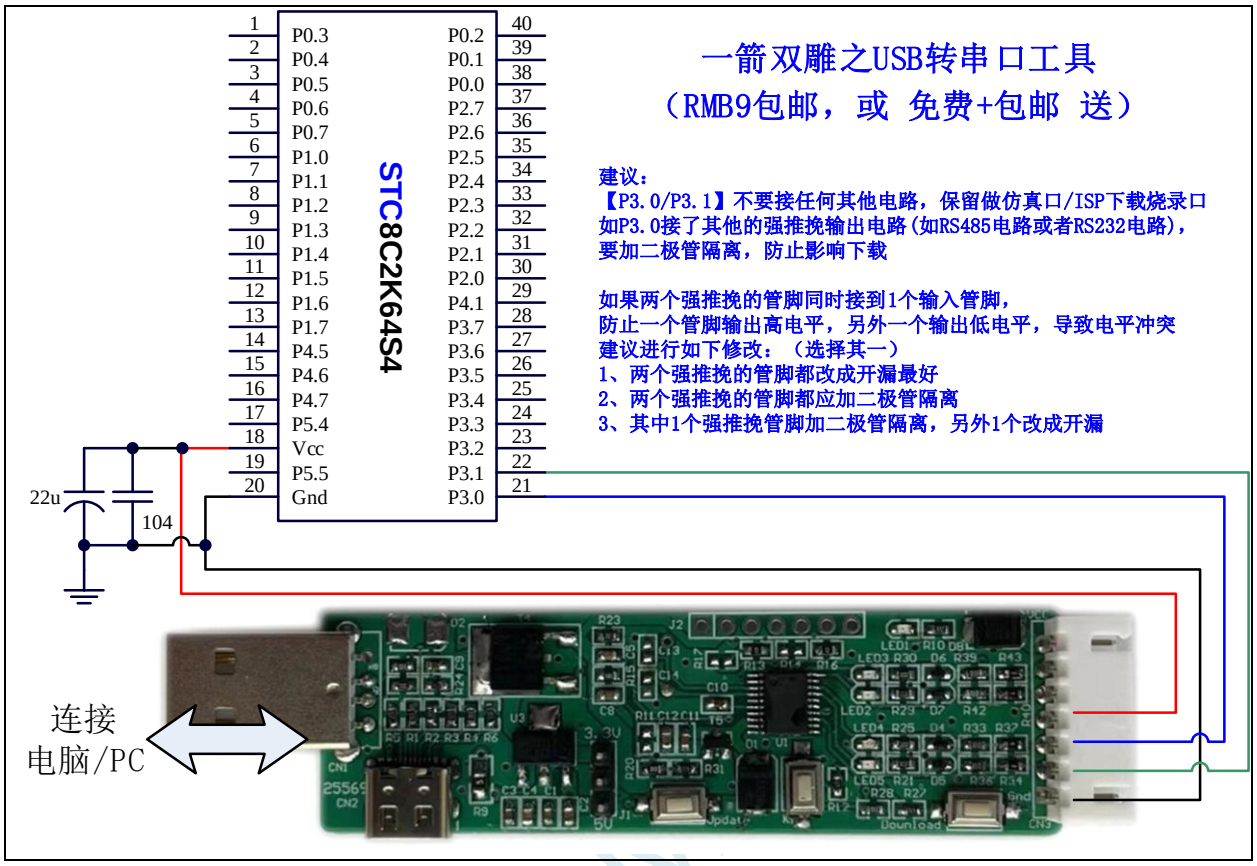
2、将 P3.2 与 GND 短接，实验箱板子上的 P3.2/INT0 按键按下

3、给目标芯片重新上电。若目标芯片已经停电，直接上电即可；若目标芯片处于通电状态，则需给目标芯片停电再上电（冷启动）。等待 AIapp-ISP 下载软件中自动识别出“（HID1）USB-Writer”识别出来后,就与 P3.2 状态无关了。

4、点击下载软件中的“下载/编程”按钮（注意：USB 下载与串口下载的操作顺序不同，**千万千万不要先点下载按钮，一定到等到电脑端识别出“（HID1）USB-Writer”设备后，才能点下载按钮开始下载**）

STC MCU

5.11.6 使用一箭双雕之 USB 转串口工具下载



ISP 下载步骤：

1、 按照如图所示的连接方式将 USB 转串口工具和目标芯片连接

2、 点击 AIapp-ISP 下载软件中的“下载/编程”按钮

3、 开始 ISP 下载

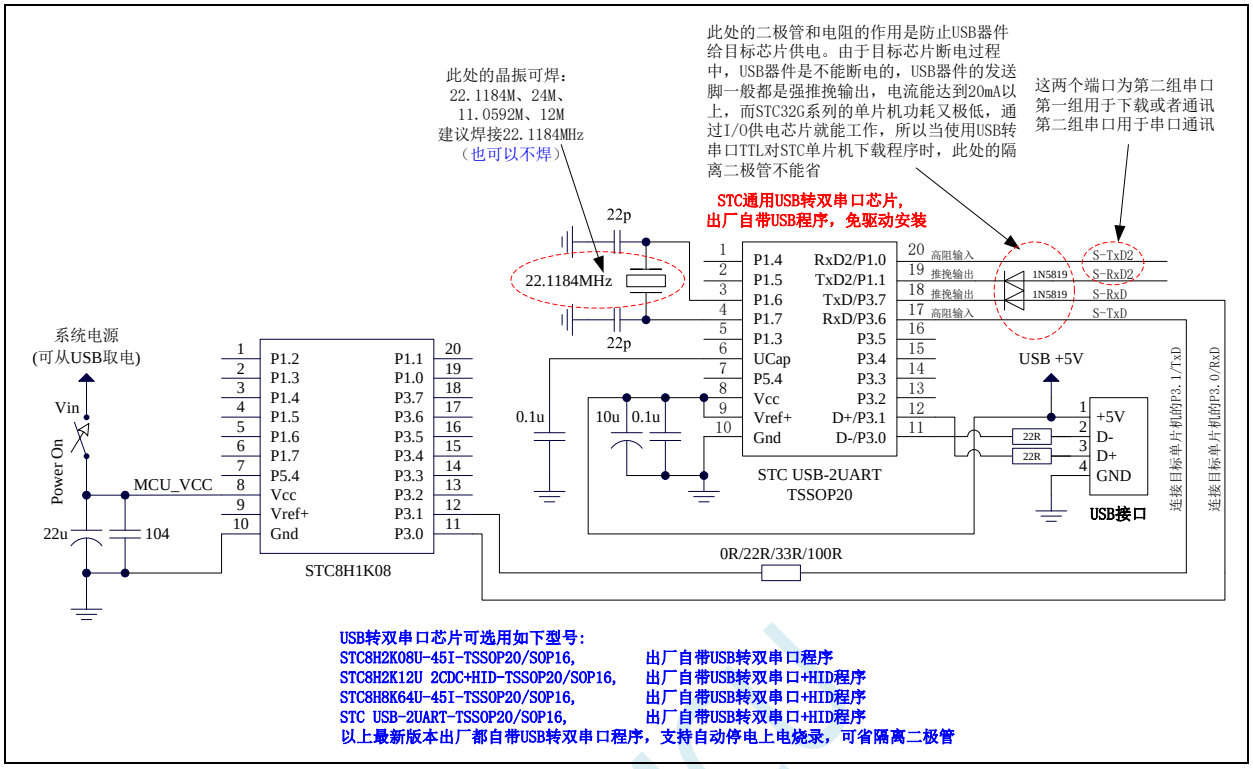
注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。





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

5.11.7　使用 USB 转双串口/TTL 下载（有外部晶振）



ISP 下载步骤：

1、 给目标芯片停电，注意不能给“STC USB-2UART”芯片停电

2、 由于“STC USB-2UART”芯片的发送脚是强推挽输出，必须在目标芯片的 P3.0 口和“STC USB-2UART”的发送脚之间串接一个二极管，否则目标芯片无法完全断电，达不到给目标芯片停电的目标。

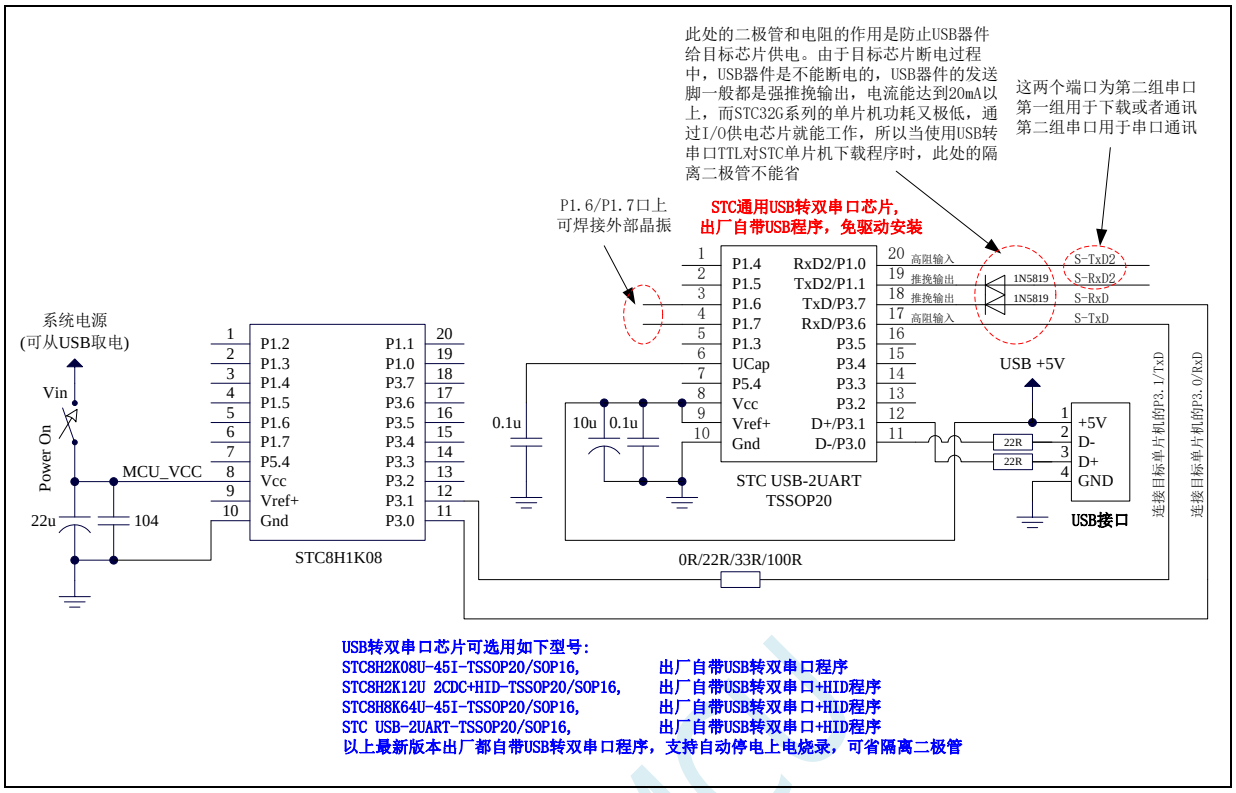
3、 点击 AIapp-ISP 下载软件中的“下载/编程”按钮

4、 给目标芯片上电

5、 开始 ISP 下载

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

5.11.8　使用 USB 转双串口/TTL 下载（无外部晶振）



ISP 下载步骤：

1、给目标芯片停电，注意不能给“STC USB-2UART”芯片停电

2、由于“STC USB-2UART”芯片的发送脚是强推挽输出，必须在目标芯片的 P3.0 口和“STC USB-2UART”的发送脚之间串接一个二极管，否则目标芯片无法完全断电，达不到给目标芯片停电的目标。

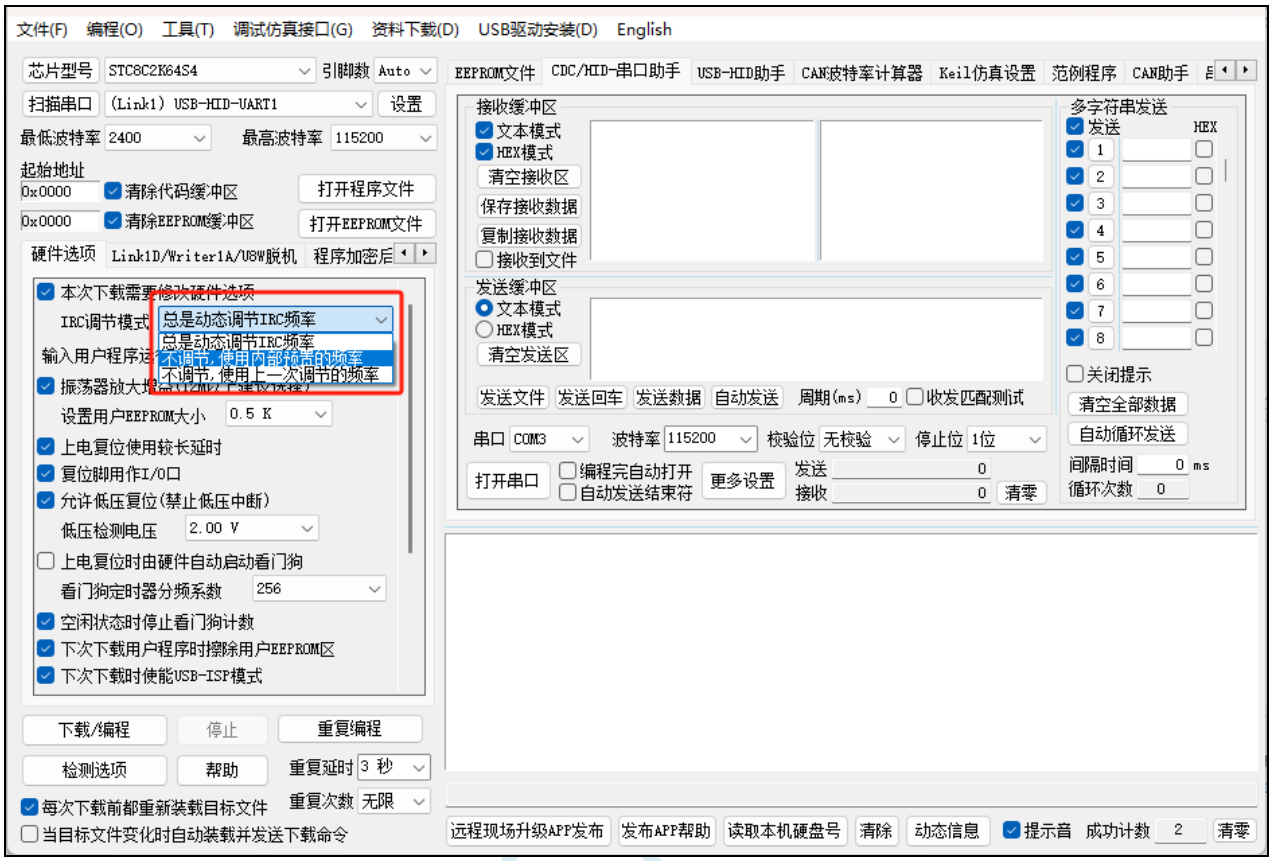
3、点击 AIapp-ISP 下载软件中的“下载/编程”按钮

4、给目标芯片上电

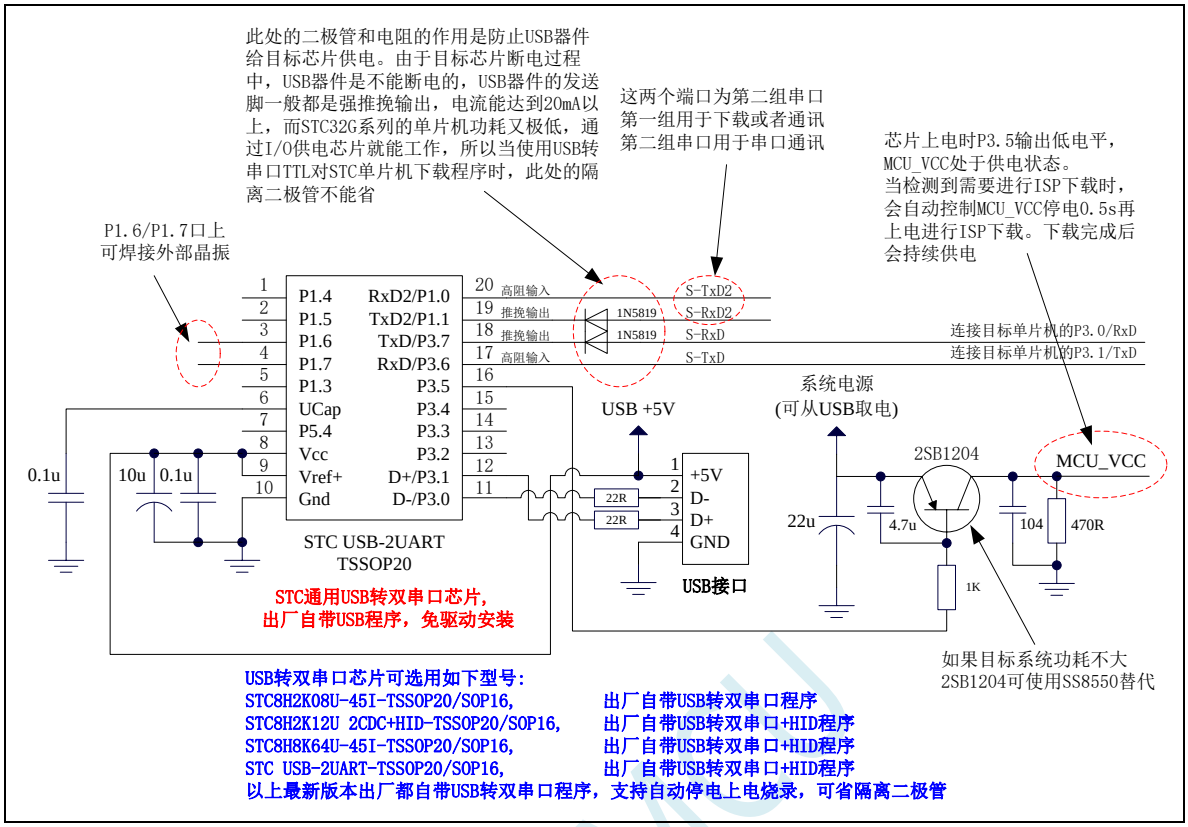
5、开始 ISP 下载

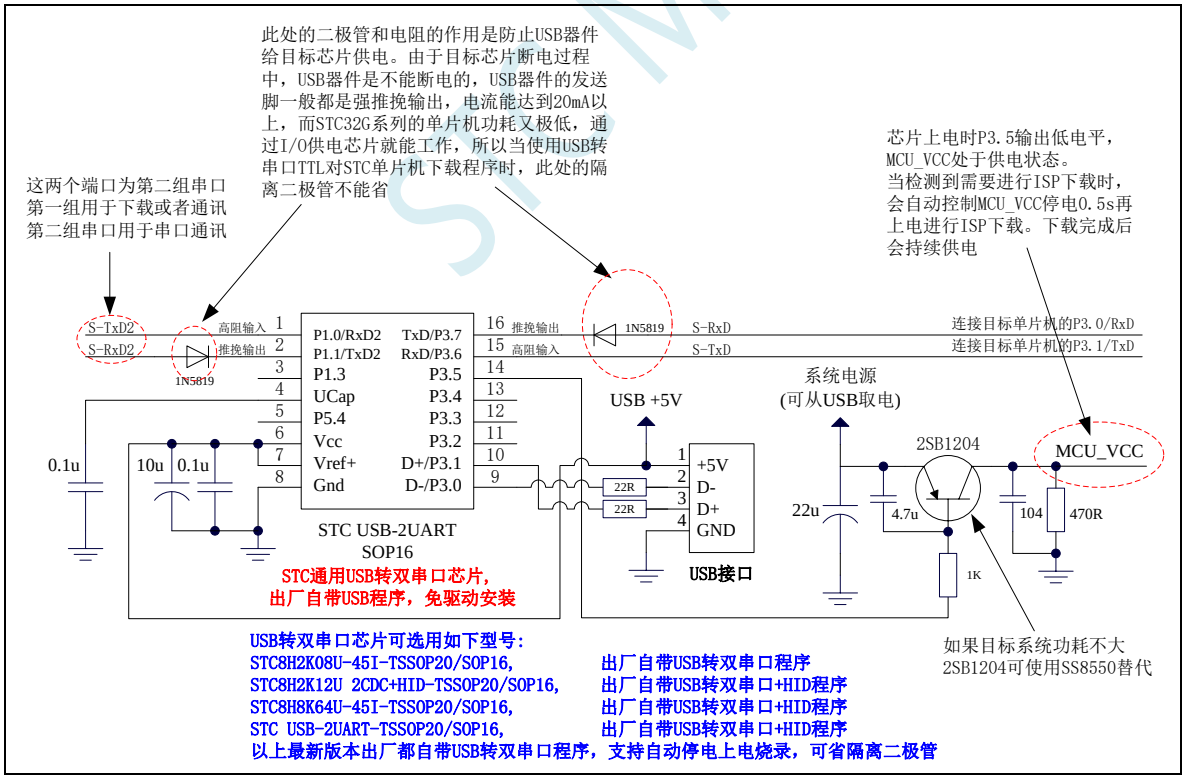
注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

注意：如果使用无外部晶振的 USB 转双串口/TTL 下载时，强烈建议 ISP 下载选项“选择 IRC 调节模式”选择“不调节，使用内部预置的频率”选项，这样可以避免调节频率时将无外部晶振的 USB 转双串口/TTL 工具本身的频率误差代入目标芯片。如下图：

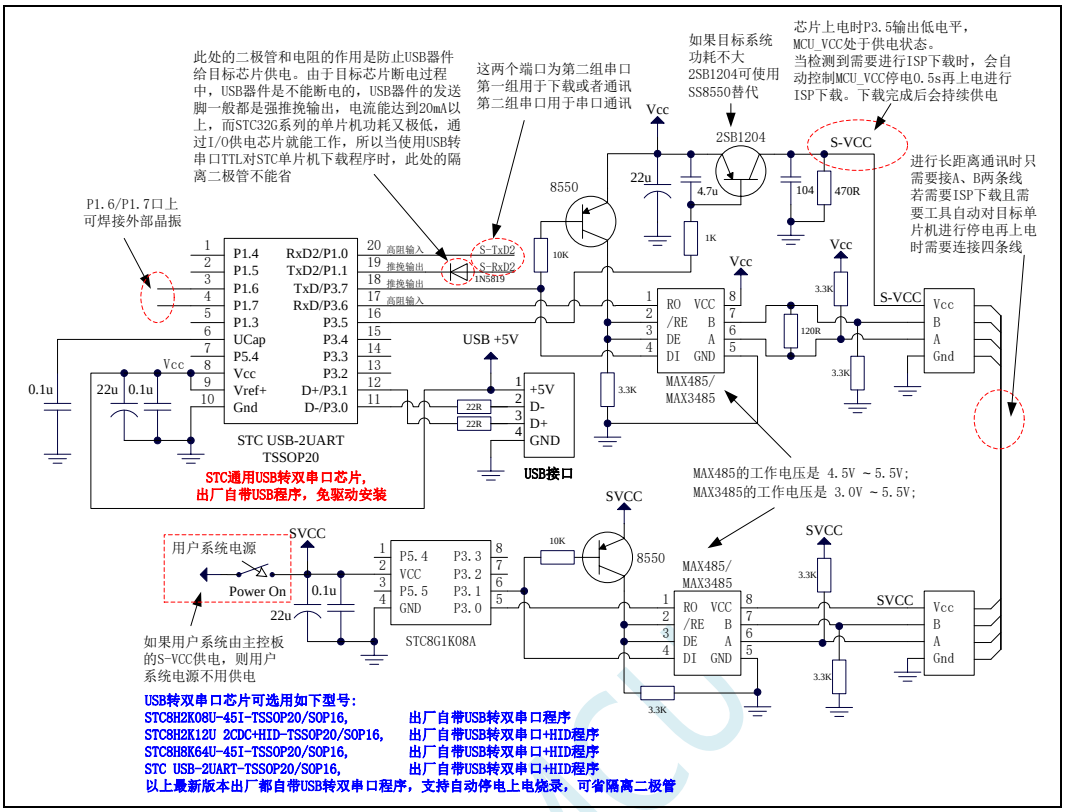


5.11.9　使用 USB 转双串口/TTL 下载（自动停电/上电）

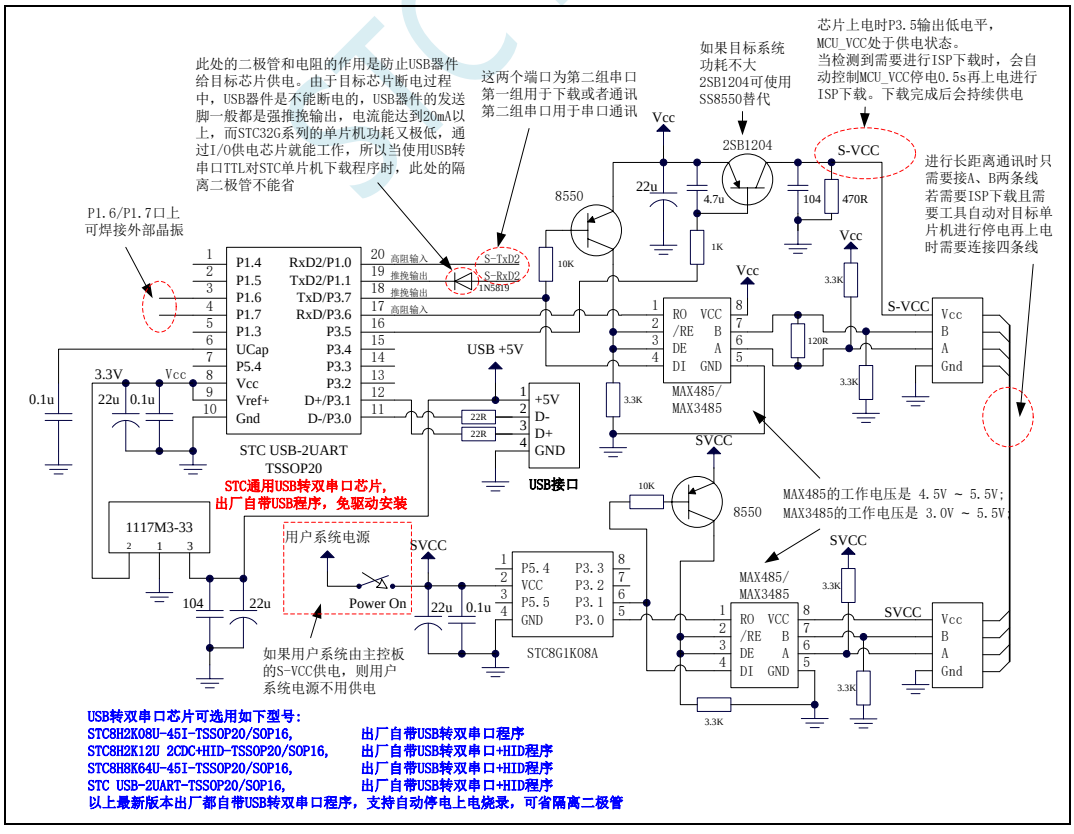




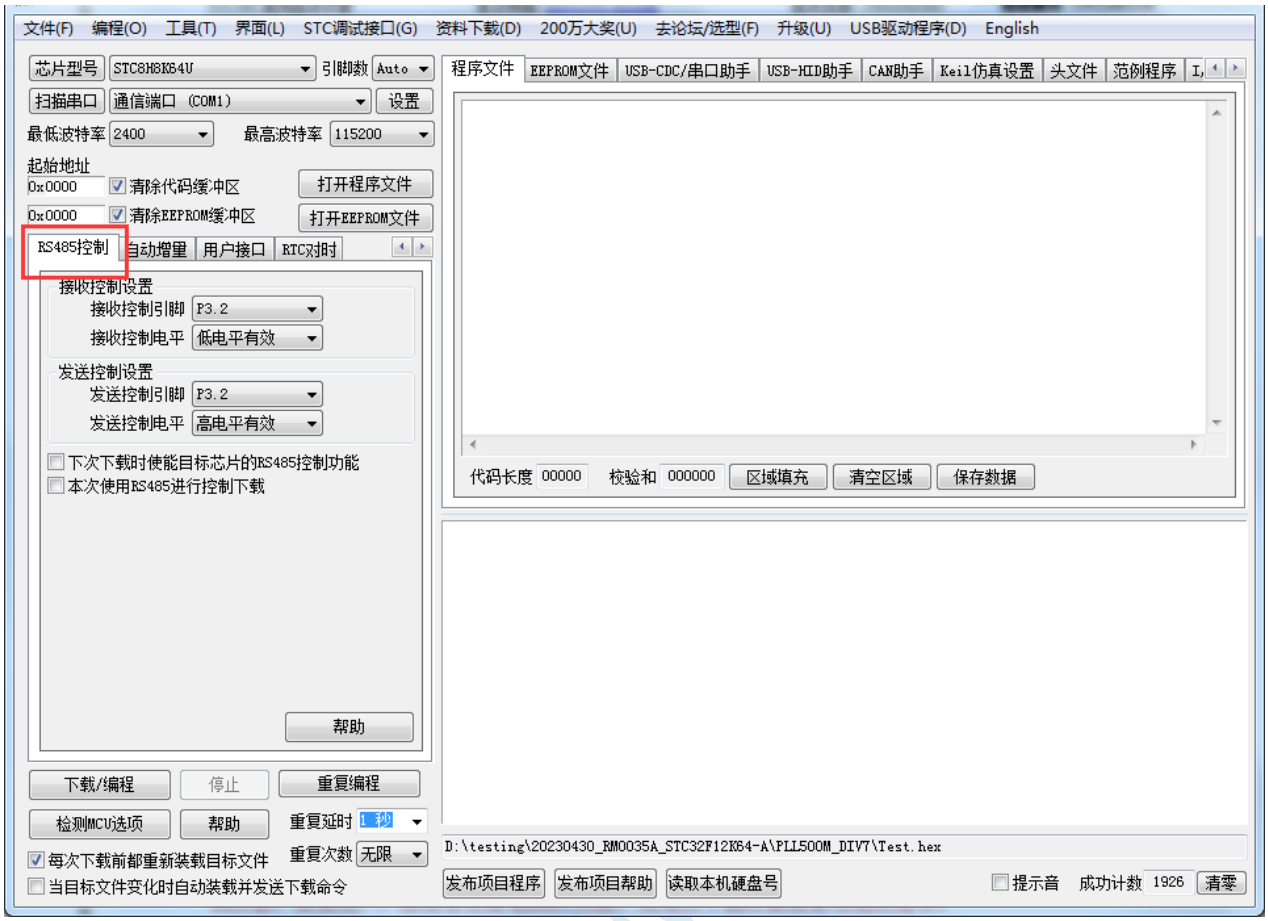
5.11.10　使用 USB 转双串口/RS485 下载（5.0V）



5.11.11　使用 USB 转双串口/RS485 下载（3.3V）



AIapp-ISP 下载软件中 RS485 相关设置界面如下图：



设置项详细说明如下

“接收控制设置”：设置控制 RS485 接收脚的 I/O 口以及控制有效电平

“发送控制设置”：设置控制 RS485 发送脚的 I/O 口以及控制有效电平

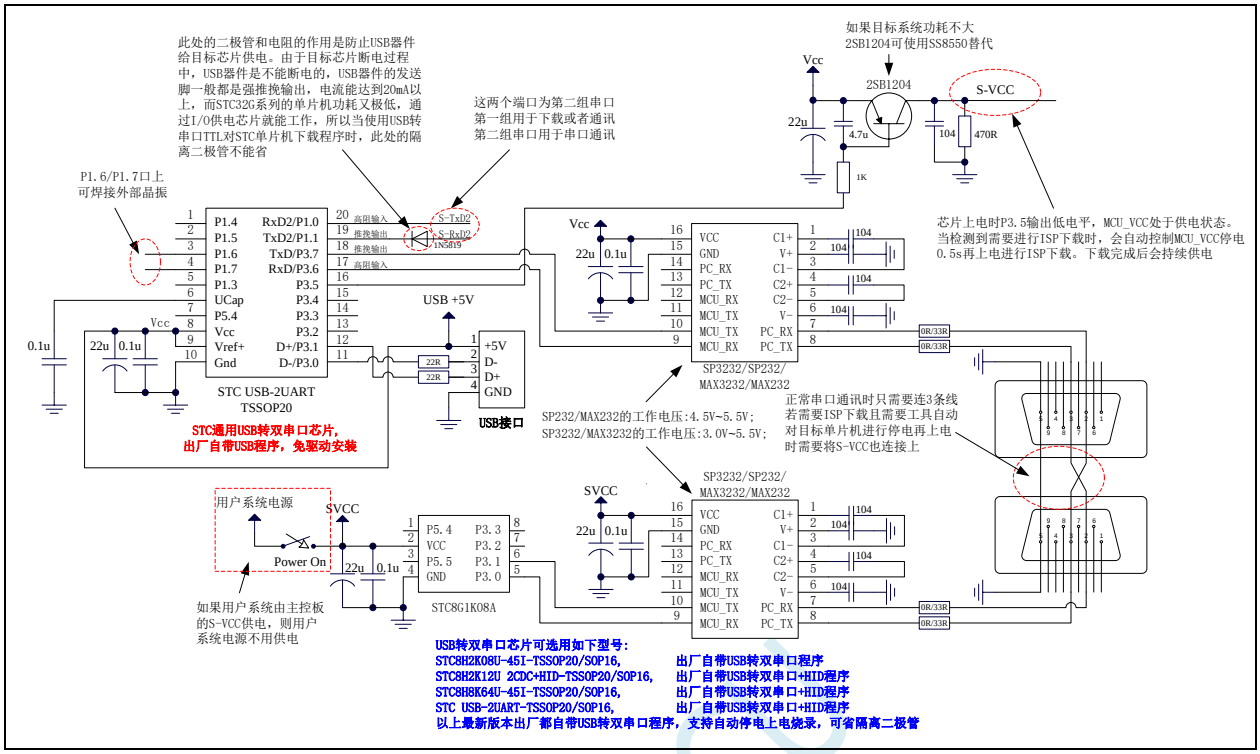
“下次下载时使能目标芯片的 RS485 控制功能”：设置目标单片机下次 ISP 下载时使能 RS485 控制（特别注意：如果用户产品需要使能 RS485 功能，则每次下载时都需要勾选此选项）

“本次使用 RS485 进行控制下载”：本次 AIapp-ISP 下载软件使用 RS485 模式对目标单片机进行下载。

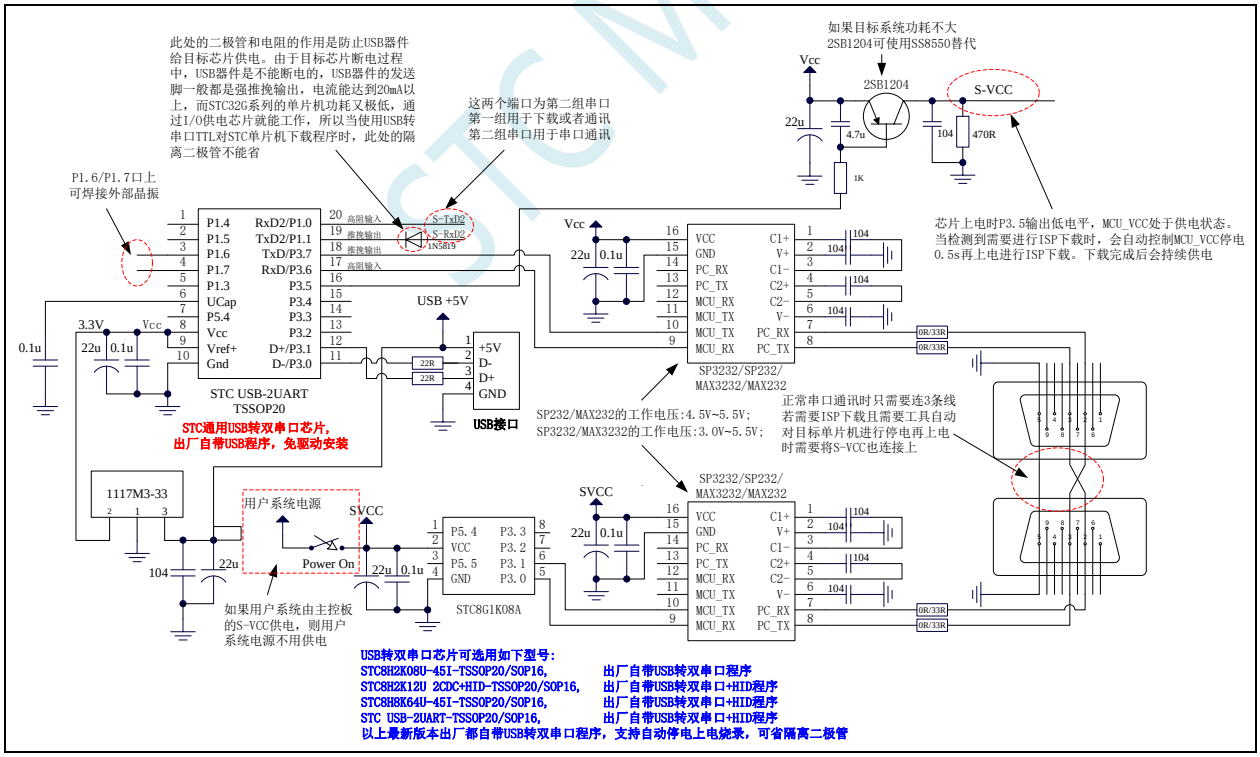
7.3.x 固件版本的单片机，需要固件版本等于或大于 7.3.12 才能很好的支持 RS485

7.4.x 固件版本的单片机都可很好的支持 RS485

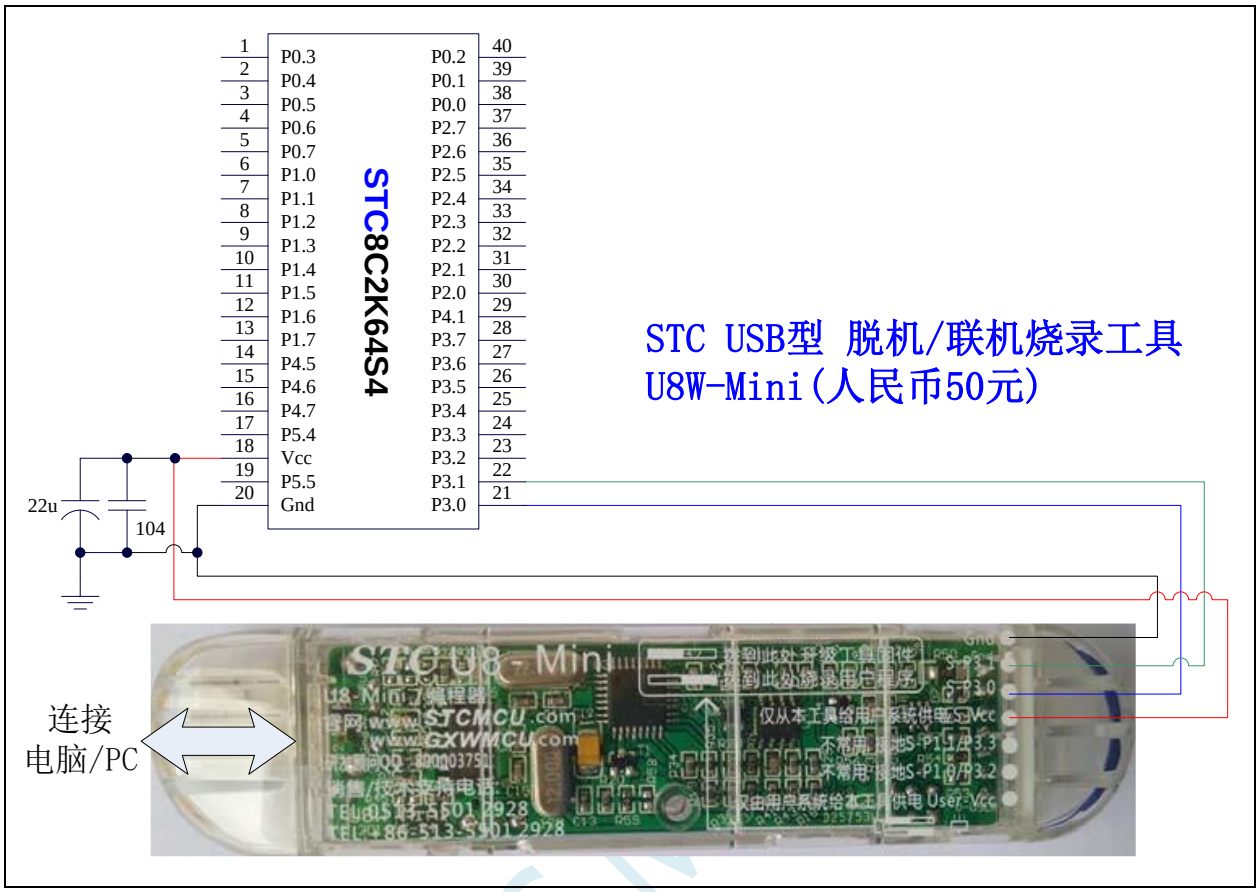
5.11.12　使用 USB 转双串口/RS232 下载（5.0V）



5.11.13　使用 USB 转双串口/RS232 下载（3.3V）



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



ISP 下载步骤：

1、 按照如图所示的连接方式将 U8-Mini 和目标芯片连接

2、 点击 AIapp-ISP 下载软件中的“下载/编程”按钮

3、 开始 ISP 下载

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

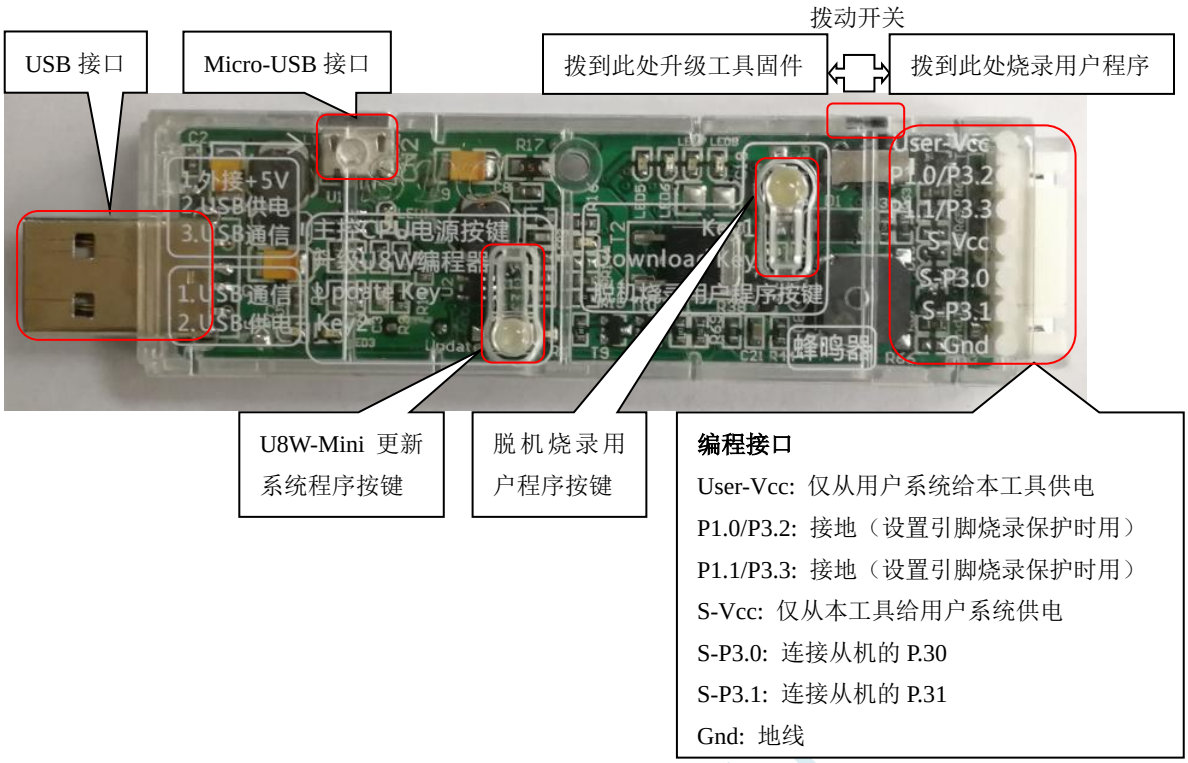
注意：目前有发现使用 USB 线供电进行 ISP 下载时，由于 USB 线太细，在 USB 线上的压降过大，导致 ISP 下载时供电不足，所以请在使用 USB 线供电进行 ISP 下载时，务必使用 USB 加强线。

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

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

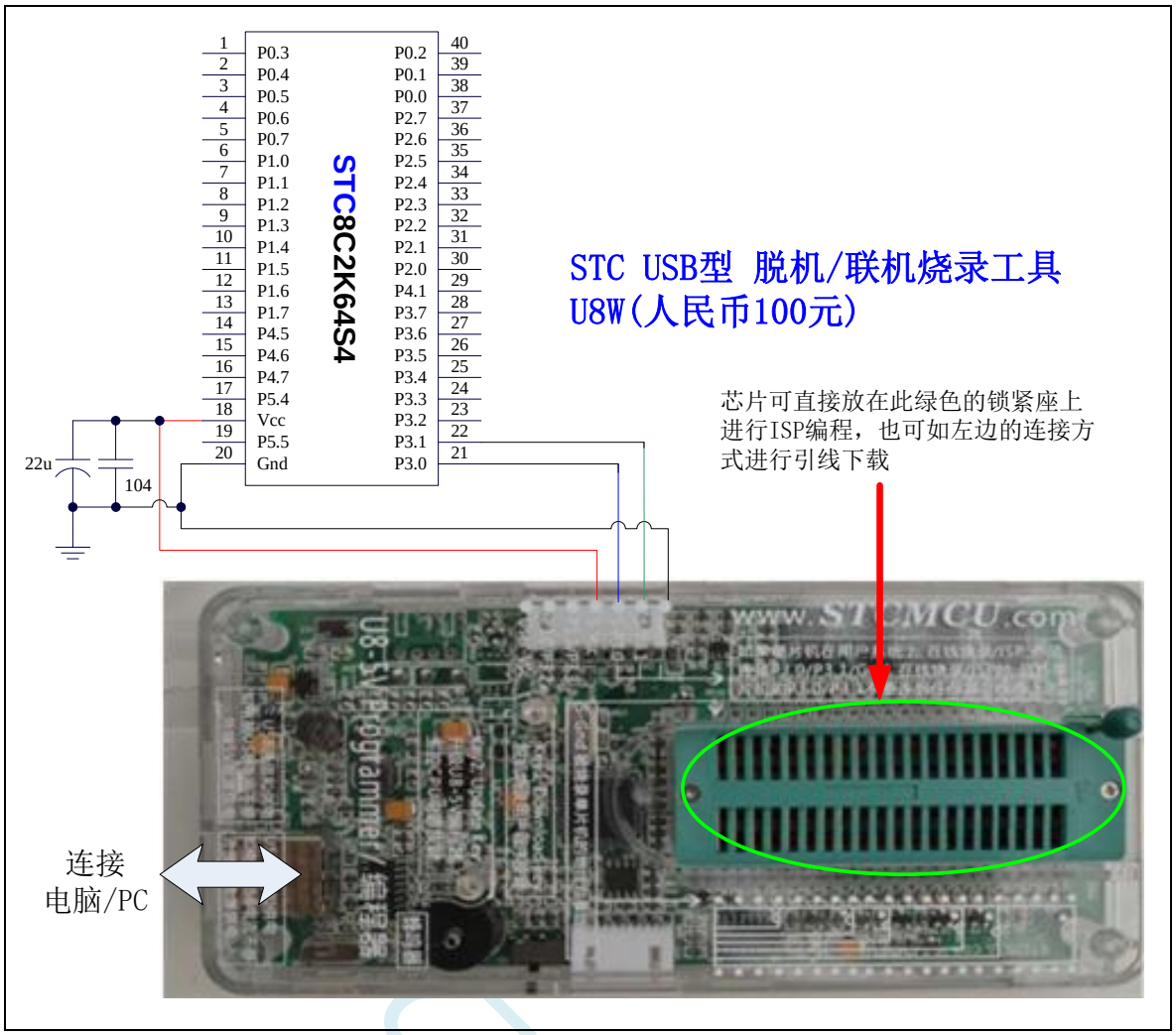
2、 U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后放开 Key2（电源）按键 后，再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（**按下 Key1 ➔ 按下 Key2 ➔ 松开 Key2 ➔ 松开 Key1**）

3、 进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电源）按键 即可


拨动开关
USB 接口
Micro-USB 接口
拨到此处升级工具固件
拨到此处烧录用户程序
User-Vcc
P1.0/P3.2
P1.1/P3.3
S-Vcc
S-P3.0
S-P3.1
Gnd
U8W-Mini 更新系统程序按键
脱机烧录用户程序按键
编程接口
User-Vcc: 仅从用户系统给本工具供电
P1.0/P3.2: 接地（设置引脚烧录保护时用）
P1.1/P3.3: 接地（设置引脚烧录保护时用）
S-Vcc: 仅从本工具给用户系统供电
S-P3.0: 连接从机的 P.30
S-P3.1: 连接从机的 P.31
Gnd: 地线

STC MCU

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



ISP 下载步骤（连线方式）：

1、 按照如图所示的连接方式将 U8W 和目标芯片连接

2、 点击 AIapp-ISP 下载软件中的“下载/编程”按钮

3、 开始 ISP 下载

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

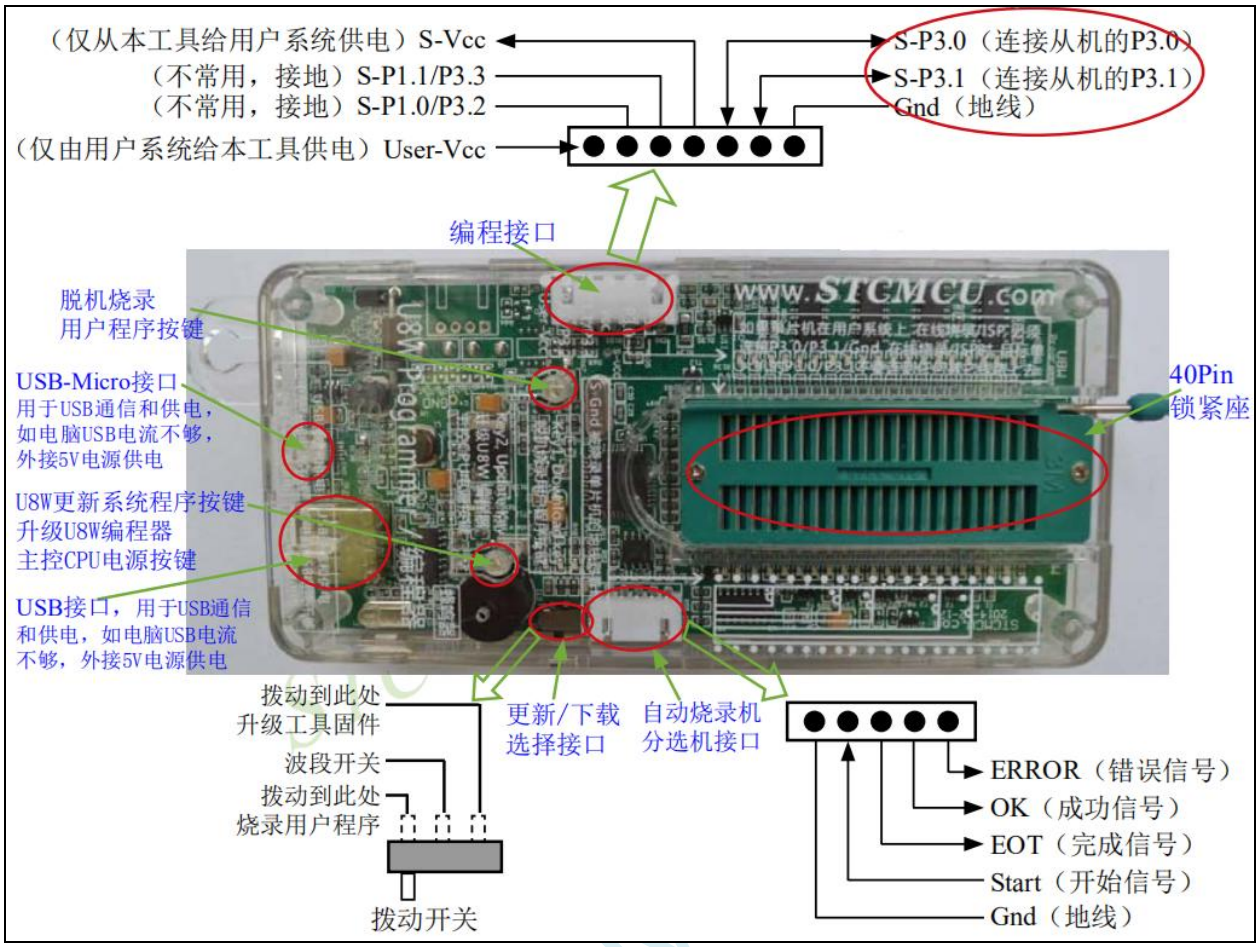
ISP 下载步骤（在板方式）：

1、 将芯片按照 1 脚靠近锁紧扳手、管脚向下靠齐的方向放置好目标芯片

2、 点击 AIapp-ISP 下载软件中的“下载/编程”按钮

3、 开始 ISP 下载

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



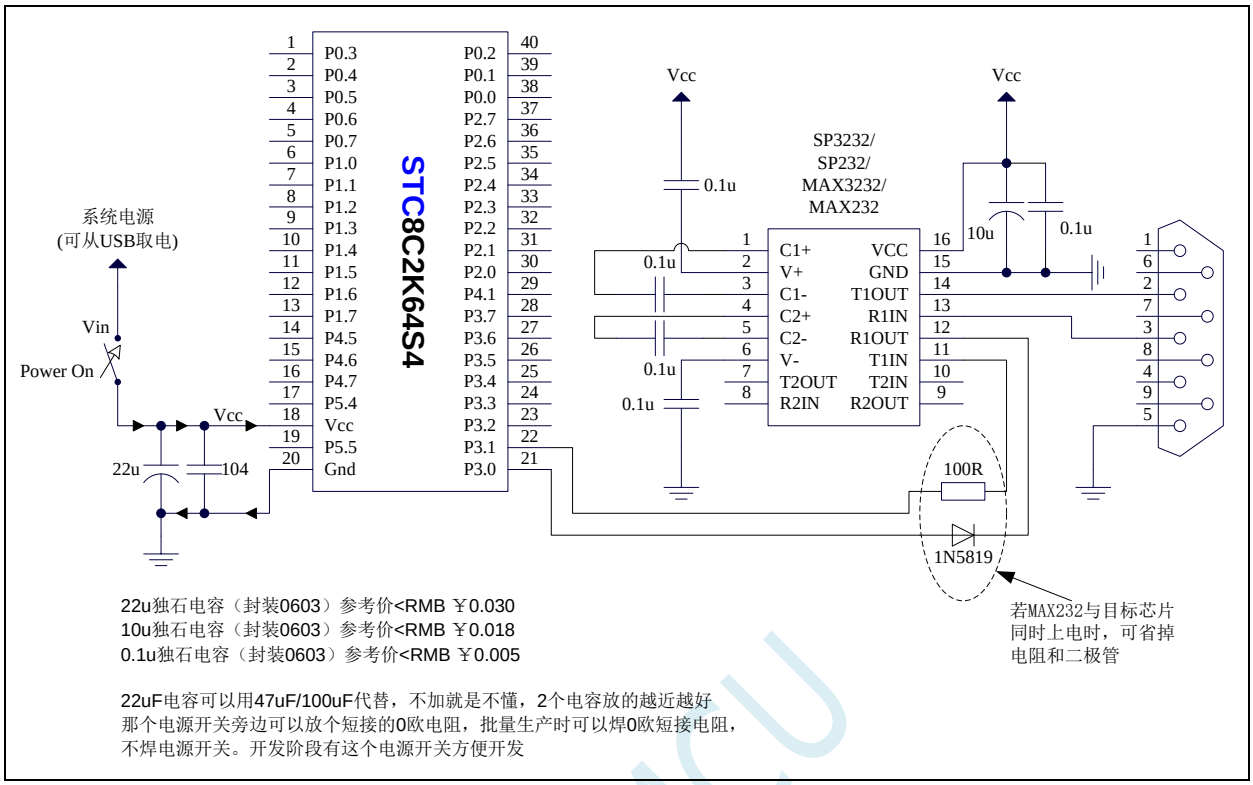
若要使用 U8W 进行仿真，首先必须将 U8W 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下：

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

2、 U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后放开 Key2（电源）按键 后，再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（**按下 Key1 ➔ 按下 Key2 ➔ 松开 Key2 ➔ 松开 Key1**）

3、 进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电源）按键 即可

5.11.16 使用 RS-232 转换器下载，也可支持仿真



ISP 下载步骤：

1、 给目标芯片停电

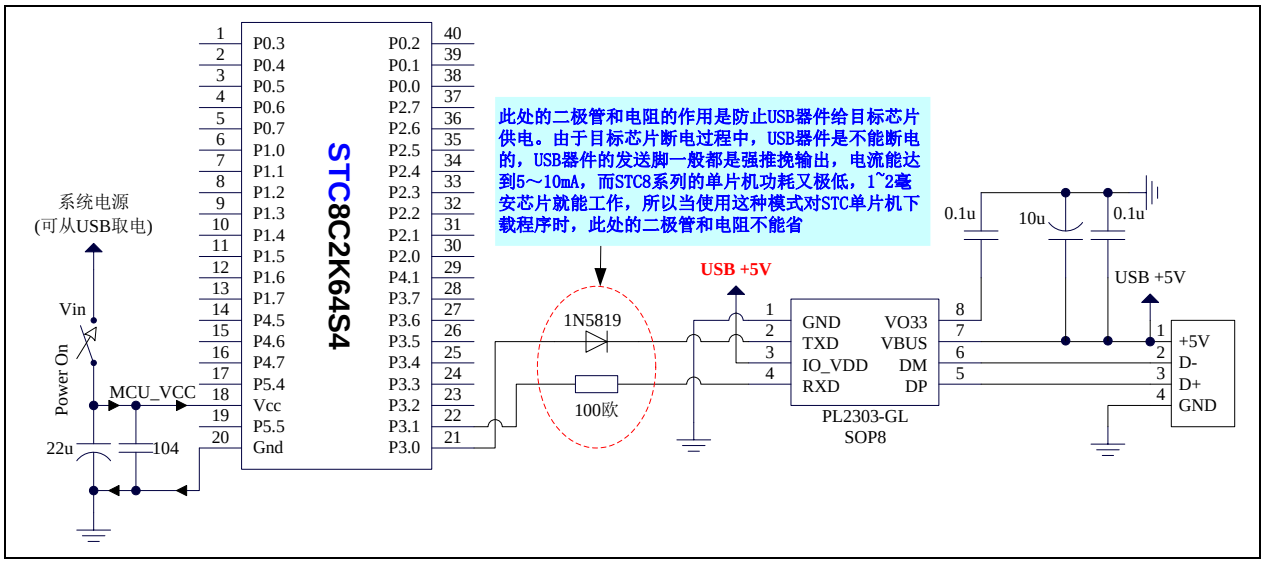
2、 点击 AIapp-ISP 下载软件中的“下载/编程”按钮

3、 给目标芯片上电

4、 开始 ISP 下载

STC MCU

5.11.17 使用 PL2303-GL 下载，也可支持仿真



ISP 下载步骤：

1、 给目标芯片停电，注意不能给 USB 转串口芯片停电（如：CH340、PL2303-GL 等）
　　注意：PL2303-SA 的部分波特率误差非常大，建议使用 PL2303-GL

2、 由于 USB 转串口芯片的发送脚一般都是强推挽输出，必须在目标芯片的 P3.0 口和 USB 转串口芯片的发送脚之间串接一个二极管，否则目标芯片无法完全断电，达不到给目标芯片停电的目标。

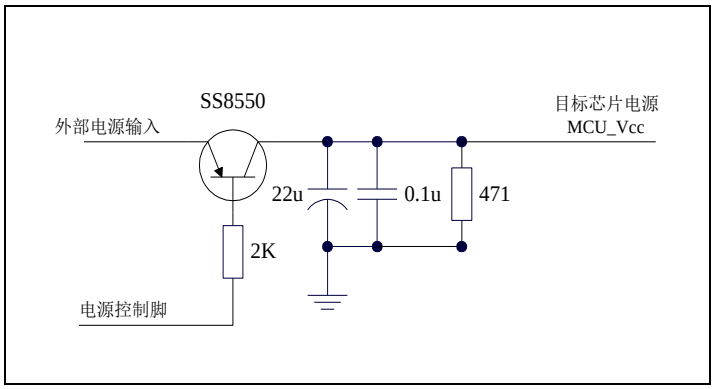
3、 点击 AIapp-ISP 下载软件中的“下载/编程”按钮

4、 给目标芯片上电

5、 开始 ISP 下载

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

5.11.18 单片机电源控制参考电路

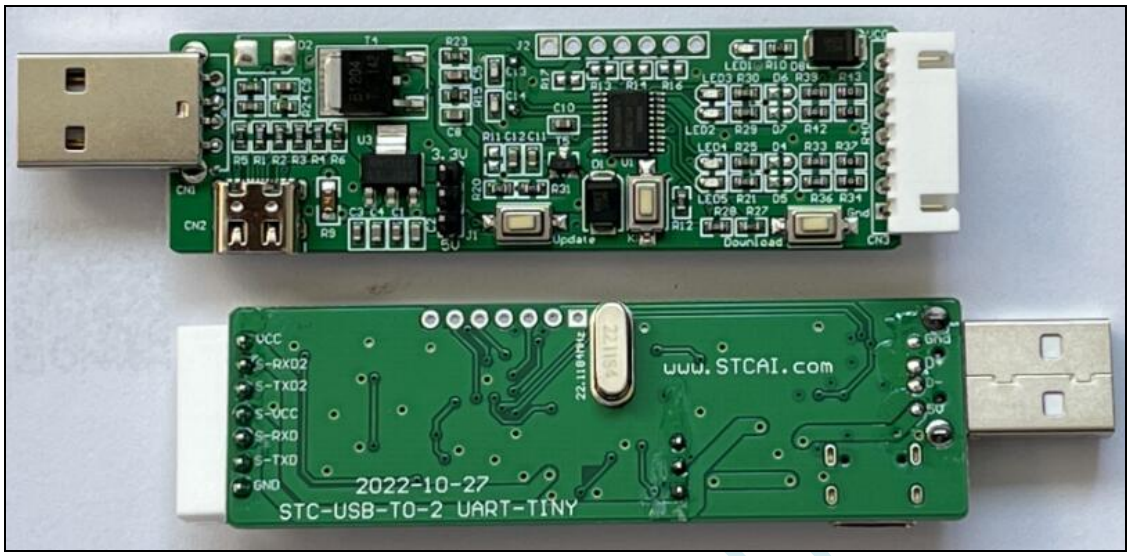


STC MCU

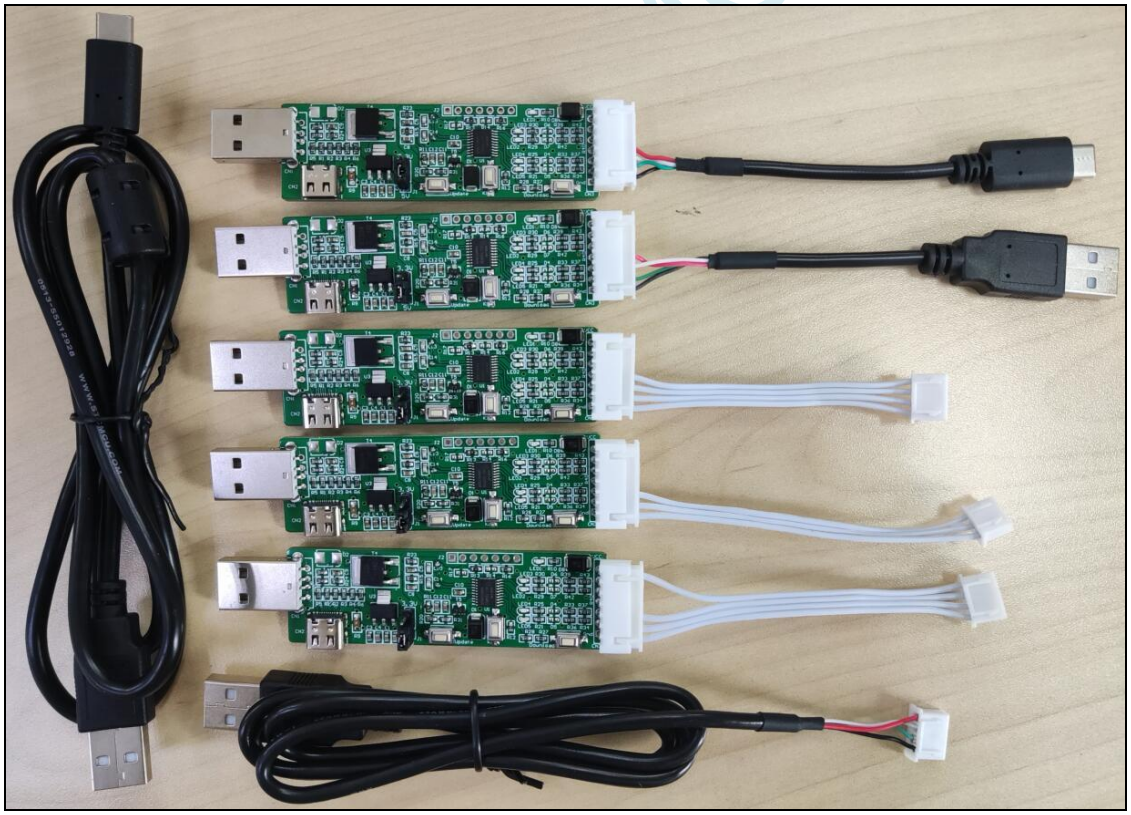
5.12 用 STC 一箭双雕之 USB 转双串口仿真 STC8 系列 MCU

先简单介绍下一箭双雕之 USB 转双串口工具

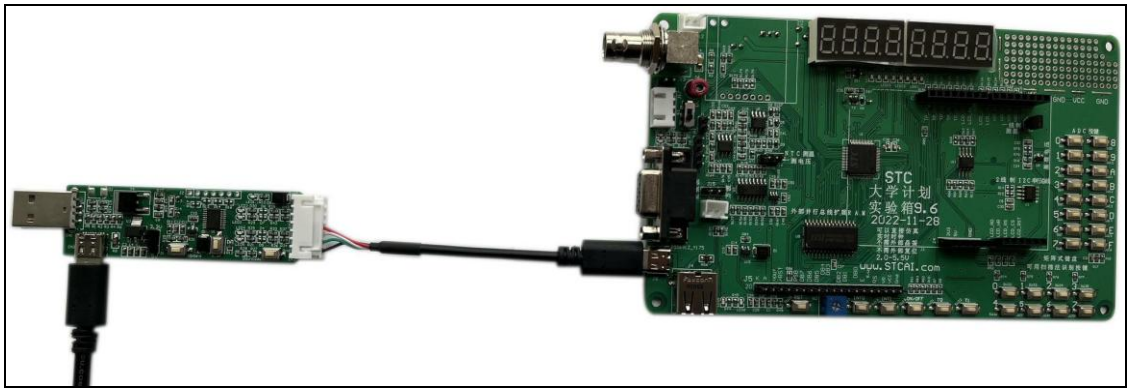
1、一箭双雕之 USB 转双串口工具外观图：



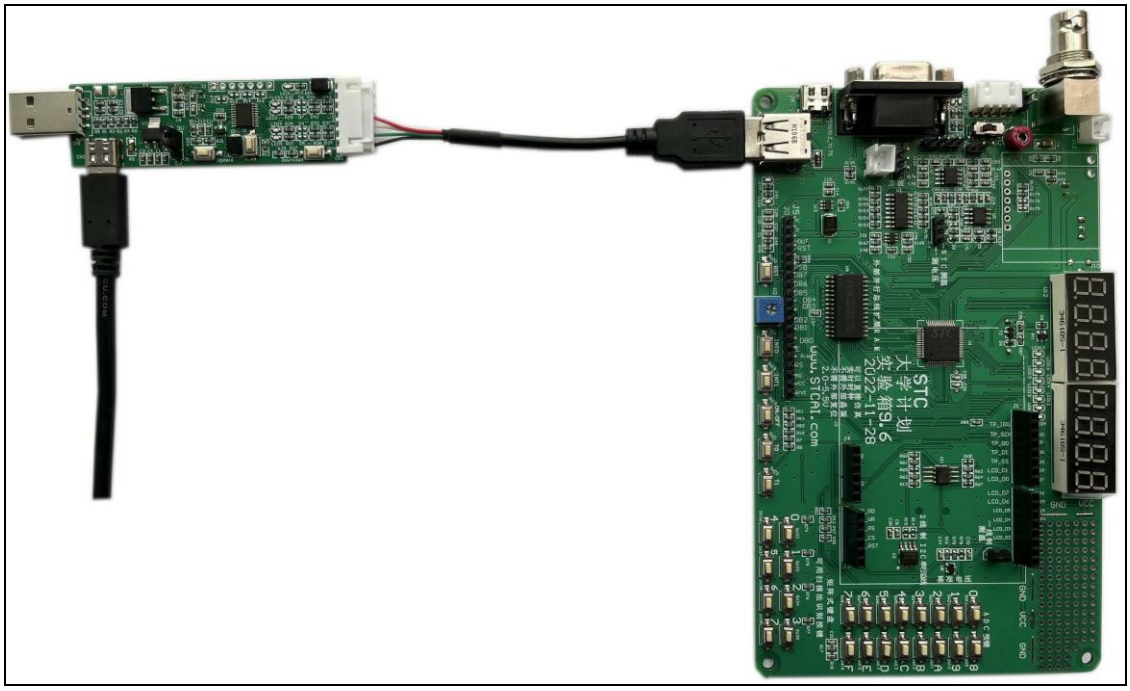
2、一箭双雕之 USB 转双串口工具几种常用连接线：



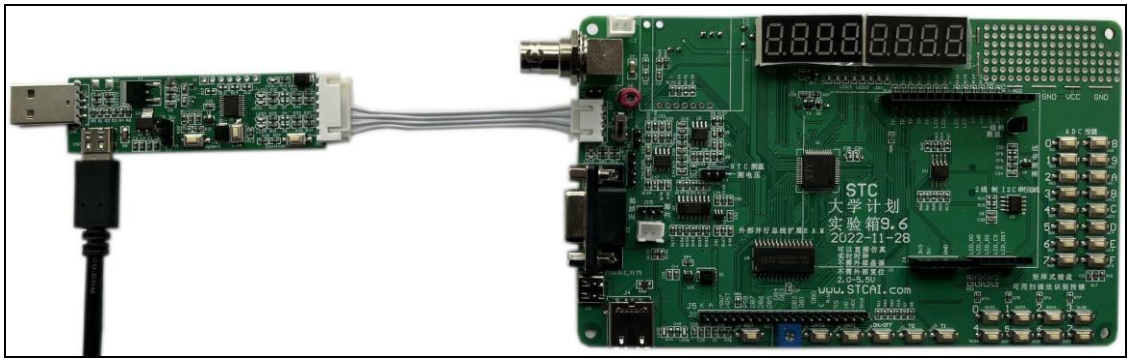
3、一箭双雕用 SIP7-USB-TypeC 对 STC8 系列 MCU 进行仿真/烧录， 硬件连接图如下：



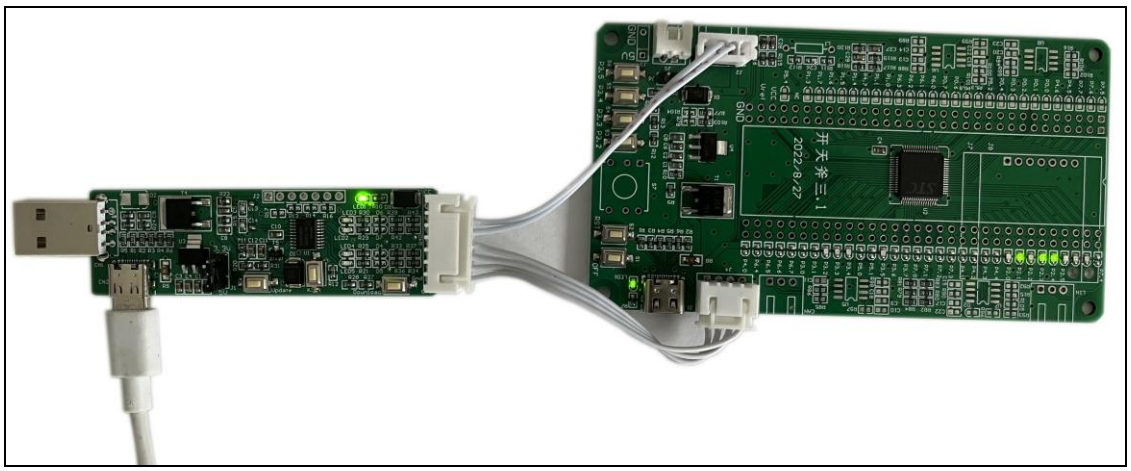
4、一箭双雕用 SIP7-USB-TypeA 对 STC8 系列 MCU 进行仿真/烧录， 硬件连接图如下：



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



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

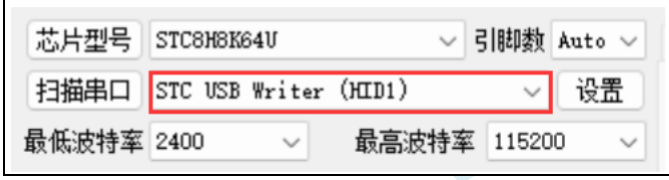


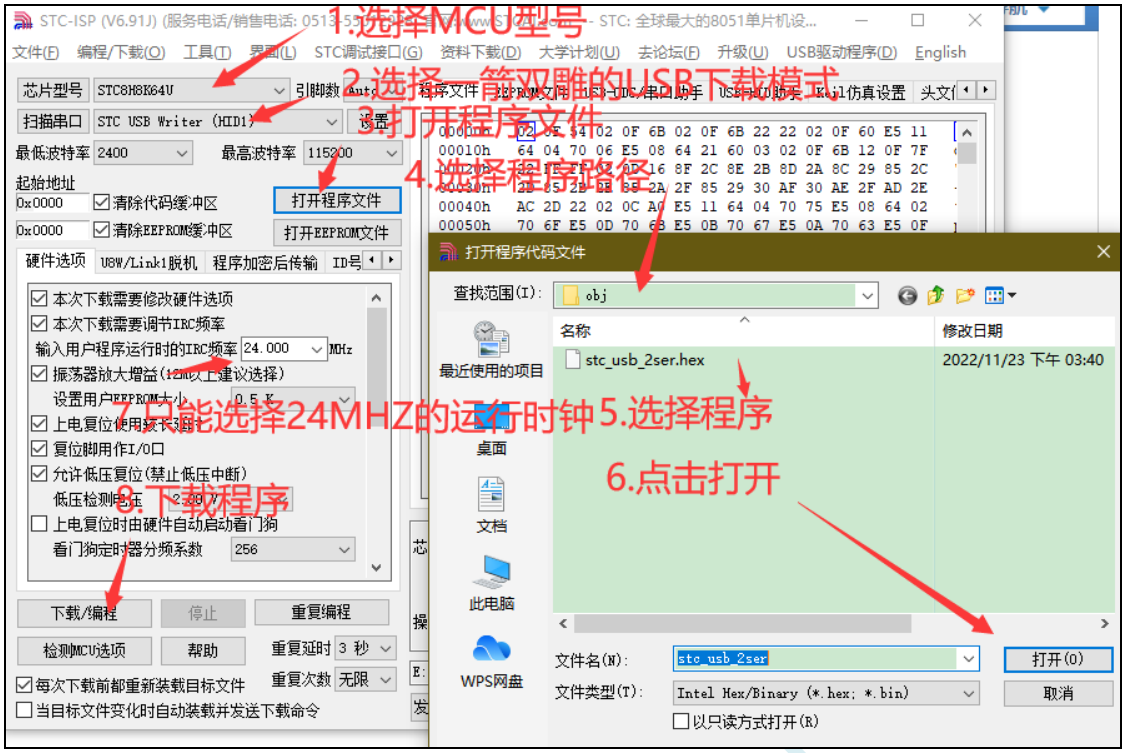
7、将一箭双雕设置成普通的下载工具，可以参考这个官网论坛的这个帖子：
https://www.stcaimcu.com/forum.php?mod=viewthread&tid=240&highlight=%E4%B8%80%E7%AE%AD%E5%8F%8C%E9%9B%95

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

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

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







软件设置如下：

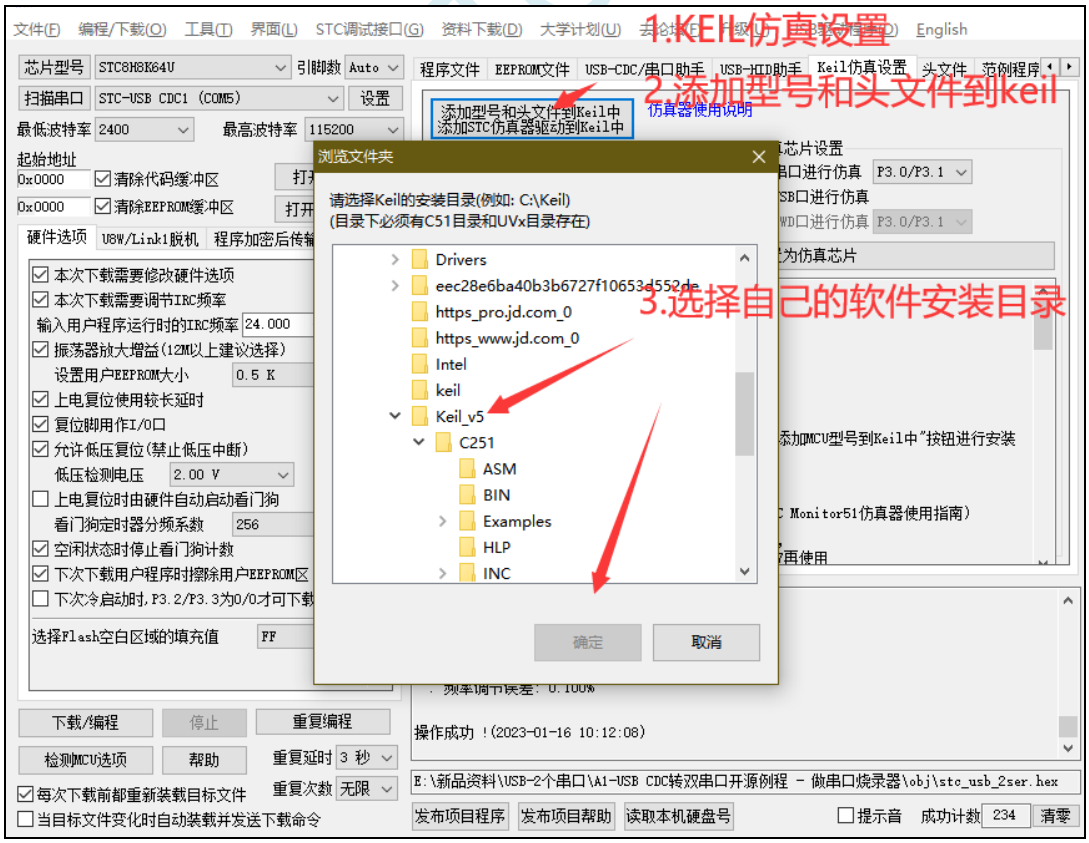
8、先去官网下载最新的 AIapp-ISP 软件，截止至目前最新版本是 AIapp-ISP，特别是仿真这块，AIapp-ISP 的 stcmon51 仿真驱动程序版本已更新至 v1.18，经内部反复测试已经非常稳定。

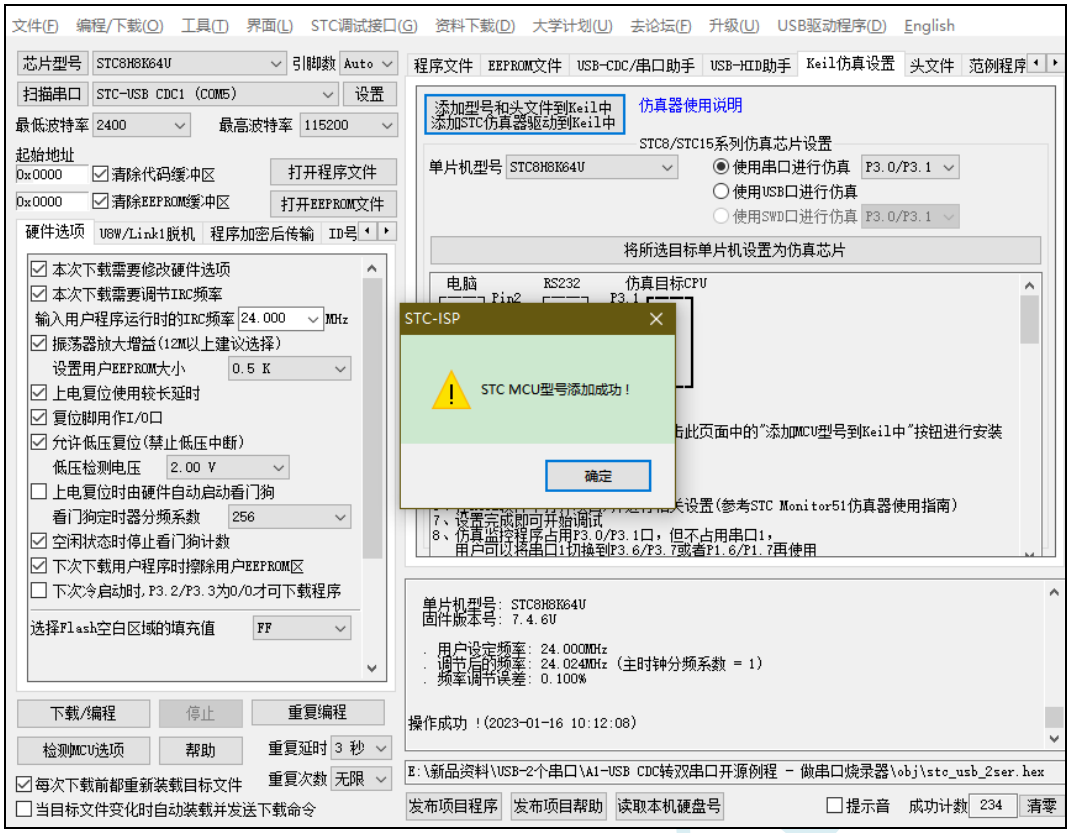
（下载地址：工具软件-深圳国芯人工智能有限公司 https://www.stcai.com/gjrj）



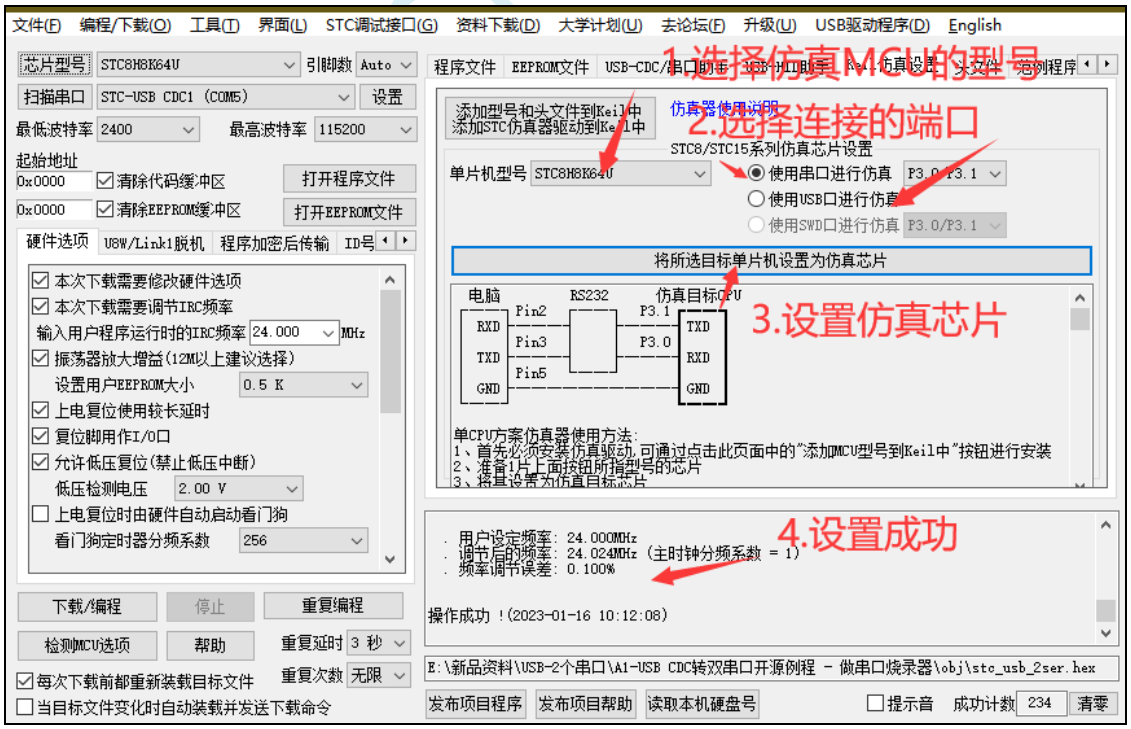
9、更新 KEIL 中的 STC 的资源：添加 STC 仿真器的固件和芯片型号到 KEIL 中。

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

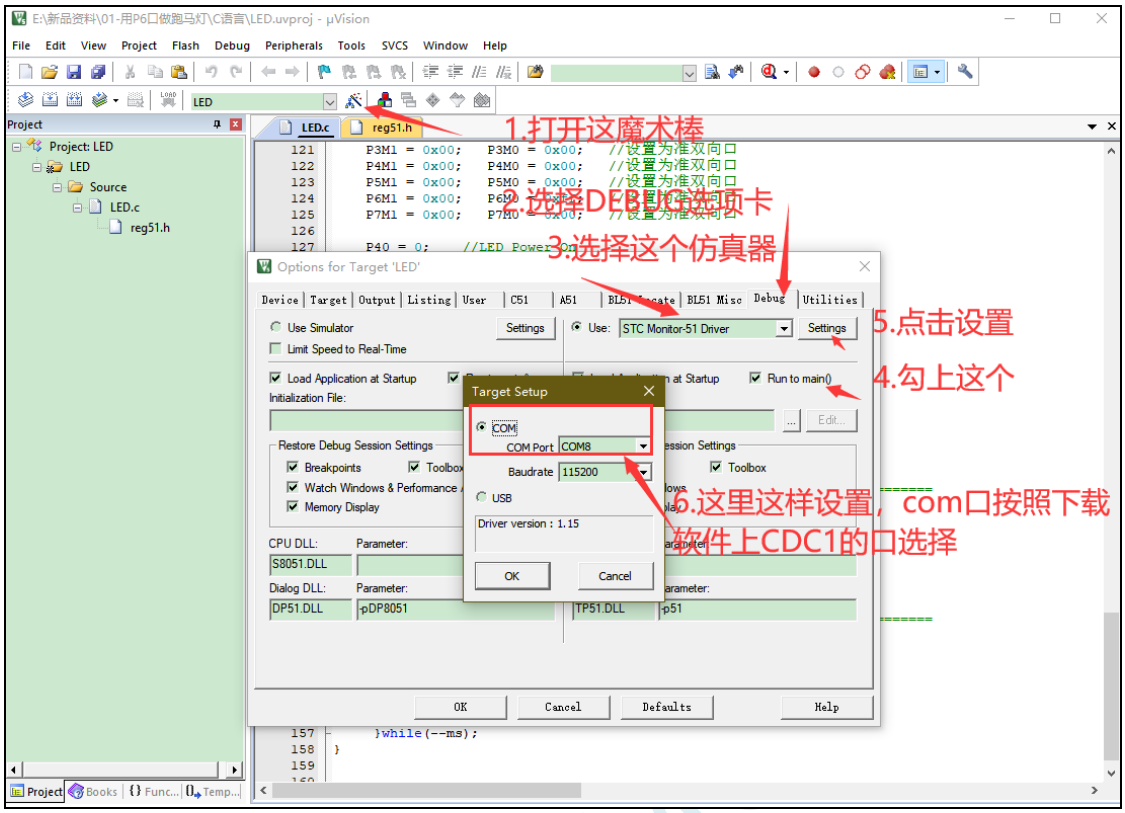




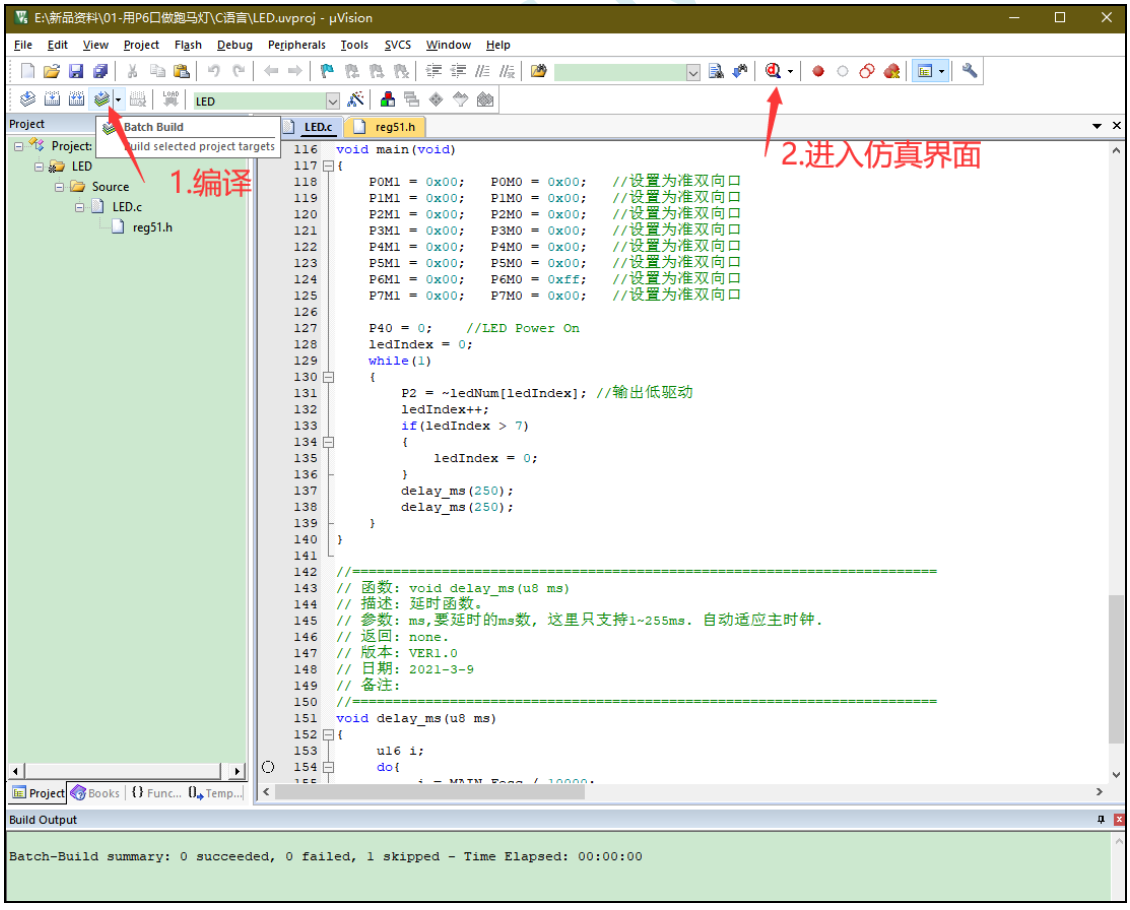
10、现在开始进行 KEIL 仿真的步骤，先将 STC8H8K64U 设置为仿真芯片，STC8H8K64U 目前仅支持串口和 USB 直接仿真。（这里选择了 P3.0/3.1 作为仿真端口，所以程序里不能出现任何占用 3.0 和 3.1 引脚的功能，仿真注意事项贴中也会说明，先用点亮一个 LED 的程序进行测试，比较容易观察结果！）
此时连接 STC8 的芯片，然后进行如下的设置将开天斧设置成仿真的主控芯片。
（注意一下这里的 IRC 频率一定要和程序里设置的主时钟一样！！）



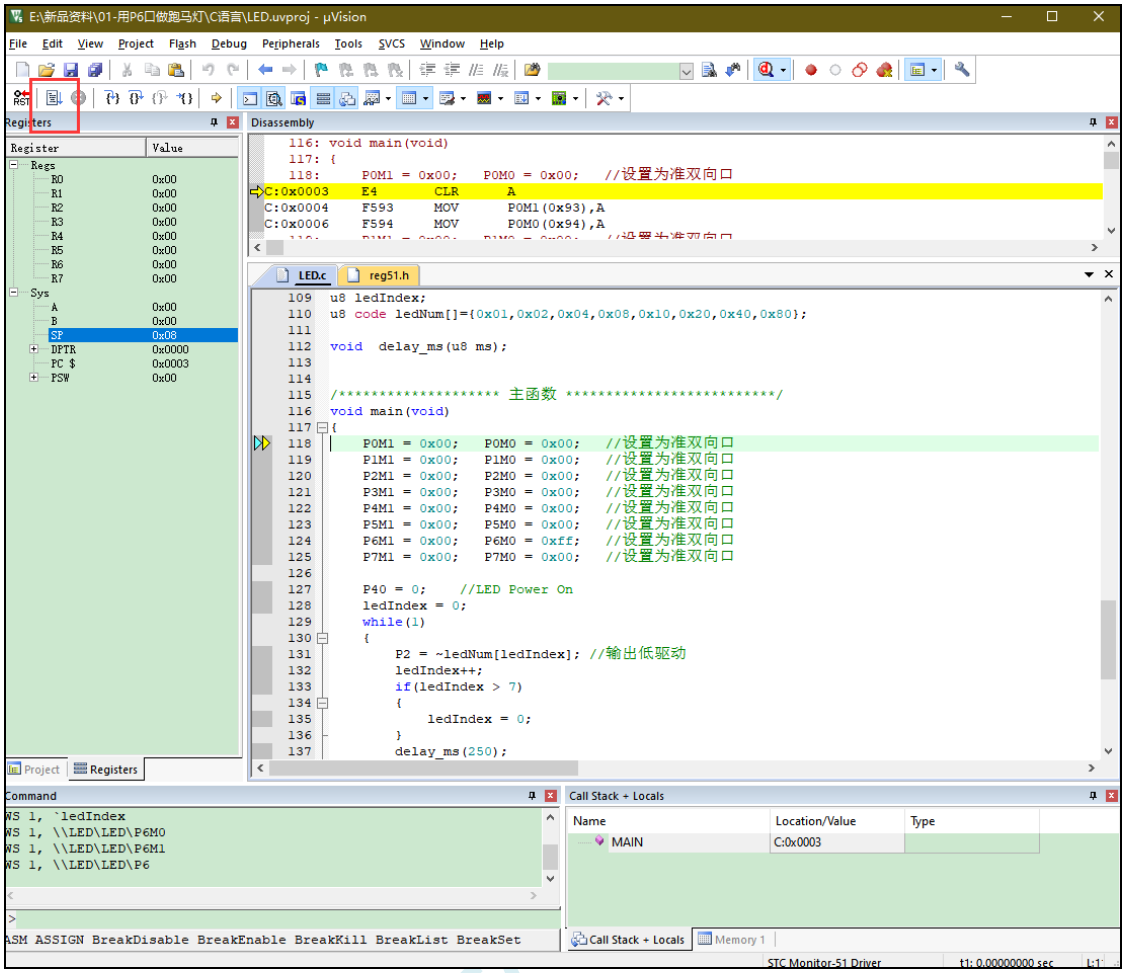
11、开天斧设置为仿真芯片（如何在 keil 中创建工程不在此贴详述），打开 KEIL 中建好的工程项目，直接去 KEIL 中设置仿真的路径。



12、这样就可以编译并且调试了。



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



5.13 AIapp-ISP 下载软件高级应用

5.13.1 发布项目程序

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

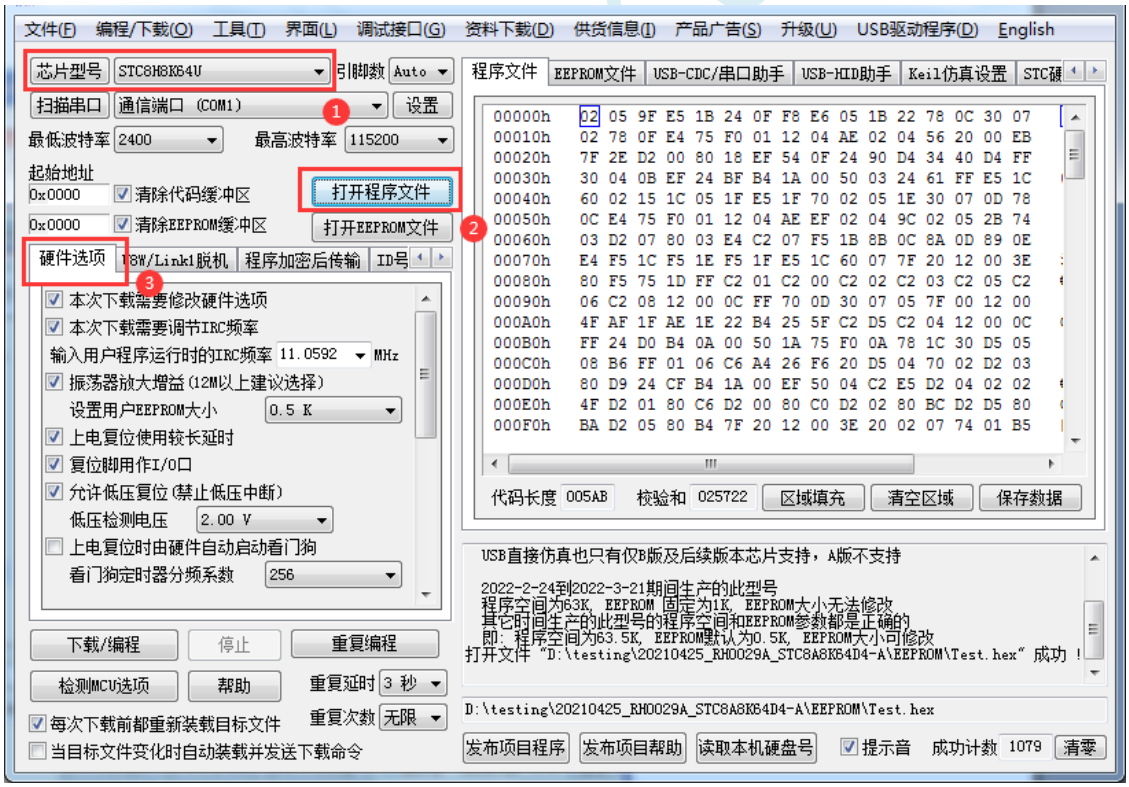
关于界面，用户可以自己进行定制（用户可以自行修改发布项目程序的标题、按钮名称以及帮助信息），同时用户还可以指定目标电脑的硬盘号和目标芯片的 ID 号，指定目标电脑的硬盘号后，便可以控制发布应用程序只能在指定的电脑上运行（防止烧录人员将程序轻易从电脑盗走,如通过网络发走,如通过U盘烤走,防不胜防,当然盗走你的电脑那就没办法那,所以 STC 的脱机下载工具比电脑烧录安全,能限制可烧录芯片数量,让前台文员小姐烧,让老板娘烧都可以），拷贝到其它电脑，应用程序不能运行。同样的，当指定了目标芯片的 ID 号后，那么用户代码只能下载到具有相应 ID 号的目标芯片中（对于一台设备要卖几千万的产品特别有用---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

发布项目程序详细的操作步骤如下：

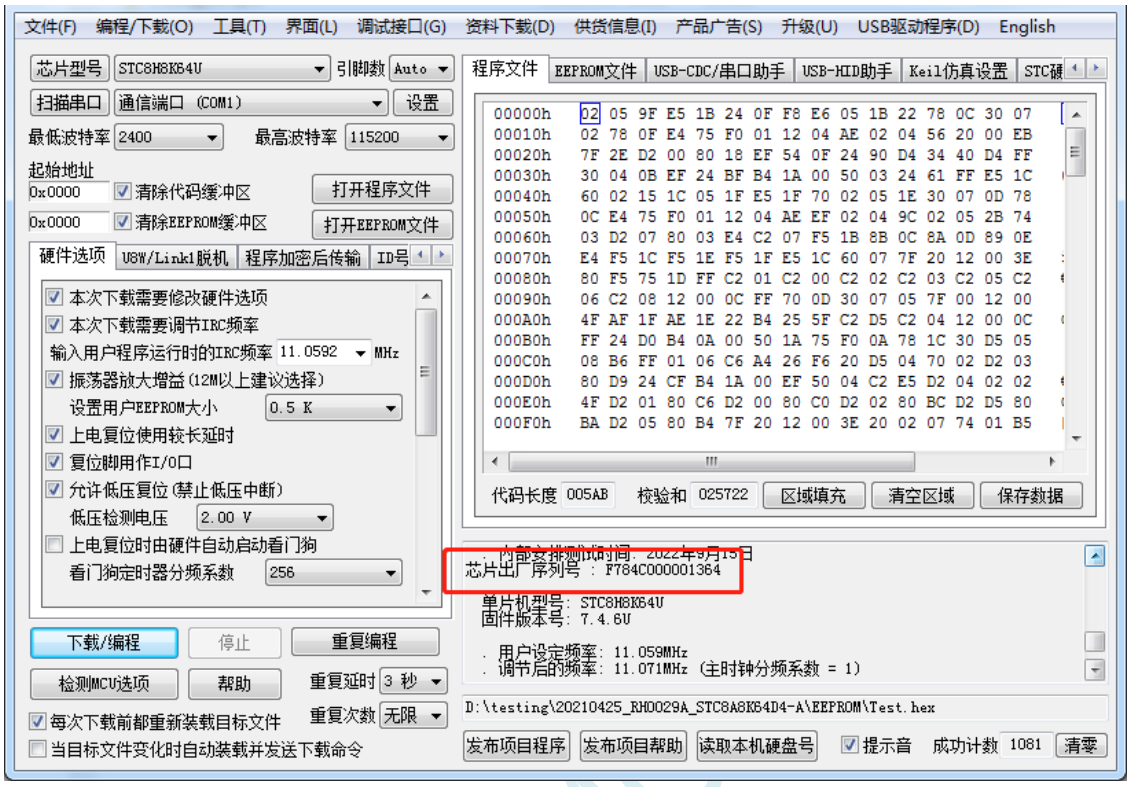
1、首先选择目标芯片的型号

2、打开程序代码文件

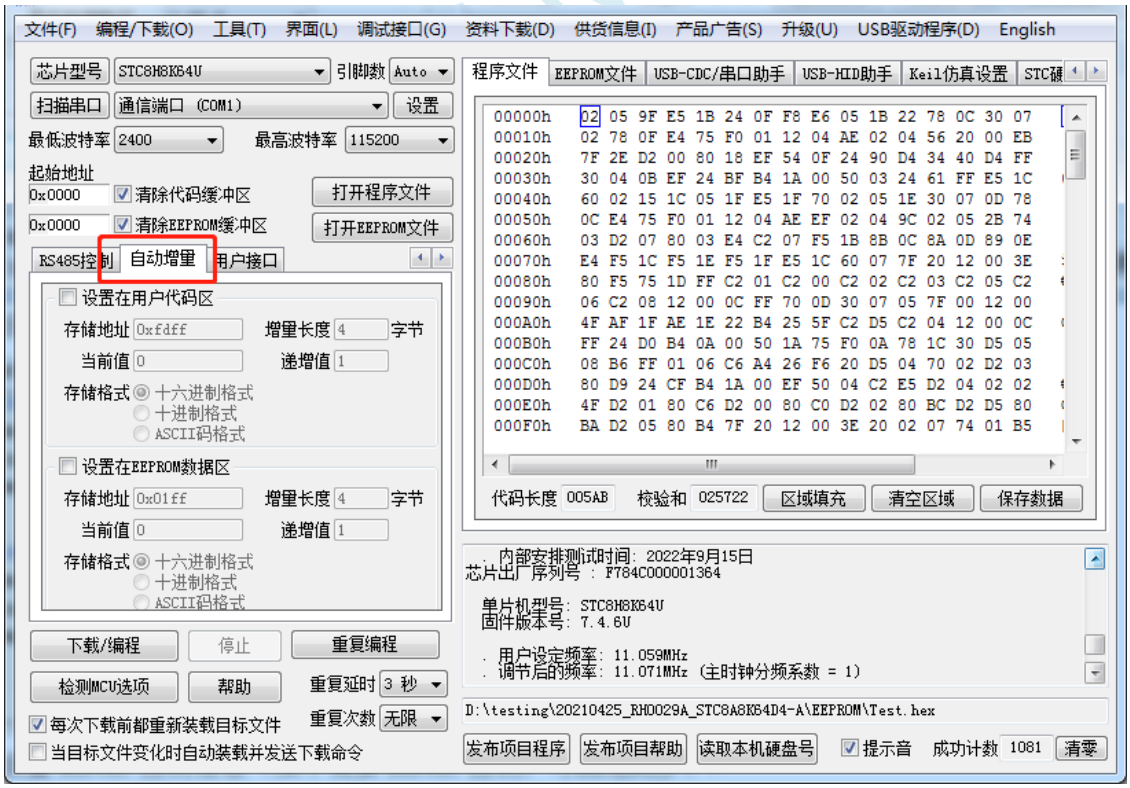
3、设置好相应的硬件选项



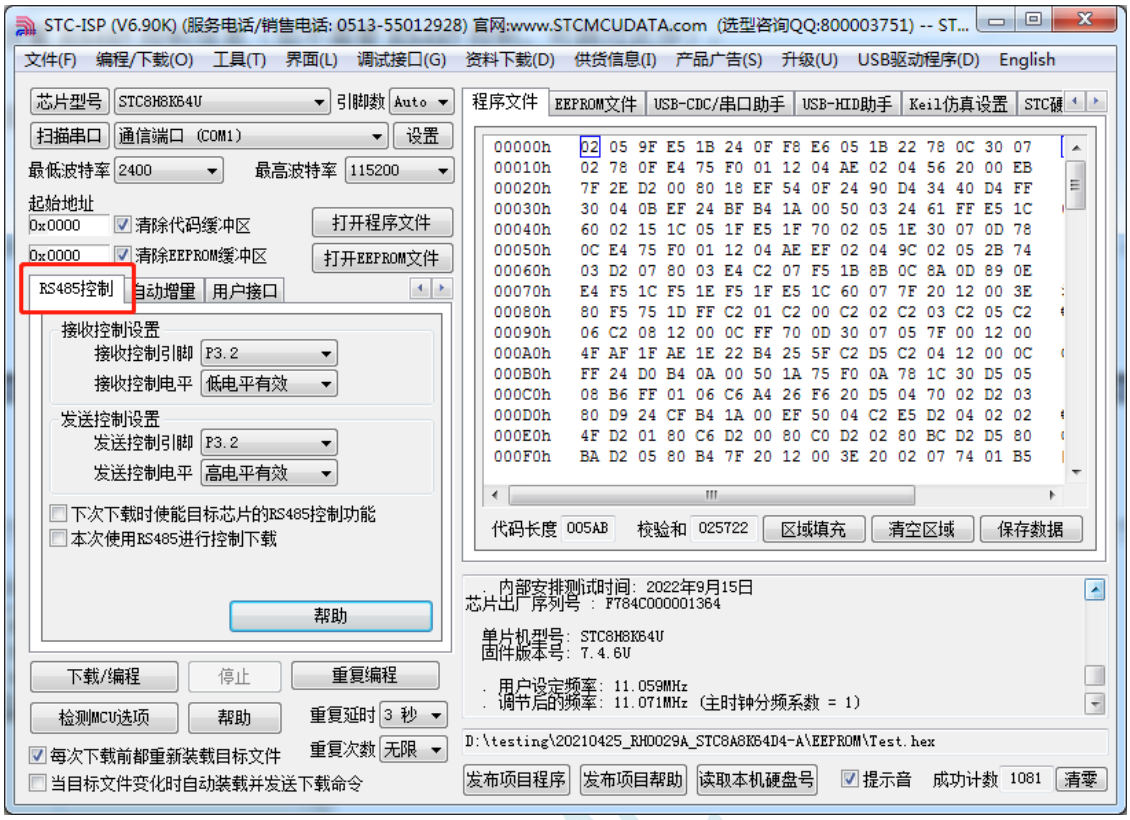
4、试烧一下芯片，并记下目标芯片的 ID 号，如下图所示，该芯片的 ID 号即为“F784C000001364”（如不需要对目标芯片的 ID 号进行校验，可跳过此步）



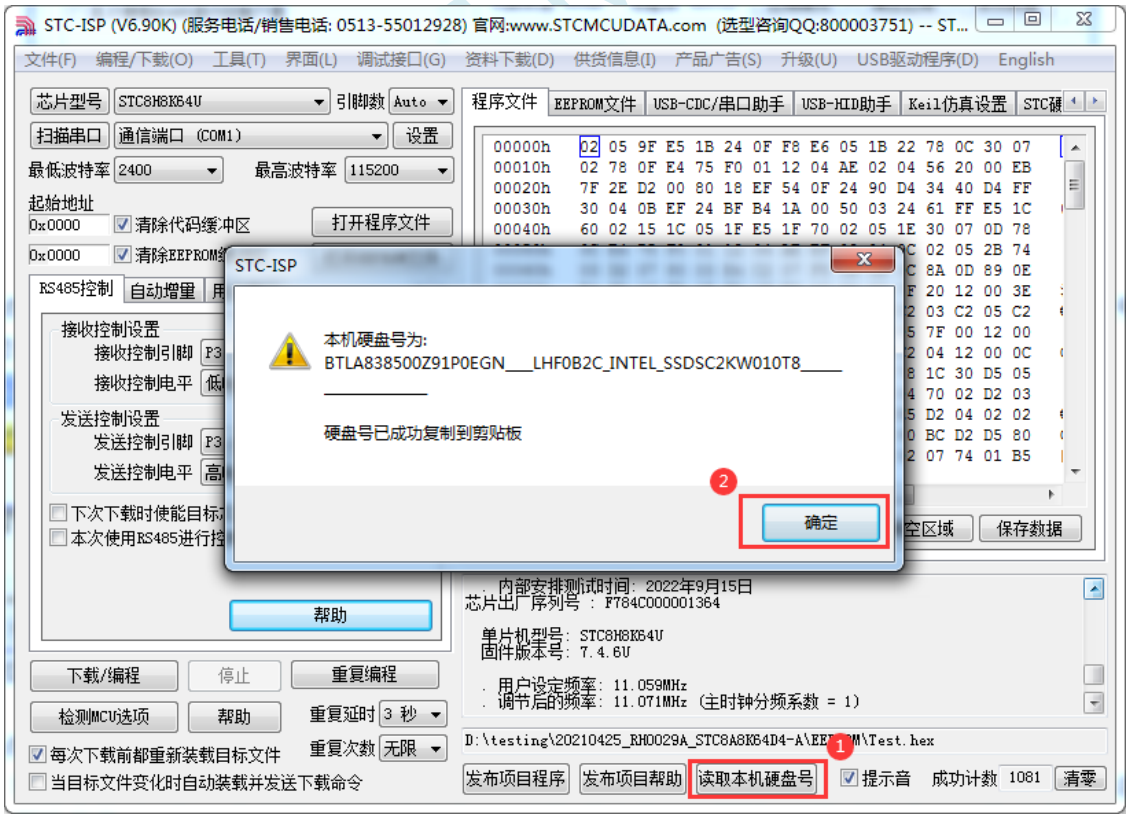
5、设置自动增量（如不需要自动增量，可跳过此步）



6、设置 RS485 控制信息（如不需要 RS485 控制，可跳过此步）



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

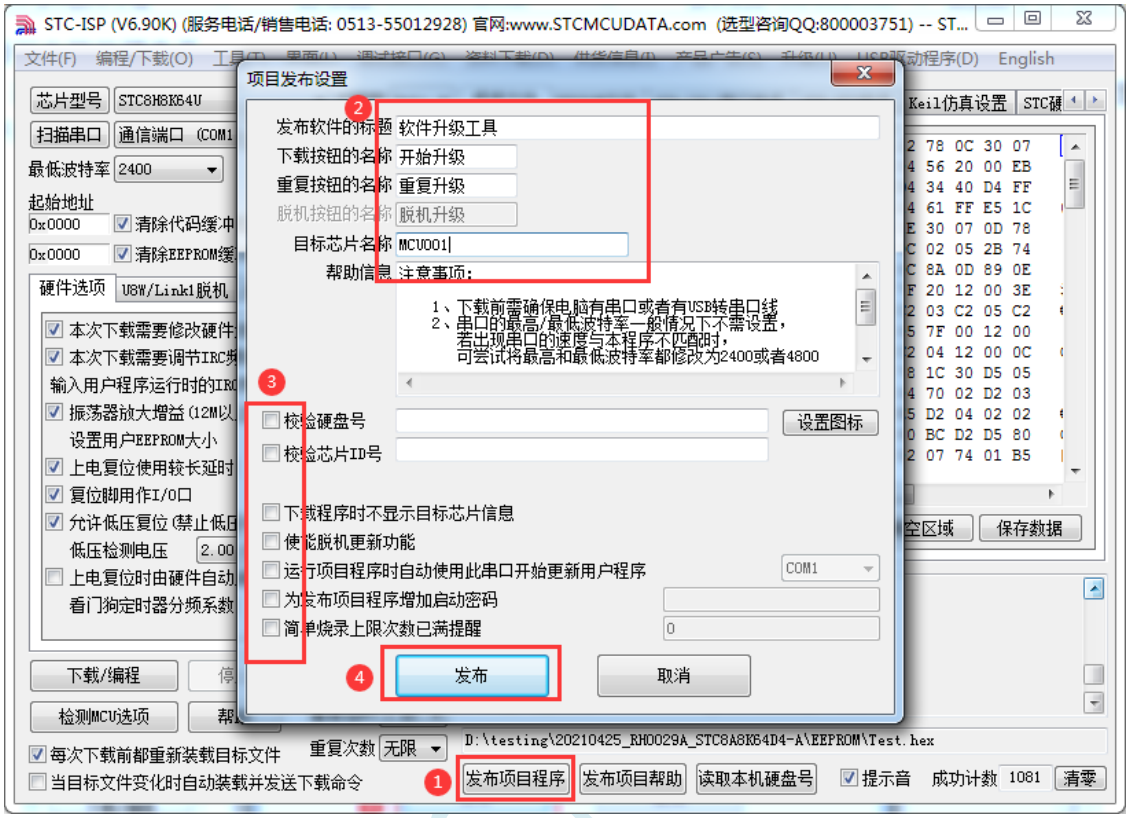


8、点击“发布项目程序”按钮，进入发布应用程序的设置界面。

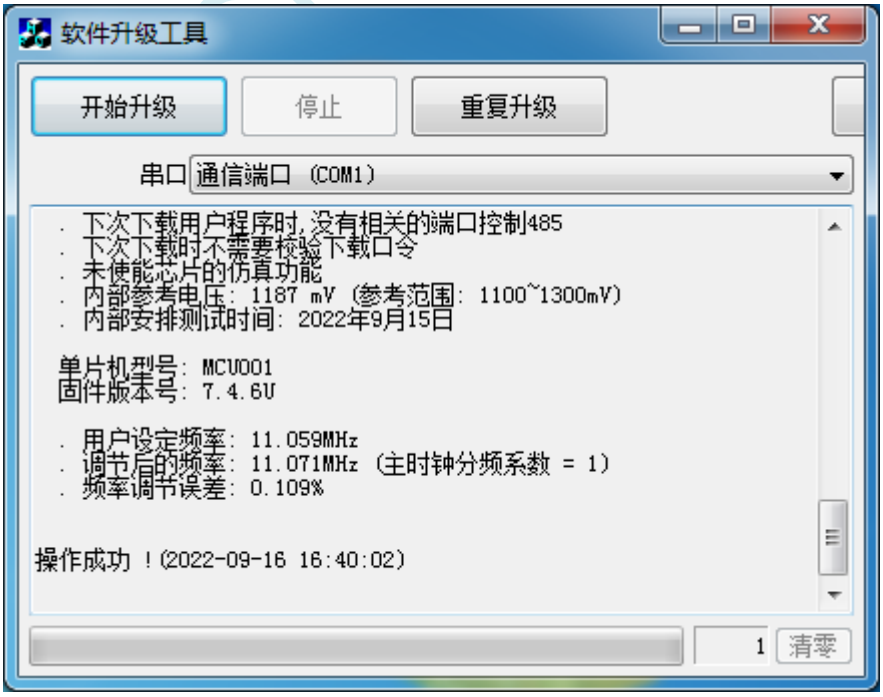
9、根据各自的需要，修改发布软件的标题、下载按钮的名称、重复下载按钮的名称、自动增量的名称以及帮助信息

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

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



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



5.13.2　程序加密后传输（防烧录时串口分析出程序）

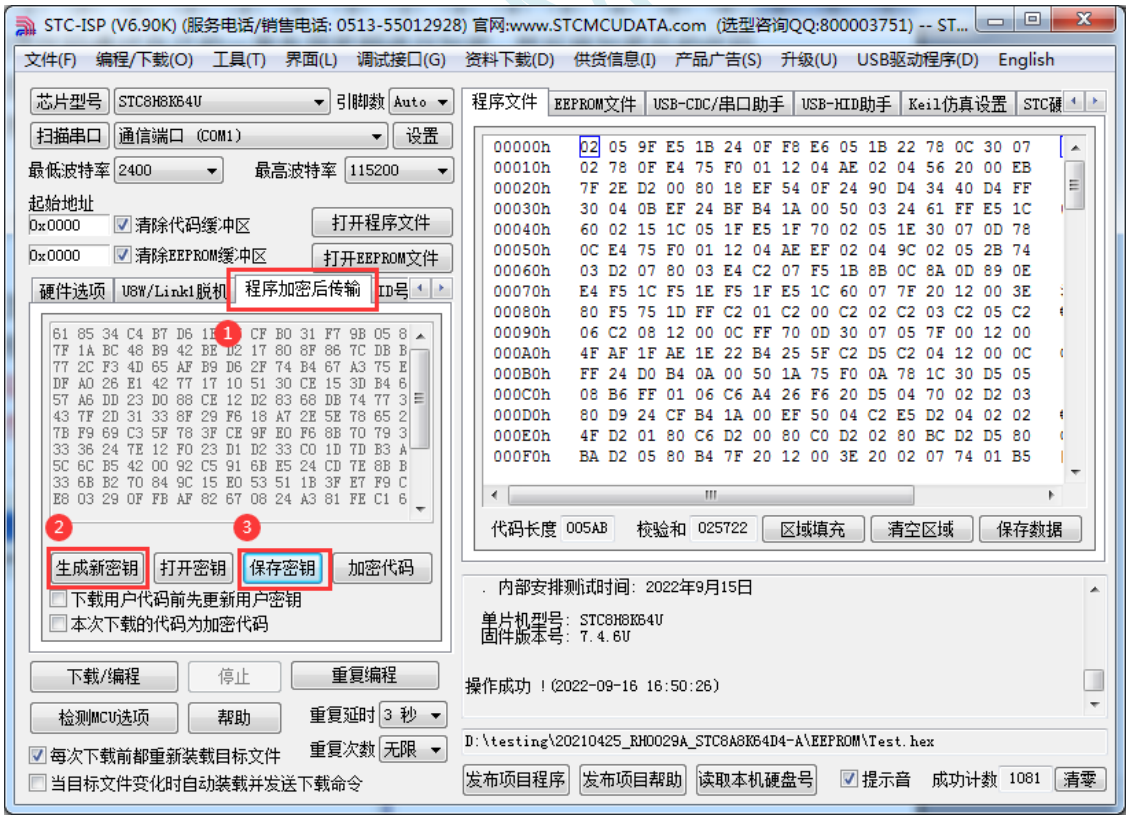
目前，所有的普通串口下载烧录编程都是采用明码通信的（电脑和目标芯片通信时，或脱机下载板和目标芯片通信时），问题：如果烧录人员通过分析下载烧录编程时串口通信的数据，高手是可以在烧录时在串口上引 2 根线出来，通过分析串口通信的数据分析出实际的用户程序代码的。当然用 STC 的脱机下载板烧程序总比用电脑烧程序强（防止烧录人员将程序轻易从电脑盗走，如通过网络发走，如通过 U 盘烤走，防不胜防，当然盗走你的电脑那就没办法那，所以 STC 的脱机下载工具比电脑烧录安全，让前台文员小姐烧，让老板娘烧都可以）。即使是 STC 全球首创的脱机下载工具，对于要防止天才的不法分子在脱机下载工具烧录的过程中通过分析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

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

程序加密后传输功能的使用需要如下的几个步骤：

1、生成并保存新的密钥

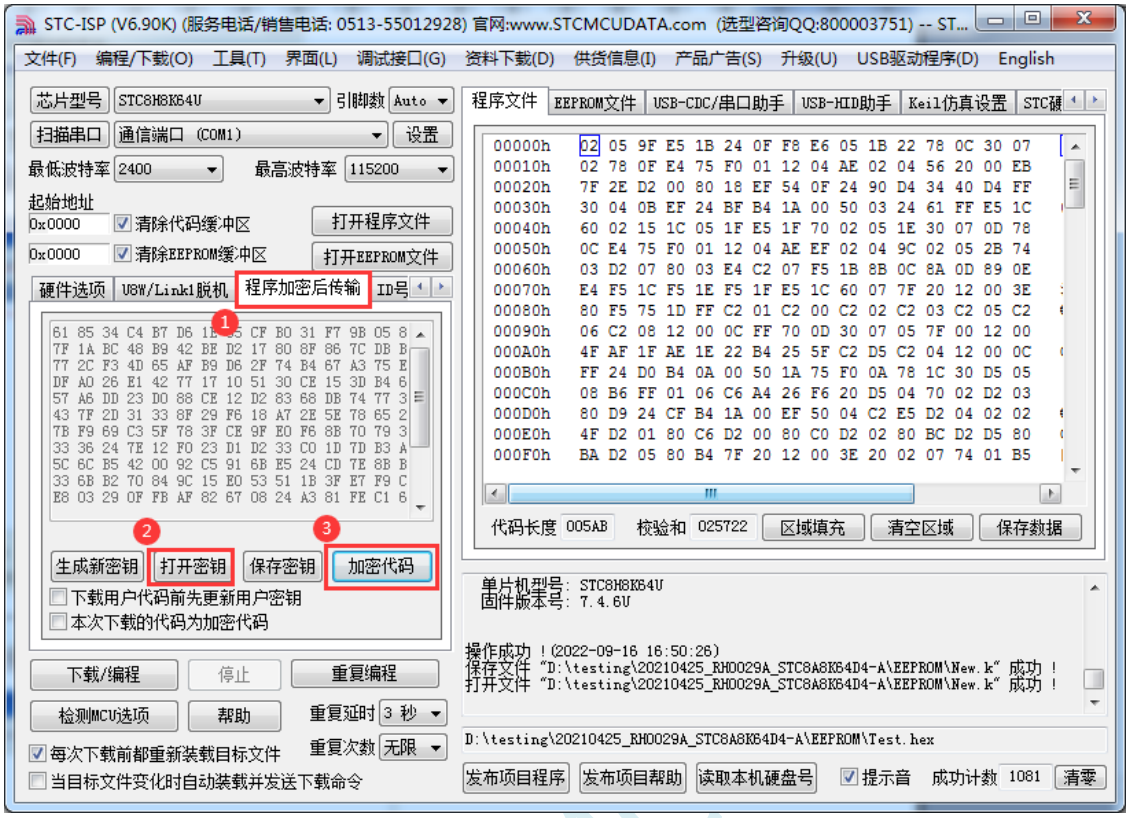
如下图，进入到“程序加密后传输”页面，点击“生成新密钥”按钮，即可在缓冲区显示新生成的 256 字节的密钥。然后点击“保存密钥”按钮，即可将生成的新密钥保存为以“.K”为扩展名的密钥文件（**注意：这个密钥文件一定要保存好，以后发布的代码文件都需要使用这个密钥加密，而且这个密钥的生成是非重复的，即任何时候都不可能生成两个完全相同的密钥，所以一旦密钥文件丢失将无法重新获得**）。例如我们将密钥保存为“New.k”。



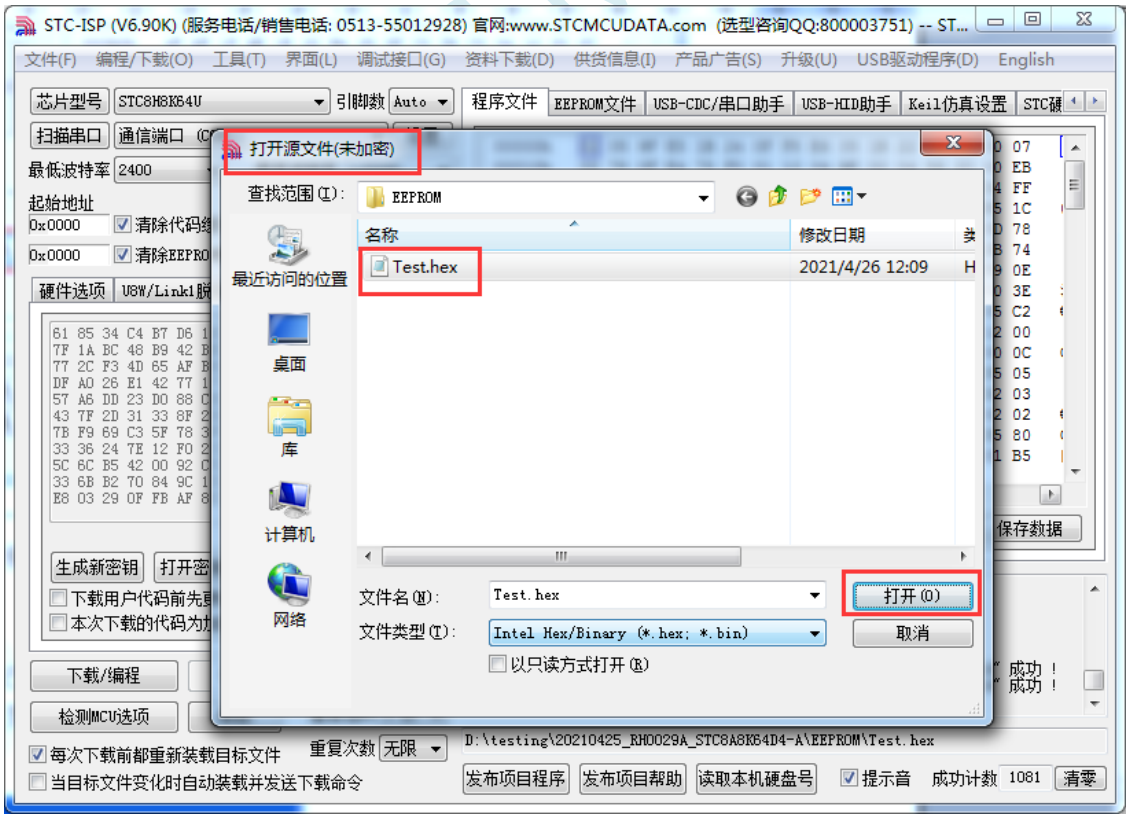
2、对代码文件加密

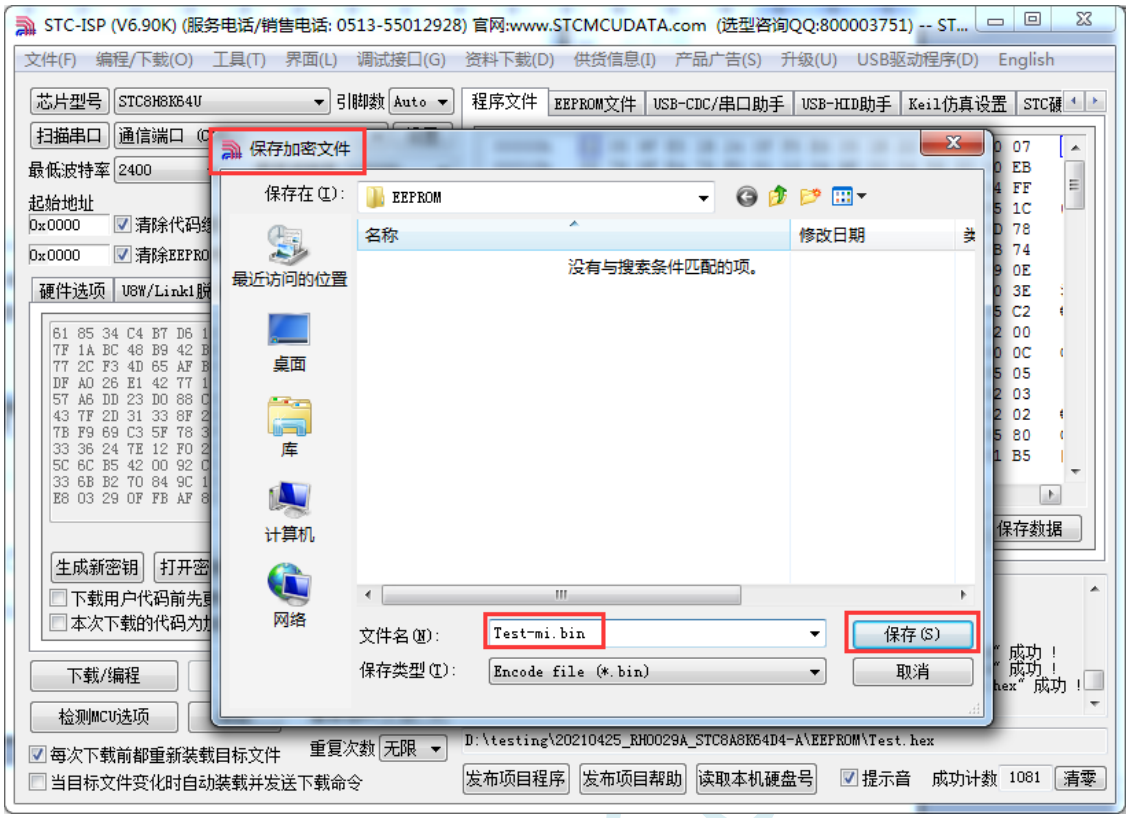
加密文件前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“程序加密后传输”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，例如“New.k”，然后返回到“程序加密后传输”页面中点击“加密代码”按钮，如下图所示，首

先会弹出“打开源文件（未加密）”的对话框，此时选择的是原始的未加密的代码文件



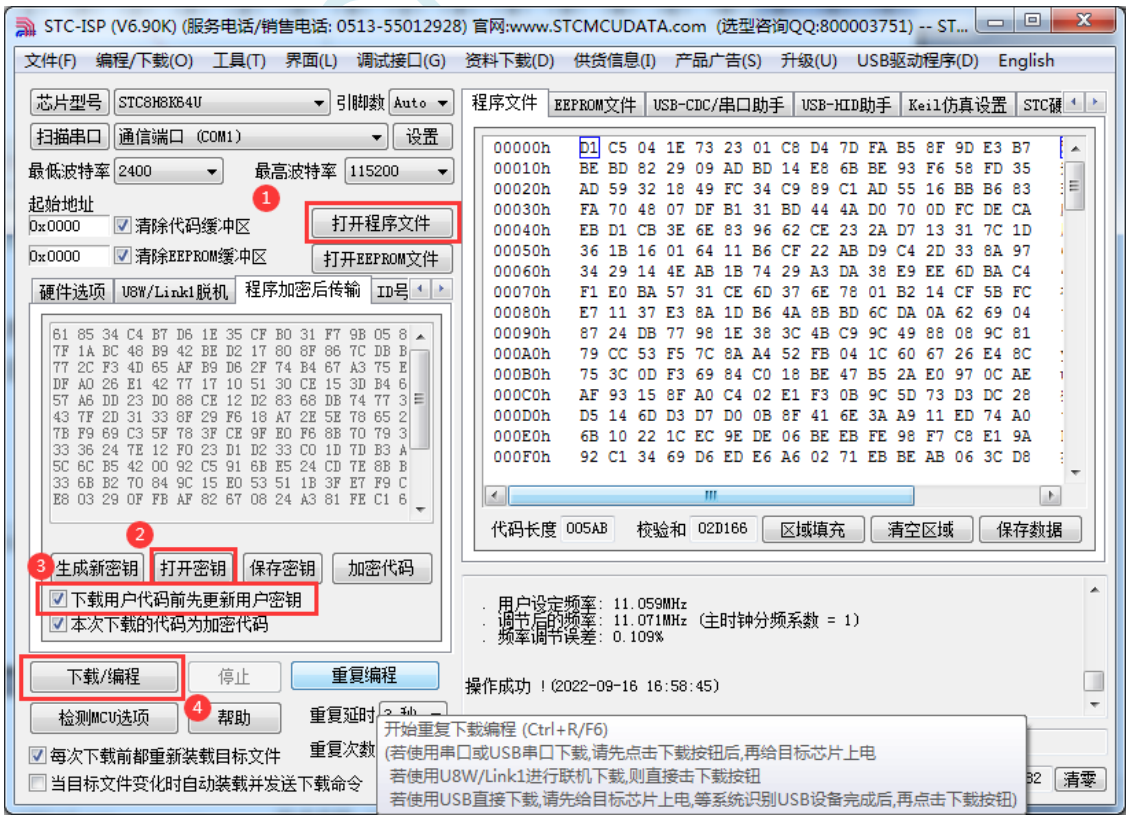
点击打开按钮后，马上有会弹出一个类似的对话框，但此时是对加密后的文件进行保存的对话框。如下图所示，点击保存按钮即可保存加密后的文件。





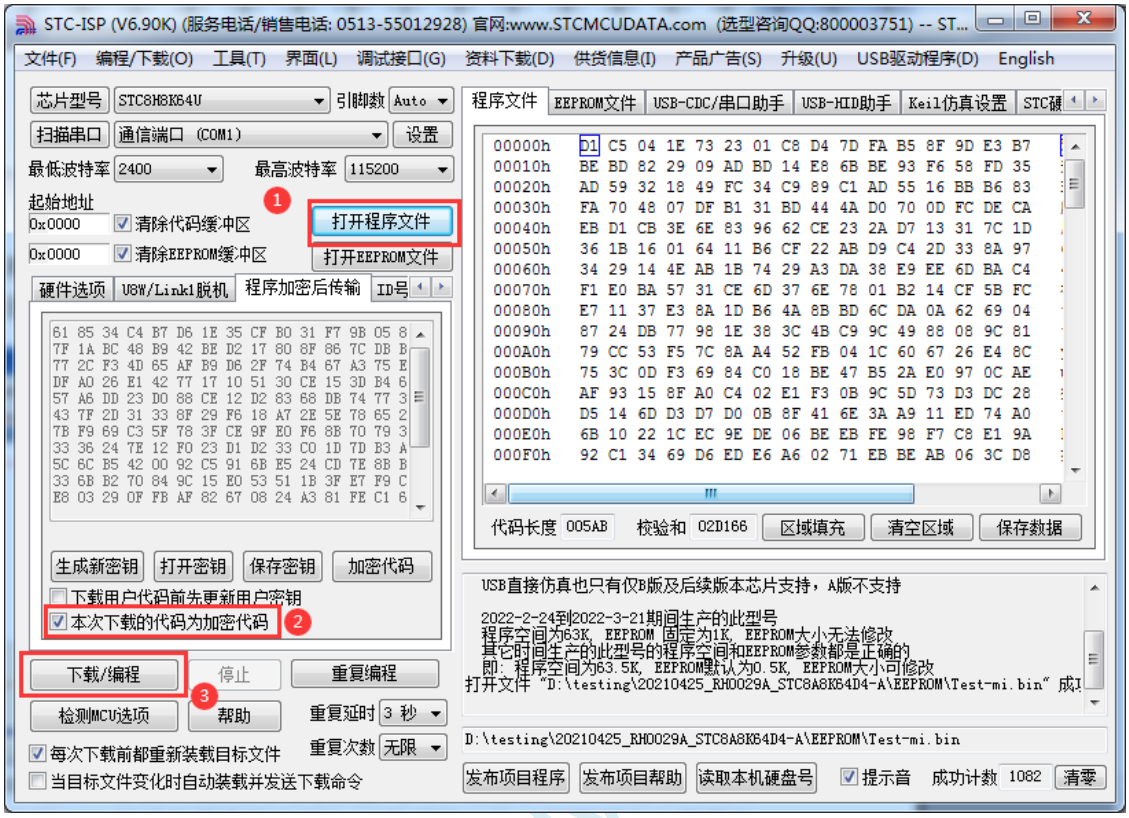
3、将用户密钥更新到目标芯片中

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



4、加密更新用户代码

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

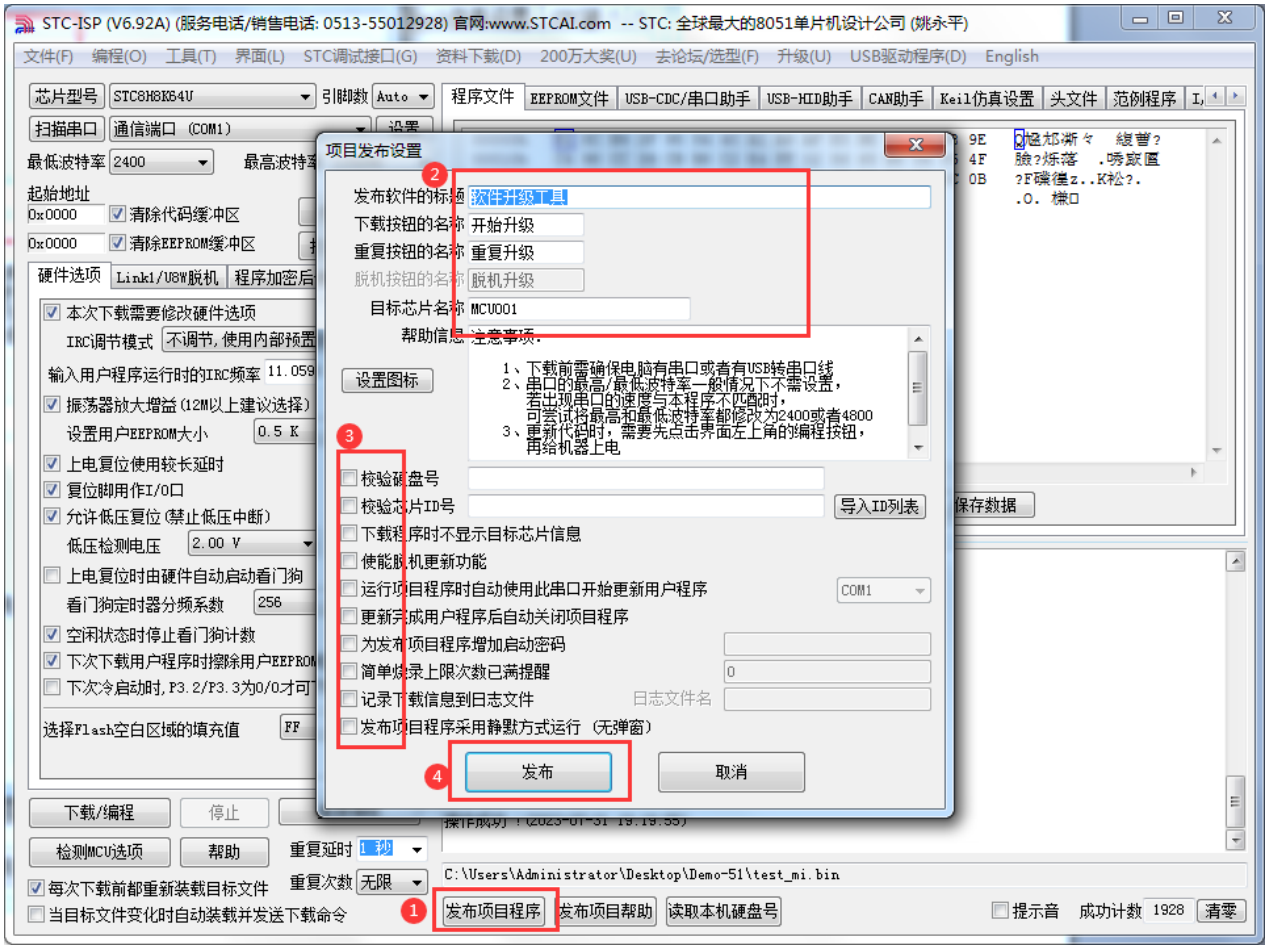


对于一片新的 STC 单片机，可将步骤 3 和步骤 4 合并完成，即将密钥更新到目标单片机的同时也可将加密后的代码一并下载到单片机中，若已经执行过步骤 3（即已经将密钥更新到目标芯片中了），则后续的代码更新就只需要按照步骤 4，只需要在“程序加密后传输”页面中选择“本次下载的代码为加密代码”的选项（“下载用户代码前先更新用户密钥”选项不需要选了），然后打开我们之前加过密后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载即可完成用用户自己专用的加密文件更新用户代码的目的（防止在烧录程序时被烧录人员通过监测串口分析出代码的目的）。

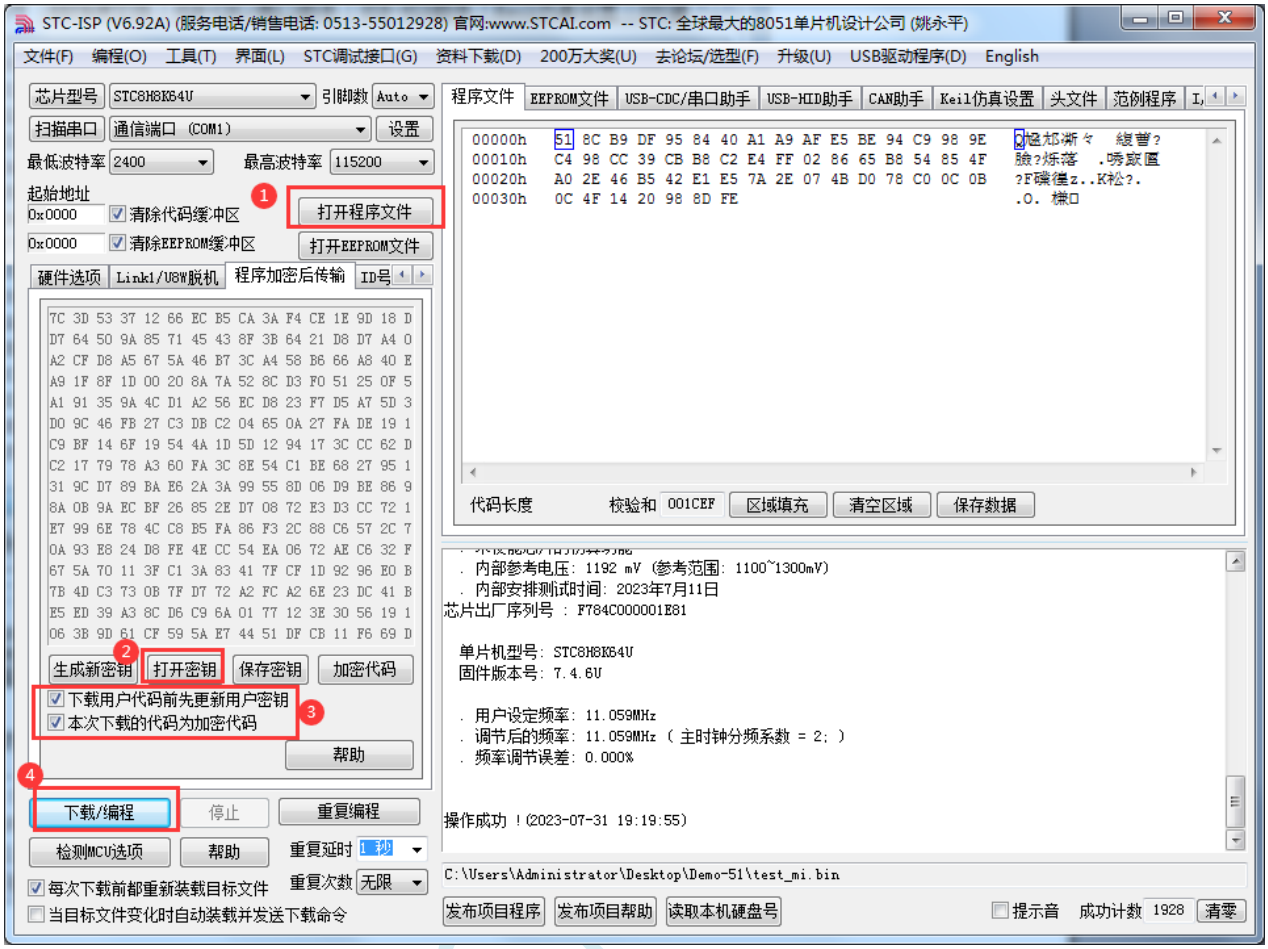
5.13.3　发布项目程序+程序加密后传输结合使用

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

发布项目程序可参考 5.16.1 章节步骤，示意图如下：



程序加密后传输可参考 5.16.2 章节步骤，示意图如下：

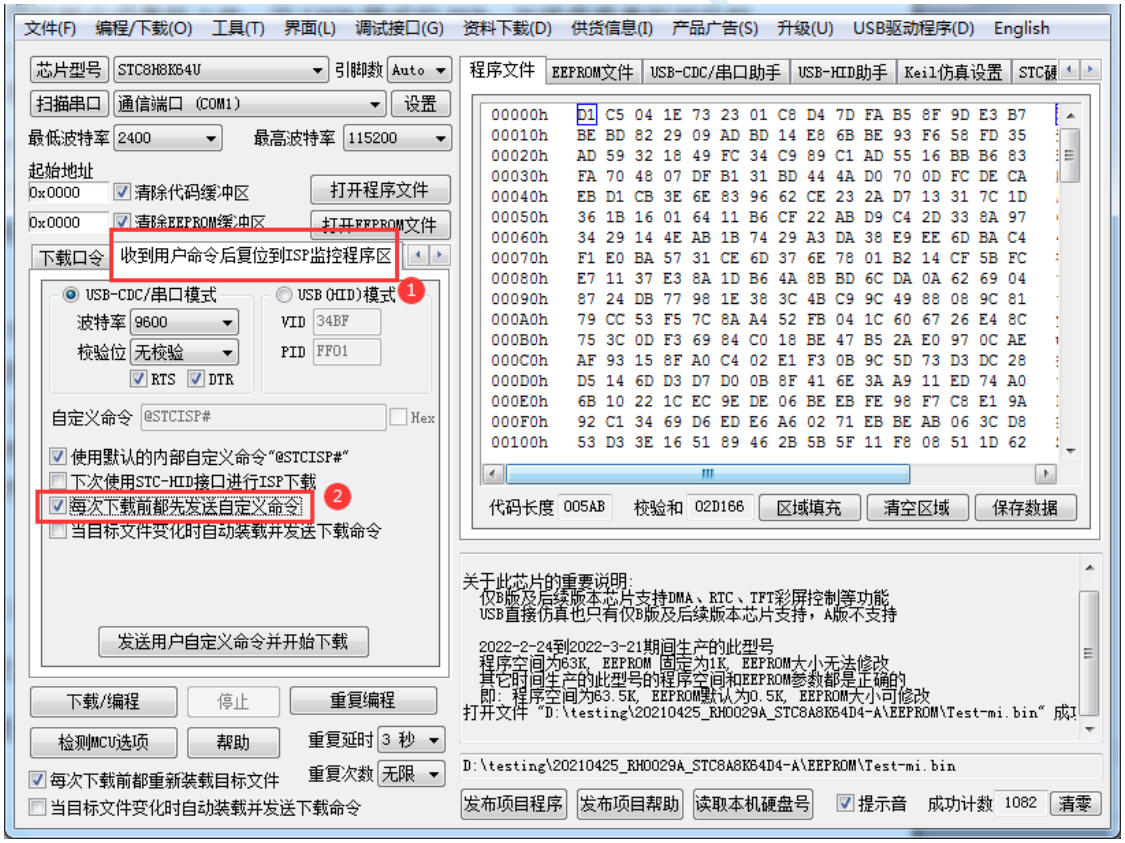


5.13.4　用户自定义下载（实现不停电下载）

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

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

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



实现不停电ISP下载的步骤如下：

1、编写用户代码，并在用户代码中添加串口命令监控程序

（参考代码如下，测试单片机型号为STC8H8K64U）

```
#include "stc8h.h"

#define    FOSC        11059200UL
```

```
#define     BAUD        (65536 - FOSC/4/115200)

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

void uart_isr()   interrupt 4
{
    char dat;

    if (TI)
    {
        TI = 0;
    }

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

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

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

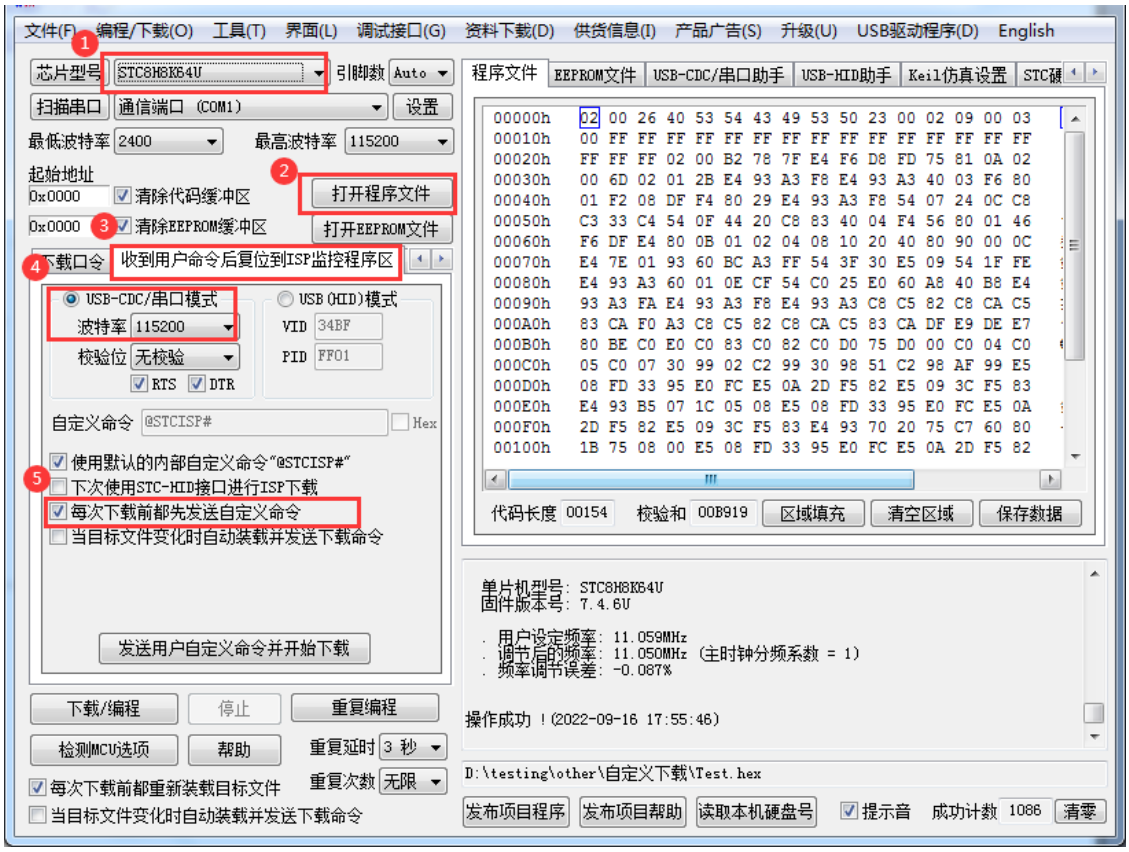
    SCON = 0x50;                            //串口初始化
    AUXR = 0x40;
    TMOD = 0x00;
    TH1 = BAUD >> 8;
    TL1 = BAUD;
```

```
    TR1 = 1;
    ES = 1;
    EA = 1;

    index = 0;                          //初始化命令

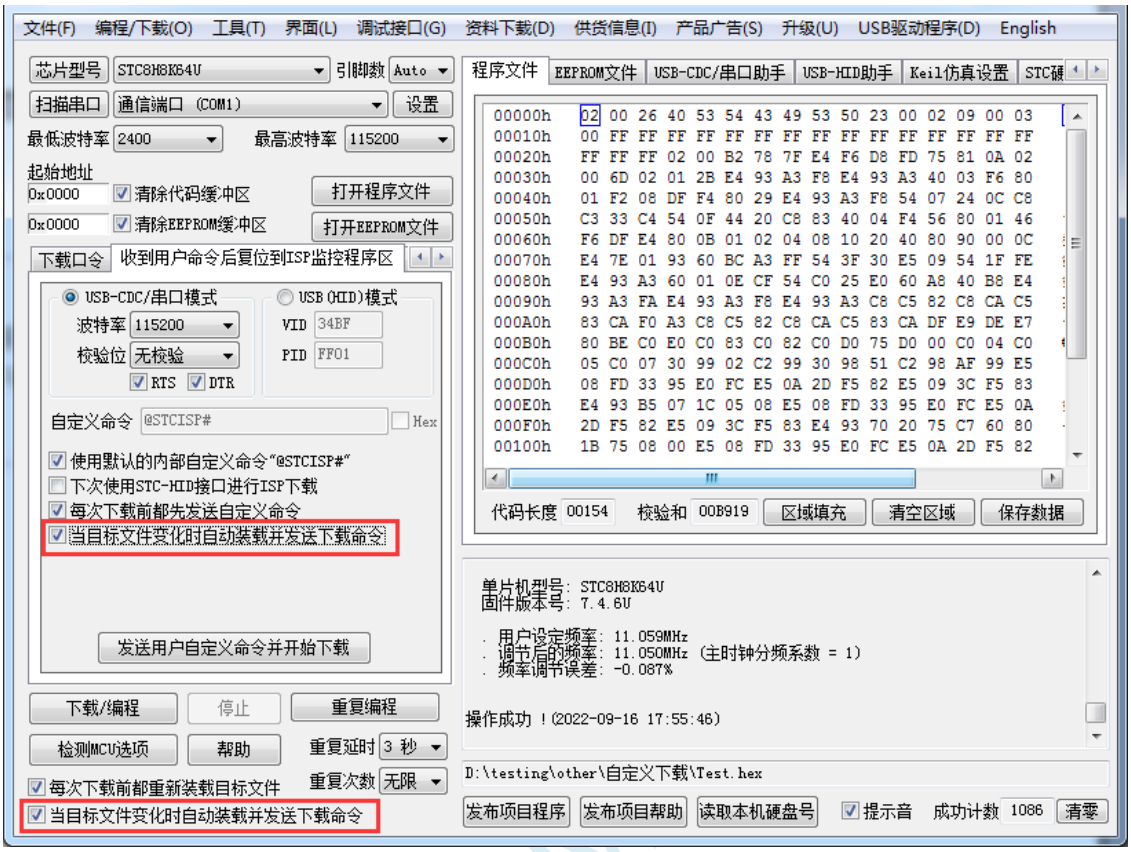
    while (1);
}
```

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



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

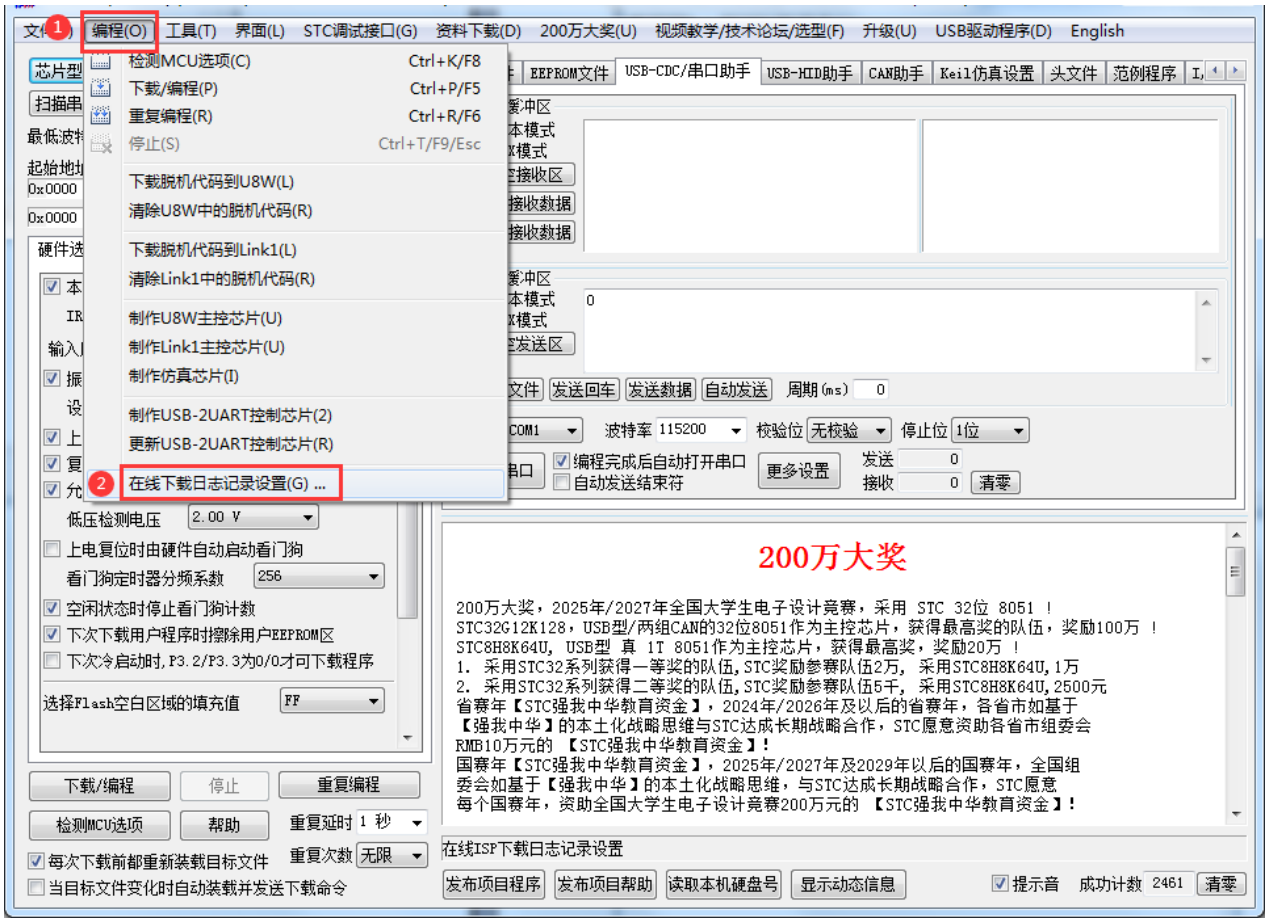
4、AIapp-ISP 还可实现项目开发阶段，完全自动下载功能，即当下载软件侦测到目标代码被更新了，就会自动发送下载命令。要实现这个功能只需要勾选下图中的两个选项中的任意一个即可



5.13.5 如何简单的控制下载次数，通过 ID 号来限制实际可以下载的 MCU 数量

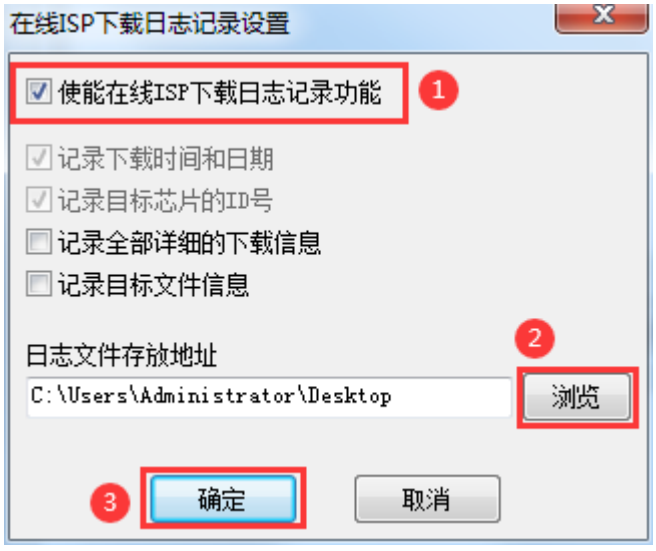
————————下载日志+发布项目高级应用

第一步、打开下载日志记录功能



1、打开“编程”菜单

2、点击“在线下载日志记录设置”，打开下面窗口

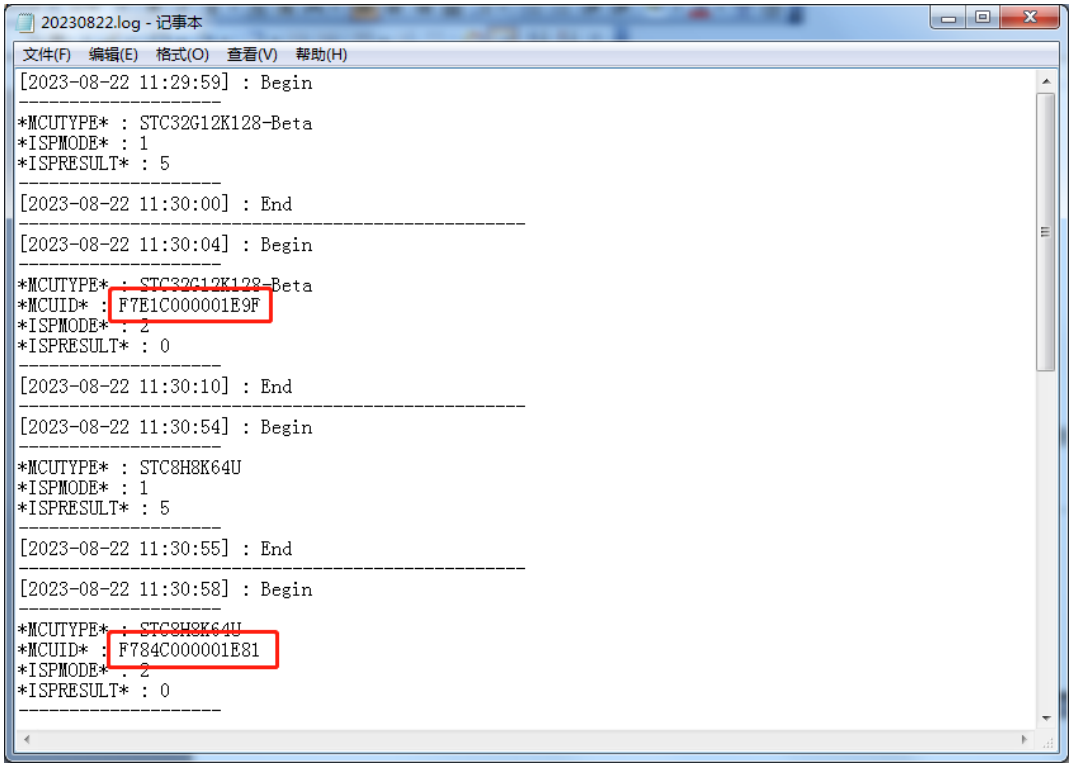


1、勾选“使能在线 ISP 下载日志记录功能”
2、点击“浏览”按钮选择日志文件存放目录
3、点击“确定”进行确认
设置完成后，接下来所有的 ISP 在线下载的下载信息都会自动记录到文件中，日志文件的文件名为当天的日期，扩展名为 log

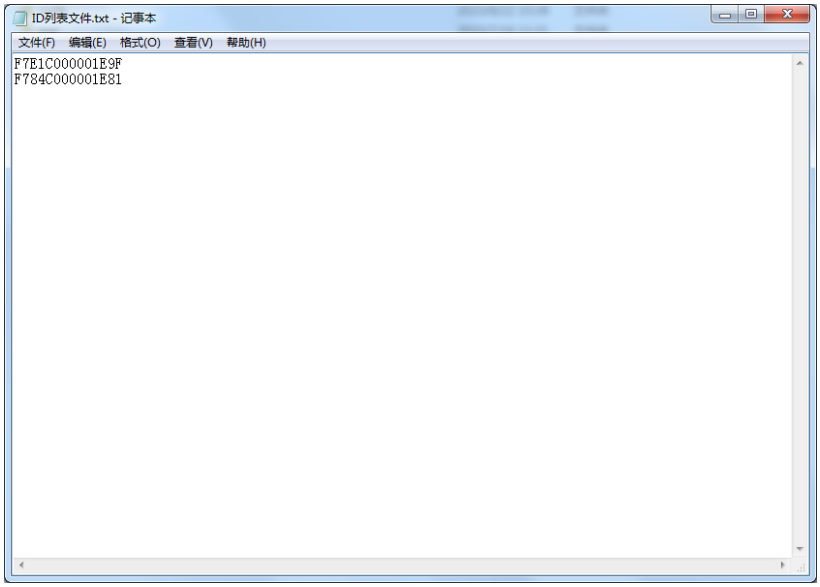
第二步、从下载日志文件中导出 ID 号列表到列表文件

（注：Ver6.92D 版本及之后的 ISP 软件可自动从列表中导入 ID 号，如需自动导入可跳过此步）

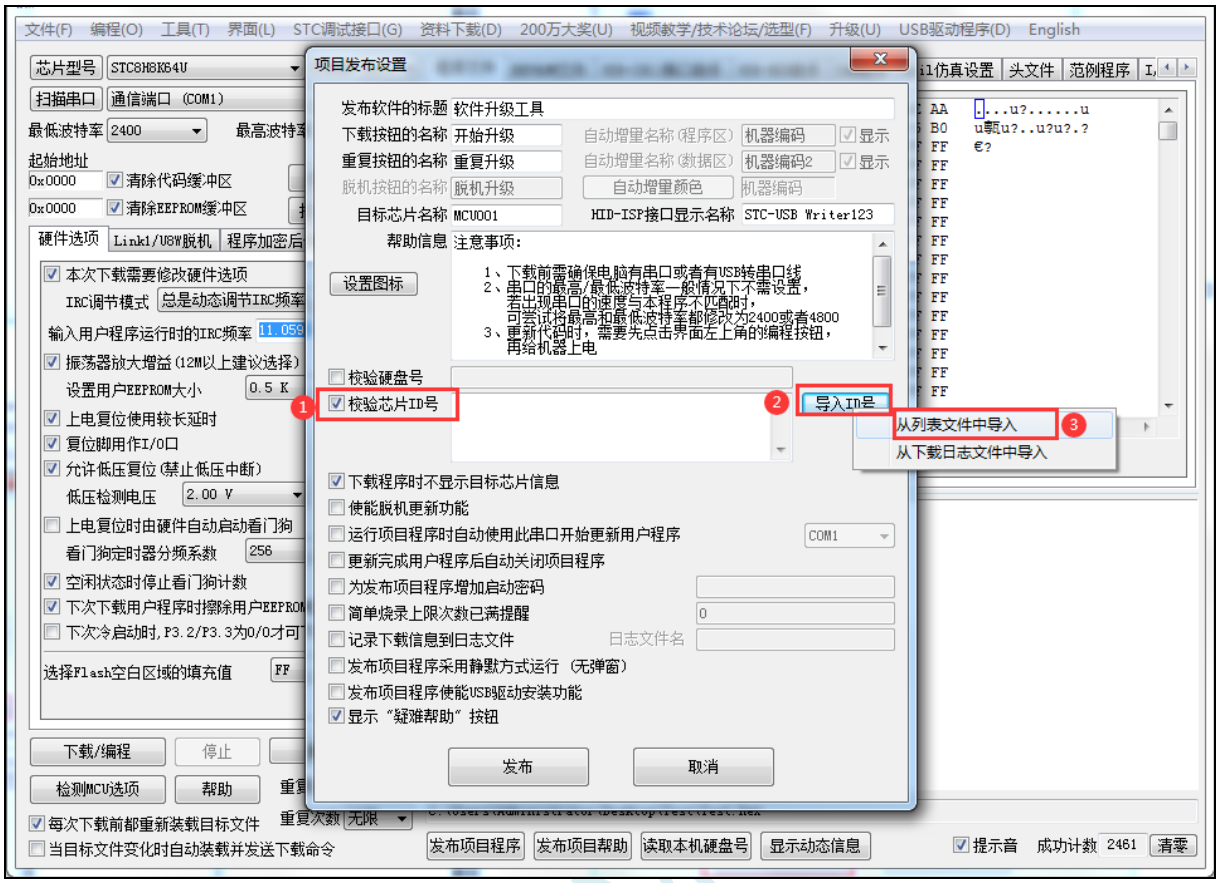
1、从日志文件存放目录中打开目标日期的日志文件（例如打开 2023 年 8 月 22 日的日志，则打开日志文件存放目录中的“20230822.log”）。日志记录格式如下图：



2、从日志文件中复制 ID 号到一个列表文件中，如下图



第三步、发布项目程序时导入列表文件中的 ID（如果需要从日志中自动导入，可跳到第四步）



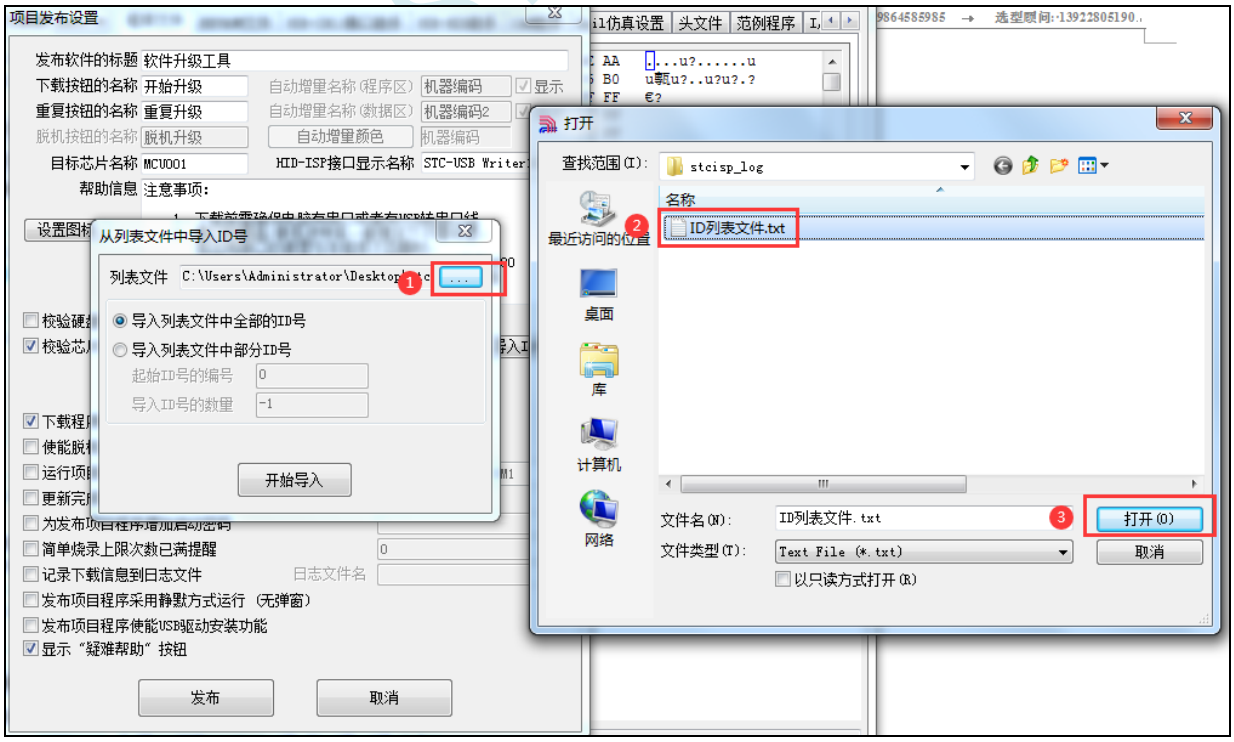
1、点击 AIapp-ISP 下载界面中的“发布项目程序”按钮

2、勾选“校验芯片 ID 号”

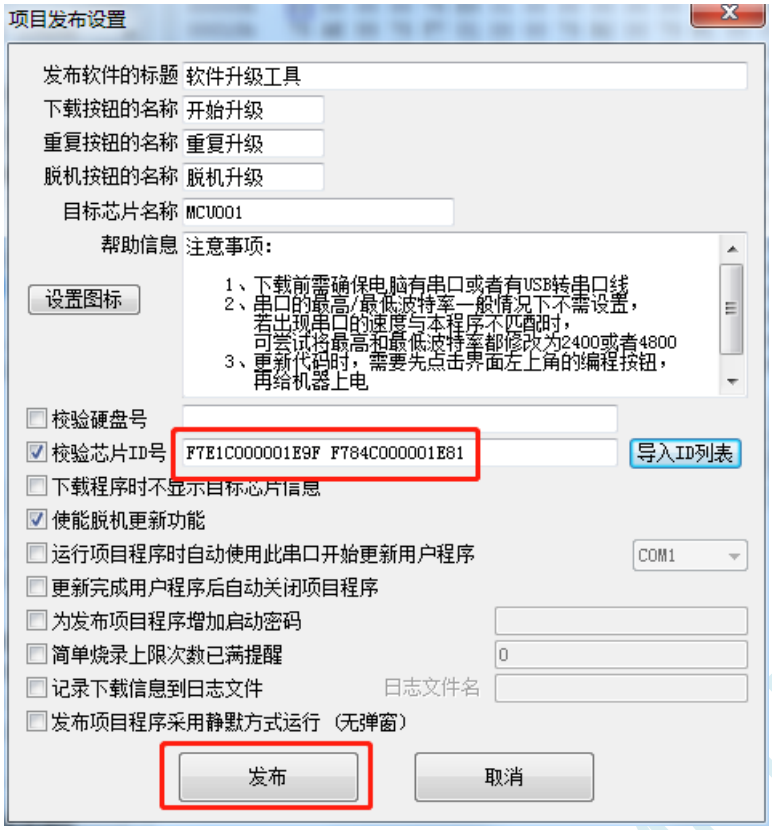
3、点击“导入 ID 号”

4、选择“从列表文件中导入”

5、打开上一步导出的列表文件

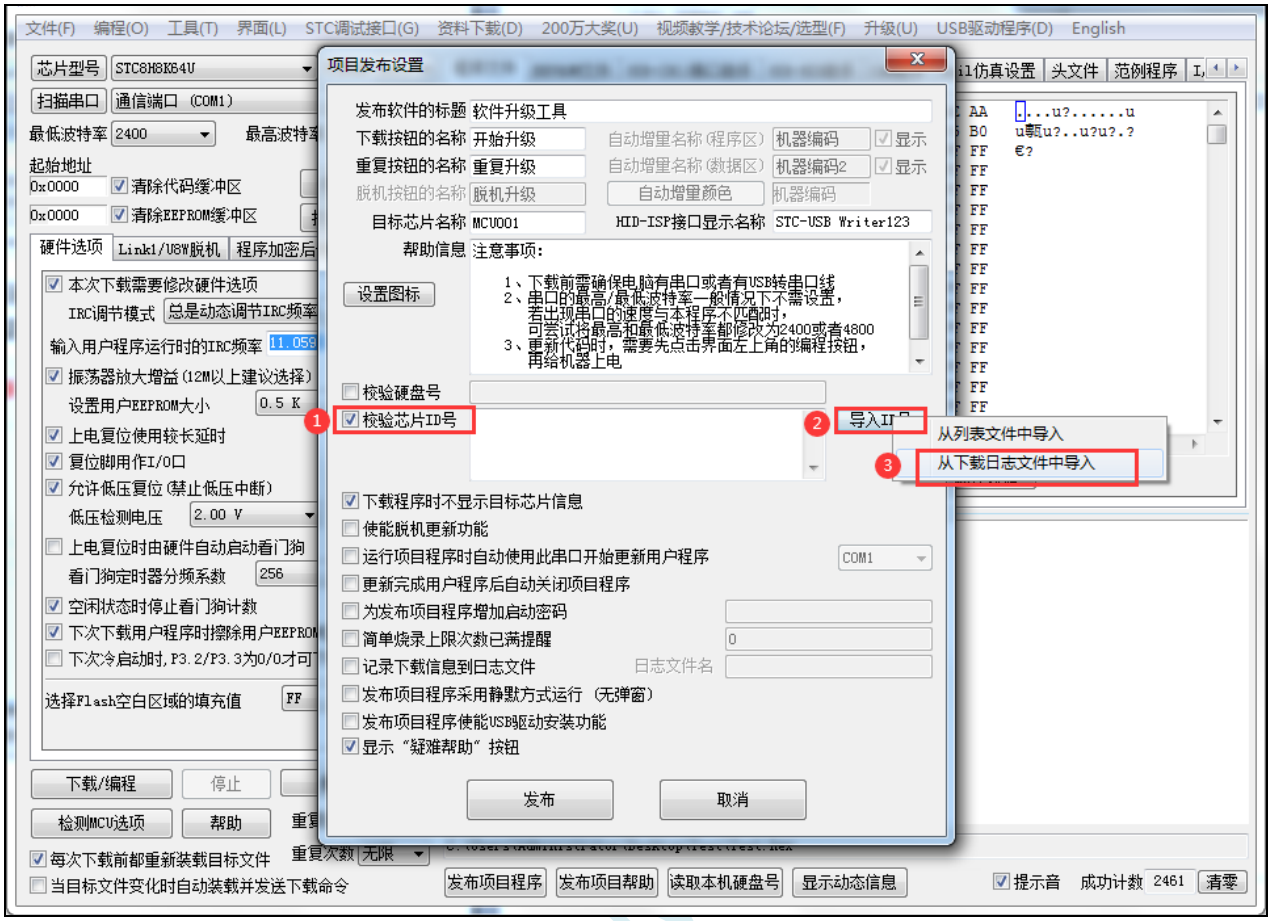


6、列表导入成功后，在下面的 ID 号文本框内会显示刚刚导入的全部 ID 号



7、最后点击“发布”按钮即可发布项目。

第四步、发布项目程序时从日志文件中自动导入 ID

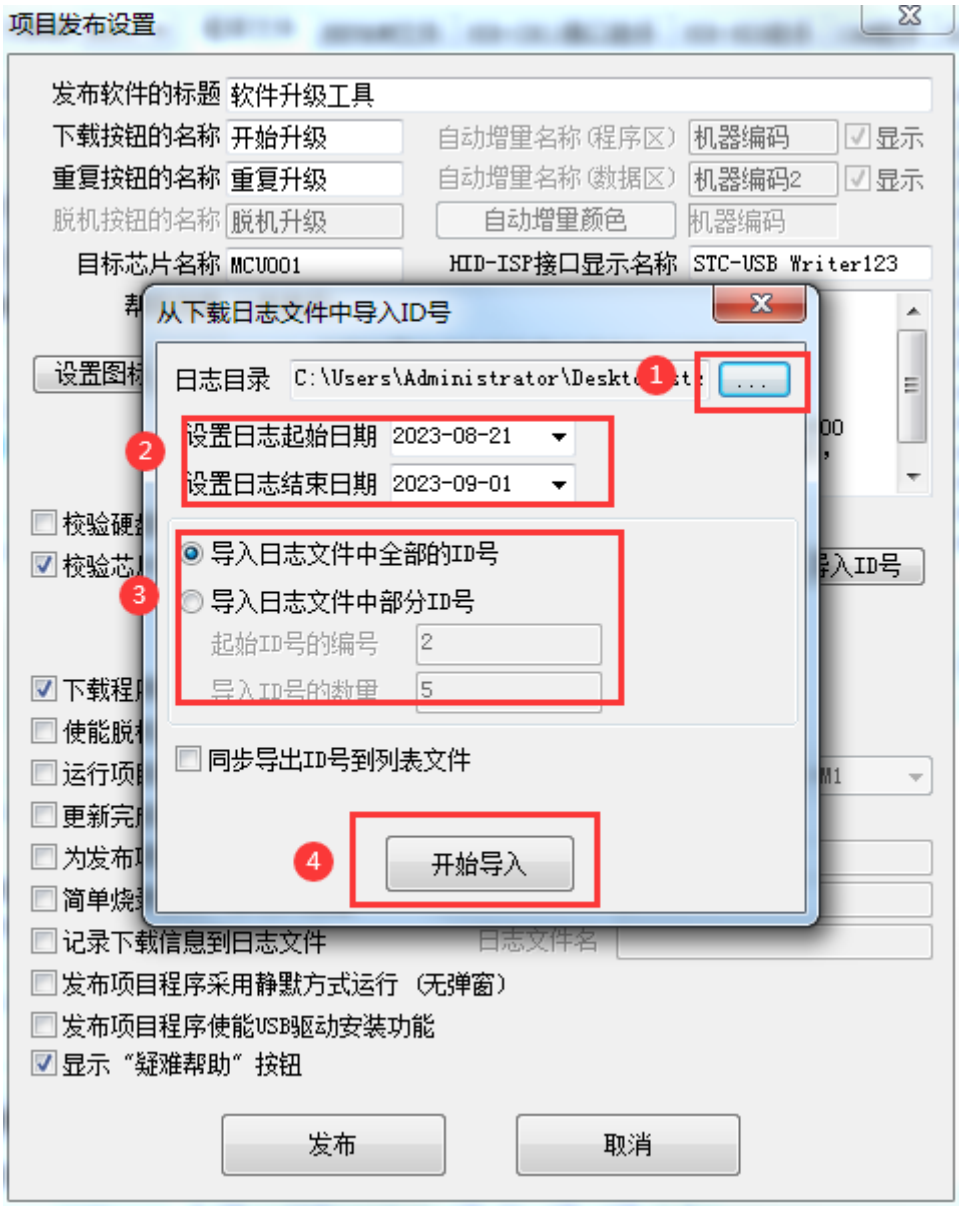


1、点击 AIapp-ISP 下载界面中的“发布项目程序”按钮

2、勾选“校验芯片 ID 号”

3、点击“导入 ID 号”

4、选择“从下载日志文件中导入”

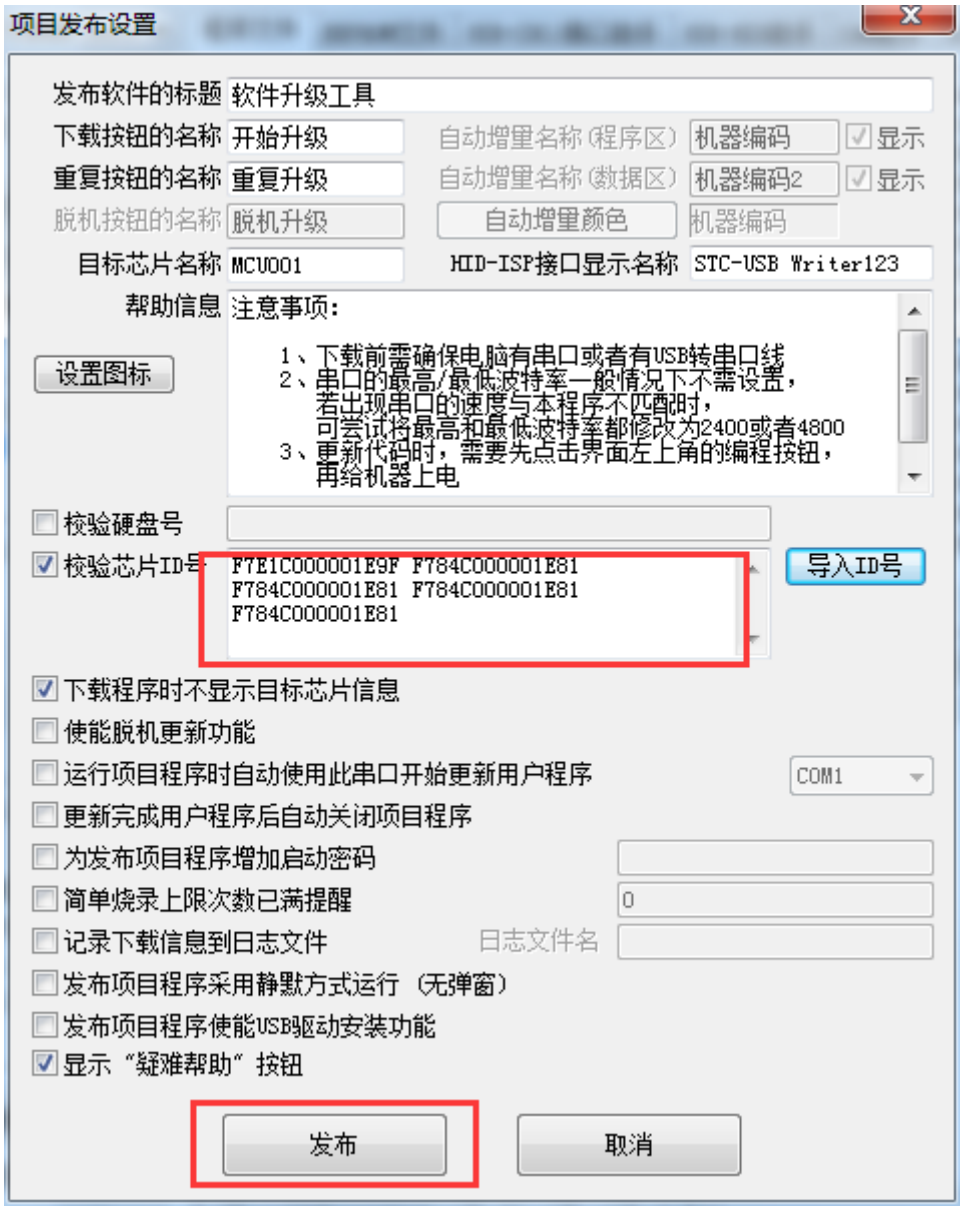


5、打开日志保存目录

6、设置需要导入日志的起始时间和结束时间

7、选择需要导入的 ID 号的序号

8、列表导入成功后，在下面的 ID 号文本框内会显示刚刚导入的全部 ID 号



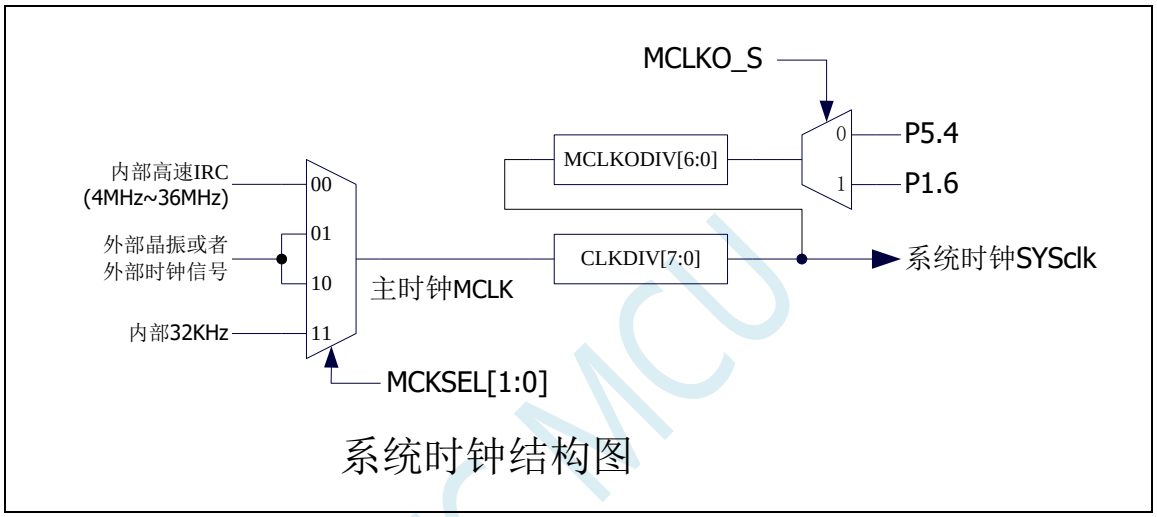
9、最后点击“发布”按钮即可发布项目。

6 时钟、复位、省电模式与系统电源管理，芯片上电工作过程

6.1 系统时钟控制

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

单片机进入掉电模式后，时钟控制器将会关闭所有的时钟源



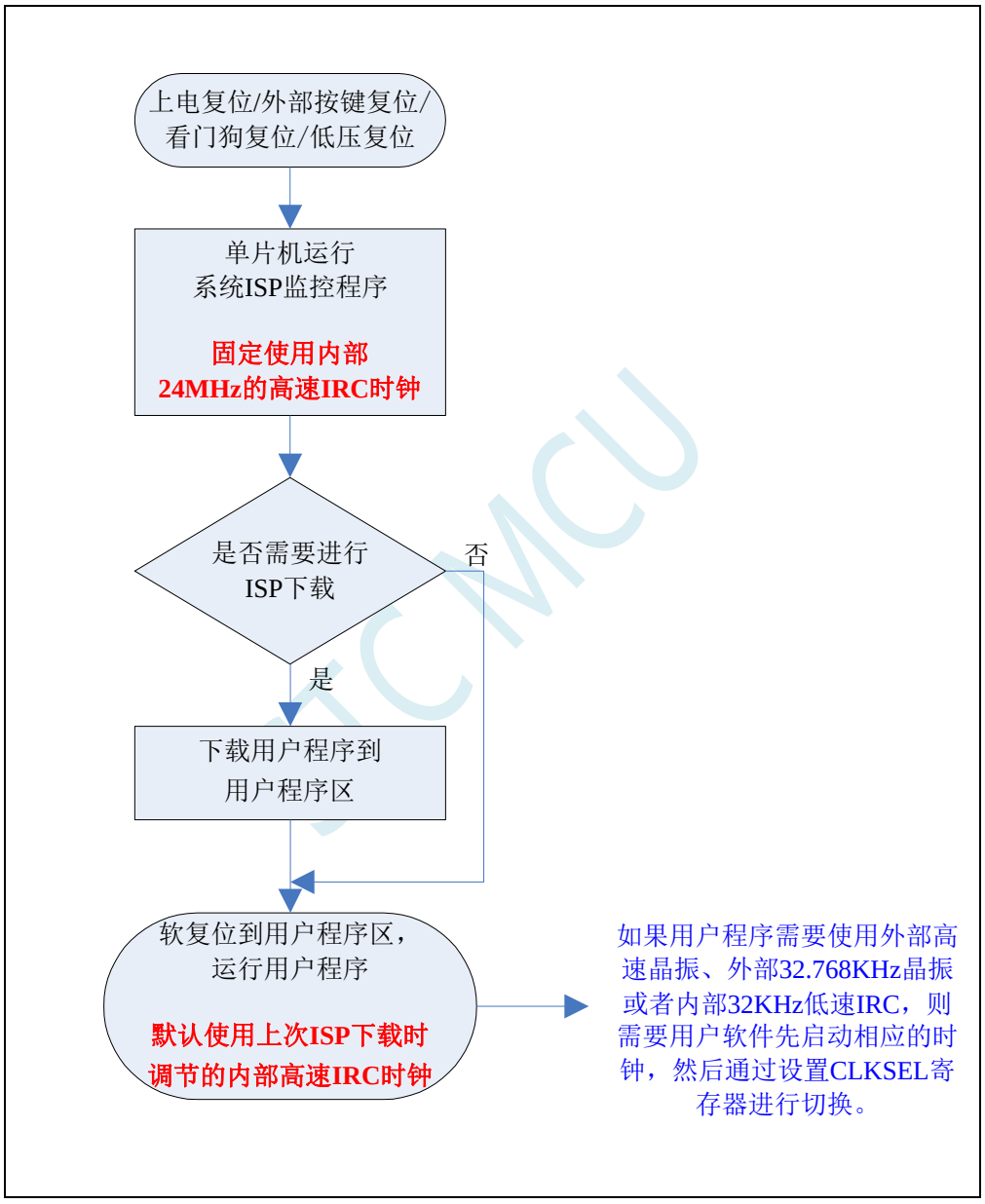
系统时钟结构图

STCMCU

6.2 芯片上电工作过程：

上电复位/复位脚复位/看门狗复位/低压检测复位时，芯片默认从 ISP 系统程序开始执行代码，此时固定使用内部 24MHz 的高速 IRC 时钟，当需要下载用户程序且下载完成后复位到用户程序区或者不需要下载直接复位到用户程序区时，默认会使用上次用户下载时所调节的高速 IRC 时钟，如果用户程序需要使用外部高速晶振、外部 32.768KHz 晶振或者内部 32KHz 低速 IRC，则需要用户软件先启动相应的时钟，然后通过设置 CLKSEL 寄存器进行切换。

启动流程如下：



6.3 相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CLKSEL	时钟选择寄存器	FE00H	-						MCKSEL[1:0]		xxxx,xx00
CLKDIV	时钟分频寄存器	FE01H									nnnn,nnnn
HIRCCR	内部高速振荡器控制寄存器	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	内部 32K 振荡器控制寄存器	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	主时钟输出控制寄存器	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
IRCDB	内部高速振荡器稳定时间控制	FE06H									1000,0000

STC MCU

6.3.1 系统时钟选择寄存器（CLKSEL）

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

MCKSEL[1:0]：主时钟源选择

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

6.3.2 时钟分频寄存器（CLKDIV）

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

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

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

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

6.3.3 内部高速高精度 IRC 控制寄存器（HIRCCR）

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

ENHIRC：内部高速高精度 IRC 使能位

0：关闭内部高精度 IRC

1：使能内部高精度 IRC

HIRCST：内部高精度 IRC 频率稳定标志位。（只读位）

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

STC MCU

6.3.4 外部振荡器控制寄存器（XOSCCR）

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

ENXOSC：外部晶体振荡器使能位

0：关闭外部晶体振荡器

1：使能外部晶体振荡器

XITYPE：外部时钟源类型

0：外部时钟源是外部时钟信号（或有源晶振）。信号源只需连接单片机的 XTALI（P1.7）**（此时 P1.6 口固定为高阻输入模式，可用于读取外部数字信号或当作 ADC 输入，但一般不建议使用，因为旁边的 P1.7 口有高频振荡信号会对 P1.6 的信号有影响）**

1：外部时钟源是晶体振荡器。信号源连接单片机的 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。

6.3.5 内部 32KHz 低速 IRC 控制寄存器（IRC32KCR）

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

ENIRC32K：内部 32K 低速 IRC 使能位

0：关闭内部 32K 低速 IRC

1：使能内部 32K 低速 IRC

IRC32KST：内部 32K 低速 IRC 频率稳定标志位。（只读位）

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

6.3.6 主时钟输出控制寄存器（MCLKOCR）

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

MCLKODIV[6:0]：主时钟输出分频系数

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

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

MCLKO_S：系统时钟输出管脚选择

0：系统时钟分频输出到 P5.4 口

1：系统时钟分频输出到 P1.6 口

STC MCU

6.3.7 内部高速 IRC 时钟恢复振荡到稳定需要等待的时钟数控制寄存器（IRCDB）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRCDB	FE06H								

IRCDB[7:0]：内部高速 IRC 时钟从主时钟停振/省电模式（掉电模式）恢复振荡到稳定需要等待的时钟数控制寄存器。**上电复位后初始值为 80H。**

IRCDB	系统时钟数
0	255 个时钟
1	0 个时钟
2	1 个时钟
3	2 个时钟
…	…
128	**127 个时钟（上电初始值）**
…	…
255	254 个时钟

【寄存器说明】:

1、芯片使用内部高速 IRC 时钟作为系统时钟时，当芯片从主时钟停振/省电模式（掉电模式）被唤醒后，系统会等待 IRCDB 寄存器设置的等待时钟数，再将时钟提供给 CPU 和系统并开始继续运行用户程序。建议在主时钟停振语句的下一句后面增加 7~9 个 NOP。

2、需要设置等待多少个时钟再给 CPU 供应时钟，让 CPU 开始跑程序，根据用户实际使用的工作频率来确定。（目前测试部分芯片运行在 33MHz 附近时，需要将 IRCDB 设置为 0x10 时最稳定）

3、如果用户的工作频率较高时，强烈建议：用户程序在进入主时钟停振/省电模式前将内部高速 IRC 频率调整到 24MHz，再进入主时钟停振/省电模式。等芯片唤醒后再将内部高速 IRC 调整到用户需要的工作频率。内部高速 IRC 时钟，出厂时有多种校准值，用户程序可以任意选择预置的校准时钟频率。

6.4 STC8C 系列内部 IRC 频率调整

STC8C 系列单片机内部均集成有一颗高精度内部 IRC 振荡器。在用户使用 ISP 下载软件进行下载时，ISP 下载软件会根据用户所选择/设置的频率自动进行调整，一般频率值可调整到±0.3%以下，调整后的频率在全温度范围内（－40℃～85℃）的温漂可达-1.35%～1.30%。

STC8C 系列内部 IRC 有两个频段，频段的中心频率分别为 20MHz 和 33MHz，20M 频段的调节范围约为 14.7MHz～26MHz，33M 频段的调节范围约为 24.5MHz～42.2MHz（注意：不同的芯片以及不同的生成批次可能会有约 5%左右的制造误差）。经实际测试，部分芯片的最高工作频率只能为 39MHz，所以为了安全起见，建议用户在 ISP 下载时设置 IRC 频率不要高于 35MHz。

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

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

内部 IRC 频率调整主要使用下面的 4 个寄存器进行调整

相关寄存器

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

6.4.1 IRC 频段选择寄存器（IRCBAND）

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

SEL：频段选择

0：选择 20MHz 频段

1：选择 33MHz 频段

6.4.2 内部 IRC 频率调整寄存器（IRTRIM）

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

IRTRIM[7:0]：内部高精度 IRC 频率调整寄存器

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

6.4.3 内部 IRC 频率微调寄存器（LIRTRIM）

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

LIRTRIM[1:0]：内部高精度 IRC 频率微调寄存器

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

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

6.4.4 时钟分频寄存器（CLKDIV）

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

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

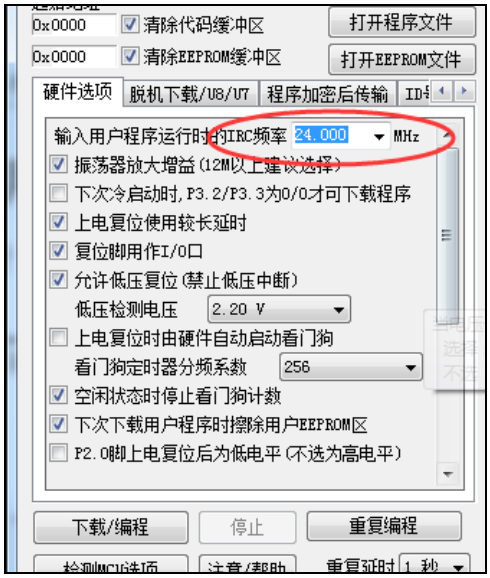
CLKDIV	系统时钟频率
0	MCLK/1
1	MCLK/1
2	MCLK/2
3	MCLK/3
…	…
x	MCLK/x
…	…
255	MCLK/255

STC8C 系列内部的两个频段的可调范围分别为 14.7MHz～26MHz 和 24.5MHz～42.2MHz。虽然 33MHz 频段的上限可调到 40MHz 以上，但芯片内部的 Flash 程序存储器无法运行到 40MHz 以上的速度，所以用户在 ISP 下载时设置内部 IRC 频率不能高于 40MHz，一般建议用户设置为 35MHz 以下。若用户需要较低的工作频率时，可使用 CLKDIV 寄存器对调节后的频率进行分频，例如用户需要 11.0592MHz 的频率，使用内部 IRC 直接调整是无法得到这个频率的，但可将内部 IRC 调整到 22.1184MHz，在使用 CLKDIV 进行 2 分频即可得到 11.0592MHz。

6.4.5　分频出 3MHz 用户工作频率，并用户动态改变频率追频示例

为得到 3MHz 的频率，可使用 24MHz÷8 的方法。

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



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

C 语言代码

```
//测试工作频率为24MHz

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

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

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

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

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

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

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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 24MHz

P_SW2       DATA    0BAH
IRTRIM      DATA    09FH

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

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
```

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

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

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

        JMP     $

        END
```

STC MCU

6.5 系统复位

STC8C 系列单片机的复位分为硬件复位和软件复位两种。

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

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

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

- 写 IAP_CONTR 的 SWRST 所触发的复位

相关寄存器

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

STC MCU

6.5.1 看门狗复位（WDT_CONTR）

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

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

WDT_CONTR（看门狗控制寄存器）

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

WDT_FLAG：看门狗溢出标志

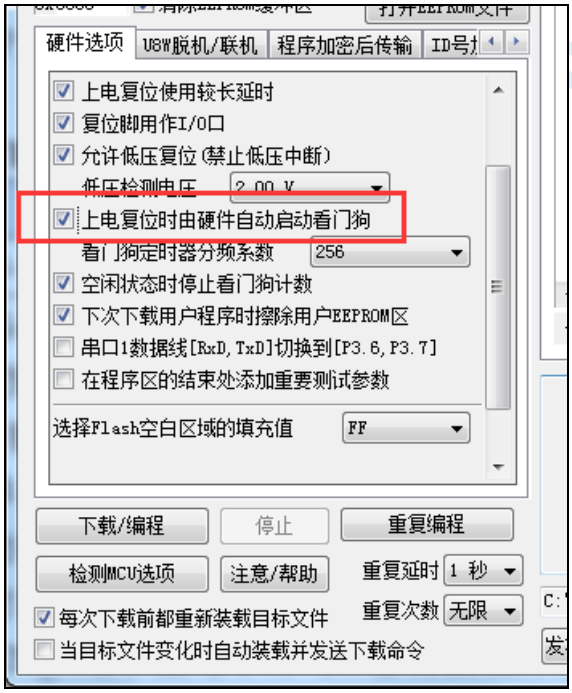
看门狗发生溢出时，硬件自动将此位置 1，需要软件清零。

EN_WDT：看门狗使能位

0：对单片机无影响

1：启动看门狗定时器。

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



CLR_WDT：看门狗定时器清零

0：对单片机无影响

1：清零看门狗定时器，硬件自动将此位复位

IDL_WDT：IDLE 模式时的看门狗控制位

0：IDLE 模式时看门狗停止计数

1：IDLE 模式时看门狗继续计数

WDT_PS[2:0]：看门狗定时器时钟分频系数

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

看门狗溢出时间计算公式如下：

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

STC MCU

6.5.2 软件复位（IAP_CONTR）

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

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

SWBS：软件复位启动选择

0：软件复位后从用户程序区开始执行代码。用户数据区的数据保持不变。

1：软件复位后从系统 ISP 区开始执行代码。用户数据区的数据会被初始化。

SWRST：软件复位触发位

0：对单片机无影响

1：触发软件复位

6.5.3 低压复位（RSTCFG）

RSTCFG（复位配置寄存器）

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

ENLVR：低压复位控制位

0：禁止低压复位。当系统检测到低压事件时，会产生低压中断

1：使能低压复位。当系统检测到低压事件时，自动复位

P54RST：RST 管脚功能选择

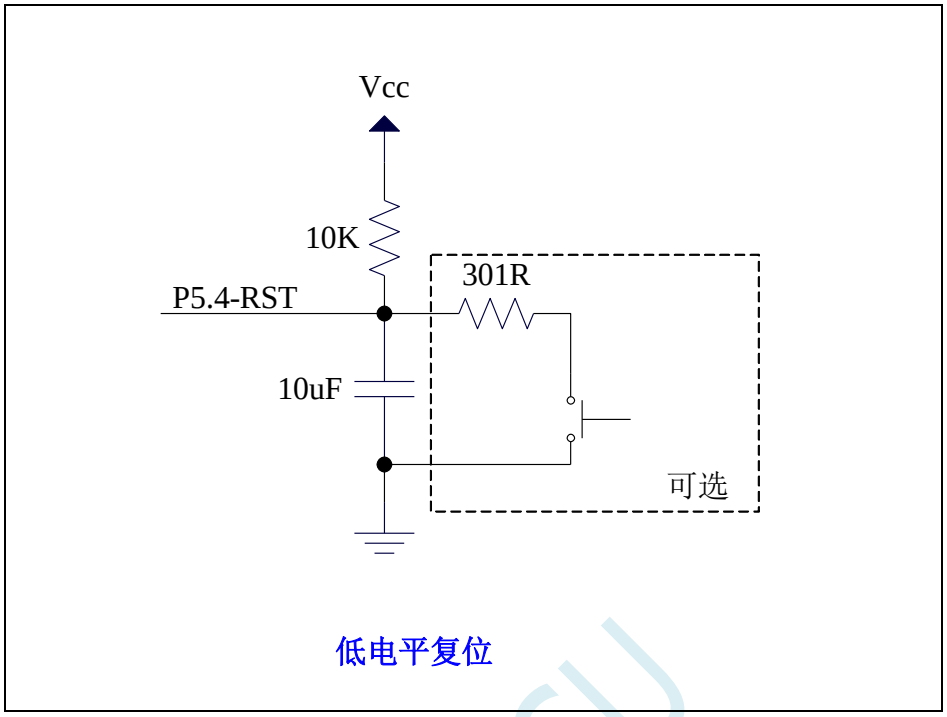
0：RST 管脚用作普通 I/O 口（P5.4）

1：RST 管脚用作复位脚（低电平复位）

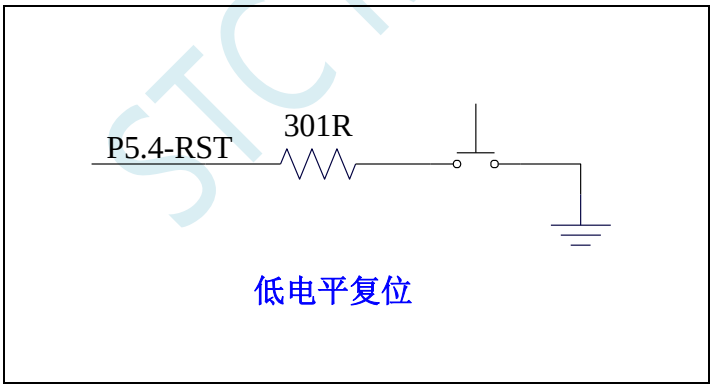
LVDS[1:0]：低压检测门槛电压设置

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

6.5.4 低电平上电复位参考电路（一般不需要）

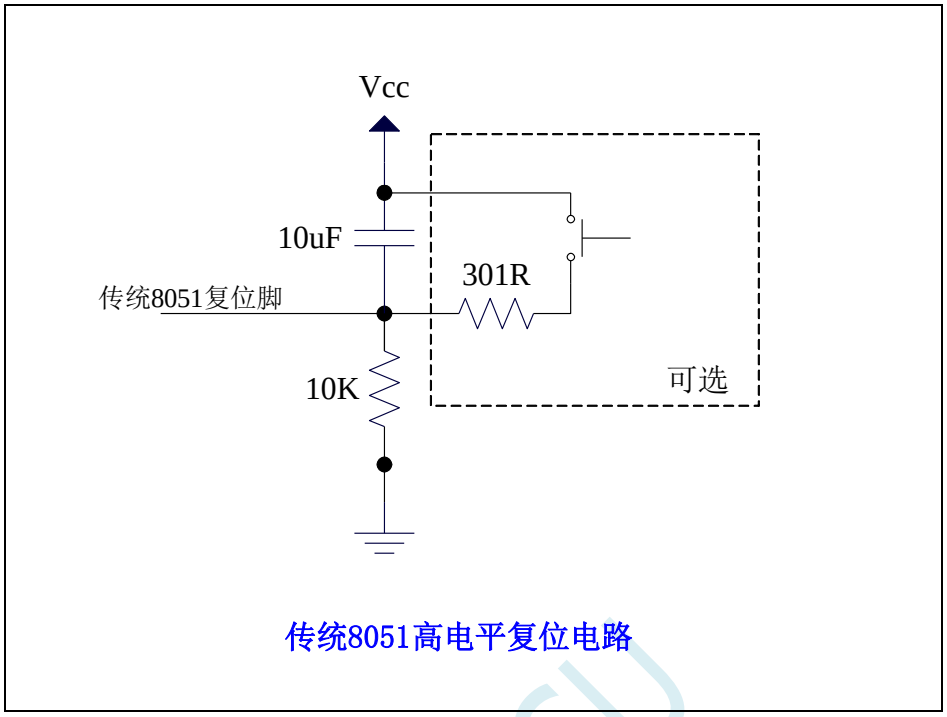


6.5.5 低电平按键手动复位参考电路



STC MCU

6.5.6 传统 8051 高电平上电复位参考电路



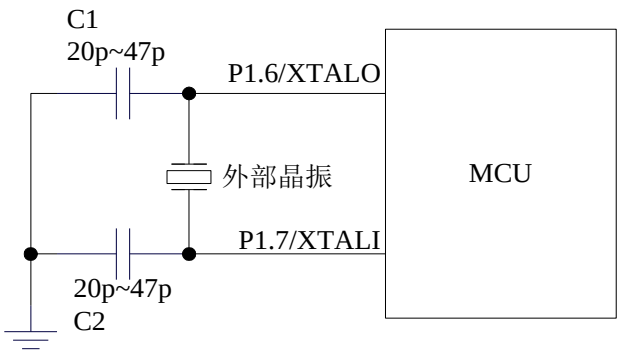
传统8051高电平复位电路

上图为传统 8051 的高电平复位电路，STC8C 的复位为低电平复位，与传统复位电路不同

STC MCU

6.6 外部晶振及外部时钟电路

6.6.1 外部晶振输入电路



注意：

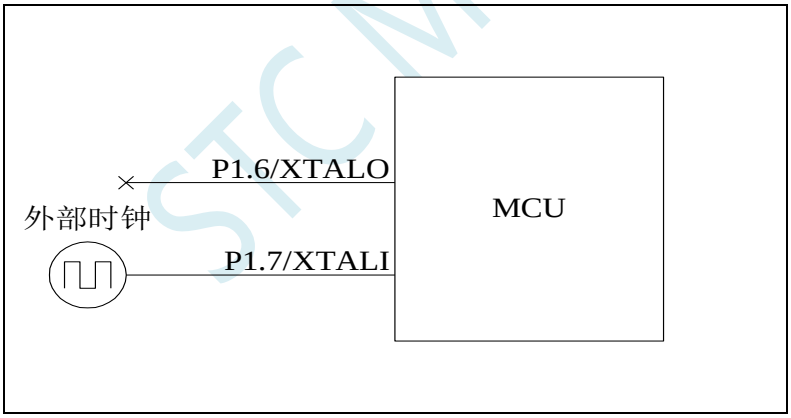
1、C1,C2两个电容一定不能省，若没有这两个电容，晶振有可能不起振

2、RTC时钟必须使用外部32768晶振，外挂电容要<=20pF，例如20p/15p/10p

3、若外部连接高速晶振（例如：24MHz），外挂电容47pF为宜

4、外部晶振会增加额外的系统功耗，32768晶振增加1.5uA的功耗，
高速晶振（例如：24MHz）会增加2mA以上的功耗

6.6.2 外部时钟输入电路（P1.6 为高阻输入模式，可当输入口使用）



注：当使用内部时钟时，P1.6/P1.7 都可以当普通 I/O 使用。当 P1.7 口外接有源时钟或外接其他时钟源时，P1.6 口为高阻输入模式，可当输入口使用，此时 P1.6 的端口模式不可改变。有 RTC 功能的 MCU，外部 32768 时钟可从 P1.7 口输入。

6.7 时钟停振/省电模式与系统电源管理

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

6.7.1 电源控制寄存器（PCON）

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

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

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

PD：时钟停振模式/掉电模式/停电模式控制位

0：无影响

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

IDL：IDLE（空闲）模式控制位

0：无影响

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

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

6.7.2 内部高速 IRC 时钟恢复振荡到稳定需要等待的时钟数控制寄存器（IRCDB）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IRCDB	FE06H								

IRCDB[7:0]：内部高速 IRC 时钟从主时钟停振/省电模式（掉电模式）恢复振荡到稳定需要等待的时钟数控制寄存器。**上电复位后初始值为 80H。**

IRCDB	系统时钟数
0	255 个时钟
1	0 个时钟
2	1 个时钟
3	2 个时钟
…	…
128	**127 个时钟（上电初始值）**
…	…
255	254 个时钟

【寄存器说明】:

1、芯片使用内部高速 IRC 时钟作为系统时钟时，当芯片从主时钟停振/省电模式（掉电模式）被唤醒后，系统会等待 IRCDB 寄存器设置的等待时钟数，再将时钟提供给 CPU 和系统并开始继续运行用户程序。建议在主时钟停振语句的下一句后面增加 7~9 个 NOP。

2、需要设置等待多少个时钟再给 CPU 供应时钟，让 CPU 开始跑程序，根据用户实际使用的工作频率来确定。（目前测试部分芯片运行在 33MHz 附近时，需要将 IRCDB 设置为 0x10 时最稳定）

3、如果用户的工作频率较高时，强烈建议：用户程序在进入主时钟停振/省电模式前将内部高速 IRC 频率调整到 24MHz，再进入主时钟停振/省电模式。等芯片唤醒后再将内部高速 IRC 调整到用户需要的工作频率。内部高速 IRC 时钟，出厂时有多种校准值，用户程序可以任意选择预置的校准时钟频率。

6.8 掉电唤醒定时器

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

6.8.1 掉电唤醒定时器计数寄存器（WKTCL，WKTCH）

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

WKTEN：掉电唤醒定时器的使能控制位

0：停用掉电唤醒定时器

1：启用掉电唤醒定时器

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

这里请注意：用户在寄存器{WKTCH[6:0],WKTCL[7:0]}中写入的值必须比实际计数值少 1。如用户需计数 10 次，则将 9 写入寄存器{WKTCH[6:0],WKTCL[7:0]}中。同样，如果用户需计数 32767 次，则应对{WKTCH[6:0],WKTCL[7:0]}写入 7FFEH（即 32766）。（**计数值 0 和计数值 32767 为内部保留值，用户不能使用**）。内部掉电唤醒定时器有自己的内部时钟，掉电唤醒定时器计数一次的时间就是由该时钟决定的。内部掉电唤醒定时器的时钟频率约为 32KHz，误差较大。用户可以通过读 RAM 区 F8H 和 F9H 的内容（F8H 存放频率的高字节，F9H 存放低字节）来获取内部掉电唤醒专用定时器出厂时所记录的时钟频率。

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

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

假设 F_{wt}=32KHz，则有：

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

6.9 范例程序

6.9.1 选择系统时钟源

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr       P_SW2       =   0xba;

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

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

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

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

```
    P_SW2 = 0x00;
*/

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

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

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

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

            ORG     0000H
            LJMP    MAIN

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

            MOV     P_SW2,#80H
```

STC MCU

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

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

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

        JMP     $

        END
```

6.9.2　主时钟分频输出

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr     P_SW2     =   0xba;

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

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

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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

MCLKOCR     EQU     0FE05H

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
```

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

        MOV     P_SW2,#80H
;       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
```

6.9.3 看门狗定时器应用

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

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

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

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

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

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

汇编代码

```
;测试工作频率为 11.0592MHz

WDT_CONTR   DATA    0C1H

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
```

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

;               MOV     WDT_CONTR,#23H      ;使能看门狗,溢出时间约为0.5s
                MOV     WDT_CONTR,#24H      ;使能看门狗,溢出时间约为1s
;               MOV     WDT_CONTR,#27H      ;使能看门狗,溢出时间约为8s
                CLR     P3.2                ;测试端口
LOOP:
;               MOV     WDT_CONTR,#33H      ;清看门狗,否则系统复位
                MOV     WDT_CONTR,#34H      ;清看门狗,否则系统复位
;               MOV     WDT_CONTR,#37H      ;清看门狗,否则系统复位

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

                END
```

6.9.4　软复位实现自定义下载

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

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

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

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

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

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

汇编代码

```
;测试工作频率为 11.0592MHz

IAP_CONTR   DATA    0C7H

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

            ORG     0000H
            LJMP    MAIN

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

            SETB    P3.2
            SETB    P3.3
```

STC MCU

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

        END
```

6.9.5 低压检测

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

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

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

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

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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

RSTCFG      DATA    0FFH
ENLVR       EQU     40H             ;RSTCFG.6
LVD2V0      EQU     00H             ;LVD@2.0V
LVD2V4      EQU     01H             ;LVD@2.4V
LVD2V7      EQU     02H             ;LVD@2.7V
LVD3V0      EQU     03H             ;LVD@3.0V

ELVD        BIT     IE.6
LVDF        EQU     20H             ;PCON.5

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

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF  ;清中断标志
            CPL     P3.2            ;测试端口
            RETI
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
```

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

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

        END
```

6.9.6 省电模式

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

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

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

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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     0003H
            LJMP    INT0ISR

            ORG     0100H
INT0ISR:
            CPL     P3.4                ;测试端口
            RETI
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
```

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

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

                END
```

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

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

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

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

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

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

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

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

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

    IT0 = 0;                                    //使能 INT0 上升沿和下降沿中断
//  IT0 = 1;                                    //使能 INT0 下降沿中断
    EX0 = 1;                                    //使能 INT0 中断

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

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

    EA = 1;

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

    _nop_();
    _nop_();
```

```
    _nop_();

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

汇编代码

```
;测试工作频率为 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN

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

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

```
        RETI

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

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

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

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

        SETB    EA

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

        END
```

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

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
```

```
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T4T3M   =   0xd1;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2         0x04
#define ET3         0x20
#define ET4         0x40
sfr     AUXINTIF=   0xef;
#define T2IF        0x01
#define T3IF        0x02
#define T4IF        0x04

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

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

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

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

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

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

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

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

STC MCU

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

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

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

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

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

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

    EA = 1;

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

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

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

汇编代码

```
;测试工作频率为 11.0592MHz

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

```
T4L         DATA    0D3H
T4H         DATA    0D2H
T4T3M       DATA    0D1H
AUXR        DATA    8EH

IE2         DATA    0AFH
ET2         EQU     04H
ET3         EQU     20H
ET4         EQU     40H

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

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

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

            ORG     0100H
TM0ISR:
            CPL     P1.0            ;测试端口
            RETI
TM1ISR:
            CPL     P1.0            ;测试端口
            RETI
TM2ISR:
            CPL     P1.0            ;测试端口
            RETI
TM3ISR:
            CPL     P1.0            ;测试端口
            RETI
TM4ISR:
            CPL     P1.0            ;测试端口
            RETI

MAIN:
```

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

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

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

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

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

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

        SETB    EA

        MOV     PCON,#02H           ;MCU 进入掉电模式
        NOP                         ;T0/T1/T2/T3/T4 的外部管脚唤醒后，
                                    ;不进入中断服务程序，只是继续执行程序，
                                    ;与 INT0/INT1/INT2/INT3/INT4 的掉电唤醒不一样
        NOP                         ;建议多加几个 NOP 指令，如 3 个以上
        NOP
        NOP
LOOP:
        CPL     P1.1
        JMP     LOOP

        END
```

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

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

sbit    P11     =   P1^1;

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

void UART1_Isr() interrupt 4
{
}

void UART2_Isr() interrupt 8
{
}

void UART3_Isr() interrupt 17
{
}

void UART4_Isr() interrupt 18
{
}

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

STC MCU

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

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

    P_SW2 = 0x00;                   //RXD2/P1.0 下降沿唤醒
//  P_SW2 = 0x01;                   //RXD2_2/P4.6 下降沿唤醒

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

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

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

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

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

汇编代码

```
;测试工作频率为 11.0592MHz

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

P_SW1       DATA    0A2H
P_SW2       DATA    0BAH

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

```
P5M1        DATA    0C9H
P5M0        DATA    0CAH

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

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

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

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

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

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

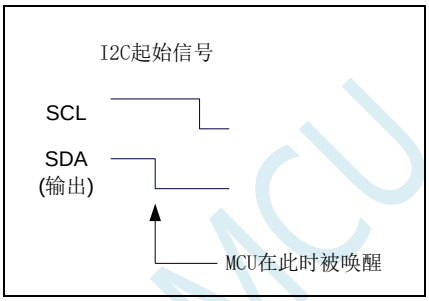
            MOV     P_SW2,#00H          ;RXD4/P0.2 下降沿唤醒
;           MOV     P_SW2,#04H          ;RXD4_2/P5.2 下降沿唤醒

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

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

        END
```

6.9.10　使用 I2C 的 SDA 脚唤醒 MCU 省电模式



C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     P_SW2       =   0xba;

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

sbit    P11         =   P1^1;

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

void i2c_isr() interrupt 24
```

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

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

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

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

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

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    DPH
            PUSH    DPL
            ORL     PSW2,#80H
            MOV     DPTR,#I2CSLST
            MOVX    A,@DPTR
            ANL     A,#NOT 40H
            MOVX    @DPTR,A
            POP     DPL
            POP     DPH
            POP     ACC
            RETI

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

            MOV     P_SW2,#00H          ;SDA/P1.4 下降沿唤醒
//          MOV     P_SW2,#10H          ;SDA_2/P2.4 下降沿唤醒
//          MOV     P_SW2,#30H          ;SDA_4/P3.3 下降沿唤醒
            ORL     P_SW2,#80H
            MOV     DPTR,#I2CCFG
            MOV     A,#80H
            MOVX    @DPTR,A             ;使能 I2C 模块的从机模式
            MOV     DPTR,# I2CSLCR
            MOV     A,#40H              ;使能起始信号中断
            SETB    EA

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

```
END
```

6.9.11　使用掉电唤醒定时器唤醒省电模式

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

sbit        P11         =   P1^1;

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

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

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

```
        P11 = ~P11;
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

WKTCL       DATA        0AAH
WKTCH       DATA        0ABH

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

            ORG         0000H
            LJMP        MAIN

            ORG         0100H

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

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

LOOP:
            NOP
            NOP
            MOV         PCON,#02H           ;MCU 进入掉电模式
            NOP
            NOP
            NOP
            NOP
            CPL         P1.1
            JMP         LOOP

            END
```

6.9.12　LVD 中断唤醒省电模式，建议配合使用掉电唤醒定时器

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

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

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

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

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

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

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

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

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

汇编代码

```
;测试工作频率为 11.0592MHz

RSTCFG      DATA    0FFH
ENLVR       EQU     40H             ;RSTCFG.6
LVD2V0      EQU     00H             ;LVD@2.0V
LVD2V4      EQU     01H             ;LVD@2.4V
LVD2V7      EQU     02H             ;LVD@2.7V
LVD3V0      EQU     03H             ;LVD@3.0V

ELVD        BIT     IE.6
LVDF        EQU     20H             ;PCON.5

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

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
```

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

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

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

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

        END
```

6.9.13　比较器中断唤醒省电模式，建议配合使用掉电唤醒定时器

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

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

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

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

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

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

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

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

汇编代码

```
;测试工作频率为 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

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

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR

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

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

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

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

LOOP:
```

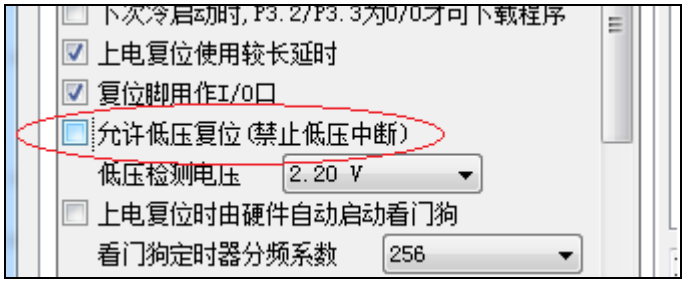
```
        CPL     P1.1
        JMP     LOOP

        END
```

6.9.14 使用 LVD 功能检测工作电压（电池电压）

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

（建议使用 ADC 的第 15 通道检测电池电压，见 ADC 章节）



C 语言代码

```
//测试工作频率为11.0592MHz

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

#define FOSC            11059200UL
#define T1MS            (65536 - FOSC/4/100)

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

#define LVDF            0x20                    //PCON.5

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

void delay()
{
    int i;
```

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

void main()
{
    unsigned char power;

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

    PCON &= ~LVDF;
    RSTCFG = LVD3V0;

    while (1)
    {
        power = 0x0f;

        RSTCFG = LVD3V0;
        delay();
        PCON &= ~LVDF;
        delay();
        if (PCON & LVDF)
        {
            power >>= 1;
            RSTCFG = LVD2V7;
            delay();
            PCON &= ~LVDF;
            delay();
            if (PCON & LVDF)
            {
                power >>= 1;
                RSTCFG = LVD2V4;
                delay();
                PCON &= ~LVDF;
                delay();
                if (PCON & LVDF)
                {
                    power >>= 1;
                    RSTCFG = LVD2V2;
                    delay();
                    PCON &= ~LVDF;
                    delay();
                    if (PCON & LVDF)
```

STC MCU

```
                    {
                        power >>= 1;
                    }
                }
            }
        }
        RSTCFG = LVD3V0;
        P2 = ~power;    //P2.3~P2.0 显示电池电量
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

RSTCFG      DATA    0FFH
LVD2V0      EQU     00H             ;LVD@2.0V
LVD2V4      EQU     01H             ;LVD@2.4V
LVD2V7      EQU     02H             ;LVD@2.7V
LVD3V0      EQU     03H             ;LVD@3.0V

LVDF        EQU     20H             ;PCON.5

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

            ORG     0000H
            JMP     MAIN

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

            ANL     PCON,#NOT LVDF
            MOV     RSTCFG,#LVD3V0
LOOP:
```

STC MCU

```
        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
```

STC MCU

7 存储器

STC8C 系列单片机的程序存储器和数据存储器是各自独立编址的。由于没有提供访问外部程序存储器的总线，单片机的所有程序存储器都是片上 Flash 存储器，不能访问外部程序存储器。

STC8C 系列单片机内部集成了大容量的数据存储器。STC8C 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。

STC8C2K64S4-44Pin/32Pin 系列、STC8C2K64S2-44Pin/32Pin/20Pin 系列单片内部集成了 64K 字节的 Flash 程序存储器（ROM）。

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

由于相邻中断入口地址的间隔区间仅仅有 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。

STC8C 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

STC8C 系列单片机内部集成的 RAM 可用于存放程序执行的中间结果和过程数据。

单片机系列	内部直接访问 RAM（DATA）	内部间接访问 RAM（IDATA）	内部扩展 RAM（XDATA）
STC8C2K64S4-44Pin/32Pin 系列	128 字节	128 字节	2048 字节
STC8C2K64S2-44Pin/32Pin/20Pin 系列	128 字节	128 字节	2048 字节

7.2.1 内部 RAM

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

内部 RAM 的结构如下图所示：

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

7.2.2 程序状态寄存器（PSW）

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

CY：进/借位标志位。

AC：辅助进/借位标志位。

F0：用户标志位 0。

RS1，RS0：工作寄存器选择位

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

OV：溢出标志位。

F1：用户标志位 1。

P：奇偶校验标志位。

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

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

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

7.2.3 内部扩展 RAM，XRAM，XDATA

STC8C 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 语言中定义变量为 pdata 类型后，编译器会自动将变量分配在 XDATA 的 0000H~00FFH 区域，并使用 MOVX @Ri,A 和 MOVX A@Ri 进行访问。

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

7.2.4 辅助寄存器（AUXR）

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

EXTRAM：扩展 RAM 访问控制

0：访问内部扩展 RAM。

1：内部扩展 RAM 被禁用。

7.2.5　外部扩展 RAM，XRAM，XDATA

STC8C 系列封装管脚数为 40 及其以上的单片机具有扩展 64KB 外部数据存储器的能力。访问外部数据存储器期间，WR/RD/ALE 信号要有效。STC8C 系列单片机新增了一个控制外部 64K 字节数据总线速度的特殊功能寄存器 BUS_SPEED，说明如下：

7.2.6　总线速度控制寄存器（BUS_SPEED）

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

RW_S[1:0]：RD/WR 控制线选择位

00：P4.4 为 RD，P4.3 为 WR

01：P3.7 为 RD，P3.6 为 WR

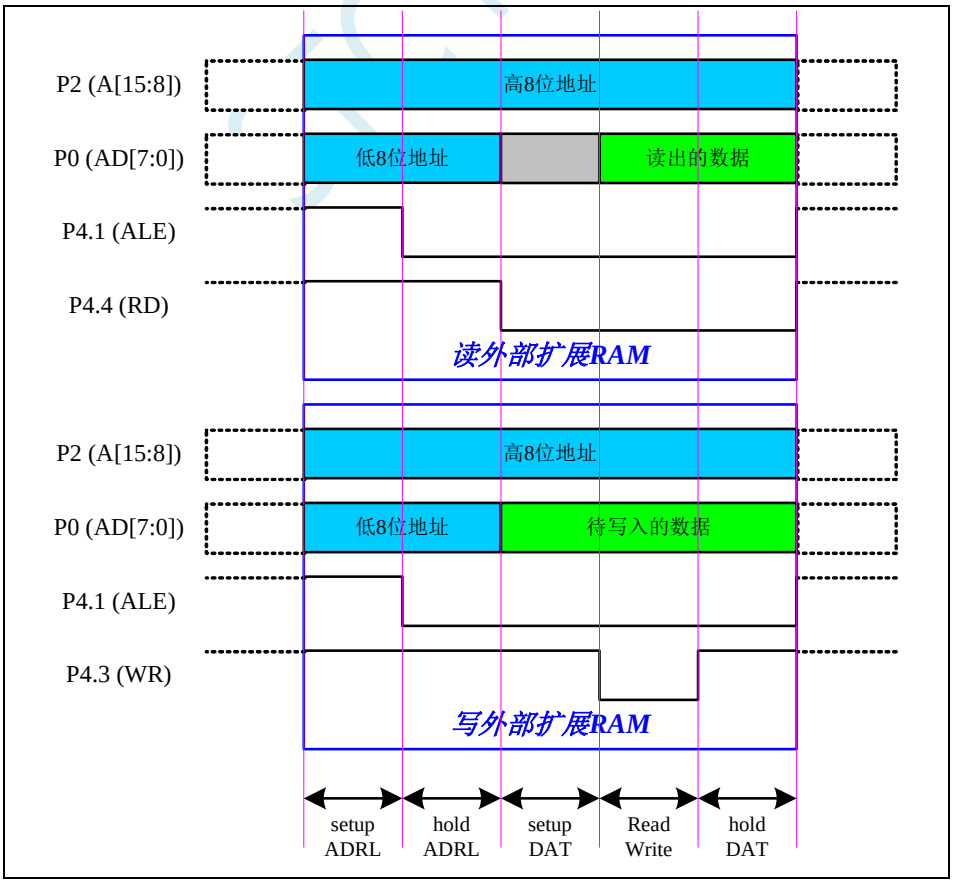
10：P4.2 为 RD，P4.0 为 WR

11：保留

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

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

读写外部扩展 RAM 时序如下图所示：



7.2.7 8051 中可位寻址的数据存储器

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

数据存储器地址	位寻址地址							
	B7	B6	B5	B4	B3	B2	B1	B0
F8H（P7）	FFH F8H.7	FEH F8H.6	FDH F8H.5	FCH F8H.4	FBH F8H.3	FAH F8H.2	F9H F8H.1	F8H F8H.0
F0H（B）	F7H F0H.7	F6H F0H.6	F5H F0H.5	F4H F0H.4	F3H F0H.3	F2H F0H.2	F1H F0H.1	F0H F0H.0
E8H（P6）	EFH E8H.7	EEH E8H.6	EDH E8H.5	ECH E8H.4	EBH E8H.3	EAH E8H.2	E9H E8H.1	E8H E8H.0
E0H（ACC）	E7H E0H.7	E6H E0H.6	E5H E0H.5	E4H E0H.4	E3H E0H.3	E2H E0H.2	E1H E0H.1	E0H E0H.0
D8H（CCON）	DFH D8H.7	DEH D8H.6	DDH D8H.5	DCH D8H.4	DBH D8H.3	DAH D8H.2	D9H D8H.1	D8H D8H.0
D0H（PSW）	D7H D0H.7	D6H D0H.6	D5H D0H.5	D4H D0H.4	D3H D0H.3	D2H D0H.2	D1H D0H.1	D0H D0H.0
C8H（P5）	CFH C8H.7	CEH C8H.6	CDH C8H.5	CCH C8H.4	CBH C8H.3	CAH C8H.2	C9H C8H.1	C8H C8H.0
C0H（P4）	C7H C0H.7	C6H C0H.6	C5H C0H.5	C4H C0H.4	C3H C0H.3	C2H C0H.2	C1H C0H.1	C0H C0H.0
B8H（IP）	BFH B8H.7	BEH B8H.6	BDH B8H.5	BCH B8H.4	BBH B8H.3	BAH B8H.2	B9H B8H.1	B8H B8H.0
B0H（P3）	B7H B0H.7	B6H B0H.6	B5H B0H.5	B4H B0H.4	B3H B0H.3	B2H B0H.2	B1H B0H.1	B0H B0H.0
A8H（IE）	AFH A8H.7	AEH A8H.6	ADH A8H.5	ACH A8H.4	ABH A8H.3	AAH A8H.2	A9H A8H.1	A8H A8H.0
A0H（P2）	A7H A0H.7	A6H A0H.6	A5H A0H.5	A4H A0H.4	A3H A0H.3	A2H A0H.2	A1H A0H.1	A0H A0H.0
98H（SCON）	9FH 98H.7	9EH 98H.6	9DH 98H.5	9CH 98H.4	9BH 98H.3	9AH 98H.2	99H 98H.1	98H 98H.0
90H（P1）	97H 90H.7	96H 90H.6	95H 90H.5	94H 90H.4	93H 90H.3	92H 90H.2	91H 90H.1	90H 90H.0
88H（TCON）	8FH 88H.7	8EH 88H.6	8DH 88H.5	8CH 88H.4	8BH 88H.3	8AH 88H.2	89H 88H.1	88H 88H.0
80H（P0）	87H 80H.7	86H 80H.6	85H 80H.5	84H 80H.4	83H 80H.3	82H 80H.2	81H 80H.1	80H 80H.0
2FH	7FH 2FH.7	7EH 2FH.6	7DH 2FH.5	7CH 2FH.4	7BH 2FH.3	7AH 2FH.2	79H 2FH.1	78H 2FH.0
2EH	77H 2EH.7	76H 2EH.6	75H 2EH.5	74H 2EH.4	73H 2EH.3	72H 2EH.2	71H 2EH.1	70H 2EH.0
2DH	6FH 2DH.7	6EH 2DH.6	6DH 2DH.5	6CH 2DH.4	6BH 2DH.3	6AH 2DH.2	69H 2DH.1	68H 2DH.0
2CH	67H 2CH.7	66H 2CH.6	65H 2CH.5	64H 2CH.4	63H 2CH.3	62H 2CH.2	61H 2CH.1	60H 2CH.0
2BH	5FH 2BH.7	5EH 2BH.6	5DH 2BH.5	5CH 2BH.4	5BH 2BH.3	5AH 2BH.2	59H 2BH.1	58H 2BH.0
2AH	57H 2AH.7	56H 2AH.6	55H 2AH.5	54H 2AH.4	53H 2AH.3	52H 2AH.2	51H 2AH.1	50H 2AH.0
29H	4FH 29H.7	4EH 29H.6	4DH 29H.5	4CH 29H.4	4BH 29H.3	4AH 29H.2	49H 29H.1	48H 29H.0

数据存储器地址	位寻址地址							
	B7	B6	B5	B4	B3	B2	B1	B0
28H	47H 28H.7	46H 28H.6	45H 28H.5	44H 28H.4	43H 28H.3	42H 28H.2	41H 28H.1	40H 28H.0
27H	3FH 27H.7	3EH 27H.6	3DH 27H.5	3CH 27H.4	3BH 27H.3	3AH 27H.2	39H 27H.1	38H 27H.0
26H	37H 26H.7	36H 26H.6	35H 26H.5	34H 26H.4	33H 26H.3	32H 26H.2	31H 26H.1	30H 26H.0
25H	2FH 25H.7	2EH 25H.6	2DH 25H.5	2CH 25H.4	2BH 25H.3	2AH 25H.2	29H 25H.1	28H 25H.0
24H	27H 24H.7	26H 24H.6	25H 24H.5	24H 24H.4	23H 24H.3	22H 24H.2	21H 24H.1	20H 24H.0
23H	1FH 23H.7	1EH 23H.6	1DH 23H.5	1CH 23H.4	1BH 23H.3	1AH 23H.2	19H 23H.1	18H 23H.0
22H	17H 22H.7	16H 22H.6	15H 22H.5	14H 22H.4	13H 22H.3	12H 22H.2	11H 22H.1	10H 22H.0
21H	0FH 21H.7	0EH 21H.6	0DH 21H.5	0CH 21H.4	0BH 21H.3	0AH 21H.2	09H 21H.1	08H 21H.0
20H	07H 20H.7	06H 20H.6	05H 20H.5	04H 20H.4	03H 20H.3	02H 20H.2	01H 20H.1	00H 20H.0

STC MCU

7.3 存储器中的特殊参数，在 ISP 下载时可烧录进程序 FLASH

STC8C 系列单片机内部的数据存储器和程序存储器中保存有与芯片相关的一些特殊参数，包括：全球唯一 ID 号、32K 掉电唤醒定时器的频率、内部 1.19V 参考信号源值以及 IRC 参数。

这些参数在 Flash 程序存储器（ROM）中的存放地址分别如下：

参数名称	保存地址					参数说明
	STC8C2K16S4 STC8C2K16S2	STC8C2K32S4 STC8C2K32S2	STC8C2K48S4 STC8C2K48S2	STC8C2K60S4 STC8C2K60S2	STC8C2K64S4 STC8C2K64S2	
全球唯一 ID 号	3FF9H~3FFFH	7FF9H~7FFFH	0BFF9H~0BFFFH	0EFF9H~0EFFFH	0FDF9H~0FDFFH	7 字节
内部 1.19V 参考信号源-BGV	3FF7H~3FF8H	7FF7H~7FF8H	0BFF7H~0BFF8H	0EFF7H~0EFF8H	0FDF7H~0FDF8H	毫伏（高字节在前）
32K 掉电唤醒定时器的频率	3FF5H~3FF6H	7FF5H~7FF6H	0BFF5H~0BFF6H	0EFF5H~0EFF6H	0FDF5H~0FDF6H	Hz （高字节在前）
22.1184MHz 的 IRC 参数	3FF4H	7FF4H	0BFF4H	0EFF4H	0FDF4H	—
24MHz 的 IRC 参数	3FF3H	7FF3H	0BFF3H	0EFF3H	0FDF3H	—
20MHz 的 IRC 参数	3FF2H	7FF2H	0BFF2H	EFF2H	FDF2H	
27MHz 的 IRC 参数	3FF1H	7FF1H	0BFF1H	EFF1H	FDF1H	
30MHz 的 IRC 参数	3FF0H	7FF0H	0BFF0H	EFF0H	FDF0H	
33.1776MHz 的 IRC 参数	3FEFH	7FEFH	0BFEFH	EFEFH	FDEFH	
35MHz 的 IRC 参数	3FEEH	7FEEH	0BFEEH	EFEEH	FDEEH	
36.864MHz 的 IRC 参数	3FEDH	7FEDH	0BFEDH	EFEDH	FDEDH	
保留	3FECH	7FECH	0BFECH	EFECH	FDECH	
保留	3FEBH	7FEBH	0BFEBH	EFEBH	FDEBH	
20M 频段的 VRTRIM 参数	3FEAH	7FEAH	0BFEAH	EFEAH	FDEAH	
35M 频段的 VRTRIM 参数	3FE9H	7FE9H	0BFE9H	EFE9H	FDE9H	

这些参数在数据存储器（RAM）中的存放地址分别如下：

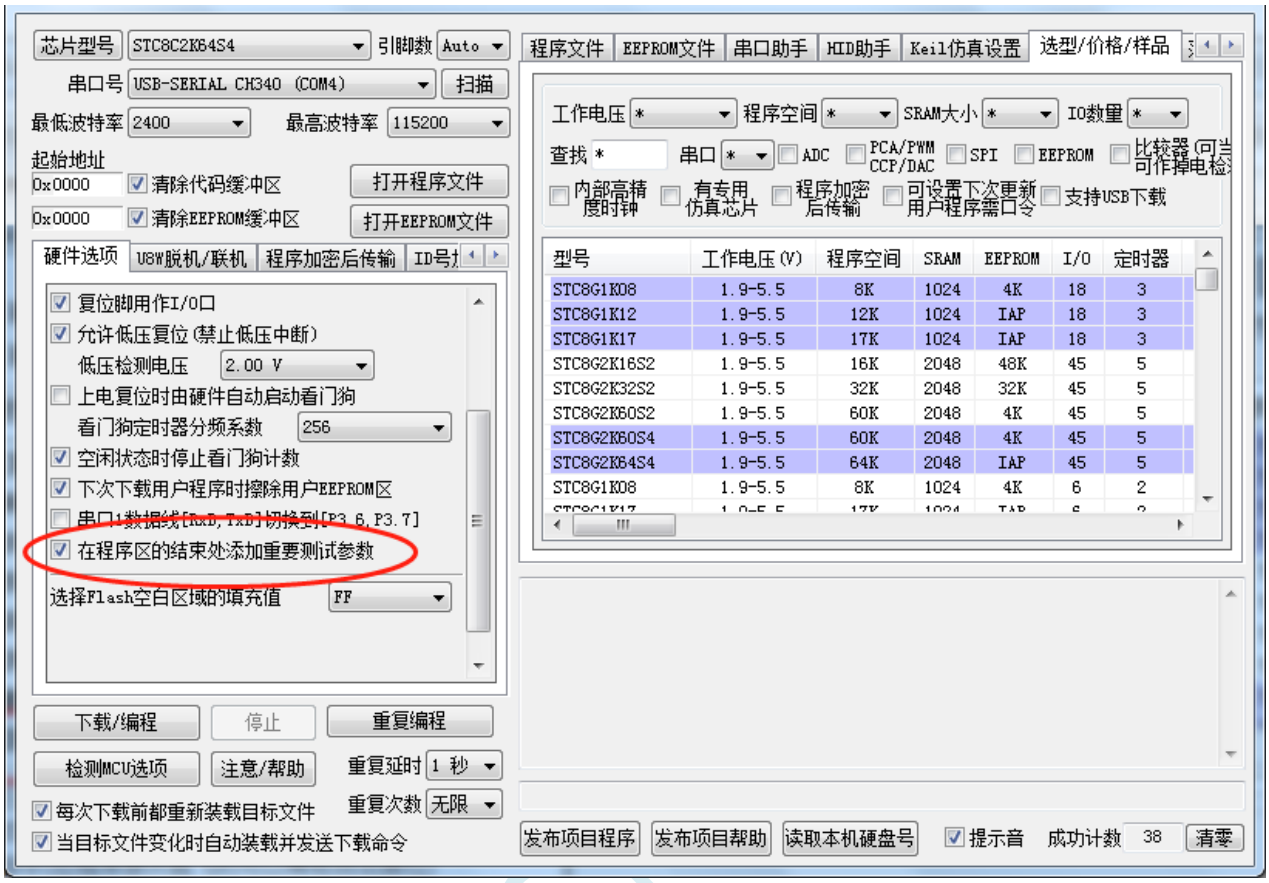
参数名称	保存地址	参数说明
内部 1.19V 参考信号源-BGV	idata：0EFH~0F0H	毫伏（高字节在前）
全球唯一 ID 号	idata：0F1H~0F7H	7 字节
32K 掉电唤醒定时器的频率	idata：0F8H~0F9H	Hz （高字节在前）
22.1184MHz 的 IRC 参数	idata：0FAH	—
24MHz 的 IRC 参数	idata：0FBH	—

特别说明

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 FLASH 程序存储器（ROM）中的 ID 数据。

2、由于 STC8C2K64S4、STC8C2K64S2 这几个型号的 EEPROM 的大小用户是可以自己设置的，有可能将保存重要参数的 FLASH 程序存储器（ROM）空间设置为 EEPROM 而人为的将重要参数擦除或修改，所以使用这个型号进行 ID 号进行加密时可能需要考虑这个问题。

3、默认情况下，Flash 程序存储器（ROM）中只有全球唯一 ID 号的数据，而内部 1.19V 参考信号源值、32K 掉电唤醒定时器的频率以及 IRC 参数都是没有的，需要在 ISP 下载时选择如下图所示的选项才可用。



7.3.1　读取内部 1.19V 参考信号源-BGV 值 (从 Flash 程序存储器（ROM）中读取)

C 语言代码

```
//测试工作频率为11.0592MHz

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

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

sfr        AUXR        =    0x8e;

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

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

bit     busy;
int     *BGV;

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

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

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

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

    BGV = (int code *)0xeff7;                   // STC8C2K60S4
    UartInit();
    ES = 1;
    EA = 1;
    UartSend(*BGV >> 8);                        //读取内部 1.19V 参考信号源的高字节
```

STC MCU

```
    UartSend(*BGV);                         //读取内部 1.19V 参考信号源的低字节

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA    8EH
BGV         EQU     0EFF7H              ;STC8C2K60S4

BUSY        BIT     20H.0

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

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

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

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

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

        LCALL   UART_INIT
        SETB    ES
        SETB    EA

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

LOOP:
        JMP     LOOP

        END
```

7.3.2 读取内部 1.19V 参考信号源 BGV 值 (从 RAM 中读取)

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr     AUXR    =   0x8e;

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

```
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

bit     busy;
int     *BGV;

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

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

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

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

    P5M0 = 0x00;
    P5M1 = 0x00;

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

STC MCU

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

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA    8EH
BGV         DATA    0EFH

BUSY        BIT     20H.0

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

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

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

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

MAIN:
```

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

        LCALL   UART_INIT
        SETB    ES
        SETB    EA

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

LOOP:
        JMP     LOOP

        END
```

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

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr AUXR = 0x8e;

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

```
bit         busy;
char        *ID;

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

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

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

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

    char i;

    ID = (char code *)0xeff9;                   // STC8C2K60S4
    UartInit();
    ES = 1;
    EA = 1;

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

STC MCU

```
        UartSend(ID[i]);
    }

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA    8EH
ID          EQU     0EFF9H              ; STC8C2K60S4

BUSY        BIT     20H.0

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

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

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

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

```
                RET

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

                LCALL       UART_INIT
                SETB        ES
                SETB        EA

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

LOOP:
                JMP         LOOP

                END
```

7.3.4 读取全球唯一 ID 号 (从 RAM 中读取)

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr       AUXR          =     0x8e;

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

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

bit     busy;
char    *ID;

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

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

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

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

    char i;

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

STC MCU

```
    EA = 1;

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA    8EH
ID          DATA    0F1H

BUSY        BIT     20H.0

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

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

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

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

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

            LCALL   UART_INIT
            SETB    ES
            SETB    EA

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

LOOP:
            JMP     LOOP

            END
```

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

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr       AUXR        =   0x8e;

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

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

bit     busy;
int     *F32K;

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

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

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

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

STC MCU

```
    F32K = (int code *)0xeff5;                  // STC8C2K60S4
    UartInit();
    ES = 1;
    EA = 1;

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA    8EH
F32K        EQU     0EFF5H          ; STC8C2K60S4

BUSY        BIT     20H.0

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

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

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

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

STC MCU

```
            RET

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

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

            LCALL   UART_INIT
            SETB    ES
            SETB    EA

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

LOOP:
            JMP     LOOP

            END
```

7.3.6 读取 32K 掉电唤醒定时器的频率 (从 RAM 中读取)

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sfr       AUXR        =   0x8e;

sfr       P0M1        =   0x93;
```

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

bit     busy;
int     *F32K;

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

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

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

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

STC MCU

```
    P5M1 = 0x00;

    F32K = (int idata *)0xf8;
    UartInit();
    ES = 1;
    EA = 1;

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA        8EH
F32K        DATA        0F8H

BUSY        BIT         20H.0

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

            ORG         0000H
            LJMP        MAIN
            ORG         0023H
            LJMP        UART_ISR

            ORG         0100H

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

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

```
                CLR     BUSY
                RET

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

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

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

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

LOOP:
                JMP     LOOP

                END
```

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

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

// 下表为 STC8C2K60S4 的参数列表
```

```
#define   ID_ROMADDR          ((unsigned char code *)0xeff9)
#define   VREF_ROMADDR        (*(unsigned int code *)0xeff7)
#define   F32K_ROMADDR        (*(unsigned int code *)0xeff5)
#define   T22M_ROMADDR        (*(unsigned char code *)0xeff4)      //22.1184MHz
#define   T24M_ROMADDR        (*(unsigned char code *)0xeff3)      //24MHz
#define   T20M_ROMADDR        (*(unsigned char code *)0xeff2)      //20MHz
#define   T27M_ROMADDR        (*(unsigned char code *)0xeff1)      //27MHz
#define   T30M_ROMADDR        (*(unsigned char code *)0xeff0)      //30MHz
#define   T33M_ROMADDR        (*(unsigned char code *)0xefef)      //33.1776MHz
#define   T35M_ROMADDR        (*(unsigned char code *)0xefee)      //35MHz
#define   T36M_ROMADDR        (*(unsigned char code *)0xefed)      //36.864MHz
#define   VRT20M_ROMADDR      (*(unsigned char code *)0xefea)      //VRTRIM_20M
#define   VRT35M_ROMADDR      (*(unsigned char code *)0xefe9)      //VRTRIM_35M

sfr       P_SW2       =     0xba;
sfr       IRCBAND     =     0x9d;
sfr       IRTRIM      =     0x9f;
sfr       VRTRIM      =     0xa6;

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

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

//  //选择 20MHz
//  P_SW2 = 0x80;
//  CLKDIV = 0x04;
//  IRTRIM = T20M_ROMADDR;
//  VRTRIM = VRT20M_ROMADDR;
//  IRCBAND = 0x00;
//  CLKDIV = 0x00;

//  //选择 22.1184MHz
//  P_SW2 = 0x80;
//  CLKDIV = 0x04;
//  IRTRIM = T22M_ROMADDR;
```

STC MCU

```
//      VRTRIM = VRT20M_ROMADDR;
//      IRCBAND = 0x00;
//      CLKDIV = 0x00;

        //选择 24MHz
        P_SW2 = 0x80;
        CLKDIV = 0x04;
        IRTRIM = T24M_ROMADDR;
        VRTRIM = VRT20M_ROMADDR;
        IRCBAND = 0x00;
        CLKDIV = 0x00;

//      //选择 27MHz
//      P_SW2 = 0x80;
//      CLKDIV = 0x04;
//      IRTRIM = T27M_ROMADDR;
//      VRTRIM = VRT35M_ROMADDR;
//      IRCBAND = 0x01;
//      CLKDIV = 0x00;

//      //选择 30MHz
//      P_SW2 = 0x80;
//      CLKDIV = 0x04;
//      IRTRIM = T30M_ROMADDR;
//      VRTRIM = VRT35M_ROMADDR;
//      IRCBAND = 0x01;
//      CLKDIV = 0x00;

//      //选择 33.1776MHz
//      P_SW2 = 0x80;
//      CLKDIV = 0x04;
//      IRTRIM = T33M_ROMADDR;
//      VRTRIM = VRT35M_ROMADDR;
//      IRCBAND = 0x01;
//      CLKDIV = 0x00;

//      //选择 35MHz
//      P_SW2 = 0x80;
//      CLKDIV = 0x04;
//      IRTRIM = T35M_ROMADDR;
//      VRTRIM = VRT35M_ROMADDR;
//      IRCBAND = 0x01;
//      CLKDIV = 0x00;

        while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

; 下表为 STC8C2K60S4 的参数列表
ID_ROMADDR          EQU     0EFF9H
VREF_ROMADDR        EQU     0EFF7H
F32K_ROMADDR        EQU     0EFF5H
T22M_ROMADDR        EQU     0EFF4H          //22.1184MHz
T24M_ROMADDR        EQU     0EFF3H          //24MHz
T20M_ROMADDR        EQU     0EFF2H          //20MHz
T27M_ROMADDR        EQU     0EFF1H          //27MHz
```

```
T30M_ROMADDR    EQU     0EFF0H          //30MHz
T33M_ROMADDR    EQU     0EFEFH          //33.1776MHz
T35M_ROMADDR    EQU     0EFEEH          //35MHz
T36M_ROMADDR    EQU     0EFEDH          //36.864MHz
VRT20M_ROMADDR  EQU     0EFEAH          //VRTRIM_20M
VRT35M_ROMADDR  EQU     0EFE9H          //VRTRIM_35M

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

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

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

            ORG     0000H
            LJMP    MAIN

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

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

STC MCU

```
;               MOV         IRCBAND,#00H
;               MOV         A,#0
;               MOV         DPTR,#CLKDIV
;               MOV         P_SW2,#00H

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

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

;               ;选择 27MHz
;               MOV         P_SW2,#80H
;               MOV         A,#4
;               MOV         DPTR,#CLKDIV
;               MOV         DPTR,#T27M_ROMADDR
;               CLR         A
;               MOVC        A,@A+DPTR
;               MOV         IRTRIM,A
;               MOV         DPTR,#VRT35M_ROMADDR
;               CLR         A
;               MOVC        A,@A+DPTR
;               MOV         VRTRIM,A
;               MOV         IRCBAND,#01H
;               MOV         A,#0
;               MOV         DPTR,#CLKDIV
;               MOV         P_SW2,#00H

;               ;选择 30MHz
;               MOV         P_SW2,#80H
;               MOV         A,#4
```

STC MCU

```
;               MOV     DPTR,#CLKDIV
;               MOV     DPTR,#T30M_ROMADDR
;               CLR     A
;               MOVC    A,@A+DPTR
;               MOV     IRTRIM,A
;               MOV     DPTR,#VRT35M_ROMADDR
;               CLR     A
;               MOVC    A,@A+DPTR
;               MOV     VRTRIM,A
;               MOV     IRCBAND,#01H
;               MOV     A,#0
;               MOV     DPTR,#CLKDIV
;               MOV     P_SW2,#00H

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

;               ;选择 35MHz
;               MOV     P_SW2,#80H
;               MOV     A,#4
;               MOV     DPTR,#CLKDIV
;               MOV     DPTR,#T35M_ROMADDR
;               CLR     A
;               MOVC    A,@A+DPTR
;               MOV     IRTRIM,A
;               MOV     DPTR,#VRT35M_ROMADDR
;               CLR     A
;               MOVC    A,@A+DPTR
;               MOV     VRTRIM,A
;               MOV     IRCBAND,#01H
;               MOV     A,#0
;               MOV     DPTR,#CLKDIV
;               MOV     P_SW2,#00H

;               ;选择 36.864MHz
;               MOV     P_SW2,#80H
;               MOV     A,#4
;               MOV     DPTR,#CLKDIV
;               MOV     DPTR,#T36M_ROMADDR
;               CLR     A
;               MOVC    A,@A+DPTR
;               MOV     IRTRIM,A
;               MOV     DPTR,#VRT35M_ROMADDR
;               CLR     A
;               MOVC    A,@A+DPTR
```

STC MCU

```
;               MOV         VRTRIM,A
;               MOV         IRCBAND,#01H
;               MOV         A,#0
;               MOV         DPTR,#CLKDIV
;               MOV         P_SW2,#00H

                JMP         $

                END
```

7.3.8 用户自定义内部 IRC 频率 (从 RAM 中读取)

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

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

char        *IRC22M;
char        *IRC24M;

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

    IRC22M = (char idata *)0xfa;
```

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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH
CLKDIV      EQU     0FE01H

IRTRIM      DATA    09FH

IRC22M      DATA    0FAH
IRC24M      DATA    0FBH

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

            ORG     0000H
            LJMP    MAIN

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

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

STC MCU

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

JMP     $

END
```

STC MCU

8 特殊功能寄存器

8.1 STC8C2K64S4-44Pin/32Pin 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							IP3H	AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H								IP3
D0H	PSW	T4T3M	TH4	TL4	TH3	TL3	T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2					
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH	S3CON	S3BUF	TA	IE2
A0H	P2	BUS_SPEED	P_SW1					
98H	SCON	SBUF	S2CON	S2BUF		IRCBAND	LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH	S4CON	S4BUF		PCON

可位寻址　　不可位寻址

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

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FEA0H			TM2PS	TM3PS	TM4PS			
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H	P0IE	P1IE						
FE28H	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR		
FE20H	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR		
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS		
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU		
FE00H	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	
FD30H	P0IM1	P1IM1	P2IM1	P3IM1	P4IM1	P5IM1		
FD20H	P0IM0	P1IM0	P2IM0	P3IM0	P4IM0	P5IM0		
FD10H	P0INTF	P1INTF	P2INTF	P3INTF	P4INTF	P5INTF		
FD00H	P0INTE	P1INTE	P2INTE	P3INTE	P4INTE	P5INTE		
FCF0H	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON

STC MCU

8.2 STC8C2K64S2-44Pin/32Pin/20Pin 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							IP3H	AUXINTIF
E0H	ACC			DPS	DPL1	DPH1	CMPCR1	CMPCR2
D8H								IP3
D0H	PSW	T4T3M	TH4	TL4	TH3	TL3	T2H	T2L
C8H	P5	P5M1	P5M0			SPSTAT	SPCTL	SPDAT
C0H	P4	WDT_CONTR	IAP_DATA	IAP_ADDRH	IAP_ADDRL	IAP_CMD	IAP_TRIG	IAP_CONTR
B8H	IP	SADEN	P_SW2					
B0H	P3	P3M1	P3M0	P4M1	P4M0	IP2	IP2H	IPH
A8H	IE	SADDR	WKTCL	WKTCH			TA	IE2
A0H	P2	BUS_SPEED	P_SW1					
98H	SCON	SBUF	S2CON	S2BUF		IRCBAND	LIRTRIM	IRTRIM
90H	P1	P1M1	P1M0	P0M1	P0M0	P2M1	P2M0	
88H	TCON	TMOD	TL0	TL1	TH0	TH1	AUXR	INTCLKO
80H	P0	SP	DPL	DPH				PCON

可位寻址　　不可位寻址

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

	0/8	1/9	2/A	3/B	4/C	5/D	6/E	7/F
FEA0H			TM2PS	TM3PS	TM4PS			
FE88H	I2CMSAUX							
FE80H	I2CCFG	I2CMSCR	I2CMSST	I2CSLCR	I2CSLST	I2CSLADR	I2CTxD	I2CRxD
FE30H	P0IE	P1IE						
FE28H	P0DR	P1DR	P2DR	P3DR	P4DR	P5DR		
FE20H	P0SR	P1SR	P2SR	P3SR	P4SR	P5SR		
FE18H	P0NCS	P1NCS	P2NCS	P3NCS	P4NCS	P5NCS		
FE10H	P0PU	P1PU	P2PU	P3PU	P4PU	P5PU		
FE00H	CLKSEL	CLKDIV	HIRCCR	XOSCCR	IRC32KCR	MCLKOCR	IRCDB	
FD30H	P0IM1	P1IM1	P2IM1	P3IM1	P4IM1	P5IM1		
FD20H	P0IM0	P1IM0	P2IM0	P3IM0	P4IM0	P5IM0		
FD10H	P0INTF	P1INTF	P2INTF	P3INTF	P4INTF	P5INTF		
FD00H	P0INTE	P1INTE	P2INTE	P3INTE	P4INTE	P5INTE		
FCF0H	MD3	MD2	MD1	MD0	MD5	MD4	ARCON	OPCON

8.3 特殊功能寄存器列表

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

STC8C 系列能进行位寻址的寄存器：P0（80H）、TCON（88H）、P1（90H）、SCON（98H）、P2（A0H）、IE（A8H）、P3（B0H）、IP（B8H）、P4（C0H）、P5（C8H）、PSW（D0H）、ACC（E0H）、B（F0H）

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	P0 端口	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111
SP	堆栈指针	81H									0000,0111
DPL	数据指针（低字节）	82H									0000,0000
DPH	数据指针（高字节）	83H									0000,0000
S4CON	串口 4 控制寄存器	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI	0000,0000
S4BUF	串口 4 数据寄存器	85H									0000,0000
PCON	电源控制寄存器	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL	0011,0000
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	定时器 0 低 8 位寄存器	8AH									0000,0000
TL1	定时器 1 低 8 位寄存器	8BH									0000,0000
TH0	定时器 0 高 8 位寄存器	8CH									0000,0000
TH1	定时器 1 高 8 位寄存器	8DH									0000,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
INTCLKO	中断与时钟输出控制	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
P1	P1 端口	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P1M1	P1 口配置寄存器 1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	1111,1111
P1M0	P1 口配置寄存器 0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0	0000,0000
P0M1	P0 口配置寄存器 1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	1111,1111
P0M0	P0 口配置寄存器 0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	0000,0000
P2M1	P2 口配置寄存器 1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1	1111,1111
P2M0	P2 口配置寄存器 0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0	0000,0000
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	串口 1 数据寄存器	99H									0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0x00,0000
S2BUF	串口 2 数据寄存器	9BH									0000,0000
IRCBAND	IRC 频段选择检测	9DH	-	-	-	-	-	-	-	SEL	xxxx,xxxn
LIRTRIM	IRC 频率微调寄存器	9EH	-	-	-	-	-	-	LIRTRIM[1:0]		xxxx,xxnn
IRTRIM	IRC 频率调整寄存器	9FH	IRTRIM[7:0]								nnnn,nnnn
P2	P2 端口	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
BUS_SPEED	总线速度控制寄存器	A1H	RW_S[1:0]		-	-	-	SPEED[2:0]			00xx,x000
P_SW1	外设端口切换寄存器 1	A2H	S1_S[1:0]		-	-	SPI_S[1:0]		0	-	nnxx,000x
IE	中断允许寄存器	A8H	EA	ELVD	-	ES	ET1	EX1	ET0	EX0	00x0,0000
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	S3ST4	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI	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	
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	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P3M1	P3 口配置寄存器 1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1	1111,1100
P3M0	P3 口配置寄存器 0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0	0000,0000
P4M1	P4 口配置寄存器 1	B3H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1	1111,1111
P4M0	P4 口配置寄存器 0	B4H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0	0000,0000
IP2	中断优先级控制寄存器 2	B5H	-	PI2C	PCMP	PX4	-	-	PSPI	PS2	x000,xx00
IP2H	高中断优先级控制寄存器 2	B6H	-	PI2CH	PCMPH	PX4H	-	-	PSPIH	PS2H	x000,xx00
IPH	高中断优先级控制寄存器	B7H	-	PLVDH	-	PSH	PT1H	PX1H	PT0H	PX0H	x0x0,0000
IP	中断优先级控制寄存器	B8H	-	PLVD	-	PS	PT1	PX1	PT0	PX0	x0x0,0000
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	-	-	S2_S	0x00,0xx0
P4	P4 端口	C0H	P47	P46	P45	P44	P43	P42	P41	P40	1111,1111
WDT_CONTR	看门狗控制寄存器	C1H	WDT_FLAG	-	EN_WDT	CLR_WDT	IDL_WDT	WDT_PS[2:0]			0xn0,nnnn
IAP_DATA	IAP 数据寄存器	C2H									1111,1111
IAP_ADDRH	IAP 高地址寄存器	C3H									0000,0000
IAP_ADDRL	IAP 低地址寄存器	C4H									0000,0000
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	-	CMD[1:0]		xxxx,xx00
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx
P5	P5 端口	C8H	-	-	P55	P54	-	-	-	-	xx11,xxxx
P5M1	P5 口配置寄存器 1	C9H	-	-	P55M1	P54M1	-	-	-	-	xx11,xxxx
P5M0	P5 口配置寄存器 0	CAH	-	-	P55M0	P54M0	-	-	-	-	xx00,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	F1	P	0000,0000
T4T3M	定时器 4/3 控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
T4H	定时器 4 高字节	D2H									0000,0000
T4L	定时器 4 低字节	D3H									0000,0000
T3H	定时器 3 高字节	D4H									0000,0000
T3L	定时器 3 低字节	D5H									0000,0000
T2H	定时器 2 高字节	D6H									0000,0000
T2L	定时器 2 低字节	D7H									0000,0000
IP3	中断优先级控制寄存器 3	DFH	-	-	-	-	-	-	PS4	PS3	xxxx,xx00
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

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR2	比较器控制寄存器 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,0000
IP3H	高中断优先级控制寄存器 3	EEH	-	-	-	-	-	-	PS4H	PS3H	xxxx,xx00
AUXINTIF	扩展外部中断标志寄存器	EFH	-	INT4IF	INT3IF	INT2IF	-	-	-	T2IF	x000,xxx0
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]		xnxn,xxnn

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

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CLKSEL	时钟选择寄存器	FE00H	-	-	-	-	-	-	MCKSEL[1:0]		xxxx,xx00
CLKDIV	时钟分频寄存器	FE01H									nnnn,nnnn
HIRCCR	内部高速振荡器控制寄存器	FE02H	ENHIRC	-	-	-	-	-	-	HIRCST	1xxx,xxx0
XOSCCR	外部晶振控制寄存器	FE03H	ENXOSC	XITYPE	-	-	-	-	-	XOSCST	00xx,xxx0
IRC32KCR	内部 32K 振荡器控制寄存器	FE04H	ENIRC32K	-	-	-	-	-	-	IRC32KST	0xxx,xxx0
MCLKOCR	主时钟输出控制寄存器	FE05H	MCLKO_S	MCLKODIV[6:0]							0000,0000
IRCDB	内部高速振荡器去抖控制	FE06H	IRCDB_PAR[7:0]								1000,0000
P0PU	P0 口上拉电阻控制寄存器	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU	0000,0000
P1PU	P1 口上拉电阻控制寄存器	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU	0000,0000
P2PU	P2 口上拉电阻控制寄存器	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU	0000,0000
P3PU	P3 口上拉电阻控制寄存器	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU	0000,0000
P4PU	P4 口上拉电阻控制寄存器	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU	0000,0000
P5PU	P5 口上拉电阻控制寄存器	FE15H	-	-	P55PU	P54PU	-	-	-	-	xx00,xxxx
P0NCS	P0 口施密特触发控制寄存器	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS	0000,0000
P1NCS	P1 口施密特触发控制寄存器	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS	0000,0000
P2NCS	P2 口施密特触发控制寄存器	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS	0000,0000
P3NCS	P3 口施密特触发控制寄存器	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS	0000,0000
P4NCS	P4 口施密特触发控制寄存器	FE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS	0000,0000
P5NCS	P5 口施密特触发控制寄存器	FE1DH	-	-	P55NCS	P54NCS	-	-	-	-	xx00,xxxx
P0SR	P0 口电平转换速率寄存器	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	1111,1111
P1SR	P1 口电平转换速率寄存器	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,1111
P2SR	P2 口电平转换速率寄存器	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,1111
P3SR	P3 口电平转换速率寄存器	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,1111
P4SR	P4 口电平转换速率寄存器	FE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR	1111,1111
P5SR	P5 口电平转换速率寄存器	FE25H	-	-	P55SR	P54SR	-	-	-	-	xx11,xxxx
P0DR	P0 口驱动电流控制寄存器	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	1111,1111
P1DR	P1 口驱动电流控制寄存器	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,1111
P2DR	P2 口驱动电流控制寄存器	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,1111
P3DR	P3 口驱动电流控制寄存器	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,1111
P4DR	P4 口驱动电流控制寄存器	FE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR	1111,1111

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P5DR	P5 口驱动电流控制寄存器	FE2DH	-	-	P55DR	P54DR	-	-	-	-	xx11,xxxx
P0IE	P0 口输入使能控制寄存器	FE30H	-	P06IE	P05IE	P04IE	P03IE	P02IE	P01IE	P00IE	x111,1111
P1IE	P1 口输入使能控制寄存器	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE	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	I2CSLADR[7:1]							MA	0000,0000
I2CTXD	I^2C 数据发送寄存器	FE86H									0000,0000
I2CRXD	I^2C 数据接收寄存器	FE87H									0000,0000
I2CMSAUX	I^2C 主机辅助控制寄存器	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0
TM2PS	定时器 2 时钟预分频寄存器	FEA2H									0000,0000
TM3PS	定时器 3 时钟预分频寄存器	FEA3H									0000,0000
TM4PS	定时器 4 时钟预分频寄存器	FEA4H									0000,0000
MD3	MDU 数据寄存器	FCF0H	MD3[7:0]								0000,0000
MD2	MDU 数据寄存器	FCF1H	MD2[7:0]								0000,0000
MD1	MDU 数据寄存器	FCF2H	MD1[7:0]								0000,0000
MD0	MDU 数据寄存器	FCF3H	MD0[7:0]								0000,0000
MD5	MDU 数据寄存器	FCF4H	MD5[7:0]								0000,0000
MD4	MDU 数据寄存器	FCF5H	MD4[7:0]								0000,0000
ARCON	MDU 模式控制寄存器	FCF6H	MODE[2:0]			SC[4:0]					0000,0000
OPCON	MDU 操作控制寄存器	FCF7H	-	MDOV	-	-	-	-	RST	ENOP	0000,0000
P0INTE	P0 口中断使能寄存器	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE	0000,0000
P1INTE	P1 口中断使能寄存器	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE	0000,0000
P2INTE	P2 口中断使能寄存器	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE	0000,0000
P3INTE	P3 口中断使能寄存器	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE	0000,0000
P4INTE	P4 口中断使能寄存器	FD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE	0000,0000
P5INTE	P5 口中断使能寄存器	FD05H	-	-	P55INTE	P54INTE	-	-	-	-	xx00,xxxx
P0INTF	P0 口中断标志寄存器	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF	0000,0000
P1INTF	P1 口中断标志寄存器	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF	0000,0000
P2INTF	P2 口中断标志寄存器	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF	0000,0000
P3INTF	P3 口中断标志寄存器	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF	0000,0000
P4INTF	P4 口中断标志寄存器	FD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF	0000,0000
P5INTF	P5 口中断标志寄存器	FD15H	-	-	P55INTF	P54INTF	-	-	-	-	xx00,xxxx
P0IM0	P0 口中断模式寄存器 0	FD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0	0000,0000
P1IM0	P1 口中断模式寄存器 0	FD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0	0000,0000
P2IM0	P2 口中断模式寄存器 0	FD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0	0000,0000
P3IM0	P3 口中断模式寄存器 0	FD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0	0000,0000
P4IM0	P4 口中断模式寄存器 0	FD24H	P47IM0	P46IM0	P45IM0	P44IM0	P43IM0	P42IM0	P41IM0	P40IM0	0000,0000
P5IM0	P5 口中断模式寄存器 0	FD25H	-	-	P55IM0	P54IM0	-	-	-	-	xx00,xxxx
P0IM1	P0 口中断模式寄存器 1	FD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1	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	
P1IM1	P1 口中断模式寄存器 1	FD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1	0000,0000
P2IM1	P2 口中断模式寄存器 1	FD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1	0000,0000
P3IM1	P3 口中断模式寄存器 1	FD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1	0000,0000
P4IM1	P4 口中断模式寄存器 1	FD34H	P47IM1	P46IM1	P45IM1	P44IM1	P43IM1	P42IM1	P41IM1	P40IM1	0000,0000
P5IM1	P5 口中断模式寄存器 1	FD35H	-	-	P55IM1	P54IM1	-	-	-	-	xx00,xxxx

注：特殊功能寄存器初始值意义

0： 初始值为 0；

1： 初始值为 1；

n： 初始值与 ISP 下载时的硬件选项有关；

x： 不存在这个位，初始值不确定

STC MCU

9 I/O 口

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）、开漏模式。可使用软件对 I/O 口的工作模式进行容易配置。

关于 I/O 的注意事项：

1 P3.0 和 P3.1 口上电后的状态为弱上拉/准双向口模式

2 除 P3.0 和 P3.1 外，其余所有 IO 口上电后的状态均为高阻输入状态，用户在使用 IO 口前必须先设置 IO 口模式

3 芯片上电时如果不需要使用 USB 进行 ISP 下载，P3.0/P3.1/P3.2 这 3 个 I/O 口不能同时为低电平，否则会进入 USB 下载模式而无法运行用户代码

4 芯片上电时，若 P3.0 和 P3.1 同时为低电平，P3.2 口会短时间由高阻输入状态切换到双向口模式，用以读取 P3.2 口外部状态来判断是否需要进入 USB 下载模式

5 当使用 P5.4 当作复位脚时，这个端口内部的 4K 上拉电阻会一直打开；但 P5.4 做普通 I/O 口时，基于这个 I/O 口与复位脚共享管脚的特殊考量，端口内部的 4K 上拉电阻依然会打开大约 6.5 毫秒时间，再自动关闭（当用户的电路设计需要使用 P5.4 口驱动外部电路时，请务必考虑上电瞬间会有 6.5 毫秒时间的高电平的问题）

9.1 I/O 口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0	P0 端口	80H	P07	P06	P05	P04	P03	P02	P01	P00	1111,1111
P1	P1 端口	90H	P17	P16	P15	P14	P13	P12	P11	P10	1111,1111
P2	P2 端口	A0H	P27	P26	P25	P24	P23	P22	P21	P20	1111,1111
P3	P3 端口	B0H	P37	P36	P35	P34	P33	P32	P31	P30	1111,1111
P4	P4 端口	C0H	P47	P46	P45	P44	P43	P42	P41	P40	1111,1111
P5	P5 端口	C8H	-	-	P55	P54	-	-	-	-	xx11,xxxx
P0M1	P0 口配置寄存器 1	93H	P07M1	P06M1	P05M1	P04M1	P03M1	P02M1	P01M1	P00M1	1111,1111
P0M0	P0 口配置寄存器 0	94H	P07M0	P06M0	P05M0	P04M0	P03M0	P02M0	P01M0	P00M0	0000,0000
P1M1	P1 口配置寄存器 1	91H	P17M1	P16M1	P15M1	P14M1	P13M1	P12M1	P11M1	P10M1	1111,1111
P1M0	P1 口配置寄存器 0	92H	P17M0	P16M0	P15M0	P14M0	P13M0	P12M0	P11M0	P10M0	0000,0000
P2M1	P2 口配置寄存器 1	95H	P27M1	P26M1	P25M1	P24M1	P23M1	P22M1	P21M1	P20M1	1111,1111
P2M0	P2 口配置寄存器 0	96H	P27M0	P26M0	P25M0	P24M0	P23M0	P22M0	P21M0	P20M0	0000,0000
P3M1	P3 口配置寄存器 1	B1H	P37M1	P36M1	P35M1	P34M1	P33M1	P32M1	P31M1	P30M1	n111,1100
P3M0	P3 口配置寄存器 0	B2H	P37M0	P36M0	P35M0	P34M0	P33M0	P32M0	P31M0	P30M0	n000,0000
P4M1	P4 口配置寄存器 1	B3H	P47M1	P46M1	P45M1	P44M1	P43M1	P42M1	P41M1	P40M1	1111,1111
P4M0	P4 口配置寄存器 0	B4H	P47M0	P46M0	P45M0	P44M0	P43M0	P42M0	P41M0	P40M0	0000,0000
P5M1	P5 口配置寄存器 1	C9H	-	-	P55M1	P54M1	-	-	-	-	xx11,xxxx
P5M0	P5 口配置寄存器 0	CAH	-	-	P55M0	P54M0	-	-	-	-	xx00,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	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU	0000,0000
P1PU	P1 口上拉电阻控制寄存器	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU	0000,0000
P2PU	P2 口上拉电阻控制寄存器	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU	0000,0000
P3PU	P3 口上拉电阻控制寄存器	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU	0000,0000
P4PU	P4 口上拉电阻控制寄存器	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU	0000,0000
P5PU	P5 口上拉电阻控制寄存器	FE15H	-	-	P55PU	P54PU	-	-	-	-	xx00,xxxx
P0NCS	P0 口施密特触发控制寄存器	FE18H	P07NCS	P06NCS	P05NCS	P04NCS	P03NCS	P02NCS	P01NCS	P00NCS	0000,0000
P1NCS	P1 口施密特触发控制寄存器	FE19H	P17NCS	P16NCS	P15NCS	P14NCS	P13NCS	P12NCS	P11NCS	P10NCS	0000,0000
P2NCS	P2 口施密特触发控制寄存器	FE1AH	P27NCS	P26NCS	P25NCS	P24NCS	P23NCS	P22NCS	P21NCS	P20NCS	0000,0000
P3NCS	P3 口施密特触发控制寄存器	FE1BH	P37NCS	P36NCS	P35NCS	P34NCS	P33NCS	P32NCS	P31NCS	P30NCS	0000,0000
P4NCS	P4 口施密特触发控制寄存器	FE1CH	P47NCS	P46NCS	P45NCS	P44NCS	P43NCS	P42NCS	P41NCS	P40NCS	0000,0000
P5NCS	P5 口施密特触发控制寄存器	FE1DH	-	-	P55NCS	P54NCS	-	-	-	-	xx00,xxxx
P0SR	P0 口电平转换速率寄存器	FE20H	P07SR	P06SR	P05SR	P04SR	P03SR	P02SR	P01SR	P00SR	1111,1111
P1SR	P1 口电平转换速率寄存器	FE21H	P17SR	P16SR	P15SR	P14SR	P13SR	P12SR	P11SR	P10SR	1111,1111
P2SR	P2 口电平转换速率寄存器	FE22H	P27SR	P26SR	P25SR	P24SR	P23SR	P22SR	P21SR	P20SR	1111,1111
P3SR	P3 口电平转换速率寄存器	FE23H	P37SR	P36SR	P35SR	P34SR	P33SR	P32SR	P31SR	P30SR	1111,1111
P4SR	P4 口电平转换速率寄存器	FE24H	P47SR	P46SR	P45SR	P44SR	P43SR	P42SR	P41SR	P40SR	1111,1111
P5SR	P5 口电平转换速率寄存器	FE25H	-	-	P55SR	P54SR	-	-	-	-	xx11,xxxx
P0DR	P0 口驱动电流控制寄存器	FE28H	P07DR	P06DR	P05DR	P04DR	P03DR	P02DR	P01DR	P00DR	1111,1111
P1DR	P1 口驱动电流控制寄存器	FE29H	P17DR	P16DR	P15DR	P14DR	P13DR	P12DR	P11DR	P10DR	1111,1111
P2DR	P2 口驱动电流控制寄存器	FE2AH	P27DR	P26DR	P25DR	P24DR	P23DR	P22DR	P21DR	P20DR	1111,1111
P3DR	P3 口驱动电流控制寄存器	FE2BH	P37DR	P36DR	P35DR	P34DR	P33DR	P32DR	P31DR	P30DR	1111,1111
P4DR	P4 口驱动电流控制寄存器	FE2CH	P47DR	P46DR	P45DR	P44DR	P43DR	P42DR	P41DR	P40DR	1111,1111
P5DR	P5 口驱动电流控制寄存器	FE2DH	-	-	P55DR	P54DR	-	-	-	-	xx11,xxxx
P0IE	P0 口输入使能控制寄存器	FE30H	-	P06IE	P05IE	P04IE	P03IE	P02IE	P01IE	P00IE	x111,1111
P1IE	P1 口输入使能控制寄存器	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE	1111,1111

9.1.1 端口数据寄存器（Px）

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

读写端口状态

写 0：输出低电平到端口缓冲区

写 1：输出高电平到端口缓冲区

读：直接读端口管脚上的电平

9.1.2 端口模式配置寄存器（PxM0，PxM1）

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

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	开漏模式

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

9.1.3 端口上拉电阻控制寄存器（PxPU）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	FE15H	-	-	P55PU	P54PU	-	-	-	-

端口内部4.1K上拉电阻控制位（注：P3.0和P3.1口上的上拉电阻可能会略小一些）

0：禁止端口内部的 4.1K 上拉电阻

1：使能端口内部的 4.1K 上拉电阻

9.1.4 端口施密特触发控制寄存器（PxNCS）

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

端口施密特触发控制位

0：使能端口的施密特触发功能。（上电复位后默认使能施密特触发）

1：禁止端口的施密特触发功能。

9.1.5 端口电平转换速度控制寄存器（PxSR）

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

控制端口电平转换的速度

0：电平转换速度快，相应的上下冲会比较大

1：电平转换速度慢，相应的上下冲比较小

9.1.6 端口驱动电流控制寄存器（PxDR）

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

控制端口的驱动能力

0：增强驱动能力

1：一般驱动能力

9.1.7 端口数字信号输入使能控制寄存器（PxIE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0IE	FE30H	-	P06IE	P05IE	P04IE	P03IE	P02IE	P01IE	P00IE
P1IE	FE31H	P17IE	P16IE	P15IE	P14IE	P13IE	P12IE	P11IE	P10IE

数字信号输入使能控制

0：禁止数字信号输入。若 I/O 被当作比较器输入口、ADC 输入口或者触摸按键输入口等模拟口时，进入时钟停振模式前，必须设置为 0，否则会有额外的耗电。

1：使能数字信号输入。若 I/O 被当作数字口时，必须设置为 1，否 MCU 无法读取外部端口的电平。

STC MCU

9.2 配置 I/O 口

每个 I/O 的配置都需要使用两个寄存器进行设置。

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

即 P0M0 的第 0 位和 P0M1 的第 0 位组合起来配置 P0.0 口的模式
即 P0M0 的第 1 位和 P0M1 的第 1 位组合起来配置 P0.1 口的模式
其他所有 I/O 的配置都与此类似。

PnM0 与 PnM1 的组合方式如下表所示

PnM1	PnM0	I/O 口工作模式
0	0	准双向口（传统8051端口模式，弱上拉） 灌电流可达20mA，拉电流为270～150μA（存在制造误差）
0	1	推挽输出（强上拉输出，可达20mA，要加限流电阻）
1	0	高阻输入（电流既不能流入也不能流出）
1	1	开漏模式（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。 **===【开漏工作模式】，对外设置输出为 1，等同于 【高阻输入】** **===【开漏工作模式】,【打开内部上拉电阻	或外部加上拉电阻】，简单等同于 【准双向口】**

注：n = 0, 1, 2, 3, 4, 5, 6, 7

注意：

虽然每个I/O口在弱上拉（准双向口）/强推挽输出/开漏模式时都能承受20mA的灌电流（还是要加限流电阻，如1K、560Ω、472Ω等），在强推挽输出时能输出20mA的拉电流（也要加限流电阻），但整个芯片的工作电流推荐不要超过70mA，即从Vcc流入的电流建议不要超过70mA，从Gnd流出电流建议不要超过70mA，整体流入/流出电流建议都不要超过70mA。（STC8C1K08A-8Pin系列和STC8C1K08-8Pin系列，整个芯片的工作电流推荐不要超过35mA，即从Vcc流入的电流建议不要超过35mA，从Gnd流出电流建议不要超过35mA，整体流入/流出电流建议都不要超过35mA）

9.3 I/O 的结构图

9.3.1 准双向口（弱上拉）

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

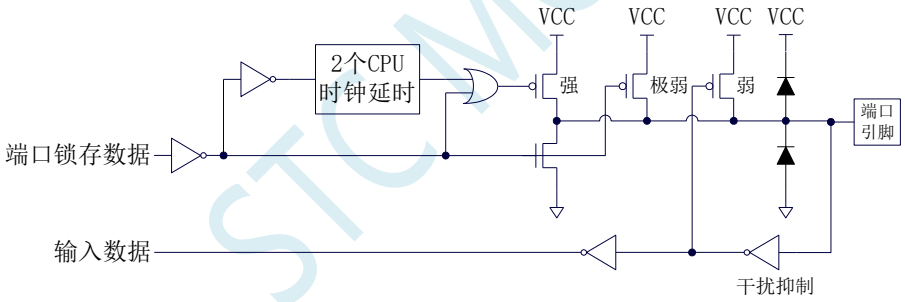
在 3 个上拉晶体管中，有 1 个上拉晶体管称为“弱上拉”，当端口寄存器为 1 且引脚本身也为 1 时打开。此上拉提供基本驱动电流使准双向口输出为 1。如果一个引脚输出为 1 而由外部装置下拉到低时，弱上拉关闭而“极弱上拉”维持开状态，为了把这个引脚强拉为低，外部装置必须有足够的灌电流能力使引脚上的电压降到门槛电压以下。对于 5V 单片机，“弱上拉”晶体管的电流约 250uA；对于 3.3V 单片机，“弱上拉”晶体管的电流约 150uA。

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

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

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

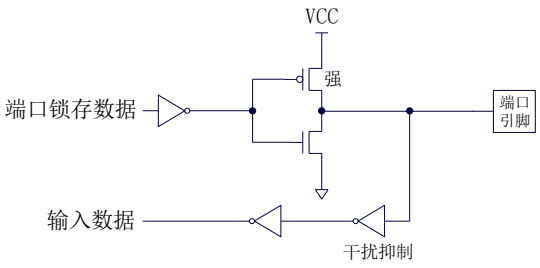
准双向口（弱上拉）输出如下图所示：



9.3.2 推挽输出

强推挽输出配置的下拉结构与开漏模式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：

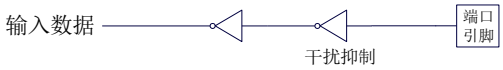


9.3.3 高阻输入

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 开漏模式

===【开漏工作模式】，对外设置输出为 1，等同于 【高阻输入】

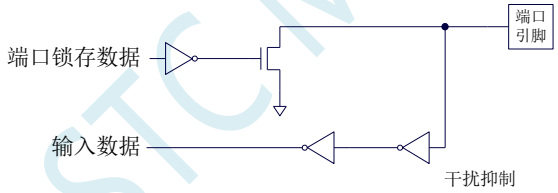
===【开漏工作模式】,【打开内部上拉电阻 | 或外部加上拉电阻】，简单等同于 【准双向口】

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

开漏端口带有一个施密特触发输入以及一个干扰抑制电路。

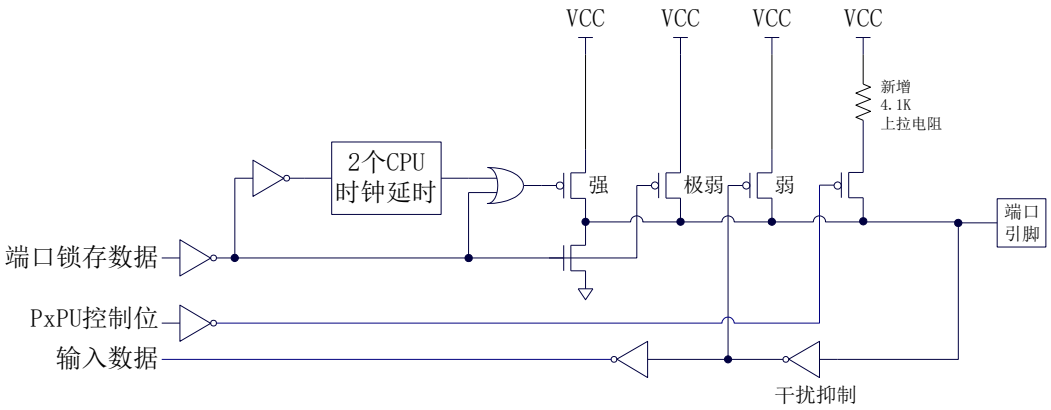
输出端口配置如下图所示：



STCMCU

9.3.5 新增 4.1K 上拉电阻

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



端口上拉电阻控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0PU	FE10H	P07PU	P06PU	P05PU	P04PU	P03PU	P02PU	P01PU	P00PU
P1PU	FE11H	P17PU	P16PU	P15PU	P14PU	P13PU	P12PU	P11PU	P10PU
P2PU	FE12H	P27PU	P26PU	P25PU	P24PU	P23PU	P22PU	P21PU	P20PU
P3PU	FE13H	P37PU	P36PU	P35PU	P34PU	P33PU	P32PU	P31PU	P30PU
P4PU	FE14H	P47PU	P46PU	P45PU	P44PU	P43PU	P42PU	P41PU	P40PU
P5PU	FE15H	-	-	P55PU	P54PU	-	-	-	-

端口内部4.1K上拉电阻控制位（注：P3.0和P3.1口上的上拉电阻可能会略小一些）

0：禁止端口内部的 4.1K 上拉电阻

1：使能端口内部的 4.1K 上拉电阻

9.3.6 如何设置 I/O 口对外输出速度

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

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

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

9.3.7 如何设置 I/O 口电流驱动能力

若需要改变 I/O 口的电流驱动能力，可通过设置 PxDR 寄存器来实现

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

9.3.8 如何降低 I/O 口对外辐射

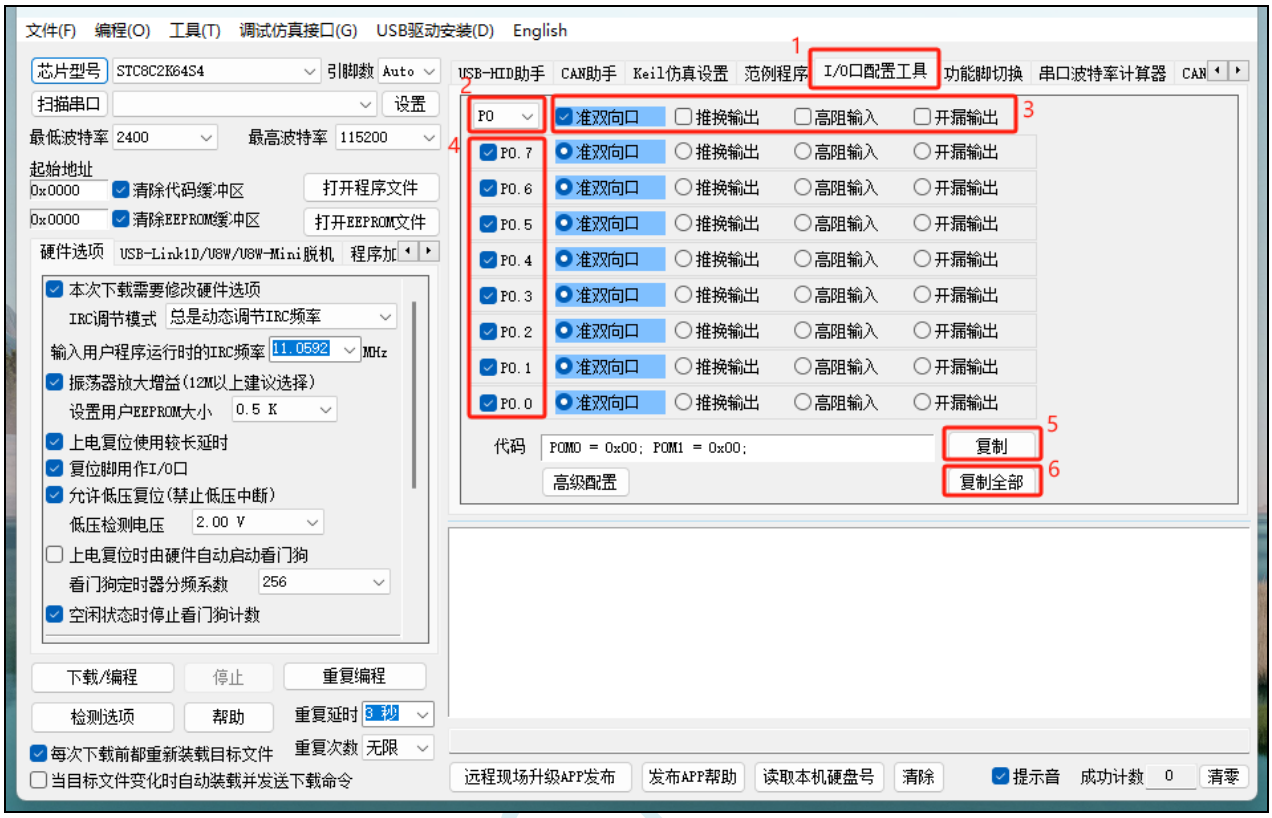
由于设置 PxSR 寄存器，可用于控制 I/O 口电平转换速度，设置 PxDR 寄存器，可用于控制 I/O 口驱动电流大小

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

STC MCU

9.4 AIapp-ISP | I/O 口配置工具

9.4.1 普通配置模式

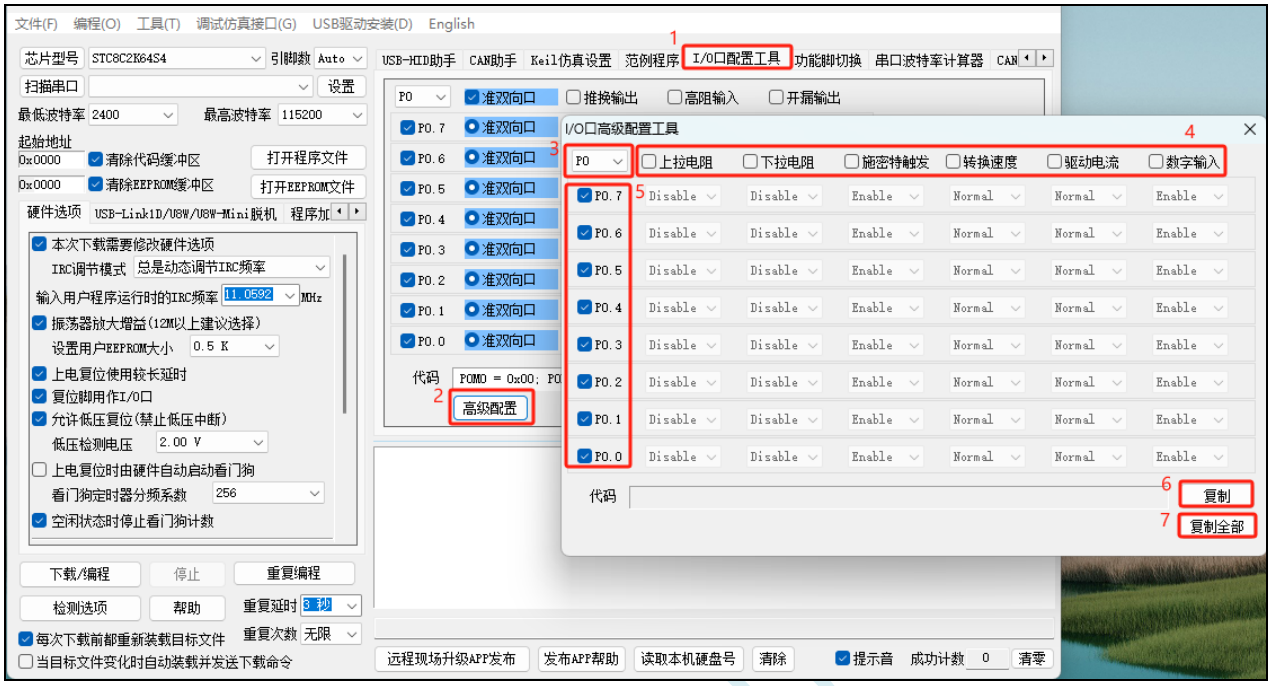


普通模式可以配置所有 I/O 的 4 种模式：

准双向口模式
推挽输出模式
高阻输入模式
开漏模式

①：在下载软件中选择“I/O 口配置工具”功能页，进入 I/O 口配置界面
②：选择需要配置的 I/O 口组，支持 P0~P7
③：整组 I/O 进行设置（将整组 P0/P1/…/P7 设置为）准双向口模式、推挽输出模式等
④：选择使能配置每组 I/O 中的端口
⑤：复制当前组 I/O 的配置代码
⑥：复制 P0~P7 口的全部配置代码

9.4.2　高级配置模式



高级模式可以配置 I/O：

使能/关闭上拉电阻
使能/关闭下拉电阻
使能/关闭施密特触发功能
选择 I/O 转换速度
选择 I/O 口驱动电流
使能/关闭 I/O 的数组输入功能

①：在下载软件中选择“I/O 口配置工具”功能页，进入 I/O 口配置界面
②：点击界面中的“高级配置”按钮进入 I/O 口高级配置界面
③：选择需要配置的 I/O 口组，支持 P0~P7
④：选择需要的配置模式
⑤：选择使能配置每组 I/O 中的端口
⑥：复制当前组 I/O 的配置代码
⑦：复制 P0~P7 口的全部配置代码

9.5 范例程序

9.5.1 端口模式设置

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     P0M0    =   0x94;
sfr     P0M1    =   0x93;
sfr     P1M0    =   0x92;
sfr     P1M1    =   0x91;
sfr     P2M0    =   0x96;
sfr     P2M1    =   0x95;
sfr     P3M0    =   0xb2;
sfr     P3M1    =   0xb1;
sfr     P4M0    =   0xb4;
sfr     P4M1    =   0xb3;
sfr     P5M0    =   0xca;
sfr     P5M1    =   0xc9;
sfr     P6M0    =   0xcc;
sfr     P6M1    =   0xcb;
sfr     P7M0    =   0xe2;
sfr     P7M1    =   0xe1;

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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

STC MCU

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

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH

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

            JMP     $

            END
```

9.5.2 双向口读写操作

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

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

    P0M0 = 0x00;                    //设置 P0.0~P0.7 为双向口模式
    P0M1 = 0x00;

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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN

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

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

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

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

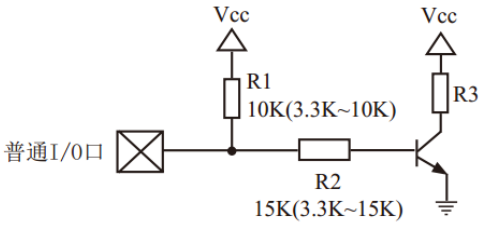
SETB    P0.0                ;读取端口前先使能内部弱上拉电阻
NOP                         ;等待两个时钟
NOP
MOV     C,P0.0              ;读取端口状态

JMP     $

END
```

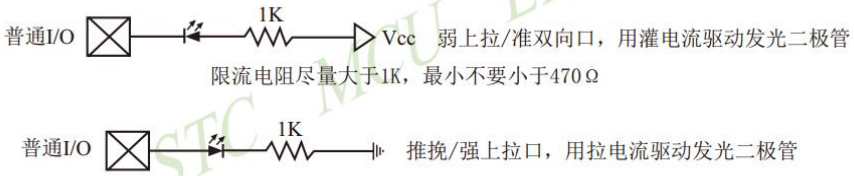
STC MCU

9.6 一种典型三极管控制电路



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

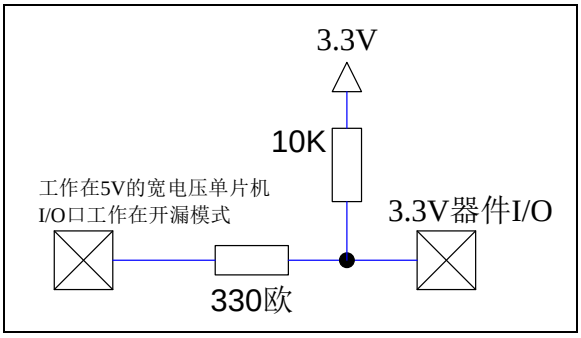
9.7 典型发光二极管控制电路



9.8　混合电压供电系统 3V/5V 器件 I/O 口互连

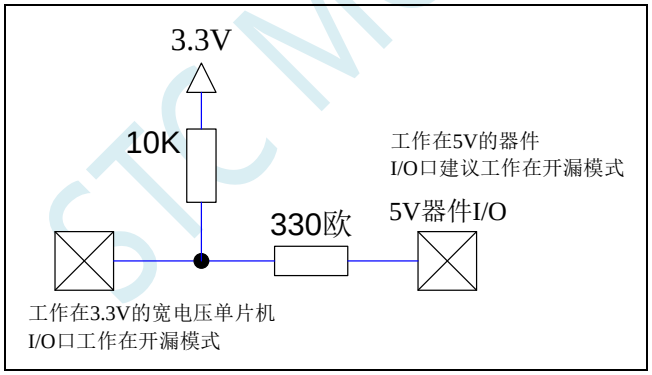
宽电压单片机工作在 5V 时:

如需要直接连接 3.3V 器件时，为防止 3.3V 器件承受不了 5V，可将相应的单片机 I/O 口先串一个 330Ω 的限流电阻到 3.3V 器件 I/O 口，程序初始化时将单片机的 I/O 口设置成开漏工作模式，断开内部上拉电阻，相应的 3.3V 器件 I/O 口外部加 10K 上拉电阻到 3.3V 器件的 Vcc，这样高电平是 3.3V，低电平是 0V，输入输出一切正常。宽电压单片机 2.2V 以上肯定是高电平。



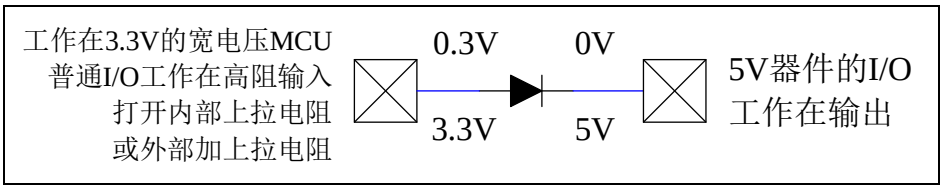
宽电压 MCU 如工作在 3.3V 时，如需要连接到 5V 器件:

1、如果对方 5V 器件可以工作在开漏模式:

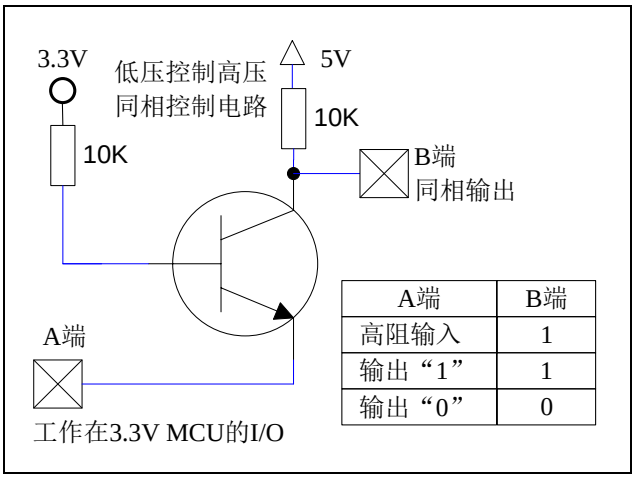


2、如是连接到 5V 器件的高阻输入，则可以直接相连，或串个 300 欧电阻相连。工作在 3.3V 的宽电压单片机的 I/O 可以工作在强推挽输出模式，或者工作在准双向口/弱上拉模式。

3、如果对方是 5V 输出，我方可工作在高阻输入模式，打开内部上拉电阻，外部串接一个锗二极管隔离，隔离高压 5V。外部电压高于单片机工作电压时锗二极管截止，I/O 口因内部上拉到高电平，所以读 I/O 口状态是高电平；外部电压为低时隔离的锗二极管导通，I/O 口被钳位在 0.3V，小于 0.8V 时单片机读 I/O 口状态是低电平。锗二极管可以使用 1N5817/1N5819，不要使用硅二极管。2.2V 以上是高电平，0.8V 以下是低电平。

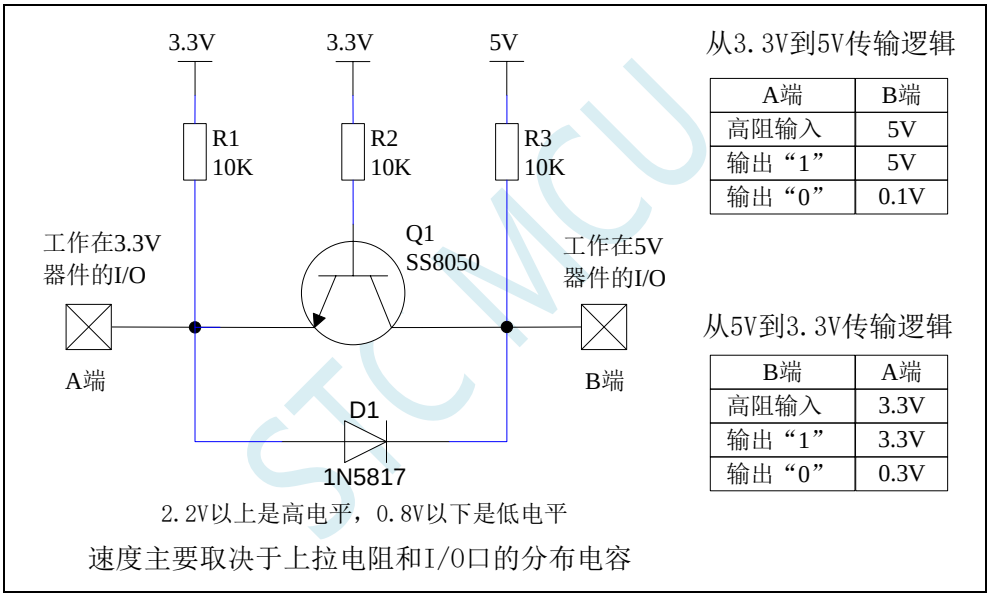


4、宽电压单片机工作在 3.3V 时，如需要控制 5V 器件，还可以用下图同相控制电路：



A端	B端
高阻输入	1
输出“1”	1
输出“0”	0

5、工作在 3.3V 的器件和工作在 5V 的器件打交道，还可以用如下【三极管+二极管】双向电路：



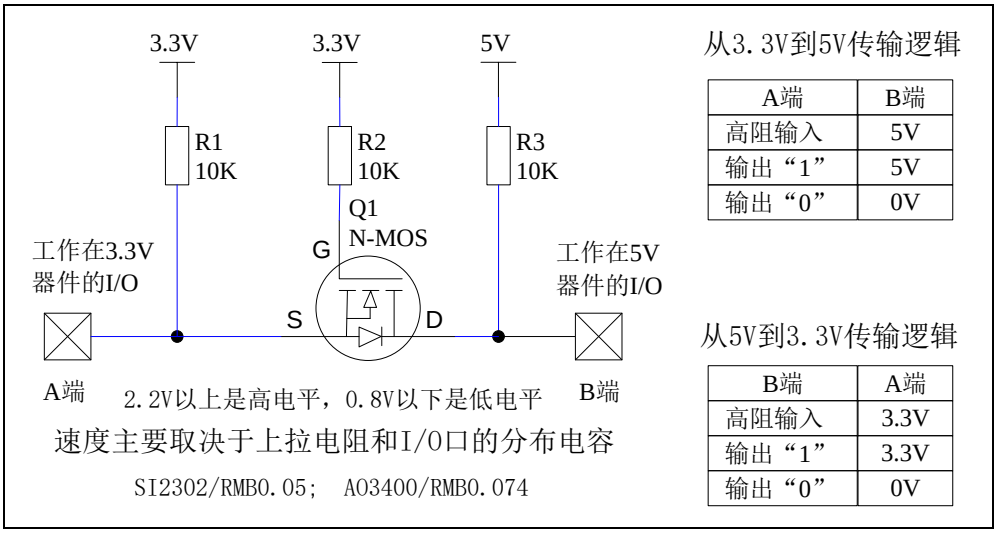
从3.3V到5V传输逻辑

A端	B端
高阻输入	5V
输出“1”	5V
输出“0”	0.1V

从5V到3.3V传输逻辑

B端	A端
高阻输入	3.3V
输出“1”	3.3V
输出“0”	0.3V

6、工作在 3.3V 的器件和工作在 5V 的器件打交道，还可以用如下 MOS 管双向电路：



从3.3V到5V传输逻辑

A端	B端
高阻输入	5V
输出“1”	5V
输出“0”	0V

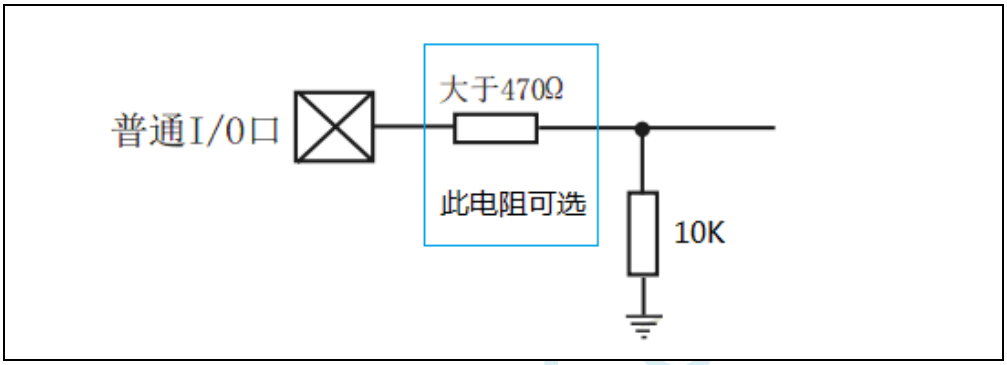
从5V到3.3V传输逻辑

B端	A端
高阻输入	3.3V
输出“1”	3.3V
输出“0”	0V

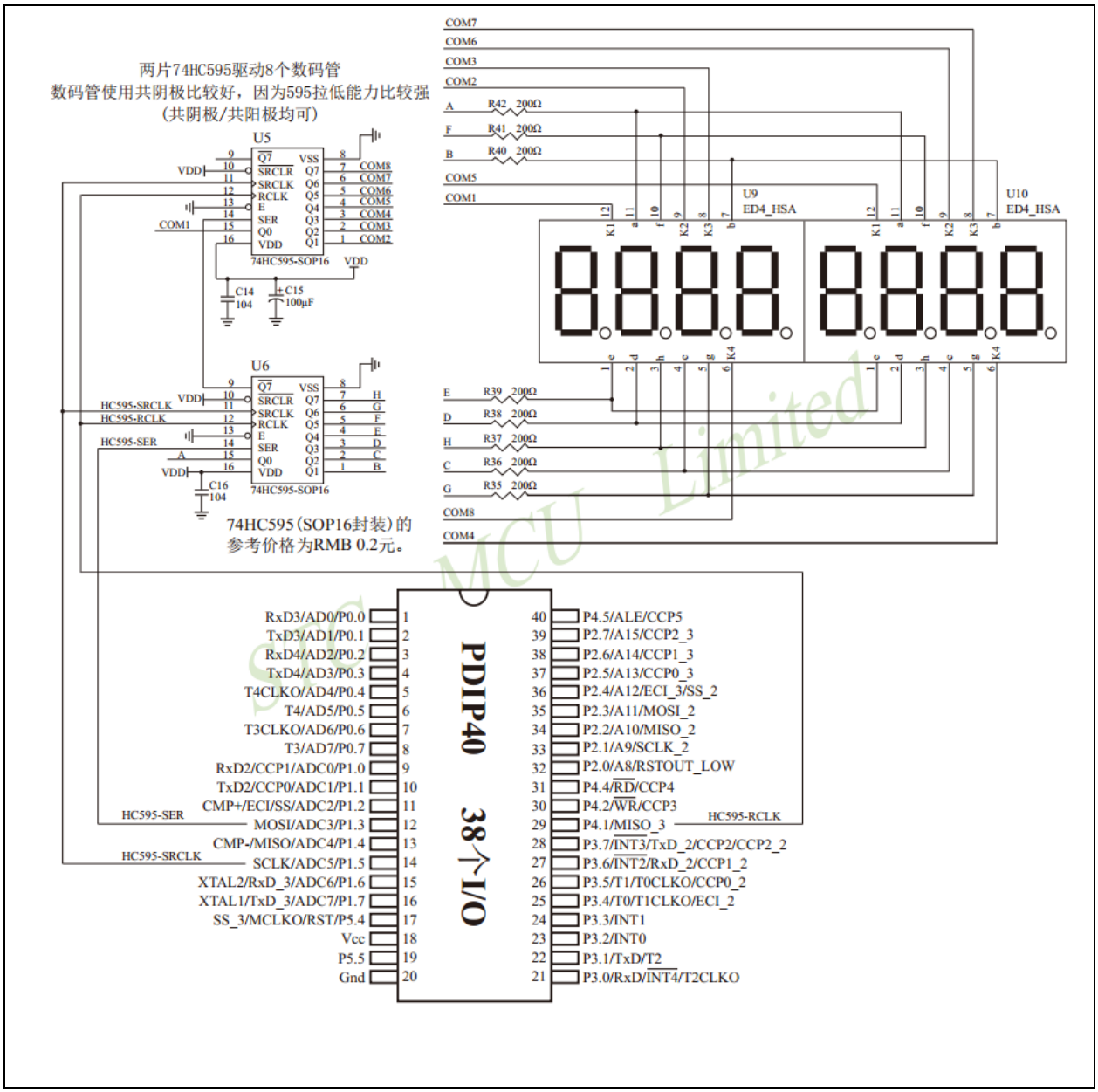
9.9 如何让 I/O 口上电复位时为低电平

传统 8051 单片机上电复位时普通 I/O 口为弱上拉(准双向口)高电平输出，而很多实际应用要求上电时某些 I/O 口为低电平输出，否则所控制的系统(如马达)就会误动作，新一代 STC8C 系列和 STC8H 系列单片机由于所有的 I/O 复位后是高阻输入(除 P3.0/P3.1 是传统的弱上拉)，加一个下拉电阻就可保证上电时为低电平，后续要改为高电平，只需要将 I/O 的模式改为强推挽输出，对外输出高电平即可。

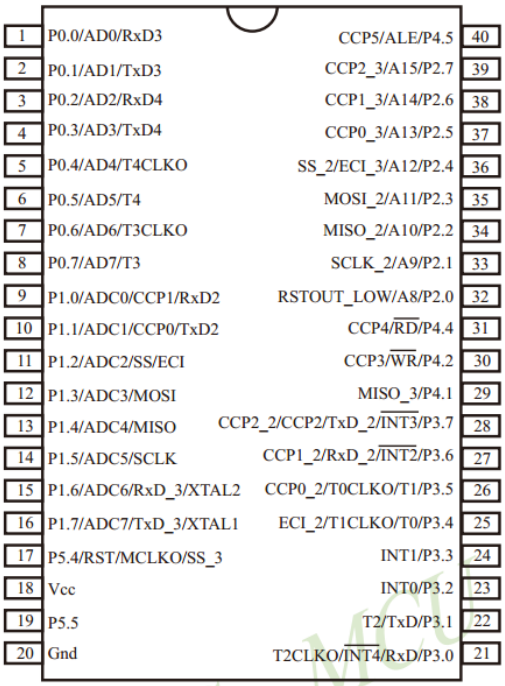
如下：在 STC 的单片机 I/O 口上加一个下拉电阻(10K 左右)，这样上电复位时，除了下载口 P3.0 和 P3.1 为弱上拉(准双向口)外，其他 I/O 口均为高阻输入模式，而外部有下拉电阻，所以该 I/O 口上电复位时外部为低电平。如果要将此 I/O 口驱动为高电平，可将此 I/O 口设置为强推挽输出，而强推挽输出时，I/O 口驱动电流可达 20mA，故肯定可以将该口驱动为高电平输出。

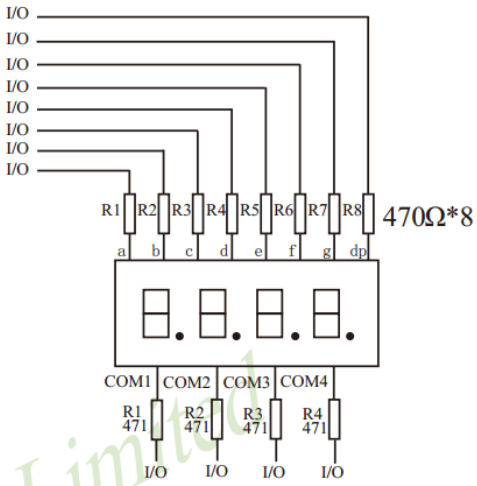


9.10 利用 74HC595 驱动 8 个数码管(串行扩展,3 根线)的线路图



9.11 I/O 口直接驱动 LED 数码管应用线路图

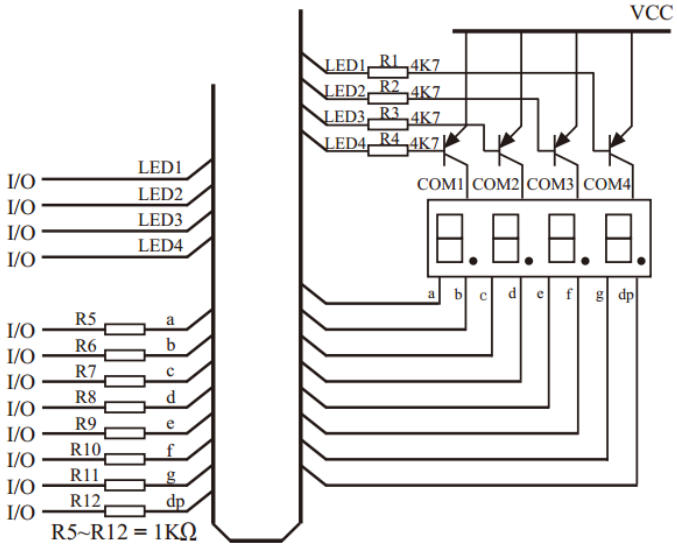




I/O口动态扫描驱动4个共阴极数码管参考电路图

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

I/O 口动态扫描驱动4个共阳极数码管参考电路图



STC MCU Limited

9.12 用 STC 系列 MCU 的 I/O 口直接驱动段码 LCD

当产品需要段码 LCD 显示时，如果使用不带 LCD 驱动器的 MCU，则需要外接 LCD 驱动 IC，这会增加成本。事实上，很多小项目，比如大量的小家电，需要显示的段码不多，常见的是 4 个 8 带小数点或时钟的冒号“:”，这样如果使用 IO 口直接扫描显示，则会降低成本，工作更可靠。

但是，本方案不合适驱动太多的段（ 占用 IO 太多），也不合适非常低功耗的场合（驱动会有几百 uA 电流）。

段码 LCD 驱动简单原理：如图 1 所示。

LCD 是一种特殊的液态晶体，在电场的作用下晶体的排列方向会发生扭转，因而改变其透光性，从而可以看到显示内容。LCD 有一个扭转电压阀值，当 LCD 两端电压高于此阀值时，显示内容，低于此阀值时，不显示。通常 LCD 有 3 个参数：工作电压、DUTY（对应 COM 数）和 BIAS（即偏压，对应阀值），比如 3.0V、1/4 DUTY、1/3 BIAS，表示 LCD 显示电压为 3.0V，4 个 COM，阀值大约是 1.5V，当加在某段 LCD 两端电压为 3.0V 时显示，而加 1.0V 时不显示。但是 LCD 对于驱动电压的反应不是很敏感的，比如加 2V 时，可能会微弱显示，这就是通常说的“鬼影”。所以要保证驱动显示时，要大于阀值电压比较多，而不显示时，要用比阀值小比较多的电压。

注意：LCD 要用交流驱动，其两端不能加直流电压，否则时间稍长就会损坏，所以要保证加在 LCD 两端的驱动电压的平均电压为 0。LCD 使用时分割扫描法，任何时候一个 COM 扫描有效，另外的 COM 处于无效状态。

驱动 1/4Duty 1/2BIAS 3V 的方案电路见图 1，LCD 扫描原理见图 3，MCU 为 3.0V 或 3.3V 工作，并且每个 COM 都串一个 20K 电阻接到一个电容 C1，RC 滤波后得到一个中点电压 1/2VDD。在轮到某个 COM 扫描时，连接的 IO 设置成推挽输出，其余 COM 设置成高阻，如果与本 COM 连接的 SEG 不显示，则 SEG 输出与 COM 同相，如果显示，则反相。扫描完后，这个 COM 的 IO 就设置成高阻。每个 COM 通过 20K 电阻连接到电容 C1 上的 1/2VDD 电压，而 SEG 根据是否显示输出高低电平，这样加在 LCD 段上的电压，显示时是+-VDD，不显示时是+-1/2VDD，保证了 LCD 两端平均直流电压为 0。

驱动 1/4Duty 1/3BIAS 3V 的方案电路见图 4，LCD 扫描原理见图 5，MCU 为 5V 工作，SEG 接的 IO 通过电阻分压输出 1.5V、3.5V，COM 接的 IO 通过电阻分压输出 0.5V、2.5V（高阻时）、4.5V，分压电阻公共点接到一个电容 C1，RC 滤波后得到一个中点电压 1/2VDD。在轮到某个 COM 扫描时，设置成推挽输出，如果与本 COM 连接的 SEG 不显示，则 SEG 输出与 COM 同相，如果显示，则反相。扫描完后，这个 COM 的 IO 就设置成高阻，这样这个 COM 就通过 47K 电阻连接到 2.5V 电压，而 SEG 根据是否显示输出高低电平，这样加在 LCD 上的电压，显示时是+-3.0V，不显示时是+-1.0V，完全满足 LCD 的扫描要求。

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

图 1：驱动 1/4Duty 1/2BIAS 3V LCD 的电路

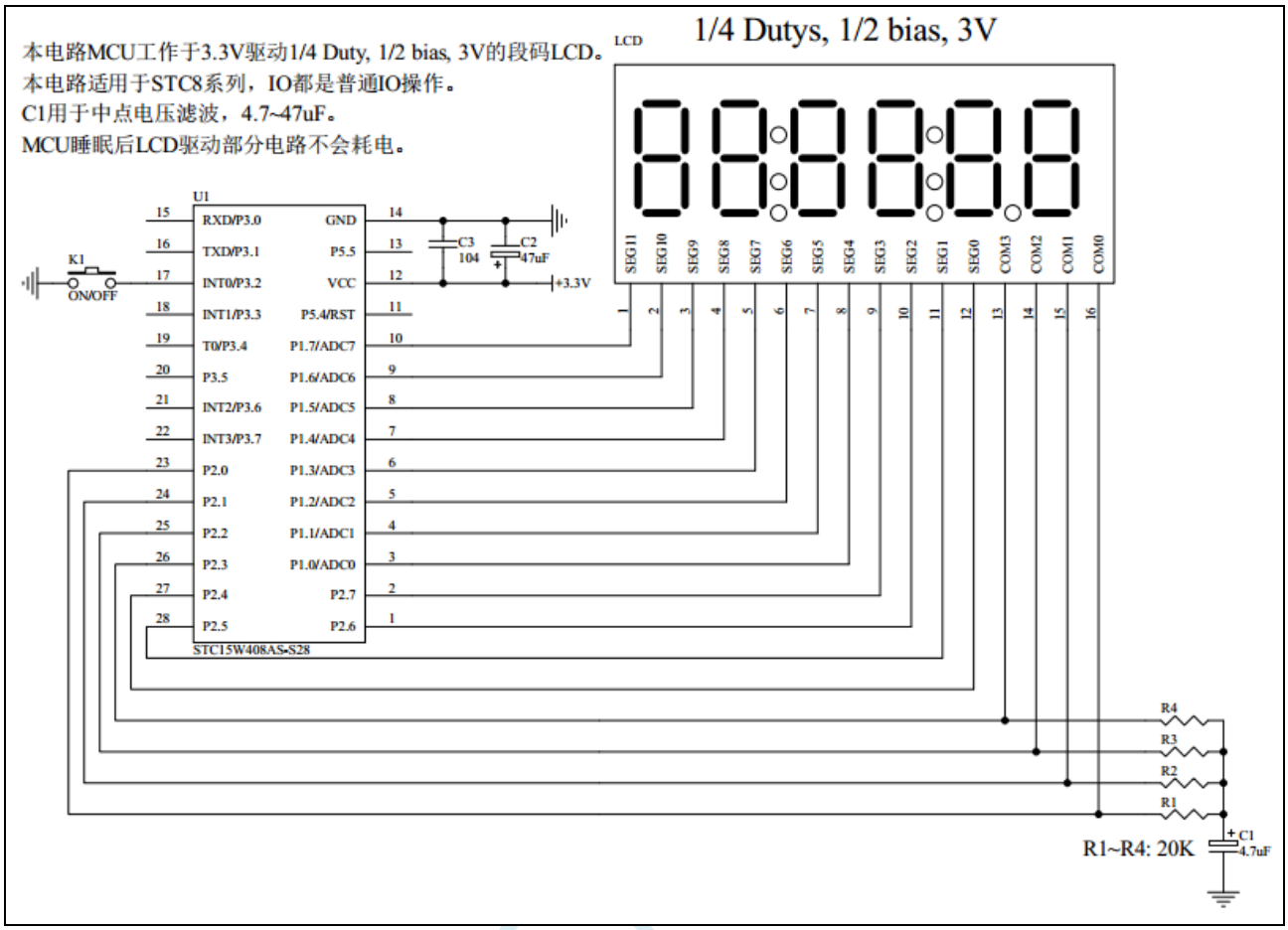


图 2：段码名称图

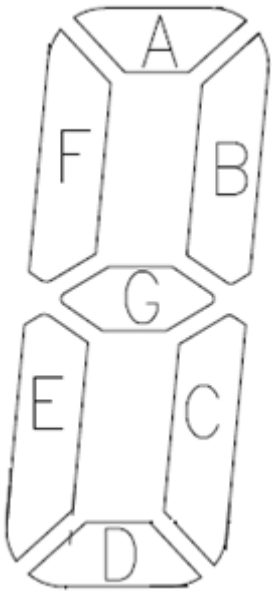


图 3：1/4Duty 1/2BIAS 扫描原理图

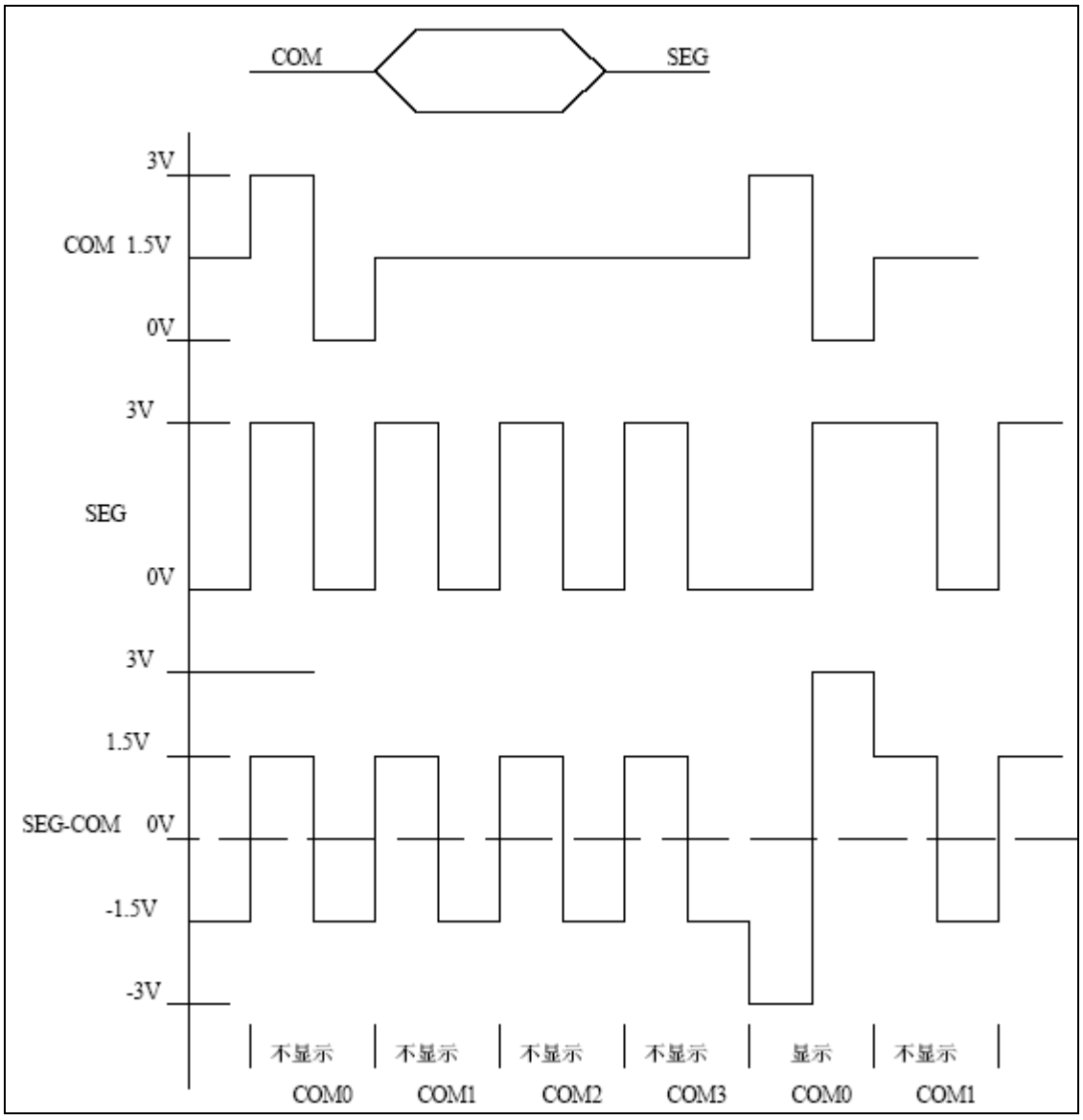


图 4：驱动 1/4Duty 1/3BIAS 3V LCD 的电路

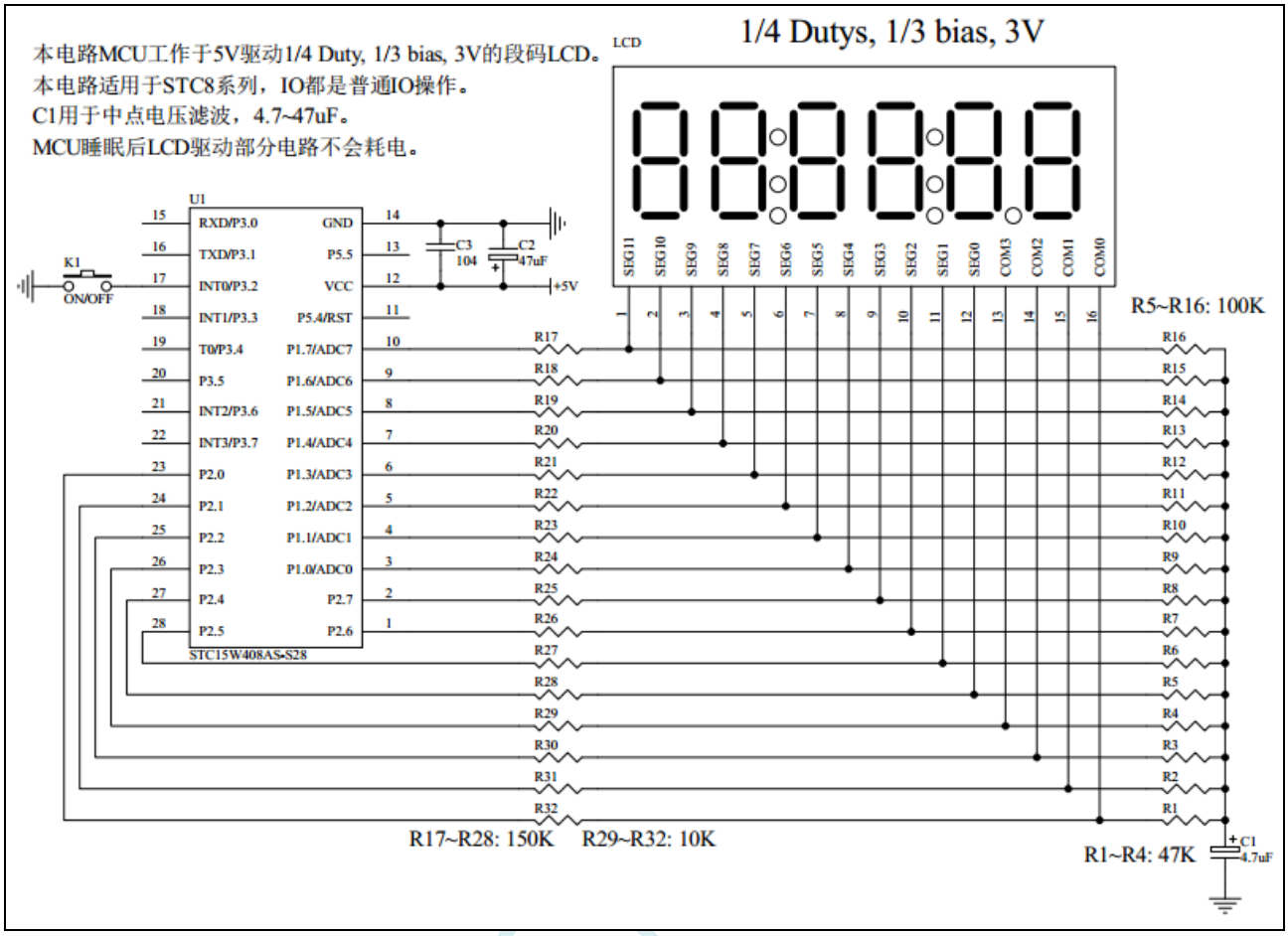
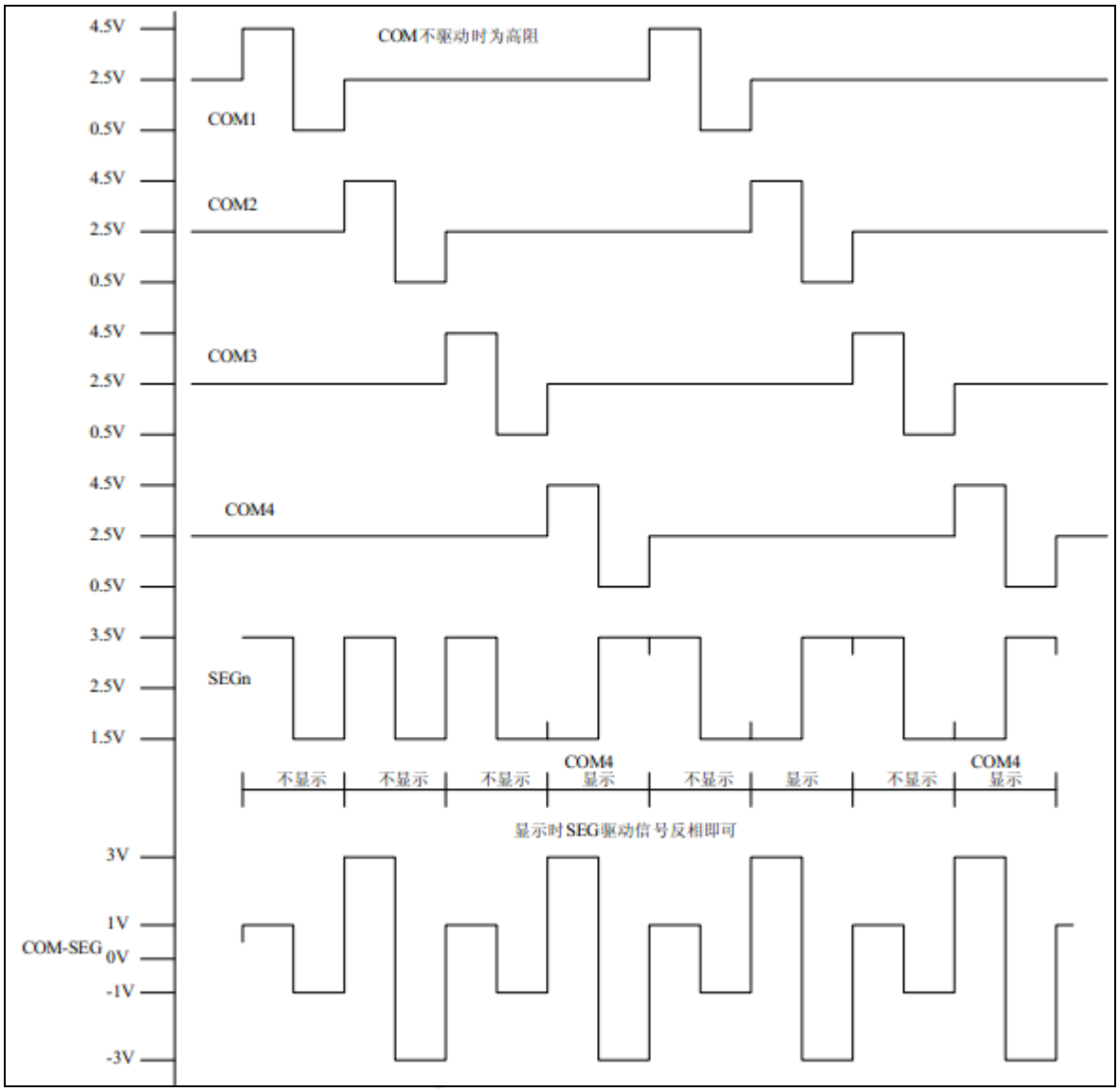


图 5：1/4Duty 1/3BIAS 扫描原理图



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

图 6：LCD 真值表和显存影射表

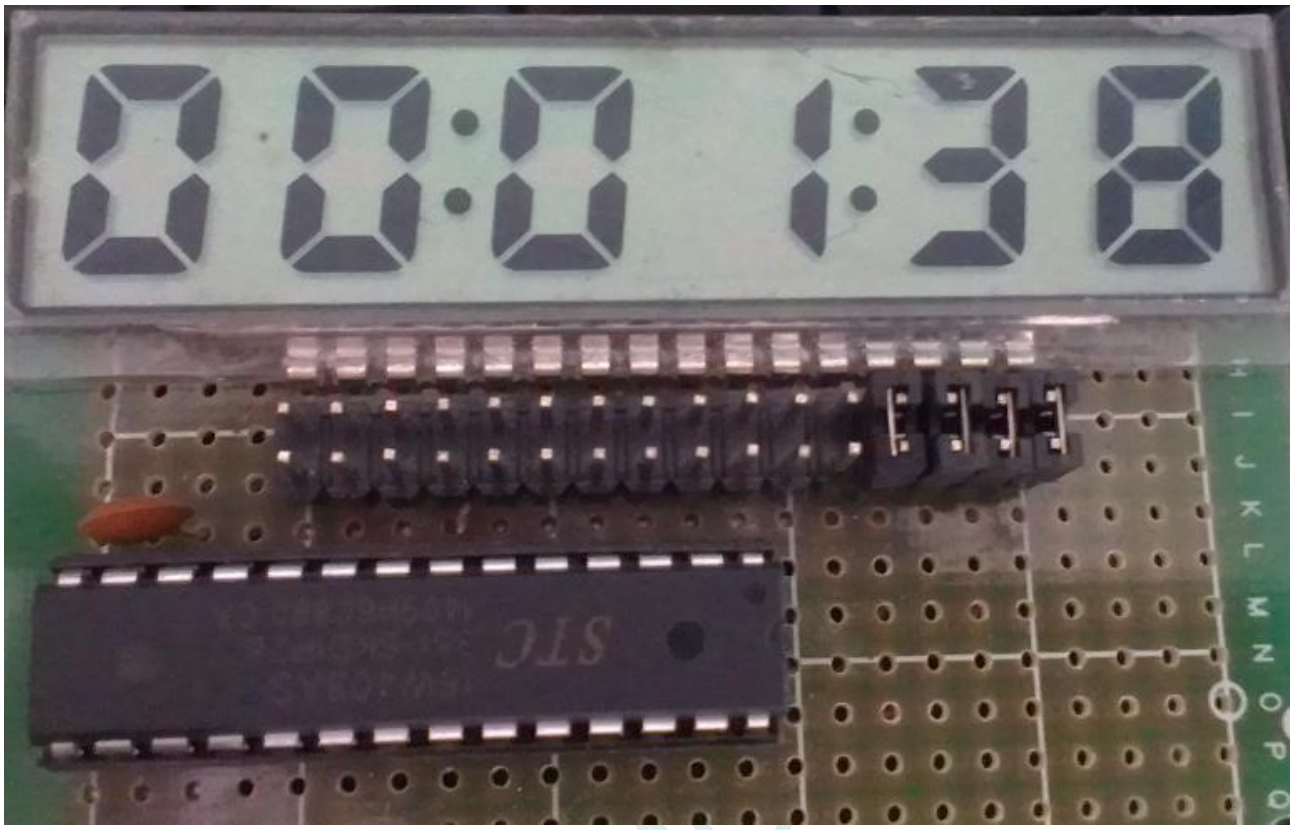
LCD真值表：

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

显存影射表：

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

图 7：驱动效果照片



本 LCD 扫描程序仅需要两个函数：

1、 LCD 段码扫描函数　void　LCD_scan(void)

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

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

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

另外，用宏来显示、熄灭或闪烁冒号或小数点。

汇编代码

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

;*************************************************************
P0M1        DATA        0x93
P0M0        DATA        0x94
P1M1        DATA        0x91
P1M0        DATA        0x92
P2M1        DATA        0x95
P2M0        DATA        0x96
P3M1        DATA        0xB1
P3M0        DATA        0xB2
P4M1        DATA        0xB3
P4M0        DATA        0xB4
```

```
P5M1        DATA    0xC9
P5M0        DATA    0xC
P6M1        DATA    0xCB
P6M0        DATA    0xCC
P7M1        DATA    0xE1
P7M0        DATA    0xE2
AUXR        DATA    0x8E
INTCLKO     DATA    0x8F
IE2         DATA    0xAF
P4          DATA    0xC0
T2H         DATA    0xD6
T2L         DATA    0xD7

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

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

LCD_buff    DATA    40H             ;40H~47H

;*******************************************************
            ORG     0000H
            LJMP    F_Main

            ORG     000BH
            LJMP    F_Timer0_Interrupt

;*******************************************************
            ORG     0100H
F_Main:
            CLR     A
            MOV     P3M1, A         ;设置为准双向口
            MOV     P3M0, A
            MOV     P5M1, A         ;设置为准双向口
            MOV     P5M0, A

            MOV     P1M1, #0        ; segment 设置为推挽输出
            MOV     P1M0, #0ffh
            ANL     P2M1, #NOT 0f0h ; segment 设置为推挽输出
            ORL     P2M0, #0f0h
            ORL     P2M1, #00fH     ;全部COM 输出高阻, COM 为中点电压
            ANL     P2M0, #0f0H
            MOV     SP, #0D0H
            MOV     PSW, #0
            USING   0               ;选择第 0 组 R0~R7
```

```
;*************************************************************
                MOV     R2, #8
                MOV     R0, #LCD_buff
L_ClearLcdRam:
                MOV     @R0, #0
                INC     R0
                DJNZ    R2, L_ClearLcdRam

                LCALL   F_Timer0_init
                SETB    EA

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

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

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

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

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

                CPL     B_Second
                JNB     B_Second, L_Main_Loop

                INC     second
                MOV     A, second
                CJNE    A, #60, L_Main_Load
                MOV     second, #0              ; 1 分钟到
                INC     minute
                MOV     A, minute
                CJNE    A, #60, L_Main_Load
                MOV     minute, #0;
                INC     hour
                MOV     A, hour
                CJNE    A, #24, L_Main_Load
                MOV     hour, #0                ;24 小时到
L_Main_Load:
                LCALL   F_LoadRTC               ;显示时间
                LJMP    L_Main_Loop

;*************************************************************

F_Timer0_init:
                CLR     TR0                     ; 停止计数
                ANL     TMOD, #0f0H
                SETB    ET0                     ; 允许中断
                ORL     TMOD, #0                ; 工作模式, 0: 16 位自动重装
```

```
        ANL     INTCLKO, #NOT 0x01      ; 不输出时钟
        ORL     AUXR, #0x80             ; 1T mode
        MOV     TH0, #HIGH (-22118)     ; 2ms
        MOV     TL0, #LOW  (-22118)     ;
        SETB    TR0                     ; 开始运行
        RET

;**********************************************************
F_Timer0_Interrupt:     ;Timer0 1ms 中断函数
        PUSH    PSW                     ;PSW 入栈
        PUSH    ACC                     ;ACC 入栈
        PUSH    AR0
        PUSH    AR7
        PUSH    DPH
        PUSH    DPL

        LCALL   F_LCD_scan
        SETB    B_2ms

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

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

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

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

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

        MOV     R6, #5                  ;LCD_load(5,second/10);
        MOV     A, second
```

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

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

            RET

;**********************************************************
T_COM:
            DB      008H, 004H, 002H, 001H

F_LCD_scan:
            MOV     A, scan_index       ;j = scan_index >> 1;
            CLR     C
            RRC     A
            MOV     R7, A               ;R7 = j
            ADD     A, #LCD_buff
            MOV     R0, A               ;R0 = LCD_buff[j]
            ORL     P2M1, #00fH         ;全部 COM 输出高阻, COM 为中点电压
            ANL     P2M0, #0f0H

            MOV     A, scan_index
            JNB     ACC.0, L_LCD_Scan2  ;if(scan_index & 1)   //反相扫描
            MOV     A, @R0              ;P1 = ~LCD_buff[j];
            CPL     A
            MOV     P1, A
            MOV     A, R0               ;P2 = ~(LCD_buff[j|4] & 0xf0);
            ADD     A, #4
            MOV     R0, A
            MOV     A, @R0
            ANL     A, #0f0H
            CPL     A
            MOV     P2, A
            SJMP    L_LCD_Scan3

L_LCD_Scan2:                            ;正相扫描
            MOV     A, @R0              ;P1 = LCD_buff[j];
            MOV     P1, A
            MOV     A, R0               ;P2 = (LCD_buff[j|4] & 0xf0);
            ADD     A, #4
            MOV     R0, A
            MOV     A, @R0
            ANL     A, #0f0H
            MOV     P2, A

L_LCD_Scan3:
            MOV     DPTR, #T_COM        ;某个 COM 设置为推挽输出
            MOV     A, R7
            MOVC    A, @A+DPTR
            ORL     P2M0, A
            CPL     A
            ANL     P2M1, A
```

```
                INC         scan_index              ;if(++scan_index == 8)      scan_index = 0;
                MOV         A, scan_index
                CJNE        A, #8, L_QuitLcdScan
                MOV         scan_index, #0

L_QuitLcdScan:
                RET

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

;****************** 对第 1~6 数字装载显示函数 算法简单 ***************************
F_LCD_load:                                         ;R6 为第几个数字，为 1~6，R7 为要显示的数字
                MOV         DPTR, #T_Display        ;i = t_display[dat];
                MOV         A, R7
                MOVC        A, @A+DPTR
                MOV         B, A                    ;要显示的数字

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

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

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

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

L_NotLoadChar1:
                CJNE        A, #2, L_NotLoadChar2
                MOV         R0,#LCD_buff
```

```
        MOV     A, @R0
        MOV     C, B.3          ;D
        MOV     ACC.4, C
        MOV     @R0, A

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

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

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

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

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

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

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

```
                MOV     ACC.3, C
                MOV     @R0, A
                RET

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

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

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

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

L_NotLoadChar4:
                CJNE    A, #5, L_NotLoadChar5
                MOV     R0,#LCD_buff+4
                MOV     A, @R0
                MOV     C, B.3          ;D
                MOV     ACC.6, C
                MOV     @R0, A

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

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

STC MCU

```
            MOV     @R0, A

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

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

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

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

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

            E                       N                       D
```

C 语言代码

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

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

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

sfr AUXR = 0x8e;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;

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

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

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

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

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

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

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

#define P1n_standard(bitn)      P1M1 &= ~(bitn), P1M0 &= ~(bitn)
#define P1n_push_pull(bitn)     P1M1 &= ~(bitn), P1M0 |= (bitn)
#define P1n_pure_input(bitn)    P1M1 |= (bitn), P1M0 &= ~(bitn)
#define P1n_open_drain(bitn)    P1M1 |= (bitn), P1M0 |= (bitn)

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

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

```
/*********************本地函数声明*************************/
void LCD_load(u8 n,u8 dat);
void LCD_scan(void);
void LoadRTC(void);
void delay_ms(u8 ms);

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

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

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

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

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

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

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

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

```
                }
            }

            if(!P32)                                //键按下，准备睡眠
            {
                LCD_CLR_2M;                         //显示时分间隔:
                LCD_CLR_4M;                         //显示分秒间隔:
                LCD_load(1,DIS_BLACK);
                LCD_load(2,DIS_BLACK);
                LCD_load(3,0);
                LCD_load(4,0x0F);
                LCD_load(5,0x0F);
                LCD_load(6,DIS_BLACK);

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

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

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

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

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

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

/********************* Timer0 中断函数************************/
void timer0_int (void) interrupt 1
{
    LCD_scan();
```

```
    B_2ms = 1;
}

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

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

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

/******************** 对第1~6 数字装载显示函数 *****************************/
void  LCD_load(u8 n, u8 dat)                            //n 为第几个数字，dat 为要显示的数字
{
    u8 code t_display[]={                               //标准字库
        // 0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
        0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x77,0x7C,0x39,0x5E,0x79,0x71,
        //black  -
        0x00,0x40
    };
    u8 code T_LCD_mask[4]   = {~0xc0,~0x30,~0x0c,~0x03};
    u8 code T_LCD_mask4[4] = {~0x40,~0x10,~0x04,~0x01};
    u8 i,k;
    u8 *p;

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

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

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

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

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

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

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

STC MCU

10 指令系统

10.1 寻址方式

寻址方式是每一种计算机的指令集中不可缺少的部分。寻址方式规定了数据的来源和目的地。对不同的程序指令，来源和目的地的规定也会不同。在 STC 单片机中的寻址方式可概括为：

- 立即寻址
- 直接寻址
- 间接寻址
- 寄存器寻址
- 相对寻址
- 变址寻址
- 位寻址

10.1.1 立即寻址

立即寻址也称立即数，它是在指令操作数中直接给出参加运算的操作数，其指令格式如下：

如：MOV A, #70H

这条指令的功能是将立即数 70H 传送到累加器 A 中。

10.1.2 直接寻址

在直接寻址方式中，指令操作数域给出的是参加运算操作数地址。直接寻址方式只能用来表示特殊功能寄存器、内部数据寄存器和位地址空间。其中特殊功能寄存器和位地址空间只能用直接寻址方式访问。

如：ANL 70H, #48H

表示 70H 单元中的数与立即数 48H 相“与”，结果存放在 70H 单元中。其中 70H 为直接地址，表示内部数据存储器 RAM 中的一个单元。

10.1.3 间接寻址

间接寻址采用 R0 或 R1 前添加“@”符号来表示。例如，假设 R1 中的数据是 40H，内部数据存储器 40H 单元所包含的数据为 55H，那么如下指令：

MOV A, @R1

把数据 55H 传送到累加器。

10.1.4 寄存器寻址

寄存器寻址是对选定的工作寄存器 R7~R0、累加器 A、通用寄存器 B、地址寄存器和进位 C 中的数进行操作。其中寄存器 R7~R0 由指令码的低 3 位表示，ACC、B、DPTR 及进位位 C 隐含在指令码中。因此，寄存器寻址也包含一种隐含寻址方式。

寄存器工作区的选择由程序状态字寄存器 PSW 中的 RS1、RS0 来决定。指令操作数指定的寄存器均指当前工作区中的寄存器。

如：INC R0 ;(R0)+1 → R0

10.1.5 相对寻址

相对寻址是将程序计数器 PC 中的当前值与指令第二字节给出的数相加，其结果作为转移指令的转移地址。转移地址也称为转移目的地址，PC 中的当前值称为基地址，指令第二字节给出的数称为偏移量。由于目的地址是相对于 PC 中的基地址而言，所以这种寻址方式称为相对寻址。偏移量为带符号的数，所能表示的范围为+127～–128。这种寻址方式主要用于转移指令。

如：JC 80H ;C=1 跳转

表示若进位位 C 为 0，则程序计数器 PC 中的内容不改变，即不转移。若进位位 C 为 1，则以 PC 中的当前值为基地址，加上偏移量 80H 后所得到的结果作为该转移指令的目的地址。

10.1.6 变址寻址

在变址寻址方式中，指令操作数指定一个存放变址基值的变址寄存器。变址寻址时，偏移量与变址基值相加，其结果作为操作数的地址。变址寄存器有程序计数器 PC 和地址寄存器 DPTR。

如：MOVC A, @A+DPTR

表示累加器 A 为偏移量寄存器，其内容与地址寄存器 DPTR 中的内容相加，其结果作为操作数的地址，取出该单元中的数送入累加器 A。

10.1.7 位寻址

位寻址是指对一些内部数据存储器 RAM 和特殊功能寄存器进行位操作时的寻址。在进行位操作时，借助于进位位 C 作为位操作累加器，指令操作数直接给出该位的地址，然后根据操作码的性质对该位进行位操作。位地址与字节直接寻址中的字节地址形式完全一样，主要由操作码加以区分，使用时应注意。

如：MOV C, 20H ;片内位单元位操作型指令

10.2 指令表

助记符		指令说明	字节	时钟
ADD	A,Rn	寄存器内容加到累加器	1	1
ADD	A,direct	直接地址单元的数据加到累加器	2	1
ADD	A,@Ri	间接地址单元的数据加到累加器	1	1
ADD	A,#data	立即数加到累加器	2	1
ADDC	A,Rn	寄存器带进位加到累加器	1	1
ADDC	A,direct	直接地址单元的数据带进位加到累加器	2	1
ADDC	A,@Ri	间接地址单元的数据带进位加到累加器	1	1
ADDC	A,#data	立即数带进位加到累加器	2	1
SUBB	A,Rn	累加器带借位减寄存器内容	1	1
SUBB	A,direct	累加器带借位减直接地址单元的内容	2	1
SUBB	A,@Ri	累加器带借位减间接地址单元的内容	1	1
SUBB	A,#data	累加器带借位减立即数	2	1
INC	A	累加器加1	1	1
INC	Rn	寄存器加1	1	1
INC	direct	直接地址单元加1	2	1
INC	@Ri	间接地址单元加1	1	1
DEC	A	累加器减1	1	1
DEC	Rn	寄存器减1	1	1
DEC	direct	直接地址单元减1	2	1
DEC	@Ri	间接地址单元减1	1	1
INC	DPTR	地址寄存器DPTR加1	1	1

MUL	AB	A乘以B, B存放高字节, A存放低字节	1	2
DIV	AB	A除以B, B存放余数, A存放商	1	6
DA	A	累加器十进制调整	1	3
ANL	A,Rn	累加器与寄存器相与	1	1
ANL	A,direct	累加器与直接地址单元相与	2	1
ANL	A,@Ri	累加器与间接地址单元相与	1	1
ANL	A,#data	累加器与立即数相与	2	1
ANL	direct,A	直接地址单元与累加器相与	2	1
ANL	direct,#data	直接地址单元与立即数相与	3	1
ORL	A,Rn	累加器与寄存器相或	1	1
ORL	A,direct	累加器与直接地址单元相或	2	1
ORL	A,@Ri	累加器与间接地址单元相或	1	1
ORL	A,#data	累加器与立即数相或	2	1
ORL	direct,A	直接地址单元与累加器相或	2	1
ORL	direct,#data	直接地址单元与立即数相或	3	1
XRL	A,Rn	累加器与寄存器相异或	1	1
XRL	A,direct	累加器与直接地址单元相异或	2	1
XRL	A,@Ri	累加器与间接地址单元相异或	1	1
XRL	A,#data	累加器与立即数相异或	2	1
XRL	direct,A	直接地址单元与累加器相异或	2	1
XRL	direct,#data	直接地址单元与立即数相异或	3	1
CLR	A	累加器清0	1	1
CPL	A	累加器取反	1	1
RL	A	累加器循环左移	1	1
RLC	A	累加器带进位循环左移	1	1
RR	A	累加器循环右移	1	1
RRC	A	累加器带进位循环右移	1	1
SWAP	A	累加器高低半字节交换	1	1
CLR	C	清零进位位	1	1
CLR	bit	清0直接地址位	2	1
SETB	C	置1进位位	1	1
SETB	bit	置1直接地址位	2	1
CPL	C	进位位求反	1	1
CPL	bit	直接地址位求反	2	1
ANL	C,bit	进位位和直接地址位相与	2	1
ANL	C,/bit	进位位和直接地址位的反码相与	2	1
ORL	C,bit	进位位和直接地址位相或	2	1

ORL	C,/bit	进位位和直接地址位的反码相或	2	1
MOV	C,bit	直接地址位送入进位位	2	1
MOV	bit,C	进位位送入直接地址位	2	1
MOV	A,Rn	寄存器内容送入累加器	1	1
MOV	A,direct	直接地址单元中的数据送入累加器	2	1
MOV	A,@Ri	间接地址中的数据送入累加器	1	1
MOV	A,#data	立即数送入累加器	2	1
MOV	Rn,A	累加器内容送入寄存器	1	1
MOV	Rn,direct	直接地址单元中的数据送入寄存器	2	1
MOV	Rn,#data	立即数送入寄存器	2	1
MOV	direct,A	累加器内容送入直接地址单元	2	1
MOV	direct,Rn	寄存器内容送入直接地址单元	2	1
MOV	direct,direct	直接地址单元中的数据送入另一个直接地址单元	3	1
MOV	direct,@Ri	间接地址中的数据送入直接地址单元	2	1
MOV	direct,#data	立即数送入直接地址单元	3	1
MOV	@Ri,A	累加器内容送间接地址单元	1	1
MOV	@Ri,direct	直接地址单元数据送入间接地址单元	2	1
MOV	@Ri,#data	立即数送入间接地址单元	2	1
MOV	DPTR,#data16	16位立即数送入数据指针	3	1
MOVC	A,@A+DPTR	以DPTR为基地址变址寻址单元中的数据送入累加器	1	4
MOVC	A,@A+PC	以PC为基地址变址寻址单元中的数据送入累加器	1	3
MOVX	A,@Ri	扩展地址(8位地址)的内容送入累加器A中	1	3[1]
MOVX	A,@DPTR	扩展RAM(16位地址)的内容送入累加器A中	1	2[1]
MOVX	@Ri,A	将累加器A的内容送入扩展RAM(8位地址)中	1	3[1]
MOVX	@DPTR,A	将累加器A的内容送入扩展RAM(16位地址)中	1	2[1]
PUSH	direct	直接地址单元中的数据压入堆栈	2	1
POP	direct	栈底数据弹出送入直接地址单元	2	1
XCH	A,Rn	寄存器与累加器交换	1	1
XCH	A,direct	直接地址单元与累加器交换	2	1
XCH	A,@Ri	间接地址与累加器交换	1	1
XCHD	A,@Ri	间接地址的低半字节与累加器交换	1	1
ACALL	addr11	短调用子程序	2	3
LCALL	addr16	长调用子程序	3	3
RET		子程序返回	1	3
RETI		中断返回	1	3
AJMP	addr11	短跳转	2	3
LJMP	addr16	长跳转	3	3

SJMP	rel	相对跳转	2	3
JMP	@A+DPTR	相对于DPTR的间接跳转	1	4
JZ	rel	累加器为零跳转	2	1/3[2]
JNZ	rel	累加器非零跳转	2	1/3[2]
JC	rel	进位位为1跳转	2	1/3[2]
JNC	rel	进位位为0跳转	2	1/3[2]
JB	bit,rel	直接地址位为1则跳转	3	1/3[2]
JNB	bit,rel	直接地址位为0则跳转	3	1/3[2]
JBC	bit,rel	直接地址位为1则跳转, 该位清0	3	1/3[2]
CJNE	A,direct,rel	累加器与直接地址单元不相等跳转	3	2/3[3]
CJNE	A,#data,rel	累加器与立即数不相等跳转	3	1/3[2]
CJNE	Rn,#data,rel	寄存器与立即数不相等跳转	3	2/3[3]
CJNE	@Ri,#data,rel	间接地址单元与立即数不相等跳转	3	2/3[3]
DJNZ	Rn,rel	寄存器减1后非零跳转	2	2/3[3]
DJNZ	direct,rel	直接地址单元减1后非零跳转	3	2/3[3]
NOP		空操作	1	1

[1]:访问外部扩展 RAM 时，指令的执行周期与寄存器 BUS_SPEED 中的 SPEED[2:0]位有关

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

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

[4]:上表中每条指令的执行时钟数为实际测量的数据，下面指令描述部分为原始规格，部分指令的执行时钟数可能与实际不符，请以上表的数据为准。

10.3 指令详解（中文）

ACALL addr11

功能： 绝对调用

说明： ACALL 指令实现无条件调用位于 addr11 参数所表示地址的子例程。在执行该指令时，首先将 PC 的值增加 2，即使得 PC 指向 ACALL 的下一条指令，然后把 16 位 PC 的低 8 位和高 8 位依次压入栈，同时把栈指针两次加 1。然后，把当前 PC 值的高 5 位、ACALL 指令第 1 字节的 7~5 位和第 2 字节组合起来，得到一个 16 位目的地址，该地址即为即将调用的子例程的入口地址。要求该子例程的起始地址必须与紧随 ACALL 之后的指令处于同 1 个 2KB 的程序存储页中。ACALL 指令在执行时不会改变各个标志位。

举例： SP 的初始值为 07H，标号 SUBRTN 位于程序存储器的 0345H 地址处，如果执行位于地址 0123H 处的指令：

ACALL SUBRTN

那么 SP 变为 09H，内部 RAM 地址 08H 和 09H 单元的内容分别为 25H 和 01H，PC 值变为 0345H。

指令长度(字节)： 2

执行周期：3

二进制编码：

A10	A9	A8	1	0	0	0	1		A7	A6	A5	A4	A3	A2	A1	A0

注意：a10 a9 a8 是 11 位目标地址 addr11 的 A10~A8 位，a7 a6 a5 a4 a3 a2 a1 a0 是 addr11 的 A7~A0 位。

操作：ACALL

(PC)←(PC)+ 2

(SP)←(SP) + 1

((SP)) ←(PC_{7-0})

(SP)←(SP) + 1

((SP))←(PC_{15-8})

(PC_{10-0})←页码地址

ADD A, <src-byte>

功能：加法

说明：ADD 指令可用于完成把 src-byte 所表示的源操作数和累加器 *A* 的当前值相加。并将结果置于累加器 *A* 中。根据运算结果，若第 7 位有进位则置进位标志为 1，否则清零；若第 3 位有进位则置辅助进位标志为 1，否则清零。如果是无符号整数相加则进位置位，显示当前运算结果发生溢出。如果第 6 位有进位生成而第 7 位没有，或第 7 位有进位生成而第 6 位没有，则置 OV 为 1，否则 OV 被清零。在进行有符号整数的相加运算的时候，OV 置位表示两个正整数之和为一负数，或是两个负整数之和为一正数。

本类指令的源操作数可接受 4 种寻址方式：寄存器寻址、直接寻址、寄存器间接寻址和立即寻址。

举例：假设累加器 *A* 中的数据为 0C3H(11000011B)，R0 的值为 0AAH(10101010B)。执行如下指令：

ADD A, R0

累加器 *A* 中的结果为 6DH(01101101B)，辅助进位标志 AC 被清零，进位标志 C 和溢出标志 OV 被置 1。

ADD A, Rn

指令长度（字节）：1

执行周期：1

二进制编码：

0	0	1	0	1	r	r	r

操作：ADD

(A)← (A) + (Rn)

ADD A, direct

指令长度（字节）：2

执行周期：1

二进制编码：

0	0	1	0	0	1	0	1		direct address

操作：ADD

(A)← (A) + (direct)

ADD A, @Ri

指令长度（字节）：1

执行周期：1

二进制编码：

0	0	1	0	0	1	1	i

操作：ADD

(A)← (A) + ((Ri))

ADD A, #data

指令长度(字节)：　2

执行周期：　1

二进制编码：

0	0	1	0	0	1	0	0		immediate data

操作：　ADD

(A)← (A) + #data

ADDC A, <src-byte>

功能：　带进位的加法

说明：　执行 ADDC 指令时，把 src-byte 所代表的源操作数连同进位标志一起加到累加器 *A* 上，并将结果置于累加器 *A* 中。根据运算结果，若在第 7 位有进位生成，则将进位标志置 1，否则清零；若在第 3 位有进位生成，则置辅助进位标志为 1，否则清零。如果是无符号数整数相加，进位的置位显示当前运算结果发生溢出。

如果第 6 位有进位生成而第 7 位没有，或第 7 位有进位生成而第 6 位没有，则将 OV 置 1，否则将 OV 清零。在进行有符号整数相加运算的时候，OV 置位，表示两个正整数之和为一负数，或是两个负整数之和为一正数。

本类指令的源操作数允许 4 种寻址方式：寄存器寻址、直接寻址、寄存器间接寻址和立即寻址。

举例：　假设累加器 *A* 中的数据为 0C3H(11000011B)，*R*0 的值为 0AAH(10101010B)，进位标志为 1，执行如下指令：

ADDC A,R0

累加器 *A* 中的结果为 6EH(01101110B)，辅助进位标志 AC 被清零，进位标志 C 和溢出标志 OV 被置 1。

ADDC A, Rn

指令长度(字节)：　1

执行周期：　1

二进制编码：

0	0	1	1	1	r	r	r

操作：　ADDC

(A)← (A) + (C) + (Rn)

ADDC A, direct

指令长度(字节)：　2

执行周期：　1

二进制编码：

0	0	1	1	0	1	0	1		direct address

操作：　ADDC

(A)← (A) + (C) + (direct)

ADDC A, @Ri

指令长度(字节)：　1

执行周期：　1

二进制编码：

0	0	1	1	0	1	1	i

操作：　ADDC

(A)← (A) + (C) + ((Ri))

ADDC A, #data

指令长度(字节)：　2

执行周期：　1

二进制编码：

0	0	1	1	0	1	0	0		immediate data

操作：　ADDC

(A)← (A) + (C) + #data

AJMP addr11

功能：　绝对跳转

说明：　AJMP 指令用于将程序转到相应的目的地址去执行，该地址在程序执行过程之中产生，由 PC 值（两次递增之后）的高 5 位、操作码的 7~5 位和指令的第 2 字节连接形成。要求跳转的目的地址和 AJMP 指令的后一条指令的第 1 字节位于同一 2KB 的程序存储页内。

举例：　假设标号 JMPADR 位于程序存储器的 0123H，指令：

AJMP JMPADR

位于 0345H，执行完该指令后 PC 值变为 0123H。

指令长度(字节)：　2

执行周期：　3

二进制编码：

A10	A9	A8	0	0	0	0	1		A7	A6	A5	A4	A3	A2	A1	A0

注意：目的地址的 A10－A8＝a10~a8，A7－A0＝a7~a0。

操作：　AJMP

(PC)←(PC)+ 2

(PC_{10-0})←page address

ANL <dest-byte> , <src-byte>

功能：　对字节变量进行逻辑与运算

说明：　ANL 指令将由<dest-byte>和<src-byte>所指定的两个字节变量逐位进行逻辑与运算，并将运算结果存放在<dest-byte>所指定的目的操作数中。该指令的执行不会影响标志位。

两个操作数组合起来允许 6 种寻址模式。当目的操作数为累加器时，源操作数允许寄存器寻址、直接寻址、寄存器间接寻址和立即寻址。当目的操作数是直接地址时，源操作数可以是累加器或立即数。

注意：当该指令用于修改输出端口时，读入的原始数据来自于输出数据的锁存器而非输入引脚。

举例：　如果累加器的内容为 0C3H(11000011B)，寄存器 0 的内容为 55H(01010101B)，那么指令:

ANL A,R0

执行结果是累加器的内容变为 41H(01000001H)。

当目的操作数是可直接寻址的数据时，ANL 指令可用来把任何 RAM 单元或者硬件寄存器中的某些位清零。屏蔽字节将决定哪些位将被清零。屏蔽字节可能是常数，也可能是累加器在计算过程中产生。如下指令:

ANL Pl, #01110011B

将端口 1 的位 7、位 3 和位 2 清零。

ANL A, Rn

指令长度(字节)：　1

执行周期：　1

二进制编码：

0	1	0	1	1	r	r	r

操作：　ANL

(A)← (A) ∧ (Rn)

ANL A, direct

指令长度(字节)：　2

执行周期：　1

二进制编码：

0	1	0	1	0	1	0	1		direct address

操作：ANL

(A)← (A) ∧ (direct)

ANL A, @Ri

指令长度(字节)：1

执行周期：1

二进制编码：

0	1	0	1	0	1	1	i

操作：ANL

(A)← (A) ∧ ((Ri))

ANL A, #data

指令长度(字节)：2

执行周期：1

二进制编码：

0	1	0	1	0	1	0	0		immediate data

操作：ANL

(A)← (A) ∧ #data

ANL direct, A

指令长度(字节)：2

执行周期：1

二进制编码：

0	1	0	1	0	0	1	0		direct address

操作：ANL

(direct)←(direct) ∧ (A)

ANL direct, #data

指令长度(字节)：3

执行周期：1

二进制编码：

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

操作：ANL

(direct)←(direct) ∧ #data

ANL C, <src-bit>

功能：对位变量进行逻辑与运算

说明：如果 src-bit 表示的布尔变量为逻辑 0，清零进位标志位；否则，保持进位标志的当前状态不变。在汇编语言程序中，操作数前面的“/”符号表示在计算时需要先对被寻址位取反，然后才作为源操作数，但源操作数本身不会改变。该指令在执行时不会影响其他各个标志位。

源操作数只能采取直接寻址方式。

举例：下面的指令序列当且仅当 P1.0＝1、ACC.7＝1 和 OV＝0 时，将进位标志 C 置 1:

```
MOV C, P1.0          ; LOAD CARRY WITH INPUT PIN STATE
ANL C, ACC.7         ; AND CARRY WITH ACCUM. BIT.7
ANL C, /OV           ;AND WITH INVERSE OF OVERFLOW FLAG
```

ANL C, bit

指令长度(字节)：2

执行周期：1

二进制编码：

1	0	0	0	0	0	1	0		bit address

操作：ANL

(C) ←(C) ∧ (bit)

ANL C, /bit

指令长度(字节)：2

执行周期：1

二进制编码：

1	0	1	1	0	0	0	0		bit address

操作：ANL

(C)← (C) ∧($\overline{bit}$)

CJNE <dest-byte>, <src-byte>, rel

功能：若两个操作数不相等则转移

说明：CJNE 首先比较两个操作数的大小，如果二者不等则程序转移。目标地址由位于 CJNE 指令最后 1 个字节的有符号偏移量和 PC 的当前值（紧邻 CJNE 的下一条指令的地址）相加而成。如果目标操作数作为一个无符号整数，其值小于源操作数对应的无符号整数，那么将进位标志置 1，否则将进位标志清零。但操作数本身不会受到影响。

<dest-byte>和<src-byte>组合起来，允许 4 种寻址模式。累加器 *A* 可以与任何可直接寻址的数据或立即数进行比较，任何间接寻址的 RAM 单元或当前工作寄存器都可以和立即常数进行比较。

举例：设累加器 *A* 中值为 34H，*R7* 包含的数据为 56H。如下指令序列:

```
        CJNE R7,#60H, NOT_EQ
;       . . .     . . . . . .      ; R7 = 60H.
NOT_EQ: JC        REQ_LOW          ; IF R7 < 60H.
;       . . .     . . . . . .      ; R7 > 60H.
```

的第 1 条指令将进位标志置 1，程序跳转到标号 NOT_EQ 处。接下去，通过测试进位标志，可以确定 *R7* 是大于 60H 还是小于 60H。

假设端口 1 的数据也是 34H，那么如下指令：

```
WAIT: CJNE A,P1,WAIT
```

清除进位标志并继续往下执行，因为此时累加器的值也为 34H，即和 P1 口的数据相等。（如果 *P*l 端口的数据是其他的值，那么程序在此不停地循环，直到 *P*1 端口的数据变成 34H 为止。）

CJNE A, direct, rel

指令长度(字节)：3

执行周期：2 或 3

二进制编码：

1	0	1	1	0	1	0	1		direct address		rel. address

操作：

```
(PC) ←(PC) + 3
IF (A) < > (direct)
THEN
    (PC) ←(PC) +relative offset
IF (A) < (direct)
THEN
    (C) ← 1
ELSE
    (C) ← 0
```

CJNE A, #data, rel

指令长度(字节)：　3

执行周期：　1 或 3

二进制编码：

1	0	1	1	0	1	0	0		immediate data		rel. address

操作：　(PC) ←(PC) + 3

IF (A) < > *(data)*

THEN

　　(PC) ←(PC) +*relative offset*

IF (A) < *(data)*

THEN

　　(C) ← 1

ELSE

　　(C) ← 0

CJNE Rn, #data, rel

指令长度(字节)：　3

执行周期：　2 或 3

二进制编码：

1	0	1	1	1	r	r	r		immediate data		rel. address

操作：　(PC) ←(PC) + 3

IF (Rn) < > *(data)*

THEN

　　(PC) ←(PC) +*relative offset*

IF (Rn) < *(data)*

THEN

　　(C) ← 1

ELSE

　　(C) ← 0

CJNE @Ri, #data, rel

指令长度(字节)：　3

执行周期：　2 或 3

二进制编码：

1	0	1	1	0	1	1	i		immediate data		rel. address

操作：　(PC) ←(PC) + 3

IF (Ri) < > *(data)*

THEN

　　(PC) ←(PC) +*relative offset*

IF (Ri) < *(data)*

THEN

　　(C) ← 1

ELSE

　　(C) ← 0

CLR A

功能：　清除累加器

说明：　该指令用于将累加器 *A* 的所有位清零，不影响标志位。

举例：　假设累加器 *A* 的内容为 5CH(01011100B)，那么指令：

CLR A

执行后，累加器的值变为 00H(00000000B)。

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	1	0	0	1	0	0

操作： CLR

(A)←0

CLR bit

功能： 清零指定的位

说明： 将 bit 所代表的位清零，没有标志位会受到影响。CLR 可用于进位标志 C 或者所有可直接寻址的位。

举例： 假设端口 1 的数据为 5DH(01011101B)，那么指令：

CLR P1.2

执行后，Pl 端口被设置为 59H(01011001B)。

CLR C

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	0	0	0	0	1	1

操作： CLR

(C)←0

CLR bit

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	1	0	0	0	0	1	0		bit address

操作： CLR

(bit)←0

CPL A

功能： 累加器 *A* 求反

说明： 将累加器 *A* 的每一位都取反，即原来为 1 的位变为 0，原来为 0 的位变为 1。该指令不影响标志位。

举例： 设累加器 *A* 的内容为 5CH(01011100B)，那么指令：

CPL A

执行后，累加器的内容变成 0A3H（10100011B）。

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	1	1	0	1	0	0

操作： CPL

(A)← ($\bar{A}$)

CPL bit

STC MCU

功能： 将 bit 所表示的位求反

说明： 将 bit 变量所代表的位取反，即原来位为 1 的变为 0，原来为 0 的变为 1。没有标志位会受到影响。CPL 可用于进位标志 C 或者所有可直接寻址的位。

注意：如果该指令被用来修改输出端口的状态，那么 bit 所代表的数据是端口锁存器中的数据，而不是从引脚上输入的当前状态。

举例： 设 *P*1 端口的数据为 5BH(01011011B)，那么指令：

```
CPL P1.1
CPL P1.2
```

执行完后，*P*l 端口被设置为 5DH(01011101B)。

CPL C

指令长度（字节）： 1

执行周期： 1

二进制编码：

1	0	1	1	0	0	1	1

操作： CPL

(C)← ($\bar{C}$)

CPL bit

指令长度（字节）： 2

执行周期： 1

二进制编码：

1	0	1	1	0	0	1	0		bit address

操作： CPL

(bit)← ($\overline{bit}$)

DAA

功能： 在加法运算之后，对累加器 *A* 进行十进制调整

说明： DA 指令对累加器 *A* 中存放的由此前的加法运算产生的 8 位数据进行调整（ADD 或 ADDC 指令可以用来实现两个压缩 BCD 码的加法），生成两个 4 位的数字。

如果累加器的低 4 位（位 3~位 0）大于 9 (xxxx1010~xxxx 1111），或者加法运算后，辅助进位标志 AC 为 1，那么 DA 指令将把 6 加到累加器上，以在低 4 位生成正确的 BCD 数字。若加 6 后，低 4 位向上有进位，且高 4 位都为 1，进位则会一直向前传递，以致最后进位标志被置 1；但在其他情况下进位标志并不会被清零，进位标志会保持原来的值。

如果进位标志为 1，或者高 4 位的值超过 9 (1010xxxx~1111xxxx)，那么 DA 指令将把 6 加到高 4 位，在高 4 位生成正确的 BCD 数字，但不清除标志位。若高 4 位有进位输出，则置进位标志为 1，否则，不改变进位标志。进位标志的状态指明了原来的两个 BCD 数据之和是否大于 99，因而 DA 指令使得 CPU 可以精确地进行十进制的加法运算。注意，OV 标志不会受影响。

DA 指令的以上操作在一个指令周期内完成。实际上，根据累加器 *A* 和机器状态字 PSW 中的不同内容，DA 把 00H、06H、60H、66H 加到累加器 *A* 上，从而实现十进制转换。

注意：如果前面没有进行加法运算，不能直接用 DA 指令把累加器 *A* 中的十六进制数据转换为 BCD 数，此外，如果先前执行的是减法运算，DA 指令也不会有所预期的效果。

举例： 如果累加器中的内容为 56H（01010110B），表示十进制数 56 的 BCD 码，寄存器 3 的内容为 67H（01100111B），表示十进制数 67 的 BCD 码。进位标志为 1，则指令：

```
ADDC A,R3
```

DA A

先执行标准的补码二进制加法，累加器 *A* 的值变为 0BEH，进位标志和辅助进位标志被清零。

接着，DA 执行十进制调整，将累加器 *A* 的内容变为 24H（00100100B），表示十进制数 24 的 BCD 码，也就是 56、67 及进位标志之和的后两位数字。DA 指令会把进位标志置位 1，这表示在进行十进制加法时，发生了溢出。56、67 以及 1 的和为 124。

把 BCD 格式的变量加上 0lH 或 99H，可以实现加 1 或者减 1。假设累加器的初始值为 30H（表示十进制数 30），指令序列：

ADD A, #99H

DA A

将把进位 C 置为 1，累加器 *A* 的数据变为 29H，因为 30＋99＝129。加法和的低位数据可以看作减法运算的结果，即 30－1＝29。

指令长度（字节）： 1

执行周期： 3

二进制编码：

1	1	0	1	0	1	0	0

操作： DA

-contents of Accumulator are BCD

IF [[(A_{3-0}) > 9] V [(AC) = 1]]

THEN(A_{3-0}) ←(A_{3-0}) + 6

AND

IF [[(A_{7-4}) > 9] V [(C) = 1]]

THEN (A_{7-4})←(A_{7-4}) + 6

DEC byte

功能： 把 BYTE 所代表的操作数减 1

说明： BYTE 所代表的变量被减去 l。如果原来的值为 00H，那么减去 1 后，变成 0FFH。没有标志位会受到影响。该指令支持 4 种操作数寻址方式：累加器寻址、寄存器寻址、直接寻址和寄存器间接寻址。

注意：当 DEC 指令用于修改输出端口的状态时，BYTE 所代表的数据是从端口输出数据锁存器中获取的，而不是从引脚上读取的输入状态。

举例： 假设寄存器 0 的内容为 7FH（01111111B），内部 RAM 的 7EH 和 7FH 单元的内容分别为 00H 和 40H。则指令：

DEC @R0

DEC R0

DEC @R0

执行后，寄存器 0 的内容变成 7EH，内部 RAM 的 7EH 和 7FH 单元的内容分别变为 0FFH 和 3FH。

DEC A

指令长度（字节）： 1

执行周期： 1

二进制编码：

0	0	0	1	0	1	0	0

操作： DEC

(A)← (A) −1

DEC Rn

指令长度(字节)：　1

执行周期：　1

二进制编码：

0	0	0	1	1	r	r	r

操作：　DEC

(Rn)←(Rn) - 1

DEC direct

指令长度(字节)：　2

执行周期：　1

二进制编码：

0	0	0	1	0	1	0	1		Direct address

操作：　DEC

(direct)←(direct) - 1

DEC @Ri

指令长度(字节)：　1

执行周期：　1

二进制编码：

0	0	0	1	0	1	1	i

操作：　DEC

((Ri))←((Ri)) - 1

DIV AB

功能：　除法

说明：　DIV 指令把累加器 *A* 中的 8 位无符号整数除以寄存器 *B* 中的 8 位无符号整数，并将商置于累加器 *A* 中，余数置于寄存器 *B* 中。进位标志 C 和溢出标志 OV 被清零。

例外：如果寄存器 *B* 的初始值为 00H（即除数为 0），那么执行 DIV 指令后，累加器 *A* 和寄存器 *B* 中的值是不确定的，且溢出标志 OV 将被置位。但在任何情况下，进位标志 C 都会被清零。

举例：　假设累加器的值为 251 (0FBH 或 11111011B)，寄存器 *B* 的值为 18 (12H 或 00010010B)。

则指令：

DIV AB

执行后，累加器的值变成 13 (0DH 或 00001101B)，寄存器 *B* 的值变成 17 (11H 或 00010001B)，正好符合 251＝13×18＋17。进位和溢出标志都被清零。

指令长度(字节)：　1

执行周期：　6

二进制编码：

1	0	0	0	0	1	0	0

操作：　DIV

$(A)_{15\text{-}8}(B)_{7\text{-}0} \leftarrow (A)/(B)$

DJNZ <byte>, <rel-addr>

功能：　减 1，若非 0 则跳转

说明：　DJNZ 指令首先将第 1 个操作数所代表的变量减 1，如果结果不为 0，则转移到第 2 个操作数所指定的地址处去执行。如果第 1 个操作数的值为 00H，则减 1 后变为 0FFH。该指令不影响标志位。跳转目标地址的计算：首先将 PC 值加 2（即指向下一条指令的首字节），然后将第 2 操作数表示的有符号的相对偏移量加到 PC 上去即可。

byte 所代表的操作数可采用寄存器寻址或直接寻址。

注意：如果该指令被用来修改输出引脚上的状态，那么 byte 所代表的数据是从端口输出数据锁存器中获取的，而不是直接读取引脚。

举例：　假设内部 RAM 的 40H、50H 和 60H 单元分别存放着 01H、70H 和 15H，则指令：

```
DJNZ 40H, LABEL_1
DJNZ 50H, LABEL_2
DJNZ 60H, LABEL_3
```

执行之后，程序将跳转到标号 LABEL2 处执行，且相应的 3 个 RAM 单元的内容变成 00H、6FH 和 15H。之所以第 1 个跳转没被执行，是因为减 1 后其结果为 0，不满足跳转条件。

使用 DJNZ 指令可以方便地在程序中实现指定次数的循环，此外用一条指令就可以在程序中实现中等长度的时间延迟（2~512 个机器周期）。指令序列：

```
        MOV R2,#8
TOOOLE: CPL P1.7
        DJNZ R2, TOOGLE
```

将使得 P1.7 的电平翻转 8 次，从而在 P1.7 产生 4 个脉冲，每个脉冲将持续 3 个机器周期，其中 2 个为 DJNZ 指令的执行时间，1 个为 CPL 指令的执行时间。

DJNZ Rn, rel

指令长度（字节）：　2

执行周期：　2 或 3

二进制编码：

1	1	0	1	1	r	r	r		rel. address

操作：

```
DJNZ
(PC) ←(PC) + 2
(Rn) ←(Rn) – 1
IF (Rn) > 0 or (Rn) < 0
    THEN
        (PC) ←(PC)+ rel
```

DJNZ direct, rel

指令长度（字节）：　3

执行周期：　2 或 3

二进制编码：

1	1	0	1	0	1	0	1		direct address		rel. address

操作：

```
DJNZ
(PC) ←(PC) + 2
(direct) ←(direct) – 1
IF (direct) > 0 or (direct) < 0
    THEN
        (PC) ←(PC)+ rel
```

INC <byte>

功能：　加 1

说明：　INC 指令将<byte>所代表的数据加 1。如果原来的值为 FFH，则加 1 后变为 00H，该指令不影响标志位。支持 3 种寻址模式：寄存器寻址、直接寻址、寄存器间接寻址。

注意：如果该指令被用来修改输出引脚上的状态，那么 byte 所代表的数据是从端口输出数据锁存器中获取的，而不是直接读的引脚。

举例：　假设寄存器 0 的内容为 7EH(01111110B)，内部 RAM 的 7E 单元和 7F 单元分别存放着 0FFH 和 40H，则指令序列：

INC @R0

INC R0

INC @R0

执行完毕后，寄存器 0 的内容变为 7FH，而内部 RAM 的 7EH 和 7FH 单元的内容分别变成 00H 和 41H。

INC A

指令长度（字节）： 1

执行周期： 1

二进制编码：

0	0	0	0	0	1	0	0

操作： INC

(A) ←(A)+1

INC Rn

指令长度（字节）： 1

执行周期： 1

二进制编码：

0	0	0	0	1	r	r	r

操作： INC

(Rn) ←(Rn)+1

INC direct

指令长度（字节）： 2

执行周期： 1

二进制编码：

0	0	0	0	0	1	0	1		direct address

操作： INC

(direct) ←(direct)+1

INC @Ri

指令长度（字节）： 1

执行周期： 1

二进制编码：

0	0	0	0	0	1	1	i

操作： INC

((Ri))←((Ri)) + 1

INC DPTR

功能： 数据指针加 1

说明： 该指令实现将 DPTR 加 1 功能。需要注意的是，这是 16 位的递增指令，低位字节 DPL 从 FFH 增加 1 之后变为 00H，同时进位到高位字节 DPH。该操作不影响标志位。

该指令是唯一 1 条 16 位寄存器递增指令。

举例： 假设寄存器 DPH 和 DPL 的内容分别为 12H 和 0FEH，则指令序列：

IINC DPTR

INC DPTR

INC DPTR

执行完毕后，DPH 和 DPL 变成 13H 和 01H。

指令长度（字节）： 1

执行周期： 1

二进制编码：

1	0	1	0	0	0	1	1

操作： INC

(DPTR) ← (DPTR) + 1

JB bit, rel

功能： 若位数据为 1 则跳转

说明： 如果 bit 代表的位数据为 1，则跳转到 rel 所指定的地址处去执行；否则，继续执行下一条指令。跳转的目标地址按照如下方式计算：先增加 PC 的值，使其指向下一条指令的首字节地址，然后把 rel 所代表的有符号的相对偏移量（指令的第 3 个字节）加到 PC 上去，新的 PC 值即为目标地址。该指令只是测试相应的位数据，但不会改变其数值，而且该操作不会影响标志位。

举例： 假设端口 1 的输入数据为 11001010B，累加器的值为 56H（01010110B）。则指令：

JB P1.2, LABEL1

JB ACC.2, LABEL2

将导致程序转到标号 LABEL2 处去执行。

指令长度(字节)： 3

执行周期： 1 或 3

二进制编码：

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

操作： JB

(PC) ← (PC) + 3

IF (bit) = 1

THEN

(PC) ← (PC)+ rel

JBC bit, rel

功能： 若位数据为 1 则跳转并将其清零

说明： 如果 bit 代表的位数据为 1，则将其清零并跳转到 rel 所指定的地址处去执行。如果 bit 代表的位数据为 0，则继续执行下一条指令。跳转的目标地址按照如下方式计算：先增加 PC 的值，使其指向下一条指令的首字节地址，然后把 rel 所代表的有符号的相对偏移量（指令的第 3 个字节）加到 PC 上去，新的 PC 值即为目标地址，而且该操作不会影响标志位。注意：如果该指令被用来修改输出引脚上的状态，那么 byte 所代表的数据是从端口输出数据锁存器中获取的，而不是直接读取引脚。

举例： 假设累加器的内容为 56H(01010110B)，则指令序列：

JBC ACC.3, LABEL1

JBC ACC.2, LABEL2

将导致程序转到标号 LABEL2 处去执行，且累加器的内容变为 52H（01010010B）。

指令长度(字节)： 3

执行周期： 1 或 3

二进制编码：

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

操作： JB

(PC) ← (PC) + 3

IF (bit) = 1

THEN

(bit) ← 0

(PC) ← (PC)+ rel

JC rel

功能： 若进位标志为 1，则跳转

说明： 如果进位标志为 1，则程序跳转到 rel 所代表的地址处去执行；否则，继续执行下面的指令。跳转的目标地址按照如下方式计算：先增加 PC 的值，使其指向紧接 JC 指令的下一条指令的首地址，然后把 rel 所代表的有符号的相对偏移量（指令的第 2 个字节）加到 PC 上去，新的 PC 值即为目标地址。该操作不会影响标志位。

举例： 假设进位标志此时为 0，则指令序列：

```
JC LABEL1
CPL C
JC LABEL2
```

执行完毕后，进位标志变成 1，并导致程序跳转到标号 LABEL2 处去执行。

指令长度(字节)： 2

执行周期： 1 或 3

二进制编码：

0	1	0	0	0	0	0	0		rel. address

操作：

```
JC
(PC) ← (PC) + 2
IF (C) = 1
    THEN
        (PC) ← (PC)+ rel
```

JMP @A+DPTR

功能： 间接跳转

说明： 把累加器 *A* 中的 8 位无符号数据和 16 位的数据指针的值相加，其和作为下一条将要执行的指令的地址，传送给程序计数器 PC。执行 16 位的加法时，低字节 DPL 的进位会传到高字节 DPH。累加器 *A* 和数据指针 DPTR 的内容都不会发生变化。不影响任何标志位。

举例： 假设累加器 *A* 中的值是偶数（从 0 到 6）。下面的指令序列将使得程序跳转到位于跳转表 JMP_TBL 的 4 条 AJMP 指令中的某一条去执行：

```
        MOV DPTR, #JMP_TBL
        JMP @A+DPTR
JMP-TBL: AJMP LABEL0
        AJMP LABEL1
        AJMP LABEL2
        AJMP LABEL3
```

如果开始执行上述指令序列时，累加器 *A* 中的值为 04H，那么程序最终会跳转到标号 LABEL2 处去执行。

注意：AJMP 是一个 2 字节指令，因而在跳转表中，各个跳转指令的入口地址依次相差 2 个字节。

指令长度(字节)： 1

执行周期： 4

二进制编码：

0	1	1	1	0	0	1	1

操作：

```
JMP
(PC) ← (A) + (DPTR)
```

JNB bit, rel

功能：　如果 bit 所代表的位不为 1 则跳转

说明：　如果 bit 所表示的位为 0，则转移到 rel 所代表的地址去执行；否则，继续执行下一条指令。跳转的目标地址如此计算：先增加 PC 的值，使其指向下一条指令的首字节地址，然后把 rel 所代表的有符号的相对偏移量（指令的第 3 个字节）加到 PC 上去，新的 PC 值即为目标地址。该指令只是测试相应的位数据，但不会改变其数值，而且该操作不会影响标志位。

举例：　假设端口 1 的输入数据为 11001010B，累加器的值为 56H（01010110B）。则指令序列：

```
JNB P1.3, LABEL1
JNB ACC.3, LABEL2
```

执行后将导致程序转到标号 LABEL2 处去执行。

指令长度（字节）：　3

执行周期：　1 或 3

二进制编码：

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

操作：

```
JNB
(PC) ←(PC) + 3
IF (bit) = 0
   THEN (PC) ← (PC) + rel
```

JNC rel

功能：　若进位标志非 1 则跳转

说明：　如果进位标志为 0，则程序跳转到 rel 所代表的地址处去执行；否则，继续执行下面的指令。跳转的目标地址按照如下方式计算：先增加 PC 的值加 2，使其指向紧接 JNC 指令的下一条指令的地址，然后把 rel 所代表的有符号的相对偏移量（指令的第 2 个字节）加到 PC 上去，新的 PC 值即为目标地址。该操作不会影响标志位。

举例：　假设进位标志此时为 1，则指令序列：

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

执行完毕后，进位标志变成 0，并导致程序跳转到标号 LABEL2 处去执行。

指令长度（字节）：　2

执行周期：　1 或 3

二进制编码：

0	1	0	1	0	0	0	0		rel. address

操作：

```
JNC
(PC) ←(PC) + 2
IF (C) = 0
   THEN (PC) ← (PC) + rel
```

JNZ rel

功能：　如果累加器的内容非 0 则跳转

说明：　如果累加器 A 的任何一位为 1，那么程序跳转到 rel 所代表的地址处去执行，如果各个位

都为 0，继续执行下一条指令。跳转的目标地址按照如下方式计算：先把 PC 的值增加 2，然后把 rel 所代表的有符号的相对偏移量（指令的第 2 个字节）加到 PC 上去，新的 PC 值即为目标地址。操作过程中累加器的值不会发生变化，不会影响标志位。

举例：　设累加器的初始值为 00H，则指令序列：

```
JNZ LABEL1
INC A
JNZ LAEEL2
```

执行完毕后，累加器的内容变成 01H，且程序将跳转到标号 LABEL2 处去执行。

指令长度(字节)：　2

执行周期：　1 或 3

二进制编码：

0	1	1	1	0	0	0	0		rel. address

操作：　JNZ

(PC) ←(PC) + 2

IF (A) ≠ 0

　THEN (PC) ← (PC) + rel

JZ rel

功能：　若累加器的内容为 0 则跳转

说明：　如果累加器 A 的任何一位为 0，那么程序跳转到 rel 所代表的地址处去执行，如果各个位都为 0，继续执行下一条指令。跳转的目标地址按照如下方式计算：先把 PC 的值增加 2，然后把 rel 所代表的有符号的相对偏移量（指令的第 2 个字节）加到 PC 上去，新的 PC 值即为目标地址。操作过程中累加器的值不会发生变化，不会影响标志位。

举例：　设累加器的初始值为 01H，则指令序列：

```
JZ LABEL1
DEC A
JZ LAEEL2
```

执行完毕后，累加器的内容变成 00H，且程序将跳转到标号 LABEL2 处去执行。

指令长度(字节)：　2

执行周期：　1 或 3

二进制编码：

0	1	1	0	0	0	0	0		rel. address

操作：　JZ

(PC) ←(PC) + 2

IF (A) = 0

　THEN (PC) ← (PC) + rel

LCALL addr16

功能：　长调用

说明：　LCALL 用于调用 addr16 所指地址处的子例程。首先将 PC 的值增加 3，使得 PC 指向紧随 LCALL 的下一条指令的地址，然后把 16 位 PC 的低 8 位和高 8 位依次压入栈（低位字节在先），同时把栈指针加 2。然后再把 LCALL 指令的第 2 字节和第 3 字节的数据分别装入 PC 的高位字节 DPH 和低位字节 DPL，程序从新的 PC 所对应的地址处开始执行。因而子例程可以位于 64KB 程序存储空间的任何地址处。该操作不影响标志位。

举例：　栈指针的初始值为 07H，标号 SUBRTN 被分配的程序存储器地址为 1234H。则执行如

下位于地址 0123H 的指令后：

LCALL SUBRTN

栈指针变成 09H，内部 RAM 的 08H 和 09H 单元的内容分别为 26H 和 01H，且 PC 的当前值为 1234H。

指令长度（字节）：3

执行周期：3

二进制编码：

0	0	0	1	0	0	1	0		addr15-addr8		addr7-addr0

操作：LCALL

(PC) ←(PC) + 3

(SP) ←(SP) + 1

((SP)) ←(PC_{7-0})

(SP) ←(SP) + 1

((SP)) ←(PC_{15-8})

(PC) ← $addr_{15-0}$

LJMP addr16

功能：长跳转

说明：LJMP 使得 CPU 无条件跳转到 addr16 所指的地址处执行程序。把该指令的第 2 字节和第 3 字节分别装入程序计数器 PC 的高位字节 DPH 和低位字节 DPL。程序从新 PC 值对应的地址处开始执行。该 16 位目标地址可位于 64KB 程序存储空间的任何地址处。该操作不影响标志位。

举例：假设标号 JMPADR 被分配的程序存储器地址为 1234H。则位于地址 1234H 的指令：

LJMP JMPADR

执行完毕后，PC 的当前值变为 1234H。

指令长度（字节）：3

执行周期：3

二进制编码：

0	0	0	0	0	0	1	0		addr15-addr8		addr7-addr0

操作：LJMP

(PC) ← $addr_{15-0}$

MOV <dest-byte> , <src-byte>

功能：传送字节变量

说明：将第 2 操作数代表字节变量的内容复制到第 1 操作数所代表的存储单元中去。该指令不会改变源操作数，也不会影响其他寄存器和标志位。

MOV 指令是迄今为止使用最灵活的指令，源操作数和目的操作数组合起来，寻址方式可达 15 种。

举例：假设内部 RAM 的 30H 单元的内容为 40H，而 40H 单元的内容为 10H。端口 1 的数据为 11001010B（0CAH）。则指令序列：

```
MOV R0, #30H        ; R0< = 30H
MOV A, @R0          ; A < = 40H
MOV R1, A           ; R1 < = 40H
MOV B, @Rl          ; B < = 10H
MOV @Rl, Pl         ; RAM (40H) < = 0CAH
```

MOV P2, P1 ; P2 #0CAH

执行完毕后，寄存器 0 的内容为 30H，累加器和寄存器 1 的内容都为 40H，寄存器 B 的内容为 10H，RAM 中 40H 单元和 P2 口的内容均为 0CAH。

MOV A,Rn

指令长度（字节）： 1

执行周期： 1

二进制编码：

1	1	1	0	1	r	r	r

操作： MOV

(A) ←(Rn)

***MOV A,direct**

指令长度（字节）： 2

执行周期： 1

二进制编码：

1	1	1	0	0	1	0	1		direct address

操作： MOV

(A)← (direct)

注意：**MOV A, ACC** 是无效指令。

MOV A,@Ri

指令长度（字节）： 1

执行周期： 1

二进制编码：

1	1	1	0	0	1	1	i

操作： MOV

(A) ←((Ri))

MOV A,#data

指令长度（字节）： 2

执行周期： 1

二进制编码：

0	1	1	1	0	1	0	0		immediate data

操作： MOV

(A)←#data

MOV Rn, A

指令长度（字节）： 1

执行周期： 1

二进制编码：

1	1	1	1	1	r	r	r

操作： MOV

(Rn) ←(A)

MOV Rn,direct

指令长度（字节）： 2

执行周期： 1

二进制编码：

1	0	1	0	1	r	r	r		direct address

操作： MOV

(Rn)← (direct)

MOV Rn,#data

指令长度（字节）： 2

执行周期： 1

二进制编码：

0	1	1	1	1	r	r	r		immediate data

操作： MOV

(Rn)←#data

MOV direct, A

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	1	1	1	0	1	0	1		direct address

操作： MOV

(direct)← (A)

MOV direct, Rn

指令长度(字节)： 2

执行周期： 2

二进制编码：

1	0	0	0	1	r	r	r		direct address

操作： MOV

(direct)← (Rn)

MOV direct, direct

指令长度(字节)： 3

执行周期： 1

二进制编码：

1	0	0	0	0	1	0	1		dir.addr. (src)		dir.addr. (dest)

操作： MOV

(direct)← (direct)

MOV direct, @Ri

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	0	0	0	1	1	i		direct address

操作： MOV

(direct)←((Ri))

MOV direct, #data

指令长度(字节)： 3

执行周期： 1

二进制编码：

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

操作： MOV

(direct)←#data

MOV @Ri, A

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	1	1	0	1	1	i

操作： MOV

((Ri))←(A)

MOV @Ri, direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	1	0	0	1	1	i		direct address

操作： MOV

((Ri))←(direct)

MOV @Ri, #data

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	1	1	0	1	1	i		immediate data

操作： MOV

((Ri))←#data

MOV <dest-bit> , <src-bit>

功能： 传送位变量

说明： 将<src-bit>代表的布尔变量复制到<dest-bit>所指定的数据单元中去，两个操作数必须有一个是进位标志，而另外一个是可直接寻址的位。本指令不影响其他寄存器和标志位。

举例： 假设进位标志 C 的初值为 1，端口 P3 中的数据是 11000101B，端口 1 的数据被设置为 35H(00110101B)。则指令序列：

MOV P1.3, C

MOV C, P3.3

MOV P1.2, C

执行后，进位标志被清零，端口 1 的数据变为 39H（00111001B）。

MOV C, bit

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	1	0	0	0	1	0		bit address

操作： MOV

(C)← (bit)

MOV bit, C

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	0	0	1	0	0	1	0		bit address

操作： MOV

(bit)← (C)

MOV DPTR, #data 16

功能： 将 16 位的常数存放到数据指针

说明： 该指令将 16 位常数传递给数据指针 DPTR。16 位的常数包含在指令的第 2 字节和第 3 字节中。其中 DPH 中存放的是#data16 的高字节，而 DPL 中存放的是#data16 的低字节。不影响标志位。

该指令是唯一一条能一次性移动 16 位数据的指令。

举例： 指令：

MOV DPTR, #1234H

将立即数 1234H 装入数据指针寄存器中。DPH 的值为 12H，DPL 的值为 34H。

指令长度(字节)： 3

执行周期： 1

二进制编码：

1	0	0	1	0	0	0	0		immediate data15-8		immediate data7-0

操作： MOV

(DPTR) ← #data$_{15-0}$

DPH DPL ← #data$_{15-8}$ #data$_{7-0}$

MOVC A , @A+ <base-reg>

功能： 把程序存储器中的代码字节数据（常数数据）转送至累加器 *A*

说明： MOVC 指令将程序存储器中的代码字节或常数字节传送到累加器 *A*。被传送的数据字节的地址是由累加器中的无符号 8 位数据和 16 位基址寄存器（DPTR 或 PC）的数值相加产生的。如果以 PC 为基址寄存器，则在累加器内容加到 PC 之前，PC 需要先增加到指向紧邻 MOVC 之后的语句的地址；如果是以 DPTR 为基址寄存器，则没有此问题。在执行 16 位的加法时，低 8 位产生的进位会传递给高 8 位。本指令不影响标志位。

举例： 假设累加器 *A* 的值处于 0~4 之间，如下子例程将累加器 *A* 中的值转换为用 DB 伪指令（定义字节）定义的 4 个值之一：

```
REL-PC: INC A
        MOVC A, @A+PC
        RET
        DB 66H
        DB 77H
        DB 88H
        DB 99H
```

如果在调用该子例程之前累加器的值为 01H，执行完该子例程后，累加器的值变为 77H。MOVC 指令之前的 INC A 指令是为了在查表时越过 RET 而设置的。如果 MOVC 和表格之间被多个代码字节所隔开，那么为了正确地读取表格，必须将相应的字节数预先加到累加器 *A* 上。

MOVC A,@A+DPTR

指令长度（字节）： 1

执行周期： 4

二进制编码：

1	0	0	1	0	0	1	1

操作： MOVC

(A) ←((A)+(DPTR))

MOVC A,@A+PC

指令长度（字节）： 1

执行周期： 3

二进制编码：

1	0	0	0	0	0	1	1

操作： MOVC

(PC) ←(PC)+1

(A) ←((A)+(PC))

MOVX <dest-byte> , <src-byte>

功能： 外部传送

说明： MOVX 指令用于在累加器和外部数据存储器之间传递数据。因此在传送指令 MOV 后附加了 X。MOVX 又分为两种类型，它们之间的区别在于访问外部数据 RAM 的间接地址是 8 位的还是 16 位的。

对于第 1 种类型，当前工作寄存器组的 R0 和 R1 提供 8 位地址到复用端口 P0。对于外部 I/O 扩展译码或者较小的 RAM 阵列，8 位的地址已经够用。若要访问较大的 RAM 阵列，可在端口引脚上输出高位的地址信号。此时可在 MOVX 指令之前添加输出指令，对这些

端口引脚施加控制。

对于第 2 种类型，通过数据指针 DPTR 产生 16 位的地址。当 P2 端口的输出缓冲器发送 DPH 的内容时，P2 的特殊功能寄存器保持原来的数据。在访问规模较大的数据阵列时，这种方式更为有效和快捷，因为不需要额外指令来配置输出端口。

在某些情况下，可以混合使用两种类型的 MOVX 指令。在访问大容量的 RAM 空间时，既可以用数据指针 DP 在 P2 端口上输出地址的高位字节，也可以先用某条指令，把地址的高位字节从 P2 端口上输出，再使用通过 R0 或 Rl 间址寻址的 MOVX 指令。

举例： 假设有一个分时复用地址/数据线的外部 RAM 存储器，容量为 256B（如：Intel 的 8155 RAM / I/O / TIMER），该存储器被连接到 8051 的端口 P0 上，端口 P3 被用于提供外部 RAM 所需的控制信号。端口 Pl 和 P2 用作通用输入/输出端口。R0 和 Rl 中的数据分别为 12H 和 34H，外部 RAM 的 34H 单元存储的数据为 56H，则下面的指令序列：

```
MOVX A, @R1
MOVX @R0, A
```

将数据 56H 复制到累加器 A 以及外部 RAM 的 12H 单元中。

MOVX A,@Ri

指令长度（字节）： 1

执行周期： 3 或 1

二进制编码：

1	1	1	0	0	0	1	i

操作： MOVX

(A) ← ((Ri))

MOVX A,@DPTR

指令长度（字节）： 1

执行周期： 1 或 2

二进制编码：

1	1	1	0	0	0	0	0

操作： MOVX

(A) ←((DPTR))

MOVX @Ri, A

指令长度（字节）： 1

执行周期： 3 或 1

二进制编码：

1	1	1	1	0	0	1	i

操作： MOVX

((Ri))←(A)

MOVX @DPTR, A

指令长度（字节）： 1

执行周期： 2 或 1

二进制编码：

1	1	1	1	0	0	0	0

操作： MOVX

(DPTR)←(A)

MUL AB

功能： 乘法

说明： 该指令可用于实现累加器和寄存器 B 中的无符号 8 位整数的乘法。所产生的 16 位乘积的低 8 位存放在累加器中，而高 8 位存放在寄存器 B 中。若乘积大于 255(0FFH)，则置位溢出标志；否则清零标志位。在执行该指令时，进位标志总是被清零。

举例： 假设累加器 A 的初始值为 80(50H)，寄存器 B 的初始值为 160 (0A0H)，则指令：

MUL AB

求得乘积 12800 (3200H)，所以寄存器 B 的值变成 32H (00110010B)，累加器被清零，溢出标志被置位，进位标志被清零。

指令长度(字节)： 1

执行周期： 2

二进制编码：

1	0	1	0	0	1	0	0

操作： $(A)_{7\text{-}0} \leftarrow (A)\times(B)$

$(B)_{15\text{-}8}$

NOP

功能： 空操作

说明： 执行本指令后，将继续执行随后的指令。除了 PC 外，其他寄存器和标志位都不会有变化。

举例： 假设期望在端口 P2 的第 7 号引脚上输出一个长时间的低电平脉冲，该脉冲持续 5 个机器周期（精确）。若是仅使用 SETB 和 CLR 指令序列，生成的脉冲只能持续 1 个机器周期。因而需要设法增加 4 个额外的机器周期。可以按照如下方式来实现所要求的功能（假设中断没有被启用）：

```
CLR P2.7
NOP
NOP
NOP
NOP
SETB P2.7
```

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	0	0	0	0	0	0

操作： NOP

(PC)←(PC)+1

ORL <dest-byte> , <src-byte>

功能： 两个字节变量的逻辑或运算

说明： ORL 指令将由<dest-byte>和<src_byte>所指定的两个字节变量进行逐位逻辑或运算，结果存放在<dest-byte>所代表的数据单元中。该操作不影响标志位。

两个操作数组合起来，支持 6 种寻址方式。当目的操作数是累加器 *A* 时，源操作数可以采用寄存器寻址、直接寻址、寄存器间接寻址或者立即寻址。当目的操作数采用直接寻址方式时，源操作数可以是累加器或立即数。

注意：如果该指令被用来修改输出引脚上的状态，那么<dest-byte>所代表的数据是从端口输出数据锁存器中获取的数据，而不是从引脚上读取的数据。

举例： 假设累加器 *A* 中数据为 0C3H (11000011B)，寄存器 *R*0 中的数据为 55H(01010101)，则指令：

ORL A, R0

执行后，累加器的内容变成 0D7H(11010111B)。当目的操作数是直接寻址数据字节时，ORL 指令可用来把任何 RAM 单元或者硬件寄存器中的各个位设置为 1。究竟哪些位会被置 1 由屏蔽字节决定，屏蔽字节既可以是包含在指令中的常数，也可以是累加器 *A* 在运行过程

中实时计算出的数值。执行指令：

ORL P1, #00110010B

之后，把 1 口的第 5、4、1 位置 1。

ORL A, Rn

指令长度（字节）：1

执行周期：1

二进制编码：

0	1	0	0	1	r	r	r

操作：ORL

(A) ←(A)∨(Rn)

ORL A, direct

指令长度（字节）：2

执行周期：1

二进制编码：

0	1	0	0	0	1	0	1		direct address

操作：ORL

(A) ←(A)∨(direct)

ORL A, @Ri

指令长度（字节）：1

执行周期：1

二进制编码：

0	1	0	0	0	1	1	i

操作：ORL

(A) ←(A)∨((Ri))

ORL A, #data

指令长度（字节）：2

执行周期：1

二进制编码：

0	1	0	0	0	1	0	0		immediate data

操作：ORL

(A) ←(A)∨#data

ORL direct, A

指令长度（字节）：2

执行周期：1

二进制编码：

0	1	0	0	0	0	1	0		direct address

操作：ORL

(direct)← (direct)∨(A)

ORL direct, #data

指令长度（字节）：3

执行周期：1

二进制编码：

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

操作：ORL

(direct)← (direct)∨#data

ORL C, <src-bit>

功能：位变量的逻辑或运算

说明：如果<src-bit>所表示的位变量为 1，则置位进位标志；否则，保持进位标志的当前状态不变。在汇编语言中，位于源操作数之前的“/”表示将源操作数取反后使用，但源操作数本身不

发生变化。在执行本指令时，不影响其他标志位。

举例： 当执行如下指令序列时，当且仅当 P1.0＝1 或 ACC.7＝1 或 OV＝0 时，置位进位标志 C：

```
MOV C, P1.0         ;LOAD CARRY WITH INPUT PIN P10
ORL C, ACC.7        ;OR CARRY WITH THE ACC.BIT 7
ORL C, /OV          ;OR CARRY WITH THE INVERSE OF OV
```

ORL C, bit

指令长度（字节）： 2

执行周期： 1 或 4

二进制编码：

0	1	1	1	0	0	1	0		bit address

操作： ORL

(C) ←(C)∨(bit)

ORL C, /bit

指令长度（字节）： 2

执行周期： 1

二进制编码：

1	0	1	0	0	0	0	0		bit address

操作： ORL

(C) ←(C)∨($\overline{bit}$)

POP direct

功能： 出栈

说明： 读取栈指针所指定的内部 RAM 单元的内容，栈指针减 1。然后，将读到的内容传送到由 direct 所指示的存储单元（直接寻址方式）中去。该操作不影响标志位。

举例： 设栈指针的初值为 32H，内部 RAM 的 30H~32H 单元的数据分别为 20H、23H 和 01H。则执行指令：

```
POP DPH
POP DPL
```

之后，栈指针的值变成 30H，数据指针变为 0123H。此时指令：

```
POP SP
```

将把栈指针变为 20H。

注意：在这种特殊情况下，在写入出栈数据（20H）之前，栈指针先减小到 2FH，然后再随着 20H 的写入，变成 20H。

指令长度（字节）： 2

执行周期： 1

二进制编码：

1	1	0	1	0	0	0	0		direct address

操作： POP

(direct)←((SP))

(SP) ← (SP) - 1

PUSH direct

功能： 压栈

说明： 栈指针首先加 1，然后将 direct 所表示的变量内容复制到由栈指针指定的内部 RAM 存储单元中去。该操作不影响标志位。

举例： 设在进入中断服务程序时栈指针的值为 09H，数据指针 DPTR 的值为 0123H。则执行如下指令序列：

PUSH DPL

PUSH DPH

之后，栈指针变为 0BH，并把数据 23H 和 01H 分别存入内部 RAM 的 0AH 和 0BH 存储单元之中。

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	1	0	0	0	0	0	0		direct address

操作： PUSH

(SP) ←(SP) + 1

((SP)) ← (direct)

RET

功能： 从子例程返回

说明： 执行 RET 指令时，首先将 PC 值的高位字节和低位字节从栈中弹出，栈指针减 2。然后，程序从形成的 PC 值所对应的地址处开始执行，一般情况下，该指令和 ACALL 或 LCALL 配合使用。改指令的执行不影响标志位。

举例： 设栈指针的初值为 0BH，内部 RAM 的 0AH 和 0BH 存储单元中的数据分别为 23H 和 01H。则指令：

RET

执行后，栈指针变为 09H。程序将从 0123H 地址处继续执行。

指令长度(字节)： 1

执行周期： 3

二进制编码：

0	0	1	0	0	0	1	0

操作： RET

(PC_{15-8}) ←((SP))

(SP) ←(SP) -1

(PC_{7-0}) ←((SP))

(SP) ←(SP) -1

RETI

功能： 中断返回

说明： 执行该指令时，首先从栈中弹出 PC 值的高位和低位字节，然后恢复中断启用，准备接受同优先级的其他中断，栈指针减 2。其他寄存器不受影响。但程序状态字 PSW 不会自动恢复到中断前的状态。程序将继续从新产生的 PC 值所对应的地址处开始执行，一般情况下是此次中断入口的下一条指令。在执行 RETI 指令时，如果有一个优先级较低的或同优先级的其他中断在等待处理，那么在处理这些等待中的中断之前需要执行 1 条指令。

举例： 设栈指针的初值为 0BH，结束在地址 0123H 处的指令执行结束期间产生中断，内部 RAM 的 0AH 和 0BH 单元的内容分别为 23H 和 01H。则指令：

RETI

执行完毕后，栈指针变成 09H，中断返回后程序继续从 0123H 地址开始执行。

指令长度(字节)： 1

执行周期：3

二进制编码：

0	0	1	1	0	0	1	0

操作：RETI

$(PC_{15-8}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

$(PC_{7-0}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

RL A

功能：将累加器 *A* 中的数据位循环左移

说明：将累加器中的 8 位数据均左移 1 位，其中位 7 移动到位 0。该指令的执行不影响标志位。

举例：设累加器的内容为 0C5H（11000101B），则指令：

RL A

执行后，累加器的内容变成 8BH（10001011B），且标志位不受影响。

指令长度(字节)：1

执行周期：1

二进制编码：

0	0	1	0	0	0	1	1

操作：RL

(An+1) ← (An)　　n = 0-6

(A0) ←(A7)

RLC A

功能：带进位循环左移

说明：累加器的 8 位数据和进位标志一起循环左移 1 位。其中位 7 移入进位标志，进位标志的初始状态值移到位 0。该指令不影响其他标志位。

举例：假设累加器 *A* 的值为 0C5H(11000101B)，则指令：

RLC A

执行后，将把累加器 *A* 的数据变为 8BH(10001011B)，进位标志被置位。

指令长度(字节)：1

执行周期：1

二进制编码：

0	0	1	1	0	0	1	1

操作：RLC

(An+1) ←(An)　　n = 0-6

(A0) ←(C)

(C) ←(A7)

RR A

功能：将累加器的数据位循环右移

说明：将累加器的 8 个数据位均右移 1 位，位 0 将被移到位 7，即循环右移，该指令不影响标志位。

举例：设累加器的内容为 0C5H（11000101B），则指令：

RR A

执行后累加器的内容变成 0E2H（11100010B），标志位不受影响。

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	0	0	0	0	1	1

操作： RR

(An) ←(An+1)　　n = 0 - 6

(A7) ←(A0)

RRC A

功能： 带进位循环右移

说明： 累加器的 8 位数据和进位标志一起循环右移 1 位。其中位 0 移入进位标志，进位标志的初始状态值移到位 7。该指令不影响其他标志位。

举例： 假设累加器的值为 0C5H(11000101B)，进位标志为 0，则指令：

RRC A

执行后，将把累加器的数据变为 62H(01100010B)，进位标志被置位。

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	0	0	1	0	0	1	1

操作： RRC

(An+1) ←(An)　　n = 0 - 6

(A7) ←(C)

(C) ←(A0)

SETB <bit>

功能： 置位

说明： SETB 指令可将相应的位置 1，其操作对象可以是进位标志或其他可直接寻址的位。该指令不影响其他标志位。

举例： 设进位标志被清零，端口 1 的输出状态为 34H(00110100B)，则指令：

SETB C

SETB P1.0

执行后，进位标志变为 1，端口 1 的输出状态变成 35H(00110101B)。

SETB C

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	0	1	0	0	1	1

操作： SETB

(C) ← 1

SETB bit

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	1	0	1	0	0	1	0		bit address

操作： SETB

(bit) ← 1

SJMP rel

功能： 短跳转

说明： 程序无条件跳转到 rel 所示的地址去执行。目标地址按如下方法计算：首先 PC 值加 2，然后将指令第 2 字节（即 rel）所表示的有符号偏移量加到 PC 上，得到的新 PC 值即短跳转的目标地址。所以，跳转的范围是当前指令（即 SJMP）地址的前 128 字节和后 127 字节。

举例： 设标号 RELADR 对应的指令地址位于程序存储器的 0123H 地址，则指令：

SJMP RELADR

汇编后位于 0100H。当执行完该指令后，PC 值变成 0123H。

注意：在上例中，紧接 SJMP 的下一条指令的地址是 0102H，因此，跳转的偏移量为 0123H−0102H＝21H。另外，如果 SJMP 的偏移量是 0FEH，那么构成只有 1 条指令的无限循环。

指令长度(字节)： 2

执行周期： 3

二进制编码：

1	0	0	0	0	0	0	0		rel. address

操作： SJMP

(PC) ←(PC)+2

(PC) ← (PC)+rel

SUBB A, <src-byte>

功能： 带借位的减法

说明： SUBB 指令从累加器中减去<src-byte>所代表的字节变量的数值及进位标志，减法运算的结果置于累加器中。如果执行减法时第 7 位需要借位，SUBB 将会置位进位标志(表示借位)；否则，清零进位标志。（如果在执行 SUBB 指令前，进位标志 C 已经被置位，这意味着在前面进行多精度的减法运算时，产生了借位。因而在执行本条指令时，必须把进位连同源操作数一起从累加器中减去。）如果在进行减法运算的时候，第 3 位处向上有借位，那么辅助进位标志 AC 会被置位；如果第 6 位有借位；而第 7 位没有，或是第 7 位有借位，而第 6 位没有， 则溢出标志 OV 被置位。

当进行有符号整数减法运算时，若 OV 置位，则表示在正数减负数的过程中产生了负数；或者，在负数减正数的过程中产生了正数。

源操作数支持的寻址方式：寄存器寻址、直接寻址、寄存器间接寻址和立即数寻址。

举例： 设累加器中的数据为 0C9H(11001001B)。寄存器 *R2* 的值为 54H(01010100B)，进位标志 C 被置位。则如下指令：

SUBB A, R2

执行后，累加器的数据变为 74H(01110100B)，进位标志 C 和辅助进位标志 AC 被清零，溢出标志 C 被置位。

注意：0C9H 减去 54H 应该是 75H，但在上面的计算中，由于在 SUBB 指令执行前，进位标志 C 已经被置位，因而最终结果还需要减去进位标志，得到 74H。因此，如果在进行单精度或者多精度减法运算前，进位标志 C 的状态未知，那么应改采用 CLR C 指令把进位标志 C 清零。

SUBB A, Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	0	0	1	1	r	r	r

操作：SUBB

(A) ← (A) - (C) - (Rn)

SUBB A, direct

指令长度（字节）：2

执行周期：1

二进制编码：

1	0	0	1	0	1	0	1		direct address

操作：SUBB

(A) ← (A) - (C) - (direct)

SUBB A, @Ri

指令长度（字节）：1

执行周期：1

二进制编码：

1	0	0	1	0	1	1	i

操作：SUBB

(A) ← (A) - (C) - ((Ri))

SUBB A, #data

指令长度（字节）：2

执行周期：1

二进制编码：

1	0	0	1	0	1	0	0		immediate data

操作：SUBB

(A) ← (A) - (C) - #data

SWAP A

功能：交换累加器的高低半字节

说明：SWAP 指令把累加器的低 4 位（位 3~位 0）和高 4 位（位 7~位 4）数据进行交换。实际上 SWAP 指令也可视为 4 位的循环指令。该指令不影响标志位。

举例：设累加器的内容为 0C5H（11000101B），则指令：

SWAP A

执行后，累加器的内容变成 5CH（01011100B）。

指令长度（字节）：1

执行周期：1

二进制编码：

1	1	0	0	0	1	0	0

操作：SWAP

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

XCH A, <byte>

功能：交换累加器和字节变量的内容

说明：XCH 指令将<byte>所指定的字节变量的内容装载到累加器，同时将累加器的旧内容写入<byte>所指定的字节变量。指令中的源操作数和目的操作数允许的寻址方式：寄存器寻址、直接寻址和寄存器间接寻址。

举例：设 R0 的内容为地址 20H，累加器的值为 3FH (00111111B)。内部 RAM 的 20H 单元的内容为 75H (01110101B)。则指令：

XCH A, @R0

执行后，内部 RAM 的 20H 单元的数据变为 3FH (00111111B)，累加器的内容变为 75H(01110101B)。

XCH A, Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	0	0	1	r	r	r

操作： XCH

(A) ↔ (Rn)

XCH A, direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

1	1	0	0	0	1	0	1		direct address

操作： XCH

(A) ↔ (direct)

XCH A, @Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	0	0	0	1	1	i

操作： XCH

(A) ↔ ((Ri))

XCHD A, @Ri

功能： 交换累加器和@Ri 对应单元中的数据的低 4 位

说明： XCHD 指令将累加器内容的低半字节（位 0~3，一般是十六进制数或 BCD 码）和间接寻址的内部 RAM 单元的数据进行交换，各自的高半字（位 7~4）节不受影响。另外，该指令不影响标志位。

举例： 设 R0 保存了地址 20H，累加器的内容为 36H (00110110B)。内部 RAM 的 20H 单元存储的数据为 75H (011110101B)。则指令：

XCHD A, @R0

执行后，内部 RAM 20H 单元的内容变成 76H (01110110B)，累加器的内容变为 35H(00110101B)。

指令长度(字节)： 1

执行周期： 1

二进制编码：

1	1	0	1	0	1	1	i

操作： XCHD

(A_{3-0}) ↔ (Ri_{3-0})

XRL <dest-byte>, <src-byte>

功能： 字节变量的逻辑异或

说明： XRL 指令将<dest-byte>和<src-byte>所代表的字节变量逐位进行逻辑异或运算，结果保存在<dest-byte>所代表的字节变量里。该指令不影响标志位。

两个操作数组合起来共支持 6 种寻址方式：当目的操作数为累加器时，源操作数可以采用寄存器寻址、直接寻址、寄存器间接寻址和立即数寻址；当目的操作数是可直接寻址的数据时，源操作数可以是累加器或者立即数。

注意：如果该指令被用来修改输出引脚上的状态，那么 dest-byte 所代表的数据就是从端口输出数据锁存器中获取的数据，而不是从引脚上读取的数据。

举例： 如果累加器和寄存器 0 的内容分别为 0C3H (11000011B)和 0AAH(10101010B)，则指令：

XRL A, R0

执行后，累加器的内容变成 69H (01101001B)。

当目的操作数是可直接寻址字节数据时，该指令可把任何 RAM 单元或者寄存器中的各个位取反。具体哪些位会被取反，在运行过程当中确定。指令：

XRL P1, #00110001B

执行后，P1 口的位 5、4、0 被取反。

XRL A, Rn

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	1	1	0	1	r	r	r

操作： XRL

(A) ←(A) ∀ (Rn)

XRL A, direct

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	1	0	0	1	0	1		direct address

操作： XRL

(A) ←(A) ∀ (direct)

XRL A, @Ri

指令长度(字节)： 1

执行周期： 1

二进制编码：

0	1	1	0	0	1	1	i

操作： XRL

(A) ←(A) ∀ ((Ri))

XRL A, #data

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	1	0	0	1	0	0		immediate data

操作： XRL

(A) ←(A) ∀ #data

XRL direct, A

指令长度(字节)： 2

执行周期： 1

二进制编码：

0	1	1	0	0	0	1	0		direct address

操作： XRL

(direct) ←(direct) ∀ (A)

XRL direct, #data

指令长度(字节)： 3

执行周期： 1

二进制编码：

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

操作： XRL

(direct)← (direct) ∀ #data

10.4 指令详解（英文）

ACALL addr11

Function: Absolute Call

Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the Stack Pointer twice.
The destination address is obtained by successively concatenating the five high-order bits of the incremented PC opcode bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

Example: Initially SP equals 07H. The label “SUBRTN” is at program memory location 0345H. After executing the instruction,
ACALL SUBRTN
at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

Bytes: 2

Cycles: 3

Encoding:

A10	A9	A8	1	0	0	0	1		A7	A6	A5	A4	A3	A2	A1	A0

Operation: ACALL
(PC)←(PC)+ 2
(SP)←(SP) + 1
((SP)) ←(PC_{7-0})
(SP)←(SP) + 1
((SP))←(PC_{15-8})
(PC_{10-0})←page address

ADD A, <src-byte>

Function: Add

Description: ADD adds the byte variable indicated to the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct register-indirect, or immediate.

Example: The Accumulator holds 0C3H(11000011B) and register 0 holds 0AAH (10101010B). The instruction,

ADD A, R0

will leave 6DH (01101101B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADD A, Rn

Bytes: 1

Cycles: 1

Encoding:

0	0	1	0	1	r	r	r

Operation: ADD

(A)← (A) + (Rn)

ADD A, direct

Bytes: 2

Cycles: 1

Encoding:

0	0	1	0	0	1	0	1		direct address

Operation: ADD

(A)← (A) + (direct)

ADD A, @Ri

Bytes: 1

Cycles: 1

Encoding:

0	0	1	0	0	1	1	i

Operation: ADD

(A)← (A) + ((Ri))

ADD A, #data

Bytes: 2

Cycles: 1

Encoding:

0	0	1	0	0	1	0	0		immediate data

Operation: ADD

(A)← (A) + #data

ADDC A, <src-byte>

Function: Add with Carry

Description: ADDC simultaneously adds the byte variable indicated, the Carry flag and the Accumulator, leaving the result in the Accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not out of bit 6;

otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C3H(11000011B) and register 0 holds 0AAH (10101010B) with the Carry. The instruction,

ADDC A,R0

will leave 6EH (01101110B) in the Accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADDC A, Rn

Bytes: 1

Cycles: 1

Encoding:

0	0	1	1	1	r	r	r

Operation: ADDC

(A)← (A) + (C) + (Rn)

ADDC A, direct

Bytes: 2

Cycles: 1

Encoding:

0	0	1	1	0	1	0	1		direct address

Operation: ADDC

(A)← (A) + (C) + (direct)

ADDC A, @Ri

Bytes: 1

Cycles: 1

Encoding:

0	0	1	1	0	1	1	i

Operation: ADDC

(A)← (A) + (C) + ((Ri))

ADDC A, #data

Bytes: 2

Cycles: 1

Encoding:

0	0	1	1	0	1	0	0		immediate data

Operation: ADDC

(A)← (A) + (C) + #data

AJMP addr11

Function: Absolute Jump

Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (after incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.

Example: The label “JMPADR” is at program memory location 0123H. The instruction,

AJMP JMPADR

is at location 0345H and will load the PC with 0123H.

Bytes: 2

Cycles: 3

Encoding:

A10	A9	A8	0	0	0	0	1		A7	A6	A5	A4	A3	A2	A1	A0

Operation: AJMP

(PC)←(PC)+ 2

(PC_{10-0})←page address

ANL <dest-byte> , <src-byte>

Function: Logical-AND for byte variables

Description: ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch not the input pins.

Example: If the Accumulator holds 0C3H(11000011B) and register 0 holds 55H (01010101B) then the instruction,

ANL A,R0

will leave 41H (01000001B) in the Accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the Accumulator at run-time.

The instruction,

ANL Pl, #01110011B

will clear bits 7, 3, and 2 of output port 1.

ANL A, Rn

Bytes: 1

Cycles: 1

Encoding:

0	1	0	1	1	r	r	r

Operation: ANL

(A)← (A) ∧ (Rn)

ANL A, direct

Bytes: 2

Cycles: 1

Encoding:

0	1	0	1	0	1	0	1		direct address

Operation: ANL

(A)← (A) ∧ (direct)

ANL A, @Ri

Bytes: 1

Cycles: 1

Encoding:

0	1	0	1	0	1	1	i

Operation: ANL

(A)← (A) ∧ ((Ri))

ANL A, #data

Bytes: 2

Cycles: 1

Encoding:

0	1	0	1	0	1	0	0		immediate data

Operation: ANL

(A)← (A) ∧ #data

ANL direct, A

Bytes: 2

Cycles: 1

Encoding:

0	1	0	1	0	0	1	0		direct address

Operation: ANL

(direct)←(direct) ∧ (A)

ANL direct, #data

Bytes: 3

Cycles: 1

Encoding:

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

Operation: ANL

(direct)←(direct) ∧ #data

ANL C, <src-bit>

Function: Logical-AND for bit variables

Description: If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash (" / ") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected.* No other flags are affected.

Only direct addressing is allowed for the source operand.

Example: Set the carry flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

```
MOV C, P1.0          ; LOAD CARRY WITH INPUT PIN STATE
ANL C, ACC.7         ; AND CARRY WITH ACCUM. BIT.7
ANL C, /OV           ;AND WITH INVERSE OF OVERFLOW FLAG
```

ANL C, bit

Bytes: 2

Cycles: 1

Encoding:

1	0	0	0	0	0	1	0		bit address

Operation: ANL

(C) ←(C) ∧ (bit)

ANL C, /bit

Bytes: 2

Cycles: 1

Encoding:

1	0	1	1	0	0	0	0		bit address

Operation: ANL

(C)← (C) ∧ ($\overline{bit}$)

CJNE <dest-byte>, <src-byte>, rel

Function: Compare and Jump if Not Equal

Description: CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the Accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The Accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```
                CJNE R7,#60H, NOT_EQ
;               . . .       . . . . . .         ; R7 = 60H.
NOT_EQ:  JC     REQ_LOW      ; IF R7 < 60H.
;               . . .       . . . . . .         ; R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT-EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to Port 1 is also 34H, then the instruction,

WAIT: CJNE A,P1,WAIT

clears the carry flag and continues with the next instruction in sequence, since the Accumulator does equal the data read from P1. (If some other value was being input on Pl, the program will loop at this point until the P1 data changes to 34H.)

CJNE A, direct, rel

Bytes: 3

Cycles: 2 or 3

Encoding:

1	0	1	1	0	1	0	1		direct address		rel. address

Operation:

(PC) ←(PC) + 3

IF (A) < > *(direct)*

THEN

(PC) ←(PC) +*relative offset*

IF (A) < *(direct)*

THEN

(C) ← 1

ELSE

(C) ← 0

CJNE A, #data, rel

Bytes: 3

Cycles: 1 or 3

Encoding:

1	0	1	1	0	1	0	0		immediate data		rel. address

Operation:

(PC) ←(PC) + 3

IF (A) < > *(data)*

THEN

(PC) ←(PC) +*relative offset*

IF (A) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← 0

CJNE Rn, #data, rel

Bytes: 3

Cycles: 2 or 3

Encoding:

1	0	1	1	1	r	r	r		immediate data		rel. address

Operation: (PC) ←(PC) + 3

IF (Rn) < > *(data)*

THEN

(PC) ←(PC) +*relative offset*

IF (Rn) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← 0

CJNE @Ri, #data, rel

Bytes: 3

Cycles: 2 or 3

Encoding:

1	0	1	1	0	1	1	i		immediate data		rel. address

Operation: (PC) ←(PC) + 3

IF (Ri) < > *(data)*

THEN

(PC) ←(PC) +*relative offset*

IF (Ri) < *(data)*

THEN

(C) ← 1

ELSE

(C) ← 0

CLR A

Function: Clear Accumulator

Description: The Accumulator is cleared (all bits set on zero). No flags are affected.

Example: The Accumulator contains 5CH (01011100B). The instruction,

CLR A

will leave the Accumulator set to 00H (00000000B).

Bytes: 1

Cycles: 1

Encoding:

1	1	1	0	0	1	0	0

Operation: CLR

(A)←0

CLR bit

Function: Clear bit

Description: The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

Example: Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR P1.2

will leave the port set to 59H (01011001B).

CLR C

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	0	1	1

Operation: CLR

(C)←0

CLR bit

Bytes: 2

Cycles: 1

Encoding:

1	1	0	0	0	0	1	0		bit address

Operation: CLR

(bit)←0

CPL A

Function: Complement Accumulator

Description: Each bit of the Accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to a zero and vice-versa. No flags are affected.

Example: The Accumulator contains 5CH(01011100B). The instruction,

CPL A

will leave the Accumulator set to 0A3H (10100011B).

Bytes: 1

Cycles: 1

Encoding:

1	1	1	1	0	1	0	0

Operation: CPL

(A)← ($\bar{A}$)

CPL bit

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.

Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Example: Port 1 has previously been written with 5BH(01011011B). The instruction,

CPL P1.1

CPL P1.2

will leave the port set to 5DH(01011101B).

CPL C

Bytes: 1

Cycles: 1

Encoding:

1	0	1	1	0	0	1	1

Operation: CPL

(C)← ($\bar{C}$)

CPL bit

Bytes: 2

Cycles: 1

Encoding:

1	0	1	1	0	0	1	0		bit address

Operation: CPL

(bit)← ($\overline{bit}$)

DA A

Function: Decimal-adjust Accumulator for Addition

Description: DA A adjusts the eight-bit value in the Accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If Accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the Accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set or if the four high-order bits now exceed nine(1010xxxx- 1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the Accumulator, depending on initial Accumulator and PSW conditions.

Note: DA A cannot simply convert a hexadecimal number in the Accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Example: The Accumulator holds the value 56H(01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence.

```
ADDC A,R3
DA   A
```

will first perform a standard twos-complement binary addition, resulting in the value 0BEH (10111110) in the Accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the Accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56,67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the Accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,
ADD A, #99H
DA A
will leave the carry set and 29H in the Accumulator, since 30+99=129. The low-order byte of the sum can be interpreted to mean 30 – 1 = 29.

Bytes: 1

Cycles: 3

Encoding:

1	1	0	1	0	1	0	0

Operation: DA

-contents of Accumulator are BCD

IF [[(A_{3-0}) > 9] V [(AC) = 1]]

THEN(A_{3-0}) ←(A_{3-0}) + 6

AND

IF [[(A_{7-4}) > 9] V [(C) = 1]]

THEN (A_{7-4})←(A_{7-4}) + 6

DEC byte

Function: Decrement

Description: The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,
DEC @R0
DEC R0
DEC @R0
will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

DEC A

Bytes: 1

Cycles: 1

Encoding:

0	0	0	1	0	1	0	0

Operation: DEC

(A)← (A) −1

DEC Rn

Bytes: 1

Cycles: 1

Encoding:

0	0	0	1	1	r	r	r

Operation: DEC

(Rn)←(Rn) - 1

DEC direct

Bytes: 2

Cycles: 1

Encoding:

0	0	0	1	0	1	0	1		Direct address

Operation: DEC

(direct)←(direct) - 1

DEC @Ri

Bytes: 1

Cycles: 1

Encoding:

0	0	0	1	0	1	1	i

Operation: DEC

((Ri))←((Ri)) - 1

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer in the Accumulator by the unsigned eight-bit integer in register B. The Accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: if B had originally contained 00H, the values returned in the Accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The Accumulator contains 251(OFBH or 11111011B) and B contains 18(12H or 00010010B). The instruction,

DIV AB

will leave 13 in the Accumulator (0DH or 00001101B) and the value 17 (11H or 00010001B) in B, since 251 = (13×18) + 17. Carry and OV will both be cleared.

Bytes: 1

Cycles: 6

Encoding:

1	0	0	0	0	1	0	0

Operation: DIV

$(A)_{15\text{-}8}(B)_{7\text{-}0} \leftarrow (A)/(B)$

DJNZ <byte>, <rel-addr>

Function: Decrement and Jump if Not Zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the

second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

DJNZ 40H, LABEL_1

DJNZ 50H, LABEL_2

DJNZ 60H, LABEL_3

will cause a jump to the instruction at label LABEL_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was not taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

```
        MOV R2,#8
TOOOLE: CPL P1.7
        DJNZ R2, TOOGLE
```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output Port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

DJNZ Rn, rel

Bytes: 2

Cycles: 2 or 3

Encoding:

1	1	0	1	1	r	r	r		rel. address

Operation: DJNZ

(PC) ←(PC) + 2

(Rn) ←(Rn) – 1

IF (Rn) > 0 or (Rn) < 0

THEN

(PC) ←(PC)+ rel

DJNZ direct, rel

Bytes: 3

Cycles: 2 or 3

Encoding:

1	1	0	1	0	1	0	1		direct address		rel. address

Operation: DJNZ

(PC) ←(PC) + 2

(direct) ←(direct) – 1

IF (direct) > 0 or (direct) < 0

THEN

(PC) ←(PC)+ rel

INC <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H.No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: Register 0 contains 7EH (01111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

INC @R0

INC R0

INC @R0

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

INC A

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	0	1	0	0

Operation: INC

(A) ←(A)+1

INC Rn

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	1	r	r	r

Operation: INC

(Rn) ←(Rn)+1

INC direct

Bytes: 2

Cycles: 1

Encoding:

0	0	0	0	0	1	0	1		direct address

Operation: INC

(direct) ←(direct)+1

INC @Ri

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	0	1	1	i

Operation: INC

((Ri))←((Ri)) + 1

INC DPTR

Function: Increment Data Pointer

Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2_{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order-byte (DPH). No flags are affected.

This is the only 16-bit register which can be incremented.

Example: Register DPH and DPL contains 12H and 0FEH, respectively. The instruction sequence,

IINC DPTR

INC DPTR

INC DPTR

will change DPH and DPL to 13H and 01H.

Bytes: 1

Cycles: 1

Encoding:

1	0	1	0	0	0	1	1

Operation: INC

(DPTR) ← (DPTR) + 1

JB bit, rel

Function: Jump if Bit set

Description: If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified. No flags are affected*

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56 (01010110B). The instruction sequence,

JB P1.2, LABEL1

JB ACC.2, LABEL2

will cause program execution to branch to the instruction at label LABEL2.

Bytes: 3

Cycles: 1 or 3

Encoding:

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

Operation: JB

(PC) ← (PC) + 3

IF (bit) = 1

THEN

(PC) ← (PC)+ rel

JBC bit, rel

Function: Jump if Bit is set and Clear bit

Description: If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *The bit will not be cleared if it is already a zero.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, not the input pin.

Example: The Accumulator holds 56H (01010110B). The instruction sequence,

JBC ACC.3, LABEL1

JBC ACC.2, LABEL2

will cause program execution to continue at the instruction identified by the label LABEL2, with the Accumulator modified to 52H (01010010B).

Bytes: 3

Cycles: 1 or 3

Encoding:

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

Operation: JB

(PC) ← (PC) + 3

IF (bit) = 1

THEN

(bit) ← 0

(PC) ← (PC)+ rel

JC rel

Function: Jump if Carry is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Example: The carry flag is cleared. The instruction sequence,

JC LABEL1

CPL C

JC LABEL2

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

Bytes: 2

Cycles: 1 or 3

Encoding:

0	1	0	0	0	0	0	0		rel. address

Operation: JC

(PC) ← (PC) + 2

IF (C) = 1

THEN

(PC) ← (PC)+ rel

JMP @A+DPTR

Function: Jump indirect

Description: Add the eight-bit unsigned contents of the Accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^{16}): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the Accumulator nor the Data Pointer is altered. No flags are affected.

Example: An even number from 0 to 6 is in the Accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP_TBL:

MOV DPTR, #JMP_TBL

JMP @A+DPTR

JMP-TBL: AJMP LABEL0

AJMP LABEL1

AJMP LABEL2

AJMP LABEL3

If the Accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1

Cycles: 4

Encoding:

0	1	1	1	0	0	1	1

Operation: JMP

(PC) ← (A) + (DPTR)

JNB bit, rel

Function: Jump if Bit is not set

Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The Accumulator holds 56H (01010110B). The instruction sequence,

JNB P1.3, LABEL1

JNB ACC.3, LABEL2

will cause program execution to continue at the instruction at label LABEL2.

Bytes: 3

Cycles: 1 or 3

Encoding:

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

Operation: JNB

(PC) ←(PC) + 3

IF (bit) = 0

THEN (PC) ← (PC) + rel

JNC rel

Function: Jump if Carry not set

Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Example: The carry flag is set. The instruction sequence,

JNC LABEL1

CPL C

JNC LABEL2

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

Bytes: 2

Cycles: 1 or 3

Encoding:

0	1	0	1	0	0	0	0		rel. address

Operation: JNC

(PC) ←(PC) + 2

IF (C) = 0

THEN (PC) ← (PC) + rel

JNZ rel

Function: Jump if Accumulator Not Zero

Description: If any bit of the Accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally holds 00H. The instruction sequence,

JNZ LABEL1

INC A

JNZ LAEEL2

will set the Accumulator to 01H and continue at label LABEL2.

Bytes: 2

Cycles: 1 or 3

Encoding:

0	1	1	1	0	0	0	0		rel. address

Operation: JNZ

(PC) ←(PC) + 2

IF (A) ≠ 0

THEN (PC) ← (PC) + rel

JZ rel

Function: Jump if Accumulator Zero

Description: If all bits of the Accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The Accumulator is not modified. No flags are affected.

Example: The Accumulator originally contains 01H. The instruction sequence,

JZ LABEL1

DEC A

JZ LAEEL2

will change the Accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 1 or 3

Encoding:

0	1	1	0	0	0	0	0		rel. address

STCMCU

Operation: JZ

(PC) ←(PC) + 2

IF (A) = 0

THEN (PC) ← (PC) + rel

LCALL addr16

Function: Long call

Description: LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the Stack Pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address.
The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

Example: Initially the Stack Pointer equals 07H. The label “SUBRTN” is assigned to program memory location 1234H. After executing the instruction,
LCALL SUBRTN
at location 0123H, the Stack Pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1234H.

Bytes: 3

Cycles: 3

Encoding:

0	0	0	1	0	0	1	0		addr15-addr8		addr7-addr0

Operation: LCALL

(PC) ←(PC) + 3

(SP) ←(SP) + 1

((SP)) ←(PC_{7-0})

(SP) ←(SP) + 1

((SP)) ←(PC_{15-8})

(PC) ← $addr_{15-0}$

LJMP addr16

Function: Long Jump

Description: LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

Example: The label “JMPADR” is assigned to the instruction at program memory location 1234H. The instruction,
LJMP JMPADR
at location 0123H will load the program counter with 1234H.

Bytes: 3

Cycles: 3

Encoding:

0	0	0	0	0	0	1	0		addr15-addr8		addr7-addr0

Operation: LJMP

(PC) ← addr15-0

MOV <dest-byte> , <src-byte>

Function: Move byte variable

Description: The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

Example: Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

MOV R0, #30H ; R0< = 30H

MOV A, @R0 ; A < = 40H

MOV R1, A ; R1 < = 40H

MOV B, @Rl ; B < = 10H

MOV @Rl, Pl ; RAM (40H) < = 0CAH

MOV P2, P1 ; P2 #0CAH

leaves the value 30H in register 0,40H in both the Accumulator and register 1,10H in register B, and 0CAH(11001010B) both in RAM location 40H and output on port 2.

MOV A,Rn

Bytes: 1

Cycles: 1

Encoding:

1	1	1	0	1	r	r	r

Operation: MOV

(A) ←(Rn)

***MOV A,direct**

Bytes: 2

Cycles: 1

Encoding:

1	1	1	0	0	1	0	1		direct address

Operation: MOV

(A)← (direct)

***MOV A, ACC is not a valid instruction.**

MOV A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1	1	1	0	0	1	1	i

Operation: MOV

(A) ←((Ri))

MOV A,#data

Bytes: 2

Cycles: 1

Encoding:

0	1	1	1	0	1	0	0		immediate data

Operation: MOV

(A)←#data

MOV Rn, A

Bytes: 1

Cycles: 1

Encoding:

1	1	1	1	1	r	r	r

Operation: MOV

(Rn) ←(A)

MOV Rn,direct

Bytes: 2

Cycles: 1

Encoding:

1	0	1	0	1	r	r	r		direct address

Operation: MOV

(Rn)← (direct)

MOV Rn,#data

Bytes: 2

Cycles: 1

Encoding:

0	1	1	1	1	r	r	r		immediate data

Operation: MOV

(Rn)←#data

MOV direct, A

Bytes: 2

Cycles: 1

Encoding:

1	1	1	1	0	1	0	1		direct address

Operation: MOV

(direct)← (A)

MOV direct, Rn

Bytes: 2

Cycles: 1

Encoding:

1	0	0	0	1	r	r	r		direct address

Operation: MOV

(direct)← (Rn)

MOV direct, direct

Bytes: 3

Cycles: 1

Encoding:

1	0	0	0	0	1	0	1		dir.addr. (src)		dir.addr. (dest)

Operation: MOV

(direct)← (direct)

MOV direct, @Ri

Bytes: 2

Cycles: 1

Encoding:

1	0	0	0	0	1	1	i		direct address

Operation: MOV

(direct)←((Ri))

MOV direct, #data

Bytes: 3

Cycles: 1

Encoding:

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

Operation: MOV

(direct)←#data

MOV @Ri, A

Bytes: 1

Cycles: 1

Encoding:

1	1	1	1	0	1	1	i

Operation: MOV

((Ri))←(A)

MOV @Ri, direct

Bytes: 2

Cycles: 1

Encoding:

1	0	1	0	0	1	1	i		direct address

Operation: MOV

((Ri))←(direct)

MOV @Ri, #data

Bytes: 2

Cycles: 1

Encoding:

0	1	1	1	0	1	1	i		immediate data

Operation: MOV

((Ri))←#data

MOV <dest-bit> , <src-bit>

Function: Move bit data

Description: The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

Example: The carry flag is originally set. The data present at input Port 3 is 11000101B. The data previously written to output Port 1 is 35H (00110101B).

MOV P1.3, C

MOV C, P3.3

MOV P1.2, C

will leave the carry cleared and change Port 1 to 39H (00111001B).

MOV C, bit

Bytes: 2

Cycles: 1

Encoding:

1	0	1	0	0	0	1	0		bit address

Operation: MOV

(C)← (bit)

MOV bit, C

Bytes: 2

Cycles: 1

Encoding:

1	0	0	1	0	0	1	0		bit address

Operation: MOV

STC MCU

(bit)← (C)

MOV DPTR , #data 16

Function: Load Data Pointer with a 16-bit constant

Description: The Data Pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

This is the only instruction which moves 16 bits of data at once.

Example: The instruction,

MOV DPTR, #1234H

will load the value 1234H into the Data Pointer: DPH will hold 12H and DPL will hold 34H.

Bytes: 3

Cycles: 1

Encoding:

1	0	0	1	0	0	0	0		immediate data15-8		immediate data7-0

Operation: MOV

(DPTR) ← #data$_{15\text{-}0}$

DPH DPL ← #data$_{15\text{-}8}$ #data$_{7\text{-}0}$

MOVC A , @A+ <base-reg>

Function: Move Code byte

Description: The MOVC instructions load the Accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit. Accumulator contents and the contents of a sixteen-bit base register, which may be either the Data Pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the Accumulator; otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example: A value between 0 and 3 is in the Accumulator. The following instructions will translate the value in the Accumulator to one of four values defined by the DB (define byte) directive.

```
REL-PC: INC A
        MOVC A, @A+PC
        RET
        DB 66H
        DB 77H
        DB 88H
        DB 99H
```

If the subroutine is called with the Accumulator equal to 01H, it will return with 77H in the Accumulator. The INC A before the MOVC instruction is needed to “get around” the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the Accumulator instead.

MOVC A,@A+DPTR

Bytes: 1

Cycles: 4

Encoding:

1	0	0	1	0	0	1	1

Operation: MOVC

(A) ←((A)+(DPTR))

MOVC A,@A+PC

Bytes: 1

Cycles: 3

Encoding:

1	0	0	0	0	0	1	1

Operation: MOVC

(PC) ←(PC)+1

(A) ←((A)+(PC))

MOVX <dest-byte> , <src-byte>

Function: Move External

Description: The MOVX instructions transfer data between the Accumulator and a byte of external data memory, hence the “X” appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or for a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the Data Pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the Data Pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

Example: An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel 8155 RAM/ I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H.

Location 34H of the external RAM holds the value 56H. The instruction sequence,

MOVX A, @R1

MOVX @R0, A

copies the value 56H into both the Accumulator and external RAM location 12H.

MOVX A,@Ri

Bytes: 1

Cycles: 3 or 1

Encoding:

1	1	1	0	0	0	1	i

Operation: MOVX

(A) ← ((Ri))

MOVX A,@DPTR

Bytes: 1

Cycles: 2 or 1

Encoding:

1	1	1	0	0	0	0	0

Operation: MOVX

(A) ←((DPTR))

MOVX @Ri, A

Bytes: 1

Cycles: 3 or 1

Encoding:

1	1	1	1	0	0	1	i

Operation: MOVX

((Ri))←(A)

MOVX @DPTR, A

Bytes: 1

Cycles: 2 or 1

Encoding:

1	1	1	1	0	0	0	0

Operation: MOVX

(DPTR)←(A)

MUL AB

Function: Multiply

Description: MUL AB multiplies the unsigned eight-bit integers in the Accumulator and register B. The low-order byte of the sixteen-bit product is left in the Accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

Example: Originally the Accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

MUL AB

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the Accumulator is cleared. The overflow flag is set, carry is cleared.

Bytes: 1

Cycles: 2

Encoding:

1	0	1	0	0	1	0	0

Operation: $(A)_{7\text{-}0} \leftarrow (A) \times (B)$

$(B)_{15\text{-}8}$

NOP

Function: No Operation

Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Example: It is desired to produce a low-going output pulse on bit 7 of Port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence.

CLR P2.7

NOP

NOP

NOP

NOP

SETB P2.7

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	0	0	0	0

Operation: NOP

(PC)←(PC)+1

ORL <dest-byte> , <src-byte>

Function: Logical-OR for byte variables

Description: ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.

Example: If the Accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

ORL A, R0

will leave the Accumulator holding the value 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the Accumulator at run-time. The instruction,

ORL P1, #00110010B

will set bits 5,4, and 1of output Port 1.

ORL A, Rn

Bytes: 1

Cycles: 1

Encoding:

0	1	0	0	1	r	r	r

Operation: ORL

(A) ←(A)∨(Rn)

ORL A, direct

Bytes: 2

Cycles: 1

Encoding:

0	1	0	0	0	1	0	1		direct address

Operation: ORL

(A) ←(A)∨(direct)

ORL A, @Ri

Bytes: 1

Cycles: 1

Encoding:

0	1	0	0	0	1	1	i

Operation: ORL

(A) ←(A)∨((Ri))

ORL A, #data

Bytes: 2

Cycles: 1

Encoding:

0	1	0	0	0	1	0	0		immediate data

Operation: ORL

(A) ←(A)∨#data

ORL direct, A

Bytes: 2

Cycles: 1

Encoding:

0	1	0	0	0	0	1	0		direct address

Operation: ORL

(direct)← (direct)∨(A)

ORL direct, #data

Bytes: 3

Cycles: 1

Encoding:

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

Operation: ORL

(direct)← (direct)∨#data

ORL C, <src-bit>

Function: Logical-OR for bit variables

Description: Set the carry flag if the Boolean value is a logical 1; leave the carry in its current state otherwise. A slash (" / ") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

Example: Set the carry flag if and only if P1.0 = 1, ACC. 7 = 1, or OV = 0:

```
MOV C, P1.0         ;LOAD CARRY WITH INPUT PIN P10
ORL C, ACC.7        ;OR CARRY WITH THE ACC.BIT 7
ORL C, /OV          ;OR CARRY WITH THE INVERSE OF OV
```

ORL C, bit

Bytes: 2

Cycles: 1 or 4

Encoding:

0	1	1	1	0	0	1	0		bit address

Operation: ORL

(C) ←(C)∨(bit)

ORL C, /bit

Bytes: 2

Cycles: 1

Encoding:

1	0	1	0	0	0	0	0		bit address

Operation: ORL

(C) ←(C)∨($\overline{bit}$)

POP direct

Function: Pop from stack

Description: The contents of the internal RAM location addressed by the Stack Pointer is read, and the Stack Pointer is decremented by one. The value read is then transferred to the directly addressed byte indicated. No flags are affected.

Example: The Stack Pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

POP DPH

POP DPL

will leave the Stack Pointer equal to the value 30H and the Data Pointer set to 0123H. At this point the instruction,

POP SP

will leave the Stack Pointer set to 20H. Note that in this special case the Stack Pointer was decremented to 2FH before being loaded with the value popped (20H).

Bytes: 2

Cycles: 1

Encoding:

1	1	0	1	0	0	0	0		direct address

Operation: POP

(direct)←((SP))

(SP) ← (SP) - 1

PUSH direct

Function: Push onto stack

Description: The Stack Pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the Stack Pointer. Otherwise no flags are affected.

Example: On entering an interrupt routine the Stack Pointer contains 09H. The Data Pointer holds the value 0123H. The instruction sequence,

PUSH DPL

PUSH DPH

will leave the Stack Pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

Bytes: 2

Cycles: 1

Encoding:

1	1	0	0	0	0	0	0		direct address

Operation: PUSH

(SP) ←(SP) + 1

((SP)) ← (direct)

RET

Function: Return from subroutine

Description: RET pops the high-and low-order bytes of the PC successively from the stack, decrementing the Stack Pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Example: The Stack Pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RET

will leave the Stack Pointer equal to the value 09H. Program execution will continue at location 0123H.

Bytes: 1

Cycles: 3

Encoding:

0	0	1	0	0	0	1	0

Operation: RET

$(PC_{15-8}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

$(PC_{7-0}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

RETI

Function: Return from interrupt

Description: RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The Stack Pointer is left decremented by two. No other registers are affected; the PSW is not automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

Example: The Stack Pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RETI

will leave the Stack Pointer equal to 09H and return program execution to location 0123H.

Bytes: 1

Cycles: 3

Encoding:

0	0	1	1	0	0	1	0

Operation: RETI

$(PC_{15-8}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

$(PC_{7-0}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

RL A

Function: Rotate Accumulator Left

Description: The eight bits in the Accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

RL A

leaves the Accumulator holding the value 8BH (10001011B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0	0	1	0	0	0	1	1

Operation: RL

(An+1) ← (An) n = 0-6

(A0) ←(A7)

RLC A

Function: Rotate Accumulator Left through the Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

RLC A

leaves the Accumulator holding the value 8BH (10001011B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0	0	1	1	0	0	1	1

Operation: RLC

(An+1) ←(An) n = 0-6

(A0) ←(C)

(C) ←(A7)

RR A

Function: Rotate Accumulator Right

Description: The eight bits in the Accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

RR A

leaves the Accumulator holding the value 0E2H (11100010B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	0	0	1	1

Operation: RR

(An) ←(An+1) n = 0 - 6

(A7) ←(A0)

STC MCU

RRC A

Function: Rotate Accumulator Right through the Carry flag

Description: The eight bits in the Accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

RRC A

leaves the Accumulator holding the value 62H (01100010B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0	0	0	1	0	0	1	1

Operation: RRC

(An+1) ←(An) n = 0 - 6

(A7) ←(C)

(C) ←(A0)

SETB <bit>

Function: Set bit

Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.

Example: The carry flag is cleared. Output Port 1 has been written with the value 34H (00110100B). The instructions,

SETB C

SETB P1.0

will leave the carry flag set to 1 and change the data output on Port 1 to 35H (00110101B).

SETB C

Bytes: 1

Cycles: 1

Encoding:

1	1	0	1	0	0	1	1

Operation: SETB

(C) ← 1

SETB bit

Bytes: 2

Cycles: 1

Encoding:

1	1	0	1	0	0	1	0		bit address

Operation: SETB

(bit) ← 1

SJMP rel

Function: Short Jump

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after

incrementing the PC twice. Therefore, the range of destinations allowed is from 128bytes preceding this instruction to 127 bytes following it.

Example: The label “RELADR” is assigned to an instruction at program memory location 0123H. The instruction,

SJMP RELADR

will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.

(*Note:* Under the above conditions the instruction following SJMP will be at 102H.Therefore, the displacement byte of the instruction will be the relative offset (0123H - 0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be an one-instruction infinite loop).

Bytes: 2

Cycles: 3

Encoding:

1	0	0	0	0	0	0	0		rel. address

Operation: SJMP

(PC) ←(PC)+2

(PC) ← (PC)+rel

SUBB A, <src-byte>

Function: Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the Accumulator, leaving the result in the Accumulator. SUBB sets the carry (borrow)flag if a borrow is needed for bit 7, and clears C otherwise.(If C was set before executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the Accumulator along with the source operand).AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Example: The Accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

SUBB A, R2

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

SUBB A, Rn

Bytes: 1

Cycles: 1

Encoding:

1	0	0	1	1	r	r	r

Operation: SUBB

(A) ← (A) - (C) - (Rn)

SUBB A, direct

Bytes: 2

Cycles: 1

Encoding:

1	0	0	1	0	1	0	1		direct address

Operation: SUBB

(A) ← (A) - (C) - (direct)

SUBB A, @Ri

Bytes: 1

Cycles: 1

Encoding:

1	0	0	1	0	1	1	i

Operation: SUBB

(A) ← (A) - (C) - ((Ri))

SUBB A, #data

Bytes: 2

Cycles: 1

Encoding:

1	0	0	1	0	1	0	0		immediate data

Operation: SUBB

(A) ← (A) - (C) - #data

SWAP A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the Accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction.

No flags are affected.

Example: The Accumulator holds the value 0C5H (11000101B). The instruction,

SWAP A

leaves the Accumulator holding the value 5CH (01011100B).

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	1	0	0

Operation: SWAP

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

XCH A, <byte>

Function: Exchange Accumulator with byte variable

Description: XCH loads the Accumulator with the contents of the indicated variable, at the same time writing the original Accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

Example: R0 contains the address 20H. The Accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCH A, @R0

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

XCH A, Rn

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	1	r	r	r

Operation: XCH

(A) ↔ (Rn)

XCH A, direct

Bytes: 2

Cycles: 1

Encoding:

1	1	0	0	0	1	0	1		direct address

Operation: XCH

(A) ↔ (direct)

XCH A, @Ri

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	1	1	i

Operation: XCH

(A) ↔ ((Ri))

XCHD A, @Ri

Function: Exchange Digit

Description: XCHD exchanges the low-order nibble of the Accumulator (bits 3-0), generally representing a hexadecimal or BCD digit, with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

Example: R0 contains the address 20H. The Accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD A, @R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the accumulator.

Bytes: 1

Cycles: 1

Encoding:

1	1	0	1	0	1	1	i

Operation: XCHD

(A_{3-0}) ↔ (Ri_{3-0})

XRL <dest-byte>, <src-byte>

Function: Logical Exclusive-OR for byte variables

Description: XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the Accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the Accumulator or immediate data.

(*Note*: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, not the input pins.)

Example: If the Accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

XRL A, R0

will leave the Accumulator holding the value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combination of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the Accumulator at run-time. The instruction,

XRL P1, #00110001B

will complement bits 5,4 and 0 of output Port 1.

XRL A, Rn

Bytes: 1

Cycles: 1

Encoding:

0	1	1	0	1	r	r	r

Operation: XRL

(A) ←(A) ∀ (Rn)

XRL A, direct

Bytes: 2

Cycles: 1

Encoding:

0	1	1	0	0	1	0	1		direct address

Operation: XRL

(A) ←(A) ∀ (direct)

XRL A, @Ri

Bytes: 1

Cycles: 1

Encoding:

0	1	1	0	0	1	1	i

Operation: XRL

(A) ←(A) ∀ ((Ri))

XRL A, #data

Bytes: 2

Cycles: 1

Encoding:

0	1	1	0	0	1	0	0		immediate data

Operation: XRL

(A) ←(A) ∀ #data

XRL direct, A

Bytes: 2

Cycles: 1

Encoding:

0	1	1	0	0	0	1	0		direct address

Operation: XRL

(direct) ←(direct) ∀ (A)

XRL direct, #data

Bytes: 3

Cycles: 1

Encoding:

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

Operation: XRL

(direct)← (direct) ∀ #data

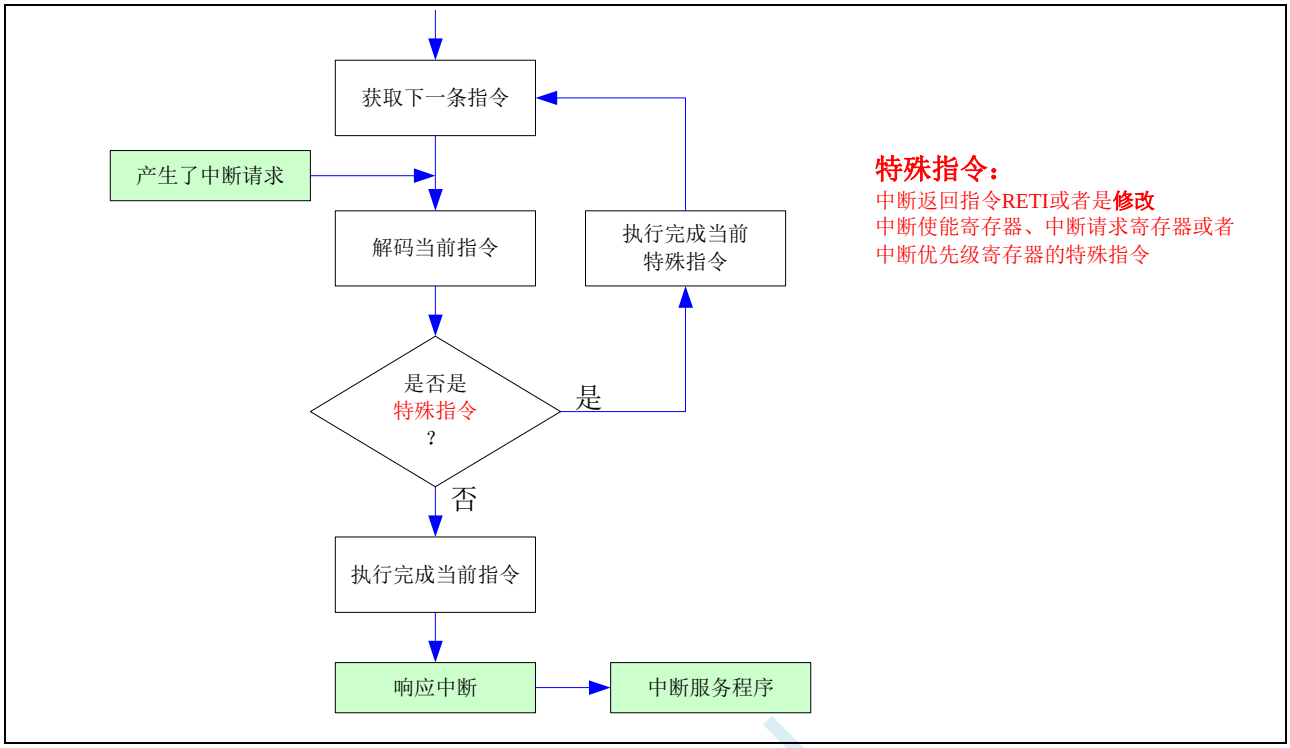
10.5 多级流水线内核的中断响应

STC 的增强型 8051（例如：STC8G/STC8H 系列）和 32 位 8051（例如：STC32G 系列）的 MCU 内核为多级流水线设计，在中断响应方面的设计和传统的 8051（例如：STC89C52 系列）略有差异。

对于传统的 8051（例如：STC89C52 系列）：

如果当前正在执行的指令是中断返回指令 RETI 或者是访问中断使能寄存器、中断请求寄存器或者中断优先级寄存器的特殊指令时，CPU 等当前的这条特殊的指令执行完，再执行一条指令才能响应中断请求；

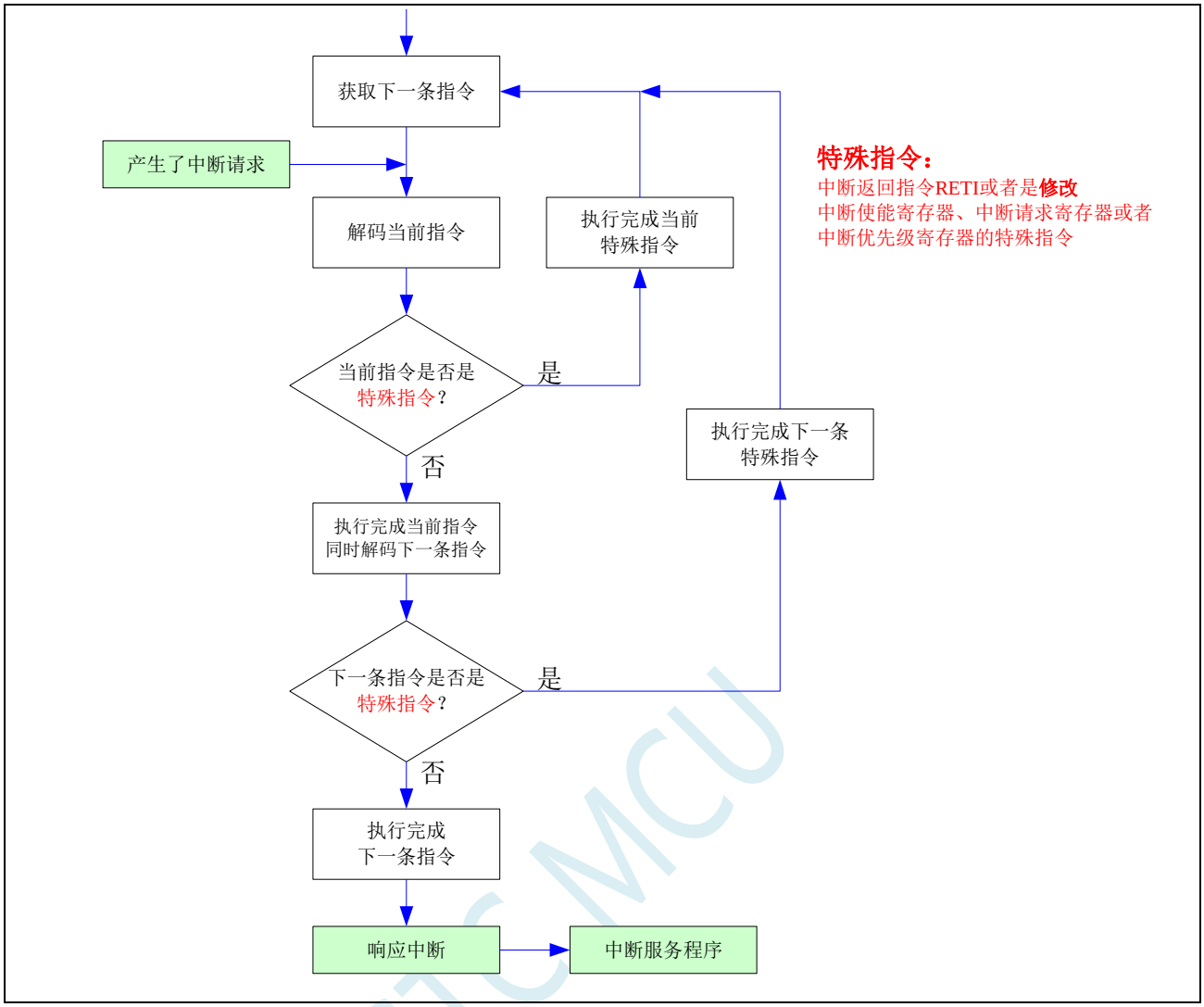
如果当前正在执行的指令不是上面所指的特殊指令，则等当前指令执行完成后就立即响应中断请求；



对于 STC 的增强型 8051 单片机（例如：STC8G/STC8H 系列），由于是多级流水线设计，响应中断上会比传统的 8051（例如：STC89C52 系列）再多执行一条语句：

如果当前正在执行的指令是中断返回指令 RETI 或者是访问中断使能寄存器、中断请求寄存器或者中断优先级寄存器的特殊指令时，CPU 等当前的这条特殊的指令执行完，同时解码下一条指令，直到下一条指令不是特殊指令，则等下一条指令执行完成才能响应中断请求；

如果当前正在执行的指令不是上面所指的特殊指令，则等当前指令执行完成后，同时会解码下一条指令，如果下一条也不是特殊指令，则会等下一条指令执行完成后再立即响应中断请求；


获取下一条指令
产生了中断请求
解码当前指令
执行完成当前
特殊指令
当前指令是否是
特殊指令?
是
否
执行完成下一条
特殊指令
执行完成当前指令
同时解码下一条指令
下一条指令是否是
特殊指令?
是
否
执行完成
下一条指令
响应中断
中断服务程序
特殊指令：
中断返回指令RETI或者是修改
中断使能寄存器、中断请求寄存器或者
中断优先级寄存器的特殊指令

STC MCU

11 中断系统

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

中断系统是为使 CPU 具有对外界紧急事件的实时处理能力而设置的。

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

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

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

11.1 STC8C 系列中断源

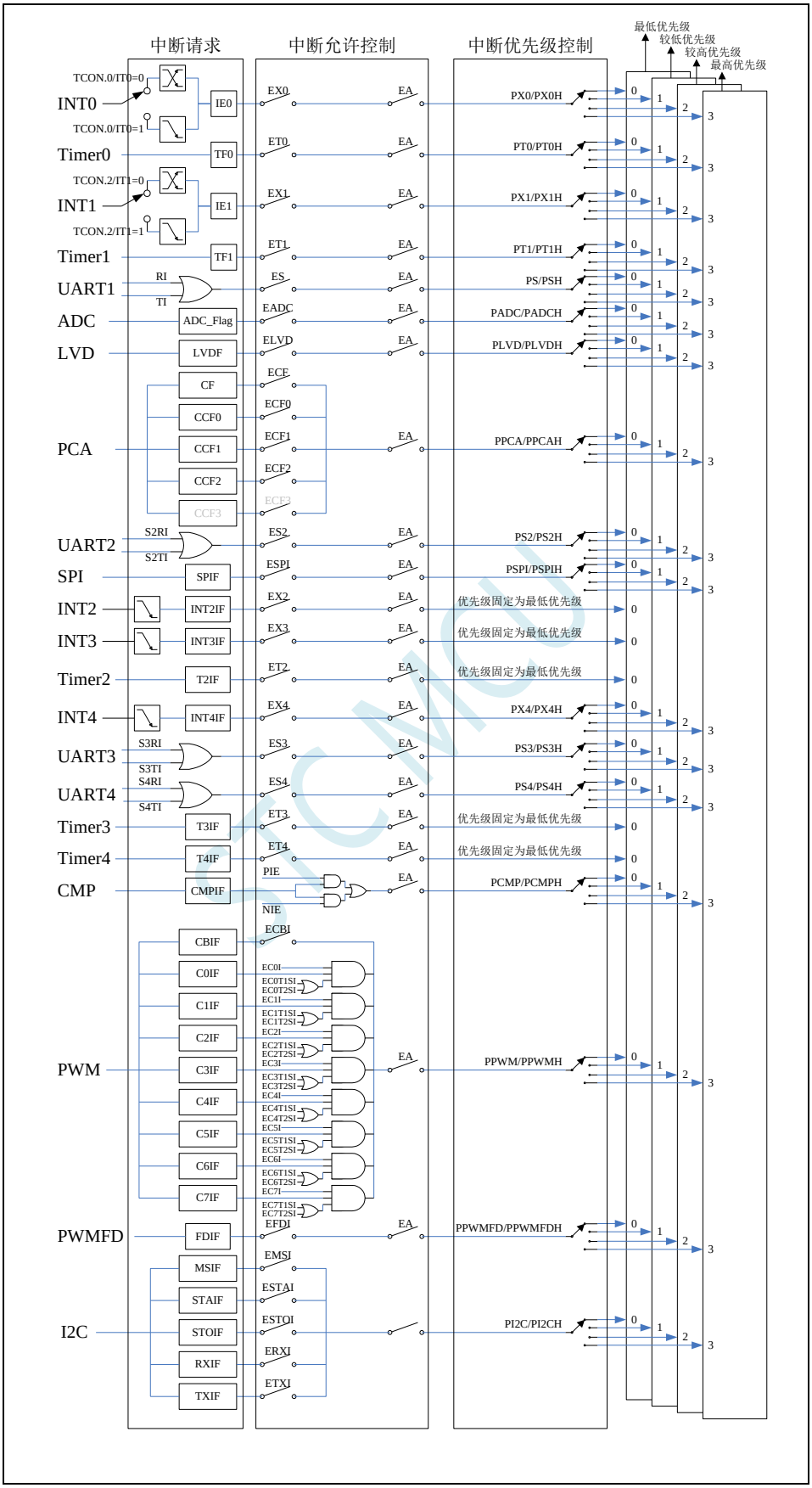
下表中√表示对应的系列有相应的中断源

中断源	STC8C2K64S4系列	STC8C2K64S2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）	√	

中断源	STC8C2K64S4系列	STC8C2K64S2系列
串口 4 中断（UART4）	√	
定时器 3 中断（Timer3）	√	√
定时器 4 中断（Timer4）	√	√
比较器中断（CMP）	√	√
I2C 总线中断	√	√
P0 口中断	√	√
P1 口中断	√	√
P2 口中断	√	√
P3 口中断	√	√
P4 口中断	√	√
P5 口中断	√	√

STC MCU

11.2 STC8C 中断结构图



11.3 STC8C 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
LVD	0033H	6	PLVD,PLVDH	0/1/2/3	LVDF	ELVD
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
P0 中断	012BH	37		0	P0INTF	P0INTE
P1 中断	0133H	38		0	P1INTF	P1INTE
P2 中断	013BH	39		0	P2INTF	P2INTE
P3 中断	0143H	40		0	P3INTF	P3INTE
P4 中断	014BH	41		0	P4INTF	P4INTE
P5 中断	0153H	42		0	P5INTF	P5INTE

在 C 语言中声明中断服务程序

```
void    INT0_Routine(void)      interrupt  0;
void    TM0_Rountine(void)      interrupt  1;
void    INT1_Routine(void)      interrupt  2;
void    TM1_Rountine(void)      interrupt  3;
void    UART1_Routine(void)     interrupt  4;
void    LVD_Routine(void)       interrupt  6;
```

```
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  P0Int_Routine(void)     interrupt   37;
//void  P1Int_Routine(void)     interrupt   38;
//void  P2Int_Routine(void)     interrupt   39;
//void  P3Int_Routine(void)     interrupt   40;
//void  P4Int_Routine(void)     interrupt   41;
//void  P5Int_Routine(void)     interrupt   42;
```

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

STC MCU

11.4 中断相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IE	中断允许寄存器	A8H	EA	ELVD	-	ES	ET1	EX1	ET0	EX0	00x0,0000
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	-	PS	PT1	PX1	PT0	PX0	x0x0,0000
IPH	高中断优先级控制寄存器	B7H	-	PLVDH	-	PSH	PT1H	PX1H	PT0H	PX0H	x0x0,0000
IP2	中断优先级控制寄存器 2	B5H	-	PI2C	PCMP	PX4	-	-	PSPI	PS2	x000,xx00
IP2H	高中断优先级控制寄存器 2	B6H	-	PI2CH	PCMPH	PX4H	-	-	PSPIH	PS2H	x000,xx00
IP3	中断优先级控制寄存器 3	DFH	-	-	-	-	-	-	PS4	PS3	xxxx,xx00
IP3H	高中断优先级控制寄存器 3	EEH	-	-	-	-	-	-	PS4H	PS3H	xxxx,xx00
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
AUXINTIF	扩展外部中断标志寄存器	EFH	-	INT4IF	INT3IF	INT2IF	-	T4IF	T3IF	T2IF	x000,x000
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0000,0000
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
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
TSSTA2	触摸按键状态寄存器 2	FB47H	TSIF	TSDOV	-	-	TSDNCHN[3:0]				00xx,0000
P0INTE	P0 口中断使能寄存器	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE	0000,0000
P1INTE	P1 口中断使能寄存器	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE	0000,0000
P2INTE	P2 口中断使能寄存器	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE	0000,0000
P3INTE	P3 口中断使能寄存器	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE	0000,0000
P4INTE	P4 口中断使能寄存器	FD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE	0000,0000
P5INTE	P5 口中断使能寄存器	FD05H	-	-	-	P54INTE	P53INTE	-	-	-	-
P0INTF	P0 口中断标志寄存器	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF	0000,0000
P1INTF	P1 口中断标志寄存器	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF	0000,0000
P2INTF	P2 口中断标志寄存器	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF	0000,0000
P3INTF	P3 口中断标志寄存器	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF	0000,0000
P4INTF	P4 口中断标志寄存器	FD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF	0000,0000
P5INTF	P5 口中断标志寄存器	FD15H	-	-	-	P54INTF	P53INTF	-	-	-	-

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	-	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

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

STC MCU

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 T2 中断

ESPI：SPI 中断允许位。

0：禁止 SPI 中断

1：允许 SPI 中断

ES2：串行口 2 中断允许位。

0：禁止串行口 2 中断

1：允许串行口 2 中断

INTCLKO（外部中断与时钟输出控制寄存器）

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

EX4：外部中断 4 中断允许位。

0：禁止 INT4 中断

1：允许 INT4 中断

EX3：外部中断 3 中断允许位。

0：禁止 INT3 中断

1：允许 INT3 中断

EX2：外部中断 2 中断允许位。

0：禁止 INT2 中断

1：允许 INT2 中断

CMPCR1（比较器控制寄存器 1）

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

PIE：比较器上升沿中断允许位。

0：禁止比较器上升沿中断

1：允许比较器上升沿中断

NIE：比较器下降沿中断允许位。

0：禁止比较器下降沿中断

1：允许比较器下降沿中断

I2C 控制寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-	MSCMD[3:0]			
I2CSLCR	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

EMSI：I²C主机模式中断允许位。

0：禁止 I²C 主机模式中断

1：允许 I²C 主机模式中断

ESTAI：I²C从机接收START事件中断允许位。

0：禁止 I²C 从机接收 START 事件中断

1：允许 I²C 从机接收 START 事件中断

ERXI：I²C从机接收数据完成事件中断允许位。

0：禁止 I²C 从机接收数据完成事件中断

1：允许 I²C 从机接收数据完成事件中断

ETXI：I²C从机发送数据完成事件中断允许位。

0：禁止 I²C 从机发送数据完成事件中断

1：允许 I²C 从机发送数据完成事件中断

ESTOI：I²C从机接收STOP事件中断允许位。

0：禁止 I²C 从机接收 STOP 事件中断

1：允许 I²C 从机接收 STOP 事件中断

端口中断使能寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0INTE	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE
P1INTE	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE
P2INTE	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE
P3INTE	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE
P4INTE	FD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE
P5INTE	FD05H	-	-	P55INTE	P54INTE	-	-	-	-

PnINTE.x：端口中断使能控制位（n=0~7，x=0~7）

0：关闭 Pn.x 口中断功能

1：使能 Pn.x 口中断功能

11.4.2 中断请求寄存器（中断标志位）

定时器控制寄存器

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

TF1：定时器1溢出中断标志。中断服务程序中，硬件自动清零。

TF0：定时器0溢出中断标志。中断服务程序中，硬件自动清零。

IE1：外部中断1中断请求标志。中断服务程序中，硬件自动清零。

IE0：外部中断0中断请求标志。中断服务程序中，硬件自动清零。

中断标志辅助寄存器

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

INT4IF：外部中断4中断请求标志。中断服务程序中硬件自动清零。

INT3IF：外部中断3中断请求标志。中断服务程序中硬件自动清零。

INT2IF：外部中断2中断请求标志。中断服务程序中硬件自动清零。

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

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

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

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。

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。

端口中断标志寄存器

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0INTF	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF
P1INTF	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF
P2INTF	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF
P3INTF	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF
P4INTF	FD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF
P5INTF	FD15H	-	-	-	P54INTF	P53INTF	P52INTF	P51INTF	P50INTF
P6INTF	FD16H	P67INTF	P66INTF	P65INTF	P64INTF	P63INTF	P62INTF	P61INTF	P60INTF
P7INTF	FD17H	P77INTF	P76INTF	P75INTF	P74INTF	P73INTF	P72INTF	P71INTF	P70INTF

PnINTF.x：端口中断请求标志位（n=0~7，x=0~7）

0：Pn.x 口没有中断请求

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

11.4.3 中断优先级寄存器

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

中断优先级控制寄存器

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

PX0H,PX0：外部中断0中断优先级控制位

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

PT0H,PT0：定时器0中断优先级控制位

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

PX1H,PX1：外部中断1中断优先级控制位

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

PT1H,PT1：定时器1中断优先级控制位

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

PSH,PS：串口1中断优先级控制位

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

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

STC MCU

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）

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）

PS3H,PS3：串口3中断优先级控制位

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

PS4H,PS4：串口4中断优先级控制位

00：串口 4 中断优先级为 0 级（最低级）
01：串口 4 中断优先级为 1 级（较低级）
10：串口 4 中断优先级为 2 级（较高级）
11：串口 4 中断优先级为 3 级（最高级）

STC MCU

11.5 范例程序

11.5.1 INT0 中断（上升沿和下降沿），可同时支持上升沿和下降沿

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

void INT0_Isr() interrupt 0
{
    if (P32)                                    //判断上升沿和下降沿
    {
        P10 = !P10;                             //测试端口
    }
    else
    {
        P11 = !P11;                             //测试端口
    }
}

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

    IT0 = 0;                                    //使能 INT0 上升沿和下降沿中断
    EX0 = 1;                                    //使能 INT0 中断
    EA = 1;
```

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

汇编代码

```
;测试工作频率为 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     0003H
            LJMP    INT0ISR

            ORG     0100H
INT0ISR:
            JB      P3.2,RISING         ;判断上升沿和下降沿
            CPL     P1.0                ;测试端口
            RETI
RISING:
            CPL     P1.1                ;测试端口
            RETI

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

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

            END
```

11.5.2 INT0 中断（下降沿）

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sbit        P10         =   P1^0;

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

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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
```

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

            ORG         0000H
            LJMP        MAIN
            ORG         0003H
            LJMP        INT0ISR

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

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

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

            END
```

11.5.3 INT1 中断（上升沿和下降沿），可同时支持上升沿和下降沿

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

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

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

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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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

```
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0013H
            LJMP    INT1ISR

            ORG     0100H
INT1ISR:
            JB      INT1,RISING         ;判断上升沿和下降沿
            CPL     P1.0                ;测试端口
            RETI
RISING:
            CPL     P1.1                ;测试端口
            RETI

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

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

            END
```

11.5.4 INT1 中断（下降沿）

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

sbit    P10     =   P1^0;

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

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

    IT1 = 1;                                    //使能 INT1 下降沿中断
    EX1 = 1;                                    //使能 INT1 中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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

            ORG         0000H
            LJMP        MAIN
            ORG         0013H
            LJMP        INT1ISR

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

```
        RETI

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

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

        END
```

11.5.5 INT2 中断（下降沿），只支持下降沿中断

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

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

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

    INTCLKO = EX2;                          //使能 INT2 中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     0053H
            LJMP    INT2ISR

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

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

STC MCU

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

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

        END
```

11.5.6 INT3 中断（下降沿），只支持下降沿中断

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

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

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

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

    INTCLKO = EX3;                          //使能 INT3 中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     005BH
            LJMP    INT3ISR

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

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

STC MCU

```
        MOV     P5M1, #00H

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

        END
```

11.5.7　INT4 中断（下降沿），只支持下降沿中断

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

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

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

```
    INTCLKO = EX4;                          //使能 INT4 中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     0083H
            LJMP    INT4ISR

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

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

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

            END
```

STC MCU

11.5.8 定时器 0 中断

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

sbit        P10         =   P1^0;

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

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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

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

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

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

            JMP     $

            END
```

11.5.9 定时器 1 中断

C 语言代码

```
//测试工作频率为 11.0592MHz
```

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

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

sbit    P10     =   P1^0;

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

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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
```

```
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     001BH
            LJMP    TM1ISR

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

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

            MOV     TMOD,#00H
            MOV     TL1,#66H            ;65536-11.0592M/12/1000
            MOV     TH1,#0FCH
            SETB    TR1                 ;启动定时器
            SETB    ET1                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

11.5.10 定时器 2 中断

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
```

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

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

sbit      P10       =  P1^0;

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

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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     0063H
            LJMP    TM2ISR

            ORG     0100H
TM2ISR:
            CPL     P1.0                ;测试端口
            RETI

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

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

            JMP     $

            END
```

11.5.11 定时器 3 中断

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
```

```
#include "intrins.h"

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

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

sbit        P10             =   P1^0;

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

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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T3L             DATA            0D5H
```

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     009BH
            LJMP    TM3ISR

            ORG     0100H
TM3ISR:
            CPL     P1.0                ;测试端口
            RETI

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

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

            JMP     $

            END
```

STC MCU

11.5.12 定时器 4 中断

C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr         T3L             =   0xd5;
sfr         T3H             =   0xd4;
sfr         T4L             =   0xd3;
sfr         T4H             =   0xd2;
sfr         T4T3M           =   0xd1;
sfr         IE2             =   0xaf;
#define     ET3                 0x20
#define     ET4                 0x40
sfr         AUXINTIF        =   0xef;
#define     T3IF                0x02
#define     T4IF                0x04

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

sbit        P10             =   P1^0;

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

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

    T4L = 0x66;                         //65536-11.0592M/12/1000
    T4H = 0xfc;
```

```
    T4T3M = 0x80;                   //启动定时器
    IE2 = ET4;                      //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T3L         DATA    0D5H
T3H         DATA    0D4H
T4L         DATA    0D3H
T4H         DATA    0D2H
T4T3M       DATA    0D1H
IE2         DATA    0AFH
ET3         EQU     20H
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T3IF        EQU     02H
T4IF        EQU     04H

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

            ORG     0000H
            LJMP    MAIN
            ORG     00A3H
            LJMP    TM4ISR

            ORG     0100H
TM4ISR:
            CPL     P1.0                ;测试端口
            RETI

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

STC MCU

```
        MOV     P5M1, #00H

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

        JMP     $

        END
```

11.5.13 UART1 中断

C 语言代码

```
//测试工作频率为 11.0592MHz

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

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

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

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

void UART1_Isr() interrupt 4
{
    if (TI)
    {
        TI = 0;                         //清中断标志
        P10 = !P10;                     //测试端口
    }
    if (RI)
    {
        RI = 0;                         //清中断标志
        P11 = !P11;                     //测试端口
    }
}

void main()
```

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

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

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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

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

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART1ISR

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

```
            CPL     P1.1                ;测试端口
ISREXIT:
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     SCON,#50H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#15H           ;启动定时器
            SETB    ES                  ;使能串口中断
            SETB    EA
            MOV     SBUF,#5AH           ;发送测试数据

            JMP     $

            END
```

11.5.14 UART2 中断

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     S2CON   =   0x9a;
sfr     S2BUF   =   0x9b;
sfr     IE2     =   0xaf;
#define ES2         0x01

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
```

```
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P12     =   P1^2;
sbit    P13     =   P1^3;

void UART2_Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;                 //清中断标志
        P12 = !P12;                     //测试端口
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;                 //清中断标志
        P13 = !P13;                     //测试端口
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    S2CON = 0x10;
    T2L = 0xe8;                         //65536-11059200/115200/4=0FFE8H
    T2H = 0xff;
    AUXR = 0x14;                        //启动定时器
    IE2 = ES2;                          //使能串口中断
    EA = 1;
    S2BUF = 0x5a;                       //发送测试数据

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T2L     DATA    0D7H
T2H     DATA    0D6H
AUXR    DATA    8EH
S2CON   DATA    9AH
S2BUF   DATA    9BH
IE2     DATA    0AFH
ES2     EQU     01H
```

```
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0043H
            LJMP    UART2ISR

            ORG     0100H
UART2ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     A,S2CON
            JNB     ACC.1,CHECKRI
            ANL     S2CON,#NOT 02H      ;清中断标志
            CPL     P1.2                ;测试端口
CHECKRI:
            MOV     A,S2CON
            JNB     ACC.0,ISREXIT
            ANL     S2CON,#NOT 01H      ;清中断标志
            CPL     P1.3                ;测试端口
ISREXIT:
            POP     PSW
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     S2CON,#10H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H           ;启动定时器
            MOV     IE2,#ES2            ;使能串口中断
            SETB    EA
            MOV     S2BUF,#5AH          ;发送测试数据
```

```
        JMP     $

        END
```

11.5.15 UART3 中断

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     S3CON   =   0xac;
sfr     S3BUF   =   0xad;
sfr     IE2     =   0xaf;
#define ES3     0x08

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P12     =   P1^2;
sbit    P13     =   P1^3;

void UART3_Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;                 //清中断标志
        P12 = !P12;                     //测试端口
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;                 //清中断标志
        P13 = !P13;                     //测试端口
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
```

```
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    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);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH
ES3         EQU     08H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3ISR

            ORG     0100H
UART3ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     A,S3CON
            JNB     ACC.1,CHECKRI
            ANL     S3CON,#NOT 02H      ;清中断标志
```

```
                CPL     P1.2                    ;测试端口
CHECKRI:
                MOV     A,S3CON
                JNB     ACC.0,ISREXIT
                ANL     S3CON,#NOT 01H          ;清中断标志
                CPL     P1.3                    ;测试端口
ISREXIT:
                POP     PSW
                POP     ACC
                RETI

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                MOV     S3CON,#10H
                MOV     T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     T2H,#0FFH
                MOV     AUXR,#14H               ;启动定时器
                MOV     IE2,#ES3                ;使能串口中断
                SETB    EA
                MOV     S3BUF,#5AH              ;发送测试数据

                JMP     $

                END
```

11.5.16 UART4 中断

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     S4CON   =   0x84;
sfr     S4BUF   =   0x85;
sfr     IE2     =   0xaf;
#define ES4         0x10

sfr     P0M1    =   0x93;
```

```
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P12     =   P1^2;
sbit    P13     =   P1^3;

void UART4_Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;                 //清中断标志
        P12 = !P12;                     //测试端口
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;                 //清中断标志
        P13 = !P13;                     //测试端口
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    S4CON = 0x10;
    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);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz
```

```
T2L     DATA    0D7H
T2H     DATA    0D6H
AUXR    DATA    8EH
S4CON   DATA    84H
S4BUF   DATA    85H
IE2     DATA    0AFH
ES4     EQU     10H

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     0093H
        LJMP    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     P1.2                ;测试端口
CHECKRI:
        MOV     A,S4CON
        JNB     ACC.0,ISREXIT
        ANL     S4CON,#NOT 01H      ;清中断标志
        CPL     P1.3                ;测试端口
ISREXIT:
        POP     PSW
        POP     ACC
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H
```

STC MCU

```
        MOV     S4CON,#10H
        MOV     T2L,#0E8H               ;65536-11059200/115200/4=0FFE8H
        MOV     T2H,#0FFH
        MOV     AUXR,#14H               ;启动定时器
        MOV     IE2,#ES4                ;使能串口中断
        SETB    EA
        MOV     S4BUF,#5AH              ;发送测试数据

        JMP     $

        END
```

11.5.17 LVD 中断

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     RSTCFG  =   0xff;
#define ENLVR       0x40            //RSTCFG.6
#define LVD2V2      0x00            //LVD@2.2V
#define LVD2V4      0x01            //LVD@2.4V
#define LVD2V7      0x02            //LVD@2.7V
#define LVD3V0      0x03            //LVD@3.0V
sbit    ELVD    =   IE^6;
#define LVDF        0x20            //PCON.5

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;
sbit    P10     =   P1^0;

void LVD_Isr() interrupt 6
{
    PCON &= ~LVDF;                  //清中断标志
    P10 = !P10;                     //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
```

```
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    PCON &= ~LVDF;                  //上电需要清中断标志
    RSTCFG = LVD3V0;                //设置 LVD 电压为 3.0V
    ELVD = 1;//使能 LVD 中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

RSTCFG      DATA    0FFH
ENLVR       EQU     40H             ;RSTCFG.6
LVD2V2      EQU     00H             ;LVD@2.2V
LVD2V4      EQU     01H             ;LVD@2.4V
LVD2V7      EQU     02H             ;LVD@2.7V
LVD3V0      EQU     03H             ;LVD@3.0V
ELVD        BIT     IE.6
LVDF        EQU     20H             ;PCON.5

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0033H
            LJMP    LVDISR

            ORG     0100H
LVDISR:
            ANL     PCON,#NOT LVDF  ;清中断标志
            CPL     P1.0            ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
```

```
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        ANL     PCON,#NOT LVDF      ;上电需要清中断标志
        MOV     RSTCFG,# LVD3V0     ;设置 LVD 电压为 3.0V
        SETB    ELVD                ;使能 LVD 中断
        SETB    EA
        JMP     $

        END
```

11.5.18 SPI 中断

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
sfr     SPDAT       =   0xcf;
sfr     IE2         =   0xaf;
#define ESPI            0x02

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    P10         =   P1^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //清中断标志
    P10 = !P10;                     //测试端口
}

void main()
{
```

```
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x50;                   //使能 SPI 主机模式
    SPSTAT = 0xc0;                  //清中断标志
    IE2 = ESPI;                     //使能 SPI 中断
    EA = 1;
    SPDAT = 0x5a;                   //发送测试数据

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H    ;清中断标志
            CPL     P1.0            ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
```

STC MCU

```
MOV     P0M1, #00H
MOV     P1M0, #00H
MOV     P1M1, #00H
MOV     P2M0, #00H
MOV     P2M1, #00H
MOV     P3M0, #00H
MOV     P3M1, #00H
MOV     P4M0, #00H
MOV     P4M1, #00H
MOV     P5M0, #00H
MOV     P5M1, #00H

MOV     SPCTL,#50H          ;使能 SPI 主机模式
MOV     SPSTAT,#0C0H        ;清中断标志
MOV     IE2,#ESPI           ;使能 SPI 中断
SETB    EA
MOV     SPDAT,#5AH          ;发送测试数据

JMP     $

END
```

11.5.19 比较器中断

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     CMPCR1  =   0xe6;
sfr     CMPCR2  =   0xe7;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                    //清中断标志
    P10 = !P10;                         //测试端口
}

void main()
```

```
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CMPCR2 = 0x00;
    CMPCR1 = 0x80;                  //使能比较器模块
    CMPCR1 |= 0x30;                 //使能比较器边沿中断
    CMPCR1 &= ~0x08;                //P3.6 为 CMP+输入脚
    CMPCR1 |= 0x04;                 //P3.7 为 CMP-输入脚
    CMPCR1 |= 0x02;                 //使能比较器输出
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR

            ORG     0100H
CMPISR:
            ANL     CMPCR1,#NOT 40H     ;清中断标志
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
```

```
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     CMPCR2,#00H
        MOV     CMPCR1,#80H             ;使能比较器模块
        ORL     CMPCR1,#30H             ;使能比较器边沿中断
        ANL     CMPCR1,#NOT 08H         ;P3.6 为CMP+输入脚
        ORL     CMPCR1,#04H             ;P3.7 为CMP-输入脚
        ORL     CMPCR1,#02H             ;使能比较器输出
        SETB    EA

        JMP     $

        END
```

11.5.20 I2C 中断

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#d
efine   I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
```

```
sfr     P5M0    =   0xca;
sbit    P10     =   P1^0;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;                   //清中断标志
        P10 = !P10;                         //测试端口
    }
    _pop_(P_SW2);
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;
    I2CCFG = 0xc0;                          //使能 I2C 主机模式
    I2CMSCR = 0x80;                         //使能 I2C 中断;
    P_SW2 = 0x00;
    EA = 1;

    P_SW2 = 0x80;
    I2CMSCR = 0x81;                         //发送起始命令
    P_SW2 = 0x00;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

P0M1        DATA    093H
```

```
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    DPL
            PUSH    DPH
            PUSH    P_SW2
            MOV     P_SW2,#80H
            MOV     DPTR,#I2CMSST
            MOVX    A,@DPTR
            ANL     A,#NOT 40H          ;清中断标志
            MOVX    @DPTR,A
            CPL     P1.0                ;测试端口
            POP     P_SW2
            POP     DPH
            POP     DPL
            POP     ACC
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW2,#80H
            MOV     A,#0C0H             ;使能 I2C 主机模式
            MOV     DPTR,#I2CCFG
            MOVX    @DPTR,A
            MOV     A,#80H              ;使能 I2C 中断
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            MOV     P_SW2,#00H
            SETB    EA
```

```
MOV     P_SW2,#80H
MOV     A,#081H                 ;发送起始命令
MOV     DPTR,#I2CMSCR
MOVX    @DPTR,A
MOV     P_SW2,#00H

JMP     $

END
```

STC MCU

12 普通 I/O 口均可中断，不是传统外部中断

STC8C 系列支持所有的 I/O 中断，且支持 4 种中断模式：下降沿中断、上升沿中断、低电平中断、高电平中断。每组 I/O 口都有独立的中断入口地址，且每个 I/O 可独立设置中断模式。

12.1 I/O 口中断相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
P0INTE	P0 口中断使能寄存器	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE	0000,0000
P1INTE	P1 口中断使能寄存器	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE	0000,0000
P2INTE	P2 口中断使能寄存器	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE	0000,0000
P3INTE	P3 口中断使能寄存器	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE	0000,0000
P4INTE	P4 口中断使能寄存器	FD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE	0000,0000
P5INTE	P5 口中断使能寄存器	FD05H	-	-	P55INTE	P54INTE	-	-	-	-	xx00,xxxx
P0INTF	P0 口中断标志寄存器	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF	0000,0000
P1INTF	P1 口中断标志寄存器	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF	0000,0000
P2INTF	P2 口中断标志寄存器	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF	0000,0000
P3INTF	P3 口中断标志寄存器	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF	0000,0000
P4INTF	P4 口中断标志寄存器	FD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF	0000,0000
P5INTF	P5 口中断标志寄存器	FD15H	-	-	P55INTF	P54INTF	-	-	-	-	xx00,xxxx
P0IM0	P0 口中断模式寄存器 0	FD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0	0000,0000
P1IM0	P1 口中断模式寄存器 0	FD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0	0000,0000
P2IM0	P2 口中断模式寄存器 0	FD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0	0000,0000
P3IM0	P3 口中断模式寄存器 0	FD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0	0000,0000
P4IM0	P4 口中断模式寄存器 0	FD24H	P47IM0	P46IM0	P45IM0	P44IM0	P43IM0	P42IM0	P41IM0	P40IM0	0000,0000
P5IM0	P5 口中断模式寄存器 0	FD25H	-	-	P55IM0	P54IM0	-	-	-	-	xx00,xxxx
P0IM1	P0 口中断模式寄存器 1	FD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1	0000,0000
P1IM1	P1 口中断模式寄存器 1	FD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1	0000,0000
P2IM1	P2 口中断模式寄存器 1	FD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1	0000,0000
P3IM1	P3 口中断模式寄存器 1	FD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1	0000,0000
P4IM1	P4 口中断模式寄存器 1	FD34H	P47IM1	P46IM1	P45IM1	P44IM1	P43IM1	P42IM1	P41IM1	P40IM1	0000,0000
P5IM1	P5 口中断模式寄存器 1	FD35H	-	-	P55IM1	P54IM1	-	-	-	-	xx00,xxxx

12.1.1 端口中断使能寄存器（PxINTE）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0INTE	FD00H	P07INTE	P06INTE	P05INTE	P04INTE	P03INTE	P02INTE	P01INTE	P00INTE
P1INTE	FD01H	P17INTE	P16INTE	P15INTE	P14INTE	P13INTE	P12INTE	P11INTE	P10INTE
P2INTE	FD02H	P27INTE	P26INTE	P25INTE	P24INTE	P23INTE	P22INTE	P21INTE	P20INTE
P3INTE	FD03H	P37INTE	P36INTE	P35INTE	P34INTE	P33INTE	P32INTE	P31INTE	P30INTE
P4INTE	FD04H	P47INTE	P46INTE	P45INTE	P44INTE	P43INTE	P42INTE	P41INTE	P40INTE
P5INTE	FD05H	-	-	P55INTE	P54INTE	-	-	-	-

PnINTE.x：端口中断使能控制位（n=0~7，x=0~7）

0：关闭 Pn.x 口中断功能

1：使能 Pn.x 口中断功能

12.1.2 端口中断标志寄存器（PxINTF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0INTF	FD10H	P07INTF	P06INTF	P05INTF	P04INTF	P03INTF	P02INTF	P01INTF	P00INTF
P1INTF	FD11H	P17INTF	P16INTF	P15INTF	P14INTF	P13INTF	P12INTF	P11INTF	P10INTF
P2INTF	FD12H	P27INTF	P26INTF	P25INTF	P24INTF	P23INTF	P22INTF	P21INTF	P20INTF
P3INTF	FD13H	P37INTF	P36INTF	P35INTF	P34INTF	P33INTF	P32INTF	P31INTF	P30INTF
P4INTF	FD14H	P47INTF	P46INTF	P45INTF	P44INTF	P43INTF	P42INTF	P41INTF	P40INTF
P5INTF	FD15H	-	-	P55INTF	P54INTF	-	-	-	-

PnINTF.x：端口中断请求标志位（n=0~7，x=0~7）

0：Pn.x 口没有中断请求

1：Pn.x 口有中断请求，若使能中断，则会进入中断服务程序。标志位需软件清 0。

12.1.3 端口中断模式配置寄存器（PxIM0，PxIM1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
P0IM0	FD20H	P07IM0	P06IM0	P05IM0	P04IM0	P03IM0	P02IM0	P01IM0	P00IM0
P0IM1	FD30H	P07IM1	P06IM1	P05IM1	P04IM1	P03IM1	P02IM1	P01IM1	P00IM1
P1IM0	FD21H	P17IM0	P16IM0	P15IM0	P14IM0	P13IM0	P12IM0	P11IM0	P10IM0
P1IM1	FD31H	P17IM1	P16IM1	P15IM1	P14IM1	P13IM1	P12IM1	P11IM1	P10IM1
P2IM0	FD22H	P27IM0	P26IM0	P25IM0	P24IM0	P23IM0	P22IM0	P21IM0	P20IM0
P2IM1	FD32H	P27IM1	P26IM1	P25IM1	P24IM1	P23IM1	P22IM1	P21IM1	P20IM1
P3IM0	FD23H	P37IM0	P36IM0	P35IM0	P34IM0	P33IM0	P32IM0	P31IM0	P30IM0
P3IM1	FD33H	P37IM1	P36IM1	P35IM1	P34IM1	P33IM1	P32IM1	P31IM1	P30IM1
P4IM0	FD24H	P47IM0	P46IM0	P45IM0	P44IM0	P43IM0	P42IM0	P41IM0	P40IM0
P4IM1	FD34H	P47IM1	P46IM1	P45IM1	P44IM1	P43IM1	P42IM1	P41IM1	P40IM1
P5IM0	FD25H	-	-	P55IM0	P54IM0	-	-	-	-
P5IM1	FD35H	-	-	P55IM1	P54IM1	-	-	-	-

配置端口的模式

PnIM1.x	PnIM0.x	Pn.x 口中断模式
0	0	下降沿中断
0	1	上升沿中断
1	0	低电平中断
1	1	高电平中断

STC MCU

12.2 范例程序

12.2.1 P0 口下降沿中断

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P0M0        =   0x94;
sfr         P0M1        =   0x93;
sfr         P1M0        =   0x92;
sfr         P1M1        =   0x91;
sfr         P2M0        =   0x96;
sfr         P2M1        =   0x95;
sfr         P3M0        =   0xb2;
sfr         P3M1        =   0xb1;
sfr         P4M0        =   0xb4;
sfr         P4M1        =   0xb3;
sfr         P5M0        =   0xca;
sfr         P5M1        =   0xc9;
sfr         P6M0        =   0xcc;
sfr         P6M1        =   0xcb;
sfr         P7M0        =   0xe2;
sfr         P7M1        =   0xe1;

sfr         P_SW2       =   0xba;

#define     P0INTE      (*(unsigned char volatile xdata *)0xfd00)
#define     P0INTF      (*(unsigned char volatile xdata *)0xfd10)
#define     P0IM0       (*(unsigned char volatile xdata *)0xfd20)
#define     P0IM1       (*(unsigned char volatile xdata *)0xfd30)

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 |= 0x80;
    P0IM0 = 0x00;                   //下降沿中断
    P0IM1 = 0x00;
    P0INTE = 0xff;                  //使能 P0 口中断
    P_SW2 &= ~0x80;

    EA = 1;
```

```
    while (1);
}

//由于中断向量大于31，在KEIL 中无法直接编译
//必须借用第13 号中断入口地址
void common_isr() interrupt 13
{
    unsigned char psw2_st;
    unsigned char intf;

    psw2_st = P_SW2;
    P_SW2 |= 0x80;
    intf = P0INTF;
    if (intf)
    {
        P0INTF = 0x00;
        if (intf & 0x01)
        {
                                                    //P0.0 口中断
        }
        if (intf & 0x02)
        {
                                                    //P0.1 口中断
        }
        if (intf & 0x04)
        {
                                                    //P0.2 口中断
        }
        if (intf & 0x08)
        {
                                                    //P0.3 口中断
        }
        if (intf & 0x10)
        {
                                                    //P0.4 口中断
        }
        if (intf & 0x20)
        {
                                                    //P0.5 口中断
        }
        if (intf & 0x40)
        {
                                                    //P0.6 口中断
        }
        if (intf & 0x80)
        {
                                                    //P0.7 口中断
        }
    }
    P_SW2 = psw2_st;
}

// ISR.ASM
//将下面的代码保存为ISR.ASM，然后将文件加入到项目中即可

            CSEG        AT 012BH                ;P0 口中断入口地址
            JMP         P0INT_ISR
P0INT_ISR:
```

```
        JMP     006BH                   ;借用 13 号中断的入口地址
        END
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M0    DATA    094H
P0M1    DATA    093H
P1M0    DATA    092H
P1M1    DATA    091H
P2M0    DATA    096H
P2M1    DATA    095H
P3M0    DATA    0B2H
P3M1    DATA    0B1H
P4M0    DATA    0B4H
P4M1    DATA    0B3H
P5M0    DATA    0CAH
P5M1    DATA    0C9H
P6M0    DATA    0CCH
P6M1    DATA    0CBH
P7M0    DATA    0E2H
P7M1    DATA    0E1H

P_SW2   DATA    0BAH

P0INTE  XDATA   0FD00H
P0INTF  XDATA   0FD10H
P0IM0   XDATA   0FD20H
P0IM1   XDATA   0FD30H

        ORG     0000H
        LJMP    MAIN

        ORG     012BH                   ;P0 口中断入口地址
P0INT_ISR:
        PUSH    ACC
        PUSH    B
        PUSH    DPL
        PUSH    DPH
        PUSH    P_SW2

        MOV     DPTR,#P0INTF
        MOVX    A,@DPTR
        MOV     B,A
        CLR     A
        MOVX    @DPTR,A
        MOV     A,B
CHECKP00:
        JNB     ACC.0,CHECKP01
        NOP                             ;P0.0 口中断
CHECKP01:
        JNB     ACC.1,CHECKP02
        NOP                             ;P0.1 口中断
CHECKP02:
        JNB     ACC.2,CHECKP03
        NOP                             ;P0.2 口中断
```

STC MCU

```
CHECKP03
        JNB     ACC.3,CHECKP04
        NOP                             ;P0.3 口中断
CHECKP04:
        JNB     ACC.4,CHECKP05
        NOP                             ;P0.4 口中断
CHECKP05:
        JNB     ACC.5,CHECKP06
        NOP                             ;P0.5 口中断
CHECKP06:
        JNB     ACC.6,CHECKP07
        NOP                             ;P0.6 口中断
CHECKP07:
        JNB     ACC.7,P0ISREXIT
        NOP                             ;P0.7 口中断

P0ISREXIT:
        POP     P_SW2
        POP     DPH
        POP     DPL
        POP     B
        POP     ACC
        RETI

        ORG     0200H
MAIN:
        MOV     SP, #5FH

        MOV     P0M0,#00H
        MOV     P0M1,#00H
        MOV     P1M0,#00H
        MOV     P1M1,#00H
        MOV     P2M0,#00H
        MOV     P2M1,#00H
        MOV     P3M0,#00H
        MOV     P3M1,#00H

        ORL     P_SW2,#80H
        CLR     A
        MOV     DPTR,# P0IM0            ; 下降沿中断
        MOVX    @DPTR,A
        MOV     DPTR,# P0IM1
        MOVX    @DPTR,A
        MOV     DPTR,# P0INTE
        MOV     A,#0FFH
        MOVX    @DPTR,A                 ;使能 P0 口中断
        ANL     P_SW2,#7FH

        SETB    EA

        JMP     $

        END
```

12.2.2 P1 口上升沿中断

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M0    =   0x94;
sfr     P0M1    =   0x93;
sfr     P1M0    =   0x92;
sfr     P1M1    =   0x91;
sfr     P2M0    =   0x96;
sfr     P2M1    =   0x95;
sfr     P3M0    =   0xb2;
sfr     P3M1    =   0xb1;
sfr     P4M0    =   0xb4;
sfr     P4M1    =   0xb3;
sfr     P5M0    =   0xca;
sfr     P5M1    =   0xc9;
sfr     P6M0    =   0xcc;
sfr     P6M1    =   0xcb;
sfr     P7M0    =   0xe2;
sfr     P7M1    =   0xe1;

sfr     P_SW2   =   0xba;

#define P1INTE  (*(unsigned char volatile xdata *)0xfd01)
#define P1INTF  (*(unsigned char volatile xdata *)0xfd11)
#define P1IM0   (*(unsigned char volatile xdata *)0xfd21)
#define P1IM1   (*(unsigned char volatile xdata *)0xfd31)

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 |= 0x80;
    P1IM0 = 0xff;                   //上升沿中断
    P1IM1 = 0x00;
    P1INTE = 0xff;                  //使能 P1 口中断
    P_SW2 &= ~0x80;

    EA = 1;

    while (1);
}

//由于中断向量大于 31，在 KEIL 中无法直接编译
//必须借用第 13 号中断入口地址
void common_isr() interrupt 13
```

```
{
    unsigned char psw2_st;
    unsigned char intf;

    psw2_st = P_SW2;
    P_SW2 |= 0x80;
    intf = P1INTF;
    if (intf)
    {
        P1INTF = 0x00;
        if (intf & 0x01)
        {
                                        //P1.0 口中断
        }
        if (intf & 0x02)
        {
                                        //P1.1 口中断
        }
        if (intf & 0x04)
        {
                                        //P1.2 口中断
        }
        if (intf & 0x08)
        {
                                        //P1.3 口中断
        }
        if (intf & 0x10)
        {
                                        //P1.4 口中断
        }
        if (intf & 0x20)
        {
                                        //P1.5 口中断
        }
        if (intf & 0x40)
        {
                                        //P1.6 口中断
        }
        if (intf & 0x80)
        {
                                        //P1.7 口中断
        }
    }
    P_SW2 = psw2_st;
}

// ISR.ASM
//将下面的代码保存为ISR.ASM，然后将文件加入到项目中即可

            CSEG    AT 0133H            ;P1 口中断入口地址
            JMP     P1INT_ISR
P1INT_ISR:
            JMP     006BH               ;借用 13 号中断的入口地址
            END
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M0        DATA    094H
P0M1        DATA    093H
P1M0        DATA    092H
P1M1        DATA    091H
P2M0        DATA    096H
P2M1        DATA    095H
P3M0        DATA    0B2H
P3M1        DATA    0B1H
P4M0        DATA    0B4H
P4M1        DATA    0B3H
P5M0        DATA    0CAH
P5M1        DATA    0C9H
P6M0        DATA    0CCH
P6M1        DATA    0CBH
P7M0        DATA    0E2H
P7M1        DATA    0E1H

P_SW2       DATA    0BAH

P1INTE      XDATA   0FD01H
P1INTF      XDATA   0FD11H
P1IM0       XDATA   0FD21H
P1IM1       XDATA   0FD31H

            ORG     0000H
            LJMP    MAIN

            ORG     0133H               ;P1 口中断入口地址
P1INT_ISR:
            PUSH    ACC
            PUSH    B
            PUSH    DPL
            PUSH    DPH
            PUSH    P_SW2

            MOV     DPTR,#P1INTF
            MOVX    A,@DPTR
            MOV     B,A
            CLR     A
            MOVX    @DPTR,A
            MOV     A,B
CHECKP10:
            JNB     ACC.0,CHECKP11
            NOP                         ;P1.0 口中断
CHECKP11:
            JNB     ACC.1,CHECKP12
            NOP                         ;P1.1 口中断
CHECKP12:
            JNB     ACC.2,CHECKP13
            NOP                         ;P1.2 口中断
CHECKP13
            JNB     ACC.3,CHECKP14
            NOP                         ;P1.3 口中断
CHECKP14:
            JNB     ACC.4,CHECKP15
            NOP                         ;P1.4 口中断
CHECKP15:
```

```
            JNB     ACC.5,CHECKP16
            NOP                             ;P1.5 口中断
CHECKP16:
            JNB     ACC.6,CHECKP17
            NOP                             ;P1.6 口中断
CHECKP17:
            JNB     ACC.7,P1ISREXIT
            NOP                             ;P1.7 口中断

P1ISREXIT:
            POP     P_SW2
            POP     DPH
            POP     DPL
            POP     B
            POP     ACC
            RETI

            ORG     0200H
MAIN:
            MOV     SP, #5FH

            MOV     P0M0,#00H
            MOV     P0M1,#00H
            MOV     P1M0,#00H
            MOV     P1M1,#00H
            MOV     P2M0,#00H
            MOV     P2M1,#00H
            MOV     P3M0,#00H
            MOV     P3M1,#00H

            ORL     P_SW2,#80H
            CLR     A
            MOV     DPTR,# P1IM0            ; 下降沿中断
            MOVX    @DPTR,A
            MOV     DPTR,# P1IM1
            MOVX    @DPTR,A
            MOV     DPTR,# P1INTE
            MOV     A,#0FFH
            MOVX    @DPTR,A                 ;使能 P1 口中断
            ANL     P_SW2,#7FH

            SETB    EA

            JMP     $

            END
```

12.2.3 P2 口低电平中断

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"
```

```
sfr     P0M0    =   0x94;
sfr     P0M1    =   0x93;
sfr     P1M0    =   0x92;
sfr     P1M1    =   0x91;
sfr     P2M0    =   0x96;
sfr     P2M1    =   0x95;
sfr     P3M0    =   0xb2;
sfr     P3M1    =   0xb1;
sfr     P4M0    =   0xb4;
sfr     P4M1    =   0xb3;
sfr     P5M0    =   0xca;
sfr     P5M1    =   0xc9;
sfr     P6M0    =   0xcc;
sfr     P6M1    =   0xcb;
sfr     P7M0    =   0xe2;
sfr     P7M1    =   0xe1;

sfr     P_SW2   =   0xba;

#define P2INTE  (*(unsigned char volatile xdata *)0xfd02)
#define P2INTF  (*(unsigned char volatile xdata *)0xfd12)
#define P2IM0   (*(unsigned char volatile xdata *)0xfd22)
#define P2IM1   (*(unsigned char volatile xdata *)0xfd32)

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 |= 0x80;
    P2IM0 = 0x00;                   //低电平中断
    P2IM1 = 0xff;
    P2INTE = 0xff;                  //使能 P2 口中断
    P_SW2 &= ~0x80;

    EA = 1;

    while (1);
}

//由于中断向量大于 31，在 KEIL 中无法直接编译
//必须借用第 13 号中断入口地址
void common_isr() interrupt 13
{
    unsigned char psw2_st;
    unsigned char intf;

    psw2_st = P_SW2;
    P_SW2 |= 0x80;
```

```
    intf = P2INTF;
    if (intf)
    {
        P2INTF = 0x00;
        if (intf & 0x01)
        {
                                                    //P2.0 口中断
        }
        if (intf & 0x02)
        {
                                                    //P2.1 口中断
        }
        if (intf & 0x04)
        {
                                                    //P2.2 口中断
        }
        if (intf & 0x08)
        {
                                                    //P0.3 口中断
        }
        if (intf & 0x10)
        {
                                                    //P2.4 口中断
        }
        if (intf & 0x20)
        {
                                                    //P2.5 口中断
        }
        if (intf & 0x40)
        {
                                                    //P2.6 口中断
        }
        if (intf & 0x80)
        {
                                                    //P2.7 口中断
        }
    }
    P_SW2 = psw2_st;
}

// ISR.ASM
//将下面的代码保存为ISR.ASM，然后将文件加入到项目中即可

            CSEG    AT 013BH            ;P2 口中断入口地址
            JMP     P2INT_ISR
P2INT_ISR:
            JMP     006BH               ;借用 13 号中断的入口地址
            END
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M0        DATA    094H
P0M1        DATA    093H
P1M0        DATA    092H
P1M1        DATA    091H
```

```
P2M0        DATA    096H
P2M1        DATA    095H
P3M0        DATA    0B2H
P3M1        DATA    0B1H
P4M0        DATA    0B4H
P4M1        DATA    0B3H
P5M0        DATA    0CAH
P5M1        DATA    0C9H
P6M0        DATA    0CCH
P6M1        DATA    0CBH
P7M0        DATA    0E2H
P7M1        DATA    0E1H

P_SW2       DATA    0BAH

P2INTE      XDATA   0FD02H
P2INTF      XDATA   0FD12H
P2IM0       XDATA   0FD22H
P2IM1       XDATA   0FD32H

            ORG     0000H
            LJMP    MAIN

            ORG     013BH               ;P2 口中断入口地址
P2INT_ISR:
            PUSH    ACC
            PUSH    B
            PUSH    DPL
            PUSH    DPH
            PUSH    P_SW2

            MOV     DPTR,#P2INTF
            MOVX    A,@DPTR
            MOV     B,A
            CLR     A
            MOVX    @DPTR,A
            MOV     A,B
CHECKP20:
            JNB     ACC.0,CHECKP21
            NOP                         ;P2.0 口中断
CHECKP21:
            JNB     ACC.1,CHECKP22
            NOP                         ;P2.1 口中断
CHECKP22:
            JNB     ACC.2,CHECKP23
            NOP                         ;P2.2 口中断
CHECKP23
            JNB     ACC.3,CHECKP24
            NOP                         ;P2.3 口中断
CHECKP24:
            JNB     ACC.4,CHECKP25
            NOP                         ;P2.4 口中断
CHECKP25:
            JNB     ACC.5,CHECKP26
            NOP                         ;P2.5 口中断
CHECKP26:
            JNB     ACC.6,CHECKP27
            NOP                         ;P2.6 口中断
CHECKP27:
```

```
            JNB     ACC.7,P2ISREXIT
            NOP                             ;P2.7 口中断

P2ISREXIT:
            POP     P_SW2
            POP     DPH
            POP     DPL
            POP     B
            POP     ACC
            RETI

            ORG     0200H
MAIN:
            MOV     SP, #5FH

            MOV     P0M0,#00H
            MOV     P0M1,#00H
            MOV     P1M0,#00H
            MOV     P1M1,#00H
            MOV     P2M0,#00H
            MOV     P2M1,#00H
            MOV     P3M0,#00H
            MOV     P3M1,#00H

            ORL     P_SW2,#80H
            CLR     A
            MOV     DPTR,# P2IM0            ;低电平中断
            MOVX    @DPTR,A
            MOV     DPTR,# P2IM1
            MOVX    @DPTR,A
            MOV     DPTR,# P2INTE
            MOV     A,#0FFH
            MOVX    @DPTR,A                 ;使能 P2 口中断
            ANL     P_SW2,#7FH

            SETB    EA

            JMP     $

            END
```

12.2.4 P3 口高电平中断

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M0    =   0x94;
sfr     P0M1    =   0x93;
sfr     P1M0    =   0x92;
sfr     P1M1    =   0x91;
sfr     P2M0    =   0x96;
sfr     P2M1    =   0x95;
```

```
sfr     P3M0    =   0xb2;
sfr     P3M1    =   0xb1;
sfr     P4M0    =   0xb4;
sfr     P4M1    =   0xb3;
sfr     P5M0    =   0xca;
sfr     P5M1    =   0xc9;
sfr     P6M0    =   0xcc;
sfr     P6M1    =   0xcb;
sfr     P7M0    =   0xe2;
sfr     P7M1    =   0xe1;

sfr     P_SW2   =   0xba;

#define P3INTE  (*(unsigned char volatile xdata *)0xfd03)
#define P3INTF  (*(unsigned char volatile xdata *)0xfd13)
#define P3IM0   (*(unsigned char volatile xdata *)0xfd23)
#define P3IM1   (*(unsigned char volatile xdata *)0xfd33)

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 |= 0x80;
    P3IM0 = 0xff;                   //高电平中断
    P3IM1 = 0xff;
    P3INTE = 0xff;                  //使能 P3 口中断
    P_SW2 &= ~0x80;

    EA = 1;

    while (1);
}

//由于中断向量大于 31，在 KEIL 中无法直接编译
//必须借用第 13 号中断入口地址
void common_isr() interrupt 13
{
    unsigned char psw2_st;
    unsigned char intf;

    psw2_st = P_SW2;
    P_SW2 |= 0x80;
    intf = P3INTF;
    if (intf)
    {
        P3INTF = 0x00;
        if (intf & 0x01)
        {
```

STC MCU

```
                                                    //P3.0 口中断
        }
        if (intf & 0x02)
        {
                                                    //P3.1 口中断
        }
        if (intf & 0x04)
        {
                                                    //P3.2 口中断
        }
        if (intf & 0x08)
        {
                                                    //P3.3 口中断
        }
        if (intf & 0x10)
        {
                                                    //P3.4 口中断
        }
        if (intf & 0x20)
        {
                                                    //P3.5 口中断
        }
        if (intf & 0x40)
        {
                                                    //P3.6 口中断
        }
        if (intf & 0x80)
        {
                                                    //P3.7 口中断
        }
    }
    P_SW2 = psw2_st;
}

// ISR.ASM
//将下面的代码保存为ISR.ASM，然后将文件加入到项目中即可

            CSEG    AT 0143H            ;P3 口中断入口地址
            JMP     P3INT_ISR
P3INT_ISR:
            JMP     006BH               ;借用 13 号中断的入口地址
            END
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M0        DATA    094H
P0M1        DATA    093H
P1M0        DATA    092H
P1M1        DATA    091H
P2M0        DATA    096H
P2M1        DATA    095H
P3M0        DATA    0B2H
P3M1        DATA    0B1H
P4M0        DATA    0B4H
P4M1        DATA    0B3H
```

```
P5M0        DATA    0CAH
P5M1        DATA    0C9H
P6M0        DATA    0CCH
P6M1        DATA    0CBH
P7M0        DATA    0E2H
P7M1        DATA    0E1H

P_SW2       DATA    0BAH

P3INTE      XDATA   0FD03H
P3INTF      XDATA   0FD13H
P3IM0       XDATA   0FD23H
P3IM1       XDATA   0FD33H

            ORG     0000H
            LJMP    MAIN

            ORG     0143H               ;P3 口中断入口地址
P3INT_ISR:
            PUSH    ACC
            PUSH    B
            PUSH    DPL
            PUSH    DPH
            PUSH    P_SW2

            MOV     DPTR,#P3INTF
            MOVX    A,@DPTR
            MOV     B,A
            CLR     A
            MOVX    @DPTR,A
            MOV     A,B
CHECKP30:
            JNB     ACC.0,CHECKP31
            NOP                         ;P3.0 口中断
CHECKP31:
            JNB     ACC.1,CHECKP32
            NOP                         ;P3.1 口中断
CHECKP32:
            JNB     ACC.2,CHECKP33
            NOP                         ;P3.2 口中断
CHECKP33
            JNB     ACC.3,CHECKP34
            NOP                         ;P3.3 口中断
CHECKP34:
            JNB     ACC.4,CHECKP35
            NOP                         ;P3.4 口中断
CHECKP35:
            JNB     ACC.5,CHECKP36
            NOP                         ;P3.5 口中断
CHECKP36:
            JNB     ACC.6,CHECKP37
            NOP                         ;P3.6 口中断
CHECKP37:
            JNB     ACC.7,P3ISREXIT
            NOP                         ;P3.7 口中断

P3ISREXIT:
            POP     P_SW2
            POP     DPH
```

```
        POP     DPL
        POP     B
        POP     ACC
        RETI

        ORG     0200H
MAIN:
        MOV     SP, #5FH

        MOV     P0M0,#00H
        MOV     P0M1,#00H
        MOV     P1M0,#00H
        MOV     P1M1,#00H
        MOV     P2M0,#00H
        MOV     P2M1,#00H
        MOV     P3M0,#00H
        MOV     P3M1,#00H

        ORL     P_SW2,#80H
        CLR     A
        MOV     DPTR,# P3IM0            ;高电平中断
        MOVX    @DPTR,A
        MOV     DPTR,# P3IM1
        MOVX    @DPTR,A
        MOV     DPTR,# P3INTE
        MOV     A,#0FFH
        MOVX    @DPTR,A                 ;使能 P3 口中断
        ANL     P_SW2,#7FH

        SETB    EA

        JMP     $

        END
```

STC MCU

13 定时器/计数器

STC8C 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。

STCMCU

13.1 定时器的相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
TCON	定时器控制寄存器	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	0000,0000
TMOD	定时器模式寄存器	89H	GATE	C/T	M1	M0	GATE	C/T	M1	M0	0000,0000
TL0	定时器 0 低 8 位寄存器	8AH									0000,0000
TL1	定时器 1 低 8 位寄存器	8BH									0000,0000
TH0	定时器 0 高 8 位寄存器	8CH									0000,0000
TH1	定时器 1 高 8 位寄存器	8DH									0000,0000
AUXR	辅助寄存器 1	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2	0000,0001
INTCLKO	中断与时钟输出控制寄存器	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO	x000,x000
WKTCL	掉电唤醒定时器低字节	AAH									1111,1111
WKTCH	掉电唤醒定时器高字节	ABH	WKTEN								0111,1111
T4T3M	定时器 4/3 控制寄存器	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO	0000,0000
T4H	定时器 4 高字节	D2H									0000,0000
T4L	定时器 4 低字节	D3H									0000,0000
T3H	定时器 3 高字节	D4H									0000,0000
T3L	定时器 3 低字节	D5H									0000,0000
T2H	定时器 2 高字节	D6H									0000,0000
T2L	定时器 2 低字节	D7H									0000,0000

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
TM2PS	定时器 2 时钟预分频寄存器	FEA2H									0000,0000
TM3PS	定时器 3 时钟预分频寄存器	FEA3H									0000,0000
TM4PS	定时器 4 时钟预分频寄存器	FEA4H									0000,0000

13.2 定时器 0/1

13.2.1 定时器 0/1 控制寄存器（TCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TCON	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1：T1溢出中断标志。T1被允许计数以后，从初值开始加1计数。当产生溢出时由硬件将TF1位置“1”，并向CPU请求中断，一直保持到CPU响应中断时，才由硬件清“0”（也可由查询软件清“0”）。

TR1：定时器T1的运行控制位。该位由软件置位和清零。当GATE（TMOD.7）=0，TR1=1时就允许T1开始计数，TR1=0时禁止T1计数。当GATE（TMOD.7）=1，TR1=1且INT1输入高电平时，才允许T1计数。

TF0：T0溢出中断标志。T0被允许计数以后，从初值开始加1计数，当产生溢出时，由硬件置“1”TF0，向CPU请求中断，一直保持CPU响应该中断时，才由硬件清0（也可由查询软件清0）。

TR0：定时器T0的运行控制位。该位由软件置位和清零。当GATE（TMOD.3）=0，TR0=1时就允许T0开始计数，TR0=0时禁止T0计数。当GATE（TMOD.3）=1，TR0=1且INT0输入高电平时，才允许T0计数，TR0=0时禁止T0计数。

IE1：外部中断1请求源（INT1/P3.3）标志。IE1=1，外部中断向CPU请求中断，当CPU响应该中断时由硬件清“0”IE1。

IT1：外部中断源1触发控制位。IT1=0，上升沿或下降沿均可触发外部中断1。IT1=1，外部中断1程控为下降沿触发方式。

IE0：外部中断0请求源（INT0/P3.2）标志。IE0=1外部中断0向CPU请求中断，当CPU响应外部中断时，由硬件清“0”IE0（边沿触发方式）。

IT0：外部中断源0触发控制位。IT0=0，上升沿或下降沿均可触发外部中断0。IT0=1，外部中断0程控为下降沿触发方式。

STC MCU

13.2.2 定时器 0/1 模式寄存器（TMOD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TMOD	89H	T1_GATE	T1_C/T	T1_M1	T1_M0	T0_GATE	T0_C/T	T0_M1	T0_M0

T1_GATE：控制定时器1，置1时只有在INT1脚为高及TR1控制位置1时才可打开定时器/计数器1。

T0_GATE：控制定时器0，置1时只有在INT0脚为高及TR0控制位置1时才可打开定时器/计数器0。

T1_C/T：控制定时器1用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T1/P3.5外部脉冲进行计数）。

T0_C/T：控制定时器0用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T0/P3.4外部脉冲进行计数）。

T1_M1/T1_M0：定时器定时器/计数器1模式选择

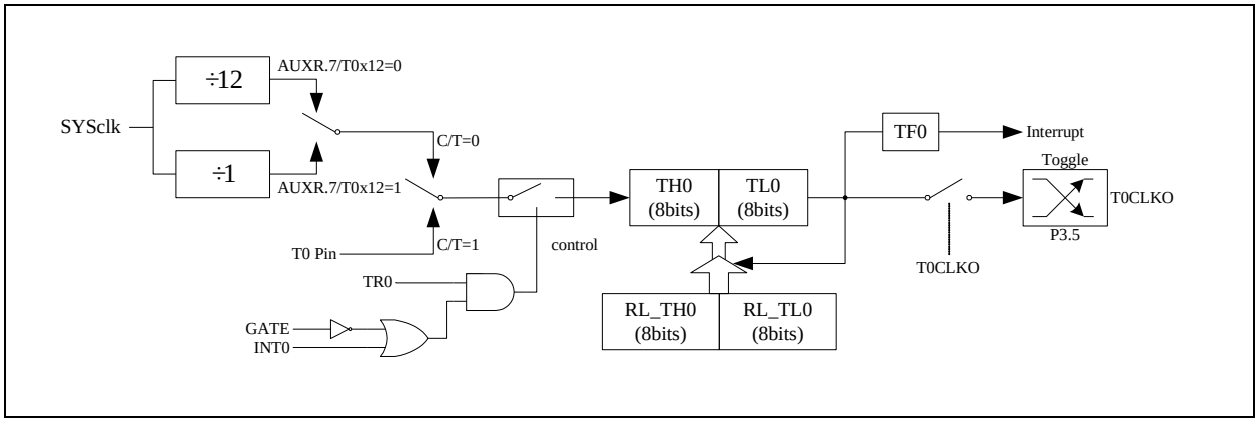
T1_M1	T1_M0	定时器/计数器1工作模式
0	0	16位自动重载模式 当[TH1,TL1]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[TH1,TL1]中。
0	1	16位不自动重载模式 当[TH1,TL1]中的16位计数值溢出时，定时器1将从0开始计数
1	0	8位自动重载模式 当TL1中的8位计数值溢出时，系统会自动将TH1中的重载值装入TL1中。
1	1	T1停止工作

T0_M1/T0_M0：定时器定时器/计数器0模式选择

T0_M1	T0_M0	定时器/计数器0工作模式
0	0	16位自动重载模式 当[TH0,TL0]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[TH0,TL0]中。
0	1	16位不自动重载模式 当[TH0,TL0]中的16位计数值溢出时，定时器0将从0开始计数
1	0	8位自动重载模式 当TL0中的8位计数值溢出时，系统会自动将TH0中的重载值装入TL0中。
1	1	不可屏蔽中断的16位自动重载模式 与模式0相同，不可屏蔽中断，中断优先级最高，高于其他所有中断的优先级，并且不可关闭，可用作操作系统的系统节拍定时器，或者系统监控定时器。

13.2.3 定时器 0 模式 0（16 位自动重装载模式）

此模式下定时器/计数器 0 作为可自动重装载的 16 位计数器，如下图所示：



定时器/计数器 0 的模式 0：16 位自动重装载模式

当 GATE=0（TMOD.3）时，如 TR0=1，则定时器计数。GATE=1 时，允许由外部输入 INT0 控制定时器 0，这样可实现脉宽测量。TR0 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T0 对内部系统时钟计数，T0 工作在定时方式。当 C/T=1 时，多路开关连接到外部脉冲输入 P3.4/T0，即 T0 工作在计数方式。

STC 单片机的定时器 0 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T0 的速率由特殊功能寄存器 AUXR 中的 T0x12 决定，如果 T0x12=0，T0 则工作在 12T 模式；如果 T0x12=1，T0 则工作在 1T 模式

定时器 0 有两个隐藏的寄存器 RL_TH0 和 RL_TL0。RL_TH0 与 TH0 共有同一个地址，RL_TL0 与 TL0 共有同一个地址。当 TR0=0 即定时器/计数器 0 被禁止工作时，对 TL0 写入的内容会同时写入 RL_TL0，对 TH0 写入的内容也会同时写入 RL_TH0。当 TR0=1 即定时器/计数器 0 被允许工作时，对 TL0 写入内容，实际上不是写入当前寄存器 TL0 中，而是写入隐藏的寄存器 RL_TL0 中，对 TH0 写入内容，实际上也不是写入当前寄存器 TH0 中，而是写入隐藏的寄存器 RL_TH0，这样可以巧妙地实现 16 位重装载定时器。当读 TH0 和 TL0 的内容时，所读的内容就是 TH0 和 TL0 的内容，而不是 RL_TH0 和 RL_TL0 的内容。

当定时器 0 工作在模式 0（TMOD[1:0]/[M1,M0]=00B）时，[TH0,TL0]的溢出不仅置位 TF0，而且会自动将[RL_TH0,RL_TL0]的内容重新装入[TH0,TL0]。

当 T0CLKO/INTCLKO.0=1 时，P3.5/T1 管脚配置为定时器 0 的时钟输出 T0CLKO。输出时钟频率为 T0 溢出率/2。

如果 C/T=0，定时器/计数器 T0 对内部系统时钟计数，则：

T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出时钟频率 = (SYSclk)/(65536-[RL_TH0, RL_TL0])/2

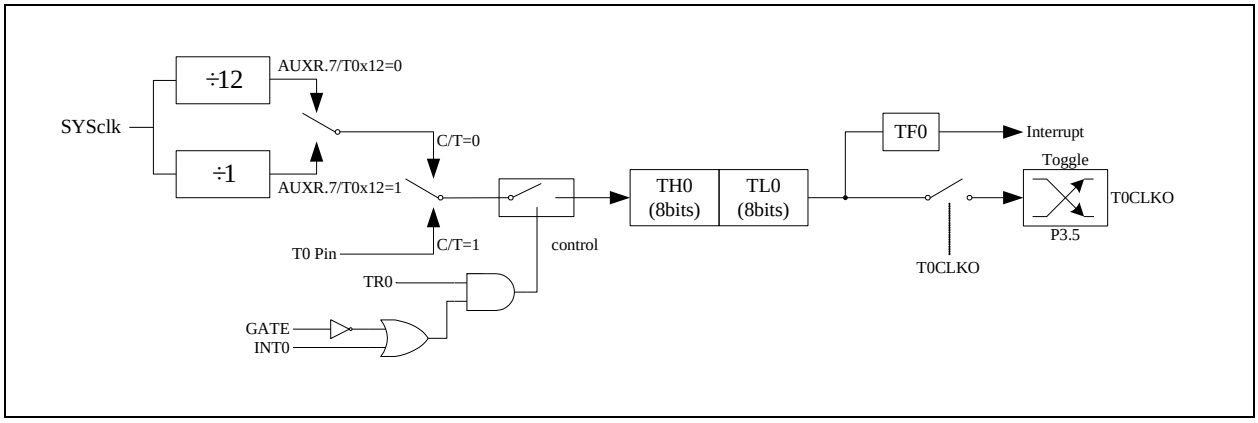
T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出时钟频率 = (SYSclk)/12/(65536-[RL_TH0, RL_TL0])/2

如果 C/T=1，定时器/计数器 T0 是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (65536-[RL_TH0, RL_TL0])/2

13.2.4　定时器 0 模式 1（16 位不可重装载模式）

此模式下定时器/计数器 0 工作在 16 位不可重装载模式，如下图所示



定时器/计数器 0 的模式 1：16 位不可重装载模式

此模式下，定时器/计数器 0 配置为 16 位不可重装载模式，由 TL0 的 8 位和 TH0 的 8 位所构成。TL0 的 8 位溢出向 TH0 进位，TH0 计数溢出置位 TCON 中的溢出标志位 TF0。

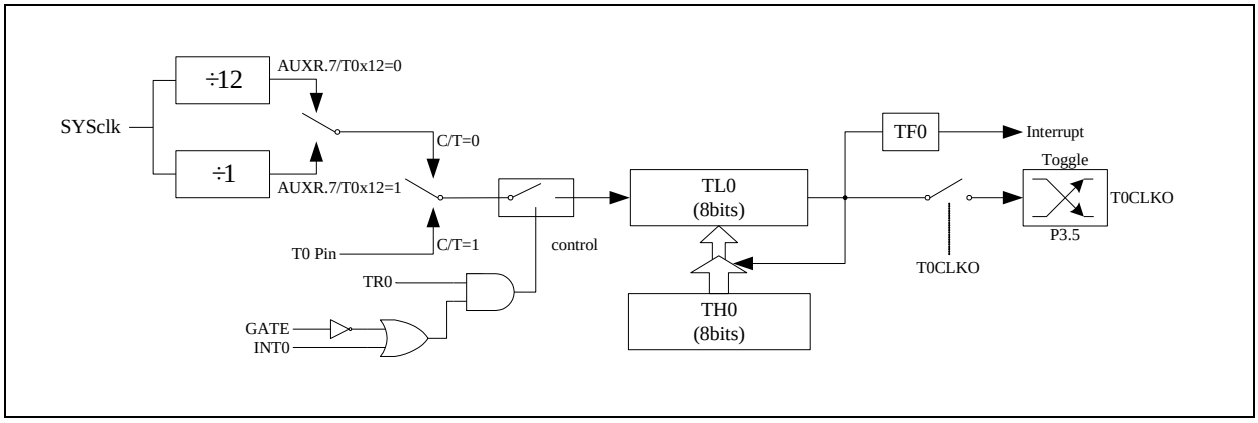
当 GATE=0(TMOD.3)时，如 TR0=1，则定时器计数。GATE=1 时，允许由外部输入 INT0 控制定时器 0，这样可实现脉宽测量。TR0 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T0 对内部系统时钟计数，T0 工作在定时方式。当 C/T=1 时，多路开关连接到外部脉冲输入 P3.4/T0，即 T0 工作在计数方式。

STC 单片机的定时器 0 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T0 的速率由特殊功能寄存器 AUXR 中的 T0x12 决定，如果 T0x12=0，T0 则工作在 12T 模式；如果 T0x12=1，T0 则工作在 1T 模式

13.2.5 定时器 0 模式 2（8 位自动重装载模式）

此模式下定时器/计数器 0 作为可自动重装载的 8 位计数器，如下图所示：



定时器/计数器 0 的模式 2：8 位自动重装载模式

TL0 的溢出不仅置位 TF0，而且将 TH0 的内容重新装入 TL0，TH0 内容由软件预置，重装时 TH0 内容不变。

当 T0CLKO/INTCLKO.0=1 时，P3.5/T1 管脚配置为定时器 0 的时钟输出 T0CLKO。输出时钟频率为 T0 溢出率/2。

如果 C/T=0，定时器/计数器 T0 对内部系统时钟计数，则：

T0 工作在 1T 模式（AUXR.7/T0x12=1）时的输出时钟频率 = (SYSclk)/(256-TH0)/2

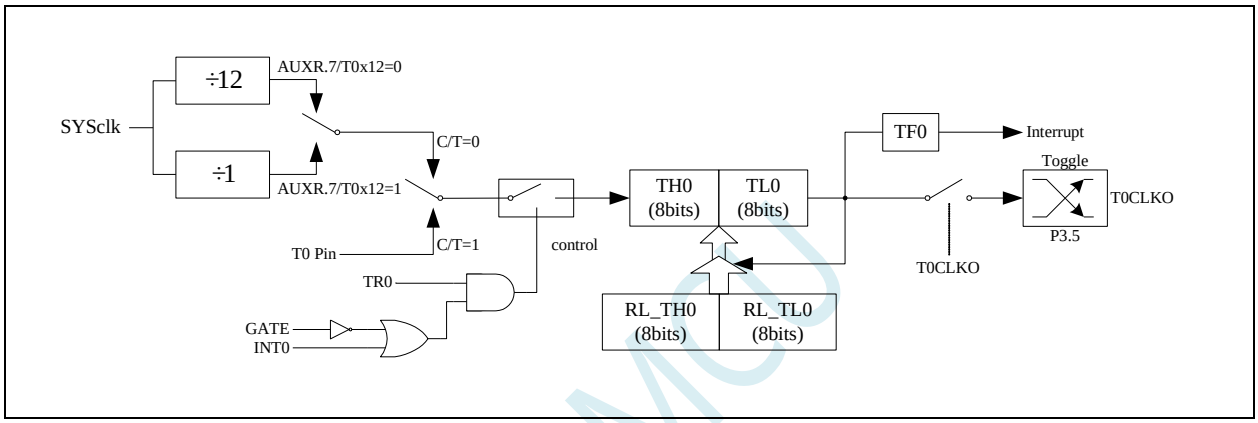
T0 工作在 12T 模式（AUXR.7/T0x12=0）时的输出时钟频率 = (SYSclk)/12/(256-TH0)/2

如果 C/T=1，定时器/计数器 T0 是对外部脉冲输入(P3.4/T0)计数，则：

输出时钟频率 = (T0_Pin_CLK) / (256-TH0)/2

13.2.6 定时器 0 模式 3（不可屏蔽中断 16 位自动重装载，实时操作系统节拍器）

对定时器/计数器 0，其工作模式模式 3 与工作模式 0 是一样的（下图定时器模式 3 的原理图，与工作模式 0 是一样的）。唯一不同的是：当定时器/计数器 0 工作在模式 3 时，只需允许 ET0/IE.1(定时器/计数器 0 中断允许位)，不需要允许 EA/IE.7(总中断使能位)就能打开定时器/计数器 0 的中断，此模式下的定时器/计数器 0 中断与总中断使能位 EA 无关，一旦工作在模式 3 下的定时器/计数器 0 中断被打开(ET0=1)，那么该中断是不可屏蔽的，该中断的优先级是最高的，即该中断不能被任何中断所打断，而且该中断打开后既不受 EA/IE.7 控制也不再受 ET0 控制，当 EA=0 或 ET0=0 时都不能屏蔽此中断。故将此模式称为不可屏蔽中断的 16 位自动重装载模式。

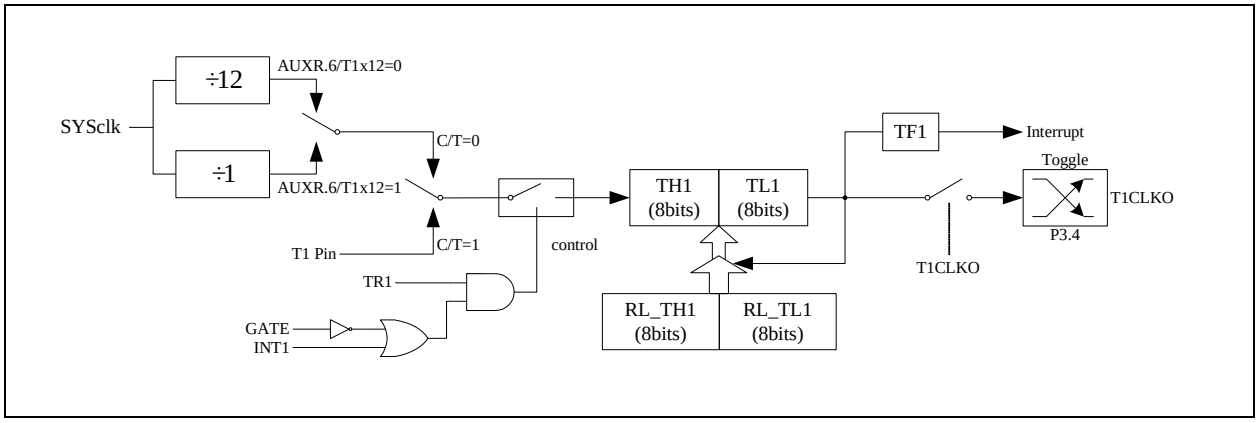


定时器/计数器 0 的模式 3：不可屏蔽中断的 16 位自动重装载模式

注意：当定时器/计数器 0 工作在模式 3(不可屏蔽中断的 16 位自动重装载模式)时，不需要允许 EA/IE.7(总中断使能位)，只需允许 ET0/IE.1(定时器/计数器 0 中断允许位)就能打开定时器/计数器 0 的中断，此模式下的定时器/计数器 0 中断与总中断使能位 EA 无关。一旦此模式下的定时器/计数器 0 中断被打开后，该定时器/计数器 0 中断优先级就是最高的，它不能被其它任何中断所打断(不管是比定时器/计数器 0 中断优先级低的中断还是比其优先级高的中断，都不能打断此时的定时器/计数器 0 中断)，而且该中断打开后既不受 EA/IE.7 控制也不再受 ET0 控制了，清零 EA 或 ET0 都不能关闭此中断。

13.2.7 定时器 1 模式 0（16 位自动重装载模式）

此模式下定时器/计数器 1 作为可自动重装载的 16 位计数器，如下图所示：



定时器/计数器 1 的模式 0：16 位自动重装载模式

当 GATE=0（TMOD.7）时，如 TR1=1，则定时器计数。GATE=1 时，允许由外部输入 INT1 控制定时器 1，这样可实现脉宽测量。TR1 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T1 对内部系统时钟计数，T1 工作在定时方式。当 C/T=1 时，多路开关连接到外部脉冲输入 P3.5/T1，即 T1 工作在计数方式。

STC 单片机的定时器 1 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T1 的速率由特殊功能寄存器 AUXR 中的 T1x12 决定，如果 T1x12=0，T1 则工作在 12T 模式；如果 T1x12=1，T1 则工作在 1T 模式

定时器 1 有两个隐藏的寄存器 RL_TH1 和 RL_TL1。RL_TH1 与 TH1 共有同一个地址，RL_TL1 与 TL1 共有同一个地址。当 TR1=0 即定时器/计数器 1 被禁止工作时，对 TL1 写入的内容会同时写入 RL_TL1，对 TH1 写入的内容也会同时写入 RL_TH1。当 TR1=1 即定时器/计数器 1 被允许工作时，对 TL1 写入内容，实际上不是写入当前寄存器 TL1 中，而是写入隐藏的寄存器 RL_TL1 中，对 TH1 写入内容，实际上也不是写入当前寄存器 TH1 中，而是写入隐藏的寄存器 RL_TH1，这样可以巧妙地实现 16 位重装载定时器。当读 TH1 和 TL1 的内容时，所读的内容就是 TH1 和 TL1 的内容，而不是 RL_TH1 和 RL_TL1 的内容。

当定时器 1 工作在模式 1（TMOD[5:4]/[M1,M0]=00B）时，[TH1,TL1]的溢出不仅置位 TF1，而且会自动将[RL_TH1,RL_TL1]的内容重新装入[TH1,TL1]。

当 T1CLKO/INTCLKO.1=1 时，P3.4/T0 管脚配置为定时器 1 的时钟输出 T1CLKO。输出时钟频率为 T1 溢出率/2。

如果 C/T=0，定时器/计数器 T1 对内部系统时钟计数，则：

T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出时钟频率 = (SYSclk)/(65536-[RL_TH1, RL_TL1])/2

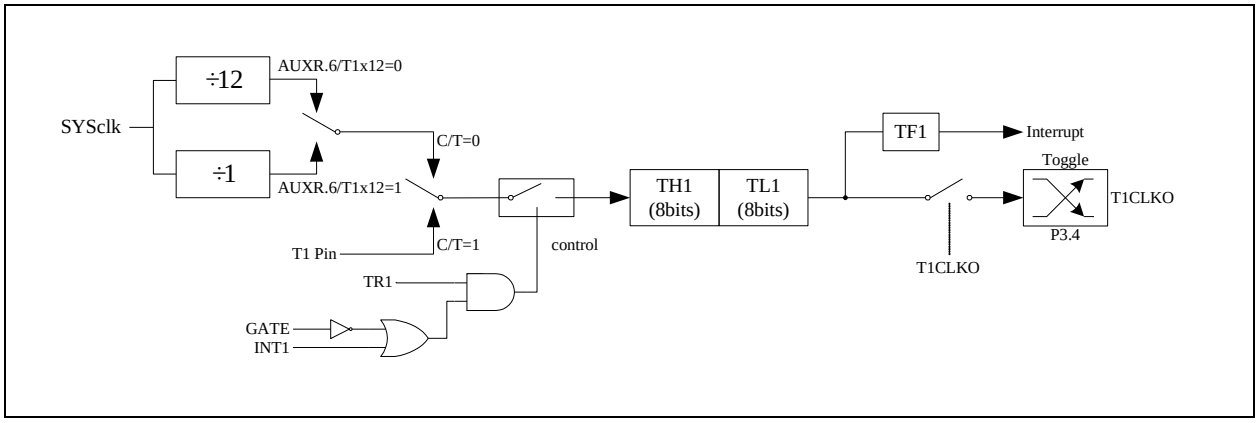
T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出时钟频率 = (SYSclk)/12/(65536-[RL_TH1, RL_TL1])/2

如果 C/T=1，定时器/计数器 T1 是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (65536-[RL_TH1, RL_TL1])/2

13.2.8　定时器 1 模式 1（16 位不可重装载模式）

此模式下定时器/计数器 1 工作在 16 位不可重装载模式，如下图所示



定时器/计数器 1 的模式 1：16 位不可重装载模式

此模式下，定时器/计数器 1 配置为 16 位不可重装载模式，由 TL1 的 8 位和 TH1 的 8 位所构成。TL1 的 8 位溢出向 TH1 进位，TH1 计数溢出置位 TCON 中的溢出标志位 TF1。

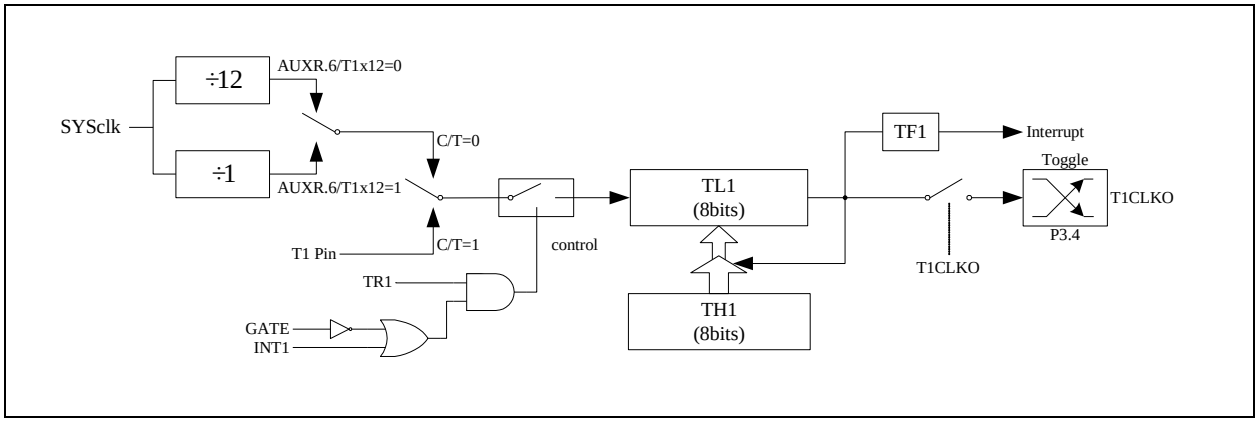
当 GATE=0(TMOD.7)时，如 TR1=1，则定时器计数。GATE=1 时，允许由外部输入 INT1 控制定时器 1，这样可实现脉宽测量。TR1 为 TCON 寄存器内的控制位，TCON 寄存器各位的具体功能描述见上节 TCON 寄存器的介绍。

当 C/T=0 时，多路开关连接到系统时钟的分频输出，T1 对内部系统时钟计数，T1 工作在定时方式。当 C/T=1 时，多路开关连接到外部脉冲输入 P3.5/T1，即 T1 工作在计数方式。

STC 单片机的定时器 1 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T1 的速率由特殊功能寄存器 AUXR 中的 T1x12 决定，如果 T1x12=0，T1 则工作在 12T 模式；如果 T1x12=1，T1 则工作在 1T 模式

13.2.9 定时器 1 模式 2（8 位自动重装载模式）

此模式下定时器/计数器 1 作为可自动重装载的 8 位计数器，如下图所示：



定时器/计数器 1 的模式 2：8 位自动重装载模式

TL1 的溢出不仅置位 TF1，而且将 TH1 的内容重新装入 TL1，TH1 内容由软件预置，重装时 TH1 内容不变。

当 T1CLKO/INTCLKO.1=1 时，P3.4/T0 管脚配置为定时器 1 的时钟输出 T1CLKO。输出时钟频率为 T1 溢出率/2。

如果 C/T=0，定时器/计数器 T1 对内部系统时钟计数，则：

T1 工作在 1T 模式（AUXR.6/T1x12=1）时的输出时钟频率 = (SYSclk)/(256-TH1)/2

T1 工作在 12T 模式（AUXR.6/T1x12=0）时的输出时钟频率 = (SYSclk)/12/(256-TH1)/2

如果 C/T=1，定时器/计数器 T1 是对外部脉冲输入(P3.5/T1)计数，则：

输出时钟频率 = (T1_Pin_CLK) / (256-TH1)/2

13.2.10 定时器 0 计数寄存器（TL0，TH0）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL0	8AH								
TH0	8CH								

当定时器/计数器0工作在16位模式（模式0、模式1、模式3）时，TL0和TH0组合成为一个16位寄存器，TL0为低字节，TH0为高字节。若为8位模式（模式2）时，TL0和TH0为两个独立的8位寄存器。

13.2.11 定时器 1 计数寄存器（TL1，TH1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TL1	8BH								
TH1	8DH								

当定时器/计数器1工作在16位模式（模式0、模式1）时，TL1和TH1组合成为一个16位寄存器，TL1为低字节，TH1为高字节。若为8位模式（模式2）时，TL1和TH1为两个独立的8位寄存器。

13.2.12 辅助寄存器 1（AUXR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T0x12：定时器0速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

T1x12：定时器1速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

13.2.13 中断与时钟输出控制寄存器（INTCLKO）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

T0CLKO：定时器0时钟输出控制

0：关闭时钟输出

1：使能 P3.5 口的是定时器 0 时钟输出功能

当定时器 0 计数发生溢出时，P3.5 口的电平自动发生翻转。

T1CLKO：定时器1时钟输出控制

0：关闭时钟输出

1：使能 P3.4 口的是定时器 1 时钟输出功能

当定时器 1 计数发生溢出时，P3.4 口的电平自动发生翻转。

STC MCU

13.2.14 定时器 0 定时计算公式

定时器模式	定时器速度	周期计算公式
模式0/3 （16位自动重载）	1T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk}$ （自动重载）
	12T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk} \times 12$ （自动重载）
模式1 （16位不自动重载）	1T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk}$ （需软件装载）
	12T	定时器周期 = $\frac{65536 - [TH0, TL0]}{SYSclk} \times 12$ （需软件装载）
模式2 （8位自动重载）	1T	定时器周期 = $\frac{256 - TH0}{SYSclk}$ （自动重载）
	12T	定时器周期 = $\frac{256 - TH0}{SYSclk} \times 12$ （自动重载）

STC MCU

13.2.15 定时器 1 定时计算公式

定时器模式	定时器速度	周期计算公式
模式0 （16位自动重载）	1T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk}$ （自动重载）
	12T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk} \times 12$ （自动重载）
模式1 （16位不自动重载）	1T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk}$ （需软件装载）
	12T	定时器周期 = $\frac{65536 - [TH1, TL1]}{SYSclk} \times 12$ （需软件装载）
模式2 （8位自动重载）	1T	定时器周期 = $\frac{256 - TH1}{SYSclk}$ （自动重载）
	12T	定时器周期 = $\frac{256 - TH1}{SYSclk} \times 12$ （自动重载）

STC MCU

13.3 定时器 2（24 位定时器，8 位预分频+16 位定时）

13.3.1 辅助寄存器 1（AUXR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

T2R：定时器2的运行控制位

0：定时器 2 停止计数

1：定时器 2 开始计数

T2_C/T：控制定时器2用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T2/P1.2外部脉冲进行计数）。

T2x12：定时器2速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

13.3.2 中断与时钟输出控制寄存器（INTCLKO）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
INTCLKO	8FH	-	EX4	EX3	EX2	-	T2CLKO	T1CLKO	T0CLKO

T2CLKO：定时器2时钟输出控制

0：关闭时钟输出

1：使能 P1.3 口的是定时器 2 时钟输出功能

当定时器 2 计数发生溢出时，P1.3 口的电平自动发生翻转。

13.3.3 定时器 2 计数寄存器（T2L，T2H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T2L	D7H								
T2H	D6H								

定时器/计数器2的工作模式固定为16位重载模式，T2L和T2H组合成为一个16位寄存器，T2L为低字节，T2H为高字节。当[T2H,T2L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T2H,T2L]中。

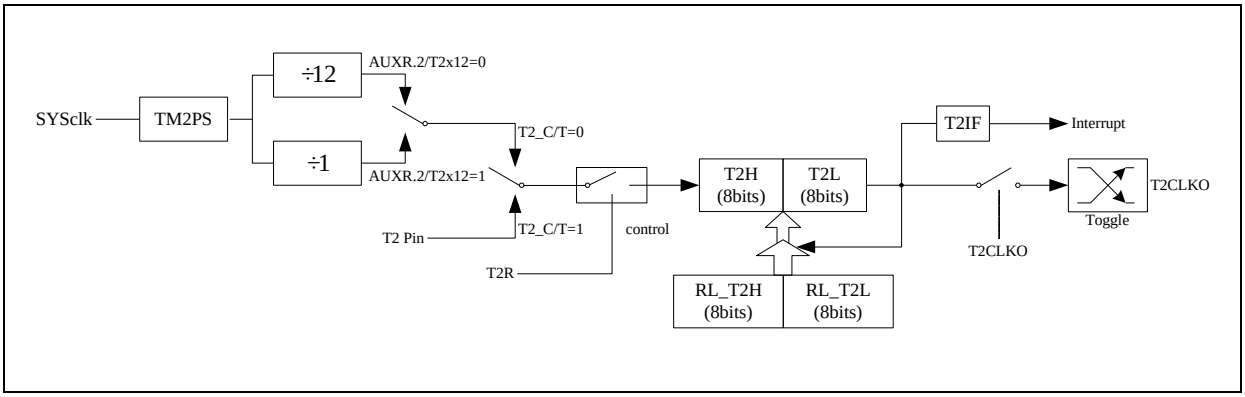
13.3.4 定时器 2 的 8 位预分频寄存器（TM2PS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TM2PS	FEA2H								

定时器2的时钟 ＝ 系统时钟SYSclk ÷ (TM2PS + 1)

13.3.5 定时器 2 工作模式

定时器/计数器 2 的原理框图如下：



定时器/计数器 2 的工作模式：16 位自动重装载模式

T2R/AUXR.4 为 AUXR 寄存器内的控制位，AUXR 寄存器各位的具体功能描述见上节 AUXR 寄存器的介绍。

当 T2_C/T=0 时，多路开关连接到系统时钟输出，T2 对内部系统时钟计数，T2 工作在定时方式。当 T2_C/T=1 时，多路开关连接到外部脉冲输 T2，即 T2 工作在计数方式。

STC 单片机的定时器 2 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T2 的速率由特殊功能寄存器 AUXR 中的 T2x12 决定，如果 T2x12=0，T2 则工作在 12T 模式；如果 T2x12=1，T2 则工作在 1T 模式

定时器 2 有两个隐藏的寄存器 RL_T2H 和 RL_T2L。RL_T2H 与 T2H 共有同一个地址，RL_T2L 与 T2L 共有同一个地址。当 T2R=0 即定时器/计数器 2 被禁止工作时，对 T2L 写入的内容会同时写入 RL_T2L，对 T2H 写入的内容也会同时写入 RL_T2H。当 T2R=1 即定时器/计数器 2 被允许工作时，对 T2L 写入内容，实际上不是写入当前寄存器 T2L 中，而是写入隐藏的寄存器 RL_T2L 中，对 T2H 写入内容，实际上也不是写入当前寄存器 T2H 中，而是写入隐藏的寄存器 RL_T2H，这样可以巧妙地实现 16 位重装载定时器。当读 T2H 和 T2L 的内容时，所读的内容就是 T2H 和 T2L 的内容，而不是 RL_T2H 和 RL_T2L 的内容。

[T2H,T2L]的溢出不仅置位中断请求标志位（T2IF），使 CPU 转去执行定时器 2 的中断程序，而且会自动将[RL_T2H,RL_T2L]的内容重新装入[T2H,T2L]。

13.3.6 定时器 2 计算公式

定时器速度	周期计算公式
1T	定时器周期 = $\frac{65536 - [T2H, T2L]}{SYSclk/(TM2PS+1)}$ （自动重载）
12T	定时器周期 = $\frac{65536 - [T2H, T2L]}{SYSclk/(TM2PS+1)} \times 12$ （自动重载）

13.4 定时器 3/4（24 位定时器，8 位预分频+16 位定时）

13.4.1 定时器 4/3 控制寄存器（T4T3M）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4T3M	D1H	T4R	T4_C/T	T4x12	T4CLKO	T3R	T3_C/T	T3x12	T3CLKO

T4R：定时器4的运行控制位

0：定时器 4 停止计数

1：定时器 4 开始计数

T4_C/T：控制定时器4用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T4/P0.6外部脉冲进行计数）。

T4x12：定时器4速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

T4CLKO：定时器4时钟输出控制

0：关闭时钟输出

1：使能 P0.7 口的是定时器 4 时钟输出功能

当定时器 4 计数发生溢出时，P0.7 口的电平自动发生翻转。

T3R：定时器3的运行控制位

0：定时器 3 停止计数

1：定时器 3 开始计数

T3_C/T：控制定时器3用作定时器或计数器，清0则用作定时器（对内部系统时钟进行计数），置1用作计数器（对引脚T3/P0.4外部脉冲进行计数）。

T3x12：定时器3速度控制位

0：12T 模式，即 CPU 时钟 12 分频（FOSC/12）

1：1T 模式，即 CPU 时钟不分频分频（FOSC/1）

T3CLKO：定时器3时钟输出控制

0：关闭时钟输出

1：使能 P0.5 口的是定时器 3 时钟输出功能

当定时器 3 计数发生溢出时，P0.5 口的电平自动发生翻转。

13.4.2 定时器 3 计数寄存器（T3L，T3H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T3L	D5H								
T3H	D4H								

定时器/计数器3的工作模式固定为16位重载模式，T3L和T3H组合成为一个16位寄存器，T3L为低字节，T3H为高字节。当[T3H,T3L]中的16位计数值溢出时，系统会自动将内部16位重载寄存器中的重载值装入[T3H,T3L]中。

13.4.3 定时器 4 计数寄存器（T4L，T4H）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
T4L	D3H								
T4H	D2H								

定时器/计数器 4 的工作模式固定为 16 位重载模式，T4L 和 T4H 组合成为一个 16 位寄存器，T4L 为低字节，T4H 为高字节。当[T4H,T4L]中的 16 位计数值溢出时，系统会自动将内部 16 位重载寄存器中的重载值装入[T4H,T4L]中。

13.4.4 定时器 3 的 8 位预分频寄存器（TM3PS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TM3PS	FEA3H								

定时器3的时钟 ＝ 系统时钟SYSclk ÷ (TM3PS + 1)

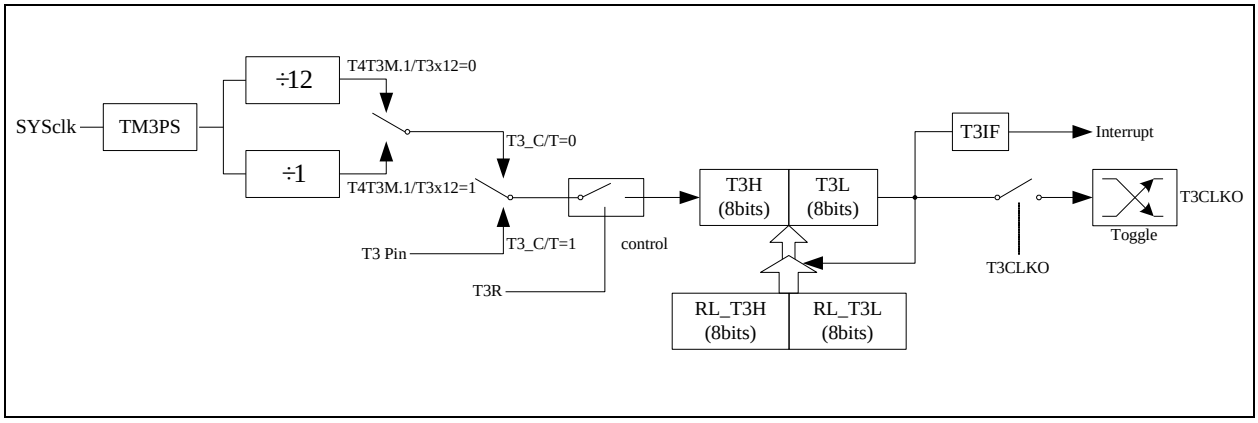
13.4.5 定时器 4 的 8 位预分频寄存器（TM4PS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TM4PS	FEA4H								

定时器4的时钟 ＝ 系统时钟SYSclk ÷ (TM4PS + 1)

13.4.6 定时器 3 工作模式

定时器/计数器 3 的原理框图如下：



定时器/计数器 3 的工作模式：16 位自动重装载模式

T3R/T4T3M.3 为 T4T3M 寄存器内的控制位，T4T3M 寄存器各位的具体功能描述见上节 T4T3M 寄存器的介绍。

当 T3_C/T=0 时，多路开关连接到系统时钟输出，T3 对内部系统时钟计数，T3 工作在定时方式。当 T3_C/T=1 时，多路开关连接到外部脉冲输 T3，即 T3 工作在计数方式。

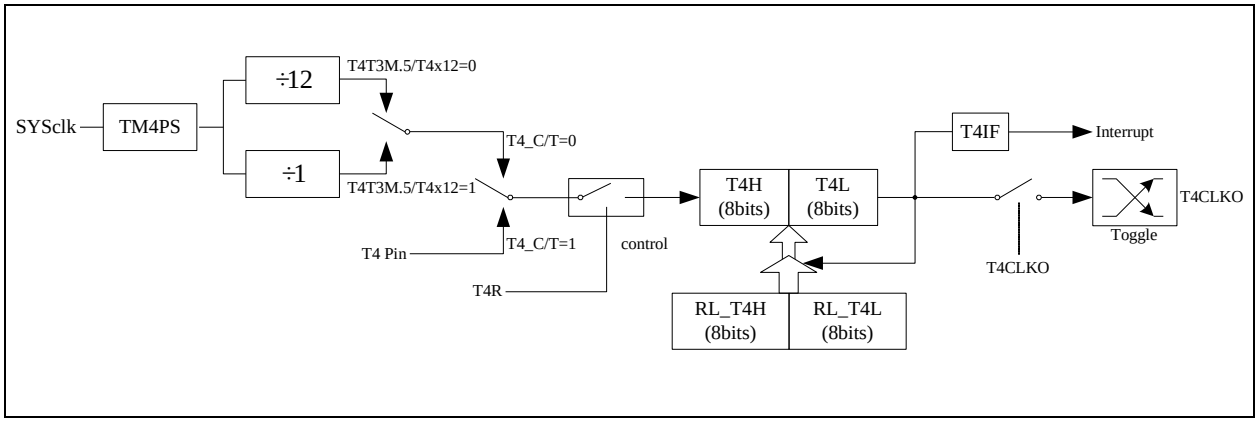
STC 单片机的定时器 3 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T3 的速率由特殊功能寄存器 T4T3M 中的 T3x12 决定，如果 T3x12=0，T3 则工作在 12T 模式；如果 T3x12=1，T3 则工作在 1T 模式

定时器 3 有两个隐藏的寄存器 RL_T3H 和 RL_T3L。RL_T3H 与 T3H 共有同一个地址，RL_T3L 与 T3L 共有同一个地址。当 T3R=0 即定时器/计数器 3 被禁止工作时，对 T3L 写入的内容会同时写入 RL_T3L，对 T3H 写入的内容也会同时写入 RL_T3H。当 T3R=1 即定时器/计数器 3 被允许工作时，对 T3L 写入内容，实际上不是写入当前寄存器 T3L 中，而是写入隐藏的寄存器 RL_T3L 中，对 T3H 写入内容，实际上也不是写入当前寄存器 T3H 中，而是写入隐藏的寄存器 RL_T3H，这样可以巧妙地实现 16 位重装载定时器。当读 T3H 和 T3L 的内容时，所读的内容就是 T3H 和 T3L 的内容，而不是 RL_T3H 和 RL_T3L 的内容。

[T3H,T3L]的溢出不仅置位中断请求标志位（T3IF），使 CPU 转去执行定时器 3 的中断程序，而且会自动将[RL_T3H,RL_T3L]的内容重新装入[T3H,T3L]。

13.4.7 定时器 4 工作模式

定时器/计数器 4 的原理框图如下：



定时器/计数器 4 的工作模式：16 位自动重装载模式

T4R/T4T3M.7 为 T4T3M 寄存器内的控制位，T4T3M 寄存器各位的具体功能描述见上节 T4T3M 寄存器的介绍。

当 T4_C/T=0 时，多路开关连接到系统时钟输出，T4 对内部系统时钟计数，T4 工作在定时方式。当 T4_C/T=1 时，多路开关连接到外部脉冲输 T4，即 T4 工作在计数方式。

STC 单片机的定时器 4 有两种计数速率：一种是 12T 模式，每 12 个时钟加 1，与传统 8051 单片机相同；另外一种是 1T 模式，每个时钟加 1，速度是传统 8051 单片机的 12 倍。T4 的速率由特殊功能寄存器 T4T3M 中的 T4x12 决定，如果 T4x12=0，T4 则工作在 12T 模式；如果 T4x12=1，T4 则工作在 1T 模式

定时器 4 有两个隐藏的寄存器 RL_T4H 和 RL_T4L。RL_T4H 与 T4H 共有同一个地址，RL_T4L 与 T4L 共有同一个地址。当 T4R=0 即定时器/计数器 4 被禁止工作时，对 T4L 写入的内容会同时写入 RL_T4L，对 T4H 写入的内容也会同时写入 RL_T4H。当 T4R=1 即定时器/计数器 4 被允许工作时，对 T4L 写入内容，实际上不是写入当前寄存器 T4L 中，而是写入隐藏的寄存器 RL_T4L 中，对 T4H 写入内容，实际上也不是写入当前寄存器 T4H 中，而是写入隐藏的寄存器 RL_T4H，这样可以巧妙地实现 16 位重装载定时器。当读 T4H 和 T4L 的内容时，所读的内容就是 T4H 和 T4L 的内容，而不是 RL_T4H 和 RL_T4L 的内容。

[T4H,T4L]的溢出不仅置位中断请求标志位（T4IF），使 CPU 转去执行定时器 4 的中断程序，而且会自动将[RL_T4H,RL_T4L]的内容重新装入[T4H,T4L]。

13.4.8 定时器 3 计算公式

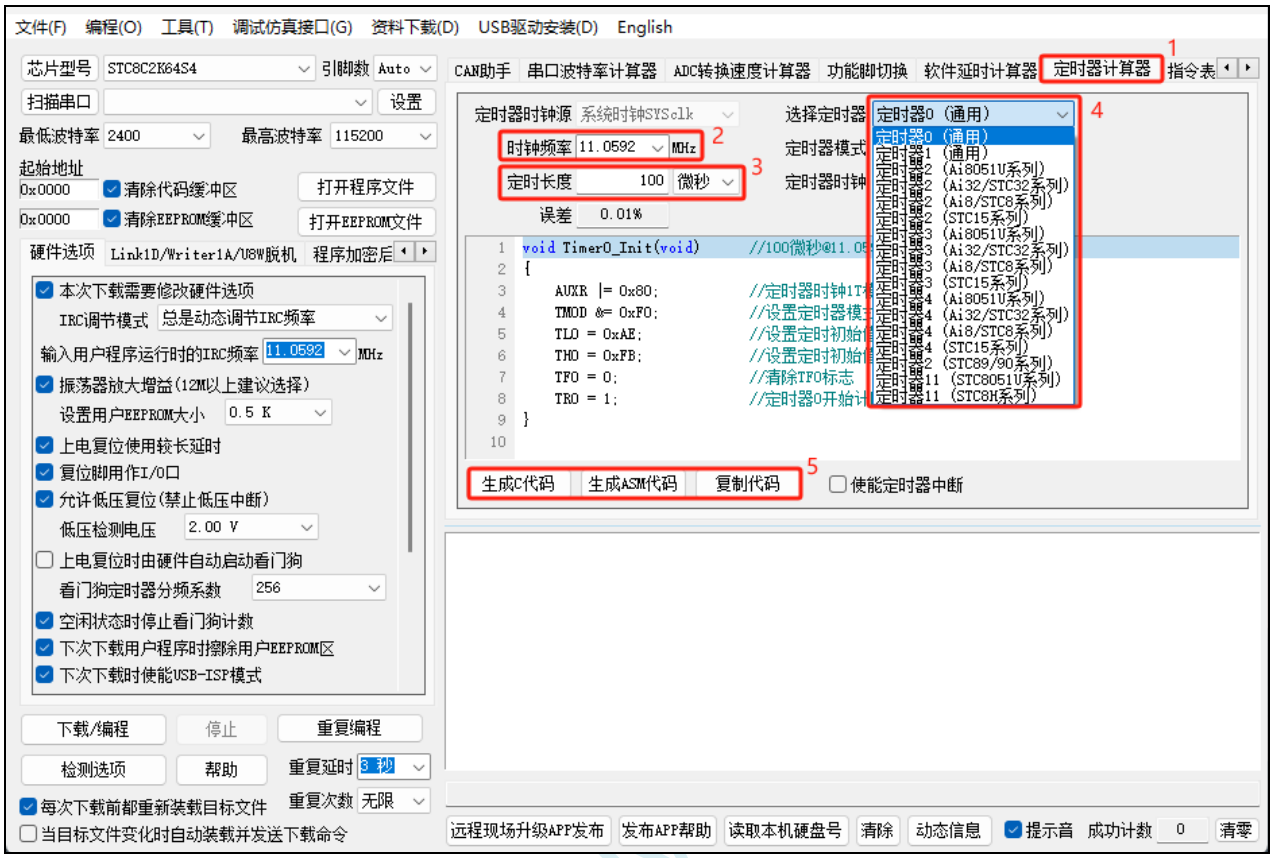
定时器速度	周期计算公式
1T	定时器周期 = $\frac{65536 - [T3H, T3L]}{SYSclk/(TM3PS+1)}$ （自动重载）
12T	定时器周期 = $\frac{65536 - [T3H, T3L]}{SYSclk/(TM3PS+1)} \times 12$ （自动重载）

13.4.9 定时器 4 计算公式

定时器速度	周期计算公式
1T	定时器周期 = $\frac{65536 - [T4H, T4L]}{SYSclk/(TM4PS+1)}$ （自动重载）
12T	定时器周期 = $\frac{65536 - [T4H, T4L]}{SYSclk/(TM4PS+1)} \times 12$ （自动重载）

STC MCU

13.5 AIapp-ISP | 定时器计算器工具



①：在下载软件中选择“定时器计算器”功能页，进定时器代码生成界面

②：设置系统工作频率（单位：MHz）

③：设置定时时间长度（单位：毫秒/微秒）

④：选择目标定时器，并设置定时器工作模式

⑤：选择是否需要使能定时器中断

⑥：手动生成 C 代码或者 ASM 代码，复制范例

13.6 范例程序

13.6.1 定时器 0（模式 0－16 位自动重载），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                                //模式 0
    TL0 = 0x66;                                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                    //启动定时器
    ET0 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

STC MCU

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#00H           ;模式 0
            MOV     TL0,#66H            ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0                 ;启动定时器
            SETB    ET0                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

13.6.2 定时器 0（模式 1－16 位不自动重载），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM0_Isr() interrupt 1
{
    TL0 = 0x66;                 //重设定时参数
    TH0 = 0xfc;
    P10 = !P10;                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x01;                //模式 1
    TL0 = 0x66;                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                    //启动定时器
    ET0 = 1;                    //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1        DATA        093H
P0M0        DATA        094H
```

```
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            MOV     TL0,#66H            ;重设定时参数
            MOV     TH0,#0FCH
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#01H           ;模式 1
            MOV     TL0,#66H            ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0                 ;启动定时器
            SETB    ET0                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

13.6.3 定时器 0（模式 2—8 位自动重载），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
```

```
#include "intrins.h"

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x02;                                //模式 2
    TL0 = 0xf4;                                 //256-11.0592M/12/76K
    TH0 = 0xf4;
    TR0 = 1;                                    //启动定时器
    ET0 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
P1M0        DATA        092H
P2M1        DATA        095H
P2M0        DATA        096H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
```

```
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#02H           ;模式 2
            MOV     TL0,#0F4H           ;256-11.0592M/12/76K
            MOV     TH0,#0F4H
            SETB    TR0                 ;启动定时器
            SETB    ET0                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

13.6.4　定时器 0（模式 3－16 位自动重载不可屏蔽中断），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
```

```
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x03;                                //模式 3
    TL0 = 0x66;                                 //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                                    //启动定时器
    ET0 = 1;                                    //使能定时器中断
//  EA = 1;                                     //不受 EA 控制

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     000BH
```

```
                LJMP        TM0ISR

                ORG         0100H
TM0ISR:
                CPL         P1.0                ;测试端口
                RETI

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                MOV         TMOD,#03H           ;模式 3
                MOV         TL0,#66H            ;65536-11.0592M/12/1000
                MOV         TH0,#0FCH
                SETB        TR0                 ;启动定时器
                SETB        ET0                 ;使能定时器中断
;               SETB        EA                  ;不受 EA 控制

                JMP         $

                END
```

13.6.5 定时器 0（外部计数－扩展 T0 为外部下降沿中断）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P0M1        =       0x93;
sfr         P0M0        =       0x94;
sfr         P1M1        =       0x91;
sfr         P1M0        =       0x92;
sfr         P2M1        =       0x95;
sfr         P2M0        =       0x96;
sfr         P3M1        =       0xb1;
sfr         P3M0        =       0xb2;
sfr         P4M1        =       0xb3;
sfr         P4M0        =       0xb4;
sfr         P5M1        =       0xc9;
sfr         P5M0        =       0xca;

sbit        P10         =       P1^0;
```

```
void TM0_Isr() interrupt 1
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x04;                                //外部计数模式
    TL0 = 0xff;
    TH0 = 0xff;
    TR0 = 1;                                    //启动定时器
    ET0 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     000BH
            LJMP    TM0ISR

            ORG     0100H
TM0ISR:
            CPL     P1.0                    ;测试端口
            RETI

MAIN:
```

```
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     TMOD,#04H           ;外部计数模式
        MOV     TL0,#0FFH
        MOV     TH0,#0FFH
        SETB    TR0                 ;启动定时器
        SETB    ET0                 ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

13.6.6　定时器 0（测量脉宽－INT0 高电平宽度）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     AUXR    =   0x8e;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void INT0_Isr() interrupt 0
{
    P0 = TL0;                       //TL0 为测量值低字节
    P1 = TH0;                       //TH0 为测量值高字节
}

void main()
```

```
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    AUXR = 0x80;                    //1T 模式
    TMOD = 0x08;                    //使能 GATE,INT0 为 1 时使能计时
    TL0 = 0x00;
    TH0 = 0x00;
    while (P32);                    //等待 INT0 为低
    TR0 = 1;                        //启动定时器
    IT0 = 1;                        //使能 INT0 下降沿中断
    EX0 = 1;
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR    DATA    8EH
P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     0003H
        LJMP    INT0ISR

        ORG     0100H
INT0ISR:
        MOV     P0,TL0          ;TL0 为测量值低字节
        MOV     P1,TH0          ;TH0 为测量值高字节
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
```

```
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     AUXR,#80H       ;1T 模式
        MOV     TMOD,#08H       ;使能 GATE,INT0 为 1 时使能计时
        MOV     TL0,#00H
        MOV     TH0,#00H
        JB      P3.2,$          ;等待 INT0 为低
        SETB    TR0             ;启动定时器
        SETB    IT0             ;使能 INT0 下降沿中断
        SETB    EX0
        SETB    EA

        JMP     $

        END
```

13.6.7　定时器 0（模式 0），时钟分频输出

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     INTCLKO     =   0x8f;

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
```

```
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                    //模式 0
    TL0 = 0x66;                     //65536-11.0592M/12/1000
    TH0 = 0xfc;
    TR0 = 1;                        //启动定时器
    INTCLKO = 0x01;                 //使能时钟输出

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

INTCLKO     DATA    8FH
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#00H       ;模式 0
            MOV     TL0,#66H        ;65536-11.0592M/12/1000
            MOV     TH0,#0FCH
            SETB    TR0             ;启动定时器
```

```
        MOV     INTCLKO,#01H            ;使能时钟输出
        JMP     $
        END
```

13.6.8　定时器 1（模式 0－16 位自动重载），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;                             //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                            //模式 0
    TL1 = 0x66;                             //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                //启动定时器
    ET1 = 1;                                //使能定时器中断
    EA = 1;
```

```
    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     001BH
            LJMP    TM1ISR

            ORG     0100H
TM1ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#00H           ;模式 0
            MOV     TL1,#66H            ;65536-11.0592M/12/1000
            MOV     TH1,#0FCH
            SETB    TR1                 ;启动定时器
            SETB    ET1                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

13.6.9 定时器 1（模式 1－16 位不自动重载），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        P10         =   P1^0;

void TM1_Isr() interrupt 3
{
    TL1 = 0x66;                                 //重设定时参数
    TH1 = 0xfc;
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x10;                                //模式 1
    TL1 = 0x66;                                 //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                                    //启动定时器
    ET1 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     001BH
            LJMP    TM1ISR

            ORG     0100H
TM1ISR:
            MOV     TL1,#66H            ;重设定时参数
            MOV     TH1,#0FCH
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#10H           ;模式 1
            MOV     TL1,#66H            ;65536-11.0592M/12/1000
            MOV     TH1,#0FCH
            SETB    TR1                 ;启动定时器
            SETB    ET1                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

13.6.10 定时器 1（模式 2—8 位自动重载），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x20;                                //模式 2
    TL1 = 0xf4;                                 //256-11.0592M/12/76K
    TH1 = 0xf4;
    TR1 = 1;                                    //启动定时器
    ET1 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
```

```
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     001BH
            LJMP    TM1ISR

            ORG     0100H
TM1ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     TMOD,#20H           ;模式 2
            MOV     TL1,#0F4H           ;256-11.0592M/12/76K
            MOV     TH1,#0F4H
            SETB    TR1                 ;启动定时器
            SETB    ET1                 ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

13.6.11 定时器 1（外部计数一扩展 T1 为外部下降沿中断）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
```

```
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM1_Isr() interrupt 3
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x40;                                //外部计数模式
    TL1 = 0xff;
    TH1 = 0xff;
    TR1 = 1;                                    //启动定时器
    ET1 = 1;                                    //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH
```

```
                ORG         0000H
                LJMP        MAIN
                ORG         001BH
                LJMP        TM1ISR

                ORG         0100H
TM1ISR:
                CPL         P1.0                ;测试端口
                RETI

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                MOV         TMOD,#40H           ;外部计数模式
                MOV         TL1,#0FFH
                MOV         TH1,#0FFH
                SETB        TR1                 ;启动定时器
                SETB        ET1                 ;使能定时器中断
                SETB        EA

                JMP         $

                 END
```

13.6.12 定时器 1（测量脉宽－INT1 高电平宽度）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
```

```
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sfr     AUXR    =   0x8e;

void INT1_Isr() interrupt 2
{
    P0 = TL1;                       //TL1 为测量值低字节
    P1 = TH1;                       //TH1 为测量值高字节
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    AUXR = 0x40;                    //1T 模式
    TMOD = 0x80;                    //使能 GATE,INT1 为 1 时使能计时
    TL1 = 0x00;
    TH1 = 0x00;
    while (INT1);                   //等待 INT1 为低
    TR1 = 1;                        //启动定时器
    IT1 = 1;                        //使能 INT1 下降沿中断
    EX1 = 1;
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR    DATA    8EH
P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
```

```
            ORG     0013H
            LJMP    INT1ISR

            ORG     0100H
INT1ISR:
            MOV     P0,TL1              ;TL1 为测量值低字节
            MOV     P1,TH1              ;TH1 为测量值高字节
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     AUXR,#40H           ;1T 模式
            MOV     TMOD,#80H           ;使能 GATE,INT1 为 1 时使能计时
            MOV     TL1,#00H
            MOV     TH1,#00H
            JB      INT1,$              ;等待 INT1 为低
            SETB    TR1                 ;启动定时器
            SETB    IT1                 ;使能 INT1 下降沿中断
            SETB    EX1
            SETB    EA

            JMP     $

            END
```

13.6.13 定时器 1（模式 0），时钟分频输出

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     INTCLKO     =   0x8f;

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
```

```
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    TMOD = 0x00;                    //模式 0
    TL1 = 0x66;                     //65536-11.0592M/12/1000
    TH1 = 0xfc;
    TR1 = 1;                        //启动定时器
    INTCLKO = 0x02;                 //使能时钟输出

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

INTCLKO     DATA    8FH
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
```

```
MOV     P2M1, #00H
MOV     P3M0, #00H
MOV     P3M1, #00H
MOV     P4M0, #00H
MOV     P4M1, #00H
MOV     P5M0, #00H
MOV     P5M1, #00H

MOV     TMOD,#00H           ;模式 0
MOV     TL1,#66H            ;65536-11.0592M/12/1000
MOV     TH1,#0FCH
SETB    TR1                 ;启动定时器
MOV     INTCLKO,#02H        ;使能时钟輸出

JMP     $

END
```

13.6.14 定时器 1（模式 0）做串口 1 波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
```

```
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
```

STC MCU

```
            if (rptr != wptr)
            {
                UartSend(buffer[rptr++]);
                rptr &= 0x0f;
            }
        }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA        8EH

BUSY        BIT         20H.0
WPTR        DATA        21H
RPTR        DATA        22H
BUFFER      DATA        23H                 ;16 bytes

P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
P1M0        DATA        092H
P2M1        DATA        095H
P2M0        DATA        096H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P4M1        DATA        0B3H
P4M0        DATA        0B4H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN
            ORG         0023H
            LJMP        UART_ISR

            ORG         0100H

UART_ISR:
            PUSH        ACC
            PUSH        PSW
            MOV         PSW,#08H

            JNB         TI,CHKRI
            CLR         TI
            CLR         BUSY
CHKRI:
            JNB         RI,UARTISR_EXIT
            CLR         RI
            MOV         A,WPTR
            ANL         A,#0FH
            ADD         A,#BUFFER
            MOV         R0,A
            MOV         @R0,SBUF
            INC         WPTR
UARTISR_EXIT:
            POP         PSW
            POP         ACC
```

```
        RETI

UART_INIT:
        MOV     SCON,#50H
        MOV     TMOD,#00H
        MOV     TL1,#0E8H               ;65536-11059200/115200/4=0FFE8H
        MOV     TH1,#0FFH
        SETB    TR1
        MOV     AUXR,#40H
        CLR     BUSY
        MOV     WPTR,#00H
        MOV     RPTR,#00H
        RET

UART_SEND:
        JB      BUSY,$
        SETB    BUSY
        MOV     SBUF,A
        RET

UART_SENDSTR:
        CLR     A
        MOVC    A,@A+DPTR
        JZ      SENDEND
        LCALL   UART_SEND
        INC     DPTR
        JMP     UART_SENDSTR
SENDEND:
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        LCALL   UART_INIT
        SETB    ES
        SETB    EA

        MOV     DPTR,#STRING
        LCALL   UART_SENDSTR

LOOP:
        MOV     A,RPTR
        XRL     A,WPTR
        ANL     A,#0FH
        JZ      LOOP
        MOV     A,RPTR
        ANL     A,#0FH
```

STC MCU

```
        ADD     A,#BUFFER
        MOV     R0,A
        MOV     A,@R0
        LCALL   UART_SEND
        INC     RPTR
        JMP     LOOP

STRING: DB      'Uart Test !',0DH,0AH,00H

        END
```

13.6.15 定时器 1（模式 2）做串口 1 波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define FOSC 11059200UL
#define BRT (256 - FOSC / 115200 / 32)

sfr AUXR = 0x8e;

sfr P0M1 = 0x93;
sfr P0M0 = 0x94;
sfr P1M1 = 0x91;
sfr P1M0 = 0x92;
sfr P2M1 = 0x95;
sfr P2M0 = 0x96;
sfr P3M1 = 0xb1;
sfr P3M0 = 0xb2;
sfr P4M1 = 0xb3;
sfr P4M0 = 0xb4;
sfr P5M1 = 0xc9;
sfr P5M0 = 0xca;

bit busy;
char wptr;
char rptr;
char buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
```

STC MCU

```
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x20;
    TL1 = BRT;
    TH1 = BRT;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

STC MCU

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA        8EH

BUSY        BIT         20H.0
WPTR        DATA        21H
RPTR        DATA        22H
BUFFER      DATA        23H                 ;16 bytes

P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
P1M0        DATA        092H
P2M1        DATA        095H
P2M0        DATA        096H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P4M1        DATA        0B3H
P4M0        DATA        0B4H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN
            ORG         0023H
            LJMP        UART_ISR

            ORG         0100H

UART_ISR:
            PUSH        ACC
            PUSH        PSW
            MOV         PSW,#08H

            JNB         TI,CHKRI
            CLR         TI
            CLR         BUSY
CHKRI:
            JNB         RI,UARTISR_EXIT
            CLR         RI
            MOV         A,WPTR
            ANL         A,#0FH
            ADD         A,#BUFFER
            MOV         R0,A
            MOV         @R0,SBUF
            INC         WPTR
UARTISR_EXIT:
            POP         PSW
            POP         ACC
            RETI

UART_INIT:
            MOV         SCON,#50H
            MOV         TMOD,#20H
            MOV         TL1,#0FDH           ;256-11059200/115200/32=0FDH
            MOV         TH1,#0FDH
```

STC MCU

```
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     SBUF,A
                RET

UART_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SENDEND
                LCALL   UART_SEND
                INC     DPTR
                JMP     UART_SENDSTR
SENDEND:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART_SEND
                INC     RPTR
                JMP     LOOP
```

STC MCU

```
STRING:     DB      'Uart Test !',0DH,0AH,00H

            END
```

13.6.16 定时器 2（16 位自动重载），用作定时

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2         0x04
sfr     AUXINTIF=   0xef;
#define T2IF        0x01

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
```

```
    T2L = 0x66;                         //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                        //启动定时器
    IE2 = ET2;                          //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0063H
            LJMP    TM2ISR

            ORG     0100H
TM2ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H
```

STC MCU

```
        MOV     T2L,#66H                ;65536-11.0592M/12/1000
        MOV     T2H,#0FCH
        MOV     AUXR,#10H               ;启动定时器
        MOV     IE2,#ET2                ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

13.6.17 定时器 2（外部计数一扩展 T2 为外部下降沿中断）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2         0x04
sfr     AUXINTIF=   0xef;
#define T2IF        0x01

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM2_Isr() interrupt 12
{
    P10 = !P10;                         //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
```

STC MCU

```
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T2L = 0xff;
    T2H = 0xff;
    AUXR = 0x18;                    //设置外部计数模式并启动定时器
    IE2 = ET2;                      //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0063H
            LJMP    TM2ISR

            ORG     0100H
TM2ISR:
            CPL     P1.0            ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
```

STC MCU

```
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     T2L,#0FFH
        MOV     T2H,#0FFH
        MOV     AUXR,#18H       ;设置外部计数模式并启动定时器
        MOV     IE2,#ET2        ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

13.6.18 定时器 2，时钟分频输出

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     INTCLKO =   0x8f;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
```

STC MCU

```
    P5M1 = 0x00;

    T2L = 0x66;                         //65536-11.0592M/12/1000
    T2H = 0xfc;
    AUXR = 0x10;                        //启动定时器
    INTCLKO = 0x04;                     //使能时钟输出

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
INTCLKO     DATA    8FH

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     T2L,#66H            ;65536-11.0592M/12/1000
            MOV     T2H,#0FCH
            MOV     AUXR,#10H           ;启动定时器
            MOV     INTCLKO,#04H        ;使能时钟输出

            JMP     $

            END
```

STC MCU

13.6.19 定时器 2 做串口 1 波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
```

STC MCU

```
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H

BUSY    BIT     20H.0
```

```
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#15H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART_SEND:
            JB      BUSY,$
            SETB    BUSY
```

STC MCU

```
                MOV     SBUF,A
                RET

UART_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SENDEND
                LCALL   UART_SEND
                INC     DPTR
                JMP     UART_SENDSTR
SENDEND:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

13.6.20 定时器 2 做串口 2 波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;
sfr       S2CON       =   0x9a;
sfr       S2BUF       =   0x9b;
sfr       IE2         =   0xaf;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart2Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}

void Uart2Init()
{
    S2CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

STC MCU

```
void Uart2Send(char dat)
{
    while (busy);
    busy = 1;
    S2BUF = dat;
}

void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H
S2CON   DATA    9AH
S2BUF   DATA    9BH
IE2     DATA    0AFH

BUSY    BIT     20H.0
WPTR    DATA    21H
```

```
RPTR        DATA    22H
BUFFER      DATA    23H             ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0043H
            LJMP    UART2_ISR

            ORG     0100H

UART2_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S2CON
            JNB     ACC.1,CHKRI
            ANL     S2CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART2ISR_EXIT
            ANL     S2CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S2BUF
            INC     WPTR
UART2ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART2_INIT:
            MOV     S2CON,#10H
            MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART2_SEND:
            JB      BUSY,$
            SETB    BUSY
```

```
                MOV         S2BUF,A
                RET

UART2_SENDSTR:
                CLR         A
                MOVC        A,@A+DPTR
                JZ          SEND2END
                LCALL       UART2_SEND
                INC         DPTR
                JMP         UART2_SENDSTR
SEND2END:
                RET

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART2_INIT
                MOV         IE2,#01H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART2_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART2_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

13.6.21 定时器 2 做串口 3 波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;
sfr       S3CON       =   0xac;
sfr       S3BUF       =   0xad;
sfr       IE2         =   0xaf;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

STC MCU

```
void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H
S3CON   DATA    0ACH
S3BUF   DATA    0ADH
IE2     DATA    0AFH

BUSY    BIT     20H.0
WPTR    DATA    21H
```

```
RPTR        DATA    22H
BUFFER      DATA    23H             ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR

            ORG     0100H

UART3_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S3CON
            JNB     ACC.1,CHKRI
            ANL     S3CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART3ISR_EXIT
            ANL     S3CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S3BUF
            INC     WPTR
UART3ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART3_INIT:
            MOV     S3CON,#10H
            MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART3_SEND:
            JB      BUSY,$
            SETB    BUSY
```

```
                MOV         S3BUF,A
                RET

UART3_SENDSTR:
                CLR         A
                MOVC        A,@A+DPTR
                JZ          SEND3END
                LCALL       UART3_SEND
                INC         DPTR
                JMP         UART3_SENDSTR
SEND3END:
                RET

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART3_INIT
                MOV         IE2,#08H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART3_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART3_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

13.6.22 定时器 2 做串口 4 波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;
sfr       S4CON       =   0x84;
sfr       S4BUF       =   0x85;
sfr       IE2         =   0xaf;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}
```

STC MCU

```
void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H
S4CON   DATA    84H
S4BUF   DATA    85H
IE2     DATA    0AFH

BUSY    BIT     20H.0
WPTR    DATA    21H
```

```
RPTR        DATA    22H
BUFFER      DATA    23H             ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4_ISR

            ORG     0100H

UART4_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S4CON
            JNB     ACC.1,CHKRI
            ANL     S4CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART4ISR_EXIT
            ANL     S4CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S4BUF
            INC     WPTR
UART4ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART4_INIT:
            MOV     S4CON,#10H
            MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART4_SEND:
            JB      BUSY,$
            SETB    BUSY
```

```
                MOV     S4BUF,A
                RET

UART4_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SEND4END
                LCALL   UART4_SEND
                INC     DPTR
                JMP     UART4_SENDSTR
SEND4END:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART4_INIT
                MOV     IE2,#10H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART4_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART4_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

STC MCU

13.6.23 定时器 3（16 位自动重载），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2     0x04
#define ET3     0x20
#define ET4     0x40
sfr     AUXINTIF=   0xef;
#define T2IF    0x01
#define T3IF    0x02
#define T4IF    0x04

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM3_Isr() interrupt 19
{
    P10 = !P10;                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T3L = 0x66;                 //65536-11.0592M/12/1000
```

STC MCU

```
    T3H = 0xfc;
    T4T3M = 0x08;                   //启动定时器
    IE2 = ET3;                      //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
ET3         EQU     20H
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H
T3IF        EQU     02H
T4IF        EQU     04H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     009BH
            LJMP    TM3ISR

            ORG     0100H
TM3ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
```

STC MCU

```
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     T3L,#66H                ;65536-11.0592M/12/1000
        MOV     T3H,#0FCH
        MOV     T4T3M,#08H              ;启动定时器
        MOV     IE2,#ET3                ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

13.6.24 定时器 3（外部计数－扩展 T3 为外部下降沿中断）

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2         0x04
#define ET3         0x20
#define ET4         0x40
sfr     AUXINTIF=   0xef;
#define T2IF        0x01
#define T3IF        0x02
#define T4IF        0x04

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;
```

```
sbit        P10             =     P1^0;

void TM3_Isr() interrupt 19
{
    P10 = !P10;                         //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T3L = 0x66;                         //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x0c;                       //设置外部计数模式并启动定时器
    IE2 = ET3;                          //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz
T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
ET3         EQU     20H
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H
T3IF        EQU     02H
T4IF        EQU     04H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
```

```
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     009BH
            LJMP    TM3ISR

            ORG     0100H
TM3ISR:
            CPL     P1.0                ;测试端口
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     T3L,#66H            ;65536-11.0592M/12/1000
            MOV     T3H,#0FCH
            MOV     T4T3M,#0CH          ;设置外部计数模式并启动定时器
            MOV     IE2,#ET3            ;使能定时器中断
            SETB    EA

            JMP     $

            END
```

13.6.25 定时器 3，时钟分频输出

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
```

```
sfr     T2H     =   0xd6;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T3L = 0x66;                 //65536-11.0592M/12/1000
    T3H = 0xfc;
    T4T3M = 0x09;               //使能时钟输出并启动定时器

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M   DATA    0D1H
T4L     DATA    0D3H
T4H     DATA    0D2H
T3L     DATA    0D5H
T3H     DATA    0D4H
T2L     DATA    0D7H
T2H     DATA    0D6H

P1M1    DATA    091H
P1M0    DATA    092H
P0M1    DATA    093H
P0M0    DATA    094H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
```

```
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     T3L,#66H            ;65536-11.0592M/12/1000
            MOV     T3H,#0FCH
            MOV     T4T3M,#09H          ;使能时钟输出并启动定时器

            JMP     $

            END
```

13.6.26 定时器 3 做串口 3 波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC      11059200UL
#define   BRT       (65536 - FOSC / 115200 / 4)

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     S3CON   =   0xac;
sfr     S3BUF   =   0xad;
sfr     IE2     =   0xaf;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
```

```
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x50;
    T3L = BRT;
    T3H = BRT >> 8;
    T4T3M = 0x0a;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
```

STC MCU

```
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH
```

```
                ORG     0000H
                LJMP    MAIN
                ORG     008BH
                LJMP    UART3_ISR

                ORG     0100H

UART3_ISR:
                PUSH    ACC
                PUSH    PSW
                MOV     PSW,#08H

                MOV     A,S3CON
                JNB     ACC.1,CHKRI
                ANL     S3CON,#NOT 02H
                CLR     BUSY
CHKRI:
                JNB     ACC.0,UART3ISR_EXIT
                ANL     S3CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S3BUF
                INC     WPTR
UART3ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART3_INIT:
                MOV     S3CON,#50H
                MOV     T3L,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     T3H,#0FFH
                MOV     T4T3M,#0AH
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART3_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     S3BUF,A
                RET

UART3_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SEND3END
                LCALL   UART3_SEND
                INC     DPTR
                JMP     UART3_SENDSTR
SEND3END:
                RET

MAIN:
                MOV     SP, #5FH
```

```
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART3_INIT
                MOV         IE2,#08H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART3_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART3_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

13.6.27 定时器 4（16 位自动重载），用作定时

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
```

```
#define   ET2             0x04
#define   ET3             0x20
#define   ET4             0x40
sfr       AUXINTIF    =   0xef;
#define   T2IF            0x01
#define   T3IF            0x02
#define   T4IF            0x04

sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

sbit      P10         =   P1^0;

void TM4_Isr() interrupt 20
{
    P10 = !P10;                     //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T4L = 0x66;                     //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x80;                   //启动定时器
    IE2 = ET4;                      //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M       DATA        0D1H
T4L         DATA        0D3H
T4H         DATA        0D2H
```

```
T3L     DATA    0D5H
T3H     DATA    0D4H
T2L     DATA    0D7H
T2H     DATA    0D6H
AUXR    DATA    8EH
IE2     DATA    0AFH
ET2     EQU     04H
ET3     EQU     20H
ET4     EQU     40H
AUXINTIF DATA   0EFH
T2IF    EQU     01H
T3IF    EQU     02H
T4IF    EQU     04H

P1M1    DATA    091H
P1M0    DATA    092H
P0M1    DATA    093H
P0M0    DATA    094H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN
        ORG     00A3H
        LJMP    TM4ISR

        ORG     0100H
TM4ISR:
        CPL     P1.0            ;测试端口
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     T4L,#66H        ;65536-11.0592M/12/1000
        MOV     T4H,#0FCH
        MOV     T4T3M,#80H      ;启动定时器
        MOV     IE2,#ET4        ;使能定时器中断
        SETB    EA

        JMP     $
```

STC MCU

END

13.6.28 定时器 4（外部计数一扩展 T4 为外部下降沿中断）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     AUXR    =   0x8e;
sfr     IE2     =   0xaf;
#define ET2     0x04
#define ET3     0x20
#define ET4     0x40
sfr     AUXINTIF=   0xef;
#define T2IF    0x01
#define T3IF    0x02
#define T4IF    0x04

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;

void TM4_Isr() interrupt 20
{
    P10 = !P10;                 //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
```

```
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    T4L = 0x66;                 //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0xc0;               //设置外部计数模式并启动定时器
    IE2 = ET4;                  //使能定时器中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
AUXR        DATA    8EH
IE2         DATA    0AFH
ET2         EQU     04H
ET3         EQU     20H
ET4         EQU     40H
AUXINTIF    DATA    0EFH
T2IF        EQU     01H
T3IF        EQU     02H
T4IF        EQU     04H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00A3H
            LJMP    TM4ISR

            ORG     0100H
TM4ISR:
            CPL     P1.0                ;测试端口
```

```
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     T4L,#66H                ;65536-11.0592M/12/1000
        MOV     T4H,#0FCH
        MOV     T4T3M,#0C0H             ;设置外部计数模式并启动定时器
        MOV     IE2,#ET4                ;使能定时器中断
        SETB    EA

        JMP     $

        END
```

13.6.29 定时器 4，时钟分频输出

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     T4T3M   =   0xd1;
sfr     T4L     =   0xd3;
sfr     T4H     =   0xd2;
sfr     T3L     =   0xd5;
sfr     T3H     =   0xd4;
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;

sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;
```

```
void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P3M1 = 0x00;

    T4L = 0x66;                         //65536-11.0592M/12/1000
    T4H = 0xfc;
    T4T3M = 0x90;                       //使能时钟输出并启动定时器

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
```

STC MCU

```
MOV     P2M1, #00H
MOV     P3M0, #00H
MOV     P3M1, #00H
MOV     P4M0, #00H
MOV     P4M1, #00H
MOV     P5M0, #00H
MOV     P5M1, #00H

MOV     T4L,#66H                ;65536-11.0592M/12/1000
MOV     T4H,#0FCH
MOV     T4T3M,#90H              ;使能时钟输出并启动定时器

JMP     $

END
```

13.6.30 定时器 4 做串口 4 波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       T4T3M       =   0xd1;
sfr       T4L         =   0xd3;
sfr       T4H         =   0xd2;
sfr       T3L         =   0xd5;
sfr       T3H         =   0xd4;
sfr       T2L         =   0xd7;
sfr       T2H         =   0xd6;
sfr       S4CON       =   0x84;
sfr       S4BUF       =   0x85;
sfr       IE2         =   0xaf;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];
```

STC MCU

```
void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart4Init();
    IE2 = 0x10;
```

STC MCU

```
    EA = 1;
    Uart4SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
S4CON       DATA    84H
S4BUF       DATA    85H
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H             ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4_ISR

            ORG     0100H

UART4_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S4CON
```

STC MCU

```
                JNB     ACC.1,CHKRI
                ANL     S4CON,#NOT 02H
                CLR     BUSY
CHKRI:
                JNB     ACC.0,UART4ISR_EXIT
                ANL     S4CON,#NOT 01H
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,S4BUF
                INC     WPTR
UART4ISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART4_INIT:
                MOV     S4CON,#50H
                MOV     T4L,#0E8H               ;65536-11059200/115200/4=0FFE8H
                MOV     T4H,#0FFH
                MOV     T4T3M,#0A0H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART4_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     S4BUF,A
                RET

UART4_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SEND4END
                LCALL   UART4_SEND
                INC     DPTR
                JMP     UART4_SENDSTR
SEND4END:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART4_INIT
```

```
                MOV     IE2,#10H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART4_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART4_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

STC MCU

14 串口通信

STC8C 系列单片机具有 4 个全双工异步串行通信接口。每个串行口由 2 个数据缓冲器、一个移位寄存器、一个串行控制寄存器和一个波特率发生器等组成。每个串行口的数据缓冲器由 2 个互相独立的接收、发送缓冲器构成，可以同时发送和接收数据。

STC8C 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。

14.1 串口相关寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
SCON	串口 1 控制寄存器	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	0000,0000
SBUF	串口 1 数据寄存器	99H									0000,0000
S2CON	串口 2 控制寄存器	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI	0100,0000
S2BUF	串口 2 数据寄存器	9BH									0000,0000
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

STC MCU

14.2 串口 1

14.2.1 串口 1 控制寄存器（SCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SCON	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

SM0/FE：当PCON寄存器中的SMOD0位为1时，该位为帧错误检测标志位。当UART在接收过程中检测到一个无效停止位时，通过UART接收器将该位置1，必须由软件清零。当PCON寄存器中的SMOD0位为0时，该位和SM1一起指定串口1的通信工作模式，如下表所示：

SM0	SM1	串口1工作模式	功能说明
0	0	模式0	同步移位串行方式
0	1	模式1	可变波特率8位数据方式
1	0	模式2	固定波特率9位数据方式
1	1	模式3	可变波特率9位数据方式

SM2：允许模式 2 或模式 3 多机通信控制位。当串口 1 使用模式 2 或模式 3 时，如果 SM2 位为 1 且 REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 RB8）来筛选地址帧，若 RB8=1，说明该帧是地址帧，地址信息可以进入 SBUF，并使 RI 为 1，进而在中断服务程序中再进行地址号比较；若 RB8=0，说明该帧不是地址帧，应丢掉且保持 RI=0。在模式 2 或模式 3 中，如果 SM2 位为 0 且 REN 位为 1，接收收机处于地址帧筛选被禁止状态，不论收到的 RB8 为 0 或 1，均可使接收到的信息进入 SBUF，并使 RI=1，此时 RB8 通常为校验位。模式 1 和模式 0 为非多机通信方式，在这两种方式时，SM2 应设置为 0。

REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

TB8：当串口 1 使用模式 2 或模式 3 时，TB8 为要发送的第 9 位数据，按需要由软件置位或清 0。在模式 0 和模式 1 中，该位不用。

RB8：当串口 1 使用模式 2 或模式 3 时，RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 和模式 1 中，该位不用。

TI：串口 1 发送中断请求标志位。在模式 0 中，当串口发送数据第 8 位结束时，由硬件自动将 TI 置 1，向主机请求中断，响应中断后 TI 必须用软件清零。在其他模式中，则在停止位开始发送时由硬件自动将 TI 置 1，向 CPU 发请求中断，响应中断后 TI 必须用软件清零。

RI：串口 1 接收中断请求标志位。在模式 0 中，当串口接收第 8 位数据结束时，由硬件自动将 RI 置 1，向主机请求中断，响应中断后 RI 必须用软件清零。在其他模式中，串行接收到停止位的中间时刻由硬件自动将 RI 置 1，向 CPU 发中断申请，响应中断后 RI 必须由软件清零。

14.2.2 串口 1 数据寄存器（SBUF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SBUF	99H								

SBUF：串口 1 数据接收/发送缓冲区。SBUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 SBUF 进行读操作，实际是读取串口接收缓冲区，对 SBUF 进行写操作则是触发串口开始发送数据。

14.2.3 电源管理寄存器（PCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
PCON	87H	SMOD	SMOD0	LVDF	POF	GF1	GF0	PD	IDL

SMOD：串口 1 波特率控制位

0：串口 1 的各个模式的波特率都不加倍

1：串口 1 模式 1（定时器 1 的模式 2 作为波特率发生器时有效）、模式 2、模式 3（定时器 1 的模式 2 作为波特率发生器时有效）的波特率加倍

SMOD0：帧错误检测控制位

0：无帧错检测功能

1：使能帧错误检测功能。此时 SCON 的 SM0/FE 为 FE 功能，即为帧错误检测标志位。

14.2.4 辅助寄存器 1（AUXR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
AUXR	8EH	T0x12	T1x12	UART_M0x6	T2R	T2_C/T	T2x12	EXTRAM	S1ST2

UART_M0x6：串口 1 模式 0 的通讯速度控制

0：串口 1 模式 0 的波特率不加倍，固定为 Fosc/12

1：串口 1 模式 0 的波特率 6 倍速，即固定为 Fosc/12*6 ＝ Fosc/2

S1ST2：串口 1 波特率发生器选择位

0：选择定时器 1 作为波特率发生器

1：选择定时器 2 作为波特率发生器

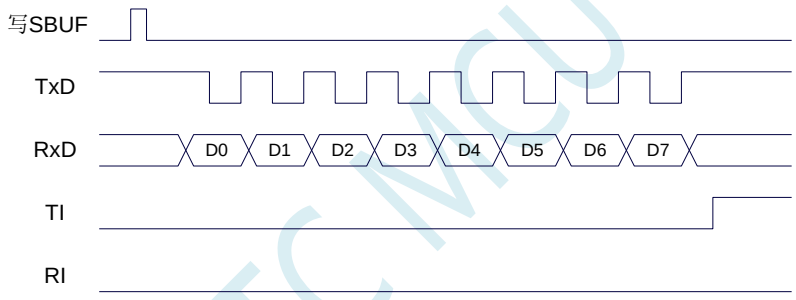
STC MCU

14.2.5 串口 1 模式 0，模式 0 波特率计算公式

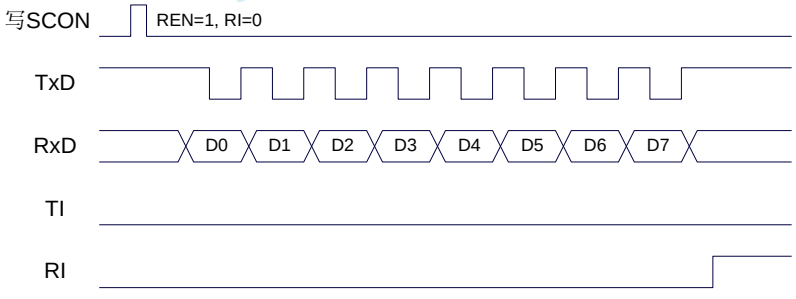
当串口 1 选择工作模式为模式 0 时，串行通信接口工作在同步移位寄存器模式，当串行口模式 0 的通信速度设置位 UART_M0x6 为 0 时，其波特率固定为系统时钟时钟的 12 分频（SYSclk/12）；当设置 UART_M0x6 为 1 时，其波特率固定为系统时钟频率的 2 分频（SYSclk/2）。RxD 为串行通讯的数据口，TxD 为同步移位脉冲输出脚，发送、接收的是 8 位数据，低位在先。

模式 0 的发送过程：当主机执行将数据写入发送缓冲器 SBUF 指令时启动发送，串行口即将 8 位数据以 SYSclk/12 或 SYSclk/2（由 UART_M0x6 确定是 12 分频还是 2 分频）的波特率从 RxD 管脚输出(从低位到高位)，发送完中断标志 TI 置 1，TxD 管脚输出同步移位脉冲信号。当写信号有效后，相隔一个时钟，发送控制端 SEND 有效(高电平)，允许 RxD 发送数据，同时允许 TxD 输出同步移位脉冲。一帧(8 位)数据发送完毕时，各控制端均恢复原状态，只有 TI 保持高电平，呈中断申请状态。在再次发送数据前，必须用软件将 TI 清 0。

模式 0 的接收过程：首先将接收中断请求标志 RI 清零并置位允许接收控制位 REN 时启动模式 0 接收过程。启动接收过程后，RxD 为串行数据输入端，TxD 为同步脉冲输出端。串行接收的波特率为 SYSclk/12 或 SYSclk/2（由 UART_M0x6 确定是 12 分频还是 2 分频）。当接收完成一帧数据（8 位）后，控制信号复位，中断标志 RI 被置 1，呈中断申请状态。当再次接收时，必须通过软件将 RI 清 0



发送数据（串口1模式0）



接收数据（串口1模式0）

STC MCU

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 $= \frac{\text{SYSclk}}{2}$

STC MCU

14.2.6　串口 1 模式 1，模式 1 波特率计算公式

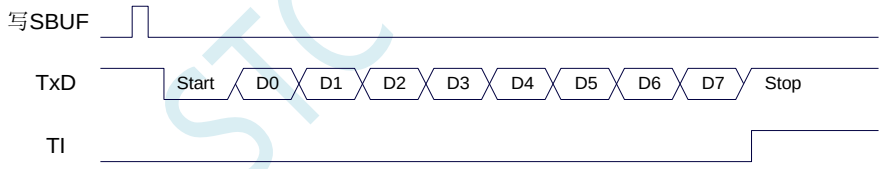
当软件设置 SCON 的 SM0、SM1 为“01”时，串行口 1 则以模式 1 进行工作。此模式为 8 位 UART 格式，一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，即可根据需要进行设置波特率。TxD 为数据发送口，RxD 为数据接收口，串行口全双工接受/发送。

模式 1 的发送过程：串行通信模式发送时，数据由串行发送端 TxD 输出。当主机执行一条写 SBUF 的指令就启动串行通信的发送，写“SBUF”信号还把“1”装入发送移位寄存器的第 9 位，并通知 TX 控制单元开始发送。移位寄存器将数据不断右移送 TxD 端口发送，在数据的左边不断移入“0”作补充。当数据的最高位移到移位寄存器的输出位置，紧跟其后的是第 9 位“1”，在它的左边各位全为“0”，这个状态条件，使 TX 控制单元作最后一次移位输出，然后使允许发送信号“SEND”失效，完成一帧信息的发送，并置位中断请求位 TI，即 TI=1，向主机请求中断处理。

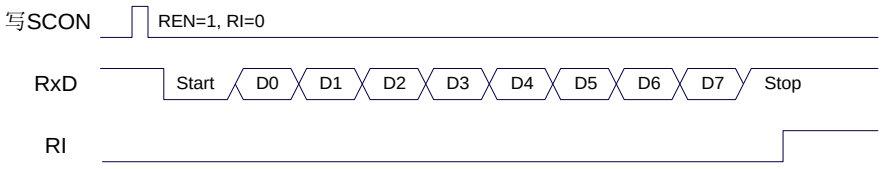
模式 1 的接收过程：当软件置位接收允许标志位 REN，即 REN=1 时，接收器便对 RxD 端口的信号进行检测，当检测到 RxD 端口发送从“1”→“0”的下降沿跳变时就启动接收器准备接收数据，并立即复位波特率发生器的接收计数器，将 1FFH 装入移位寄存器。接收的数据从接收移位寄存器的右边移入，已装入的 1FFH 向左边移出，当起始位"0"移到移位寄存器的最左边时，使 RX 控制器作最后一次移位，完成一帧的接收。若同时满足以下两个条件：

- RI=0；
- SM2=0 或接收到的停止位为 1。

则接收到的数据有效，实现装载入 SBUF，停止位进入 RB8，RI 标志位被置 1，向主机请求中断，若上述两条件不能同时满足，则接收到的数据作废并丢失，无论条件满足与否，接收器重又检测 RxD 端口上的"1"→"0"的跳变，继续下一帧的接收。接收有效，在响应中断后，RI 标志位必须由软件清 0。通常情况下，串行通信工作于模式 1 时，SM2 设置为"0"。



发送数据（串口1模式1）



接收数据（串口1模式1）

串口 1 的波特率是可变的，其波特率可由定时器 1 或者定时器 2 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 1 模式 1 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536 - \frac{SYSclk}{4\times波特率}$
	12T	定时器2重载值 = $65536 - \frac{SYSclk}{12\times4\times波特率}$
定时器1模式0	1T	定时器1重载值 = $65536 - \frac{SYSclk}{4\times波特率}$
	12T	定时器1重载值 = $65536 - \frac{SYSclk}{12\times4\times波特率}$
定时器1模式2	1T	定时器1重载值 = $256 - \frac{2^{SMOD}\times SYSclk}{32\times波特率}$
	12T	定时器1重载值 = $256 - \frac{2^{SMOD}\times SYSclk}{12\times32\times波特率}$

下面为常用频率与常用波特率所对应定时器的重载值

频率 (MHz)	波特率	定时器 2		定时器 1 模式 0		定时器 1 模式 2			
						SMOD=1		SMOD=0	
		1T 模式	12T 模式	1T 模式	12T 模式	1T 模式	12T 模式	1T 模式	12T 模式
11.0592	115200	FFE8H	FFFEH	FFE8H	FFFEH	FAH	-	FDH	-
	57600	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	38400	FFB8H	FFFAH	FFB8H	FFFAH	EEH	-	F7H	-
	19200	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	9600	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
18.432	115200	FFD8H	-	FFD8H	-	F6H	-	FBH	-
	57600	FFB0H	-	FFB0H	-	ECH	-	F6H	-
	38400	FF88H	FFF6H	FF88H	FFF6H	E2H	-	F1H	-
	19200	FF10H	FFECH	FF10H	FFECH	C4H	FBH	E2H	-
	9600	FE20H	FFD8H	FE20H	FFD8H	88H	F6H	C4H	FBH
22.1184	115200	FFD0H	FFFCH	FFD0H	FFFCH	F4H	FFH	FAH	-
	57600	FFA0H	FFF8H	FFA0H	FFF8H	E8H	FEH	F4H	FFH
	38400	FF70H	FFF4H	FF70H	FFF4H	DCH	FDH	EEH	-
	19200	FEE0H	FFE8H	FEE0H	FFE8H	B8H	FAH	DCH	FDH
	9600	FDC0H	FFD0H	FDC0H	FFD0H	70H	F4H	B8H	FAH

14.2.7 串口 1 模式 2，模式 2 波特率计算公式

当 SM0、SM1 两位为 10 时，串行口 1 工作在模式 2。串行口 1 工作模式 2 为 9 位数据异步通信 UART 模式，其一帧的信息由 11 位组成：1 位起始位，8 位数据位（低位在先），1 位可编程位（第 9 位数据）和 1 位停止位。发送时可编程位（第 9 位数据）由 SCON 中的 TB8 提供，可软件设置为 1 或 0，或者可将 PSW 中的奇/偶校验位 P 值装入 TB8（TB8 既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位）。接收时第 9 位数据装入 SCON 的 RB8。TxD 为发送端口，RxD 为接收端口，以全双工模式进行接收/发送。

模式 2 的波特率固定为系统时钟的 64 分频或 32 分频（取决于 PCON 中 SMOD 的值）

串口 1 模式 2 的波特率计算公式如下表所示（SYSclk 为系统工作频率）：

SMOD	波特率计算公式
0	波特率 $= \frac{SYSclk}{64}$
1	波特率 $= \frac{SYSclk}{32}$

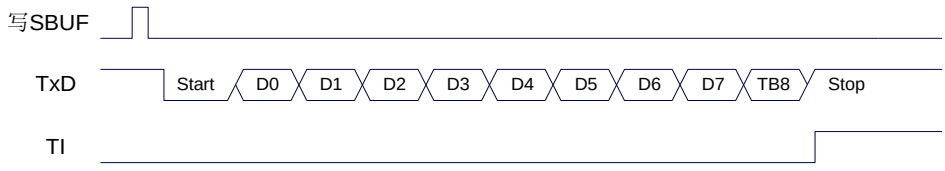
模式 2 和模式 1 相比，除波特率发生源略有不同，发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收/发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

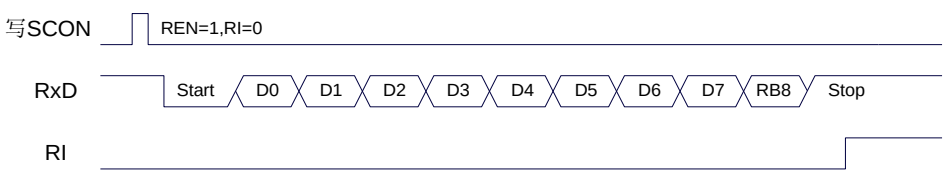
- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 RxD 输入端口的跳变信息，接收下一帧的输入信息。在模式 2 中，接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信 Ð 议的约定，为多机通信提供了方便。



发送数据（串口1模式2）



接收数据（串口1模式2）

14.2.8 串口 1 模式 3，模式 3 波特率计算公式

当 SM0、SM1 两位为 11 时，串行口 1 工作在模式 3。串行通信模式 3 为 9 位数据异步通信 UART 模式，其一帧的信息由 11 位组成：1 位起始位，8 位数据位（低位在先），1 位可编程位（第 9 位数据）和 1 位停止位。发送时可编程位（第 9 位数据）由 SCON 中的 TB8 提供，可软件设置为 1 或 0，或者可将 PSW 中的奇/偶校验位 P 值装入 TB8（TB8 既可作为多机通信中的地址数据标志位，又可作为数据的奇偶校验位）。接收时第 9 位数据装入 SCON 的 RB8。TxD 为发送端口，RxD 为接收端口，以全双工模式进行接收/发送。

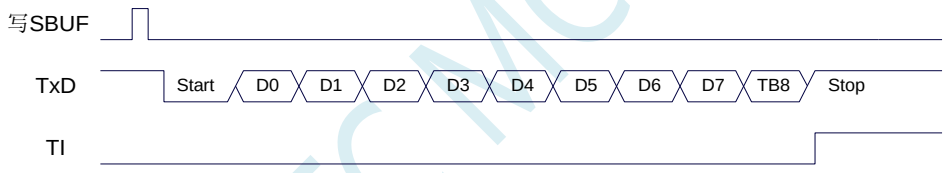
模式 3 和模式 1 相比，除发送时由 TB8 提供给移位寄存器第 9 数据位不同外，其余功能结构均基本相同，其接收‘发送操作过程及时序也基本相同。

当接收器接收完一帧信息后必须同时满足下列条件：

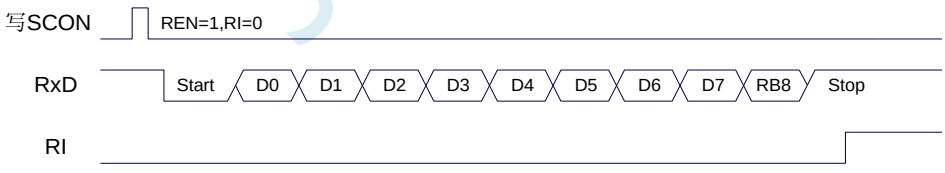
- RI=0
- SM2=0 或者 SM2=1 且接收到的第 9 数据位 RB8=1。

当上述两条件同时满足时，才将接收到的移位寄存器的数据装入 SBUF 和 RB8 中，RI 标志位被置 1，并向主机请求中断处理。如果上述条件有一个不满足，则刚接收到移位寄存器中的数据无效而丢失，也不置位 RI。无论上述条件满足与否，接收器又重新开始检测 RxD 输入端口的跳变信息，接收下一帧的输入信息。在模式 3 中，接收到的停止位与 SBUF、RB8 和 RI 无关。

通过软件对 SCON 中的 SM2、TB8 的设置以及通信协议的约定，为多机通信提供了方便。



发送数据（串口1模式3）



接收数据（串口1模式3）

串口 1 模式 3 的波特率计算公式与模式 1 是完全相同的。请参考模式 1 的波特率计算公式。

14.2.9 自动地址识别，从机地址控制寄存器（SADDR，SADEN）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SADDR	A9H								
SADEN	B9H								

SADDR：从机地址寄存器

SADEN：从机地址屏蔽位寄存器

自动地址识别功能典型应用在多机通讯领域，其主要原理是从机系统通过硬件比较功能来识别来自于主机串口数据流中的地址信息，通过寄存器 SADDR 和 SADEN 设置的本机的从机地址，硬件自动对从机地址进行过滤，当来自于主机的从机地址信息与本机所设置的从机地址相匹配时，硬件产生串口中断；否则硬件自动丢弃串口数据，而不产生中断。当众多处于空闲模式的从机链接在一起时，只有从机地址相匹配的从机才会从空闲模式唤醒，从而可以大大降低从机 MCU 的功耗，即使从机处于正常工作状态也可避免不停地进入串口中断而降低系统执行效率。

要使用串口的自动地址识别功能，首先需要将参与通讯的 MCU 的串口通讯模式设置为模式 2 或者模式 3（通常都选择波特率可变的模式 3，因为模式 2 的波特率是固定的，不便于调节），并开启从机的 SCON 的 SM2 位。对于串口模式 2 或者模式 3 的 9 位数据位中，第 9 位数据（存放在 RB8 中）为地址/数据的标志位，当第 9 位数据为 1 时，表示前面的 8 位数据（存放在 SBUF 中）为地址信息。当 SM2 被设置为 1 时，从机 MCU 会自动过滤掉非地址数据（第 9 位为 0 的数据），而对 SBUF 中的地址数据（第 9 位为 1 的数据）自动与 SADDR 和 SADEN 所设置的本机地址进行比较，若地址相匹配，则会将 RI 置“1”，并产生中断，否则不予处理本次接收的串口数据。

从机地址的设置是通过 SADDR 和 SADEN 两个寄存器进行设置的。SADDR 为从机地址寄存器，里面存放本机的从机地址。SADEN 为从机地址屏蔽位寄存器，用于设置地址信息中的忽略位，设置方法如下：

例如

SADDR ＝ 11001010

SADEN ＝ 10000001

则匹配地址为 1xxxxxx0

即，只要主机送出的地址数据中的 bit0 为 0 且 bit7 为 1 就可以和本机地址相匹配

再例如

SADDR ＝ 11001010

SADEN ＝ 00001111

则匹配地址为 xxxx1010

即，只要主机送出的地址数据中的低 4 位为 1010 就可以和本机地址相匹配，而高 4 为被忽略，可以为任意值。

主机可以使用广播地址（FFH）同时选中所有的从机来进行通讯。

14.3 串口 2

14.3.1 串口 2 控制寄存器（S2CON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2CON	9AH	S2SM0	-	S2SM2	S2REN	S2TB8	S2RB8	S2TI	S2RI

S2SM0：指定串口2的通信工作模式，如下表所示：

S2SM0	串口2工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S2SM2：允许串口 2 在模式 1 时允许多机通信控制位。在模式 1 时，如果 S2SM2 位为 1 且 S2REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 S2RB8）来筛选地址帧：若 S2RB8=1，说明该帧是地址帧，地址信息可以进入 S2BUF，并使 S2RI 为 1，进而在中断服务程序中再进行地址号比较；若 S2RB8=0，说明该帧不是地址帧，应丢掉且保持 S2RI=0。在模式 1 中，如果 S2SM2 位为 0 且 S2REN 位为 1，接收收机处于地址帧筛选被禁止状态。不论收到的 S2RB8 为 0 或 1，均可使接收到的信息进入 S2BUF，并使 S2RI=1，此时 S2RB8 通常为校验位。模式 0 为非多机通信方式，在这种方式时，要设置 S2SM2 应为 0。

S2REN：允许/禁止串口接收控制位

0：禁止串口接收数据

1：允许串口接收数据

S2TB8：当串口 2 使用模式 1 时，S2TB8 为要发送的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位，按需要由软件置位或清 0。在模式 0 中，该位不用。

S2RB8：当串口 2 使用模式 1 时，S2RB8 为接收到的第 9 位数据，一般用作校验位或者地址帧/数据帧标志位。在模式 0 中，该位不用。

S2TI：串口 2 发送中断请求标志位。在停止位开始发送时由硬件自动将 S2TI 置 1，向 CPU 发请求中断，响应中断后 S2TI 必须用软件清零。

S2RI：串口 2 接收中断请求标志位。串行接收到停止位的中间时刻由硬件自动将 S2RI 置 1，向 CPU 发中断申请，响应中断后 S2RI 必须由软件清零。

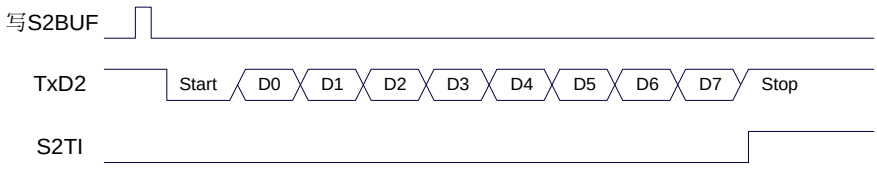
14.3.2 串口 2 数据寄存器（S2BUF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S2BUF	9BH								

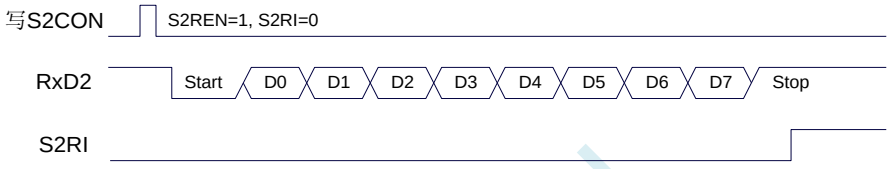
S2BUF：串口 1 数据接收/发送缓冲区。S2BUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 S2BUF 进行读操作，实际是读取串口接收缓冲区，对 S2BUF 进行写操作则是触发串口开始发送数据。

14.3.3　串口 2 模式 0，模式 0 波特率计算公式

串行口 2 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD2 为数据发送口，RxD2 为数据接收口，串行口全双工接受/发送。



发送数据（串口2模式0）



接收数据（串口2模式0）

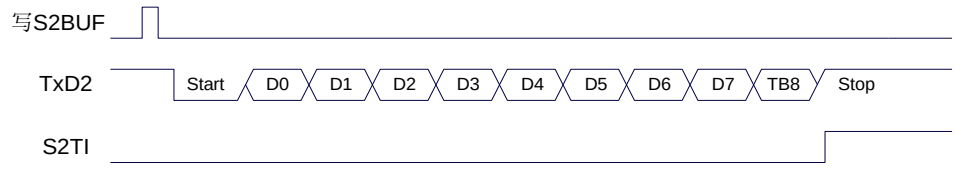
串口 2 的波特率是可变的，其波特率由定时器 2 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

串口 2 模式 0 的波特率计算公式如下表所示：（SYSclk 为系统工作频率）

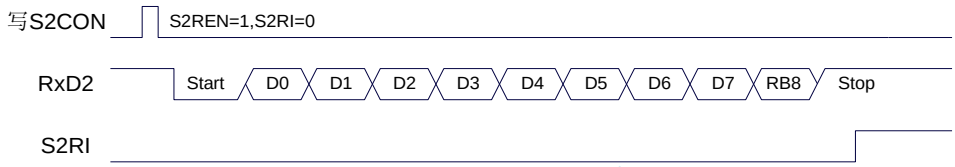
选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536-\frac{SYSclk}{4\times波特率}$
	12T	定时器2重载值 = $65536-\frac{SYSclk}{12\times4\times波特率}$

14.3.4 串口 2 模式 1，模式 1 波特率计算公式

串行口 2 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD2 为数据发送口，RxD2 为数据接收口，串行口全双工接受/发送。



发送数据（串口2模式1）



接收数据（串口2模式1）

串口 2 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

STC MCU

14.4 串口 3

14.4.1 串口 3 控制寄存器（S3CON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3CON	ACH	S3SM0	S3ST3	S3SM2	S3REN	S3TB8	S3RB8	S3TI	S3RI

S3SM0：指定串口3的通信工作模式，如下表所示：

S3SM0	串口3工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S3ST3：选择串口 3 的波特率发生器

0：选择定时器 2 为串口 3 的波特率发生器

1：选择定时器 3 为串口 3 的波特率发生器

S3SM2：允许串口 3 在模式 1 时允许多机通信控制位。在模式 1 时，如果 S3SM2 位为 1 且 S3REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 S3RB8）来筛选地址帧：若 S3RB8=1，说明该帧是地址帧，地址信息可以进入 S3BUF，并使 S3RI 为 1，进而在中断服务程序中再进行地址号比较；若 S3RB8=0，说明该帧不是地址帧，应丢掉且保持 S3RI=0。在模式 1 中，如果 S3SM2 位为 0 且 S3REN 位为 1，接收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。

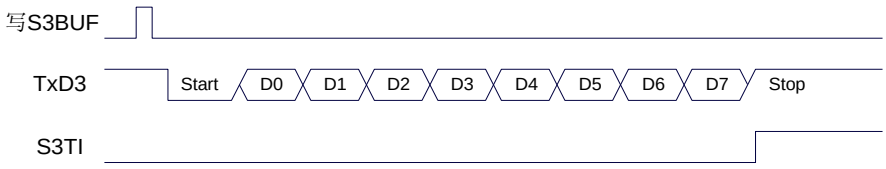
14.4.2 串口 3 数据寄存器（S3BUF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S3BUF	ADH								

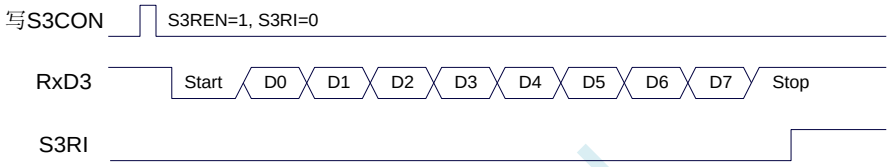
S3BUF：串口 1 数据接收/发送缓冲区。S3BUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 S3BUF 进行读操作，实际是读取串口接收缓冲区，对 S3BUF 进行写操作则是触发串口开始发送数据。

14.4.3　串口 3 模式 0，模式 0 波特率计算公式

串行口 3 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD3 为数据发送口，RxD3 为数据接收口，串行口全双工接受/发送。



发送数据（串口3模式0）



接收数据（串口3模式0）

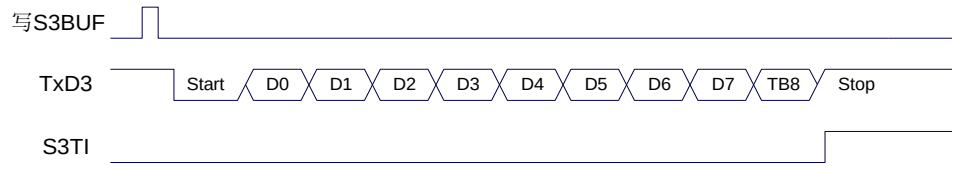
串口 3 的波特率是可变的，其波特率可由定时器 2 或定时器 3 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

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）

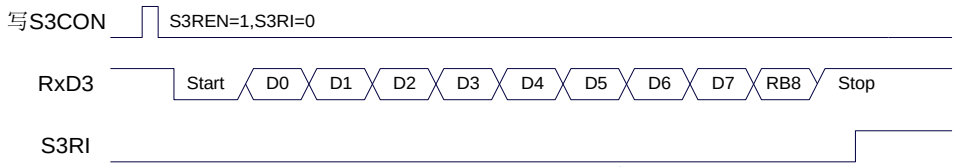
选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536-\frac{\text{SYSclk}}{4\times\text{波特率}}$
	12T	定时器2重载值 = $65536-\frac{\text{SYSclk}}{12\times4\times\text{波特率}}$
定时器3	1T	定时器3重载值 = $65536-\frac{\text{SYSclk}}{4\times\text{波特率}}$
	12T	定时器3重载值 = $65536-\frac{\text{SYSclk}}{12\times4\times\text{波特率}}$

14.4.4 串口 3 模式 1，模式 1 波特率计算公式

串行口 3 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD3 为数据发送口，RxD3 为数据接收口，串行口全双工接受/发送。



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。

STC MCU

14.5 串口 4

14.5.1 串口 4 控制寄存器（S4CON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S4CON	84H	S4SM0	S4ST4	S4SM2	S4REN	S4TB8	S4RB8	S4TI	S4RI

S4SM0：指定串口4的通信工作模式，如下表所示：

S4SM0	串口4工作模式	功能说明
0	模式0	可变波特率8位数据方式
1	模式1	可变波特率9位数据方式

S4ST4：选择串口 4 的波特率发生器

0：选择定时器 2 为串口 4 的波特率发生器

1：选择定时器 4 为串口 4 的波特率发生器

S4SM2：允许串口 4 在模式 1 时允许多机通信控制位。在模式 1 时，如果 S4SM2 位为 1 且 S4REN 位为 1，则接收机处于地址帧筛选状态。此时可以利用接收到的第 9 位（即 S4RB8）来筛选地址帧：若 S4RB8=1，说明该帧是地址帧，地址信息可以进入 S4BUF，并使 S4RI 为 1，进而在中断服务程序中再进行地址号比较；若 S4RB8=0，说明该帧不是地址帧，应丢掉且保持 S4RI=0。在模式 1 中，如果 S4SM2 位为 0 且 S4REN 位为 1，接收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。

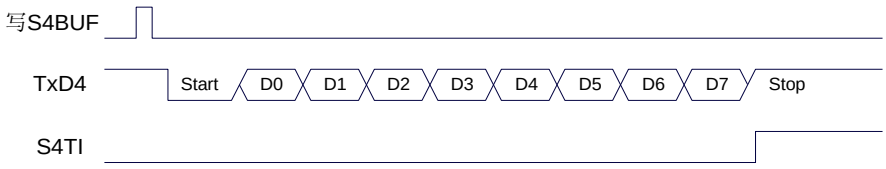
14.5.2 串口 4 数据寄存器（S4BUF）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
S4BUF	85H								

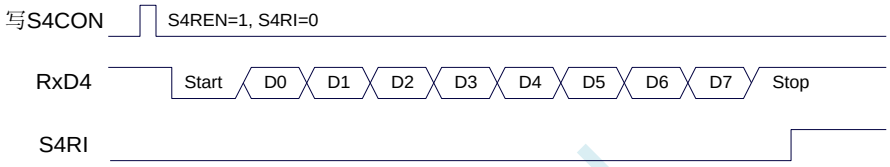
S4BUF：串口 1 数据接收/发送缓冲区。S4BUF 实际是 2 个缓冲器，读缓冲器和写缓冲器，两个操作分别对应两个不同的寄存器，1 个是只写寄存器（写缓冲器），1 个是只读寄存器（读缓冲器）。对 S4BUF 进行读操作，实际是读取串口接收缓冲区，对 S4BUF 进行写操作则是触发串口开始发送数据。

14.5.3　串口 4 模式 0，模式 0 波特率计算公式

串行口 4 的模式 0 为 8 位数据位可变波特率 UART 工作模式。此模式一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD4 为数据发送口，RxD4 为数据接收口，串行口全双工接受/发送。



发送数据（串口4模式0）



接收数据（串口4模式0）

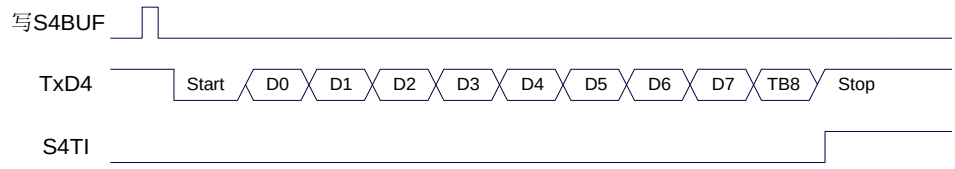
串口 4 的波特率是可变的，其波特率可由定时器 2 或定时器 4 产生。当定时器采用 1T 模式时（12 倍速），相应的波特率的速度也会相应提高 12 倍。

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）

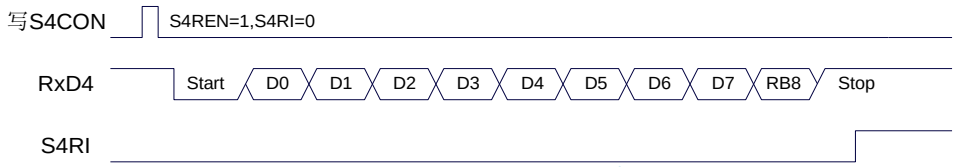
选择定时器	定时器速度	波特率计算公式
定时器2	1T	定时器2重载值 = $65536-\frac{\text{SYSclk}}{4\times\text{波特率}}$
	12T	定时器2重载值 = $65536-\frac{\text{SYSclk}}{12\times4\times\text{波特率}}$
定时器4	1T	定时器4重载值 = $65536-\frac{\text{SYSclk}}{4\times\text{波特率}}$
	12T	定时器4重载值 = $65536-\frac{\text{SYSclk}}{12\times4\times\text{波特率}}$

14.5.4 串口 4 模式 1，模式 1 波特率计算公式

串行口 4 的模式 1 为 9 位数据位可变波特率 UART 工作模式。此模式一帧信息为 11 位：1 位起始位，9 位数据位（低位在先）和 1 位停止位。波特率可变，可根据需要进行设置波特率。TxD4 为数据发送口，RxD4 为数据接收口，串行口全双工接受/发送。



发送数据（串口4模式1）



接收数据（串口4模式1）

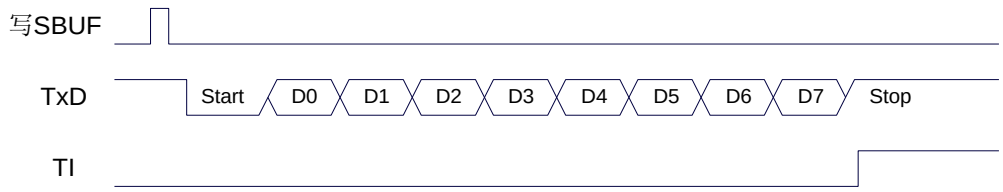
串口 4 模式 1 的波特率计算公式与模式 0 是完全相同的。请参考模式 0 的波特率计算公式。

STC MCU

14.6 串口注意事项

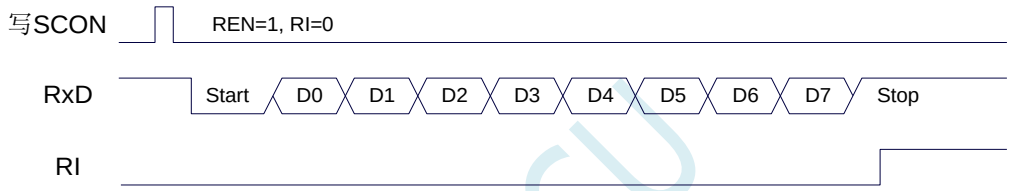
关于串口中断请求有如下问题需要注意：（串口 1、串口 2、串口 3、串口 4 均类似，下面以串口 1 为例进行说明）

8 位数据模式时，发送完成约 1/3 个停止位后产生 TI 中断请求，如下图所示：



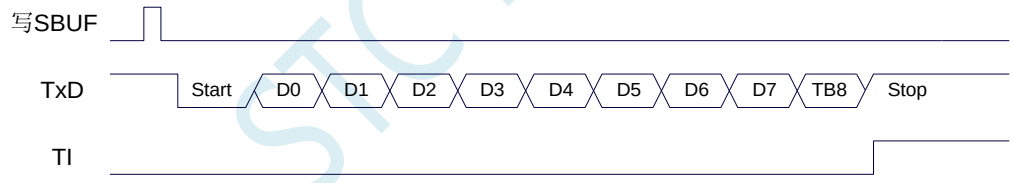
发送数据（8位数据）

8 位数据模式时，接收完成一半个停止位后产生 RI 中断请求，如下图所示：



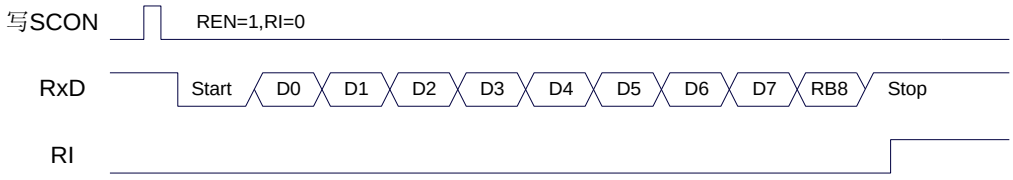
接收数据（8位数据）

9 位数据模式时，发送完成约 1/3 个停止位后产生 TI 中断请求：



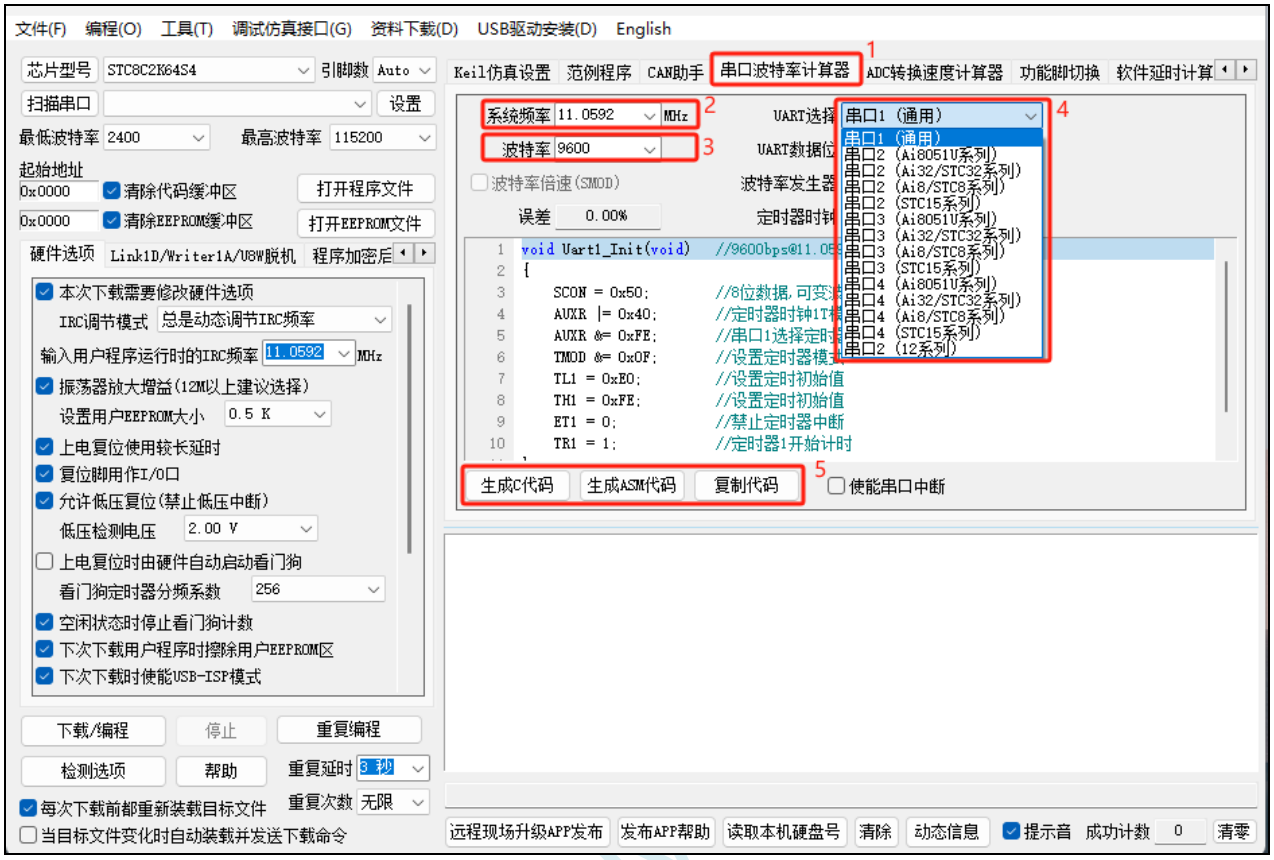
发送数据（9位数据）

9 位数据模式时，一半个停止位后产生 RI 中断请求，如下图所示：



接收数据（9位数据）

14.7 AIapp-ISP | 串口波特率计算器工具



①：在下载软件中选择“串口波特率计算器”功能页，进入串口代码生成界面

②：设置系统工作频率（单位：MHz）

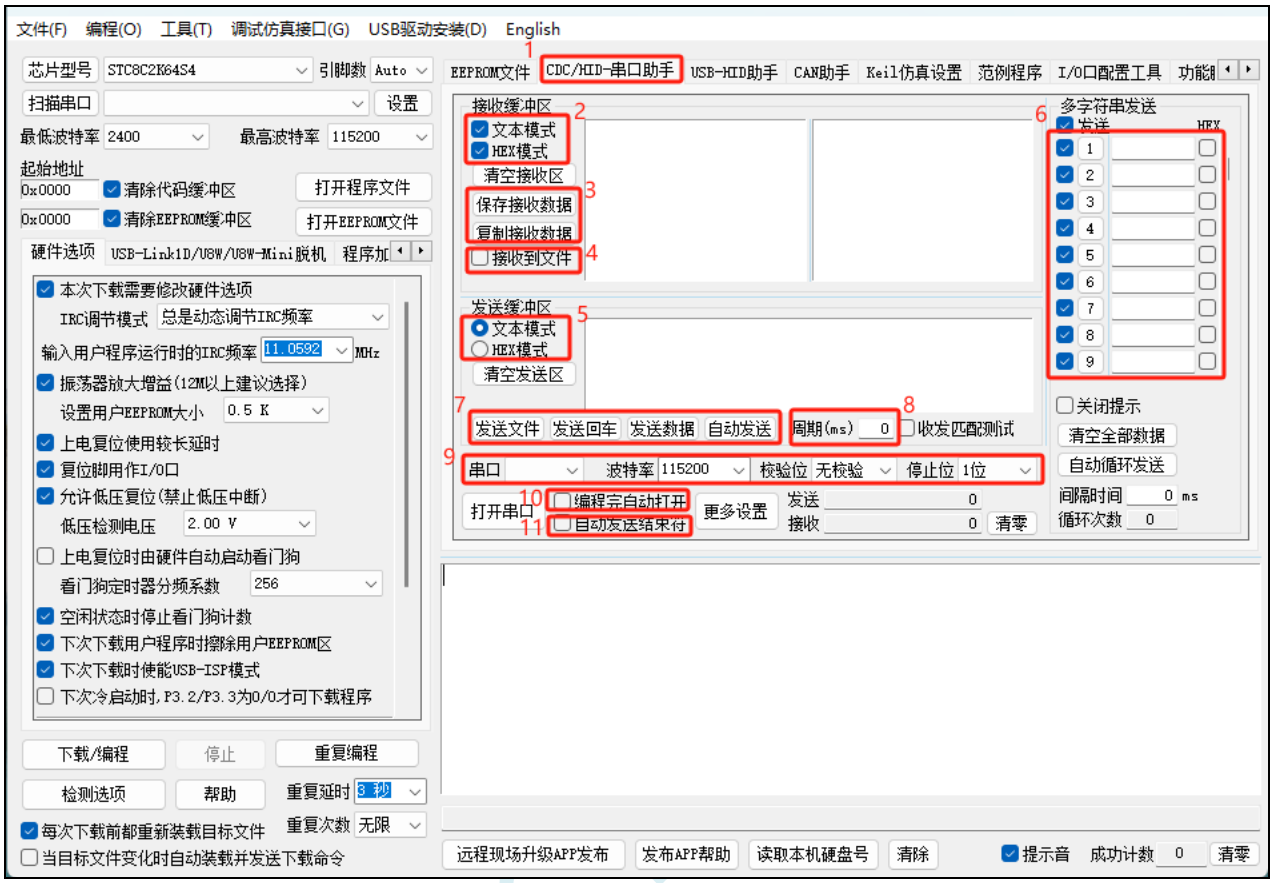
③：设置串口波特率

④：选择目标串口，并设置串口模式、波特率发生器等参数

⑤：手动生成 C 代码或者 ASM 代码，复制范例

14.8 AIapp-ISP | 串口助手/USB-CDC

串口助手主界面



①：在下载软件中选择“USB-CDC/串口助手”功能页，进入串口助手界面

②：选择接收数据的显示格式

③：保存/复制接收的数据

④：设置接收数据自动存储到文件

⑤：选择发送数据的格式

⑥：“多字符串”控制界面

⑦：数据/文件发送按钮

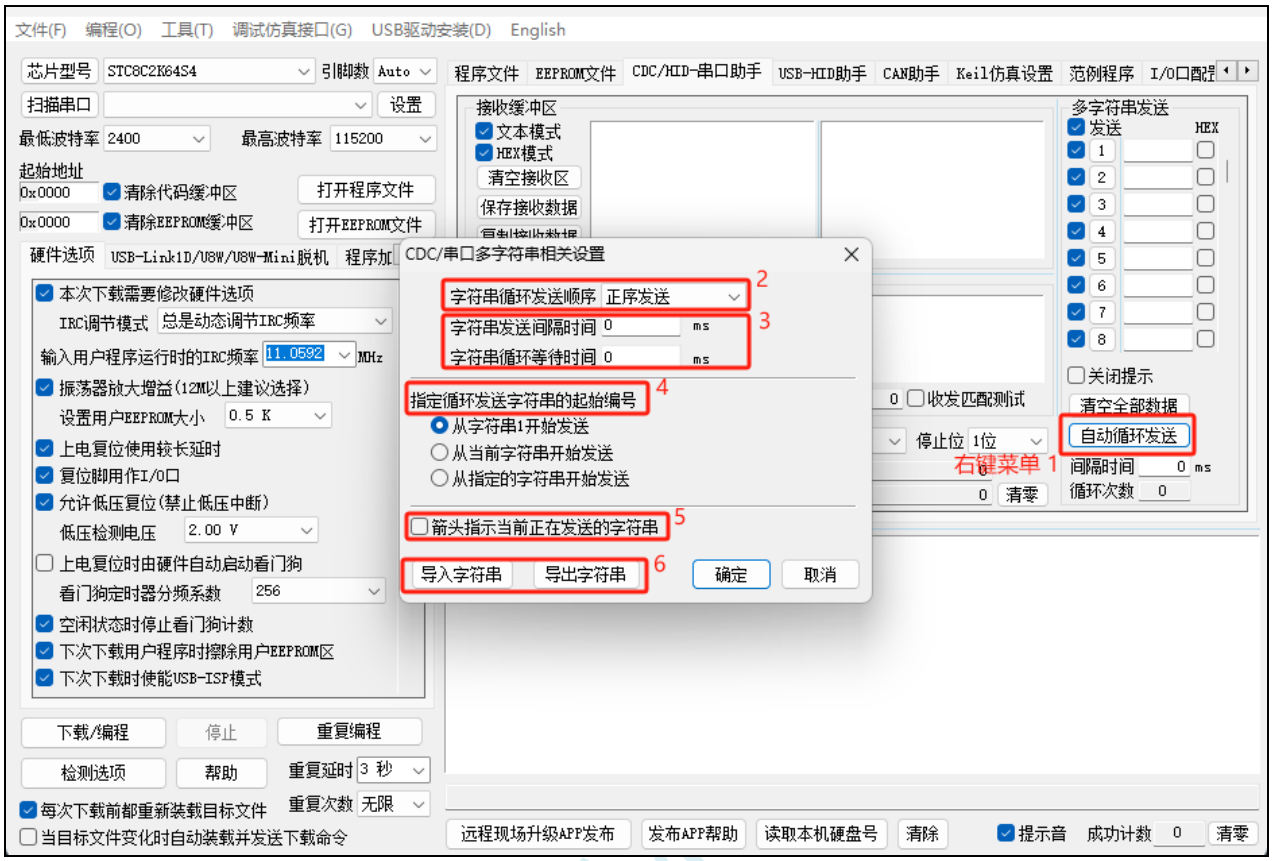
⑧：设置重复下载的周期

⑨：选择串口号，设置串口参数

⑩：设置 ISP 下载完成后是否自动打开串口

⑪：设置发送完成数据后，是否自动发送结束符

多字符串设置界面



①：鼠标右键点击“自动循环发送”按钮，点击右键菜单“设置...”进入多字符串设置界面

②：设置多字符串循环发送顺序

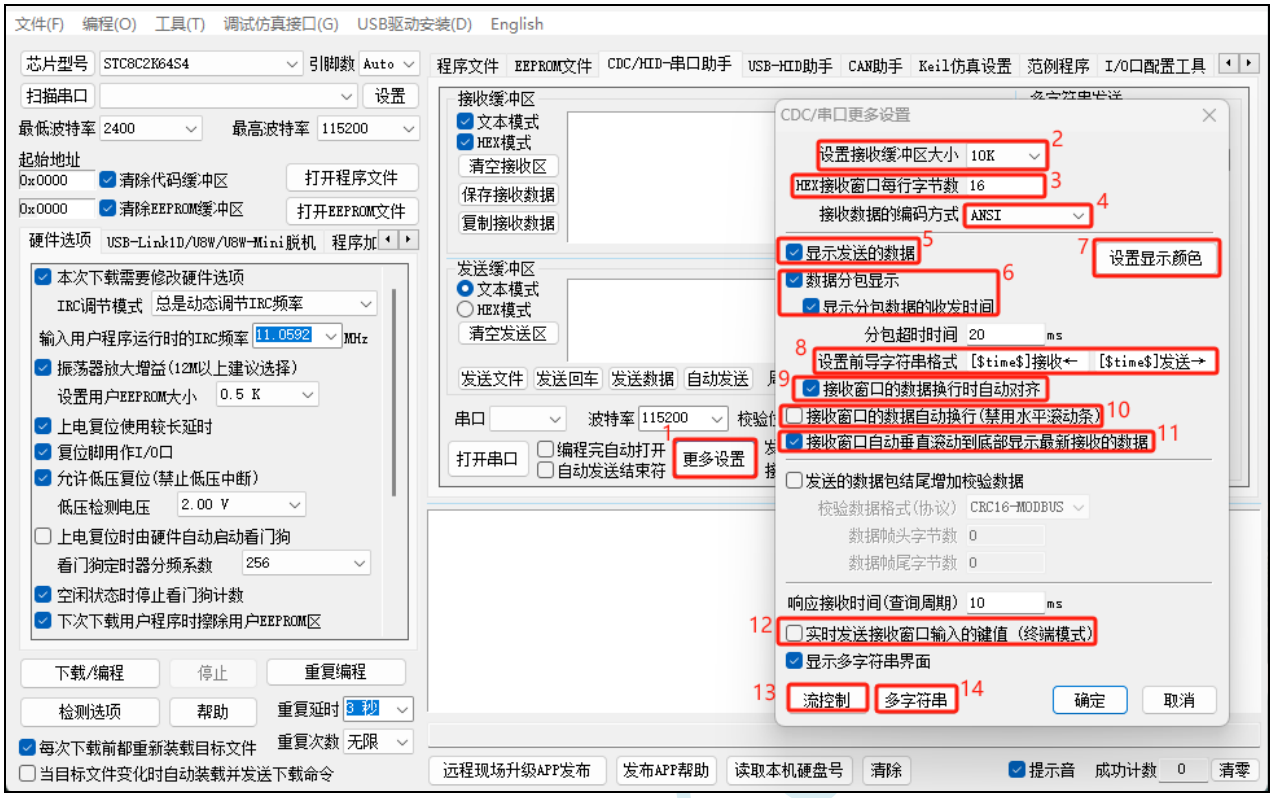
③：设置多字符串循环发送间隔时间

④：设置多字符串循环发送的起始编号

⑤：设置是否需要用箭头指示当前正在发送的字符串

⑥：多字符串导出到文件或从文件导入

串口助手更多设置



①：点击“更多设置”按钮，进入串口助手更多设置界面

②：设置接收缓存大小（缓存越大，软件反应越慢）

③：设置 HEX 接收数据每行显示的数据个数

④：设置接收数据的汉字编码格式

支持如下汉字编码格式：

ANSI：GB2312 汉字编码

UTF8：UNICODE 互联网常用编码

UTF16-LE：小端 UTF16 编码

UTF16=BE：大端 UTF16 编码

⑤：设置是否显示发送的数据

⑥：设置数据是否自动分包显示

⑦：设置界面的显示颜色

⑧：设置接收窗口数据显示的前导字符

⑨：设置接收数据的换行显示模式

⑩：设置接收窗口是否自动换行

⑪：设置发送数据是否自动追加校验数据

支持如下校验格式：

ADD8：字节校验和

ADD8N：字节校验和补码

ADD16-LE：小端双字节校验和

ADD16-BE：大端双字节校验和

XOR8：字节异或

CRC16-MODBUS：MODBUS 协议的 CRC16

CRC16-USB：USB 协议的 CRC16

CRC16-XMODEM：XMODEM 协议的 CRC16

CRC32：32 位 CRC 校验

⑫：设置是否使能终端模式（终端模式：将光标定位到接收窗口，按下键盘按键实时发送相应的键码）
⑬：进入流控制设置界面
⑭：进入多字符串界面设置界面

STC MCU

14.9 范例程序

14.9.1 串口 1 使用定时器 2 做波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
```

```
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
    while (busy);
    busy = 1;
    SBUF = dat;
}

void UartSendStr(char *p)
{
    while (*p)
    {
        UartSend(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR    DATA    8EH
T2H     DATA    0D6H
T2L     DATA    0D7H

BUSY    BIT     20H.0
```

```
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#15H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART_SEND:
            JB      BUSY,$
            SETB    BUSY
```

STC MCU

```
                MOV     SBUF,A
                RET

UART_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SENDEND
                LCALL   UART_SEND
                INC     DPTR
                JMP     UART_SENDSTR
SENDEND:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

14.9.2 串口 1 使用定时器 1（模式 0）做波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void UartIsr() interrupt 4
{
    if (TI)
    {
        TI = 0;
        busy = 0;
    }
    if (RI)
    {
        RI = 0;
        buffer[wptr++] = SBUF;
        wptr &= 0x0f;
    }
}

void UartInit()
{
    SCON = 0x50;
    TMOD = 0x00;
    TL1 = BRT;
    TH1 = BRT >> 8;
    TR1 = 1;
    AUXR = 0x40;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void UartSend(char dat)
{
```

STC MCU

```
        while (busy);
        busy = 1;
        SBUF = dat;
}

void UartSendStr(char *p)
{
        while (*p)
        {
                UartSend(*p++);
        }
}

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        UartInit();
        ES = 1;
        EA = 1;
        UartSendStr("Uart Test !\r\n");

        while (1)
        {
                if (rptr != wptr)
                {
                        UartSend(buffer[rptr++]);
                        rptr &= 0x0f;
                }
        }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA        8EH

BUSY        BIT         20H.0
WPTR        DATA        21H
RPTR        DATA        22H
BUFFER      DATA        23H                 ;16 bytes

P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
P1M0        DATA        092H
P2M1        DATA        095H
```

```
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0023H
            LJMP    UART_ISR

            ORG     0100H

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            JNB     TI,CHKRI
            CLR     TI
            CLR     BUSY
CHKRI:
            JNB     RI,UARTISR_EXIT
            CLR     RI
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,SBUF
            INC     WPTR
UARTISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART_INIT:
            MOV     SCON,#50H
            MOV     TMOD,#00H
            MOV     TL1,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     TH1,#0FFH
            SETB    TR1
            MOV     AUXR,#40H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     SBUF,A
            RET

UART_SENDSTR:
            CLR     A
            MOVC    A,@A+DPTR
            JZ      SENDEND
```

STC MCU

```
                LCALL   UART_SEND
                INC     DPTR
                JMP     UART_SENDSTR
SENDEND:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

14.9.3 串口 1 使用定时器 1（模式 2）做波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define     FOSC            11059200UL
```

```
#define	BRT		(256 - FOSC / 115200 / 32)

sfr	AUXR		=	0x8e;

sfr	P0M1		=	0x93;
sfr	P0M0		=	0x94;
sfr	P1M1		=	0x91;
sfr	P1M0		=	0x92;
sfr	P2M1		=	0x95;
sfr	P2M0		=	0x96;
sfr	P3M1		=	0xb1;
sfr	P3M0		=	0xb2;
sfr	P4M1		=	0xb3;
sfr	P4M0		=	0xb4;
sfr	P5M1		=	0xc9;
sfr	P5M0		=	0xca;

bit	busy;
char	wptr;
char	rptr;
char	buffer[16];

void UartIsr() interrupt 4
{
	if (TI)
	{
		TI = 0;
		busy = 0;
	}
	if (RI)
	{
		RI = 0;
		buffer[wptr++] = SBUF;
		wptr &= 0x0f;
	}
}

void UartInit()
{
	SCON = 0x50;
	TMOD = 0x20;
	TL1 = BRT;
	TH1 = BRT;
	TR1 = 1;
	AUXR = 0x40;
	wptr = 0x00;
	rptr = 0x00;
	busy = 0;
}

void UartSend(char dat)
{
	while (busy);
	busy = 1;
	SBUF = dat;
}

void UartSendStr(char *p)
{
```

STC MCU

```
    while (*p)
    {
        UartSend(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    ES = 1;
    EA = 1;
    UartSendStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            UartSend(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA        8EH

BUSY        BIT         20H.0
WPTR        DATA        21H
RPTR        DATA        22H
BUFFER      DATA        23H                 ;16 bytes

P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
P1M0        DATA        092H
P2M1        DATA        095H
P2M0        DATA        096H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P4M1        DATA        0B3H
P4M0        DATA        0B4H
P5M1        DATA        0C9H
P5M0        DATA        0CAH
```

```
                ORG     0000H
                LJMP    MAIN
                ORG     0023H
                LJMP    UART_ISR

                ORG     0100H

UART_ISR:
                PUSH    ACC
                PUSH    PSW
                MOV     PSW,#08H

                JNB     TI,CHKRI
                CLR     TI
                CLR     BUSY
CHKRI:
                JNB     RI,UARTISR_EXIT
                CLR     RI
                MOV     A,WPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     @R0,SBUF
                INC     WPTR
UARTISR_EXIT:
                POP     PSW
                POP     ACC
                RETI

UART_INIT:
                MOV     SCON,#50H
                MOV     TMOD,#20H
                MOV     TL1,#0FDH       ;256-11059200/115200/32=0FDH
                MOV     TH1,#0FDH
                SETB    TR1
                MOV     AUXR,#40H
                CLR     BUSY
                MOV     WPTR,#00H
                MOV     RPTR,#00H
                RET

UART_SEND:
                JB      BUSY,$
                SETB    BUSY
                MOV     SBUF,A
                RET

UART_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SENDEND
                LCALL   UART_SEND
                INC     DPTR
                JMP     UART_SENDSTR
SENDEND:
                RET

MAIN:
```

STC MCU

```
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART_INIT
                SETB    ES
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

14.9.4 串口 2 使用定时器 2 做波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;
sfr       S2CON       =   0x9a;
sfr       S2BUF       =   0x9b;
```

```
sfr     IE2     =   0xaf;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void Uart2Isr() interrupt 8
{
    if (S2CON & 0x02)
    {
        S2CON &= ~0x02;
        busy = 0;
    }
    if (S2CON & 0x01)
    {
        S2CON &= ~0x01;
        buffer[wptr++] = S2BUF;
        wptr &= 0x0f;
    }
}

void Uart2Init()
{
    S2CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart2Send(char dat)
{
    while (busy);
    busy = 1;
    S2BUF = dat;
}

void Uart2SendStr(char *p)
{
    while (*p)
    {
        Uart2SEND(*p++);
    }
```

STC MCU

```
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart2Init();
    IE2 = 0x01;
    EA = 1;
    Uart2SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart2SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S2CON       DATA    9AH
S2BUF       DATA    9BH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
```

```
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0043H
            LJMP    UART2_ISR

            ORG     0100H

UART2_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S2CON
            JNB     ACC.1,CHKRI
            ANL     S2CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART2ISR_EXIT
            ANL     S2CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S2BUF
            INC     WPTR
UART2ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART2_INIT:
            MOV     S2CON,#10H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART2_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     S2BUF,A
            RET

UART2_SENDSTR:
            CLR     A
            MOVC    A,@A+DPTR
            JZ      SEND2END
            LCALL   UART2_SEND
            INC     DPTR
            JMP     UART2_SENDSTR
SEND2END:
            RET

MAIN:
```

STC MCU

```
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART2_INIT
                MOV     IE2,#01H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART2_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART2_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

14.9.5 串口 3 使用定时器 2 做波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       AUXR        =   0x8e;
sfr       T2H         =   0xd6;
sfr       T2L         =   0xd7;
sfr       S3CON       =   0xac;
sfr       S3BUF       =   0xad;
```

```
sfr     IE2     =   0xaf;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x10;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x14;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
    while (*p)
    {
        Uart3SEND(*p++);
    }
```

STC MCU

```
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
```

```
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     008BH
            LJMP    UART3_ISR

            ORG     0100H

UART3_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S3CON
            JNB     ACC.1,CHKRI
            ANL     S3CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART3ISR_EXIT
            ANL     S3CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S3BUF
            INC     WPTR
UART3ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART3_INIT:
            MOV     S3CON,#10H
            MOV     T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T2H,#0FFH
            MOV     AUXR,#14H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART3_SEND:
            JB      BUSY,$
            SETB    BUSY
            MOV     S3BUF,A
            RET

UART3_SENDSTR:
            CLR     A
            MOVC    A,@A+DPTR
            JZ      SEND3END
            LCALL   UART3_SEND
            INC     DPTR
            JMP     UART3_SENDSTR
SEND3END:
            RET

MAIN:
```

```
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART3_INIT
                MOV     IE2,#08H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART3_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART3_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

14.9.6 串口 3 使用定时器 3 做波特率发生器

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       T4T3M   =   0xd1;
sfr       T4L     =   0xd3;
sfr       T4H     =   0xd2;
sfr       T3L     =   0xd5;
sfr       T3H     =   0xd4;
```

```
sfr     T2L     =   0xd7;
sfr     T2H     =   0xd6;
sfr     S3CON   =   0xac;
sfr     S3BUF   =   0xad;
sfr     IE2     =   0xaf;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

bit     busy;
char    wptr;
char    rptr;
char    buffer[16];

void Uart3Isr() interrupt 17
{
    if (S3CON & 0x02)
    {
        S3CON &= ~0x02;
        busy = 0;
    }
    if (S3CON & 0x01)
    {
        S3CON &= ~0x01;
        buffer[wptr++] = S3BUF;
        wptr &= 0x0f;
    }
}

void Uart3Init()
{
    S3CON = 0x50;
    T3L = BRT;
    T3H = BRT >> 8;
    T4T3M = 0x0a;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart3Send(char dat)
{
    while (busy);
    busy = 1;
    S3BUF = dat;
}

void Uart3SendStr(char *p)
{
```

STC MCU

```
    while (*p)
    {
        Uart3SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart3Init();
    IE2 = 0x08;
    EA = 1;
    Uart3SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart3SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M       DATA    0D1H
T4L         DATA    0D3H
T4H         DATA    0D2H
T3L         DATA    0D5H
T3H         DATA    0D4H
T2L         DATA    0D7H
T2H         DATA    0D6H
S3CON       DATA    0ACH
S3BUF       DATA    0ADH
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
```

```
P1M0            DATA        092H
P2M1            DATA        095H
P2M0            DATA        096H
P3M1            DATA        0B1H
P3M0            DATA        0B2H
P4M1            DATA        0B3H
P4M0            DATA        0B4H
P5M1            DATA        0C9H
P5M0            DATA        0CAH

                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
```

STC MCU

```
                JZ          SEND3END
                LCALL       UART3_SEND
                INC         DPTR
                JMP         UART3_SENDSTR
SEND3END:
                RET

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART3_INIT
                MOV         IE2,#08H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART3_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART3_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

14.9.7 串口 4 使用定时器 2 做波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"
```

```
#define	FOSC		11059200UL
#define	BRT		(65536 - FOSC / 115200 / 4)

sfr	AUXR	=	0x8e;
sfr	T2H	=	0xd6;
sfr	T2L	=	0xd7;
sfr	S4CON	=	0x84;
sfr	S4BUF	=	0x85;
sfr	IE2	=	0xaf;

sfr	P0M1	=	0x93;
sfr	P0M0	=	0x94;
sfr	P1M1	=	0x91;
sfr	P1M0	=	0x92;
sfr	P2M1	=	0x95;
sfr	P2M0	=	0x96;
sfr	P3M1	=	0xb1;
sfr	P3M0	=	0xb2;
sfr	P4M1	=	0xb3;
sfr	P4M0	=	0xb4;
sfr	P5M1	=	0xc9;
sfr	P5M0	=	0xca;

bit	busy;
char	wptr;
char	rptr;
char	buffer[16];

void Uart4Isr() interrupt 18
{
	if (S4CON & 0x02)
	{
		S4CON &= ~0x02;
		busy = 0;
	}
	if (S4CON & 0x01)
	{
		S4CON &= ~0x01;
		buffer[wptr++] = S4BUF;
		wptr &= 0x0f;
	}
}

void Uart4Init()
{
	S4CON = 0x10;
	T2L = BRT;
	T2H = BRT >> 8;
	AUXR = 0x14;
	wptr = 0x00;
	rptr = 0x00;
	busy = 0;
}

void Uart4Send(char dat)
{
	while (busy);
	busy = 1;
	S4BUF = dat;
```

STC MCU

```
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

AUXR        DATA    8EH
T2H         DATA    0D6H
T2L         DATA    0D7H
S4CON       DATA    84H
S4BUF       DATA    85H
IE2         DATA    0AFH

BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
```

```
P1M0            DATA        092H
P2M1            DATA        095H
P2M0            DATA        096H
P3M1            DATA        0B1H
P3M0            DATA        0B2H
P4M1            DATA        0B3H
P4M0            DATA        0B4H
P5M1            DATA        0C9H
P5M0            DATA        0CAH

                ORG         0000H
                LJMP        MAIN
                ORG         0093H
                LJMP        UART4_ISR

                ORG         0100H

UART4_ISR:
                PUSH        ACC
                PUSH        PSW
                MOV         PSW,#08H

                MOV         A,S4CON
                JNB         ACC.1,CHKRI
                ANL         S4CON,#NOT 02H
                CLR         BUSY
CHKRI:
                JNB         ACC.0,UART4ISR_EXIT
                ANL         S4CON,#NOT 01H
                MOV         A,WPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         @R0,S4BUF
                INC         WPTR
UART4ISR_EXIT:
                POP         PSW
                POP         ACC
                RETI

UART4_INIT:
                MOV         S4CON,#10H
                MOV         T2L,#0E8H           ;65536-11059200/115200/4=0FFE8H
                MOV         T2H,#0FFH
                MOV         AUXR,#14H
                CLR         BUSY
                MOV         WPTR,#00H
                MOV         RPTR,#00H
                RET

UART4_SEND:
                JB          BUSY,$
                SETB        BUSY
                MOV         S4BUF,A
                RET

UART4_SENDSTR:
                CLR         A
                MOVC        A,@A+DPTR
```

STC MCU

```
                JZ          SEND4END
                LCALL       UART4_SEND
                INC         DPTR
                JMP         UART4_SENDSTR
SEND4END:
                RET

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                LCALL       UART4_INIT
                MOV         IE2,#10H
                SETB        EA

                MOV         DPTR,#STRING
                LCALL       UART4_SENDSTR

LOOP:
                MOV         A,RPTR
                XRL         A,WPTR
                ANL         A,#0FH
                JZ          LOOP
                MOV         A,RPTR
                ANL         A,#0FH
                ADD         A,#BUFFER
                MOV         R0,A
                MOV         A,@R0
                LCALL       UART4_SEND
                INC         RPTR
                JMP         LOOP

STRING:         DB          'Uart Test !',0DH,0AH,00H

                END
```

14.9.8 串口 4 使用定时器 4 做波特率发生器

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"
```

```
#define   FOSC        11059200UL
#define   BRT         (65536 - FOSC / 115200 / 4)

sfr       T4T3M       =   0xd1;
sfr       T4L         =   0xd3;
sfr       T4H         =   0xd2;
sfr       T3L         =   0xd5;
sfr       T3H         =   0xd4;
sfr       T2L         =   0xd7;
sfr       T2H         =   0xd6;
sfr       S4CON       =   0x84;
sfr       S4BUF       =   0x85;
sfr       IE2         =   0xaf;

sfr       P0M1        =   0x93;
sfr       P0M0        =   0x94;
sfr       P1M1        =   0x91;
sfr       P1M0        =   0x92;
sfr       P2M1        =   0x95;
sfr       P2M0        =   0x96;
sfr       P3M1        =   0xb1;
sfr       P3M0        =   0xb2;
sfr       P4M1        =   0xb3;
sfr       P4M0        =   0xb4;
sfr       P5M1        =   0xc9;
sfr       P5M0        =   0xca;

bit       busy;
char      wptr;
char      rptr;
char      buffer[16];

void Uart4Isr() interrupt 18
{
    if (S4CON & 0x02)
    {
        S4CON &= ~0x02;
        busy = 0;
    }
    if (S4CON & 0x01)
    {
        S4CON &= ~0x01;
        buffer[wptr++] = S4BUF;
        wptr &= 0x0f;
    }
}

void Uart4Init()
{
    S4CON = 0x50;
    T4L = BRT;
    T4H = BRT >> 8;
    T4T3M = 0xa0;
    wptr = 0x00;
    rptr = 0x00;
    busy = 0;
}

void Uart4Send(char dat)
```

STC MCU

```
{
    while (busy);
    busy = 1;
    S4BUF = dat;
}

void Uart4SendStr(char *p)
{
    while (*p)
    {
        Uart4SEND(*p++);
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    Uart4Init();
    IE2 = 0x10;
    EA = 1;
    Uart4SENDStr("Uart Test !\r\n");

    while (1)
    {
        if (rptr != wptr)
        {
            Uart4SEND(buffer[rptr++]);
            rptr &= 0x0f;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

T4T3M   DATA    0D1H
T4L     DATA    0D3H
T4H     DATA    0D2H
T3L     DATA    0D5H
T3H     DATA    0D4H
T2L     DATA    0D7H
T2H     DATA    0D6H
S4CON   DATA    84H
S4BUF   DATA    85H
IE2     DATA    0AFH
```

```
BUSY        BIT     20H.0
WPTR        DATA    21H
RPTR        DATA    22H
BUFFER      DATA    23H                 ;16 bytes

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     0093H
            LJMP    UART4_ISR

            ORG     0100H

UART4_ISR:
            PUSH    ACC
            PUSH    PSW
            MOV     PSW,#08H

            MOV     A,S4CON
            JNB     ACC.1,CHKRI
            ANL     S4CON,#NOT 02H
            CLR     BUSY
CHKRI:
            JNB     ACC.0,UART4ISR_EXIT
            ANL     S4CON,#NOT 01H
            MOV     A,WPTR
            ANL     A,#0FH
            ADD     A,#BUFFER
            MOV     R0,A
            MOV     @R0,S4BUF
            INC     WPTR
UART4ISR_EXIT:
            POP     PSW
            POP     ACC
            RETI

UART4_INIT:
            MOV     S4CON,#50H
            MOV     T4L,#0E8H           ;65536-11059200/115200/4=0FFE8H
            MOV     T4H,#0FFH
            MOV     T4T3M,#0A0H
            CLR     BUSY
            MOV     WPTR,#00H
            MOV     RPTR,#00H
            RET

UART4_SEND:
```

```
                JB      BUSY,$
                SETB    BUSY
                MOV     S4BUF,A
                RET

UART4_SENDSTR:
                CLR     A
                MOVC    A,@A+DPTR
                JZ      SEND4END
                LCALL   UART4_SEND
                INC     DPTR
                JMP     UART4_SENDSTR
SEND4END:
                RET

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                LCALL   UART4_INIT
                MOV     IE2,#10H
                SETB    EA

                MOV     DPTR,#STRING
                LCALL   UART4_SENDSTR

LOOP:
                MOV     A,RPTR
                XRL     A,WPTR
                ANL     A,#0FH
                JZ      LOOP
                MOV     A,RPTR
                ANL     A,#0FH
                ADD     A,#BUFFER
                MOV     R0,A
                MOV     A,@R0
                LCALL   UART4_SEND
                INC     RPTR
                JMP     LOOP

STRING:         DB      'Uart Test !',0DH,0AH,00H

                END
```

14.9.9 串口多机通讯

现参考 STC15 系列数据手册，后续补充

STC MCU

14.9.10 串口转 LIN 总线

C 语言代码

```
//测试工作频率为 22.1184MHz

/*************  功能说明  **************
本例程基于 STC8H8K64U 为主控芯片的实验箱 8 进行编写测试，STC8G、STC8H 系列芯片可通用参考.
通过 UART 接口连接 LIN 收发器实现 LIN 总线信号收发测试例程.
UART1 通过串口工具连接电脑.
UART2 外接 LIN 收发器(TJA1020/1)，连接 LIN 总线.
将电脑串口发送的数据转发到 LIN 总线; 从 LIN 总线接收到的数据转发到电脑串口.
默认传输速率：9600 波特率，发送 LIN 数据前切换波特率，发送 13 个显性间隔信号.
下载时，选择时钟 22.1184MHz (用户可自行修改频率).
*****************************************/

#include "reg51.h"
#include "intrins.h"

#define    MAIN_Fosc        22118400L

typedef    unsigned char    u8;
typedef    unsigned int     u16;
typedef    unsigned long    u32;

sfr        AUXR     =   0x8E;
sfr        S2CON    =   0x9A;
sfr        S2BUF    =   0x9B;
sfr        TH2      =   0xD6;
sfr        TL2      =   0xD7;
sfr        IE2      =   0xAF;
sfr        INTCLKO  =   0x8F;
sfr        P_SW1    =   0xA2;
sfr        P_SW2    =   0xBA;

sfr        P4       =   0xC0;
sfr        P5       =   0xC8;
sfr        P6       =   0xE8;
sfr        P7       =   0xF8;
sfr        P1M1     =   0x91;
sfr        P1M0     =   0x92;
sfr        P0M1     =   0x93;
sfr        P0M0     =   0x94;
sfr        P2M1     =   0x95;
sfr        P2M0     =   0x96;
sfr        P3M1     =   0xB1;
sfr        P3M0     =   0xB2;
sfr        P4M1     =   0xB3;
sfr        P4M0     =   0xB4;
sfr        P5M1     =   0xC9;
sfr        P5M0     =   0xCA;
sfr        P6M1     =   0xCB;
sfr        P6M0     =   0xCC;
sfr        P7M1     =   0xE1;
sfr        P7M0     =   0xE2;

sbit       P00      =   P0^0;
sbit       P01      =   P0^1;
```

```
sbit    P02     =   P0^2;
sbit    P03     =   P0^3;
sbit    P04     =   P0^4;
sbit    P05     =   P0^5;
sbit    P06     =   P0^6;
sbit    P07     =   P0^7;
sbit    P10     =   P1^0;
sbit    P11     =   P1^1;
sbit    P12     =   P1^2;
sbit    P13     =   P1^3;
sbit    P14     =   P1^4;
sbit    P15     =   P1^5;
sbit    P16     =   P1^6;
sbit    P17     =   P1^7;
sbit    P20     =   P2^0;
sbit    P21     =   P2^1;
sbit    P22     =   P2^2;
sbit    P23     =   P2^3;
sbit    P24     =   P2^4;
sbit    P25     =   P2^5;
sbit    P26     =   P2^6;
sbit    P27     =   P2^7;
sbit    P30     =   P3^0;
sbit    P31     =   P3^1;
sbit    P32     =   P3^2;
sbit    P33     =   P3^3;
sbit    P34     =   P3^4;
sbit    P35     =   P3^5;
sbit    P36     =   P3^6;
sbit    P37     =   P3^7;
sbit    P40     =   P4^0;
sbit    P41     =   P4^1;
sbit    P42     =   P4^2;
sbit    P43     =   P4^3;
sbit    P44     =   P4^4;
sbit    P45     =   P4^5;
sbit    P46     =   P4^6;
sbit    P47     =   P4^7;
sbit    P50     =   P5^0;
sbit    P51     =   P5^1;
sbit    P52     =   P5^2;
sbit    P53     =   P5^3;
sbit    P54     =   P5^4;
sbit    P55     =   P5^5;
sbit    P56     =   P5^6;
sbit    P57     =   P5^7;

sbit    SLP_N   =   P2^4;                   //0: Sleep

/****************************** 用户定义宏 ***********************************/

#define Baudrate1           (65536UL - (MAIN_Fosc /  4) / 9600UL)
#define Baudrate2           (65536UL - (MAIN_Fosc /  4) / 9600UL)

#define Baudrate_Break      (65536UL - (MAIN_Fosc /  4) / 6647UL)    //发送显性间隔信号波特率

#define UART1_BUF_LENGTH    32
#define UART2_BUF_LENGTH    32
```

STC MCU

```
#define     LIN_ID          0x31

u8 TX1_Cnt;                                         //发送计数
u8 RX1_Cnt;                                         //接收计数
u8 TX2_Cnt;                                         //发送计数
u8 RX2_Cnt;                                         //接收计数
bit B_TX1_Busy;                                     //发送忙标志
bit B_TX2_Busy;                                     //发送忙标志
u8 RX1_TimeOut;
u8 RX2_TimeOut;

u8 xdata RX1_Buffer[UART1_BUF_LENGTH];              //接收缓冲
u8 xdata RX2_Buffer[UART2_BUF_LENGTH];              //接收缓冲

void UART1_config(u8 brt);
void UART2_config(u8 brt);
void PrintString1(u8 *puts);
void delay_ms(u8 ms);
void UART1_TxByte(u8 dat);
void UART2_TxByte(u8 dat);
void Lin_Send(u8 *puts);
void SetTimer2Baudraye(u16 dat);

//========================================================================
// 函数: void main(void)
// 描述: 主函数。
// 参数: none.
// 返回: none.
// 版本: VER1.0
//========================================================================
void main(void)
{
    u8 i;

    P0M1 = 0; P0M0 = 0;                             //设置为准双向口
    P1M1 = 0; P1M0 = 0;                             //设置为准双向口
    P2M1 = 0; P2M0 = 0;                             //设置为准双向口
    P3M1 = 0; P3M0 = 0;                             //设置为准双向口
    P4M1 = 0; P4M0 = 0;                             //设置为准双向口
    P5M1 = 0; P5M0 = 0;                             //设置为准双向口
    P6M1 = 0; P6M0 = 0;                             //设置为准双向口
    P7M1 = 0; P7M0 = 0;                             //设置为准双向口

    UART1_config(1);
    UART2_config(2);
    EA = 1;                                         //允许全局中断
    SLP_N =  1;

    PrintString1("STC8H8K64U UART1 Test Programme!\r\n"); //UART1 发送一个字符串

    while (1)
    {
        delay_ms(1);
        if(RX1_TimeOut > 0)
        {
            if(--RX1_TimeOut == 0)                  //超时,则串口接收结束
            {
                if(RX1_Cnt > 0)
                {
```

```
                Lin_Send(RX1_Buffer);                   //将 UART1 收到的数据发送到 LIN 总线上
            }
            RX1_Cnt = 0;
        }
    }

    if(RX2_TimeOut > 0)
    {
        if(--RX2_TimeOut == 0)                          //超时,则串口接收结束
        {
            if(RX2_Cnt > 0)
            {
                for (i=0; I < RX2_Cnt; i++)             //遇到停止符 0 结束
                {
                    UART1_TxByte(RX2_Buffer[i]); //从 LIN 总线收到的数据发送到 UART1
                }
            }
            RX2_Cnt = 0;
        }
    }
}
}

//=========================================================================
// 函数: void delay_ms(unsigned  char ms)
// 描述:  延时函数。
// 参数: ms,要延时的 ms 数,  这里只支持 1~255ms.  自动适应主时钟.
// 返回: none.
//=========================================================================
void delay_ms(u8 ms)
{
    u16 i;
    do{
        i = MAIN_Fosc / 13000;
        while(--i);                                     //14T per loop
    }while(--ms);
}

//=========================================================================
// 函数: u8                Lin_CheckPID(u8      id)
// 描述: ID 码加上校验符，转成 PID 码。
// 参数: ID 码.
// 返回: PID 码.
//=========================================================================
u8 Lin_CheckPID(u8 id)
{
    u8 returnpid ;
    u8 P0 ;
    u8 P1 ;

    P0 = (((id)^(id>>1)^(id>>2)^(id>>4))&0x01)<<6;
    P1 = ((~((id>>1)^(id>>3)^(id>>4)^(id>>5)))&0x01)<<7;

    returnpid = id|P0|P1 ;

    return returnpid ;
}

//=========================================================================
```

```
// 函数: u8 LINCalcChecksum(u8 *dat)
// 描述: 计算校验码。
// 参数: 数据场传输的数据.
// 返回: 校验码.
//==========================================================================
static u8 LINCalcChecksum(u8 *dat)
{
    u16 sum = 0;
    u8 i;

    for(I = 0; i < 8; i++)
    {
    sum += dat[i];
    if(sum & 0xFF00)
    {
        sum = (sum & 0x00FF) + 1;
    }
    }
    sum ^= 0x00FF;
    return (u8)sum;
}

//==========================================================================
// 函数: void Lin_SendBreak(void)
// 描述: 发送显性间隔信号。
// 参数: none.
// 返回: none.
//==========================================================================
void Lin_SendBreak(void)
{
    SetTimer2Baudraye(Baudrate_Break);
    UART2_TxByte(0);
    SetTimer2Baudraye(Baudrate2);
}

//==========================================================================
// 函数: void Lin_Send(u8 *puts)
// 描述: 发送 LIN 总线报文。
// 参数: 待发送的数据场内容.
// 返回: none.
//==========================================================================
void Lin_Send(u8 *puts)
{
    u8 i;

    Lin_SendBreak();                                    //Break
    UART2_TxByte(0x55);                                 //SYNC
    UART2_TxByte(Lin_CheckPID(LIN_ID));                 //LIN ID
    for(i=0;i<8;i++)
    {
        UART2_TxByte(puts[i]);
    }
    UART2_TxByte(LINCalcChecksum(puts));
}

//==========================================================================
// 函数: void UART1_TxByte(u8 dat)
// 描述: 发送一个字节.
// 参数: 无.
```

```
// 返回: 无.
//========================================================================
void UART1_TxByte(u8 dat)
{
    SBUF = dat;
    B_TX1_Busy = 1;
    while(B_TX1_Busy);
}

//========================================================================
// 函数: void UART2_TxByte(u8 dat)
// 描述: 发送一个字节.
// 参数: 无.
// 返回: 无.
//========================================================================
void UART2_TxByte(u8 dat)
{
    S2BUF = dat;
    B_TX2_Busy = 1;
    while(B_TX2_Busy);
}

//========================================================================
// 函数: void PrintString1(u8 *puts)
// 描述: 串口 1 发送字符串函数。
// 参数: puts:                字符串指针.
// 返回: none.
//========================================================================
void PrintString1(u8 *puts)
{
    for    (; *puts != 0; puts++)                    //遇到停止符 0 结束
    {
        SBUF = *puts;
        B_TX1_Busy = 1;
        while(B_TX1_Busy);
    }
}

//========================================================================
// 函数: void PrintString2(u8 *puts)
// 描述: 串口 2 发送字符串函数。
// 参数: puts:                字符串指针.
// 返回: none.
//========================================================================
//void PrintString2(u8 *puts)
//{
//    for (; *puts != 0;    puts++)                    //遇到停止符 0 结束
//    {
//        S2BUF = *puts;
//        B_TX2_Busy = 1;
//        while(B_TX2_Busy);
//    }
//}

//========================================================================
// 函数: SetTimer2Baudraye(u16 dat)
// 描述: 设置 Timer2 做波特率发生器。
// 参数: dat: Timer2 的重装值.
// 返回: none.
```

STC MCU

```
//========================================================================
void SetTimer2Baudraye(u16 dat)
{
    AUXR &= ~(1<<4);                    //Timer stop
    AUXR &= ~(1<<3);                    //Timer2 set As Timer
    AUXR |= (1<<2);                     //Timer2 set as 1T mode
    TH2  = dat / 256;
    TL2 = dat % 256;
    IE2 &= ~(1<<2);                     //禁止中断
    AUXR |= (1<<4);                     //Timer run enable
}

//========================================================================
// 函数: void UART1_config(u8 brt)
// 描述: UART1 初始化函数。
// 参数: brt: 选择波特率, 2: 使用 Timer2 做波特率, 其它值: 使用 Timer1 做波特率.
// 返回: none.
//========================================================================
void UART1_config(u8 brt)
{
    /********** 波特率使用定时器 2 ****************/
    if(brt == 2)
    {
        AUXR |= 0x01;                   //S1 BRT Use Timer2;
        SetTimer2Baudraye(Baudrate1);
    }

    /********** 波特率使用定时器 1 ****************/
    else
    {
        TR1  = 0;
        AUXR &= ~0x01;                  //S1 BRT Use Timer1;
        AUXR |= (1<<6);                 //Timer1 set as   1T mode
        TMOD &= ~(1<<6);                //Timer1 set As   Timer
        TMOD &= ~0x30;                  //Timer1_16bitAutoReload;
        TH1 = (u8)(Baudrate1 / 256);
        TL1 = (u8)(Baudrate1 % 256);
        ET1 = 0;                        //禁止中断
        INTCLKO &= ~0x02;               //不输出时钟
        TR1 = 1;
    }
    /*************************************************/

    SCON = (SCON & 0x3f) | 0x40;        //UART1 模式:  0x00: 同步移位输出,
                                        //             0x40: 8 位数据,可变波特率,
                                        //             0x80: 9 位数据,固定波特率,
                                        //             0xc0: 9 位数据,可变波特率
//  PS  = 1;                            //高优先级中断
    ES  = 1;                            //允许中断
    REN = 1;                            //允许接收
    P_SW1 &= 0x3f;
//  P_SW1 |= 0x80;                      //UART1switch to:   0x00: P3.0 P3.1,
                                        //                  0x40: P3.6 P3.7,
                                        //                  0x80: P1.6 P1.7,
                                        //                  0xC0: P4.3 P4.4

    B_TX1_Busy = 0;
    TX1_Cnt = 0;
    RX1_Cnt = 0;
```

STC MCU

```
}

//========================================================================
// 函数: void UART2_config(u8 brt)
// 描述: UART2 初始化函数。
// 参数: brt: 选择波特率, 2: 使用 Timer2 做波特率, 其它值: 无效.
// 返回: none.
//========================================================================
void UART2_config(u8 brt)
{
    if(brt == 2)
    {
        SetTimer2Baudraye(Baudrate2);

        S2CON &= ~(1<<7);                   //8 位数据,1 位起始位,1 位停止位, 无校验
        IE2 |= 1;                           //允许中断
        S2CON |= (1<<4);                    //允许接收
        P_SW2 &= ~0x01;
//      P_SW2     |= 1;                     //UART2 switch to: 0: P1.0/P1.1, 1: P4.6/P4.7

        B_TX2_Busy = 0;
        TX2_Cnt = 0;
        RX2_Cnt = 0;
    }
}

//========================================================================
// 函数: void UART1_int     (void) interrupt UART1_VECTOR
// 描述: UART1 中断函数。
// 参数: nine.
// 返回: none.
//========================================================================
void UART1_int (void) interrupt   4
{
    if(RI)
    {
        RI = 0;
        if(RX1_Cnt >= UART1_BUF_LENGTH) RX1_Cnt = 0;
        RX1_Buffer[RX1_Cnt] = SBUF;
        RX1_Cnt++;
        RX1_TimeOut = 5;
    }

    if(TI)
    {
        TI = 0;
        B_TX1_Busy = 0;
    }
}

//========================================================================
// 函数: void UART2_int     (void) interrupt UART2_VECTOR
// 描述: UART2 中断函数。
// 参数: nine.
// 返回: none.
//========================================================================
void UART2_int (void) interrupt   8
{
    if((S2CON & 1) != 0)
```

```
    {
        S2CON &= ~1;                                    //Clear Rx flag
        if(RX2_Cnt >= UART2_BUF_LENGTH) RX2_Cnt = 0;
        RX2_Buffer[RX2_Cnt] = S2BUF;
        RX2_Cnt++;
        RX2_TimeOut = 5;
    }

    if((S2CON &  2) != 0)
    {
        S2CON &= ~2;                                    //Clear Tx flag
        B_TX2_Busy = 0;
    }
}
```

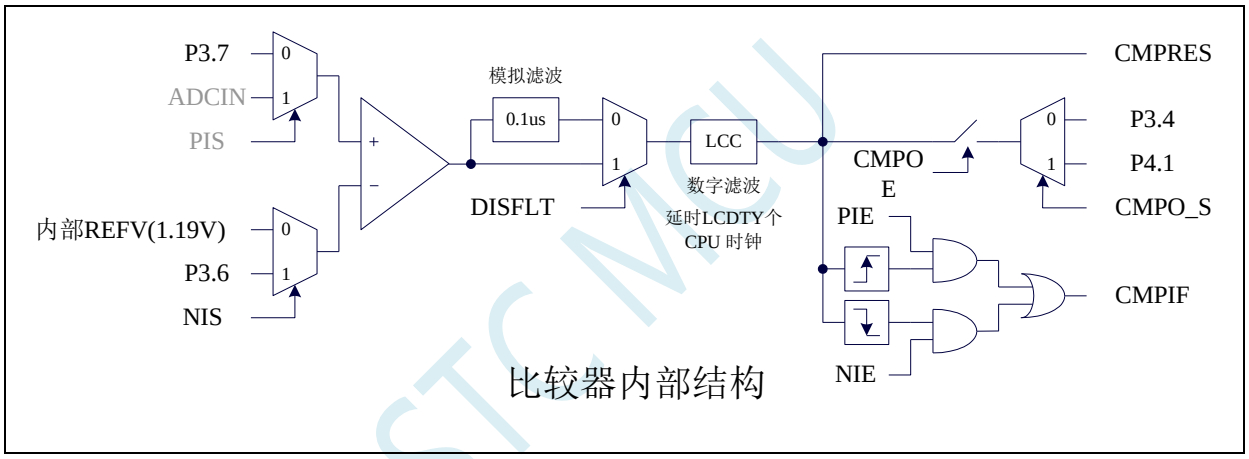
STC MCU

15 比较器，掉电检测，内部 1.19V 参考信号源（BGV）

STC8C 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，负极可以是 P3.6 端口或者是内部 BandGap 经过 OP 后的 REFV 电压（内部固定比较电压）。**通过多路选择器和分时复用可实现多个比较器的应用**

比较器内部有可程序控制的两级滤波：模拟滤波和数字滤波。模拟滤波可以过滤掉比较输入信号中的毛刺信号，数字滤波可以等待输入信号更加稳定后再进行比较。比较结果可直接通过读取内部寄存器位获得，也可将比较器结果正向或反向输出到外部端口。将比较结果输出到外部端口可用作外部事件的触发信号和反馈信号，可扩大比较的应用范围。

15.1 比较器内部结构图



15.2 比较器相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
CMPCR1	比较器控制寄存器 1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES	0000,0000
CMPCR2	比较器控制寄存器 2	E7H	INVCMPO	DISFLT	LCDTY[5:0]						0000,0000

15.2.1 比较器控制寄存器 1（CMPCR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR1	E6H	CMPEN	CMPIF	PIE	NIE	PIS	NIS	CMPOE	CMPRES

CMPEN：比较器模块使能位

0：关闭比较功能

1：使能比较功能

CMPIF：比较器中断标志位。当 PIE 或 NIE 被使能后，若产生相应的中断信号，硬件自动将 CMPIF 置 1，并向 CPU 提出中断请求。此标志位必须用户软件清零。

（注意：没有使能比较器中断时，硬件不会设置此中断标志，即使用查询方式访问比较器时，不能查询此中断标志）

PIE：比较器上升沿中断使能位。

0：禁止比较器上升沿中断。

1：使能比较器上升沿中断。使能比较器的比较结果由 0 变成 1 时产生中断请求。

NIE：比较器下降沿中断使能位。

0：禁止比较器下降沿中断。

1：使能比较器下降沿中断。使能比较器的比较结果由 1 变成 0 时产生中断请求。

NIS：比较器的负极选择位

0：选择内部 BandGap 经过 OP 后的电压 REFV 作为比较器负极输入源 （芯片在出厂时，内部参考电压调整为 **1.19V**）。

1：选择外部端口 P3.6 为比较器负极输入源。

CMPOE：比较器结果输出控制位

0：禁止比较器结果输出

1：使能比较器结果输出。比较器结果输出到 P3.4 或者 P4.1（由 P_SW2 中的 CMPO_S 进行设定）

CMPRES：比较器的比较结果。此位为只读。

0：表示 CMP+的电平低于 CMP-的电平

1：表示 CMP+的电平高于 CMP-的电平

CMPRES 是经过数字滤波后的输出信号，而不是比较器的直接输出结果。

15.2.2　比较器控制寄存器 2（CMPCR2）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
CMPCR2	E7H	INVCMPO	DISFLT	LCDTY[5:0]					

INVCMPO：比较器结果输出控制

0：比较器结果正向输出。若 CMPRES 为 0，则 P3.4/P4.1 输出低电平，反之输出高电平。

1：比较器结果反向输出。若 CMPRES 为 0，则 P3.4/P4.1 输出高电平，反之输出低电平。

DISFLT：模拟滤波功能控制

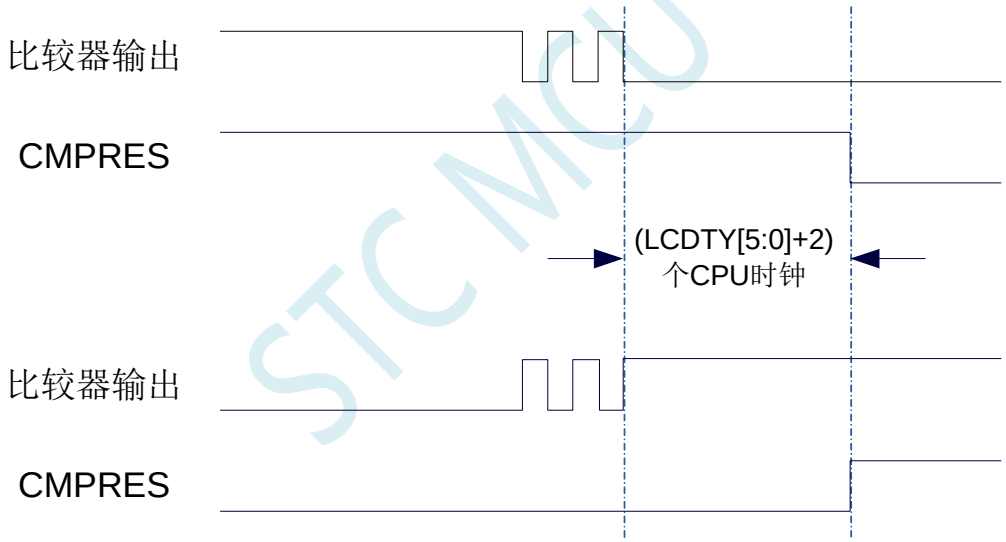
0：使能 0.1us 模拟滤波功能

1：关闭 0.1us 模拟滤波功能，可略微提高比较器的比较速度。

LCDTY[5:0]：数字滤波功能控制

数字滤波功能即为数字信号去抖动功能。当比较结果发生上升沿或者下降沿变化时，比较器侦测变化后的信号必须维持 LCDTY 所设置的 CPU 时钟数不发生变化，才认为数据变化是有效的；否则将视同信号无变化。

注意：当使能数字滤波功能后，芯片内部实际的等待时钟需额外增加两个状态机切换时间，即若 LCDTY 设置为 0 时，为关闭数字滤波功能；若 LCDTY 设置为非 0 值 n（n=1~63）时，则实际的数字滤波时间为（n+2）个系统时钟



STC MCU

15.3 范例程序

15.3.1 比较器的使用（中断方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         CMPCR1          =   0xe6;
sfr         CMPCR2          =   0xe7;

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        P10             =   P1^0;
sbit        P11             =   P1^1;

void CMP_Isr() interrupt 21
{
    CMPCR1 &= ~0x40;                        //清中断标志
    if (CMPCR1 & 0x01)
    {
        P10 = !P10;                         //上升沿中断测试端口
    }
    else
    {
        P11 = !P11;                         //下降沿中断测试端口
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
```

STC MCU

```
    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80;                    //比较器正向输出
//  CMPCR2 |= 0x80;                     //比较器反向输出
    CMPCR2 &= ~0x40;                    //使能 0.1us 滤波
//  CMPCR2 |= 0x40;                     //禁止 0.1us 滤波
//  CMPCR2 &= ~0x3f;                    //比较器结果直接输出
    CMPCR2 |= 0x10;                     //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPCR1 |= 0x30;                     //使能比较器边沿中断
//  CMPCR1 &= ~0x20;                    //禁止比较器上升沿中断
//  CMPCR1 |= 0x20;                     //使能比较器上升沿中断
//  CMPCR1 &= ~0x10;                    //禁止比较器下降沿中断
//  CMPCR1 |= 0x10;                     //使能比较器下降沿中断
    CMPCR1 &= ~0x08;                    //P3.7 为 CMP+输入脚
//  CMPCR1 |= 0x08;                     //ADC 输入脚为 CMP+输入脚
//  CMPCR1 &= ~0x04;                    //内部 1.19V 参考信号源为 CMP-输入脚
    CMPCR1 |= 0x04;                     //P3.6 为 CMP-输入脚
//  CMPCR1 &= ~0x02;                    //禁止比较器输出
    CMPCR1 |= 0x02;                     //使能比较器输出
    CMPCR1 |= 0x80;                     //使能比较器模块

    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00ABH
            LJMP    CMPISR

            ORG     0100H
CMPISR:
            PUSH    ACC
            ANL     CMPCR1,#NOT 40H     ;清中断标志
            MOV     A,CMPCR1
            JB      ACC.0,RSING
FALLING:
```

```
                CPL     P1.0                    ;下降沿中断测试端口
                POP     ACC
                RETI
RSING:
                CPL     P1.1                    ;上升沿中断测试端口
                POP     ACC
                RETI

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                MOV     CMPCR2,#00H
                ANL     CMPCR2,#NOT 80H         ;比较器正向输出
;               ORL     CMPCR2,#80H             ;比较器反向输出
                ANL     CMPCR2,#NOT 40H         ;使能 0.1us 滤波
;               ORL     CMPCR2,#40H             ;禁止 0.1us 滤波
;               ANL     CMPCR2,#NOT 3FH         ;比较器结果直接输出
                ORL     CMPCR2,#10H             ;比较器结果经过 16 个去抖时钟后输出
                MOV     CMPCR1,#00H
                ORL     CMPCR1,#30H             ;使能比较器边沿中断
;               ANL     CMPCR1,#NOT 20H         ;禁止比较器上升沿中断
;               ORL     CMPCR1,#20H             ;使能比较器上升沿中断
;               ANL     CMPCR1,#NOT 10H         ;禁止比较器下降沿中断
;               ORL     CMPCR1,#10H             ;使能比较器下降沿中断
                ANL     CMPCR1,#NOT 08H         ;P3.7 为 CMP+输入脚
;               ORL     CMPCR1,#08H             ;ADC 输入脚为 CMP+输入脚
;               ANL     CMPCR1,#NOT 04H         ;内部 1.19V 参考信号源为 CMP-输入脚
                ORL     CMPCR1,#04H             ;P3.6 为 CMP-输入脚
;               ANL     CMPCR1,#NOT 02H         ;禁止比较器输出
                ORL     CMPCR1,#02H             ;使能比较器输出
                ORL     CMPCR1,#80H             ;使能比较器模块
                SETB    EA

                JMP     $

                END
```

15.3.2 比较器的使用（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
```

```
#include "intrins.h"

sfr     CMPCR1  =   0xe6;
sfr     CMPCR2  =   0xe7;

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    P10     =   P1^0;
sbit    P11     =   P1^1;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    CMPCR2 = 0x00;
    CMPCR2 &= ~0x80;                    //比较器正向输出
//  CMPCR2 |= 0x80;                     //比较器反向输出
    CMPCR2 &= ~0x40;                    //使能 0.1us 滤波
//  CMPCR2 |= 0x40;                     //禁止 0.1us 滤波
//  CMPCR2 &= ~0x3f;                    //比较器结果直接输出
    CMPCR2 |= 0x10;                     //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPCR1 |= 0x30;                     //使能比较器边沿中断
//  CMPCR1 &= ~0x20;                    //禁止比较器上升沿中断
//  CMPCR1 |= 0x20;                     //使能比较器上升沿中断
//  CMPCR1 &= ~0x10;                    //禁止比较器下降沿中断
//  CMPCR1 |= 0x10;                     //使能比较器下降沿中断
    CMPCR1 &= ~0x08;                    //P3.7 为 CMP+输入脚
//  CMPCR1 |= 0x08;                     //ADC 输入脚为 CMP+输入脚
//  CMPCR1 &= ~0x04;                    //内部 1.19V 参考信号源为 CMP-输入脚
    CMPCR1 |= 0x04;                     //P3.6 为 CMP-输入脚
//  CMPCR1 &= ~0x02;                    //禁止比较器输出
    CMPCR1 |= 0x02;                     //使能比较器输出
    CMPCR1 |= 0x80;                     //使能比较器模块

    while (1)
    {
```

```
        P10 = CMPCR1 & 0x01;                    //读取比较器比较结果
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CMPCR1      DATA    0E6H
CMPCR2      DATA    0E7H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

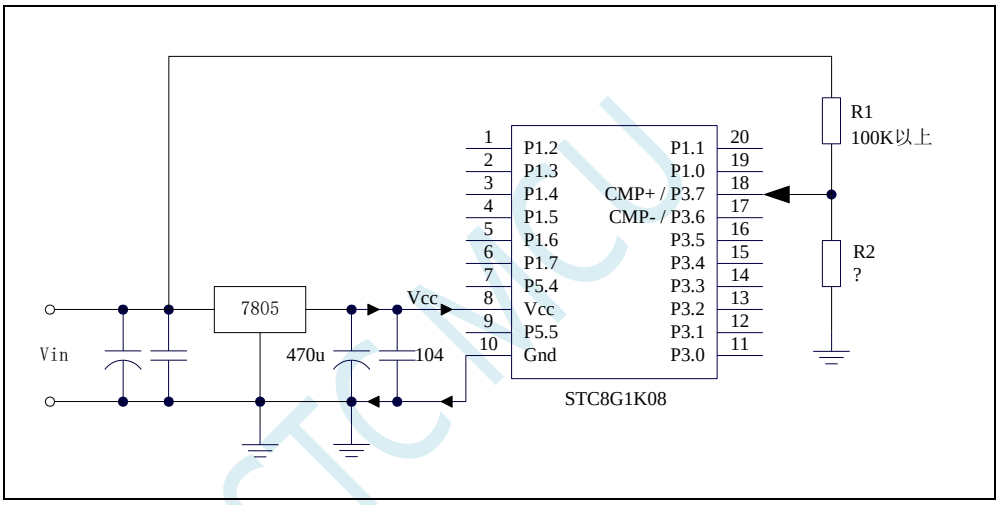
            MOV     CMPCR2,#00H
            ANL     CMPCR2,#NOT 80H     ;比较器正向输出
;           ORL     CMPCR2,#80H         ;比较器反向输出
            ANL     CMPCR2,#NOT 40H     ;使能 0.1us 滤波
;           ORL     CMPCR2,#40H         ;禁止 0.1us 滤波
;           ANL     CMPCR2,#NOT 3FH     ;比较器结果直接输出
            ORL     CMPCR2,#10H         ;比较器结果经过 16 个去抖时钟后输出
            MOV     CMPCR1,#00H
            ORL     CMPCR1,#30H         ;使能比较器边沿中断
;           ANL     CMPCR1,#NOT 20H     ;禁止比较器上升沿中断
;           ORL     CMPCR1,#20H         ;使能比较器上升沿中断
;           ANL     CMPCR1,#NOT 10H     ;禁止比较器下降沿中断
;           ORL     CMPCR1,#10H         ;使能比较器下降沿中断
            ANL     CMPCR1,#NOT 08H     ;P3.7 为 CMP+输入脚
;           ORL     CMPCR1,#08H         ;ADC 输入脚为 CMP+输入脚
;           ANL     CMPCR1,#NOT 04H     ;内部 1.19V 参考信号源为 CMP-输入脚
```

```
        ORL     CMPCR1,#04H         ;P3.6 为 CMP-输入脚
;       ANL     CMPCR1,#NOT 02H     ;禁止比较器输出
        ORL     CMPCR1,#02H         ;使能比较器输出
        ORL     CMPCR1,#80H         ;使能比较器模块

LOOP:
        MOV     A,CMPCR1
        MOV     C,ACC.0
        MOV     P1.0,C              ;读取比较器比较结果
        JMP     LOOP

        END
```

15.3.3　比较器作外部掉电检测（掉电过程中应及时保存用户数据到 EEPROM 中）

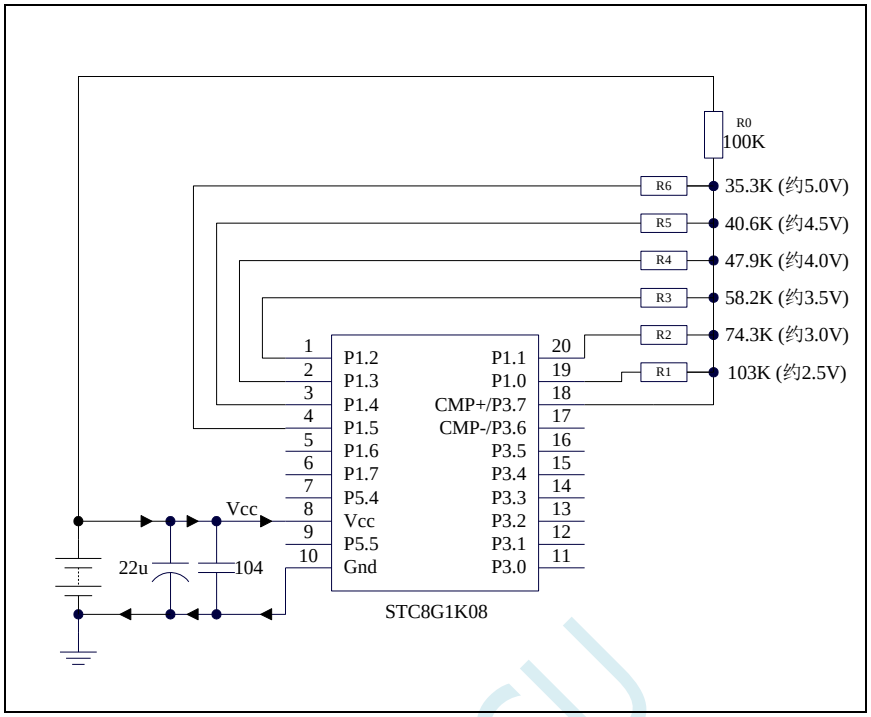


上图中电阻 R1 和 R2 对稳压块 7805 的前端电压进行分压，分压后的电压作为比较器 CMP+的外部输入与内部 1.19V 参考信号源进行比较。

一般当交流电在 220V 时，稳压块 7805 前端的直流电压为 11V，但当交流电压降到 160V 时，稳压块 7805 前端的直流电压为 8.5V。当稳压块 7805 前端的直流电压低于或等于 8.5V 时，该前端输入的直流电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+，CMP+端输入电压低于内部 1.19V 参考信号源，此时可产生比较器中断，这样在掉电检测时就有充足的时间将数据保存到 EEPROM 中。当稳压块 7805 前端的直流电压高于 8.5V 时，该前端输入的直流电压被电阻 R1 和 R2 分压到比较器正极输入端 CMP+，CMP+端输入电压高于内部 1.19V 参考信号源，此时 CPU 可继续正常工作。

内部 1.19V 参考信号源即为内部 BandGap 经过 OP 后的电压 REFV（**芯片在出厂时，内部参考电压调整为 1.19V**）。具体的数值要通过读取内部 1.19V 参考信号源在内部 RAM 区或者 Flash 程序存储器（ROM）区所占用的地址的值获得。对于 STC8 系列，内部 1.19V 参考信号源值在 RAM 和 Flash 程序存储器（ROM）中的存储地址请参考“存储器中的特殊参数”章节

15.3.4　比较器检测工作电压（电池电压）



上图中，利用电阻分压的原理可以近似的测量出 MCU 的工作电压（选通的通道，MCU 的 I/O 口输出低电平，端口电压值接近 Gnd，未选通的通道，MCU 的 I/O 口输出开漏模式的高，不影响其他通道）。

比较器的负端选择内部 1.19V 参考信号源，正端选择通过电阻分压后输入到 CMP+管脚的电压值。

初始化时 P1.5~P1.0 口均设置为开漏模式，并输出高。首先 P1.0 口输出低电平，此时若 Vcc 电压低于 2.5V 则比较器的比较值为 0，反之若 Vcc 电压高于 2.5V 则比较器的比较值为 1；

若确定 Vcc 高于 2.5V，则将 P1.0 口输出高，P1.1 口输出低电平，此时若 Vcc 电压低于 3.0V 则比较器的比较值为 0，反之若 Vcc 电压高于 3.0V 则比较器的比较值为 1；

若确定 Vcc 高于 3.0V，则将 P1.1 口输出高，P1.2 口输出低电平，此时若 Vcc 电压低于 3.5V 则比较器的比较值为 0，反之若 Vcc 电压高于 3.5V 则比较器的比较值为 1；

若确定 Vcc 高于 3.5V，则将 P1.2 口输出高，P1.3 口输出低电平，此时若 Vcc 电压低于 4.0V 则比较器的比较值为 0，反之若 Vcc 电压高于 4.0V 则比较器的比较值为 1；

若确定 Vcc 高于 4.0V，则将 P1.3 口输出高，P1.4 口输出低电平，此时若 Vcc 电压低于 4.5V 则比较器的比较值为 0，反之若 Vcc 电压高于 4.5V 则比较器的比较值为 1；

若确定 Vcc 高于 4.5V，则将 P1.4 口输出高，P1.5 口输出低电平，此时若 Vcc 电压低于 5.0V 则比较器的比较值为 0，反之若 Vcc 电压高于 5.0V 则比较器的比较值为 1。

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     CMPCR1    =    0xe6;
sfr     CMPCR2    =    0xe7;
```

```
sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sfr     P2M0    =   0x96;
sfr     P2M1    =   0x95;

void delay ()
{
    char i;

    for (i=0; i<20; i++);
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    unsigned char v;

    P1M0 = 0x3f;                    //P1.5~P1.0 初始化为开漏模式
    P1M1 = 0x3f;
    P1 = 0xff;

    CMPCR2 = 0x10;                  //比较器结果经过 16 个去抖时钟后输出
    CMPCR1 = 0x00;
    CMPCR1 &= ~0x08;                //P3.7 为 CMP+输入脚
    CMPCR1 &= ~0x04;                //内部 1.19V 参考信号源为 CMP-输入脚
    CMPCR1 &= ~0x02;                //禁止比较器输出
    CMPCR1 |= 0x80;                 //使能比较器模块

    while (1)
    {
        v = 0x00;                   //电压<2.5V
        P1 = 0xfe;                  //P1.0 输出 0
        delay();
        if (!(CMPCR1 & 0x01)) goto ShowVol;
        v = 0x01;                   //电压>2.5V
        P1 = 0xfd;                  //P1.1 输出 0
```

```
            delay();
            if (!(CMPCR1 & 0x01)) goto ShowVol;
            v = 0x03;                                   //电压>3.0V
            P1 = 0xfb;                                  //P1.2 输出 0
            delay();
            if (!(CMPCR1 & 0x01)) goto ShowVol;
            v = 0x07;                                   //电压>3.5V
            P1 = 0xf7;                                  //P1.3 输出 0
            delay();
            if (!(CMPCR1 & 0x01)) goto ShowVol;
            v = 0x0f;                                   //电压>4.0V
            P1 = 0xef;                                  //P1.4 输出 0
            delay();
            if (!(CMPCR1 & 0x01)) goto ShowVol;
            v = 0x1f;                                   //电压>4.5V
            P1 = 0xdf;                                  //P1.5 输出 0
            delay();
            if (!(CMPCR1 & 0x01)) goto ShowVol;
            v = 0x3f;                                   //电压>5.0V
ShowVol:
            P1 = 0xff;
            P0 = ~v;
        }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

CMPCR1      DATA        0E6H
CMPCR2      DATA        0E7H

P2M0        DATA        096H
P2M1        DATA        095H
P0M1        DATA        093H
P0M0        DATA        094H
P1M1        DATA        091H
P1M0        DATA        092H
P2M1        DATA        095H
P2M0        DATA        096H
P3M1        DATA        0B1H
P3M0        DATA        0B2H
P4M1        DATA        0B3H
P4M0        DATA        0B4H
P5M1        DATA        0C9H
P5M0        DATA        0CAH

            ORG         0000H
            LJMP        MAIN

            ORG         0100H
MAIN:
            MOV         SP, #5FH
            MOV         P0M0, #00H
            MOV         P0M1, #00H
            MOV         P1M0, #00H
            MOV         P1M1, #00H
            MOV         P2M0, #00H
            MOV         P2M1, #00H
```

```
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P1M0,#00111111B     ;P1.5~P1.0 初始化为开漏模式
        MOV     P1M1,#00111111B
        MOV     P1,#0FFH
        MOV     CMPCR2,#10H         ;比较器结果经过 16 个去抖时钟后输出
        MOV     CMPCR1,#00H
        ANL     CMPCR1,#NOT 08H     ;P3.7 为CMP+输入脚
        ANL     CMPCR1,#NOT 04H     ;内部 1.19V 参考信号源为CMP-输入脚
        ANL     CMPCR1,#NOT 02H     ;禁止比较器输出
        ORL     CMPCR1,#80H         ;使能比较器模块
LOOP:
        MOV     R0,#00000000B       ;电压<2.5V
        MOV     P1,#11111110B       ;P1.0 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00000001B       ;电压>2.5V
        MOV     P1,#11111101B       ;P1.1 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00000011B       ;电压>3.0V
        MOV     P1,#11111011B       ;P1.2 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00000111B       ;电压>3.5V
        MOV     P1,#11110111B       ;P1.3 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00001111B       ;电压>4.0V
        MOV     P1,#11101111B       ;P1.4 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00011111B       ;电压>4.5V
        MOV     P1,#11011111B       ;P1.5 输出 0
        CALL    DELAY
        MOV     A,CMPCR1
        JNB     ACC.0,SKIP
        MOV     R0,#00111111B       ;电压>5.0V
SKIP:
        MOV     P1,#11111111B
        MOV     A,R0
        CPL     A
        MOV     P0,A                ;P0.5~P0.0 口显示电压
        JMP     LOOP

DELAY:
        MOV     R0,#20
        DJNZ    R0,$
        RET
```

END

STC MCU

16 IAP/EEPROM/DATA-FLASH

STC8C 系列单片机内部集成了大容量的 EEPROM。利用 ISP/IAP 技术可将内部 Data Flash 当 EEPROM，擦写次数在 10 万次以上。EEPROM 可分为若干个扇区，每个扇区包含 512 字节。

注意：EEPROM 的写操作只能将字节中的 1 写为 0，当需要将字节中的 0 写为 1，则必须执行扇区擦除操作。EEPROM 的读/写操作是以 1 字节为单位进行，而 EEPROM 擦除操作是以 1 扇区（512 字节）为单位进行，在执行擦除操作时，如果目标扇区中有需要保留的数据，则必须预先将这些数据读取到 RAM 中暂存，待擦除完成后再将保存的数据和需要更新的数据一起再写回 EEPROM/DATA-FLASH。

所以在使用时，建议同一次修改的数据放在同一个扇区，不是同一次修改的数据放在不同的扇区，不一定要用满。数据存储器的擦除操作是按扇区进行的（每扇区 512 字节）。

EEPROM 可用于保存一些需要在应用过程中修改并且掉电不丢失的参数数据。在用户程序中，可以对 EEPROM 进行字节读/字节编程/扇区擦除操作。在工作电压偏低时，建议不要进行 EEPROM 操作，以免发送数据丢失的情况。

16.1 EEPROM 操作时间

- 读取 1 字节：4 个系统时钟（使用 MOVC 指令读取更方便快捷）
- 编程 1 字节：约 30～40us（实际的编程时间为 6~7.5us，但还需要加上状态转换时间和各种控制信号的 SETUP 和 HOLD 时间）
- 擦除 1 扇区（512 字节）：约 4～6ms

EEPROM 操作所需时间是硬件自动控制的，用户只需要正确设置 IAP_TPS 寄存器即可。

IAP_TPS＝系统工作频率/1000000（小数部分四舍五入进行取整）

例如：系统工作频率为 12MHz，则 IAP_TPS 设置为 12

又例如：系统工作频率为 22.1184MHz，则 IAP_TPS 设置为 22

再例如：系统工作频率为 5.5296MHz，则 IAP_TPS 设置为 6

STCMCU

16.2 EEPROM 相关的寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
IAP_DATA	IAP 数据寄存器	C2H									1111,1111
IAP_ADDRH	IAP 高地址寄存器	C3H									0000,0000
IAP_ADDRL	IAP 低地址寄存器	C4H									0000,0000
IAP_CMD	IAP 命令寄存器	C5H	-	-	-	-	-	-	CMD[1:0]		xxxx,xx00
IAP_TRIG	IAP 触发寄存器	C6H									0000,0000
IAP_CONTR	IAP 控制寄存器	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-	0000,xxxx
IAP_TPS	IAP 等待时间控制寄存器	F5H	-	-	IAPTPS[5:0]						xx00,0000

16.2.1 EEPROM 数据寄存器（IAP_DATA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_DATA	C2H								

在进行 EEPROM 的读操作时，命令执行完成后读出的 EEPROM 数据保存在 IAP_DATA 寄存器中。在进行 EEPROM 的写操作时，在执行写命令前，必须将待写入的数据存放在 IAP_DATA 寄存器中，再发送写命令。擦除 EEPROM 命令与 IAP_DATA 寄存器无关。

16.2.2 EEPROM 地址寄存器（IAP_ADDR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_ADDRH	C3H								
IAP_ADDRL	C4H								

EEPROM 进行读、写、擦除操作的目标地址寄存器。IAP_ADDRH 保存地址的高字节，IAP_ADDRL 保存地址的低字节

16.2.3 EEPROM 命令寄存器（IAP_CMD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CMD	C5H	-	-	-	-	-	-	CMD[1:0]	

CMD[1:0]：发送EEPROM操作命令

00：空操作

01：读 EEPROM 命令。读取目标地址所在的 1 字节。

10：写 EEPROM 命令。写目标地址所在的 1 字节。注意：写操作只能将目标字节中的 1 写为 0，而不能将 0 写为 1。一般当目标字节不为 FFH 时，必须先擦除。

11：擦除 EEPROM。擦除目标地址所在的 1 页（1 扇区/512 字节）。注意：擦除操作会一次擦除 1 个扇区（512 字节），整个扇区的内容全部变成 FFH。

16.2.4 EEPROM 触发寄存器（IAP_TRIG）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TRIG	C6H								

设置完成 EEPROM 读、写、擦除的命令寄存器、地址寄存器、数据寄存器以及控制寄存器后，需要向触发寄存器 IAP_TRIG 依次写入 5AH、A5H（顺序不能交换）两个触发命令来触发相应的读、写、擦除操作。操作完成后，EEPROM 地址寄存器 IAP_ADDRH、IAP_ADDRL 和 EEPROM 命令寄存器 IAP_CMD 的内容不变。如果接下来要对下一个地址的数据进行操作，需手动更新地址寄存器 IAP_ADDRH 和寄存器 IAP_ADDRL 的值。

注意：每次 EEPROM 操作时，都要对 IAP_TRIG 先写入 5AH，再写入 A5H，相应的命令才会生效。写完触发命令后，CPU 会处于 IDLE 等待状态，直到相应的 IAP 操作执行完成后 CPU 才会从 IDLE 状态返回正常状态继续执行 CPU 指令。

16.2.5 EEPROM 控制寄存器（IAP_CONTR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_CONTR	C7H	IAPEN	SWBS	SWRST	CMD_FAIL	-	-	-	-

IAPEN：EEPROM操作使能控制位

0：禁止 EEPROM 操作

1：使能 EEPROM 操作

SWBS：软件复位选择控制位，（需要与SWRST配合使用）

0：软件复位后从用户代码开始执行程序

1：软件复位后从系统 ISP 监控代码区开始执行程序

SWRST：软件复位控制位

0：无动作

1：产生软件复位

CMD_FAIL：EEPROM操作失败状态位，需要软件清零

0：EEPROM 操作正确

1：EEPROM 操作失败

16.2.6 EEPROM 等待时间控制寄存器（IAP_TPS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
IAP_TPS	F5H	-	-	IAPTPS[5:0]					

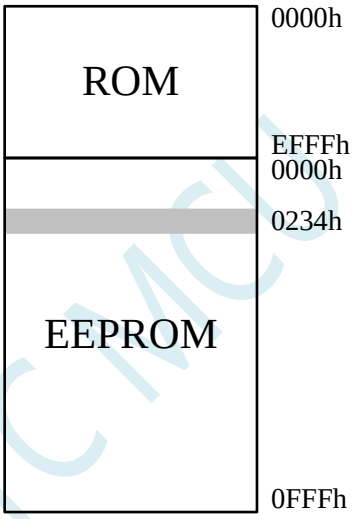
需要根据工作频率进行设置

若工作频率为12MHz，则需要将IAP_TPS设置为12；若工作频率为24MHz，则需要将IAP_TPS设置为24，其他频率以此类推。

16.3 EEPROM 大小及地址

STC8C 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。

STC8C 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 STC8C2K60S4 这个型号为例，对目标地址进行详细说明：



STC8G2K60S4

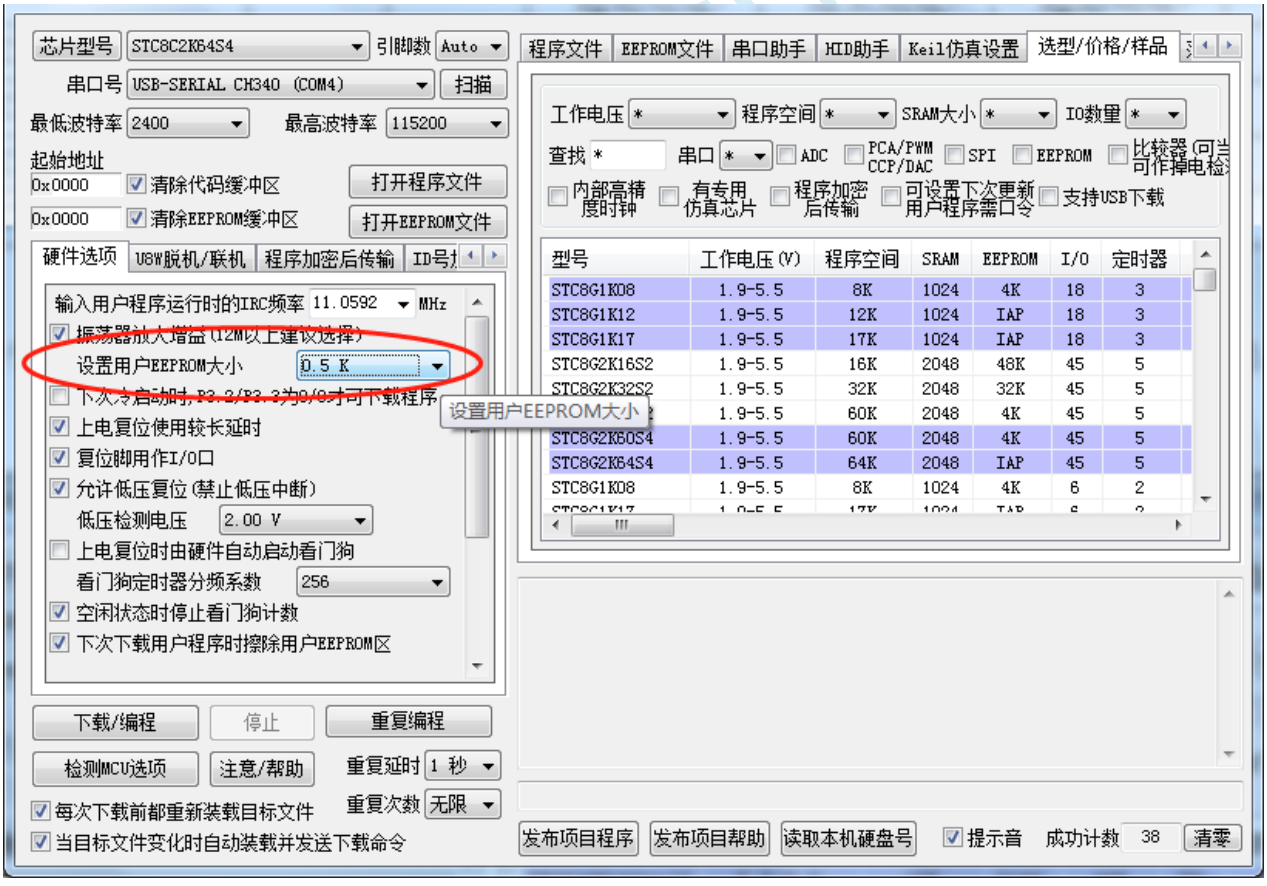
STC8C2K60S4 的程序空间为 60K 字节（0000h~EFFFh），EEPROM 空间为 4K（0000h~0FFFh）。当需要对 EEPROM 物理地址 0234h 的单元进行读、写、擦除时，若使用 IAP 方式进行访问时，设置的目标地址为 0234h，即 IAP_ADDRH 设置 02h，IAP_ADDRL 设置 34h，然后设置相应的触发命令即可对 0234h 单元进行正确操作了。但若是使用 MOVC 方式读取 EEPROM 的 0234h 单元，则必须在 0234h 的基础上还有加上 Flash 程序存储器（ROM）空间的大小 F000h，即必须将 DPTR 设置为 F234h，然后才能使用 MOVC 指令进行读取。

注意：由于擦除是以 512 字节为单位进行操作的，所以执行擦除操作时所设置的目标地址的低 9 位是无意义的。例如：执行擦除命令时，设置地址 0234H/0200H/0300H/03FFH，最终执行擦除的动作都是相同的，都是擦除 0200H~03FFH 这 512 字节。

不同型号内部 EEPROM 的大小及访问地址会存在差异，针对各个型号 EEPROM 的详细大小和地址请参考下表

型号	大小	扇区	IAP 方式读/写/擦除		MOVC 读取	
			起始地址	结束地址	起始地址	结束地址
STC8C2K16S4	48K	96	0000h	BFFFh	4000h	FFFFh
STC8C2K32S4	32K	64	0000h	7FFFh	8000h	FFFFh
STC8C2K48S4	16K	32	0000h	3FFFh	C000h	FFFFh
STC8C2K60S4	4K	8	0000h	0FFFh	F000h	FFFFh
STC8C2K64S4	用户自定义[1]					
STC8C2K16S2	48K	96	0000h	BFFFh	4000h	FFFFh
STC8C2K32S2	32K	64	0000h	7FFFh	8000h	FFFFh
STC8C2K48S2	16K	64	0000h	3FFFh	C000h	FFFFh
STC8C2K60S2	4K	8	0000h	0FFFh	F000h	FFFFh
STC8C2K64S2	用户自定义[1]					

[1]：这个为特殊型号，这个型号的 EEPROM 大小是可用在 ISP 下载时用户自己设置的。如下图所示：



用户可用根据自己的需要在整个 FLASH 空间中规划出任意不超过 FLASH 大小的 EEPROM 空间，但需要注意：**EEPROM 总是从后向前进行规划的**。

例如：STC8C2K64S4 这个型号的 FLASH 为 64K，此时若用户想分出其中的 4K 作为 EEPROM 使用，则 EEPROM 的物理地址即为 64K 的最后 4K，物理地址为 F000h~FFFFh，当然，用户若使用 IAP 的方式

进行访问，目标地址仍然从 0000h 开始，到 0FFFh 结束，当使用 MOVC 读取则需要从 F000h 开始，到 FFFFh 结束。

STC MCU

16.4 范例程序

16.4.1 EEPROM 基本操作

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sfr         IAP_DATA    =   0xC2;
sfr         IAP_ADDRH   =   0xC3;
sfr         IAP_ADDRL   =   0xC4;
sfr         IAP_CMD     =   0xC5;
sfr         IAP_TRIG    =   0xC6;
sfr         IAP_CONTR   =   0xC7;
sfr         IAP_TPS     =   0xF5;

void IapIdle()
{
    IAP_CONTR = 0;                  //关闭 IAP 功能
    IAP_CMD = 0;                    //清除命令寄存器
    IAP_TRIG = 0;                   //清除触发寄存器
    IAP_ADDRH = 0x80;               //将地址设置到非 IAP 区域
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    char dat;

    IAP_CONTR = 0x80;               //使能 IAP
    IAP_TPS = 12;                   //设置等待参数 12MHz
    IAP_CMD = 1;                    //设置 IAP 读命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_TRIG = 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;
```

STC MCU

```
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = 0x80;                   //使能 IAP
    IAP_TPS = 12;                       //设置等待参数 12MHz
    IAP_CMD = 2;                        //设置 IAP 写命令
    IAP_ADDRL = addr;                   //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;              //设置 IAP 高地址
    IAP_DATA = dat;                     //写 IAP 数据
    IAP_TRIG = 0x5a;                    //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                    //写触发命令(0xa5)
    _nop_();
    IapIdle();                          //关闭 IAP 功能
}

void IapErase(int addr)
{
    IAP_CONTR = 0x80;                   //使能 IAP
    IAP_TPS = 12;                       //设置等待参数 12MHz
    IAP_CMD = 3;                        //设置 IAP 擦除命令
    IAP_ADDRL = addr;                   //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;              //设置 IAP 高地址
    IAP_TRIG = 0x5a;                    //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                    //写触发命令(0xa5)
    _nop_();                            //
    IapIdle();                          //关闭 IAP 功能
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IapErase(0x0400);
    P0 = IapRead(0x0400);               //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);               //P1=0x12

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H
IAP_ADDRL   DATA    0C4H
```

```
IAP_CMD     DATA    0C5H
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H
IAP_TPS     DATA    0F5H

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H

IAP_IDLE:
            MOV     IAP_CONTR,#0        ;关闭 IAP 功能
            MOV     IAP_CMD,#0          ;清除命令寄存器
            MOV     IAP_TRIG,#0         ;清除触发寄存器
            MOV     IAP_ADDRH,#80H      ;将地址设置到非 IAP 区域
            MOV     IAP_ADDRL,#0
            RET

IAP_READ:
            MOV     IAP_CONTR,#80H      ;使能 IAP
            MOV     IAP_TPS,#12         ;设置等待参数 12MHz
            MOV     IAP_CMD,#1          ;设置 IAP 读命令
            MOV     IAP_ADDRL,DPL       ;设置 IAP 低地址
            MOV     IAP_ADDRH,DPH       ;设置 IAP 高地址
            MOV     IAP_TRIG,#5AH       ;写触发命令(0x5a)
            MOV     IAP_TRIG,#0A5H      ;写触发命令(0xa5)
            NOP
            MOV     A,IAP_DATA          ;读取 IAP 数据
            LCALL   IAP_IDLE            ;关闭 IAP 功能
            RET

IAP_PROGRAM:
            MOV     IAP_CONTR,#80H      ;使能 IAP
            MOV     IAP_TPS,#12         ;设置等待参数 12MHz
            MOV     IAP_CMD,#2          ;设置 IAP 写命令
            MOV     IAP_ADDRL,DPL       ;设置 IAP 低地址
            MOV     IAP_ADDRH,DPH       ;设置 IAP 高地址
            MOV     IAP_DATA,A          ;写 IAP 数据
            MOV     IAP_TRIG,#5AH       ;写触发命令(0x5a)
            MOV     IAP_TRIG,#0A5H      ;写触发命令(0xa5)
            NOP
            LCALL   IAP_IDLE            ;关闭 IAP 功能
            RET

IAP_ERASE:
            MOV     IAP_CONTR,#80H      ;使能 IAP
```

```
                MOV         IAP_TPS,#12             ;设置等待参数 12MHz
                MOV         IAP_CMD,#3              ;设置 IAP 擦除命令
                MOV         IAP_ADDRL,DPL           ;设置 IAP 低地址
                MOV         IAP_ADDRH,DPH           ;设置 IAP 高地址
                MOV         IAP_TRIG,#5AH           ;写触发命令(0x5a)
                MOV         IAP_TRIG,#0A5H          ;写触发命令(0xa5)
                NOP
                LCALL       IAP_IDLE                ;关闭 IAP 功能
                RET

MAIN:
                MOV         SP, #5FH
                MOV         P0M0, #00H
                MOV         P0M1, #00H
                MOV         P1M0, #00H
                MOV         P1M1, #00H
                MOV         P2M0, #00H
                MOV         P2M1, #00H
                MOV         P3M0, #00H
                MOV         P3M1, #00H
                MOV         P4M0, #00H
                MOV         P4M1, #00H
                MOV         P5M0, #00H
                MOV         P5M1, #00H

                MOV         DPTR,#0400H
                LCALL       IAP_ERASE
                MOV         DPTR,#0400H
                LCALL       IAP_READ
                MOV         P0,A                    ;P0=0FFH
                MOV         DPTR,#0400H
                MOV         A,#12H
                LCALL       IAP_PROGRAM
                MOV         DPTR,#0400H
                LCALL       IAP_READ
                MOV         P1,A                    ;P1=12H

                SJMP        $

                END
```

16.4.2　使用 MOVC 读取 EEPROM

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
```

```
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sfr         IAP_DATA    =   0xC2;
sfr         IAP_ADDRH   =   0xC3;
sfr         IAP_ADDRL   =   0xC4;
sfr         IAP_CMD     =   0xC5;
sfr         IAP_TRIG    =   0xC6;
sfr         IAP_CONTR   =   0xC7;
sfr         IAP_TPS     =   0xF5;

#define     IAP_OFFSET      0xF000H             //STC8C2K60S4
void IapIdle()
{
    IAP_CONTR = 0;                              //关闭 IAP 功能
    IAP_CMD = 0;                                //清除命令寄存器
    IAP_TRIG = 0;                               //清除触发寄存器
    IAP_ADDRH = 0x80;                           //将地址设置到非 IAP 区域
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    addr += IAP_OFFSET;                         //使用 MOVC 读取 EEPROM 需要加上相应的偏移
    return *(char code *)(addr);                //使用 MOVC 读取数据
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = 0x80;                           //使能 IAP
    IAP_TPS = 12;                               //设置等待参数 12MHz
    IAP_CMD = 2;                                //设置 IAP 写命令
    IAP_ADDRL = addr;                           //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;                      //设置 IAP 高地址
    IAP_DATA = dat;                             //写 IAP 数据
    IAP_TRIG = 0x5a;                            //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                            //写触发命令(0xa5)
    _nop_();
    IapIdle();                                  //关闭 IAP 功能
}

void IapErase(int addr)
{
    IAP_CONTR = 0x80;                           //使能 IAP
    IAP_TPS = 12;                               //设置等待参数 12MHz
    IAP_CMD = 3;                                //设置 IAP 擦除命令
    IAP_ADDRL = addr;                           //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;                      //设置 IAP 高地址
    IAP_TRIG = 0x5a;                            //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                            //写触发命令(0xa5)
    _nop_();                                    //
    IapIdle();                                  //关闭 IAP 功能
}

void main()
{
```

STC MCU

```
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    IapErase(0x0400);
    P0 = IapRead(0x0400);                    //P0=0xff
    IapProgram(0x0400, 0x12);
    P1 = IapRead(0x0400);                    //P1=0x12

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

IAP_DATA    DATA    0C2H
IAP_ADDRH   DATA    0C3H
IAP_ADDRL   DATA    0C4H
IAP_CMD     DATA    0C5H
IAP_TRIG    DATA    0C6H
IAP_CONTR   DATA    0C7H
IAP_TPS     DATA    0F5H

IAP_OFFSET  EQU     0F000H              ;STC8C2K60S4

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H

IAP_IDLE:
            MOV     IAP_CONTR,#0        ;关闭 IAP 功能
            MOV     IAP_CMD,#0          ;清除命令寄存器
            MOV     IAP_TRIG,#0         ;清除触发寄存器
            MOV     IAP_ADDRH,#80H      ;将地址设置到非 IAP 区域
            MOV     IAP_ADDRL,#0
```

```
        RET

IAP_READ:
        MOV     A,#LOW IAP_OFFSET       ;使用 MOVC 读取 EEPROM 需要加上相应的偏移
        ADD     A,DPL
        MOV     DPL,A
        MOV     A,@HIGH IAP_OFFSET
        ADDC    A,DPH
        MOV     DPH,A
        CLR     A
        MOVC    A,@A+DPTR               ;使用 MOVC 读取数据
        RET

IAP_PROGRAM:
        MOV     IAP_CONTR,#80H          ;使能 IAP
        MOV     IAP_TPS,#12             ;设置等待参数 12MHz
        MOV     IAP_CMD,#2              ;设置 IAP 写命令
        MOV     IAP_ADDRL,DPL           ;设置 IAP 低地址
        MOV     IAP_ADDRH,DPH           ;设置 IAP 高地址
        MOV     IAP_DATA,A              ;写 IAP 数据
        MOV     IAP_TRIG,#5AH           ;写触发命令(0x5a)
        MOV     IAP_TRIG,#0A5H          ;写触发命令(0xa5)
        NOP
        LCALL   IAP_IDLE                ;关闭 IAP 功能
        RET

IAP_ERASE:
        MOV     IAP_CONTR,#80H          ;使能 IAP
        MOV     IAP_TPS,#12             ;设置等待参数 12MHz
        MOV     IAP_CMD,#3              ;设置 IAP 擦除命令
        MOV     IAP_ADDRL,DPL           ;设置 IAP 低地址
        MOV     IAP_ADDRH,DPH           ;设置 IAP 高地址
        MOV     IAP_TRIG,#5AH           ;写触发命令(0x5a)
        MOV     IAP_TRIG,#0A5H          ;写触发命令(0xa5)
        NOP
        LCALL   IAP_IDLE                ;关闭 IAP 功能
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     DPTR,#0400H
        LCALL   IAP_ERASE
        MOV     DPTR,#0400H
        LCALL   IAP_READ
        MOV     P0,A                    ;P0=0FFH
        MOV     DPTR,#0400H
```

```
        MOV     A,#12H
        LCALL   IAP_PROGRAM
        MOV     DPTR,#0400H
        LCALL   IAP_READ
        MOV     P1,A                    ;P1=12H

        SJMP    $

        END
```

16.4.3 使用串口送出 EEPROM 数据

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define FOSC        11059200UL
#define BRT         (65536 - FOSC / 115200 / 4)

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sfr     AUXR        =   0x8e;
sfr     T2H         =   0xd6;
sfr     T2L         =   0xd7;

sfr     IAP_DATA    =   0xC2;
sfr     IAP_ADDRH   =   0xC3;
sfr     IAP_ADDRL   =   0xC4;
sfr     IAP_CMD     =   0xC5;
sfr     IAP_TRIG    =   0xC6;
sfr     IAP_CONTR   =   0xC7;
sfr     IAP_TPS     =   0xF5;

void UartInit()
{
    SCON = 0x5a;
    T2L = BRT;
    T2H = BRT >> 8;
    AUXR = 0x15;
}

void UartSend(char dat)
{
```

```
    while (!TI);
    TI = 0;
    SBUF = dat;
}

void IapIdle()
{
    IAP_CONTR = 0;                  //关闭 IAP 功能
    IAP_CMD = 0;                    //清除命令寄存器
    IAP_TRIG = 0;                   //清除触发寄存器
    IAP_ADDRH = 0x80;               //将地址设置到非 IAP 区域
    IAP_ADDRL = 0;
}

char IapRead(int addr)
{
    char dat;

    IAP_CONTR = 0x80;               //使能 IAP
    IAP_TPS = 12;                   //设置等待参数 12MHz
    IAP_CMD = 1;                    //设置 IAP 读命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();
    dat = IAP_DATA;                 //读 IAP 数据
    IapIdle();                      //关闭 IAP 功能

    return dat;
}

void IapProgram(int addr, char dat)
{
    IAP_CONTR = 0x80;               //使能 IAP
    IAP_TPS = 12;                   //设置等待参数 12MHz
    IAP_CMD = 2;                    //设置 IAP 写命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_DATA = dat;                 //写 IAP 数据
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();
    IapIdle();                      //关闭 IAP 功能
}

void IapErase(int addr)
{
    IAP_CONTR = 0x80;               //使能 IAP
    IAP_TPS = 12;                   //设置等待参数 12MHz
    IAP_CMD = 3;                    //设置 IAP 擦除命令
    IAP_ADDRL = addr;               //设置 IAP 低地址
    IAP_ADDRH = addr >> 8;          //设置 IAP 高地址
    IAP_TRIG = 0x5a;                //写触发命令(0x5a)
    IAP_TRIG = 0xa5;                //写触发命令(0xa5)
    _nop_();                        //
    IapIdle();                      //关闭 IAP 功能
}
```

```
void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    UartInit();
    IapErase(0x0400);
    UartSend(IapRead(0x0400));
    IapProgram(0x0400, 0x12);
    UartSend(IapRead(0x0400));

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

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

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H

UART_INIT:
```

```
        MOV     SCON,#5AH
        MOV     T2L,#0E8H       ;65536-11059200/115200/4=0FFE8H
        MOV     T2H,#0FFH
        MOV     AUXR,#15H
        RET

UART_SEND:
        JNB     TI,$
        CLR     TI
        MOV     SBUF,A
        RET

IAP_IDLE:
        MOV     IAP_CONTR,#0    ;关闭 IAP 功能
        MOV     IAP_CMD,#0      ;清除命令寄存器
        MOV     IAP_TRIG,#0     ;清除触发寄存器
        MOV     IAP_ADDRH,#80H  ;将地址设置到非 IAP 区域
        MOV     IAP_ADDRL,#0
        RET

IAP_READ:
        MOV     IAP_CONTR,#80H  ;使能 IAP
        MOV     IAP_TPS,#12     ;设置等待参数 12MHz
        MOV     IAP_CMD,#1      ;设置 IAP 读命令
        MOV     IAP_ADDRL,DPL   ;设置 IAP 低地址
        MOV     IAP_ADDRH,DPH   ;设置 IAP 高地址
        MOV     IAP_TRIG,#5AH   ;写触发命令(0x5a)
        MOV     IAP_TRIG,#0A5H  ;写触发命令(0xa5)
        NOP
        MOV     A,IAP_DATA      ;读取 IAP 数据
        LCALL   IAP_IDLE        ;关闭 IAP 功能
        RET

IAP_PROGRAM:
        MOV     IAP_CONTR,#80H  ;使能 IAP
        MOV     IAP_TPS,#12     ;设置等待参数 12MHz
        MOV     IAP_CMD,#2      ;设置 IAP 写命令
        MOV     IAP_ADDRL,DPL   ;设置 IAP 低地址
        MOV     IAP_ADDRH,DPH   ;设置 IAP 高地址
        MOV     IAP_DATA,A      ;写 IAP 数据
        MOV     IAP_TRIG,#5AH   ;写触发命令(0x5a)
        MOV     IAP_TRIG,#0A5H  ;写触发命令(0xa5)
        NOP
        LCALL   IAP_IDLE        ;关闭 IAP 功能
        RET

IAP_ERASE:
        MOV     IAP_CONTR,#80H  ;使能 IAP
        MOV     IAP_TPS,#12     ;设置等待参数 12MHz
        MOV     IAP_CMD,#3      ;设置 IAP 擦除命令
        MOV     IAP_ADDRL,DPL   ;设置 IAP 低地址
        MOV     IAP_ADDRH,DPH   ;设置 IAP 高地址
        MOV     IAP_TRIG,#5AH   ;写触发命令(0x5a)
        MOV     IAP_TRIG,#0A5H  ;写触发命令(0xa5)
        NOP
        LCALL   IAP_IDLE        ;关闭 IAP 功能
        RET

MAIN:
```

```
MOV     SP, #5FH
MOV     P0M0, #00H
MOV     P0M1, #00H
MOV     P1M0, #00H
MOV     P1M1, #00H
MOV     P2M0, #00H
MOV     P2M1, #00H
MOV     P3M0, #00H
MOV     P3M1, #00H
MOV     P4M0, #00H
MOV     P4M1, #00H
MOV     P5M0, #00H
MOV     P5M1, #00H

LCALL   UART_INIT
MOV     DPTR,#0400H
LCALL   IAP_ERASE
MOV     DPTR,#0400H
LCALL   IAP_READ
LCALL   UART_SEND
MOV     DPTR,#0400H
MOV     A,#12H
LCALL   IAP_PROGRAM
MOV     DPTR,#0400H
LCALL   IAP_READ
LCALL   UART_SEND

SJMP    $

END
```

STC MCU

17 同步串行外设接口 SPI

STC8C 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。STC8C 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

17.1.1 SPI 状态寄存器（SPSTAT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPSTAT	CDH	SPIF	WCOL	-	-	-	-	-	-

SPIF：SPI 中断标志位。

当发送/接收完成 1 字节的数据后，硬件自动将此位置 1，并向 CPU 提出中断请求。当 SSIG 位被设置为 0 时，由于 SS 管脚电平的变化而使得设备的主/从模式发生改变时，此标志位也会被硬件自动置 1，以标志设备模式发生变化。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

WCOL：SPI 写冲突标志位。

当 SPI 在进行数据传输的过程中写 SPDAT 寄存器时，硬件将此位置 1。

注意：此标志位必须用户通过软件方式向此位写 1 进行清零。

17.1.2 SPI 控制寄存器（SPCTL），SPI 速度控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPCTL	CEH	SSIG	SPEN	DORD	MSTR	CPOL	CPHA	SPR[1:0]	

SSIG：SS 引脚功能控制位

0：SS 引脚确定器件是主机还是从机

1：忽略 SS 引脚功能，使用 MSTR 确定器件是主机还是从机

SPEN：SPI 使能控制位

0：关闭 SPI 功能

1：使能 SPI 功能

DORD：SPI 数据位发送/接收的顺序

0：先发送/接收数据的高位（MSB）

1：先发送/接收数据的低位（LSB）

MSTR：器件主/从模式选择位

设置主机模式：

若 SSIG＝0，则 SS 管脚必须为高电平且设置 MSTR 为 1

若 SSIG＝1，则只需要设置 MSTR 为 1（忽略 SS 管脚的电平）

设置从机模式：

若 SSIG＝0，则 SS 管脚必须为低电平（与 MSTR 位无关）

若 SSIG＝1，则只需要设置 MSTR 为 0（忽略 SS 管脚的电平）

CPOL：SPI 时钟极性控制

0：SCLK 空闲时为低电平，SCLK 的前时钟沿为上升沿，后时钟沿为下降沿

1：SCLK 空闲时为高电平，SCLK 的前时钟沿为下降沿，后时钟沿为上升沿

CPHA：SPI 时钟相位控制

0：数据 SS 管脚为低电平驱动第一位数据并在 SCLK 的后时钟沿改变数据，前时钟沿采样数据（必须 SSIG＝0）

1：数据在 SCLK 的前时钟沿驱动，后时钟沿采样

SPR[1:0]：SPI 时钟频率选择

SPR[1:0]	SCLK 频率
00	SYSclk/4
01	SYSclk/8
10	SYSclk/16
11	SYSclk/32

17.1.3 SPI 数据寄存器（SPDAT）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
SPDAT	CFH								

SPI 发送/接收数据缓冲器。

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）

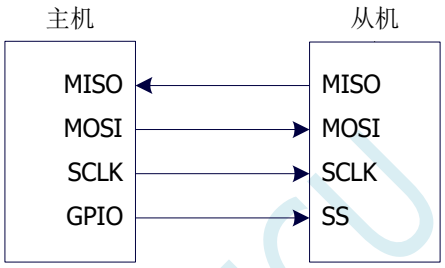
17.2.1 单主单从

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

STC MCU

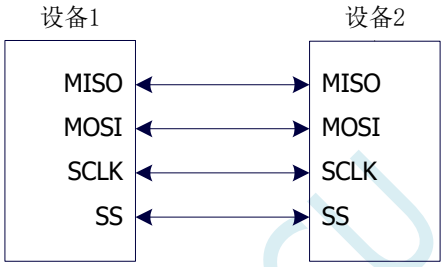
17.2.2 互为主从

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

STC MCU

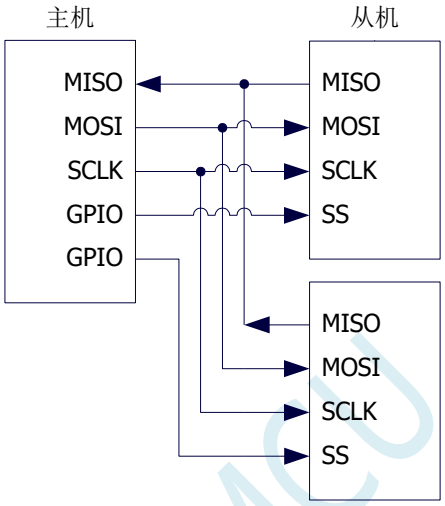
17.2.3 单主多从

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

STC MCU

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 I/O
1	0	0	0	输出	输入	输入	**从机模式**，且被选中
1	0	0	1	高阻	输入	输入	**从机模式**，但未被选中
1	0	1→0	0	输出	输入	输入	**从机模式**，不忽略 SS 且 MSTR 为 1 的主机模式，当 SS 管脚被拉低时，MSTR 将被硬件自动清零，工作模式将被被动设置为从机模式
1	0	1	1	输入	高阻	高阻	**主机模式**，空闲状态
					输出	输出	**主机模式**，激活状态
1	1	0	x	输出	输入	输入	**从机模式**
1	1	1	x	输入	输出	输出	**主机模式**

从机模式的注意事项：

当 CPHA＝0 时，SSIG 必须为 0（即不能忽略 SS 脚）。在每次串行字节开始还发送前 SS 脚必须拉低，并且在串行字节发送完后须重新设置为高电平。SS 管脚为低电平时不能对 SPDAT 寄存器执行写操作，否则将导致一个写冲突错误。CPHA=0 且 SSIG=1 时的操作未定义。

当 CPHA＝1 时，SSIG 可以置 1（即可以忽略脚）。如果 SSIG＝0，SS 脚可在连续传输之间保持低有效（即一直固定为低电平）。这种方式适用于固定单主单从的系统。

主机模式的注意事项：

在 SPI 中，传输总是由主机启动的。如果 SPI 使能（SPEN=1）并选择作为主机时，主机对 SPI 数据寄存器 SPDAT 的写操作将启动 SPI 时钟发生器和数据的传输。在数据写入 SPDAT 之后的半个到一个 SPI 位时间后，数据将出现在 MOSI 脚。写入主机 SPDAT 寄存器的数据从 MOSI 脚移出发送到从机的 MOSI 脚。同时从机 SPDAT 寄存器的数据从 MISO 脚移出发送到主机的 MISO 脚。

传输完一个字节后，SPI 时钟发生器停止，传输完成标志（SPIF）置位，如果 SPI 中断使能则会产生一个 SPI 中断。主机和从机 CPU 的两个移位寄存器可以看作是一个 16 位循环移位寄存器。当数据从主机移位传送到从机的同时，数据也以相反的方向移入。这意味着在一个移位周期中，主机和从机的数据相互交换。

通过 SS 改变模式

如果 SPEN=1，SSIG=0 且 MSTR=1，SPI 使能为主机模式，并将 SS 脚可配置为输入模式化或准双向口模式。这种情况下，另外一个主机可将该脚驱动为低电平，从而将该器件选择为 SPI 从机并向其发送数据。为了避免争夺总线，SPI 系统将该从机的 MSTR 清零，MOSI 和 SCLK 强制变为输入模式，而 MISO 则变为输出模式，同时 SPSTAT 的 SPIF 标志位置 1。

用户软件必须一直对 MSTR 位进行检测，如果该位被一个从机选择动作而被动清零，而用户想继续将 SPI 作为主机，则必须重新设置 MSTR 位，否则将一直处于从机模式。

写冲突

SPI 在发送时为单缓冲，在接收时为双缓冲。这样在前一次发送尚未完成之前，不能将新的数据写

入移位寄存器。当发送过程中对数据寄存器 SPDAT 进行写操作时，WCOL 位将被置 1 以指示发生数据写冲突错误。在这种情况下，当前发送的数据继续发送，而新写入的数据将丢失。

当对主机或从机进行写冲突检测时，主机发生写冲突的情况是很罕见的，因为主机拥有数据传输的完全控制权。但从机有可能发生写冲突，因为当主机启动传输时，从机无法进行控制。

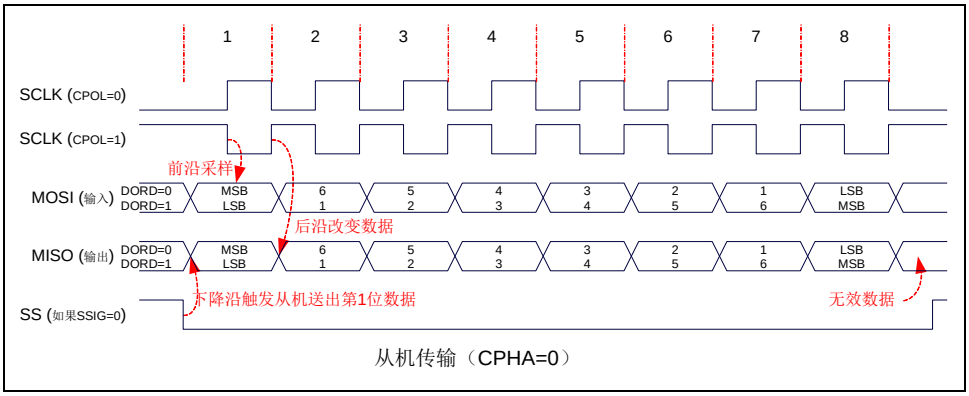
接收数据时，接收到的数据传送到一个并行读数据缓冲区，这样将释放移位寄存器以进行下一个数据的接收。但必须在下个字符完全移入之前从数据寄存器中读出接收到的数据，否则，前一个接收数据将丢失。

WCOL 可通过软件向其写入“1”清零。

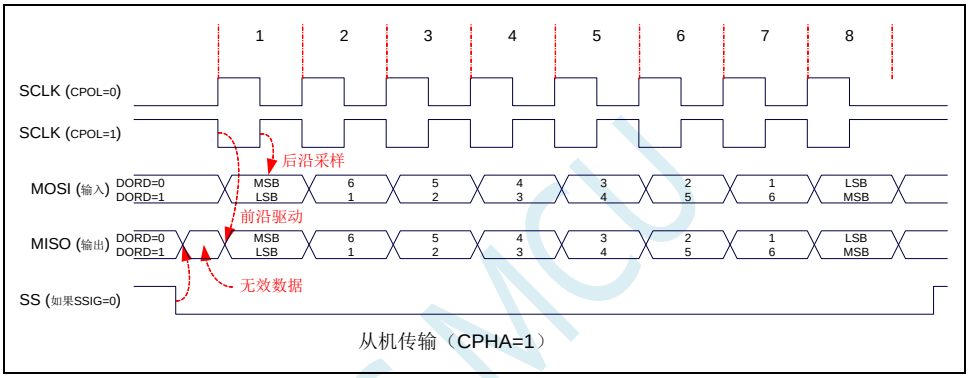
STC MCU

17.4 数据模式

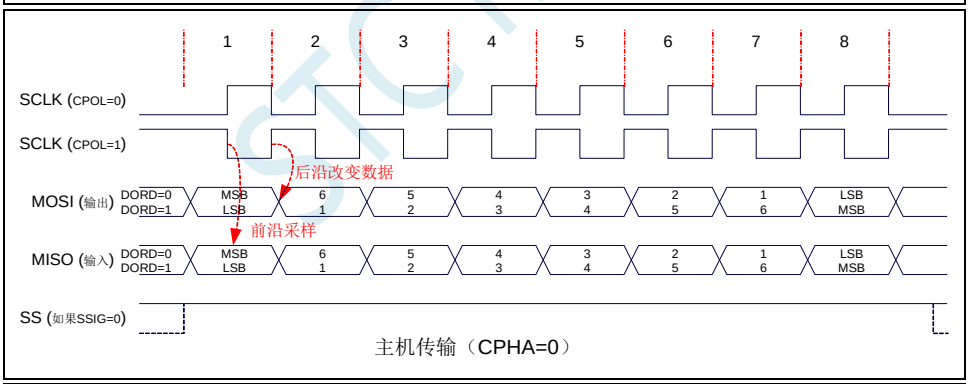
SPI 的时钟相位控制位 CPHA 可以让用户设定数据采样和改变时的时钟沿。时钟极性位 CPOL 可以让用户设定时钟极性。下面图例显示了不同时钟相位、极性设置下 SPI 通讯时序。



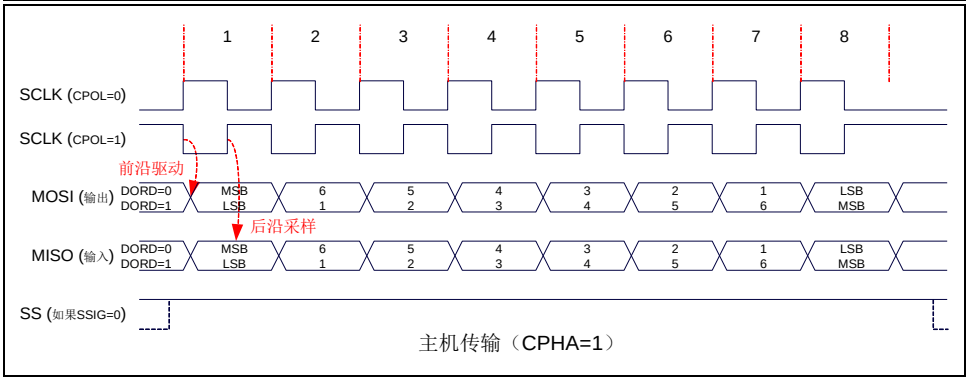
从机传输（CPHA=0）



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）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SS          =   P1^0;
sbit        LED         =   P1^1;

bit         busy;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //清中断标志
    SS = 1;                         //拉高从机的 SS 管脚
    busy = 0;
    LED = !LED;                     //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
```

STC MCU

```
    LED = 1;
    SS = 1;
    busy = 0;

    SPCTL = 0x50;                                   //使能 SPI 主机模式
    SPSTAT = 0xc0;                                  //清中断标志
    IE2 = ESPI;                                     //使能 SPI 中断
    EA = 1;

    while (1)
    {
        while (busy);
        busy = 1;
        SS = 0;                                     //拉低从机 SS 管脚
        SPDAT = 0x5a;                               //发送测试数据
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

BUSY        BIT     20H.0
SS          BIT     P1.0
LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H        ;清中断标志
            SETB    SS                  ;拉高从机的 SS 管脚
            CLR     BUSY
            CPL     LED
            RETI
```

```
MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        SETB    LED
        SETB    SS
        CLR     BUSY

        MOV     SPCTL,#50H          ;使能 SPI 主机模式
        MOV     SPSTAT,#0C0H        ;清中断标志
        MOV     IE2,#ESPI           ;使能 SPI 中断
        SETB    EA

LOOP:
        JB      BUSY,$
        SETB    BUSY
        CLR     SS                  ;拉低从机 SS 管脚
        MOV     SPDAT,#5AH          ;发送测试数据
        JMP     LOOP

        END
```

17.5.2　SPI 单主单从系统从机程序（中断方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
sfr     SPDAT       =   0xcf;
sfr     IE2         =   0xaf;
#define ESPI            0x02

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
```

```
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sbit    LED     =   P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //清中断标志
    SPDAT = SPDAT;                  //将接收到的数据回传给主机
    LED = !LED;                     //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;                   //使能 SPI 从机模式
    SPSTAT = 0xc0;                  //清中断标志
    IE2 = ESPI;                     //使能 SPI 中断
    EA = 1;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
```

```
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
            CPL     LED
            RETI

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     SPCTL,#40H          ;使能 SPI 从机模式
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     IE2,#ESPI           ;使能 SPI 中断
            SETB    EA

            JMP     $

            END
```

17.5.3　SPI 单主单从系统主机程序（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P0M1    =   0x93;
sfr     P0M0    =   0x94;
sfr     P1M1    =   0x91;
sfr     P1M0    =   0x92;
sfr     P2M1    =   0x95;
sfr     P2M0    =   0x96;
sfr     P3M1    =   0xb1;
sfr     P3M0    =   0xb2;
```

```
sfr     P4M1    =   0xb3;
sfr     P4M0    =   0xb4;
sfr     P5M1    =   0xc9;
sfr     P5M0    =   0xca;

sfr     SPSTAT  =   0xcd;
sfr     SPCTL   =   0xce;
sfr     SPDAT   =   0xcf;
sfr     IE2     =   0xaf;
#define ESPI        0x02

sbit    SS      =   P1^0;
sbit    LED     =   P1^1;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    SS = 1;

    SPCTL = 0x50;                       //使能 SPI 主机模式
    SPSTAT = 0xc0;                      //清中断标志

    while (1)
    {
        SS = 0;                         //拉低从机 SS 管脚
        SPDAT = 0x5a;                   //发送测试数据
        while (!(SPSTAT & 0x80));       //查询完成标志
        SPSTAT = 0xc0;                  //清中断标志
        SS = 1;                         //拉高从机的 SS 管脚
        LED = !LED;                     //测试端口
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT  DATA    0CDH
SPCTL   DATA    0CEH
SPDAT   DATA    0CFH
IE2     DATA    0AFH
ESPI    EQU     02H

SS      BIT     P1.0
LED     BIT     P1.1
```

```
P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    LED
            SETB    SS

            MOV     SPCTL,#50H          ;使能 SPI 主机模式
            MOV     SPSTAT,#0C0H        ;清中断标志

LOOP:
            CLR     SS                  ;拉低从机 SS 管脚
            MOV     SPDAT,#5AH          ;发送测试数据
            MOV     A,SPSTAT            ;查询完成标志
            JNB     ACC.7,$-2
            MOV     SPSTAT,#0C0H        ;清中断标志
            SETB    SS
            CPL     LED
            JMP     LOOP

            END
```

17.5.4 SPI 单主单从系统从机程序（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz
```

```
#include "reg51.h"
#include "intrins.h"

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        LED         =   P1^1;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                          //清中断标志
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    SPCTL = 0x40;                           //使能 SPI 从机模式
    SPSTAT = 0xc0;                          //清中断标志

    while (1)
    {
        while (!(SPSTAT & 0x80));           //查询完成标志
        SPSTAT = 0xc0;                      //清中断标志
        SPDAT = SPDAT;                      //将接收到的数据回传给主机
        LED = !LED;                         //测试端口
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz
```

```
SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

LED         BIT     P1.1

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     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
```

17.5.5 SPI 互为主从系统程序（中断方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         SPSTAT      =   0xcd;
sfr         SPCTL       =   0xce;
sfr         SPDAT       =   0xcf;
sfr         IE2         =   0xaf;
#define     ESPI            0x02

sfr         P0M1        =   0x93;
sfr         P0M0        =   0x94;
sfr         P1M1        =   0x91;
sfr         P1M0        =   0x92;
sfr         P2M1        =   0x95;
sfr         P2M0        =   0x96;
sfr         P3M1        =   0xb1;
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SS          =   P1^0;
sbit        LED         =   P1^1;
sbit        KEY         =   P0^0;

void SPI_Isr() interrupt 9
{
    SPSTAT = 0xc0;                  //清中断标志
    if (SPCTL & 0x10)
    {                               //主机模式
        SS = 1;                     //拉高从机的 SS 管脚
        SPCTL = 0x40;               //重新设置为从机待机
    }
    else
    {                               //从机模式
        SPDAT = SPDAT;              //将接收到的数据回传给主机
    }
    LED = !LED;                     //测试端口
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;
```

STC MCU

```
    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;                           //使能 SPI 从机模式进行待机
    SPSTAT = 0xc0;                          //清中断标志
    IE2 = ESPI;                             //使能 SPI 中断
    EA = 1;

    while (1)
    {
        if (!KEY)                           //等待按键触发
        {
            SPCTL = 0x50;                   //使能 SPI 主机模式
            SS = 0;                         //拉低从机 SS 管脚
            SPDAT = 0x5a;                   //发送测试数据
            while (!KEY);                   //等待按键释放
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

SS          BIT     P1.0
LED         BIT     P1.1
KEY         BIT     P0.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     004BH
            LJMP    SPIISR

            ORG     0100H
SPIISR:
            PUSH    ACC
            MOV     SPSTAT,#0C0H        ;清中断标志
            MOV     A,SPCTL
            JB      ACC.4,MASTER
```

```
SLAVE:
        MOV     SPDAT,SPDAT             ;将接收到的数据回传给主机
        JMP     ISREXIT
MASTER:
        SETB    SS                      ;拉高从机的 SS 管脚
        MOV     SPCTL,#40H              ;重新设置为从机待机
ISREXIT:
        CPL     LED
        POP     ACC
        RETI

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        SETB    SS
        SETB    LED
        SETB    KEY

        MOV     SPCTL,#40H              ;使能 SPI 从机模式进行待机
        MOV     SPSTAT,#0C0H            ;清中断标志
        MOV     IE2,#ESPI               ;使能 SPI 中断
        SETB    EA

LOOP:
        JB      KEY,LOOP                ;等待按键触发
        MOV     SPCTL,#50H              ;使能 SPI 主机模式
        CLR     SS                      ;拉低从机 SS 管脚
        MOV     SPDAT,#5AH              ;发送测试数据
        JNB     KEY,$                   ;等待按键释放
        JMP     LOOP

        END
```

17.5.6　SPI 互为主从系统程序（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     SPSTAT      =   0xcd;
sfr     SPCTL       =   0xce;
```

```
sfr     SPDAT       =   0xcf;
sfr     IE2         =   0xaf;
#define ESPI            0x02

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    SS          =   P1^0;
sbit    LED         =   P1^1;
sbit    KEY         =   P0^0;

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    LED = 1;
    KEY = 1;
    SS = 1;

    SPCTL = 0x40;                   //使能 SPI 从机模式进行待机
    SPSTAT = 0xc0;                  //清中断标志

    while (1)
    {
        if (!KEY)                   //等待按键触发
        {
            SPCTL = 0x50;           //使能 SPI 主机模式
            SS = 0;                 //拉低从机 SS 管脚
            SPDAT = 0x5a;           //发送测试数据
            while (!KEY);           //等待按键释放
        }
        if (SPSTAT & 0x80)
        {
            SPSTAT = 0xc0;          //清中断标志
            if (SPCTL & 0x10)
            {                       //主机模式
                SS = 1;             //拉高从机的 SS 管脚
                SPCTL = 0x40;       //重新设置为从机待机
```

```
            }
            else
            {                                   //从机模式
                SPDAT = SPDAT;                  //将接收到的数据回传给主机
            }
            LED = !LED;                         //测试端口
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

SPSTAT      DATA    0CDH
SPCTL       DATA    0CEH
SPDAT       DATA    0CFH
IE2         DATA    0AFH
ESPI        EQU     02H

SS          BIT     P1.0
LED         BIT     P1.1
KEY         BIT     P0.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            SETB    SS
            SETB    LED
            SETB    KEY
```

STC MCU

```
        MOV     SPCTL,#40H          ;使能 SPI 从机模式进行待机
        MOV     SPSTAT,#0C0H        ;清中断标志

LOOP:
        JB      KEY,SKIP            ;等待按键触发
        MOV     SPCTL,#50H          ;使能 SPI 主机模式
        CLR     SS                  ;拉低从机 SS 管脚
        MOV     SPDAT,#5AH          ;发送测试数据
        JNB     KEY,$               ;等待按键释放
SKIP:
        MOV     A,SPSTAT
        JNB     ACC.7,LOOP
        MOV     SPSTAT,#0C0H        ;清中断标志
        MOV     A,SPCTL
        JB      ACC.4,MASTER
SLAVE:
        MOV     SPDAT,SPDAT         ;将接收到的数据回传给主机
        CPL     LED
        JMP     LOOP
MASTER:
        SETB    SS                  ;拉高从机的 SS 管脚
        MOV     SPCTL,#40H          ;重新设置为从机待机
        CPL     LED
        JMP     LOOP

        END
```

STC MCU

18 I²C 总线

STC8C 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，STC8C 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

STC8C 系列的 I²C 总线提供了两种操作模式：主机模式（SCL 为输出口，发送同步时钟信号）和从机模式（SCL 为输入口，接收同步时钟信号）

STC 创新：STC 的 I²C 串行总线控制器工作在从机模式时，SDA 管脚的下降沿信号可以唤醒进入掉电模式的 MCU。（注意：由于 I²C 传输速度比较快，MCU 唤醒后第一包数据一般是不正确的）

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[5:0]						0000,0000
I2CMSCR	I²C 主机控制寄存器	FE81H	EMSI	-	-	-	MSCMD[3:0]				0xxx,0000
I2CMSST	I²C 主机状态寄存器	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO	00xx,xx00
I2CSLCR	I²C 从机控制寄存器	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST	x000,0xx0
I2CSLST	I²C 从机状态寄存器	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	TXING	SLACKI	SLACKO	0000,0000
I2CSLADR	I²C 从机地址寄存器	FE85H	I2CSLADR[7:1]							MA	0000,0000
I2CTXD	I²C 数据发送寄存器	FE86H									0000,0000
I2CRXD	I²C 数据接收寄存器	FE87H									0000,0000
I2CMSAUX	I²C 主机辅助控制寄存器	FE88H	-	-	-	-	-	-	-	WDTA	xxxx,xxx0

18.2 I²C 主机模式

18.2.1 I2C 配置寄存器（I2CCFG），总线速度控制

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CCFG	FE80H	ENI2C	MSSL	MSSPEED[5:0]					

ENI2C：I²C 功能使能控制位

0：禁止 I²C 功能

1：允许 I²C 功能

MSSL：I²C 工作模式选择位

0：从机模式

1：主机模式

MSSPEED[5:0]：I²C 总线速度（等待时钟数）控制，**I2C 总线速度＝F_{OSC} / 2 / (MSSPEED * 2 + 4)**

MSSPEED[5:0]	对应的时钟数
0	4
1	6
2	8
…	…
x	2x+4
…	…
62	128
63	130

只有当 I²C 模块工作在主机模式时，MSSPEED 参数设置的等待参数才有效。此等待参数主要用于主机模式的以下几个信号：

T_{SSTA}：起始信号的建立时间（Setup Time of START）

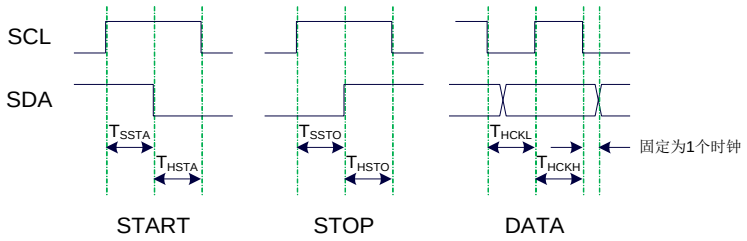
T_{HSTA}：起始信号的保持时间（Hold Time of START）

T_{SSTO}：停止信号的建立时间（Setup Time of STOP）

T_{HSTO}：停止信号的保持时间（Hold Time of STOP）

T_{HCKL}：时钟信号的低电平保持时间（Hold Time of SCL Low）

T_{HCKH}：时钟信号的高电平保持时间（Hold Time of SCL High）



例 1：当 MSSPEED＝10 时，T_{SSTA}＝T_{HSTA}＝T_{SSTO}＝T_{HSTO}＝T_{HCKL}＝24/FOSC

例 2：当 24MHz 的工作频率下需要 400K 的 I2C 总线速度时，

MSSPEED＝(24M / 400K / 2 - 4) / 2＝13

18.2.2 I2C 主机控制寄存器（I2CMSCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSCR	FE81H	EMSI	-	-	-	MSCMD[3:0]			

EMSI：主机模式中断使能控制位

0：关闭主机模式的中断

1：允许主机模式的中断

MSCMD[3:0]：主机命令

0000：待机，无动作。

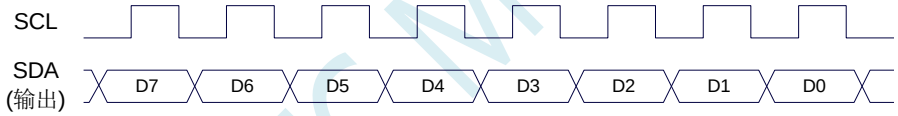
0001：起始命令。

发送 START 信号。如果当前 I²C 控制器处于空闲状态，即 MSBUSY（I2CMSST.7）为 0 时，写此命令会使控制器进入忙状态，硬件自动将 MSBUSY 状态位置 1，并开始发送 START 信号；**若当前 I²C 控制器处于忙状态，写此命令可触发发送 START 信号**。发送 START 信号的波形如下图所示：



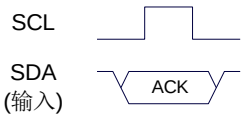
0010：发送数据命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 8 个时钟，并将 I2CTXD 寄存器里面数据按位送到 SDA 管脚上（先发送高位数据）。发送数据的波形如下图所示：



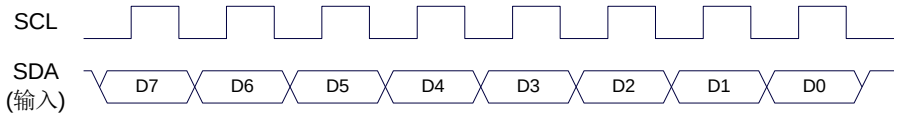
0011：接收 ACK 命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 1 个时钟，并将从 SDA 端口上读取的数据保存到 MSACKI（I2CMSST.1）。接收 ACK 的波形如下图所示：



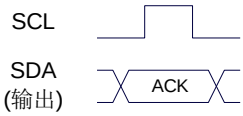
0100：接收数据命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 8 个时钟，并将从 SDA 端口上读取的数据依次左移到 I2CRXD 寄存器（先接收高位数据）。接收数据的波形如下图所示：



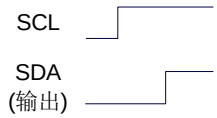
0101：发送 ACK 命令。

写此命令后，I²C 总线控制器会在 SCL 管脚上产生 1 个时钟，并将 MSACKO（I2CMSST.0）中的数据发送到 SDA 端口。发送 ACK 的波形如下图所示：



0110：停止命令。

发送 STOP 信号。写此命令后，I^2C 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。

18.2.3 I2C 主机辅助控制寄存器（I2CMSAUX）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSAUX	FE88H	-	-	-	-	-	-	-	WDTA

WDTA：主机模式时 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。

18.2.4 I2C 主机状态寄存器（I2CMSST）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CMSST	FE82H	MSBUSY	MSIF	-	-	-	-	MSACKI	MSACKO

MSBUSY：主机模式时 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。

STC MCU

18.3 I²C 从机模式

18.3.1 I2C 从机控制寄存器（I2CSLCR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLCR	FE83H	-	ESTAI	ERXI	ETXI	ESTOI	-	-	SLRST

ESTAI：从机模式时接收到 START 信号中断允许位

0：禁止从机模式时接收到 START 信号时发生中断

1：使能从机模式时接收到 START 信号时发生中断

ERXI：从机模式时接收到 1 字节数据后中断允许位

0：禁止从机模式时接收到数据后发生中断

1：使能从机模式时接收到 1 字节数据后发生中断

ETXI：从机模式时发送完成 1 字节数据后中断允许位

0：禁止从机模式时发送完成数据后发生中断

1：使能从机模式时发送完成 1 字节数据后发生中断

ESTOI：从机模式时接收到 STOP 信号中断允许位

0：禁止从机模式时接收到 STOP 信号时发生中断

1：使能从机模式时接收到 STOP 信号时发生中断

SLRST：复位从机模式

18.3.2 I2C 从机状态寄存器（I2CSLST）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLST	FE84H	SLBUSY	STAIF	RXIF	TXIF	STOIF	-	SLACKI	SLACKO

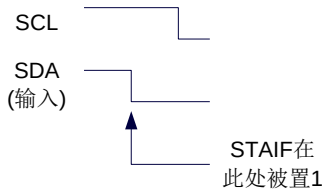
SLBUSY：从机模式时 I²C 控制器状态位（只读位）

0：控制器处于空闲状态

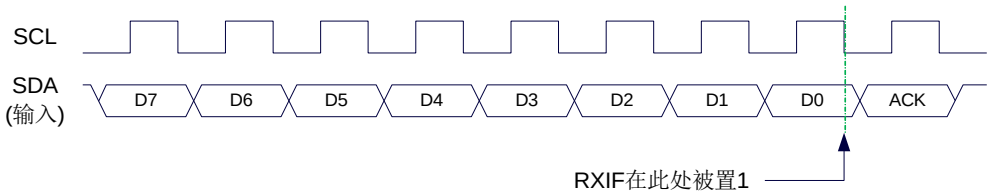
1：控制器处于忙碌状态

当 I²C 控制器处于从机模式时，在空闲状态下，接收到主机发送 START 信号后，控制器会继续检测之后的设备地址数据，若设备地址与当前 I2CSLADR 寄存器中所设置的从机地址相同时，控制器便进入到忙碌状态，忙碌状态会一直维持到成功接收到主机发送 STOP 信号，之后状态会再次恢复到空闲状态。

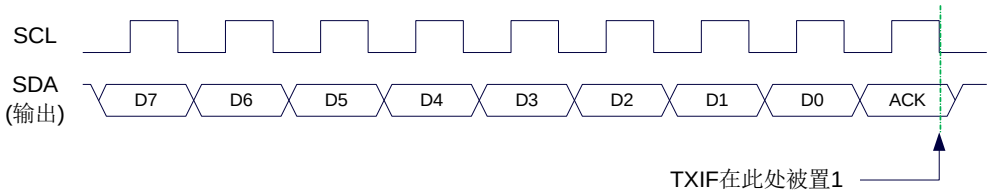
STAIF：从机模式时接收到 START 信号后的中断请求位。从机模式的 I²C 控制器接收到 START 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STAIF 位必须用软件清零。STAIF 被置 1 的时间点如下图所示：



RXIF：从机模式时接收到 1 字节的数据后的中断请求位。从机模式的 I²C 控制器接收到 1 字节的数据后，在第 8 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 RXIF 位必须用软件清零。RXIF 被置 1 的时间点如下图所示：



TXIF：从机模式时发送完成 1 字节的数据后的中断请求位。从机模式的 I²C 控制器发送完成 1 字节的数据并成功接收到 1 位 ACK 信号后，在第 9 个时钟的下降沿时硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 TXIF 位必须用软件清零。TXIF 被置 1 的时间点如下图所示：

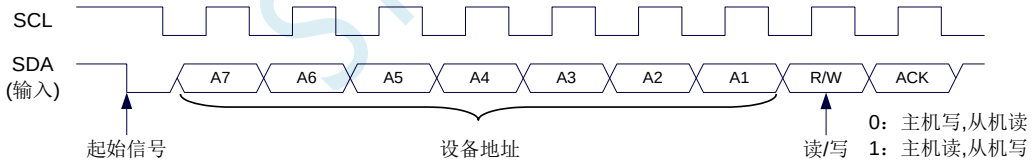


STOIF：从机模式时接收到 STOP 信号后的中断请求位。从机模式的 I²C 控制器接收到 STOP 信号后，硬件会自动将此位置 1，并向 CPU 发请求中断，响应中断后 STOIF 位必须用软件清零。STOIF 被置 1 的时间点如下图所示：



SLACKI：从机模式时，接收到的 ACK 数据。

SLACKO：从机模式时，准备将要发送出去的 ACK 信号。



18.3.3　I2C 从机地址寄存器（I2CSLADR）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CSLADR	FE85H	I2CSLADR[7:1]							MA

I2CSLADR[7:1]：从机设备地址

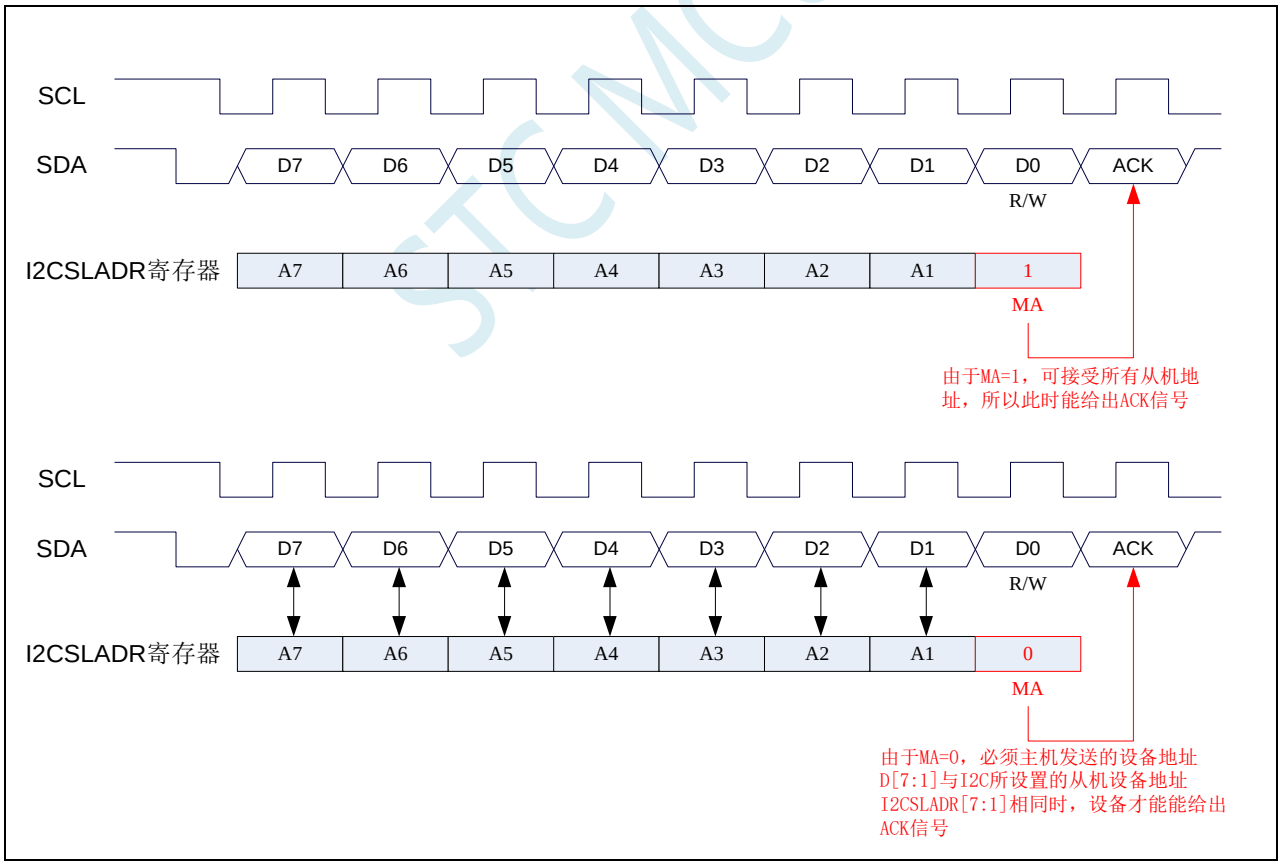
当 I²C 控制器处于从机模式时，控制器在接收到 START 信号后，会继续检测接下来主机发送出的设备地址数据以及读/写信号。当主机发送出的设备地址与 I2CSLADR[7:1]中所设置的从机设备地址相同时，控制器才会向 CPU 发出中断求，请求 CPU 处理 I²C 事件；否则若设备地址不同，I²C 控制器继续监控，等待下一个起始信号，对下一个设备地址继续比较。

MA：从机设备地址比较控制

0：设备地址必须与 I2CSLADR[7:1]相同

1：忽略 I2CSLADR[7:1]中的设置，接受所有的设备地址

说明：I2C 总线协议规定 I2C 总线上最多可挂载 128 个 I2C 设备（理论值），不同的 I2C 设备用不同的 I2C 从机设备地址进行识别。I2C 主机发送完成起始信号后，发送的第一个数据（DATA0）的高 7 位即为从机设备地址（DATA0[7:1]为 I2C 设备地址），最低位为读写信号。当 I2C 设备从机地址寄存器 MA（I2CSLADR.0）为 1 时，表示 I2C 从机能够接受所有的设备地址，此时主机发送的任何设备地址，即 DATA0[7:1]为任何值，从机都能响应。当 I2C 设备从机地址寄存器 MA（I2CSLADR.0）为 0 时，主机发送的设备地址 DATA0[7:1]必须与从机的设备地址 I2CSLADR[7:1]相同时才能访问此从机设备



18.3.4 I2C 数据寄存器（I2CTXD，I2CRXD）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
I2CTXD	FE86H								
I2CRXD	FE87H								

I2CTXD 是 I^2C 发送数据寄存器，存放将要发送的 I^2C 数据

I2CRXD 是 I^2C 接收数据寄存器，存放接收完成的 I^2C 数据

STC MCU

18.4 范例程序

18.4.1 I²C 主机模式访问 AT24C256（中断方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =   0xba;

#define     I2CCFG          (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR         (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST         (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR         (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST         (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR        (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD          (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD          (*(unsigned char volatile xdata *)0xfe87)

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

bit         busy;

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;
    if (I2CMSST & 0x40)
    {
        I2CMSST &= ~0x40;                       //清中断标志
        busy = 0;
    }
    _pop_(P_SW2);
}

void Start()
{
    busy = 1;
    I2CMSCR = 0x81;                             //发送 START 命令
    while (busy);
```

```
}

void SendData(char dat)
{
    I2CTXD = dat;                       //写数据到数据缓冲区
    busy = 1;
    I2CMSCR = 0x82;                     //发送 SEND 命令
    while (busy);
}

void RecvACK()
{
    busy = 1;
    I2CMSCR = 0x83;                     //发送读 ACK 命令
    while (busy);
}

char RecvData()
{
    busy = 1;
    I2CMSCR = 0x84;                     //发送 RECV 命令
    while (busy);
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                     //设置 ACK 信号
    busy = 1;
    I2CMSCR = 0x85;                     //发送 ACK 命令
    while (busy);
}

void SendNAK()
{
    I2CMSST = 0x01;                     //设置 NAK 信号
    busy = 1;
    I2CMSCR = 0x85;                     //发送 ACK 命令
    while (busy);
}

void Stop()
{
    busy = 1;
    I2CMSCR = 0x86;                     //发送 STOP 命令
    while (busy);
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
```

STC MCU

```
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;
    EA = 1;

    Start();    //发送起始命令
    SendData(0xa0);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    SendData(0x00);                         //发送存储地址低字节
    RecvACK();
    SendData(0x12);                         //写测试数据 1
    RecvACK();
    SendData(0x78);                         //写测试数据 2
    RecvACK();
    Stop();                                 //发送停止命令

    Delay();                                //等待设备写数据

    Start();                                //发送起始命令
    SendData(0xa0);                         //发送设备地址+写命令
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    SendData(0x00);                         //发送存储地址低字节
    RecvACK();
    Start();                                //发送起始命令
    SendData(0xa1);                         //发送设备地址+读命令
    RecvACK();
    P0 = RecvData();                        //读取数据 1
    SendACK();
    P2 = RecvData();                        //读取数据 2
    SendNAK();
    Stop();                                 //发送停止命令

    P_SW2 = 0x00;

    while (1);
}
```

STC MCU

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

BUSY        BIT     20H.0

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    DPL
            PUSH    DPH

            MOV     DPTR,#I2CMSST       ;清中断标志
            MOVX    A,@DPTR
            ANL     A,#NOT 40H
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            CLR     BUSY                ;复位忙标志

            POP     DPH
            POP     DPL
            POP     ACC
            RETI

START:
            SETB    BUSY
            MOV     A,#10000001B        ;发送 START 命令
            MOV     DPTR,#I2CMSCR
```

```
        MOVX    @DPTR,A
        JMP     WAIT
SENDDATA:
        MOV     DPTR,#I2CTXD            ;写数据到数据缓冲区
        MOVX    @DPTR,A
        SETB    BUSY
        MOV     A,#10000010B            ;发送 SEND 命令
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
RECVACK:
        SETB    BUSY
        MOV     A,#10000011B            ;发送读 ACK 命令
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
RECVDATA:
        SETB    BUSY
        MOV     A,#10000100B            ;发送 RECV 命令
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        CALL    WAIT
        MOV     DPTR,#I2CRXD            ;从数据缓冲区读取数据
        MOVX    A,@DPTR
        RET
SENDACK:
        MOV     A,#00000000B            ;设置 ACK 信号
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        SETB    BUSY
        MOV     A,#10000101B            ;发送 ACK 命令
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
SENDNAK:
        MOV     A,#00000001B            ;设置 NAK 信号
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        SETB    BUSY
        MOV     A,#10000101B            ;发送 ACK 命令
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
STOP:
        SETB    BUSY
        MOV     A,#10000110B            ;发送 STOP 命令
        MOV     DPTR,#I2CMSCR
        MOVX    @DPTR,A
        JMP     WAIT
WAIT:
        JB      BUSY,$                  ;等待命令发送完成
        RET

DELAY:
        MOV     R0,#0
        MOV     R1,#0
DELAY1:
        NOP
        NOP
```

```
        NOP
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H

        MOV     A,#11100000B        ;设置 I2C 模块为主机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A
        SETB    EA

        CALL    START               ;发送起始命令
        MOV     A,#0A0H
        CALL    SENDDATA            ;发送设备地址+写命令
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址高字节
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址低字节
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#12H              ;写测试数据 1
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#78H              ;写测试数据 2
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP                ;发送停止命令

        CALL    DELAY               ;等待设备写数据

        CALL    START               ;发送起始命令
        MOV     A,#0A0H             ;发送设备地址+写命令
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址高字节
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址低字节
```

```
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START                   ;发送起始命令
        MOV     A,#0A1H                 ;发送设备地址+读命令
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA                ;读取数据 1
        MOV     P0,A
        CALL    SENDACK
        CALL    RECVDATA                ;读取数据 2
        MOV     P2,A
        CALL    SENDNAK
        CALL    STOP                    ;发送停止命令

        JMP     $

        END
```

18.4.2 I²C 主机模式访问 AT24C256（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    SDA         =   P1^4;
sbit    SCL         =   P1^5;

void Wait()
{
```

```
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                         //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                           //写数据到数据缓冲区
    I2CMSCR = 0x02;                         //发送 SEND 命令
    Wait();
}

void 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_();
```

STC MCU

```
            _nop_();
        }
}

void main()
{
        P0M0 = 0x00;
        P0M1 = 0x00;
        P1M0 = 0x00;
        P1M1 = 0x00;
        P2M0 = 0x00;
        P2M1 = 0x00;
        P3M0 = 0x00;
        P3M1 = 0x00;
        P4M0 = 0x00;
        P4M1 = 0x00;
        P5M0 = 0x00;
        P5M1 = 0x00;

        P_SW2 = 0x80;

        I2CCFG = 0xe0;                          //使能 I2C 主机模式
        I2CMSST = 0x00;

        Start();                                //发送起始命令
        SendData(0xa0);                         //发送设备地址+写命令
        RecvACK();
        SendData(0x00);                         //发送存储地址高字节
        RecvACK();
        SendData(0x00);                         //发送存储地址低字节
        RecvACK();
        SendData(0x12);                         //写测试数据 1
        RecvACK();
        SendData(0x78);                         //写测试数据 2
        RecvACK();
        Stop();                                 //发送停止命令

        Delay();                                //等待设备写数据

        Start();                                //发送起始命令
        SendData(0xa0);                         //发送设备地址+写命令
        RecvACK();
        SendData(0x00);                         //发送存储地址高字节
        RecvACK();
        SendData(0x00);                         //发送存储地址低字节
        RecvACK();
        Start();                                //发送起始命令
        SendData(0xa1);                         //发送设备地址+读命令
        RecvACK();
        P0 = RecvData();                        //读取数据 1
        SendACK();
        P2 = RecvData();                        //读取数据 2
        SendNAK();
        Stop();                                 //发送停止命令

        P_SW2 = 0x00;

        while (1);
}
```

STC MCU

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
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
```

STC MCU

```
            MOVX    A,@DPTR
            RET
SENDACK:
            MOV     A,#00000000B        ;设置 ACK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDNAK:
            MOV     A,#00000001B        ;设置 NAK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
STOP:
            MOV     A,#00000110B        ;发送 STOP 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
WAIT:
            MOV     DPTR,#I2CMSST       ;清中断标志
            MOVX    A,@DPTR
            JNB     ACC.6,WAIT
            ANL     A,#NOT 40H
            MOVX    @DPTR,A
            RET

DELAY:
            MOV     R0,#0
            MOV     R1,#0
DELAY1:
            NOP
            NOP
            NOP
            NOP
            DJNZ    R1,DELAY1
            DJNZ    R0,DELAY1
            RET

MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     P_SW2,#80H
```

STC MCU

```
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
```

18.4.3 I²C 主机模式访问 PCF8563

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =   0xba;

#define     I2CCFG              (*(unsigned char volatile xdata *)0xfe80)
#define     I2CMSCR             (*(unsigned char volatile xdata *)0xfe81)
#define     I2CMSST             (*(unsigned char volatile xdata *)0xfe82)
#define     I2CSLCR             (*(unsigned char volatile xdata *)0xfe83)
#define     I2CSLST             (*(unsigned char volatile xdata *)0xfe84)
#define     I2CSLADR            (*(unsigned char volatile xdata *)0xfe85)
#define     I2CTXD              (*(unsigned char volatile xdata *)0xfe86)
#define     I2CRXD              (*(unsigned char volatile xdata *)0xfe87)

sfr         P0M1            =   0x93;
sfr         P0M0            =   0x94;
sfr         P1M1            =   0x91;
sfr         P1M0            =   0x92;
sfr         P2M1            =   0x95;
sfr         P2M0            =   0x96;
sfr         P3M1            =   0xb1;
sfr         P3M0            =   0xb2;
sfr         P4M1            =   0xb3;
sfr         P4M0            =   0xb4;
sfr         P5M1            =   0xc9;
sfr         P5M0            =   0xca;

sbit        SDA             =   P1^4;
sbit        SCL             =   P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                             //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                               //写数据到数据缓冲区
    I2CMSCR = 0x02;                             //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                             //发送读 ACK 命令
    Wait();
}

char RecvData()
```

STC MCU

```
{
    I2CMSCR = 0x04;                         //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
    I2CMSST = 0x00;                         //设置 ACK 信号
    I2CMSCR = 0x05;                         //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                         //设置 NAK 信号
    I2CMSCR = 0x05;                         //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                         //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;
```

STC MCU

```
    Start();                                    //发送起始命令
    SendData(0xa2);                             //发送设备地址+写命令
    RecvACK();
    SendData(0x02);                             //发送存储地址
    RecvACK();
    SendData(0x00);                             //设置秒值
    RecvACK();
    SendData(0x00);                             //设置分钟值
    RecvACK();
    SendData(0x12);                             //设置小时值
    RecvACK();
    Stop();                                     //发送停止命令

    while (1)
    {
        Start();                                //发送起始命令
        SendData(0xa2);                         //发送设备地址+写命令
        RecvACK();
        SendData(0x02);                         //发送存储地址
        RecvACK();
        Start();                                //发送起始命令
        SendData(0xa3);                         //发送设备地址+读命令
        RecvACK();
        P0 = RecvData();                        //读取秒值
        SendACK();
        P2 = RecvData();                        //读取分钟值
        SendACK();
        P3 = RecvData();                        //读取小时值
        SendNAK();
        Stop();                                 //发送停止命令

        Delay();
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
```

```
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
START:
            MOV     A,#00000001B        ;发送 START 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDDATA:
            MOV     DPTR,#I2CTXD        ;写数据到数据缓冲区
            MOVX    @DPTR,A
            MOV     A,#00000010B        ;发送 SEND 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVACK:
            MOV     A,#00000011B        ;发送读 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
RECVDATA:
            MOV     A,#00000100B        ;发送 RECV 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            CALL    WAIT
            MOV     DPTR,#I2CRXD        ;从数据缓冲区读取数据
            MOVX    A,@DPTR
            RET
SENDACK:
            MOV     A,#00000000B        ;设置 ACK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
SENDNAK:
            MOV     A,#00000001B        ;设置 NAK 信号
            MOV     DPTR,#I2CMSST
            MOVX    @DPTR,A
            MOV     A,#00000101B        ;发送 ACK 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
STOP:
            MOV     A,#00000110B        ;发送 STOP 命令
            MOV     DPTR,#I2CMSCR
            MOVX    @DPTR,A
            JMP     WAIT
WAIT:
            MOV     DPTR,#I2CMSST       ;清中断标志
```

```
        MOVX    A,@DPTR
        JNB     ACC.6,WAIT
        ANL     A,#NOT 40H
        MOVX    @DPTR,A
        RET

DELAY:
        MOV     R0,#0
        MOV     R1,#0
DELAY1:
        NOP
        NOP
        NOP
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H

        MOV     A,#11100000B            ;设置 I2C 模块为主机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A

        CALL    START                   ;发送起始命令
        MOV     A,#0A2H
        CALL    SENDDATA                ;发送设备地址+写命令
        CALL    RECVACK
        MOV     A,#002H                 ;发送存储地址
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#00H                  ;设置秒值
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#00H                  ;设置分钟值
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#12H                  ;设置小时值
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP                    ;发送停止命令
```

```
LOOP:
        CALL    START           ;发送起始命令
        MOV     A,#0A2H         ;发送设备地址+写命令
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#002H         ;发送存储地址
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START           ;发送起始命令
        MOV     A,#0A3H         ;发送设备地址+读命令
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA        ;读取秒值
        MOV     P0,A
        CALL    SENDACK
        CALL    RECVDATA        ;读取分钟值
        MOV     P2,A
        CALL    SENDACK
        CALL    RECVDATA        ;读取小时值
        MOV     P3,A
        CALL    SENDNAK
        CALL    STOP            ;发送停止命令

        CALL    DELAY

        JMP     LOOP

        END
```

18.4.4　I²C 从机模式（中断方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
```

```
sfr         P3M0        =   0xb2;
sfr         P4M1        =   0xb3;
sfr         P4M0        =   0xb4;
sfr         P5M1        =   0xc9;
sfr         P5M0        =   0xca;

sbit        SDA         =   P1^4;
sbit        SCL         =   P1^5;

bit         isda;                               //设备地址标志
bit         isma;                               //存储地址标志
unsigned char           addr;
unsigned char pdata     buffer[256];

void I2C_Isr() interrupt 24
{
    _push_(P_SW2);
    P_SW2 |= 0x80;

    if (I2CSLST & 0x40)
    {
        I2CSLST &= ~0x40;                       //处理 START 事件
        isda = 1;                               //若为重复起始信号时必须作此设置
    }
    else if (I2CSLST & 0x20)
    {
        I2CSLST &= ~0x20;                       //处理 RECV 事件
        if (isda)
        {
            isda = 0;                           //处理 RECV 事件（RECV DEVICE ADDR）
        }
        else if (isma)
        {
            isma = 0;                           //处理 RECV 事件（RECV MEMORY ADDR）
            addr = I2CRXD;
            I2CTXD = buffer[addr];
        }
        else
        {
            buffer[addr++] = I2CRXD;            //处理 RECV 事件（RECV DATA）
        }
    }
    else if (I2CSLST & 0x10)
    {
        I2CSLST &= ~0x10;                       //处理 SEND 事件
        if (I2CSLST & 0x02)
        {
            I2CTXD = 0xff;                      //接收到 NAK 则停止读取数据
        }
        else
        {
            I2CTXD = buffer[++addr];            //接收到 ACK 则继续读取数据
        }
    }
    else if (I2CSLST & 0x08)
    {
        I2CSLST &= ~0x08;                       //处理 STOP 事件
        isda = 1;
        isma = 1;
```

STC MCU

```
    }

    _pop_(P_SW2);
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0x81;                  //使能 I2C 从机模式
    I2CSLADR = 0x5a;                //设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                    //即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                    //由于 MA 为 0,主机发送的设备地址必须与
                                    //I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                    //主机若需要写数据则要发送 5AH(0101_1010B)
                                    //主机若需要读数据则要发送 5BH(0101_1011B)
    I2CSLST = 0x00;
    I2CSLCR = 0x78;                 //使能从机模式中断
    EA = 1;

    isda = 1;                       //用户变量初始化
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA        0BAH

I2CCFG      XDATA       0FE80H
I2CMSCR     XDATA       0FE81H
I2CMSST     XDATA       0FE82H
I2CSLCR     XDATA       0FE83H
I2CSLST     XDATA       0FE84H
I2CSLADR    XDATA       0FE85H
I2CTXD      XDATA       0FE86H
I2CRXD      XDATA       0FE87H

SDA         BIT         P1.4
SCL         BIT         P1.5
ISDA        BIT         20H.0           ;设备地址标志
```

```
ISMA        BIT     20H.1           ;存储地址标志

ADDR        DATA    21H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN
            ORG     00C3H
            LJMP    I2CISR

            ORG     0100H
I2CISR:
            PUSH    ACC
            PUSH    PSW
            PUSH    DPL
            PUSH    DPH
            MOV     DPTR,#I2CSLST   ;检测从机状态
            MOVX    A,@DPTR
            JB      ACC.6,STARTIF
            JB      ACC.5,RXIF
            JB      ACC.4,TXIF
            JB      ACC.3,STOPIF
ISREXIT:
            POP     DPH
            POP     DPL
            POP     PSW
            POP     ACC
            RETI
STARTIF:
            ANL     A,#NOT 40H      ;处理 START 事件
            MOVX    @DPTR,A
            SETB    ISDA
            JMP     ISREXIT
RXIF:
            ANL     A,#NOT 20H      ;处理 RECV 事件
            MOVX    @DPTR,A
            MOV     DPTR,#I2CRXD
            MOVX    A,@DPTR
            JBC     ISDA,RXDA
            JBC     ISMA,RXMA
            MOV     R0,ADDR         ;处理 RECV 事件（RECV DATA）
            MOVX    @R0,A
            INC     ADDR
            JMP     ISREXIT
RXDA:
            JMP     ISREXIT         ;处理 RECV 事件（RECV DEVICE ADDR）
RXMA:
```

```
                MOV     ADDR,A          ;处理 RECV 事件（RECV MEMORY ADDR）
                MOV     R0,A
                MOVX    A,@R0
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     ISREXIT
TXIF:
                ANL     A,#NOT 10H      ;处理 SEND 事件
                MOVX    @DPTR,A
                JB      ACC.1,RXNAK
                INC     ADDR
                MOV     R0,ADDR
                MOVX    A,@R0
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     ISREXIT
RXNAK:
                MOVX    A,#0FFH
                MOV     DPTR,#I2CTXD
                MOVX    @DPTR,A
                JMP     ISREXIT
STOPIF:
                ANL     A,#NOT 08H      ;处理 STOP 事件
                MOVX    @DPTR,A
                SETB    ISDA
                SETB    ISMA
                JMP     ISREXIT

MAIN:
                MOV     SP, #5FH
                MOV     P0M0, #00H
                MOV     P0M1, #00H
                MOV     P1M0, #00H
                MOV     P1M1, #00H
                MOV     P2M0, #00H
                MOV     P2M1, #00H
                MOV     P3M0, #00H
                MOV     P3M1, #00H
                MOV     P4M0, #00H
                MOV     P4M1, #00H
                MOV     P5M0, #00H
                MOV     P5M1, #00H

                MOV     P_SW2,#80H

                MOV     A,#10000001B    ;使能 I2C 从机模式
                MOV     DPTR,#I2CCFG
                MOVX    @DPTR,A
                MOV     A,#01011010B    ;设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                        ;即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                        ;由于 MA 为 0,主机发送的设备地址必须与
                                        ;I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                        ;主机若需要写数据则要发送 5AH(0101_1010B)
                                        ;主机若需要读数据则要发送 5BH(0101_1011B)
                MOV     DPTR,#I2CSLADR
                MOVX    @DPTR,A
                MOV     A,#00000000B
                MOV     DPTR,#I2CSLST
                MOVX    @DPTR,A
```

```
        MOV     A,#01111000B            ;使能从机模式中断
        MOV     DPTR,#I2CSLCR
        MOVX    @DPTR,A

        SETB    ISDA                    ;用户变量初始化
        SETB    ISMA
        CLR     A
        MOV     ADDR,A
        MOV     R0,A
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A

        SETB    EA

        SJMP    $

        END
```

18.4.5　I²C 从机模式（查询方式）

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr     P_SW2       =   0xba;

#define I2CCFG      (*(unsigned char volatile xdata *)0xfe80)
#define I2CMSCR     (*(unsigned char volatile xdata *)0xfe81)
#define I2CMSST     (*(unsigned char volatile xdata *)0xfe82)
#define I2CSLCR     (*(unsigned char volatile xdata *)0xfe83)
#define I2CSLST     (*(unsigned char volatile xdata *)0xfe84)
#define I2CSLADR    (*(unsigned char volatile xdata *)0xfe85)
#define I2CTXD      (*(unsigned char volatile xdata *)0xfe86)
#define I2CRXD      (*(unsigned char volatile xdata *)0xfe87)

sfr     P1M1        =   0x91;
sfr     P1M0        =   0x92;
sfr     P0M1        =   0x93;
sfr     P0M0        =   0x94;
sfr     P2M1        =   0x95;
sfr     P2M0        =   0x96;
sfr     P3M1        =   0xb1;
sfr     P3M0        =   0xb2;
sfr     P4M1        =   0xb3;
sfr     P4M0        =   0xb4;
sfr     P5M1        =   0xc9;
sfr     P5M0        =   0xca;

sbit    SDA         =   P1^4;
sbit    SCL         =   P1^5;

bit     isda;                           //设备地址标志
```

```
bit             isma;                                   //存储地址标志
unsigned char           addr;
unsigned char pdata     buffer[256];

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0x81;                                      //使能 I2C 从机模式
    I2CSLADR = 0x5a;                                    //设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                                        //即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                                        //由于 MA 为 0,主机发送的设备地址必须与
                                                        //I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                                        //主机若需要写数据则要发送 5AH(0101_1010B)
                                                        //主机若需要读数据则要发送 5BH(0101_1011B)
    I2CSLST = 0x00;
    I2CSLCR = 0x00;                                     //禁止从机模式中断

    isda = 1;                                           //用户变量初始化
    isma = 1;
    addr = 0;
    I2CTXD = buffer[addr];

    while (1)
    {
        if (I2CSLST & 0x40)
        {
            I2CSLST &= ~0x40;                           //处理 START 事件
            isda = 1;                                   //若为重复起始信号时必须作此设置
        }
        else if (I2CSLST & 0x20)
        {
            I2CSLST &= ~0x20;                           //处理 RECV 事件
            if (isda)
            {
                isda = 0;                               //处理 RECV 事件（RECV DEVICE ADDR）
            }
            else if (isma)
            {
                isma = 0;                               //处理 RECV 事件（RECV MEMORY ADDR）
                addr = I2CRXD;
                I2CTXD = buffer[addr];
            }
            else
            {
                buffer[addr++] = I2CRXD;                //处理 RECV 事件（RECV DATA）
```

```
            }
        }
        else if (I2CSLST & 0x10)
        {
            I2CSLST &= ~0x10;                   //处理 SEND 事件
            if (I2CSLST & 0x02)
            {
                I2CTXD = 0xff;                  //接收到 NAK 则停止读取数据
            }
            else
            {
                I2CTXD = buffer[++addr];        //接收到 ACK 则继续读取数据
            }
        }
        else if (I2CSLST & 0x08)
        {
            I2CSLST &= ~0x08;                   //处理 STOP 事件
            isda = 1;
            isma = 1;
        }
    }
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5
ISDA        BIT     20H.0               ;设备地址标志
ISMA        BIT     20H.1               ;存储地址标志

ADDR        DATA    21H

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
```

```
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H

        MOV     A,#10000001B            ;使能 I2C 从机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#01011010B            ;设置从机设备地址寄存器 I2CSLADR=0101_1010B
                                        ;即 I2CSLADR[7:1]=010_1101B,MA=0B。
                                        ;由于 MA 为 0,主机发送的设备地址必须与
                                        ;I2CSLADR[7:1]相同才能访问此 I2C 从机设备。
                                        ;主机若需要写数据则要发送 5AH(0101_1010B)
                                        ;主机若需要读数据则要发送 5BH(0101_1011B)
        MOV     DPTR,#I2CSLADR
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CSLST
        MOVX    @DPTR,A
        MOV     A,#00000000B            ;禁止从机模式中断
        MOV     DPTR,#I2CSLCR
        MOVX    @DPTR,A

        SETB    ISDA                    ;用户变量初始化
        SETB    ISMA
        CLR     A
        MOV     ADDR,A
        MOV     R0,A
        MOVX    A,@R0
        MOV     DPTR,#I2CTXD
        MOVX    @DPTR,A

LOOP:
        MOV     DPTR,#I2CSLST           ;检测从机状态
        MOVX    A,@DPTR
        JB      ACC.6,STARTIF
        JB      ACC.5,RXIF
        JB      ACC.4,TXIF
        JB      ACC.3,STOPIF
        JMP     LOOP
STARTIF:
        ANL     A,#NOT 40H              ;处理 START 事件
        MOVX    @DPTR,A
        SETB    ISDA
```

```
            JMP     LOOP
RXIF:
            ANL     A,#NOT 20H          ;处理 RECV 事件
            MOVX    @DPTR,A
            MOV     DPTR,#I2CRXD
            MOVX    A,@DPTR
            JBC     ISDA,RXDA
            JBC     ISMA,RXMA
            MOV     R0,ADDR             ;处理 RECV 事件（RECV DATA）
            MOVX    @R0,A
            INC     ADDR
            JMP     LOOP
RXDA:
            JMP     LOOP                ;处理 RECV 事件（RECV DEVICE ADDR）
RXMA:
            MOV     ADDR,A              ;处理 RECV 事件（RECV MEMORY ADDR）
            MOV     R0,A
            MOVX    A,@R0
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     LOOP
TXIF:
            ANL     A,#NOT 10H          ;处理 SEND 事件
            MOVX    @DPTR,A
            JB      ACC.1,RXNAK
            INC     ADDR
            MOV     R0,ADDR
            MOVX    A,@R0
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     LOOP
RXNAK:
            MOVX    A,#0FFH
            MOV     DPTR,#I2CTXD
            MOVX    @DPTR,A
            JMP     LOOP
STOPIF:
            ANL     A,#NOT 08H          ;处理 STOP 事件
            MOVX    @DPTR,A
            SETB    ISDA
            SETB    ISMA
            JMP     LOOP

            END
```

18.4.6 测试 I²C 从机模式代码的主机代码

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         P_SW2           =     0xba;
```

```
#define   I2CCFG          (*(unsigned char volatile xdata *)0xfe80)
#define   I2CMSCR         (*(unsigned char volatile xdata *)0xfe81)
#define   I2CMSST         (*(unsigned char volatile xdata *)0xfe82)
#define   I2CSLCR         (*(unsigned char volatile xdata *)0xfe83)
#define   I2CSLST         (*(unsigned char volatile xdata *)0xfe84)
#define   I2CSLADR        (*(unsigned char volatile xdata *)0xfe85)
#define   I2CTXD          (*(unsigned char volatile xdata *)0xfe86)
#define   I2CRXD          (*(unsigned char volatile xdata *)0xfe87)

sfr       P1M1      =     0x91;
sfr       P1M0      =     0x92;
sfr       P0M1      =     0x93;
sfr       P0M0      =     0x94;
sfr       P2M1      =     0x95;
sfr       P2M0      =     0x96;
sfr       P3M1      =     0xb1;
sfr       P3M0      =     0xb2;
sfr       P4M1      =     0xb3;
sfr       P4M0      =     0xb4;
sfr       P5M1      =     0xc9;
sfr       P5M0      =     0xca;

sbit      SDA       =     P1^4;
sbit      SCL       =     P1^5;

void Wait()
{
    while (!(I2CMSST & 0x40));
    I2CMSST &= ~0x40;
}

void Start()
{
    I2CMSCR = 0x01;                         //发送 START 命令
    Wait();
}

void SendData(char dat)
{
    I2CTXD = dat;                           //写数据到数据缓冲区
    I2CMSCR = 0x02;                         //发送 SEND 命令
    Wait();
}

void RecvACK()
{
    I2CMSCR = 0x03;                         //发送读 ACK 命令
    Wait();
}

char RecvData()
{
    I2CMSCR = 0x04;                         //发送 RECV 命令
    Wait();
    return I2CRXD;
}

void SendACK()
{
```

STC MCU

```
    I2CMSST = 0x00;                         //设置 ACK 信号
    I2CMSCR = 0x05;                         //发送 ACK 命令
    Wait();
}

void SendNAK()
{
    I2CMSST = 0x01;                         //设置 NAK 信号
    I2CMSCR = 0x05;                         //发送 ACK 命令
    Wait();
}

void Stop()
{
    I2CMSCR = 0x06;                         //发送 STOP 命令
    Wait();
}

void Delay()
{
    int i;

    for (i=0; i<3000; i++)
    {
        _nop_();
        _nop_();
        _nop_();
        _nop_();
    }
}

void main()
{
    P0M0 = 0x00;
    P0M1 = 0x00;
    P1M0 = 0x00;
    P1M1 = 0x00;
    P2M0 = 0x00;
    P2M1 = 0x00;
    P3M0 = 0x00;
    P3M1 = 0x00;
    P4M0 = 0x00;
    P4M1 = 0x00;
    P5M0 = 0x00;
    P5M1 = 0x00;

    P_SW2 = 0x80;

    I2CCFG = 0xe0;                          //使能 I2C 主机模式
    I2CMSST = 0x00;

    Start();                                //发送起始命令
    SendData(0x5a);                         //发送设备地址(010_1101B)+写命令(0B)
    RecvACK();
    SendData(0x00);                         //发送存储地址
    RecvACK();
    SendData(0x12);                         //写测试数据 1
    RecvACK();
    SendData(0x78);                         //写测试数据 2
```

STC MCU

```
    RecvACK();
    Stop();                                 //发送停止命令

    Start();                                //发送起始命令
    SendData(0x5a);                         //发送设备地址(010_1101B)+写命令(0B)
    RecvACK();
    SendData(0x00);                         //发送存储地址高字节
    RecvACK();
    Start();                                //发送起始命令
    SendData(0x5b);                         //发送设备地址(010_1101B)+读命令(1B)
    RecvACK();
    P0 = RecvData();                        //读取数据 1
    SendACK();
    P2 = RecvData();                        //读取数据 2
    SendNAK();
    Stop();                                 //发送停止命令

    P_SW2 = 0x00;

    while (1);
}
```

汇编代码

```
;测试工作频率为 11.0592MHz

P_SW2       DATA    0BAH

I2CCFG      XDATA   0FE80H
I2CMSCR     XDATA   0FE81H
I2CMSST     XDATA   0FE82H
I2CSLCR     XDATA   0FE83H
I2CSLST     XDATA   0FE84H
I2CSLADR    XDATA   0FE85H
I2CTXD      XDATA   0FE86H
I2CRXD      XDATA   0FE87H

SDA         BIT     P1.4
SCL         BIT     P1.5

P1M1        DATA    091H
P1M0        DATA    092H
P0M1        DATA    093H
P0M0        DATA    094H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
START:
            MOV     A,#00000001B        ;发送 START 命令
```

```
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
SENDDATA:
                MOV     DPTR,#I2CTXD        ;写数据到数据缓冲区
                MOVX    @DPTR,A
                MOV     A,#00000010B        ;发送 SEND 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
RECVACK:
                MOV     A,#00000011B        ;发送读 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
RECVDATA:
                MOV     A,#00000100B        ;发送 RECV 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                CALL    WAIT
                MOV     DPTR,#I2CRXD        ;从数据缓冲区读取数据
                MOVX    A,@DPTR
                RET
SENDACK:
                MOV     A,#00000000B        ;设置 ACK 信号
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A
                MOV     A,#00000101B        ;发送 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
SENDNAK:
                MOV     A,#00000001B        ;设置 NAK 信号
                MOV     DPTR,#I2CMSST
                MOVX    @DPTR,A
                MOV     A,#00000101B        ;发送 ACK 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
STOP:
                MOV     A,#00000110B        ;发送 STOP 命令
                MOV     DPTR,#I2CMSCR
                MOVX    @DPTR,A
                JMP     WAIT
WAIT:
                MOV     DPTR,#I2CMSST       ;清中断标志
                MOVX    A,@DPTR
                JNB     ACC.6,WAIT
                ANL     A,#NOT 40H
                MOVX    @DPTR,A
                RET

DELAY:
                MOV     R0,#0
                MOV     R1,#0
DELAY1:
                NOP
                NOP
                NOP
```

```
        NOP
        DJNZ    R1,DELAY1
        DJNZ    R0,DELAY1
        RET

MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        MOV     P_SW2,#80H

        MOV     A,#11100000B        ;设置 I2C 模块为主机模式
        MOV     DPTR,#I2CCFG
        MOVX    @DPTR,A
        MOV     A,#00000000B
        MOV     DPTR,#I2CMSST
        MOVX    @DPTR,A

        CALL    START               ;发送起始命令
        MOV     A,#5AH              ;
        CALL    SENDDATA            ;发送设备地址(010_1101B)+写命令(0B)
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#12H              ;写测试数据 1
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#78H              ;写测试数据 2
        CALL    SENDDATA
        CALL    RECVACK
        CALL    STOP                ;发送停止命令

        CALL    DELAY               ;等待设备写数据

        CALL    START               ;发送起始命令
        MOV     A,#5AH              ;发送设备地址(010_1101B)+写命令(0B)
        CALL    SENDDATA
        CALL    RECVACK
        MOV     A,#000H             ;发送存储地址
        CALL    SENDDATA
        CALL    RECVACK
        CALL    START               ;发送起始命令
        MOV     A,#5BH              ;发送设备地址(010_1101B)+读命令(1B)
        CALL    SENDDATA
        CALL    RECVACK
        CALL    RECVDATA            ;读取数据 1
        MOV     P0,A
```

```
        CALL    SENDACK
        CALL    RECVDATA            ;读取数据 2
        MOV     P2,A
        CALL    SENDNAK
        CALL    STOP                ;发送停止命令

        JMP     $

        END
```

STC MCU

19 增强型双数据指针

STC8C 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

19.1 相关的特殊功能寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
DPL	数据指针（低字节）	82H									0000,0000
DPH	数据指针（高字节）	83H									0000,0000
DPL1	第二组数据指针（低字节）	E4H									0000,0000
DPH1	第二组数据指针（高字节）	E5H									0000,0000
DPS	DPTR 指针选择器	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL	0000,0xx0
TA	DPTR 时序控制寄存器	AEH									0000,0000

19.1.1 第 1 组 16 位数据指针寄存器（DPTR0）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPL	82H								
DPH	83H								

DPL为低8位数据（低字节）

DPH为高8位数据（高字节）

DPL和DPH组合为第一组16位数据指针寄存器DPTR0

19.1.2 第 2 组 16 位数据指针寄存器（DPTR1）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPL1	E4H								
DPH1	E5H								

DPL1为低8位数据（低字节）

DPH1为高8位数据（高字节）

DPL1和DPH1组合为第二组16位数据指针寄存器DPTR1

STC MCU

19.1.3 数据指针控制寄存器（DPS）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
DPS	E3H	ID1	ID0	TSL	AU1	AU0	-	-	SEL

ID1：控制DPTR1自动递增方式

0：DPTR1 自动递增

1：DPTR1 自动递减

ID0：控制DPTR0自动递增方式

0：DPTR0 自动递增

1：DPTR0 自动递减

TSL：DPTR0/DPTR1自动切换控制（自动对SEL进行取反）

0：关闭自动切换功能

1：使能自动切换功能

当 TSL 位被置 1 后，每当执行完成相关指令后，系统会自动将 SEL 位取反。

与 TSL 相关的指令包括如下指令：

```
MOV     DPTR,#data16
INC     DPTR
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
```

AU1/AU0：使能DPTR1/DPTR0使用ID1/ID0控制位进行自动递增/递减控制

0：关闭自动递增/递减功能

1：使能自动递增/递减功能

注意：在写保护模式下，AU0 和 AU1 位无法直接单独使能，若单独使能 AU1 位，则 AU0 位也会被自动使能，若单独使能 AU0，没有效果。若需要单独使能 AU1 或者 AU0，则必须使用 TA 寄存器触发 DPS 的保护机制（参考 TA 寄存器的说明）。另外，只有执行下面的 3 条指令后才会对 DPTR0/DPTR1 进行自动递增/递减操作。3 条相关指令如下：

```
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
```

SEL：选择DPTR0/DPTR1作为当前的目标DPTR

0：选择 DPTR0 作为目标 DPTR

1：选择 DPTR1 作为目标 DPTR

SEL 选择目标 DPTR 对下面指令有效：

```
MOV     DPTR,#data16
INC     DPTR
MOVC    A,@A+DPTR
MOVX    A,@DPTR
MOVX    @DPTR,A
JMP     @A+DPTR
```

19.1.4 数据指针控制寄存器（TA）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
TA	AEH								

TA寄存器是对DPS寄存器中的AU1和AU0进行写保护的。由于程序无法对DPS中的AU1和AU0进行单独的写入，所以当需要单独使能AU1或者AU0时，必须使用TA寄存器进行触发。TA寄存器是只写寄存器。当需要对AU1或者AU0进行单独使能时，必须按照如下的步骤进行操作：

```
CLR     EA          ;关闭中断（必需）
MOV     TA,#0AAH    ;写入触发命令序列 1
                    ;此处不能有其他任何指令
MOV     TA,#55H     ;写入触发命令序列 2
                    ;此处不能有其他任何指令
MOV     DPS,#xxH    ;写保护暂时关闭，可向 DPS 中写入任何值
                    ;DSP 再次进行写保护状态
SETB    EA          ;打开中断（如有必要）
```

STC MCU

19.2 范例程序

19.2.1 示例代码 1

将程序空间 1000H～1003H 的 4 个字节数据反向复制到扩展 RAM 的 0100H～0103H 中，即

C:1000H -> X:0103H
C:1001H -> X:0102H
C:1002H -> X:0101H
C:1003H -> X:0100H

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1        DATA    093H
P0M0        DATA    094H
P1M1        DATA    091H
P1M0        DATA    092H
P2M1        DATA    095H
P2M0        DATA    096H
P3M1        DATA    0B1H
P3M0        DATA    0B2H
P4M1        DATA    0B3H
P4M0        DATA    0B4H
P5M1        DATA    0C9H
P5M0        DATA    0CAH

            ORG     0000H
            LJMP    MAIN

            ORG     0100H
MAIN:
            MOV     SP, #5FH
            MOV     P0M0, #00H
            MOV     P0M1, #00H
            MOV     P1M0, #00H
            MOV     P1M1, #00H
            MOV     P2M0, #00H
            MOV     P2M1, #00H
            MOV     P3M0, #00H
            MOV     P3M1, #00H
            MOV     P4M0, #00H
            MOV     P4M1, #00H
            MOV     P5M0, #00H
            MOV     P5M1, #00H

            MOV     DPS,#00100000B      ;使能 TSL,并选择 DPTR0
            MOV     DPTR,#1000H         ;将 1000H 写入 DPTR0 后选择 DPTR1 为 DPTR
            MOV     DPTR,#0103H         ;将 0103H 写入 DPTR1 中
            MOV     DPS,#10111000B      ;设置 DPTR1 为递减模式,DPTR0 为递加模式,使能 TSL
                                        ;AU0 和 AU1,并选择 DPTR0 为当前的 DPTR
            MOV     R7,#4               ;设置数据复制个数
COPY_NEXT:
            CLR     A                   ;
            MOVC    A,@A+DPTR           ;从 DPTR0 所指的程序空间读取数据,
                                        ;完成后 DPTR0 自动加 1 并将 DPTR1 设置为 DPTR
            MOVX    @DPTR,A             ;将 ACC 的数据写入到 DPTR1 所指的 XDATA 中,
                                        ;完成后 DPTR1 自动减 1 并将 DPTR0 设置为 DPTR
```

```
        DJNZ    R7,COPY_NEXT        ;

        SJMP    $

        END
```

19.2.2 示例代码 2

将扩展 RAM 的 0100H～0103H 中的数据依次发送到 P0 口

汇编代码

```
;测试工作频率为 11.0592MHz

P0M1    DATA    093H
P0M0    DATA    094H
P1M1    DATA    091H
P1M0    DATA    092H
P2M1    DATA    095H
P2M0    DATA    096H
P3M1    DATA    0B1H
P3M0    DATA    0B2H
P4M1    DATA    0B3H
P4M0    DATA    0B4H
P5M1    DATA    0C9H
P5M0    DATA    0CAH

        ORG     0000H
        LJMP    MAIN

        ORG     0100H
MAIN:
        MOV     SP, #5FH
        MOV     P0M0, #00H
        MOV     P0M1, #00H
        MOV     P1M0, #00H
        MOV     P1M1, #00H
        MOV     P2M0, #00H
        MOV     P2M1, #00H
        MOV     P3M0, #00H
        MOV     P3M1, #00H
        MOV     P4M0, #00H
        MOV     P4M1, #00H
        MOV     P5M0, #00H
        MOV     P5M1, #00H

        CLR     EA              ;关闭中断
        MOV     TA,#0AAH        ;写入 DPS 写保护触发命令 1
        MOV     TA,#55H         ;写入 DPS 写保护触发命令 2
        MOV     DPS,#00001000B  ;DPTR0 递增,单独使能 AU0,并选择 DPTR0
        SETB    EA              ;打开中断
        MOV     DPTR,#0100H     ;将 0100H 写入 DPTR0 中
        MOVX    A,@DPTR         ;从 DPTR0 所指的 XRAM 读取数据后 DPTR0 自动加 1
        MOV     P0,A            ;数据输出到 P0 口
        MOVX    A,@DPTR         ;从 DPTR0 所指的 XRAM 读取数据后 DPTR0 自动加 1
        MOV     P0,A            ;数据输出到 P0 口
        MOVX    A,@DPTR         ;从 DPTR0 所指的 XRAM 读取数据后 DPTR0 自动加 1
        MOV     P0,A            ;数据输出到 P0 口
        MOVX    A,@DPTR         ;从 DPTR0 所指的 XRAM 读取数据后 DPTR0 自动加 1
        MOV     P0,A            ;数据输出到 P0 口
```

```
        SJMP    $

        END
```

STC MCU

20 MDU16 硬件 16 位乘除法器

STC8C 系列型号的单片机内部集成了 MDU16/16 位硬件乘除法器。

支持如下数据运算：

- 数据规格化（需要 3～20 个时钟的运算时间）
- 逻辑左移（需要 3～18 个时钟的运算时间）
- 逻辑右移（需要 3～18 个时钟的运算时间）
- 16 位乘以 16 位（需要 10 个时钟的运算时间）
- 16 位除以 16 位（需要 9 个时钟的运算时间）
- 32 位除以 16 位（需要 17 个时钟的运算时间）

所有的操作都是基于无符号整形数据类型。

20.1 相关的特殊功能寄存器

符号	描述	地址	位地址与符号								复位值
			B7	B6	B5	B4	B3	B2	B1	B0	
MD3	MDU 数据寄存器	FCF0H	MD3[7:0]								0000,0000
MD2	MDU 数据寄存器	FCF1H	MD2[7:0]								0000,0000
MD1	MDU 数据寄存器	FCF2H	MD1[7:0]								0000,0000
MD0	MDU 数据寄存器	FCF3H	MD0[7:0]								0000,0000
MD5	MDU 数据寄存器	FCF4H	MD5[7:0]								0000,0000
MD4	MDU 数据寄存器	FCF5H	MD4[7:0]								0000,0000
ARCON	MDU 模式控制寄存器	FCF6H	MODE[2:0]			SC[4:0]					0000,0000
OPCON	MDU 操作控制寄存器	FCF7H	-	MDOV	-	-	-	-	RST	ENOP	0000,0000

STCMCU

20.1.1 操作数 1 数据寄存器（MD0～MD3）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MD3	FCF0H	MD3[7:0]							
MD2	FCF1H	MD2[7:0]							
MD1	FCF2H	MD1[7:0]							
MD0	FCF3H	MD0[7:0]							

20.1.2 操作数 2 数据寄存器（MD4～MD5）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
MD5	FCF4H	MD5[7:0]							
MD4	FCF5H	MD4[7:0]							

32位除以16位除法：

被除数：{MD3,MD2,MD1,MD0}

除数：{MD5,MD4}

商：{MD3,MD2,MD1,MD0}

余数：{MD5,MD4}

16位除以16位除法：

被除数：{MD1,MD0}

除数：{MD5,MD4}

商：{MD1,MD0}

余数：{MD5,MD4}

16位乘以16位乘法：

被乘数：{MD1,MD0}

乘数：{MD5,MD4}

积：{MD3,MD2,MD1,MD0}

32 位逻辑左移/逻辑右移

操作数：{MD3,MD2,MD1,MD0}

32 位数据规格化：

操作数：{MD3,MD2,MD1,MD0}

STC MCU

20.1.3 MDU 模式控制寄存器（ARCON），运算所需时钟数

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
ARCON	FCF6H	MODE[2:0]			SC[4:0]				

MODE[2:0]：MDU模式选择

MODE[2:0]	模式	时钟数	操作说明
1	逻辑右移	3～18	将{MD3,MD2,MD1,MD0}中的数据右移SC[4:0]位，MD3的高位补0
2	逻辑左移	3～18	将{MD3,MD2,MD1,MD0}中的数据左移SC[4:0]位，MD0的低位补0
3	数据规格化	3～20	对{MD3,MD2,MD1,MD0}中的数据进行逻辑左移，将数据高位的0全部移出，使MD3的最高位为1，逻辑左移的位数被记录在SC[4:0]中
4	16位×16位	10	{MD1,MD0}×{MD5,MD4}＝{MD3,MD2,MD1,MD0}
5	16位÷16位	9	{MD1,MD0}÷{MD5,MD4}＝{MD1,MD0}…{MD5,MD4}
6	32位÷16位	17	{MD3,MD2,MD1,MD0}÷{MD5,MD4}＝{MD3,MD2,MD1,MD0}…{MD5,MD4}
其他	无效		

SC[4:0]：数据移动位数

当 MDU 为移动模式时，SC 用于设置左移/右移的位数

当 MDU 为数据规格化模式时，SC 为数据规格化后数据所移动的实际位数

20.1.4 MDU 操作控制寄存器（OPCON）

符号	地址	B7	B6	B5	B4	B3	B2	B1	B0
OPCON	FCF7H	-	MDOV	-	-	-	-	RST	ENOP

MDOV：MDU溢出标志位（只读标志位）

在如下几种情况时，MDOV 会被硬件自动置 1：

1、除数为 0 时；

2、乘法的积大于 0FFFFH 时；

当软件写 OPCON.0（EN）或者写 ARCON 时，硬件会自动清除 MDOV

RST：软件复位 MDU 乘除单元。写 1 触发软件复位，MDU 复位完成后硬件自动清零。

注：软件复位 MDU 乘除单元时，ARCON 寄存器的值会被清除。

ENOP：MDU 模块使能。写 1 触发 MDU 模块开始计算，当 MDU 计算完成后，硬件自动将 ENOP 清零。

软件可以在对 ENOP 置 1 后，循环的查询 ENOP，当 ENOP 由 1 变 0 则表示计算完成。

20.2 关于 MDU16 的网友应用杂谈（提供思路，仅供参考）

网友 1：“数据规格化用下面的一个简单例子说明”

1、 有一个 7 位小数精度的数据：0.0000123，由于数据位宽有限，如果需要有效利用位宽，就需要把前面的数据左移，比如左移后数据为 0.123e-4，其中指数-4 保存在另一个寄存器，记录左移的次数就是记录指数的大小。原来寄存器数据转换为 0.123。这样就把数据右边的位宽腾出来，可以保证后续计算的精度。上面只是用十进制简单的说明规格化原理，二进制原理也是一样的。其中浮点和定点（整数）转换就必须使用规格化的原理，如果两个浮点数相加减时的指数不一样，也需要进行规格化处理（这个过程叫作对阶）。如果两个浮点数的指数相差非常大，相加减时就会出现大数吃小数的问题。比如：0.123e+4 - 0.12e-4 = 0.123e+4 - 0.0000000012e+4 = 0.123e+4.结果就是被减数，这是因为在减操作前，两个浮点数的指数需要完全一致（对阶），需要把指数小的浮点数进行移位，使指数变为+4.但是数据宽度是有限的 7 位小数精度，0.0000000012e+4 这个数右边的数据会被截断变为 0.0000000e+4 = 0.

网友 2：“关于 STC8C 的 MDU 功能，我分享一点自己的体会，有不对的请大家批评指教，共同提高。”

1、 功能 1 和 2 对于缩减和扩展整数数据很有效。首先在进行双操作数运算时，如果两个数的长度不一样，需要转换为相同的长度进行才进行运算。比如 32 位整数乘 8 位整数，就要将 8 位转换为 32 位。其次对 AD 采样的结果，转换为指定的位数精度时也需要位移。最后，比如对网络通讯，需要提取数据的某几位出来进行命令解析或者数据分解合成，位移都是很重要的。由于 8051 只有移动 1 位的指令，多位移动需要借助额外的循环代码，需要很多个指令周期，因此使用 MDU 将比 51 汇编指令快数倍。

2、 功能 3 是整数转换为浮点数必须的功能。对于满精度的 32 位整数，实现这个功能一般要超过 100 个指令周期，因此 MDU 对转速度的提升是比较大的。由于像 AD 设备输出、像各种三轴加速度输出，一般都是整数的（比如 16 位的），要进行实数运算，要进行三角函数运算，整数的输出必须要转换为浮点数，而且每次采集数据都要进行这数据类型转换，需要转换的次数就很多了。对于高速数据采集和像无人机控制这样的应用，如果采用 DMU 对整体性能的提高就很可观了。

3、 功能 6 是定点实数运算必须的除法功能，功能 4 是功能 6 对应的 16 位 x16 位结果为 32 位的乘法运算。功能 6 的最常见应用是数据处理中的标度转换，比如对于将参考电压为 5 伏的 10 位 AD 采集的整数转换 3 位数码管的 2 位固定小数点进行显示的运算公式为：N32=ADN*500/1023。这时只要（1）将 AD 采样值 AND 送 MX（DM1MD0），（2）送 500 到 NX（MD5MD4），（3）执行功能 4，结果是 32 位的了，（4）送 1023 到 NX（MD5MD4），（3）执行功能 6，16 位的结果就在 MX 中了，取回来就行了。另一个常见的应用就是在 TFT 之类的点阵屏上画点和线，比如数字示波器，这些都需要进行坐标变换的乘除法-先乘为 32 位整数，再除以 16 位整数得到 16 位结果。

4、 功能 4 和功能 6 的组合是实现离散卷积的硬件基础。如果不采用浮点加速硬件，实现浮点数的四则运算比实现整数的四则运算要慢一个量级，因此前辈们发明了用整数变量来实现卷积的方法。首先比如我们常见的将 JPG 图像数据转换为 RGB 图像数据或者相反，就需要进行傅里叶变换，由于图像数据的长度是固定的（8 位或者 16 位），因此就可以用离散傅里叶变换来实现，其中基本只用到 8 位或者 16 位的整数乘法和极少量的 32 位乘除法。这样，我们早期的数码相机才有可能实现。其次 PS 图像处理中常见的各种模板处理，也使用的是二维矩阵卷积方法，也是需要巨量的对整数的（8 位图像视图像大小需要 16 位和 32 位的中间计算结果）乘加计算，使用离散卷积将极高的提高运算速度。因此有 MDU 的 STC8 单片机不仅可以用于实时采集和显示图像，也可以实时处理图像。最后人工智能也涉及大量的矢量和矩阵运算，比如神经网络卷积，这些都可以用功能 4 和功能 6 的组合实现，MDU 应该可以在小型智能化场景中得到应用。只是要实现这

些功能，需要 STC8 的增强型双数据指针的配合，需要专门的知识结构，专门编制出函数库来提供给用户使用，才能发挥 STC8 的 MDU 巨大优势。

STC MCU

20.3 范例程序

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define   MD3U32          (*(unsigned long volatile xdata *)0xfcf0)
#define   MD3U16          (*(unsigned int volatile xdata *)0xfcf0)
#define   MD1U16          (*(unsigned int volatile xdata *)0xfcf2)
#define   MD5U16          (*(unsigned int volatile xdata *)0xfcf4)

#define   MD3             (*(unsigned char volatile xdata *)0xfcf0)
#define   MD2             (*(unsigned char volatile xdata *)0xfcf1)
#define   MD1             (*(unsigned char volatile xdata *)0xfcf2)
#define   MD0             (*(unsigned char volatile xdata *)0xfcf3)
#define   MD5             (*(unsigned char volatile xdata *)0xfcf4)
#define   MD4             (*(unsigned char volatile xdata *)0xfcf5)
#define   ARCON           (*(unsigned char volatile xdata *)0xfcf6)
#define   OPCON           (*(unsigned char volatile xdata *)0xfcf7)

sfr       P_SW2       =   0xBA;

//////////////////////////////////////////////////////////////////////////
//16 位乘 16 位
//////////////////////////////////////////////////////////////////////////
unsigned long res;
unsigned int dat1, dat2;

P_SW2 |= 0x80;                  //访问扩展寄存器 xsfr
MD1U16 = dat1;                  //dat1 用户给定
MD5U16 = dat2;                  //dat2 用户给定
ARCON = 4 << 5;                 //16 位*16 位,乘法模式
OPCON = 1;                      //启动计算
while((OPCON & 1) != 0);        //等待计算完成
res = MD3U32;                   //32 位结果

//////////////////////////////////////////////////////////////////////////
//32 位除以 16 位
//////////////////////////////////////////////////////////////////////////
unsigned long res;
unsigned long dat1;
unsigned int dat2;

P_SW2 |= 0x80;                  //访问扩展寄存器 xsfr
MD3U32 = dat1;                  //dat1 用户给定
MD5U16 = data2;                 //dat2 用户给定
ARCON = 6 << 5;                 //32 位/16 位,除法模式
OPCON = 1;                      //启动计算
while((OPCON & 1) != 0);        //等待计算完成
res = MD3U32;                   //32 位商，16 位余数在 MD5U16 中

//////////////////////////////////////////////////////////////////////////
//左移或右移:
```

```
//////////////////////////////////////////////////////////////////////////
unsigned long res;
unsigned long dat1;
unsigned char num;                          //移位的位数, 用户给定

MD3U32 = dat1;                              //dat1 用户给定
ARCON = (2 << 5) + num;                     //32 位左移模式
//ARCON = (1 << 5) + num;                   //32 位右移模式
OPCON = 1;                                  //启动计算
while((OPCON & 1) != 0);                    //等待计算完成
res = MD3U32;                               //32 位结果
```

STC MCU

附录A STC8C 系列单片机取代 STC8F 系列的注意事项

■ I/O 口

STC8C 系列单片机上电后，I/O 的模式与 STC8F 系列不一样。STC8F 系列单片机所有 I/O 口上电后都是 8051 的准双向口模式，而 STC8C 系列单片机的 I/O 中，除了 ISP 下载脚 P3.0/P3.1 为准双向口模式外，其余的所有 I/O 口在上电后都是高阻输入模式。传统的 8051 单片机上电后即为准双向口模式并输出高电平，经常有客户的系统中使用 I/O 驱动马达或者 LED 灯，因此会出现单片机上电的瞬间马达会动一下或者 LED 会闪一下。STC8C 系列的 I/O 上电后为高阻输入模式，就可避免马达和 LED 的这种误动作。

由于 STC8C 系列单片机的 I/O 中，除了 ISP 下载脚 P3.0/P3.1 为准双向口模式外，其余的所有 I/O 口在上电后都是高阻输入模式，所以当用户需要 STC8C 系列的 I/O 口向外输出信号前，必须先使用 PxM0 和 PxM1 两个寄存器对 I/O 的工作模式进行设置。

■ 复位脚

STC8C 系列和 STC8F 系列的 P5.4 口一般情况下是当作普通 I/O 口使用的，当用户在 ISP 下载时设置了 P5.4 为复位脚功能时，P5.4 口则为单片机的复位脚（RESET 脚）。对于 STC8F 系列，复位脚为高电平时单片机处于复位状态，低电平时单片机解除复位状态。而 STC8C 系列与 STC8F 系列的复位电平是向反的，即对于 STC8C 系列，复位脚为低电平时，单片机处于复位状态，高电平时单片机解除复位状态。

所以当用户使能 P5.4 口的复位脚功能是需要注意复位电平的问题。

■ EEPROM

STC8F 系列的 EEPROM 擦除和编程的等待时间用寄存器 IAP_CONTR 的 Bit2-Bit0 设置，设置的只是一个大概的频率范围值，STC8C 系列新增了一个寄存器 IAP_TPS（SFR 地址：0F5H），专用于设置 EEPROM 擦除和编程的等待时间，且用户不需要去计算，只需要根据当前 CPU 的工作频率，直接填入 IAP_TPS 即可，硬件会自动计算等待时间。（比如：当前 CPU 的工作频率为 24MHz，则只需要向 IAP_TPS 填入 24 即可）

附录B 电气特性

B.1 绝对最大额定值

参数	最小值	最大值	单位	说明
存储温度	-55	+150	℃	
工作温度	-40	+85	℃	若工作温度高于 85℃（如 125℃附近），由于内部 IRC 时钟的频率在高温时的温漂大，建议使用外部高温时钟或或晶振。另外温度高时频率跑不快，如果必须使用内部 IRC 时钟，建议使用 24M 以下的工作频率；如果系统必须运行在较高频率，则请使用外部高可靠有源时钟。 若工作温度为-55℃附近，则工作电压不能太低，强烈建议 MCU-VCC 电压不要低于 3.0V，另外电源的上升速度也必须尽量快，最好能控制在毫秒级
工作电压	1.9	5.5	V	
VDD 对地电压	-0.3	+5.5	V	
I/O 口对地电压	-0.3	VDD+0.3	V	

STC MCU

B.2 直流特性（3.3V）

（VSS=0V，VDD=3.3V，测试温度=25℃）

标号	参数	范围				测试环境及说明
		最小值	典型值	最大值	单位	
I_{PD}	掉电模式电流	-	0.4	-	uA	
I_{WKT}	掉电唤醒定时器	-	1.4	-	uA	
I_{LVD}	低压检测模块功耗	-	10	-	uA	
I_{CMP}	比较器功耗	-	90	-	uA	
I_{IDL}	空闲模式电流（内部 32KHz）	-	0.48	-	mA	相当于传统 8051 的 0.5M
	空闲模式电流（6MHz）	-	0.88	-	mA	相当于传统 8051 的 79M
	空闲模式电流（12MHz）	-	1.00	-	mA	相当于传统 8051 的 158M
	空闲模式电流（24MHz）	-	1.16	-	mA	相当于传统 8051 的 317M
I_{NOR}	正常模式电流（内部 32KHz）	-	0.48	-	mA	相当于传统 8051 的 0.5M
	正常模式电流（500KHz）	-	0.88	-	mA	相当于传统 8051 的 7M
	正常模式电流（600KHz）	-	0.88	-	mA	相当于传统 8051 的 8M
	正常模式电流（700KHz）	-	0.90	-	mA	相当于传统 8051 的 9M
	正常模式电流（800KHz）	-	0.91	-	mA	相当于传统 8051 的 11M
	正常模式电流（900KHz）	-	0.91	-	mA	相当于传统 8051 的 12M
	正常模式电流（1MHz）	-	0.94	-	mA	相当于传统 8051 的 13M
	正常模式电流（2MHz）	-	1.05	-	mA	相当于传统 8051 的 26M
	正常模式电流（3MHz）	-	1.17	-	mA	相当于传统 8051 的 40M
	正常模式电流（4MHz）	-	1.26	-	mA	相当于传统 8051 的 53M
	正常模式电流（5MHz）	-	1.40	-	mA	相当于传统 8051 的 66M
	正常模式电流（6MHz）	-	1.49	-	mA	相当于传统 8051 的 79M
	正常模式电流（12MHz）	-	2.09	-	mA	相当于传统 8051 的 158M
	正常模式电流（24MHz）	-	3.16	-	mA	相当于传统 8051 的 317M
V_{IL1}	输入低电平	-	-	0.99	V	打开施密特触发
		-	-	1.07	V	关闭施密特触发
V_{IH1}	输入高电平（普通 I/O）	1.18	-	-	V	打开施密特触发
		1.09	-	-	V	关闭施密特触发
V_{IH2}	输入高电平（复位脚）	1.18	-	0.99	V	
I_{OL1}	输出低电平的灌电流	-	20	-	mA	端口电压 0.45V
I_{OH1}	输出高电平电流（双向模式）	-	110	-	uA	
I_{OH2}	输出高电平电流（推挽模式）	-	20	-	mA	端口电压 2.4V
I_{IL}	逻辑 0 输入电流	-	-	50	uA	端口电压 0V
I_{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	端口电压 2.0V
R_{PU}	I/O 口上拉电阻	5.8	5.9	6.0	KΩ	
I/O 速度	I/O 大电流驱动，I/O 快速转换		25		MHz	PxDR=0, PxSR=0
	I/O 小电流驱动，I/O 快速转换		22		MHz	PxDR=1, PxSR=0
	I/O 大电流驱动，I/O 慢速转换		16		MHz	PxDR=0, PxSR=1
	I/O 小电流驱动，I/O 慢速转换		12		MHz	PxDR=1, PxSR=1
比较器	最快速度		10		MHz	关闭所有模拟和数字滤波
	模拟滤波时间		0.1		us	

标号	参数	范围				测试环境及说明
		最小值	典型值	最大值	单位	
	数字滤波时间		0		系统时钟	LCDTY＝0
			n+2			LCDTY=n (n=1~63)
I_{PD2}	使能比较器时掉电模式功耗	-	400	-	uA	
I_{PD3}	使能 LVD 时掉电模式功耗	-	470	-	uA	

STC MCU

B.3 直流特性（5.0V）

（VSS=0V，VDD=5.0V，测试温度=25℃）

标号	参数	范围				测试环境及说明
		最小值	典型值	最大值	单位	
I_{PD}	掉电模式电流	-	0.6	-	uA	
I_{WKT}	掉电唤醒定时器	-	3.6	-	uA	
I_{LVD}	低压检测模块功耗	-	30	-	uA	
I_{CMP}	比较器功耗	-	90	-	uA	
I_{IDL}	空闲模式电流（内部 32KHz）	-	0.58	-	mA	相当于传统 8051 的 0.5M
	空闲模式电流（6MHz）	-	0.98	-	mA	相当于传统 8051 的 79M
	空闲模式电流（12MHz）	-	1.10	-	mA	相当于传统 8051 的 158M
	空闲模式电流（24MHz）	-	1.25	-	mA	相当于传统 8051 的 317M
I_{NOR}	正常模式电流（内部 32KHz）	-	0.58	-	mA	相当于传统 8051 的 0.5M
	正常模式电流（500KHz）		0.97		mA	相当于传统 8051 的 7M
	正常模式电流（600KHz）		0.97		mA	相当于传统 8051 的 8M
	正常模式电流（700KHz）		1.00		mA	相当于传统 8051 的 9M
	正常模式电流（800KHz）		1.01		mA	相当于传统 8051 的 11M
	正常模式电流（900KHz）		1.01		mA	相当于传统 8051 的 12M
	正常模式电流（1MHz）		1.03		mA	相当于传统 8051 的 13M
	正常模式电流（2MHz）		1.15		mA	相当于传统 8051 的 26M
	正常模式电流（3MHz）		1.27		mA	相当于传统 8051 的 40M
	正常模式电流（4MHz）		1.35		mA	相当于传统 8051 的 53M
	正常模式电流（5MHz）		1.49		mA	相当于传统 8051 的 66M
	正常模式电流（6MHz）	-	1.59	-	mA	相当于传统 8051 的 79M
	正常模式电流（12MHz）	-	2.19	-	mA	相当于传统 8051 的 158M
	正常模式电流（24MHz）	-	3.27	-	mA	相当于传统 8051 的 317M
V_{IL1}	输入低电平	-	-	1.32	V	打开施密特触发
		-	-	1.48	V	关闭施密特触发
V_{IH1}	输入高电平（普通 I/O）	1.60	-	-	V	打开施密特触发
		1.54	-	-	V	关闭施密特触发
V_{IH2}	输入高电平（复位脚）	1.60	-	1.32	V	
I_{OL1}	输出低电平的灌电流	-	20	-	mA	端口电压 0.45V
I_{OH1}	输出高电平电流（双向模式）	-	200	-	uA	
I_{OH2}	输出高电平电流（推挽模式）	-	20	-	mA	端口电压 2.4V
I_{IL}	逻辑 0 输入电流	-	-	50	uA	端口电压 0V
I_{TL}	逻辑 1 到 0 的转移电流	100	270	600	uA	端口电压 2.0V
R_{PU}	I/O 口上拉电阻	4.1	4.2	4.4	KΩ	
I/O 速度	I/O 大电流驱动，I/O 快速转换		36		MHz	PxDR=0, PxSR=0
	I/O 小电流驱动，I/O 快速转换		32		MHz	PxDR=1, PxSR=0
	I/O 大电流驱动，I/O 慢速转换		26		MHz	PxDR=0, PxSR=1
	I/O 小电流驱动，I/O 慢速转换		22		MHz	PxDR=1, PxSR=1
比较器	最快速度		10		MHz	关闭所有模拟和数字滤波
	模拟滤波时间		0.1		us	

标号	参数	范围				测试环境及说明
		最小值	典型值	最大值	单位	
	数字滤波时间		0		系统时钟	LCDTY＝0
			n+2			LCDTY=n (n=1~63)
I_{PD2}	使能比较器时掉电模式功耗	-	460	-	uA	
I_{PD3}	使能 LVD 时掉电模式功耗	-	520	-	uA	

I_{PD3}	使能 LVD 时掉电模式功耗	-	520	-	uA	

B.4 I/O 口驱动能力（驱动电流对应的 I/O 上的电压）

（VSS=0V，VDD=5.0V，测试温度=25℃）

普通 I/O 推挽输出 1		
	一般推力	强推力
10mA	4.50V	4.72V
20mA	4.00V	4.49V
30mA	3.40V	4.24V
40mA	2.31V	3.96V
50mA	-	3.65V
60mA	-	3.25V
70mA	-	2.75V
80mA	-	1.65V

普通 I/O 推挽输出 0/准双向口输出 0/开漏模式 0		
	一般推力	强推力
10mA	0.34V	0.20V
20mA	0.66V	0.35V
30mA	1.04V	0.52V
40mA	1.70V	0.68V
50mA	-	0.88V
60mA	-	1.14V
70mA	-	1.44V
80mA	-	1.93V

B.5 内部 IRC 温漂特性（参考温度 25℃）

温度	范围		
	最小值	典型值	最大值
-40℃～85℃		-1.38%～+1.42%	
-20℃～65℃		-0.88%～+1.05%	

B.6 低压复位门槛电压（测试温度 25℃）

级别	电压		
	最小值	典型值（实测值）	最大值
POR		（1.69V~1.82V）	
LVR0		2.0V（1.88V~1.99V）	
LVR1		2.4V（2.28V~2.45V）	
LVR2		2.7V（2.58V~2.76V）	
LVR3		3.0V（2.86V~3.06V）	

STC MCU

附录C　关于回流焊前是否要烘烤

根据国际湿气敏感性等级 3（MSL3）规范的要求，贴片元器件在拆开真空包装后，168 小时内，7 天内，必须回流焊贴片完成，如未完成，必须再次高温烘烤。

LQFP/QFN/DFN 托盘能耐 100 度以上的高温，拆开真空包装后 7 天内必须回流焊贴片完成，否则回流焊前必须重新烘烤：110~125℃，4~8 个小时都可以。

SOP/TSSOP 塑料管耐不了 100 度以上的高温，拆开真空包装后 7 天内必须回流焊贴片完成，否则回流焊前去除耐不了 100 度以上高温的塑料管，放到金属托盘中，重新烘烤：110~125℃，4~8 个小时都可以，再人工放回塑料管。所以尽量选择 LQFP/QFN/DFN 封装形式，方便重新烘烤。

STCAI 贴片 MCU 最小真空包装数量：

真空托盘包装，是最高等级包装，可保存 N 年，回流焊前方便重新烘烤。拆开真空包装后，7 天内必须贴片完成！托盘耐高温 150 度，包装成本高。

LQFP64：　最小真空包装，1600pcs/包；拆开最小真空包装后，160pcs/盘
LQFP48：　最小真空包装，2500pcs/包；拆开最小真空包装后，250pcs/盘
LQFP44：　最小真空包装，1600pcs/包；拆开最小真空包装后，160pcs/盘
LQFP32：　最小真空包装，2500pcs/包；拆开最小真空包装后，250pcs/盘

QFN64/8*8：最小真空包装，2600pcs/包；拆开最小真空包装后，260pcs/盘
QFN48/6*6：最小真空包装，4900pcs/包；拆开最小真空包装后，490pcs/盘
QFN32/4*4：最小真空包装，4900pcs/包；拆开最小真空包装后，490pcs/盘
QFN20/3*3：最小真空包装，4900pcs/包；拆开最小真空包装后，490pcs/盘
DFN8/3*3：　最小真空包装，4900pcs/包；拆开最小真空包装后，490pcs/盘

管装真空包装，塑料管是普通材料，不耐高温，包装成本低，回流焊前不方便重新烘烤！拆开真空包装后，7 天内必须贴片完成！

SOP28：　最小真空包装，2080pcs/包；拆开最小真空包装后，26pcs/管
SOP20：　最小真空包装，2880pcs/包；拆开最小真空包装后，36pcs/管
TSSOP28：最小真空包装，7500pcs/包；拆开最小真空包装后，50pcs/管
TSSOP20：最小真空包装，14400pcs/包；拆开最小真空包装后，72pcs/管
SOP16：　最小真空包装，5000pcs/包；拆开最小真空包装后，50pcs/管
SOP8：　最小真空包装，10000pcs/包；拆开最小真空包装后，100pcs/管

请按最小包装的整数倍买，否则零头部分，回流焊前需要重新烘烤。

【举例 1】：TSSOP20，你买了 1000pcs，你以为是整数，其实是零头，因为最小真空包装是 14400pcs/包，你拿到的并不是一个最小真空包装，而是已拆开真空包装不知道多少天的零头部分，必须重新烘烤。
【举例 2】：TSSOP20，你买了 10000pcs，你以为是整数，其实是零头，因为最小真空包装是 14400pcs/包，你拿到的并不是一个最小真空包装，而是已拆开真空包装不知道多少天的零头部分，必须重新烘烤。
【举例 3】：LQFP64，你买了 2000pcs，则拿到的是 1 个最小真空包装/1600 + 400 个零头
===400 个零头你回流焊前需重新烘烤；
===LQFP64：最小真空包装，1600pcs/包；拆开最小真空包装后，160pcs/盘

编带真空包装，编带是普通材料，不耐高温，包装成本低，回流焊前不方便重新烘烤！拆开真空包装后，7 天内必须贴片完成！据说回流焊前，可以 55℃烘烤 6 小时，再贴片 / 回流焊。

TSSOP20，　编带：最小真空包装，3000pcs/卷；
SOP8，　编带：最小真空包装，4000pcs/卷。

附录D 如何使用万用表检测芯片 I/O 口好坏

根据国际湿气敏感性等级 3（MSL3）规范的要求，贴片元器件在拆开真空包装后，168 小时内，7 天内，必须回流焊贴片完成，如未完成，必须再次高温烘烤。如果没有高温烘烤的流程，直接进行回流焊，则可能由于芯片内外受热不均导致芯片内部金属线被拉断，最终出现的现象是芯片 I/O 口损坏。

STC 的单片机在芯片设计时，每个 I/O 口都有两个分别接到 VCC 和 GND 的保护二极管，用万用表的二极管检测档可以进行测量。可使用此方法简单判断 I/O 管脚的好坏情况。使用万用表测量方法如下（注：这里使用的是数字万用表）

首先将万用表调到二极管检测档，被测芯片不要供电，将万用表的红表笔连接到被测芯片的 **GND 管脚**，黑表笔依次测量每个 I/O 口，如果万用表显示的参数为 0.45～0.75V 左右，则表示芯片的内部 I/O 到 GND 的保护二极管正常，即打线也是完好的，若显示的参数为开路/断开的状态，则表示芯片内部的打线已断路。

上面是检测芯片内部的打线情况的方法。

另外，如果用户板上，单片机的管脚没有加保护电路，一旦出现过流或者过压都可能导致 I/O 烧坏。为了检测管脚是否被烧坏，除了使用上面的方法检测 I/O 口到 GND 的保护二极管外，还需要检测 I/O 口到 VCC 的保护二极管。使用万用表检测 I/O 口到 VCC 的保护二极管的方法如下：

首先将万用表调到二极管检测档，被测芯片不要供电，将万用表的黑表笔连接到被测芯片的 **VCC 管脚**，红表笔依次测量每个 I/O 口，如果万用表显示的参数为 0.45～0.75V 左右，则表示芯片的内部 I/O 到 VCC 的保护二极管正常，若显示的参数为开路/断开的状态，则表示芯片此端口已被损坏。

STCMCU

附录E 大批量生产，如何省去专门的烧录人员，如何无烧录环节

大批量生产，你在将由 STC 的 MCU 作为主控芯片的控制板组装到设备里面之前在你将 STC MCU 贴片到你的控制板完成之后，你必须测试你的控制板的好坏。不要说 100%，直通无问题，那是抬杠，不是搞生产，只要生产，就会虚焊，短路，部分元件贴错，部分元件采购错。

所以在贴片回来后，组装到外壳里面之前，你必须要测试，你的含有 STC MCU 控制板的好坏，好的去组装，坏的去维修抢救。

测试，大批量生产，必须有测试架/下面接上我们的脱机烧录工具 U8W/U8W-Mini/STC-USB Link1，还要接上其他控制部分

通过 USER-VCC、P3.0、P3.1、GND 连接，要工人每次都开电源
通过 S-VCC、P3.0、P3.1、GND 连接，不要你开电源，STC 的脱机工具给你自动供电

外面帮你做一个测试架的成本 500 元以下，就是有机玻璃，夹具，顶针。
1 个测试你控制板是否正常的工人管理 2 - 3 个 测试架

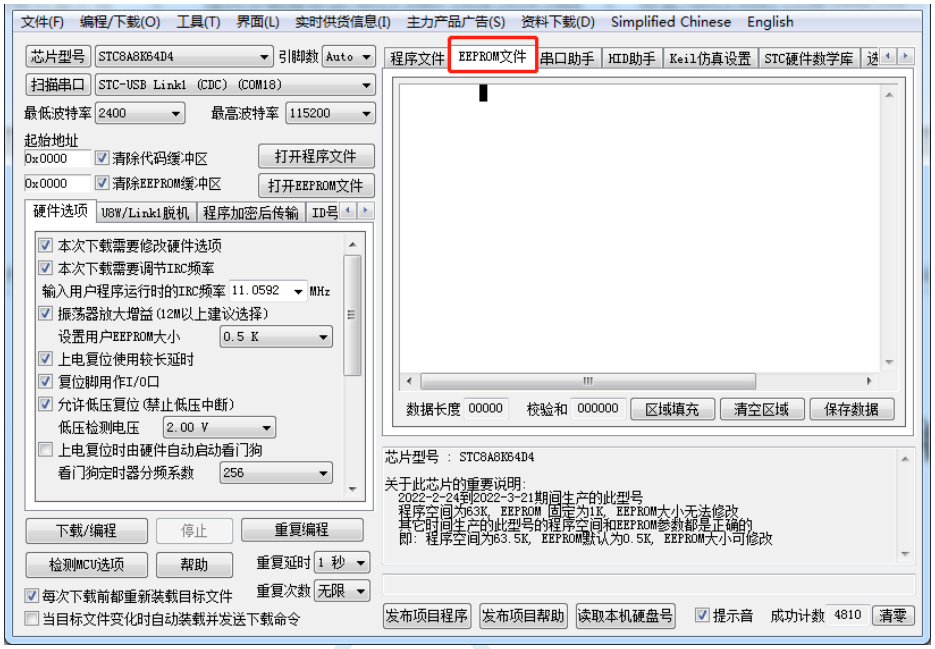
操作流程：

1、 将你的 STC MCU 控制板 卡到测试架 1 上
2、 将你的 STC MCU 控制板 卡到测试架 2 上，测试架 1 上的程序已烧录完成/感觉不到烧录时间
3、 测试 测试架 1 上的 STC 主控板功能是否正常，正常放到正常区，不正常，放到不正常区
4、 给测试架 1 卡上新的未测试的无程序的控制板
5、 测试 测试架 2 上的未测试控制板/程序不知何时早就不知不觉的烧好了，换新的未测试未烧录的控制板
6、 循环步骤 3 到步骤 5

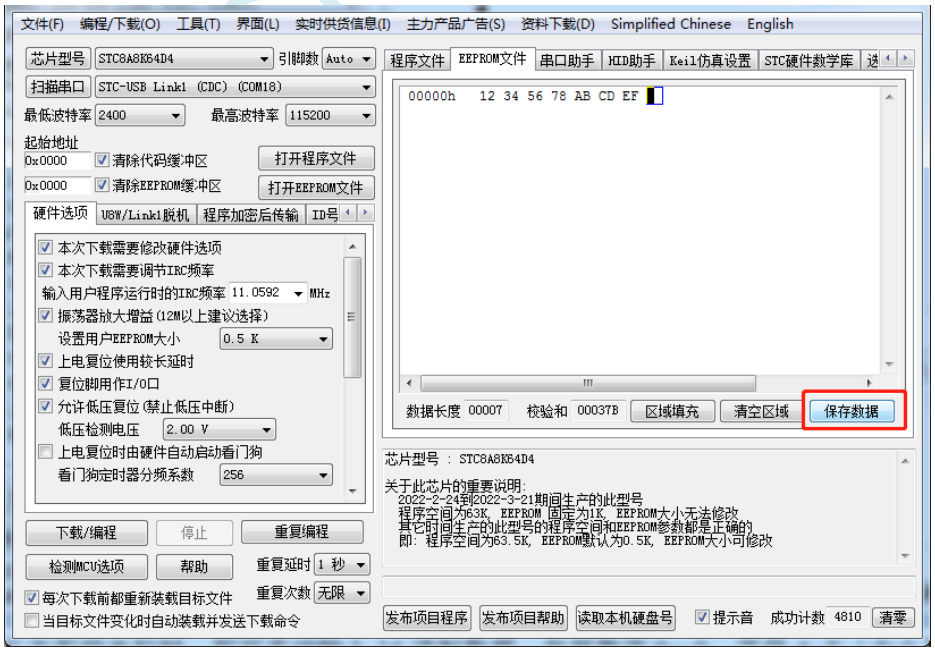
========不需要安排烧录人员

附录F　如何使用 AIapp-ISP 下载软件制作和编辑 EEPROM 文件

打开任意版本 AIapp-ISP 下载软件，选择“EEPROM”页面，单击数据窗口，如下所示



当出现黑色长方形的光标时，即可手动输入 16 进制数据，包括数字 0~9、字母 A~F（大小写通用）数据输入完成后，点击“保存数据”按钮即可保存 EEPROM 数据



附录G　编译器（汇编器）/仿真器使用指南

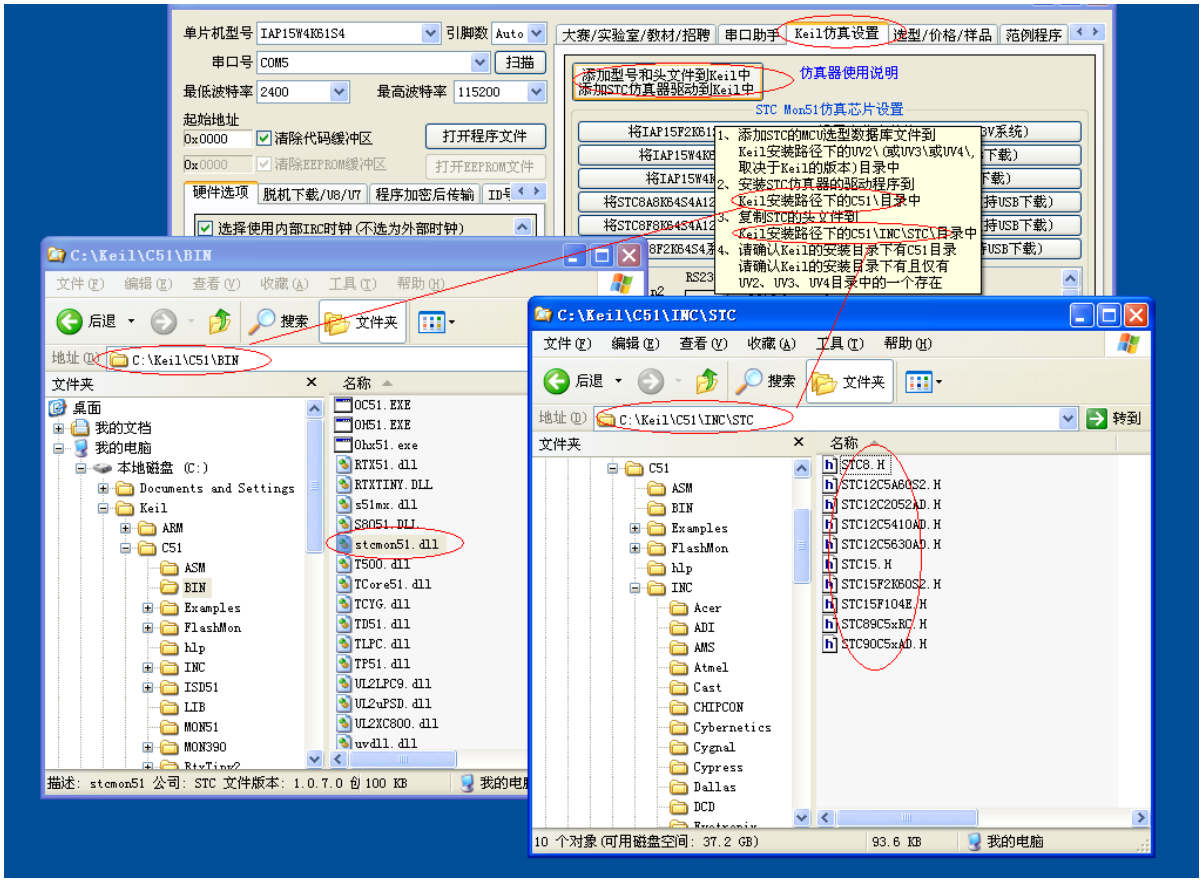
A：STC 单片机应使用何种编译器/汇编器？

Q：任何老式的 8051 编译器/汇编器都可以支持，现流行使用 Keil C51

A：Keil 环境中，应如何包含头文件

Q：按照下面图示的步骤安装完驱动和头文件后，新建项目时选择 STC 相应的单片机型号，在源文件中直接使用“#include <STC8C.h>”即可完成头文件的包含。如果新建项目时选择的 Intel 的 8052/87C52/87C54/87C58 或 Philips 的 P87C52/P87C54/P87C58 编译，头文件包含<reg51.h>即可，不过 STC 新增的特殊功能寄存器则需要用户自己声明。

1、安装 Keil 版本的仿真驱动



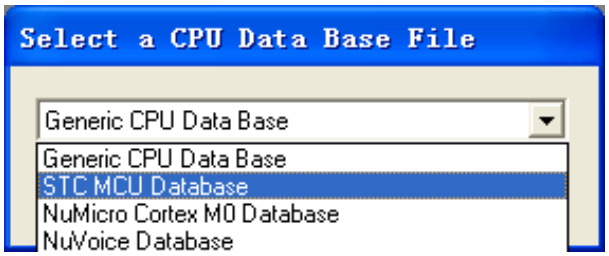
如上图，首先选择“Keil 仿真设置”页面，点击“添加 MCU 型号到 Keil 中”，在出现的如下的目录选择窗口中，定位到 Keil 的安装目录(一般可能为“C:\Keil\”)，“确定”后出现下图中右边所示的提示信息，表示安装成功。添加头文件的同时也会安装 STC 的 Monitor51 仿真驱动 STCMON51.DLL，驱动与头文件的安装目录如上图所示。



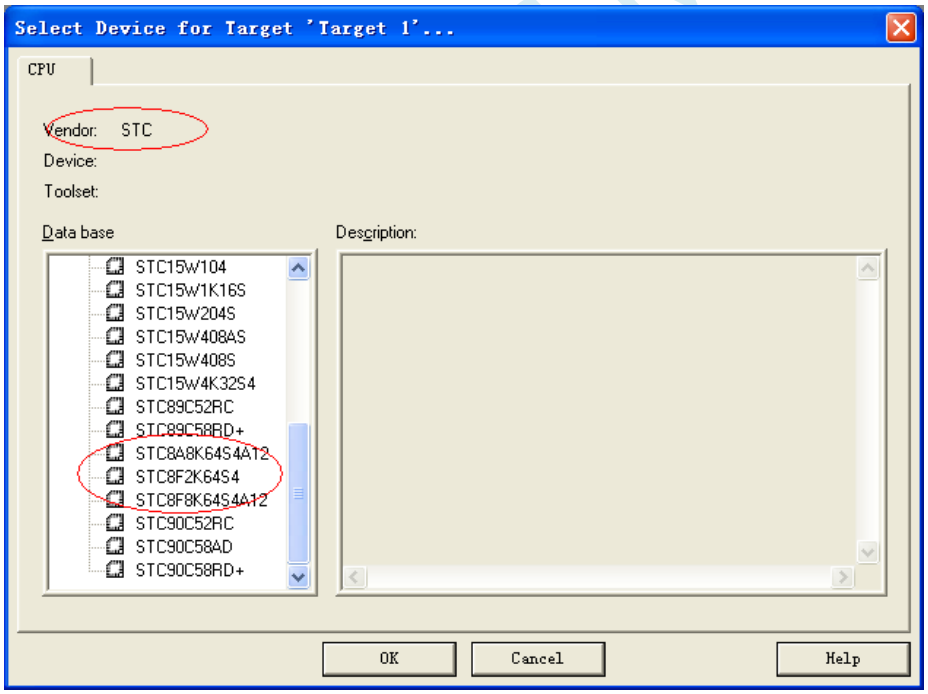


2、在 Keil 中创建项目

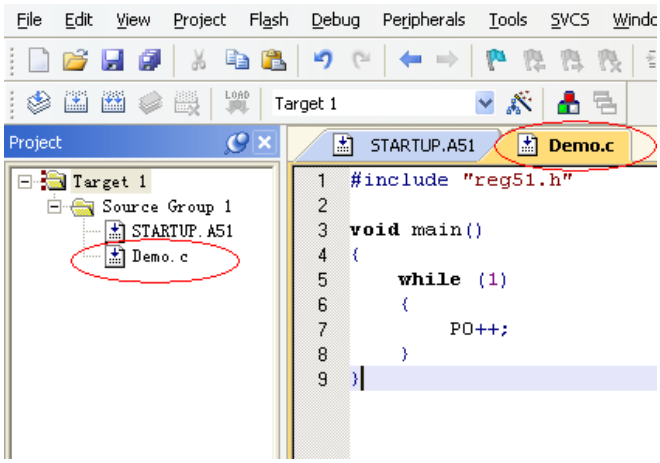
若第一步的驱动安装成功，则在 Keil 中新建项目时选择芯片型号时，便会有“STC MCU Database”的选择项，如下图



然后从列表中选择响应的 MCU 型号，我们在此选择“STC8A8K64S4A12”的型号，点击“确定”完成选择



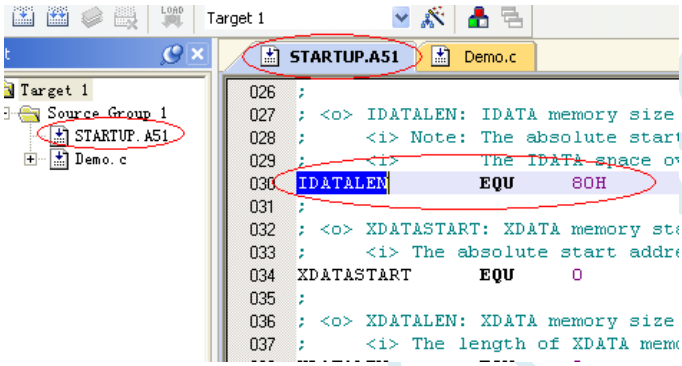
添加源代码文件到项目中，如下图：



保存项目，若编译无误，则可以进行下面的项目设置了

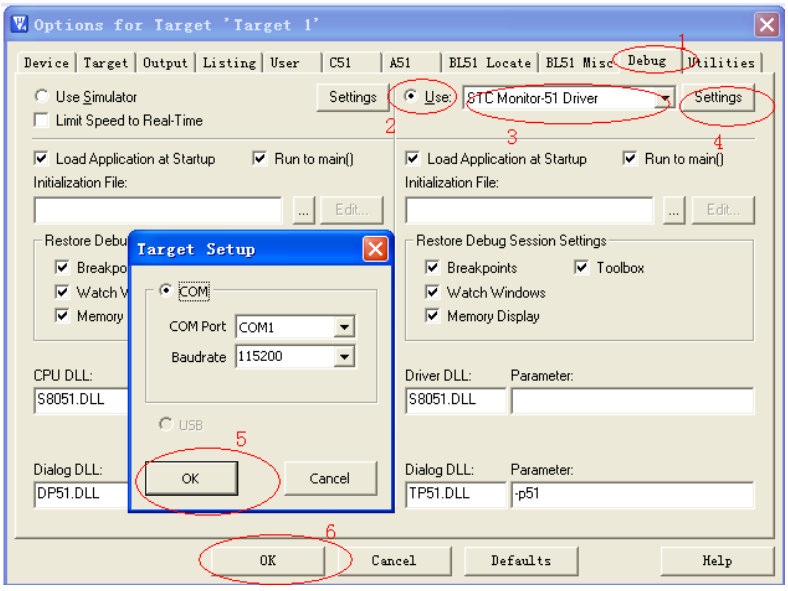
附加说明一点：

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。

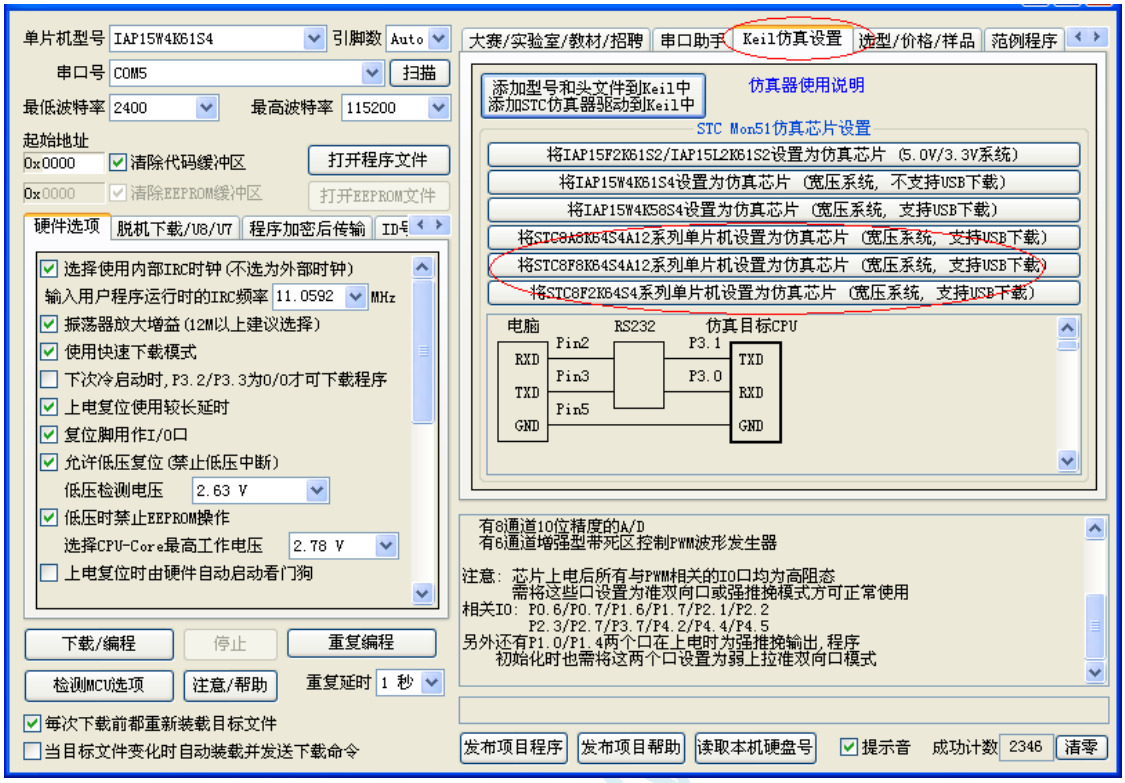
3、项目设置，选择 STC 仿真驱动



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。

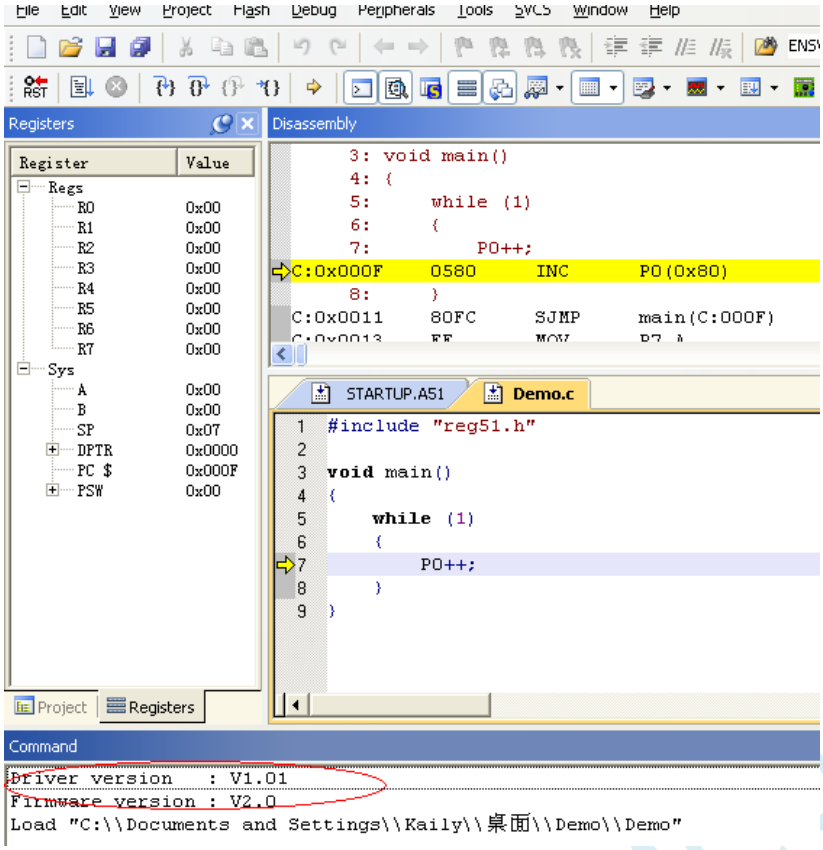
5、开始仿真

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。

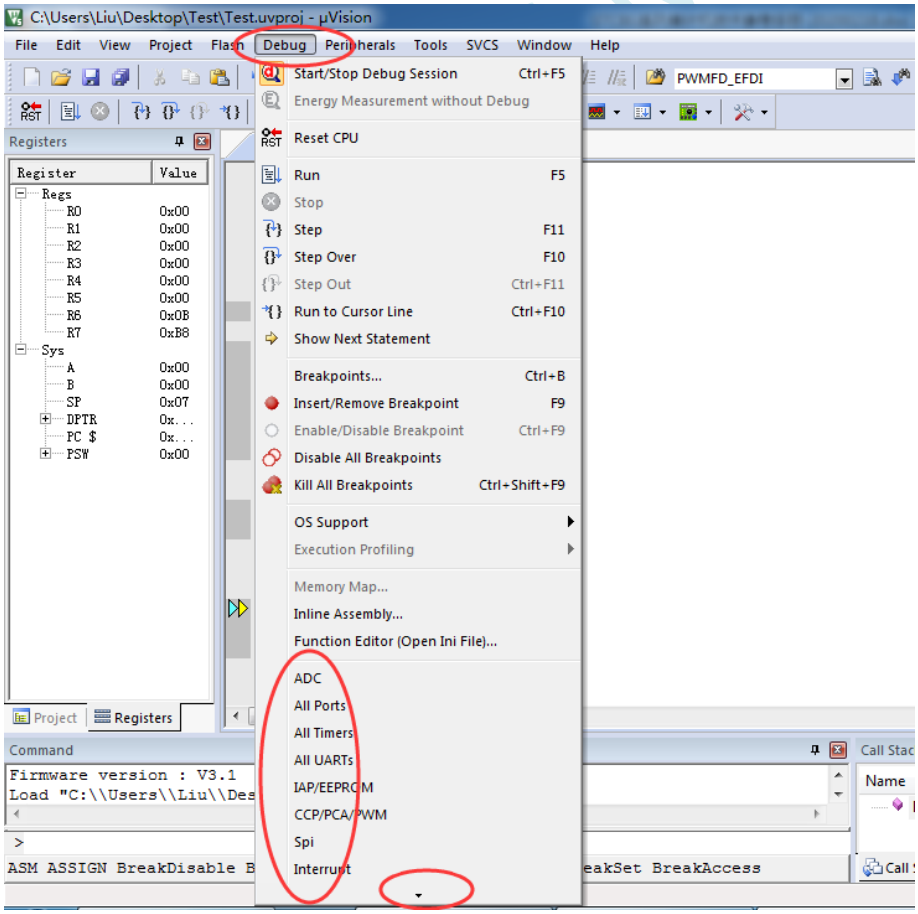
若硬件连接无误的话，将会进入到类似于下面的调试界面，并在命令输出窗口显示当前的仿真驱动版本号和当前仿真监控代码固件的版本号

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）。



6、仿真过程中，寄存器的查看

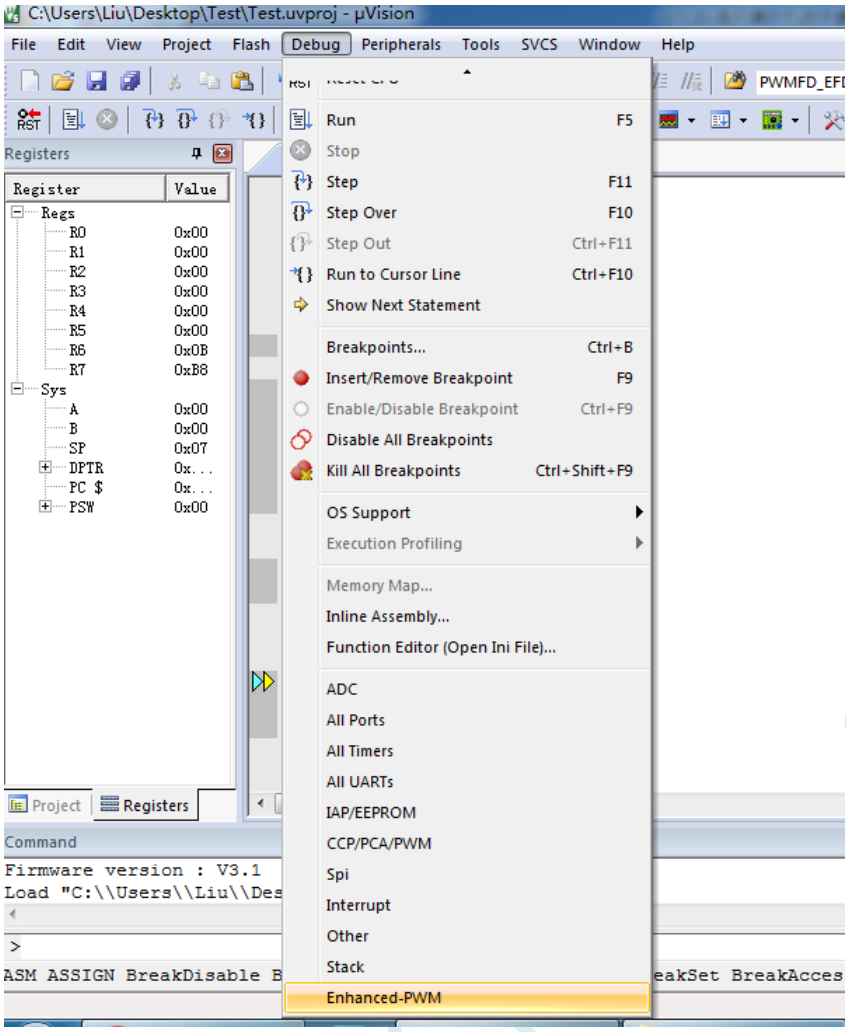
在仿真的过程中，可查看 MCU 相关的寄存器。所有的寄存器列表在“Debug”菜单的底端。如下图所示：



在上图“Debug”菜单的最底端，还有一个黑色的小三角，这表示还有隐藏的项目（主要是由于显示版

面大小的原因）

将鼠标仿真小三角上即可自动拖出所有的项目，如下图：



仿真注意事项：

1、 仿真监控程序占用 P3.0/P3.1 两个端口，但不占用串口 1，用户可以将串口 1 切换到 P3.6/P3.7 或者 P1.6/P1.7 再使用

2、 仿真监控程序占用内部扩展 RAM(XDATA)的最后 768 字节，用户不可对这个区域的 XDATA 进行写操作

附录H　AIapp-ISP 下载软件高级应用

H.1　发布项目程序

发布项目程序功能主要是将用户的程序代码与相关的选项设置打包成为一个可以直接对目标芯片进行下载编程的超级简单的用户自己界面的可执行文件。

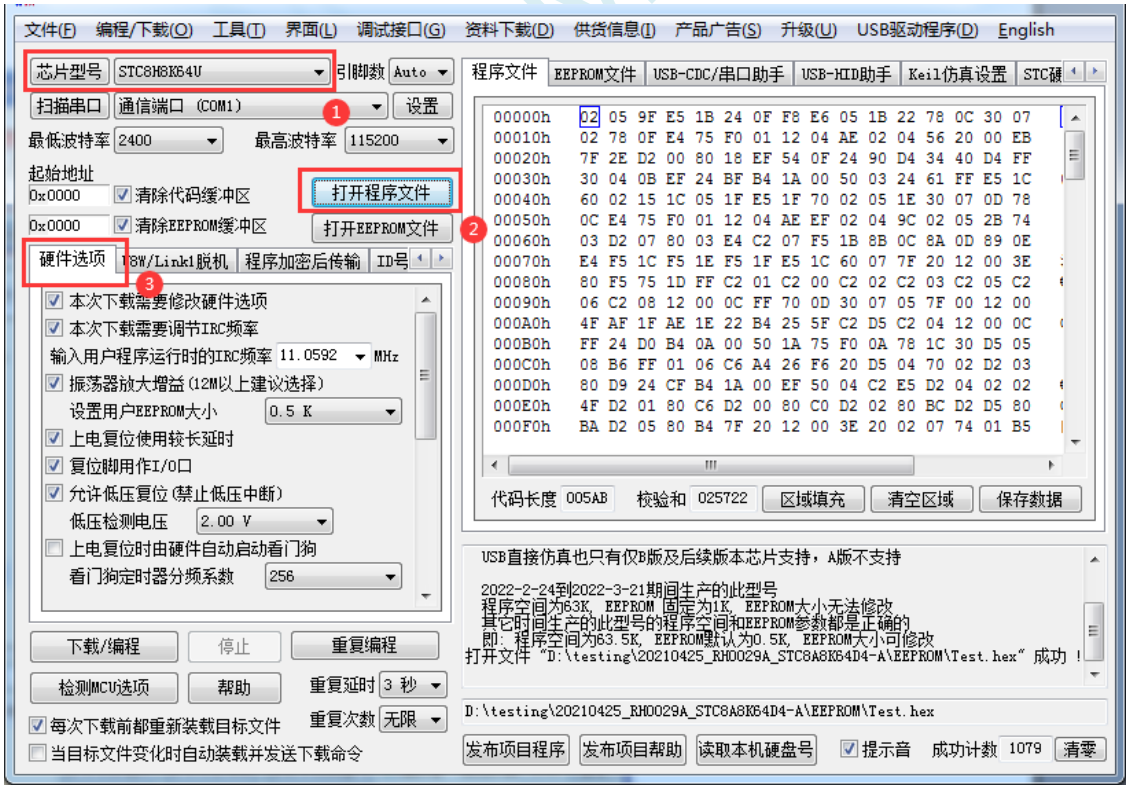
关于界面，用户可以自己进行定制（用户可以自行修改发布项目程序的标题、按钮名称以及帮助信息），同时用户还可以指定目标电脑的硬盘号和目标芯片的 ID 号，指定目标电脑的硬盘号后，便可以控制发布应用程序只能在指定的电脑上运行（防止烧录人员将程序轻易从电脑盗走,如通过网络发走,如通过U盘烤走,防不胜防,当然盗走你的电脑那就没办法那,所以 STC 的脱机下载工具比电脑烧录安全,能限制可烧录芯片数量,让前台文员小姐烧,让老板娘烧都可以），拷贝到其它电脑，应用程序不能运行。同样的，当指定了目标芯片的 ID 号后，那么用户代码只能下载到具有相应 ID 号的目标芯片中（对于一台设备要卖几千万的产品特别有用---坦克,可以发给客户自己升级,不需冒着生命危险跑到战火纷飞的伊拉克升级软件啦），对于 ID 号不一致的其它芯片，不能进行下载编程。

发布项目程序详细的操作步骤如下：

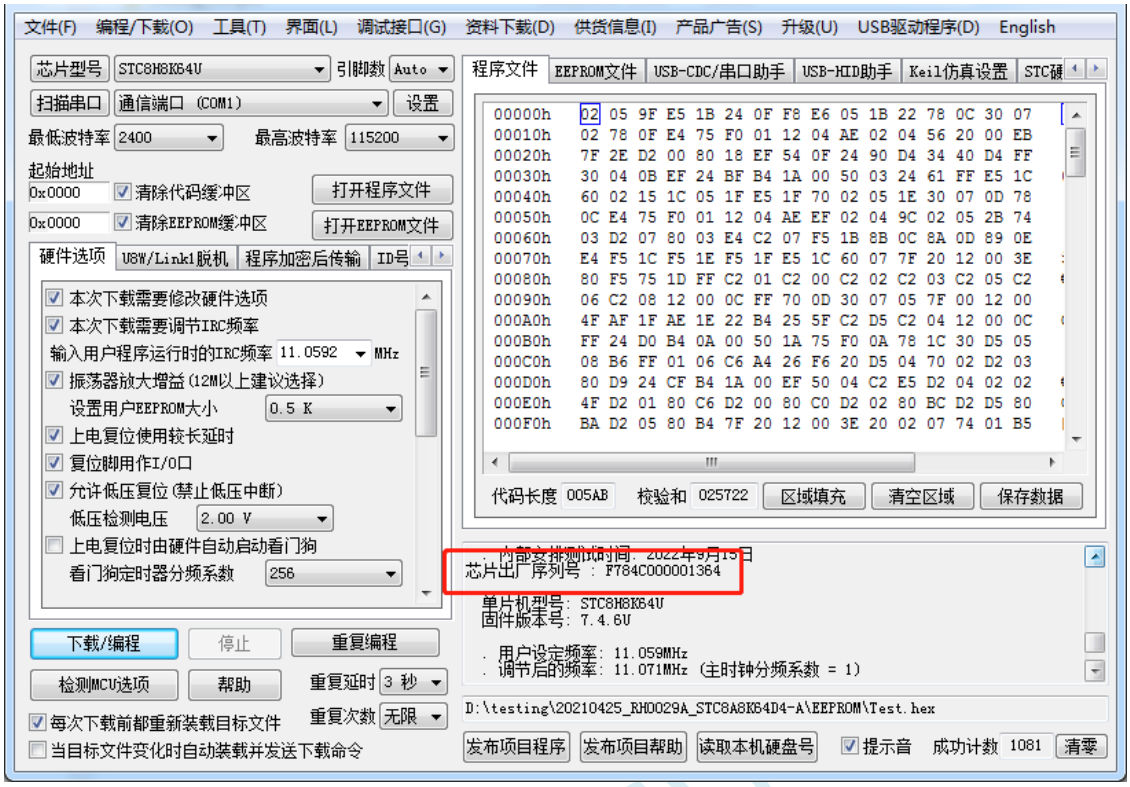
1、首先选择目标芯片的型号

2、打开程序代码文件

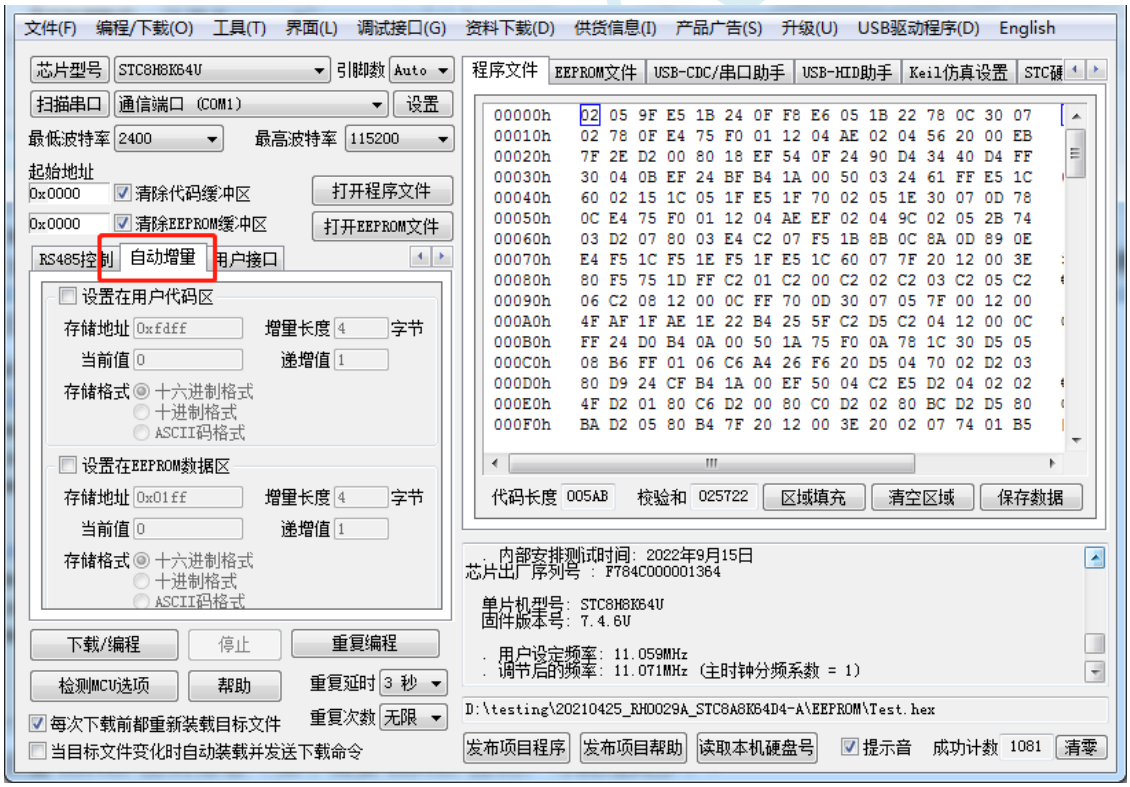
3、设置好相应的硬件选项



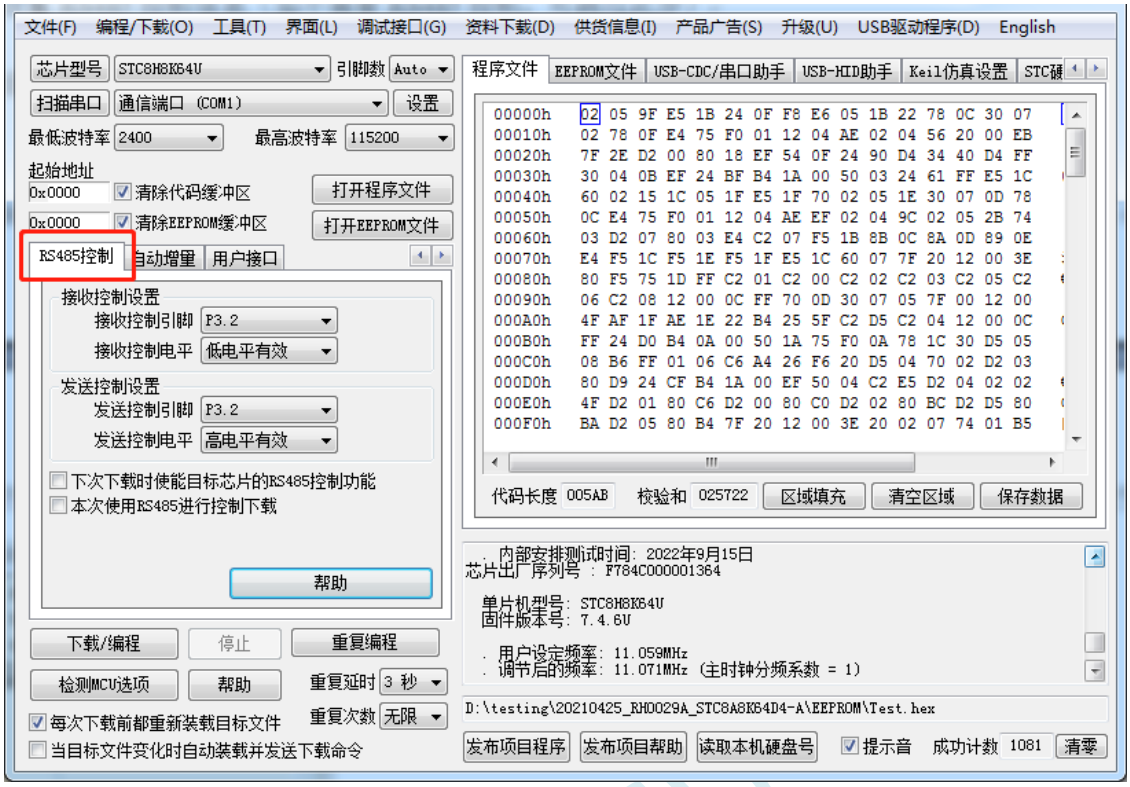
4、试烧一下芯片，并记下目标芯片的 ID 号，如下图所示，该芯片的 ID 号即为“F784C000001364”（如不需要对目标芯片的 ID 号进行校验，可跳过此步）



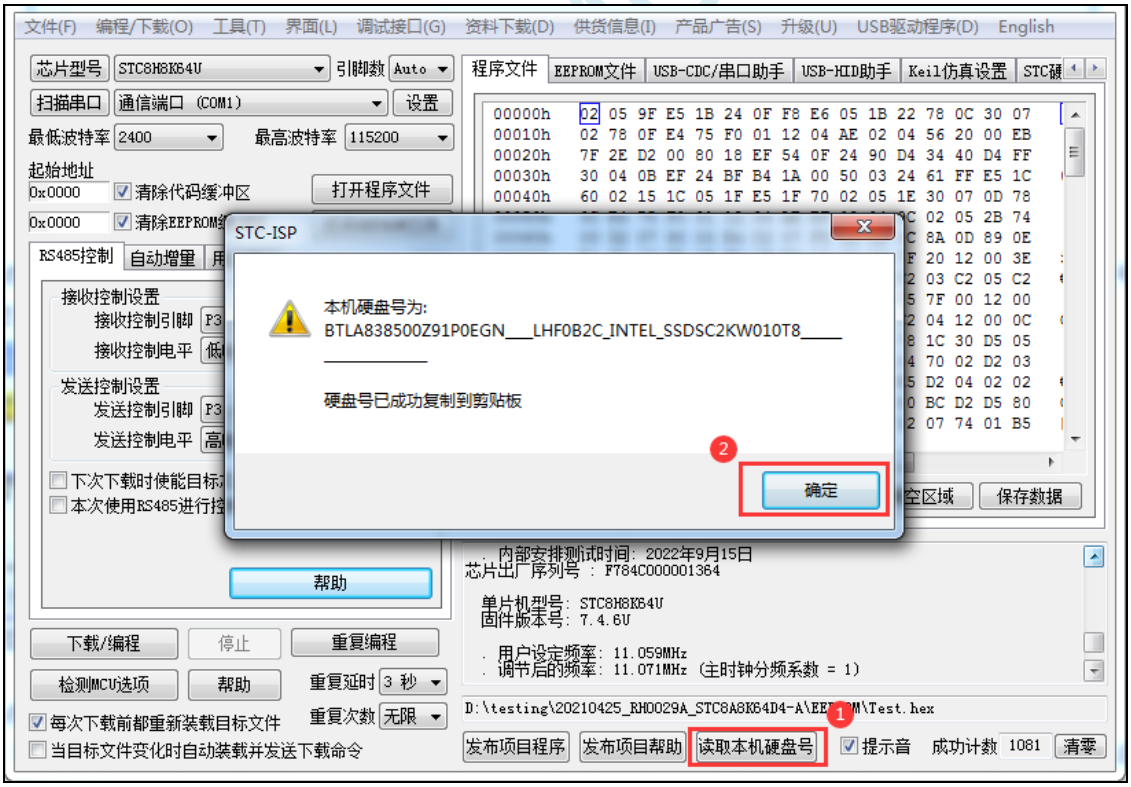
5、设置自动增量（如不需要自动增量，可跳过此步）



6、设置 RS485 控制信息（如不需要 RS485 控制，可跳过此步）



7、点击界面上的“读取本机硬盘号”按钮，并记下目标电脑的硬盘号（如不需要对目标电脑的硬盘号进行校验，可跳过此步）



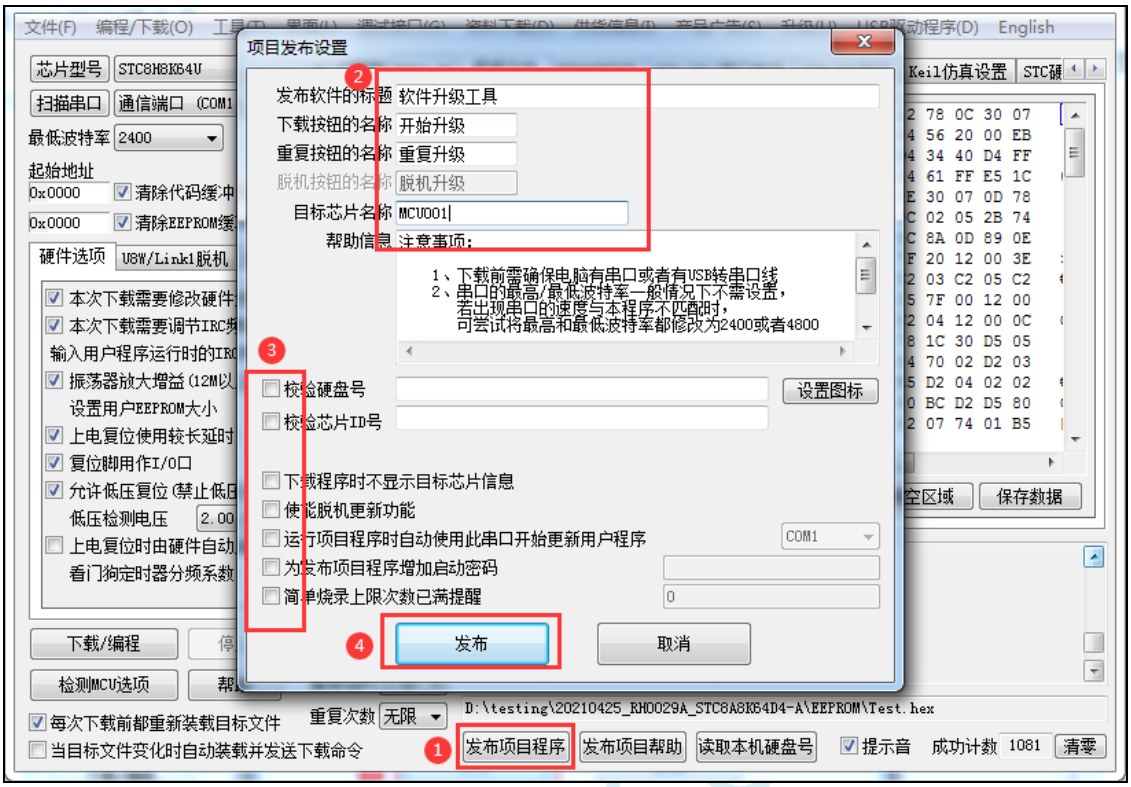
8、点击“发布项目程序”按钮，进入发布应用程序的设置界面。

9、根据各自的需要，修改发布软件的标题、下载按钮的名称、重复下载按钮的名称、自动增量的名称以及帮助信息

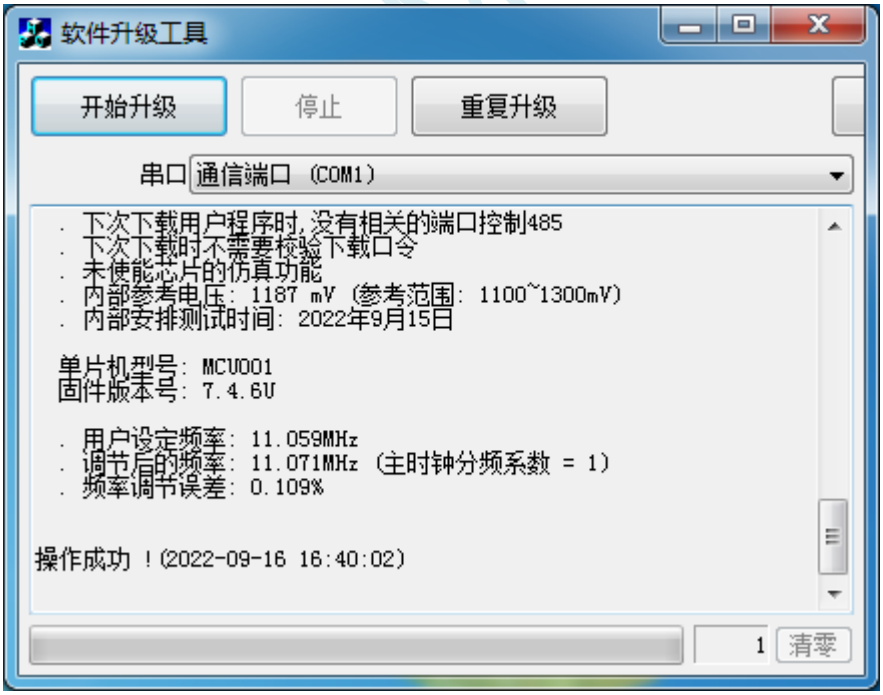
10、若需要校验目标电脑的硬盘号,则需要勾选上“校验硬盘号”，并在后面的文本框内输入前面所记下的目标电脑的硬盘号

11、若需要校验目标芯片的 ID 号，则需要勾选上“校验芯片 ID 号”，并在后面的文本框内输入前面

所记下的目标芯片的 ID 号



12、最后点击发布按钮，将项目发布程序保存，即可得到相应的可执行文件。发布的项目程序打界面如下图



H.2　程序加密后传输（防烧录时串口分析出程序）

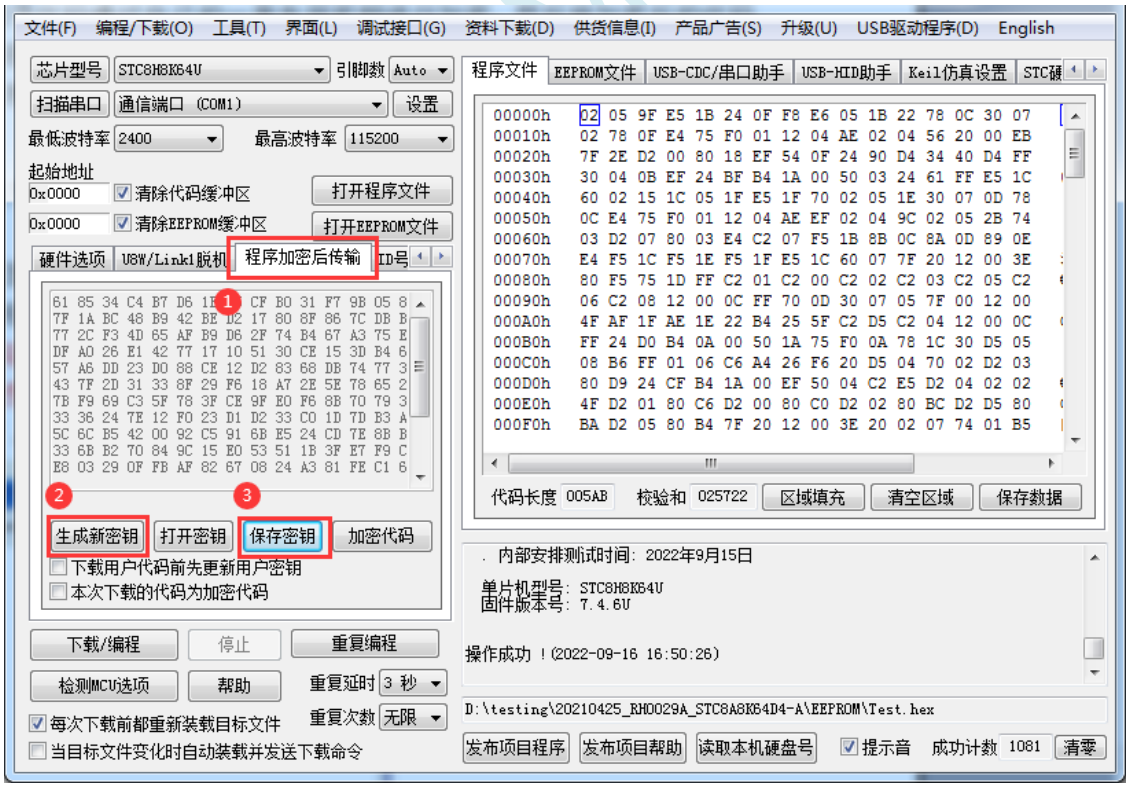
目前，所有的普通串口下载烧录编程都是采用明码通信的（电脑和目标芯片通信时，或脱机下载板和目标芯片通信时），问题：如果烧录人员通过分析下载烧录编程时串口通信的数据，高手是可以在烧录时在串口上引 2 根线出来，通过分析串口通信的数据分析出实际的用户程序代码的。当然用 STC 的脱机下载板烧程序总比用电脑烧程序强（防止烧录人员将程序轻易从电脑盗走，如通过网络发走，如通过 U 盘烤走，防不胜防，当然盗走你的电脑那就没办法那，所以 STC 的脱机下载工具比电脑烧录安全，让前台文员小姐烧，让老板娘烧都可以）。即使是 STC 全球首创的脱机下载工具，对于要防止天才的不法分子在脱机下载工具烧录的过程中通过分析串口通信的数据分析出实际的用户程序代码，也是没有办法达到要求的，这就需要用到最新的 STC 单片机所提供的程序加密后传输功能。

程序加密后传输下载是用户先将程序代码通过自己的一套专用密钥进行加密，然后将加密后的代码再通过串口下载，此时下载传输的是加密文件，通过串口分析出来的是加密后的乱码，如不通过派人潜入你公司盗窃你电脑里面的加密密钥，就无任何价值，便可起到防止在烧录程序时被烧录人员通过监测串口分析出代码的目的。

程序加密后传输功能的使用需要如下的几个步骤：

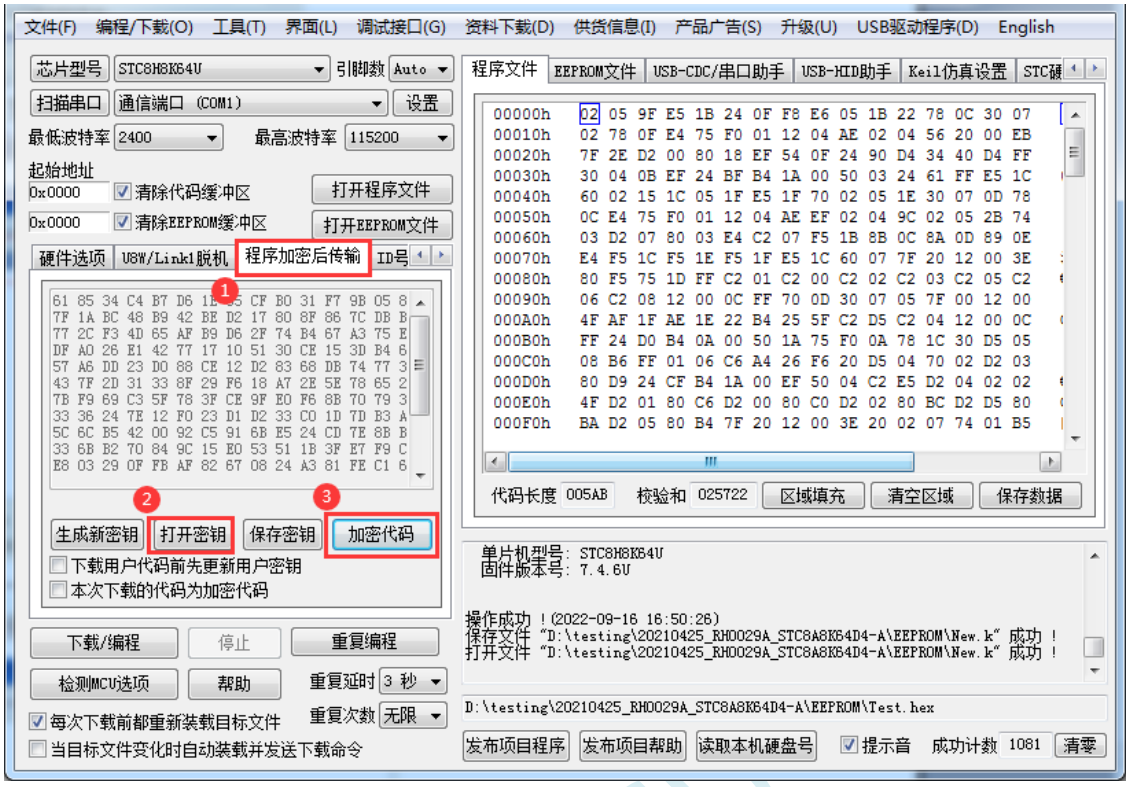
1、生成并保存新的密钥

如下图，进入到“程序加密后传输”页面，点击“生成新密钥”按钮，即可在缓冲区显示新生成的 256 字节的密钥。然后点击“保存密钥”按钮，即可将生成的新密钥保存为以“.K”为扩展名的密钥文件（**注意：这个密钥文件一定要保存好，以后发布的代码文件都需要使用这个密钥加密，而且这个密钥的生成是非重复的，即任何时候都不可能生成两个完全相同的密钥，所以一旦密钥文件丢失将无法重新获得**）。例如我们将密钥保存为“New.k”。

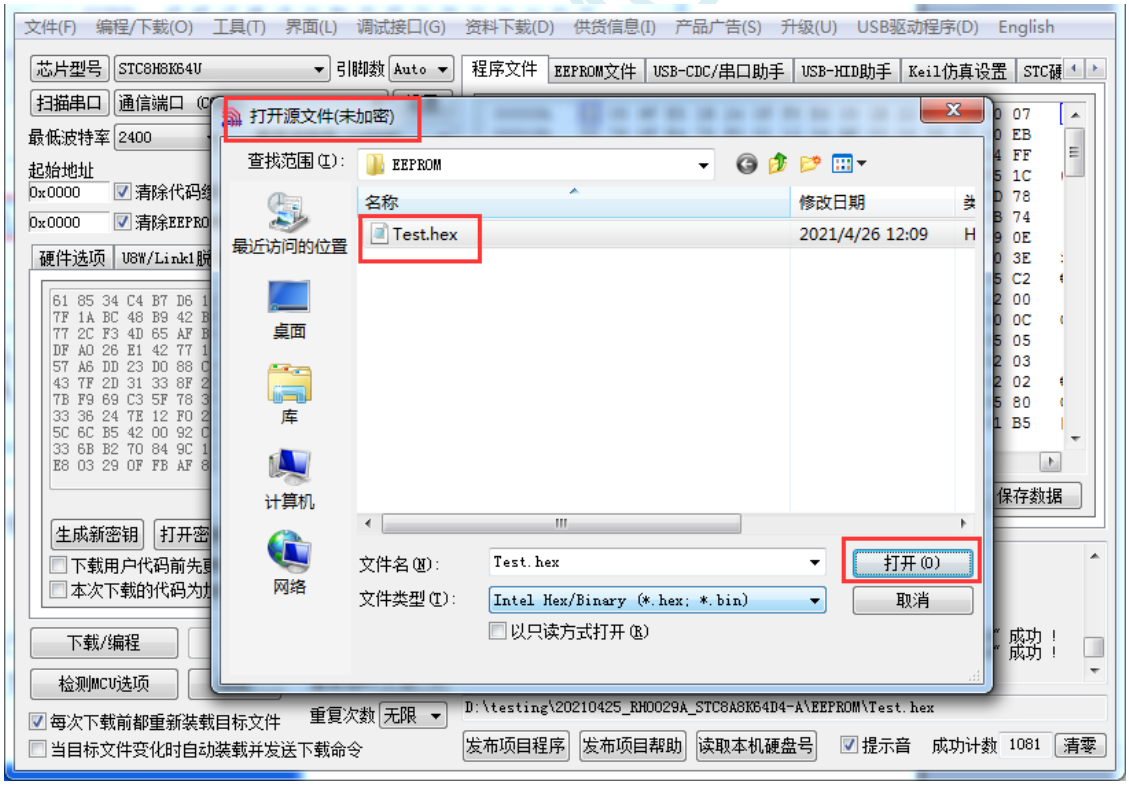


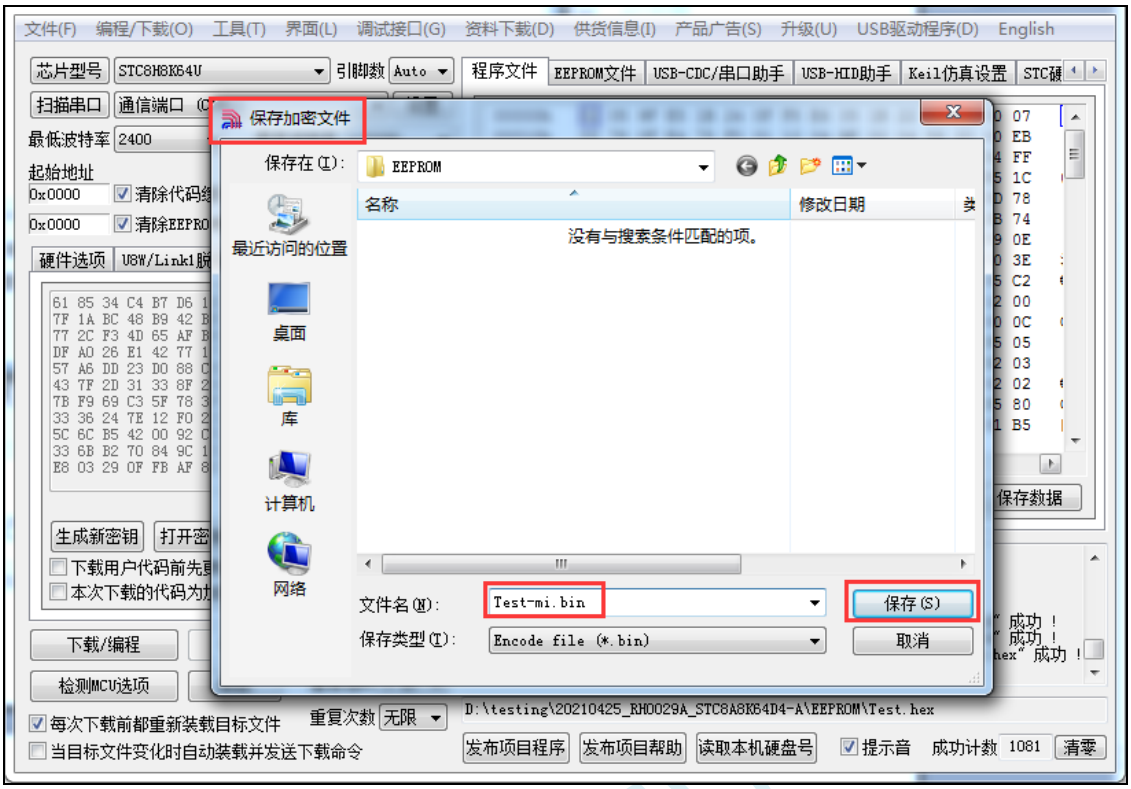
2、对代码文件加密

加密文件前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“程序加密后传输”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，例如“New.k”，然后返回到“程序加密后传输”页面中点击“加密代码”按钮，如下图所示，首先会弹出“打开源文件（未加密）”的对话框，此时选择的是原始的未加密的代码文件



点击打开按钮后，马上有会弹出一个类似的对话框，但此时是对加密后的文件进行保存的对话框。如下图所示，点击保存按钮即可保存加密后的文件。





3、将用户密钥更新到目标芯片中

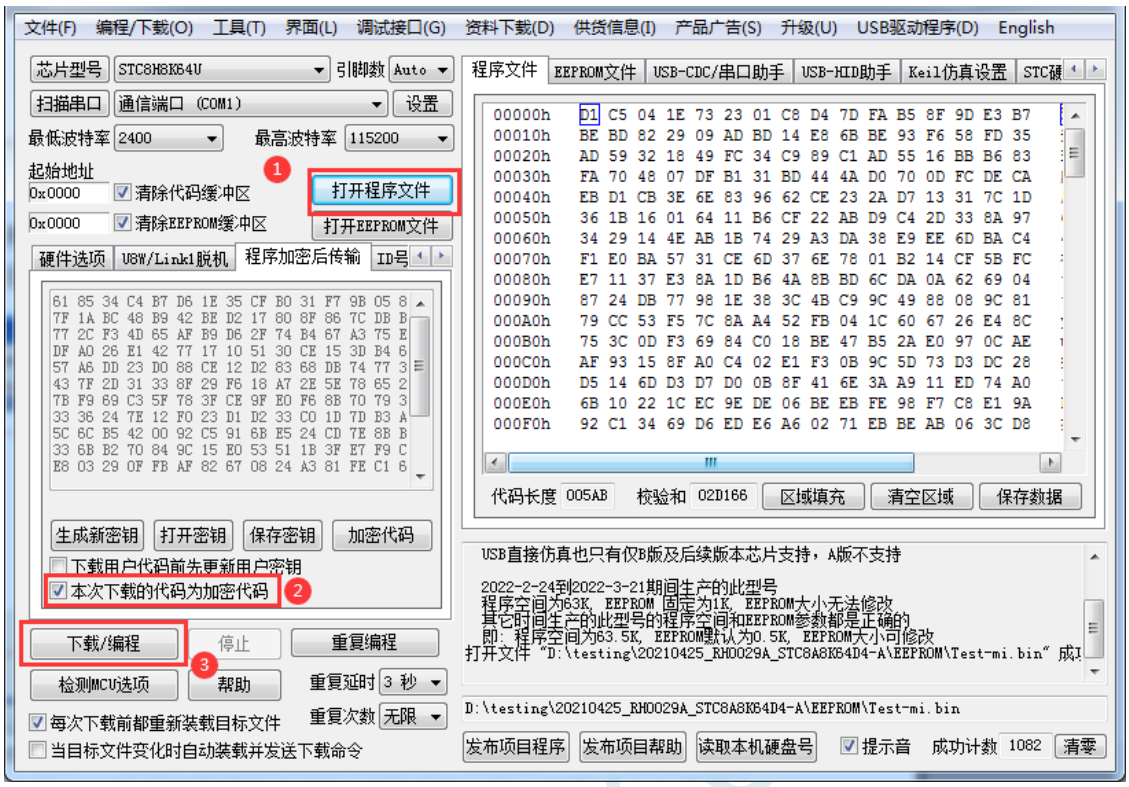
更新密钥前，需要先打开我们自己的密钥。若缓冲区中存放的已经是我们的密钥，则不要再打开。如下图，在“自定义加密下载”页面中点击“打开密钥”按钮，打开我们之前保存的密钥文件，例如“New.k”。密钥打开后，如下图所示，勾选上“下载用户代码前先更新用户密钥”选项和“本次下载的代码为加密代码”的选项，然后打开我们之前加密过后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载完成即可更新用户密钥。



4、加密更新用户代码

密钥更新成功后，目标芯片便具有接收加密代码并还原的功能。此时若需要再次升级/更新代码，

则只需要参考第二步的方法，将目标代码进行加密，然后如下图



对于一片新的 STC 单片机，可将步骤 3 和步骤 4 合并完成，即将密钥更新到目标单片机的同时也可将加密后的代码一并下载到单片机中，若已经执行过步骤 3（即已经将密钥更新到目标芯片中了），则后续的代码更新就只需要按照步骤 4，只需要在“程序加密后传输”页面中选择“本次下载的代码为加密代码”的选项（“下载用户代码前先更新用户密钥”选项不需要选了），然后打开我们之前加过密后的文件，打开后点击界面左下角的“下载/编程”按钮，按正常方式对目标芯片下载即可完成用用户自己专用的加密文件更新用户代码的目的（防止在烧录程序时被烧录人员通过监测串口分析出代码的目的）。

H.3　发布项目程序+程序加密后传输结合使用

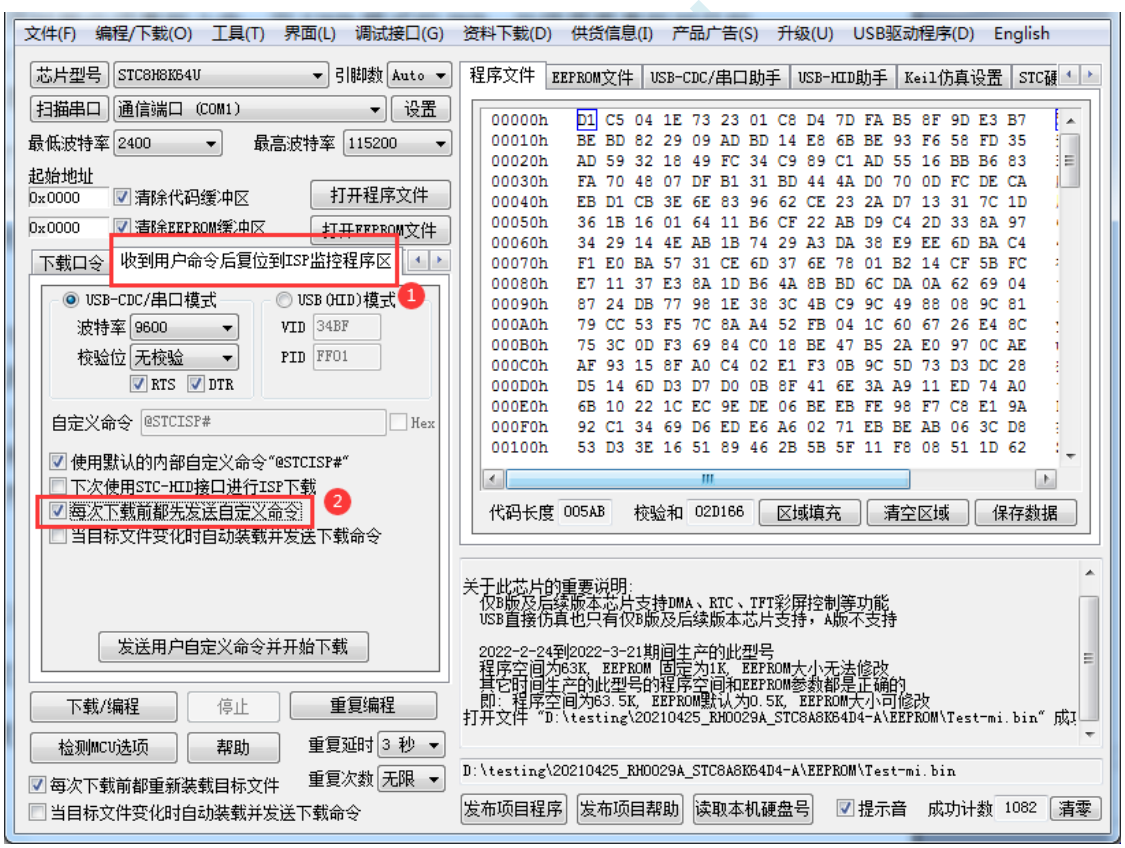
发布项目程序与程序加密后传输两项新的特殊功能可以结合在一起使用。首先程序加密后传输可以确保用户代码在烧录编程时串口通信传输过程当中的保密性，而发布项目程序可实现让最终使用者远程升级功能（方案公司的人员不需要亲自到场）。所以两项功能结合起来使用，非常适用于方案公司/生产商在软件需要更新时，让最终使用者自己对终端产品进行软件更新的目的，又确保现场烧录人员无法通过串口分析出有用程序，强烈建议方案公司使用。

H.4　用户自定义下载（实现不停电下载）

将用户的目标程序下载到STC单片机是通过执行单片机内部的ISP系统代码和上位机进行串口或者USB通讯来实现的。但STC单片机内部的ISP系统代码只有在每次重新停电再上电时才会被执行，这就要求用户每次需要对目标单片机更新程序时就必须重新上电，而USB模式的ISP，处理需要重新对目标芯片上电外，还需要在上电时将P3.2口下拉到GND。对于处于开发阶段的项目，需要频繁的修改代码、更新代码，每次下载都需要重新上电会导致操作非常麻烦。

STC单片机在硬件设计时，增加了一个软复位寄存器（IAP_CONTR），让用户可以通过设置此寄存器来决定CPU复位后重新执行用户代码还是复位到ISP区执行ISP系统代码。当向IAP_CONTR寄存器写入0x20时，CPU复位后重新执行用户代码；当向IAP_CONTR寄存器写入0x60时，CPU复位后复位到ISP区执行ISP系统代码。

要实现不停电进行ISP下载，用户可以在程序中设计一段代码，例如检测一个特殊的按键、或者监控串口等待一个特殊的串口命令，当检测到满足下载条件时，就通过软件触发软复位寄存器复位到ISP区执行ISP系统代码，从而实现不停电ISP下载。当触发条件是外部按键时，则在用户代码中实时监控按键状态即可。若要实现AIapp-ISP软件和用户触发软复位完全同步，则需要使用AIapp-ISP软件中所提供的“收到用户命令后复位到ISP监控程序区”这个功能。



实现不停电 ISP 下载的步骤如下：

1、编写用户代码，并在用户代码中添加串口命令监控程序

（参考代码如下，测试单片机型号为 STC8H8K64U）

```
#include "stc8h.h"

#define    FOSC        11059200UL
#define    BAUD        (65536 - FOSC/4/115200)

char code *STCISPCMD = "@STCISP#";          //自定义下载命令
char index;

void uart_isr()   interrupt 4
{
    char dat;

    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;
        dat  = SBUF;                        //接收串口数据

        if (dat == STCISPCMD[index])        //判断接收的数据和当前的命令字符是否匹配
        {
            index++;                        //若匹配则索引+1
            if (STCISPCMD[index] == '\0')   //判断命令是否配完成
                IAP_CONTR = 0x60;           //若匹配完成则软复位到 ISP
        }
        else
        {
            index = 0;                      //若不匹配,则需要从头开始
            if (dat == STCISPCMD[index])
                index++;
        }
    }
}

void main()
{
    P0M0 = 0x00; P0M1 = 0x00;
    P1M0 = 0x00; P1M1 = 0x00;
    P2M0 = 0x00; P2M1 = 0x00;
```

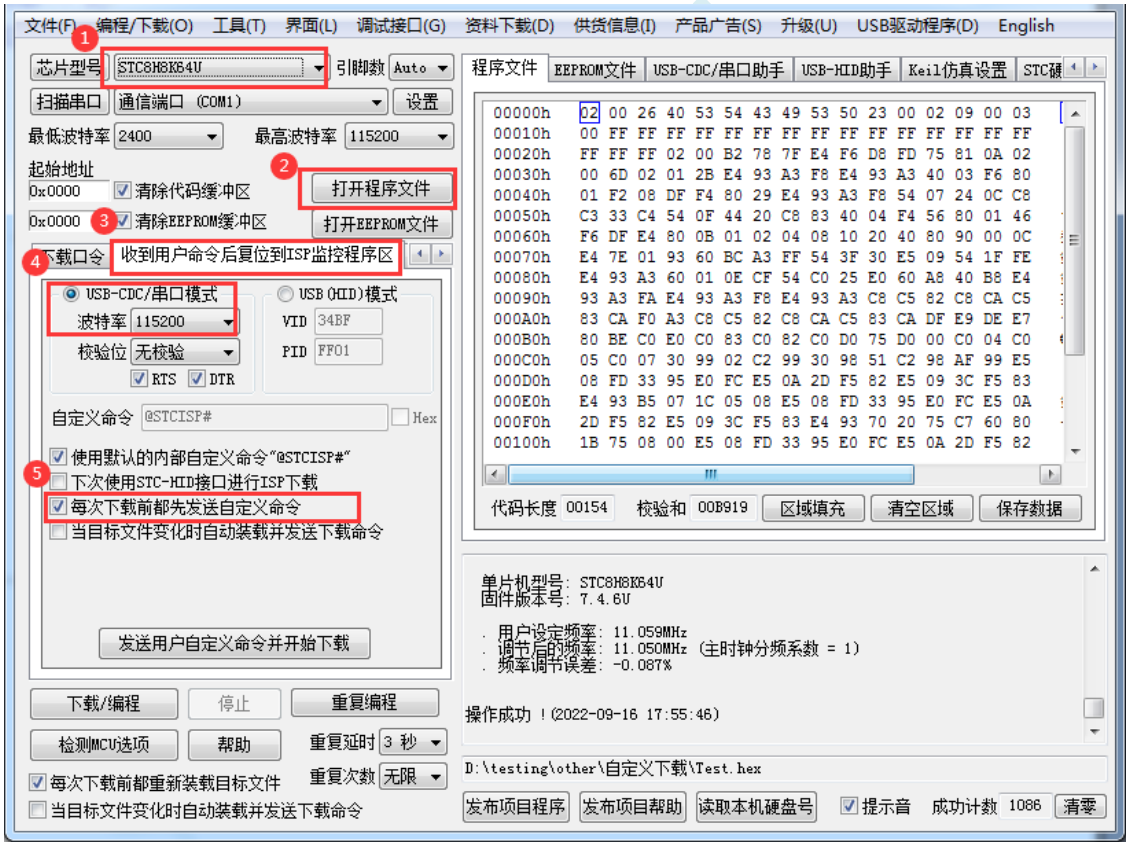
```
    P3M0 = 0x00; P3M1 = 0x00;

    SCON = 0x50;                    //串口初始化
    AUXR = 0x40;
    TMOD = 0x00;
    TH1 = BAUD >> 8;
    TL1 = BAUD;
    TR1 = 1;
    ES = 1;
    EA = 1;

    index = 0;                      //初始化命令

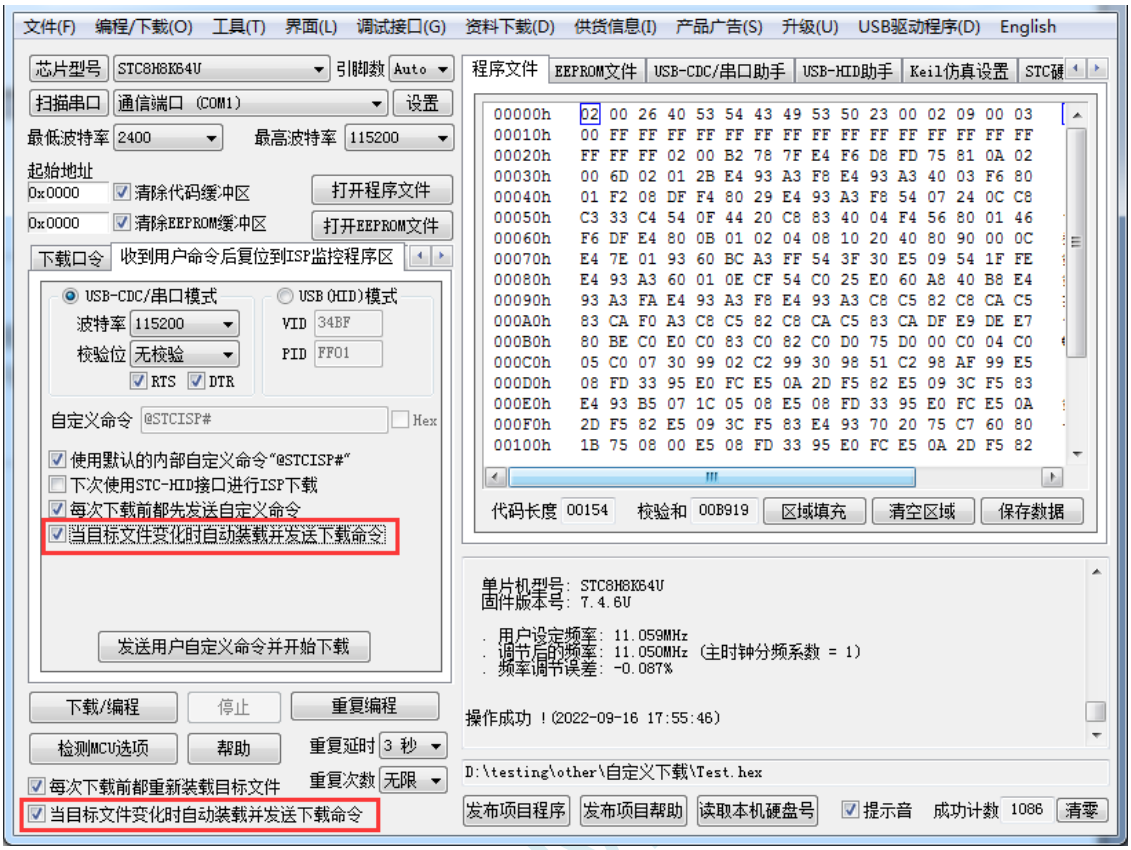
    while (1);
}
```

2、按下图所示的步骤进行设置自定义下载命令（范例使用 STC 默认命令“@STCISP#”）



3、第一次下载时需要对目标单片机重新上电，之后的每次更新只需要点击下载软件中的“下载/编程”按钮，下载软件自动将下载命令发送给目标单片机，目标单片机接收到命令后自动复位到系统 ISP 区，即可实现不停电更新用户代码。

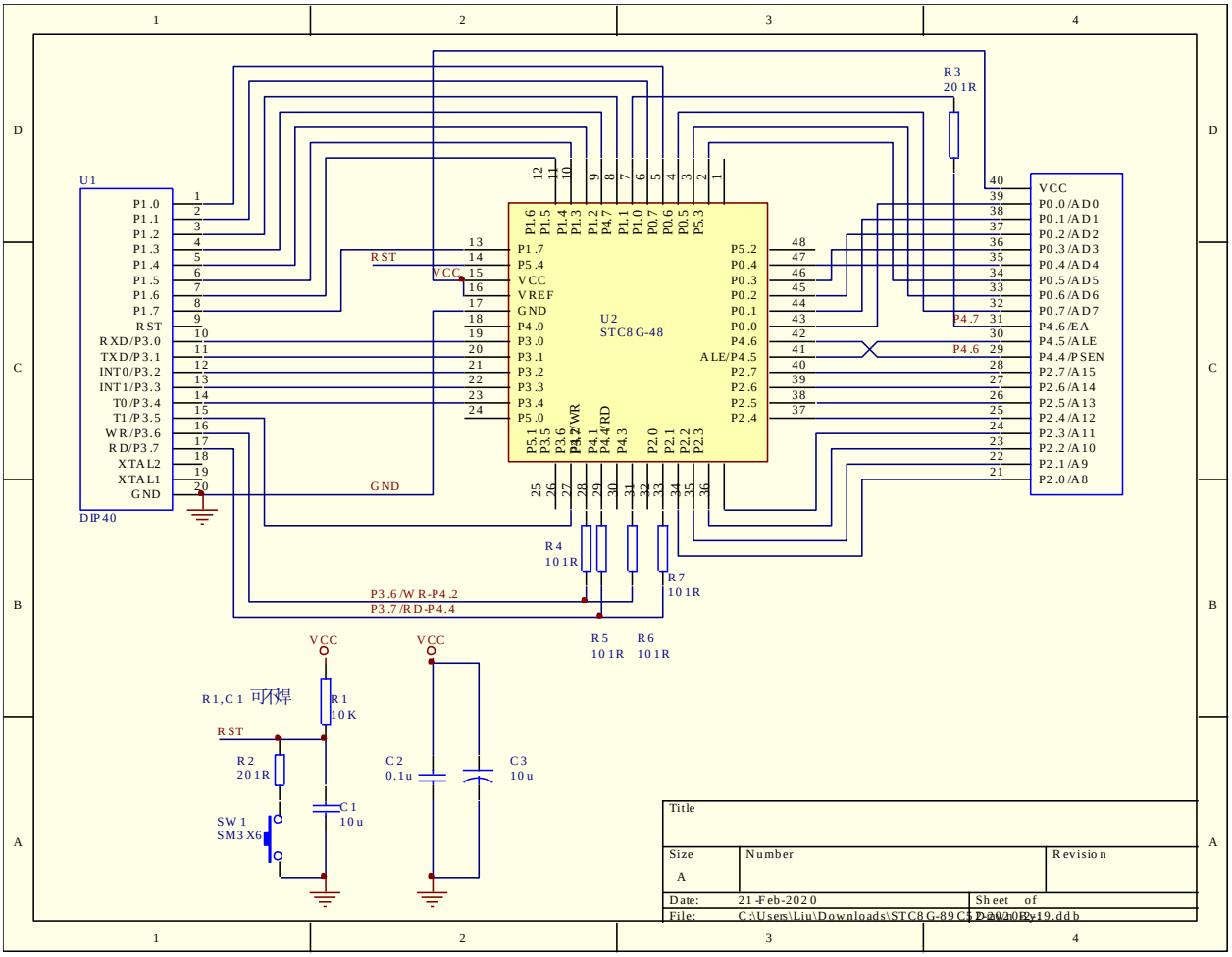
4、AIapp-ISP 还可实现项目开发阶段，完全自动下载功能，即当下载软件侦测到目标代码被更新了，就会自动发送下载命令。要实现这个功能只需要勾选下图中的两个选项中的任意一个即可

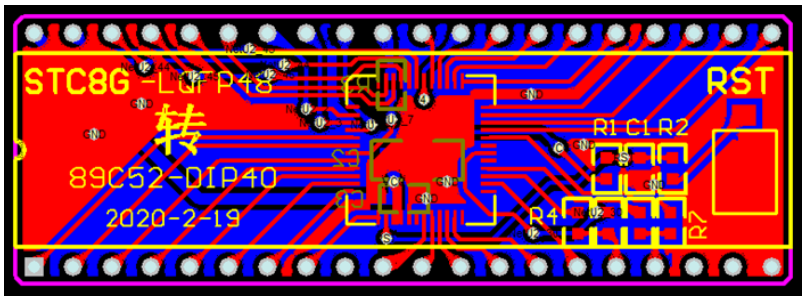


附录I　如何让传统的 8051 单片机学习板可仿真

传统的 8051 单片机学习板不具有仿真功能，让传统的 8051 单片机学习板可仿真需要借助转换板，转换板的实物图如下图所示，转换后的引脚排布与传统 8051 的脚位基本一致，从而可以实现标准 8051 学习板的仿真功能。

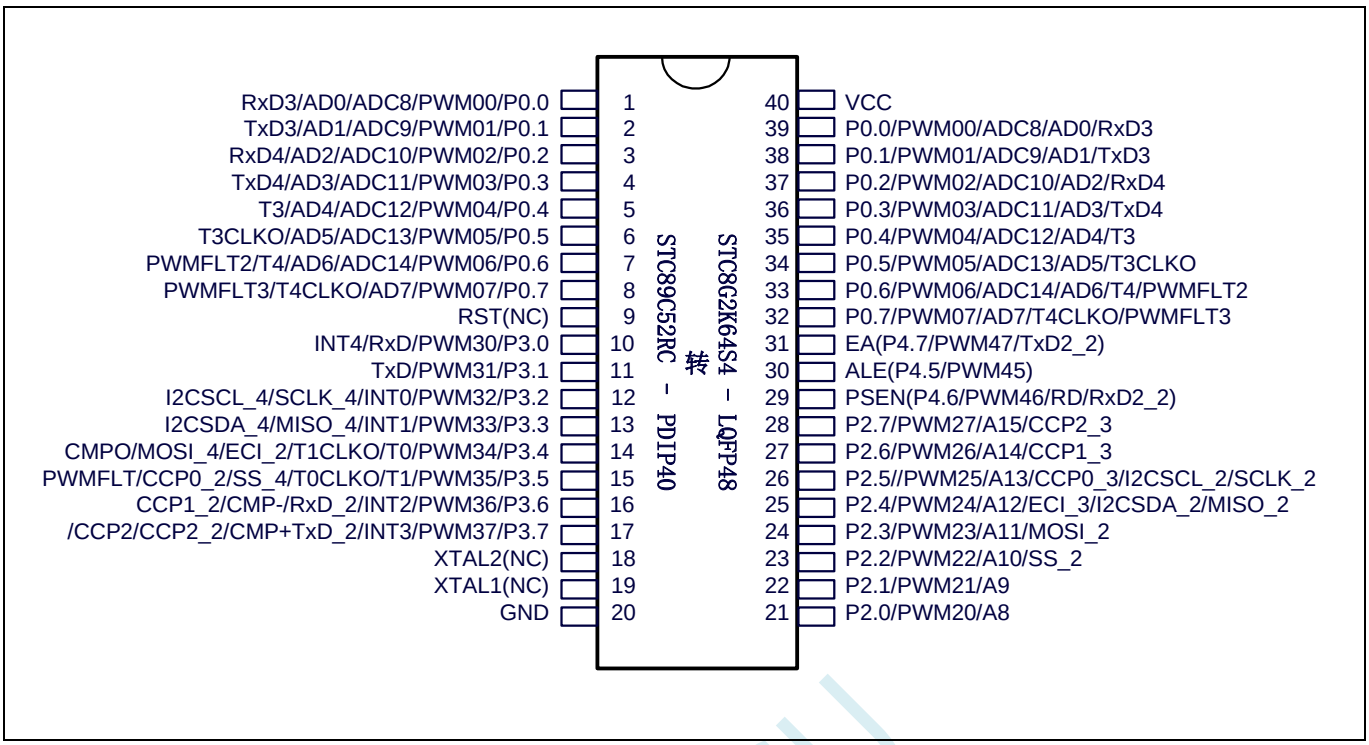
下图是转换板的原理图和 PCB 板图





该转换板可进行 STC8C 系列 LQFP48 转 STC89C52RC/STC89C58RD+系列仿真用。

下图为转换板功能示意图



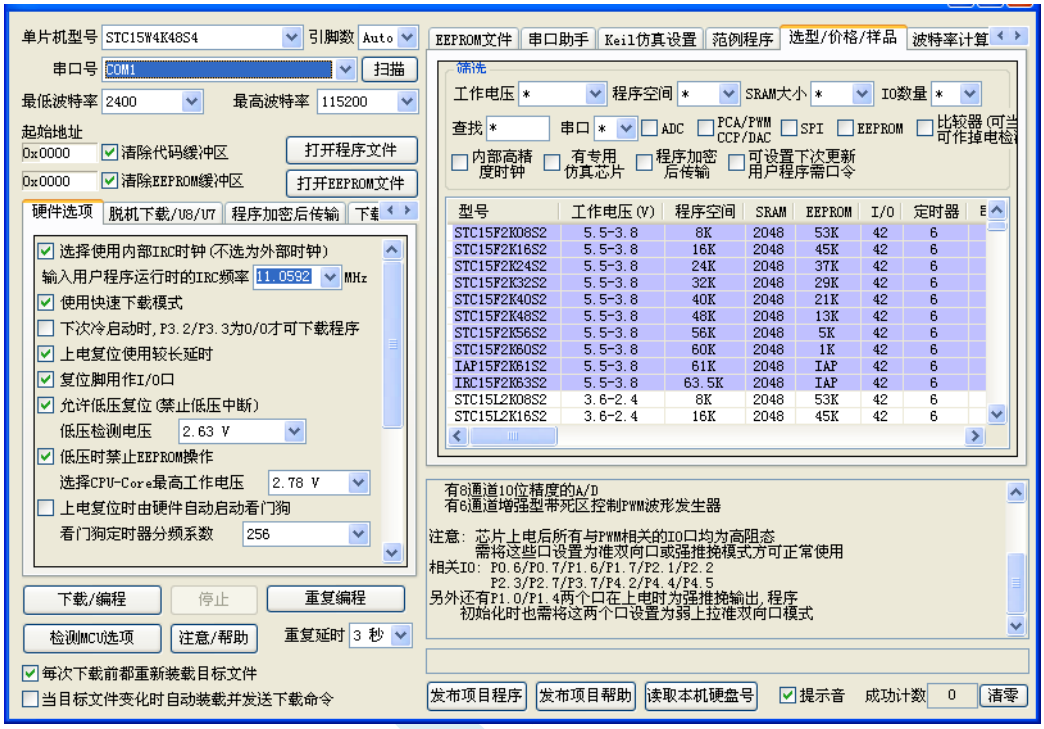
注意：

- ✓ 由于内置高精准 R/C 时钟，因此不需要外部晶振,XTAL1 和 XTAL2 是空的
- ✓ WR 和 RD 是(WR/P4.2 和 RD/P4.4)，而不是传统的(WR/P3.6 和 RD/P3.7)。
  （转换板中，P4.2 与 P3.6 连接在一起，P4.4 与 P3.7 连接在一起。当用户需要用此转换板访问外部总线时，需要将 P3.6 和 P3.7 设置为高阻输入模式，从而使 P4.2 和 P4.4 正常输出总线读写信号；若不需要访问外部总线，则需将 P4.2 和 P4.4 设置高阻输入模式，3.6 和 P3.7 即为普通 I/O。）
- ✓ 由于 STC8C 系列 MCU 是低电平复位，与传统 8051 的高电平复位不兼容，因此 RST 管脚是悬空，而用转换板上的复位按键加复位电路取代

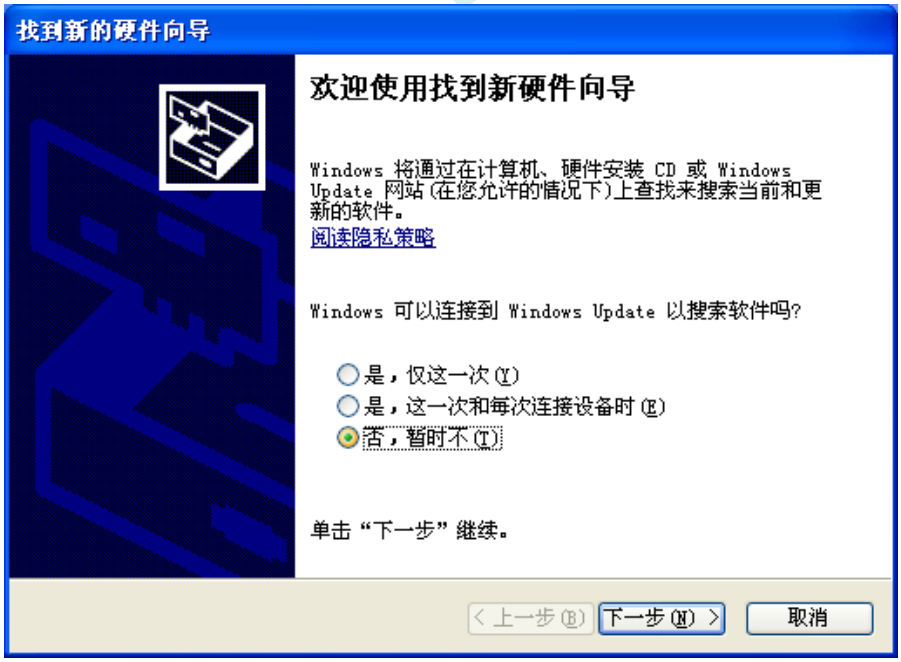
附录J　STC-USB 驱动程序安装说明

Windows XP 安装方法

打开 V6.79 版（或者更新的版本）的 AIapp-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录



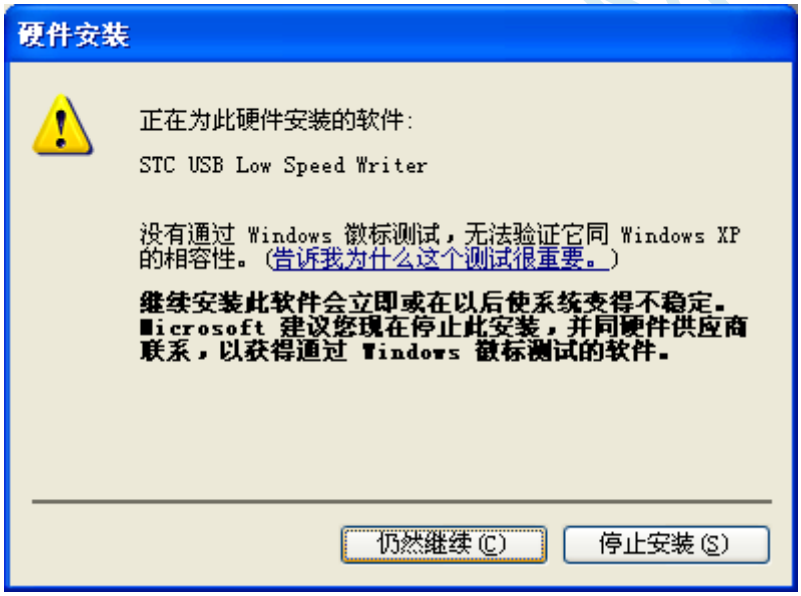
插入 USB 设备，系统找到设备后自动弹出如下对话框，选择其中的“否，暂时不”项



在下面的对话框中选择“自动安装软件(推荐)”项



在弹出的下列对话框中，选择“仍然继续”按钮



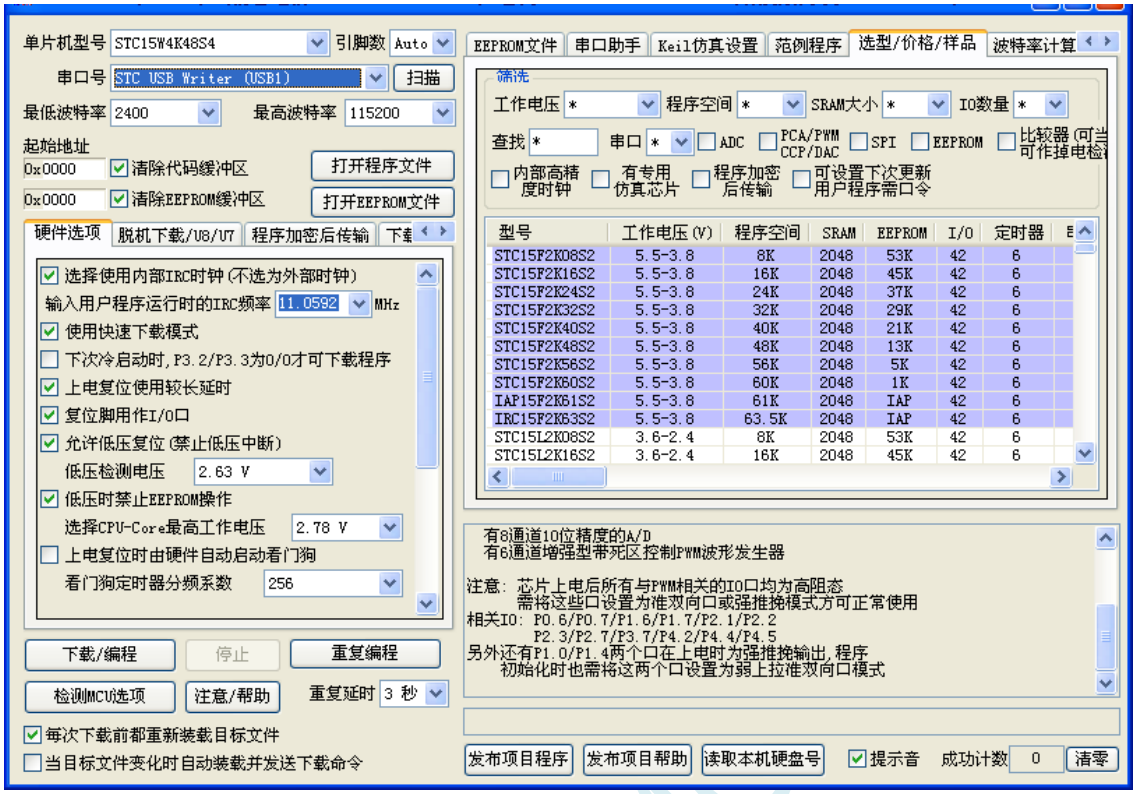
接下系统会自动安装驱动，如下图



出现下面的对话框表示驱动安装完成

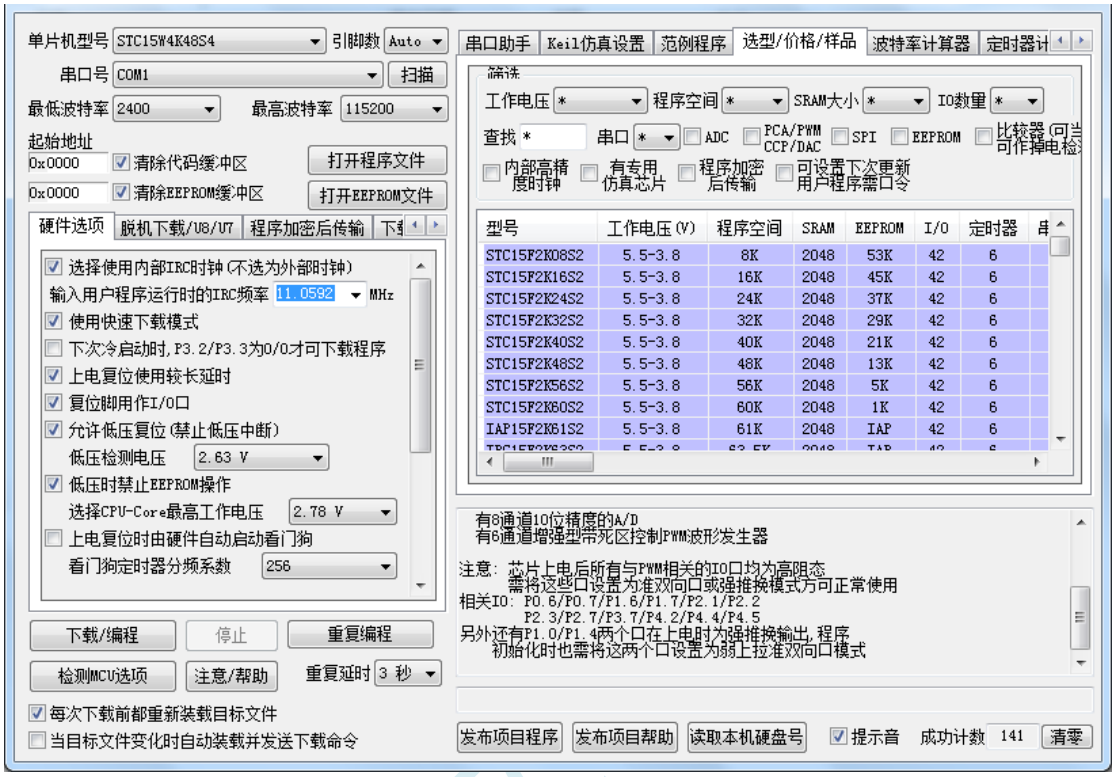


此时，之前打开的 AIapp-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)” ，如下图：

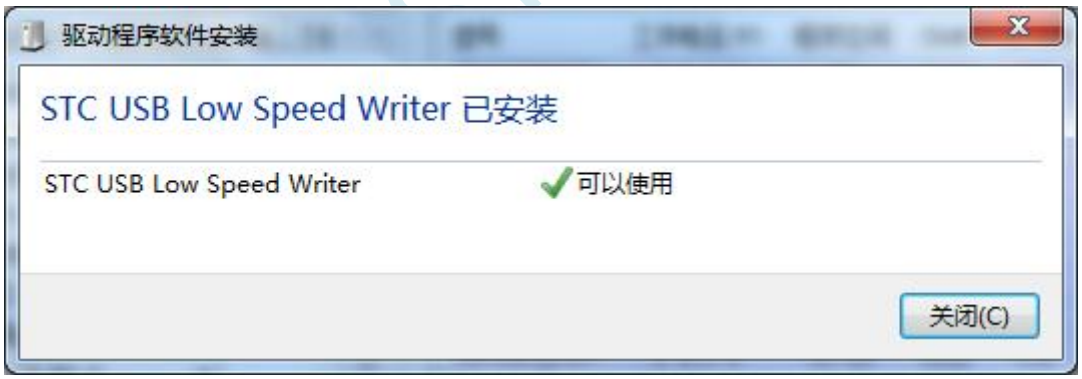


Windows 7（32 位）安装方法

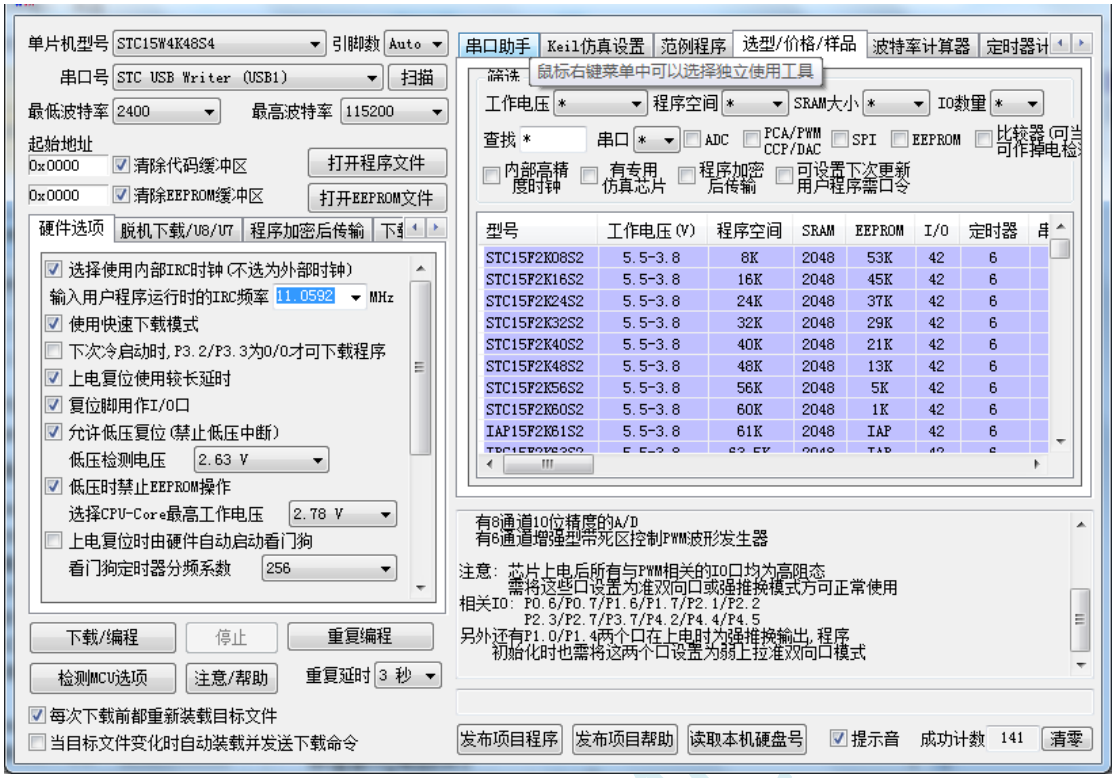
打开 V6.79 版（或者更新的版本）的 AIapp-ISP 下载软件，下载软件会自动将驱动文件复制到相关的系统目录



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 AIapp-ISP 下载软件中的串口号列表会自动选择所插入的 USB 设备，并显示设备名称为“STC USB Writer (USB1)”，如下图：

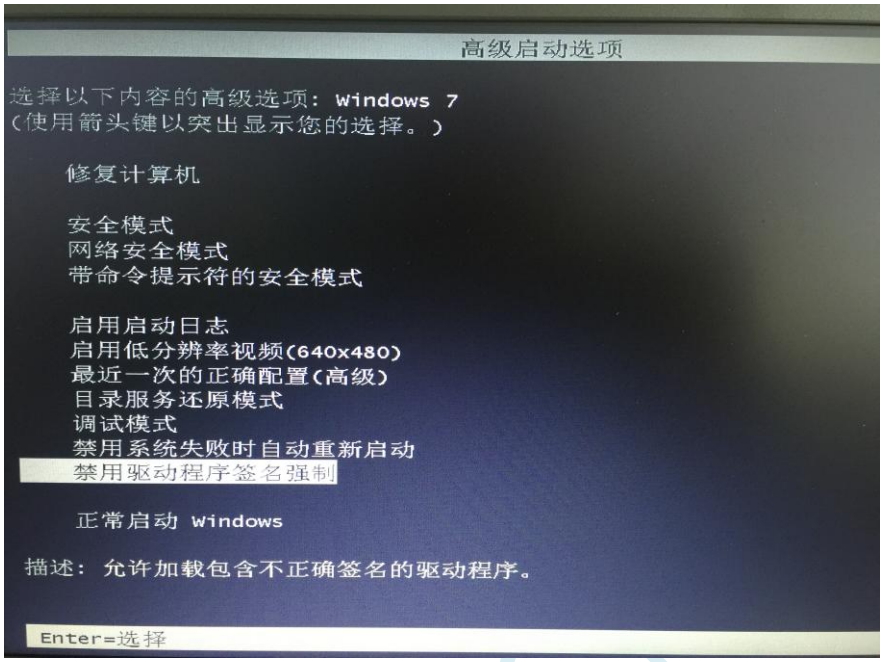


注：若 Windows 7 下，系统并没有自动安装驱动，则驱动的安装方法请参考 **Windows 8（32 位）的安装方法**

Windows 7（64 位）安装方法

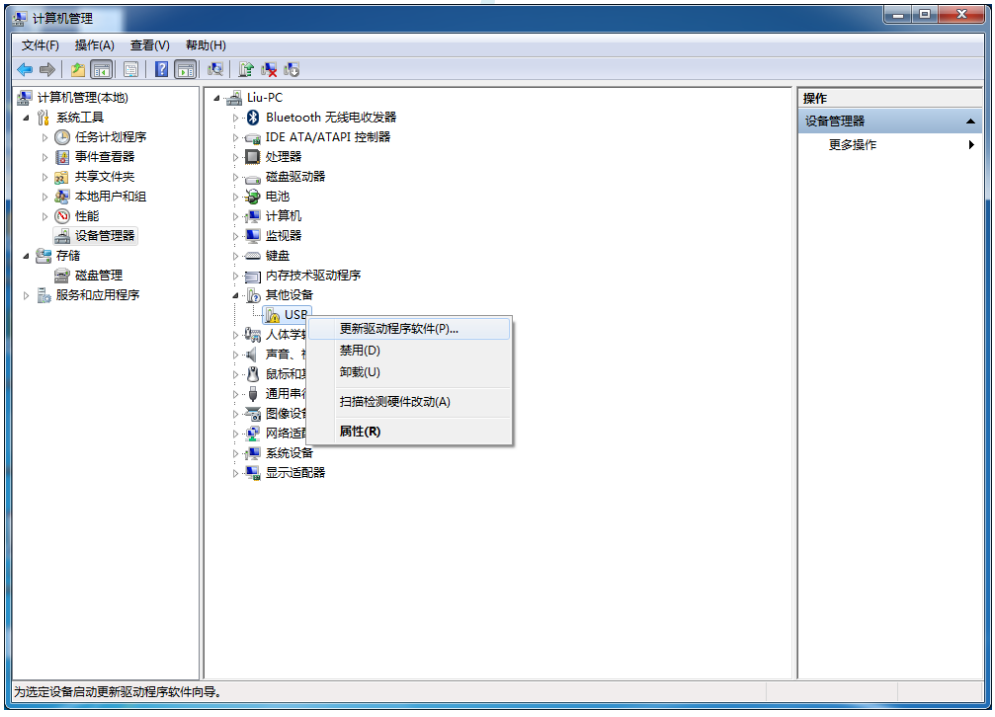
由于 Windows7 64 位操作系统在默认状态下，对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前，需要按照如下步骤，暂时跳过数字签名，即可顺利安装成功。

首先重启电脑，并一直按住 F8，直到出现下面启动画面



选择“禁用驱动程序签名强制”。启动后即可暂时关闭数字签名验证功能

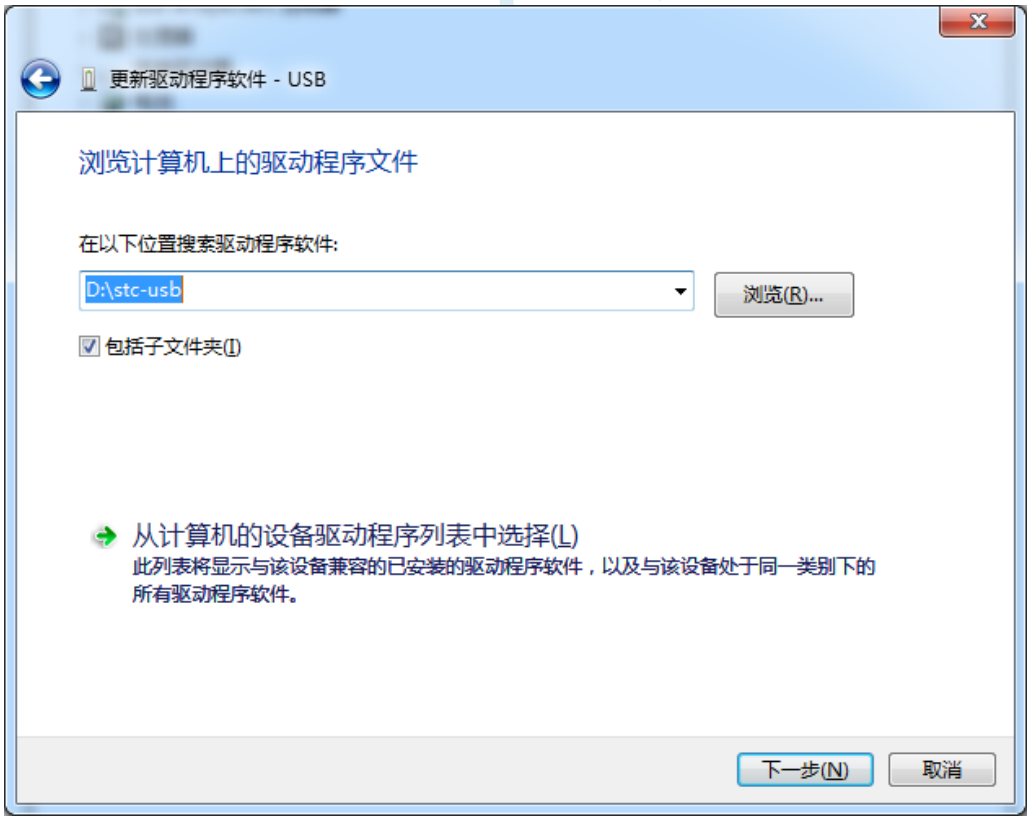
插入 USB 设备，并打开“设备管理器”。找到设备列表中带黄色感叹号的 USB 设备，在设备的右键菜单中，选择“更新驱动程序软件”



在下面的对话框中选择“浏览计算机以查找驱动程序软件”



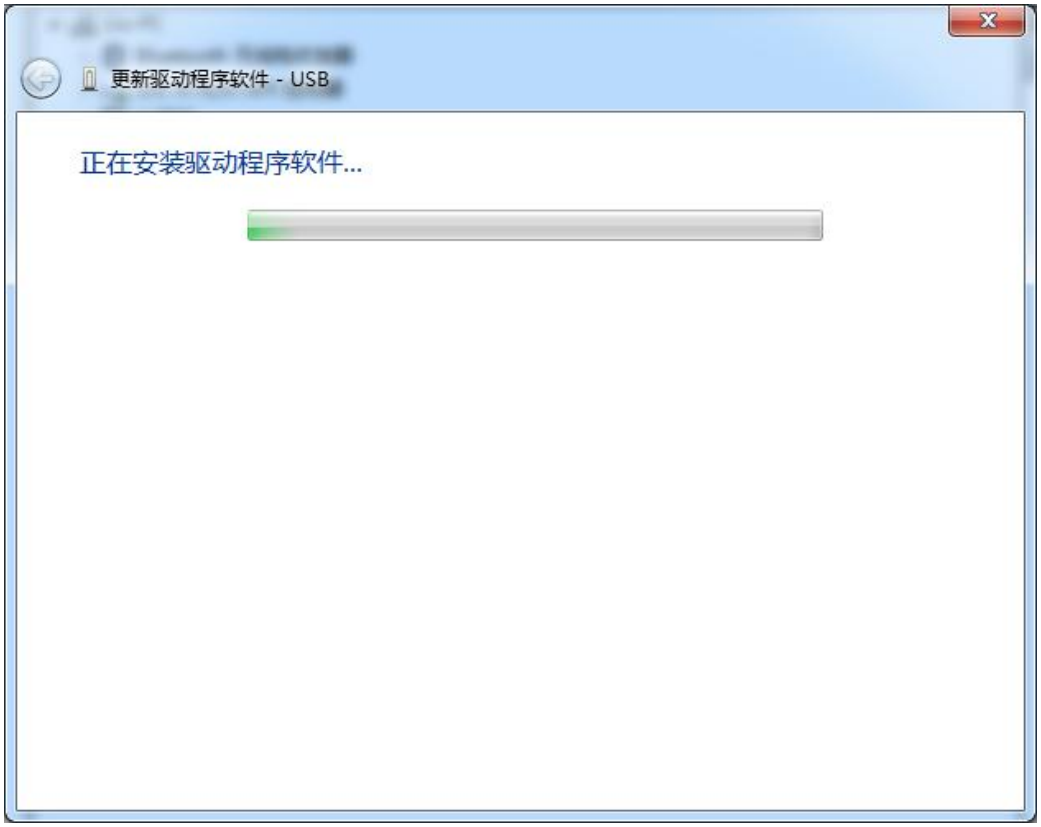
单击下面对话框中的“浏览”按钮，找到之前 STC-USB 驱动程序的存放目录（例如：之前的示例目录为 “ D:\STC-USB ”， 用 户 将 路 径 定 位 到 实 际 的 解 压 目 录 ）



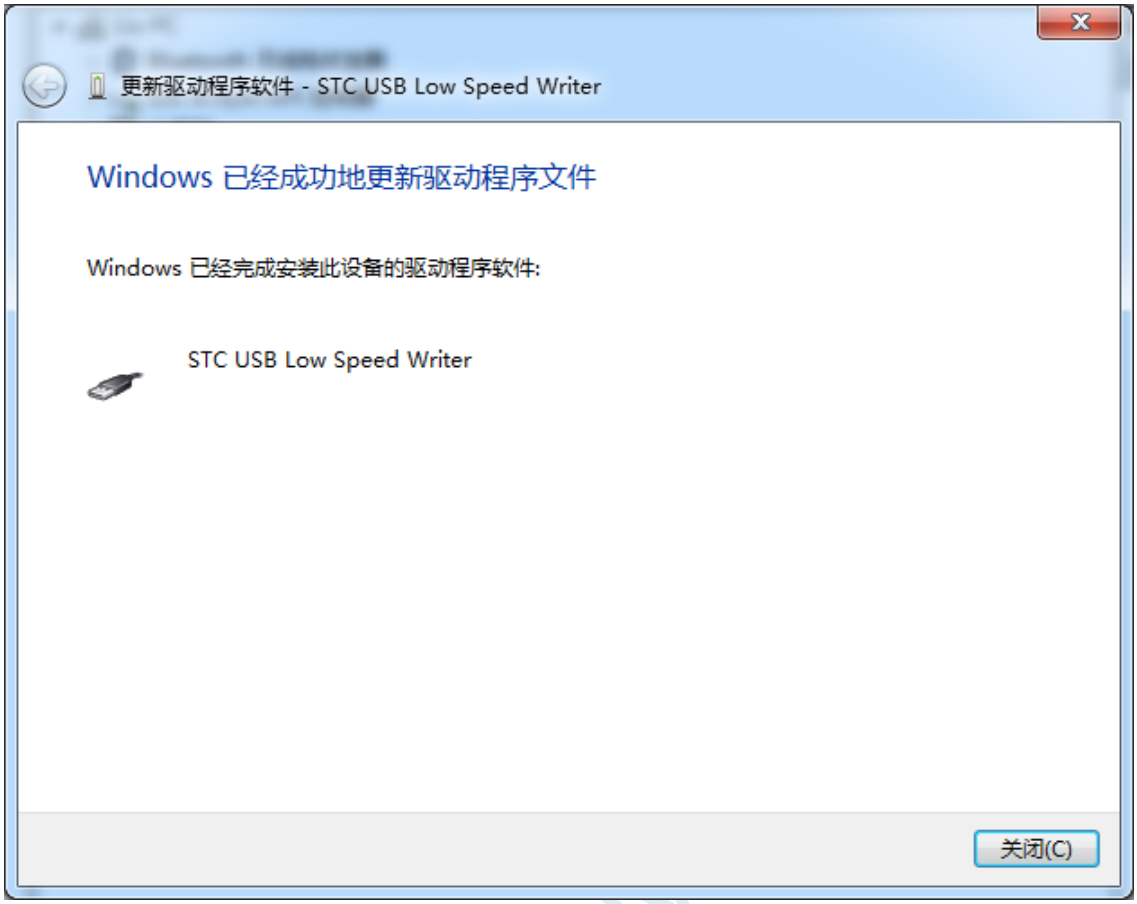
驱动程序开始安装时，会弹出如下对话框，选择“始终安装此驱动程序软件”



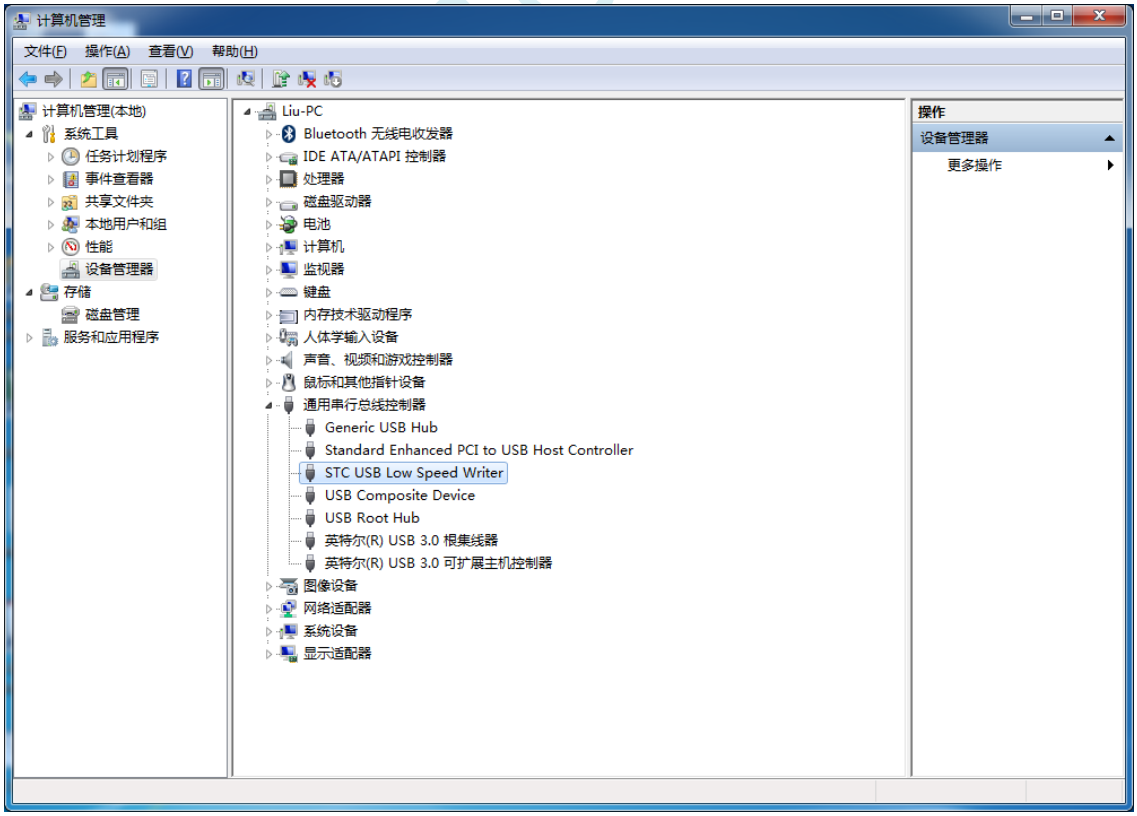
接下来，系统会自动安装驱动，如下图



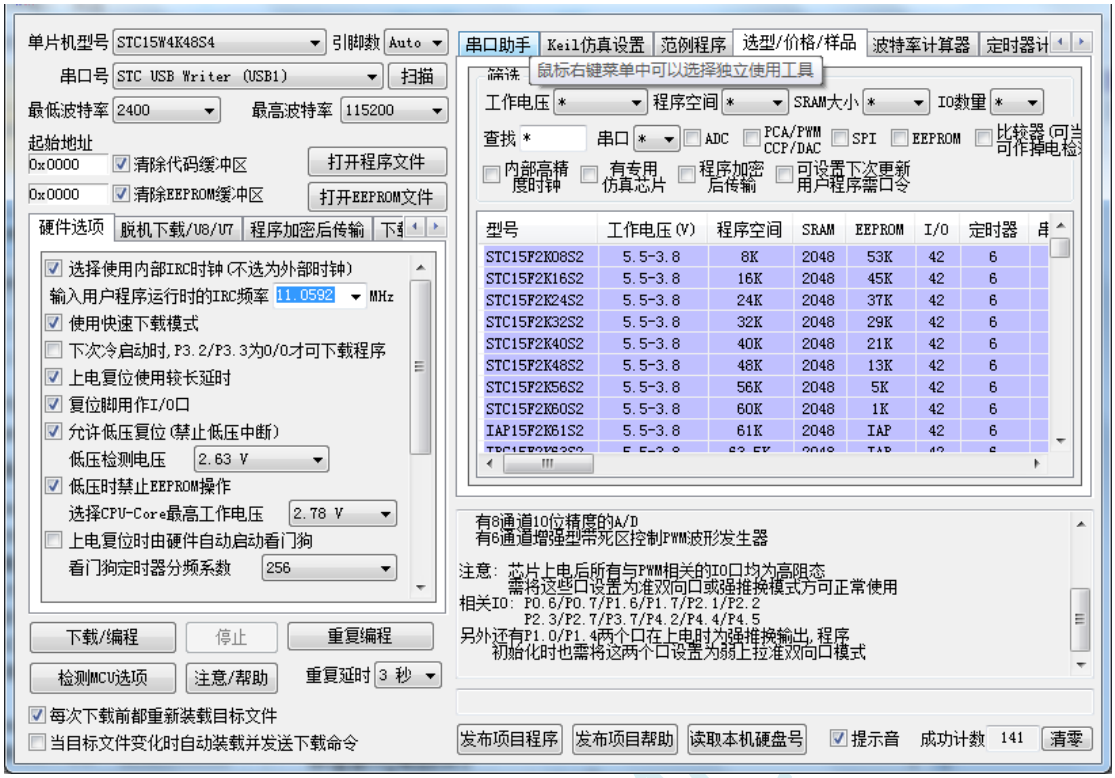
出现下面的对话框表示驱动安装完成



此时在设备管理器中，之前带有黄色感叹号的设备，此时会显示为“STC USB Low Speed Writer”的设备名



在之前打开的 AIapp-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：

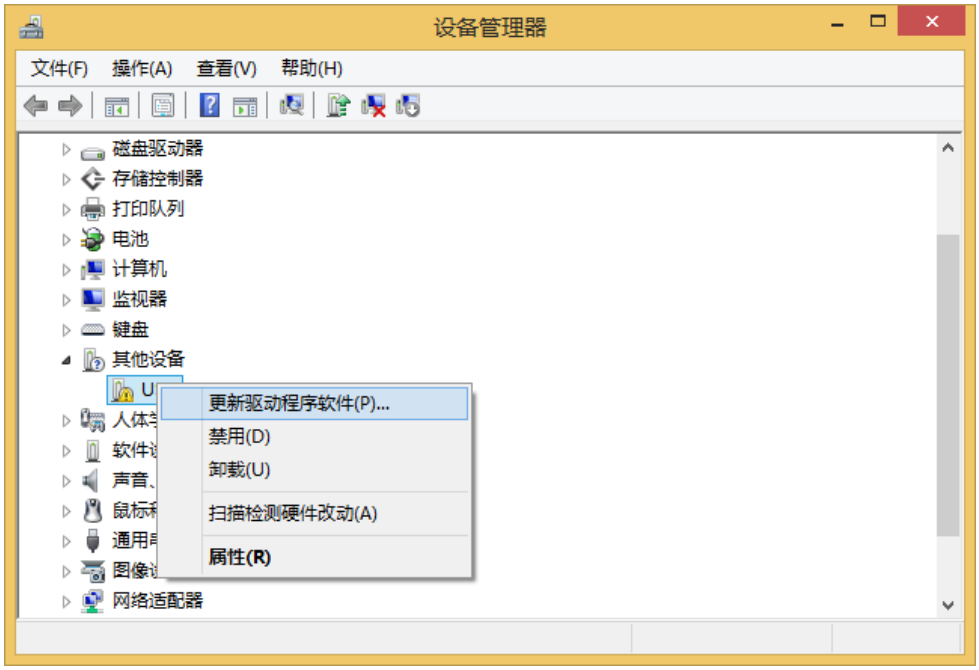


Windows 8（32 位）安装方法

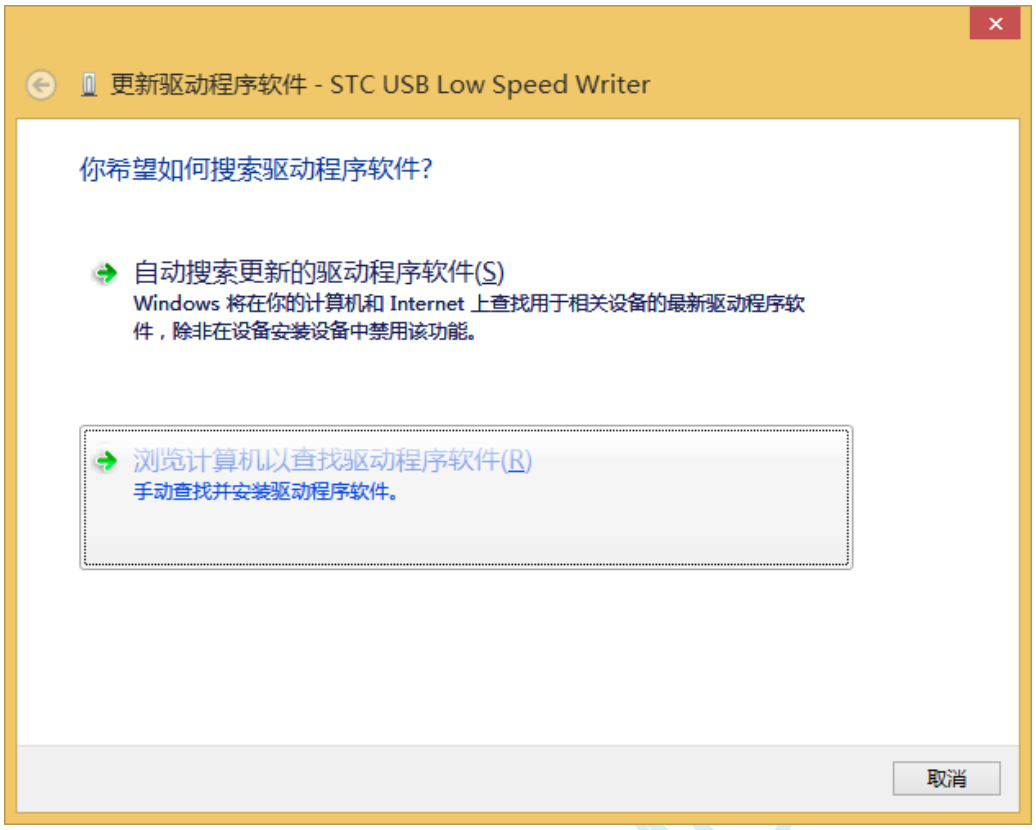
打开 V6.79 版（或者更新的版本）的 AIapp-ISP 下载软件（由于权限的原因，在 Windows 8 中下载软件不会将驱动文件复制到相关的系统目录，需要用户手动安装。首先从 STC 官方网站下载“stc-isp-15xx-v6.79.zip”（或更新版本），下载后解压到本地磁盘，则 STC-USB 的驱动文件也会被解压到当前解压目录中的“STC-USB Driver”中（例如将下载的压缩文件“stc-isp-15xx-v6.79.zip”解压到“F:\”，则 STC-USB 驱动程序在“F:\STC-USB Driver”目录中））



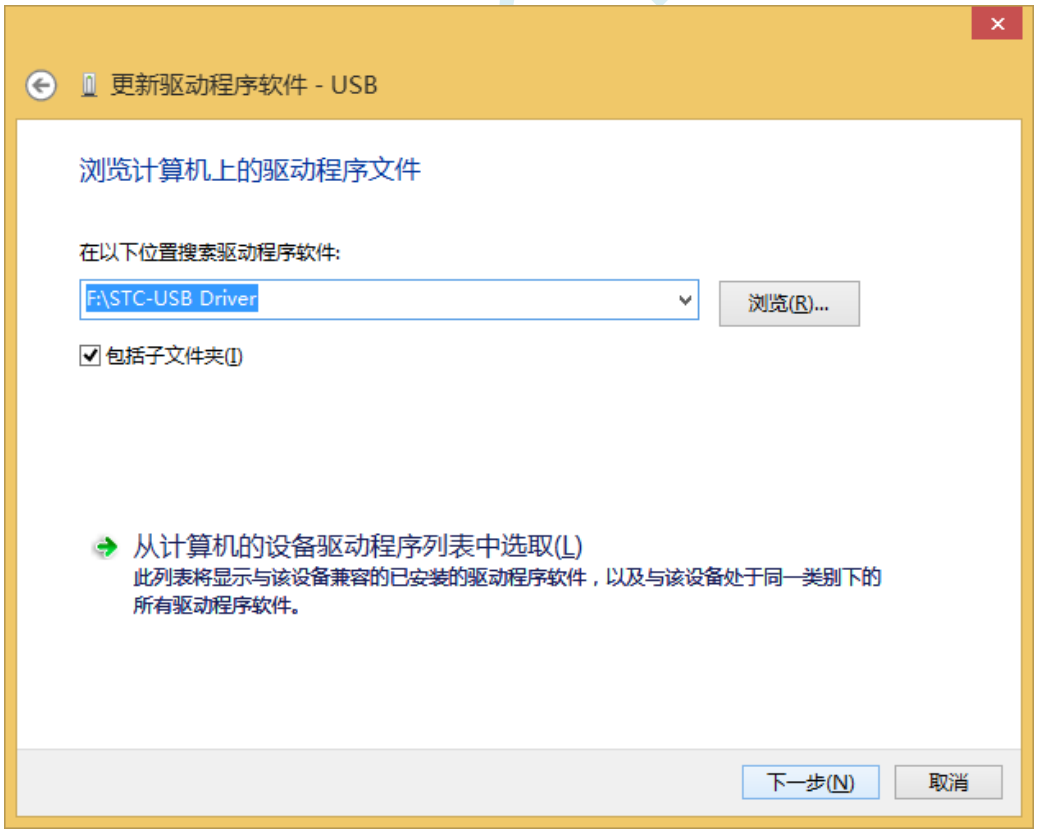
插入 USB 设备，并打开“设备管理器”。找到设备列表中带黄色感叹号的 USB 设备，在设备的右键菜单中，选择“更新驱动程序软件”



在下面的对话框中选择“浏览计算机以查找驱动程序软件”



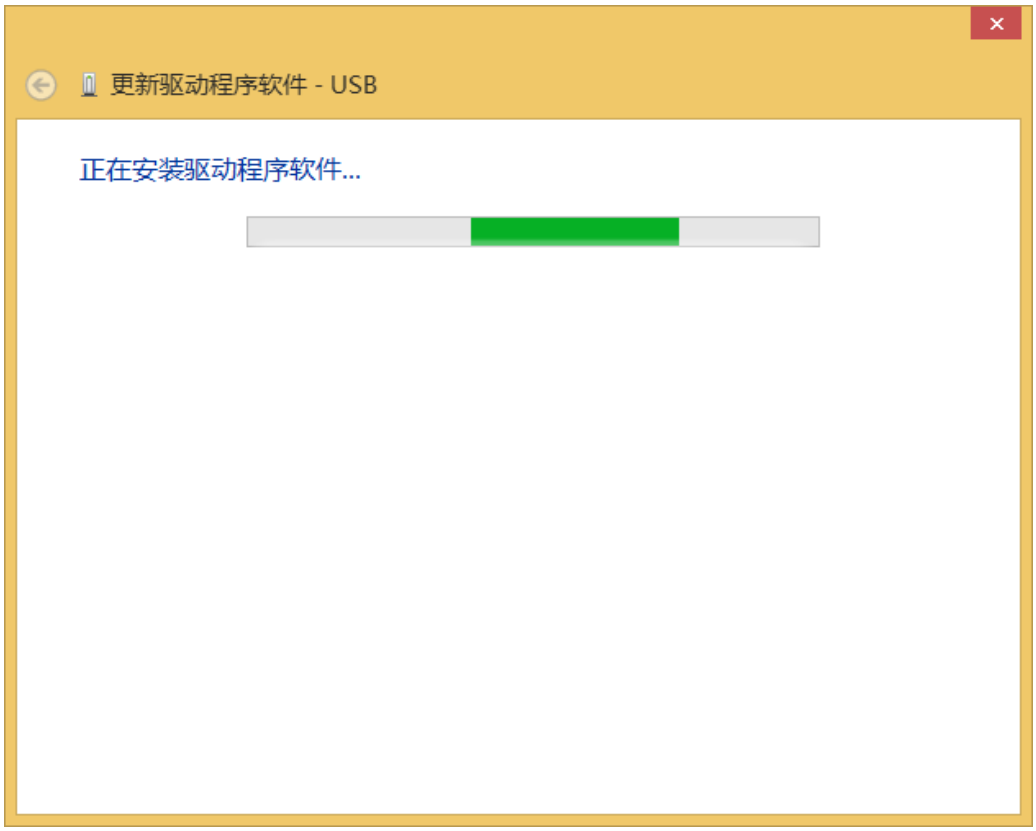
单击下面对话框中的“浏览”按钮，找到之前 STC-USB 驱动程序的存放目录（例如：之前的示例目录为“F:\STC-USB Driver”，用户将路径定位到实际的解压目录）



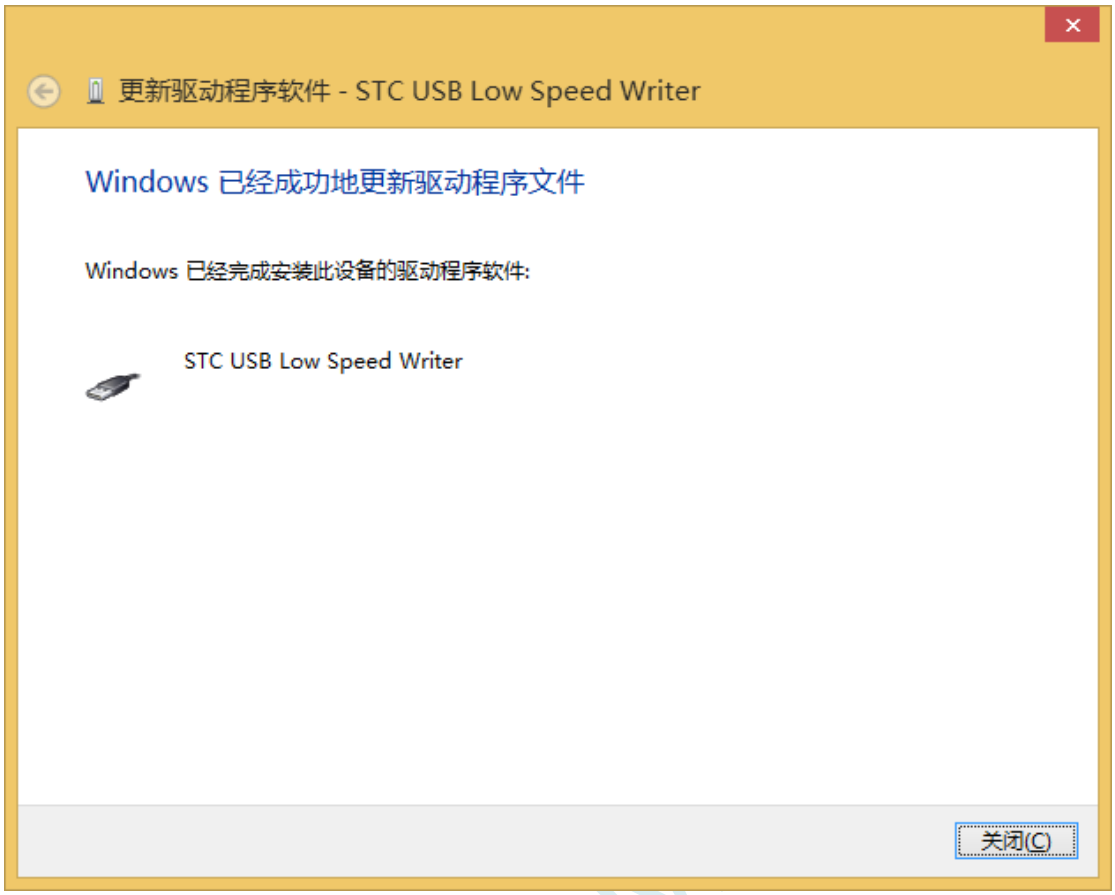
驱动程序开始安装时，会弹出如下对话框，选择“始终安装此驱动程序软件”



接下来，系统会自动安装驱动，如下图



出现下面的对话框表示驱动安装完成



此时在设备管理器中，之前带有黄色感叹号的设备，此时会显示为“STC USB Low Speed Writer”的设备名



在之前打开的 AIapp-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：



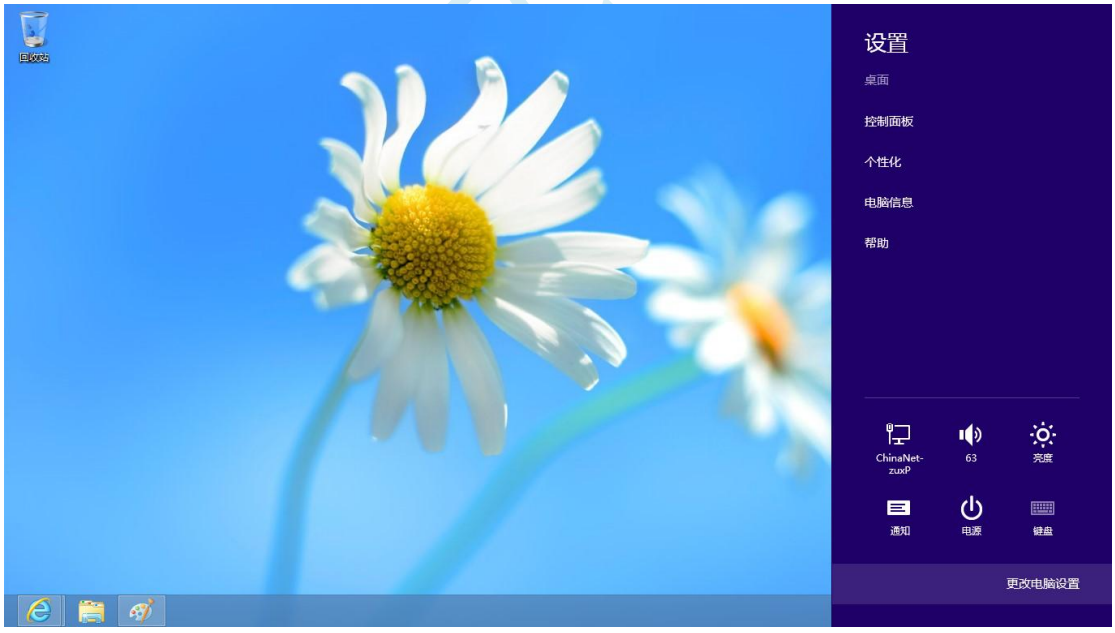
Windows 8（64 位）安装方法

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”



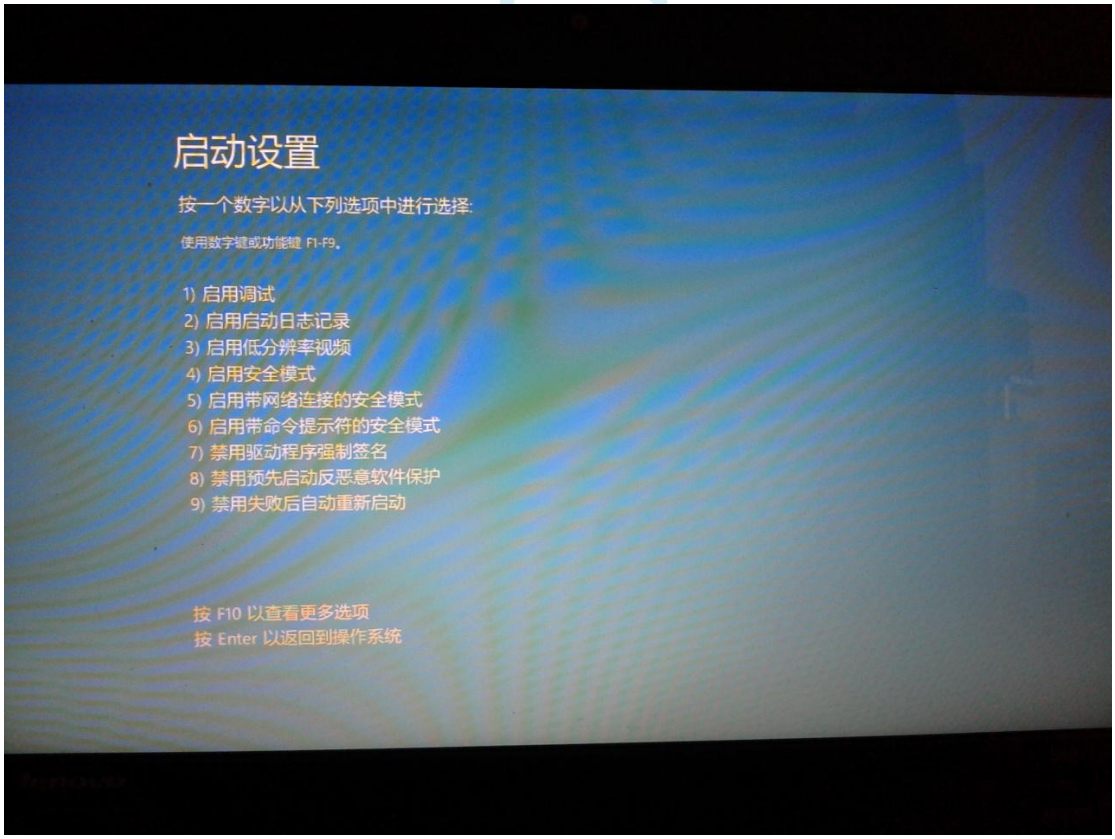
在下面的“高级选项”界面中，选择“启动设置”



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

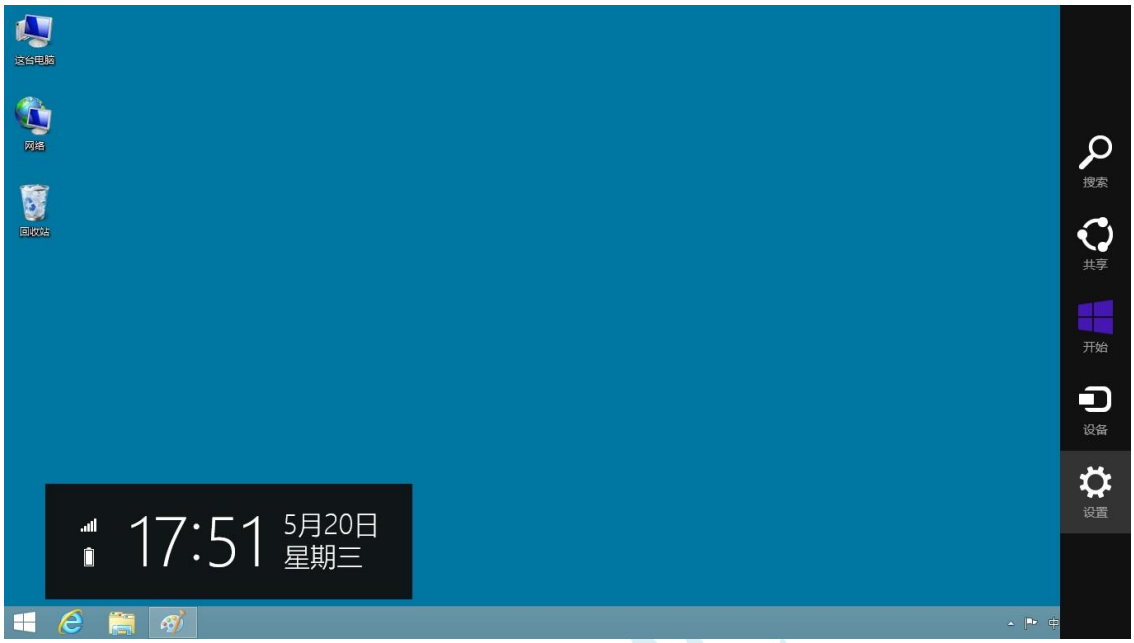


启动到 Windows 8 后，按照 **Windows 8（32 位）的安装方法**即可完成驱动的安装

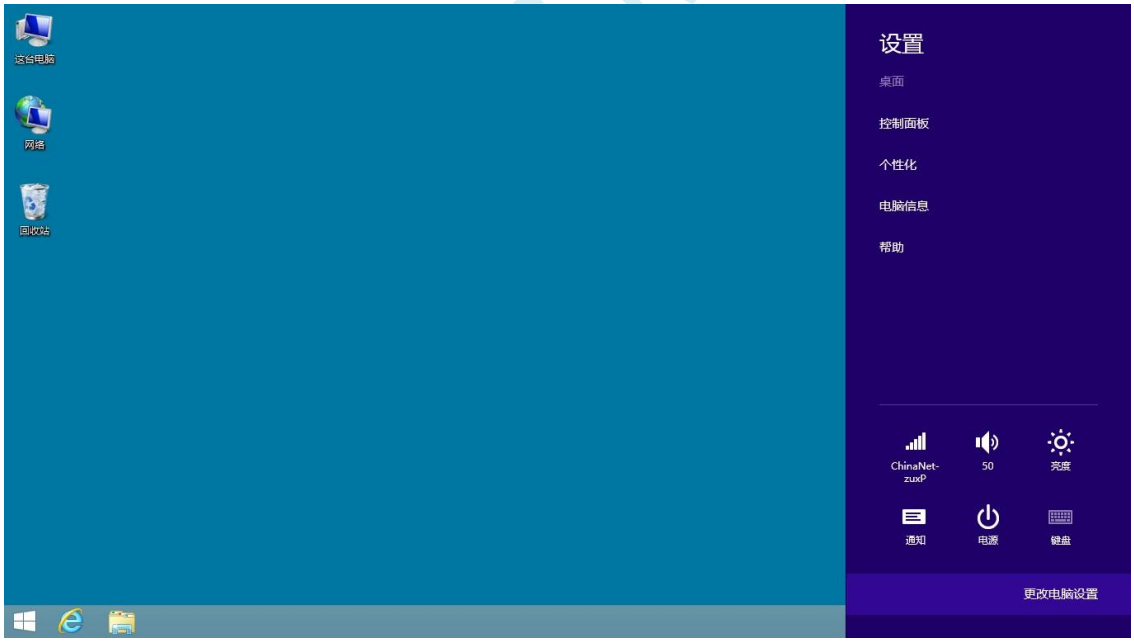
Windows 8.1（64 位）安装方法

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。

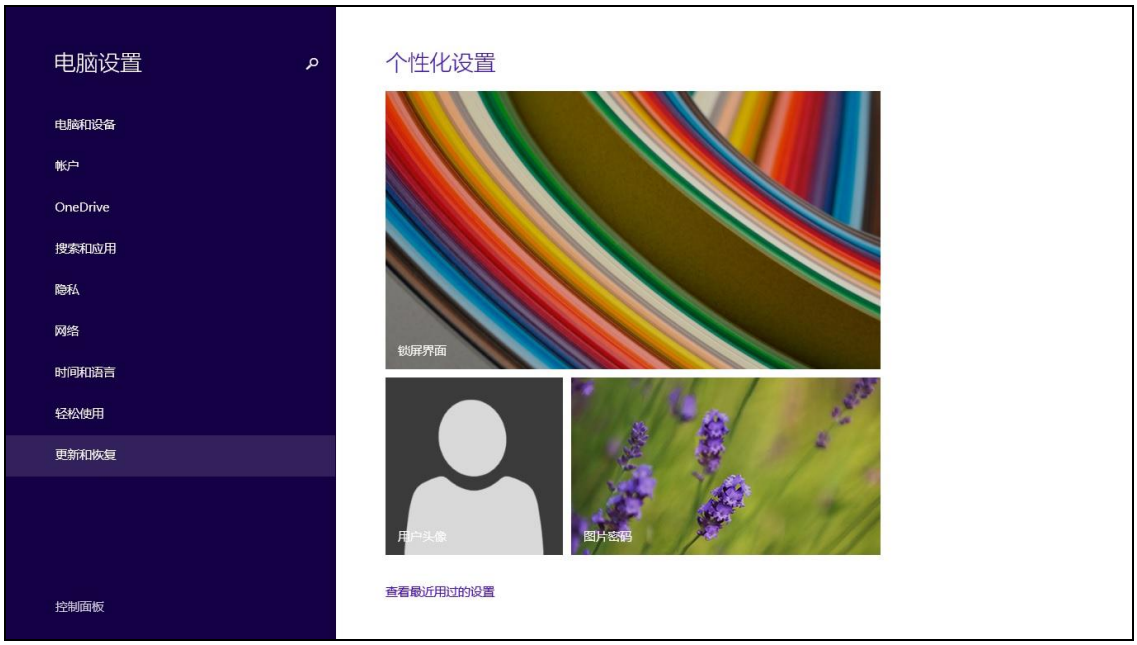
首先将鼠标移动到屏幕的右下角，选择其中的“设置”按钮



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”



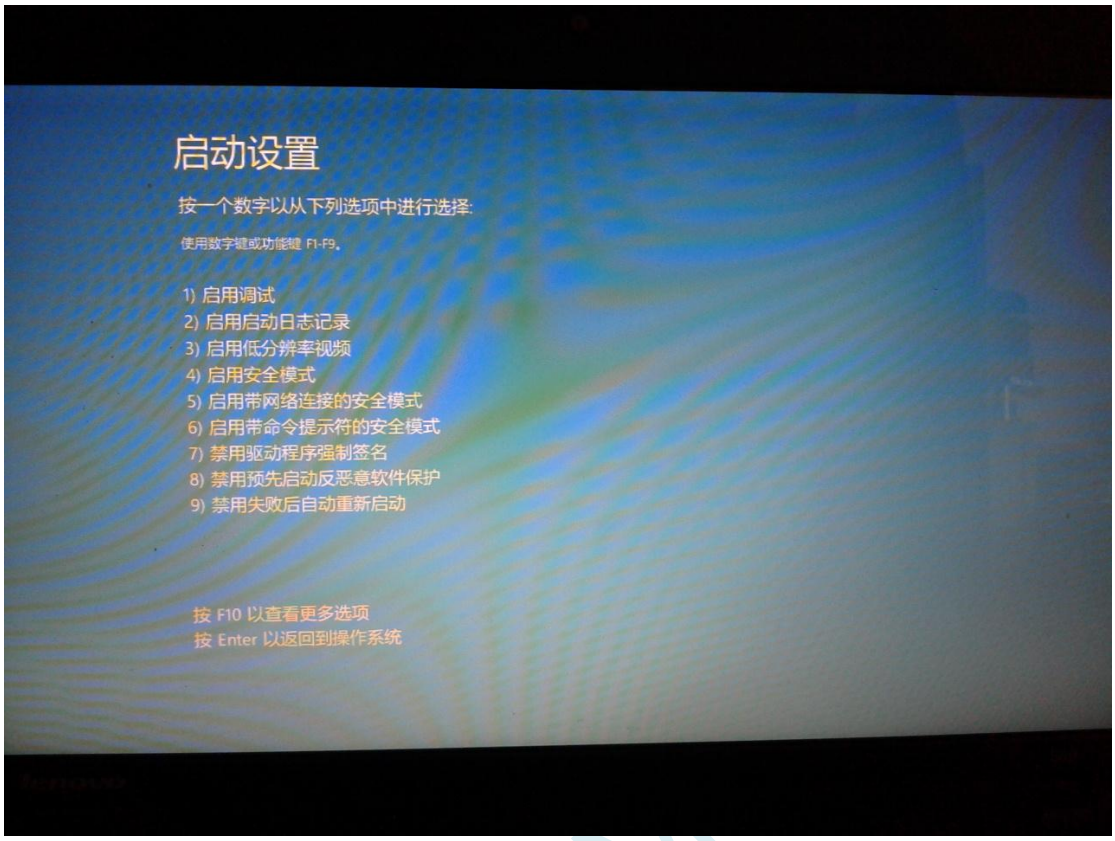
在下面的“高级选项”界面中，选择“启动设置”



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



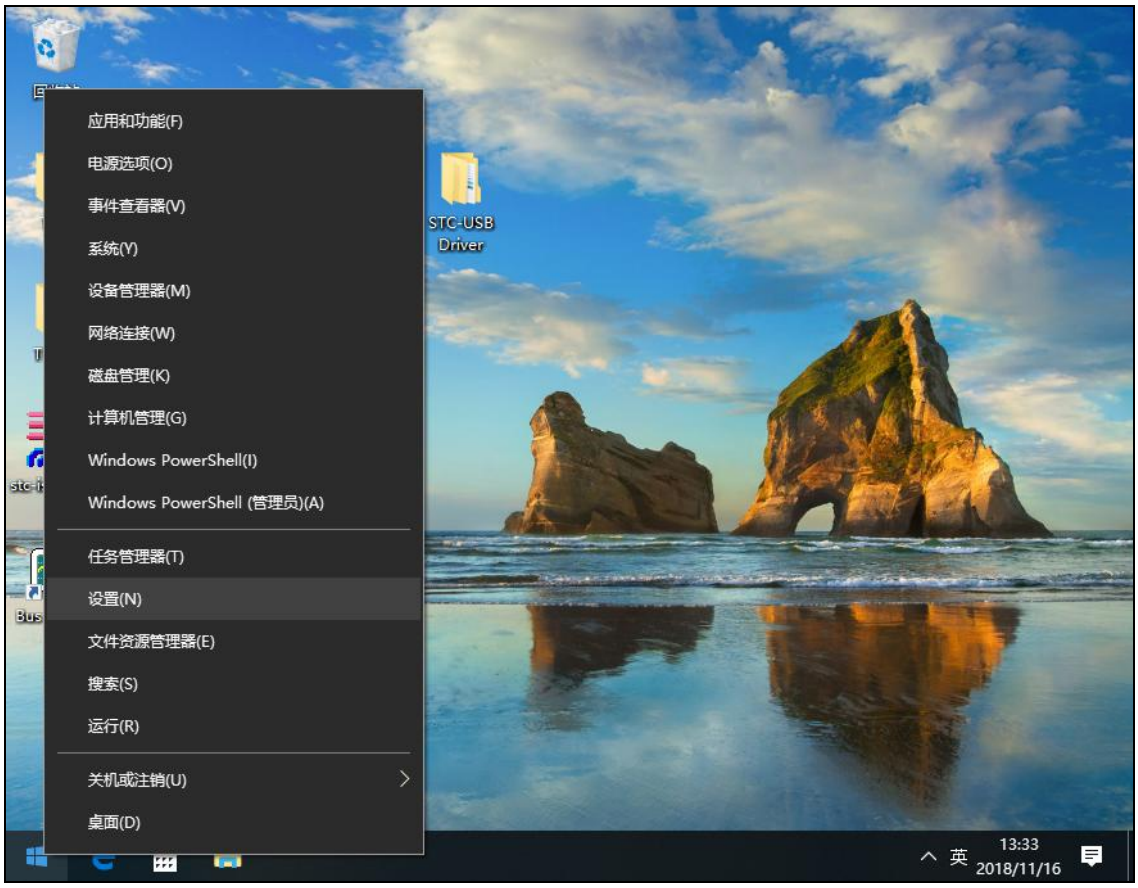
启动到 Windows 8 后，按照 **Windows 8（32 位）的安装方法**即可完成驱动的安装

Windows10（64 位）安装方法

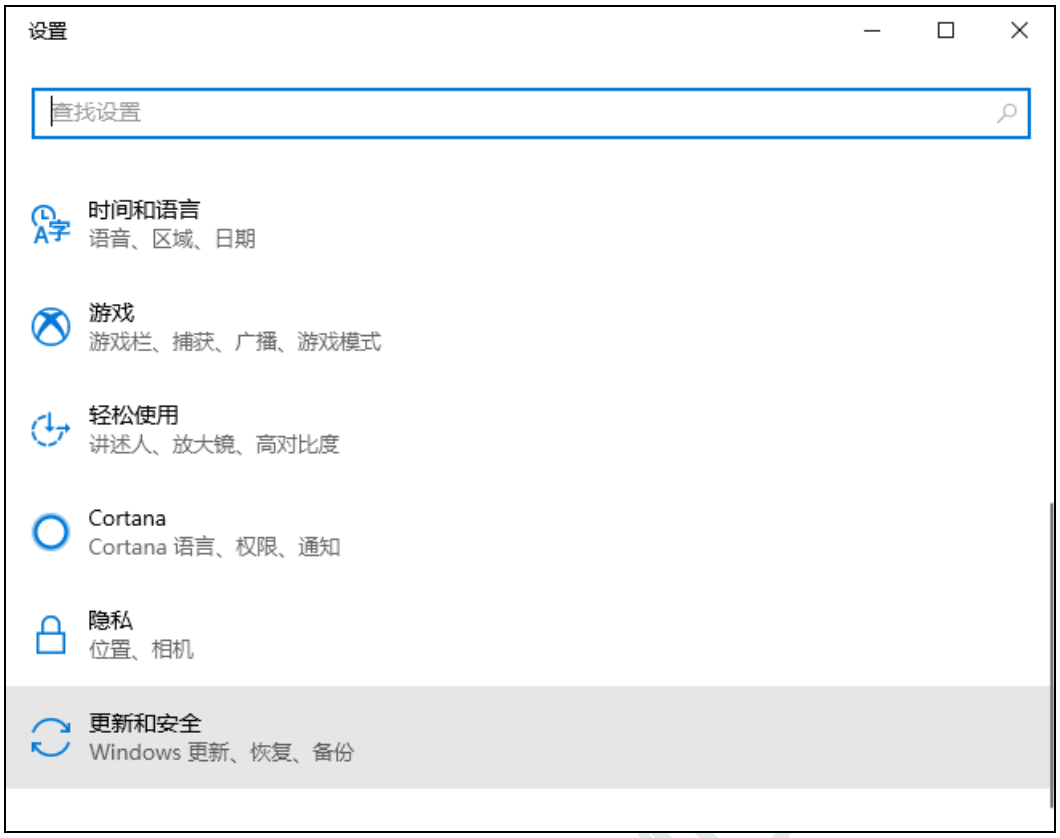
由于 Windows10 64 位操作系统在默认状态下，对于没有数字签名的驱动程序是不能安装成功的。所以在安装 STC-USB 驱动前，需要按照如下步骤，暂时跳过数字签名，即可顺利安装成功。

安装驱动前需要从 STC 官网下载的 AIapp-ISP 下载软件压缩包中将“STC-USB Driver”文件夹解压缩到硬盘中。将具有 USB 下载功能的芯片准备好，但先不要连接电脑

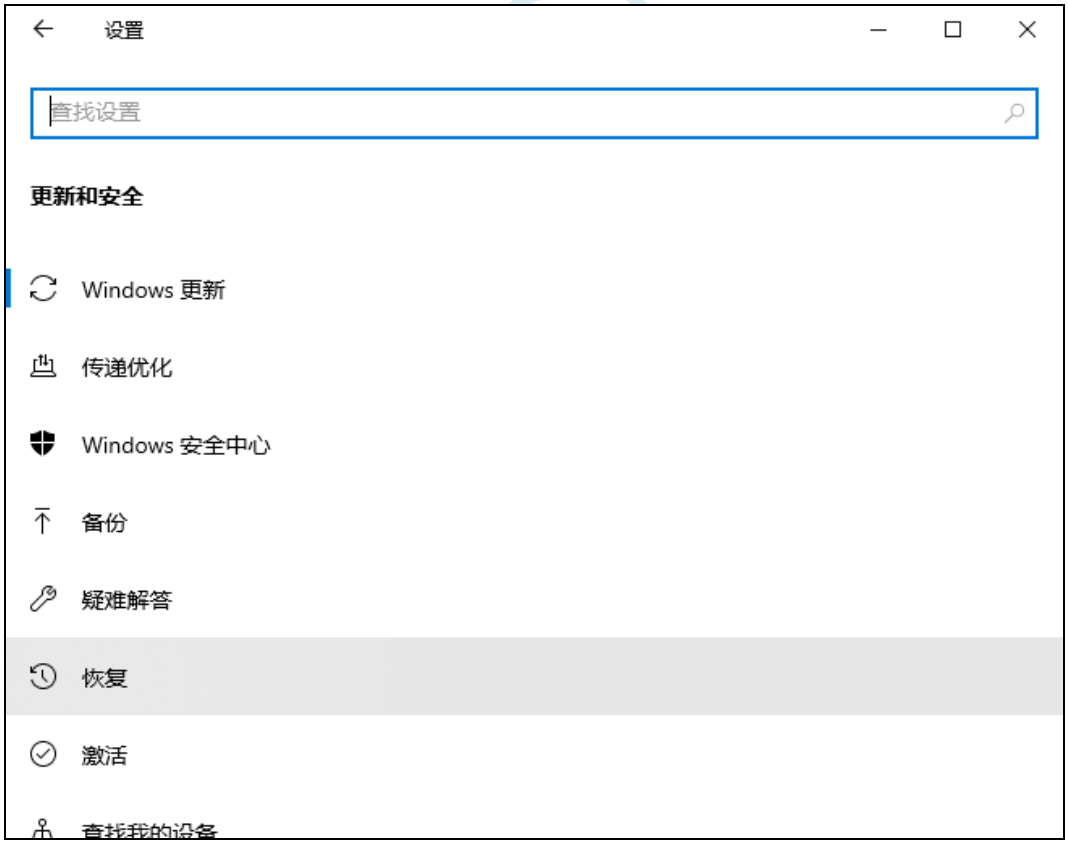
鼠标右键点击“开始”菜单，选择“设置”选项



然后在设置界面中选择“更新和安全”项



然后在设置界面中选择“恢复”项



在恢复界面中，点击“高级启动”项中的“立即重新启动”按钮



在电脑重启前，系统会先进入如下的启动菜单，选择“疑难解答”项



在疑难解答界面中选择“高级选项”



然后选择“查看更多恢复选项”



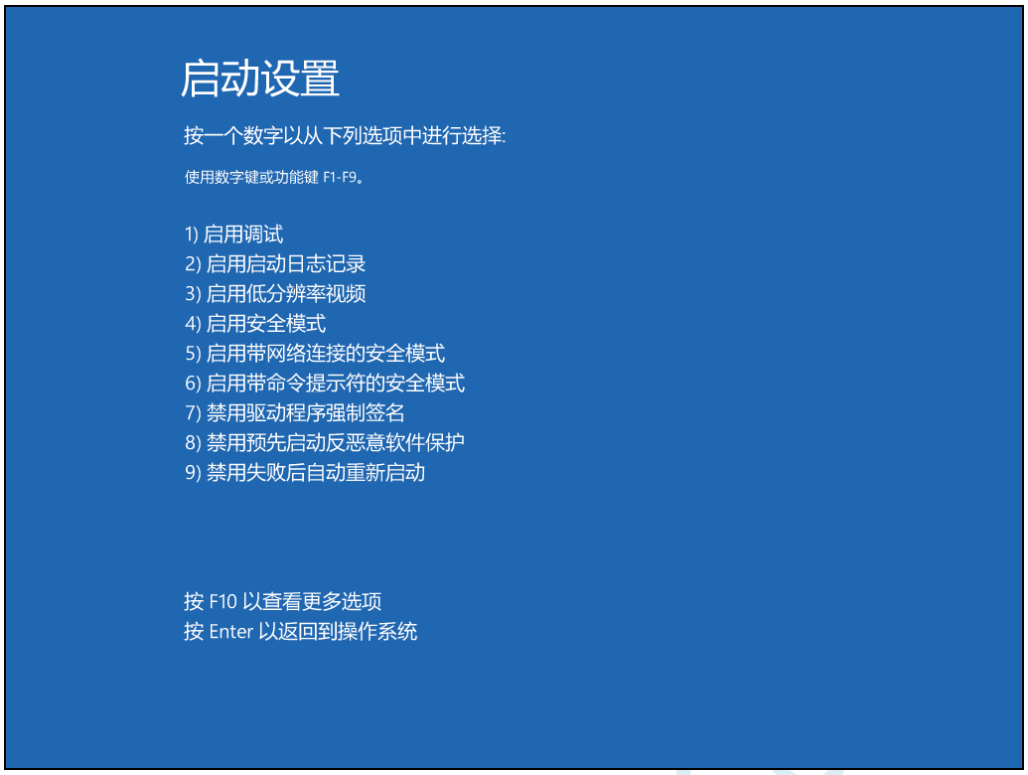
选择“启动设置”项



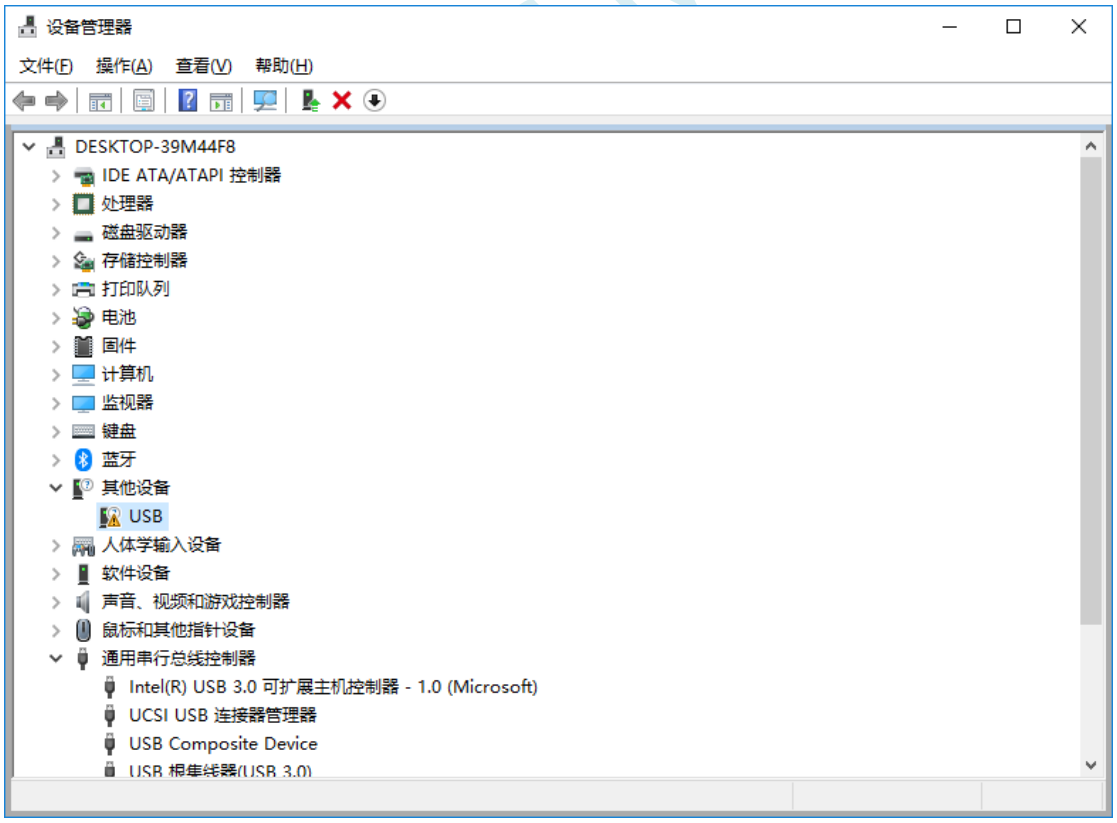
出现如下画面后，点击“重启”按钮重启电脑



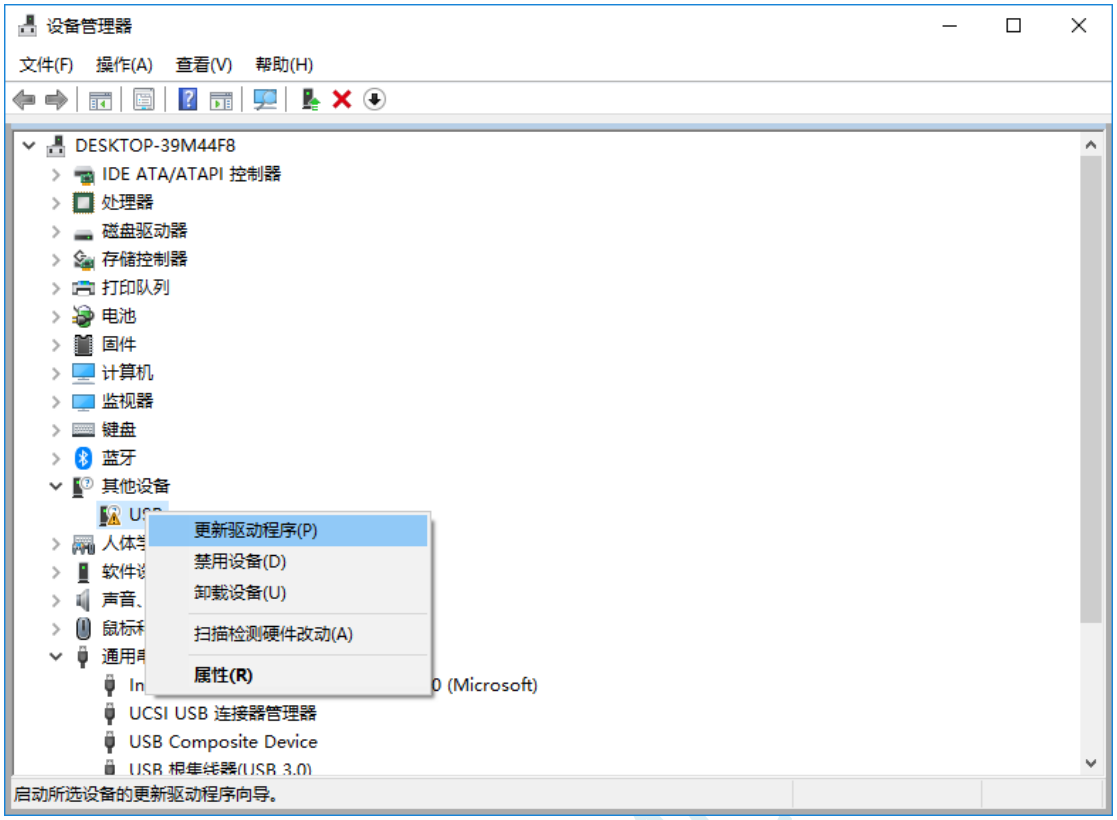
电脑重启后，会弹出“启动设置”界面，按“F7”按钮来选择“禁止驱动程序强制签名”项



电脑启动完成后，将准备好的芯片用 USB 线与电脑相连，并打开“设备管理器”，此时由于驱动还没有开始安装，所以在设备管理器中会显示为一个带感叹号的未知设备



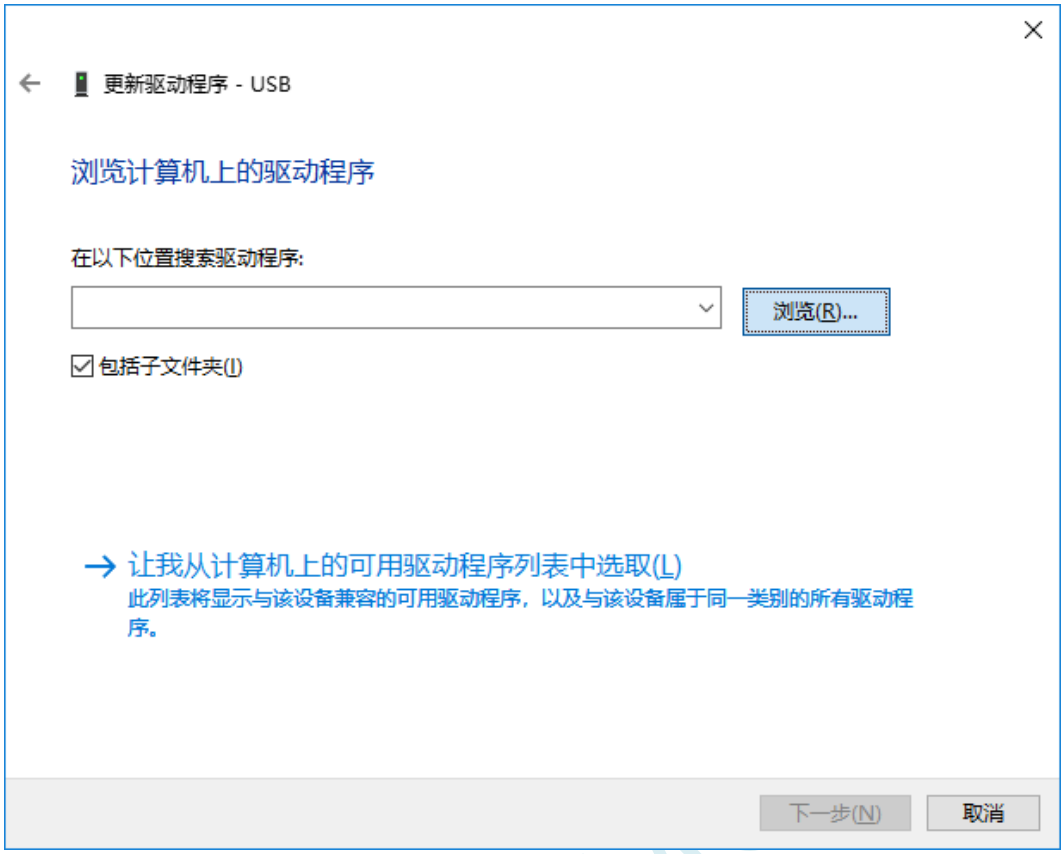
鼠标右键单击未知设备，选择右键菜单中的“更新驱动程序”



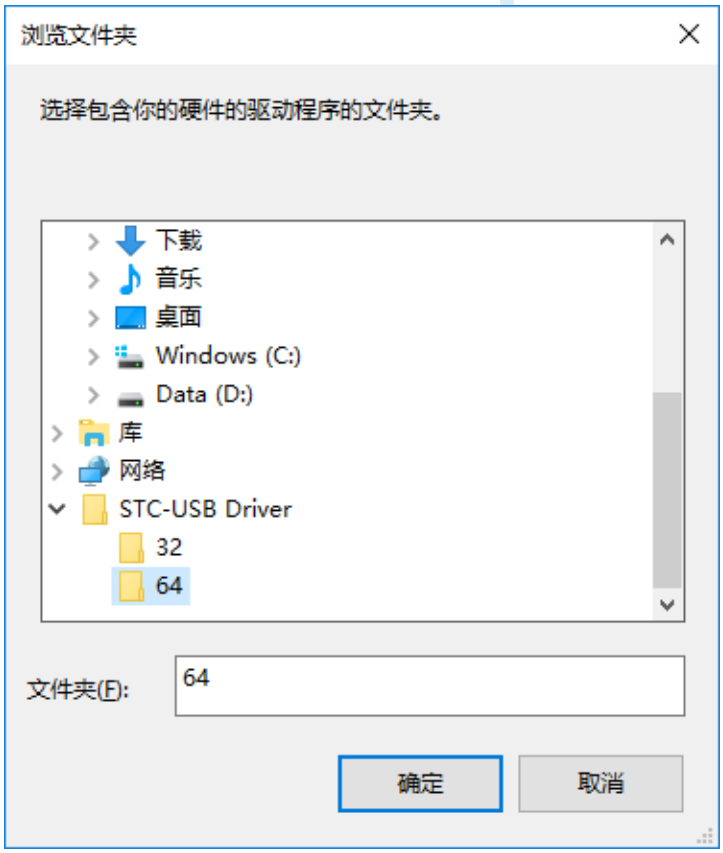
在弹出的驱动安装程序选择画面中，选择“浏览我的计算机以查找驱动程序软件”项



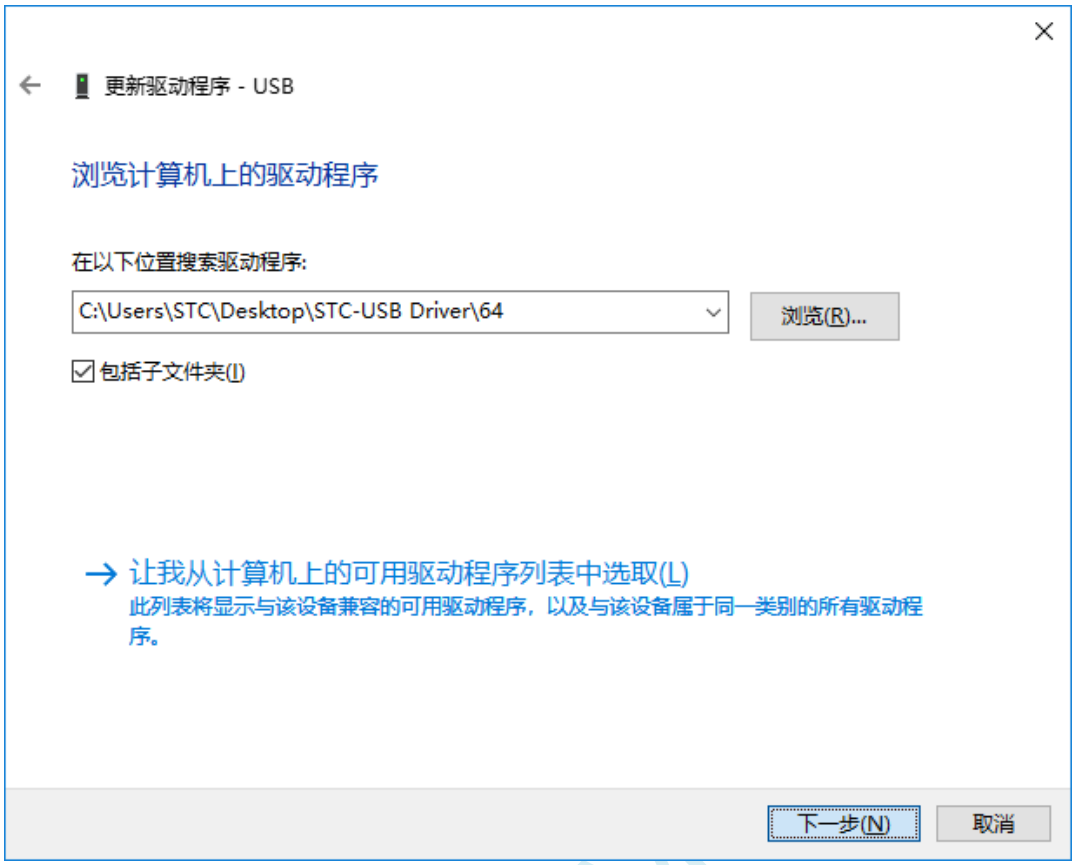
在如下界面中，点击“浏览”按钮



找到之前解压缩到硬盘中的“STC-USB Driver”目录，选择目录中的“64”目录，并确定



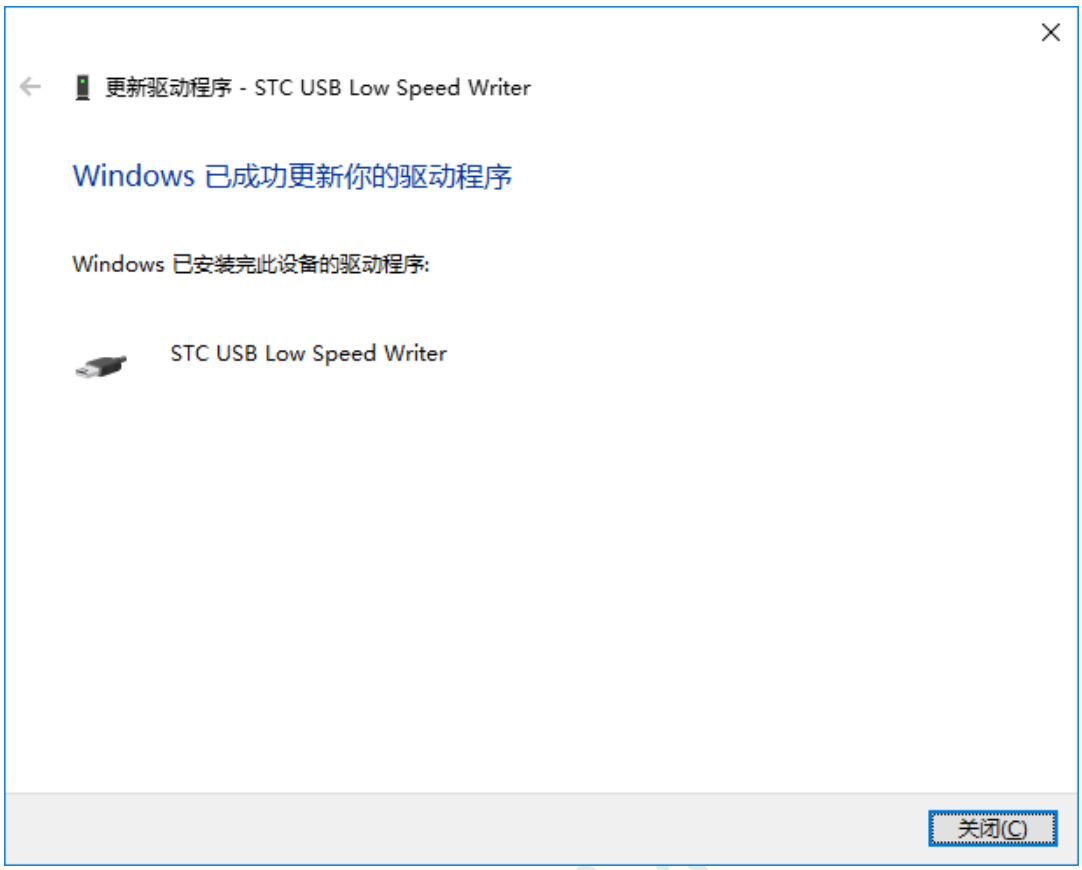
点击“下一步”开始安装驱动



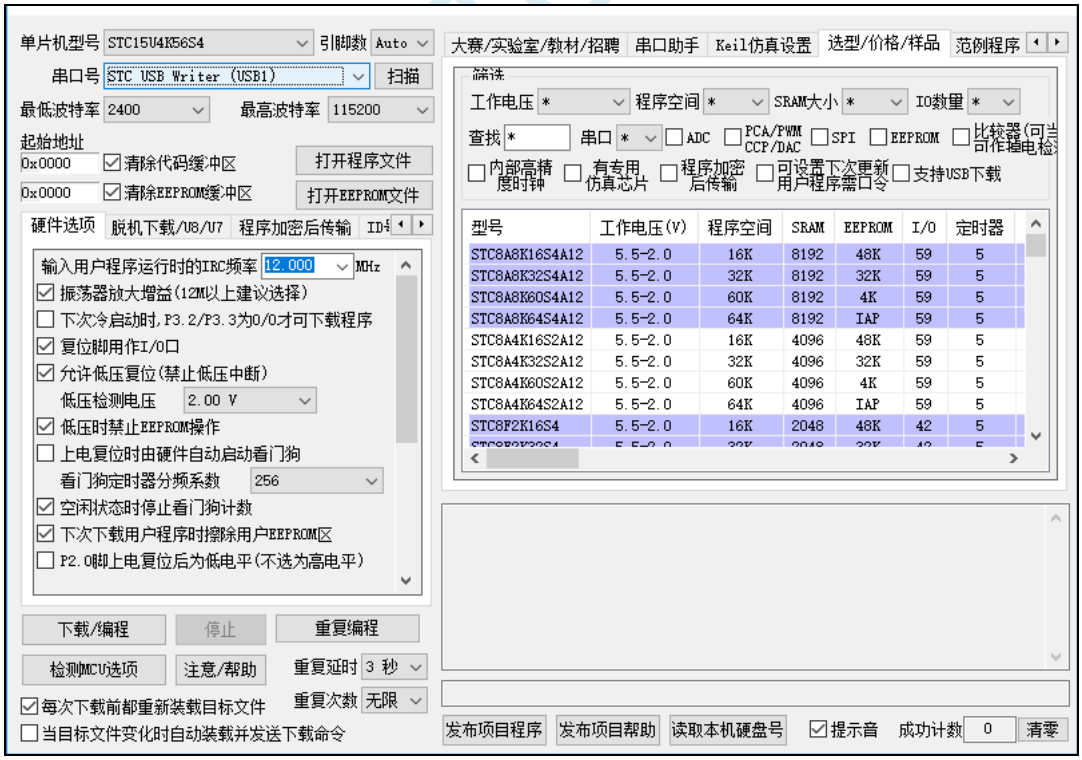
驱动安装的过程中，会弹出如下的警告画面，选择“始终安装此驱动程序软件”



出现下面的画面时，驱动程序就安装成功了

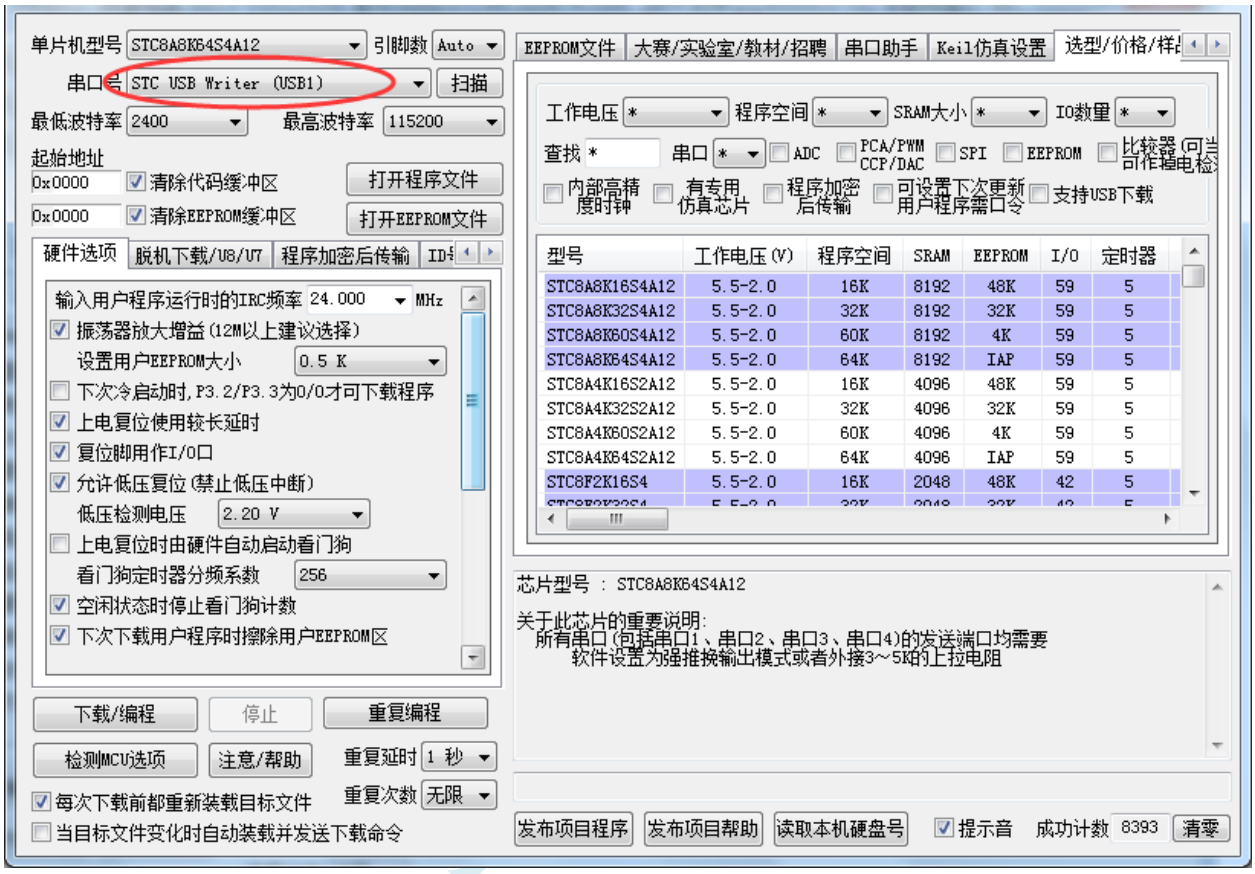


在回到 AIapp-ISP 的下载软件，此时“串口号”的下拉列表中已自动选择了“STC USB Writer (USB1)”，即可使用 USB 进行 ISP 下载了

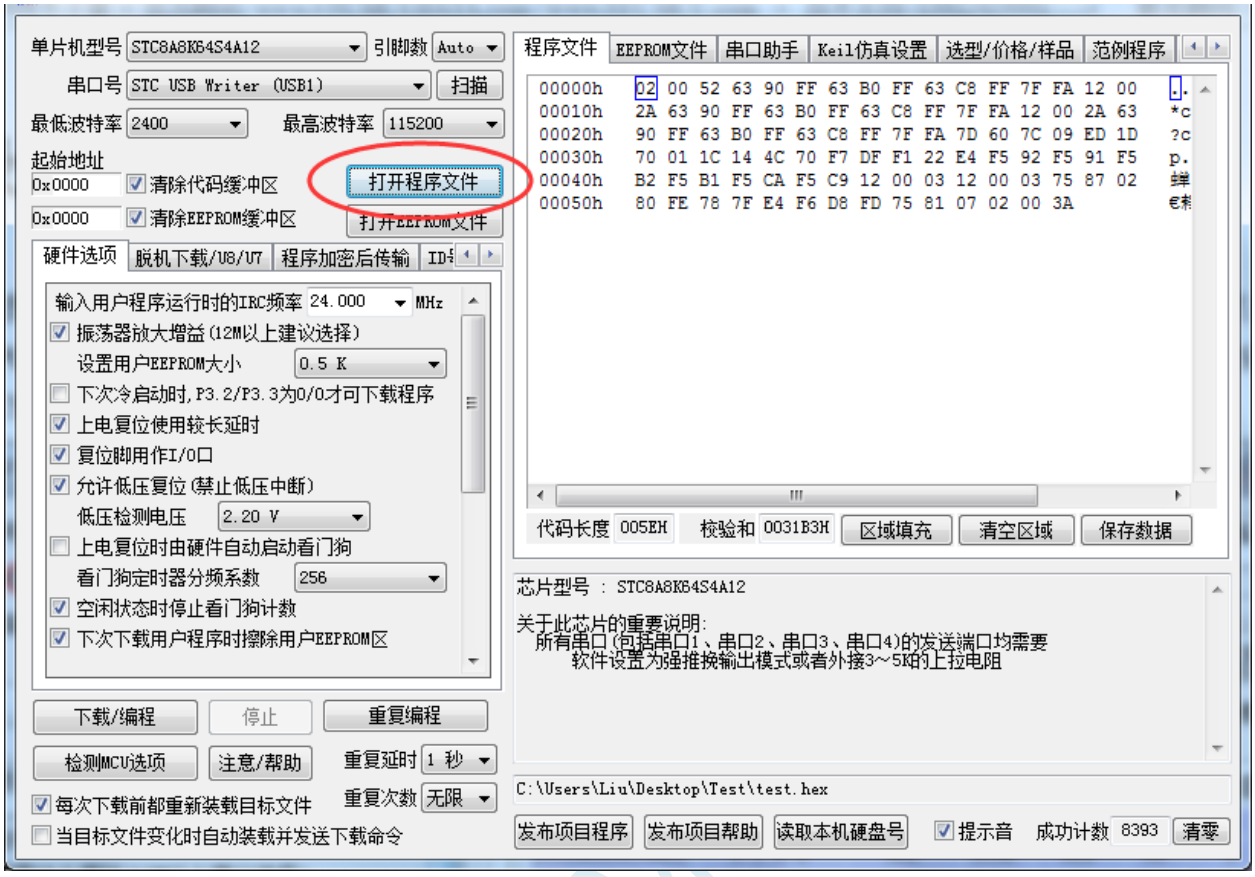


附录K　USB 下载步骤演示

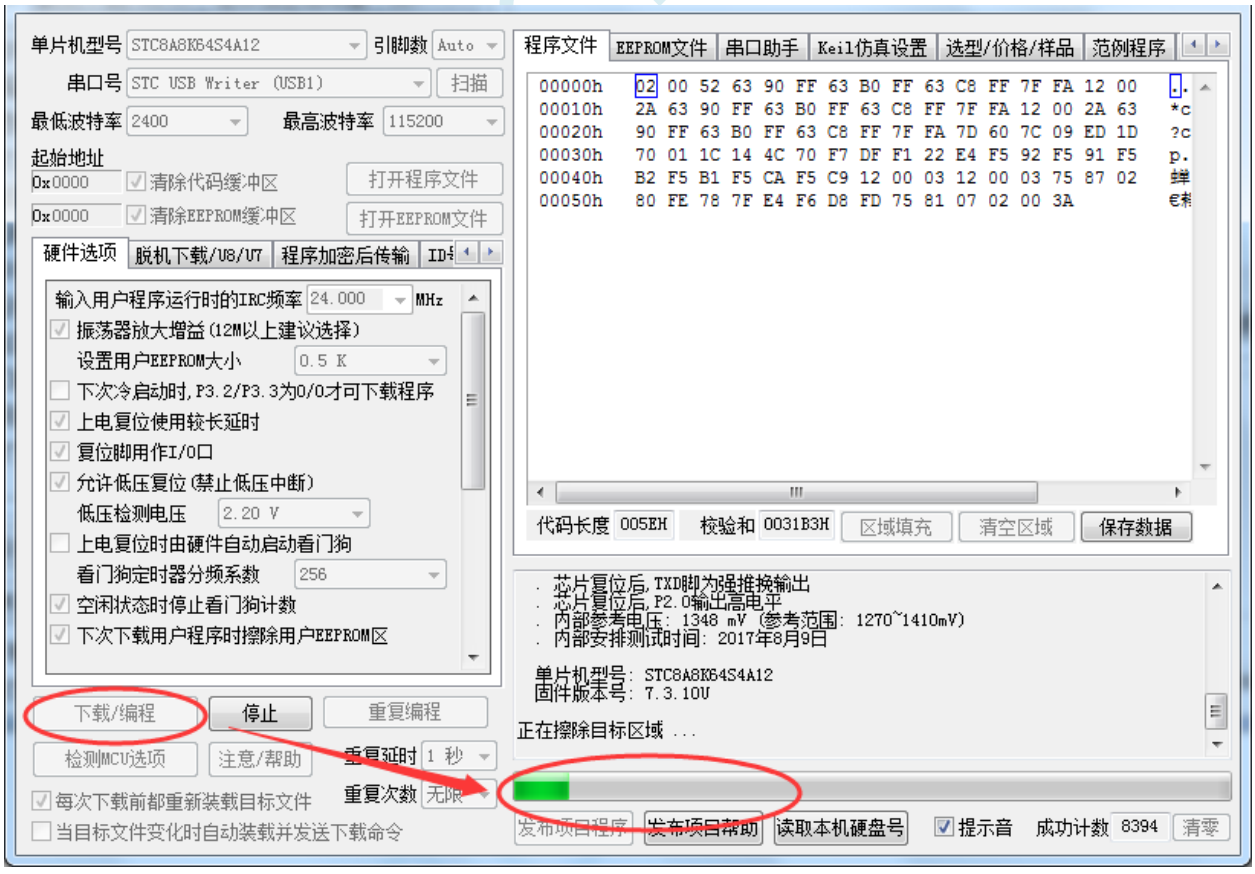
1、首先参考 P5.1.5 章的应用线路图连接好单片机，并将目标芯片的 P3.2 口连接到 Gnd，然后将系统连接到 PC 端的 USB 端口上。打开 ISP 下载软件，即可在下载软件的串口号中自动搜索到“STC USB Writer (USB1)”的 USB 设备



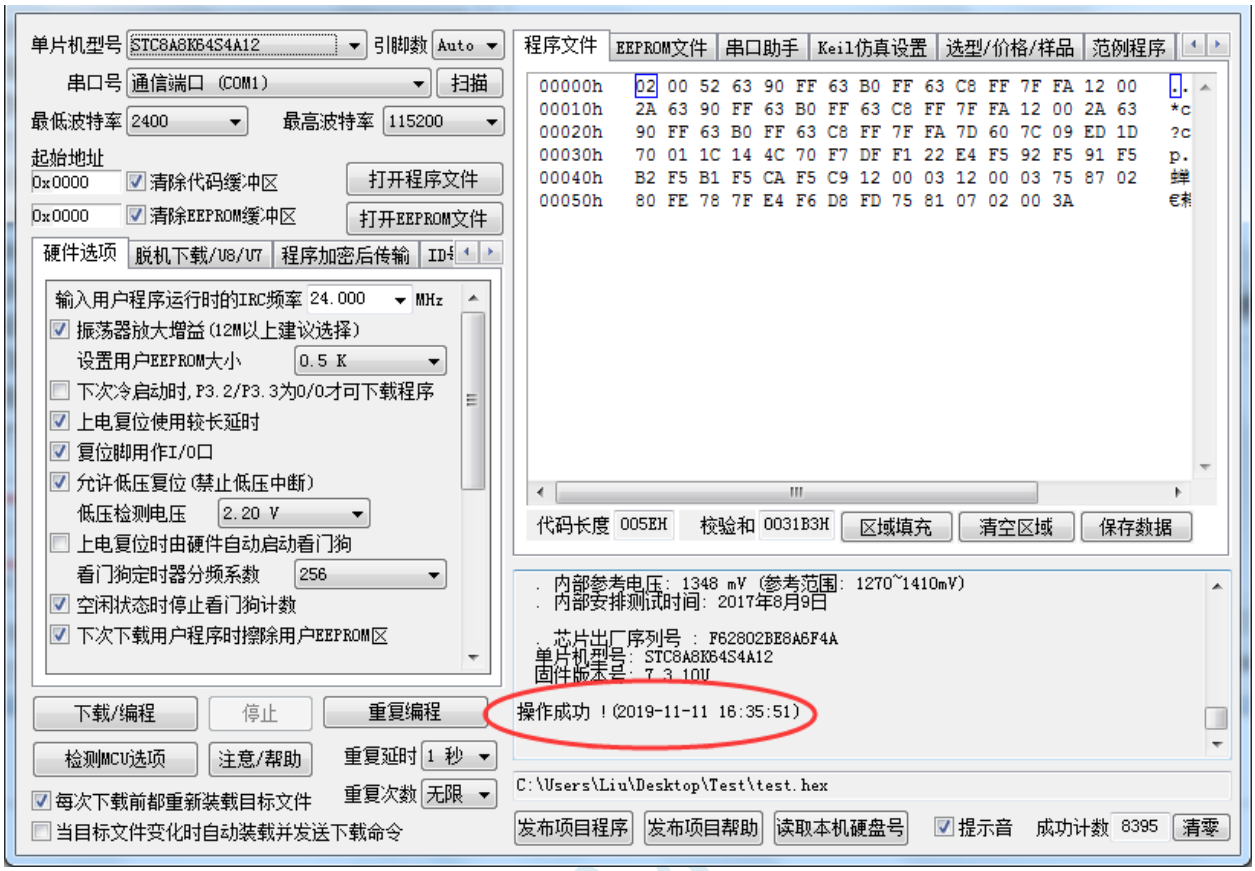
2、打开用户代码程序



3、点击“下载/编程”按钮开始下载用户代码

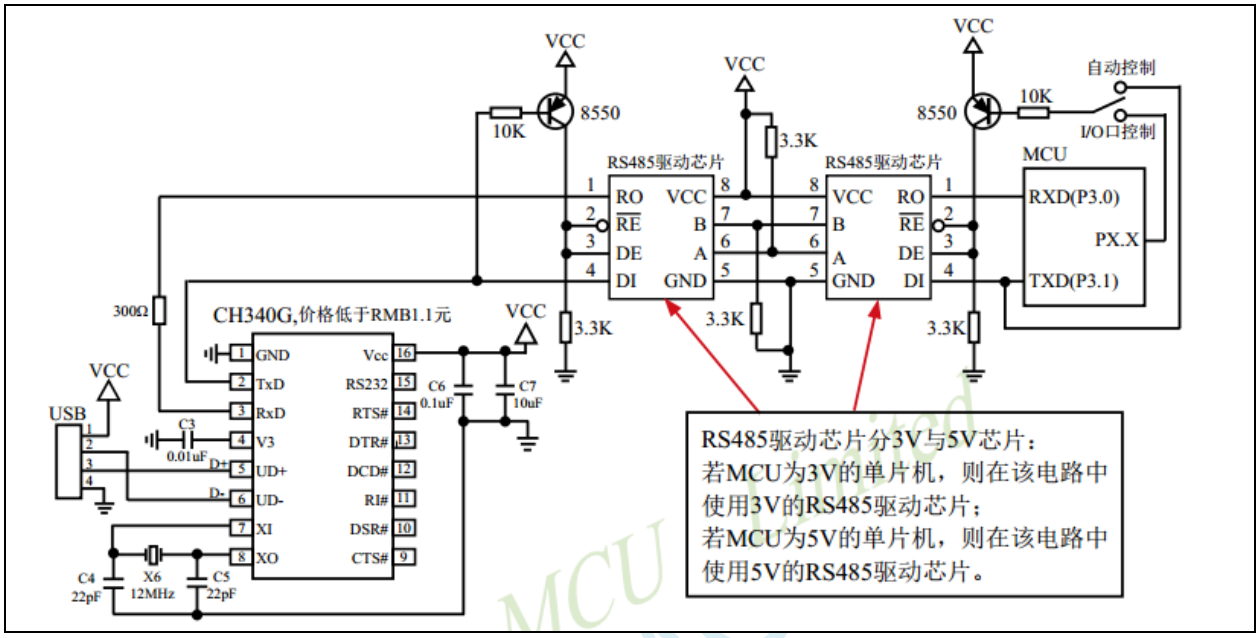


4、直到提示“操作成功”，表示程序代码下载完成。

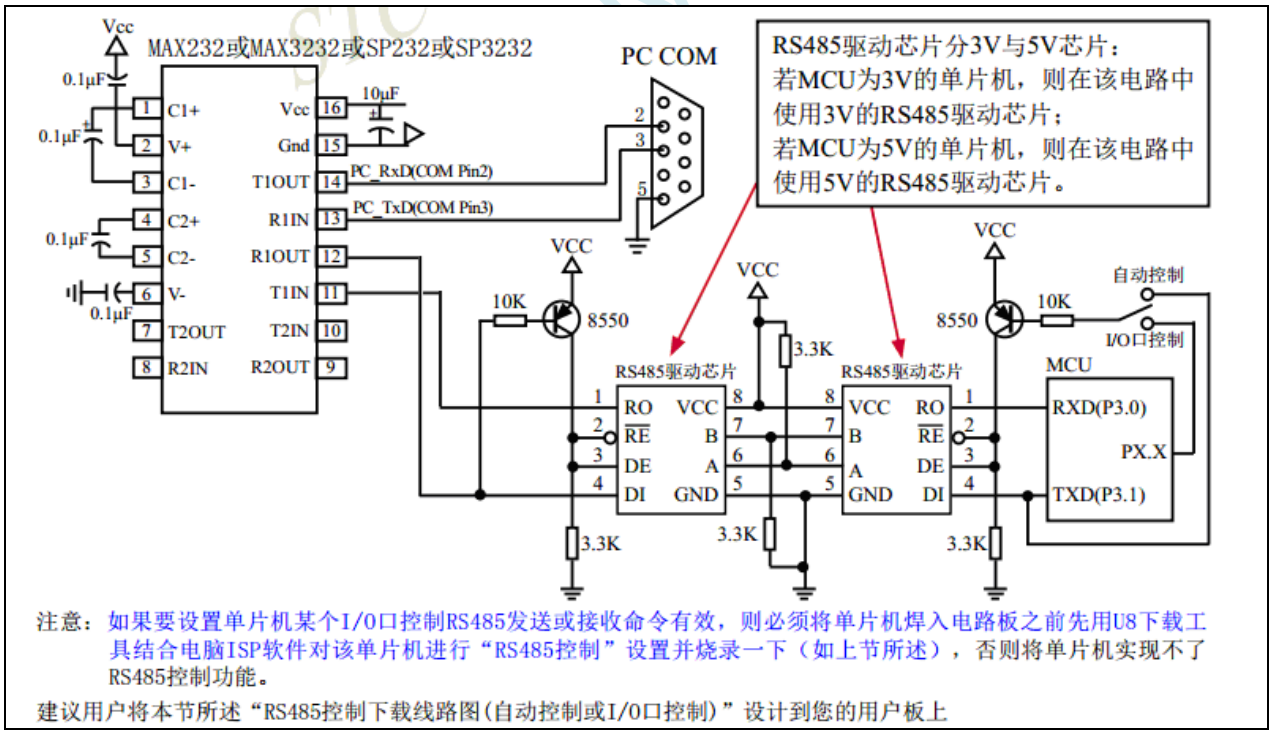


附录L　RS485 自动控制或 I/O 口控制线路图

1、利用 USB 转串口连接电脑的 RS485 控制下载线路图(自动控制或 I/O 口控制)



2、利用 RS232 转串口连接电脑的 RS485 控制下载线路图(自动控制或 I/O 口控制)



注意：如果要设置单片机某个I/O口控制RS485发送或接收命令有效，则必须将单片机焊入电路板之前先用U8下载工具结合电脑ISP软件对该单片机进行“RS485控制”设置并烧录一下（如上节所述），否则将单片机实现不了RS485控制功能。

建议用户将本节所述“RS485控制下载线路图(自动控制或I/O口控制)”设计到您的用户板上

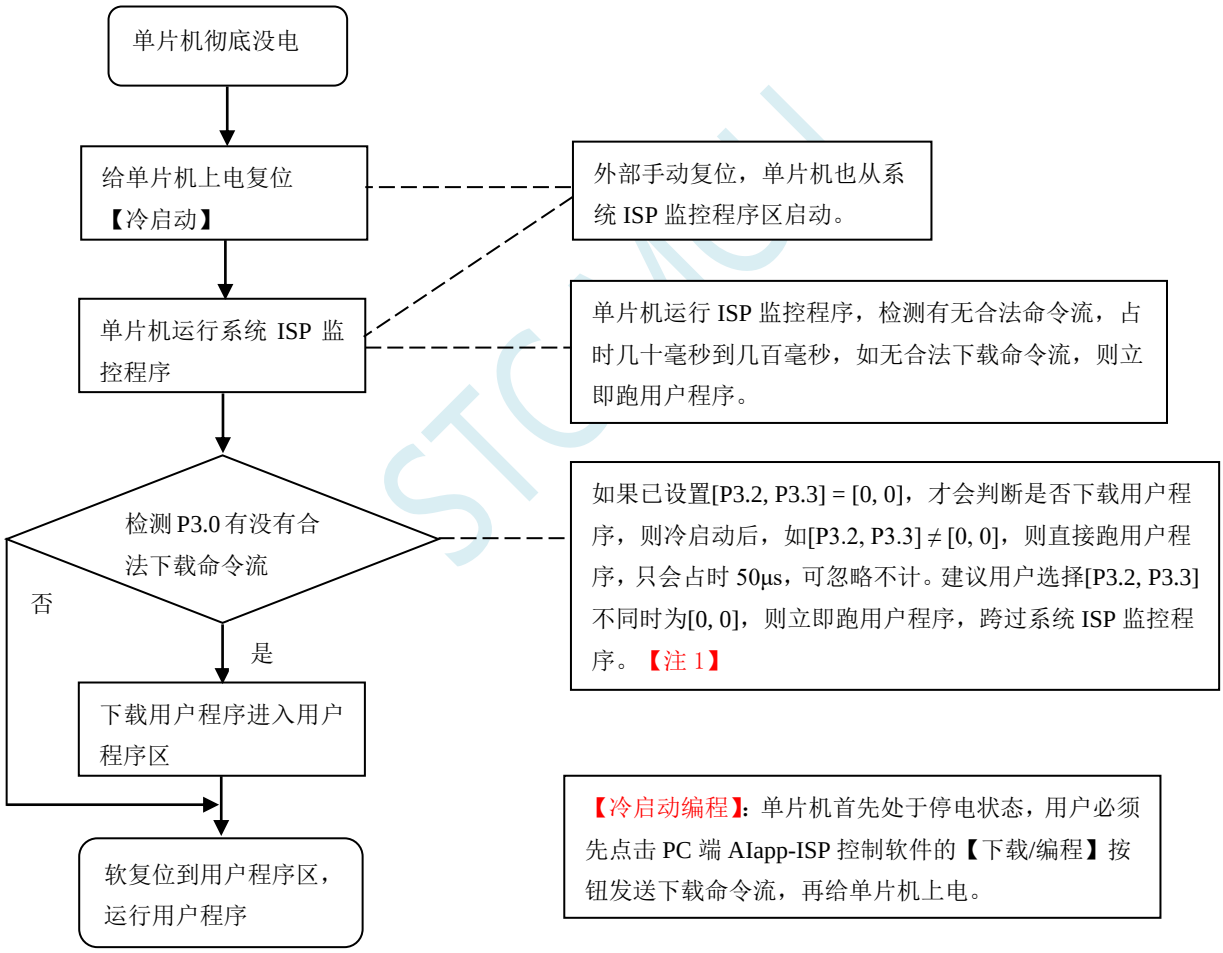
附录M　STC 工具使用说明书

M.1　概述

U8W/U8W-Mini 是一款集在线联机下载和脱机下载于一体的编程工具系列。STC 通用 USB 转串口工具则是支持在线下载与在线仿真的编程工具。

工具类型	在线下载	脱机下载	烧录座下载	在线仿真	价格(人民币)
U8W	支持	支持	支持	需设置直通模式	100 元
U8W-Mini	支持	支持	不支持	需设置直通模式	50 元
通用 USB 转串口	支持	不支持	不支持	支持	30 元

M.2　系统可编程（ISP）流程说明

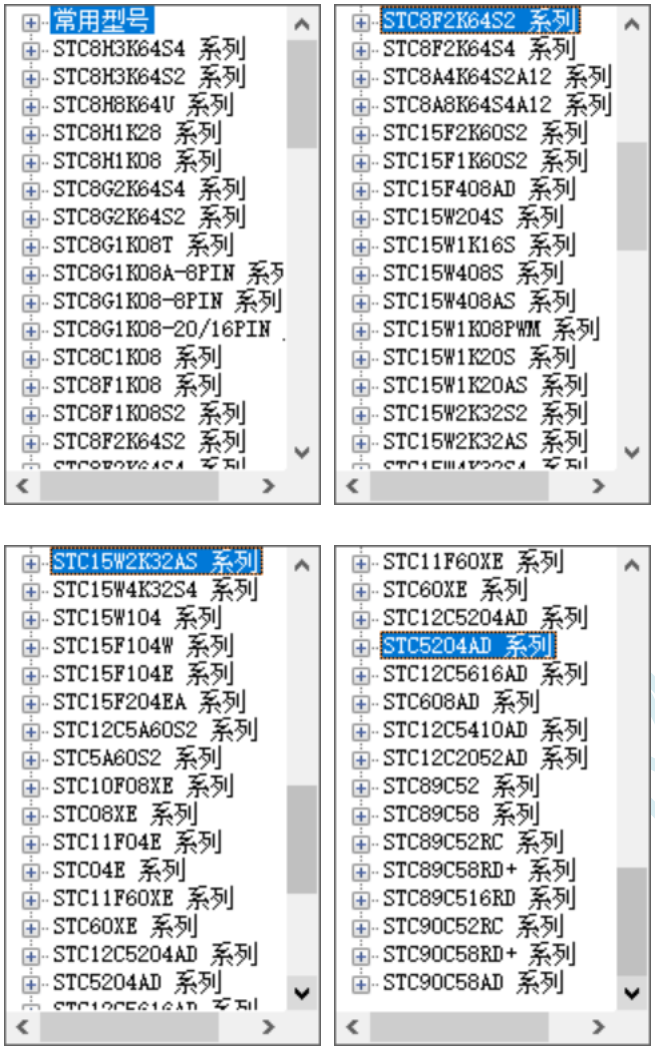


注意：因 [P3.0, P3.1] 作下载/仿真用(下载/仿真接口仅可用 [P3.0, P3.1])，故建议用户将串口 1 放在 P3.6/P3.7 或 P1.6/P1.7 ，若用户不想切换，坚持使用 P3.0/P3.1 工作或作为串口 1 进行通信，则务必在下载程序时，在软件上勾选“下次冷启动时，P3.2/P3.3 为 0/0 时才可以下载程序”。【注 1】

【注 1】：STC15，STC8 系列及以后新出的芯片的烧录保护引脚为 P3.2/P3.3，之前早期芯片的烧录保护引脚为 P1.0/P1.1。

M.3 USB 型联机/脱机下载工具 U8W/U8W-Mini

U8W/U8W-Min 的应用范围可支持 STC 目前的全部系列的 MCU, Flash 程序空间和 EEPROM 数据空间不受限制。支持包括如下和即将推出的 STC 全系列芯片：



脱机下载工具可以在脱离电脑的情况下进行下载工作，可用于批量生产和远程升级。脱机下载板可支持自动增量、下载次数限制以及用户程序加密后传输等多种功能。

下图为 U8W 工具的正反面图以及 U8W-Mini 的正反面图：



U8W正面图



U8W反面图


U8W-Mini正反面图

U8W-Mini工具的体积仅有一个U盘大小，其功能与U8W相同，但无锁紧座，价格仅为RMB 50元，欢迎来电订购！

另外还有如下的一些线材与工具相搭配使用，如：

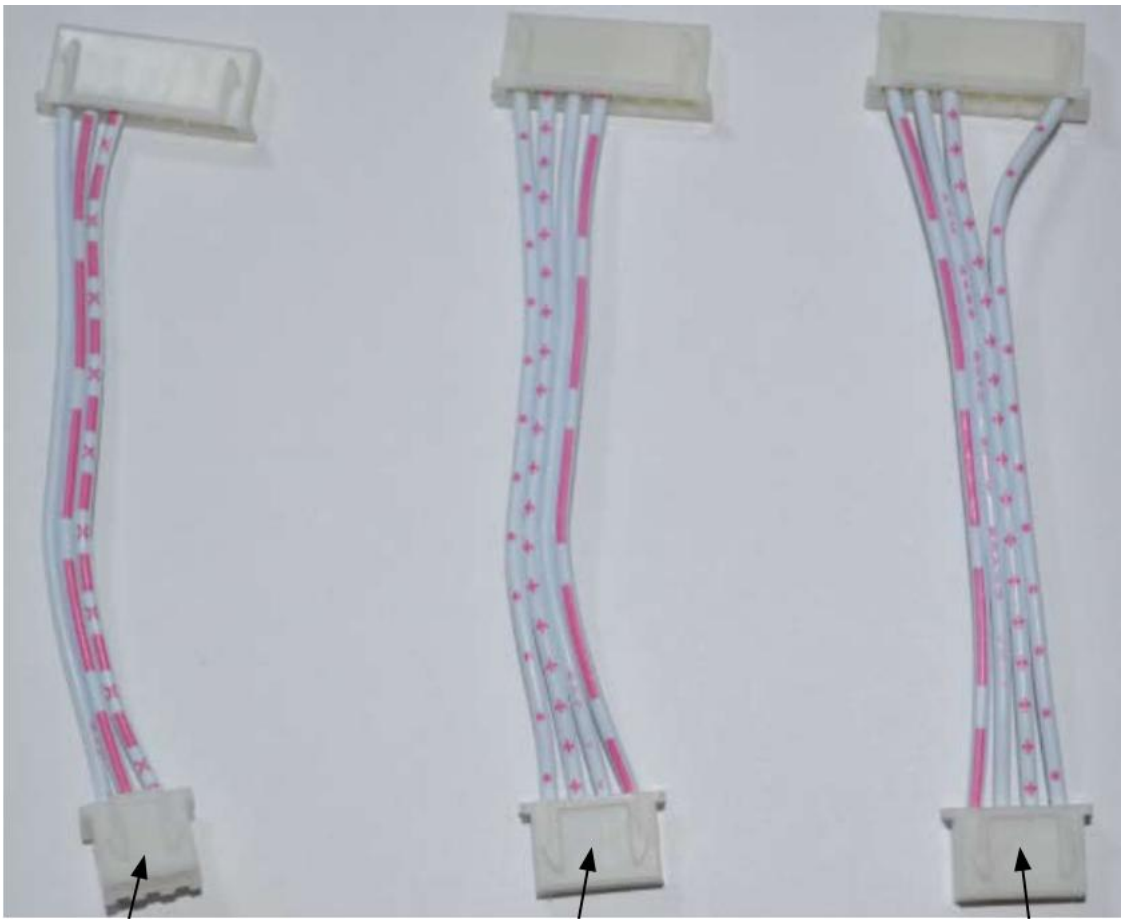
（1）两头公的 USB 连接线(如下图左所示）及 USB-Micro 连接线(如下图右所示)：





注意：此 USB 线为我公司特别定制的 USB 加强线，可确保直接用 USB 供电时能够下载成功。而市面上一些比较劣质的两头公的 USB 线，内阻太大而导致压降很大（如 USB 空载时的电压为 5.0V 左右，当使用劣质的 USB 线连接 U8W/U8W-Mini/U8/U8-Mini，到我们的下载板上的电压可能降到 4.2V 或者更低，从而导致芯片处于复位状态而无法成功下载）。

（2）U8W/U8W-Mini 与用户系统连接的下载连接线(即 U8W/U8W-Mini 与用户板上的目标单片机的连接线)，如下图所示：



U8W/U8W-Mini 与用户系统各自独立供电的连接线

U8W/U8W-Mini 给用户系统供电的连接线

用户系统给 U8W/U8W-Mini 供电的连接线

M.3.1 安装 U8W/U8W-Mini 驱动程序

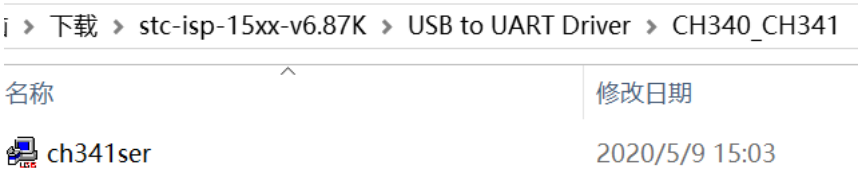
U8W/U8W-Mini 下载板上使用了一颗 CH340 的 USB 转串口通用芯片。这样可以省去部分没有串口的电脑必须额外买一个 USB 转串口工具才可下载的麻烦。但 CH340 和其它 USB 转串口工具一样，在使用之前必须先安装驱动程序。

通过下载 AIapp-ISP 软件包获取驱动程序

以下是 STC 官网（www.STCMCUDATA.com）提供的 AIapp-ISP 软件包下载位置：

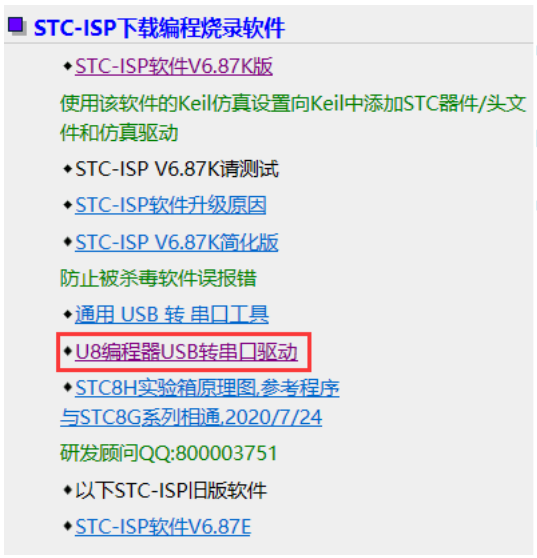


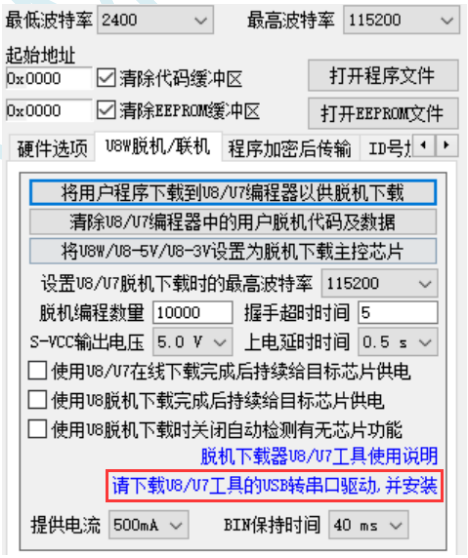
下载后进行解压，CH340 的驱动安装包路径 stc-isp-15xx-v6.87K\USB to UART Driver\CH340_CH341：



通过 STC 的官方网站或在最新的 AIapp-ISP 下载软件中手动下载驱动程序

在 STC 的官方网站上或在最新的 AIapp-ISP 下载软件中手动下载驱动程序，驱动的下载链接为： U8 编程器 USB 转串口驱动 （ http://www.stcmcu.com/STCISP/CH341SER.exe）。网站上及 AIapp-ISP 下载软件上的驱动地址如下图所示：



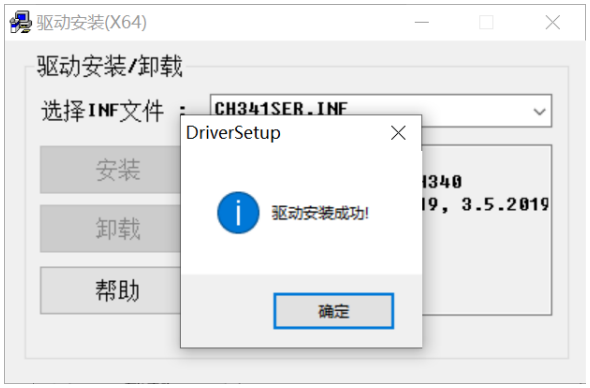


安装 U8W/U8W-Mini 的驱动程序

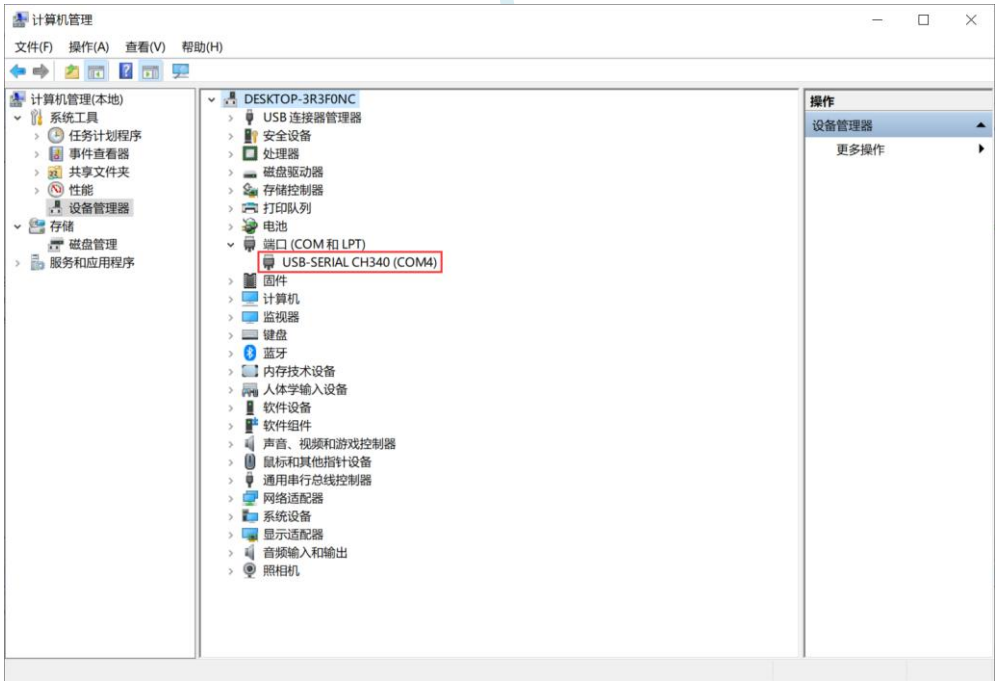
驱动程序下载到本机后，直接双击可执行程序并运行，出现下图所示的界面，点击“安装”按钮开始自动安装驱动：



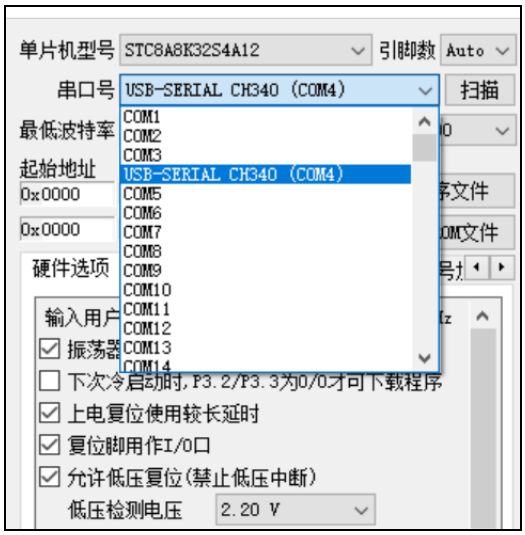
然后弹出驱动安装成功对话框，点击“确定”按钮完成安装：



然后使用 STC 提供的 USB 连接线将 U8W/U8W-Mini 下载板连接到电脑，打开电脑的设备管理器，在端口设备类下面，如果有类似“USB-SERIAL CH340 (COMx)”的设备，就表示 U8W/U8W-Mini 可以正常使用了。如下图所示（不同的电脑，串口号可能会不同）：

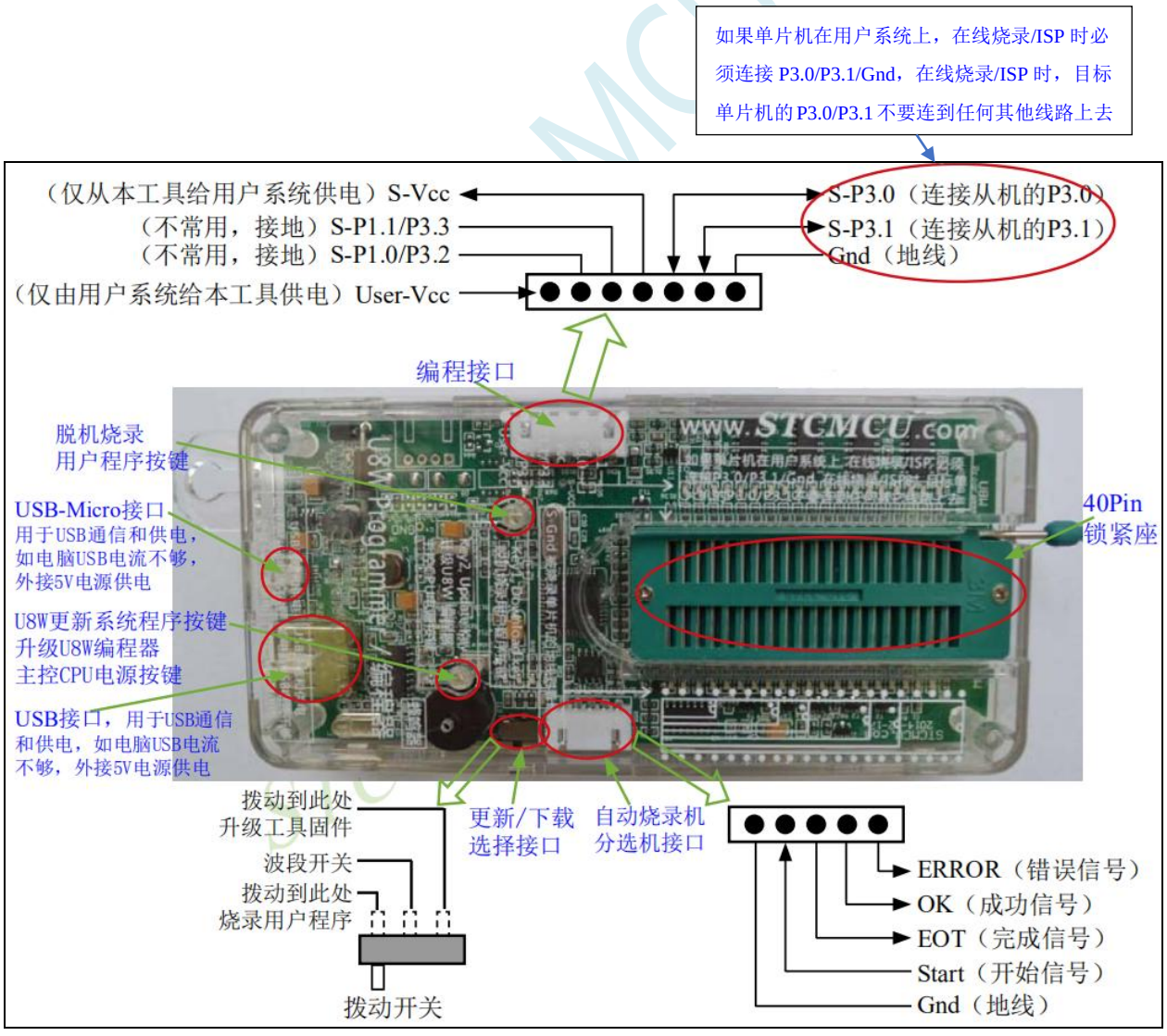


注意：在后面使用 AIapp-ISP 下载软件时，选择的串口号必须选择与此相对应的串口号，如下图所示：



M.3.2 U8W 的功能介绍

下面详细介绍 U8W 工具的各主要接口及功能：

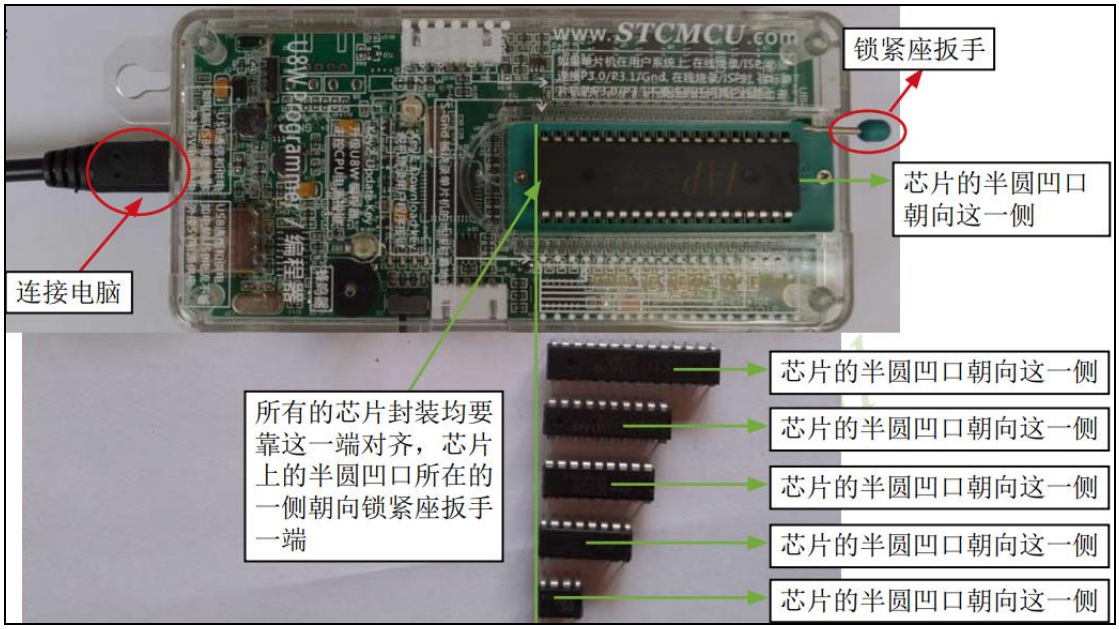


编程接口：根据不同的供电方式，使用不同的下载连接线连接 U8W 下载板和用户系统。
U8W 更新系统程序按键：用于更新 U8W 工具，当有新版本的 U8W 固件时，需要按下此按键对 U8W 的主控芯片进行更新（注意：必须先将更新/下载选择接口上的拨动开关拨动到升级工具固件）。
脱机下载用户程序按钮：开始脱机下载按钮。首先 PC 将脱机代码下载到 U8W 板上，然后使用下载连接线将用户系统连接到 U8W，再按下此按钮即可开始脱机下载（每次上电时也会立即开始下载用户代码）。
更新/下载选择接口：当需要对 U8W 的底层固件进行升级时，需将此拨动开关拨到升级工具固件处，当需通过 U8W 对目标芯片进行烧录程序，则需将拨动开关拨到烧录用户程序处。
（拨动开关连接方式请参考上图）
自动烧录机/分选机接口：是用于控制自动烧录机/分选机进行自动生产的控制接口。

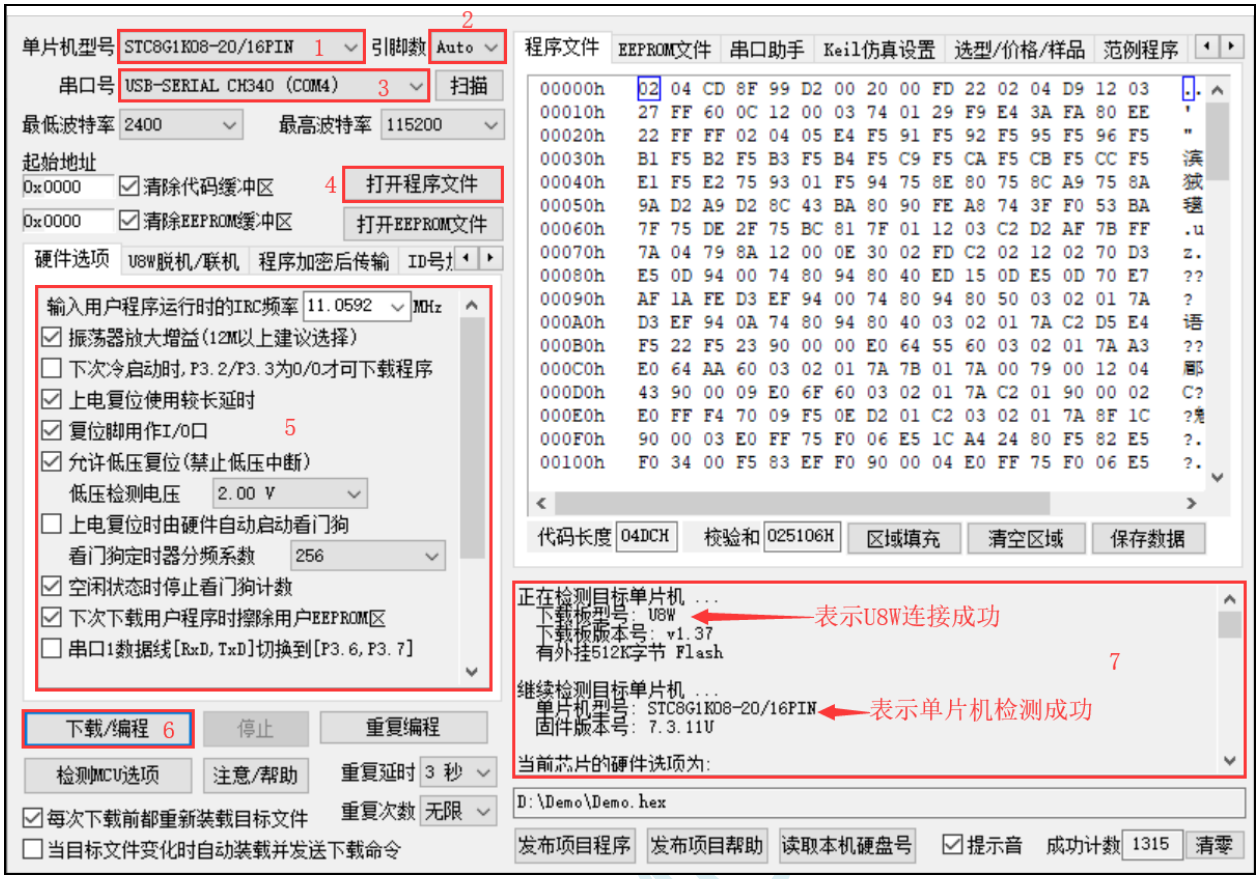
M.3.3 U8W 的在线联机下载使用说明

目标芯片安装于 U8W 锁紧座上并由 U8W 连接电脑进行在线联机下载

首先使用 STC 提供的 USB 连接线将 U8W 连接电脑，再将目标单片机按如下图所示的方向安装在 U8W 上：



然后使用 AIapp-ISP 下载软件下载程序，步骤如下：



1　选择单片机型号；

2　选择引脚数，芯片直接安装于 U8W 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；

3　选择 U8W 所对应的串口号；

4　打开目标文件（HEX 格式或者 BIN 格式）；

5　设置硬件选项；

6　点击“下载/编程”按钮开始烧录；

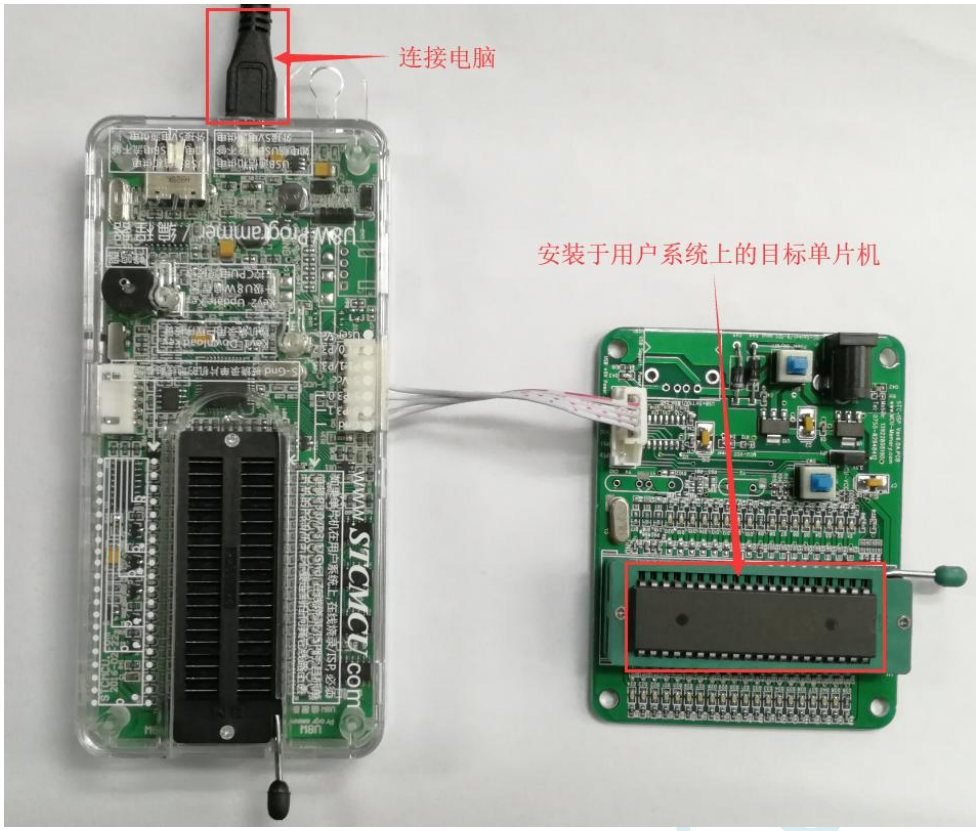
7　显示烧录过程的步骤信息，烧录完成提示“操作成功！”。

当信息框中有输出下载板的版本号信息以及外挂 Flash 的相应信息时，表示已正确检测到 U8W 下载工具。下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

建议用户用最新版本的 AIapp-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。

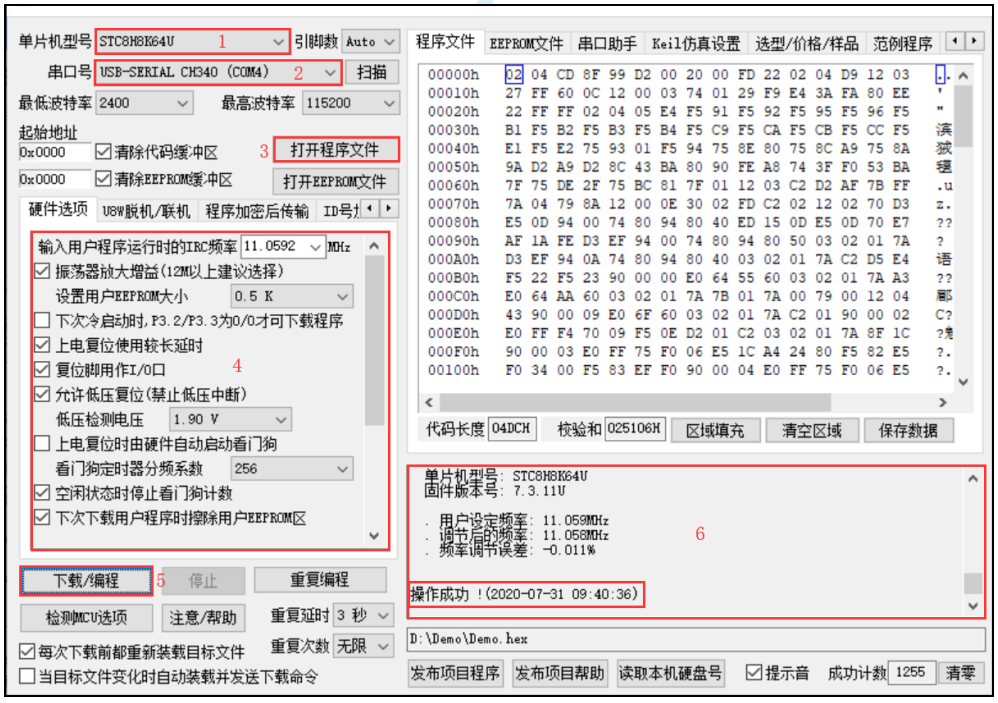
目标芯片通过用户系统引线连接 U8W 并由 U8W 连接电脑进行在线联机下载

首先使用 STC 提供的 USB 连接线将 U8W 连接电脑，再将 U8W 通过下载线与用户系统的目标单片机相连接，连接方式如下图所示：



然后使用 AIapp-ISP 下载软件下载程序，步骤如下：

1. 选择单片机型号；
2. 选择 U8W 所对应的串口号；
3. 打开目标文件（HEX 格式或者 BIN 格式）；
4. 设置硬件选项；
5. 点击“下载/编程”按钮开始烧录；
6. 显示烧录过程的步骤信息，烧录完成提示“操作成功！”。



当信息框中有输出下载板的版本号信息以及外挂 Flash 的相应信息时，表示已正确检测到 U8W 下载工具。下载的过程中， U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

建议用户用最新版本的 AIapp-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。

M.3.4 U8W 的脱机下载使用说明

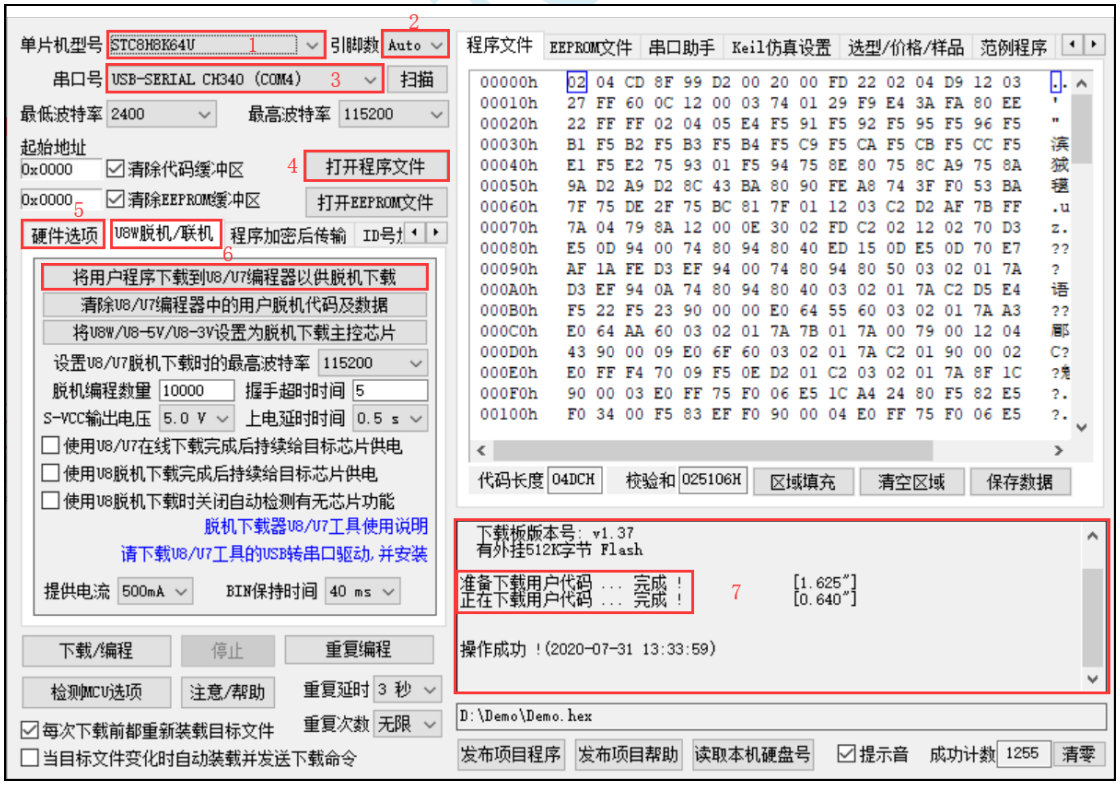
目标芯片安装于 U8W 座锁紧上并通过 USB 连接电脑给 U8W 供电进行脱机下载

使用 USB 给 U8W 供电从而进行脱机下载的步骤如下：

（1）使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图：



（2）在 AIapp-ISP 下载软件中按如下图所示的步骤进行设置：



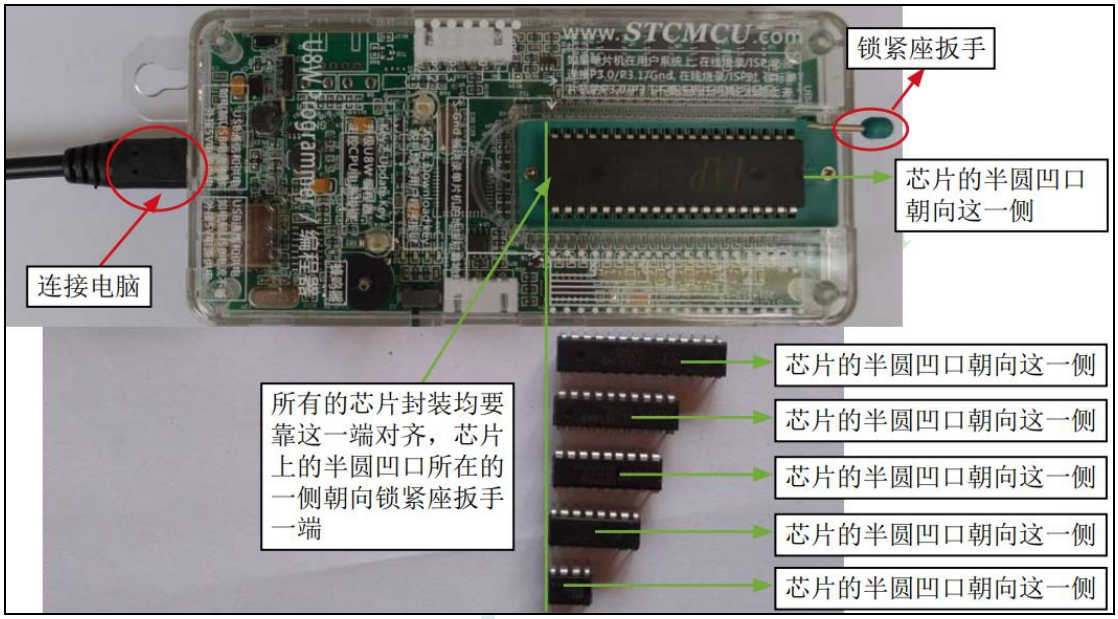
1. 选择单片机型号；

2. 选择引脚数，芯片直接安装于 U8W 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

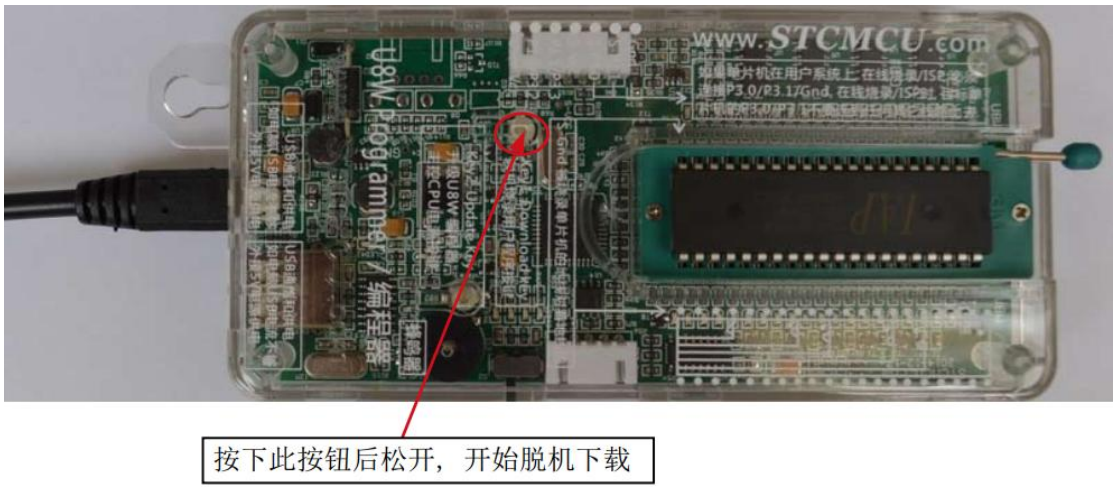
按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W 下载工具中。

建议用户用最新版本的 AIapp-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。

（3）再将目标单片机如下图所示的方向放在 U8W 下载工具，如下图所示：



（4）然后按下如下图所示的按钮后松开，即可开始脱机下载：



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

脱机下载即插即用烧录功能介绍：

1. 以上步骤完成(1)、(2)步之后 U8W 连接电脑上电时默认处于即插即用烧录状态；

2. 按照第(3)步指示将芯片放入烧录座，在锁紧座扳手的同时，U8W 会自动开始烧录；
3. 通过指示灯显示烧录过程跟烧录结果；
4. 烧录完成后松开座扳手，取出芯片；
5. 重复 2，3，4 步骤可进行连续烧录，省掉按按钮触发烧录的动作。

目标芯片由用户系统引线连接 U8W 并通过 USB 连接电脑给 U8W 供电进行脱机下载

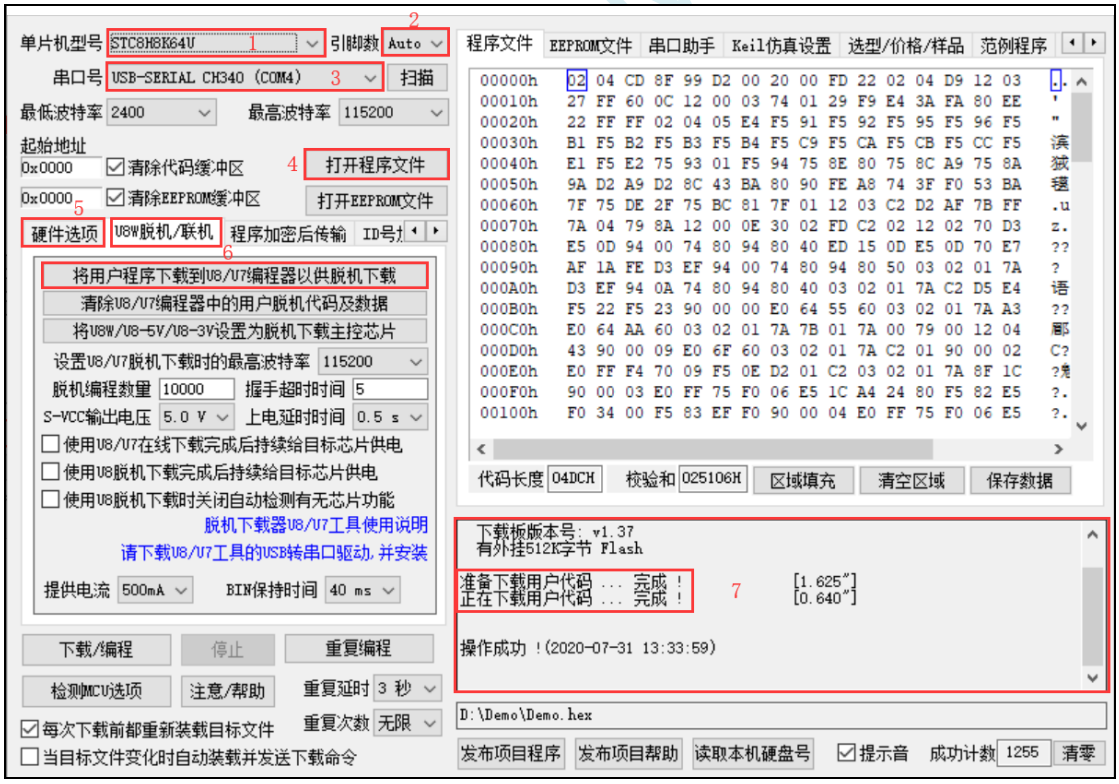
使用 USB 给 U8W 供电从而进行脱机下载的步骤如下：

（1）使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图：



（2）在 AIapp-ISP 下载软件中按如下图所示的步骤进行设置：

建议用户用最新版本的 AIapp-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。

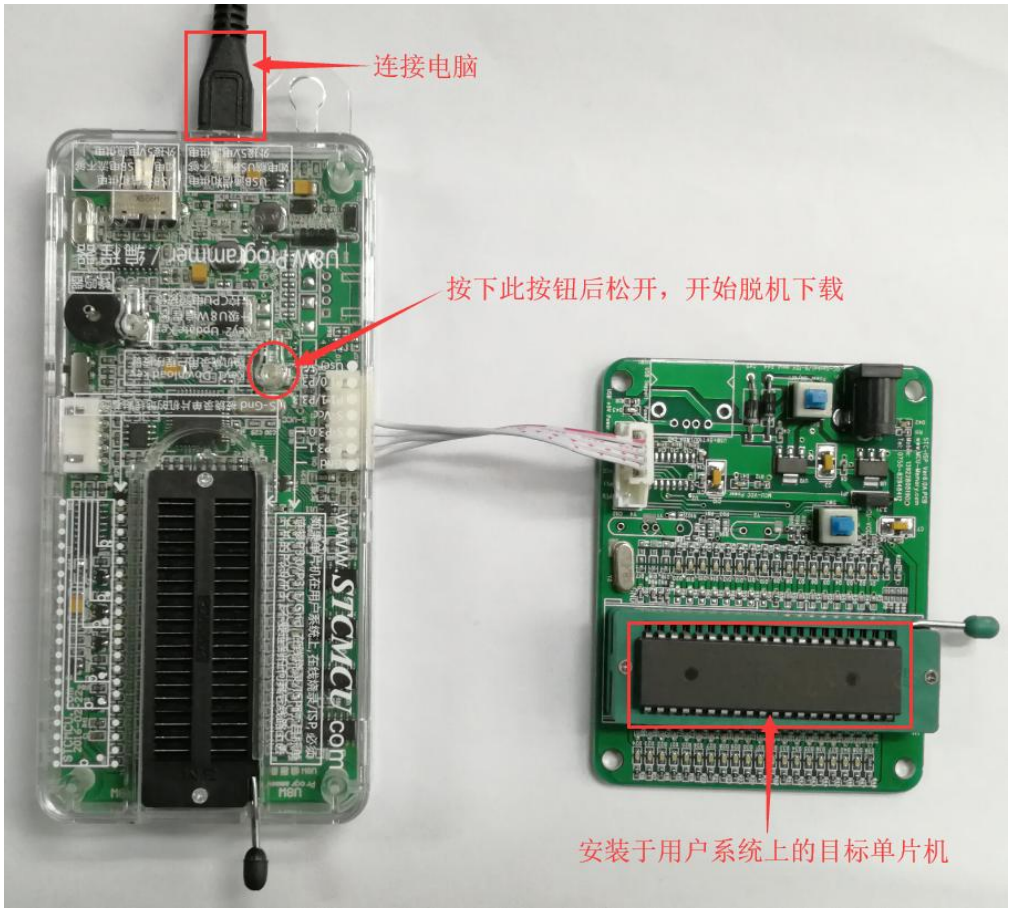


1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；

5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W 下载工具中。

（3）然后使用连接线连接电脑、将 U8W 下载工具以及用户系统（目标单片机）如下图所示的方式连接起来，并按下图所示的按钮后松开，即可开始脱机下载：



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

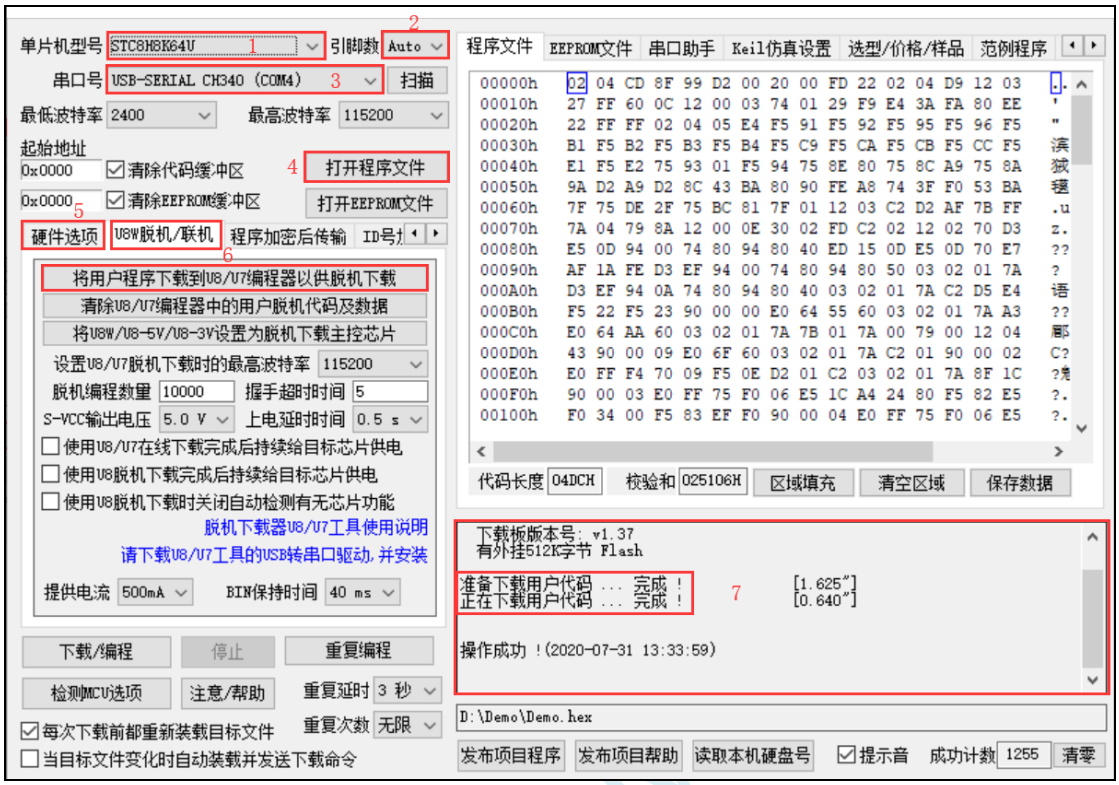
目标芯片由用户系统引线连接 U8W 并通过用户系统给 U8W 供电进行脱机下载

（1）首先使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图：



（2）在 AIapp-ISP 下载软件中按如下图所示的步骤进行设置：

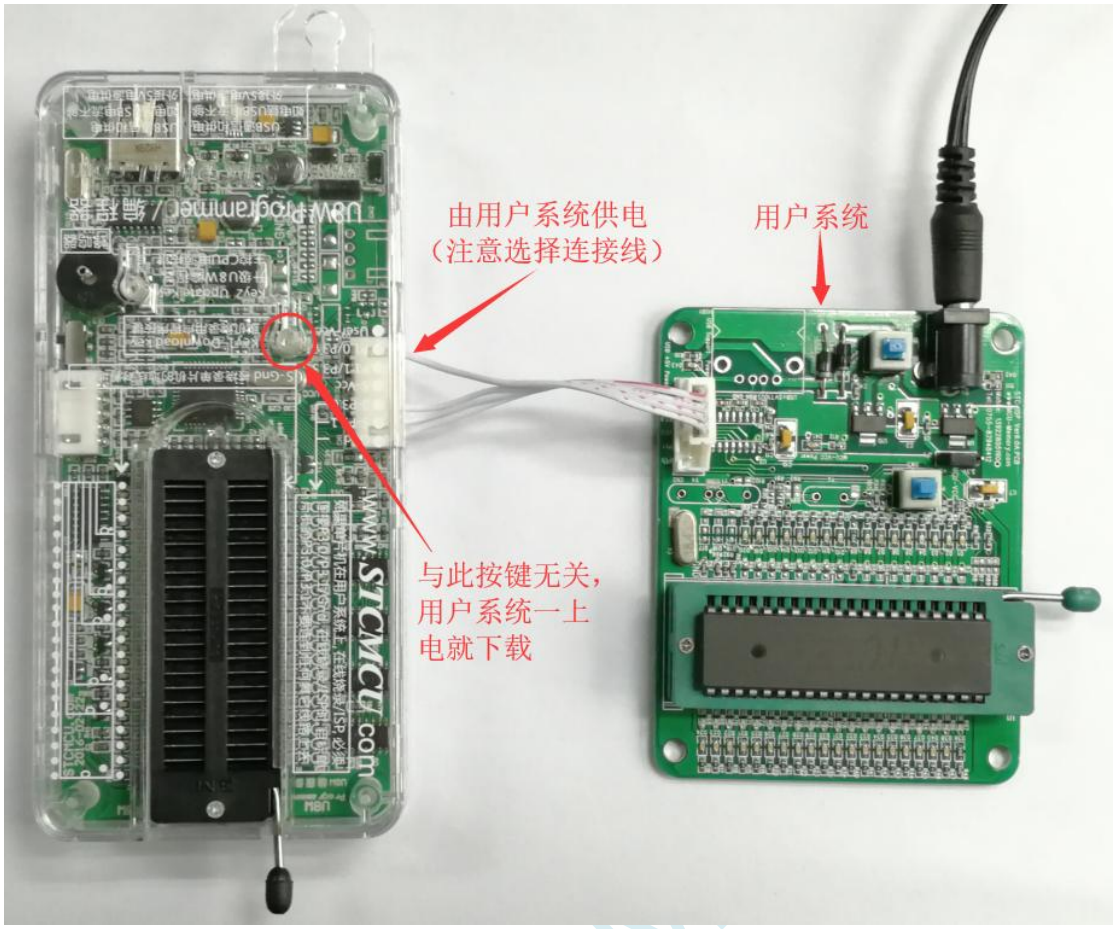
建议用户用最新版本的 AIapp-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。



1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W 下载工具中。

（3）然后按下图所示的方式连接 U8W 与用户系统，给用户系统供电，即可开始脱机下载：



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

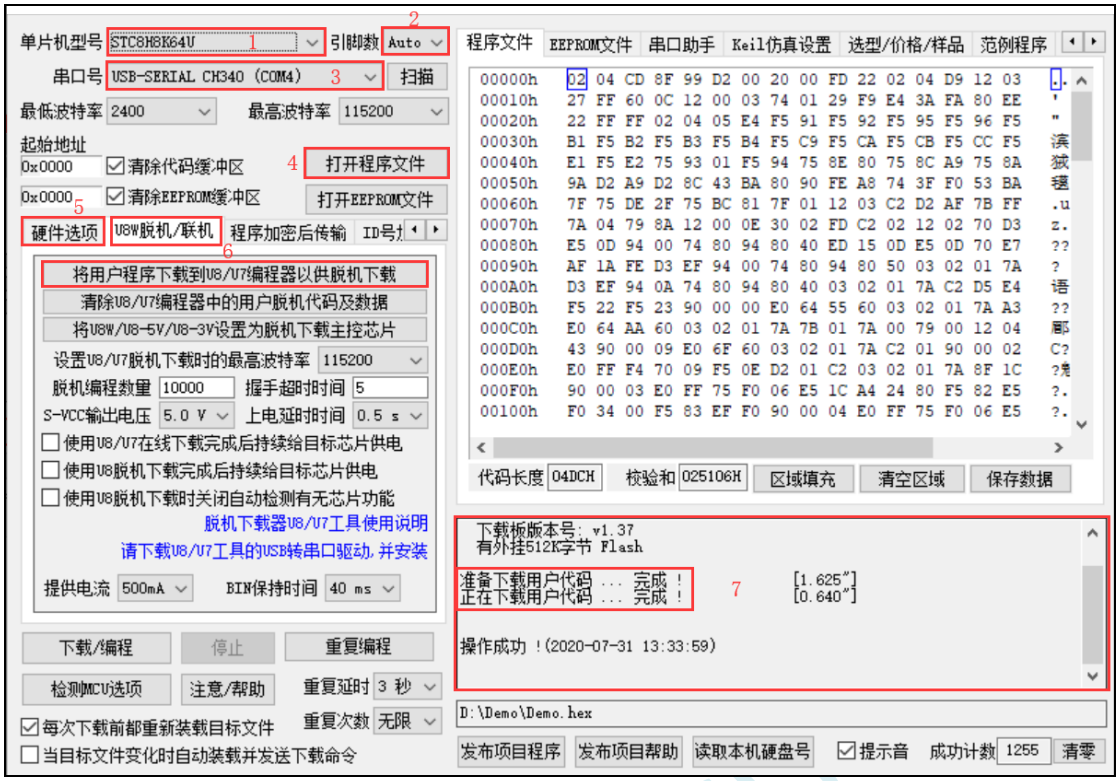
目标芯片由用户系统引线连接 U8W 且 U8W 与用户系统各自独立供电进行脱机下载

（1）首先使用 STC 提供的 USB 连接线将 U8W 下载板连接到电脑，如下图：



（2）在 AIapp-ISP 下载软件中按如下图所示的步骤进行设置：

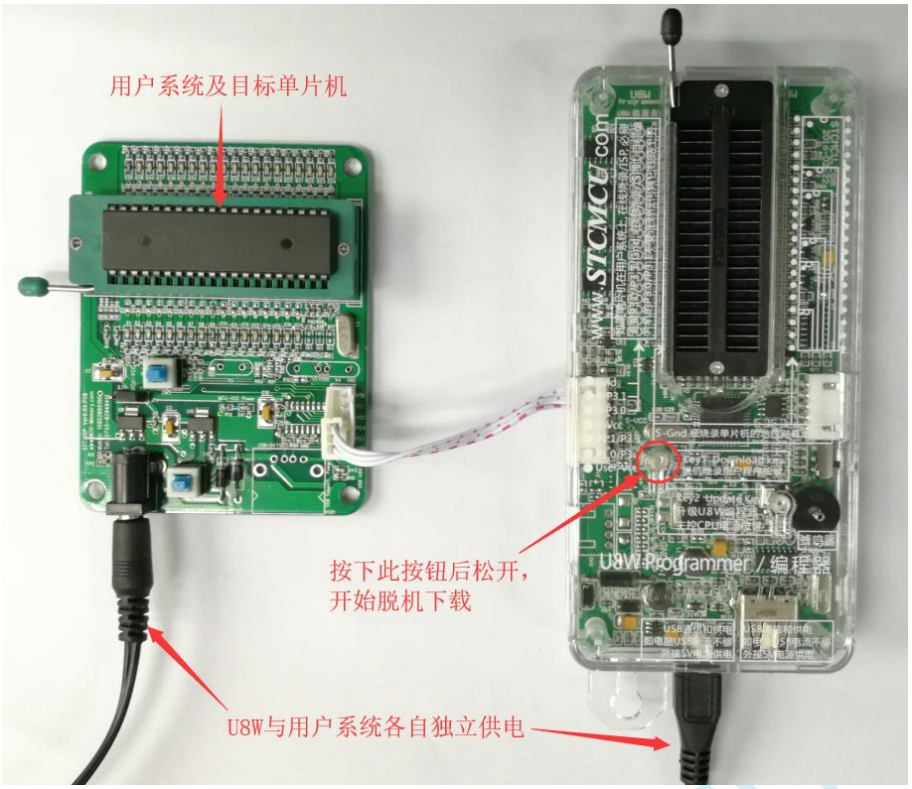
建议用户用最新版本的 AIapp-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。



1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W 下载工具中。

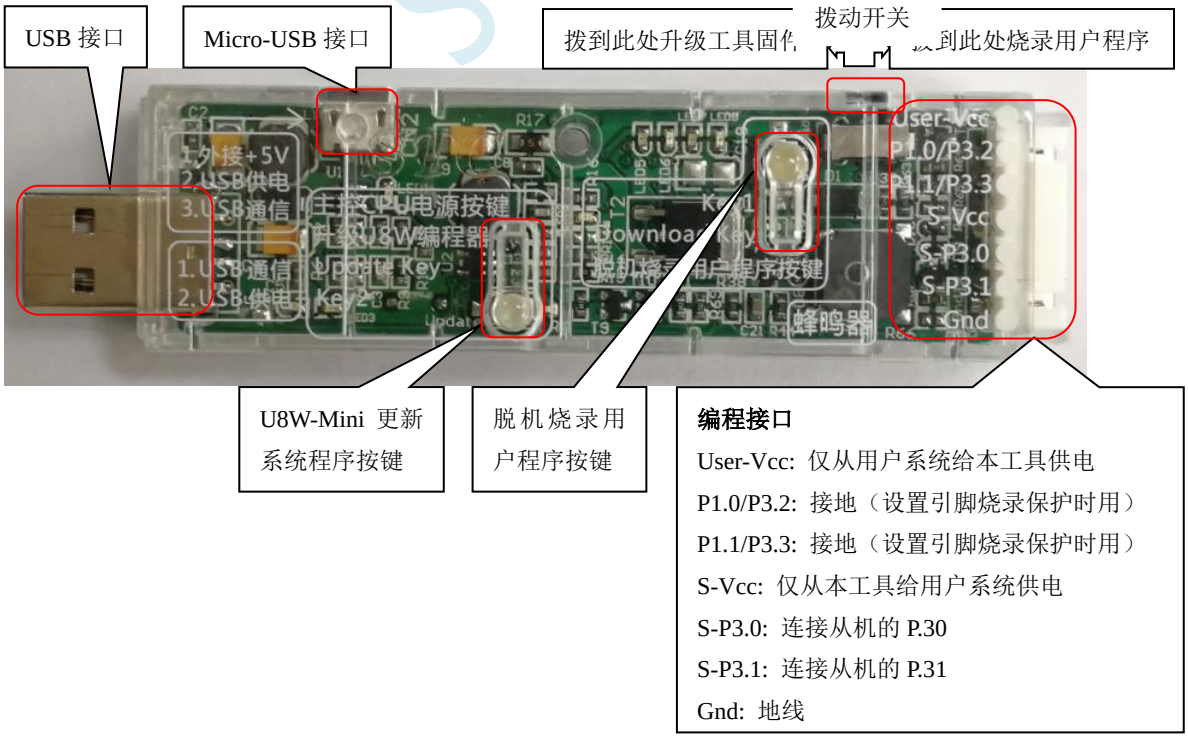
（3）然后按下图所示的方式连接 U8W 与用户系统，并将图中所示按钮先按下后松开，准备开始脱机下载，最后给用户系统上电/开电源，下载用户程序正式开始：



下载的过程中，U8W 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

M.3.5 U8W-Mini 的功能介绍

下面详细介绍 U8W-Mini 工具的各主要接口及功能：



编程接口：根据不同的供电方式，使用不同的下载连接线连接 U8W-Mini 下载板和用户系统。

U8W-Mini 更新系统程序按键：用于更新 U8W-Mini 工具，当有新版本的 U8W 固件时，需要按下此按键对 U8W-Mini 的主控芯片进行更新（注意：必须先将更新/下载选择接口上的拨动开关拨动到升级工具固件）。

脱机下载用户程序按钮：开始脱机下载按钮。首先 PC 将脱机代码下载到 U8W-Mini 上，然后使用下载连接线将用户系统连接到 U8W-Mini，再按下此按钮即可开始脱机下载（每次上电时也会立即开始下载用户代码）。

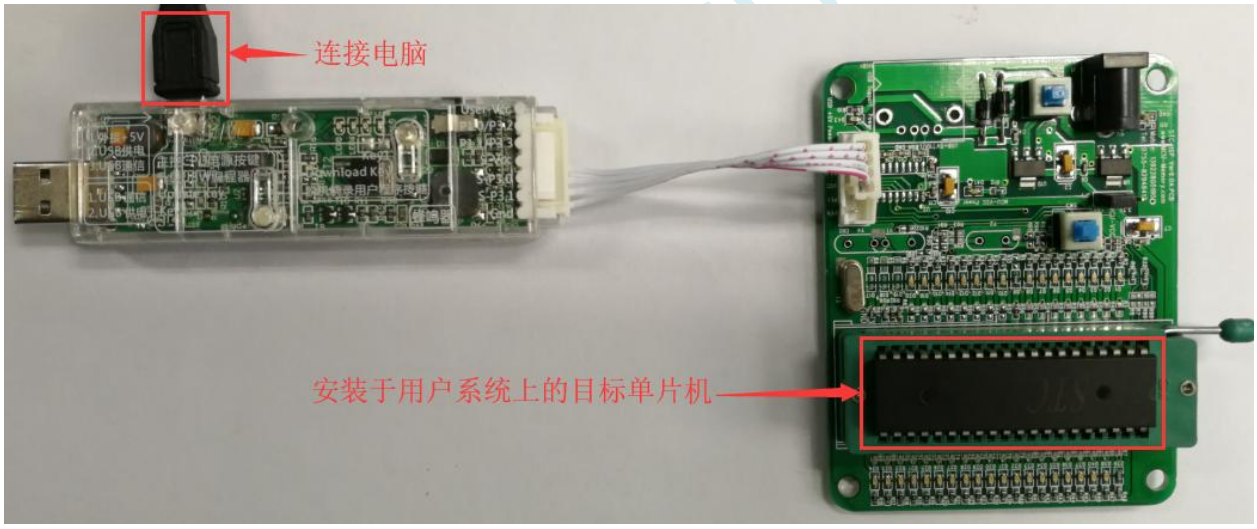
更新/下载选择接口：当需要对 U8W-Mini 的底层固件进行升级时，需将此拨动开关拨动到升级工具固件处，当需通过 U8W-Mini 对目标芯片进行烧录程序，则需将拨动开关拨动到烧录用户程序处。（拨动开关连接方式请参考上图）

USB 接口：USB 接口与 Micro-USB 接口是相同的功能，用户根据需要连接其中一个接口到电脑即可。

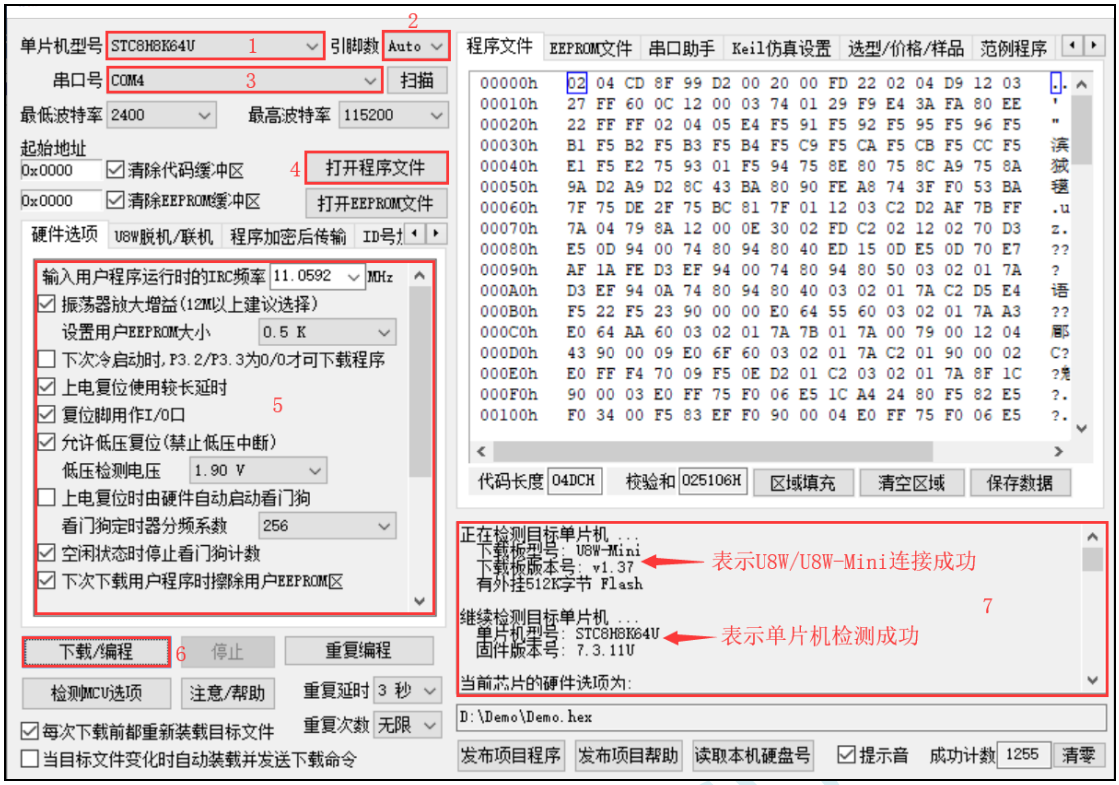
M.3.6 U8W-Mini 的在线联机下载使用说明

目标芯片通过用户系统引线连接 U8W-Mini 并由 U8W-Mini 连接电脑进行在线联机下载

首先使用 STC 提供的 USB 连接线将 U8W-Mini 连接电脑，再将 U8W-Mini 通过下载线与用户系统的目标单片机相连接，连接方式如下图所示：



然后使用 AIapp-ISP 下载软件下载程序，步骤如下：



1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W-Mini 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W-Mini 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 点击“下载/编程”按钮开始烧录；
7. 显示烧录过程的步骤信息，烧录完成提示“操作成功！”。

当信息框中有输出下载板的版本号信息以及外挂 Flash 的相应信息时，表示已正确检测到 U8W-Mini 下载工具。

下载的过程中，U8W-Mini 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

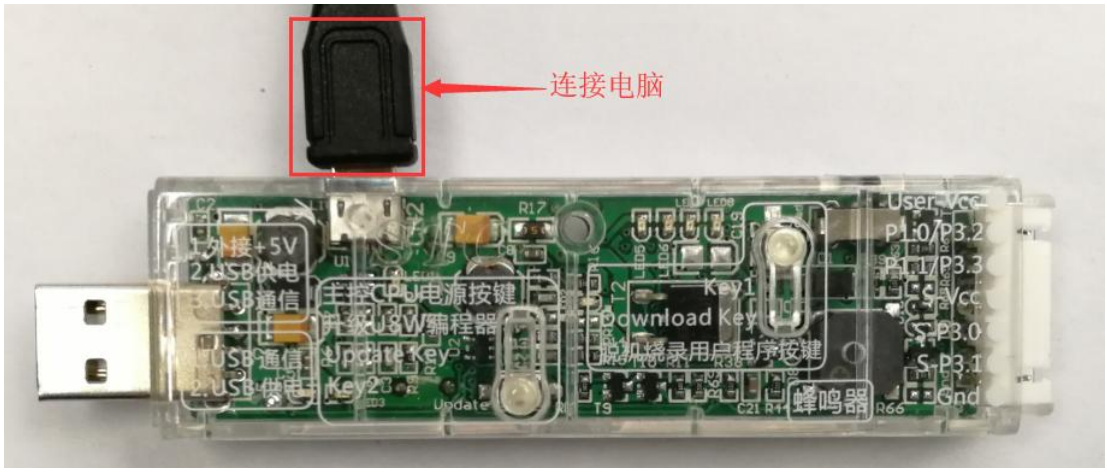
建议用户用最新版本的 AIapp-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。

M.3.7 U8W-Mini 的脱机下载使用说明

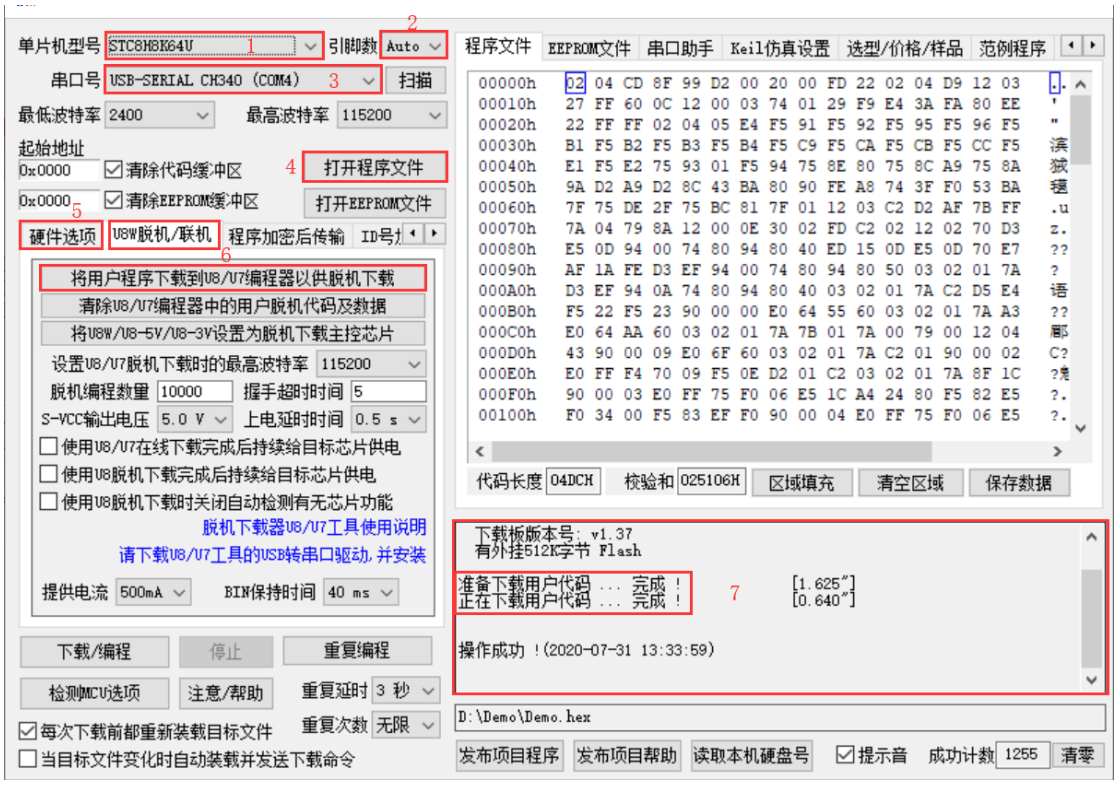
目标芯片由用户系统引线连接 U8W-Mini 并通过 USB 连接电脑给 U8W-Mini 供电进行脱机下载

使用 USB 给 U8W-Mini 供电从而进行脱机下载的步骤如下：

（1）使用 STC 提供的 USB 连接线将 U8W-Mini 下载板连接到电脑，如下图：



（2）在 AIapp-ISP 下载软件中按如下图所示的步骤进行设置：

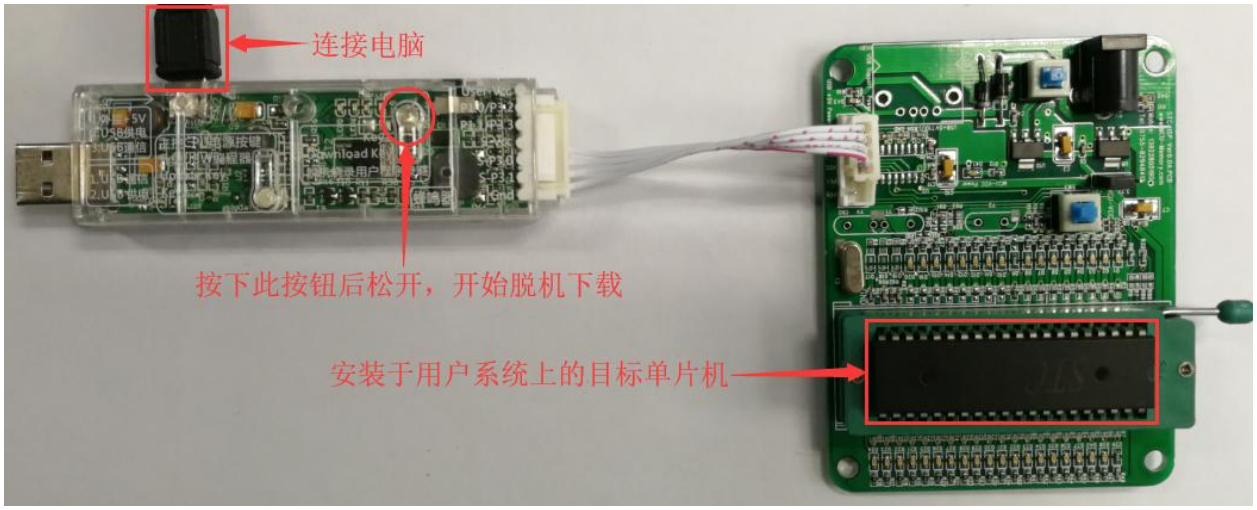


1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W-Mini 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W-Mini 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W-Mini 下载工具中。

建议用户用最新版本的 AIapp-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。

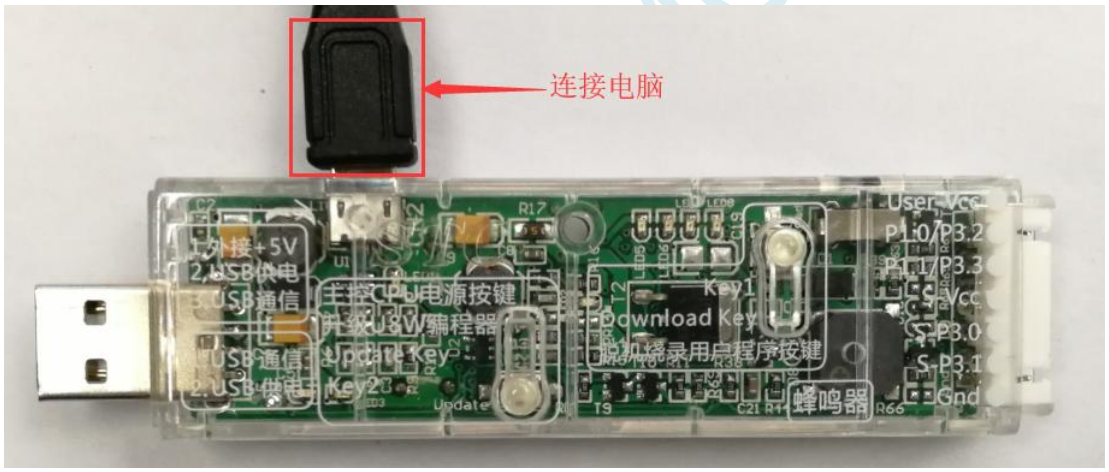
（3）然后使用连接线连接电脑、将 U8W-Mini 下载工具以及用户系统（目标单片机）如下图所示的方式连接起来，并按下图所示的按钮后松开，即可开始脱机下载：



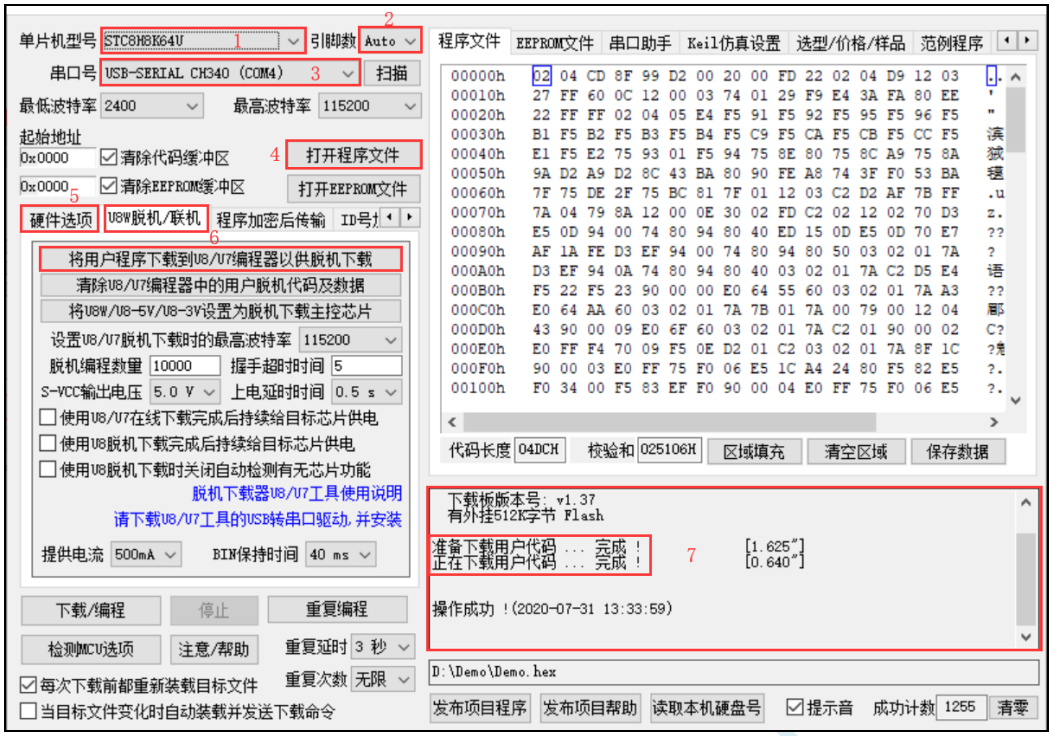
下载的过程中，U8W-Mini 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

目标芯片由用户系统引线连接 U8W-Mini 并通过用户系统给 U8W-Mini 供电进行脱机下载

（1）首先使用 STC 提供的 USB 连接线将 U8W-Mini 下载板连接到电脑，如下图：



（2）在 AIapp-ISP 下载软件中按如下图所示的步骤进行设置：

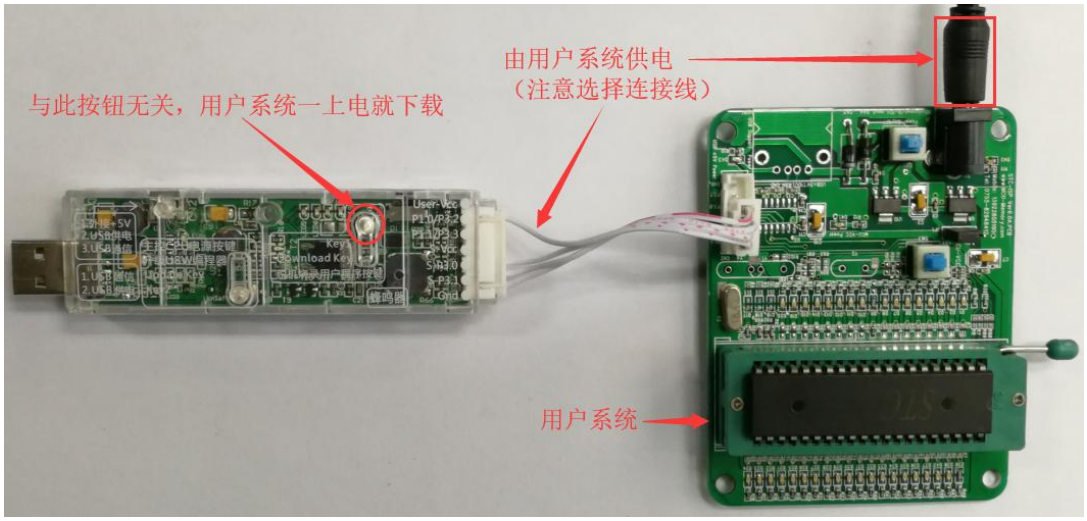


1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W-Mini 上下载时，一定要选择正确的引脚数，否则将会下载失败；
3. 选择 U8W-Mini 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；
7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W-Mini 下载工具中。

建议用户用最新版本的 AIapp-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。

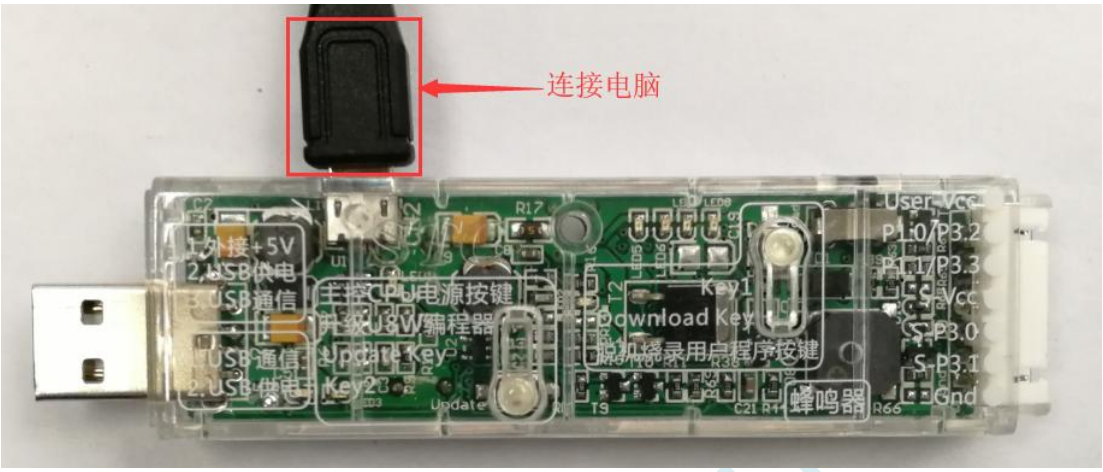
（3）然后按下图所示的方式连接 U8W-Mini 与用户系统，用户系统一上电就开始脱机下载：



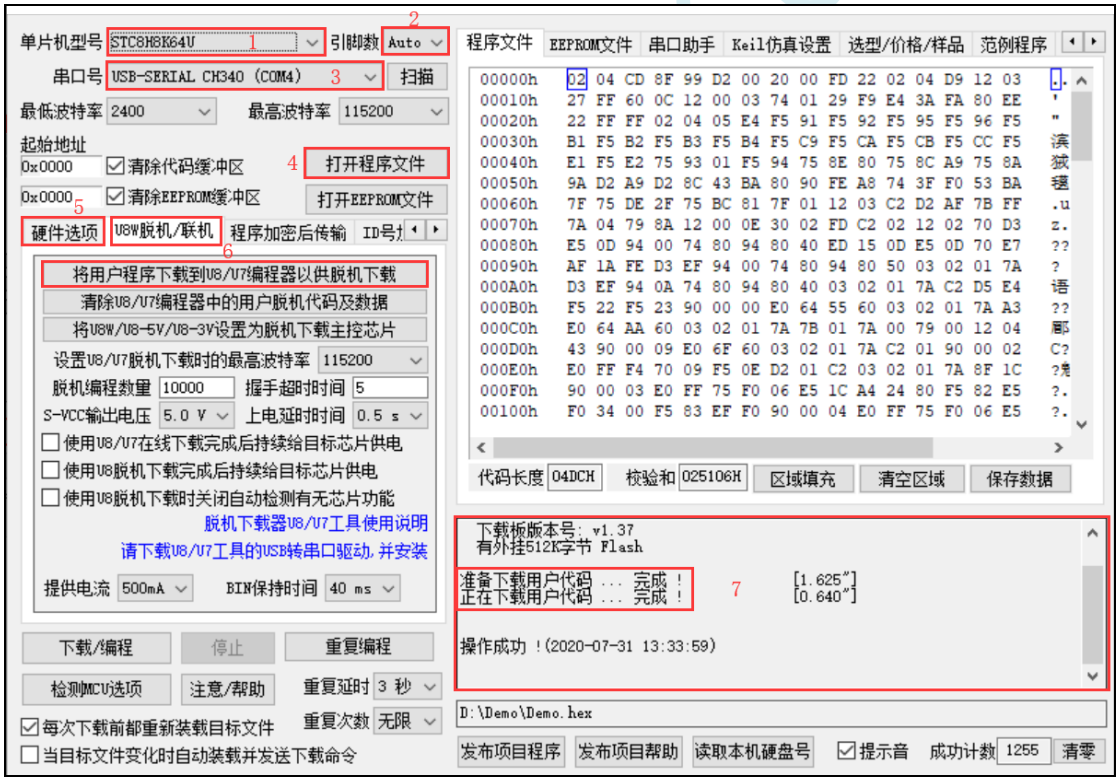
下载的过程中，U8W-Mini 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

目标芯片由用户系统引线连接 U8W-Mini 且 U8W-Mini 与用户系统各自独立供电进行脱机下载

（1）首先使用 STC 提供的 USB 连接线将 U8W-Mini 下载板连接到电脑，如下图：



（2）在 AIapp-ISP 下载软件中按如下图所示的步骤进行设置：



1. 选择单片机型号；
2. 选择引脚数，芯片直接安装于 U8W-Mini 上下载时，一定要注意选择正确的引脚数，否则将会下载失败；
3. 选择 U8W-Mini 所对应的串口号；
4. 打开目标文件（HEX 格式或者 BIN 格式）；
5. 设置硬件选项；
6. 选择“U8W 脱机/联机”标签，设置脱机编程选项，注意 S-VCC 输出电压与目标芯片工作电压匹配；

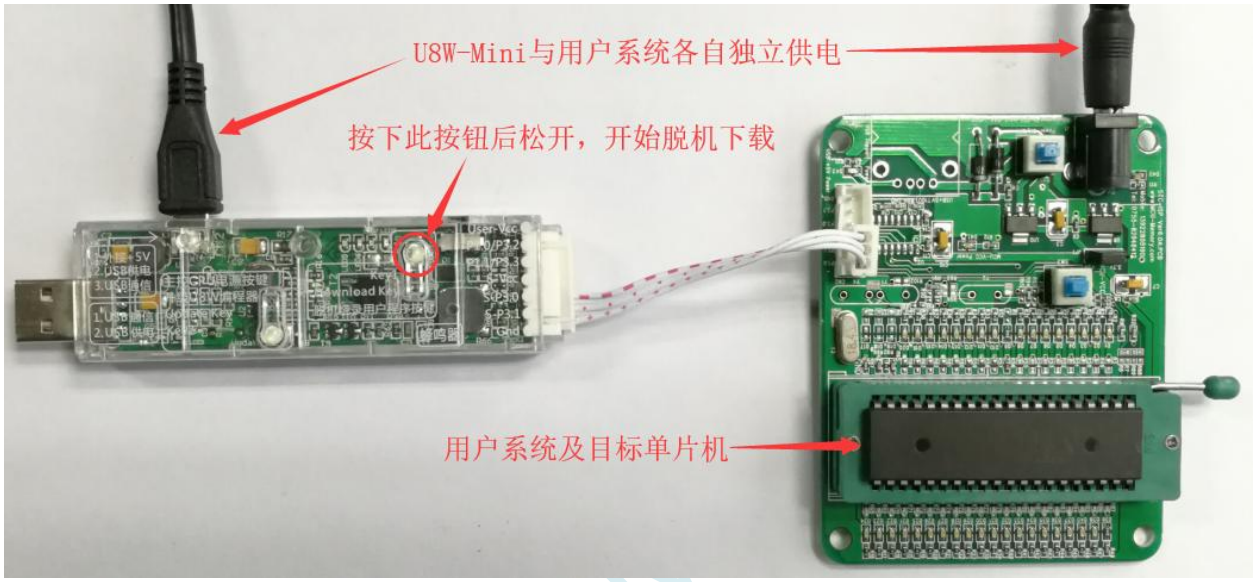
点击“将用户程序下载到 U8/U7 编程器以供脱机下载”按钮；

7. 显示设置过程的步骤信息，设置完成提示“操作成功！”。

按照上图的步骤，操作完成后，若下载成功则表示用户代码和相关的设置选项都已下载到 U8W-Mini 下载工具中。

建议用户用最新版本的 AIapp-ISP 下载软件（请随时留意 STC 官方网站 http://www.STCAI.com 中 AIapp-ISP 下载软件的更新，强烈建议用户在官方网站 http://www.STCAI.com 中下载最新版本的软件使用）。

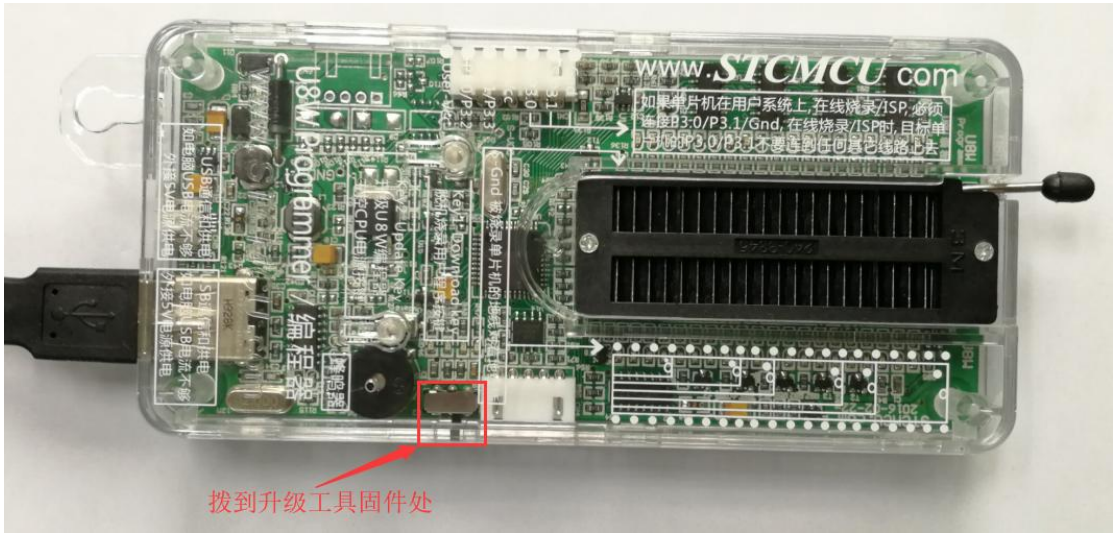
（3）然后按下图所示的方式连接 U8W-Mini 与用户系统，并将图中所示按钮先按下后松开，准备开始脱机下载，最后给用户系统上电/开电源，下载用户程序正式开始：



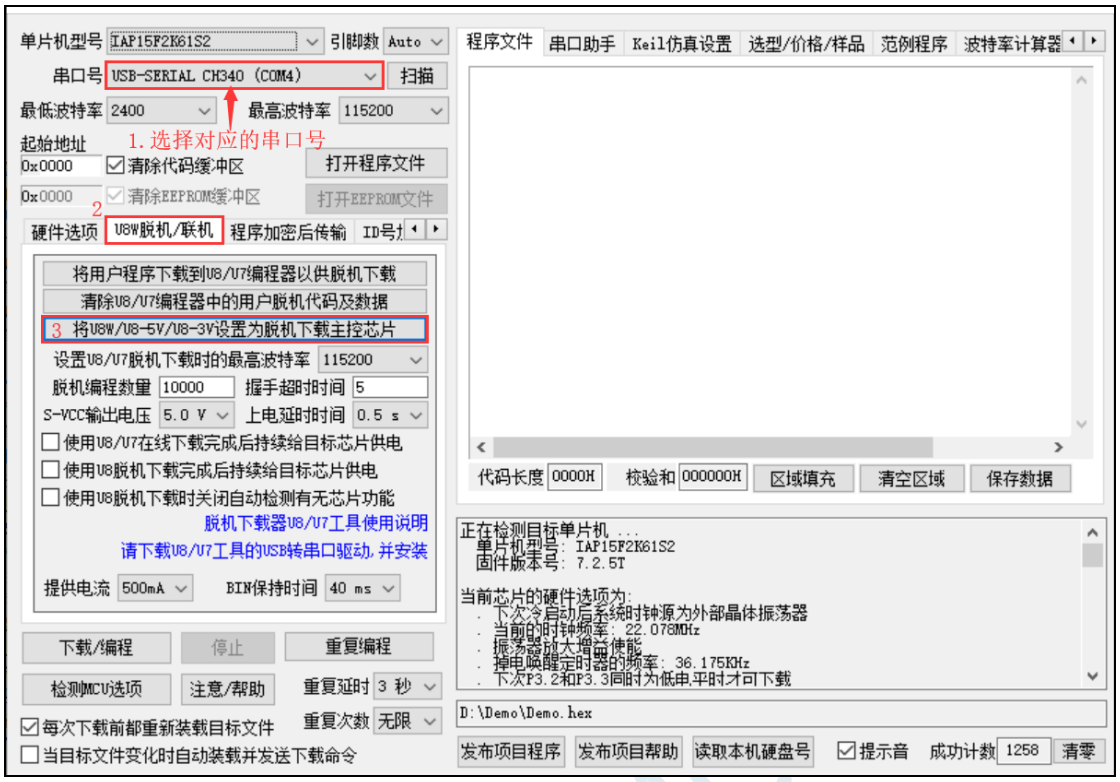
下载的过程中，U8W-Mini 下载工具上的 4 个 LED 会以跑马灯的模式显示。下载完成后，若下载成功，则 4 个 LED 会同时亮、同时灭；若下载失败，则 4 个 LED 全部不亮。

M.3.8 制作/更新 U8W/U8W-Mini

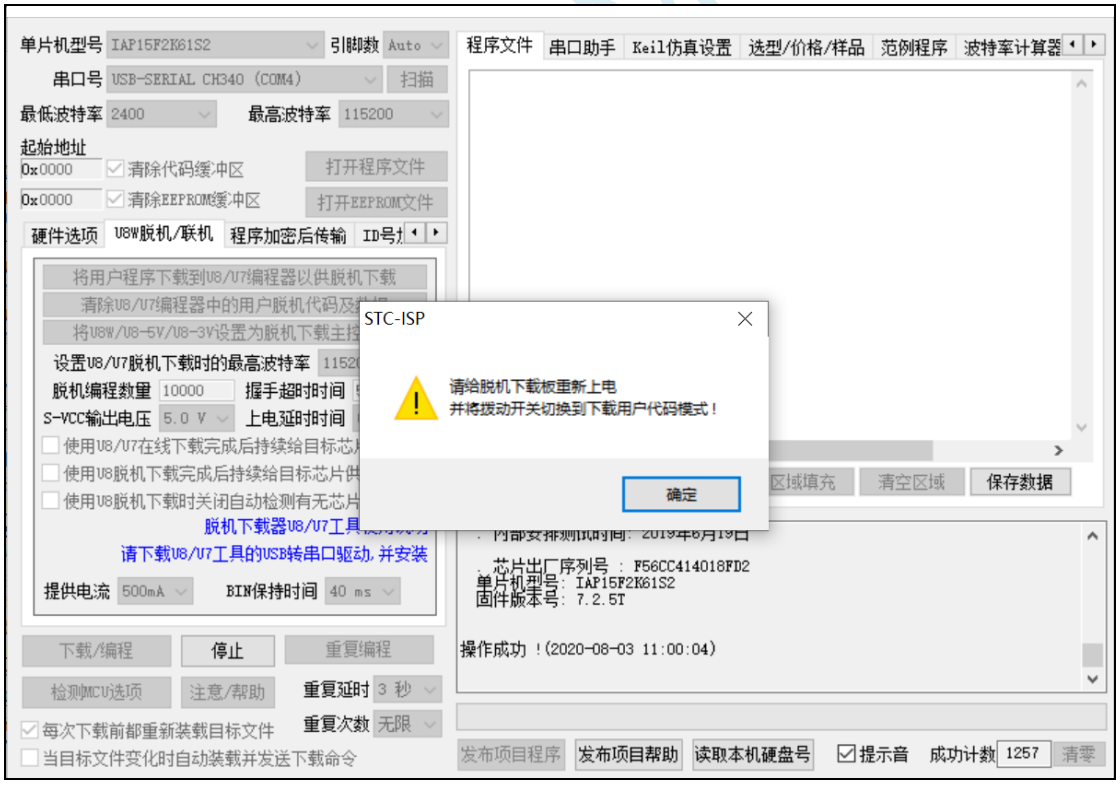
制作 U8W/U8W-Mini 下载母片的过程类似，为节约篇幅，下文以 U8W 为例，详述如何制作 U8W 下载母片。在制作 U8W 下载母片之前需要将 U8W 下载板的“更新/下载选择接口”拨到“升级工具固件”，如下图所示：



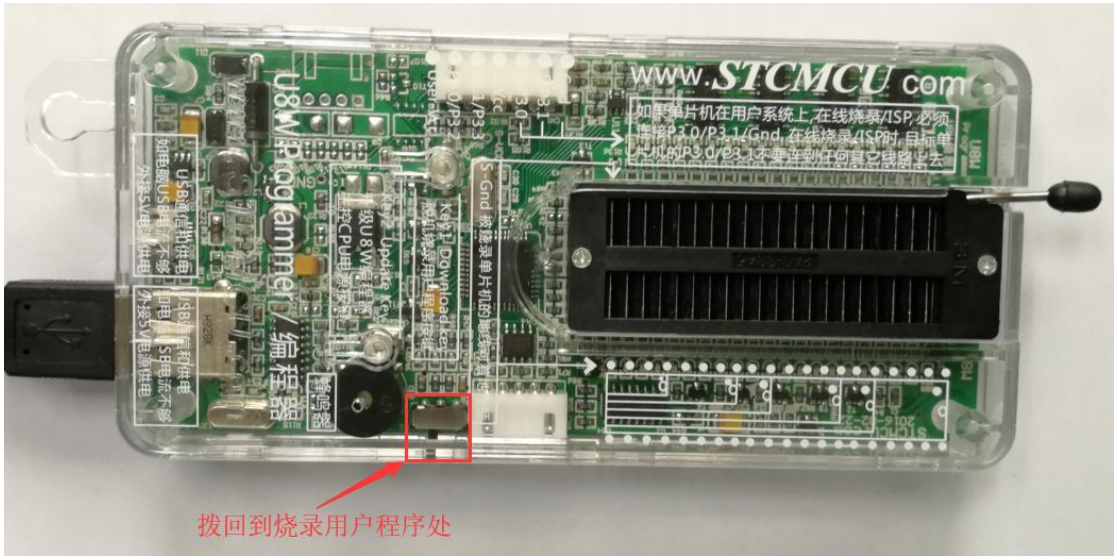
然后在 AIapp-ISP 下载程序中的“U8W 脱机/联机”页面中点击“将 U8W/U8-5V/U8-3V 设置为脱机下载主控芯片”按钮，如下图：（注意：一定要选择 U8W 所对应的串口）



在出现如下画面表示 U8W 控制芯片制作完成：



制作完成后，一定不要忘记将 U8W 的“更新/下载选择接口”拨回到“烧录用户程序”模式，并将 U8W 下载工具重新上电，如下图所示：（否则将不能正常进行烧录）



M.3.9 U8W/U8W-Mini 设置直通模式（可用于仿真）

若要使用 U8W/U8-Mini 进行仿真，首先必须将 U8W/U8-Mini 设置为直通模式。U8W/U8W-Mini 实现 USB 转串口直通模式的方法如下：

1. 首先 U8W/U8W-Mini 固件必须升级到 v1.37 及以上版本；
2. U8W/U8W-Mini 上电后为正常下载模式，此时按住工具上的 Key1（下载）按键不要松开，再按一下 Key2（电源）按键，然后放开 Key2（电源）按键 后，再松开 Key1（下载）按键，U8W/U8W-Mini 会进入 USB 转串口直通模式。（按下 Key1 ➔ 按下 Key2 ➔ 松开 Key2 ➔ 松开 Key1）；
3. 进入直通模式的 U8W/U8W-Mini 工具只是简单的 USB 转串口不具备脱机下载功能，若需要恢复 U8W/U8W-Mini 的原有功能，只需要再次单独按一下 Key2（电源）按键即可。

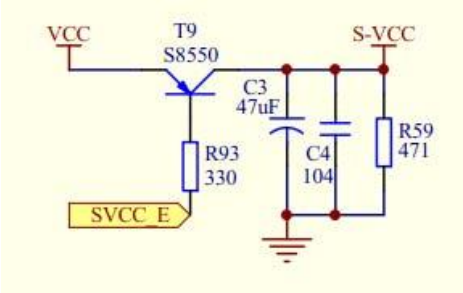
M.3.10　U8W/U8W-Mini 的参考电路

USB 型联机/脱机下载板 U8W/U8W-Mini 为用户提供了如下的常用控制接口：

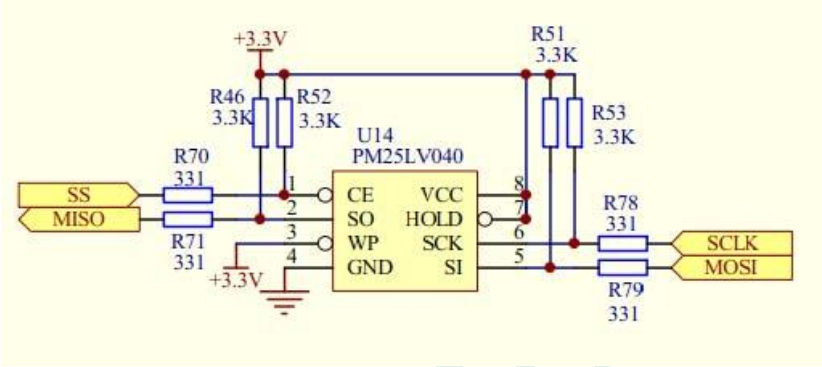
脚位功能	端口	功能描述
电源控制脚	P2.6	低位有效
下载通讯脚	P1.0	串口 RXD，连接目标芯片的 TXD（P3.1）
	P1.1	串口 TXD，连接目标芯片的 RXD（P3.0）
编程按键	P3.6	低有效
显示	P3.2	LED1
	P3.3	LED2
	P3.4	LED3
	P5.5	LED4
外挂串行 Flash 控制脚	P2.4	Flash 的 CE 脚
	P2.2	Flash 的 SO 脚
	P2.3	Flash 的 SI 脚
	P2.1	Flash 的 SCLK 脚

脚位功能	端口	功能描述
全自动烧录工具 分选机信号	P3.6	起始信号
	P1.5	完成信号
	P5.4	OK 信号（良品信号）
	P3.7	ERROR 信号（不良品信号）
蜂鸣器（BEEP）控制	P2.5	高有效（高电平发出声音）

电源控制部分参考电路图：

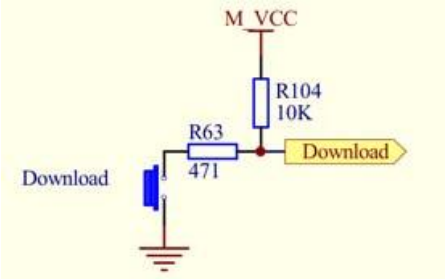


Flash 控制部分参考电路图：

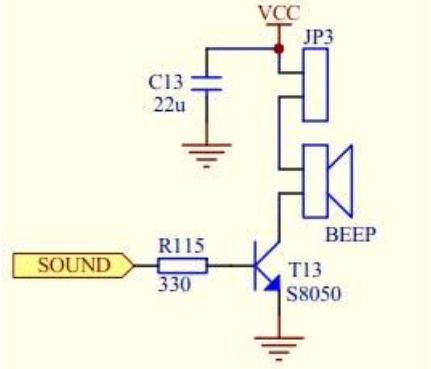


用户程序大于 41K 时需要此 Flash 存储器

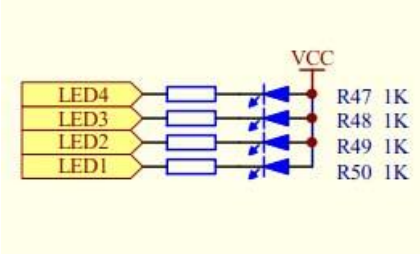
按键部分参考电路图：



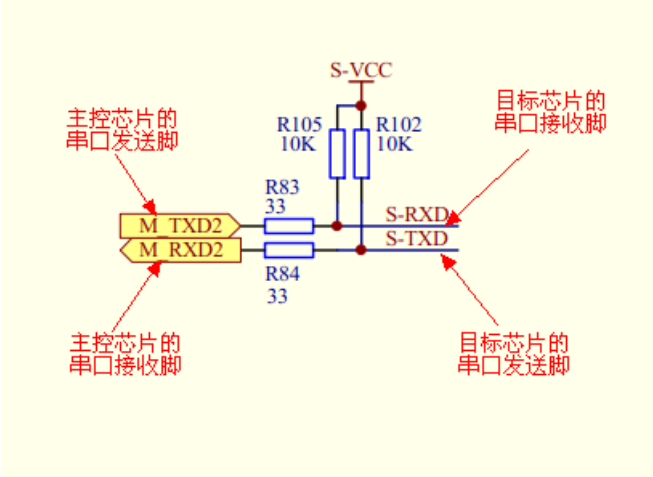
蜂鸣器部分参考电路图：



LED 显示部分参考电路图：



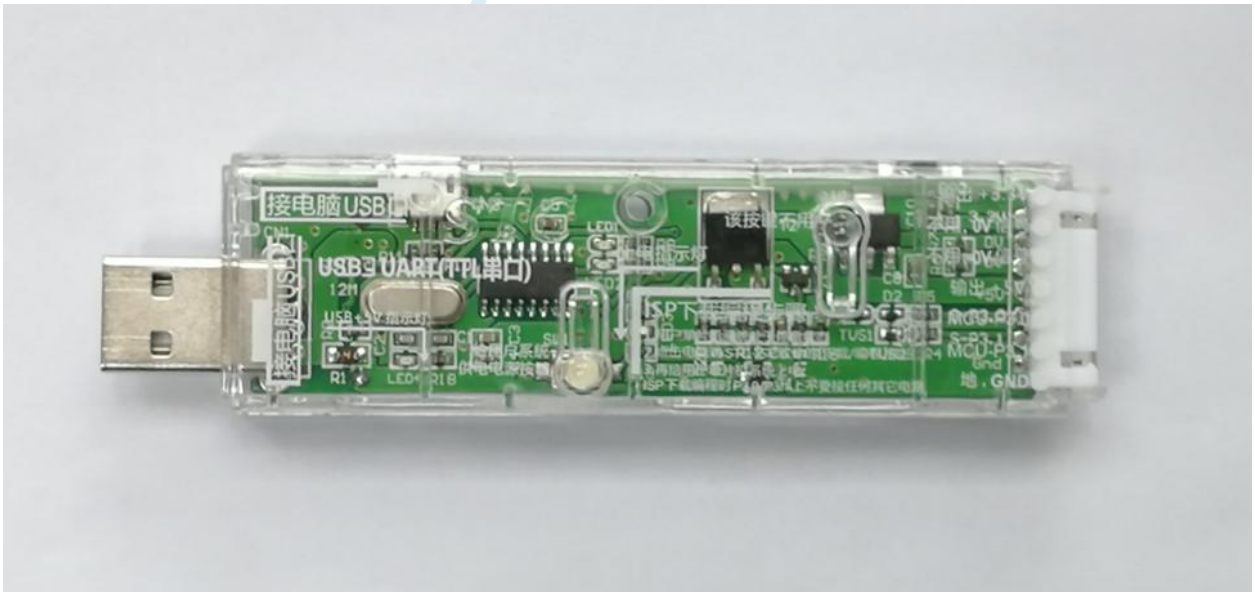
串口通讯脚连接部分参考电路图：



M.4　STC 通用 USB 转串口工具

M.4.1 STC 通用 USB 转串口工具外观图

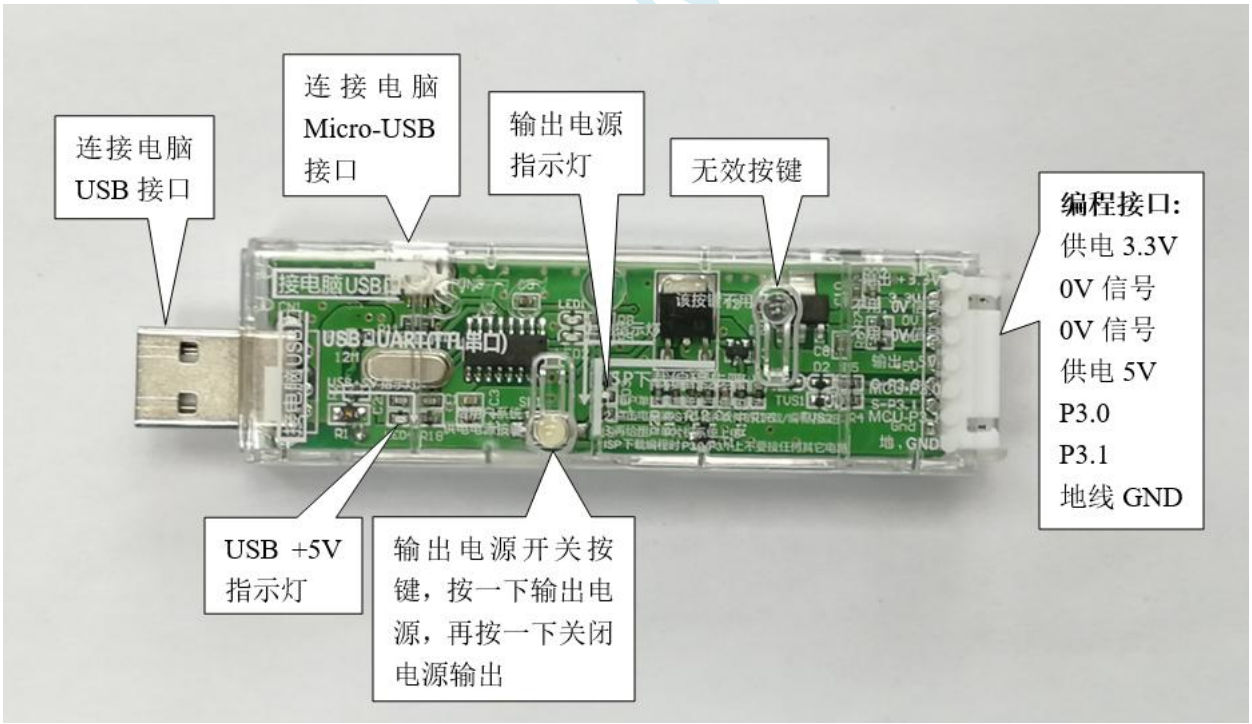
正面：



背面：



M.4.2 STC 通用 USB 转串口工具布局图



在此，需要对“电源开关”进行说明：

此按钮的作用与自锁开关相同，在开关按钮第一次按时，开关接通电源并保持，即自锁，在开关按钮第二次按时，开关断开电源。鉴于自锁开关使用过程中容易损坏的特点，我们设计了一套利用轻触开关替代自锁开关功能的电路，提高工具的使用寿命。

而对于 STC 的单片机，要想进行 ISP 下载，则必须是在上电复位时接收到串口命令才会开始执行 ISP

程序，所以使用 STC 通用 USB 转串口工具下载程序到 MCU 的正确步骤为：

1. 使用 STC 通用 USB 转串口工具将待烧录 MCU 与电脑进行连接；
2. 打开 STC 的 ISP 下载软件；
3. 选择单片机型号；
4. 选择 STC 通用 USB 转串口工具所对应的串口；
5. 打开目标文件（HEX 格式或者 BIN 格式）；
6. 点击 ISP 下载软件中的“下载/编程”按钮；
7. 按一下 STC 通用 USB 转串口工具上的“电源开关”给 MCU 供电，即可开始下载。
【冷启动烧录】

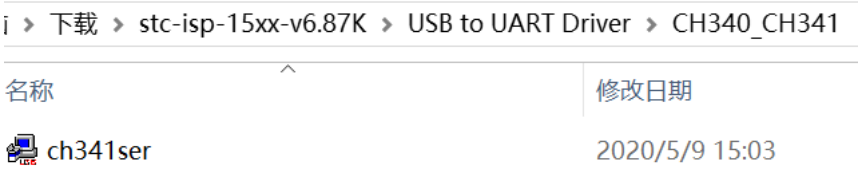
此外，USB 接口与 Micro-USB 接口是相同的功能，用户根据需要连接其中一个接口到电脑即可。
编程接口的 0V 信号脚内部有 470 欧姆电阻接地，如果设置了 P1.0/P1.1=0/0 或者 P3.2/P3.3=0/0 时才能下载，可将 P1.0，P1.1 或者 P3.2，P3.3 接到 0V 信号脚。

M.4.3 STC 通用 USB 转串口工具驱动安装

STC 通用 USB 转串口工具采用 CH340 USB 转串口芯片（可以外挂晶振，更精准），只要下载通用的 CH340 串口驱动程序进行安装即可，以下是 STC 官网（www.STCMCUDATA.com）提供的 CH341SER 串口驱动下载位置：



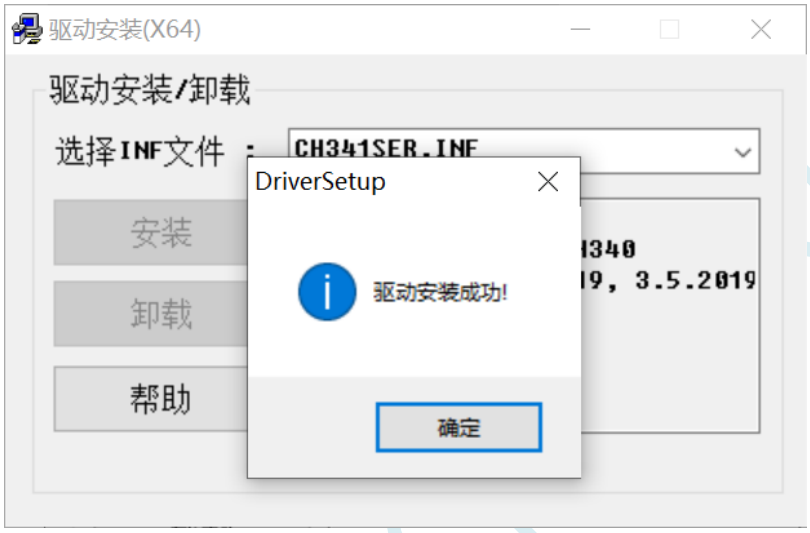
下载后进行解压，CH340 的驱动安装包路径 stc-isp-15xx-v6.87K\USB to UART Driver\CH340_CH341：



以 STC 官网提供的 CH341SER 串口驱动为例，双击“CH341SER.exe”安装包，在弹出的主界面点击“安装”按钮开始安装驱动程序：

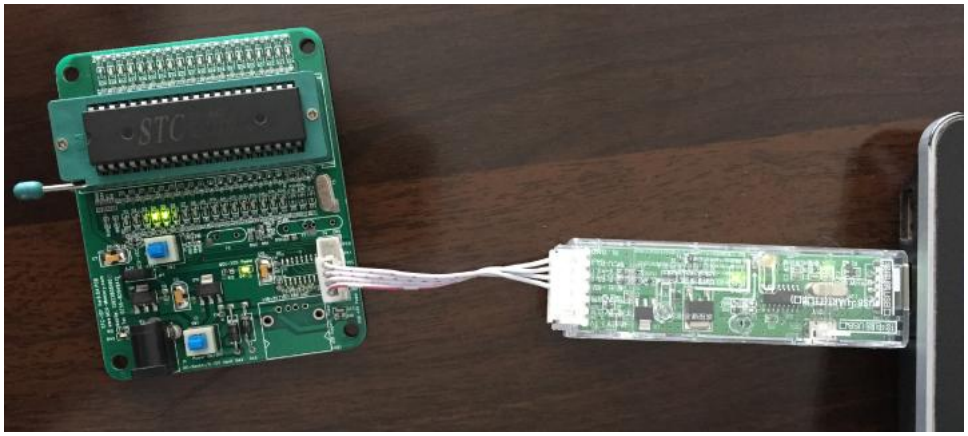


然后弹出驱动安装成功对话框，点击“确定”按钮完成安装：



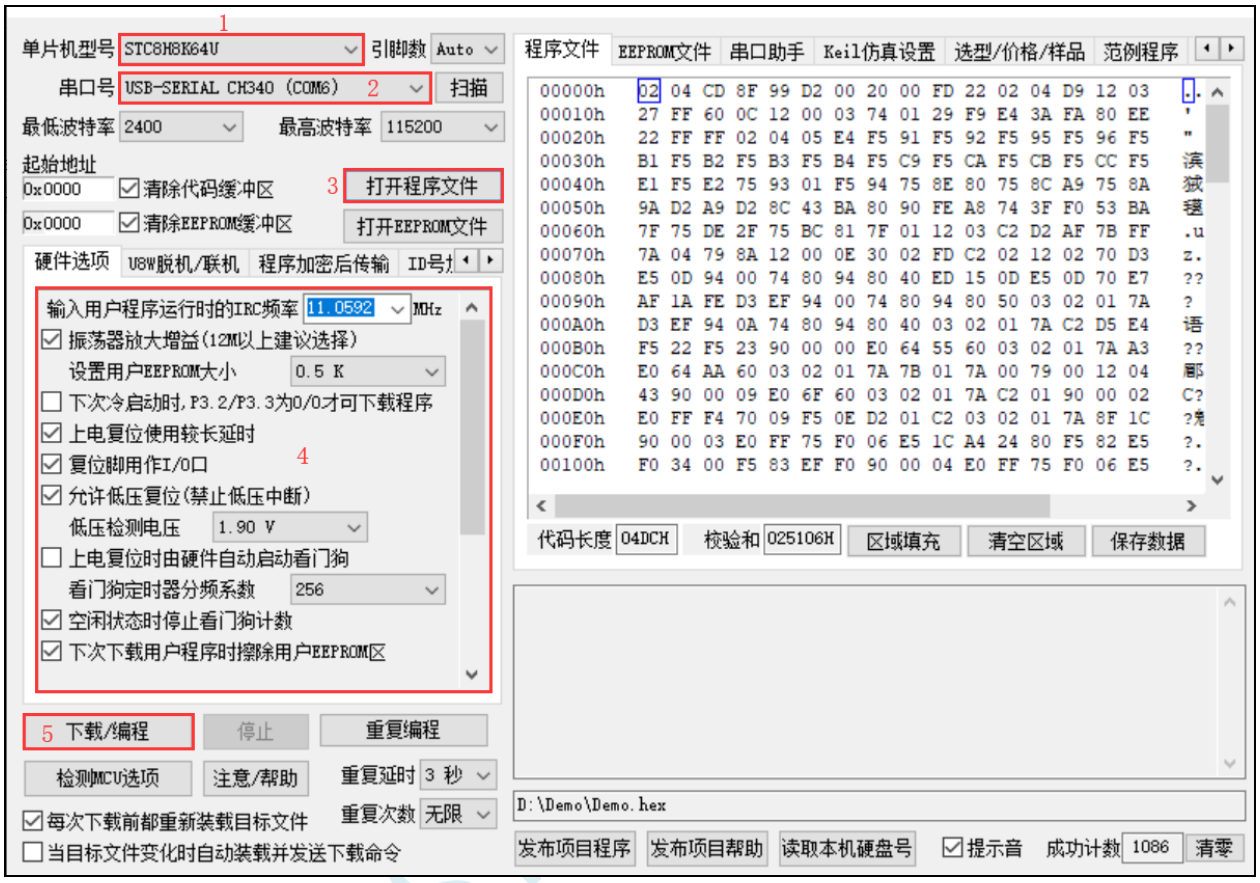
M.4.4 使用 STC 通用 USB 转串口工具下载程序到 MCU

1. 使用 STC 通用 USB 转串口工具将待烧录 MCU 与电脑进行连接：

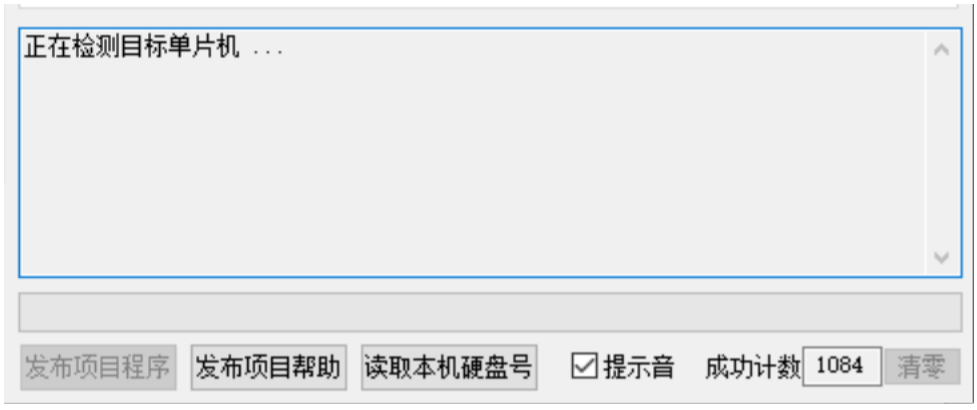


2. 打开 AIapp-ISP 软件；
3. 选择烧录芯片对应的型号；

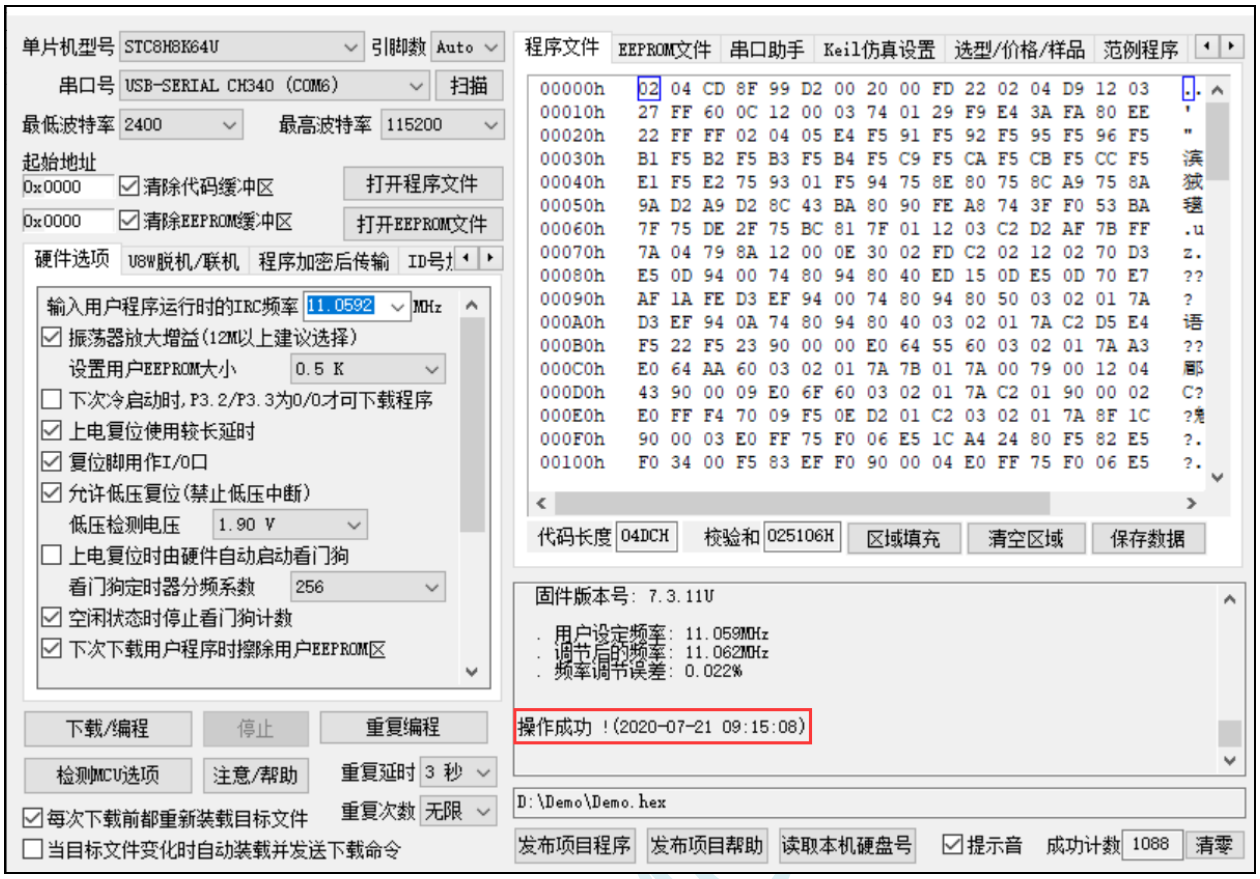
4. 选择 STC 通用 USB 转串口工具所识别的串口号（当 STC 通用 USB 转串口工具与电脑正确连接后，软件会自动扫描并识别名称为“USB-SERIAL CH340 (COMx)”串口，具体的 COM 编号会因电脑不同而不同）。当有多个 USB 转串口线与电脑相连时，则必须手动选择；
5. 加载烧录程序；
6. 设置烧录选项；
7. 点击“下载/编程”按钮；



8. 右下角提示框显示“正在检测目标单片机 ...”时按一下 STC 通用 USB 转串口工具上的“电源开关”给 MCU 供电，即可开始下载【冷启动烧录】；



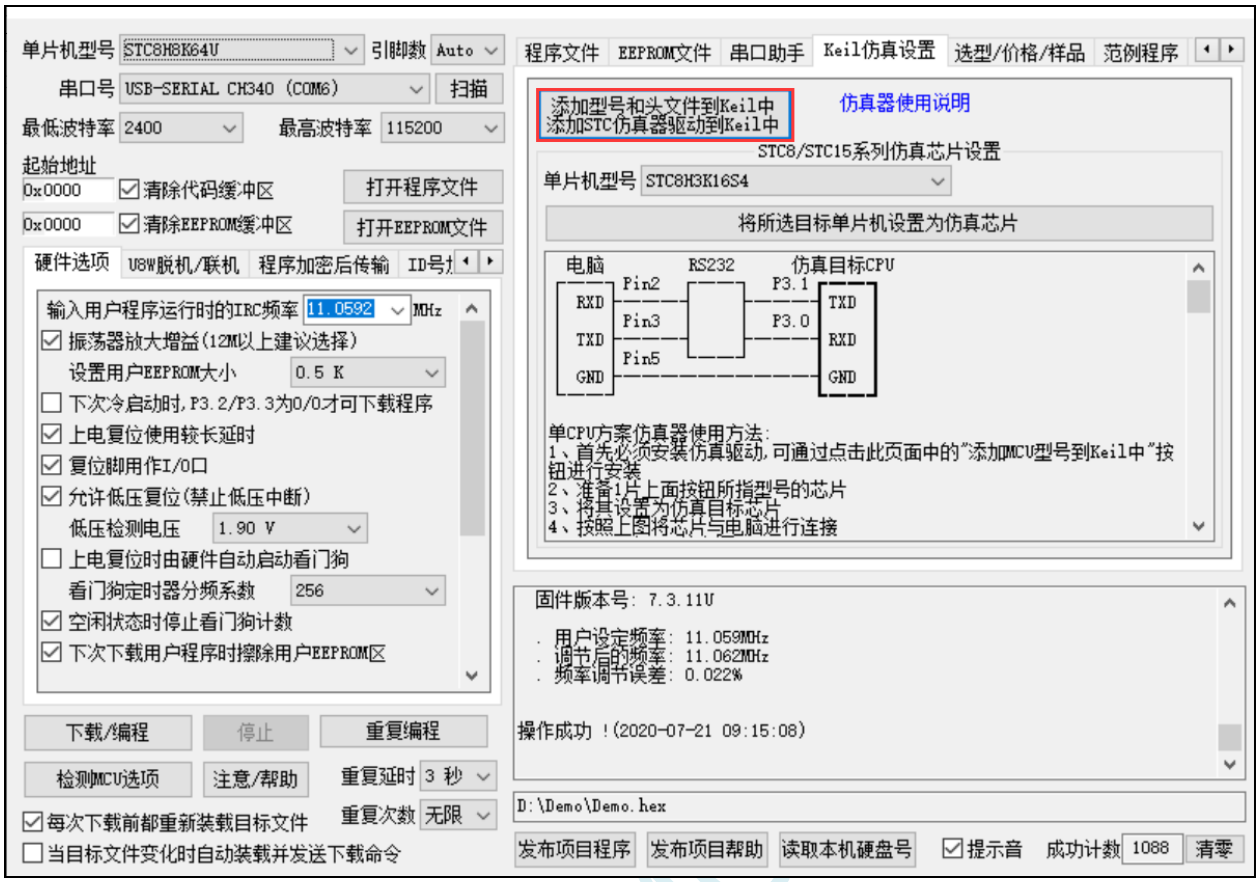
9. 等待下载结束，若下载成功，右下角提示框会显示“操作成功!”。



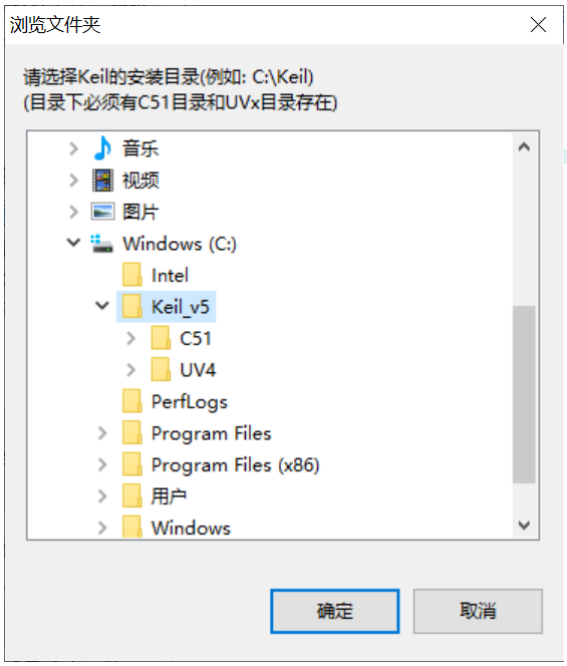
M.4.5 使用 STC 通用 USB 转串口工具仿真用户代码

目前 STC 的仿真都是基于 Keil 环境的，所以若需要使用 STC 通用 USB 转串口工具仿真用户代码，则必须要安装 Keil 软件。

Keil 软件安装完成后，还需要安装 STC 的仿真驱动。STC 的仿真驱动的安装步骤如下：
首先开 AIapp-ISP 下载软件；
然后在软件右边功能区的“Keil 仿真设置”页面中点击“添加型号和头文件到 Keil 中 添加 STC 仿真器驱动到 Keil 中”按钮：



按下后会出现如下画面：

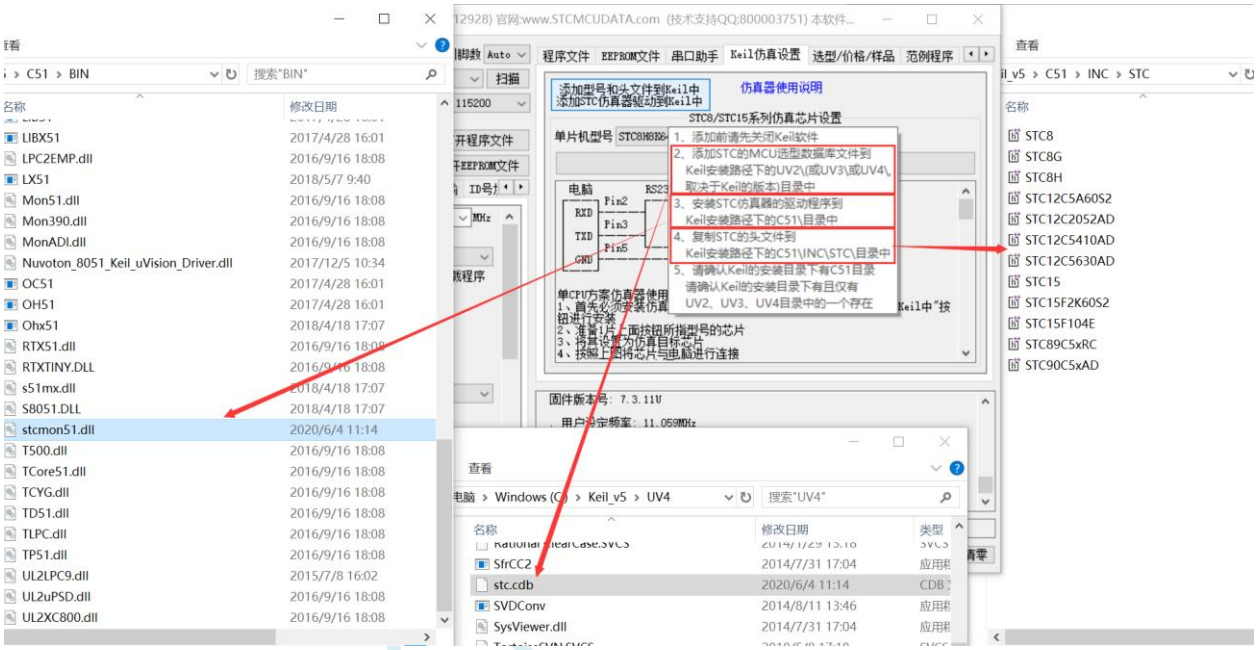


将目录定位到 Keil 软件的安装目录，然后确定。

安装成功后会弹出如下的提示框：

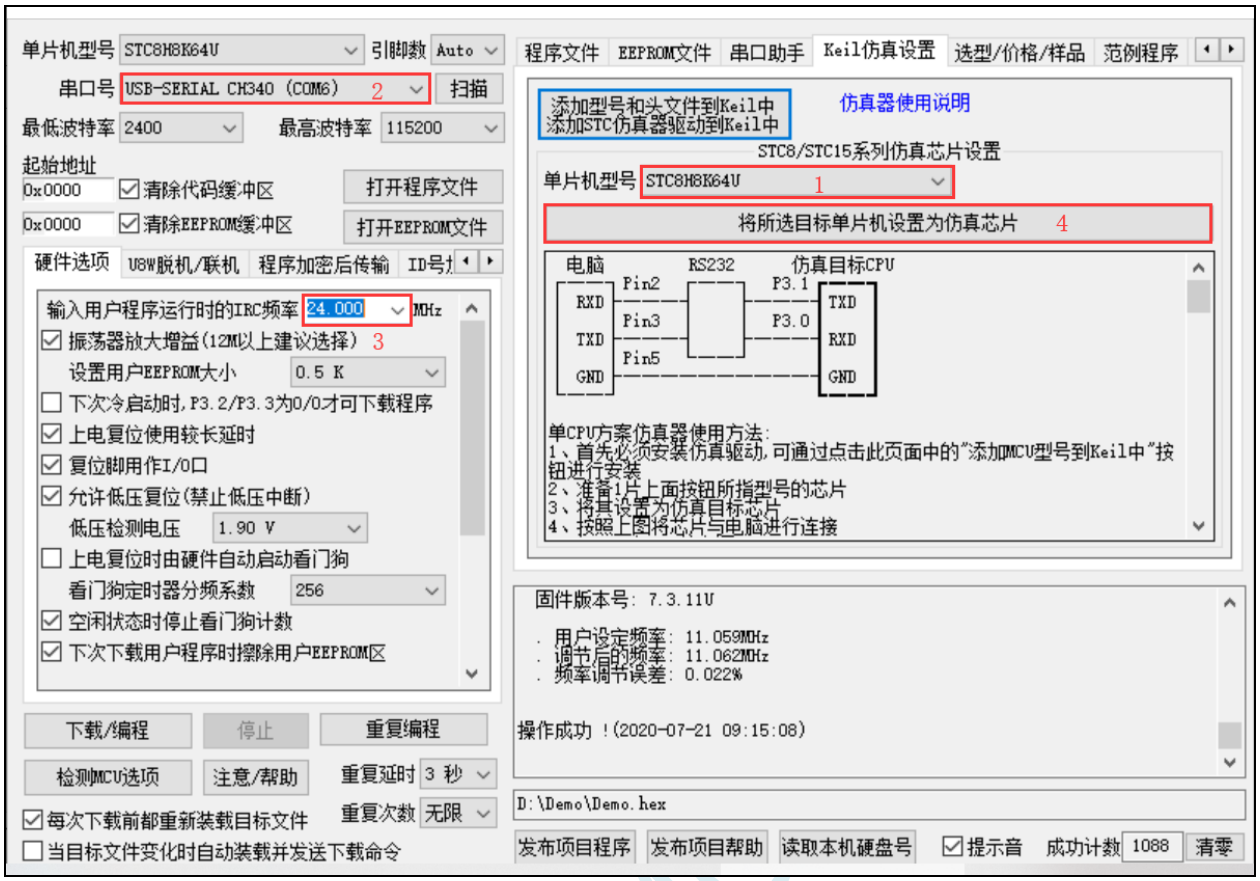


在 Keil 的相关目录中可以看到如下的文件，即表示驱动正确安装了。



由于在默认状态下，STC 的主控芯片并不是一颗仿真芯片，不具有仿真功能，所以若需要进行仿真，则还需要将 STC 的主控芯片设置为仿真芯片。
制作仿真芯片步骤如下：

首先使用 STC 通用 USB 转串口工具将 MCU 与电脑进行连接；
然后打开 STC 的 ISP 下载软件，并在串口号的下拉列表中选择串口工具所对应的串口号；
选择 MCU 单片机型号；
选择用户程序运行的 IRC 频率，制作仿真芯片时选择的频率与所仿真的用户程序所设置的频率一致，才能达到真实的运行效果。

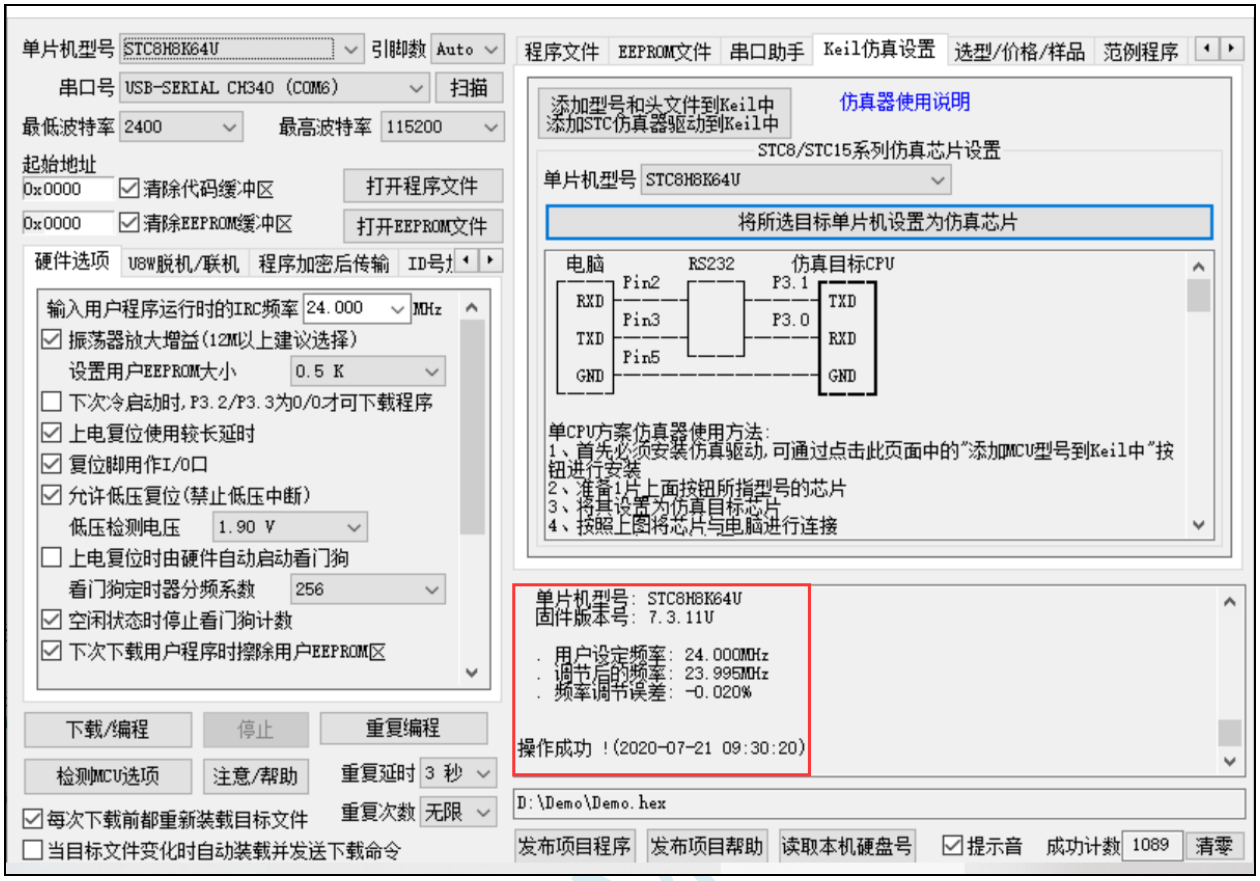


然后在软件右边功能区的“Keil 仿真设置”页面中点击“将所选目标单片机设置为仿真芯片”按钮，按下后会出现如下画面：



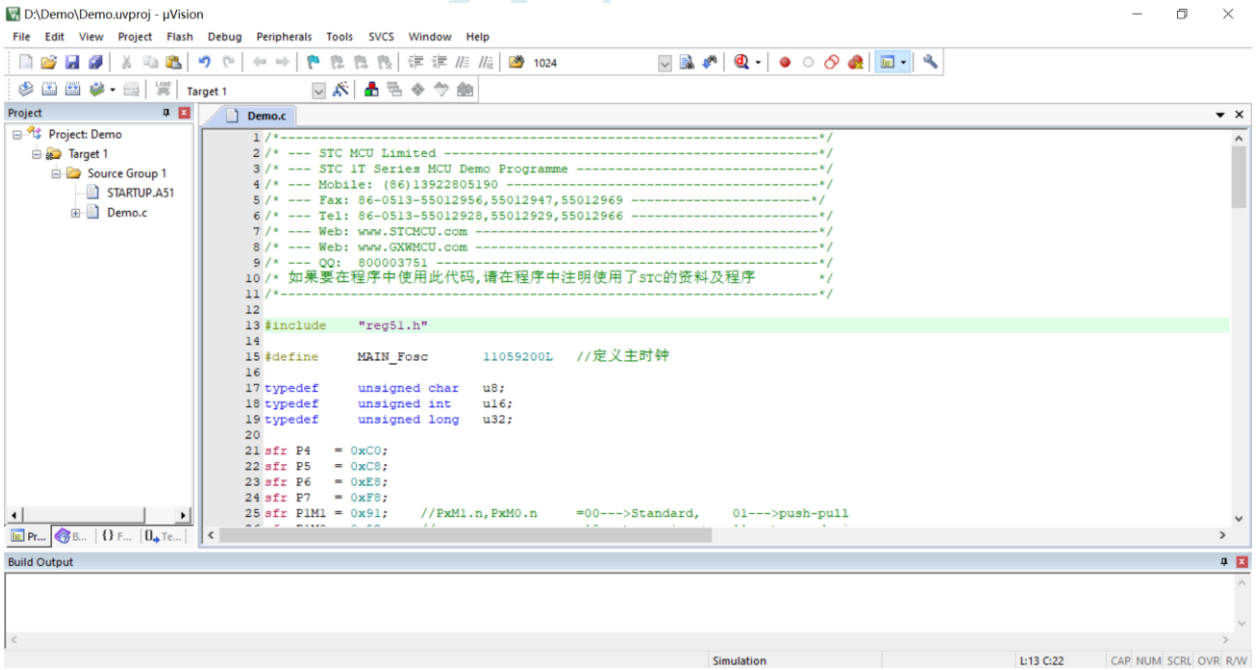
接下来需要按一下 STC 通用 USB 转串口工具上的“**电源开关**”给 MCU 供电【冷启动】，即可开始制作仿真芯片。

若设置成功，会出现如下的画面：



到此，仿真芯片便制作成功了。

接下来我们打开一个项目进行仿真：

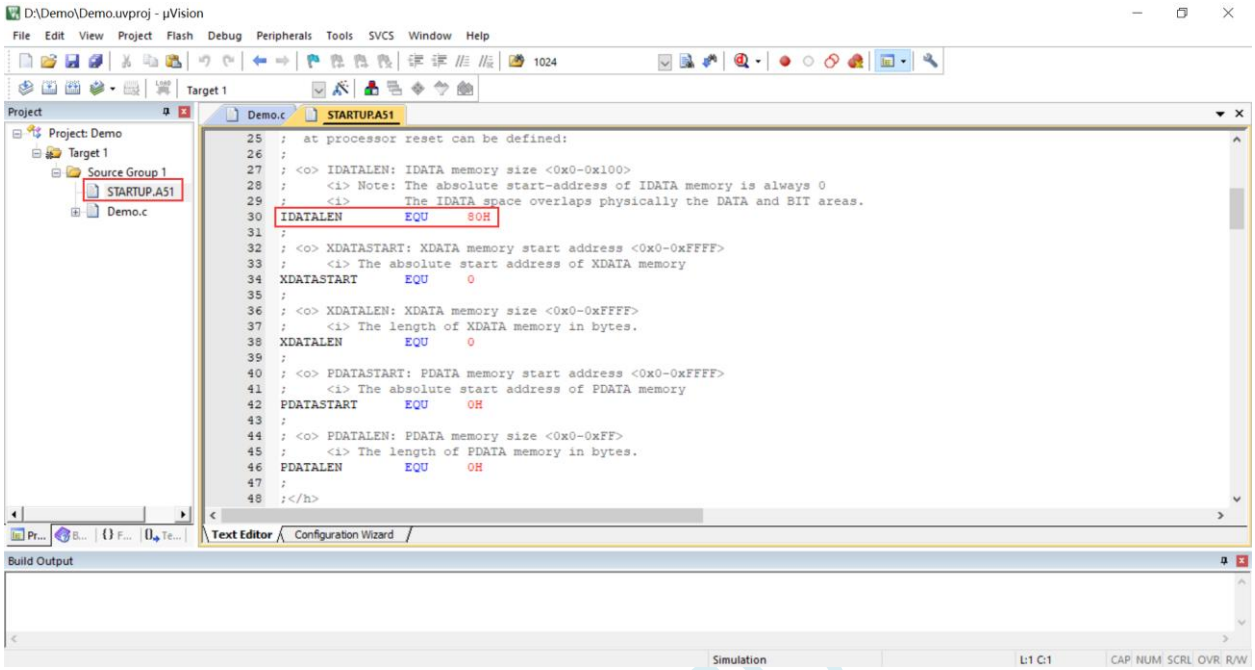


然后进行下面的项目设置：

附加说明一点：

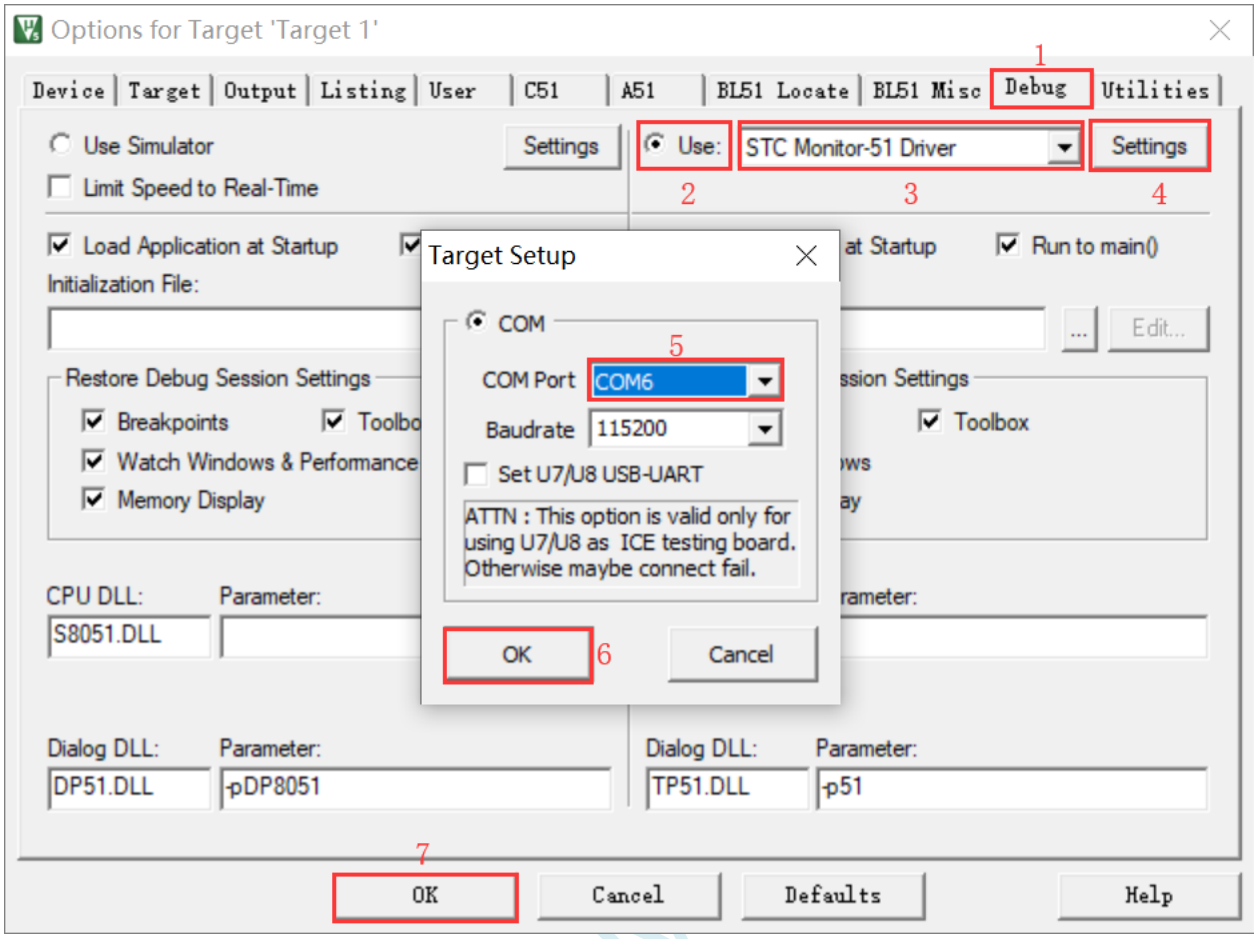
当创建的是 C 语言项目，且有将启动文件“STARTUP.A51”添加到项目中时，里面有一个命名为“IDATALEN”的宏定义，它是用来定义 IDATA 大小的一个宏，默认值是 128，即十六进制的 80H，同

时它也是启动文件中需要初始化为 0 的 IDATA 的大小。所以当 IDATA 定义为 80H，那么 STARTUP.A51 里面的代码则会将 IDATA 的 00-7F 的 RAM 初始化为 0；同样若将 IDATA 定义为 0FFH，则会将 IDATA 的 00-FF 的 RAM 初始化为 0。



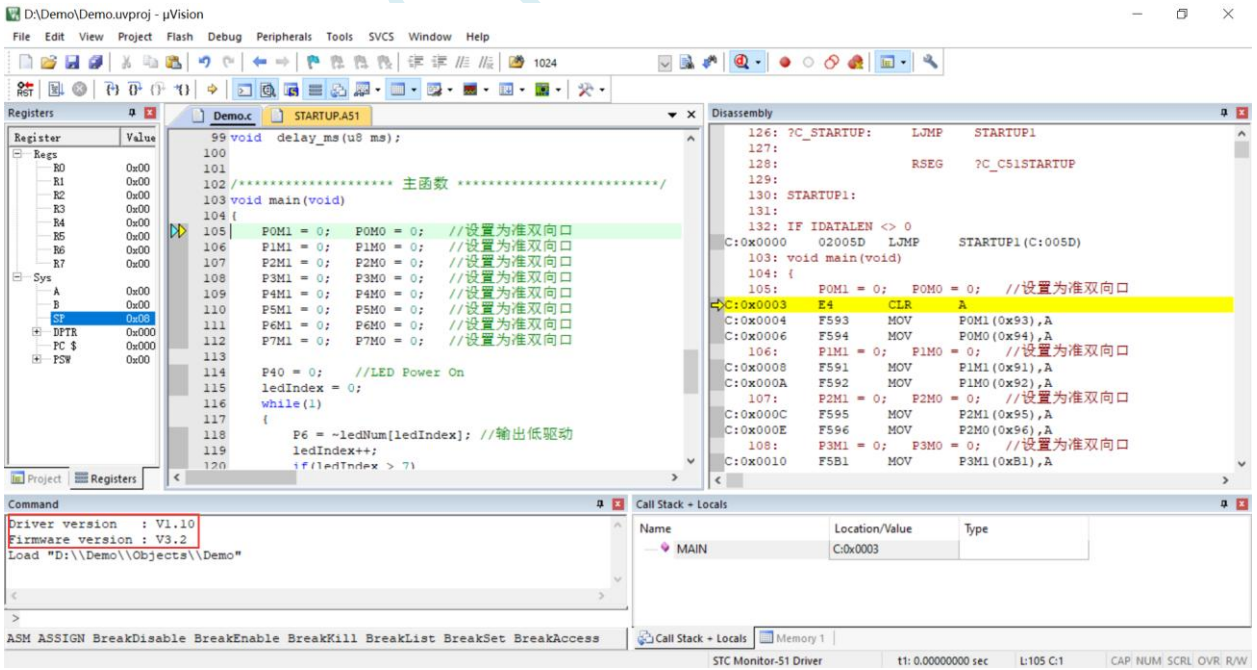
我们所选的 STC8H 系列的单片机的 IDATA 大小为 256 字节（00-7F 的 DATA 和 80H-FFH 的 IDATA），但由于在 RAM 的最后 17 个字节有写入 ID 号以及相关的测试参数，若用户在程序中需要使用这一部分数据，则一定不要将 IDATALEN 定义为 256。

按下快捷键“Alt+F7”或者选择菜单“Project”中的“Option for Target ‘Target1’”
在“Option for Target ‘Target1’”对话框中对项目进行配置：
第 1 步、进入到项目的设置页面，选择“Debug”设置页；
第 2 步、选择右侧的硬件仿真“Use …”；
第 3 步、在仿真驱动下拉列表中选择“STC Monitor-51 Driver”项；
第 4 步、点击“Settings”按钮，进入串口的设置画面；
第 5 步、对串口的端口号和波特率进行设置，串口号要选择 STC 通用 USB 转串口工具所对应的串口，波特率一般选择 115200 或者 57600。
确定完成仿真设置。
详细步骤如下图所示：

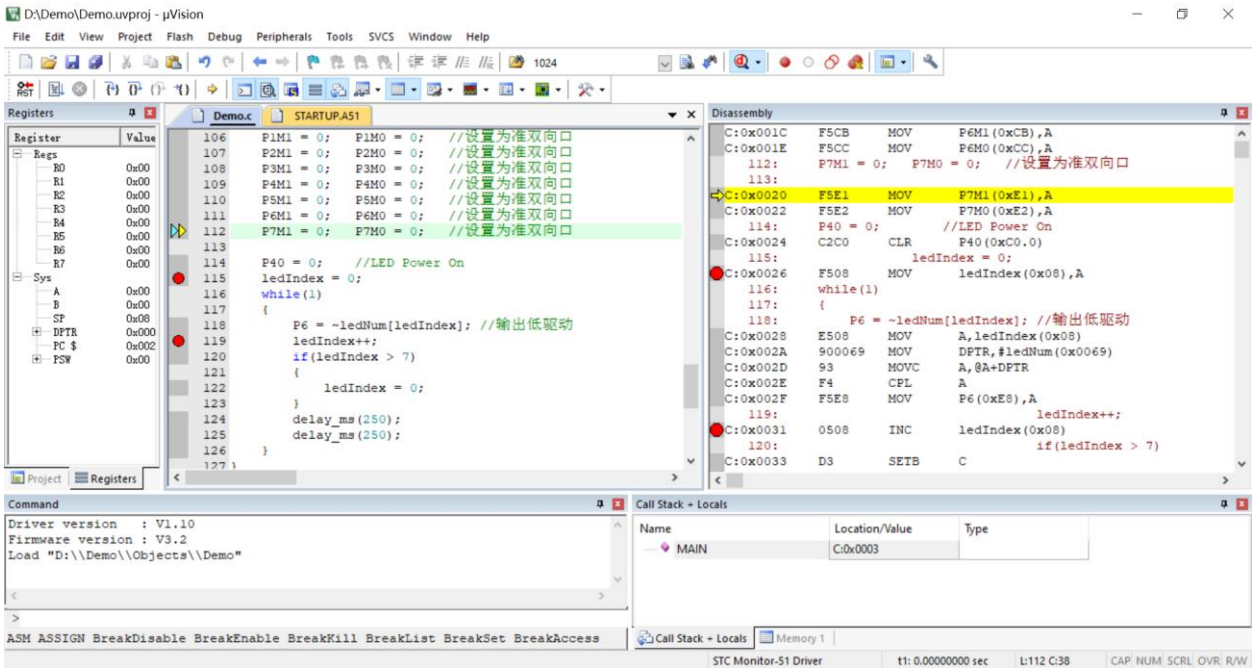


完成了上面所有的工作后，即可在 Keil 软件中按“Ctrl+F5”开始仿真调试。

若硬件连接无误的话，将会进入到类似于下面的调试界面，并在命令输出窗口显示当前的仿真驱动版本号和当前仿真监控代码固件的版本号，如下图所示：



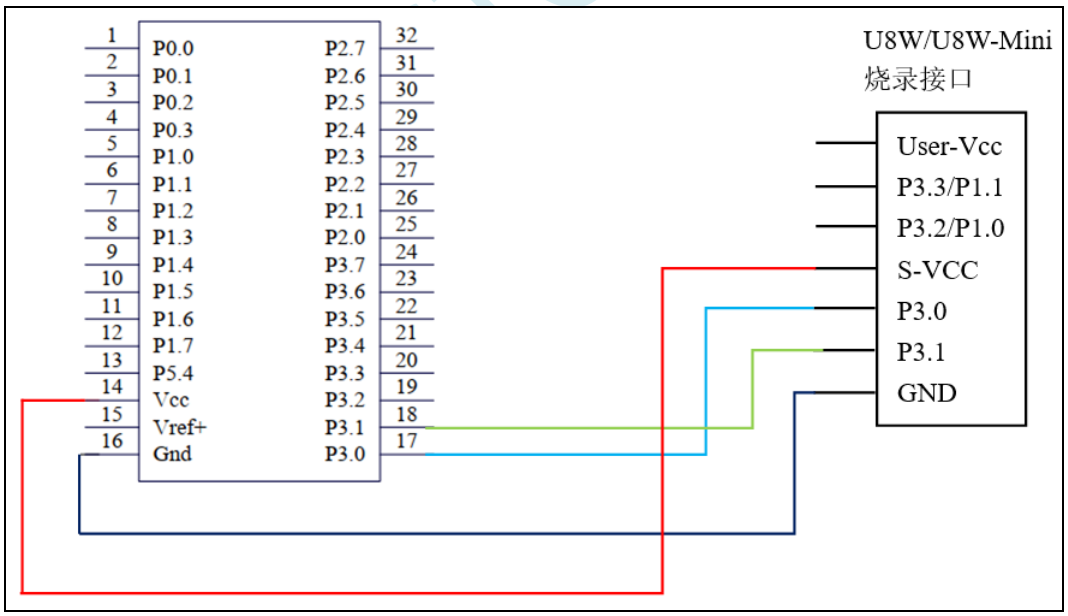
仿真调试过程中，可执行复位、全速运行、单步运行、设置断点等多中操作。



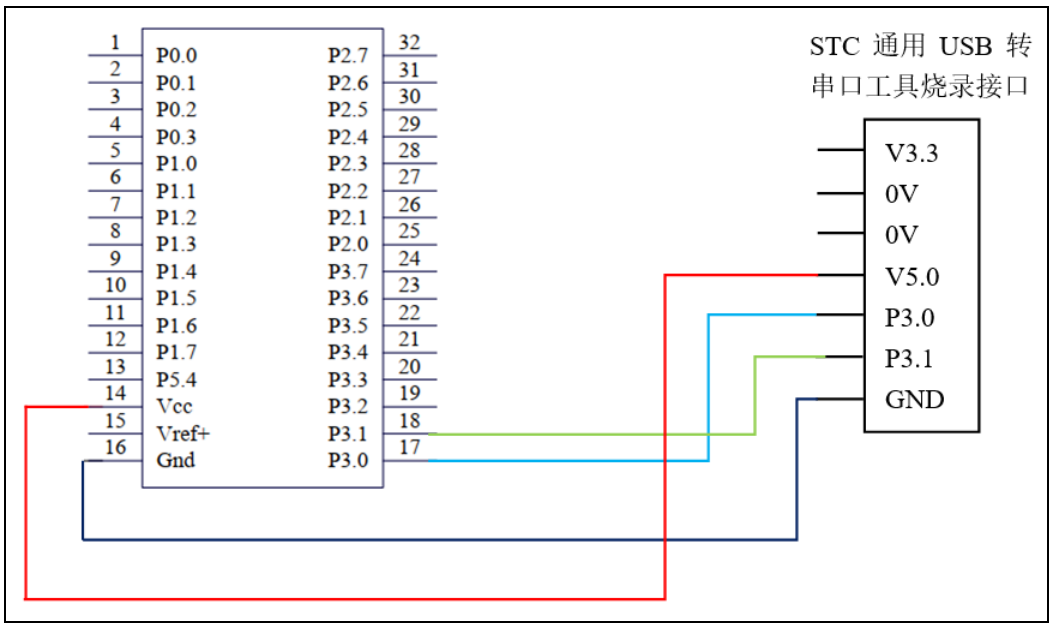
如上图所示，可在程序中设置多个断点，断点设置的个数目前最大允许 20 个（理论上可设置任意个，但是断点设置得过多会影响调试的速度）。

M.5　应用线路图

M.5.1 U8W 工具应用参考线路图

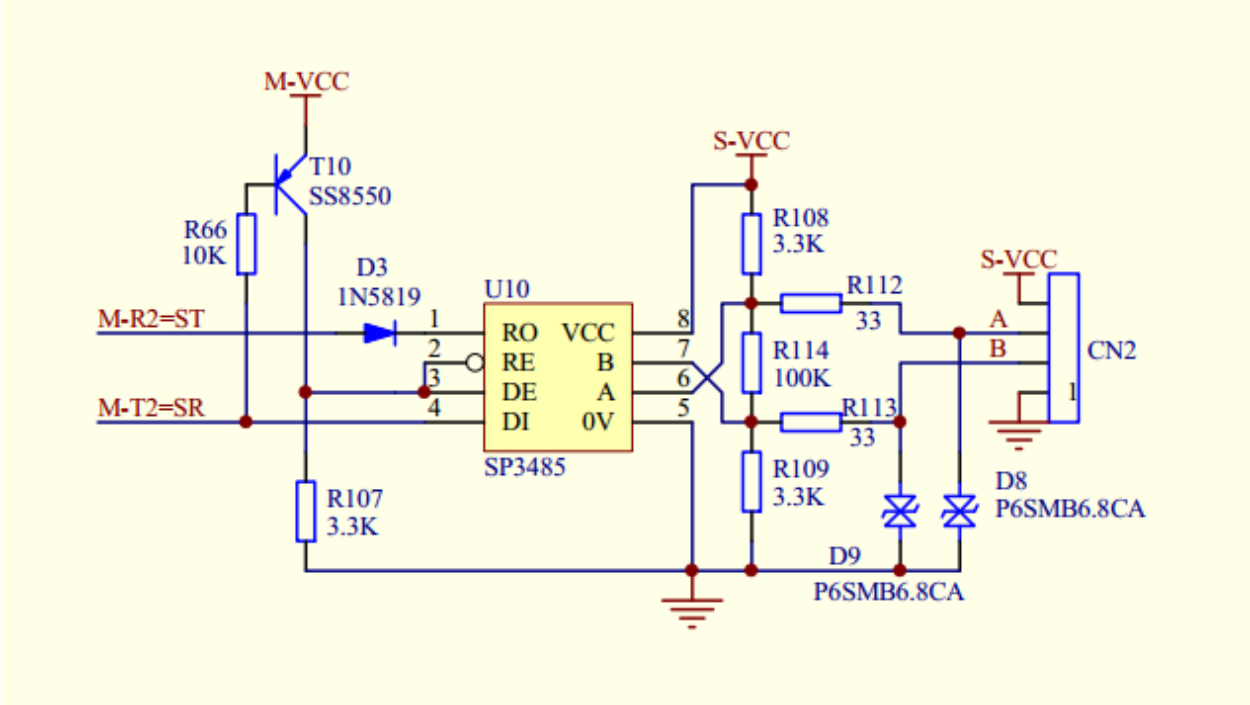


M.5.2 STC 通用 USB 转串口工具应用参考线路图


1 P0.0
2 P0.1
3 P0.2
4 P0.3
5 P1.0
6 P1.1
7 P1.2
8 P1.3
9 P1.4
10 P1.5
11 P1.6
12 P1.7
13 P5.4
14 Vcc
15 Vref+
16 Gnd
32 P2.7
31 P2.6
30 P2.5
29 P2.4
28 P2.3
27 P2.2
26 P2.1
25 P2.0
24 P3.7
23 P3.6
22 P3.5
21 P3.4
20 P3.3
19 P3.2
18 P3.1
17 P3.0
STC 通用 USB 转
串口工具烧录接口
V3.3
0V
0V
V5.0
P3.0
P3.1
GND

STC MCU

附录N U8W 下载工具中 RS485 部分线路图



BOM 清单：

标号	型号	封装	备注
U10	SP3485EN	SOP8	RS485 芯片
R66	10K	0603	电阻
R107	3.3K	0603	电阻
R108	3.3K	0603	电阻
R109	3.3K	0603	电阻
R112	33R	0603	电阻
R113	33R	0603	电阻
R114	100K	0603	电阻
T10	SS8550	SOT-23	PNP 三极管
D3	1N5819	0603	肖特基二极管
D8	P6SMB6.8CA	DO-214AA	TVS 二极管
D9	P6SMB6.8CA	DO-214AA	TVS 二极管
CN2		SIP4	通信接口

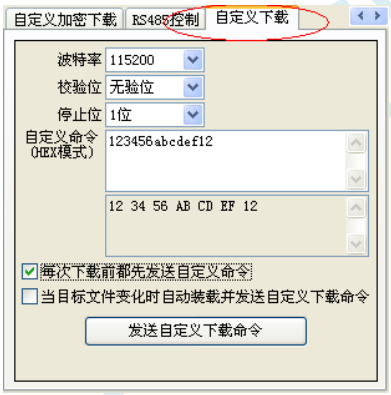
附录O　运行用户程序时收到用户命令后自动启动 ISP 下载(不停电)

“用户自定义下载”与“用户自定义加密下载”是两种完全不同功能。相对用户自定义加密下载的功能而言，用户自定义下载的功能要简单一些。

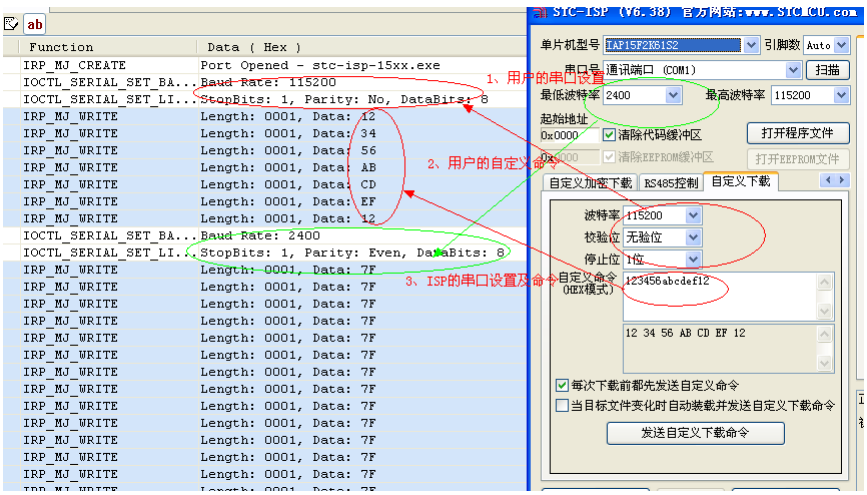
具体的功能为：电脑或脱机下载板在开始发送真正的 ISP 下载编程握手命令前，先发送用户自定义的一串命令（关于这一串串口命令，用户可以根据自己在应用程序中的串口设置来设置波特率、校验位以及停止位），然后再立即发送 ISP 下载编程握手命令。

“用户自定义下载”这一功能主要是在项目的早期开发阶段，实现不断电（不用给目标芯片重新上电）即可下载用户代码。具体的实现方法是：用户需要在自己的程序中加入一段检测自定义命令的代码，当检测到后，执行一句“MOV IAP_CONTR,#60H”的汇编代码或者“IAP_CONTR = 0x60;”的 C 语言代码，MCU 就会自动复位到 ISP 区域执行 ISP 代码。

如下图所示，将自定义命令设置为波特率为 115200、无校验位、一位停止位的命令序列：0x12、0x34、0x56、0xAB、0xCD、0xEF、0x12。当勾选上“每次下载前都先发送自定义命令”的选项后，即可实现自定义下载功能



点击“发送自定义下载命令”或者点击界面左下角的“下载/编程”按钮，应用程序便会发送如下所示的串口数据



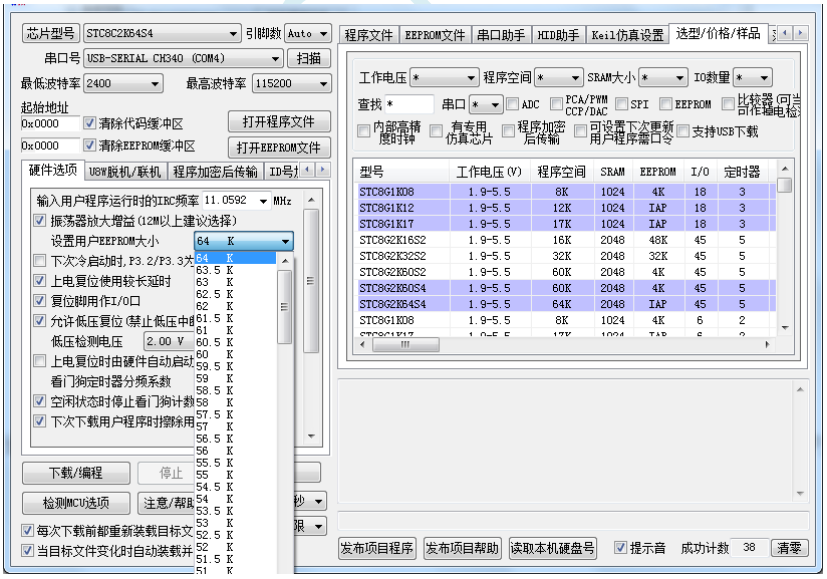
附录P　使用 STC 的 IAP 系列单片机开发自己的 ISP 程序

随着 IAP（In-Application-Programming）技术在单片机领域的不断发展，给应用系统程序代码升级带来了极大的方便。STC 的串口 ISP（In-System-Programming）程序就是使用 IAP 功能来对用户的程序进行在线升级的，但是出于对用户代码的安全着想，底层代码和上层应用程序都没有开源，为此 STC 推出了 IAP 系列单片机，即整颗 MCU 的 Flash 空间，用户均可在自己的程序中进行改写，从而使得有用户需要开发自己的 ISP 程序的想法得以实现。

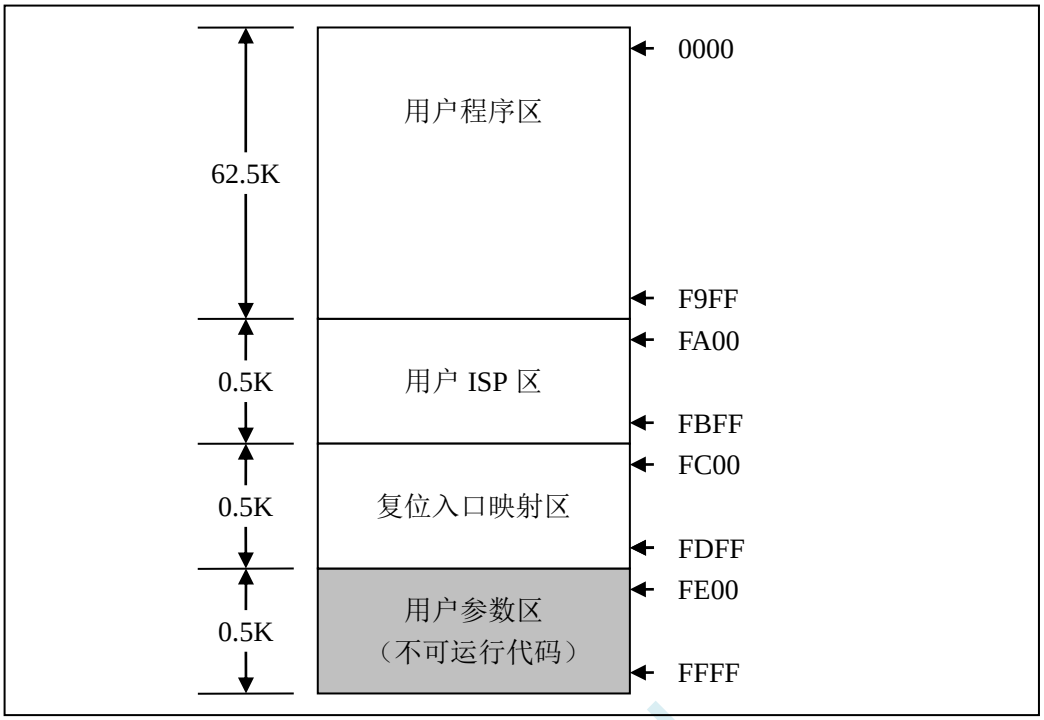
STC8C 系列单片机中的所有可以在 ISP 下载时用户自定义 EEPROM 大小的型号均为 IAP 系列单片机。目前 STC8H 系列有如下型号的单片机为 IAP 系列：STC8C2K64S4。本文以 STC8C2K64S4 为例，详细说明使用 STC 的 IAP 单片机开发用户自己的 ISP 程序的方法，并给出了基于 Keil 环境的汇编和 C 源码。

第一步：内部 FLASH 规划

由于 STC8C 系列的 IAP 型号单片机的 EEPROM 是在 ISP 下载时用户自己设置的，所以若用户需要实现自己的 ISP，则在下载用户自己的 ISP 程序时，需要按照下图是方式，将全部的 64K 都设置为 EEPROM，让用户程序空间和 EEPROM 空间完全重合，这样才能实现用户对自己程序空间进行修改和更新。

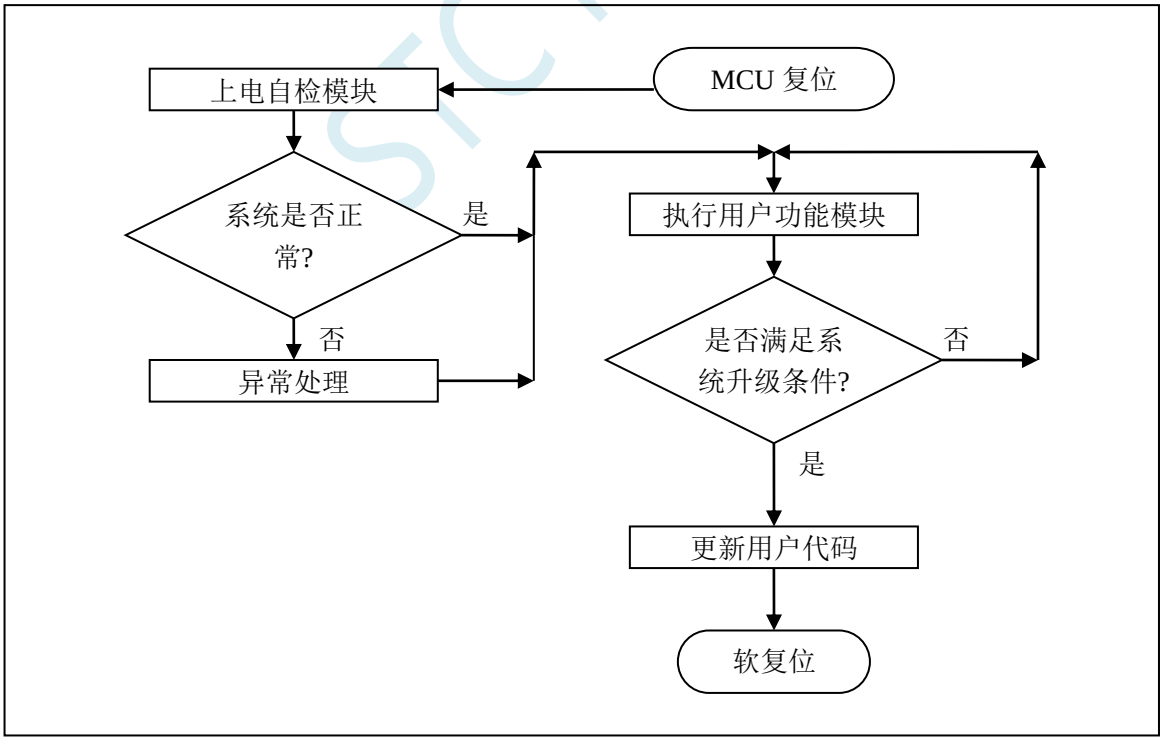


下面假设用户已将整个的 64K 的程序空间已全部设置为 EEPROM，现将整个 64K 程序空间作如下划分：



FLASH 空间中，从地址 0000H 开始的连续 62.5K 字节的空间为用户程序区。当满足特定的下载条件时，需要用户将 PC 跳转到用户 ISP 程序区，此时可对用户程序区进行擦除和改写，以达到更新用户程序的目的。

第二步、程序的基本框架



第三步、下位机固件程序说明

下位机固件程序包括两部分：ISP（ISP 代码）和 AP（用户代码）

ISP 代码（汇编代码）

```
;测试工作频率为 11.0592MHz

UARTBAUD    EQU     0FFE8H          ;定义串口波特率      (65536-11059200/4/115200)

AUXR        DATA    08EH            ;附加功能控制寄存器
WDT_CONTR   DATA    0C1H            ;看门狗控制寄存器
IAP_DATA    DATA    0C2H            ;IAP 数据寄存器
IAP_ADDRH   DATA    0C3H            ;IAP 高地址寄存器
IAP_ADDRL   DATA    0C4H            ;IAP 低地址寄存器
IAP_CMD     DATA    0C5H            ;IAP 命令寄存器
IAP_TRIG    DATA    0C6H            ;IAP 命令触发寄存器
IAP_CONTR   DATA    0C7H            ;IAP 控制寄存器
IAP_TPS     DATA    0F5H            ;IAP 等待时间控制寄存器

ISPCODE     EQU     0FA00H          ;ISP 模块入口地址(1 页),同时也是外部接口地址
APENTRY     EQU     0FC00H          ;应用程序入口地址数据(1 页)

            ORG     0000H

            LJMP    ISP_ENTRY       ;系统复位入口

RESET:
            MOV     SCON,#50H       ;设置串口模式(8 位数据位,无校验位)
            MOV     AUXR,#40H       ;定时器 1 为 1T 模式
            MOV     TMOD,#00H       ;定时器 1 工作于模式 0(16 位重装载)
            MOV     TH1,#HIGH UARTBAUD  ;设置重载值
            MOV     TL1,#LOW UARTBAUD
            SETB    TR1             ;启动定时器 1
NEXT1:
            MOV     R0,#16
NEXT2:
            JNB     RI,$            ;等待串口数据
            CLR     RI
            MOV     A,SBUF
            CJNE    A,#7FH,NEXT1    ;判断是否为 7F
            DJNZ    R0,NEXT2
            LJMP    ISP_DOWNLOAD    ;跳转到下载界面

            ORG     ISPCODE

ISP_DOWNLOAD:
            CLR     A
            MOV     PSW,A           ;ISP 模块使用第 0 组寄存器
            MOV     IE,A            ;关闭所有中断
            CLR     RI              ;清除串口接收标志
            SETB    TI              ;置串口发送标志
            CLR     TR0
            MOV     SP,#5FH         ;设置堆栈指针

            MOV     A,#5AH          ;返回 5A  55 到 PC,表示 ISP 擦除模块已准备就绪
            LCALL   ISP_SENDUART
            MOV     A,#055H
            LCALL   ISP_SENDUART
            LCALL   ISP_RECVACK     ;接收应答数据

            MOV     IAP_ADDRL,#0    ;首先在第 2 页起始地址写 "LJMP ISP_ENTRY" 指令
            MOV     IAP_ADDRH,#02H
            LCALL   ISP_ERASEIAP
```

```
                MOV     A,#02H
                LCALL   ISP_PROGRAMIAP          ;编程用户代码复位向量代码
                MOV     A,#HIGH ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ;编程用户代码复位向量代码
                MOV     A,#LOW ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ;编程用户代码复位向量代码

                MOV     IAP_ADDRL,#0            ;用户代码地址从 0 开始
                MOV     IAP_ADDRH,#0
                LCALL   ISP_ERASEIAP
                MOV     A,#02H
                LCALL   ISP_PROGRAMIAP          ;编程用户代码复位向量代码
                MOV     A,#HIGH ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ;编程用户代码复位向量代码
                MOV     A,#LOW ISP_ENTRY
                LCALL   ISP_PROGRAMIAP          ;编程用户代码复位向量代码

                MOV     IAP_ADDRL,#0            ;新代码缓冲区地址
                MOV     IAP_ADDRH,#02H
                MOV     R7,#124                 ;擦除 62.5K 字节
ISP_ERASEAP:
                LCALL   ISP_ERASEIAP
                INC     IAP_ADDRH               ;目标地址+512
                INC     IAP_ADDRH
                DJNZ    R7,ISP_ERASEAP          ;判断是否擦除完成

                MOV     IAP_ADDRL,#LOW APENTRY
                MOV     IAP_ADDRH,#HIGH APENTRY
                LCALL   ISP_ERASEIAP

                MOV     A,#5AH                  ;返回 5A  A5 到 PC,表示 ISP 编程模块已准备就绪
                LCALL   ISP_SENDUART
                MOV     A,#0A5H
                LCALL   ISP_SENDUART
                LCALL   ISP_RECVACK             ;接收应答数据

                LCALL   ISP_RECVUART            ;接收长度高字节
                MOV     R0,A
                LCALL   ISP_RECVUART            ;接收长度低字节
                MOV     R1,A
                CLR     C                       ;将总长度-3
                MOV     A,#03H
                SUBB    A,R1
                MOV     DPL,A
                CLR     A
                SUBB    A,R0
                MOV     DPH,A                   ;总长度补码存入 DPTR

                LCALL   ISP_RECVUART            ;映射用户代码复位入口代码到映射区
                LCALL   ISP_PROGRAMIAP          ;0000
                LCALL   ISP_RECVUART
                LCALL   ISP_PROGRAMIAP          ;0001
                LCALL   ISP_RECVUART
                LCALL   ISP_PROGRAMIAP          ;0002

                MOV     IAP_ADDRL,#03H          ;用户代码起始地址
                MOV     IAP_ADDRH,#00H
ISP_PROGRAMNEXT:
                LCALL   ISP_RECVUART            ;接收代码数据
```

```
                LCALL   ISP_PROGRAMIAP          ;编程到用户代码区
                INC     DPTR
                MOV     A,DPL
                ORL     A,DPH
                JNZ     ISP_PROGRAMNEXT         ;长度检测

ISP_SOFTRESET:
                MOV     IAP_CONTR,#20H          ;软件复位系统
                SJMP    $


ISP_ENTRY:
                MOV     WDT_CONTR,#17H          ;清看门狗
                MOV     IAP_CONTR,#80H          ;使能 IAP 功能
                MOV     IAP_TPS,#11             ;设置 IAP 等待时间参数
                MOV     IAP_ADDRL,#LOW ISP_DOWNLOAD
                MOV     IAP_ADDRH,#HIGH ISP_DOWNLOAD
                MOV     IAP_DATA,#00H           ;测试数据 1
                MOV     IAP_CMD,#1              ;读命令
                MOV     IAP_TRIG,#5AH           ;触发 ISP 命令
                MOV     IAP_TRIG,#0A5H
                MOV     A,IAP_DATA
                CJNE    A,#0E4H,ISP_ENTRY       ;若无法读出数据则需要等待电压稳定
                INC     IAP_ADDRL               ;测试地址 FC01H
                MOV     IAP_DATA,#45H           ;测试数据 2
                MOV     IAP_CMD,#1              ;读命令
                MOV     IAP_TRIG,#5AH           ;触发 ISP 命令
                MOV     IAP_TRIG,#0A5H
                MOV     A,IAP_DATA
                CJNE    A,#0F5H,ISP_ENTRY       ;若无法读出数据则需要等待电压稳定

                MOV     SCON,#50H               ;设置串口模式(8 位数据位,无校验位)
                MOV     AUXR,#40H               ;定时器 1 为 1T 模式
                MOV     TMOD,#00H               ;定时器 1 工作于模式 0(16 位重装载)
                MOV     TH1,#HIGH UARTBAUD      ;设置重载值
                MOV     TL1,#LOW UARTBAUD
                SETB    TR1                     ;启动定时器 1
                SETB    TR0

                LCALL   ISP_RECVUART            ;检测是否有串口数据
                JC      GOTOAP
                MOV     R0,#16
ISP_CHECKNEXT:
                LCALL   ISP_RECVUART            ;接收同步数据
                JC      GOTOAP
                CJNE    A,#7FH,GOTOAP           ;判断是否为 7F
                DJNZ    R0,ISP_CHECKNEXT
                MOV     A,#5AH                  ;返回 5A  69 到 PC,表示 ISP 模块已准备就绪
                LCALL   ISP_SENDUART
                MOV     A,#69H
                LCALL   ISP_SENDUART
                LCALL   ISP_RECVACK             ;接收应答数据
                LJMP    ISP_DOWNLOAD            ;跳转到下载界面
GOTOAP:
                CLR     A                       ;将 SFR 恢复为复位值
                MOV     TCON,A
                MOV     TMOD,A
                MOV     TL0,A
                MOV     TH0,A
```

```
                MOV     TL1,A
                MOV     TH1,A
                MOV     SCON,A
                MOV     AUXR,A
                LJMP    APENTRY             ;正常运行用户程序

ISP_RECVACK:
                LCALL   ISP_RECVUART
                JC      GOTOAP
                XRL     A,#7FH
                JZ      ISP_RECVACK         ;跳过同步数据
                CJNE    A,#25H,GOTOAP       ;应答数据 1 检测
                LCALL   ISP_RECVUART
                JC      GOTOAP
                CJNE    A,#69H,GOTOAP       ;应答数据 2 检测
                RET

ISP_RECVUART:
                CLR     A
                MOV     TL0,A               ;初始化超时定时器
                MOV     TH0,A
                CLR     TF0
                MOV     WDT_CONTR,#17H      ;清看门狗
ISP_RECVWAIT:
                JBC     TF0,ISP_RECVTIMEOUT ;超时检测
                JNB     RI,ISP_RECVWAIT     ;等待接收完成
                MOV     A,SBUF              ;读取串口数据
                CLR     RI                  ;清除标志
                CLR     C                   ;正确接收串口数据
                RET
ISP_RECVTIMEOUT:
                SETB    C                   ;超时退出
                RET

ISP_SENDUART:
                MOV     WDT_CONTR,#17H      ;清看门狗
                JNB     TI,ISP_SENDUART     ;等待前一个数据发送完成
                CLR     TI                  ;清除标志
                MOV     SBUF,A              ;发送当前数据
                RET

ISP_ERASEIAP:
                MOV     WDT_CONTR,#17H      ;清看门狗
                MOV     IAP_CMD,#3          ;擦除命令
                MOV     IAP_TRIG,#5AH       ;触发 ISP 命令
                MOV     IAP_TRIG,#0A5H
                NOP
                NOP
                NOP
                NOP
                RET

ISP_PROGRAMIAP:
                MOV     WDT_CONTR,#17H      ;清看门狗
                MOV     IAP_CMD,#2          ;编程命令
                MOV     IAP_DATA,A          ;将当前数据送 IAP 数据寄存器
                MOV     IAP_TRIG,#5AH       ;触发 ISP 命令
                MOV     IAP_TRIG,#0A5H
                NOP
```

```
        NOP
        NOP
        NOP
        MOV     A,IAP_ADDRL             ;IAP 地址+1
        ADD     A,#01H
        MOV     IAP_ADDRL,A
        MOV     A,IAP_ADDRH
        ADDC    A,#00H
        MOV     IAP_ADDRH,A
        RET


        ORG     APENTRY
        LJMP    RESET


        END
```

ISP 代码包括如下外部接口模块：

ISP_DOWNLOAD：程序下载入口地址，绝对地址 FA00H

ISP_ENTRY：上电系统自检程序（系统自动调用）

对于用户程序而言，用户只需要在满足下载条件时，将 PC 值跳转到 ISPPROGRAM （即 FA00H 的绝对地址），即可实现代码更新。

用户代码（C 语言代码）

```
//测试工作频率为 11.0592MHz

#include "reg51.h"

#define   FOSC              11059200L           //系统时钟频率
#define   BAUD              (65536 - FOSC/4/115200)     //定义串口波特率
#define   ISPPROGRAM        0xfa00              //ISP 下载程序入口地址

sfr       AUXR      =       0x8e;               //波特率发生器控制寄存器
sfr       P1M0      =       0x92;
sfr       P1M1      =       0x91;

void (*IspProgram)() = ISPPROGRAM;              //定义指针函数
char cnt7f;                                     //Isp_Check 内部使用的变量

void uart() interrupt 4                         //串口中断服务程序
{
    if (TI) TI = 0;                             //发送完成中断
    if (RI)                                     //接收完成中断
    {
        if (SBUF == 0x7f)
        {
            cnt7f++;
            if (cnt7f >= 16)
            {
                IspProgram();                   //调用下载模块(****重要语句****)
            }
        }
        else
```

```
        {
            cnt7f = 0;
        }
        RI = 0;                                     //清接收完成标志
    }
}

void main()
{
    SCON = 0x50;                                    //定义串口模式为 8bit 可变,无校验位
    AUXR = 0x40;
    TH1  = BAUD >> 8;
    TL1  = BAUD;
    TR1  = 1;
    ES = 1;                                         //使能串口中断
    EA = 1;                                         //打开全局中断开关

    P1M0 = 0;
    P1M1 = 0;

    while (1)
    {
        P1++;
    }
}
```

用户代码（汇编代码）

```
;测试工作频率为 11.0592MHz

UARTBAUD    EQU     0FFE8H          ;定义串口波特率      (65536-11059200/4/115200)
ISPPROGRAM  EQU     0FA00H          ;ISP 下载程序入口地址

AUXR        DATA    08EH            ;附件功能控制寄存器

CNT7F       DATA    60H             ;接收 7F 的计数器

            ORG     0000H
            LJMP    START           ;系统复位入口

            ORG     0023H
            LJMP    UART_ISR        ;串口中断入口

UART_ISR:
            PUSH    ACC
            PUSH    PSW
            JNB     TI,CHECKRI      ;检测发送中断
            CLR     TI              ;清除标志
CHECKRI:
            JNB     RI,UARTISR_EXIT ;检测接收中断
            CLR     RI              ;清除标志
            MOV     A,SBUF
            CJNE    A,#7FH,ISNOT7F
            INC     CNT7F
            MOV     A,CNT7F
            CJNE    A,#16,UARTISR_EXIT
            LJMP    ISPPROGRAM      ;调用下载模块(****重要语句****)
ISNOT7F:
            MOV     CNT7F,#0
```

```
UARTISR_EXIT:
        POP     PSW
        POP     ACC
        RETI

START:
        MOV     R0,#7FH             ;清 RAM
        CLR     A
        MOV     @R0,A
        DJNZ    R0,$-1
        MOV     SP,#7FH             ;初始化 SP

        MOV     SCON,#50H           ;设置串口模式(8 位可变,无校验位)
        MOV     AUXR,#15H           ;BRT 工作于 1T 模式,启动 BRT
        MOV     TMOD,#00H           ;定时器 1 工作于模式 0(16 位重装载)
        MOV     TH1,#HIGH UARTBAUD  ;设置重载值
        MOV     TL1,#LOW UARTBAUD
        SETB    TR1                 ;启动定时器 1
        SETB    ES                  ;使能串口中断
        SETB    EA                  ;开中断总开关

MAIN:
        INC     P0
        SJMP    MAIN

        END
```

STC MCU

用户代码可以使用 C 或者汇编语言编写，但对于汇编代码需要注意一点：位于 0000H 的复位入口地址处的指令必须是一个长跳转语句（类似 LJMP START）。在用户代码中，需要设置好串口，并在满足下载条件时，将 PC 值跳转到 ISPPROGRAM （即 FA00H 的绝对地址），以实现代码更新。对于汇编代码，我们可以使用“LJMP 0FA00H”指令进行调用，如下图

```
UARTBAUD    EQU     0FFE8H          ;定义串口波特率 (65536-11059200/4/115200)
ISPPROGRAM  EQU     0FA00H          ;ISP下载程序入口地址

AUXR        DATA    08EH            ;附件功能控制寄存器
```

```
18          CLR     TI                      ;清除标志
19  CHECKRI:
20          JNB     RI,UARTISR_EXIT         ;检测接收中断
21          CLR     RI                      ;清除标志
22          MOV     A,SBUF
23          CJNE    A,#7FH,ISNOT7F
24          INC     CNT7F
25          MOV     A,CNT7F
26          CJNE    A,#16,UARTISR_EXIT
27          LJMP    ISPPROGRAM              ;调用下载模块(****重要语句****)
28  ISNOT7F:
29          MOV     CNT7F,#0
30  UARTISR_EXIT:
31          POP     PSW
32          POP     ACC
33          RETI
34
35  START:
```

在 C 代码中，必须定义一个函数指针变量，并将此变量赋值为 0xFA00，然后再调用，如下图

```
#include "reg51.h"

#define FOSC            11059200L               //系统时钟频率
#define BAUD            (65536 - FOSC/4/115200) //定义串口波特率
#define ISPPROGRAM      0xfa00                  //ISP下载程序入口地址

sfr AUXR    =   0x8e;                           //波特率发生器控制寄存器
sfr P1M0    =   0x92;
sfr P1M1    =   0x91;

void (*IspProgram)() = ISPPROGRAM;              //定义指针函数
char cnt7f;                                     //Isp_Check内部使用的变量

void uart() interrupt 4                         //串口中断服务程序
{
    if (TI) TI = 0;                             //发送完成中断
    if (RI)                                     //接收完成中断
    {
        if (SBUF == 0x7f)
        {
            cnt7f++;
            if (cnt7f >= 16)
            {
                IspProgram();                   //调用下载模块(****重要语句****)
            }
        }
        else
        {
            cnt7f = 0;
        }
        RI = 0;                         //清接收完成标志
    }
```

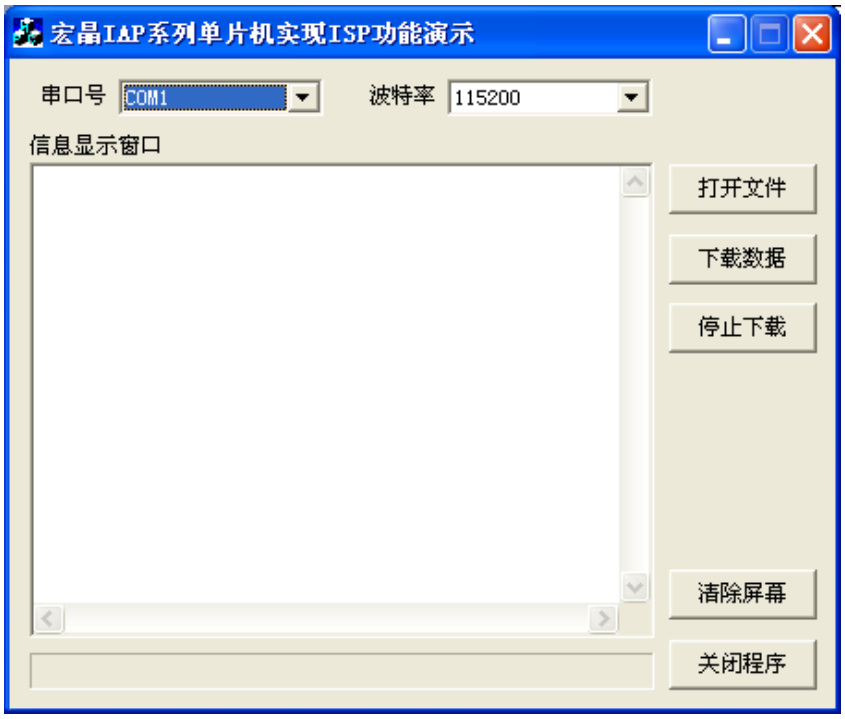
第四步、上位机应用程序说明

上位机的程序是基于 MFC 的对话框项目，对于串口的访问是直接调用 Windows 的 API 函数，而没有使用串口控件，从而省去的控件的注册以及系统版本不兼容的诸多问题。界面较简单，只是为这一功能的实现提供了一个框架，其他的功能及要求均还可以往上面添加。

上位机程序的核心模块是基于类 CISPDlg 的一个友元函数“UINT Download(LPVOID pParam);”，此函数负责与下位机通讯，发送各种通讯命令来完成对用户程序的更新。用户可以根据各自不同的需求增加命令。

第五步、上位机应用程序的使用方法

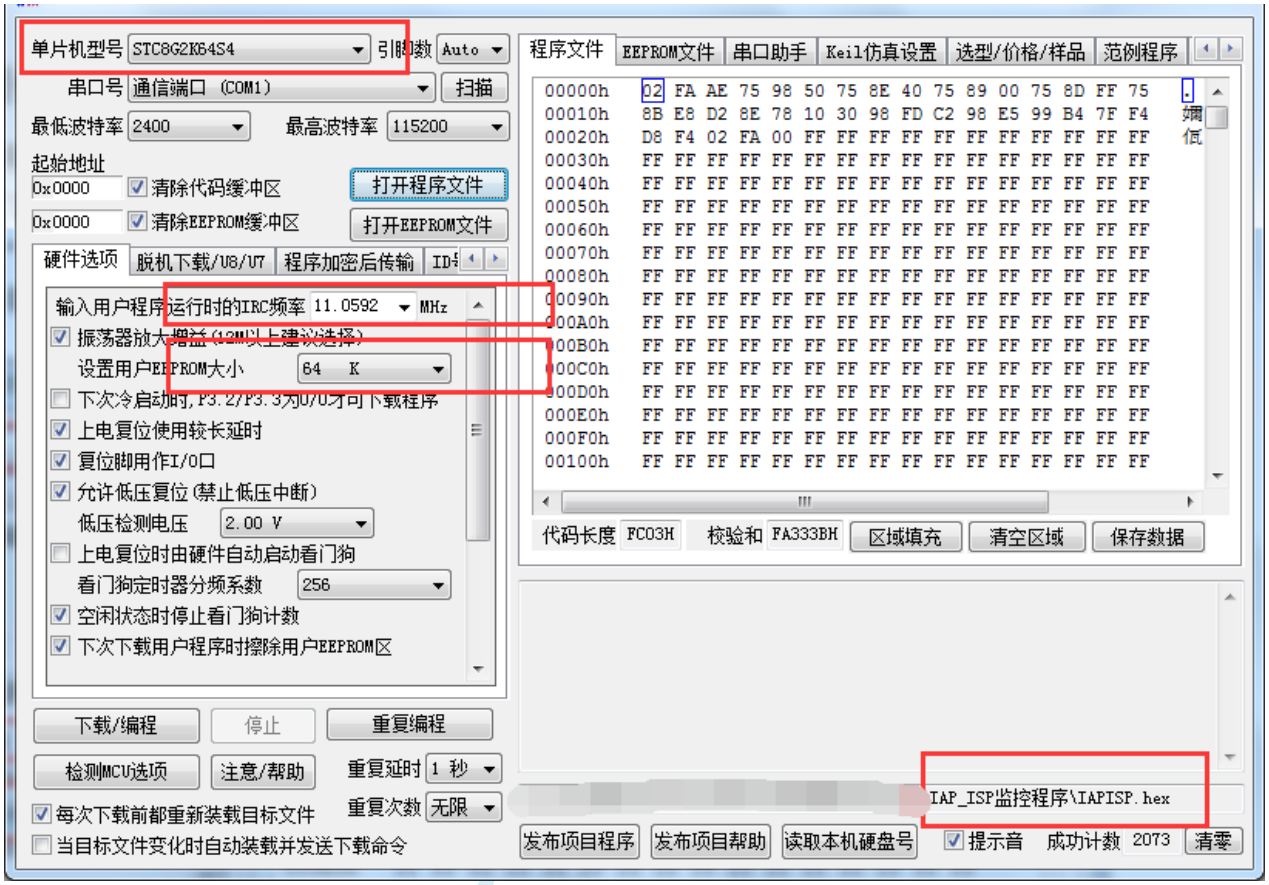
- 打开上位机界面，如下图



- 选择串口号，设置与下位机相同的串口波特率
- 打开要下载的源数据文件，Bin 或者 Intel hex 格式均可以
- 点击“下载数据”按钮即可开始下载数据

第六步、下位机固件程序的使用方法

下位机的目标文件有两个“IAPISP.hex”和“AP.hex”，对于一块新的单片机，第一次必须使用 ISP 下载工具将“IAPISP.hex”写入到芯片内，如下图所示。之后再更新便不再需要写“IAPISP.hex”这个文件了，附件中的“AP.hex”只是一个用户程序的模板，当满足下载条件时，用户只需要将 PC 值跳转到 FA00H 的地址，即可实现代码更新。



附录Q 用户程序复位到系统区进行 ISP 下载的方法（不停电）

当项目处于开发阶段时，需要反复的下载用户代码到目标芯片中进行代码验证，而 STC 的单片机进行正常的 ISP 下载都需要对目标芯片进行重新上电，从而会使得项目在开发阶段比较繁琐。为此 STC 单片机增加了一个特殊功能寄存器 IAP_CONTR，当用户向此寄存器写入 0x60，即可实现软件复位到系统区，进而实现不停电就可进行 ISP 下载。

但是用户如何判断是否正在进行 ISP 下载？何时向寄存器 IAP_CONTR 写 0x60 触发软复位？就这两个问题，下面分别介绍四种判断方法：

使用 P3.0 口检测串口起始信号

STC 单片机的串口 ISP 固定使用 P3.0 和 P3.1 两个端口，当 ISP 下载软件开始下载时，会发送握手命令到单片机的 P3.0 口。若用户的 P3.0 和 P3.1 只是专门用于 ISP 下载，则可使用 P3.0 口检测串口的起始信号来判断 ISP 下载。

C 语言代码

```
//测试工作频率为 11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         IAP_CONTR   =   0xc7;
sfr         P3M0        =   0xb2;
sfr         P3M1        =   0xb1;

sbit        P30         =   P3^0;

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;
    P30 = 1;

    while (1)
    {
        if (!P30) IAP_CONTR = 0x60;          //P3.0 的低电平即为串口起始信号
                                             //软件复位到系统区

        ...                                  //用户代码
    }
}
```

使用 P3.0/INT4 口的下降沿中断，检测串口起始信号

方法 B 与方法 A 类似，不同在于方法 A 使用的是查询方式，方法 B 使用中断方式。因为 STC 单片机的 P3.0 口为 INT4 的中断口。

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

sfr         IAP_CONTR   =   0xc7;
sfr         INTCLKO     =   0x8f;
sfr         P3M0        =   0xb2;
sfr         P3M1        =   0xb1;

void Int4Isr() interrupt 16                 //INT4 中断服务程序
{
    IAP_CONTR = 0x60;                       //串口起始信号触发 INT4 中断
                                            //软件复位到系统区
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    INTCLKO |= 0x40;                        //使能 INT4 中断
    EA = 1;

    while (1)
    {
        ...                                 //用户代码
    }
}
```

使用 P3.0/RxD 口的串口接收，检测 ISP 下载软件发送的 7F

方法 A 与方法 B 都非常简单，但容易受干扰，如果 P3.0 口有任何一个干扰信号，都会触发软件复位，所以方法 C 是对串口数据进行校验。

STC 的 ISP 下载软件进行 ISP 下载时，首先都会使用最低波特率（一般是 2400）+偶校验 9+1 位停止位连续发送握手命令 7F，因此用户可以在程序中，将串口设置为 9 位数据位+2400 波特率，然后持续检测 7F，比如连续检测到 8 个 7F 表示可确定需要进行 ISP 下载，此时再触发软件复位。

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"
```

```
#define   FOSC        11059200UL
#define   BR2400      (65536 - FOSC / 4 / 2400)

sfr       IAP_CONTR   =   0xc7;
sfr       AUXR        =   0x8e;
sfr       P3M0        =   0xb2;
sfr       P3M1        =   0xb1;

char cnt7f;

void UartIsr() interrupt 4                          //串口中断服务程序
{
    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;
        if ((SBUF == 0x7f) && (RB8 == 1))           //ISP 下载软件发送的握手命令 7F
                                                    //7F 的偶校验位为 1
        {
            if (++cnt7f == 8)                       //当连续检测到 8 个 7F 后
                IAP_CONTR = 0x60;                   //复位到系统区
        }
        else
        {
            cnt7f = 0;
        }
    }
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    SCON = 0xd0;                                    //设置串口为 9 位数据位
    TMOD = 0x00;
    AUXR = 0x40;
    TH1 = BR2400 >> 8;                              //设置串口波特率为 2400
    TL1 = BR2400;
    TR1 = 1;
    ES = 1;
    EA = 1;

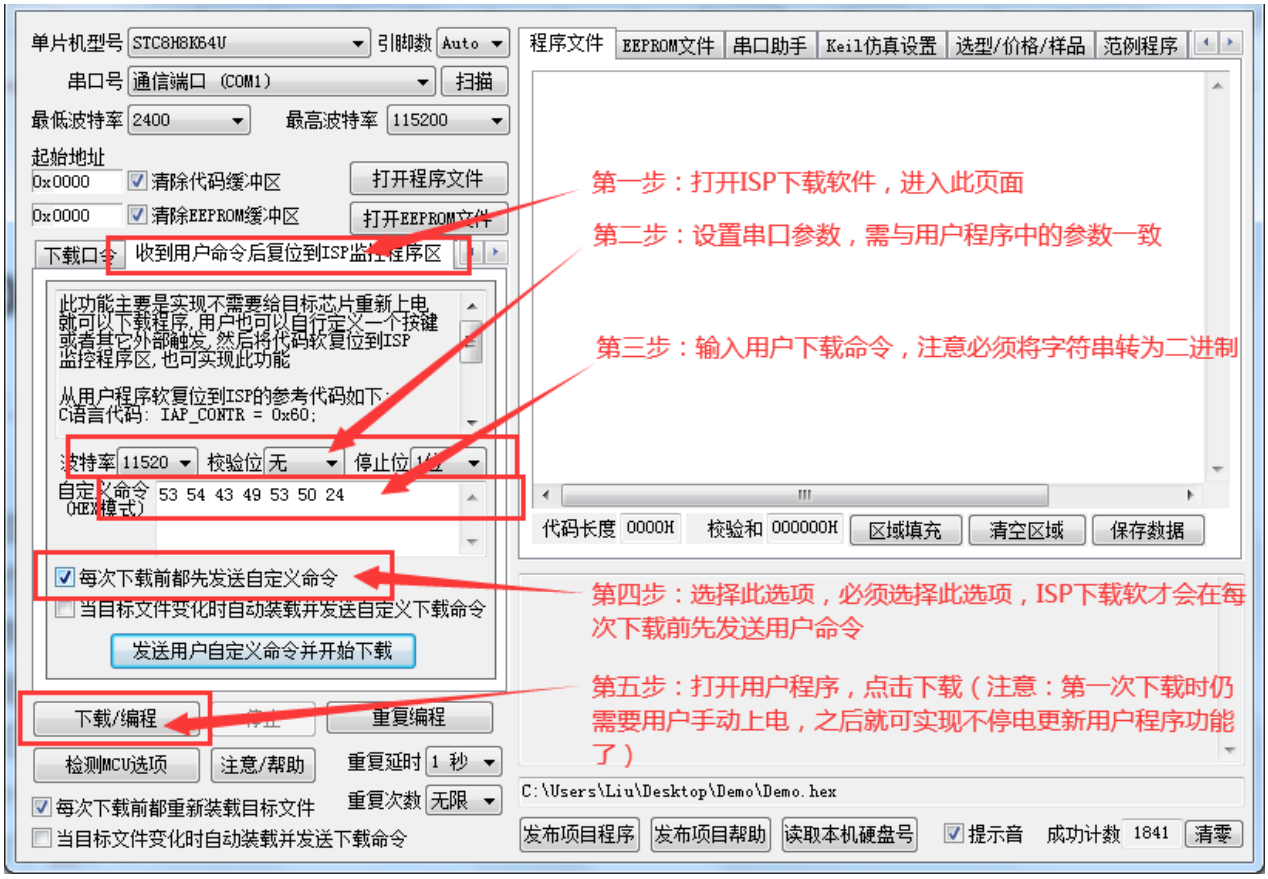
    cnt7f = 0;

    while (1)
    {
        ...                                         //用户代码
    }
}
```

使用 P3.0/RxD 串口接收，检测 ISP 下载软件发送的用户下载命令

如果用户代码中需要使用串口进行通信，则上面的 3 中方法可能都不太适用，此时可以使用 STC 的 ISP 下载软件提供的接口，定制一组专用的用户下载命令（可指定波特率、校验位和停止位），若使能此功能，ISP 下载软件在进行 ISP 下载前，会使用用户指定的波特率、校验位和停止位发送用户下载命令，然后再发送握手命令。用户只需要在自己的代码中监控串口命令序列，当检测到有正确的用户下载命令时，软件复位到系统区即可实现不停电进行 ISP 功能。

下面假设用户下载命令为字符串“STCISP$”，串口设置为波特率 115200，无校验位和 1 位停止位。ISP 下载软件中的设置如下图：



用户示例代码如下：

C 语言代码

```
//测试工作频率为11.0592MHz

#include "reg51.h"
#include "intrins.h"

#define    FOSC         11059200UL
#define    BR115200     (65536 - FOSC / 4 / 115200)

sfr        IAP_CONTR    =    0xc7;
sfr        AUXR         =    0x8e;
sfr        P3M0         =    0xb2;
sfr        P3M1         =    0xb1;
```

```
char stage;

void UartIsr() interrupt 4                          //串口中断服务程序
{
    char dat;

    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;

        dat = SBUF;
        switch (stage)
        {
        case 0:
        default:
L_Check1st:
            if (dat == 'S') stage = 1;
            else stage = 0;
            break;
        case 1:
            if (dat == 'T') stage = 2;
            else goto L_Check1st;
            break;
        case 2:
            if (dat == 'C') stage = 3;
            else goto L_Check1st;
            break;
        case 3:
            if (dat == 'I') stage = 4;
            else goto L_Check1st;
            break;
        case 4:
            if (dat == 'S') stage = 5;
            else goto L_Check1st;
            break;
        case 5:
            if (dat == 'P') stage = 6;
            else goto L_Check1st;
            break;
        case 6:
            if (dat == '$')                         //当检测到正确的用户下载命令时
                IAP_CONTR = 0x60;                   //复位到系统区
            else goto L_Check1st;
        break;
        }
    }
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    SCON = 0x50;                                    //设置用户串口模式为 8 位数据位
```

STC MCU

```
    TMOD = 0x00;
    AUXR = 0x40;
    TH1 = BR2400 >> 8;                          //设置串口波特率为115200
    TL1 = BR2400;
    TR1 = 1;
    ES = 1;
    EA = 1;

    stage = 0;

    while (1)
    {
        ...                                     //用户代码
    }
}
```

STC MCU

附录R　使用第三方 MCU 对 STC8C 系列单片机进行 ISP 下载范例程序

C 语言代码

```
//注意:使用本代码对 STC8C 系列的单片机进行下载时,必须要执行了 Download 代码之后,
//才能给目标芯片上电,否则目标芯片将无法正确下载

#include "reg51.h"

typedef    bit              BOOL;
typedef    unsigned char    BYTE;
typedef    unsigned short   WORD;

//宏、常量定义
#define    FALSE            0
#define    TRUE             1
#define    LOBYTE(w)        ((BYTE)(WORD)(w))
#define    HIBYTE(w)        ((BYTE)((WORD)(w) >> 8))

#define    MINBAUD          2400L
#define    MAXBAUD          115200L

#define    FOSC             11059200L          //主控芯片工作频率
#define    BR(n)            (65536 - FOSC/4/(n))    //主控芯片串口波特率计算公式
#define    T1MS             (65536 - FOSC/1000)     //主控芯片 1ms 定时初值

#define    FUSER            24000000L          //STC8C 系列目标芯片工作频率
#define    RL(n)            (65536 - FUSER/4/(n))   //STC8C 系列目标芯片串口波特率计算公式

sfr        AUXR = 0x8e;
sfr        P3M1 = 0xB1;
sfr        P3M0 = 0xB2;

//变量定义
BOOL f1ms;                          //1ms 标志位
BOOL UartBusy;                      //串口发送忙标志位
BOOL UartReceived;                  //串口数据接收完成标志位
BYTE UartRecvStep;                  //串口数据接收控制
BYTE TimeOut;                       //串口通讯超时计数器
BYTE xdata TxBuffer[256];           //串口数据发送缓冲区
BYTE xdata RxBuffer[256];           //串口数据接收缓冲区
char code DEMO[256];                //演示代码数据

//函数声明
void Initial(void);
void DelayXms(WORD x);
BYTE UartSend(BYTE dat);
void CommInit(void);
void CommSend(BYTE size);
BOOL Download(BYTE *pdat, long size);
```

```
//主函数入口
void main(void)
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    Initial();
    if (Download(DEMO, 256))
    {
        //下载成功
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0x00;
        DelayXms(500);
        P3 = 0xff;
    }
    else
    {
        //下载失败
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
        DelayXms(500);
        P3 = 0xf3;
        DelayXms(500);
        P3 = 0xff;
    }

    while (1);
}

//1ms 定时器中断服务程序
void tm0(void) interrupt 1
{
    static BYTE Counter100;

    f1ms = TRUE;
    if (Counter100-- == 0)
    {
        Counter100 = 100;
        if (TimeOut) TimeOut--;
    }
}
```

STC MCU

```
//串口中断服务程序
void uart(void) interrupt 4
{
    static WORD RecvSum;
    static BYTE RecvIndex;
    static BYTE RecvCount;
    BYTE dat;

    if (TI)
    {
        TI = 0;
        UartBusy = FALSE;
    }

    if (RI)
    {
        RI = 0;
        dat  = SBUF;
        switch (UartRecvStep)
        {
        case 1:
            if (dat != 0xb9) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 2:
            if (dat != 0x68) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 3:
            if (dat != 0x00) goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 4:
            RecvSum = 0x68 + dat;
            RecvCount = dat - 6;
            RecvIndex = 0;
            UartRecvStep++;
            break;
        case 5:
            RecvSum += dat;
            RxBuffer[RecvIndex++] = dat;
            if (RecvIndex == RecvCount)     UartRecvStep++;
            break;
        case 6:
            if (dat != HIBYTE(RecvSum))     goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 7:
            if (dat != LOBYTE(RecvSum))     goto L_CheckFirst;
            UartRecvStep++;
            break;
        case 8:
            if (dat != 0x16) goto L_CheckFirst;
            UartReceived = TRUE;
            UartRecvStep++;
            break;
L_CheckFirst:
        case 0:
        default:
```

STC MCU

```
                CommInit();
                UartRecvStep = (dat == 0x46     ? 1 : 0);
                break;
            }
        }
    }

//系统初始化
void Initial(void)
{
    UartBusy = FALSE;

    SCON = 0xd0;                    //串口数据模式必须为 8 位数据+1 位偶检验
    AUXR = 0xc0;
    TMOD = 0x00;
    TH0 = HIBYTE(T1MS);
    TL0 = LOBYTE(T1MS);
    TR0 = 1;
    TH1 = HIBYTE(BR(MINBAUD));
    TL1 = LOBYTE(BR(MINBAUD));
    TR1 = 1;
    ET0 = 1;
    ES = 1;
    EA = 1;
}

//Xms 延时程序
void DelayXms(WORD x)
{
    do
    {
        f1ms = FALSE;
        while (!f1ms);
    } while (x--);
}

//串口数据发送程序
BYTE UartSend(BYTE dat)
{
    while (UartBusy);

    UartBusy = TRUE;
    ACC = dat;
    TB8 = P;
    SBUF = ACC;

    return dat;
}

//串口通讯初始化
void CommInit(void)
{
    UartRecvStep = 0;
    TimeOut = 20;
    UartReceived = FALSE;
}

//发送串口通讯数据包
void CommSend(BYTE size)
```

STC MCU

```
{
    WORD sum;
    BYTE i;

    UartSend(0x46);
    UartSend(0xb9);
    UartSend(0x6a);
    UartSend(0x00);
    sum = size + 6 + 0x6a;
    UartSend(size + 6);
    for (i=0; i<size; i++)
    {
        sum  += UartSend(TxBuffer[i]);
    }
    UartSend(HIBYTE(sum));
    UartSend(LOBYTE(sum));
    UartSend(0x16);
    while (UartBusy);

    CommInit();
}

//对 STC15H 系列的芯片进行 ISP 下载程序
BOOL Download(BYTE *pdat, long size)
{
    BYTE arg;
    BYTE offset;
    BYTE cnt;
    WORD addr;

    //握手
    CommInit();
    while (1)
    {
        if (UartRecvStep == 0)
        {
            UartSend(0x7f);
            DelayXms(10);
        }
        if (UartReceived)
        {
            arg = RxBuffer[4];
            if (RxBuffer[0] == 0x50) break;
            return FALSE;
        }
    }

    //设置参数(设置从芯片使用最高的波特率以及等待时间等参数)
    TxBuffer[0] = 0x01;
    TxBuffer[1] = arg;
    TxBuffer[2] = 0x40;
    TxBuffer[3] = HIBYTE(RL(MAXBAUD));
    TxBuffer[4] = LOBYTE(RL(MAXBAUD));
    TxBuffer[5] = 0x00;
    TxBuffer[6] = 0x00;
    TxBuffer[7] = 0x97;
    CommSend(8);
    while (1)
    {
```

STC MCU

```
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x01) break;
            return FALSE;
        }
    }

    //准备
    TH1 = HIBYTE(BR(MAXBAUD));
    TL1 = LOBYTE(BR(MAXBAUD));
    DelayXms(10);
    TxBuffer[0] = 0x05;
    TxBuffer[1] = 0x00;
    TxBuffer[2] = 0x00;
    TxBuffer[3] = 0x5a;
    TxBuffer[4] = 0xa5;
    CommSend(5);
    while (1)
    {
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x05) break;
            return FALSE;
        }
    }

    //擦除
    DelayXms(10);
    TxBuffer[0] = 0x03;
    TxBuffer[1] = 0x00;
    TxBuffer[2] = 0x00;
    TxBuffer[3] = 0x5a;
    TxBuffer[4] = 0xa5;
    CommSend(5);
    TimeOut = 100;
    while (1)
    {
        if (TimeOut == 0) return FALSE;
        if (UartReceived)
        {
            if (RxBuffer[0] == 0x03) break;
            return FALSE;
        }
    }

    //写用户代码
    DelayXms(10);
    addr = 0;
    TxBuffer[0] = 0x22;
    TxBuffer[3] = 0x5a;
    TxBuffer[4] = 0xa5;
    offset = 5;
    while (addr < size)
    {
        TxBuffer[1] = HIBYTE(addr);
        TxBuffer[2] = LOBYTE(addr);
        cnt    = 0;
```

STC MCU

```
        while (addr < size)
        {
            TxBuffer[cnt+offset] = pdat[addr];
            addr++;
            cnt++;
            if (cnt >= 128) break;
        }
        CommSend(cnt + offset);
        while (1)
        {
            if (TimeOut == 0) return FALSE;
            if (UartReceived)
            {
                if ((RxBuffer[0] == 0x02) && (RxBuffer[1] == 'T')) break;
                return FALSE;
            }
        }
        TxBuffer[0] = 0x02;
    }

    ////写硬件选项
    ////如果不需要修改硬件选项,此步骤可直接跳过,此时所有的硬件选项
    ////都维持不变,MCU 的频率为上一次所调节频率
    ////若写硬件选项,MCU 的内部 IRC 频率将被固定写为 24M, ,其他选项恢复为出厂设置
    ////建议:第一次使用 AIapp-ISP 下载软件将从芯片的硬件选项设置好
    ////以后再使用主芯片对从芯片下载程序时不写硬件选项
    //DelayXms(10);
    //for (cnt=0; cnt<128; cnt++)
    //{
    //    TxBuffer[cnt] = 0xff;
    //}
    //TxBuffer[0] = 0x04;
    //TxBuffer[1] = 0x00;
    //TxBuffer[2] = 0x00;
    //TxBuffer[3] = 0x5a;
    //TxBuffer[4] = 0xa5;
    //TxBuffer[33] = arg;
    //TxBuffer[34] = 0x00;
    //TxBuffer[35] = 0x01;
    //TxBuffer[41] = 0xbf;
    //TxBuffer[42] = 0xbd;                  //P5.4 为 I/O 口
    ////TxBuffer[42] = 0xad;                //P5.4 为复位脚
    //TxBuffer[43] = 0xf7;
    //TxBuffer[44] = 0xff;
    //CommSend(45);
    //while (1)
    //{
    //    if (TimeOut == 0) return FALSE;
    //    if (UartReceived)
    //    {
    //        if ((RxBuffer[0] == 0x04) && (RxBuffer[1] == 'T')) break;
    //        return FALSE;
    //    }
    //}

    // 下载完成
    return TRUE;
}
```

```
char code DEMO[256] =
{
    0x80,0x00,0x75,0xB2,0xFF,0x75,0xB1,0x00,0x05,0xB0,0x11,0x0E,0x80,0xFA,0xD8,0xFE,
    0xD9,0xFC,0x22,
};
```

备注：用户若需要设置不同的工作频率，可参考 7.3.7 和 7.3.8 章的范例代码

STC MCU

附录S 使用第三方应用程序调用 STC 发布项目程序对单片机进行 ISP 下载

使用 STC 的 ISP 下载软件生成的发布项目程序为可执行的 EXE 格式文件，用户可直接双击发布的项目程序运行进行 ISP 下载，也可在第三方的应用程序中调用发布项目程序进行 ISP 下载。下面介绍两种调用的方法。

简单调用

在第三方应用程序中只是简单创建发布项目程序的进程，其他的所有下载操作均在发布项目程序中进行，第三方应用程序此时只需要等待发布项目程序操作完成后，清理现场即可。

VC 代码

```
BOOL IspProcess()
{
    //定义相关变量
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    CString path;

    //发布项目程序的完整路径
    path = _T("D:\\Work\\Upgrade.exe");

    //变量初始化
    memset(&si, 0, sizeof(STARTUPINFO));
    memset(&pi, 0, sizeof(PROCESS_INFORMATION));

    //设置启动变量
    si.cb = sizeof(STARTUPINFO);
    GetStartupInfo(&si);
    si.wShowWindow = SW_SHOWNORMAL;
    si.dwFlags = STARTF_USESHOWWINDOW;

    //创建发布项目程序进程
    if (CreateProcess(NULL, (LPTSTR)(LPCTSTR)path, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
    {
        //等待发布项目程序操作完成
        //由于此处会阻塞主进程，所以建议新建工作进程，在工作进程中进行等待
        WaitForSingleObject(pi.hProcess,INFINITE);

        //清理工作
        CloseHandle(pi.hThread);
        CloseHandle(pi.hProcess);

        return TRUE;
    }
    else
    {
        AfxMessageBox(_T("创建进程失败 !"));

        return FALSE;
```

STC MCU

```
    }
}
```

高级调用

在第三方应用程序创建发布项目程序的进程，并在第三方应用程序中进行包括选择串口、开始 ISP 编程、停振 ISP 编程以及关闭发布项目程序等的全部 ISP 下载操作，而不需要在发布项目程序中进行界面互动。

VC 代码

```
//定义回调函数参数的数据结构
struct CALLBACK_PARAM
{
    DWORD dwProcessId;                          //主进程 ID
    HWND hMainWnd;                              //主窗口句柄
};

//枚举窗口的回调函数，用于获取主窗口句柄
BOOL CALLBACK EnumWindowCallBack(HWND hWnd, LPARAM lParam)
{
    CALLBACK_PARAM *pcp = (CALLBACK_PARAM *)lParam;
    DWORD id;

    GetWindowThreadProcessId(hWnd, &id);
    if ((pcp->dwProcessId == id) && (GetParent(hWnd) == NULL))
    {
        pcp->hMainWnd = hWnd;

        return FALSE;
    }

    return TRUE;
}

BOOL IspProcess()
{
    //定义相关变量
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    CALLBACK_PARAM cp;
    CString path;

    //发布项目程序中部分控件的 ID
    const UINT ID_PROGRAM       = 1013;
    const UINT ID_STOP          = 1012;
    const UINT ID_COMPORT       = 1001;
    const UINT ID_PROGRESS      = 1000;

    //发布项目程序的完整路径
    path = _T("D:\\Work\\Upgrade.exe");

    //变量初始化
    memset(&si, 0, sizeof(STARTUPINFO));
    memset(&pi, 0, sizeof(PROCESS_INFORMATION));
    memset(&cp, 0, sizeof(CALLBACK_PARAM));

    //设置启动变量
```

```
    si.cb = sizeof(STARTUPINFO);
    GetStartupInfo(&si);
    si.wShowWindow = SW_SHOWNORMAL;                    //此处若设置为SW_HIDE,就不会显示发布项目程序
                                                       //的操作界面,所有的ISP 操作都可在后台进行
    si.dwFlags = STARTF_USESHOWWINDOW;

    //创建发布项目程序进程
    if (CreateProcess(NULL, (LPTSTR)(LPCTSTR)path, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
    {
        //等待发布项目程序进程初始化完成
        WaitForInputIdle(pi.hProcess, 5000);

        //获取发布项目程序的主窗口句柄
        cp.dwProcessId = pi.dwProcessId;
        cp.hMainWnd = NULL;
        EnumWindows(EnumWindowCallBack, (LPARAM)&cp);

        if (cp.hMainWnd != NULL)
        {
            HWND hProgram;
            HWND hStop;
            HWND hPort;

            //获取发布项目程序主窗口中部分控件句柄
            hProgram = ::GetDlgItem(cp.hMainWnd, ID_PROGRAM);
            hStop = ::GetDlgItem(cp.hMainWnd, ID_STOP);
            hPort = ::GetDlgItem(cp.hMainWnd, ID_COMPORT);

            //设置发布项目程序中的串口号, 第3 个参数为0:COM1, 1:COM2, 2:COM3, ...
            ::SendMessage(hPort, CB_SETCURSEL, 0, 0);

            //触发编程按钮开始ISP 编程
            ::SendMessage(hProgram, BM_CLICK, 0, 0);

            //等待编程完成
            //由于此处会阻塞主进程，所以建议新建工作进程，在工作进程中进行等待
            while (!::IsWindowEnabled(hProgram));

            //编程完成后关闭发布项目程序
            ::SendMessage(cp.hMainWnd, WM_CLOSE, 0, 0);
        }

        //等待进程结束
        WaitForSingleObject(pi.hProcess,INFINITE);

        //清理工作
        CloseHandle(pi.hThread);
        CloseHandle(pi.hProcess);

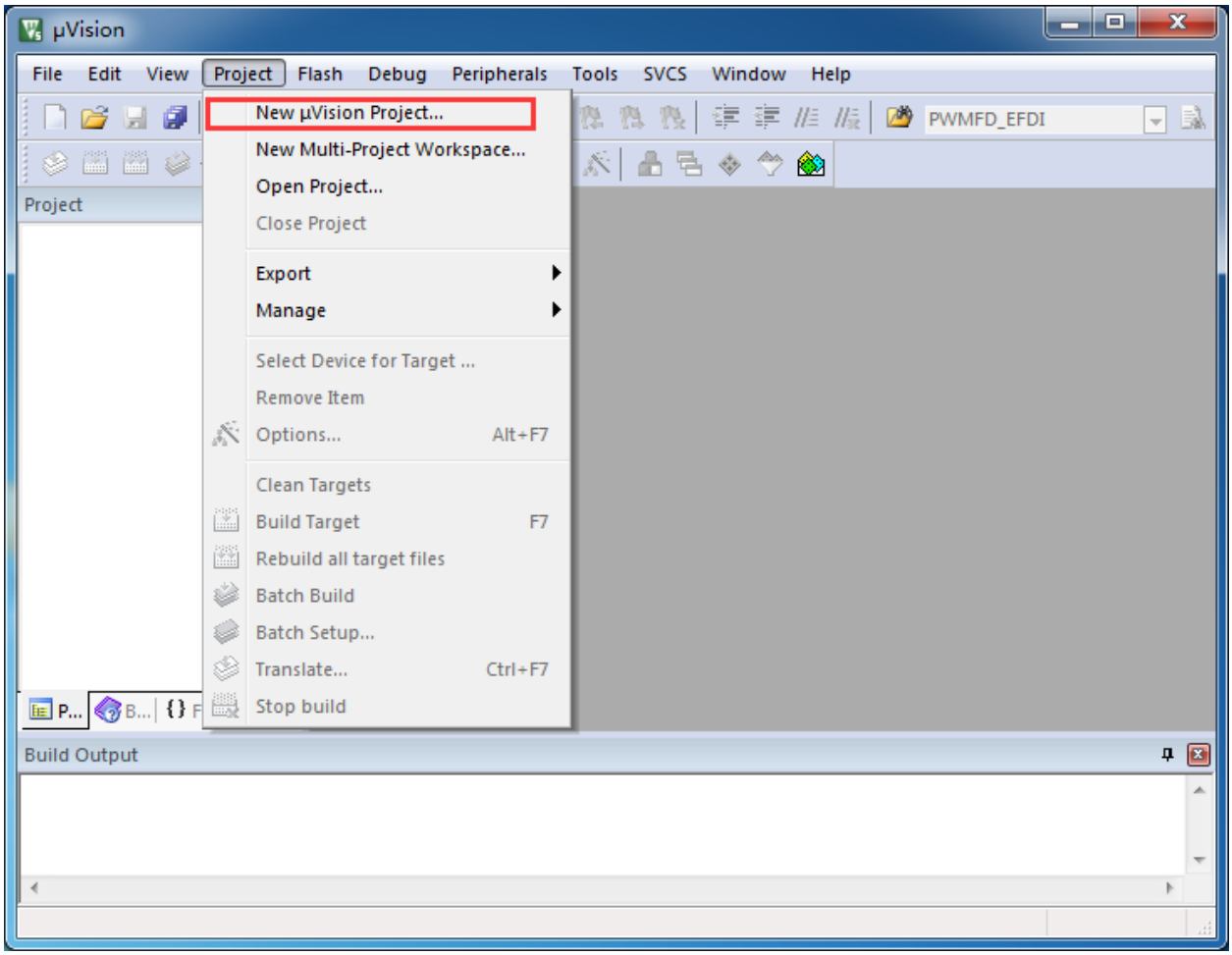
        return TRUE;
    }
    else
    {
        AfxMessageBox(_T("创建进程失败 !"));

        return FALSE;
    }
}
```

附录T　在 Keil 中建立多文件项目的方法

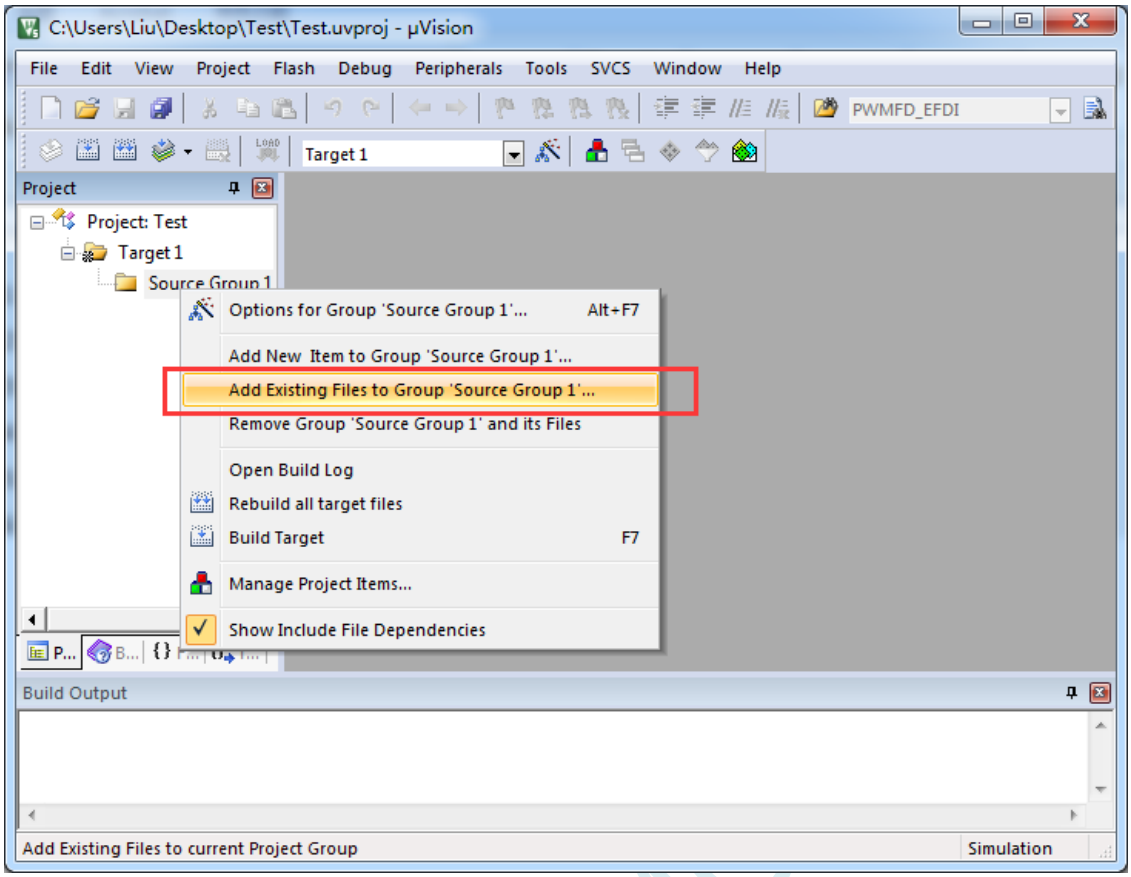
在 Keil 中，一般比较小的项目都只有一个源文件，但对于一些稍微复杂的项目往往需要多个源文件建立多文件项目的方法如下：

1、首先打开 Keil，在菜单“Project”中选择“New uVision Project ...”

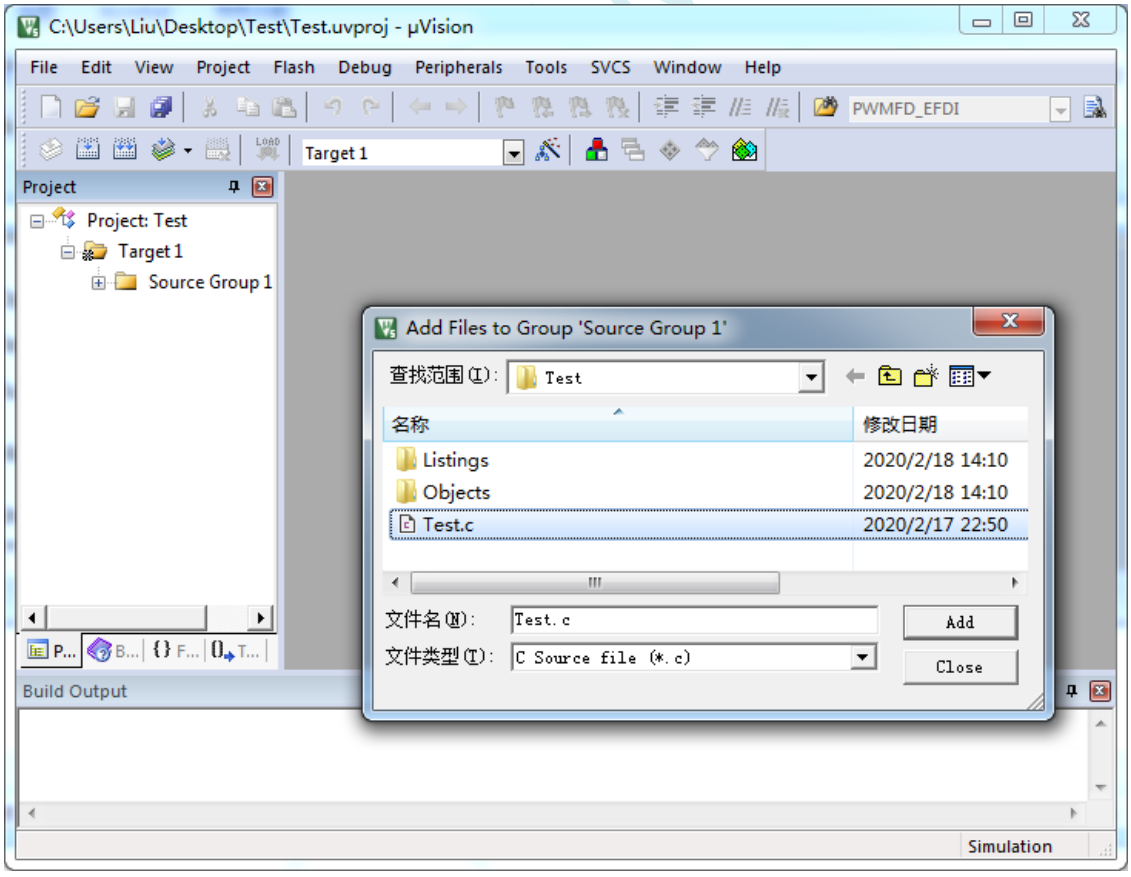


即可完成一个空项目的建立

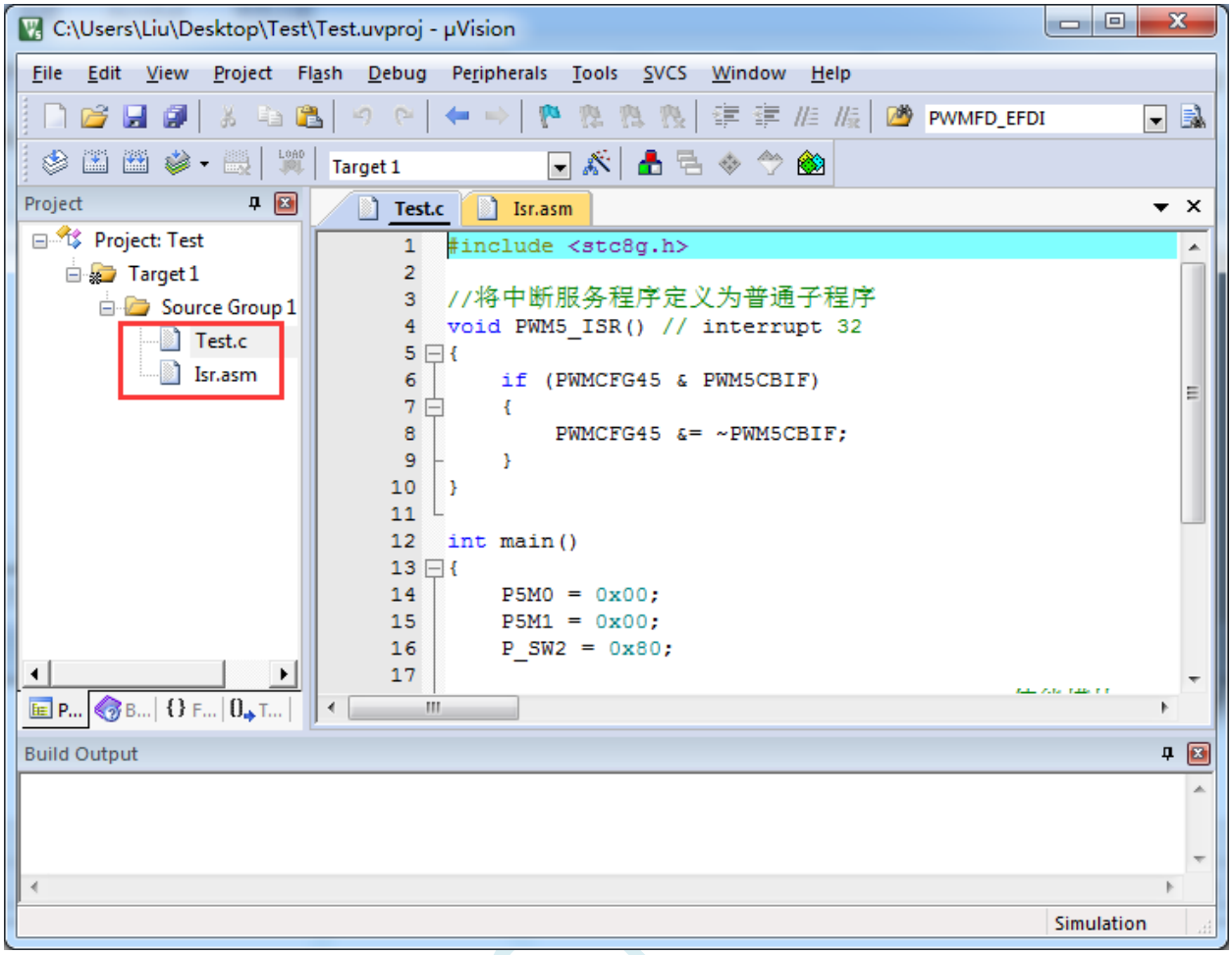
2、在空项目的项目树中，鼠标右键单击“Source Group 1”，并选择右键菜单中的“Add Existing Files to Group "Source Group 1" ...”



3、在弹出的文件对话框中，多次添加源文件

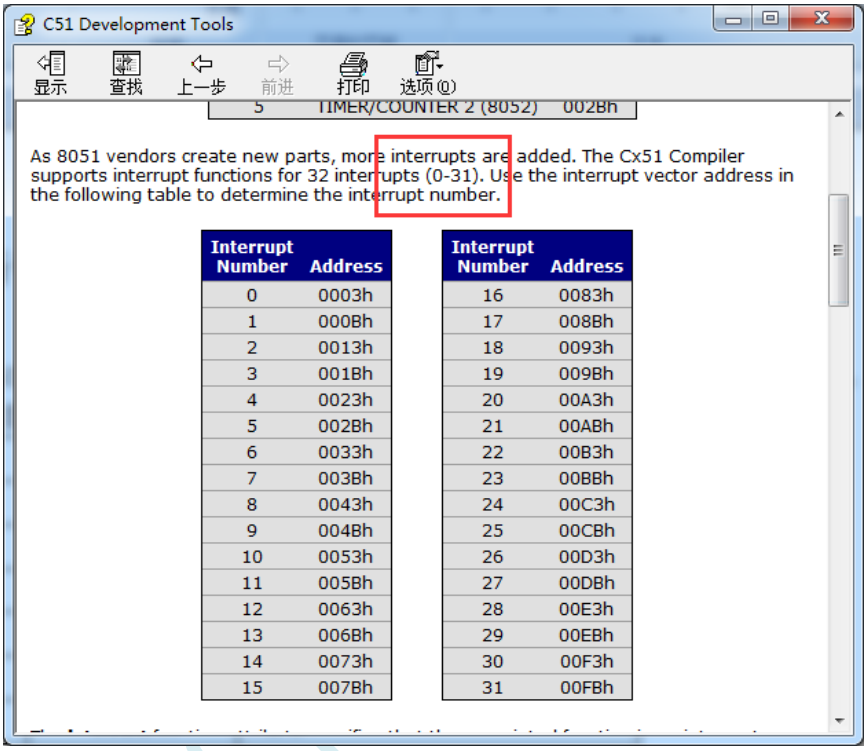


如下图所示即可完成多文件项目的建立


C:\Users\Liu\Desktop\Test\Test.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
PWMFD_EFDI
Target 1
Project
Project: Test
Target 1
Source Group 1
Test.c
Isr.asm
Test.c
Isr.asm
1 #include <stc8g.h>
2
3 //将中断服务程序定义为普通子程序
4 void PWM5_ISR() // interrupt 32
5 {
6 if (PWMCFG45 & PWM5CBIF)
7 {
8 PWMCFG45 &= ~PWM5CBIF;
9 }
10 }
11
12 int main()
13 {
14 P5M0 = 0x00;
15 P5M1 = 0x00;
16 P_SW2 = 0x80;
17
Build Output
Simulation

附录U　关于中断号大于 31 在 Keil 中编译出错的处理

在 Keil 的 C51 编译环境下，中断号只支持 0~31，即中断向量必须小于 0100H。



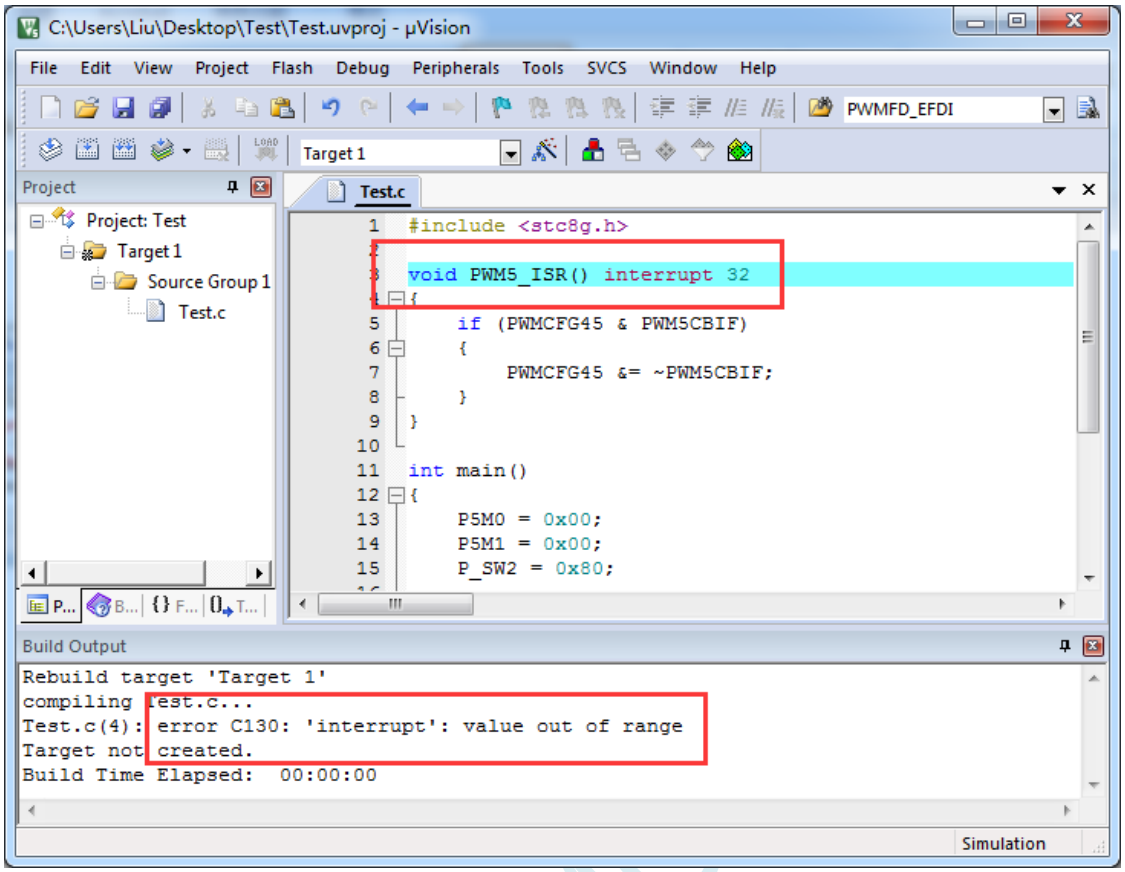
Interrupt Number	Address	Interrupt Number	Address
0	0003h	16	0083h
1	000Bh	17	008Bh
2	0013h	18	0093h
3	001Bh	19	009Bh
4	0023h	20	00A3h
5	002Bh	21	00ABh
6	0033h	22	00B3h
7	003Bh	23	00BBh
8	0043h	24	00C3h
9	004Bh	25	00CBh
10	0053h	26	00D3h
11	005Bh	27	00DBh
12	0063h	28	00E3h
13	006Bh	29	00EBh
14	0073h	30	00F3h
15	007Bh	31	00FBh

下表是 STC 目前所有系列的中断列表：

中断号	中断向量	中断类型
0	0003 H	INT0
1	000B H	定时器 0
2	0013 H	INT1
3	001B H	定时器 1
4	0023 H	串口 1
5	002B H	ADC
6	0033 H	LVD
7	003B H	PCA
8	0043 H	串口 2
9	004B H	SPI
10	0053 H	INT2
11	005B H	INT3
12	0063 H	定时器 2
13	**006B H**	
14	0073 H	系统内部中断
15	007B H	系统内部中断

中断号	中断向量	中断类型
16	0083 H	INT4
17	008B H	串口 3
18	0093 H	串口 4
19	009B H	定时器 3
20	00A3 H	定时器 4
21	00AB H	比较器
22	00B3 H	波形发生器 0
23	00BB H	波形发生器异常 0
24	00C3 H	I2C
25	00CB H	USB
26	00D3 H	PWM1
27	00DB H	PWM2
28	00E3 H	波形发生器 1
29	00EB H	波形发生器 2
30	00F3 H	波形发生器 3
31	00FB H	波形发生器 4
32	**0103 H**	**波形发生器 5**
33	**010B H**	**波形发生器异常 2**
34	**0113 H**	**波形发生器异常 4**
35	**011B H**	**触摸按键**
36	**0123 H**	**RTC**
37	**012B H**	**P0 口中断**
38	**0133 H**	**P1 口中断**
39	**013B H**	**P2 口中断**
40	**0143 H**	**P3 口中断**
41	**014B H**	**P4 口中断**
42	**0153 H**	**P5 口中断**
43	**015B H**	**P6 口中断**
44	**0163 H**	**P7 口中断**
45	**016B H**	**P8 口中断**
46	**0173 H**	**P9 口中断**

不难发现，从波形发生器 5 中断开始，后面所有的中断服务程序，在 keil 中均会编译出错，如下图所示：



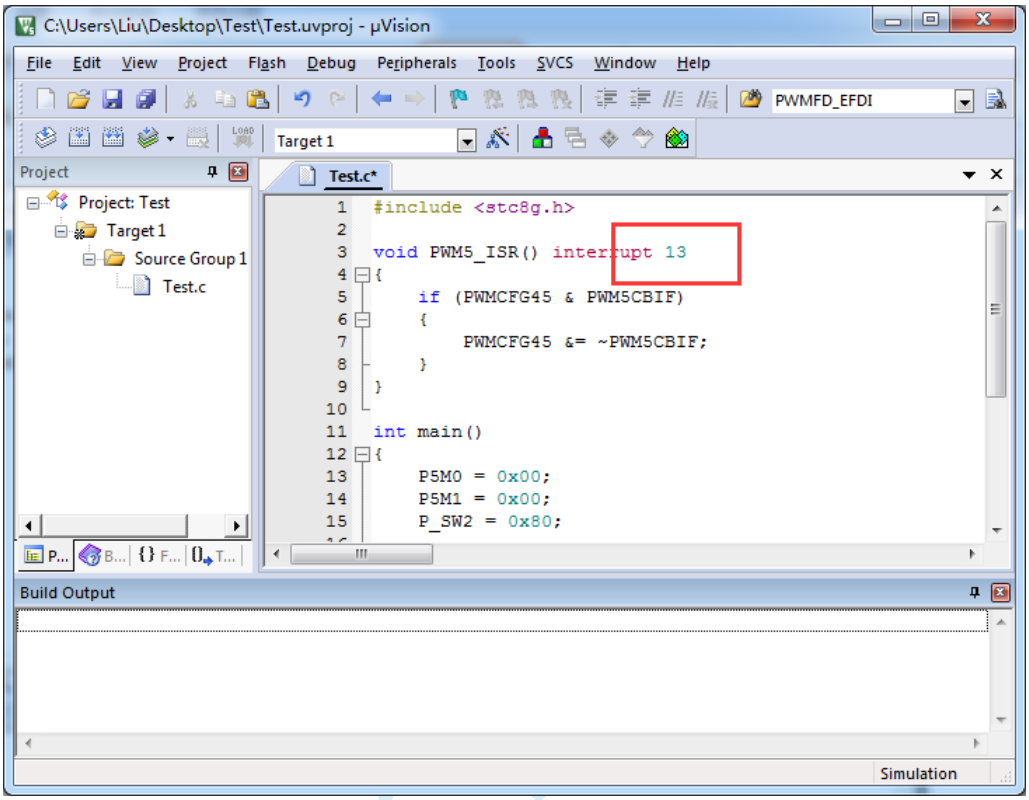
处理这种错误有如下三种方法：（均需要借助于汇编代码，优先推荐使用方法 1）

方法 1：借用 13 号中断向量

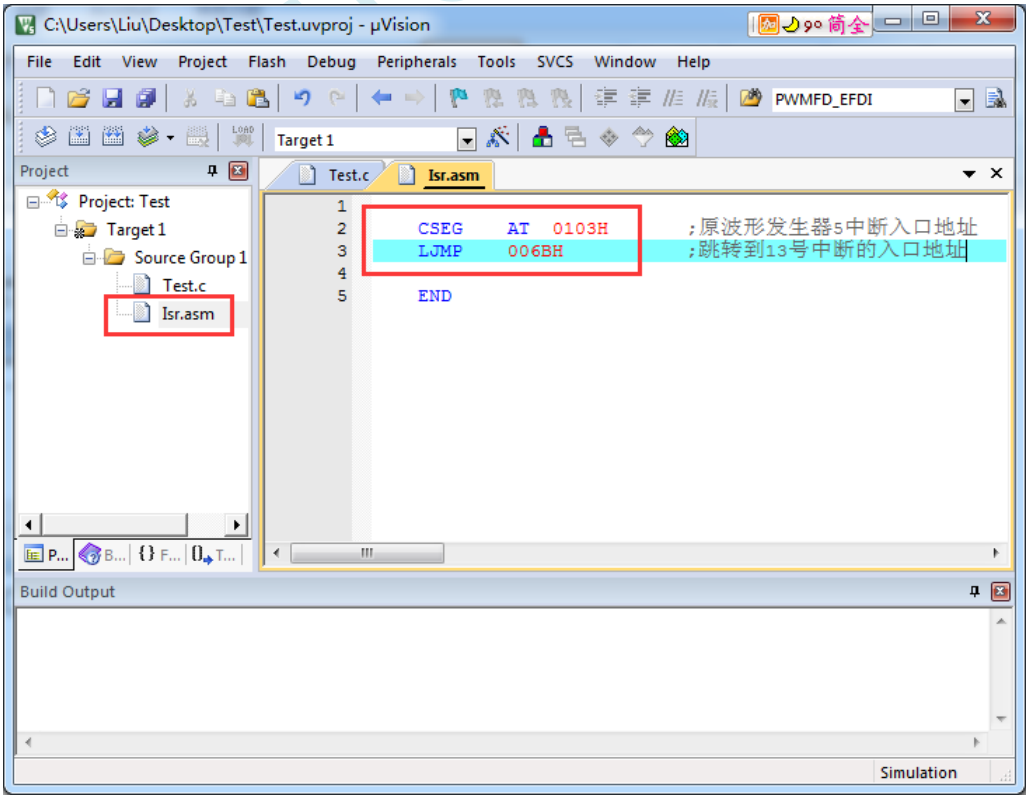
0~31 号中断中，第 13 号是保留中断号，我们可以借用此中断号

操作步骤如下：

1、将我们报错的中断号改为“13”，如下图：

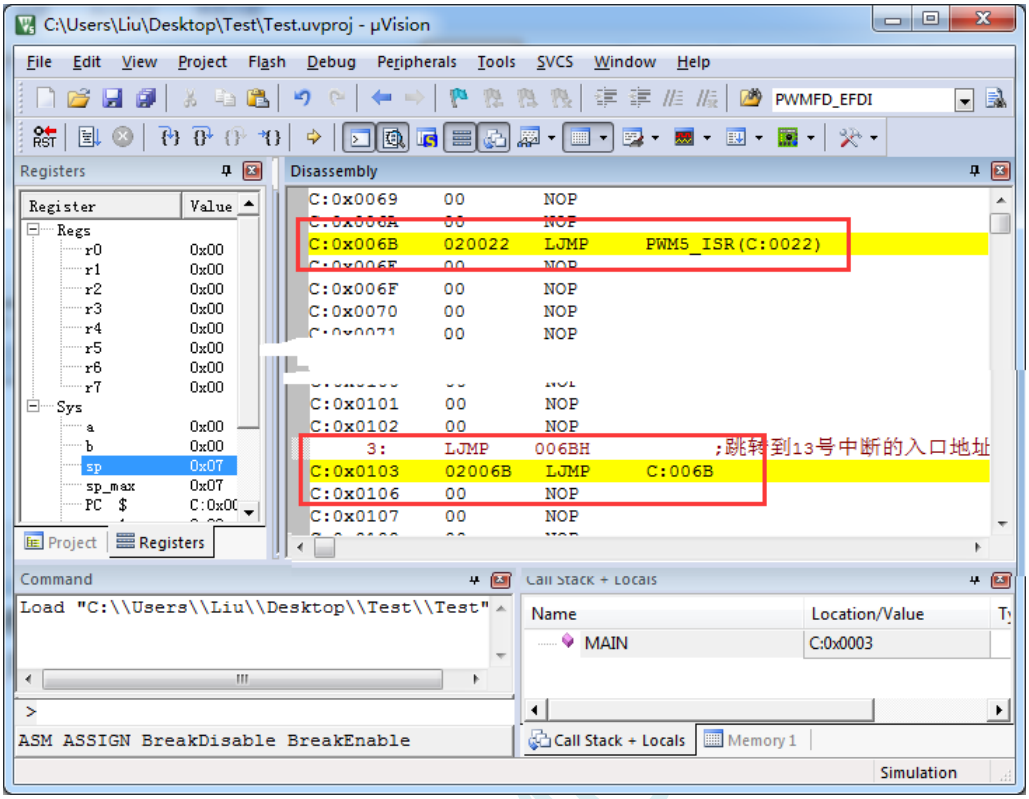


2、新建一个汇编语言文件，比如“isr.asm”，加入到项目，并在地址“0103H”的地方添加一条“LJMP 006BH”，如下图：

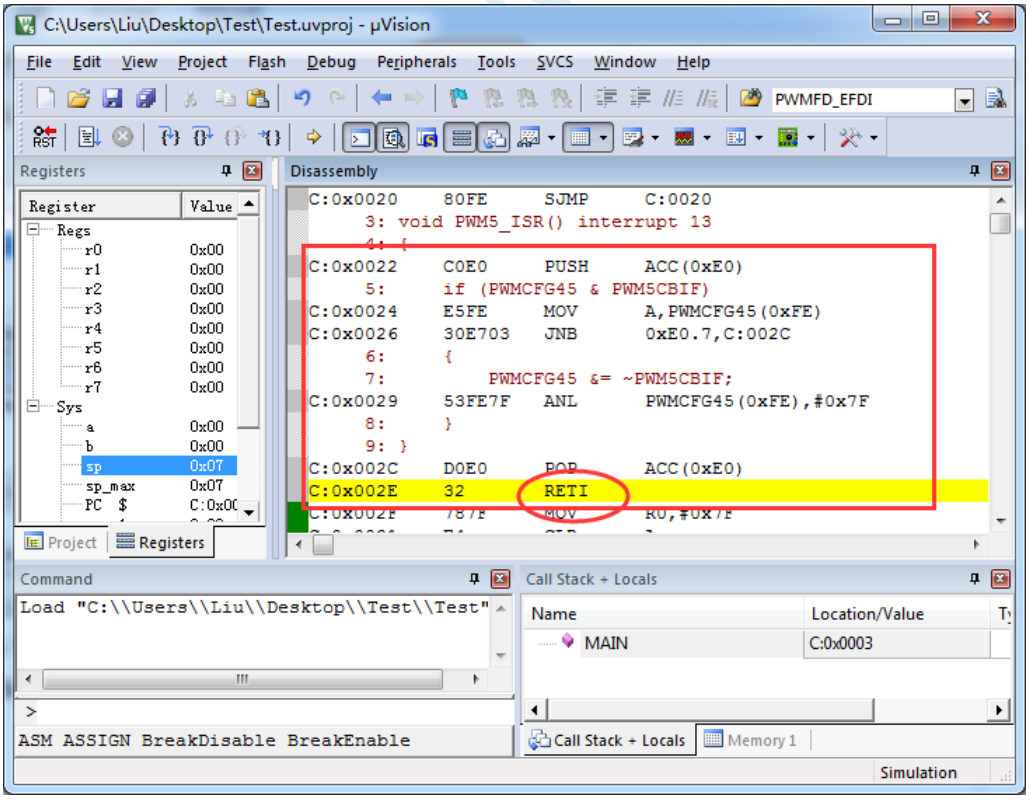


3、编译即可通过。

此时经过 Keil 的 C51 编译器编译后，在 006BH 处有一条“LJMP　PWM5_ISR”，在 0103H 处有一条“LJMP　006BH”，如下图：



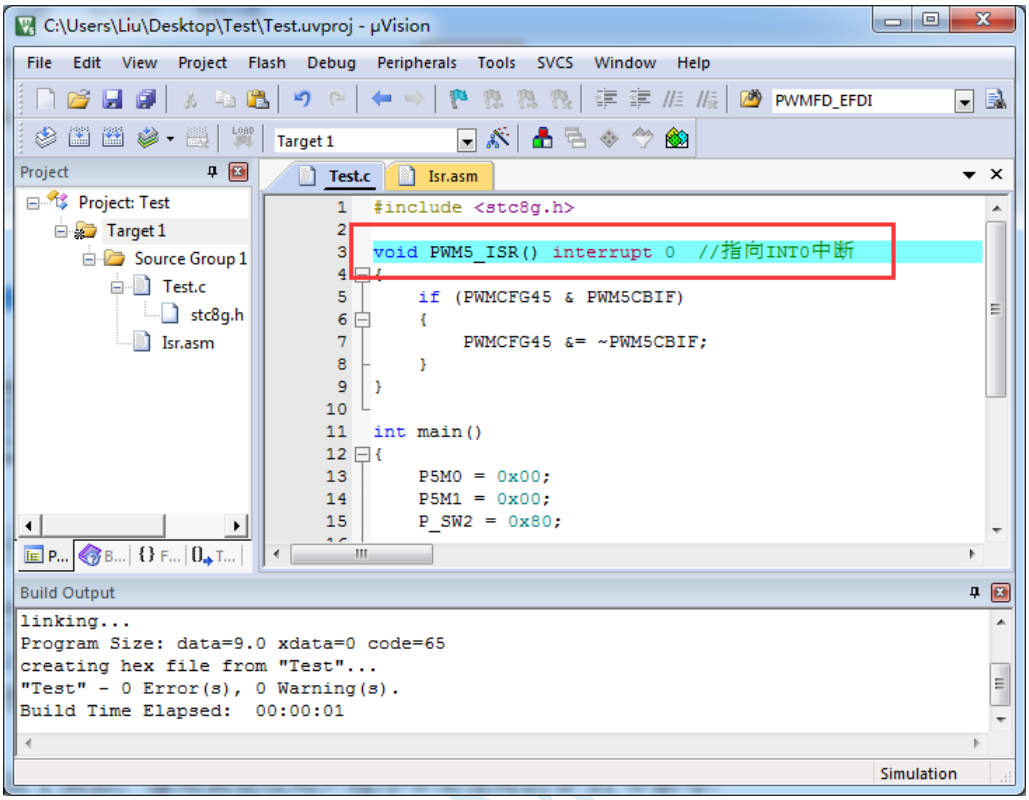
当发生 PWM5 中断时，硬件会自动跳转到 0103H 地址执行“LJMP　006BH”，然后在 006BH 处再执行“LJMP　PWM5_ISR”即可跳转到真正的中断服务程序，如下图：

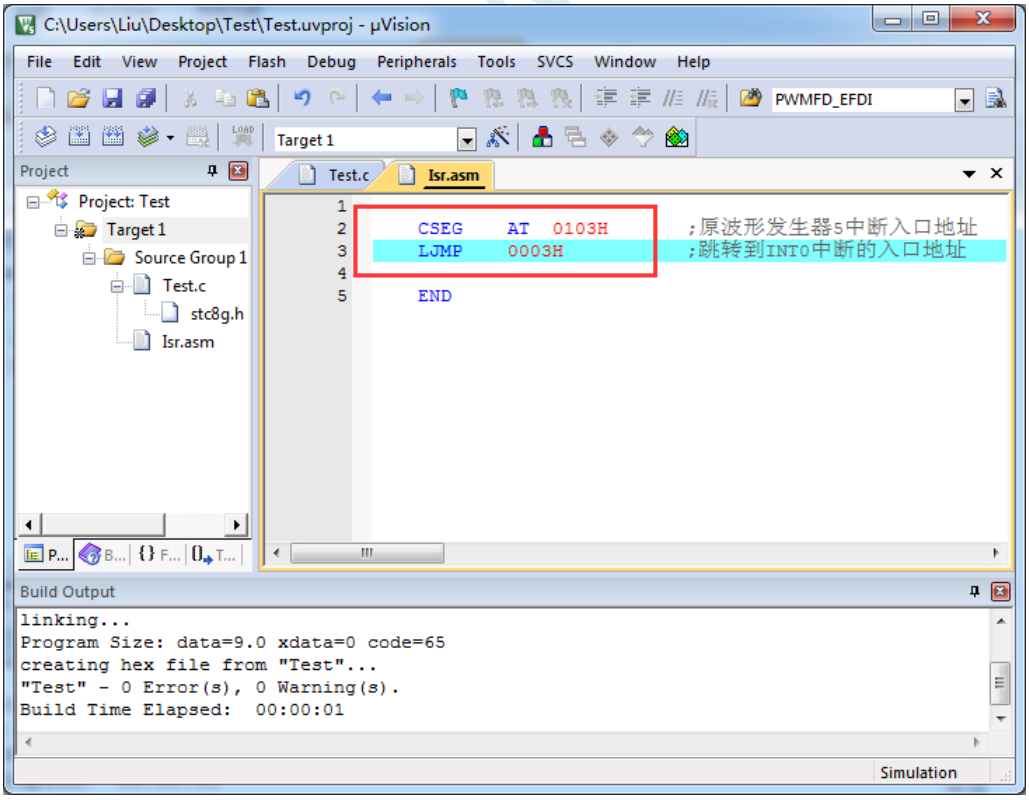


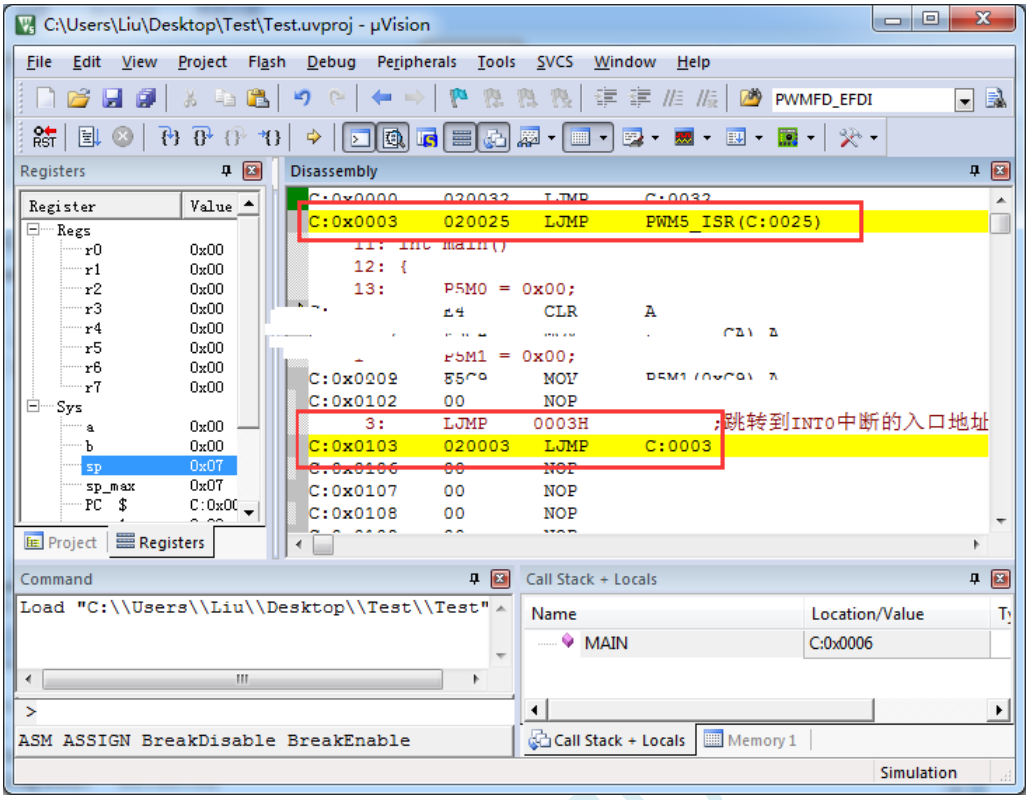
中断服务程序执行完成后，再通过 RETI 指令返回。整个中断响应过程只是多执行了一条 LJMP 语句而已。

方法 2：与方法 1 类似，借用用户程序中未使用的 0~31 的中断号

比如在用户的代码中，没有使用 INT0 中断，则可将上面的代码作类似与方法 1 的修改：





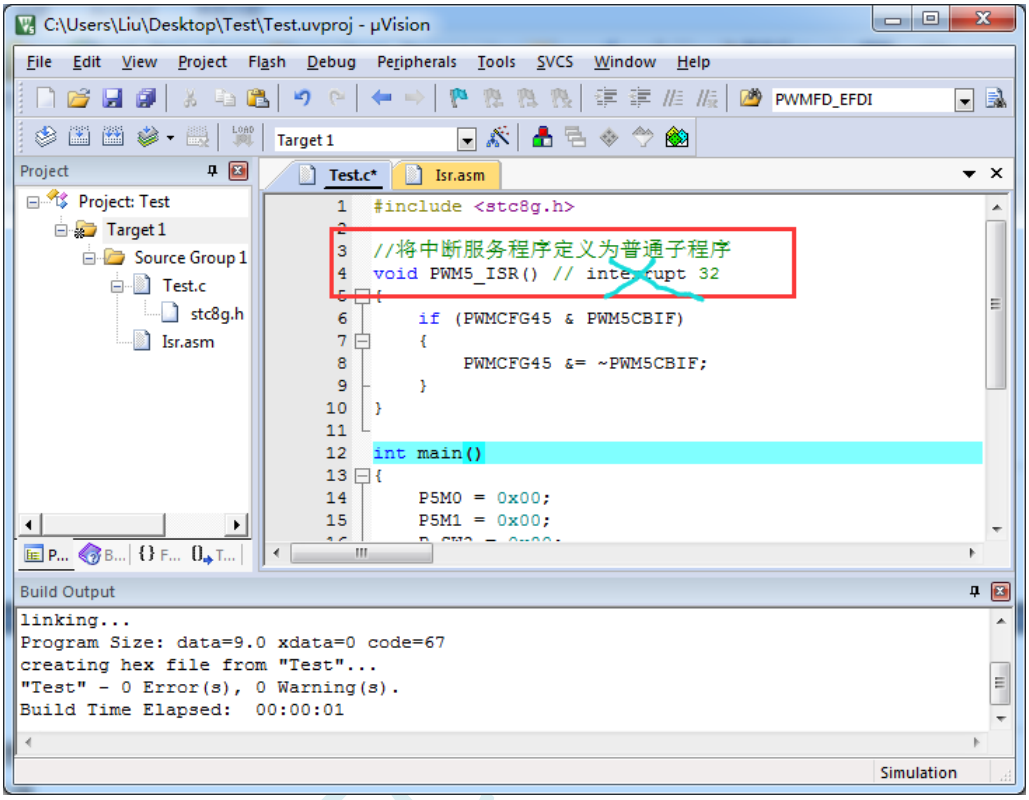


执行效果与方法 1 相同，此方法适用于需要重映射多个中断号大于 31 的情况。

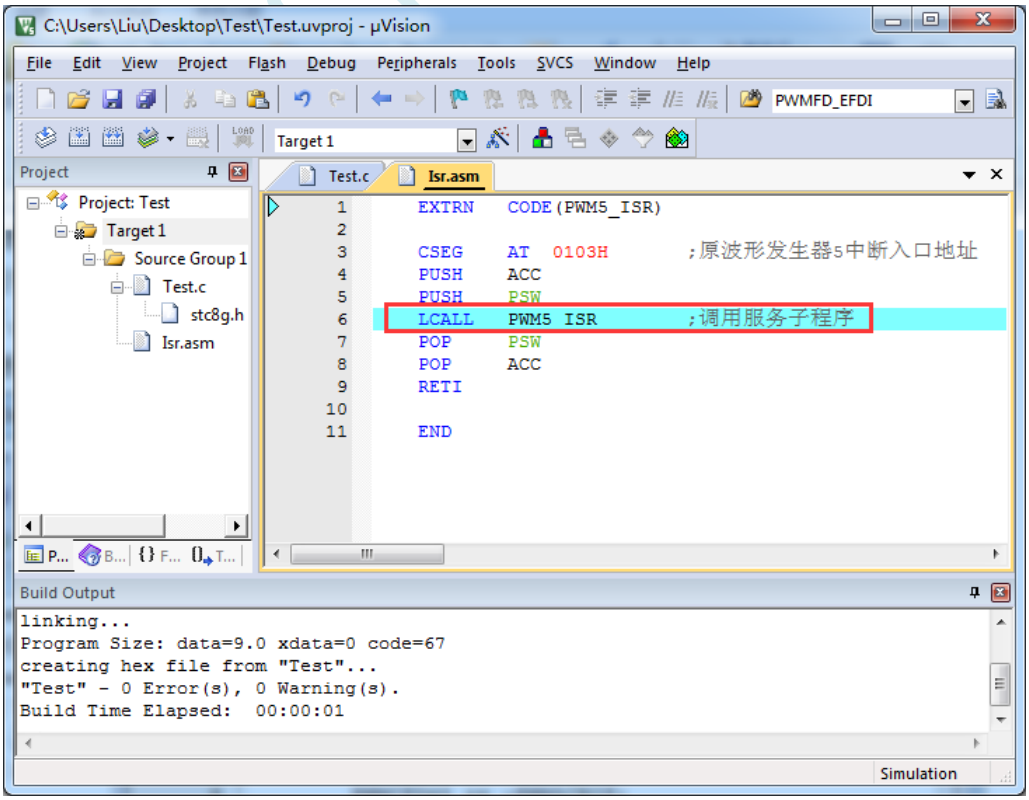
方法 3：将中断服务程序定义成子程序，然后在汇编代码中的中断入口地址中使用 LCALL 指令执行服务程序

操作步骤如下：

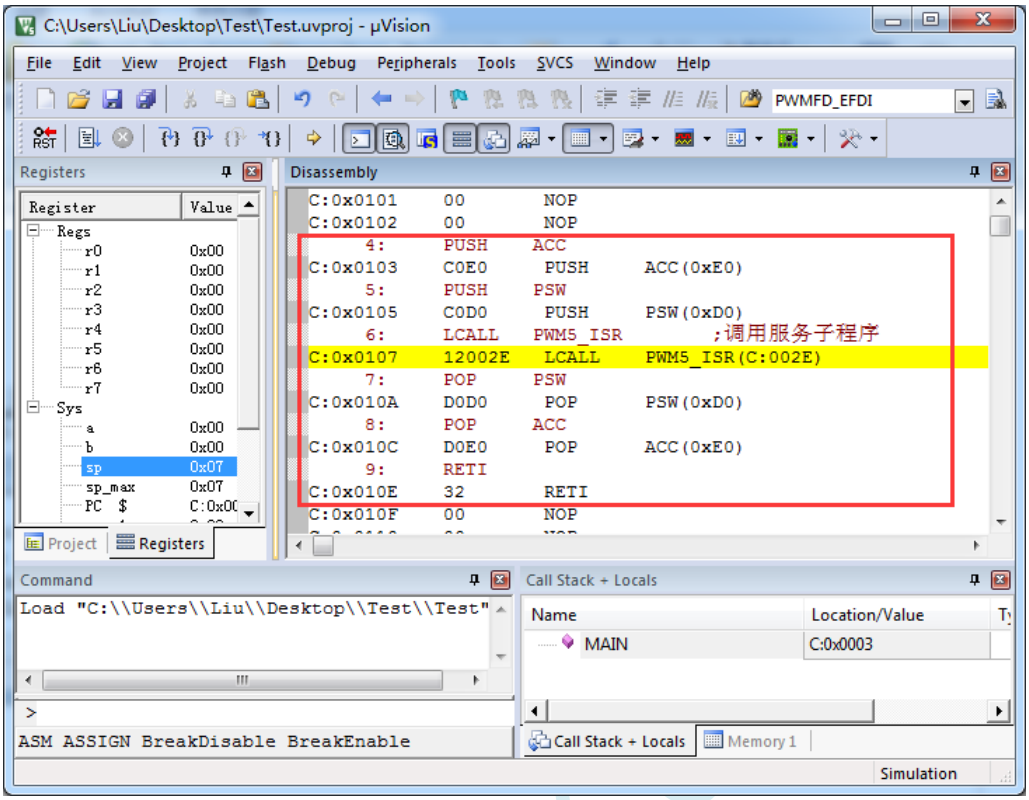
1、首先将中断服务程序去掉“interrupt”属性，定义成普通子程序



2、然后在汇编文件的 0103H 地址输入如下图所示的代码



3、编译通过后，即可发现在 0103H 地址的地方即为中断服务程序

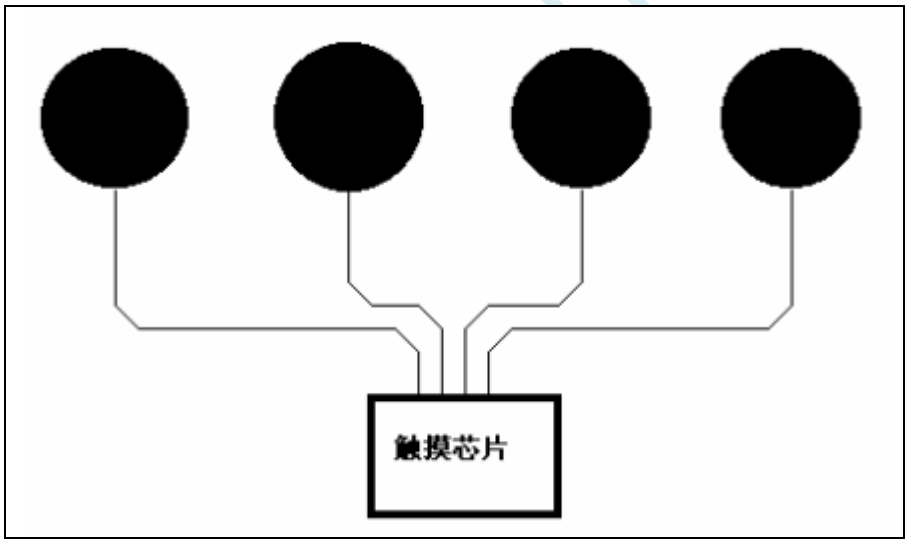


此方法不需要重映射中断入口，不过这种方法有一个问题，在汇编文件中具体需要将哪些寄存器压入堆栈，需要用户查看 C 程序的反汇编代码来确定。一般包括 PSW、ACC、B、DPL、DPH 以及 R0~R7。除 PSW 必须压栈外，其他哪些寄存器在用户子程序中有使用，就必须将哪些寄存器压栈。

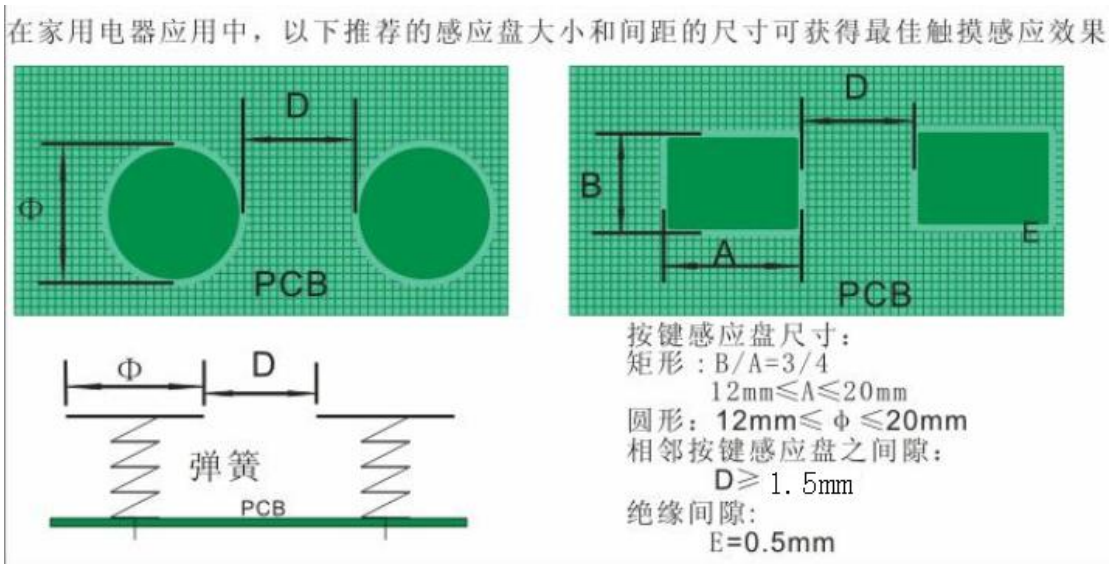
附录V 触摸按键的 PCB 设计指导

触摸按键对 PCB 设计的要求比较严格，否则其效果会大打折扣甚至失败。建议用户在设计 PCB 时遵循以下几点原则：

1. **遵循通常的数模混合电路设计的基本原则**。
   电容式触摸按键模块集成了精密电容测量的模拟电路，因此进行 PCB 设计时应该把它看成一个独立的模拟电路对待。遵循通常的数模混合电路设计的基本原则。
2. **采用星形接地**
   触摸芯片的地线不要和其他电路公用，应该单独连到板子电源输入的接地点，也就是通常说的采用“星形接地”。
3. **电源上产生的噪声对触摸芯片的影响**
   电源纹波、噪声应该尽量小，最好用一根独立的走线从板子的供电点取电并增加滤波措施，不要和其他的电路共用电源回路。
4. **IC 与感应盘的连线尽量等长，让其有近似的分布电容，入下图所示。**



5. **按键感应盘（电容传感器）大小和间隙**
   在满足面板的美学设计要求的情况下，必须通过合理安排的感应盘大小和间隔尺寸，来获得最佳的触摸感应效果。感应盘放在底层，IC 也放在底层，感应盘与 IC 连线不要有过孔。相邻感应盘边沿的间隔最好在 1.5mm 以上（下图中的尺寸 D），如果 PCB 面积允许，尽量取大一些间隔。铺铜与感应盘的间隔为 0.5mm（下图中的尺寸 E）。



6. **铺铜处理**

   底层可以铺网格铜或实铜均可，注意铺铜与感应盘的间隔为 0.5mm。顶层印刷按键的丝印信息，丝印的外框形状与底层感应盘一致，顶层对应底层感应盘的地方不能铺铜，否则会屏蔽掉触摸动作。顶层铺铜与底层铺铜一样即可。

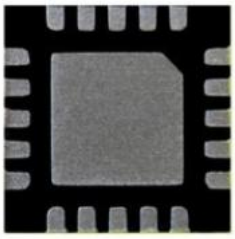
7. **走线处理**

   感应盘与 IC 的连线使用比较小的线宽为好，比如 10~15mil 之间。感应盘到触摸芯片的连线不要跨越强干扰、高频、大电流的线。感应盘到触摸芯片的连线周围 1.5mm 内不要走其他信号线，越远离越好。顶层对应底层感应盘和连接线的地方，最好不要放任何线。

STC MCU

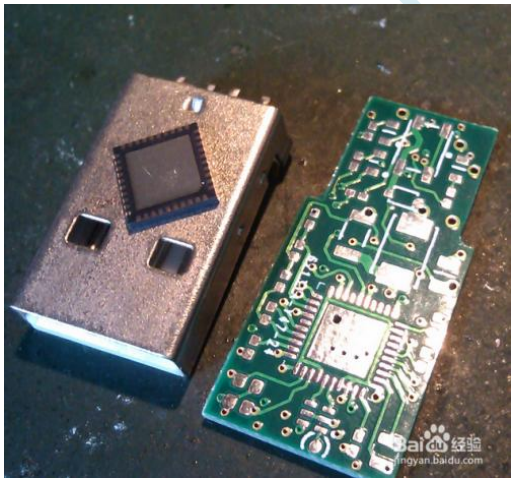
附录W QFN/DFN 封装元器件焊接方法

STC 产品的封装形式中，增加了现在比较流行的 QFN 和 DFN 的封装。由于这种封装形式的芯片芯片的管脚在芯片底部，手工焊接有一定的难度。市面上有专门做工程样品焊接的小公司，可承接工程样品打样。如用户需要自行焊接，可参考下面的焊接方法。



1、 首先需要准备如下工具：电烙铁、热风枪、镊子、固定架等工具

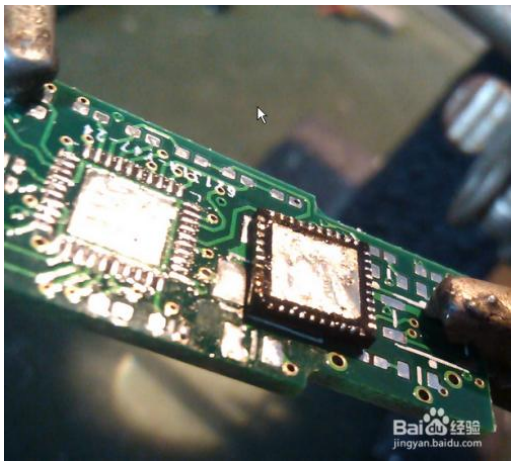
2、 需要焊接的 PCB 板和芯片如下图：



3、 先给板上芯片的焊盘上锡：



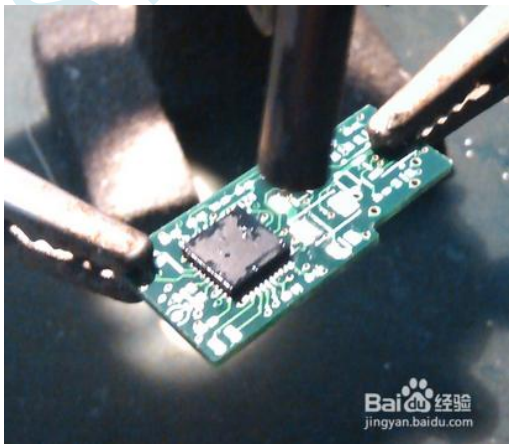
4、 然后给芯片底部上锡，这个上完锡后要弄平，尽量减少锡，但不能没有。



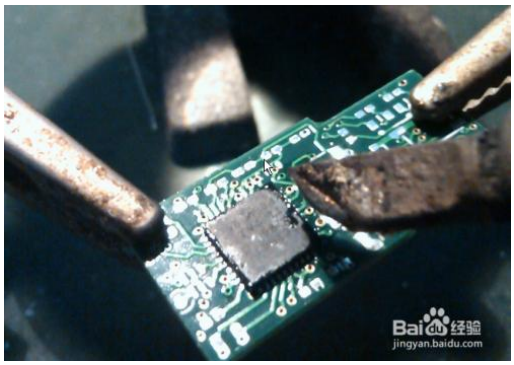
5、 调整热风枪温度，实际出风大概在 240 度左右，因为风枪质量不一样，根据实际情况调节。



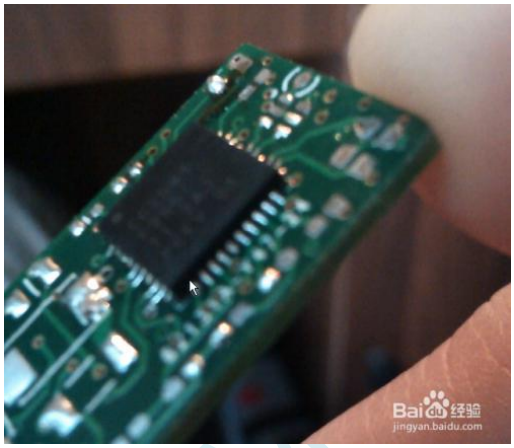
6、 把芯片放到焊盘上，一定要放正，然后用热风枪对着它吹，速度要均匀，直到锡溶化，一般 20 秒内。

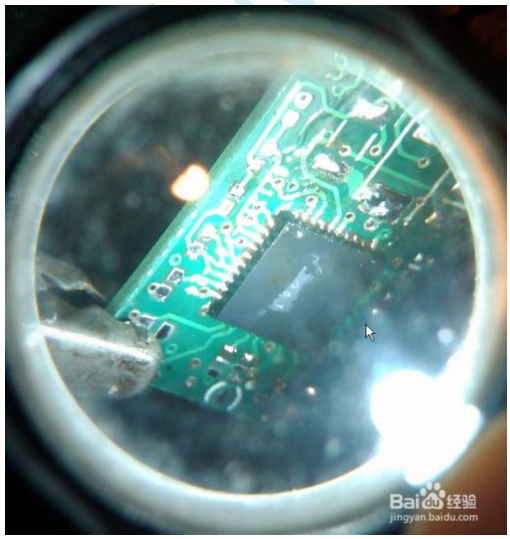


7、 用烙铁给芯片侧引脚上锡


Bai经验
jingyan.baidu.com

8、 焊接完成后的效果


Bai经验
jingyan.baidu.com


Bai经验
jingyan.baidu.com

附录X　可以自己去生产的【USB 转双串口】资料

【USB转双串口】量产PCB/SCH开源，芯片出厂自带USB程序

Ai8H2K12U-45MHz-SOP8，USB转单串口，RMB0.95

Ai8H2K12U-45MHz-SOP16，USB转双串口，RMB1.1

USB插头支持：USB-TypeA、USB-TypeC

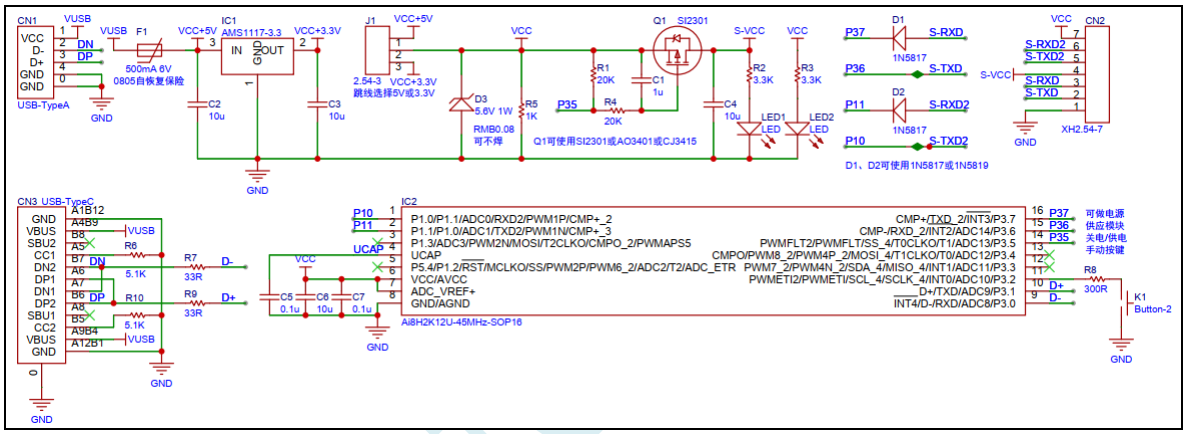
支持任意波特率，最高到10Mbps，程序早已稳定，免驱动安装

全自动停电/上电，ISP下载编程烧录器

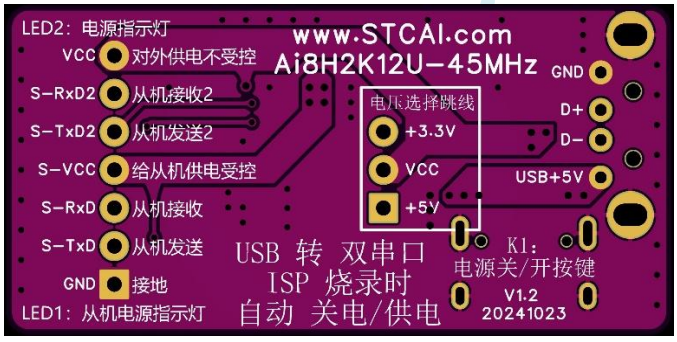
有手动电源开/关按键，可做电源供电模块

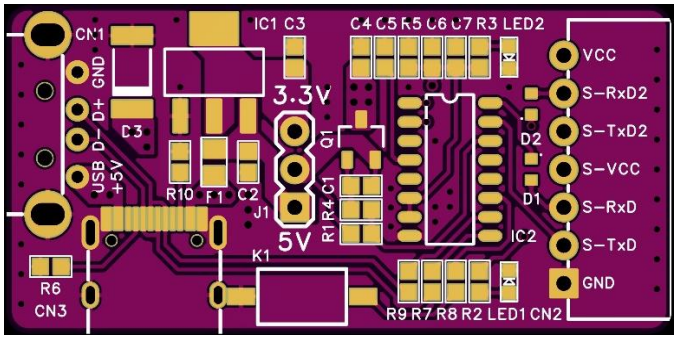
立创开源SCH/PCB，V1.2-20241023：

原理图：



制板量产的PCB链接见下面链接：





附录Y　关于 Keil 软件中 0xFD 问题的说明

众所周知，Keil 软件的 8051 和 80251 编译器的所有版本都有一个叫做 0xFD 的问题，主要表现在字符串中不能含有带 0xFD 编码的汉字，否则 Keil 软件在编译时会跳过 0xFD 而出现乱码。

关于这个问题，Keil 官方的回应是：0xfd、0xfe、0xff 这 3 个字符编码被 Keil 编译器内部使用，所以代码中若包含有 0xfd 的字符串时，0xfd 会被编译器自动跳过。

Keil 官方提供的解决方法：在带有 0xfd 编码的汉字后增加一个 0xfd 即可。例如：

```
printf("数学");                    //Keil 编译后打印会显示乱码
printf("数\xfd 学");               //显示正常
```

这里的“\xfd”是标准 C 代码中的转义字符，“\x”表示其后的 1~2 个字符为 16 进制数。“\xfd”表示将 16 进制数 0xfd 插入到字符串中。

由于“数”的汉字编码是 0xCAFD，Keil 在编译时会将 FD 跳过，而只将 CA 编译到目标文件中，后面通过转义字符手动再补一个 0xfd 到目标文件中，就形成完整的 0xCAFD，从而可正常显示。

关于 0xFD 的补丁网上有很多，基本只对旧版本的 Keil 软件有效。打补丁的方法均是在可执行文件中查找关键代码[80 FB FD]，并修改为[80 FB FF]，这种修改方法查找的关键代码过于简单，很容易修改到其它无关的地方，导致编译出来的目标文件运行时出现莫名其妙的问题。所以，代码中的字符串有包含如下的汉字时，建议使用 Keil 官方提供的解决方法进行解决

GB2312 中，包含 0xfd 编码的汉字如下：

褛饼昌除待谍洱俘庚过糊积箭烬君魁
例笼慢谬凝琵讫驱三升数她听妄锡淆
旋妖引育札正铸　佚冽邶埤荤蔟摭啐
帻猃恺泯潺姬纨琮絮辇挛臊忐睚铨稞
瘕颀螨簸酏鲈鳊鼾

另外，Keil 项目路径名的字符中也不能含有带 0xFD 编码的汉字，否则 Keil 软件会无法正确编译此项目。

附录Z 单片机是否可以提供裸芯

Q：单片机是否可以提供裸芯？

A：暂不提供裸芯。若需要芯片面积小，可使用用 DFN8、QFN20、QFN32、QFN48 等小体积封装

STC MCU

附录AA 官方网址说明

目前 STC 公司的高速、稳定的官方网址为：www.STCAI.com。
而旧的网址 www.STCMCU.com，由于服务器不稳定，网速慢等因素，所以建议大家不要使用。

STC MCU

附录BB STC8 系列命名花絮

STC8A： 字母“A”代表 ADC，是 STC 12 位 ADC 的起航产品

STC8F： 无 ADC、PWM 和 PCA 功能，现 STC8F 的改版芯片与原始的 STC8F 管脚完全兼容，但内部设计进行了优化和更新，用户需要修改程序，所以命名为 STC8C

STC8C： 字母“C”代表改版，是 STC8F 的改版芯片

STC8G： 字母“G”最初是芯片生产时打错字了，后来将错就错，定义 G 系列为“GOOD”系列，STC8G 系列简单易学

STC8H： 字母“H”取自“高”的英文单词“High”的首字母，“高”表示“16 位高级 PWM”

STC MCU

附录CC 更新记录

- **2025/03/04**

1. 增加：下载 Keil 中断号拓展工具，及安装
2. 增加：“关于回流焊前是否要烘烤”中“STCAI 贴片 MCU 最小真空包装”内容
3. 调整部分章节的位置

- **2025/2/14**

1. 增加：Keil 小技巧----在多文件项目的源文件中声明并使用外部变量和外部函数

- **2025/1/24**

1. 修正文档中错别字、笔误、乱码等

- **2024/12/12**

1. 增加：内部高速 IRC 时钟恢复振荡到稳定需要等待的时钟数控制寄存器（IRCDB）

- **2024/11/29**

1. 增加：AIapp-ISP | I/O 口配置工具普通配置模式和高级配置模式
2. 增加：AIapp-ISP | 定时器计算器工具
3. 增加：AIapp-ISP | 串口波特率计算器工具
4. 增加：AIapp-ISP | 串口助手/USB-CDC
5. 更新 STC8C2K64S4 的下载/仿真线路图
6. 更新 STC8C2K64S2 的下载/仿真线路图

- **2024/11/28**

1. 增加：USB-Link1D 支持 脱机下载 说明
2. 增加：USB-Link1D 支持 脱机下载，如何免烧录环节
3. 增加：USB-Writer1A 编程器/烧录器・支持・插在・锁紧座上・烧录
4. 增加：USB-Writer1A 支持 自动烧录机，通信协议和接口
5. 增加：USB-Link1D 工具附送的各种强大的人性化配线使用说明及实际应用
6. 增加：附录--可以自己去生产的【USB 转双串口】资料
7. 更新 ISP 下载相关硬件选项的说明
8. 更新混合电压供电系统 3V/5V 器件 I/O 口互连参考线路图

- **2024/5/31**

1. 更正文档中的笔误

2. 更正文档中 INTCLKO/INT_CLKO 寄存器名称不统一的问题

2024/5/13

1. 更新 USB 转双串口芯片的价格
2. 增加 STC-USB Link1D 工具主控芯片的制作步骤

2024/2/2

1. 增加中断响应说明
2. 增加小商城二维码
3. 在每个管脚图中增加芯片系列名称
4. 整理文档结构，让寄存器说明尽量显示在同一页
5. 更正文档中的笔误
6. 时钟章节增加 MCU 上电工作过程说明

2023/11/24

1. 手册封面增加小商城二维码
2. 单片机系列的每个封装都增加目录标题
3. 更新每个封装图中的工具图片
4. 更新 I/O 基于注意事项说明
5. 增加 USB 转双串口芯片介绍章节
6. 增加使用 STC-USB Link1D 工具和 USB 转双串口工具仿真和下载示意图
7. 更新模拟 USB 下载说明
8. 增加一箭双雕工具使用说明章节
9. 增加编译器介绍和项目设置章节
10. 更加选型价格表

2023/9/12

1. 最小系统的管脚图中增加下载线路图
2. 增加下载流程图说明

2023/1/17

1. 增加“用 STC 一箭双雕之 USB 转双串口仿真 STC8 系列 MCU”章节

2023/1/10

1. 增加论坛链接
2. 更新参考线路图说明

2022/12/23

1. 更新 I2C 从机代码范例程序（提高代码兼容性）

- **2022/11/14**

1. ISP 下载参考线路图中的电容统一建议使用 22uF+0.1uF（104）的组合

- **2022/10/31**

1. 增加使用 STC-USB Link1D 工具进行 ISP 下载的参考线路图
2. 增加软件模拟 USB 进行 ISP 下载的参考线路图

- **2022/9/20**

1. 更新选型价格表
2. 更新官方网址
3. 增加 STC-ISP 高级应用章节
4. 增加关闭驱动程序强制数字签名说明章节

- **2022/8/4**

1. 修正比较器章节的错别字
2. 增加 STC8C2K48S2 型号

- **2022/3/9**

1. 更新特殊功能寄存器列表（STC8C 系列只有 P0IE 和 P1IE）
2. 更正文档中的笔误

- **2021/12/17**

1. 修正定时器 2/3/4 的定时计算公式
2. 更新选型和价格表

- **2021/6/26**

1. 增加定时器 2/3/4 中断标志位的相关说明

- **2021/3/8**

1. 增加 STC8C2K16S2 系列的 TSSOP20 的管脚图和管脚说明
2. 附录中增加 STC8 系列命名花絮

- **2021/2/26**

1. 增加有关模拟 USB 下载的说明
2. 增加定时器 2、定时器 3、定时器 4 的 8 位时钟预分频寄存器说明

STC MCU

- **2021/2/4**

1. 更正 CLKDIV 寄存器的复位初始值
2. 增加特殊功能寄存器初始值说明
3. 在管脚图下增加应用参考线路图

- **2020/12/4**

1. 更正部分范例程序中的错误
2. 增加 STC8C2K64S2 系列
3. 更新 STC8C2K64S4 系列和 STC8C2K64S2 系列的参考价格
4. 更新中断结构图
5. 更正文档中的描述错误
6. 增加串口转 LIN 总线范例程序

- **2020/10/16**

1. 创建 STC8C 系列单片机技术参考手册文档

STC MCU

本系列产品标准销售合同

一．产品质量标准：货物为全新正品。符合 ROHS 质量标准。

二．供方责任：如是供方质量问题，经双方确认后，需方退回芯片，有一换一，质保一年。

三．需方责任：

A、验收：在快递送货到时，需方确认数量无误，无芯片散落，无管脚变形，无其他品质异常情况后再签收。如有异常需方不能签收，由快递公司承担责任。一经需方签收，需方就是认可供方已按要求完成该订单，不再有其他连带责任。

B、保管及贴片加工：根据国际湿敏度 3（MSL3）规范的要求，贴片元器件在拆开真空包装后，168 小时内，7 天内，必须回流焊贴片完成。LQFP/QFN/DFN 托盘能耐 100 度以上的高温，拆开真空包装后 7 天内必须回流焊贴片完成，如未完成，回流焊前必须重新烘烤：110~125℃，4~8 个小时都可以。SOP/TSSOP 塑料管耐不了 100 度以上的高温，拆开真空包装后 7 天内必须回流焊贴片完成，否则回流焊前先去除耐不了 100 度以上高温的塑料管，放到金属托盘中，重新烘烤：110~125℃，4~8 个小时都可以

由于经常有客户退回来的货物中含有来历不明产品，且贴片原器件拆开真空包装后，需要在 168 小时/7 天内完成回流焊贴片工序。

我司无产能对退回器件再进行重新详细检测，再进行重新烘烤，无能力对客户退回的所谓未拆封芯片进行评估，为保证全体客户的利益，产品一经出库，概不退换，以确保品质，确保所有客户的安全。

四．解决纠纷方式：对本合同不详尽之处或产生争议，双方协商解决。协商不成在供方所在地申请仲裁。

五．其他条款：合同一式两份。自双方签署起生效。供方若因外力因素而导致无法交货，供方应及时通知需方，并重新协商本合同相关事宜，需方免除供方应承担的义务。本合同未能列入条款可在合同附件详细列入。

六．本合同双方代表签字且款到后方可生效。

备注：如特殊情况，买方买的型号要更换成其他型号，供方也同意的：

1，开机 13 小时高温烘烤，　1000 元一次

2，开机测试 RMB500 一次，　+0.2 元/片

产　品　授　权　书

致：江苏国芯科技有限公司

深圳国芯人工智能有限公司设计的集成电路产品的知识产权归深圳国芯人工智能有限公司所有。现授权江苏国芯科技有限公司可从事我司产品在中国的推广和销售工作。



授权单位：　深圳国芯人工智能有限公司

授权至有效期：　2030 年 1 月 1 日前

自主产权，生产可控

深圳国芯人工智能有限公司是中华人民共和国大陆独资企业，按中国法律法规独立运营的企业，注册地址在深圳市前海深港合作区前湾一路 1 号 A 栋 201 室。

本手册所描述的器件是在中国境内自主研发，具备独立自主知识产权。

产品核心研发在中国境内，具备芯片设计、封装设计、结构设计、可靠性设计、器件仿真、工艺模拟等全部设计能力；产品核心研发团队人员及带头人全部为我国境内人员组成，其中研发团队带头人研发从业年限十年以上，具备长期、稳定的后续支持能力，具有在我国境内申请的专利证书及软件著作权等。

晶圆制造：本器件设计完成后的晶圆制造加工，在中华人民共和国大陆境内的晶圆厂加工制造完成，受中华人民共和国法律法规管理监管和控制，完全可控。

封装制造：本器件设计完成后的封装制造，在中华人民共和国大陆境内的封装厂加工完成，受中华人民共和国法律法规管理监管和控制，完全可控。

测试：本器件设计完成后的测试，在中华人民共和国大陆境内测试完成，受中华人民共和国法律法规管理监管和控制，完全可控。

本器件全部关键工艺均在我国自有生产线上完成，可以长期供货，无被断供的困扰。

特此说明。

深圳国芯人工智能有限公司