# 附录I　STC 仿真使用说明书

## I.1　概述

STC8G/8H 系列单片机均支持在线仿真。支持包括下载用户代码、芯片复位、全速运行、单步运行、设置断点（理论断点个数为无限个，但为了提高仿真效率，目前限制为最多 20 个断点）、查看变量等基本的仿真操作，方便用户调试代码，查找代码中的逻辑错误，进而缩短项目开发周期。

仿真接口可为 USB 或者串口，单片机本身就是仿真器，不需要额外的仿真器即可实现全部的仿真功能。相应的 USB 口或者串口本为仿真专用端口，但当关闭仿真功能后，用户将可随意将仿真接口当作 GPIO、USB 或者串口进行使用。

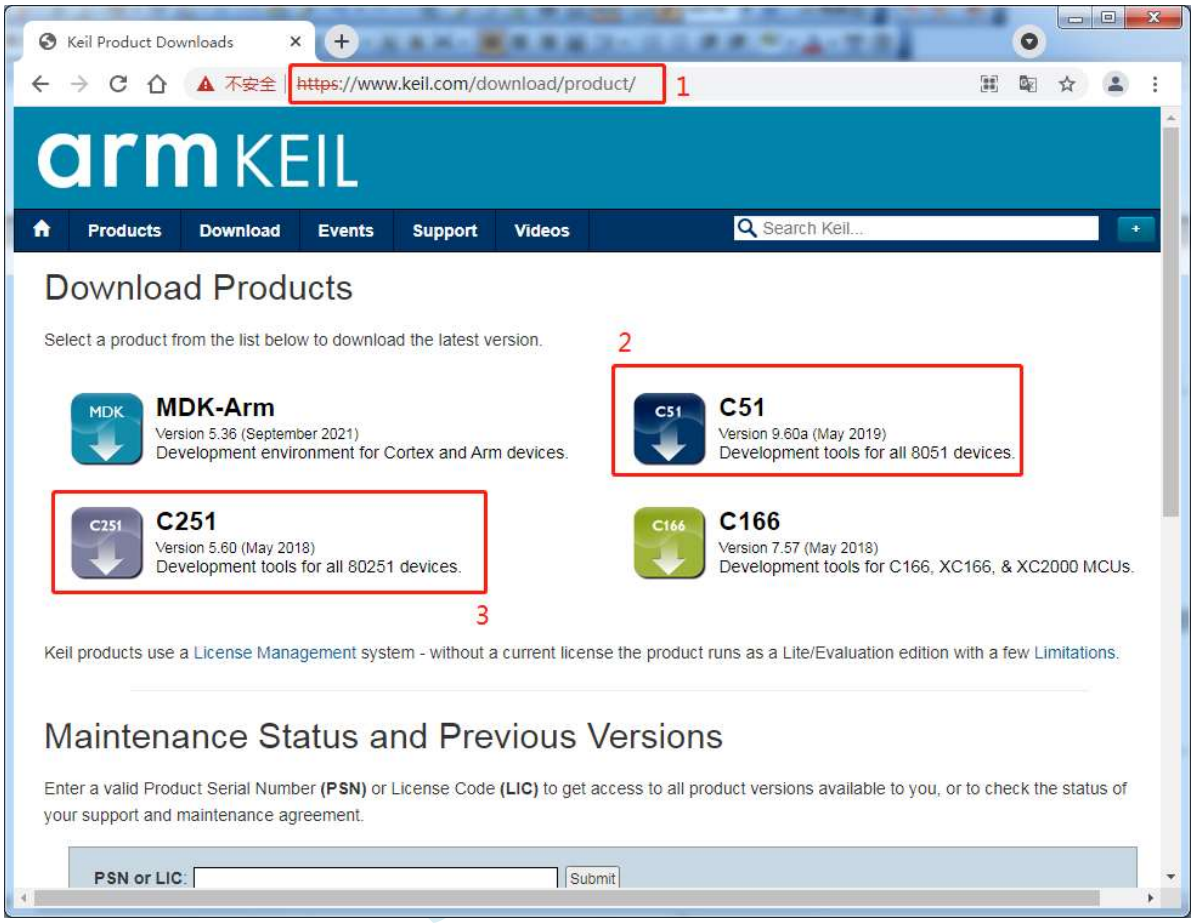
目前所有单片机的仿真模式均为软件监控仿真模式，会占用系统部分资源，各系列单片机仿真时所占用的资源如下表所示：

| 单片机系列 | 仿真接口 | 占用资源 | | |
|---|---|---|---|---|
| | | 端口 | 数据存储器（XDATA） | 程序存储器 |
| STC8H8K64U 系列-B 版本 | USB | D+，D- | 768 字节（1D00H-1FFFH） | 0 字节 |
| | 串口 | P3.0，P3.1 | 768 字节（1D00H-1FFFH） | 0 字节 |
| | | P3.6，P3.7 | | |
| | | P1.6，P1.7 | | |
| | | P4.3，P4.4 | | |
| STC8H8K64U 系列-A 版本 | 串口 | P3.0，P3.1 | 768 字节（1D00H-1FFFH） | 0 字节 |
| | | P3.6，P3.7 | | |
| | | P1.6，P1.7 | | |
| | | P4.3，P4.4 | | |
| STC8H4K64T 系列 | 串口 | P3.0，P3.1 | 768 字节（0D00H-0FFFH） | 0 字节 |
| STC8H3K64S4 系列 | 串口 | P3.0，P3.1 | 768 字节（0900H-0BFFH） | 0 字节 |
| STC8H1K16 系列 | 串口 | P3.0，P3.1 | 768 字节（0100H-03FFH） | 0 字节 |
| STC8H1K08 系列 | 串口 | P3.0，P3.1 | 768 字节（0100H-03FFH） | 0 字节 |
| STC8G2K64S4 系列 | 串口 | P3.0，P3.1 | 768 字节（0500H-07FFH） | 0 字节 |
| STC8G1K08 系列 | 串口 | P3.0，P3.1 | 768 字节（0100H-03FFH） | 0 字节 |
| STC8C2K64S4 系列 | 串口 | P3.0，P3.1 | 768 字节（0500H-07FFH） | 0 字节 |
| STC8A8K64D4 系列 | 串口 | P3.0，P3.1 | 768 字节（1D00H-1FFFH） | 0 字节 |
| STC8A8K64S4A12 系列 | 串口 | P3.0，P3.1 | 768 字节（1D00H-1FFFH） | 0 字节 |
| STC8F2K64S4 系列 | 串口 | P3.0，P3.1 | 768 字节（0500H-07FFH） | 0 字节 |
| STC8F1K08S2 系列 | 串口 | P3.0，P3.1 | 768 字节（0100H-03FFH） | 0 字节 |
| IAP15W4K58S4 | 串口 | P3.0，P3.1 | 768 字节（0D00H-0FFFH） | 0 字节 |
| IAP15F2K61S2 | 串口 | P3.0，P3.1 | 768 字节（0500H-07FFH） | 0 字节 |

## I.2 安装 Keil 软件

STC 单片机的仿真基于 Keil 开发环境，所以在进行仿真前，必须先安装 Keil 软件。

可在下图所示的地址下载 C51 和 C251 开发包



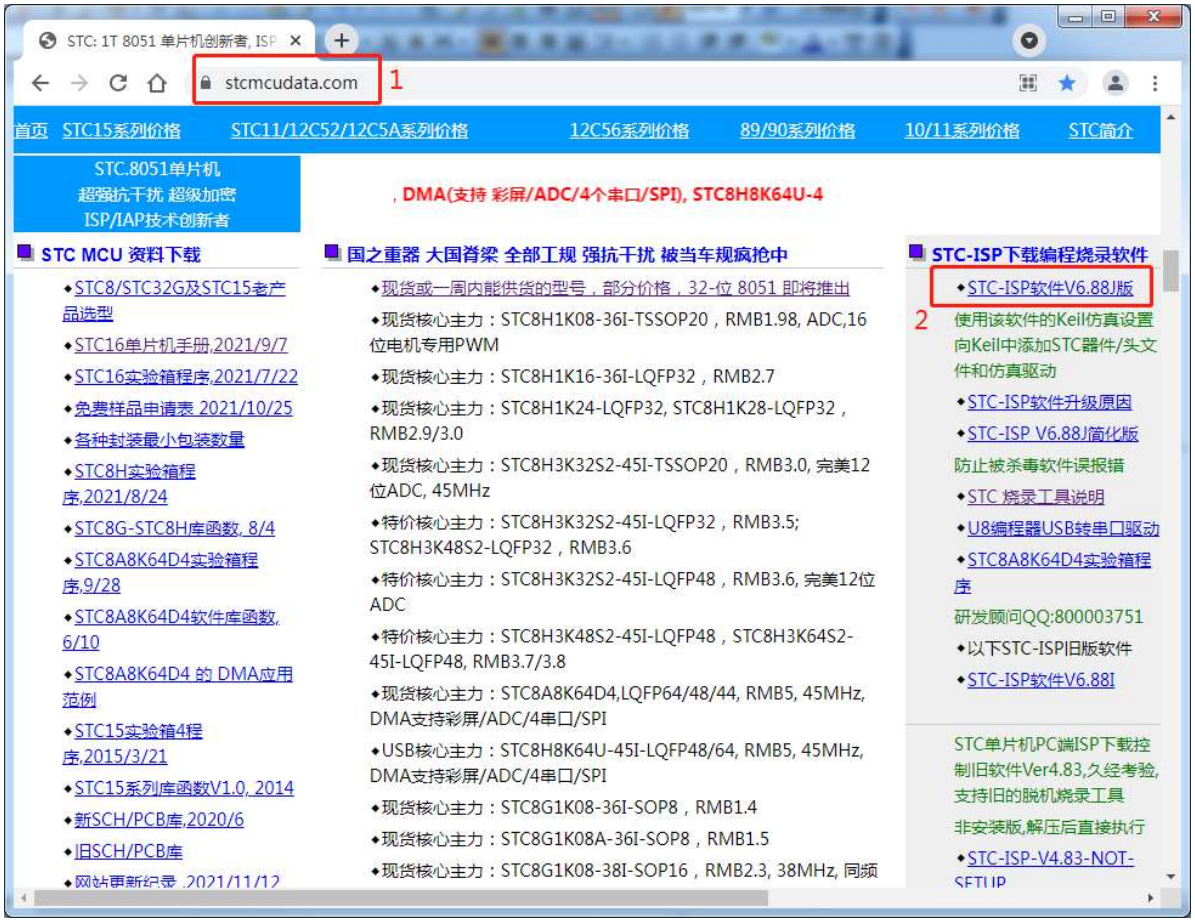

**注意：最新的 Keil-UV5 软件默认是不包含 8051 和 80251 的工具包的，必须手动下载并安装。**

## I.3 安装仿真驱动

安装完成 Keil 开发环境后，还需要安装 STC 专用仿真驱动程序。

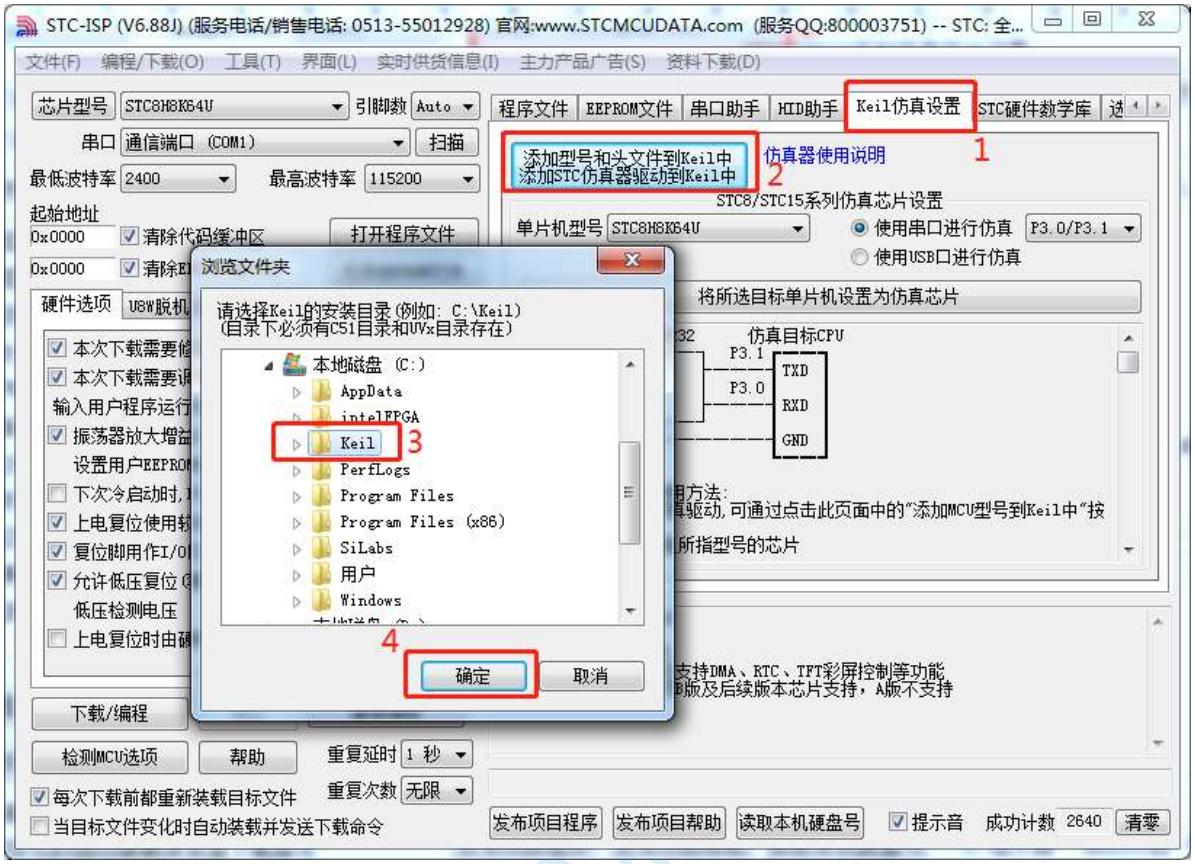
步骤如下：

首先从 STC 官网下载最新的 STC-ISP 下载软件

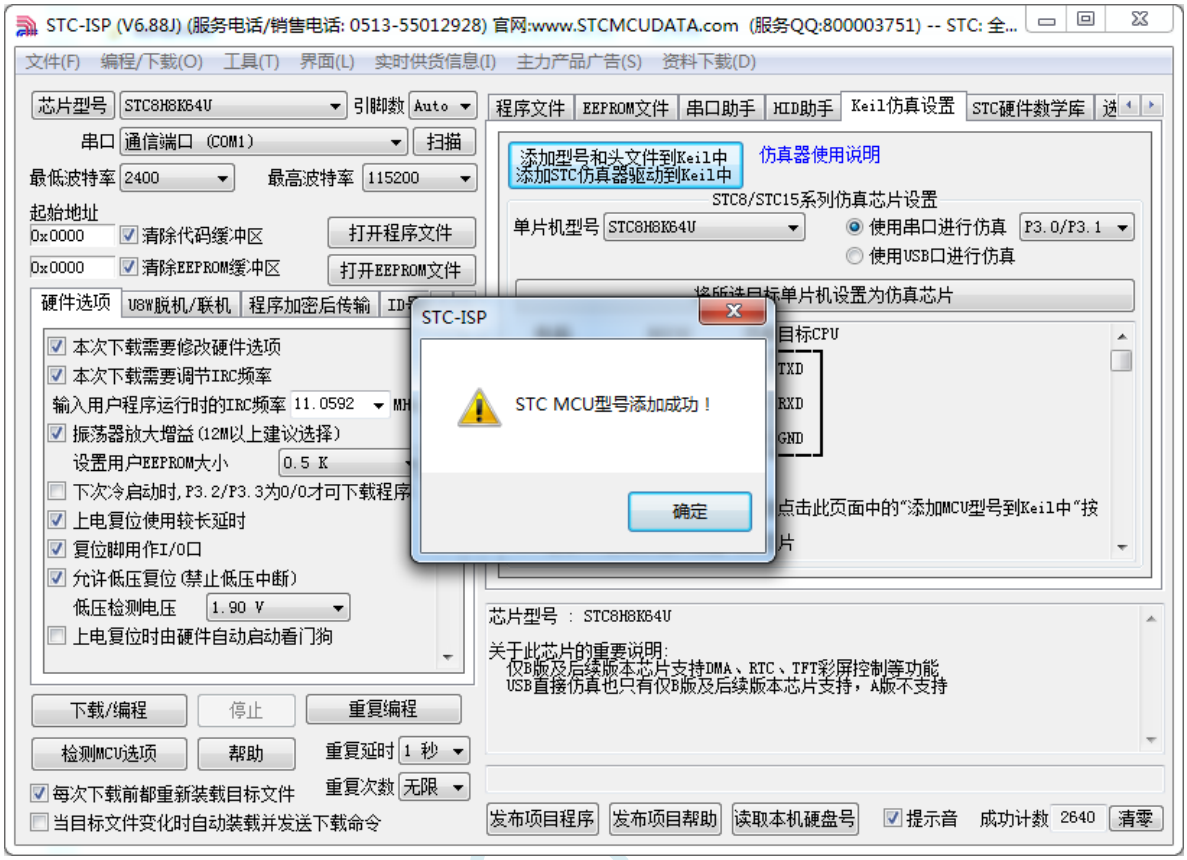



下载并解压完成后，打开软件包中的“stc-isp-vxx.exe”可执行文件


点击下载软件“Keil 仿真设置”页面中的“添加型号和头文件...”按钮（如下图“2”）


STC-ISP (V6.88J) (服务电话/销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (服务QQ:800003751) -- STC: 全...
文件(F) 编程/下载(O) 工具(T) 界面(L) 实时供货信息(I) 主力产品广告(S) 资料下载(D)
芯片型号 STC8H8K64U 引脚数 Auto
串口 通信端口 (COM1) 扫描
最低波特率 2400 最高波特率 115200
起始地址
0x0000 清除代码缓冲区 打开程序文件
0x0000
硬件选项 U8W脱机
本次下载需要修
本次下载需要调
输入用户程序运行
振荡器放大增益
设置用户EEPROM
下次冷启动时,
上电复位使用较
复位脚用作I/O
允许低压复位
低压检测电压
上电复位时由硬
下载/编程
检测MCU选项 帮助
每次下载前都重新装载目标文件
当目标文件变化时自动装载并发送下载命令
重复延时 1 秒
重复次数 无限
程序文件 EEPROM文件 串口助手 HID助手 Keil仿真设置 STC硬件数学库
添加型号和头文件到Keil中
添加STC仿真器驱动到Keil中
仿真器使用说明
1
2
STC8/STC15系列仿真芯片设置
单片机型号 STC8H8K64U
使用串口进行仿真 P3.0/P3.1
使用USB口进行仿真
将所选目标单片机设置为仿真芯片
仿真目标CPU
P3.1
P3.0
TXD
RXD
GND
用方法:
真驱动,可通过点击此页面中的"添加MCU型号到Keil中"按
所指型号的芯片
支持DMA、RTC、TFT彩屏控制等功能
B版及后续版本芯片支持,A版不支持
发布项目程序 发布项目帮助 读取本机硬盘号 提示音 成功计数 2640 清零
浏览文件夹
请选择Keil的安装目录(例如: C:\Keil)
(目录下必须有C51目录和UVx目录存在)
本地磁盘 (C:)
AppData
intelFPGA
Keil
3
PerfLogs
Program Files
Program Files (x86)
SiLabs
用户
Windows
4
确定
取消

在弹出的“浏览文件夹”窗口中，选中 Keil 的安装目录（一般 Keil 的安装目录为“c:\keil”），点击确定后，若弹出“STC MCU 型号添加成功”则表示驱动已安装完成。



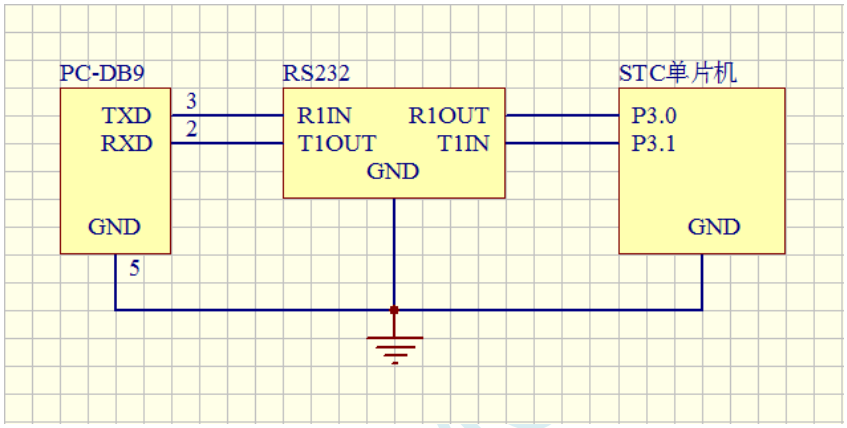
# I.4 串口直接仿真

## I.4.1 制作串口仿真芯片

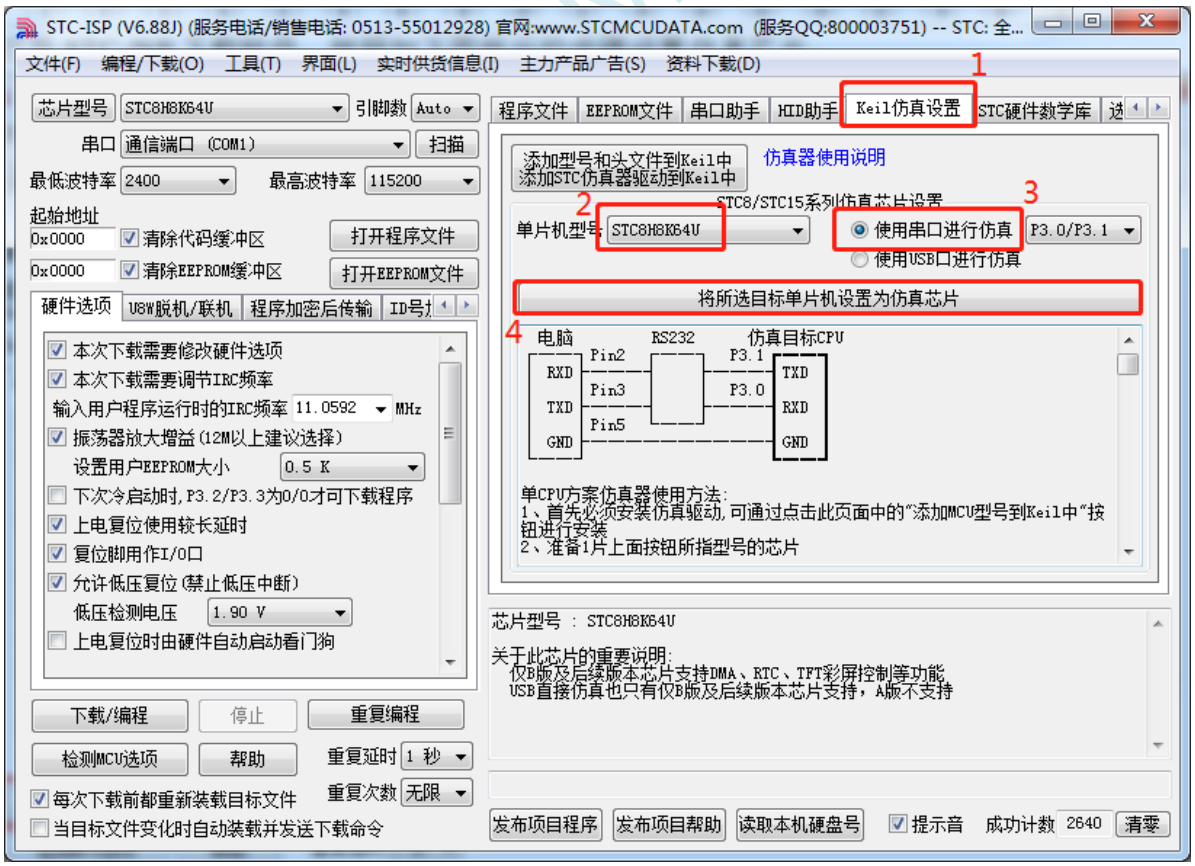
STC 单片机出厂时，仿真功能默认是关闭的，若要使用仿真功能，则需使用 STC-ISP 下载软件将目标单片机设置为仿真芯片。

设置步骤如下：

首先将目标芯片如下图所示的方式和电脑的串口连接在一起，并将单片机断电




打开 STC-ISP 下载软件，按照如下图所示的步骤设置仿真芯片

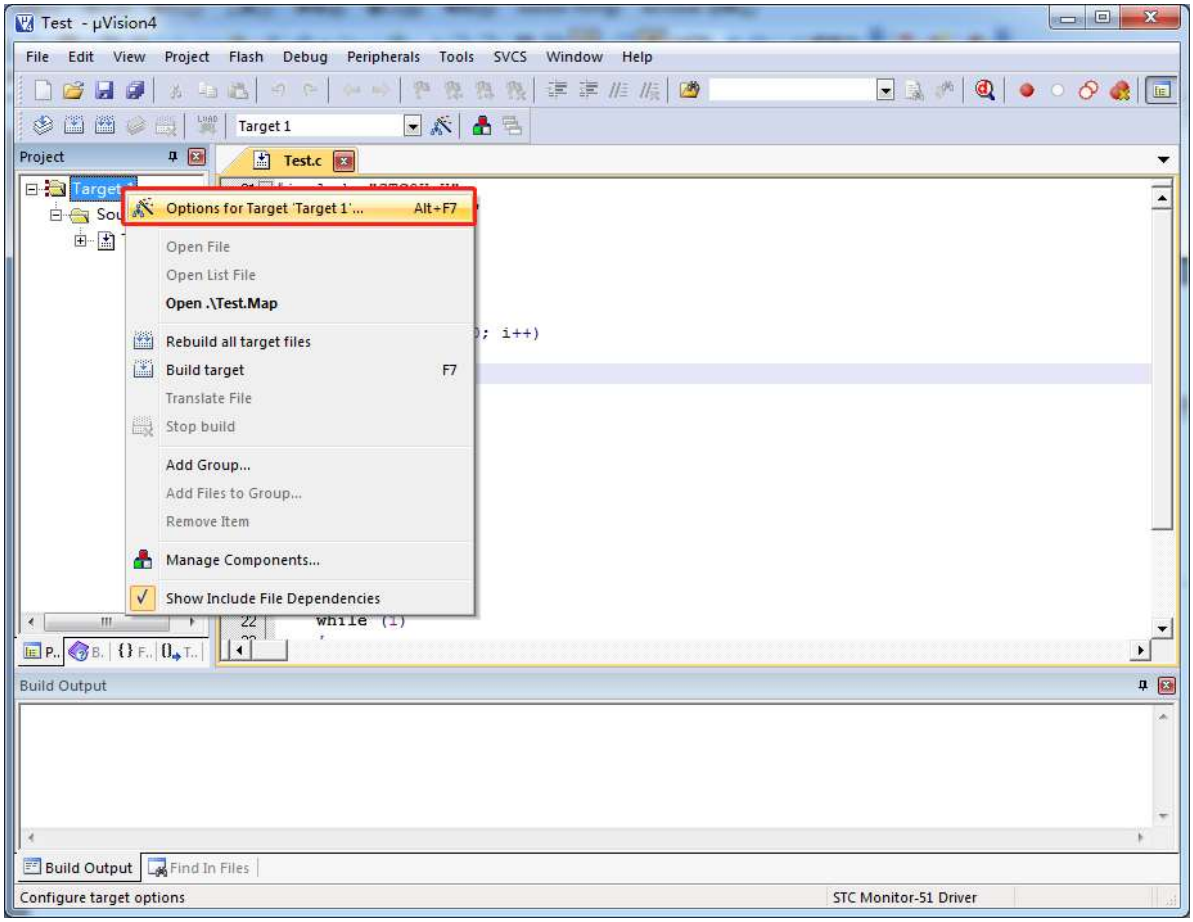


当出现如下画面时，再给单片机上电

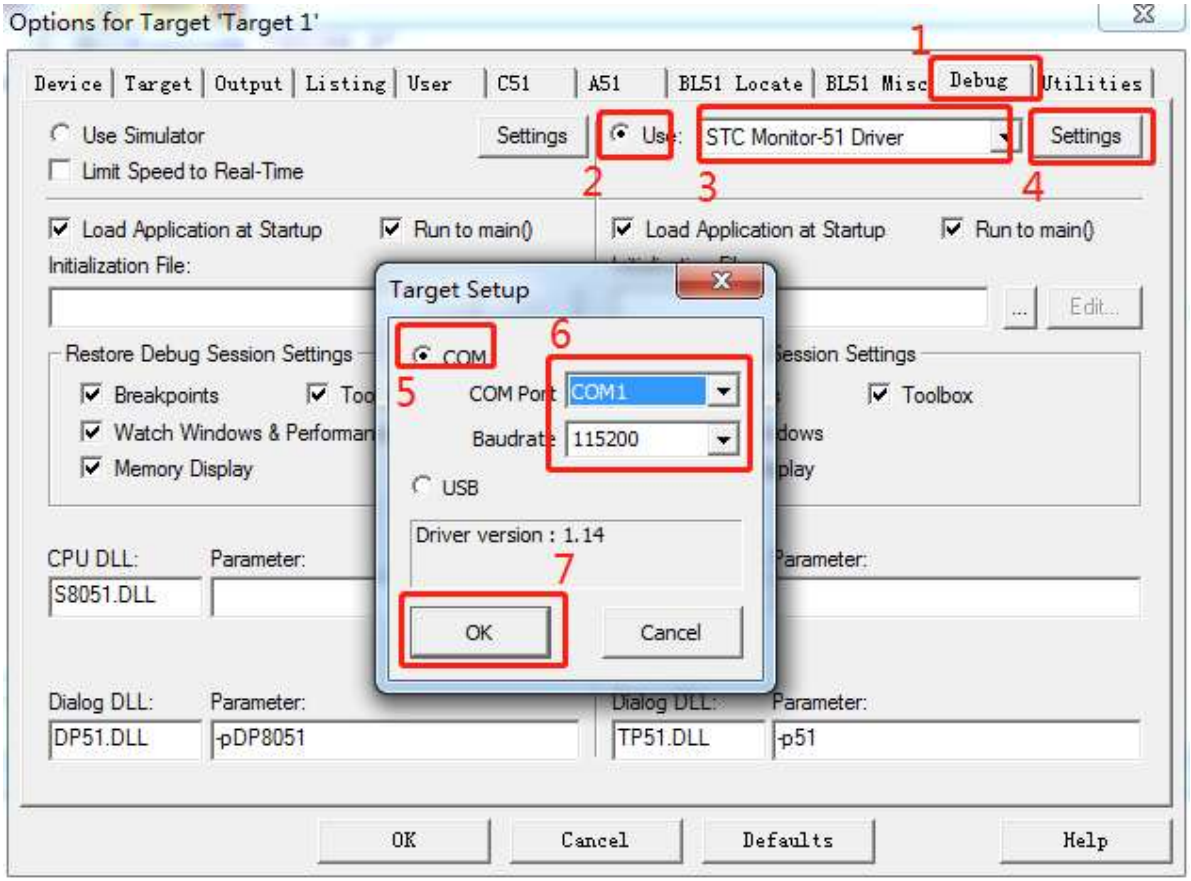
下载完成后，仿真芯片即制作完成

## I.4.2 在 Keil 软件中进行串口仿真设置

在 Keil 软件中打开项目文件，并在下图所示的右键菜单中点击“Options for ...”



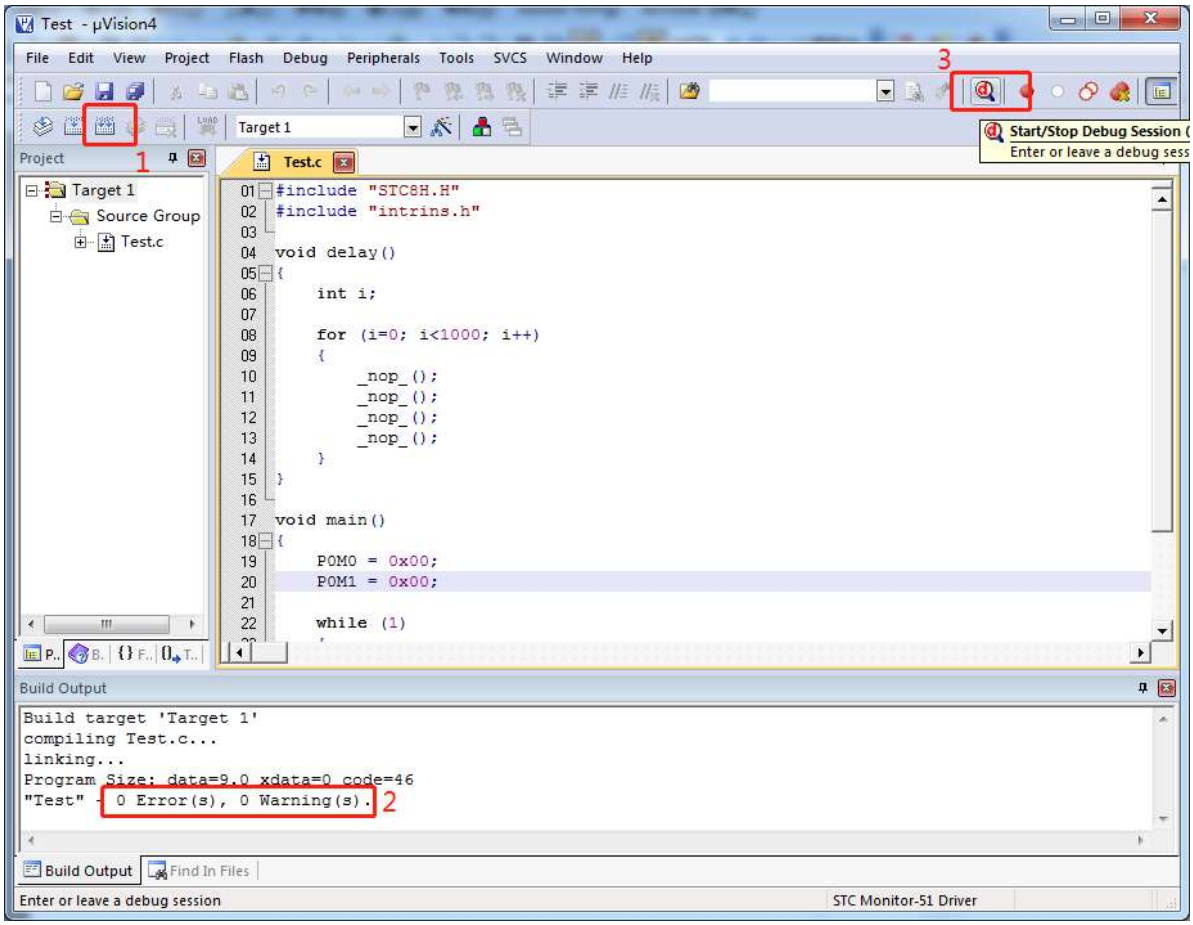
在项目选项中，按如下图所示的步骤进行串口仿真设置

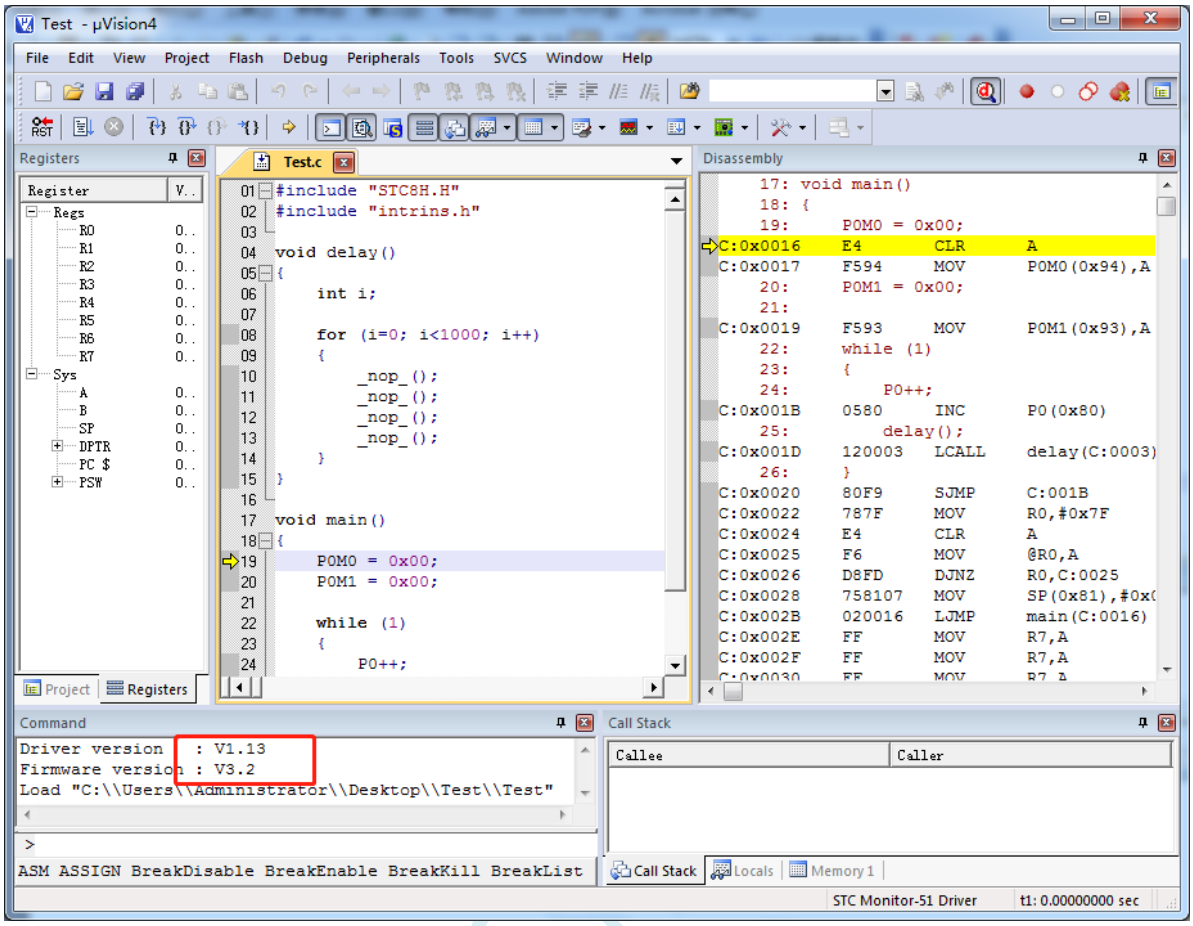



注意：串口请根据实际的连接进行选择，波特率一般选择 115200

## I.4.3 在 Keil 软件中使用串口进行仿真

在 Keil 环境下，编辑完成源代码，并编译无误后，即可开始仿真






若芯片制作和连接均无误，则会如上图所示显示仿真驱动版本，并可正确下载用户代码到单片机，接下来便可进行运行、单步、断点等调试功能了。

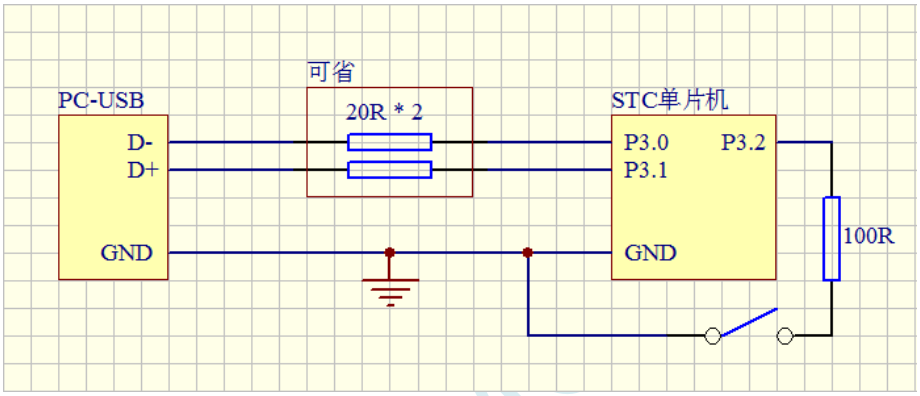
# I.5 USB 直接仿真（目前只有 STC8H8K64U-B 版本芯片支持）

## I.5.1 制作 USB 仿真芯片

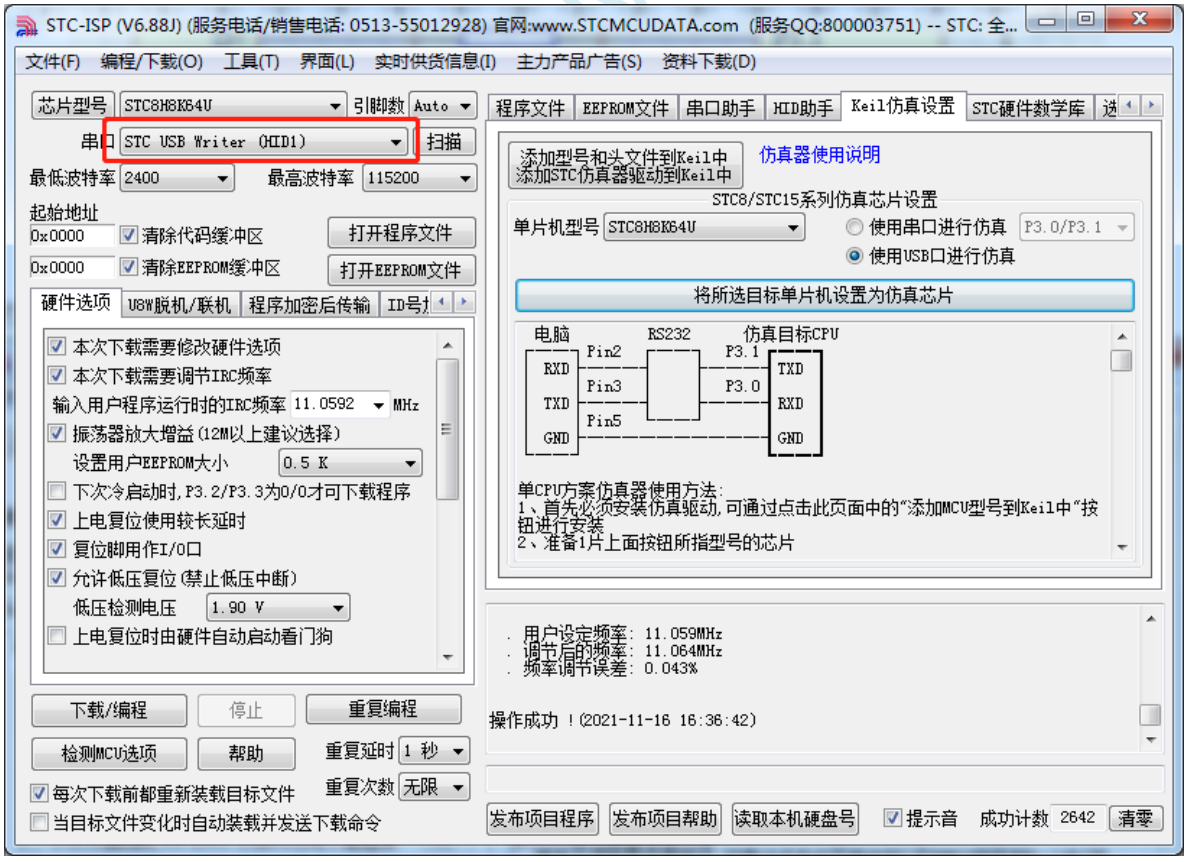
制作 USB 仿真芯片，可按照 4.1 小节的步骤，使用串口 ISP 制作，也可以使用 USB-ISP 的方法制作，本小节将介绍如何使用 USB-ISP 制作。

设置步骤如下：

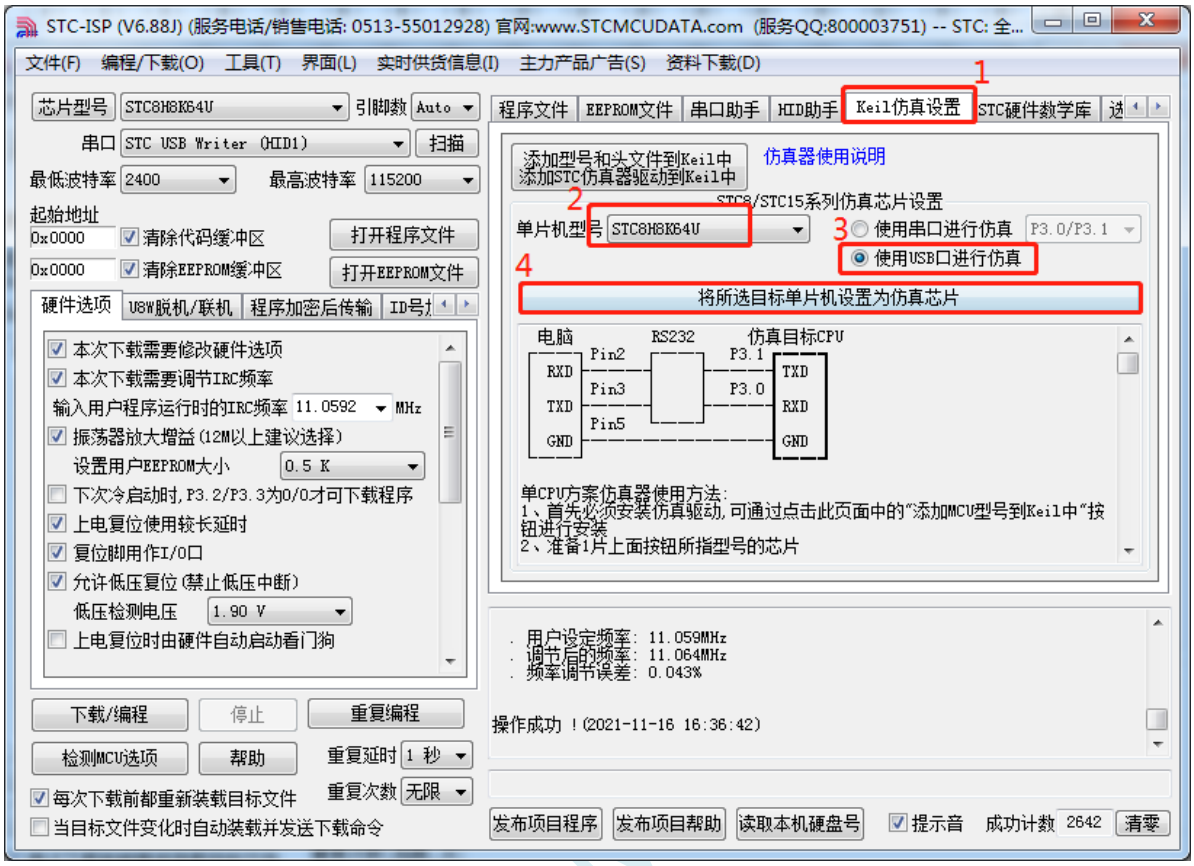
首先将目标芯片如下图所示的方式和电脑的串口连接在一起，并将 P3.2 短路通过开关连接到 GND，然后给单片机上电




若在 ISP 软件中能自动扫描到“STC USB Writer (HID1)”表示连接正确



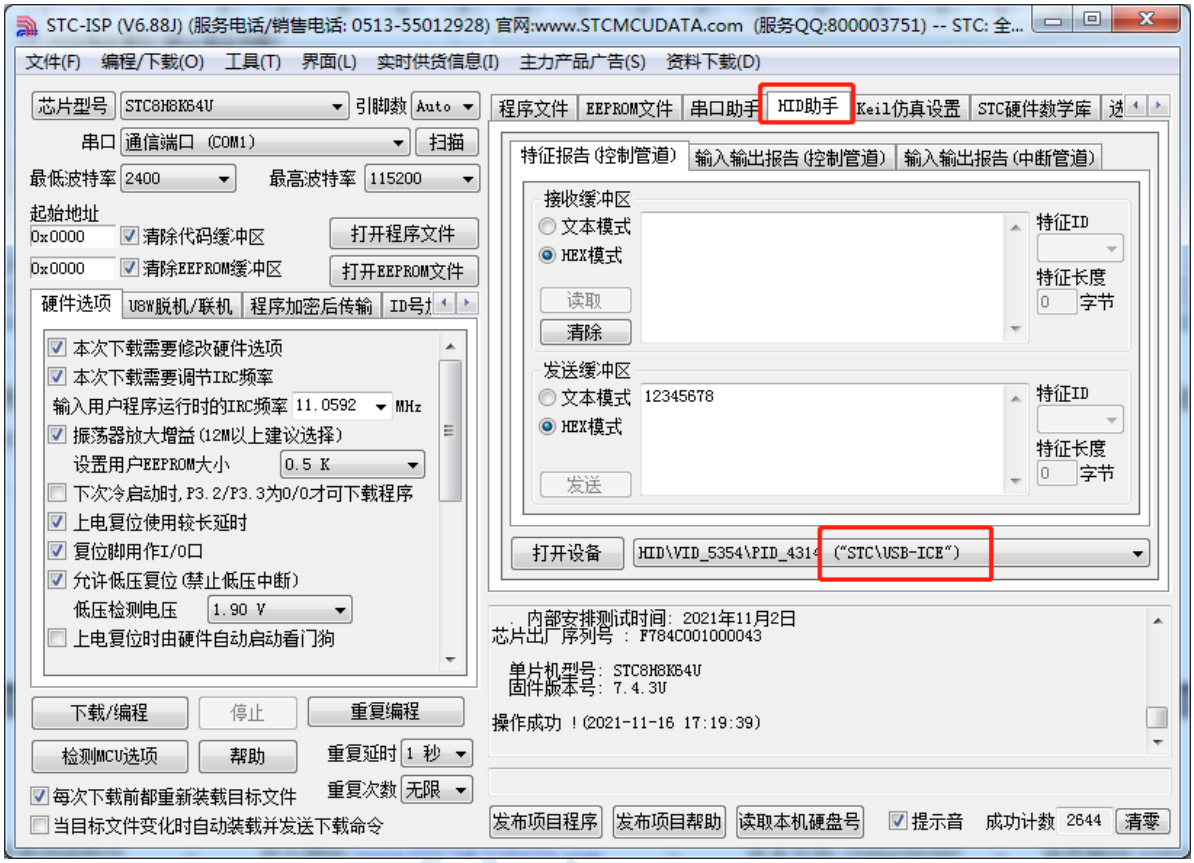
接下来在 STC-ISP 下载软件中，按照如下图所示的步骤设置仿真芯片



下载完成后如下图所示

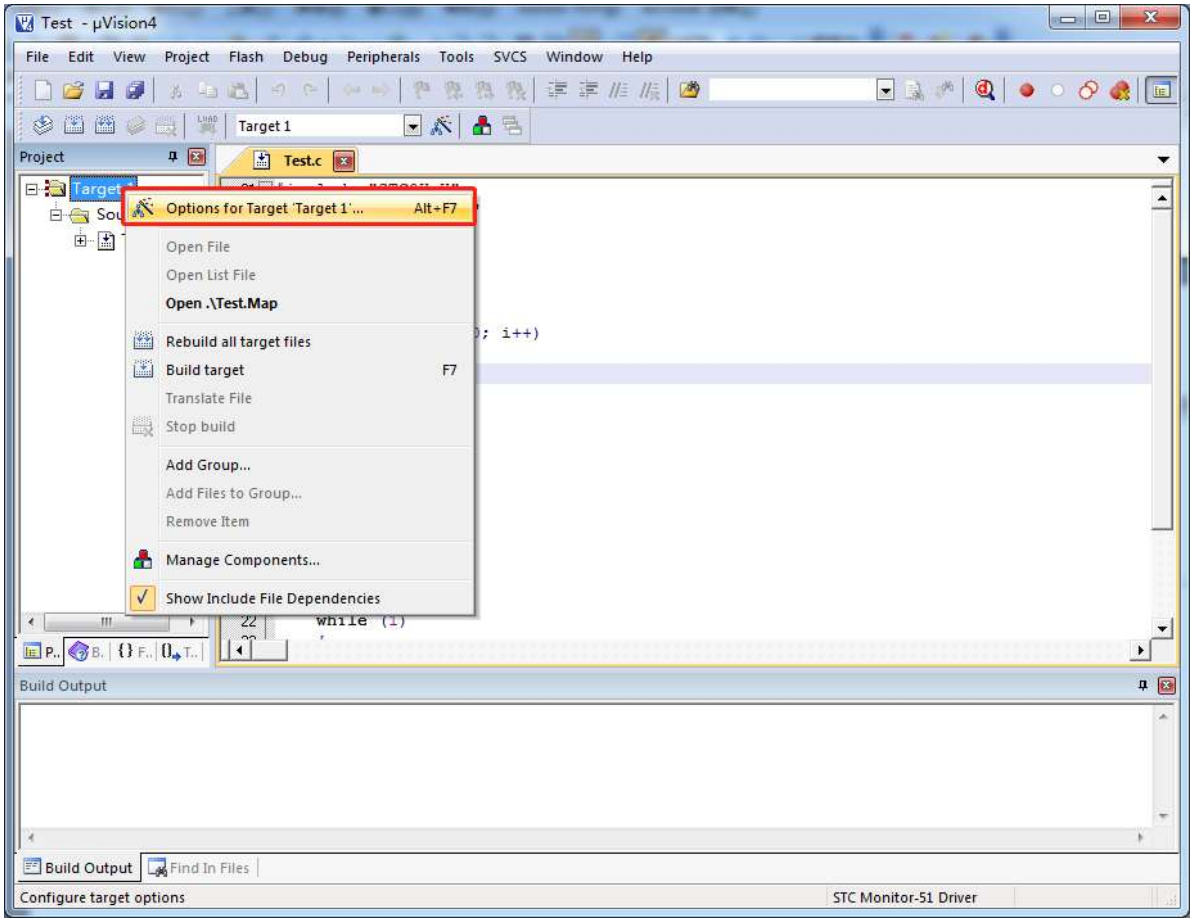

STC-ISP (V6.88J) (服务电话/销售电话: 0513-55012928) 官网:www.STCMCUDATA.com (服务QQ:800003751) -- STC: 全...
文件(F) 编程/下载(O) 工具(T) 界面(L) 实时供货信息(I) 主力产品广告(S) 资料下载(D)
芯片型号 STC8H8K64U 引脚数 Auto
串口 通信端口 (COM1) 扫描
最低波特率 2400 最高波特率 115200
起始地址
0x0000 清除代码缓冲区 打开程序文件
0x0000 清除EEPROM缓冲区 打开EEPROM文件
硬件选项 U8W脱机/联机 程序加密后传输 ID号
本次下载需要修改硬件选项
本次下载需要调节IRC频率
输入用户程序运行时的IRC频率 11.0592 MHz
振荡器放大增益(12M以上建议选择)
设置用户EEPROM大小 0.5 K
下次冷启动时, P3.2/P3.3为0/0才可下载程序
上电复位使用较长延时
复位脚用作I/O口
允许低压复位(禁止低压中断)
低压检测电压 1.90 V
上电复位时由硬件自动启动看门狗
下载/编程 停止 重复编程
检测MCU选项 帮助 重复延时 1 秒
每次下载前都重新装载目标文件 重复次数 无限
当目标文件变化时自动装载并发送下载命令
程序文件 EEPROM文件 串口助手 HID助手 Keil仿真设置 STC硬件数学库
添加型号和头文件到Keil中
添加STC仿真器驱动到Keil中
仿真器使用说明
STC8/STC15系列仿真芯片设置
单片机型号 STC8H8K64U
使用串口进行仿真 P3.0/P3.1
使用USB口进行仿真
将所选目标单片机设置为仿真芯片
电脑 RS232 仿真目标CPU
RXD Pin2 P3.1 TXD
TXD Pin3 P3.0 RXD
GND Pin5 GND
单CPU方案仿真器使用方法:
1、首先必须安装仿真驱动,可通过点击此页面中的"添加MCU型号到Keil中"按钮进行安装
2、准备1片上面按钮所指型号的芯片
内部安排测试时间: 2021年11月2日
芯片出厂序列号 : F784C001000043
单片机型号: STC8H8K64U
固件版本号: 7.4.3U
操作成功 !(2021-11-16 17:18:36)
发布项目程序 发布项目帮助 读取本机硬盘号 提示音 成功计数 2643 清零

制作完成后，需要将 P3.2 口的接地开关断开，并重新对单片机上电，若在下载软件的中“HID 助手”中能检测到“STC\USB-ICE”设备，则表示 USB 仿真芯片制作成功

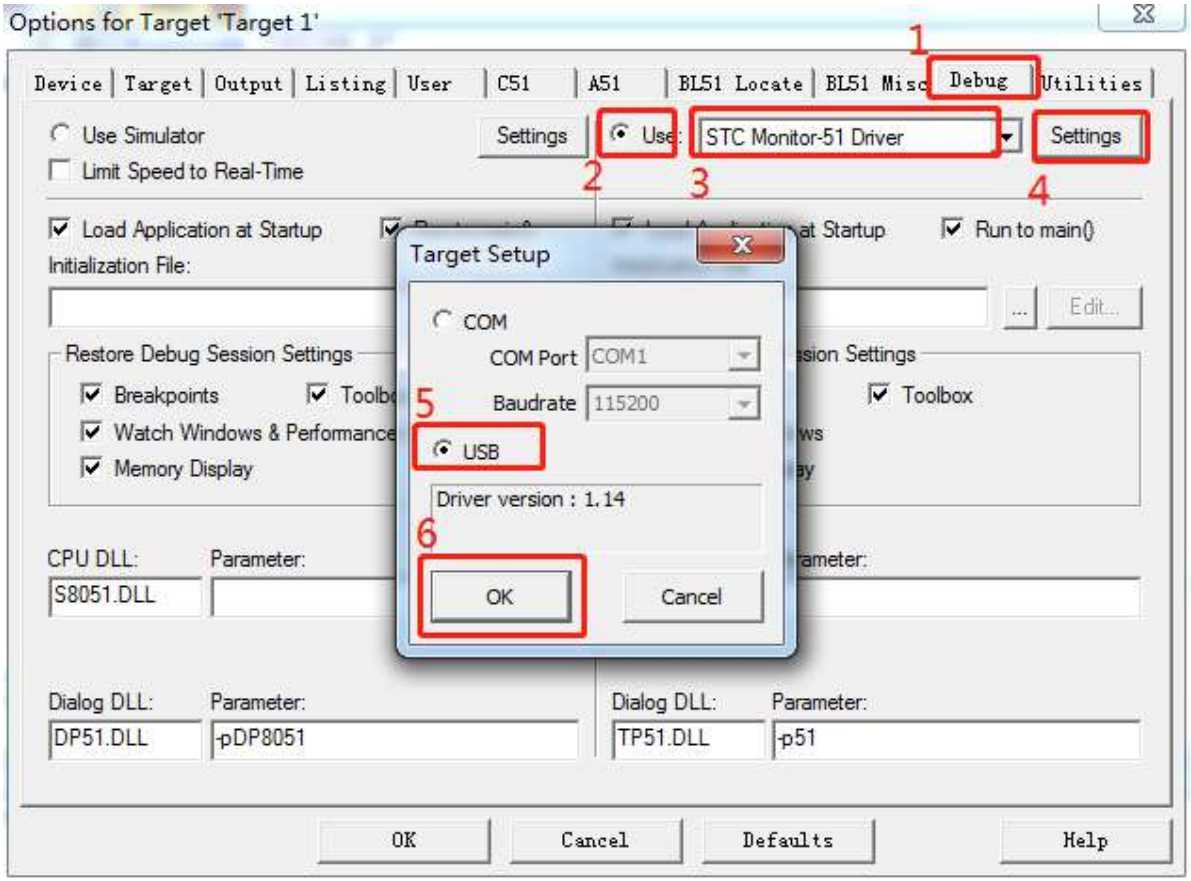


## I.5.2 在 Keil 软件中进行 USB 仿真设置

在 Keil 软件中打开项目文件，并在下图所示的右键菜单中点击“Options for ...”

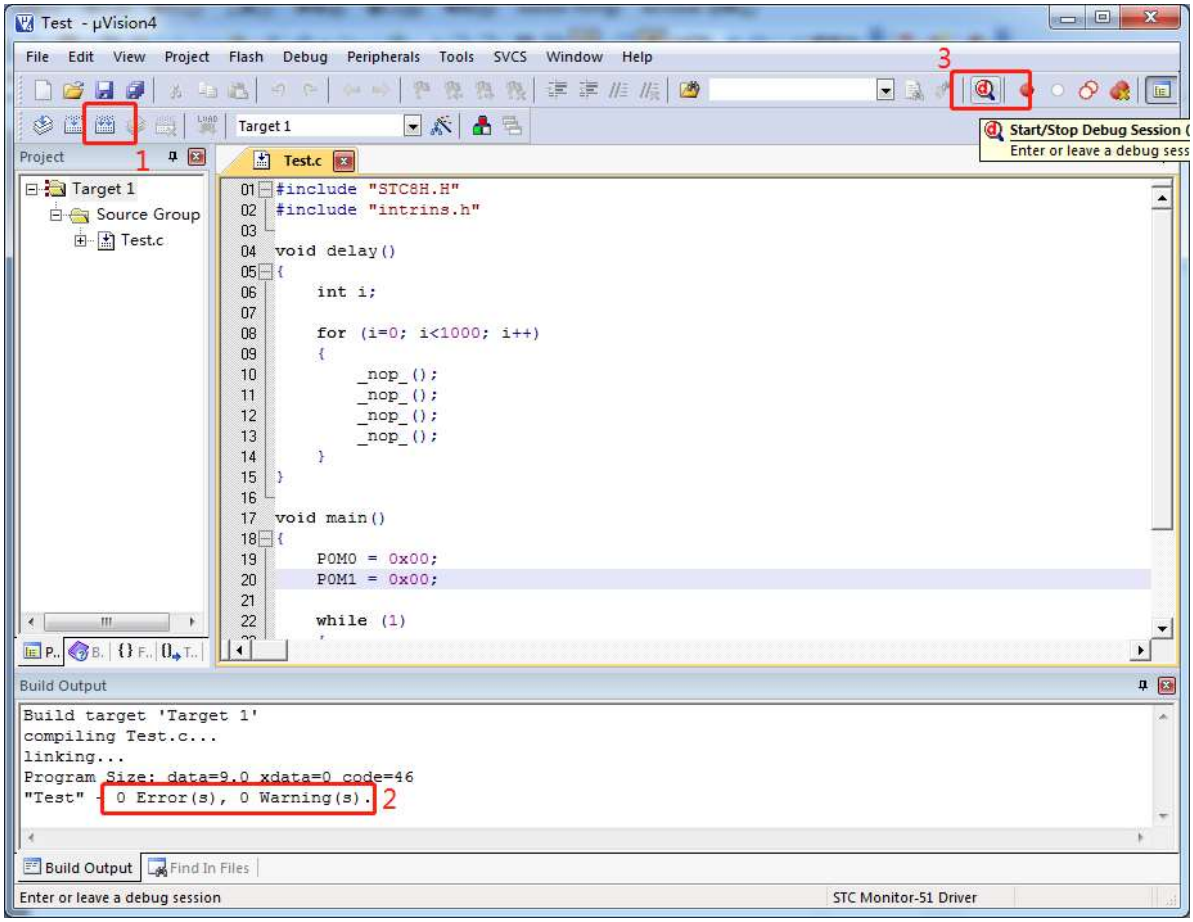


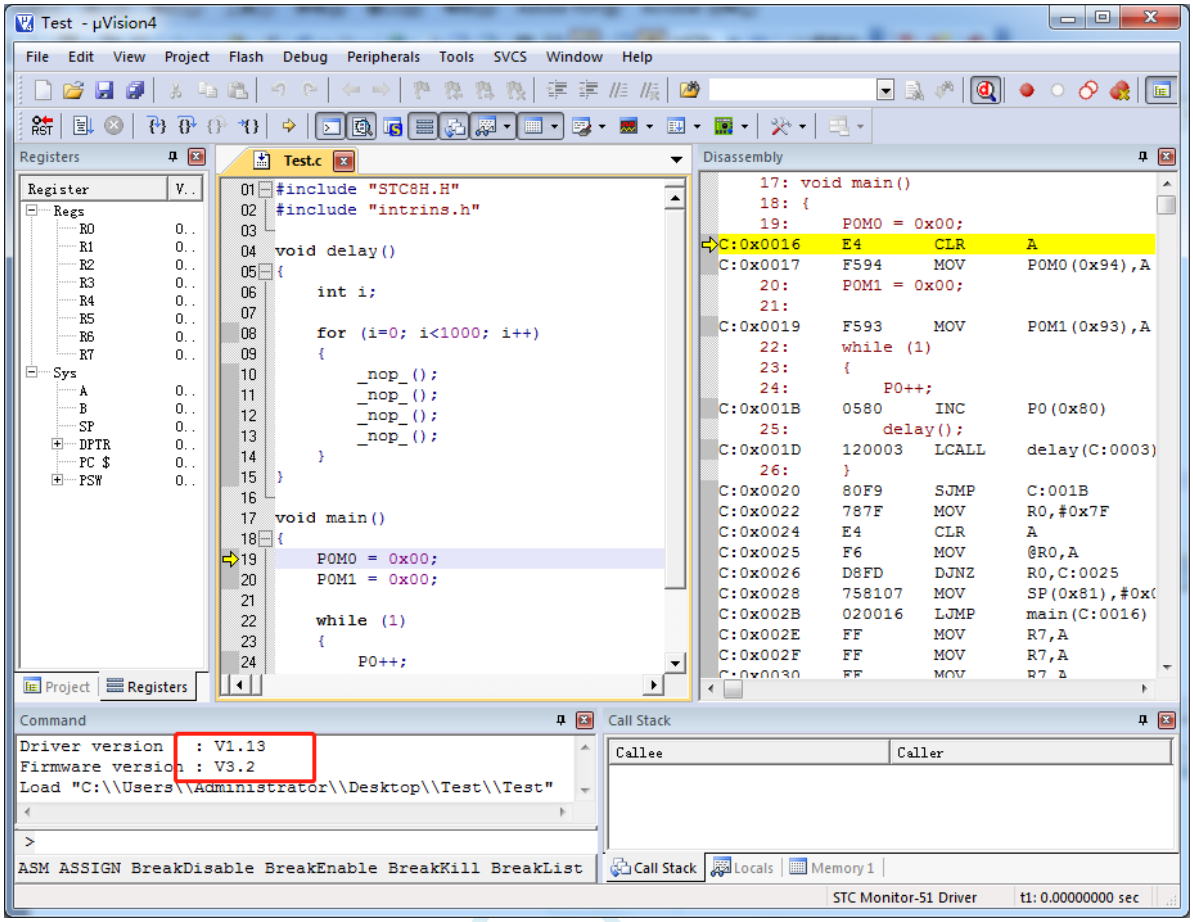
在项目选项中，按如下图所示的步骤进行 USB 仿真设置



## I.5.3 在 Keil 软件中使用 USB 进行仿真

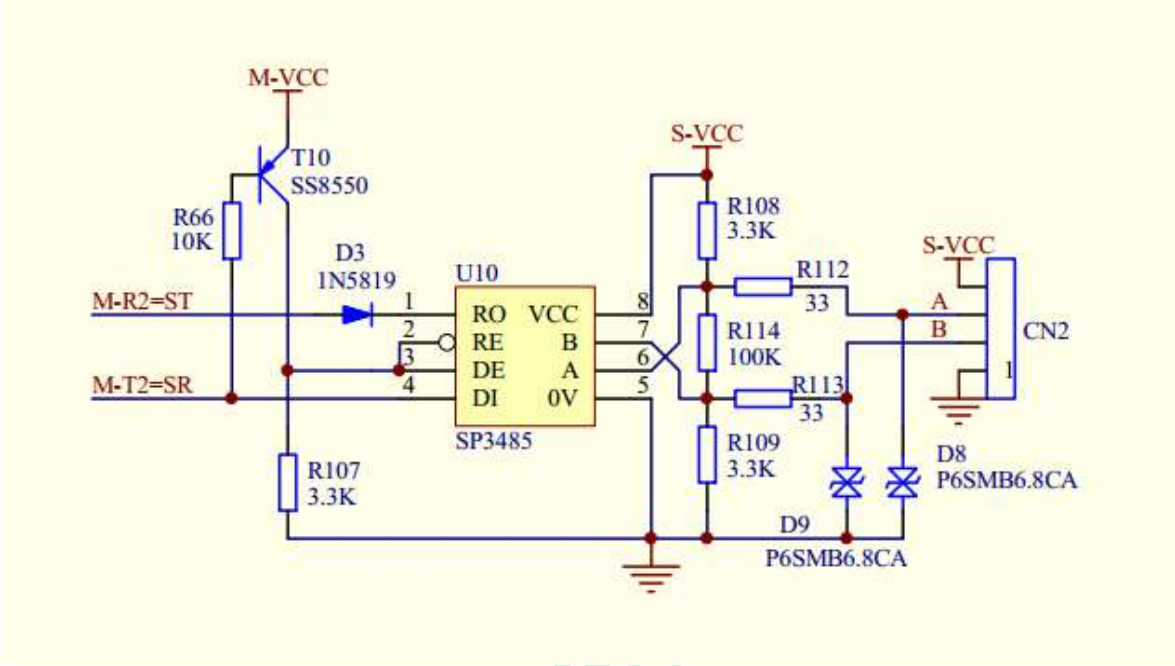
在 Keil 环境下，编辑完成源代码，并编译无误后，即可开始仿真






若芯片制作和连接均无误，则会如上图所示显示仿真驱动版本，并可正确下载用户代码到单片机，接下来便可进行运行、单步、断点等调试功能了。

# 附录J　U8W 下载工具中 RS485 部分线路图




BOM 清单：

| 标号 | 型号 | 封装 | 备注 |
|---|---|---|---|
| U10 | SP3485EN | SOP8 | RS485 芯片 |
| R66 | 10K | 0603 | 电阻 |
| R107 | 3.3K | 0603 | 电阻 |
| R108 | 3.3K | 0603 | 电阻 |
| R109 | 3.3K | 0603 | 电阻 |
| R112 | 33R | 0603 | 电阻 |
| R113 | 33R | 0603 | 电阻 |
| R114 | 100K | 0603 | 电阻 |
| T10 | SS8550 | SOT-23 | PNP 三极管 |
| D3 | 1N5819 | 0603 | 肖特基二极管 |
| D8 | P6SMB6.8CA | DO-214AA | TVS 二极管 |
| D9 | P6SMB6.8CA | DO-214AA | TVS 二极管 |
| CN2 | | SIP4 | 通信接口 |

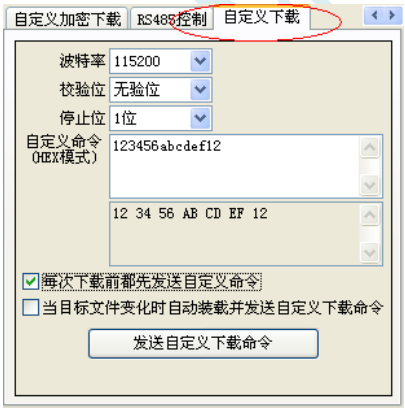
# 附录K 运行用户程序时收到用户命令后自动启动 ISP 下载(不停电)

“用户自定义下载”与“用户自定义加密下载”是两种完全不同功能。相对用户自定义加密下载的功能而言，用户自定义下载的功能要简单一些。

具体的功能为：电脑或脱机下载板在开始发送真正的 ISP 下载编程握手命令前，先发送用户自定义的一串命令（关于这一串串口命令，用户可以根据自己在应用程序中的串口设置来设置波特率、校验位以及停止位），然后再立即发送 ISP 下载编程握手命令。

“用户自定义下载”这一功能主要是在项目的早期开发阶段，实现不断电（不用给目标芯片重新上电）即可下载用户代码。具体的实现方法是：用户需要在自己的程序中加入一段检测自定义命令的代码，当检测到后，执行一句“MOV IAP_CONTR,#60H”的汇编代码或者“IAP_CONTR = 0x60;”的 C 语言代码，MCU 就会自动复位到 ISP 区域执行 ISP 代码。

如下图所示，将自定义命令设置为波特率为 115200、无校验位、一位停止位的命令序列：0x12、0x34、0x56、0xAB、0xCD、0xEF、0x12。当勾选上“每次下载前都先发送自定义命令”的选项后，即可实现自定义下载功能




点击“发送自定义下载命令”或者点击界面左下角的“下载/编程”按钮，应用程序便会发送如下所示的串口数据


Function
Data ( Hex )
IRP_MJ_CREATE Port Opened - stc-isp-15xx.exe
IOCTL_SERIAL_SET_BA... Baud Rate: 115200
IOCTL_SERIAL_SET_LI... StopBits: 1, Parity: No, DataBits: 8
IRP_MJ_WRITE Length: 0001, Data: 12
IRP_MJ_WRITE Length: 0001, Data: 34
IRP_MJ_WRITE Length: 0001, Data: 56
IRP_MJ_WRITE Length: 0001, Data: AB
IRP_MJ_WRITE Length: 0001, Data: CD
IRP_MJ_WRITE Length: 0001, Data: EF
IRP_MJ_WRITE Length: 0001, Data: 12
IOCTL_SERIAL_SET_BA... Baud Rate: 2400
IOCTL_SERIAL_SET_LI... StopBits: 1, Parity: Even, DataBits: 8
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
IRP_MJ_WRITE Length: 0001, Data: 7F
1. 用户的串口设置
2. 用户的自定义命令
3. isp的串口设置及命令
单片机型号
扫描
最低波特率 2400
最高波特率 115200
打开程序文件
自定义加密下载
485控制
自定义下载
波特率 115200
校验位 无校验位
停止位 1位
自定义命令 (HEX模式)
12 34 56 AB CD EF 12
每次下载前都先发送自定义命令
当目标文件变化时自动装载并发送自定义下载命令
发送自定义下载命令

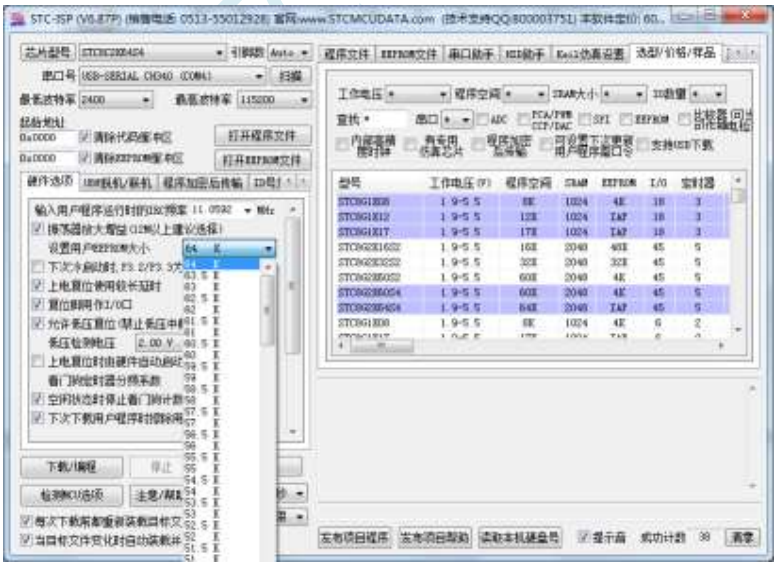
# 附录L 使用 STC 的 IAP 系列单片机开发自己的 ISP 程序

随着 IAP（In-Application-Programming）技术在单片机领域的不断发展，给应用系统程序代码升级带来了极大的方便。STC 的串口 ISP（In-System-Programming）程序就是使用 IAP 功能来对用户的程序进行在线升级的，但是出于对用户代码的安全着想，底层代码和上层应用程序都没有开源，为此 STC 推出了 IAP 系列单片机，即整颗 MCU 的 Flash 空间，用户均可在自己的程序中进行改写，从而使得有用户需要开发自己的 ISP 程序的想法得以实现。

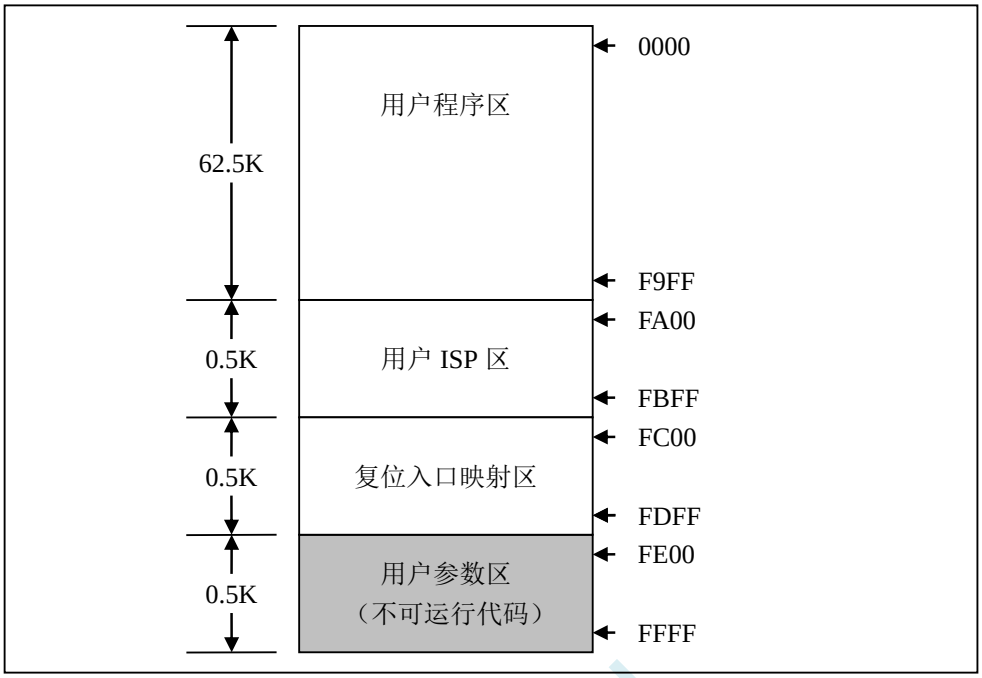
STC8A8K64D4 系列单片机中的所有可以在 ISP 下载时用户自定义 EEPROM 大小的型号均为 IAP 系列单片机。目前 STC8H 系列有如下型号的单片机为 IAP 系列：STC8A8K64D4。本文以 STC8A8K64D4 为例，详细说明使用 STC 的 IAP 单片机开发用户自己的 ISP 程序的方法，并给出了基于 Keil 环境的汇编和 C 源码。

## 第一步：内部 FLASH 规划

由于 STC8A8K64D4 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。

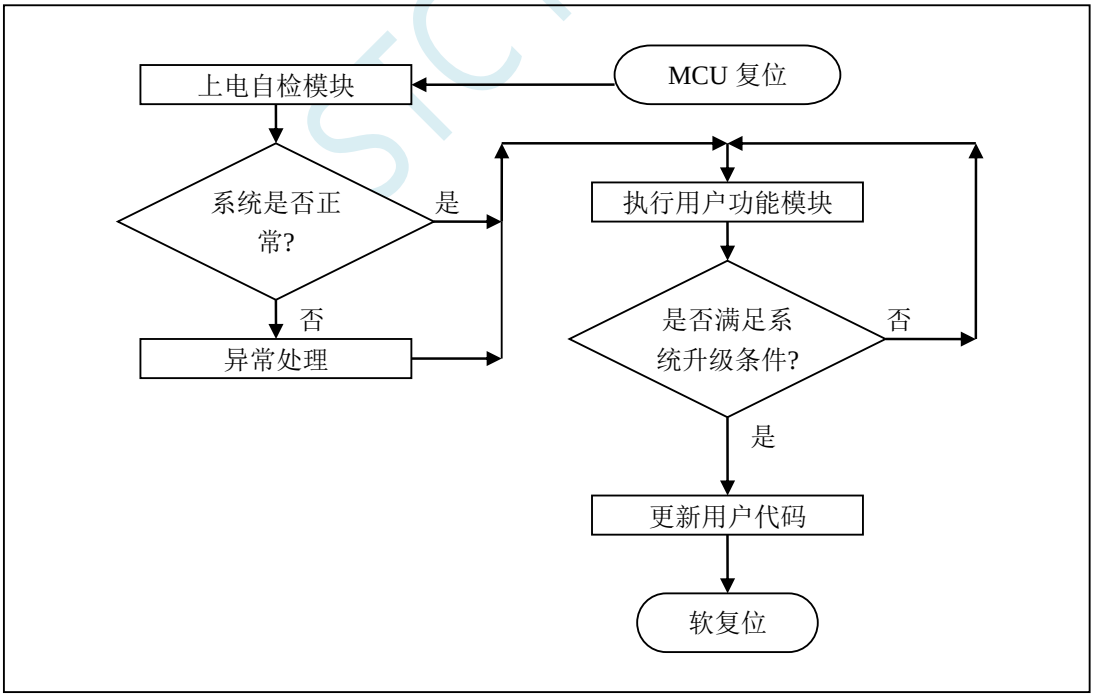


下面假设用户已将整个的 64K 的程序空间已全部设置为 EEPROM，现将整个 64K 程序空间作如下划分：




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
```

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
    }
```

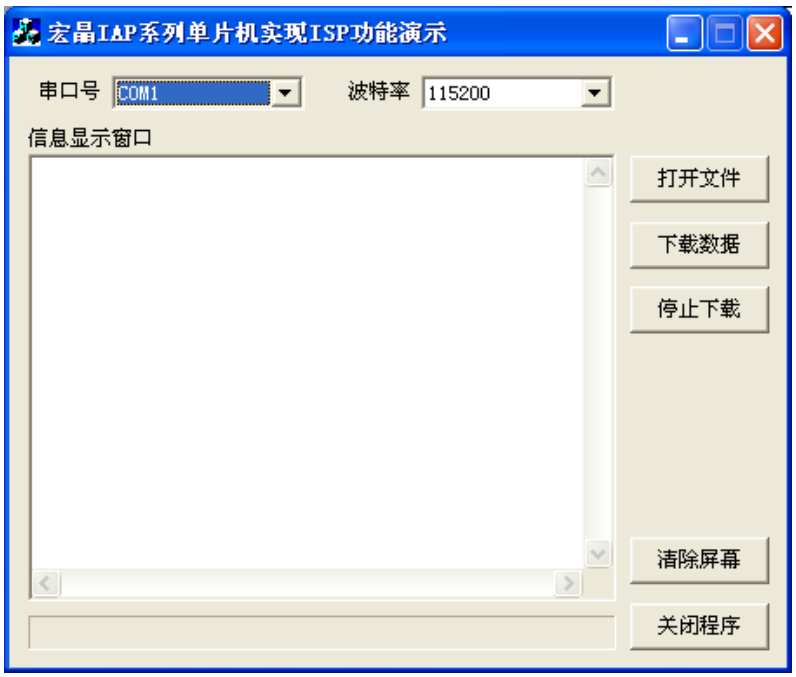
## 第四步、上位机应用程序说明

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

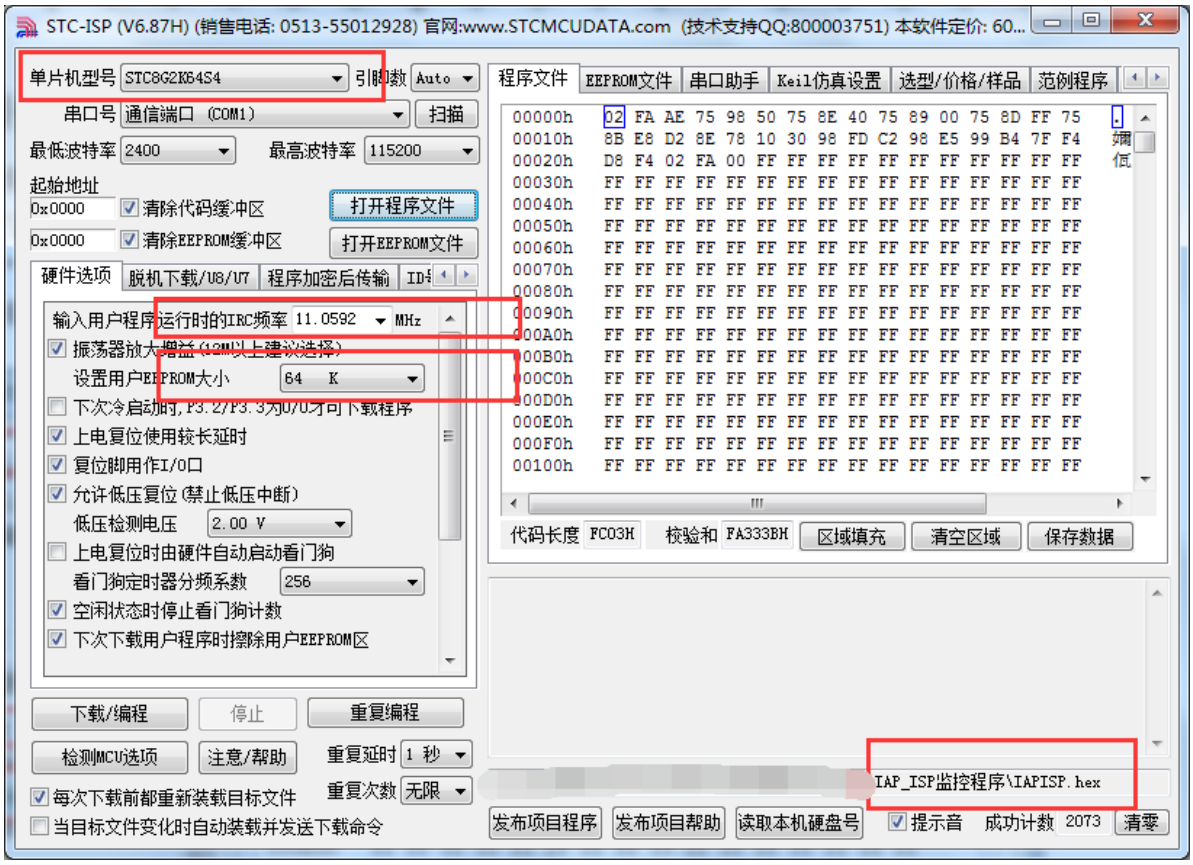
- 打开上位机界面，如下图




- 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M 用户程序复位到系统区进行 ISP 下载的方法（不停电）

当项目处于开发阶段时，需要反复的下载用户代码到目标芯片中进行代码验证，而 STC 的单片机进行正常的 ISP 下载都需要对目标芯片进行重新上电，从而会使得项目在开发阶段比较繁琐。为此 STC 单片机增加了一个特殊功能寄存器 IAP_CONTR，当用户向此寄存器写入 0x60，即可实现软件复位到系统区，进而实现不停电就可进行 ISP 下载。

但是用户如何判断是否正在进行 ISP 下载？何时向寄存器 IAP_CONTR 写 0x60 触发软复位？就这两个问题，下面分别介绍四种判断方法：

## 使用 P3.0 口检测串口起始信号

STC 单片机的串口 ISP 固定使用 P3.0 和 P3.1 两个端口，当 ISP 下载软件开始下载时，会发送握手命令到单片机的 P3.0 口。若用户的 P3.0 和 P3.1 只是专门用于 ISP 下载，则可使用 P3.0 口检测串口的起始信号来判断 ISP 下载。

### C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr         IAP_CONTR    =    0xc7;
sfr         P3M0         =    0xb2;
sfr         P3M1         =    0xb1;

sbit        P30          =    P3^0;

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;
    P30 = 1;

    while (1)
    {
        if (!P30) IAP_CONTR = 0x60;          //P3.0 的低电平即为串口起始信号
                                             //软件复位到系统区

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

# 使用 P3.0/INT4 口的下降沿中断，检测串口起始信号

方法 B 与方法 A 类似，不同在于方法 A 使用的是查询方式，方法 B 使用中断方式。因为 STC 单片机的 P3.0 口为 INT4 的中断口。

### C 语言代码

```
//测试工作频率为 11.0592MHz

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

sfr         IAP_CONTR   =   0xc7;
sfr         INTCLKO     =   0x8f;
sfr         P3M0        =   0xb2;
sfr         P3M1        =   0xb1;

void Int4Isr() interrupt 16                 //INT4 中断服务程序
{
    IAP_CONTR = 0x60;                       //串口起始信号触发 INT4 中断
                                            //软件复位到系统区
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    INTCLKO |= 0x40;                        //使能 INT4 中断
    EA = 1;

    while (1)
    {
        ...                                 //用户代码
    }
}
```

# 使用 P3.0/RxD 口的串口接收，检测 ISP 下载软件发送的 7F

方法 A 与方法 B 都非常简单，但容易受干扰，如果 P3.0 口有任何一个干扰信号，都会触发软件复位，所以方法 C 是对串口数据进行校验。

STC 的 ISP 下载软件进行 ISP 下载时，首先都会使用最低波特率（一般是 2400）+偶校验 9+1 位停止位连续发送握手命令 7F，因此用户可以在程序中，将串口设置为 9 位数据位+2400 波特率，然后持续检测 7F，比如连续检测到 8 个 7F 表示可确定需要进行 ISP 下载，此时再触发软件复位。

### C 语言代码

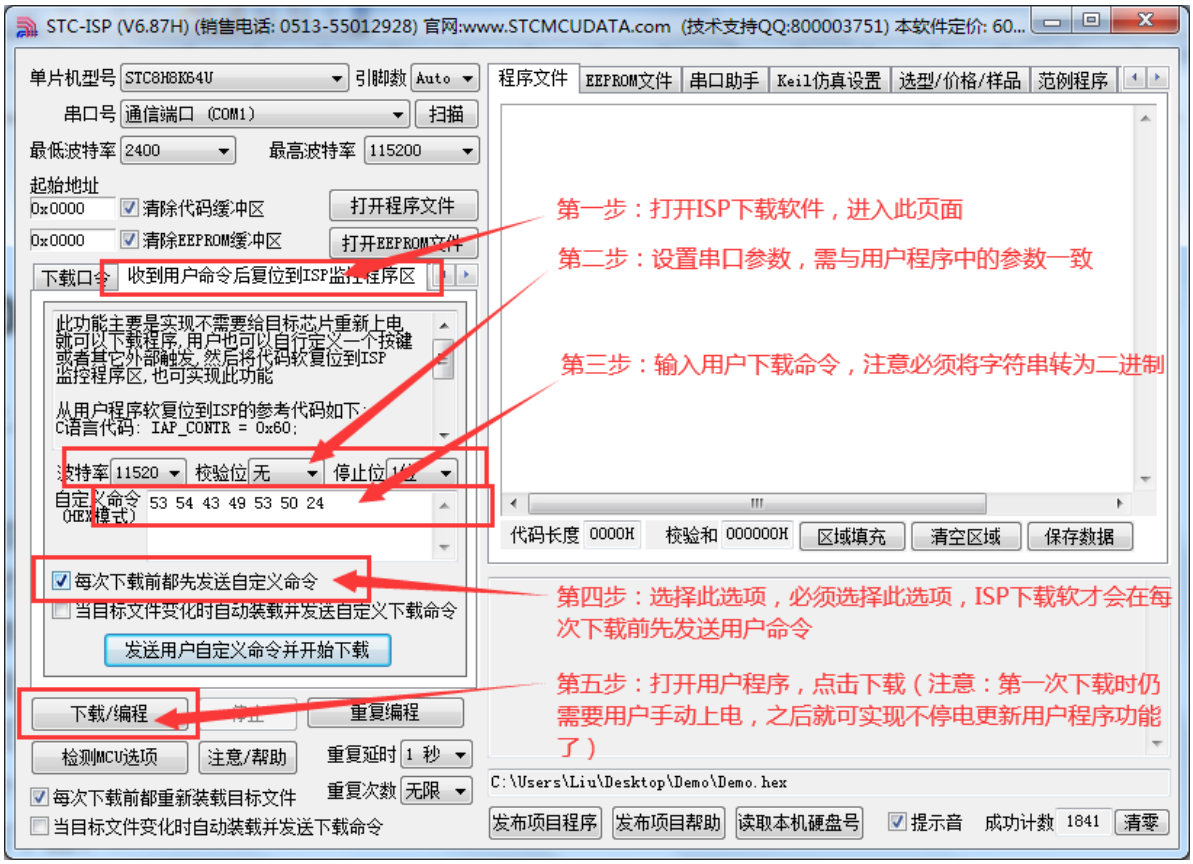
```
//测试工作频率为 11.0592MHz
```

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

#define   FOSC          11059200UL
#define   BR2400        (65536 - FOSC / 4 / 2400)

sfr       IAP_CONTR   =   0xc7;
sfr       AUXR        =   0x8e;
sfr       P3M0        =   0xb2;
sfr       P3M1        =   0xb1;

char cnt7f;

void UartIsr() interrupt 4                          //串口中断服务程序
{
    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;
        if ((SBUF == 0x7f) && (RB8 == 1))           //ISP 下载软件发送的握手命令 7F
                                                    //7F 的偶校验位为 1
        {
            if (++cnt7f == 8)                       //当连续检测到 8 个 7F 后
                IAP_CONTR = 0x60;                   //复位到系统区
        }
        else
        {
            cnt7f = 0;
        }
    }
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;

    SCON = 0xd0;                                    //设置串口为 9 位数据位
    TMOD = 0x00;
    AUXR = 0x40;
    TH1 = BR2400 >> 8;                              //设置串口波特率为 2400
    TL1 = BR2400;
    TR1 = 1;
    ES = 1;
    EA = 1;

    cnt7f = 0;

    while (1)
    {
        ...                                         //用户代码
    }
}
```

# 使用 P3.0/RxD 串口接收，检测 ISP 下载软件发送的用户下载命令

如果用户代码中需要使用串口进行通信，则上面的 3 中方法可能都不太适用，此时可以使用 STC 的 ISP 下载软件提供的接口，定制一组专用的用户下载命令（可指定波特率、校验位和停止位），若使能此功能，ISP 下载软件在进行 ISP 下载前，会使用用户指定的波特率、校验位和停止位发送用户下载命令，然后再发送握手命令。用户只需要在自己的代码中监控串口命令序列，当检测到有正确的用户下载命令时，软件复位到系统区即可实现不停电进行 ISP 功能。

下面假设用户下载命令为字符串“STCISP$”，串口设置为波特率 115200，无校验位和 1 位停止位。ISP 下载软件中的设置如下图：




用户示例代码如下：

**C 语言代码**

```
//测试工作频率为11.0592MHz

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

#define    FOSC        11059200UL
#define    BR115200    (65536 - FOSC / 4 / 115200)

sfr    IAP_CONTR  =  0xc7;
sfr    AUXR       =  0x8e;
sfr    P3M0       =  0xb2;
```

```
sfr         P3M1        =   0xb1;

char stage;

void UartIsr() interrupt 4                          //串口中断服务程序
{
    char dat;

    if (TI)
    {
        TI = 0;
    }

    if (RI)
    {
        RI = 0;

        dat = SBUF;
        switch (stage)
        {
        case 0:
        default:
L_Check1st:
            if (dat == 'S') stage = 1;
            else stage = 0;
            break;
        case 1:
            if (dat == 'T') stage = 2;
            else goto L_Check1st;
            break;
        case 2:
            if (dat == 'C') stage = 3;
            else goto L_Check1st;
            break;
        case 3:
            if (dat == 'I') stage = 4;
            else goto L_Check1st;
            break;
        case 4:
            if (dat == 'S') stage = 5;
            else goto L_Check1st;
            break;
        case 5:
            if (dat == 'P') stage = 6;
            else goto L_Check1st;
            break;
        case 6:
            if (dat == '$')                         //当检测到正确的用户下载命令时
                IAP_CONTR = 0x60;                   //复位到系统区
            else goto L_Check1st;
        break;
        }
    }
}

void main()
{
    P3M0 = 0x00;
    P3M1 = 0x00;
```

```
    SCON = 0x50;                                //设置用户串口模式为 8 位数据位
    TMOD = 0x00;
    AUXR = 0x40;
    TH1 = BR2400 >> 8;                          //设置串口波特率为 115200
    TL1 = BR2400;
    TR1 = 1;
    ES = 1;
    EA = 1;

    stage = 0;

    while (1)
    {
        ...                                     //用户代码
    }
}
```

# 附录N 使用第三方 MCU 对 STC8A8K64D4 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 STC8A8K64D4 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          //STC8A8K64D4 系列目标芯片工作频率
#define    RL(n)           (65536 - FUSER/4/(n))  //STC8A8K64D4 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--;
    }
}
```

```
//串口中断服务程序
void uart(void) interrupt 4
{
	static WORD RecvSum;
	static BYTE RecvIndex;
	static BYTE RecvCount;
	BYTE dat;

	if (TI)
	{
		TI = 0;
		UartBusy = FALSE;
	}

	if (RI)
	{
		RI = 0;
		dat  = SBUF;
		switch (UartRecvStep)
		{
		case 1:
			if (dat != 0xb9) goto L_CheckFirst;
			UartRecvStep++;
			break;
		case 2:
			if (dat != 0x68) goto L_CheckFirst;
			UartRecvStep++;
			break;
		case 3:
			if (dat != 0x00) goto L_CheckFirst;
			UartRecvStep++;
			break;
		case 4:
			RecvSum = 0x68 + dat;
			RecvCount = dat - 6;
			RecvIndex = 0;
			UartRecvStep++;
			break;
		case 5:
			RecvSum += dat;
			RxBuffer[RecvIndex++] = dat;
			if (RecvIndex == RecvCount)	UartRecvStep++;
			break;
		case 6:
			if (dat != HIBYTE(RecvSum))	goto L_CheckFirst;
			UartRecvStep++;
			break;
		case 7:
			if (dat != LOBYTE(RecvSum))	goto L_CheckFirst;
			UartRecvStep++;
			break;
		case 8:
			if (dat != 0x16) goto L_CheckFirst;
			UartReceived = TRUE;
			UartRecvStep++;
			break;
L_CheckFirst:
		case 0:
```

```
            default:
                CommInit();
                UartRecvStep = (dat == 0x46       ? 1 : 0);
                break;
            }
        }
}

//系统初始化
void Initial(void)
{
    UartBusy = FALSE;

    SCON = 0xd0;                        //串口数据模式必须为8 位数据+1 位偶检验
    AUXR = 0xc0;
    TMOD = 0x00;
    TH0 = HIBYTE(T1MS);
    TL0 = LOBYTE(T1MS);
    TR0 = 1;
    TH1 = HIBYTE(BR(MINBAUD));
    TL1 = LOBYTE(BR(MINBAUD));
    TR1 = 1;
    ET0 = 1;
    ES = 1;
    EA = 1;
}

//Xms 延时程序
void DelayXms(WORD x)
{
    do
    {
        f1ms = FALSE;
        while (!f1ms);
    } while (x--);
}

//串口数据发送程序
BYTE UartSend(BYTE dat)
{
    while (UartBusy);

    UartBusy = TRUE;
    ACC = dat;
    TB8 = P;
    SBUF = ACC;

    return dat;
}

//串口通讯初始化
void CommInit(void)
{
    UartRecvStep = 0;
    TimeOut = 20;
    UartReceived = FALSE;
}

//发送串口通讯数据包
```

```
void CommSend(BYTE size)
{
	WORD sum;
	BYTE i;

	UartSend(0x46);
	UartSend(0xb9);
	UartSend(0x6a);
	UartSend(0x00);
	sum = size + 6 + 0x6a;
	UartSend(size + 6);
	for (i=0; i<size; i++)
	{
		sum  += UartSend(TxBuffer[i]);
	}
	UartSend(HIBYTE(sum));
	UartSend(LOBYTE(sum));
	UartSend(0x16);
	while (UartBusy);

	CommInit();
}

//对 STC15H 系列的芯片进行 ISP 下载程序
BOOL Download(BYTE *pdat, long size)
{
	BYTE arg;
	BYTE offset;
	BYTE cnt;
	WORD addr;

	//握手
	CommInit();
	while (1)
	{
		if (UartRecvStep == 0)
		{
			UartSend(0x7f);
			DelayXms(10);
		}
		if (UartReceived)
		{
			arg = RxBuffer[4];
			if (RxBuffer[0] == 0x50) break;
			return FALSE;
		}
	}

	//设置参数(设置从芯片使用最高的波特率以及等待时间等参数)
	TxBuffer[0] = 0x01;
	TxBuffer[1] = arg;
	TxBuffer[2] = 0x40;
	TxBuffer[3] = HIBYTE(RL(MAXBAUD));
	TxBuffer[4] = LOBYTE(RL(MAXBAUD));
	TxBuffer[5] = 0x00;
	TxBuffer[6] = 0x00;
	TxBuffer[7] = 0x97;
	CommSend(8);
	while (1)
```

```
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x01) break;
        return FALSE;
    }
}

//准备
TH1 = HIBYTE(BR(MAXBAUD));
TL1 = LOBYTE(BR(MAXBAUD));
DelayXms(10);
TxBuffer[0] = 0x05;
TxBuffer[1] = 0x00;
TxBuffer[2] = 0x00;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
CommSend(5);
while (1)
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x05) break;
        return FALSE;
    }
}

//擦除
DelayXms(10);
TxBuffer[0] = 0x03;
TxBuffer[1] = 0x00;
TxBuffer[2] = 0x00;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
CommSend(5);
TimeOut = 100;
while (1)
{
    if (TimeOut == 0) return FALSE;
    if (UartReceived)
    {
        if (RxBuffer[0] == 0x03) break;
        return FALSE;
    }
}

//写用户代码
DelayXms(10);
addr = 0;
TxBuffer[0] = 0x22;
TxBuffer[3] = 0x5a;
TxBuffer[4] = 0xa5;
offset = 5;
while (addr < size)
{
    TxBuffer[1] = HIBYTE(addr);
    TxBuffer[2] = LOBYTE(addr);
```

```
        cnt   = 0;
        while (addr < size)
        {
            TxBuffer[cnt+offset] = pdat[addr];
            addr++;
            cnt++;
            if (cnt >= 128) break;
        }
        CommSend(cnt + offset);
        while (1)
        {
            if (TimeOut == 0) return FALSE;
            if (UartReceived)
            {
                if ((RxBuffer[0] == 0x02) && (RxBuffer[1] == 'T')) break;
                return FALSE;
            }
        }
        TxBuffer[0] = 0x02;
    }

    ////写硬件选项
    ////如果不需要修改硬件选项,此步骤可直接跳过,此时所有的硬件选项
    ////都维持不变,MCU 的频率为上一次所调节频率
    ////若写硬件选项,MCU 的内部 IRC 频率将被固定写为 24M, ,其他选项恢复为出厂设置
    ////建议:第一次使用 STC-ISP 下载软件将从芯片的硬件选项设置好
    ////以后再使用主芯片对从芯片下载程序时不写硬件选项
    //DelayXms(10);
    //for (cnt=0; cnt<128; cnt++)
    //{
    //    TxBuffer[cnt] = 0xff;
    //}
    //TxBuffer[0] = 0x04;
    //TxBuffer[1] = 0x00;
    //TxBuffer[2] = 0x00;
    //TxBuffer[3] = 0x5a;
    //TxBuffer[4] = 0xa5;
    //TxBuffer[33] = arg;
    //TxBuffer[34] = 0x00;
    //TxBuffer[35] = 0x01;
    //TxBuffer[41] = 0xbf;
    //TxBuffer[42] = 0xbd;                  //P5.4 为 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

# 附录O 使用第三方应用程序调用 STC 发布项目程序对单片机进行 ISP 下载

使用 STC 的 ISP 下载软件生成的发布项目程序为可执行的 EXE 格式文件，用户可直接双击发布的项目程序运行进行 ISP 下载，也可在第三方的应用程序中调用发布项目程序进行 ISP 下载。下面介绍两种调用的方法。

## 简单调用

在第三方应用程序中只是简单创建发布项目程序的进程，其他的所有下载操作均在发布项目程序中进行，第三方应用程序此时只需要等待发布项目程序操作完成后，清理现场即可。

**VC 代码**

```
BOOL IspProcess()
{
    //定义相关变量
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    CString path;

    //发布项目程序的完整路径
    path = _T("D:\\Work\\Upgrade.exe");

    //变量初始化
    memset(&si, 0, sizeof(STARTUPINFO));
    memset(&pi, 0, sizeof(PROCESS_INFORMATION));

    //设置启动变量
    si.cb = sizeof(STARTUPINFO);
    GetStartupInfo(&si);
    si.wShowWindow = SW_SHOWNORMAL;
    si.dwFlags = STARTF_USESHOWWINDOW;

    //创建发布项目程序进程
    if (CreateProcess(NULL, (LPTSTR)(LPCTSTR)path, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
    {
        //等待发布项目程序操作完成
        //由于此处会阻塞主进程，所以建议新建工作进程，在工作进程中进行等待
        WaitForSingleObject(pi.hProcess,INFINITE);

        //清理工作
        CloseHandle(pi.hThread);
        CloseHandle(pi.hProcess);

        return TRUE;
    }
    else
    {
        AfxMessageBox(_T("创建进程失败 !"));
```

```
            return FALSE;
        }
}
```

# 高级调用

在第三方应用程序创建发布项目程序的进程，并在第三方应用程序中进行包括选择串口、开始 ISP 编程、停振 ISP 编程以及关闭发布项目程序等的全部 ISP 下载操作，而不需要在发布项目程序中进行界面互动。

**VC 代码**

```
//定义回调函数参数的数据结构
struct CALLBACK_PARAM
{
    DWORD dwProcessId;                  //主进程 ID
    HWND hMainWnd;                      //主窗口句柄
};

//枚举窗口的回调函数，用于获取主窗口句柄
BOOL CALLBACK EnumWindowCallBack(HWND hWnd, LPARAM lParam)
{
    CALLBACK_PARAM *pcp = (CALLBACK_PARAM *)lParam;
    DWORD id;

    GetWindowThreadProcessId(hWnd, &id);
    if ((pcp->dwProcessId == id) && (GetParent(hWnd) == NULL))
    {
        pcp->hMainWnd = hWnd;

        return FALSE;
    }

    return TRUE;
}

BOOL IspProcess()
{
    //定义相关变量
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    CALLBACK_PARAM cp;
    CString path;

    //发布项目程序中部分控件的 ID
    const UINT ID_PROGRAM       = 1013;
    const UINT ID_STOP          = 1012;
    const UINT ID_COMPORT       = 1001;
    const UINT ID_PROGRESS      = 1000;

    //发布项目程序的完整路径
    path = _T("D:\\Work\\Upgrade.exe");

    //变量初始化
    memset(&si, 0, sizeof(STARTUPINFO));
    memset(&pi, 0, sizeof(PROCESS_INFORMATION));
    memset(&cp, 0, sizeof(CALLBACK_PARAM));
```

```
//设置启动变量
si.cb = sizeof(STARTUPINFO);
GetStartupInfo(&si);
si.wShowWindow = SW_SHOWNORMAL;                    //此处若设置为SW_HIDE,就不会显示发布项目程序
                                                   //的操作界面,所有的ISP 操作都可在后台进行
si.dwFlags = STARTF_USESHOWWINDOW;

//创建发布项目程序进程
if (CreateProcess(NULL, (LPTSTR)(LPCTSTR)path, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))
{
    //等待发布项目程序进程初始化完成
    WaitForInputIdle(pi.hProcess, 5000);

    //获取发布项目程序的主窗口句柄
    cp.dwProcessId = pi.dwProcessId;
    cp.hMainWnd = NULL;
    EnumWindows(EnumWindowCallBack, (LPARAM)&cp);

    if (cp.hMainWnd != NULL)
    {
        HWND hProgram;
        HWND hStop;
        HWND hPort;

        //获取发布项目程序主窗口中部分控件句柄
        hProgram = ::GetDlgItem(cp.hMainWnd, ID_PROGRAM);
        hStop = ::GetDlgItem(cp.hMainWnd, ID_STOP);
        hPort = ::GetDlgItem(cp.hMainWnd, ID_COMPORT);

        //设置发布项目程序中的串口号, 第3 个参数为0:COM1, 1:COM2, 2:COM3, ...
        ::SendMessage(hPort, CB_SETCURSEL, 0, 0);

        //触发编程按钮开始ISP 编程
        ::SendMessage(hProgram, BM_CLICK, 0, 0);

        //等待编程完成
        //由于此处会阻塞主进程，所以建议新建工作进程，在工作进程中进行等待
        while (!::IsWindowEnabled(hProgram));

        //编程完成后关闭发布项目程序
        ::SendMessage(cp.hMainWnd, WM_CLOSE, 0, 0);
    }

    //等待进程结束
    WaitForSingleObject(pi.hProcess,INFINITE);

    //清理工作
    CloseHandle(pi.hThread);
    CloseHandle(pi.hProcess);

    return TRUE;
}
else
{
    AfxMessageBox(_T("创建进程失败 !"));

    return FALSE;
}
```

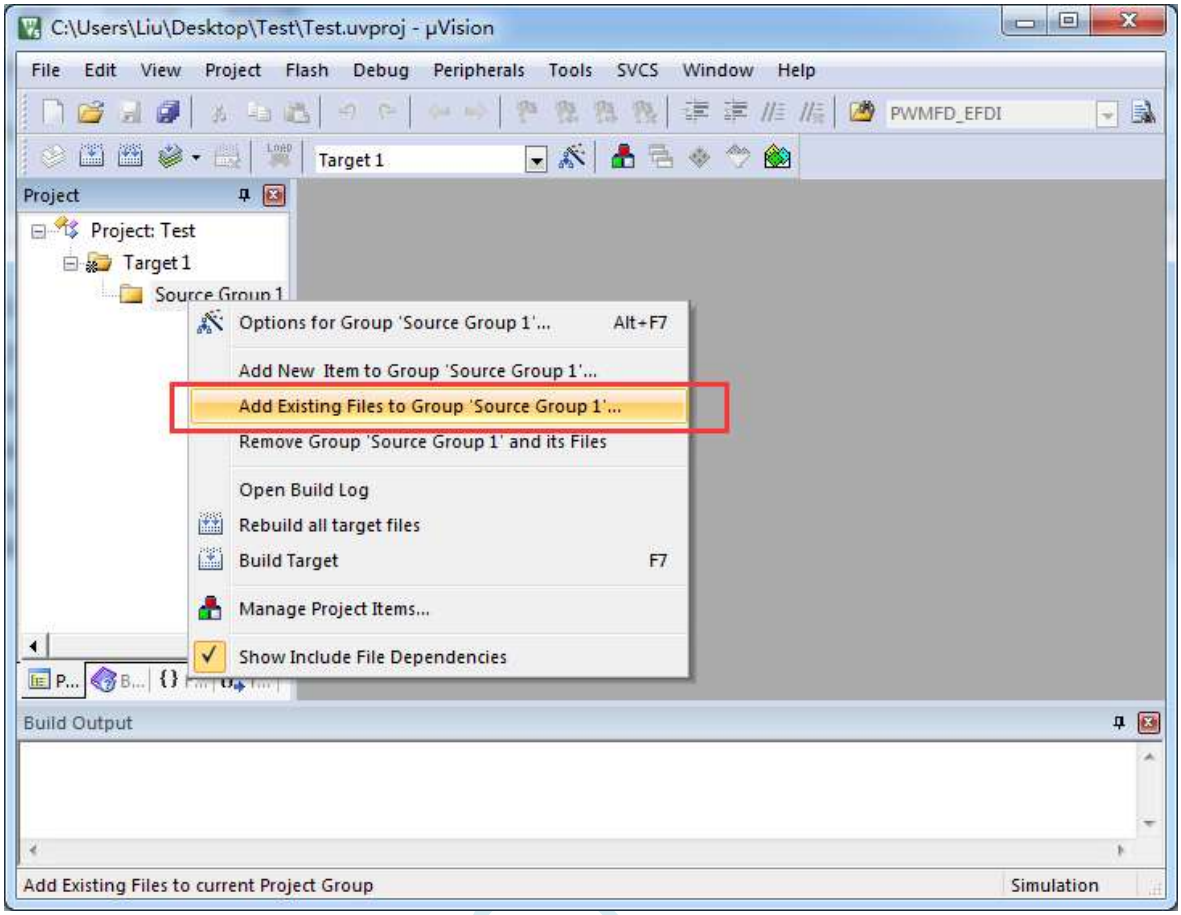
```
}
```

# 附录P 在 Keil 中建立多文件项目的方法

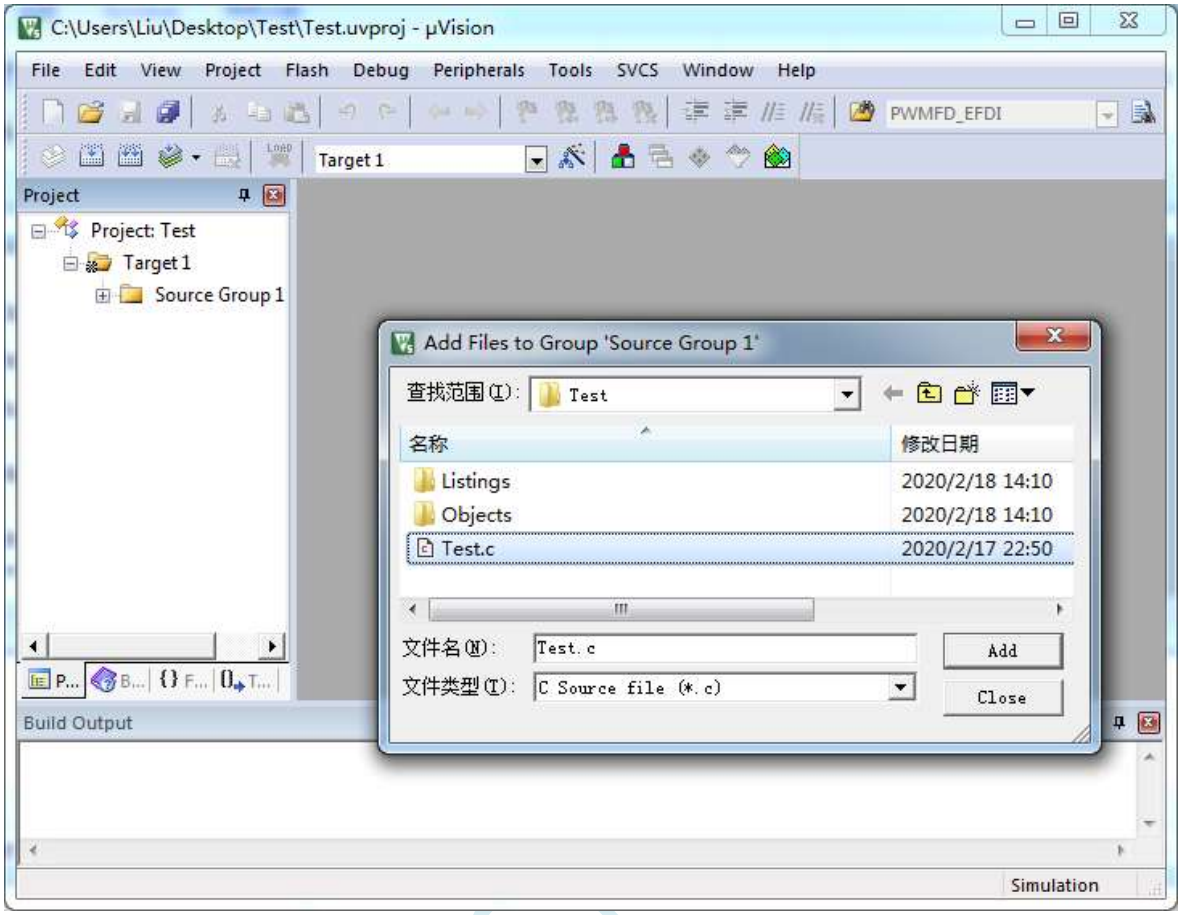
在 Keil 中，一般比较小的项目都只有一个源文件，但对于一些稍微复杂的项目往往需要多个源文件
建立多文件项目的方法如下：
1、首先打开 Keil，在菜单“Project”中选择“New uVision Project ...”


即可完成一个空项目的建立

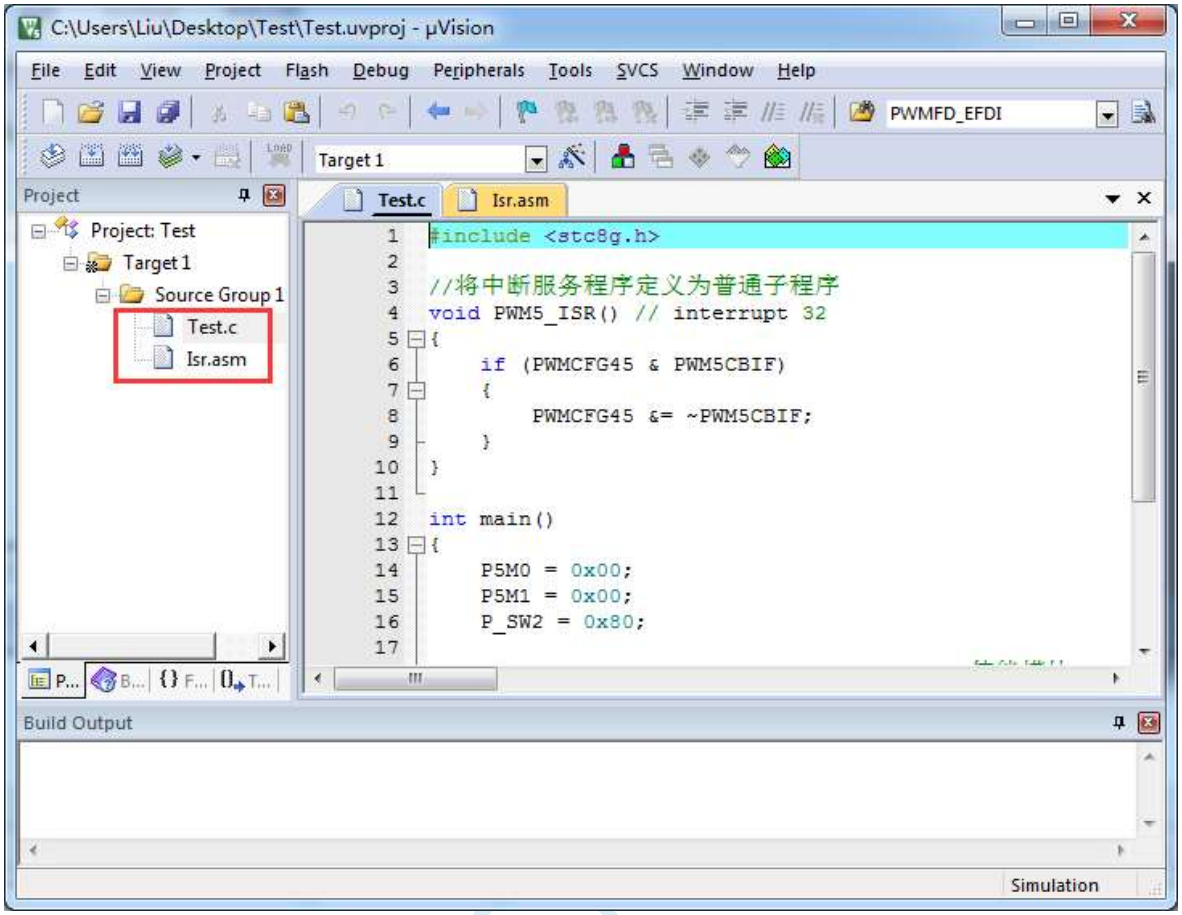
2、在空项目的项目树中，鼠标右键单击“Source Group 1”，并选择右键菜单中的“Add Existing Files to Group "Source Group 1" ...”




3、在弹出的文件对话框中，多次添加源文件

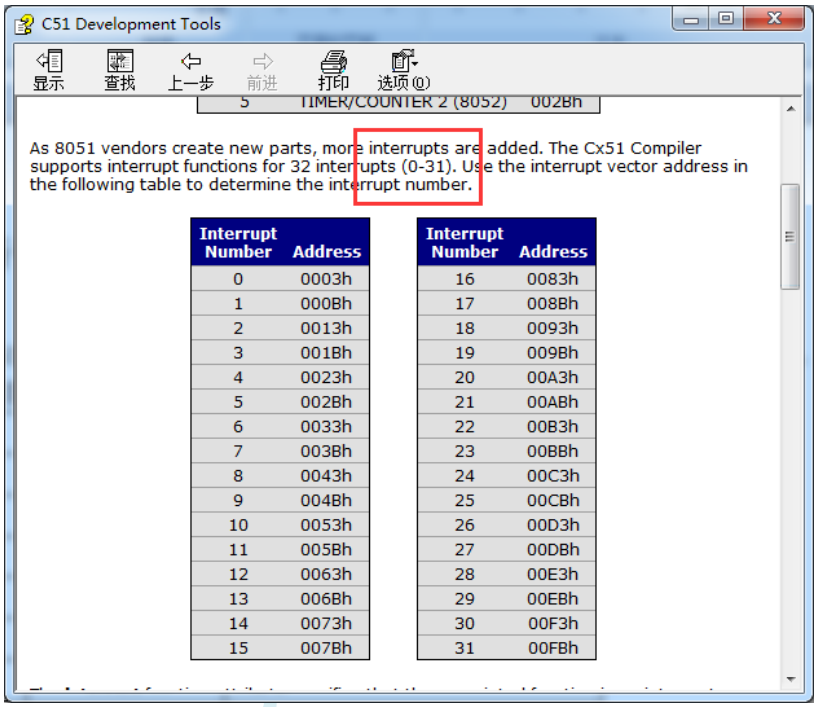



如下图所示即可完成多文件项目的建立


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Q 关于中断号大于 31 在 Keil 中编译出错的处理

在 Keil 的 C51 编译环境下，中断号只支持 0~31，即中断向量必须小于 0100H。



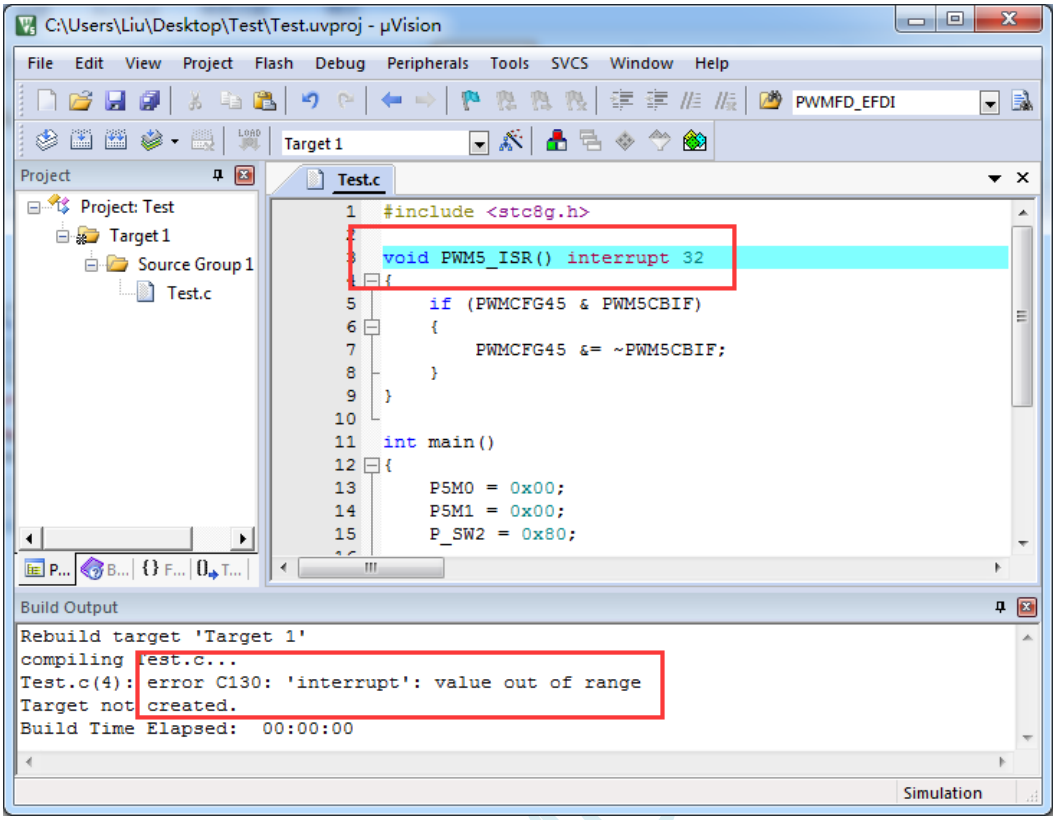

| Interrupt Number | Address | Interrupt Number | Address |
|---|---|---|---|
| 0 | 0003h | 16 | 0083h |
| 1 | 000Bh | 17 | 008Bh |
| 2 | 0013h | 18 | 0093h |
| 3 | 001Bh | 19 | 009Bh |
| 4 | 0023h | 20 | 00A3h |
| 5 | 002Bh | 21 | 00ABh |
| 6 | 0033h | 22 | 00B3h |
| 7 | 003Bh | 23 | 00BBh |
| 8 | 0043h | 24 | 00C3h |
| 9 | 004Bh | 25 | 00CBh |
| 10 | 0053h | 26 | 00D3h |
| 11 | 005Bh | 27 | 00DBh |
| 12 | 0063h | 28 | 00E3h |
| 13 | 006Bh | 29 | 00EBh |
| 14 | 0073h | 30 | 00F3h |
| 15 | 007Bh | 31 | 00FBh |

下表是 STC 目前所有系列的中断列表：

| 中断号 | 中断向量 | 中断类型 |
|---|---|---|
| 0 | 0003 H | INT0 |
| 1 | 000B H | 定时器 0 |
| 2 | 0013 H | INT1 |
| 3 | 001B H | 定时器 1 |
| 4 | 0023 H | 串口 1 |
| 5 | 002B H | ADC |
| 6 | 0033 H | LVD |
| 7 | 003B H | PCA |
| 8 | 0043 H | 串口 2 |
| 9 | 004B H | SPI |
| 10 | 0053 H | INT2 |
| 11 | 005B H | INT3 |
| 12 | 0063 H | 定时器 2 |
| **13** | **006B H** | |
| 14 | 0073 H | 系统内部中断 |

| | | |
|---|---|---|
| 15 | 007B H | 系统内部中断 |
| 16 | 0083 H | INT4 |
| 17 | 008B H | 串口 3 |
| 18 | 0093 H | 串口 4 |
| 19 | 009B H | 定时器 3 |
| 20 | 00A3 H | 定时器 4 |
| 21 | 00AB H | 比较器 |
| 22 | 00B3 H | 波形发生器 0 |
| 23 | 00BB H | 波形发生器异常 0 |
| 24 | 00C3 H | I2C |
| 25 | 00CB H | USB |
| 26 | 00D3 H | 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：



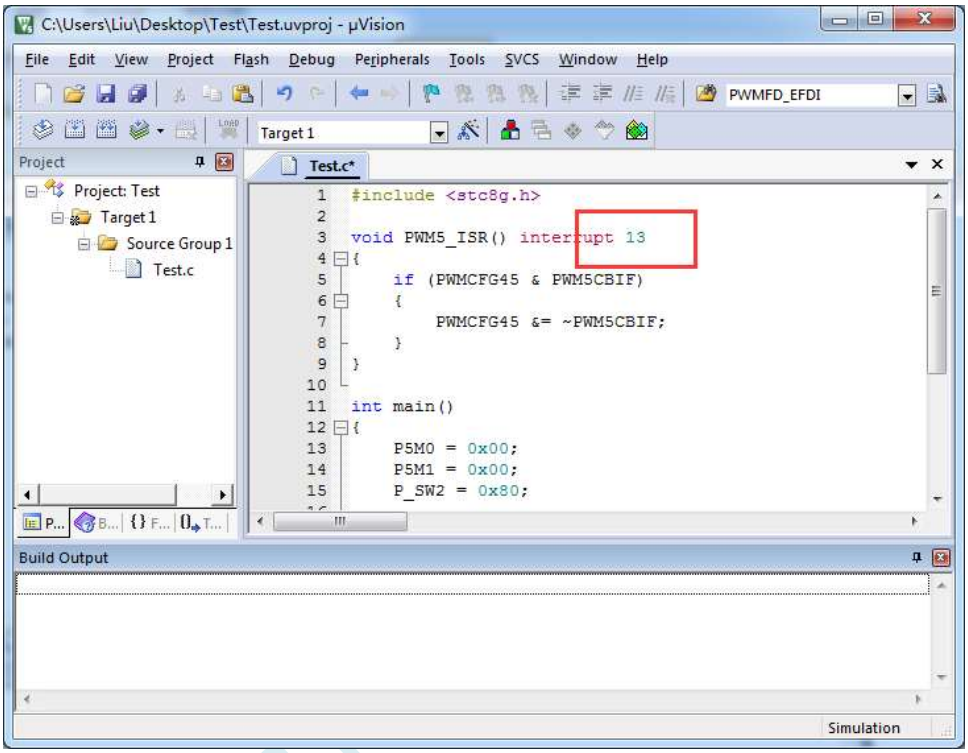

处理这种错误有如下三种方法：（均需要借助于汇编代码，优先推荐使用方法 1）

## 方法 1：借用 13 号中断向量

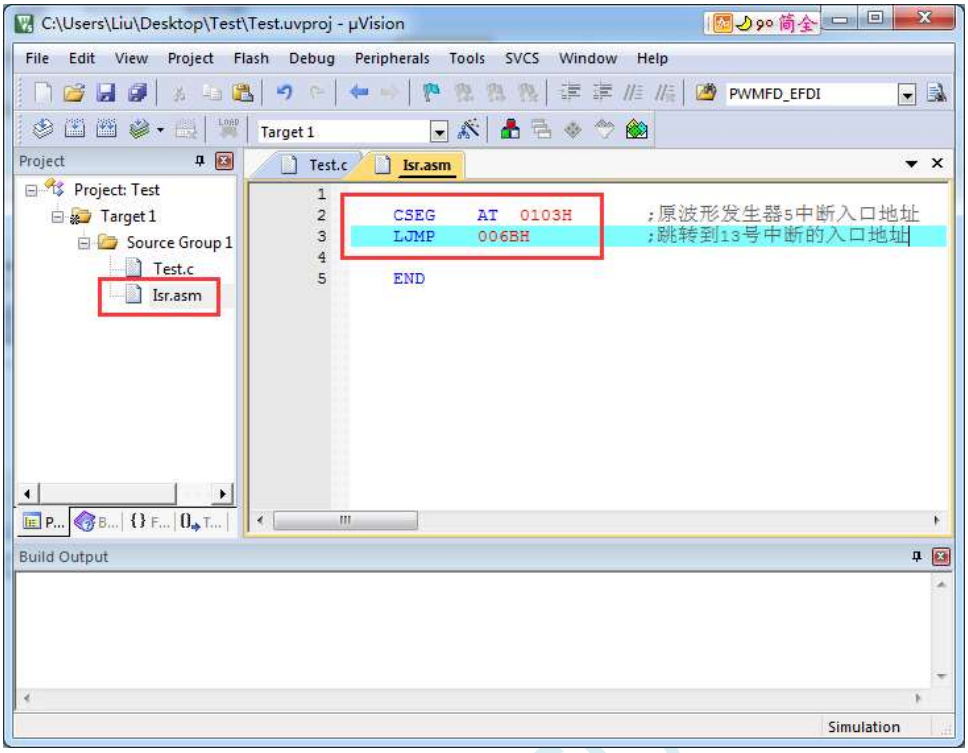
0~31 号中断中，第 13 号是保留中断号，我们可以借用此中断号

操作步骤如下：

1、将我们报错的中断号改为“13”，如下图：

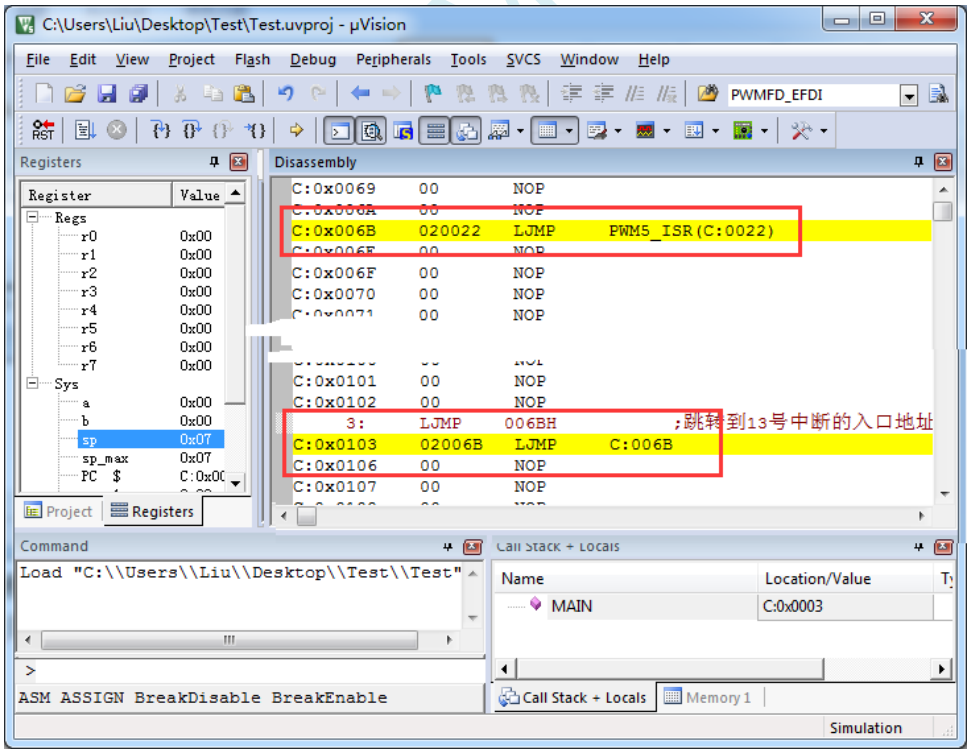



2、新建一个汇编语言文件，比如“isr.asm”，加入到项目，并在地址“0103H”的地方添加一条“LJMP 006BH”，如下图：

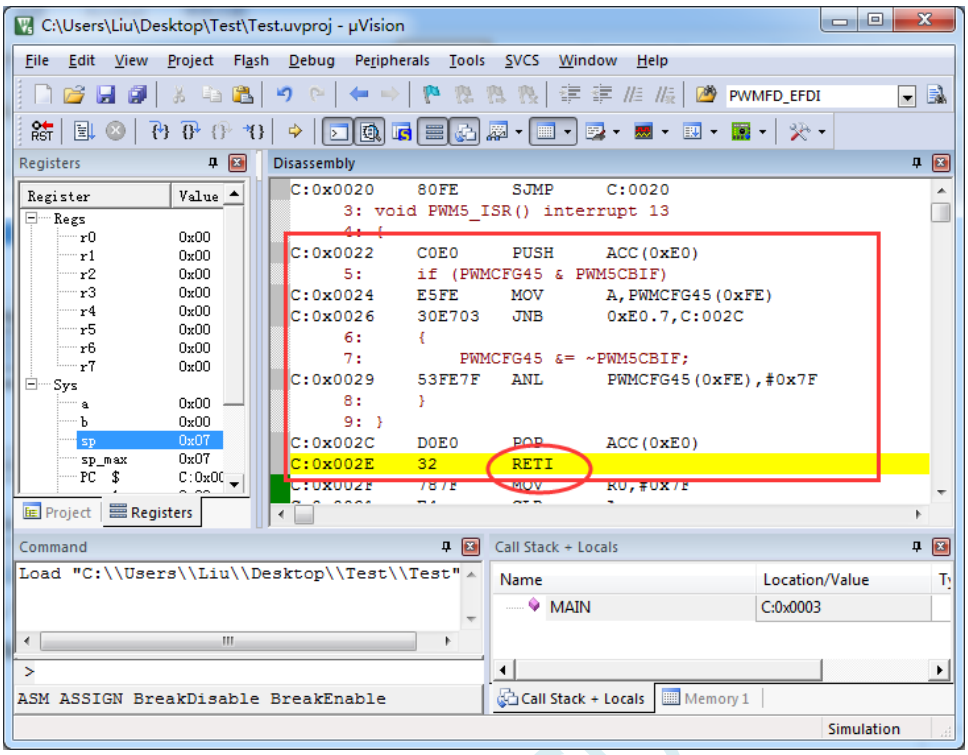



3、编译即可通过。

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：



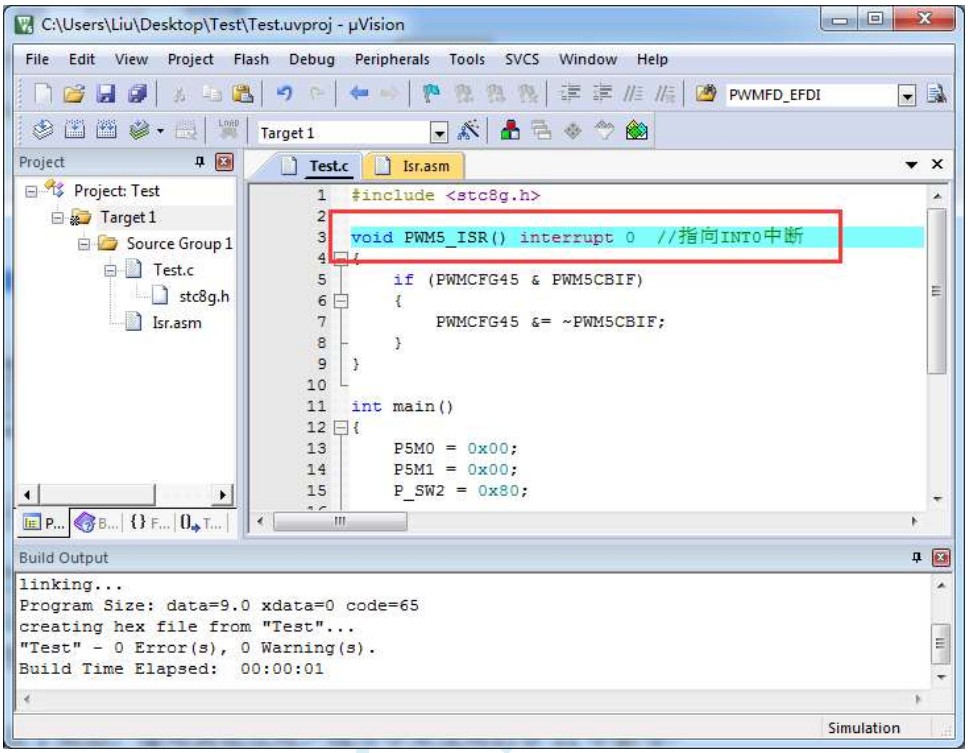

当发生 PWM5 中断时，硬件会自动跳转到 0103H 地址执行“LJMP 006BH”，然后在 006BH 处再执行“LJMP PWM5_ISR”即可跳转到真正的中断服务程序，如下图：

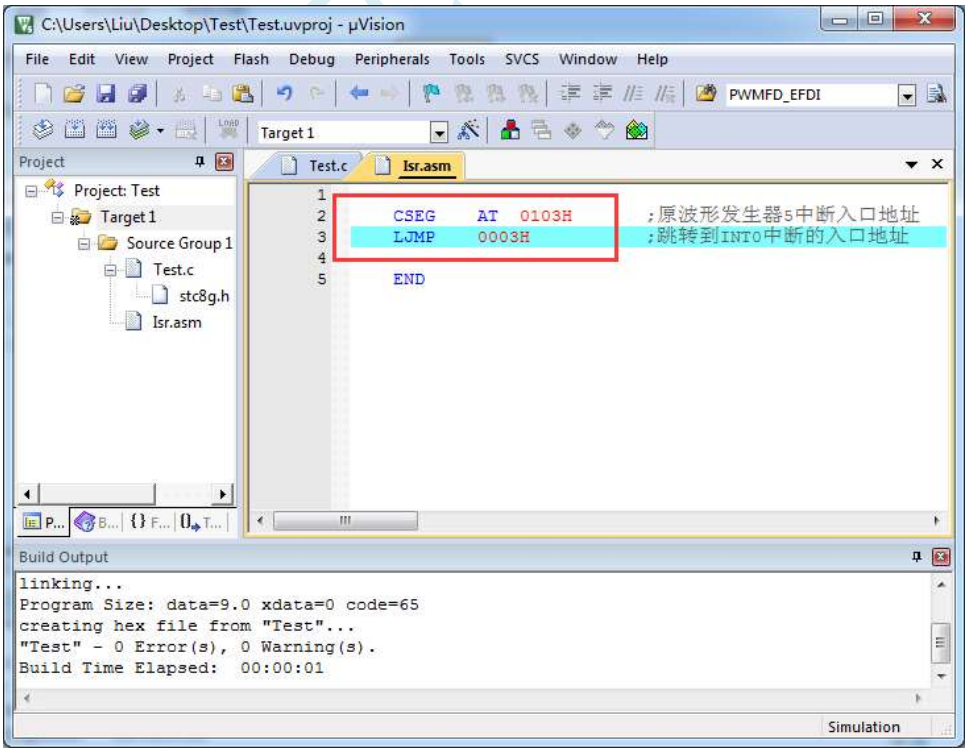


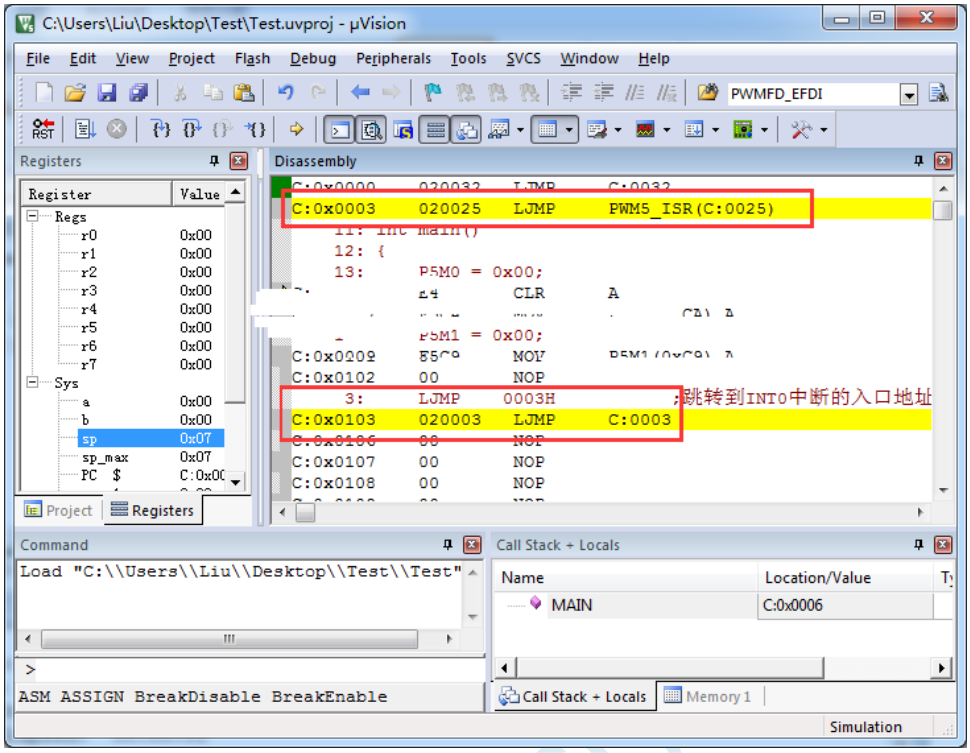

中断服务程序执行完成后，再通过 RETI 指令返回。整个中断响应过程只是多执行了一条 LJMP 语句而已。

## 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：



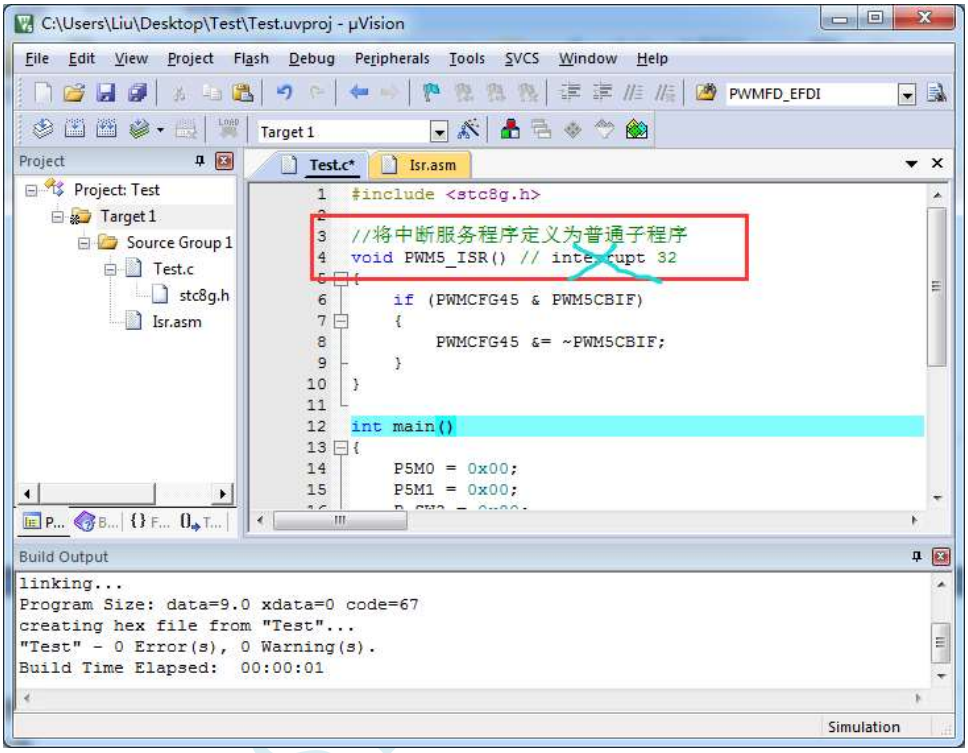



执行效果与方法 1 相同，此方法适用于需要重映射多个中断号大于 31 的情况。

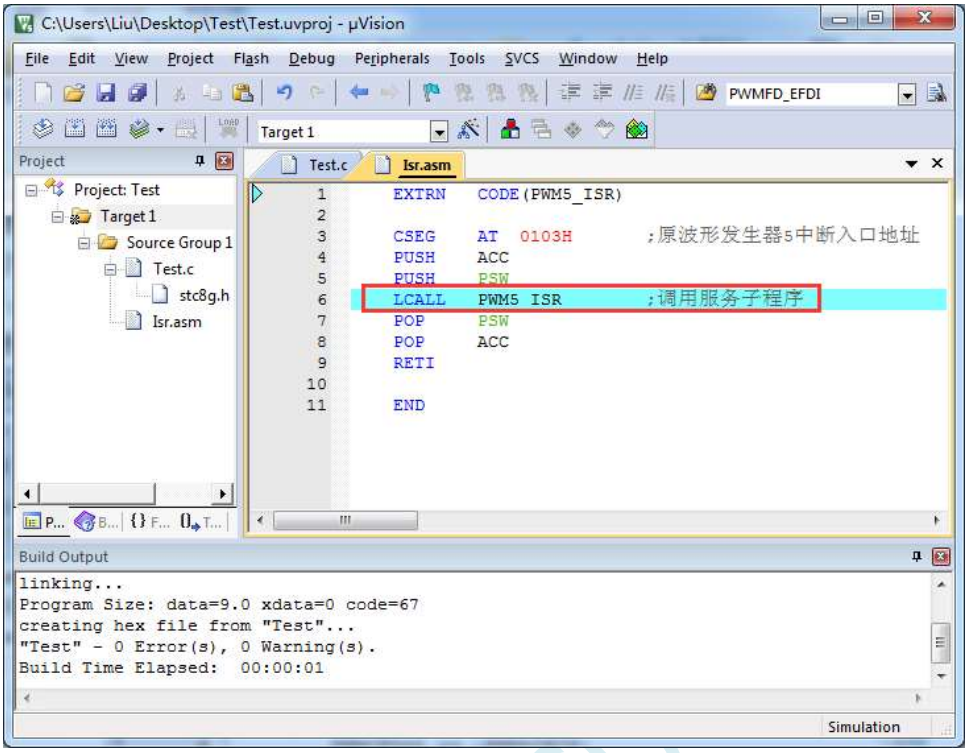
## 方法 3：将中断服务程序定义成子程序，然后在汇编代码中的中断入口地址中使用 LCALL 指令执行服务程序

操作步骤如下：

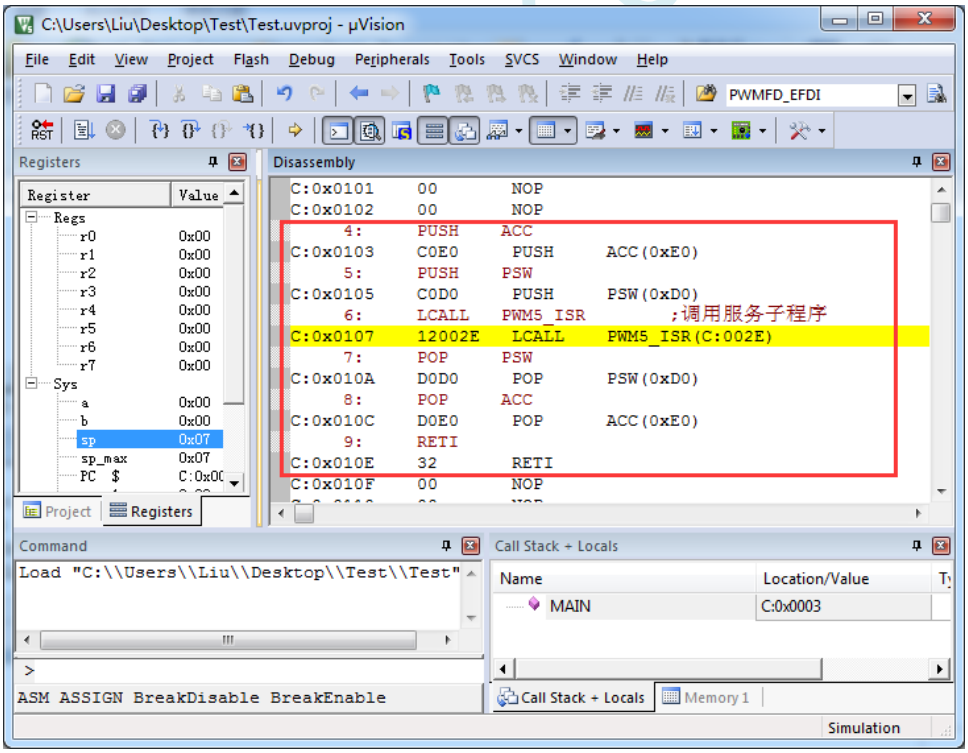
1、首先将中断服务程序去掉“interrupt”属性，定义成普通子程序




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R 电气特性

## R.1 绝对最大额定值

| 参数 | 最小值 | 最大值 | 单位 | 说明 |
|---|---|---|---|---|
| 存储温度 | -55 | +150 | ℃ | |
| 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。<br>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 | |

## R.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） | 200 | 270 | - | uA | |
| $I_{OH2}$ | 输出高电平电流（推挽模式） | - | 20 | - | mA | 端口电压 2.4V |
| $I_{IL}$ | 逻辑 0 输入电流 | - | - | 50 | uA | 端口电压 0V |
| $I_{TL}$ | 逻辑 1 到 0 的转移电流 | 100 | 270 | 600 | uA | 端口电压 2.0V |
| $R_{PU}$ | I/O 口上拉电阻 | 5.8 | 5.9 | 6.0 | KΩ | |
| I/O 速度 | I/O 大电流驱动，I/O 快速转换 | | 25 | | MHz | PxDR=0, PxSR=0 |
| | I/O 小电流驱动，I/O 快速转换 | | 22 | | MHz | PxDR=1, PxSR=0 |
| | I/O 大电流驱动，I/O 慢速转换 | | 16 | | MHz | PxDR=0, PxSR=1 |
| | I/O 小电流驱动，I/O 慢速转换 | | 12 | | MHz | PxDR=1, PxSR=1 |
| 比较器 | 最快速度 | | 10 | | MHz | 关闭所有模拟和数字滤波 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | 模拟滤波时间 | | 0.1 | | us | |
| | 数字滤波时间 | | 0 | | 系统时钟 | 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 | |

## R.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 | 270 | - | uA | |
| $I_{OH2}$ | 输出高电平电流（推挽模式） | - | 20 | - | mA | 端口电压 2.4V |
| $I_{IL}$ | 逻辑 0 输入电流 | - | - | 50 | uA | 端口电压 0V |
| $I_{TL}$ | 逻辑 1 到 0 的转移电流 | 100 | 270 | 600 | uA | 端口电压 2.0V |
| $R_{PU}$ | I/O 口上拉电阻 | 4.1 | 4.2 | 4.4 | KΩ | |
| I/O 速度 | I/O 大电流驱动，I/O 快速转换 | | 36 | | MHz | PxDR=0, PxSR=0 |
| | I/O 小电流驱动，I/O 快速转换 | | 32 | | MHz | PxDR=1, PxSR=0 |
| | I/O 大电流驱动，I/O 慢速转换 | | 26 | | MHz | PxDR=0, PxSR=1 |
| | I/O 小电流驱动，I/O 慢速转换 | | 22 | | MHz | PxDR=1, PxSR=1 |
| 比较器 | 最快速度 | | 10 | | MHz | 关闭所有模拟和数字滤波 |

| | | | | | | |
|---|---|---|---|---|---|---|
| | 模拟滤波时间 | | 0.1 | | us | |
| | 数字滤波时间 | | 0 | | 系统时钟 | 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 | |

## R.4 内部 IRC 温漂特性（参考温度 25℃）

| 温度 | 范围 | | |
|---|---|---|---|
| | 最小值 | 典型值 | 最大值 |
| -40℃～85℃ | | -1.38%～+1.42% | |
| -20℃～65℃ | | -0.88%～+1.05% | |

## R.5 低压复位门槛电压（测试温度 25℃）

| 级别 | 电压 | | |
|---|---|---|---|
| | 最小值 | 典型值（实测值） | 最大值 |
| POR | | （1.69V~1.82V） | |
| LVR0 | | 2.0V（1.88V~1.99V） | |
| LVR1 | | 2.4V（2.28V~2.45V） | |
| LVR2 | | 2.7V（2.58V~2.76V） | |
| LVR3 | | 3.0V（2.86V~3.06V） | |

# 附录S 应用注意事项

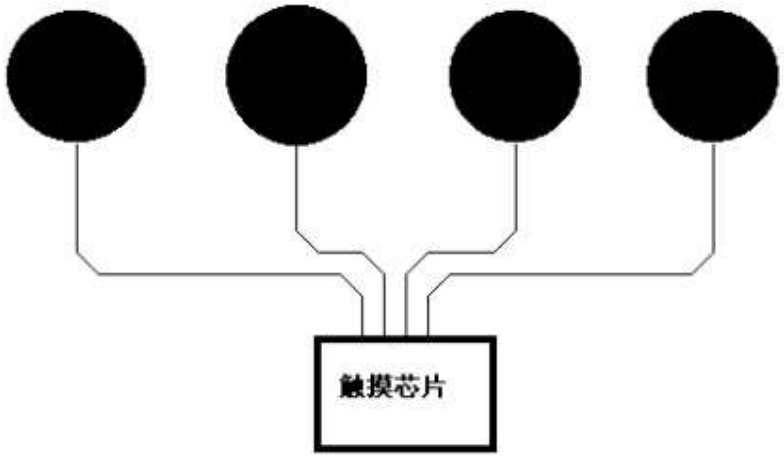
## S.1 STC8A8K64D4-64Pin/48Pin 系列

1. A 版芯片送样中

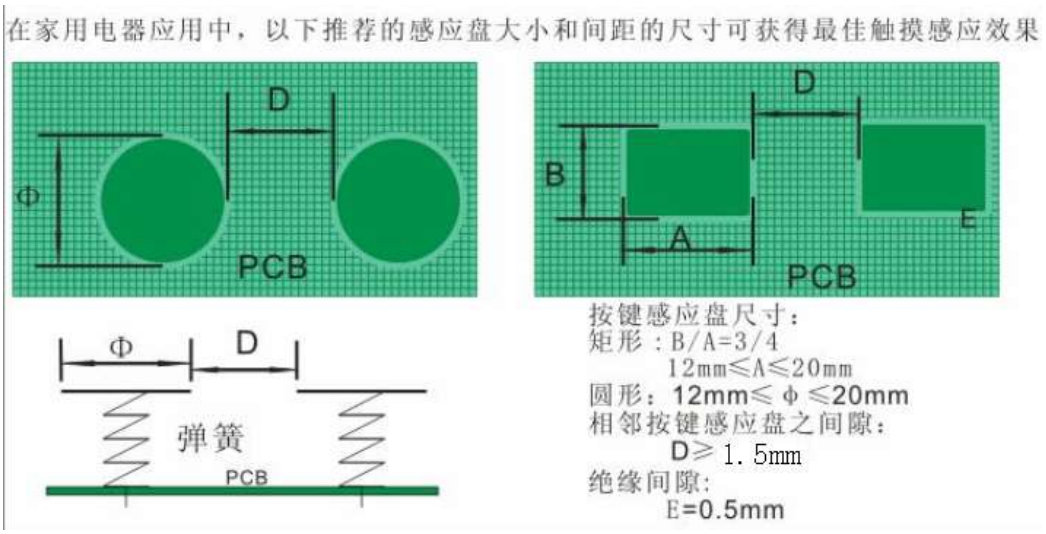
# 附录T 触摸按键的 PCB 设计指导

触摸按键对 PCB 设计的要求比较严格，否则其效果会大打折扣甚至失败。建议用户在设计 PCB 时遵循以下几点原则：

1. **遵循通常的数模混合电路设计的基本原则**。
   电容式触摸按键模块集成了精密电容测量的模拟电路，因此进行 PCB 设计时应该把它看成一个独立的模拟电路对待。遵循通常的数模混合电路设计的基本原则。
2. **采用星形接地**
   触摸芯片的地线不要和其他电路公用，应该单独连到板子电源输入的接地点，也就是通常说的采用“星形接地”。
3. **电源上产生的噪声对触摸芯片的影响**
   电源纹波、噪声应该尽量小，最好用一根独立的走线从板子的供电点取电并增加滤波措施，不要和其他的电路共用电源回路。
4. **IC 与感应盘的连线尽量等长，让其有近似的分布电容，入下图所示。**




5. **按键感应盘（电容传感器）大小和间隙**
   在满足面板的美学设计要求的情况下，必须通过合理安排的感应盘大小和间隔尺寸，来获得最佳的触摸感应效果。感应盘放在底层，IC 也放在底层，感应盘与 IC 连线不要有过孔。相邻感应盘边沿的间隔最好在 1.5mm 以上（下图中的尺寸 D），如果 PCB 面积允许，尽量取大一些间隔。铺铜与感应盘的间隔为 0.5mm（下图中的尺寸 E）。




6. **铺铜处理**

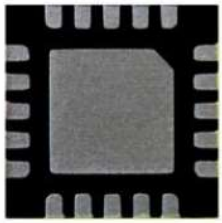
   底层可以铺网格铜或实铜均可，注意铺铜与感应盘的间隔为 0.5mm。顶层印刷按键的丝印信息，丝印的外框形状与底层感应盘一致，顶层对应底层感应盘的地方不能铺铜，否则会屏蔽掉触摸动作。顶层铺铜与底层铺铜一样即可。

7. **走线处理**

   感应盘与 IC 的连线使用比较小的线宽为好，比如 10~15mil 之间。感应盘到触摸芯片的连线不要跨越强干扰、高频、大电流的线。感应盘到触摸芯片的连线周围 1.5mm 内不要走其他信号线，越远离越好。顶层对应底层感应盘和连接线的地方，最好不要放任何线。

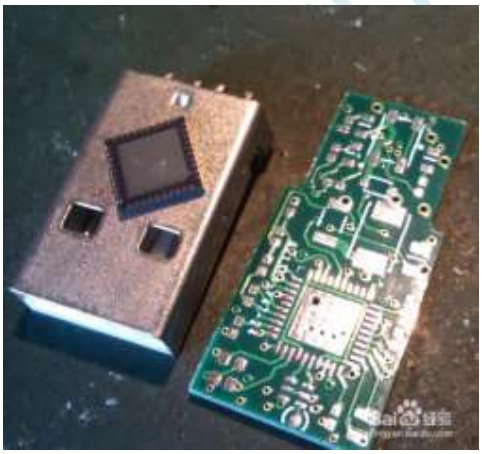
# 附录U QFN/DFN 封装元器件焊接方法

STC 产品的封装形式中，增加了现在比较流行的 QFN 和 DFN 的封装。由于这种封装形式的芯片芯片的管脚在芯片底部，手工焊接有一定的难度。市面上有专门做工程样品焊接的小公司，可承接工程样品打样。如用户需要自行焊接，可参考下面的焊接方法。



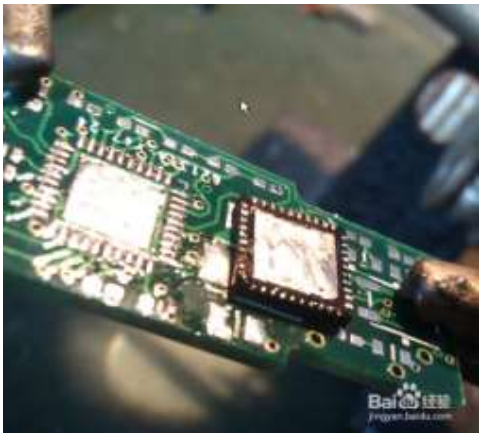
1、 首先需要准备如下工具：电烙铁、热风枪、镊子、固定架等工具

2、 需要焊接的 PCB 板和芯片如下图：



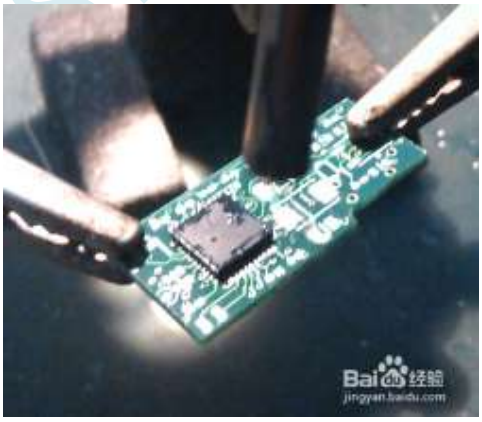
3、 先给板上芯片的焊盘上锡：


4、 然后给芯片底部上锡，这个上完锡后要弄平，尽量减少锡，但不能没有。



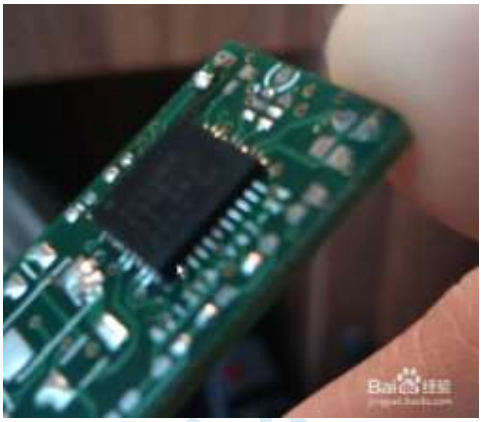
5、 调整热风枪温度，实际出风大概在 240 度左右，因为风枪质量不一样，根据实际情况调节。

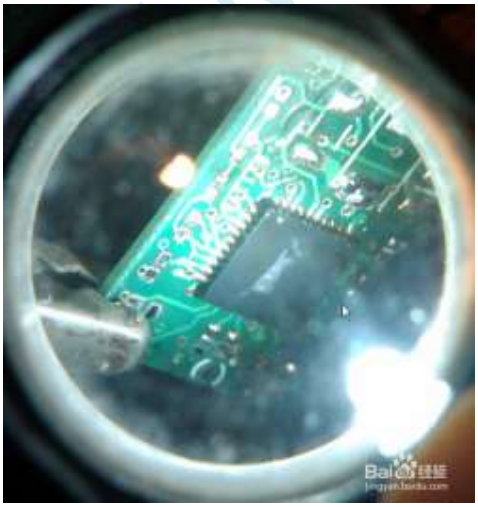

6、 把芯片放到焊盘上，一定要放正，然后用热风枪对着它吹，速度要均匀，直到锡溶化，一般 20 秒内。



7、 用烙铁给芯片侧引脚上锡

8、 焊接完成后的效果





# 附录V STC8A8K64D4 系列单片机取代 STC8A8K64S4A12 系列的注意事项

## ■ I/O 口

STC8A8K64D4 系列单片机上电后，I/O 的模式与 STC8A8K64S4A12 系列不一样。STC8A8K64S4A12 系列单片机所有 I/O 口上电后都是 8051 的准双向口模式，而 STC8A8K64D4 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。STC8A8K64D4 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 STC8A8K64D4 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 STC8A8K64D4 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

STC8A8K64D4 系列和 STC8A8K64S4A12 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 STC8A8K64S4A12 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 STC8A8K64D4 系列与 STC8A8K64S4A12 系列的复位电平是向反的，即对于 STC8A8K64D4 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

STC8A8K64S4A12 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，STC8A8K64D4 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）

## ■ ADC

STC8A8K64D4 系列的 ADC 在功能上完全覆盖兼容 STC8A8K64S4A12 系列。STC8A8K64D4 系列在 STC8A8K64S4A12 系列 ADC 的基础上新增了外部触发功能以及自动多次转换取平均值等新功能。

## ■ 比较器

STC8A8K64D4 系列的比较器正端输入为 4 路可选、负端输入为两路可选，输入选择设置在寄存器 CMPEXCFG 中。STC8A8K64S4A12 系列不一致。

## ■ SPI

STC8A8K64D4 系列的 4 种 SPI 时钟频率分别：SYSclk/4、SYSclk/8、SYSclk/16 和 SYSclk/2。STC8A8K64S4A12 系列的 4 种 SPI 时钟频率分别：SYSclk/4、SYSclk/8、SYSclk/16 和 SYSclk/32。

## ■ PCA/CCP/PWM

STC8A8K64D4 系列的 PCA 相关 SFR 中，前 3 组模块的 SFR 与 STC8A8K64S4A12 系列是相同的，第 4 组模块（PCA3/CCP3/PWM3）的控制寄存器在 XFR 区域，与 STC8A8K64S4A12 系列不兼容。（具体为 CCAPM3、CCAP3L、CCAP3H 和 PCA_PWM3）

## ■ 15 位增强型 PWM

STC8A8K64D4 系列的增强型 PWM 相关 SFR 地址与 STC8A8K64S4A12 系列不兼容。

# 附录W 更新记录

- **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. 更正文档中的笔误
  2. 更新选型价格表
  3. 更新官方网址
  4. 增加 STC-ISP 高级应用章节
  5. 增加关闭驱动程序强制数字签名说明章节

- **2022/9/2**
  1. 修正范例程序中声明变量位置的错误

- **2022/8/4**
  1. 更正 STC8A8K64D4 管脚图（P2.0 去除 RSTSV 功能）
  2. 修正比较器章节的错别字

- **2022/3/9**
  1. 更正文档中的笔误

- **2022/2/18**
  1. 更正文档中的笔误
  2. 修正 DMA 章节中 ADC 数据结构的描述错误部分

- **2021/12/17**
  1. 所有管脚图中复位脚名称修改为 NRST
  2. 修正定时器 2/3/4 的定时计算公式

- **2021/11/30**

1. 更正文档中的错别字
2. BMM 全部更名为 DMA
3. 增加 EEPROM 应用范例程序
4. 更新选型价格表
5. 增加 LQFP44 的封装尺寸图
6. 对只读特殊功能寄存器（CHIPID）增加详细说明
7. 增加附录“STC 仿真使用说明书”章节

- **2021/10/6**

8. 修改部分章节标题
9. 更正文档中已发现的错别字
10. 更新 LCM 章节中 8 位数据和 16 位数据的端口切换表格
11. 在中断系统章节的中断源表格中增加外部中断说明
12. 附录中仿真器章节的使用新版本软件进行截图
13. 在附录中增加“如何测试 I/O 口”章节

- **2021/9/26**

14. 更新特性及价格表中的部分描述
15. 更新技术支持电话
16. 增加串口 DMA 的超时处理和数据校验的范例程序
17. 在增强型 PWM 章节增加有关归零中断的使用注意事项

- **2021/8/26**

1. 修正 ADC 章节范例程序中的注释错误

- **2021/6/29**

1. 增加 LQFP44 管脚图

- **2021/6/26**

1. 修正比较器结构图中的错误描述
2. 修正增强型 PWM 输出频率计算公式

- **2021/5/10**

1. 增加定时器 2/3/4 中断标志位的相关说明
2. 修正串口 1/2/3/4 的 DMA 读取触发控制位

- **2021/4/28**

1. 创建 STC8A8K64D4 系列单片机技术参考手册文档

# 产　品　授　权　书

**致：江苏国芯科技有限公司**

STC8A8K64D4 系列产品的知识产权归深圳国芯人工智能有限公司所有。现授权江苏国芯科技有限公司可从事 STC8A8K64D4 系列产品在中国的推广和销售工作。

**授权单位： 深圳国芯人工智能有限公司**

**授权时限：2019 年 10 月 24 日 - - 2024 年 12 月 31 日**

# 自主产权，生产可控

深圳国芯人工智能有限公司是中华人民共和国大陆独资企业，按中国法律法规独立运营的企业，注册地址在深圳市前海深港合作区前湾一路 1 号 A 栋 201 室。

本手册所描述的器件是在中国境内自主研发，具备独立自主知识产权。

产品核心研发在中国境内，具备芯片设计、封装设计、结构设计、可靠性设计、器件仿真、工艺模拟等全部设计能力；产品核心研发团队人员及带头人全部为我国境内人员组成，其中研发团队带头人研发从业年限十年以上，具备长期、稳定的后续支持能力，具有在我国境内申请的专利证书及软件著作权等。

晶圆制造：本器件设计完成后的晶圆制造加工，在中华人民共和国大陆境内的晶圆厂加工制造完成，受中华人民共和国法律法规管理监管和控制，完全可控。

封装制造：本器件设计完成后的封装制造，在中华人民共和国大陆境内的封装厂加工完成，受中华人民共和国法律法规管理监管和控制，完全可控。

测试：本器件设计完成后的测试，在中华人民共和国大陆境内测试完成，受中华人民共和国法律法规管理监管和控制，完全可控。

本器件全部关键工艺均在我国自有生产线上完成，可以长期供货，无被断供的困扰。

特此说明。

深圳国芯人工智能有限公司