WinScope IDE

上海中基国威电子股份有限公司

SHANGHAI SINOMICON ELECTRONICS CO., LTD

2022 年 1 月 28 日

声明：本软件为上海中基国威电子股份有限公司研制并使用，公司保留对产品在可靠性、功能和设计方面的改进作进一步说明的权利。本文档的更改，恕不另行通知。

目　录

1 WinScope IDE 的简介

1.1 概述

WinScope IDE 集成开发环境(以下简称 WinScope 或 WinScope IDE)是上海中基国威电子股份有限公司为开发 SINOMICON 单片机产品而开发的一个可实时仿真的专用开发平台。让用户在开发和使用上更加的便捷，WinScope IDE 提供好友的视窗界面，以便于进行程序的编辑、编译及除错，同时配合我们自主研发的 SN-Link 仿真器，提供了对 MCU 的多种实时仿真功能，包括 Trace 跟踪、步进(StepIn)、步越(StepOver)、步出(StepOut)、断点设定等功能。WinScope IDE 开发平台提供即插即用的 USB 接口，并通过网格检测不定期更新软件服务包，通过免开壳在线升级仿真器，以保证设计者可以拥有功能齐全，版本最新的开发工具、以提高产品应用方案的开发效率。

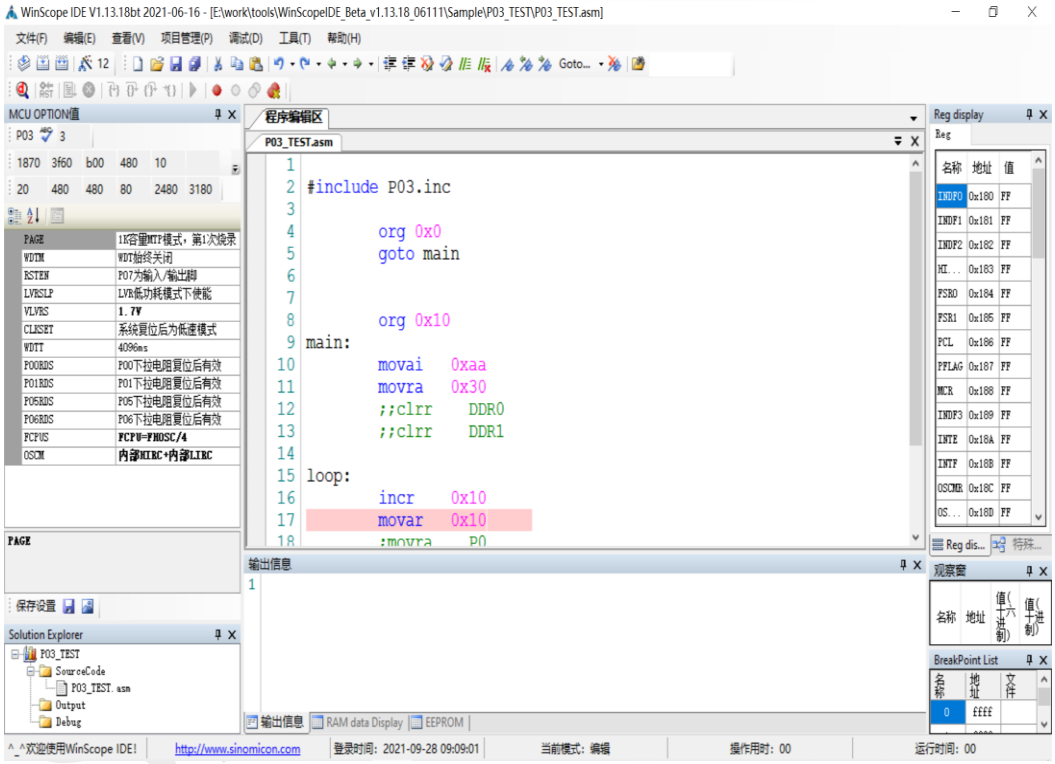


图 1 WinScope IDE 界面

WinScope IDE（WinScope Integrated Development Environment）主要的组件是 SN-Link 仿真器，它提供了上海中基 OTP 系列单片机的实时仿真功能。不同的 MCU 型号通过外接的 EV 小板而定。该开发环境可以完成从项目的建立和管理，编译，目标代码的生成，到仿真等完整的开发流程。WinScope IDE 开发平台具主要有以下功能特点：

硬件：

- USB 接口，方便与 PC 连接
- USB 供电，应用板功率小于 100MA 时无需外部单独供电
- 仿真器内部提供可调的振荡器频率，一般无需外接晶振
- 支持最多 15 个硬件断点

- 实时显示单片机运行状态
- 同时支持公司现有 SM11、SM51 系列单片机仿真

软件:

- 友好的视窗软件界面，免除开发环境的熟悉周期
- 工程项目管理可以随时添加或删除项目文件
- 功能强大的代码编辑器有效提高编程速度
- 提供智能提示，代码折叠，关键字高亮显示
- 编辑器提供快速查找替换，正则表达式查找功能
- 支持源代码级调试仿真
- RISC 内核单片机支持多个源文件，程序模块化
- RISC 内核单片机支持建立自己的库文件，头文件
- 编译出错行号提示 和自动建立连接对应
- 仿真过程中可以随时修改 RAM，REG 等参数

1.2 组成

WinScope IDE 的组成如表 1.1 所示。

功能组成	项目管理、程序编辑器、编译及目标代码生成、程序调试、硬件仿真器
界面组成	标题栏、菜单栏、右键菜单、工具栏、窗口、状态栏
模式组成	编辑模式、调试模式

编辑模式与调试模式：编辑模式是用于维护文件、编写程序的，调试模式是在连接好硬件仿真器后，用来仿真调试程序的。

1.3 开发流程

使用 WinScope IDE 进行单片机开应用开发的步骤如下:

新建项目，选择芯片型号，保存路径，设置配置选项
新建 ASM/C 文件 并添加到工程
打开 ASM/C 文件，进行代码编辑
编译和构建(链接)工程
纠正程序中书写和语法错误，并重新编译连接
连接仿真器，下载程序到仿真器，进入仿真模式
对程序进行仿真调试
仿真通过后将生成的 S19 文件使用烧写器和烧写软件 烧到单片机中

1.4 WinScope IDE 的安装与配置

1.4.1 安装要求

WinScope IDE 软件必须要求满足最小系统为：

- 操作系统：Win XP、Win 7、Vista、Win8.1、Win10；
- 硬盘空间：300MB 以上
- 内存：256MB 以上

1.4.2 安装

目前软件为免安装版，但要求操作系统必须安装 Framwork .net 3.5 组件，如果电脑中没有此组件，推荐到微软官方下载。下载。Framwork.net 3.5 组件链接地址：
http://download.microsoft.com/download/6/0/f/60fc5854-3cb8-4892-b6db-bd4f42510f28/dotnetfx35.exe

当用户接上 SN-Link 仿真器时，如果当前电脑是第一次连接仿真器，电脑会提示发现新硬件。此时需要安装 USB 驱动。驱动程序放在 WinScope IDE 软件包 Driver 目录下，安装步骤如下：(以 Win7 系统为例)：

a. 当系统提示从 Windows Update 中获取驱动软件时，选择跳过。如**错误!未找到引用源。**：

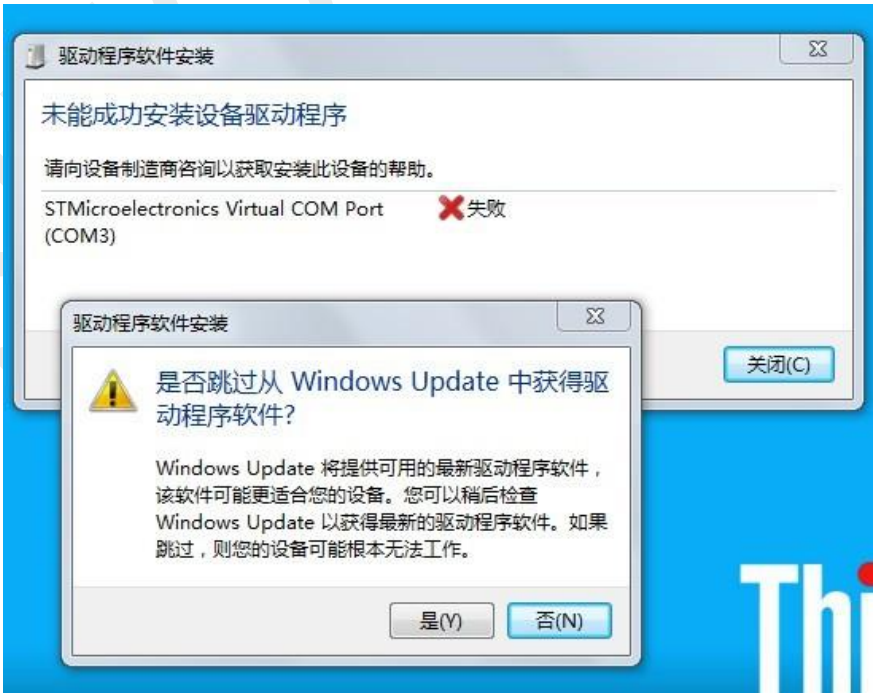


图 2

b.打开设备管理器。控制面板--》系统和安全--》系统 --》设备管理器。出现如下图 3 所示对话框：

图 3

c.在设备资源管理器中，找到 “端口 (COM 和 LPT)”(请使用 COM1~COM5) 如下所示图 4：

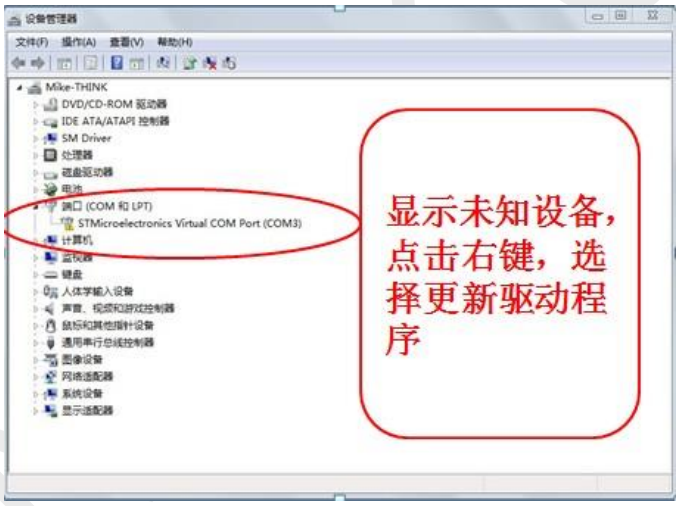


图 4

d.点击右键后，出现如下话框，按红字部分提示操作。如图 5：

图 5

e.选择软件目录下的 Driver 文件夹，然后下一步。驱动安装成功后，会出现如下图 6 示提示：

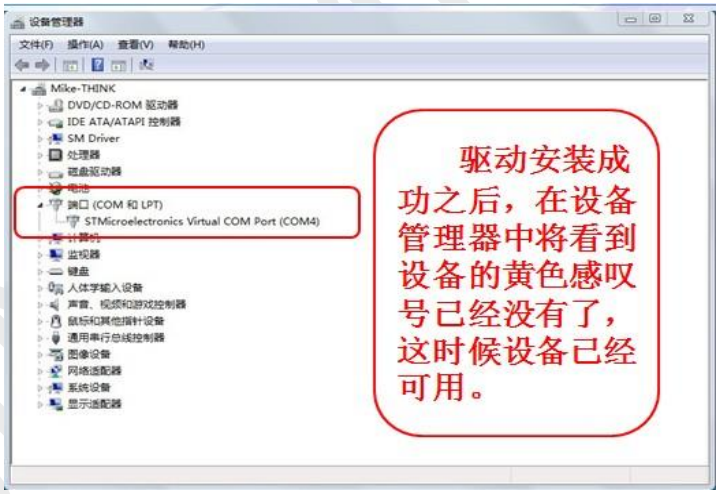


图 6

这时，USB 的驱动已经安装完成。程序编译通过后，直接点点进入调试模式即可进行仿真。

2 WinScope IDE 的界面

WinScope IDE 界面由标题栏、菜单栏、右键菜单、工具栏、窗口、状态栏组成。

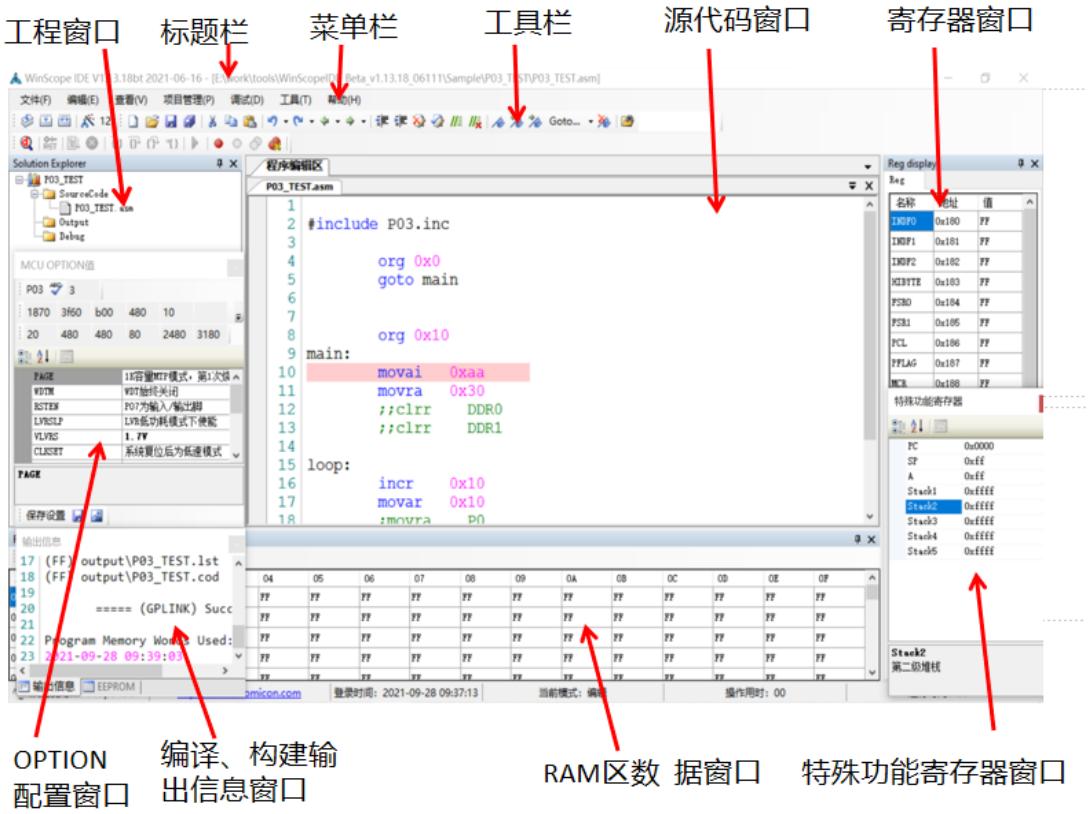


图 7 WinScope IDE 主界面

2.1 窗口

WinScope IDE 的工具栏可以用鼠标左键进行拖动，项目管理器窗口、MCU OPTION 值窗口、程序编辑区窗口、寄存器窗口、特殊功能寄存器窗口、断点设置列表窗口、信息输出窗口、RAM数据窗口等显示的窗口都可以浮动、停靠、隐藏，如图 8 所示。这些窗口通过菜单栏的【查看】菜单操作可以关闭或打开。并且软件会自动保存新的布局。

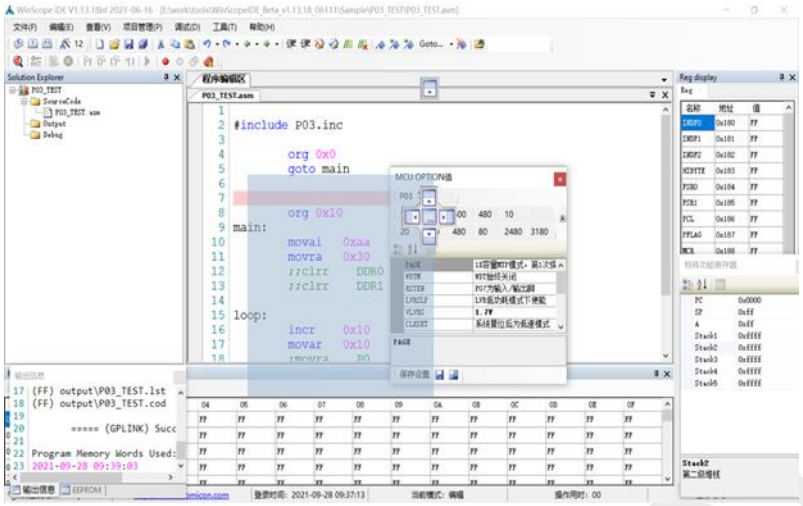


图 8 WinScope IDE 主界面

2.2 菜单的说明

2.2.1 文件菜单

菜单项	快捷键	图标	功能描述
新建/项目			创建项目/文件
打开/项目			打开已有项目、文件
关闭			关闭当前窗口
关闭项目			关闭整个项目
保存			保存当前文件
另存为			另外取名保存当前文件
全部保存			保存所有已打开的文件（工程及当前文件）

2.2.2 编辑

菜单项	快捷键	图标	描述
撤消	Crtl+Z		撤消上一次操作
重复	Ctrl+Y		重做上一次操作
剪切	Ctrl+X		将选中的文本剪切到剪贴板
复制	Ctrl+C		将选 中的文本复制到剪贴板
粘贴	Ctrl+V		粘贴剪贴板的文件到当前光标处

删除	Delete		
全选	Ctrl+A		选中文本编辑窗所有内容
快速查找	Ctrl+F		查找当前文件中相关字附
查找一下个	F3		查找下一个匹配字符串
快速替换	Ctrl+H		替换已找到的字符串
高级----			高级菜单
转换为大写			将选定的内容转换为大写
转换为小写			将选定的内容转换为小大写
注释选定内容			将选定内容注释
取消注释选定内容			取消选定内容的注释
增加行缩进	Tab		增加选定内容的行缩进
减少行缩进	Shift+Tab		减少选定行的缩进
向前定位	Alt+左方向键		向前定位，将光标定位到上一位置
向后定位	Alt+右方向键		向后定位，将光标定位到上一位置
添加/移除书签	Ctrl+F2		在当前行放置书签/移除书签
上一书签	Alt+F2		移动光标到上一书签
下一书签	F2		移送光标到下一书签
清除所有书签			清除当前文件的所有书签
默认快捷键功能	Home		将光标移到当前行的开始
	End		将光标移到当前行的结尾
	Ctrl+Home		将光标移到当前文件的开始
	Ctrl+End		将光标移到当前文件的结尾
	Ctrl+左方向键		将光标移到当前单词左侧
	Ctrl+右方向键		将光标移到当前单词右侧
	Ctrl+K		在编辑状态下，出现关键字提示
	Esc		在编辑状态下，隐藏关键字提示

2.2.3 查看菜单

项目管理			打开项目管理窗口
子程序列表			显示各子函数列表
构建输出窗			打开输出信息窗口
寄存器			打开功能寄存器窗口
特殊功能寄存器			打开特殊功能寄存器窗口
监视窗口			打开监视窗口，添加临时变量观察窗
断点列表			打开断点列表
RAM 区数据			打开 RAM 区数据，观察 RAM 区的数据变动
芯片配置			打开芯片设置窗，修改芯片的 OPTION 设置

2.2.4 项目管理菜单

新建项目			创建项目
打开项目			打开已有项目
关闭项目			关闭整个项目
编译/汇编			编译当前文件
生成 S 代码			编译所有 ASM/C 文件并生成 S19 文件
重新生成代码并下载			重新生成 S19 文件并下载

2.2.5 调试菜单

启动/停止调试	Ctrl+F5		启动或停止调试模式
CPU 复位			使单片机 PC 回到开始位置
全速运行	F5		全速运行直到下一有效断点
停止运行	Shift+F5		停止运行
步进	F11		单步运行，遇到函数则进入函数内部单步运行
步越	F10		单步运行程序，遇到函数不进入
步出	Shift+F11		跳出当前子函数，回到上一级程序的下一语句
运行到光标处[1]	Ctrl+F10		运行到光标所在行
禁止断点且运行			全速且不响应断点

放置/移除断点[2]	F9		在当前行插入或移除断点
禁止/允许断点	Ctrl+F9		使当前行的断点有效或无效
禁止所有断点			使整个工程的断点无效
清楚的所有断点	Ctrl+Shift+F9		清楚整个工程的所有断点

2.2.6 工具菜单

合并 S19 文件			
选项			设置字体、制表符大小、语言选择
仿真系统选项			非程序区填充功能、芯片电压选择功能

2.2.6.1 非程序区填充功能

功能说明，如图 2.2.6 所示：
功能一：非程序区，是否需要填充。
功能二：填充数据格式，可选全 0，或全 1。
该功能设置，记忆在项目工程文件中，不同的项目工程，可设置不同的选项。

2.2.6.2 电压选择功能

仅支持 V0.7 及以上仿真器；该功能设置，记忆在项目工程文件中，不同的项目工程，可设置不同的选项。

注：仿真器硬件为 V0.5 时，该电压选择无效。V0.5 的仿真器，电压输出可进行手动调节。

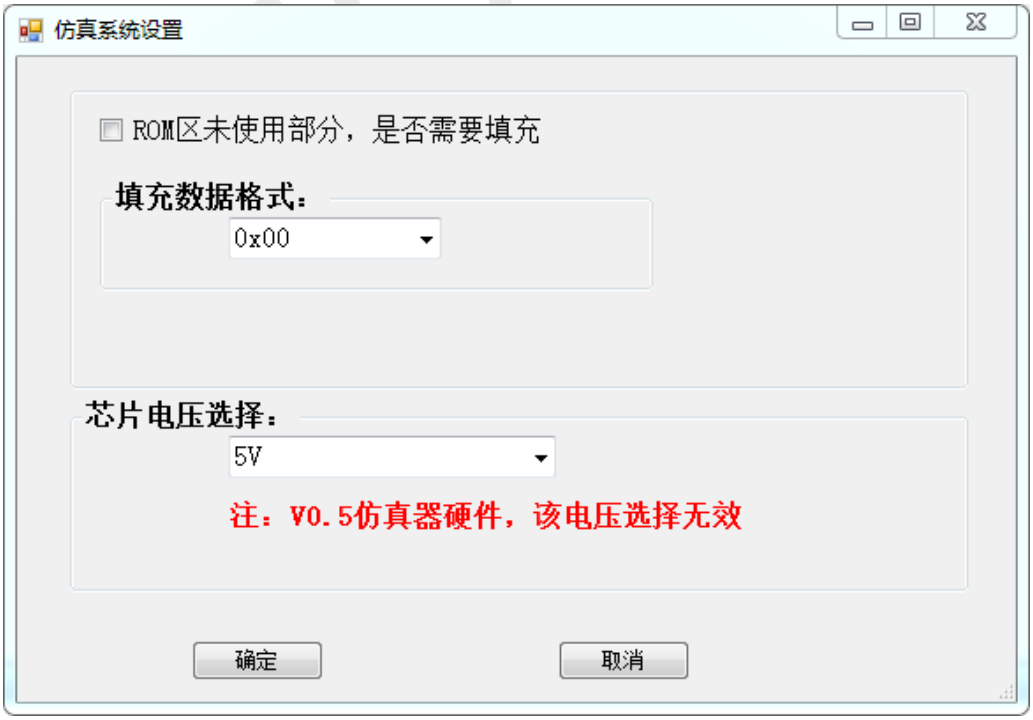


图 2.2.6　仿真系统设置

2.2.7 帮助菜单

帮助			
检查更新			查看更新到最新版本
在线注册			
关于 IDE			版本信息

注：[1]运行到光标处，目前不可以使用。
[2]放置/移除断点，该功能可以直接在程序编辑区的代码行标号前点击设置或移除。

2.3 右键菜单

WinScope IDE 提供了右键菜单，在程序编辑区可以使用右键菜单，方便快速执行命令（图 2.4 所示）。

图 2.4　右键菜单

3 WinScope IDE 项目管理

WinScope IDE 目前支持的是单个项目的管理。要建新项目时，可以直接按“新建项目”的菜单，当前的文件自动被关闭，或者先关闭当前的项目后再新建项目；如果要打开另一个项目，也可以按添加项目一样添加，当前的项目也会关闭；或者先关闭当前项目再添加。

3.1 项目窗口

项目窗口的内容包括项目名称、源代码文件夹、Output 文件夹、Debug 文件夹。程序文件必须添加到 SourceCode 文件夹下。当双击 ASM 文件可以在程序编辑区打开对应的文件。如下图 9

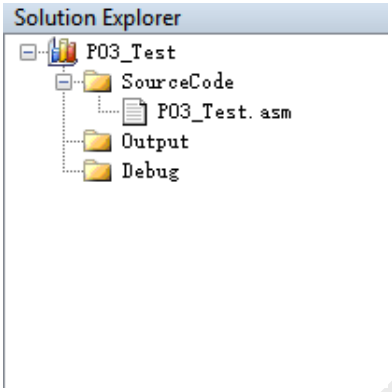


图 9

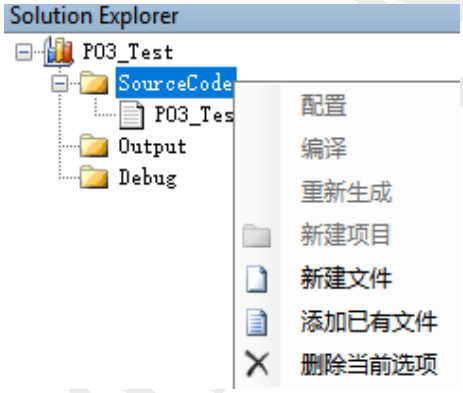


图 10

右键单击项目名图标，可以进行对项目进行编译和重新生成，新建文件夹。当右键选中 SourceCode、Output、Debug 中的一个文件夹后可以新建或者添加已有文件到项目中。目前只支持添加存放于工程目录下的文件，其它路径文件将会在打开时报错。 如图 10 的下拉菜单，如图要删除某个 ASM 文件时，先选 中 ASM 文件然后在对应的文件夹上右键选择删除即可。

3.2 新建项目

如图 11 选择主菜单【文件】\【新建】\【项目】或者【项目管理】\【新建项目】,进入到新建项目的向导对话框，如图 12,在对话框中,可以选择芯片的型号、命名项目的名称、选择项目的存放路径，然后进入下一步设置 MCU OPTION 值，这些操作完成后进入下一步会显示上面操作的所有信息,如项目名称、存放路径等。完成后的项目管理窗口如图 13(例:项目取名 P03_Test)，但还需要新建文件才能在程序编辑区写源代码,选中 SourceCode 后右击会弹出一个下拉菜单，选择“新建文件”命名（例：文件命名为 P03_Test.asm）保存后，就可以在程序编辑区编写源代码了（如图 14)。如果需要删除文件（如：P03_Test.asm)，先选中需要删除的文件，然后再 SourceCode 上右击出现的下拉菜单，点击“删除当前选项”就可以删除文件。

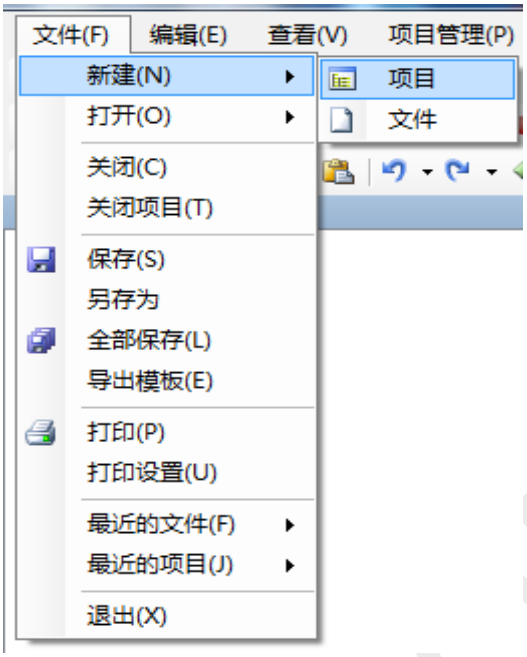


图 11

图 12 新建项目向导

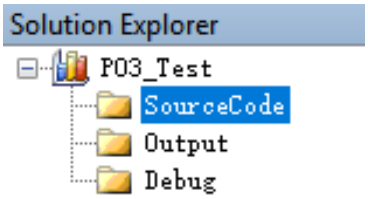


图 13 项目窗口

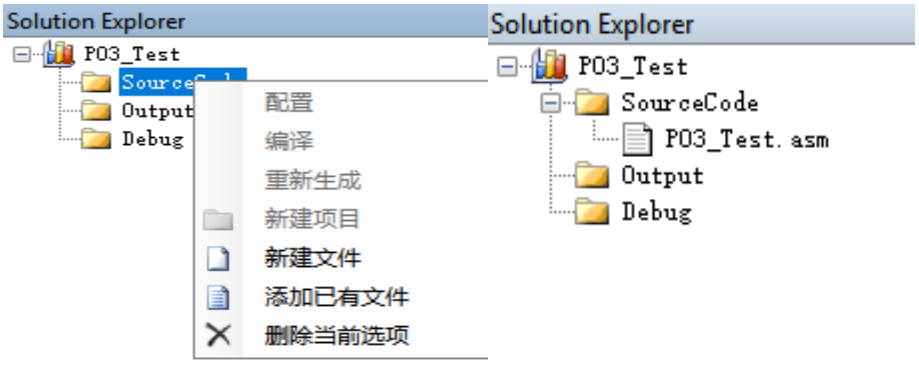


图 14 新建文件

3.3 打开已有项目

选择主菜单【文件】\【打开】\【项目\解决方案】或者【项目管理】\【打开项目】,将已有的项目添加到项目管理窗口。

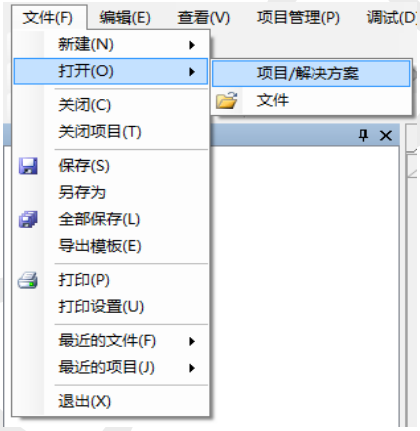


图 15 项目添加

3.4 MCU OPTION 值设置

在新建项目过程中的新建项目向导下，选择好芯片型号，命名好项目名称、选择好项目的存放路径后，点击“下一步”，弹出 MCU OPTION 值的设置对话框。芯片型号在弹出 MCU OPTION 值对话框之前已经选好，MCU OPTION 值可以点击 按分类顺序排列，也可以点击 按字母顺序排列，根据开发项目的需要在这里配置 OPTION 值。当选中 OPTION 值的某一项时，将会在显示窗下方出现对应项的配置说明如下图 16，选中 WDTE 后，在下方出现“看门狗设置：True：使能 WDT False：关闭 WDT”的说明。OPTION 值可以请允许开发过程中随时进行修改，不过需要点击下方的保存设置后才会更新到项目中。如果不点击保存，重新进入仿真模式时是会按照当前显示的值进行重新设置的，但此 OPTION 值不会被保存到项目中，下次打开时仍然是重新设置的值。

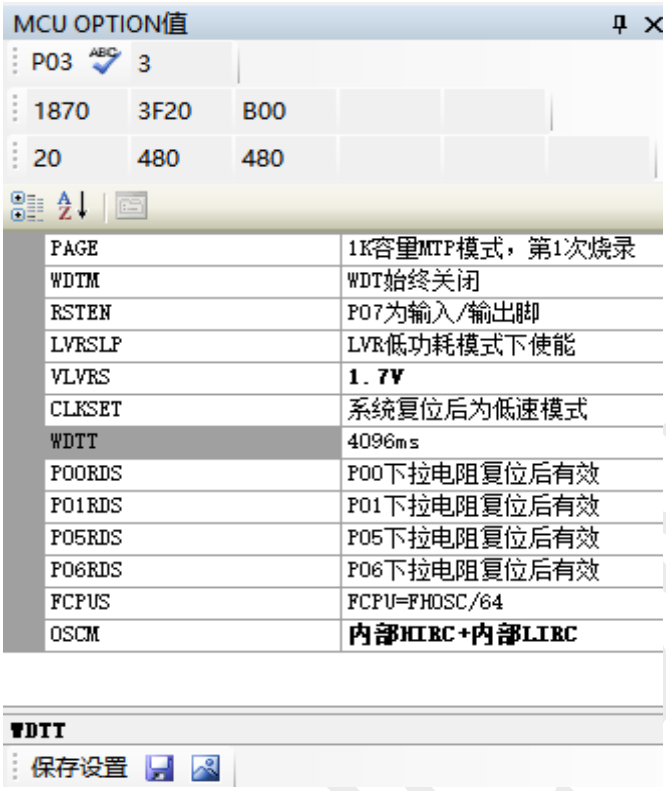


图 16 MCU OPTION 值设置

3.5 编译项目

在程序编辑区中编写好源代码保存后，编译文件，点击 是编译当前文件，在编译输出窗口会提示编译成功或者编译出错，编译成功后如图 17 所示。当编译出错时，WinScope IDE 会跳出错误提示框提示编译程序出错。

```
(FF) System\P03_Test.o
(FF) System\P03_Test.lst

                    errors: 0
                  warnings: 0
                  messages: 0
        warnings suppressed: 0
        messages suppressed: 0

===== (GPASM) Successful =====
```

图 17 编译当前文件

如果点击 ，会编译项目中的所有文件并生成 S19 文件，在编译“输出信息”窗口中也会提示项目下所有文件的编译情况。点击 会编译项目中的文件、生成 S19 文件并自动进

入仿真模式。当编译出错时，WinScope IDE 将自动停止仿真器的连接，退出仿真模式。每次修改代码后都需要编译下载 S19 文件，这样才能够更新仿真结果。

4 WinScope IDE 调试

4.1 WinScope IDE 调试

WinScope IDE 可以通过设置断点、步进、步越、步出、运行到光标处、禁止断点且运行等方式进行调试。菜单如图 18

图 18 调试菜单

4.1.1 启动/停止调试

当程序编译成功并设置好调试目标后，选择【调试】\【启动/停止调试】或者直接点击  ，程序将进入调试状态，当需要退出调试模式时，同样是选择【调试】\【启动/停止调试】或者直接点击  ，如果是出于全速运行状态，则需要先停止运行再停止调试。启动/停止调试快捷键是 Ctrl+F5。

4.1.2 断点、禁止断点且运行

断点使得调试程序方便了很多，能够在需要的位置暂停执行，与单步不同的是可以让程序一直运行到断点处才暂停，加快了调试的过程。无论处于编辑模式还是调试模式，在程序编辑区的代码行前面（快捷键 F9），都可以设置/消除断点。

禁止断点且运行是执行程序时忽略已经设置的断点，直到有停止运行命令才会停止。

4.1.3 复位、全速运行、停止运行、运行到光标处

复位会使单片机的 PC 回到开始位置；全速运行是指运行程序直到碰到断点或停止运行命令才会停止，快捷键和图表分别是 F5、  ；停止运行是停止当前运行的程序，直到再次执行运行命令，图表是  ；运行到光标处（只有 HC05 核系列有用，其它系列的不可用）是指全速运行程序到光标所在行然后停止，如果运行过程中有断点会停在断点所在行（快捷键是 Ctrl+F10）。

4.1.4 步进、步越、步出

步进是逐句的执行指令，遇到函数调用则进入函数内部执行（快捷键 F11）。步越不响应子程序是指逐句的执行指令，遇到函数调用不进入内部执行，而是将函数当一条语句执行（快捷键 F10）。步出是跳出当前函数或子程序是执行指令到当前函数的结束行然后停止（快捷键 Shift+F11）。

注意：当程序在主程序中运行时，不允许点击“步出”菜单，否则会产生意想不到的情况。

4.1.5 放置/移除断点、禁止/允许断点、禁止所有断点、清除所有断点

放置/移除断点，无论处于编辑模式还是调试模式，在程序编辑区的代码行前面（快捷键 F9），都可以设置/移除断点，还可以直接在代码行数字标号前空白处单击来设置断点或去除断点。清楚所有断点，将已经设置的断点全部清除。

4.2 观察调试信息

4.2.1 寄存器窗口

寄存器窗口显示了目标 CPU 的寄存器在调试过程中的状态，只在调试模式时可用。通过寄存器窗口可以实时观察寄存器的值，当寄存器的值被改变时会以红色的形式突出显示。另外通过双击数值项可以修改寄存器的值。

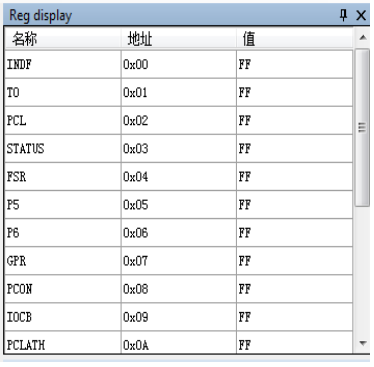
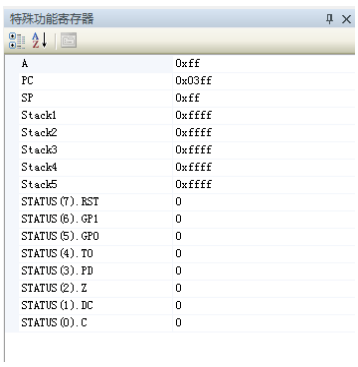


图 19 寄存器窗口



特殊功能寄存器窗口

4.2.2 特殊功能寄存器窗口

特殊功能寄存器窗口显示目标单片机在调试过程中的状态，此窗口为只读窗口，调试过程中不允许修改窗口中的参数。

4.2.3 可读写数据存储窗口

可读写数据存储窗口（RAM 区）如图 20，在调试时可用，该窗口主要用于观察连续内存。可以根据需要，在地址框输入地址后回车，能够查找对应地址的位置。显示窗中的数值以十六进制显示。如果要改变 RAM 区的内容，将光标定位在要修改的地方，直接修改，这里的数据可以是两种形式，比如 01 可以写成 0x01,只能输入十六进制数，且小于或等于 0xff。 超过 0xff 会跳出如图 21 对话框。

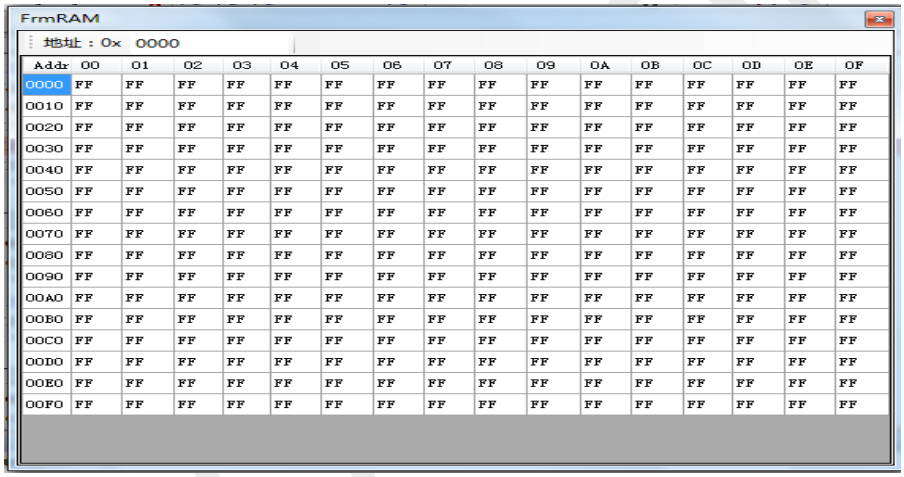


图 20 RAM 数据存储窗口

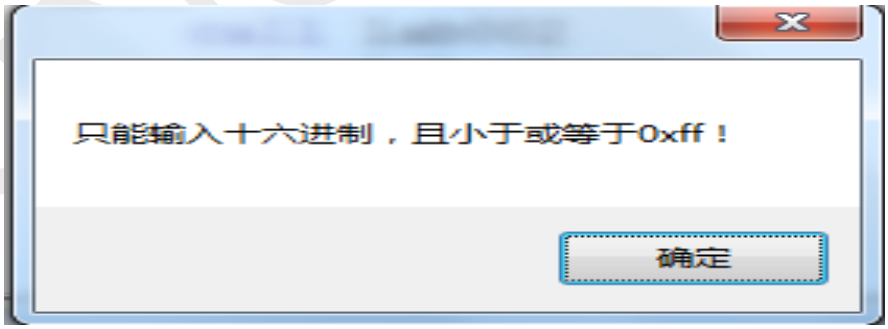


图 21 RAM 数据存储对话框

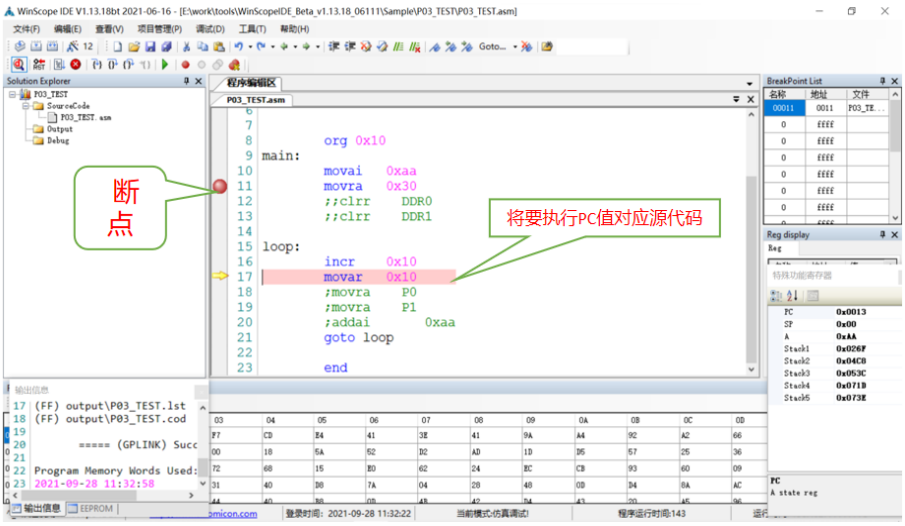


图 22 仿真模式下视图

状态栏:

当前模式:包括编辑模式和仿真调式两种,其中仿真调试中分为 Busy,Stop,Wait,Monitor 几种状态。

5 SN-Link 仿真器

5.1 SN-Link 仿真器的构成

SN-Link 仿真器配件如下：

- A-B USB 线 1 根
- 主机 1 台
- 对应 MCU 型号仿真 EV 板 1 块
- 连接目标板排线 1 根

图 23 SN-Link 实物图

如图 23 所示 标注 USB 的一端直接与电脑 PC 机相连，标注 JTAG 接口的一端与各种型号的 EV 板相连，针对不同的型号使用相对应的 EV 板进行仿真，EV 板上提供了对应芯片的引脚，可用排线直接连到的用户的目标板上。用户可以选择目标板由仿真器供电或者单独供电。

5.2 各型号 EV 板说明

5.2.1 P01

PCB 标注丝印	功能说明
J2	当目标板需要仿真器供电时，需要短接一起
14PIN 8PIN 6PIN 6PIN	对应规格书各种脚位排列 接口
Y1	当选择外振时，需要焊接对应频率的晶振
J3	如果外接晶振为 8M，则可以直接从仿真器引入，引入时把 J3 短接一起即可。
J1	与 SN-Link 仿真器接口

5.2.2 P02

PCB 标注丝印	功能说明
J2	当目标板需要仿真器供电时，需要短接一起
10PIN 8PIN 8PIN 6PIN	对应规格书各种脚位排列 接口
Y1	当选择外振时，需要焊接对应频率的晶振
J1	与 SN-Link 仿真器接口

5.2.3 P03

PCB 标注丝印	功能说明
U5	当目标板需要仿真器供电时，需要短接一起
U8, J2, U4	对应规格书各种脚位排列 接口
Y2	当选择外振时，需要焊接对应频率的晶振
U6	如果外接晶振为 16M，则可以直接从仿真器引入，引入时把 U6 及 R4 短接一起即可。
J1	与 SN-Link 仿真器接口

5.2.4 P04

PCB 标注丝印	功能说明
U5	当目标板需要仿真器供电时，需要短接一起

J2 J3 J4	对应规格书各种脚位排列 接口
Y1	当选择外振时，需要焊接对应频率的晶振
J1	与 SN-Link 仿真器接口

5.2.5 P05

PCB 标注丝印	功能说明
J2	当目标板需要仿真器供电时，需要短接一起
40PIN	对应规格书各种脚位排列接口
Y1	当选择外振时，需要焊接对应频率的晶振
J3	如果外接晶振为 8M，则可以直接从仿真器引入，引入时把 J3 短接一起即可。
J1	与 SN-Link 仿真器接口

5.2.6 P06

PCB 标注丝印	功能说明
J2	当目标板需要仿真器供电时，需要短接一起
U2,U3	对应规格书各种脚位排列接口
Y1	当选择外振时，需要焊接对应频率的晶振
J1	与 SN-Link 仿真器接口

5.2.7 P07

PCB 标注丝印	功能说明
J2	当目标板需要仿真器供电时，需要短接一起
24PIN 14PIN	对应规格书各种脚位排列接口
Y1	当选择外振时，需要焊接对应频率的晶振
J1	与 SN-Link 仿真器接口

5.2.8 P08

PCB 标注丝印	功能说明
J2	当目标板需要仿真器供电时，需要短接一起

P1	对应规格书各种脚位排列 接口
J1	与 SN-Link 仿真器接口

5.2.9 SM110X

PCB 标注丝印	功能说明
J2	当目标板需要仿真器供电时，需要短接一起
14PIN 8PIN 6PIN 6PIN	对应规格书各种脚位排列 接口
Y1	当选择外振时，需要焊接对应频率的晶振
J3	如果外接晶振为 8M，则可以直接从仿真器引入，引入时把 J3 短接一起即可。
J1	与 SN-Link 仿真器接口

5.2.10 SM111X

PCB 标注丝印	功能说明
J2	当目标板需要仿真器供电时，需要短接一起
14PIN 8PIN 6PIN 6PIN	对应规格书各种脚位排列 接口
Y1	当选择外振时，需要焊接对应频率的晶振
J3	如果外接晶振为 8M，则可以直接从仿真器引入，引入时把 J3 短接一起即可。
J1	与 SN-Link 仿真器接口

5.2.11 SM112X

PCB 标注丝印	功能说明
J2	当目标板需要仿真器供电时，需要短接一起
14PIN 8PIN 6PIN 6PIN	对应规格书各种脚位排列接口
Y1	当选择外振时，需要焊接对应频率的晶振
J3	如果外接晶振为 8M，则可以直接从仿真器引入，引入时把 J3 短接一起即可。
J1	与 SN-Link 仿真器接口

5.2.12 SM113X

PCB 标注丝印	功能说明
J2	当目标板需要仿真器供电时，需要短接一起
P1 P2 P3	对应规格书各种脚位排列接口
Y1	当选择外振时，需要焊接对应频率的晶振
J3	如果外接晶振为 8M，则可以直接从仿真器引入，引入时把 J3 短接一起即可。
J1	与 SN-Link 仿真器接口

5.2.13 SM511X

PCB 标注丝印	功能说明
J2	当目标板需要仿真器供电时，需要短接一起
16PIN	对应规格书各种脚位排列 接口
Y1	当选择外振时，需要焊接对应频率的晶振
J3	选择外接晶振时，J3 短接，可以直接从仿真器引入时钟代替外部晶振
J1	与 SN-Link 仿真器接口

5.2.14 SM512X

PCB 标注丝印	功能说明
J2	当目标板需要仿真器供电时，需要短接一起
16PIN 14PIN	对应规格书各种脚位排列 接口
J1	与 SN-Link 仿真器接口

6 WinScope IDE 快速开发实例

本章通过一个实际例子让用户快速的掌握如何使用 WinScope IDE 进行项目的开发。假设：为了简单测试仿真器的好坏，现在我需要对 P03 编写一段代码。代码中包含几个简单的功能：

- 对 RAM 进行读写测试，我随意选几个地址
- 对 P0 端口的 P07 设置为外部中断，P06 为输出
- 开启定时器 0 的定时中断
- 测试函数的调用与返回

功能定义好后，现在可以开始了。

6.1 建立项目

双击  ，打开仿真软件，如图 24 所示。

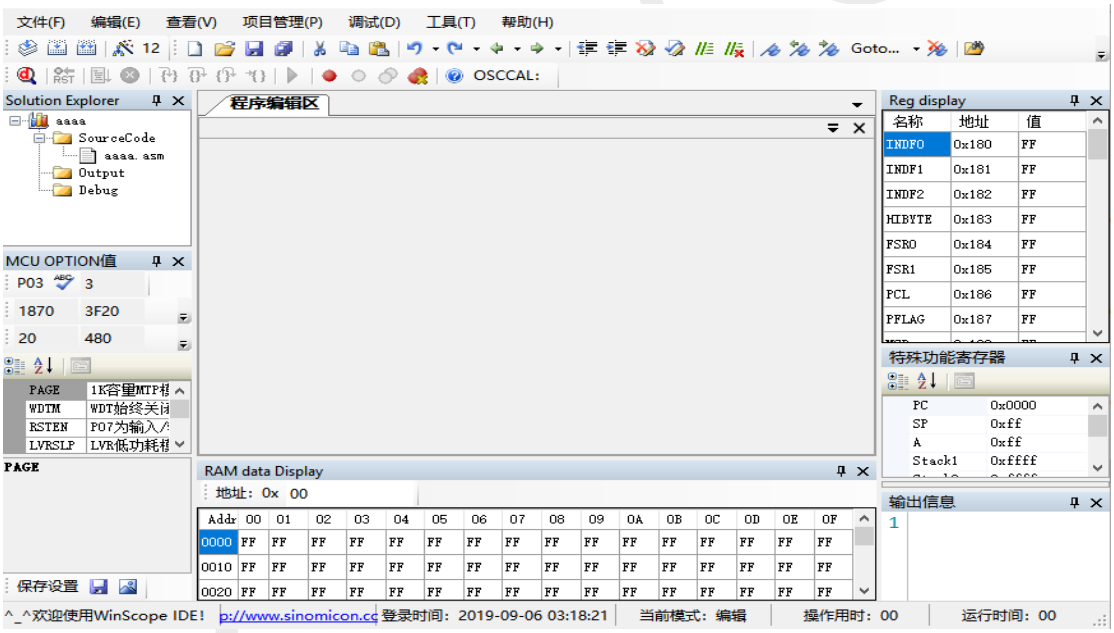


图 24 WinScope IDE 界面

在菜单栏中选择“项目管理”→“新建项目”这时会出现如图 25 所示对话框，在对话框中选择芯片型号“P03”，在项目名称一栏输入“P03_Test”，在项目存放路径中选择你需要存放项目的地方。 建议选择的路径和项目名称 中间不要含有空格、星号等不附合 C 命名标准的字符。当你选择完芯片型号之后编译器会自动的选择，不需要手工干预。

图 25 新建项目向导

设置完路径，点击下一步，会出现如图 26 所示对应型号的 OPTION 设置窗口。我们按照如图 26 所示设置，OPTION 值为 3。设置完后点保存，然后一直下一步至完成。

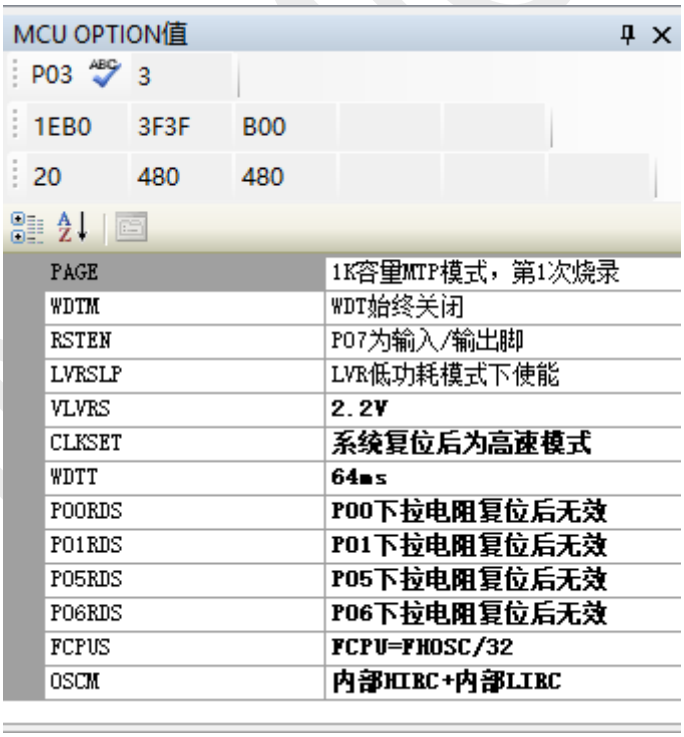


图 26

点击“完成”按键之后，可以看到 WinScope IDE 窗口的工程管理窗中已经按照“P03_Test”名称建立了一个新的工程。但这时工程还是空的，我们需要添加一个 ASM 文件。添加的方法：右键选中“SourceCode”文件夹→“添加文件”如图 27 左所示。根据提示将路径指向刚才建立项目的文件夹下，输入文件名“P03_Test.asm”然后点击保存后，可以看到图 27 右边的图。

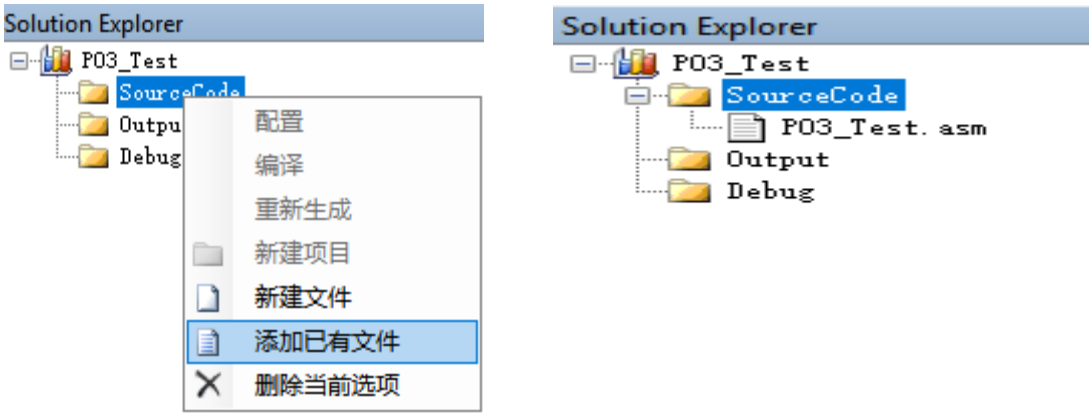


图 27 在项目中添加文件

6.2 代码编辑

项目建立完成后，可以开始编辑代码。在编辑的过程中默认规定以”fn_”开头的标号会与“RTS”或者“RTI”进行配对，可以进行代码折叠。如下图 28 所示 fn_int0 代码段已经折叠起来，双击标号将会展开。在编辑中可以使用书签，查找等功能进行快速定位。

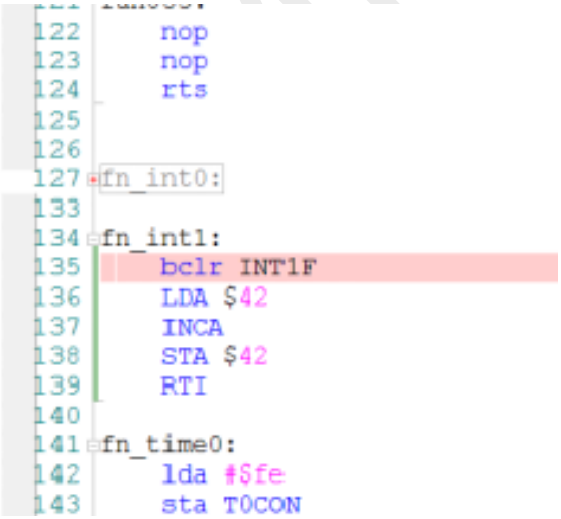


图 28 代码折叠

6.3 编译、生成代码

代码编写完之后，可以对文件进行编译除错。需要注意的是 WinScope 中有三个非常相似的功能菜单，用户需要对他们加以区分。如图 29 所示。编译/汇编只编译当前打开的文件，而生成代码则是编译整个工程项里的文件并把中间文件生成 S19 目标文件。所以我们需要选择的是“生成代码”的菜单。而“重新生成并下载” 菜单是连同进入仿真模式的功能。

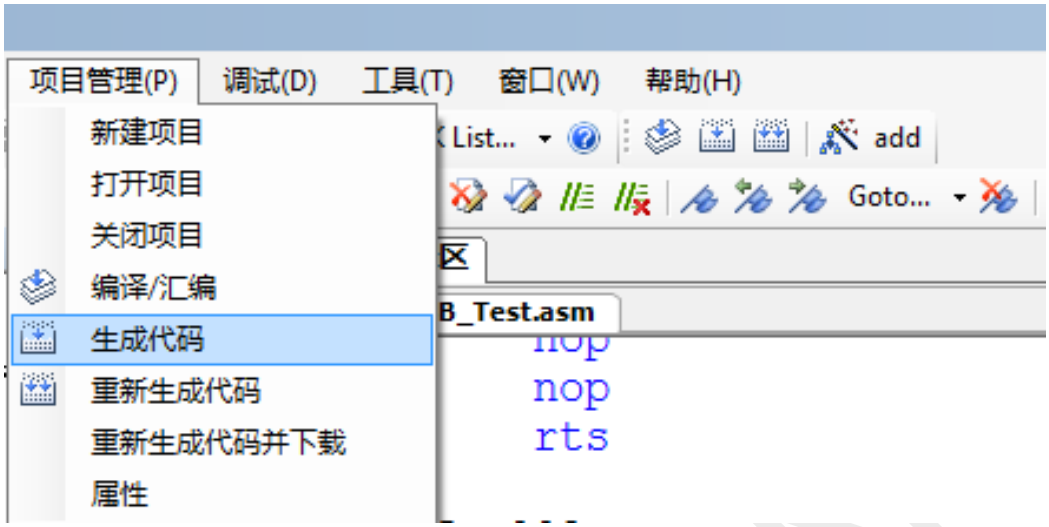


图 29 编译和生成代码菜单区分

当编译出错时，WinScope IDE 会提示编译出错，并在构建输出框输出相关出错信息。如下图 30 所示，双击输出信息框中的出错行号，将会自动对应到文本编辑窗中的对位置。如图 30，我们看到 241 行 指令 JBST 出错 ，应该为 JBSET。

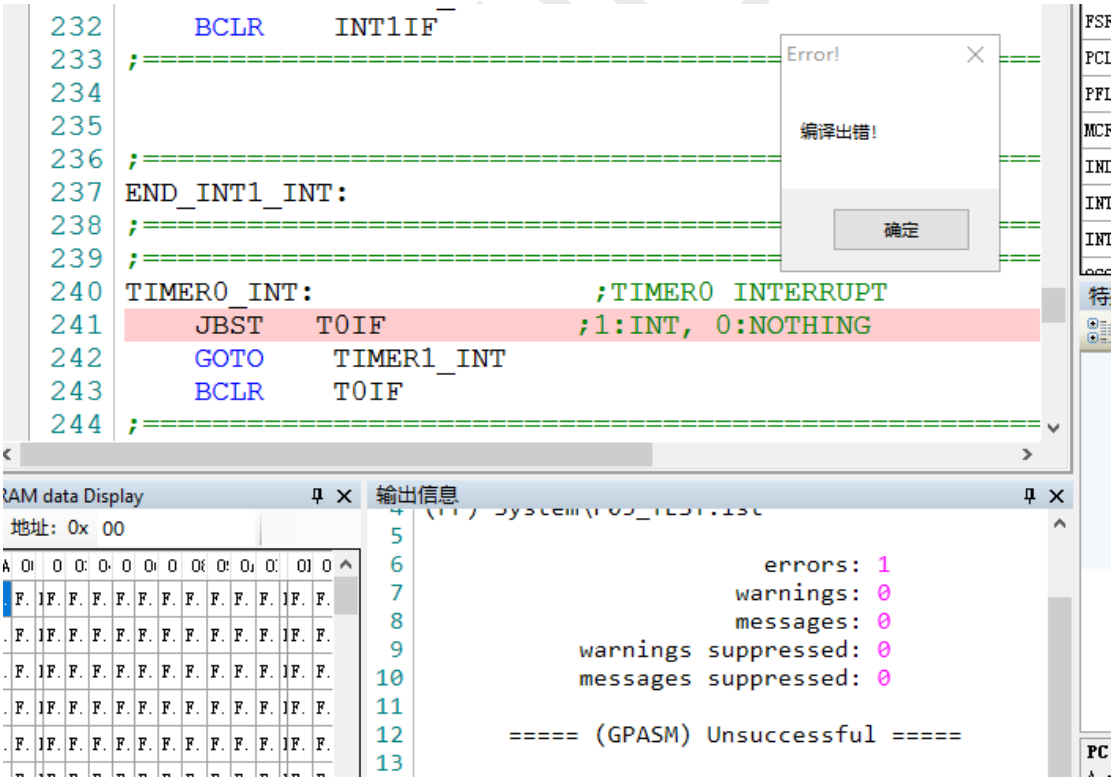
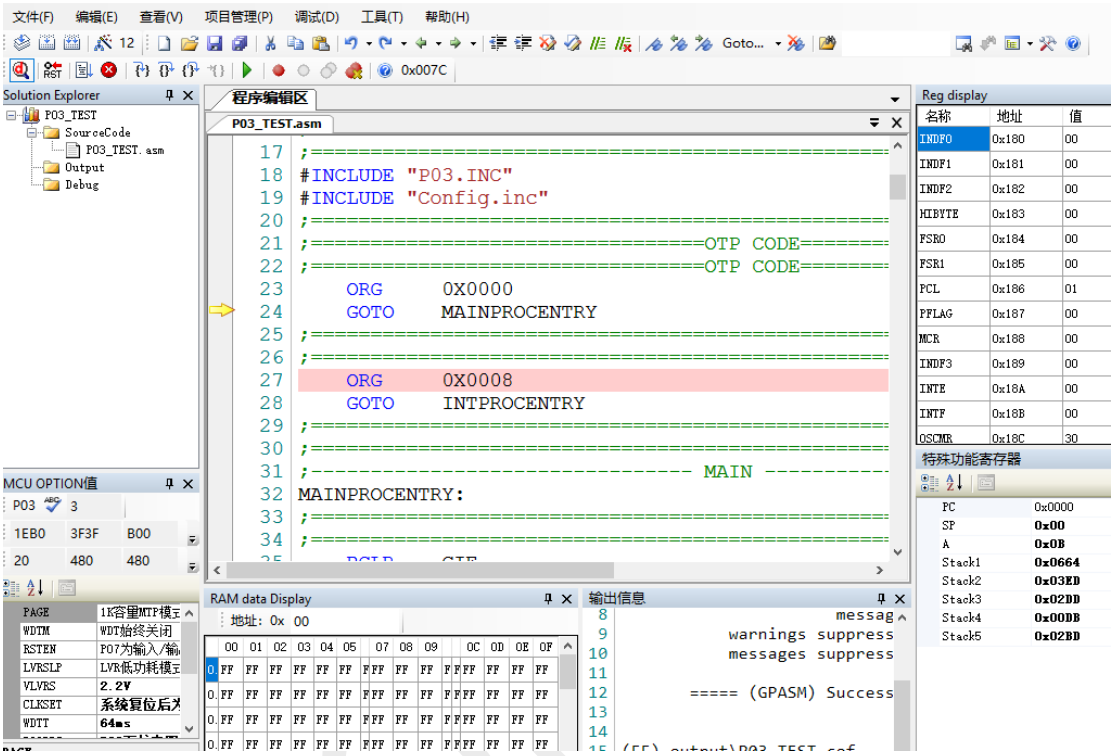


图 30 编译出错提示

6.4 进入调试模式、

当编译生成代码通过后，即可以连接仿真器进行代码仿真。点击工具栏上的图标，进入仿真模式，如果仿真器连接失败将会提示找不到端口等出错提示。当正确进入模式后看到如下图所示，PC 光标会复位到程序入口地址，RAM,REG 等窗口会被刷新。



这时候，离成功能已经很接近了。我们选择在定时器中断程序 time0 中设置一个断点，然后点击 全速运行程序，PC 指针将停留在定时器中断的断点处，如下图 31 所示。这时候可以查看各寄存器，RAM 变量的参数

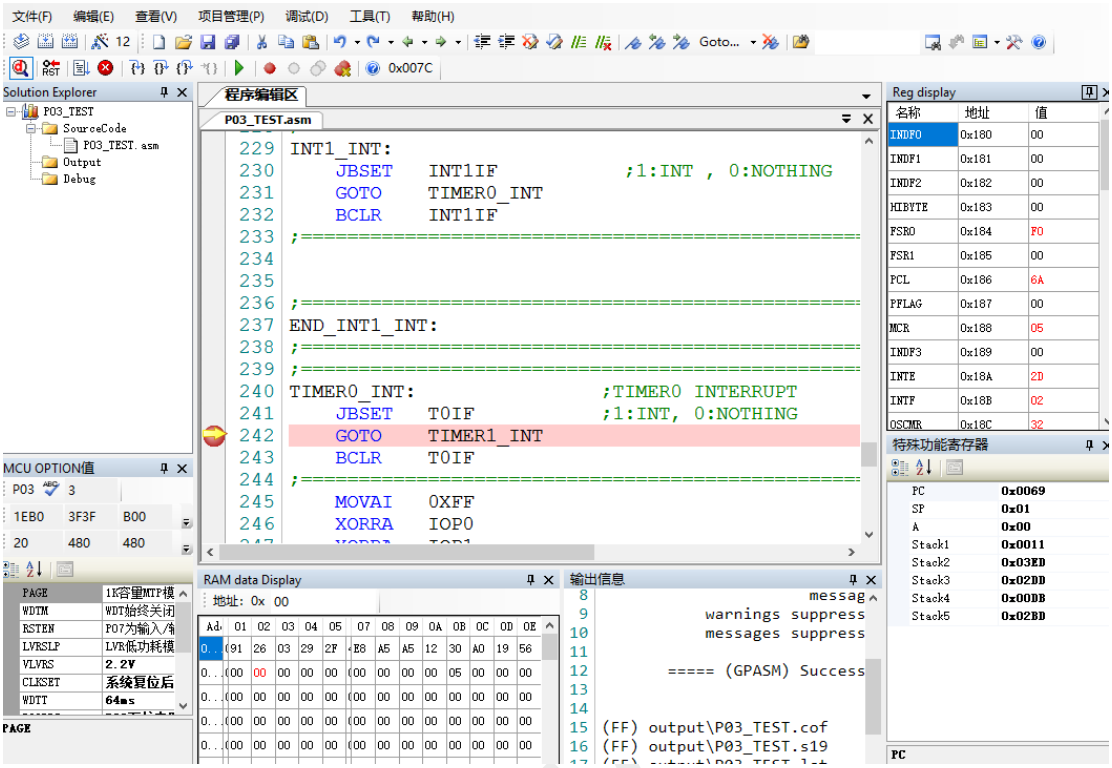


图 31 PC 停留在断点处

6.5 项目后续工作

仿真功能都正确之后，就可以到项目存放的目录下把 S19 文件烧写到实际的芯片中进行实物测试了。RISC 内核单片机的 S19 文件是 OUTPUT 子目录中。

7 RISC 汇编器介绍

7.1 汇编器语法

汇编程序由语句和空白组成。空白可以是一个空格或多个空格或制表符。使用空白的目的是使代码便于他人阅读。除非在字符常量内部，否则任何空白的意义与一个空格相同。每个语句后均跟有新行，它的一般格式如下。

[label:] [mnemonic [operands]] [;comment]

或

[label:] [directive [arguments]] [;comment]

Label 标号

Mnemonic 助记符

Directive 伪指令

Oprand 操作数

Argument 参数

Comment 注释

7.1.1 标号（Lable)

标号是从所有字母、数字的集合以及两个特殊字符（下划线（_）和句点（.））中挑选出来的一个或多个字符。 除非是局部符号这一特殊情况，否则标号不能以十进制数开始。标号是区分大小写的；它没有长度限制，并且所有字符都有意义。标号定义之后必须紧跟一个冒号。冒号后面可跟有空格、制表符、换行符、汇编器助记符或伪指令。标号本身可以单独放在一行，其代表的地址就是下一行代码的地址。链接完成后，标号的值为存储器中单元的绝对地址。

7.1.2 助记符(Mnemonic)

助记符告诉汇编器对哪些机器指令进行汇编。 例如，加（ADD)、跳转（goto）或移动（MOVRA)。与您自己创建的标号不同，助记符由 RISC 内核编译器提供。助记符不区分大小写。

7.1.3 伪指令(Directive)

汇编器伪指令是源代码中出现的命令，但不会直接翻译为机器代码。伪指令用于控制汇编器的输入、输出和数据分配。伪指令的第一个字符是句点（.)。欲知更多有关可用伪指令的信息，请参见后面相关“汇编器伪指令”章节。

7.1.4 操作数

操作数和助记符之间必须用一个或多个空格或制表符隔开。操作数由立即数、寄存器，目标选择和累加器选择组成。十六进制数的标志是以0x开头可H结尾。 八进制数的标志是用O结尾。二进制数的标志是用B结尾。十进制数对开头或结尾的字符没有特殊要求。

示例：0xe、16O、1110B和14都代表立即数值14。

7.1.5 参数和注释

每个伪指令可使用 0-3 个参数，这些参数为伪指令提供其他有关如何执行命令的信息。参数之间必须用一个或多个空格或制表符隔开。必须用逗号隔开多个参数。
注释：在汇编器中，使用分号“;”进行注释单行。如：Movai 0x16 ;mov 0x16 to A reg

7.2 常量表示

7.2.1 数字常量

数字常量分为整数、浮点数、定点数。整数是在 C 语言中适合 long 的数字。浮点数字是 IEEE 754 浮点数字。定点数字是 Q-15 定点格式的。

7.2.2 字符常量

有两种类型的字符常量。字符常量：用一个字节表示一个字符，并且其值可以在数字表达式中使用。字符串可能包含多个字节，并且它们的值不能在算术表达式中使用。

单个字符可以被写成一个单引号后面紧跟着该字符的形式，或是使用一对单引号引用该字符的形式。 汇编器接受以下转义字符来代表特殊控制字符：

转义字符	说明	十六进制值
\a	报警字符	07
\b	后退字符	08
\f	换页字符	0C
\n	换行字符	0A
\r	回车字符	0D
\t	水平制表字符	09
\v	垂直制表字符	0B
\\	反斜杠	5C

\?	问号字符	3F
\"	双引号	22

表 9-2-1 转义字符

数字表达式中字符常量的值是该字符的机器字节宽度的代码。汇编器假设字符代码是 ASCII 码。

字符串被写在一对双引号之间。可能包括双引号或空字符。在字符串中添加特殊字符的方法是在这些特殊字符前用反斜杠\进行转义。应用于字符串的转义序列同样也适用于字符。

7.3 表达式语法和运算

表达式指定地址和数字值。在表达式前面和/或后面可以有一段空白。表达式的结果必须是一个绝对数字，或是偏移到特定段的偏移量。如果表达式的值不是绝对的，并且没有足够的信息使汇编器在查看表达式时能确定其所在的段，在这种情况下，汇编器将终止其操作并返回一条错误消息。

7.3.1 整数表达式

整数表达式是由运算符分隔的一个或多个参数。参数是符号、数字或子表达式。子表达式是一个左括号“（”后面跟着一个整数表达式和一个右括号“）”；或者是一个后面有参数的前缀运算符。在整数表达式中，如果包含程序空间地址符号，将按照程序计数器（PC）的单位进行相关计算。程序计数器每执行一个指令字就会加1。例如，要跳转到L标号所在指令的下一条指令，就应指定目标为L+1（这里的1代表1条指令）。

示例：goto L+1

7.3.2 运算符

运算符是诸如+或%之类的算术函数。前缀运算符后面跟有一个参数。中缀运算符在两个参数之间。运算符的前面和/或后面可以是空白。前缀运算符具有比中缀运算符高的优先级。中缀运算符的优先级顺序由其类型决定。

前缀运算符：

汇编器有两个前缀运算符。每个运算符使用一个参数，参数必须是绝对的。

运算符	说明	示例
-	取负值。取二进制补码	-1 编译后为 0xff

~	位非。1的补码	~1编译后为 0xfe

表7-3-1 前缀运算符

中缀运算符:

中缀运算符的两边各有一个参数。运算符根据其类型具有不同的优先级，如下表所示，但是优先级相同的运算按照从左到右的顺序执行。除了+或-之外，运算符都必须是绝对的，而且结果也必须是绝对的。

运算符	说明	示例
+	加	2+6 (=8)
-	减	6-2 (=4)
*	乘	5*4 (=20)
/	斜杠与C运算符“/”相同	23/4 (=5)
%	求余	30%4 (=2)
<<	左移 与C左移运算符相同	0x13<<2 (=0x4c)
>>	右移 与C右移运算符相同	0x4c>>2 (=0x13)
位操作		
&	位与	4&6 (=4)
^	位异或	4^6 (=2)
|	位或	2|4 (=6)

表7-3-2 中缀运算符

7.4 伪指令

为了增加源程序的可读性和可维护性，我们引入了伪指令的概念。伪指令本身不会产生可执行的汇编指令，但它们可以帮组“管理”你编写的程序，其实用性和必要性绝不亚正真的汇编指令。我们在此着重介绍最常用的几种伪指令。

○ **#include 或 include**

#include 伪指令的作用是把另外一个文件的内容全部包含复制到本伪指令所在的位置。被包含复制的文件可以是任何形式的文本文件，当然文件中的内容和语法结构必须是汇编器能够识别的。最经常被“include”的是针对RISC单片机内部特殊功能寄存器定义的包含头文件，它们全部放在WinScope IDE的安装目录INC 和BIN文件夹下，每一个RISC型号单片机有一个对应的预定义包含头文件，扩展名是“.inc”。除了一些符号预定义文件可以把现有的其

它程序文件作为一个代码模块直接“包含”进来作为自己程序的一部分。如下例：

#include <P03.inc> ;把预定义的P03 寄存器符号包含到此处#include ” fun001.asm” ；把现有的程序文件包含进来作为自己代码的一部分

请注意被包含文件的引用方式。一种是<>尖括号引用，这种引用意味着让编译器去默认的路径下寻找该文件，WinScope默认的寄存器预定义文件存放路径即为WinScope安装后的目录；另一种是” ” 双引号引用，这种引用方式的意思是指示编译器从引号中指定的全程文件路径下寻找该文件。上例”fun001.asm”没有指定路径，即意味着在当前项目路径下寻找fun001.asm 文件。如果编译器找不到被包含的文件，将会有错误信息告知。

○ **List**

list 伪指令可以设定程序编译时的一些信息，例如所选单片机的型号，编译时选择的缺省数制等。例如：

list p=P03, r=DEC ;单片机型号为P03，无特别指明的数字为十进制数

○ **#define / #undefine**

#define 的作用是定义常数符号，即用一个符号变量替换另一个符号串或变量。被替换的可以是任意字母数字组成的符号但替换者本身不能是一个纯数字。例如：

#define DELAY_TIME 1000 ;定义常数符号，即用DELAY_TIME 符号代替1000

#define KEY1 PORTB,7 ;用KEY1 符号代替端口PORTB 的第7 引脚

用#define 伪指令定义符号后，可使程序中的变量或指令变得更具实际意义，也使程序变得更易维护。指令“bset PORTB,7”和“bset KEY1”在事先用了上例中的#define 后编译的结果是一样的，但明显地后者看起来更容易理解，一看就知道这是在测试编号为KEY1的一个按键。而且如果你的硬件设计改动了KEY1 所接的单片机引脚，只要改动这一处#define 重新定义引脚位置，程序的其它部分无需任何修改，再编译一次即可得到更新后的软件代码。一个好的编程习惯是事先把一些代表实际意义的变量、单片机的输入输出引脚在硬件电路中的实际功能等用#define 伪指令定义成简单直观的符号名字，然后在程序中直接用其符号名字而不用简单机械的数字形式。替换的工作由编译器在编译时自动完成。它会先扫描你的源程序代码，把事先#define 的符号名改回成被替换的字符串，然后再继续编译生产机器码。

○ **equ**

equ 顾名思义是“等于”的意思，其作用和#define 伪指令有点类似，也是用一个符号名字替换其它数字变量，但它只能替换立即数。如果要替换一个符号名字，则此符号名必须事先用#define 或equ 伪指令已经定义替换了一个立即数。例如：

#define MyCount 0x70 ;定义MyCount 符号替换立即数0x70

```
w_temp equ 0x20 ;符号名w_temp 等于0x20
count1 equ MyCount ;符号名count1 等同于MyCount ;如果MyCount 没有事先定义则会产生一个错误
```

在绝对定位的编程模式中 equ 被经常用于定义用户自己的变量，即用一个符号名代替一个固定的存储单元地址，上例中的w_temp 定义即属于此类。用equ 方式定义的符号在汇编后可以生成相关的调试信息，可以通过各种变量观察的方式显示此符号所代表的内存地址处的数据内容，但用#define 方式定义的符号则不能产生调试信息。要注意equ 伪指令本身并没有限定所定义的一定是一个变量地址，它只是一个简单的符号和数字替换而已，其意义必须和具体的指令结合才能确定。

○ **cblock / endc**

用equ 伪指令可以给一个符号变量分配一个地址。但在一个程序设计过程中往往需要定义很多变量，你当然可以给每一个变量逐个用equ 的方法分配一个地址空间。但如果变量很多，这样做就显得非常麻烦，你必须自己安排每个变量的地址，小心不能出现地址重叠；若要在已定义分配好的变量间插入新的变量，那就必须重新逐个安排随后变量的地址等等。
cblock/endc 伪指令可以轻松解决有很多变量定义的场合出现的这些问题，我们把它叫作变量块连续定义。具体用法如下：
cblock 伪指令声明变量块的起始地址，endc 伪指令声明变量块定义结束，cblock/endc中间可以插入任意多的变量声明。其地址编排由编译器自动计算：第一个变量地址分配从起始地址开始，然后按所声明变量保留的字节数自动分配后面变量的地址，变量所需保留的字节数用“:”加后面的数字表示，如果只有一个字节“:1”可以省略不写。以下例来说明：

```
cblock 0x20 ;变量定义起始地址为0x20
w_temp ;w_temp 地址为0x20，占一个字节
status_temp ;status_temp 地址为0x21，占一个字节
buffer:8 ;buffer 的起始地址为0x22，并保留8 个字节单元
var1 ;var1 的地址为0x2a，占一个字节
var2 ;var2 的地址为0x2b，占一个字节
endc ;结束变量连续定义
```

用 cblock 方式定义的变量和用equ 方式定义的变量一样在汇编后可以生成相关的调试信息，可以通过各种变量观察的方式显示此符号所代表的内存地址和其中的数据内容，所以实际编程时一般无需关心计算每个变量的具体地址。程序员要注意的用这种方式连续定义很多变量时不要让变量块跨越所处bank 的边界。你可以在cblock 中随意插入新定义的变量，或通过改变起始地址的方式使变量块挪到其它内存地址处，地址的更新由编译器代劳。

- **org**

org 用以定义程序代码的起始地址，通过此伪指令你可以把程序定位到任何可用的程序空间，它实现的是程序代码绝对定位，如例：

```
org 0x0000 ;定义复位入口地址，以下指令从地址0x0000 开始
  goto main ;
```

- **dt**

dt 的作用是定义表格数据。RISC汇编指令实现表格定义的最基本指令是“RETAI xx”，表格中的每一个字节数据都以指令“RETAI”的形式出现。若表格较大，就需要很多“RETAI”指令，比较麻烦，可读性也差。这时我们可以用此“dt”伪指令替代“RETAI”实现很多数据的表格定义。如例：

```
dt 0 ; RETAI 0
dt 1, 2, ’ 3’  ; RETAI 1
                    ; RETAI 2
                  ; RETAI 0x33 (’ 3’ 的ASCII 码)
dt ” ABC”  ; RETAI ’ A’
              ; RETAI ’ B’
              ; RETAI ’ C’
```

- **fill**

fill 伪指令可以实现对程序空间连续自动填充某一特定的指令数据，被填充的可以是一个立即数（实际肯定代表某一条指令），也可以是一条形象的汇编指令。基本上在一个设计中都有一些程序空间没有写上具体的指令编码（空白处），在单片机正常运行时这些地方的指令是不会被执行到的。但在有干扰的情况下程序跑飞正好落在这些非法指令处时，就有必要设置软件陷阱捕捉这些非法跳转，让程序恢复正常运行。如果要程序员一个一个地址去分析哪里有空的指令单元然后又用特殊指令一条一条填入，这是根本行不通的。fill 伪指令在这时就派上用场了。

```
fill 0x0000, 5 ;从当前地址处连续5 个程序字填成0x0000(NOP 指令)
fill (goto $), NEXT_BLOCK-$ ;从当前地址开始到标号NEXT_BLOCK 前所有程序空间填上
goto $ （死循环）指令
org 0x0100
NEXT_BLOCK
```

- **end**

end 伪指令告诉汇编编译器编译工作到此为止，end 后面所有的信息，不管正确与否，一概不管。绝大多数情形下你的程序的最后一行应该是“end”。无论如何，end 必须出现在程序中，不然编译器会报错，无法进行编译工作。

○ **high 和 low**

一个16 位的数在8 位单片机中必须被拆解成高8 位一个字节（高字节）和低8 位一个字节（低字节）才能用指令一条条处理，类似的处理在对两字节变量赋立即数初值和基于PC 相对跳转查表前设定PCL寄存器时经常碰到。WinScope提供了high 和low 两个运算符分别计算一个立即数的高字节和低字节。例如：

```
#define abc 0x20a5
movai low(sks);把 0xa5 赋给了 A
ai high(sks);把 0X20 赋给了 A
```

8 RISC C 语言开发与调试

8.1 项目建立

本章介绍如何在 WinSope IDE 下建立 **RISC** C 语言项目。具本使用方法如下：

1. 建立项目。点击菜单中“项目管理”然后选择“新建项目”，这时会出现如下对话框：

图 36 新建项目

2. 选择相关的 RISC 核芯片型号，项目名称填写需要按照 C 命名规则进行。项目存放路径选择，路径中不允许出现中文名路径。开发语言，请选择 C 语言。例如：我们选择 P03 型号，项目名称为：P03_c_test。然后点击下一步。这时会出现如下图所示对话框:

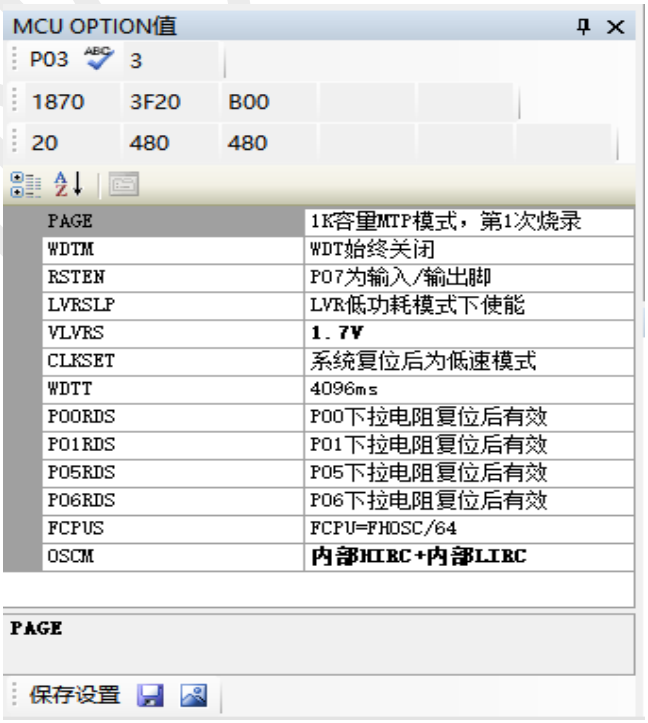


图 37OPTION 设置

3. 根据项目的需要，对芯片的 OPTION 进行设置。设置完成后，点击下面的保存设置。然后回到原来的对话框，点击“完成”。

8.2 添加文件

项目建立完成之后，开始添加 C 文件。开发环境对 C 文件有如下要求：

1. 必须包含 main 函数
2. 所有 C 文件只能放在项目文件夹根目录下

8.3 乘除法使用说明

RISC C 乘除法运算规则如下：

1. 只涉及到 unsigned char、unsigned int、unsigned long 三种类型的参数和结果
2. char*char=char 、 int*int=int 、 long*long=long 、 char/char=char 、 int/int=int 、 long/long=long，以上六种计算形式请直接使用，如：a=0x2，b=0x0E，temp=a*b；
3. char*char=int (宏名:_MULINT_CC、调用函数:mulint_cc(unsigned char, unsigned char))
   int*char=int(宏名:_MULINT_IC、调用函数:mulint_ic(int,unsigned char))
   int*char=long(宏名:_MULLONG_IC、调用函数:mullong_ic(int,unsigned char))
   int*int=long(宏名:_MULLONG_II、调用函数:mullong_ii(int,int))
   long*char=long(宏名:_MULLONG_LC、调用函数:mullong_lc(long,unsigned char))
   long*int=long(宏名:_MULLONG_LI、调用函数:mullong_li(long,int))
   int/char=char(宏名:_DIVUCHAR_IC、调用函数:divuchar_ic(unsigned int, unsigned char))
   int/int=char(宏名:_DIVUCHAR_II、调用函数:divuchar_ii(unsigned int, unsigned int))
   int/char=int(宏名:_DIVUINT_IC、调用函数:divuint_ic(unsigned int, unsigned char))
   long/char=int(宏名:_DIVUINT_LC、调用函数:divuint_lc(unsigned long, unsigned char))
   long/int=char(宏名:_DIVUCHAR_LI、调用函数:divuchar_li(unsigned long, unsigned int))
   long/int=int(宏名:_DIVUINT_LI、调用函数:divuint_li(unsigned long, unsigned int))
   long/long=char(宏名:_DIVUCHAR_LL、调用函数: divuchar_ll(unsigned long, unsigned long))
   long/char=long(宏名:_DIVULONG_LC、调用函数: divulong_lc(unsigned long, unsigned char))
   long/int=long(宏名:_DIVULONG_LI、调用函数:divulong_li(unsigned long, unsigned int))
   long/long=int(宏名:_DIVUINT_LL、调用函数:divuint_ll(unsigned long, unsigned long))
   以上 16 种计算形式统一定义于\tools\share\include\sm-cacul.h 中，所有的方法采用 C 条件调用方式书写，只有定义了相应的宏才会对相应的计算方法进行编译解析，因此不会造

成不必要的 ram、rom 资源消耗，具体使用方法如下：

如 char*char=int，需调用 mulint_cc 函数实现，参考代码：

```
#define _MULINT_CC      // 定义所需计算方式宏
#include <sm-cacul.h>   // 包含计算方法所在的头文件
                        // 宏和头文件的定义顺序必须是先宏后头文件
...
temp = mulint_cc(left,right);
...
```

8.4 RISC C 使用注意事项

RISC C 使用需注意如下要求：

1. 调用 ASM 方法

```
__asm
  汇编代码
__endasm;
```

另外需要注意的是，调用汇编代码使用到寄存器时，需加“_”前缀，如“_T0CR”

2. 位定义方法请参考.h 头文件里面的定义方法，如下:

```
typedef struct {
        unsigned char bit0      : 1;
        unsigned char bit1      : 1;
        unsigned char bit2      : 1;
        unsigned char bit3      : 1;
        unsigned char bit4      : 1;
        unsigned char bit5      : 1;
        unsigned char bit6      : 1;
        unsigned char bit7      : 1;
} BITS_T;
```

3. 指定地址变量定义方法，如下：

指定 ram 地址：

```
__sfr __at 0x17 a0;
typedef union {
  struct {
        unsigned char a00:1;
        unsigned char a01:1;
        unsigned char a02:1;
        unsigned char a03:1;
```

```
            unsigned char a04:1;
            unsigned char a05:1;
            unsigned char a06:1;
            unsigned char a07:1;
        };
    } __a0bits_t;
    volatile __a0bits_t __at 0x17 a0bits;
    #define a00         a0bits.a00      /* bit 0 */
    #define a01         a0bits.a01      /* bit 1 */
    #define a02         a0bits.a02      /* bit 2 */
    #define a03         a0bits.a03      /* bit 3 */
    #define a04         a0bits.a04      /* bit 4 */
    #define a05         a0bits.a05      /* bit 5 */
    #define a06         a0bits.a06      /* bit 6 */
    #define a07         a0bits.a07      /* bit 7 */
```

也可以用 unsigned char __at 0x17 a0;方法进行定义

指定 rom 地址：

```
    const unsigned char __at 0x17 a0;
```

4. ".h"、".c" 文件的文件名必须由字母或数字或下划线组成
5. 编译器默认占用前七个寄存器地址，请勿重复定义
6. 临时变量已修改能正常使用，但是目前还不能查看变量的值，如还有问题，请联系软件开发人员
7. 代码输入，同一行建议不要写多行代码
8. switch 语句已修改能正常使用，如还有问题，请联系软件开发人员
9. 中断函数定义

```
void int_isr(void) __interrupt
{
__asm
    movra       _ABuf
    swapar  _STATUS
    movra       _StatusBuf
__endasm;
.....
code
.....
__asm
    swapar  _StatusBuf
    movra       _STATUS
```

```
        swapr   _ABuf
        swapar  _ABuf
    __endasm;
}
```

请将 ABuf、StatusBuf 定义成全局变量；另外，在中断函数中尽量使用不要过多层的调用函数，以防止堆栈溢出，可以定义标志位来处理

10. Ram 使用情况查看已在输出信息框中列出，格式为：

```
RAM USAGE MAP ('X' = Used,  '-' = Unused)
0000 : XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX----- ------------
----
```

8.5 编译说明

8.5.1 数据类型长度

Type	Size(byte)	Scope
char	1	-128 ~ 127
unsigned char	1	0 ~ 255
short	2	-32768 ~ 32767
unsigned short	2	0 ~ 65535
int	2	-32768 ~ 32767
unsigned int	2	0 ~ 65535
long	4	-2147483648 ~ 2147483647
unsigned long	4	0 ~ 4294967295
long long	4	-2147483648 ~ 2147483647
unsigned long long	4	0 ~ 4294967295
float	4	/
double	4	/
long double	4	/

8.5.2 Bit 变量的定义和使用

1. Bit 变量可被声明为 global、static、local、extern 以及函数返回值，函数参数等形式
2. Bit 类型赋值时能够接收的常量值只 0 和 1
3. Bit 类型除了支持同类型变量之间的赋值外，还可以与 1 byte 类型相互赋值，当 1byte 类型赋值给 bit 类型时，取变量最后一个 bit 赋与 bit 变量。Bit 变量不支持与大于 1byte 的类型

相互赋值

4. Bit 类型可以用于 if，while，do while，for 语句中，与同类型的变量或 0/1 进行相等与不等的比较
5. Bit 类型用于 switch 语句时，switch 的条件判断表达式可以是 b 或 !b，case 只能使用 0，1 两种常量值
6. Bit 类型仅支持逻辑（&，|，^）运算，不支持其他任何二元（如+，-，×，/等运算）
7. Bit 类型不可声明或定义为指针，数组等类型
8. Bit 类型不能在 Struct/Union 中使用

8.5.3 Sbit 数据类型的声明

sbit bit_name = var_name:bit_number;
sbit bit_name = address: bit_number;
bit_name: 所声明 sbit 变量的名称。
var_name: 指向变量的名称。
address: 指向某个地址，必须是十六进制或十进制常数值。
bit_number: 指定对应的 Bit 位置，十六进制或十进制常数值。范围：0 ~ 7。
1. Sbit 类型继承了 bit 类型的属性和限制。
2. Sbit 类型定义的变量必须先对其绑定地址后使用。
3. 不支持 extern sbit 变量。

8.5.4 中断函数的声明

1. 定义中断函数的方式需要使用 void __interrupt [0x8] FunName(void)，如:

```
void __interrupt[0x08] interrupt_isr(void)
{
__asm
movra    _ABuf
swapar   _STATUS  //或者_PFLAG，不同芯片状态寄存器名称不同
movra    _StatusBuf
__endasm;
//.....
//code
//.....
__asm
swapar   _StatusBuf
movra    _STATUS  //或者_PFLAG，不同芯片状态寄存器名称不同
swapr    _ABuf
swapar   _ABuf
__endasm;
}
```

进入中断函数时系统会备份所有寄存器，中断函数结束前，前述备份的寄存器均会被还原。使用者若其他额外的备份需求，需要自行撰写代码进行备份。
中断函数不能被其它函数调用
2. 定义中断子函数
定义和声明中断子函数时必须使用__interrupt 关键字，如：

```
int __interrupt callbyisr(int x)
{
return x;
}
```

中断子函数只可被中断函数、中断子函数直接或间接调用

8.5.5 内嵌汇编（暂不支持）

8.5.6 函数使用说明

1. strcpy - 拷贝字符串
格式：

```
#include <string.h>
char *strcpy(char *dest, const char *src);
```

说明：
函数 strcpy() 将 src 指向的字符串(包含字符串结束符“\0”)复制到 dest 所指向的数组中。字符串不能重复，目标字符串 dest 必须足够大来承接拷贝。在 src 长度小于 n 长度的情况下，dest 的余数被 null 填充。
返回值：
函数 strcpy()返回指向 dest 字符串的指针。

2. strncpy - 复制字符串
格式：

```
#include <string.h>
char *strncpy(char *dest, const char *src, size_t n);
```

说明：
函数 strncpy()与 strcpy()相似，除非 src 中多于 n bytes 被复制。因此，若 src 的前 n bytes 中没有 null byte，不以 null 结尾。在 src 长度小于 n 长度的情况下，dest 的余数以 null 填充。
返回值：
函数 strncpy() 返回指向字符串 dest 的指针。

3. strcat - 连接两个字符串
格式：

#include <string.h>
char *strcat(char *dest, const char *src);
说明:
函数 strcat() 将 src 字符串附加到 dest 字符串并在 dest 结尾覆盖“\0”，然后在结尾处添加“\0”。字符串不能重复，dest 字符串必须足够大存放结果。
返回值:
函数 strcat()返回指向 dest 字符串的指针。

4. strncat - 连接两个字符串
格式:
#include <string.h>
char *strncat(char *dest, const char *src, size_t n);
说明:
函数 strncat()与 strcat()相似，除 了只将 src 首部的 n 个字符被附加到 dest。
将 src 字符串附加到 dest 字符串并在 dest 结尾覆盖“\0”，然后在结尾处添加“\0”。字符串不能重复，dest 字符串必须足够大存放结果。
返回值:
函数 strncat()返回指向 dest 最终字符串的指针。

5. strcmp - 比较两个字符串
格式:
#include <string.h>
int strcmp(const char *s1, const char *s2);
说明:
函数 strcmp() 比较两个字符串 s1 和 s2。当 s1<s2 时，返回值<0；当 s1=s2 时，返回值=0；当 s1>s2 时，返回值>0。
返回值:
若 s1(或者前面 n byte)被查询到，函数 strcmp()返回值:
当 s1<s2 时，返回值<0
当 s1=s2 时，返回值=0
当 s1>s2 时，返回值>0

6. strncmp - 比较两个字符串
格式:
#include <string.h>
int strncmp(const char *s1, const char *s2, size_t n);
说明:
与 strcmp()函数相似，只比较 s1 和 s2 开始的 n 个字符。

返回值：
若 s1(或者前面 n byte)被查询到，函数 strncmp()返回值：
当 s1<s2 时，返回值<0
当 s1=s2 时，返回值=0
当 s1>s2 时，返回值>0

7. strchr - 查找字符串中某个字符首次出现的位置
格式：
#include <string.h>
char *strchr(const char *s, char c);
说明
函数 strchr() 返回首次出现 c 的位置的指针，如果 s 中不存在 c 则返回 null。
这里“字符”单位是 byte，这些函数在较长字符或多 byte 字符下不能正常运行。
返回值：
函数 strchr() 返回首次出现所查找字符位置的指针，如果不存在该字符则返回 NULL。

8. strrchr - 定位字符串中某个字符
格式：
#include <string.h>
char *strrchr(const char *s, int c);
说明：
函数 strrchr()返回指向字符串 s 中的字符 c 最后出现位置的指针。
这里“字符”单位是 byte-这些函数在较长字符或多 byte 字符下不能正常运行。
返回值：
函数 strrchr()返回指向字符串中的目标字符最后出现位置的指针，未找到的情况下返回 NULL。

9. strspn - 计算某个字符串在字符集中出现的次数
格式：
#include <string.h>
int strspn(const char *s, const char *accept);
说明：
函数 strspn()计算 s 初始值长度，并在 accept 字符中查找 s 初始值出现的次数。
返回值：
函数 strspn()返回 s 初始化字符在 accept 中出现的次数。

10. strcspn - 本函数用来比较二字符串并计算出不同处的字符串长度
格式:
#include <string.h>

int strcspn(const char *s, const char *reject);
说明:
本函数用来比较 s 和 reject 二字符串，并计算出不同处的字符串长度。
返回值:
返回 s 和 reject 不同处字符串长度。

11. strpbrk - search a string for any of a set of characters
格式:
#include <string.h>
char *strpbrk(const char *s, const char *accept);
说明:
在字符串 s 中寻找字符串 accept 中任何一个字符相匹配的第一个字符的位置。
返回值:
函数 strpbrk() 返回指向 s 中和 accept 任何一个字符相匹配的第一个字符位置的指针，若找不到则返回 NULL。

12. strstr - 定位子字符串
格式:
#include <string.h>
char *strstr(const char *haystack, char *needle);
说明:
函数 strstr()在 haystack 中查找子字符串 needle 第一次出现的位置。字符串结束符“\0'”不参与比较。
返回值:
返回指向第一次出现子字符串首位的指针，如果没找到则返回 NULL。

13. strlen - 计算字符串长度
格式:
#include <string.h>
int strlen(const char *s);
说明:
函数 strlen()计算字符串 s 的长度，不包含字符串结束符“\0”。
返回值:
函数 strlen()返回字符串 s 中字符的个数。

14. strtok - extract tokens from strings
格式:
#include <string.h>

char *strtok(char *s, const char *delim);
说明:
一个 token 是在字符串 delim 中未出现的非空字符串，以“\0”或者 delim 中出现的某个字符结束。
函数 strtok()用于将字符串分解为不同的 token。第一次调用 strtok()时 s 为第一个参数。其余调用许建第一个参数设置为 NULL。每次调用返回指向下一个 token 的指针，在没有 token 时返回 NULL。
若 token 以分隔符结束，该分隔符被 \0 覆盖，且指向下个字符的指针被用作下一个 strtok()调用。分隔符字符串 delim 在每次调用 strtok()函数时有所不同。
返回值:
函数 strtok()返回指向下一个 token 的指针，若没有 token 则返回 NULL 。

15. memcpy - 复制内存区域
格式:
#include <string.h>
void *memcpy(void *dest, void *src, size_t n);
说明:
函数 memcpy()从内存区域 src 复制 n byte 到 dest 所指内存区域。内存空间不能重复。若内存区域重复，使用 memmove(3) 。
返回值:
函数 memcpy() 返回指向 dest 的指针。

16. memcmp - 比较内存区域
格式:
#include <string.h>
int memcmp(void *s1, void *s2, size_t n);
说明:
函数 memcmp()比较内存区域 s1 和 s2 的前 n 个 byte。当 s1<s2 时，返回值<0；当 s1=s2 时，返回值=0；当 s1>s2 时，返回值>0。
返回值:
函数 memcmp()返回值： 当 s1<s2 时，返回值<0；当 s1=s2 时，返回值=0；当 s1>s2 时，返回值>0。

17. memset - 在内存中填充常量个字节
格式:
#include <string.h>
void *memset(void *s, int c, size_t n);
说明:

函数 memset() 将 s 所指向的前 n 个 byte 内存中的每个字节的内容全部设置为 c 指定的 ASCII 值。
返回值:
函数 memset()返回指向内存空间 s 的指针。

18. memmove - 复制一块内存区域
格式:
#include <string.h>
void *memmove(void *dest, void *src, size_t n);
说明:
函数 memmove()复制 src 内存区域中的 n 个 byte 到 dest 内存区域。内存区域可能重复。
返回值:
函数 memmove()返回指向 dest 的指针。

19. strerror - 返回字符串描述的错误代码
格式:
#include <string.h>
char *strerror(int errnum);
说明:
函数 strerror() 返回“errno.h”中定义的描述错误代码的字符串，由 errnum 传递参数。该字符串必须没有被应用程序修改。没有哭函数将修改该字符串。
返回值:
函数 strerror() 返回合适的错误描述字符串，在错误代码不明确的情况下返回未知的错误信息。的值 errno 调用成功时不会改变，调用失败时被设置为非 0 值。

20. memchr - 在内存中扫描某个字符
格式:
#include <string.h>
void *memchr(void *s, int c, size_t n);
说明:
函数 memchr()扫描 s 所指向的内存区域中查找字符 c。在与 c 相匹配(被翻译为 unsigned character)的第一个字节处停止。
返回值:
函数 memchr()返回与 c 相匹配的第一个字节；否则返回 NULL。

21. sinf - sine 函数
格式:
#include <math.h>

float sinf(float x);
说明:
函数 sinf()返回 x 的正弦值，x 是给定的弧度值。
返回值:
函数 sinf()返回值在-1~1 之间。

22. cosf - cosine 函数
格式:
#include <math.h>
float cosf(float x);
说明:
函数 cosf()返回 x 的余弦值，x 是给定的弧度值。
返回值:
函数 cosf()返回值在-1~1 之间。

23. tanf - tangent 函数
格式:
#include <math.h>
float tan(float x);
说明:
函数 tanf()返回 x 的正切值，x 是给定的弧度值。

24. asinf - 反正弦函数
格式:
#include <math.h>
float asinf(float x);
说明:
函数 asinf() 计算 x 的反正弦值；即其正弦值为 x。若 x 在-1~1 的范围外，asinf()调用失败，设置为 errno。
返回值:
函数 asinf()返回弧度的反正弦值，且值被定义在-PI/2 ~ PI/2(包括边界)之间。

25. acosf - 反余弦函数
格式:
#include <math.h>
float acosf(const float x);
说明:
函数 acosf() 计算 x 的反余弦值；其余弦值为 x。若 x 在-1~1 的范围外，acosf()调用失败，设

置为 errno。
返回值:
函数 acosf() 返回弧度的反余弦值，且值被定义在 0 ~ PI （包括边界）之间。

26. atanf - 反正切函数
格式:
#include <math.h>
float atanf(const float x);
说明:
函数 atanf()计算 x 的反正切值；即其正切值为 x。
返回值:
函数 atanf()返回弧度的反正切值，且其值在 -PI/2 ~PI/2 (包含边界) 范围内。

27. atan2f - 含有两个变量的反正切函数
格式:
#include <math.h>
float atan2f(float y, float x);
说明:
函数 atanf()计算 x 和 y 的反正切值。除两个参数都用于决定结果是否为 90 外，与计算 y/x 的反正切值相似。
返回值:
函数 atanf()返回值为弧度，在 -PI ~ PI（包括边界）范围内。

28. sinhf - 双曲线正弦函数
格式:
#include <math.h>
float sinhf(float x);
说明:
函数 sinhf() 返回 x 的双曲线正弦值，被定义为：(exp(x) - exp(-x)) / 2。

29. coshf - 双曲线余弦函数
格式:
#include <math.h>
float coshf(float x);
说明:
函数 coshf()返回 x 的双曲线余弦值，被定义为：(exp(x) + exp(-x)) / 2。

30. tanhf - 双曲线正切函数

格式:
#include <math.h>
float tanhf(float x);
说明:
函数 tanhf()返回 x 的双曲线正切值，被定义为：sinhf(x) / coshf(x)。

31. Expf - 幂函数
格式:
#include <math.h>
float expf(float x);
说明:
函数 expf()返回 e(自然对数体系中的底数)的 x 次方所得的幂值。

32. logf - 对数函数
格式:
#include <math.h>
float logf(float x);
说明:
函数 logf() 返回 x 的自然对数。
log10f - 对数函数
格式:
#include <math.h>
float log10f(float x);
说明:
函数 log10f() 返回参数 x 以 10 为底的对数。

33. powf - 幂函数
格式:
#include <math.h>
float powf(float x, float y);
说明:
函数 powf()返回以 x 为底的 y 次幂。

34. sqrtf - 平方根函数
格式:
#include <math.h>
float sqrt(float x);
说明:

函数 sqrtf()返回 x 的非负数平方根。若调用失败，若 x 为负数，这设置 errno 为 EDOM。

35. fabsf - 浮点数绝对值函数
格式:
#include <math.h>
float fabsf(float x);
说明:
函数 fabsf 返回浮点数 x 的绝对值。

36. frexpf - 将浮点数分为整数部分和小数部分的函数
格式:
#include <math.h>
float frexpf(float x, int *exp);
说明:
函数 frexpf()用于建 x 分解为标准小数和 exp 中的指数两个部分
返回值:
函数 frexpf()返回标准小数。若 x 不是 0，标准小数是 2 的 x 次方所得的幂，范围在 1/2(包含 1/2) ~1(不包含 1)之间。若 x 为 0，标准小数值为 0 且被存放至 exp 中。

37. ldexpf - 浮点数与以 2 为底的幂的乘法
格式:
#include <math.h>
float ldexpf(float x, int exp);
说明:
函数 ldexpf()返回参数 x 与 2 的 exp 次方的乘积：x* (2 exp)。

38. ceilf - ceiling 函数：不小于参数的最小整数
格式:
#include <math.h>
float ceilf(float x);
说明:
该函数找到不小于 x 的最近整数。
返回值:
函数返回参数不小于 x 的最小整数。若 x 是整数或无穷大，返回 x 本身。

39. floorf - 查找不大于参数的最大整数的函数
格式:
#include <math.h>

float floor(float x);
说明:
查找不大于 x 的最近整数。
返回值:
函数返回参数不大于 x 的最大整数。若 x 为整数或者无穷大，返回 x 本身。

40. modff - 该函数将浮点数分割为小数部分和整数部分。
格式:
#include <math.h>
float modff(float x, float * y);
说明:
函数 modff 将浮点数 x 分割为小数部分和整数部分，两部分均与 x 的标记相同。返回 x 的已标记小数部分。将整数部分存储为浮点数 y。
返回值:
该函数返回 x 的已标记小数部分。无返回错误。

41. fmodf - 浮点数求余函数
格式:
#include <math.h>
float fmodf(float x, float y);
说明:
函数计算参数 x/y 的余数。返回 x - n * y，其中 n 为 x/y 的商，在 0 到整数范围里。
返回值:
函数返回参数 x/y 的余数，除非为 0，在函数调用失败时，设置 errno。

42. atof - convert a string to a double
格式:
#include <stdlib.h>
float atof(const char * nptr);
说明:
函数 atof()将 nptr 所指的字符串初始位置转换为 double 类型。
返回值:
转换后的值。

43. atoi, atol- 将字符串转换为整数
格式:
#include <stdlib.h>
int atoi(const char * nptr);

long atol(const char * nptr);
说明:
函数 atoi()将 nptr 指向的字符串初始位置转换为 int 类型
函数 atol()转换字符串初始位置为其返回值类型 long
返回值:
转换后的值。

44. abs- 计算整数的绝对值
格式:
#include <stdlib.h>
int abs(int j);
说明:
函数 abs()计算整数参数 j 的绝对值。
返回值:
返回函数参数相应类型的绝对值。

45. labs- 计算整数的绝对值
格式:
#include <stdlib.h>
long int labs(long j);
说明:
函数 labs() 计算函数参数 j 的绝对值。
返回值:
返回函数参数相应类型的绝对值。

46. div - 计算整数除法的商和余数
格式:
#include <stdlib.h>
div_t div(int numer, int denom);
说明:
函数 div()计算 numer/denom 的值，并返回包含两个成员 quot 和 rem 的 div_t 结构体的商和余数。
返回值:
The div_t structure。

47. ldiv - 计算 long int 型整数的商和余数
格式:
#include <stdlib.h>

ldiv_t ldiv(long int numer, long int denom);
说明:
函数 div()计算 numer/denom 的值，并返回包含两个 long int 成员（quot 和 rem）的 ldiv_t 结构体的商和余数。 商与 0 接近。
返回值:
机构体 ldiv_t。

48. rand - 产生随机数字
格式:
#include <stdlib.h>
int rand(void);
说明:
函数 rand()返回 0 ~ RAND_MAX 间的一个伪随机整数。
若提供了随机数种子值(seed value)，rand()则自动赋值为 1。
返回值:
函数 rand()返回 0 ~ RAND_MAX 中的一个随机值。

49. srand - 产生随机数字
格式:
#include <stdlib.h>
void srand(unsigned int seed);
说明:
函数 srand()将其参数设置为序列种子，并返回伪随机整数序列。这些序列是重复调用相同种子值的 srand()产生的。
若未提供种子值，srand()则自动赋值为 1。
返回值:
不返回任何值。

50. strtol - 将字符串转换为 long 类型
格式:
#include <stdlib.h>
long strtol(const char *nptr, char **endptr, int base);
说明:
函数 strtol()根据所给的基数，将 nptr 中字符串的初始位置转换为一个 long int 值，范围 2 ~ 36 (包含边界值)。或者为特殊值 0。
字符串须以任意个空格开头（取决于 isspace(3)），后面是+号或-号。若基数是 0 或者 16，字符串可能要以 0x 开头的十六进制数；否则，以 0 为基数会被当作十进制数，若后面的字符还是 0，则是八进制数。

字符串的余数被转换为 long int 型，在当前进制下的第一个不合法字符处结束。十进制中，A 在大小写的情况下都是 10，B 是 11，以此类推，Z 为 35。
若 endptr 非 NULL，strtol()存储*endptr 的第一个不合法字符。若没有任何数字，strtol()存储*endptr 中 nptr 的初始值，并返回 0。若*nptr 不是\0，**endptr 则返回\0，整个字符串是合法的。
返回值:
函数 strtol()返回转换结果，除非结果下溢或者上溢。若下溢，strtol()返回 LONG_MIN；若上溢，strtol()返回 LONG_MAX。若两者同时发生，errno 被设置为 ERANGE。

51. strtoul - 将字符串转换为 unsigned long 类型
格式:
#include <stdlib.h>
unsigned long strtoul(const char *nptr, char** endptr, int base);
说明:
函数 strtoul()根据所给的基数，将 nptr 中字符串的初始位置转换为一个 unsigned int 值，范围 2 ~ 36 (包含边界值)。或者为特殊值 0。
字符串须以任意个空格开头（取决于 isspace(3)），后面是+号或-号。若基数是 0 或者 16，字符串可能要以 0x 开头的十六进制数；否则，以 0 为基数会被当作十进制数，若后面的字符还是 0，则是八进制数。
字符串的余数被转换为 long int 型，在当前进制下的第一个不合法字符处结束。十进制中，A 在大小写的情况下都是 10，B 是 11，以此类推，Z 为 35。
若 endptr 非 NULL，strtoul()存储*endptr 的第一个不合法字符。若没有任何数字，strtoul()存储*endptr 中 nptr 的初始值，并返回 0。若*nptr 不是\0，**endptr 则返回\0，整个字符串是合法的。
返回值:
函数 strtoul 返回转换结果；若有负号存在而否定了转换结果，除非初始值溢出，strtoul()返回 ULONG_MAX 且将全局变量 errno 设置为 ERANGE。

附录一 RISC 指令集

RISC 内核目前有两种产品分别为 14 位和 16 位两种。16 位比 14 位多出几要指令。

助记符	指令说明	周期数	影响标志位
ADDAR R	R+A-->R	1	Z,DC,C
ADDRA R	R+A-->A	1	Z,DC,C
ANDAR R	R&A-->A	1	Z
ANDRA R	R&A-->R	1	Z
CLRR R	0-->R	1	Z
CLRA	0-->A	1	Z
COMAR R	/R-->A	1	Z
COMRA R	/R-->R	1	Z
DECAR R	R-1-->A	1	Z
DECR R	R-1-->R	1	Z
DJZAR R	R-1-->A,SKIP if 0	1(2)	-
DJZR R	R-1-->R,SKIP if 0	1(2)	-
INCAR R	R+1-->A	1	Z
INCR R	R+1-->R	1	Z
JZAR R	R+1-->A,SKIP if 0	1(2)	-
JZR R	R+1-->R,SKIP if 0	1(2)	-
ORAR R	R|A-->A	1	Z
ORRA R	R|A-->R	1	Z
MOVAR R	R-->A	1	Z
MOVR R	R-->R	1	Z
MOVRA R	A-->R	1	-
RLAR R	R<<1-->A	1	C
RLR R	R<<1-->R	1	C
RRAR R	R>>1-->A	1	C
RRR R	R>>1-->R	1	C

RSUBAR R	R-A-->A	1	Z,DC,C
RSUBRA R	R-A-->R	1	Z,DC,C
SWAPAR R	R 半字节交换-->A	1	-
SWAPR R	R 半字节交换-->R	1	-
XORAR R	R 异或 A-->A	1	Z
XORRA R	R 异或 A-->R	1	Z
JBCLR R,b	if R[b]==0,SKIP	1(2)	-
JBSET R,b	if R[b]==1,SKIP	1(2)	-
BCLR R,b	0-->R[b]	1	-
BSET R,b	1-->R[b]	1	-
ADDAI I	A+I-->A	1	Z,DC,C
ANDAI I	A&I-->A	1	Z
ORAI I	A|I-->A	1	Z
MOVAI I	I-->A	1	-
RETAI I	Stack-->PC,I-->A	2	-
ISUBAI I	I-A-->A	1	Z,DC,C
XORAI I	A 异或 I-->A	1	Z
ADCAI I	I+A+C-->A	1	Z,DC,C
ISBCAI I	I-A-/C-->A	1	Z,DC,C
MULAR R	R*A -> HIBYTE,A	1	-
MULRA R	R*A -> HIBYTE,R	1	-
ADCAR R	R+A+C-->A	1	Z,DC,C
ADCRA R	R+A+C-->R	1	Z,DC,C
RSBCAR R	R-A-/C-->A	1	Z,DC,C
RSBCRA R	R-A-/C-->R	1	Z,DC,C
ASUBAR R	A-R-->A	1	Z,DC,C
ASUBRA R	A-R-->R	1	Z,DC,C
ASBCAR R	A-R-/C-->A	1	Z,DC,C
ASBCRA R	A-R-/C-->R	1	Z,DC,C

CLRWDT	0-->WDTCNT	1	-
RETIE	Stack-->PC,1-->GIE	2	-
RETURN	Stack-->PC	2	-
STOP	进入待机模式	1	-
NOP	None Operation	1	-
DAA	加法后十进制调整	1	DC，C
DSA	减法后十进制调整	1	DC，C
CALL I	I-->PC,PC-->Stack	2	-
GOTO I	I-->PC	2	-

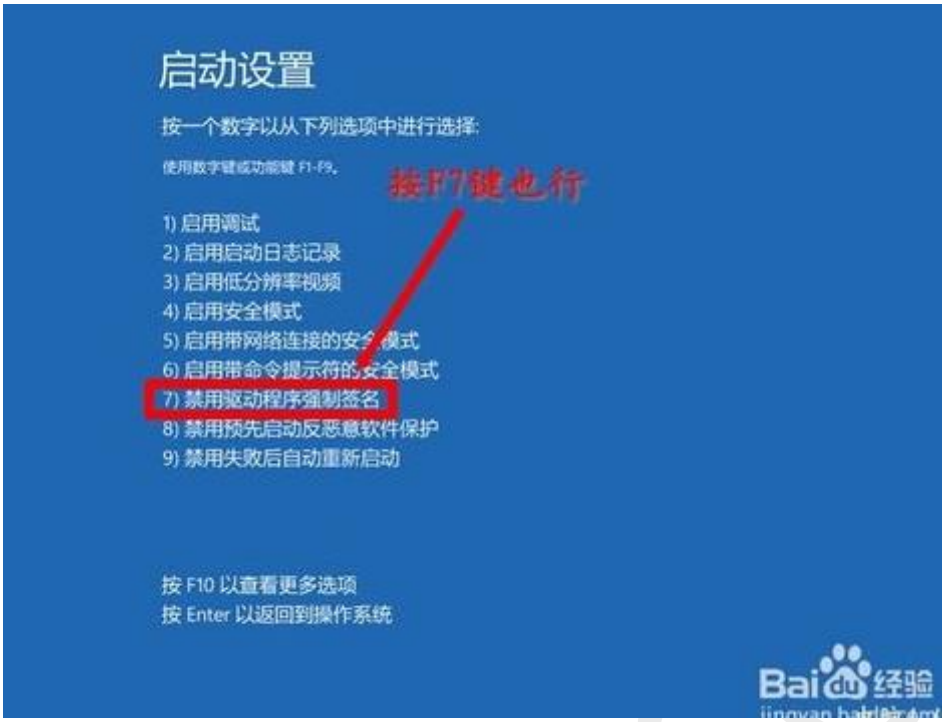
附录二 win 8.1 强制禁用数字签名方法

目前烧写软件、仿真软件安装驱动时需把我们的 Driver 中对应系统位文件夹中的 usbser.sys 文件拷贝到 C:\Windows\System32\drivers 中，然后才能进行驱动程序的安装。由于 win8 对第三方数字签名要求严格，故给出在 win8 中针对禁用数字签名的方法，如下：

1. 按 Win+C 组合键，调出 Charm 菜单→设置
2. 点选左边设置选项卡中的 常规 菜单，再点击右边的 立即启动 即会重启电脑。
3. 来到这个选项卡界面，点选 疑难解答 选项。

4. 再在 高级选项 中选择 Windows 启动设置，继续点选 重新启动。

5. 然后会来到这个设置界面，选择 禁用驱动程序强制签名 （或按 F7 键）重启电脑即可。



附录三 SM112X 芯片仿真 SM110X 说明

使用 SM112X 仿真芯片编写 SM110X 程序 ，
配置位 P0SEL，P0RES，LOADS，RCOUT 都必须设置 1