



如何基于STM8S系列MCU 进行项目开发

简介

意法半导体的STM8S系列8位闪存微控制器为工业应用和家电市场提供理想解决方案。最新版的微处理器内核，结合3段流水线架构，使STM8S微控制器具备最优异的性能。直观的开发环境简单易用，使产品上市时间更短。

- **全新STM8微控制器内核**

3级流水线哈佛架构，24 MHz时最高处理性能20 MIPS (www.stmicroelectronics.com.cn/stm8)

- **先进的嵌入式130nm EEPROM技术**

ST独有的嵌入式非易失性存储器，EEPROM存储器性能优异，采用高密度CMOS制程，并提供最出色的模拟特性

- **最先进的外设接口**

最基本的外设接口，如高速SPI、I2C、USART、LIN-UART、CAN、IrDa、智能卡、CAN，以及高端16位定时器和快速、精确的模数转换器

- **成套的功能丰富的开发工具**

从提供单线调试接口的入门级配置，到具有跟踪、评估和代码覆盖分析功能的复杂仿真器，各种开发工具应有尽有。第三方编译器集成在综合开发环境（IDE）内

目录

1	准备	3
1.1	如何连接到STM8S芯片主页	3
1.2	下载STM8S系列相关资料	4
1.3	下载及安装软件工具	4
1.3.1	STVD IDE集成开发环境	4
1.3.2	COSMIC C语言编译器	6
1.3.3	安装调试工具	10
2	STM8项目开发举例	11
2.1	硬件设计	11
2.1.1	电源	11
2.1.2	Vcap	12
2.1.3	复位电路	12
2.1.4	时钟	13
2.1.5	I/O口的分配	13
2.2	软件设计	13
2.2.1	项目建立	13
2.2.2	软件编写注意事项	19
2.2.3	在线调试	19
3	进一步掌握STVD/COSMIC	25
3.1	如何分配变量到指定的地址	25
3.2	如何在COSMIC C文件中使用汇编语言	25
3.3	如何观察RAM/FLASH/EEPROM的最终分配情况	26
3.4	如何生成hex格式的输出文件	27
3.5	什么是MEMORY MODEL	27
3.6	.lcf 文件的作用	28
3.7	如何实现位操作	30
3.8	_stext是什么以及初始化程序库的意义	32

1 准备

1.1 如何连接到STM8S芯片主页

在进行STM8系列的芯片开发前，请先到ST公司主页下载相关文档。可通过以下网址连接到ST公司MCU产品的主页：

<http://mcu.st.com> 或者 <http://www.st.com/mcu>

或中文网站：

<http://www.stmicroelectronics.com.cn/mcu/>

The first screenshot shows the STMicroelectronics homepage with the 'PRODUCTS' menu highlighted. The second screenshot shows the 'Microcontrollers' page with '8-bit Microcontrollers' circled in the 'Microcontroller Families' section. The third screenshot shows the 'Documents and files for family STM8 - 8-bit Microcontrollers' page, with a red arrow pointing to the 'Documents and Files for STM8S family' link.

Documents and files for family STM8 - 8-bit Microcontrollers

Please report broken links !

Available Documents
Advertising | Application Note | Brochure / Flyer | Certification | Datasheet | Errata Sheet | Firmware | Presentation - Marketing | Release Note | Software for Tools | User Manual | Reference Manual | Release Note |

Development Tools
Data Briefing | Reference Schematics | Release Note | Software for Tools | Software Patches | User Manual |

Reference	Description	Version	Date	Size	File	File
	STM8S PRODUCT PRESENTATION USED DURING PRESS CONFERENCES		Mar-2009	1213 KB		

Reference	Description	Version	Date	Size	File	File
AN2857	STM8S family power management	2	Jul-2009			
AN2658	Using the analog to digital converter of the STM8 microcontroller	2	Jul-2009			
AN2687	STM8S20xxx LCD software driver	1	Apr-2009			
AN2762	Getting started with the STM8S	2	Apr-2009			

STM8S系列相关的资料 and 软件都可以在这个页面找到

1.2 下载STM8S系列相关资料

所有的STM8S芯片的相关资料都可以在1.1节中所提到的网页内下载。进行STM8S系列MCU的开发首先需要以下资料：

- Reference Manual:
RM0016 STM8S microcontroller family 对STM8S系列MCU各模块的功能做详细介绍
- Programming manual:
PM0044 STM8 CPU programming 详细介绍STM8S系列MCU的CPU的指令集，寻址方式。
PM0047 STM8 FLASH programming 详细介绍STM8S系列Flash的编程方式。
- Datasheet: 数据手册
简单罗列了相应MCU的具体所包含的功能模块。并对引脚定义和电气特性和封装、订购信息做了说明。用户可根据所选择的STM8S系列的具体型号找到相应的数据手册。
- Application note: 应用笔记

1.3 下载及安装软件工具



在开始STM8S系列MCU开发前，需要下载并安装下列软件。

1.3.1 STVD IDE集成开发环境

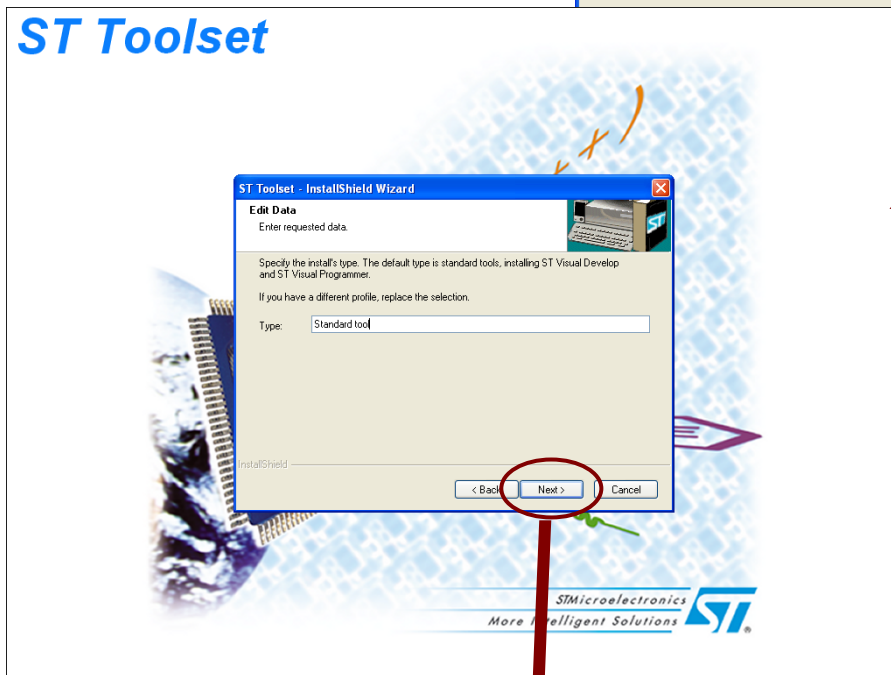
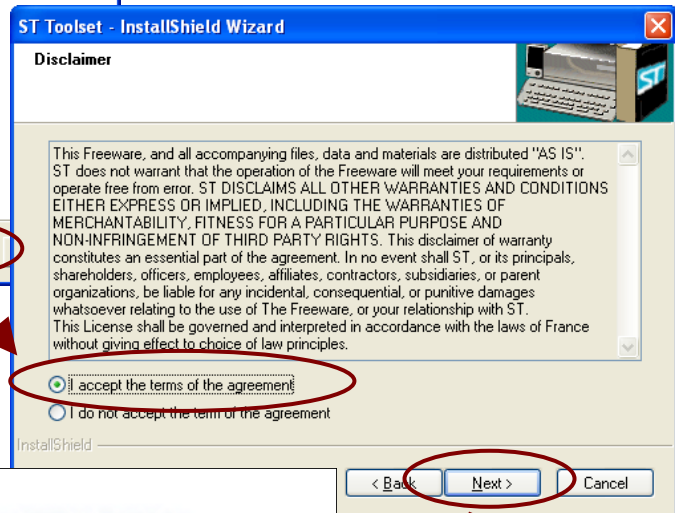
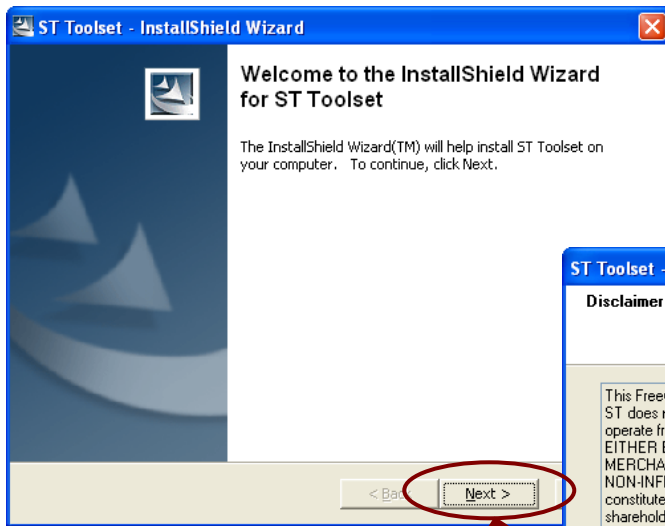
STVD是ST公司为ST的8位单片机的开发提供的一个免费的集成开发调试软件。

- STVD可以支持大部分支持STM8的在线调试工具（Rlink, ST-Link）
- 包含免费的汇编工具
- 可以选择外挂第三方的C语言编译器
- 捆绑专用编程工具STVP

可以从1.1节中所示的网页里找到下载链接。

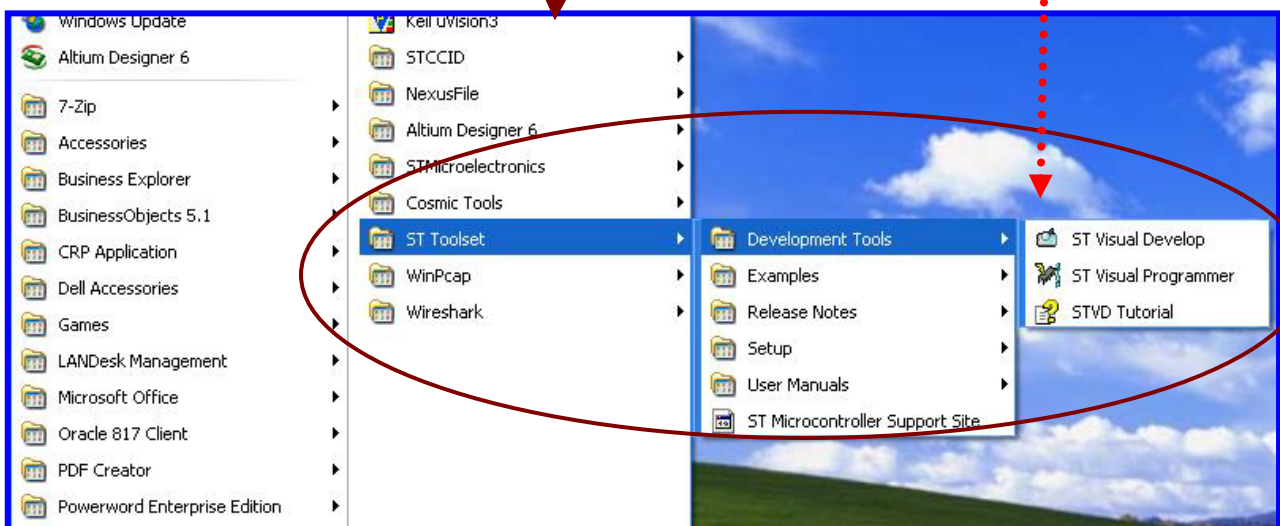
Software for Tools						
Reference	Description	Version	Date	Size	File	File
ST toolset	ST Visual Develop (STVD) 4.1.2 and ST Visual Programmer (STVP) 3.1.1 software releases – Software package includes IDE with advanced editor, project builder (supports Cosmic, and Raisonance C toolchains, and included ST assembler/linker), debugger with simulator, plus programming interface. Supports ST emulators, in-circuit debugger, Raisonance RLink and ST MCU programming tools.	Pack 18	May-2009			

下载完成后，运行可执行文件开始安装。安装界面如下



安装完成后的目录结构:

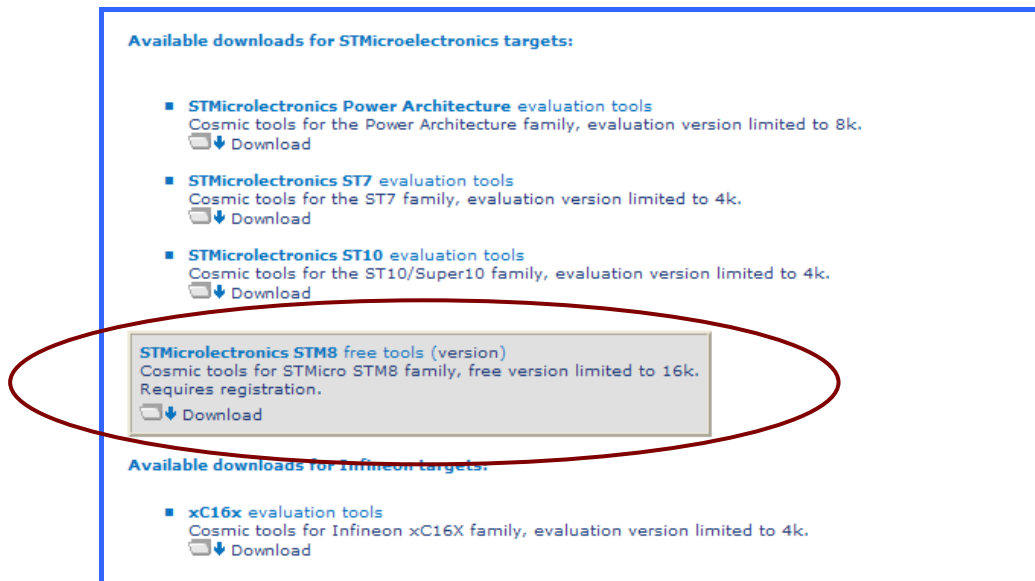
- STVD 集成开发环境
- STVP 专用编程软件
- STVD 教程



1.3.2 COSMIC C语言编译器

可以从COSMIC 公司网站下载免费的16k版本的C语言编译器。

<http://www.cosmicsoftware.com/download.php>



Cosmic Software
Supporting Embedded Innovation Since 1983

ABOUT US NEWS & EVENTS CONTACT US PRODUCTS & SERVICES SUPPORT DOWNLOAD

Home / Download / stm8 FREE 16k

Download of the FREE stm8 16k version

Fill and submit the form below to download the free stm8 compiler 16K version.
To use this product you must register with Cosmic Software. The installation procedure will instruct you to send a message to Cosmic Software at stm8_16k@cosmic.fr to perform this registration. As a result you will receive the appropriate free license for this product.

* Name

* Company

Address

ZIP Code

City

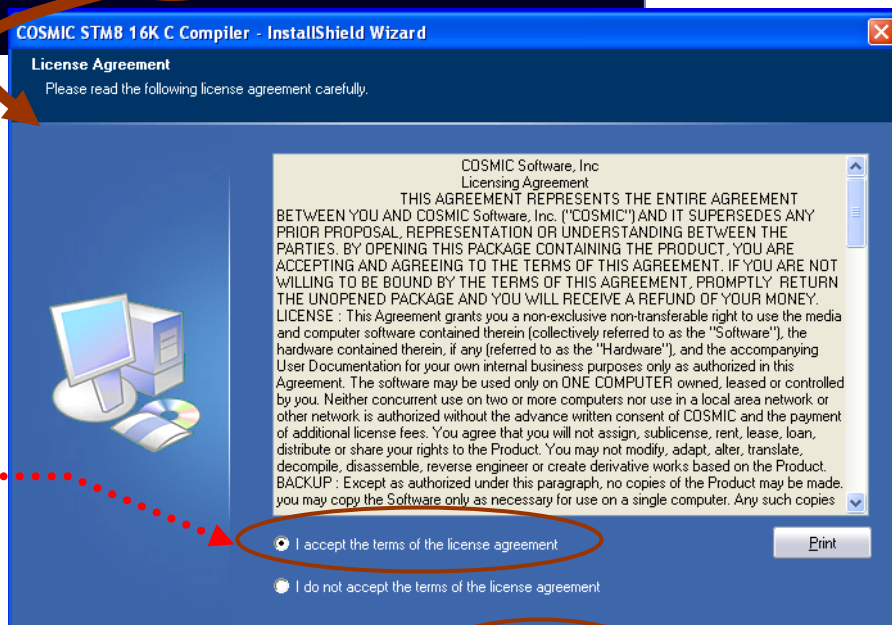
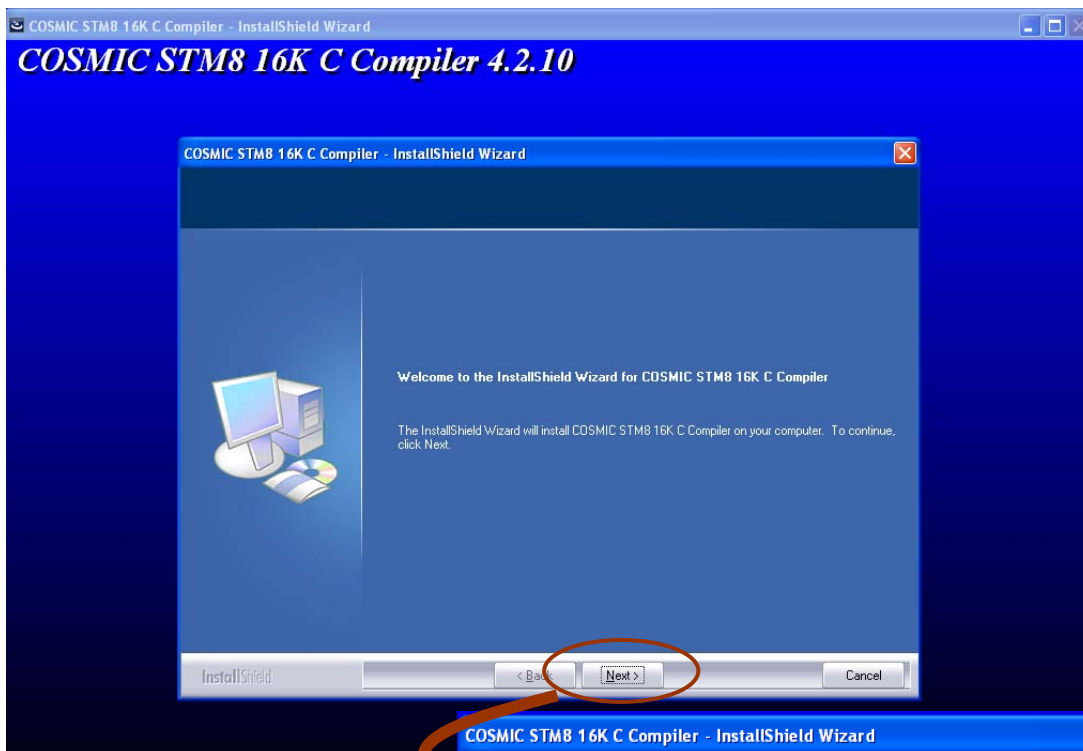
* Country -- Select --

Phone

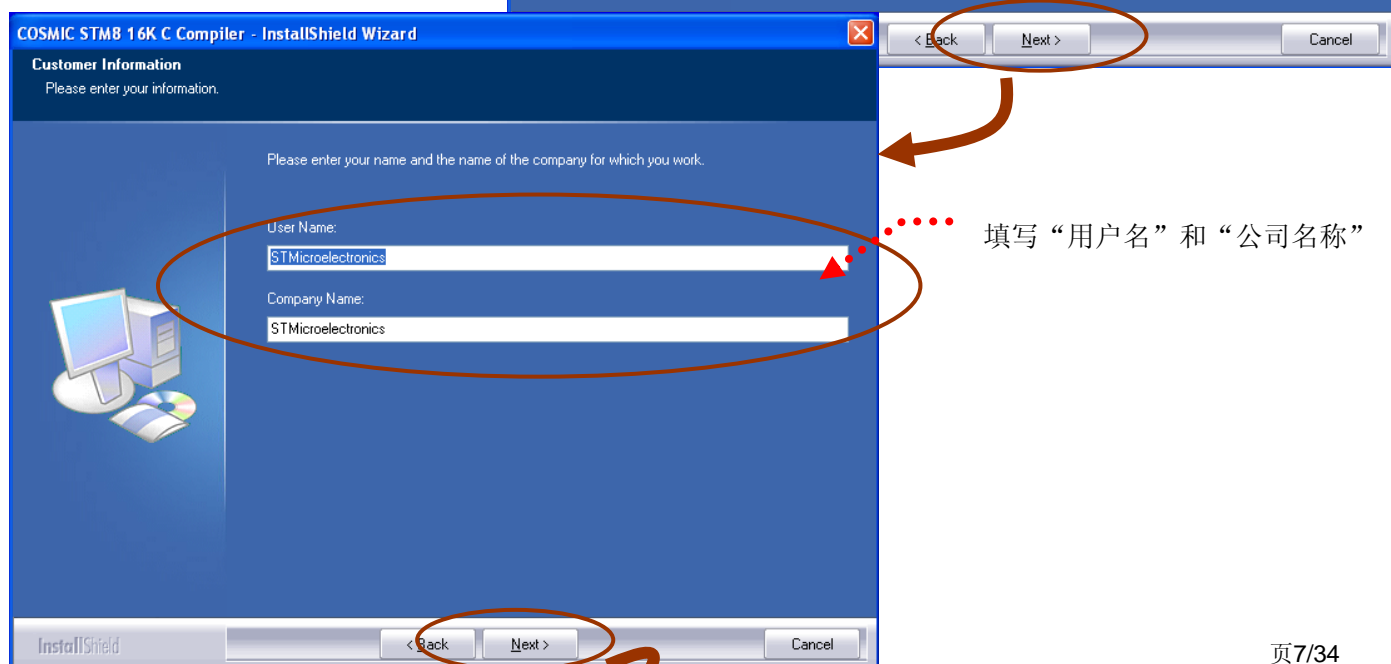
Fax

* E-mail

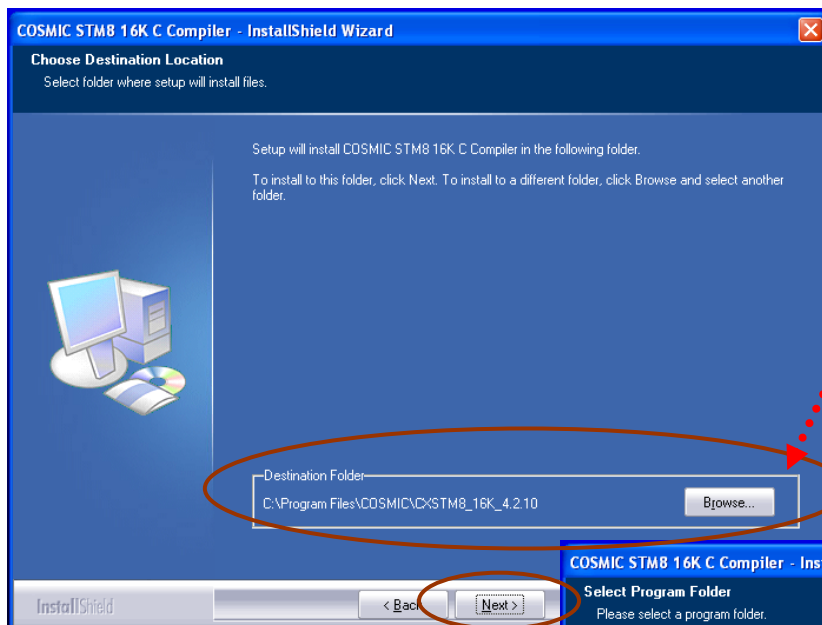
填写完上图中的注册信息后会弹出下载链接。下载完安装包后，按默认选项安装



选择同意 Licensing Agreement 所列的条款。

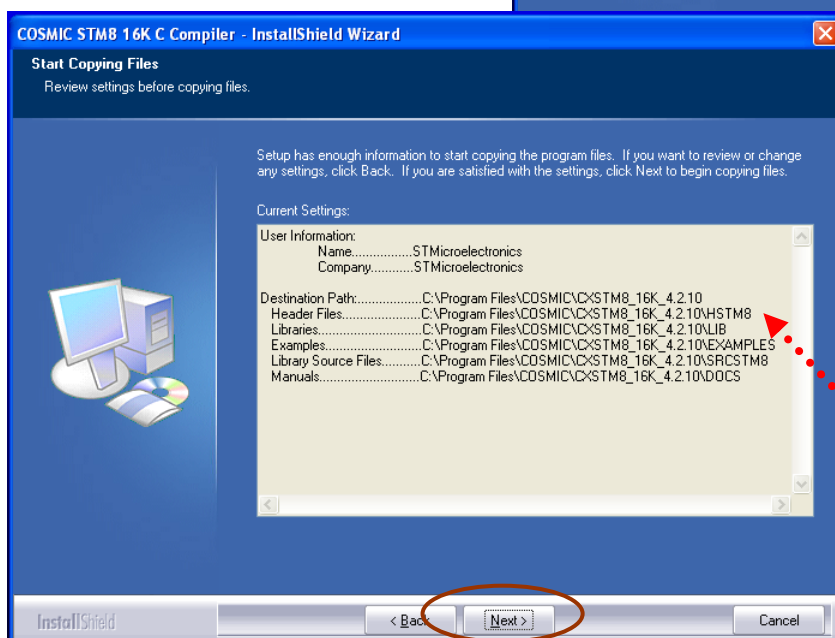
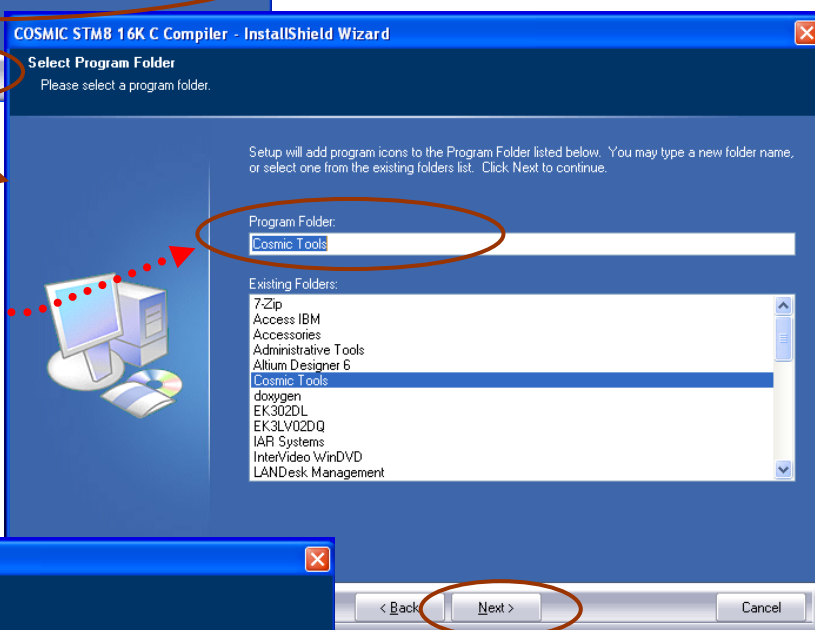


填写“用户名”和“公司名称”

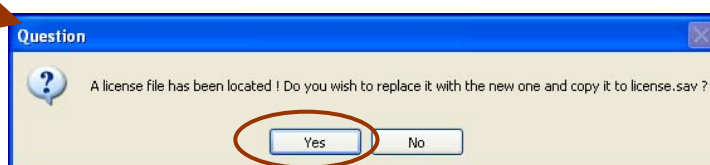


选择COSMIC C 编译器的安装路径，或直接安装在默认路径下

选择显示在程序文件夹中的文件夹名称，建议用默认的“Cosmic Tools”

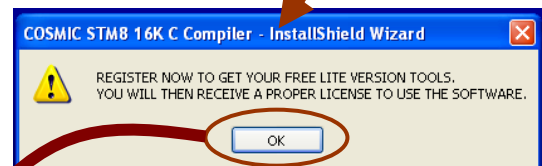
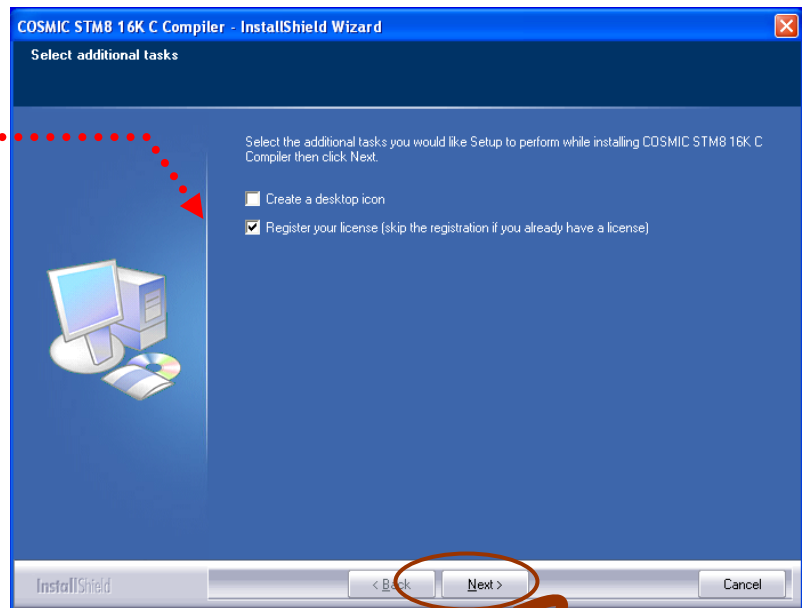


将前面几步骤所填的信息供用户核实。若有问题，按“Back”返回前页修改



选择:

- 1) 在桌面创建快捷方式。
- 2) 注册license(如果已经有了license文件, 可以不选择此项)。



REGISTER NOW TO GET YOUR FREE LITE VERSION TOOLS

PRODUCT=LXSTM816K
HOSTID="001641575fda 444553544200 00166f8de535"
USER=jacky xu
DISPLAY=SHZ10290
HOSTNAME=SHZ10290
DISK_SERIAL_NUM=c05b3ad6

User *: Jacky XU

Company *: STMicroelectronics

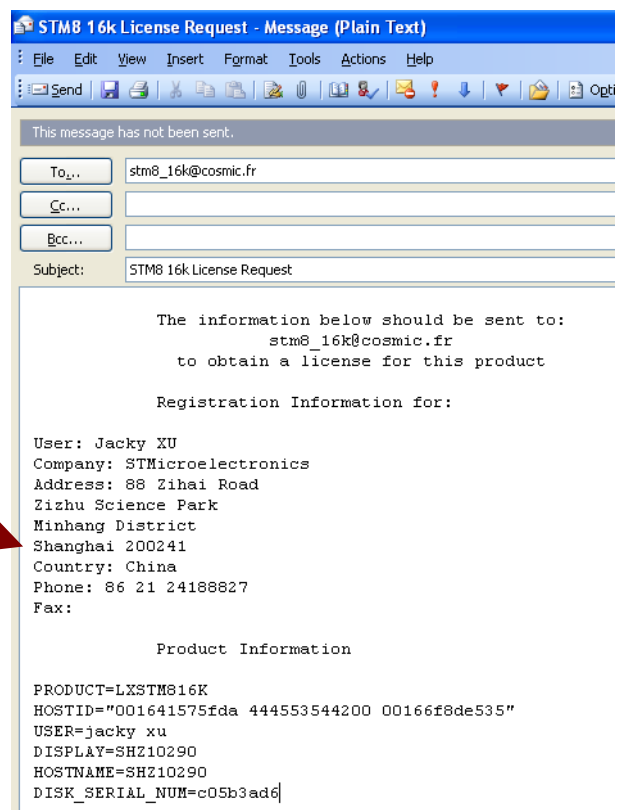
Address *:
88 Zihai Road
Zizhu Science Park
Minhang District
Shanghai 200241

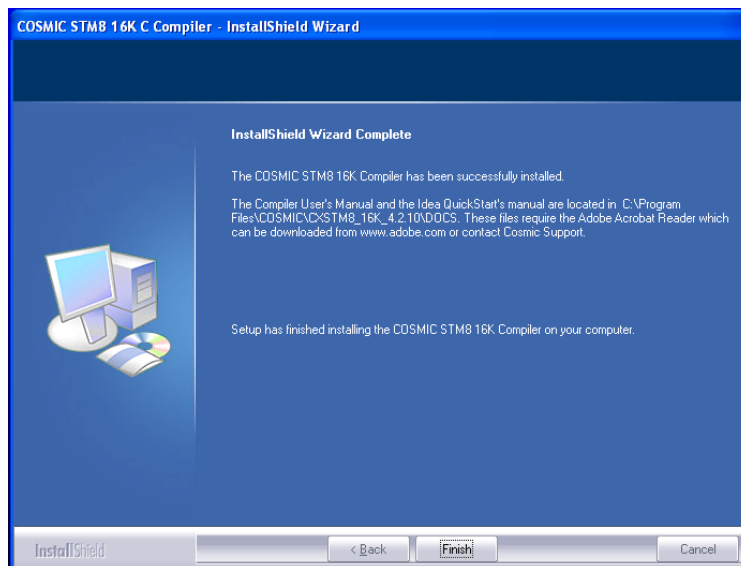
Country *: China

Phone *: 86 21 24188827 Fax: (Optional)

Register by Email Edit Email and Register Write to File Cancel

*: required.

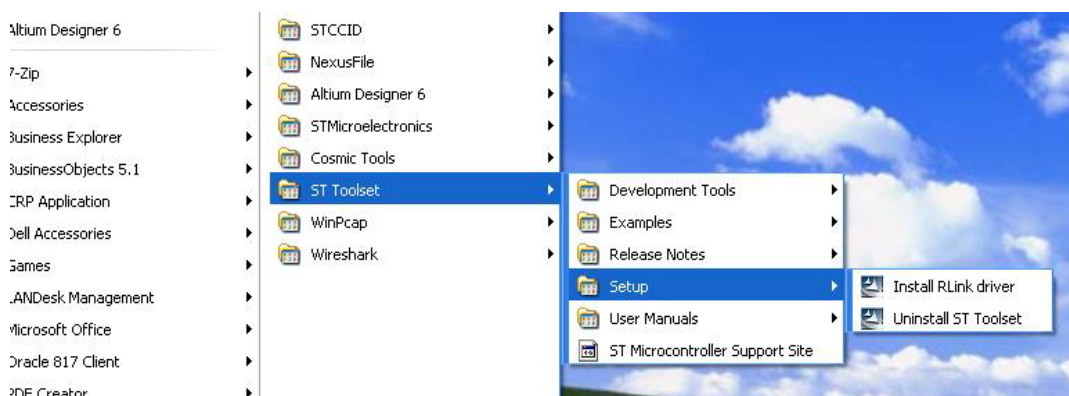




1.3.3 安装调试工具

在使用ST-LINK或者RLINK进行调试前，需要安装相应的驱动程序。

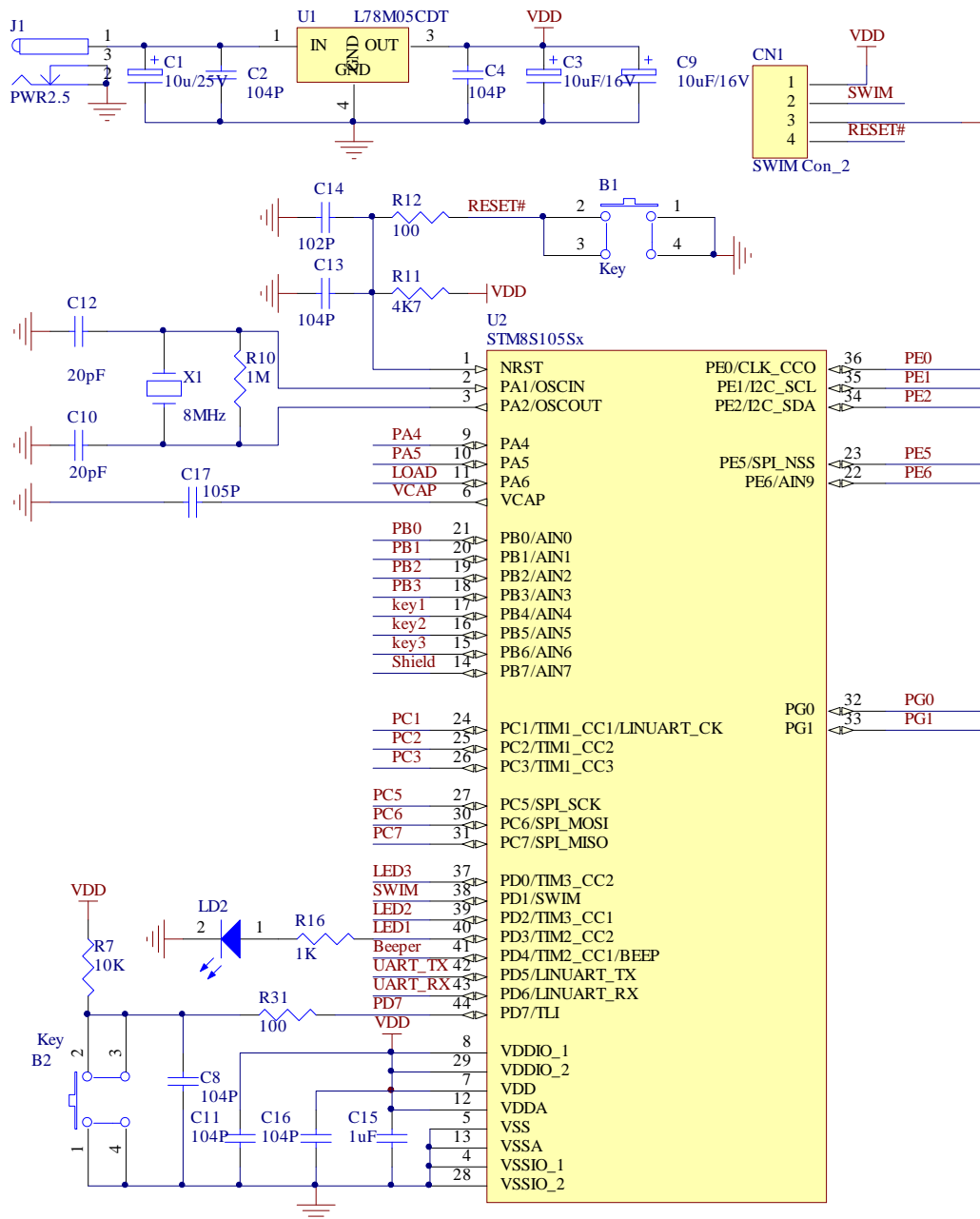
STVD自帶了ST-LINK的驱动。如果要使用RLINK可按照如下方式安装。



2 STM8 项目开发举例

2.1 硬件设计

下面的原理图利用STM8 MCU 实现按键点灯功能（以STM8S105S4-PKT评估板为例）。本节将通过这个原理图向大家介绍如何开始STM8的硬件开发，以及需要注意的地方。



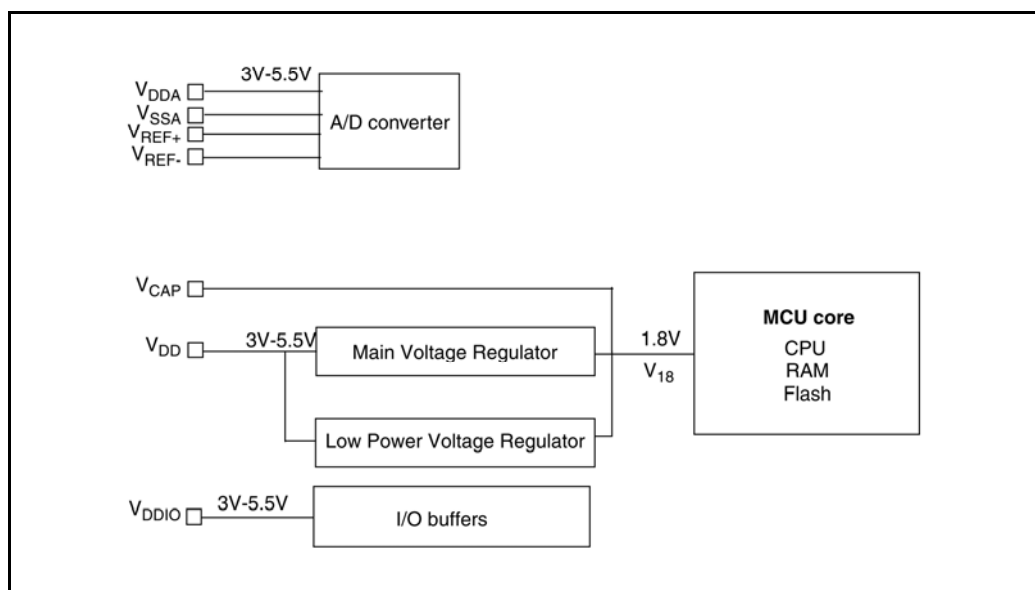
2.1.1 电源

STM8S系列单片机的工作电压约为2.95V ~ 5.5V（具体的电压以Datasheet提供的数据为准）。因此在设计时要注意保证MCU的供电电源在这个范围之内。

对于不同封装的STM8S MCU，最多会有下面这些电源引脚：

- VDD/VSS: 主电源 (3 V to 5.5 V)
- VDDIO/VSSIO: I/O口供电电源 (3 V to 5.5 V)

- VDDA/VSSA: 模拟电路的供电电源
- VREF+/VREF-: AD转换模块的参考电压



注意：要保证MCU的正常工作，必须将芯片所有的电源引脚都连接到相应的供电电源上。

每对VDD,VSS只须都必须加去耦电容。若VDD, VDDIO, VDDA是相邻的，则可供用一个去耦电容。例如，上面原理图中的引脚7和引脚8可以供用一个去耦电容。

在部分封装的MCU上，可能会有Vref+和Vref-引脚。这两个引脚的输入电压是ADC模块的参考电压。在没有这两个引脚的MCU上，这两个引脚是在MCU内部直接连接到VDDA和VSSA上。如果有外部电路提供参考电压，需要注意Vref+和Vref-的电压必须在VDDA和VSSA的范围内，例如：对于STM8S207来说：

$$2.75V < Vref+ < VDDA$$

$$VSSA < Vref- < 0.5V$$

2.1.2 Vcap

Vcap引脚是STM8S系列MCU内核供电电源的引出脚。为了保证内核能够正常运行，必须在Vcap引脚加去耦电容，并且要求距离MCU越近越好。在参考手册上，对这个去耦电容的要求是470nf，考虑到有些电容的偏差比较大，我们建议这个引脚上的电容取680nf~1uF比较合适。注意不能使用电解电容，其较差的高频特性不适合用于此处。

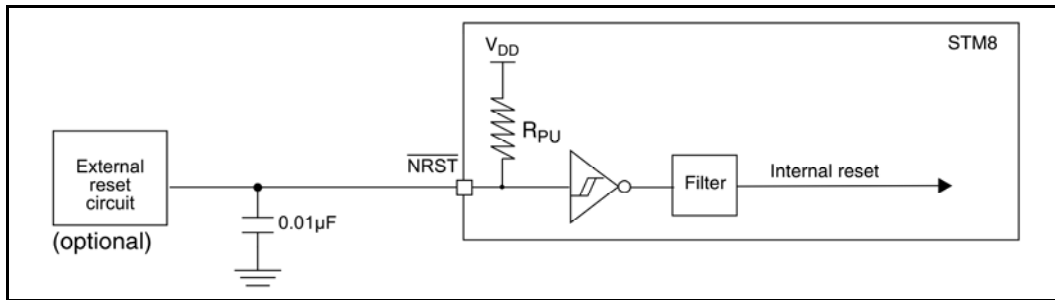
2.1.3 复位电路

STM8S的复位引脚NRST是一个输入/开漏输出的双向引脚，内部有一个约30~60k左右的弱上拉。

NRST引脚上最小500ns的低电平就会使MCU产生复位。同时NRST引脚也可利用开漏输出来使其他外部设备产生复位。

不管是什么原因引起的MCU复位都会在NRST引脚产生最少20us的低电平。

一般来说，复位电路可采用传统的外部RC方式，如上面的原理图所示。另外，由于MCU本身有内部弱上拉，因此外部的上拉电阻也可以不加。下图是数据手册提供的推荐电路。



2.1.4 时钟

STM8可使用外时钟或内时钟，当使用外时钟时，如果MCU主频超过16MHz，要在选项字节中配置等待周期为1。STM8的内时钟为16MHz，可根据需要进一步分频。其内部有3或4位的频率微调器，经过校正后其频率误差理论上可不大于0.5%(频率微调器为3位)或0.25%(频率微调器为4位)。

2.1.5 I/O口的分配

1. 要注意选项字节的配置，尤其注意I/O重映射功能状态是否与实际项目相符合
2. STM8的I2C接口为真正的开漏接口，意味着其没有内部上拉电阻和对电源的保护二极管。
3. 并非所有的I/O口都是大电流口，当需要I/O有很强驱动能力时要检查其是否需要外加驱动。
4. SWIM接口要保证上电时为稳定电平以防止MCU误进入调试模式。

2.2 软件设计

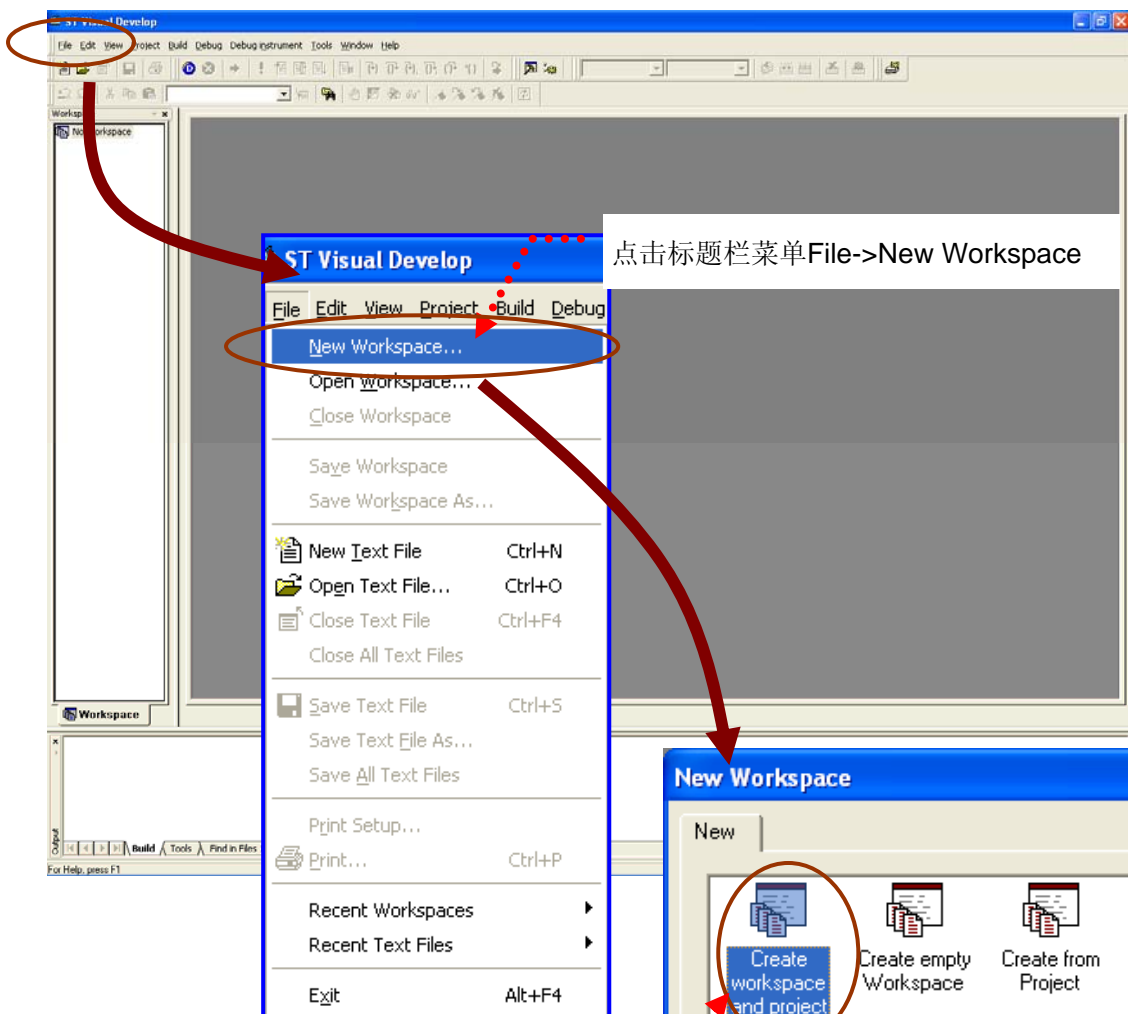
2.2.1 项目建立

STVD采用Workspace—Project的结构来进行管理及项目开发。一个Workspace可拥有一个或多个Projects，不同的Project之间可共享相同的源文件。下面将详述如何创建一个新的工程项目。

创建一个新目录，用于存放新工程项目相关文件。

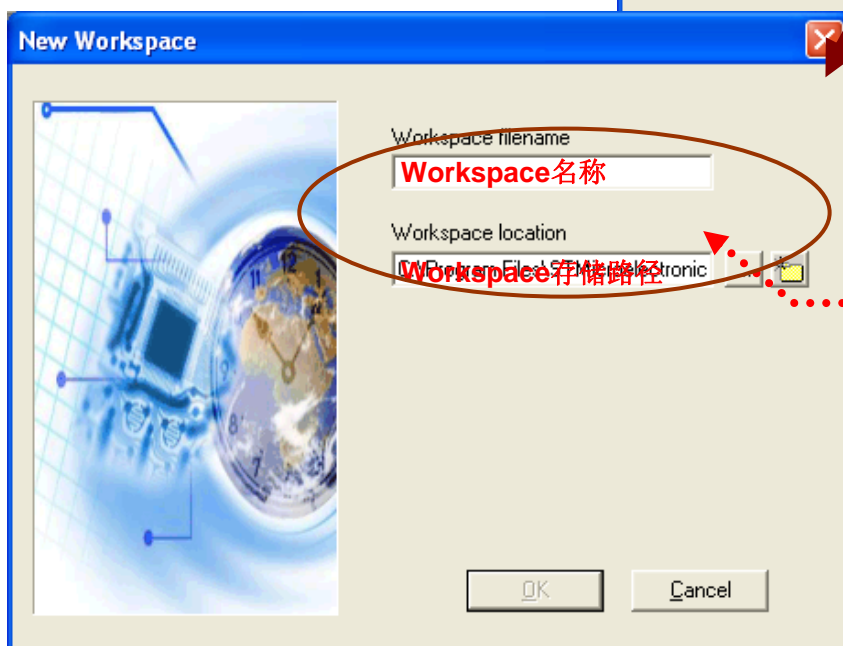
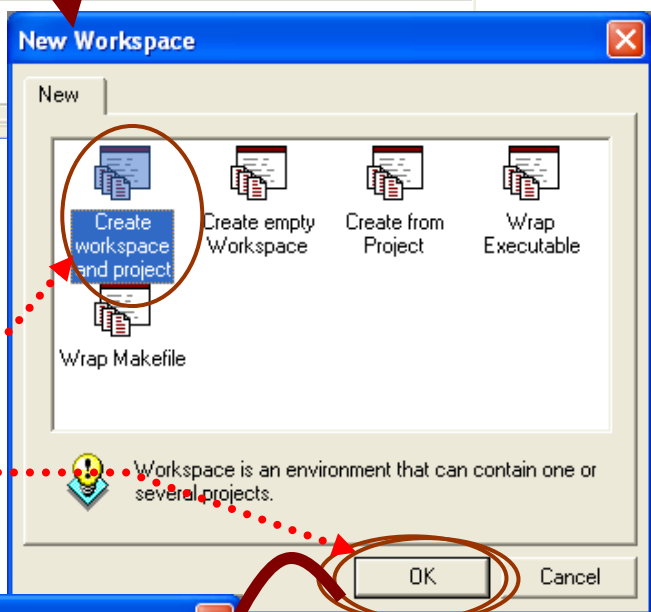
在本例中，所创建的目录为C:\STM8_NewProject1

运行STVD，程序打开后如下图所示。

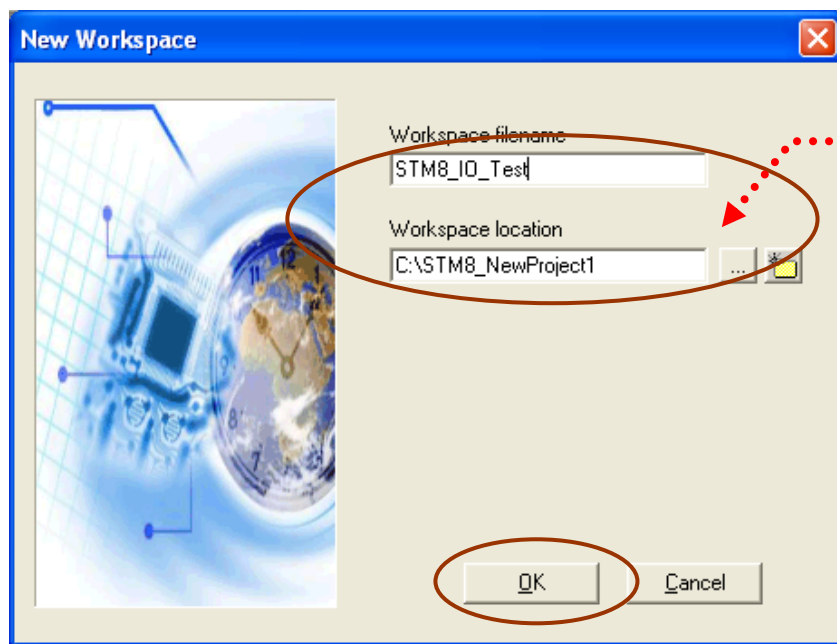


系统弹出 **New Workspace** 窗口，
选择第一项 **Create workspace and project** .

击OK按钮，系统弹出新的窗口
如下，用于设置Workspace。



填入Workspace名称，并选择
存储路径



本例对 Workspace filename 起名为 **STM8_IO_Test**。

存储路径为：

C:\STM8_NewProject1

确认后，弹出的 Project 及 Toolchain配置窗口

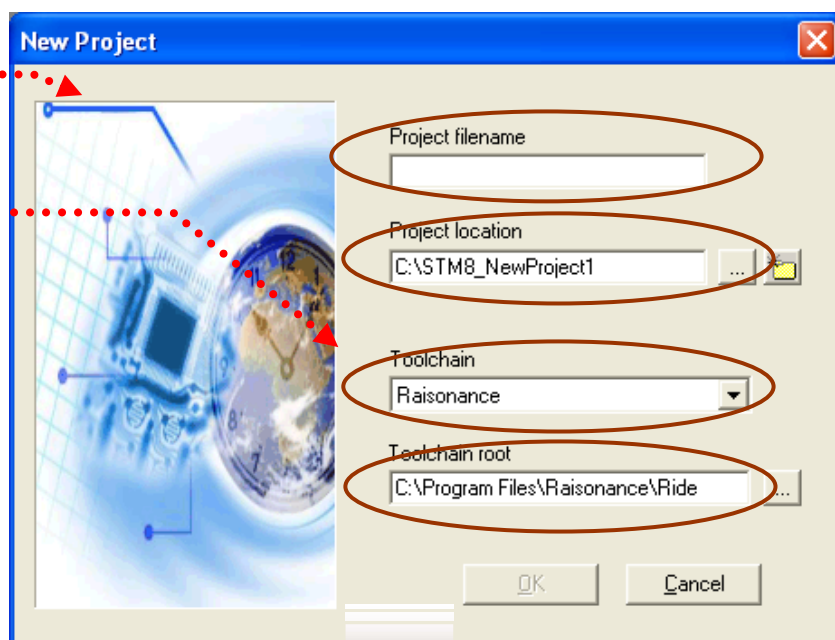
按照下面描述分别填入相关项：

Project filename: IO_Test。

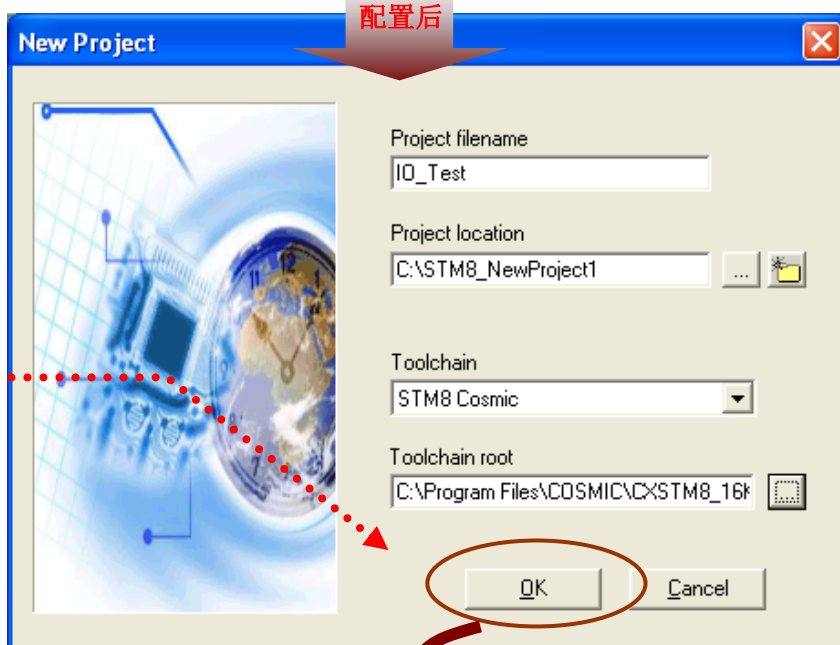
Project location: 无需改变，保留（默认与 Workspace 同路径）。

Toolchain : 默认编译器为 Raisonance，使用下拉菜单，将之选为 STM8 Cosmic。

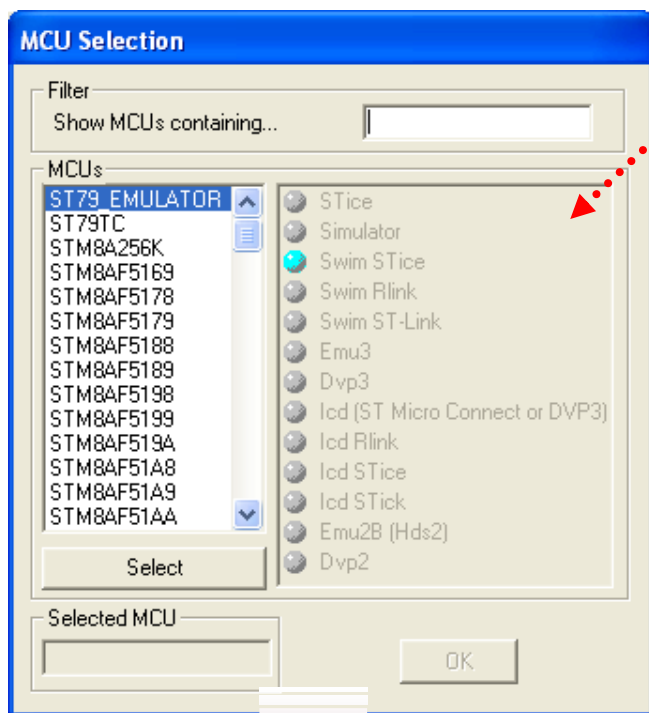
Toolchain root : 选择 STM8 COSMIC编译器的安装地址。



配置后

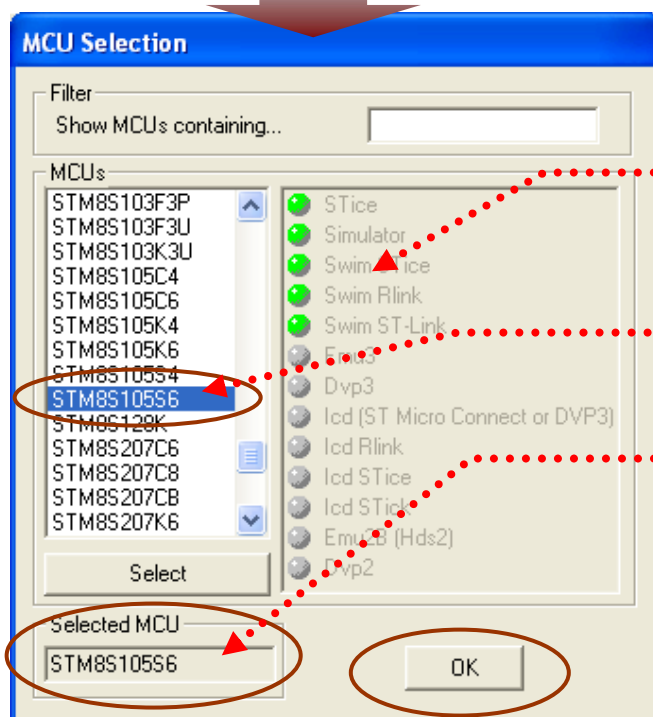


配置完成后的窗口如图所示，确认后进入MCU 选择窗口。



MCU 选择窗口

配置后



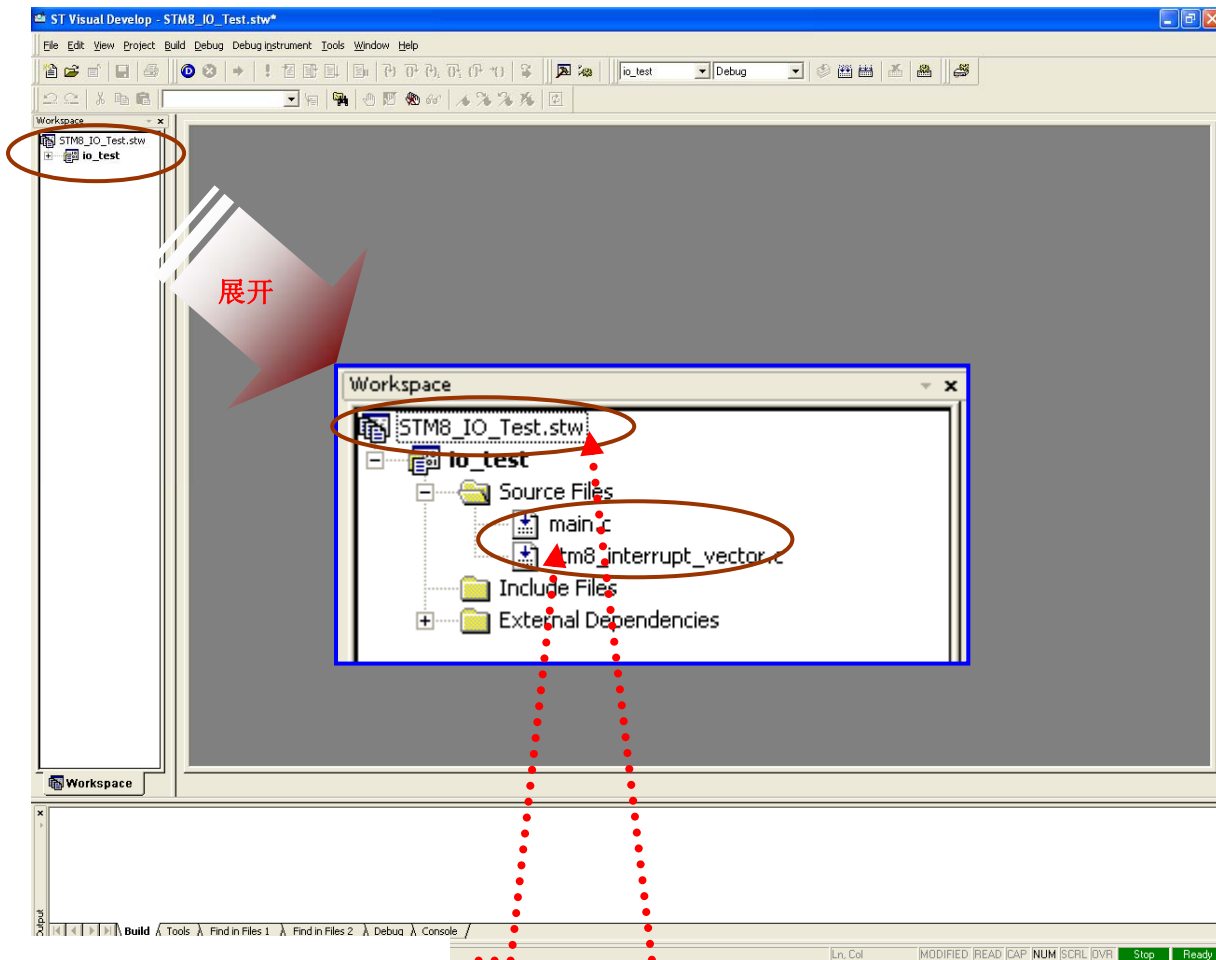
显示支持当前选中的MCU的所有工具。

选择目标MCU,并双击。

已选择的MCU会出现在此栏中

本例使用STM8 mini kit2做为目标板, 因此选中STM8S105S4并双击之, 使之出现在Selected MCU 栏中。

点击 OK 按钮, 系统完成基本设置并进入主界面, 如下图所示。

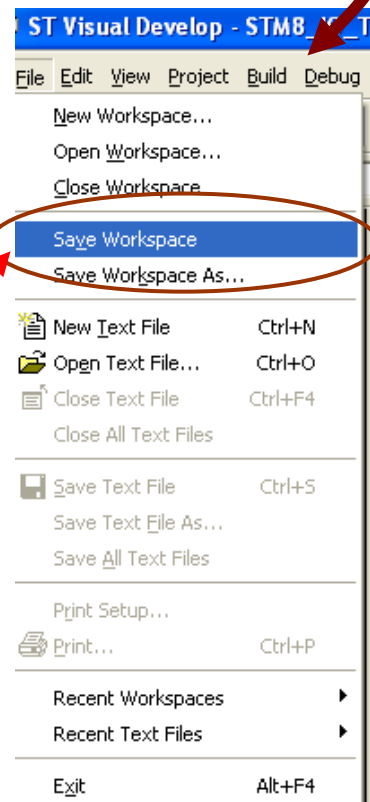


将屏幕左侧io_test项目栏展开，
可以看见系统已自动生成了两个C
文件：

main.c

stm8_interrupt_vector.c

同时在标题栏上出现了workspace的名
字:STM8_IO_Test.stw。细心的读者可
能注意到名字后面跟了一个*号，其表示
workspace出现了变化，需要保存。



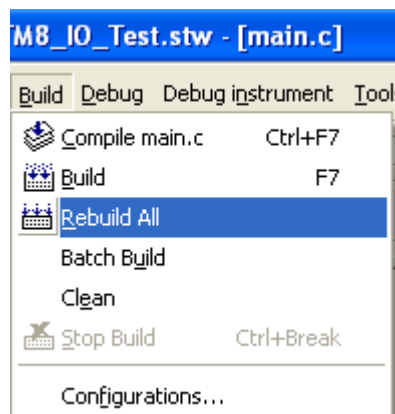
点击标题栏菜单File->Save workspace

保存完毕后，标题栏上项目名称后的*号消失。至此，一个最基本的STM8 工程项目已经建立完毕。但这个工程项目没有包含对所使用MCU的寄存器的描述，一个简单的解决方法是从STVD的安装目录中(默认是C:\Program Files\STMicroelectronics\st_toolset)将 \include 目录下实际应用所使用MCU的对应 .h文件复制到项目目录中。

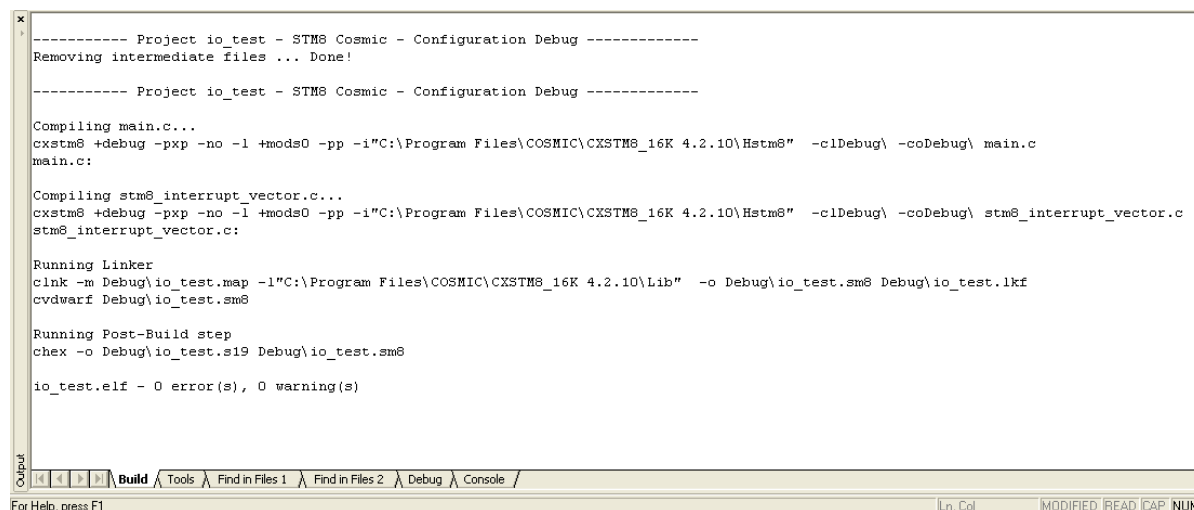
本例中使用STM8S105S4，因此用户需要复制STM8S105C_S.h到C:\STM8_NewProject1，同时在main.c中加入 #include" STM8S105C_S.h"。

注：STM8S105C_S.h 包含了对STM8S105C系列(即48脚封装系列)以及S系列(即44脚封装系列)寄存器地址的定义。

点击标题栏菜单Build->Rebuild All 进行编译及链接，如下图所示：

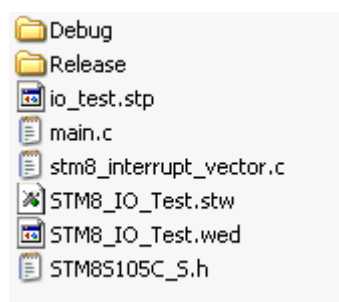


主界面output windows 会输出编译链接的结果如下：



至此，一个最基本的 STM8S105S 的工程项目已经建立完毕。

此时，C:\STM8_NewProject1 目录下的内容如下：



2.2.2 软件编写注意事项

在进行STM8软件编写时，请着重注意以下要点：

1. 时钟分配

主时钟是否正常起振并稳定，各个外设时钟是否开启

2. 选项字节配置(option bytes)

A. I/O重映射功能状态是否与实际项目相符合

B. 如果看门狗使用硬件方法使能，则看门狗在复位后立即有效，主程序必须喂狗。

C. 若果MCU主频高于16MHz，则需要配置选项字节的MCU等待周期为1

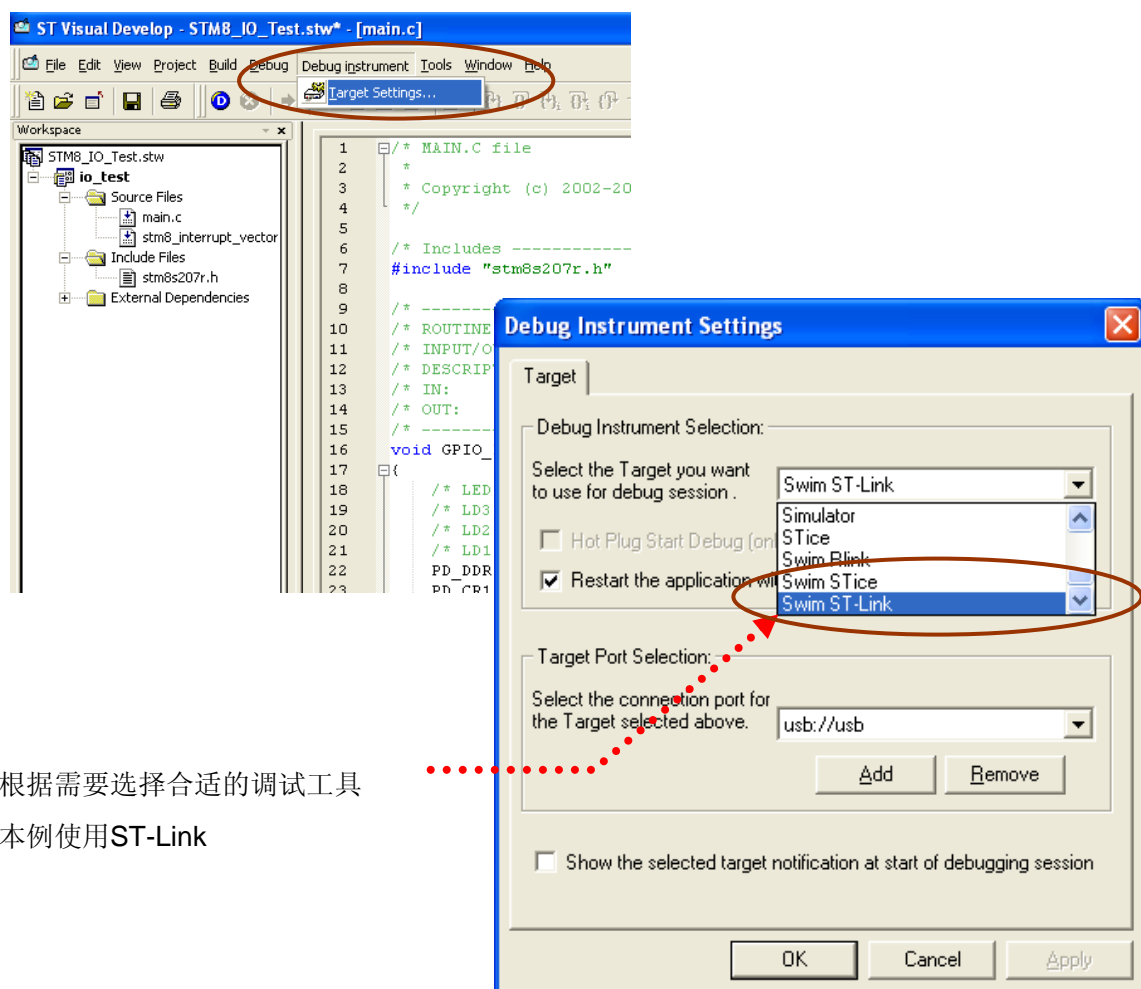
3. 有一些状态寄存器的位的清零是通过读该寄存器来实现的，所以对这样的寄存器操作要清楚其后果。

4. 建议将常用的变量分配在Zero page中，这样可以提高这些变量的访问速度。对于不常用的变量可以用@near定义在0xFF以外区域（相对来说，访问速度略慢）。用户可以根据实际情况决定。

2.2.3 在线调试

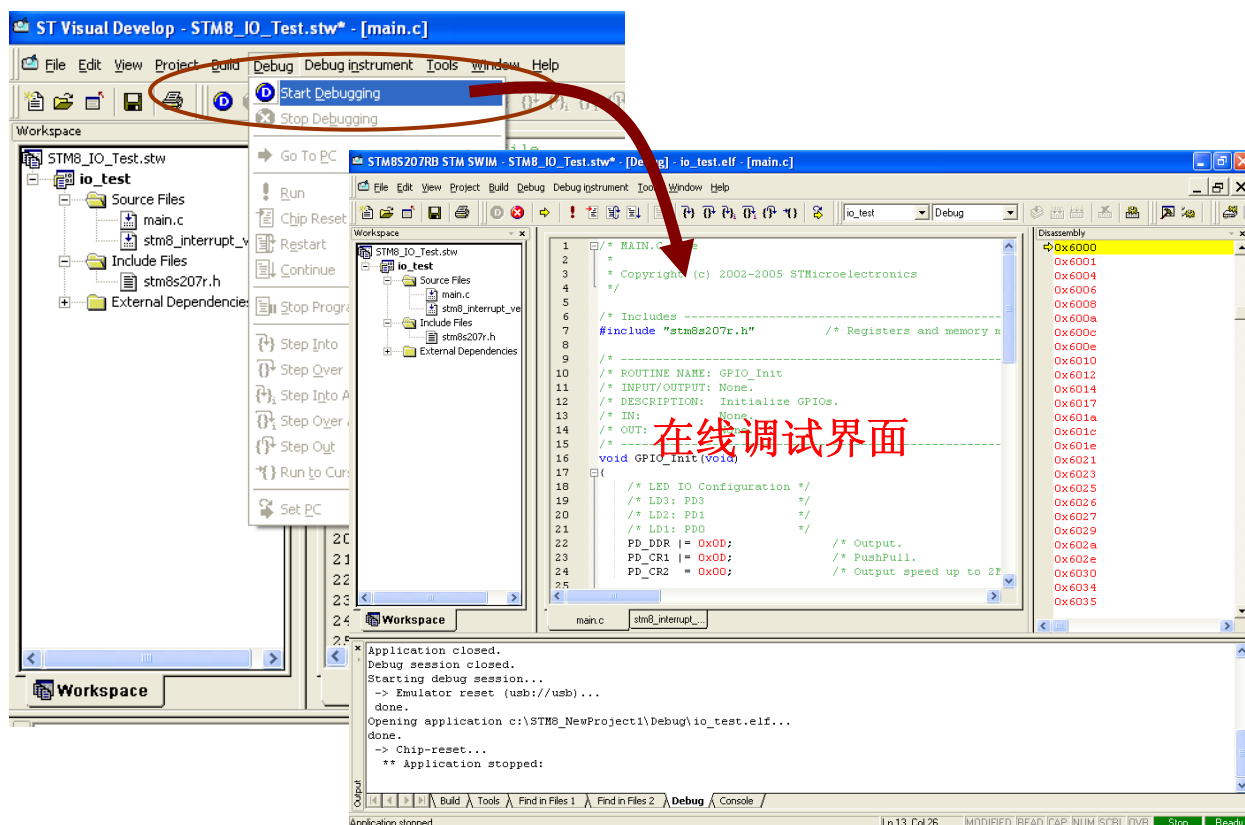
程序编译通过后，将工具和目标板连接好后，就可以准备开始在线调试了。在线调试可按如下步骤进行：

1) 通过 Debug instrument → Target Settings菜单，选择合适的在线调试工具，如下图：



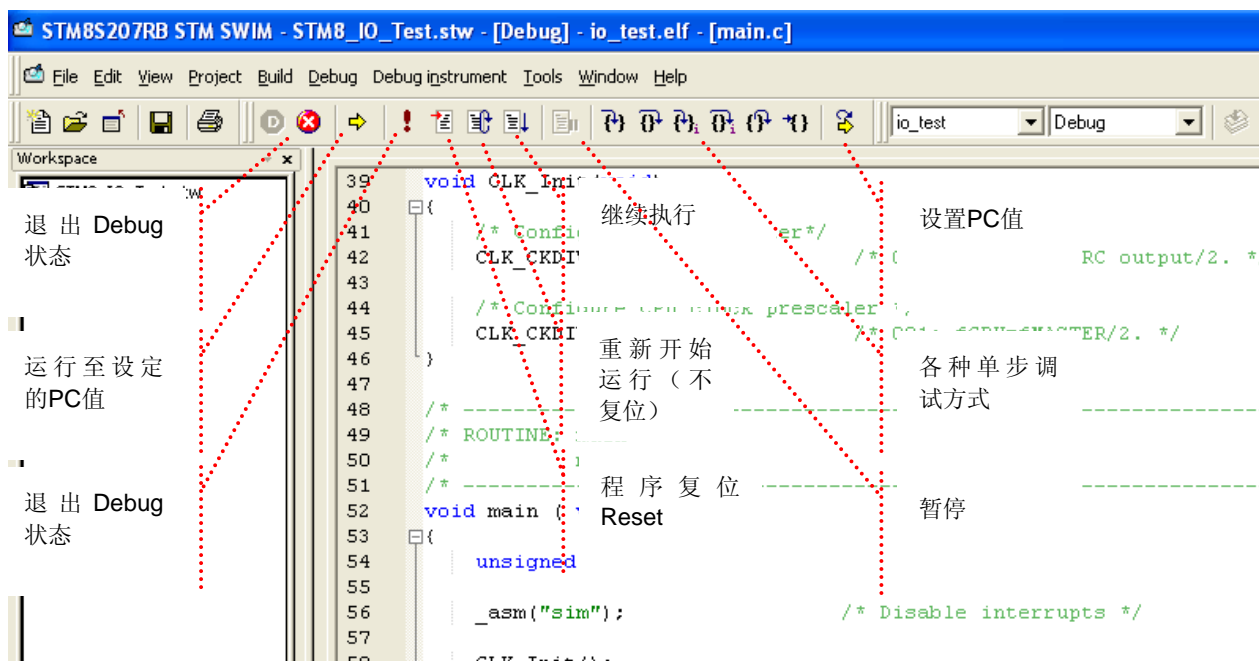
根据需求选择合适的调试工具
本例使用ST-Link

2) 通过菜单Debug→ Start Debugging 或者快捷键进入在线调试模式。

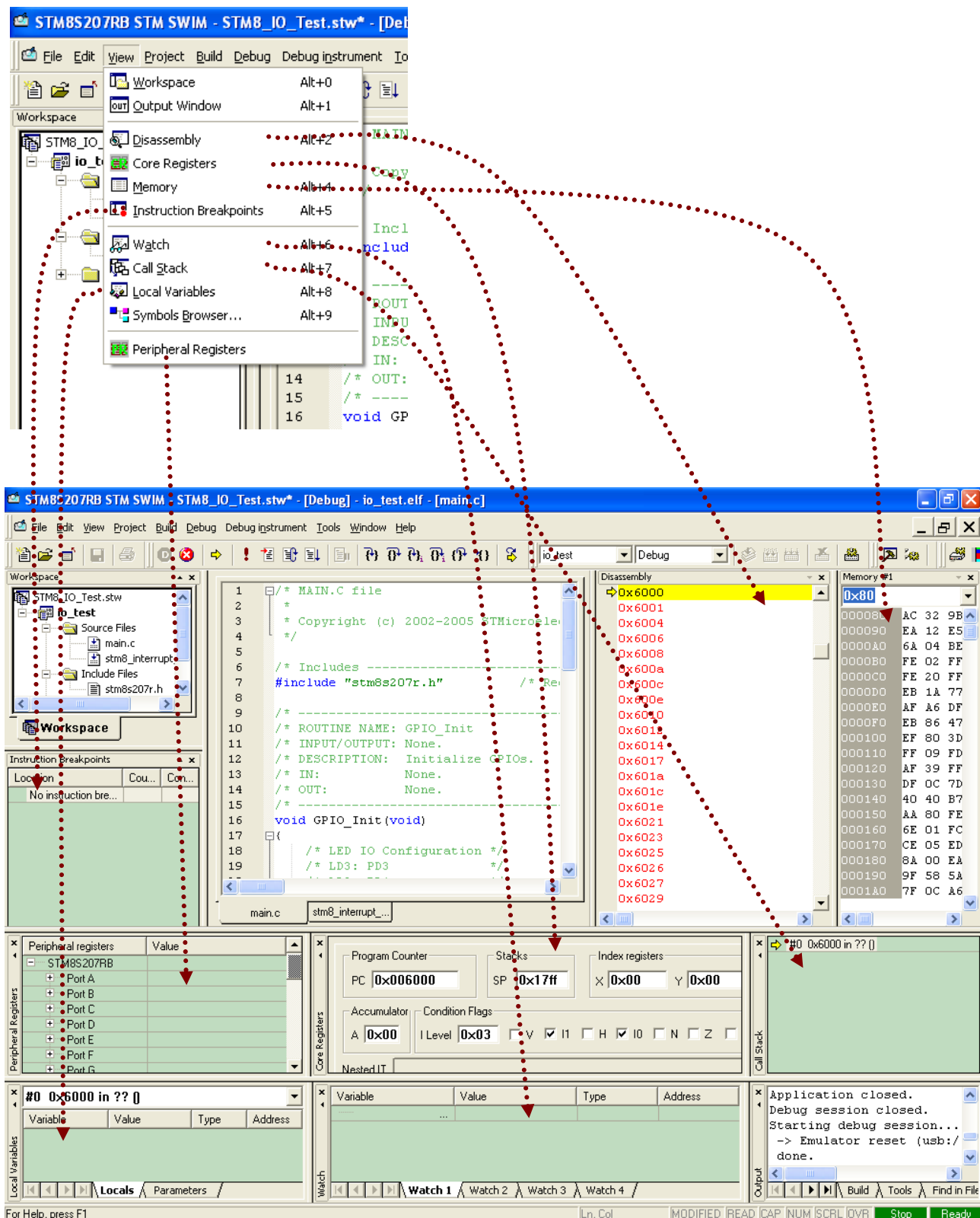


3) 程序运行。

进入在线调试界面后，可以通过快捷键或者菜单来运行或者复位程序，以及进行单步调试。下图为快捷键的简单说明：



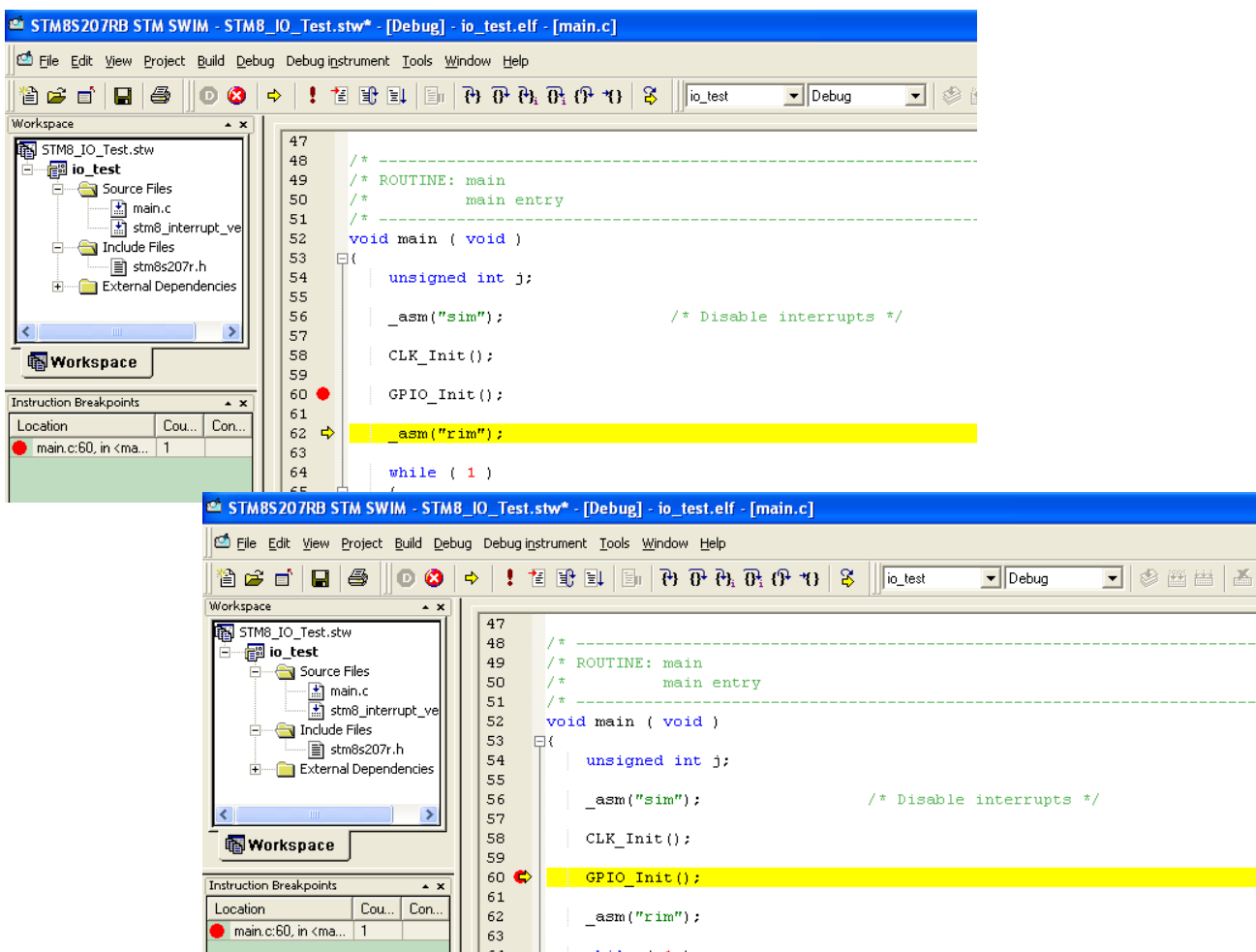
可以通过菜单 View 来打开各项观察窗，查看程序运行情况：



View 菜单下拉项	说明	应用
Disassembly	反汇编窗口	1) 查看C语言程序被编译后具体用什么指令来实现的 2) 查看是否有语句被优化掉
Core Registers	内核寄存器窗口	查看6个内核寄存器的状态
Memory	存储器窗口	查看整个存储器的状态，具有Read/Write on the fly功能，可以在运行时读取存储器状态，或者实时修改存储器值
Instruction Breakpoints	指令断点	管理所有设置的指令断点。
Watch	观察窗口	根据调试需要添加需要观察的变量或寄存器，用来观察值的变化。并且具有Read/Write on the fly功能，可以实时读取或者修改变量值。
Call Stack	堆栈窗口	观察堆栈的使用情况。
Local Variables	局部变量	观察局部变量的值
Peripherals registers	外设寄存器窗口	包含所有外设寄存器变量，方便观察。但由于访问部分寄存器可能会造成某些状态位的变化，因此外设寄存器窗口不具有Read/Write on the fly功能。

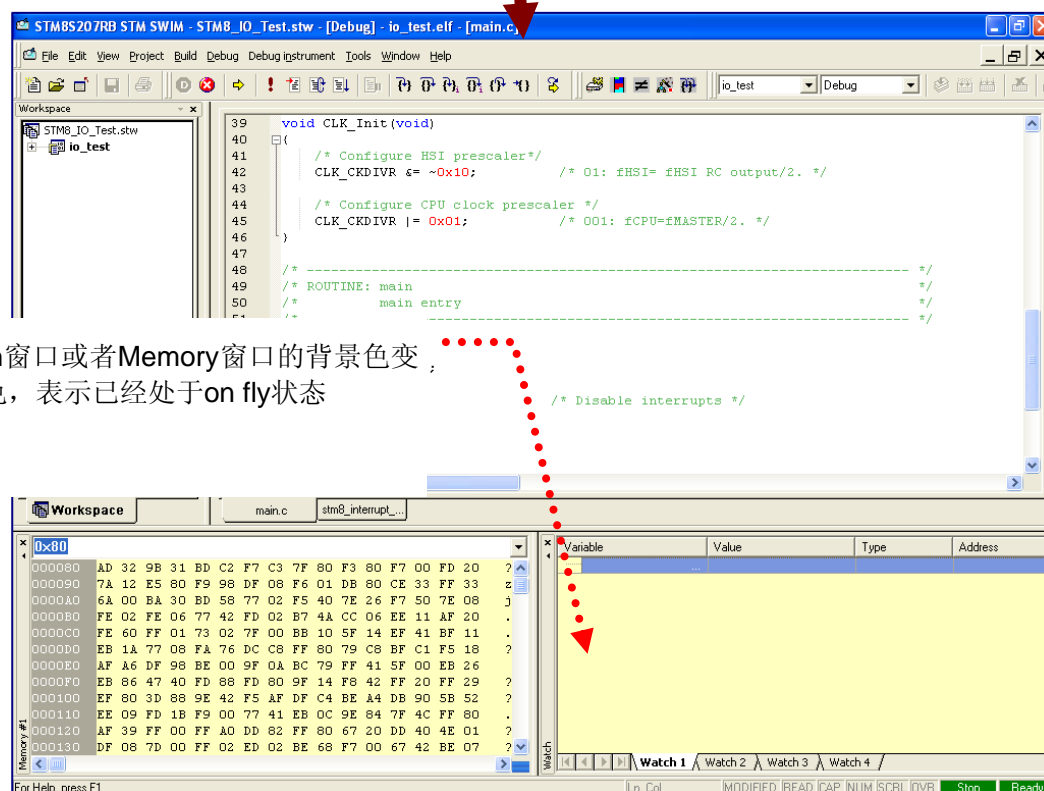
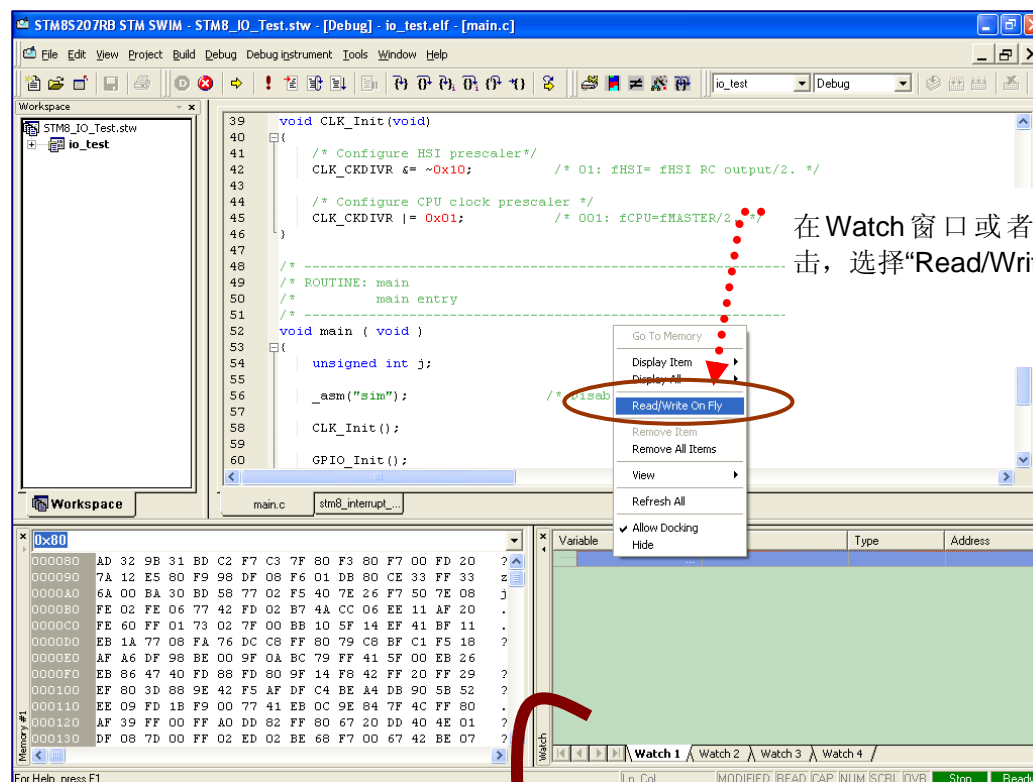
4) 设置断点

STM8软件断点无数量限制，但是不能将中断设在中断向量表内。



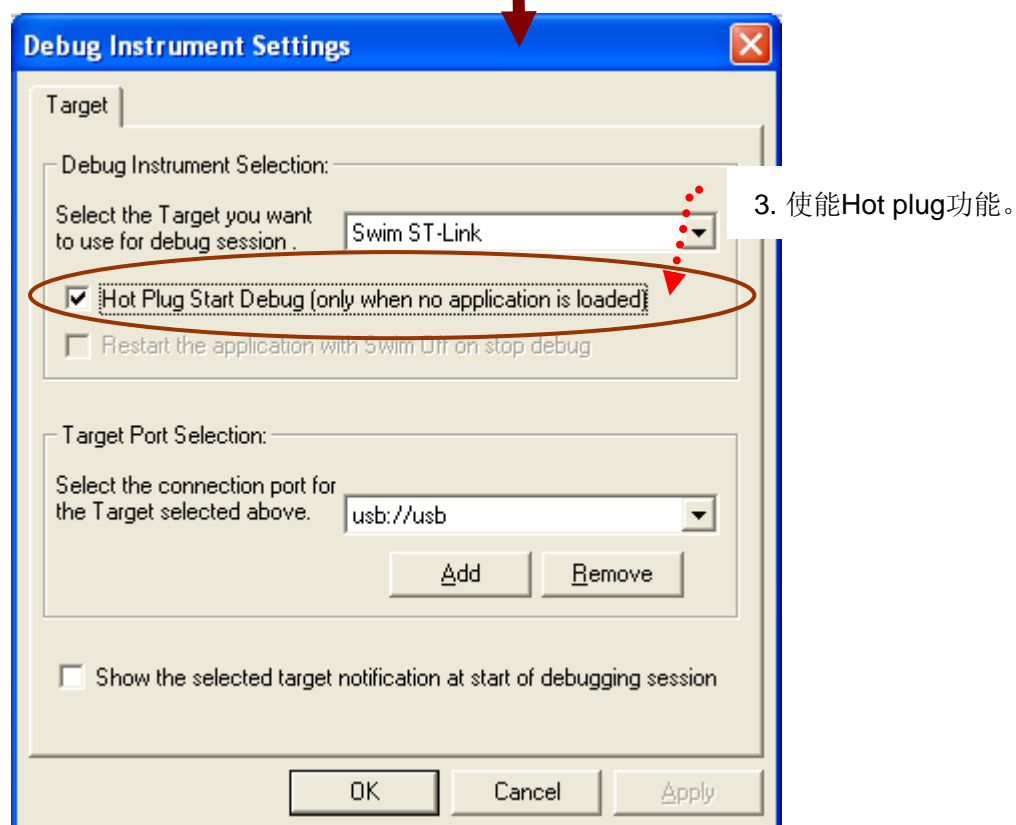
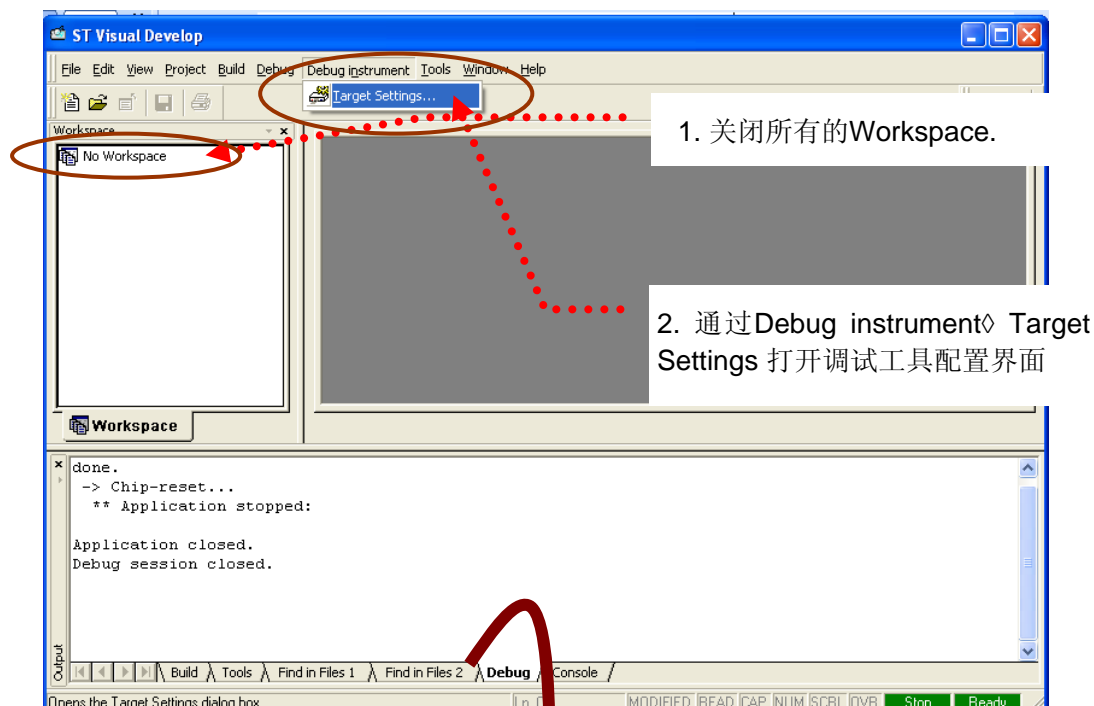
5) RD/WR on fly功能

STM8S在调试时支持RD/WR on fly功能，用户可以在程序运行时，直接观察变量的变化。也可以在不中断程序运行的条件下直接修改寄存器或者变量的值。



6) Hot plug功能

支持hot plug功能。当程序在运行时，可以通过SWIM接口在不影响程序连续运行的条件下，通过STVD窗口观测存储器内各个值的变化。（前提是不设读保护）。



4. 重新进入在线调试界面，即可看到存储器内的值随着程序的变量发生变化。

3 进一步掌握STVD/COSMIC

3.1 如何分配变量到指定的地址

举例：

```
unsigned char temp_A@0x00; //定义无符号变量temp_A，强制其地址为0x00
```

```
unsigned char temp_B@0x100; //定义无符号变量temp_B，强制其地址为0x100
```

```
@tiny unsigned char temp_C; //定义无符号变量temp_C，由编译器自动在地址小于0x100的RAM中为其分配一个地址
```

```
@near unsigned char temp_D; //定义无符号变量temp_D，由编译器自动在地址大于0xFF的RAM中为其分配一个地址
```

另外也可以采用伪指令“**pragma**”将函数或者变量定义到指定的**section**中，例如：

```
#pragma section [name] // 将下面定义的未初始化变量定义到.name section中
```

```
Unsigned char data1;
```

```
Unsigned int data2;
```

```
.....(任何需要定义在.name section中的变量)
```

```
.....
```

```
#pragma section [] // 返回到正常的section.
```

(关于**section**的定义方法可以参考3.7节的描述。)

注意：**pragma**伪指令可以用来定位函数，初始化变量或者未初始化变量。这三者用不同的括号区分。

(name)：代码

[name]：未初始化变量

{name}：初始化变量

3.2 如何在COSMIC C文件中使用汇编语言

在COSMIC C文件中使用汇编语言常见的方法有如下两种：使用**#asm ...#endasm**组合格式或**_asm(...)**；单行格式。

举例1：

```
unsigned char temp_A;
```

```
Void func1(void)
```

```
{
```

```
...
```

```
#asm
```

```
PUSH A
```

```
LD A,(X)
```

```
LD _temp_A,A
```

```
POP A
```

```

#endasm
...
}

```

注：在C嵌汇编环境下使用全局变量，要在该全局变量名称前加下划线“_”。

举例2:

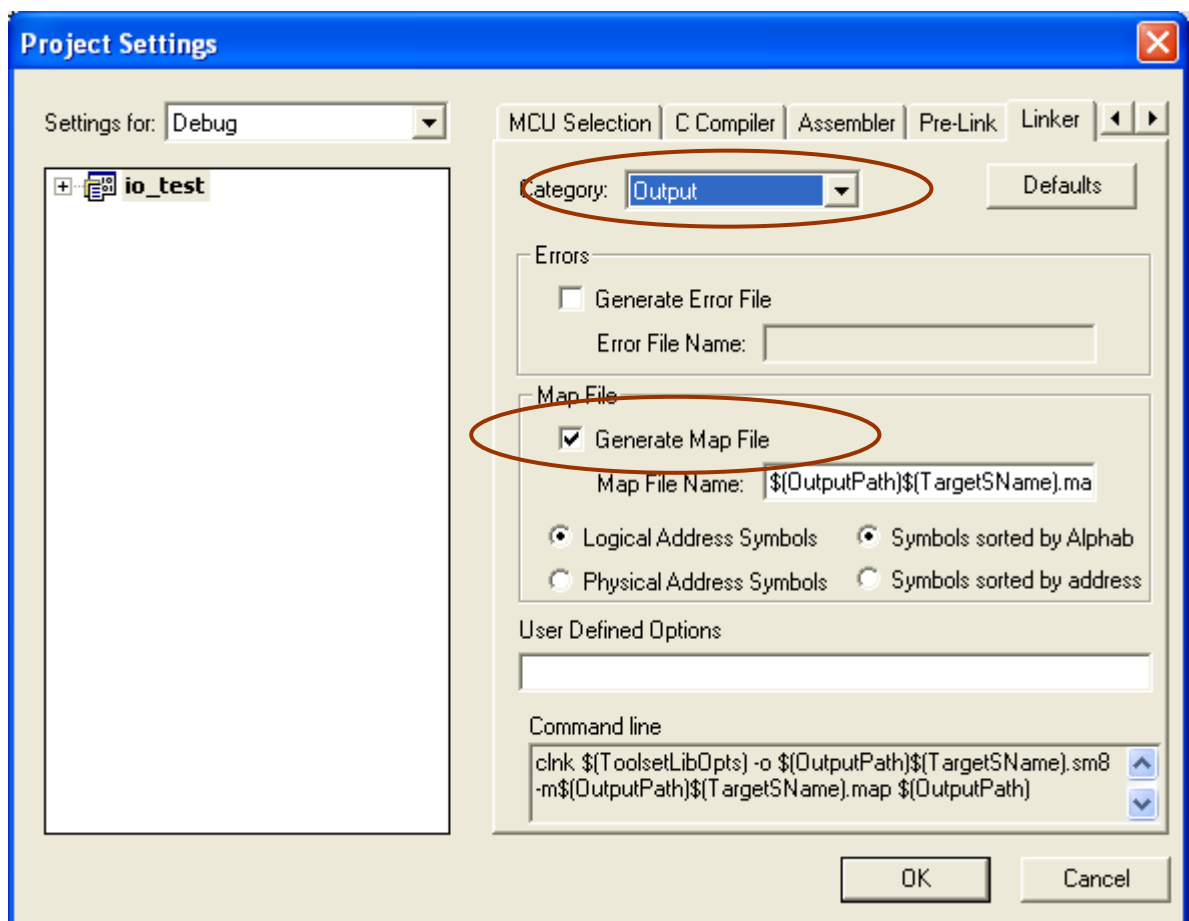
```

Void func1(void)
{
    ...
    _asm("rim");
    _asm("nop");
    ...
}

```

3.3 如何观察RAM/FLASH/EEPROM的最终分配情况

在Project->settings->linker选项页中，将Category选为Output，再勾选Generate Map File。



点击OK按钮后，再次编译链接该项目，如果成功则会在项目输出目录中(本例是在C:\STM8_NewProject1\debug 目录下)生成 .map 文件。该文件详细地列出RAM/FLASH/EEPROM的分配使用情况。

3.4 如何生成hex格式的输文件

在Project->settings->PostBuild选项页中，在commands栏内加入下行命令：

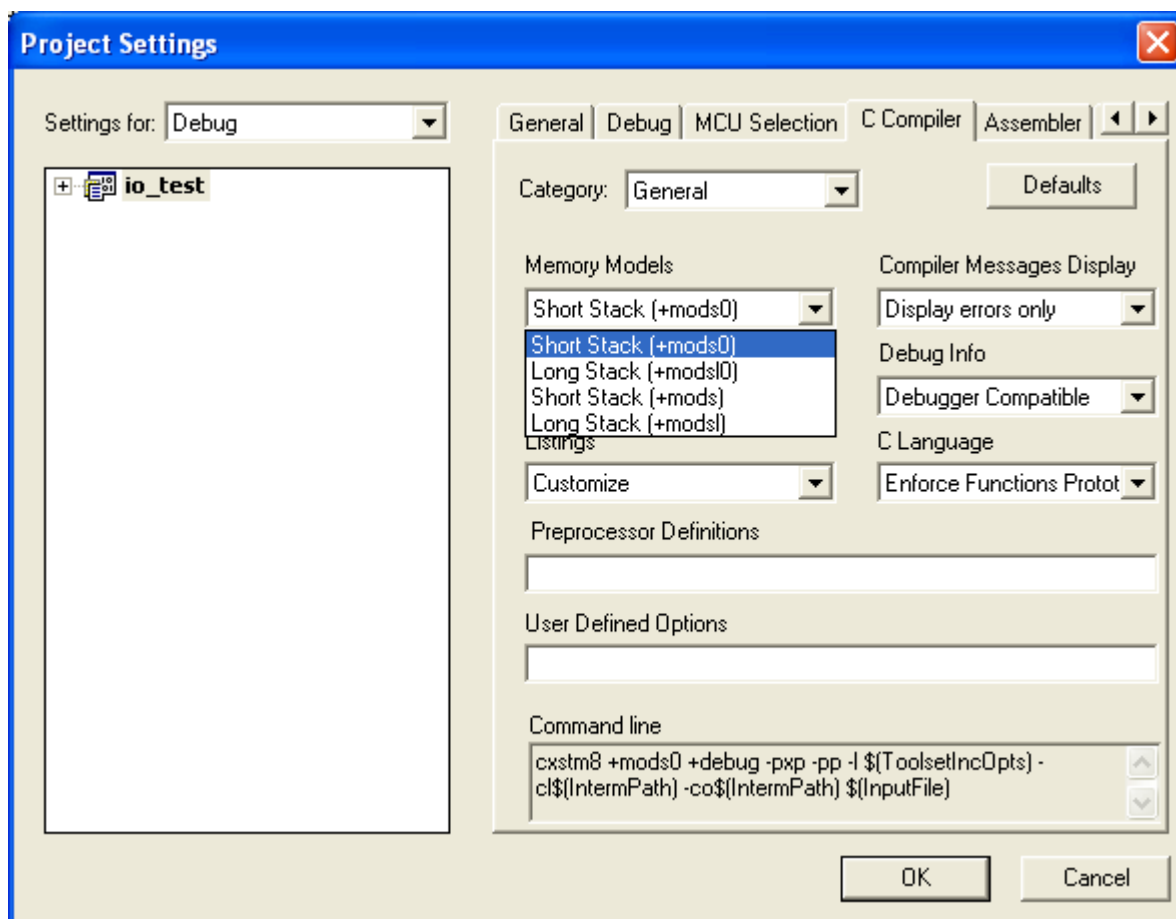
```
chex -fi -o $(OutputPath)$(TargetSName).hex $(OutputPath)$(TargetSName).sm8
```

再次编译链接该项目，如果成功则会在项目输出目录中(本例是在C:\STM8_NewProject1\debug目录下)生成 .hex 文件。

3.5 什么是MEMORY MODEL

STM8的C编译器支持多种存储器模式。用户可以根据应用的需要选择最适合的配置。可以根据需要选择采用2个字节的寻址方式（仅适用于64k以内的程序）或者3字节的寻址方式。也可以规定将变量默认为定义在存储器的哪一区域：zero page内，还是zero page 外。下面对几种供选择的MEMORY MODEL做简单说明。

在 Project->settings->C Compiler选项页中，将Category选为General，里面有一个Memory Models选项栏如下：



在下拉菜单中共有4种MEMORY MODEL可供选择：

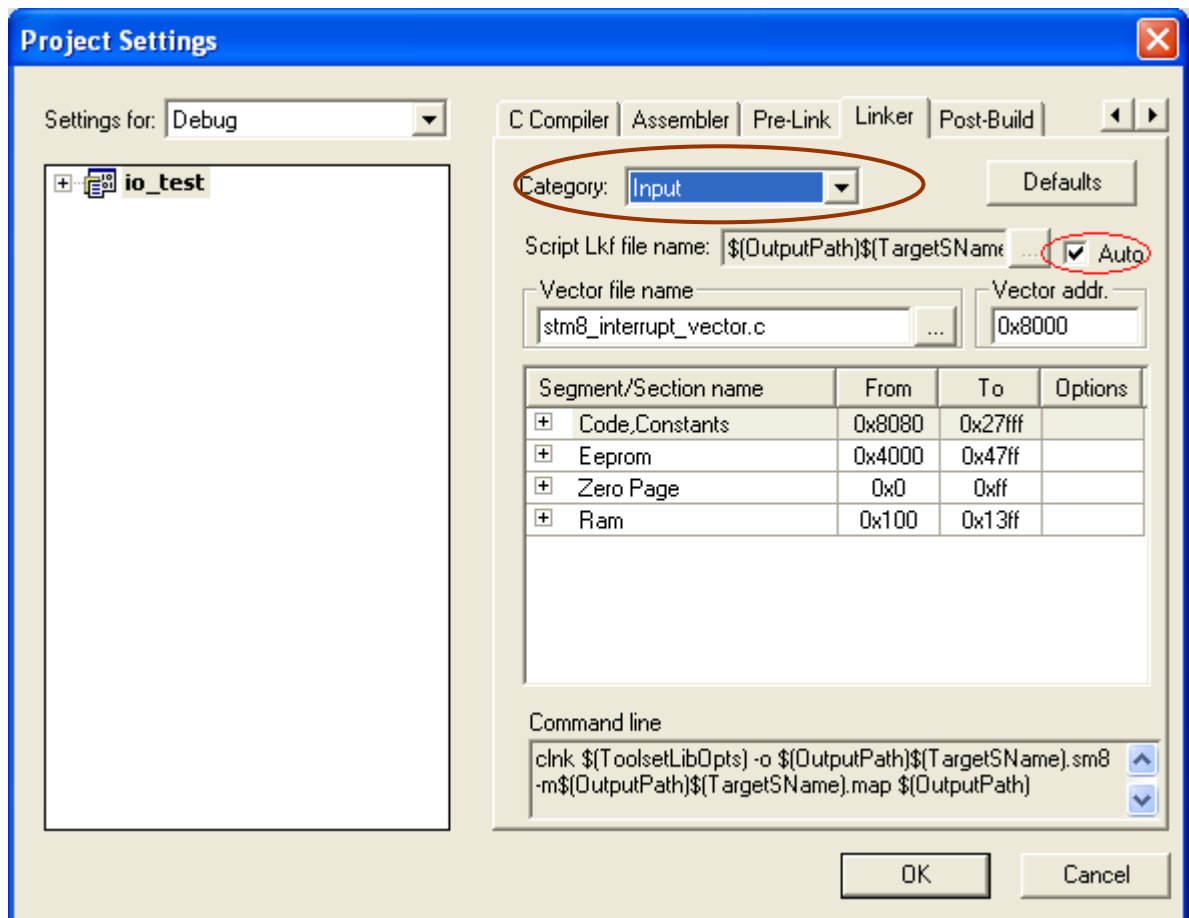
- 程序地址空间在64K以内(即程序容量小于32K)
 - ◆ mods0,
 - ◆ modsl0,
- 程序地址哦那个键在64K以上（即程序容量大于32K）

- ◆ mods,
- ◆ modsl

	MODS0	MODSL0	MODS	MODSL
名称	Stack Short 短堆栈模式	Stack Long 长堆栈模式	Stack Short 短堆栈模式	Stack Long 长堆栈模式
程序地址空间	程序所用到的地址空间在 64K 范围内		程序所用到的地址空间超出 64K 范围	
指针默认类型	函数指针和数据指针默认为@near (2 bytes)		函数指针默认为@far(地址为 3 字节); 数据指针默认为@near	
全局变量默认类型	所有全局变量的地址默认为 1 个字节。对于地址超出 1 个字节的变量, 必须用@near 定义	所有全局变量默认为 Long 型。若要将变量地址定义为 1 个字节, 必须用@tiny 定义	所有全局变量的地址默认为 1 个字节。对于地址超出 1 个字节的变量, 必须用@near 定义	所有全局变量默认为 Long 型。若要将变量地址定义为 1 个字节, 必须用@tiny 定义

3.6 .lcf 文件的作用

.lcf 文件在程序链接时决定如何具体分配RAM/ROM的空间。在Project Settings – Linker – Category(Input)选项页中, 当“Auto”选择框被勾选时, 由系统自动生成.LCF文件, 否则由用户指定。



当“Auto”选择框被勾选时，.lcf文件会自动生成在项目主目录下的 debug/ 和 release/ 目录中。下面以上图所示 io_test Project的 lcf 文件为例，来进一步理解.lcf。

在.lcf中，以“#”开头的行是注释行，为方便用户理解，将原注释删除，代之以中文注释如下：

定义(+seg)一个常量段(.const)，开始(b)于0x8080，最大分配(m)0x1ff80个字节(即不超过

0x27FFF)，为该段起名(n)为.const(和常量段的保留字同名)，需要初始化的变量的初始值存

放于此段(-it)

+seg .const -b 0x8080 -m 0x1ff80 -n .const -it

定义(+seg)一个程序段(.text)，紧跟(-a)在.const段后面(和.const 共同位于0x8080 –

0x27FFF)，为该段起名(n)为. text (和程序段的保留字同名)。

+seg .text -a .const -n .text

定义(+seg)一个EEPROM段(.eeprom)，开始(b)于0x4000，最大分配(m)0x800个字节(即不超

过0x47FF)，为该段起名(n)为. eeprom (和EEPROM段的保留字同名)。

+seg .eeprom -b 0x4000 -m 0x800 -n .eeprom

.bsct段服务于定义在0页(地址小于0x100)以内需要初始化的全局变量(如 @tiny char a = 9;)

+seg .bsct -b 0x0 -m 0x100 -n .bsct

.ubsct段服务于定义在0页(地址小于0x100)以内不需要初始化的全局变量(如 @tiny char b;)

+seg .ubsct -a .bsct -n .ubsct

.bit表示位域段，定义后即可在程序中使用_Bool变量(如 _Bool c = 1;)，-id表示该段需要初始化。

+seg .bit -a .ubsct -n .bit -id

这是ST7时代(STM8是基于ST7发展而来的)由于物理堆栈小，速度慢，使用内存来模拟堆栈的变通手段。

+seg .share -a .bit -n .share -is

.data段服务于定义在0页(地址大于0xFF)以外需要初始化的全局变量(如 @near char d = 8;)

+seg .data -b 0x100 -m 0x1300 -n .data

.bss段服务于定义在0页(地址大于0xFF)以内不需要初始化的全局变量(如 @ near char e;)

+seg .bss -a .data -n .bss

段定义结束，下面放置的库及Obj文件中的变量、常量、程序就按照上面的规定进行分配。

#初始化程序

```

crti0.sm8
#用户程序
Debug\main.o
...

# 一些必要的cosmic库
libi0.sm8
libm0.sm8

# 重定义常量段，开始于0x8000，用于放置中断向量表(STM8硬件决定此位置)
# -k 用于程序冗余代码优化，详情可参考cosmic用户手册。
+seg .const -b 0x8000 -k
# 中断向量
Debug\stm8_interrupt_vector.o

#定义了三个变量，用于系统初始化
+def __endzp=@.ubsct                # end of uninitialized zpage
+def __memory=@.bss                 # end of bss segment
+def __stack=0x17ff                 # 不同的芯片__stack内容不同，由系统自动生成

```

3.7 如何实现位操作

Cosmic C 编译器支持位变量的操作，可以将其定义成 `_Bool` 类型。`_Bool` 类型的变量只包含两种值 `true` (1) 或者 `false` (0)。若将一个表达式赋值给 `_Bool` 变量，则编译器会将表达式与 0 做比较，然后将布尔值赋给 `_Bool` 变量。因此，任何整型或者表达式的值都可以赋给 `_Bool` 变量。但是，布尔变量不能定义位数组，只能定义成结构体或者联合。而且，`_Bool` 变量会被打包成字节的形式。

编译器会将所有的全局 `_Bool` 变量打包成字节形式，存放在 `.bit section` 中。局部 `_Bool` 变量也会被打包成字节形式。但是 `_Bool` 类型的参数会被扩展成一个单字节。

具体的关于位变量的定义和使用可参考如下例子：

定义位变量：

```
_Bool in_range;
```

```
_Bool p_valid;
```

```
char *ptr;
```

使用位变量：

```
in_range = (value >= 10) && (value <= 20);
```

```
p_valid = ptr; /* p_valid is true if ptr not 0 */
```

```
if (p_valid && in_range)
```

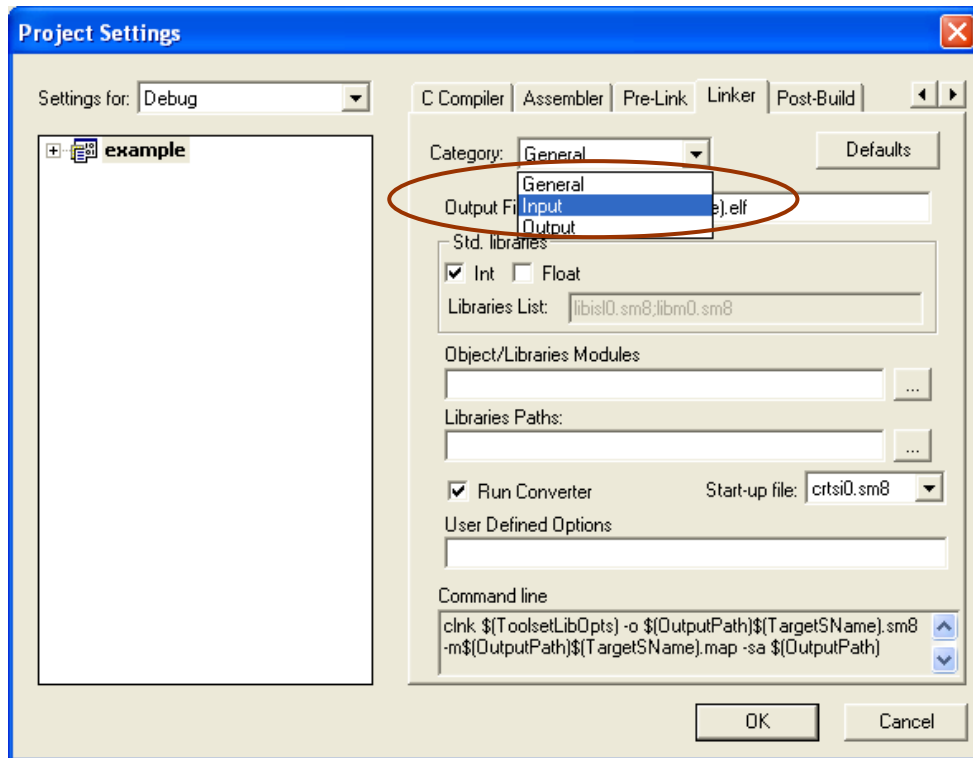
```
    *ptr = value;
```

在使用位变量时，若程序编译时提示如下错误：

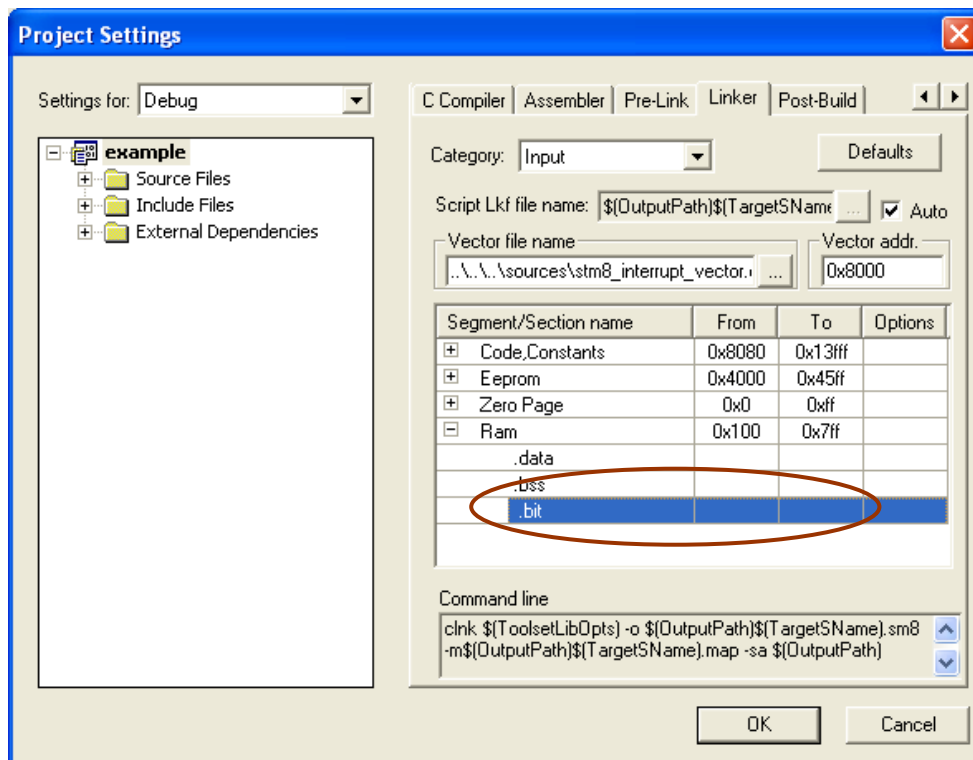
```
#error clnk Debug\example.lkf:1 no default placement for segment .bit
The command: "clnk -l"C:\Program Files\COSMIC\CXSTM8_16K_4.2.10\Lib"
-o Debug\example.sm8 -mDebug\example.map -sa Debug\example.lkf " has
failed, the returned value is: 1
exit code=1.
```

实际上是由于，在项目中没有定义.bit section。可按照如下步骤，手工添加.bit section：

打开项目链接配置窗口：Project → Settings → Linker，选择 Input 目录项



在Zero page 或者 Ram 里面定义一个.bit section.



然后重新编译一下就可以了。

3.8 __stext是什么以及初始化程序库的意义

初始化程序库包含一些重要的函数，以便为C语言的运行创建一个运行环境。初始化启动文件包含如下的标准模块：

初始化bss section (如果有用到的话)

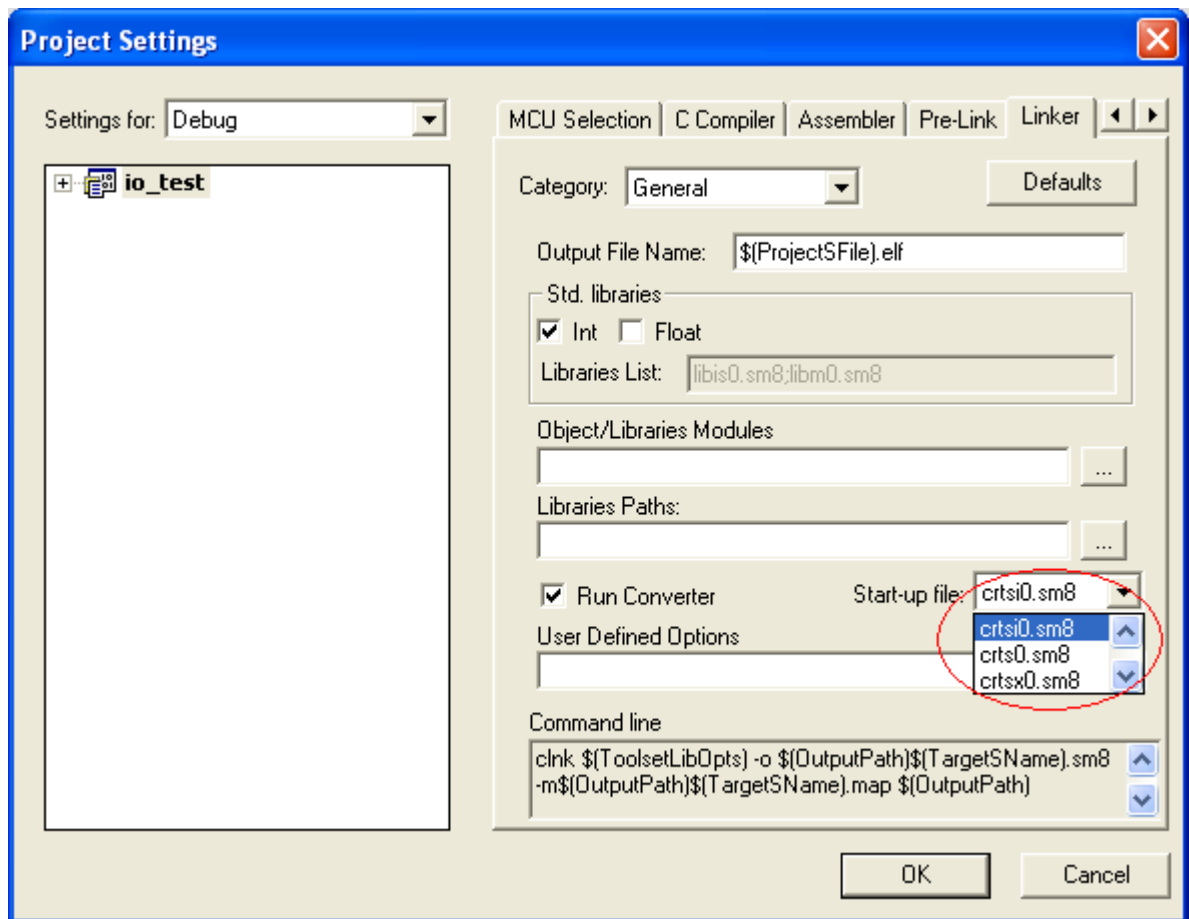
若有需要，将ROM复制到RAM

堆栈指针初始化

f_main 或者其他程序入口

从C环境返回的退出序列。大部分的用户必须修改所提供的推出序列以满足执行环境的需要。

要使用__stext，需要在程序链接时使用指定的startup file，如下图所示：



为了使用startup file，中断向量表中RESET入口地址也应该为__stext，如下图所示：

```

struct interrupt_vector const_vectab[] = {
    {0x82, (interrupt_handler_t) __stext}, /* reset */
    {0x82, NonHandledInterrupt}, /* trap */
    {0x82, TLI_Interrupt}, /* irq0 */
    {0x82, NonHandledInterrupt}, /* irq1 */
    {0x82, NonHandledInterrupt}, /* irq2 */
    {0x82, NonHandledInterrupt}, /* irq3 */
    {0x82, NonHandledInterrupt}, /* irq4 */
    {0x82, NonHandledInterrupt}, /* irq5 */
    {0x82, NonHandledInterrupt}, /* irq6 */
    {0x82, NonHandledInterrupt}, /* irq7 */
    {0x82, NonHandledInterrupt}, /* irq8 */
}

```

这样MCU复位后，如果没有使能bootloader，程序就会从__stext开始执行。以crt0.sm8的源程序为例，看一看__stext初始化中的操作。

```

xref f_main, __stack
xdef f_exit, __stext, f__stext
switch .text
__stext:
f__stext:
ldw x, #__stack ; 堆栈初始化
ldw sp, x
callf f_main ; 执行主程序
f_exit:

```

```
jra f_exit    ; 如果主程序退出，则在此处死循环  
end
```

请注意crt.s8是一个最基本的初始化代码，不包括对变量的初始化操作，如果想对变量进行初始化操作，有两个常见的办法：

1. 在变量定义时初始化(如 `int my_var = 0x1000;`)，然后选择合适的初始化程序(crt.s8/crtx.s8/ crt0.s8/ crt0x.s8...具体如何选择请参考COSMIC C用户手册，在此不详细展开)。
2. 当变量定义时不做初始化，在主程序中进行相关操作。(推荐)

```
int my_var;  
Main()  
{  
    Init();  
    ....  
}  
void Init(void)  
{  
    my_var = 0x1000;  
}
```