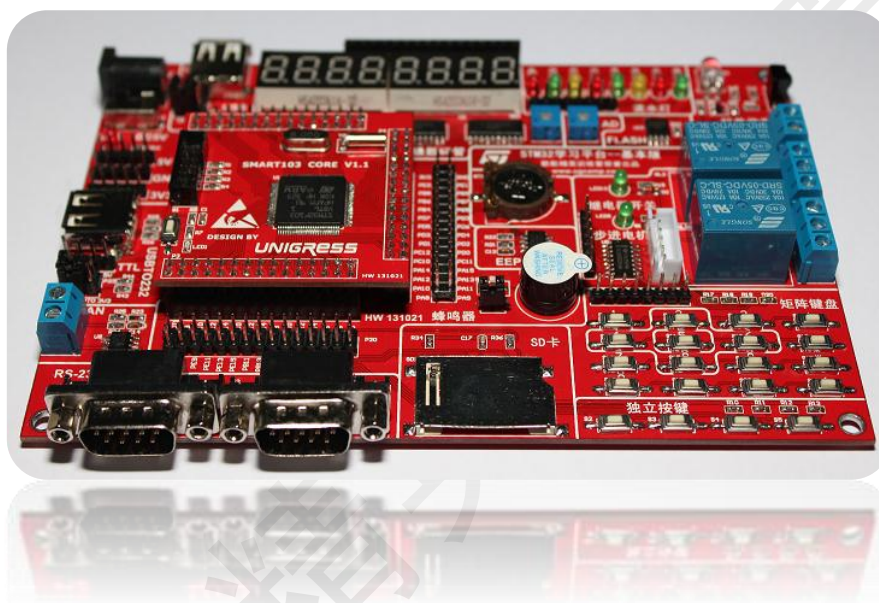


SMART103 平台实验手册手册



联航精英训练营课程研究中心

文档信息

文档名称: 联航训练营实验手册

版本号: 1.0

版本日期: 2014-01-15

质量审查阶段: N/A

编写者: Allen Zhang

准备日期:

审查者:

审查日期:

版权信息

本手册及其所含信息由联航精英训练营拥有，未经书面授权，不得将材料泄露给第三方。

SMART103 空工程建立实验

1.1 RealView MDK 软件开发环境简介

RealView MDK 全称 RealView MDK 中国版开发套件，源自德国 Keil 公司，被全球超过 10 万的嵌入式开发工程师验证和使用，是 ARM 公司目前最新推出的对各种嵌入式处理器的软件开发工具。RealView MDK 集成了业内最领先的技术，包括 μ Vision4 集成开发环境与 RealView 编译器，支持 ARM7、ARM9 和最新的 Cortex-M3 核处理器，自动配置启动代码，集成 Flash 烧写模块，强大的 Simulation 设备模拟，性能分析等功能，与 ARM 之前的工具包 ADS 等相比，RealView 编译器的最新版本可将性能改善超过 20%。

1.2 RealView MDK 的突出特性

- | | |
|------------|---------------------------------------|
| 菜鸟的阿拉伯飞毯 | — 启动代码生成向导，自动引导，一日千里 |
| 高手的无剑胜有剑 | — 软件模拟器，完全脱离硬件的软件开发过程 |
| 专家的哈雷望远镜 | — 性能分析器，看得更远、看得更细、看得更清 |
| 未来战士的激光剑 | — Cortex-M3 支持 |
| 业界最优秀的编译器 | — RealView 编译器，代码更小，性能更高配备 ULINK2 仿真器 |
| | — 无需安装驱动 |
| Flash 编程模块 | — 轻松实现 Flash 烧写 |
| 绝对的高性价比 | — 国际品质，本土价格 |

1.3 STM32F10x 固件库 v3.5 简介

1.3.1 STM32F10x 标准外设库 V3.5 简介

STM32F10x 标准外设库是一个固件函数包，它由程序、数据结构和宏组成，包括了微控制器所有外设的性能特征。该函数库还包括每一个外设的驱动描述和应用实例。通过使用本固件函数库，无需深入掌握细节，用户也可以轻松应用每一个外设。因此，使用本固件函数库可以大大减少用户的程序编写时间，进而降低开发成本。

每个外设驱动都由一组函数组成，这组函数覆盖了该外设所有功能。每个器件的开发都由一个通用 API(Application Programming Interface 应用编程界面)驱动，API 对该驱动程序的结构，函数和参数名称都进行了标准化。

所有的驱动源代码都符合“Strict ANSI-C”标准（项目于范例文件符合扩充 ANSI-C 标准）。ST 公司已经把驱动源代码文档化，他们同时兼容 MISRA-C2004 标准。由于整个固态函数库按照“Strict ANSI-C”标准编写，它不受不同开发环境的影响。仅对话启动文件取决于开发环境。

该固件库通过校验所有库函数的输入值来实现实时错误检测。该动态校验提高了软件的鲁棒性。实时检测适合于用户应用程序的开发和调试。但这会增加成本，可以在最终应用程序代码中移去，以优化代码大小和执行速度。

因为该固件库是通用的，并且包括了所有外设的功能，所以应用程序代码的大小和执行速度可能不是最优的。对大多数应用程序来说，用户可以直接使用之，对于那些在代码大小和执行速度方面有严格要求的应用程序，该固件库驱动程序可以作为如何设置外设的一份参考资料，根据实际需求对其进行调整。

1.3.2 CMSIS 架构简介

ARM 公司于 2008 年 11 月 12 日发布了 ARM Cortex 微控制器软件接口标准(CMSIS: Cortex Microcontroller Software Interface Standard)。CMSIS 是独立于供应商的 Cortex-M 处理器系列硬件抽象层，为芯片厂商和中间件供应商提供了连续的、简单的处理器软件接口，简化了软件复用，降低了 Cortex-M3 上操作系统的移植难度，并缩短了新入门的微控制器开发者的学习时间和新产品的上市时间。

根据近期的调查研究，软件开发已经被嵌入式行业公认为最主要的开发成本。图 1 为近年来软件开发与硬件开发成本对比图。因此，ARM 与 Atmel、IAR、Keil、hami-nary Micro、Micrium、NXP、SEGGER 和 ST 等诸多芯片和软件厂商合作，将所有 Cortex 芯片厂商产品的软件接口标准化，制定了 CMSIS 标准。此举意在降低软件开发成本，尤其针对新设备项目开发，或者将已有软件移植到其他芯片厂商提供的基于 Cortex 处理器的平台的情况。有了该标准，芯片厂商就能够将他们的资源专注于产品外设特性的差异化，并且消除对微控制器进行编程时需要维持的不同的、互相不兼容的标准的需求，从而达到降低开发成本的目的。

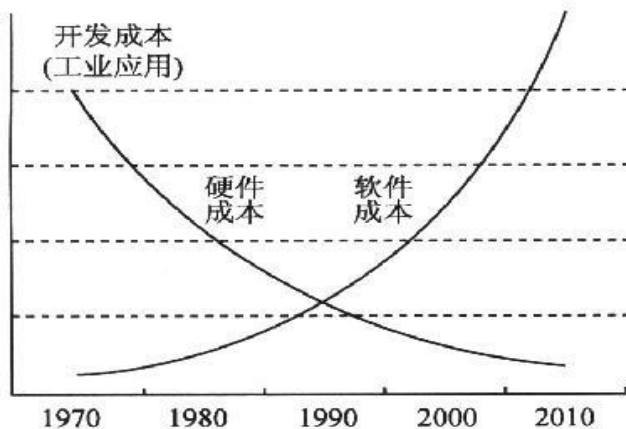


图 1 软件与硬件开发成本对比

如图 2 所示,基于 CMSIS 标准的软件架构主要分为以下 4 层:用户应用层、操作系统及中间件接口层、CMSIS 层、硬件寄存器层。其中 CMSIS 层起着承上启下的作用:一方面该层对硬件寄存器层进行统一实现,屏蔽了不同厂商对 Cortex-M 系列微处理器核内外设寄存器的不同定义;另一方面又向上层的操作系统及中间件接口层和应用层提供接口,简化了应用程序开发难度,使开发人员能够在完全透明的情况下进行应用程序开发。也正是如此, CMSIS 层的实现相对复杂。

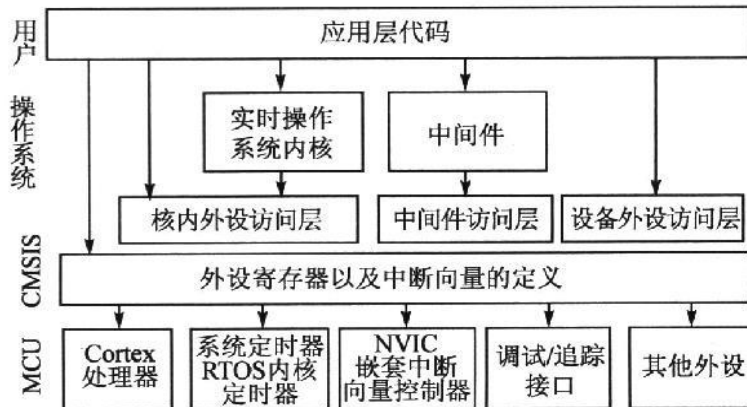


图 2 基于 CMSIS 标准的软件架构

CMSIS 层主要分为 3 部分。

1、核内外设访问层 CPAL(Core Peripheral Access Layer) 由 ARM 负责实现。包括对寄存器地址的定义,对核寄存器、NVIC、调试子系统的访问接口定义以及对特殊用途寄存器的访问接口(如 CONTROL 和 xPSR)定义。由于对特殊寄存器的访问以内联方式定义,所以 ARM 针对不同的编译器统一用 `_INLINE` 来屏蔽差异。该层定义的接口函数均是可重入的。

2、中间件访问层 MWAL(Middleware Access Layer) 由 ARM 负责实现,

但芯片厂商需要针对所生产的设备特性对该层进行更新。该层主要负责定义一些中间件访问的 API 函数，例如为 TCP / IP 协议栈、SD / MMC、USB 协议以及实时操作系统的访问与调试提供标准软件接口。该层在 1.1 标准中尚未实现。

3、设备外设访问层 DPAL(Device Peripheral Access Layer) 由芯片厂商负责实现。该层的实现与 CPAL 类似，负责对硬件寄存器地址以及外设访问接口进行定义。该层可调用 CPAL 层提供的接口函数，同时根据设备特性对异常向量表进行扩展，以处理相应外设的中断请求。

1.3.3 STM32F10x 标准外设库 V3.5 的文件结构

固件库的源码可以访问 ST 公司的官方网站来获取，目前最近的固件库版本为 V3.5，下载地址：<http://www.st.com/web/en/catalog/tools/PF257890#>。

STSW-STM32054 STM32F10x standard peripheral library

● Active

Design Resources Top

Sample & Buy Top







点击下载

Part Number	Version	Marketing Status	Order From ST
STSW-STM32054	3.5.0	Active	Download

(*) Suggested Resale Price per unit (USD) for BUDGETARY USE ONLY. For quotes, prices in local currency, please contact your local [ST Sales Office](#) or our [Distributors](#)

(**) The Material Declaration forms available on st.com may be generic documents based on the most commonly used package within a package family. For this reason, they may not be 100% accurate for a specific device. Please contact our [sales support](#) for information on specific devices.

将下载的压缩包解压后，可看到如下图所示的目录结构。

	_htmresc	2014/1/8 10:33	文件夹
	Libraries	2014/1/8 10:33	文件夹
	Project	2014/1/8 10:33	文件夹
	Utilities	2014/1/8 10:33	文件夹
	Release_Notes.html	2011/4/7 10:37	Maxthon Docum... 111 KB
	stm32f10x_stdperiph_lib_um.chm	2011/4/7 10:44	编译的 HTML 帮... 19,189 KB

固件库代码被放置在 Libraries 目录下，在使用时需要将这个目录拷贝到我们建立的工程目录下。Project 目录放置的是固件库的工程模板以及示例代码，具体的使用方法可以参考其中的 html 文件。Utilities 目录下放置的是 ST 公司评估板的板级相关文件，里面的代码非常有参考价值，我们后面有很多实验都是参考的其中代码。

1.3.4 Libraries 目录的文件结构

在 Libraries 目录下包含 2 个目录，分别是：

CMSIS：提供了对 STM32F10x 系列芯片的 Cortex-M3 内核的支持。

STM32F10x_StdPeriph_Driver：提供了 STM32F10x 系列芯片所有外设的操作函数。

➤ CMSIS 目录简介

CM3 文件夹下的两个文件夹分别包括了核内外设访问层 CPAL 的文件 core_cm3.h/.cpp，STM32F10x 系列 MCU 的设备外设访问层 DPAL 头文件 stm32f10x.h，设备外设访问层系统 DPALS 文件 system_stm32fx.h/.cpp 以及不同 IDE 环境下的启动引导代码。

Documentation 文件夹下的 CMSIS_Core.htm 文件，描述了何为 CMSIS(Cortex Microcontroller Software Interface Standard)。

➤ STM32F10x_StdPeriph_Driver 目录简介

inc 文件夹即 include 的缩写，其中包含所有外设操作函数的头文件。

src 文件夹即 source 的缩写，其中包含所有外设操作函数的源文件。

1.3.5 固件库的命名规则

1) 固件库的文件名都是以“stm32f10x_”做为前缀，结合“xxx 外设名缩写”组成，其中“xxx”为任一外设名的缩写，如外设为 adc，则该文件名为 stm32f10x_adc.c。

外设文件名缩写如下表所示：

外设文件名缩写	外设函数名缩写	外设/单元
adc	ADC	模数转换器
bkp	BKP	备份域寄存器
can	CAN	控制器局域网模块
dma	DMA	直接内存存取控制器
exti	EXTI	外部中断事件控制器
flash	FLASH	Flash 闪存控制器
fsmc	FSMC	可变静态存储控制器
gpio	GPIO	通用输入输出端口控制器
i2c	I2C	I2C 总线控制器

iwdg	IWDG	独立看门狗
nvic	NVIC	嵌套中断向量控制器
pwr	PWR	电源/功耗控制
rcc	RCC	复位与时钟控制器
rtc	RTC	实时时钟
spi	SPI	SPI 总线控制器
tim	TIM	通用定时器
usart	USART	通用同步异步收发器
wwdg	WWDG	窗口看门狗

- 2) 常量仅被应用于一个文件的, 定义于该文件中; 若被应用于多个文件中, 则在对应头文件中定义。所有常量都由英文字母大写书写。
- 3) 外设函数的命名以该外设的缩写加下划线为开头。每个单词的第一个字母都由英文字母大写书写, 例如: SPI_SendData。
- 4) 在函数名中, 只允许存在一个下划线, 用以分隔外设缩写和函数名的其它部分。
- 5) 名为 PPP_Init 的函数, 其功能是根据 PPP_InitTypeDef 中指定的参数, 初始化外设 PPP, 例如: TIM_Init。

1.4 建立 STM32 空工程

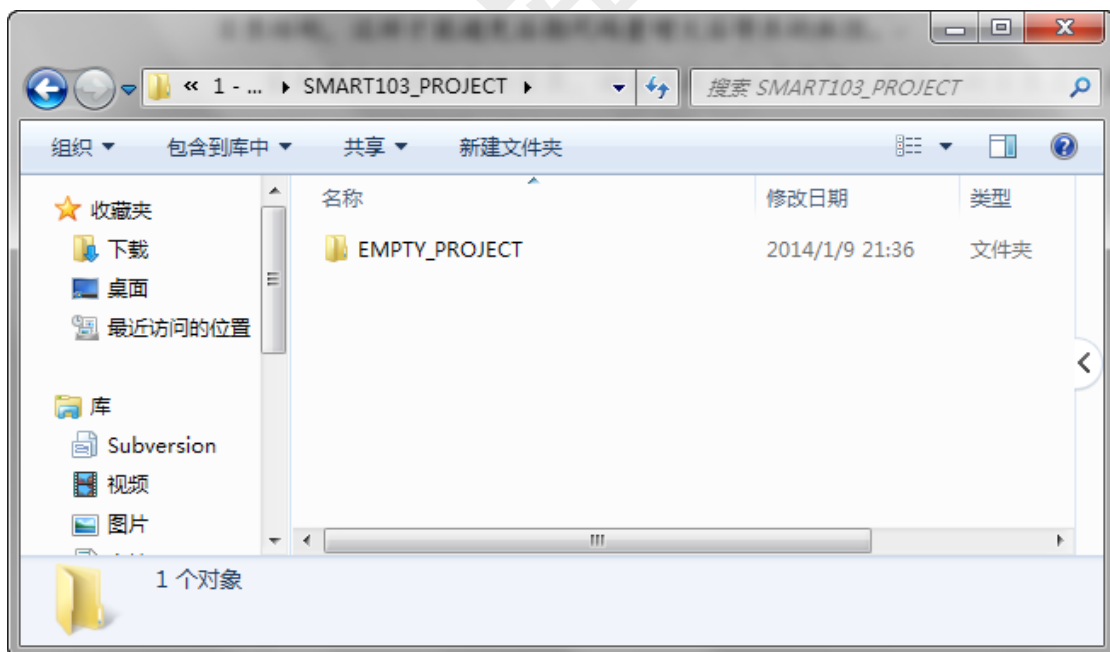
1.4.1 建立固件库工程的目的

在这个实验中我们要建立一个针对 SMART103 平台并使 ST 公司 STM32F10x 标准外设库的一个空白工程, 这个工程并不能实现 SMART103 平台上的任何功能, 它仅是为描述了如何使用固件库建立工程, 但是这个实验又非常重要, 因为在后面所有的实验都是基于这个空白工程来完成的。

1.4.2 建立工程目录及拷贝必要的文件

做为一个软件开发者, 必须要养成良好的源代码管理习惯, 如果条件允许的情况下最好使用 cvs、svn 或者 git 等版本控制软件对源码进行管理。如果是一个初学者在不了解版本控制的情况下开发软件, 就要规划好工程及源码的位置和目录结构, 这样才能避免后期代码量增大后带来的麻烦。

1. 建立工程所在的目录, 结合本实验的内容将工程所在的目录名设置为“EMPTY_PROJECT”, 建立该目录。



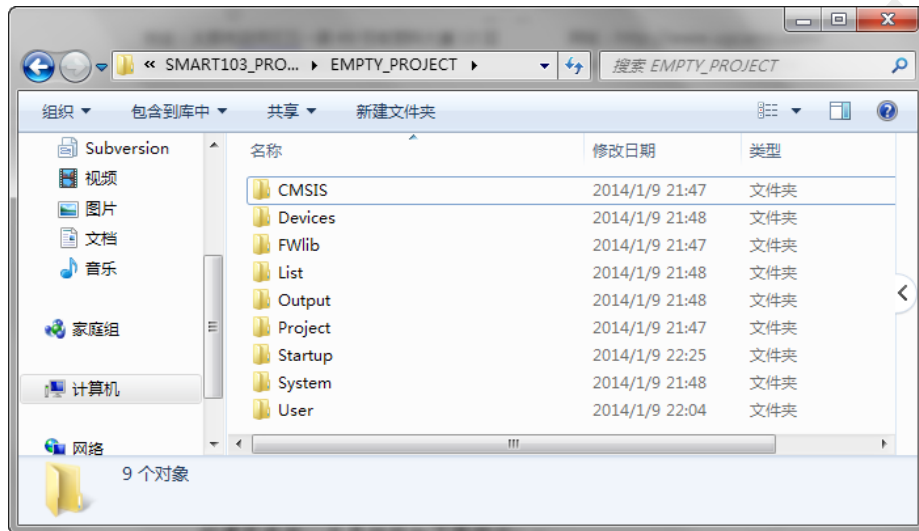
2. 将上述 STM32 固件库中 Libraries 目录下的两个文件夹拷贝到 EMPTY_PROJECT 目录下, 并将 STM32F10x_StdPeriph_Driver 目录名重命名为 FWlib。

3. 在 EMPTY_PROJECT 目录下建立以下几个目录

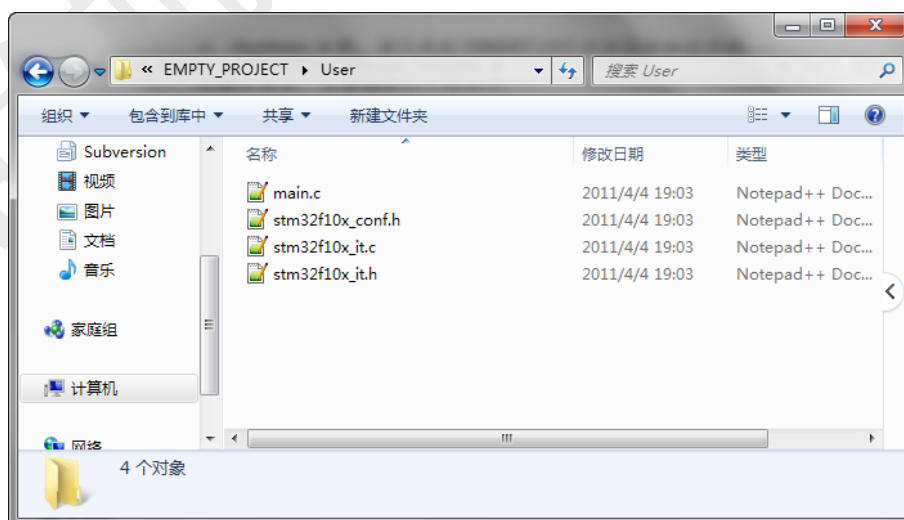
✧ Project 目录: 用于存放 keil 开发环境的工程文件。

- ✧ Output 目录：用于存放编译过程中生成的中间文件。
- ✧ List 目录：用于存放 keil 生成的 list 文件。
- ✧ Startup 目录：用于存放 STM32 处理器的启动引导代码。
- ✧ User 目录：用于存放工程中主要的用户代码，如 main.c。
- ✧ Devices 目录：用于存放 SMART103 平台板级硬件支持代码。
- ✧ System 目录：用于存放 SMART103 平台系统相关代码。

创建完成后，目录结构如下图所示：



4. 拷贝固件库中的 Project-> STM32F10x_StdPeriph_Template 目录下的“stm32f10x_conf.h”文件，“stm32f10x_it.c”文件，“stm32f10x_it.h”文件以及“main.c”文件到上一步创建的 User 目录下。



其中 main.c 文件是 ST 公司针对其评估板所设计的与 SMART103 平台并不相同，因此我们只需要根据其结构保留主函数即可，将 main.c 中原有代码删

除，加入如下代码：

```
#include "stm32f10x.h"

int main(void)
{
    while(1)
    {
    }
}
```

5. 复制启动引导代码，由于 STSMART103 平台使用的是 STM32F103VBT6 处理器，属于 STM3210x 系列中的中等密度 FLASH，因此在启动引导代码需要使用尾缀为“md”的启动引导文件。复制固件库中 Libraries-> CMSIS-> CM3-> DeviceSupport-> ST-> STM32F10x-> startup-> arm 目录下的“[startup_stm32f10x_md.s](#)”文件到工程中的 Startup 目录下。

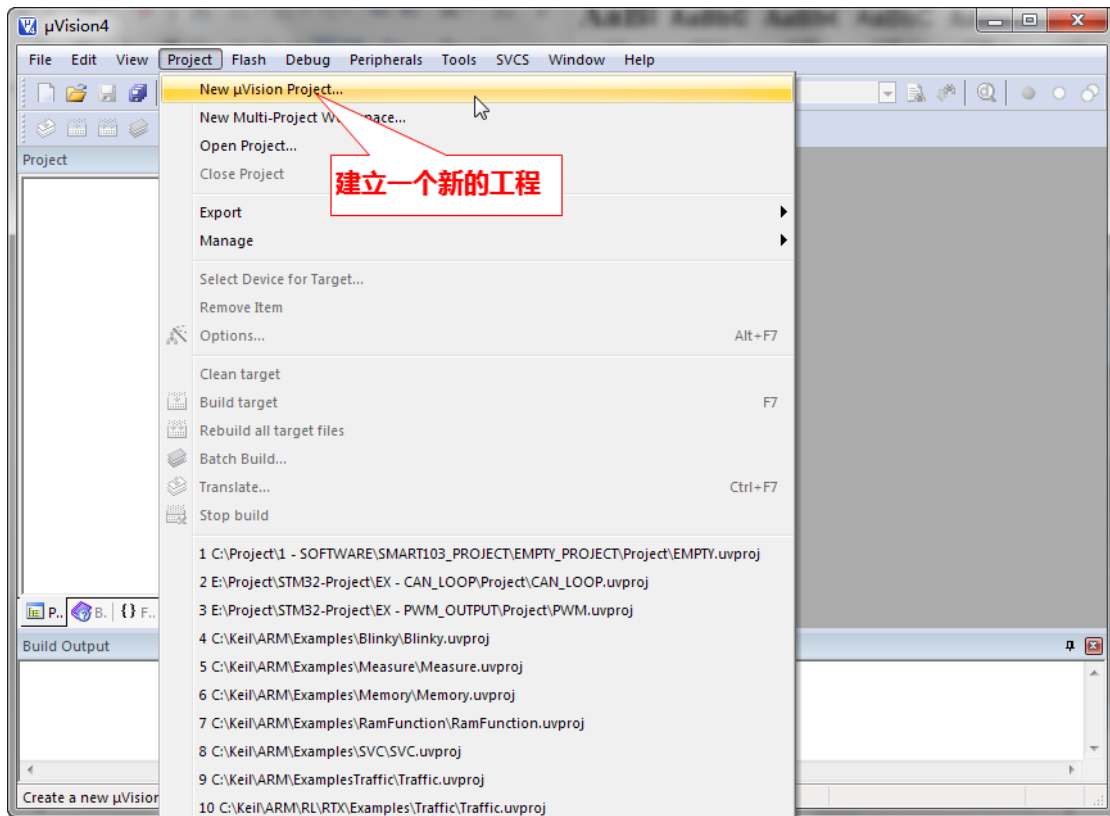
至此我们就完成了工程目录的创建及库文件到拷贝工作，现在我们的工程中有很多目录是空的，这其中的一些目录虽然在本实验中并没有使用到，但是你会在后续的实验中发现他们的用武之地。

1.4.3 使用 keil 建立工程并进行相应的配置

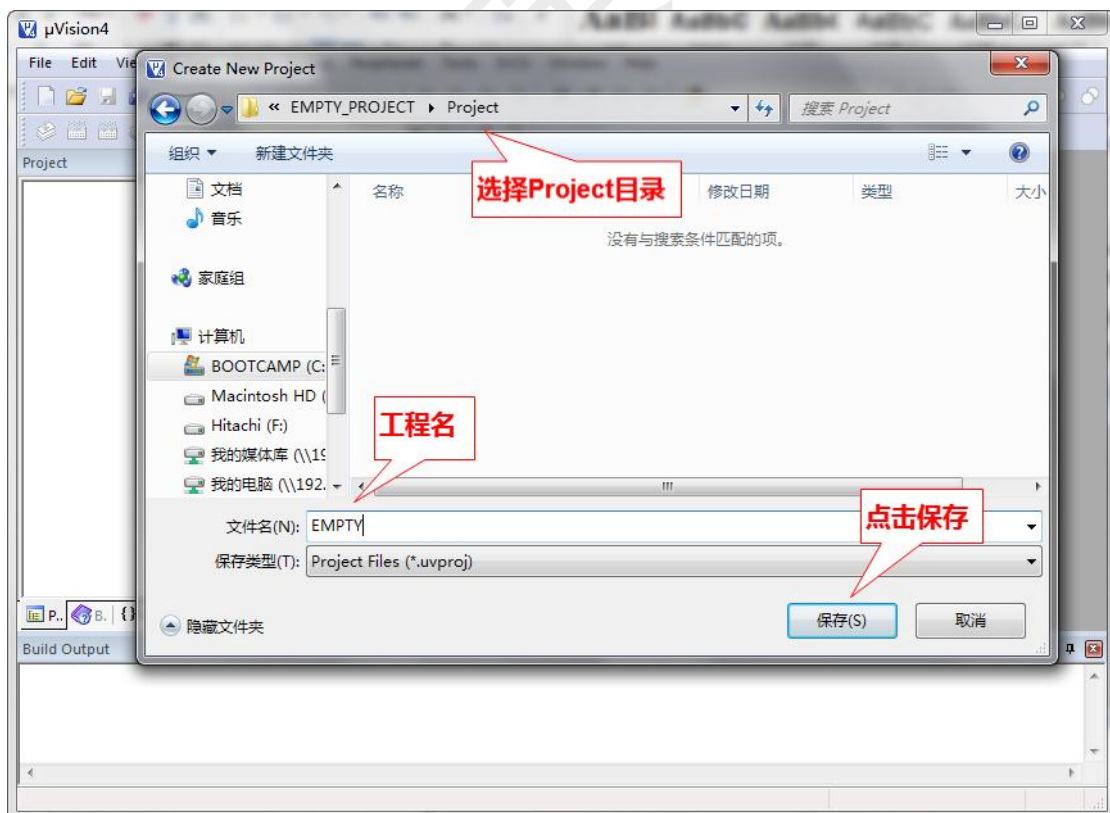
点击桌面图标，启动 μ Vision4



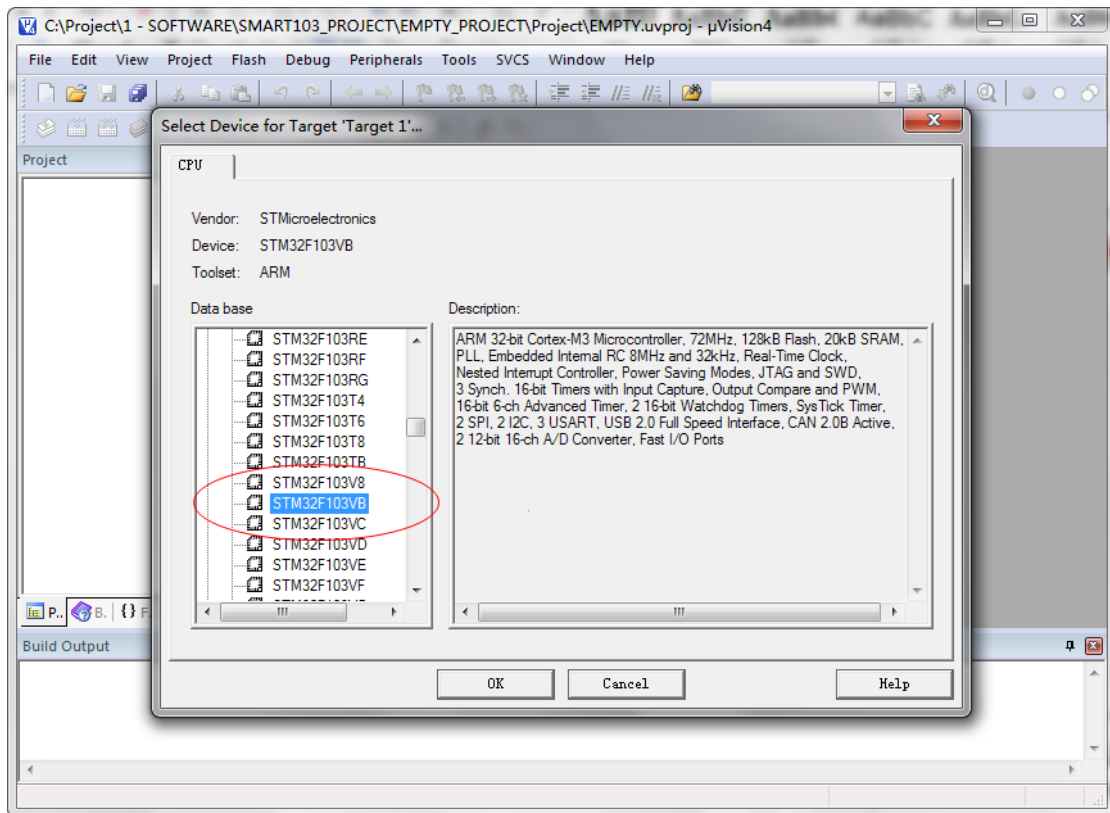
打开 keil 后建立一个基于 [STM32F103VB](#) 处理器的工程，我们需要在建立工程的时候选择对应的处理器。



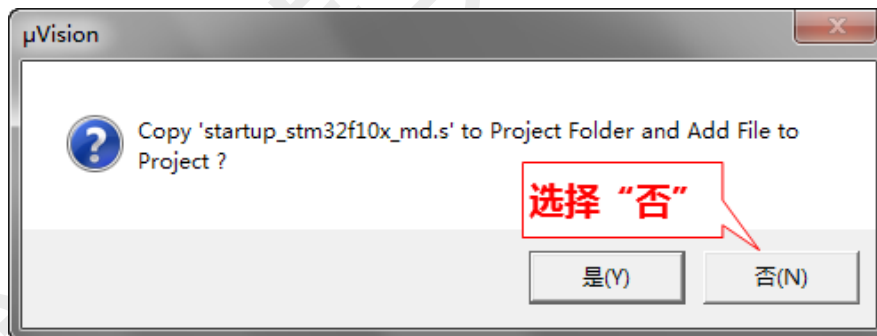
在弹出的对话框中，打开 1.4.2 中建立的 Project 目录，作为工程文件的保存目录，然后为工程文件设置一个名字，这里我们同样设为 EMPTY，点击保存。



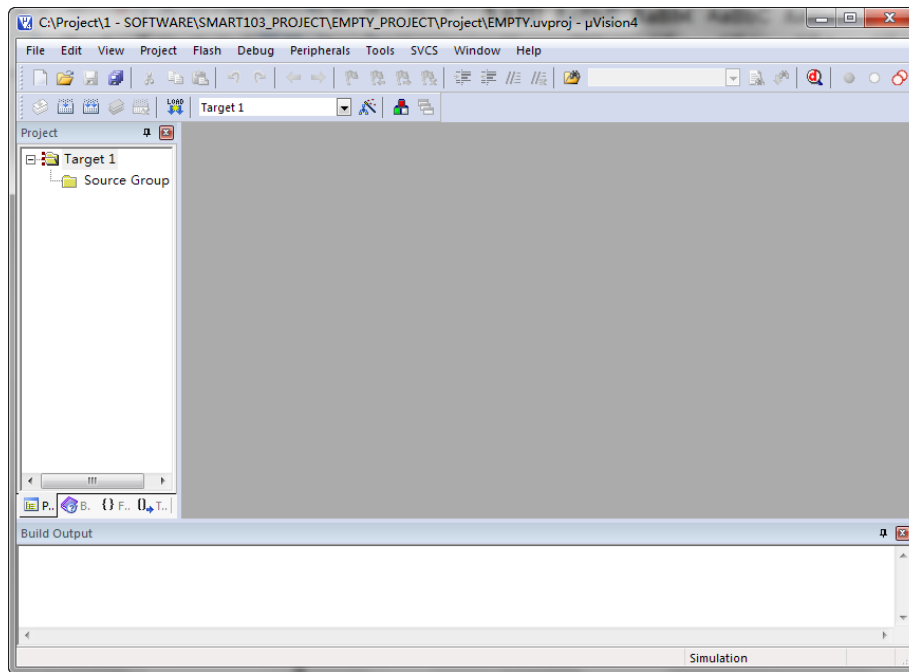
下一步，在弹出的对话框中选择处理器，这里我们选择 ST 公司的 STM32F103VB 处理器，然后点击 OK 继续。



选择好处理器后，μVision4 会询问我们是否将该处理器的启动引导代码添加到当前的工程中。由于 SMART103 平台使用固件库中提供的启动引导代码，所以在弹出的对话框中选择“否”，手动添加这部分启动引导代码。



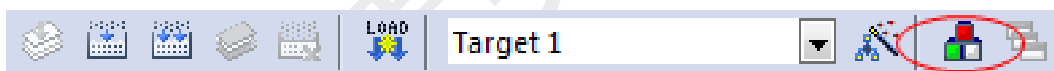
以上完成了针对于 SMART103 平台 μVision4 工程的建立，在我们对工程进行编译之前还需要对工程进行相应的配置以保证工程能够顺利的编译通过。



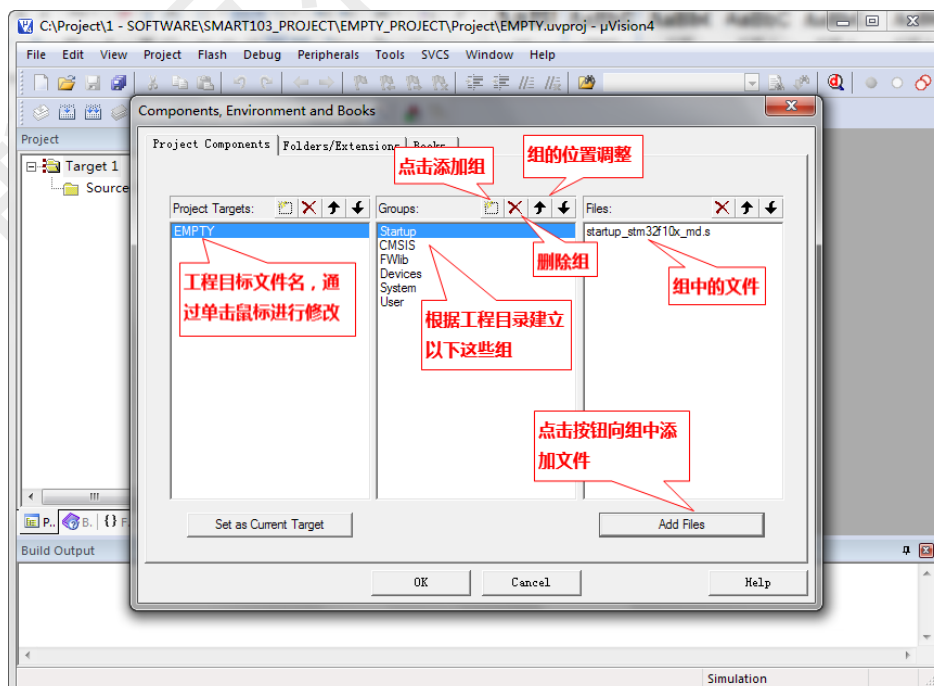
1.4.4 添加工程源码

➤ 设置工程组及相关文件

打开 μVision 的“Components, Environment and Books”菜单按钮，对工程的组(Group)结构以及组中的源码进行配置。

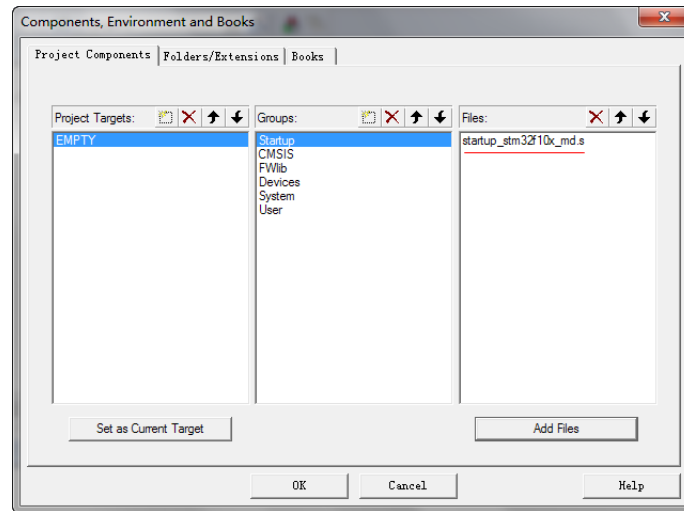


在弹出的窗口中将 1.4.2 节中建立的目录和源码文件，添加到工程的组中。

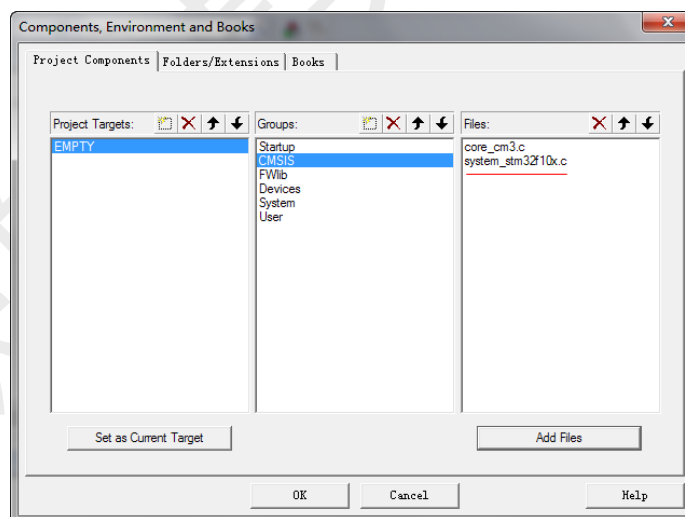


在组的文件中添加以下的内容：

- ✧ **Startup 组**：点击“Add Files”按钮，添加工程目录下 Startup 文件夹中的 `startup_stm32f10x_md.s` 文件。



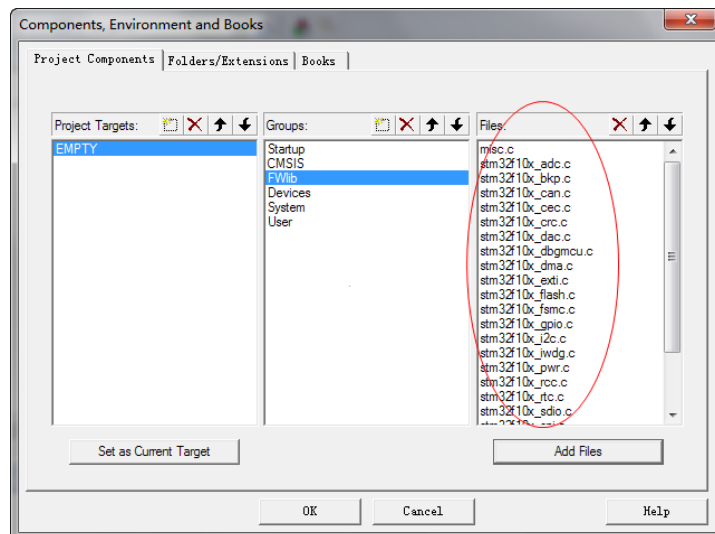
- ✧ **CMSIS 组**：点击“Add Files”按钮，添加以下文件
 - 添加工程目录下 CMSIS-> CM3-> CoreSupport 文件夹下的 `core_cm3.c` 文件。
 - 添加工程目录下 CMSIS-> CM3-> DeviceSupport-> ST-> STM32F10x 文件夹下的 `system_stm32f10x.c` 文件。



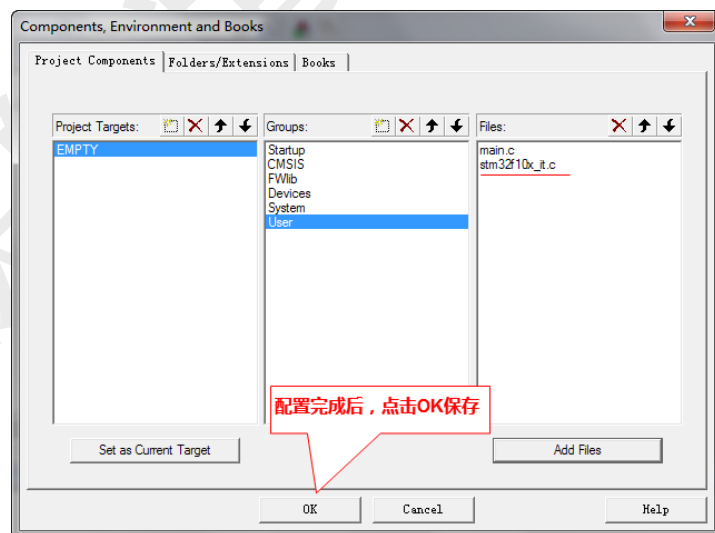
- ✧ **FWlib 组**：组中的内容实际上是固件库中 STM32F10x_StdPeriph_Driver 目录下的内容，我们只需要将 src 目录下的文件添加到组中即可。

原则上是使用哪些外设则添加这些外设的操作文件，不使用的外设则不添加到工程中。在这里为了简单起见，我们将 src 目录下的所有文件都添加到 FWlib 组中。

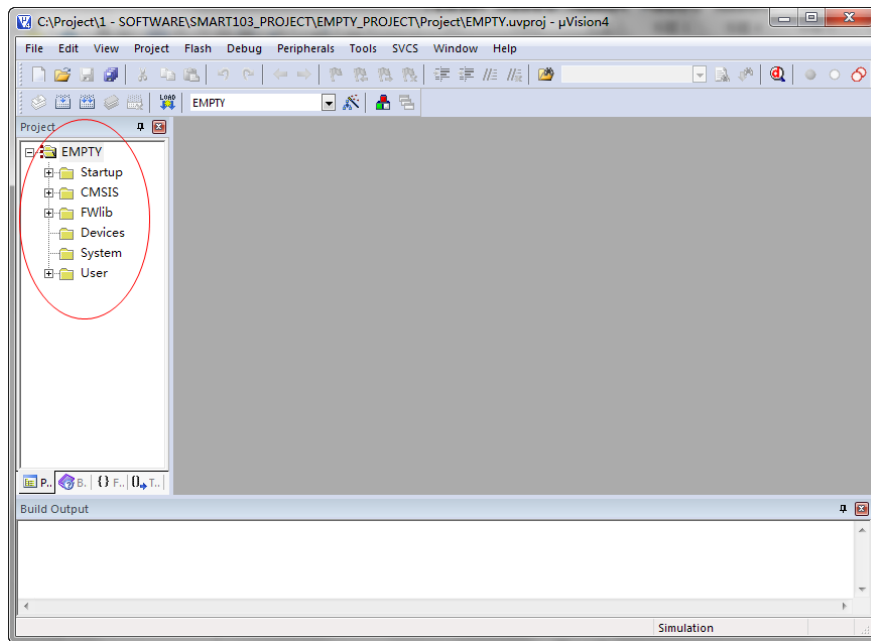
点击“Add Files”按钮，将工程目录下 FWlib-> src 文件夹中的所有文件头添加到组中。



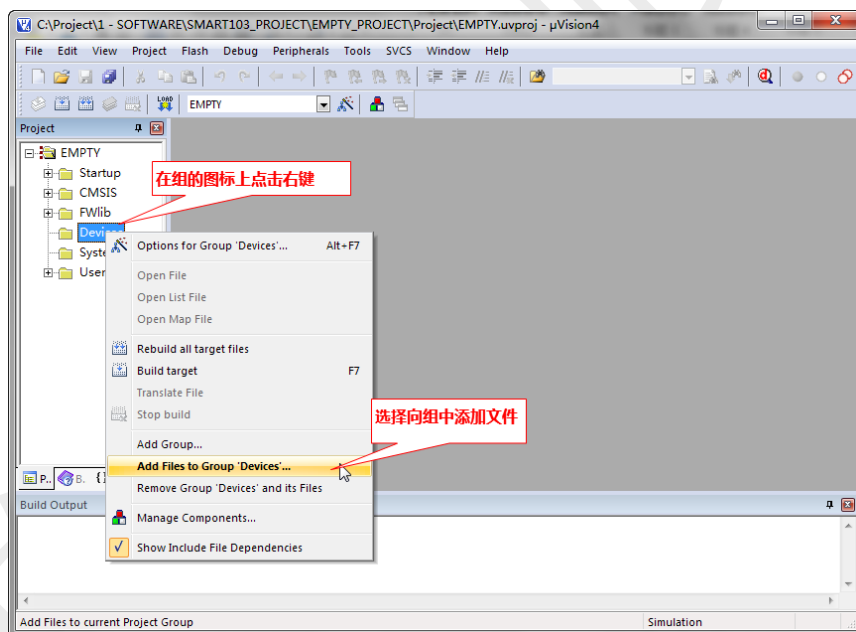
- ✧ **Devices 组**：目前为空，不需要添加任何文件。在后面的实验中我们会根据 SMART103 具体设备情况，向其中添加设备操作文件。
- ✧ **System 组**：目前为空，不需要添加任何文件。
- ✧ **User 组**：点击 “Add Files”按钮，将工程目录中 User 文件夹下的 **main.c** 及 **stm32f10x_it.c** 添加到组中。
 - main.c 文件为工程的主函数所在文件。
 - stm32f10x_it.c 文件是 STM32 处理器中断处理函数所在的文件。



完成以上配置后，我们就将工程编译所需的源码都添加到相应的组中了，有了这样的分组可以便于对源码进行管理，上一步骤建好的工程组如下图所示：

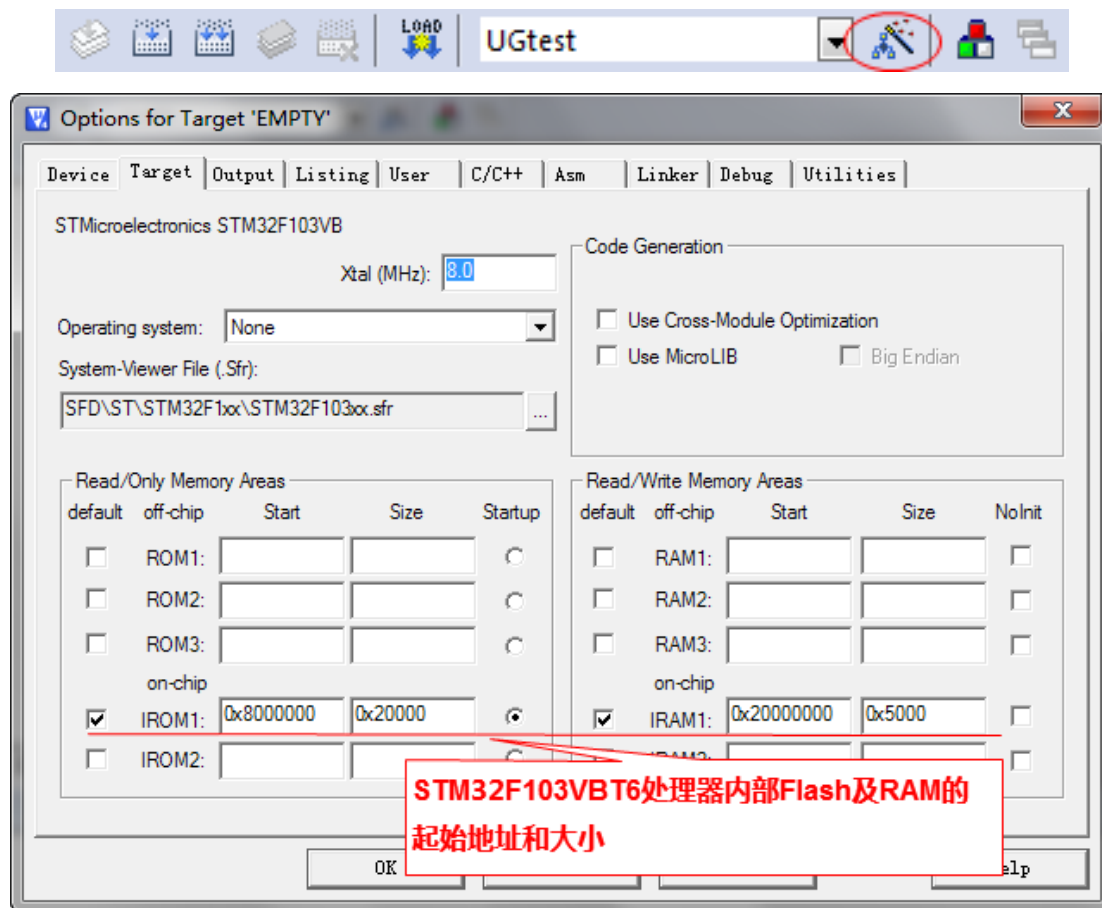


除了前面介绍的向组中添加源码的方法外，我们还可以在上图的组的图标上点击右键，选择“Add Files to Group”添加源码。



1.4.5 配置编译选项

在 1.4.4 节中我们已经将工程所需的源码添加到了工程中，在编译代码之前需要对工程的编译选项进行一些配置。通过 Project->Options for Target 菜单项打开“Options for Target”对话框，或者使用下图中的工具栏按钮，打开该对话框，在其中对如何编译生成的目标文件进行一系列的配置。



其中 Target 目标属性如下表所示：

对话框项	描述
Xtal	设备的晶振(XTAL)频率。大多数基于 ARM 的微控制器都使用片上 PLL 产生 CPU 时钟。所以，一般情况下 CPU 的时钟与 XTAL 的频率是不同的。
Read/Only Memory Area	片内、片外的 ROM 区地址以及大小
Read/Write Memory Areas	目标硬件的片内和片外的 RAM 区地址以及大小
Code Generation	代码生成规则，是否使用优化及 C 的小型库

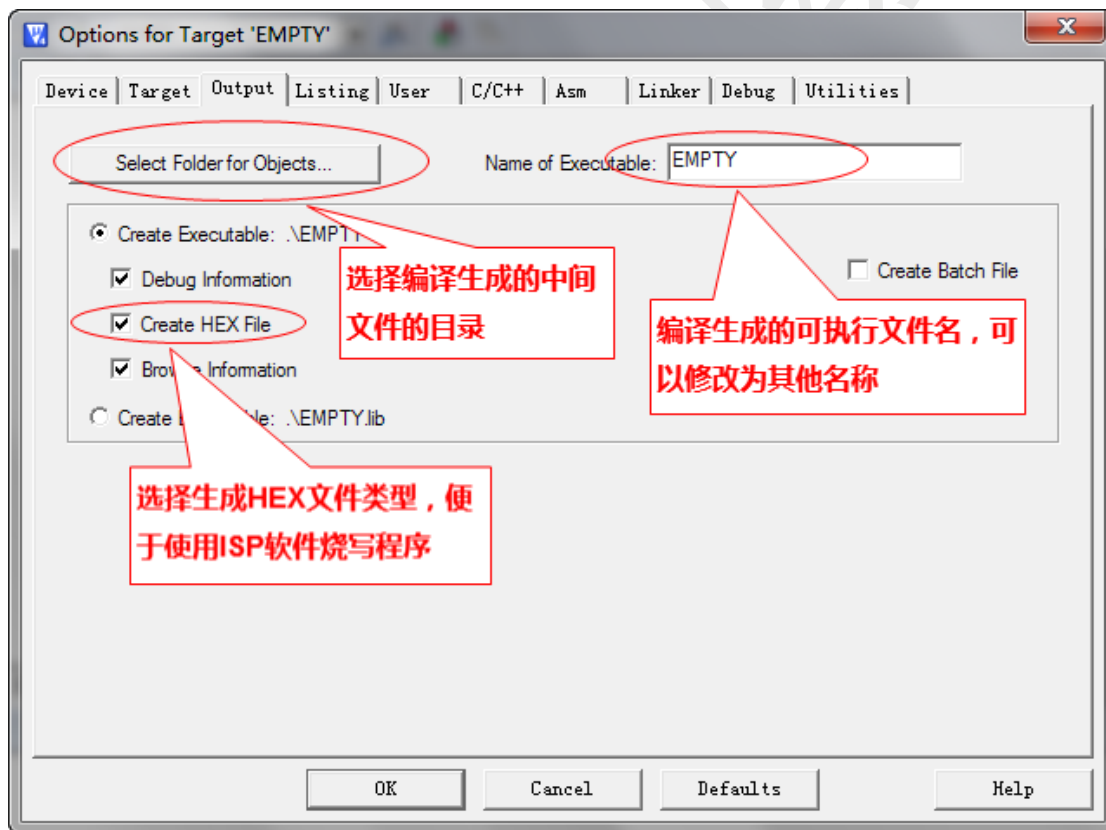
其他选项卡功能简介：

对话框项	描述
Device	从 μ Vision 的设备数据库中选择选择设备。
Target	为应用程序指定硬件环境。
Output	定义工具链的输出文件，在编译完成后运行用户程序。

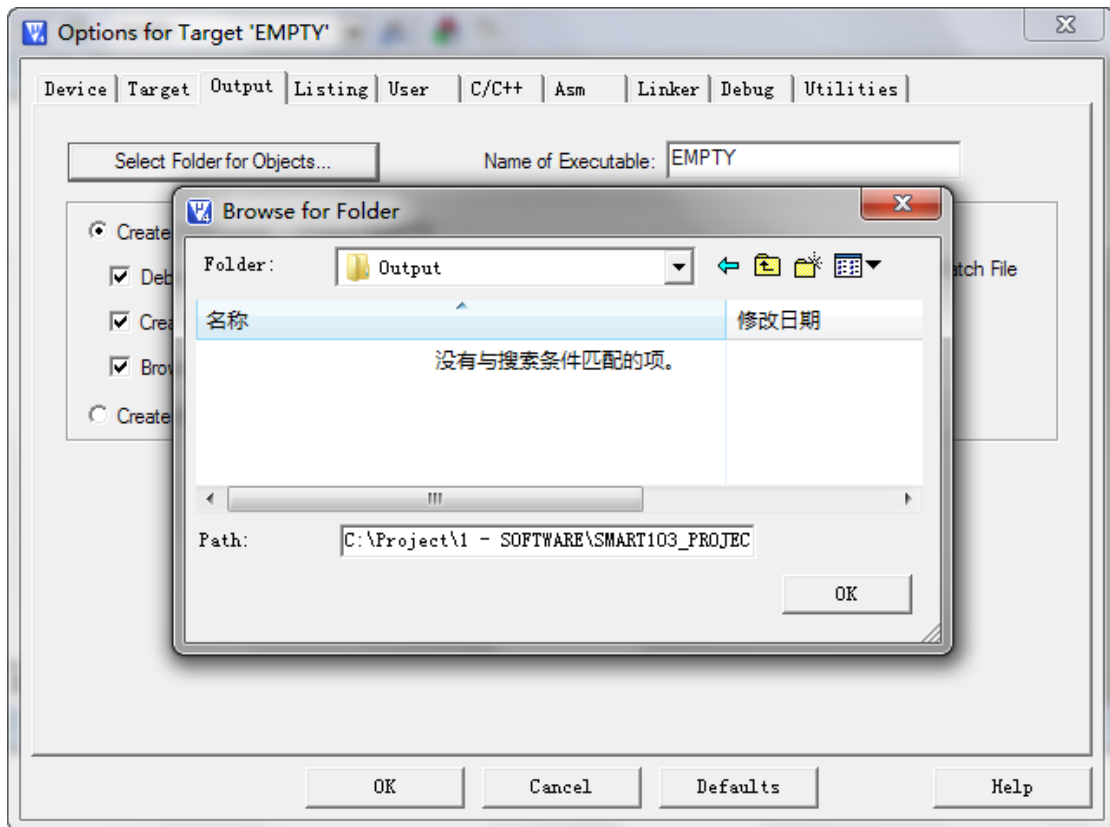
Listing	指定工具链产生的所有列表文件。
User	用户自定义编译命令
C/C++	设置 C 编译器的工具选项，例如代码优化和变量分配。
Asm	设置汇编器的工具选项，如宏处理。
Linker	设置链接器的相关选项。一般来说，链接器的设置需要配置目标系统的存储分配。设置链接器定义存储器类型和段的位置。
Debug	µVision 调试器的设置。
Utilities	配置 Flash 编程实用工具。

在了解了上述选项卡功能后，针对于 SMART103 平台的工程项目需要进行以下几步的配置。

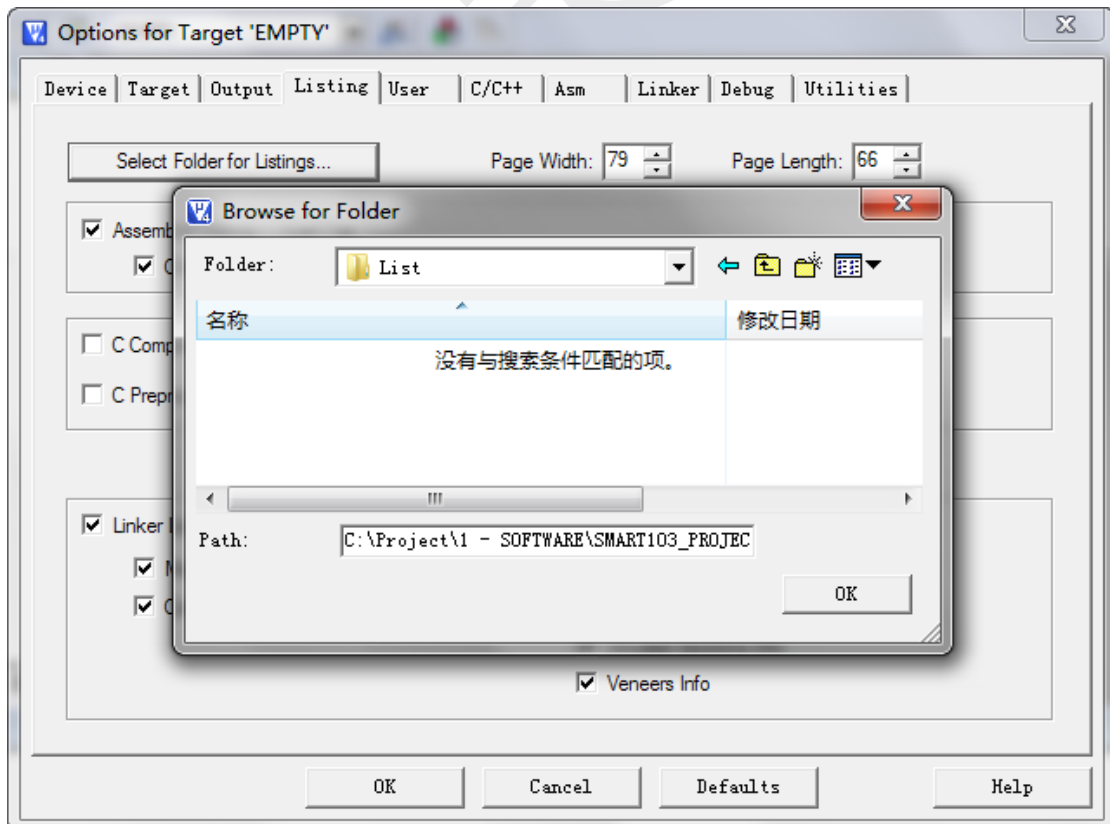
第一步，配置 Output 选项卡，设置编译生成的中间文件的目录，以及最终编译生成的可执行文件的名称及类型。



选择工程目录下的 Output 文件夹放置编译生成的中间文件，通过“Select Folder for Objects...”按钮选中这个目录。



第二步，配置 Listing 选项卡，同配置 Output 类似，选择工程目录下的 List 文件夹放置编译生成的列表文件，通过“Select Folder for Listings...”设置。

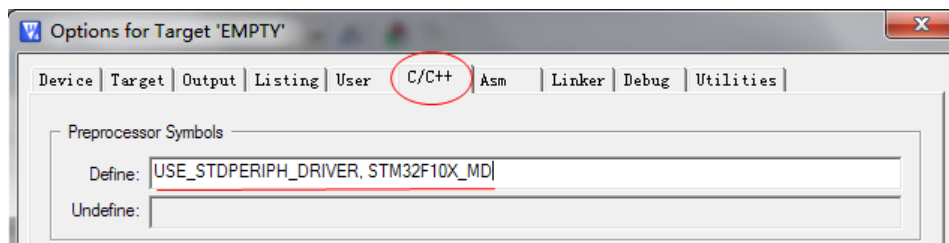


第三步，配置 C/C++ 选项卡，在这个选项卡中我们需要设置预编译的符号

定义，以及用于搜索头文件的路径。

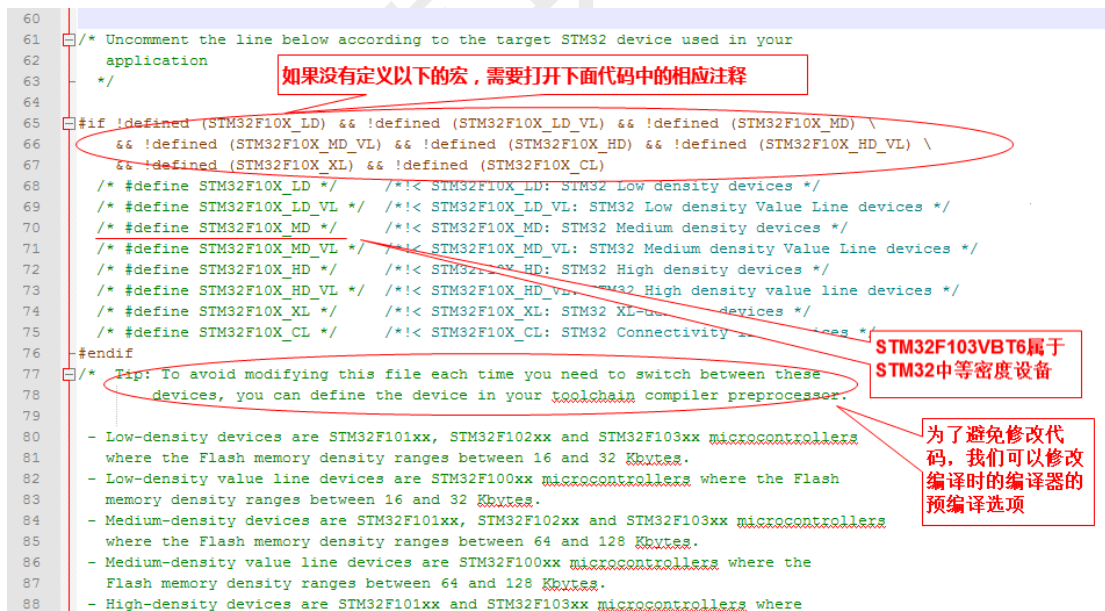
◇ 添加预编译宏定义

在下图所示位置添加 `USE_STDPERIPH_DRIVER` 和 `STM32F10X_MD` 两个宏定义。



说明：这一个步非常重要，如果不在这里定义这两个宏的话，我们就必须修改固件库中的源码，才能保证编译通过。但是每次都去固件库中修改源码又非常麻烦，因此我们在编译器选项中定义就方便很多。

那为什么要定义这两个宏呢？因为我们使用的固件库是一个针对 STM32F10x 系列处理器的，每个处理器的特性都不一样。也就是它们使用的内部 Flash、RAM 的大小和所包含的外部控制器都不一致，因此为了能够能够让固件库兼容所有的处理器，所以代码中通过一系列的条件编译来实现。下面就是节选自 `stm32f10x.h` 头文件中的部分代码。



通过阅读代码中的注释发现在使用不同的 STM32 产品时，需要根据使用的具体产品定义相关的宏，否者的话需要打开代码中的注释。SMART103 平台使用的 STM32F103VBT6 属于 STM32 中等密度设备，因此需要定义在编译时定义 `STM32F10X_MD` 宏，否则的话需要去掉上图代码中红色标示处的代码注释。

在这里我们推荐在编译时定义 STM32F10X_MD 宏的方法，这样更简单而且便于修改。

```

93  the Flash memory density ranges between 512 and 1024 Kbytes.
94  - Connectivity line devices are STM32F105xx and STM32F107xx microcontrollers.
95  */
96
97  #if !defined (STM32F10X_LD) && !defined (STM32F10X_LD_VL) && !defined (STM32F10X_MD) && !defined (STM32F10X_MD_V
98  #error "Please select first the target STM32F10x device used in your application (in stm32f10x.h file)"
99  #endif
100
101  #if !defined USE_STDPERIPH_DRIVER
102  /**
103   * @brief Comment the line below if you will not use the peripherals drivers.
104   * In this case, these drivers will not be included and the application code will
105   * be based on direct access to peripherals registers
106   */
107   #define USE_STDPERIPH_DRIVER
108  #endif
109
110  /**
111   * @brief In the following line adjust the value of External High Speed oscillator (HSE)
112   * used in your application
113
114   Tip: To avoid modifying this file each time you need to use different HSE, you
115   can define the HSE value in your toolchain compiler preprocessor.
116   */
117  #if !defined HSE_VALUE
118  #ifdef STM32F10X_CL
119  #define HSE_VALUE ((uint32_t)25000000) /*!< Value of the External oscillator in Hz */
120  #else
121  #define HSE_VALUE ((uint32_t)8000000) /*!< Value of the External oscillator in Hz */

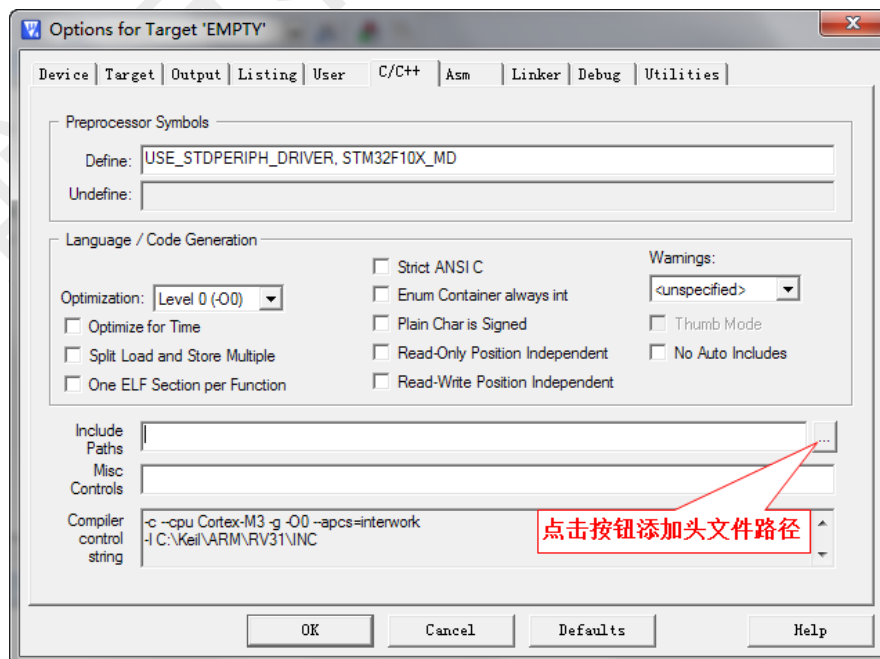
```

如果未定义这个宏，那么在访问外设时将不使用ST提供的固件库，而是使用直接操作外设寄存器的方式

而 USE_STDPERIPH_DRIVER 这个宏决定了是否使用固件库中提供的外设操作函数来操作外设，如果未定义这个宏，那么在访问外设时将不使用固件库中的函数而是直接操作外设寄存器。因此我们需要在编译器的预编译选项中定义这个宏。

✧ 添加头文件路径

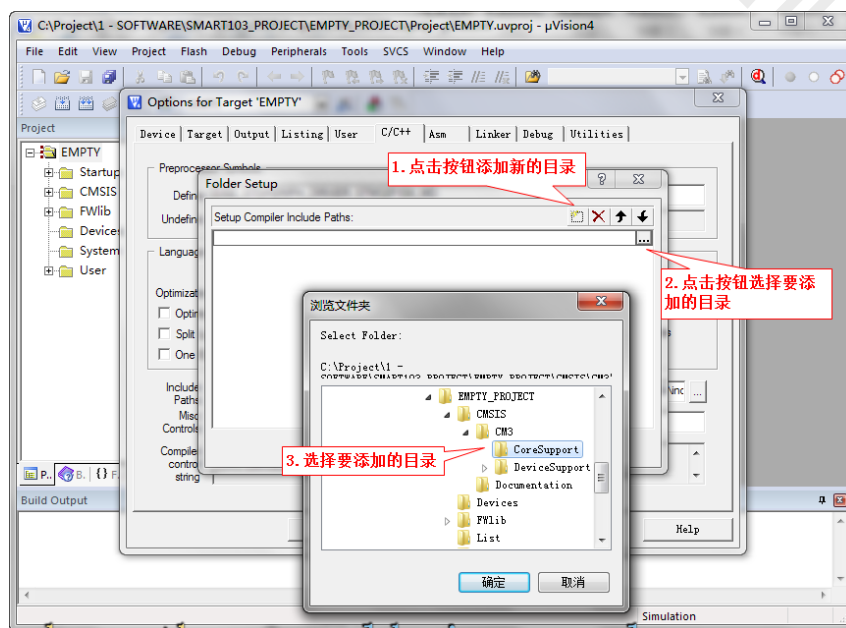
这步的目的是让编译器能够正确找到我们在代码中 include 的头文件，点击下图中的按钮添加头文件路径。



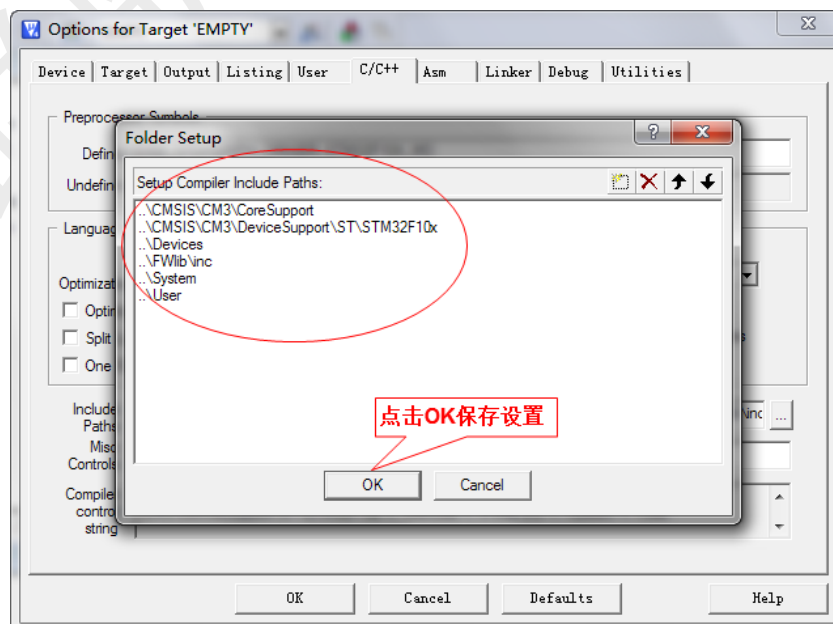
在弹出的窗口中添加头文件路径，我们需要将工程目录下的这些目录都添加到头文件目录中。

- CMSIS\CM3\CoreSupport 目录
- CMSIS\CM3\DeviceSupport\ST\STM32F10x 目录
- Devices 目录
- FWlib\inc 目录
- System 目录
- User 目录

添加方法如下图所示：



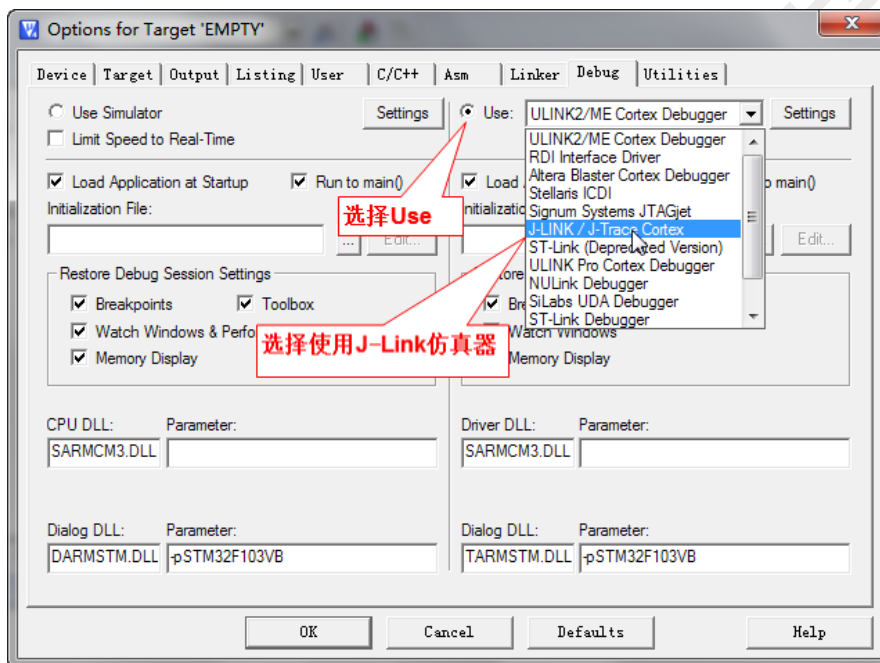
添加完成后，如下图所示：



◇ 配置 **DEBUG** 选项

这个选项卡中可以对 **DEBUG** 功能进行设置,如果你手头上有 J-Link、ULink 这类仿真器,可以在这个选项卡中设置使用这些仿真器进行程序的在线调试。如果没有这些硬件仿真器的话也没有关系,μVision 也提供了软件仿真的功能,使用软件仿真的话,这个选项卡就不需要设置,也就是说使用默认设置就好。

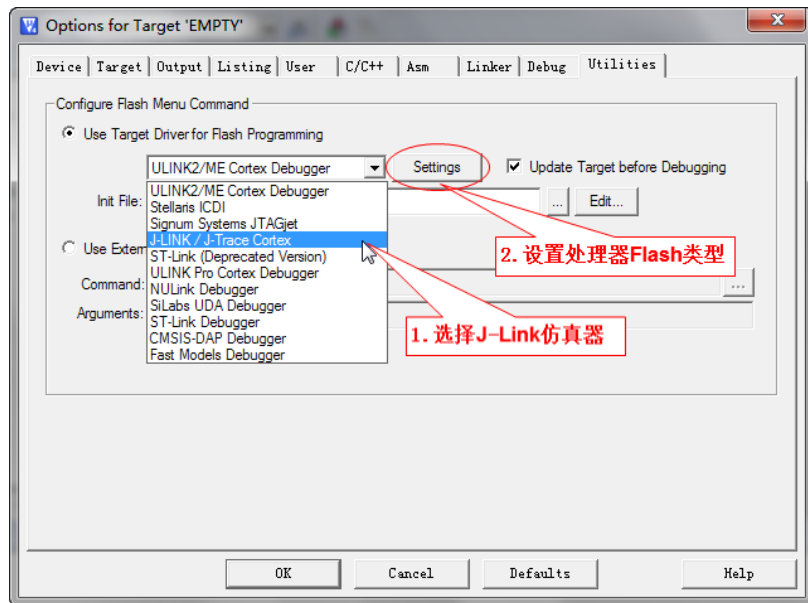
配置硬件仿真器的方法很简单,这里我们添加一个 J-Link 仿真器做为我们的在线调试工具,只需要在该选项卡的右侧选择 **use**,并在下拉菜单中找到 J-Link 选项即可。



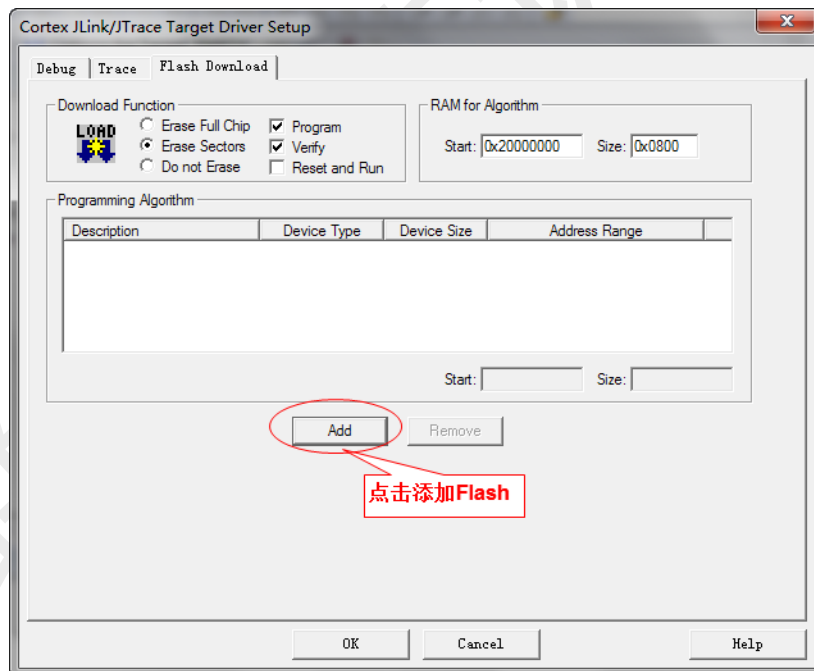
◇ 配置 **Utilities** 选项

这个选项卡用来设置如何向片内 Flash 中烧写程序,如果使用 ISP 下载方式的话这个选项卡不需要设置。这里我们添加一个 J-Link 下载程序的方式,添加方法是先选择使用的仿真器,然后再添加一个具体设备的 Flash 类型。

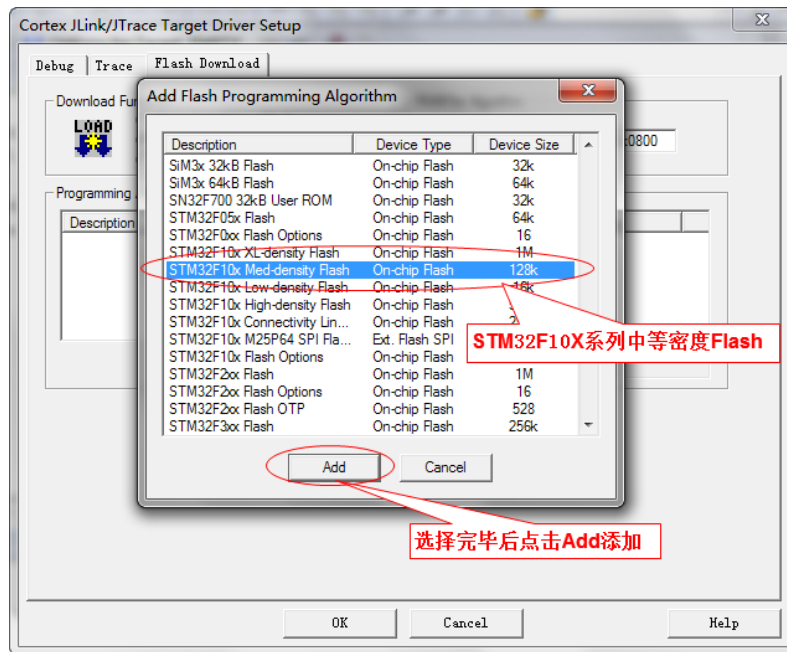
首先,添加 J-Link 仿真器。



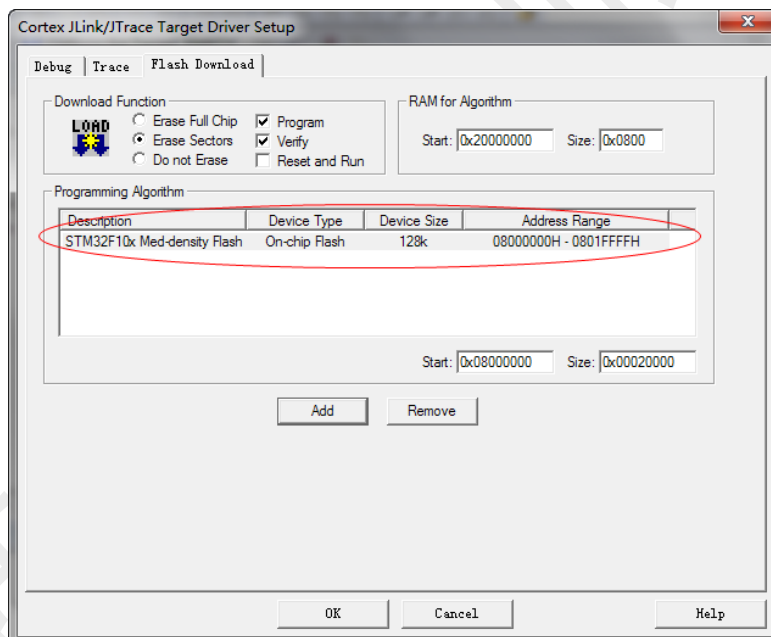
然后点击“Settings”按钮，在弹出的窗口中设置 Flash Download 类型。由于 SMART103 使用的是 STM32F103VBT6 处理器，属于 STM32 处理器系列中的中等密度 Flash，容量为 128K，所以需要在“Flash Download”选项卡中添加一个 STM32F10x 的中等密度的 Flash。



在弹出的 Flash 类型中，选择“STM32F10X Med-density Flash”类型 Flash，容量大小为 128K。选择完成后点击 Add 按钮添加该选项。

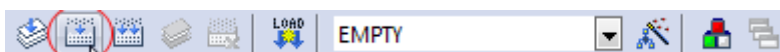


选择完毕后，如下图所示。至此，所有需要配置的内容均已经配置完成，我们可以编译一下这个工程，检验一下设置是否正确。

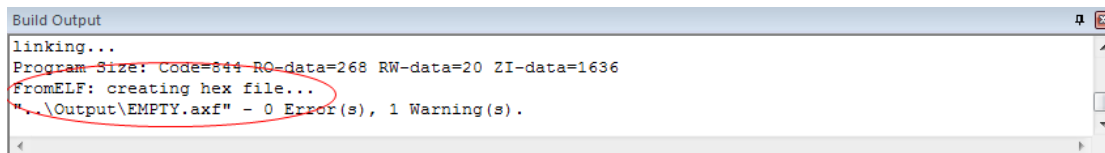


1.5 编译链接及调试

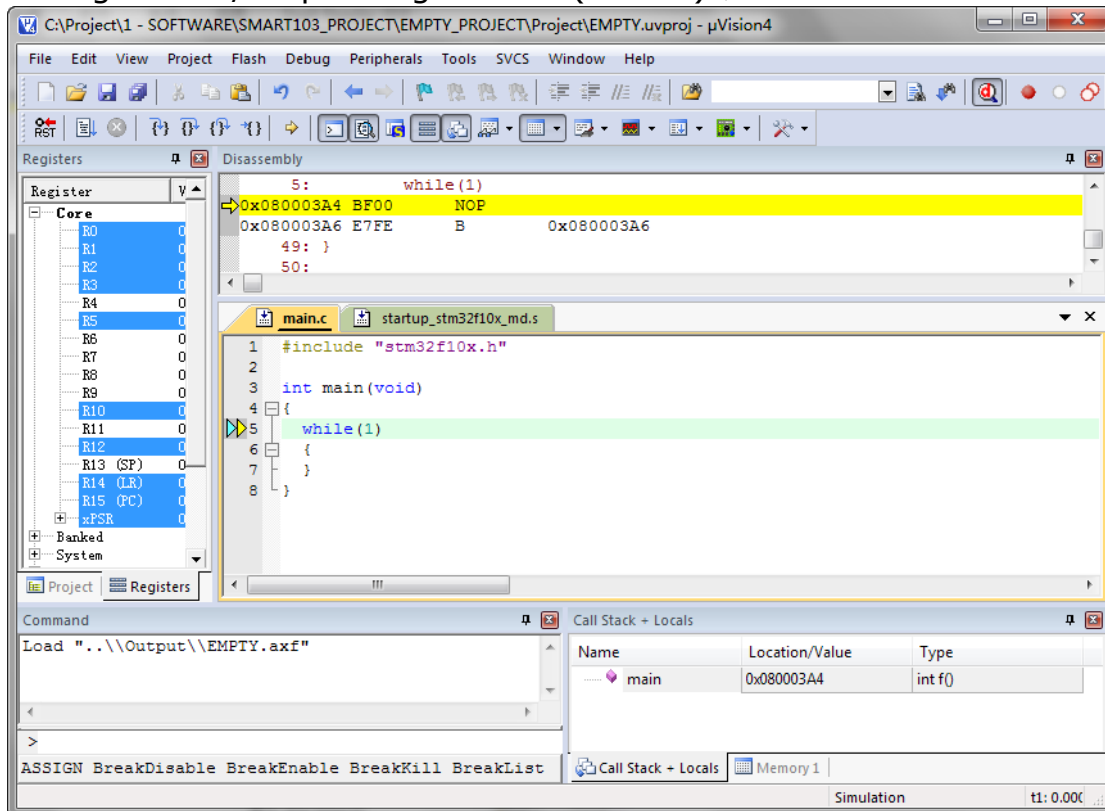
单击 Build Target (F7) 工具栏按钮将编译所有的源文件，链接应用程序。



当编译有语法错误的程序时，μVision4 将在 Output Window -> Build 窗口中显示错误和警告信息。单击这些信息行，μVision 将会定位到相应的源代码处。



源代码编译成功产生应用程序后就可开始调试了，点击 Debug->Start/Stop debug session (Ctrl F5)即进入调试模式。



进入调试模式之后，可以选择单步、全速运行。可以设置断点等常规的调试。所有有关调试的操作都可以在 Debug 菜单下找到。

常用的调试手段：

- 单步、全速运行程序

F10 单步运行， F5 全速运行。

- 对于各种模式下的寄存器，可以在左边的窗口查看

对于 ARM 的 7 种模式下的寄存器，都可以查看。当处理器处于任何一种模式时，可以查看 Current 中所有的寄存器的值，处理器从一种状态改变到另外一种状态时，该模式下物理上独立的寄存器将会被用到。

- 设置断点

选中需要设置断点的行，然后 F9 即在改行设置断点，程序运行到此处就停止运行。

- 查看变量的实时值

对于 local 的变量，打开 View->Watch&Call Stack Window，在此 Window 中，选择 Locals tab 就可以查看所有的 local 变量。

对于全局变量，选择 Watch window 中的 Watch #1，加入你需要查看的变量就可以查看实时的全局变量的值。

- 外设模块仿真

选择 Simulator 仿真模式，可以通过 RealView MDK 强大的仿真功能来调试程序。打开 Peripheral->GPIO 可以看到每一个 GPIO pin 的实时状态信息。全速运行程序后，GPIO 的状态就开始按照程序的控制开始变化。

更新说明

时间	变更内容
2014.1.11	完成空工程建立实验 V1.0