STM32F10x 标准外设库结构分析

Research On STM32F10x Standard Peripherals Library

BY:AIRWOLF

QQ:67792012

E-mail:Airwolf0992@163.com

前言 说一说为什么写这篇文章

学习 STM32 近一周了，同时也在网上找了很多的资料，下载了神州、Alientek、半壶水、芯达、红牛、旺斯泰等学习板的光盘或说明书。加之自己的奋斗版 V3 实验板的资料。为了上手快点，打算从库函数的使用看起，问题随之而来，如何建立一个合理的项目工程文件。网上有许多教程，关于不通的开发环境的，按照教程说明能够建立起一个完整的工程，但是自己总感觉怪怪的，为什么要这样建立工程，到底那些头文件里都有哪些函数的申明和数据类型定义、寄存器映射等。只有这样才能建立符合自己需求的工程。遂下定决心写一篇关于标准外设库结构的文章，和大家一起学习，共同探讨，同时也能帮助和我有一样疑虑的朋友。

我用的是 3.5 版本的库，开发环境用的是 RealView MDK-ARM V 4.12。

第一部分 STM32F10x 标准外设库概述

STMF10x 标准外设库是是一个固件函数包，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，使用本固态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。

该固态函数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。但这会增加了成本，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。

第二部分 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 处理器的微控制器的情况。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。

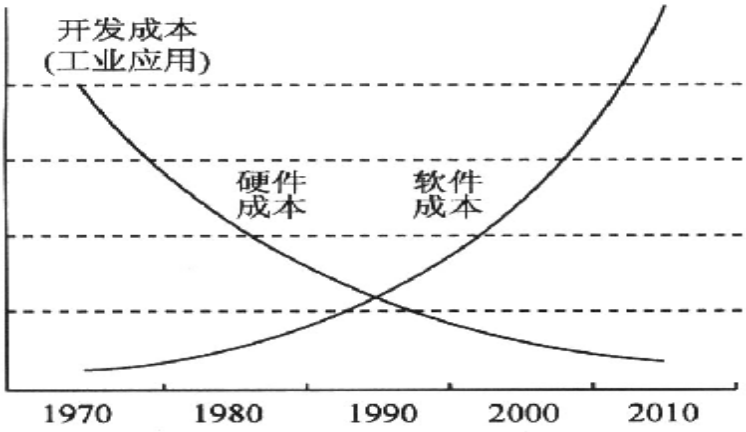


图 1　软件与硬件开发成本对比

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。

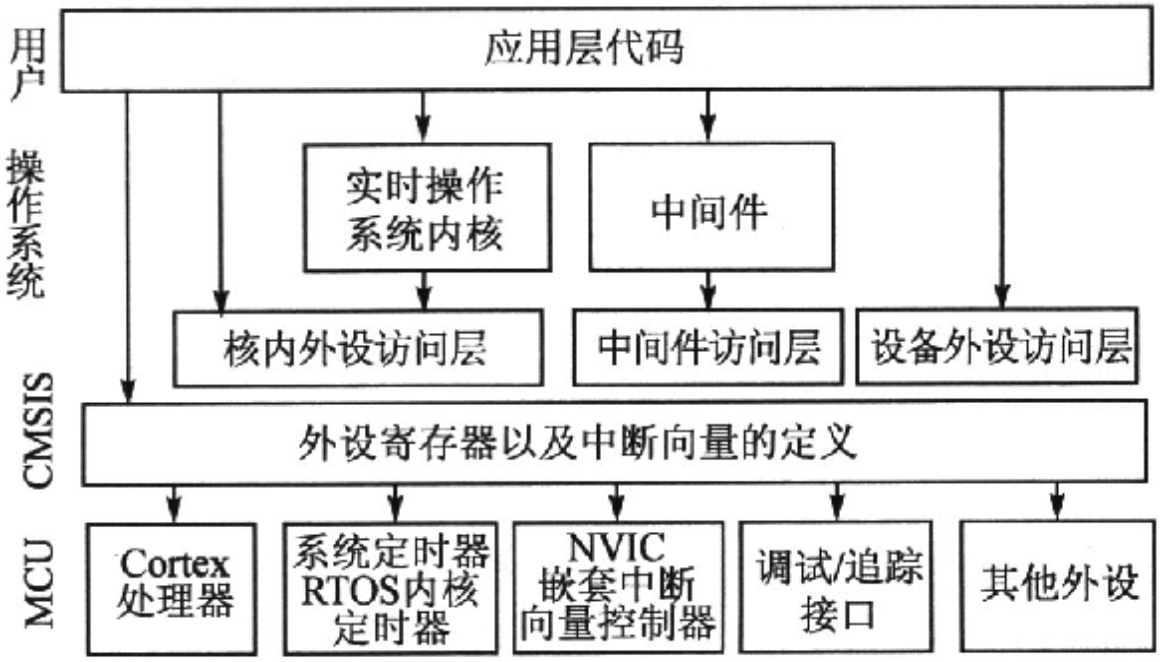


图 2　基于 CMSIS 标准的软件架构

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。

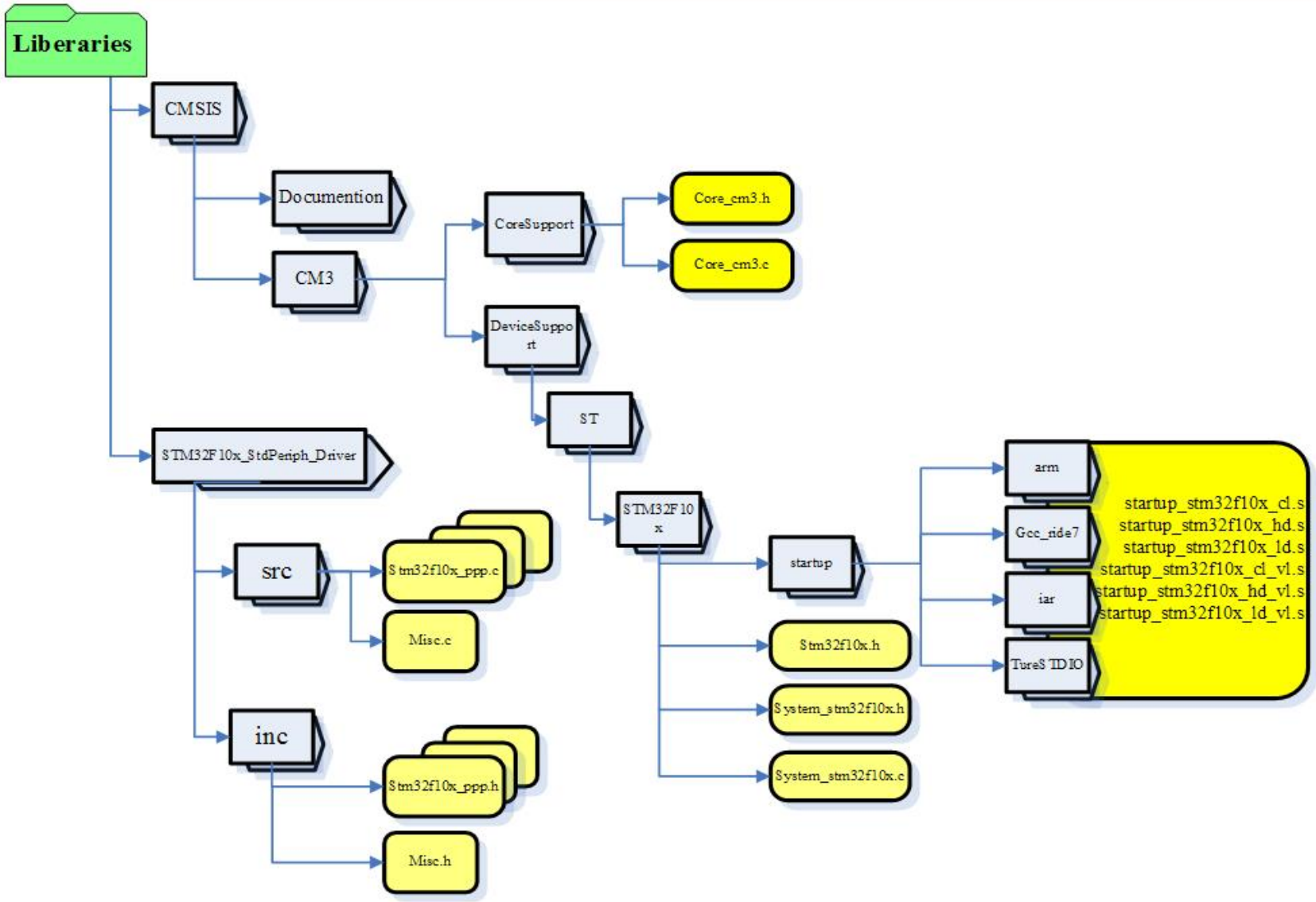
第三部分 标准外设库的 CMSIS 结构

为了搞清出标准外设库的结构，我们先来看一下其物理组成，即其有哪些文件和如何分布的。然后再一一介绍每个文件的内容及所包含的函数。再次就可以分析出其逻辑结构。

首先请看下图，这是从网上下载的 STM32F10x_StdPeriph_Lib.zip 文件解压后其 stm32f10x_stdperiph_lib\STM32F10x_StdPeriph_Lib_V3.5.0 下 Libraries 目录的结构。Libraries 目录就是我们编程时所需要运用的，我们可以把它 copy 到工程文件夹下，方便调用。

具体的文件我就不再描述了，大家看图便知。下面着重看看每个文件夹的作用及其包含文件的作用。

STM32F10x Standard Peripherals Library V3.5 Structure



CMSIS 文件夹提供了对 STM32F10x 系列芯片的 Cortex M3 内核的支持。

Documentation 下有个 CMSIS_Core.htm 文件，其描述了 Cortex Microcontroller Software Interface Standard (CMSIS)。

CM3 文件夹下的两个文件夹分别包括了核内外设访问层 CPAL 的头文件 core_cm3.h 及为 STM32F10x 系列 MCU 写的设备外设访问层 DPAL 头文件 stm32f10x.h、设备外设访问层系统 DPALS 头文件 system_stm32fx.h。

（一） core_cm3.h

0、内对 Lint 进行了配置。

1、最重要的是调用了“stdint.h”文件，该文件由编译环境提供，对 8 位、16 位、32 位等整数类型的定义及其范围进行了规范，还定义了大数输出如：UINT_LEAST8_MAX。主要用来屏蔽不同编译器之前的差异。这种扩展整数类型的定义非常清晰，从类型名字上就可以看出它的长度，这有利于编写可移植的代码。

2、指示寄存器的访问权限。CMSIS 定义以下 3 种标识符来指定访问权限：_I(volatileconst)、_O(volatile)和_IO(volatile)。其中_I 用来指定只读权限，_O 指定只写权限，_IO 指定读写权限。

3、对 CM3 核内的寄存器进行了定义。

定义了 NVIC 类型结构体 NVIC_Type、SCB 结构体 SCB_Type、SysTick 类型结构体 SysTick_Type、ITM 类型结构体 ITM_Type、MPU 类型结构体 MPU_Type、InterruptType 类型结构体 InterruptType_Type、CoreDebug 类型结构体 CoreDebug_Type。定义了各种寄存器。

4、对 CM3 硬件内存地址进行了映射。

5、对硬件抽象层的寄存器进行了定义，包括 Cortex_M3 核的全局变量声明和定义，并定义一些静态功能函数。。用于 C 语言文件内调用汇编语句如: __NOP、__ISB()、__DSB()等。其原型在 core_m3.c 文件内。

6、调试

嵌入式软件开发中的一个基本需求就是能通过终端来输出调试信息，一般可通过 2 种方式实现: 一种是使用串口线连接板上的 UART 和 PC 上的 COM 口，通过 PC 上的超级终端来查看调试信息; 另一种则是采用半主机机制，但有可能不被所用的工具链支持。基于 Cortex-M3 核的软件调试突破了这样的限制，Cortex-M3 内核提供了一个 ITM(InstrumentationTraceMacrocell)接口,通过SWV(SerialWireViewer)可调试由 SWO 引脚接收到的 ITM 数据。ITM 实现了 32 个通用的数据通道，基于这样的实现，CMSIS 规定用通道 0 作为终端来输出调试信息，通道 31 用于操作系统的输出调试(特权模式访问)。在 core_cm3. h 中定义了 ITM_SendChar()函数，因此可通过调用该函数来重写 fputc，以在应用程序中通过 printf 打印调试信息，并可通过 ITMViewer 查看这些调试信息。有了这样的实现，嵌入式软件开发者就可以在不配置串口和使用终端调试软件的情况下输出调试信息，在一定程度上减少了工作量。

(6)安全机制

在嵌入式软件开发过程中，代码的安全性和健壮性一直是开发人员所关注的，因此 CMSIS 在这方面也作出了努力，所有的 CMSIS 代码都基于 MISRA-C2004(MotorIndustrySoftwareReliabilityAssociationfortheCprogramminglanguage)标准。MIRSA-C2004 制定了一系列安全机制用来保证

驱动层软件的安全性，是嵌入式行业都应遵循的标准。对于不符合 MISRA 标准的，编译器会提示错误或警告，这主要取决于开发者所使用的工具链。

（二） stm32f10x.h

属于 CMSIS 的 DPAL，包括了 STM32F10x 系列处理器所有的外设寄存器定义、位定义和不同容量 STM32F10x 的内存映射。

1、可以通过该文件配置如下内容：目标芯片、是否使用库文件、个别特殊的参数，如 HSE 的频率等。

2、定义了数据类型、结构体和所有外设的内存映射

3、访问外设寄存器达到宏

4、中断异常的定义

CMSIS 对异常和中断标识符、中断处理函数名以及中断向量异常号都有严格的要求。异常和中断标识符需加后缀_IRQn，系统异常向量号必须为负值，而设备的中断向量号是从 0 开始递增。

CMSIS 对系统异常处理函数以及普通的中断处理函数名的定义也有所不同。系统异常处理函数名需加后缀_Handler，而普通中断处理函数名则加后缀_IRQHandler。这些异常中断处理函数被定义为 weak 属性，以便在其他的文件中重新实现时不出现重复定义的错误。这些处理函数的地址用来填充中断异常向量表，并在启动代码中给以声明，例如：NMI_Handler、MemManage_Handler、SysTick_Handler、WWDG_IRQHandler 等。

这里特别强调三点：

一是如果选择用外设库来编程则必须在编译器的 PreProcessor Symbols 的 Define 中写入“USE_STDPERIPH_DRIVER, STM32F10X_HD”。这

个在很多的教程中都有些过的。请打开 stm32f10x.h 并查看 8184 行。只有这样才能调用库的函数的 stm32f10x.h;

二是要定义存储器的类型，按照存储容量分为低容量、中容量和高容量;

三是要定义 HSE 的频率。内部定义了 25M 和 8M 两种。

（三）system_stm32f10x.h 和 system_ stm32f10x.c

该文件提供了两个函数和一个全局变量。

- SystemInit()函数用来设置系统时钟（系统时钟源、PLL 倍频因子、AHB/APBx 的预分频及其 Flash）该函数在启动后的复位后中被调用。

- SystemCoreClock 全局变量包括了内核时钟(HCLK),可以用来在程序中设置 SysTick 和配置其他参数; *

- SystemCoreClockUpdate()函数用来更新系统内核时钟，当系统内核时钟变化后必须执行该函数进行更新

当系统复位后，系统时钟利用的内部时钟源的 8MHz,然后通过"startup_stm32f10x_xx.s"调用 SystemInit() 系统主时钟。 *

如果系统启动不成功，则 SystemInit()函数不会生效， HIS 依旧运行于 8MHz

HSE 默认频率为 8MHz。可在 stm32f10x.h 中修改“HSE_VALUE”来改变此值。当 HSE 被用作系统时钟时其直接接入 PLL。所以需要用户必须根据自己的 HSE 实际频率更改该参数

ST 定义的系统初始化函数 SystemInit()，以及 SystemInit()函数的原型可以在 system_stm32f10x.c 中找到。

（四）startup 文件夹

该文件夹下根据 4 种不同编译环境编写了启动的汇编代码，

这些汇编代码分主要为 STM32F10x 完成系统系统所必须的初始化，主要有：初始化堆栈指针 SP、程序指针 PC、设置中断矢量、配置系统时钟，系统启动完毕好后运行主程序。

这些汇编文件时根据不同内存容量的芯片来区分的，使用时需要注意。

第四部分 标准外设库的外设库结构

该部分主要有 inc 和 src 两个文件夹。顾名思义 inc 即 INCLUDE，内部都是头文件，而 src 及 SOURCE，内部都是源文件。

该部分包括了所有外设的操作函数，固态函数库遵从以下命名规则：

PPP 表示任一外设缩写，例如：ADC。系统、源程序文件和头文件命名都以“stm32f10x_”作为开头，例如：stm32f10x_conf.h。

常量仅被应用于一个文件的，定义于该文件中；被应用于多个文件的，在对应头文件中定义。

所有常量都由英文字母大写书写。

寄存器作为常量处理。他们的命名都由英文字母大写书写。在大多数情况下，他们采用如下缩写规范

缩写	外设/单元
ADC	模数转换器
BKP	备份寄存器
CAN	控制器局域网模块
DMA	直接内存存取控制器
EXTI	外部中断事件控制器
FLASH	闪存存储器
GPIO	通用输入输出
I2C	内部集成电路
IWDG	独立看门狗
NVIC	嵌套中断向量列表控制器
PWR	电源/功耗控制
RCC	复位与时钟控制器
RTC	实时时钟
SPI	串行外设接口
SysTick	系统嘀嗒定时器
TIM	通用定时器
TIM1	高级控制定时器
USART	通用同步异步接收发射端
WWDG	窗口看门狗

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。

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。

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。

除上述 stm32f10x_PPP 文件外，还有一个 misc（miscellaneous）文件，该文件主要包括了杂项的函数声明。主要是 NVIC、SYSTick 的操作函数。

该部分具体内容可以参考库自带的 chm 文件。

第五部分 建立一个工程模板

本节内容引用自网络

1 建立目录

1.1 新建工程目录: “RS422”，你也可以根据自己的需要命名此顶层目录;

1.2 在目录“RS422”下新建 “RVMDK”目录，表示采用 ARM MDK 开发环境;

1.3 在“RVMDK”目录下新建目录“V1”，表示软件版本 V1.0，这个好处在于下次将整个目录复制一下改为“V2”，软件版本就是 V2.0 了。

1.4 在“V1”目录下新建 “Libraries”目录;

2.5 在“V1” 目录下新建 “Project” 目录。

1.6 在“Project”目录下新建“OBJ”、“LIST”、“Pro”3 个目录。

1.7 在“V1” 目录下新建“USER”目录。

1.8 在“USER”目录下新建“INC”、“SRC”2 个目录。

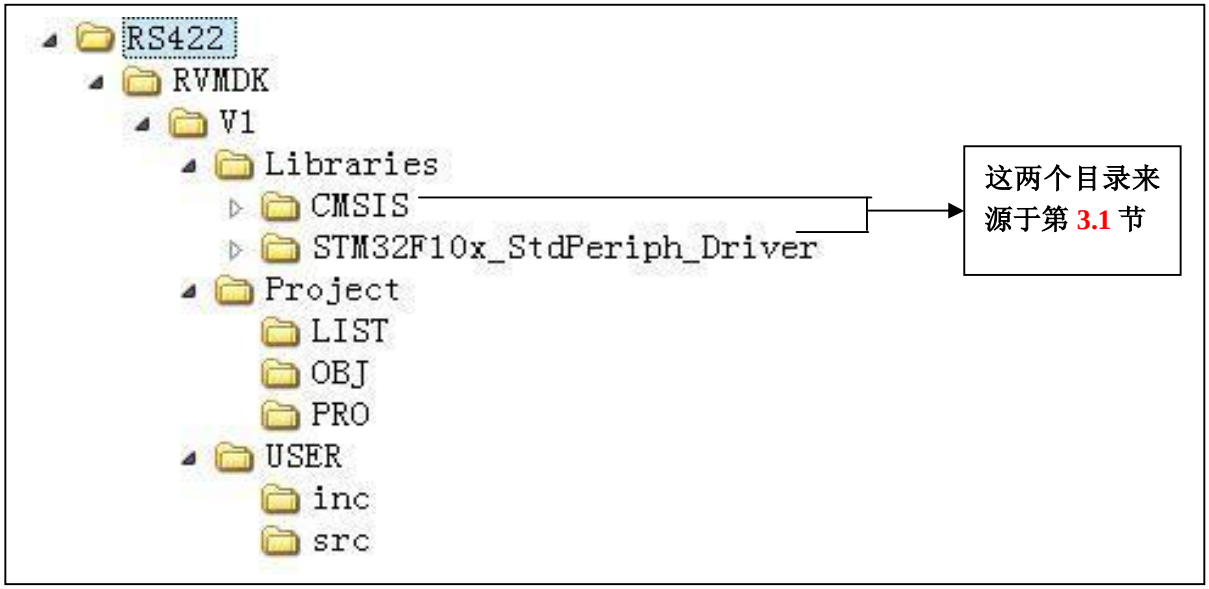


图 1: 目 录 结 构 图

2 拷贝文件 （已经写的很详细了，就不上图了）

2.1 将固件库目录“STM32F10X_StdPeriph_lib_V3.5.0” è “Libraries”下的所有目录拷贝到工程目录“RS422” è“RVMDK” è “V1” è “Libraries”目录下。

2.2 将固件库目录 “STM32F10X_StdPeriph_lib_V3.5.0” è “Project” è “STM32F10x_StdPeriph_Examples” è “GPIO” è “IOToggle”目录下的“stm32f10x_it.c”、“system_stm32f10x.c”拷贝到工程目录“RS422” è“RVMDK” è“V1” è“USER” è“SRC”目录下，在此“SRC”目录下新建“main.c”文件，“main.c”先“神马”也不写。

2.3 将固件库目录“STM32F10X_StdPeriph_lib_V3.5.0” è“Project” è“STM32F10x_StdPeriph_Examples” è“GPIO” è“IOToggle”目录下的“stm32f10x_conf.h”、“stm32f10x_it.h” 这 2 个文件拷贝到工程目录 “RS422” è“RVMDK” è “V1” è“USER” è“INC”目录下。

3 建立工程:

3.1 我用的是 ARM 的 MDK4.14 开发环境，运行“Keil uVision4” ；

3.2 点击主菜单栏 “Project” è“New uVision Project”，选择在工程目录的“V1” è“Project” è“Pro”目录下命名新建工程为“RS422_MODULE.uvproj”（当然也可以命名为你自己需要的工程名）;

3.3 接下来出现 CPU 选择窗口，选择 CPU 为 “STMicroelectronics” è“STM32F103ZE” （这个大家根据自己的需要选择），点击 “OK” 按钮;

3.4 接下来出现“Copy STM32 Startup Code to Project Folder and ADD File to Project ?”提示时选择“否”（在后面的步骤中会根据 CPU 选择启动文件的，这里不用选择），完成工程建立。

4 工程管理

上一节新建的工程还是空空的，这一步要将它实例化。

4.1 在“Project”窗口中用鼠标左键点击“Target 1”，再点击右键弹出菜单选择菜单中的 “Manage Components”子菜单（见图 2），出现“Components，Environment and Books”窗口（见图 3）。

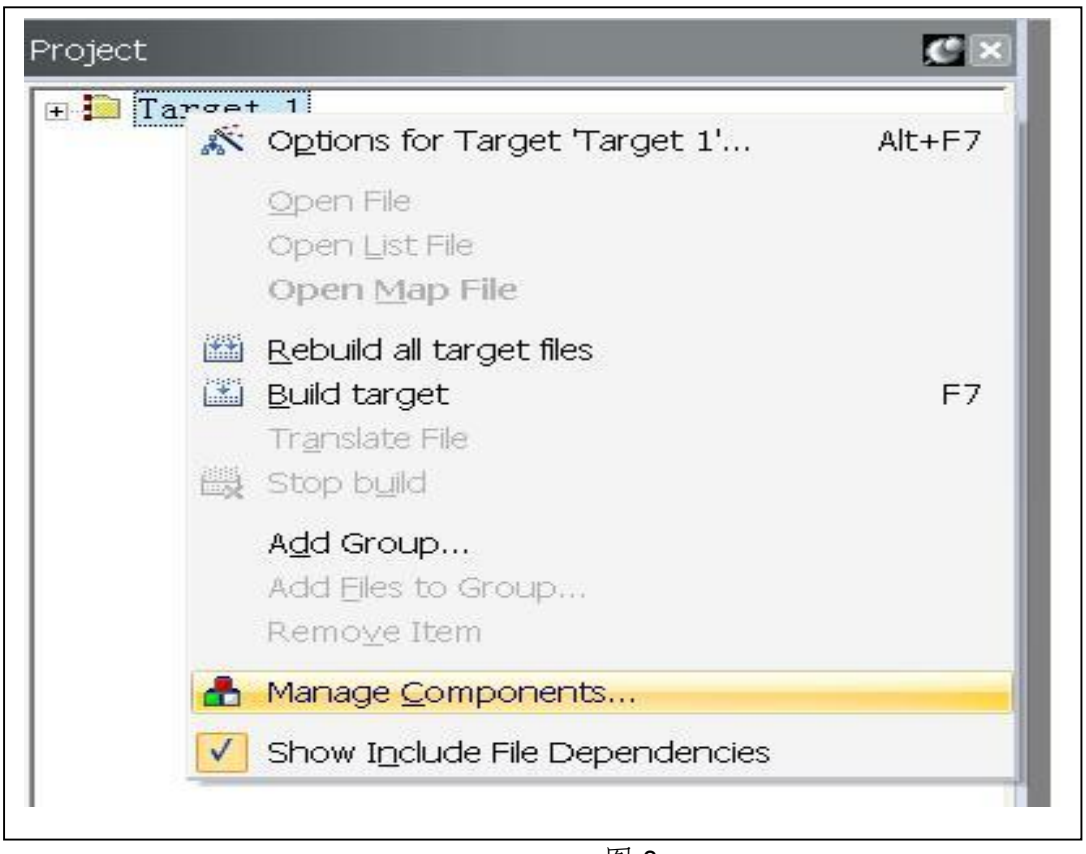


图 2

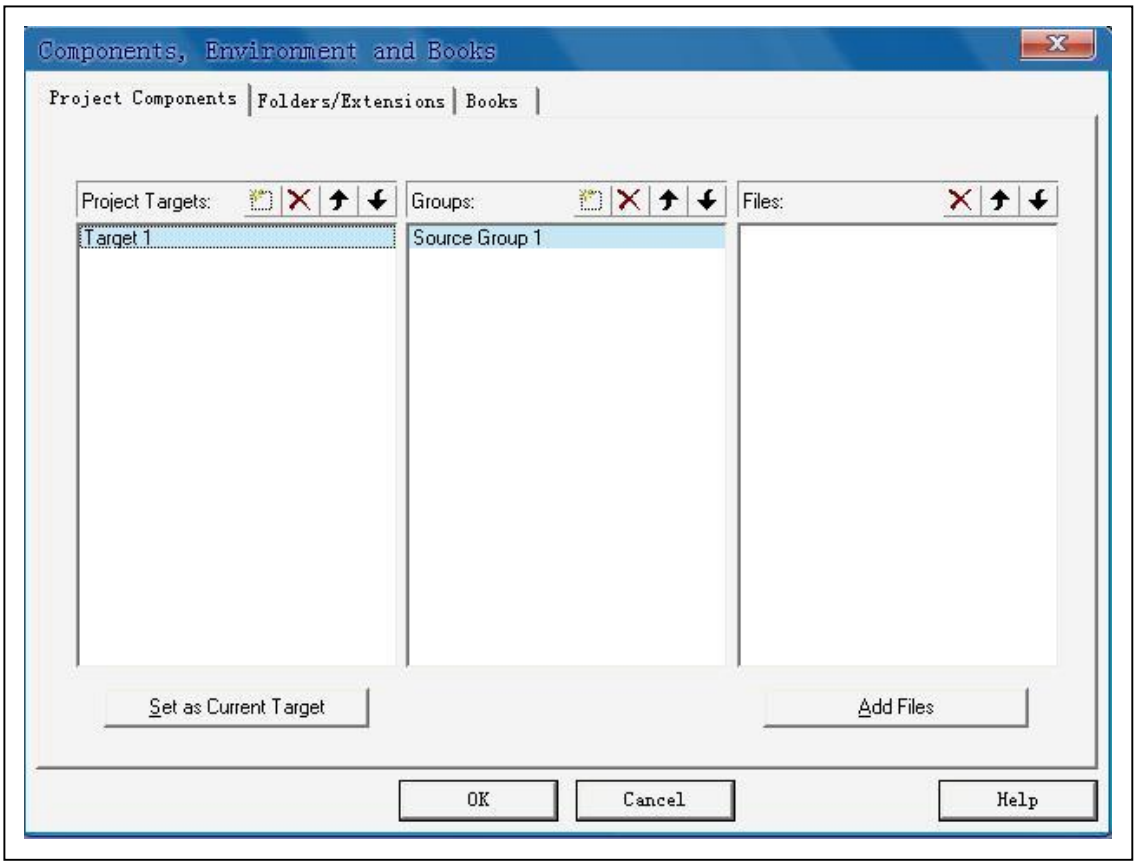


图 3

4.2 用鼠标双击“Project Targets”栏中的“Target 1”将“Target 1”改为“RS422_MODULE”（见图 4），在中间的“Groups”栏中添加“USER”、“STM32_LIB”、“MDK_STARTUP”、“CMSIS”4 个条目（见图 5）。

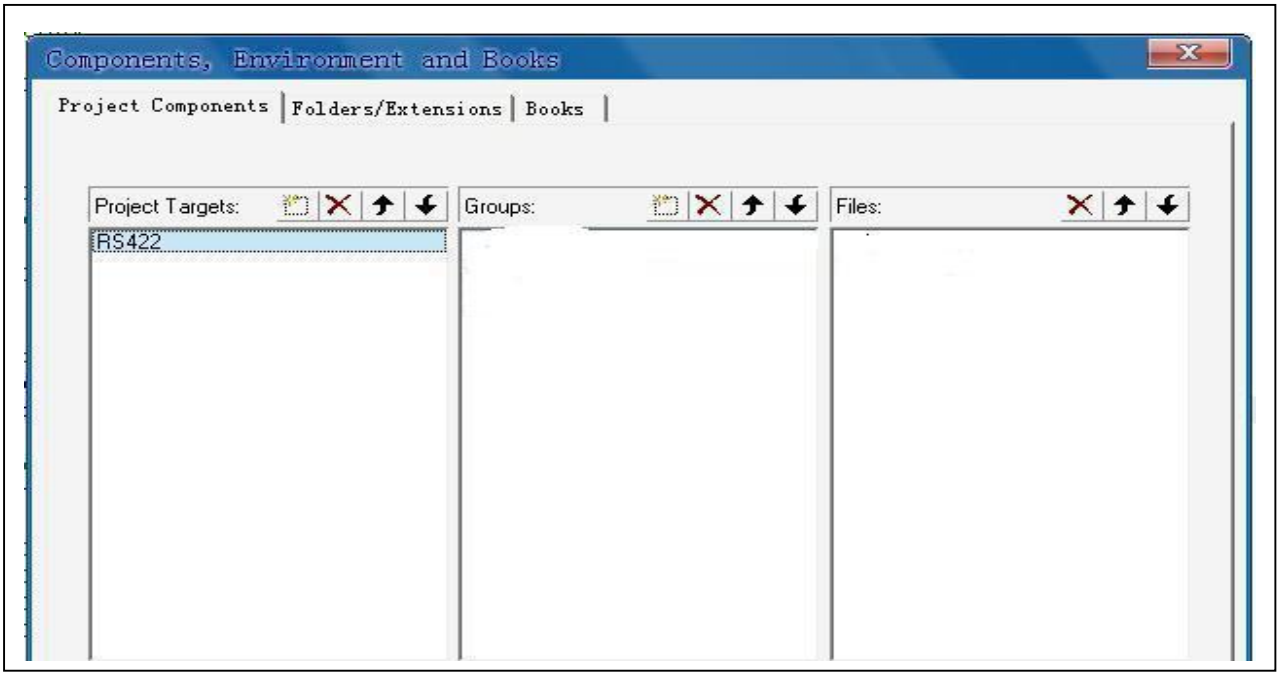

Components, Environment and Books
Project Components
Folders/Extensions
Books
Project Targets:
Groups:
Files:
RS422

图 4

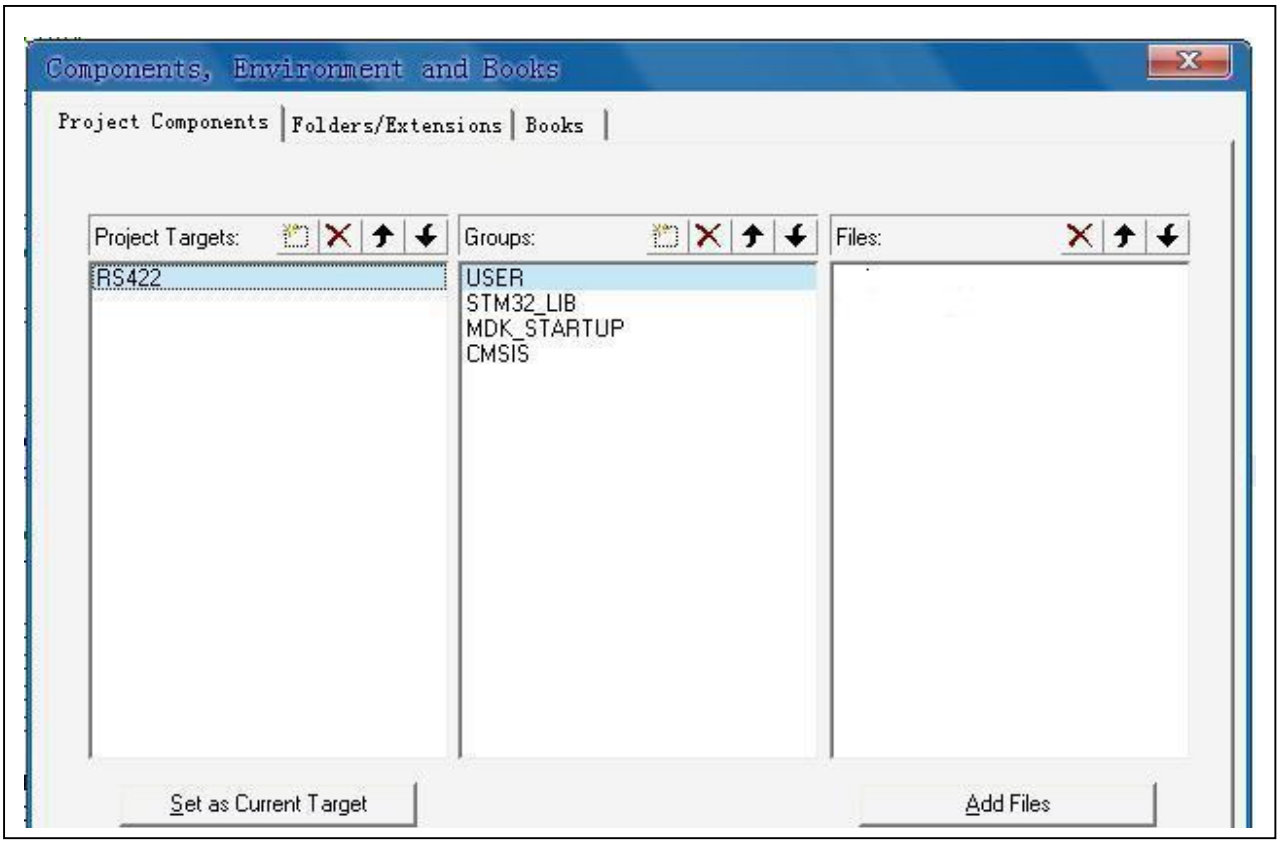

Components, Environment and Books
Project Components
Folders/Extensions
Books
Project Targets:
Groups:
Files:
RS422
USER
STM32_LIB
MDK_STARTUP
CMSIS
Set as Current Target
Add Files

图 5

4.3 在“USER”条目的“Files”栏中添加目录“V1” è “USER” è “SRC”下的“main.c”、“stm3210x_it.c”2 个文件，通过窗口的“Add Files”按钮可以添加文件 （见图 6）。

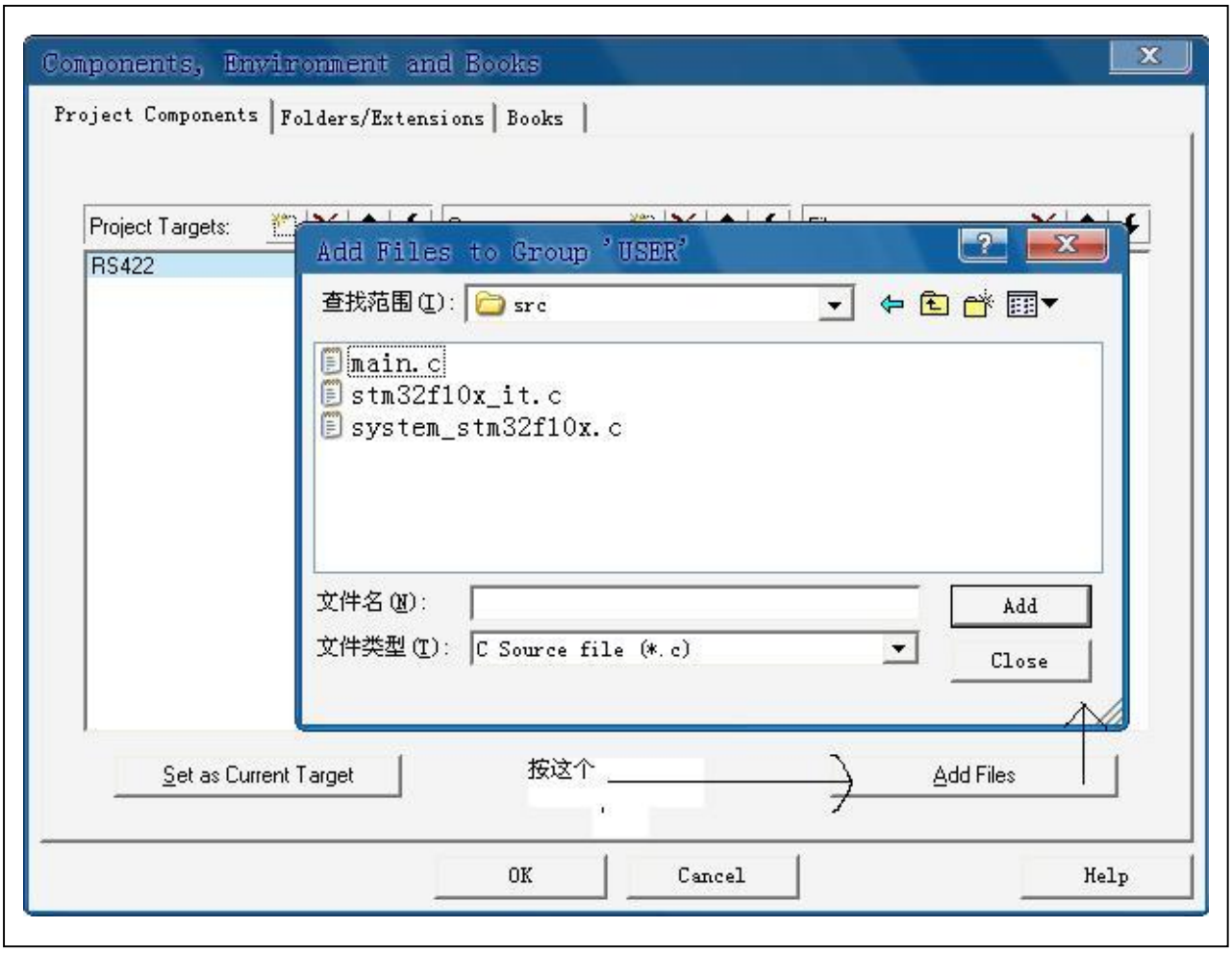


图 6

4.4 在 “STM32_LIB” 条目的 “Files” 栏中添加目录“V1” è “Libraries” è “STM32F10x_StdPeriph_Driver” è “SRC”目录下的“stm3210x_misc.c”、“stm3210x_rcc.c”、“stm3210x_gpio.c”、“stm3210x_usart.c”4 个文件（见图 7），其中“stm3210x_misc.c”、“stm3210x_rcc.c” 这 2 个文件是必须的（我自己的理解）。 因为我的程序要用到“GPIO”和“串口”，所以又添加了“stm3210x_gpio.c”、

“stm3210x_usart.c”这 2 个文件，大家在开发中如果用到 STM32 的其他功能，再添加相应的接口库文件就可以了。

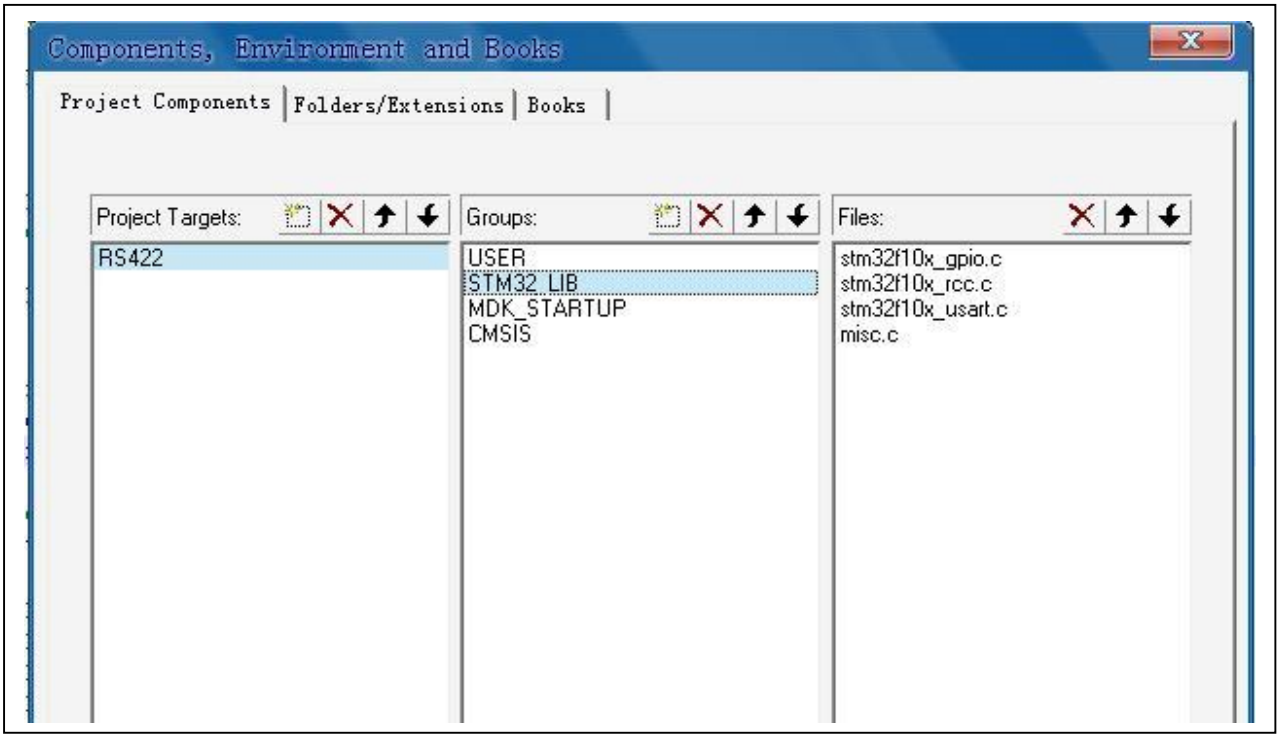


图 7

4.5 在“MDK_STARTUP”条目的“Files” 栏中添加目录“V1” **è** “Libraries” **è** “CMSIS” **è** “CM3” **è** “DeviceSupport” **è** “ST” **è** “STM32F10x” **è** “startup” **è** “arm”（我的神啊：ST 的目录够深的）下的 “startup_stm3210x_hd.s”这个文件（见图 8），因为我选用的 CPU 是“STM32103ZE” 是高容量 FLASH 的 CPU 所以选择这个文件。

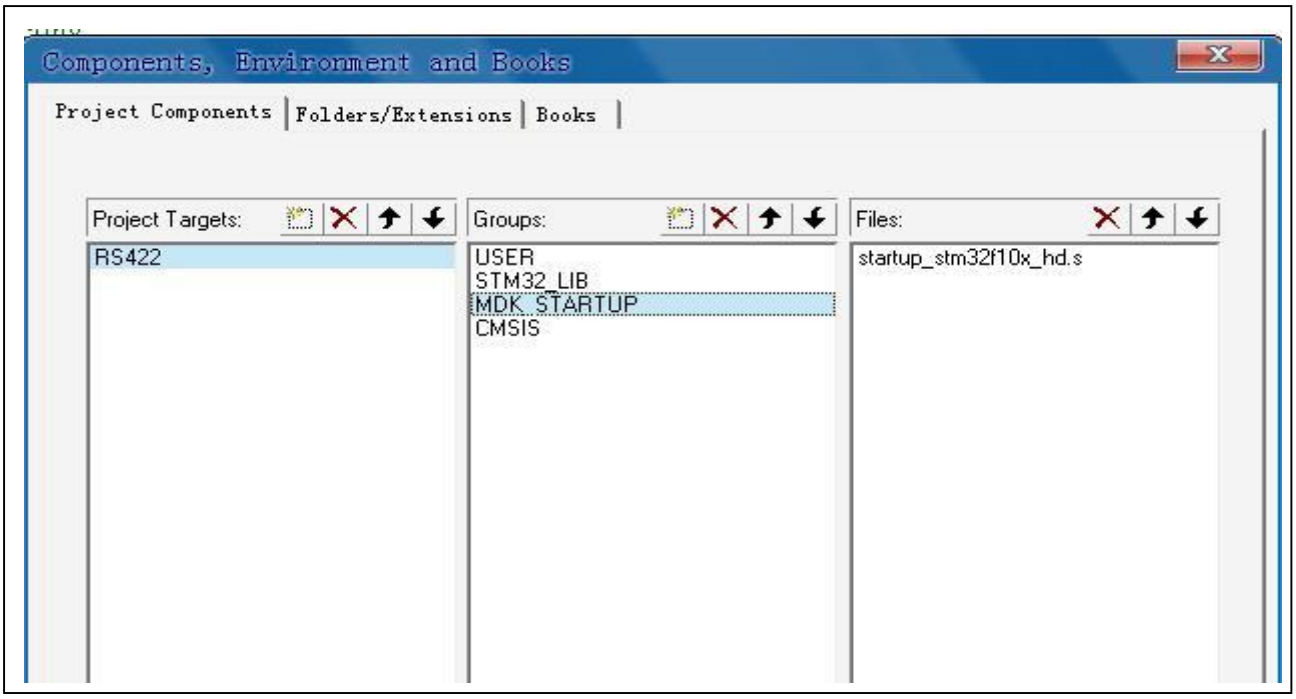


图 8

4.6 在 “CMSIS” 条目的 “Files” 栏中添加目录 “V1” **è** “Libraries” **è** “CMSIS” **è** “CM3” **è** “CoreSupport” 下的 “core_cm3.c” 和目录 “V1” **è** “USER” **è** “SRC” 下的 “system_stm32f10x.c” 这 2 个文件（见图 9）。

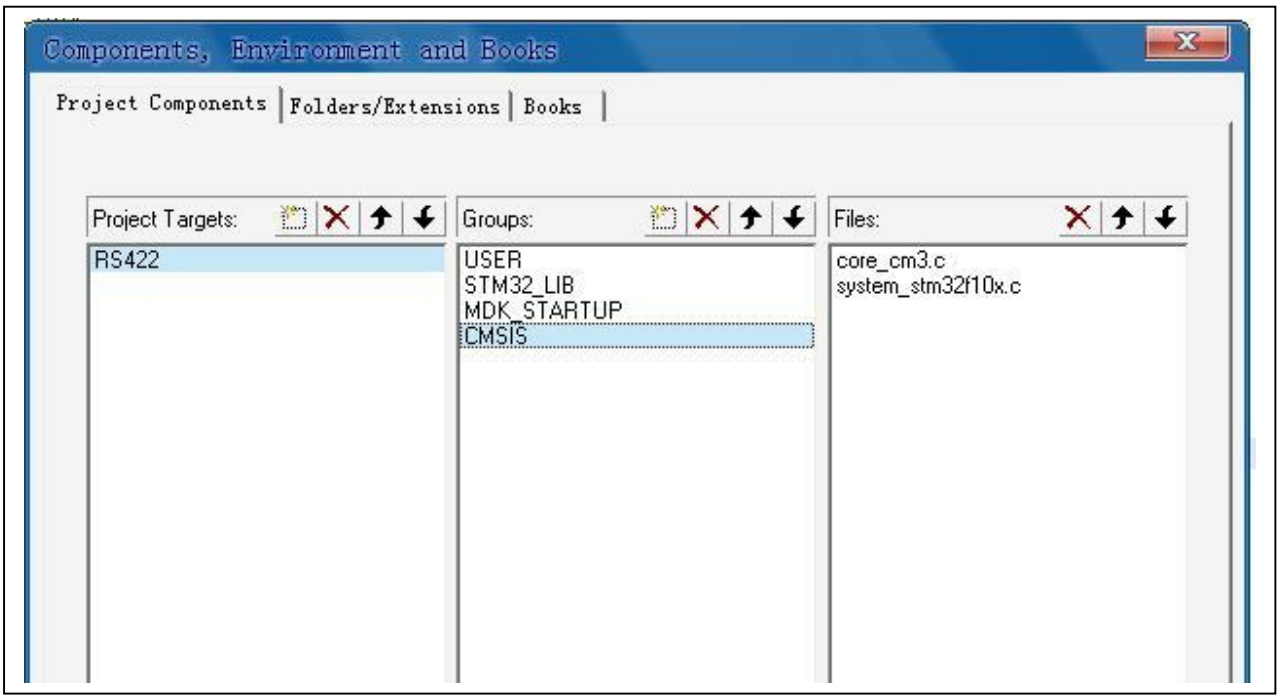


图 9

4.7 退出“Keil uVision4”开发环境，将工程目录“RS422”去掉文件夹的只读属性，并“应用到子目录和所有文件”，这是因为 STM32 固件库下载下来是只读的，无法修改库中的文件。去掉只读属性后再进入“Keil u Vision4”，然后打开工程 “RS422_MODULE.uvproj”。

5 工程设置

5.1 在“Project”窗口中用鼠标左键点击最顶层的“RS422”，再点击右键弹出菜单选择菜单中的 “Options for Target ‘RS422’...”子菜单（见图 10），出现“Options for Target ‘RS422’”窗口（见图 11），可以按图 11 进行设置。

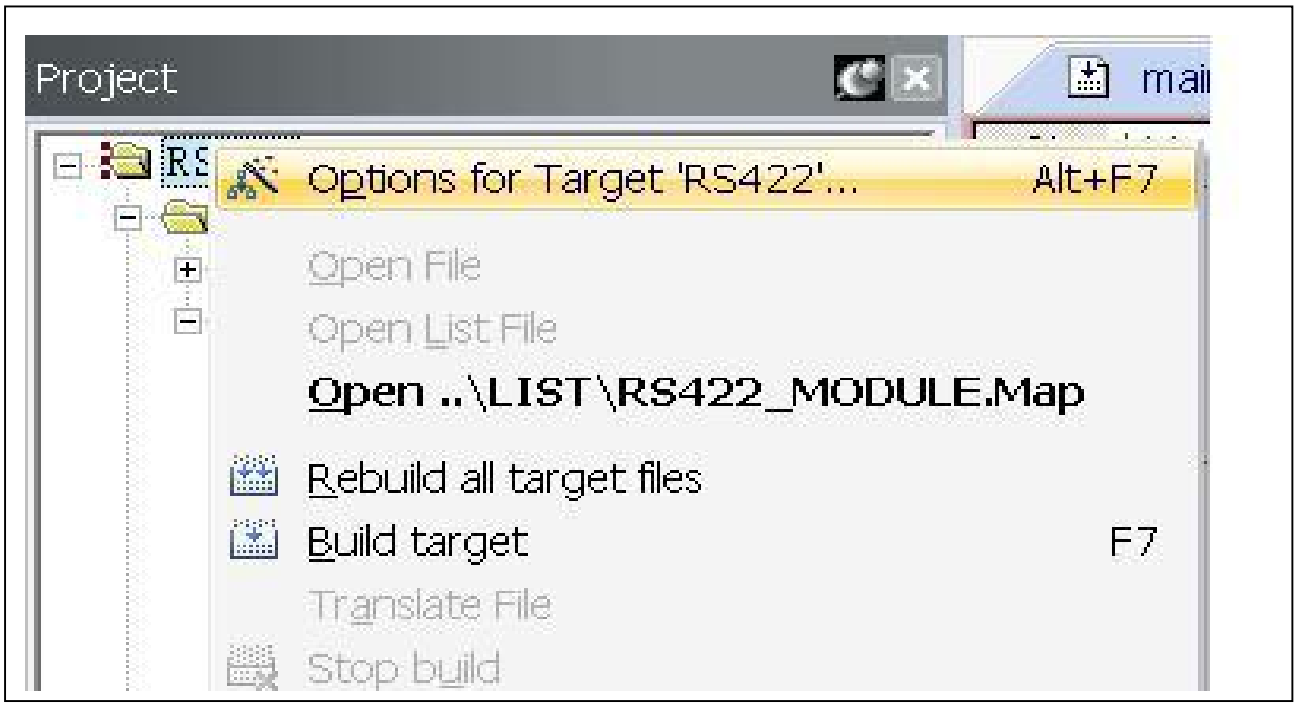

Project
Options for Target 'RS422'... Alt+F7
Open File
Open List File
Open ..\LIST\RS422_MODULE.Map
Rebuild all target files
Build target F7
Translate File
Stop build

图 10

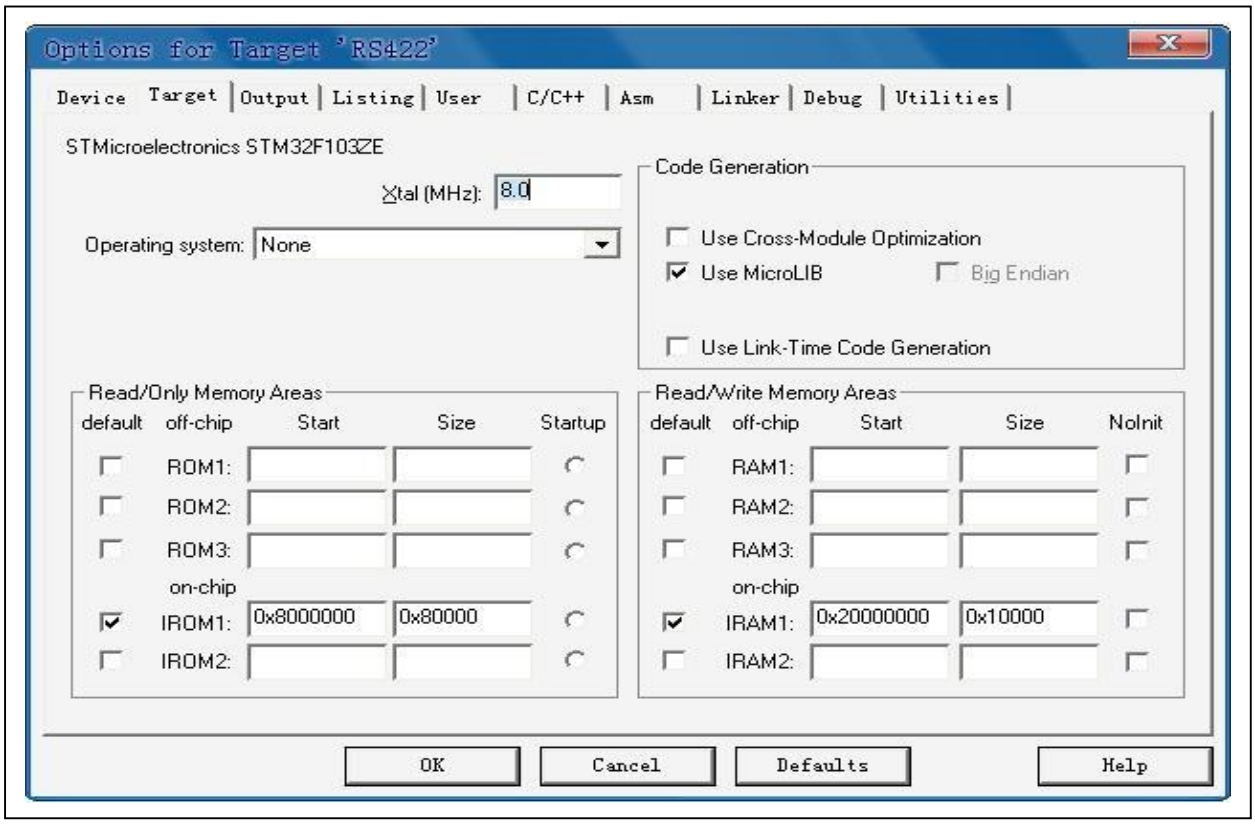

Options for Target 'RS422'
Device Target Output Listing User C/C++ Asm Linker Debug Utilities
STMicroelectronics STM32F103ZE
Xtal (MHz): 8.0
Operating system: None
Code Generation
Use Cross-Module Optimization
Use MicroLIB
Big Endian
Use Link-Time Code Generation
Read/Only Memory Areas
default off-chip Start Size Startup
ROM1:
ROM2:
ROM3:
on-chip
IROM1: 0x8000000 0x80000
IROM2:
Read/Write Memory Areas
default off-chip Start Size NoInit
RAM1:
RAM2:
RAM3:
on-chip
IRAM1: 0x20000000 0x10000
IRAM2:
OK
Cancel
Defaults
Help

图 11

6.2 选择顶层的“Output ”，进入“Output”设置页，点击下面的“Select Folder for Objects...”按钮，选择目录“V1” è “Project” è “OBJ”为目标文件目录，选中“Create HEX File” （见图 12）。

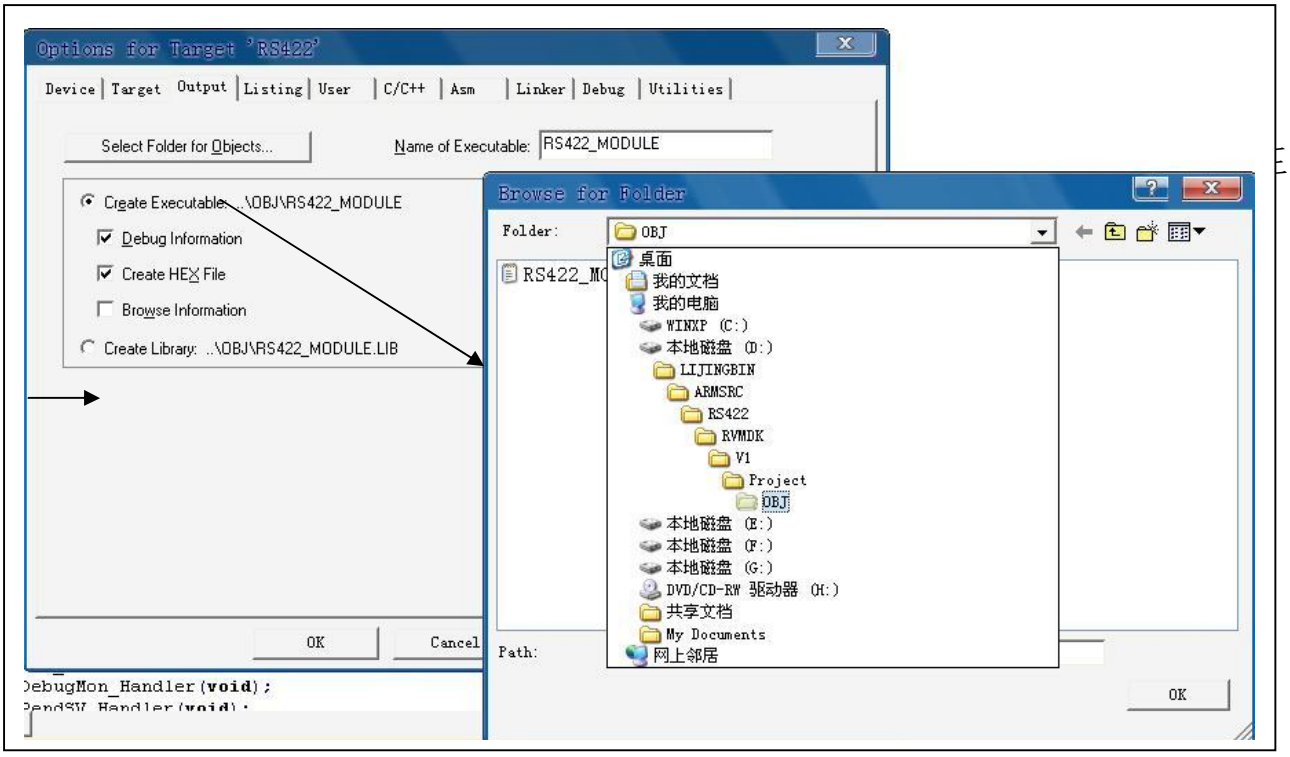


图 12

6.3 选择顶层的“Listing”页，进入“Listing”设置页，点击下面的“Select Folder for Listings...”按钮，选择目录“V1” è “Project” è “LIST”目录为 list 文件生成目录（见图 13）。

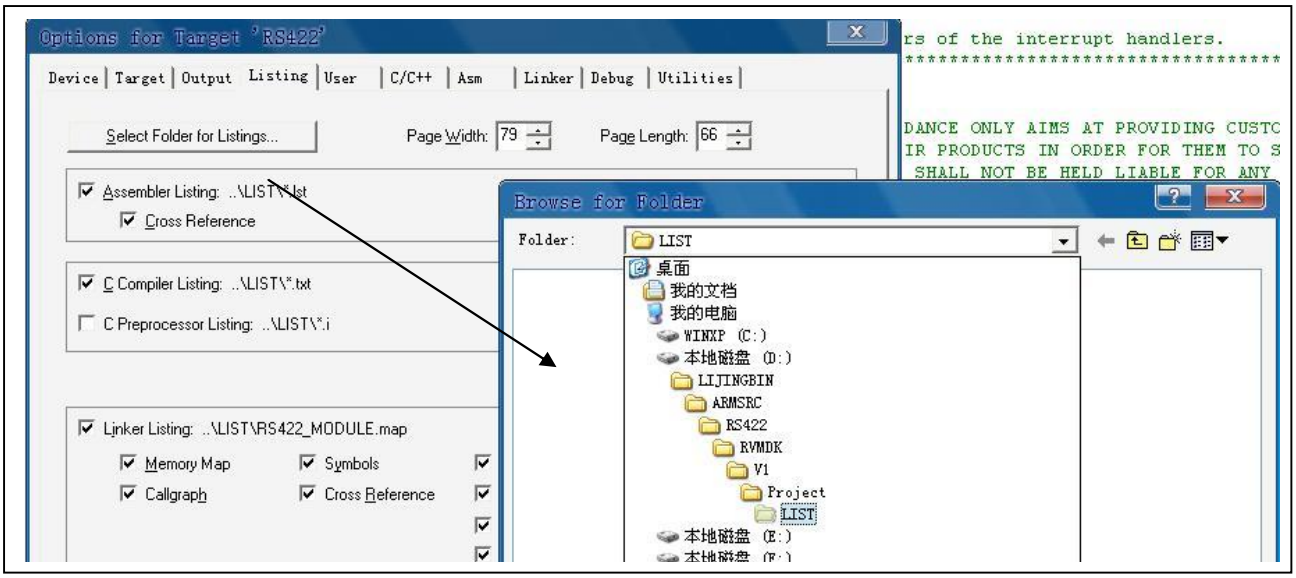


图 13

5.4 这是关键的一步，选择顶层的“C/C++”页，进入“C/C++”设置页，在“Preprocessor Symbols”的“Define: ”文本框中填入“STM32F10X_HD,USE_STDPERIPH_DRIVER”。如果不填编译会报错，大家可以试一下，这是因为在固件库“stm32f10x.h”中有如下的片断（见图 14），需要根据你的 CPU 来去掉相应行的注释，我选的是 STM32 系列高容量的 CPU，所以需要去掉“stm32f10x.h”中第“0054”行“/*define STM32F10X_HD */”这一行的注释符，但是不能换一个 CPU 就改一次注释吧，老去改文件自己哪天不定就改的迷糊了，幸好编译器提供了这个功能，只要按本步骤的方法加入“STM32F10X_HD”就可以了，省得自己换了 CPU 还得把文件找出来更改。加入“USE_STDPERIPH_DRIVER”是同样的道理，这个出现在“stm32f10x.h”的第“8280”行（见图 15），但是固件库中并未定义“USE_STDPERIPH_DRIVER”，所以也需要用这种办法灵活的加入，否则编译时就不会链接“stm32f10x_conf.h”，这个文件可是大有用处，打开看看吧，这里就不描述了。

图 15

5.5 引用固件库文件所在的目录也在顶层的“C/C++”页中进行设置，上面写的啰里啰唆的，已经太多了，所以写在这里吧。如果现在编译程序，会报错的，看看出错提示，有一些库文件跑到开发工具 Keil 的安装目录下链接去了，这是因为没有设置 STM32 固件库的目录，编译器就默认到“Keil”根目录下的某某目录找去了。我们工程用到的 STM32 3.4.0 固件库的相关文件在上面第 3 节中已经直接拷贝到工程里了，这个是因为 STM32 库升级到 3.4 已经经历了好多个版本了，而安装完 KEIL 后并不是最新的 3.4 库，不能说哪天系统 OVER 了，装一次 KEIL 就升级一次库吧，干脆就把库文件直接放在工程中，即使将工程拷贝到其他机器上也可以编译，不会因为没有库文件而报错，唉！又啰嗦了这么多。在窗口的“Include Paths”旁边

的文本框后有一个按钮，点击调出“Folder Setup”窗口。这里要添加 4 个目录（见图 16），因为这 4 个目录里都有“.h ”文件:

“ RS422\RVMDK\V1\Libraries\STM32F10x_StdPeriph_Driver\inc” ;

“ RS422\RVMDK\V1\Libraries\CMSIS\CM3\CoreSupport ” ;

“ RS422\RVMDK\V1\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x ” ;

“ RS422\RVMDK\V1\USER\inc ” ;

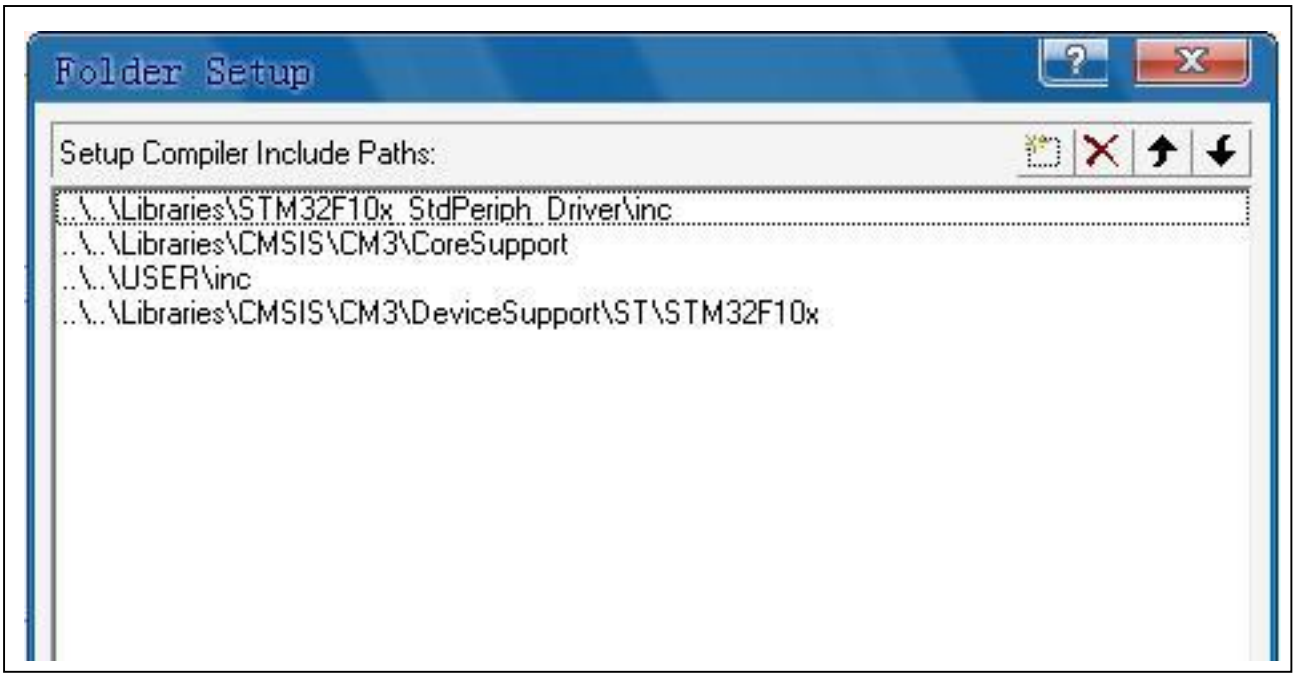


图 16

5.6 其他“C/C++”按图 17 设置默认就可以了。

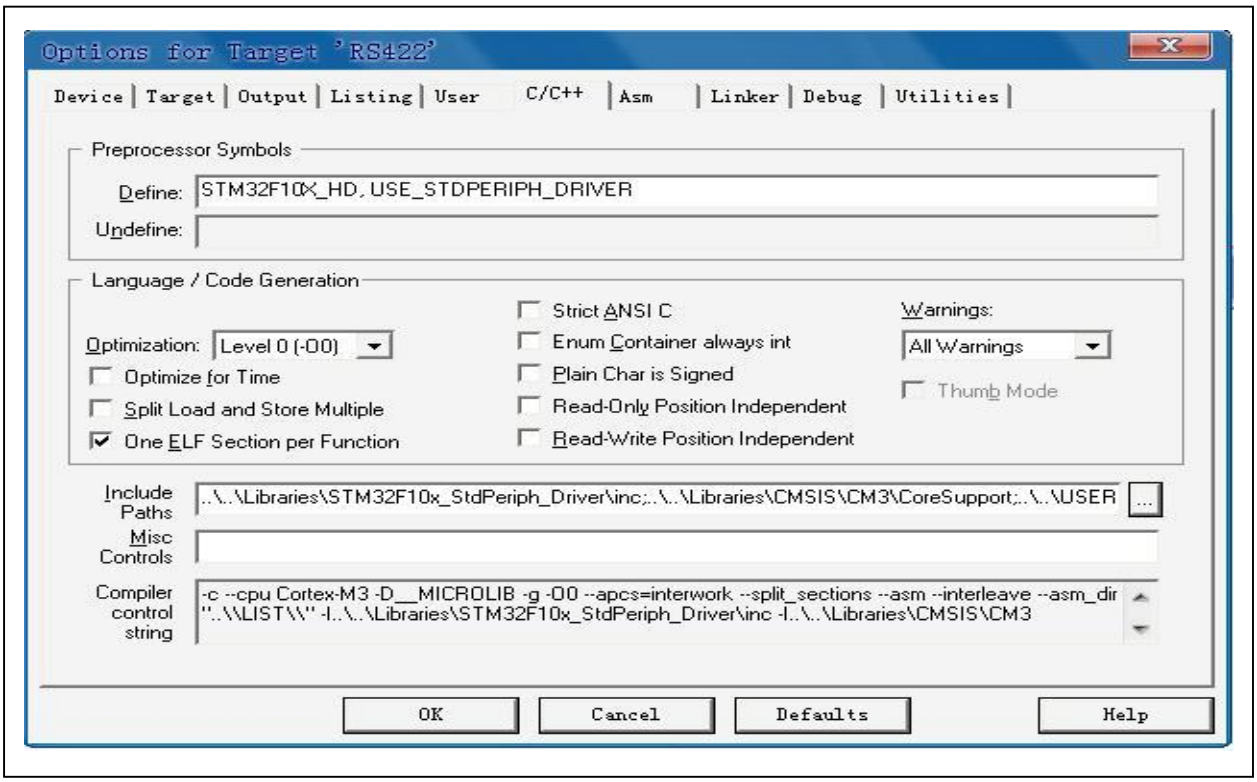


图 17

6. main.c 文件

因为这是一个没有任何功能的工程框架，所以 main.c 可以这么写:

```
#include "stm32f10x.h"
main ()
{
  while (1)
   {
      ;
   }
}
```

编译程序，没有警告和错误，框架就算生成了，想添加自己的代码就可以以后添加了，自己的代码放在工程目录“RS422” è“RVMDK” è“V1” è“USER”下的“INC”或“SRC”目录下，“INC”下放你的“.h”文件，“SRC”目录下放你的“.C”文件 。

以上是本人对 STM32 工程框架的实例实现，如果路过的有更好的，希望可以交流。

BY:AIRWOLF

E-mail:Airwolf0992@163.com

QQ:67792012