第二章 IAR 的使用和 STM32 固件库模板的建立

1 IAR 工程的建立

我们在第一章中已经讲过 IAR 的安装，安装完成之后，可以在桌面看到 IAR 的快捷方式图标“IAR Embedded Workbench”。我们也可以在开始->所有程序->“IAR Embedded Workbench”里面找到我们的 IAR IDE 应用程序，如图 1 所示。

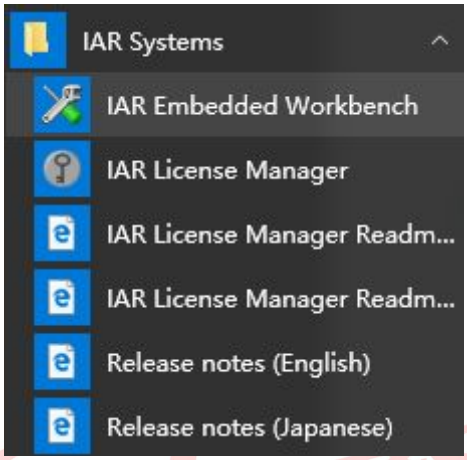


图 1 IAR 应用程序目录

打开 IAR，就可以直接进入 IAR 应用里面，如图 2 所示。

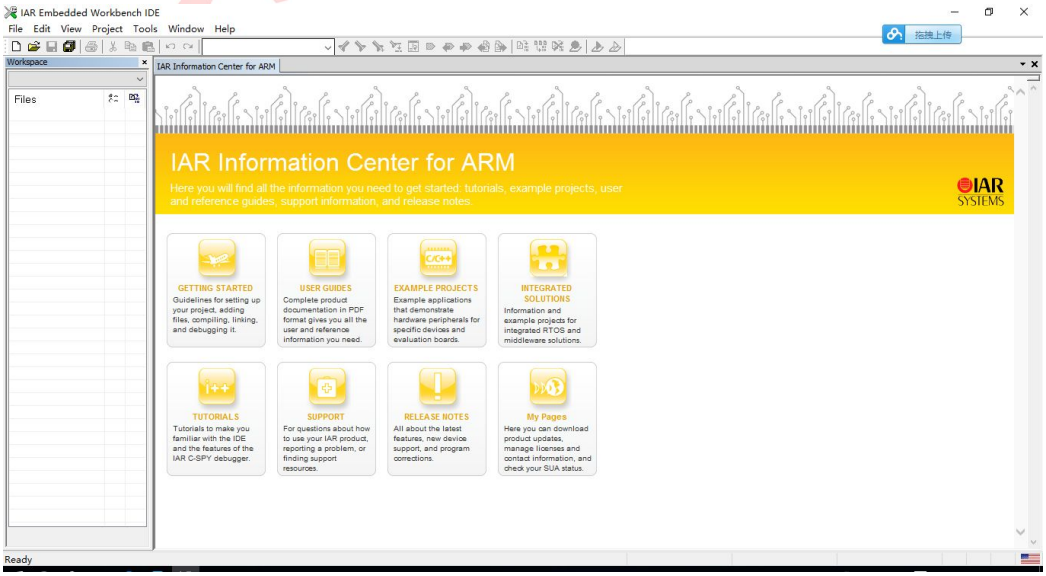


图 2 IAR 的初始化界面

由于 IAR 的界面比较简单，因此我们就不一一分析每个菜单的作用，以后多用，我们自然而然可以很好的掌握，这里我们具体讲解如何建立工作区以及工程。

1.1 工作区的建立

新建 IAR 工作空间，首先是菜单 File 里选择 Open 再选择 Workspace，如图 3 中红圈所示。

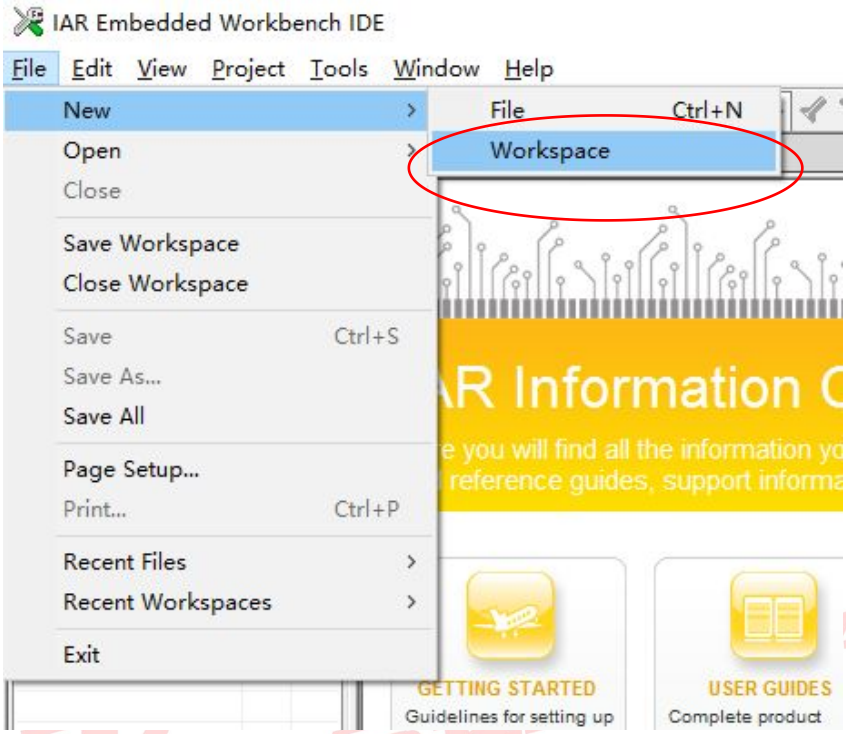


图 3 IAR 建立工作区

接着就会看到一片空白。这时就是新的“办公区”了，如图 4 所示。有了工作空间就可以建立工程了。

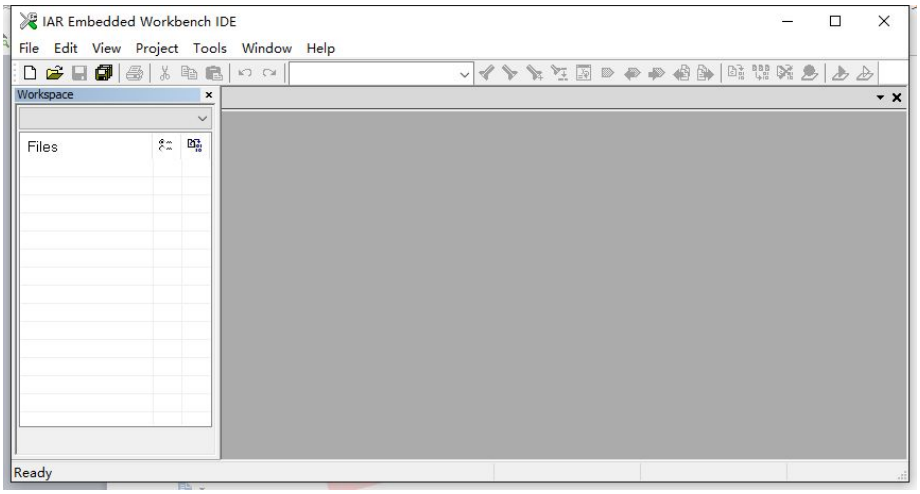


图 4 IAR 工作区界面

1.2 工程建立

首先，点击“菜单” -> “Project-Create New Project”就可以进入工程建立的页面，如图 5 所示。之后就会出现一个建立工程的界面，如图 6 所示。

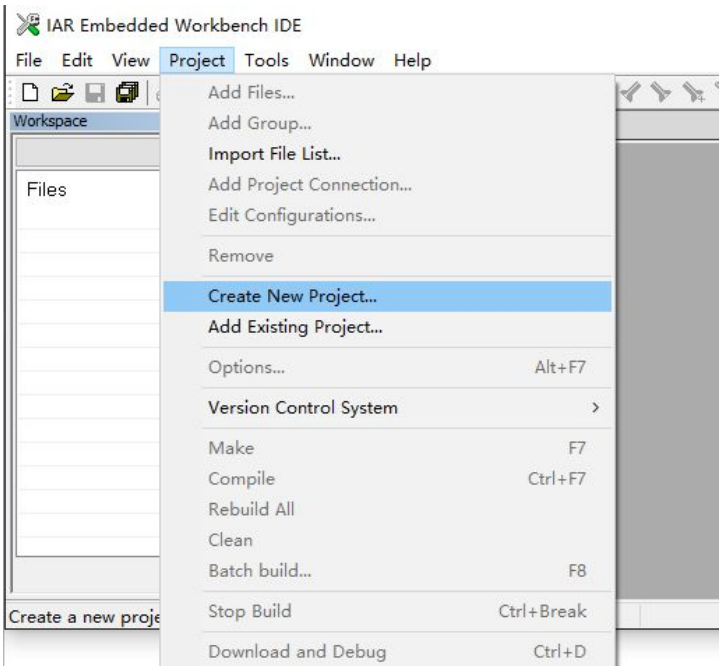


图 5 建立工程

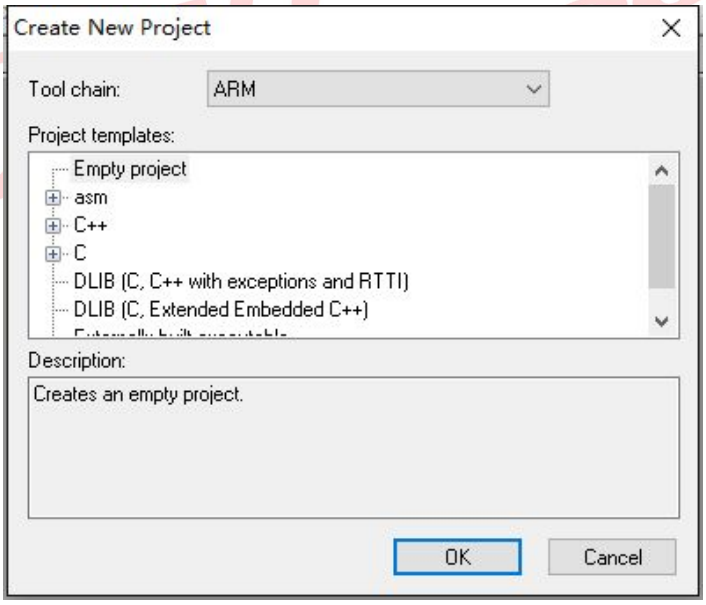


图 6 工程建立页面

图 6 中有个“Tool Chain:ARM”这里选的是芯片的类型，IAR 有很多其他专用的开发集成环境，比如 IAR for ARM，IAR for 430，IAR for 8051 等等。这里只安装了 ARM 的编译器，因此此选项只有 ARM。它的下面有个“Project templates”（工程模板），“Project templates”里面有四个选项，第一个是“Empty project”，

表示建立一个空白的工程；第二个是“asm”，表示一个汇编程序开发的工程；第三个是“C++”，表示建立一个 C++开发工程；第四个是“C”，表示建立一个 C 语言开发的工程的，因为我们需要建立 STM32 固件库的开发环境，因此在这里只需建立一个空白的工程就可以了。单击以选择图 7 中红圈里面的内容。

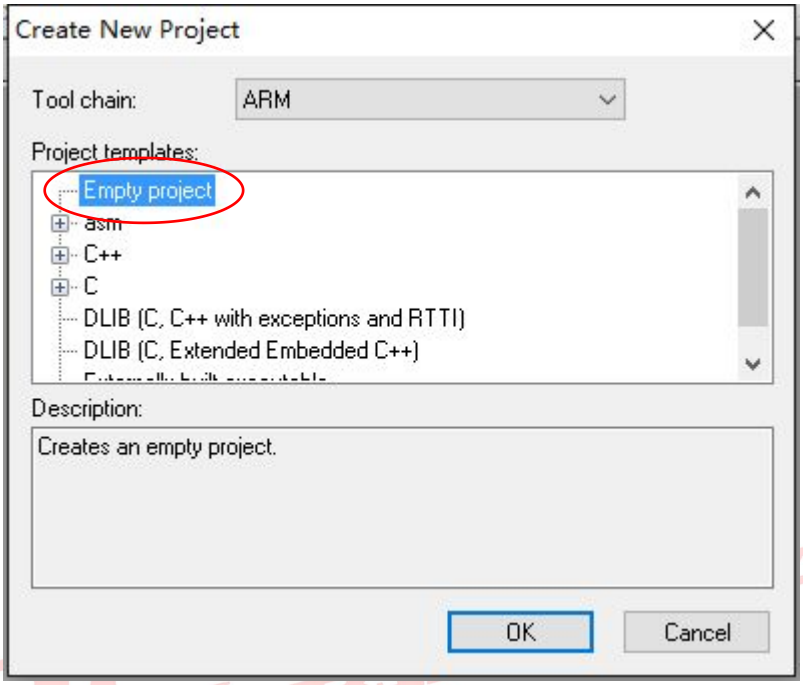


图 7 Empty project 的建立

接着会弹出一个保存的界面，如图 8 所示，由于我们要建立一个 STM32 固件库的标准模板，因此我们把它取名为“STM32F10X Template”，并且保存在桌面建立的“STM32F10X Template”文件夹里面。选好路径之后，单击“保存”即可完成工程的建立。

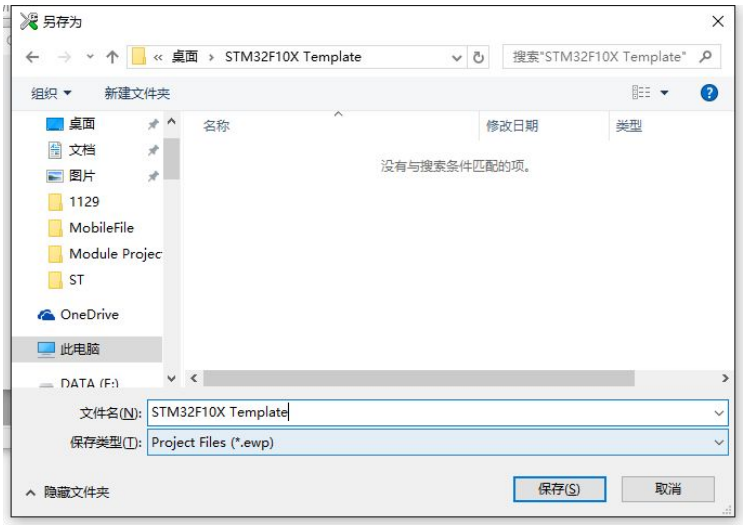


图 8 保存工程

在这里，需要另外注意一点，一个工作区里面可以按照上述 1.2 的步骤，再建立多个工程。

建立好的工程界面如图 9 所示，在图 9 中，我们也标注了工程界面每个区域的功能。

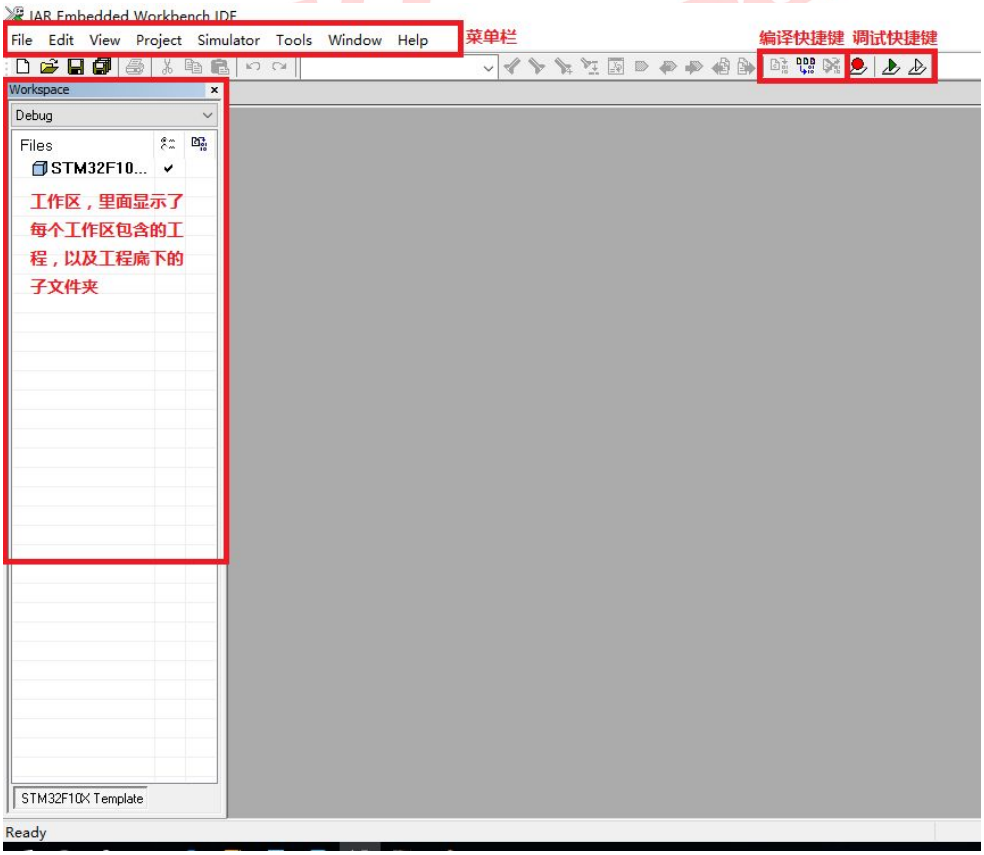


图 8 工程界面的区域分划

最后，我们需要保存工作区，选择“File”->“Save Workspace”就可以保存工程，如图 9 所示。之后会出现保存路径的对话框，我们还是将此工作区保存在桌面建立的“STM32F10X Template”文件夹里面，如图 10 所示。

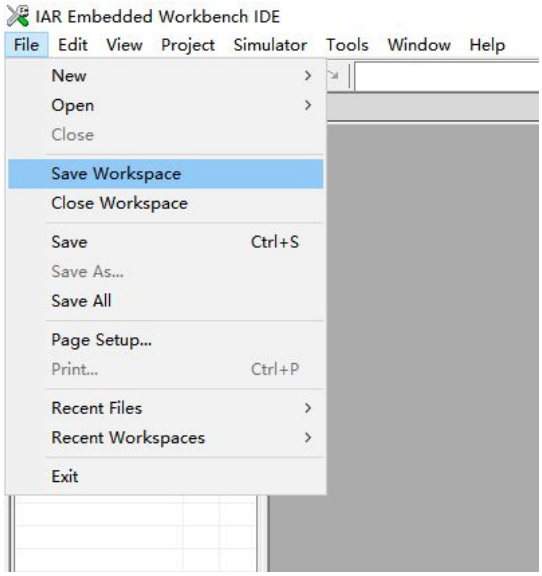


图 9 保存工作区

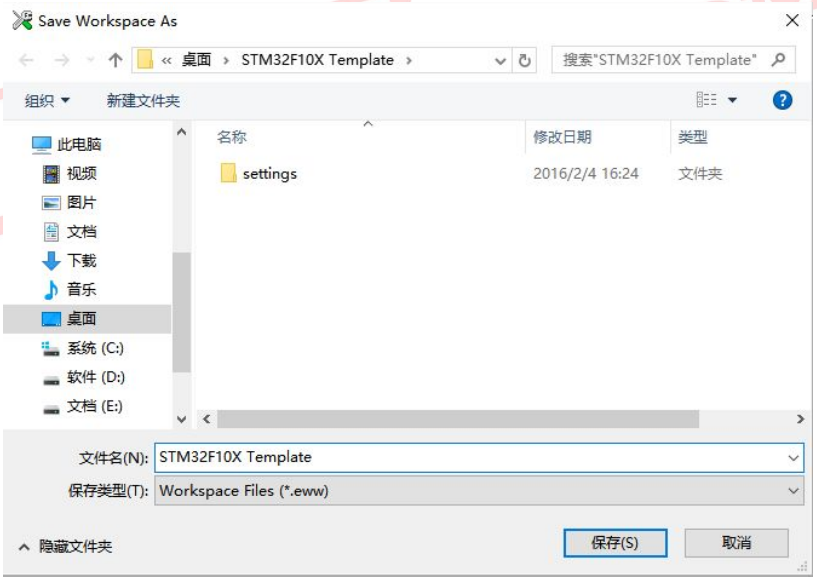


图 10 工作区保存路径

2 STM32 标准固件库的介绍

固件库，我们可以近似的把它认知为硬件的系统函数库，它是 ST 官方发布的基于 C 语言和汇编代码构成的 C 库，我们可以直接调用它提供给我们的库函数，进行 STM32 单片机的硬件配置和操作，目前针对 STM32F103 系列的固件库为“STM32F10x_StdPeriph_Lib_V3.5.0”，ST 的官方网站上能下载到，也可以直接去我们的百度云共享上面下载，下载链接 http://pan.baidu.com/s/1dDKTn4P，或是去我们我要自学网视频下方获取资料处下载。

下载好之后解压，我们便可以看到整个文件的目录树，如图 11 所示。

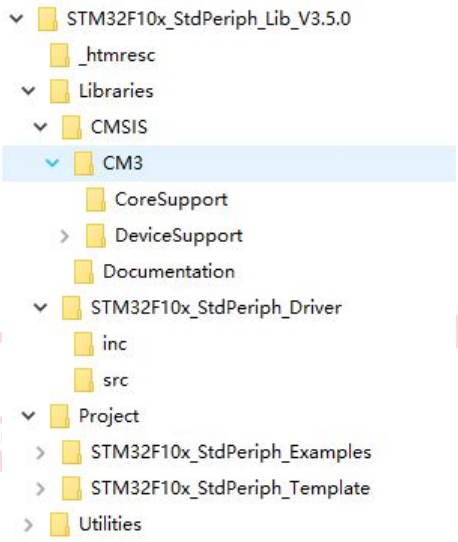


图 11 STM32 标准固件库目录树

其中各个文件夹的具体包含内容如下：

“_htmresc”文件夹存放了 STM32 固件库的 LOGO 图片；

“Libraries”文件夹是 STM32 固件库的主要库函数存放地，里面包含了两个文件夹：

“CMSIS”里面包含了一个“CM3”文件夹，“CM3”里面存放了一些和 ARM Cortex-M3 有关的启动文件；

“STM32F10x_StdPeriph_Driver”文件夹里面就是 STM32 外设的驱动文件，例如 GPIO，CAN，USART 等等；

“Project”文件夹里面包含了两个文件夹，这里面存放的是一些利用标准的固件库建立起来的模板和例程：

“STM32F10x_StdPeriph_Examples”里面存放了 STM32 外设固件的标准例程；

“STM32F10x_StdPeriph_Template”里面存放了各个不同 IDE 的模板。

“Utilities”里面包含了用于 STM3210B-EVAL 和 STM3210E-EVAL 评估板的专用驱动。

详细解说，推荐大家一篇博客，连接如下：

“http://www.cnblogs.com/emouse/archive/2011/11/29/2268441.html”。

3 STM32 固件库模板的建立

虽然 STM32 固件库里面提供了很多例程给我们，但是都是基于 ST 官方发行的一些评估板的，在实际应用当中，我们可能由于某些原因，不可能和官方评估板做的一模一样，因此需要在 STM32 标准例程的基础上，进行移植和修改。

3.1 移植前准备

移植前，我们首先需要按照第 1 节中讲解的过程，建立一个 IAR 的工程，然后把“STM32F10x_StdPeriph_Lib_V3.5.0.rar”固件库解压，得到“STM32F10x_StdPeriph_Lib_V3.5.0”文件夹

3.1.1 建立根目录文件夹管理代码

我们需要层叠的文件压，便于源码的管理，因此我们需要在“STM32F10X Template”根文件夹里面建立“USER”，“CMSIS”，“FWLIB”几个文件夹，如图 12 所示。其中，“USER”文件夹，主要存放用户应用程序和中断文件，如 main.c 等等；“CMSIS”主要存放内核和系统相关的代码；“FWLIB”主要存放 STM32 标准外设的驱动代码。

图 12 建立根目录文件夹

3.1.2 复制固件库文件

此步骤为移植固件库的关键步骤，即是把“STM32F10x_StdPeriph_Lib_V3.5.0”文件夹里面的相关内容复制到我们建立的模板文件夹里。需要复制的文件夹如下表：

固件库文件		模板文件夹	备注
STM32F10x_StdPeriph_Lib_V3.5.0\STM32F10x_StdPeriph_Lib_V3.5.0\STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x\startup\iar\ 里面的相关内容，见下文	--->	STM32F10X Template\CMSIS\Startup	Cortex-M3 内核相关启动代码
CMSIS\CM3\DeviceSupport\ST\STM32F10x\system_stm32f10x.h 和 system_stm32f10x.c 和 stm32f10x.h	--->	STM32F10X Template\CMSIS	STM32 系统配置代码
STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\STM32F10x_StdPeriph_Driver\inc 文件夹和 src 文件夹	--->	STM32F10X Template\FWLIB	STM32 外设驱动代码
STM32F10x_StdPeriph_Lib_V3.5.0\Project\STM32F10x_StdPeriph_Template\stm32f10x_conf.h 和 stm32f10x_it.c	--->	STM32F10X Template\USER	STM32 中断管理代码
STM32F10x_StdPeriph_Lib_V3.5.0\Project\STM32F10x_StdPeriph_Template\EWARM\stm32f10x_flash.icf 和 stm32f10x_flash_extsram.icf 和 stm32f10x_nor.icf 和 stm32f10x_ram.icf	--->	STM32F10X Template\	程序内存分布代码

表 1 代码复制源和目的

这里还需要再注意一点，如果你使用的是 Keil 那么需要复制 STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\CoreSupport\里面的 core_cm3.c 和 core_cm3.h 文件，但是由于 IAR 内部提供了此文件，因此如果复制过去，会影响系统的编译。

STM32F10x_StdPeriph_Lib_V3.5.0\STM32F10x_StdPeriph_Lib_V3.5.0\STM32F1

0x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x\startup\iar\里面的相关内容，需要选择性添加到工程里面，目前我们先全部复制过来。

3.1.3 IAR 添加管理文件夹

至此，我们最基本的源码文件已经复制到我们建立的工程文件夹里面，接下来要做的就是在 IAR 里面创造代码管理文件夹，以及将相关的代码添加到文件夹里面。

首先，我们先来建立 IAR 的文件夹，操作方法为右击工程名，点击“Add”，选择“Add Group...”，如图 13 所示。

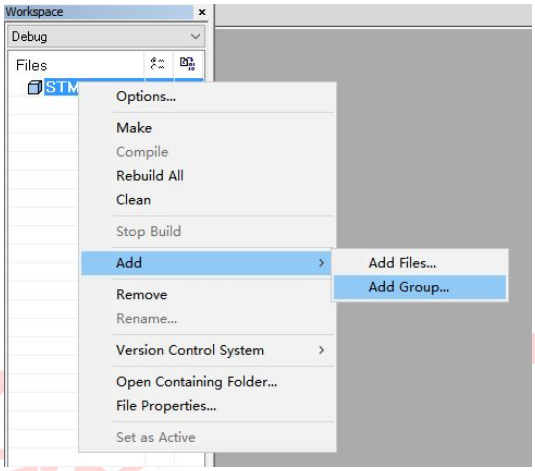


图 13 建立文件夹

之后会出现如图 14 的对话框来输入文件夹名称。在“Group name”里面输入相关的名称，点击“OK”即可以完成相应的步骤。

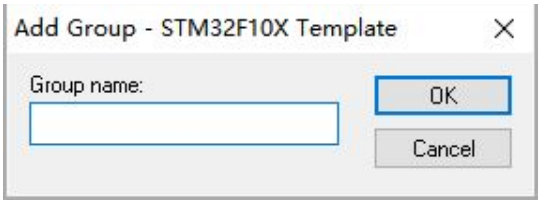


图 14 输入文件夹名称

在建立好的文件夹上面右击，然后点击“Add”，选择“Add Group...”，即可以实现阶层文件夹的建立，如图 15 所示，正如我们前面提到的，IAR 相比于 Keil 最大的一个特点，即是可以建立阶层文件夹，来层次性地管理代码，因此我们需要利用此优点。

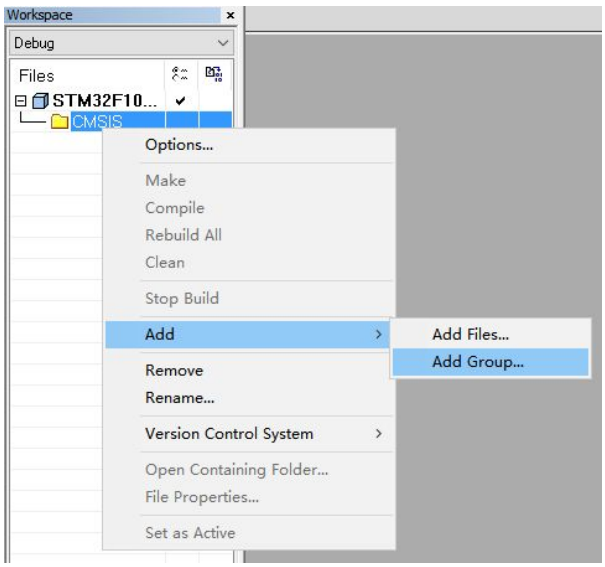


图 15 建立阶层文件夹

重复上述所讲的步骤，我们依次建立文件夹，建立好的工作区如图 16 所示。

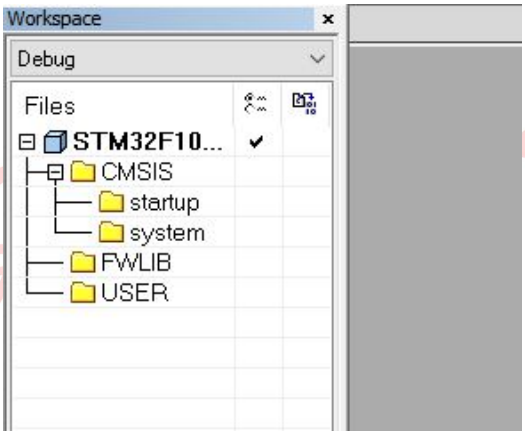


图 16 IAR 代码管理阶层文件夹

3.1.3 IAR 添加源码

接下来，我们要做的事就很简单了，将“STM32F10X Template”里面的文件添加到 IAR 里相应的目录即可。添加方式很简单，只要在想要添加的文件夹上面右击，选择“Add”，然后“Add File”即会弹出相应的对话框供用户选择文件，选择好文件之后，单击“确定”即可以添加进去。如图 17 所示。

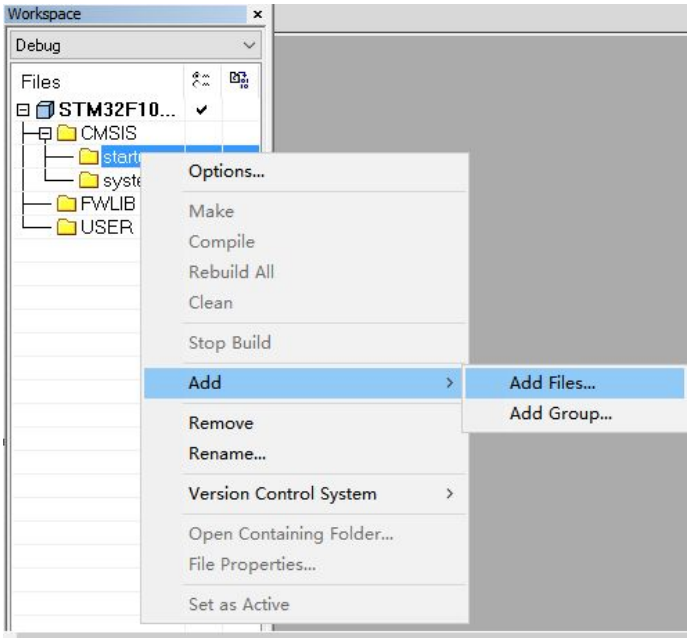


图 17 添加源码文件

依次按照上述方式，并且按照表 2 中的源和目的，完成源码的添加。

源	目的
STM32F10X Template\CMSIS\Startup 相应文件，见下文	CMSIS/startup/
STM32F10X Template\CMSIS\system_stm32f10x.c	CMSIS/system/
STM32F10X Template\FWLIB\src\文件夹里的文件	FWLIB
STM32F10X Template\USER\stm32f10x_it.c	USER

表 2 IAR 添加源码

关于 STM32F10X Template\CMSIS\Startup 文件夹里面文件的选择，需要按照对应芯片的容量来选择，正如我们第一节中介绍一样，如表 3 所示。此时，我们所有的文件添加都已经完成，最终的 IAR 工作区如图 18 所示。

启动文件	芯片	Flash 容量
startup_stm32f10x_cl.s	互联型的器件 STM32F105xx，STM32F107xx	-
startup_stm32f10x_hd.s	大容量的 STM32F101xx，STM32F102xx，STM32F103xx	256~512K
startup_stm32f10x_hd_vl.s	大容量的 STM32F100xx	
startup_stm32f10x_ld.s	小容量的 STM32F101xx，STM32F102xx，STM32F103xx	16~32K
startup_stm32f10x_ld_vl.s	小容量的 STM32F100xx	
startup_stm32f10x_md.s	中容量的 STM32F101xx，STM32F102xx，STM32F103xx	64K~128K
startup_stm32f10x_md_vl.s	中容量的 STM32F100xx	
startup_stm32f10x_xl.s	在 512K 到 1024K 字节的 STM32F101xx，STM32F102xx，STM32F103xx	512K~1024K

表 3 启动文件选择

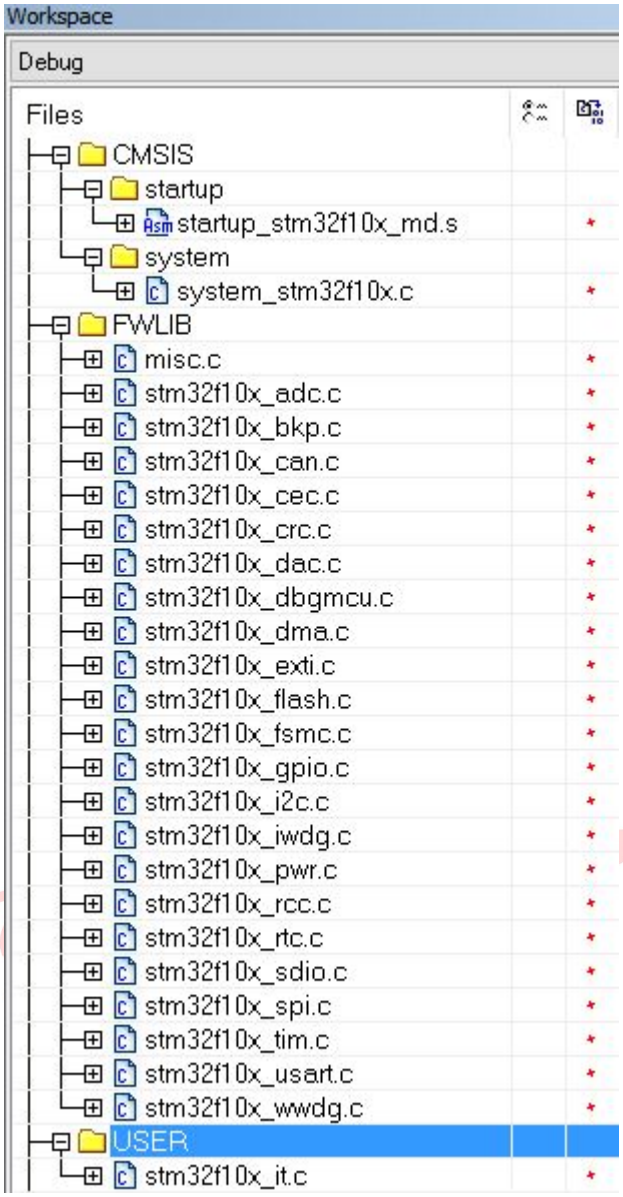


图 18 IAR 工作区文件夹

3.2 IDE 的配置

3.2.1 第一次编译

IAR 的工程建立好之后，既可以进行编译了，编译的方式有三种，其一为选择“Project”->“Make”，如图 19 所示；其二为键盘的快捷键“F7”；其三为点击图 20 里面的红圈按钮。

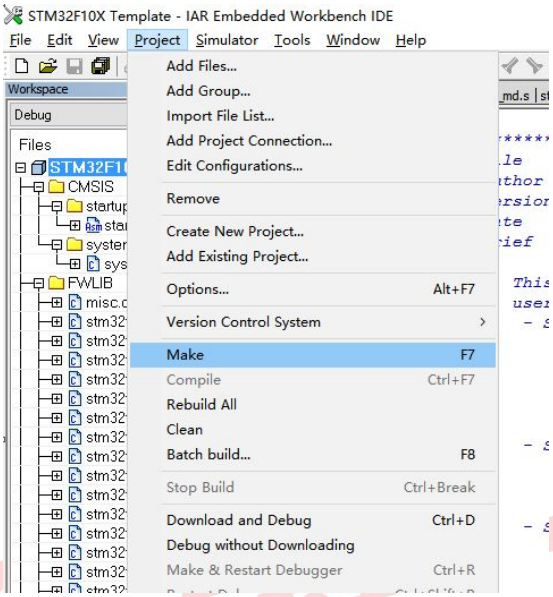


图 19 “Project”选项里面的“Make”

图 20 编译的快捷选项

如果我们在此基础上进行编译，必定会报一连串的错误，如图 21 所示。

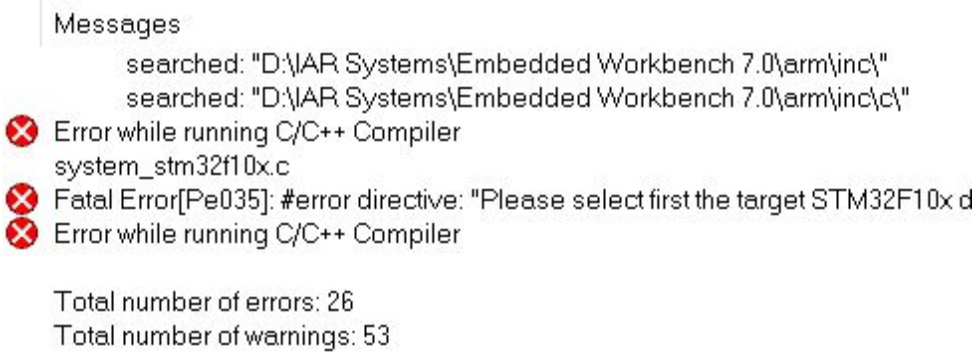


图 21 第一次编译的错误

查看底部的信息窗口，我们可以发现，里面的很多报错信息是“Fatal

Error[Pe1696]: cannot open source file..... .h”。显而易见是告诉我们无法找到所包含的头文件信息。因此我们需要对之前复制的头文件路径进行包含。

右击工程名->选择“Options”，之后就会出现如图 22 所示的选项窗口。

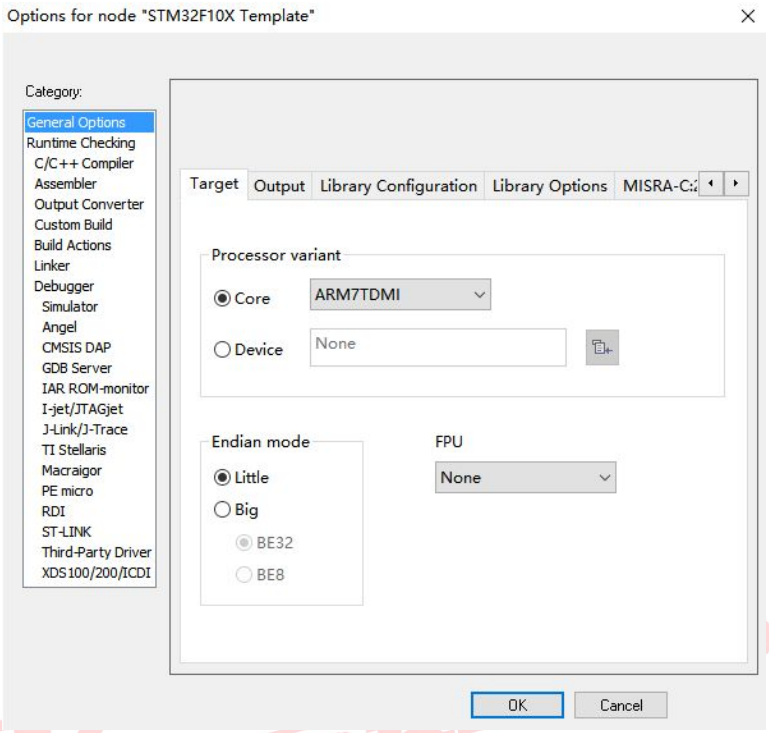
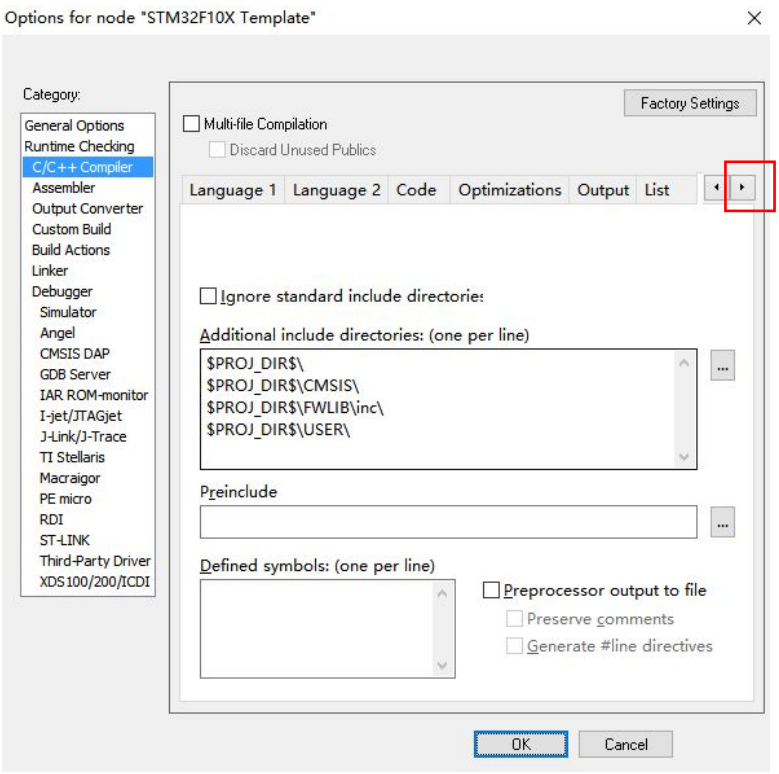


图 22 选项窗口

选择“Category”里面的“C/C++ Compiler”,然后点击图 23 中红圈所示的箭头，即可找到“Preprocessor”，然后在“Additional include directories：”输入头文件的包含路径。



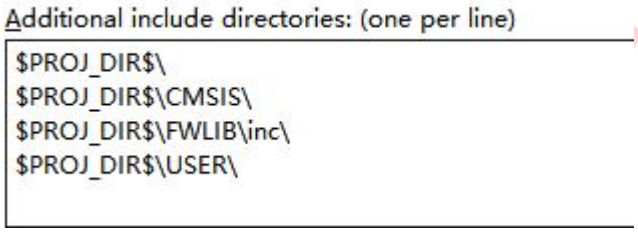


图 23 头文件包含路径

我们可以看到，其中的包含路径为：

$PROJ_DIR$\

$PROJ_DIR$\CMSIS\

$PROJ_DIR$\FWLIB\inc\

$PROJ_DIR$\USER\

每一行里面用回车隔开，我们仔细观察上面的路径，“$PROJ_DIR$”即代表根目录文件夹“STM32F10X Template”。添加好之后，选择“OK”，并且进行第二次编译。

3.2.2 第二次编译

在第二次编译中，我们可以看到，还是有很多错误，如图 24 所示。

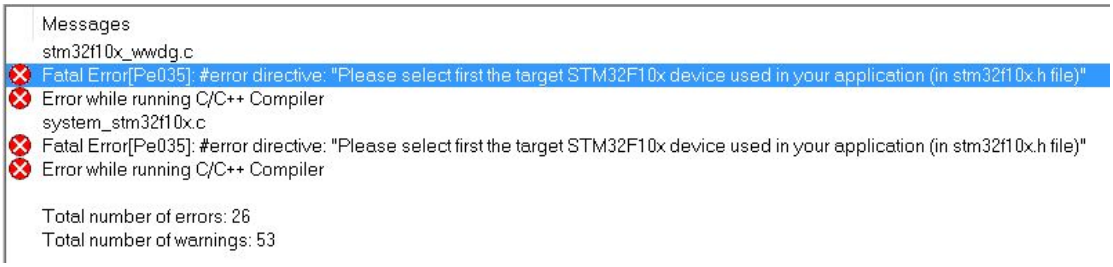


图 24 第二次编译的错误

但是仔细观察，不难发现这些错误主要为：“Fatal Error[Pe035]: #error directive: "Please select first the target STM32F10x device used in your application (in stm32f10x.h file)" ”。这是因为没有选择“stm32f10x.h”里面匹配的 STM32 器件。打开“stm32f10x.h”文件，我们会发现文件里面有芯片容量的宏定义，如图 25 所示。

```
#if !defined (STM32F10X_LD) && !defined (STM32F10X_LD_VL
  /* #define STM32F10X_LD */      /*!< STM32F10X_LD: STM3
  /* #define STM32F10X_LD_VL */   /*!< STM32F10X_LD_VL: S
  /* #define STM32F10X_MD */      /*!< STM32F10X_MD: STM3
  /* #define STM32F10X_MD_VL */   /*!< STM32F10X_MD_VL: S
  /* #define STM32F10X_HD */      /*!< STM32F10X_HD: STM3
  /* #define STM32F10X_HD_VL */   /*!< STM32F10X_HD_VL: S
  /* #define STM32F10X_XL */      /*!< STM32F10X_XL: STM3
  /* #define STM32F10X_CL */      /*!< STM32F10X_CL: STM3
#endif
```

图 25 “stm32f10x.h”里面芯片容量的宏定义

修改方式有三种，最简单的一种是直接把宏定义的屏蔽符号删除，但是直接这样设置，会发现还是修改无效，这是因为“stm32f10x.h”是只读文件，所以需要手动修改此文件的属性，但是我们还是不建议大家修改库文件。

第二种设置方式还是在“Options”里面，选择 Category”里面的“C/C++ compiler”，找到“Preprocessor”，在“Defined Symbol”里面定义所需要的芯片容量，由于我们使用的是中等容量的，因此定义“STM32F10X_MD”,如图 26 所示，然后点击“OK”进行第三次编译。

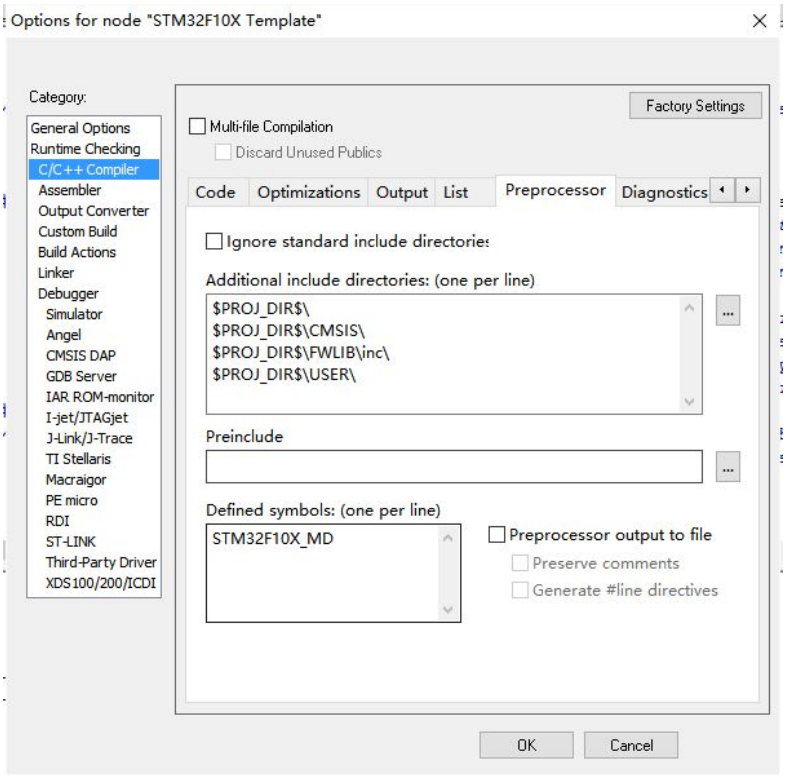


图 26 定义芯片容量

3.2.3 第三次编译

尽管我们做了上述两次修改，我们还是会发现此项目编译不过。如图 27 所示。

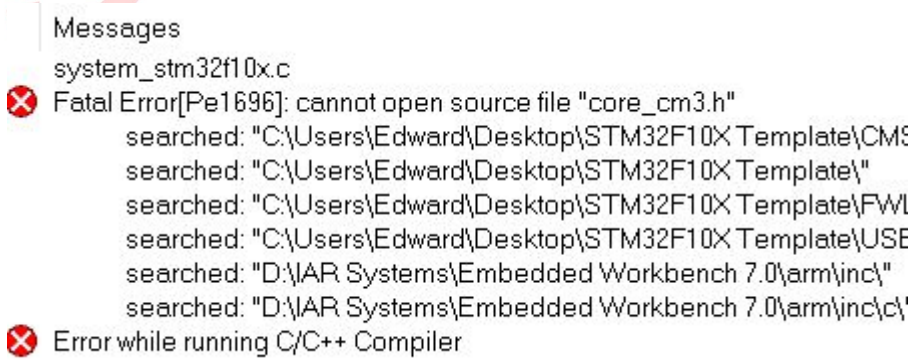


图 27 第三次编译错误信息

这是因为没有包含“core_cm3.h”文件，解决方式还是两种，其一为将库函数里面的“core_cm3.h”文件加入进工程。但是我们上文中已经提到过，这样编译可能会出现其他错误，因此我们直接使用第二种方法，即使用 IAR 自带的“core_cm3.h”文件。

设置方法为两步，首先，在“Option”->“General”->“Target”里面选择芯

片类型，如图 28 所示。

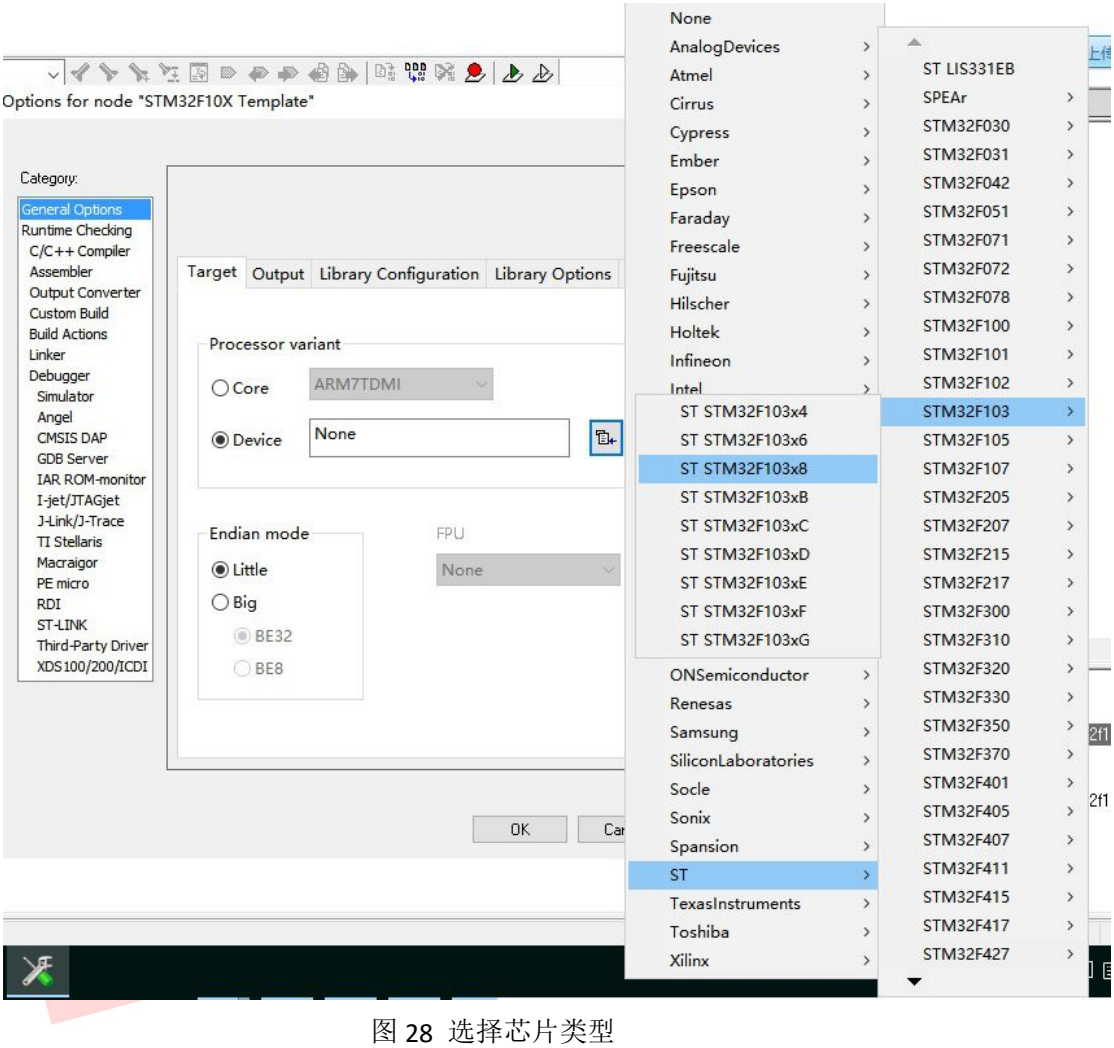


图 28 选择芯片类型

接着在在“Option”->“General”->“Library Configuration”里面勾选“use CMSIS”，点击“OK”，如图 29 所示，然后进行第四次编译。

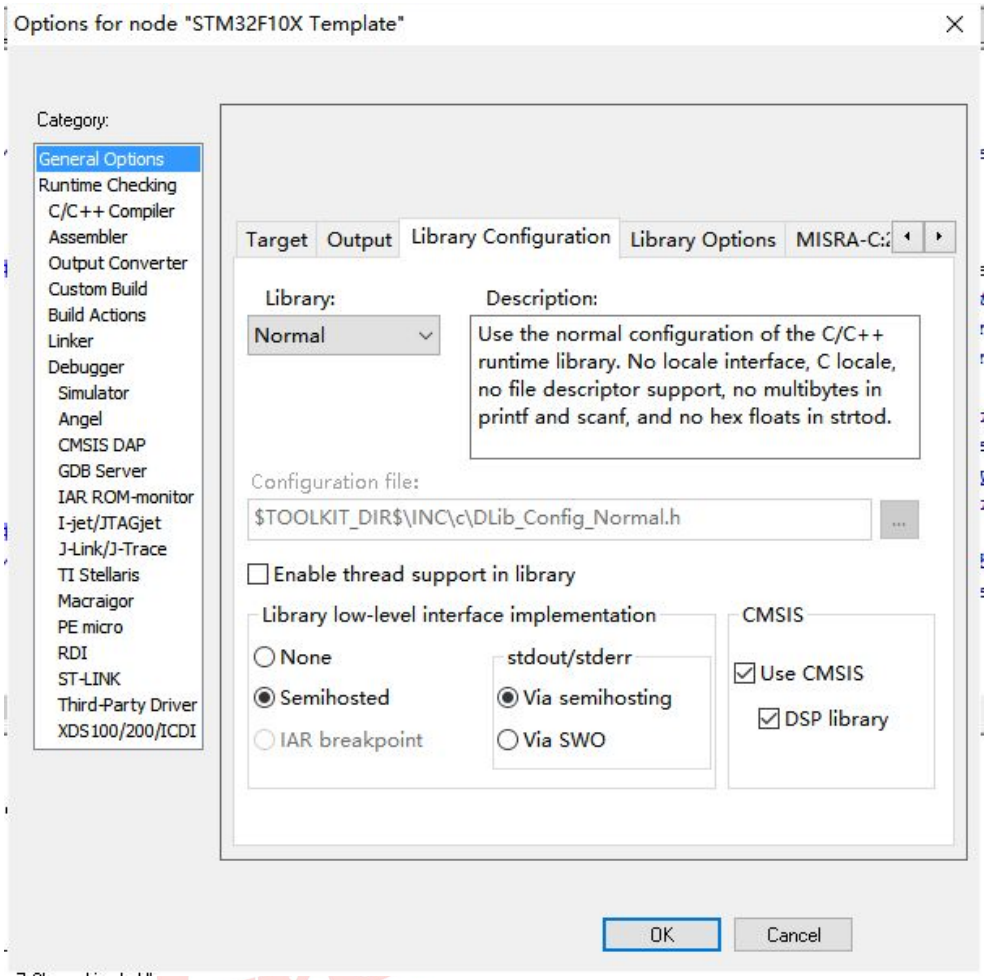


图 29 选择 IAR 内部的 CMSIS

3.2.4 第四次编译

现在，我们会发现错误少了很多，如图 30 所示，仅仅是缺少 main.c。因此我们需要在根目录的“USER”文件夹里面新建一个“main.c”文件，并且把它添加进 IAR 的 USER 文件夹里面。

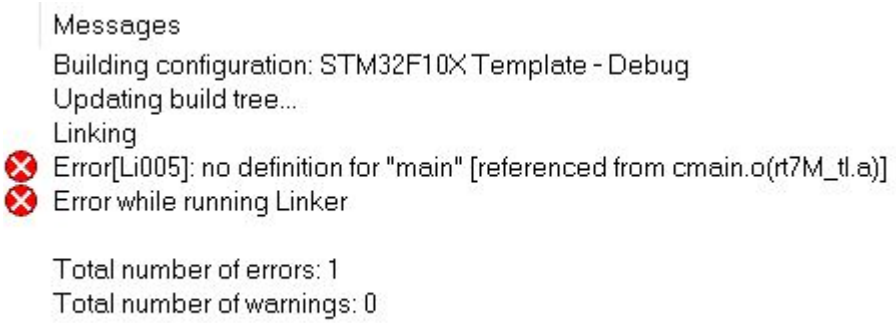


图 30 第四次编译错误信息

将 main.c 加入进工程之后，需要编写如下代码。

```
#include "stm32f10x.h"
int main()
{
   return 0;
}
```

3.2.5 第五次编译

在进行第五次编译，我们会发现错误为 0，如图 31 所示。此时，标准固件库模板就完成了，以后写代码时，可以直接使用此模板工程进行编程，以避免很多重复的工作。

图 31 第五次编译信息

对于初学者来说，可能初次使用 IAR 时，会感觉非常麻烦，这也是正常的，但是比 Keil 要简单的多，包括 IAR 选项的阶层关系，因此大家只需要多练习几次，就可以对我们本章内容有一个熟悉的掌握。

3.2.6 补充

当很多读者沉浸在此时的喜悦时，我还是要告诉大家一个不好的消息，如果用固件库模板去调用驱动程序时，还是会有很多错误，大家可以预先使用下述代码进行测试。

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
int main()
{
```

```
    GPIO_SetBits(GPIOA,0);
    return 0;
}
```

如果你试过了，恭喜你会获得一个意外的收获，解决方法就是在“Options”里面，选择 Category”里面的“C/C++ compiler”，找到“Preprocessor”，在“Defined Symbol”添加“USE_STDPERIPH_DRIVER”，如图 32 所示。

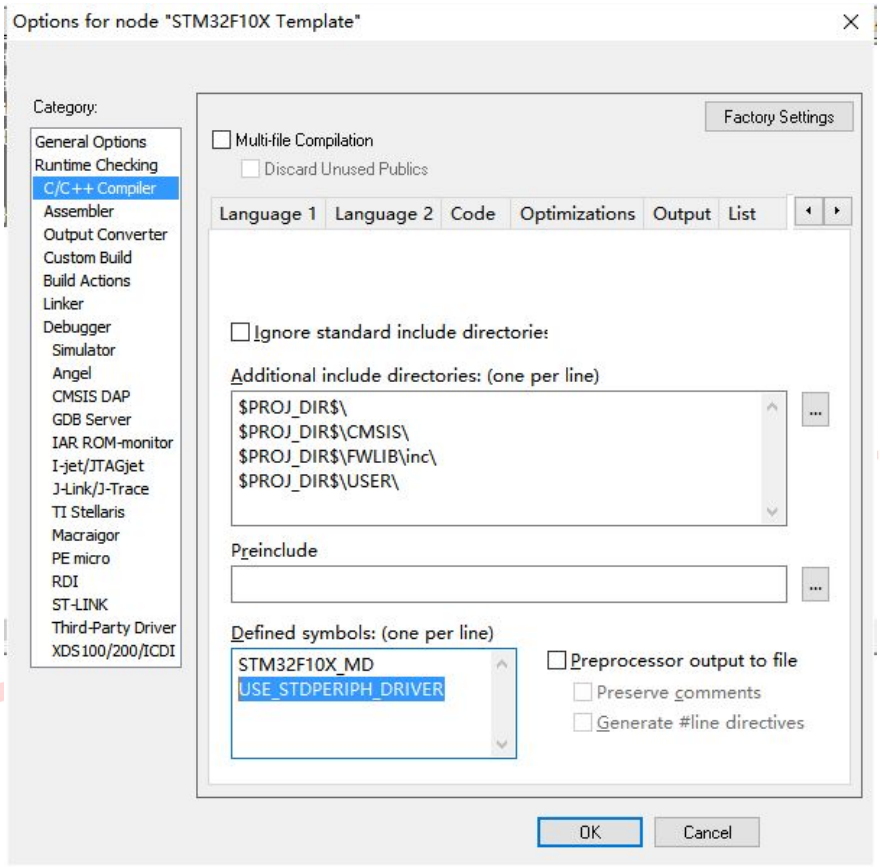


图 32 补充

最后一步，我们需要设置链接选项里面的 flash 配置文件，在选项里面找到“Linker”标签页，在“Linker Configuration file”里面，把“Override Default”选项勾选上，然后定位我们刚刚复制过来的 icf 文件，由于我们只使用了内部自带的 flash，因此定位文件“stm32f10x_flash.icf”，选择好之后，点击“OK”，如图 33 所示，至此，一个标准的 STM32F10x 工程模板就已经建立好了。为了避免文件路径变更导致的 icf 文件重设置，我们可以采用相对路径的方式，手动输入“$PROJ_DIR$\stm32f10x_flash.icf”。

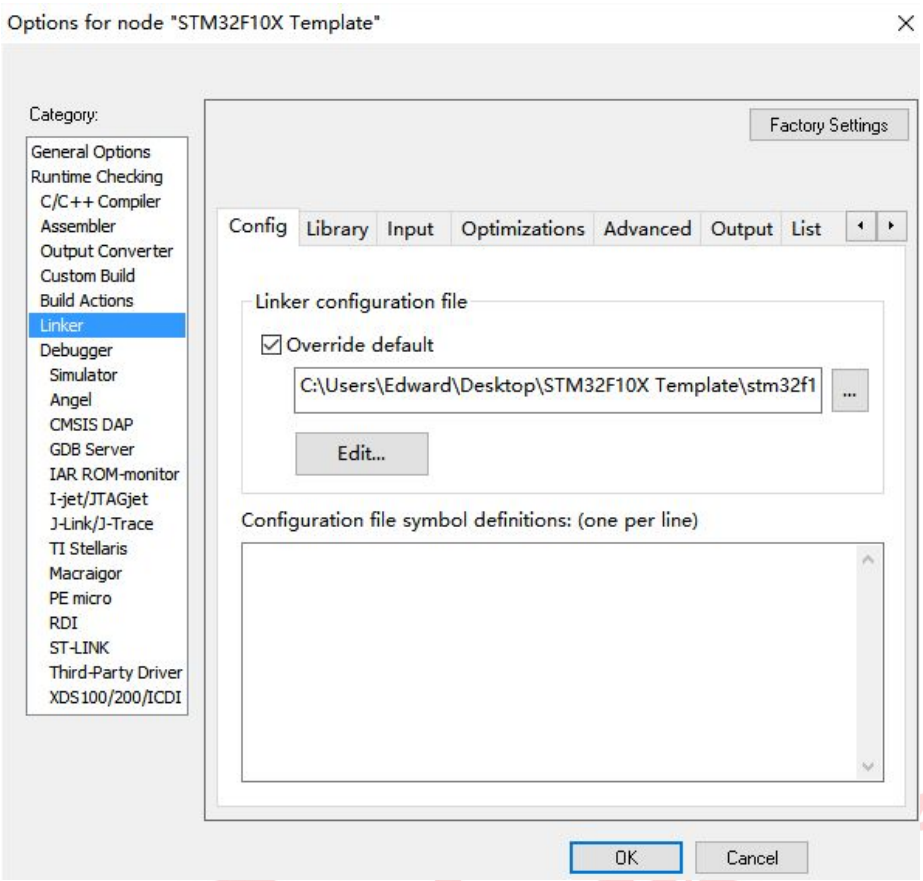


图 33 Linker 选项

微信扫码关注“乐创客”