第 3 章

MDK 开发环境

本章主要介绍本书的软件开发环境——MDK5 及其应用。首先介绍 ST 官方固件库；接着介绍 MDK5 软件及其安装和注册的过程；然后介绍怎样基于 ST 官方固件库来新建一个工程模板；最后介绍针对开发板下载程序的 3 种方式，以及在 MDK5 中对程序进行调试的方法。

本章的学习目标如下：

- 了解 STM32 应用开发的两种方式；
- 理解 ST 官方固件库的概念以及它在 STM32 应用开发过程中的作用；
- 了解 CIMIS 的含义以及定义它的目的；
- 掌握 ST 官方固件库的目录结构以及其中的重要文件/文件夹的作用；
- 掌握 MDK5 的安装及注册过程；
- 掌握在 MDK5 中基于 ST 官方固件库新建工程模板的方法；
- 掌握开发板下载程序的 3 种方式；
- 掌握在 MDK5 中对程序进行调试的方法。

3.1 STM32 官方固件库

意法半导体公司为了方便用户开发程序，提供了一套丰富的 STM32 固件库，这个 STM32 官方固件库到底是什么？它对开发程序有什么用处？用它进行开发与直接操作寄存器进行开发这两种方式又有什么区别和联系？

3.1.1 库开发与寄存器开发

很多用户都是在学习了 51 单片机之后再学习 STM32 单片机的。在 51 单片机的应用程序开发过程中，往往是通过直接操作 51 单片机的寄存器来对它进行控制的。例如，如果要控制 51 单片机的某个 I/O 口的电平状态，以端口 0 为例，假定要使端口 0 的全部引脚都输出低电平，则可以通过操作寄存器 P0 来完成，程序代码如下：

```
P0 = 0x00;
```

在 STM32 单片机的应用程序开发过程中，要使 GPIOA 的所有引脚都输出低电平，同样可以通过操作相应的寄存器 GPIOA_ODR 来完成，程序代码如下：

```
GPIOA_ODR = 0x0000;
```

但是,相比于51单片机只有几十个寄存器,STM32单片机的寄存器数目一般都要有上百个,甚至数百个,如果继续用这种直接操作寄存器的方式来对STM32单片机进行应用程序的开发,就需要熟练地掌握每个寄存器的用法,这不仅非常麻烦,还容易出错。

基于上述原因,意法半导体公司推出了自己的官方固件库,固件库是许多函数的集合,这些函数的作用为:向下与STM32单片机的各个寄存器打交道,向上为用户提供调用函数的接口(API),不同的函数实现不同的功能。也就是说,固件库中的每一个函数对底层寄存器的操作都被封装起来,它只向用户提供一个实现该操作的接口,用户不需要知道该函数具体是怎样操作以及操作哪个/些寄存器,只需要知道该函数是实现了什么样的功能,并懂得怎样使用该函数即可。

还是上面的例子,也可以通过调用固件库中的函数来实现,固件库中有一个GPIO_Write()函数,它的程序代码为:

```
void GPIO_Write(GPIO_TypeDef * GPIOx, uint16_t PortVal)
{
  /* Check the parameters */
  assert_param(IS_GPIO_ALL_PERIPH(GPIOx));

  GPIOx->ODR = PortVal;
}
```

可能大家现在还不能完全明白该函数中每一条代码的具体含义,但是应大体看出,该函数要实现的主要功能是:给GPIOx(参数)的ODR寄存器赋值PortVal(参数)。

所以,可以操作如下:

```
GPIO_Write(GPIOA, 0x0000);
```

这样,通过调用库函数而无须再去操作寄存器,就实现了上面的功能。或许这个简单的操作还不足以说明固件库功能的强大之处,但当对STM32单片机外设的工作原理有了一定的了解后,再去回看固件库,会发现库函数大多都是按照其实现的功能来命名的,而且STM32单片机的寄存器大多都是32位的,不像51单片机操作起来那样简单、不易出错,那时就感受到通过调用固件库进行应用程序开发的方便之处。

这里需要说明的是,有了STM32官方固件库,并不代表在应用程序的开发过程中就不再需要跟STM32单片机的寄存器打交道了。任何一款处理器,无论它多么高级,都是通过操作它的寄存器来对它进行控制。调用库函数,归根结底,还是通过对STM32单片机的寄存器进行操作来实现其相应的功能。所以,在对STM32单片机进行应用程序开发的过程中,仅仅掌握固件库是远远不够的,还需要理解STM32单片机及其外设的工作原理,然后需要通过了解库函数的实现细节,来加深对这些工作原理的理解,在这一系列过程中,仍然需要不断地和STM32单片机的寄存器打交道,只有在掌握了STM32单片机及其外设的工作原理,并了解了库函数的实现细节后,库函数的使用才能取得事半功倍的效果。此外,固件库不是万能的,它只是意法半导体公司给大家提供的一个能够实现许多不同功能的函数

的集合。但是，在某些应用程序的开发过程中，这些函数可能并不能满足实际的开发需求，在这种情况下，只能通过自己直接操作寄存器来完成要实现的功能。所以，在应用库函数时，只有在很好地理解了它的实现细节的情况下，才能够在实际的应用程序开发过程中做到举一反三，游刃有余。

通过本节的学习，应该对 STM32 官方固件库的作用有了一个基本的认识，并且能够看出通过操作库函数与直接操作寄存器这两种应用程序开发方式的不同。后面将结合具体的实例进一步讲解对 STM32 官方固件库的使用。

3.1.2　CMSIS

STM32 官方固件库是一系列函数的集合，这些函数需要满足什么要求呢？这里涉及标准，即本节要介绍的 CMSIS。

前面介绍过 ARM 公司和意法半导体公司，ARM 公司是一个做芯片标准的公司，它负责的是芯片内核的架构设计，而像意法半导体公司、TI 公司等不做标准，它们负责的是在 ARM 公司提供的芯片内核的基础上设计自己的芯片。因此，所有 Cortex-M3 内核的芯片的内核架构都是相同的，不同的是它们的存储器容量、片上外设、I/O 或其他模块，对于这些资源，不同的公司可以根据自己的不同需求进行不同的设计，即使是同一家公司设计的基于 Cortex-M3 内核的不同款芯片，它们的这些资源也不尽相同，比如 STM32F103VCT 和 STM32F103ZET，它们在片上外设方面就有很大的不同。在这里通过图 3-1 来更好地对它进行表述。

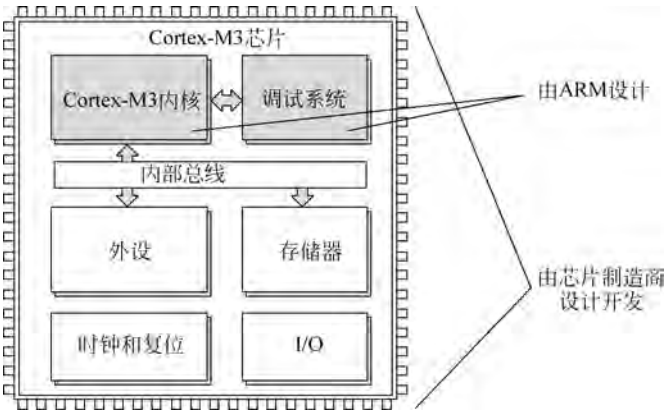

图 3-1　CM3 内核芯片的设计架构

由图 3-1 可以看出，所有基于 Cortex-M3 内核的芯片都必须遵循 ARM 公司设计的 Cortex-M3 内核的标准，而 ARM 公司为了让不同芯片公司设计生产出的不同的基于 Cortex-M3 标准的芯片在软件上能够基本兼容，就和各芯片生产商提出了一套需要大家共同遵守的标准——CMSIS（Cortex Microcontroller Software Interface Standard），翻译为“Cortex 微控制器软件接口标准”，而 ST 官方固件库就是根据这套标准进行设计的。

CMSIS 是为 Cortex-M 处理器系列提供的与（芯片）生产商无关的硬件抽象层，它定义了通用的工具接口。CMSIS 使得处理器和外设能够具有一致的设备支持和简单的软件接口，同时简化了软件的复用，减少了单片机开发人员的学习过程，并且缩短了新设备推向市

场的时间。

CMSIS是为了能够同各个芯片和软件供应商密切合作而定义的,它提供了一套通用的方式来连接外设、实时操作系统以及中间级组件。CMSIS的目的就是使得来自多个中间级供应商的软件能够统一。

CMSIS应用程序的基本结构如图3-2所示。

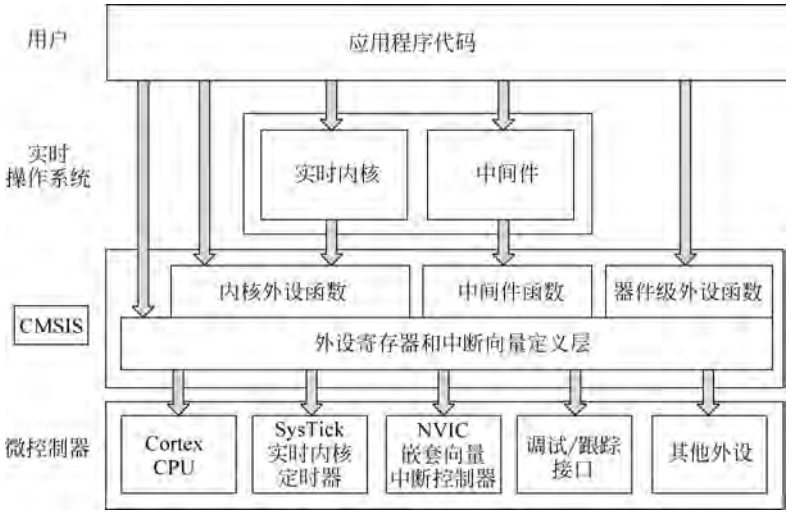

图3-2 CMSIS应用程序的基本结构

由图3-2可以看出,CMSIS层在整个系统中处于中间层,它向下负责与内核以及各个外设打交道,向上为实时操作系统和用户程序提供可调用的函数接口。如果没有CMSIS标准,各个芯片生产商可能就会按照自己喜欢的风格来设计库函数,这会使得芯片在软件上难以兼容,而CMSIS的作用就是强制规定,所有的芯片生产商都必须按照此标准来进行设计。

举个简单的例子。在使用Cortex-M3内核的单片机的时候需要首先对整个系统进行初始化,CMSIS规定,系统初始化函数的名字必须为SystemInit,所以,各个芯片公司在编写自己库函数的时候就必须用SystemInit()函数首先对系统进行初始化。此外,CMSIS还对各个外设驱动文件以及一些重要函数的命名进行规范,如3.1.1节中提到的函数GPIO_Write(),就必须按照CMSIS的规范来命名。这就保证了各芯片生产商生产出的Cortex-M3芯片在软件上具有较好的兼容性。

3.1.3 STM32官方固件库包

本节介绍STM32官方固件库包的结构及其所包含的内容。ST官方提供的固件库完整包可以在官方网站下载,固件库是在不断完善升级的,所以会有不同的版本,这里为大家提供的是V3.5版本的固件库。

将下载的固件库包解压缩到STM32F10x_StdPeriph_Lib_V3.5.0文件夹后,可以看到文件夹中的内容如图3-3所示。

文件夹的结构如图3-4所示。

图 3-3　ST 官方固件库包解压缩后的文件夹

从图 3-3 中可以看到，固件库包 STM32F10x_StdPeriph_Lib_V3.5.0 中包含 4 个子文件夹，其中，最重要的是 Libraries 文件夹，从图 3-4 中可以看到，该文件夹又包含了 2 个子文件夹，分别是 CMSIS 文件夹和 STM32F10x_StdPeriph_Driver 文件夹，固件库中几乎所有的核心文件都在这两个文件夹或其子文件夹中。

CMSIS 文件夹包含了两个子文件夹，分别是 CM3 文件夹和 Documentation 文件夹，后者包含的是文档，无须太在意；从图 3-4 中可以看出，CM3 又包含两个子文件夹，分别是 CoreSupport 文件夹和 DeviceSupport 文件夹。进入 CoreSupport 文件夹，可以看到，该文件夹包含两个文件——core_cm3.c 和 core_cm3.h，如图 3-5 所示。

这两个文件是 CMSIS 的核心文件，提供进入 CM3 内核的接口，是 ARM 公司提供的，对所有的 CM3 内核的芯片都是一样的。所以，对于开发者来说，这两个文件永远都不需要修改，大家也无须再对它进行深入研究。

图 3-4　ST 官方固件库文件夹的结构

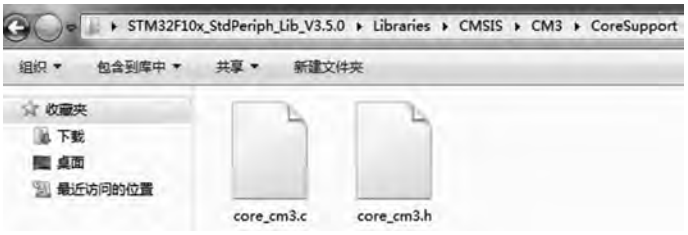

图 3-5　CoreSupport 文件夹的内容

对于 CM3 文件夹下的另一个子文件夹 DeviceSupport，一直双击到进入 DeviceSupport\ST\STM32F10x 子文件夹下，可以看到，其中包含的文件和子文件夹如图 3-6 所示。

图 3-6 中有 3 个文件：stm32f10x.h、system_stm32f10x.c 和 system_stm32f10x.h。其中，源文件 system_stm32f10x.c 和它对应的头文件 system_stm32f10x.h 的功能是设置系统以及总线的时钟，文件中有一个非常重要的函数——SystemInit()，这个函数在系统启动

图 3-6　STM32F10x 文件夹的内容

的时候必须被调用，以设置系统的时钟。stm32f10x. h 头文件也非常重要，它几乎在 STM32 单片机整个应用程序开发过程中都会被用到，因为它包含了许多重要的结构体以及宏的定义。此外，它还包含了系统寄存器定义声明以及包装内存操作，具体将会在后面介绍。

DeviceSupport\ST\STM32F10x 文件夹中还包括一个 startup 子文件夹。顾名思义，这个文件夹中放的是启动相关的文件，双击进入该文件夹，可以看到，其中包含 arm、gcc_ride7、iar 和 TrueSTUDIO 这 4 个子文件夹，再双击进入 arm 文件夹，可以看到，其中包含 8 个以 startup 开头以. s 为扩展名的文件，如图 3-7 所示。其他 3 个文件夹也类似。

图 3-7　相关启动文件所在的文件夹

这 8 个文件都是用于启动相关的文件，根据芯片容量的不同有不同的应用。这里的容量是指芯片 Flash 的大小，判断方法如下：

小容量：16KB≤Flash≤32KB

中容量：64KB≤Flash≤128KB

大容量：256KB≤Flash≤512KB

超大容量：Flash≥1024KB

这 8 个文件适用的芯片情况如下：

startup_stm32f10x_cl. s——STM32F105xx/STM32F107xx。

startup_stm32f10x_hd. s——大容量的 STM32F101xx/STM32F102xx/STM32F103xx。

startup_stm32f10x_hd_vl. s——大容量的 STM32F100xx。

startup_stm32f10x_ld. s——小容量的 STM32F101xx/STM32F102xx/STM32F103xx。

startup_stm32f10x_ld_vl. s——小容量的 STM32F100xx。

startup_stm32f10x_md. s——中容量的 STM32F101xx/STM32F102xx/STM32F103xx。

startup_stm32f10x_md_vl.s——中容量的 STM32F100xx。

startup_stm32f10x_xl.s——超大容量的 STM32F101xx/STM32F103xx。

对于天信通采用的开发板，当然是选择 startup_stm32f10x_cl.s 启动文件。

至此，CMSIS 文件夹中的内容全部都介绍完了。现在，回到 Libraries 文件夹中，看看它的另一个子文件夹 STM32F10x_StdPeriph_Driver 中的内容。双击进入该文件夹，可以看到，它主要包含 inc 和 src 两个子文件夹。可以看到，inc 文件夹包含一组.h 文件，src 文件夹包含一组.c 文件，而且它们之间是互相对应的，即在 inc 文件夹中的每一个.h 文件在 src 文件夹中都有一个同名的.c 文件与之相对应，如图 3-8 所示。

图 3-8　inc 文件夹中的.h 文件列表和 src 文件夹中的.c 文件列表

这两个文件夹中所包含的是固件库中的核心文件，src 文件夹中包含的是固件库中的源文件（src 是 source 的简写），inc 中包含的是与之对应的固件库中的头文件（inc 是 include 的缩写），每一对源文件和头文件对应的是芯片的一个外设的相关操作函数，从文件的名称中也能大体看出。

对于 Libraries 文件夹就全部介绍完了。该文件夹中的许多文件，在新建工程时都会用到，相信到时大家会对它们有更深刻的理解。

最后，再回到固件库包文件夹 STM32F10x_StdPeriph_Lib_V3.5.0，看看其他文件或文件夹。双击进入 Project 文件夹，可以看到其中包含两个子文件夹，分别是 STM32F10x_StdPeriph_Examples 和 STM32F10x_StdPeriph_Template。对于 STM32F10x_StdPeriph_Examples，顾名思义，其中存放的是 ST 官方提供的芯片外设固件的实例程序，这些程序对今后的学习和开发都十分重要，可以参考其中的源代码，将其修改后变为自己的代码来驱动开发板的相关外设（其实，市面上许多开发板配套的例程都参考了其中的例程源代码）。STM32F10x_StdPeriph_Template 中存放的是工程模板相关的文件和文件夹，其中的许多文件在新建工程时也会用到。

STM32F10x_StdPeriph_Lib_V3.5.0 中的 Utilities 文件夹下存放的是官方评估版的一些相关源码，可以略过。

最后，固件库包中还包含一个 stm32f10x_stdperiph_lib_um.chm 文件，直接打开可以看到，这是固件库的一个帮助文档，这个文档是英文的，在学习和开发过程中，经常需要查阅该文档。

3.2　MDK5 简介

MDK 是 RealView MDK 的简称，源自德国的 Keil 公司。全球有超过 10 万嵌入式开发工程师使用 MDK，目前其最新版本为 MDK5.14，该版本使用 μVision5 IDE 集成开发环境，

是目前针对 ARM 处理器,尤其是 Cortex-M 内核处理器的最佳开发工具。

MDK5 向后兼容 MDK4 和 MDK3 等,同时又加强了针对 Cortex-M 微控制器开发的支持,并且对传统的开发模式和界面进行了升级。MDK 有两个组成部分:MDK Core 和 Software Packs。其中 Software Packs 可独立于工具链进行新芯片的支持和中间库的升级,如图 3-9 所示。

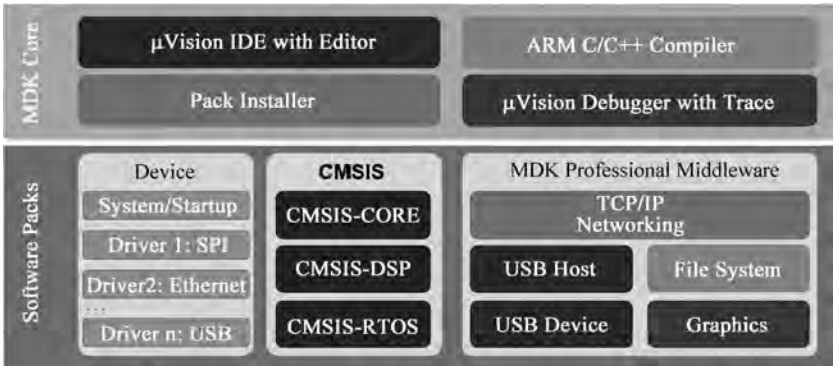

图 3-9 MDK 的两个组成部分

从图 3-9 可以看出,MDK Core 又分成 4 部分:µVision IDE with Editor(编辑器)、ARM C/C++Compiler(编译器)、Pack Installer(包安装器)、µVision Debugger with Trace(调试跟踪器)。µVision IDE 从 MDK4.7 版开始就加入了代码提示和语法动态检测等实用功能,相对于以往的 IDE 改进很大。

Software Packs(包安装器)又分为 Device(芯片支持)、CMSIS(Cortex 微控制器软件接口标准)和 MDK Professional Middleware(中间库)3 部分,通过包安装器,可以安装最新的组件,从而支持新的器件,提供新的设备驱动库以及最新例程等,加速产品开发进度。

以往版本的 MDK 将所有组件都包含到一个安装包里,显得太“笨重”。MDK5 则与它们不同,MDK Core 是一个独立的安装包,它并不包含器件支持和设备驱动等组件,但是一般都会包含 CMSIS 组件,大小为 350MB 左右,相比 MDK4.70A 的超过 500MB“瘦身”不少。器件支持、设备驱动、CMSIS 等组件,则可以在安装完 MDK5 后,双击 MDK5 的 Build Toolbar 的最后一个图标调出 Pack Installer,来进行各种组件的安装。

在安装完 MDK5 后,为了让 MDK5 能够支持 STM32F107 芯片的开发,还需要安装 STM32F1 的器件支持包 Keil.STM32F1xx_DFP.1.0.5.pack。

3.3 MDK5 的安装

3.3.1 MDK5 的安装步骤

在 MDK5 文件夹中,有给大家提供的 MDK5 安装及注册软件以及 STM32F1 的器件支持包,如图 3-10 所示。

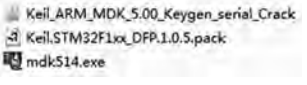

图 3-10 安装 MDK5 的相关文件夹

首先双击 mdk514.exe,在弹出的对话框中单击 Next

按钮，如图 3-11 所示。

在弹出的对话框中，选中“I agree to all the terms of the preceding License Agreement”，单击 Next 按钮，如图 3-12 所示。

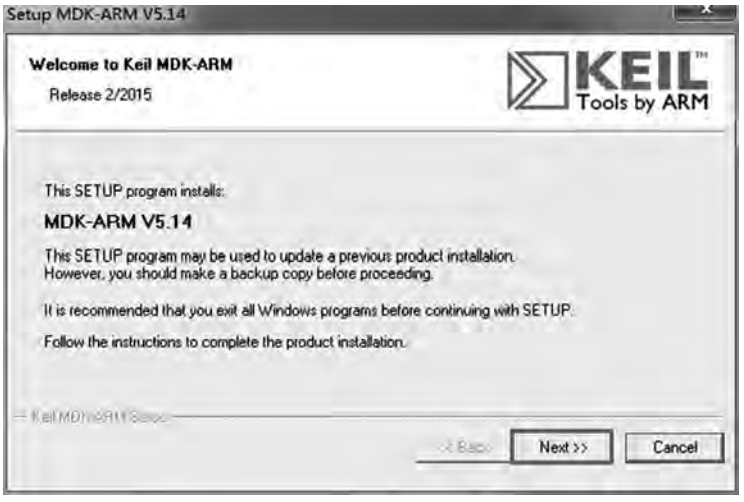

图 3-11 MDK5 的安装界面(1)

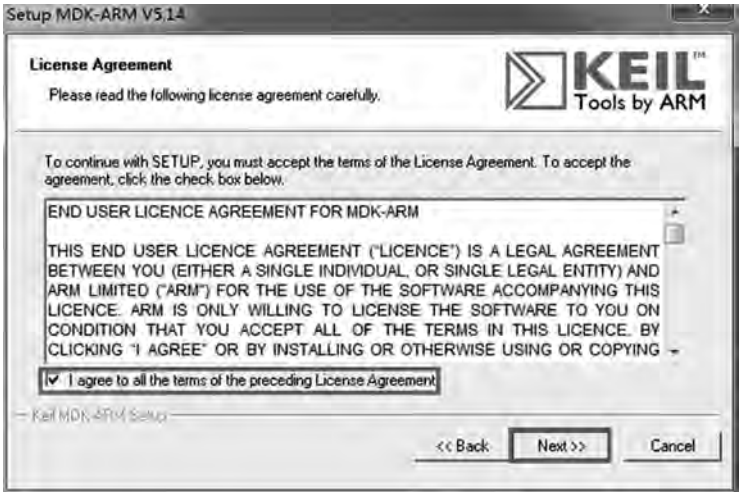

图 3-12 MDK5 的安装界面(2)

在弹出的对话框中，分别单击 2 个 Browse 按钮可以分别选择软件和支持包的安装路径，这里要注意的是，安装路径不能包含中文名字，选择好后单击 Next 按钮，如图 3-13 所示。

在弹出的对话框中填写相应的姓名、公司和邮箱的信息后，单击 Next 按钮，如图 3-14 所示。

然后，软件会进入安装过程，如图 3-15 所示。

在软件安装的最后阶段，会弹出询问是否要安装 ULINK Drivers 的对话框，单击“安装”按钮，如图 3-16 所示。

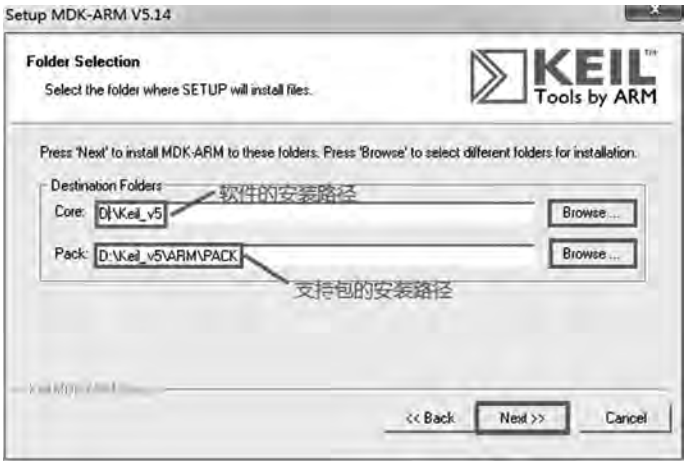

图 3-13　MDK5 的安装界面(3)

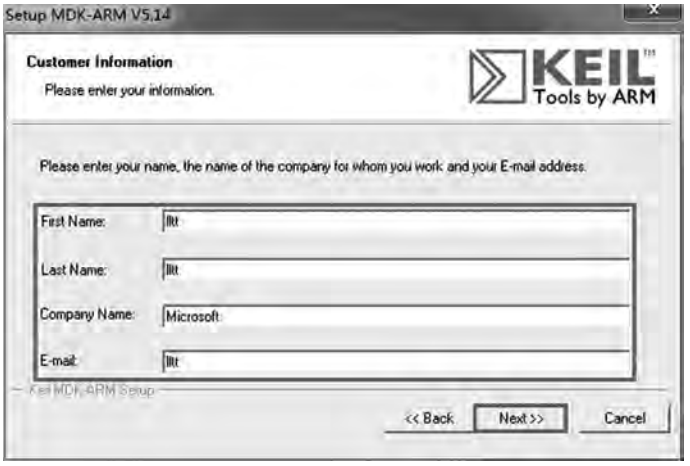

图 3-14　MDK5 的安装界面(4)

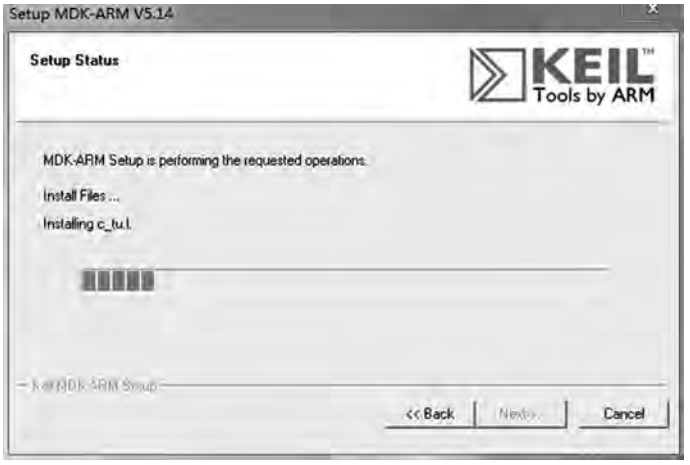

图 3-15　MDK5 的安装界面(5)

图 3-16　MDK5 的安装界面(6)

最后,单击 Finish 按钮,完成软件的安装,如图 3-17 所示。

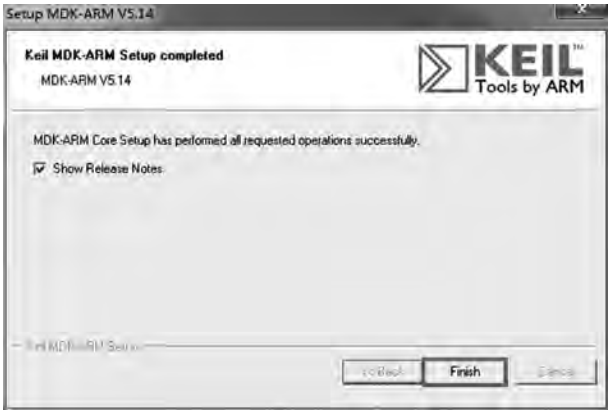

图 3-17　MDK5 的安装完成界面

可以看到,MDK 会自动弹出 Pack Installer 界面,如图 3-18 所示。

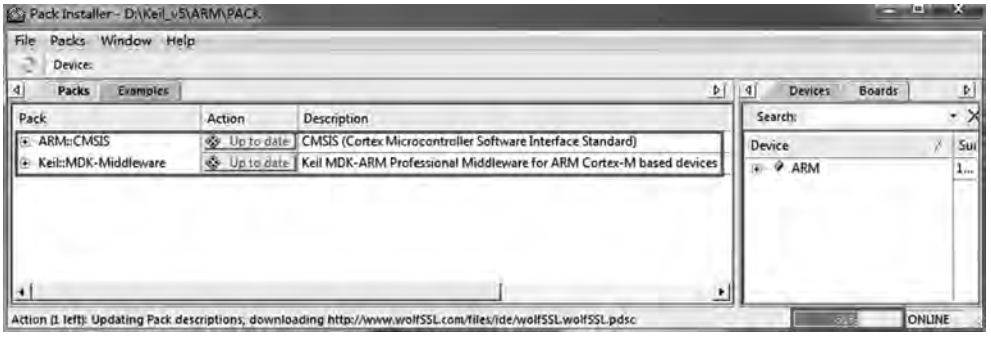

图 3-18　MDK 弹出的 Pack Installer 界面

从图 3-18 可以看出,CMSIS 和 MDK 的中间支持包已经在 MDK5.14 的安装过程中安装好了。对于其他各种支持包,程序会自动去 Keil 的官网下载,不过这个过程可能会失败,如图 3-19 所示。

图 3-19　下载失败

单击“确定”按钮,关闭 Pack Installer 安装界面。所有的支持包都可以在官网下载。

对于 STM32F107 开发板,至少需要安装 CMSIS 和 STM32F107 的器件支持包,因为 CMSIS 在 MDK5.14 的安装过程中已经安装好了,所以无须再下载安装,只需安装 STM32F107 的器件支持包。这个器件支持包已经为大家准备好了,即图 3-10 中 MDK 文件夹中的 Keil.STM32F1xx_DFP.1.0.5.pack 文件,注意,这个文件只是 STM32F1 系列芯片的器件支持包,对其他系列的芯片不适用。

双击该文件,在弹出的对话框中,单击 Next 按钮,如图 3-20 所示。

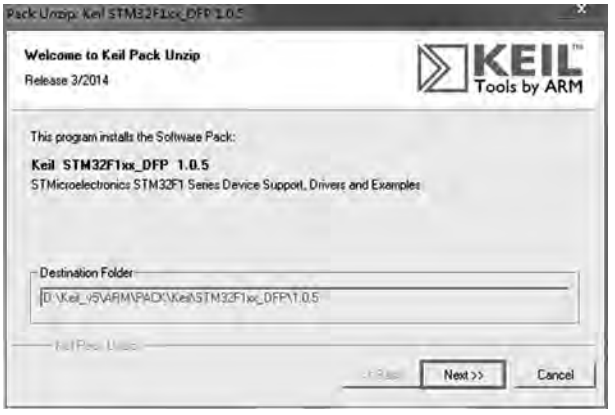

图 3-20　STM32F1 系列芯片器件支持包的安装界面

软件开始进入安装过程,如图 3-21 所示。

最后,单击 Finish 按钮完成安装,如图 3-22 所示。

3.3.2　MDK5 的注册

双击桌面上的 Keil μVision5 图标,打开 MDK5 软件,如图 3-23 所示。

在打开的 MDK5 的软件界面中,单击菜单命令 File→License Management,如图 3-24 所示。

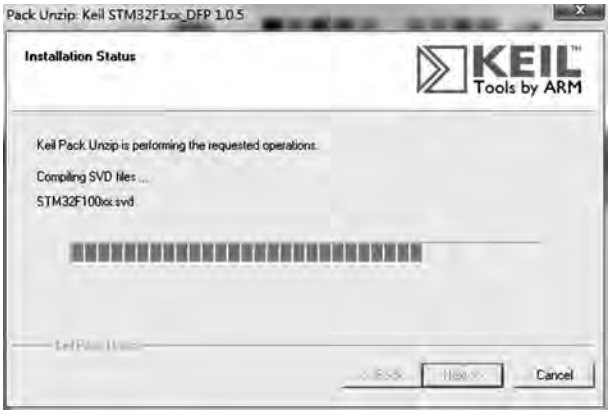

图 3-21　STM32F1 系列芯片器件支持包的安装界面

图 3-22　STM32F1 系列芯片器件支持包的安装完成界面

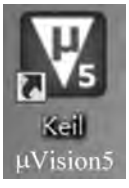

图 3-23　Keil μVision5 图标

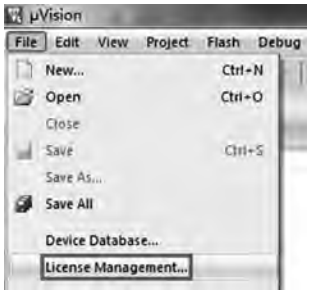

图 3-24　在 MDK5 的菜单栏中单击菜单命令 File→License Management

在弹出的对话框中，可以看到，现在的软件还是评估版的，如图 3-25 所示。

使用评估版软件是有限制的——不能编译超过 32KB 的程序代码，所以，需要对软件进行注册后，才能正常使用。给大家提供的注册软件在图 3-10 所示的 MDK5 文件夹中的 Keil_ARM_MDK_5.00_Keygen_serial_Crack 子文件夹中，进入该文件夹，双击 Keil_ARM_MDK_5.00_Keygen_serial_Crack.exe，如图 3-26 所示（如果遇到因杀毒软件而禁止运行的

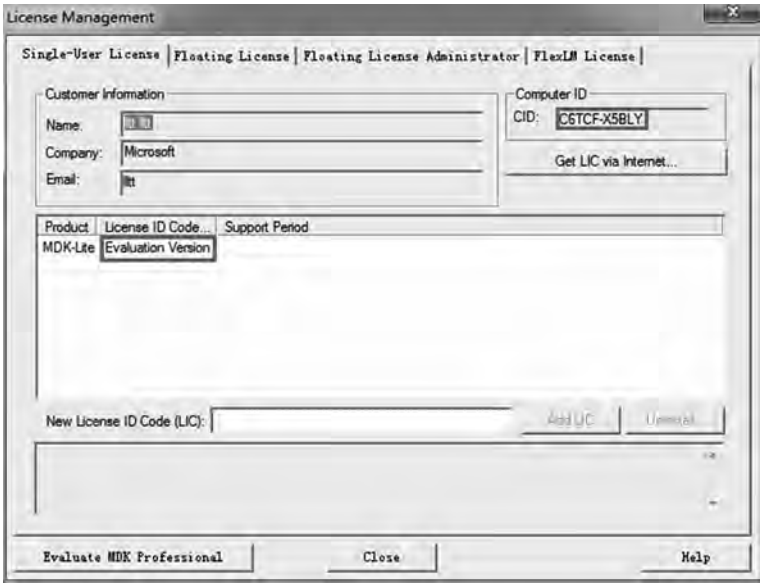

图 3-25　License Management 对话框

情况,就先关闭杀毒软件)。

在弹出的对话框中,将图 3-25 中的 CID 复制到相应的 CID 文本框中,在 Target 下拉列表框中选择 ARM,在下面的下拉列表框中选择 MDK Professional,然后单击 Generate 按钮,文本框中会生成一串注册码,如图 3-27 所示。

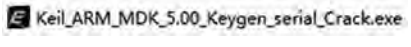

图 3-26　MDK5 的相关注册软件

图 3-27　MDK5 的注册软件界面

将此注册码再复制到图 3-25 中 New License ID Code(LIC)文本框中,并单击 Add LIC 按钮,如图 3-28 所示。

可以看到,软件注册成功。

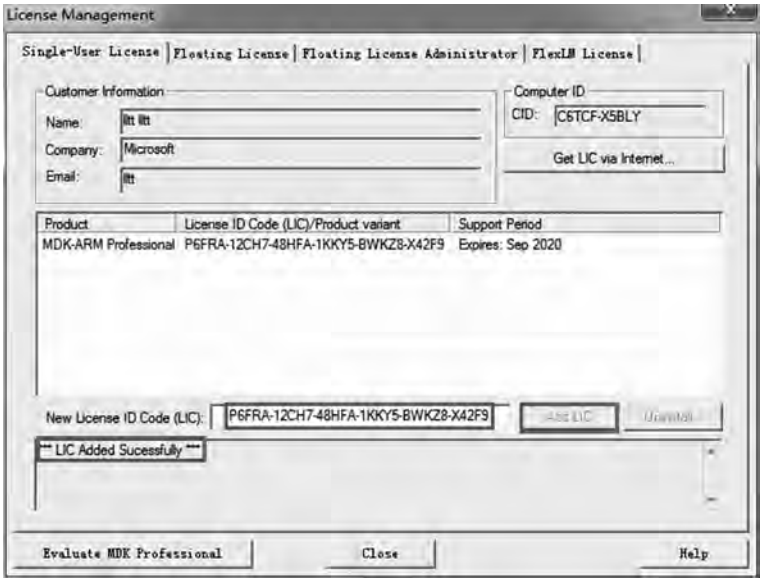

图 3-28 软件注册成功

3.4 基于固件库新建工程模板

本节介绍怎样在 MDK5 中基于 ST 官方固件库来新建一个工程。首先，需要新建一个文件夹，将其命名为 Template。

打开 MDK5.14，单击菜单命令 Project→New μVision Project，如图 3-29 所示。

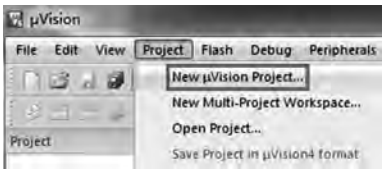

图 3-29 在 MDK5 的菜单栏中单击菜单命令 Project→New μVision Project

在弹出的对话框中，找到刚才建立的 Template 文件夹，并在其中新建一个 USER 文件夹，然后双击 USER 文件夹，我们的工程就建立在其中，将其命名为 Template，并单击“保存”按钮，如图 3-30 所示。

然后，会弹出一个为工程选择设备的对话框，在其中为建立的工程选择相关类型的芯片，因为使用的开发板的芯片是 STM32F107VCT6，所以选择 STMicroelectronics→STM32F1 Series→STM32F107→STM32F107VC，然后单击 OK 按钮，如图 3-31 所示。

注意，这里只有像前面那样安装了相关的器件支持包，才会有相应的芯片可供选择。然后，MDK5 会弹出 Manage Run-Time Environment 对话框，如图 3-32 所示。

这是 MDK5 新增的一个功能，在这个对话框中，可以根据实际情况添加自己需要的组件，从而便于应用程序的开发。这里不对它进行详细介绍，直接单击 Cancel 按钮即可。

现在，MDK5 软件界面如图 3-33 所示，工程只是初步建立起了一个框架，还需要添加相关的启动文件、外设驱动文件等。

再进入工程安装的 USER 文件夹中，可以看到，现在已生成两个文件夹和两个文件，如图 3-34 所示。

图 3-30　Template 文件夹中的 USER 文件夹

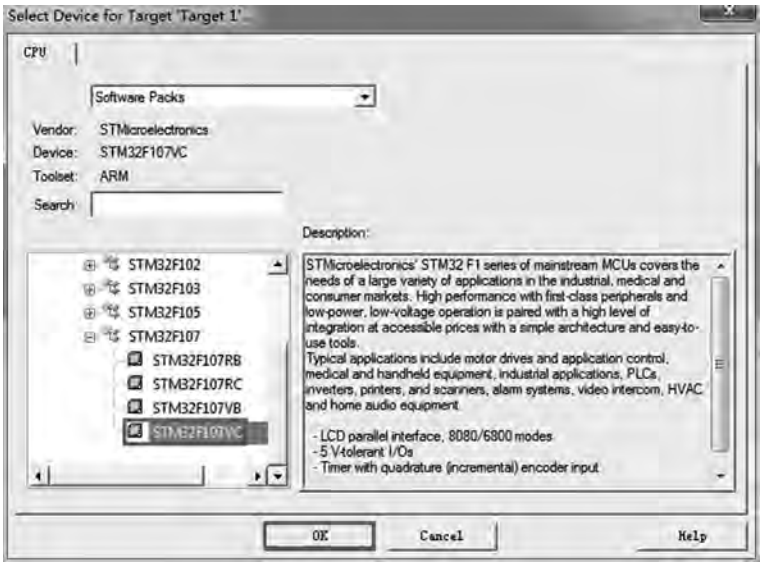

图 3-31　Select Device for Target 对话框

其中，Template. uvprojx 就是我们建立的工程文件。Listings 和 Objects 这两个文件夹是在新建工程的过程中由 MDK5 自动生成的，用来存放工程在以后编译过程中将会产生的中间文件。为了能够使 MDK5. 14 新建的工程与之前版本新建的工程更好地兼容，将这两个文件夹都删除掉，我们会在后面的步骤中新建一个 OBJ 文件夹，用来代替它们完成相应的功能，即存放工程在编译过程中产生的中间文件。当然，也可以不删除它们，只是不会用到它们而已。

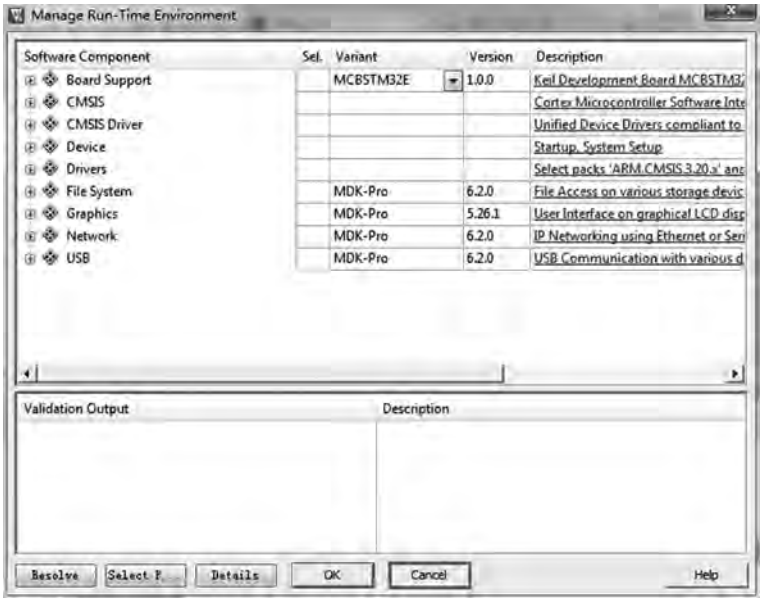

图 3-32　Manage Run-Time Environment 对话框

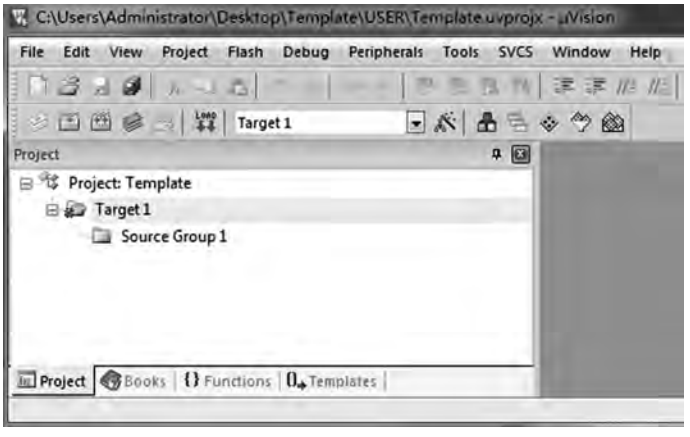

图 3-33　工程初步框架

图 3-34　工程的 USER 文件夹

接下来,在工程的 Template 文件夹下,新建 3 个与 USER 文件夹同一级的文件夹,分别将它们命名为 CORE、OBJ 和 FWLIB,如图 3-35 所示。

图 3-35 工程的 Template 文件夹

我们知道,OBJ 将被用来存放工程在编译过程中产生的中间文件;CORE 被用来存放工程的核心文件和启动文件;顾名思义,FWLIB 被用来存放 ST 官方提供的库函数文件。此外,USER 除了存放 Template 工程文件,还会被用来存放主函数文件 main.c 以及其他相关文件等。

下面将 ST 官方固件库中新建工程时用到的相关文件分别复制到上面的 4 个文件夹中。

首先,找到 ST 官方固件库包 STM32F10x_StdPeriph_Lib_V3.5.0,定位到 STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\STM32F10x_StdPeriph_Driver,在该文件夹下可以看到有两个子文件夹 inc 和 src,将它们复制到 Template 工程目录下的 FWLIB 子文件夹中,如图 3-36 所示。

图 3-36 Template 工程目录下的 FWLIB 子文件夹

然后,将固件库包 STM32F10x_StdPeriph_Lib_V3.5.0 定位到 STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\CoreSupport,将该文件夹下的两个文件 core_cm3.c 和 core_cm3.h 复制到 Template 工程目录下的 CORE 子文件夹中,再将固件库包定位到 STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x\startup\arm,将该文件夹下的 startup_stm32f10x_cl.s 文件也复制到 Template 工程目录下的 CORE 子文件夹中,如图 3-37 所示,这里选择 startup_stm32f10x_cl.s 文件,是为了对应开发板的 STM32F107 芯片。

接下来,再将固件库包定位到 STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x,将该文件夹下的 stm32f10x.h、system_stm32f10x.c 和 system_stm32f10x.h 这 3 个文件复制到 Template 工程目录下的 USER 子文件夹中,最后,将固件库包定位到 STM32F10x_StdPeriph_Lib_V3.5.0\Project\STM32F10x_StdPeriph_Template,将该文件夹下的 main.c、stm32f10x_conf.h、stm32f10x_it.c 和 stm32f10x_it.h 这 4 个文件复制到 Template 工程目录下的 USER 子文件夹中,system_stm32f10x.c 文件因为刚刚已经复制过了,所以无须再复制,如图 3-38 所示。

在将 STM32 固件库包中的相关文件复制到 Template 工程目录下的几个子文件夹后,还需要将它们添加到 Template 工程中。

图 3-37　Template 工程目录下的 CORE 子文件夹

图 3-38　Template 工程目录下的 USER 子文件夹

在 MDK5 软件界面中左侧的 Project 窗口中(如果没有出现,可以通过单击菜单命令 View→Project Window 调出),右击 Target1,在弹出的快捷菜单中选择 Manage Project Items 命令,或直接单击菜单栏的图标,在弹出的对话框中,在 Project Targets 一栏,将 Target1 更名为 Template,使之与我们的工程名相同;在 Groups 一栏,将 Source Group1 删除,并添加 3 个文件夹 CORE、FWLIB 和 USER,然后单击 OK 按钮,如图 3-39 所示。

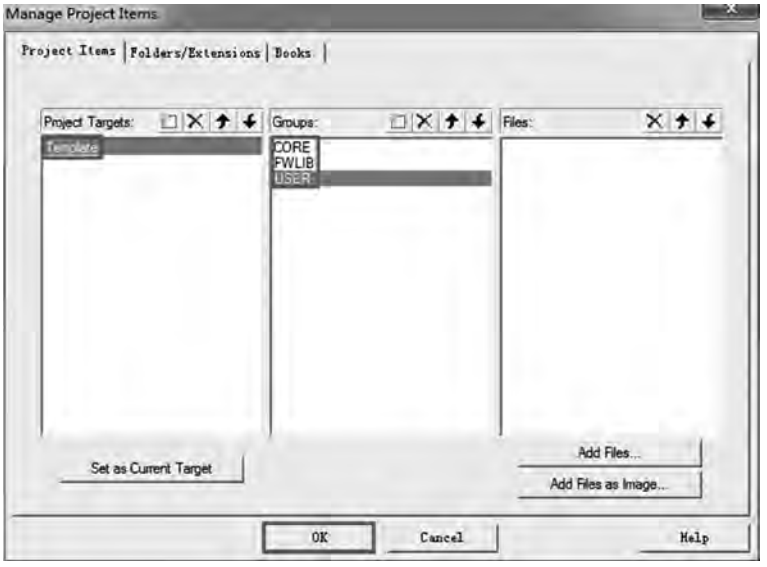

图 3-39　Manage Project Items 对话框

现在,在 MDK5 软件界面左侧的 Project 框中,就可以看到刚才修改和添加的文件夹,如图 3-40 所示。

然后,再次用刚才的方法进入 Manage Project Items 对话框,将相关文件添加到 Template 工程的 CORE、FWLIB 和 USER 这 3 个子文件夹中。

首先,在 Groups 一栏中选择 CORE 子文件夹,并在 Files 一栏的下面单击 Add Files 按钮,在弹出的对话框中找到 CORE 子文件夹(默认在 Template 文件夹中寻找),然后选中该文件夹下的 core_cm3.c 和 startup_stm32f10x_cl.s 文件(注意,这里因为有.s 文件,所以需

在文件类型下拉列表中选择 All Files),最后单击 Add 按钮,如图 3-41 所示。

图 3-40 Template 工程目录结构

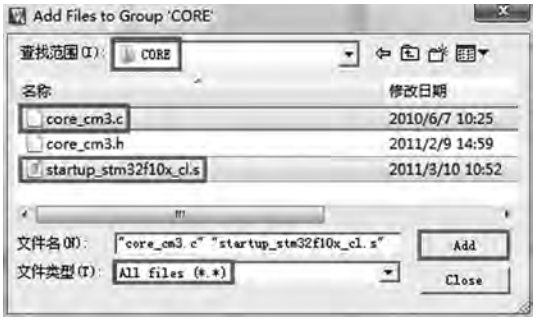

图 3-41 向工程的 CORE 子文件夹添加文件

然后,单击 Close 按钮关闭此对话框,在 Manage Project Items 对话框中的 Files 一栏可以看到这两个文件已被添加到 CORE 子文件夹中,如图 3-42 所示。

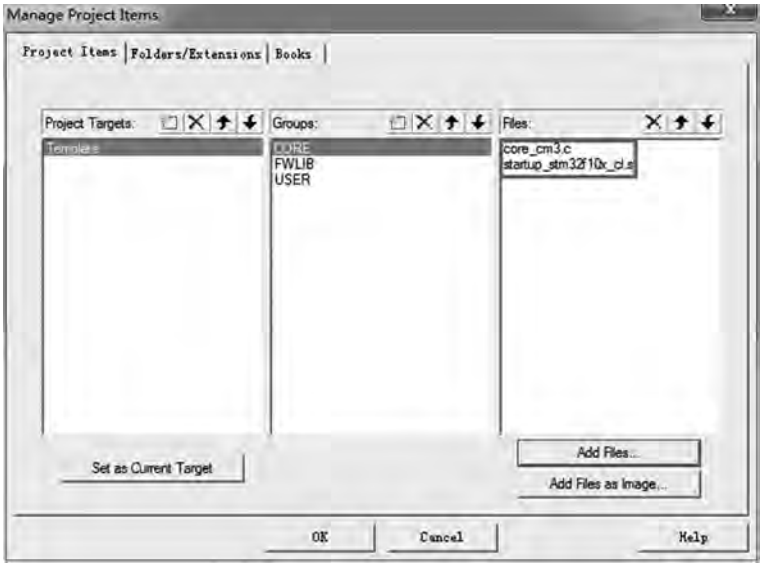

图 3-42 相关文件已被添加到工程的 CORE 子文件夹中

用相同的方法,将相关文件添加到 FWLIB 和 USER 这两个子文件夹中。

在 Manage Project Items 对话框中的 Groups 一栏中,选择 FWLIB 子文件夹,然后单击 Add Files 按钮,在弹出的对话框中,找到 FWLIB 子文件夹,因为添加的是. c 文件而不是. h 文件,所以,再单击进入 FWLIB 文件夹下的 src 子文件夹,然后,按 Ctrl+A 快捷键,选中所有的文件,并单击 Add 按钮,将它们全部添加到 FWLIB 子文件夹中,分别如图 3-43 和图 3-44 所示。

在图 3-43 中,因为要添加的文件全部都是. c 文件,所以文件类型可以默认选择为 C Source file。需要注意的是,在实际开发过程中,一般是用到哪个外设,才向 FWLIB 子文件夹添加这个外设相关的. c 文件,以免工程太大,编译起来太慢,这里是为了讲解需要才将它们一次性全部加进来。

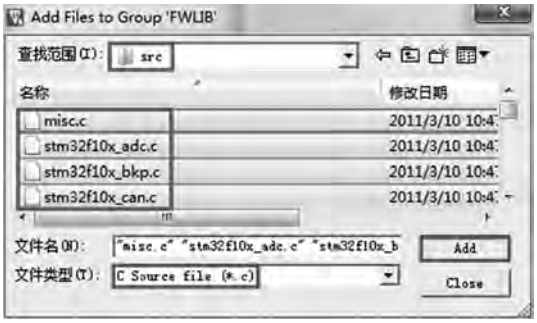

图 3-43 向工程的 FWLIB 子文件夹中添加文件

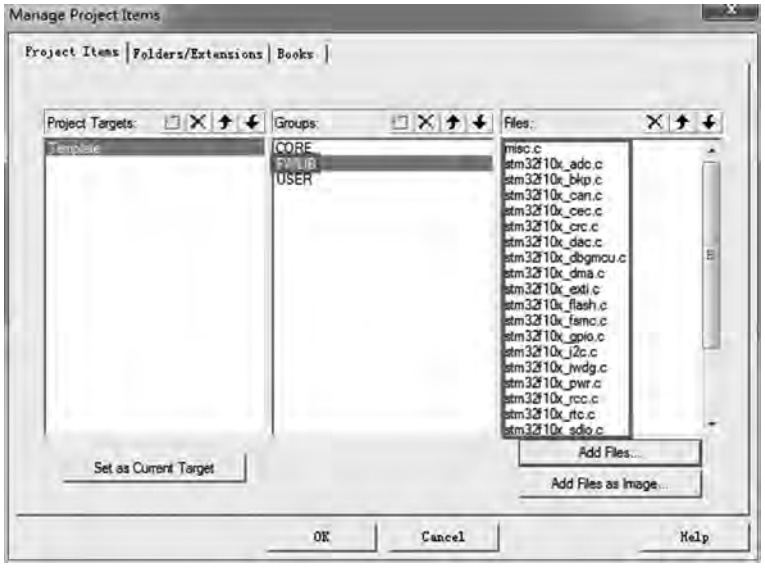

图 3-44 相关文件已被添加到工程的 FWLIB 子文件夹中

最后，用相同的方法将 Template 工程的 USER 子文件夹下的 main.c、stm32f10x_it.c 和 system_stm32f10x.c 文件添加到 Template 工程的 USER 子文件夹下，并单击 OK 按钮，如图 3-45 所示。

在 MDK5 界面左侧的 Project 框中，可以看到各文件都被相应地添加到 Template 文件夹下的 CORE、FWLIB 和 USER 3 个子文件夹中，如图 3-46 所示。

下面生成工程。在生成之前，首先应当选择工程在生成过程中产生的中间文件存放的位置。在 MDK5 的工具栏上单击 图标(也可选择菜单命令 Project→Options for Target 'Template')，如图 3-47 所示。

在弹出的对话框中，在最顶部的标签列表中选择 Output，然后单击 Select Folder for Objects 按钮，在弹出的对话框中，选择存放位置为 Template 工程目录下的 OBJ 子文件夹，然后单击 OK 按钮，回到 Options for Target 'Template'对话框，再次单击 OK 按钮，如图 3-48 所示。

单击工具栏的 图标(也可以选择菜单命令 Project→Build Target)，如图 3-49 所示。试着生成工程。

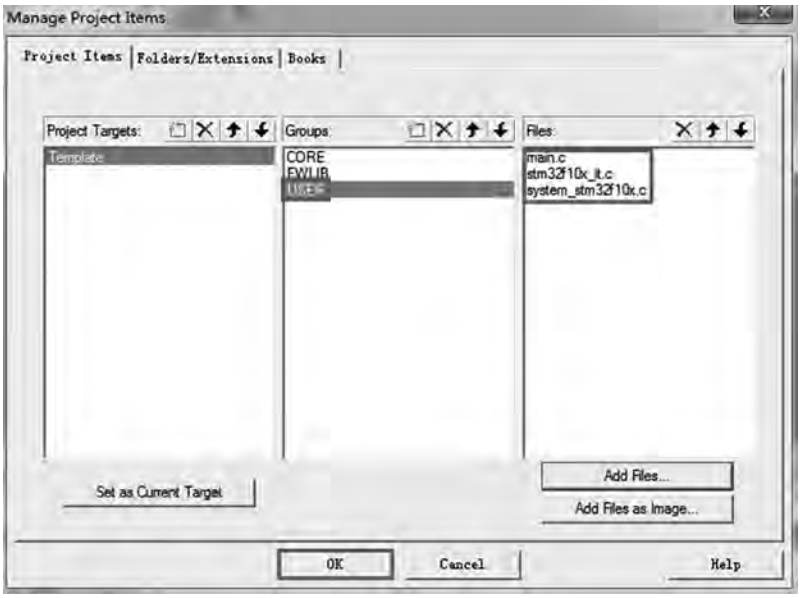

图 3-45　将相关文件添加到 USER 子文件夹中

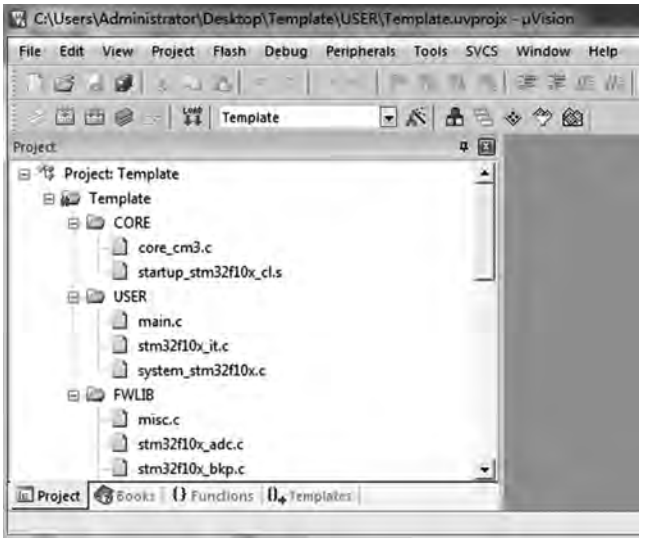

图 3-46　Template 工程目录结构

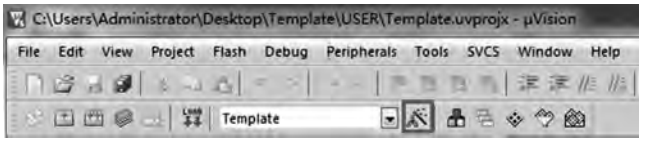

图 3-47　工程选项图标

图 3-48　选择生成工程的中间文件存放的目录

图 3-49　Build Target 图标

可以看到，在 MDK5 下方的 Build Output 框中出现了许多错误信息，如图 3-50 所示。

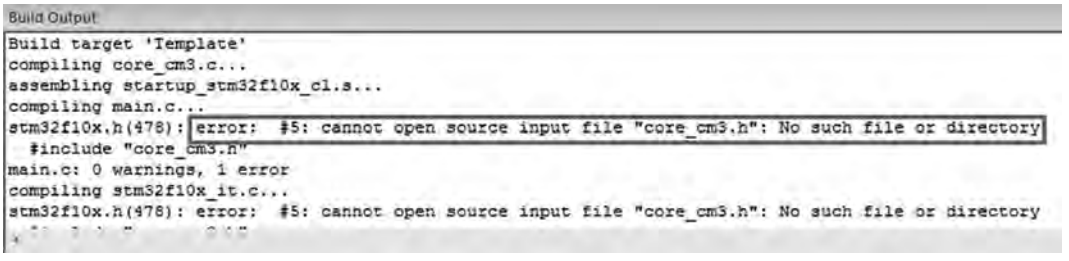

图 3-50　Build Output 框

这是因为，还没有将头文件的路径包含到工程中，也就是说，需要告诉 MDK 工程所使用到的每一个头文件的具体路径，否则 MDK 不会自动去寻找。

现在再次单击 图标，在弹出的对话框中，选择 C/C++ 标签，然后单击下方 Include Paths 文本框后边的“…”按钮，如图 3-51 所示。

在弹出的对话框中，添加工程中用到的头文件的具体路径，在 Template 工程目录中共有 3 个子文件夹，分别是 CORE、FWLIB\inc 和 USER，将它们添加进来，单击 OK 按钮，如图 3-52 所示。

这里需要注意的是，必须添加包含头文件的最后一级文件夹，因为 MDK 只会进入我们添加的某个文件夹的当前这一级去寻找头文件，而不会进入该文件夹的子文件夹中去寻找，

图 3-51　Options for Target 'Template'

图 3-52　添加头文件的路径

例如,FWLIB 文件夹下的 inc 子文件夹下包含工程要用到的头文件,那么必须添加 FWLIB/inc 而非 FWLIB。

因为 3.5 版本的库函数在配置和选择外设的时候是通过宏定义完成的,所以需要配置一个全局的宏定义变量。因此,在如图 3-51 所示的 Options for Target 'Template'对话框中,在上方 Preprocessor Symbols 下的 Define 文本框中输入"STM32F10X_CL,USE_STDPERIPH_DRIVER",单击 OK 按钮,如图 3-53 所示。

注意,这两个宏之间是逗号而不是句号。此外,STM32F10X_CL 对应的是 STM32F107 芯片,如果是其他类型的芯片,则需进行相应修改。

现在,再次尝试生成工程,可以看到,还是出现了一个错误,如图 3-54 所示。

stm32_eval.h 是意法半导体公司提供的几种测试评估样板相关的文件,这里用不到它,也没有包含它,所以会报错。可以将下面一段代码复制到 main.c 文件中。

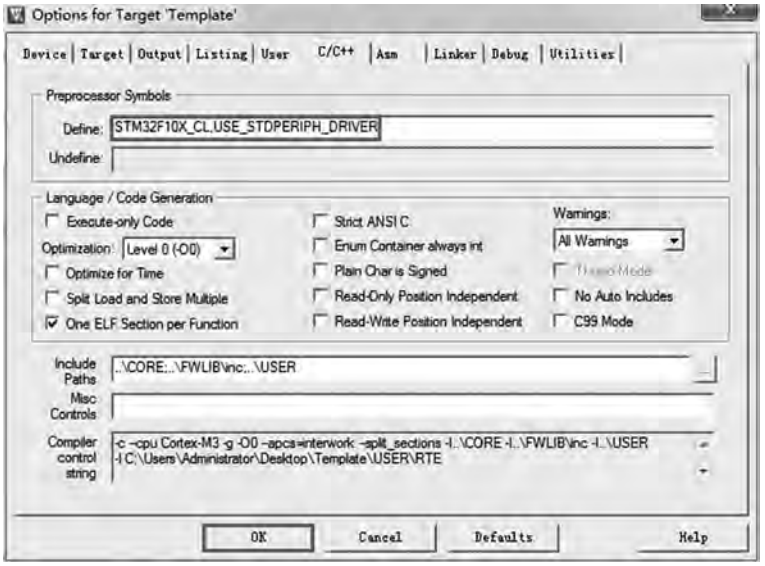

图 3-53　输入预处理器宏定义

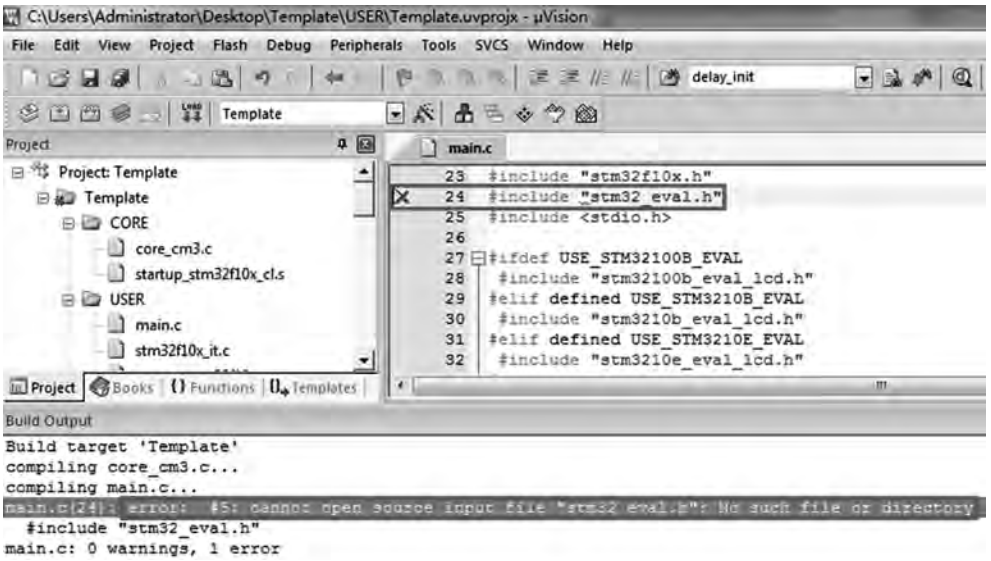

图 3-54　工程生成过程中报错

```
#include "stm32f10x.h"

void Delay(u32 count)
{
    u32 i = 0;
    while(i++< count);
}

int main(void)
```

```
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE, ENABLE);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_11 | GPIO_Pin_13 | GPIO_Pin_14;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOE, &GPIO_InitStructure);
    GPIO_SetBits(GPIOE, GPIO_Pin_9 | GPIO_Pin_11 | GPIO_Pin_13 | GPIO_Pin_14);

    while(1)
    {
     GPIO_ResetBits(GPIOE, GPIO_Pin_9 | GPIO_Pin_11 | GPIO_Pin_13 | GPIO_Pin_14);
        Delay(3000000);
        GPIO_SetBits(GPIOE, GPIO_Pin_9 | GPIO_Pin_11 | GPIO_Pin_13 | GPIO_Pin_14);
     Delay(3000000);
    }
}
```

然后再次生成工程,这次可以发现,工程生成成功,如图 3-55 所示。

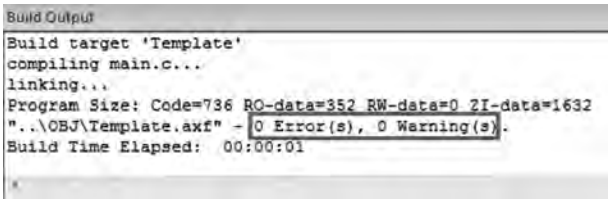

图 3-55 工程生成成功

3.5 程序的下载和调试

在建立完一个工程并编写好相应的程序后,就需要将程序下载到开发板中来观察运行结果。MDK 提供了强大的程序仿真功能,可以让程序在 MDK 中仿真运行或仿真调试。此外,也可以通过 JLINK 仿真器将程序下载到开发板中来对其进行硬件调试。

3.5.1 程序的下载

天信通 STM32F107 开发板共有 3 种程序下载的方式,分别为通过串口下载、通过 USB 下载和通过 JLINK 仿真器下载。

1. 串口下载

通过串口下载程序需要使用 ST 官方提供的 Flash Loader Demonstrator 软件,首先需要安装此软件,安装文件在 Flash_Loader_Demonstrator 文件夹中,安装步骤非常简单,只需要保留默认设置操作,安装完成后,在“开始”→“所有程序”→STMicroelectronics→Flash Loader Demonstrator 中就可以看到该软件,如图 3-56 所示。

下面介绍怎样用 Flash Loader Demonstrator 软件通过串口给开发板下载程序,操作步骤如下:

图 3-56　Flash Loader Demonstrator 软件

(1) 完成硬件方面的连接。先通过串口线或 USB 转串口线(接开发板的 RS232 接口)将开发板和 PC 连接起来,再通过跳线帽将开发板的串口下载跳线端子 J4 短接,最后通过 5V 电源适配器(不要用 USB 线)给开发板供电,如果一切正常,开发板电源指示灯 D6 会亮。

(2) 在“开始”→“所有程序”→STMicroelectronics→Flash Loader Demonstrator 中打开 Flash Loader Demo 软件,会出现如图 3-57 所示的串口参数设置界面,在界面中设置相应的串口参数。其中,Port Name(即 COM 口)需要根据实际情况来进行设置(可在 PC 的设备管理器的端口中查看,本例中为 COM3),其他串口参数可以按照图 3-57 进行设置,其中,Baud Rate(即波特率)一般设为 115 200bps,Timeout(超时时间)一般设为 5s。

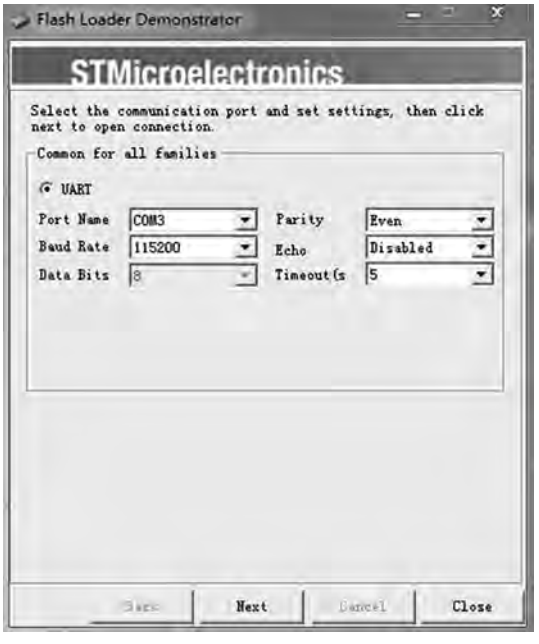

图 3-57　串口参数设置界面

(3) 按下开发板上的 RESET 按键(即 S5 键),然后松开,此时开发板就进入了串口下载模式,并等待着 PC 的连接,现在单击图 3-57 中的 Next 按钮,如果连接成功,则会出现如图 3-58 所示的连接成功界面;如果连接不成功,则会弹出相应的提示对话框,需要根据提示重新进行连接。

图 3-58 连接成功界面

(4) 如果连接成功,则在如图 3-58 所示的连接成功界面中单击 Next 按钮,会出现如图 3-59 所示的设备选择界面,在设备选择界面的 Target 后面选择 STM32_Connectivity-line_256K,并单击 Next 按钮。

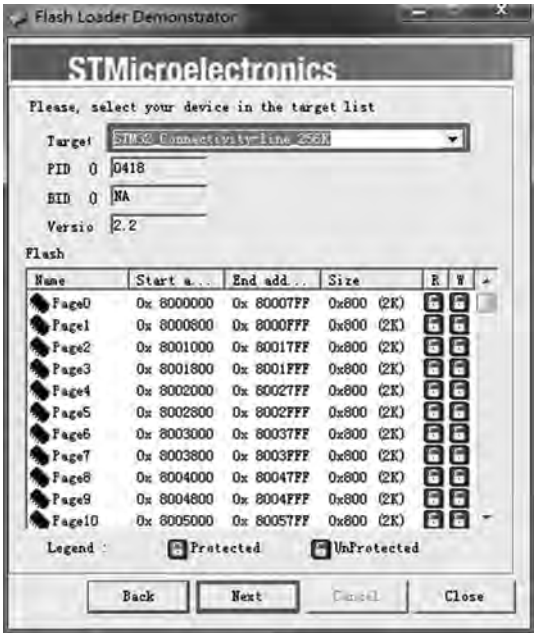

图 3-59 设备选择界面

(5) 在出现的如图 3-60 所示的操作选择界面中，可选择进行相关程序的擦除、下载、上传、Flash 保护的使能/失能以及选项字节配置等操作。这里主要介绍程序下载操作。在如图 3-60 所示的操作选择界面中，选中 Download to device 单选按钮，然后单击 Download form file 文本框右面的“…”按钮，选择要下载的. hex 文件，并在文本框下面选中 Erase necessary page 单选按钮，再选中 Jump to the user program 复选框，最后单击 Next 按钮。

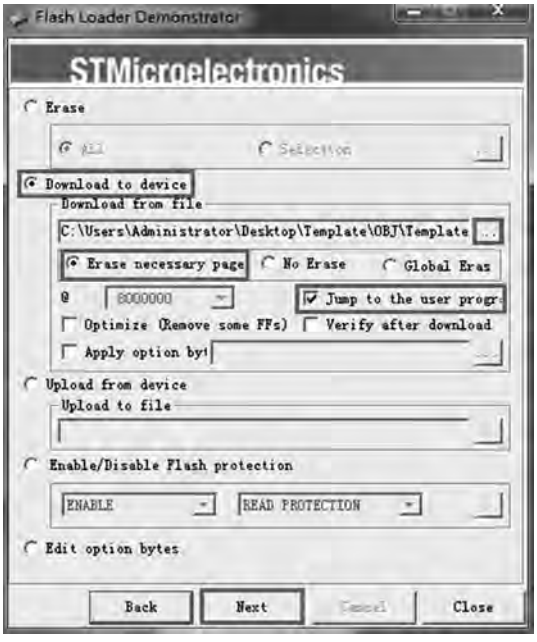

图 3-60　操作选择界面

下面具体说明以上的操作。

在单击 Download form file 文本框右面的“…”按钮选择要下载的. hex 文件时，会弹出如图 3-61 所示的对话框，需要将“查找范围”定位到要下载程序的工程目录下的 OBJ 子文件夹，因为 3.4 节新建工程模板时将此文件夹作为存放编译过程中所产生中间文件的位置，还需要将文件类型选择为 hex Files，然后选择之前在编译过程中生成的 Template. hex 文件，并单击“打开”按钮。

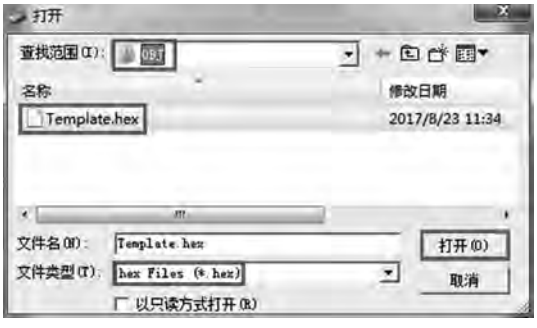

图 3-61　选择文件对话框

需要注意的是，在之前编译工程的时候，必须在工程选项的 Output 选项卡中选中 Creat HEX File 复选框，如图 3-62 所示，否则不会生成. hex 文件。

图 3-62　工程选项对话框

Download form file 文本框下面有 3 个选项，其中，如果选择 Erase necessary page，则在向开发板下载程序的过程中会首先擦除开发板主控芯片 Flash 中需要擦除的数据；如果选择 No Erase，则在下载过程中不会擦除 Flash 中的数据；如果选择 Global Erase，则在下载过程中会首先擦除 Flash 中所有的数据。很显然，应当选择 Erase necessary page。因为如果选择 No Erase，则下载完一次程序后 Flash 中可能还存有以前残留的数据没有被覆盖掉；如果选择 Global Erase，则每次下载程序都需要将 Flash 中的数据全部擦除一次，不仅没有必要，而且会减少 Flash 的使用寿命。这里需要说明的是，如果在如图 3-60 所示的操作选择页面中，先选中了 Erase 单选按钮将 Flash 中的相关数据擦除掉，则这里可以选择 No Erase。

最后，选中 Jump to the user program 复选框，会使程序在下载完成后，不需要从开发板的 J4 端子拿下跳线帽并且再按下 RESET 按键就可以直接在开发板中运行，在程序下载完成后还可以直接返回如图 3-57 所示的串口参数下载界面重新开始下一次的程序下载。

(6) 如果一切正常，会出现程序下载成功的界面，如图 3-63 所示。

然后，程序开始在开发板中运行。对于在 3. 4 节中建立的工程，其程序的运行结果是：LED 灯 D2、D3、D4 和 D5 大约以 200ms 的间隔不停地闪烁。

如果单击图 3-63 中的 Back 按钮，则会出现如图 3-57 所示的串口参数设置的界面，可以重新开始下一次的程序下载；若单击 Close 按钮，则关闭软件。

2. USB 下载

通过 USB 下载程序需要使用 ST 官方提供的 DfuSe Demonstration 软件，首先需要安装此软件，安装文件在 um0412_DfuSe_Demo_V3. 0 文件夹下面。

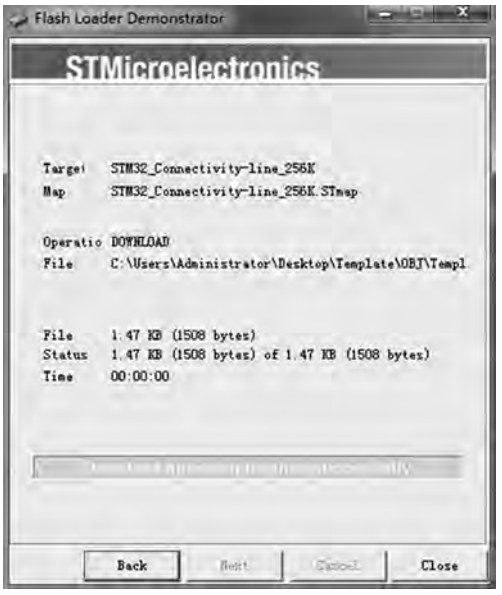

图 3-63　程序下载成功界面

安装步骤非常简单，只需要保留默认设置进行操作，安装完成后，在“开始”→“所有程序”→STMicroelectronics→DfuSe 中可以看到该软件，如图 3-64 所示。

图 3-64　DfuSe Demonstration 软件

下面介绍怎样用 DfuSe Demonstration 软件通过 USB 给开发板下载程序，具体操作步骤如下：

(1) 完成硬件方面的连接。先通过 Mini USB 线(接开发板的 Mini USB 接口)将开发板和 PC 连接起来，再通过跳线帽将开发板的串口下载跳线端子 J4 短接，最后通过 5V 电源适配器(或 USB 线)给开发板供电，如果一切正常，开发板电源指示灯 D6 会亮，PC 的 Windows Update 会自动给开发板的 STM 主控芯片安装 DFU 模式下的驱动，安装好后，可以在设备管理器的“通用串行总线控制器”下看到相关选项，如图 3-65 所示。

如果 PC 没有自动安装此驱动，则可以通过手动的方式对其进行安装，驱动文件在安装 DfuSe Demonstration 软件的根目录下的其中一个名为 Driver 的子文件夹下，具体路径为：

图 3-65　STM 芯片的 USB 驱动选项

C:\Program Files (x86)\STMicroelectronics\Software\DfuSe\Driver(选择默认路径安装),如图 3-66 所示,其中,32 位机选择 x86 文件夹,64 位机选择 x64 文件夹。

图 3-66　STM 芯片在 DFU 模式下驱动的安装文件的目录

(2) 通过“开始”→“所有程序”→STMicroelectronics→DfuSe→DfuSe Demonstration 菜单命令,打开 DfuSe Demonstration 软件,软件界面如图 3-67 所示。

通过这个软件,可以将一个工程的.dfu 类型的目标文件下载到开发板中,但通过 MDK 生成的一般都是.hex 类型的目标文件,怎样将.hex 类型的文件转化为.dfu 类型的文件呢?其实,在安装 DfuSe Demonstration 软件的时候,ST 公司已经给大家提供了相关的软件,就是如图 3-64 所示的 DFU File Manager。

(3) 通过“开始”→“所有程序”→STMicroelectronics→DfuSe→DFU File Manager 菜单命令,打开 DFU File Manager 软件,因为要通过一个.hex 文件生成一个.dfu 文件,所以在弹出的对话框中选择“I want to GENERATE a DFU file from S19, HEX or Binary”,然后单击 OK 按钮,如图 3-68 所示。

在弹出的对话框中,单击“S19 or Hex...”按钮,如图 3-69 所示。

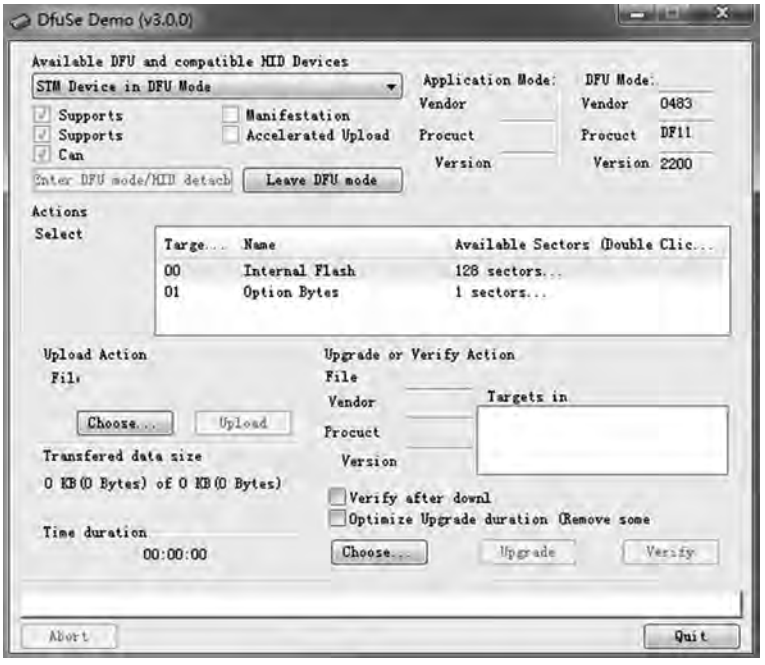

图 3-67 DfuSe Demonstration 软件界面

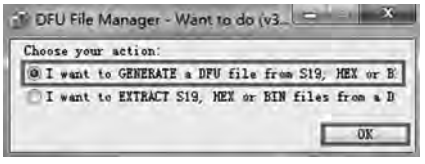

图 3-68 DFU File Manager-Want to do 对话框

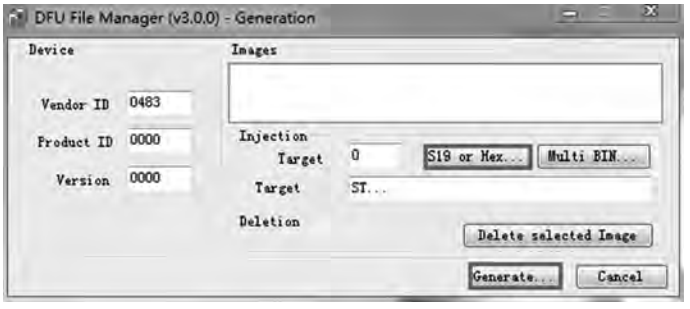

图 3-69 DFU File Manager-Generation 对话框

在弹出的对话框中,“文件类型”选择 hex Files,然后找到 3.4 节新建工程目录下的 OBJ 子文件夹,选择 Template.hex 文件,单击“打开”按钮,如图 3-70 所示。

然后在图 3-69 中单击 Generate 按钮,在弹出的对话框中选择要生成的.dfu 文件的名字和要存放的位置,选择文件名为 Generate.dfu,存放在 OBJ 文件夹下,然后单击“保存”按钮,如图 3-71 所示。

图 3-70　查找文件对话框

图 3-71　选择生成的.dfu 文件的名字和存放路径

如果弹出如图 3-72 所示的对话框，则表明相关的.dfu 文件成功生成。

图 3-72　.dfu 文件成功生成

(4) 现在再回到如图 3-67 所示的 DfuSe Demonstration 软件界面，单击下面的 Choose 按钮，在弹出的对话框中选择刚才生成的.dfu 文件，然后单击 Upgrade 按钮，并在弹出的对话框中单击“是”按钮，如图 3-73 所示。如果一切顺利，会显示如图 3-74 所示的 Upgrade successful 的界面。

现在，将开发板 J4 端子的跳线帽取下，并按下开发板的 RESET 按键，可以看到，程序开始在开发板中运行，LED 灯 D2、D3、D4 和 D5 大约以 200ms 的间隔不停地闪烁。

3. JLINK 下载

前面分别介绍了如何通过串口和 USB 来给开发板下载程序。除了这两种下载方式外，还可以通过 JLINK 仿真器来给开发板下载程序。下面以 JLINK V8 仿真器为例，介绍怎样通过 JLINK 仿真器来给开发板下载程序，JLINK V8 仿真器及其相关数据连接线如图 3-75 所示。

图 3-73 Upgrade 界面

图 3-74 Upgrade successful 界面

首先,需要安装 JLINK 仿真器的驱动程序。将 JLINK 仿真器通过其 USB 端的数据连接线连接到 PC,如果 PC 连接到 Internet,那么 PC 会自动给 JLINK 仿真器安装相关的驱动程序,安装好后,在设备管理器的通用串行总线控制器下可以看到 JLINK 仿真器的驱动选项,如图 3-76 所示。

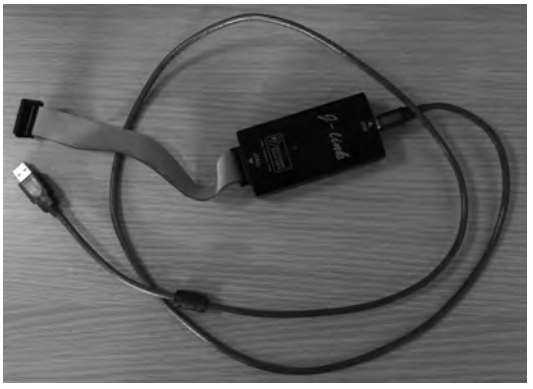

图 3-75　JLINK V8 仿真器及其相关数据连接线

图 3-76　JLINK 仿真器的驱动图标

操作步骤如下：

(1) 通过 JLINK 仿真器及其相关数据连接线将开发板和 PC 连接起来(JLINK 仿真器的 JTAG 一端的数据连接线连接到开发板的 JTAG 接口，即 J16)，通过 5V 电源适配器给开发板供电，如果一切正常，那么开发板的电源指示灯 D6 会亮。

(2) 用 MDK 打开相关的工程文件，单击工具栏上的 图标，打开 Options for Target 对话框，选择 Debug 选项卡，然后选中 Use 单选按钮，并在其对应的下拉列表框中选择 J-LINK/J-TRACE Cortex，如图 3-77 所示。

(3) 在弹出的对话框中，选择 Debug 选项卡(默认选项)，可以看到 JLINK 仿真器的相关信息，然后在下面的“ort:”下拉列表框中选择 SW，在 Max 下拉列表框中选择 10MHz，并单击“确定”按钮，如图 3-78 所示。“ort:”对应的是 JLINK 的下载模式。注意，JLINK V8 仿真器支持 JTAG 和 SWD 两种下载/调试模式，同时 STM32 也支持这两种模式，但因为

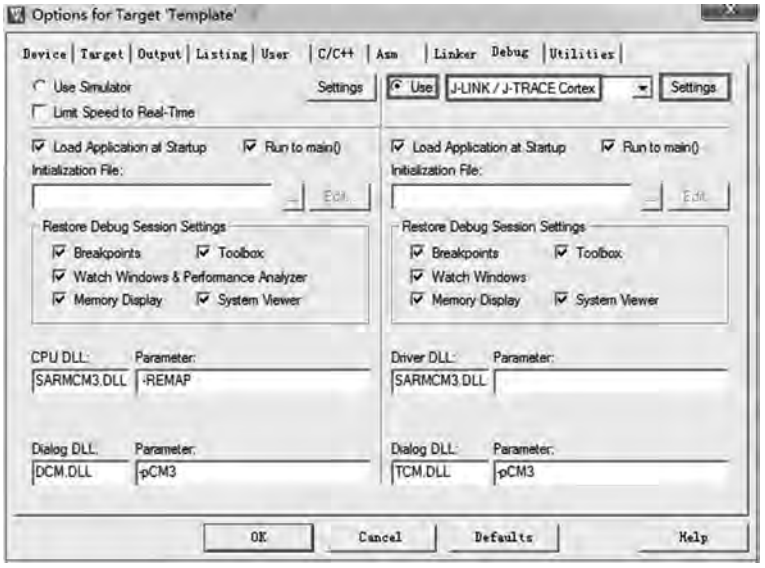

图 3-77　Debug 选项卡设置

SWD 只需要占用两根 I/O 口线，JTAG 模式则需要占用太多的 I/O 口线，所以，这里选择用 SWD 模式。Max 对应的是下载/调试的最大时钟频率，可以通过单击它下面的 Auto Clk 按钮来对其自动进行设置，这里一般设置为 10MHz 即可。注意，如果 USB 数据线性能比较差，那么这里可能会出问题，可以尝试通过设置更低的最大时钟频率来解决此问题。

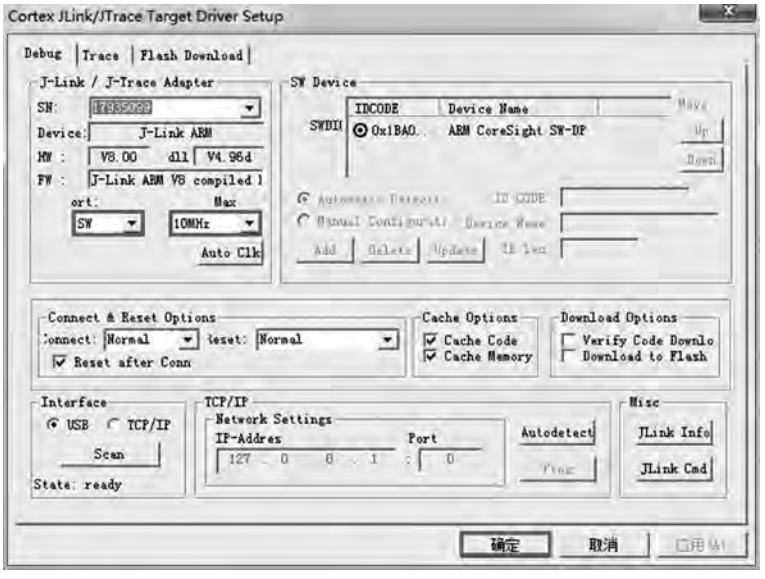

图 3-78　JLINK 模式设置

（4）回到 Options for Target 'Template'对话框中，选择 Utilities 选项卡，并在它下面选中 Use Debug Driver 复选框，表示选择 JLINK 来给开发板的 Flash 编程，然后单击 Settings 按钮，如图 3-79 所示。

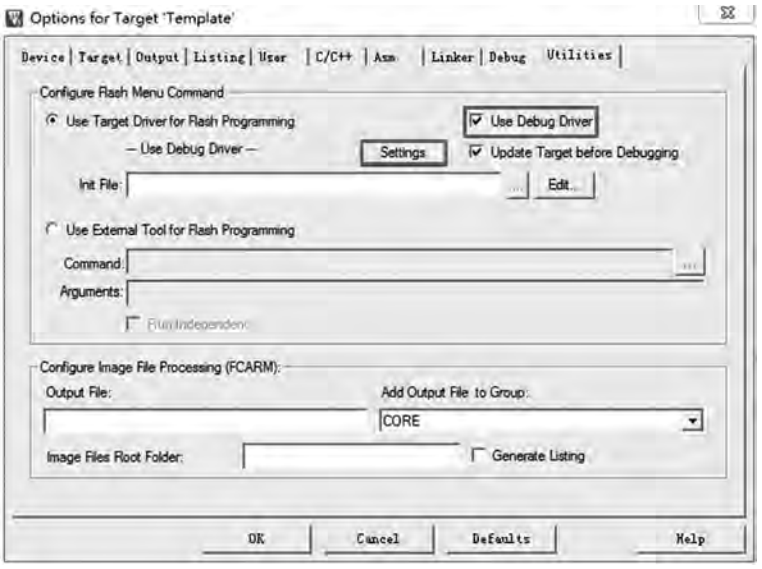

图 3-79　Utilities 选项卡设置

在弹出的如图 3-80 所示的 Flash 编程算法设置对话框中，MDK 在一般情况下会根据新建工程时选择的器件类型，自动设置 Flash 的编程算法，因为选择的是 STM32F107VCT6，Flash 容量是 256KB，所以在 Programming Algorithm 下面的列表中，默认会出现容量为 256K 的 STM32F10x Connectivity Line Flash 的算法，如果没有出现，则需要单击下面的 Add 按钮，在弹出的对话框中选择合适的算法手动进行添加。然后，选中 Reset and Run 复选框，这样通过 JLINK 下载完程序后，程序会自动在开发板中运行。最后，单击“确定”按钮，回到 Options for Target 对话框中，再单击 OK 按钮，就完成了 JLINK 下载需要的所有设置。

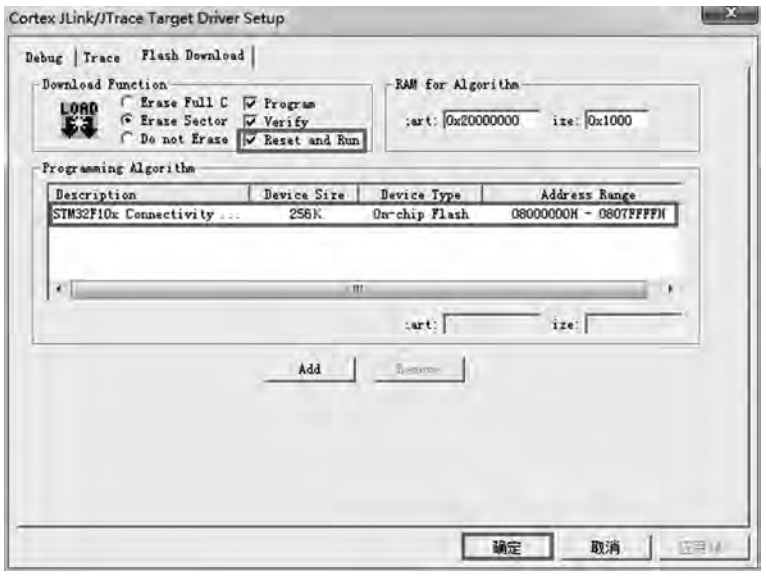

图 3-80　Flash 编程算法设置

(5) 在 MDK 的工具栏中单击 Download 图标，如图 3-81 所示，经过一段下载过程后，在 MDK 的 Build Output 窗口中会显示下载成功，如图 3-82 所示。

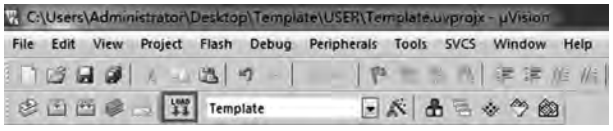

图 3-81　Download 图标

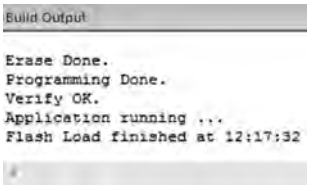

图 3-82　通过 JLINK 下载成功

需要注意的是，用 JLINK 下载程序时，开发板的串口跳线端子 J4 无须用跳线帽连接，如果用跳线帽连接了，则在下载完成后，程序不会自动在开发板中运行，这时，只需取下跳线帽，然后按下 RESET 按键即可。

3.5.2　程序的调试

在将程序下载到开发板后，程序开始在开发板中运行。但其运行结果不一定会像我们期望的那样，这可能是因为编写的程序有问题，要想快速找到问题所在，尤其是在大型程序中，就需要对程序进行调试。调试方法有两种：通过 MDK 的软件仿真调试和通过 JLINK 的硬件实际调试。

MDK 的一个强大之处就在于它为程序提供了强大的软件仿真调试功能，通过软件仿真调试，可以观察程序模拟运行的整个过程，并在此过程中观察硬件相关的许多寄存器中的值的变化，通过这种方式，不必频繁地向开发板下载程序就能发现程序的问题，这在一定程度上延长了开发板主控芯片 Flash 的使用寿命(STM32 芯片 Flash 的使用寿命一般在 10 000 次读写操作左右)。但是，软件仿真调试毕竟不是万能的，许多问题必须通过硬件实际调试才能够发现。

但 MDK 目前只支持对 STM32F103 系列芯片的软件仿真调试，对其他系列的芯片，则不支持或只是部分支持，主要存在的问题是：PC 和 SP 不能被自动装载，存储器不能被访问，中断服务程序不能被执行或触发，外设寄存器不能被读或写等。要解决上述问题，需要进行相应的设置和操作，这里不对其进行详述。

下面介绍怎样用 JLINK 对程序进行硬件的实际调试。

用 MDK 打开 3.4 节建立的工程文件，在进行调试之前，还需要进行相关的设置。在工具栏上单击 图标，打开 Options for Target 'Template'对话框，选择 Target 选项卡，确认芯片类型是否正确，如图 3-83 所示。

然后在 Options for Target 'Template'对话框中选择 Debug 选项卡，在它的下面选中 Use 单选按钮，并在其对应的下拉列表框中选择 J-LINK/J-TRACE Cortex，这里与用 JLINK 下载时的设置相同，接着选中 Run to main()复选框，然后在下方的两个 Dialog DLL 文本框中分别输入 DARMSTM.DLL 和 TARMSTM.DLL，在两个 Parameter 文本框中都输入-pSTM32F107VC，最后单击 OK 按钮，如图 3-84 所示。选中 Run to main()复选框，则在对程序进行调试时，程序指针开始会跳过 startup_stm 32f10x_cl.s 启动文件中的相关代码而直接指向 main()函数的起始处，否则，程序指针会指向 startup_stm32f10x_cl.s 启动文件

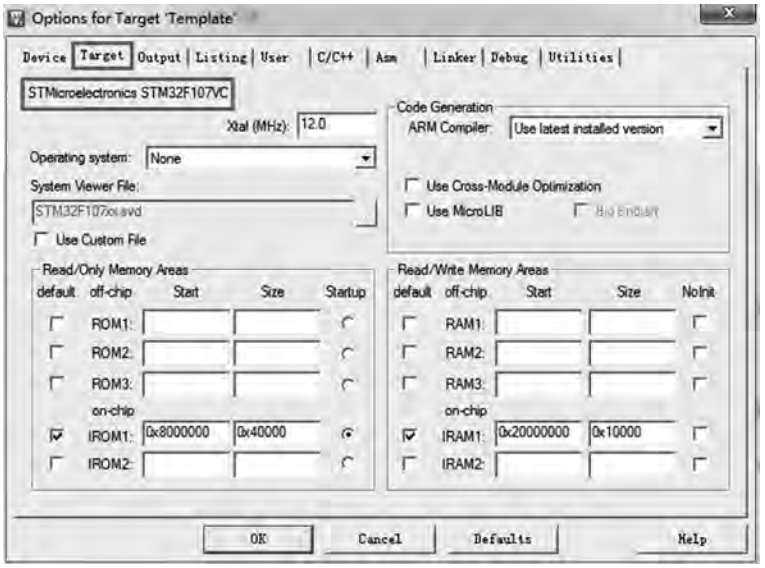

图 3-83　Target 选项卡

的 Reset_Handler 处。两个 Dialog DLL 和两个 Parameter 的设置，则用于支持 STM32F107VC 的软硬件仿真，即可以通过单击 MDK 菜单栏的 Peripherals 图标，在弹出的相关外设的对话框中观察程序运行的结果。

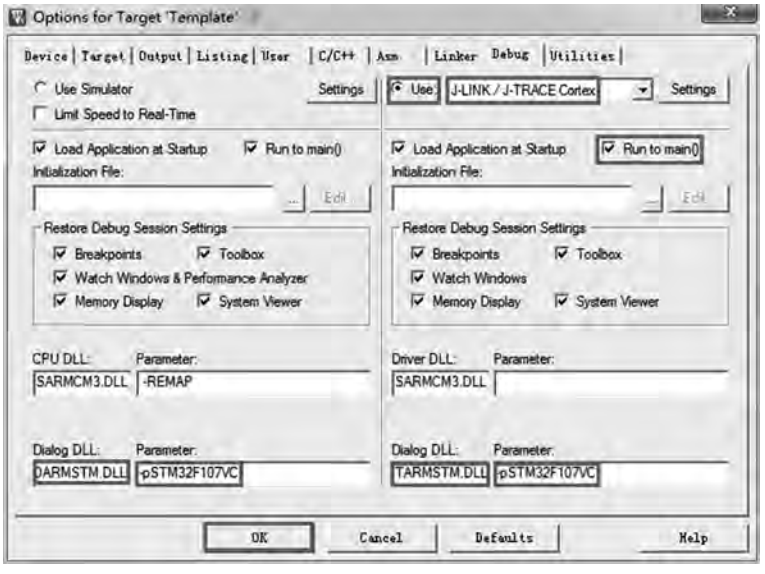

图 3-84　Debug 选项卡

接下来，单击 MDK 工具栏上的调试图标 ，对程序进行调试(在对程序进行调试前不要忘记先编译工程)，如图 3-85 所示。

可以看到，MDK 的工具栏中出现了一行新的用于调试操作的图标，将鼠标指针置于这些图标之上，可以看到对这些图标所对应操作的简单说明，程序指针指向 main()函数的起始处，如图 3-86 所示。

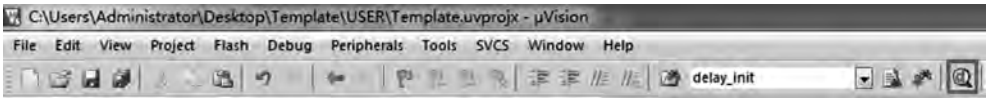

图 3-85　调试图标

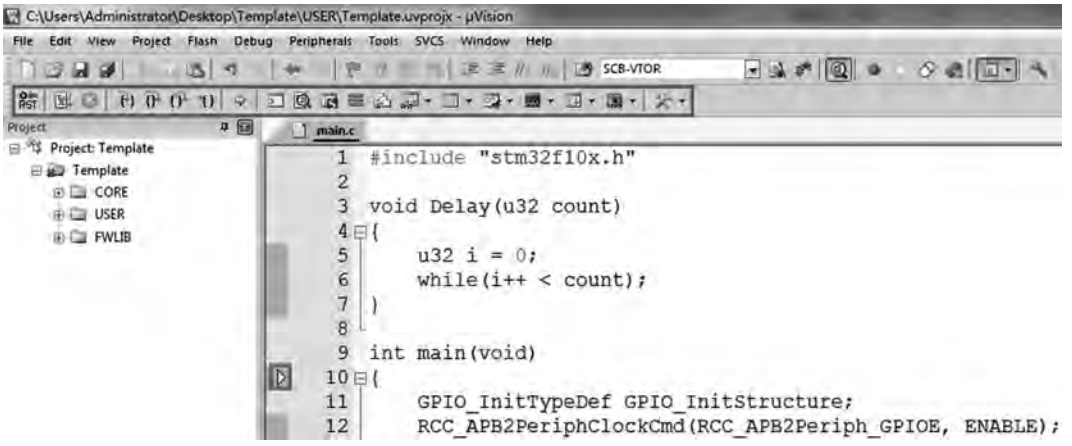

图 3-86　进入调试模式

接着将这行用于调试操作的图标单独提取出来，如图 3-87 所示。

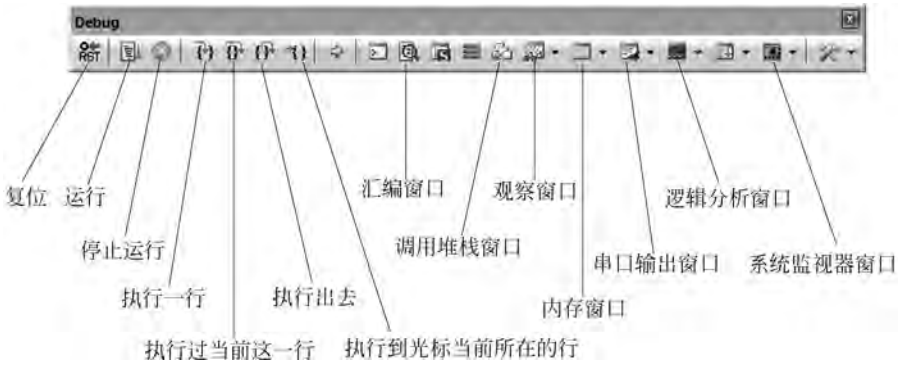

图 3-87　Debug 工具条

下面对这些图标进行详细介绍：

图标表示 Reset(Reset the CPU)，即复位，单击该图标将使程序指针指向整个程序的起始处，即 startup_stm32f10x_cl. s 启动文件的 Reset_Handler 处(进入如图 3-76 所示的调试模式时，程序指针指向 main()函数的起始处，是由于之前在如图 3-74 所示的 Options for Target 'Template'对话框中选中了 Run to main()复选框)。这相当于按下开发板上的 RESET 按键，实现了一次硬件复位操作。

图标表示 Run(Start code execution)，即运行，单击该图标将使程序在开发板中全速运行。

图标表示 Stop(Stop code execution)，即停止运行，单击该图标将使程序停止运行，程序指针指向下一条将要执行的语句。

图标表示 Step(Step one line),即执行一行,单击该图标,如果程序指针当前指向的行没有函数调用语句,那么将一次性执行完这一行的所有语句,然后程序指针会指向下一行的第一条语句;如果程序指针当前指向的行有函数调用语句,那么程序指针将会进入该行第一个被调用的函数中,并指向该函数的第一条语句,对于一行有多个函数调用语句的情况,持续单击该图标,将会在执行完第一个被调用函数的所有语句之后,再进入后面被调用的函数。注意,以上描述考虑了一行有多条语句的情况,对于一行只有一条语句的情况当然也是适用的。在一般情况下,程序中一行只有一条语句,尤其是对于函数调用语句。

图标表示 Step Over(Step over the current line),即执行过当前这一行,单击该图标,不管程序指针当前指向的行有没有函数调用语句,都将一次性执行完该行的所有语句,然后程序指针指向下一行的第一条语句。对于当前行没有函数调用语句的情况,它与 图标作用相同。对于当前行有函数调用语句的情况,它会将所有函数调用的过程都分别当作一条语句来执行,而不会分别进入各个被调用的函数中,并且会一次性执行完当前行的所有语句。注意,程序中一行一般只有一条语句。

图标表示 Step Out(Step out of the current function),即执行出去,它与 图标相对应,它们往往结合使用。单击 图标使程序指针进入某个被调用的函数中,单击 图标将使程序从该函数中程序指针当前指向的语句处一次性一直执行到函数结尾处,然后程序指针会指向调用该函数语句的下一条语句。

图标表示 Run to Cursor Line(Run to the current cursor line),即执行到光标当前所在的行,单击该图标,将使程序从程序指针当前指向的语句处一直执行到光标当前所在行的上一条语句,程序指针会指向光标当前所在的行的第一条语句。

图标表示 Dissemble Window(Show or hide the Dissemble Window),即汇编窗口,单击该图标,可以打开(或关闭)汇编窗口,查看相应的汇编程序。

图标表示 Call Stack Window(Show or hide the Call Stack Window),即调用堆栈窗口,单击该图标,可以打开(或关闭)调用堆栈窗口,查看被调用的堆栈。

图标表示 Watch Window(Show or hide the Watch Window),即观察窗口,单击该图标,可以打开(或关闭)观察窗口,通过该窗口可以观察到需要观察的程序中的变量。

图标表示 Memory Window(Show or hide the Memory Window),即内存窗口,单击该图标,可以打开(或关闭)内存窗口,通过该窗口可以观察到需要观察的内存中的数据。

图标表示 Serial Window(Show or hide the Serial Window),即串口输出窗口,单击该图标,可以打开(或关闭)串口输出窗口,通过该窗口可以观察到需要观察的串口输出的数据。

图标表示 Logic Analysis Window(Show or hide the Logic Analysis Window),即逻辑分析窗口,单击该图标,可以打开(或关闭)逻辑分析窗口,通过在该窗口中单击 Setup 按钮,可加入一些需要观察的 I/O 口,在程序调试的过程中,可以非常直观地观察这些 I/O 口的电平状态。

图标表示 System Viewer Window(Show or hide the System Viewer Window),即系统监视器窗口,单击该图标,可以打开(或关闭)系统监视器窗口,通过该窗口可以观察系统的各种特殊功能寄存器中的值。

此外，还可以通过在程序中设置断点的方式，来帮助我们调试程序，这需要通过单击工具栏上的●图标，如图 3-88 所示。添加断点后，当程序运行到断点处时，就会停止运行。

图 3-88　添加/删除断点图标

下面结合程序来具体介绍对这些操作的应用。

首先，单击 Debug 工具条中的图标，打开观察窗口 Watch1，然后在 Name 一列输入 GPIO_InitStructure，它是程序中的一个变量的名称，也可以直接右击程序中的该变量，选择 Add ‘GPIO_InitStructure’ to…→Watch1 命令，可以看到，在窗口中会显示出该变量的值和类型，因为 GPIO_InitStructure 是一个结构体类型的变量，所以可以在窗口中单击它左边的小加号，查看其成员变量，如图 3-89 所示。

Watch 1

Name	Value	Type
GPIO_InitStructure	0x20000658	struct <untagged>
GPIO_Pin	0x0000	unsigned short
GPIO_Speed	0x00	enum (uchar)
GPIO_Mode	0x00 GPIO_Mode_AIN	enum (uchar)
<Enter expression>		

图 3-89　观察窗口

然后，将光标置于程序的第 16 行，然后通过单击工具栏上的●图标给程序在这一行添加一个断点（也可以通过右击该行或在该行最左边的灰色区域单击的方式来添加），如图 3-90 所示。

main.c

```
1  #include "stm32f10x.h"
2
3  void Delay(u32 count)
4  {
5      u32 i = 0;
6      while(i++ < count);
7  }
8
9  int main(void)
10 {
11     GPIO_InitTypeDef GPIO_InitStructure;
12     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE, ENABLE);
13     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_11 | GPIO_Pin_13 | GPIO_Pin_14;
14     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
15     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
16     GPIO_Init(GPIOE, &GPIO_InitStructure);
17     GPIO_SetBits(GPIOE, GPIO_Pin_9 | GPIO_Pin_11 | GPIO_Pin_13 | GPIO_Pin_14 );
```

图 3-90　添加断点

注意，显示行号可以通过以下操作：单击 MDK 工具栏的 Edit→Configuration 菜单命令，在弹出的对话框中选择 Edit 选项卡（默认选项），然后在对话框中左下方的 C/C++Files 区域中，选中 Show Line Numbers 复选框，如图 3-91 所示。

通过该对话框可以设置程序中文本的字体、大小、颜色等，还可以设置文本的编码格式、Tab 键的空格数、关键字等，这样可以方便大家更好地编写程序。

在图 3-90 中，程序第 16 行最左边的灰色区域会出现一个红色的实心圆点，表示该行已添加了一个断点，如果再次单击该行最左边的灰色区域，则该实心圆点会消失，表明删除了

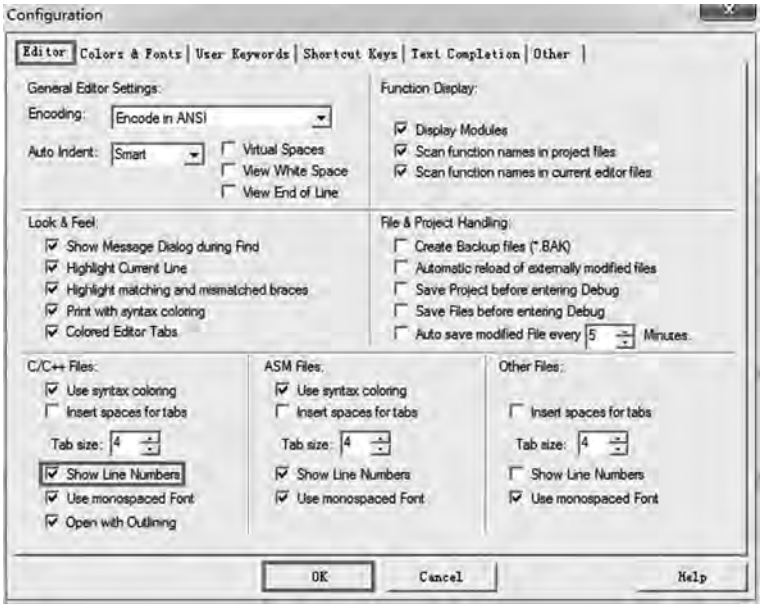

图 3-91　Configuration 对话框

该断点。注意,红色实心圆点左边有一个浅蓝色的小三角,表示光标位于该行。

单击 Debug 工具栏中的图标,程序开始运行,并停止在我们添加的断点处,如图 3-92 所示。

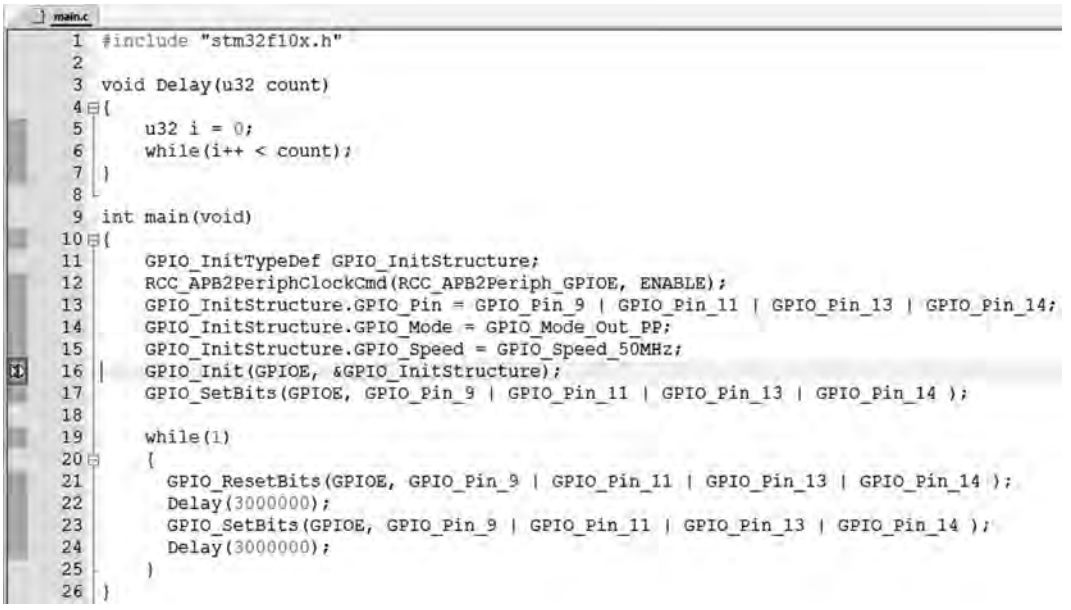

图 3-92　程序停止在断点处

注意,这时在观察窗口 Watch1 中,可以看到变量 GPIO_InitStructure 的 3 个成员变量的值发生了变化,它们的区域底色也变为深绿色以作为提示,如图 3-93 所示。

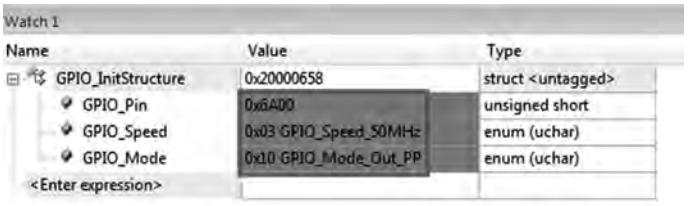

图 3-93　观察窗口

现在，大略浏览如图 3-92 所示的 main()函数中的第 11～15 行的代码。初学者可能现在还不能理解这几行代码的具体含义，但是有一定 C 语言基础的读者应该能大体看出，这几行代码的作用是：定义了一个 GPIO_InitTypeDef 结构体类型的变量 GPIO_InitStructure，然后给它的 3 个成员变量 GPIO_Pin、GPIO_Mode 和 GPIO_Speed 赋值，所以在图 3-93 中，这 3 个成员变量的值分别由如图 3-89 所示的 3 个 0 变为了赋值后的结果。

通过这种方法，可以观察程序中各个变量的值的变化，如果想将某个变量从观察窗口的列表中删除，可以右击该变量，选择 Remove Watch 命令。

在 Debug 工具栏中单击 图标，选择 GPIO→GPIOE 选项，如图 3-94 所示。

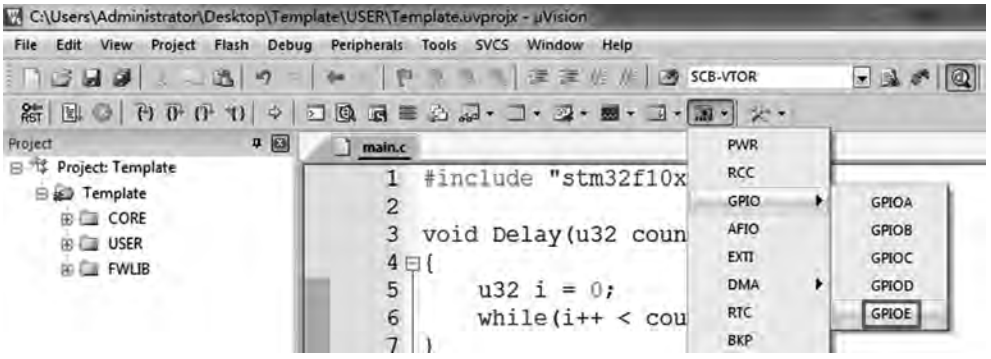

图 3-94　选择 GPIO→GPIOE 选项

在 MDK 的右侧出现了一个名为 GPIOE 的窗口，其中列出了与 GPIOE 相关的 7 个寄存器，如图 3-95 所示。

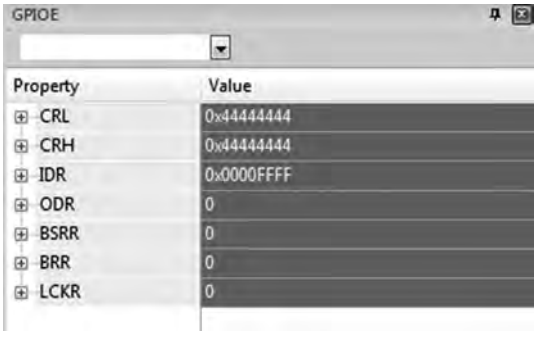

图 3-95　GPIOE 相关寄存器窗口(1)

单击 Debug 工具栏中的图标，单步执行图 3-92 中的第 16 行语句，可以发现，GPIOE 窗口中有些寄存器的值发生了变化，如图 3-96 所示。

图 3-96　GPIOE 相关寄存器窗口(2)

再次单击 Debug 工具栏中的图标，单步执行图 3-92 中的第 17 行语句，可以发现，GPIOE 窗口中的值又有改变，如图 3-97 所示。

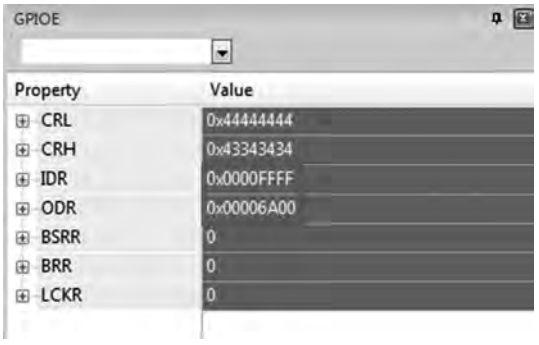

图 3-97　GPIOE 相关寄存器窗口(3)

然后，继续不断地单击 Debug 工具栏中的图标，可以发现，程序在图 3-92 中程序的第 19～25 行之间循环反复地执行，而 GPIOE 相关寄存器中的值也在图 3-96 和图 3-97 之间不断地变化，这实际上也对应了程序的运行结果——4 个 LED 灯不停地闪烁。当大家学习完本书第 3 篇中的 5.4.1 节后，再来看以上的程序调试过程，应该会有更深刻的理解。

本章小结

本章对软件开发环境——MDK5 做了讲解，对开发环境的安装、新建工程、下载调试做了详细的说明。首先介绍 ST 官方固件库；接着介绍 MDK5 软件及其安装和注册的过程；然后介绍怎样基于 ST 官方固件库来新建一个工程模板；最后介绍针对开发板下载程序的 3 种方式，以及在 MDK5 中对程序进行调试的方法。