第 3 章

软件环境配置与使用入门

本章要点

- STM32 开发方式；
- 软件资源安装与配置；
- 基于 CubeMX 的 HAL 库开发模式；
- CMSIS-DAP 调试器使用；
- 编译器优化与 Volatile 关键字。

基于 STM32CubeMX 的 HAL 库的嵌入式系统开发之所以高效、快捷、可移植性强，在一定程度上得益于软件开发平台的高效配置。相比于传统的寄存器或是标准库开发，HAL 库开发的软件环境配置更为复杂，涉及软件众多，甚至成为嵌入式学习的拦路虎。为帮助读者顺利搭建适合自己的嵌入式软件平台，本章将详细介绍各相关软件的安装、配置和使用方法。

微课视频

3.1 STM32 开发方式

嵌入式系统软件设计的编程语言分为汇编语言和高级语言两种，目前广泛使用 C 语言进行嵌入式系统应用开发，而依据开发库的不同，STM32 开发方式又可以划分为 STM32Snippets 库、标准外设库(Standard Peripheral Library)、STM32Cube HAL 库、STM32Cube LL 库 4 种。

3.1.1 STM32Snippets 库

STM32Snippets 可翻译为“代码片段”，其实就是常说的“寄存器”开发 STM32 的底层驱动代码。STM32Snippets 库是高度优化的示例代码集合，使用符合 CMSIS 的直接寄存器访问来减少代码开销，从而在各种应用程序中最优化 STM32 微控制器的性能。

早期的 51 单片机、AVR 单片机和 PIC 单片机均采用的是寄存器开发方式。尤其是我们十分熟悉的 51 单片机，开发一个 51 单片机程序，一般是采用汇编语言或是 C 语言编写控制程序，操作相应寄存器(例如 P1、IE、IP、TMOD、T1 等)，实现相应控制功能。

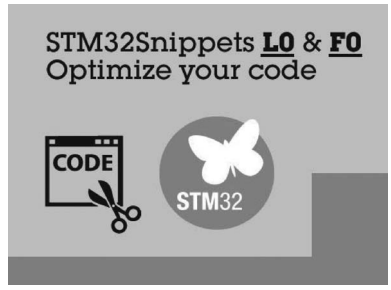

图 3-1 STM32Snippets 库

ST 官方仅提供 STM32F0 和 STM32L0 两个系列的 STM32Snippets 库，如图 3-1 所示，但寄存器开发方式适用所有的 STM32 微控制器。对于没有提供 STM32Snippets 库的微控制器开发时只需包含该系列芯片的寄存器定义头文件便可使用寄存器开发方式，而这一步操作往往是在工程模板创建时已经完成。

虽然寄存器开发方式直接、高效，但是 STM32 片上资源十分丰富，要记住每个寄存器名称和操作方式是十分困难的，且编写出来的程序可读性、可维护性和可移植性都比较差。因此，除对速度要求较高和需要反复执行的代码外，一般不使用寄存器开发方式。值得注意的是，寄存器开发方式是其他一切开发方式的基础，所有开发模式本质上操作的是寄存器，有时在其他开发模式中直接操作寄存器会起到事半功倍的效果。

3.1.2 标准外设库

为帮助嵌入式工程师从查找、记忆芯片手册中解脱出来，ST 公司于 2007 年推出标准外设库(Standard Peripheral Libraries，SPL)，也称标准库，STM32 标准库是根据 ARM Cortex 微控制器软件接口标准(Cortex Microcontroller Software Interface Standard，CMSIS)而设计的。CMSIS 标准由 ARM 和芯片生产商共同提出，让不同的芯片公司生产的 ARM Cortex-M 微控制器能在软件上基本兼容。

STM32 标准库是一个或一个以上的完整的软件包(称为固件包)，包括所有的标准外设的设备驱动程序，其本质是一个固件函数包(库)，由程序、数据结构和各种宏组成，包括了微控制器所有外设的性能特征。该库还包括每一个外设的驱动描述和应用实例，为开发者访问底层硬件提供了一个中间 API(Application Programming Interface，应用编程接口)。通过使用标准库，开发者无须深入掌握底层硬件细节，就可以轻松应用每一个外设，就像在标准 C 语言编程中调用 printf()一样。每个外设驱动都由一组函数组成，这组函数覆盖了该外设的所有功能。每个器件的开发都由一个通用 API 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。

标准外设库如图 3-2 所示，目前，其支持 STM32F0、STM32F1、STM32F2、STM32F3、STM32F4、STM32L1 系列，不支持 STM32F7、STM32H7、STM32MP1、STM32L0、STM32L4、STM32L5、STM32G0、STM32G4 等后面推出的系列。

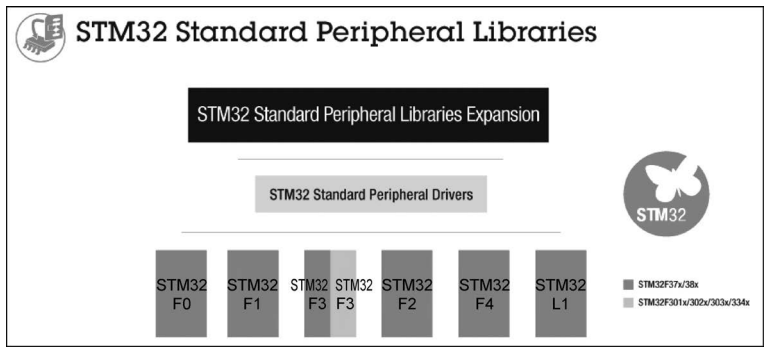

图 3-2 标准外设库

3.1.3 STM32CubeMX HAL 库

标准外设库不仅明显降低了开发门槛和难度，缩短了开发周期，降低开发成本，而且提高了程序的可读性和可维护性，给 STM32 微控制器开发带来极大的便利。但是不同系列微控制器的标准库是不通用的，差别较大，给代码复用和程序移植带来挑战，进而影响项目开发效率。

2014 年 ST 公司推出硬件抽象层(Hardware Abstraction Layer，HAL)库，HAL 库比标准库抽象性更好，所有 API 具有统一的接口。STM32Cube HAL 库如图 3-3 所示，基于 HAL 库的程序可以在 STM32 全系列微控制器内迁移，可移植性好；借助图形化配置工具 STM32CubeMX 可以自动生成初始化代码和工程模板，高效便捷，是 ST 公司主推的一种开发方式。

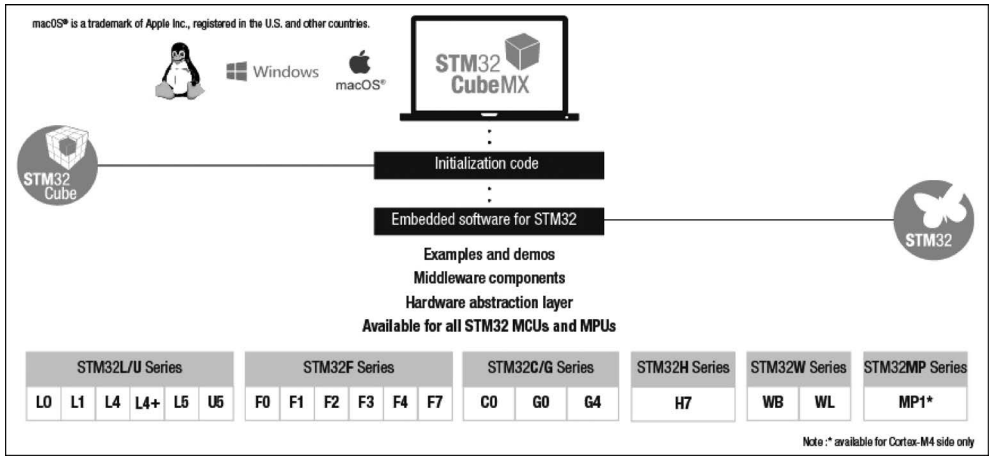

图 3-3　STM32Cube HAL 库

3.1.4　STM32CubeMX LL 库

基于 STM32CubeMX 的 HAL 库开发高效、快捷，支持 STM32 全系列产品，但是其抽象层次高，多层函数嵌套，代码冗余度相对较高，对芯片容量小，性价比要求高的应用场合有时难以胜任。所以，ST 公司对于部分产品在推出其 HAL 库时，同步推出底层（Low-Layer，LL）库，LL 库中大部分代码直接操作寄存器，更接近硬件，代码量少，但应用方法和 HAL 库并无区别。

3.1.5　开发方式比较与选择

ST 公司提供的 4 种嵌入式开发方式各有所长，为帮助初学者选择合适的开发方式，下面对 4 种开发方式进行比较，并给出作者的选择建议。

1. 开发方式比较

表 3-1 分别列出了 4 种开发方式在可移植性（Portability）、优化程度（Optimization Memory&Mips）、易用性（Usability）、意愿性（Readiness）和硬件覆盖（Hardware Coverage）程度方面的对比结果。

表 3-1　开发方式对比

开　发　库		可移植性	优化程度	易　用　性	意　愿　性	硬件覆盖程度
CODE STM32Snippets 库			+++			+
标准库		++	++	+	++	+++
STM32 CubeMX	HAL 库	+++	+	++	+++	+++
	LL 库	+	+++	+	++	++

由表 3-1 可知，STM32Snippets 库（寄存器）开发方式除优化性能方面表现较好，其他性能均较差，所以目前已较少为嵌入式工程师使用。标准库各方面性能均处于中间位置，有着不错的性能表现，另外由于标准库推出较早，在 STM32 嵌入式开发中获得了广泛的应用。HAL 库除优化程度表现欠佳外，其他方面均具有最优性能，综合性能最好，是未来嵌入式开发的发展方向。LL 库除优化程度这一性能指标和 HAL 库的表现相反外，其他性能表现的趋势二者是一致的，但 LL 库的表现稍弱于 HAL 库。

2. 开发方式选择

各种开发方式的性能表现决定了其应用场合，嵌入式开发者应根据应用场合和自身技术背景选择适合自己和项目的嵌入式开发方式。选择没有固定规则，且带有一定的主观性，此处只给出一些选择建议，

仅供参考。

(1) 如果开发者使用的是小容量、少引脚的微控制器，并且想利用好存储器中的每一位，追求最高性价比，因为硬件抽象是需要成本的，那么寄存器开发或者 STM32Cube LL 库开发将是最佳选择。

(2) 作为有过 8 位单片机开发经验的开发者，如果习惯直接寄存器操作，那么寄存器开发或 STM32Cube LL 库开发将是一个很好的起点。如果更喜欢 C 语言编程，那么建议使用 STM32Cube HAL 库开发或标准库开发。

(3) 有过标准库开发经验的嵌入式工程师，且将来可能仅使用同一系列的微控制器(如 STM32F1 系列)，可以继续使用标准库进行开发。如果开发者计划在未来使用不同的 STM32 系列，那么作者建议考虑使用 STM32Cube 库开发，因为这更容易在系列之间移植。

(4) 如果设计者希望代码具有好的可移植性同时保持较高的优化性能，则开发者可以使用 STM32Cube HAL 库开发方式，并用特定的优化替换一些冗余度高的调用，从而保持最大的可移植性和隔离不可移植但经过优化的区域。也可以使用 STM32Cube HAL 库和 STM32Cube LL 库混合编程的方法来达到上述相同的效果。但需要注意的是，对于同一外设，不可同时使用 HAL 进程和 LL 进程。

对于一个控制系统来说，稳定可靠是最重要的，标准库在同一系列芯片之间代码可直接复用，所以其依然是很多 STM32 开发者难以割舍的情怀。近年来，随着微控制器存储容量成倍增长，主频持续提高，淡化了人们对性能的考量，开发人员更关心软件开发效率和产品迁移，基于 STM32CubeMX 的 HAL 库开发方式逐渐流行起来。本书也是以此开发方式讲解嵌入式系统开发的，有时为了快捷访问和优化性能，也会在部分模块内直接操作寄存器。


微课视频

3.2 软件资源安装与配置

基于 STM32CubeMX 的 HAL 库开发主要需要图形化配置软件 STM32CubeMX、Java 开发环境、MDK-ARM 集成开发环境(Integrated Development Environment，IDE)、芯片器件包、HAL 固件包。上述 5 个软件资源的安装又可划分为两个主要部分。第一部分的重点是安装 STM32CubeMX 开发工具，安装之前需要先安装 Java 运行环境(Java Runtime Environment，JRE)，安装之后还需要在 STM32CubeMX 中添加芯片的 HAL 固件包。第二部分重点是 MDK-ARM 集成开发环境的安装，同样安装完成之后需要添加芯片的器件包。

3.2.1 JRE 安装

1. 下载 JRE 文件

在浏览器地址栏输入网址：https://www.java.com/en/download/manual.jsp，打开如图 3-4 所示的 JRE 下载界面，单击“Windows Offline(64-bit)”链接开始下载 64 位 Windows 操作系统离线 JRE 安装文件。

下载完成的 JRE 安装文件，名称为 jre-8u341-windows-x64.exe，大小为 83.4 MB，版本号为 8.0.3410.10。

2. 安装 JRE 程序

双击 JRE 安装文件，开始安装 Java 运行环境，JRE 安装界面如图 3-5 所示。一般无须更改任何设置，直接单击“安装”按钮开始安装，待出现“您已成功安装 Java”对话框，单击“关闭”按钮完成安装。

3.2.2 STM32CubeMX 安装

STM32CubeMX 软件是意法半导体公司推出的一款具有划时代意义的软件开发工具，它是 ST 公司

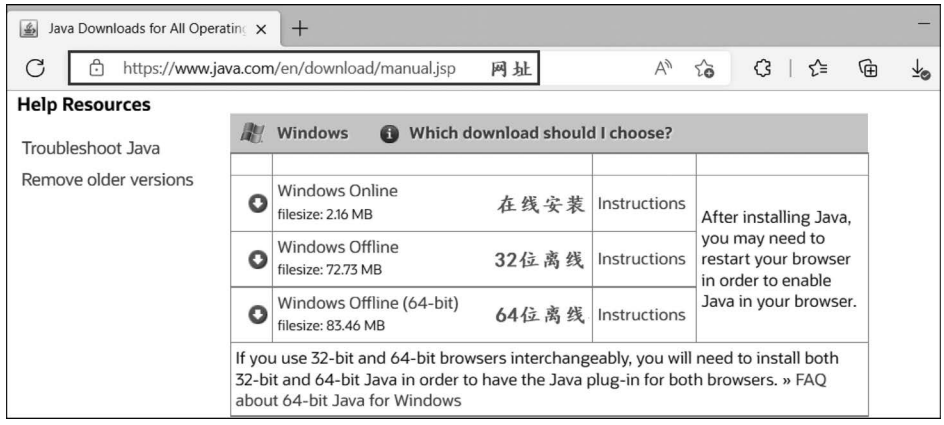

图 3-4　JRE 下载界面

图 3-5　JRE 安装界面

STM32Cube 计划中的一部分。该软件是一个图形化开发工具，用于配置和初始化其旗下全系列基于 ARM Cortex 内核的 32 位微控制器，并可以根据不同的集成开发环境，如 IAR、KEIL 和 GCC 等，生成相应的软件开发项目和 C 代码。简单地说，STM32CubeMX 软件是一款图形化的初始化 C 代码生成器，在本书的后续表述中也会将其简称为 CubeMX 或 Cube。

1. STM32CubeMX 下载

在 ST 官网(www.st.com)首页搜索栏输入 STM32CubeMX，在搜索结果页面单击 STM32CubeMX 链接，进入产品介绍页面，继续单击 Get Software 进入图 3-6 所示 STM32CubeMX 下载页面。上述操作也可以通过在地址栏输入如下网址完成：

https://www.st.com/en/development-tools/stm32cubemx.html#get-software

在图 3-6 中，需要根据具体操作系统选择相应的软件进行下载，大多数读者使用的是 Windows 操作系统，需要下载 STM32CubeMX-Win 安装包，同时软件要不断升级，也可以通过 Select version 下载列表框选择不同的软件版本，如果是下载最新的版本则可以单击 Get latest 进入下载链接。

选择好类别和版本之后，还需要接受许可协议，注册和登录账号才能完成下载。作者下载的安装包文件名为 en.stm32cubemx-win_v6-6-1.zip，版本号为 V6.6.1，大小为 454MB。

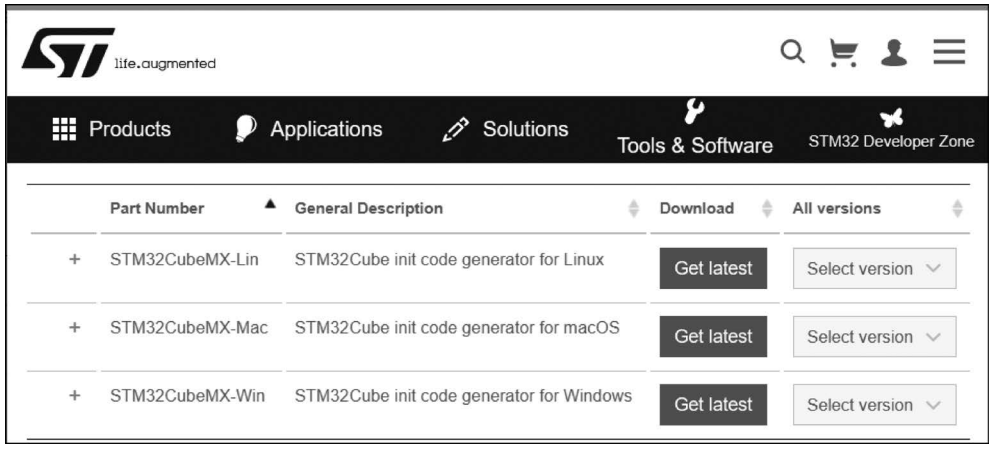

图 3-6 STM32CubeMX 下载页面

2. STM32CubeMX 软件安装

双击安装包中的可执行文件 SetupSTM32CubeMX-6.6.1-Win.exe，启动安装程序，安装界面如图 3-7 所示。

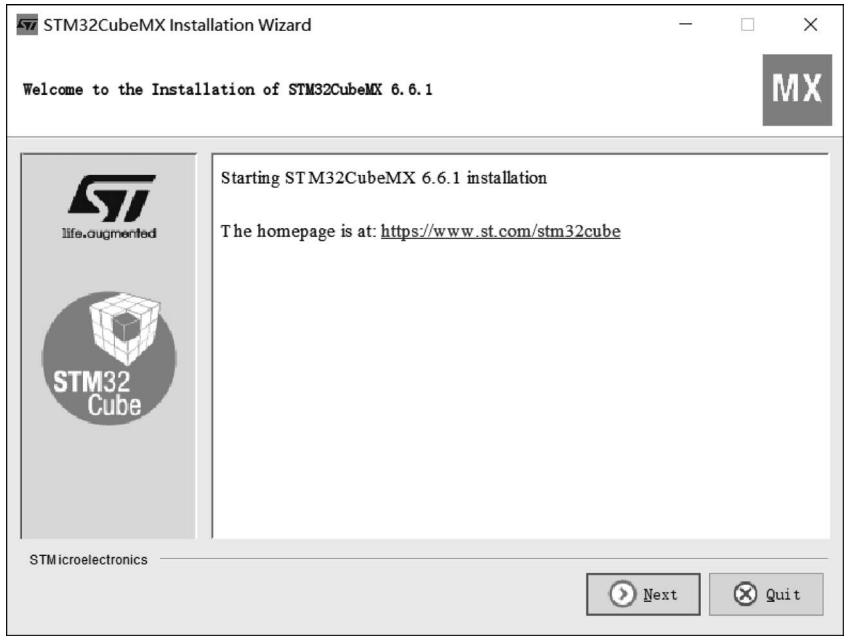

图 3-7 安装界面

依次单击 Next 按钮，接受许可协议，同意隐私条款，自定义安装路径，然后开始程序安装，经过短暂的等待，STM32CubeMX 程序就已经成功安装。

3.2.3 HAL 固件包安装

STM32CubeMX 支持全系列 STM32 芯片的开发，而芯片初始化代码和资源管理是基于 HAL/LL 固件包的，但 CubeMX 没有必要也不可能将所有芯片的固件包都集成到开发环境当中，因此还需要添加可能用到芯片系列的 HAL/LL 的固件包。因为本书是基于 HAL 库的开发，所以此处仅添加 HAL 固件包。

固件包有两种安装方式，一种是在线安装，另一种是将固件包下载后本地安装。本节采用在线安装

方式安装 STM32F4 固件包，采用本地安装方式安装 STM32F1 固件包，但在实际安装时均推荐在线安装。

1. 固件包文件夹设置

STM32CubeMX 软件安装完之后会在桌面和开始菜单生成一个快捷方式，还可以将其添加到开始屏幕或任务栏。双击其快捷方式即可运行 STM32CubeMX 软件，启动界面如图 3-8 所示。

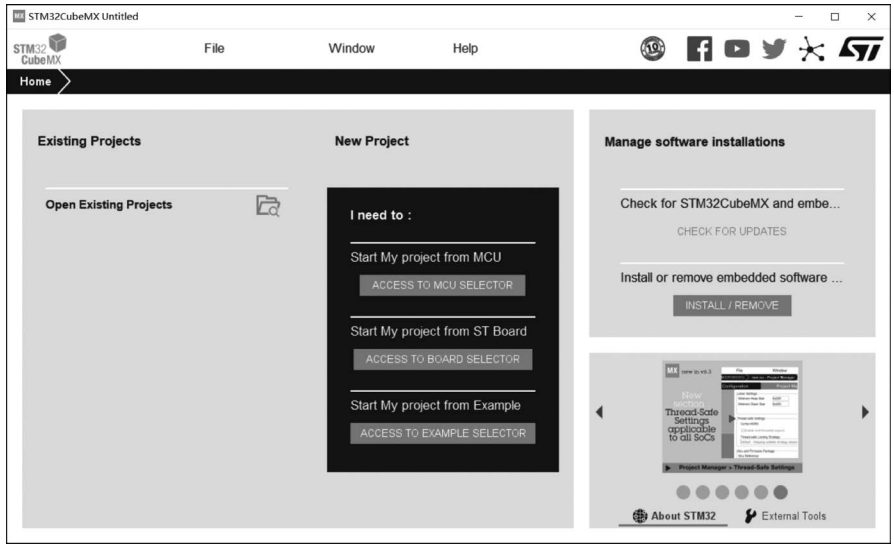

图 3-8　STM32CubeMX 启动界面

在安装固件包之前需要对软件环境和库文件夹进行设置，以便后续使用过程中更加得心应手。由于 C 盘容量经常告急，作者一般将嵌入式学习的软件安装在计算机最后一个分区 G 盘，所以还需要将软件库文件夹也设置在 G 盘。

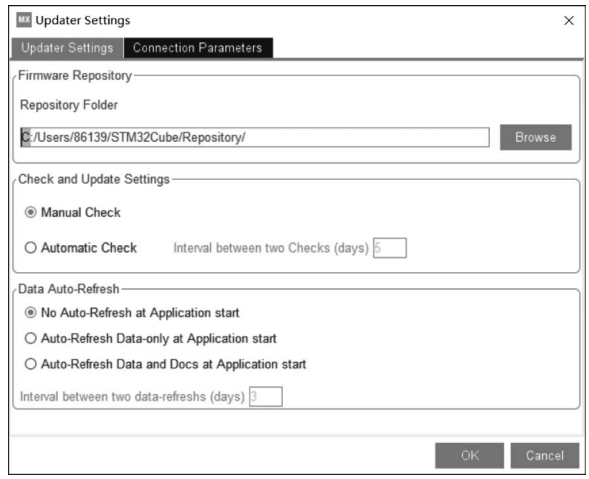

图 3-9　软件库文件夹设置

启动界面最上方共有三个菜单项即 File，Window、Help，依次选择 Help→Updater Settings 菜单项，打开图 3-9 所示软件库文件夹设置对话框。其中 Repository Folder 选项就是需要设置的软件库文件夹，所有的 MCU 固件包和扩展包均安装到此目录下，这个文件夹一经设置并且安装一个固件包之后就不能再更改，此处只需将盘符由 C 修改为 G 即可。如果使用默认路径或是浏览选择其他路径也是可以的，但需要注意路径名中尽量不要带有中文或空格。在图 3-9 中还可以对软件更新和数据更新选项进行设置。

2. 固件包在线安装

如果个人计算机已成功连接至网络，固件包安装一般采用在线安装方式，该方式方便快捷，还可以在线更新。在图 3-8 启动界面依次单击 Help→Manage embedded software packages 菜单项，打开如图 3-10 所示的嵌入式软件包管理对话框。这里将 STM32Cube MCU 固件包和 STM32Cube 扩展包统称为嵌入式软件包。

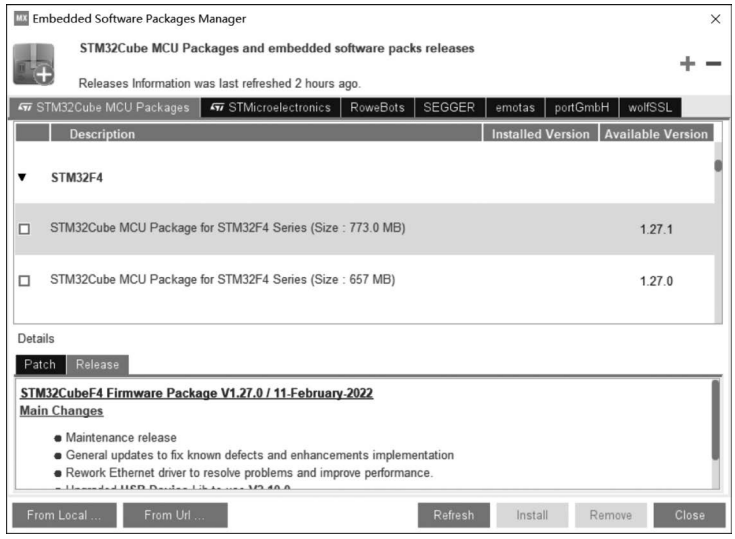

图 3-10　嵌入式软件包管理对话框

在图 3-10 对话框中找到所用芯片系列，如 STM32F4，单击左侧的下三角按钮后会展开不同版本，最前面的一般是最新版本，选择一个版本，将复选框选中，单击对话框下面的 Install 按钮开始安装，等待一段时间，当复选框变成绿色填充时表示固件包已成功安装。

3. 固件包本地安装

如果个人计算机不具备联网条件，且已获取固件包安装文件，可以采用本地安装方式。下面以安装 STM32F1 系列固件包为例，讲解本地安装方式。

在 ST 公司官网(www. st. com)首页搜索栏输入关键字 STM32CubeF1，进入 STM32CubeF1 Active 页面，单击 Get Software 按钮进入 HAL 库下载页面，如图 3-11 所示。在下载资源列表中有两个软件包可以下载，其中 STM32CubeF1 为基础包，Patch_CubeF1 为补丁包，所以这两个文件都需要下载，且均选择最新版本。下载完成的基础包文件为 en. stm32cubef1. zip，版本为 v1. 8. 0，大小为 109MB，补丁包文件为 en. patch_cubef1_v1-8-4. zip，版本为 v1. 8. 4，大小为 51. 4MB。

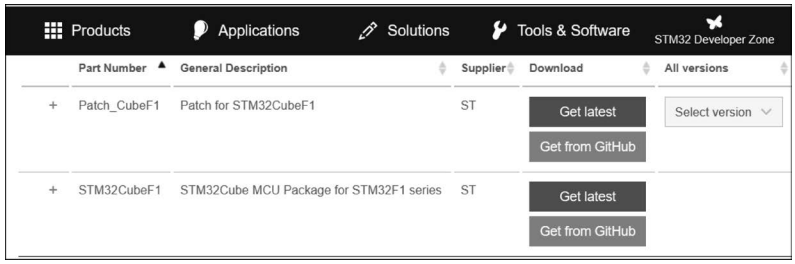

图 3-11　HAL 库下载页面

在图 3-8 启动界面依次单击 Help→Manage embedded software packages 菜单项，打开嵌入式软件包管理对话框，单击左下角的 From Local 按钮，浏览选择 STM32CubeF1 基础包，如图 3-12 所示，单击“打开”按钮开始安装，安装完成之后，软件包管理对话框固件包列表中 STM32Cube MCU Package for STM32F1 Series 列表项前的复选框绿色填充，并出现了版本号 1. 8. 0，表示基础包安装完成。

如果采用相同方法安装补丁包，则会出现依赖错误(Missing dependency for this package)，表示不可以在软件包管理对话框中同时安装基础包和补丁包。

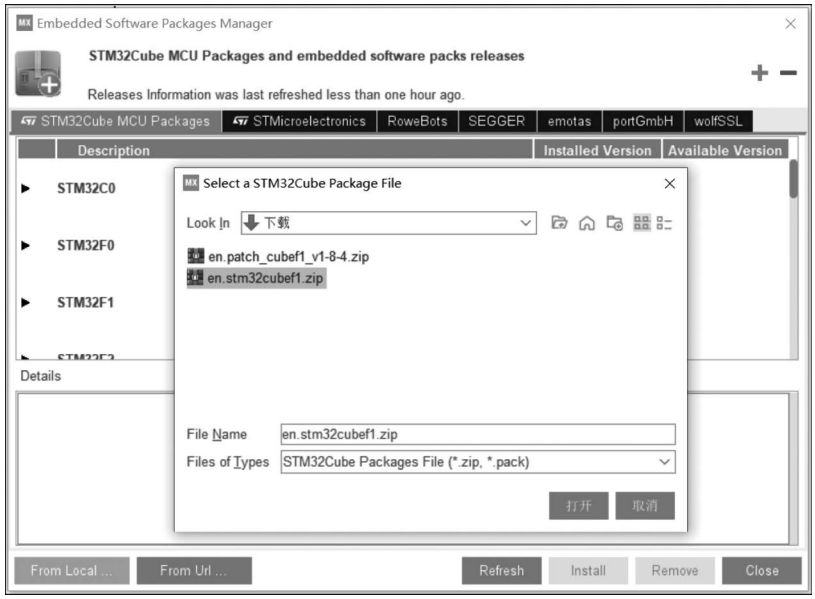

图 3-12　基础包离线安装

解决上述问题的方法是将下载好的基础包和补丁包复制到库文件夹，默认路径为 C：/Users/86139/STM32Cube/Repository/，若已修改请将其替换为新的目标路径，并对这两个文件进行重新命名。如果在重新命名时记不清命名规则，可以从已安装的固件包名称得到相应启示。

基础包：en. stm32cubef1. zip→STM32Cube_FW_F1_V180. zip。

补丁包：en. patch_cubef1_v1-8-4. zip→STM32Cube_FW_F1_V184. zip。

再次打开如图 3-13 所示的补丁包离线安装对话框，选中固件包 1. 8. 4 版本前面的复选框，单击 Install 按钮开始安装（操作方法同在线安装），此时 STM32CubeMX 会检测到补丁包已存在，跳过软件下载程序，直接进行解压步骤，安装完成之后，1. 8. 4 版本的固件包前面复选框同样进行了绿色填充。至此 STM32CubeF1 固件包基础包和补丁包全部安装完成。

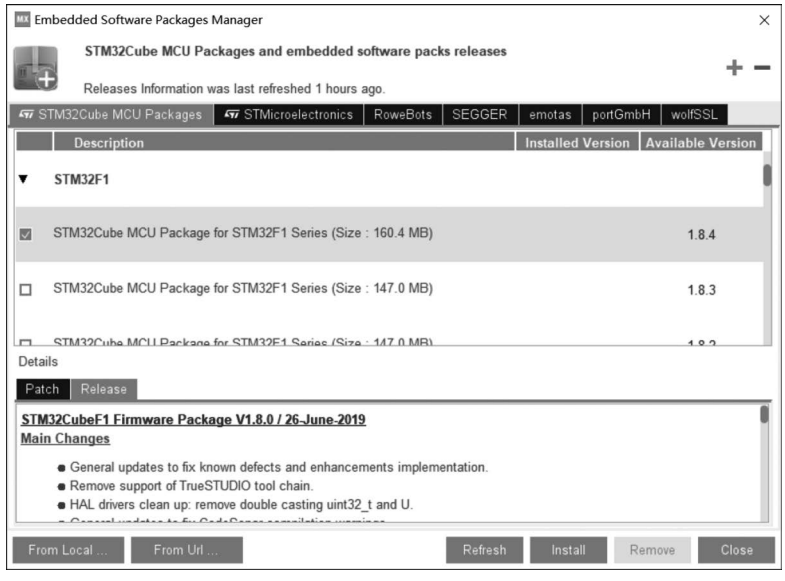

图 3-13　补丁包离线安装

由上述操作可知，STM32Cube 固件包离线安装是对基础包和补丁包分别进行下载和安装的，过程较为复杂，所以除非存在无法联网等特殊情况，推荐使用在线安装。

3.2.4 MDK-ARM 安装

1. MDK-ARM 简介

MDK-ARM 源自德国的 KEIL 公司，也称 KEIL MDK-ARM、KEIL ARM、KEIL MDK、Realview MDK、I-MDK、μVision5(μVision4)等，全球超过 10 万的嵌入式开发工程师使用 MDK-ARM。目前最新版本为 MDK 5.37，该版本使用 μVision5 集成开发环境，是目前针对 ARM 处理器，尤其是 ARM Cortex-M 内核处理器的最佳开发工具。

MDK5 向后兼容 MDK4 和 MDK3 等，以前的项目同样可以在 MDK5 上进行开发(需安装兼容包)，MDK5 同时加强了针对 ARM 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升级，如图 3-14 所示，MDK5 由两个部分组成：MDK Tools(MDK 工具)和 Software Packs(软件包)。其中，MDK Tools 包含 MDK-Core 和 Arm C/C++Compiler 两部分；Software Packs 可以独立于工具链进行新芯片支持和中间库的升级。

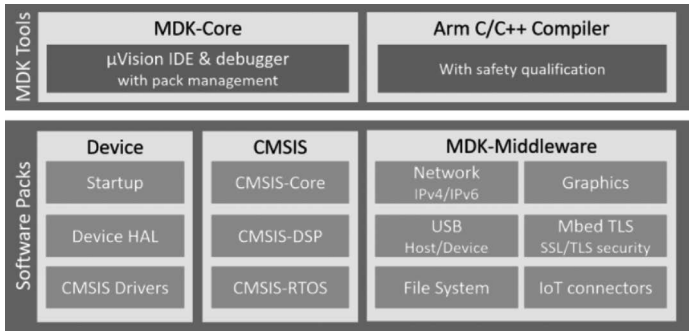

图 3-14 MDK5 组成

2. MDK-ARM 下载

在 KEIL 官网(www.keil.com)首页，单击顶端 Download 链接，首先出现一个 Overview 页面，继续单击 Product Downloads 选项，进入如图 3-15 所示的产品下载列表页面。

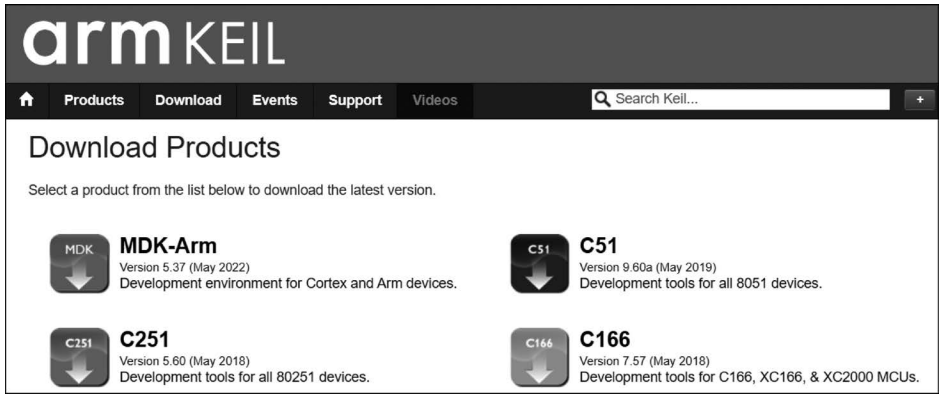

图 3-15 产品下载列表

在图中有 4 个下载选项，第 1 个选项为 MDK-ARM，适用于 ARM 内核开发工具，最新版本为 5.37，其他选项用于相应型号单片机的开发。单击 MDK-ARM 软件图标进入下载页面，填写并提交用户信息

之后，软件便开始下载。

在最新的 MDK 5.37 版本中仅内置 Compiler 6.18，没有预装 Compiler 5，这将会导致在早期的 MDK5 或 MDK4 创建的文件无法编译。虽然用户也可以手动添加 Compiler 5，但这个操作比较麻烦，也会进一步增加安装文件占用空间，因此，如果读者不需要打开以前创建的工程，可以直接安装最新版本，否则可以和作者一样选用上一个版本的安装文件 MDK535.EXE，文件大小为 890MB。

3. MDK 软件安装

双击下载完成的 MDK535.EXE 可执行文件启动安装程序，单击 Next 按钮，同意许可协议，进入安装路径设置，默认安装于 C 盘，此处将其修改为 G 盘，MDK 安装界面如图 3-16 所示，继续单击 Next 按钮，填写用户信息开始软件安装，当安装界面出现 Finish 按钮，表示软件已安装完成，单击此按钮退出安装程序。

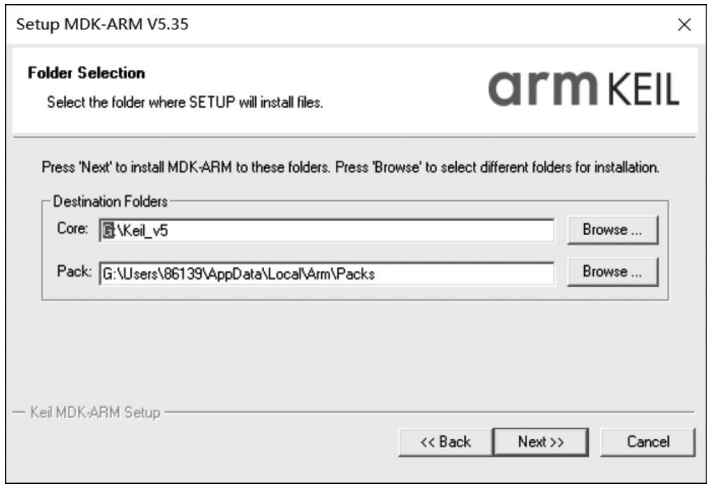

图 3-16　MDK 安装界面

3.2.5　器件包安装

随着芯片系列、种类越来越多，MDK-ARM 软件越来越难以将所有组件都集成到一个安装包中，所以和 MDK4 版本不同，从 MDK5 开始，MDK-Core 是一个独立的安装包，基于 μVision，对 ARM Cortex-M 设备提供支持，提供安装程序用于下载、安装和管理软件包，可随时将软件包添加到 MDK-Core，使新的设备支持和中间件更新独立于工具链。

在 MDK 5.35 安装完成后，要让 MDK5 支持 STM32F4 和 STM32F1 系列芯片开发，还要安装 STM32F4 的器件包(Keil.STM32F4xx_DFP.2.16.0.pack)和 STM32F1 的器件包(Keil.STM32F1xx_DFP.2.4.0.pack)，安装方式依然分为在线安装和离线安装，实际安装时推荐使用离线安装。

1. 器件包在线安装

MDK-ARM 软件安装成功之后，会在桌面和开始菜单栏创建快捷方式，双击或单击快捷方式启动 Keil μVision5，单击调试工具栏最右边 Pack Installer 图标，打开如图 3-17 所示器件包在线安装对话框。

在图 3-17 器件包安装管理器中，在 Device 栏，先选择芯片厂家 STMicroelectronics，再选择 STM32F4 系列，Summary 栏显示该系列器件的数量，在 Pack 栏选择安装文件 STM32F4xx_DFP，单击 Action 栏的 Install 按钮开始安装，此时按钮转变为灰色不可用状态，对话框的最下面显示器件包安装进度和安装方式。当器件包安装完成，Install 按钮前面绿色图标会被点亮。

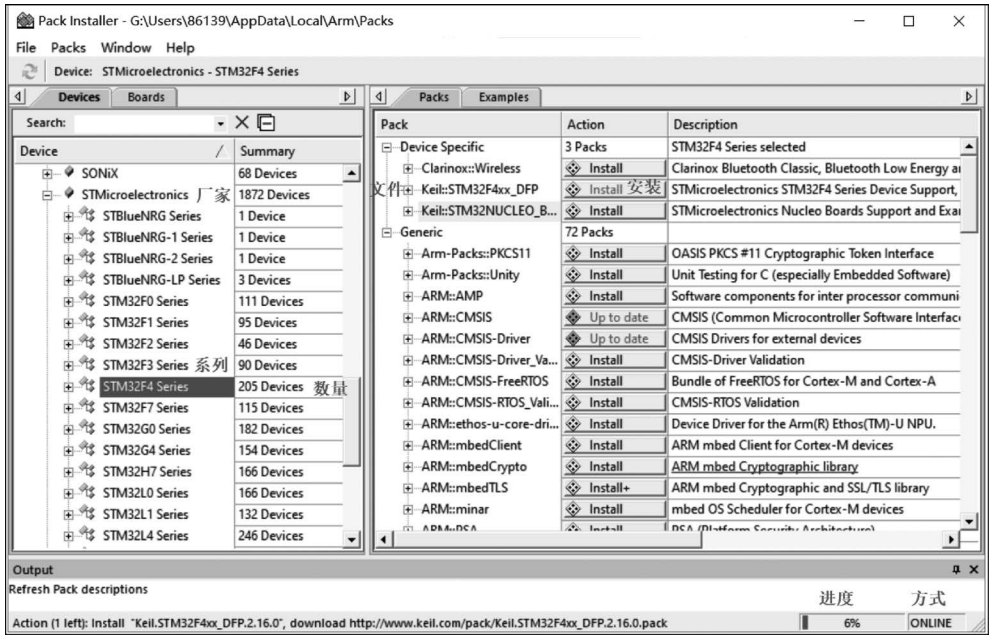

图 3-17 器件包在线安装

2. 器件包离线安装

如果已经获取器件包文件，则也可以采用离线安装的方式进行。

首先需要前往官方下载地址(https://www.keil.com/dd2/pack/)下载最新的器件包，在资源浏览页面中，首先找到芯片厂家 STMicroelectronics，然后找到要下载的产品系列名称 STM32F1 Series Device Support，Drivers and Examples，单击后面的下载按钮开始资源下载。以 STM32F1 为例，下载完成后文件名称为 Keil.STM32F1xx_DFP.2.4.0.pack，版本号为 2.4.0，大小为 47.9MB。

双击下载文件开始器件包安装，器件包离线安装页面如图 3-18 所示，安装文件会自动识别 MDK-ARM 的安装路径，无须任何更改，单击 Next 按钮开始安装，出现 Finish 按钮提示安装完成。当器件包安装完成，打开图 3-17 所示器件包安装管理器，和在线安装方式一样，Install 按钮前面绿色图标也将被点亮。

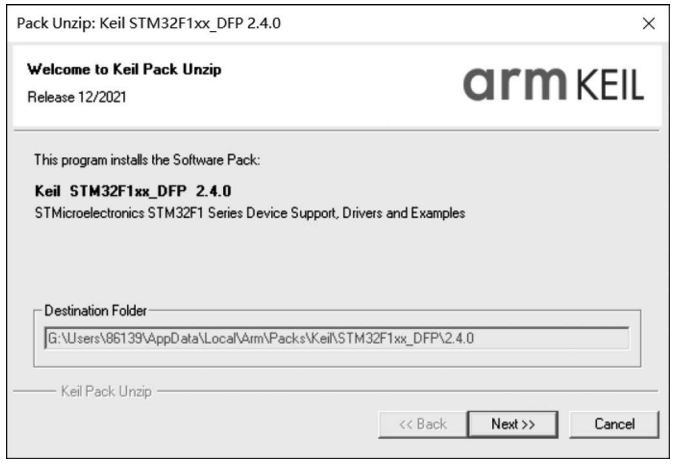

图 3-18 器件包离线安装

作者在安装的过程中发现，无论是在线安装还是离线安装方式，器件包的下载速度均十分缓慢，如果可以通过共享获得器件包，采用**离线**安装方式更简单、快捷，是推荐安装方式。

3.2.6　MDK-ARM 注册

MDK-ARM 作为 ARM Cortex-M 内核微控制器最全面的解决方案，提供了丰富的产品线，其支持能力和产品特性如图 3-19 所示。MDK-Lite 是免费评估版，默认即是安装此版本，要想获得开发环境全面支持和更好使用性能，还需要将其注册为 MDK-Professional 版本。

	Microcontroller Development Kit Editions			
Components	MDK-Professional	MDK-Plus	MDK-Essential	MDK-Lite
µVision IDE with Editor and Pack Installer	✔	✔	✔	✔
Arm C/C++ Compiler (armcc)	✔	✔	✔	32KB
Arm Macro Assembler (armasm)	✔	✔	✔	✔
Arm Linker (armlink)	✔	✔	✔	32 KB
Arm Utilities (fromelf)	✔	✔	✔	✔
Arm C and C ++ Libraries	✔	✔	✔	✔
Arm C Micro-Library (microlib)	✔	✔	✔	✔
µVision Debugger	✔	✔	✔	32 KB
CMSIS and Middleware Libraries				
CMSIS-CORE, CMSIS-DSP, CMSIS-RTOS RTX	✔	✔	✔	✔
File System, Graphic, Network IPv4, USB Device	✔	✔		
File System, Graphic, Network IPv4/IPv6, USB Host/Device	✔			
Arm Processor Support				
Arm Cortex-M0, M0+, M1, M3, M4, M7	✔	✔	✔	✔
Arm Cortex-M23, M33 (non-secure)	✔	✔	✔	
Arm Cortex-M23, M33 (secure)	✔	✔		
Arm7, Arm9, Cortex-R4	✔	✔		
SecurCore	✔	✔		

图 3-19　MDK-ARM 支持能力和产品特性

在桌面或开始菜单找到 Keil µVision5 快捷方式，右击选择以管理员身份运行。单击 File→License Management 菜单项，打开授权管理对话框，如图 3-20 所示，将图中 Computer ID 填写到注册软件 CID 文本框，注册目标选择 ARM，版本选择 Professional，复制软件生成的 License ID，并填写到图中 License ID Code(LIC)编辑框，单击 Add LIC 按钮完成注册。注册成功会在授权管理对话框显示使用期限和“ *** LIC Added Successfully *** ”提示信息。

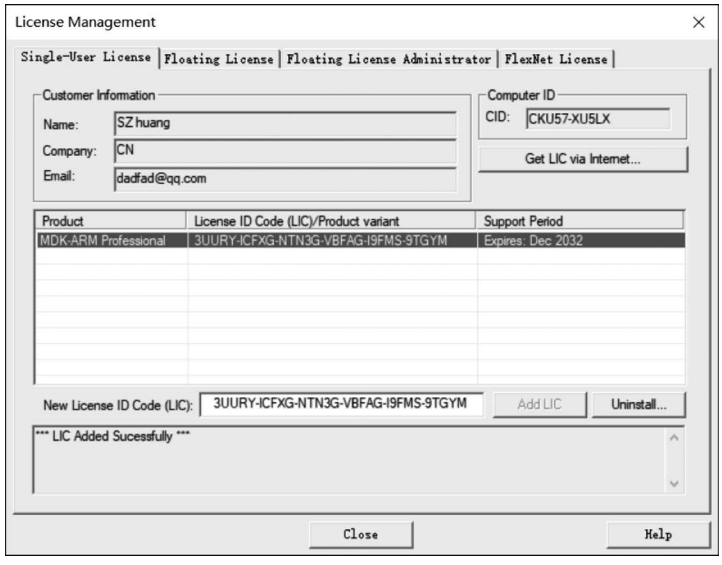

图 3-20　MDK 授权管理对话框

3.2.7 软件安装总结

软件平台配置主要包括两方面工作。一方面是软件资源下载，一般是从官方网站直接下载，除因兼容性问题选择了 MDK-ARM 的上一个版本 MDK 5.35，其余软件和资源包均为最新版本。另一方面是软件安装，本书围绕着两个软件展开：第一个软件是图形化配置工具 STM32CubeMX 的安装，安装之前需要先安装 JRE，安装完后还需要安装芯片的固件包(HAL/LL)，在固件包安装过程中，推荐使用在线安装方式；第二个软件是 32 位 MCU 集成开发环境 MDK 5.35，安装完成之后还需要安装芯片器件包，器件包推荐离线安装。

3.3 基于 STM32CubeMX 的 HAL 开发方式

基于 STM32CubeMX 的 HAL 开发涉及软件众多，对于初学者有时可能不知从何开始。为此，作者设计了本书第一个也是最简单的项目实例，即配置开发板 LED 指示灯 L1 引脚为输出模式(默认输出低电平)，编写 LED 周期闪烁应用程序，连接调试器，下载程序并复位运行。

3.3.1 STM32CubeMX 生成初始化代码

使用标准库进行嵌入式开发的第一步就是建立适合自己的工程模板，并编译通过，此外，在使用外设之前需要花较多精力对其初始化，然后才是应用程序的编写。而借助 STM32CubeMX 可以轻松完成前面两步，显著减少了代码量，可靠性也得到进一步的提高。

1. 选择 MCU 芯片


微课视频

运行 STM32CubeMX 软件，其初始界面如图 3-21 所示，各部分功能如图中标注所示，可以在该界面打开或新建工程。其中新建工程又分为 3 种方式，第 1 种是 Start My project from MCU(选择一款 MCU)新建工程，这是最常用的方式，其他 2 种方式分别是 Start My project from ST Board(选择 ST 评估板)和 Start My project from Example(参考例程)新建工程。

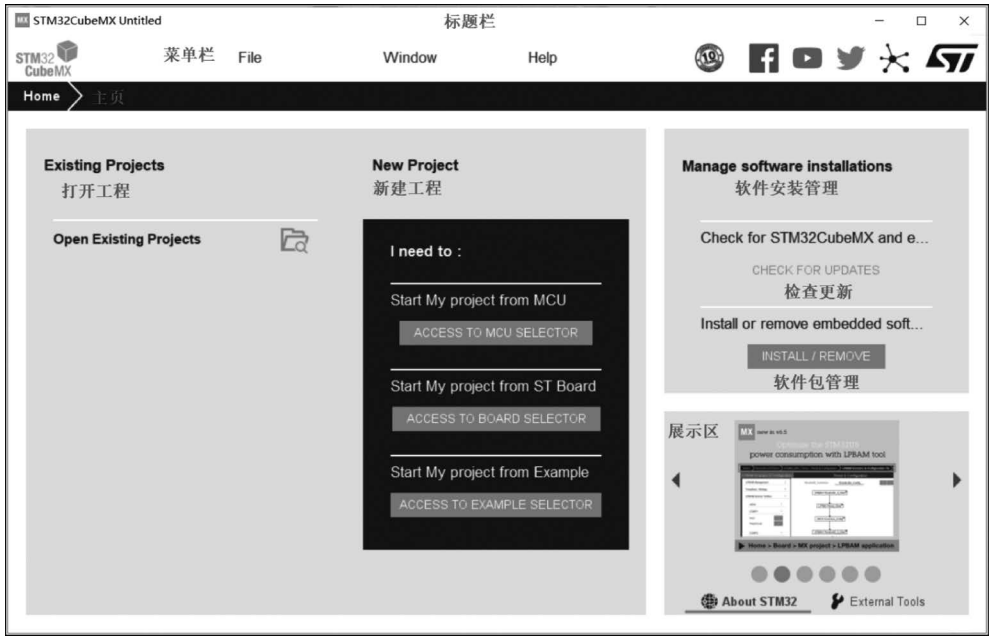

图 3-21 STM32CubeMX 初始界面

选择第 1 种方式新建工程，单击 ACCESS TO MCU SELECTOR 选项打开如图 3-22 所示 MCU/MPU 芯片选择对话框。

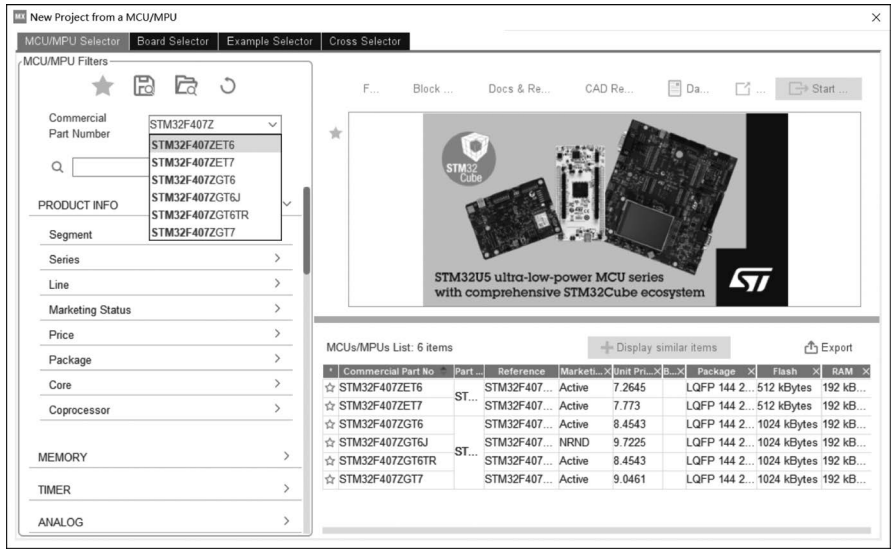

图 3-22　MCU/MPU 芯片选择

为方便查找芯片，该对话框中设置了各种筛选条件，比方按产品信息、存储器、定时器、模数转换器等，每一个筛选类别又细分为多个子项目。最快捷简单的方法是在 Commercial Part Number 列表组合框中输入芯片名称（如 STM32F407ZET6），MCUs/MPUs List 列表将会列出相应的芯片，其上部也会给出芯片主要性能介绍，单击列表前面的星形符号，可以收藏此芯片，双击芯片名称，完成 MCU 选择，跳转至工程创建对话框，如图 3-23 所示。配套步骤是按照工程创建流程组织的，在完成芯片选择之后，还需要经过 Pinout & Configuration（引脚及资源配置）、Clock Configuration（时钟配置）、Project Manager（工程管理）等步骤才能初始化外设和生成项目工程。

图 3-23　工程创建对话框

2. 配置 GPIO 引脚

下面以 PF0 引脚配置为例，讲解 GPIO(通用目的输入输出)引脚的配置。PF0 引脚配置如图 3-24 所示，首先在工程创建对话框中的引脚视图上找到 PF0 引脚，也可以使用右下方的查找工具输入引脚名称进行快速查找。在 PF0 引脚上用鼠标左键选择引脚功能为 GPIO_Output。然后展开左侧最上面的 System Core(系统内核)配置组别，选择 GPIO 子项，此刻在配置类别和引脚视图中间增加了 GPIO Mode and Configuration 配置区域，这一区域划分为 Mode 和 Configuration 两个子区，但是对本项目来说，GPIO 引脚无须配置工作模式，仅需配置 Configuration 选项即可。

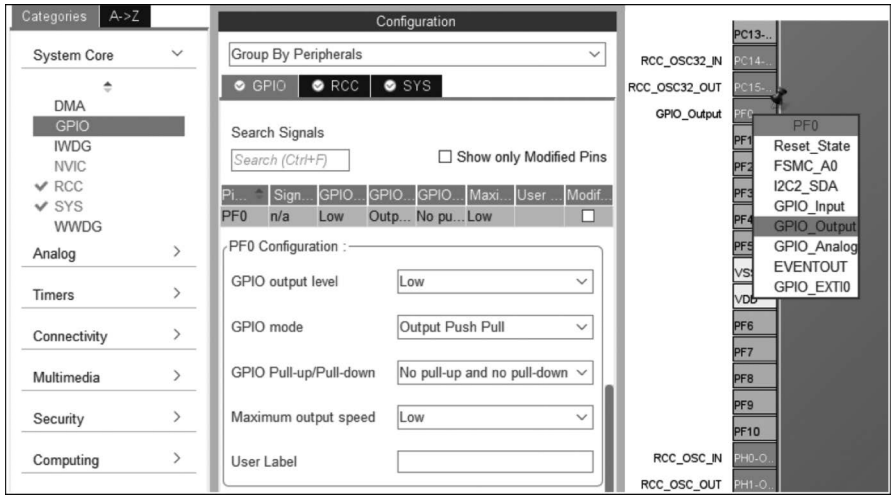

图 3-24 PF0 引脚配置

根据项目设计要求，需要将 PF0 引脚配置为输出模式，默认输出电平为低电平，没有上拉或下拉，最大输出速度为低，其实上述设置均为 GPIO_Output 模式的默认设置，此处无须修改。

3. 配置时钟源(RCC)

完成引脚配置之后，还需要配置 System Core 下面的 RCC 子项，时针源配置过程及结果如图 3-25 所示，此处实际上是配置系统的时钟源。其中 High Speed Clock(HSE)和 Low Speed Clock(LSE)均有 3 个选项：Disable(禁用外部时钟)、BYPASS Clock Source(外部有源时钟)、Crystal/Ceramic Resonator(外部无源陶瓷晶振)三个选项。由第 2 章开发板硬件电路可知，开发板的外部高速时钟(HSE)和外部低速时钟(LSE)引脚均外接石英晶体振荡器，所以 HSE 和 LSE 均应选择 Crystal/Ceramic Resonator。

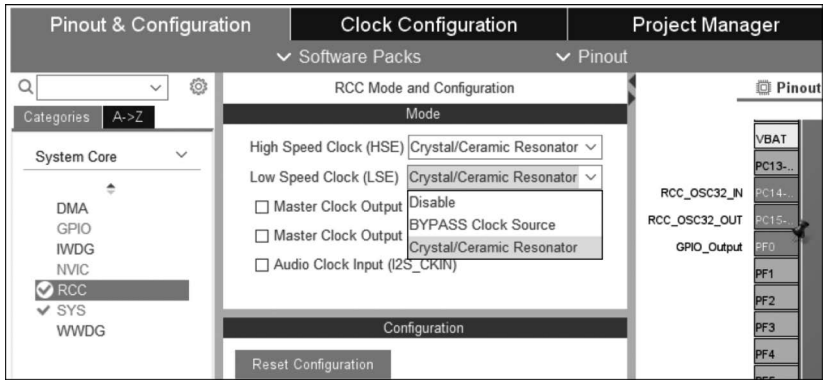

图 3-25 时钟源配置过程及结果

4. 配置调试方式

调试方式配置如图 3-26 所示，选择 System Core 类别下的 SYS 子项对调试方式进行配置，由第 2 章硬件电路可知，开发板设计板载 CMSIS-DAP 调试器采用 Serial Wire Debug(SWD，串行线调试)方式，所以在 SYS 模式配置中 Debug 应当选择 Serial Wire 调试方式。此时引脚视图中 SWD 调试用到的 PA13 和 PA14 变为绿色，并对其调试功能进行了标注。调试方式一定要设置，否则可能导致工程无法调试下载。在图 3-26 中的 Timebase Source(时基时钟)需要保持默认值 SysTick 定时器，不要修改。

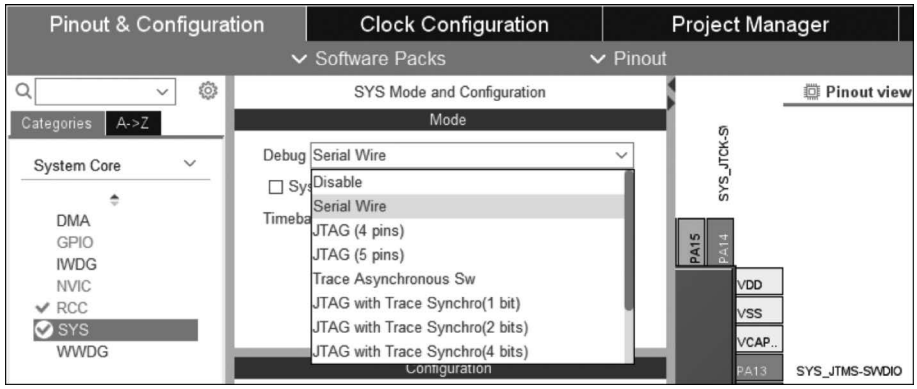

图 3-26　调试方式配置

5. 配置系统时钟

如图 3-27 所示，单击工程创建对话框流程控制按钮 Clock Configuration 进入系统时钟配置界面，此处只需要配置系统时钟，配置步骤根据图 3-27 中序号依次开展。

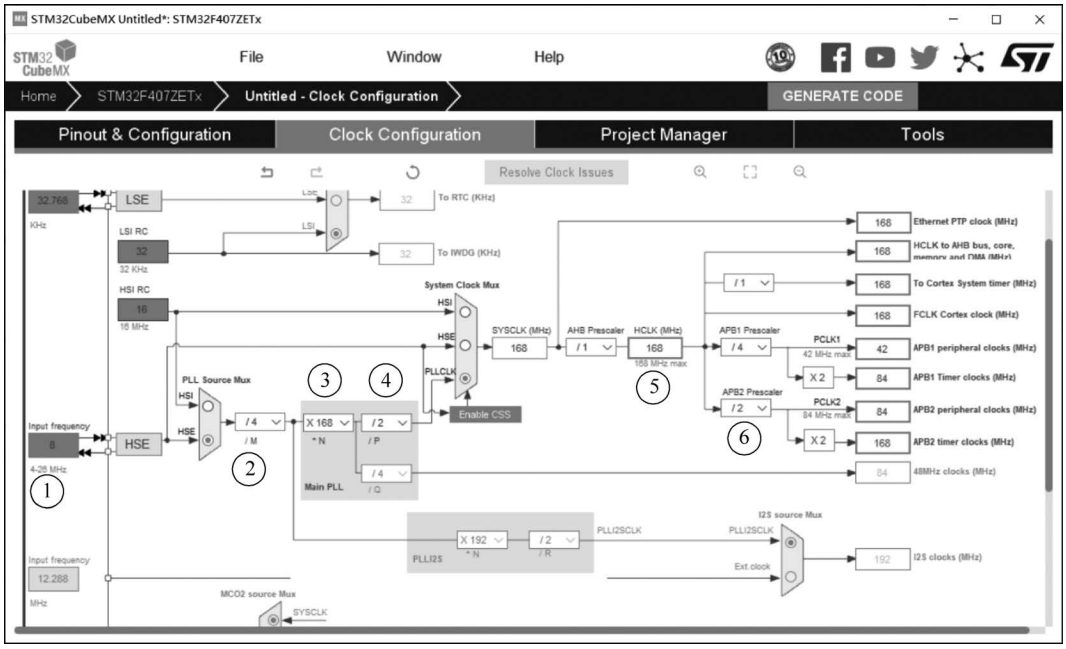

图 3-27　系统时钟配置

时钟配置第①步选择 HSE 作为系统的时钟源，并在 Input frequency 频率输入框中输入数字 8，表示频率为 8MHz。第②步设置分频系数“/M”为“/4”，外部 8MHz 经 4 分频后频率为 2MHz。第③步将“＊N”倍频系数设置为“×168”，2MHz 经 168 倍频后频率为 336MHz。第④步将“/P”分频系数设置为

“/2”，336MHz 再经 2 分频后频率为 168MHz。第⑤步将 System Clock Mux 设置为 PLLCLK，AHB 分频系数保持默认值 1，此时 SYSCLK 和 HCLK 时钟频率均为 STM32F407 最高频率 168MHz。第⑥步将 APB1 分频系数设为 4，PCLK1 工作于最高允许频率 42MHz，APB2 分频系数设为 2，PCLK2 工作于最高允许频率 84MHz。

工程创建时可以将系统时钟配置在一个很广的范围内，但是为了最大限度发挥 CPU 潜能，一般将其配置在最高工作频率 168MHz，这一频率也是标准库例程的默认工作频率。即使将系统时钟配置在 168MHz 主频上，也有很多种组合，上述配置选项只是一个参考实例。

6. 工程选项配置

在完成引脚及资源配置和时钟配置之后，下一步就是工程管理配置，单击主界面的 Project Manager 标签进入工程管理配置，如图 3-28 所示，在左侧配置类别列表中有 3 个子项，分别为 Project（工程）、Code Generator（代码生成）、Advanced Settings（高级选项），一般只需要配置前两项。

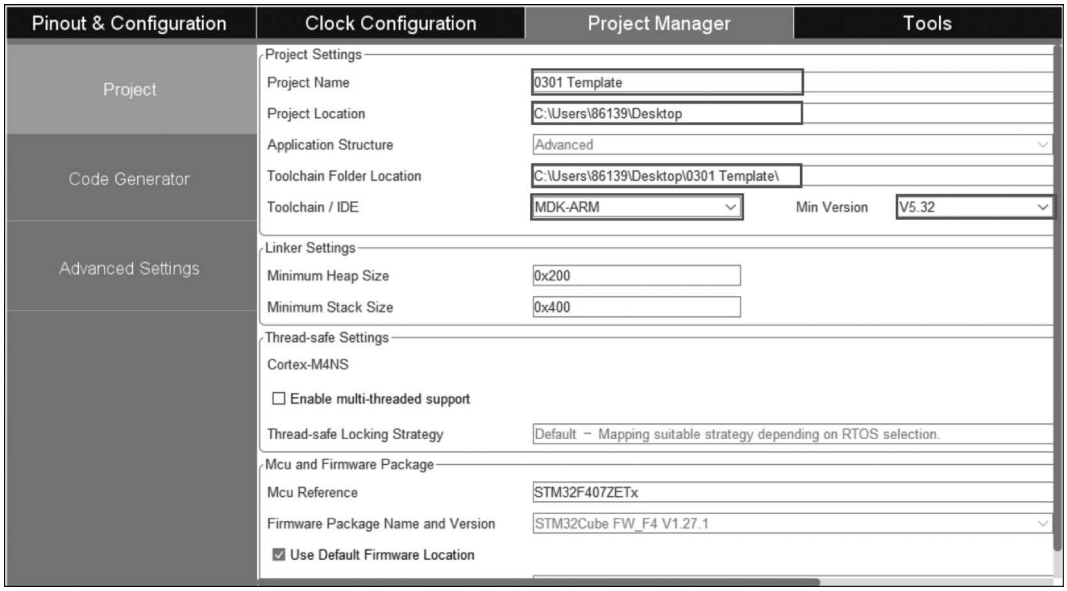

图 3-28　工程管理配置

Project 选项配置界面一般只需设置图中框线标出的地方，即设置工程文件名称、工程路径、工具链文件夹路径。其实只需输入工程名称和工程路径，工具链文件夹路径是二者的合成。STM32CubeMX 在工程文件路径创建一个以工程名称命名的文件夹，工具链文件夹及其他文件均存放在这一文件夹内。

为便于交流和学习，本书工程名称采用统一命名格式，即章节（2 位）＋序号（2 位）＋空格＋项目主题。例如本章的项目名称为 0301 Template，表示第 3 章第 1 个项目，重点讲解工程开发的模板结构，为便于后续章节共用模板，本章创建工程名称为 Template，项目备份时再将名称更改为 0301 Template。

工具链/集成开发环境（Toolchain/IDE）这一选项也十分重要，由组合框下拉列表选项可知，STM32CubeMX 支持的工具链有 EWARM、MDK-ARM、STM32CubeIDE、Makefile 4 种。本书使用的集成开发环境是 MDK 5.35，所以 Toolchain/IDE 选项选择 MDK-ARM，Min Version 选项是用来选择开发工具的版本号的，但是列表中并没有 V5.35 这一选项，只需要选择 STM32CubeMX 所支持的最新版本 V5.32 就可以，或者直接选择 V5。

配置完 Project 选项之后，还需要配置 Code Generator 选项，这部分配置实际取决于开发者的使用习

惯，本书的代码生成选项配置情况如图 3-29 所示，其中重要部分使用框线标出，具体步骤如下。

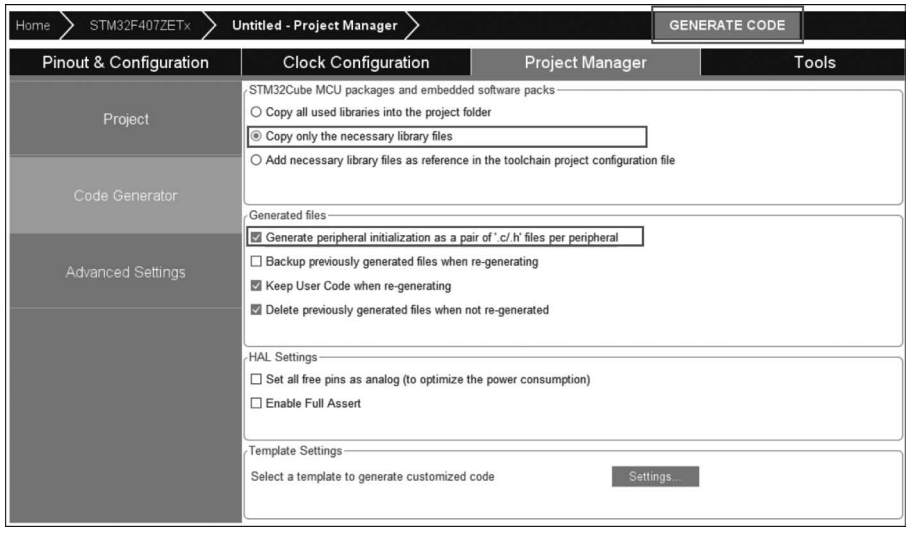

图 3-29　Code Generator 选项配置

第 1 步，STM32Cube MCU packages and embedded software packs（器件包和软件包）复制方式选择，建议选择 Copy only the necessary library files（仅复制必需的文件），否则全盘复制会使文件很大。

第 2 步，Generated files（生成文件）方式选择，该选项组列出了 4 个选项，各选项相互之间并没有联系，可以选中（打钩）或取消，其中第 1 项和第 2 项默认为未选中，第 3 项和第 4 项默认为选中。

其中第 1 项 Generate peripheral initialization as a pair of '. c/. h' files per peripheral 询问是否为每个外设生成一对". c/. h"文件。假设初始化了 GPIO 外设，选中该选项则会生成 gpio. c 和 gpio. h 两个文件，否则将所有外设的初始化函数全部放于 main. c 中。这一选项是否选中不会对代码生成和程序执行产生任何影响，仅会影响文件组织结构。为了程序开发的条理性，建议选中此选项。第 2～4 项采用默认选项，无须更改。


微课视频

至此，基于 STM32CubeMX 的 HAL 库初始化代码生成的全部配置工作已经完成，单击配置界面右上方的 GENERATE CODE 按钮，即可生成包含外设初始化代码的 MDK-ARM 工程文件。

3.3.2　MDK-ARM 集成开发

使用 STM32CubeMX 初始化外设时，还生成一个采用该芯片开发的工程模板，用户可以直接在此模板上进行应用程序开发，减少了工作量，提升了效率，而代码编辑、编译、下载、调试是在 MDK-ARM 集成开发环境中完成的。

1. 工程模板结构

STM32CubeMX 软件在生成代码时会在指定路径创建以工程名命名的工程文件夹，其目录结构如图 3-30 所示，工程模板文件夹根目录下有 3 个文件夹和 2 个文件。

1）Core 文件夹

Core 文件夹存放的是用户文件，包含两个子文件夹：一个是 Inc 文件夹，用于存放头文件；另一个是 Src 文件夹，用于存放源文件。

2）Drivers 文件夹

Drivers 文件夹是固件库驱动程序，也包含两个子文件夹，其中 CMSIS 存放内核驱动程序，STM32F4xx_HAL_Driver 存放 STM32F4 的 HAL 库的驱动程序。STM32F4 的 HAL 库驱动程序驱动

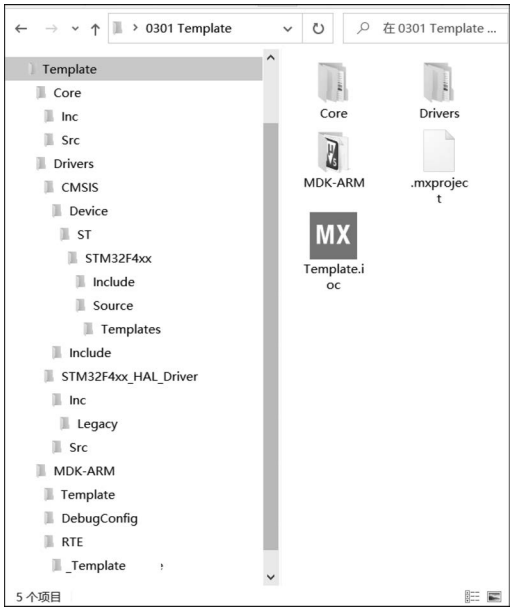

图 3-30　工程模板目录结构

的每一个外设都有一个源文件和一个头文件，分别存放于 Src 子文件夹和 Inc 子文件夹。

3）MDK-ARM 文件夹

MDK-ARM 文件夹存放 MDK-ARM 工程相关文件，Template. uvprojx 是 MDK5 工程文件。Template 子文件夹用于存放编译输出文件，数量较多，占用空间较大，备份时可以仅保留. axf 和. hex 两个文件。

4）. mxproject 文件

. mxproject 是 STM32CubeMX 的配置文件。

5）. ioc 文件

Template. ioc 文件是 STM32CubeMX 的项目文件，如果需要更改外设配置信息，可双击打开该项目文件，更改相关配置重新生成工程文件即可。

2. MDK-ARM 软件使用

双击打开 MDK-ARM 文件夹中的工程文件 Template. uvprojx，软件主界面如图 3-31 所示。

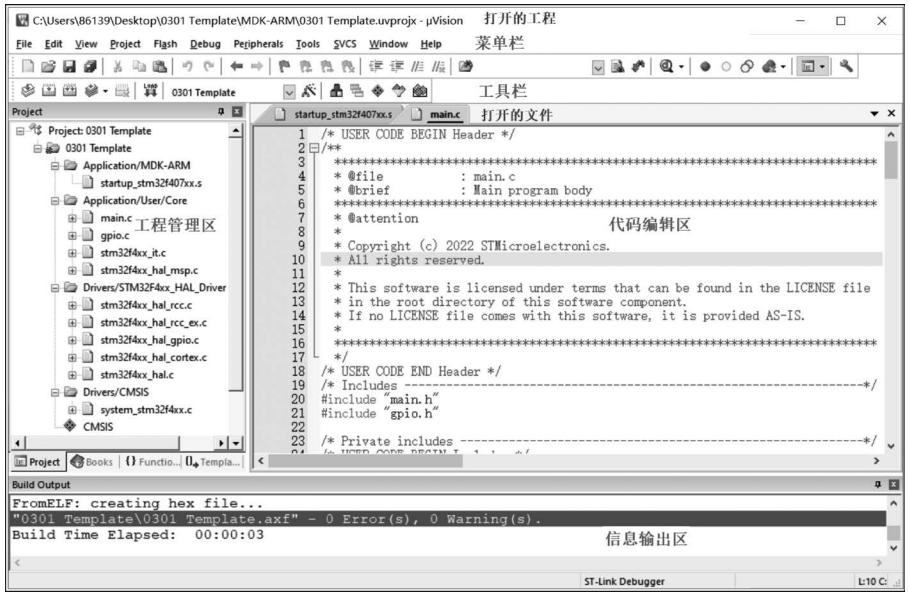

图 3-31　MDK-ARM 软件主界面

1）标题栏

标题栏位于软件界面最上方，左边显示打开工程的路径，右边是最小化、还原、最大化三个按钮。

2）菜单栏

菜单栏位于标题栏下方，包含软件的全部操作，有 File、Edit、View、Project、Flash、Debug、Peripherals、Tools、SVCS、Window、Help 共 11 个菜单命令。

3）工具栏

工具栏位于菜单栏下方，包含软件常见操作命令。在软件使用过程，虽然所有的命令都可以通过菜单栏查找到，但使用工具栏更便捷一些。

4）工程管理区

工程管理区位于界面中部左侧，和 8 位单片机简单的文件结构不一样，STM32 项目开发文件必须以工程方式进行组织，且在一个工程中需要对文件按类别进行分组。单击工程管理区分组名称前面的“+/−”号可以展开或收起分组的文件目录。

选择 Project→Manage→Project items 菜单，或工具栏中的 图标即可以打开 Manage Project Items 对话框，如图 3-32 所示，在此对话框中，可对工程文件、分组名称、包含文件进行更改和配置，修改结果会同步更新到工程管理区。

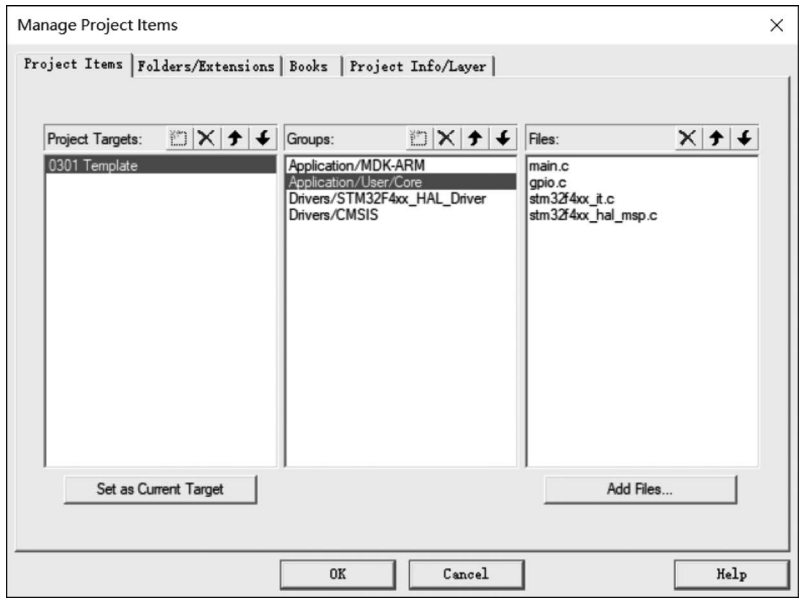

图 3-32　项目分组管理对话框

在后续嵌入式开发中，涉及新建文件或项目移植时需要更改项目分组结构也可以在工程管理区分组文件夹上双击添加文件，或者选中分组文件单击鼠标右键，选择“删除”命令进行删除。

5）代码编辑区

代码编辑区位于界面中部右侧，双击工程管理区项目分组下的任一文件，即可将此文件在代码编辑区打开，此文件处于编辑状态，编辑器支持同时打开多个文件。

编辑器的字体、颜色、缩进等个性化选项都是可以设置的。单击 Edit→Configuration 菜单打开编辑器设置对话框，其中有很多选项卡，第一个选项卡是 Editor，为了更好地支持中文注释，可以将 Encoding 选项设为 Chinese GB2312(Simplified)，也可以更改编辑制表位缩进字符 Tab size 等内容。第二个选项卡是 Colors and Fonts，用于设置编辑器的字体和颜色，一般只修改编辑器的字体，操作方式如图 3-33 所示。在 Windows 列表框选择一类文件，如 C/C++Editor files，保持 Element 列表框中的默认选项 Text，在 Font 面板中选择相应的字体、字号和颜色完成设置即可。

6）信息输出区

信息输出区位于主界面下端，多数情况输出的是编译信息，下面就介绍如何设置编译选项，并对工程进行编译。

单击 Project→Options for Target 菜单项，或单击工具栏中的 图标，或按下 Alt+F7 快捷键均可以打开工程选项对话框，如图 3-34 所示。工程文件有很多重要设置均在这一对话框进行设置，此处仅讲解编译相关设置。

图 3-33　编辑器字体设置

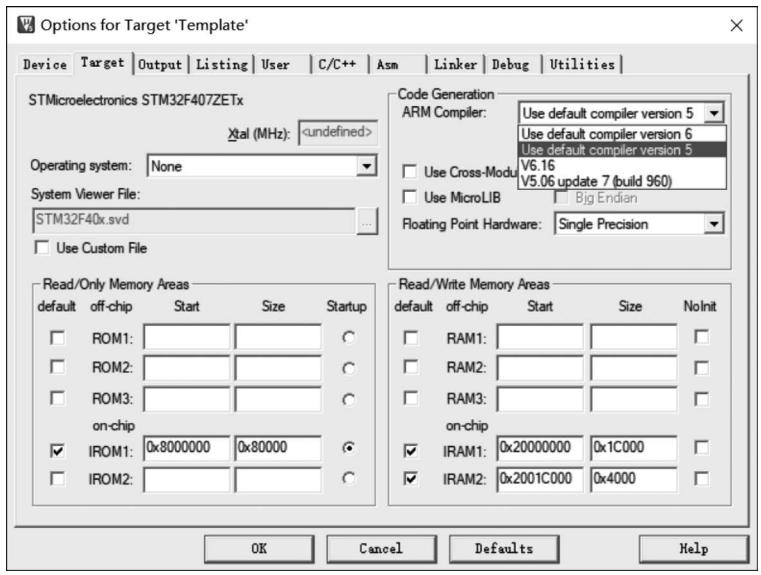

图 3-34　“工程选项”对话框——编译选项设置

该对话框中 Target 选项卡的 Code Generation 区域的 ARM Compiler 列表框有 4 个可选项，用于指定工程所使用的编译器。4 个选项实际上对应两个编译器，Use default compiler version 6 和 V6.16 是一个编译器，即编译器 6；Use default compiler version 5 和 V5.06 update 7(build 960)是一个编译器，即编译器 5。

编译器选择原则是：如果使用早期的工程只能选择编译器 5，如果是最近创建的工程，编译器 5 和编译器 6 都可以。编译器 5 的编译速度较慢，但可以生成文件跟踪链接，有利于快速组织代码，且为 STM32CubeMX 创建工程模板默认选项。此处推荐选择**编译器 5**。

如果选择编译器 6 编译项目，工程选项对话框中原 C/C++选项卡就会更改为 C/C++(AC6)，读者可以通过此处快速地了解工程所采用的编译器版本。

在图 3-31MDK-ARM 软件主界面工具栏中最下面一行为 Build 工具栏，其命令图解如图 3-35 所示。Build 工具栏总共有 5 个编译命令。第 1 个命令仅编译当前活动文件，不进行链接和生成目标文件。第 2 个命令是编译修改过的目标文件，即所谓的增量编译(Build)。第 3 个命令是重新编译(Rebuild)所有目标文件，不管文件是否有改动。如果工程是首次编译，增量编译和重新编译效果是一样的，都是将所有文件全部编译，如果工程已经编译过了，且工程较大，选择增量编译要比重新编译快得多！第 4 个命令是批量编译。第 5 个命令是停止编译。

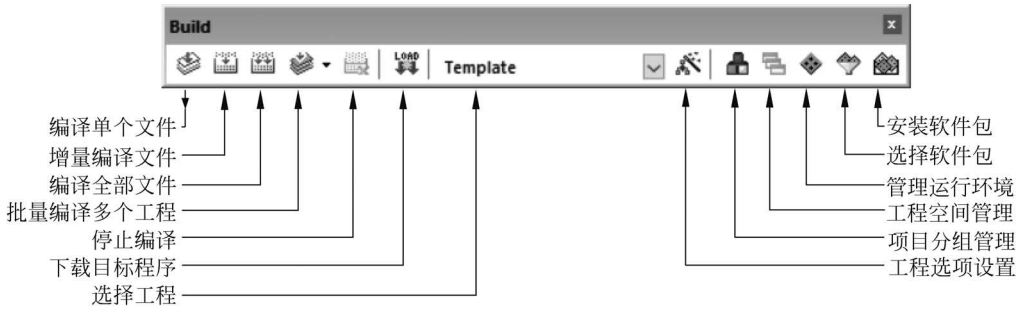

图 3-35　Build 工具栏命令图解

作者在实际使用过程中发现，如果选择编译器 5，则使用增量编译比较快；如果使用编译器 6，因为编译器本身编译速度比较快，就无所谓哪一种编译方式；工作空间若只有一个工程，则无须使用批量编译。

选择编译器 5，单击 Rebuild 按钮，开始项目编译，信息输出区输出编译结果，如图 3-36 所示，“0 Error(s)，0 Warning(s)”表示工程模板创建正确，可以在此基础上进行应用程序开发。

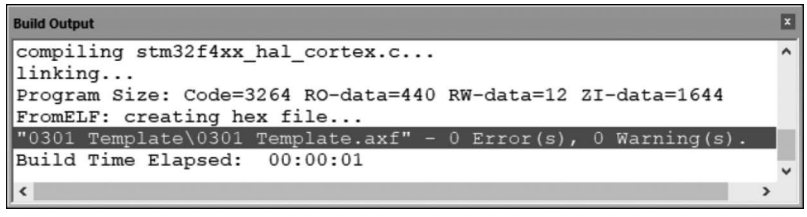

图 3-36　编译信息输出窗口

3. 代码分析及组织方式

基于 STM32CubeMX 的 HAL 库开发，需要对程序框架和代码结构进行分析，然后进行应用程序代码快速组织。

1）代码分析

STM32CubeMX 创建的 MDK 工程包含文件及其分组，具体情况如表 3-2 所示。

表 3-2　工程包含文件及其分组

文件分组	文件名称	文件功能
Application/MDK-ARM	startup_stm32f407xx. s	芯片启动文件
Application/User/Core	main. c	用户主文件
	gpio. c	GPIO 函数文件
	stm32f4xx_it. c	中断服务程序文件
	stm32f4xx_hal_msp. c	MCU 支持文件

续表

文件分组	文件名称	文件功能
Drivers/STM32F4xx_HAL_Driver	stm32f4xx_hal_rcc.c	时钟 HAL 库驱动文件
	stm32f4xx_hal_rcc_ex.c	扩展 RCC 驱动文件
	stm32f4xx_hal_gpio.c	GPIO 的 HAL 库驱动文件
	……	……
Drivers/CMSIS	system_stm32f4xx.c	STM32F4 系统文件

芯片启动文件和系统文件只需要包含进工程，用户在编程的时候一般无须关心。C 语言有且仅有一个 main()函数，用户程序是从 main()函数开始执行的，main()函数是编写在 main.c 文件中的，其部分代码如下：

```
#include "main.h"
#include "gpio.h"
void SystemClock_Config(void);
int main(void)
{
    /* USER CODE BEGIN 1 */                    //用户程序沙箱开始
    /* 用户程序代码 */
    /* USER CODE END 1 */                      //用户程序沙箱结束
    /* 复位所有外设,初始化闪存接口和 SysTick 定时器 */

    HAL_Init();
    /* 配置系统时钟 */
    SystemClock_Config();
    /* 初始化所有配置的外设 */
    MX_GPIO_Init();
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}
void SystemClock_Config(void)
{
    //代码省略
}
void Error_Handler(void)
{
    //代码省略
}
```

对 main.c 文件进一步分析，文件首先包含 main.h 和 gpio.h 两个文件，在包含语句中的文件名上右击，选择 Open document ‘main.h’命令，操作方法如图 3-37 所示，打开 main.h 头文件。由源代码可知 main.h 主要工作是包含 stm32f4xx_hal.h 头文件，并对 main.c 中定义的函数进行声明。用同样的方法打开 gpio.h 文件，可知其主要工作是对 gpio.c 定义的函数进行声明。

用户程序设计从 main()函数开始，芯片启动完成之后自动转入主函数执行。在 main()函数中，调用 HAL_Init()初始化 HAL 库，其功能是复位所有外设、初始化 Flash 接口、配置系统定时器 SysTick 周期为 1ms。调用 SystemClock_Config()函数进行系统时钟配置，其代码是由 STM32CubeMX 根据用户设

定参数生成的。调用 MX_GPIO_Init()函数对用户配置的 GPIO 引脚进行初始化，初始化代码由 STM32CubeMX 根据用户设定生成。

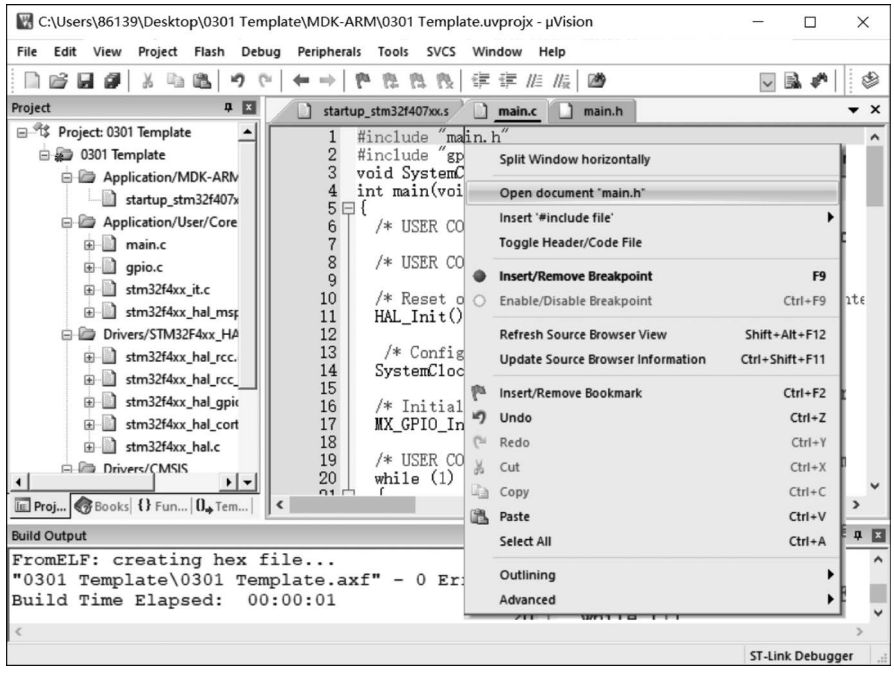

图 3-37　在包含语句中的文件名上右击打开 main.h 头文件

函数 SystemClock_Config()是直接定义在 main.c 文件中，另外还有一个异常处理函数 Error_Handler()也存放在 main.c 中，而 GPIO 初始化函数 MX_GPIO_Init()存放在 gpio.c 文件中。

如果想查看 MX_GPIO_Init()函数源码，可以将光标置于函数名称上右击，选择 Go To Definition of 'MX_GPIO_Init'菜单，即可打开函数所在文件 gpio.c，并定位到函数所在位置，如图 3-38 所示。

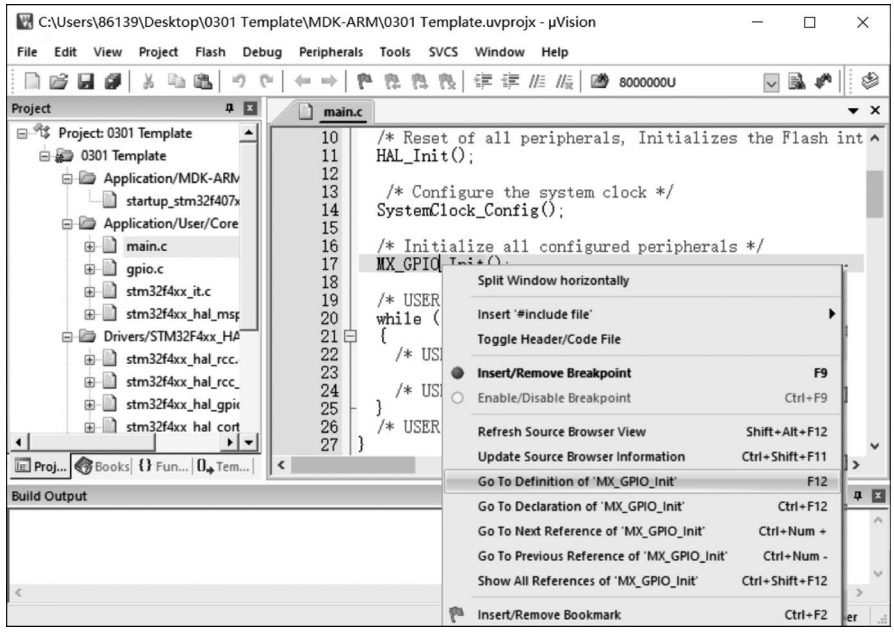

图 3-38　右击函数并选择命令进行跳转

由函数的 MX_GPIO_Init()源代码可知，其主要工作包括开端口时钟，设置 PF0 初始化电平，为 GPIO 初始化结构体的成员依次赋值和调用 HAL_GPIO_Init 完成引脚初始化。

上述所有代码均由 STM32CubeMX 生成，仅为初始化代码和程序框架，要实现运算、控制功能，还需在此基础上进一步开发。由于在系统设计过程中可能需要更改系统方案，对外设再次进行配置，为了保证用户编写程序不受重新配置影响，STM32CubeMX 在生成工程时特意提供一个个程序沙箱，用于放置用户代码。

每一个程序沙箱均为一对程序注释，以 USER CODE BEGIN 开始，至 USER CODE END 结束，如 main.c 程序代码中的加粗部分。用户必须将程序写在这两个注释的中间，否则重新生成工程时用户代码将丢失。

2）代码组织

通过上述初始化之后，PF0 输出电平为低电平，经编译、下载到微控制器 Flash 存储器中，L1 指示灯是一直亮的。下面对该项目进行一点改动，通过编写程序让 L1 以一定周期闪烁，也通过此实例介绍 MDK-ARM 的代码组织技巧。

第 1 步，将 GPIO 写输出端口电平函数 HAL_GPIO_WritePin()复制到 main.c，右击打开函数定义，查看其功能。

第 2 步，在函数调用变量处依次右击查看其定义，由此可知函数最后一个参数是用来设置端口电平的，可以取 GPIO_PIN_RESET 和 GPIO_PIN_SET 两者之一。

第 3 步，改写 HAL_GPIO_WritePin()函数，使 PF0 也可以输出高电平，即 L1 指示灯可以熄灭。

第 4 步，修改程序，实现 L1 指示灯周期闪烁，所有增加程序均应写在程序沙箱里。参考程序代码如下，为便于读者查看，将程序沙箱的注释语句作加粗显示。

```
int main(void)
{
    /* USER CODE BEGIN 1 */                    //用户程序沙箱
    uint32_t i;
    /* USER CODE END 1 */                      //用户程序沙箱
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
/* USER CODE BEGIN WHILE */                    //用户程序沙箱
    while (1)
    {
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_RESET);
        for(i=0;i<12000000;i++) ;
        HAL_GPIO_WritePin(GPIOF, GPIO_PIN_0, GPIO_PIN_SET);
        for(i=0;i<12000000;i++) ;
        /* USER CODE END WHILE */              //用户程序沙箱
  }
}
```

3.4 CMSIS-DAP 调试器使用

作者所设计的嵌入式开发板板载 CMSIS-DAP 调试器，可以将其添加到 MDK 集成开发环境中，就像以前使用 ST-Link、ULINK 等调试器一样。

3.4.1 调试器连接与驱动安装

使用一端是 A 型口，另一端是 B 型口的 USB 数据线分别连接开发板和 PC，如果计算机安装

Windows 10 以上操作系统，计算机会自动安装好驱动程序，调试器是开源、免驱动的。如果是 Windows 7 操作系统，还需要手动安装驱动，在设备管理器更新驱动程序，浏览找到驱动文件即可完成安装。

3.4.2　调试选项设置与程序下载

打开 MDK-ARM 工程，单击 Project→Options for Target 菜单项，或单击工具栏中的图标，或按下 Alt＋F7 快捷键均可以打开“工程选项”对话框，如图 3-39 所示。

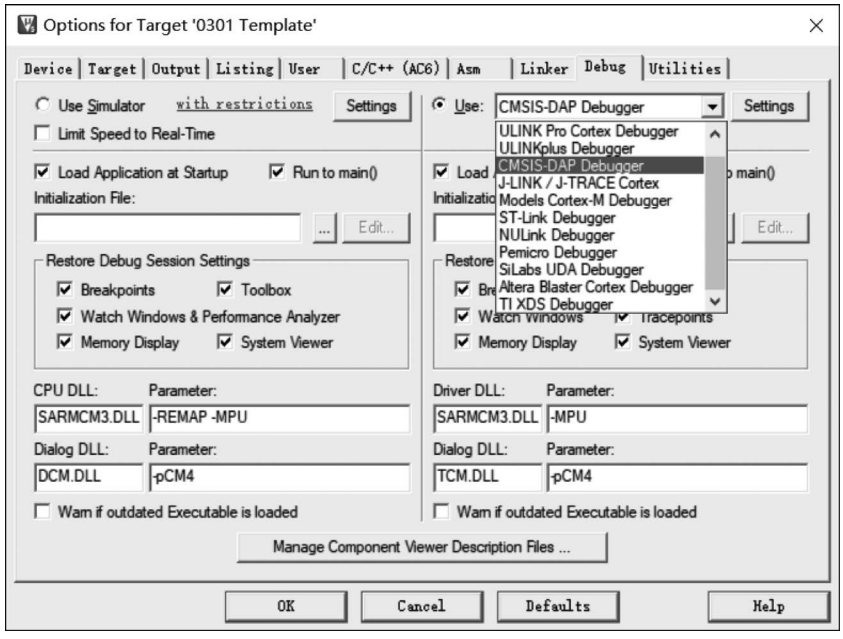

图 3-39　“工程选项”对话框——调试器选择

选择 Debug 选项卡，选中右侧 Use 单选按钮，在右边的下拉列表中选择 CMSIS-DAP Debugger，单击 Settings 按钮，打开调试选项设置对话框，如图 3-40 所示。

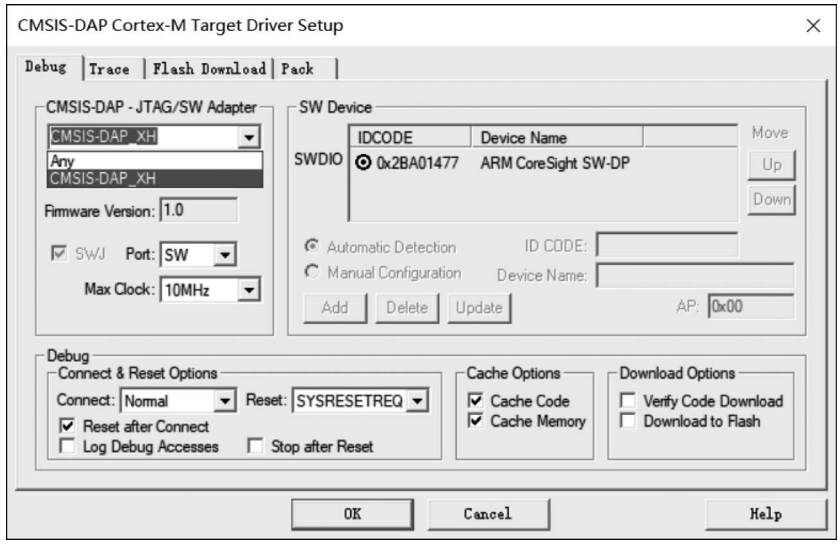

图 3-40　“调试选项设置”对话框

首先设置 Debug 选项卡，在 CMSIS-DAP-JTAG/SW Adapter 选项区域，选择具体调试设备 CMSIS-

DAP_XH，调试器名称在调试器驱动程序中定义，默认为 CMSIS-DAP。Port 是调试方式选择，调试器与开发板主控芯片只有 SWD 方式连接，没有 JTAG 方式连接，所以只能选择 SW，时钟频率 Max Clock 选择 10MHz。完成上述设置在 SW Device 选项区域会显示调试器序列号和名称，表示识别成功。

其次设置 Flash Download 选项卡，如图 3-41 所示，在 Download Function 选项区域，有 3 个关于擦除的单选按钮，分别是 Erase Full Chip(全芯片擦除)、Erase Sectors(只擦除扇区)、Do not Erase(不进行擦除)，此处保留默认设置 Erase Sectors 即可。右侧的 Program(编程)和 Verify(校验)两个复选框默认是选中的，还需要将 Reset and Run(复位并运行)选项选中，以使调试器下载完程序复位 MCU 并运行。MDK-ARM 会根据芯片类型自动填充 SRAM 和 Flash 存储器的大小和地址范围，无须设置。

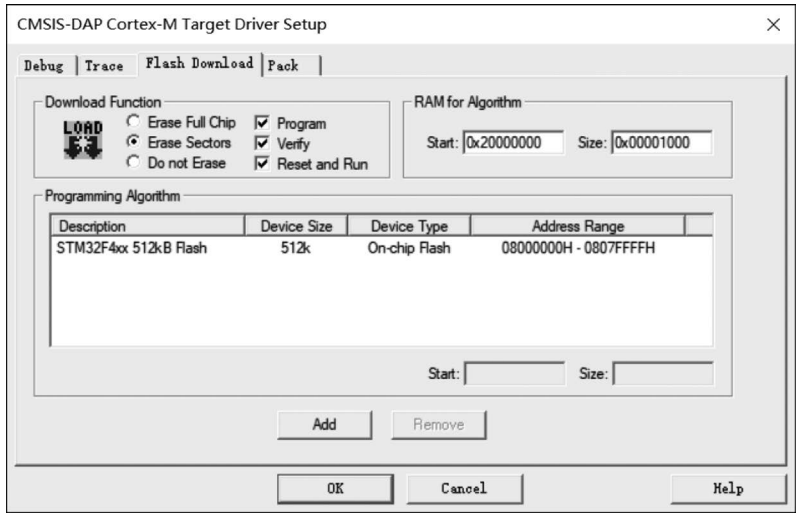

图 3-41　Flash Download 选项卡

完成上述选项设置，单击工具栏 Rebuild 按钮，再单击 Flash→Download 菜单或者工具栏中的 Load 按钮，或按 F8 快捷键，程序开始下载，完成之后复位 MCU 并运行，如图 3-42 所示。观察程序运行效果，检查是否满足系统设计要求，不满足则修改程序直至达到预期目标。

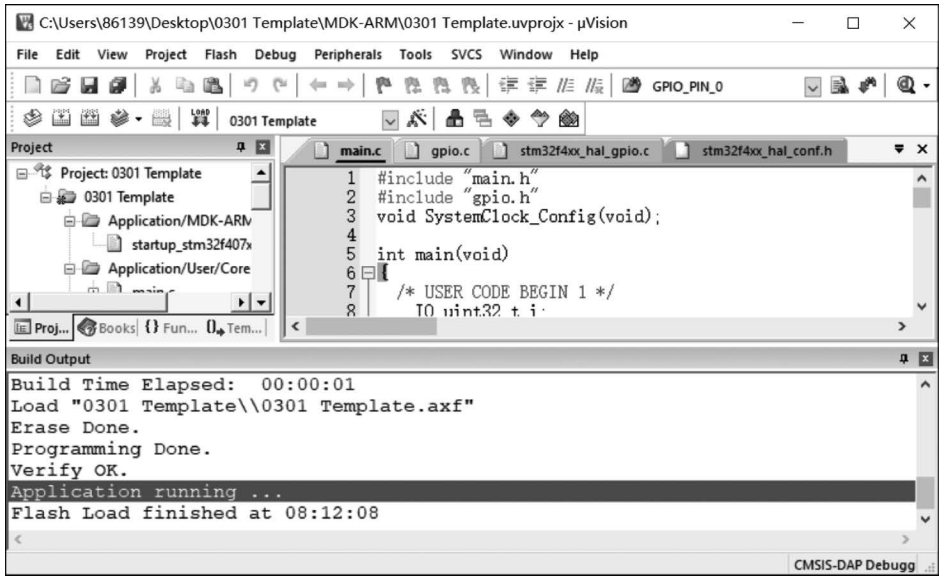

图 3-42　程序下载运行

3.5　开发经验小结——编译器优化与 volatile 关键字

3.5.1　编译器优化

MDK ARM 编译器会对程序代码进行编译优化，以获得更好的空间和时间性能。

例如，如下代码：

```
a = 2; a = 3; b = 4; b = 5; c = a + b;
```

可以优化为如下代码：

```
a = 3; b = 5; c = a + b;
```

如此一来，代码长度和运行时间均得到优化，提升了代码性能。上述例程仅用于说明编译器优化原理，实际优化要远比其复杂得多，对此不作进一步的探讨。

编译器优化减小了代码空间，提升了程序运行速度，但也使得程序运行时间变得不确定，而且有些场合是不允许优化的。那有没有办法不让编译器对变量进行优化呢？答案当然是肯定的。

3.5.2　volatile 关键字

1. volatile 的基本概念

volatile 意为易变的、不稳定的。简单地说，就是不让编译器进行优化，即每次读取或者修改 volatile 变量的值时，都必须重新从内存或者寄存器中读取或者修改。在嵌入式开发中，volatile 关键字主要用于以下场合：

（1）中断服务程序中修改的供其他程序检测的变量。

（2）多任务环境下各任务间共享的标志。

（3）存储器映射的硬件寄存器。

2. volatile 应用实例

对于本章实践的 L1 周期闪烁项目，在同一硬件平台和相同的控制程序下，由于编译优化选项设置的不同，闪烁的快慢是有差异的！如果想要软件延时时间确定，只需要使用 volatile 关键字修饰循环变量的定义即可，此时编译器将不再对其进行优化，具体语句如下：

```
volatile uint32_t i;
```

为便于记忆和使用，在 HAL 库中给出了 volatile 关键字的宏定义，具体代码如下，读者使用关键字和宏定义的效果一样。

```
#define __IO volatile
```

本章小结

本章重点讲解了 3 部分内容，第 1 部分内容是嵌入式开发方式的选择，本书详细介绍了 4 种开发模式，并从可移植性、优化性能、易用性、意愿性、硬件支持面等多方面进行对比，最终选择综合表现优异的基于 STM32CubeMX 的 HAL 库开发方式，该方式也是目前主流的开发方式。第 2 部分内容是软件资源安装与配置，详细讲解了 JRE、STM32CubeMX、固件包、MDK-ARM 和器件包的安装，讲解较为详细，读者自行配置软件平台时，根据实际情况可以省略部分步骤。第 3 部分内容是通过一个简单项目实例，讲

解了如何运用 STM32CubeMX 生成初始化代码，并在 MDK-ARM 进行代码组织，编译成功之后再下载运行，项目十分简单，但包含嵌入式开发的完整过程，具有很好的参考意义。

思考拓展

(1) STM32 开发方式中执行速度最快，占用空间最小的是哪一种？
(2) STM32 开发方式中可移植性最好、开发效率最高的是哪一种？
(3) STM32 产品线中哪些系列既有标准库又有 HAL 库？
(4) 根据本书介绍的方法下载软件，完成软件平台的搭建。
(5) 参照本书的示例程序，创建一个工程项目，并验证其正确性。
(6) 选择不同编译器对项目进行编译，体会编译器性能差异。
(7) 将 CMSIS-DAP 调试器添加进 MDK-ARM，并下载程序验证。
(8) 打开工程进行代码分析，并列举一些常见的快速组织代码技巧。