第 3 章

STM32CubeMX 和 HAL 库

本章讲述 STM32CubeMX 和 HAL 库，包括安装 STM32CubeMX、安装 MCU 固件包、软件功能与基本使用和 HAL 库。

3.1 安装 STM32CubeMX

STM32CubeMX 软件是 ST 公司为 STM32 系列微控制器快速建立工程并初始化使用到的外设、GPIO 等而设计的，大大缩短了开发时间。同时，该软件不仅能配置 STM32 外设，还能进行第三方软件系统的配置，如 FreeRTOS、FAT 32、LWIP 等。STM32CubeMX 还有一个功能，就是进行功耗预估。此外，这款软件可以输出 PDF、TXT 文档，显示所开发工程中的 GPIO 等外设的配置信息，供开发者进行原理图设计等。

STM32CubeMX 是 ST 官方推出的一款针对 ST 的 MCU/MPU 的跨平台图形化工具，支持 Linux、macOS、Windows 系统，支持 ST 全系列产品，目前包括 STM32L0、STM32L1、STM32L4、STM32L5、STM32F0、STM32F1、STM32F2、STM32F3、STM32F4、STM32F7、STM32G0、STM32G4、STM32H7、STM32WB、STM32WL、STM32MP1，其对接的底层接口是 HAL 库，STM32CubeMX 除了集成 MCU/MPU 的硬件抽象层，还集成了 RTOS、文件系统、USB、网络、显示、嵌入式 AI 等中间件，这样开发者就能够很轻松地完成 MCU/MPU 的底层驱动的配置，留出更多精力开发上层功能逻辑，能够更进一步提高嵌入式开发效率。

STM32CubeMX 软件的特点如下。

(1) 集成了 ST 公司的每款型号的 MCU/MPU 的可配置的图形界面，能够自动提示 I/O 冲突并且对于复用 I/O 可自动分配。

(2) 具有动态验证的时钟树。

(3) 能够很方便地使用所集成的中间件。

(4) 能够估算 MCU/MPU 在不同主频运行下的功耗。

(5) 能够输出不同编译器的工程，如能够直接生成 MDK、EWArm、STM32CubeIDE、MakeFile 等工程。

为了使开发人员能够更加快捷有效地进行 STM32 的开发，ST 公司推出了一套完整的 STM32Cube 开发组件。STM32Cube 主要包括两部分：一是 STM32CubeMX 图形化配置工具，直接在图形界面简单配置下生成初始化代码，并对外设做进一步的抽象，让开发人员只专注于应用的开发；二是基于 STM32 微控制器的固件集 STM32Cube 软件资料包。

从 ST 公司官网可下载 STM32CubeMX 软件最新版本的安装包，本书使用的版本是 6.6.1。解压安装包后，运行其中的安装程序，按照安装向导的提示进行安装。安装过程中会出现如图 3-1 所示的界面，需要勾选第 1 个复选框后才可以继续安装，第 2 个复选框可以不用勾选。

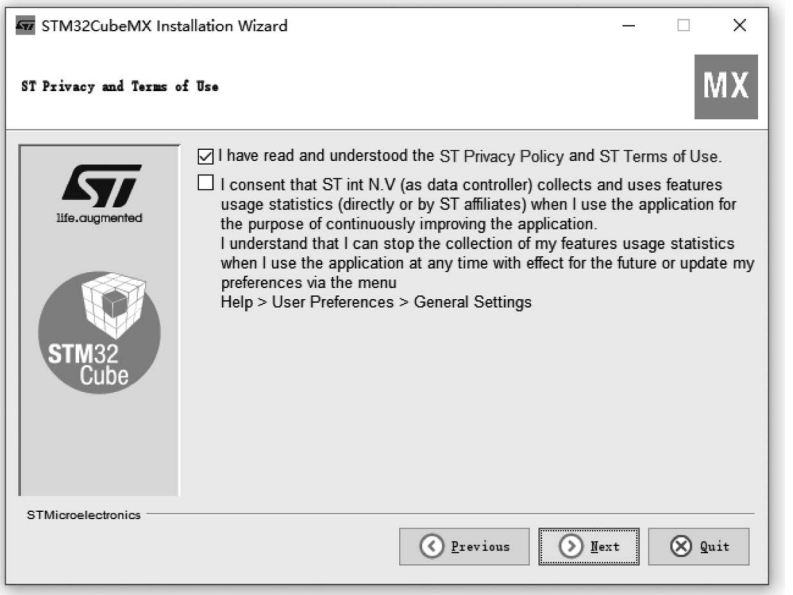

图 3-1　STM32CubeMX 安装向导

在安装过程中，用户要设置软件安装的目录。安装目录中不能带有汉字、空格和非下画线的符号，因为 STM32CubeMX 对中文的支持不太好。还需要安装器件的 MCU 固件包，所以最好将它们安装在同一个根目录下。例如，根目录为 C:/Program Files/STMicroelectronics/STM32Cube/，可将 STM32CubeMX 的安装目录设置为 C:/Program Files/STMicroelectronics/STM32Cube/STM32CubeMX。

3.2　安装 MCU 固件包

安装 MCU 固件包，包括软件库文件夹设置和管理嵌入式软件包两部分。

3.2.1　软件库文件夹设置

在安装完 STM32CubeMX 后，若要进行后续的各种操作，必须在 STM32CubeMX 中设置一个软件库文件夹（Repository Folder）。在 STM32CubeMX 中安装 MCU 固件包和 STM32Cube 扩展包时，都安装到此目录下。

双击桌面上的 STM32CubeMX 图标，软件启动后的界面如图 3-2 所示。

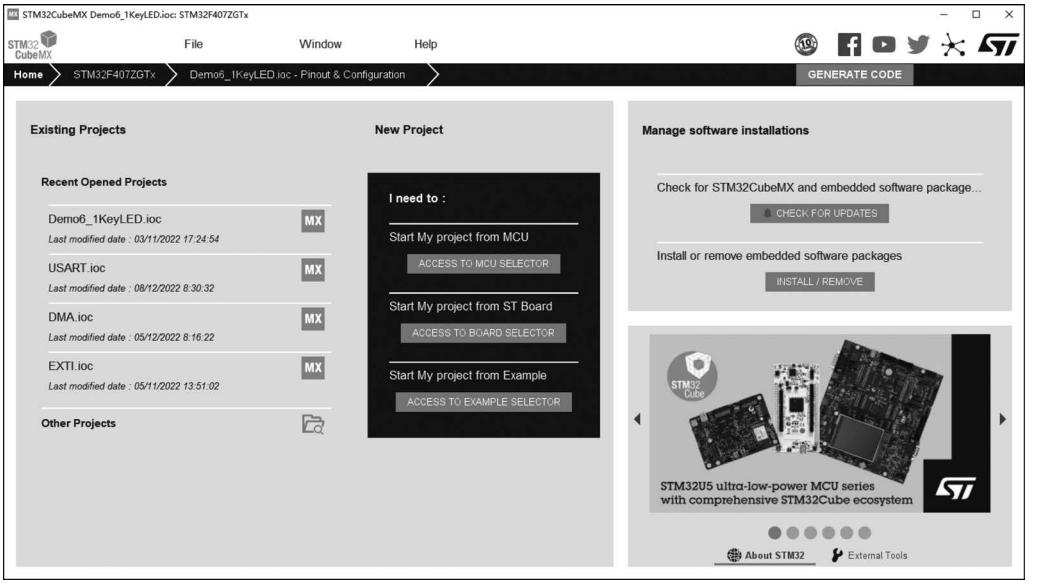

图 3-2　STM32CubeMX 界面

界面的最上方有 3 个主菜单项，执行 Help→Updater Settings 菜单命令，弹出 Updater Settings 对话框，如图 3-3 所示。首次启动 STM32CubeMX 后，立刻执行这个菜单命令可能提示软件更新已经在后台运行，需要稍微等待一段时间。

图 3-3 中，Repository Folder 就是需要设置的软件库文件夹，所有 MCU 固件包和扩展包要安装到此目录下。这个文件夹一经设置并且安装了一个固件包之后就不能再更改。不要使用默认的软件库文件夹，因为默认的是用户工作目录下的文件夹，可能带有汉字或空格，安装后会导致使用出错。设置软件库文件夹为 C:/Users/lenovo/STM32Cube/Repository/。

Check and Update Settings 用于设置 STM32CubeMX 软件的更新方式，Data Auto-Refresh 用于设置在 STM32CubeMX 启动时是否自动刷新已安装软件库的数据和文档。为了加快软件启动速度，可以将其分别设置为 Manual Check（手动检查更新软件）和 No Auto-Refresh at Application start（不在 STM32CubeMX 启动时自动刷新）。STM32CubeMX 启动后，用户可以通过相应的菜单命令检查 STM32CubeMX 软件，更新或刷新数据。

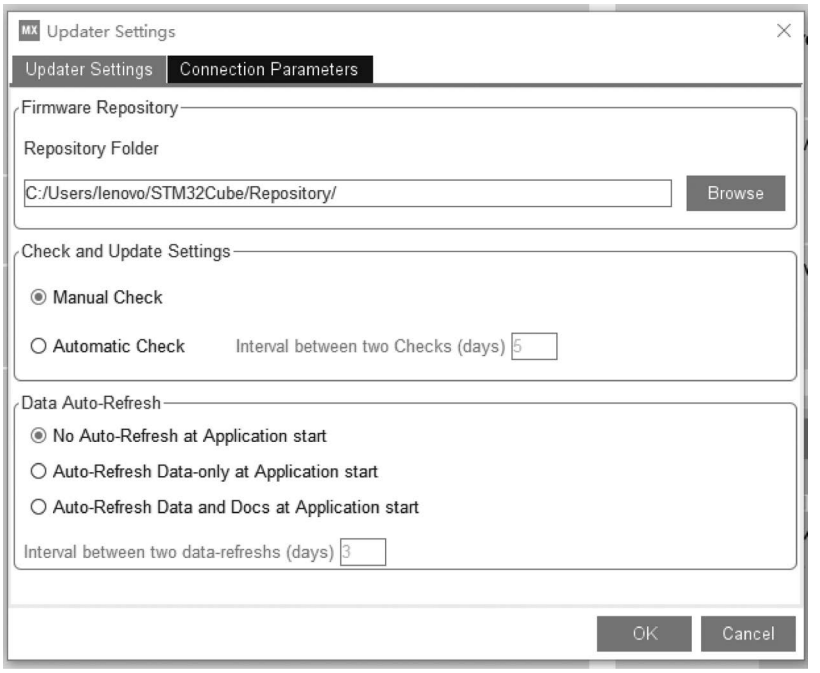

图 3-3 Updater Settings 对话框

Connection Parameters 选项卡用于设置网络连接参数。如果没有网络代理，直接选择 No Proxy(无代理)即可；如果有网络代理，就设置自己的网络代理参数。

3.2.2 管理嵌入式软件包

设置了软件库文件夹，就可以安装 MCU 固件包和扩展包了。执行 Help→Manage embedded software packages 菜单命令，弹出 Embedded Software Packages Manager(嵌入式软件包管理器)对话框，如图 3-4 所示。这里将 STM32Cube MCU 固件包和 STM32Cube 扩展包统称为嵌入式软件包。

该对话框包含多个选项卡，STM32Cube MCU Packages 选项卡管理 STM32 所有系列 MCU 的固件包。每个系列对应一个节点，节点展开后是这个系列 MCU 不同版本的固件包。固件包经常更新，在 STM32CubeMX 里最好只保留一个最新版本的固件包。如果在 STM32CubeMX 里打开一个用旧的固件包设计的项目，会有提示将项目迁移到新的固件包版本，一般都能成功自动迁移。

对话框下方有几个按钮，它们可用于完成不同的操作功能。

(1) From Local 按钮：从本地文件安装 MCU 固件包。如果从 ST 官网下载了固件包的压缩文件，如 en. stm32cubef4_vl-8-4. zip 是 1. 8. 4 版本的 STM32CubeF4 固件包压缩文件，那么单击 From Local 按钮后，选择这个压缩文件(无须解压)，就可以安装这个固件包。但是要注意，这个压缩文件不能存放在软件库根目录下。

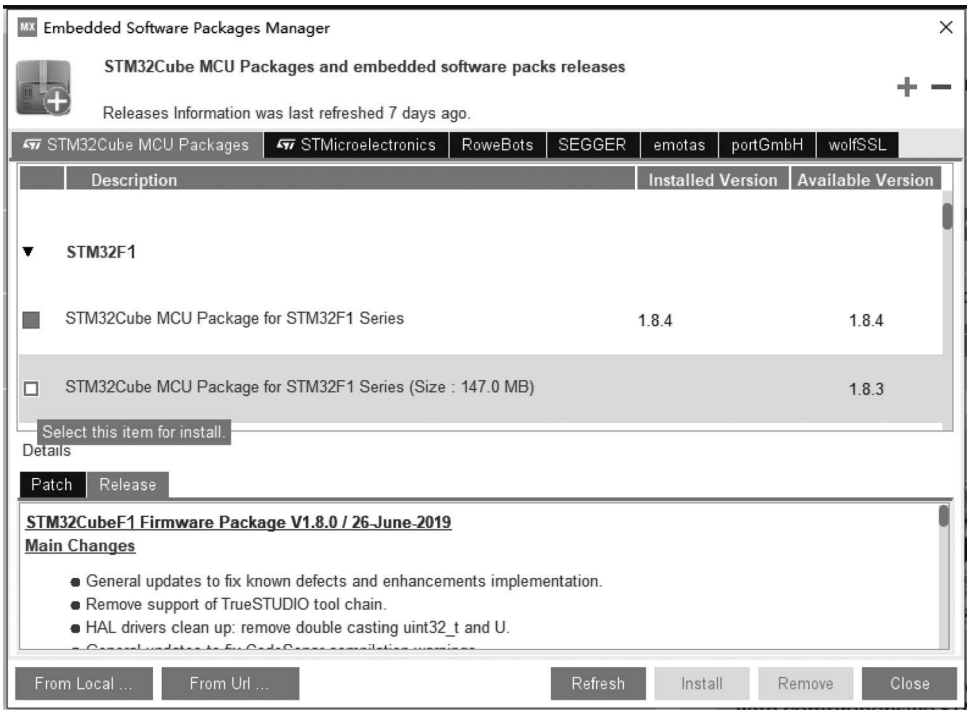

图 3-4　Embedded Software Packages Manager 对话框

(2) From Url 按钮:需要输入一个 URL 网址,从指定网站下载并安装固件包,一般不使用这种方式。

(3) Refresh 按钮:刷新目录树,以显示是否有新版本的固件包。应该偶尔刷新一下,以保持更新到最新版本。

(4) Install 按钮:在目录树里选择一个版本的固件包,如果这个版本的固件包还没有安装,这个按钮就可用。单击 Install 按钮,将自动从 ST 官网下载相应版本的固件包并安装。

(5) Remove 按钮:在目录树里选择一个版本的固件包,如果已经安装了这个版本的固件包,这个按钮就可用。单击 Remove 按钮,将删除这个版本的固件包。

本书示例都是基于 STM32F407ZGT6 开发的,所以需要安装 STM32CubeF4 固件包。在图 3-4 对话框中选择最新版本的 STM32Cube MCU Package for STM32F4Series,然后单击 Install 按钮,将会联网自动下载和安装 STM32CubeF4 固件包。固件包自动安装到所设置的软件库目录下,并自动建立一个子目录。将固件包安装后目录下的所有程序称为固件库,如 1.26.0 版本的 STM32CubeF4 固件包安装后的固件库目录为 C:/Users/lenovo/STM32Cube/Repository/STM32Cube_FW_F4_V1.26.0。

STMicroelectronics 选项卡的管理内容如图 3-5 所示,其中 ST 公司提供的一些 STM32Cube 扩展包,包括人工智能库 X-CUBE-AI、图形用户界面库 X-CUBE-TOUCHGFX 等,以及一些芯片的驱动程序,如 MEMS、BLE、NFC 芯片的驱动库。

图 3-5 STMicroelectronics 选项卡的管理内容

3.3 软件功能与基本使用

在设置了软件库文件夹并安装了 STM32CubeF4 固件包之后，就可以开始用 STM32CubeMX 创建项目并进行操作了。在开始针对开发板开发实际项目之前，我们需要先熟悉 STM32CubeMX 的一些界面功能和操作。

3.3.1 软件界面

软件界面包含初始主界面和主菜单功能两部分。

1. 初始主界面

启动 STM32CubeMX 之后的初始界面如图 3-2 所示。STM32CubeMX 从 5.0 版本开始使用了一种比较新颖的用户界面，与一般的 Windows 应用软件界面不太相同，也与 4.x 版本的 STM32CubeMX 界面相差很大。

初始界面主要分为 3 个功能区，具体如下。

1）主菜单栏

界面最上方是主菜单栏，有 3 个主菜单项，分别是 File、Window 和 Help。这 3 个菜单项有下拉菜单，用户可通过下拉菜单项进行一些操作。主菜单栏右侧是一些快捷按钮，单击

这些按钮就会用浏览器打开相应的网站，如 ST 社区、ST 官网等。

2）标签导航栏

主菜单栏下方是标签导航栏。在新建或打开项目后，标签导航栏可以在 STM32CubeMX 的 3 个主要视图之间快速切换。这 3 个视图如下。

（1）Home（主页）视图，即如图 3-2 所示的界面。

（2）新建项目视图，新建项目时显示的一个对话框，用于选择具体型号的 MCU 或开发板创建项目。

（3）项目管理视图，用于对创建或打开的项目进行 MCU 图形化配置、中间件配置、项目管理等操作。

3）工作区

窗口其他区域都是工作区。STM32CubeMX 使用的是单文档界面，工作区会根据当前操作的内容显示不同的界面。

图 3-2 的工作区显示的是 Home 视图，Home 视图的工作区可以分为以下 3 个功能区域。

（1）Existing Projects 区域，显示最近打开过的项目，单击某个项目就可以打开此项目。

（2）New Project 区域，有 3 个按钮用于新建项目，选择 MCU 创建项目，选择开发板创建项目，或交叉选择创建项目。

（3）Manage software installations 区域，有两个按钮：CHECK FOR UPDATES 按钮用于检查 STM32CubeMX 和嵌入式软件包的更新信息；INSTALL/REMOVE 按钮用于打开如图 3-4 所示的对话框。

Home 视图上的这些按钮的功能都可以通过主菜单里的菜单项实现操作。

2. 主菜单功能

STM32CubeMX 有 3 个主菜单项，软件的很多功能操作都是通过这些菜单项实现的。

1）File 菜单

（1）New Project（新建项目），用于创建新的项目。STM32CubeMX 的项目文件扩展名是.ioc，一个项目只有一个文件。新建项目对话框是软件的 3 个视图之一，界面功能比较多，在后面具体介绍。

（2）Load Project（加载项目），选择一个已经存在的.ioc 项目文件并载入项目。

（3）Import Project（导入项目），选择一个.ioc 项目文件并导入其中的 MCU 设置到当前项目。注意，只有新项目与导入项目的 MCU 型号一致且新项目没有做任何设置时才可以导入其他项目的设置。

（4）Save Project（保存项目），保存当前项目。如果新建的项目第 1 次保存，会提示设置项目名称，需要选择一个文件夹，项目会自动以最后一级文件夹的名称作为项目名称。

（5）Save Project As（项目另存为），将当前项目保存为另一个项目文件。

（6）Close Project（关闭项目），关闭当前项目。

（7）Generate Report（生成报告），为当前项目的设置内容生成一个 PDF 报告文件，

PDF 报告文件名称与项目名称相同，并自动保存在项目文件所在的文件夹里。

(8) Recent Projects(最近的项目)，显示最近打开过的项目列表，用于快速打开项目。

(9) Exit(退出)，退出 STM32CubeMX。

2) Window 菜单

(1) Outputs(输出)，一个复选菜单项，被勾选时，在工作区的最下方显示一个输出子窗口，显示一些输出信息。

(2) Font Size(字体大小)，有 3 个子菜单项，用于设置软件界面字体大小，需重启 STM32CubeMX 后才生效。

3) Help 菜单

(1) Help(帮助)，显示 STM32CubeMX 的英文版用户手册 PDF 文档，文档有 300 多页，是一个很全面的使用手册。

(2) About(关于)，显示关于本软件的对话框。

(3) Docs&Resources(文档和资源)，只有在打开或新建一个项目后此菜单项才有效。选择此项会弹出一个对话框，显示与项目所用 MCU 型号相关的技术文档列表，包括数据手册、参考手册、编程手册、应用笔记等。这些都是 ST 公司官方的资料文档，单击即可打开 PDF 文档。首次单击一个文档时会自动从 ST 官网下载文档并保存到软件库根目录下，如 D:/STM32Dev/Repository。这避免了每次查看文档都要从 ST 公司官网搜索的麻烦，也便于管理。

(4) Refresh Data(刷新数据)，会弹出如图 3-6 所示的 Data Refresh 对话框，用于刷新 MCU 和开发板的数据，或下载所有官方文档。

图 3-6 Data Refresh 对话框

(5) User Preferences(用户选项)，用于设置用户选项，只有一个需要设置的选项，即是否允许软件收集用户使用习惯。

(6) Check For Updates(检查更新)，用于检查 STM32CubeMX 软件、各系列 MCU 固件包、STM32Cube 扩展包是否有新版本需要更新。

(7) Manage Embedded Software Packages(管理嵌入式软件包)，会弹出如图 3-4 所示的对话框，对嵌入式软件包进行管理。

(8) Updater Settings(更新设置),会弹出如图 3-3 所示的对话框,用于设置软件库文件夹,设置软件检查更新方式和数据刷新方式。

3.3.2 新建项目

新建项目包含选择 MCU 创建项目、选择开发板新建项目和交叉选择 MCU 新建项目 3 部分。

1. 选择 MCU 创建项目

执行 File→New Project 菜单命令,或单击 Home 视图上的 ACCESS TO MCU SELECTOR 按钮,都可以弹出如图 3-7 所示的 New Project 对话框,用于选择 MCU 或开发板以新建项目。

STM32CubeMX 界面上一些地方使用了 MCU/MPU,是为了表示 STM32 系列 MCU 和 MPU。因为 STM32MP 系列推出较晚,型号较少,STM32 系列一般就是指 MCU。除非特殊说明或为与界面上的表示一致,为了表达的简洁,本书后面一般用 MCU 统一表示 MCU 和 MPU。

New Project 对话框中,MCU/MPU Selector 选项卡用于选择具体型号的 MCU 创建项目;Board Selector 选项卡用于选择一个开发板创建项目;Cross Selector 选项卡用于对比某个 STM32 MCU 或其他厂家的 MCU,选择一个合适的 STM32 MCU 创建项目。

图 3-7 所示为 MCU/MPU Selector 选项卡,用于选择 MCU。

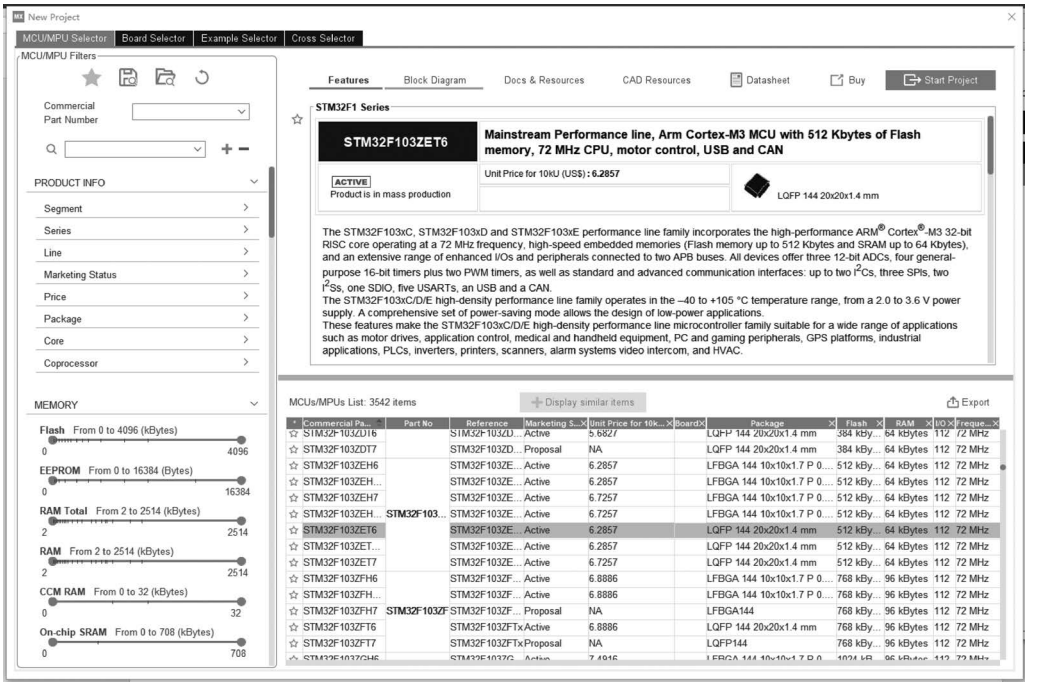

图 3-7 New Project 对话框

图 3-7 的界面有如下几个功能区域。

(1) MCUs/MPUs List,通过筛选或搜索的 MCU 列表,列出了器件的具体型号、封装、Flash、RAM 等参数。在这个区域可以进行如下操作。

① 单击列表项左侧的星星图标,可以收藏条目(★)或取消收藏(☆)。

② 单击列表上方的 Display similar items 按钮,可以将相似的 MCU 添加到列表中显示,然后按钮标题切换为 Hide similar items,再次单击就隐藏相似条目。

③ 单击列表右上方的 Export 按钮,可以将列表内容导出为一个 Excel 文件。

④ 在列表中双击一个条目时就以所选的 MCU 新建一个项目,关闭此对话框进入项目管理视图。

⑤ 在列表中单击一个条目时,将在其上方的资料区域显示该 MCU 的资料。

(2) MCU 资料显示区域,在 MCU 列表中单击一个条目时,就在此区域显示这个具体型号 MCU 的资料,有多个选项卡和按钮操作。

① Features 选项卡显示选中型号 MCU 的基本特性参数,左侧的星星图标表示是否收藏此 MCU。

② Block Diagram 选项卡显示 MCU 的功能模块图,如果是第 1 次显示某 MCU 的模块图,会自动从网上下载模块图片并保存到软件库根目录下。

③ Docs & Resources 选项卡显示 MCU 相关的文档和资源列表,包括数据手册、参考手册、编程手册、应用笔记等。单击某个文档时,如果没有下载,就会自动下载并保存到软件库根目录下;如果已经下载,就会用 PDF 阅读器打开文档。

④ Datasheet 按钮:如果数据手册未下载,会自动下载数据手册然后显示;否则会用 PDF 阅读器打开数据手册。数据手册自动保存在软件库根目录下。

⑤ Buy 按钮:用浏览器打开 ST 公司网站的购买页面。

⑥ Start Project 按钮:用选择的 MCU 创建项目。

(3) MCU/MPU Filters,用于 MCU 筛选的一些功能操作,上方有一个工具栏,有 4 个按钮。

① Show Favorites 按钮:显示收藏的 MCU 列表。单击 MCU 列表条目前面的星星图标,可以收藏或取消收藏某个 MCU。

② Save Search 按钮:保存当前搜索条件为某个搜索名称。在设置了某种筛选条件后可以保存为一个搜索名称,然后再单击 Load Searches 按钮时选择此搜索名称,就可以快速使用以前用过的搜索条件。

③ Load Searches 按钮:会显示一个弹出菜单,列出所有保存的搜索名称,单击某一项就可以快速载入以前设置的搜索条件。

④ Reset All Filters 按钮:复位所有筛选条件。

在此工具栏的下方有一个搜索框,用于设置器件型号进行搜索。可以在搜索框中输入 MCU 的型号,如 STM32F103,就会在 MCU 列表中看到所有 STM32F103xx 型号的 MCU。

MCU 的筛选主要通过以下几组条件进行设置。

① Core(内核)：筛选内核，选项中列出了 STM32 支持的所有 Cortex 内核，如图 3-8 所示。

② Series(系列)：选择内核后会自动更新可选的 STM32 系列列表，图 3-9 只显示了列表的一部分。

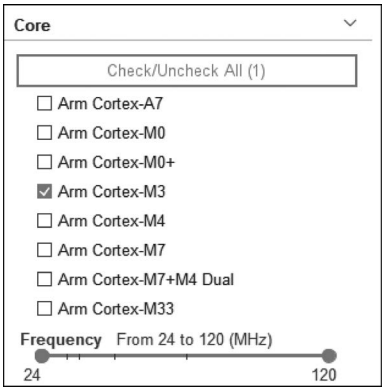

图 3-8 选择 Cortex 内核

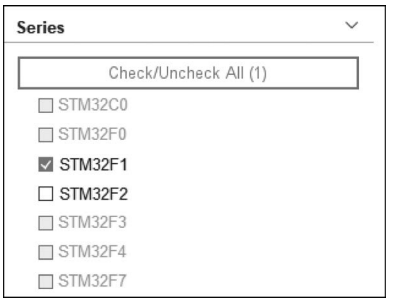

图 3-9 选择 STM32 系列

③ Line(产品线)：选择某个 STM32 系列后会自动更新产品线列表中的可选范围。例如，选择了 STM32F1 系列之后，产品线列表中只有 STM32F1xx 的器件可选。图 3-10 所示为产品线列表的一部分。

④ Package(封装)：根据封装选择器件。用户可以根据已设置的其他条件缩小封装的选择范围。图 3-11 所示为封装列表的一部分。

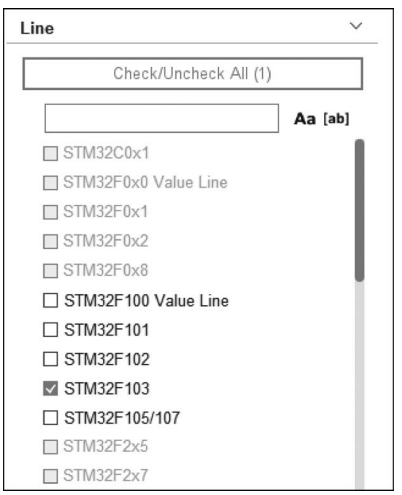

图 3-10 选择产品线

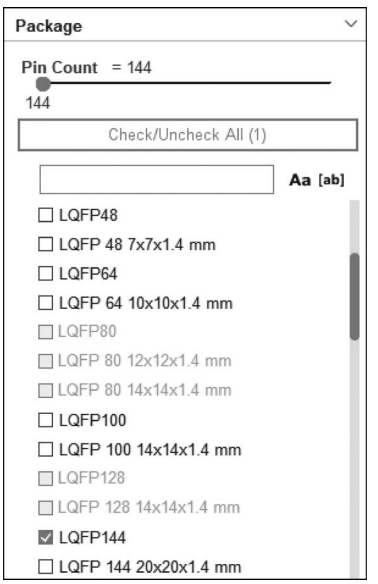

图 3-11 选择封装

⑤ Other(其他)：还可以设置价格、I/O 引脚数、Flash 大小、RAM 大小、主频等筛选条件。

MCU 筛选的操作非常灵活，并不需要按照条件顺序依次设置，可以根据自己的需要进行设置。例如，如果已知 MCU 的具体型号，可以直接在器件型号搜索框中输入型号；如果是根据外设选择 MCU，可以直接在外设中进行设置后筛选，如果得到的 MCU 型号比较多，再根据封装、Flash 容量等进一步筛选。设置好的筛选条件可以保存为一个搜索名称，通过 Load Searches 按钮选择保存的搜索名称，可以重复执行搜索。

2. 选择开发板新建项目

用户还可以在 New Project 对话框中选择开发板新建项目，如图 3-12 所示。STM32CubeMX 目前仅支持 ST 官方的开发板。

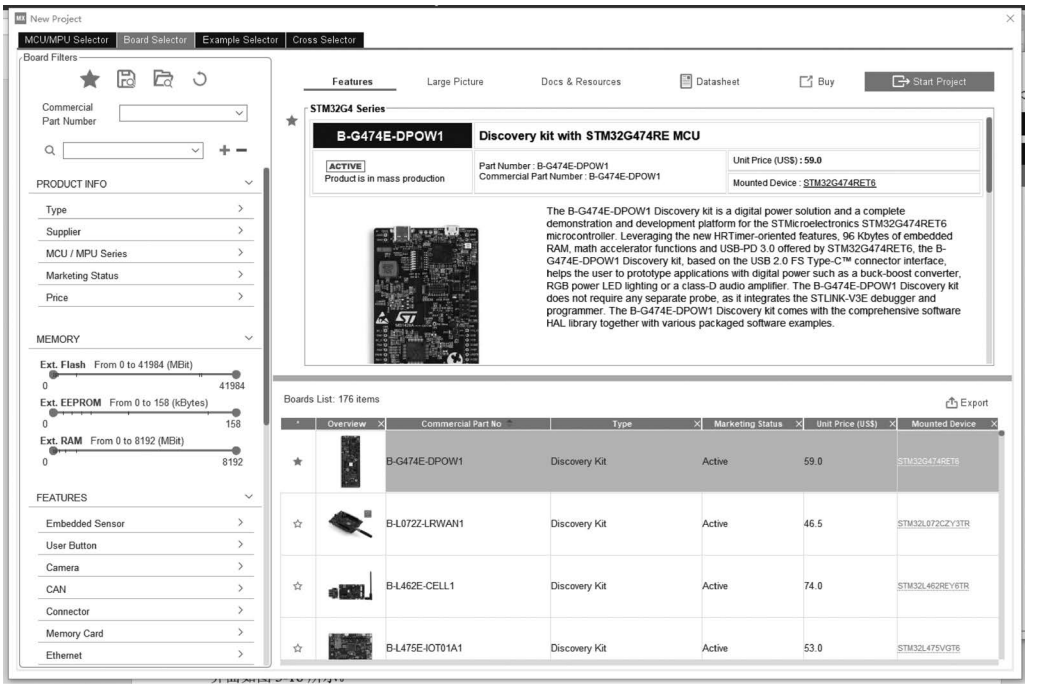

图 3-12　选择开发板新建项目

3. 交叉选择 MCU 新建项目

New Project 对话框的 Cross Selector 选项卡用于交叉选择 MCU 新建项目，如图 3-13 所示。

交叉选择就是针对其他厂家的一个 MCU 或一个 STM32 具体型号的 MCU，选择一个性能和外设资源相似的 MCU。交叉选择对于在一个已有设计基础上选择新的 MCU 重新设计非常有用。例如，一个原有的设计用的是 TI 的 MSP4305529 单片机，需要改用 STM32 MCU 重新设计，就可以通过交叉选择找到一个性能、功耗、外设资源相似的 STM32 MCU。再如，一个原有的设计使用 STM32F103，但是发现 STM32F103 的 SRAM 和处理速度不

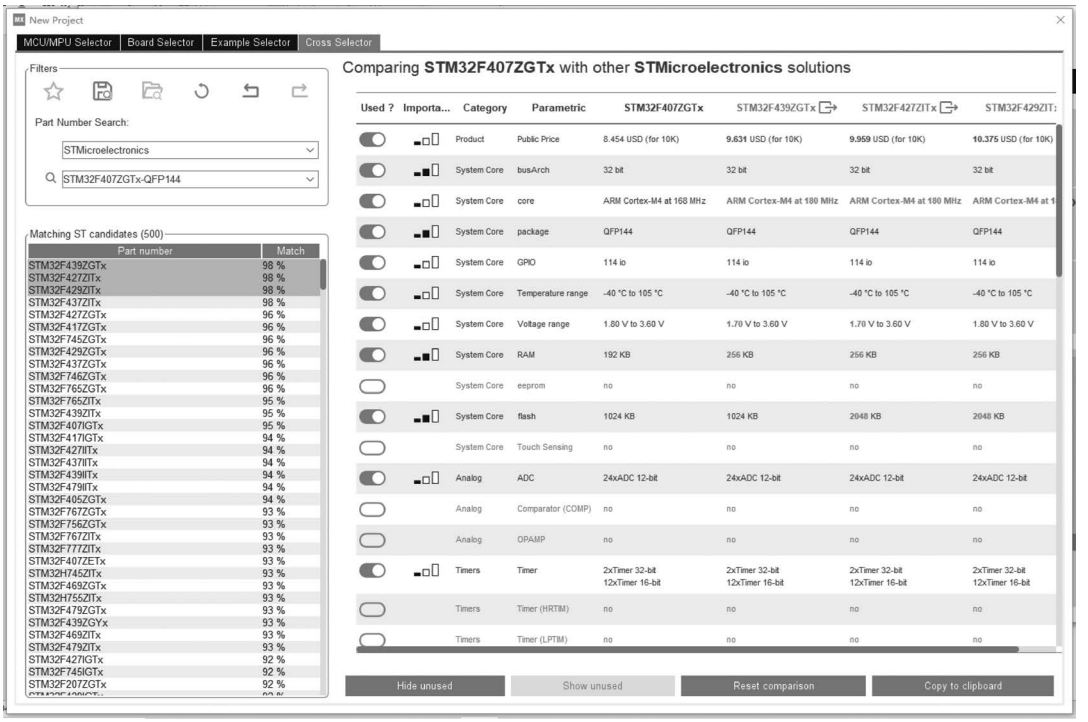

图 3-13　交叉选择 MCU 新建项目

够，需要选择一个性能更高且引脚与 STM32F103 完全兼容的 STM32 MCU，就可以使用交叉选择。

Filters 区域的 Part Number Search 部分用于选择原有 MCU 的厂家和型号，如 NXP、Microchip、ST、TI 等，选择厂家后会在第 2 个下拉列表框中列出厂家的 MCU 型号。选择厂家和 MCU 型号后，会在下方的 Matching ST candidates(500)列表中显示可选的 STM32 MCU，并且有一个匹配百分比表示匹配程度。

在候选 STM32 MCU 列表中可以选择一个或多个 MCU，然后在右边的区域会显示原 MCU 与候选 STM32 MCU 的具体参数对比。通过这样的对比，用户可以快速地找到能替换原 MCU 的 STM32 MCU。其他一些按钮的功能操作就不具体介绍了，读者可自行尝试使用。

3.3.3　MCU 图形化配置界面总览

选择一个 MCU 创建项目后，界面上显示的是项目操作视图。因为本书所用开发板上的 MCU 型号是 STM32F407ZGT6，所以选择 STM32F407ZGT6 新建一个项目进行操作。这个项目只是用于熟悉 STM32CubeMX 软件的基本操作，并不需要下载到开发板上，所以可以随意操作。读者选择其他型号的 MCU 创建项目也是可以的。

如图 3-14 所示，MCU 图形化配置界面主要由主菜单栏、标签导航栏和工作区 3 部分组成。

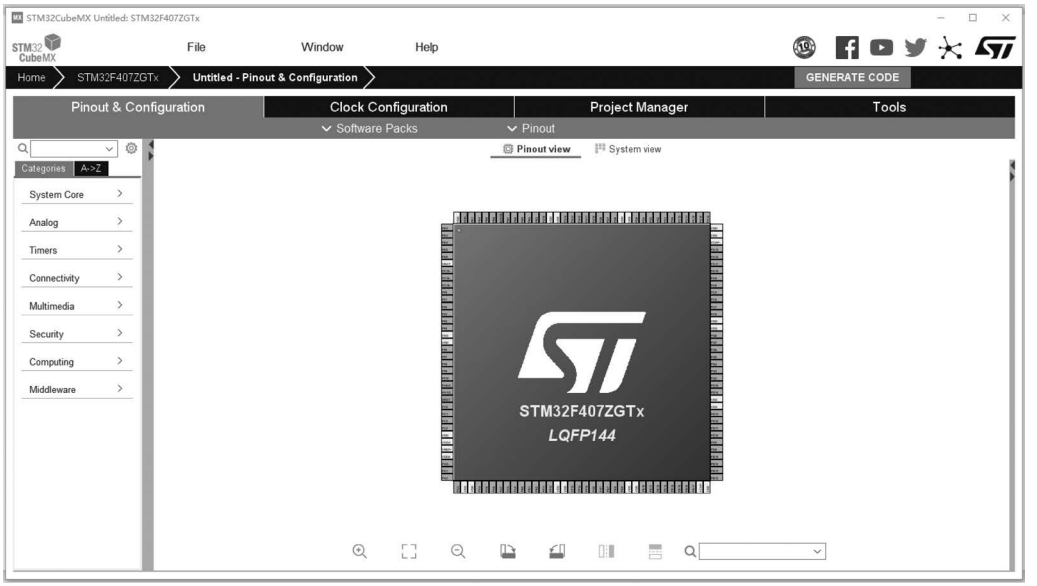

图 3-14　MCU 图形化配置界面

最上方的主菜单栏一直保持不变，标签导航栏现在有 3 个层级，最后一个层级显示了当前工作界面的名称。导航栏的最右侧有一个 GENERATE CODE 按钮，用于图形化配置 MCU 后生成 C 语言代码。工作区是一个多页界面，有 4 个工作界面。

(1) Pinout & Configuration(引脚与配置)：对 MCU 的系统内核、外设、中间件和引脚进行配置，是主要的工作界面。

(2) Clock Configuration(时钟配置)：通过图形化的时钟树对 MCU 的各个时钟信号频率进行配置。

(3) Project Manager(项目管理)：对项目进行各种设置。

(4) Tools(工具)：进行功耗计算、DDR SDRAM 适用性分析(仅用于 STM32MP1 系列)。

3.3.4　MCU 配置

引脚与配置界面是 MCU 图形化配置的主要工作界面，如图 3-14 所示。这个界面包括 Component List(组件列表)、Mode and Configuration(模式与配置)、Pinout view(引脚视图)、System view(系统视图)和一个工具栏。

1. 组件列表

位于工作区左侧的是 MCU 可以配置的系统内核、外设和中间件列表，每项称为一个组件(Component)。组件列表有两种显示方式：分组显示和按字母顺序显示。单击 Categories

或 A→Z 选项卡就可以在这两种显示方式之间切换。

在列表上方的搜索框内输入文字，按 Enter 键就可以根据输入的文字快速定位某个组件，如搜索 RCC。搜索框右侧的图标按钮有两个弹出菜单项，分别是 Expand All 和 Collapse All，在分组显示时可以展开全部分组和收起全部分组。

在分组显示状态下，主要有以下一些分组（每个分组的具体条目与 MCU 型号有关，这里以 STM32F103ZE 为例）。

（1）System Core（系统内核），包括 DMA、GPIO、IWDG、NVIC、RCC、SYS 和 WWDG。

（2）Analog（模拟），片上的 ADC 和 DAC。

（3）Timers（定时器），包括 RTC 和所有定时器。

（4）Connectivity（通信连接），各种外设接口，包括 CAN、ETH、FSMC、I2C、SDIO、SPI、UART、USART、USB_OTG_FS、USB_OTG_HS 等。

（5）Multimedia（多媒体），各种多媒体接口，包括数字摄像头接口 DCMI 和数字音频接口 I2S。

（6）Security（安全），只有一个 RNG（随机数发生器）。

（7）Computing（计算），计算相关的资源，只有一个 CRC（循环冗余校验）。

（8）Middleware（中间件），MCU 固件库里的各种中间件，主要有 FatFS、FreeRTOS、LibJPEG、LwIP、PDM2PCM、USB_Device、USB_Host 等。

（9）Additional Software（其他软件），组件列表中默认是没有这个分组的。如果在嵌入式软件管理器中安装了 STM32Cube 扩展包，那么就可以通过单击 Pinout & Configuration 工作界面菜单栏的 Additional Software 按钮打开一个对话框，将该扩展包安装到组件面板的 Additional Software 分组中。

当鼠标指针在组件列表的某个组件上停留时，会显示这个组件的上下文帮助（Contextual Help），如图 3-15 所示。上下文帮助显示了组件的简单信息，如果需要知道更详细的信息，可以单击上下文帮助中的 details and documentation（细节和文档）超链接，显示其数据手册、参考手册、应用笔记等文档的链接，单击就可以下载并显示 PDF 文档，而且会自动定位文档中的相应界面。

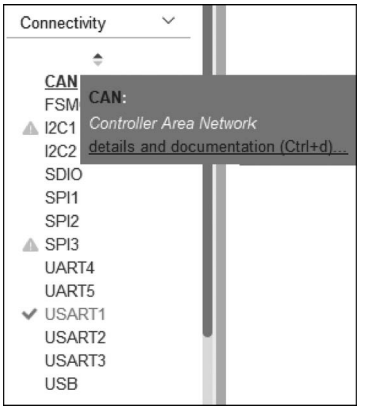

图 3-15　组件的上下文帮助功能和可用标记

在初始状态下，组件列表的各项前面没有任何图标，在对 MCU 的各个组件进行一些设置后，组件列表的各项前面会出现一些图标（见图 3-15），表示组件的可用性信息。因为 MCU 引脚基本都有复用功能，设置某个组件可用后，其他一些组和可用标记件可能就不能使用了。这些图标的意义如表 3-1 所示。

表 3-1 组件列表条目前图标的意义

图标示例	意 义
CAN1	组件前面没有任何图标，黑色字体，表示这个组件还没有被配置，其可用引脚也没有被占用
√SPI1	表示这个组件的模式和参数已经配置好了
⊘UART1	表示这个组件的可用引脚已经被其他组件占用，不能再配置这个组件了
▲ADC1	表示这个组件的某些可用引脚或资源被其他组件占用，不能完全随意配置，但还是可以配置的。例如，ADC1 有 16 个可用输入引脚，当部分引脚被占用后不能再被配置为 ADC1 的输入引脚，就会显示这样的图标
USB_HOST	灰色字体，表示这个组件因为一些限制不能使用。例如，要使用中间件 USB_HOST，需要启用 USB_OTG 接口并配置为 Host 后才可以使用

2. 模式与配置

在组件列表中单击一个组件后，就会在其右侧显示模式与配置(Mode and Configuration)界面。这个界面分为上、下两部分，上方是模式设置，下方是参数配置，这两部分的显示内容与选择的具体组件有关。

例如，图 3-14 显示的是 System Core 分组中 RCC 组件的模式与配置。RCC 用于设置 MCU 的两个外部时钟源，模式选择界面中高速外部(High Speed External，HSE)时钟源的下拉列表框中有以下 3 个选项。

(1) Disable：禁用外部时钟源。

(2) BYPASS Clock Source：使用外部有源时钟信号源。

(3) Crystal/Ceramic Resonator：使用外部晶体振荡器作为时钟源。

当 HSE 模式设置为 Disable 时，MCU 使用内部高速 RC 振荡器产生的 16MHz 信号作为时钟源。其他两项要根据实际的电路进行选择。

低速外部(Low Speed External，LSE)时钟可用作 RTC 的时钟源，其下拉列表框中的选项与 HSE 相同。若 LSE 模式设置为 Disable，RTC 就使用内部低速 RC 振荡器产生的 32kHz 时钟信号。开发板上有外接的 32.768kHz 晶体振荡电路，所以可以将 LSE 设置为 Crystal/Ceramic Resonator。如果设计中不需要使用 RTC，不需要提供 LSE 时钟，就可以将 LSE 设置为 Disable。

参数配置部分分为多个界面，且界面内容与选择的组件有关，一般有如下界面。

(1) Parameter Settings(参数设置)：组件的参数设置。例如，对于 USART1，参数设置包括波特率、数据位数(8 位或 9 位)、是否有奇偶校验位等。

(2) NVIC Settings(中断设置)：设置是否启用中断，但不能设置中断的优先级，只能显示中断优先级设置结果。中断的优先级需要在 System Core 分组的 NVIC 组件中设置。

(3) DMA Settings(DMA 设置)：是否使用 DMA，以及 DMA 的具体设置。DMA 流的中断优先级需要在 System Core 分组的 NVIC 组件中设置。

(4) GPIO Settings(GPIO 设置)：显示组件的 GPIO 引脚设置结果，不能在此修改 GPIO 设置。外设的 GPIO 引脚是自动设置的，GPIO 引脚的具体参数，如上拉或下拉、引脚

速率等，需要在 System Core 分组的 GPIO 组件中设置。

(5) User Constants(用户常量)：用户自定义的一些常量，这些自定义常量可以在 STM32CubeMX 中使用，生成代码时，这些自定义常量会被定义为宏，放入 main.h 文件中。

每种组件的模式和参数设置界面都不一样，我们在后续章节介绍各种系统功能和外设时会具体介绍它们的模式和参数设置操作。

3. 引脚视图

图 3-14 工作区的右侧显示了 MCU 的引脚图，直观地表示了各引脚的设置情况。通过组件列表对某个组件进行模式和参数设置后，系统会自动在引脚图上标识出使用的引脚。例如，设置 RCC 组件的 HSE 使用外部晶振后，系统会自动将 Pin23 和 Pin24 引脚设置为 RCC_OSC_IN 和 RCC_OSC_OUT，这两个名称就是引脚的信号(Signal)。

在 MCU 的引脚图上，亮黄色的引脚是电源或接地引脚，黄绿色的引脚是只有一种功能的系统引脚，包括系统复位引脚 NRST、BOOT0 引脚和 PDR_ON 引脚，这些引脚不能进行配置。其他未配置功能的引脚为灰色，已经配置功能的引脚为绿色。

引脚视图下方有一个工具栏，通过工具栏按钮可以进行放大、缩小、旋转等操作，通过鼠标滚轮也可以缩放，按住鼠标左键可以拖动 MCU 引脚图。

对引脚功能的分配一般通过组件的模式设置进行，STM32CubeMX 会根据 MCU 的引脚使用情况自动为组件分配引脚。例如，USART1 可以定义在 PA9 和 PA10 引脚上，也可以定义在 PB6 和 PB7 引脚上。如果 PA9 和 PA10 引脚未被占用，定义 USART1 的模式为 Asynchronous(异步)时，就自动定义在 PA9 和 PA10 引脚上。如果这两个引脚被其他功能占用了，如定义为 GPIO 输出引脚用于驱动 LED，那么定义 USART1 为异步模式时就会自动使用 PB6 和 PB7 引脚。

所以，如果是在电路的初始设计阶段，可以根据电路的外设需求在组件中设置模式，让软件自动分配引脚，这样可以减少工作量，而且更准确。当然，用户也可以直接在引脚图上定义某个引脚的功能。

在 MCU 的引脚图上，当鼠标指针停留到某个引脚上时会显示这个引脚的上下文帮助信息，主要显示的是引脚编号和名称。单击引脚，会出现一个引脚功能选择菜单。图 3-16 所示为单击 PA9 引脚时出现的引脚功能选择菜单。这个菜单列出了 PA9 引脚所有可用的功能，其中的几个解释如下。

(1) Reset_State：恢复为复位后的初始状态。

(2) GPIO_Input：作为 GPIO 输入引脚。

(3) GPIO_Output：作为 GPIO 输出引脚。

(4) TIM1_CH2：作为定时器 TIM1 的输入通道 2。

(5) USART1_TX：作为 USART1 的 TX 引脚。

(6) GPIO_EXTI9：作为外部中断 EXTI9 的输入引脚。

引脚功能选择菜单的菜单项由具体的引脚决定，手动选择了功能的引脚上会出现一个图钉图标，表示这是绑定了信号的引脚。不管是软件自动设置的引脚还是手动设置的引脚，

都可以重新为引脚手动设置信号。例如，通过设置 USART1 组件为 Asynchronous 模式，软件会自动设置 PA9 引脚为 USART1_TX，PA10 引脚为 USART1_RX。但是，如果电路设计需要将 USART1_RX 改用 PB7 引脚，就可以手动将 PB7 引脚设置为 USART1_RX，这时 PA10 引脚会自动变为复位初始状态。

手动设置引脚功能时，容易引起引脚功能冲突或设置不全的错误，出现这类错误的引脚会自动用橘黄色显示。例如，直接手动设置 PA9 和 PA10 为 USART1 的两个引脚，但是引脚会显示为橘黄色。这是因为在组件中没有启用 USART1 并为其选择模式，在组件列表中选择 USART1 并设置其模式为 Asynchronous 之后，PA9 和 PA10 引脚就变为绿色了。

用户还可以右击一个引脚，弹出一个快捷菜单，如图 3-17 所示。不过，只有设置了功能的引脚，才有该快捷菜单。此快捷菜单有 3 个菜单项。

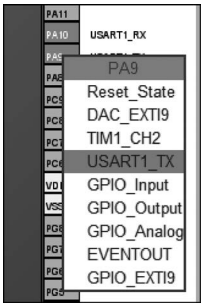

图 3-16　PA9 引脚的引脚功能选择菜单

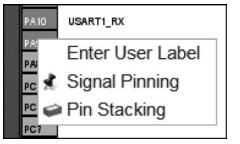

图 3-17　引脚的快捷菜单

(1) Enter User Label(输入用户标签)：用于输入一个用户定义的标签，这个标签将取代原来的引脚信号名称显示在引脚旁边。例如，在将 PA10 引脚设置为 USART1_RX 后，可以再为其定义标签 GPS_RX，这样在实际的电路中更容易看出引脚的功能。

(2) Signal Pinning(信号绑定)：单击此菜单项后，引脚上将会出现一个图钉图标，表示将这个引脚与功能信号(如 USART1_TX)绑定了，这个信号就不会再自动改变引脚，只可以手动改变引脚。对于已经绑定信号的引脚，此菜单项会变为 Signal Unpinning，就是解除绑定。对于未绑定信号的引脚，软件在自动分配引脚时可能会重新为此信号分配引脚。

(3) Pin Stacking/Pin Unstacking(引脚叠加/引脚解除叠加)：这个菜单项的功能不明确，手册里没有任何说明，ST 官网上也没有明确解答。不要单击此菜单项，否则影响生成的 C 语言代码。

4. 系统视图

如图 3-14 所示，芯片图片的上方有两个按钮：Pinout view(引脚视图)和 System view(系统视图)，单击这两个按钮可以在引脚视图和系统视图之间切换显示。图 3-18 所示为系统视图界面，界面上显示了 MCU 已经设置的各种组件，便于对 MCU 已经设置的系统资源和外设有一个总体的了解。

在系统视图界面单击某个组件时，在工作区的组件列表中就会显示此组件，在模式与配置界面中就会显示此组件的设置内容，以便进行查看和修改。

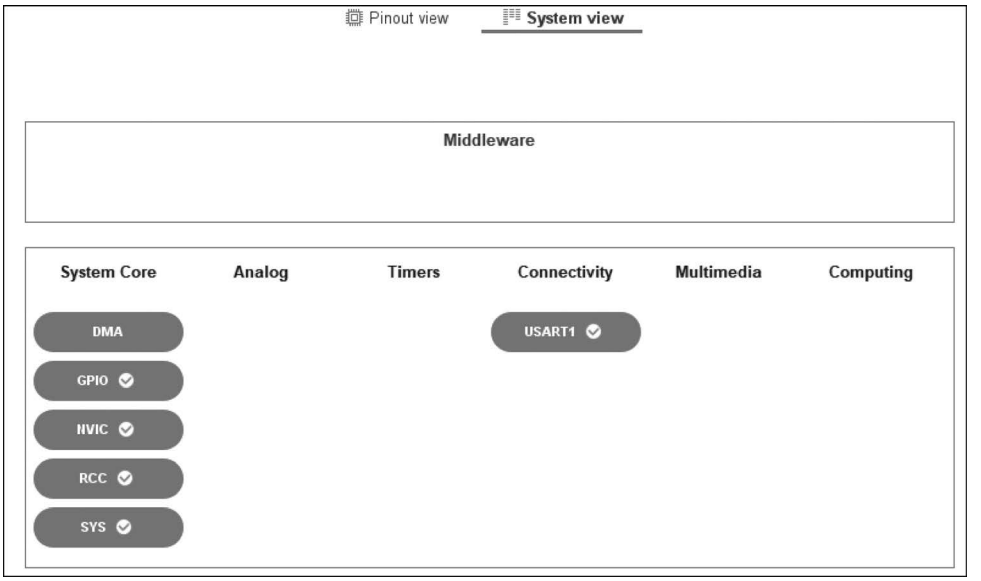

图 3-18　系统视图界面

5. Pinout 菜单

在引脚视图的上方还有一个工具栏，有两个按钮：Additional Software 和 Pinout。单击 Additional Software 按钮会弹出一个对话框，用于选择已安装的 STM32Cube 扩展包，添加到组件面板的 Additional Software 组中。

单击 Pinout 按钮会出现一个下拉菜单，如图 3-19 所示。

各菜单项的功能描述如下。

（1）Undo Mode and pinout：撤销上一次的模式设置和引脚分配操作。

（2）Redo Mode and pinout：重做上一次的撤销操作。

（3）Keep Current Signals Placement（保持当前信号的配置）：如果勾选此项，将保持当前设置的各个信号的引脚配置，也就是在后续自动配置引脚时，前面配置的引脚不会再改动。这样有时会引起引脚配置困难，如果是在设计电路阶段，可以取消此选项，让软件自动分配各外设的引脚。

（4）Show User Label（显示用户标签）：如果勾选此项，将显示引脚的用户定义标签，否则显示其已设置的信号名称。

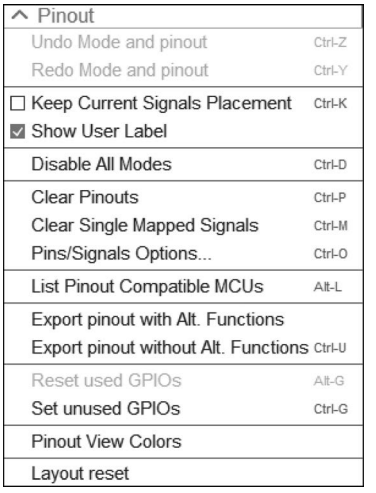

图 3-19　引脚视图上方的 Pinout 下拉菜单

(5) Disable All Modes(禁用所有模式)：取消所有外设和中间件的模式设置，复位全部相关引脚。但是，不会改变设置的普通 GPIO 输入或输出引脚。例如，不会复位用于 LED 的 GPIO 输出引脚。

(6) Clear Pinouts(清除引脚分配)：让所有引脚变成复位初始状态。

(7) Clear Single Mapped Signals(清除单边映射的信号)：清除那些定义了引脚的信号，但是没有关联外设的引脚，也就是橘黄色底色标识的引脚。必须先解除信号的绑定后才可以清除，也就是去除引脚上的图钉图标。

(8) Pins/Signals Options(引脚/信号选项)：会弹出一个如图 3-20 所示的 Pins/Signals Options 对话框，显示 MCU 已经设置的所有引脚名称、关联的信号名称和用户定义标签。可以按住 Shift 键或 Ctrl 键选择多行，然后右击弹出快捷菜单，通过菜单项进行引脚与信号的批量绑定或解除绑定。

(9) List Pinout Compatible MCUs(列出引脚分配兼容的 MCU)：会弹出一个对话框，显示与当前项目的引脚配置兼容的 MCU 列表。此功能可用于电路设计阶段选择与电路兼容的不同型号的 MCU。例如，可以选择一个与电路完全兼容但 Flash 更大或主频更高的 MCU。

(10) Export pinout with Alt. Functions：将具有复用功能的引脚的定义导出为一个 .csv 文件。

(11) Export pinout without Alt. Functions：将没有复用功能的引脚的定义导出为一个 .csv 文件。

(12) Set unused GPIOs(设置未使用的 GPIO 引脚)：弹出如图 3-21 所示的 Set unused GPIOs 对话框，对 MCU 未使用的 GPIO 引脚进行设置，可设置为 Input、Output 或 Analog 模式。一般设置为 Analog，以降低功耗。注意，要进行此项设置，必须在 SYS 组件中设置了调试引脚，如设置为 5 线 JTAG。

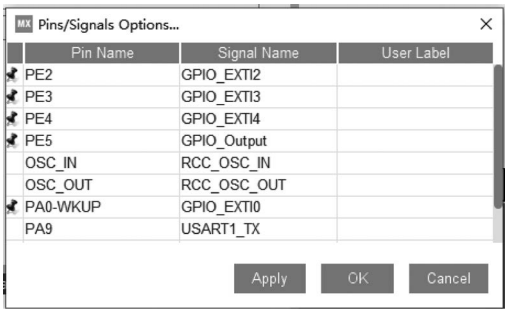

图 3-20 Pins/Signals Options 对话框

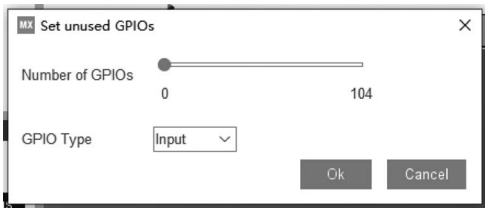

图 3-21 Set unused GPIOs 对话框

(13) Reset used GPIOs(复位已用的 GPIO 引脚)：弹出一个对话框，复位那些通过 Set unused GPIOs 对话框设置的 GPIO 引脚，可以选择复位的引脚个数。

(14) Layout reset(布局复位)：将 Pinout&Configuration 界面的布局恢复为默认状态。

3.3.5 时钟配置

MCU 图形化设置的第 2 个工作界面是时钟配置界面。为了充分演示时钟配置的功能，我们先设置 RCC 的模式，将 HSE 设置为 Crystal/Ceramic Resonator，并且启用 Master Clock Output，如图 3-22 所示。

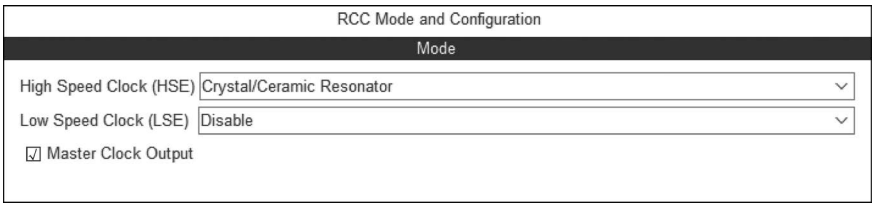

图 3-22 RCC 模式设置

MCO(Master Clock Output)是 MCU 向外部提供时钟信号的引脚，其中 MCO2 与音频时钟输入(Audio Clock Input，I2S_CKIN)共用 PC9 引脚，所以使用 MCO2 之后就不能再使用 I2S_CKIN 了。此外，我们需要启用 RTC，以便演示设置 RTC 的时钟源。

在 STM32CubeMX 的工作区单击 Clock Configuration 选项卡，它非常直观地显示了 MCU 的时钟树，使各种时钟信号的配置变得非常简单。

时钟源、时钟信号或选择器的作用如下。

(1) HSE(高速外部)时钟源。当设置 RCC 的 HSE 模式为 Crystal/Ceramic Resonator 时，用户可以设置外部振荡电路的晶振频率。例如，开发板上使用的是 8MHz 晶振，在其中输入 8 之后按 Enter 键，软件就会根据 HSE 的频率自动计算所有相关时钟频率并刷新显示。注意，HSE 的频率设置范围为 4～16MHz。

(2) HSI(高速内部)RC 振荡器。MCU 内部的高速 RC 振荡器，可产生频率为 8MHz 的时钟信号。

(3) PLL 时钟源选择器和主锁相环。锁相环(PLL)时钟源选择器可以选择 HSE 或 HSI 作为锁相环的时钟信号源，PLL 的作用是通过倍频和分频产生高频的时钟信号。在 Clock Configuration 选项卡中带有除号(/)的下拉列表框是分频器，用于将一个频率除以一个系数，产生分频的时钟信号；带有乘号(×)的下拉列表框是倍频器，用于将一个频率乘以一个系数，产生倍频的时钟信号。

主锁相环(Main PLL)输出两路时钟信号，一路是 PLLCLK，进入系统时钟选择器；另一路输出 48MHz 时钟信号，USB-OTG FS、USB-OTG HS、SDIO、RNG 都需要使用这个 48MHz 时钟信号。还有一个专用的锁相环 PLLI2S，用于产生精确时钟信号供 I2S 接口使用，以获得高品质的音效。

(4) 系统时钟选择器。系统时钟 SYSCLK 是直接或间接为 MCU 上的绝大部分组件提供时钟信号的时钟源，系统时钟选择器可以从 HSI、HSE、PLLCLK 这 3 个信号中选择一个作为 SYSCLK。

系统时钟选择器的下方有一个 Enable CSS 按钮，CSS(Clock Security System)是时钟安全系统，只有直接或间接使用 HSE 作为 SYSCLK 时，此按钮才有效。如果开启了 CSS，MCU 内部会对 HSE 时钟信号进行监测，当 HSE 时钟信号出现故障时，会发出一个 CSSI(Clock Security System Interrupt)信号，并自动切换到使用 HSI 作为系统时钟源。

(5) 系统时钟 SYSCLK。STM32F407 的 SYSCLK 最高频率是 168MHz，但是在 Clock Configuration 选项卡中的 SYSCLK 文本框中不能直接修改 SYSCLK 的值。可以看出，SYSCLK 直接作为以太网精确时间协议(Precision Time Protocol，PTP)的时钟信号，经过 AHB Prescaler(AHB 预分频器)后生成 HCLK 时钟信号。

(6) HCLK 时钟。SYSCLK 经过 AHB 分频器后生成 HCLK 时钟，HCLK 就是 CPU 的时钟信号，CPU 的频率就由 HCLK 的频率决定。HCLK 还为 APB1 总线和 APB2 总线等提供时钟信号。HCLK 最高频率为 72MHz。用户可以在 HCLK 文本框中直接输入需要设置的 HCLK 频率，按 Enter 键后软件将自动配置计算。

可以看到，HCLK 为其右侧的多个部分直接或间接提供时钟信号。

① HCLK to AHB bus，core，memory and DMA：HCLK 直接为 AHB 总线、内核、存储器和 DMA 提供时钟信号。

② To Cortex System timer：HCLK 经过一个分频器后作为 Cortex 系统定时器(也就是 Systick 定时器)的时钟信号。

③ FCLK Cortex clock：直接作为 Cortex 的 FCLK(Free-Running Clock)时钟信号。

④ APB1 peripheral clocks：HCLK 经过 APB1 分频器后生成外设时钟信号 PCLK1，为外设总线 APB1 上的外设提供时钟信号。

⑤ APB1 Timer clocks：PCLK1 经过 2 倍频后生成 APB1 定时器时钟信号，为 APB1 总线上的定时器提供时钟信号。

⑥ APB2 peripheral clocks：HCLK 经过 APB2 分频器后生成外设时钟信号 PCLK2，为外设总线 APB2 上的外设提供时钟信号。

⑦ APB2 timer clocks：PCLK2 经过 2 倍频后生成 APB2 定时器时钟信号，为 APB2 总线上的定时器提供时钟信号。

(7) 音频时钟输入。如果在 RCC 模式设置中勾选了 Audio Clock Input(I2S_CKIN)复选框，就可以在此输入一个外部的时钟源，作为 I2S 接口的时钟信号。

(8) MCO 时钟输出和选择器。MCO 是 MCU 为外部设备提供的时钟源，当勾选 Master Clock Output 复选框后，就可以在相应引脚输出时钟信号。

Clock Configuration 选项卡显示了 MCO2 的时钟源选择器和输出分频器，MCO1 的选择器和输出通道也与此类似，由于幅面限制没有显示出来。MCO2 的输出可以从 4 个时钟信号源中选择，还可以再分频后输出。

(9) LSE(低速外部)时钟源。如果在 RCC 模式设置中启用 LSE，就可以选择 LSE 作为 RTC 的时钟源。LSE 固定为 32.768kHz，因为经过多次分频后，可以得到精确的 1Hz 信号。

(10) LSI(低速内部)RC 振荡器。MCU 内部的 LSI RC 振荡器产生频率为 32kHz 的时钟信号,它可以作为 RTC 的时钟信号,也直接作为 IWDG(独立看门狗)的时钟信号。

(11) RTC 时钟选择器。如果启用 RTC,就可以通过 RTC 时钟选择器为 RTC 设置一个时钟源。RTC 时钟选择器有 3 个可选的时钟源:LSI、LSE 和 HSE 经分频后的时钟信号 HSE_RTC。要使 RTC 精确度高,应该使用 32.768kHz 的 LSE 作为时钟源,因为 LSE 经过多次分频后可以产生 1Hz 的精确时钟信号。

弄清楚 Clock Configuration 选项卡中的这些时钟源和时钟信号的作用后,进行 MCU 的各种时钟信号的配置就很简单了,因为都是图形化界面的操作,不用像传统编程那样必须弄清楚相关寄存器并计算寄存器的值了,这些底层的寄存器设置将由 STM32CubeMX 自动完成,并生成代码。

在 Clock Configuration 选项卡中可以进行以下操作。

(1) 直接在某个时钟信号的文本框中输入数值,按 Enter 键后由软件自动配置各个选择器、分频器、倍频器。例如,如果希望设置 HCLK 为 50MHz,在 HCLK 的文本框中输入 50 后按 Enter 键即可。

(2) 可以手动修改选择器、分频器、倍频器的设置,以便手动调节某个时钟信号的频率。

(3) 当某个时钟的频率设置错误时,其所在的文本框会以紫色底纹显示。

(4) 在某个时钟信号文本框上右击,会弹出一个快捷菜单,其中包含 Lock 和 Unlock 两个菜单项,用于对时钟频率进行锁定和解锁。如果一个时钟频率被锁定,其文本框会以灰色底纹显示。在软件自动计算频率时,系统尽量不会改变已锁定时钟信号的频率,如果必须改动,会弹出一个对话框提示解锁。

(5) 单击工具栏上的 Reset Clock Configuration 按钮,会将整棵时钟树复位到初始默认状态。

(6) 工具栏上的其他一些按钮可以进行撤销、重复、缩放等操作。

用户所做的这些时钟配置都涉及寄存器的底层操作,STM32CubeMX 在生成代码时会自动生成时钟初始化配置的程序。

3.3.6 项目管理

1. 功能概述

对 MCU 系统功能和各种外设的图形化配置,主要是在引脚配置和时钟配置两个工作界面完成的,完成这些工作后,一个 MCU 的配置就完成了。STM32CubeMX 的重要作用就是将这些图形化的配置结果导出为 C 语言代码。

STM32CubeMX 工作区的第 3 个工作界面是 Project Manager(项目管理器),如图 3-23 所示。这是一个多页界面,有以下 3 个工作界面。

(1) Project 界面:用于设置项目名称、保存路径、导出代码的 IDE 软件等。

(2) Code Generator 界面:用于设置生成 C 语言代码的一些选项。

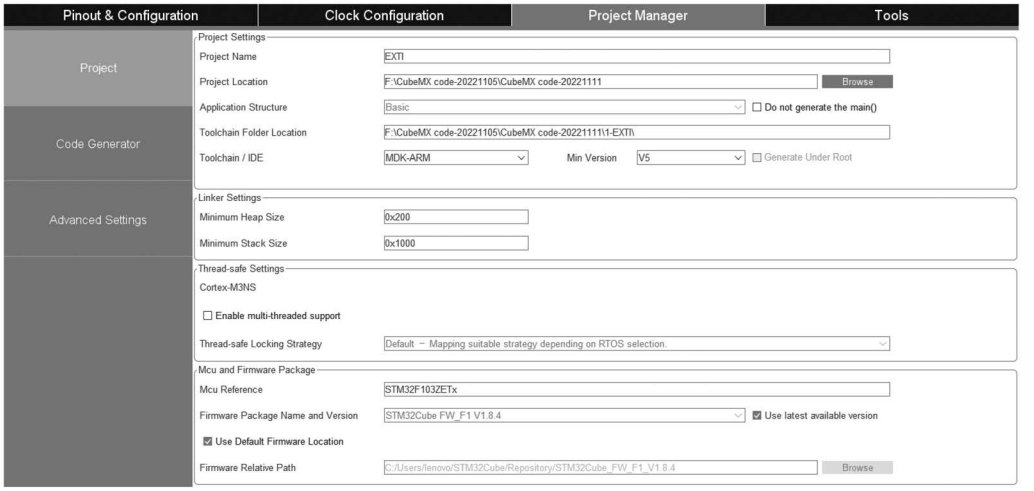

图 3-23　项目管理器

（3）Advanced Settings 界面：生成 C 语言代码的一些高级设置，如外设初始化代码是使用 HAL 库还是 LL 库。

2. 项目基本信息设置

新建的 STM32CubeMX 项目首次保存时会弹出一个选择文件夹的对话框，用户选择一个文件夹后，项目会被保存到文件夹下，并且项目名称与最后一级文件夹的名称相同。

例如，保存项目时选择的文件夹是 D:/Demo/MDK/1-LED/，那么项目会被保存到此目录下，并且项目文件名为 LED.ioc。

对于保存过的项目，就不能再修改图 3-23 中的 Project Name 和 Project Location 两个文本框中的内容了。项目管理器中还有以下一些设置项。

（1）Application Structure（应用程序结构），有 Basic 和 Advanced 两个选项。

① Basic：建议用于只使用一个中间件或者不使用中间件的项目。在这种结构中，IDE 配置文件夹与源代码文件夹同级，用子目录组织代码。

② Advanced：当项目里使用多个中间件时，建议使用这种结构，这样对于中间件的管理容易一些。

（2）Do not generate the main()复选框：如果勾选此项，导出的代码将不生成 main()函数。但是，C 语言的程序肯定是需要一个 main()函数的，所以不勾选此项。

（3）Toolchain Folder Location，也就是导出的 IDE 项目所在的文件夹，默认与 STM32CubeMX 项目文件在同一个文件夹。

（4）Toolchain/IDE：从一个下拉列表框中选择导出 C 语言程序的工具链或 IDE 软件，如图 3-24 所示。

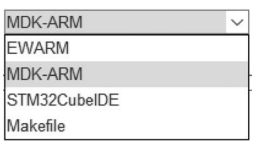

图 3-24　可选的工具链/IDE 软件列表

本书使用的 IDE 软件是 Keil MDK，Toolchain/IDE 选择 MDK-ARM。

(5) Linker Settings(链接器设置)：用于设置应用程序的堆(Heap)的最小大小，默认值是 0x200 和 0x400。

(6) Mcu and Firmware Package(MCU 和固件包)：MCU 固件库默认使用已安装的最新固件库版本。如果系统中有一个 MCU 系列多个版本的固件库，就可以在此重选固件库。如果勾选 Use Default Firmware Location 复选框，则表示使用默认的固件库路径，也就是所设置的软件库目录下的相应固件库目录。

3. 代码生成器设置

Code Generator 界面如图 3-25 所示，用于设置生成代码时的一些特性。

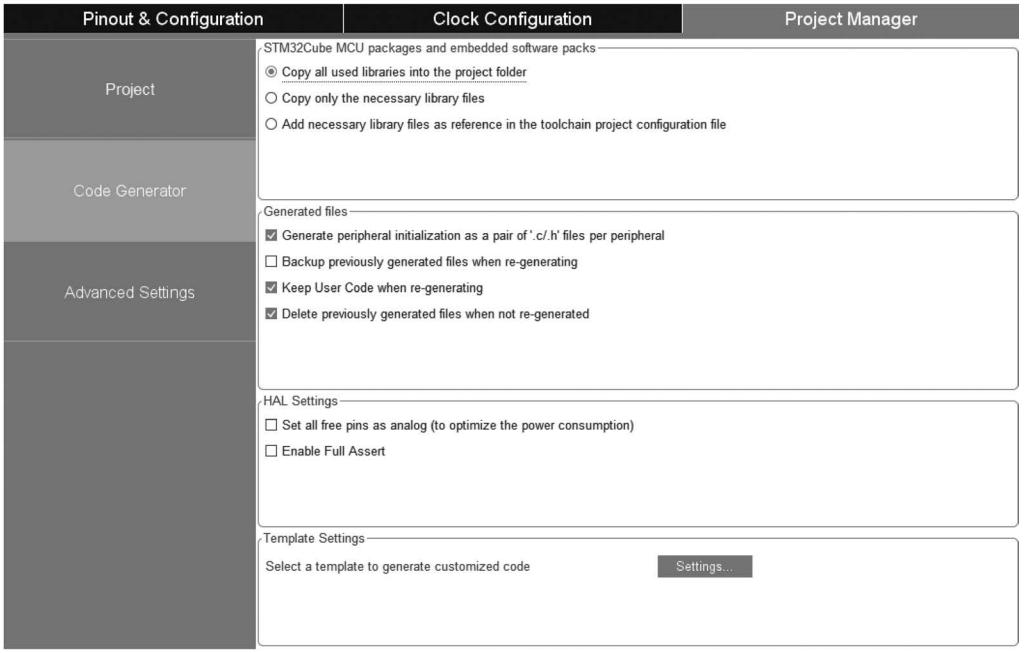

图 3-25 Code Generator 界面

(1) STM32Cube MCU packages and embedded software packs 选项：用于设置固件库和嵌入式软件库复制到 IDE 项目里的方式，有以下 3 种方式。

① Copy all used libraries into the project folder：将所有用到的库都复制到项目文件夹下。

② Copy only the necessary library files：只复制必要的库文件，即只复制与用户配置相关的库文件，默认选择这一项。

③ Add necessary library files as reference in the toolchain project configuration file：将必要的库文件以引用的方式添加到项目的配置文件中。

(2) Generated files 选项：生成 C 语言代码文件的一些选项。

① Generate peripheral initialization as a pair of'. c/. h' files per peripheral：勾选此项

后，为每种外设生成的初始化代码将会有.c和.h两个文件。例如，对于GPIO引脚的初始化程序，将有gpio.h和gpio.c两个文件，否则所有外设初始化代码存放在main.c文件里。虽然默认不勾选此项，但推荐勾选，特别是当项目用到的外设比较多时，而且使用.c和.h文件对更方便，也是更好的编程习惯。

② Backup previously generated files when re-generating：如果勾选此项，STM32CubeMX在重新生成代码时，就会将前面生成的文件备份到一个名为Backup的子文件夹里，并在.c和.h文件名后面再添加一个.bak扩展名。

③ Keep User Code when re-generating：重新生成代码时保留用户代码。这个选项只应用于STM32CubeMX自动生成文件中代码沙箱段(在后面会具体介绍此概念)的代码，不会影响用户自己创建的文件。

④ Delete previously generated files when not re-generated：删除那些以前生成的不需要再重新生成的文件。例如，前一次配置中用到了SDIO，生成的代码中有sdio.h和sdio.c文件，而重新配置时取消了SDIO，如果勾选了此项，重新生成代码时就会删除前面生成的sdio.h和sdio.c文件。

(3) HAL Settings选项：用于设置HAL。

① Set all free pins as analog (to optimize power consumption)：设置所有自由引脚的类型为Analog，这样可以优化功耗。

② Enable Full Assert：启用或禁用Full Assert功能。在生成的stm32f4xx_hal_conf.h文件中有一个宏定义USE_FULL_ASSERT，如果禁用Full Assert功能，这行宏定义代码就会被注释掉：

```
#define  USE_FULL_ASSERT  1U
```

如果启用Full Assert功能，那么HAL库中每个函数都会对函数的输入参数进行检查，如果检查出错，会返回出错代码的文件名和所在行。

(4) Template Settings选项：用于设置自定义代码模板。一般不用此功能，直接使用STM32CubeMX自己的代码模板就很好。

4. 高级设置

Advanced Settings界面如图3-26所示，分为上、下两个列表。

(1) Driver Selector列表：用于选择每个组件的驱动库类型。该列表列出了所有已配置的组件，如USART、RCC等，第2列是组件驱动库类型，有HAL和LL两种库可选。

HAL是高级别的驱动程序，MCU上所有组件都有HAL驱动程序。HAL的代码与具体硬件的关联度低，易于在不同系列的器件之间移植。

LL是进行寄存器级别操作的驱动程序，它的性能更加优化，但是需要对MCU的底层和外设比较熟悉，与具体硬件的关联度高，在不同系列之间进行移植时工作量大。并不是MCU上所有组件都有LL驱动程序，软件复杂度高的外设没有LL驱动程序，如SDIO、USB-OTG等。

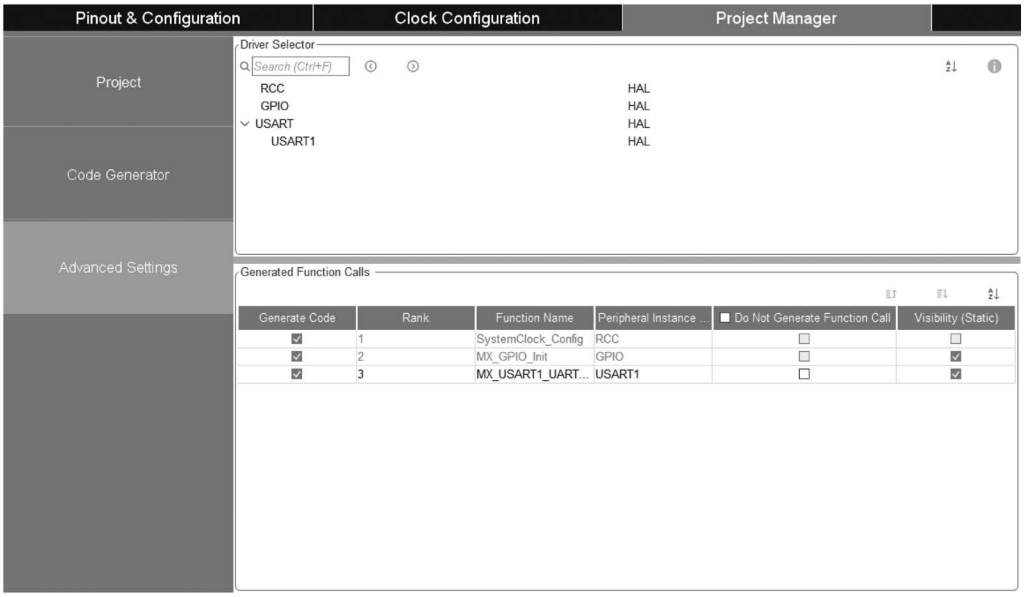

图 3-26　Advanced Settings 界面

本书完全使用 HAL 库进行示例程序设计，不会混合使用 LL 库，以保持总体统一。

(2) Generated Function Calls 列表：对生成函数的调用方法进行设置。图 3-26 下方的表格列出了 MCU 配置的系统功能和外设的初始化函数，列表中的各列如下。

① Function Name 列是生成代码时将要生成的函数名称，这些函数名称是自动确定的，不能修改。

② Do Not Generate Function Call 列：如果勾选了此项，在 main()函数的外设初始化部分不会调用这个函数，但是函数的完整代码还是会生成的，如何调用由编程者自己处理。

③ Visibility(Static)列用于指定是否在函数原型前面加上 static 关键字，使函数变为文件内的私有函数。如果在图 3-25 中勾选了 Generate peripheral initialization as a pair of '.c/.h'files per peripheral 复选框，则无论是否勾选 Visibility(Static)复选框，外设的初始化函数原型前面都不会加 static 关键字，因为在.h 文件中声明的函数原型对外界就是可见的。

3.3.7　生成报告和代码

在对 MCU 进行各种配置以及对项目进行设置后，用户就可以生成报告和代码。

执行 File→Generate Report 菜单命令，会在 STM32CubeMX 项目文件目录下生成一个同名的 PDF 文件。这个 PDF 文件里有对项目的基本描述、MCU 型号描述、引脚配置图、引脚定义表格、时钟树、各种外设的配置信息等，是对 STM32CubeMX 项目的一个很好的总结性报告。

保存 STM32CubeMX 项目并在项目管理界面做好生成代码的设置后，用户随时可以

单击导航栏右端的 GENERATE CODE 按钮，为选定的 MDK-Arm 软件生成代码。如果是首次生成代码，将自动生成 MDK-Arm 项目框架，生成项目所需的所有文件；如果 MDK-Arm 项目已经存在，再次生成代码时只会重新生成初始化代码，不会覆盖用户在沙箱段内编写的代码，也不会删除用户在项目中创建的程序文件。

STM32CubeMX 软件的工作区还有一个 Tools 选项卡，用于进行 MCU 的功耗计算，这会涉及 MCU 的低功耗模式。

3.4 HAL 库

最近兴起的 HAL 库就是 ST 公司目前主推的研发方式，其更新速度比较快，可以通过官方推出的 STM32CubeMX 工具一键生成代码，大大缩短开发周期。使用 HAL 库的优势主要就是不需要开发工程师再设计所用的 MCU 型号，只需要专注于功能的软件开发工作即可。

3.4.1 HAL 库简介

HAL 是 Hardware Abstraction Layer 的缩写，中文名称是硬件抽象层。HAL 库是 ST 公司 STM32 的 MCU 最新推出的抽象层嵌入式软件，为更方便地实现跨 STM32 产品的最大可移植性。和标准外设库(也称标准库)对比起来，STM32 的 HAL 库更加抽象，ST 公司最终的目的是要实现在 STM32 系列 MCU 之间无缝移植，甚至在其他 MCU 也能实现快速移植。

ST 公司为开发者提供了非常方便的开发库，有 SPL 库(标准外设库)、HAL 库、LL 库(Low-Layer，底层库)3 种。前者是 ST 的老库，已经停止更新了，后两者是 ST 现在主推的开发库。

相比于标准外设库，STM32Cube HAL 库表现出更高的抽象整合水平，HAL API 集中关注各外设的公共函数功能，这样便于定义一套通用的用户友好的 API，从而可以轻松实现从一个 STM32 产品移植到另一个不同的 STM32 系列产品。HAL 库是 ST 公司未来主推的库，ST 公司新出的芯片已经没有标准外设库了，如 F7 系列。目前，HAL 库已经支持 STM32 全线产品。

HAL 库的特点如下。

(1) 最大可移植性。

(2) 提供了一整套一致的中间件组件，如 RTOS、USB、TCP/IP 和图形等。

(3) 通用的用户友好的 API。

(4) ST 公司新出的芯片已经没有标准库。

(5) HAL 库已经支持 STM32 全线产品。

通常新手在入门 STM32 时，首先都要先选择一种开发方式，不同的开发方式会导致编程的架构完全不同。一般都会选用标准外设库和 HAL 库，而极少部分人会通过直接配置寄存器进行开发。

1. 直接配置寄存器

不少先学习了 MCS-51 单片机的读者可能会知道，会有一小部分人是通过汇编语言直接操作寄存器实现功能的，这种方法到了 STM32 就变得不太容易行得通了，因为 STM32 的寄存器数量是 MCS-51 单片机的数十倍，如此多的寄存器根本无法全部记忆，开发时需要经常翻查芯片的数据手册，此时直接操作寄存器就变得非常费力了。但还是会有很小一部分人喜欢直接操作寄存器，因为这样更接近原理，知其然也知其所以然。

2. 标准外设库

STM32 有非常多的寄存器，而导致了开发困难，为此 ST 公司就为每款芯片都编写了一份库文件。在这些 .c 和.h 文件中，包括一些常用量的宏定义，把一些外设也通过结构体变量封装起来，如 GPIO 等。所以，只需要配置结构体变量成员就可以修改外设的配置寄存器，从而选择不同的功能。这也是目前最多人使用的方式，也是学习 STM32 接触最多的一种开发方式，这里就不再阐述了。

3. HAL 库

HAL 库是 ST 公司目前主推的开发方式，库如其名，很抽象，一眼看上去不太容易知道它的作用是什么。它的出现比标准外设库要晚，但其实和标准外设库一样，都是为了节省程序开发的时间，而且 HAL 库尤其有效，如果说标准外设库集成了实现功能需要配置的寄存器，那么 HAL 库的一些函数甚至可以做到某些特定功能的集成。也就是说，同样的功能，标准外设库可能要用几句话，HAL 库只需一句话就够了。并且，HAL 库也很好地解决了程序移植的问题，不同型号的 STM32 芯片的标准外设库是不一样的。例如，在 STM32F4 上开发的程序移植到 STM32F1 上是不通用的，而使用 HAL 库，只要使用的是相同的外设，程序基本可以完全复制粘贴。注意是相同外设，也就是不能无中生有，如 STM32F7 比 STM32F1 要多几个定时器，不能明明没有这个定时器却非要配置，但其实这种情况不多，一般都可以直接复制粘贴。使用 ST 公司研发的 STMCube 软件，可以通过图形化的配置功能，直接生成整个使用 HAL 库的工程文件。

4. HAL 库与标准外设库的区别

STM32 的开发中，可以这样操作寄存器：

```
GPIOF->BSRR = 0x00000001;                    //这里是针对 STM32F1 系列
```

这种方法当然可以，但是需要掌握每个寄存器的用法，才能正确使用 STM32，而对于 STM32 这种级别的 MCU，记忆数百个寄存器又谈何容易。于是 ST 公司推出了官方标准外设库，标准外设库将这些寄存器底层操作都封装起来，提供一整套 API 供开发者调用，大多数场合下，不需要知道操作的是哪个寄存器，只需要知道调用哪些函数即可。

例如，控制 BRR 寄存器实现电平控制，官方库封装了一个函数，如下。

```
void GPIO_ResetBits(GPIO_TypeDef * GPIOx,uint_t GPIO_Pin)
{
    GPIOx->BRR = GPIO_Pin
}
```

这时不需要再直接操作 BRR 寄存器了，只需要知道怎么调用 GPIO_ResetBits()函数就可以了。在对外设的工作原理有一定的了解之后，再去看标准外设库函数，基本上由函数名称就能得知这个函数的功能是什么、该怎么使用，这样开发就方便很多。

标准外设库自推出以来受到广大工程师推崇，现在很多工程师和公司还在使用标准外设库函数开发。不过，ST 官方已经不再更新 STM32 标准外设库，而是力推新的 HAL 库。

例如，控制 BSRR 寄存器实现电平控制，官方 HAL 库封装了一个函数，如下。

```
void HAL_GPIO_WritePin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
{
      /* Check the parameters */
      assert_param(IS_GPIO_PIN(GPIO_Pin));
      assert_param(IS_GPIO_PIN_ACTION(PinState));
      if (PinState != GPIO_PIN_RESET)
      {
          GPIOx->BSRR = GPIO_Pin;
      }
      else
      {
          GPIOx->BSRR = (uint32_t)GPIO_Pin << 16u;
      }
}
```

这时不需要再直接操作 BSRR 寄存器了，只需要知道怎么使用 HAL_GPIO_WritePin()这个函数就可以了。

HAL 库和标准外设库一样，都是固件库，是由 ST 官方硬件抽象层而设计的软件函数包，由程序、数据结构和宏组成，包括了 STM32 所有外设的性能特征。这些固件库为开发者底层硬件提供了中间 API，通过使用固件库，无须掌握底层细节，开发者就可以轻松应用每个外设。

HAL 库和标准外设库本质上是一样的，都是提供底层硬件操作 API，而且在使用上也是大同小异。有标准外设库基础的读者对 HAL 库的使用也很容易入手。ST 官方之所以这几年大力推广 HAL 库，是因为 HAL 的结构更加容易整合 STM32Cube，而 STM32CubeMX 是 ST 公司这几年极力推荐的程序生成开发工具。所以，这几年新出的 STM32 芯片，ST 公司直接只提供 HAL 库。

在 ST 公司的官方声明中，HAL 库是大势所趋。标准外设库和 HAL 库虽然都是对外设进行操作的函数库，但由于官方已经停止更新标准外设库，而且标准外设库在 STM32 创建工程和初始化时不能由 STM32CubeMX 软件代码生成使用，也就是说 STM32CubeMX 软件在生成代码时，工程项目和初始化代码就自动生成，这个工程项目和初始化代码里面使用的函数都是基于 HAL 库的。STM32CubeMX 是一个图形化的工具，也是配置和初始化 C 代码生成器，与 STM32CubeMX 配合使用的是 HAL 库。

1）外设句柄定义

用户代码的第一大部分是对于外设句柄的处理。在结构上，HAL 库将每个外设抽象

成一个名为 ppp_HandleTypeDef 的结构体，其中 ppp 就是每个外设的名字。所有函数都工作在 ppp_HandleTypeDef 指针之下。

（1）多实例支持：每个外设/模块实例都有自己的句柄，因此实例资源是独立的。

（2）外围进程相互通信：该句柄用于管理进程例程之间的共享数据资源。

2）三种编程方式

HAL 库对所有函数模型也进行了统一。在 HAL 库中，支持 3 种编程模式：轮询模式、中断模式、DMA 模式（如果外设支持），分别对应以下 3 种类型的函数（以 ADC 为例）。

```
HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef * hadc);
HAL_StatusTypeDef HAL_ADC_Stop(ADC_HandleTypeDef * hadc);
HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef * hadc);
HAL_StatusTypeDef HAL_ADC_Stop_IT(ADC_HandleTypeDef * hadc);
HAL_StatusTypeDef HAL_ADC_Start_DMA(ADC_HandleTypeDef * hadc, uint32_t * pData, uint32_t
Length);
HAL_StatusTypeDef HAL_ADC_Stop_DMA(ADC_HandleTypeDef * hadc);
```

其中，函数名中带_IT 的表示工作在中断模式下；带_DMA 的工作在 DMA 模式下（注意 DMA 模式下也是开启中断的）；其他就是轮询模式（没有开启中断的）。至于使用何种方式，就看自己的选择了。此外，新的 HAL 库架构下统一采用宏的形式对各种中断等进行配置（原来标准外设库一般都是各种函数）。针对每种外设，主要有以下宏。

```
__HAL_PPP_ENABLE_IT(__HANDLE__, __INTERRUPT__)          //使能一个指定的外设中断
__HAL_PPP_DISABLE_IT(__HANDLE__, __INTERRUPT__)         //失能一个指定的外设中断
__HAL_PPP_GET_IT (__HANDLE__, __INTERRUPT__)            //获得一个指定的外设中断状态
__HAL_PPP_CLEAR_IT (__HANDLE__, __INTERRUPT__)          //清除一个指定的外设的中断状态
__HAL_PPP_GET_FLAG (__HANDLE__, __FLAG__)               //获取一个指定的外设的标志状态
__HAL_PPP_CLEAR_FLAG (__HANDLE__, __FLAG__)             //清除一个指定的外设的标志状态
__HAL_PPP_ENABLE(__HANDLE__)                            //使能外设
__HAL_PPP_DISABLE(__HANDLE__)                           //失能外设
__HAL_PPP_XXXX (__HANDLE__, __PARAM__)                  //指定外设的宏定义
__HAL_PPP_GET IT_SOURCE (__HANDLE__, __INTERRUPT__)     //检查中断源
```

3）三大回调函数

在 HAL 库的源代码中，到处可见一些以__weak 开头的函数，而且这些函数有些已经被实现了，如

```
__weak HAL_Status TypeDef HAL_InitTick(uint32_t TickPriority)
{
    /* Configure the SysTick to have interrupt in 1ms time basis */
    HAL_SYSTICK_Config(SystemCoreClock/1000U);
    /* Configure the SysTick IRQ priority */
    HAL_NVIC_SetPriority(SysTick_IRQn, TickPriority ,OU);
    /* Return function status */
    return HAL_OK;
}
```

有些则没有被实现，如

```
__weak void HAL_SPI_TxCpltCallback(SPI_Handle TypeDef * hspi)
{
    /* Prevent unused argument(s) compilation warning */
    UNUSED(hspi);
    /* NOTE:This function should not be modified, when the callback is
    needed,the HAL_SPI_TxCpltCallback should be implemented in the
    user file */
}
```

在 HAL 库中，很多回调函数前面使用__weak 修饰符，这里有必要讲解__weak 修饰符的作用。

weak 顾名思义是“弱”的意思，所以如果函数名称前面加上__weak 修饰符，一般称这个函数为弱函数。加上了__weak 修饰符的函数，用户可以在用户文件中重新定义一个同名函数，最终编译器编译时，会选择用户定义的函数，如果用户没有重新定义这个函数，那么编译器就会执行__weak 声明的函数，并且编译器不会报错。

所有带有__weak 关键字的函数表示，都可以由用户自己实现。如果出现了同名函数，且不带__weak 关键字，那么连接器就会采用外部实现的同名函数。通常来说，HAL 库负责整体处理和 MCU 外设的处理逻辑，并将必要部分以回调函数的形式提供给用户，用户只需要在对应的回调函数中修改即可。

HAL 库包含以下 3 种用户级别回调函数(其中 PPP 为外设名)。

(1) 外设系统级初始化/解除初始化回调函数：HAL_PPP_MspInit()和 HAL_PPP_MspDeInit()。

例如，__weak void HAL_SPI_MspInit(SPI_HandleTypeDef * hspi)，在 HAL_PPP_Init()函数中被调用，用来初始化底层相关的设备(GPIOs、Clock、DMA 和 Interrupt)

(2) 处理完成回调函数：HAL_PPP_ProcessCpltCallback()，其中 Process 指具体某种处理，如 UART 的 Tx。

例如，__weak void HAL_SPI_RxCpltCallback(SPI_HandleTypeDef * hspi)，当外设或 DMA 工作完成时触发中断，该回调函数会在外设中断处理函数或 DMA 的中断处理函数中被调用。

(3) 错误处理回调函数：HAL_PPP_ErrorCallback()。

例如，__weak void HAL_SPI_ErrorCallback(SPI_HandleTypeDef * hspi)，当外设或 DMA 出现错误时触发中断，该回调函数会在外设中断处理函数或 DMA 的中断处理函数中被调用。

绝大多数用户代码均在以上 3 种回调函数中实现。

HAL 库结构中，在每次初始化前(尤其是在多次调用初始化前)，先调用对应的解除初始化(DeInit)函数是非常有必要的。某些外设多次初始化时不调用返回会导致初始化失

败。完成回调函数有多种，如串口的完成回调函数有 HAL_UART_TxCpltCallback()和 HAL_UART_TxHalfCpltCallback()等。

5. HAL 库移植使用的基本步骤

HAL 库移植使用的基本步骤如下。

(1) 复制 stm32f4xx_hal_msp_template.c 文件，参照该模板，依次实现用到的外设的 HAL_PPP_MspInit()和 HAL_PPP_MspDeInit()函数。

(2) 复制 stm32f4xx_hal_conf_template.h 文件，用户可以在此文件中自由裁剪，配置 HAL 库。

(3) 在使用 HAL 库时，必须先调用 HAL_StatusTypeDef HAL_Init(void)函数。该函数在 stm32f4xx_hal.c 文件中定义，也就意味着必须首先实现 HAL_MspInit(void)和 HAL_MspDeInit(void)函数。

(4) HAL 库与标准外设库不同，HAL 库使用 RCC 中的函数配置系统时钟，用户需要单独写时钟配置函数(标准外设库默认在 system_stm32f4xx.c 文件中)。

(5) 关于中断，HAL 库提供了中断处理函数，只需要调用 HAL 库提供的中断处理函数。用户自己的代码，不建议先写到中断中，而应该写到 HAL 库提供的回调函数中。

(6) 对于每个外设，HAL 库都提供了回调函数，回调函数用来实现用户自己的代码。整个调用结构由 HAL 库自己完成。

例如，UART 中，HAL 库提供了 void HAL_UART_IRQHandler(UART_HandleTypeDef * huart)函数，触发中断后，用户只需要调用该函数即可；同时，自己的代码写在对应的回调函数中即可，如下所示。

```
void HAL_UART_TxCpltCallback(UART_HandleTypeDef * huart);
void HAL_UART_TxHalfCpltCallback(UART_HandleTypeDef * huart);
void HAL_UART_RxCpltCallback(UART_HandleTypeDef * huart);
void HAL_UART_RxHalfCpltCallback(UART_HandleTypeDef * huart);
void HAL_UART_ErrorCallback(UART_HandleTypeDef * huart);
```

综上所述，使用 HAL 库编写程序(针对某个外设)的基本结构(以串口为例)如下。

(1) 配置外设句柄。例如，建立 UartConfig.c 文件，在其中定义串口句柄 UART_HandleTypeDef huart，接着初始化句柄

```
HAL_StatusTypeDef HAL_UART_Init(UART_HandleTypeDef huart)
```

(2) 编写 MSP 函数。例如，建立 UartMsp.c 文件，在其中实现 void HAL_UART_MspInit(UART_HandleTypeDef huart)和 void HAL_UART_MspDeInit(UART_HandleTypeDef * huart)函数。

(3) 实现对应的回调函数。例如，建立 UartCallBack.c 文件，在其中实现三大回调函数中的完成回调函数和错误回调函数。

3.4.2 HAL 库与标准外设库和 LL 库的区别

ST 公司为开发者提供了非常方便的开发库。到目前为止，有标准外设库、HAL 库和 LL 库 3 种。其中，标准外设库与 HAL 库是最常用的库，LL 库只是最近新添加的库。

(1) 标准外设库(Standard Peripherals Library)是对 STM32 芯片的一个完整的封装。以前对芯片外设的操作都是直接操作寄存器，这使得开发非常困难。ST 公司为了解决这个难题，给每款芯片都编写了一份库文件，也就是标准外设库。这也是目前使用最多的 ST 库，几乎均使用 C 语言实现。但是，标准外设库是针对某一系列芯片而言的，没有可移植性。

(2) HAL 库是 ST 公司目前主推的开发方式。它的出现比标准外设库要晚，但其实和标准外设库一样，都是为了节省程序开发的时间，而且 HAL 库更有效。如果说标准外设库为了实现对芯片外设的操作功能，集成了需要配置的寄存器，那么 HAL 库的一些函数甚至做到了某些特定功能的集成。也就是说，同样的功能，标准库可能要用几句话，HAL 库只需用一句话就够了，并且 HAL 库也很好地解决了程序移植的问题。

(3) LL(Low Layer)库是 ST 公司最近新增的库，与 HAL 库捆绑发布，文档也是和 HAL 库文档在一起的。例如，在 STM32F3x 的 HAL 库说明文档中，ST 公司新增了 LL 库这一章节，但是在 STM32F2x 的 HAL 库文档中就没有。

LL 库更接近硬件层，对需要复杂上层协议栈的外设不适用。LL 库直接操作寄存器，支持所有外设。使用方法如下。

① 独立使用。该库完全独立实现，可以完全抛开 HAL 库，只用 LL 库编程完成。在使用 STM32CubeMX 生成项目时，直接选 LL 库即可。如果使用了复杂的外设，如 USB，则会调用 HAL 库。

② 混合使用。与 HAL 库结合使用。

3.4.3 回调函数

下面介绍回调函数的含义、使用，以及回调函数与普通函数的调用区别。

1. 什么是回调函数

回调函数就是一个被作为参数传递的函数。在 C 语言中，回调函数只能使用函数指针实现，在 C++、Python、ECMAScript 等更现代的编程语言中还可以使用仿函数或匿名函数。回调函数的使用可以大大提升编程的效率，这使得它在现代编程中应用广泛。同时，有一些需求必须要使用回调函数实现。

函数指针的调用，即是一个通过函数指针调用函数。如果把函数的指针(地址)作为参数传递给另一个函数，当这个指针被用来调用其所指向的函数时，就说它是回调函数。

也就是说，把一段可执行的代码像传递参数那样传给其他代码，而这段代码会在某个时刻被调用执行，就叫作回调。如果代码立即被执行，就称为同步回调；如果在之后晚些的某个时间再执行，则称为异步回调。

2. 为什么要使用回调函数

回调函数的作用是“解耦”，普通函数代替不了回调函数的这个作用，这是回调函数最大的特点。

```
#include<stdio.h>
#include<freeLib.h>
 // 回调函数
int Callback()
{
    func();
    return 0;
}

// 主程序
int main()
{
    Library(Callback);
    return 0;
}
```

3. 回调函数与普通函数的调用区别

（1）在主程序中，把回调函数像参数一样传入库函数。这样一来，只要改变传入库函数的参数，就可以实现不同的功能，且不需要修改库函数的实现，变得很灵活，这就是解耦。

（2）主函数和回调函数是在同一层的，而库函数在另外一层。如果库函数不可见，我们修改不了库函数的实现，也就是说，不能通过修改库函数让库函数调用普通函数，那么就只能通过传入不同的回调函数实现了，这也是在日常工作中常见的情况。

使用回调函数会有间接调用，因此会有一些额外的传参和访存开销，对于对时间要求较高的 MCU 代码要慎用。

回调函数的使用是对函数指针的应用，函数指针的概念本身很简单，但是把函数指针应用于回调函数就体现了一种解决问题的策略，一种设计系统的思想。

回调函数的缺点如下。

（1）回调函数固然能解决一部分系统架构问题，但是绝不能在系统内到处都是。如果用户发现系统内到处都是回调函数，那么一定要重构系统。

（2）回调函数本身是一种破坏系统结构的设计思路，回调函数会绝对地改变系统的运行轨迹、执行顺序和调用顺序。

3.4.4 MSP 的作用

MSP 全称为 MCU Support Package，即 MCU 支持包。名称中带有 MspInit 的函数，它们的作用是进行 MCU 级别硬件初始化设置，并且通常会被上一层的初始化函数所调用，这样做的目的是把 MCU 相关的硬件初始化剥夺出来，方便用户代码在不同型号的 MCU

上移植。stm32f4xx_hal_msp.c 文件定义了两个函数：HAL_MspInit()和 HAL_MspDeInit()。这两个函数分别被 stm32f4xx_hal.c 文件中的 HAL_Init()和 HAL_DeInit()函数调用。HAL_MspInit()函数的主要作用是进行 MCU 相关的硬件初始化操作。例如，要初始化某些硬件，可以将硬件相关的初始化配置写在 HAL_MspDeinit()函数中。这样的话，在系统启动调用了 HAL_Init()函数之后，会自动调用硬件初始化函数。

实际上，我们在工程模板中直接删掉 stm32f4xx_hal_msp.c 文件也不会对程序运行产生任何影响。

3.4.5 HAL 库的基本问题

下面介绍 HAL 库的基本数据类型和一些通用定义。

1. 基本数据类型

对 STM32 系列 MCU 编程使用的是 C 语言或 C++语言。C 语言整数类型的定义比较多，STM32 编程中一般使用简化的定义符号，如表 3-2 所示。

表 3-2 STM32 编程中的数据类型简化定义符号

数 据 类 型	C 语言等效定义	数据长度/B
int8_t	signed char	1
uint8_t	unsigned char	1
int16_t	signed short	2
uint16_t	unsigned short	2
int32_t	signed int	4
uint32_t	unsigned int	4
int64_t	long long int	8
uint64_t	unsigned long long int	8

2. 一些通用定义

在 HAL 库中，有一些类型或常量是经常用到的，具体如下。

(1) stm32f4xx_hal_def.h 文件中表示函数返回值类型的枚举类型 HAL_StatusTypeDef，定义如下。

```
typedef enum
{
      HAL_OK          = 0x000,
      HAL_ERROR       = 0x010,
      HAL_BUSY        = 0x02U,
      HAL_TIMEOUT     = 0x03U
}HAL_StatusTypeDef;
```

很多函数返回值的类型是 HAL_StatusTypeDef，以表示函数运行是否成功或处于其他状态。

（2）stm32f4xx.h 文件中几个通用的枚举类型和常量，定义如下。

```
typedef enum
{
      RESET = 0U,
      RESET = 0U
      SET = !RESET
}FlagStatus,ITStatus;                    //一般用于判断标志位是否置位
typedef enum
{
      DISABLE = 0U,
      ENABLE = !DISABLE
}FunctionalState;                        //一般用于设置某个逻辑型参数的值
typedef enum
{
      SUCCESS = 0U,
      ERROR = !SUCCESS
} ErrorStatus;                           //一般用于函数返回值，表示成功或失败两种状态
```