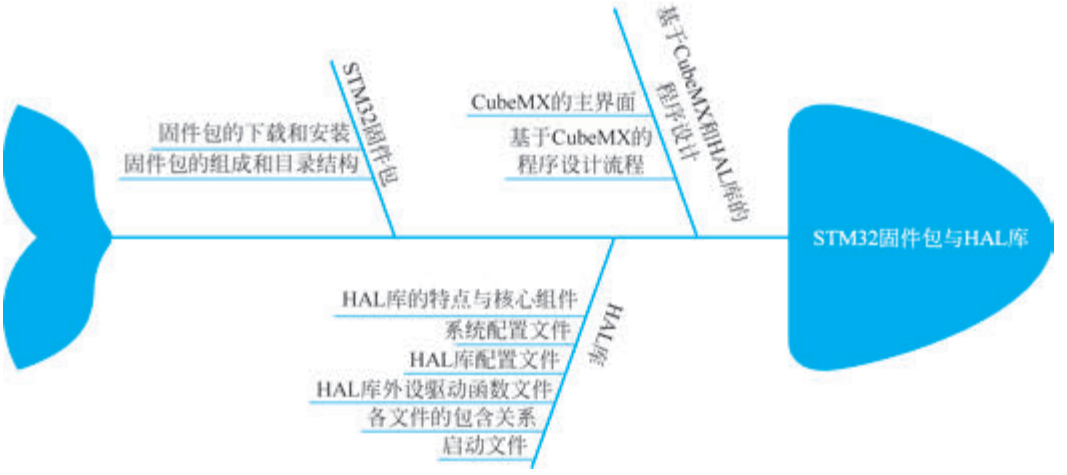
第 3 章

STM32 固件包与 HAL 库



第 3 章思维导图

在早期的 8 位微机系统程序开发过程中，一般采用基于寄存器的开发方式，通过直接访问芯片内部寄存器的方法来控制芯片功能。由于这类微机资源有限，这种程序开发方式的难度相对较低。对于 STM32 微控制器而言，由于其外设资源的丰富性，寄存器数量和复杂性显著增加，采用这种开发方式将导致程序可读性差、维护困难，从而严重影响开发效率，并增加程序维护与交流的成本。

采用库开发方式可以有效解决上述问题，特别是 ST 公司在推出各种型号 MCU 的同时，提供了功能十分强大的固件包、HAL 库和丰富的开发工具，极大地方便了相关应用系统的程序设计和调试。本章将简要介绍 STM32 的固件包和 HAL 库，以及基于 CubeIDE 和 HAL 库的程序设计。

3.1 STM32 固件包

STM32 固件包构建在一个通用且模块化的架构之上，即使开发者对所用 MCU 了解不深，也能方便地编写应用程序，并提高程序代码的复用性，方便代码在不同设备间的移植。

针对不同型号和系列的 STM32 MCU，固件包都有对应的版本。例如，STM32CubeF4 固件包专为基于 Cortex-M4 的 STM32F4 系列 MCU 设计，该固件包集成了在 STM32F4 系列微控制器上开发应用程序所需的所有通用嵌入式软件组件，形成一个全面的软件包。

固件包内包含了多种软件组件，例如 HAL 库、低级层库、中间件和标准外设库等。其中，HAL 库是开发者经常使用的组件之一，它提供了一套高级 API，使得开发者在不深入了解底层硬件细节的情况下，也能实现对硬件的访问和控制。HAL 库的设计目标是简化开发流程，提升开发效率，同时保持代码的可读性和可维护性。

3.1.1 固件包的下载和安装

这里以本书介绍的 STM32F407 系列 MCU 为例，介绍 STM32 固件包的下载和安装方法。

1. 固件包的下载

在 ST 公司官网主页搜索框中输入 CubeF4，在跳转的页面中单击查找到的 STM32CubeF4，再单击 Get latest，即可进入 CubeF4MCU 固件包下载页面，如图 3-1 所示。注意此时需要联网并用实际邮箱进行注册登录。

图 3-1 CubeF4MCU 固件包下载页面

目前 STM32F4 系列 MCU 固件包最新版本为 V1.28.0。图 3-1 中的第一行为补丁包，两个固件包下载后都是压缩文件，文件名分别为 en.stm32cubef4-v1-28-0.zip 和 en.stm32cubef4-v1-28-1.zip。

2. 固件包的安装

为了安装固件包，启动 CubeIDE 窗口后，单击 Help/Configuration Tool 菜单下的 Manage Embedded Software Packages 菜单命令，打开如图 3-2 所示的嵌入式软件包管理器。

在软件包管理器中单击并进入 STM32Cube MCU Packages 选项卡，其中将列出所有 STM32 系列 MCU 可用的固件包。在固件包列表中，每个版本前面有一个小方框，方框中如果有颜色填充，表示已经安装有相应的固件包，否则表示相应版本的固件包还没有安装。例如，图 3-2 中显示已经安装了 V1.28.0 版本，但还没有安装 V1.28.1 版本。

在固件包列表中单击固件包前面的小方框，再单击管理器下方的 From Local 按键，即可安装事先下载的固件包，也可以单击 Install 按键进行在线下载和安装。此外，在某行前面的小方框中单击后出现一个红色小叉，再单击 Remove 按键，可以将选中的固件包卸载和删除。

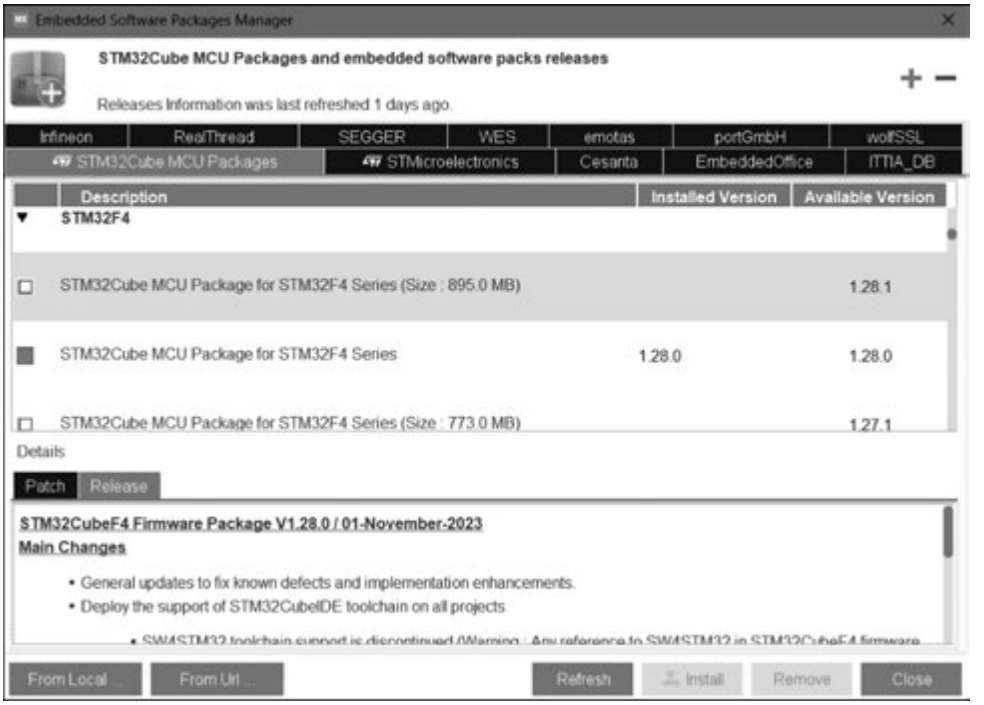


图 3-2　嵌入式软件包管理器

CubeIDE 的固件包默认安装在 C:\Users\用户名\STM32Cube\Repository 文件夹，通过 CubeIDE 主窗口中的 Window/Preferences 菜单命令，在打开的对话框中单击 STM32Cube/Firmware Updater 选项可以指定固件包安装到硬盘上的其他位置。安装成功后，在该文件夹中即可找到相应的固件包文件夹，例如 STM32Cube_FW_F4_V1.28.0 文件夹中存放的是 V1.28.0 版本的固件包。

3.1.2　固件包的组成和目录结构

STM32CubeF4 固件包包括 MCU 硬件的 LL 库和 HAL 库 API 函数，以及可以在 ST 公司开发板上运行的一系列示例。针对 STM32F4 系列 MCU 的 STM32CubeF4 固件包还包含一组中间件层（Middleware Level）和应用层（Application Level）。STM32CubeF4 固件包的组成如图 3-3 所示。

1. 固件包的组成

固件包采用分层结构，包括应用层、中间件层，以及 HAL 和 LL 的 API（最底层）。应用层是基于中间件服务层、底层抽象层和板级外设应用的示例层。中间件层包括各种中间件组件及其示例。中间件组件是一组库，其中典型的有 USB 主机和器件库、FatFS 文件系统、FreeRTOS 库等，这件组件之间通过调用相应的 API 函数实现交互，并通过特定回调函数和静态宏函数实现与低层驱动程序之间的交互。固件包的最底层包括板级支持包（Board Support Package，BSP）以及 HAL 和 LL 库的 API。

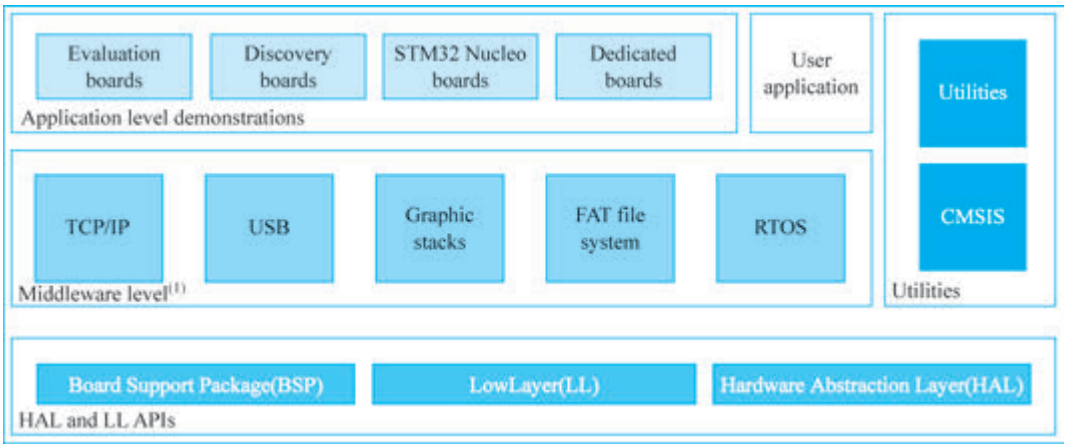


图 3-3 STM32CubeF4 固件包的组成

1) BSP

BSP 提供了一组与硬件电路板上的硬件组件(例如音频编解码器、I/O 扩展器、触摸屏、SRAM 驱动程序、LCD 驱动等)相关的 API,主要由两部分组成,即与电路板上的外部设备相关而与 STM32 无关的组件驱动函数以及与特定硬件电路板相关的设备驱动函数。

2) HAL 库 API

该层提供了底层驱动程序和硬件接口方法,以便与上层应用程序、库和堆栈进行交互。该层提供了通用的多实例,并面向应用功能的 API。例如,为 MCU 的通信外设(如 I2S 和 UART)提供了允许初始化和配置外设的 API 函数,可以实现基于轮询、中断或 DMA 方式的数据传输管理等。

3) LL 库 API

该层提供了寄存器级别的低级 API 函数,具有更好的优化特性,但移植性更低,使用时需要对底层硬件及其外设有比较深入的了解。这些 API 函数包括:外围设备的参数化初始化函数、相关函数的复位与初始化函数,以及直接寄存器访问函数。LL 库与 HAL 库完全独立,能够独立工作,覆盖了外设的所有功能特性。

2. 固件包的目录结构

STM32CubeF4 固件包最终体现为一系列文件夹和文件,其中主要包括 Documentation(文档)、Drivers(驱动)、Middlewares(中间件)、Projects(工程)和 Utilities(实用工具)等文件夹。STM32CubeF4 固件包的目录结构如图 3-4 所示。

在 Documentation 文件夹中只有一个文件 STM32CubeF4GettingStarted.pdf,该文件是固件包的一个总体说明文档。Middlewares 文件夹中是 ST 公司提供的中间件和其他第三方软件。Projects 文件夹中提供了 ST 公司生产的各种型号开发板和基于不同集成开发环境的示例工程。Utilities 文件夹中提供了一些实用工具。

在整个目录结构中,最重要的是 Drivers 文件夹,其中包括 BSP、CMSIS 和 STM32F4xx_HAL_Driver 三个子文件夹。在 CMSIS\Device\ST\STM32F4xx\子文件夹中又包括

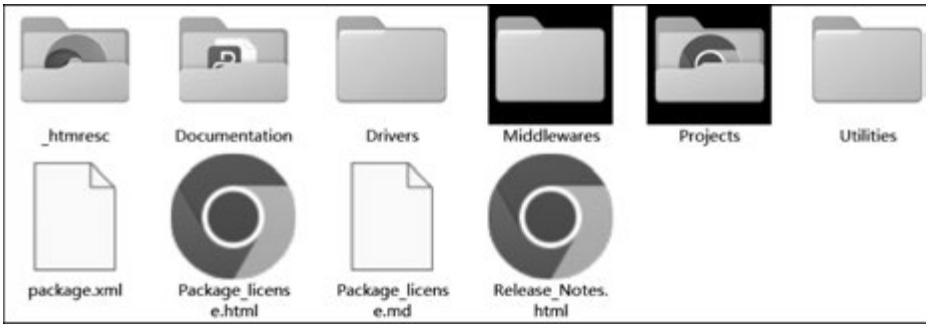


图 3-4　STM32CubeF4 固件包的目录结构

Include 和 Source 两个子文件夹，其中存放的是与 STM32F4 系列 MCU 器件相关的头文件和源文件，例如 system_ stm32f4xx. c/. h、stm32f4xx. h 和 stm32f407xx. h 文件。

在 STM32F4xx_HAL_ Driver 文件夹中有子文件夹 Inc 和 Src，这两个文件夹中所有文件的文件名中都有前缀 stm32f4xx_ hal 或 stm32f4xx_ ll，分别表示 HAL 库文件和 LL 库文件。例如，在 Inc 文件夹中有 stm32f4xx_ hal. h、stm32f4xx_hal_conf_template. h、stm32f4xx_ hal_def. h 和 stm32f4xx_hal_ cortex. h 等 HAL 库头文件。这些头文件和源文件构成了 HAL 库的主要内容，在后续基于 HAL 库的程序设计中都将大量用到。


第 4 集
微课视频

3. CMSIS

上述 STM32CubeF4 固件包遵循 Cortex 微控制器软件接口标准(Cortex-Microcontroller-Software-Interface Standard，CMSIS)，该标准由 Arm 公司与多家芯片和软件供应商合作定义，通过提供内核与外设、实时操作系统及中间设备之间的通用接口，为开发者建立了一致的软硬件交互规范。这不仅简化了软件的复用，降低了微控制器开发人员的学习成本，还缩短了新设备的上市周期。

此外，CMSIS 也提供了一系列的中间件组件，包括实时操作系统(Real-Time Operating-System，RTOS)的抽象层、硬件抽象层驱动程序以及用于实现各种通信协议的软件包。这些中间件组件进一步简化了开发流程，使得开发者可以专注于应用层的开发，而不必从零开始构建底层功能。由于 CMSIS 具有这些特性，它已经成为嵌入式系统开发中不可或缺的一部分。

CMSIS 可以分为多个软件层次，分别由 Arm 公司和芯片供应商提供各层次组件，其中芯片供应商主要提供微控制器外设访问层(提供片上所有外设的定义)和外设访问函数。Arm 为多种编译器提供的软件组件包括内核设备访问层和中间设备访问层，其中内核设备访问层包含用来访问内核的寄存器设备的名称、地址定义和基础函数，以及为 RTOS 定义的独立于微控制器的接口，而中间设备访问层为应用程序提供访问外设的通用方法。

3.2　HAL 库

HAL 库是一种用于嵌入式系统开发的软件库，尤其针对 STM32 微控制器设计。HAL 库是 STM32 微控制器开发中的重要工具之一，它提供了标准化的接口和丰富的功能，能够

帮助开发者更加高效地开发嵌入式应用程序。

HAL库适用于需要快速开发、降低开发难度和提高开发效率的场景，尤其适合初学者和需要快速上手STM32微控制器开发的开发者。同时，对于需要跨平台开发的项目，HAL库也是一个很好的选择。

3.2.1 HAL库的特点与核心组件

HAL库简化了硬件访问的过程，使得开发者可以更加专注于应用层的开发。同时，HAL库提供了标准化的接口和丰富的示例代码，加速了开发过程并降低了开发难度。此外，HAL库还支持跨平台开发，使得开发者可以轻松地在不同的STM32微控制器之间迁移代码。

1. HAL库的特点

概括起来，HAL库具有如下特点。

(1) 标准化接口：HAL库提供了一组标准化的API，用于访问STM32微控制器的各种硬件功能，如GPIO、USART、ADC等。这些API提供了一致的函数命名和参数，使得开发者能够轻松地在不同的STM32微控制器上使用相同的代码来配置和初始化硬件资源。

(2) 底层驱动：HAL库包含底层驱动代码，这些代码负责控制和管理硬件外设，处理与硬件的通信、状态监测和错误处理等任务。这使得开发者可以更专注于应用程序的开发，而无须深入了解底层硬件的细节。

(3) 跨平台支持：由于HAL库提供了标准化的接口，开发者可以轻松地在不同的STM32微控制器之间迁移代码，而无须进行大量的修改。这为产品的开发和维护带来了更大的灵活性。

(4) 丰富的示例和模板：HAL库通常提供丰富的示例代码和模板，这些示例可以作为开发的起点，帮助开发者快速上手并理解如何使用不同的硬件功能。

2. HAL库的核心组件

STM32固件包中的HAL库由如下3个核心组件构成。

(1) HAL Core：HAL库内核，提供了与MCU内核相关的通用硬件操作接口，如时钟管理、NVIC配置等。这些接口是HAL库的基础，为开发者提供了对STM32微控制器内核功能的访问。

(2) HAL Drivers：HAL库驱动，针对特定片上外设模块的驱动程序，如GPIO、ADC、USART等。这些驱动程序封装了与硬件交互的细节，为开发者提供了简单易用的API函数来操作这些硬件模块。

(3) HAL Configuration Tool：图形化配置工具，允许开发者配置STM32微控制器的硬件参数，并生成初始化代码。这使得开发者可以更加便捷地使用HAL库初始化硬件模块。

在应用程序设计和HAL库的移植过程中，与上述核心组件相对应的HAL库文件可以分为3类，即系统配置文件、HAL库配置文件和HAL库外设驱动函数文件。

3.2.2 系统配置文件

系统配置文件主要包括内核接口文件和 STM32 MCU 配置文件。对 CM4 内核来说，其接口文件名为 core_cm4.h，该文件位于固件包的 Driver\CMSIS\Core\Include 文件夹中，其中定义和封装了与内核外设相关的宏、结构体及操作访问函数，任何基于 CM4 的 STM32 应用程序中都必须包含该头文件。

与 STM32 系列 MCU 相关的配置文件主要有 stm32f4xx.h、stm32f407xx.h 和 system_stm32f4xx.h 等，这几个文件位于固件包的 Drivers\CMSIS\Device\ST\STM32F4xx\Include 文件夹中。

1. 文件 stm32f4xx.h

该文件是 CMSIS 标准中 STM32F4xx 芯片外设访问底层头文件，其中指定了应用系统所需的 STM32 F4xx 器件，并指定是否启用 HAL 库。该文件的主要代码如下。

```
#if !defined (STM32F405xx) && !defined (STM32F415xx) && !defined (STM32F407xx)
    … && !defined (STM32F423xx)
   /* #define STM32F405xx */
        /* #define STM32F415xx */
        /* #define STM32F407xx */
        … …
#endif
#if !defined (USE_HAL_DRIVER)
        … …
        /* #define USE_HAL_DRIVER */
#endif
```

上述两个条件编译语句分别用于定义 STM32F407xx 和 USE_HAL_DRIVER 宏，从而指定采用基于 HAL 库对 STM32F407xx 系统进行程序设计。

有了上述宏定义后，该文件后面有如下条件编译语句。

```
#if defined (USE_HAL_DRIVER)
        #include "stm32f4xx_hal.h"
#endif
```

其作用是，如果定义了 USE_HAL_DRIVER 宏，则导入头文件 stm32f4xx_hal.h。因此，在基于 HAL 库的程序开发中，就自动导入了该头文件，从而能够在程序设计中使用 HAL 库。

文件 stm32f4xx.h 中还有如下条件编译语句。

```
#if defined(STM32F405xx)
        #include "stm32f405xx.h"
#elif defined(STM32F415xx)
        #include "stm32f415xx.h"
#elif defined(STM32F407xx)
        #include "stm32f407xx.h"
… …
#endif
```

这些语句的作用是如果定义了(指定)某个 STM32F4xx 型号的芯片,则导入相应的头文件。例如,如果在创建工程时指定 MCU 的型号为 STM32F407xx,则导入头文件 stm32f407xx.h。

2. 文件 stm32f407xx.h

该文件中定义了 STM32F407 系列 MCU 中所有片上外设的结构体和存储器地址映射、外设寄存器声明和位定义,以及外设寄存器访问宏。例如,在该文件中有如下外设存储器映射定义。

```
#define   PERIPH_BASE            0x40000000UL
#define   APB1PERIPH_BASE        PERIPH_BASE
#define   APB2PERIPH_BASE        (PERIPH_BASE + 0x00010000UL)
#define   AHB1PERIPH_BASE        (PERIPH_BASE + 0x00020000UL)
#define   AHB2PERIPH_BASE        (PERIPH_BASE + 0x10000000UL)
#define   GPIOA_BASE             (AHB1PERIPH_BASE + 0x0000UL)
#define   GPIOB_BASE             (AHB1PERIPH_BASE + 0x0400UL)
… …
#define   GPIOI_BASE             (AHB1PERIPH_BASE + 0x2000UL)
```

以及如下 GPIO 外设结构体指针定义。

```
#define GPIOA              ((GPIO_TypeDef *) GPIOA_BASE)
#define GPIOB              ((GPIO_TypeDef *) GPIOB_BASE)
… …
#define GPIOI              ((GPIO_TypeDef *) GPIOI_BASE)
```

其中,GPIO_TypeDef 是定义在同一个文件中的 GPIO 外设结构体,该结构体中封装了 GPIO 外设内部的所有寄存器,其完整定义如下。

```
typedef struct
{
    __IO uint32_t MODER;            //模式寄存器
    __IO uint32_t OTYPER;           //输出类型寄存器
    __IO uint32_t OSPEEDR;          //输出速度寄存器
    __IO uint32_t PUPDR;            //上拉/下拉寄存器
    __IO uint32_t IDR;              //输入数据寄存器
    __IO uint32_t ODR;              //输出数据寄存器
    __IO uint32_t BSRR;             //置位/复位寄存器
    __IO uint32_t LCKR;             //锁定寄存器
    __IO uint32_t AFR[2];           //复用功能寄存器
} GPIO_TypeDef;
```

注意该结构体与第 2 章自行定义的同名结构体之间的区别。

3. 文件 system_ stm32f4xx.h

该文件是 STM32F4xx 系列 MCU 器件系统头文件,其中声明了如下两个函数。

(1) SystemInit()函数:在系统复位跳转到 main()函数之前的启动过程中,实现浮点数单元(Float Point Unit,FPU)的初始化设置、中断向量表的定位和外部存储器配置等操作。

(2) SystemCoreClockUpdate()函数:根据时钟寄存器更新系统内核时钟变量

SystemCoreClock，用于指定内核时钟 HCLK，设置 SysTick 或配置其他参数。

3.2.3 HAL 库配置文件

在基于 HAL 库的应用程序设计中，必须包含 HAL 库配置文件，以便指定启用 HAL 库，并根据应用程序的需要对 HAL 库进行适当的裁减和配置。其中常用的文件有 stm32f4xx_hal.h、stm32f4xx_hal_def.h 和 stm32f4xx_hal_cortex.h、stm32f4xx_hal_conf_template.h 等，这些文件都位于固件包的 Drivers\STM32F4xx_HAL_Driver\Inc\文件夹中。

1. 文件 stm32f4xx_hal.h

该文件中声明了实现 HAL 库的初始化、系统滴答定时和 HAL 库延时，以及 IO 重映射等库函数。在该文件对应的源文件中，定义了一个非常重要的函数 HAL_Init()，用于初始化 HAL 库。该函数的原型声明如下。

```
HAL_StatusTypeDef HAL_Init(void);
```

其中，函数的返回值为 HAL_StatusTypeDef 类型，可能的取值由文件 stm32f4xx_hal_def.h 中定义的枚举类型 HAL_StatusTypeDef 定义。

函数 HAL_Init()实现的具体功能如下。

(1) 配置 Flash 预取、指令缓存和数据缓存。

(2) 配置 SysTick 每隔 1ms 产生一个中断，该定时中断由内部 16MHz 的 HSI 时钟产生。

(3) 配置 NVIC 中断优先级分组为 4。

(4) 调用用户文件 stm32f4xx_hal_msp.c 中的 HAL_MspInit()回调函数，实现全局底层硬件初始化。

2. 文件 stm32f4xx_hal_def.h

该文件是通用 HAL 库资源定义文件，其中包含了 HAL 库中通用数据类型的定义和声明、相关枚举和结构体的定义，以及宏的定义等。例如，在该文件中定义了如下 HAL 函数操作结果返回值枚举类型。

```
typedef enum {
    HAL_OK      = 0x00U,
    HAL_ERROR   = 0x01U,
    HAL_BUSY    = 0x02U,
    HAL_TIMEOUT = 0x03U
} HAL_StatusTypeDef;
```

3. 文件 stm32f4xx_hal_cortex.h

该文件中有一些 Cortex 内核通用函数声明和定义，例如 NVIC 中的中断优先级配置、系统软复位以及 SysTick 配置等。

4. 文件 stm32f4xx_hal_conf_template.h

该文件是 HAL 配置模板文件，用于实现 HAL 库的裁减配置。在基于 HAL 库的工程创建和程序设计中，必须将该文件复制到应用工程中，并将其重命名为 stm32f4xx_hal_conf.h。该文件中的主要代码举例如下。

```
#ifdef HAL_GPIO_MODULE_ENABLED          //如果启用了 GPIO 外设
    #include "stm32f4xx_hal_gpio.h"     //则导入 GPIO 驱动函数头文件
#endif
```

3.2.4 HAL 库外设驱动函数文件

除了上述对 HAL 库本身进行初始化、裁减和定义的头文件以外，在 STM32F4xx_HAL_Driver 文件夹中还包含了所有外设驱动 HAL 库头文件和源文件。这些文件实现所有 API 函数及相关数据类型的声明和定义，文件名中都含有“_hal”，从而直观表明这是 HAL 库外设驱动函数文件。

下面以 stm32f4xx_hal_gpio.c/.h 为例，介绍 HAL 库外设驱动函数文件的基本内容。这两个文件分别为源文件和对应的头文件，源文件中定义了 GPIO 引脚初始化函数和 GPIO 引脚读写操作函数，头文件中是相关的宏定义和函数的原型声明。

1. GPIO 引脚初始化函数

该函数能够实现 GPIO 引脚的初始化配置。函数中的主要代码与在第 2 章中自定义的 GPIO_Init()函数相同。在 stm32f4xx_hal_gpio.h 头文件中，其原型声明如下。

```
void HAL_GPIO_Init(GPIO_TypeDef *GPIOx, GPIO_InitTypeDef *GPIO_Init);
```

其中，入口参数 GPIOx 是 GPIO 结构体指针，可以取值为 GPIOA、GPIOB、GPIOC 等；GPIO_Init 是 GPIO 初始化结构体指针，该结构体中保存了 GPIO 的模式配置参数。

上述两个结构体的定义分别在 stm32f407xx.h 和 stm32f4xx_hal_gpio.h 头文件中，以第二个结构体为例，其定义如下。

```
typedef struct{
    uint32_t Pin;              //GPIO 引脚
    uint32_t Mode;             //GPIO 工作模式
    uint32_t Pull;             //GPIO 上下拉模式
    uint32_t Speed;            //GPIO 工作速度
    uint32_t Alternate;        //GPIO 复用模式
}GPIO_InitTypeDef;
```

在上述结构体中，各成员的可能取值定义在同一个头文件 stm32f4xx_hal_gpio.h 中。例如，结构体成员 Mode 的取值可以是如下宏定义。

```
#define GPIO_MODE_INPUT          MODE_INPUT                    //浮空输入模式
#define GPIO_MODE_OUTPUT_PP      MODE_OUTPUT | OUTPUT_PP       //上下拉输出模式
```

```
#define GPIO_MODE_OUTPUT_OD     MODE_OUTPUT | OUTPUT_OD     //开漏输出模式
#define GPIO_MODE_AF_PP         MODE_AF | OUTPUT_PP         //上下拉复用模式
#define GPIO_MODE_AF_OD         MODE_AF | OUTPUT_OD         //开漏复用模式
#define GPIO_MODE_ANALOG        MODE_ANALOG                 //模拟模式
```

上述宏定义中的 MODE_INPUT、MODE_OUTPUT 等也是宏常量，在头文件 stm32f4xx_hal_gpio.h 中定义了这些宏常量的取值分别为 0、1、2 等。

2. GPIO 引脚读写操作函数

在 HAL 库的 stm32f4xx_hal_gpio.c/.h 文件中，定义了如下 3 个函数实现 GPIO 引脚的读写操作。

```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin);
void HAL_GPIO_WritePin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin,
                       GPIO_PinState PinState);
void HAL_GPIO_TogglePin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin);
```

以写操作函数为例，在 HAL 库中的定义如下。

```
void HAL_GPIO_WritePin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin,
                                        GPIO_PinState PinState){
    …… 
    if(PinState != GPIO_PIN_RESET) {
        GPIOx->BSRR = GPIO_Pin;
    }
    else {
        GPIOx->BSRR = (uint32_t)GPIO_Pin << 16U;
    }
}
```

该函数能够实现对指定 GPIO 端口 GPIOx 中指定引脚 GPIO_Pin 的写操作，其中各参数的含义如下。

（1）参数 GPIOx：GPIO 结构体变量，指定 GPIO 外设，取值为 GPIOA、GPIOB 等。

（2）参数 GPIO_Pin：指定 GPIO 引脚，取值可以为宏常量 GPIO_PIN_0～GPIO_PIN_15。

（3）参数 PinState：枚举类型 GPIO_PinState，用于设置指定 GPIO 引脚的高低电平，取值为该枚举类型的成员 GPIO_PIN_RESET（默认值为 0，表示输出低电平）或 GPIO_PIN_SET（默认值为 1，表示输出高电平）。枚举类型 GPIO_PinState 的定义也在头文件 stm32f4xx_hal_gpio.h 中。

3.2.5 各文件的包含关系

HAL 库中各文件之间的包含关系如图 3-5 所示。在基于 HAL 库的 STM32 工程中，任何应用程序都将从主程序文件 main.c 中的 main()函数开始运行。在该文件中将导入对应的 main.h 头文件和 gpio.h 文件。

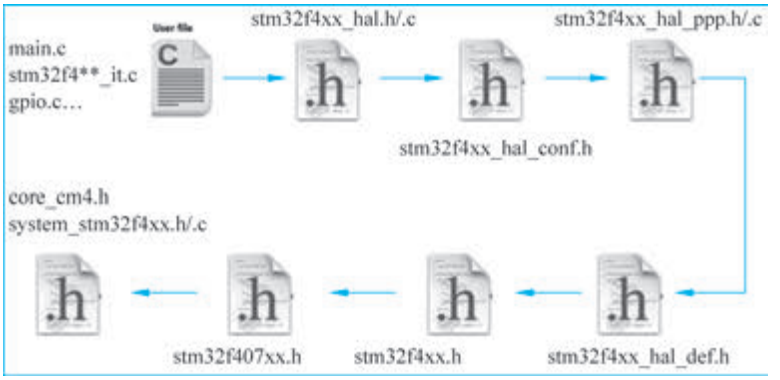


图 3-5 HAL 库中各文件之间的包含关系

在 main.h 文件中，将导入 stm32f4xx_hal.h 头文件，该头文件中提供了 HAL 模块驱动的函数声明，其中将继续导入 stm32f4xx_hal_conf.h 文件。如果工程中启用了某个外设，在该文件中将自动导入相应的头文件和源文件 stm32f4xx_hal_ppp.h/.c。例如，当启用 GPIO 外设时，将在 stm32f4xx_hal_conf.h 文件中利用条件编译语句导入 stm32f4xx_hal_gpio.h 头文件。

在 stm32f4xx_hal_gpio.h 头文件中，将导入 stm32f4xx_hal_def.h 文件，对 HAL 库中通用数据类型的定义和声明、相关枚举和结构体和宏等进行定义。该文件又将继续导入 stm32f4xx.h，并根据在应用工程中所指定的 MCU 型号自动导入 stm32f407xx.h 等文件，实现对于 MUC 中外设的结构体和存储器地址映射、外设寄存器声明以及外设寄存器访问宏的定义等。最后，导入 CM4 内核配置头文件 core_cm4.h 和系统配置头文件 system_stm32f4xx.h。

3.2.6 启动文件

启动文件是程序启动运行时首先执行的一个引导程序，其目的是在目标开发板上建立 C 语言的运行环境。在固件包的 Drivers\CMSIS\Device\ST\STM32F4xx\Source\Templates\arm 文件夹中，存放了针对 STM32F4 系列各种型号 MCU 的启动文件。例如，针对 STM32F407 系列 MCU 的启动文件名为 startup_stm32f407xx.s。

启动文件用汇编语言编写，由 ST 公司官方提供，在 CubeIDE 中创建工程时会自动添加到工程中。启动文件最前面的注释概括了该文件的主要功能如下。

(1) 初始化堆栈指针 SP。

(2) 初始化程序计数器 PC。

(3) 设置中断向量表的入口地址。

(4) 调用上述 SystemInit()函数，设置 C 库的分支入口“__main”，最终调用 main()函数，开始执行应用程序。

在编写 STM32 应用程序时，所有 STM32 工程都需要启动文件，一般将其放在工程文件夹的 Startup 子文件夹中。

3.3 基于 CubeMX 和 HAL 库的程序设计

CubeMX 是 STM32Cube 生态的一部分。CubeMX 采用图形化的界面对应用系统的时钟等各种外设和中间件进行配置，并根据这些配置自动生成初始化工程代码，实现 HAL 库的移植裁减和配置等。因此，采用 CubeMX 能够使用户专注于应用功能代码的开发，极大地提高程序开发的效率。

CubeMX 具有如下主要特性。

(1) 简单的项目创建方法：允许用户选择基于微控制器、硬件开发平台或者 ST 公司提供的示例工程进行工程的创建。

(2) 便捷的引脚模式和工作参数配置：利用引脚封装视图可以很方便地实现 MCU 芯片引脚的模式和工作参数配置。

(3) 独特的时钟树配置：提供了一个图形化的时钟树以便对系统时钟进行配置，时钟树可以通过采用求解器来解决配置冲突，并根据用户的设置提供最接近的配置解决方案。


第 5 集
微课视频

(4) 完善的工程代码：生成的工程代码基于 STM32Cube 固件包，其中包括 main.c 文件及配置和初始化 C 头文件，以及必需的 HAL 库和中间件库副本。在生成的工程代码中定义了用户专用部分，允许用户在这些位置添加 C 语言代码。

3.3.1 CubeMX 的主界面

启动 CubeIDE 后，新建 STM32 工程。为了采用基于 HAL 库的程序设计方法，在图 2-5 所示工程属性配置对话框中 Targeted Project Type(目标工程类型)下面确认勾选 STM32Cube。在新建的工程中，默认将创建一个与工程同名的.ioc 文件。打开该文件，即可进入 CubeMX 主界面，如图 3-6 所示。

CubeMX 主界面采用图形化的方式，实现工程的初始化配置并启用所需外设，同时配置外设的相关参数。CubeIDE 将根据这些配置，自动创建初始化工程代码。在主界面的顶部是 4 个选项卡，单击可以切换到引脚和配置(Pinout & Configuration)、时钟配置(Clock Configuration)、工程管理器(Project Manager)和工具(Tools)面板。

1. 引脚和配置面板

一个完整的引脚和配置面板如图 3-7 所示，该面板又包括如下几个视图和子面板。

1) 组件列表视图

引脚及配置选项卡面板左侧的组件列表视图按字母顺序和组件类别，以列表的形式显示由所选 MCU 所支持的所有外设和中间件。

图 3-6 CubeMX 主界面

图 3-7 引脚和配置面板

2）引脚封装和系统视图

默认情况下，引脚及配置选项卡面板右侧显示的是引脚封装视图（Pinout View），如图 3-8 所示，该视图中显示所选 MCU 的引脚封装。

在引脚封装视图中，主要的操作是对所用的 GPIO 引脚进行工作模式配置。具体操作步骤是：首先，单击相应的引脚，在弹出的快捷菜单中将列出指定引脚可用的复用模式；然后，单击选中期望的复用模式，该选项将用绿色高亮显示。此外，在引脚封装视图中，右键单击某引脚，在弹出的快捷菜单中单击 Enter User Label（输入用户标签），在弹出的对话框中可以为指定引脚设置信号标签。

图 3-8　引脚封装视图

单击引脚封装视图上方的标签 System View，可以由引脚封装视图切换到如图 3-9 所示的系统视图。系统视图提供了所有软件可配置组件的简介，单击相应按键可以打开给定组件的配置选项(模式和配置面板)。按键图标的颜色反映了当前各组件的配置状态。系统视图实际上与工程页面窗口左侧的组件列表是等效的。

3) 模式和配置子面板

在引脚及配置面板左侧的组件列表视图中单击选中一个组件或外设后，将在引脚及配置选项卡面板的中间打开如图 3-10 所示的模式和配置子面板 Mode and Configuration。在这两个子面板中可以进一步调整所选组件的功能模式，并配置其初始化参数。

对不同的外设和组件，子面板中所提供的选项卡各不相同。大多数外设都有 Parameter Settings(参数设置)、User Constants(用户常数)、NVIC Settings(NVIC 设置)、DMA Settings(DMA 设置)和 GPIO Settings(GPIO 设置)这几个选项卡。其中，参数设置

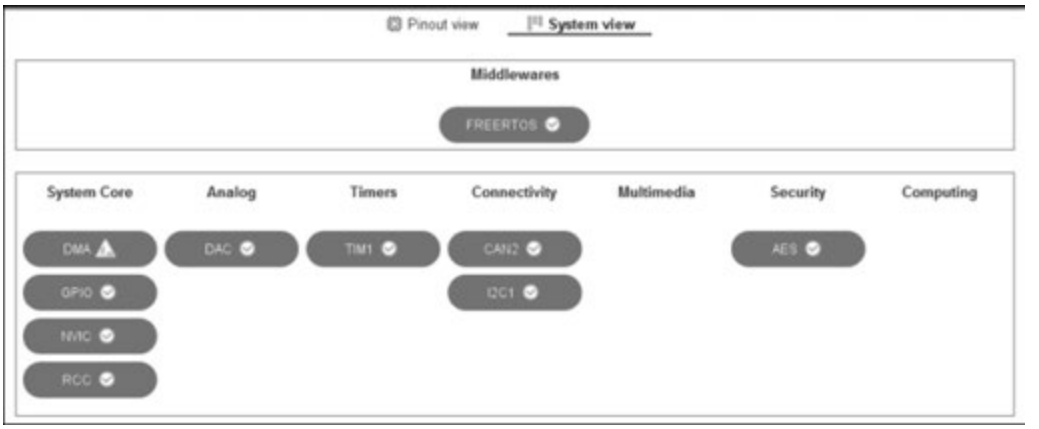


图 3-9 系统视图

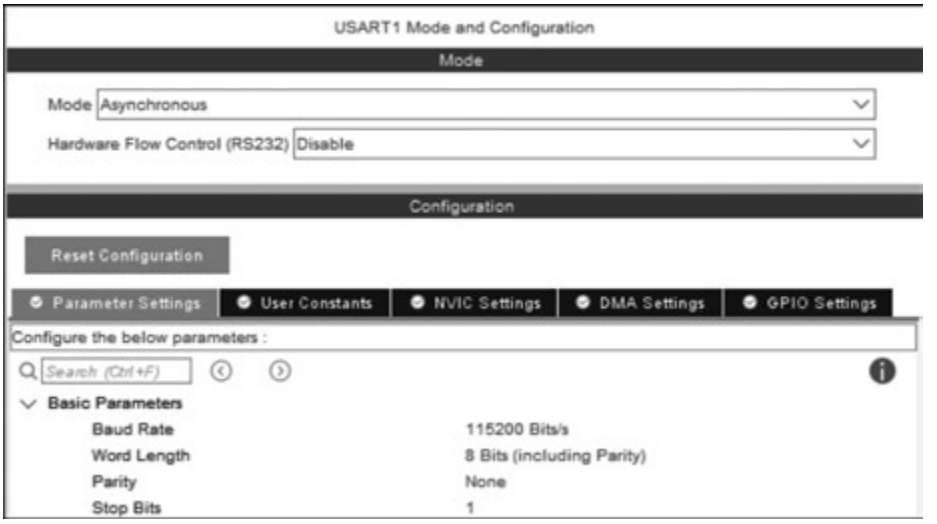


图 3-10 模式和配置子面板

选项卡用于配置所需外设组件的参数，用户常数选项卡用于设置在整个工程中通用的常数，后面 3 个选项卡分别用于配置 NVIC、DMA 和 GPIO 外设的参数。

2. 时钟配置面板

时钟配置面板以图形化的方式展示了时钟树的结构，包括时钟路径、时钟源、系统时钟、各总线时钟的配置关系，以及分频器和乘法器等时钟生成元件。时钟树如图 3-11 所示。在该视图中，通过下拉菜单和按键可以修改其中各模块和部件的参数，以满足实际应用系统的要求。

1）时钟源

STM32F4 系列 MCU 的时钟源主要包括 HSI、HSE、LSI 和 LSE。对 STM32F407 系列 MCU 来说，HSE 时钟源的频率为 4～26MHz，而 HSI、LSI 和 LSE 的时钟频率分别为 16MHz、32kHz、32.768kHz。

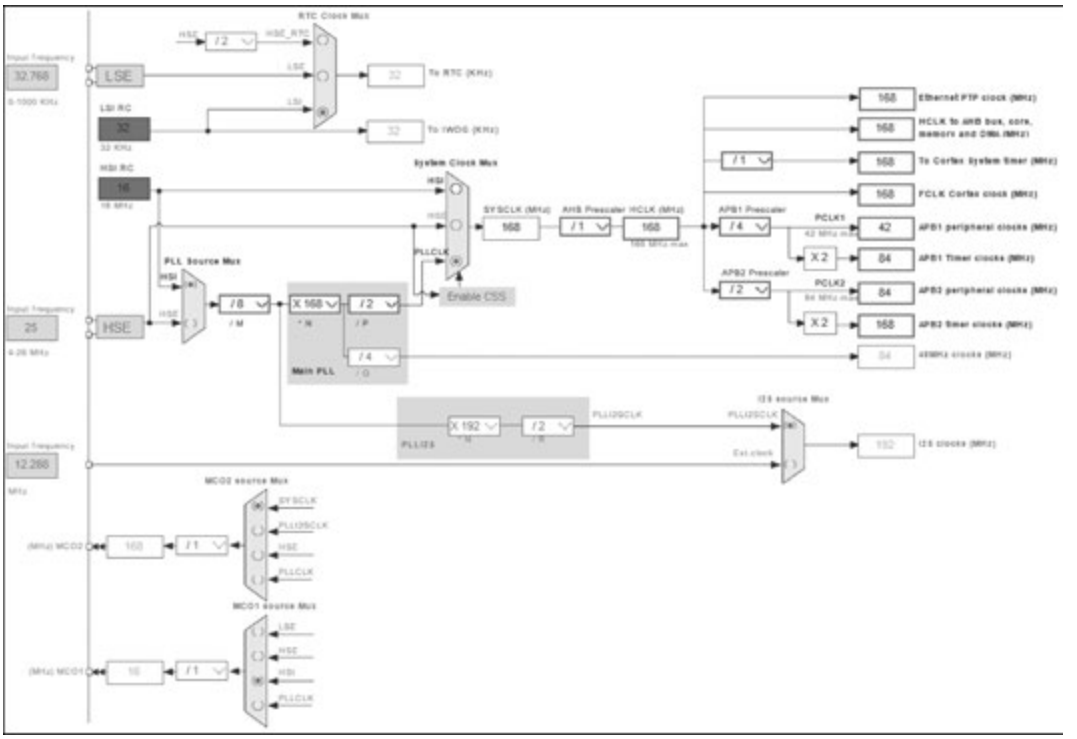


图 3-11　时钟树

MCU 芯片上电时默认由内部的 HSI 时钟启动，只有用户进行了硬件和软件的配置，芯片才会根据用户配置调试尝试切换到对应的外部时钟源。

2）系统时钟

系统时钟为整个 MCU 芯片提供时序信号。在 STM32F4 系列 MCU 中，系统时钟可以配置为 HSE、HSI 或 PLLCLK 中的一个，通过时钟树中的 System Clock Mux（系统时钟开关）进行切换和选择配置，从而得到系统时钟频率，一般为 168MHz。

3）总线时钟

不同的外设分别挂接在不同的总线上，为了启用某个外设，必须配置相应的总线时钟。例如，STM32F407 的 GPIO 外设挂接在 AHB1 上，如果在时钟树中配置 AHB Prescale 为 1，则 168MHz 系统时钟不经过分频，直接得到 AHB1 总线时钟（图中标注为 HCLK）。

此外，设置 APB1 和 APB2 的分频系数分别为 2 和 4，则配置总线时钟 PCLK1 和 PCLK2 分别为 42MHz 和 84MHz。

3. 工程管理器面板

在 CubeMX 主界面中，工程管理器面板如图 3-12 所示，包括 3 个子面板，即 Project（工程）、Code Generator（代码生成）和 Advanced Settings（高级设置）。

1）工程子面板

用于指定工程名及工程存放的位置，并编译工程所需的工具链和固件等。对本书中

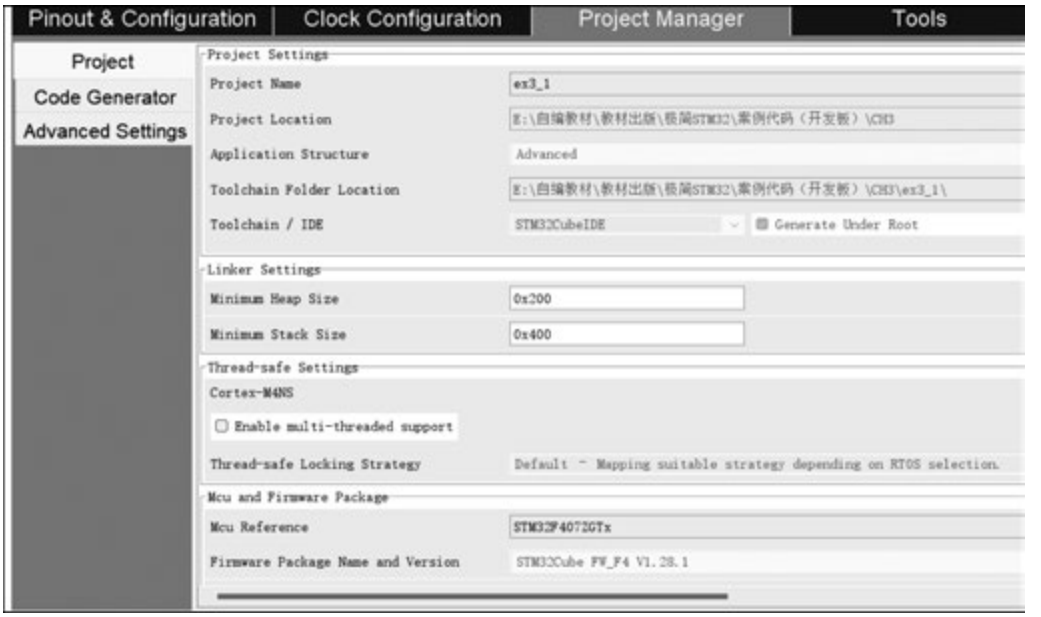


图 3-12 工程管理器面板

的所有工程，Toolchain(工具链)都自动选择为 STM32CubeIDE。该面板的所有配置基本上都不需要重新修改。此外，在面板下方显示了所选择的 MCU 器件型号及所安装的固件包。

2) 代码生成子面板

在该面板中，可以设置生成工程代码的一些选项，例如外设初始化代码的位置、固件库复制链接选项、用户代码模板选择等。对本书中后面介绍的所有案例，一种典型配置是如图 3-13 所示的工程管理器面板中的代码生成子面板。该配置表示在生成的工程代码中，只复制需要的 HAL 库文件，并且对每个外设都生成一对源文件和头文件。例如，对 GPIO 外设，在工程中将同时生成 gpio.c 和 gpio.h 两个文件。

3) 高级设置子面板

该面板中又包括 3 个子面板。在 Driver Selector(驱动选择)子面板中，可以选择基于 HAL 库还是 LL 库为指定的外设生成初始化代码。在 Generated Function Calls(生成函数调用)面板中，可以选择是否生成函数调用、是否采用静态函数调用等。在 Register Callback(寄存器回调)面板中，可以为指定外设选择是否生成寄存器回调代码，并添加到 stm32xxxx_hal_conf.h 文件中。

4. 工具面板

工具面板中主要提供了一个称为序列发生器(Sequence Generator)的实用工具，该工具允许用户根据所选的微控制器型号、电池类型等参数，对系统的平均电流、电池预期寿命以及最高工作温度等进行估算。

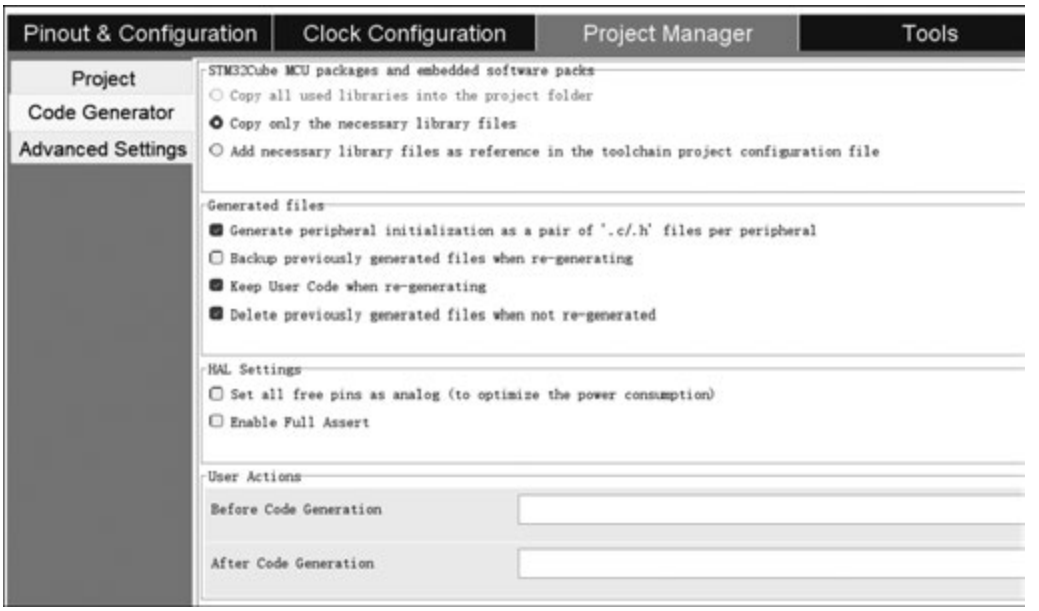


图 3-13　工程管理器面板中的代码生成子面板

3.3.2　基于 CubeMX 的程序设计流程

在 CubeIDE 中基于 CubeMX 实现 STM32 应用程序的编写，可以简单概括为两个步骤。首先利用 CubeMX 进行初始化配置并生成初始工程代码框架，之后编写应用程序实现具体的应用功能。上述两个步骤可以反复交叉进行。

1. 工程的创建

首先启动 STM32CubeIDE，并指定工作空间。在 CubeIDE 主窗口中，单击 File/New/STM32 Project 菜单命令，开始新建 STM32 工程。此时将弹出“选择目标器件”对话框，选择目标开发板所用的 MCU 器件型号。在接下来的 Setup STM32 Project（设置 STM32 工程）对话框中输入工程名，其他选项保持默认值，特别注意 Target Project Type（目标工程类型）已经默认勾选为 STM32 Project。

单击 Next 按键，弹出 Firmware Library Package Setup（固件包设置）对话框，其中可以看到已经安装的固件库版本及安装路径。此外，确保在 Code Generator Options 下面勾选默认选项 Copy only the necessary library files，该选项表示只将固件包和 HAL 库需要的文件复制到工程中。

单击对话框中的 Finish 按键，结束工程创建，此时在 CubeIDE 的主窗口中，除了显示左侧的工程浏览器以外，在窗口右侧将自动打开后缀为. ioc 的 CubeMX 文件，并显示 CubeMX 主界面。在后续的操作步骤中，如果该主界面已经关闭，在工程浏览器中双击工程的 CubeMX 文件，又可重新打开。

2. CubeMX 初始化配置

1）时钟源和系统时钟配置

在 CubeMX 的主页界面单击左侧 System Core 下面的 RCC 选项，在 High Speed Clock 框中选择 Crystal/Ceramic Resonator，表示使用外部晶振作为 HSE 时钟源。再单击窗口上面的 Clock Configuration(时钟配置)标签，打开时钟树。

大多数情况下，在时钟树中设置 HSE 左侧的 Input Frequency(输入频率)等于目标开发板上时钟电路中外部晶振的频率，在 PLL Source Mux 处选择 HSE 通道，锁相环 PLL 的分频系数 M、倍频系数 N 和分频系数 P 分别设为 25、336 和 2，系统时钟开关选择 PLLCLK，则配置系统时钟正好为(25÷25)×(336÷2)=168MHz。

根据上述时钟树配置，168MHz 系统时钟不经过分频，直接得到 AHB1 总线时钟 HCLK 的频率等于系统时钟，该频率也就是 GPIO 外设工作的最高速率。

2）GPIO 功能引脚配置

在引脚封装视图窗口配置硬件电路所用 GPIO 引脚的工作模式，之后单击窗口右侧列表中的 GPIO，将在窗口中部显示 GPIO 模式配置对话框，在这里对 GPIO 各引脚的参数进行配置。

3）外设的启用和参数配置

根据实际应用系统实现的功能不同，在工程中可能还需要启用其他外设，例如定时器、串口等。为此，可以在 CubeMX 主界面左侧的外设列表中找到并单击外设，进一步在窗口中间的模式和配置面板中对其工作模式、参数等进行配置。

4）工程配置及源代码生成

在 CubeMX 工程管理器的 Code Generator 面板中，可以对生成工程代码的相关属性进行配置。之后单击窗口工具栏上的保存按键，将根据上述配置自动生成工程文件夹和相关初始工程代码。

3. 应用程序的编写和下载运行

在根据上述配置生成工程代码的过程中，CubeMX 主要执行如下操作。

(1) 从固件包中复制相关文件到 Drivers 文件夹下的 CMSIS 和 STM32F4_HAL _Driver 文件夹。如果配置启用了中间件，相应的驱动文件将复制到 Middleware 文件夹。

(2) 根据配置生成初始化 C 语言代码源文件和头文件，并存放到工程中 Core 文件夹下的 Inc 和 Src 子文件夹中。默认情况下，将同时生成 main.c 和 main.h 文件。根据 CubeMX 中配置和启用的外设，还可能生成其他文件。

在 main.c 文件中，将调用 HAL_init()函数复位 MCU 和外设，并初始化 Flash 存储器及系统滴答定时器等，配置和初始化系统时钟及未用的 GPIO 外设，为每个启用的外设定义和调用外设初始化函数及句柄结构体(Handle Structure)，这些句柄结构体将作为参数传递到相应的 HAL 初始化函数。

在 main.h 文件中定义了引脚信号标签和用户常数等内容，并包含了 stm32f4xx_hal.h 头文件，因而可以在 main.c 文件中调用 HAL 库中的各种驱动函数。

在生成的初始工程代码中，根据需要实现的系统功能编写应用程序代码。在基于 HAL 库的程序设计中，编写程序代码主要在 main.c 文件中进行。对于 BSP 的开发和设计，还需要创建一系列文件夹和文件，用于编写和存放与系统硬件组件相关的程序代码。

程序编写完毕后，单击 Build 按键对工程代码进行编译。编译成功后得到的 ELF 或 HEX 文件存放在工程文件夹的 Debug 子文件夹下，可以通过仿真器或串口下载运行。

【实践案例 3-1】 基于 CubeIDE 和 HAL 库的程序设计

本案例要求利用开发板上的 K2 按键控制红色 LED 的闪烁。每按动(按下再释放)一次按键 K2，红色 LED 在亮与灭之间切换一次。开发板上的按键 K2 连接电路如图 3-14 所示。当 K2 按下时，PC13 输出高电平；按键释放后，PC13 输出低电平。图中的电容 C96 用于实现按键消抖。

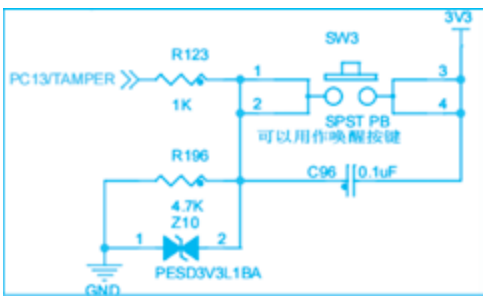


图 3-14 开发板上的按键 K2 连接电路

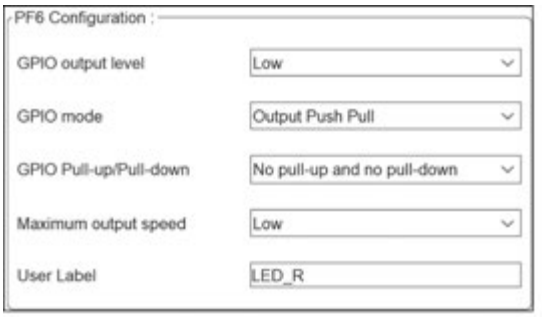
首先在硬盘上新建文件夹 CH3 作为工作空间，用于存放本章各案例的工程代码。之后启动 CubeIDE 新建工程，在弹出的“选择目标器件”对话框中选择与开发板上相同的 STM32F407ZGT6，并设置 STM32 工程名。

1. CubeMX 初始化配置

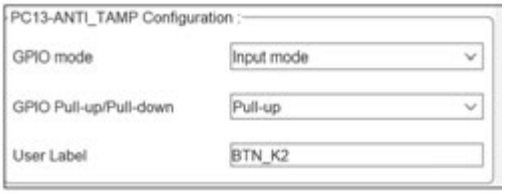
在时钟配置面板中按照前面的介绍配置系统时钟和 AHB1 时钟。这些配置在所有案例中都是首先需要执行的，后面将不再重复介绍。

在开发板上，红色 LED 和按键 K2 分别连接在 PF6 和 PC13 引脚上。因此需要配置这两个 GPIO 引脚分别为输出和输入模式。为此，在引脚封装视图窗口的右下角框中分别输入 PF6 和 PC13，将自动在引脚封装图中找到这两个引脚。单击该引脚，在弹出菜单中分别配置 PF6 和 PC13 为 GPIO_Output 和 GPIO_Input 模式。

之后单击窗口右侧列表中的 GPIO，将在窗口中部显示 GPIO 模式配置对话框。单击选中 PF6 引脚，将出现如图 3-15(a)所示的 PF6 引脚属性配置。利用同样的方法配置 PC13，PC13 引脚属性配置如图 3-15(b)所示。



(a) PF6引脚属性配置



(b) PC13引脚属性配置

图 3-15 GPIO 引脚属性配置

2. 初始工程代码解析

图 3-16 为根据上述配置并执行了一次编译以后得到的 STM32 工程目录结构，其中主要有 Includes、Core、Drivers 和 Debug 文件夹。此外，启动文件、系统调用文件 syscalls.c 和两个链接器脚本文件 STM32F407ZGTX_FLASH.ld、STM32F407ZGTX_RAM.ld 已经自动复制到工程中。

图 3-16 STM32 工程目录结构

在 Drivers 文件夹中有 CMSIS 和 STM32F4xx_HAL_Driver 两个子文件夹，这两个文件夹的目录结构与固件包中的同名文件夹相同，实际上是将固件包和 HAL 库作了必要的裁剪，并将工程所需的库文件直接复制过来得到的。创建不同的工程所需的器件、外设各不相同，因此这两个文件夹中的文件也有区别。

Core 文件夹是在工程中进行应用程序编写主要用到的文件夹，工程中各种应用程序源文件和头文件（例如 main.c、gpio.c 和 main.h、gpio.h）分别存放在其中的 Inc 和 Src 子文件夹中。此外，Startup 子文件夹中只有一个启动文件 startup_stm32f407zgtx.s。

在生成的初始工程代码中，Core/Inc 和 Core/Src 文件夹下有主函数文件 main.c/.h，由于在 CubeMX 中配置启用了 PC6 和 PC13 两个 GPIO 引脚，因此同时会生成 gpio.c 和 gpio.h 文件。

1）主函数

主函数头文件 main.h 中的主要代码如下。

```
/* Private defines ---------------------------------------------- */
#define BTN_K2_Pin          GPIO_PIN_13
#define BTN_K2_GPIO_Port    GPIOC
#define LED_R_Pin           GPIO_PIN_6
#define LED_R_GPIO_Port     GPIOF
```

在上述代码后面，利用 define 语句定义了 4 个宏，分别代表在 CubeMX 中配置的两个 GPIO 引脚，这两个引脚的用户标签（User Label）分别为 LED_R 和 BTN_K。

主函数源文件 main.c 的主要代码如下。

```
#include "main.h"                    //导入头文件
#include "gpio.h"
/* Private includes ---------------------------------------------- */
/* USER CODE BEGIN Includes */
// 用户添加的 Include 语句，导入其他所需头文件
/* USER CODE END Includes */
/* Private typedef ---------------------------------------------- */
/* USER CODE BEGIN PTD */
//自定义私有数据类型别名
```

```
/* USER CODE END PTD */
/* Private variables ---------------------------------------- */
/* USER CODE BEGIN PV */
//私有变量定义
/* USER CODE END PV */
/* Private function prototypes ------------------------------ */
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */
//私有函数声明
/* USER CODE END PFP */
/* Private user code ---------------------------------------- */
/* USER CODE BEGIN 0 */
//私有用户代码
/* USER CODE END 0 */
int main(void){                    //主函数
    /* USER CODE BEGIN 1 */
    // 用户代码 1
    /* USER CODE END 1 */
    //MCU Configuration---------------------------------- */
    HAL_Init();                    //HAL 库初始化
    /* USER CODE BEGIN Init */
    // 其他所需的 HAL 初始化代码
    /* USER CODE END Init */
    /* Configure the system clock */
    SystemClock_Config();          //配置系统时钟
    /* USER CODE BEGIN SysInit */
    /* USER CODE END SysInit */
    /* Initialize all configured peripherals */
    MX_GPIO_Init();                //GPIO 初始化
    /* USER CODE BEGIN 2 */
    //其他用户初始化代码
    /* USER CODE END 2 */
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1) {
        /* USER CODE END WHILE */
        // 用户代码
        /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}
void SystemClock_Config(void){     //系统时钟配置函数
    …… 
}
/* USER CODE BEGIN 4 */
//用户代码
/* USER CODE END 4 */
…… 
```

上述代码中的中文部分是编者添加的注释和说明。整个主程序源文件采用标准 C 语言程序结构。在主函数 main()之前需要导入所需的头文件，并进行该程序中所用的私有变

量和宏的定义以及该文件中后面定义的函数原型的声明等。

在 main()函数中,依次调用 HAL_Init()、SystemClock_Config()和 MX_GPIO_Init()函数,实现 HAL 库、系统时钟和 GPIO 外设的初始化。main()函数的最后是一条 while 死循环语句,在其中可以添加用户代码具体的应用功能。

用户编写应用程序的主要工作就是在这两个文件中添加必要的用户代码,包括所需的其他私有变量、宏定义、自定义函数,以及实现具体功能的用户代码。在这部分代码中,用户还可以根据需要自行添加其他初始化代码。

需要注意的是,用户自己添加的所有代码应当放在上述模板框架中的指定区域,这些区域前后分别用两行注释/* USER CODE BEGIN# */和/* USER CODE END# */进行标注,称为用户代码保护区。如果用户代码没有放在保护区内,则在修改 CubeMX 中的工程和外设配置并重新生成工程代码后,原来的用户代码都将不复存在,需要用户重新编写。

2) SystemClock_Config()函数

在 main()函数中调用了 SystemClock_Config()函数,根据 CubeMX 中对系统时钟的配置生成相应的代码,实现 RCC 振荡器、系统时钟和各总线时钟的初始化。该函数中的主要代码如下。

```
void SystemClock_Config(void){
    RCC_OscInitTypeDef     RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef     RCC_ClkInitStruct = {0};
    …… 
    // 初始化 RCC 振荡器
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    …… 
    // 初始化 CPU、AHB 和 APB 总线时钟
    RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|
                        RCC_CLOCKTYPE_SYSCLK|
                        RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
    RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
    RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
    …… 
}
```

在上述函数中,首先定义了结构体 RCC_OscInitTypeDef 和 RCC_ClkInitTypeDef 变量,用于保存在 CubeMX 中对时钟树的配置。之后,调用 HAL_RCC_OscConfig()和 HAL_RCC_ClockConfig()两个函数设置 RCC 中的各寄存器,从而实现 RCC 及系统时钟和各总线时钟的配置。这些代码根据 CubeMX 中时钟树的配置自动生成,一般不用修改。

3) MX_GPIO_Init()函数

该函数根据 CubeMX 中对 GPIO 引脚的配置,实现 GPIO 外设的初始化。函数定义在 gpio.h 文件中,其完整定义代码如下。

```
void MX_GPIO_Init(void){
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    __HAL_RCC_GPIOC_CLK_ENABLE();              //GPIO 外设时钟使能
    __HAL_RCC_GPIOF_CLK_ENABLE();
    __HAL_RCC_GPIOH_CLK_ENABLE();
    HAL_GPIO_WritePin(LED_R_GPIO_Port, LED_R_Pin, GPIO_PIN_RESET);
                                               //配置 GPIO 引脚初始输出电平
    GPIO_InitStruct.Pin = BTN_K2_Pin;          //配置 PC13 引脚的工作模式
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    HAL_GPIO_Init(BTN_K2_GPIO_Port, &GPIO_InitStruct);
    GPIO_InitStruct.Pin = LED_R_Pin;           //配置 PF6 引脚的工作模式
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(LED_R_GPIO_Port, &GPIO_InitStruct);
}
```

上述函数实现的主要功能是：根据 CubeMX 中配置的 GPIO 参数设置 GPIO 初始化结构体 GPIO_InitStruct，并调用 HAL_GPIO_Init()函数实现 PF6 和 PC13 引脚的初始化；调用 HAL_GPIO_WritePin()函数设置 PF6 引脚输出的初始电平为低电平，从而使开发板上的红灯点亮。函数 HAL_GPIO_Init()定义在 stm32f4xx_hal_gpio.c 文件中。

上述函数中还调用了如下 3 个 HAL 库中的宏函数。

```
__HAL_RCC_GPIOC_CLK_ENABLE();
__HAL_RCC_GPIOF_CLK_ENABLE();
__HAL_RCC_GPIOH_CLK_ENABLE();
```

这几个宏函数实现的主要操作是通过调用 SET_BIT 宏将 RCC 寄存器 AHB1ENR 中的 GPIOFEN 位置位，从而打开 GPIOF 外设的总线时钟。

3. 应用程序编写

本案例要求控制开发板上的红色 LED 周期闪烁，也就是让 PF6 引脚输出的高低电平每隔一段时间切换一次。为此需要调用如下函数。

```
HAL_GPIO_TogglePin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
```

该函数定义在 stm32f4xx_hal_gpio.c 文件中，打开该文件或者 stm32f4xx_hal_gpio.h 头文件，找到该函数，将该函数的声明直接复制到主函数中，并正确设置函数的参数即可。

此外，程序中需要不断检测按键 K2 是否按下，为此还需调用如下函数。

```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
```

该函数不断读取检测指定 GPIO 引脚，并返回 GPIO_PinState 枚举类型的结果。该枚举类型两个成员，即 GPIO_PIN_SET 和 GPIO_PIN_RESET，分别代表指定 GPIO 引脚为高电平和低电平。

上述两个函数和枚举类型 GPIO_PinState 都定义在 stm32f4xx_hal_gpio.h 头文件中。在程序中，右击这些数据类型和函数名，在弹出的快捷菜单中单击 Open Declaration（打开声明）菜单命令，即可自动跳转到相应的定义和声明处。

根据硬件电路连接，当 K2 未按下时，PC13 引脚上低电平；当 K2 释放后，PC13 引脚上变为高电平。因此，在程序中需要两次调用 HAL_GPIO_ReadPin()函数进行检测，分别等待按键 K2 按下和释放。当检测到 K2 按下一次，并等到释放后，将 PF6 引脚输出的高低电平取反一次，从而控制红色 LED 亮灭切换一次。在主函数中，相应的代码如下。

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1){
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
    while(HAL_GPIO_ReadPin(BTN_K2_GPIO_Port, BTN_K2_Pin) == GPIO_PIN_RESET);
                                                          //等待按键 K2 按下
    while(HAL_GPIO_ReadPin(BTN_K2_GPIO_Port, BTN_K2_Pin) == GPIO_PIN_SET);
                                                          //等待 K2 释放
    HAL_GPIO_TogglePin(LED_R_GPIO_Port, LED_R_Pin);       //LED 亮灭切换
}
/* USER CODE END 3 */
```

【实践练习 3-1】 基于 CubeIDE 和 HAL 库的程序设计

本案例要求利用 CubeMX 配置和创建工程，通过调用 GPIO 的 HAL 库函数实现开发板上红、绿、蓝三颜色 LED 每隔一段时间轮流点亮。

本章小结

由于 STM32 的外设资源丰富，其寄存器的数量和复杂度远高于传统单片机，因而在程序设计中广泛采用库开发方式。本章对 STM32 的固件包和 HAL 库，以及相应的开发工具 CubeMX 作了简要介绍。

1. STM32 固件包

（1）STM32 固件包采用分层结构，其中提供了多种软件组件，包括 HAL 库、LL 库、中间件、标准外设库等。

（2）STM32 固件包遵循 CMSIS 标准，该标准是 Arm Cortex 微控制器软件接口标准，由 Arm 公司与多家不同的芯片和软件供应商合作定义，提供了内核与外设、实时操作系统和中间设备之间的通用接口。

2. HAL 库

（1）HAL 库简化了硬件访问的过程，使得开发者可以更加专注于应用层的开发。同时，HAL 库提供了标准化的接口和丰富的示例代码，加速了开发过程并降低了开发难度。HAL 库还支持跨平台开发，开发者可以轻松地在不同的 STM32 微控制器之间迁移代码。

（2）STM32 固件包中的 HAL 库由 3 个核心组件构成，即 HAL Core、HAL Drivers 和 HAL Configuration Tool。

（3）在应用程序设计和 HAL 库的移植过程中，需要用到的库文件可以分为 3 类，即系统配置、HAL 库配置文件和 HAL 库外设驱动函数文件。

3. 基于 CubeMX 和 HAL 库的程序设计

（1）CubeMX 是 ST 公司 STM32Cube 生态的一部分，是适用于所有 STM32 型号的配置工具。CubeMX 主界面采用图形化的方式，实现工程的初始化配置、启用所需外设，并配置外设的相关参数。通过主界面顶部的 4 个选项卡，可以在各种面板之间进行切换，实现引脚、时钟配置和工程管理等。

（2）基于 CubeMX 进行 STM32 应用程序的开发，首先要在 CubeMX 中配置时钟源和系统时钟，启用所需的外设，并进行功能参数及初始化配置。之后 CubeMX 根据用户配置生成相应的初始工程代码。在 CubeIDE 打开生成的工程代码后，再添加和修改代码实现要求的系统功能。

（3）在基于 HAL 库的 GPIO 应用程序设计中，首先根据硬件电路连接在 CubeMX 中配置所用 GPIO 引脚的工作模式、工作速度、用户标签和初始电平等参数。在生成的工程代码中，将自动创建 MX_GPIO_Init()函数以实现 GPIO 外设的初始化。编写应用程序的主要工作只需要在 main.c 文件中通过调用定义在 stm32f4xx_hal_gpio.c/.h 文件中的 GPIO 操作函数即可实现具体的应用功能。

本章习题

一、选择题

1. 在 HAL 库固件包中，与内核外设相关的宏、结构体及操作访问函数定义在（　　）文件中。

A. core_cm4.h　　B. stm32f4xx.h

C. stm32f407xx.h　　D. system_ stm32f4xx.h

2. 在基于 HAL 库的程序中，必须导入头文件（　　）中，以便实现 HAL 库的初始化等。

A. stm32f4xx.h　　B. stm32f4xx_hal.h

C. stm32f407xx.h　　D. stm32f4xx_hal_gpio.h

3. STM32F407 系列 MCU 中所有外设的结构体定义、外设寄存器声明以及外设寄存器访问宏定义都在（　　）文件中。

A. stm32f4xx.h　　B. stm32f4xx_hal.h

C. stm32f407xx.h　　D. stm32f4xx_hal_gpio.h

4. 下面的语句能够使 PG10 引脚输出低电平，而其他引脚输出电平保持不变的是（　　）。

A. HAL_GPIO_WritePin(GPIOG,GPIO_PIN_10,GPIO_PIN_SET)

B. HAL_GPIO_WritePin(GPIOG,GPIO_PIN_10,GPIO_PIN_RESET)

C. HAL_GPIO_TogglePin(GPIOG,GPIO_PIN_10)

D. HAL_GPIO_ReadPin(GPIOG,GPIO_PIN_10)

5. 在 CubeMX 的时钟树配置中,对 STM32F407 系列 MCU,通常配置系统时钟为(　　)MHz。

A. 168　　B. 84　　C. 42　　D. 25

6. 在 CubeMX 中,当配置某个 GPIO 引脚为(　　)模式时,可以进一步将其配置为上/下拉或浮空模式。

A. 输出　　B. 输入　　C. 模拟　　D. 复用

二、填空题

1. STM32 的固件包是基于_____标准建立的,该标准由 Arm 公司与多家不同的芯片和软件供应商合作定义。

2. 在执行 STM32F407xx 应用程序时,必须首先调用 SystemInit()函数实现系统初始化。在 HAL 库中,该函数的声明在头文件_____中。

3. 在 STM32 应用程序中,决定是否使用 HAL 库的条件编译语句在文件_____中。

4. 在 CubeMX 的时钟树配置中,对 STM32F407 系列 MCU,通常配置 AHB 和 APB1 的时钟频率分别为_____和_____。

5. 在 CubeMX 中,当配置某个 GPIO 引脚为输出模式时,可以进一步将该引脚配置为_____或 _____模式。

6. 在 CubeMX 自动生成的工程代码中,函数_____将被自动创建,并在该函数中能够自动调用 HAL 库中的_____函数对 GPIO 引脚进行初始化配置。

三、问答题

1. 简述 CMSIS。

2. 简述在 STM32 应用程序的开发过程中,CubeMX 的主要作用。

3. 定义函数 Get_PinNum()实现如下功能:将入口参数 GPIO_Pin(uint16_t 类型)转换为 GPIO 的引脚序号(uint8_t 类型)返回。例如,给定 GPIO_Pin=0x0004,则返回 GPIO 引脚序号 2。

4. 修改 HAL 库中的 GPIO 引脚输出函数 HAL_GPIO_WritePin(),要求:函数的入口和出口参数不变且在函数体中能够利用 ODR 实现引脚的写操作。