第 3 章 使用 HAL 库开发 GPIO

CHAPTER 3

3.1 STM32CubeMX 概述

STM32CubeMX 是 ST 公司近年来大力推荐的 STM32 图形化配置工具，允许用户使用图形化向导来生成 C 语言工程，支持多种工具链，比如 MDK、IAR 和 TrueStudio 等，目的是使开发者的工作更轻松，开发效率更高。STM32CubeMX 软件内部集成中间件组件，可以提供对 RTOS、USB、TCP/IP 以及图形功能的中间层支持。STM32CubeMX 的主要功能包括：

(1) 通过图形向导配置引脚、时钟树、外设和中间件，并生成相应的 C 代码。

(2) 生成指定集成开发环境对应的完整项目源代码。

(3) 基于用户定义的应用顺序计算功耗。

(4) 可以直接从 ST 公司的官网导入 STM32Cube 嵌入式软件库。

(5) 集成了软件更新程序，能使 STM32CubeMX 版本保持最新。

要使用 STM32CubeMX 软件配置并生成工程项目，需要安装下面三个软件。

(1) JRE(java runtime environment)：Java 运行环境，STM32CubeMX 软件依赖 Java(V1.7 及以上版本)，本书使用 JavaSetup8u221(64 位 Windows 10 安装版)。

(2) STM32CubeMX 软件：本书使用 SetupSTM32CubeMX-6.2.0-Win，安装软件时采用默认配置即可。

(3) HAL 库：STM32 HAL 固件库，ST 公司官方推出的一套抽象层嵌入式软件。

3.2 STM32 HAL 库

3.2.1 HAL 固件库简介

ST 公司为开发者提供了三种非常方便的开发库：SPL(standard peripheral libraries，标准外设库)、HAL(hardware abstraction layer，硬件抽象层)库和 LL(low-layer，底层)库。HAL 库的设计初衷就是为编程者提供规范化的函数和宏指令来降低操作 STM32 的难度，使编程者避开烦琐的寄存器操作。相比标准外设库，HAL 库表现出了更高的抽象整合水平，它的 API 集中关注各个外设的公共函数功能，便于定义一套通用的、对用户友好的 API 函数接口，从而可以轻松地在各种 STM32 产品间进行软件移植。HAL 库的主要特点如下

所述。

(1) 提供了通用的应用程序编程接口(API),覆盖了外设的常见功能,为不同家族芯片间的软件移植(如 F1→F0)提供了可能。

(2) 支持轮询、中断和 DMA 共 3 种 API 编程模型。

(3) 所有 API 均符合 RTOS 规范:可重入 API 及轮询模式下全部使用超时参数。

(4) 支持用户回调功能机制,当外设中断或错误产生时,将会调用用户回调(callback)函数做相应处理。

(5) 支持对象锁定机制,提供了更加安全的硬件访问方式,防止软件对共享资源的多重访问。

(6) 所有阻塞过程都使用可编程的超时机制,提高了软件的可靠性和实时性。

STM32 HAL 库可以从官网下载,也可以直接从本书的配套资料中复制,本书例程基于 STM32Cube_FW_F1_V1.8.0 库。HAL 库可以选择在线或者离线安装,一般采用离线安装的方式:打开安装好的 STM32CubeMX 软件,单击菜单栏的 Help→Updater Settings,选择 HAL 库文件夹路径,再将下载的 STM32Cube_FW_F1_V1.8.0 库解压到该路径下即可,如图 3-1 所示。

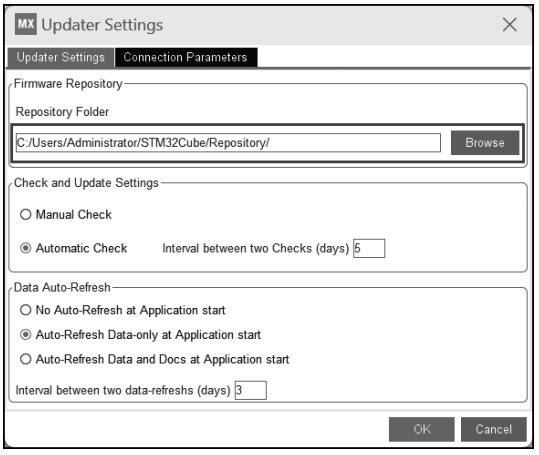

图 3-1 选择 HAL 库文件夹路径示意图

3.2.2 HAL 库文件

解压 STM32Cube_FW_Fl_V1.8.0 HAL 库文件,打开库文件目录,库中各文件夹的内容说明如图 3-2 所示。

在使用 HAL 库开发时,需要把目录 Drives 下的 CMSIS、STM32F1xx_HAL_Driver 文件夹中内核与外设的库文件添加到工程中。

1. CMSIS 文件目录

CMSIS 文件目录下有很多文件夹,文件夹的具体说明如图 3-3 所示,平时使用最多的是 Include 和 Device 中的文件。

文件夹 Include 包含 CMSIS 标准核内设备函数层的与 Cortex-M 内核相关的头文件,这些头文件给基于 Cortex-M 内核设计 SOC 的芯片商提供了进入内核的接口,其他使用该

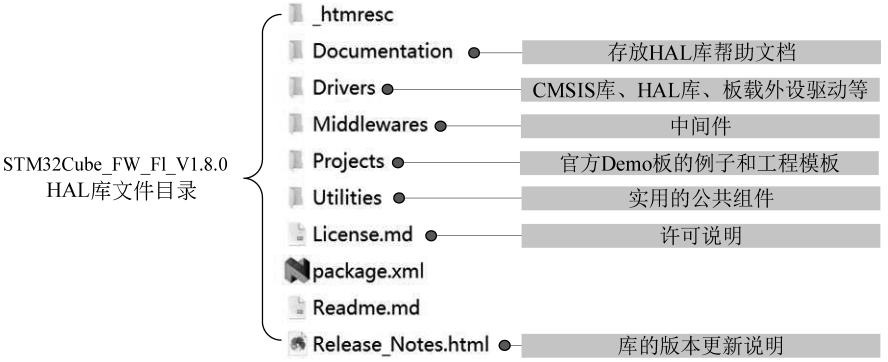

图 3-2　STM32Cube_FW_Fl_V1.8.0 HAL 库文件目录说明

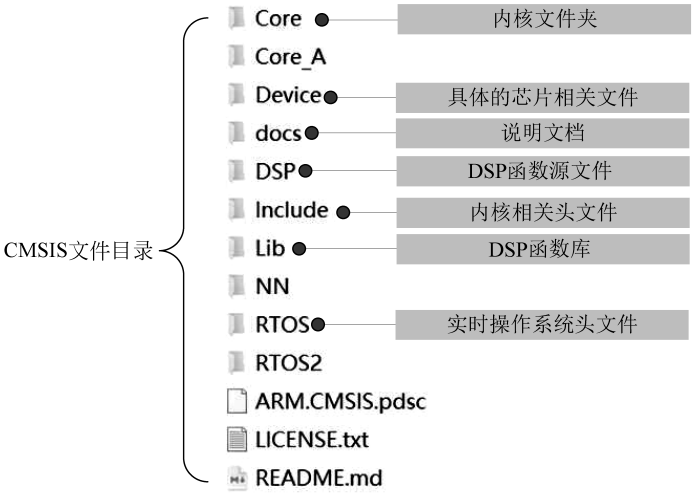

图 3-3　CMSIS 文件目录说明

内核的公司也使用这些头文件。其中，core_cm3.h 是一个比较重要的头文件，由 ARM 公司提供，遵守 CMSIS 标准。该文件包含一些与编译器相关的条件编译语句，这些语句用于屏蔽不同编译器的差异。所有 Cortex-M3 芯片的库都包含头文件 core_cm3.h。

头文件 stdint.h 包含在 core_cm3.h 中，是一个独立于处理器的 ANSI C 文件。它定义了几个在不同芯片平台上具有固定大小的整数类型，以便代码移植。下面列出了部分类型定义：

```
/* 有符号整数类型 */
typedef   signed              char int8_t;
typedef   signed short        int int16_t;
typedef   signed              int int32_t;
typedef   signed              __INT64 int64_t;

/* 无符号整数类型 */
typedef   unsigned            char uint8_t;
typedef   unsigned short      int uint16_t;
typedef   unsigned            int uint32_t;
typedef   unsigned            __INT64 uint64_t;
```

文件夹 Device 中存放了一些由 ST 公司提供的与具体芯片直接相关的文件，它主要包括以下三个文件。

(1) 启动文件：使用汇编语言编写，要根据编译平台来选择。MDK 使用的启动文件在文件夹 arm 中，2.5.2 节中使用的启动文件 startup_stm32f103xe.s 就是从这里复制的。

(2) stm32f103xe.h 文件：与芯片相关的文件，包含芯片所有外设寄存器的地址和结构体类型定义，使用 HAL 库的地方都要包含该文件。

(3) system_stm32f1xx.c 文件：包含 STM32 上电后初始化系统时钟、扩展外部存储器用的函数，例如第 2 章介绍的被启动文件调用、用于初始化时钟的 SystemInit 函数，调用这个函数后，系统时钟被初始化为 72MHz，如有需要可以修改该函数，设置所需的时钟频率。

2. STM32F1xx_HAL_Driver 文件目录

如图 3-4 所示，STM32F1xx_HAL_Driver 文件目录下有 Inc 和 Src 两个文件夹，这两个文件夹用于存放 ST 公司编写的 STM32F1 片上外设的驱动文件，每个外设都有一对 *.h 和 *.c 文件。

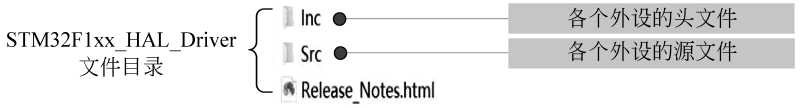

图 3-4　STM32F1xx_HAL_Driver 文件目录结构

表 3-1 对 STM32 外设驱动库文件进行了解释说明，表中 ppp 表示外设的名称，如 gpio。

表 3-1　STM32 外设驱动库文件解释

文　　件	解　　释
stm32f1xx_hal_ppp.c	外设驱动程序主文件，如 stm32f1xx_hal_gpio.c
stm32f1xx_hal_ppp.h	驱动程序主文件的头文件，包括常见的数据类型、枚举、结构和宏定义等，如 stm32f1xx_hal_gpio.h
stm32f1xx_hal_ppp_ex.c	外设扩展驱动文件，如 stm32f1xx_hal_gpio_ex.c
stm32f1xx_hal_ppp_ex.h	外设扩展驱动的头文件，如 stm32f1xx_hal_gpio_ex.h
stm32f1xx_hal.c	HAL 通用 API，如 HAL_Init、HAL_DeInit、HAL_Delay
stm32f1xx_hal.h	HAL 头文件
stm32f1xx_hal_def.h	HAL 的通用数据类型、枚举、结构和宏定义等

3. 文件 stm32f1xx_it.c 和 stm32f1xx_hal_conf.h

在\Projects\STM32F103RB-Nucleo\Templates\目录下，存放了官方的一个 HAL 库工程模板。在建立自己的工程时，需要将该模板中的三个文件，即 src\stm32f1xx_it.c、inc\stm32f1xx_it.h 和 inc\stm32f1xx_hal_conf.h 添加到工程中，其中：

(1) stm32f1xx_it.c：专门用来存放中断服务函数，文件中已经定义了一些系统异常(特殊中断)的服务函数。

(2) stm32f1xx_hal_conf.h：HAL 库的配置文件，用来对 HAL 库进行裁剪，该文件被包含在 stm32f1xx_hal.h 文件中。

3.3　GPIO 的 HAL 库用法

3.3.1　GPIO 寄存器结构体 GPIO_TypeDef

GPIO 寄存器是通过 HAL 库中的结构体数据类型 GPIO_TypeDef 封装的，该结构体数据类型被定义在 stm32f103xe.h 文件中，代码如下：

```
typedef struct
{
    __IO uint32_t CRL;              /*端口配置低寄存器,地址偏移: 0x00*/
    __IO uint32_t CRH;              /*端口配置高寄存器,地址偏移: 0x04*/
    __IO uint32_t IDR;              /*数据输入寄存器,地址偏移: 0x08*/
    __IO uint32_t ODR;              /*数据输出寄存器,地址偏移: 0x0C*/
    __IO uint32_t BSRR;             /*位设置/清除寄存器,地址偏移: 0x10*/
    __IO uint32_t BRR;              /*端口位清除寄存器,地址偏移: 0x14*/
    __IO uint32_t LCKR;             /*端口配置锁定寄存器,地址偏移: 0x18*/
} GPIO_TypeDef;
```

结构体数据类型 GPIO_TypeDef 的成员名称和排列次序与 GPIO 端口的七个寄存器是一一对应的，__IO 表示 volatile，其具体含义参见 2.5.2 节中的介绍。

下面看一下 stm32f103xe.h 文件中 GPIOA，…，GPIOG 的定义，从端口基地址的宏定义可以看出 GPIO 是挂接在 APB2 总线上的，代码如下：

```
#define PERIPH_BASE       0x40000000UL                      /*外设基地址*/

#define APB1PERIPH_BASE   PERIPH_BASE                       /*总线基地址*/
#define APB2PERIPH_BASE   (PERIPH_BASE + 0x00010000UL)
#define AHBPERIPH_BASE    (PERIPH_BASE + 0x00020000UL)

#define GPIOA_BASE        (APB2PERIPH_BASE + 0x00000800UL)  /*端口基地址*/
#define GPIOB_BASE        (APB2PERIPH_BASE + 0x00000C00UL)
#define GPIOC_BASE        (APB2PERIPH_BASE + 0x00001000UL)
#define GPIOD_BASE        (APB2PERIPH_BASE + 0x00001400UL)
#define GPIOE_BASE        (APB2PERIPH_BASE + 0x00001800UL)
#define GPIOF_BASE        (APB2PERIPH_BASE + 0x00001C00UL)
#define GPIOG_BASE        (APB2PERIPH_BASE + 0x00002000UL)

#define GPIOA             ((GPIO_TypeDef *)GPIOA_BASE)
#define GPIOB             ((GPIO_TypeDef *)GPIOB_BASE)
#define GPIOC             ((GPIO_TypeDef *)GPIOC_BASE)
#define GPIOD             ((GPIO_TypeDef *)GPIOD_BASE)
#define GPIOE             ((GPIO_TypeDef *)GPIOE_BASE)
#define GPIOF             ((GPIO_TypeDef *)GPIOF_BASE)
#define GPIOG             ((GPIO_TypeDef *)GPIOG_BASE)
```

有了这些宏以后，就可以用便捷的方式访问 GPIO 的寄存器了，比如访问寄存器 GPOIC_CRL 的代码如下：

```
GPIOC->CRL |= (1 << 4 * 0);          /* 配置引脚 PC0 为通用推挽输出,速度为 10MHz */
```

3.3.2 GPIO 初始化结构体 GPIO_InitTypeDef

HAL 库定义的结构体 GPIO_InitTypeDef 用来将初始化 GPIO 所需用到的参数封装起来,以方便用户使用,该结构体数据类型的定义如下:

```
typedef struct
{
    uint32_t Pin;                                    /* 要配置引脚的编号 */
    uint32_t Mode;                                   /* 引脚的工作模式 */
    uint32_t Pull;                                   /* 引脚的上拉和下拉电阻选择 */
    uint32_t Speed;                                  /* 引脚的速度 */
} GPIO_InitTypeDef;
```

在初始化 GPIO 前,定义一个该类型的结构体变量,再根据要求对该变量的各个成员进行赋值,然后把这个变量作为输入参数调用 GPIO 初始化函数去配置引脚,函数的执行过程实质上就是配置 GPIO 寄存器的过程。下面对这几个结构体成员进行介绍。

1. 成员 Pin

选择要配置的引脚,可选择的范围如下:

```
#define GPIO_PIN_0      ((uint16_t)0x0001)      /* Pin 0 selected */
#define GPIO_PIN_1      ((uint16_t)0x0002)      /* Pin 1 selected */
#define GPIO_PIN_2      ((uint16_t)0x0004)      /* Pin 2 selected */
#define GPIO_PIN_3      ((uint16_t)0x0008)      /* Pin 3 selected */
#define GPIO_PIN_4      ((uint16_t)0x0010)      /* Pin 4 selected */
#define GPIO_PIN_5      ((uint16_t)0x0020)      /* Pin 5 selected */
#define GPIO_PIN_6      ((uint16_t)0x0040)      /* Pin 6 selected */
#define GPIO_PIN_7      ((uint16_t)0x0080)      /* Pin 7 selected */
#define GPIO_PIN_8      ((uint16_t)0x0100)      /* Pin 8 selected */
#define GPIO_PIN_9      ((uint16_t)0x0200)      /* Pin 9 selected */
#define GPIO_PIN_10     ((uint16_t)0x0400)      /* Pin 10 selected */
#define GPIO_PIN_11     ((uint16_t)0x0800)      /* Pin 11 selected */
#define GPIO_PIN_12     ((uint16_t)0x1000)      /* Pin 12 selected */
#define GPIO_PIN_13     ((uint16_t)0x2000)      /* Pin 13 selected */
#define GPIO_PIN_14     ((uint16_t)0x4000)      /* Pin 14 selected */
#define GPIO_PIN_15     ((uint16_t)0x8000)      /* Pin 15 selected */
#define GPIO_PIN_All    ((uint16_t)0xFFFF)      /* All pins selected */

#define GPIO_PIN_MASK   0x0000FFFFu             /* PIN mask for assert test */
```

还可以使用或运算符“|”一次选中多个引脚,例如下面的代码同时选择了 3、4、5 引脚:

```
GPIO_InitTypeDef GPIO_InitStruct = {0};
GPIO_InitStruct.Pin = GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5;
```

2. 成员 Mode

2.3.3 节介绍了 GPIO 端口可以配置的 8 种工作模式:推挽输出、开漏输出、复用推挽

输出、复用开漏输出、上拉输入、下拉输入、浮空输入和模拟输入。由于上拉和下拉是可选配置，所有 I/O 端口均可以配置为外部中断的输入端，因此在 HAL 库中，GPIO 工作模式的定义如下：

```
#define GPIO_MODE_INPUT 0x00000000u                    /*输入模式*/
#define GPIO_MODE_OUTPUT_PP 0x00000001u                /*推挽输出模式*/
#define GPIO_MODE_OUTPUT_OD 0x00000011u                /*开漏输出模式*/
#define GPIO_MODE_AF_PP 0x00000002u                    /*复用推挽输出模式*/
#define GPIO_MODE_AF_OD 0x00000012u                    /*复用开漏输出模式*/
#define GPIO_MODE_AF_INPUT GPIO_MODE_INPUT             /*复用输入模式*/

#define GPIO_MODE_ANALOG 0x00000003u                   /*模拟输入模式*/

#define GPIO_MODE_IT_RISING 0x10110000u                /*外部中断,上升沿触发检测*/
#define GPIO_MODE_IT_FALLING 0x10210000u               /*外部中断,下降沿触发检测*/
#define GPIO_MODE_IT_RISING_FALLING 0x10310000u        /*外部中断,双沿触发检测*/

#define GPIO_MODE_EVT_RISING 0x10120000u           /*外部事件模式,上升沿触发检测*/
#define GPIO_MODE_EVT_FALLING 0x10220000u          /*外部事件模式,下降沿触发检测*/
#define GPIO_MODE_EVT_RISING_FALLING 0x10320000u   /*外部事件模式,双沿触发检测*/
```

3. 成员 Pull

引脚的弱上拉和下拉电阻有如下三种配置选项：

```
#define  GPIO_NOPULL    0x00000000u                    /*无上拉和下拉电阻*/
#define  GPIO_PULLUP    0x00000001u                    /*带上拉电阻*/
#define  GPIO_PULLDOWN  0x00000002u                    /*带下拉电阻*/
```

4. 成员 Speed

引脚的输出速度等级有如下三种配置选项：

```
#define  GPIO_SPEED_FREQ_LOW (GPIO_CRL_MODE0_1)     /*低速,最大频率为 2MHz*/
#define  GPIO_SPEED_FREQ_MEDIUM (GPIO_CRL_MODE0_0)  /*中速,最大频率为 10MHz*/
#define  GPIO_SPEED_FREQ_HIGH (GPIO_CRL_MODE0)      /*高速,最大频率为 50MHz*/
```

引脚的输出速度是指 I/O 支持的高、低电平状态的最高切换频率，支持的频率越高，功耗越大。通过配置引脚的输出速度等级，选择相应的输出驱动模块，可以达到最佳噪声控制和降低功耗的目的。

3.3.3　GPIO 相关 HAL 库函数

GPIO 的 HAL 库操作函数及宏定义在 stm32f1xx_hal_gpio.c 和 stm32f1xx_hal_gpio.h 文件中。

1. 函数 HAL_GPIO_Init

函数 HAL_GPIO_Init 说明如表 3-2 所示。

表 3-2 函数 HAL_GPIO_Init 说明

函数原型	void HAL_GPIO_Init(GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)
功能描述	根据 GPIO_Init 中设定的参数初始化 GPIOx 的寄存器,实现配置 GPIOx 端口的功能
输入参数	GPIOx: x=A/B/C/D/E,选择 GPIO 端口外设
	GPIO_Init: 指向包含 GPIO 配置信息的 GPIO_InitTypeDef 结构体指针
注意事项	在 HAL 库中,每个外设都对应一个结构体 XXX_InitTypeDef(XXX 为外设名称),HAL 库函数 HAL_XXX_Init()利用该结构体类型变量去设置外设相应的寄存器,从而初始化外设
应用示例	GPIO_InitTypeDef GPIO_InitStruct = {0}; GPIO_InitStruct.Pin =GPIO_PIN_0 ; /*选择引脚 0*/ GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; /*推挽输出模式*/ GPIO_InitStruct.Pull = GPIO_PULLUP; /*带上拉电阻*/ GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW; /*低速*/ HAL_GPIO_Init(GPIOA, &GPIO_InitStruct); /*使用初始化参数配置 PA0 引脚*/

2. 函数 HAL_GPIO_ReadPin

函数 HAL_GPIO_ReadPin 说明如表 3-3 所示。

表 3-3 函数 HAL_GPIO_ReadPin 说明

函数原型	GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
功能描述	读取指定引脚的电平状态
输入参数	GPIOx: x=A/B/C/D/E,选择 GPIO 端口外设
	GPIO_Pin: 指定的引脚,范围为 GPIO_PIN_0~GPIO_PIN_15,GPIO_PIN_All
返回值	指定引脚的电平状态

应用示例	/*引脚 PE2 是否为低电平*/
	if (HAL_GPIO_ReadPin(GPIOE, GPIO_PIN_2) == GPIO_PIN_RESET)
	{
	…
	}

该函数返回值的数据类型为枚举类型 GPIO_PinState,该枚举类型定义了引脚的两种电平状态:

```
typedef enum
{
    GPIO_PIN_RESET = 0u,
    GPIO_PIN_SET
} GPIO_PinState;
```

为了探究 HAL 库函数 HAL_GPIO_ReadPin 的实现原理,下面列出了该函数的源代码,它是通过判断端口输入数据寄存器(IDR)相应位的值得到引脚的电平状态:

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

```
    GPIO_PinState bitstatus;

    assert_param(IS_GPIO_PIN(GPIO_Pin));     /* 检查参数 */

    if ((GPIOx->IDR & GPIO_Pin) != (uint32_t)GPIO_PIN_RESET)
    {
        bitstatus = GPIO_PIN_SET;            /* 置位态,即高电平 */
    }
    else
    {
        bitstatus = GPIO_PIN_RESET;          /* 复位态,即低电平 */
    }
    return bitstatus;
}
```

3. 函数 HAL_GPIO_WritePin

函数 HAL_GPIO_WritePin 说明如表 3-4 所示。

表 3-4　函数 HAL_GPIO_WritePin 说明

函数原型	void HAL_GPIO_WritePin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
功能描述	设置指定引脚输出高电平或者低电平
输入参数	GPIOx：x=A/B/C/D/E,选择 GPIO 端口外设
	GPIO_Pin：指定的引脚,范围为 GPIO_PIN_0～GPIO_PIN_15,GPIO_PIN_All
	PinState：引脚的电平状态,枚举类型
注意事项	该函数使用 GPIO 的 BSRR 寄存器进行设置,支持原子操作
应用示例	HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, GPIO_PIN_RESET);　/* PC0 输出低电平 */

下面列出 HAL 库实现函数 HAL_GPIO_WritePin 的源代码：

```
void HAL_GPIO_WritePin(GPIO_TypeDef *GPIOx,
                       uint16_t GPIO_Pin, GPIO_PinState PinState)
{
    if (PinState != GPIO_PIN_RESET)
    {
        GPIOx->BSRR = GPIO_Pin;
    }
    else
    {
        /* 高 16 位清除对应的 ODRy 位,低 16 位设置对应的 ODRy 位 */
        GPIOx->BSRR = (uint32_t)GPIO_Pin << 16u;
    }
}
```

3.4 STM32CubeMX 应用示例

3.4.1 硬件设计

LED 电路原理图见 2.5.1 节。如图 3-5 所示，四个输入按键 KEY1～KEY4 的一端并联接地，另一端连接引脚 PE2～PE5。它们在没有被按下时，与其相连的 GPIO 引脚接到电源+3.3V，引脚输入状态为高电平；当按键按下后，与其相连的 GPIO 引脚接 GND，引脚输入状态为低电平。因此只要检测出引脚的输入电平，就可以确定按键的状态。唤醒按键 WK_UP 的检测情况与此相反，请自行分析。

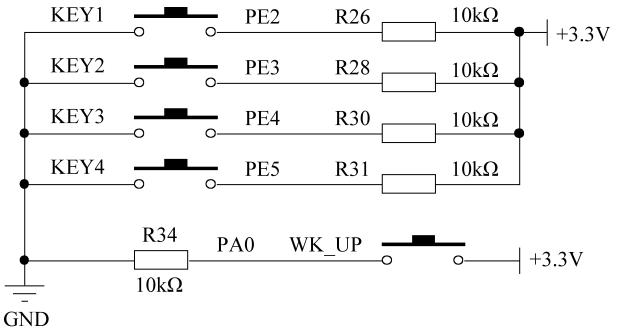

图 3-5 开发板按键原理图

3.4.2 STM32CubeMX 工程配置

启动 STM32CubeMX 软件，主界面如图 3-6 所示，图中①处打开现有的工程，图中②处创建新的工程。

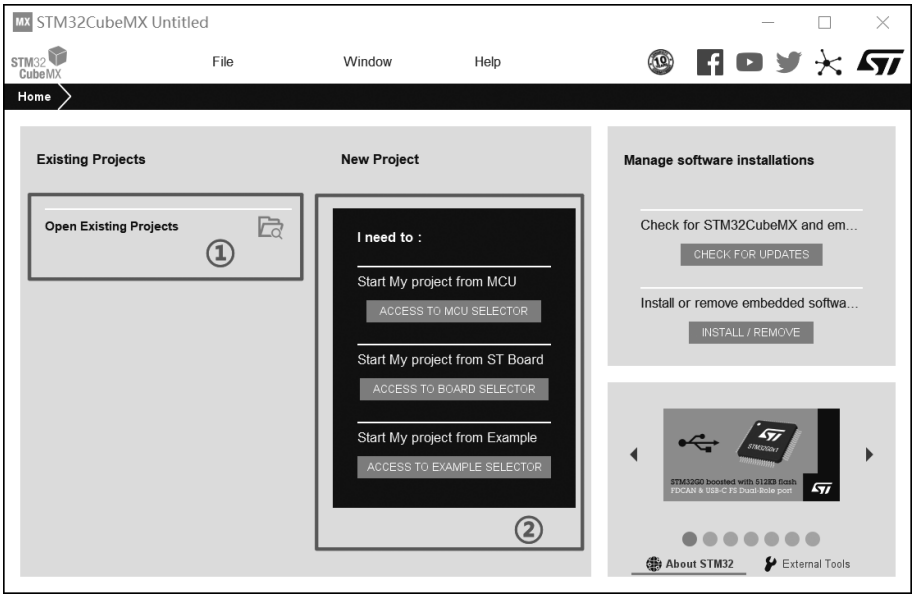

图 3-6 STM32CubeMX 主界面

可以选择使用 MCU 或者开发板来创建工程。本例选择使用 MCU 来创建工程，具体的配置步骤如下。

1. 选择 MCU 型号

如图 3-7 所示，根据开发板使用的 MCU 型号，在①处的搜索框中输入 STM32F103VE，然后选择图中②处的 STM32F103VETx 为实际使用型号。单击③处，开始工程配置。

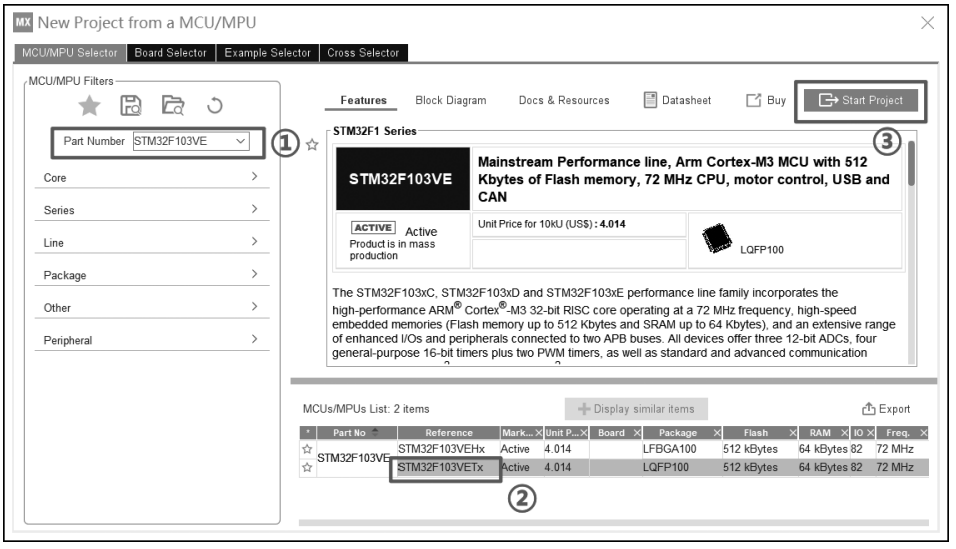

图 3-7　选择 MCU 型号

2. 配置系统时钟

如图 3-8 所示，打开 RCC 选项，选择“Crystal/Ceramic Resonator”作为 HSE，即使用外部晶振作为 HSE 的时钟源。

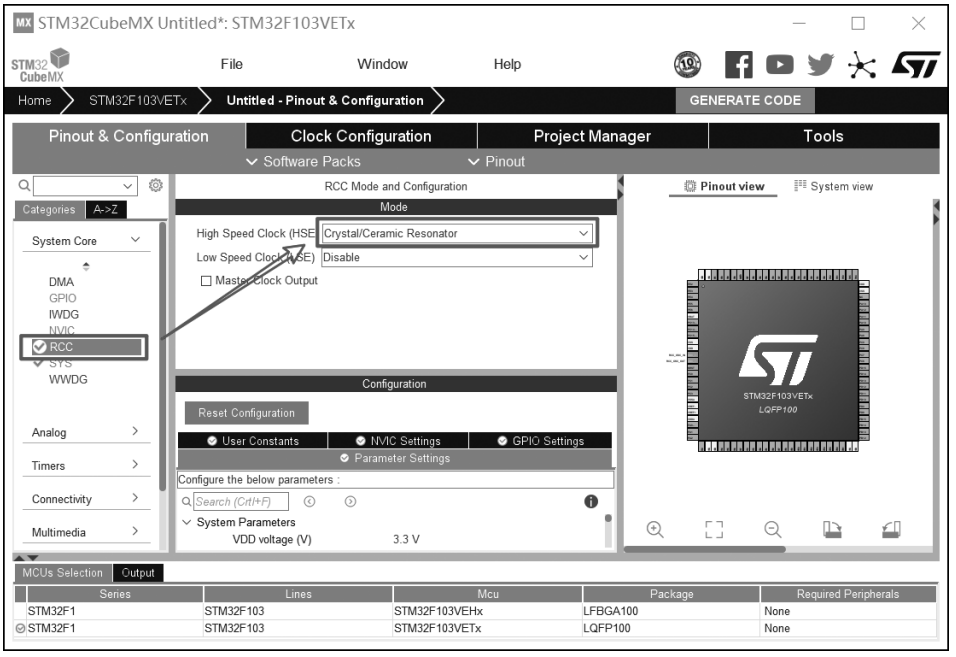

图 3-8　选择时钟源

单击 Clock Configuration 进入时钟配置界面，如图 3-9 所示，开发板上外部晶振为 8MHz，因此在输入框 Input frequency 填入 8，HSE Prediv 值选择/1，通道选择外接 HSE，倍频系数 PLLMul 选择×9；系统时钟选择由 PLLCLK 输入，设定为 72MHz；APB1 分频系数选择/2，即 PCLK1 为 36MHz；APB2 分频系数选择/1，即 PCLK2 为 72MHz。

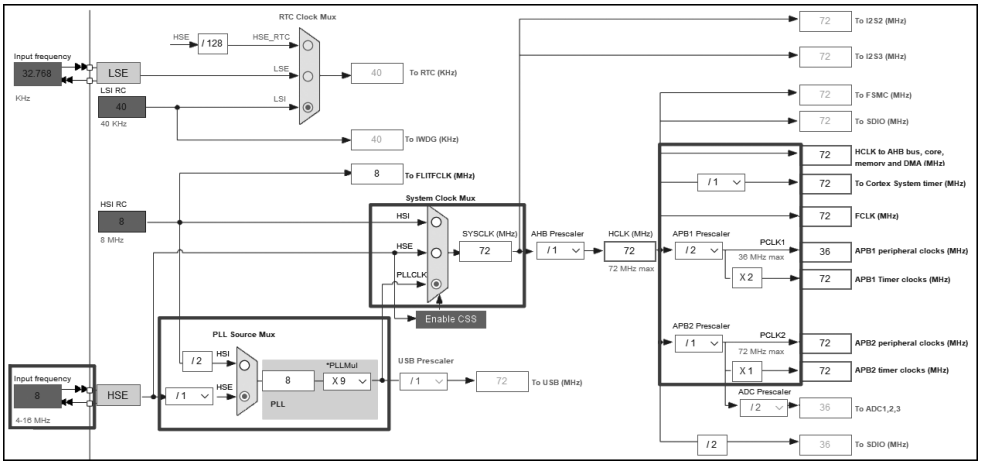

图 3-9　时钟配置界面

为简化上面的配置过程，也可以在软件中将 HCLK 的值直接修改为 72 后，按 Enter 键，软件会自动更改所有配置。

3. 配置 I/O 口

本示例使用两个按键来控制 LED1 和 LED2 的亮与灭，与它们连接的引脚为 PC0 和 PC1。如图 3-10 所示，单击 Pinout & Configuration，选择 System Core，再选择 GPIO，在图 3-10①处的搜索框输入 PC0 进行搜索可以定位引脚的位置，搜索到的引脚会闪烁提示，配置 PC0 的属性为 GPIO_Output。

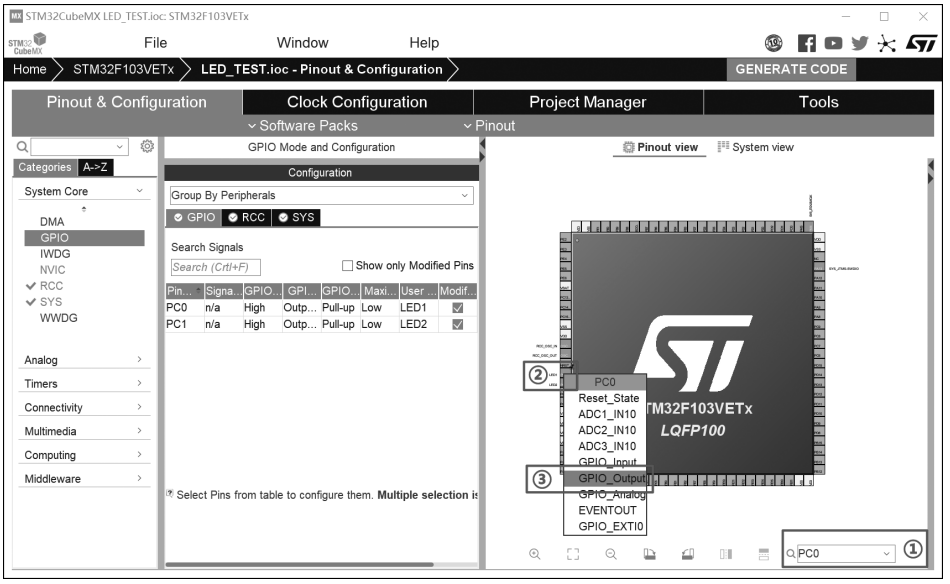

图 3-10　选择引脚 PC0 并配置其属性

配置引脚 PC0、PC1 为高电平(High,初始输出电平)、推挽输出模式(Output Push Pull)、带上拉电阻(Pull-up)及低速(Low)模式,并将用户标签设置为 LED1 和 LED2。配置引脚 PC0(LED1)属性示意图如图 3-11 所示。

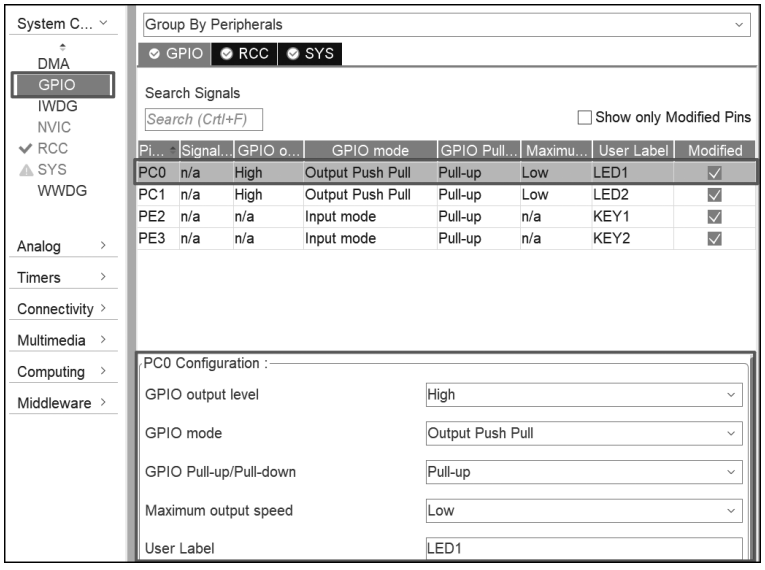

图 3-11　配置引脚 PC0(LED1)属性示意图

配置和按键 KEY1、KEY2 相连接的引脚 PE2、PE3 为输入模式(Input mode)、带上拉电阻(Pull-up),并将用户标签设置为 KEY1 和 KEY2。配置引脚 PE2(KEY1)属性示意图如图 3-12 所示。

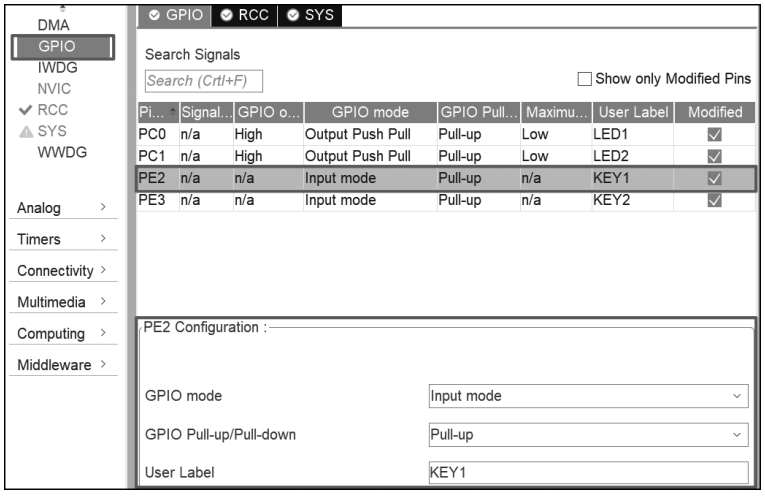

图 3-12　配置引脚 PE2(KEY1)属性示意图

为了防止出现烧录代码以后仿真器无法连接的情况,将选项 Pinout & Configuration 中的 SYS Debug 属性配置为 Serial Wire,如图 3-13 所示。

4. 工程管理配置

如图 3-14 所示,在主视窗中选择 Project Manager 选项,然后选择 Project 标签,配置工

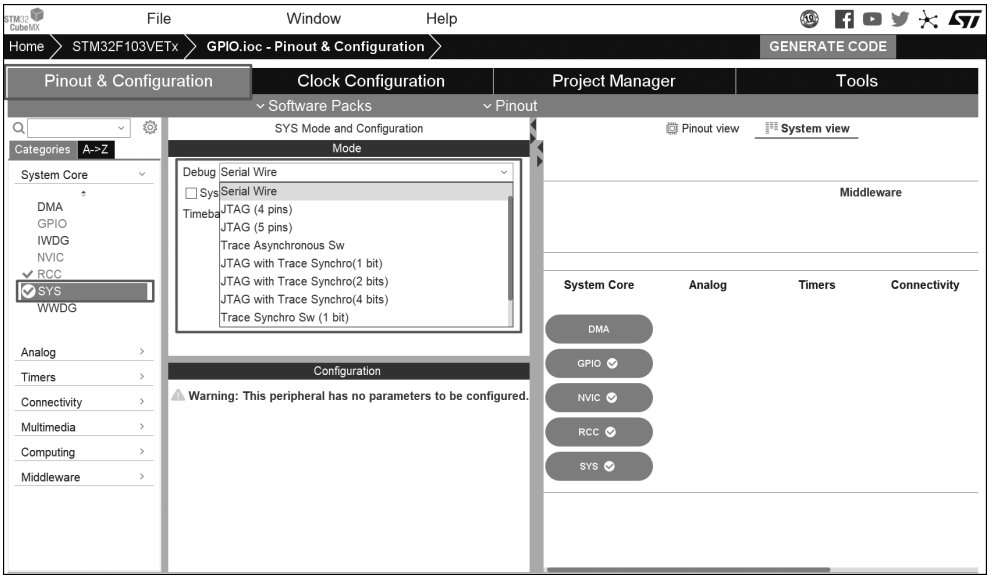

图 3-13　配置 SYS Debug 属性

程的名称、保存路径、使用的 IDE 工具及堆栈大小，注意不要使用中文路径和中文工程名称，以避免配置时出现一些奇怪的错误。在此标签下，也可以选择将 HAL 库存放在默认位置或指定路径下。

图 3-14　工程管理配置

如图 3-15 所示，选择 Code Generator 标签，设置代码生成选项。若在图中选择 Keep User Code when re-generating 功能，当重新生成工程时，工程代码中所有在英文注释 USER CODE BEGIN 和 USER CODE END 之间添加的用户代码会被保留，不会被 STM32CubeMX 删除。一般 STM32CubeMX 工程都会选择此功能。

5. 生成工程代码

完成上述配置后，单击主界面右上角的 GENERATE CODE（见图 3-13）按钮创建

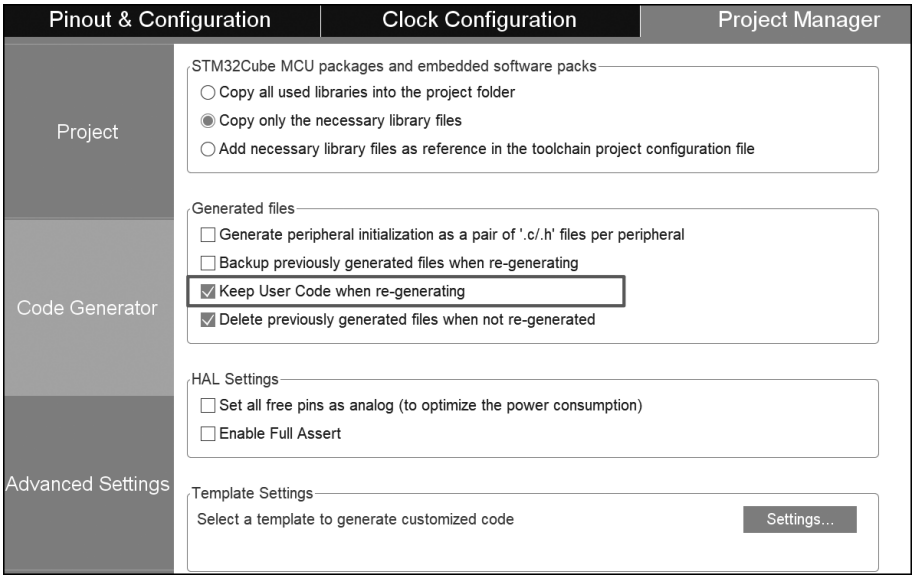

图 3-15　用户代码保留功能设置

MDK 工程。工程文件结构如图 3-16 所示，工程已经创建了组(Groups)并将 HAL 库的相关文件添加到了对应的组中。Application/User 组中的三个文件 main.c、stm32f1xx_it.c 和 stm32f1xx_hal_msp.c 是用户编写代码的主战场，Drivers/STM32F1xx_HAL_Driver 组中添加了外设模块的驱动文件，文件名称为 stm32f1xx_hal_ppp.c 或 stm32f1xx_hal_ppp_ex.c。

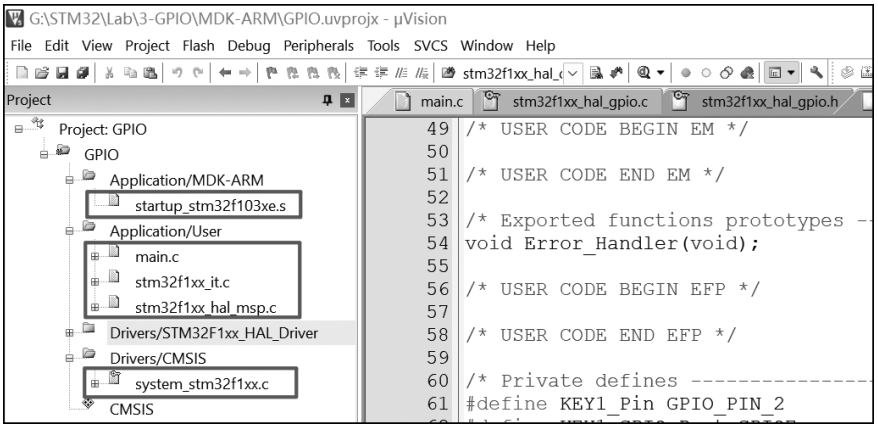

图 3-16　工程文件结构

使用 HAL 库构建的 STM32 应用程序，一般包含如表 3-5 所示的这些文件，表中列出了它们的作用。

表 3-5　STM32 应用程序包含的文件及其作用

文　件	作　用
main.c/.h	主程序文件
stm32f1xx_it.c/.h	中断和异常处理文件

续表

文　　件	作　　用
stm32f1xx_hal_msp.c	根据具体的 MCU 型号，对片上外设进行初始化设置
system_stm32f1xx.c	包含 STM32 上电后初始化系统时钟、扩展外部存储器用的函数
startup_stm32f103xe.s	启动文件，包含复位处理程序和异常向量
stm32f1xx_hal_conf.h	允许用户为特定的应用程序定制 HAL 驱动，通常直接使用默认配置

3.4.3　main 文件解析

STM32CubeMX 生成的工程文件中，比较重要的是 main.c 文件，下面列出其完整代码。

```
#include "main.h"

/* 私有函数声明 ---------------------------- */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);

int main(void)
{
    HAL_Init();                                  /* 初始化 HAL 库 */
    SystemClock_Config();                        /* 配置系统时钟 */
    MX_GPIO_Init();                              /* GPIO 配置 */

    /* USER CODE BEGIN WHILE */
    while (1)
    {
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
        if (HAL_GPIO_ReadPin(KEY1_GPIO_Port, KEY1_Pin) == 0) /* 按键 1 按下 */
        {
            /* 点亮 LED1 */
            HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
            /* 熄灭 LED2 */
            HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_SET);
        }

        if (HAL_GPIO_ReadPin(KEY2_GPIO_Port, KEY2_Pin) == 0) /* 按键 2 按下 */
        {
            /* 熄灭 LED1 */
            HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);
            /* 点亮 LED2 */
            HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);
        }

        HAL_Delay(10);                           /* 延时 10ms */
```

```
    }
    /* USER CODE END 3 */
}

/* 系统时钟配置 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /* 根据 RCC_OscInitTypeDef 结构中的指定参数初始化 RCC 振荡器 */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
    RCC_OscInitStruct.HSIState = RCC_HSI_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    /* 初始化 MCU、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;
    RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
    {
        Error_Handler();
    }
}

/* GPIO 初始化函数 */
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* GPIO 端口时钟使能 */
    __HAL_RCC_GPIOE_CLK_ENABLE();
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();

    /* 初始时 LED1、LED2 均熄灭 */
    HAL_GPIO_WritePin(GPIOC, LED1_Pin | LED2_Pin, GPIO_PIN_SET);
```

```
    /* 配置引脚: KEY1_Pin、KEY2_Pin */
    GPIO_InitStruct.Pin = KEY1_Pin | KEY2_Pin;          /* 选择引脚 PE2、PE3 */
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;             /* 输入模式 */
    GPIO_InitStruct.Pull = GPIO_PULLUP;                 /* 带上拉电阻 */
    HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);             /* 初始化 PE2、PE3 引脚 */

    /* 配置引脚: LED1_Pin、LED2_Pin */
    GPIO_InitStruct.Pin = LED1_Pin | LED2_Pin;          /* 选择引脚 PC0、PC1 */
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;         /* 推挽输出模式 */
    GPIO_InitStruct.Pull = GPIO_PULLUP;                 /* 带上拉电阻 */
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;        /* 低速 */
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);             /* 初始化 PC0、PC1 引脚 */
}

/* 错误处理函数 */
void Error_Handler(void)
{
    __disable_irq();
    while (1)
    {}
}

#ifdef  USE_FULL_ASSERT
/* 报告 assert_param 参数对应的错误发生的文件和代码行 */
void assert_failed(uint8_t * file, uint32_t line)
{
    /* ex: printf("Wrong parameters value: file %s on line %d\r\n",
                  file, line) */
}
#endif /* USE_FULL_ASSERT */
```

上面的大部分代码都是由 STM32CubeMX 软件生成的，下面对该文件中涉及的几个 HAL 库函数进行说明。

1. HAL 库初始化函数 HAL_Init

函数 HAL_Init 说明如表 3-6 所示。

表 3-6　函数 HAL_Init 说明

函数原型	HAL_StatusTypeDef　HAL_Init(void)
功能描述	初始化 HAL 库，重置所有外设，初始化 Flash 接口和 SysTick 等
返回值	HAL_OK
注意事项	该函数必须在主函数中最先执行
应用示例	HAL_Init();

2. 系统时钟配置函数 SystemClock_Config

开发板使用频率为 8MHz 的外部晶振，因此在文件 stm32f1xx_hal_conf.h 中设定 HSE_VALUE 的值为 8000000U，以此来匹配实际晶振的频率，配置的宏定义如下：

```
#if !defined  (HSE_VALUE)
    #define HSE_VALUE    8000000U            /* 外部晶振频率,Hz */
#endif /* HSE_VALUE */
```

每次系统上电时,启动文件中的复位中断服务函数(Reset handler)会调用 HAL 库函数 SystemInit,但该函数并没有像标准库中的同名函数一样初始化时钟配置,所以使用 HAL 库时,必须在主函数 main 中调用函数 SystemClock_Config 完成时钟的初始化,设置芯片的工作频率为 72MHz。

3. GPIO 配置函数 MX_GPIO_Init

考虑到硬件可能会发生更改的情况,例如 LED 的控制引脚发生了改变,希望用户程序只需要做最少的修改便可在新的硬件环境下正常运行,STM32CubeMX 在头文件 main.h 中把与 LED 和 KEY 相关的 GPIO 引脚和端口号使用宏进行了定义:

```
#define KEY1_Pin GPIO_PIN_2
#define KEY1_GPIO_Port GPIOE
#define KEY2_Pin GPIO_PIN_3
#define KEY2_GPIO_Port GPIOE

#define LED1_Pin GPIO_PIN_0
#define LED1_GPIO_Port GPIOC
#define LED2_Pin GPIO_PIN_1
#define LED2_GPIO_Port GPIOC
```

函数 MX_GPIO_Init 用来配置与 LED1、LED2 和 KEY1、KEY2 相连接的控制引脚。该函数首先使能 GPIO 端口时钟,然后通过调用函数 HAL_GPIO_WritePin 让引脚输出高电平,以确保在芯片上电时,LED1 和 LED2 都是熄灭状态。根据引脚的硬件电路,对 GPIO_InitTypeDef 结构体类型变量赋值,再调用函数 HAL_GPIO_Init 初始化 GPIO。

4. 用户代码

一般将用户代码添加在类似/* USER CODE BEGIN 3 */和/* USER CODE END 3 */之间,避免再次使用 STM32CubeMX 生成工程代码时,添加的用户代码被软件删除。

本例添加的用户代码在 while(1)无限循环中判断引脚 PE2、PE3 的电平状态,若检测到低电平,则表示与该引脚相连的按键被按下,再控制 LED1 和 LED2 的亮或灭。延时函数 HAL_Delay 实现间隔 10ms 检测一次按键的状态。

3.4.4　编译下载

工程编译成功后,如图 3-17 所示,编译结果显示,0 错误和 0 警告,代码占用 Flash 的大小为 3052(2700+352)字节,所用 SRAM 的大小为 1040(16+1024)字节。

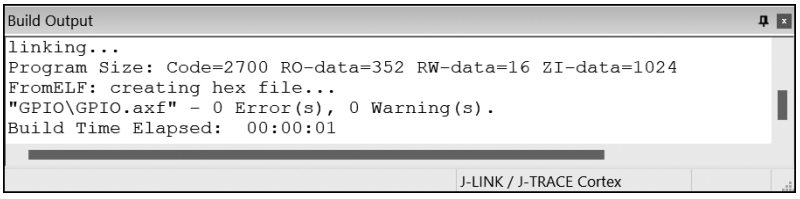

图 3-17　代码编译结果

按照2.5.4节介绍的方法配置下载工具J-LINK，将程序下载到开发板运行。初始时LED1、LED2都处于熄灭状态，若按下KEY1按键，LED1点亮，LED2熄灭；若按下KEY2按键，LED2点亮，LED1熄灭。

本应用示例演示了使用STM32CubeMX软件配置STM32工程的完整过程，在实际操作过程中，读者可以参考相关资料反复练习，以达到熟练使用该工具的目的。

练习题

1. 说明STM32 HAL库文件的组织结构，以及STM32工程必须包含哪些文件。

2. 说明结构体GPIO_InitTypeDef的四个成员的作用，以及怎样使用该结构体。

3. 使用STM32CubeMX配置一个工程，实现PC端口的8个LED进行流水灯闪烁(各个LED先从左至右，再从右至左依次点亮)。

4. 使用HAL库函数编程实现：按键KEY1按下，LED1以1s为周期闪烁；按键KEY2按下，LED1停止闪烁。