第 3 章

CHAPTER 3

通用输入/输出模块

与 51 单片机不同，STM32 处理器拥有更多的输入/输出引脚，其对外围设备的驱动能力更强，包括更多更灵活的外围设备控制方式，更多更强大的外围设备功能。在使用 STM32 处理器这些功能之前，必须对其进行正确配置。本章所介绍的通用输入/输出(General Purpose Input/Output，GPIO)是 STM32 处理器中非常重要的片内外围设备，是实现外围设备数据输入到 STM32 处理器或 STM32 处理器输出数据到外围设备的关键。更重要的是，本章的内容是学习后面章节的基础。本章首先介绍 GPIO 的基本概念及其工作原理，然后以 STM32F103 为例讲述 GPIO 的编程开发，最后通过一个开发案例说明如何使用 STM32F103 的 GPIO。

学习目标

- 掌握输入/输出模块的有关概念；
- 掌握 STM32 的 GPIO；
- 了解 STM32 的 GPIO 库函数；
- 掌握 STM32 的 GPIO 开发流程；
- 熟练使用 STM32CubeMX 工具开发 GPIO 项目；
- 熟练应用 Keil 软件编译环境和仿真环境；
- 了解 Proteus 仿真环境。

3.1 输入/输出

输入/输出(Input/Output，I/O)是指相对微控制器而言作为外围设备的输入或输出的标准双向输入/输出接口，例如采集传感器输入的数据、是否有按键产生、点亮 LED 灯等。一般按照不同的工作模式，可进一步将微控制器芯片的 I/O 分为准双向 I/O、推挽输出、高阻态、开漏。

准双向 I/O 模式是指该引脚既可以接收来自外围设备的输入信号，又可以输出信号给外围设备。

推挽输出是指利用相对来说比较大的电流对外围电路设备进行驱动，在该模式输出高电平或低电平。

高阻态是指输入/输出的电阻非常大，该模式又分为高阻输入或高阻输出。其中，高阻输入是指以较高的输入阻抗来获取外围设备的输入信号，例如 A/D 采集数据作为输入；高

阻输出是指既不输出高电平也不输出低电平。

开漏与准双向 I/O 模式相近，该模式既可以读取 I/O 接口的输入电平，又可以输出高电平和低电平。如果作输出，默认输出低电平，如需要输出高电平，则需根据具体的应用选择适合的上拉电阻，并搭建外围电路来提高 I/O 接口的驱动能力。

3.2　STM32 的 GPIO

通用输入/输出(General Purpose Input/Output，GPIO)是指 STM32 微控制器片内外设中可配置的输入/输出接口。通过对 GPIO 引脚进行必要的功能初始化就可实现与外部设备连接的功能，从而达到 STM32 微控制器与外部设备通信来控制设备以及采集数据的目的。GPIO 的引脚对应了不同功能的寄存器，配置 STM32 的 GPIO 实际上是更改引脚所对应寄存器的值，以便实现某种输入/输出功能。表 3-1 列出了 STM32F10xxx 系列芯片内置外设的起始地址。

表 3-1　寄存器组起始地址

起 始 地 址	外　　设	总　　线
0x5000 0000 - 0x5003 FFFF	USB OTG 全速	
0x4003 0000 - 0x4FFF FFFF	保留	AHB
0x4002 8000 - 0x4002 9FFF	以太网	
0x4002 3400 - 0x4002 3FFF	保留	
0x4002 3000 - 0x4002 33FF	CRC	
0x4002 2000 - 0x4002 23FF	闪存存储器接口	
0x4002 1400 - 0x4002 1FFF	保留	
0x4002 1000 - 0x4002 13FF	复位和时钟控制(RCC)	AHB
0x4002 0800 - 0x4002 0FFF	保留	
0x4002 0400 - 0x4002 07FF	DMA2	
0x4002 0000 - 0x4002 03FF	DMA1	
0x4001 8400 - 0x4001 7FFF	保留	
0x4001 8000 - 0x4001 83FF	SDIO	
0x4001 4000 - 0x4001 7FFF	保留	
0x4001 3C00 - 0x4001 3FFF	ADC3	
0x4001 3800 - 0x4001 3BFF	USART1	
0x4001 3400 - 0x4001 37FF	TIM8 定时器	
0x4001 3000 - 0x4001 33FF	SPI1	
0x4001 2C00 - 0x4001 2FFF	TIM1 定时器	
0x4001 2800 - 0x4001 2BFF	ADC2	
0x4001 2400 - 0x4001 27FF	ADC1	
0x4001 2000 - 0x4001 23FF	GPIO 端口 G	APB2
0x4001 2000 - 0x4001 23FF	GPIO 端口 F	
0x4001 1800 - 0x4001 1BFF	GPIO 端口 E	
0x4001 1400 - 0x4001 17FF	GPIO 端口 D	
0x4001 1000 - 0x4001 13FF	GPIO 端口 C	
0x4001 0C00 - 0x4001 0FFF	GPIO 端口 B	
0x4001 0800 - 0x4001 0BFF	GPIO 端口 A	
0x4001 0400 - 0x4001 07FF	EXTI	
0x4001 0000 - 0x4001 03FF	AFIO	

续表

起始地址	外设	总线
0x4000 7800 - 0x4000 FFFF	保留	
0x4000 7400 - 0x4000 77FF	DAC	
0x4000 7000 - 0x4000 73FF	电源控制(PWR)	
0x4000 6C00 - 0x4000 6FFF	后备寄存器(BKP)	
0x4000 6800 - 0x4000 6BFF	bxCAN2	
0x4000 6400 - 0x4000 67FF	bxCAN1	
0x4000 6000 - 0x4000 63FF	USB/CAN 共享的 512 字节 SRAM	
0x4000 5C00 - 0x4000 5FFF	USB 全速设备寄存器	
0x4000 5800 - 0x4000 5BFF	I2C2	
0x4000 5400 - 0x4000 57FF	I2C1	
0x4000 5000 - 0x4000 53FF	UART5	
0x4000 4C00 - 0x4000 4FFF	UART4	
0x4000 4800 - 0x4000 4BFF	USART3	
0x4000 4400 - 0x4000 47FF	USART2	APB1
0x4000 4000 - 0x4000 3FFF	保留	
0x4000 3C00 - 0x4000 3FFF	SPI3/I2S3	
0x4000 3800 - 0x4000 3BFF	SPI2/I2S3	
0x4000 3400 - 0x4000 37FF	保留	
0x4000 3000 - 0x4000 33FF	独立看门狗(IWDG)	
0x4000 2C00 - 0x4000 2FFF	窗口看门狗(WWDG)	
0x4000 2800 - 0x4000 2BFF	RTC	
0x4000 1800 - 0x4000 27FF	保留	
0x4000 1400 - 0x4000 17FF	TIM7 定时器	
0x4000 1000 - 0x4000 13FF	TIM6 定时器	
0x4000 0C00 - 0x4000 0FFF	TIM5 定时器	
0x4000 0800 - 0x4000 0BFF	TIM4 定时器	
0x4000 0400 - 0x4000 07FF	TIM3 定时器	
0x4000 0000 - 0x4000 03FF	TIM2 定时器	

每个 GPIO 引脚都有两个 32 位配置寄存器 GPIOx_CRL 和 GPIOx_CRH，两个 32 位数据寄存器 GPIOx_IDR 和 GPIOx_ODR，一个 32 位置位/复位寄存器 GPIOx_BSRR，一个 16 位复位寄存器 GPIOx_BRR 和一个 32 位锁定寄存器 GPIOx_LCKR。根据 STM32 芯片数据手册中列出的每个 I/O 引脚的特定功能，GPIO 的每个引脚可以由软件配置成多种功能模式。借助 GPIO 的多种配置模式，STM32 可以实现最简单、最直观的动作，例如，检测按键信号、LED 的开关等。另外，STM32 的 GPIO 还可用于串行和并行通信、存储器读/写等功能，可以有效避免某些嵌入式应用 GPIO 引脚不足或片内存储器存储空间不足等问题。

由于 STM32 的 GPIO 引脚数目过多，为便于使用往往将这些引脚进行分组(定义为端口)，每个端口都有 16 个输入/输出引脚，分别由 0～15 个标号表示。端口通常以大写英文字母 A 表示开始，16 个引脚为一组，以此类推。例如，GPIOA、GPIOB、GPIOC、GPIOD、GPIOE、GPIOF 和 GPIOG 等。又如，GPIOB 端口共有 16 个引脚，分别标记为 PB0～PB15，也

可以表示为 GPIOB0～GPIOB15。图 3-1 为 48 引脚 LQFP 封装的 STM32F103C8Tx 的引脚示意图。

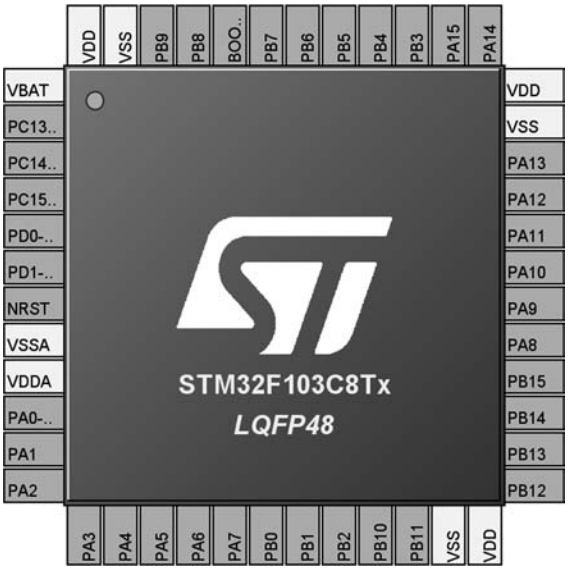

图 3-1　LQFP 封装的 STM32F103C8Tx

图 3-2 给出了 APB2 时钟 32 位使能寄存器每位的功能，其中寄存器的 2～8 位用来配置 GPIOx 端口的使能位。与 51 单片机不同，STM32 的 GPIOx 端口可以不同时工作，必须通过配置寄存器的控制位使能 GPIOx 端口，这有利于降低 STM32 的能耗。例如，仅 GPIOA 工作而其余端口不工作，则 APB2 时钟使能寄存器的控制位可以设定位为 0x04。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved													TIM17	TIM16	TIM15
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	URT1	Res.	SPI1	TIM1	Res.	ADC1	IOPG	IOPF	IOPE	IOPD	IOPC	IOPB	IOPA	Res.	AFIO

图 3-2　APB2 时钟使能寄存器

STM32 的大多数引脚可以配置为多功能双向的输入/输出，可以连接到 GPIO 端口或复选功能(Alternative Functions，AF)。STM32 的大多数引脚通过 AF 技术兼具其他专用功能。作为一个标准的命名约定，如果使用库函数进行程序开发，STM32 的引脚已经在 stm32f10x_gpio.h 头文件中定义，例如，PA8 表示端口 A 的第 8 位，PE1 表示端口 E 的第 1 位。STM32 的 GPIO 在工作时通常有三种不同的状态，分别是输入态、输出态和高阻态。输入态是指 GPIO 引脚被配置为输入模式，可以获取来自外围设备的高低电平(高电平为 1，低电平为 0)。输出态是指 GPIO 引脚配置为主动向外围设备输出高低电平，可以控制外围设备进行相应动作。高阻态是一种特殊的状态，是指 GPIO 引脚内部电阻的阻值无穷大，它的输出既不是高电平也不是低电平，对与其关联的输出信号不产生影响。由于这三种状态的作用有着很大的不同，需要根据实际开发的嵌入式应用的需求来配置相应的状态。

STM32 的 GPIO 引脚的基本结构如图 3-1 所示。与 51 单片机不同，STM32 的每个

GPIO 引脚可按功能需求灵活配置，这个配置过程是通过端口配置低位寄存器 GPIOx_CRL、端口配置高位寄存器 GPIOx_CRH、端口输入数据寄存器 GPIOx_IDR、端口输出数据寄存器 GPIOx_ODR、端口位清除寄存器 GPIOx_BSRR、端口位清除寄存器 GPIOx_BRR 和端口配置锁定寄存器 GPIOx_LCKR 共 7 个寄存器来实现的。为了使用方便，STM32 的标准库函数头文件 stm32f10x_gpio.h 通过结构体的形式定义了这 7 个寄存器，利用该结构体可以灵活配置 GPIO 引脚功能，具体代码如下：

```
typedef struct
{
volatile uint32_t CRL;
volatile uint32_t CRH;
volatile uint32_t IDR;
volatile uint32_t ODR;
volatile uint32_t BSRR;
volatile uint32_t BRR;
volatile uint32_t LCKR;
}GPIO_TypeDef;
```

需要说明的是，这些寄存器必须按 32 位字被访问。由于 STM32 的供电电压是 3.3V，但很多外围设备的输入电压是 5V，为兼容 5V 的外围设备输入，STM32 的 GPIO 引脚大部分是兼容 5V 电压输入的，具体兼容 5V 的 GPIO 引脚可以从该芯片的数据手册引脚描述章节查到(I/O Level 标有 V_{DD_FT} 的就是 5V 电平兼容的)。按照不同的输入/输出要求，可以对 GPIO 引脚的寄存器进行配置，同时也可以利用 stm32f10x_gpio.h 的库常量配置 GPIO 引脚，具体如表 3-2 所示。

表 3-2 GPIO 引脚配置

配置模式	功能	库常量
通用输入	浮空输入	GPIO_Mode_IN_FLOATING
	上拉输入	GPIO_Mode_IPU
	下拉输入	GPIO_Mode_IPD
	模拟输入	GPIO_Mode_AIN
通用输出	推挽式输出	GPIO_Mode_Out_PP
	开漏输出	GPIO_Mode_Out_OD
复用功能输出	推挽式复用功能	GPIO_Mode_AF_PP
	开漏复用功能	GPIO_Mode_AF_OD

STM32 的标准库函数头文件 stm32f10x_gpio.h 定义了上述 8 种引脚的配置模式，头文件中以枚举类型给出了各个模式在芯片中的地址，具体代码如下：

```
typedef enum
{
  GPIO_Mode_AIN = 0x0,
  GPIO_Mode_IN_FLOATING = 0x04,
  GPIO_Mode_IPD = 0x28,
  GPIO_Mode_IPU = 0x48,
  GPIO_Mode_Out_OD = 0x14,
```

```
  GPIO_Mode_Out_PP = 0x10,
  GPIO_Mode_AF_OD = 0x1C,
  GPIO_Mode_AF_PP = 0x18
}GPIOMode_TypeDef;
```

图 3-3 是 GPIO 引脚内部结构示意图。该图内部所示的输入驱动器中所有 GPIO 引脚都有一个内部弱上拉和弱下拉，当 GPIO 引脚配置为输入时，首先输出驱动器被禁止；其次图中的 TTL 肖特基触发器输入被激活；然后根据 GPIO 引脚的配置（浮空输入、上拉输入或下拉输入）的不同，弱上拉和下拉电阻将被连接；出现在 GPIO 引脚上的数据在每个 APB2 时钟周期被采样到输入数据寄存器；最后 STM32 内部处理器对输入数据寄存器执行读访问，便可得到 GPIO 引脚输入的高低电平信号。

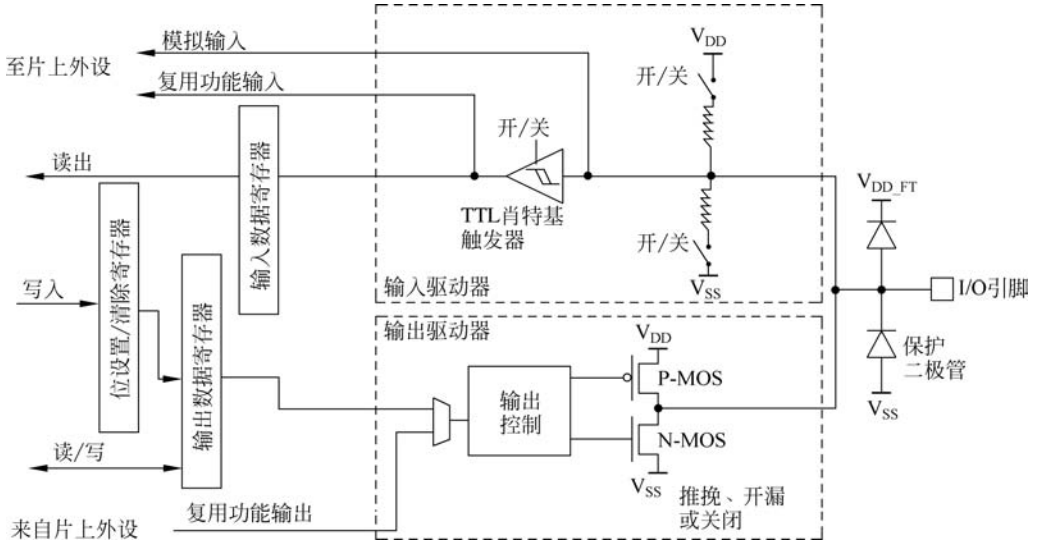

图 3-3　GPIO 引脚内部结构

GPIO 引脚输入/输出模式具体如下：

（1）浮空输入（Input Floating）：该模式下 GPIO 引脚内部既不连上拉电阻又不连下拉电阻，而是直接经 TTL 肖特基触发器输入 GPIO 引脚的高低电平信号保存到输入数据寄存器。

（2）上拉输入（Input Pull-up）：该模式下 GPIO 引脚通过开关将电阻连接到电源 V_{DD}。当 GPIO 引脚有输入信号时，输入信号电平保存到输入数据寄存器。该端口在默认情况下输入为高电平。

（3）下拉输入（Input Pull-down）：该模式下 GPIO 引脚通过开关将电阻连接到电源 V_{SS}。一旦芯片的 GPIO 引脚接收到来自外部的输入信号，芯片会将外部的输入信号电平保存在与引脚对应的输入数据寄存器中。该端口在默认情况下输入为低电平。

（4）模拟输入（Analog Input）：该模式下 TTL 肖特基触发器输入关闭，既不接上拉电阻也不连接下拉电阻，引脚信号连接到芯片内部的片上外设，其典型应用是 A/D 模拟输入，对外部模拟信号进行采集。

默认情况下，大多数引脚被重置为浮空输入，这确保当系统通电时不会发生硬件冲突。

例如，将按钮一端连接到 PA1 引脚，利用库函数判断按钮是否被按下，具体如下：

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
GPIO_Init(GPIOA,&GPIO_InitStructure);
```

例如，利用 GPIO 库函数读取 PA1 引脚的高低电平信息，代码如下：

```
GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_1);
```

如图 3-3 所示的输出驱动器部分，当 GPIO 引脚配置为输出时，首先 STM32 内部处理器位设置/清除寄存器输出高低电平驱动信号给输出数据寄存器；然后利用输出控制逻辑使能内部的 PMOS 或 NMOS；最后由 GPIO 引脚输出相应信号。

(1) 推挽式输出(Output Push-Pull)：该模式下 GPIO 引脚输出高电平时，则输出控制逻辑使能 P-MOS；相反，则输出控制逻辑使能 N-MOS。由于使用了 MOSFET 管，增加了 GPIO 引脚的输出电流，进而有利于驱动负载大的外围设备。

(2) 开漏输出(Output Open-Drain)：该模式下 GPIO 引脚直接与 MOSFET 管的漏极相连，处于悬空状态。此时，如外围电路中无上拉电阻时，GPIO 引脚输出低电平。只有在 GPIO 引脚的外围电路中加上拉电阻才能输出高电平。

(3) 推挽式复用功能(Alternate Function Push-Pull)：该模式下 GPIO 引脚可以作为多个外设引脚使用，但一个引脚某一时刻只能使用复用功能中的一个，由片上外设进行控制。

(4) 开漏复用功能(Alternate Function Open-Drain)：该模式下与推挽式复用功能相近，但想要输出高电平需要接上拉电阻。

当配置如上所示的输出时还应考虑驱动电路的响应速度，每个 GPIO 引脚有 3 种输出速度可供选择，即 50MHz、10MHz 和 2MHz。一般来讲，实际开发中出于信号稳定性和降低功耗等原因，需要结合系统实际情况配置 GPIO 的响应速度，尽量使用与 GPIO 引脚要求一致的最低速度。一般常用的外设建议采用 2MHz 的输出速度，例如 LED、蜂鸣器等；而片上外设 I^2C、SPI 等使用复用功能输出时，应配置高响应速度 10MHz，甚至是 50MHz。

在 STM32 的标准固件库中，stm32f10x_gpio.h 头文件定义了 3 种输出速度模式，具体如下：

```
typedef enum
{
  GPIO_Speed_10MHz = 1,
  GPIO_Speed_2MHz,
  GPIO_Speed_50MHz
}GPIOSpeed_TypeDef;
```

如果利用固件库配置 GPIO 引脚，则首先选定 GPIO 引脚，指定选择引脚的响应速度，并且需要说明引脚的功能，固件库中 stm32f10x_gpio.h 头文件给出了引脚的定义，具体如下：

```
typedef struct
{
  uint16_t GPIO_Pin;                    /* 指定要配置的 GPIO 引脚 */
```

```
  GPIOSpeed_TypeDef GPIO_Speed;       /* 指定所选引脚的速度 */
  GPIOMode_TypeDef GPIO_Mode;         /* 指定所选引脚的模式 */
}GPIO_InitTypeDef;
```

例如，对于闪烁灯实验而言，利用库文件配置 PB8 为 2MHz 输出，具体如下：

```
//stm32f10x_gpio.h
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_StructInit(&GPIO_InitStructure);
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_Init(GPIOB , &GPIO_InitStructure);
```

GPIO 库函数提供了读写单个引脚和整个端口的操作，对于 GPIO 端口在捕获并行数据时特别有用。

例如，若下面利用 GPIO 库函数 GPIO_WriteBit 对 PB8 引脚输出高低电平进行设置，若通过 PB8 引脚的高低电平变化可以控制闪烁灯开关。如果控制 PB8 为高电平，调用库函数 GPIO_WriteBit(GPIOB,GPIO_Pin_8,Bit_SET)来设置 GPIOB 端口的第 8 个引脚为高电平；相反，调用库函数 GPIO_WriteBit(GPIOB,GPIO_Pin_8,Bit_RESET)来设置端口的第 8 个引脚为低电平。

STM32 中 USART 之类的外围设备需要与 GPIO 复用的引脚，即共用同一个引脚。在使用这些外设之前，外设需要的任何输出必须配置为复用输出功能。

例如，USART1 的 Tx 与 PA9 有相同的引脚，如果使能 Tx，则需要配置 PA9 的输出模式为推挽式复用功能，具体如下：

```
GPIO_InitStruct.GPIO_PIN = GPIO_Pin_9;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_PP;
GPIO_Init(GPIOA , &GPIO_InitStruct);
```

3.3　STM32 的 GPIO 库函数

3.3.1　GPIO 模块的标准库函数

ST 公司为了便于嵌入式应用系统的实现，向使用者提供了 GPIO 模块的标准外设库接口函数。嵌入式开发者可以使用 ST 公司定义的函数对 GPIO 引脚进行配置和使用，从而规避了直接读写 GPIO 寄存器而引发错误的可能性。在实际项目开发中，往往需要用到标准外设库函数，这就要求创建工程时，把标准库的头文件 stm32f10x_gpio.h 加入工程。在项目开发过程中，想了解该头文件的功能时可以查看 GPIO 库函数的源码，该头文件所对应的源文件是 stm32f10x_gpio.c。文件 stm32f10x_gpio.h 声明了 GPIO 共 18 种库函数的定义，具体函数头定义如下：

```
void GPIO_DeInit(GPIO_TypeDef * GPIOx);
/**
 * @brief 该函数表示将 GPIOx 外围寄存器反初始化为它们的默认重置值。
```

```
 * @param GPIOx: x 可以是 A～G 来选择 GPIO 外围设备。
 * @retval 无
 */
void GPIO_AFIODeInit(void);
/**
 * @brief 该函数是将 AF(重新映射、事件控制和 EXTI 配置)寄存器反初始化为它们的默认重置值。
 * @retval 无
 */
void GPIO_Init(GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_InitStruct);
/**
 * @brief 该函数是根据 GPIO_InitStruct 中的指定参数初始化 GPIOx 外围设备。
 * @param GPIOx:指定 GPIO 的具体端口,x 可以是 A～G 其中一个端口。
 * @param GPIO_InitStruct 是指向 GPIO_InitTypeDef 结构的指针,包含指定的 GPIO 外围设备的配
 * 置信息。
 * @retval 无
 */
void GPIO_StructInit(GPIO_InitTypeDef * GPIO_InitStruct);
/**
 * @brief 该函数是用它的默认值填充每个 GPIO_InitStruct 成员。
 * @param GPIO_InitStruct: 指向将要初始化的 GPIO_InitTypeDef 结构的指针。
 * @retval 无
 */
uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin);
/**
 * @brief 该函数是读取指定的输入端口引脚。
 * @param GPIOx: x 可以是 A～G 来选择 GPIO 外围设备。
 * @param GPIO_Pin: 指定要读取的端口位。这个参数可以是 GPIO_Pin_x,其中 x 可以是 0～15。
 * @retval 输入端口引脚值。
 */
uint16_t GPIO_ReadInputData(GPIO_TypeDef * GPIOx);
/**
 * @brief 该函数是读取指定的 GPIO 输入数据端口。
 * @param GPIOx: x 可以是 A～G 来选择 GPIO 外围设备。
 * @retval GPIO 输入数据端口值。
 */
uint8_t GPIO_ReadOutputDataBit(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin);
/**
 * @brief 该函数是读取指定的输出端口引脚。
 * @param GPIOx: x 可以是 A～G 来选择 GPIO 外围设备。
 * @param GPIO_Pin: 指定要读取的端口位。这个参数可以是 GPIO_Pin_x,其中 x 可以是 0～15。
 * @retval 输出端口引脚值。
 */
uint16_t GPIO_ReadOutputData(GPIO_TypeDef * GPIOx);
/**
 * @brief 该函数是读取指定的 GPIO 输出数据引脚。
 * @param GPIOx: x 可以是 A～G 来选择 GPIO 外围设备。
 * @retval GPIO 输出数据端口值。
 */
void GPIO_SetBits(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin);
/**
 * @brief 该函数是设置选定的数据端口引脚。
```

```
 * @param GPIOx: x 可以是 A～G 来选择 GPIO 外围设备。
 * @param GPIO_Pin: 指定要写入的端口位。这个参数可以是 GPIO_Pin_x 的任意组合,其中 x 可以
 * 是 0～15。
 * @retval 无
 */
void GPIO_ResetBits(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin);
/**
 * @brief 该函数是设置选定的数据端口位。
 * @param GPIOx: x 可以是 A～G 来选择 GPIO 外围设备。
 * @param GPIO_Pin: 指定要写入的端口位。这个参数可以是 GPIO_Pin_x 的任意组合,其中 x 可以
 * 是 0～15。
 * @retval 无
 */
void GPIO_WriteBit(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, BitAction BitVal);
/**
 * @brief 该函数是设置或清除选定的数据端口位。
 * @param GPIOx: x 可以是 A～G 来选择 GPIO 外围设备。
 * @param GPIO_Pin: 指定要写入的端口位。这个参数可以是 GPIO_Pin_x,其中 x 可以是 0～15。
 * @param BitVal: 指定要写入所选位的值。该参数可以是 BitAction 枚举值中的一个:
 * @arg Bit_RESET: 清除端口引脚。
 * @arg Bit_SET: 设置端口引脚。
 * @retval 无
 */
void GPIO_Write(GPIO_TypeDef * GPIOx, uint16_t PortVal);
/**
 * @brief 该函数是将数据写入指定的 GPIO 数据端口。
 * @param GPIOx: x 可以是 A～G 来选择 GPIO 外围设备。
 * @param PortVal: 指定要写入端口输出数据寄存器的值。
 * @retval 无
 */
void GPIO_PinLockConfig(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin);
/**
 * @brief 该函数是 GPIO 引脚锁定配置寄存器。
 * @param GPIOx: x 可以是 A～G 来选择 GPIO 外围设备。
 * @param GPIO_Pin: 指定要写入的端口位。这个参数可以是 GPIO_Pin_x 的任意组合,其中 x 可以
 * 是 0～15。
 * @retval 无
 */
void GPIO_EventOutputConfig(uint8_t GPIO_PortSource, uint8_t GPIO_PinSource);
/**
 * @brief 该函数是选择用作事件输出的 GPIO 引脚。
 * @param GPIO_PortSource: 选择作为事件输出源的 GPIO 端口。这个参数可以是 GPIO_
 * PortSourceGPIOx,其中 x 可以是 A～G。
 * @param GPIO_PinSource: 指定事件输出的引脚。这个参数可以是 GPIO_PinSourcex,其中 x 可以
 * 是 0～15。
 * @retval 无
 */
void GPIO_EventOutputCmd(FunctionalState NewState);
/**
 * @brief 该函数是启用或禁用事件输出。
 * @param NewState: 事件输出的新状态。此参数可以是启用或禁用。
 * @retval 无
 */
```

```
void GPIO_PinRemapConfig(uint32_t GPIO_Remap, FunctionalState NewState);
/**
 * @brief 该函数是更改指定引脚的映射。
 * @param GPIO_Remap: 选择要重新映射的引脚。
 * @paramNewState: 重新映射端口引脚的新状态。这个参数可以是启用或禁用。
 * @retval 无
 */
void GPIO_EXTILineConfig(uint8_t GPIO_PortSource, uint8_t GPIO_PinSource);
/**
 * @brief 该函数是选择作为 EXTI 线的 GPIO 引脚。
 * @param GPIO_PortSource: 选择作为 EXTI 线源的 GPIO 端口。这个参数可以是 GPIO_
 * PortSourceGPIOx,其中 x 可以是 A～G。
 * @param GPIO_PinSource: 指定要配置的 EXTI 行。这个参数可以是 GPIO_PinSourcex,其中 x 可以
 * 是 0～15。
 * @retval 无
 */
void GPIO_ETH_MediaInterfaceConfig(uint32_t GPIO_ETH_MediaInterface);
/**
 * @brief 该函数是选择以太网媒体接口。
 * @note 这个函数只适用于 STM32 连接线路设备。
 * @param GPIO_ETH_MediaInterface 指定媒体接口模式。该参数可以是以下值之一:
 * @arg GPIO_ETH_MediaInterface_MII: MII 模式
 * @arg GPIO_ETH_MediaInterface_RMII: RMII 模式
 * @retval 无
 */
```

3.3.2 GPIO 配置步骤

不同于 51 单片机,在使用 GPIO 之前,必须对它们进行配置。一般来讲,使用任何 GPIO 引脚所需的配置如图 3-4 所示。

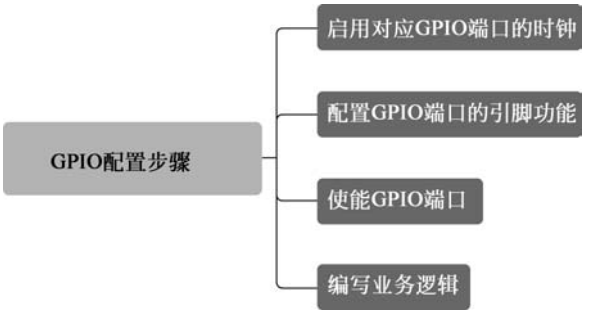

图 3-4 GPIO 配置步骤

基本初始化步骤如下:

(1) 启用相应外围设备的时钟;

(2) 配置外设功能参数,调用初始化函数,初始化外设相关的参数;

(3) 使能相应的 GPIO;

(4) 编写应用逻辑。

基于 STM32 的 GPIO 初始化程序的框架结构，如以下代码所示：

```
#include <stm32f10x.h>
#include <stm32f10x_rcc.h>
#include <stm32f10x_gpio.h>
int main(void) {
   GPIO_InitTypeDef GPIO_InitStructure
   //Enable Peripheral Clocks (1) ...
   //Configure Pins (2) ...
   //Enable the corresponding peripheral.... (3) ...
   while (1) {
   //Coding application logic... (4) ...
   }
}
```

3.4　STM32 GPIO 应用实例

3.4.1　实例标准库函数开发

本节采用基于标准固件库的设计方式，利用单个 GPIO 引脚输出高低电平控制发光二极管，并按一定时间间隔改变 I/O 口电平，达到 GPIO 的库函数实现灯光闪烁效果。图 3-5 是发光二极管 D1 与 STM32F103C6 的 PB0 接口电路的原理图。

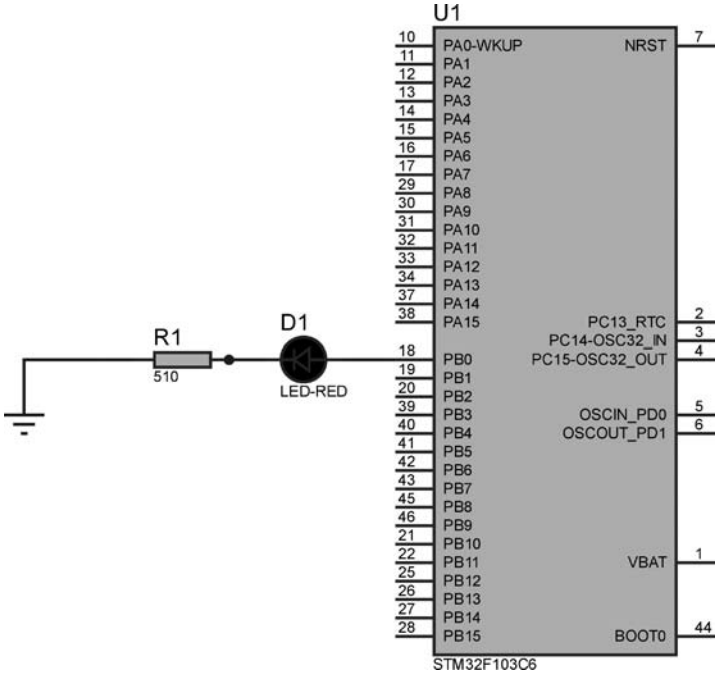

图 3-5　发光二极管与 STM32F103C6 的接口电路

图 3-5 的电阻 R1 是限流电阻。R1 阻值的改变会导致发光二极管 D1 的亮度也发生改变，R1 的阻值范围一般选用 400Ω～1kΩ 的。程序流程设计中首先需要配置 GPIO，然后主循环中不断检测发光二极管 D1 的状态，当检测到发光二极管 D1 开时，则 GPIO 的 PB0 输

出为高电平；相反，则输出低电平。图 3-6 为发光二极管闪烁程序流程图。

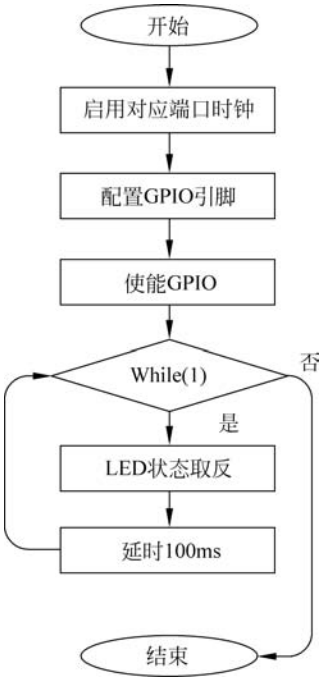

图 3-6　发光二极管闪烁程序流程

发光二极管工程文件中的源文件存放了 GPIO 的管脚定义、GPIO 初始化、全局变量声明、函数声明以及 LED 状态切换等功能，构成了发光二极管闪烁的主程序。具体代码如下：

```
#include "stm32f10x.h"
/*函数名:Delay
*功能描述:不精确的延时,延时时间 = nCount/72000000,72MHz 为 STM32 主频
*输入参数:nCount
*输出参数:无
*/
void Delay( u32 nCount)
{
      for(;nCount != 0;nCount--);
}

int main(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;                    //定义一个 GPIO_InitTypeDef 类型
                                                            //的结构体变量
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);    //开启 GPIOB 的时钟
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;               //选择要使用的 I/O 引脚,此处选
                                                            //择 PB0 引脚
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;        //设置引脚输出模式为推挽输出
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;       //设置引脚的输出频率为 50MHz
    GPIO_Init(GPIOB,&GPIO_InitStructure);                   //调用初始化库函数初始化 GPIOB
                                                            //端口
```

```
    while(1) {
          if(ReadOutputDataBit(GPIOB,GPIO_Pin_0) == 0){  //读取 PB0 状态
              GPIO_SetBits(GPIOB,GPIO_Pin_0);            //调用 GPIO_SetBits 函数,将 PB0
                                                         //置为高电平,点亮 D1
          }else{
             GPIO_ResetBits(GPIOB,GPIO_Pin_0);           //调用 GPIO_ReSetBits 函数,将
                                                         //PB0 置为低电平,熄灭 D1
          }
          Delay(72000);                                  //调用延迟函数,延迟 100ms
       }
}
```

3.4.2　基于 STM32CubeMX 的实例开发

通过 3.4.1 节的实验发现,基于标准库函数的 STM32 工程需要写很多初始化代码,这些初始化代码往往都是固定的,开发人员没必要每次都重写这些初始化程序。为解决这个问题,ST 公司推出了 STM32CubeMX 工具自动生成相应配置的初始化代码。下面以 STM32CubeMX 为例进行发光二极管闪烁程序的开发。

1. 构建 STM32CubeMX 工程

第 2 章已经介绍了 STM32CubeMX 工具的安装,找到如图 3-7 所示快捷图标,双击该图标启动 STM32CubeMX 开发工具。

图 3-7　STM32CubeMX 快捷图标

STM32CubeMX 开发工具启动后的首界面如图 3-8 所示。在 STM32CubeMX 的菜单中选中 File 选项卡,并单击“New Project...”,即可创建一个新的项目。

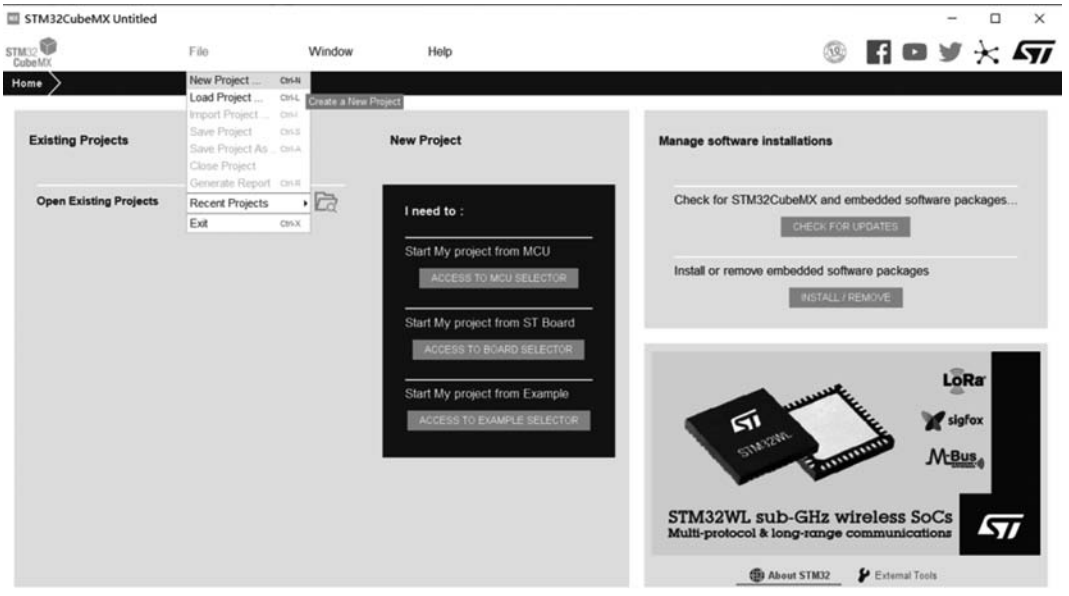

图 3-8　STM32CubeMX 开发工具首界面

与前面利用标准库函数创建工程不同，STM32CubeMX 的新工程都需要预先指定使用的芯片型号。图 3-9 是弹出的新创建项目界面，在 Part Number 搜索框中输入芯片型号。例如，STM32F103C 后按 Enter 键，检索出与输入关键词相近和封装不同的芯片信息，这里选择封装为 LQFP48 的 STM32F103C6 芯片。

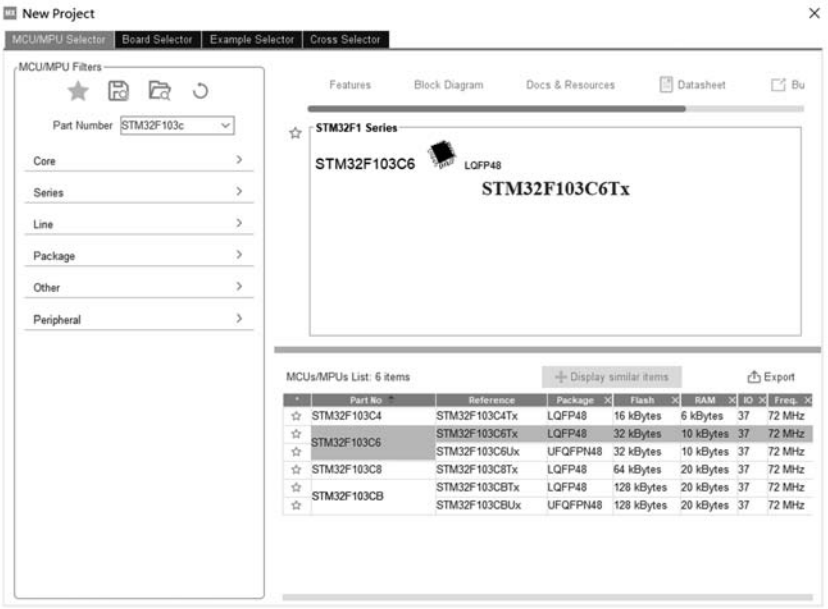

图 3-9　创建项目界面

双击 LQFP48 封装的 STM32F103C6 芯片，弹出如图 3-10 所示界面，至此工程创建完成。

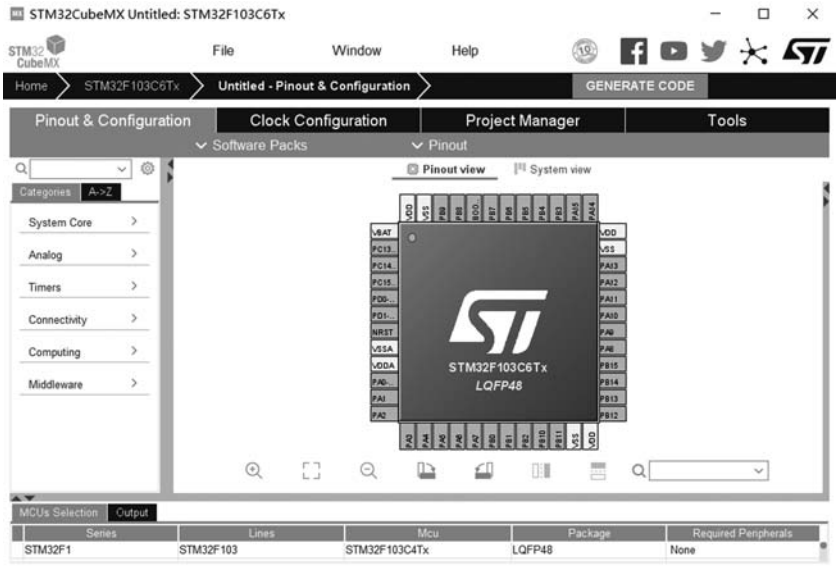

图 3-10　创建工程结束界面

1）配置 STM32CubeMX 工程

在创建了基于 STM32F103C6Tx 的工程后，还要对该芯片的引脚参数进行配置。首先配置芯片 STM32F103C6Tx 的 SYS，选择左边目录 System Core 中的 SYS 选项卡，会弹出 SYS 模式和配置（SYS Mode and Configuration）界面，如图 3-11 所示。

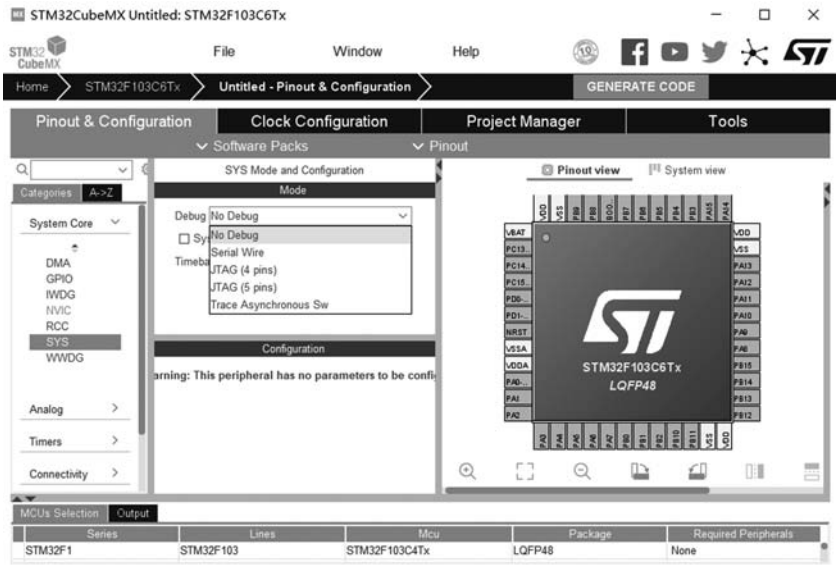

图 3-11　STM32F103C6Tx 芯片的 SYS 配置

Debug 的下拉列表中包括 No Debug、Serial Wire、JTAG（4 pins）、JTAG（5 pins）和 Trace Asynchronous Sw 五种模式。其中 No Debug 主要用于利用仿真软件调试程序。Serial Wire 是指利用 PA13 和 PA14 引脚进行串口调试。JTAG（4 pins）由 TDO4、TCK、TDI、TMS 线组成，分别为数据输出线、时钟、数据输入和模式选择。JTAG（5 pins）是在 4 线 JTAG 基础上加入了 VREF。Trace Asynchronous Sw 是指轨迹异步 SW 调试。由于 STM32CubeMX 默认是关闭调试接口的，为确保工程的完整性，可以选择 Trace Asynchronous Sw 把调试器选进来；另外，选进调试器也不会占用额外的程序代码。本节实验将利用仿真软件实现程序的调试，Debug 选项可以选为 No Debug 或 Serial Wire。

2）配置系统时钟

图 3-12 是配置 STM32F103C6Tx 的时钟。通过选择图 3-12 最左侧的 RCC，可以设置芯片的时钟源。首先从 STM32CubeMX 的引脚配置页面上找到 RCC 选项卡，单击 RCC 选项卡后右侧出现 RCC 模式和配置面板（RCC Mode and Configuration）。通过该面板配置高速时钟（HSE）和低速时钟（LSE）。考虑到在发光二极管闪烁程序中需要用到高速时钟，所以选定高速时钟后面的下拉列表框进行设置。一般而言，高速时钟和低速时钟的下拉选项卡包括禁止（Disable）、旁路时钟源（Bypass Clock Source）以及外部晶体/陶瓷谐振器（Crystal/Ceramic Resonator）三种。其中，Disable 选项表示当前工程不启用时钟源；Bypass Clock Source 是外部时钟，外部提供时钟只需要接入 OSC_IN 引脚，而 OSC_OUT 引脚悬空；而 Crystal/Ceramic Resonator 相当于石英/陶瓷晶振，需要通过外部无源晶体与芯片内部时钟的驱动电路共同配合形成时钟源，且 OSC_IN 与 OSC_OUT 引脚都要连接，

将会增加启动时间，但时钟源的精度往往较高。

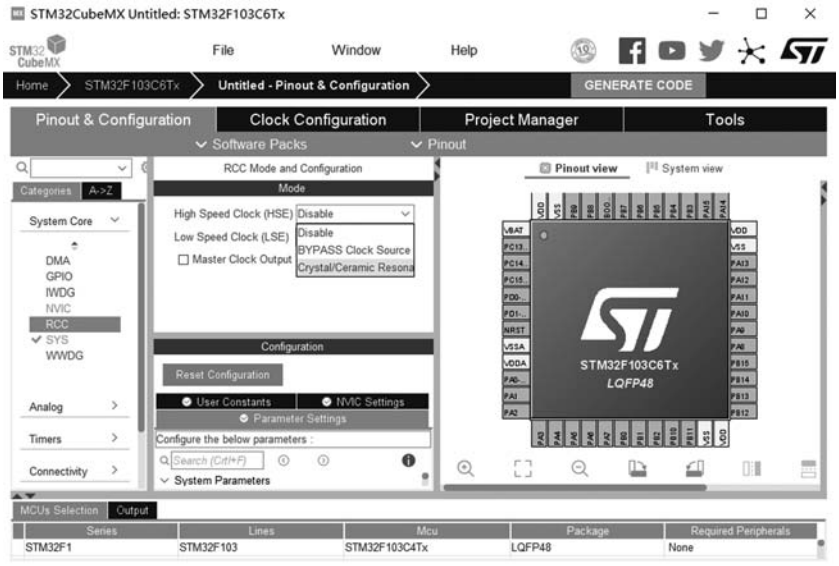

图 3-12　系统时钟选择

HSE 为本实验中采用的系统时钟源，选项选为 Crystal/Ceramic Resonator，此时 STM32F103C6Tx 的 PD0 和 PD1 引脚会被占用。

MCO 是指使能 MCO 引脚时钟输出。由于本次实验选择的是 HSE，所以只对 HSE 时钟配置，具体操作步骤如下：

（1）将 HSE 外部时钟源输入频率（Input Frequency）设置默认值 8，其取值范围为 4～16MHz；

（2）选择 PLL Source Mux 的通道，选择 PLL；

（3）双击 HCLK 频率，然后系统会自动配置成用于期望的时钟。配置前的时钟树如图 3-13 所示。

配置完成的时钟树如图 3-14 所示。

3）配置 GPIO 口功能

在完成 STM32F103C6Tx 系统时钟的配置后，需要继续配置其 GPIO 口功能。切换回 Pinout&Configuration 选项卡，打开如图 3-15 所示的界面。

开始配置 STM32F103C6Tx 芯片 GPIO 口的功能，本实验的目标是实现 PB0 连接发光二极管的闪烁，所以需要配置 PB0 引脚的功能。该引脚功能如下：

Reset_State 设置为低电平状态功能；

ADC1_IN8 是模/数转换器 1 通道 8 的数据采集功能；

ADC2_IN8 是模/数转换器 2 通道 8 的数据采集功能；

TIM1_CH2N 是高级定时器 1 的通道 2 功能；

TIM3_CH3 是通用定时器 3 的通道 3 功能；

GPIO_Input 是通用输入引脚功能；

GPIO_Output 是通用输出引脚功能；

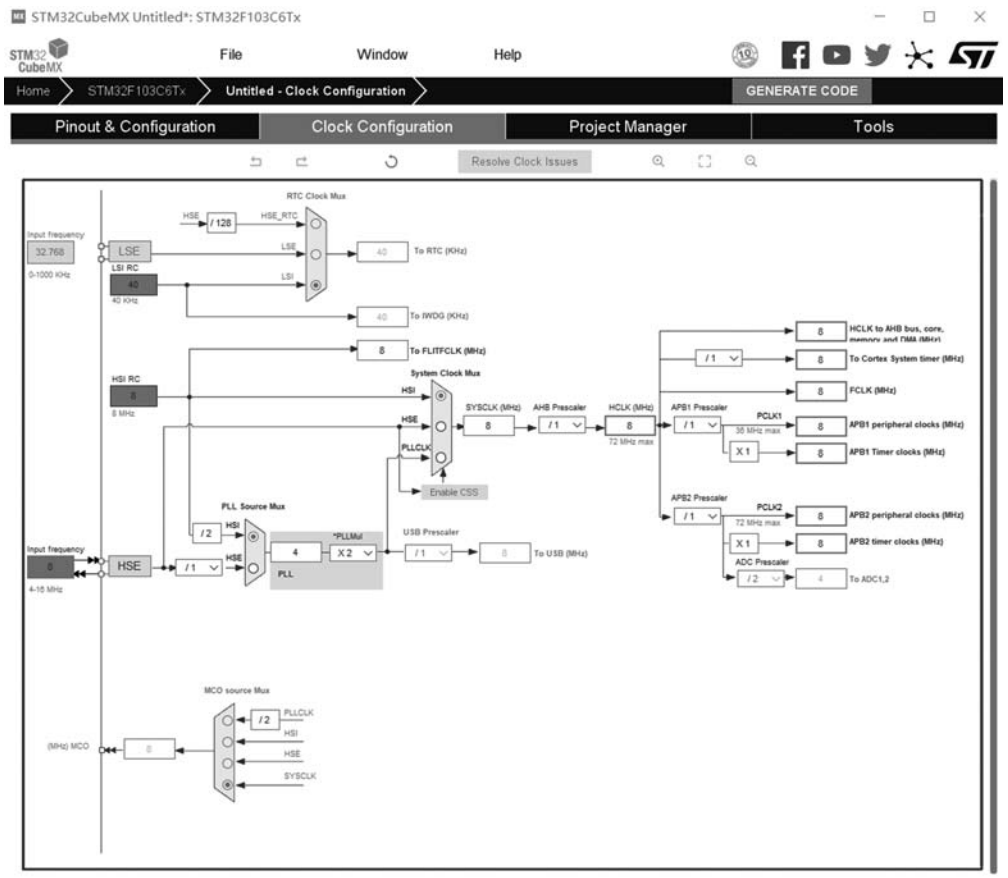

图 3-13　系统时钟配置

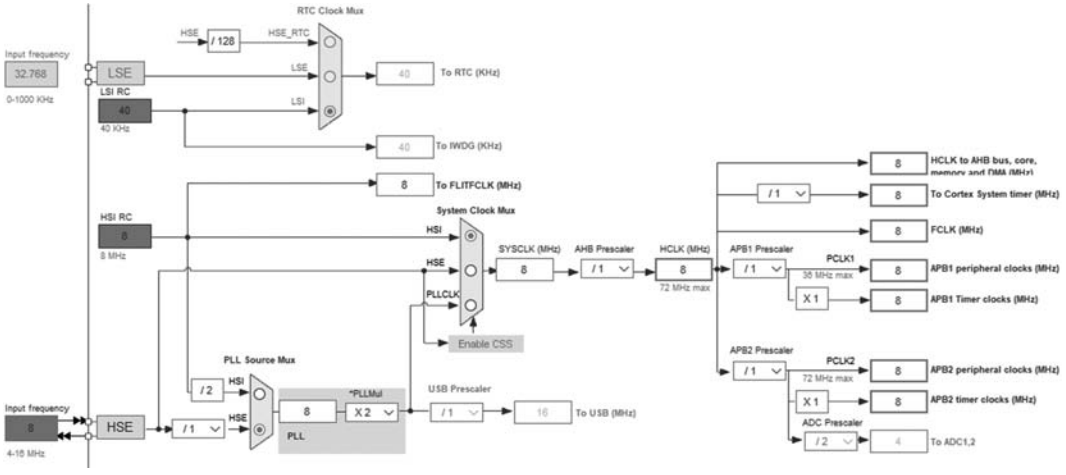

图 3-14　配置后系统时钟结构

图 3-15　GPIO 功能配置

GPIO_Analog 是通用模拟信号输出功能；

EVENTOUT 是事件输出功能；

GPIO_EXTI0 是中断 EXTI0 功能。

由图 3-5 可知，需要将 PB0 引脚配置为输出功能。单击图 3-15 中 STM32F103C6Tx 示意图的 PB0 引脚，在弹出的选项选框中，找到 GPIO_Output 选项条并选择，如图 3-16 所示。

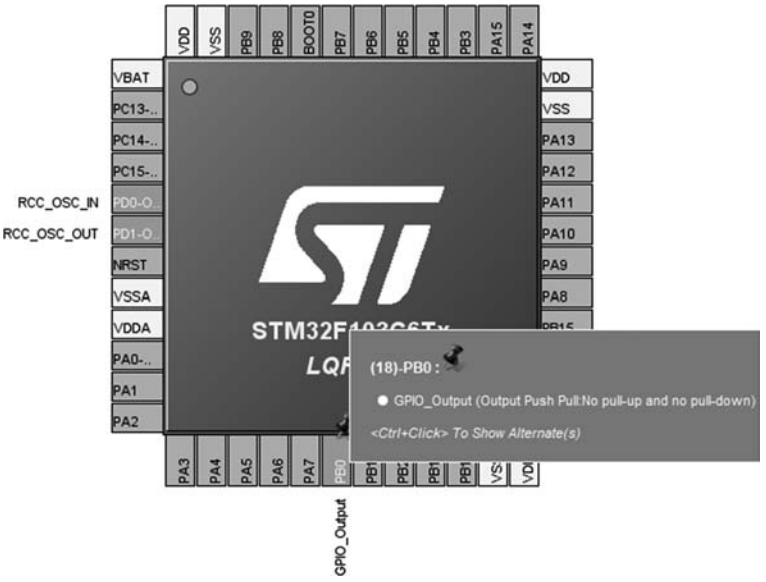

图 3-16　GPIO 功能配置

配置好 PB0 为输出后，右击 PB0 引脚可以为 GPIO 的标识设置，这个标识可以在程序中直接使用，而不需要通过 PB0 的地址进行访问，如图 3-17 所示。

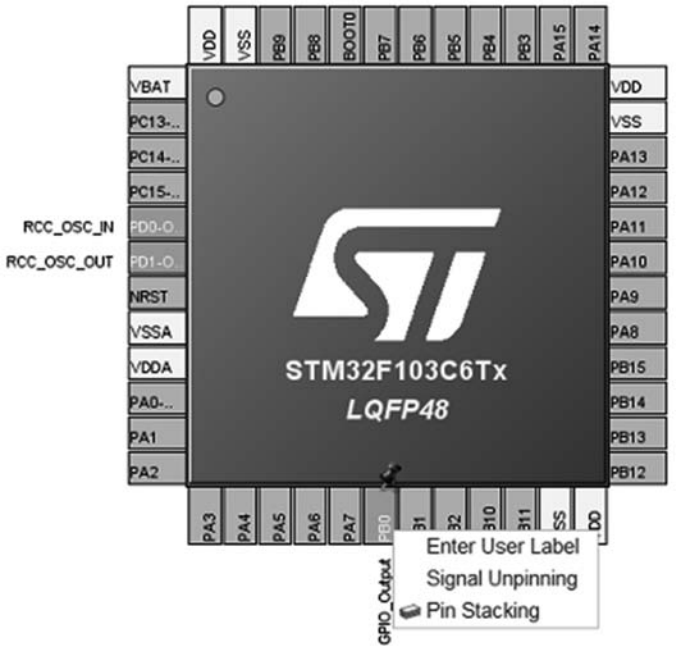

图 3-17　GPIO 标识设置

下面选择 Enter User Label 选项，本次设置为 LED-RED，如图 3-18 所示。

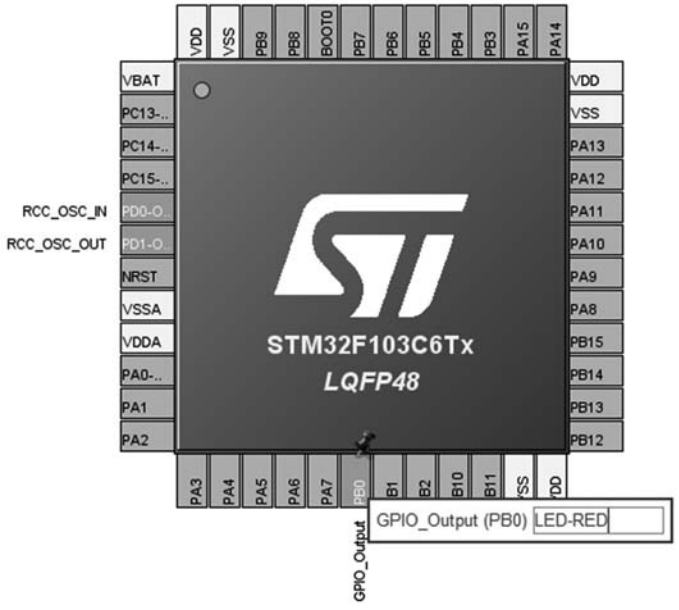

图 3-18　GPIO 标识分配

以上 PB0 引脚功能配置结束后，接下来在最左侧的 System Core 目录下单击 GPIO 选项卡。在 GPIO 模式和配置(GPIO mode and configuration)页面中列出了上面配置 PB0 引脚的信息，如图 3-19 所示。

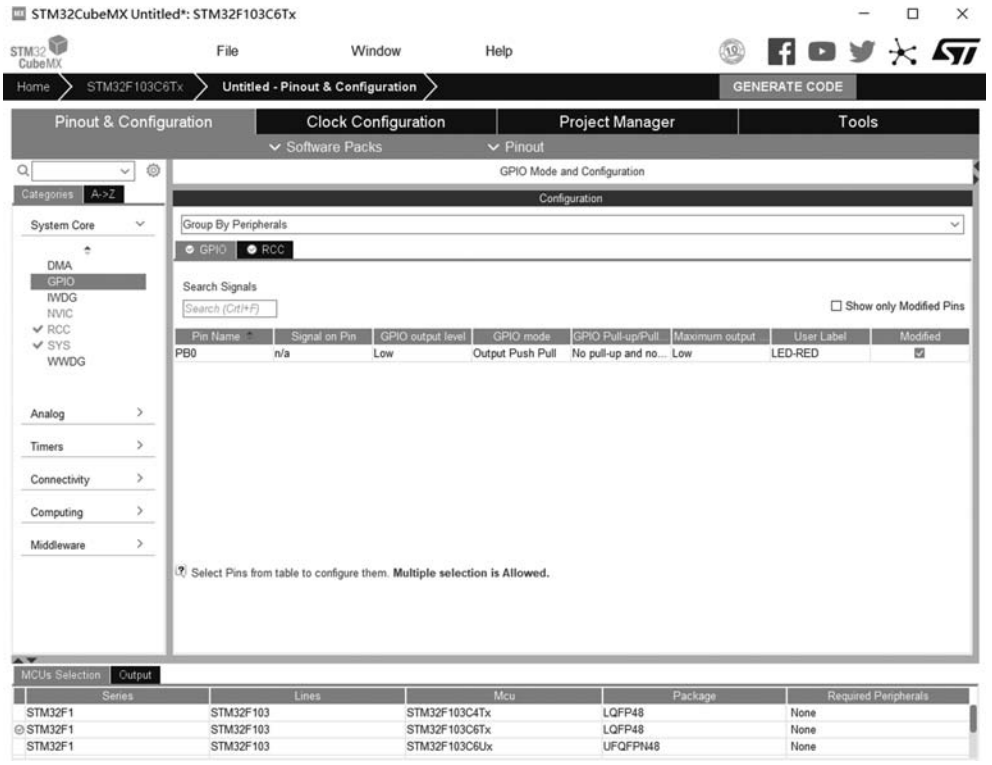

图 3-19　GPIO 的 PB0 配置信息

单击列表中的 PB0 查看引脚详细参数，如图 3-20 所示。GPIO output level 选项可以将引脚电平设置成高电平或低电平。GPIO mode 为 GPIO 模式，具体见表 3-2。GPIO Pull-up/Pull-down 为上拉电阻、下拉电阻、无上拉或下拉。Maximum output speed 为引脚速度设置，包括低速、中速、高速。User Label 为用户标签，可以给引脚设置名称，如 LED-RED。根据本实验的要求将 PB0 配置为推挽输出模式、上拉、高速输出模式，并且引脚标识为 LED-RED。

至此，STM32F103C6Tx 芯片的基本参数已经配置完成了。可以看出和我们使用库函数时的配置过程是一样的，但不同的是仅需要单击鼠标便可以完成上述操作，这正是 STM32CubeMX 的强大之处。

4) 输出配置后的工程

根据发光二极管闪烁实验的功能需求，需要对 STM32F103C6Tx 芯片进行一定的配置，为了能在仿真器或芯片中执行配置后的工程，需要使用 Project Manager 选项卡配置导出当前工程生成的源程序，进入图 3-21 所示的左侧 Project 选项卡的 Project Settings 界面进行输出配置。

其中，Project Name 为当前工程文件名，该文件名根据实验的实际功能进行自定义，在本实验中将 STM32_GPIO_LED；Project Location 作为当前工程存放的路径，也可以根据

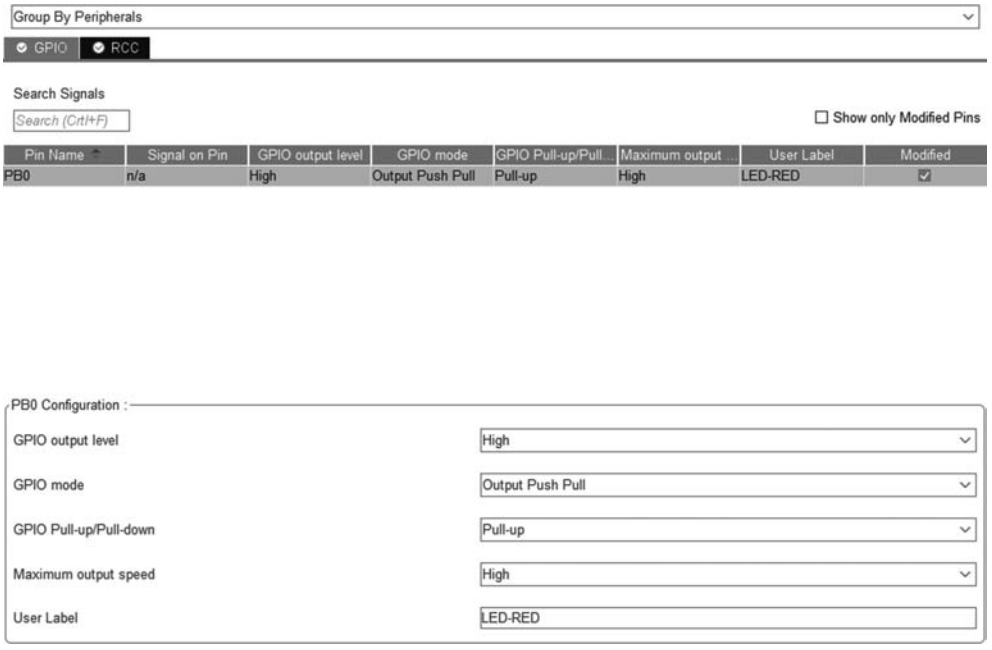

图 3-20　GPIO 详细配置界面

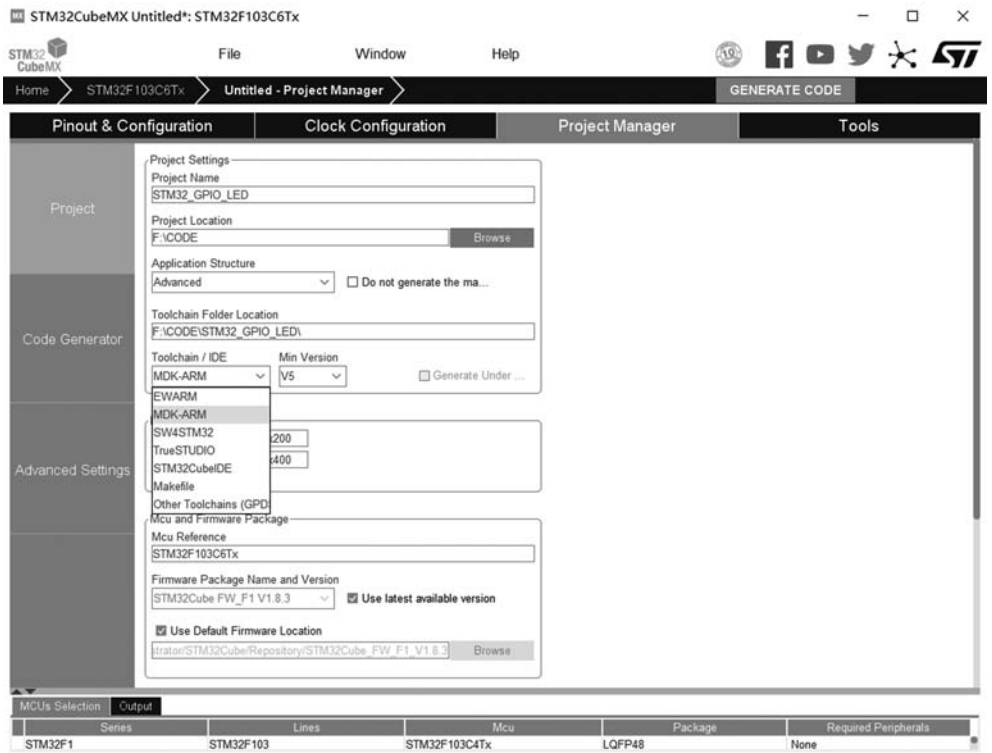

图 3-21　配置好的输出配置

硬盘的情况选择合适的路径；而 Toolchain/IDE 是非常关键的，它决定了用 STM32CubeMX 生成代码所能编译源代码的集成开发环境，即确定集成开发环境的类型。这需要根据项目的开发环境进行选择，本书用到的集成开发环境是 Keil，因此选择 MDK-ARM。Min Version 是对应集成开发环境的版本号，用于本书选用的是 Keil5，所以选择 V5 的版本；除此之外的参数，保持系统默认即可。如有特殊的要求，需要结合实际项目开发的需要进行设定。

单击左侧的 Code Generator 选项卡可以进入如图 3-22 所示的配置界面，其选择内容具体包括：为每一个外设的芯片能够生成独立的初始化头文件和源文件，选择只复制所需文件到工程，这样就会有.h 和.c 两种形式的输出文件；当代码重新生成时，文件会保留用户原有代码，而当没有代码重新生成时，就会自动删除之前系统生成的文件。

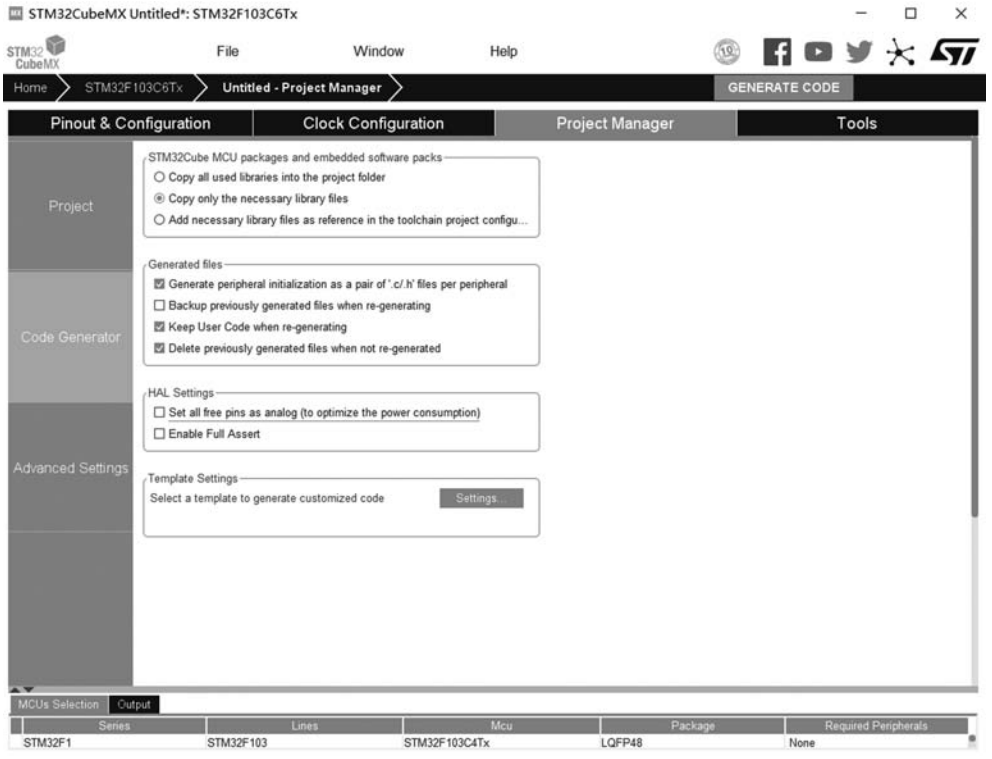

图 3-22　配置 Code Generator

5）生成代码

以上选项信息设置好后，单击 Generate Code 按钮便可生成 Keil 源代码，如图 3-23 所示。

代码生成后弹出是否打开工程对话框，如图 3-24 所示。

利用 Stm32CubeMX 生成源代码后，通过单击弹出的对话框中的“Open Project”按钮，便可以启动 Keil 5，同时用 Keil5 打开当前生成源代码的工程文件，启动后的效果如图 3-25 所示。

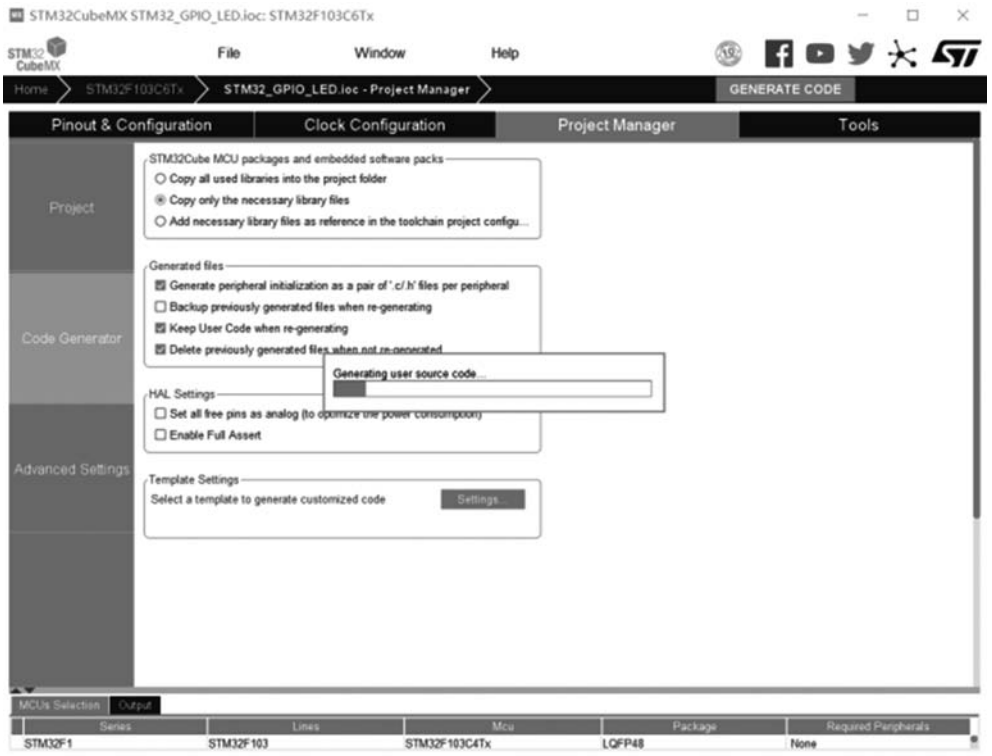

图 3-23　生成用户源代码

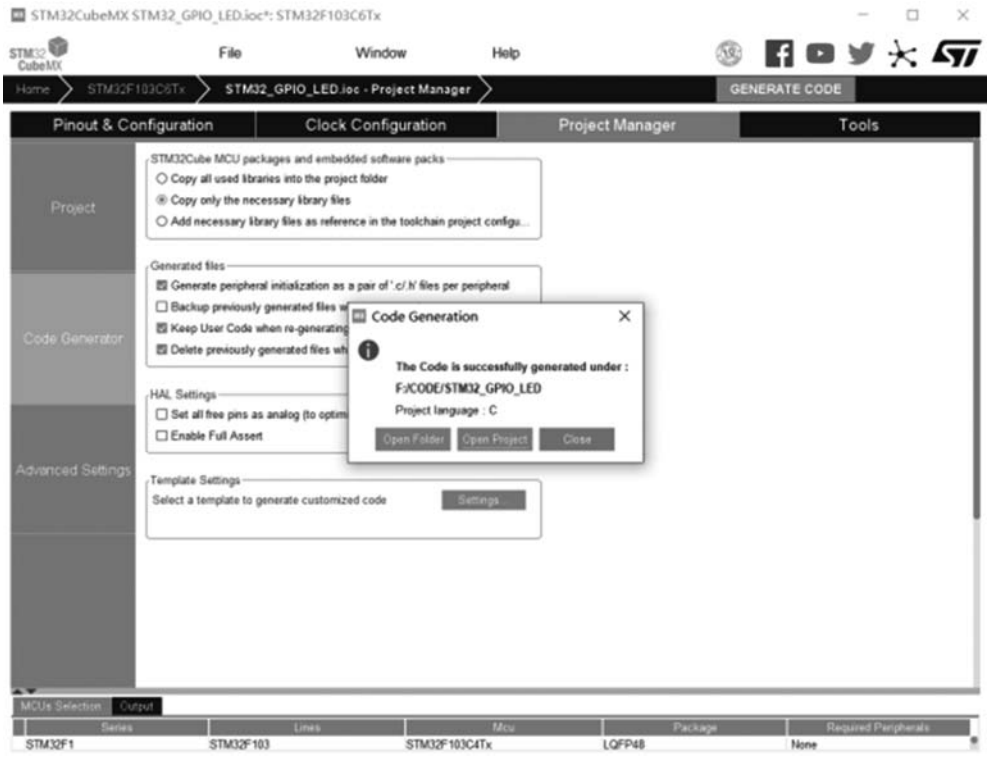

图 3-24　代码生成后弹出对话框

图 3-25　启动 Keil 开发环境

2. Keil 软件

Keil 软件启动后，进入图 3-26 所示的页面。STM32CubeMX 生成了左侧工程目录树中的源程序，接着选择左侧工程目录树中 Application/User/Core 文件下的 main.c 文件，然后在 Keil 软件的右侧显示窗打开源代码，如图 3-26 所示。

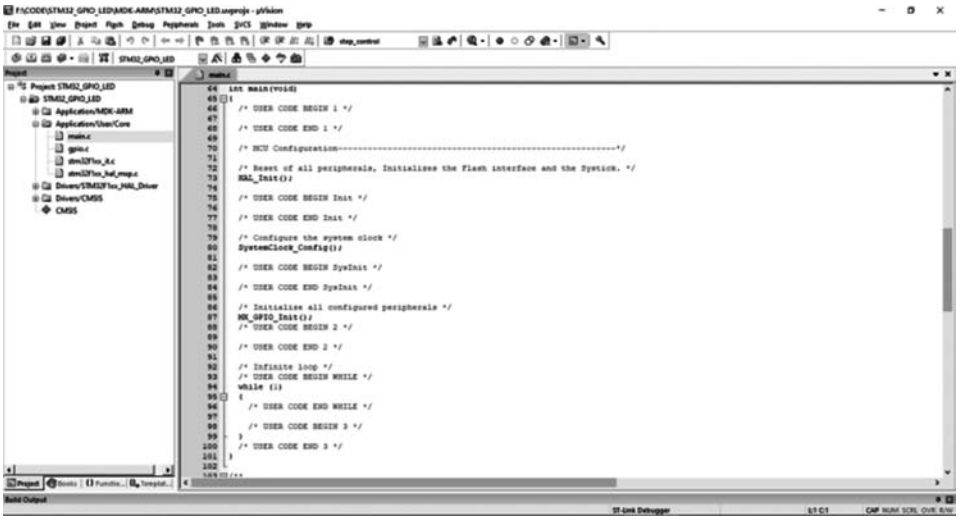

图 3-26　Keil 首页面

根据已经生成的源文件可以看到，发光二极管闪烁实验大部分的源代码都已经生成，其中包括启用相应外围设备的时钟、配置外设功能参数、调用初始化函数、初始化外设相关的参数以及使能相应的外设等，但应用业务逻辑的源代码，用户需要根据具体的应用来设计和实现此部分代码，如图 3-27 所示。

使用 Keil 软件将编写完成的发光二极管闪烁程序进行编译，继而生成.hex 文件。

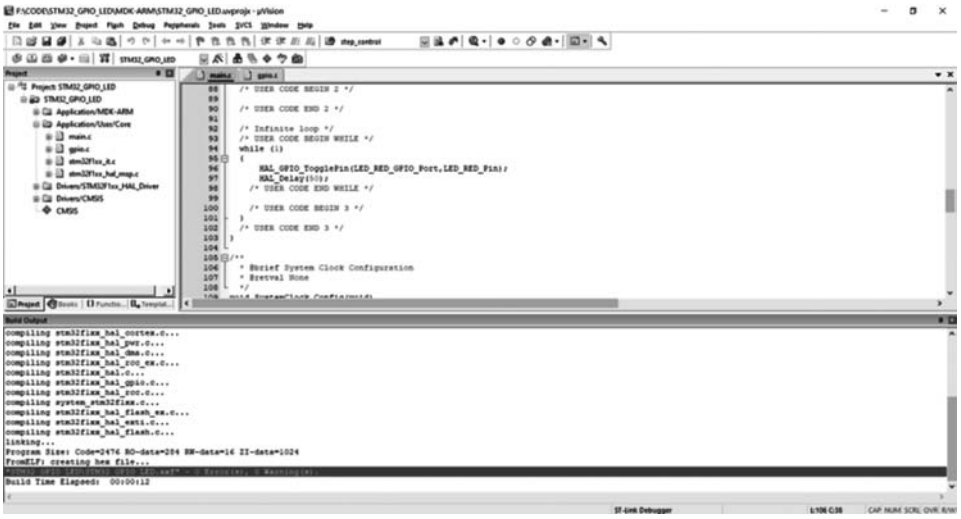

图 3-27　发光二极管闪烁程序的应用逻辑

3. Proteus 仿真

为了确定上述程序编写的准确性，可以运用 Proteus 工具构建 STM32F103C6Tx 的发光二极管闪烁的仿真环境，利用 Proteus 模拟 STM32F103C6Tx 在物理环境下的运行状态，然后通过观察 STM32F103C6Tx 引脚的变化来证明上述实验的有效性。

图 3-28　Proteus 8 Professional 快捷图标

1）创建 Proteus 工程

双击桌面上 Proteus 8 Professional 的快捷图标，启动软件，如图 3-28 所示。

软件启动后，进入图 3-29 所示的首界面。

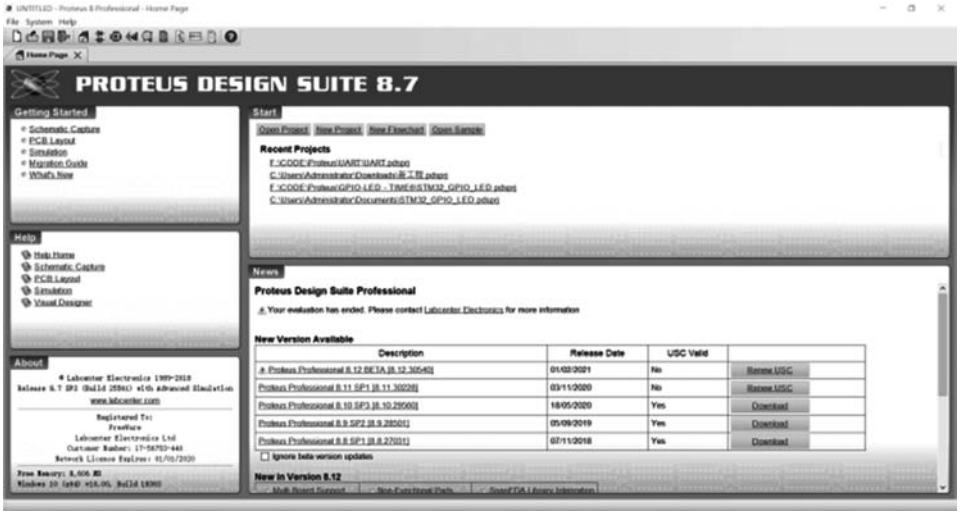

图 3-29　Proteus 8 Professional 的首界面

单击 Proteus 8 的 File 选项卡，选择 New Project 选项，打开如图 3-30 所示的新建工程界面。

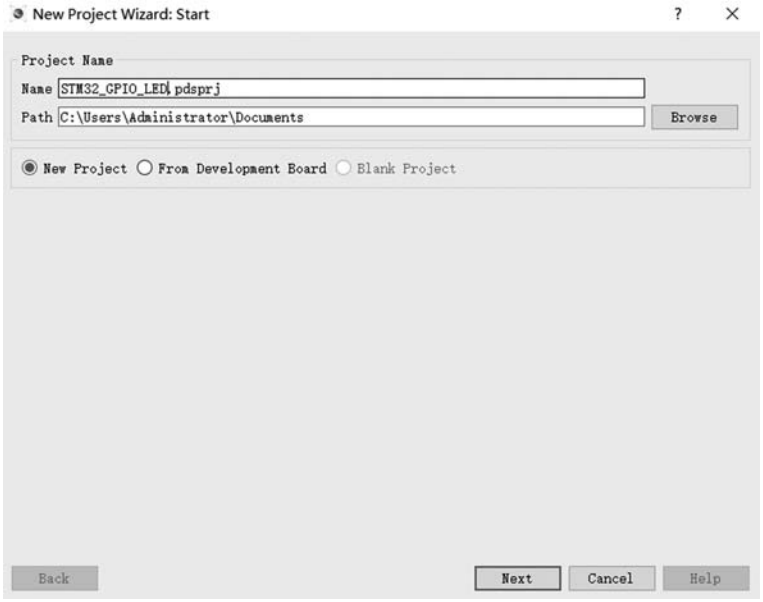

图 3-30　新建工程界面

设置好工程名和工程存放的物理位置后，单击“Next” Next 按钮，进入如图 3-31 所示的原理图大小设置界面，根据设计需要选择图纸大小。

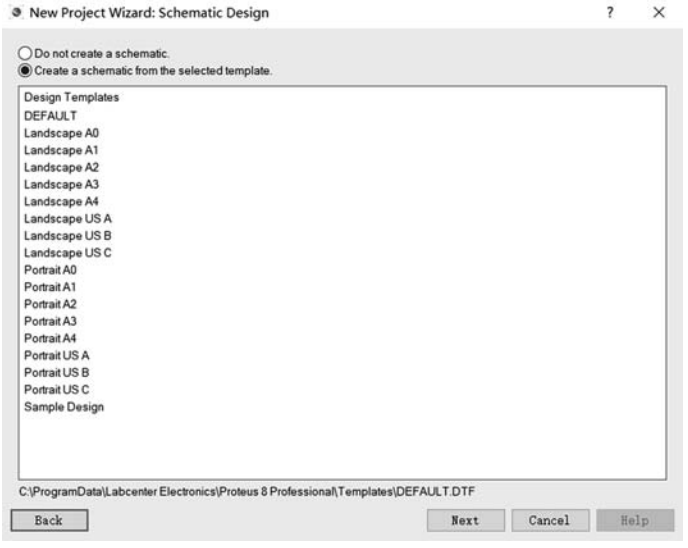

图 3-31　原理图大小设置界面

原理图的尺寸可根据元器件的多少进行选择，本实验选择 DEFAULT 模板，选择完毕后单击“Next” Next 按钮进入如图 3-32 所示的 PCB 配置界面。

如果需要创建 PCB，则可以选择创建 PCB 选项，而且需要选择合适的模板。本实验不

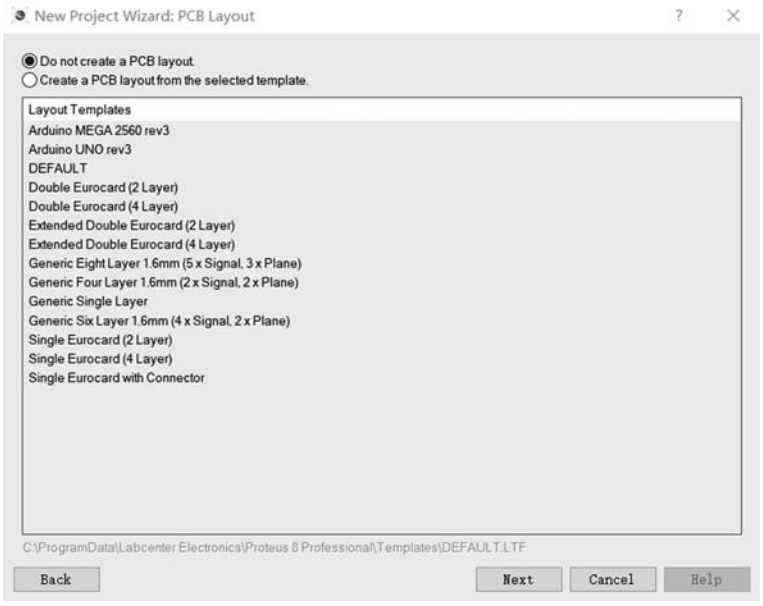

图 3-32　PCB 配置界面

需要 PCB，所以选择不创建 PCB 布局选项，即无须设计 PCB。然后选择“Next”Next按钮即可，进入如图 3-33 所示的固件库设置界面。

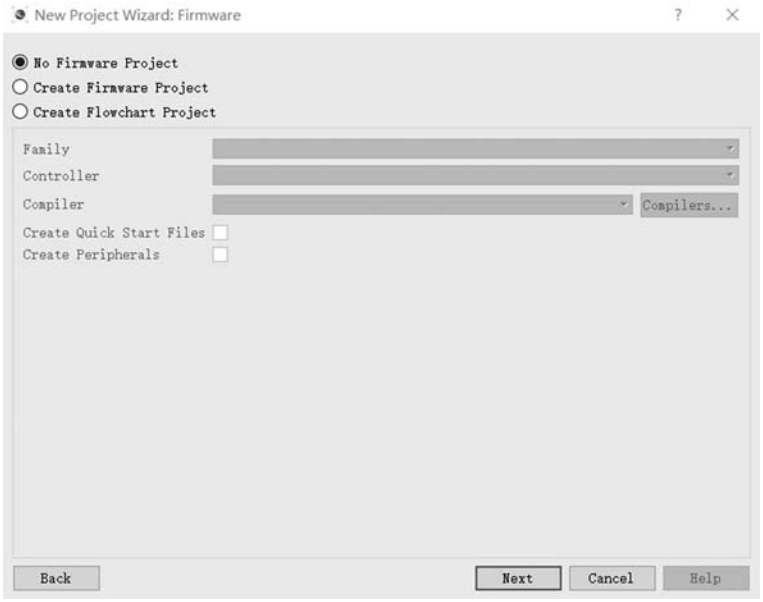

图 3-33　固件信息配置

固件库是指确保程序在控制器上正确运行所需要的中间件，需要结合项目的实际情况进行配置。本书使用 STM32CubeMX 和 Keil 联合对 STM32 进行仿真验证，所以无须用 Proteus 进行开发，故直接单击“Next”Next按钮进入图 3-34 所示的配置概要界面。

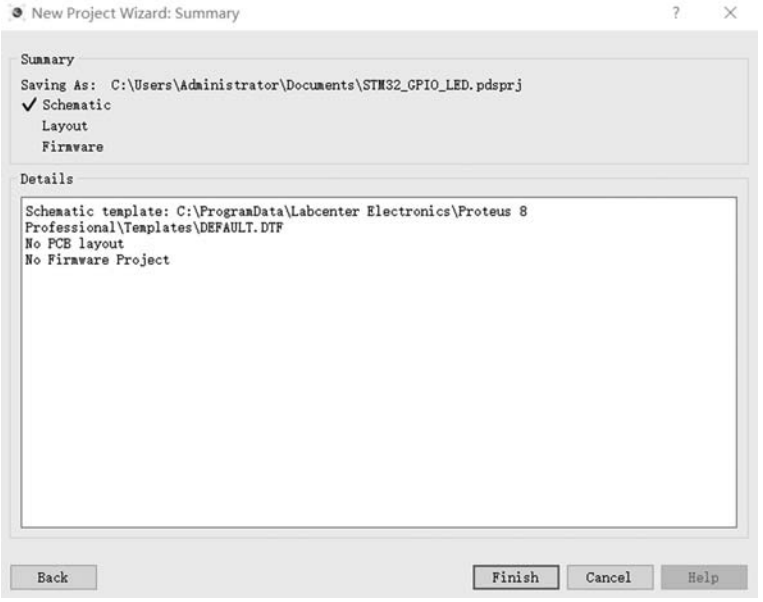

图 3-34 配置概要

单击“Finish”按钮完成仿真环境的工程创建。接下来，利用当前工程对实验进行仿真。

2）检索器件

单击 Library 选项卡，并单击从库选择部件（Pick parts from libraries），如图 3-35 所示，添加本实验所需的元器件。

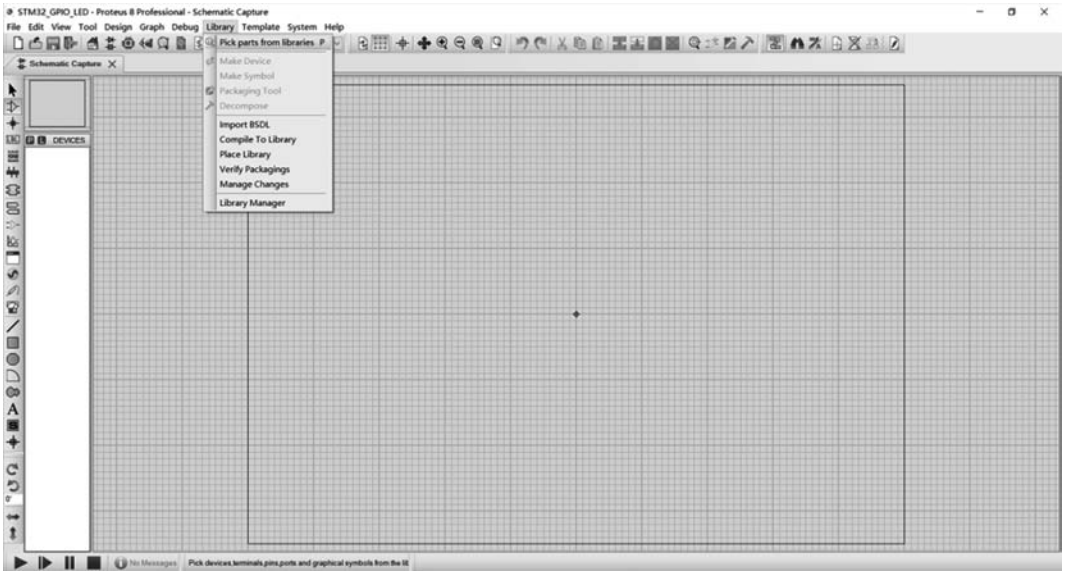

图 3-35 器件选择

进入选择元器件界面后，在 Keywords 界面输入要检索的器件并回车后，可以查看具体的器件结果。例如，输入关键词 STM32 并按回车后，右边列表会列出与 STM32 相关的

MCU，如图 3-36 所示。

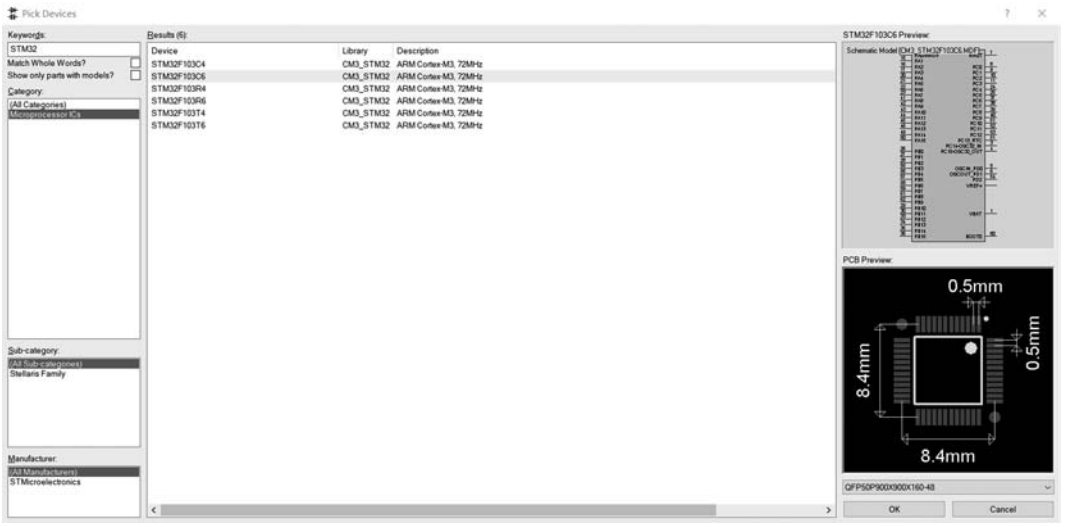

图 3-36　MCU 选择

本次选择 STM32F103C6 作为仿真实验的 MCU，如图 3-37 所示。

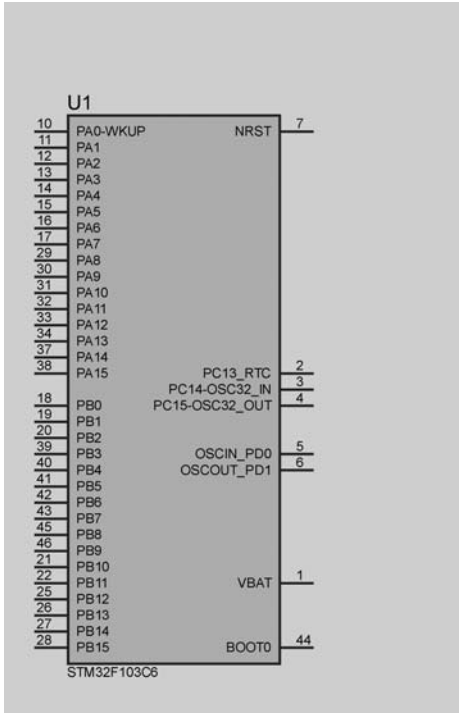

图 3-37　STM32F103C6 元件

当单击 STM32F103C6 元件，则该器件便添加到原理图中，如图 3-38 所示。

用同样的方式，选择本实验需要的发光二极管，如图 3-39 所示。

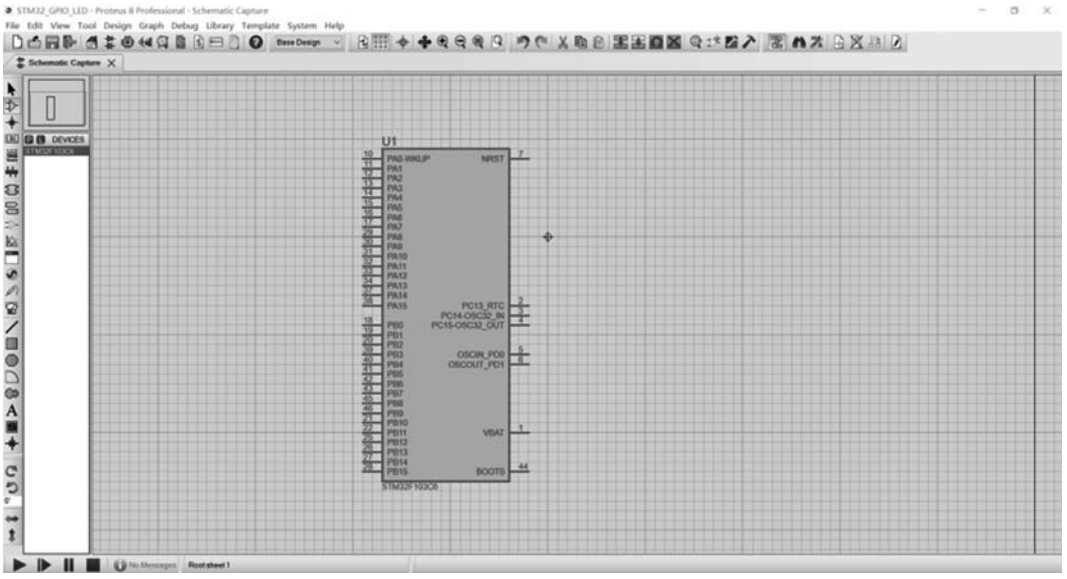

图 3-38　添加 STM32F103C6 元件

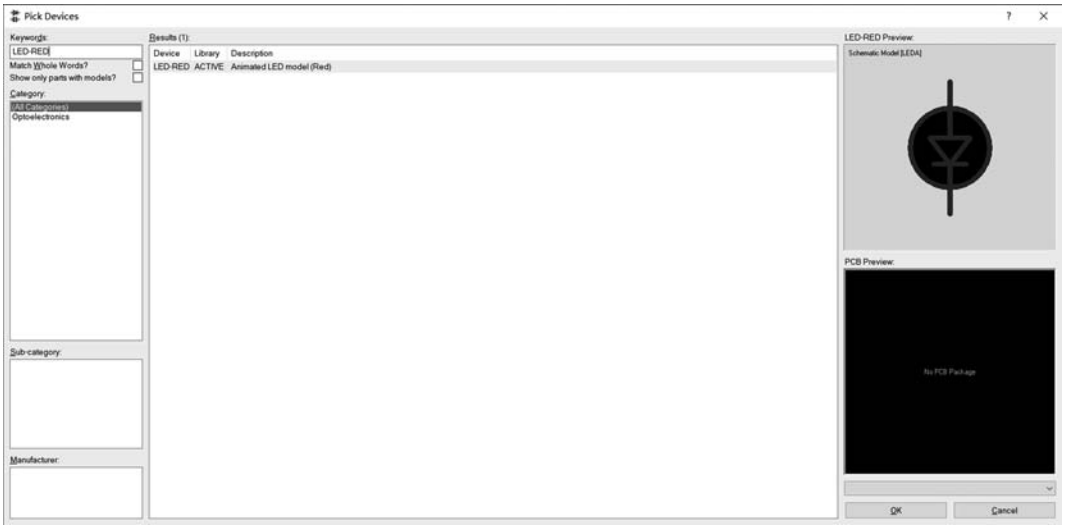

图 3-39　添加发光二极管

最后，添加限流电阻 R1 和接地，并将这些元件连接到一起。注意，D1 需要接到 STM32F103C6 元件的 PB0 引脚。发光二极管闪烁实验原理图如图 3-40 所示。

接下来配置 STM32F103C6，如图 3-41 所示。双击弹窗上的 STM32F103C6 选项，在 STM32F103C6 芯片中加载之前用 Keil 编译好的发光二极管闪烁程序并设置好 STM32F103C6 芯片的晶振频率，其余设置均保持默认值，最后单击 OK 按钮。

3）仿真

基于发光二极管闪烁实验的原理图搭建的电路运行仿真，如图 3-42 所示。单击运行仿真或者快捷键 F12 进行仿真实验。

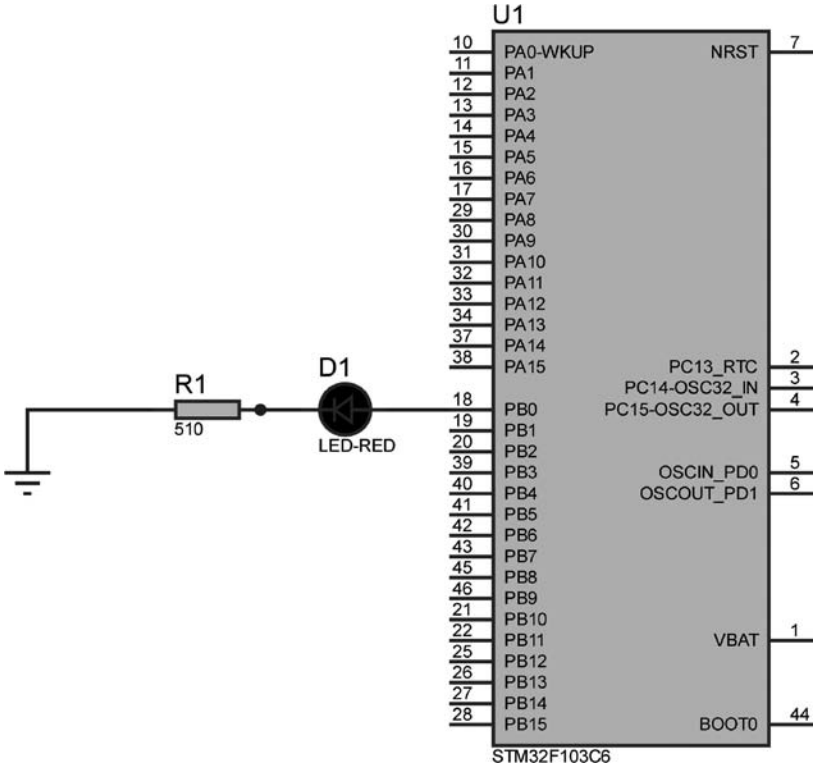

图 3-40　发光二极管闪烁实验原理图

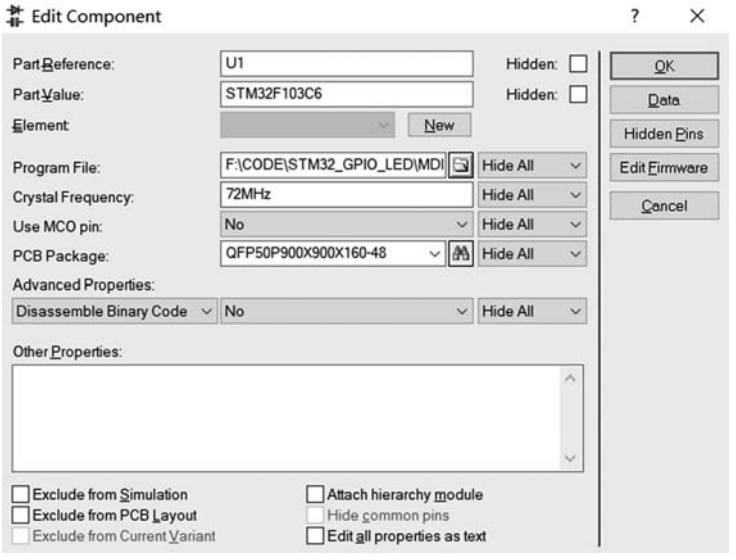

图 3-41　STM32F103C6 芯片配置

运行仿真后可以观察 STM32F103C6 引脚 PB0 和 LED-RED 的变化。根据发光二极管闪烁程序，每隔 50ms，LED-RED 状态变化一次。LED-RED 关状态的仿真结果如图 3-43 所示，这时 PB0 引脚输出低电平。

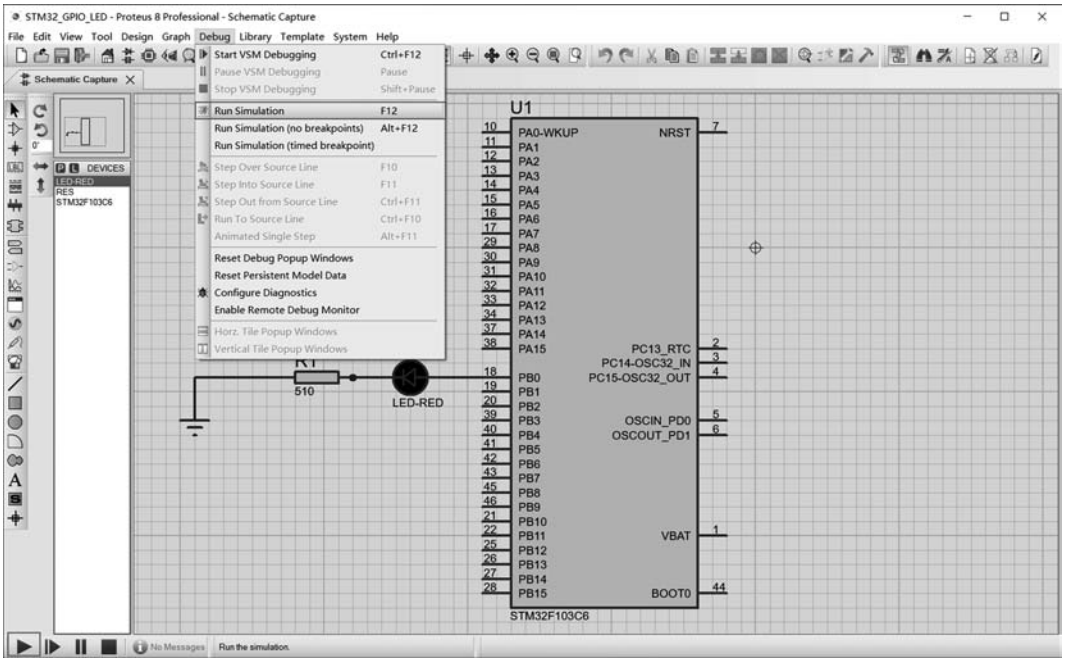

图 3-42　运行仿真

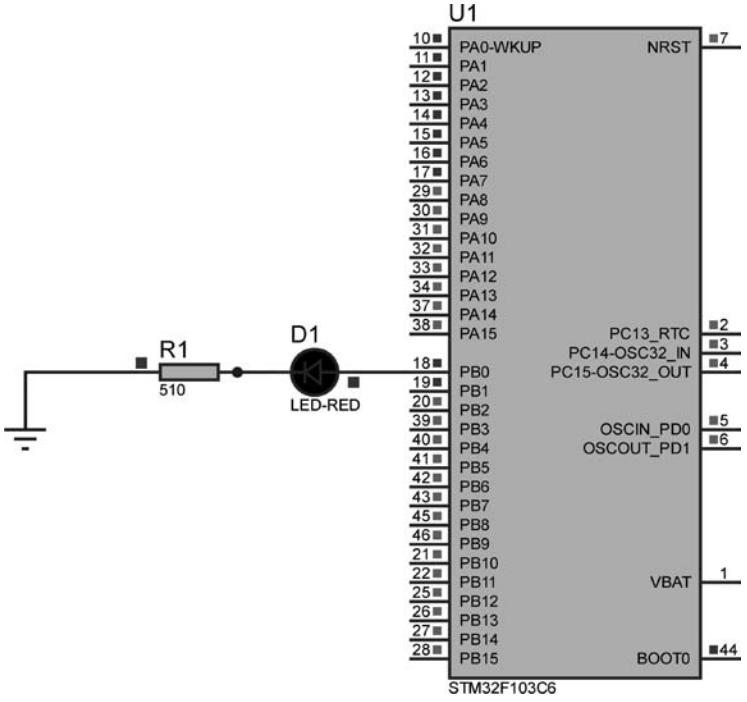

图 3-43　LED-RED 关闭

图 3-44 是 LED-RED 开状态的仿真结果。这时 PB0 引脚由程序拉高，LED-RED 变红，说明发光二极管被打开。

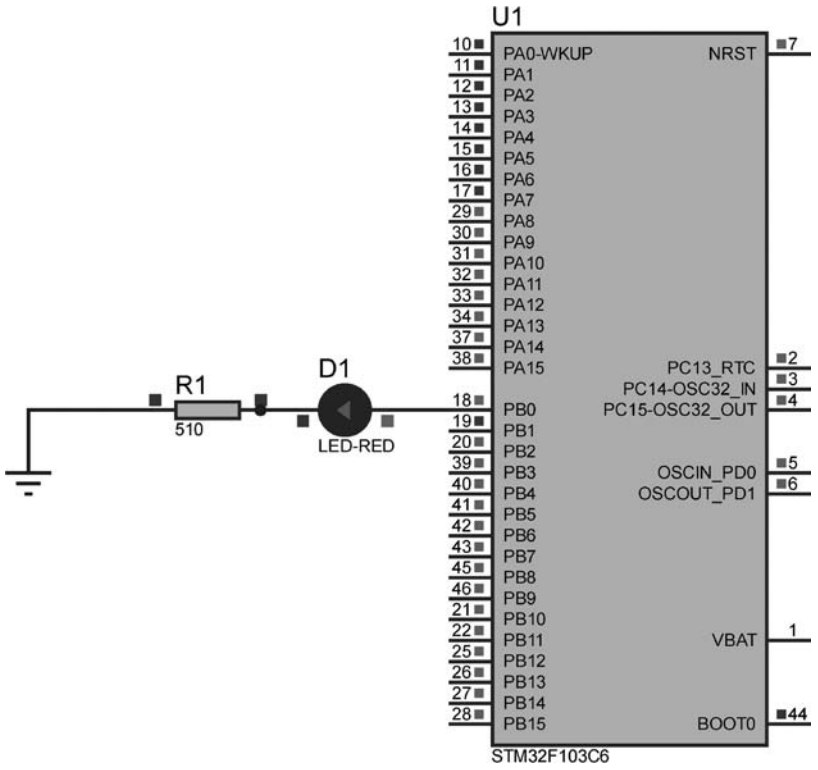

图 3-44　LED-RED 打开

本章小结

本章主要介绍了输入/输出的概念、STM32 的 GPIO、标准库函数、GPIO 配置的步骤等，最后通过一个发光二极管闪烁程序进一步说明了如何使用 GPIO。通过本章的学习，要求学生掌握输入/输出的有关概念、掌握 STM32 的 GPIO 开发流程、熟练使用 STM32CubeMX 工具生成 GPIO 项目、熟练应用 Keil 软件编译环境和仿真环境、了解 Proteus 仿真环境，对基于 STM32 的嵌入式系统开发建立初步的认识，为后面的学习打下基础。

习题 3

1. 填空题

（1）微控制器芯片将输入/输出分为_____、_____、_____和_____。

（2）_____用来配置 GPIOx 端口的使能位。

2. 选择题

（1）（　　）指输出较大的电流来驱动外围电路设备，该模式既可以输出高电平又可以输出低电平。

A. 准双向 I/O　　B. 推挽输出　　C. 高阻态　　D. 开漏

(2)(　　)是指 STM32 微控制器片内外设中可配置的输入/输出接口。

A. GPIO　　B. EXTI　　C. DMA　　D. NVIC

(3) STM32 的 GPIO 的每个端口都有(　　)个输入/输出引脚。

A. 8　　B. 16　　C. 32　　D. 0

(4) 关于 STM32 的 GPIO 引脚的寄存器描述不正确的是(　　)。

A. 每个 GPIO 引脚都有两个 32 位配置寄存器

B. 每个 GPIO 引脚都有两个 32 位数据寄存器

C. 每个 GPIO 引脚都有一个 32 位锁定寄存器

D. 每个 GPIO 引脚都有一个 32 位复位寄存器

(5) 每个 GPIO 引脚有三种输出速度,不包括(　　)。

A. 50MHz　　B. 10MHz　　C. 2MHz　　D. 100MHz

3. 简答题

(1) 简述什么是输入/输出,及各种工作模式的含义。

(2) 简述 GPIO 引脚包含几种输入和输出模式,及每种模式的含义。

(3) 简述 GPIO 配置步骤。