第3章 通用输入/输出口

通用输入/输出口是微控制器与外部设备交互的重要接口。GPIO提供了灵活的输入/输出功能，可以配置为数字输入、数字输出、复用功能输出以及外部中断触发等功能。本章将深入探讨STM32F407 GPIO模块的寄存器、引脚模式、HAL库函数以及设计实例。从基础概念到设计应用，有助于更好地掌握STM32F407的输入/输出操作，为开发高效、可靠的嵌入式系统打下坚实基础。

知识目标：

- 阐述输入/输出模块的相关概念；
- 阐述STM32F407的GPIO构成；
- 说明STM32F407 GPIO的HAL库函数；
- 说明GPIO的开发流程。

能力目标：

- 运用STM32CubeMX软件配置GPIO工程；
- 运用Keil MDK软件编写程序，并进行分析；
- 运用Proteus软件搭建GPIO仿真电路；
- 设计基于GPIO的实例。

素质目标：

- 通过实例实践，培养学生的动手能力和解决问题的能力；
- 强调团队合作和沟通的重要性，让学生在实践中学会与他人协作完成任务；
- 鼓励学生进行创新性思考和探索，将GPIO应用于更多领域，提升创新能力和实践技能。

3.1 嵌入式系统的输入/输出

嵌入式系统的输入/输出是系统与外界环境或内部组件之间进行数据交换的关键部分。在嵌入式系统中，I/O扮演着至关重要的角色，它使得系统能够接收来自外部设备（如传感器、键盘等）的输入信息，并通过输出设备（如显示器、电机等）将处理结果反馈给外部环境。

嵌入式系统的I/O设备种类繁多，包括但不限于数字I/O、模拟I/O、串行通信、并行通信、网络通信等。这些设备通过特定的接口与嵌入式处理器相连，实现数据的传输和控制。

在嵌入式系统设计中，I/O 驱动是连接硬件设备和软件系统的桥梁。驱动程序负责控制硬件设备的工作状态，如初始化设备、读写数据、处理中断等。此外，嵌入式系统的 I/O 操作还涉及数据缓冲、中断处理、同步与互斥等复杂问题。为了提高系统的实时性和可靠性，需要深入理解 I/O 设备的特性和工作原理，并合理设计 I/O 驱动和应用程序。

嵌入式系统的 I/O 是系统与外界环境交互的窗口，其性能和可靠性直接影响到整个系统的功能和性能。因此，在嵌入式系统设计中，要重视 I/O 设备的选择和驱动程序的开发。

3.2　STM32F407 的 GPIO

STM32F407 的 GPIO 是微控制器与外部设备通信、控制及数据采集的重要接口。STM32F407 的 GPIO 引脚具有高度的灵活性和可配置性，能够满足多种应用需求。

STM32F407 的 GPIO 引脚被分成多个组，每组包含一定数量的引脚，便于管理和配置。GPIO 引脚具备基本的输入/输出功能，在输出模式下，STM32F407 可以通过控制引脚输出高、低电平，实现对外部设备的控制，如控制 LED 灯的亮灭或驱动继电器等；在输入模式下，GPIO 引脚可以检测外部电平信号，用于读取按键状态或接收外部设备的信号。

STM32F407 的 GPIO 支持多种工作模式，包括浮空输入、上拉输入、下拉输入、模拟输入、开漏输出、推挽输出、复用开漏输出和复用推挽输出等。这些模式允许开发者根据具体的应用场景选择合适的配置，以满足不同的需求。GPIO 引脚内置了保护二极管和上下拉电阻等保护机制，以防止引脚受到过高或过低的电压冲击，从而保护芯片免受损坏。STM32F407 的 GPIO 引脚配置非常灵活，开发者可以通过软件编程来配置引脚的工作模式、输出速度等参数，以满足不同的应用需求。

在使用 STM32F407 单片机的 GPIO 之前，需要对其进行配置。GPIO 引脚配置步骤为：启用对应 GPIO 的时钟；配置 GPIO 的引脚功能；使能 GPIO；编写业务逻辑。在实际应用中，还需要考虑 GPIO 引脚的电气特性，如电压和电流规格等，以避免超出规格范围导致单片机或外部设备损坏。

STM32F407 的 GPIO 是微控制器控制、与外部设备通信及数据采集的重要接口，具有高度的灵活性和可配置性，广泛应用于各种嵌入式系统，通过合理选择 GPIO 工作模式和参数进行 GPIO 引脚配置，可以实现与外部设备的有效通信和控制。

3.2.1　STM32F407 GPIO 寄存器

STM32F407 系列寄存器组起始地址如表 3-1 所示。

表 3-1　STM32F407 系列寄存器组起始地址

总　线	起 始 地 址	外　设
AHB3	0xA000 0000～0xA000 0FFF	FSMC 控制寄存器
	0x9000 0000～0x9FFF FFFF	FSMC bank 4
	0x8000 0000～0x8FFF FFFF	FSMC bank 3
	0x7000 0000～0x7FFF FFFF	FSMC bank 2
	0x6000 0000～0x6FFF FFFF	FSMC bank 1

续表

总　线	起始地址	外　设
AHB2	0x5006 0800～0x5006 0BFF	RNG
	0x5005 0400～0x5006 07FF	HASH
	0x5005 0000～0x5005 03FF	DCMI
	0x5006 0000～0x5006 03FF	CRYP
	0x5000 0000～0x5003 FFFF	USB OTG FS
AHB1	0x4004 0000～0x4007 FFFF	USB OTG HS
	0x4002 B000～0x4002 BBFF	DMA2D
	0x4002 8000～0x4002 93FF	以太网 MAC
	0x4002 6400～0x4002 67FF	DMA2
	0x4002 6000～0x4002 63FF	DMA1
	0x4002 4000～0x4002 4FFF	BKPSRAM
	0x4002 3C00～0x4002 3FFF	闪存接口寄存器
	0x4002 3800～0x4002 3BFF	RCC
	0x4002 3000～0x4002 33FF	CRC
	0x4002 2800～0x4002 2BFF	GPIOK
	0x4002 2400～0x4002 27FF	GPIOJ
	0x4002 2000～0x4002 23FF	GPIOI
	0x4002 1C00～0x4002 1FFF	GPIOH
	0x4002 1800～0x4002 1BFF	GPIOG
	0x4002 1400～0x4002 17FF	GPIOF
	0x4002 1000～0x4002 13FF	GPIOE
	0x4002 0C00～0x4002 0FFF	GPIOD
	0x4002 0800～0x4002 0BFF	GPIOC
	0x4002 0400～0x4002 07FF	GPIOB
	0x4002 0000～0x4002 03FF	GPIOA
APB2	0x4001 6800～0x4001 6BFF	LCD-TFT
	0x4001 5800～0x4001 5BFF	SAI1
	0x4001 5400～0x4001 57FF	SPI6
	0x4001 5000～0x4001 53FF	SPI5
	0x4001 4800～0x4001 4BFF	TIM11
	0x4001 4400～0x4001 47FF	TIM10
	0x4001 4000～0x4001 43FF	TIM9
	0x4001 3C00～0x4001 3FFF	EXTI
	0x4001 3800～0x4001 3BFF	SYSCFG
	0x4001 3400～0x4001 37FF	SPI4
	0x4001 3000～0x4001 33FF	SPI1
	0x4001 2C00～0x4001 2FFF	SDIO
	0x4001 2000～0x4001 23FF	ADC1—ADC2—ADC3
	0x4001 1400～0x4001 17FF	USART6
	0x4001 1000～0x4001 13FF	USART1
	0x4001 0400～0x4001 07FF	TIM8
	0x4001 0000～0x4001 03FF	TIM1

续表

总　线	起始地址	外　设
APB1	0x4000 3400～0x4000 37FF	I2S2ext
	0x4000 3000～0x4000 33FF	IWDG
	0x4000 2C00～0x4000 2FFF	WWDG
	0x4000 2800～0x4000 2BFF	RTC & BKP 寄存器
	0x4000 2400～0x4000 27FF	保留
	0x4000 2000～0x4000 23FF	TIM14
	0x4000 1C00～0x4000 1FFF	TIM13
	0x4000 1800～0x4000 1BFF	TIM12
	0x4000 1400～0x4000 17FF	TIM7
	0x4000 1000～0x4000 13FF	TIM6
	0x4000 0C00～0x4000 0FFF	TIM5
	0x4000 0800～0x4000 0BFF	TIM4
	0x4000 0400～0x4000 07FF	TIM3
	0x4000 0000～0x4000 03FF	TIM2

每个 GPIO 具有 4 个 32 位配置寄存器(GPIOx_MODER、GPIOx_OTYPER、GPIOx_OSPEEDR、GPIOx_PUPDR),2 个 32 位数据寄存器(GPIOx_IDR、GPIOx_ODR),1 个 32 位置位/复位寄存器(GPIOx_BSRR),1 个 32 位锁定寄存器(GPIOx_LCKR)和 2 个 32 位复用功能(Alternative Function,AF)寄存器(GPIOx_AFRH、GPIOx_AFRL)。

1. 配置寄存器

每个 GPIO 具有 4 个 32 位存储器映射的配置寄存器(GPIOx_MODER、GPIOx_OTYPER、GPIOx_OSPEEDR、GPIOx_PUPDR),可配置多达 16 个 GPIO。GPIOx_MODER 寄存器用于选择 GPIO 方向(输入、输出、AF、模拟)。GPIOx_OTYPER 和 GPIOx_OSPEEDR 寄存器分别用于选择输出类型(推挽或开漏)和速度(无论 GPIO 方向如何,都会直接将 GPIO 速度引脚连接到相应的 GPIOx_OSPEEDR 寄存器位)。无论 GPIO 用于输入还是输出,GPIOx_PUPDR 寄存器都用于选择上拉/下拉电阻。

2. 数据寄存器

每个 GPIO 具有 2 个 32 位数据寄存器:输入和输出数据寄存器,即 GPIOx_IDR 和 GPIOx_ODR。GPIOx_ODR 用于存储待输出数据,可对其进行读/写访问。通过 GPIO 输入的数据存储到输入数据寄存器 GPIOx_IDR 中,它是一个只读寄存器。

3. 置位/复位寄存器

置位/复位寄存器(GPIOx_BSRR)是一个 32 位寄存器,它允许应用程序在输出数据寄存器 GPIOx_ODR 中对各个单独的数据位执行置位和复位操作。置位/复位寄存器的大小是 GPIOx_ODR 的 2 倍。GPIOx_ODR 中的每个数据位对应 GPIOx_BSRR 中的两个控制位:BSRR(i) 和 BSRR(i+SIZE)。当写入 1 时,BSRR(i)位会置位对应的 ODR(i)位;BSRR(i+SIZE)位会清零 ODR(i)对应的位。在 GPIOx_BSRR 中向任何位写入 0 都不会对 GPIOx_ODR 中的对应位产生任何影响。如果在 GPIOx_BSRR 中同时尝试对某个位执行置位和清零操作,则置位操作优先。使用 GPIOx_BSRR 寄存器更改 GPIOx_ODR 中各位的值是一个“单次”操作,不会锁定 GPIOx_ODR 位。随时都可以直接访问 GPIOx_ODR

位。GPIOx_BSRR 寄存器允许对 GPIO 进行原子操作(原子操作指不会被中断的操作,在多任务环境下非常重要),在对 GPIOx_ODR 进行位操作时,软件无须禁止中断:在一次原子 AHB1 写访问中,可以修改一位或多位。

4. 锁定寄存器

GPIO 锁定寄存器 GPIOx_LCKR 用于锁定 GPIO 的配置,以防止意外更改。一旦锁定寄存器被设置,GPIO 的配置(如模式、输出类型、速度、上拉/下拉等)就不能再被更改,除非先解锁该寄存器,可以锁定的寄存器包括 GPIOx_MODER、GPIOx_OTYPER、GPIOx_OSPEEDR、GPIOx_PUPDR、GPIOx_AFRL 和 GPIOx_AFRH。GPIOx_LCKR 是一个 32 位的寄存器。它的每位都对应一个 GPIO 引脚,当某位被设置为 1 时,对应的 GPIO 引脚配置就被锁定。

5. 复用功能寄存器

GPIO 复用功能低位寄存器 GPIOx_AFRL 和 GPIO 复用功能高位寄存器 GPIOx_AFRH 用于选择每个 GPIO 可用的 16 个复用功能。GPIOx_AFRL 寄存器用于选择 GPIO 低 8 位(即 GPIOx0～GPIOx7)的复用功能,GPIOx_AFRH 寄存器用于选择 GPIO 高 8 位(即 GPIOx8～GPIOx15)的复用功能。由于复用选择信号由复用功能输入和复用功能输出共用,所以只需为每个 GPIO 的复用功能输入/输出选择一个通道即可,对于每个 GPIO 而言,应用程序一次只能为其选择一个可用的外设功能。


视频讲解

3.2.2 STM32F407 GPIO 引脚模式

STM32F407 每个 GPIO 位可自由编程,但是 GPIO 寄存器必须作为 32 位字、半字或字节访问。GPIOx_BSRR 寄存器的目的是允许对任何 GPIO 寄存器进行原子读取/修改访问,通过这种方式,在读取和修改访问之间发生中断请求也不会有问题。GPIO 位的基本结构如图 3-1 所示。

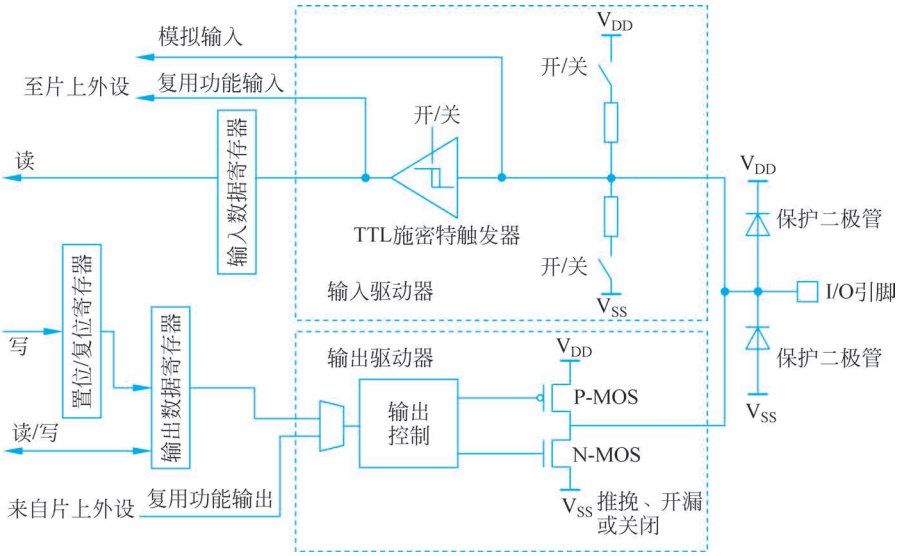

图 3-1 GPIO 位的基本结构

1. 输入/输出模式

GPIO 引脚输入/输出模式具体如下。

(1) 浮空输入：数字输入，读取引脚电平，若引脚悬空，则电平不确定；

(2) 上拉输入：数字输入，读取引脚电平，内部连接上拉电阻，悬空时默认高电平；

(3) 下拉输入：数字输入，读取引脚电平，内部连接下拉电阻，悬空时默认低电平；

(4) 模拟输入：数字输入，GPIO 无效，引脚直接接入内部模数转换器；

(5) 开漏输出：数字输出，输出引脚电平，高电平为高阻态，低电平接 V_{SS}；

(6) 推挽输出：数字输出，输出引脚电平，高电平接 V_{DD}，低电平接 V_{SS}；

(7) 复用开漏输出：数字输出，由片上外设控制，高电平为高阻态，低电平接 V_{SS}；

(8) 复用推挽输出：数字输出，由片上外设控制，高电平接 V_{DD}，低电平接 V_{SS}。

在复位期间及复位刚刚完成后，复用功能尚未激活，GPIO 被配置为浮空输入模式。复位后，调试引脚处于复用功能上拉/下拉状态：引脚 PA15 为 JTDI 处于上拉状态；引脚 PA14 为 JTCK/SWCLK 处于下拉状态；引脚 PA13 为 JTMS/SWDAT 处于下拉状态；引脚 PB4 为 NJTRST 处于上拉状态；引脚 PB3 为 JTDO 处于浮空状态。

GPIO 作为输入时，输出缓冲器被关闭，施密特触发器输入被打开，根据 GPIOx_PUPDR 寄存器中的值决定是否打开上拉和下拉电阻，输入数据寄存器每隔 1 个 AHB1 时钟周期对 GPIO 引脚上的数据进行一次采样，对输入数据寄存器的读访问可获取 GPIO 状态。GPIO 位的输入配置如图 3-2 所示。

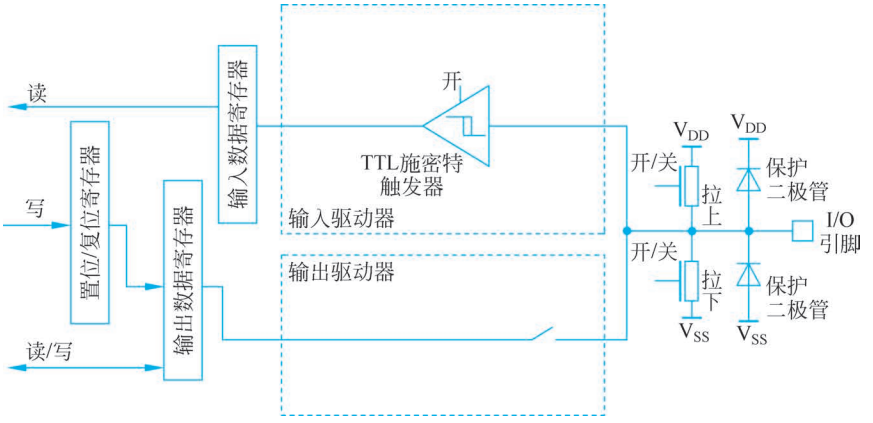

图 3-2　GPIO 位的输入配置

GPIO 作为输出时，输出缓冲器被打开，开漏模式输出低电平时 N-MOS 导通，而输出为高电平时端口保持高阻态(P-MOS 断开)；推挽模式输出低电平时 N-MOS 导通，而输出高电平时 P-MOS 导通，施密特触发器输入被打开，根据 GPIOx_PUPDR 寄存器中的值决定是否打开弱上拉电阻和下拉电阻，输入数据寄存器每隔 1 个 AHB1 时钟周期对 GPIO 引脚上的数据进行一次采样，对输入数据寄存器的读访问可获取 GPIO 状态，对输出数据寄存器的读访问可获取最后的写入值。GPIO 位的输出配置如图 3-3 所示。

2. 复用及模拟模式

GPIO 通过多路复用器，只允许一个外设的复用功能连接到 GPIO 引脚一次，因而，共享同一 GPIO 引脚的外围设备之间就不会发生冲突。每个 GPIO 引脚都有一个多路复用

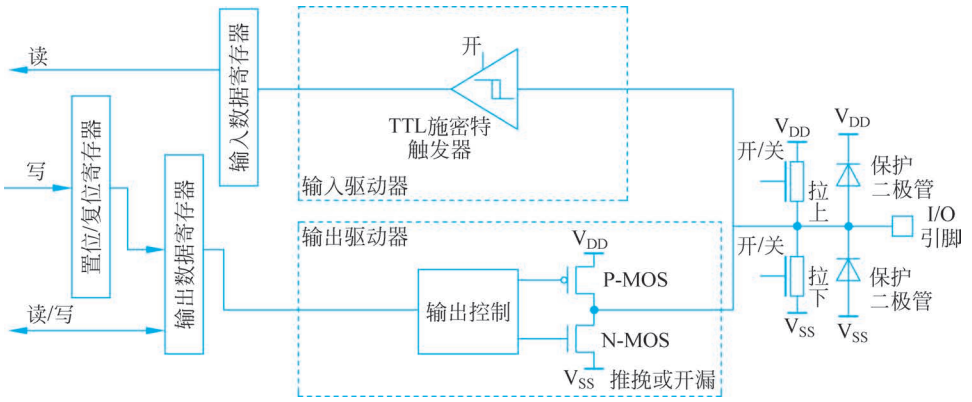

图 3-3　GPIO 位的输出配置

器，该多路复用器具有 16 个复用功能输入（AF0～AF15），可以通过 GPIOx_AFRL（用于引脚 0～7）和 GPIOx_AFRH（用于引脚 8～15）进行寄存器配置。重置后，所有 GPIO 都连接到系统的复用功能 0（AF0）；外围设备的复用功能从 AF1 映射到 AF13；Cortex-M4F EVENTOUT 映射在 AF15 上。STM32F407 引脚复用功能选择如图 3-4 所示。

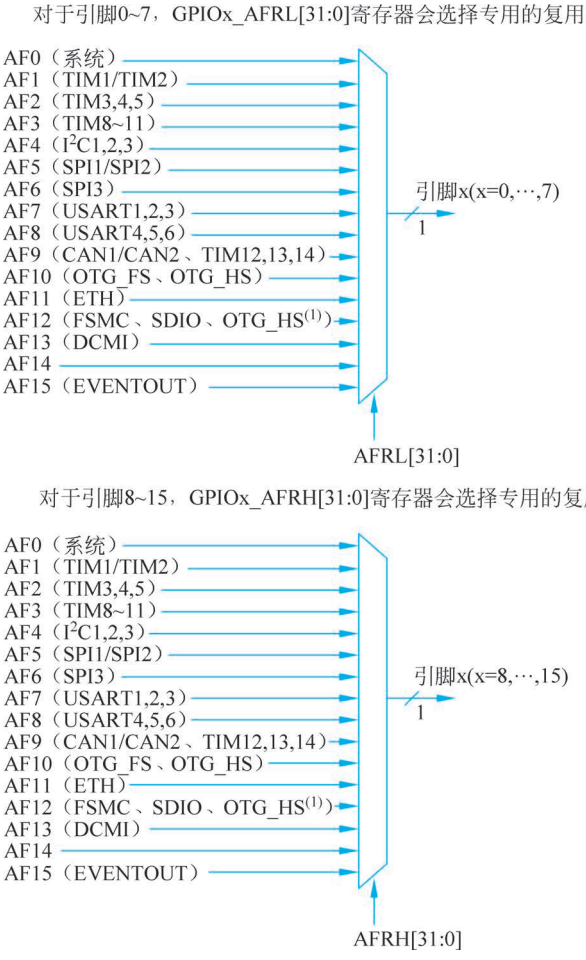

图 3-4　STM32F407 引脚复用功能选择

对于模数转换器和数模转换器，在 GPIOx_MODER 寄存器中将所需 GPIO 配置为模拟通道。对于其他外设，在 GPIOx_MODER 寄存器中将所需 GPIO 配置为复用功能，通过 GPIOx_OTYPER、GPIOx_PUPDR 和 GPIOx_OSPEEDER 寄存器，分别选择类型、上拉/下拉以及输出速度，在 GPIOx_AFRL 或 GPIOx_AFRH 寄存器中，将 GPIO 连接到所需引脚。

对 GPIO 作为复用功能时，可将输出缓冲器配置为开漏或推挽，输出缓冲器由来自外设的信号驱动（发送器使能和数据），施密特触发器输入被打开，根据 GPIOx_PUPDR 寄存器中的值决定是否打开弱上拉电阻和下拉电阻，输入数据寄存器每隔 1 个 AHB1 时钟周期对 GPIO 引脚上的数据进行一次采样，对输入数据寄存器的读访问可获取 GPIO 状态。GPIO 位的复用功能配置如图 3-5 所示。

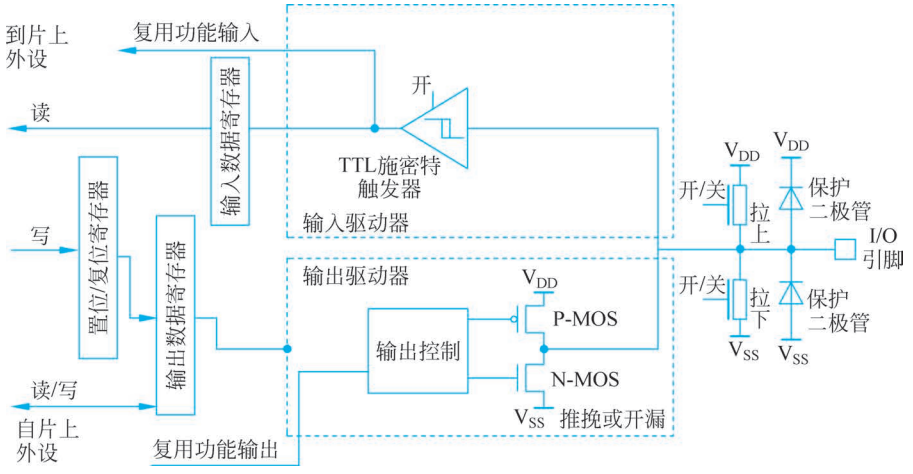

图 3-5　GPIO 位的复用功能配置

对 GPIO 作为模拟配置时，输出缓冲器被禁止，施密特触发器输入停用，GPIO 引脚的每个模拟输入的功耗变为 0，施密特触发器的输出被强制处理为恒定值 0，弱上拉和下拉电阻被关闭，对输入数据寄存器的读访问值为 0。GPIO 位的高阻态模拟输入配置如图 3-6 所示。

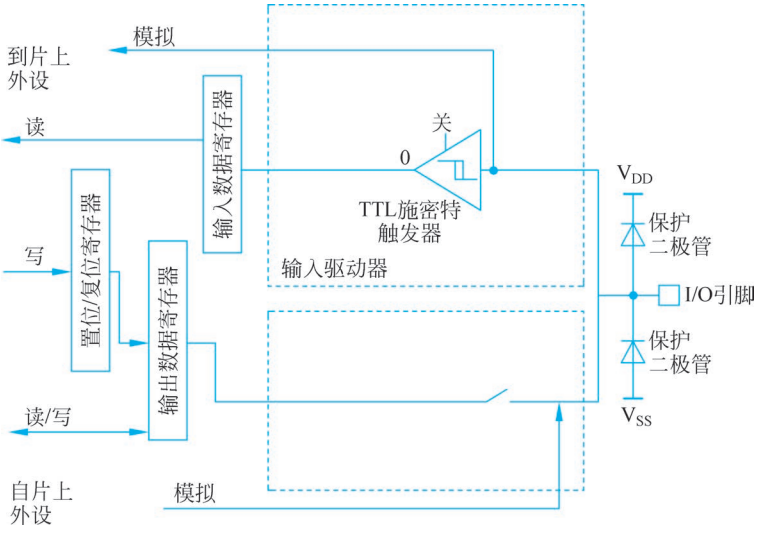

图 3-6　GPIO 位的高阻态模拟输入配置


视频讲解

3.3 STM32F407 GPIO HAL 库函数

STM32F407 GPIO HAL 库函数如表 3-2 所示。

表 3-2 STM32F407 GPIO HAL 库函数

函 数 名	功 能 描 述
HAL_GPIO_Init()	初始化 GPIO 引脚
HAL_GPIO_DeInit()	GPIO 引脚恢复为默认状态
HAL_GPIO_ReadPin()	读取 GPIO 引脚的电平状态
HAL_GPIO_WritePin()	设置 GPIO 引脚的电平状态
HAL_GPIO_TogglePin()	翻转 GPIO 引脚的电平状态
HAL_GPIO_LockPin()	锁定 GPIO 引脚的状态，防止其被更改

1. 初始化 GPIO 引脚

函数原型为 void HAL_GPIO_Init(GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)，用于初始化指定的 GPIO 引脚。

其中，GPIOx 指定要初始化的 GPIO；GPIO_Init 提供初始化 GPIO 引脚所需的各种配置参数。

函数 HAL_GPIO_Init()无返回值。

结构体 GPIO_InitTypeDef 成员如下。

(1) Pin：指定要配置的引脚。

(2) Mode：设置引脚的工作模式，如输入、输出、复用功能、外部中断等。

(3) Pull：配置引脚的上下拉电阻。

(4) Speed：设置引脚的速度等级。

(5) Alternate：当引脚配置为复用功能时，指定要连接的外设。

2. GPIO 引脚恢复为默认状态

函数原型为 void HAL_GPIO_DeInit(GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)，用于将指定的 GPIO 引脚恢复为默认状态。

其中，GPIOx 指定要恢复的 GPIO，GPIO_Pin 指定要恢复的引脚。

函数 HAL_GPIO_DeInit()无返回值。

3. 读取 GPIO 引脚的电平状态

函数原型为 GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)，用于读取指定的 GPIO 引脚电平状态。

其中，GPIOx 指定要读取电平状态的 GPIO，GPIO_Pin 指定要读取的引脚。

函数 HAL_GPIO_ReadPin()的返回值为引脚的电平状态，GPIO_PIN_SET 表示高电平，GPIO_PIN_RESET 表示低电平。

4. 设置 GPIO 引脚的电平状态

函数原型为 void HAL_GPIO_WritePin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)，用于设置指定的 GPIO 引脚的电平状态。

其中，GPIOx 指定要设置电平状态的 GPIO，GPIO_Pin 指定要设置的引脚，PinState 指

要设置的电平状态(GPIO_PIN_SET 指设置为高电平,GPIO_PIN_RESET 指设置为低电平)。

函数 HAL_GPIO_WritePin()无返回值。

5. 翻转 GPIO 引脚的电平状态

函数原型为 void HAL_GPIO_TogglePin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin),用于翻转指定的 GPIO 引脚的电平状态。

其中,GPIOx 指定要翻转电平状态的 GPIO,GPIO_Pin 指定要翻转的引脚。

函数 HAL_GPIO_TogglePin()无返回值。

6. 锁定 GPIO 引脚的状态

函数原型为 HAL_StatusTypeDef HAL_GPIO_LockPin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin),用于锁定指定的 GPIO 引脚的状态,防止其被更改。

其中,GPIOx 指定要锁定的 GPIO,GPIO_Pin 指定要锁定的引脚。

函数 HAL_GPIO_LockPin()的返回值如下:HAL_OK 表示函数执行成功,HAL_ERROR 表示函数执行失败,HAL_BUSY 表示函数当前正忙,HAL_TIMEOUT 表示函数执行超时。

3.4　GPIO 实例


视频讲解

本节实例为通过按键控制发光二极管闪烁。通过 GPIO 引脚输出高低电平控制发光二极管,并通过改变高低电平的时间间隔实现二极管闪烁。

3.4.1　STM32CubeMX 工程

1. 新建工程及芯片配置

双击软件图标启动 STM32CubeMX 软件。STM32CubeMX 软件界面如图 3-7 所示,切换到 File 选项卡,并单击 New Project 创建新工程。

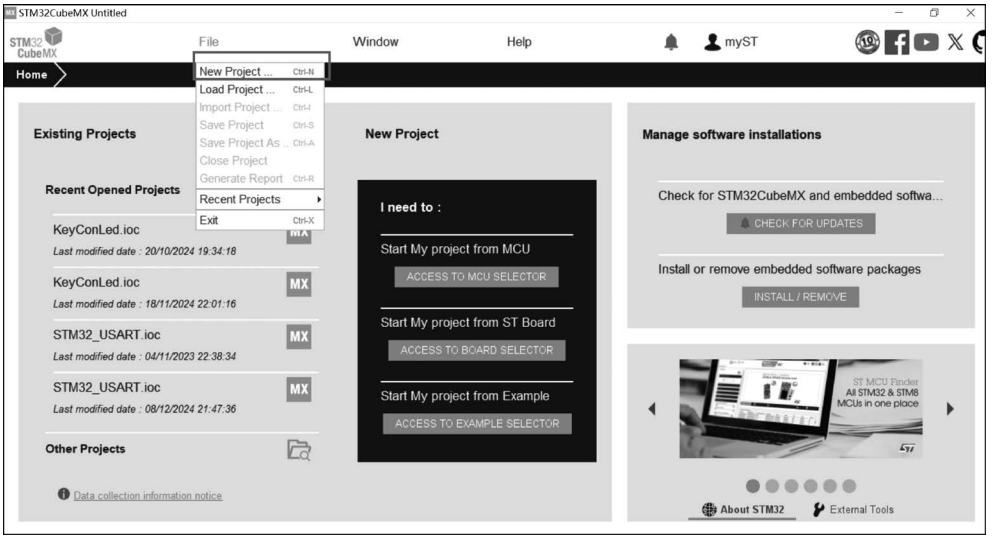

图 3-7　STM32CubeMX 软件界面

STM32CubeMX 新工程界面如图 3-8 所示。新工程需要确定所使用的芯片型号，在 Commercial Part Number 文本框中输入使用的芯片型号，例如，输入 STM32F407V，检索出与输入关键词相近且封装不同的芯片型号，选择所用的芯片型号 STM32F407VET6。

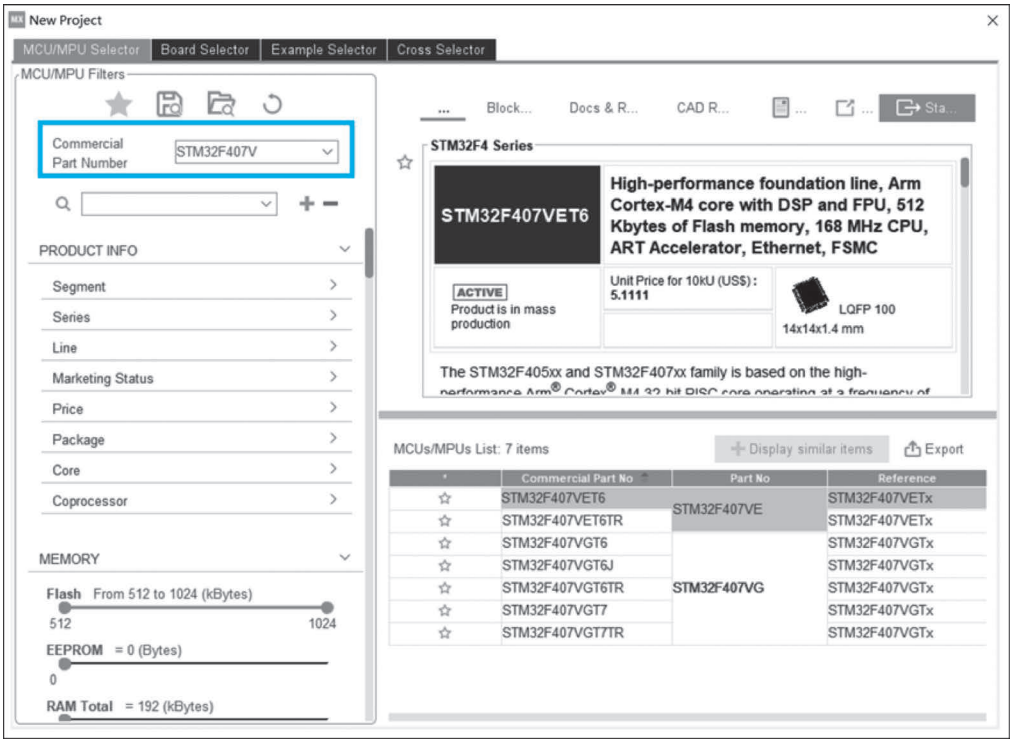

图 3-8　STM32CubeMX 新工程界面

单击右上角 Start Project，在 Pinout & Configuration 界面中给出 STM32F407VET6 芯片引脚视图，如图 3-9 所示。

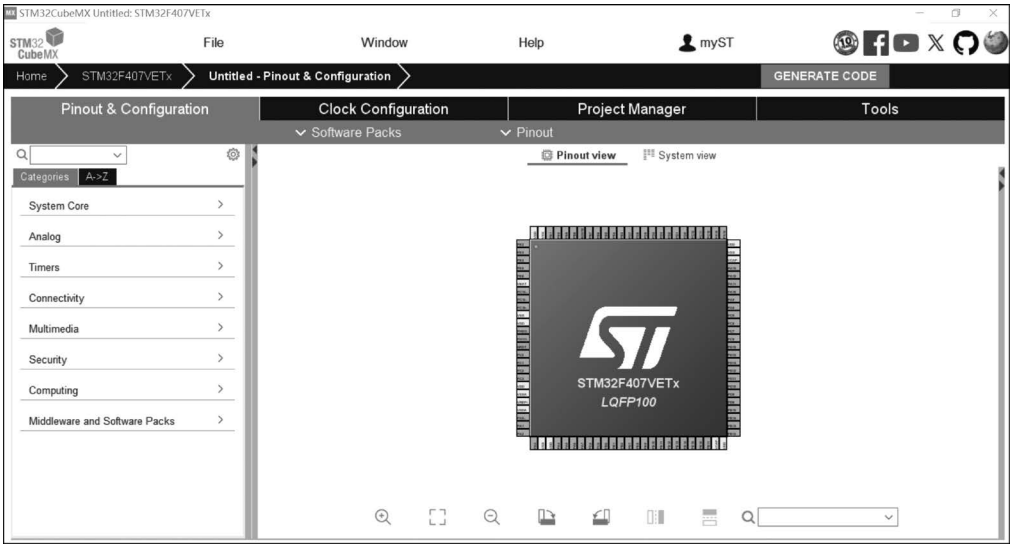

图 3-9　Pinout & Configuration 界面

选择左侧 System Core 中的 SYS，SYS Mode and Configuration 配置如图 3-10 所示，选择 Debug 选项为 Serial Wire。

图 3-10　SYS Mode and Configuration 配置

2. 配置系统时钟

芯片的时钟源通过选择 RCC 设置，选择 System Core 中的 RCC。RCC Mode and Configuration 如图 3-11 所示，配置高速外部时钟 HSE 和低速外部时钟 LSE。通常来说，高速外部时钟和低速外部时钟的选项卡中包括禁止（Disable）、旁路时钟源（Bypass Clock Source），以及外部晶体陶瓷谐振器（Crystal/Ceramic Resonator）3 种选项。其中，Disable 选项为不启用外部时钟源；Bypass Clock Source 选项是不使用外部晶体陶瓷谐振器或其他

图 3-11　RCC Mode and Configuration

的时钟源，而是直接通过外部导入时钟信号；Crystal/Ceramic Resonator 选项为使用外部晶体谐振器，通过外部晶体谐振器与芯片内部时钟的驱动电路协作形成时钟源，精度较高。

设置高速外部时钟 HSE，选择 Crystal/Ceramic Resonator，如图 3-12 所示，PH0 引脚和 PH1 引脚被占用，分别用于 RCC_OSC_IN 和 RCC_OSC_OUT 的连接。

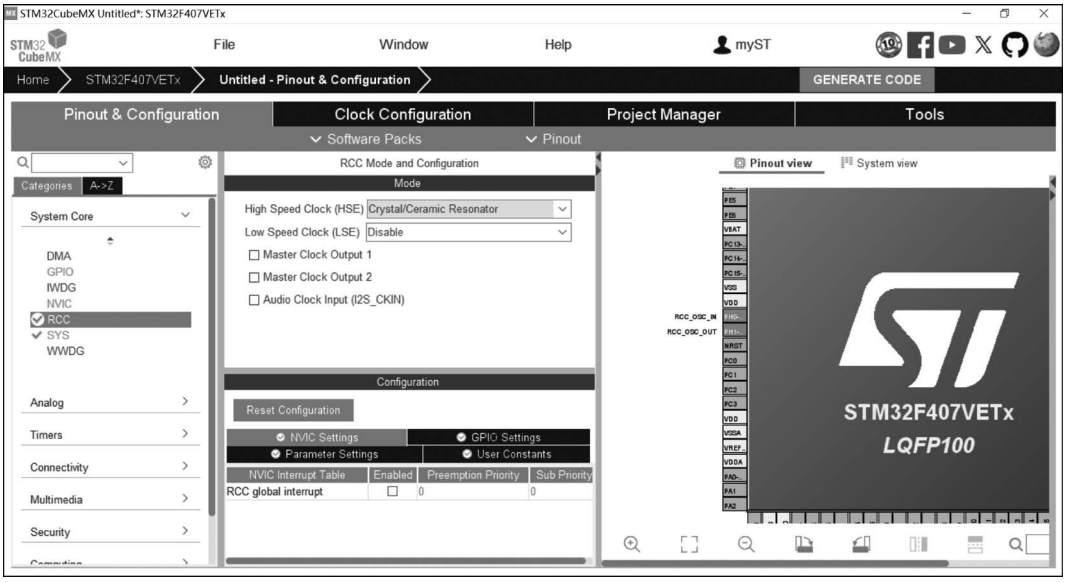

图 3-12　选择 Crystal/Ceramic Resonator

切换到 Clock Configuration 选项卡，如图 3-13 所示。

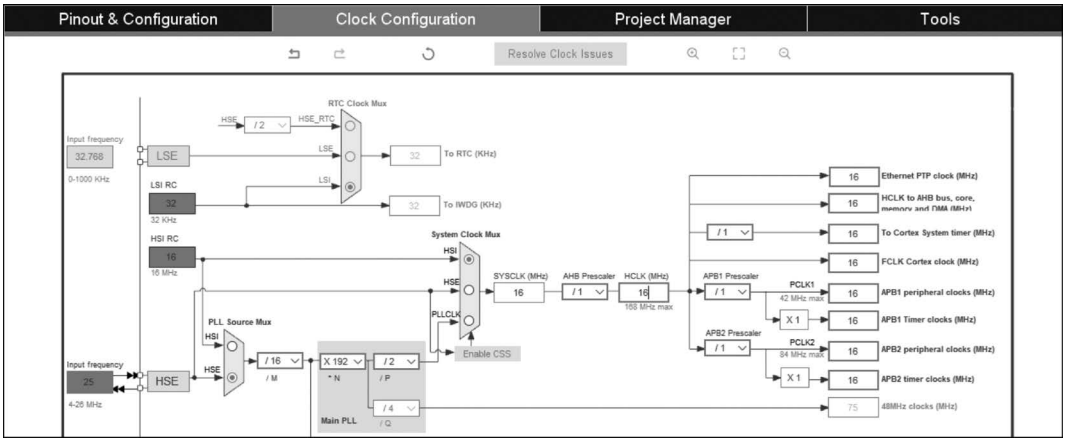

图 3-13　Clock Configuration 选项卡

HSE 外部时钟源输入频率（Input Frequency）取值范围为 4～26MHz，默认值为 25MHz，输入设置频率为 8MHz。选用 16MHz 内部高速晶振 HSI，在 HCLK（MHz）文本框中输入 16 后按 Enter 键，Clock Wizard 界面如图 3-14 所示，单击 OK 按钮，自动完成系统时钟配置。配置系统

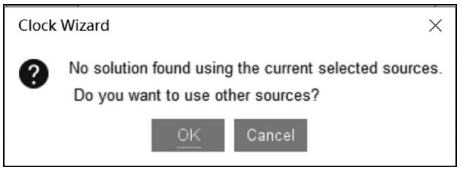

图 3-14　Clock Wizard 界面

工作频率为 16MHz，如图 3-15 所示。

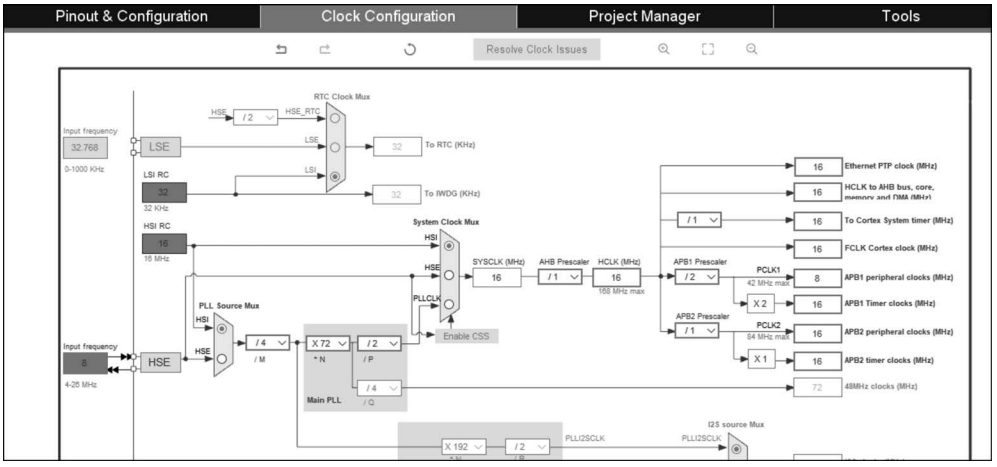

图 3-15　配置系统工作频率为 16MHz

3. 配置 GPIO 引脚工作模式

切换到 Pinout & Configuration 选项卡，配置 GPIO PA0 引脚工作模式，如图 3-16 所示。

PA0 引脚工作模式如下。

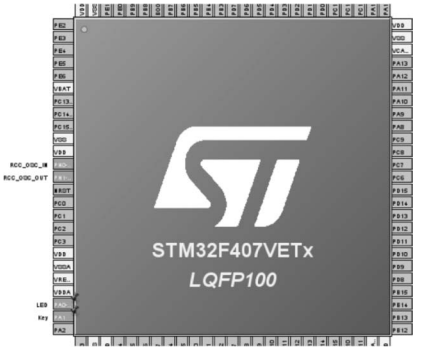

图 3-16　GPIO PA0 引脚工作模式

（1）Reset_State：复位状态，设置为低电平。

（2）ADC1_IN0：模数转换器 1 通道 0 的数据采集。

（3）ADC2_IN0：模数转换器 2 通道 0 的数据采集。

（4）ADC3_IN0：模数转换器 3 通道 0 的数据采集。

（5）ETH_CRS：以太网载波监听信号。

（6）SYS_WKUP：系统唤醒。

（7）TIM2_CH1：通用定时器 2 的通道 1。

（8）TIM2_ETR：TIM2 定时器的外部时钟源模式。

（9）TIM5_ CH1：通用定时器 5 的通道 1。

（10）TIM8_ETR：TIM8 定时器的外部时钟源模式。

（11）UART4_TX：串口 4 通信发送端。

（12）USART2_CTS：USART2 通信接口清除发送。

（13）GPIO_Input：通用输入。

（14）GPIO_Output：通用输出。

（15）GPIO_Analog：通用模拟信号输出。

（16）EVENTOUT：事件输出。

（17）GPIO_EXT10：中断 EXT10。

本节实例通过单个 GPIO 引脚输出高低电平控制发光二极管，并通过改变高低电平的时间间隔实现二极管闪烁，因而将 PA0 引脚工作模式配置为通用输出功能（GPIO_Output）。

4. GPIO 引脚参数配置

切换到 System Core 中的 GPIO，GPIO Mode and Configuration 界面如图 3-17 所示，配置 PA0 引脚详细参数。GPIO output level 选项中可以将引脚电平设置为高电平或低电平；GPIO mode 下拉列表框中选择输出模式为推挽或开漏；GPIO Pull-up/Pull-down 下拉列表框中选择上拉电阻、下拉电阻、无上拉和下拉；Maximum output speed 下拉列表框中选择引脚输出速度为低速、中速、高速；User Label 为用户标签，用于设置引脚名称。本节实例将 PA0 引脚配置为高电平输出、推挽式输出模式、上拉电阻、高速输出，引脚标签为 LED。

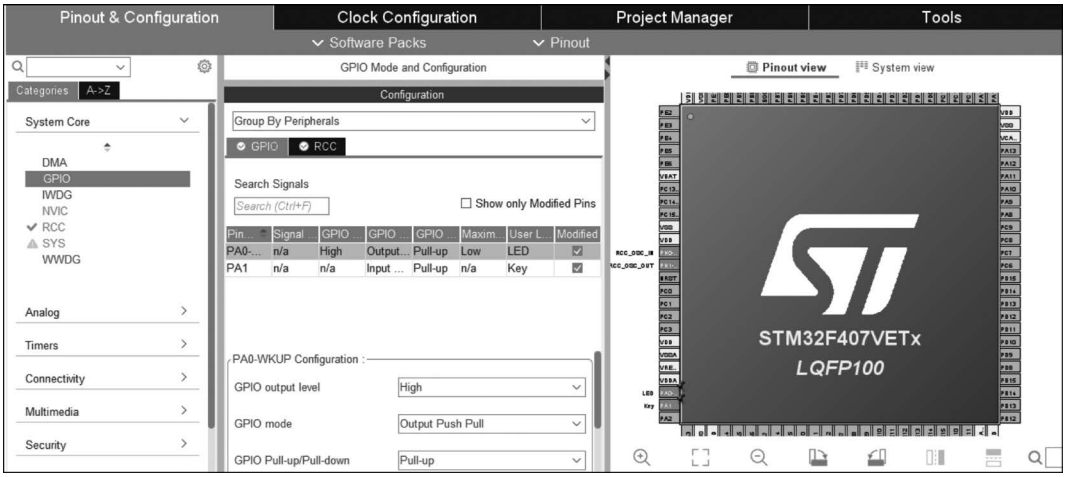

图 3-17　GPIO Mode and Configuration 界面

将 PA1 引脚配置为输入功能(GPIO_Input)，如图 3-18 所示。

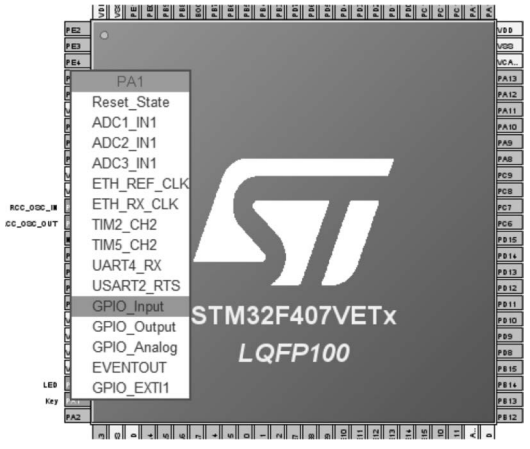

图 3-18　PA1 引脚配置

本节实例将 PA1 引脚配置为上拉电阻，引脚标签为 key，如图 3-19 所示。

5. 工程管理

切换到 Project Manager 选项卡，配置工程参数如图 3-20 所示。在 Project Name 文本框里输入项目名称，如 LED；Project Location 为当前工程的存放路径；Toolchain/IDE 选择代码的开发环境，由于采用 Keil MDK 作为集成开发环境，因此选择 MDK-ARM；Min Version 选择版本为 V5.32。

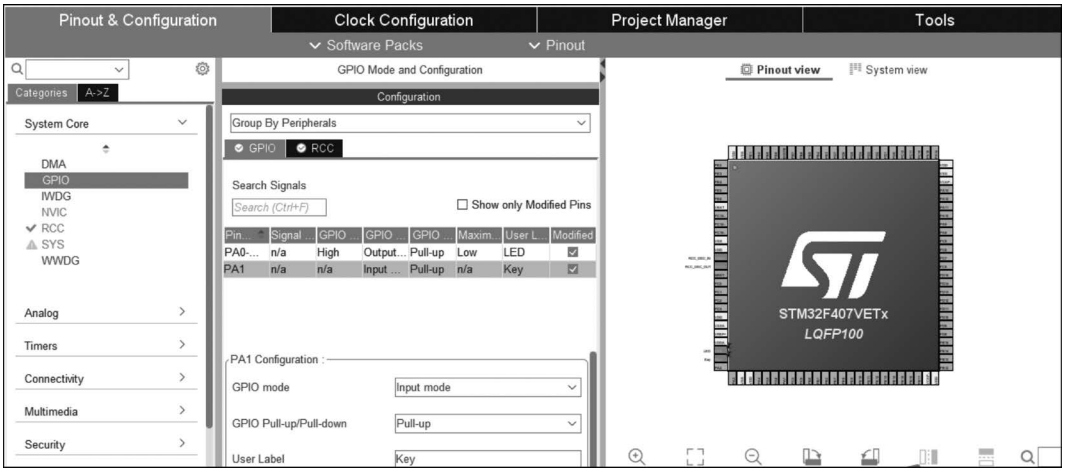

图 3-19　PA1 引脚配置

图 3-20　配置工程参数

切换到左侧的 Code Generator 选项卡，代码生成器配置如图 3-21 所示。勾选 Copy only the necessary library files、Generate peripheral initialization as a pair of '.c/.h' files per peripheral、Keep User Code when re-generating、Delete previously generated files when not re-generated 对应的复选框。

参数设置完成后单击 GENERATE CODE，生成源代码如图 3-22 所示。代码生成提示框如图 3-23 所示。

3.4.2　Keil MDK 程序

单击图 3-23 中的 Open Project，Keil MDK 工程文件界面如图 3-24 所示，单击左侧工程目录树中 Application/User/Core 文件下的 main.c 文件，右侧显示 main.c 文件源代码。

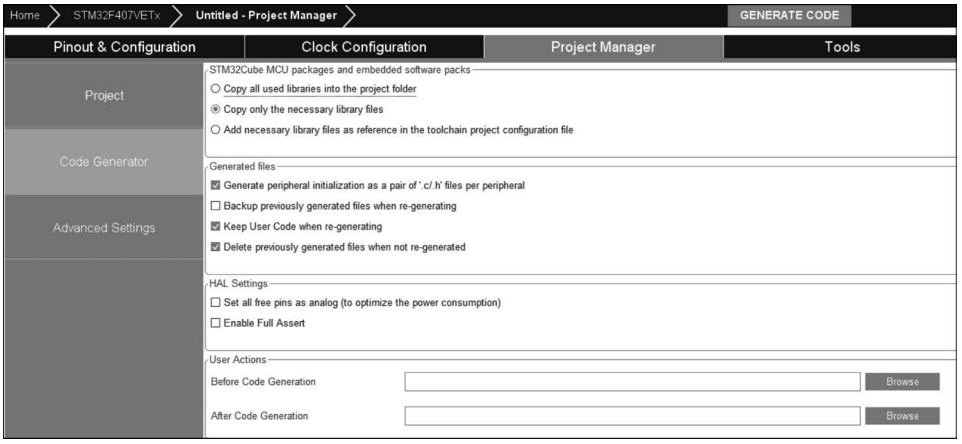

图 3-21　代码生成器配置

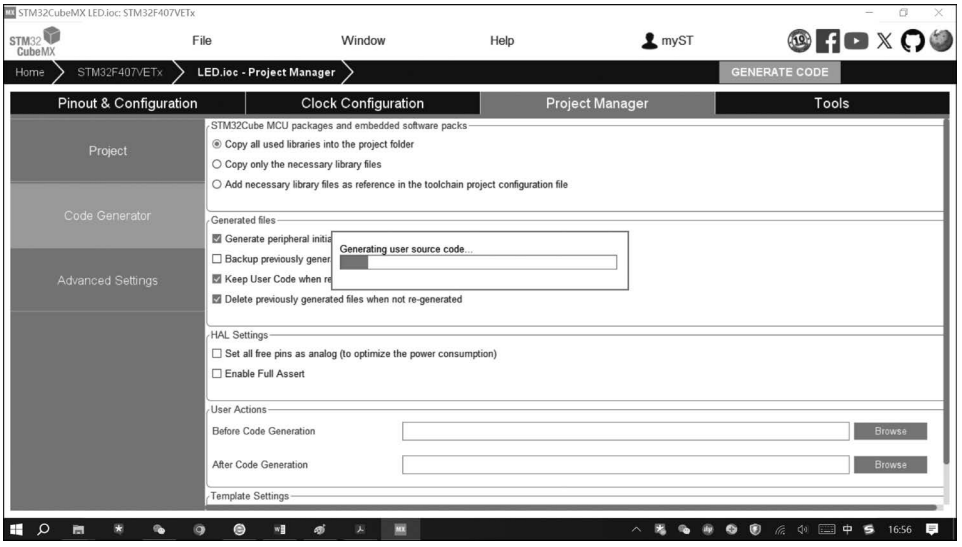

图 3-22　生成源代码

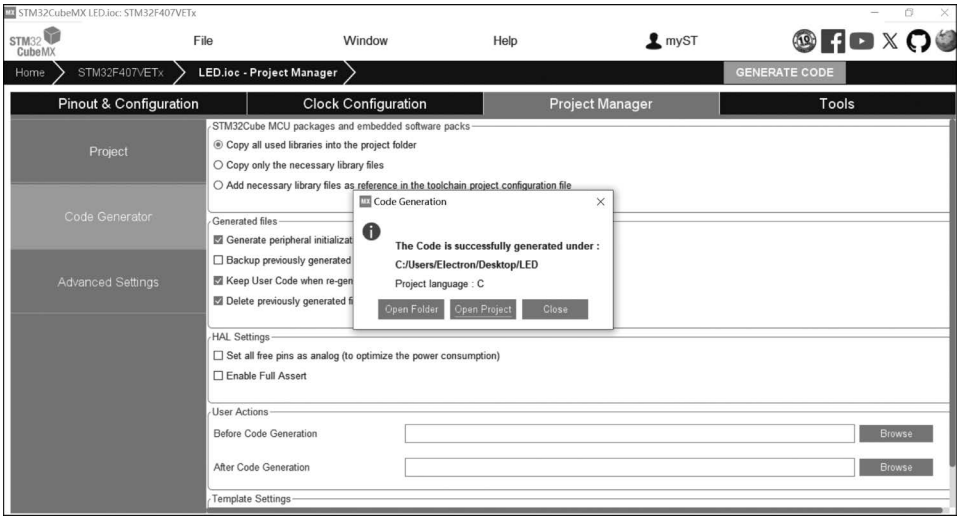

图 3-23　代码生成提示框

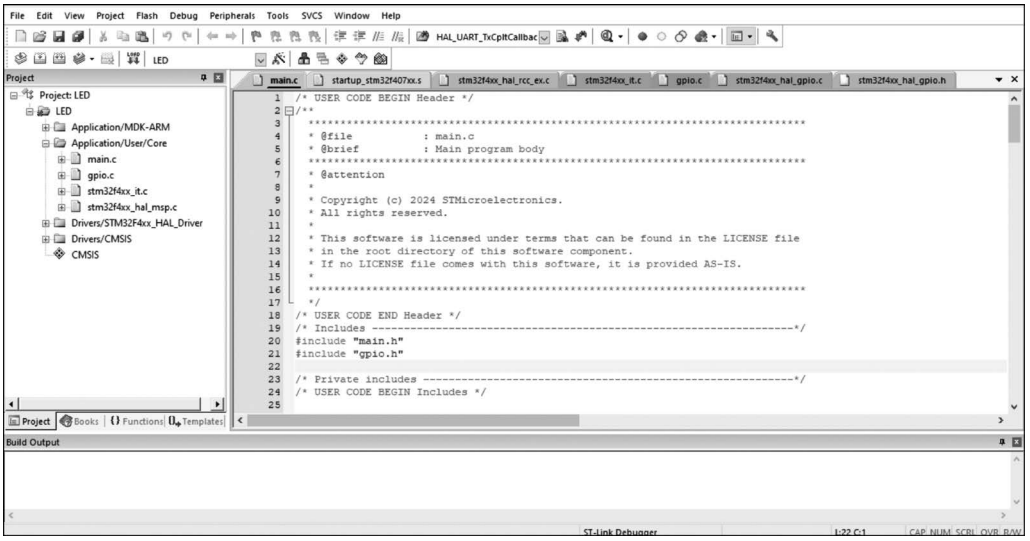

图 3-24　Keil MDK 工程文件界面

通过 STM32CubeMX 软件配置，工程文件的相关外设配置代码已大部分生成。HAL_Init() 函数会初始化 HAL 库的内部数据结构和变量；SystemClock_Config() 函数配置 STM32 微控制器的时钟系统，包括 HSI、HSE、PLL 等，以满足应用程序对时钟频率的需求；MX_GPIO_Init()函数配置指定的 GPIO 引脚，包括引脚号、模式（输入、输出、复用功能、模拟等）、速度、上拉/下拉电阻等，并初始化 GPIO 时钟，使能 GPIO 时钟，以便能够对其进行配置。

下一步根据需求编写应用逻辑代码，在 Keil MDK 软件中编写发光二极管闪烁程序代码。需要注意的是，编写的代码要在注释语句的 BEGIN 与 END 之间，因为 STM32CubeMX 重新生成代码时会自动删除不在注释语句 BEGIN 与 END 之间的代码，如下所示。

```
/* USER CODE BEGIN 2 */
  GPIO_PinState KeyStaus;
/* USER CODE END 2 */
```

在 main.c 文件的 while(1)循环中加入以下代码：

```
KeyStaus = HAL_GPIO_ReadPin(GPIOA,Key_Pin);
if(KeyStaus == GPIO_PIN_RESET)
{
  HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_0);
}
  HAL_Delay(300);
```

按键控制发光二极管闪烁程序如图 3-25 所示，在 Keil MDK 软件中对该工程进行编译和构建，生成.hex 文件。

3.4.3　Proteus 仿真电路

通过 Proteus 仿真软件构建发光二极管闪烁的仿真环境，模拟 STM32F407VET6 芯片的运行状态。

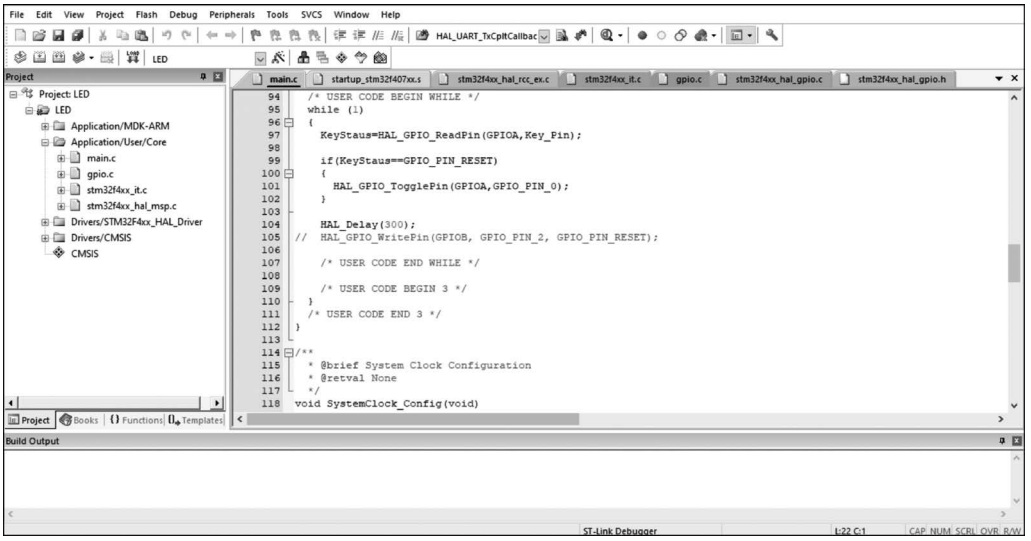

图 3-25　按键控制发光二极管闪烁程序

1. 创建 Proteus 工程

双击软件图标启动仿真软件，进入 Proteus 软件主页，单击菜单栏 File→New Project，弹出新建工程向导界面，如图 3-26 所示。

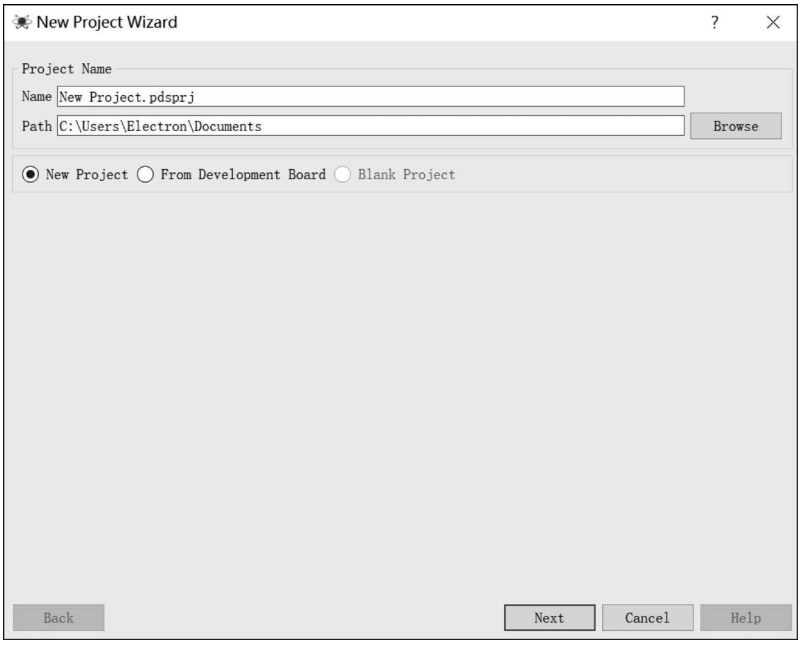

图 3-26　新建工程向导界面

设置工程名为 STM32F407_LED，并选择工程存放的位置，单击 Next 按钮，进入下一步，创建如图 3-27 所示的原理图。

可根据设计需求选择图纸大小，例程选择 DEFAULT 模板，然后单击 Next 按钮，进入下一步，PCB 布局如图 3-28 所示。

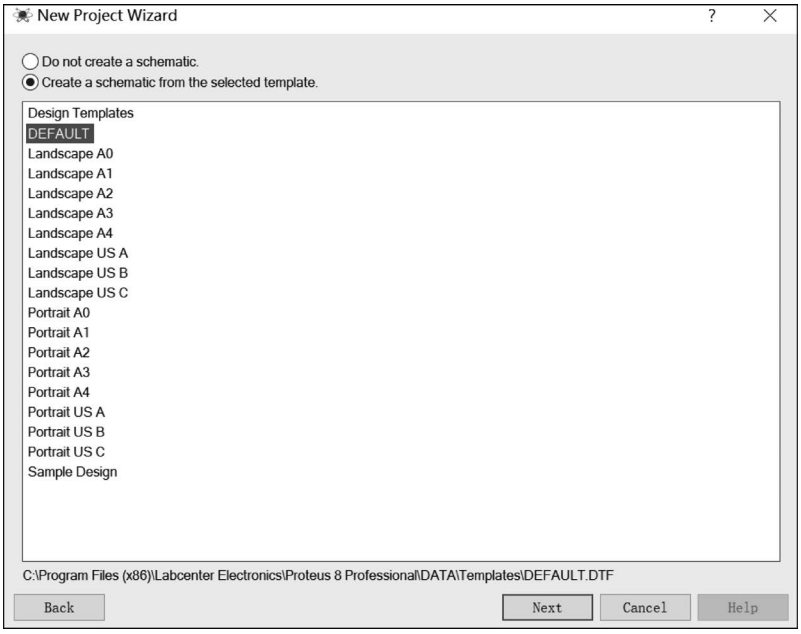

图 3-27　创建原理图

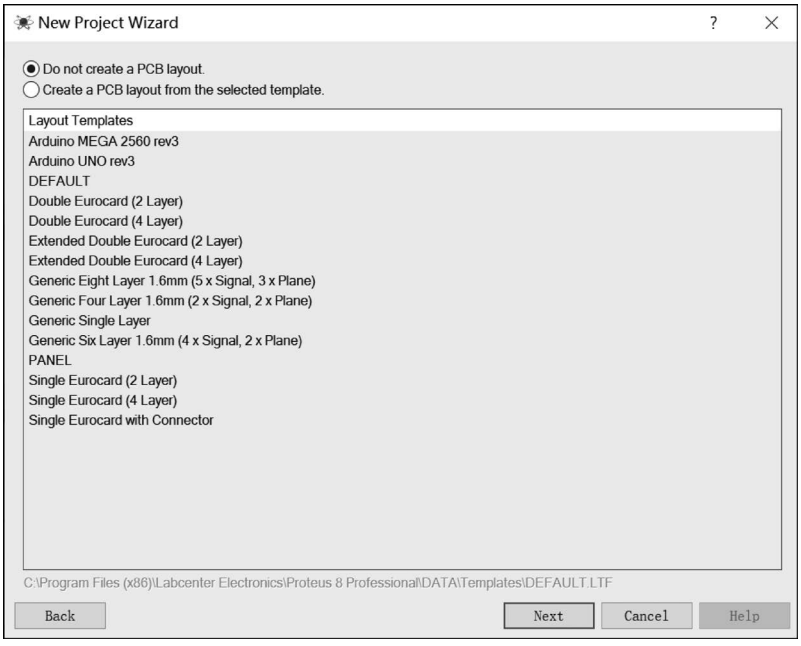

图 3-28　PCB 布局

本节实例不需要 PCB 布局，选择 Do not create a PCB layout，单击 Next 按钮，进行下一步，新建工程类型选择如图 3-29 所示。

在图 3-29 中选择 No Firmware Project，单击 Next 按钮，进入下一步，总结界面如图 3-30 所示。

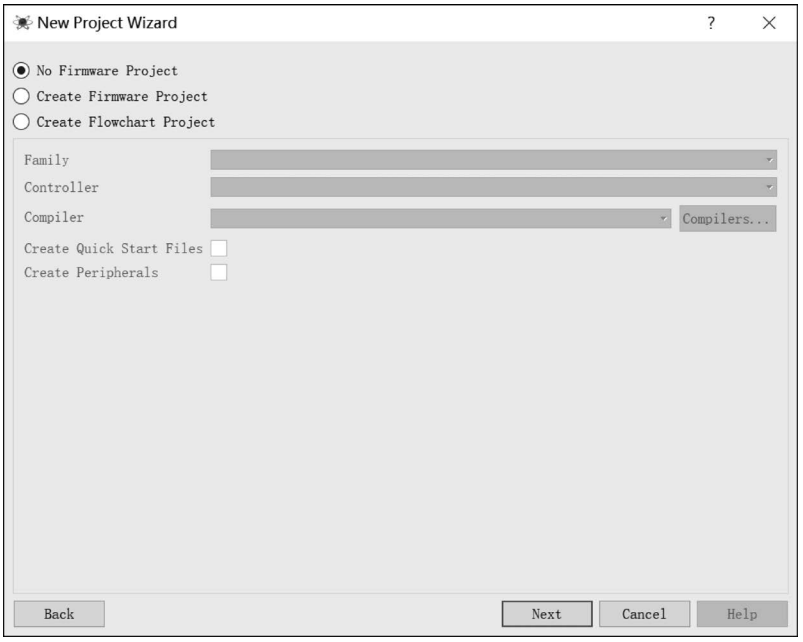

图 3-29　新建工程类型选择

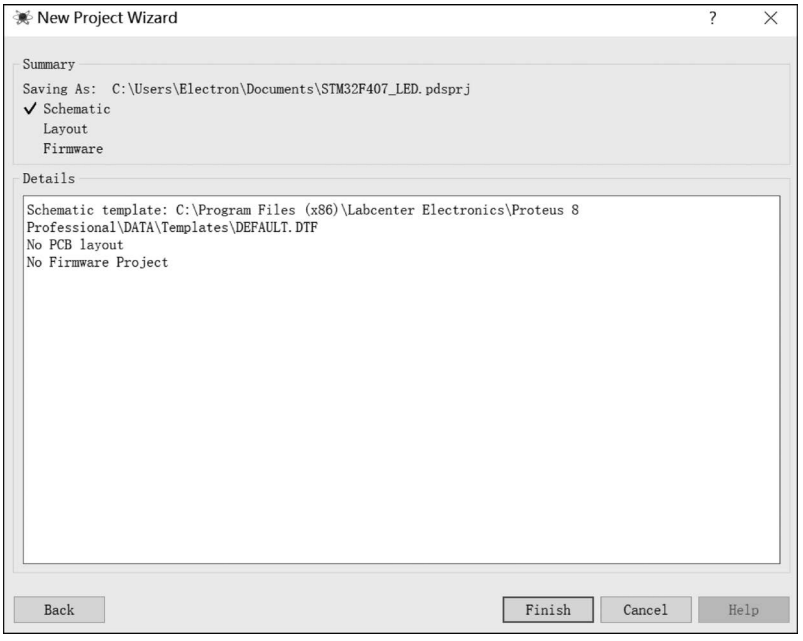

图 3-30　总结界面

2. 原理图绘制

在图 3-30 中单击 Finish 按钮，完成 Proteus 仿真工程创建，原理图绘制界面如图 3-31 所示。

单击左侧 DEVICES 中的 P 快捷键，Pick Devices 对话框如图 3-32 所示，在 Keywords 文本框中输入需要查找的元器件型号关键字。

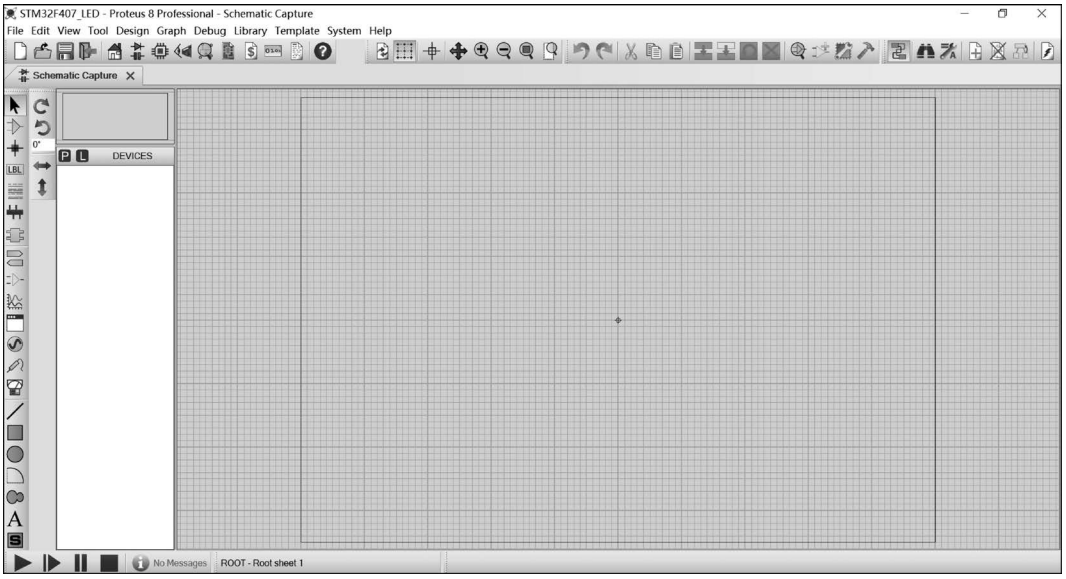

图 3-31　原理图绘制界面

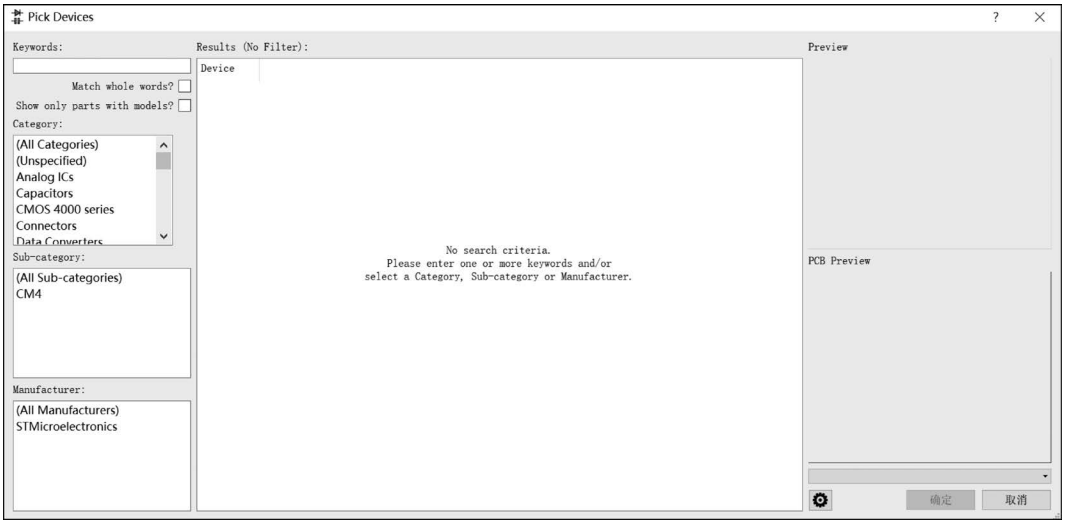

图 3-32　Pick Devices 对话框

选择 STM32F407VET6 作为仿真芯片，在 Keywords 文本框中输入 STM32F407VET6，STM32F407VET6 仿真模型如图 3-33 所示。单击“确定”按钮，将 STM32F407VET6 芯片放置于原理图中。

在 Pick Devices 的 Keywords 文本框中输入 LED-RED，发光二极管 LED-RED 仿真模型如图 3-34 所示。单击“确定”按钮，将发光二极管 LED-RED 仿真模型放置在原理图中。

在 Pick Devices 的 Keywords 文本框中输入 SWITCH，开关 SWITCH 仿真模型如图 3-35 所示。单击“确定”按钮，将开关 SWITCH 仿真模型放置在原理图中。

在 Pick Devices 的 Keywords 文本框中输入 RES，电阻 RES 仿真模型如图 3-36 所示。

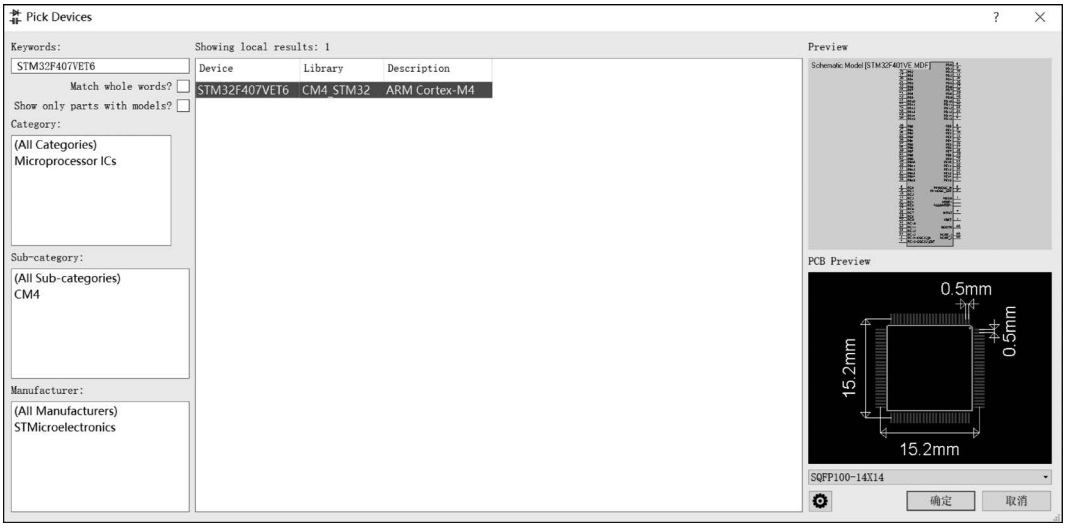

图 3-33　STM32F407VET6 仿真模型

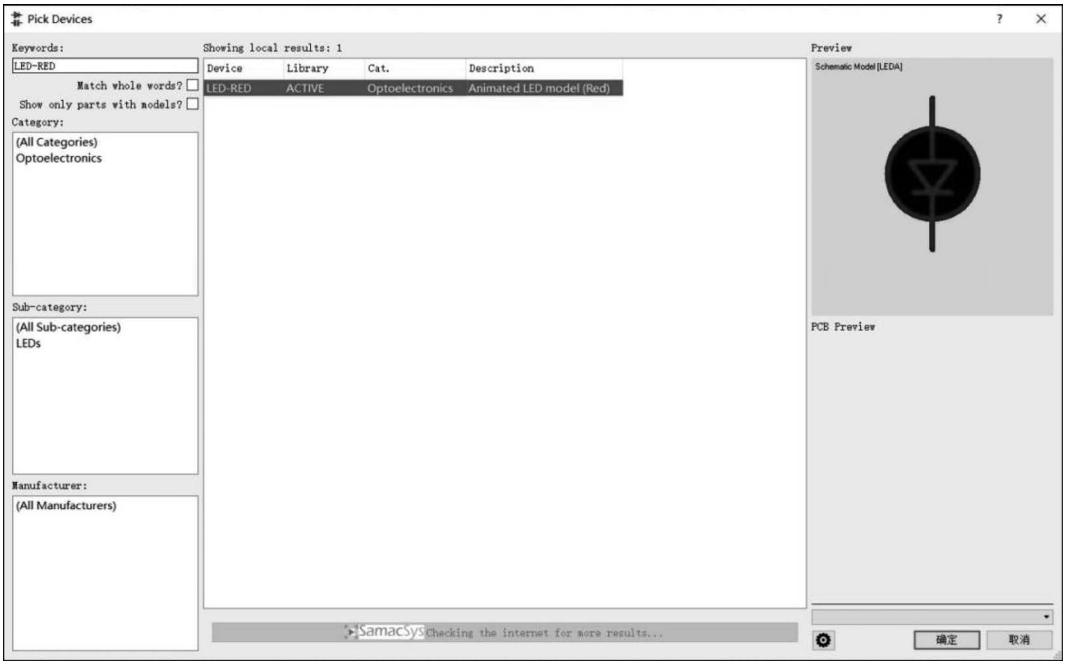

图 3-34　发光二极管 LED-RED 仿真模型

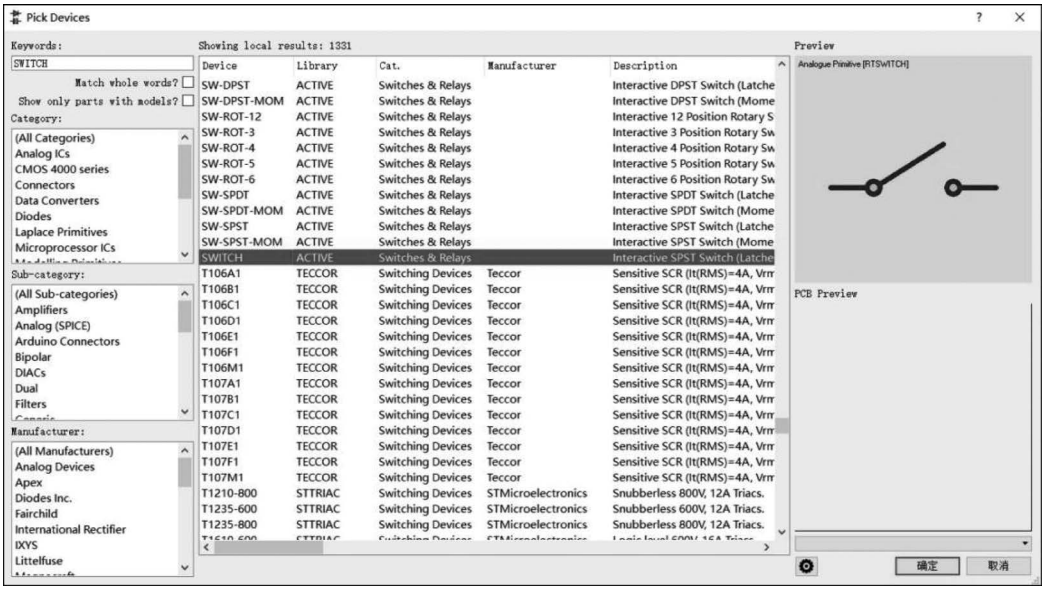

图 3-35　开关 SWITCH 仿真模型

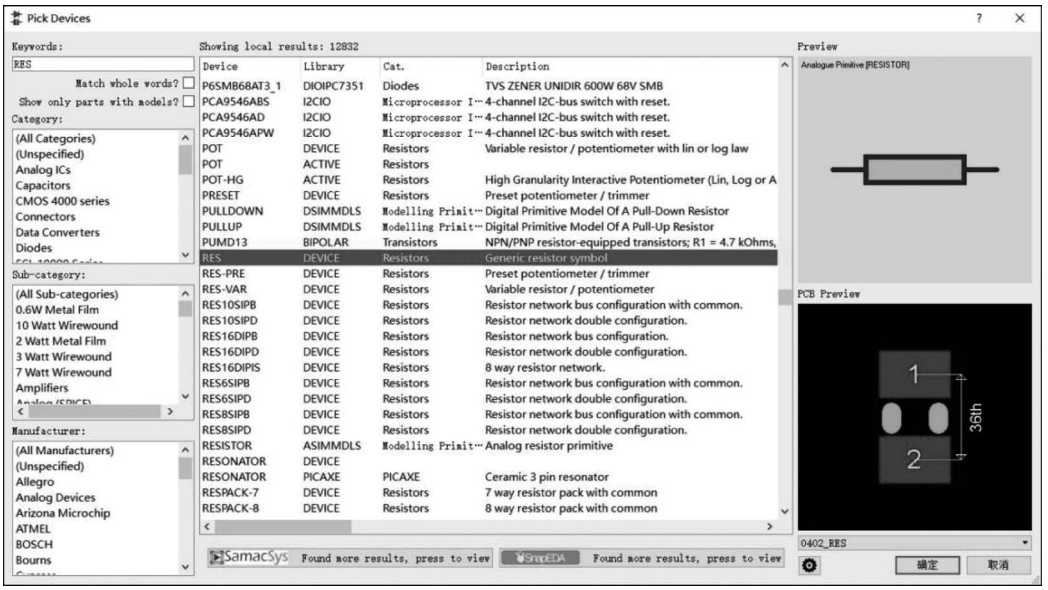

图 3-36　电阻 RES 仿真模型

在 Proteus 原理图界面中，单击左侧工具栏的 Terminals Mode，如图 3-37 所示，分别选择 POWER 和 GROUND 放入原理图中。

将原理图中的各类元器件连接，仿真电路图如图 3-38 所示。

双击 STM32F407VET6 元件，将 Keil MDK 软件编译建立的按键控制发光二极管闪烁程序的.hex 文件加载到元件中，如图 3-39 所示，单击 OK 按钮。

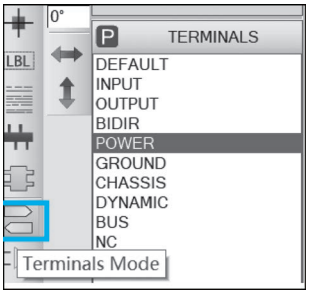

图 3-37　Terminals Mode

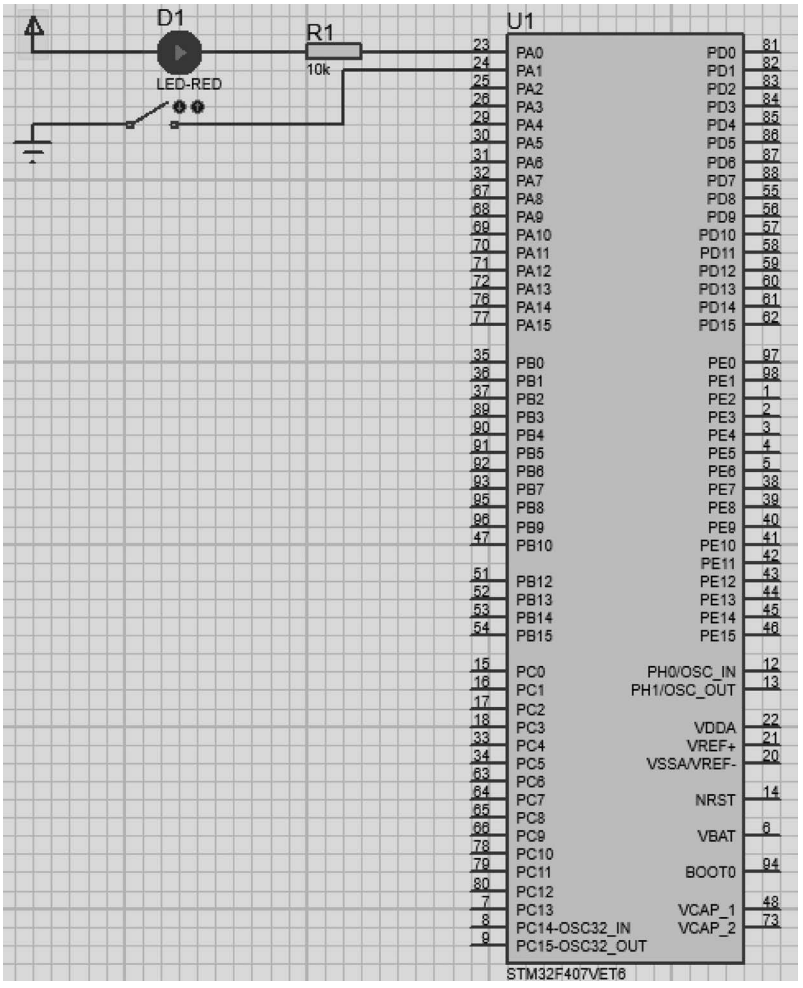

图 3-38　仿真电路图

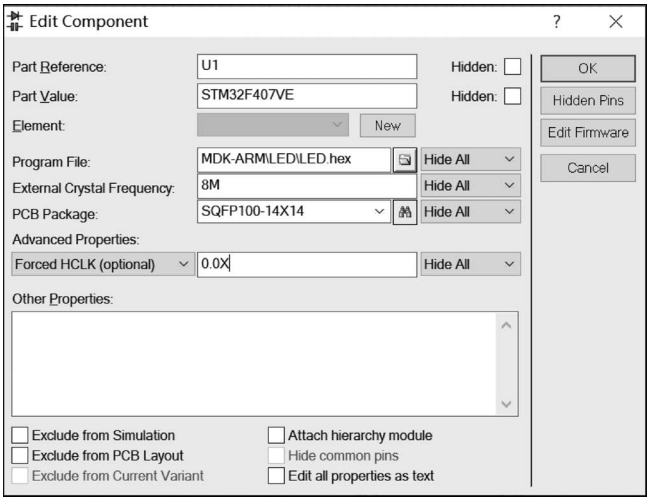

图 3-39　加载.hex 文件

3. 运行仿真

单击菜单栏 Debug 中的 Run Simulation，开始仿真选项运行仿真。仿真运行开始后观察引脚 PA0 和发光二极管 LED-RED，由于开关打开，此时 LED-RED 两端都是高电平，LED-RED 不亮，开关打开时的仿真电路如图 3-40 所示。

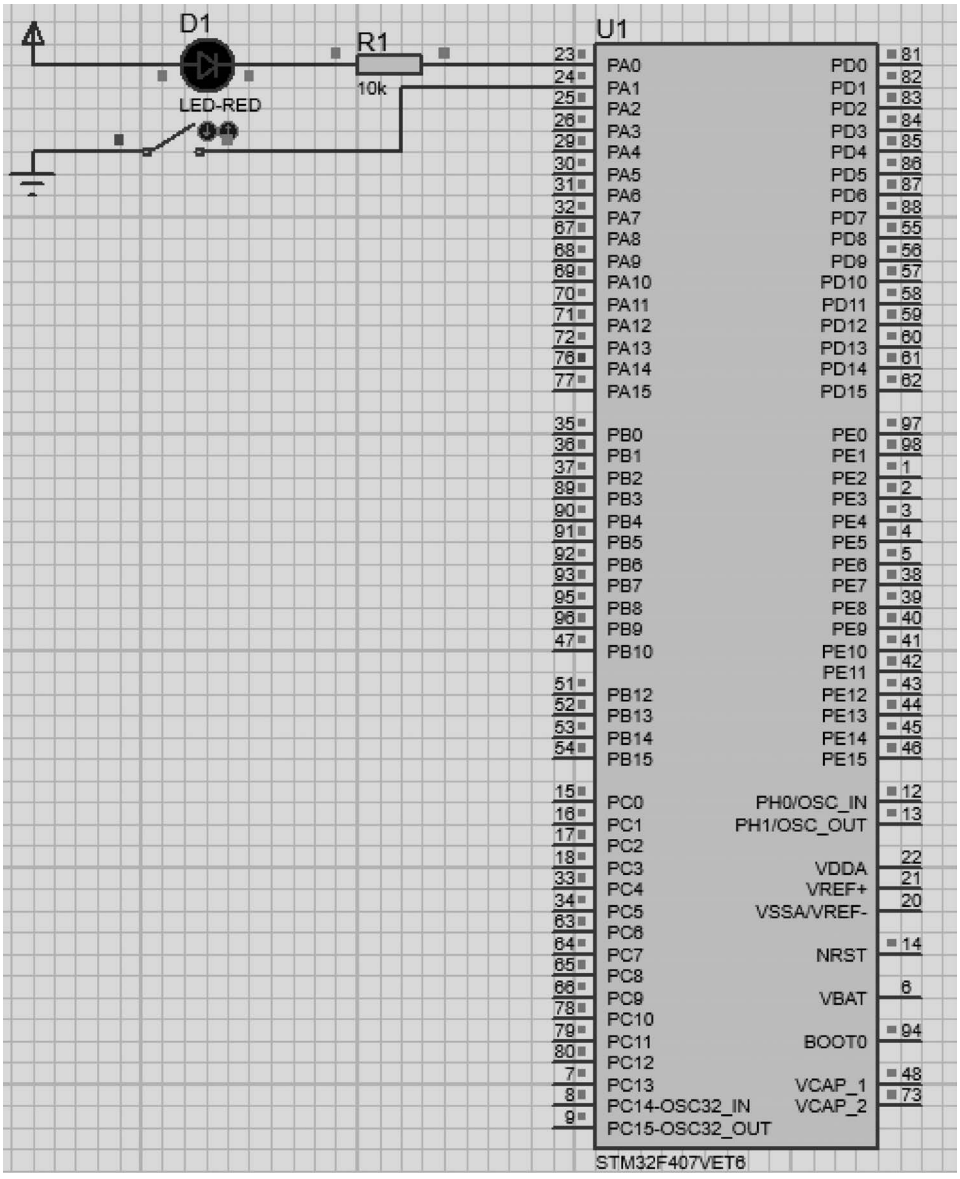

图 3-40　开关打开时的仿真电路

开关关闭，每隔 300ms，PA0 引脚电平翻转一次，发光二极管 LED-RED 状态变化一次。当 PA0 引脚输出低电平时，发光二极管 LED-RED 发光；当 PA0 引脚输出高电平时，发光二极管 LED-RED 不发光。开关关闭时发光二极管 LED-RED 发光仿真图如图 3-41 所示；开关关闭时发光二极管 LED-RED 不发光仿真图如图 3-42 所示。

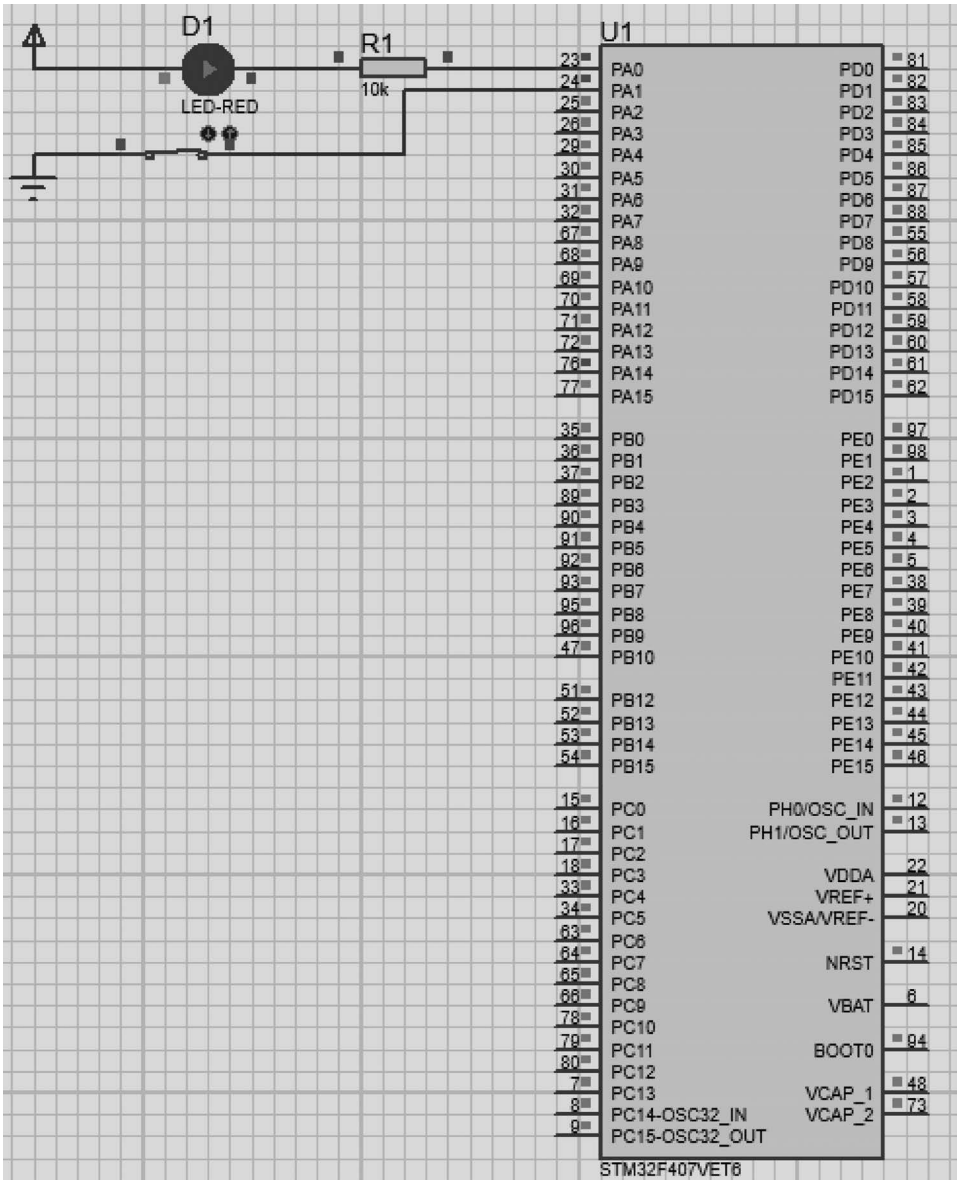

图 3-41　开关关闭时发光二极管 LED-RED 发光仿真图

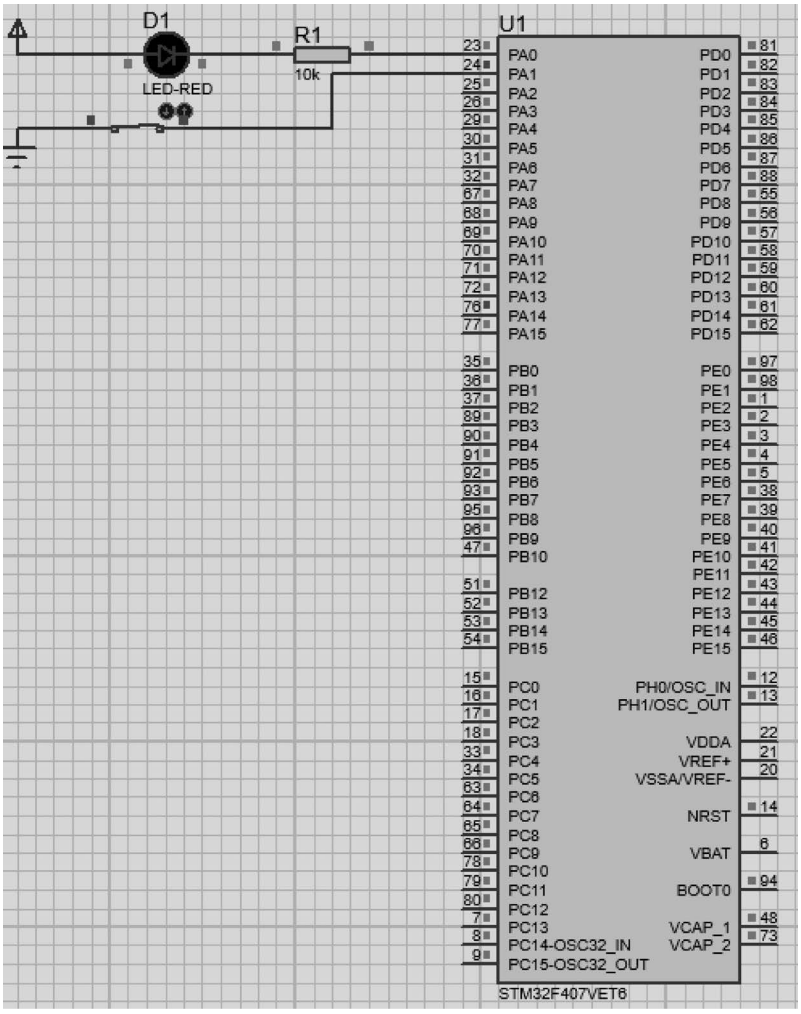

图 3-42　开关关闭时发光二极管 LED-RED 不发光仿真图

【本章小结】

本章深入探讨了 STM32 微控制器的 GPIO 功能。首先介绍了输入/输出的基本概念，为理解 GPIO 的工作原理奠定了基础。随后，详细阐述了 STM32 的 GPIO 特性，深入剖析了 STM32F407 GPIO 寄存器和 GPIO 引脚模式，有助于更深入地理解如何编程控制 GPIO 引脚。最后，通过一个具体的仿真实例——按键控制发光二极管闪烁，说明了如何综合应用 STM32CubeMX、Keil MDK、Proteus 实现 STM32 的 GPIO 控制功能，实例不仅加深了对 GPIO 操作的理解，还有助于提高学生在实际项目中对 GPIO 的应用能力。

【思政元素融入】

在通过 GPIO 引脚输出高电平和低电平来控制 LED 灯亮灭的过程中，将技术实践与思政教育相结合，培养学生的综合素质。鼓励学生亲自动手，通过搭建电路、编写代码来控制 LED 灯的亮灭，培养学生的实践能力和创新精神。这一过程不仅加深了学生对技术原理的理解，还激发了他们探索未知、解决问题的兴趣。