嵌入式系统原理及实验

顾　震

信息科学与工程学院自动化系

华东理工大学

Email: guzhen@ecust.edu.cn

课程大纲

1. 嵌入式系统导论
2. Cortex-M3微处理器
3. STM32最小系统及开发环境
4. 嵌入式C语言
5. 通用输入输出GPIO模块
6. 中断
7. 定时器原理与应用
8. USART通信原理及实现
9. DMA控制器
10. SPI与I2C通信原理及实现
11. 模数转换原理及实现
12. 人工智能辅助的嵌入式项目开发
13. 嵌入式应用前沿

5. 通用输入输出GPIO模块

本章知识与能力要求

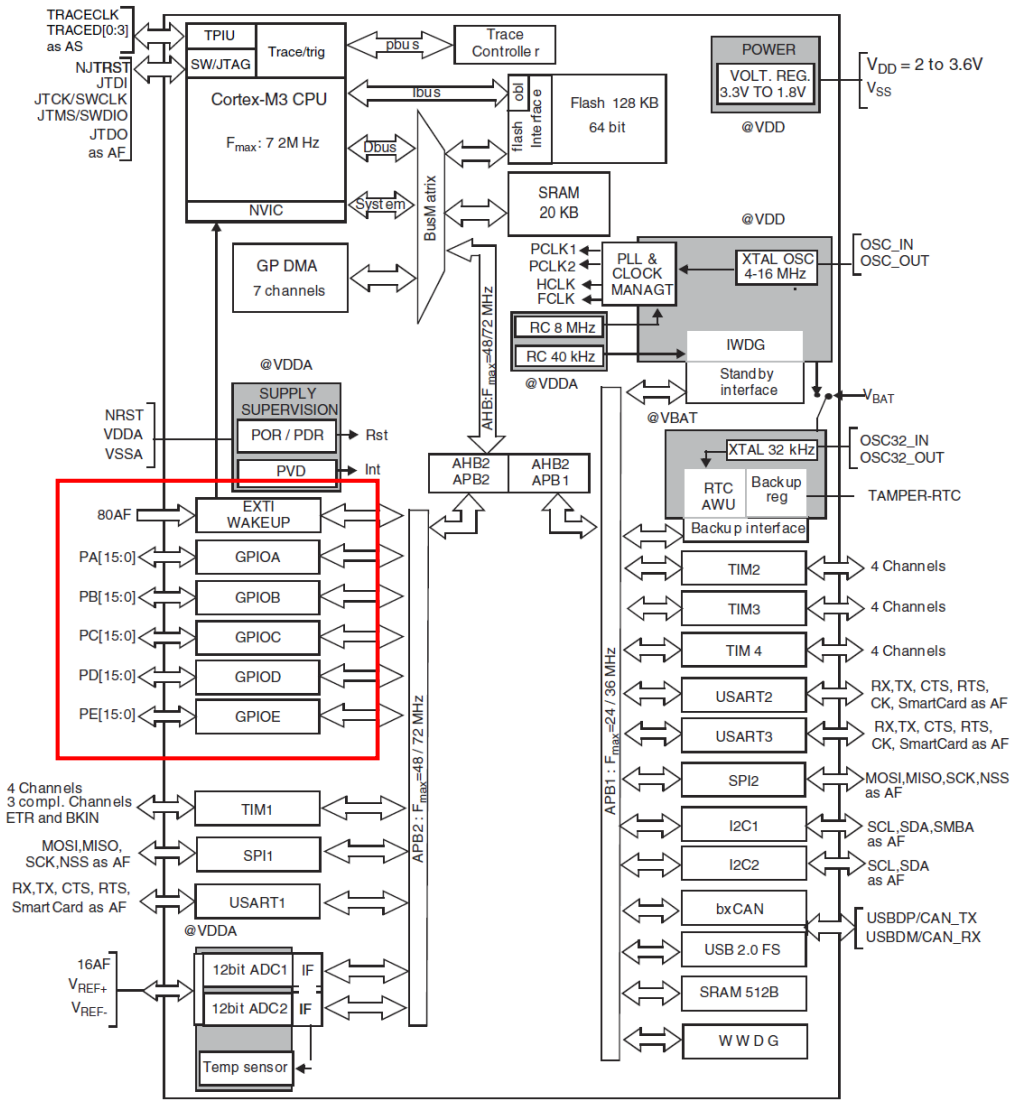
- 了解GPIO的基本概念；
- 理解STM32F103微控制器GPIO的内部结构、工作模式和使用特性；
- 理解GPIO的输入输出模式；
- 熟悉STM32F103微控制器GPIO相关的HAL库函数；
- 掌握基于HAL库实现LED灯闪烁的方法。

5.1 GPIO概述

GPIO

英文全称：General-Purpose Input/Output
中文全称：通用输入 / 输出

微控制器上的一种**通用引脚**，可通过**软件配置**为**输入或输出**模式，实现与外部设备的**数字信号交互**，是嵌入式系统中最基础、最常用的硬件接口之一。



5.2 STM32的GPIO工作原理

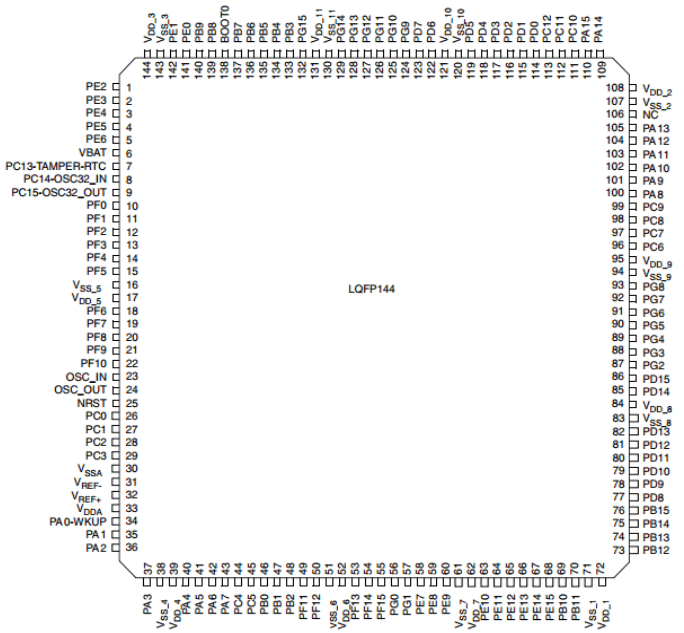
5.2.1 GPIO引脚

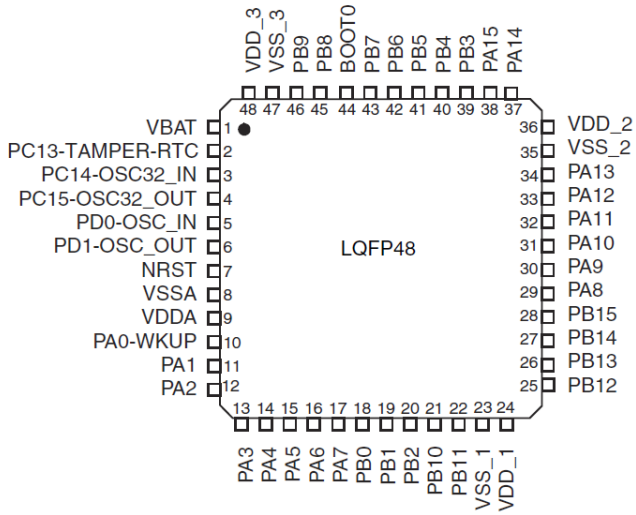
5.2.2 GPIO内部结构

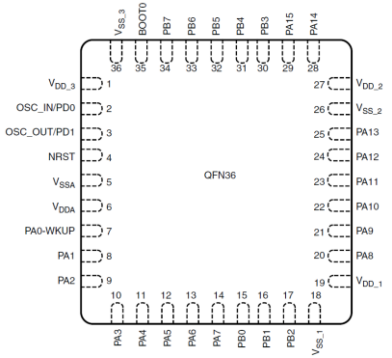
5.2.3 GPIO工作模式

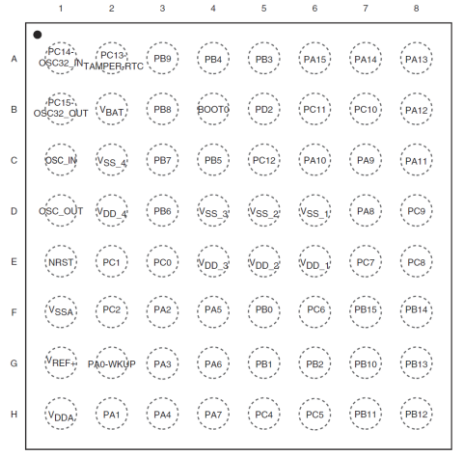
5.2.4 GPIO输出速度

5.2.1 STM32F103引脚









STM32F103共有7组GPIO，分别为：

PA组：PA0~PA15；

PB组：PB0~ PB15；

PC组：PC0~ PC15；

PD组：PD0~ PD15；

PE组：PE0~PE15；

PF组：PF0~PF15；

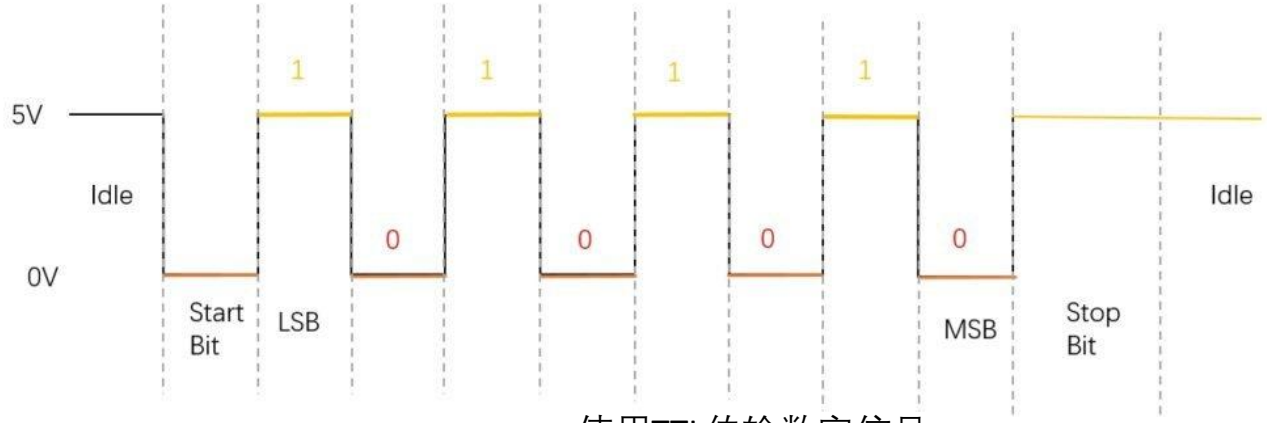
PG组：PG0~PG15；

不同封装型号可使用GPIO数量不同，但功能一致。

5.2.2 GPIO内部结构

TTL

- TTL是晶体管-晶体管逻辑（Transistor-Transistor Logic）的缩写，是一种数字逻辑电平标准，也称为**双极晶体管逻辑**或**三极管逻辑**，常用于数字电路和通信系统中。
- TTL信号是一种**二进制数字信号**，用于表示**逻辑0**和**逻辑1**
- 逻辑0通常被定义为**低电平**，而逻辑1则被定义为**高电平**



使用TTL传输数字信号

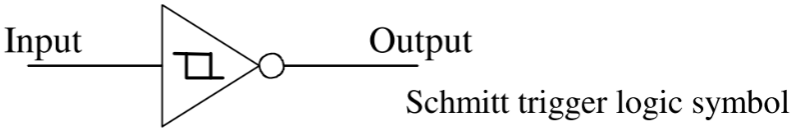
主要优点：噪声抑制能力强、功耗低、响应速度快

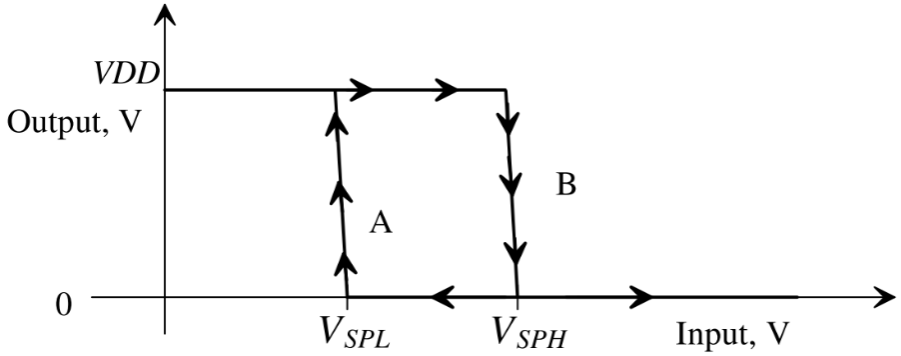
5.2.2 GPIO内部结构

施密特触发器

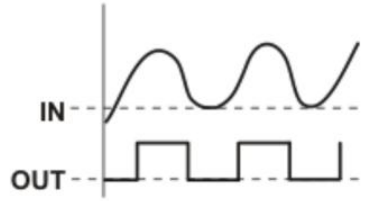
- 一种具有**滞回特性**的比较器电路，能够抗干扰并将模拟信号波形整形为方波。
- **滞回特性: 在输入信号的变化过程中，输出信号的变化具有一定的延迟和滞后现象**

常用符号

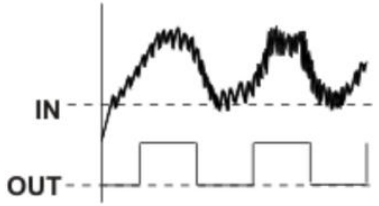




V_{SPL}开关下限电压
V_{SPH}开关上限电压



正弦波到方波



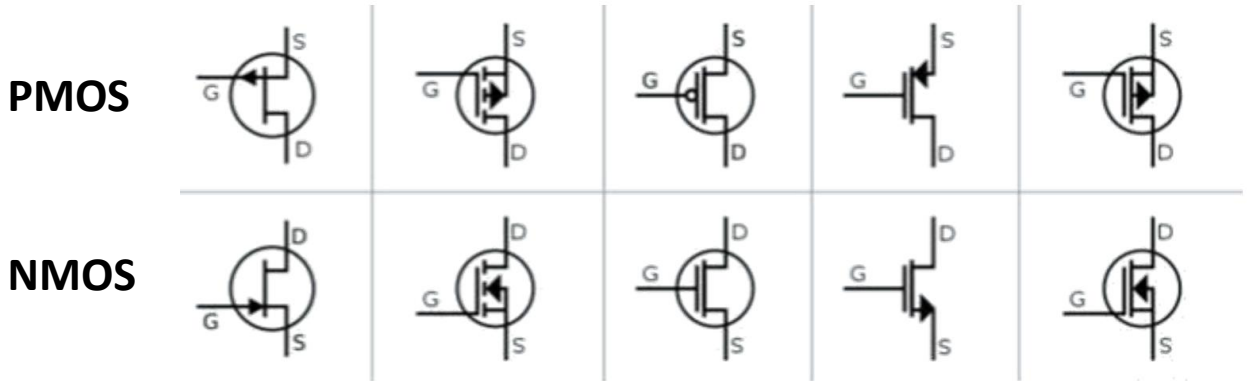
清理嘈杂信号

转换慢速边沿

5.2.2 GPIO内部结构

MOS管

MOS管是**金属（metal）-氧化物（oxide）-半导体（semiconductor）**场效应晶体管的缩写，是一种通过电场效应控制电流的半导体器件。



N 沟道 MOS 管（NMOS）

导通条件：栅极电压 > 源极电压（需超过阈值电压 V_{th}）

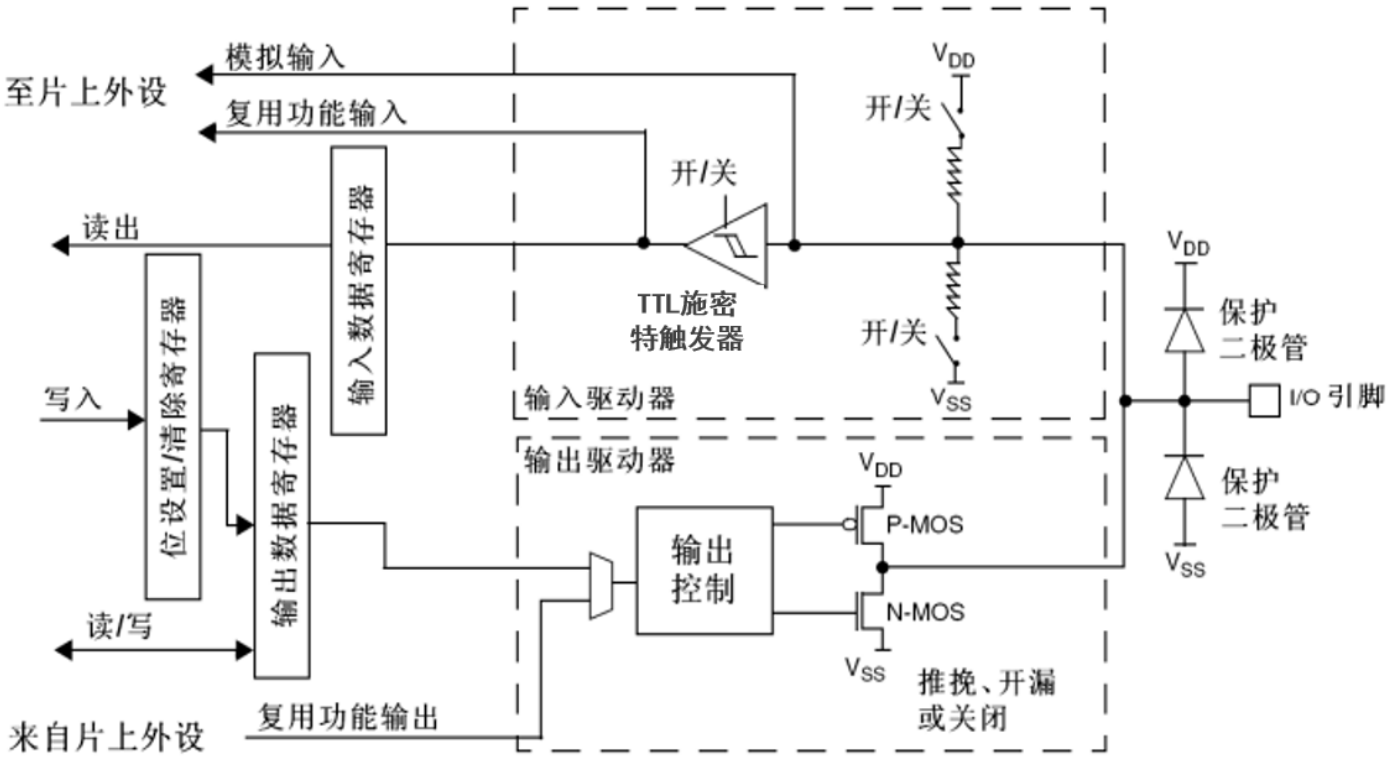
P 沟道 MOS 管（PMOS）

导通条件：栅极电压 < 源极电压（需低于阈值电压 V_{th}）

5.2.2 GPIO内部结构

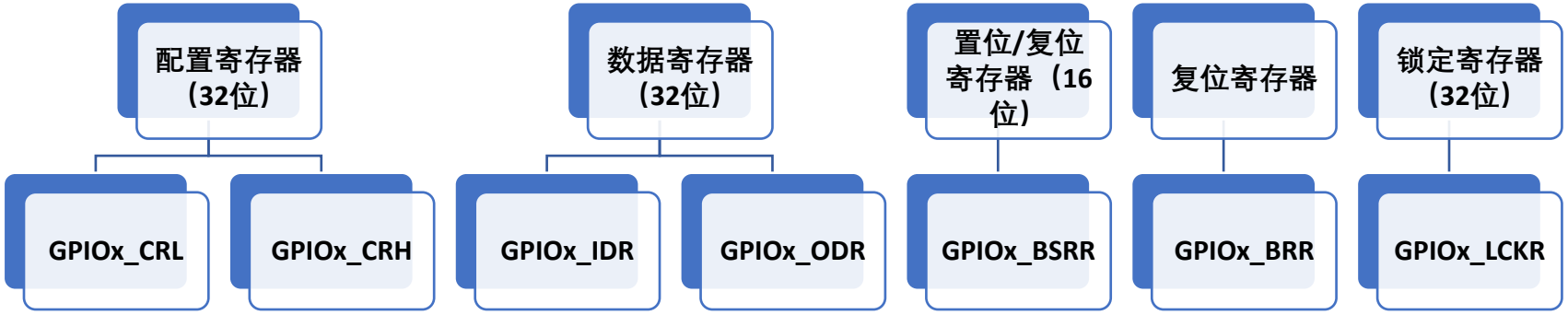
- GPIO 端口有5个主要部分构成，通过协同工作实现信号的输入 / 输出控制

引脚缓冲电路、方向控制、输出驱动、输入检测、复用功能切换

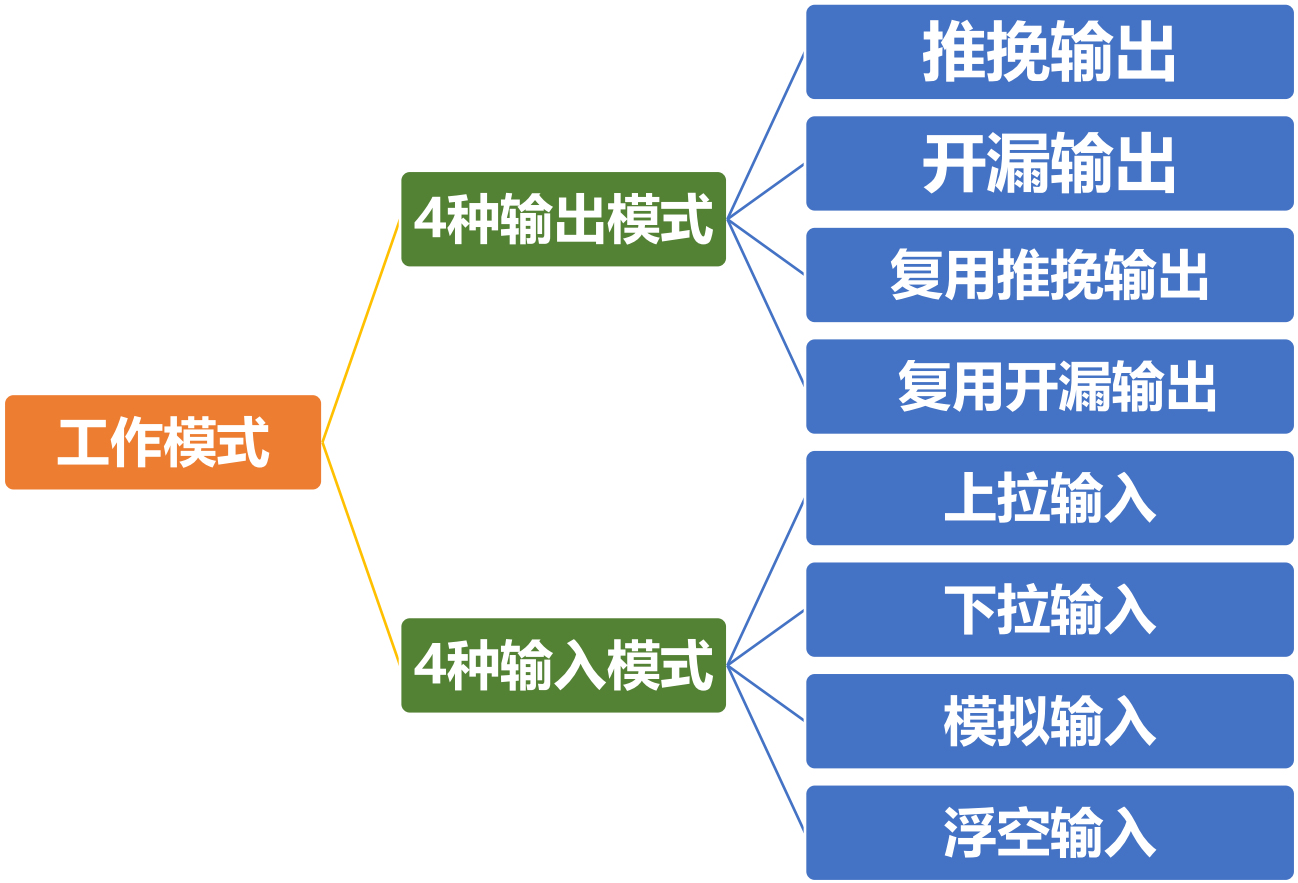


5.2.2 GPIO内部结构

- 每组GPIO端口(Px)都由7个寄存器组成，负责控制该端口的16个引脚Px0 ~ Px15。

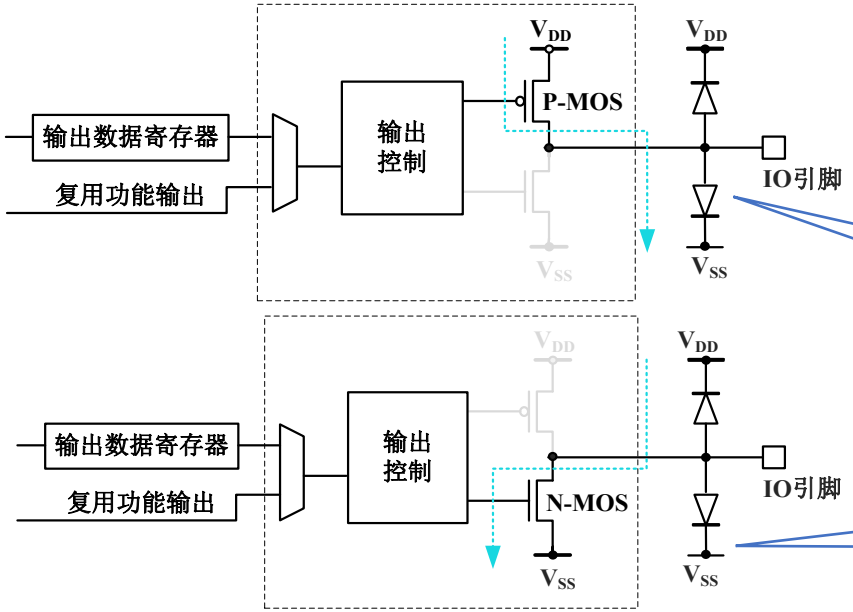


5.2.3 GPIO工作模式



5.2.3 GPIO工作模式

推挽输出模式（Push-Pull，PP）



推挽结构一般指两个MOS管受互补信号的控制，按互补对称的方式连接，任意时刻总是一个三极管导通，另一个截止。

推挽模式下，I/O引脚输出高电平时，P-MOS导通

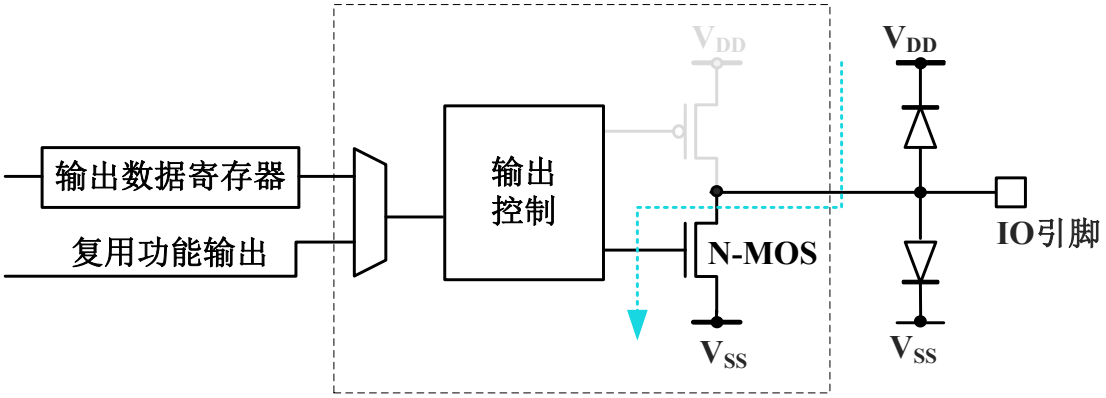
推挽模式下，I/O引脚输出低电平时，N-MOS导通

使用推挽输出模式的目的：增大输出电流，即增加输出引脚的驱动能力，提高电路的负载能力和提高开关的速度。

5.2.3 GPIO工作模式

开漏输出模式（Open-Drain，OD）

漏极开路（OD）输出**只有下拉MOS管**没有上拉MOS管，MOS管的漏极直接与I/O引脚相连，**不与电源连接，处于悬空状态**，称之为漏极开路。



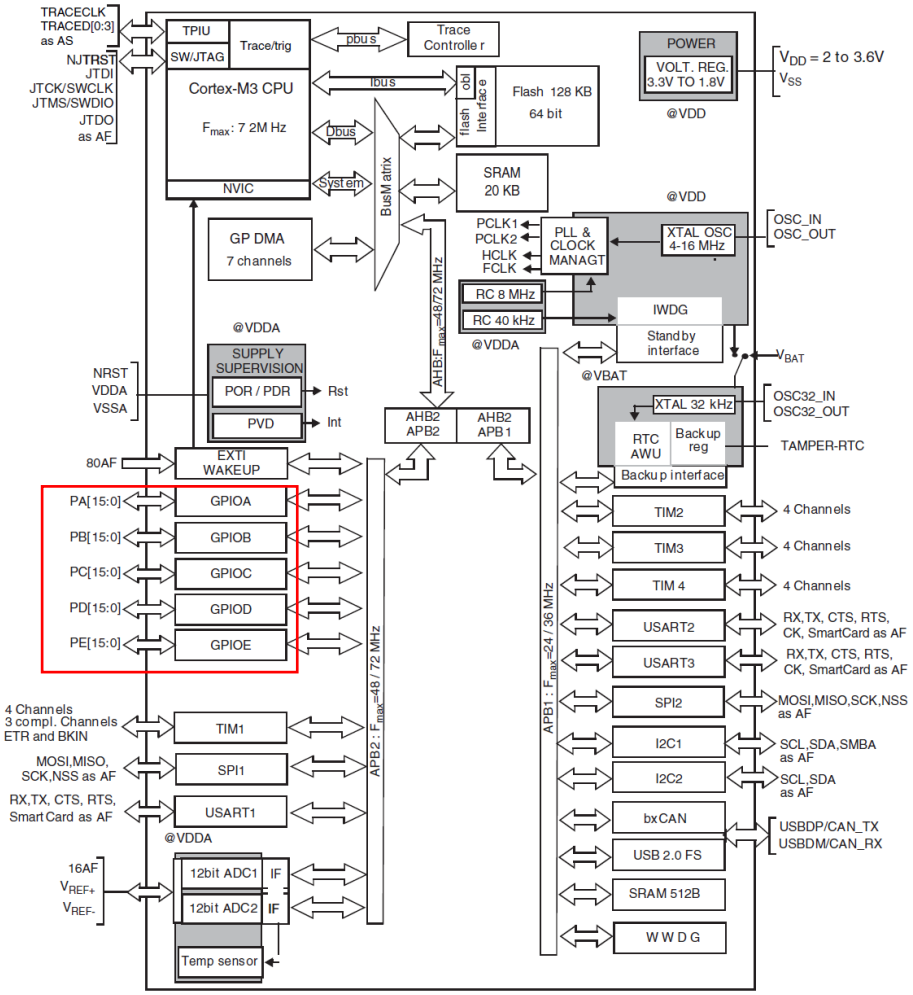
特点：

- 利用外部电路的驱动能力，**减少芯片内部的驱动**。
- 无上拉电阻时，只能输出低电平，要输出高电平，则必需外接上拉电阻。
- 开漏输出能够方便的实现“逻辑与”功能。

5.2.3 GPIO工作模式

复用功能输出模式（Alternate Function，AF）

GPIO引脚除了作为通用IO引脚外，还可作为片上外设的I/O引脚，即一个引脚可以作为多个外设引脚使用，称为复用I/O端口 AFIO （Alternate Function I/O），但一个引脚某一时刻只能使用复用功能中的一个。



5.2.3 GPIO工作模式

输入模式

上拉输入模式（Input Pull-up）

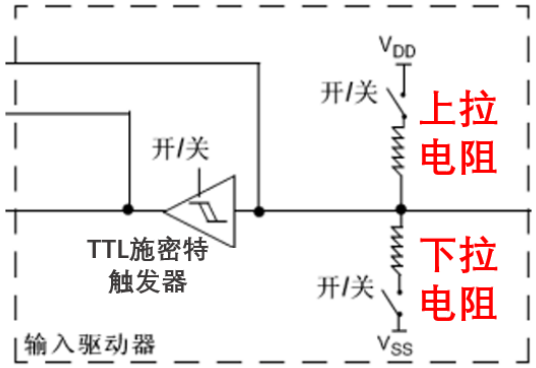
上拉输入模式，引脚内部有个上拉电阻，通过开关连接到电源VDD，当I/O引脚无输入信号时，默认输入高电平。

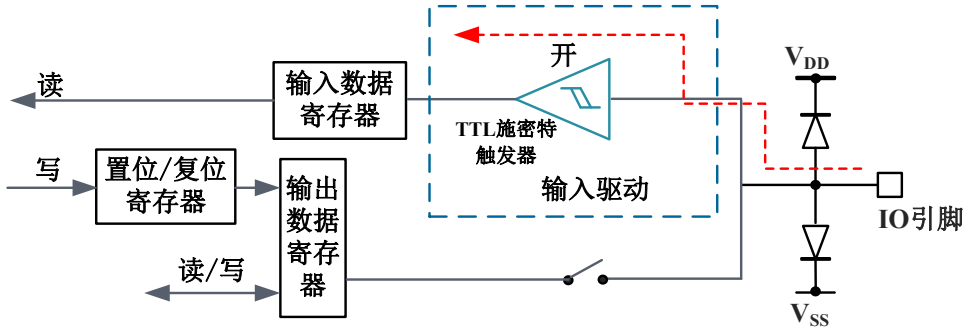
下拉输入模式（Input Pull-down）

下拉输入模式和上拉输入模式正好相反，当I/O引脚无输入信号时，默认输入低电平。

浮空输入模式

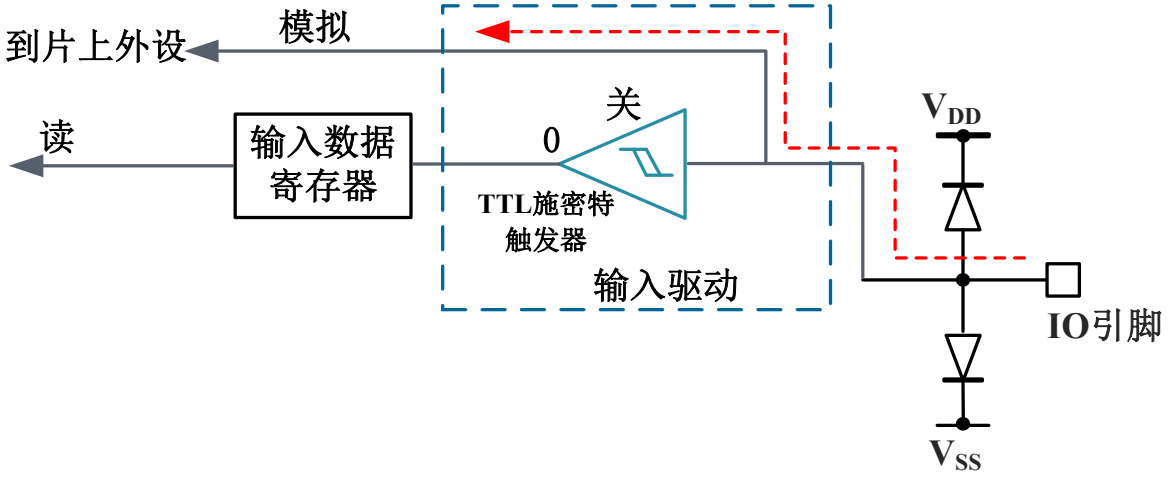
浮空输入模式下引脚内部既不接上拉电阻也不连接下拉电阻，直接经施密特触发器输入I/O引脚的信号。





5.2.3 GPIO工作模式

模拟输入模式

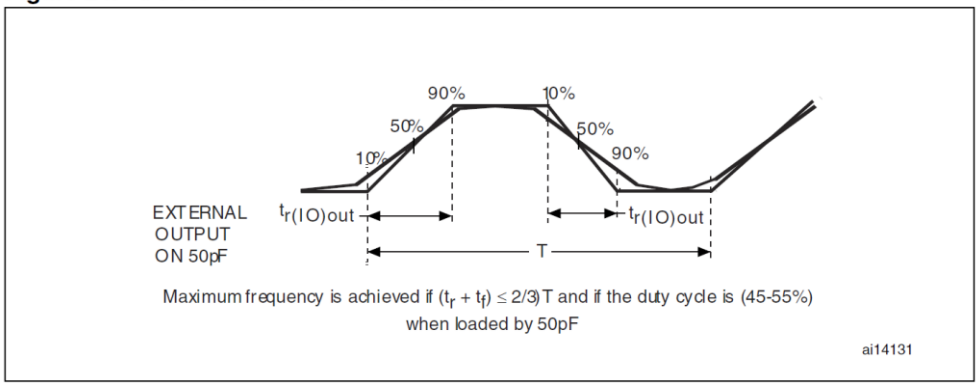


模拟输入模式下，施密特触发器关闭，既不接上拉电阻也不连接下拉电阻，引脚信号连接到芯片内部的片上外设，其典型应用是A/D模拟输入，对外部信号进行采集。

5.2.4 GPIO输出速度

- STM32F103系列微控制器的I/O引脚的输出速度有3种选择：2 MHz、10 MHz和50 MHz。
- 输出速度并不是输出信号的速度，而是指I/O口驱动电路的响应速度。
- 实际开发中，需要结合系统实际情况选择合适的响应速度，以确保信号的稳定性和降低功耗等。

Figure 24. I/O AC characteristics definition



上升沿 rising edge

数字电平从低电平变为高电平的边沿。

下降沿 falling edge

数字电平从高电平变为低电平的边沿。

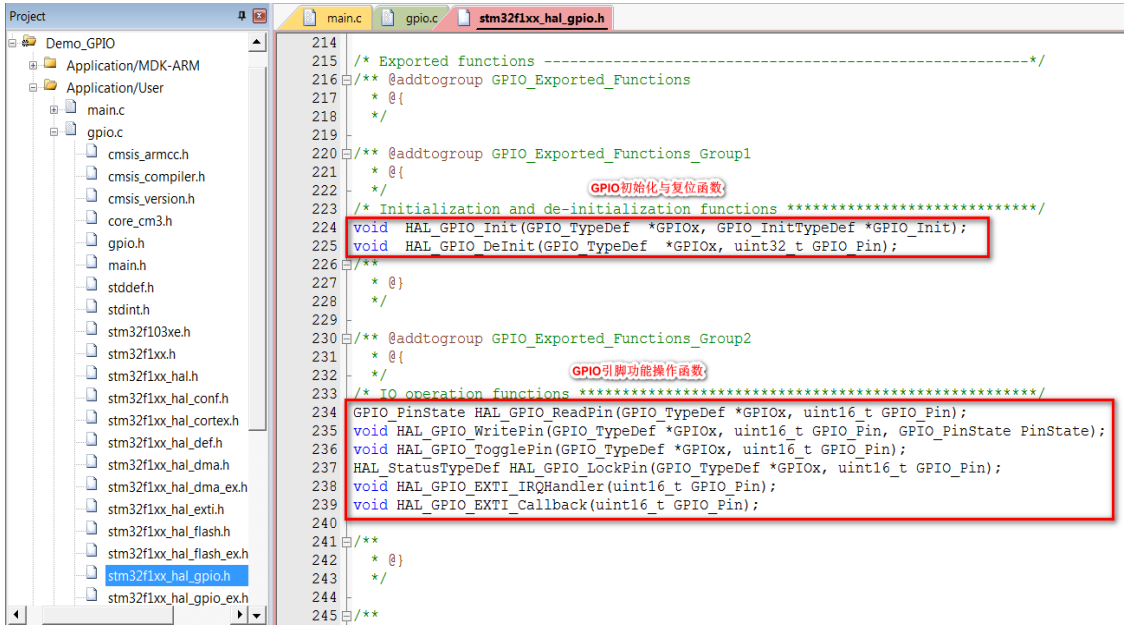
5.3 GPIO模块的HAL库接口函数及应用

5.3.1 GPIO的HAL库接口函数

5.3.2 GPIO的HAL库应用实例

5.3.3 基于HAL库开发的一般流程

5.3.1 GPIO的HAL库接口函数



GPIO的HAL库接口函数的源码在源文件stm32f1xx_hal_gpio.c中，其对应的头文件stm32f1xx_hal_gpio.h声明了GPIO所有的库函数，共8种。

5.3.1 GPIO的HAL库接口函数

类型	函数原型	功能描述
初始化及复位函数	HAL_GPIO_Init()	GPIO初始化函数
	HAL_GPIO_DeInit()	复位选定的端口引脚到初始状态
引脚功能操作函数	HAL_GPIO_ReadPin()	读取选定的端口引脚的电平状态
	HAL_GPIO_WritePin()	设置选定的端口引脚输出高电平或低电平
	HAL_GPIO_TogglePin()	设置选定端口引脚的电平状态翻转
	HAL_GPIO_LockPin()	当端口引脚电平状态改变时保持锁定时的值
	HAL_GPIO_EXTI_IRQHandler()	外部中断处理函数
	HAL_GPIO_EXTI_Callback()	中断回调函数

5.3.1 GPIO的HAL库接口函数

引脚功能操作函数

源码解析：

读取输入数据寄存器IDR的值与指定引脚进行按位与操作，结果若不为0，则返回高电平；结果若为0，则返回低电平。

GPIOx为引脚的端口号，取值为GPIOA~GPIOG，表示指向GPIO结构体的指针，用于访问端口引脚的寄存器。

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

  if ((GPIOx->IDR & GPIO_Pin) != (uint32_t)GPIO_PIN_RESET)
  {
      bitstatus = GPIO_PIN_SET;
  }
  else
  {
      bitstatus = GPIO_PIN_RESET;
  }
  return bitstatus;
}
```

5.3.1 GPIO的HAL库接口函数

GPIO_Pin为常量

GPIO_Pin_0表示0x0001；

GPIO_Pin_15表示0x8000。

GPIO_PinState表示引脚电平状态，为枚举变量，

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

取值范围为GPIO_PIN_SET =1或者GPIO_PIN_RESET = 0。

引脚号	引脚定义寄存器地址
GPIO_PIN_0	((uint16_t) 0x0001
GPIO_PIN_1	(uint16_t) 0x0002
GPIO_PIN_2	0x0004
GPIO_PIN_3	0x0008
GPIO_PIN_4	0x0010
GPIO_PIN_5	0x0020
GPIO_PIN_6	0x0040
GPIO_PIN_7	0x0080
GPIO_PIN_8	0x0100
GPIO_PIN_9	0x0200
GPIO_PIN_10	0x0400
GPIO_PIN_11	0x080
GPIO_PIN_12	0x1000
GPIO_PIN_13	0x2000
GPIO_PIN_14	0x4000
GPIO_PIN_15	0x8000
GPIO_PIN_All	0xFFFF

5.3.1 GPIO的HAL库接口函数

函数源码

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

源码解析

将指定引脚的电平状态写入置位/复位寄存器BSRR中，该寄存器为32位，其中高16位控制端口16个引脚（0~15）输出低电平，低16位控制端口16个引脚输出高电平。若设置的引脚电平状态不等于低电平，则将该引脚设置的状态位写入BSRR寄存器对应的低16位，否则，将该GPIO_Pin左移16位写入BSRR寄存器对应的高16位中。

5.3.1 GPIO的HAL库接口函数

函数源码

```
void HAL_GPIO_TogglePin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin)
{
    if ((GPIOx->ODR & GPIO_Pin) != 0x00u)
    {
        GPIOx->BRR = (uint32_t)GPIO_Pin;
    }
    else
    {
        GPIOx->BSRR = (uint32_t)GPIO_Pin;
    }
}
```

源码解析：

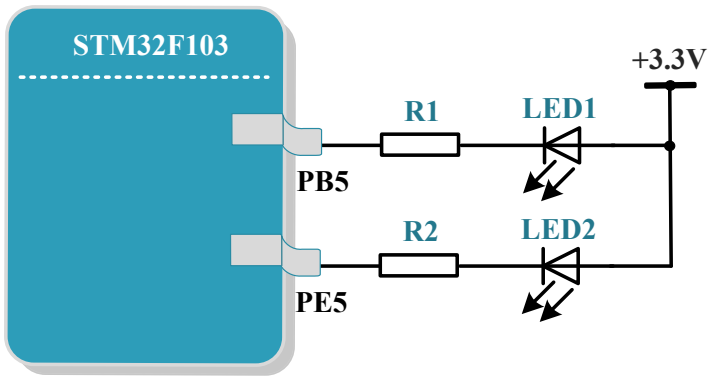
读出输出数据寄存器ODR的值与指定引脚进行按位与操作，结果若为真，表明原来为高电平，则写入BRR寄存器（BRR寄存器写入“1”有效，写入“0”不影响ODR的状态）；若结果为假，表明原来为低电平，则控制BSRR寄存器低16位部分做置位（置“1”）操作。

5.3.2 GPIO的HAL库应用实例

功能

采用基于HAL库设计方式，利用两个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IO口电平，实现灯光闪烁效果。

硬件设计

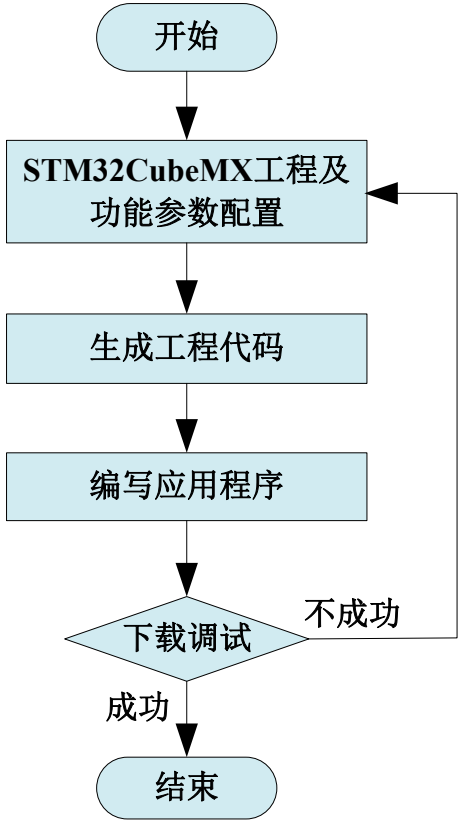


当引脚输出为低电平时，LED灯亮，当引脚输出为高电平时，LED灯灭。

5.3.2 GPIO的HAL库应用实例

基于HAL库的软件设计流程如图所示：

- 建立STM32CubeMX工程；
- 进行功能参数配置；
- 生成工程代码;
- 修改代码完成应用程序设计;
- 下载到开发板测试。



5.3.2 GPIO的HAL库应用实例

设计步骤——新建STM32CubeMX工程，选择设计采用的MCU

新建一个文件夹，用来存放后面建立的STM32CubeMX工程。

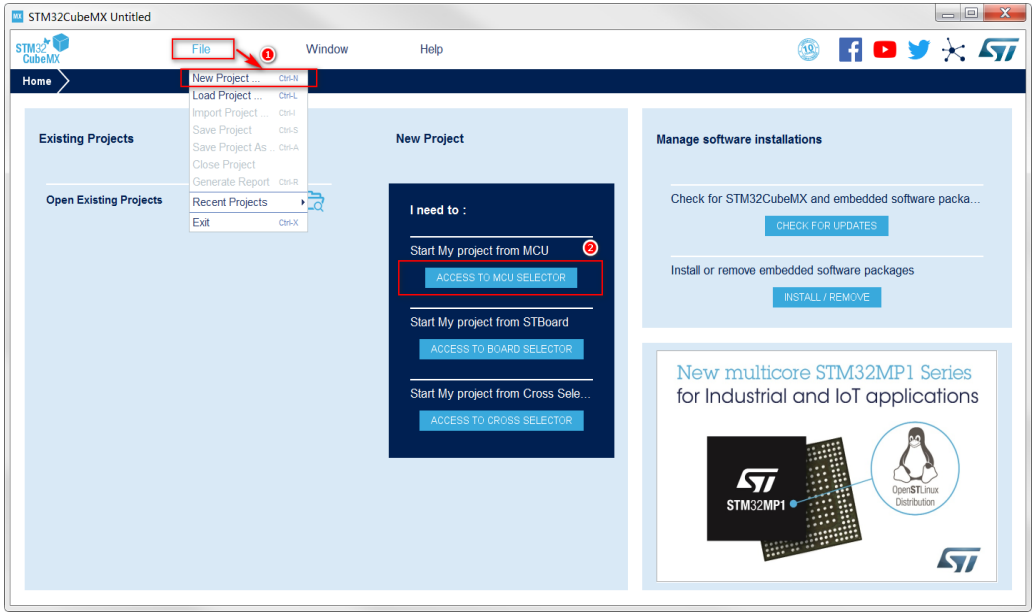
注意

所建工程文件名必须是英文名称，且必须是英文路径。

5.3.2 GPIO的HAL库应用实例

设计步骤——新建STM32CubeMX工程，选择设计采用的MCU

打开STM32CubeMX软件



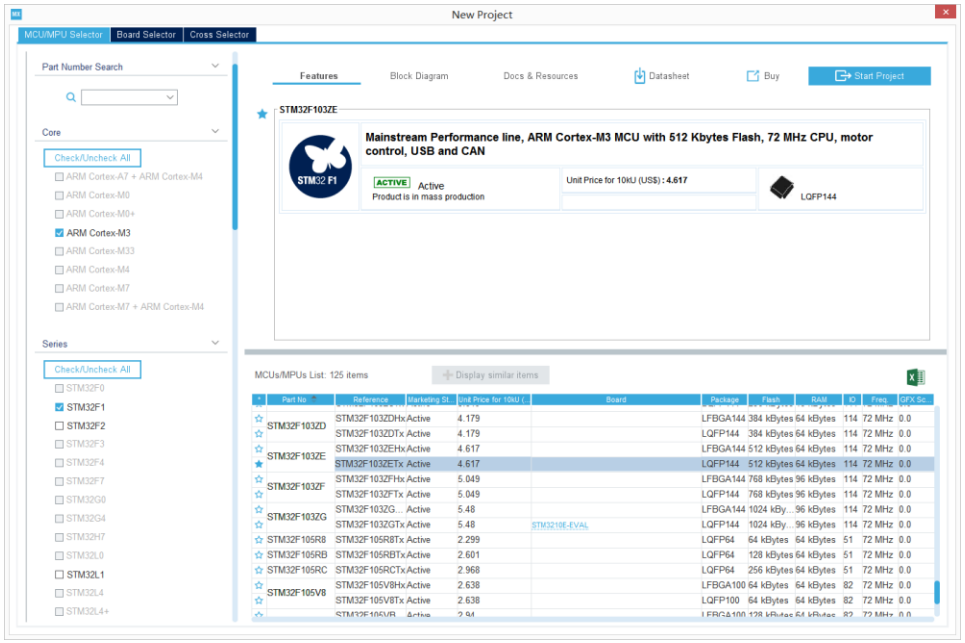
在主界面中通过单击“New Project”下“ACCESS TO MCU SELECTOR”按钮

或通过菜单栏中的“File”→“New Project”新建一个工程

5.3.2 GPIO的HAL库应用实例

设计步骤——新建STM32CubeMX工程，选择设计采用的MCU

- 选择所用MCU对应的内核，选择“ARM Cortex-M3”；
- 在“Series”选项中选择对应的系列，如“SMT32F1”；
- 在右侧“MCUs/MPUs List”中找到该系列下所使用的微控制器芯片型号
- 选中并单击对应芯片型号，在“MCUs/MPUs List”上方会显示该芯片对应的基本信息。

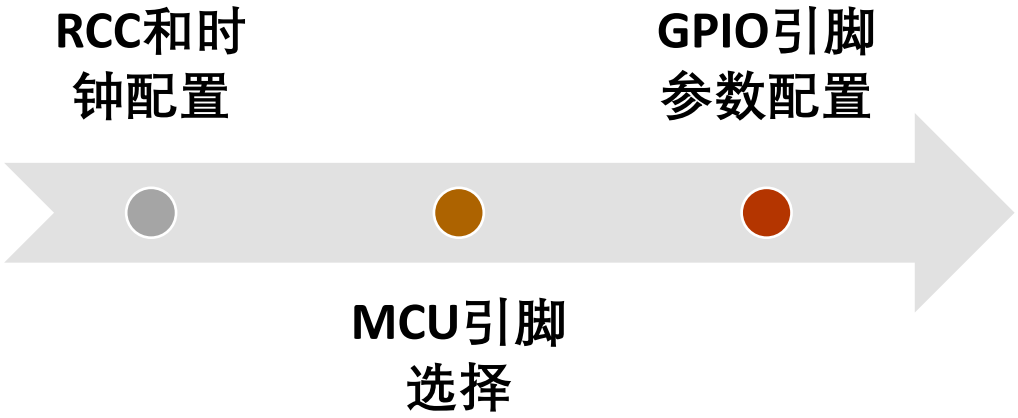


也可以通过在搜索栏内直接输入芯片型号采用搜索的方式来快速完成。

5.3.2 GPIO的HAL库应用实例

设计步骤——STM32CubeMX功能参数配置

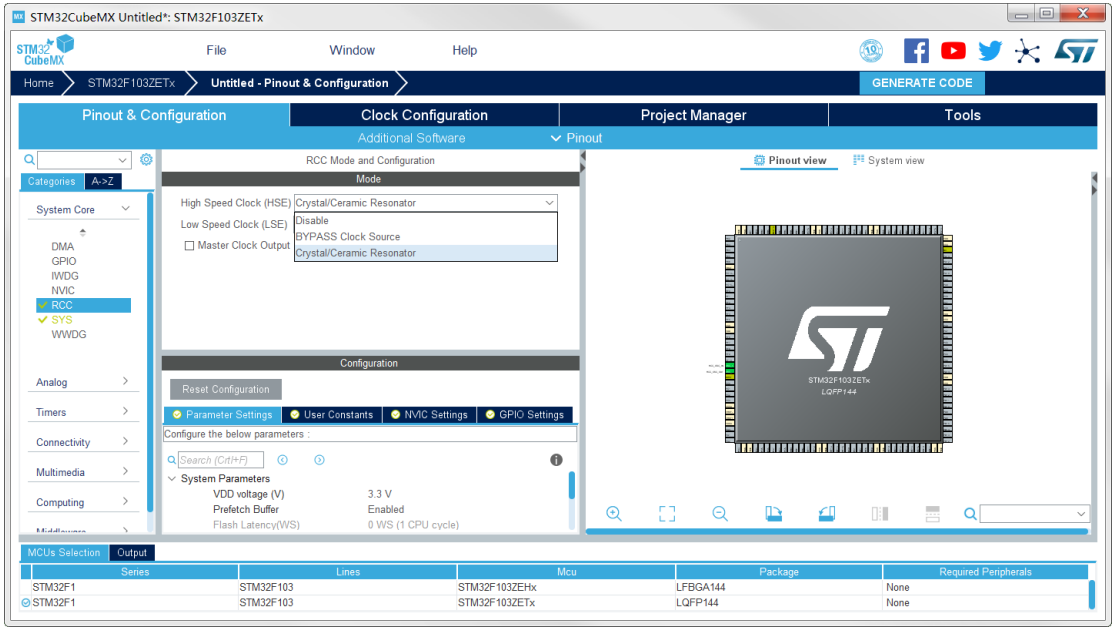
在New Project窗口完成MCU的相关设置后，双击所选择的具体芯片，进入STM32CubeMX的主界面，完成以下过程：



5.3.2 GPIO的HAL库应用实例

设计步骤——STM32CubeMX功能参数配置（RCC配置）

在“Categories”栏目中的“System Core”中，找到“RCC”选项



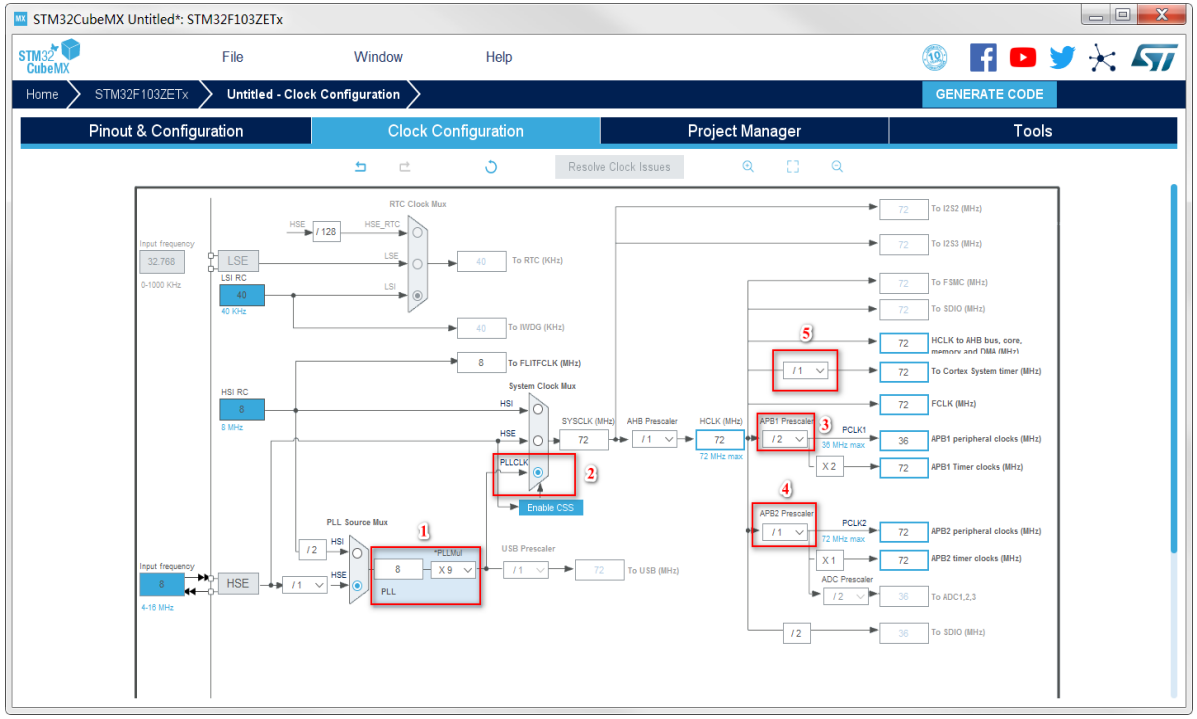
时钟信号选择HSE作为系统的外部时钟源，HSE选择“Crystal/Ceramic Resonator”（晶振/陶瓷谐振器）

LSE选择“Disable”

5.3.2 GPIO的HAL库应用实例

设计步骤——STM32CubeMX功能参数配置（时钟配置）

点开“Clock Configuration”选项栏，进行系统时钟配置

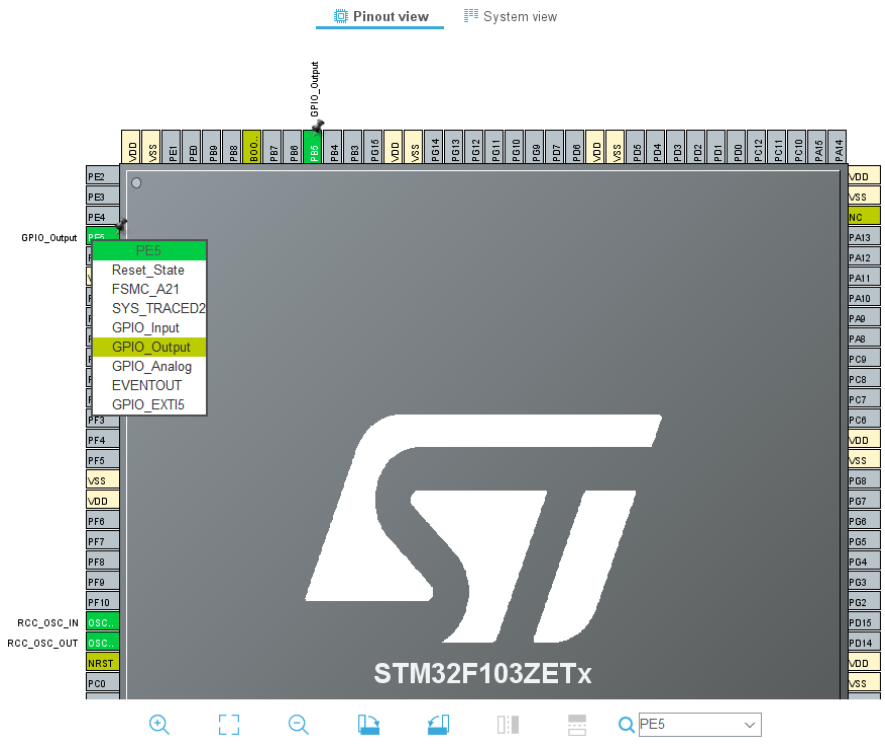


- 采用HSE外部晶振，频率为8MHz，通过PLL的9倍频，使得系统时钟SYSCLK为72MHz；
- APB2时钟与HCLK相同，所以不需要分频；
- HCLK经2分频得到APB1的时钟频率为PCLK1=36MHz，

5.3.2 GPIO的HAL库应用实例

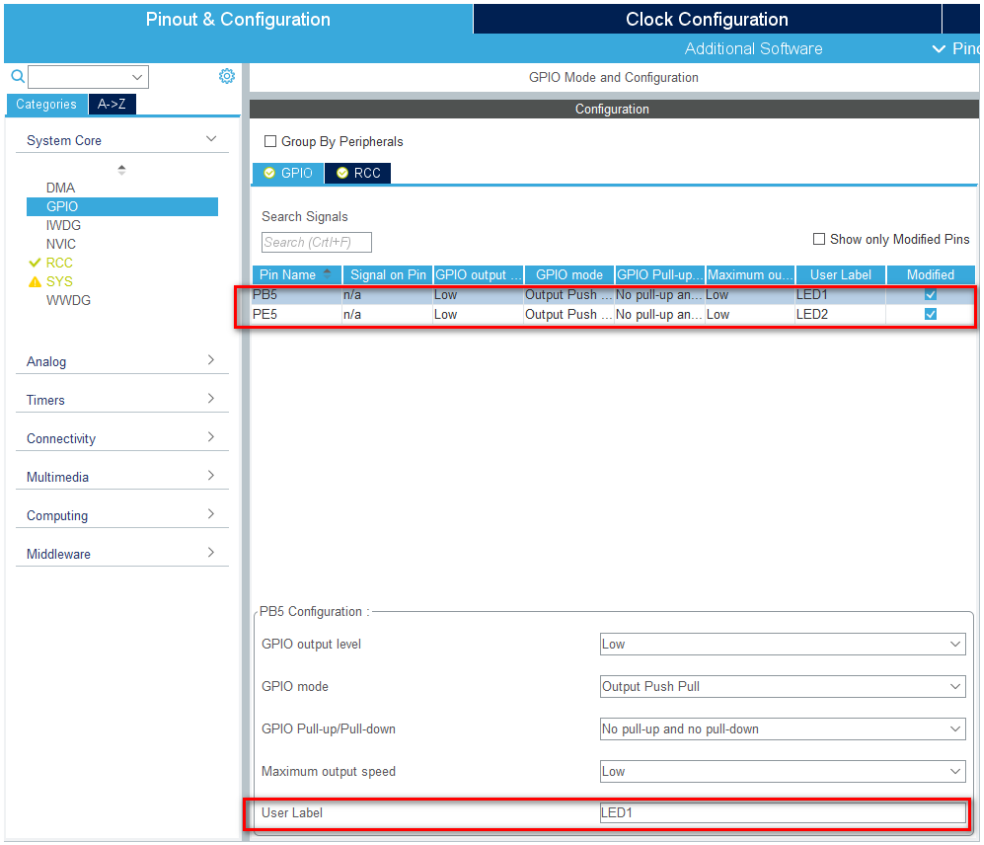
设计步骤——STM32CubeMX功能参数配置（MCU引脚选择）

在STM32CubeMX的主界面的左侧“Categories”栏目中的“System Core”中，找到“GPIO”选项，在STM32CubeMX的主界面的右侧“Pinout view”芯片引脚图中选中**PE5**、**PB5**引脚，设置为GPIO_Output选项。



5.3.2 GPIO的HAL库应用实例

设计步骤——STM32CubeMX功能参数配置（GPIO引脚参数配置）

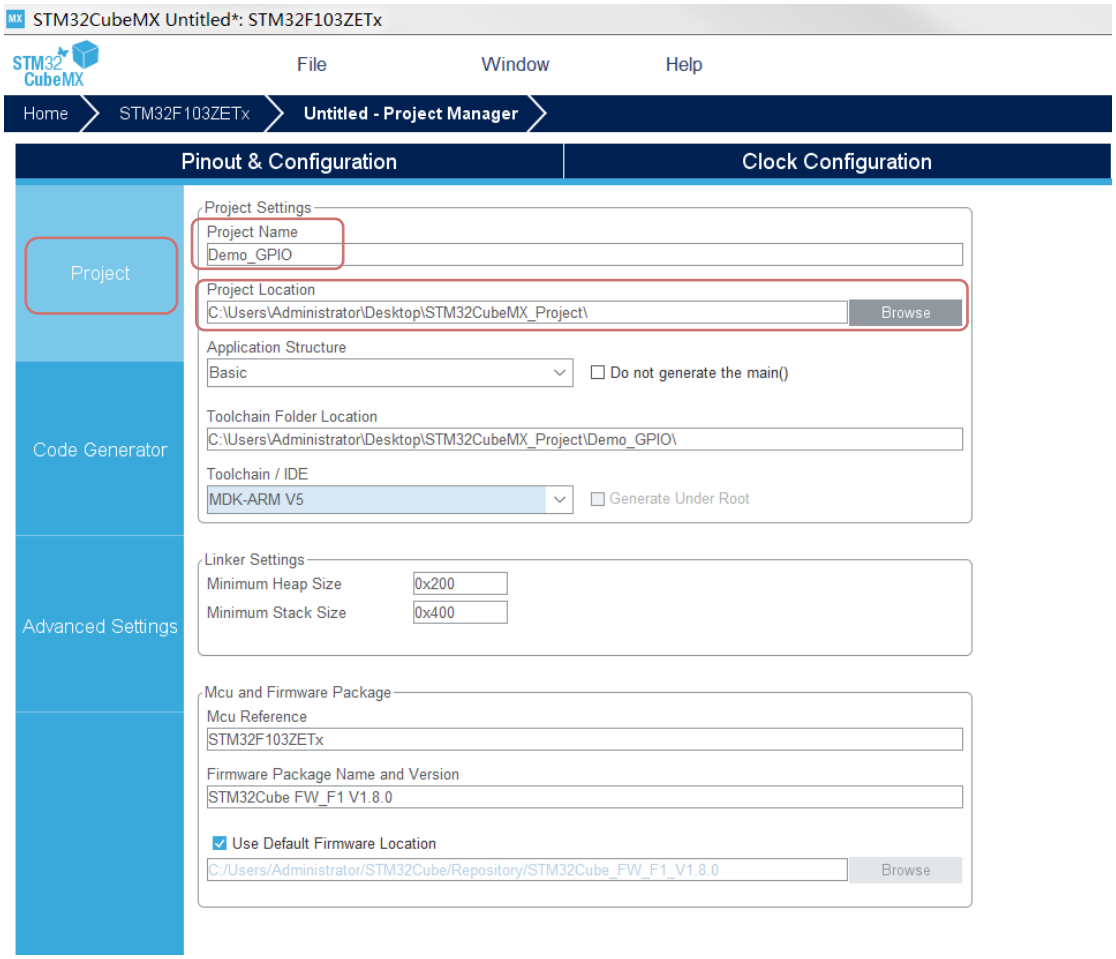


选择好引脚后，在左侧“Configuration”栏目下显示所选择的引脚，单击，对应显示该引脚的参数设置表，设置相应参数，如输出电平、模式、用户标签等。

5.3.2 GPIO的HAL库应用实例

设计步骤——生成工程代码

点击STM32CubeMX的主界面中的“Project Manager”菜单，在弹出的页面中单击“Project”项，输入项目名称“Demo_GPIO”，选择存放路径。



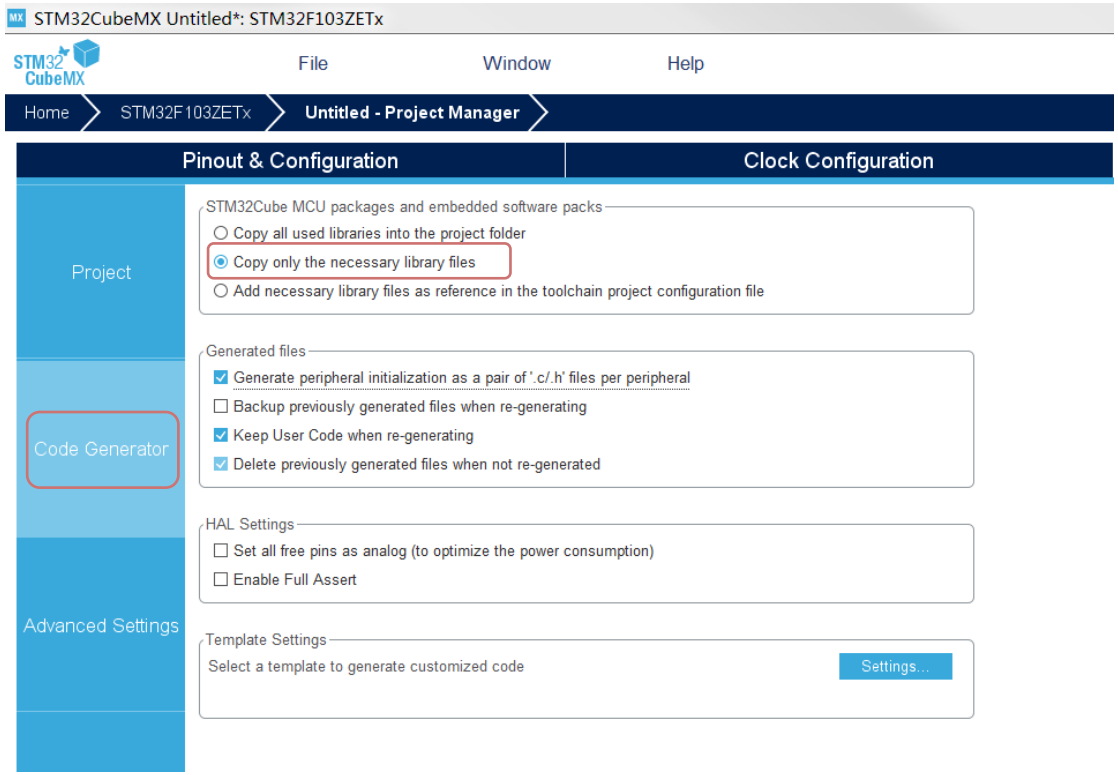
说明

用户可以自己定义，项目名称一般应反映项目内容，便于管理

5.3.2 GPIO的HAL库应用实例

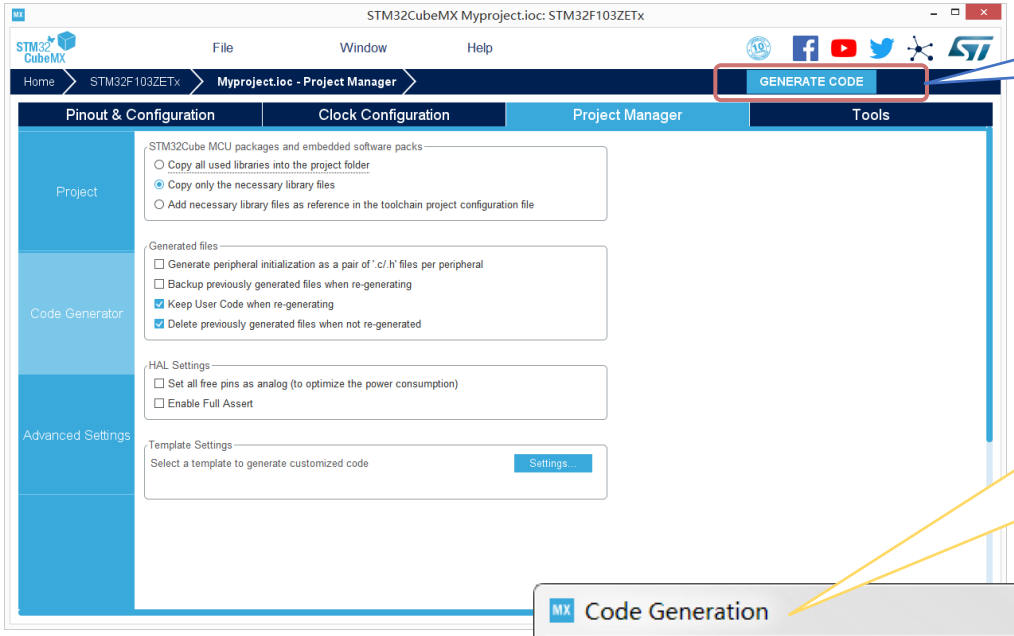
设计步骤——生成工程代码

- 点击“Code Generator”选项
- 在“STM32Cube MCU packages and embedded software packs”栏内选择第二个单选项“Copy only the necessary library files”，仅拷贝必须的库文件；
- 在“Generated files”栏内复选第1、3、4选项

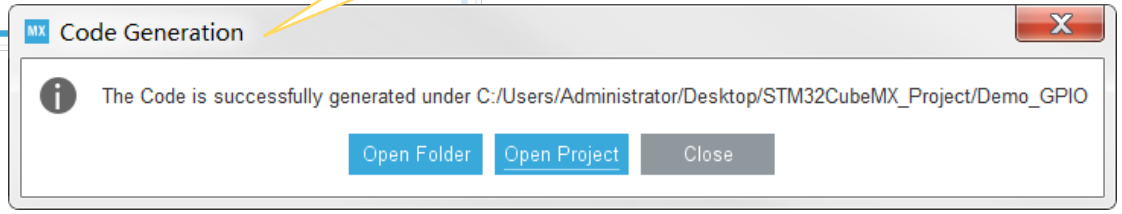


5.3.2 GPIO的HAL库应用实例

设计步骤——生成工程代码



单击右上角“GENERATE CODE”按钮，即可生成对应工程代码。



成功后弹出Code Generation窗口，提示代码生成成功，用户可以根据下一步需要选择打开文件夹、打开项目或关闭。

5.3.2 GPIO的HAL库应用实例

设计步骤——编写应用程序

本实例的用户应用程序代码写在/* USER CODE BEGIN 3 */ 和 /* USER CODE END 3 */ 之间：

```
while (1)
{
 /* USER CODE END WHILE */
 /* USER CODE BEGIN 3 */
   HAL_GPIO_TogglePin(LED1_GPIO_Port,LED1_Pin);
         //LED1---PB5状态翻转
   HAL_Delay(100); //延时100毫秒
   HAL_GPIO_TogglePin(LED2_GPIO_Port,LED2_Pin);
        //LED2---PE5状态翻转
   HAL_Delay(100); //延时100毫秒
 /* USER CODE END 3 */
}
```

使用STM32CubeMX生成的工程，其用户功能代码即应用程序代码的编写有**位置规范要求**。

说明

用户自己编写的应用程序需写在/* USER CODE BEGIN x */ 和 /* USER CODE END x */ 之间。

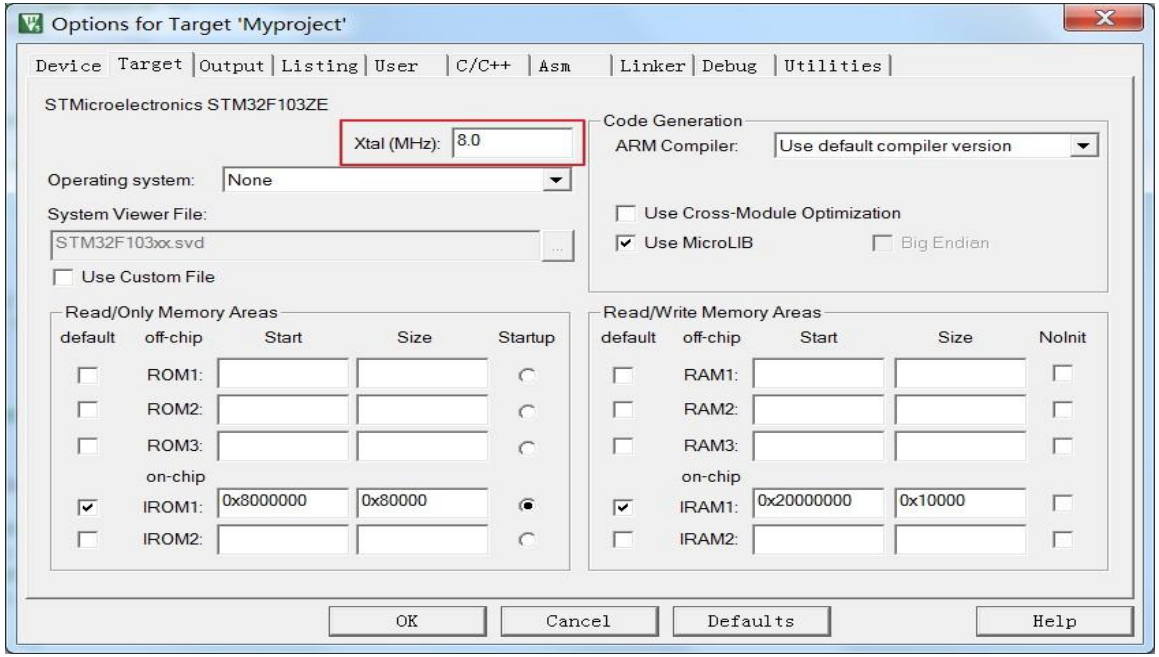
注意

写在其他地方，在使用STM32CubeMX重新配置和生成工程时，会删除该代码。

5.3.2 GPIO的HAL库应用实例

设计步骤——下载调试验证

配置Keil5相关工程，重新编译。

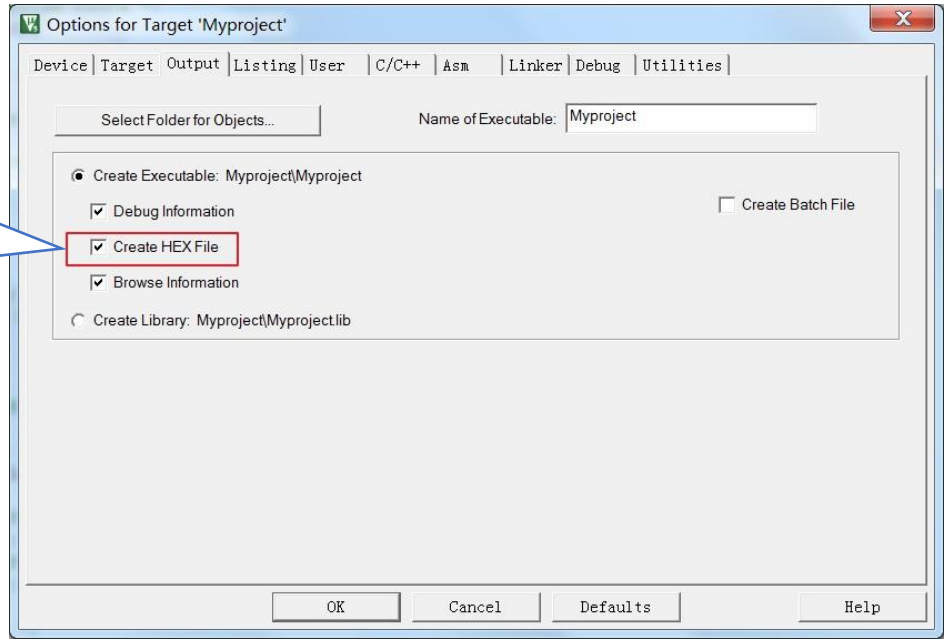


单击keil uVision5工具栏的魔法棒 “Options for Target”按钮，打开配置窗口页面，在“Target”选项卡中将“Xtal(MHz)”改为8.0，采用8MHz外部晶振

5.3.2 GPIO的HAL库应用实例

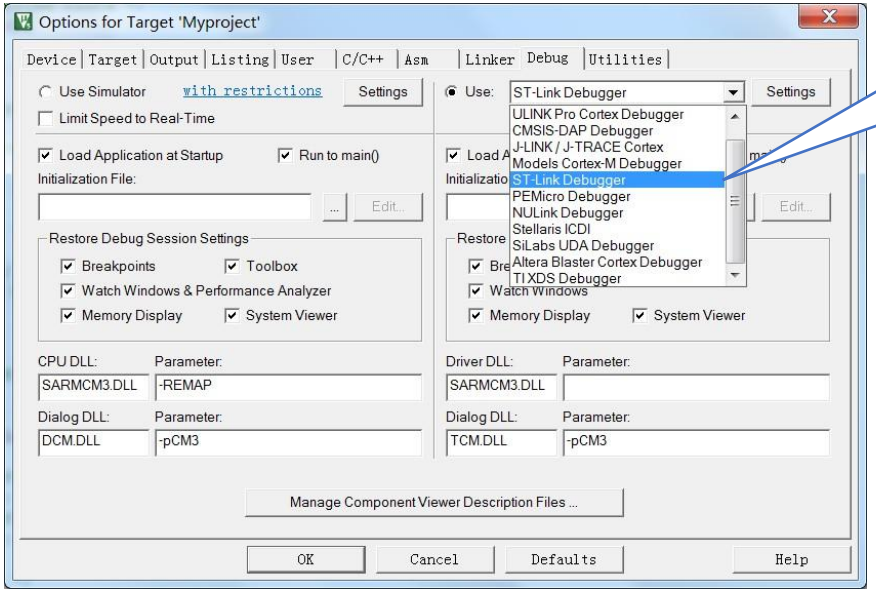
设计步骤——下载调试验证

在Output选项卡中，勾选Create HEX File选项，在工程重新编译后，会生成相应的HEX文件。



5.3.2 GPIO的HAL库应用实例

设计步骤——下载调试验证



在“Debug”选项卡中，根据所使用的开发板下载调试工具，选择相应的选项，这里选择ST-Link调试下载器

单击 “Build(F7)”编译按钮，进行重新编译。

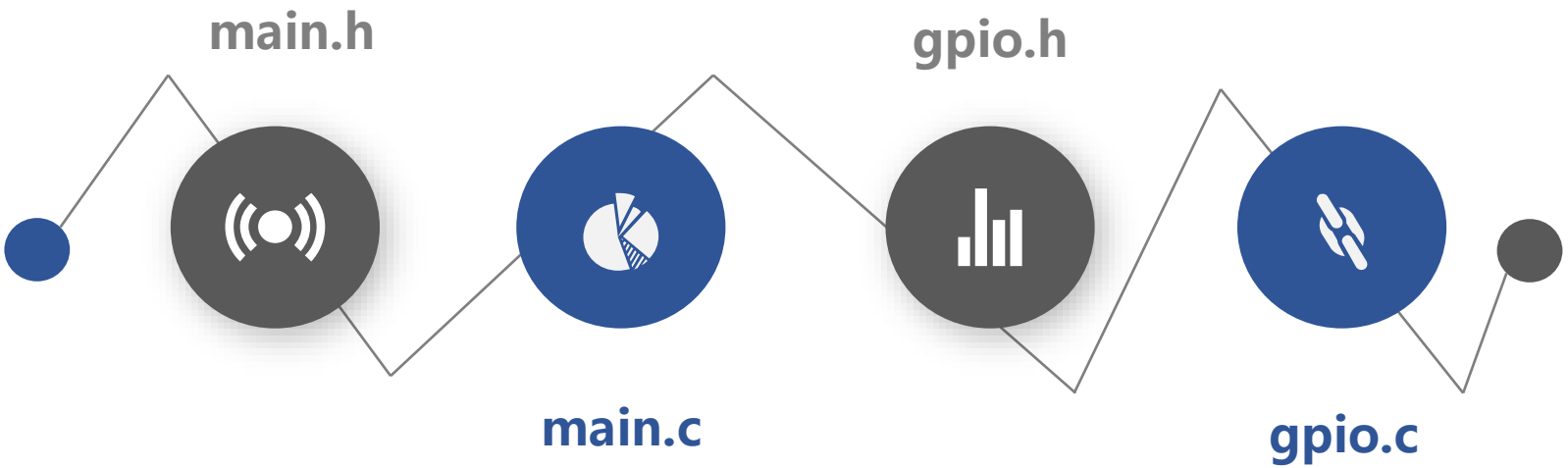
将开发板通过USB线与PC机进行连接

通过工具栏里的“Download”按钮将编译通过的程序下载到开发板上

通过观察开发板上相应LED灯的闪烁情况来验证程序的正确性。

5.3.2 GPIO的HAL库应用实例

GPIO工程源码解析



针对GPIO-LED闪烁工程相关源码共四个文件

5.3.2 GPIO的HAL库应用实例

GPIO工程源码解析

main.c

```
    /* Includes 头文件*/
#include "main.h"
#include "gpio.h"

   /* 函数声明 */
void SystemClock_Config(void); //设置系统时钟

int main(void)
{
      /* 将所有的外设复位，
         并初始化Flash和系统滴答时钟Systick */
 HAL_Init();
     /*配置系统时钟*/
 SystemClock_Config();
     /***GPIO初始化，函数实现定义在gpio.c文件中
      *此代码对应于STM32CubeMX软件中的Pinout &
      Configuration设置的引脚参数
     */
 MX_GPIO_Init();
```

```
while (1)
 {
     //以下为用户编写的应用程序代码

      /*调用HAL_GPIO_TogglePin函数
        使LED1---PB5状态翻转 */

HAL_GPIO_TogglePin(LED1_GPIO_Port,LED1_Pin);
HAL_Delay(100);
      //调用HAL库函数HAL_Delay延时100毫秒
      /*调用HAL_GPIO_TogglePin函数
        使LED2---PE5状态翻转 */
  HAL_GPIO_TogglePin(LED2_GPIO_Port,LED2_Pin);
  HAL_Delay(100);
      //调用HAL库函数HAL_Delay延时100毫秒
 }
}
```

5.3.2 GPIO的HAL库应用实例

GPIO工程源码解析

main.c

```
    /***系统时钟配置，对应于STM32CubeMX软件中的系统时钟操作*/
void SystemClock_Config(void)
{
      /*OSC外部晶振初始化结构体变量，用于打开HSE，设置PLL等 */
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
      /*SYSCLK初始化结构体变量，用于选择sysclk的时钟源、设置AHB预分频系数*/
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
     /* 给RCC_OscInitStruct的成员赋值，用于选择系统时钟源，设置PLL、PLLMul的值等 */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;//HSE为外部晶振8MHz
  RCC_OscInitStruct.HSEState = RCC_HSE_ON; //打开HSE
  RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;//设置HSE预分频系数为1
  RCC_OscInitStruct.HSIState = RCC_HSI_ON; //打开HSI（高速内部时钟，HSI=8MHz）
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;//打开PLL
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;//PLL的输入时钟选择为HSE
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;//设置PLLMul为9，倍频到72MHz
```

5.3.2 GPIO的HAL库应用实例

GPIO工程源码解析

main.c

```
      /***调用HAL_RCC_OscConfig()函数对系统时钟进行初始化配置， 通过if语句判断是否成功，
           如果初始化配置不成功，则由Error_Handler()进行处理*/
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
 {     Error_Handler();//出错处理函数    }
      /*给RCC_ClkInitStruct的成员赋值，用于选择SYSCLK的时钟源， 设置AHB、APB1、APB2的时钟，
      此代码对应于STM32CubeMX软件中的Clock Configuration配置的系统时钟操作 */
 RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
         /*将 SYSCLK的时钟源选为PLLCLK */
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
         /* 将AHB Prescaler的分频值设为 1，AHB时钟频率=72MHz*/
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
         /* 将APB1 Prescaler的分频值设为2，APB2时钟频率=PCLK1 = 36 MHz*/
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
         /*将APB2 Prescaler的分频值设为1，APB2时钟频率=PCLK2 = 72MHz */
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
         /*判断SYSCLK时钟初始化配置是否成功， 如果初始化配置不成功，则由Error_Handler()进行处理*/
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {   Error_Handler();     }
}
```

5.3.2 GPIO的HAL库应用实例

GPIO工程源码解析

main.c

```
/* 错误处理函数 */
void Error_Handler(void)
{
          //错误处理代码由用户编写
}


/* 断言函数，如果使用断言的话，需定义宏USE_FULL_ASSERT ，这里没有使用断言*/
#ifdef  USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line)
{
   //断言失败函数，具体处理代码由用户编写
}
#endif /* USE_FULL_ASSERT */
```

5.3.2 GPIO的HAL库应用实例

GPIO工程源码解析

```
/***GPIO初始化*此代码对应于STM32CubeMX软件中的Pinout & Configuration设置的引脚参数*/

void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* 使能GPIOB、GPIOE端口时钟 */
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

   /* 配置GPIO初始引脚LED2-PE5为低电平 */
  HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin, GPIO_PIN_RESET);

  /* 配置GPIO初始引脚LED1-PB5为低电平 */
  HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
```

```
    /* LED2-PE5引脚参数配置 */
  GPIO_InitStruct.Pin = LED2_Pin; //I/O引脚设置为LED2-PE5
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    //引脚模式设置为推挽输出
  GPIO_InitStruct.Pull = GPIO_NOPULL; //无上拉
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
     //输出速度为低速
  HAL_GPIO_Init(LED2_GPIO_Port, &GPIO_InitStruct);
    //调用初始化函数初始化PE5

     /* LED1-PB5引脚参数配置 */
  GPIO_InitStruct.Pin = LED1_Pin; //I/O引脚设置为LED1-PB5
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
     //引脚模式设置为推挽输出
  GPIO_InitStruct.Pull = GPIO_NOPULL; //无上拉
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
     //输出速度为低速
  HAL_GPIO_Init(LED1_GPIO_Port, &GPIO_InitStruct);
     //调用初始化函数初始化PB5
}
```

5.3.2 GPIO的HAL库应用实例

GPIO工程源码解析

main.h

```
   /*条件编译，防止头文件被重复包含*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

    /* Includes 头文件,stm32f1xx_hal.h这个头文件很重要*/
#include "stm32f1xx_hal.h"
```

```
    /* 函数声明 */
void Error_Handler(void);
    /* 端口引脚的宏定义，方便修改 */
#define LED2_Pin GPIO_PIN_5
#define LED2_GPIO_Port GPIOE
#define LED1_Pin GPIO_PIN_5
#define LED1_GPIO_Port GPIOB

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */
```

本章小结

5.1 GPIO概述

5.2 STM32的GPIO工作原理

- GPIO引脚
- GPIO内部结构
- GPIO工作模式
- GPIO输出速度

5.3 GPIO模块的HAL库接口函数及应用

- 接口函数
- 应用实例

作业

1. 思考并分析使用STM32F103RCT6，不借助外部扩展芯片的情况下，使用GPIO功能最多能独立控制多少LED小灯的亮灭。

作业提交：

学习平台s.ecust.edu.cn，提交截止时间：2025.9.28