

第六章 通用输入输出接口（二）

主讲人：漆强

电子科技大学

ytqiqiang@163.com

本章内容



基于HAL库方式控制GPIO



任务实践



硬件抽象层的设计与移植

教学目标



掌握基于库函数的程序开发方式



了解库函数的设计思想



掌握硬件抽象层的设计思想和实现方法



电子科技大学
University of Electronic Science and Technology of China

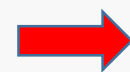
6.5 基于HAL库方式控制GPIO

```
1. #include "stm32f411xe.h"
```

// 头文件中包含所有外设寄存器的定义

```
2. int main()
```

基于寄存器方式控制GPIO



本质：配置寄存器

```
3. {
```

```
4.     uint32_t delay = 1000000;
```

// 延时变量

1

```
5.     RCC->AHB1ENR |= 1<<0;
```

```
6.     GPIOA->MODER |= 1<<(5*2);
```

```
7.     while(1)
```

```
8.     {
```

```
9.         GPIOA->BSRR |= 1<<5;
```

// 设置 bit5 为 1, 2 输出高电平, 开启 LD2

```
10.        delay = 1000000;
```

```
11.        while(delay--);
```

```
12.        GPIOA->BSRR |= 1<<(5+16);
```

```
13.        delay = 1000000;
```

```
14.        while(delay--);
```

```
15.    }
```

```
16. }
```

GPIO引脚的初始化

根据引脚的编号和功能来设置相关寄存器的对应位

GPIO引脚的操作

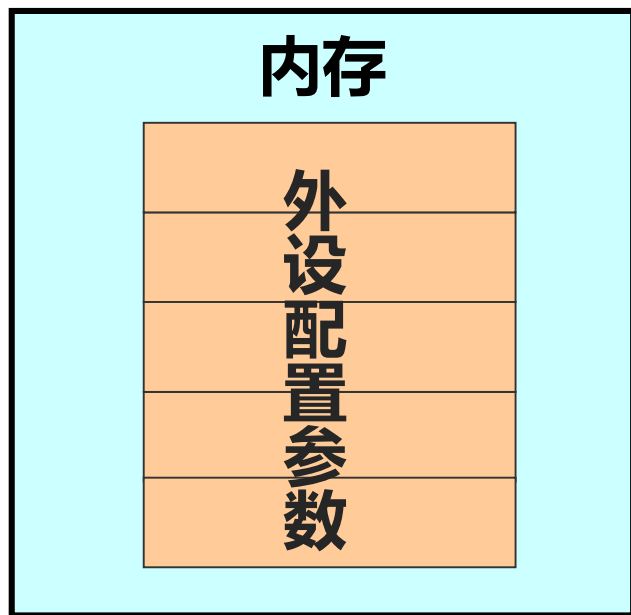
- 设置相关寄存器的对应位控制GPIO引脚输出高/低电平（ODR和BSRR）
- 读取相关寄存器的对应位获取GPIO引脚的电平状态（IDR）

简化配置过程

简化寄存器的配置过程

1

在内存中开辟一块存储区域，并对其初始化

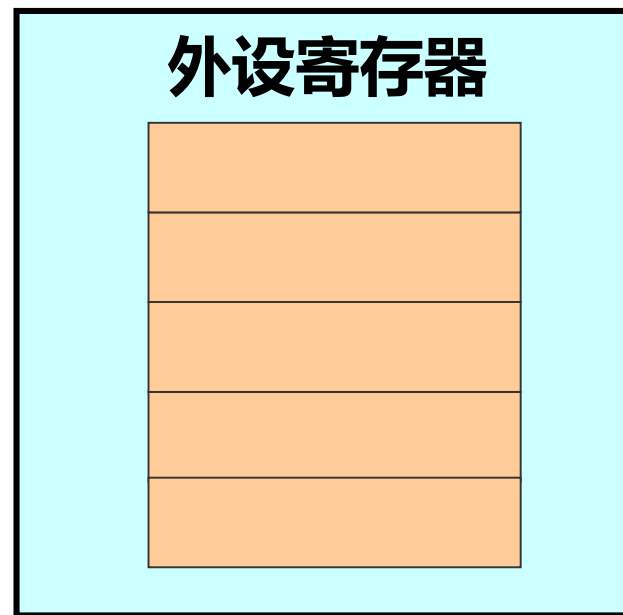


接口函数



2

利用接口函数将参数写入到对应的外设寄存器



实现方式

以HAL库的GPIO模块为例

构造存储区域

设计相应的数据类型，用于存放配置到外设寄存器的参数

stm32f4xx_hal_gpio.h文件完成数据类型的定义及接口函数的声明

设计接口函数

将数据类型中的参数写入到对应的外设寄存器中

stm32f4xx_hal_gpio.c文件完成接口函数的实现



电子科技大学
University of Electronic Science and Technology of China

1 GPIO外设的数据类型

概述

GPIO外设数据类型的概述



1 引脚初始化数据类型

```
1. typedef struct
2. {
3.     uint32_t Pin; // 指定需要配置的 GPIO 引脚, 该参数可以是 GPIO_pins 的值之一
4.     uint32_t Mode; // 指定所选引脚的工作模式, 该参数可以是 GPIO_mode 的值之一
5.     uint32_t Pull; // 指定所选引脚的上/下拉电阻, 该参数可以是 GPIO_pull 的值之一
6.     uint32_t Speed; // 指定所选引脚的速度, 该参数可以是 GPIO_speed 的值之一
7.     // 将外设连接至所选择的引脚, 该参数可以是 Alternate_function_selection 的值之一
8.     uint32_t Alternate;
9. }GPIO_InitTypeDef;
```

结构体类型, 包括5个成员变量

成员变量Pin的取值范围

宏常量定义	含义	宏常量定义	含义
GPIO_PIN_0	选择该端口的引脚0	GPIO_PIN_8	选择该端口的引脚8
GPIO_PIN_1	选择该端口的引脚1	GPIO_PIN_9	选择该端口的引脚9
GPIO_PIN_2	选择该端口的引脚2	GPIO_PIN_10	选择该端口的引脚10
GPIO_PIN_3	选择该端口的引脚3	GPIO_PIN_11	选择该端口的引脚11
GPIO_PIN_4	选择该端口的引脚4	GPIO_PIN_12	选择该端口的引脚12
GPIO_PIN_5	选择该端口的引脚5	GPIO_PIN_13	选择该端口的引脚13
GPIO_PIN_6	选择该端口的引脚6	GPIO_PIN_14	选择该端口的引脚14
GPIO_PIN_7	选择该端口的引脚7	GPIO_PIN_15	选择该端口的引脚15
GPIO_PIN_All	选择该端口所有引脚		

成员变量Mode的取值范围

宏常量定义	含义
GPIO_MODE_INPUT	浮空输入模式
GPIO_MODE_OUTPUT_PP	推挽输出模式
GPIO_MODE_OUTPUT_OD	开漏输出模式
GPIO_MODE_AF_PP	复用功能下的推挽模式
GPIO_MODE_AF_OD	复用功能下的开漏模式
GPIO_MODE_ANALOG	模拟模式

注意：与引脚中断相关的工作模式将在第七章外部中断中介绍

成员变量Pull的取值范围

宏常量定义	含义
GPIO_NOPULL	没有上拉或下拉电阻激活
GPIO_PULLUP	上拉电阻激活
GPIO_PULLDOWN	下拉电阻激活

成员变量Speed的取值范围

宏常量定义	含义
GPIO_SPEED_FREQ_LOW	引脚输出速度2MHz
GPIO_SPEED_FREQ_MEDIUM	引脚输出速度12.5MHz ~ 50MHz
GPIO_SPEED_FREQ_HIGH	引脚输出速度25MHz ~ 100MHz
GPIO_SPEED_FREQ_VERY_HIGH	引脚输出速度50MHz ~ 200MHz

成员变量Alternate的取值范围

- Alternate表示引脚的复用功能;
- 由于不同型号的STM32微控制器片内集成的外设不同, 因此该成员变量的取值范围由芯片型号决定。
- 以STM32F4系列芯片为例, 通过查阅stm32f4xx_hal_gpio_ex.h文件可以了解Alternate的取值范围;
- 该成员变量的取值一般通过CubeMX软件分配, 不需要用户手动设置;

2 引脚电平状态数据类型

```
1. typedef enum
```

枚举类型

```
2. {
```

```
3.     GPIO_PIN_RESET = 0,           // 引脚低电平状态
```

```
4.     GPIO_PIN_SET           // 引脚高电平状态
```

```
5. }GPIO_PinState;
```

使用枚举类型的好处：提高了程序的可读性，并通过限定变量的取值范围，来确保变量的合法性

3 端口数据类型：指向端口寄存器组的结构体指针

宏常量定义	含义
GPIOA	选择端口A
GPIOB	选择端口B
GPIOC	选择端口C
GPIOD	选择端口D
GPIOE	选择端口E
GPIOH	选择端口H

注意

- ① 不同型号的STM32微控制器的端口数量各不相同；
- ② 端口数据类型的定义是在以芯片型号命名的.h文件中，如STM32F411系列MCU对应的头文件stm32f411xe.h

初始化步骤

使用HAL库的引脚初始化步骤

定义变量

01

设置模式

02

调用函数

03

利用引脚初始化
结构体类型

`GPIO_InitType`
Def 定义一个结
构体变量。

按照引脚的工作
模式，依次对该
结构体的成员变
量赋值，如pin、
mode、pull等。

调用初始化函数
`HAL_GPIO_Init`
将配置参数写入
到对应的寄存器，
入口参数为端口
号和结构体变量。

演示例程：GPIO引脚初始化

1 例程目标

掌握使用HAL库进行GPIO引脚初始化的方法

2 例程内容

- ① 设置引脚PA5和PA15为推挽输出模式，无上/下拉电阻，输出低速；
- ② 设置引脚PC13为浮空输入模式；

引脚初始化程序

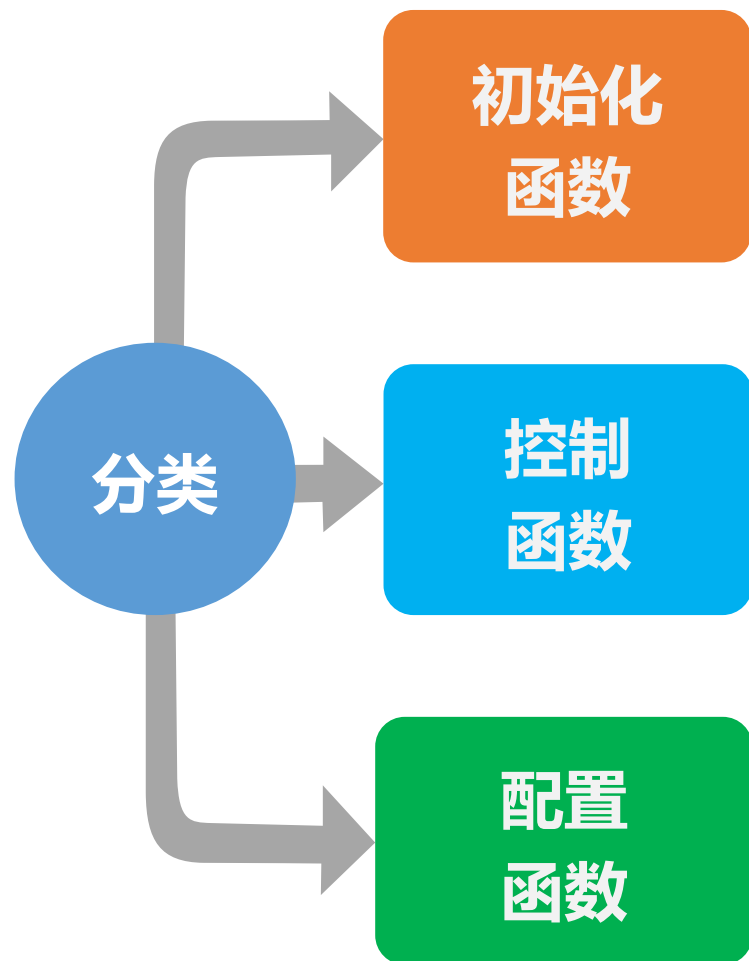
```
1.  GPIO_InitTypeDef GPIO_InitStructure = {0}; // 定义引脚初始化变量，并初始化为0
2.  // 配置引脚 PA5 和 PA15: 推挽输出，无上拉/下拉，输出低速
3.  GPIO_InitStructure.Pin = GPIO_PIN_5 | GPIO_PIN_15;
4.  GPIO_InitStructure.Mode = GPIO_MODE_OUTPUT_PP;
5.  GPIO_InitStructure.Pull = GPIO_NOPULL;
6.  GPIO_InitStructure.Speed = GPIO_SPEED_FREQ_LOW;
7.  HAL_GPIO_Init(GPIOA, &GPIO_InitStructure); // 调用初始化函数完成引脚初始化
8.  // 配置引脚 PC13: 浮空输入，无上拉/下拉
9.  GPIO_InitStructure.Pin = GPIO_PIN_13;
10. GPIO_InitStructure.Mode = GPIO_MODE_INPUT;
11. GPIO_InitStructure.Pull = GPIO_NOPULL;
12. HAL_GPIO_Init(GPIOC, &GPIO_InitStructure); // 调用初始化函数完成引脚初始化
```

引脚PA5和PA15属于同一个端口GPIOA，且配置参数相同，可以采用按位或的方式同时配置，以简化程序。

2 GPIO外设的接口函数

概述

GPIO外设接口函数的概述

初始化
函数

HAL_GPIO_Init : 用于完成引脚的初始化;
HAL_GPIO_DeInit : 用于复位引脚到初始状态;

控制
函数

HAL_GPIO_ReadPin : 用于读取引脚电平状态
HAL_GPIO_WritePin : 用于设置引脚电平状态
HAL_GPIO_TogglePin : 用于翻转引脚电平状态

配置
函数

HAL_GPIO_LockPin: 用于锁定引脚的配置

1 引脚初始化函数：HAL_GPIO_Init

接口函数：HAL_GPIO_Init

函数原型	void HAL_GPIO_Init (GPIO_TypeDef * GPIOx, GPIO_InitTypeDef * GPIO_Init)
功能描述	引脚初始化
入口参数1	GPIOx : 引脚端口号, 取值范围是 GPIOA ~ GPIOK
入口参数2	GPIO_Init : 指向引脚初始化类型GPIO_InitTypeDef的结构体指针, 该结构体包含指定引脚的配置参数
返回值	无
注意事项	该函数可以由CubeMX软件自动生成

2 引脚复位函数：HAL_GPIO_DeInit

接口函数：HAL_GPIO_DeInit

函数原型	void HAL_GPIO_DeInit (GPIO_TypeDef * GPIOx, uint32_t GPIO_Pin)
功能描述	复位引脚到初始状态
入口参数1	GPIOx: 引脚端口号, 取值范围是 GPIOA ~ GPIOK
入口参数2	GPIO_Pin: 引脚号, 取值范围是 GPIO_PIN_0 ~ GPIO_PIN_15
返回值	无
注意事项	该函数需要用户调用

3 读取引脚函数：HAL_GPIO_ReadPin

接口函数：HAL_GPIO_ReadPin

函数原型	GPIO_PinState HAL_GPIO_ReadPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
功能描述	读取引脚的电平状态
入口参数1	GPIOx: 引脚端口号, 取值范围是 GPIOA ~ GPIOK
入口参数2	GPIO_Pin: 引脚号, 取值范围是 GPIO_PIN_0 ~ GPIO_PIN_15
返回值	GPIO_PinState: 表示引脚电平状态的枚举类型变量, 当取值为: <ul style="list-style-type: none">● GPIO_PIN_SET : 表示读到高电平● GPIO_PIN_RESET : 表示读到低电平
注意事项	该函数需要用户调用

函数源码

```
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
    GPIO_PinState bitstatus;           // 定义引脚电平状态变量
    /* Check the parameters */
    assert_param(IS_GPIO_PIN(GPIO_Pin)); // 检测输入参数GPIO_Pin的合法性
    if((GPIOx->IDR & GPIO_Pin) != (uint32_t)GPIO_PIN_RESET)
    {
        bitstatus = GPIO_PIN_SET;
    }
    else
    {
        bitstatus = GPIO_PIN_RESET;
    }
    return bitstatus;
}
```

程序解析

- 读取输入数据寄存器IDR的值与指定引脚进行按位与的操作
- 结果不为0，返回高电平，表示该引脚当前的电平状态为高电平
- 结果为0，返回低电平，表示该引脚当前的电平状态为低电平

4 写入引脚函数：HAL_GPIO_WritePin

接口函数：HAL_GPIO_WritePin

函数原型	void HAL_GPIO_WritePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
功能描述	设置引脚输出高/低电平
入口参数1	GPIOx : 引脚端口号, 取值范围是 GPIOA ~ GPIOK
入口参数2	GPIO_Pin : 引脚号, 取值范围是 GPIO_PIN_0 ~ GPIO_PIN_15
入口参数3	PinState : 表示引脚电平状态的枚举类型变量, 当取值为: ● GPIO_PIN_SET : 表示输出高电平 ● GPIO_PIN_RESET : 表示输出低电平
返回值	无
注意事项	该函数需要用户调用

函数源码

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

```
{
```

```
    /* Check the parameters */
```

```
    assert_param(IS_GPIO_PIN(GPIO_Pin));
```

```
    assert_param(IS_GPIO_PIN_ACTION(PinState));
```

```
    if(PinState != GPIO_PIN_RESET)
```

```
    {
```

```
        GPIOx->BSRR = GPIO_Pin;
```

```
    }
```

```
    else
```

```
    {
```

```
        GPIOx->BSRR = (uint32_t)GPIO_Pin << 16U;
```

```
    }
```

```
}
```

检测输入参数GPIO_PIN
和PinState的合法性

程序解析

- 将设置的电平状态写入置位/复位寄存器BSRR
- 输出高电平，写入BSRR低16位中的对应位
- 输出低电平，写入BSRR高16位中的对应位

5 翻转引脚函数：HAL_GPIO_TogglePin

接口函数：HAL_GPIO_TogglePin

函数原型	void HAL_GPIO_TogglePin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
功能描述	翻转引脚状态
入口参数1	GPIOx: 引脚端口号, 取值范围是 GPIOA ~ GPIOK
入口参数2	GPIO_Pin: 引脚号, 取值范围是 GPIO_PIN_0 ~ GPIO_PIN_15
返回值	无
注意事项	该函数需要用户调用

函数源码

```
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
    /* Check the parameters */
    assert_param(IS_GPIO_PIN(GPIO_Pin));

    if ((GPIOx->ODR & GPIO_Pin) == GPIO_Pin)
    {
        GPIOx->BSRR = (uint32_t)GPIO_Pin << GPIO_NUMBER;
    }
    else
    {
        GPIOx->BSRR = GPIO_Pin;
    }
}
```

检测输入参数GPIO_PIN的合法性

代表16U，表示将常数16强制转换为无符号整型

程序解析

- 读取输出数据寄存器ODR的值与指定引脚进行按位与操作
- 结果为真，表明原来为高电平，控制BSRR寄存器高16位的对应位输出低电平
- 结果为假，表明原来为低电平，控制BSRR寄存器低16位的对应位输出高电平

6 锁定引脚函数：HAL_GPIO_LockPin

接口函数：HAL_GPIO_LockPin

函数原型	HAL_StatusTypeDef HAL_GPIO_LockPin (GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)
功能描述	锁定引脚的配置
入口参数1	GPIOx: 引脚端口号, 取值范围是 GPIOA ~ GPIOK
入口参数2	GPIO_Pin: 引脚号, 取值范围是 GPIO_PIN_0 ~ GPIO_PIN_15
返回值	HAL_StatusTypeDef: 表示操作结果的枚举类型变量, 当取值为 HAL_OK : 表示引脚锁定成功 HAL_ERROR : 表示引脚锁定失败
注意事项	该函数需要用户调用

3 HAL库的设计思想

面向对象

HAL库借鉴面向对象的设计思想

引脚的属性

引脚的编号、工作模式、上拉/下拉电阻、输出速度、复用功能

引脚的操作

引脚初始化、读取引脚、写入引脚、翻转引脚、锁定引脚

数据

数据类型

外设的属性

设计相应的数据类型来表示外设的各种功能

对象

方法

接口函数

外设的操作

设计相应的接口函数执行外设的各种操作：初始化、控制、读取外设状态等

开发特点

基于库函数的程序开发方式的特点



屏蔽底
层硬件

编程者只需要了解库函数中相关接口函数的功能，并按照要求传入参数，利用返回值完成操作即可，不需要过多了解底层硬件。

提高开
发效率

开发难度较小，开发周期较短，后期的维护升级、以及硬件平台的移植等工作量较小。

程序执
行效率

由于考虑了程序的稳健性、扩充性和可移植性，程序代码比较繁琐和臃肿，执行效率较低。

使用步骤

HAL库中GPIO模块的使用步骤

1

使能时钟

使能引脚所属端口的系统总线时钟（AHB）：调用函数
`_HAL_RCC_GPIOx_CLK_ENABLE`

2

设置参数

利用引脚初始化类型GPIO_InitTypeDef定义结构体变量，根据具体需求配置成员变量：Pin、Mode、Pull、Speed、Alternate

3

配置引脚

调用初始化函数HAL_GPIO_Init完成引脚配置，将配置参数写入对应的硬件寄存器

4

控制引脚

使用对应的控制函数完成引脚的控制：函数HAL_GPIO_ReadPin读取引脚状态；函数HAL_GPIO_WritePin设置引脚电平

强强联合

HAL库和CubeMX相结合

使用HAL库编程的问题

- ① 外设初始化过程繁琐，用户需要了解初始化数据类型，根据需求配置成员变量；
- ② 掌握成员变量的含义及取值范围



联合编程

- ① 在CubeMX中完成引脚的初始化设置：如引脚编号、引脚功能、输出模式、上拉/下拉电阻、输出速度等
- ② 掌握接口函数的使用



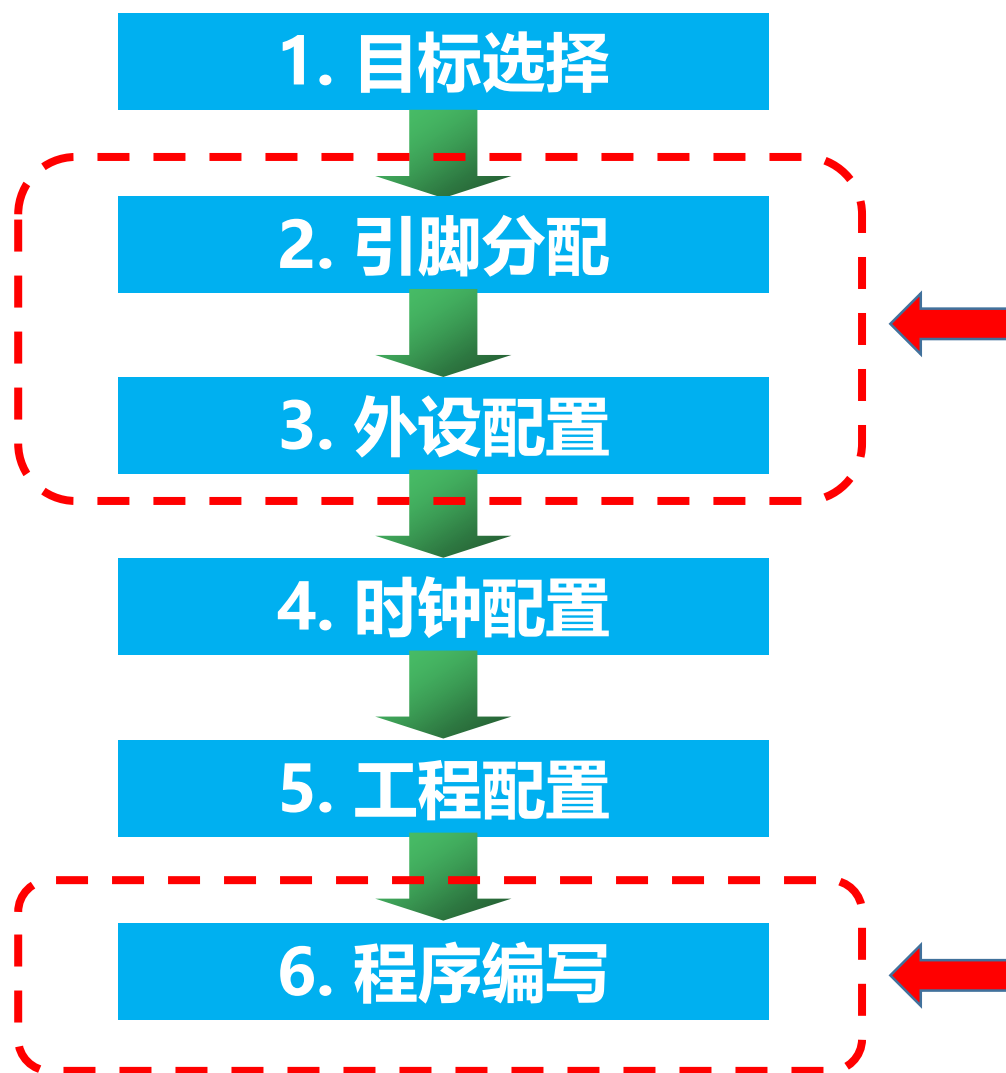
电子科技大学
University of Electronic Science and Technology of China

6.6 任务实践

各个外设模块任务实践的学习建议

- ① 建议大家在英文路径下新建一个名为CubeMX的文件夹存放所有的编程练习，在CubeMX文件夹下面再根据外设模块建立不同的子文件夹，如GPIO、EXIT、TIMER和UART等。
- ② 本课程按照阶梯式的学习方法，对于每一个外设模块的学习，都设置了基础型、进阶型和挑战型等多种任务。建议大家在练习每一个任务时，单独建立一个文件夹，存放与该任务相关的工程文件。

使用流程



对于同一个应用电路的不同任务而言，目标选择、时钟配置和工程配置等步骤的设置都是相同，用户只需要修改引脚分配和外设配置这两个步骤

根据具体任务，在用户所选择的集成开发环境中进行用户代码的添加

工程复制步骤

由于不同任务只需要修改引脚分配和外设配置，因此CubeMX的工程文件不需要每次都重新建立，只需要将之前建立的工程整体复制到新的文件夹即可，步骤如下：



- ① 新建一个文件夹，文件夹的名称和具体的工程应用相关，如驱动指示灯的任务，就可以将文件夹命名为LED；
- ② 将之前建立的一个工程文件夹中的内容整体复制到新文件夹中，并删除其中的MDK-ARM文件夹；
- ③ 修改CubeMX生成的.ioc文件名，和新的工程名称一致，如LED.ioc；
- ④ 点击重命名的IOC文件，启动CubeMX软件。按照新任务的要求完成引脚分配和外设配置后，重新生成MDK工程；
- ⑤ 修改MDK工程的相关设置：如仿真器的选择等；
- ⑥ 添加用户代码，并完成程序的编译、下载和调试等后续操作。

1 基础任务：驱动指示灯

基础任务

基础任务：驱动指示灯

1 任务目标

掌握CubeMX软件配置GPIO外设的方法

2 任务内容

控制Nucleo开发板上的指示灯LD2每隔100ms闪烁

New Project

I need to :

1

Start My project from MCU

ACCESS TO MCU SELECTOR

Start My project from STBoard

ACCESS TO BOARD SELECTOR

Start My project from Cross Sel...

ACCESS TO CROSS SELECTOR

步骤一：目标选择

- ① 启动CubeMX软件，选择“ACCESS TO MCU SELECTOR”，进入目标选择界面；
- ② 在芯片搜索框输入芯片型号“STM32F411RE”；
- ③ 在芯片列表框双击出现的芯片型号，启动芯片配置；

MCU/MPU Filters

Part Number Search

输入芯片型号

2



STM32F411RE

MCUs/MPUs List: 1 item

+ Display similar items

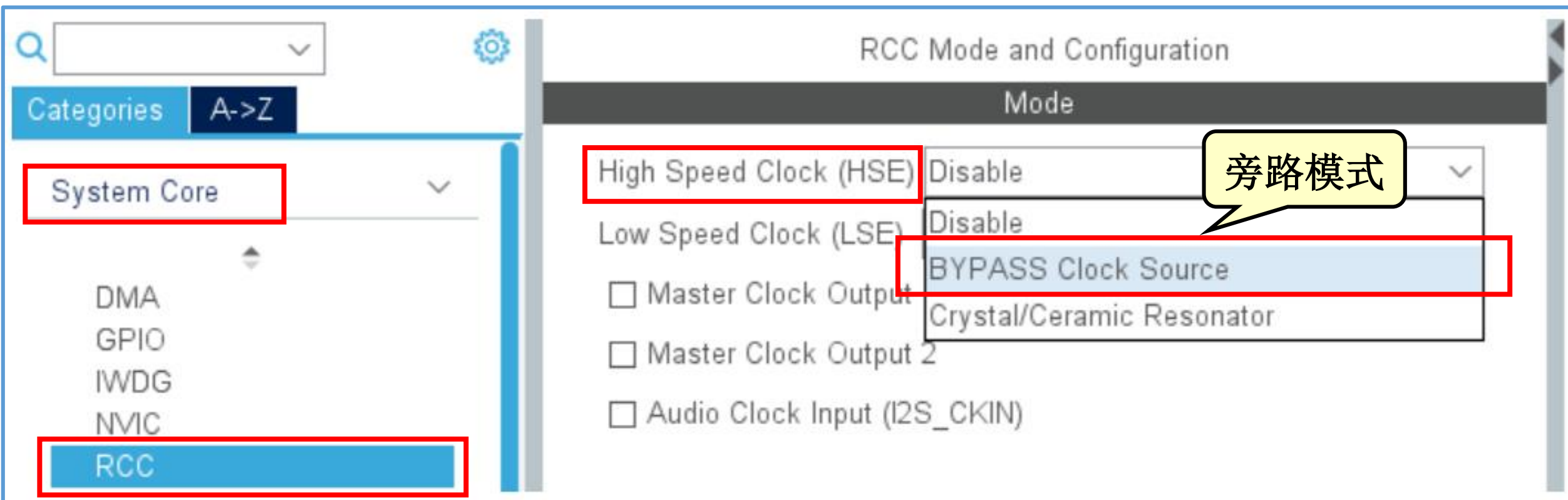
3

双击

*	Part No	Reference	Marketing ...	Unit Price for 1 ...	Board	Package	Flash	RAM	IO	Freq.	GFX Sc...
★	STM32F411RE	STM32F...	Active	3.155	NUCLEO-F411RE	LQFP64	512 kB...	128 kB...	50	100 ...	0.0

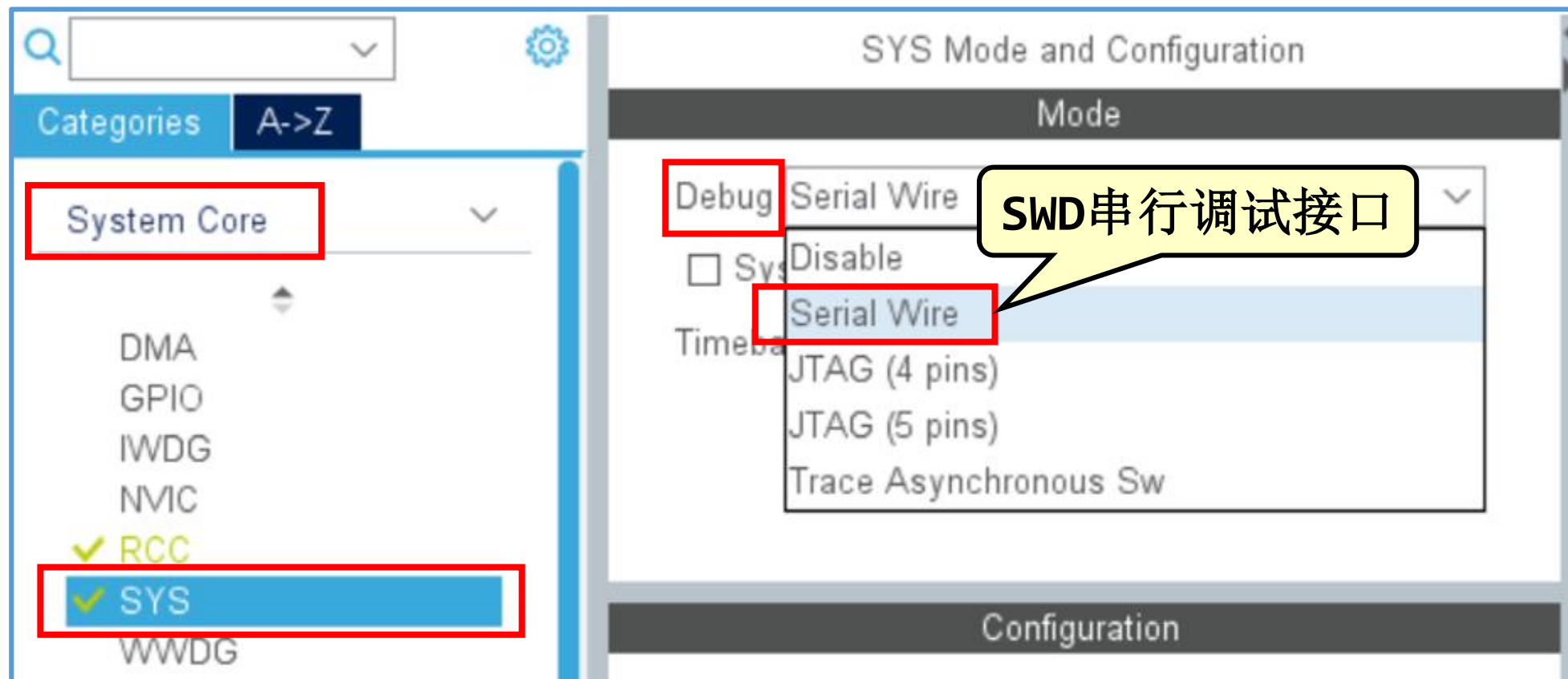
基本配置

基本配置：选择时钟源



基本配置

基本配置：配置调试接口

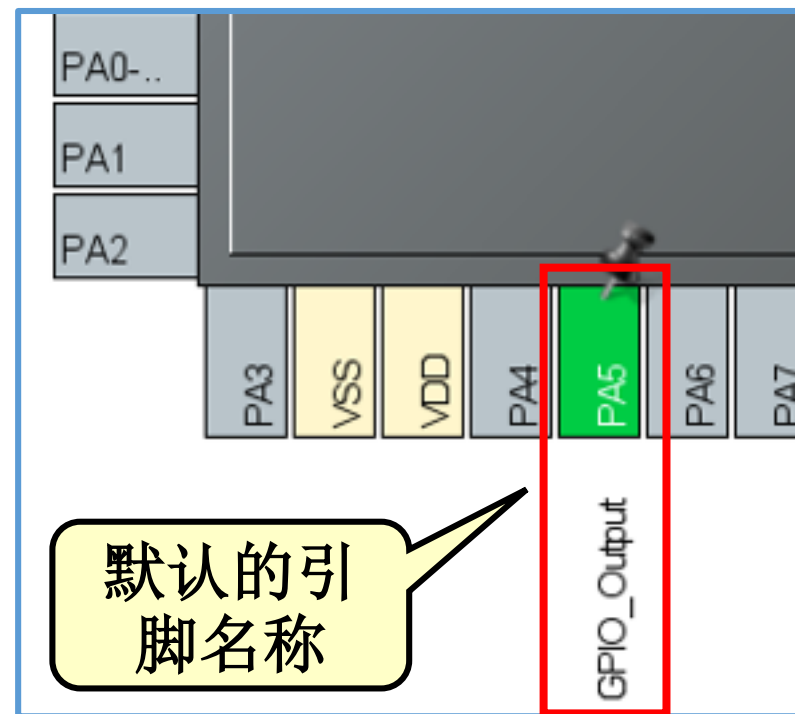
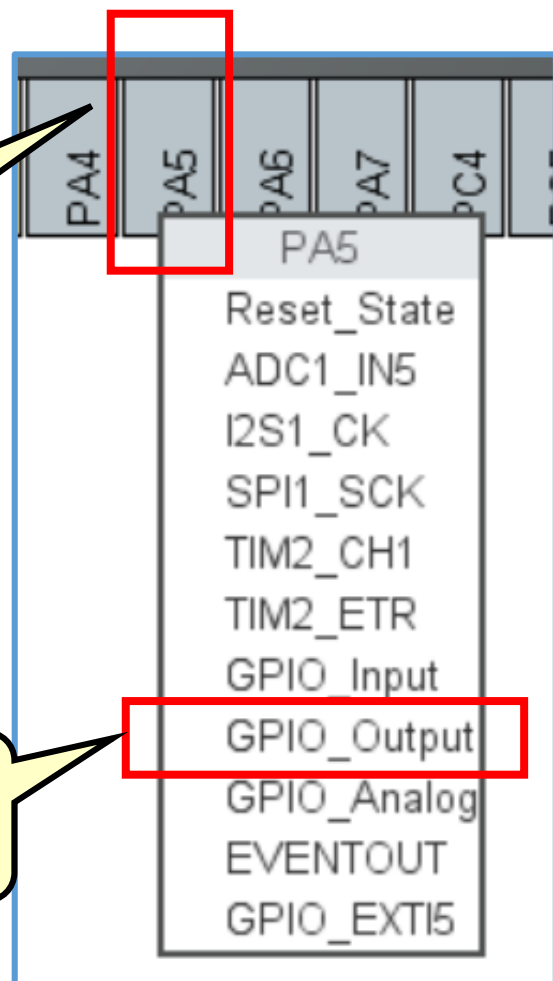


引脚分配

步骤二：引脚分配

单击PA5

控制指示灯，引脚为输出模式



默认的引脚名称

外设配置

步骤三：外设配置

The screenshot shows the STM32CubeMX interface for configuring GPIO. On the left, the 'System Core' and 'GPIO' are selected in the categories list. The main window is titled 'GPIO Mode and Configuration'. Under the 'Configuration' tab, the 'GPIO' checkbox is checked. Below this, there is a 'Search Signals' section with a search bar. At the bottom, a table lists the configured pins. The first row, 'PA5', is highlighted with a red box. A yellow callout bubble points to the 'PA5' row with the text: '在出现的引脚清单中单击需要配置的引脚PA5'.

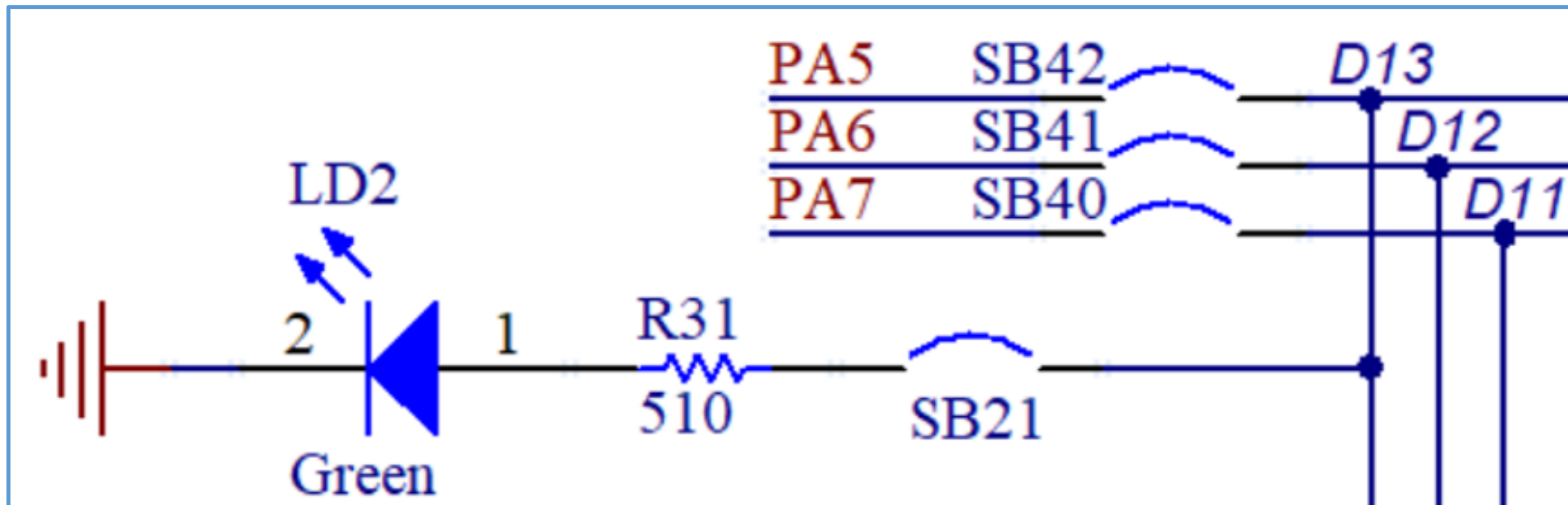
Pin...	Signal ...	GPIO ...	GPIO ...	GPIO ...	Maxim...	User L...	Modified
PA5	n/a	Low	Output...	No pull...	Low		<input type="checkbox"/>

指示灯电路

高电平驱动方式

PA5输出高电平，指示灯LD2开启

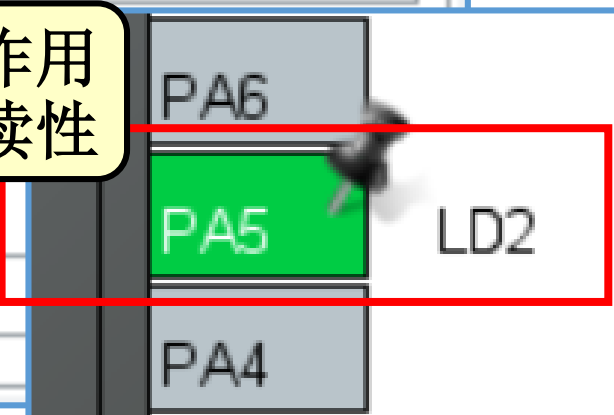
PA5输出低电平，指示灯LD2关闭



配置GPIO外设参数

PA5 Configuration :

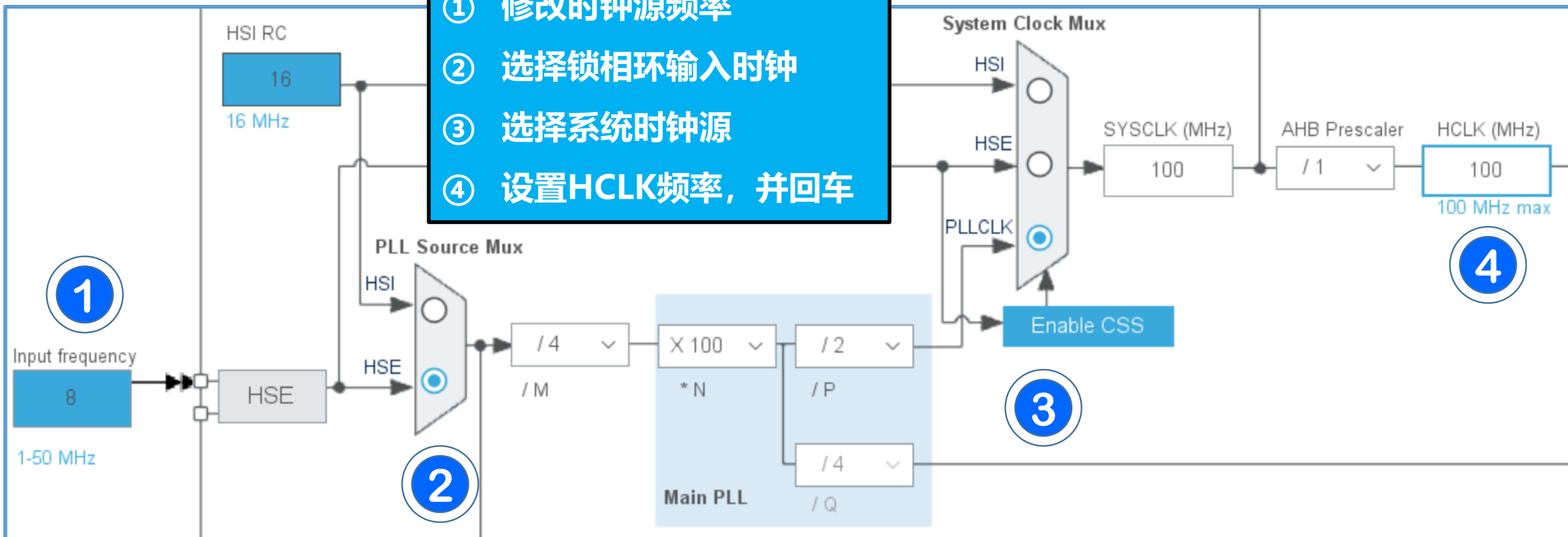
GPIO output level	初始电平	Low	确保指示灯的默认状态为关闭
GPIO mode	引脚模式	Output Push Pull	确保引脚可以输出高电平和低电平
GPIO Pull-up/Pull-down	上/下拉电阻	No pull-up and no pull-down	
Maximum output speed	引脚速度	Low	通过表明引脚的作用来提高程序的可读性
User Label	引脚名称	LD2	



时钟配置

步骤四：时钟配置

- ① 修改时钟源频率
- ② 选择锁相环输入时钟
- ③ 选择系统时钟源
- ④ 设置HCLK频率，并回车



步骤五：工程配置

Project Settings

Project Name
LED

Project Location
C:\Users\qiqiang\Desktop\CubeMX\GPIO

Application Structure
Basic ☐ Do not generate the ma...

Toolchain Folder Location
C:\Users\qiqiang\Desktop\CubeMX\GPIO\LED\

Toolchain / IDE
MDK-ARM

Min Version
V5.27 ☐ Generate Under ...

工程名称 1

保存路径 2

选择IDE 3

生成工程

GENERATE CODE

工程未打开选择
“Open Project”

工程已打开选择“Close”，并
在MDK软件中接受工程中的变化

Open Folder

Open Project

Close

The Code is successfully generated under C:/Users/qi/Desktop/CubeMX/Demo

程序编写

步骤六：程序编写

在main.h文件中对端口和引脚取了一个别名，通过表明引脚的作用来提高程序的可读性

```
28. while (1)
29. {
30.     /* USER CODE END WHILE */
31.     /* USER CODE BEGIN 3 */
32.     HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET); // 开启 LD2
33.     HAL_Delay(100); // 延时 100ms
34.     HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET); // 关闭 LD2
35.     HAL_Delay(100); // 延时 100ms
36. }
37. /* USER CODE END 3 */
```

/* Private defines -----*/
#define LD2_Pin GPIO_PIN_5
#define LD2_GPIO_Port GPIOA

在main.c文件中添加用户代码
添加位置在USER CODE BEGIN
3 和 USER CODE END 3 之间

引脚初始化程序

```
6. static void MX_GPIO_Init(void)
```

```
7. {
```

```
8.     GPIO_InitTypeDef GPIO_InitStructure = {0};
```

```
9.     /* GPIO Ports Clock Enable */
```

```
10.    __HAL_RCC_GPIOH_CLK_ENABLE();
```

```
11.    __HAL_RCC_GPIOA_CLK_ENABLE();
```

```
12.    /*Configure GPIO pin Output Level */
```

```
13.    HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
```

配置参数根据用户在
CubeMX中的设置决定

优点：外设初始化由CubeMX自动生成，用户只需要编写应用程序

```
15.    GPIO_InitStructure.Pin      = LD2_Pin;
```

```
16.    GPIO_InitStructure.Mode     = GPIO_MODE_OUTPUT_PP;
```

```
17.    GPIO_InitStructure.Pull     = GPIO_NOPULL;
```

```
18.    GPIO_InitStructure.Speed    = GPIO_SPEED_FREQ_LOW;
```

```
19.    HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStructure);
```

```
20. }
```

输出推挽

无上拉/下拉电阻

低速输出

工程设置

利用工具栏的三个图标完成工程设置、程序编译和下载



The image shows a screenshot of an IDE's toolbar with the following menu items: File, Edit, View, Project, Flash, Debug, Peripherals, and Tools. The toolbar contains several icons. Three specific icons are highlighted with red boxes and numbered circles:

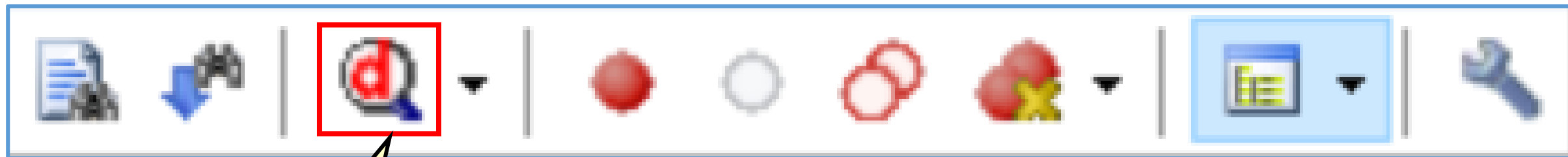
- Icon 1 (Top Right):** A blue circle with the number 1, pointing to a gear icon (Engineering Settings).
- Icon 2 (Middle Left):** A blue circle with the number 2, pointing to a document with a downward arrow icon (Compile).
- Icon 3 (Middle):** A blue circle with the number 3, pointing to a document with a downward arrow and a star icon (Load/Download).

Below the toolbar, three callout boxes provide instructions:

- 程序编译 (Compile Program):** Points to the Compile icon (Icon 2).
- 程序下载 (Download Program):** Points to the Load icon (Icon 3).
- 在Debug标签页中的选择所使用的仿真器，并在仿真器设置窗口中勾选“Reset and Run” (Select the simulator used in the Debug tab page, and check “Reset and Run” in the simulator settings window):** Points to the Engineering Settings icon (Icon 1).
- 工程设置 (Engineering Settings):** Points to the Engineering Settings icon (Icon 1).

程序调试

利用工具栏的Debug图标进入程序调试界面



进入/退出
程序调试

2 进阶任务：按键控制

进阶任务

进阶任务：按键控制

1 任务目标

掌握CubeMX软件配置GPIO外设的方法

2 任务内容

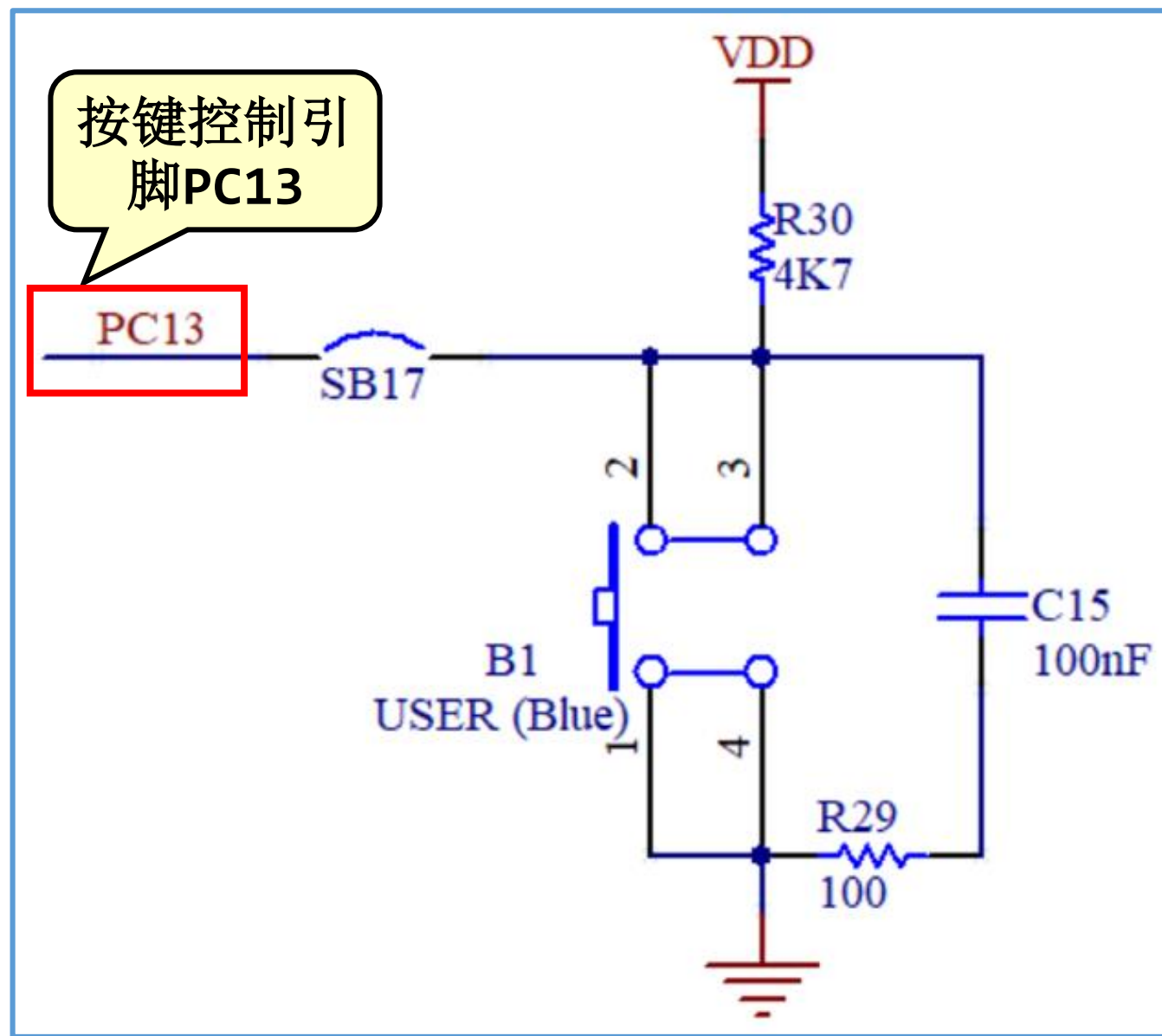
采用查询方式检测按键状态，按键按下后执行操作：翻转指示灯LD2的状态。

按键电路

上拉式按键

按键B1按下，引脚PC13读到低电平

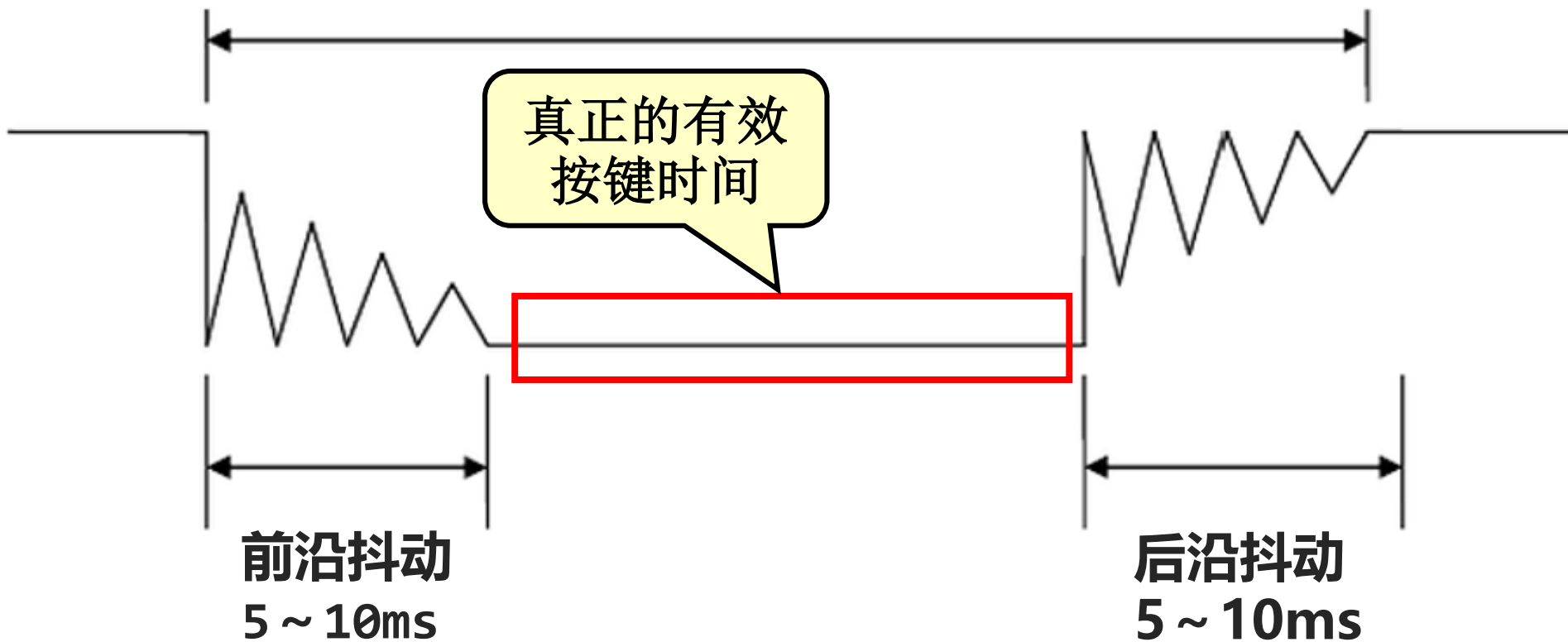
按键B1释放，引脚PC13读到高电平



按键波形

按键的机械特性造成按键的抖动

整个按键周期零点几秒到几秒



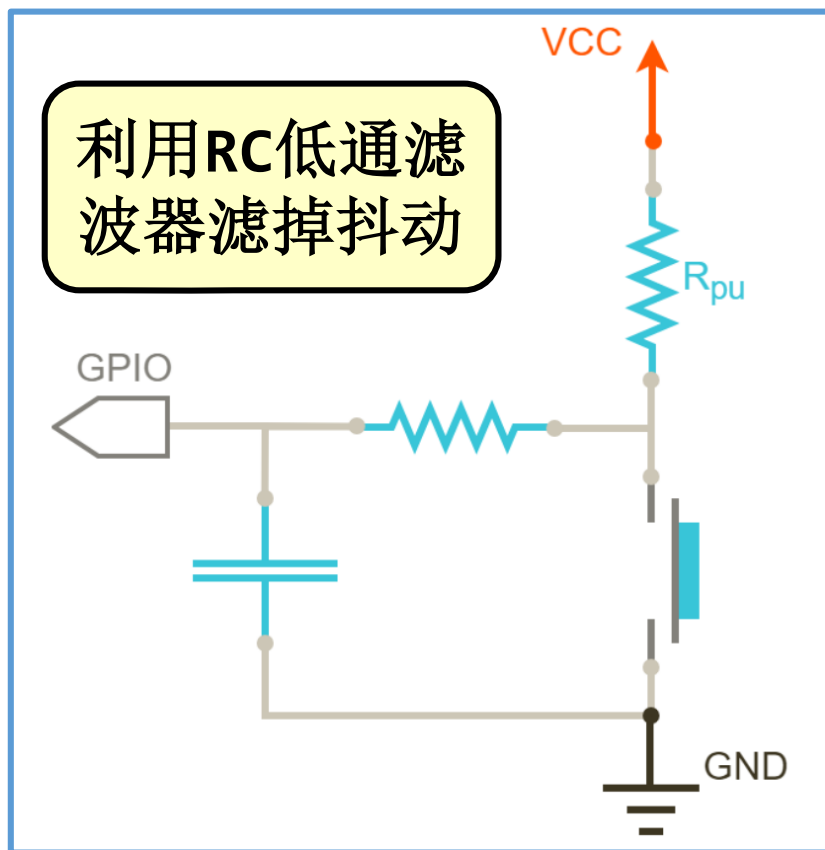
按键抖动问题

按键的抖动会导致一次按键动作被当成多次按键，为确保MCU对按键的一次闭合仅作一次处理，必须消除按键的抖动，在按键处于稳定状态时读取按键的状态。

按键消抖

两种按键消抖方法

硬件消抖



软件消抖

- ① 检测出按键闭合后执行延时程序，延时时间为 $5\text{ms} \sim 10\text{ms}$ ，用于去掉前沿抖动；
- ② 再次检测按键状态，如果保持闭合状态，才认为按下，并执行相应的按键任务；
- ③ 按键的释放可以采用延时或者循环检测的方式去掉后沿抖动；

小贴士



```
graph TD; A[1. 目标选择] --> B[2. 引脚分配]; B --> C[3. 外设配置]; C --> D[4. 时钟配置]; D --> E[5. 工程配置]; E --> F[6. 程序编写];
```

1. 目标选择

2. 引脚分配

3. 外设配置

4. 时钟配置

5. 工程配置

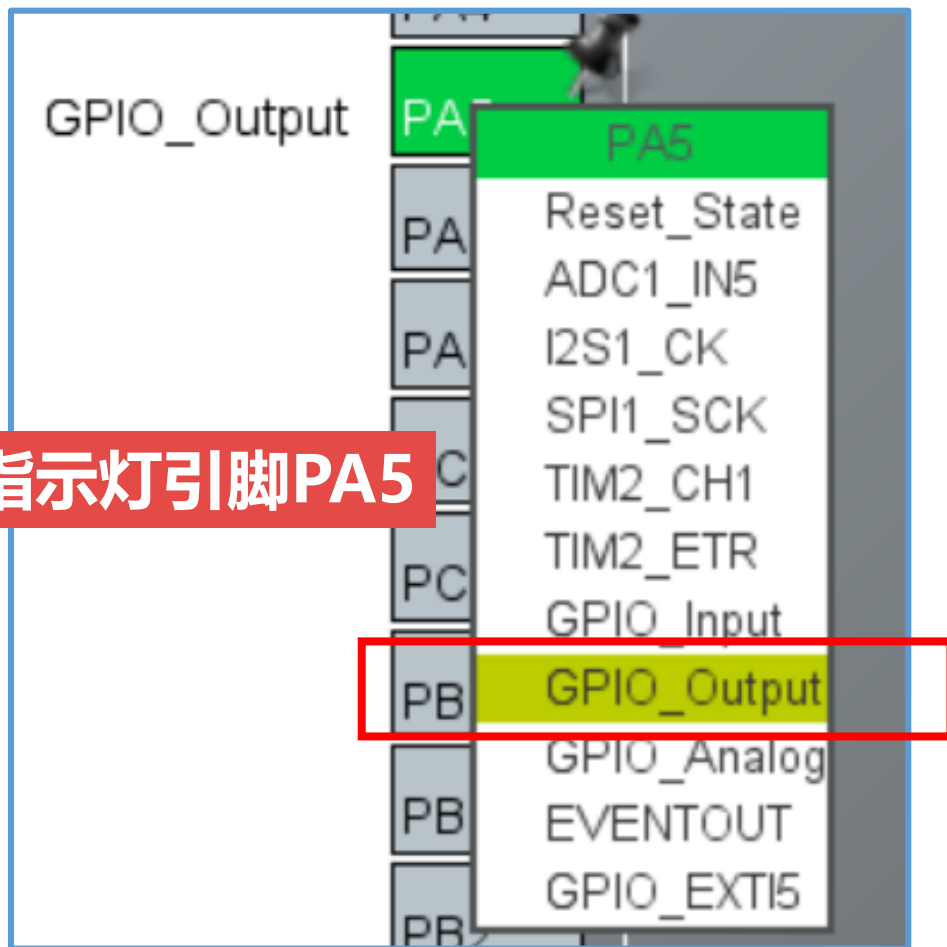
6. 程序编写

对于同一个应用电路的不同任务而言，目标选择、时钟配置和工程配置等步骤的设置都是相同，因此后续的任务实现步骤中将重点介绍引脚分配、外设配置以及程序编写等步骤

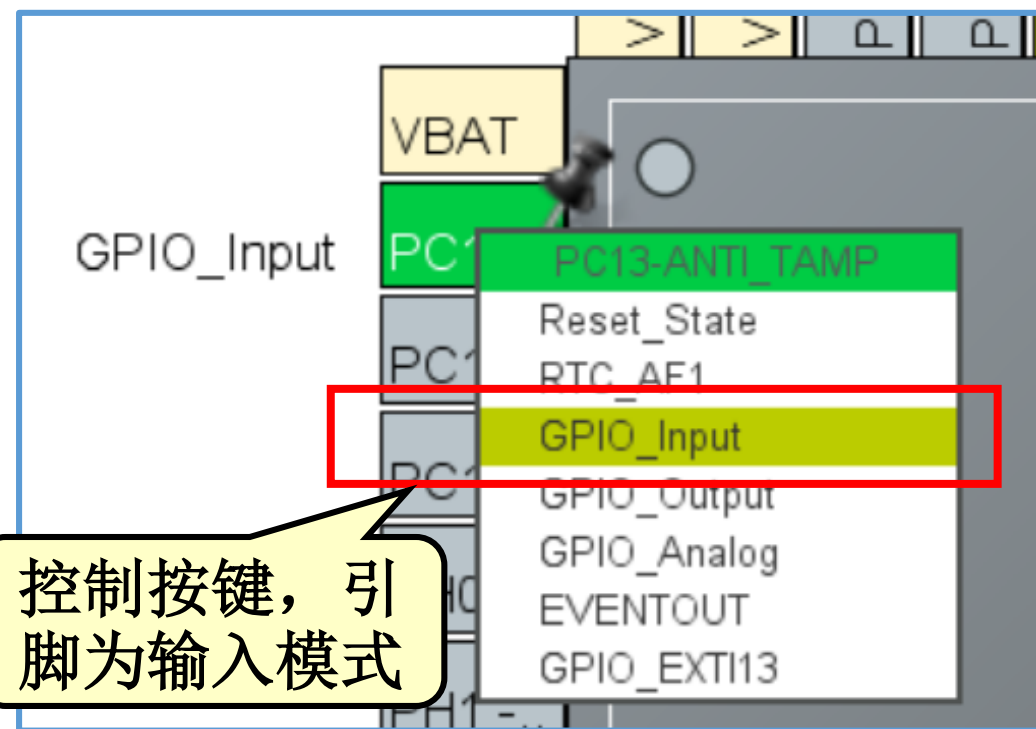
引脚分配

步骤二：引脚分配

设置指示灯引脚PA5



设置按键引脚PC13



步骤三：外设配置

PA5 Configuration :

GPIO output level	初始电平	Low
GPIO mode	引脚模式	Output Push Pull
GPIO Pull-up/Pull-down	上/下拉电阻	No pull-up and no pull-down
Maximum output speed	引脚速度	Low
User Label	引脚名称	LD2

步骤三：外设配置

PC13-ANTI_TAMP Configuration :

GPIO mode

Input mode

引脚模式

GPIO Pull-up/Pull-down

No pull-up and no pull-down

上/下拉电阻

User Label

B1

引脚名称

步骤六：程序编写

在main.c文件中添加用户代码

```
3. while (1)
4. {
5.     /* USER CODE END WHILE */
6.     /* USER CODE BEGIN 3 */
7.     if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET) //按键按下
8.     {
9.         HAL_Delay(10); //延时去抖动
10.        // 按键依然按下
11.        if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET)
12.        {
13.            HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin); //翻转 LD2
14.        }
15.        // 等待按键释放
16.        while(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET);
17.    }
18. }
19. /* USER CODE END 3 */
```

直接延时的方法会占用处理器的执行时间，降低处理器的利用效率。工程应用中常常采用状态机和定时中断去抖动

延时去抖动

3 挑战任务：利用状态机控制按键

挑战任务**挑战任务：利用状态机控制按键****1 任务目标**

掌握状态机设计思想

2 任务内容

利用状态机设计思想编写按键处理程序，按键按下后执行操作：翻转指示灯LD2的状态。

存在问题

延时方式去抖动的问题

直接延时的方法会占用处理器的执行时间，降低处理器的利用效率。工程应用中常常采用状态机和定时中断去抖动

```
7.  if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET)
8.  {
9.      HAL_Delay(10);
10.     // 按键依然按下
11.     if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET)
12.     {
13.         HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin); //翻转 LD2
14.     }
15.     // 等待按键释放
16.     while(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET);
17. }
```

状态机是一个抽象概念，表示把一个过程抽象为若干个状态之间的切换，这些状态之间存在一定的联系。状态机的设计主要包括4个要素：

1. **现态**：是指当前所处的状态。
2. **条件**：当一个条件满足，将会触发一个动作，或者执行一次状态的迁移。
3. **动作**：表示条件满足后执行动作。动作执行完毕后，可以迁移到新的状态，也可以仍旧保持原状态。动作要素不是必需的，当条件满足后，也可以不执行任何动作，直接迁移到新状态。
4. **次态**：表示条件满足后要迁往的新状态。

按键的状态机设计

一：状态定义：根据按键的波形图可以设计三个按键状态

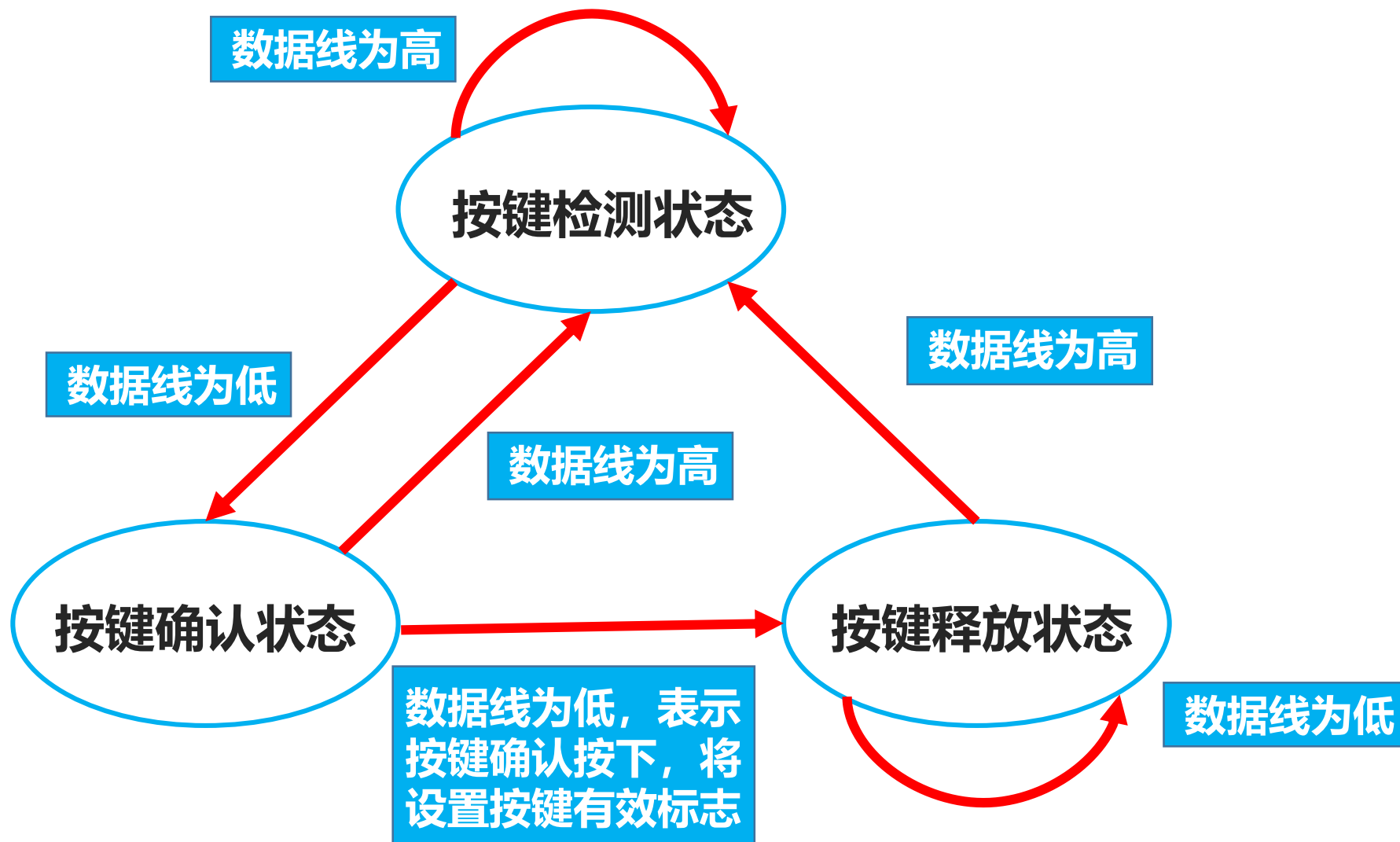
- 1. 按键检测状态：**表示按键没有按下的状态
- 2. 按键确认状态：**表示按键已经按下的状态
- 3. 按键释放状态：**表示等待按键释放的状态

二：状态转换条件（假设低电平表示按键按下）

1. 当处于按键检测状态时，如果数据线为低，则转换到按键确认状态，否则保持当前状态。
2. 当处于按键确认状态时，如果数据线为低，则转换到按键释放状态，并设置按键有效标志；如果数据线为高，则表示可能出现了干扰信号，转换到按键检测状态。
3. 当处于按键释放状态时，如果数据线为高，则转换到按键检测状态，表示完成了本次按键检测，否则保持当前状态。

状态转换图

假设采用上
拉式按键，
低电平表示
按键按下，
高电平表示
按键释放



按键的状态机编程实现

一：利用Switch Case多分支语句，通过检测按键引脚的电平来实现按键状态的转换；

二：利用定时器产生10ms的定时中断，在定时中断服务程序中调用按键状态转换函数。每次执行的间隔10ms，可以有效的消除按键抖动，并提高CPU的利用率。

Categories

A->Z

System Core

Analog

Timers

RTC

TIM1

⚠ TIM2

TIM3

TIM4

⚠ TIM5

⚠ TIM9

✓ TIM10

TIM11

Mode

☒ Activated

Channel1 Disable

☐ One Pulse Mode

Configuration

Reset Configuration

✓ Parameter Settings

✓ User Constants

✓ NVIC Settings

Configure the below parameters :

Search (Ctrl+F)

✓ Counter Settings

Prescaler (PSC - 16 bits ... 10000-1

Counter Mode Up

Counter Period (AutoRelo... 100-1

Internal Clock Division (C... No Division

auto-reload preload Disable

步骤三：外设配置

使能中断

✔ Parameter Settings	✔ User Constants	✔ NVIC Settings	
NVIC Interrupt Table	Enab...	Preemption Pri...	Sub Pri...
TIM1 update interrupt and TIM10 global ...	✔	0	0

使能定时器
10更新中断

注意：引脚分配和引脚参数配置可以参考进阶任务（按键控制）中的相关配置

步骤六：程序编写

```
1.  /* Private typedef ----- */
2.  /* USER CODE BEGIN PTD */
3.  typedef enum
4.  {
5.      KEY_CHECK = 0,    // 按键检测状态
6.      KEY_CONFIRM,      // 按键确认状态
7.      KEY_RELEASE       // 按键释放状态
8.  }KEY_STATE;
9.  /* USER CODE END PTD */
10. /* Private variables ----- */
11. /* USER CODE BEGIN PV */
12. KEY_STATE KeyState    = KEY_CHECK; // 按键状态，初值为按键检测状态
13. uint8_t   KeyFlag     = 0;         // 按键值有效标志，1：有效；0：无效
14. /* USER CODE END PV */
```

定义按键状态数据类型

定义按键状态数据类型，使用枚举类型实现，包括3个枚举常量，用来表示三种按键状态

定义变量

添加位置

头文件、数据类型及函数声明的添加位置

```
/* Private includes -----  
/* USER CODE BEGIN Includes */  
/* USER CODE END Includes */
```

用户头文件

```
/* Private define -----  
/* USER CODE BEGIN PD */  
/* USER CODE END PD */
```

用户常量定义

```
/* Private variables ----  
/* USER CODE BEGIN PV */  
/* USER CODE END PV */
```

用户变量定义

```
/* Private typedef -----  
/* USER CODE BEGIN PTD */  
/* USER CODE END PTD */
```

用户自定义数据类型

```
/* Private macro -----  
/* USER CODE BEGIN PM */  
/* USER CODE END PM */
```

用户宏函数定义

```
/* Private function prototypes  
/* USER CODE BEGIN PFP */  
/* USER CODE END PFP */
```

用户函数声明

补充

变量及函数的命名规则

1. 见名知意：利用英文单词或者其缩写形式定义变量或函数，名称要体现变量的作用或函数的功能，切记不要使用拼音来命名。
2. 变量一般采用名词形式命名，多个单词间利用大小写字母作为间隔。全局变量的首字母大写，如KeyFlag；局部变量的首字母小写，如keyFlag，便于区分全局变量和局部变量。
3. 函数一般采用动宾结构命名，首字母大写，也是利用大小写字母作为间隔，如GetValue等。
4. 宏定义和用户自定义数据类型全部采用大写字母，利用下划线作为间隔，如KEY_STATE等。

主程序代码

```
1. /* USER CODE BEGIN 2 */
2. HAL_TIM_Base_Start_IT(&htim10);    // 使能定时器10更新中断, 启动定时器10
3. /* USER CODE END 2 */
4. /* Infinite loop */
5. /* USER CODE BEGIN WHILE */
6. while (1)
7. {
8.     /* USER CODE END WHILE */
9.     /* USER CODE BEGIN 3 */
10.    if( KeyFlag == 1)                // 检测按键有效标志
11.    {
12.        KeyFlag = 0;                // 清除标志
13.        HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin); // 按键执行任务: 翻转LD2
14.    }
15. }
16. /* USER CODE END 3 */
```

后台程序，检测相关标志位，执行相应操作

定时中断回调函数

1. `/* USER CODE BEGIN 4 */`

添加位置

2. `// 定时器定时中断的回调函数`

3. `void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)`

4. `{`

相当于中断
服务程序

5. `if(htim->Instance == TIM10)`

`// 判断更新中断来源`

6. `{`

7. `switch(KeyState)`

按键检测状态

8. `{`

9. `case KEY_CHECK:`

10. `{`

11. `// 读到低电平，进入按键确认状态`

12. `if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET)`

13. `{`

14. `KeyState = KEY_CONFIRM;`

15. `}`

16. `break;`

17. `}`

- 读到低电平，说明按键按下，进入按键确认状态
- 读到高电平，说明按键没有按下，保持当前状态

定时中断回调函数

按键确认状态

```
18.  case KEY_CONFIRM:
19.  {
20.      // 读到低电平, 进入按键释放状态
21.      if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_RESET)
22.      {
23.          KeyState = KEY_RELEASE;
24.          KeyFlag = 1; // 设置有效按键标志, 表示按键按下后立即执行按键任务
25.      }
26.      // 读到高电平, 可能是干扰信号, 返回初始状态: 按键检测状态
27.      else
28.      {
29.          KeyState = KEY_CHECK;
30.      }
31.      break;
32.  }
```

定时中断回调函数

```
33. case KEY_RELEASE:
34.     {
35.         // 读到高电平, 说明按键释放, 返回初始状态: 按键检测状态
36.         if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_SET)
37.         {
38.             KeyState = KEY_CHECK;
39.             // KeyFlag = 1; // 设置有效按键标志, 表示按键释放后执行按键任务
40.         }
41.         break;
42.     }
43.     default: break;
44. }
45. }
46. }
47. /* USER CODE END 4 */
```

按键释放状态

- 读到高电平, 说明按键释放, 返回初始状态: 按键检测状态, 结束本次按键过程。
- 读到低电平, 说明按键没有释放, 保持当前状态

总结**用户代码添加位置****后台程序**

在while循环中, USER CODE BEGIN 3 和USER CODE END 3

外设启动

USER CODE BEGIN 2 和USER CODE END 2

用户函数和中断回调函数

USER CODE BEGIN 4 和USER CODE END 4



电子科技大学
University of Electronic Science and Technology of China

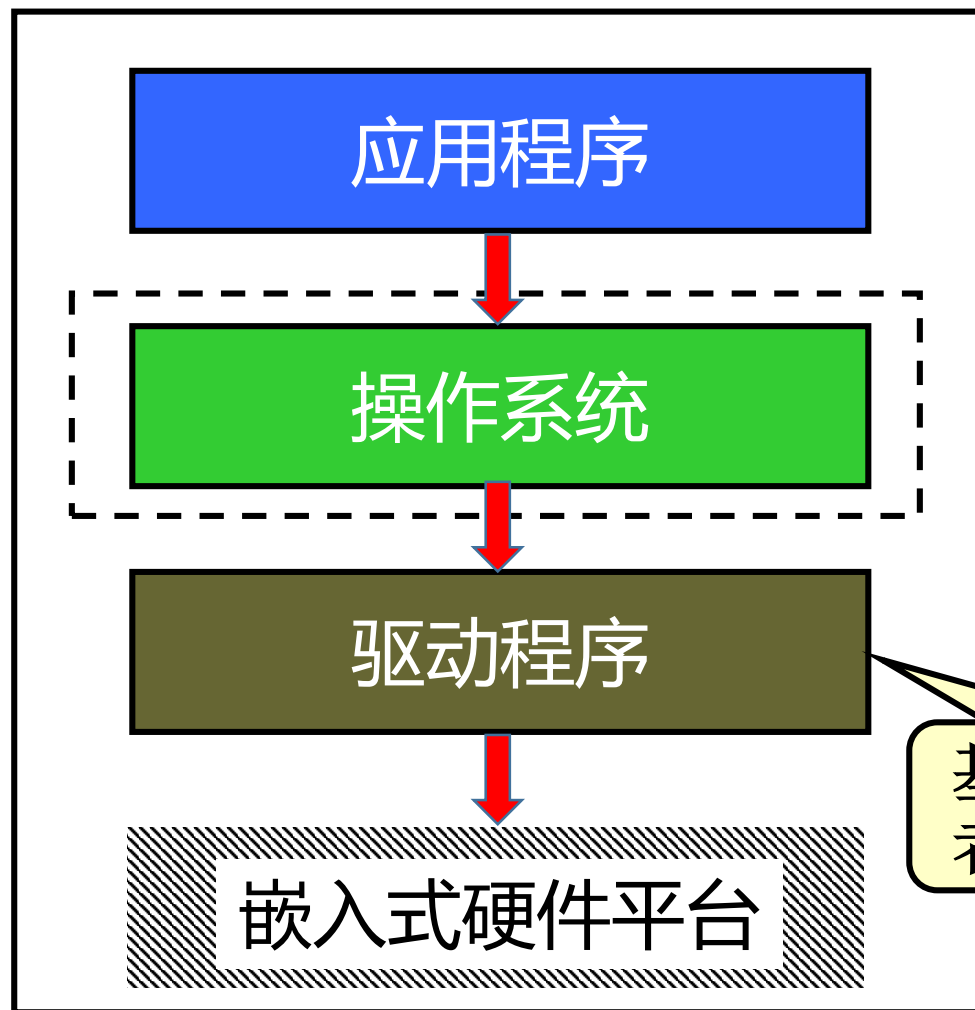
6.7 硬件抽象层设计



电子科技大学
University of Electronic Science and Technology of China

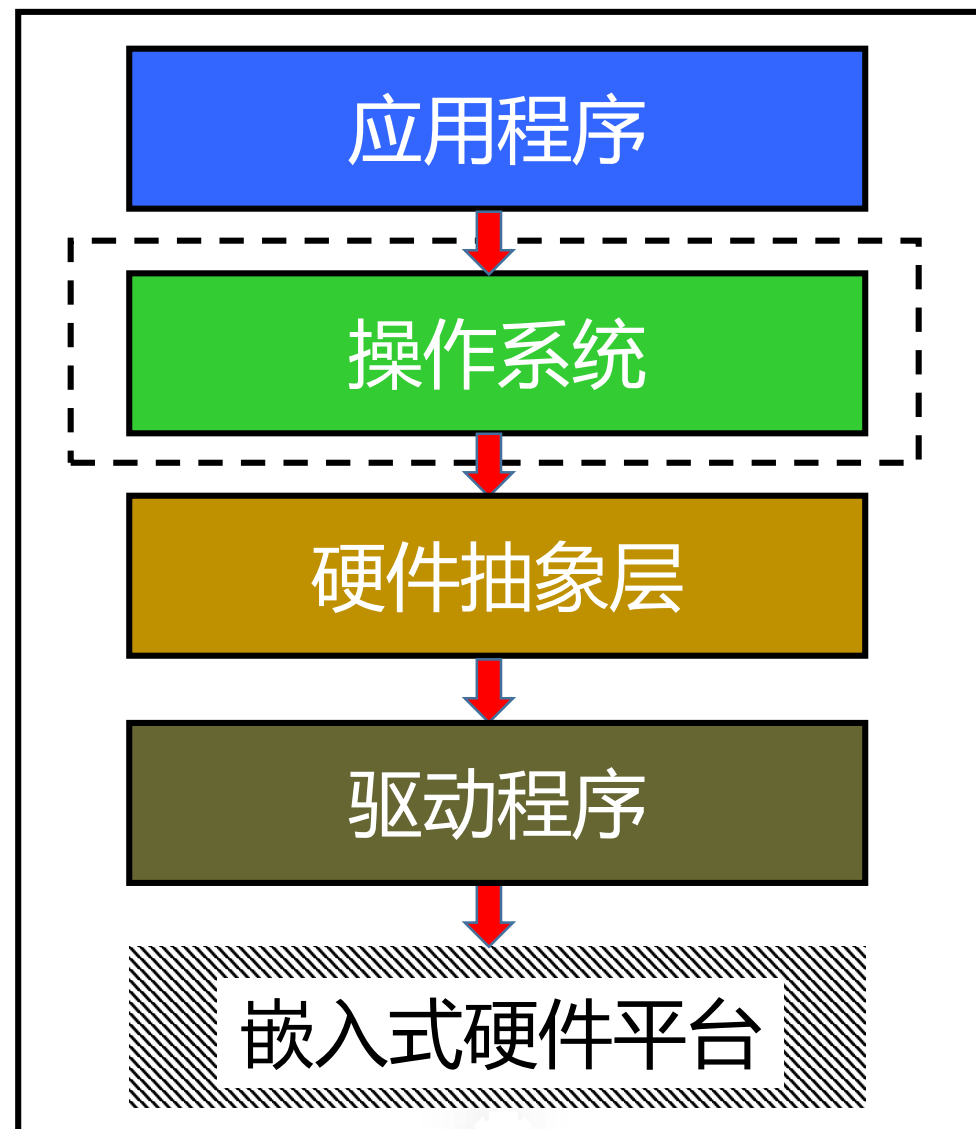
1 硬件抽象层设计思想

软件框架演变



加入HAL层

基于寄存器或者库函数实现



硬件抽象层设计思想

- 一：采用通用性设计思想，屏蔽底层硬件差异，简化程序移植过程
- 二：采用层次化设计思想，硬件抽象层调用驱动层，应用层则调用硬件抽象层
- 三：采用模块化设计思想，以外围电路模块为单元进行设计

BSP

板级支持包BSP

应用层

调用BSP提供的
接口函数API

操作系统

调用BSP提供的
接口函数API

板级支持包

BSP_LED_Init
BSP_LED_On
BSP_LED_Off

驱动程序

寄存器或固件库

硬件平台
8051/MSP430/STM32



电子科技大学
University of Electronic Science and Technology of China

2 硬件抽象层设计实现

设计方法

以外围电路模块为单元设计，分为头文件和源文件

头文件

宏定义、枚举类型、结构体类型等数据类型的定义，以及提供给外部调用的接口函数的声明。

源文件

变量的定义、硬件引脚的定义以及接口函数的具体实现。

指示灯模块的硬件抽象层设计

1

指示灯模块分成BSP_LED.h和BSP_LED.C两个文件

2

设计时假设有4个指示灯LED0~LED3，均采用高电平驱动方式，对应的控制引脚依次为PA0、PB0、PC0和PD0

指示灯模块的 BSP_LED.h 文件

文件的基本描述

```
1.  /*****
2.  * 模块名称 : 指示灯驱动模块
3.  * 文件名   : BSP_LED.h
4.  * 说明     : 头文件, 提供自定义数据类型及外部调用的接口函数的声明
5.  * 版本号 : V2.0
6.  * 修改记录 :
7.  *          版本号          日期          作者
8.  *          V2.0           2018-05-02      qiqiang
9.  *****/
10. #ifndef __BSP_LED_H
11. #define __BSP_LED_H
```

避免头文件的重复包含

// 避免重复包含

```
12. /*****
```

```
13.
```

用户自定义数据类型

```
14. *****/
```

```
15. typedef enum
```

```
16. {
```

```
17.     LED0 = 0,
```

```
18.     LED1 = 1,
```

```
19.     LED2 = 2,
```

```
20.     LED3 = 3,
```

```
21. }LED_INDEX;
```

指示灯模块的 BSP_LED.h 文件

定义指示灯序号的数据类型，使用枚举类型实现，包括LED0~LED3四个枚举常量，用来表示四个指示灯

```
22. /*****
```

```
23.
```

本模块提供给外部调用的函数

提供给外部使用函数的声明

```
24. *****/
```

```
25. extern void BSP_LED_Init    ( LED_INDEX Led ); // 指示灯初始化
```

```
26. extern void BSP_LED_On     ( LED_INDEX Led ); // 开启指示灯
```

```
27. extern void BSP_LED_Off    ( LED_INDEX Led ); // 关闭指示灯
```

```
28. extern void BSP_LED_Toggle ( LED_INDEX Led ); // 翻转指示灯状态
```

```
29. #endif
```

```
30. /***** (END OF FILE) *****/
```

指示灯模块的 BSP_LED.c 文件

文件的基本描述

```
1.  /*****
2.  * 模块名称：指示灯驱动模块
3.  * 文件名   ： BSP_Led.c
4.  * 说明     ： 驱动LED指示灯
5.  * 版本号  ： V2.0
6.  * 修改记录： 1、将LED驱动采用结构体指针数组的形式改写，移植时只需要修改LED
7.                的数量，引脚定义以及存放端口和引脚的两个数组。      2018/05/02
8.  *
9.  *                版本号                日期                作者
10. *                V2.0                2018-05-02        qiqiang
10. *****/
```

11. /*****

12. **模块移植性说明**

13. *****/

模块的移植方法

14. /*

15. 1、如果用户的LED指示灯个数小于等于4个，可以将多余指示灯的引脚定义全部定义为
16. 和第1个指示灯一样，其余代码不做任何修改

17. 2、如果用户的LED指示灯个数大于4个，需要修改如下内容：

18. (1) 增加.h文件中LED_INDEX类型的枚举常量

19. (2) 修改指示灯数量定义LEDn

20. (3) 修改指示灯控制引脚的定义

21. (4) 修改端口和引脚两个数组：GPIO_PORT[LEDn]和GPIO_PIN[LEDn]

22. (5) 修改引脚的时钟控制宏定义LEDx_GPIO_CLK_ENABLE(__INDEX__)的内容

23. */

```

24. /*******
25.
26. *****/
27. #include "stm32f4xx_hal.h"
28. #include "BSP_LED.h"
29. /*******
30.
31. *****/
32. #define LEDn                4                // 指示灯数量
33. // 指示灯0
34. #define LED0_PIN            GPIO_PIN_0        // PA0
35. #define LED0_GPIO_PORT      GPIOA
36. #define LED0_GPIO_CLK_ENABLE()  __HAL_RCC_GPIOA_CLK_ENABLE()
37. // 指示灯1
38. #define LED1_PIN            GPIO_PIN_0        // PB0
39. #define LED1_GPIO_PORT      GPIOB
40. #define LED1_GPIO_CLK_ENABLE()  __HAL_RCC_GPIOB_CLK_ENABLE()

```

添加HAL库头文件，以便使用HAL库提供的数据类型和接口函数

添加本模块自身的头文件，以便使用指示灯序号LED_INDEX这个数据类型

指示灯硬件接口定义

指示灯控制引脚的宏定义

- 引脚号
- 端口号
- 端口时钟使能

41. // 指示灯2

42. #define LED2_PIN GPIO_PIN_0 // PC0

43. #define LED2_GPIO_PORT GPIOC

44. #define LED2_GPIO_CLK_ENABLE() __HAL_RCC_GPIOC_CLK_ENABLE()

45. // 指示灯3

46. #define LED3_PIN GPIO_PIN_0 // PD0

47. #define LED3_GPIO_PORT GPIOD

48. #define LED3_GPIO_CLK_ENABLE() __HAL_RCC_GPIOD_CLK_ENABLE()

49. // 端口数组, 数组中存放的都是指示灯控制引脚所属端口

50. GPIO_TypeDef *GPIO_PORT[LEDn] = {

51. LED0_GPIO_PORT,

52. LED1_GPIO_PORT,

53. LED2_GPIO_PORT,

54. LED3_GPIO_PORT,

55. };

数据类型为指向端口寄存器的结构体指针
GPIO_TypeDef

56. // 引脚数组, 数组中存放的都是指示灯控制引脚

```
57 const uint16_t GPIO_PIN[LEDn] = {  
58     LED0_PIN,  
59     LED1_PIN,  
60     LED2_PIN,  
61     LED3_PIN,  
62     };
```

63. // 端口时钟使能宏定义

```
64. #define LEDx_GPIO_CLK_ENABLE(__INDEX__) do{ \  
65.     if((__INDEX__) == 0) \  
66.         LED0_GPIO_CLK_ENABLE(); \  
67.     else if((__INDEX__) == 1) \  
68.         LED1_GPIO_CLK_ENABLE(); \  
69.     else if((__INDEX__) == 2) \  
70.         LED2_GPIO_CLK_ENABLE(); \  
71.     else if((__INDEX__) == 3) \  
72.         LED3_GPIO_CLK_ENABLE(); \  
73. } while(0)
```

使用do while(0)的形式是
为了避免宏展开时发生错误

1 接口函数之指示灯引脚初始化: BSP_LED_Init

```
74. /*****
75.
76. *****/
77. /*****
78.  * @name      BSP_LED_Init
79.  * @brief     Configures LED GPIO.
80.  * @param[in] Led: Specifies the Led to be configured.
81.  *            This parameter can be one of following parameters:
82.  * @arg       LED0~LED3
83.  * @return    None
84.  * @note
85. *****/
```

本模块提供外部调用的函数

函数的基本描述

- 函数名称
- 函数功能
- 入口参数
- 返回值
- 使用注意

```
86. void BSP_LED_Init(LED_INDEX Led)
```

```
87. {
```

```
88.     GPIO_InitTypeDef GPIO_InitStructure = { 0 };
```

```
89.     /* Enable the GPIO_LED Clock */
```

```
90.     LEDx_GPIO_CLK_ENABLE(Led);
```

```
91.     /* Configure the GPIO_LED pin */
```

```
92.     GPIO_InitStructure.Pin      = GPIO_PIN[Led];
```

```
93.     GPIO_InitStructure.Mode     = GPIO_MODE_OUTPUT_PP;
```

```
94.     GPIO_InitStructure.Pull     = GPIO_NOPULL;
```

```
95.     GPIO_InitStructure.Speed    = GPIO_SPEED_FREQ_LOW;
```

```
96.     HAL_GPIO_Init(GPIO_PORT[Led], &GPIO_InitStructure);
```

```
97.     HAL_GPIO_WritePin(GPIO_PORT[Led], GPIO_PIN[Led], GPIO_PIN_RESET);
```

```
98. }
```

定义引脚初始化变量，并赋初值为0

使能引脚所属端口的时钟

配置引脚参数并初始化引脚

设置引脚初始电平

2 接口函数之指示灯开启: BSP_LED_On

```
99. /*****
100.  * @name      BSP_LED_On
101.  * @brief     Turns selected LED On
102.  * @param[in] Led: Specifies the Led to be set on.
103.  *           This parameter can be one of following parameters:
104.  * @arg       LED0~LED3
105.  * @return    None
106.  * @note
107.  *****/
108. void BSP_LED_On(LED_INDEX Led)
109. {
110.     HAL_GPIO_WritePin(GPIO_PORT[Led], GPIO_PIN[Led], GPIO_PIN_SET);
111. }
```

调用接口函数
HAL_GPIO_WritePin

3 接口函数之指示灯关闭: BSP_LED_Off

```
112.  /*******
113.   * @name      BSP_LED_Off
114.   * @brief     Turns selected LED Off
115.   * @param[in] Led: Specifies the Led to be set on.
116.   *           This parameter can be one of following parameters:
117.   * @arg       LED0~LED3
118.   * @return    None
119.   * @note
120.  *****/
121.  void BSP_LED_Off(LED_INDEX Led)
122.  {
123.      HAL_GPIO_WritePin(GPIO_PORT[Led], GPIO_PIN[Led], GPIO_PIN_RESET);
124.  }
```

调用接口函数
HAL_GPIO_WritePin

4 接口函数之指示灯状态翻转: BSP_LED_Toggle

```
125.  /*******
126.   * @name      BSP_LED_Toggle
127.   * @brief     Toggles the selected LED.
128.   * @param[in] Led: Specifies the Led to be toggled.
129.   *            This parameter can be one of following parameters:
130.   * @arg       LED0~LED3
131.   * @return     None
132.   * @note
133.   *****/
134. void BSP_LED_Toggle(LED_INDEX Led)
135. {
136.     HAL_GPIO_TogglePin(GPIO_PORT[Led], GPIO_PIN[Led]);
137. }
```

调用接口函数
HAL_GPIO_TogglePin

设计总结

硬件抽象层设计总结

头文件

- 只包括数据类型的定义以及提供给外部调用的接口函数的说明
- 一般不进行变量的定义以及硬件引脚的说明，也不包含任何其他的头文件，以确保头文件的通用性
- 应用层只调用头文件提供的接口函数，因此不需要修改应用层的代码

源文件

- 完成接口函数的实现
- 包含相关的头文件
- 进行模块内的变量定义以及与硬件相关的全部定义
- 调用HAL库提供的接口函数，进行二次封装，提供一个可读性更强，移植性更好的模块接口函数



电子科技大学
University of Electronic Science and Technology of China

3 硬件抽象层移植步骤

移植步骤

BSP移植步骤



1 修改BSP

修改BSP的两种情况

指示灯数量小于等于4

- ① 修改指示灯控制引脚的定义
- ② 将多余指示灯的控制引脚的定义全部修改为与第一个指示灯一样

指示灯数量大于4

- ① 增加LED_INDEX类型的枚举常量
- ② 修改指示灯数量
- ③ 修改指示灯控制引脚的定义
- ④ 修改与端口和引脚相关的两个数组
- ⑤ 修改与引脚相关的时钟控制宏定义

移植到Nucleo开发板

1

// LED接口的硬件定义

// 指示灯0

```
#define LED0_PIN          GPIO_PIN_5      // PA5
#define LED0_GPIO_PORT    GPIOA
#define LED0_GPIO_CLK_ENABLE() __HAL_RCC_GPIOA_CLK_ENABLE()
```

修改指示灯控制引脚为PA5

// 指示灯1

```
#define LED1_PIN          GPIO_PIN_5      // PA5
#define LED1_GPIO_PORT    GPIOA
#define LED1_GPIO_CLK_ENABLE() __HAL_RCC_GPIOA_CLK_ENABLE()
```

2

// 指示灯2

```
#define LED2_PIN          GPIO_PIN_5      // PA5
#define LED2_GPIO_PORT    GPIOA
#define LED2_GPIO_CLK_ENABLE() __HAL_RCC_GPIOA_CLK_ENABLE()
```

将多余的指示灯控制引脚定义修改为PA5

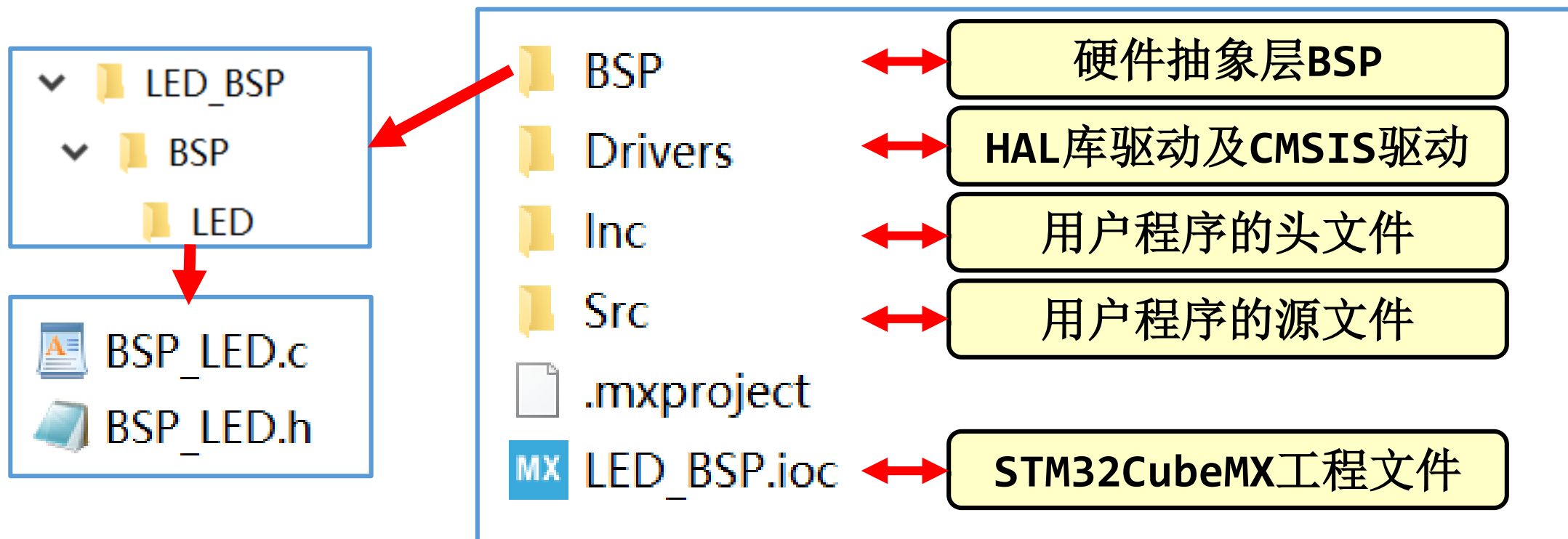
// 指示灯3

```
#define LED3_PIN          GPIO_PIN_5      // PA5
#define LED3_GPIO_PORT    GPIOA
#define LED3_GPIO_CLK_ENABLE() __HAL_RCC_GPIOA_CLK_ENABLE()
```

2 添加BSP

第一步：复制BSP文件

利用CubeMX新建工程或复制已有工程，在工程文件夹中新建BSP文件夹及LED文件夹，并将BSP_LED.c和BSP_LED.h复制到该文件夹



第二步：添加BSP文件到MDK工程

工程文件
管理图标

1

File Extensions, Books and Environment...
Manage Project Items

3

添加的BSP组
及组内文件

BSP_LED.c
BSP_LED.h

2

Groups:

Application/MDK-ARM
Application/User
Drivers/STM32F4xx_HAL_Driver
Drivers/CMSIS
BSP

点击新建组图标，新建
一个名为“BSP”的组

选中BSP组，利用
“Add Files”按
钮找到BSP\LED下
面的BSP_LED.c
和 BSP_LED.h文
件，添加到BSP组

Add Files...

第三步：添加头文件包含路径

1

工程设置图标

Options for Target...

Configure target options

2

Options for Target 'LED_BSP'

Device | Target | Output | Listing | User

C/C++

C/C++标签卡

Utilities

3

Include
Paths

../Inc;../Drivers/STM32F4xx_HAL_Driver/Inc;../Drivers/STM32F4xx_H

添加头文件路径

...

Folder Setup

Setup Compiler Include Paths:

新建头文件路径图标

BSP_LED.h所在路径

4

../Inc
../Drivers/STM32F4xx_HAL_Driver/Inc
../Drivers/STM32F4xx_HAL_Driver/Inc/Legacy
../Drivers/CMSIS/Device/ST/STM32F4xx/Include
../Drivers/CMSIS/Include

..\BSP\LED

3 编写程序

添加头文件

main.h

```
/* Private includes ----- */
```

```
/* USER CODE BEGIN Includes */
```

```
#include "BSP_LED.h" // 包含LED模块的头文件
```

```
/* USER CODE END Includes */
```


修改程序

添加初始化代码

main.c

```
/* Initialize all configured peripherals */
```

```
//MX_GPIO_Init(); // CubeMX软件生成的指示灯控制引脚初始化程序
```

```
/* USER CODE BEGIN 2 */
```

```
BSP_LED_Init(LED0); // 调用指示灯模块的初始化接口函数
```

```
/* USER CODE END 2 */
```

利用接口函数控制指示灯

main.c

```
1.  /* Infinite loop */
2.  /* USER CODE BEGIN WHILE */
3.  while (1)
4.  {
5.      /* USER CODE END WHILE */
6.      /* USER CODE BEGIN 3 */
7.      // 调用LED模块的接口函数控制指示灯
8.      BSP_LED_On(LED0);    // 开启LD2
9.      HAL_Delay(1000);     // 延时1000ms
10.     BSP_LED_Off(LED0);   // 关闭LD2
11.     HAL_Delay(1000);     // 延时1000ms
12. }
13. /* USER CODE END 3 */
```

