第一次作业

作业一

重新设计硬件，把LED接在PORTC-13口口，代码如何修改？

使用寄存器编程，在datasheet中找到使能端地址、配置输出端地址、寄存器地址等，然后分别赋值。

```
int main (void)
{

  // 打开 GPIOC 端口的时钟
  *( unsigned int * )0x40021018 |=  ( (1) << 4 );

  // 配置IO口PB5为输出 ,推挽输出，速率为10M
  *( unsigned int * )0x40011004 &= 0xFF0FFFFF;
  *( unsigned int * )0x40011004 |=  0X00100000;
  // 控制 ODR 寄存器
  *( unsigned int * )0x4001100C &= ~(1<<13); //LED on
  //*( unsigned int * )0x4001100C |= (1<<13);  //LED off

}


void SystemInit(void)
{
  // 函数体为空，目的是为了骗过编译器不报错
}
```

作业二

Configure PC13 as a pull-up input pin by 2 method refer to the example above，
1：Firmware library programming
2：Register Programming

第一种：固件库编程，如下：

```
#include "stm32f10x.h"                  // Device header

int main(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
```

```
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
  GPIO_Init(GPIOC,&GPIO_InitStructure);
}
```

固件库实际上是把寄存器地址等信息用更加人性化的方式包装了一下，用户可以直接调用包装好的函数和地址来进行配置。具体方式可以查找头文件里面的定义，然后在keil的自动补全代码中完成编程。

第二种，寄存器编程，与作业一相同：

```
int main (void)
{

  // 打开 GPIOC 端口的时钟
  *( unsigned int * )0x40021018 |=  ( (1) << 4 );

  // 配置IO口PB5为输出 ,推挽输出，速率为10M
  *( unsigned int * )0x40011004 &= 0xFF0FFFFF;
  *( unsigned int * )0x40011004 |=  0X00100000;
  // 控制 ODR 寄存器
  *( unsigned int * )0x4001100C &= ~(1<<13); //LED on
  //*( unsigned int * )0x4001100C |= (1<<13);  //LED off

}


void SystemInit(void)
{
  // 函数体为空，目的是为了骗过编译器不报错
}
```

作业三

第三次作业即用自己配置的keil工程文件完成作业二，代码与作业二相同。

```
#include "stm32f10x.h"                  // Device header

int main(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_10MHz;
```

```
  GPIO_Init(GPIOC,&GPIO_InitStructure);
}
```

在完成作业的时候遇到了一个问题，即legacy package缺失，解决方案是去报错给的网址下载legacy package然后安装就结束了。

课上笔记

课上讲述的内容部分我已经提前了解，笔记把课上不是很了解的部分记录了下来。

stm32有时钟树，所以默认所有外设时钟全部关闭（省电）

操作步骤：因此需要使用某个外设，首先需要开启（使能）时钟，然后配置模式（输入输出），最后再设置具体的开灯关灯

datasheet用法：找各种地址

3.3memorymap找基地址，然后去找对应的registor寄存器

基地址（6位）+偏移地址（2位）

寄存器本质是触发器，因此一定需要时钟。

关于使用的到底是哪个时钟：查看该器件挂载在哪一个总线上：

挂载是指将一个文件系统连接到计算机的目录树中，使得这个文件系统中的文件和目录可以在这个目录树上访问。

三个BUS总线，AHB,APB1,APB2

AHB总线、APB1总线和APB2总线分别对应不同的外设，如下：

::: info
AHB总线：CPU、DMA、SRAM、FLASH、RCC等

APB1总线：TIM2、TIM3、TIM4、TIM5、TIM6、TIM7、WWDG、I2C1、I2C2、SPI2、SPI3、USART2、USART3、USART4、USART5、USART6、CAN1、CAN2、BKPSRAM、PWR、DAC

APB2总线：USART1、USART6、ADC1、ADC2、ADC3、SPI1、TIM1、TIM8、USART1、USART6、ADC1、ADC2、ADC3、SPI1、TIM1、TIM8、USART1、USART6、ADC1、ADC2、ADC3、SPI1、TIM1、TIM8、GPIOA、GPIOB、GPIOC、

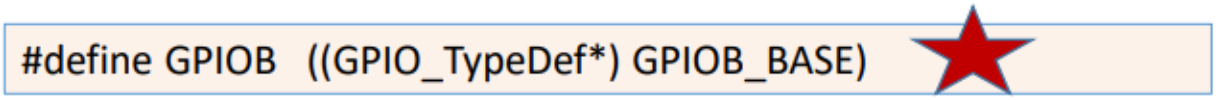
GPIOD、GPIOE、GPIOF、GPIOG、AFIO、EXTI、ADC1、ADC2、ADC3、TIM8、SPI1

因此，需要使用某个外设时，需要先开启该外设所对应的总线的时钟，然后再设置具体的开关状态。

CRL and CRH are control registers for GPIO pins. They are used to configure the mode (input, output, alternate function) and the speed of the GPIO pins. CRL is used for pins 0-7, while CRH is used for pins 8-15.

库函数（拥有大量宏定义的寄存器）

GPIO的定义大量重复，因此定义结构体



把GPIOB的基地址强制转换为一个GPIO_TypeDef结构体。

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

Register	Start Address is Base
CRL	Base +0x00
CRH	Base +0x04
IDR	Base +0x08
ODR	Base +0x0C
BSRR	Base +0x10
BRR	Base +0x14
CLKR	Base +0x18

main.c

```
// 配置IO口为输出
GPIOB->CRL &=  0xFF0FFFFF;
GPIOB->CRL |=  0x00100000;
// 控制 ODR 寄存器
GPIOB->ODR &= ~(1<<5);
//GPIOB->ODR |= (1<<5);
```

相当于定义好初始地址，之后调用该GPIO的底下的内容直接→来获得

为防止出错，库函数写作者定义了枚举类型，需要的类型只能使用枚举类里的。

```
typedef enum
{
  GPIO_Speed_10MHz = 1,       // 10MHZ      (01)b
  GPIO_Speed_2MHz,           // 2MHZ       (10)b
  GPIO_Speed_50MHz            // 50MHZ      (11)b
}GPIOSpeed_TypeDef;

typedef enum
{ GPIO_Mode_AIN = 0x0,         // 模拟输入    (0000 0000)b
  GPIO_Mode_IN_FLOATING = 0x04, // 浮空输入    (0000 0100)b
  GPIO_Mode_IPD = 0x28,        // 下拉输入    (0010 1000)b
  GPIO_Mode_IPU = 0x48,        // 上拉输入    (0100 1000)b

  GPIO_Mode_Out_OD = 0x14,     // 开漏输出    (0001 0100)b
  GPIO_Mode_Out_PP = 0x10,     // 推挽输出    (0001 0000)b
  GPIO_Mode_AF_OD = 0x1C,      // 复用开漏输出 (0001 1100)b
  GPIO_Mode_AF_PP = 0x18       // 复用推挽输出 (0001 1000)b
}GPIOMode_TypeDef;
```

```
1408 #define GPIOA                ((GPIO_TypeDef *) GPIOA_BASE)
1409 #define GPIOB                ((GPIO_TypeDef *) GPIOB_BASE)
1410 #define GPIOC                ((GPIO_TypeDef *) GPIOC_BASE)
1411 #define GPIOD                ((GPIO_TypeDef *) GPIOD_BASE)
1412 #define GPIOE                ((GPIO_TypeDef *) GPIOE_BASE)
1413 #define GPIOF                ((GPIO_TypeDef *) GPIOF_BASE)
1414 #define GPIOG                ((GPIO_TypeDef *) GPIOG_BASE)
```

去哪找库函数：**.h文件**