详解实验三的两个函数

//created by zhangyunjie

//2024.9.30

第一个函数：读取引脚的电平状态

GPIO\_PinState HAL\_GPIO\_ReadPin(GPIO\_TypeDef\* GPIOx, uint16\_t GPIO\_Pin)  
{ //GPIOx是一个结构体，包含有各种寄存器的状态；GPIO\_Pin是一个数字  
 GPIO\_PinState bitstatus; //定义变量——位状态  
  
 */\* Check the parameters \*/* assert\_param(IS\_GPIO\_PIN(GPIO\_Pin)); //检查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;  
}

概述

typedef enum  
{  
 GPIO\_PIN\_RESET = 0,  
 GPIO\_PIN\_SET  
}GPIO\_PinState;

·这个函数的有两个形参：GPIOx和GPIO\_Pin

GPIOx：是一个指向结构体的指针表示所选择的端口；

GPIO\_Pin：表示所选择的引脚；

·函数返回值是这个引脚的GPIO状态——GPIO\_PinState

·PinState是一个枚举型，包含了引脚RESET和SET两个状态

·这个函数做实现的功能就是读取x端口的指定引脚的状态

第一行——定义变量

·定义了一个位状态（enum）变量

第二行——检测输入合法性

assert\_param(IS\_GPIO\_PIN(GPIO\_Pin));

·为两层嵌套

·经查看可知，内层为一个宏定义的函数

#define GPIO\_PIN\_MASK 0x0000FFFFU

#define IS\_GPIO\_PIN(PIN) (((((uint32\_t)PIN) & GPIO\_PIN\_MASK ) != 0x00U) && ((((uint32\_t)PIN) & ~GPIO\_PIN\_MASK) == 0x00U))

通过位运算，GPIO\_Pin上有且只有低四位有数字时return 1；否则return 0；

·外层为一个宏定义的空函数，不会有任何效果

#define assert\_param(expr) ((void)0U)

Q:这个外层函数有什么意义？

第三行——判断+赋值

·查看GPIOx可知，GPIOx->IDR表示的是input data register——输入数据寄存器

if((GPIOx->IDR & GPIO\_Pin) != (uint32\_t)GPIO\_PIN\_RESET)  
{  
 bitstatus = GPIO\_PIN\_SET;  
}  
else  
{  
 bitstatus = GPIO\_PIN\_RESET;  
}

它存储着这个端口所有引脚的输入状态数据

typedef struct  
{

*…*\_\_IO uint32\_t IDR; */\*!< GPIO port input data register, Address offset: 0x10\*/* …

} GPIO\_TypeDef;

·if的判断可以理解为：通过位操作使GPIOx->IDR & GPIO\_Pin表示此引脚的输入状态

将其与GPIO\_PIN\_RESET（0）比较，结果为真赋值变量为SET（1），否则赋值为RESET（0）

·最后，return bitstatus；获得返回值，即x端口的指定引脚状态。

第二个函数：翻转引脚的输出状态

void HAL\_GPIO\_TogglePin(GPIO\_TypeDef\* GPIOx, uint16\_t GPIO\_Pin)  
{  
 uint32\_t odr;  
  
 */\* Check the parameters \*/* assert\_param(IS\_GPIO\_PIN(GPIO\_Pin));  
  
 */\* get current Output Data Register value \*/* odr = GPIOx->ODR;  
  
 */\* Set selected pins that were at low level, and reset ones that were high \*/* GPIOx->BSRR = ((odr & GPIO\_Pin) << GPIO\_NUMBER) | (~odr & GPIO\_Pin);  
}

概述

·这个函数的有两个形参：GPIOx和GPIO\_Pin

GPIOx：表示所选择的端口；GPIO\_Pin表示所选择的引脚

·函数返回值为空，该函数只进行操作，不进行返回

·这个函数做实现的功能就是反转x端口的指定引脚的输出状态

第一行——定义变量

·定义了odr（数）—— ODR表示输出数据寄存器

第二行——检验合法性

·与第一个函数类似，不赘述

第三行——获取寄存器

·获取此端口的输出数据寄存器状态，将其存入变量odr

第四行——进行翻转

GPIOx->BSRR = ((odr & GPIO\_Pin) << GPIO\_NUMBER) | (~odr & GPIO\_Pin);

·odr & GPIO\_Pin获取当前引脚（GPIO\_Pin）的状态，<<GN左移16位（移动到高十六位）

·因此给GPIOx->BSRR的数据高十六位为此引脚状态、低十六位为此引脚状态取反

·RSRR表示bit set/reset register —— 位设置/重设寄存器。

·高十六位为BR（bit reset）（若相应位为set（1）、左移后对应到了寄存器reset位置操作）

低十六位为BS（bit set）（若相应位为set（1），去反后为reset（0），对应寄存器set位置不操作）

·因此，将右侧的值赋值给GPIOx->BSRR时，即完成了对寄存器的操作，也就控制了电平的翻转

总的来说，两个函数的本质是对机器码进行运算，实现对寄存器的控制，从而实现指定功能