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

- 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

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

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

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

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