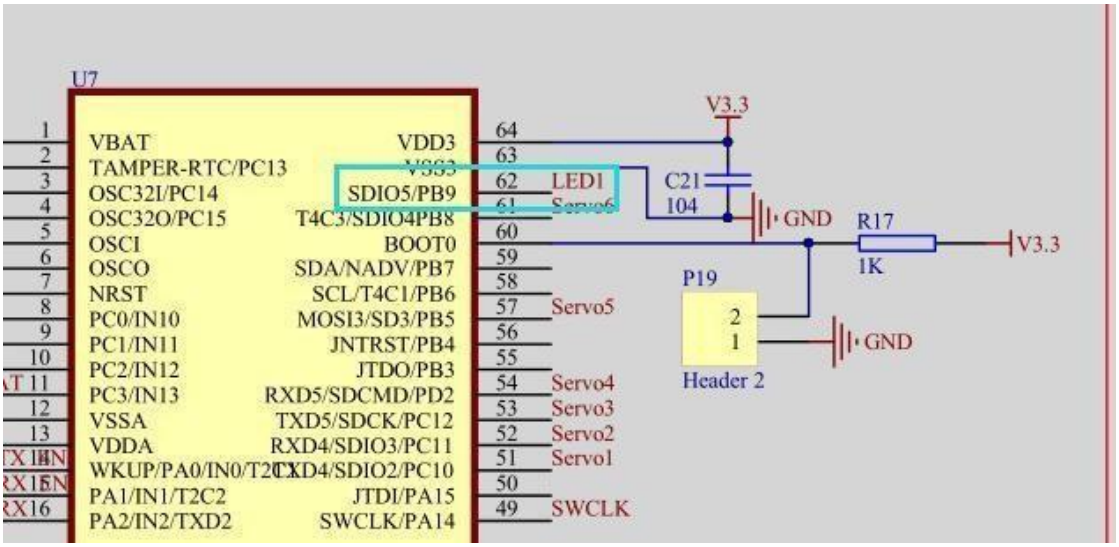
Lesson 1 Timer Controls LED

1. Project Purpose

Use the timer on the controller to control the on and off of the LED light.

2. Project Principle

If want to light up LED light, we need to know how the LED light is connected to the STM32. Through checking the schematic diagram, we know that LED is connected to the PB9 IO port of STM32, and lights up when it is low level.



From the above information, we can know that the process of lighting up LED is:

1) Configure I/O port parameters to make I/O Push–pull output or Open-drain output.

2) Set the I/O port to the low-level, which LED will be lighted up.

3. Program Analyst

```
void InitLED(void)
{

  GPIO_InitTypeDef  GPIO_InitStructure;

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);  //Enable PB port clock
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;          //LED2 PB9 port configuration
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;      // Push-pull output
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOB, &GPIO_InitStructure);  //  Configure IO

}
```

1) Execute the LED I/O port within configured in the InitLED function in the main function. Now, we can write 0 or 1 to the I/O port, and then control the LED light to turn on or off.

```
#define LED_ON    0
#define LED_OFF   1
```

```
LED = LED_ON;  //On
```

2) The above figure shows the LED light is turned on, where LED_ON is a macro definition corresponding to 0. If we track LED, we will find that LED is also a macro definition, as shown in the second figure below.

```
#define LED      PBout(9)
```

```
#define BITBAND(addr, bitnum) ((addr & 0xF0000000)+0x2000000+((addr &0xFFFFF)<<5)+(bitnum<<2))
#define MEM_ADDR(addr)  *((volatile unsigned long  *)(addr))
#define BIT_ADDR(addr, bitnum)   MEM_ADDR(BITBAND(addr, bitnum))
// IO port address mapping
#define GPIOA_ODR_Addr    (GPIOA_BASE+12) //0x4001080C
#define GPIOB_ODR_Addr    (GPIOB_BASE+12) //0x40010C0C
#define GPIOC_ODR_Addr    (GPIOC_BASE+12) //0x4001100C
#define GPIOD_ODR_Addr    (GPIOD_BASE+12) //0x4001140C
#define GPIOE_ODR_Addr    (GPIOE_BASE+12) //0x4001180C
#define GPIOF_ODR_Addr    (GPIOF_BASE+12) //0x40011A0C
#define GPIOG_ODR_Addr    (GPIOG_BASE+12) //0x40011E0C

#define GPIOA_IDR_Addr    (GPIOA_BASE+8) //0x40010808
#define GPIOB_IDR_Addr    (GPIOB_BASE+8) //0x40010C08
#define GPIOC_IDR_Addr    (GPIOC_BASE+8) //0x40011008
#define GPIOD_IDR_Addr    (GPIOD_BASE+8) //0x40011408
#define GPIOE_IDR_Addr    (GPIOE_BASE+8) //0x40011808
#define GPIOF_IDR_Addr    (GPIOF_BASE+8) //0x40011A08
#define GPIOG_IDR_Addr    (GPIOG_BASE+8) //0x40011E08

// IO port operation, only for a single IO port
// Make sure that the value of n is less than 16!
#define PAout(n)    BIT_ADDR(GPIOA_ODR_Addr,n)   //Output
#define PAin(n)     BIT_ADDR(GPIOA_IDR_Addr,n)   //Input

#define PBout(n)    BIT_ADDR(GPIOB_ODR_Addr,n)   //Output
```

3) We can find these codes in include.h and find that the LED eventually is parsed as:

LED becomes PBout(9)

PBout(9) becomes BIT_ADDR(GPIOB_ODR_Addr, 9)

BIT_ADDR(GPIOB_ODR_Addr, 9) becomes MEM_ADDR(BITBAND(GPIOB_ODR_Addr, 9)

MEM_ADDR(BITBAND(GPIOB_ODR_Addr, 9) becomes *((volatile unsigned long*)(BITBAND(GPIOB_ODR_Addr, 9)))

Eventually becomes to:

((volatile unsigned long)(((GPIOB_ODR_Addr & 0xF0000000)+0x2000000+((GPIOB_ODR_Addr & 0xFFFFF)<<5)+(9 <<2)))

The STM32 registers can not be operated in bits. To operate a specific I/O port, we must use a combination of shift and logic operations.

To facilitate the operation of the I/O port, the bit field function is provided in STM32, which can map the input and output of each I/O port to an independent address through each bit on register. Thus, you just need to write the address into data to operate the corresponding I/O port.

1) BITBAND converts the provided output register address and I/O port number into the corresponding operation address. Therefore, just change the address and write 0 or 1 to control the LED light on and off.