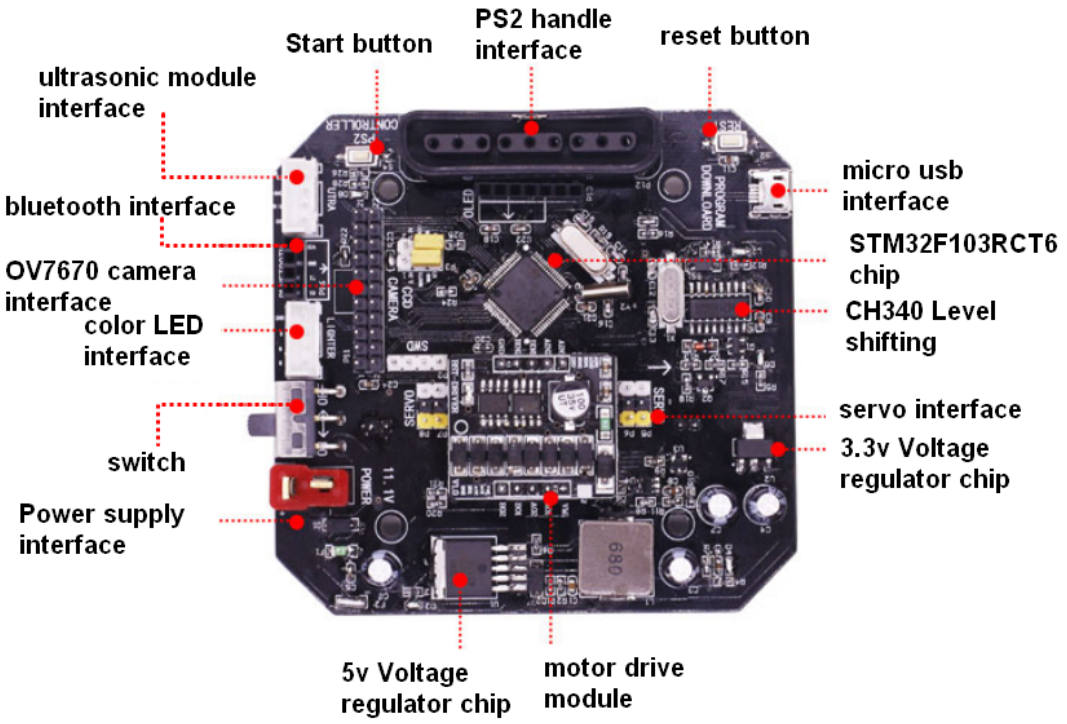
1. STM32 platform-------Light up LED

1) Preparation



1-1 STM32 expansion board

1-2 STM32 smart car

2) Purpose of Experimental

After the car is powered on, press the start button next to the PS2 logo on the expansion board, you will see the LED flashing on the expansion board, as shown in Figure 1-3. We will learn the STM32 GPIO port control.

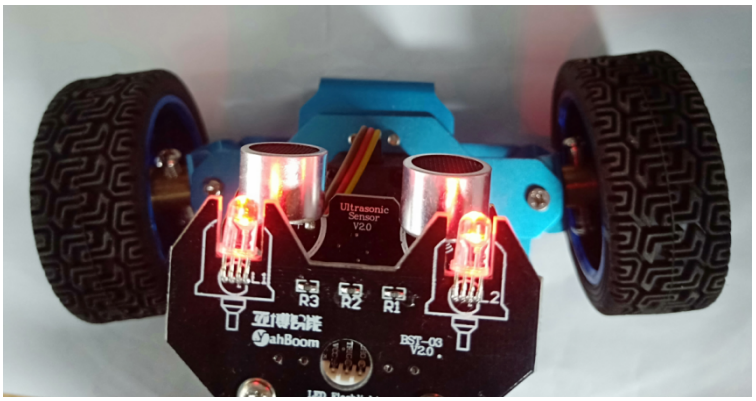


Figure 1-3(1)

Figure 1-3(2)

3) **Principle of experimental**

It can be seen from the circuit schematic diagram that the LED cathode used in this experiment is connected to the PA1 and pin of the STM32 main control board. At the same time, the LED lamp needs to be connected in series with a 10K resistor as the current limiting resistor. We only need to control the corresponding pin to a low level on the STM32 main control board to illuminate the corresponding LED.

Since the MCU pin is not enough, the LED pin and the colorful searchlight share a pin, so the colorful searchlight may also be on when the LED of the expansion board is lit, and the LED light may also be bright when controlling the colorful searchlight.

We will first explain the important registers before explaining the firmware library, this is for everyone to have a preliminary understanding of the registers. Everyone learns the firmware library, and does not need to remember the role of each register, but just understand the registers to have a general understanding of some functions of the peripherals, which is also very helpful for future learning.

First of all, in the firmware library, the corresponding library functions for GPIO

port operations and related definitions are in the files stm32f10x_gpio.h and stm32f10x_gpio.c.

The IO port of STM32 can be configured by software into the following 8 modes:
1. Input floating
2. Input pull-up
3. Input drop-down
4. Analog input
5. Open drain output
6. Push-pull output
7. Push-pull reuse function
8. Open-drain multiplexing function

Each IO port is freely programmable, but the IO port registers must be accessed as 32-bit words. Many IO ports of STM32 are 5V compatible. These IO ports possess advantages when connected to 5V level peripherals. You can check the data sheet pin description section of the chip to know Which IO ports are 5V compatible. (I/O Level FT is 5V level compatible).

Each IO port of the STM32 has 7 registers to control:

Two 32-bit port configuration registers CRL and CRH in configuration mode; CRL and CRH control the mode and output rate of each IO port.

Two 32-bit data registers IDR and ODR;

One 32-bit set/reset register BSRR;

One 16-bit reset register BRR;

One 32-bit latch register, LCKR.

If you want to know the detailed usage of each register, you can refer to the 《STM32 English Reference Manual V15》.

The STM32 IO port configuration table is shown in Table 3.1:

Configuration mode		CNF1	CNF0	MODE1	MODE0	PxODR
General purpose output	(Push-Pull)	0	0	01		0 or 1
	Open-Drain)		1	10		0 or 1
Multiplexed output function	(Push-Pull)	1	0	11		no use
	(Open-Drain)		1			no use
Input	Analog input	0	0	00		no use
	Floating input		1			no use
	Input drop-down	1	0			0
	Input pull-up					1

Table 3.1 STM32 IO port configuration table

The STM32 output mode configuration is shown in Table 3.2:

MODE[1:0]	significance
00	Reserved
01	The maximum output speed is 10Mhz
10	The maximum output speed is 2Mhz
11	The maximum output speed is 50Mhz

Table 3.2 Output mode configuration table

In firmware library development, the operation registers CRH and CRL to configure the mode and speed of the IO port are initialized by GPIO.
The function is completed:

```
void GPIO_Init(GPIO_TypeDef* GPIOx,   GPIO_InitTypeDef*
GPIO_InitStruct);
```

This function has two parameters:
The first parameter is used to specify GPIO, which ranges from GPIOA~GPIOG.
The second parameter is the initialization parameter structure pointer, and the structure type is GPIO_InitTypeDef.
Let's take a look at the definition of this structure. First open our experiment, then find the stm32f10x_gpio.c file under the FWLib group, locate the GPIO_Init function body, double-click the entry parameter type GPIO_InitTypeDef and right click and select "Go to definition of ..." to view the definition of the structure:

```
typedef struct
{
    uint16_t   GPIO_Pin;
    GPIOSpeed_TypeDef   GPIO_Speed;
    GPIOMode_TypeDef   GPIO_Mode;
}GPIO_InitTypeDef;
```

Below we use a GPIO initialization instance to explain the meaning of the member variables of this structure.
The common format for initializing GPIO by initializing the structure is:

```
GPIO_InitTypeDef   GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin   =   GPIO_Pin_1; //RED LED-->PB.1 port
configuration
GPIO_InitStructure.GPIO_Mode   =   GPIO_Mode_Out_PP; //Push-pull
output
GPIO_InitStructure.GPIO_Speed   =   GPIO_Speed_50MHz;//Speed 50MHz
```

GPIO_Init(GPIOB, &GPIO_InitStructure);//Configure GPIO according to the setting parameters

The above code means that the first port of the GPIOB is set to push-pull output mode and the speed is 50M. As can be seen from the above initialization code, the first member variable GPIO_Pin of the structure GPIO_InitStructure is used to set which IO port is to be initialized; the second member variable GPIO_Mode is used to set the output input mode of the corresponding IO port.
These patterns are the eight patterns we explained above, and are defined in MDK by an enumerated type:

```
typedef enum
{
    GPIO_Mode_AIN = 0x0, //Analog input
    GPIO_Mode_IN_FLOATING = 0x04, //Floating input
    GPIO_Mode_IPD = 0x28, //drop-down Input
    GPIO_Mode_IPU = 0x48, //pull-up Input
    GPIO_Mode_Out_OD = 0x14, //Open-drain output
    GPIO_Mode_Out_PP = 0x10, //Universal push-pull output
    GPIO_Mode_AF_OD = 0x1C, //Multiplex open drain output
    GPIO_Mode_AF_PP = 0x18 //Multiplex push-pull
}GPIOMode_TypeDef;
```

The third parameter is the IO port speed setting, which has three optional values, which are also defined by the enumeration type in MDK:

```
typedef enum
{
    GPIO_Speed_10MHz = 1,
    GPIO_Speed_2MHz,
    GPIO_Speed_50MHz
}GPIOSpeed_TypeDef;
```

To know the level of an IO port, you only need to read this register and look at the state of a bit.

Operating the IDR Register in the Firmware Library to read IO port data by the GPIO_ReadInputDataBit function:
uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)

For example, if I want to read the level status of GPIOB.1, then the method is:
GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_1);
The return value is 1 (Bit_SET) or 0 (Bit_RESET);

Set the value of the ODR register in the firmware library to control the output state of the IO port by the GPIO_Write function.
void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal);

The BRR register is the port bit clear register.

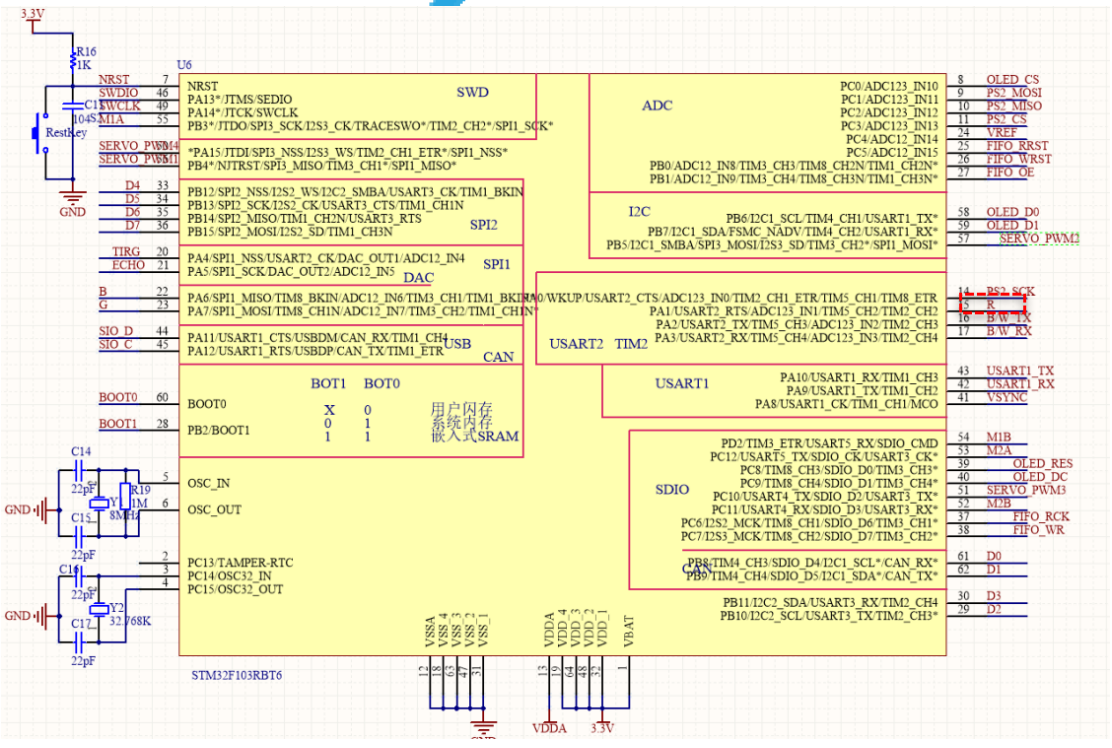
In the STM32 firmware library, the GPIO port output is set by the BSRR and BRR registers,which is done by GPIO_SetBits() and the function GPIO_ResetBits() function.
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);

In most cases, we use these two functions to set the input and output states of the GPIO port.
For example, if we want to set GPIOB.1 output 1, the method is:
GPIO_SetBits(GPIOB, GPIO_Pin_1);
Otherwise, if you want to set the GPIOB.1 output bit 0, the method is:
GPIO_ResetBits (GPIOB, GPIO_Pin_1);

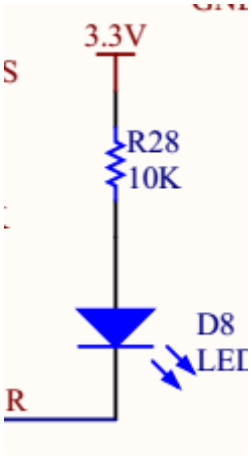
Step:
1) Eanble IO clock, call RCC_APB2PeriphClockCmd() function.
2) Initialize the IO parameters. Call the function GPIO_Init();
3）Operation IO.

4) Experimental Steps

4-1 About the schematic



4-1 STM32 main control board circuit diagram



4-2 LED interface

4-2 According to the circuit schematic:

R-----PA1(STM32)

4-3 About the code

Please see the folder named Light up LED in the code folder.