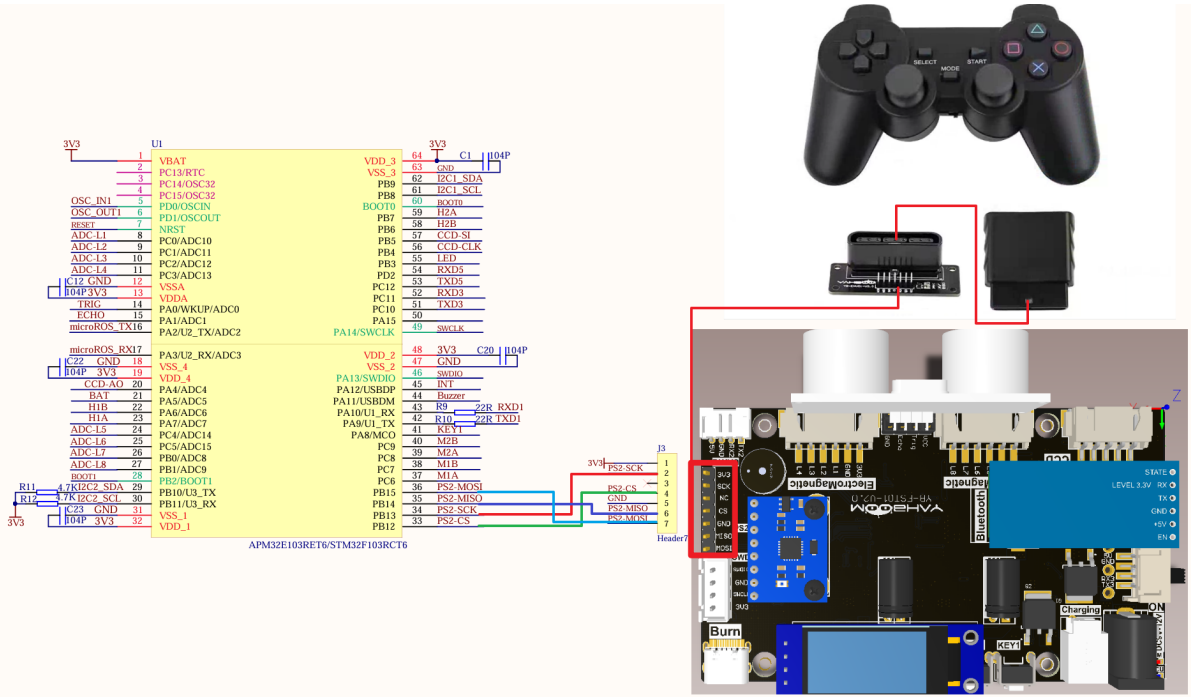
2.4G handle-Data reading (SPI)

The tutorial combines OLED and serial port to display the key data of 2.4G wireless controller.

```
The tutorial only introduces the standard library project code
```

Hardware connection



Since we have configured a special connection line, we only need to install it to the corresponding interface:

Peripherals	Development Board
PS2: CS	PB12
PS2: SCK	PB13
PS2: MISO	PB14
PS2: MOSI	PB15

Control Principle

- 2.4G Wireless Controller

The 2.4G wireless controller is mainly composed of a controller and a receiver (the receiver adapter board is convenient for pin connection with the development board).

> **Controller**

Send key information to the receiver.

> **Receiver**

Receives data sent by the controller and passes it to the development board; the development board can also configure the controller's sending mode through the data sent by the controller.

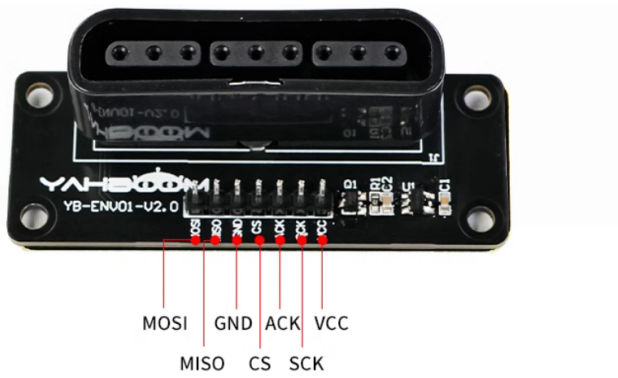
> **Handle Button Description**

Handle Button	Program Function Description	Program Definition (Part)
Direction Key (①): △	Forward	PSB_PAD_UP
Direction Key (①): ▽	Backward	PSB_PAD_DOWN
Direction Key (①): ◁	Turn Left	PSB_PAD_LEFT
Direction key (①): ▷	Turn right	PSB_PAD_RIGHT
Function key (②): ▲	Forward	PSB_TRIANGLE/PSB_GREEN
Function key (②): ×	Backward	PSB_CROSS/PSB_BLUE
Function key (②): □	Turn left	PSB_SQUARE/PSB_PINK
Function key (②): ○	Turn right	PSB_CIRCLE/PSB_RED
Joystick: ③	Control direction (valid in dual light mode)	PSB_L3/PSS_LX/PSS_LY
Joystick: ④	Control direction (valid in dual light mode)	PSB_R3/PSS_RX/PSS_RY
Button (⑤): L1/L2	Accelerate	PSB_L1/PSB_L2
Button (⑥): R1/R2	Slow down	PSB_R1/PSB_R2
Button (⑦): START	Turn off power saving mode	PSB_START
Button (⑧): MODE	Switch mode (indicator light display: single light and dual light mode)	
Button (⑨): SELECT	Not used	PSB_SELECT

- Receiver and adapter board pin description

Receiver pin	Description
DI/DAT	Signal flow, from handle to host, this signal is an 8-bit serial data, synchronously transmitted on the falling edge of the clock. The signal is read when the clock changes from high to low.

Receiver pin	Description
DI/DAT	Signal flow, from host to handle, this signal is opposite to DI, the signal is an 8-bit serial data, synchronously transmitted on the falling edge of the clock.
NC	Empty port.
GND	Receiver working power supply, power supply range 3~5V.
VDD	Used to provide the handle trigger signal. During communication, it is at a low level.
CS/SEL	Signal flow, from the host to the handle, this signal is relative to DI, the signal is an 8-bit serial data, synchronously transmitted on the falling edge of the clock.
CS/SEL	Clock signal, sent by the host, used to keep data synchronization.
NC	Empty port.
ACK	Acknowledgement signal from the handle to the host. This signal goes low at the last cycle of each 8-bit data transmission and CS remains low. If the CS signal does not go low, the PS host will try another peripheral in about 60 microseconds. The ACK port is not used during programming. (Can be ignored)



Software configuration

Pin definition

Main control chip	Pin	Main function (after reset)	Default multiplexing function	Redefine function
STM32F103RCT6	PB12	PB12	SPI2_NSS/I2C2_SMBAI/USART3_CK/TIM1_BKIN	
STM32F103RCT6	PB13	PB13	SPI2_SCK/USART3_CTS/ TIM1_CH1N	
STM32F103RCT6	PB14	PB14	SPI2_MISO/USART3_RTS/TIM1_CH2N	
STM32F103RCT6	PB15	PB15	SPI2_MOSI/TIM1_CH3N	

Software code

Since the default function of the pin is the normal IO pin function, we need to use the multiplexing function.

```
Product supporting materials source code path: Attachment → Source code summary → 2.Extended_Course → 7.Control_Handle
```

Control function

The tutorial only briefly introduces the code, you can open the project source code to read it in detail.

> **PS2_Init**

```
void PS2_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure; //Define GPIO_InitStructure structure
    //DI
    RCC_APB2PeriphClockCmd(PS_RCC_DI,ENABLE);
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IPU; //Pull-up, pull-down and floating input modes will cause abnormalities when the handle is not connected
    GPIO_InitStructure.GPIO_Pin=PS_PIN_DI;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz; //50MHZ
    GPIO_Init(PS_PORT_DI,&GPIO_InitStructure);

    //DO
    RCC_APB2PeriphClockCmd(PS_RCC_DO,ENABLE); //Turn on GPIOB clock
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP; //Multiplexed push-pull output mode
    GPIO_InitStructure.GPIO_Pin=PS_PIN_DO; //DO port
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz; //50MHZ
    GPIO_Init(PS_PORT_DO,&GPIO_InitStructure);

    //CS
    RCC_APB2PeriphClockCmd(PS_RCC_CS,ENABLE); //Turn on GPIOB clock
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP; //Multiplexed push-pull output mode
    GPIO_InitStructure.GPIO_Pin=PS_PIN_CS; //CS port
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz; //50MHZ
    GPIO_Init(PS_PORT_CS,&GPIO_InitStructure);

    //SCK
    RCC_APB2PeriphClockCmd(PS_RCC_CLK,ENABLE); //Turn on GPIOB clock
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP; //Multiplexed push-pull output mode
    GPIO_InitStructure.GPIO_Pin=PS_PIN_CLK; //CLK port
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz; //50MHZ
    GPIO_Init(PS_PORT_CLK,&GPIO_InitStructure);
}
```

> **PS2_Cmd**

```
void PS2_Cmd(u8 CMD)
```

```
{
    volatile u16 ref=0x01;
    Data[1] = 0;
    for(ref=0x01;ref<0x0100;ref<<=1)
    {
        if(ref&CMD)
        {
            DO_H;
        }
        else DO_L;
        CLK_H;
        DELAY_TIME;
        CLK_L;
        DELAY_TIME;
        CLK_H;
        if(DI)
            Data[1] = ref|Data[1];
    }
    delay_us(16);
}
```

PS2_RedLight

```
u8 PS2_RedLight(void)
{
    CS_L;
    PS2_Cmd(Comd[0]);
    PS2_Cmd(Comd[1]);
    CS_H;
    if( Data[1] == 0X73)   return 0 ;
    else return 1;
}
```

PS2_ReadData

```
void PS2_ReadData(void)
{
    volatile u8 byte=0;
    volatile u16 ref=0x01;
    CS_L;
    PS2_Cmd(Comd[0]);
    PS2_Cmd(Comd[1]);
    for(byte=2;byte<9;byte++)
    {
        for(ref=0x01;ref<0x100;ref<<=1)
        {
            CLK_H;
            DELAY_TIME;
            CLK_L;
            DELAY_TIME;
            CLK_H;
              if(DI)
              Data[byte] = ref|Data[byte];
        }
```

```
        delay_us(16);
    }
    CS_H;
}
```

PS2_DataKey

```
u8 PS2_DataKey()
{
    u8 index;
    PS2_ClearData();
    PS2_ReadData();
    Handkey=(Data[4]<<8)|Data[3];
    for(index=0;index<16;index++)
    {
        if((Handkey&(1<<(MASK[index]-1)))==0)
        return index+1;
    }
    return 0;
}
```

PS2_AnologData

```
u8 PS2_AnologData(u8 button)
{
    return Data[button];
}
```

PS2_ClearData

```
void PS2_ClearData()
{
    u8 a;
    for(a=0;a<9;a++)
        Data[a]=0x00;
}
```

PS2_Vibration

```
void PS2_Vibration(u8 motor1, u8 motor2)
{
    CS_L;
    delay_us(16);
    PS2_Cmd(0x01);
    PS2_Cmd(0x42);
    PS2_Cmd(0X00);
    PS2_Cmd(motor1);
    PS2_Cmd(motor2);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    CS_H;
```

```
    delay_us(16);
}
```

PS2_ShortPoll

```
void PS2_ShortPoll(void)
{
    CS_L;
    delay_us(16);
    PS2_Cmd(0x01);
    PS2_Cmd(0x42);
    PS2_Cmd(0X00);
    PS2_Cmd(0x00);
    PS2_Cmd(0x00);
    CS_H;
    delay_us(16);
}
```

PS2_EnterConfing

```
void PS2_EnterConfing(void)
{
    CS_L;
    delay_us(16);
    PS2_Cmd(0x01);
    PS2_Cmd(0x43);
    PS2_Cmd(0X00);
    PS2_Cmd(0x01);
    PS2_Cmd(0x00);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    CS_H;
    delay_us(16);
}
```

PS2_TurnOnAnalogMode

```
void PS2_TurnOnAnalogMode(void)
{
    CS_L;
    PS2_Cmd(0x01);
    PS2_Cmd(0x44);
    PS2_Cmd(0X00);
    PS2_Cmd(0x01);
    PS2_Cmd(0xEE);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    PS2_Cmd(0X00);
    CS_H;
    delay_us(16);
```

```
}
```

PS2_VibrationMode

```
void PS2_VibrationMode(void)
{
    CS_L;
    delay_us(16);
    PS2_Cmd(0x01);
    PS2_Cmd(0x4D);
    PS2_Cmd(0X00);
    PS2_Cmd(0x00);
    PS2_Cmd(0X01);
    CS_H;
    delay_us(16);
}
```

PS2_ExitConfing

```
void PS2_ExitConfing(void)
{
    CS_L;
    delay_us(16);
    PS2_Cmd(0x01);
    PS2_Cmd(0x43);
    PS2_Cmd(0X00);
    PS2_Cmd(0x00);
    PS2_Cmd(0x5A);
    PS2_Cmd(0x5A);
    PS2_Cmd(0x5A);
    PS2_Cmd(0x5A);
    PS2_Cmd(0x5A);
    CS_H;
    delay_us(16);
}
```

PS2_SetInit

```
void PS2_SetInit(void)
{
    PS2_ShortPoll();
    PS2_ShortPoll();
    PS2_ShortPoll();
    PS2_EnterConfing();
    PS2_TurnOnAnalogMode();
    //PS2_VibrationMode();
    PS2_ExitConfing();
}
```

PS2_Data_Show

```
void PS2_Data_Show(void)
{
```

```
    g_PS2_LX = PS2_AnalogData(PSS_LX);
    g_PS2_LY = PS2_AnalogData(PSS_LY);
    g_PS2_RX = PS2_AnalogData(PSS_RX);
    g_PS2_RY = PS2_AnalogData(PSS_RY);
    g_PS2_KEY = PS2_DataKey();


    if ((g_PS2_LX == 255) && (g_PS2_LY == 255) && (g_PS2_RX == 255) && (g_PS2_RY
== 255))
    {
        if (g_flag == 1)
        {
            printf("PS2 mode is RED or GREEN mode \r\n");
            printf("Or PS2 disconnect! \r\n");
            g_flag = 0;
        }
    }

    else
    {
        if (g_flag == 0)
        {
            printf("PS2 mode is RED^GREEN mode \r\n");
            g_flag = 1;
        }
        // Only the red green mode has the correct joystick value
        if (g_PS2_LX > 130 || g_PS2_LX < 100)
            printf("PS2_LX = %d \r\n", g_PS2_LX);
        if (g_PS2_LY > 130 || g_PS2_LY < 100)
            printf("PS2_LY = %d \r\n", g_PS2_LY);
        if (g_PS2_RX > 130 || g_PS2_RX < 100)
            printf("PS2_RX = %d \r\n", g_PS2_RX);
        if (g_PS2_RY > 130 || g_PS2_RY < 100)
            printf("PS2_RY = %d \r\n", g_PS2_RY);
    }

    switch (g_PS2_KEY)
    {
    case PSB_SELECT:
        printf("key = PSB_SELECT\r\n");
        OLED_Draw_Line("key = PSB_SELECT", 1, true, true);
        break;
    case PSB_L3:
        printf("key = PSB_L3\r\n");
        OLED_Draw_Line("key = PSB_L3", 1, true, true);
        break;
    case PSB_R3:
        printf("key = PSB_R3\r\n");
        OLED_Draw_Line("key = PSB_R3", 1, true, true);
        break;
    case PSB_START:
        printf("key = PSB_START\r\n");
        OLED_Draw_Line("key = PSB_START", 1, true, true);
        break;
    case PSB_PAD_UP:
        printf("key = PSB_PAD_UP\r\n");
```

```
            OLED_Draw_Line("key = PSB_PAD_UP", 1, true, true);
            break;
        case PSB_PAD_RIGHT:
            printf("key = PSB_PAD_RIGHT\r\n");
            OLED_Draw_Line("key = PSB_PAD_RIGHT", 1, true, true);
            break;
        case PSB_PAD_DOWN:
            printf("key = PSB_PAD_DOWN\r\n");
            OLED_Draw_Line("key = PSB_PAD_DOWN", 1, true, true);
            break;
        case PSB_PAD_LEFT:
            printf("key = PSB_PAD_LEFT\r\n");
            OLED_Draw_Line("key = PSB_PAD_LEFT", 1, true, true);
            break;
        case PSB_L2:
            printf("key = PSB_L2\r\n");
            OLED_Draw_Line("key = PSB_L2", 1, true, true);
            break;
        case PSB_R2:
            printf("key = PSB_R2\r\n");
            OLED_Draw_Line("key = PSB_R2", 1, true, true);
            break;
        case PSB_L1:
            printf("key = PSB_L1\r\n");
            OLED_Draw_Line("key = PSB_L1", 1, true, true);
            break;
        case PSB_R1:
            printf("key = PSB_R1\r\n");
            OLED_Draw_Line("key = PSB_R1", 1, true, true);
            break;
        case PSB_GREEN:
            printf("key = PSB_GREEN\r\n");
            OLED_Draw_Line("key = PSB_GREEN", 1, true, true);
            break;
        case PSB_RED:
            printf("key = PSB_RED\r\n");
            OLED_Draw_Line("key = PSB_RED", 1, true, true);
            break;
        case PSB_BLUE:
            printf("key = PSB_BLUE\r\n");
            OLED_Draw_Line("key = PSB_BLUE", 1, true, true);
            break;
        case PSB_PINK:
            printf("key = PSB_PINK\r\n");
            OLED_Draw_Line("key = PSB_PINK", 1, true, true);
            break;
        }
        delay_ms(100);
    }
```

Experimental phenomenon

The Control_Handle.hex file generated by the project compilation is located in the OBJ folder of the Control_Handle project. Find the Control_Handle.hex file corresponding to the project and use the FlyMcu software to download the program to the development board.

After the program is successfully downloaded: the serial port prints and the OLED displays the key value pressed by the wireless handle.

```
When using the serial port debugging assistant, you need to pay attention to the
serial port settings. If the settings are wrong, the phenomenon may be
inconsistent.
```