Driving RGB light strips

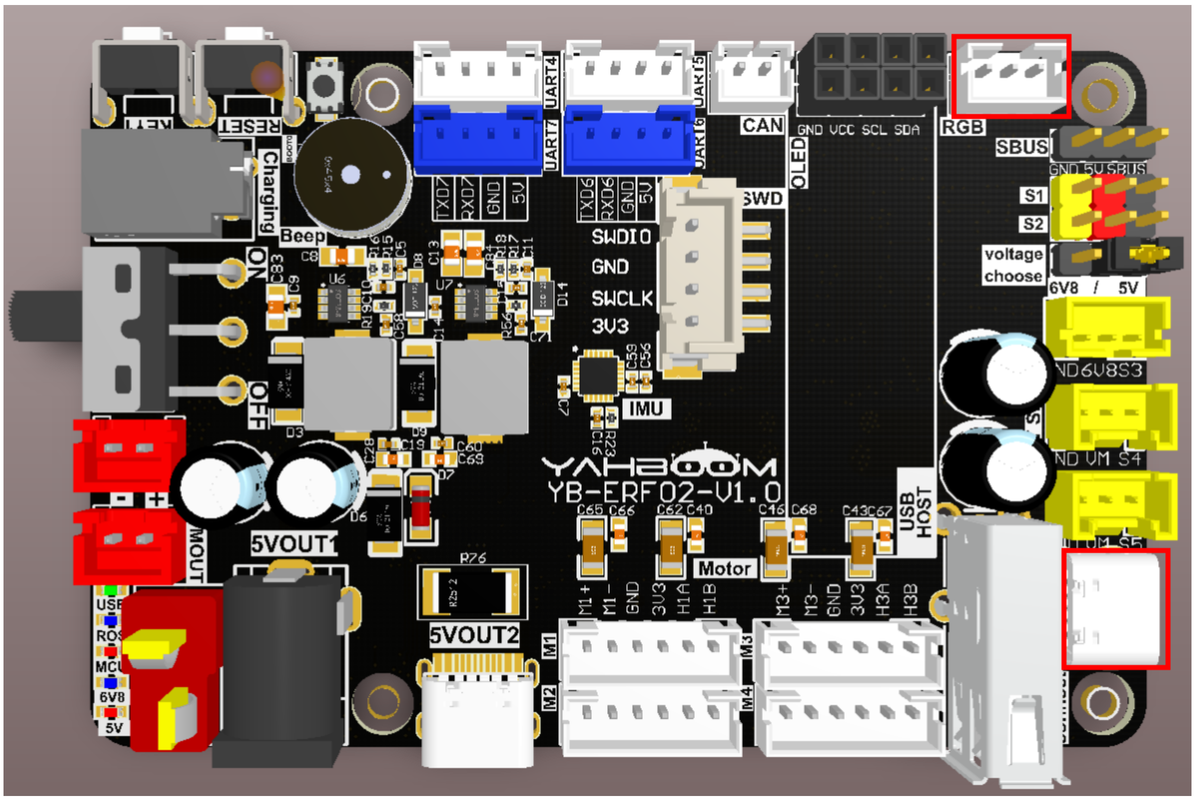
1. Experimental Purpose

Learn how to control a WS2812 RGB light strip using the SPI functionality of the STM32 controller board.

2. Hardware Connection

As shown in the figure below, the STM32 control board has an integrated RGB interface, but you need to connect an additional RGB light strip. You need to prepare your own RGB light strip and connect the type-C data cable to the computer and the USB Connect interface of the STM32 control board.

The RGB light strip is driven by the WS2812 chip, and the circuit design uses SPI+DMA to simulate the WS2812 control timing to drive the RGB light strip.

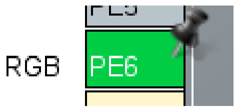


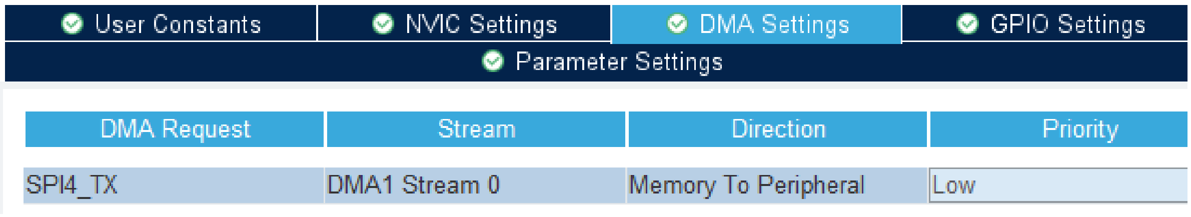
3. Core code analysis

The path corresponding to the program source code is:

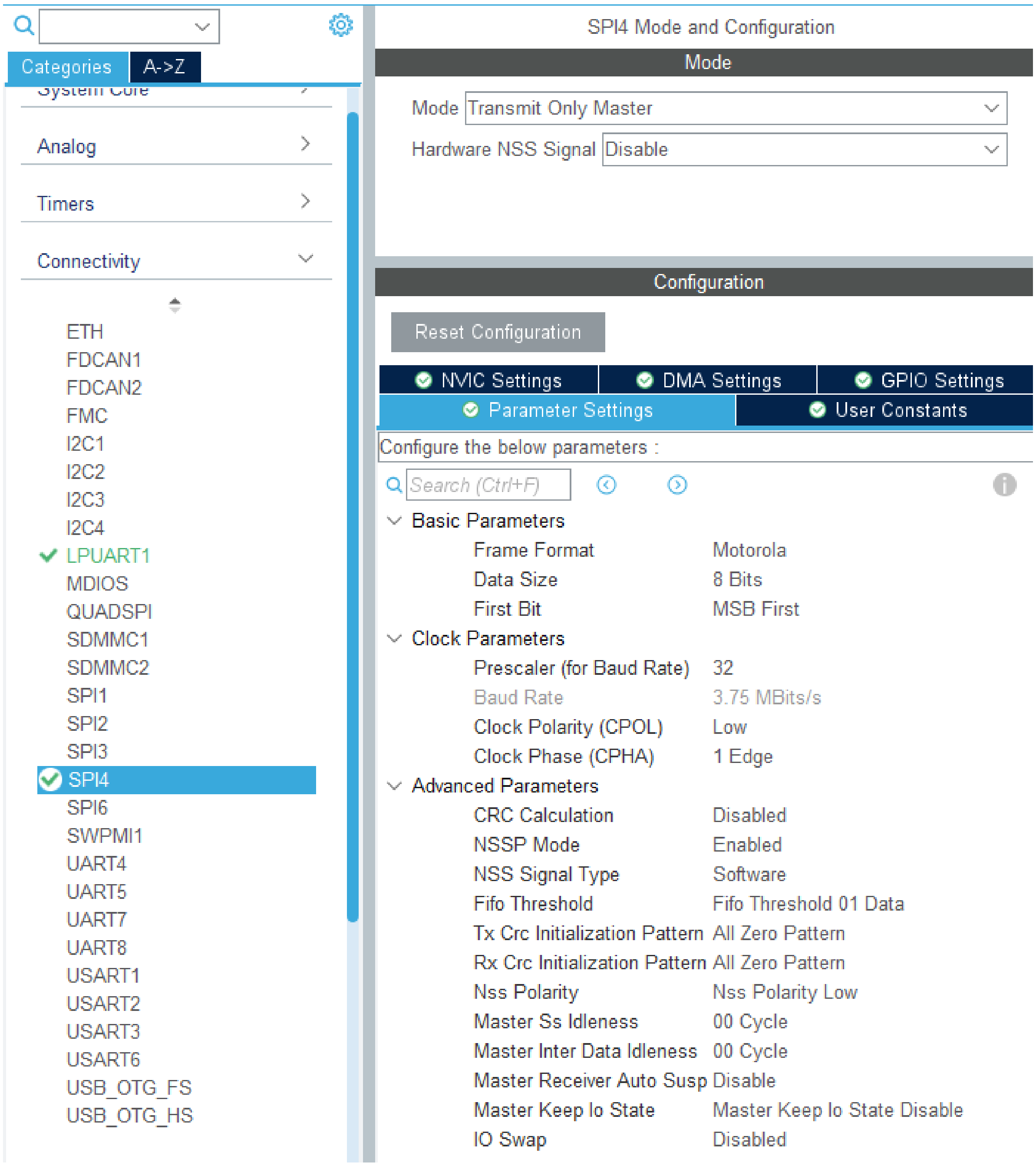
```
Board_Samples/STM32_Samples/RGB
```

According to the pin assignment, the signal pin of the RGB light strip is connected to PE6 (SPI4-MOSI), so the SPI4 is initialized to master mode. The clock frequency of SPI4 is 120MHz, divided by 32, so the final communication frequency of SPI4 is 3.75Mhz.





DMA Request	Stream	Direction	Priority
SPI4_TX	DMA1 Stream 0	Memory To Peripheral	Low



SPI4 Mode and Configuration

Mode

Mode: Transmit Only Master

Hardware NSS Signal: Disable

Configuration

Configure the below parameters :

Basic Parameters

Frame Format	Motorola
Data Size	8 Bits
First Bit	MSB First

Clock Parameters

Prescaler (for Baud Rate)	32
Baud Rate	3.75 MBits/s
Clock Polarity (CPOL)	Low
Clock Phase (CPHA)	1 Edge

Advanced Parameters

CRC Calculation	Disabled
NSSP Mode	Enabled
NSS Signal Type	Software
Fifo Threshold	Fifo Threshold 01 Data
Tx Crc Initialization Pattern	All Zero Pattern
Rx Crc Initialization Pattern	All Zero Pattern
Nss Polarity	Nss Polarity Low
Master Ss Idleness	00 Cycle
Master Inter Data Idleness	00 Cycle
Master Receiver Auto Susp	Disable
Master Keep Io State	Master Keep Io State Disable
IO Swap	Disabled

```
void MX_SPI4_Init(void)
{
  hspi4.Instance = SPI4;
  hspi4.Init.Mode = SPI_MODE_MASTER;
  hspi4.Init.Direction = SPI_DIRECTION_2LINES_TXONLY;
  hspi4.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi4.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi4.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi4.Init.NSS = SPI_NSS_SOFT;
  hspi4.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_32;
  hspi4.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi4.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi4.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi4.Init.CRCPolynomial = 0x0;
  hspi4.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
  hspi4.Init.NSSPolarity = SPI_NSS_POLARITY_LOW;
  hspi4.Init.FifoThreshold = SPI_FIFO_THRESHOLD_01DATA;
  hspi4.Init.TxCRCInitializationPattern =
SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN;
  hspi4.Init.RxCRCInitializationPattern =
SPI_CRC_INITIALIZATION_ALL_ZERO_PATTERN;
  hspi4.Init.MasterSSIdleness = SPI_MASTER_SS_IDLENESS_00CYCLE;
  hspi4.Init.MasterInterDataIdleness = SPI_MASTER_INTERDATA_IDLENESS_00CYCLE;
  hspi4.Init.MasterReceiverAutoSusp = SPI_MASTER_RX_AUTOSUSP_DISABLE;
  hspi4.Init.MasterKeepIOState = SPI_MASTER_KEEP_IO_STATE_DISABLE;
  hspi4.Init.IOSwap = SPI_IO_SWAP_DISABLE;
  if (HAL_SPI_Init(&hspi4) != HAL_OK)
  {
    Error_Handler();
  }
}
```

According to the timing requirements of ws2812, SPI's 4-bit data is used to simulate ws2812's one-bit data. 0x0E represents the ws2812's "1" code, and 0x08 represents the ws2812's "0" code.

```
//Hardware SPI simulates ws2812 timing (use SPI's 4-bit data to simulate ws2812's
1-bit data)
// The SPI communication frequency is required to be 3.75M, and the time to
transmit one bit of data is about 267ns (nanoseconds)
// ___
// | |_| 0b1110 high level
// _
// | |___| 0b1000 low level
#define TIMING_ONE 0x0E
#define TIMING_ZERO 0x08
```

Define a structure to cache the data of the RGB light strip.

```
#define RGB_CTRL_ALL 0xFF
#define MAX_RGB 8



#define RGB_BIT_WIDTH 4
#define RGB_BIT_SIZE (RGB_BIT_WIDTH*3)
#define RGB_RESET_WIDTH 100
```

```
typedef struct rgb_ws2812
{
    uint8_t reset[RGB_RESET_WIDTH];
    union
    {
        uint8_t Buff[RGB_BIT_SIZE];
        struct
        {
            uint8_t G[RGB_BIT_WIDTH]; // G First
            uint8_t R[RGB_BIT_WIDTH]; // R Second
            uint8_t B[RGB_BIT_WIDTH]; // B Third
        } RGB;
    } Strip[MAX_RGB];
} ws2812_t;
```

Convert RGB color values into ws2812 structure data.

```
static void WS2812_Set_Color_One(uint8_t index, uint32_t RGB)
{
    if (index >= MAX_RGB) return;
    uint8_t i;
    uint64_t TempR = 0, TempG = 0, TempB = 0;

    for(i = 0; i < 8; i++)
    {
        (RGB & 0x00010000) == 0 ? (TempR |= (TIMING_ZERO<<(i*RGB_BIT_WIDTH))) :
(TempR |= (TIMING_ONE<<(i*RGB_BIT_WIDTH)));
        (RGB & 0x00000100) == 0 ? (TempG |= (TIMING_ZERO<<(i*RGB_BIT_WIDTH))) :
(TempG |= (TIMING_ONE<<(i*RGB_BIT_WIDTH)));
        (RGB & 0x00000001) == 0 ? (TempB |= (TIMING_ZERO<<(i*RGB_BIT_WIDTH))) :
(TempB |= (TIMING_ONE<<(i*RGB_BIT_WIDTH)));
        RGB >>= 1;
    }
    for (i = 0; i < RGB_BIT_WIDTH; i++)
    {
        g_ws2812.Strip[index].RGB.R[i] = TempR >> (8*(RGB_BIT_WIDTH-i-1));
        g_ws2812.Strip[index].RGB.G[i] = TempG >> (8*(RGB_BIT_WIDTH-i-1));
        g_ws2812.Strip[index].RGB.B[i] = TempB >> (8*(RGB_BIT_WIDTH-i-1));
    }
}
```

There are two ways to set the color of the RGB light strip. The first is to write the merged color data, and the second is to write the RGB values separately.

```
void RGB_Set_Color_U32(uint8_t index, uint32_t color)
{
    if (index < MAX_RGB)
    {
        WS2812_Set_Color_One(index, color);
        return;
    }
    if (index == RGB_CTRL_ALL)
    {
        for (uint16_t i = 0; i < MAX_RGB; i++)
        {
            WS2812_Set_Color_One(i, color);
        }
    }
}

void RGB_Set_Color(uint8_t index, uint8_t r, uint8_t g, uint8_t b)
{
    uint32_t color = r << 16 | g << 8 | b;
    RGB_Set_Color_U32(index, color);
}
```

SPI transmits data to the RGB light strip to update the color of the RGB light strip. This function must be called after each modification of the RGB light strip color to send the cached data of the ws2812 structure to the RGB light strip via SPI.

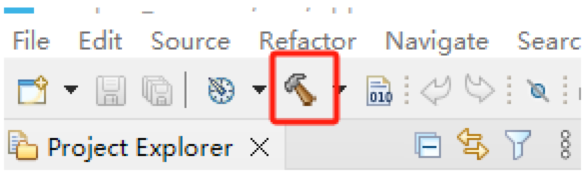
```
void RGB_Update(void)
{
    // WS2812_Send_Data((uint8_t*)&g_ws2812,
RGB_BIT_SIZE*MAX_RGB+RGB_RESET_WIDTH);
    WS2812_Send_Data((uint8_t*)&g_ws2812, sizeof(g_ws2812));
}
```

Start the computer and control the color change of the RGB light bar.

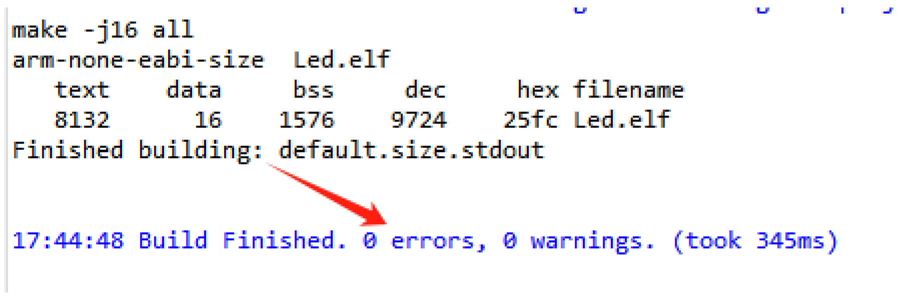
```
void App_Handle(void)
{
    RGB_Init();

    while (1)
    {
        rgb_count++;
        if (rgb_count > 100)
        {
            rgb_count = 0;
            rgb_color = (rgb_color + 1) % 5;
            printf("color:%d\n", rgb_color);
            if (rgb_color == 0)
            {
                RGB_Clear();
                RGB_Update();
            }
            else if (rgb_color == 1)
            {
                RGB_Clear();
                RGB_Set_Color(RGB_CTRL_ALL, 0xFF, 0x00, 0x00);
                RGB_Update();
            }
            else if (rgb_color == 2)
            {
                RGB_Clear();
                RGB_Set_Color(RGB_CTRL_ALL, 0x00, 0xFF, 0x00);
                RGB_Update();
            }
            else if (rgb_color == 3)
            {
                RGB_Clear();
                RGB_Set_Color(RGB_CTRL_ALL, 0x00, 0x00, 0xFF);
                RGB_Update();
            }
            else if (rgb_color == 4)
            {
                RGB_Clear();
                RGB_Set_Color(RGB_CTRL_ALL, 0xFF, 0xFF, 0xFF);
                RGB_Update();
            }
        }
        App_Led_Mcu_Handle();
        HAL_Delay(10);
    }
}
```

4. Compile, download and burn firmware

Select the project to be compiled in the file management interface of STM32CUBEIDE and click the compile button on the toolbar to start compiling.



If there are no errors or warnings, the compilation is complete.



```
make -j16 all
arm-none-eabi-size   Led.elf
   text    data     bss     dec     hex filename
   8132      16    1576    9724    25fc Led.elf
Finished building: default.size.stdout


17:44:48 Build Finished. 0 errors, 0 warnings. (took 345ms)
```

Press and hold the BOOT0 button, then press the RESET button to reset, release the BOOT0 button to enter the serial port burning mode. Then use the serial port burning tool to burn the firmware to the board.

If you have STlink or JLink, you can also use STM32CUBEIDE to burn the firmware with one click, which is more convenient and quick.

5. Experimental Results

The MCU_LED light flashes every 200 milliseconds.

The RGB light strip updates the color of the RGB light strip every second, and the color sequence is red, green, blue, white and off.