Stm32F407_USB2SPI

1.功能：

1. 使用 python 脚本实现模块接受发送数据，可以灵活定义格式
2. 3 个 CS 端口
3. 8 个种频率选择，最高 41MHz，依次二分频
4. 4 种 spi 模式选择：
    a. Mode0：闲时低电平，第一个时钟沿触发
    b. Mode1：闲时低电平，第二个时钟沿触发
    c. Mode2：闲时高电平，第一个时钟沿触发
    d. Mode3：闲时高电平，第二个时钟沿触发

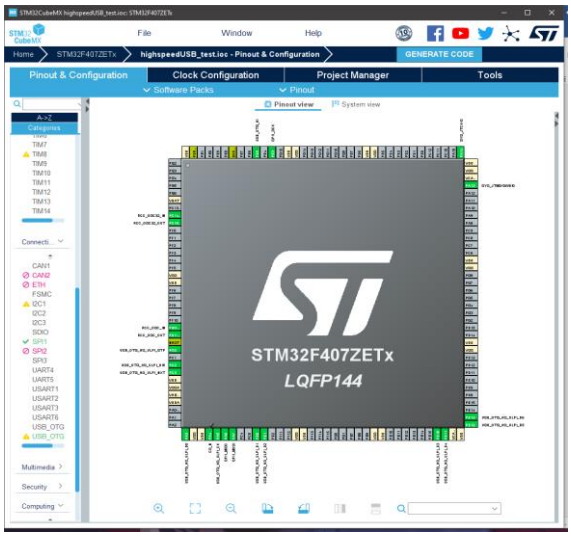
其他可拓展功能：

1. 从机模式
2. 读写函数的通用（适应不同的帧格式）
3. 数据顺序选择
4. DMA,CRC,TI?

2.原理：

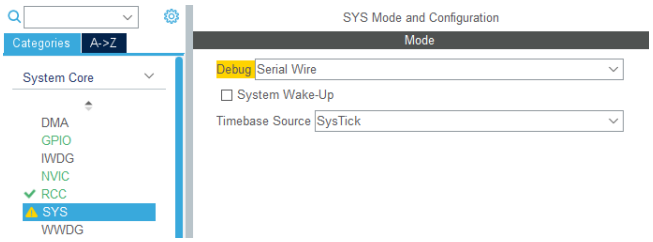
1.Stm32F407VGT6 总体配置

可以使用 STM32CubeMX 进行快速的 HAL 库开发，相比库函数更方便快捷

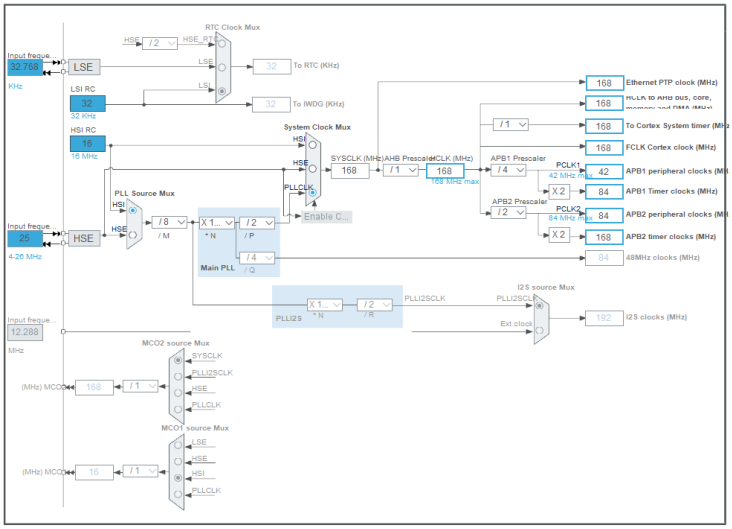


在 STM32CubeMx 中，配置模块所需要的端口，中断与中断优先级和时钟等

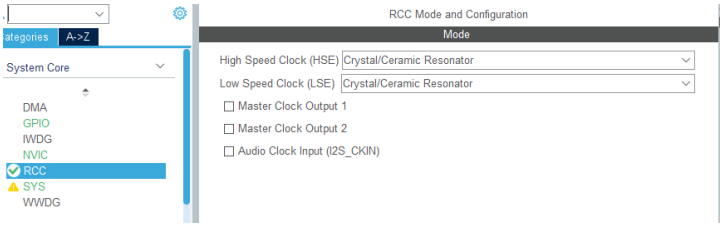
a. Debug 模式：有 ST-Link 所以使用 Serial Wire



b. 时钟树：把 SPI 所需时钟调到最高（因为本次应用没有功耗要求，所以方便起见所有时钟频率都拉到最高了）



将外部晶振作为高速时钟源



c. 使用 Keil,选择 IDE

端口方面配置 3 个作为 CS 使能端的 GPIO 口，SPI1 外设和 HighSpeedUSB

2.STM32F4_SPI

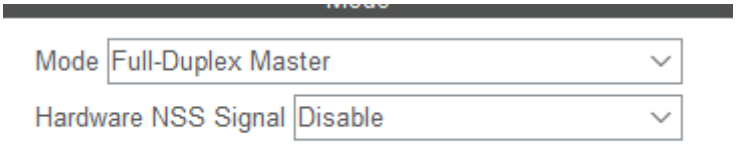
SPI 特性

- 基于三条线的全双工同步传输
- 基于双线的单工同步传输，其中一条可作为双向数据线
- 8 位或 16 位传输帧格式选择
- 主模式或从模式操作
- 多主模式功能
- 8 个主模式波特率预分频器（最大值为 $f_{PCLK}/2$）
- 从模式频率（最大值为 $f_{PCLK}/2$）
- 对于主模式和从模式都可实现更快的通信
- 对于主模式和从模式都可通过硬件或软件进行 NSS 管理：动态切换主/从操作
- 可编程的时钟极性和相位
- 可编程的数据顺序，最先移位 MSB 或 LSB
- 可触发中断的专用发送和接收标志
- SPI 总线忙状态标志
- SPI TI 模式
- 用于确保可靠通信的硬件 CRC 功能：
  - 在发送模式下可将 CRC 值作为最后一个字节发送
  - 根据收到的最后一个字节自动进行 CRC 错误校验
- 可触发中断的主模式故障、上溢和 CRC 错误标志
- 具有 DMA 功能的 1 字节发送和接收缓冲器：发送和接收请求

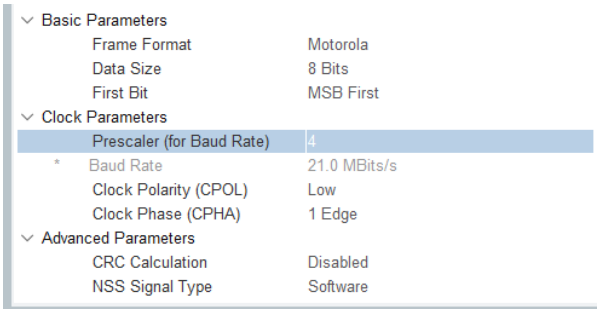
stm32F4 的 SPI 特性

配置：

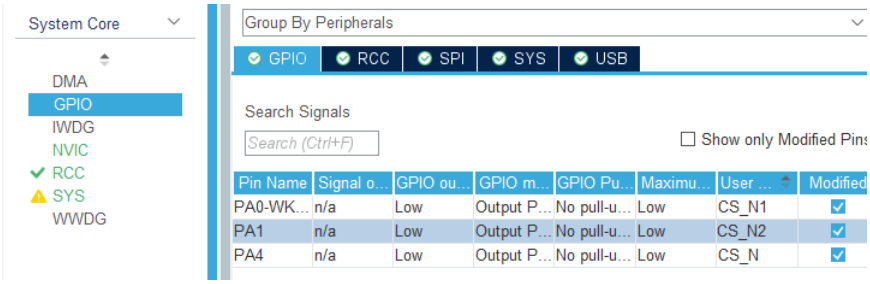
1. 在 STM32CubeMX 中，配置 SPI 为全双工主机模式，使用软件使能，所以 Hardware NSS 选择 Disable



2. SPI 参数：
   8bit 传输，默认高位先，默认四分频（也就是 21MHz 的 SCLK），Mode0，无需中断，其他默认配置



3. CS 端口：
   配置 A0,A1,A4 为三个使能端，配为 GPIO_OUTPUT



4. keil 代码部分

SPI 结构体中，

通过对 CLKPolarity 和 CLKPhase 参数的修改能够配置相应的四种模式；

```
if(mode == 0x00)
{
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
}
else if(mode == 0x01)
{
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_2EDGE;
}
else if(mode == 0x02)
{
  hspi1.Init.CLKPolarity = SPI_POLARITY_HIGH;
  hspi1.Init.CLKPhase = SPI_PHASE_2EDGE;
}
else if(mode == 0x03)
{
  hspi1.Init.CLKPolarity = SPI_POLARITY_HIGH;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
}
```

通过对 BaudRatePrescaler 参数的修改能够配置相应的分频倍率

```
if(speedflag == 0x01)   hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
else if(speedflag == 0x02)    hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_4;
else if(speedflag == 0x03)    hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
else if(speedflag == 0x04)    hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_16;
else if(speedflag == 0x05)    hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_32;
else if(speedflag == 0x06)    hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_64;
else if(speedflag == 0x07)    hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_128;
else if(speedflag == 0x08)    hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_256;
```

SPI 读/写 1Byte 的函数

```
uint8_t spi1_read_write_byte(uint8_t txdata)
{
  uint8_t rxdata;
  HAL_SPI_TransmitReceive(&hspi1, &txdata, &rxdata,
  return rxdata;
}
```

通过 usb 发送过来的配置数据，确认开启那几个 CS 端口，进行相应的软件触发高低电平，默认将 CS 端口设为高电平

在二者的基础上实现 AMC7836 的基本读写函数

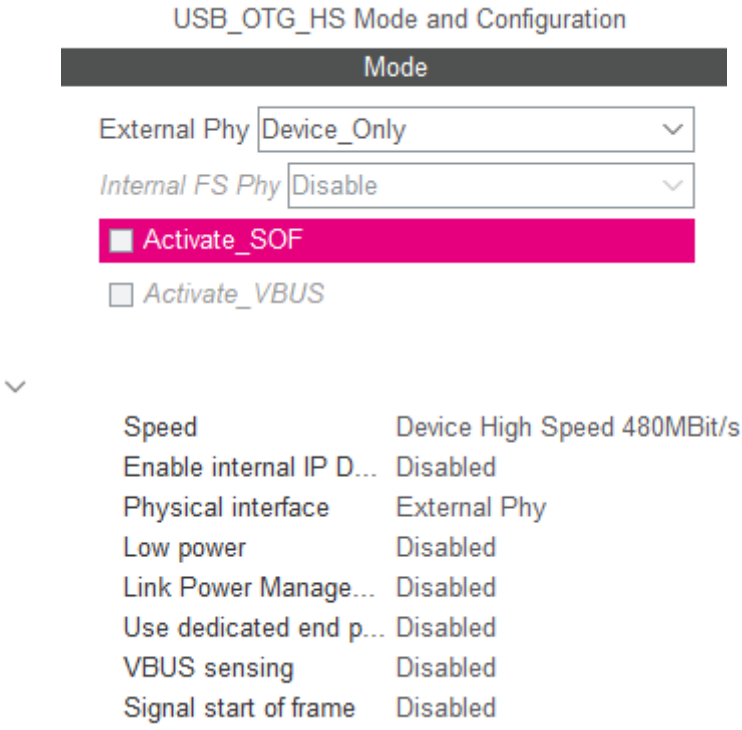
```c
uint8_t AMC7836_Read(uint8_t addr)
{
  uint8_t ret;
  ReSetCS_N();
  spi1_read_write_byte(0x80);
  spi1_read_write_byte(addr);
  ret = spi1_read_write_byte(0x00);
  SetCS_N();
  return ret;
}
void AMC7836_Write(uint8_t addr, uint8_t data)
{
  ReSetCS_N();
  spi1_read_write_byte(0x00);
  spi1_read_write_byte(addr);
  spi1_read_write_byte(data);
  SetCS_N();
}
```

3.STM32F4_USB

1).STM32F4 有 FullSpeedUSB 和 HighSpeedUSB 两个 USB 外设. FullSeepdUSB 可以直接使用，速度低；HighSpeedUSB 需要外置的驱动芯片，速度高。toomoss 的这块开发板有 USB3300 作为驱动，使用的是 HighSpeed 的端口

笔者第一次使用 USB 外设，经过了解，可以通过将 USB 配置为 Device 模式（从机模式），然后设置为虚拟串口的形式就能进行简单的通信（速度方面没有测过，不过一般 python 也不会有高速传输的需求），实现 Python 使用 Serial 串口库进行电脑通过 USB 控制 SPI 外设。

2).具体配置，同参考 1（见下文）



3).keil 代码

代码部分主要是在正点原子 F407 FullSpeed 虚拟串口实验（参考 2）的基础上移植过来的，主要是实现了 print 函数和接受中断函数，主要改动在 usbd_cdc_if 这个文件中

```
/* USER CODE BEGIN PRIVATE_FUNCTIONS_IMPLEMENTATION */
uint8_t g_usb_usart_printf_buffer[USB_USART_REC_LEN];
void usb_printf(char *fmt, ...)
{
    uint16_t i;
    va_list ap;
    va_start(ap, fmt);
    vsprintf((char *)g_usb_usart_printf_buffer, fmt, ap);
    va_end(ap);
    i = strlen((const char *)g_usb_usart_printf_buffer);
    cdc_vcp_data_tx(g_usb_usart_printf_buffer, i);
}

void cdc_vcp_data_tx(uint8_t *data, uint32_t Len)
{
    USBD_CDC_SetTxBuffer(&hUsbDeviceHS, data, Len);
    USBD_CDC_TransmitPacket(&hUsbDeviceHS);
    HAL_Delay(CDC_POLLING_INTERVAL);
}
/* USER CODE END PRIVATE_FUNCTIONS_IMPLEMENTATION */
```

发送函数

```c
uint8_t  g_usb_usart_rx_buffer[USB_USART_REC_LEN]={0};
uint16_t g_usb_usart_rx_sta=0;
static int8_t CDC_Receive_HS(uint8_t* Buf, uint32_t *Len)
{
  uint8_t i;
  uint8_t res;
  uint8_t datalen = *Len;
  /* USER CODE BEGIN 11 */
  USBD_CDC_SetRxBuffer(&hUsbDeviceHS, &Buf[0]);
  USBD_CDC_ReceivePacket(&hUsbDeviceHS);

  for (i = 0; i < datalen; i++)
  {
      res = Buf[i];

      if ((g_usb_usart_rx_sta & 0x8000) == 0)      /* 接收未完成 */
      {
          if (g_usb_usart_rx_sta & 0x4000)         /* 接收到了0x0d */
          {
              if (res != 0x0a)/* 接收错误，只有0x0d,没有0x0a*/
              {
                  g_usb_usart_rx_buffer[g_usb_usart_rx_sta & 0X3FFF] = 0x0d;
                  g_usb_usart_rx_sta++;
                  if(res !=0x0d)
                  {
                    g_usb_usart_rx_sta = g_usb_usart_rx_sta & 0x3FFF;//复位

                    g_usb_usart_rx_buffer[g_usb_usart_rx_sta & 0X3FFF] = res;
                    g_usb_usart_rx_sta++;
                    if (g_usb_usart_rx_sta > (USB_USART_REC_LEN - 1))
                    {
                        g_usb_usart_rx_sta = 0; /* 接收数据溢出 重新开始接收 */
                    }
                  }
                  if(res == 0x0d)
                  {
                    g_usb_usart_rx_sta |= 0x4000;
                  }
              }
              else
              {
                  g_usb_usart_rx_sta |= 0x8000;  /* 接收完成了 */
              }
          }
          else    /* 还没收到0X0D */
          {
              if (res == 0x0d)
              {

                  g_usb_usart_rx_sta |= 0x4000;   /* 标记接收到了0X0D */
              }
              else
              {
                  g_usb_usart_rx_buffer[g_usb_usart_rx_sta & 0X3FFF] = res;
                  g_usb_usart_rx_sta++;

                  if (g_usb_usart_rx_sta > (USB_USART_REC_LEN - 1))
                  {
                      g_usb_usart_rx_sta = 0; /* 接收数据溢出 重新开始接收 */
                  }
              }
          }
      }
  }
```

主要改动

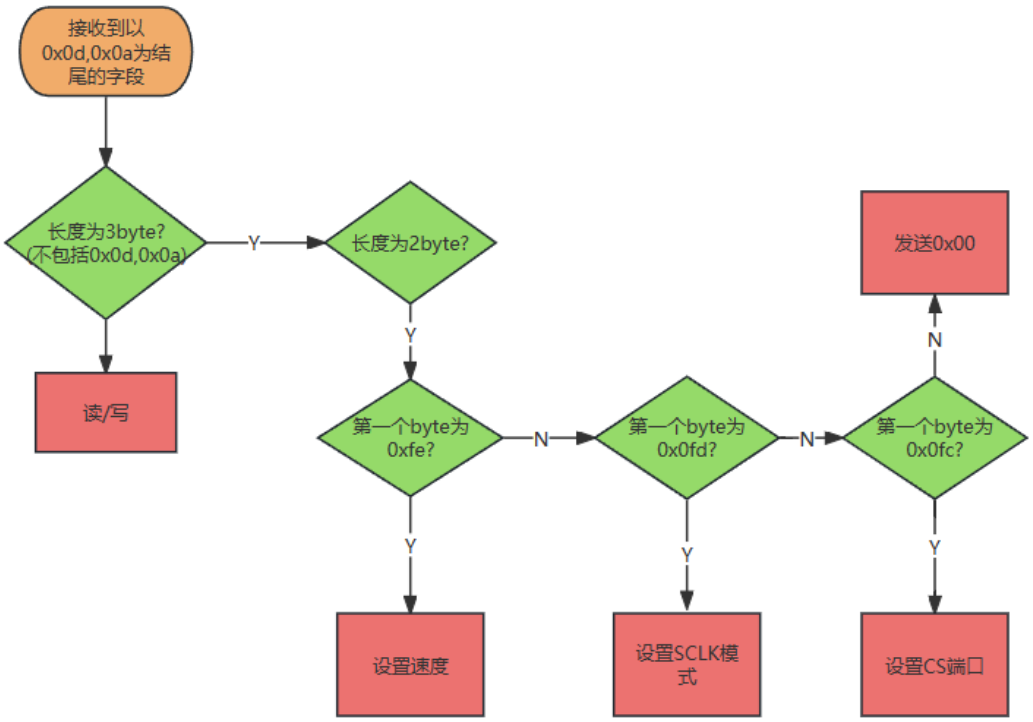
接收中断，改动原因见下文

```c
while (1)
{
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */


  if (g_usb_usart_rx_sta & 0x8000)
  {
    len = g_usb_usart_rx_sta & 0x3FFF;

    if(len == 3)
    {
    else if(len == 2)//set spi speed
    {
    g_usb_usart_rx_sta = 0; //reset state
  }
}
```

主函数中对接收数据的处理

4)发送/接收数据逻辑，帧格式

数据接收逻辑：接受 usb 发送过来的数据对 spi 进行控制



数据发送：读寄存器数据返回通过 usb_printf()函数进行字符发送，速度，模式，端口配置成功后会发送一个 byte:0x01

5).参考：
1. 调试 STM32F4 USB3300 USB HS 虚拟串口
2. 正点原子 F407FullSpeed 虚拟串口实验（STM32USB 的驱动里面也有）

4.python 代码部分

1). 因为模块使用 usb 虚拟串口，可以在 python 中使用串口库 serial 进行通信。同时因为没有类似于中断的方式在 PC 端接受数据，只有软件的读取方式，为了不丢失数据，所以通过新开一个线程进行持续的读取。（应该有更好的方法，但是我不知道）

2).串口收发部分网上随便找找都有

STM32 虚拟串口的检测：

```
ports_list = list(serial.tools.list_ports.comports())
if len(ports_list) <= 0:
    print("There are not serial device")
else:
    print("Available serial device list: ")
    for comport in ports_list:
        print(list(comport)[0], list(comport)[1])
        if list(comport)[1].count('STMicroelectronics'):
            com = list(comport)[0]
            print("you choose", com)
```

3).线程接受部分

```
def receive_thread(self):
    while True:
        data = self.ser.read(1)  # 读取一行数据
       self.buffer_queue.put(data)  # 将数据放入缓冲队列
```

初始化部分

```
self.receive_thread = threading.Thread(target=self.receive_thread)
self.receive_thread.start()  # 启动接收线程
```

读寄存器，就是把接受的 buff 中的数据读完为止，其他的配置 spi 的函数也差不多

```
#读寄存器，addr 为 1byte 的地址
def read(self,addr):
    stream = bytearray()
    self.ser.write(bytes([0x01, addr, 0x00, 0x0d, 0x0a]))#写寄存器
    time.sleep(0.01)
    while not self.buffer_queue.empty():
        data = self.buffer_queue.get()
        if len(data) != 0:
            stream.append(data[0])
    return stream.decode('utf-8')
```

3.使用方法:

1. Python 测试代码（于 STM32_USB2SPI_Test）

```
usb2spi = STM32_USB2SPI()#实例化

usb2spi.setSpeed(SPI_SPEED_d4)#设置速度
usb2spi.setMode(SPI_MODE_0)#设置 SPI 摸出
usb2spi.setCS_N(CS_OPEN, CS_OPEN, CS_OPEN)#设置 CS 端口

print(usb2spi.read(0x04))#读寄存器
```

附录:

1. 实物图