```c
#include "SPI.h"
#include "USART_SendString.h"

/*
Definition:  SPI1初始化配置模块
*/
void SPI_Hardware_Init(SPI_InitType SPI_InitStructure){
        RCC_EnableAPB2PeriphClk(RCC_APB2_PERIPH_GPIOA, ENABLE);
        RCC_EnableAPB2PeriphClk(RCC_APB2_PERIPH_SPI1, ENABLE);

        //定义CS片选引脚(GPIOA Pin_4)
        GPIO_InitType GPIO_InitStructure;
        GPIO_InitStructure.Pin = GPIO_PIN_4;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
        GPIO_InitPeripheral(GPIOA, &GPIO_InitStructure);
        //定义MOSI和CLK
        GPIO_InitStructure.Pin = GPIO_PIN_5 | GPIO_PIN_7;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_InitPeripheral(GPIOA, &GPIO_InitStructure);
        //定义MISO
        GPIO_InitStructure.Pin = GPIO_PIN_6;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPU;
        GPIO_InitPeripheral(GPIOA, &GPIO_InitStructure);

        //初始化SPI1
        SPI_InitStructure.BaudRatePres = SPI_BR_PRESCALER_256;
        SPI_InitStructure.CLKPHA = SPI_CLKPHA_FIRST_EDGE;
        SPI_InitStructure.CLKPOL = SPI_CLKPOL_LOW;
        //定义全双工
        SPI_InitStructure.DataDirection = SPI_DIR_DOUBLELINE_FULLDUPLEX;
        //这里先定义 DataLen = 8 供发送CMD使用,后续发送数据的时候可能要调整为 DataLen = 16
        SPI_InitStructure.DataLen = SPI_DATA_SIZE_8BITS;
        SPI_InitStructure.FirstBit = SPI_FB_MSB;
        //定义软件CS(片选)
        SPI_InitStructure.NSS = SPI_NSS_SOFT;
        SPI_InitStructure.SpiMode = SPI_MODE_MASTER;
        SPI_Init(SPI1, &SPI_InitStructure);

        SPI_Enable(SPI1, ENABLE);
        //拉高SC信号使失能从机选择
        Disable_CS();

}


/*
Definition:   脉冲发送模块
Description:  发送连续80个时钟脉冲,用以上电复位
*/
void SPI_Hardware_SendPulse(void){
        Disable_CS();
        for(uint8_t i = 0; i < 10; i++){
                //向SD卡发送无效位0xFF
                while(SPI_I2S_GetStatus(SPI1,SPI_I2S_TE_FLAG) == RESET);
                SPI_I2S_TransmitData(SPI1, DUMMY_BIT);
        }
}


/*
Definition:   (8.22补充机制)用以在发送CMD55初始化指令后面补充1个0xFF脉冲的模块
Description:  所有的脉冲都将在指令发送/接收完成后拉高CS电平后发送
*/
void SPI_Hardware_SendSinglePulse(void){
        while(SPI_I2S_GetStatus(SPI1,SPI_I2S_TE_FLAG) == RESET);
        SPI_I2S_TransmitData(SPI1, DUMMY_BIT);
}
```

```
/*
Definition:    发送/接收单个字节的模块(初始化用,只发送状态下不考虑返回的参数值,只接收发送Dummy Bit)
*/
uint8_t SPI_Hardware_SendGetByte(uint8_t Send_Data){
        //发送DUMMYBIT,诱发时钟
        while(SPI_I2S_GetStatus(SPI1,SPI_I2S_TE_FLAG) == RESET);
        SPI_I2S_TransmitData(SPI1, Send_Data);

        //接收Byte
        while(SPI_I2S_GetStatus(SPI1, SPI_I2S_RNE_FLAG) == RESET);
        uint8_t Byte = SPI_I2S_ReceiveData(SPI1);

        return Byte;
}



/*
Definition:    发送初始化CMD的模块
*/
void SPI_Hardware_SendInitCMD(uint8_t CMD, uint32_t Coefficient, uint8_t CMD_CRC){
        //将参数写入到Datas数组等待发送
        uint8_t Datas[6];
        Datas[0] = CMD;
        Datas[1] = Coefficient >> 24;
        Datas[2] = Coefficient >> 16;
        Datas[3] = Coefficient >> 8;
        Datas[4] = Coefficient;
        Datas[5] = CMD_CRC;
        Send_VarString("Address in SendInitCMD: %x %x %x %x\r\n", Datas[1], Datas[2], Datas[3],
Datas[4]);

        //拉低CS电平,开启通信
        Enable_CS();
        //发送数组
        uint8_t Index = 0;
        // 固定发送6字节CMD帧
        for(Index = 0; Index < 6; Index++){  // 6字节：opcode(1)+参数(4)+CRC(1)
                SPI_Hardware_SendGetByte(Datas[Index]);
        }
}


/*
Definition:    发送CMD24的模块
Notice:        (8.27 更新:)在接收正确响应之后发送0xFF, 0xFE和后续512个字节
               返回类型:参数值(写入是否正常)
*/
uint8_t SPI_Hardware_SendSectorData(uint8_t* Datas){
        uint8_t R1_React = SD_WriteData_Failed;
        //发送数据头
        SPI_Hardware_SendGetByte(Single_Block_Read_Token);
        //
        for(uint16_t i = 0; i < Sector_Length; i++){
                Send_VarString("Sector data = %d, \r\n", Datas[i]);
                SPI_Hardware_SendGetByte(Datas[i]);
        }
        //不接收CRC校验位
        SPI_Hardware_SendGetByte(DUMMY_BIT);
        SPI_Hardware_SendGetByte(DUMMY_BIT);
        //判断数值校验位
        uint8_t Data_React = SPI_Hardware_SendGetByte(DUMMY_BIT);
        //数据响应类型
    if((Data_React & 0x1F) == SD_DATA_OK)
        R1_React = SD_WriteData_Success;

        Send_VarString("Data reaction = %x\r\n", Data_React);
```

```
        //8.29新增: 添加BUSY校验
        uint8_t Is_Busy = 0x00;
        do{
                Is_Busy = SPI_Hardware_SendGetByte(DUMMY_BIT);
        }while(Is_Busy != 0xFF);

        //拉高CS引脚并发送脉冲
        Disable_CS();
        SPI_Hardware_SendSinglePulse();
        return R1_React;
}



/*
Definition:    处理长接收内容的模块
Description:   接受的长度由定义的内容而定
*/
void SPI_Hardware_GetLongReply(uint8_t* Datas, uint8_t Start_Flag, uint8_t CMD){
        uint16_t Returned_Datas = 0;
        uint16_t Index_Coefficient = 0;
        uint8_t Temp_Data = 0;
        uint8_t Enable_Read = 0;

        //基于CMD给出长度
        uint16_t Length;
        if(CMD == CMD24)
                Length = Total_Sector_Length;
        else
                Length = Total_CID_CSD_Length;

        //判断是否结束
        while(Returned_Datas != (Length)){
                //连续读取数值
                Temp_Data = SPI_Hardware_SendGetByte(DUMMY_BIT);
                Send_VarString("Temp_Data = %x\r\n", Temp_Data);
                //判断是否检测到Token
                if(Temp_Data == Start_Flag)
                        Enable_Read = 1;
                else{
                        if(Enable_Read == 1){
                                Index_Coefficient++;
                                Returned_Datas++;
                                Datas[Index_Coefficient] = Temp_Data;
                        }
                }
        }
        Send_VarString("Send Complete\r\n");
}

/*
Definition:    接收指令参数的模块,返回状态或者参数,经由指针传递数组数据
Description:   依照输入的指令返回指定参数
CMD0: R1响应,仅返回单个Byte
CMD8(R3), CMD58(R7): 返回单个Byte和32位状态位(回声)
CMD9, CMD10: 返回R1和20个字节的CID/CSD参数(含Token,16位CID/CSD,CRC16和CRC7)
*/
void SPI_Hardware_GetInitData(uint8_t CMD_Command, uint8_t *Datas_In, uint8_t* Datas_Out){
        uint8_t Returned_Datas = 0;
        uint8_t Temp_Data = 0;
        uint8_t Index_Coefficient = 0;

        //超时机制
        uint16_t Timeout = 10000;

        //循环检测,直至获取有效的R1响应
        while(1){
            Temp_Data = SPI_Hardware_SendGetByte(DUMMY_BIT);
                Send_VarString("Temp data in GetInitData = %d\r\n", Temp_Data);
```

```
                if(!(Temp_Data & 0x80)){
                    Datas_In[Index_Coefficient] = Temp_Data;
                        Send_VarString("Index %d, Get Init Data: R1 = %x\r\n", Index_Coefficient,
Datas_In[Index_Coefficient]);
                        break;
                }
                else{
                        --Timeout;
                        if(!Timeout)
                                break;
                }
        }
        //判断超时或者无效指令
        if(Datas_In[Index_Coefficient] != Card_Ready && Datas_In[Index_Coefficient] !=
In_Idle_State){
                Send_VarString("The command encounters an error \r\n");
                Send_VarString("Error Type: %x \r\n", *Datas_In);
        }
        else if(!(Timeout)){
                Send_VarString("Time exceeded \r\n");
                Datas_In[Index_Coefficient] = Invalid;
        }
        else{
                while(1){
                        //包括初始化,设置扇区,ACMD41配套
                        if(CMD_Command == CMD0 || CMD_Command == CMD16|| CMD_Command == CMD55 ||
CMD_Command == ACMD41){
                                break;
                        }
                        //写入扇区
                        else if(CMD_Command == CMD24){
                                uint8_t Get_WriteStatus = SPI_Hardware_SendSectorData(Datas_In);
                                if(Get_WriteStatus == SD_WriteData_Success)
                                        Send_VarString("Data has been delivered successfully\r\n");
                                else
                                        Send_VarString("The system encountered an error in writing
data\r\n");
                                break;
                        }
                        //特殊数值
                        else if (CMD_Command == CMD8 || CMD_Command == CMD58){
                                if(Returned_Datas == 5){
                                        SPI_Hardware_SendGetByte(DUMMY_BIT);
                                        break;
                                }
                                else{
                                        Index_Coefficient++;
                                        Datas_In[Index_Coefficient] =
SPI_Hardware_SendGetByte(DUMMY_BIT);
                                        Returned_Datas++;
                                }
                        }
                        //CID/CSD/读取扇区
                        else if(CMD_Command == CMD9 || CMD_Command == CMD10 || CMD_Command == CMD17)
{
                                SPI_Hardware_GetLongReply(Datas_In, Single_Block_Read_Token,
CMD_Command);
                                break;
                        }
                        //无其他指令,退出
                        else
                                break;
                }
        }
        //拉高CS电平,关闭通信
        if(CMD_Command != CMD55){
                Disable_CS();
        }
        SPI_Hardware_SendSinglePulse();
```

```
}


/*
Description: 专供CMD55->ACMD41流程的读写函数
*/
uint8_t SPI_Hardware_SendACMD(uint8_t SD_Card_Type){
        uint8_t CMD55_R1_Reaction;
        uint8_t ACMD41_R1_Reaction = 0x80;
        /*将ACMD41_R1_Reaction设置为0x80,除非正常输出否则输出失败标志位*/
        /*发送CMD55*/
        SPI_Hardware_SendInitCMD(CMD55, CMD55_Index, OTHER_CSC);
        SPI_Hardware_GetInitData(CMD55, &CMD55_R1_Reaction, NULL);
        if(CMD55_R1_Reaction != In_Idle_State){
                return Invalid;
        }
        else{
                if(SD_Card_Type == 1){
                        SPI_Hardware_SendInitCMD(ACMD41, ACMD41_Index_H, OTHER_CSC);
                        SPI_Hardware_GetInitData(ACMD41, &ACMD41_R1_Reaction, NULL);
                }
                else{
                        SPI_Hardware_SendInitCMD(ACMD41, ACMD41_Index_L, OTHER_CSC);
                        SPI_Hardware_GetInitData(ACMD41, &ACMD41_R1_Reaction, NULL);
                }
        }
        Send_VarString("%d\r\n",ACMD41_R1_Reaction);
        return ACMD41_R1_Reaction;
}


/*
Description:  用以实现CMD指令的输出
*/
void SPI_Hardware_SendCMD(uint8_t CMD, uint32_t Coefficient, uint8_t CMD_CRC, uint8_t *Datas_OUT,
uint8_t* Datas_IN){
        //读取指令
        SPI_Hardware_SendInitCMD(CMD, Coefficient, CMD_CRC);
        //发送参数
        SPI_Hardware_GetInitData(CMD, Datas_OUT, Datas_IN);
}
```