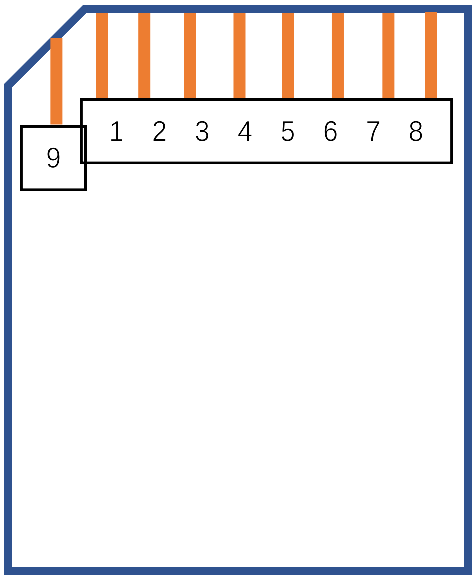
Part 5 SD Card

首先介绍 SD Card 的 2 个模式和对应的 9 个引脚的功能

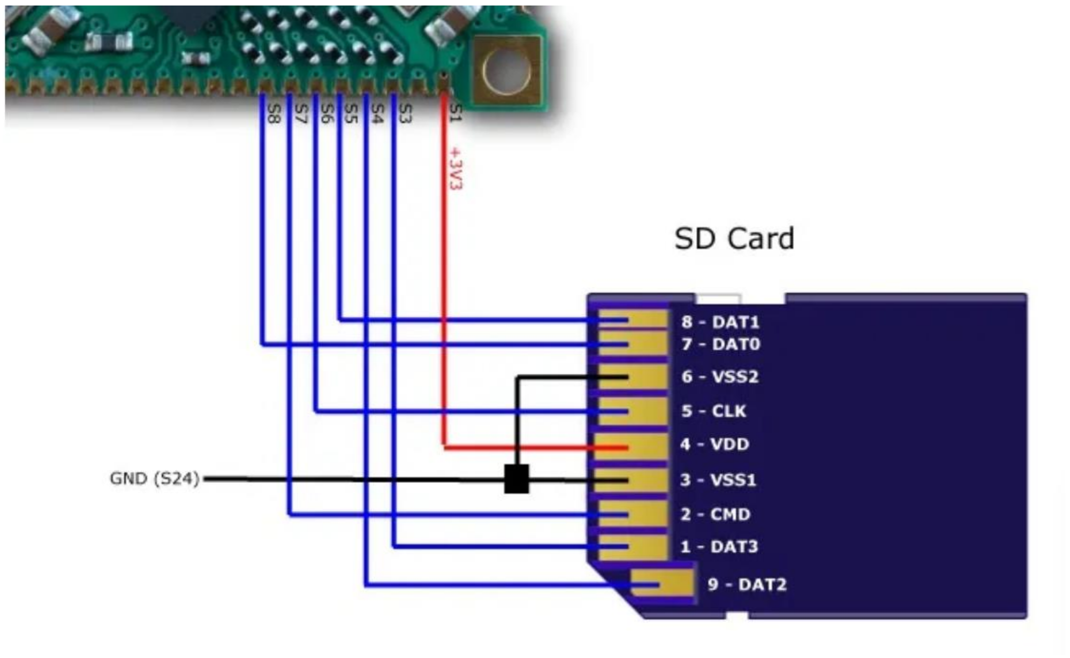


模式 1 SDIO（Secure Digital Input and Output 安全数字输入输出接口）模式

其中数据传输模式也存在两种为 4bits 和 1bit，若使用 4bits 方式传输数据，则 DAT0-3 分别使用 7，8，9，1 接口，若使用 1bits 方式传输数据，则 DAT0 使用 7 接口

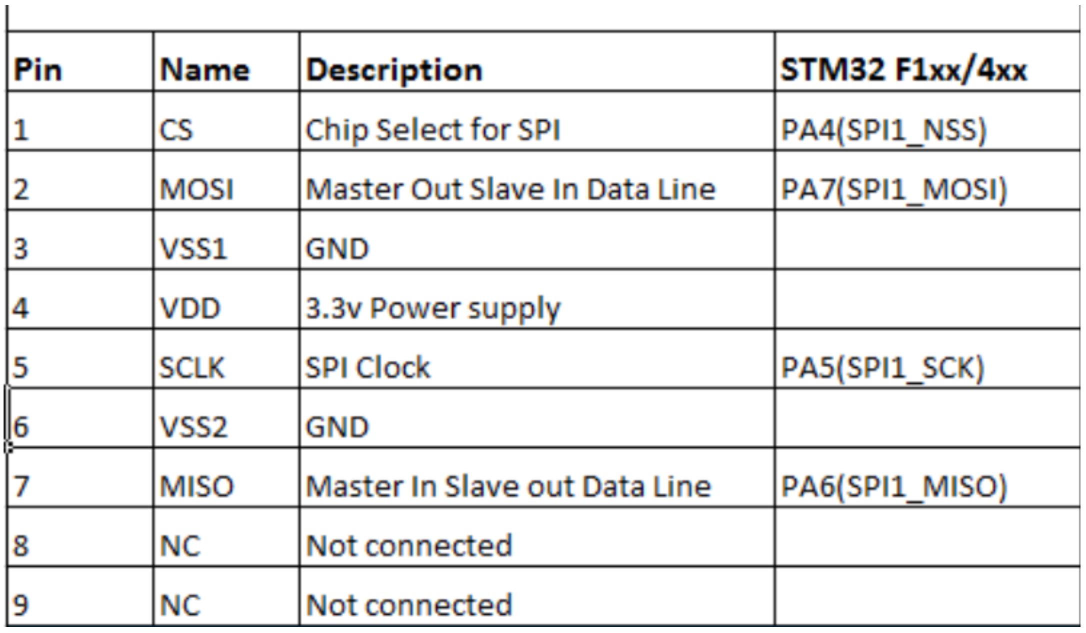
			STM32F1xx/4xx	
Pin	**Name**	**Description**	**4-bits Data**	**1-bit Data**
1	CD/DAT3	Data line3	PC11	
2	CMD	Command/Response Line	PD2	PD2
3	VSS1	GND		
4	VDD	3.3V Power supply		
5	CLK	Clock	PC12	PC12
6	VSS2	GND		
7	DAT0	Data line0	PC8	PC8
8	DAT1	Data line1	PC9	
9	DAT2	Data line2	PC10	

下图为 SD 卡与单片机的连接方式



模式 2 SPI（Serial Peripheral Interface 串行外围接口）模式

这里只用到单片机的 4 根信号线



Pin	Name	Description	STM32 F1xx/4xx
1	CS	Chip Select for SPI	PA4(SPI1_NSS)
2	MOSI	Master Out Slave In Data Line	PA7(SPI1_MOSI)
3	VSS1	GND	
4	VDD	3.3v Power supply	
5	SCLK	SPI Clock	PA5(SPI1_SCK)
6	VSS2	GND	
7	MISO	Master In Slave out Data Line	PA6(SPI1_MISO)
8	NC	Not connected	
9	NC	Not connected	

SDI 中的特殊寄存器

SDI Control Register (SDICON)

Register	Address	R/W	Description	Reset Value
SDICON	0x5A000000	R/W	SDI control register	0x0

SDICON	Bit	Description	Initial Value
Reserved	[31:9]	–	
SDMMC Reset (SDreset)	[8]	Reset whole sdmmc block. This bit is automatically cleared. 0 = Normal mode, 1 = SDMMC reset	0
Reserved	[7:6]		0
Clock Type (CTYP)	[5]	Determines which clock type is used as SDCLK. 0 = SD type, 1 = MMC type	0
Byte Order Type(ByteOrder)	[4]	Determines byte order type when you read(write) data from(to) sd host FIFO with word boundary. 0 = Type A, 1 = Type B	0
Receive SDIO Interrupt from card (RcvIOInt)	[3]	Determines whether sd host receives SDIO Interrupt from the card or not(for SDIO). 0 = Ignore, 1 = Receive SDIO Interrupt	0
Read Wait Enable(RWaitEn)	[2]	Determines read wait request signal generate when sd host waits the next block in multiple block read mode. This bit needs to delay the next block to be transmitted from the card(for SDIO). 0 = Disable(no generate), 1 = Read wait enable(use SDIO)	0
Reserved	[1]		
Clock Out Enable (ENCLK)	[0]	Determines whether SDCLK Out enable or not 0 = Disable (prescaler off), 1 = Clock enable	0

SDI Command Argument Register (SDICmdArg)

Register	Address	R/W	Description	Reset Value
SDICmdArg	0x5A000008	R/W	SDI command argument register	0x0

SDICmdArg	Bit	Description	Initial Value
CmdArg	[31:0]	Command argument	0x00000000

SDI Data Register (SDIDAT)

Register	Address	R/W	Description	Reset Value
SDIDAT	0x5A000040, 44, 48, 4C(Li/W, Li/HW, Li/B, Bi/W) 0x5A000041(Bi/HW), 0x5A000043(Bi/B)	R/W	SDI data register	0x0

SDIDAT	Bit	Description	Initial State
Data Register	[31:0]	This field contains the data to be transmitted or received over the SDI channel	0x00000000

代码部分解读：

程序主要使用 SPI 模式进行初始化，读写 SD 卡，有关报警信号的精确信息。

这里程序流程比较简单：
1）配置串口，用作程序的调试输出
2）填充将要给 SD 卡写入数据的数组 send_data（利用 Part7 的 RTC 日期时间子程序）。
3）初始化 SD 卡，根据返回 SD_Init()返回值确定 SD 卡初始化是否完成。
4）进行单块读写

主函数的代码如下：

```
int main(void)
{
    u16 i;
    USART1_Config();
    send_data[1536] = RTC.get_time()
    switch (SD_Init())
    {
    case 0:
        USART1_Puts("\r\nSD Card Init Success!\r\n");
        break;
    case 1:
        USART1_Puts("Time Out!\n");
        break;
    case 99:
        USART1_Puts("No Card!\n");
        break;
    default: USART1_Puts("unknown err\n");
        break;
    }
    SD_WriteSingleBlock(30, send_data);
    SD_ReadSingleBlock(30, receive_data);
    if (Buffercmp(send_data, receive_data, 512))
    {
        USART1_Puts("\r\n single read and write success \r\n");
        //USART1_Puts(receive_data);
    }
    while (1);
}
```

初始化 SD 卡的函数

```
u8 SD_Init(void)
{
```

```
u16 i;
u8 r1;
u16 retry;
u8 buff[6];
SPI_ControlLine();
//SD卡初始化时时钟不能超过400KHz
SPI_SetSpeed(SPI_SPEED_LOW);
//CS为低电平，片选置低，选中SD卡
SD_CS_ENABLE();
//纯延时，等待SD卡上电稳定
for (i = 0; i < 0xf00; i++);
//先产生至少74个脉冲，让SD卡初始化完成
for (i = 0; i < 10; i++)
{
    //参数可随便写，经过10次循环，产生80个脉冲
    SPI_ReadWriteByte(0xff);
}
//-----------------SD卡复位到idle状态----------------
//循环发送CMD0，直到SD卡返回0x01,进入idle状态
//超时则直接退出
retry = 0;
do
{
    //发送CMD0，CRC为0x95
    r1 = SD_SendCommand(CMD0, 0, 0x95);
    retry++;
} while ((r1 != 0x01) && (retry < 200));
//跳出循环后，检查跳出原因，
if (retry == 200) //说明已超时
{
    return 1;
}
//如果未超时，说明SD卡复位到idle结束
//发送CMD8命令，获取SD卡的版本信息
r1 = SD_SendCommand(CMD8, 0x1aa, 0x87);
//下面是SD2.0卡的初始化
if (r1 == 0x01)
{
    // V2.0的卡，CMD8命令后会传回4字节的数据，要跳过再结束本命令
    buff[0] = SPI_ReadWriteByte(0xFF);
    buff[1] = SPI_ReadWriteByte(0xFF);
    buff[2] = SPI_ReadWriteByte(0xFF);
    buff[3] = SPI_ReadWriteByte(0xFF);
    SD_CS_DISABLE();
```

```c
        //多发8个时钟
        SPI_ReadWriteByte(0xFF);
        retry = 0;
        //发卡初始化指令CMD55+ACMD41
        do
        {
            r1 = SD_SendCommand(CMD55, 0, 0);
            //应返回0x01
            if (r1 != 0x01)
                return r1;
            r1 = SD_SendCommand(ACMD41, 0x40000000, 1);
            retry++;
            if (retry > 200)
                return r1;
        } while (r1 != 0);
        //初始化指令发送完成，接下来获取OCR信息
        //-----------鉴别SD2.0卡版本开始-----------
        //读OCR指令
        r1 = SD_SendCommand_NoDeassert(CMD58, 0, 0);
        //如果命令没有返回正确应答，直接退出，返回应答
        if (r1 != 0x00)
            return r1;
        //应答正确后，会回传4字节OCR信息
        buff[0] = SPI_ReadWriteByte(0xFF);
        buff[1] = SPI_ReadWriteByte(0xFF);
        buff[2] = SPI_ReadWriteByte(0xFF);
        buff[3] = SPI_ReadWriteByte(0xFF);
        //OCR接收完成，片选置高
        SD_CS_DISABLE();
        SPI_ReadWriteByte(0xFF);
        //检查接收到的OCR中的bit30位（CSS），确定其为SD2.0还是SDHC
        //CCS=1：SDHC   CCS=0：SD2.0
        if (buff[0] & 0x40)
        {
            SD_Type = SD_TYPE_V2HC;
        }
        else
        {
            SD_Type = SD_TYPE_V2;
        }
        //-----------鉴别SD2.0卡版本结束-----------
        SPI_SetSpeed(1);       //设置SPI为高速模式
    }
}
```

读写数据的函数

```
u8  SPI_ReadWriteByte(u8 TxData)
{
    while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
    SPI_I2S_SendData(SPI1, TxData);
    while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) == RESET);
    return SPI_I2S_ReceiveData(SPI1);
}
```