sd 卡读写实验

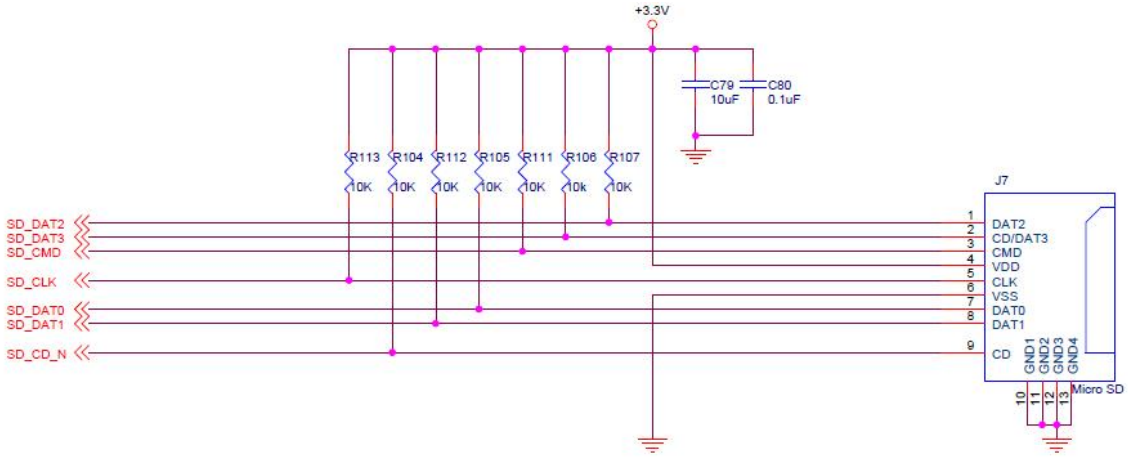
1 实验简介

SD 卡是现在嵌入式设备重要的存储模块，内部集成了 nand flash 控制器，方便了主机的的管理。本实验主要是练习对 sd 卡的扇区进行读写，通常 sd 卡都有文件系统，可以按照文件名和目录路径来读写文件，但文件系统非常复杂，本实验不做讲解，在后续的实验中我们通过搜索特定的文件头来读特殊的文件，完成音频播放、图片读取显示等。

2 实验原理

2.1 硬件描述

开发板上装有一个 Micro SD 卡座，FPGA 通过 SPI 数据总线访问 Micro SD 卡, SD 卡座和 FPGA 的硬件电路连接如下：



开发板 SD 卡电路

在 SD 卡数据读写速度要求不高的情况下，选用 SPI 通信模式可以说是一种最佳的解决方案。因为在 SPI 模式下，通过四条线就可以完成所有的数据交换。本实验将为大家介绍 FPGA 通过 SPI 总线读写 SD 卡。要完成 SD 卡的 FPGA 读写，用户需要理解 SD 卡的命令协议。

2.2 SD 卡协议简介

SD 卡的协议是一种简单的命令/响应的协议。全部命令由主机发起，SD 卡接收到命令后并返回响应数据。根据命令的不同，返回的数据内容和长度也不同。SD 卡命令是一个 6 字节组成的命令包，其中第一个字节为命令号， 命令号高位 bit7 和 bit6 为固定的“01 “，其它 6 个 bit 为具体的命令号。第 2 个字节到第 5 个字节为命令参数。第 6 个字节为 7 个 bit 的 CRC 校验加 1 个 bit 的结束位。***如果在 SPI 模式的时候，CRC 校验位为可选***。如下图所示，Command 表示命令，通常使用十进制表示名称，例如 CMD17，这个时候 Command 就是十进制的 17。SD 卡具体的协议本实验不讲解，可自行找相关资料学习。

First Byte			Bytes 2-5	Last Byte	
0	1	Command	Argument (MSB First)	CRC	1

SD Command Format

SD 卡对每个命令会返回一个响应，每个命令有一定的响应格式。响应的格式跟给它的命令号有关。在 SPI 模式中，有三种响应格式：R1，R2，R3。

Byte	Bit	Meaning
1	7	Start Bit, Always 0
	6	Parameter Error
	5	Address Error
	4	Erase Sequence Error
	3	CRC Error
	2	Illegal Command
	1	Erase Reset
	0	In Idle State

Response type R1

Byte	Bit	Meaning
1	7	Start Bit, Always 0
	6	Parameter Error
	5	Address Error
	4	Erase Sequence Error
	3	CRC Error
	2	Illegal Command
	1	Erase Reset
	0	In Idle State
2	7	Out of Range, CSD Overwrite
	6	Erase Parameter
	5	Write Protect Violation
	4	Card ECC Failed
	3	Card Controller Error
	2	Unspecified Error
	1	Write Protect Erase Skip, Lock/Unlock Failed
	0	Card Locked

Response type R2

Byte	Bit	Meaning
1	7	Start Bit, Always 0
	6	Parameter Error
	5	Address Error
	4	Erase Sequence Error
	3	CRC Error
	2	Illegal Command
	1	Erase Reset
	0	In Idle State
2-5	All	Operating Condition Register, MSB First

Response type R3

2.2.1 SD 卡 2.0 版的初始化步骤

① 上电后延时至少 74clock，等待 SD 卡内部操作完成

② 片选 CS 低电平选中 SD 卡

③ 发送 CMD0，需要返回 0x01，进入 Idle 状态

④ 为了区别 SD 卡是 2.0 还是 1.0，或是 MMC 卡，这里根据协议向上兼容的，首先发送只有 SD2.0 才有的命令 CMD8，如果 CMD8 返回无错误，则初步判断为 2.0 卡，进一步循环发送命令 CMD55+ACMD41，直到返回 0x00，确定 SD2.0 卡

⑤ 如果 CMD8 返回错误则判断为 1.0 卡还是 MMC 卡，循环发送 CMD55+ACMD41，返回无错误，则为 SD1.0 卡，到此 SD1.0 卡初始成功，如果在一定的循环次数下，返回为错误，则进一步发送 CMD1 进行初始化，如果返回无错误，则确定为 MMC 卡，如果在一定的次数下，返回为错误，则不能识别该卡，初始化结束。（通过 CMD16 可以改变 SD 卡一次性读写的长度）

⑥ CS 拉高

2.2.2 SD 卡的读步骤

① 发送 CMD17（单块）或 CMD18（多块）读命令，返回 0X00

② 接收数据开始令牌 fe（或 fc）+正式数据 512Bytes + CRC 校验 2Bytes

默认正式传输的数据长度是 512Bytes

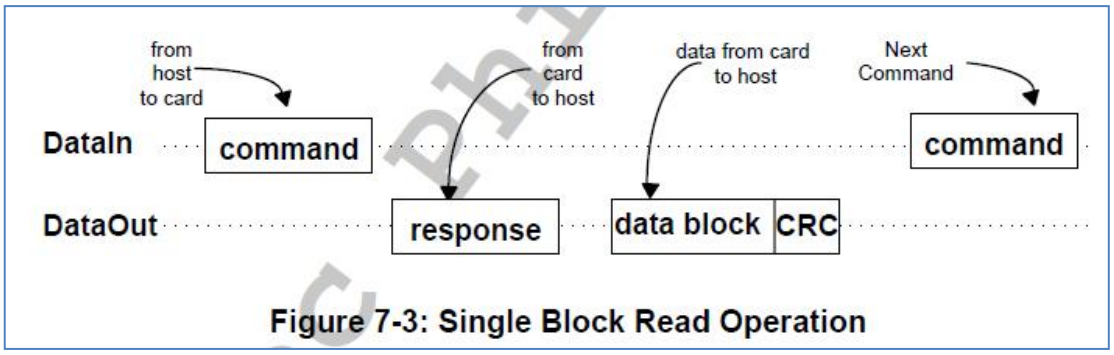


Figure 7-3: Single Block Read Operation

2.2.3 SD 卡的写步骤

① 发送 CMD24（单块）或 CMD25（多块）写命令，返回 0X00

② 发送数据开始令牌 fe（或 fc）+正式数据 512Bytes + CRC 校验 2Bytes

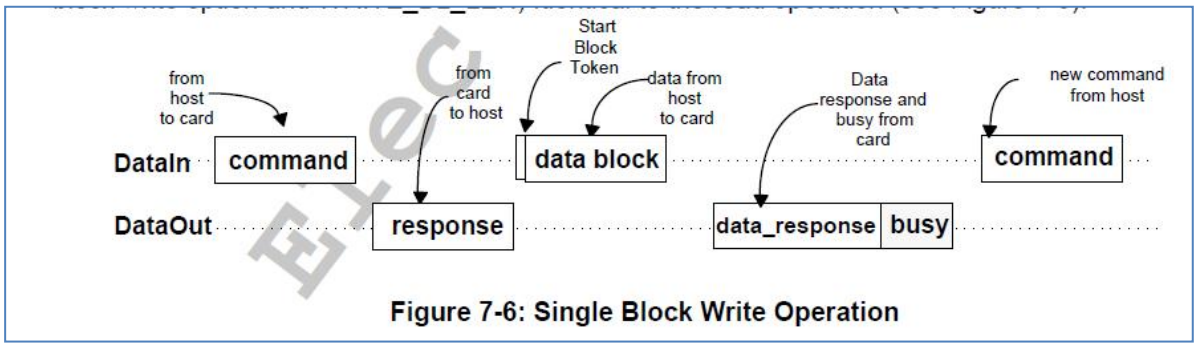
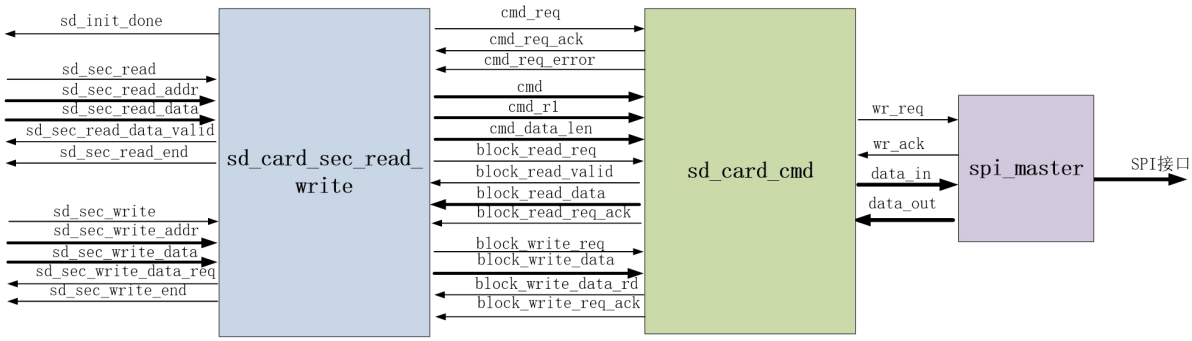


Figure 7-6: Single Block Write Operation

3 程序设计

下面主要对 sd_card_top 及其子程序进行介绍和说明。sd_card_top 包含 3 个子程序，分别为 sd_card_sec_read_write.v，sd_card_cmd.v 和 spi_master.v 文件。它们的逻辑关系如下图所示：



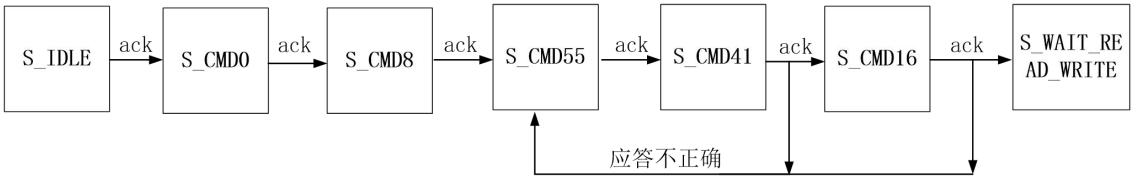
3.1 sd_card_sec_read_write

以下为 sd_card_sec_read_write 模块端口说明：

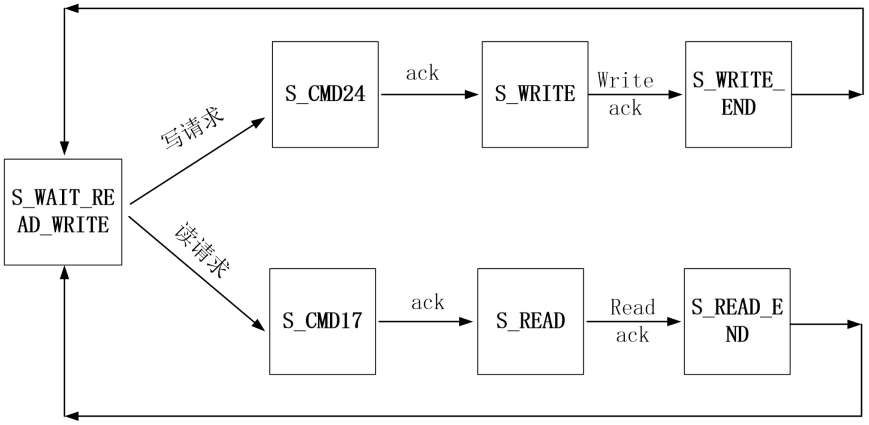
信号名称	方向	说明
clk	in	时钟输入
rst	in	异步复位输入，高复位
sd_init_done	out	sd 卡初始化完成
sd_sec_read	in	sd 卡扇区读请求
sd_sec_read_addr	in	sd 卡扇区读地址
sd_sec_read_data	out	sd 卡扇区读出的数据
sd_sec_read_data_valid	out	sd 卡扇区读出的数据有效
sd_sec_read_end	out	sd 卡扇区读完成
sd_sec_write	in	sd 卡扇区写请求
sd_sec_write_addr	in	sd 卡扇区写请求应答
sd_sec_write_data	in	sd 卡扇区写请求数据
sd_sec_write_data_req	out	sd 卡扇区写请求数据读取，提前 sd_sec_write_data 一个时钟周期
sd_sec_write_end	out	sd 卡扇区写请求完成
spi_clk_div	in	SPI 时钟分频，SPI 时钟频率=系统时钟/（(spi_clk_div + 2）*2）
cmd_req	in	sd 卡命令请求
cmd_req_ack	out	sd 卡命令请求应答
cmd_req_error	out	sd 卡命令请求错误
cmd	in	sd 卡命令，命令+参数+CRC，一共 48bit
cmd_r1	in	sd 卡命令期待的 R1 响应
cmd_data_len	in	sd 卡命令后读取的数据长度，大部分命令没有读取数据
block_read_req	in	块数据读取请求
block_read_valid	out	块数据读取数据有效
block_read_data	out	块数据读取数据

block_read_req_ack	out	块数据读取请求应答
block_write_req	in	块数据写请求
block_write_data	in	块数据写数据
block_write_data_rd	out	块数据写数据请求，提前 block_write_data 一个时钟周期
block_write_req_ack	out	块数据写请求应答

sd_card_sec_read_write 模块有一个状态机，首先完成 SD 卡初始化，下图为模块的初始化状态机转换图，首先发送 CMD0 命令，然后发送 CMD8 命令，再发送 CMD55，接着发送 ACMD41 和 CMD16。如果应答正常，sd 卡初始化完成，等待 SD 卡扇区的读写命令。



然后等待扇区读写指令，并完成扇区的读写操作，下图为模块的读写状态机转换图。



在此模块中定义了两个参数，SD 卡的初始化过程是需要先用慢时钟来发送命令和配置，等待初始化成功后再用快时钟来进行数据读写。

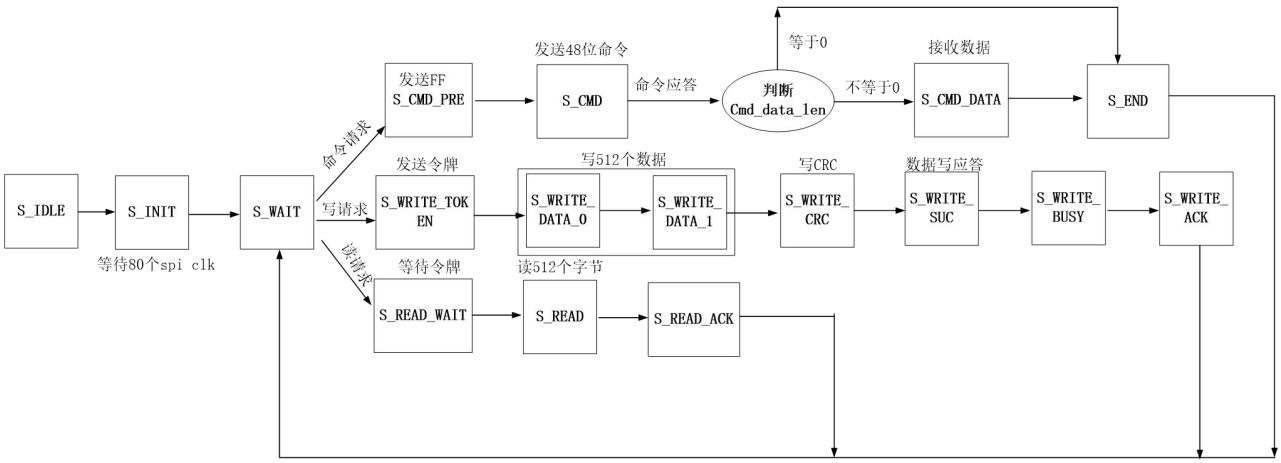
```
        parameter  SPI_LOW_SPEED_DIV = 248

        parameter  SPI_HIGH_SPEED_DIV = 0
```

3.2 sd_card_cmd

sd_card_cmd 模块端口的说明如下：

信号名称	方向	说明
sys_clk	in	时钟输入
rst	in	异步复位输入，高复位
spi_clk_div	in	SPI 时钟分频，SPI 时钟频率=系统时钟/（(spi_clk_div + 2）*2）
cmd_req	in	sd 卡命令请求
cmd_req_ack	out	sd 卡命令请求应答
cmd_req_error	out	sd 卡命令请求错误
cmd	in	sd 卡命令，命令+参数+CRC，一共 48bit
cmd_r1	in	sd 卡命令期待的 R1 响应
cmd_data_len	in	sd 卡命令后读取的数据长度，大部分命令没有读取数据
block_read_req	in	块数据读取请求
block_read_valid	out	块数据读取数据有效
block_read_data	out	块数据读取数据
block_read_req_ack	out	块数据读取请求应答
block_write_req	in	块数据写请求
block_write_data	in	块数据写数据
block_write_data_rd	out	块数据写数据请求，提前 block_write_data 一个时钟周期
block_write_req_ack	out	块数据写请求应答
nCS_ctrl	out	到 SPI master 控制器，cs 片选控制
clk_div	out	到 SPI Master 控制器，时钟分频参数
spi_wr_req	out	到 SPI Master 控制器，写一个字节请求
spi_wr_ack	in	来自 SPI Master 控制器，写请求应答
spi_data_in	out	到 SPI Master 控制器，写数据
spi_data_out	in	来自 SPI Master 控制器，读数据

sd_card_cmd 模块主要实现 sd 卡基本命令操作，还有上电初始化的 88 个周期的时钟，数据块的命令和读写的状态机如下。



从 SD2.0 的标准里我们可以看到，从主控设备写命令到 SD 卡, 最高两位 47~46 位必须为“01”，代表命令发送开始。

4.7.2 Command Format

All commands have a fixed code length of 48 bits, needing a transmission time of 1.92 µs @ 25 MHz and 0.96 µs @ 50 MHz.

Bit position	47	46	[45:40]	[39:8]	[7:1]	0
Width (bits)	1	1	6	32	7	1
Value	'0'	'1'	x	x	x	'1'
Description	start bit	transmission bit	command index	argument	CRC7	end bit

Table 4-17: Command Format

所以代码中都是将 48 位命令的高八位与十六进制 0x40 做或操作得到的结果再写入，所以才有了如下一段代码：

```
190        spi_wr_req <= 1'b1;
191        CS_reg <= 1'b0;
192        if(byte_cnt == 16'd0)
193            send_data <= (cmd[47:40] | 8'h40);
194        else if(byte_cnt == 16'd1)
195            send_data <= cmd[39:32];
196        else if(byte_cnt == 16'd2)
```

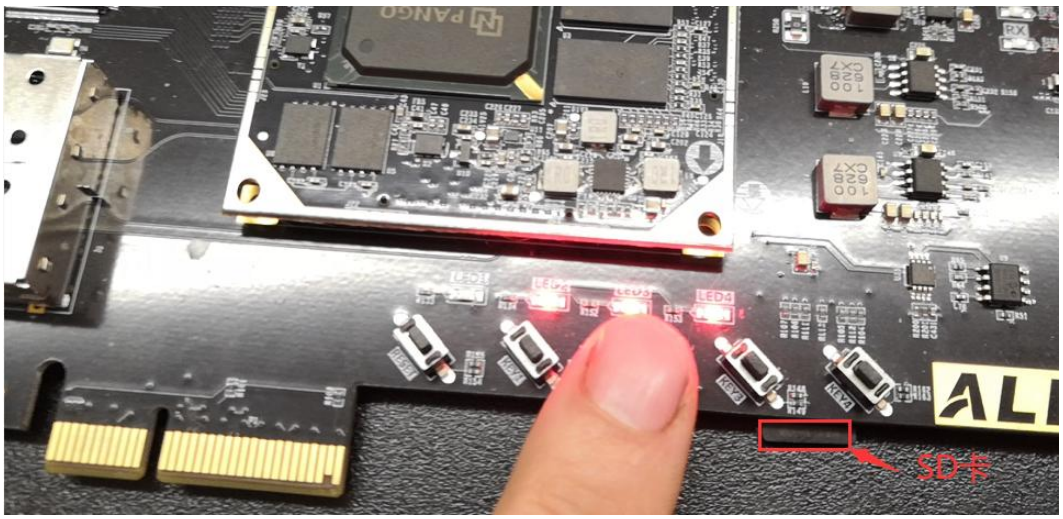
3.3 spi_master

spi_master 模块主要完成 SPI 一个字节的读写，当 SPI 状态机在 idle 的时候，检测到 wr_req 的信号为高，会产生 8 个 DCLK，并把 datain 的数据从高位依次输出到 MOSI 信号线上。MOSI 在 8 个 DCLK 的输出数据为 datain 的值 0x58。

同时 spi_master 程序也会读取 MISO 输入的数据，转换成 8 位的 data_out 数据输出实现 SPI 的一个字节的数据读取。

4 实验现象

下载实验程序后，可以看到 LED 显示一个二进制数字，这个数字是存储在 sd 卡中第一扇区的第一个数据，数据是随机的，这个时候按键 KEY2 按下，数字加一，并写入了 sd 卡，再次下载程序，可以看到直接显示更新后的数据。



开发板操作