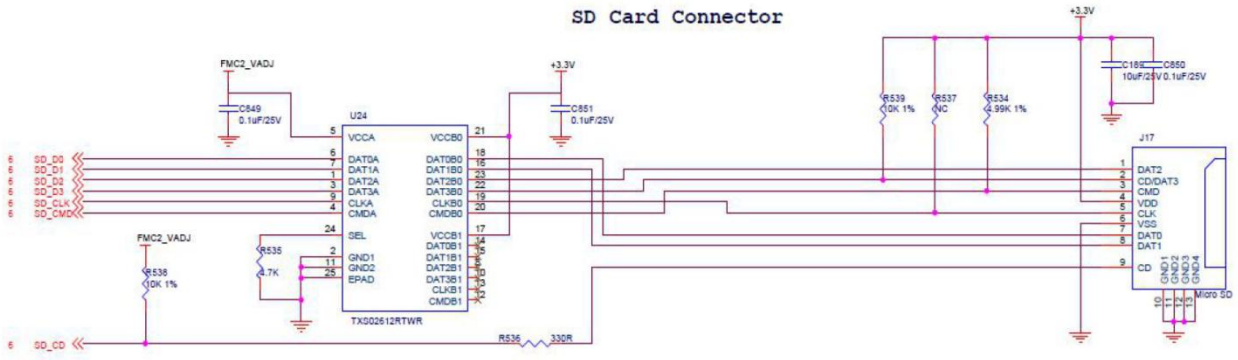
1 Experiment Introduction

The SD card is an important storage module for embedded devices, and the “nand flash” controller is integrated inside to facilitate the management of the host. This experiment is mainly to practice reading and writing the sectors of the sd card. Usually the sd card has a file system. The file can be read and written according to the file name and directory path, but the file system is very complicated. This experiment does not explain, in the subsequent experiments. In our search for a specific file by searching for a specific file header, we complete audio playback, image read display, and so on.

2 Experiment Principle

2.1 Hardware Description

The AXKU040 FPGA development board is equipped with a “Micro SD” card holder. The FPGA accesses the “Micro SD” card through the “SPI” data bus. The hardware circuit connection between the “SD card holder” and the FPGA is as follows:



SD Card on the AXKU040 FPGA Development Board

In the case of SD card data read and write speed requirements are not high, the choice of SPI communication mode can be said to be an optimal solution. Because in SPI mode, all data exchanged by four lines. This lab introduces the FPGA to read and write SD cards through the SPI bus. To complete the FPGA read and write of the SD card, the user needs to understand the command protocol of the SD card Command Protocol.

2.2 SD card protocol Introduction

The SD card protocol is a simple command/response protocol. All commands are initiated by the host, and the SD card receives the command and returns the response data. The returned data content and length are different depending on the command. The SD card command is a 6-byte command packet. The first byte is the command number. The high-order bits 7 and 6 of the command number are fixed "01". The other 6 bits are specific command numbers. The 2nd byte to the 5th byte are command parameters. The sixth byte is a 7-bit CRC check plus an end bit of 1 bit. ***The CRC check bit is optional if in SPI mode.*** As shown in the following figure, Command represents a command, usually in decimal notation, such as CMD17, when Command is decimal 17. The specific agreement of the SD card is not explained in this experiment. You can find relevant information on your own.

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

SD Command Format

The SD card returns a response for each command, and each command has a certain response format. The format of the response is related to the command number given to it. In SPI mode, there are three response formats: ***R1, R2, and 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 card version 2.0 initialization steps

① Delay at least 74clock after power-on, waiting for the internal operation of the SD card to be completed.
② Chip select CS low level to select SD card
③ Send CMD0, need to return 0x01, enter the Idle state
④ In order to distinguish whether the "SD" card is 2.0 or 1.0, or an "MMC" card, here is upward compatible according to the protocol. First, the command "CMD8" which only has "SD2.0" is sent. If the "CMD8" returns no error, the initial judgment is 2.0 card, and the command "CMD55+ACMD41" is further cyclically sent, until returning "0x00", determine "SD2.0" card
⑤ If "CMD8" returns an error, it is judged as 1.0 card or "MMC" card, and "CMD55+ACMD41" is sent cyclically. If there is no error, it is "SD1.0" card. The SD1.0 card is initially successful. If it is a certain number of cycles, it will return an error. Then, CMD1 is further sent for initialization. If no error is returned, it is determined as an MMC card. If the error is returned after a certain number of times, the card cannot be recognized, and the initialization ends. (CMD16 can change the length of SD card read and write once)
⑥ CS set to high level

2.2.2 SD card reading steps

① Send CMD17 (single block) or CMD18 (multiple block) read command, return 0X00
② Receive data start token "fe" (or "fc") + official data "512Bytes" + "CRC" check 2Bytes
The default officially transmitted data length is 512Bytes.

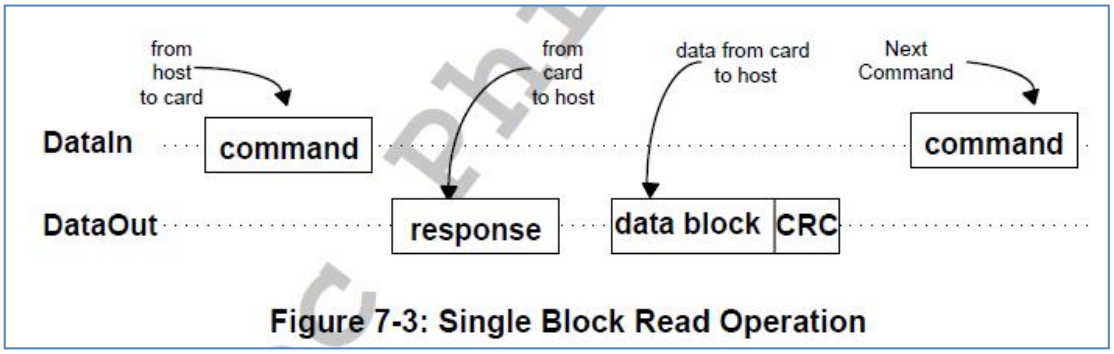


Figure 7-3: Single Block Read Operation

2.2.3 SD card writing steps

① Send CMD24 (single block) or CMD25 (multiple block) write command, return 0X00
② 2Bytes Send data start token “fe” (or “fc”) + official data 512Bytes + CRC check 2Bytes

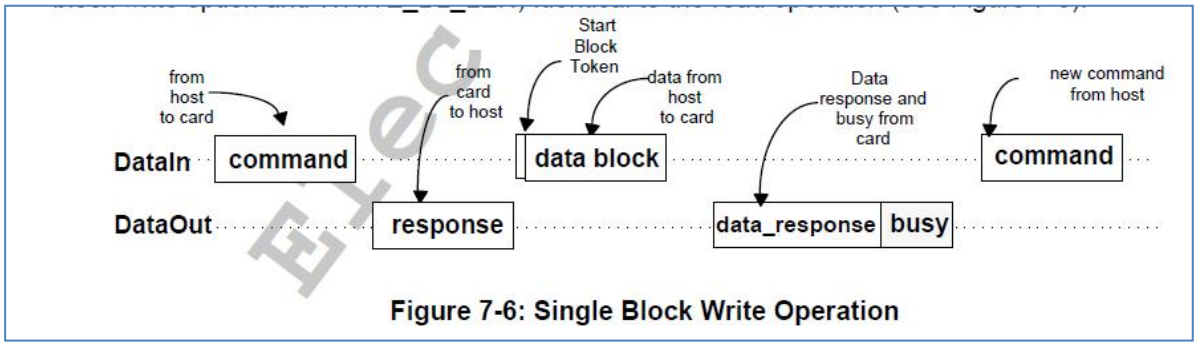
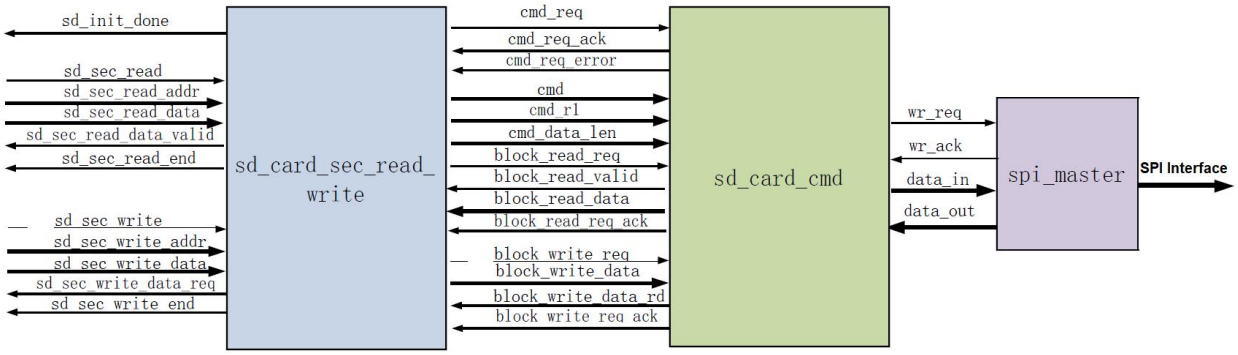


Figure 7-6: Single Block Write Operation

3 Programming

The following mainly introduces and explains “sd_card_top” and its subroutines. “Sd_card_top” contains 3 subroutines, “sd_card_sec_read_write.v”, “sd_card_cmd.v” and “spi_master.v” files. Their logical relationship is shown in the following figure



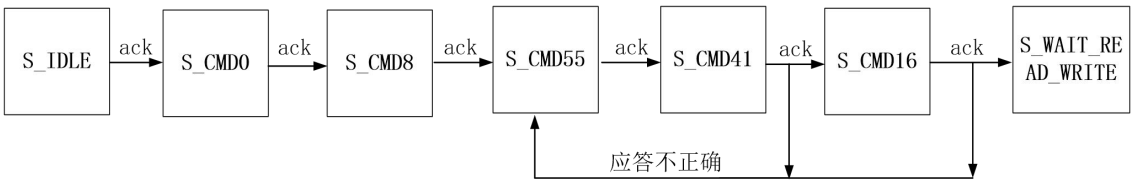
3.1 sd_card_sec_read_write

The following is the port description of the “sd_card_sec_read_write” module:

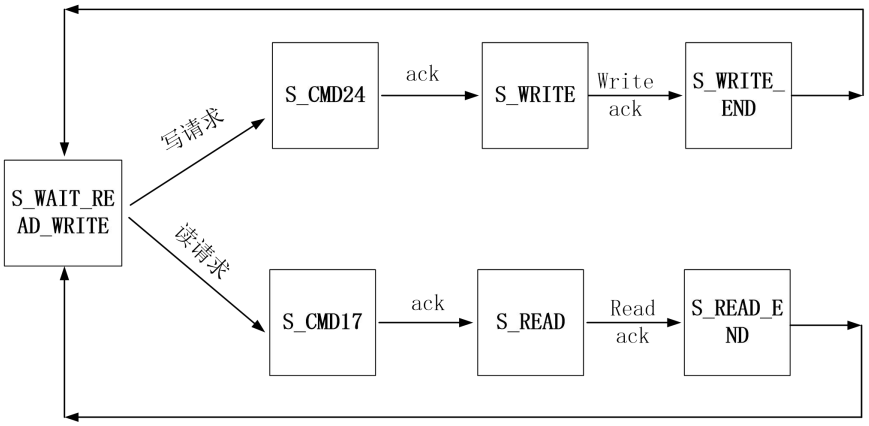
Signal Name	Direction	Description
clk	in	Clock input
rst	in	Asynchronous reset input, high reset
sd_init_done	out	Sd card initialization completed
sd_sec_read	in	Sd card sector read request
sd_sec_read_addr	in	Sd card sector read address
sd_sec_read_data	out	Sd card sector read data
sd_sec_read_data_valid	out	The data read by the sd card sector is valid.

sd_sec_read_end	out	Sd card sector read completed
sd_sec_write	in	Sd card sector write request
sd_sec_write_addr	in	Sd card sector write request respons
sd_sec_write_data	in	Sd card sector write request data
sd_sec_write_data_req	out	Sd card sector write request data read, advance "sd_sec_write_data" one clock cycle
sd_sec_write_end	out	Sd card sector write request completed
spi_clk_div	in	SPI clock division, SPI clock frequency = system clock / ((spi_clk_div + 2) * 2)
cmd_req	in	Sd card command request
cmd_req_ack	out	Sd card command request response
cmd_req_error	out	Sd card command request error
cmd	in	Sd card command, command + parameter + CRC, a total of 48bit
cmd_r1	in	The sd card command expects the R1 response
cmd_data_len	in	The length of the data read after the sd card command, most of the commands did not read the data
block_read_req	in	Block data read request
block_read_valid	out	Block data read data is valid
block_read_data	out	Block data read data
block_read_req_ack	out	Block data read request response
block_write_req	in	Block data write request
block_write_data	in	Block data write data
block_write_data_rd	out	Block data write data request, block_write_data one clock cycle in advance
block_write_req_ack	out	Block data write request response

The "sd_card_sec_read_write" module has a state machine. The "SD" card is initialized first. The following figure shows the initialization state machine conversion diagram of the module. The "CMD0" command is sent first, then the "CMD8" command is sent, then "CMD55" is sent, and then "ACMD41" and "CMD16" are sent. If the response is normal, the sd card is initialized and waiting for the read and write commands of the SD card sector.



Then wait for the sector read and write instructions, and complete the sector read and write operations, the following figure is the module read and write state machine conversion map.



Two parameters are defined in this module. The initialization process of the SD card requires that the slow clock be used to send commands and configurations. After the initialization is successful, the fast clock is used to read and write data.

```
parameter  SPI_LOW_SPEED_DIV = 248

parameter  SPI_HIGH_SPEED_DIV = 0
```

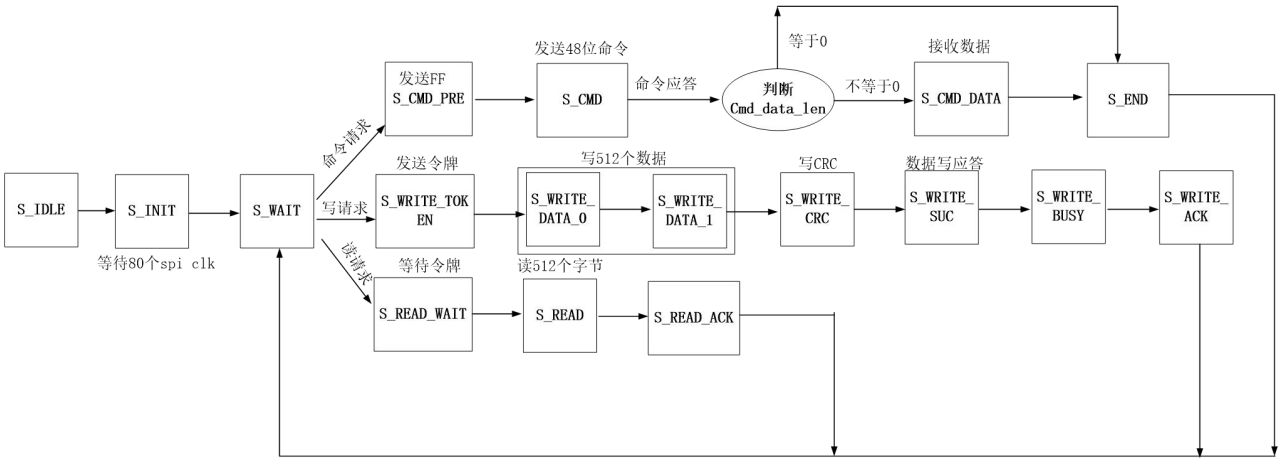
3.2 sd_card_cmd

The description of the “sd_card_cmd” module port is as follows:

Signal Name	Direction	Description
sys_clk	in	Clock input
rst	in	Asynchronous reset input, high reset
spi_clk_div	in	SPI clock division, SPI clock frequency = system clock / ((spi_clk_div + 2) * 2)
cmd_req	in	Sd card command request
cmd_req_ack	out	Sd card command request response
cmd_req_error	out	Sd card command request error
cmd	in	Sd card command, command + parameter + CRC, a total of 48bit
cmd_r1	in	The sd card command expects the R1 response
cmd_data_len	in	The length of the data read after the sd card command, most of the commands did not read the data

block_read_req	in	Block data read request
block_read_valid	out	Block data read data is valid
block_read_data	out	Block data read data
block_read_req_ack	out	Block data read request response
block_write_req	in	Block data write request
block_write_data	in	Block data write data
block_write_data_rd	out	Block data write data request, block_write_data one clock cycle in advance
block_write_req_ack	out	Block data write request response
nCS_ctrl	out	To the SPI master controller, cs chip select control
clk_div	out	To the SPI Master controller, the clock division parameter
spi_wr_req	out	Go to the SPI Master controller and write a byte request
spi_wr_ack	in	From the SPI Master controller, write request response
spi_data_in	out	Go to the SPI Master controller and write data
spi_data_out	in	From the SPI Master controller, reading data

The “sd_card_cmd” module mainly implements the basic command operation of the sd card, as well as the 88-cycle clock that is initialized by power-on. The command and read-write state machines of the data block are as follows.



From the “SD2.0” standard, we can see that from the master device write command to the “SD” card, the highest two bits “47~46” must be "01", which means the command transmission starts.

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

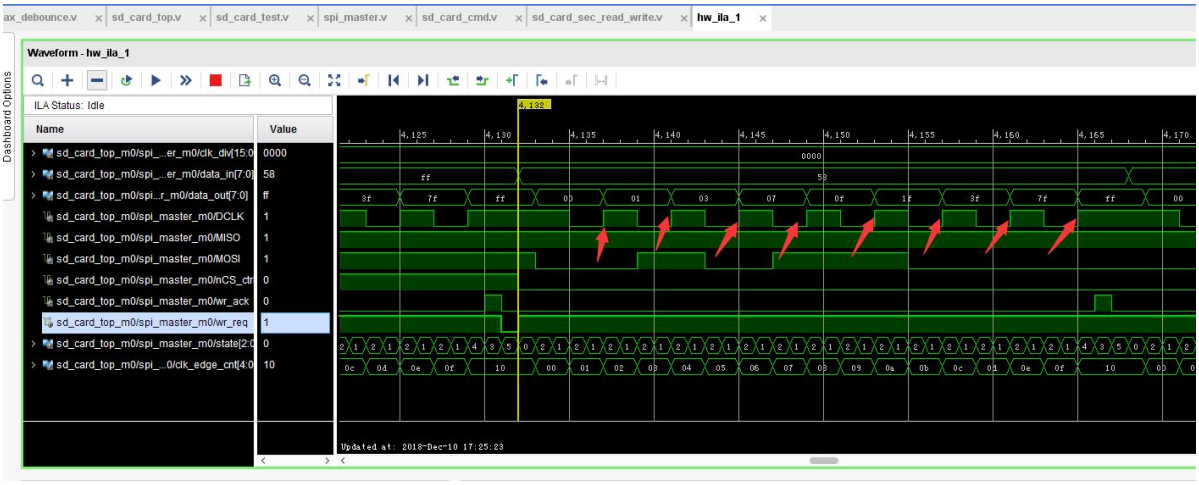
Table 4-17: Command Format

Therefore, the code is to write the result of the "eight" operation of the upper eight bits of the 48-bit command and the hexadecimal 0x40, so the following code is available:

```
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

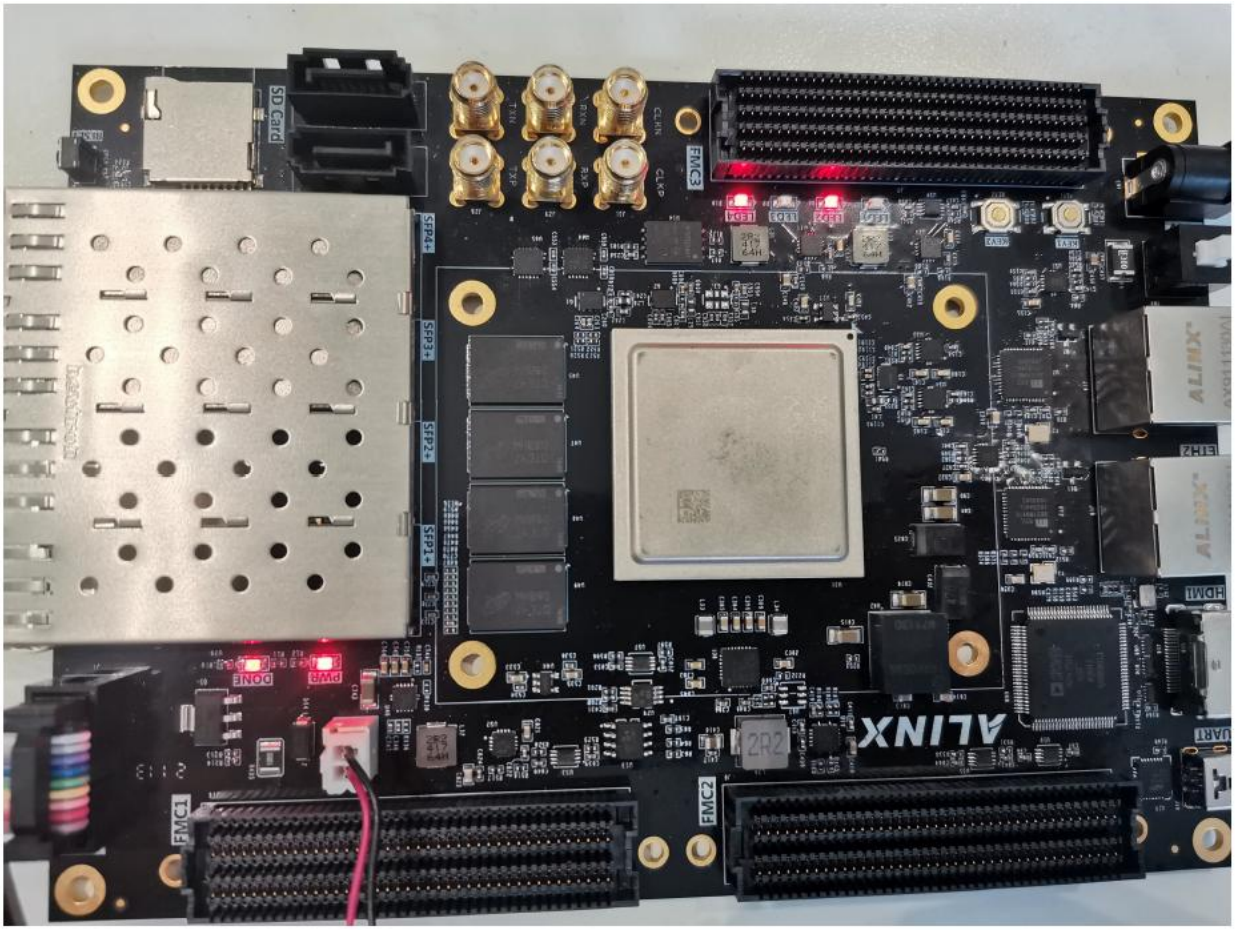
The “spi_master” module mainly completes reading and writing one byte of “SPI”. When the SPI state machine is in “idle”, it detects that the signal of wr_req is high, it will generate 8 DCLKs, and output data of “datain” from the high bit to the MOSI signal line. As shown in the ILA observation waveform below, the output data of “MOSI” at 8 “DCLKs” is the value of “datain 0x58”.



At the same time, the “spi_master” program also reads the “MISO” input data and converts it into 8-bit “data_out” data output to implement one byte of SPI data reading

4 Experiment Result

。After downloading the experimental program, you can see that the LED displays a binary number. This number is the first data stored in the first sector of the sd card. The data is random. At this time, KEY2 is pressed and the number is incremented by one. and write the sd card, download the program again, you can see the updated data directly displayed and combined with the logic analyzer "ila" view in vivado is more obvious.



AXKU040 FPGA Development Board