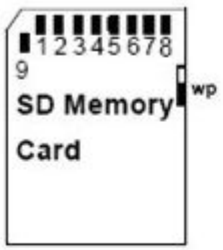
现在我们手机的内存卡多为Micro SD卡，又叫TF卡，所以Micro SD卡比SD卡常见。自己曾经也想写写SD卡的读取程序，但又不想特地再去买个SD卡，这时想起手机内存卡不是和SD卡很像吗？在网上查了以后发现SD卡和Micro SD卡其实也就大小和引脚不一样，它们的操作其实是一样的，所以网上的SD卡读写代码其实可以直接拿来用。关于SD卡和Micro SD卡的引脚定义和不同可见下两表：

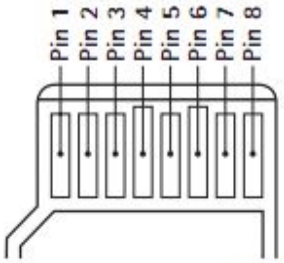


引脚编号	SD模式			SPI模式		
	名称	类型	描述	名称	类型	描述
1	CD/DAT3	IO或PP	卡检测/数据线3	#CS	I	片选
2	CMD	PP	命令/回应	DI	I	数据输入
3	V_{SS1}	S	电源地	VSS	S	电源地
4	V_{DD}	S	电源	VDD	S	电源
5	CLK	I	时钟	SCLK	I	时钟
6	V_{SS2}	S	电源地	VSS2	S	电源地
7	DAT0	IO或PP	数据线0	DO	O或PP	数据输出
8	DAT1	IO或PP	数据线1	RSV		
9	DAT2	IO或PP	数据线2	RSV		

注：S：电源供给 I：输入 O：采用推拉驱动的输出

PP：采用推拉驱动的输入输出

Micro SD
Memory Card



SD MODE

Pin No.	Name	Type	Description
1	DAT2	I/O/PP	Data Line (bit 2)
2	CD/DAT3	I/O/PP	Card Detect/Data line (Bit 3)
3	CMD	PP	Command Response
4	VDD	S	Supply Voltage
5	CLK	I	Clock
6	VSS	S	Supply Voltage Ground
7	DAT0	I/O/PP	Data Line (bit 0)
8	DAT1*	I/O/PP	Data Line (bit 1)

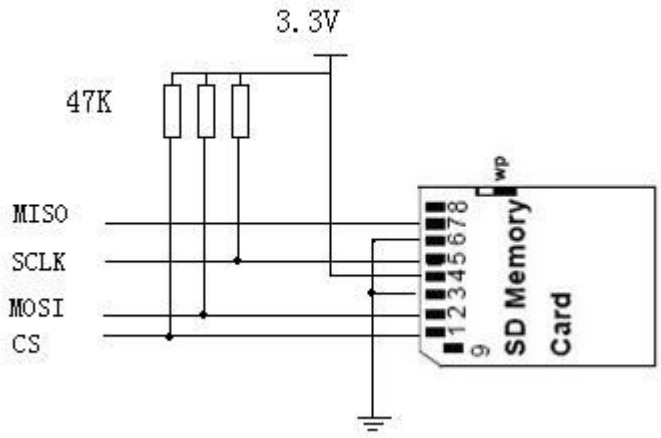
SPI MODE

Pin No.	Name	Type	Description
1	RSV	–	–
2	CS	I	Chip Select (neg true)
3	DI	I	Data In
4	VDD	S	Supply Voltage
5	SCLK	I	Clock
6	VSS	S	Supply Voltage Ground
7	DO	O/PP	Data Out
8	RSV	–	–

我们可以发现 Micro SD 卡只有8个引脚是因为比 SD 卡少了一个 Vss。当然你也可以买个卡套套在 Micro SD 卡上，这样一来大小就和 SD 卡一样大，这时候卡套上的9个引脚就和 SD 卡一样了，你可以完全当做 SD 卡来操作。

spi 下电路的连接非常简单，接上电源线 Vdd 和地线 Vss，再接上 spi 的 CS，SCLK，DI（MOSI）和 DO（MISO）就可以了，其他引脚可以放空。注意 SD 卡的电源和操作电压都为2.7-3.6V，5V 的单片机要进行电平转换或串电阻限流。还有记得 SD 卡的 CS，SCLKh 和 DI 要用10~100K 的电阻上拉。我是套了卡套接的电路，因为 Micro SD 卡的引脚太密了，不好焊接，SD 卡相对引脚好焊。因为没有卡座，而且也没专门的 PCB 我就直接焊到卡套上，诶牺牲了一个卡套。下面是我自己画的电路图：

SD Card SPI Model



上面 Micro SD 卡的硬件电路就好了，下面我们讲讲 Micro SD 卡的软件驱动和指令集。

SD 卡的命令格式如下，6字节共48位，传输时最高位(MSB)先传输：

SD卡命令格式

Byte 1			Bytes 2-5	Byte 6	
7	6	5 0	31 0	7	0
0	1	Command	Command Argument	CRC	1

SD 卡的 command（命令）占6 bit，一般叫 CMDx 或 ACMDx，比如 CMD1就是1，CMD13就是13，ACMD41就是41，依此类推。Command Argument（命令参数）占4 byte，并不是所有命令都有参数，没有参数的话该位一般就用置0。最后一个字节由7 bit CRC 校验位和1 bit 停止位组成。在 SPI 模式下，CRC 是被忽略的，可以都置1或置0.但是发送 CMD0时要记得加上 CRC，即最后1字节为0x95（因为发送 CMD0 时还未进入 SPI 模式，PS：CMD8也要，但一般大家都把发送 CMD8省略了）。

每次发送完一次命令后，SD 卡都会有回应。SD 卡的回应有多种格式，1字节的 R1，2字节的 R2 等，不过一般在 SPI 模式中我们只用到 R1，下面介绍 R1的格式：

- **In idle state**: The card is in idle state and running the initializing process.
- **Erase reset:** An erase sequence was cleared before executing because an out of erase sequence command was received.
- **Illegal command**: An illegal command code was detected.
- **Communication CRC error**: The CRC check of the last command failed.
- **Erase sequence error**: An error in the sequence of erase commands occurred.
- **Address error**: A misaligned address that did not match the block length was used in the command.
- **Parameter error**: The command's argument (e.g. address, block length) was outside the allowed range for this card.

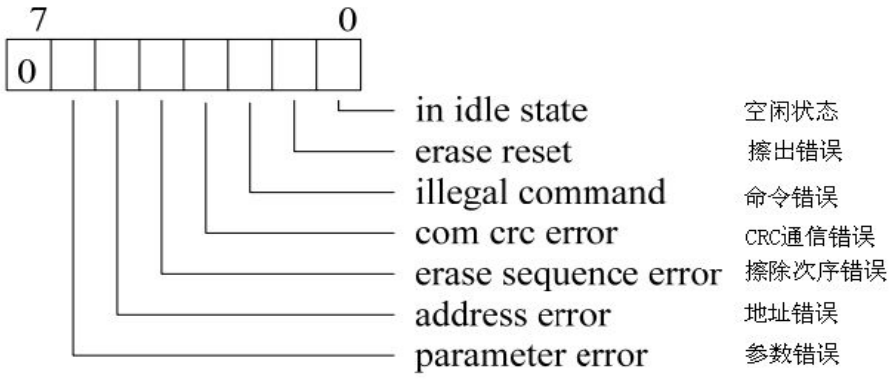


Figure 7-9: R1 Response Format

关于 SD 卡 SPI 和 command 的发送要注意以下几点：

1.SD 卡的 SPI 总线，在读入数据时 SD 卡的 SPI 是 CLK 的上升沿输入锁存，输出数据也是在上升

沿。

2.向 SD 卡写入一个 CMD 或者 ACMD 指令的过程是这样的： 首先使 CS 为低电平，SD 卡使能；其次在 SD 卡的 Din 写入指令；写入指令后还要附加8个填充时钟，是 SD 卡完成内部操作；之后在 SD 卡的 Dout 上接受回应；回应接受完毕使 CS 为低电平，再附加8个填充时钟。 3.在 SD 卡的 Din 没有数据写入时，应使 Din 保持高电平。关于这一点我可吃透了苦头，本来也记得要保持高电平的，结果不知怎的鬼使神差的置0拉低了。结果程序出现了各种奇怪的貌似偶然的错误，比如连续两次复位会有一次失败，单步调试成功全速运行又会失败。总之在这个过程中我对时序进行各种改变，每次解决一个问题后又会有新的问题出现，多少次动摇了我对 MicroSD 卡和 SD 卡的操作是一样的这个看法。因为这个低级的错误耽误了我三四天，看来细心很重要啊！我已经不止一次因为不细心浪费大量时间了，希望大家也引以为戒。

好了，现在 SD 卡的命令和回应清楚了，我们下面讲讲 SD 卡的复位，初始化和读写方法。

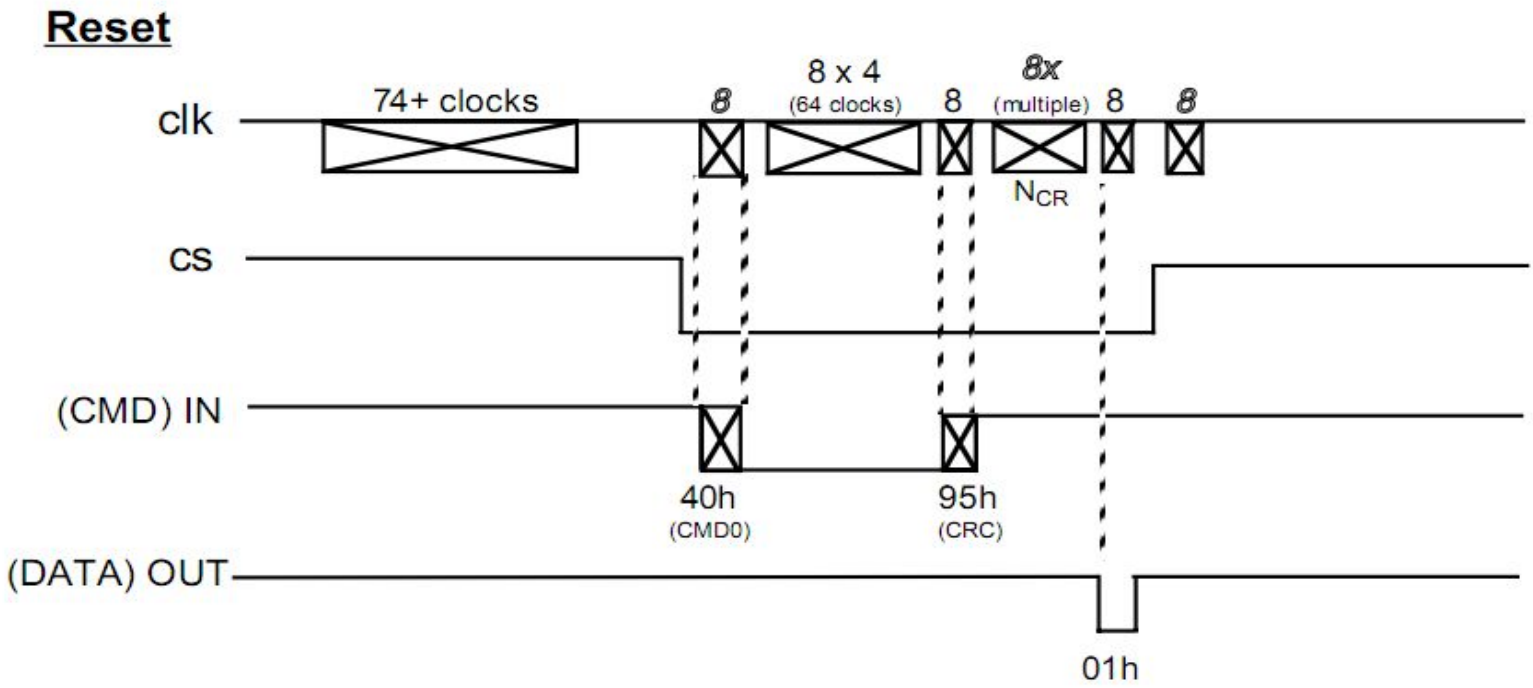
复位方法：

1.拉高 CS，发送至少74个 clk 周期来使 SD 卡达到正常工作电压和进行同步

2.选低 CS，发送 CMD0，需要收到回应0x01表示成功进入 idle 状态

3.拉高 CS，发送8个时钟

复位时序图：



初始化：

复位成功后，SD 卡就进入了 SPI 模式，接着应该进行初始化。初始化说白了有两种方法：（1）

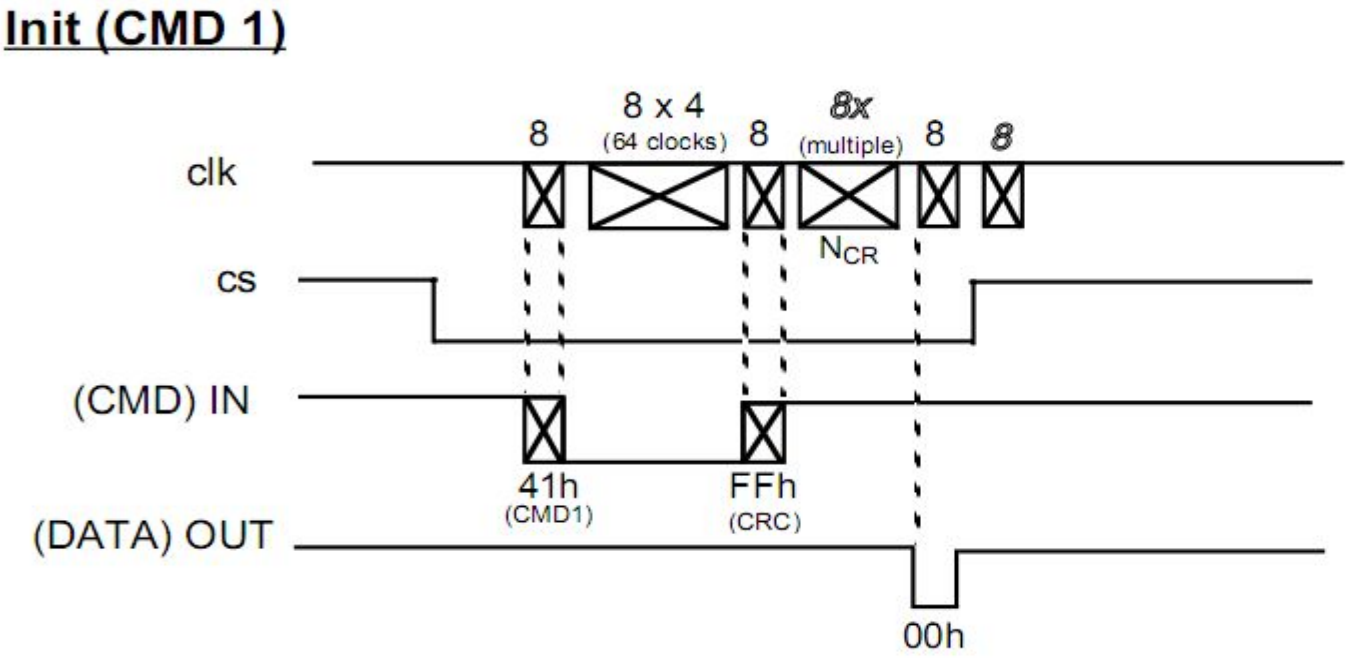
发送 CMD1，（2）发送 CMD55+ACMD41。我从网上查的资料可以看到这种说法：如果是 MMC 卡就发 CMD1，SD 卡则发 CMD55+ACMD41。但是关于 Micro SD 卡要发哪种却讲的不太清楚，网上用这两种方法都有人成功过，但有的都成功不了。我自己也碰到了这种问题，刚开始拿了自己手机上的写着 Nokia 的2GB 的 Micro SD 卡（应该是杂牌的）初始化了两天也没成功，快要放弃的时候想起来为什么不换张试试呢，于是就找室友借了他的手机内存卡，是2GB 的 Apacer 的 Micro SD 卡（当然也可能是杂牌的，室友买那卡的地方一般都是卖各种廉价电子产品的，大家都知道是杂牌的），结果一试就成功了。后来我用了令一种方法发现也可以初始化，也就是说两种方法都可以初始化成功。但我的那种怎么就不行呢？难道不是所有 Micro SD 卡都支持 SPI 模式。我在网上百度了半天也不能确定是不是所有 Micro SD 卡都支持 SPI 模式。但我想，现在 Micro SD 卡的生产公司很多，而且你也并不能保证你的 Micro SD 卡不是杂牌的。你并不知道生产厂家进行了那些改变，因为确实有些厂家生产的 SD 卡精简了一些命令。所以初始化的时候建议两种都试一下，不过我记得 SD 卡的说明书上推荐使用第二种方法。

下面是初始化方法：

（1）使用 CMD1

发送 CMD1，收到0x00表示成功

时序图如下：



（2）使用 CMD55+ACMD41

1.发送 CMD55（表示使用 ACMDx 类命令），收到0x01

2.发送 ACMD41，收到0x00表示成功

记住 SD 卡的初始化速度不能大于400kHz，所以一开始复位和初始化时 spi 的速率要设置低一点。

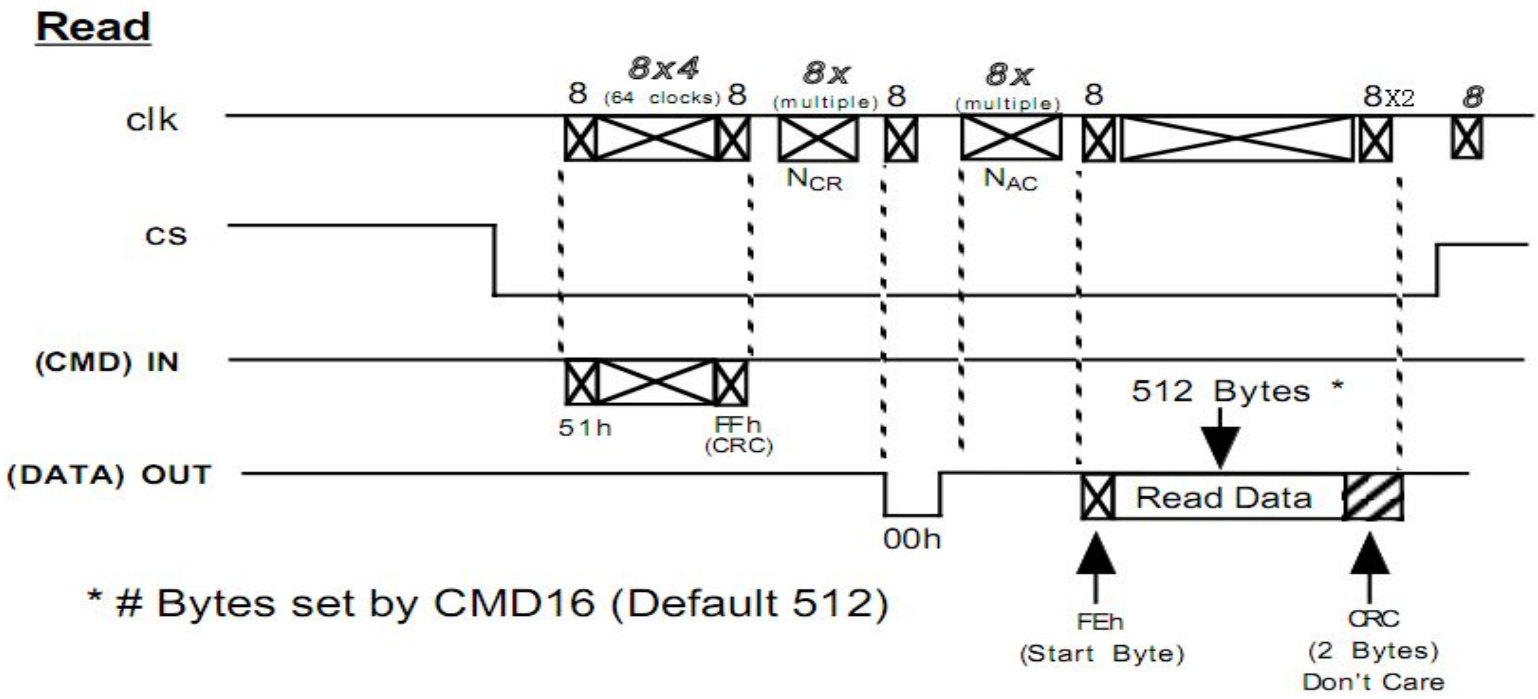
读单块和多块：

SD 卡读单块和多块的命令分别为 CMD17和 CMD18，他们的参数即要读的区域的开始地址。因为考虑到一般 SD 卡的读写要求地址对齐，所以一般我们都将地址转为块，并以扇区（块）（512Byte）为单位进行读写，比如读扇区0参数就为0，读扇区1参数就为1<<9(即地址512)，读扇区2参数就为2<<9(即地址1024)，依此类推。

读单块方法：

1. 发送 CMD17，收到0x00表示成功
2. 连续读直到读到开始字节0xFE
3. 读512个字节
4. 读两个 CRC 字节

读单块时序图：



读多块方法：

1. 发送 CMD18读，收到0x00表示成功
2. 连续读直到读到开始字节0xFE
3. 读512字节
4. 读两个 CRC 字节
5. 如果还想读下一扇区，重复2-4
6. 发送 CMD12来停止读多块操作

写单块和多块：

SD 卡用 CMD24和 CMD25来写单块和多块，参数的定义和读操作是一样的。

写单块方法：

1.发送 CMD24，收到0x00表示成功

2.发送若干时钟

3.发送写单块开始字节0xFE

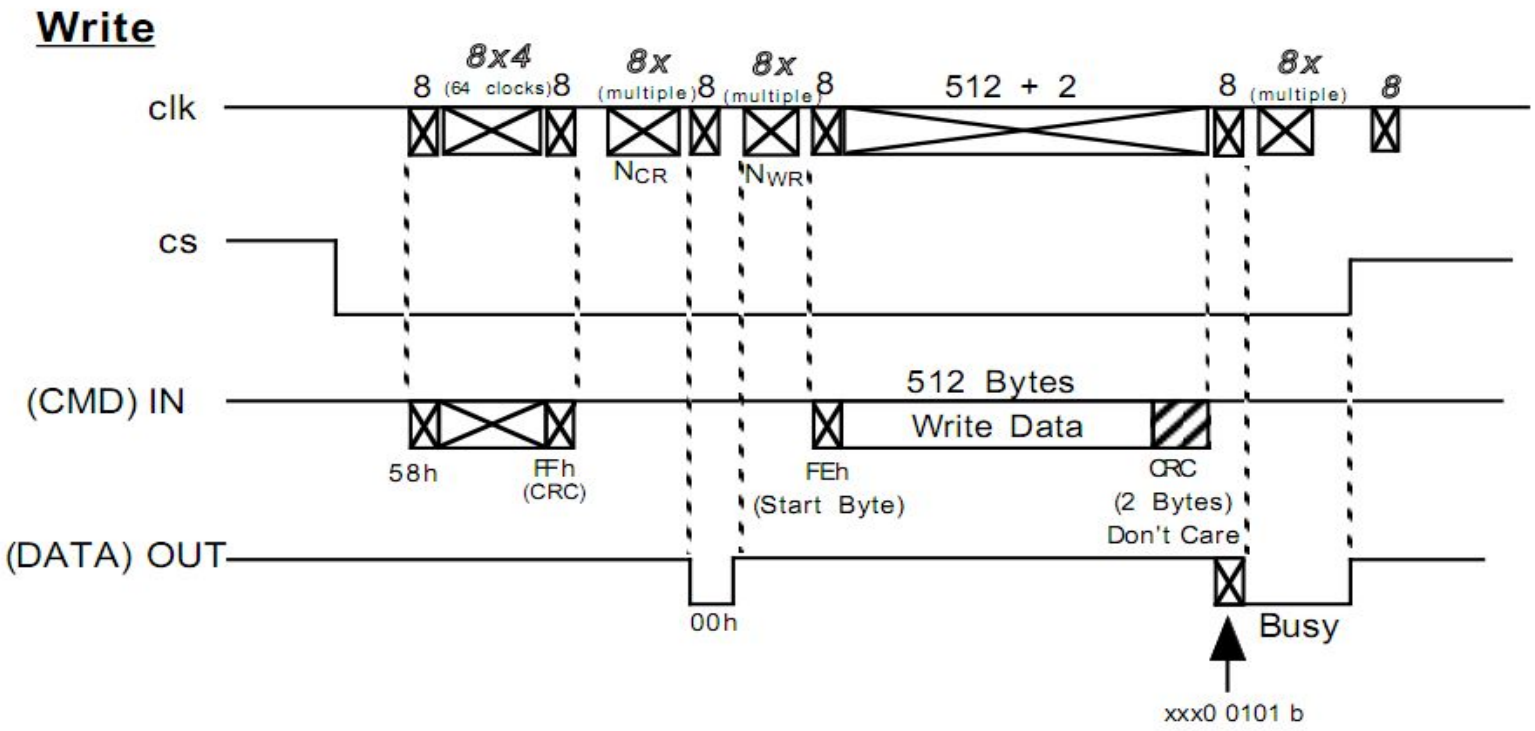
4.发送512个字节数据

5.发送2字节 CRC（可以均为0xff）

6.连续读直到读到 XXX00101表示数据写入成功

7.继续读进行忙检测（读到0x00表示 SD 卡正忙），当读到0xff 表示写操作完成

写单块时序图：



写多块方法：

1.发送 CMD25，收到0x00表示成功

2.发送若干时钟

3.发送写多块开始字节0xFC

4.发送512字节数据

5.发送两个 CRC（可以均为0xff）

6.连续读直到读到 XXX00101表示数据写入成功

7.继续读进行忙检测，直到读到0xFF 表示写操作完成

8. 如果想读下一扇区重复2-7步骤

9. 发送写多块停止字节0xFD 来停止写操作

10. 进行忙检测直到读到0xFF

上面介绍了 Micro SD 卡的硬件连接，复位、初始化、读写单块和多块的实现方法，其实你还可以读取 SD 卡 ID，给 SD 卡命名，设置密码，改变一次读写的大小，这里就不多介绍，大家可以自己看 SD 卡的官方资料。下一篇文章我就贴出自己写的基于 nios ii 的 Micro SD 卡程序，程序在实现了上面介绍的功能外，还增加了读 CSD，CID 寄存器的功能。

头文件 SD_spi_solution.h

```
#ifndef SD_SPI_SOLUTION_H_
#define SD_SPI_SOLUTION_H_

#include<system.h>
#include<alt_types.h>
#include<altera_avalon_pio_regs.h>

#define CMD0    0   /* GO_IDLE_STATE */
#define CMD55   55  /* APP_CMD */
#define ACMD41  41  /* SEND_OP_COND (ACMD) */
#define CMD1    1   /* SEND_OP_COND */
#define CMD17   17  /* READ_SINGLE_BLOCK */
#define CMD8    8   /* SEND_IF_COND */
#define CMD18   18  /* READ_MULTIPLE_BLOCK */
#define CMD12   12  /* STOP_TRANSMISSION */
#define CMD24   24  /* WRITE_BLOCK */
#define CMD25   25  /* WRITE_MULTIPLE_BLOCK */
#define CMD13   13  /* SEND_STATUS */
#define CMD9    9   /* SEND_CSD */
#define CMD10   10  /* SEND_CID */

#define CSD     9
#define CID     10

//delay 1us（actually not, it maybe is several us, I don't test it）
void usleep(alt_u8 i);

//set CS low
void CS_Enable();

//set CS high and send 8 clocks
void CS_Disable();
```

```
33
34 //write a byte
35 void SDWriteByte(alt_u8 data);
36
37 //read a byte
38 alt_u8 SDReadByte();
39
40 //send a command and send back the response
41 alt_u8  SDSendCmd(alt_u8 cmd,alt_u32 arg,alt_u8 crc);
42
43 //reset SD card
44 alt_u8 SDReset();
45
46 //initial SD card
47 alt_u8 SDInit();
48
49 //read a single sector
50 alt_u8 SDReadSector(alt_u32 addr,alt_u8 * buffer);
51
52 //read multiple sectors
53 alt_u8 SDReadMultiSector(alt_u32 addr,alt_u8 sector_num,alt_u8 * buffer);
54
55 //write a single sector
56 alt_u8 SDWriteSector(alt_u32 addr,alt_u8 * buffer);
57
58 //write multiple sectors
59 alt_u8 SDWriteMultiSector(alt_u32 addr,alt_u8 sector_num,alt_u8 * buffer);
60
61 //get CID or CSD
62 alt_u8 SDGetCIDCSD(alt_u8 cid_csd,alt_u8 * buffer);
63
64 //spi speed（0-255），0 is fastest
65 alt_u8 spi_speed;
66
67 #endif /* SD_SPI_SOLUTION_H_ */
```

源文件 SD_spi_Solution.c

```
68 #include"SD_spi_Solution.h"
69
70 alt_u8 spi_speed = 10;//the spi speed(0-255),0 is fastest
71
72 //delay 1us（actually not，it maybe is several us，I don't test it）
73 void usleep(alt_u8 i)
```

```
{
    while(i --);
}

//set CS low
void CS_Enable()
{
    //set CS low
    IOWR_ALTERA_AVALON_PIO_DATA(CS_BASE, 0x00);
}

//set CS high and send 8 clocks
void CS_Disable()
{
    //set CS high
    IOWR_ALTERA_AVALON_PIO_DATA(CS_BASE, 0x01);
    //send 8 clocks
    SDWriteByte(0xff);
}

//write a byte
void SDWriteByte(alt_u8 data)
{
    alt_u8 i;

    //write 8 bits(MSB)
    for(i = 0;i < 8;i ++)
    {
        IOWR_ALTERA_AVALON_PIO_DATA(SCLK_BASE, 0x00);
        usleep(spi_speed);
        if(data & 0x80) IOWR_ALTERA_AVALON_PIO_DATA(DI_BASE, 0x01);
        else IOWR_ALTERA_AVALON_PIO_DATA(DI_BASE, 0x00);
        data <<= 1;
        IOWR_ALTERA_AVALON_PIO_DATA(SCLK_BASE, 0x01);
        usleep(spi_speed);
    }

    //when DI is free,it should be set high
    IOWR_ALTERA_AVALON_PIO_DATA(DI_BASE, 0x01);
}

//read a byte
alt_u8 SDReadByte()
{
```

```
118     alt_u8 data = 0x00,i;
119 
120     //read 8 bit(MSB)
121     for(i = 0;i < 8;i ++)
122     {
123         IOWR_ALTERA_AVALON_PIO_DATA(SCLK_BASE, 0x00);
124         usleep(spi_speed);
125         IOWR_ALTERA_AVALON_PIO_DATA(SCLK_BASE, 0x01);
126         data <<= 1;
127         if(IORD_ALTERA_AVALON_PIO_DATA(DO_BASE)) data |= 0x01;
128         usleep(spi_speed);
129     }
130 
131     return data;
132 }
133 
134 //send a command and send back the response
135 alt_u8  SDSendCmd(alt_u8 cmd,alt_u32 arg,alt_u8 crc)
136 {
137     alt_u8 r1,time = 0;
138 
139     //send the command,arguments and CRC
140     SDWriteByte((cmd & 0x3f) | 0x40);
141     SDWriteByte(arg >> 24);
142     SDWriteByte(arg >> 16);
143     SDWriteByte(arg >> 8);
144     SDWriteByte(arg);
145     SDWriteByte(crc);
146 
147     //read the respond until responds is not '0xff' or timeout
148     do{
149         r1 = SDReadByte();
150         time ++;
151         //if time out,return
152         if(time > 254) break;
153     }while(r1 == 0xff);
154 
155     return r1;
156 }
157 
158 //reset SD card
159 alt_u8 SDReset()
160 {
161     alt_u8 i,r1,time = 0;
```

```
162
163     //set CS high
164     CS_Disable();
165
166     //send 128 clocks
167     for(i = 0;i < 16;i ++)
168     {
169         SDWriteByte(0xff);
170     }
171
172     //set CS low
173     CS_Enable();
174
175     //send CMD0 till the response is 0x01
176     do{
177         r1 = SDSendCmd(CMD0,0,0x95);
178         time ++;
179         //if time out,set CS high and return r1
180         if(time > 254)
181         {
182             //set CS high and send 8 clocks
183             CS_Disable();
184             return r1;
185         }
186     }while(r1 != 0x01);
187
188     //set CS high and send 8 clocks
189     CS_Disable();
190
191     return 0;
192 }
193
194 //initial SD card(send CMD55+ACMD41 or CMD1)
195 alt_u8 SDInit()
196 {
197     alt_u8 r1,time = 0;
198
199     //set CS low
200     CS_Enable();
201
202     //check interface operating condition
203     r1 = SDSendCmd(CMD8,0x000001aa,0x87);
204     //if support Ver1.x,but do not support Ver2.0,set CS high and return r1
205     if(r1 == 0x05)
```

```
206     {
207         //set CS high and send 8 clocks
208         CS_Disable();
209         return r1;
210     }
211     //read the other 4 bytes of response(the response of CMD8 is 5 bytes)
212     r1=SDReadByte();
213     r1=SDReadByte();
214     r1=SDReadByte();
215     r1=SDReadByte();
216 
217     do{
218         //send CMD55+ACMD41 to initial SD card
219         do{
220             r1 = SDSendCmd(CMD55,0,0xff);
221             time ++;
222             //if time out,set CS high and return r1
223             if(time > 254)
224             {
225                 //set CS high and send 8 clocks
226                 CS_Disable();
227                 return r1;
228             }
229         }while(r1 != 0x01);
230 
231         r1 = SDSendCmd(ACMD41,0x40000000,0xff);
232 
233         //send CMD1 to initial SD card
234         //r1 = SDSendCmd(CMD1,0x00ffc000,0xff);
235         time ++;
236 
237         //if time out,set CS high and return r1
238         if(time > 254)
239         {
240             //set CS high and send 8 clocks
241             CS_Disable();
242             return r1;
243         }
244     }while(r1 != 0x00);
245 
246     //set CS high and send 8 clocks
247     CS_Disable();
248 
249     return 0;
```

```
250 }
251
252 //read a single sector
253 alt_u8 SDReadSector(alt_u32 addr,alt_u8 * buffer)
254 {
255     alt_u8 r1;
256     alt_u16 i,time = 0;
257
258     //set CS low
259     CS_Enable();
260
261     //send CMD17 for single block read
262     r1 = SDSendCmd(CMD17,addr << 9,0x55);
263     //if CMD17 fail,return
264     if(r1 != 0x00)
265     {
266         //set CS high and send 8 clocks
267         CS_Disable();
268         return r1;
269     }
270
271     //continually read till get the start byte 0xfe
272     do{
273         r1 = SDReadByte();
274         time ++;
275         //if time out,set CS high and return r1
276         if(time > 30000)
277         {
278             //set CS high and send 8 clocks
279             CS_Disable();
280             return r1;
281         }
282     }while(r1 != 0xfe);
283
284     //read 512 Bits of data
285     for(i = 0;i < 512;i ++)
286     {
287         buffer[i] = SDReadByte();
288     }
289
290     //read two bits of CRC
291     SDReadByte();
292     SDReadByte();
293
```

```
294     //set CS high and send 8 clocks
295     CS_Disable();
296 
297     return 0;
298 }
299 
300 //read multiple sectors
301 alt_u8 SDReadMultiSector(alt_u32 addr,alt_u8 sector_num,alt_u8 * buffer)
302 {
303     alt_u16 i,time = 0;
304     alt_u8 r1;
305 
306     //set CS low
307     CS_Enable();
308 
309     //send CMD18 for multiple blocks read
310     r1 = SDSendCmd(CMD18,addr << 9,0xff);
311     //if CMD18 fail,return
312     if(r1 != 0x00)
313     {
314         //set CS high and send 8 clocks
315         CS_Disable();
316         return r1;
317     }
318 
319     //read sector_num sector
320     do{
321         //continually read till get start byte
322         do{
323             r1 = SDReadByte();
324             time ++;
325             //if time out,set CS high and return r1
326             if(time > 30000 || ((r1 & 0xf0) == 0x00 && (r1 & 0x0f)))
327             {
328                 //set CS high and send 8 clocks
329                 CS_Disable();
330                 return r1;
331             }
332         }while(r1 != 0xfe);
333         time = 0;
334 
335         //read 512 Bits of data
336         for(i = 0;i < 512;i ++)
337         {
```

```
338             *buffer ++ = SDReadByte();
339         }
340 
341         //read two bits of CRC
342         SDReadByte();
343         SDReadByte();
344     }while( -- sector_num);
345     time = 0;
346 
347     //stop multiple reading
348     r1 = SDSendCmd(CMD12,0,0xff);
349 
350     //set CS high and send 8 clocks
351     CS_Disable();
352 
353     return 0;
354 }
355 
356 //write a single sector
357 alt_u8 SDWriteSector(alt_u32 addr,alt_u8 * buffer)
358 {
359     alt_u16 i,time = 0;
360     alt_u8 r1;
361 
362     //set CS low
363     CS_Enable();
364 
365     do{
366         do{
367             //send CMD24 for single block write
368             r1 = SDSendCmd(CMD24,addr << 9,0xff);
369             time ++;
370             //if time out,set CS high and return r1
371             if(time > 254)
372             {
373                 //set CS high and send 8 clocks
374                 CS_Disable();
375                 return r1;
376             }
377         }while(r1 != 0x00);
378         time = 0;
379 
380         //send some dummy clocks
381         for(i = 0;i < 5;i ++)
```

```
382         {
383             SDWriteByte(0xff);
384         }
385 
386         //write start byte
387         SDWriteByte(0xfe);
388 
389         //write 512 bytes of data
390         for(i = 0;i < 512;i ++)
391         {
392             SDWriteByte(buffer[i]);
393         }
394 
395         //write 2 bytes of CRC
396         SDWriteByte(0xff);
397         SDWriteByte(0xff);
398 
399         //read response
400         r1 = SDReadByte();
401         time ++;
402         //if time out,set CS high and return r1
403         if(time > 254)
404         {
405             //set CS high and send 8 clocks
406             CS_Disable();
407             return r1;
408         }
409     }while((r1 & 0x1f)!= 0x05);
410     time = 0;
411 
412     //check busy
413     do{
414         r1 = SDReadByte();
415         time ++;
416         //if time out,set CS high and return r1
417         if(time > 60000)
418         {
419             //set CS high and send 8 clocks
420             CS_Disable();
421             return r1;
422         }
423     }while(r1 != 0xff);
424 
425     //set CS high and send 8 clocks
```

```
426     CS_Disable();
427 
428     return 0;
429 }
430 
431 //write several blocks
432 alt_u8 SDWriteMultiSector(alt_u32 addr,alt_u8 sector_num,alt_u8 * buffer)
433 {
434     alt_u16 i,time = 0;
435     alt_u8 r1;
436 
437     //set CS low
438     CS_Enable();
439 
440     //send CMD25 for multiple block read
441     r1 = SDSendCmd(CMD25,addr << 9,0xff);
442     //if CMD25 fail,return
443     if(r1 != 0x00)
444     {
445         //set CS high and send 8 clocks
446         CS_Disable();
447         return r1;
448     }
449 
450     do{
451         do{
452             //send several dummy clocks
453             for(i = 0;i < 5;i ++)
454             {
455                 SDWriteByte(0xff);
456             }
457 
458             //write start byte
459             SDWriteByte(0xfc);
460 
461             //write 512 byte of data
462             for(i = 0;i < 512;i ++)
463             {
464                 SDWriteByte(*buffer ++);
465             }
466 
467             //write 2 byte of CRC
468             SDWriteByte(0xff);
469             SDWriteByte(0xff);
```

```
470
471             //read response
472             r1 = SDReadByte();
473             time ++;
474             //if time out,set CS high and return r1
475             if(time > 254)
476             {
477                 //set CS high and send 8 clocks
478                 CS_Disable();
479                 return r1;
480             }
481         }while((r1 & 0x1f)!= 0x05);
482         time = 0;
483 
484         //check busy
485         do{
486             r1 = SDReadByte();printf("n%d",r1);
487             time ++;
488             //if time out,set CS high and return r1
489             if(time > 30000)
490             {
491                 //set CS high and send 8 clocks
492                 CS_Disable();
493                 return r1;
494             }
495         }while(r1 != 0xff);
496         time = 0;
497     }while(-- sector_num);
498 
499     //send stop byte
500     SDWriteByte(0xfd);
501 
502     //check busy
503     do{
504         r1 = SDReadByte();
505         time ++;
506         //if time out,set CS high and return r1
507         if(time > 30000)
508         {
509             //set CS high and send 8 clocks
510             CS_Disable();
511             return r1;
512         }
513     }while(r1 != 0xff);
```

```
514
515     //set CS high and send 8 clocks
516     CS_Disable();
517
518     return 0;
519 }
520
521 //get CID or CSD
522 alt_u8 SDGetCIDCSD(alt_u8 cid_csd,alt_u8 * buffer)
523 {
524     alt_u8 r1;
525     alt_u16 i,time = 0;
526
527     //set CS low
528     CS_Enable();
529
530     //send CMD10 for CID read or CMD9 for CSD
531     do{
532         if(cid_csd == CID)
533             r1 = SDSendCmd(CMD10,0,0xff);
534         else
535             r1 = SDSendCmd(CMD9,0,0xff);
536         time ++;
537         //if time out,set CS high and return r1
538         if(time > 254)
539         {
540             //set CS high and send 8 clocks
541             CS_Disable();
542             return r1;
543         }
544     }while(r1 != 0x00);
545     time = 0;
546
547     //continually read till get 0xfe
548     do{
549         r1 = SDReadByte();
550         time ++;
551         //if time out,set CS high and return r1
552         if(time > 30000)
553         {
554             //set CS high and send 8 clocks
555             CS_Disable();
556             return r1;
557         }
```

```
558     }while(r1 != 0xfe);
559 
560     //read 512 Bits of data
561     for(i = 0;i < 16;i ++)
562     {
563         *buffer ++ = SDReadByte();
564     }
565 
566     //read two bits of CRC
567     SDReadByte();
568     SDReadByte();
569 
570     //set CS high and send 8 clocks
571     CS_Disable();
572 
573     return 0;
574 }
```