***[初始化：](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)***[1、初始化读写SD卡的硬件条件（spi接口和其他有用的管腿，如写保护）  
2、上电延时过程  
3、复位卡](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)**[CMD0](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)** [4、激活卡，内部初始化](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)**[并获取存储卡的类型  
     CMD1,CMD55,ACMD41](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)** [5、查询OCR，获取卡供电情况](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)**[CMD58](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)** [6、是否使用CRC](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/) **[CMD59](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)** [7、设置读、写块数据长度，512B，](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)**[CMD16](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)** [8、读取CSD，](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)**[获取存储卡的其他参数信息](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)**[。](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)**[CMD9](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)** [9、 8clock后，禁止片选。](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)***[读取单块数据](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)*** [1、主机发送](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)**[CMD17](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)** [2、接收卡响应R1  
3、接收读数据起始令牌0xFE  
4、接收数据  
5、接收2B CRC  
6 、8clock后，禁止片选。](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)***[写入单块数据](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)*** [1、主机发送](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)**[CMD24](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)** [2、接收卡响应R1  
3、接收读数据起始令牌0xFE  
4、接收数据  
5、接收2B CRC  
6 、8clock后，禁止片选。](http://www.rosoo.net/tags.php?/SD%BF%A8%B2%D9%D7%F7/)

SD卡调试关键点：

1.      上电时要延时足够长的时间给SD卡一个准备过程，在我的程序里是5秒，根据不同的卡设置不同的延时时间。SD卡初始化第一步在发送CMD命令之前，在片选有效的情况下首先要发送至少74个时钟，否则将有可能出现SD卡不能初始化的问题。

2.      SD卡发送复位命令CMD0后，要发送版本查询命令CMD8，返回状态一般分两种，若返回0x01表示此SD卡接受CMD8,也就是说此SD卡支持版本2；若返回0x05则表示此SD卡支持版本1。因为不同版本的SD卡操作要求有不一样的地方，所以务必查询SD卡的版本号，否则也会出现SD卡无法正常工作的问题。

3.      理论上要求发送CMD58获得SD卡电压参数，但实际过程中由于事先都知道了SD卡的工作电压，因此可省略这一步简化程序。协议书上也建议尽量不要用这个命令。

4.      SD卡读写超时时间要按照协议说明书书上的给定值(读超时：100ms；写超时：250ms)，这个值要在程序中准确计算出来，否则将会出现不能正常读写数据的问题。我自己定义了一个计算公式：超时时间=(8/clk)\*arg。

5.      2GB以内的SD卡(标准卡)和2GB以上的SD卡(大容量卡)在地址访问形式上不同，这一点尤其要注意，否则将会出现无法读写数据的问题。如标准卡在读写操作时，对读或写命令令牌当中的地址域符初值0x10，表示对第16个字节以后的地址单元进行操作(前提是此SD卡支持偏移读写操作)，而对大容量卡读或写命令令牌当中的地址域符初值0x10时，则表示对第16块进行读写操作，而且大容量卡只支持块读写操作，块大小固定为512字节，对其进行字节操作将会出错。

6.      对某一块要进行写操作时最好先执行擦出命令，这样写入的速度就能大大提高。进行擦除操作时不管是标准卡还是大容量卡都按块操作执行，也就是一次擦除至少512字节。

7.      对标准卡进行字节操作时，起始和终止必须在一个物理扇区内，否则将不能进行读写操作。实际操作过程中建议用块操作以提高效率。不管是标准卡还是大容量卡一个读写命令只能对一个块进行操作，不允许跨物理层地址操作。

8.      在写数据块前要先写入若干个dummy data字节，写完一个块数据时，主机要监测MISO数据线，如果从机处于忙状态这根数据线会保持低电平，这样主机就可以根据这根数据线的状态以决定是否发送下一个命令，在从机没有释放MISO数据线之前，主机绝对不能执行其他命令，否则将会导致写入的数据出错，而且从机也不会响应主机的命令。

9.      在SPI模式下，CRC校验是被忽略的，但依然要求主从机发送CRC码，只是数值可以是任意值，一般主机的CRC码通常设为0x00或0xFF。

读多块操作和写多块操作的传输停止形式不一样，读多块操作时用用命令CMD12终止传输，而写多块操作时用Stop Tran Token(停止传输令牌，值为0xFD)终止传输。

----------------------------------------------------------------------------------------

1、  
初始化步骤：  
（1）      
延时至少74clock，等待SD卡内部操作完成，在MMC协议中有明确说明。  
（2）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原理，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，进一步发送命令循环发送CMD55+ACMD41，直到返回0x00，确定SD2.0卡初始化成功，进入Ready状态，再发送CMD58命令来判断是HCSD还是SCSD，到此SD2.0卡初始化成功。如果CMD8返回错误则进一步判断为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，初始结束。  
（5）CS拉高。  
2、  
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0xfe（或0xfc）+正式数据512Bytes + CRC校验2Bytes  
默认正式传输的数据长度是512Bytes，可用CMD16设置块长度。  
3、  
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0xfe（或0xfc）+正式数据512Bytes + CRC校验2Bytes  
4、  
擦除步骤：  
（1）      
发送CMD32，跟一个参数来指定首个要擦除的起始地址（SD手册上说是块号）  
（2）      
发送CMD33,，指定最后的地址  
（3）      
发送CMD38，擦除指定区间的内容  
此3步顺序不能颠倒。  
最后说一下我的一点体会：SD卡就是一个存储器，只不过用命令的方式来进行操作，我们只要掌 握了各条命令及操作方式，就可以灵活的操作SD卡了，另外我所了解的IC卡也是类似的原理，还有就是建议开始看MMC的协议，简单明了易懂些，有了对 MMC卡的一些了解后看SD卡协议就容易多了。

（1）图中的Y轴坐标表示SD卡支持的电压，一般都是2.7-3.6v之间。X轴为时间坐标。  
（2）power up time阶段：

    这段时间为插上SD卡，到sd卡的的供电电压达到最低要求电压的这段时间。这个时候sd clock还没正常提供，处于禁止状态，Linux在这段时间内delay 10ms。

（3）supply ramp up time阶段：

  （2）（3）阶段时间段，对应Linux mmc子系统的codes为：

/\*

\* Apply power to the MMC stack.  This is a two-stage process.

\* First, we enable power to the card without the clock running.

\* We then wait a bit for the power to stabilise.  Finally,

\* enable the bus drivers and clock to the card.

\*

\* We must \_NOT\_ enable the clock prior to power stablising.

\*

\* If a host does all the power sequencing itself, ignore the

\* initial MMC\_POWER\_UP stage.

\*/

static void mmc\_power\_up(struct mmc\_host \*host)

{

int bit;

/\* If ocr is set, we use it \*/

if (host->ocr)

  bit = ffs(host->ocr) - 1;

else

  bit = fls(host->ocr\_avail) - 1;

host->ios.vdd = bit;

if (mmc\_host\_is\_spi(host)) {

  host->ios.chip\_select = MMC\_CS\_HIGH;

  host->ios.bus\_mode = MMC\_BUSMODE\_PUSHPULL;

} else {

  host->ios.chip\_select = MMC\_CS\_DONTCARE;

  host->ios.bus\_mode = MMC\_BUSMODE\_OPENDRAIN;

}

host->ios.power\_mode = MMC\_POWER\_UP;

host->ios.bus\_width = MMC\_BUS\_WIDTH\_1;

host->ios.timing = MMC\_TIMING\_LEGACY;

mmc\_set\_ios(host);//这段code被调用之前，为状态（2）power up time时间段，这段时间内host->ios.clock被设置为0.

/\*

  \* This delay should be sufficient to allow the power supply

  \* to reach the minimum voltage.

  \*/

mmc\_delay(10);

host->ios.clock = host->f\_init;

host->ios.power\_mode = MMC\_POWER\_ON;

mmc\_set\_ios(host);//这段code之前属于supply ramp up time阶段，这段时间内host->ios.clock被设置为最小值，我们在驱动中会设置这个值，这个值应该小于等于400KHZ.

/\*

  \* This delay must be at least 74 clock sizes, or 1 ms, or the

  \* time required to reach a stable voltage.

  \*/

mmc\_delay(10);//这个延时是等sd host给出74个clocks。74个clocks期间CMD线拉高，这个时候不能发送任何命令给SD卡。这个延时过后，系统就可以发送命令给sd card了。

}

（4）CMD0阶段

对应linux驱动中的code为：

int mmc\_go\_idle(struct mmc\_host \*host)

{

int err;

struct mmc\_command cmd;

/\*

 \* Non-SPI hosts need to prevent chipselect going active during

 \* GO\_IDLE; that would put chips into SPI mode.  Remind them of

 \* that in case of hardware that won't pull up DAT3/nCS otherwise.

 \*

 \* SPI hosts ignore ios.chip\_select; it's managed according to

 \* rules that must accomodate non-MMC slaves which this layer

 \* won't even know about.

 \*/

#if 0

if (!mmc\_host\_is\_spi(host)) {

  mmc\_set\_chip\_select(host, MMC\_CS\_HIGH);

  mmc\_delay(1);

}

#endif

memset(&cmd, 0, sizeof(struct mmc\_command));

cmd.opcode = MMC\_GO\_IDLE\_STATE;

cmd.arg = 0;

cmd.flags = MMC\_RSP\_SPI\_R1 | MMC\_RSP\_NONE | MMC\_CMD\_BC;

err = mmc\_wait\_for\_cmd(host, &cmd, 0);

mmc\_delay(1);

if (!mmc\_host\_is\_spi(host)) {

  mmc\_set\_chip\_select(host, MMC\_CS\_DONTCARE);

  mmc\_delay(1);

}

host->use\_spi\_crc = 0;

return err;

}

这段时间sd进入idle状态。

（5）CMD8阶段

sd host发送cmd8来check sd host端支持的电压 host->ocr\_avail是否被sd卡支持，如果sd支持的话那么在cmd8的response中会返回这些支持的值，并且返回写入cmd8 命令中的test\_pattern值。否则就表示不支持sd host的电压范围。对应的linux mmc驱动代码为：

int mmc\_send\_if\_cond(struct mmc\_host \*host, u32 ocr)

{

struct mmc\_command cmd;

int err;

static const u8 test\_pattern = 0xAA;

u8 result\_pattern;

/\*

  \* To support SD 2.0 cards, we must always invoke SD\_SEND\_IF\_COND

  \* before SD\_APP\_OP\_COND. This command will harmlessly fail for

  \* SD 1.0 cards.

  \*/

cmd.opcode = SD\_SEND\_IF\_COND;

cmd.arg = ((ocr & 0xFF8000) != 0) << 8 | test\_pattern;

cmd.flags = MMC\_RSP\_SPI\_R7 | MMC\_RSP\_R7 | MMC\_CMD\_BCR;

err = mmc\_wait\_for\_cmd(host, &cmd, 0);

if (err)

  return err;

if (mmc\_host\_is\_spi(host))

  result\_pattern = cmd.resp[1] & 0xFF;

else

  result\_pattern = cmd.resp[0] & 0xFF;

if (result\_pattern != test\_pattern)

  return -EIO;

return 0;

}

（6） ACMD41阶段

(1)这个阶段设置ACMD41的参数OCR=0，为了去获得SD卡自身支持的电压范围值。然后与 host->ocr\_avail，SD host所能支持的电压范围取交集，如果交集为空，则返回匹配失败。

int mmc\_send\_app\_op\_cond(struct mmc\_host \*host, u32 ocr, u32 \*rocr)

{

struct mmc\_command cmd;

int i, err = 0;

BUG\_ON(!host);

memset(&cmd, 0, sizeof(struct mmc\_command));

cmd.opcode = SD\_APP\_OP\_COND;

if (mmc\_host\_is\_spi(host))

 cmd.arg = ocr & (1 << 30); /\* SPI only defines one bit \*/

else

  cmd.arg = ocr;

cmd.flags = MMC\_RSP\_SPI\_R1 | MMC\_RSP\_R3 | MMC\_CMD\_BCR;

for (i = 100; i; i--) {

  err = mmc\_wait\_for\_app\_cmd(host, NULL, &cmd, MMC\_CMD\_RETRIES);

  if (err)

   break;

  /\* if we're just probing, do a single pass \*/

  if (ocr == 0)

   break;

  /\* otherwise wait until reset completes \*/

  if (mmc\_host\_is\_spi(host)) {

   if (!(cmd.resp[0] & R1\_SPI\_IDLE))

    break;

  } else {

   if (cmd.resp[0] & MMC\_CARD\_BUSY)

    break;

  }

  err = -ETIMEDOUT;

  mmc\_delay(10);

}

if (rocr && !mmc\_host\_is\_spi(host))

  \*rocr = cmd.resp[0];

return err;

}

(7) CMD2阶段

通过这个命令或者所有SD卡的 Menufacory id ，Product id等卡的信息。

int mmc\_all\_send\_cid(struct mmc\_host \*host, u32 \*cid)

{

int err;

struct mmc\_command cmd;

BUG\_ON(!host);

BUG\_ON(!cid);

memset(&cmd, 0, sizeof(struct mmc\_command));

cmd.opcode = MMC\_ALL\_SEND\_CID;

cmd.arg = 0;

cmd.flags = MMC\_RSP\_R2 | MMC\_CMD\_BCR;

err = mmc\_wait\_for\_cmd(host, &cmd, MMC\_CMD\_RETRIES);

if (err)

 return err;

memcpy(cid, cmd.resp, sizeof(u32) \* 4);

return 0;

}

（8）CMD3取得卡的相对地址，进入数据传输模式，点对点的传输。

int mmc\_send\_relative\_addr(struct mmc\_host \*host, unsigned int \*rca)

{

int err;

struct mmc\_command cmd;

BUG\_ON(!host);

BUG\_ON(!rca);

memset(&cmd, 0, sizeof(struct mmc\_command));

cmd.opcode = SD\_SEND\_RELATIVE\_ADDR;

cmd.arg = 0;

cmd.flags = MMC\_RSP\_R6 | MMC\_CMD\_BCR;

err = mmc\_wait\_for\_cmd(host, &cmd, MMC\_CMD\_RETRIES);

if (err)

  return err;

\*rca = cmd.resp[0] >> 16;

return 0;

}