L.E. Disk Software Development System
Available Command Descriptions

Introduction

This is the list of the currently supported commands for the LE Software Development System, revision S - 5. It supersedes the LE, LD-8, LC, and LB versions.

All commands are sent to the disk drive through a memory mapped interface, using locations $D700,$D701, and $D702.

$D700 is "Port A". It handles communications from the Z80 to the Atari.
$D701 is "Port B". It handles communications from the Atari to the Z80.
$D702 is the handshake & communications ports.

Two bits of $D702 can be read to determine communications status. The top bit indicates something has been sent on PORTB to the Z80, but not read. The nextmost lower bit (bit 6), indicates that something has been sent to the Atari on PORTA and not read.

All commands are sent the same way through a common Atari subroutine, called GENCMD. GENCMD handles the details of getting the command to the Z80, timing it correctly, and ensuring nothing gets snarled up (as is easy to do with two processors). GENCMD is used as follows:

```
LDA #(command #)
JSR GENCMD
```

And it worries about the rest.

At the termination of any command, the Z80 returns the floppy controller status to indicate the command is finished. Generally one sends in the command, via GENCMD, then waits for something to appear on PORTA -- which the Z80 has sent. Once something appears on PORTA, the command is done.

A subroutine called WTASET is generally used to determine this condition. It literally "waits until PORTA is set", or until the Z80 has sent data to the Atari. For instance, a typical command (to "restore" the disk to track 0:)

```
LDA #$10    ; restore command
JSR GENCMD  ; send in command, start it going
JSR WTASET  ; wait for completion
LDA PORTA   ; get status (optional).
```

Some commands involve transferring data to/from the Z80. The trick here is to not overrun the Z80; i.e., send a data byte before the previous one sent has been read. Some subroutines are handy here: WTBSET and WTBCLEAR. For instance, if we are doing a

sector read, we must send the track and sector #. So first, we issue the read sector command:

```
LDA #$10
JSR GENCMD
```

Then wait for the PORTB to be "clear" (i.e., Z80 is ready to read in the sector number),

```
JSR WTBCLEAR
```

Then, send in the track number:

```
LDA #TRACK
STA PORTB
```

Then wait for the Z80 to read in the track number:

```
JSR WTBCLEAR
```

Then send in the sector number..

```
LDA #SECTOR
STA PORTB
```

Similarly, WTBSET can be used to determine if the Z80 has sent something to be read, as when it transfers data to the Atari.

With all of this in mind, let's go through the command list.

$12 SETDN

Sent: Disk #: 1,2,3,4,5,6,7,8
Recieved: Status of that disk.

$00 -- disk is offline (no such disk on this system)
$DD -- disk is online and double density
$FF -- disk is online and single density

SETDN is used to select a given disk drive on the LE system. It has multiple functions.

First, it is used to determine if the LE handles the drive request. Let's say you do a sector read to disk #3. A SETDN is performed, and the LE system responds with whether or not it HAS a disk #3. If not, the request is then sent to the Atari disk #3. If it DOES have a disk #3, then whether or not disk 3 is in double or single density is returned for the Atari DOS to worry about.

Second, it is used to select which disk drive successive commands are active on. For instance, if you do a RESTORE ($10), it'll seek the drive number last selected out to track 00.

Third, it physically sets up the controller chip for single/double density.

Single/double density is determined at bootup time by the four position DIP switch; on is DD, off is SD. The lights at the right side of the board indicate DD/SD; on is DD, off is SD. You cannot change status midway through, because Atari DOS wll flip out if you do; it will try to copy 256 bytes into a 128 byte buffer, or other odd things. Just reboot. (Rebooting also resets the Z80, which then rechecks the 4-position switch).

Copy systems are not set up for double density. This is a special order item only for additional cost, requiring hardware changes.

SETDN also keeps track of whether or not a given drive has ever been accessed. If it has not, and it is selected, it must be RESTOREd to get the disk head to a "known" position. Hence, the first time a disk is selected, it will be restored. (This also cures the well-known "BOOT ERROR" message on previous ROMS). Remember that a STATUS command, issued by Atari DOS, causes a SETDN for each drive (the Atari is trying to figure out how many drives it has online), so the first time you press RESET,the drives may switch on momentarily down the line from 8,7, 6.. to 1.

Also note that Atari DOS 2.0S can only "really" handle 4 drives; the DOS menu loads into the area occupied by the memory buffers for drives 5-8. If you have an 8 drive system you need 2.0D or its equivalent (a la PERCOM).

SETDN returns a 00 if LE has no drive by that number. In the Atari ROM, this tells the DOS to ship this disk request off to the normal Atari disk SIO handler. SETDN returns a FF if the drive is on and in single density; in the Atari ROM, this tells the Atari to let us handle the disk request in SD. A DD means we'll handle it in double density.

$3A FREAD128

Fast Read 128 is a quick way for reading a SD sector.

Input: Track #, then Sector #

Output: Status byte (00 means ok, FF = failed)
then, if it went well, 128 bytes of data are sent through PORT B.

Normally the Atari disk ROM worries about this for any sector request. If you must do it by hand, send in the $3A with GENCMD, wait for B to go clear, send in the track number, wait for B to clear, then the sector number. The disk currently selected will go whirr, and something will show up on PORTA. This is a status byte. If it is a FF, the read failed, and the Z80 is waiting to do something else. (Probably a bad sector or something else is wrong). If it is a 00, all is well. Start reading, off of Port B, the 128 bytes of data. Do
something like this:

```
    LDY #0
LOOP
    JSR WTBSET      ; wait for a data byte to be sent
    LDA PORTA       ; get data byte
    STA BUFFER,Y
    INY
    CPY #$80        ; 128 bytes yet?
    BNE LOOP
```

After 128 bytes, terminate this command by raising the command line. Do this by sending a 3 to $D702. This tells the Z80 that all is done.

Note: having the drive door open or other such will just cause an ERROR-144. Atari DOS will usually issue four or five retries of any command.

$3C

Fast Read 256 Bytes

Reads a double density sector. Of course, you must have the drive selected as DD. Works exactly the same as FREAD128, just transfer 256 bytes out.

$3B

Fast Write 128 Bytes

Input: Track #
Sector #
128 bytes of data

Output: Status, of how it went (FF=bad, anything else =ok);

Normally the Atari disk ROM will do this for you if you issue a legal sector request through SIOV or DSKINV. But, if you want to do it the hard way, just ship it the track, sector, 128 bytes (waiting each time for B to be clear before sending a new byte, with WTBCLR),and wait. Whirrr. A status byte will pop up on port A, indicating how things went.

$3D

Fast Write 256 Bytes

Input: Track #
Sector #
256 Bytes of Data

Output: Status, of how it went (FF=bad, anything else =ok).

Same as FWRITE128, just 256 bytes instead, and, of course, you must be in DD.

$41

Format

No input needed.

Output: $FF if failed to verify the disk, 00 = ok.
Does not try to return #'s of bad sectors like the Atari does (or other
silliness).

Formats the disk with the LE "fast" pattern, every other setor. Reads in
every sector to verify the format worked. It seems to format slower, because it does a FORMAT, then read of every sector on a track - 3 passes. It also carefully times the track to track interleaving so there is minimal delay between reading sector 18 on one track and sector 1 on the next track, something else Atari forgot about; it makes 6 seconds difference reading in the whole disk just by straghtening that out!

Atari DOS retries a format four or five times, ERROR-173 if it fails.

Trying to format a write protected disk doesn't work.

Formatting a disk with the drive door open can hang the system; there is no room in the output loop for a check of such things. Please, don't.

Note that our formatting pattern will run a bit slowly, like the old "B" rom format, on an Atari drive, and an Atari formatted disk will run just a tad faster on our drive (no transfer time to speak of). Also note that a very few copy protected disks will not boot on our drive because it is so fast;they rely on timing patterns, and our drive gets sectors back too fast. For your information, we retrieve every other sector on the disk (i.e., 1,x,2,x,3), rather than about every 9th (like the Atari). We could retrieve every one, but Atari DOS is not that fast, plus screen refresh kicks the processor off every 1/60th second anyway..

$42

Double Density Format

Same as SD format, except it does it for double density. The interleave pattern is every four sector (i.e., 1, x, x, x, 2, x, x, x, 3..) not because we can't transfer the data fast enough, but because Atari DOS can't. (Typical). Anyway, it runs around 10 times faster than PERCOM DD anyway. The two formats, PERCOM and ours, are compatible; our disks will run in theirs, but theirs will run fairly slowly in ours (at their usual speed).

RESTRICTED RELEASE COMMANDS:

$39

Fast Write 128 Bytes Deleted Sector
This works the same as FWRITE128, except the data address mark is written as an F8, not an FB which is normal. The 1771/1797 controllers call this a "deleted sector". It is NOT a bad sector! It is an extremely devious form of copy protection.

After any sector read on the Atari disk drive, a status byte is set. The $20 bit (bit 5) is set to 0 if the sector was normal, an a 1 if the sector is "deleted". (Apparently this feature was added to allow you to delete data at the disk sector level, but isn't used much). Anyway, in the course of Atari disk reads, if you issue a status command, you get that sector byte, along with three others,returned to you in DVSTAT (see the hardware manual). So what you do for copy protection is read your special "deleted sector", then do a disk status, and look for the deleted sector bit. SPECIAL NOTE: Everything in the Atari disk drive is inverted, so the status will come up as a 0 (zero) for deleted sector mark and a 1 for a normal sector. Anyway, if you don't find a deleted sector status bit, someone has cloned your disk.

What's neat about all this is that even though the Atari reads the sector and gives an ERROR-144, it still sets this special bit that is set under no other circumstances. I do not know of any Atari drive but ours that can write these deleted sector marks.

Note that data seems to be returned by the 810 disk from a deleted sector, even though there is an ERROR-144 happening. This is interesting, and should be investigated.

$3F

Write Half Sector (Generate CRC Error)

A quick way to trash a sector.

Input: Track #, Sector #.

OUTPUT: Always $FF.

All this does is write 64 bytes to a given sector, then interrupt the controller out so that the CRC bytes do not update. When the 810 reads this sector, bango -- error 144.

You can check and make SURE it is a CRC error by looking at the floppy status register. Bit #3 ($08) is 1 (invert that to a 0) if a CRC error happened. Remember, error, 144's happen from all sorts of causes; drive door not open, write protected disk on write, and so on, and for copy protection with bad sectoring, you realy should check to see that a CRC error happened.

I do not recommend using CRC bad sectoring only. Too many pirates can zap sectors with ease; many have mounted switches to trash the "write data" line on their disks, for instance, to kill a given sector.

Also note a timeout error (144) can be generated by having no-such-sector in the sector interleave pattern, using custom sector patterns. But that won't return a CRC error, so you can determine the difference.

As a practical note, everyone seems to think that a bad sector is a bad sector is a bad sector, whereas there are many different types of bad sectors, all detectable, that can be differentiated. Then, when your average pirate goes and makes a generic bad sector, generally a CRC error, you can catch him at it and reformat the disk (or something horrible.)

Also, it is a VERY good idea, when you catch a pirate in your program, to let the program "boot" to its title page. That way, they think they have a running copy. PREPPIE! and Threshold do this and have frustrated many folks.

$3E

Write half sector (CRC error) with Deleted Sector Mark.

Combination of the above. EXTREMELY hard to break. Check the status register to make sure that both a deleted sector and a CRC error occurred. Both bits 5 and 3 should be 0 (remember, 1771 is inverted data bus, so the bits are 1, but we seem them as 0).

$48
SPFORMAT

Lay out a special format on one track of your disk (SD only).

Input:
Track #, then 18 sector #'s in the interleave pattern you want.

As usual, use WTBCLEAR to not overrun the Z80.

Returned: 00 if ok, FF if it died (write protected or something).

This reformats the specified track with a standard format, using the sector interleave indicated. This has endless copy protection possibilities. For instance, you can fill a track with multiple sector #1's, then read between them to determine you get different data from a read of a same sector # -- hence, not a pirated copy, because an 810 cannot generate duplicate sector numbers. Or, you can do obscure and buzarre timing schemes based on a given interleave, where you read THIS sector, then THAT sector, and then THIS sector should should show up in a certain amount of time, or it is a pirated copy.

I'll leave the deviousness up to you; the tool is there to use.

Also,you can use this to generate Atari "C" format disks on the LE drives, sinc your production disks probably don't want our "slow" interleave pattern on them.

There are two support programs for this one. One is just a sample one-track reformatter. The other formats a whole disk to Atari "C" format.

Don't forget to write the directory, etc! When you format a disk, DOS must lay out the VTOC, boot tracks, etc.

One Extremely Devious Scheme: Lay out a few tracks with this, delaying appropriately, so that there is a definite lag between readinga sector on one track, switching tracks, and reading another sector. Make this delay very different than the Atari delay. Then, when booting, read back and forth a few times, and make sure the delay matches what you put in. This would be difficult to impossible to break for most pirates.

$50
Track Read

Requires Support Program

Input: Track #
Output: Gobs of Data

This does a full track dump, some $1E00 bytes worth, of all the bytes on the track. This includes sector marks, CRC, index marks..you name it. It is the most powerful disk examining utility possible.

This one has a short program associated with it to handle snarfing the data off the Z80 and stuffing it into the Atari's memory. To use, load the track dump program, place the track # you want to dump in byte $4FFF, and run at $4800. The disk will spin briefly as the track dumps, then from $6000 to $7E00 will be your track data.

Note that $1E00 is more than one revolution of the disk--the overlap is deliberate.

Track dumps are extremely handy for checking your interleave schemes, deleted sector marks, and so on. I strongly recommend you get a 1771 data sheet to understand what's on the track if you don't already; disks are a whole science by themseleves.

Track dump works in DD.

Track dumps work on ANY Atari disk, perdiod. Please be discrete with this tool; it cannot be protected against.

Note that track dumps are not always 100% reliable, particularly when the floppy controller is resyncing. In particular, the FE sector mark sometimes ends up as CE.

$60
Read Address Marks

This is a lot like a read sector command, except it spins the disk, and for a given track, returns you all the sector mark data. The sector mark data is the track #, side #, sector size, sector #, and CRC -- 6 bytes. The neat thing is that this returns you the interleave pattern fast, easy, and reliably.

Send in the $60, Track #, and (useless byte) sector #. (I used many parts of the FREAD128 command to save space, hence the sector # is included). Returned will be a FF if it bombed, else anything, then 128 bytes. The 128 bytes will be 6 x 20 (120) sector marks, then 8 useless bytes.

This has all sorts of potential, presently unexplored. Right now it is a fast way of giving you the interleave pattern of a disk, but that could be used (for instance) to optimize the verify process on the supercopy system, for instance.

$51 -- COPYINIT
$52 -- SUPERCOPY

These commands trigger off the supercopy (analog copy) process. The commands must be driven by an applications program, so there's no use getting into the guts of how they work here.

Quick Reference To Commands:

$10: RESTORE
$12: SETDN, input: drive #.
$3A: FREAD128, input: track #, sector #, output: status, 128 bytes
$3C: FREAD256, ""   256 bytes
$3B: FWRITE128, input: track #, sector #, 128 bytes
$3D: FWRITE256, ""   256 bytes
$41: LFORMAT
$42: LDDFORMAT, dden format
--
$3F: Write half sector (CRC error)
$3E: Write half deleted sector (CRC error, Deleted Sector)
$39: Write Deleted Sector (input same as FWRITE128)
$48: Special Track Format: Input: track #, 18 sector #'s
$50: Track Dump
$60 Read Address Maks
($51: Copy Initialize)
($52: Copy)
--
$D700: PORT A   Disk to Atari
$D701: PORT B   Atari to Disk
$D702: PORT C
Bit 0: =1 to enable SIOV trap, =0 to disable trap. (Leave 1.)
Bit 1: Command line. High to get Z80's attention, low to execute cmd.
Bit 6: Unread data awaiting on Port A (for Atari)
Bit 7: Unread data awaiting on Port B (for Disk)

Other Misc. Notes:

Use lots of bad sectors. They are very irritating to pirates because they take so long to copy, because it takes the 810 about 20 seconds to figure out a sectors is bad..not even counting retries. And writing them is a pain, too, unless they have some sort of super-sophisticated chip. So blank fill your disk with bad sectors.

BURY your copy protection code. Be devious. Remember, if someone can crack your disk code, they will produce an unprotected copy, and it is the experience of most disk manufacturers that the unprotected copies are the big piracy problem. Doing a disk access every now and the is a good idea, for instance, to write a high score. Maybe force the user to remove the write protect tab ("to write the high score") if you detect pirating, then reformatting the disk..or anything. But don't place the code near the start of the program, don't leave it "legible" (i.e., XOR it with something, then have it de-XOR'd and executed at the very minimum). Also, interrupts can be very helpful--you can have code magically appear where a pirate's trace will miss it.

Good luck, and if you have any questions, I am available M-F 8-5, Central
time, to answer them.

Dave Small