Disk Co-processor Card

Version 3.2a
March, 1999

Applix 16 16 Project

Applix Pty Ltd

1 read. me - very i nvitin g file to read.

The purpose of this file is to tell you to PLEASE, PLEASE, PLEASE read DC.DOC. It contains lots of information that you NEED to know about my SSDCC software.
My version is lots more intelligent than Applix's.
The Applix version that I have is currently V1.4a; Any references I make will relate to that version.
You may have to restrap your drives to work best with my version; you may also need to modify your controller card slightly (to stop "motor on" from gating "drive select").

read. 1- me - 1 very in viting file to read.

Your system should still work with Applix's version with these mods. It's all explained in DC.DOC.

Have a look at LOADDC.SHELL for an example of how to load DC.CMD into the SSDCC.

Please proceed to DC.DOC!!!

Greyham :-)

(happy disk driving)

1- read.
2 me - very in viting file to read.

2 dc.doc - Info about Greyham's ssdcc controller software. Version A.4e 12-1-89

The software described below is COPYRIGHT, but FREEly distributable. See copyright.doc.

Greyham might possibly maybe be able to be reached as: greyham@hades.nucleus.oz

The whole thing was written in time I didn't have.

2.1 Terminology

‘unit’ and ‘drive’ are synonymous. ‘block’ and ‘sector’ are synonymous. ‘drive characteristics’ and ‘drive parameters’ are synonymous. ‘sector per track’ considers one side of the disk only. ‘tracks’ is not affected by the number of sides.

2.2 Introduction

This version of the ssdcc controller software was written entirely in Z-80 machine code by Greyham. The code is available in EPROM, or as a core-image file that you can burn into EPROM if you have a programmer

available. You can also load the code temporarily into Z-80 RAM. This is handy because you can then run FORMAT.EXEC, DOSINIT.EXEC and DOSPUT.EXEC to write the EPROM core-image onto an MSDOS disk for EPROM programming.

2.3 EPROM Version

An EPROM image of the software is contained in /HEX. Simply burn this file into a suitably sized EPROM and replace the EPROM U3 on the SSDCC card with it. The EPROM should be 200ns at least. 250ns seems very unreliable.

Note that the EPROM size must be selected by a DIP switch on the SSDCC. See the 1616 manual for details.

If the SSDCC ever crashes (never!) the LED flashes vigourously. Just like the original.

When the system is started up with the EPROM installed, units 0 and 1 are /F0 and /F1 (as usual); units 2 and 3 are /H0 and /H1. SS/OS actually thinks these are hard-disks, so when booting you will experience a long delay after the Controller-Card version number is displayed on a level 0 reset as the SS/OS thinks it is waiting for a hard-disk to spin up. Don't be alarmed.

Booting will be much faster if you put a bootable disk in one of the drives.

There are 2 EPROM Versions: DC___.HEX and HD___.HEX. The HD version supports SCSI Hard disks; though it uses a different message number than the ROMs or the original SCSI EPROM did. The HDDVR.C program in /MRD is an MRD that replaces /H0 and /H1 and provides /H2, /H3.... for as many partitions as you have. More details on SCSI when I know what it's doing.

2.4 CMD File Version

The program is also available as a

TRSDOS .CMD file that must be loaded into the Z80 after EVERY reset, since the SSDCC will go back to its ROM program on a reset.

PLEASE NOTE: Alt-Ctrl-R or RESET will result in the loss of everything in the write cache!. If you have write caching on, you MUST wait for the automatic 'sync' to occur (either that, or run 'sync' yourself) before any level of reset, else you will corrupt your file system!. The autosync will updates the disk after the SSDCC is given nothing to do for about 1 second. In particular, DON'T RESET DURING DISK I/O!!

.CMD files can be loaded into the Z80's RAM space via RECVCMD.EXEC. The shell file LOADDC.SHELL, which runs LOADDC2.SHELL, will load the program for you.

During loading of the program, NO DISK I/O can occur; because I have no idea which regions of the SSDCC's RAM are used by it in normal operation. A VERY limited version of the disk controller software, that responds ONLY to "read Z-80 RAM", "write Z-80 RAM" and "call Z-80 program" commands called DCLIM.CMD loads into Z-80 RAM and executes to "boot-

strap" the main code into RAM. Note that all this download business must be done from the RAM disk, to prevent Disk I/O.

The .CMD file version is not needed once the EPROM is programmed, and does not support the "Fast-Copy" or "Read Sector ID" commands.

2.5 RAM Usage

The current version automatically detects how much RAM your SSDCC has installed; certainly you must have at least 8K, and then more RAM means a bigger cache. You can install any amount of RAM from 8k to 64k. So long

as you have a RAM chip in U1, the SSDCC should find the RAM OK. With only 8k, fastcopy might not work; it will return an "Out of Cache Memory" error when it runs out.

2.6 Error Retries

Certain types of disk errors are automatically retried, and the 68000 notified only if the retries) fail. Only a few errors are considered retriable, on the basis that most of them aren't fixed by simply trying again.

Error retries are as follows: Seek error: the drive is restored, and the seek retried twice. CRC error: the sector

transfer is retried four times. RNF error: If the drive is on track 0, error is declared immediately since the 1772 only reports the error after 5 disk revolutions anyhow. If not on track 0, the drive is restored, and the transfer retried once. RNF errors are usually some sort of seek error, which will often be declared as a seek error during the seek after being restored.

All other errors are declared immediately that they occur.

2.7 Drive Characteristics

Nine logical drives are supported by allowing unit codes 0 to 8. How-

ever, the trouble is convincing 1616 OS to pass such a unit code. By changing the drive select bit-map, you can have one physical drive respond to more than one logical unit, and give each different characteristics. Disk drive characteristics may be modified via the "set drive characteristics" message. Any program that changes drive characteristics should read the current characteristics with "show drive characteristics", change the desired ones and write them back with "set drive characteristics". "disk change method" in particular is hardware dependant, and should not normally be altered.

Variable characteristics are:

* step rate * double or single stepping between drive tracks. * drive select LATCH bitmap. * sector size. * sectors per track. * number of tracks. * sides/heads * caching level. * disk change method.

Each drive has its own set of characteristics, so that any mix of different drive types is allowed.

2.8 Step Rate:

This defines the "step rate" field that is included into all "type I" (seek, step and restore) commands to the FD1772. Defaults to 6ms.

2.9 Double/Single Step Between Tracks:

Normally, a single stepping pulse is issued to step the drive between tracks. This option allows a double-stepping pulse to allow 40 track disks to be read in an 80 track drive. Some care should exercised here; the tracks on a 40 track drive (48tpi) are logically twice as wide as the tracks on an 80 track drive (96tpi). This can lead to problems - reading the 40 track disk in the 80 track drive is ok; but if you write back to the disk, you may not be able to read it in the 40

track drive again. This varies a LOT depending on different drives; even drives of the same type. If you have two drives, you may find that one works better than the other. Also, formatting seems a lot more critical than normal writing; if you can get the disk formatted in a 40 track drive you'll probably have less problems.

Another thing; formatting normally destroys all information on the disk, but bear in mind that if you reformat a disk that was formatted at 80 tracks to be 40 tracks on an 80 track drive, only the even numbered tracks will be rewritten. This is no real problem, unless your disk contained

very private information (half of which will still be there, on the odd tracks); but keep it in mind because it can explain some very odd results. For instance, if you accidentally put your "40 track" disk in the 80 track drive and forget to tell the SSDCC to step twice you'll find that sector reads to odd tracks won't give seek errors, but could be accessing all sorts of weird data. Of course, an error will be thrown out when it tries to read an even track.

One other thing; track 0 is always track 0; so you can't immediately tell if you have double stepping set correctly just by looking at the directory

(which is normally only on track 0).

2.10 Drive Select LATCH bitmap:

This defines the bitmap used to select a given drive. This bitmap is ORed into the latch to select the drive, and its complement is ANDed to the latch to deselect the drive. This allows different physical units to be assigned to different logical drives. Eg: have unit 1 as /F0. Also allows extra drives to be selected if you have some sort of LATCH bit pattern to select more than 2 drives. Be super careful though; having more than one

logical drive select a single unit may corrupt disks if you have caching enabled for either logical drive!

The SSDCC never reads from the hardware latch; it stores the current setting in software. Particularly, you can rewire your SSDCC to use EJECT and INUSE as drive selects, even though the Z80 can't read their current setting.

2.11 Sector Size:

The controller CAN read sectors of different sizes. The FD1772 supports 128, 256, 512 or 1024 byte sectors, and must be informed of what the sector size is

on the drive. If more or less than this number of bytes are requested on a data transfer, you'll get a sector size error message - this should not be relied on however; in the case of a write, your disk will be corrupted since the controller can't determine the actual sector size until the data has been written. Note that 1616 OS ONLY allows 1024byte sectors!!!. The standard "block read" and "block write" commands are used with any sector size, the difference being the number of bytes passed (which the 68000 MUST know). So what does it all mean? Well, DO NOT tell

1616 OS to try to read or write a logical drive whose sector size is anything other than 1024 bytes!!!. ALL transfers of different block lengths MUST do it by talking to the SSDCC directly.

2.12 Sectors per track:

The controller is only ever passed block numbers, not sector, track and side numbers, so it has to know how many sectors are on each track. This is defined as the number of sectors on a single side of the disk. Eg: for normal 1616 OS disks, it's 5. The actual track number is found as follows:

block number track number = - number of sectors per track * number of sides

1616 OS could theore tically work with other than 10 sec- tors per track; but the format com- mand would spew because SSDDUTIL passes a 10 byte sector skew table. The format command SHOULD only be passed the right number of sector skew bytes; but of course 1616 OS assumes 10 per track.

2.13 Tr acks:

The con- troller limits all disk requests to within the valid track range to avoid seek- ing a track which is physically beyond the

drive's capabilities. This value is also used by the 'fastcopy' command to work out how many tracks to copy.

2.14 Sides/Heads:

Defines the number of sides or heads on the drive. The only sensible values are 1 (single sided) or 2 (double sided).

2.15 Caching level:

Yes, that's right; the controller does disk caching. The caching level sets how much caching is done. Caching can corrupt your disks REALLY efficiently if you aren't

just a little extra careful.

No Caching: This is the default. No caching occurs what soever. This is most definitely the safest mode. It is also the slowest.

Read Caching: (write through) Blocks go via a cache when they are read. If that block is read again, it is sent from the cache, rather than bothering with the disk drive. When writing, blocks go into the cache and are immedia tely written onto the disk, with any error code being returned to the 1616, which must wait for the error code.

Write Caching: Blocks go via the cache for all reads and writes. Write errors can never occur in this

mode, as the block simply goes into the cache and the 68000 is told that no error occurred; it trundles off and does whatever it likes, and the cache can be flushed to disk later. This has a problem in that if a persistent error occurs when the sector is eventually written to disk, there is no way to tell the 68000, and the block is simply expelled from the cache. I guess that's the price you pay for the extra speed - personally, I've NEVER yet had a data error that wasn't caused by something I did wrong so it isn't as big a problem as you might think.

After disk change is detected (either automatically, or by issuing a

"sync") the first block to be written to disk is always flushed onto the disk and the error code returned. If the disk was not write protected, then the write cache (if enabled) can be used from then on, until the disk changes again.

The caching algorithm keeps track of how often each block from any drive is accessed by keeping an array of 800 entries. When a block has to be expelled from the cache to make room for another one, the least used block is the first one to go. Maximum cache size with 64k is still less than 10% of the average disk, so knowing which blocks to keep in the

cache is pretty important. Blocks that need to be flushed to disk are always higher priority than blocks that have merely been read, since the SSDCC can write blocks at its leisure.

All blocks that have to be written to disk are written just before the SSDCC turns the drive motors off. Hence, it is now essential that you do not remove ANY disk from ANY drive whilst the motors are on. Also, you have to run a 'sync' to flush the read cache whenever you change disks, unless your drive can detect disk removal.

Special note: Setting drive characteristics (in particular, disabling

caching) AUTOMATICALLY runs a 'sync' for that unit.

2.16 Disk Change Method/RDY Signal:

The SSDCC can detect when the disk in any drive has changed, and flush the cache accordingly. Some drives can detect disk change, some can't. Three different methods are allowed: by a special RDY mode, by the DISK CHANGE signal, and by DISK CHANGE on the RDY signal. There is also a special booting mode which does not rely on the RDY signal and allows the system to boot with drives set up

for any disk change method.

0 : Detection not possible, and the meaning of the RDY signal is unknown. This is the default, and should only really be used for booting. If your drives can't do disk-change by any method, your auto-exec file should set to method 1, since it's slightly faster. It becomes necessary to 'sync' the drive before removing it's disk IF you use READ or WRITE caching. No caching doesn't require the sync. The reason for this mode is that RDY won't be valid if it's actually connected to DISK-CHANGE for method 4.

1 : Detection not possible, and RDY signal indicates drive readiness. As with method 0, disk change detection can't be done. You MUST sync when swapping disks if using the caching. The only difference is that since RDY is known to indicate drive readiness, the controller doesn't need to do a speed test every time the drives are turned on.

2 : Detection by RDY signal. The RDY signal goes active when the drive is FIRST selected, then stays active until the disk is removed (even while the motor is off!). Note that this is NOT the normal use of RDY, and will probably require

restrapping of the drive (if the drive is capable of it!). This means that the RDY signal no longer indicates that the drive is up to speed - RDY can be active even if the drive motor is off, so the controller will do a rotation timing test to find when the drive is up to speed.

3 : Detection by Disk Change. The DISK CHANGE signal indicates a change of disk. This is the usual method. The DISK CHANGE signal from the drive goes active (low) when the disk is ejected, and remains active until a 'Step' instruction is issued to the drive.

4 : Detection by Disk Change on RDY signal. The RDY

signal acts exactly like DISK CHANGE, as described above. As with detection by RDY signal, a rotation timing test is done to find when the drive is up to speed.

All outputs from the drives are enabled only when that drive is selected, and the SSDCC must scan all of the drives while it isn't busy, to see if any disk changes have occurred. Normally, the DRIVE SELECT outputs from the card are gated by MOTOR ON. Using ANY disk change method requires you to modify the card so that the drive select signal can be asserted without the motor being turned on. Otherwise, you'd have

to turn the drives on and off several times a second. This is not nice. Note that if your drives have "head load" ability, this should be done on the MOTOR ON signal, and NOT ON THE DRIVE SELECT SIGNAL ALONE; otherwise your poor headload solenoid will go on very briefly every time the SSDCC scans the drive.

Unless your drive's data says that the drive can do disk change by the special RDY signal, or via DISK CHANGE, you'll just have to use mode 1, and run 'sync' when you remove the disk if you want to use caching.

If the SSDCC detects a disk change while stuff is still in the drive's write cache, it will flash the drive LED and wait until you put the disk back in, then write everything to disk. NOTE: This is a SAFEGUARD that should NOT be relied upon!!!. Obviously, if your drives can't do disk change detection, this will rarely occur. Also, the controller can't possibly check that the disk you put in IS actually the one that it wanted.

The SSDCC may miss a Hold-RDY type disk-change signal if the disk is changed while the controller is busy servicing another drive. You should NEVER

remove a disk from the drive while ANY drive motors are on, or while the 68000 is accessing disk blocks cached in the Z80!

2.17 Error Messages

The error messages produced are a lot more comprehensive than normal. Error messages that correspond to those listed in the manual usually have the same numbers, but this should not be relied upon. When an error code is produced, the "error message" command should be run to get a textual description of the error message, which should then be displayed

to the user. This is how 1616 OS works, so it will normally display the expanded error messages. The error message text often contains numbers relating to what actually happened (Eg: drive and sector numbers). These numbers are converted to ASCII by the Z80, so the 68000 doesn't need to worry about them; it merely displays the string to the user. Error messages should be sought immediately after receiving an error code; only the error code that was returned will give a sensible error message. The parameters returned by many error messages are stored in the same

place by the SSDCC, so interrogating an incorrect error message will yield misleading results.

Error code numbers may change any time, so they aren't given here. They should never be displayed to the user either, since they are not very meaningful.

Bad error number:

The error number passed to "error message" command was invalid.

Seek Failure On Unit x Track y Side z:

The controller could not find a data record matching the correct track number following a seek, despite retrying. Probably either your drives can't handle the

step rate you've specified, or the disk is 40 track and the drive is set up for 80 track.

Controller Jammed, But Interrupted OK:

The FD1772 latched up for some reason, and didn't return to NOT BUSY state. A "Force Interrupt" command fixed the problem. This will occur if the FD1772 hangs up for some reason, such as the drive door being opened midway through a data transfer.

Controller Jammed, And Will Not Respond:

The FD1772 latched up for some reason, and didn't return to NOT BUSY state. A "Force Interrupt"

command failed to fix the problem; the FD1772 controller has failed completely for some reason. This is likely to be due to a faulty controller chip.

Read Record Not Found On Unit x, Track y, Sector z: Write Record Not Found On Unit x, Track y, Sector z:

The drive controller could not find a valid data record for sector z on track y when attempting to read or write despite retrying. The controller's impression of how many sectors are on each track is probably wrong.

Read CRC Error On Unit x, Track y, Sector z: Write CRC Error On

Unit x, Track y, Sector z:

The FD1772 found that the computed CRC did not match that written on the disk, despite retrying. A data error; bits have been dropping off your disk. I don't think it's actually possible to get a write CRC error, as the controller does not do verify reads; the 1772 may declare one if the ID field CRC is wrong though - I'm not sure.

Format Track Too Small On Unit x Track y Side z:

The track simply doesn't have enough space to hold the required number of sectors of the required size. Either sectors per

track or sector size is wrong.

No RDY Signal From Unit x:

The RDY signal from the selected drive did not become active when the drive was selected. The drive select strapping is probably wrong, or the drive door simply isn't closed.

Unit x Is Write Protected:

A write was attempted to the drive whilst it was write protected.

Bad Unit Number:

The unit number passed to the SSDCC was invalid.

Read Sector Too Small On Unit x, Track y, Sector z: Write Sector Too Small

On Unit x, Track y, Sector z:

The sector on the disk was found to be smaller than the SSDCC expected. The error is detected when the FD1772 asks for or provides less bytes than were expected. In the case of the write error, data has already been written on the disk before the error is declared.

Read Sector Too Big On Unit x, Track y, Sector z:
Write Sector Too Big On Unit x, Track y, Sector z:

The sector on the disk was found to be larger than the SSDCC expected. The error is detected when the FD1772 asks for or provides more bytes

than were expected. In the case of the write error, data has already been written on the disk before the error is declared.

Rotational Failure On Unit x:

The disk in the drive was either not rotating at all, or was rotating FAR too slowly; around 100RPM. Normal rotation speed is 300RPM.

Unit x Too Slow: Unit x Too Fast:

The drive was found to be rotating too slow/fast. Rotation speed tests are only done when the drive motors are turned on, and then only if the RDY signal does not indicate that the drive is up to speed.

SSDCC Internal Stack Failure!:

I just couldn't resist - this is to make MSDOS users feel at home. (Be thankful it doesn't print "system halted" too!). No, seriously; stack over- and underflow is tested for in the code for debugging purposes. This is extremely major; I hope you don't ever experience it. If it does occur, the SSDCC will reset itself (but still be running my software; not the ROM code.) All cache entries and drivparm's will be lost. PLEASE contact me!.

Source And Destination Diskette Characteristics Not Identical:

A fastcopy was attempted whereby at least one of the tracks, sectors per track or sides characteristics for the source and destination units were not identical. This makes a mirror image copy impossible.

Bad Block Number:

The calculated track number for the specified block was greater than the number of tracks on the disk. Either you've blown it pretty badly, or you haven't told the SSDCC how many tracks are actually on the disk. The SSDCC won't even attempt to seek to tracks that are greater than the number of tracks specified in the drive charac

teristics. This error often occurs under 1616 OS if your boot block is corrupted.

Lost Data!:

The Z80 didn't feed data to or from the WD1772 fast enough. A programming error (ie: one of mine!) that should NEVER occur.

Format In Progress On Unit x:

A background format on unit 'x' prevents your request from being serviced. Only one drive may be formatting at any given time.

Invalid Drive Characteristics For Unit x:

Drive characteristics passed to the setchar command were invalid.

Out Of Cache Memory:

You haven't installed sufficient RAM for fastcopy to operate. Install some more!.

3 Interprocessor Communication

All messages in the 1616 OS manual are implemented. The first byte of every command determines what the command is, and is sent as a COMMAND byte. All other bytes are sent as DATA bytes. If the controller gets the wrong sort of byte at any time, it may get confused. In serious cases (such as getting bad parameters to a 'format' command), it will deselect all drives and turns the motor on for about 2 seconds, before resetting. The syntaxes below MUST be followed - do not even THINK about trying to abort a command in mid-messag

3-1 Interprocessor Communication

e!!! If you really MUST abort a command, complete the message then send an "Abort" command (command 00). Also note that interrupts can occur on the 68000 in mid-message; so Interrupt routines may NOT send messages to/from the Z80!!!!

3.1 Abort command: 00

The command currently in progress is aborted. This can be used to halt 'format' and 'fastcopy' commands prematurely if something totally disastrous goes wrong, or the operator wants out. Won't abort a background format.

3.2 Block read command: 01

unit blockhigh block low <errorcode or 0> <data>

Blocks of all different sizes are read with this single command. The number of data bytes returned is determined by the sector size, and the 68000 MUST be ready to accept exactly the correct number of bytes. If 1616 OS tries to read a disk of a different sector size, it will expect 1024 bytes, but not get them, and timeout. Data is returned ONLY if a 0 errorcode is returned.

The block number is a number from 0 to whatever, which specifies which block to read. The SSDCC calculates which actual track and sector it

3- 3 Interprocessor Communication

corresponds to; so unless the SSDCC knows correctly how many sectors are on a disk, you wont get the right one.

3.3 Block write command: 02 unit blockhigh block low data <errorcode>

The logical complement to the block read. The 1616 waits for an error code before proceeding; however, if write caching is enabled, this command will ALWAYS return a zero error code, except for the first sector written to disk, to check that the drive is ok and not write protected. (It's not possible to have an error writing the block into the

cache!). If an error occurs when the block is eventually written to disk, there is no way to notify the 68000, so its bad luck. If write caching is not enable (ie: no caching or read caching only), then block write will wait for the write to complete, and actually return the error code indicating if an error occurred.

3.4 Error message command: 03 errorcode <string> <0>

Converts the arbitrary error codes returned by various commands into ASCII error messages. Do not interpret SSDCC error codes yourself, as they are very likely to change. Use this command!

3.5 Format command: 04 unit $B5 $7E nt racks skewtable <errorcode or 0>

Writes format information to every track on the drive. Note that this is not what it says in the Applix manual; they got "unit" in the wrong place. The organisation of the disk is dependant on the drive characteristics, allowing a very wide variety of disk formats to be used. The number of bytes in the skewtable will be the number of sectors per track, times the number of sides on the drive. Because of this, BLOCKDEV can only cope with this command if the drive characteristics are as per a

normal 1616 OS disk. The characteristics for the specified unit will be modified to have ntracks tracks.

There are two main disk formats that the SSDCC will generate: IBM and SONY. The only apparent difference is that the IBM format has an Index Address Mark (IAM) just after the index hole, whilst the SONY does not (it doesn't fit with 5 x 1024byte sectors per track - Applix uses the SONY format). All SONY compatible hardware will read IBM format, but IBM hardware won't necessarily read SONY format (the NEC uPD765 used in IBM PCs is one example). Format attempts to include the IAM: If an error occurs, it tries the format again without the IAM - if the

3-7 Interprocessor Communication

error still persists, it is declared. It is theoretically possible for the IAM to fit on track 0, but not fit on another track (it would have to be very borderline!) - the entire disk will be reformatted without the IAM.

If write-caching is enabled, the format will proceed in the background. The Z-80 waits for the format of track 0, side 0 to complete and returns an error code accordingly; if no error occurs, the format continues in the background. Normal disk I/O is not prevented during the rest of the format. Note that during a background format, NO automatic sync'ing is done (on any drive; except for the normal flushing of one block when another is needed and the cache is full) and manually syncing the

drive to be formatted will yield an error. In this way, BLOCKDEV may format and initialise a diskette - the initialisation info will sit in the cache until the format completes.

3.6 Type II Format command: 05 unit $B5 $7E track side doIAM skewtable <errorcode or 0>

Applix have officially abandoned this command, so I've defined it my way. It formats a single side of a single track in the same way that the Format command does. doIAM is a flag that indicates whether an

IAM should be written after the index hole. This is generally a good idea, although it might not always fit. Type II Format doesn't retry without it if it doesn't fit (like Type I Format does).

The size of the skewtable will be the number of sectors per track. Note that this is different to type I format, where the skew table covers all sides.

Note that during type II format, there is no way of the controller really being certain that the drive head is where it thinks it is, since it can't do a verify on a track that has yet to be formatted. Track 0 is an exception, since a restore is done; guaranteeing track 0. This should very rarely be a problem, but it means you must be careful if

switching the 2step setting on a drive - take extra care to get it right.

3.7 Read Z80 RAM command: 07 Z80addrh Z80addrl lengthh lengthl <data>

Reads Z80 RAM. Just like in the manual.

3.8 Write Z80 RAM command: 08 Z80addrh Z80addrl lengthh lengthl data

Writes Z80 RAM.

3.9 Call Z80 program: 09 Z80addrh Z80addrl

Calls a program in Z80 RAM. Not a good idea, as there isn't any free RAM in the Z80's space. Not with caching enabled, anyhow. This can be useful for resetting the SSDCC (going back to the ROM version) without resetting the 1616, by branching to location $0000.

3.10 Read Z80 ROM version command: 0A <ROM version>

My SSDCC software versions started at version A.0

to distinguish it from Applix's.

3.11 Announce 1616-O/S version: 0B SS OSversion

Tells the SSDCC what version of 1616 O/S is running on the 68000. This is ignored by the SSDCC at present.

3.12 Set floppy disk step rate: 0C unit rate

This is included for compatibility with the normal SSDCC ROM. Floppy disk step rate can also be set with the "set drive characteristics" command. Either method has the same

effect. The default is still 6ms, as per the manual.

3.13 Enable/Disable Z80 Interrupts: 0D flag

The Z80 can be made to interrupt the 68000 after completion of any operation; which is useful for running multitasking O/S on the 68000. SSO/S doesn't use this at all; but if flag is non-zero, the Z80 will assert EIRQ1 EVERY time it has an error code (zero or non-zero) to send to the 68000; this signals the completion of the requested operation.

3.14 Set LATCH bits: 12 bitmap

The Z80 hardware LATCH bits that correspond to the bits in bitmap are set. In particular, this provides access to BANK, allowing ZMDB to dump banked RAM.

3.15 Reset LATCH bits: 13 bitmap

The Z80 hardware LATCH bits that correspond to the bits in bitmap are reset.

3.16 Input LATCH bits: 14 <bitmap>

The value in the Z80 hardware

latch is sent back to the 68000.

3.17 Read Sector ID: 15 unit track side <errorcode> <sectornum>

The number of the first sector passing the drive head on unit, track, side is returned, if errorcode equals zero. This is useful for determining optimum sector skewing when used immediately after a disk operation. If the head was stepped to reach track, sectornum will be the sector that the seek verify is normally done on. In this case, the NEXT sector (physically, not numerically - do another read ID) will be the first one avail-

able for transfer during a normal I/O operation.

3.18 Set drive characteristics: 16 unit stepr 2step bitmap sizecode secpertrak tracks sides cachlev cngmethod <errorcode or 0>

Sets the drive characteristics. This also instructs the controller to sync the cache for 'unit', and act as if the disk has physically changed. (See "Drive characteristics" above). Values are:

stepr - step rate. 0 = 2ms, 1 = 3ms, 2 = 6ms, 3 = 12ms. Step rate can also be set with "set step rate" command.

2step - do we double step between tracks? 0 = No, 1 = Yes

bitmap - latch bitmap that selects this drive.

sizecode - sector size code. 0 = 128b, 1 = 256b, 2 = 512b, 3 = 1024b

secpertrak - number of sectors per track.

tracks - number of tracks on the drive.

sides - number of sides. 1 = Single Sided, 2 = Double sided.

cachlev - caching level. 0 = None, 1 = Read Cache, 2 Read + Readahead, 3 = Write Cache, 4 = Write + Rea dahead.

cngmet - disk change detection method. 0 = None, 1 = special

Interpr 3-
ocessor 1
Comm 8
unicati
on

HOLD RDY, 2 = by DISKCNG signal, 3 = by DISKCNG signal conneced to RDY.

A non-zero error code is returned if something really nasty happened; mainly the drive parameters being invalid for some reason. (Eg: specifying a drive to have zero sides).

3.19 Show drive characteristics: 17 unit <errorcode or 0> <stepr> <2step> <bitmap> <size code> <secpertrak> <tracks> <sides> <cachlev> <cngmethod>

3-19 Interprocessor Communication

Reads back the drive characteristics, if the unit number was valid. If not, errorcode is non-zero and no other data is returned. Bytes returned are the same ones that are passed to "set drive characteristics". If you want to change only one characteristic, read them all into a buffer, change the one you want, and write them back out again with "Set Drive Characteristics".

3.20 Fastcopy: 18 srcunit destunit <errcode>

Makes a mirror image copy of the disk in srcunit, onto the disk in destunit. You MUST have two drives for this. To copy an 800k disk with one drive would

require over 30 disk swaps, even if the controller had the full 64k of RAM for buffering. This command allows the fastest possible mirror imaging of a disk. A 'sync' is done at the beginning, to ensure that the data on the disks is up to date, and to provide enough RAM space to buffer the track.

The unit characteristics of 'sizecode', 'secpertrak', 'tracks' and 'sides' must be the same on srcunit and destunit. Provided these parameters are set to match the diskette in the drive, the command should copy any double-density disk, in any format - it needn't be a 1616OS disk.

Before the copy, track 0 of both the source and destination diskettes are scanned for Data Address Marks

(DAM's) and a sector skew table (like the one passed to "format") is built internally for each drive. Disk sectors are then read and written from each track according to the skewtables to ensure that reading the track takes only one revolution no matter what skew. One side at a time is buffered.

Destination sectors are written with the same DAM type (DAM vs Deleted DAM) that the source sectors had. This may some day be important to someone somewhere - Deleted DAMs are very rarely used these days.

3.21 Drive Revolution Timing: 19 unit <errocode> <

Interpr 3-
ocessor 2
Comm 2
unicati
on

timehigh> <timelow>

Activates the selected unit and, if no errors occur, does a drive revolution timing. The counter value that was reached is returned as a 16bit value in timehigh and timelow. This will be inversely proportional to the speed of the drive selected. The command does not wait for a RDY signal, or wait for the drive to reach speed; the first few values will indicate how long your drives take to spin up. It takes two full revolutions between returning timing values.

3.22 Sync: (Flush Cache) 1A

unit <errorcode>

All entries in the cache associated with the specified unit are flushed to disk. If unit = FF, all cache entries are flushed. This means that both the Read and Write cache are empty. This is important to ensure that the disks are updated before being removed, and to tell the system that a disk is ABOUT to change, if the drive can't do disk change detection. PLEASE, do the sync BEFORE removing the disk!. (It's SYNC, not LOG!).

If a background format is proceeding, that drive cannot be sync'd.

Interprocessor Communication 3-2 3-4

3.23 New Disk: 40 unit

This is intended to indicate to the SSDCC that the disk has changed; but since the SSDCC can detect disk change better than SSOS can, it's ignored.

Level 0 Reset: 41

Tells the SSDCC that a level 0 reset has occurred. Runs a SYNC auto matically, then rein-itialises everything. All drivparm settings are lost. SSOS sends one of these; you don't have to.

3.24 Level 1 Reset: 42

The SSDCC attempts to run a SYNC, then carries on as before.

3.25 Level 2 Reset: 43

The SSDCC attempts to run a SYNC, then carries on as before.

3.26 Hardware Mods:

I tried to keep required hardware mods to a minimum. Here are the ones you have to do:

Increase RAM size: It's all pretty pointless having disk caching but no RAM. Still, it DOES work with only 8k. It usen't to.

Enable Drive Select Outputs with Motor Off: This is essential for Disk Change detection to work. Disconnect the tracks at IC 19 pins 2 and 4. Connect pin 2 to pin 1 and pin 4 to pin 5. Note that the nor-

mal SSDCC ROM doesn't deselect the last drive once this mod is done; but that isn't a problem - it just looks odd because it leaves the drive's LED on.

Increase Drive Selection ability: If you want more than 2 drives, I suggest you rewire the EJECT and INUSE signals as drive selects DS2 and DS3. Remember to cut DS3s connection to DS1 (why did they do that?) Note that the trace you have to cut is under the 34pin connector on the component side of the board (convenient!). Drives 2 and 3 can be accessed with the appropriate drive select bitmaps.

3- Interp 2 rocesso 7 r Com munica tion

4 drives .doc - Info on how to strap popul ar dri ves.

The use of Greyham's SSDCC code allows special features of your drives to be used. These are often enabled by strapping plugs that may need to be moved from the default values. These changes generally involve disk-change detection which is done by various methods. The strapping info listed below should allow full write caching without the need to run "sync" whenever a disk is changed.

One thing to note: 5 1/4" drives usually (always?) have a termination resistor pack which should be installed only in the drive that is furthest from the SSDCC on the daisy-chain cable. The termination resistor packs should be removed from the other drive(s). (They are ALWAYS in a socket; you should not have to attack the drive with a soldering iron!).

If the drives you have aren't listed below, it merely means that I haven't been able to try one out. If you get it working, please contact me and tell me so I can include it in this list.

4.1 Common Drive

4-2 drives.doc - Info on how to strap popular drives.

Strap Names:

There's a small amount of standardisation among drive strapping names; and if the drives you have aren't explicitly mentioned below, you might find be able to match similar names and come up with something that works. Not all drives will have all jumpers, and the names may vary.

DS0, DS1, DS2, DS3 or DX0, DX1, DX2, DX3 These are the drive select jumpers. Only ONE of the four should be installed at a time, and selects that drive according to the drive select bitmap; usually as unit 0, 1, 2, 3.

MX This jumper sets the drive to ALWAYS be selected;

which rules out the use of the daisy-chain selection mechanism. This should NEVER be installed; even if you only have one drive.

HS, HM, HC, HL These are Head-Load jumpers; they won't be present if your drives head is always loaded against the disk, as is the case with most (all?) 3 1/2" drives. Their meanings are: HS - Load head on DRIVE-SELECT signal. HM - Load head from MOTOR-ON. HC - Always head-load. HL - Head-load from IN-USE. Only one of the four should be installed at any time. Head-load should not be done via DRIVE-SELECT, since the SSDCC polls the drives continually, and you will wear out your head-

4-4 .doc - Info on how to strap popular drives.

load mechanism very fast if HS is installed.
The IN-USE signal is not generally supported, so don't use HL either.
You have a choice of installing either HM or HC. HM will cause the drive head to load every time the motors go on; which inevitably wears the drive out a little faster (but then, they ARE built for this sort of thing). HC leaves the head in contact with the disk all the time, which will wear the disk and head out faster (but then, this is the case with most (all?) 3 1/2" drives anyhow).
So, depending on your personal preference, install EITHER HM or HC.

MS, MM
These are Motor-On jumpers; they control under what conditions the motor is

drives. 4-doc - 5 Info on how to strap p opular drives.

turned on. The idea is to leave the motor OFF unless any drive is being accessed, to reduce wear on the drive. MS and MM work as follows: MS only- Motor On by DRIVE-SELECT signal. MM only- Motor On by MOTOR-ON signal. MM & MS- Motor On by either MOTOR-ON or DRIVE-SELECT.

As is the case with drive-select, we don't want the motors turning on every time the drive is polled for disk-change while it is idle. So, Motor On should be controlled ONLY by the MOTOR-ON signal. Install jumper MM ONLY.

DC, 2S Don't ask me where they get the names; I would have thought 2S would have been

4-6 drives.doc - Info on how to strap popular drives.

something to do with double-sided or something, but nooooo. I've only seen these on Mitsubishi drives. They alter the meaning of the RDY signal, allowing it to do disk-change detection (which isn't common on 5 1/4" drives apparently). Jumper DC should never be installed, and jumper 2S then selects: Installed - HOLD-RDY mode. Not-Installed - Standard RDY mode.

Hold-RDY mode is a special mode whereby the RDY signal goes active when the drive is ready, and remains active even when the motors turn off; until the disk is removed from the drives.

IU Controls the selection of the front-panel LED. When installed, the

drives. 4-doc - 7 Info on how to strap popular drives.

IS-USE signal will illuminate the LED. This is not supported by the SSDCC; so don't install the jumper.

Mitsubishi M4853-1 5 1/4"

The manual I have for these drives is wrong in that it reverses the treatment of the MM and MS jumpers. The drives act as described above.

Install jumpers:HM, 2S, MM, IU, DSn
Remove Jumpers:HH, HL, HC, HS, DC, MS, MX

Set disk-change method in drivparm to 2 (Hold-Ready).

NEC FD-1036A 3 1/2"

As far as I can tell, the disk-change signal from these drives is broken; even though it's supposed to work. Luckily it

can be jumpered to the RDY signal, where it DOES work.

The jumper labelled "DCG 2" near the edge connector should be installed at the end with the "2". A drive select jumper should also be installed.

Set disk-change method in drivparm to 4 (Diskchange on RDY signal).

Mitsubishi MF351 3 1/2" Single-Sided

This drive can't do Disk-Change, but can do Hold-RDY.

Install jumpers:2S, MM, DSn Remove Jumpers:DC, MS, MX

Set disk-change method in drivparm to 2 (Hold-Ready).

Pertec FD-200 5 1/4" 40 Track Single-Sided

drives. 4-doc - 9 Info on how to strap popular drives.

Hardly a popular drive, it can’t cope even with 12ms step rate, which is the slowest the SSDCC can produce. Double-density is probably pushing it for these drives anyway.

5 files.lst - List of files on this disk

Device: /F1
Volume name: /SSDCC
Directory: /F1

AUTOEXEC0.SHELL- run by boot sector at level 0 reset.
AUTOEXEC1.SHELL - run by boot sector at level 1 reset.
AUTOEXEC2.SHELL- run by boot sector at level 2 reset.
FILES.LST- this list of file descriptions. LOADDC.SHELL- auto-loads the software version (the .cmd file)
READ.ME- brief intro to .doc files.

Device: /F1
Volume name: /SSDCC
Directory: /F1/BIN

DOSDIR.EXEC- list directory of an MSDOS disk. DOSGET.EXEC - get file from an MSDOS disk. DOSINIT.XREL- initialise an MSDOS diskette; used after format and before doswrite/dosput. Needs the file bootsector.fmt in the current directory. DOSPUT.EXEC- put a file onto an MSDOS disk. DOSREAD.EXEC - read a file from an MSDOS disk onto stdout. DOSSTAT.EXEC- display statistics from an MSDOS disk. DOSWRITE.EXEC- write a file onto an MSDOS disk from stdin. DRIVETIME.XREL- primitive drive speed indicator. DRIVPARM.XREL- set drive and diskette parameters. FASTCOPY.XREL- fast disk copier. FORMAT.XREL- intelligent

files.lst - List of files on this disk

disk formatter. GOZ80.XREL- send SSDCC an "execute Z-80 code" command. GREYBOOTV3.EXEC- My boot program that runs AUTOEXEC[0|1|2].SHELL HDCONFIG.XREL- new version of HDCONFIG that works with the new EPROM. ILATCH.XREL- Read input from Z-80 Latch. MAKECMD.XREL- Build a TRSDOS .CMD file from Z-80 memory address space. READVER.XREL- Display SSDCC version number. RECVCMD.XREL- Load a TRSDOS .CMD file into Z-80 memory address space. RLATCH.XREL- Reset bits in Z-80 output Latch. SETSTEP.XREL- Set step rate for a given drive. SLATCH.XREL- Set bits in Z-80 output Latch. SSDCCERR.XREL- Interro-

files.lst - List of files on this disk 5-3

gate SSDCC errors. SYNC.XREL- Indicate that disk is being changed; flush buffers. Necessary every disk change if you use caching and your drives can't support disk-change signals or you don't do the SSDCC card mod to output drive select signals. ZMAC.XREL- Z80 macro cross-assembler. "yacc" source isn't included. ZMDB.EXEC- MDB for Z-80 Address space. The source to this got deleted. ZMFB.XREL- MFB for Z-80 Address space. ZMWB.XREL - MWB for Z-80 Address space.

Device: /F1
Volume name: /SSDCC
Directory: /F1/CMD

DC___.CMD- Downloadable version of new disk controller. Doesn't

5-4 files.lst - List of files on this disk

support fast-copy, and SSO/S won't allow use of /H0 and /H1 (unless you have a SCSI ROM). DC LIM.CMD- Very limited version of the code for bootstrappin g it in.

Device: /F1 Volume name: /SSDCC Directory: /F1/DOC

CHANGES. DOC- Bor-ing file list-ing historical changes to the code. C OPYRIGHT .DOC- Even more boring file included for legal reasons. DC.DOC- All you ever wanted to know about the drive controller code. DRIV ES.DOC- Info on how to strap spe-cific drives for disk-change use. UTILI TYS.DOC- Description of some of the utilitys. ZMAC.DO C- Fairly pathetic info about the Z80 assembler.

files.ls t - List of files on this disk

Device: /F1
Volume name: /SSDCC
Directory: /F1/DOSREAD

DOSREAD.C- Source to the 'C' portion of DOSREAD, DOSWRITE, DOSDIR, DOSGET, DOSPUT and DOSSTAT. MAKE.SHELL- Shell file to recompile DOSxxxx from scratch. RDWR512.AS- 68k assembler routines to link to DOSREAD.C SSDD.HITECH- Hitech assembler format include file equivalent to SSDD.H

Device: /F1
Volume name: /SSDCC
Directory: /F1/FMT

BOOTSECTOR.FMT- Prototype MSDOS boot sector used by DOSINIT.

Device: /F1
Volume name: /SSDCC
Directory: /F1/HARDDISK

5-6 files.lst - List of files on this disk

ADAPTEC.C- Mark Harvey's init program for adaptec users. HARDDISK.DOC- Info explaining the story on the hard disk utilities. HDCONFIG.C- Mark Harvey's hard disk configuration program. MAKEADAPTEC.SHELL - Shell file to remake ADAPTEC.XREL MAKEHDCONFIG.SHELL - Shell file to remake HDCONFIG.XREL

Device: /F1
Volume name: /SSDCC
Directory: /F1/HD_LIB

Sources to the Hard-Disk library, written mainly by Mark Harvey.

MAKE.SHELL- Shell file to remake any library module. MAKELIB.SHELL - Shell file to rebuild the library. HDISK.LIB- The Hard-Disk library.

files.lst - List of files on this disk

Device: /F1
Volume name: /SSDCC
Directory: /F1/HEX

DC___.HEX- EPROM binary image. (Intel HEX format) HD___.HEX- EPROM binary image with SCSI support. (Intel HEX format)

Device: /F1
Volume name: /SSDCC
Directory: /F1/INC

CHECKVER.INC- Routine to check SSDCC version number. LWRXRDY.INC- Routine to wait a long time for stuff from Z-80. READ1024.INC- Routine to read a 1024byte block from the Z-80. SENDHL.INC- Routine to send a 16bit value to the Z80. SETCHAR.INC- Routine to set drive parameters. SHOCHAR.INC- Routine to read back drive param

files.lst - List of files on this disk

eters. SSDD.H- SSDCC header file with extra floppy stuff. SSDD.INC- Common include file for SSDCC programs. S SDCCERR. INC- Rou- tine to inter- rogate error codes. SYS CALLS.MA C- Syscalls interface from O/S3. WRITE102 4.INC- Rou- tine to write a 1024byte block to the Z-80. WRI TE512.INC- Routine to write a 512byte block to the Z-80. WRX RDY.INC- Routine to wait for stuff fromt the Z-80. WTX RDY.INC- Routine to wait for the Z-80 to accept stuff.

Device: /F1 Volume name: /SSDCC Directory: /F1/MRD

FDDVR.C- Source to extra floppy devices MRD. FDD VR.MRD- Extra floppy driver MRD; MAXUNIT = 2. FDMR DRIVERS- MRDRIVE RS file with

files.ls t - List of files on this disk

the extra floppy drives MRD. HDDVR.C- Source to hard disk driver MRD. HDDVR.MRD- Hard disk driver MRD. Auto-configuring; MAXUNIT = 2. HDMRDRIVERS- MRD RIVERS file with the hard disk MRD. MAKEFD.SHELL- Shell file to make FDDVR.MRD MAKEHD.SHELL- Shell file to make DDVR.MRD

Device: /F1 Volume name: /SSDCC Directory: /F1/SRC

DOSINIT.S- SSASM source to DOSINIT.XREL DRIVETIME.S- SSASM source to DRIVETIME.XREL DRIVPARM.S- SSASM source to DRIVPARM.XREL FASTCOPY.S- SSASM source to FASTCOPY.XREL FORMAT.S- SSASM source to FORMAT.XREL

5-10 files.lst - List of files on this disk

GOZ80.S-SSASM source to GOZ80.XREL GREYBOOTV3.S-SSASM source to GREYBOOT.EXEC ILATCH.S-SSASM source to ILATCH.XREL MAKE.SHELL- Shell script to remake .XREL stuff from .S files. MAKECMD.S-SSASM source to MAKECMD.XREL READVER.S-SSASM source to READVER.XREL RECVCMD.S-SSASM source to RECVCMD.XREL RLATCH.S-SSASM source to RLATCH.XREL SETSTEP.S-SSASM source to SETSTEP.XREL SLATCH.S-SSASM source to SLATCH.XREL SSDCCERR.S-SSASM source to SSDCCERR.XREL SYNC.S-SSASM source to SYNC.XREL ZMFB.S-SSASM source to Z

files.lst - List of files on this disk 5-11

MFB.XREL
ZMWB.S - SSASM source to ZMWB.XREL

5-1 files.lst
2 - List of files on this disk

6 harddisk.doc - Info on the hard disk initialisation programs.

These programs are exact copies of those written by Mark Harvey, and are included here for completeness. The disk library they link with IS different, however; since SCSI blocks are now read/written with a different message number to the floppy drives.

Hence when using the new Floppy/SCSI EPROM, the hdconfig.xrel and adaptec.xrel programs in the /BIN directory on

THIS disk MUST be used; NOT the originals. The ones on this disk are NOT compatible with the original SCSI EPROMs either.

6-harddisk.doc - Info on the hard disk initialisation programs.

7 utilitys.doc - description of 1616 utilitys by Greyham. 13-1-89

The software described below is COPYRIGHT; but with a licence permitting copies to be made, if done so for *FREE*. See copyright.doc.

<*>

This is a description of the utilities provided with my version of the SSDCC software. You should read 'dc.doc' before reading this, or it

won’t make a whole lost of sense.

Most of the utilities were needed to develop and debug the SSDCC software program. They are divided into two sections; firstly those that work with either the original SSDCC firmware, or my SSDCC software - and secondly those that work only with my SSDCC software. All utilities produce a usage message if bad parameters are offered. Try using a ’?’. Eg: recvcmd ?

7.1 RECVCMD [+|-][filename.cmd] -

Allows the downloading of TRSDOS format .CMD files into the

Z80's RAM. TRSDOS format .CMD files contain loader control information to specify where in the RAM space to load the information, and RECVCMD converts this into "Write Z-80 RAM" commands. A .CMD file must be input re-directed into the command. This allows files to be downloaded directly to the Z80 from, say, a serial port. RECVCMD displays information about where the input program loads, and stops reading when it detects the end of the .CMD file from the control information in the file.

If filename.cmd is specified, a copy of the .CMD file is written to filename.cmd. If '-' is

specified, the code is actually downloaded into the Z80's RAM. If '+' is specified, the code is downloaded into the Z80, and the Z80 is instructed to branch to the entry address specified in the .CMD file once the program is loaded into memory.

Although this command CAN be run from a physical disk drive, it should normally be run only from the RAM disk to minimise the corrupting effects of the SSDCC software.

Examples:

RD>recvcmd - <file.cmd

Load file.cmd into the Z80's RAM space.

RD>recvcmd -outfile.cmd <infile.cmd

Load infile.cmd into the Z80's RAM space and make a

copy of infile in outfile.cmd

RD>recvcmd + < infile.cmd

Load infile into the Z80's RAM space, and execute it.

7.2 GO Z80 addrs -

The Z80 is instructed (via the "Call Z80 program") to call the program at addrs in its address space.

Examples:

goz80 6000

Call the first location in the Z80's Common Bank address space.

7.3 SSDCCERR errorno

Displays the SSDCC error associated with errorno.
errorno

should never really be displayed to the user, as the "error message" command should be used to get an ASCII re presentation of the error message - that is what this com- mand does. The error message will usually con- tain numbers (Eg: track, sector, side etc) which will often be wrong, since the message is only valid immediately after the error occurred.

The error messages returned by my SSDCC software will be dif- ferent to those returned by the original. Many will have differ- ent numbers.

7.4 SE TSTEP unit ste pcode -

Sets the step rate for 'unit' according to

’stepcode’. This uses the "set step rate" command, with stepcode being the "rate" parameter. Thus, values for "stepcode" are: 0 = 2ms, 1 = 3ms, 2 = 6ms, 3 = 12ms.

7.5 MAKECMD cmdfile.cmd start1 end1 [...startn endn] entry

Creates a TRSDOS format .CMD file from the information in the Z80’s address space. Any number of data chunks can be contained in the file, so you may specify any number of start/end pairs. cmdfile.cmd is the name of the .cmd file to which the information is written, and entry is

the address to which transfer is controlled when the program starts. The .cmd file can be reloaded with RECV CMD at a later date.

Examples:

makecmd ssdccrom.cmd 0000 5fff 0000

Do a dump of the SSDCC rom, giving its entry address at 0000.

makecmd file.cmd 6000 6100 7000 7100 7500 7530 6000

Create file.cmd with the information from 6000 to 6100, 7000 to 7100 and 7500 to 7530, with entry address 6000.

7.6 ZM DB a1 [a2] ZMFB a1 a2 n1 ZM

WB a1 n1 [n2] [n3]....

These commands are very similar to the 1616OS commands of the same name (less the leading 'Z'), except that they operate on the Z80's address space. Also, zmwb doesn't have mwb's "interactive" mode. See the 1616 manual for details.

7.7 READVER -

Displays the SSDCC version number.

<*>

The utilities from here on can be run only with my SSDCC software, either because the things they aim to do don't make sense in the context of

the original firmware, or it just can't handle them.

7.8 DRIVPARM [unit [stepr 2step bitmap sizecode secpertrak tracks sides cachlev cngmet]] -

Drive parameter setting. With no parameters, it lists all drive parameters for all units. If 'unit' is specified, it lists parameters for that unit. If drive characteristics are specified, these become the drives new parameters. Characteristics must be in the correct order.

stepr - step rate. 0 = 2ms, 1 = 3ms, 2 = 6ms, 3 = 12ms. 2step - do we double step between tracks? 0 = No, 1 = Yes bitmap - latch bitmap that selects this drive. sizecode - sector size code. 0 = 128b, 1 = 256b, 2 = 512b, 3 = 1024b sec-pertrak - number of sectors per track. tracks - number of tracks. sides - number of sides. 1 = Single Sided, 2 = Double sided. cach-lev - caching level. 0 = None, 1 = Read Cache, 2 = Write Cache. cngmet - disk change. 0 = None; RDY not valid, 1 = None; RDY is valid, 2 = HOLD RDY, 3 = DISKCNG, 4 = DISKCNG on RDY

7.9 SYNC [unit] -

Flushes both read and write caches for unit, if specified; otherwise, flushes both caches for all units. Caching need not be enabled, although you don't need 'sync' if caching is disabled. Note that a sync SHOULD be run immediately after disabling caching.

You should ALWAYS run 'sync' BEFORE removing a disk from the drive if you have some form of caching enabled, and your drive cannot detect disk changes. The SSDCC will automatically flush the write cache just before turning it's drive motors off; however, this may be misleading because the motors may not have

been on, if the last access was only to the cache. sync will guarantee to flush the buffers immediately. However, do not run sync while logged onto the diskette you are about to remove!!! 1616 OS will proceed to read the root block when it goes back to the prompt, and that will stay in the cache, possibly corrupting the next disk!!!

7.10 4 DRIVETIME unit

Does continuous rotation timing tests on 'unit'. Tests continue until Alt-C (abort) is hit. A value of about 27775 is around 300RPM. If you happen to have a frequency counter and can set your drive to exactly 300RPM,

please tell me what value you get back. Precise drive speed could be found from this value, if you know the correlation.

When doing this drive rotation test, the SSDCC doesn't wait for a RDY signal from the drive; you will often get a "Rotational Failure" error at the start, either because 1) The drive doesn't start to output the INDEX signal until it's up to speed, or 2) During the spinup, the first revolution took longer than the SSDCCs maximum timing count. This is quite OK and normal.

7.11 FASTCOPY [-r] src unit dest unit

Makes a mirror

image copy of the disk in unit 'srcunit', onto the disk in 'destunit'. Source and Destination units must have identical diskette characteristics (sides, tracks and sectors per track), and these characteristics MUST match the diskette being copied!. The destination diskette must already be formatted, but needn't have any file system. Source and Destination needn't have the same skew factor, but otherwise must be identically formatted. The copy is buffered one diskette side at a time in the Z80's RAM - you can't do fastcopies with only one drive. (You'd have to swap disks 320 times for 800k!).

If the disk is found to be an SSO/S disk, the root block is scrambled in a similar manner to the SSO/S diskcopy utility. The -r option stops this from being done. The disks will then look identical to SSO/S, which will get VERY confused. DON'T use -r unless you have a REALLY good reason. Mirror-image backups is NOT a good reason.

You can quite happily make fastcopys of MSDOS disks by setting the relevant disk characteristics (40 tracks, 9 sectors per track, 2 sides), then you can use "format" to format the destination disk, and "fastcopy" to do the copy.

7.12 FORMAT unit -

Universal formatting program. This will format any disk according to the drive characteristics. As the SSDCC knows nothing about the disk file system, the diskette is not immediately usable; some sort of file system has to be placed on the disk. Eg: blockdev for 1616 OS, mkfs for Minix, dosinit for MSDOS etc.

format is capable of incredible skew variety - see the source format.s.

Note that blockdev is fully capable of formatting a disk under my version, so long as the drive characteristics have been set for a 1616OS disk.

7.13 DOSINIT unit

Initialises an empty MSDOS file system on unit. The disk in 'unit' must already have been formatted with 512 byte sectors. This will work with virtually ANY drive characteristics (so long as the sector size is 512 bytes), although I can't guarantee that MSDOS will be able to work with the end result. Give it a try. This is provided mainly to avoid have to actually USE MessDos, if at all possible.

The program requires a "Prototype Boot Sector" in the file bootsector.fmt. The MSDOS boot sector contains both diskette information and the DOS bootstrap loader.
DOSINIT

uses the prototype to get the boot-strap loader, and modifies its diskette information according to the drive's characteristics (as per drivparm). The boot sector from a disk formatted under the version of MessDos you plan to use the disk with is ideal; but virtually any one should do. One is already provided - it should work.

Needless to say, any files previously on the MessDos disk are WIPED!

7.14 SLATCH bitmap -

The "bitmap" is ORed onto the Z80's latch. This sets all the bits in the latch that are set in bitmap.

7.15 RLATCH bitmap -

The "bitmap" is NEGated and ANDed with the Z80's latch. This resets all the bits in the latch that are set in bitmap.

7.16 ILATCH

Reads input from the latch, displaying the data present there.

7.17 GREYSBOOT3

Applix's boot block program, modified so that it sets step rate to 3ms and reads 'autoexecn', where n is the boot level. autoexec0 should contain code to set your drive parameters via drivparm.

7.18 DOSREAD [-an] drive file -

DOSREAD reads a file from the MSDOS file system in 'drive', and sends it to the standard output. This may be redirected into a 1616OS file. The '-a' option does ASCII conversion of CR/LF combinations. The disk is checked for being in a sensible MSDOS format. The '-n' option relaxes this check.

DOSREAD, DOSWRITE, DOSDIR, DOSGET, DOSPUT and DOSSTAT are all identical copies of the same .exec program, derived from DOSREAD.C. The programs manipulate MSDOS (rather than 1616OS) file

systems, and were written in 'C' by Michiel Huisjes, originally for use under Minix.

The programs have been modified pretty heavily to allow them to read virtually any MSDOS file system; 3 1/2" disks in particular can now be read. There is no need to set the drive characteristics for the drive to the MSDOS parameters; the programs will set the sector size code for 512 byte sectors, read the boot block, and set the other drive characteristics from this. The original characteristics are restored when the programs exit to 1616OS. If recompiled, the programs must be linked to "rdwr512.as". They MUST be compiled to

an EXEC, rather than an XREL file. There is a make.shell file is the dosread directory that compiles the programs.

7.19 DOSWRITE [-an] drive file

DOSWRITE writes a file on the MSDOS file system in 'drive' from the standard input. The '-a' option does ASCII conversion of CR/LF combinations and the '-n' option relaxes the test that makes sure it actually IS an MSDOS disk.

7.20 DOSDIR [-nlr] drive [dir]

DOSDIR lists the

directory of the MSDOS disk in 'drive'. A directory path name may be specified to list the contents of sub directories - '/' rather than '\' should be used as the separator between the path components. '-l' gives a long listing, showing file attributes and modification date/time. '-n' relaxes the test that makes sure it actually IS an MSDOS disk.

7.21 DOSGET [-an] drive file1 [file2 [file3 ...]]] -

DOSGET reads files from the MSDOS disk in 'drive', and writes them on the current 1616 O/S device,

with the same file name. The file name(s) may contain a path component if the files are in subdirectories on the MSDOS disk, in which case the resulting 1616 O/S filenames will be the last component of the file name (ie: the path stripped off). This allows transfer of multiple files. The '-a' option does ASCII conversion of CR/LF combinations. The '-n' option relaxes the test that makes sure it actually IS an MSDOS disk.

Note that: RD>dosget 1 myfile Is equivalent to: RD>dosread 1 myfile > myfile

7.22 DOSPUT [-an] drive

file1 [file2 [file3 ...]]] -

DOSPUT writes files to the MSDOS disk in 'drive', with the same file name they had on the current O/S 1616 device. If the file name(s) contain a path component, the last component identifies the 1616 O/S file, and the files are written into a subdirectory on the MSDOS diskette. Naturally, wildcards may be used to write lots of files to the root directory of the MSDOS diskette. The '-a' option does ASCII conversion of CR/LF com binations. The '-n' option relaxes the test that makes sure it actually IS an MSDOS disk.

Note that: RD>dosput 1 myfile Is equivalent to: RD>dos write 1 myfile < myfile

7.23 DOSSTAT [-n] drive

DOSSTAT displays status information about the MSDOS diskette in 'drive'. This can be useful if the disks exact format is uncertain or strange things start happening.

Comments about assumptions in the output should be treated with caution as they indicate that boot sector data describing the disks logical dimensions was incomplete. The '-n' option relaxes the test for DOS-ness.

7.24 ZMAC(1) UNIX Programmer's Manual ZMAC(1)

NAME
zmac - macro cross-assembler for the Zilog Z80 microprocessor

SYNOPSIS
zmac [-bcdefgilLmnopst] infile

DESCRIPTION
The Zmac assembler is modelled after the Intel 8080 macro cross-assembler for the Intel 8080 by Ken Borgendale. The major features are: Full macro capabilities, Conditional assembly, A very flexible set of listing options and pseudo-ops, Symbol table output, Error report, Elimination of sequential searching, Commenting

of source, Facilities for system definition files.

Zmac assembles the specified input file (default extension .z) and produces a .hex output file. The options are:

b no binary

c produce a TRSDOS .CMD format file, instead of INTEL HEX.

d debug

e error list only

f print if skipped lines

g do not list extra code

i do not list include files

l no list

L force listing of everything

m print macro expansions

n put line numbers off

o list to standard output

p put out four \n's for eject

s don’t produce a symbol list

t don’t know what this option does

BUGS The man page is incomplete. If anyone discovers more information about using zmac, please consider helping to update the man page.

Table of Contents