1617: Disk Co-processor Card

Version 1.423
August, 1993

Applix 1616 microcomputer project
Applix pty ltd

1617 Disk Co-processor Card

Original version disk code by Andrew Morton.
Original version of this manual was written by Andrew Morton.
Additional introductory and tutorial material by Eric Lindsay, who edits and fancy prints the manuals.

Comments about this manual or the software it describes should be sent to:

Applix Pty Limited
Lot 1, Kent Street,
Yerrinbool, 2575
N.S.W. Australia
(48) 839 372

ISBN 0 947341 xx x

1
Introduction

The Applix 1616 Revision B motherboard contains a cassette port, but does not support any faster means of storing programs and data on magnetic media. Expansion of the system is by means of cards that plug into one of the four Applix expansion bus slots.

The principal function of the disk controller card (SSDCC) is to allow the use of floppy disk drives on your Applix 1616 (or similar!) computer. Your SSDCC controller comes complete with all components and programs required to control two floppy drives (drives are available separately from Applix and other suppliers).

Number of drives

The card supports one or two 3.5" or 5.25" floppy disk drives, giving you a formatted capacity of 800K each. We believe this should be sufficient for most Applix applications. Up to two drives are supported by the Version 1.4 EPROM in the Applix SSDCC.

With external modifications, and appropriate software programs (Greyham Stoney's disk cache, on Applix Shareware Disk # 7), the controller can support as many as 16 disk drives, but this is not a standard Applix configuration. An appendix to this manual, containing Greyham's extensive "read.me" notes, is available via the Users Group or Applix.

Summary of features

The SSDCC is much more than 'just a disk controller'. The card is a complete computer system, which contains its own 8MHz Z80 microprocessor, memory, an optional SCSI hard disk interface and an additional two serial ports. Its features include:

Onboard Z80H CPU running at 8MHz
8K to 32K of EPROM
8K to 64K of RAM
WD1772 floppy disk controller

and allows for...

SCSI hard disk interface
2 additional serial ports (under Z80 control) using Z8530 SCC

Optional kits

There are a variety of optional add-on components and software available for the controller card. Ask for full details on those described below.

- Hard disk kit includes an NCR5380 SCSI hard disk controller chip, and a revised EPROM (Version 2.n) containing Mark Harvey's SCSI drivers, plus various hard disk programs. An extensive appendix to this manual is supplied with the hard disk update kit. Appropriate cables and SCSI hard disks can be obtained from Applix and other suppliers. Approximate cost $99, cables $25, SCSI drives $600 and up, depending upon capacity. An additional power supply may be required.

- CP/M expansion includes a revised ZPAL chip, called CZPAL, 64k of static RAM (to replace the normal 8 k), and the ZRDOS and ZCPR3 software, as ported by Conal Walsh. This is an expanded and improved superset of CP/M 2.2. A number of utilities are provided with this conversion, and it includes the ability to read Microbee CP/M disks. Many traditional CP/M programs have been converted, including WordStar, and dBase II. Cost $150, includes 64k RAM, and all licensed software.
- Cache and special format software for floppy disks, by Greyham Stoney, is available as 'freeware'. This assists you read, write and format disks for any format that can be produced by the WD1772.

  A fast disk copy is included, together with special programs to read, write, convert and format MS-DOS disks. Many additional, detailed, error messages are provided. This frequently updated software, which is also available in EPROM, is indicated by Version A.4x etc. Extensive and very informative documentation is available on the disk. A printed version, similar to this manual, has been produced. Cost of shareware disk $5.

Manual contents

This manual is not required for normal operation of the Applix 1616, as high level disk commands are always available within the 1616/OS operating system. The manual is provided as background information for software experimenters wishing to explore the system at a reasonably fundamental level. It also includes complete construction and testing details, for those building the controller from a kit, or wishing to do repairs.

Sections 2 and 3 of this manual provide, as background reading, a detailed introduction to the low level concepts underlying disk controllers and drives. It also explains in some detail the electronic design of the SSDCC board.

Although we do not encourage you to write your own drivers, there is sufficient information for the experimenter to commence writing low level drivers (provided they know Z80 code, and have access to a Z80 assembler).

Those intending to experiment will also need to obtain copies of the manufacturer's notes on the WD1772, the disk drive, the NCR5380 SCSI controller, and a SCSI drive manual. These manuals can be made available (at cost) via the Applix Users group, but you should be warned that they are not light reading, and occupy about 500 pages!

Kit builders will find notes on construction techniques in Section 4, and a full parts list in Section 5. The actual construction and testing of the controller are covered step by step in Sections 6 and 7. The pin outs of all connectors are listed in Appendix A.

A number of block related commands for direct operation of the controller from the 1616 are described in Section 8, on SSDCC software. These commands allow most of the WD1772 and NCR5380 facilities to be used, without the need for writing low level code.

A brief description of the disk, block and directory structure is provided in Section 9, on disk organisation. This should provide sufficient information for those writing file recovery, file display, disk editor, and other related utilities.

Descriptions of the hard disk hardware and commands are provided with the hard disk upgrade kit. At present, there are no additional notes on the serial ports.

2
SSDCC Design Overview

The major sections of the SSDCC circuit are

- Z80 processor and address decode circuit
- the Z80's memory
- 1616 bus interface
- floppy disk interface
- SCSI hard disk interface
- dual serial I/O channels

Z80 introduction

The Z80 has separate memory and I/O address spaces. It also has a separate set of bus signals to control these separate memory and I/O spaces, unlike the 1616's MC68000 microprocessor which has a memory space only.

Obviously, the EPROM and RAM are mapped into the Z80's memory address space. As the Z80 has only 16 address lines, only 64k of memory can be addressed, so when two RAM chips (each of which could contain up to 32k) are used, bank selection is provided.

Access to the 1616, to the disk drive select lines, the floppy disk controller chip, SCSI controller, and the serial ports, are all by means of the Z80's I/O address space.

In normal operation with Applix software, the Z80 stack starts in memory at $7800, and builds down, while local variable use starts at $6000 and builds up. The $800 bytes from $7800 to $7fff are usually available for user programs.

If you intend to use the Z80 directly, rather than indirectly by means of the routines provided, you will need to know how to program a Z80 in assembler. Information on Z80 assembler is not provided in this (or any other) Applix manual, however many computer shops sell introductory manuals.

Note: Signals beginning with a '/' (eg /DTACK) are active low. A signal is referred to as being 'asserted' when it is in its active state. For an active low signal this is the low state. A signal in its inactive state is referred to as being 'negated'.

Note: The Z80 convention is to use the letter **H** after hexadecimal numbers (rather than using a $ sign before the number). Most Z80 hexadecimal numbers that would otherwise start with a letter have a 0 in front. I have tried to alter this manual to suit the Z80 conventions, when referring to Z80 memory and I/O addresses (don't blame me, it wasn't my idea, it is merely the convention in the Z80 world!)

Z80 address decode

The Z80 decode PAL (U8, ZPAL) and the 3 to 8 decoder (U13, 74LS138) together perform the memory and I/O enabling.

The signals 'MAP' and 'BANK', which go into the Z80 PAL, are provided for selecting a different memory map, and for overlaying memory banks. The Z80's memory map is as follows:

0000H-5FFFH	ROM (only 24k of the potential 32k available)
6000H-7FFFH	Common RAM0 bank
8000H-0FFFFH	Switching RAM bank

So that the Z80 may access all of the possible 64 kbytes of RAM whilst still reading from the ROM, the RAM is split into two 32k halves. Only one of these halves may appear in the top 32k of address space at a time.

When the 'BANK' signal is low the data in RAM0 (U1) is accessible in this address range; when 'BANK' is high the data in RAM1 (U2) is accessible. If 8 kbyte RAM chips are loaded in the board, then only the first 8 kbytes of this 32k address space are useful. 8 kbytes of RAM0 is always accessible in the common bank, and is repeated at 0E000H to 0FFFFH.

The 'MAP' signal is not used at present, in normal mode, but is used by the CP/M convention.

Z80's I/O address map

Address	Name	Function
00H	PORT	Read only input port.
08H	LATCH	Read/write disk select latch.
10H 10H	ZINTS ZCLRINT	Write: interrupt the 1616. Read: clear pending Z80 interrupt.
18H	SDATA	Read/write 1616 communications port.
20H	SCSIBASE	SCSI controller base address (20H-3FH).
40H	FDCBASE	Floppy disk controller base address (40H-5FH).
60H	SCCBASE	Serial communications controller base address (60H-7FH).

Input port

The input port (U22, 74LS244) enables a Z80 program to determine the level of the following signals:

Bit 0 The 'SCOMMAND' signal is set when the byte from the 1616 which is currently held in the receive latch is a command. This means that the MC68000 put the data there by writing to its 'SCOMMAND' output port.

Bit 1 The 'ZRXRDY' signal is high if there is a data or command byte from the MC68000 within the receive register.

Bit 2 The 'ZTXRDY' signal is high if the 1616 has read the previous byte out of the transmit register.

Bit 3 This signal determines whether the Z80 is to enter its normal operating mode or to execute its diagnostic test mode.

Drive select latch

Various disk functions are controlled by writing to U16, a 74LS273 latch, located at 08H in the Z80 I/O space. This latch controls which drive is selected, which side of the disk is selected, controls the /INUSE and /EJECT lines, the /MOTORON line (and also the LED on the card). It also produces the BANK and MAP signals, all of which are mapped as follows:

Bit 7	80H	MAP
Bit 6	40H	BANK
Bit 5	20H	Side select
Bit 4	10H	Motor on, and LED
Bit 3	8H	Drive select 1
Bit 2	4H	Drive select 0
Bit 1	2H	/EJECT line
Bit 0	1H	/INUSE line

If the Z80 reads from the same I/O location ($08, at U15, 74LS244), you can obtain the status of the disk drive /DISK CHANGE and /READY lines.

Bit 1	2H	/DISK CHANGE
Bit 0	1H	/READY

Interrupts

Although provision is made in the hardware (by means of a jumper block) for the use of the Z80 interrupts, they are not used by floppy disk (Version 1.4) software. There is capability for the Z80 and the 1616 to interrupt each other. Interrupts are not used by the software in the SSDCC at present. The interrupt mechanism is as follows:

Z80 interrupted by 1616

The 1616 writes a byte with a zero in its least significant bit (LSB) to the SINTZ port; this causes pin 9 of U14 (74LS74) to go low. This signal (available on pin 2 of the jumper block) should be connected to the Z80's /INT or /NMI signal on the interrupt strapping block. When the Z80 accepts the interrupt it should clear the interrupt signal by reading from its ZCLRINT address.

1616 interrupted by Z80

The Z80 writes a byte with zero in its LSB to the ZINTS port. This sends pin 5 of U14 (74LS74) low, holding the 1616 bus /EIRQ1 signal low. The /EIRQ1 signal must be connected to one of the 1616's interrupt pins on the 'INT LEVEL' strapping block on the 1616 main board. When the 1616 interrupt is taken, the 1616 must clear the interrupt signal from within the interrupt service routine by reading from the SCLRINT port.

SCSI interrupt

Interrupts from the SCSI to Z80 are used by some versions of the SCSI hard disk software. The non-maskable interrupt (NMI) is used, and links 3 (NMI) and 5 (SCSIRQ) must have a shorting block inserted for these version. The versions in question have an **odd** number. That is, version 2.1, version 2.3, etc. Version 2.0, 2.2 of the hard disk software does not require the NMI interrupt.

The Z80 is normally set to interrupt mode 1, and should be returned to this mode if your program changes it. The maskable interrupt has a jump vector set to $7800, and user written programs can commence here. At reset, address 7800H is set to a warm start.

Wait states

The Z80 may be forced to insert extra clock cycles into a memory or I/O transaction by forcing it into a 'wait state'. This is done when pin 13 of the ZPAL goes low; the arrangement of the

flip-flop U6 (74LS74) causes the Z80 to insert a single wait state into a memory or I/O access. Thus, the programming of the PAL determines which addresses receive wait states, and which do not.

At present all I/O transactions and ROM reads have a wait state. RAM reads and writes proceed at full speed.

About 16L8 PALs

A 16L8 PAL, used in the SSDCC, is an array of simple logic gates whose output is immediately available (or non-registered). The non-registered PAL (16L8) is used simply to detect certain combinations on its input pins, which makes it ideal for address decoding applications.

They are unlike the 16R8 PALs used on the 1616's main board. The 'R' stands for registered, which means that the 1616 PALs have latched outputs, so that changes in the registered PAL's output state can only occur on the rising edge of the PAL's clock pin. A registered PAL is used in a situation where its outputs have to be synchronised, or where memory about the PAL's previous states is needed.

Interface to 1616

The 1616 address decoding, /DTACK signal generation, status port multiplexing and hand-shaking control is done by the U24, (SPAL), the other 16L8 PAL on the SSDCC.

With this PAL we are using the PAL's ability to turn its outputs into a high-impedance (or floating, or tri-state) condition under certain input signal combinations. The /DTACK output pin is normally floating; it is actively driven only during MC68000 accesses. Similarly the D7 output pin directly drives the 1616 data bus and is floated until the MC68000 reads the SRXRDY, STXRDY or ZCOMMAND signals. When this happens the PAL routes the selected signal onto the D7 output and enables this pin to drive the bus.

Communication with the Z80

A mechanism is provided for communication between the Z80 and the 1616. This is done by directly accessing registers within the disk controller I/O space.

The 1616's MC68000 can read and write various registers within the controller at fixed addresses. You can use your monitor commands for experimenting, however the commands listed in Section 8, on software, should be used normally.

Address	R/W	Name	Function
$FFFFC1	R/W	ZDATA	Read: Data from Z80. Write: Data to Z80.
$FFFFC3	Read	SCLRINT	Clear 1616 interrupt.
$FFFFC3	Write	SINTZ	Interrupt Z80.
$FFFFC9	Read	SRXRDY	Bit 7 set if receive latch full.
$FFFFCB	Read	STXRDY	Bit 7 set if transmit latch empty.
$FFFFCD	Read	ZCOMMAND	Bit 7 set if the contents of the receive latch is a command.
$FFFFD1	Write	SCOMMAND	Write data to the data latch, set SCOMMAND bit.

1616 to SSDCC

The communication between the two processors is quite simple. If the 1616 wishes to transmit a byte to the Z80, it waits for STXRDY to go true and then writes the byte to the ZDATA port. The action of writing to ZDATA causes STXRDY to go false until the Z80 has read the new byte.

SSDCC to 1616

Similarly when the Z80 wishes to transmit to the 1616 it waits for ZTXRDY to go true, and then writes the data to the SDATA port. This causes the 1616's SRXRDY signal to go true, indicating that there is valid data in the receive port latch.

When the 1616 reads the data, SRXRDY goes false and the Z80's handshake signal ZTXRDY goes false. In fact ZTXRDY is the complement of SRXRDY and ZRXRDY is the complement of STXRDY.

Command flag

For software reasons it is desirable that the transmitting processor can add a flag to a transmitted byte, to indicate whether it is a data byte or a command byte; a command byte is one which initiates a whole transaction such as reading a disk sector. Being able to flag commands simplifies the problem of synchronising each processor's software.

The command bits are address triggered

When the 1616 writes to the data port latch, the MC68000 address line A4 is latched in U17 (74LS74) as the SCOMMAND signal.

When the Z80 sees that data is available (via ZRXRDY) it inspects the SCOMMAND signal. If this is high then the Z80 knows that the 1616's A4 signal was high when the data was written; the 1616's command port is at an address which has A4 high whereas its data port's address has A4 low, so the Z80 can differentiate between command bytes and data bytes.

A transaction between the Z80 and the 1616 involves the 1616 writing a command to the command port, and then transmitting and/or receiving data in accordance with that command. The commands that are implemented and expected are detailed in Section 8, on SSDCC software.

If writing your own commands, remember that you should never write data to the ZDATA or SCOMMAND ports when STXRDY is not asserted. Similarly, data read from the ZDATA port is meaningless if SRXRDY is not true.

3
The Floppy Disk Interface

A floppy disk system consists of:

- A disk drive or drives, with their associated built-in electronics. Applix recommend an 80 track, 3.5 inch, double sided drive, for compatibility with other users. You should contact Applix prior to buying drives, as some brands work better than others.
- A cable to the disk drive controller (a 34 way ribbon cable, with IDC connectors, available from Applix).
- A power supply to run the drives. In this case, the Applix power supply is used, with a 4 wire cable, available from Applix. The drives normally require +5 volts, and +12 volts. Note: As the cable for 3.5 inch drives can be accidently plugged in backwards, you should be careful when plugging in the power connector.
- A drive controller chip, and associated buffer and address select electronics. Applix use the Western Digital WD1772, on the disk controller card.
- A computer to control the drive controller chip. This is the Z80 on the disk controller card.
- Software to control the drive controller chip at a low level. This is contained in the Version 1.4 EPROM on the disk controller card.

In outline, the SSDCC works as below:

68000 in Applix 1616 issues high level disk commands such as *block read*, *block write*, *format*, etc. as a single command code byte sent to $FFFFD1. All other parameters (such as block number) are sent to $FFFFC1. Data is sent to and received from $FFFFC1.

Z80 in controller card receives command, parameters and data at I/O address $18. Software in EPROM alters block numbers to track and sector numbers, issues appropriate command to WD1772 disk controller chip, and drive select latch.

Drive select latch is connected to drive select lines, and side select. WD1772 produces pulses in step line to move from track to track. Tests whether it has the correct sector. Reads and writes data, passing it via the Z80 to SDATA port at I/O address $18.

Drive is connected via 34 way control and 4 way power cable to disk controller. Responds to pulses on control lines. Reads and writes sectors upon command.

We will describe the types of disk drives, and then outline how the disk controller chip works.

Types of disk drives

Floppy disk drives were devised in the mid 1970's by IBM, and used to load the initial program code for mainframes. These drives used 8 inch disks, and tended to be expensive. One manufacturer, Shugart, popularised them by making a cheaper version, using a 5.25 inch disk. Early versions of all models of disk drives tended to use only one side of the disk, however virtually all now use both sides. The early 5.25 inch drives were of lower capacity, having only 35 or 40 tracks, rather than the 77 or 80 tracks of the 8 inch drives. The 5.25 inch drives also rotated slower (300 rpm against 360 rpm), and had a data transfer rate of 125 kbits per second, only half that of the 8 inch drives. Early drives used single density (FM) data recording methods, where each data bit was interleaved between clock bits. Later double density (MFM) recording eliminated this, and the data transfer rate doubled, to 250 kbits per second.

Over the years, these restrictions were overcome, as in the IBM AT high density drive, with 80 tracks, 360 rpm rotation, double standard transfer rates, and 1.2 megabyte capacity. However the standard (IBM XT style) 5.25 inch drive still uses only 40 tracks, and 300 rpm rotation rate. Luckily, the connections to most disk drives are relatively standard. Within limits, any fairly standard (not high density) 5.25 inch disk drive can be made to work on the Applix controller.

A wide variety of 3.5 inch and similar sized disk drives were devised in the mid 1980s. Most rapidly disappeared, although there are still a few odd (and therefore incompatible) drives, such as that used by Amstrad word processors, and the Apple Macintosh. A number of earlier computers also used 40 track, or single sided, 3.5 inch drives. These drives are not supported on the Applix. Recently, IBM started using a high density drive, providing 1.44 megabyte. The Applix controller chip can not support this drive.

The Applix is designed to use an 80 track, double sided, 300 rpm, 3.5 inch drive. This provides 800 kilobytes of storage, and a data transfer rate of 250 kbits per second. These are the drives used on all recent Amiga, Atari ST, and PC clone portable computers. All the discussions to follow assume the above drive.

Chart of drive characteristics

Size inches	Tracks	TPI	Sides	Transfer Rate	Rotation Rate rpm	Sector Size	Record Method	Size in kbyte	Name
8	77	48	1	250	360	26x128	FM	250	SSSD
8	77	48	2	250	360	26x128	FM	500	DSSD
8	77	48	2	500	360	26x256	MFM	1000	DSDD
5.25	35	48	1	125	300	10x256	FM	87	TRS80
5.25	35	48	1	163	300	16x256	RLL	140	Apple
5.25	40	48	1	125	300	16x128	FM	71	CP/M SSSD
5.25 *	40	48	1	250	300	8x512	MFM	160	SSDD
5.25 *	40	48	2	250	300	9x512	MFM	360	DSDD
5.25 *	80	96	2	250	300	15x256	MFM	600	80T
5.25	80	96	2	500	360	15x512	MFM	1200	AT HD
3.25	80	135	1	Var	Var	Var	RLL	400	Mac
3.25	80	135	2	Var	Var	Var	RLL	800	Mac
3.25 *	80	135	2	250	300	9x512	MFM	720	IBM
3.25 *	80	135	2	250	300	5x1024	MFM	800	Applix
3.25	80	135	2	500	300	18x512	MFM	1440	IBM HD

Drive mechanics

The DC drive motor circuit includes an integral tachometer and servo speed control that rotates the disk at exactly 300 rpm, or 5 times a second. The read/write head is mounted on a carrier that moves back and forth over the disk. The stepper motor driving it can move it only a track at a time, to provide 135 tracks per inch. The total distance between track 0 and track 79 is only a little over a half inch, so each track occupies less than 7/1000th of an inch.

The movement from track to track **must** be identical for every drive, since they have to read disks made by other drives. This means that the disk must be seated correctly in the drive (the metal hubs of 3.5 inch disks help here). In short, disk drives are a piece of high precision equipment.

Drive electronics

The printed circuit board assembly in the drive contains surface mounted electronics to perform the following functions:

- Detect which of four possible drive numbers are selected.
- Start the drive motor rotating.
- Select the correct side of the disk to use.
- Position the read/write head to the correct track.
- Detect when the read/write head is over track 0.
- Detect when a disk is write protected.

- Detect the sector index hole in the disk.
- Generate signals in the read/write head to write data, or read magnetic flux changes from the disk.

Except for the last function, all the above involve uncomplicated circuits.

When you are to use a drive, it must receive signals to select whatever drive number it is set to, plus the motor must be turned on (and allowed a half second or so to reach correct speed). The side of the disk to use must be selected. The direction the head is to move must be selected, and then a correctly timed step pulse must be applied enough times to move the head to the required track.

If for some reason the drive controller does not know which track the head is currently positioned over, it must be moved to track 0. The step direction is set to step out, and 80 or more pulses are applied to the step pulse line. Then the controller checks for a track 0 indicator from the drive. If it doesn't get it, this sequence may be repeated, and/or an error message generated.

When the drive read/write head reaches the required track, the head is brought in contact with the disk. The head reads the disk, and the drive controller chip looks for a pattern in the format information, indicating it has the correct track number and sector. Other patterns in the format information allow the drive controller chip to start reading or writing at exactly the right point for the 1024 bytes it needs to use as data. The actual format information is not altered.

All this could be done by directly controlling the disk drive from the CPU. In the Apple, for example, the drive stepper motors and everything except the read and write data are directly operated via latched lines from the CPU. In Charles Moore's Forth engine, even the bit serial read and write data are directly generated by the CPU. However, this occupies a lot of CPU time, and the general practice is to allow a specialised disk controller chip to do most of the work.

The actual tracks are magnetically recorded on the disk when you format and initialise it. The disk controller chip takes care of keeping count of which track is being used. Tracks are also subdivided into sectors by the disk controller chip when you first format a disk. Each sector can contain 128 bytes, 256 bytes, 512 bytes, or 1024 bytes of data. The Applix uses 1024 byte sectors (IBM PCs use 512 byte sectors).

Disk speeds

The information in a sector is transferred at a rate fixed by the clock fed to the controller chip. The standard double density rate is 250,000 bits per second, or 31,250 bytes per second. The controller chip takes care of converting parallel bytes from the Z80 CPU into the serial read/write data. If you do some arithmetic using this bit rate, and the rotation rate of 5 times a second, you will see each track can contain 6250 bytes in total. However, the Applix stores only five 1024 byte sectors, or 5120 bytes per track (819,200 bytes per disk).

The missing bytes are essential for locating and identifying the material we store on disk, and are placed on the disk by the controller chip when we first format the disk, as discussed later.

It follows that, if we read all five sectors one after another, and could instantly move the read/write head to another track and start reading again, we would transfer information from the disk at 25,600 bytes per second. Several factors may constrain this.

- It takes about a half second for the disk drive motor to reach the correct speed, and we should not read data before this occurs.

- The sectors we wish to read may not follow each other. This depends both upon whether the operating system attempts to store them this way, whether we have altered their length along the way, and whether there was room to store them in a contiguous form. When files are scattered all over the disk, it is said to be 'fragmented'. There are various techniques of disk space allocation that can reduce (but usually not eliminate) fragmentation.
- We may not be able to start reading a following sector before it has moved past the read/write head. In this case, we have to wait one and one fifth revolutions (240 milliseconds) for it to come round to the read/write head before we can read the next sector.
- The next sector may be on another track. The head takes a certain time (between two and twelve milliseconds on our drives, up to 30 milliseconds on some older drives) to move from one track to the next. It also takes a certain length of time for the head to settle on the track again (some drives keep the head in contact with the track whenever they are seeking another track). If the required sector has already moved past the read/write head, we again have to wait up to 240 milliseconds. For this reason, sectors are sometimes skewed from one side of the disk to the next, or from one track to the next.

  Since we can read both sides of a disk, we can take this into account when formatting, as you will see in the command for formatting a disk. Side 0 sectors are, in order, 1,4,2,5,3, while side 1 sectors are 3,1,4,2,5. If we read alternate sides of the disk, we should be able to read ten sectors (10k) in two revolutions (400 milliseconds). You should note that the same skew is used for each track, so if we could move to the next track fast enough, we could start reading immediately. Of course, if the sector is not on the next track, the head movement time will be greater, and skewing may not help.
- Our Z80 CPU may not be able to keep up with the rate at which data is being delivered from the disk. The Z80 has about 32 µseconds to deal with each byte from the disk. If it is being interrupted to do something else (like output to the 1616), it may not be able to keep up.

  The same sort of transfer rate is required between the Z80 and the 68000 in the 1616, when we move disk bytes to their final destination.
- Finally, some disk files, such as `xrel` programs, require some preliminary processing by the 68000 before they can be moved to their correct location.

All of the above factors tend to make it difficult to achieve theoretical disk drive data transfer speeds. One encouraging sign is that most operating systems, whatever the computer, tend to approach theoretical maximum speeds more closely as they mature, and as their low level code is tuned for speed. Here are some typical times (in seconds) for file operations on a few different computers.

File size kilobyte	Applix	Amiga	Atari	Macintosh	MS-DOS
5		11			2.9
10		12			3.6
20		14.5			5.5
50		23			9.3
100		40			17.1

Disk controller chip

The disk controller uses Western Digital's WD1772 all digital floppy disk controller IC. A floppy disk controller handles the low-level control of the disk drive:

- Synchronisation of new data with that which is already on the disk
- Checking for errors in read data
- Searching for the correct disk sector
- Issuing stepping pulses to move the disk drive's read/write head to the correct track, etc.

The disk controller chip only knows about tracks, and sectors, and reading and writing them. It does not know about file names, nor where in memory files should go. This leaves the Z80 with the task of issuing commands to the controller, transferring data to or from it and handling errors which the controller detects.

We have seen many generations of floppy disk controllers over the years; it is only this latest generation which have avoided the need for setting up magic frequencies with variable capacitors, devising ingenious data separators which occasionally worked, etc. The all-digital design of the WD1772 eliminates these problems. The chip still has a few problems which can keep a programmer quiet for a few days, however ...

The WD1772 is a low cost version of the FD179x line. Although it is not able to use high density disk drives, it has the advantage of being relatively cheap, and includes a built-in data separator, plus write pre-compensation. The only additional components needed are buffers between it and the drive, and chip select circuit. It provides all the signals needed by the drive, and handles conversion of parallel data to the serial, clocked format required by the drive. The chip is designed to operate only one drive, which is why the Applix has a separate latch to control the drive select and side select lines. In the Applix, only MFM format data (double density) is used, although the chip itself can also handle the older single density FM format (the chip connection that controls this is not used).

The WD1772 appears to the Z80 as four I/O locations, starting at I/O address 40H.

- The WD1772 is operated by writing a control byte to $40, and its status determined by reading from 40H.
- Address 41H contains its track register, whose contents are automatically increased by one when the head steps in a track, and decreased by one when it steps out. During disk read, write and verify, the contents of this register are automatically compared with the format information on the track.
- The sector register is at 42H, and the contents are also automatically compared with the recorded sector information during disk reads and writes. Sectors count from 1 to 255, while tracks count from 0 to 255. This seems inconsistent and silly, but it is the way all the chips do it.
- Actual data is transferred to and from the controller chip at I/O location 43H.

Controller commands

The WD1772 accepts eleven command words in its command register. These are divided into four groups. Before using a command, you should check that the busy bit (0) of the status register ($40) is off. You would not normally use the 1616 at this level, unless you are writing your own low level disk routines. This information is included to give you a general idea of how the disk controller works. You will need a copy of the WD1772 manual if writing low level code.

Type 1 commands optionally verify the track. All Type 1 commands generate an interrupt when completed, however INTRQ, pin 28, the interrupt output, is not connected in the Applix (the Applix could use DRQ, pin 27, the data byte present line, to force an interrupt as each byte of data becomes available.) Obviously, if more than one drive is present, the track register must be updated prior to issuing these commands. This is a consequence of the drive controller chip being designed to operate one drive only (well, if you built controller chips for a living, would you make it easy to avoid buying extras?) The Type 1 commands are:

- Restore. Locate track 0, update track register to 0.
- Seek. Assumes track register is correct, and data register contains the desired track. Steps to the desired track.
- Step. Steps one track in whatever direction was last used.
- Step-in. One step in the direction of track 79.
- Step-out. One step in the direction of track 0.

Type II commands read and write sectors. Prior to issuing these commands, you must load the sector register with the number of the desired sector. If the correct track and sector is not found within four disk revolutions, the command produces an error (status bit 4 set). An interrupt is produced upon either completion or error. The command can be used to read and write multiple sectors, of up to a full track, by setting the **m** flag within the commands.

- Read sector. Checks for correct track and sector, then reads a sector (or multiple sectors). Generates a pulse on DRQ, pin 27, which is not used by the Applix. The Z80 must poll the DRQ bit in the status register before the next byte is available. The lost data status flag is set if a read is not done in time.
- Write sector. Checks for correct track and sector, reads formatting information to locate start of data field, and generates DRQ, pin 27. It then expects the first data byte to be loaded in the data register. The lost data status bit is set if each following byte of data is not supplied in time for the write.

Type III commands can be used for diagnostics and track formatting.

- Read address. This command reads the first ID field it finds from a formatted disk. You would need to position the head on the desired track before using it. It produces six bytes, which contain the actual track address, the side number, the sector address, the sector length, and two CRC bytes. The controller chip verifies the CRC values, and sets the CRC error status bit if an error is encountered.
- Read track. This powerful diagnostic command ignores all formatting, and all CRC errors. It reads **all** bytes between one index pulse and the next. This includes all gap, header and ID field bytes. Because synchronisation is not performed, the gap bytes may read incorrectly. Probably of use in determining details of ‘foreign’ formats, nature of disk errors, and the like.
- Write track, or formatting the disk. To use this, position the head over the desired track. The command writes from one index pulse to the next. It produces the correct header and syncronisation information, while the Z80 provides the gap and data bytes.

There is only one type IV command, and that is a forced interrupt, used to terminate a multiple sector read or write, or to force the status register into Type 1 mode. You must wait 16 µseconds after a forced interrupt before issuing another command.

The bit patterns for all commands are summarised below:

Type	Command	7	6	5	4	3	2	1	0
I	Restore	0	0	0	0	h	V	r_1	r_2
I	Seek	0	0	0	1	h	V	r_1	r_2
I	Step	0	0	1	u	h	V	r_1	r_2
I	Step-in	0	1	0	u	h	V	r_1	r_2
I	Step-out	0	1	1	u	h	V	r_1	r2
II	Read sector	1	0	0	m	h	E	0	0
II	Write sector	1	0	1	m	h	E	P	a_0
III	Read address	1	1	0	0	h	E	0	0
III	Read track	1	1	1	0	h	E	0	0
III	Write track	1	1	1	1	h	E	P	0
IV	Interrupt	1	1	0	1	I_3	I_2	I_1	I_0

The various flags in the commands have means as follows:

h Motor on flag, bit 3. 0 = enable spin up sequence.
V Verify flag, bit 2. 1 = verify destination track.
u Update flag, bit 4. 1 = update track register automatically.
r_1, r_2 Stepping rate, bits 1, 0. 6, 12, 2 and 3 milliseconds.
m Multiple sector flag, bit 4. 1 = multiple sectors.
a_0 Data address mark, bit 0. 0 = write normal data mark.
E 30ms head settling delay, bit 2. 0 = none, 1 = 30 ms delay.
P Write precompensation, bit 1. 0 = enable.
I_1 The interrupt features are not used.

The status register bit meanings change, according to the type of command being issued. Here is a summary.

Bit	Name	Status Register Meaning
7	Motor On	Reflects status of MOTOR ON line
6	Write Protect	During a write, set on indicates a write protected disk.
5	Record Type / Spin Up	Type I commands, set on indicates motor spin up sequence has completed six revolutions. Type II and III commands, shows record type, where 0 = data mark, 1 = deleted data mark.
4	Record Not Found	Set on, means track, sector or side not found.
3	CRC Error	If bit 4 is also set on, bit 3 is set on if an error is found in an ID field. Otherwise it indicates an error in the data field.
2	Lost Data/ Track 00	Set on, computer did not respond to DRQ in one byte time. Resets to zero when updated. During Type I commands, set on indicates track 00 has been reached.
1	Data Request/ Index	Copy of DRQ line output. Used in polled read and write of the data register, as done by Applix. When set on, indicates the data register is full on a read, or empty on a write. Resets to zero when updated. On Type I commands, shows status of index pulse.
0	Busy	Set on, indicates a command is being executed.

Format byte patterns

The general pattern of disk bytes follows the IBM double density standard. Note that some of the bytes on disk are generated internally by the controller, not by your format byte pattern. After the index pulse, there are 60 bytes of 4EH. This is followed by the repeated sector pattern, approximately as follows:

Name	Number	Value	Description
	12	00	Synchronisation bytes
	3	F5	Writes A1
	1	FE	ID field address mark
Track	1		Track number (0 thru 4C, potentially F3)
Side	1		Side number 0 or 1
Sector	1		Sector number (1 thru 5, potentially F3)
Length	1	03	Sector length (00 = 128, 01 = 256, 02 = 512, 03 = 1024)
	1	F7	(2 CRC's written)
Gap II	22	4E	
	12	00	
	3	F5	Writes A1
	1	FB	Data address mark
Data	1024		Your data
	1	F7	(2 CRC's written)
	24	4E	

Disk select latch

The disk select latch (U16, 74LS273) at I/O address $08 produces signals which are set up by the Z80 and which are used for selecting between multiple drives (DS0 and DS1), ejecting the disk, selecting the desired side of the disk, etc. These are commands that can not be directly produced by the disk controller chip.

Bit 7	80H	MAP
Bit 6	40H	BANK
Bit 5	20H	Side select, side 0 or 1 or disk drive
Bit 4	10H	Motor on, and LED
Bit 3	8H	Drive select 1
Bit 2	4H	Drive select 0
Bit 1	2H	/EJECT line
Bit 0	1H	/INUSE line

If the Z80 reads from the same I/O location ($08, at U15, 74LS244), you can obtain the status of the disk drive /DISK CHANGE and /READY lines.

Bit 1	$2H	/DISK CHANGE
Bit 0	$1H	/READY

Summary

The information in this section provides sufficient detail for the programmer to start working on low level disk routines. Applix does not, however, suggest or encourage the use of this information within normal programs. You should use the existing block device routines and system calls detailed in the *Applix Programmer's Manual*. In particular, if you write your own low level routines, don't expect Applix to ensure that other programs always run with them.

4
Construction Technique

The recommended construction method is called Progressive Assembly and Test (PAT) and is the technique that will be described in this construction manual.

PAT involves breaking the construction into a number of small, simple steps, essentially breaking the SSDCC's complex circuit into a number of smaller ones. As each step is completed, a number of simple checks are done ensuring that this step is working. If something is wrong, it is corrected at this stage, before proceeding to the next step. The steps are arranged in such a way that each will work only if the ones preceding it work, so you **must never** go on to the next step until the present one works.

By constructing the SSDCC in this fashion, at the completion of the last step, everything should work. Also, by breaking the circuit up into a number of smaller ones, it gives a better understanding of how the SSDCC functions.

Who can build this kit

It is expected that you have constructed electronic kits before (e.g. the 1616) and/or have experience in digital electronics at either a hobbyist or professional level. If you are a complete beginner it might be best to try a smaller project first or to make arrangements with a friend to help you just in case you get into trouble.

'Fix It' guarantee

If your built-up SSDCC fails to operate and you can not trouble shoot the fault(s), you may wish to take advantage of the Applix 'Fix It' guarantee. Our technicians will check and repair your SSDCC as required, for the flat fee of $100.

This fee includes replacement of any necessary components that may have been damaged during or after construction. Your SSDCC must be unmodified and must be constructed using decent IC sockets to use the 'Fix It' service.

If a kit is so badly constructed as to make repair impossible, we reserve the right to return the computer (and the service fee) in the condition received. The 'Fix It' fee covers only the actual SSDCC; it does not cover your 1616, power supply, keyboard, other expansion boards etc. It is a 'Fix It' service only, not a construction service.

Built and tested

If you feel you are not capable of building the SSDCC, you may wish to return the kit in its original condition and purchase a built and tested SSDCC or arrange a refund. We can not accept a return once construction has commenced, if any of the component packs have been opened, or if the kit or cartons have been damaged in any way. Please contact Applix concerning pricing of the built and tested SSDCC, and the returns procedure.

Getting started

Before getting started there are a few extras you will have to get. Below is a list of what would be considered the minimum required.

( ) A pair of fine cutters. (Dick Smith Cat #T-3205 or similar)
( ) A fine tipped soldering iron. (Dick Smith Cat #T-2000 is ideal)
( ) A roll of fine solder. (Multicore 60% tin/40% lead with no copper, medium active flux is recommended)
( ) Small screwdrivers etc.
( ) Alligator clips, small pieces of wire etc.
( ) A working 1616 motherboard, power supply, keyboard, etc.
( ) Cabling: You will be required to make cables to connect the SSDCC to your disk drive(s), as discussed in appendix A.
( ) A 3.5" double sided 80 track floppy disk drive(s). Some people use 5.25" drives, but these are not considered standard, and are best used as a second drive.
( ) Test Equipment. The minimum required here is a multimeter and a logic probe (Dick Smith cat #Q-1272 or similar). An oscilloscope is an advantage, but if all goes well, is not actually required.

5
Parts List

Before doing anything, check off the parts list against the parts supplied with your SSDCC kit. Do the same for any expansion kits (SSDCC I/O Kit etc.) that you may also have purchased. If you find anything incorrect, or any parts missing, you should contact Applix immediately for the part(s) to be sent or replaced.

Basic kit

Resistors

( )	R1	330	(org-org-brn)†
( )	R2	3k3	(org-org-red)
( )	R3	3k3	(org-org-red)
( )	R4	3k3	(org-org-red)
( )	R5	3k3	(org-org-red)
( )	R6	470	(yel-pur-brn)

†Notes: R1 may be marked 680 on some boards; use the 330Ω supplied. All resistors are 5% tolerance (this means they have four colour bands, the first three of which are indicated above, and the last band is gold).

Resistor networks

( )	RN5	180	(181) (10 pin, 9 resistors)

Notes: RN1,2,3 and 4 are used in the SCSI hard disk interface and are supplied in the optional SSDCC SCSI Kit. Resistor networks must be inserted the correct way round. Look for a small dot at one end. As component values in this area are not critical, your kit may come with resistors of slightly different values.

Capacitors

( )	C1	10uF	tantalum (106)
( )	C2	.1uF	monolithic (104)
( )	C3	.1uF	monolithic (104)
( )	C4	.1uF	monolithic (104)
( )	C5	.1uF	monolithic (104)
( )	C6	.1uF	monolithic (104)
( )	C7	.1uF	monolithic (104)
( )	C8	.1uF	monolithic (104)
( )	C9	.1uF	monolithic (104)
( )	C10	.1uF	monolithic (104)
( )	C11	.1uF	monolithic (104)
( )	C12	.1uF	monolithic (104)
( )	C13	.1uF	monolithic (104)
( )	C14	.1uF	monolithic (104)
( )	C15	.1uF	monolithic (104)
( )	C16	.1uF	monolithic (104)
( )	C17	10uF	tantalum (106)
( )	C18	.1uF	monolithic (104)
( )	C19	.1uF	monolithic (104)
( )	C20	.1uF	monolithic (104)
( )	C21	.1uF	monolithic (104)

( )	C22	10uF	tantalum (106)
( )	C23	10uF	tantalum (106)
( )	C24	.1uF	monolithic (104)
( )	C25	.1uF	monolithic (104)
( )	C26	10uF	tantalum (106)
( )	C27	.1uF	monolithic (104)

Notes: The kit may be supplied with either low leakage electrolytic or tantalum capacitors. Tantalum and electrolytic capacitors must be installed the correct way round. Monolithics (usually small blue ones) do not.

Integrated Circuits

	Position	Number	Description
( )	U1	6264/8264	8K static RAM
( )	U3	27128/27256	SSDCC-B ROM
( )	U4	Z80H	Z80H CPU
( )	U6	74LS74	Dual D type flip flops with clear
( )	U7	74F74	Dual D type flip flops with clear
( )	U8	ZPAL 16L8	Z80 decoding PAL (8K)
( )	U11	74LS04	Hex inverters
( )	U13	74LS138	3 to 8 line decoder
( )	U14	74LS74	Dual D type flip flops with clear
( )	U15	74LS244	Tri-state buffer
( )	U16	74LS273	Octal latch
( )	U17	74LS74	Dual D type flip flops with clear
( )	U18	74LS74	Dual D type flip flops with clear
( )	U19	74LS38	Quad 2 input NAND gates (OC)
( )	U20	74LS32	Quad 2 input OR gates
( )	U21	74LS06	Hex inverters (OC)
( )	U22	74LS244	Tri-state buffer
( )	U23	WD1772	Western Digital disk controller
( )	U24	SPAL 16L8	MC68000 decoding PAL
( )	U25	74LS30	8 input NAND gate
( )	U26	74LS30	8 input NAND gate
( )	U28	74LS374	Tri-state octal latch
( )	U29	74LS374	Tri-state octal latch

Notes: Integrated circuits **must** be inserted the correct way round. Look for a small indentation at one end, and match it with that marked on the circuit board.

U5,9,12 are all spare and are not used.

U7 may be marked 74LS74 on some boards.

U10 is used in the SCSI hard disk interface and is supplied in the optional SSDCC SCSI Kit.

U21 may not be an LS version, and may be marked 7406.

U27,30,31,32,33 are used in the serial port and are supplied in the optional SSDCC I/O kit.

U2 allows expansion of the SSDCC memory and is not supplied (or required) for floppy disk operation. If you have additional RAM, you may add it once the SSDCC is built and completely tested. No DIP switches, links or software changes are needed. If extra memory is added, Greyham Stoney's Shareware #7 disk drive utilities will make use of it.

Miscellaneous

( )	LED1	5mm red LED
( )	SW	4 way DIL switch (only SW1 and SW2 are used)
( )	OSC	16.000MHz TTL oscillator
( )	SSDCCPCB	SSDCC printed circuit board

Connectors

( )	Floppy disk data	34 way right angle IDC header connector
( )	Z80 interrupts	6 way IDC strip (2x3 pins)
( )	Floppy disk power	4 way R/A PCB mount mini KK (M5046-4A)
( )	Edge connector	80 way R/A PCB mount (2x40)

IC socket kit

( )	15 x	14 pin IC socket
( )	1 x	16 pin IC socket
( )	7 x	20 pin IC socket
( )	4 x	28 pin IC socket
( )	3 x	40 pin IC socket

SCSI kit

Integrated Circuits

( )	U10	5380	SCSI controller

Resistor networks

( )	RN1	330	(331)
( )	RN2	220	(221)
( )	RN3	330	(331)
( )	RN4	220	(221)

Connectors

( )	SCSI interface	50 way right angle IDC header connector
( )	Hard disk power	4 way R/A PCB mount mini KK (M5046-4A)

Note: SSDCC-S ROM required, and 64K memory recommended, but not required.

I/O kit

Integrated Circuits

( )	U27	Z8530	Dual serial communications controller (SCC)
( )	U30	1489	RS232C line receiver
( )	U31	1488	RS232C line driver
( )	U32	1489	RS232C line receiver
( )	U33	1488	RS232C line driver

Connectors

( )	Serial A	PCB mount right angle male DB9
( )	Serial B	PCB mount right angle male DB9

( )	SJA-0	18 way IDC strip (2x9 pins)
( )	SJA-1	18 way IDC strip (2x9 pins)
( )	SJB-0	18 way IDC strip (2x9 pins)
( )	SJB-1	18 way IDC strip (2x9 pins)
( )	Shunts to suit strips (total 26)	

6
Construction: Part 1

1616 requirements

Before you begin construction of the SSDCC you must first solder in one or more expansion sockets (available from Applix, and elsewhere) to your 1616 mother board. After soldering in the connector(s), verify that your 1616 still works correctly.

Your 1616 should preferably be fitted with 1616/OS Version 4 EPROMs, although the SSDCC will operate from Version 2.0 up (the SSDCC will not operate in a 1616 containing Version 1 EPROMs). Disk users should update to the current 1616/OS, Version 4, as soon as possible, as most of the features mentioned in the version 4 manuals do not operate in version 2. We remind users that Versions 2 and 3 are no longer supported with bug fixes or programs. Last upgrade for these was Version 3.2c in early 1989.

The 6522 VIA modification (3K3 pullup resistors on pins 35, 36, 37 and 38, to +5 volts) should be made to your Applix 1616 motherboard. Otherwise your system may lock up at times (about one system in five exhibits this problem, which is due to a design problem in the 6522 chip).

Hints, tips and notes

You must always turn the power off and wait about 30 seconds before plugging and/or unplugging your SSDCC from your 1616 mother board and inserting and/or soldering in any components.

Many of the components are polarised. This means that they must go in a particular way. These include the LED, oscillator, tantalum or electrolytic capacitors, resistor networks and the integrated circuits (ICs).

Incorrect insertion will probably damage the component and possibly others connected to it. The SSDCC PCB's component overlay (the white outlines on the board) indicates the orientation of polarised devices. Integrated circuit orientation is indicated by a small indentation at one end, which should align with the same type of marking on the printed circuit board. Resistor networks are marked with a small dot at one end. Tantalum and electrolytic capacitors are marked on the board with a +, however in some brands of capacitor, the opposite leg is marked with a -.

Although resistors and monolithic caps are not polarised, and will not cause problems regardless of which direction they face, it is advisable to orient them all the same way. This makes reading the values easier later on and also makes the board much neater. For resistors, orienting them left to right (with the gold band to the right) makes it easier to read them.

Integrated circuits and sockets

We highly recommended that the SSDCC be constructed using IC sockets. This makes construction and replacement of faulty ICs easier, and it also allows you to take advantage of the Applix 'Fix It' service, if worst comes to worst. It is very important to use high quality sockets, as intermittent problems caused by using low quality sockets are hard to find, and make the operation of your computer unreliable.

Orientation of the IC sockets is also very important. Most sockets have a notch at one end to indicate pin 1. Match this with the orientation markings on the SSDCC PCB.

Great care must be taken when handling ICs. The Z80H, EPROM, 5380, SCC and RAM ICs are particularly sensitive to static electricity. ICs must be inserted the correct way and in the correct place. Before inserting an IC always check the following:

- You have turned the power off to your 1616 and removed the SSDCC.
- You have the correct IC.
- You are plugging it into the correct socket.
- You are plugging it in the correct way round.
- You haven't bent any pins under or out.
- All of the above again!

It is a good idea to mark pin 1 of each IC with a small dop of liquid paper or to a make a little stick-on.

Soldering

The most expensive part of the whole SSDCC kit is the printed circuit board. It is also the hardest thing to replace if damaged. The board has been especially designed for manual construction. It has a solder resist mask on both the component side and the wiring side for easier soldering and a comprehensive component overlay to aid construction. The components are all soldered on the side that has the white component overlay.

A fine tipped soldering iron is a must. Any tip larger than about 4mm may damage the board and damage the tracks or pads. An iron temperature of about 350 degrees is recommended. It is advisable to fit a new tip to your iron before starting this project and, if your soldering is a bit rusty, spend a bit of time practicing on a junk board. A few minutes practice could be the difference between success or failure.

Use only high quality fine solder and clean the tip of your iron on a damp sponge regularly. Always apply solder to the hot joint, don't try to take it to the board on the tip of your iron. The solder should flow easily through the plated-through holes; the solder should not flow through on to the component side of the board. Just one more point- keep your soldering neat! Do not use excessive amounts of solder. This only results in an ugly board and increases the risk of solder splashes and bridging. Check your soldering regularly. Try and keep everything neat and don't rush!

Board modifications

Some future upgrades of the SSDCC require modifications to the existing printed circuit board. These modifications are easier to make prior to construction. If you contemplate upgrades, you may prefer to make the following modifications prior to commencing construction.

- Drive select modification, for future use of more than two drives. Drive Select line 1 (pin 12 of the drive connector) and Drive Select line 3 (pin 6 of the drive connector) are linked, under the drive connector. Cut the track between them, being careful that DS1 (pin 12) remains connected to pin 3 of U19, a 74LS38. This will require a wire link.
- Five volt line, labelled B, to pin 25 of SCSI connector. This line needs to be cut if you intend running some brands of SCSI hard disks. As some brands may be damaged, cutting the line now may be worthwhile. You should note that other brands of SCSI drive will not operate correctly unless this +5 volt line is connected.

- Hard disk interrupt modifications, used with Mark Harvey's Version 2.1, (and all *odd* numbered versions of hard disk software - even numbered versions are identical, except they do not use interrupts, and don't require board modifications). These require an extra chip, and are best done only if you intend to obtain a hard disk in the near future. Details are in the hard disk supplement to this manual.

Construction

As mentioned in the introduction, we will construct the SSDCC in small steps using the Progressive Assembly and Test (PAT) method. Remember, it is pointless to jump steps, or to proceed to the next step until it is verified that the current one is working correctly.

Step 1
Checkout PCB

Just before commencing construction, spend a few minutes examining your board. Hold it up to the light and look closely at the back and front for anything odd. If you find something wrong contact, Applix before going any further. It is pointless to start soldering a board that is faulty.

With your multimeter switched to resistance, measure for open circuits between the following points:

( ) COM and +12 ( ) COM and +5
( ) +12 and +5

The easiest place to locate these point is at the floppy disk power connector (marked by a 1 cm white rectangle near the top of the board close to U27). This has all the power lines clearly marked on it.

PROBLEM

It is highly unlikely that you should find a short between any of these points. If you do, contact Applix for a free board replacement.

Finally, just before we begin construction, gently wipe the board with a clean soft cloth.

Step 2
Resistors

Insert and solder into place all the resistors. Just before soldering them in double check that the resistors are the right values. After soldering, turn over the board and neatly trim off the leads.

Step 3
IC Sockets, resistor networks

Solder in all the IC sockets. It is best to do about 10 at a time. Make sure that you solder the correct sized socket into each position and that the notch on the IC socket matches the small dot identifying pin 1 on the PCB.

It is a good idea to solder in the top left hand pin and the bottom right hand pin of the socket first. Then holding the PCB upright, gently push the socket from the top side of the board whilst touching the soldered pins with your iron to make sure that the socket is flush with the board before soldering it in completely.

Solder in the resistor network (including RN1,2,3 and 4 if you have the SSDCC SCSI kit).

Match the small dot identifying pin 1 of the resistor network with the dot on the PCB.

Step 4
Edge connector, LED

Solder in LED1, ensuring you have it the correct way round. The LED has one leg longer than the other. This is the anode, it connects to the positive side of the circuit, and is indicated by the triangle part of the symbol on the PCB. The short leg is the cathode, often marked *k*, connects to the negative side of the circuit, and is indicated by the bar part of the symbol on the PCB, and also sometimes has a flat face on one side of the rim of the LED.

Next solder in the 80 way right angle edge connector, with the pins facing down off the edge of the PCB.

Plug the SSDCC into 1616 mother board. Connect your power supply and switch it on.

Measure all the power supplies for the correct voltages. Verify your 1616 is still working.

PROBLEM

If there is any smoke, explosions etc, turn the power off immediately. Give the PCB a good visual inspection and double check for shorts, solder-bridges, reverse polarisation etc. Replace any damaged parts. Test your power supply.

Step 5
Monolithic caps

Solder in all the monolithic caps (these are usually the small blue ones). Do not solder in the tantalum capacitors as yet (the tantalum have a + sign next to their position on the PCB component overlay).

Plug the SSDCC into 1616 mother board. Connect your power supply and switch it on.

Measure all the power supplies for the correct voltages. Verify your 1616 is still working.

PROBLEM

Double check soldering and SSDCC PCB. If the fault cannot be found cut one leg of each cap (on the top side of board) until the faulty cap is located.

Step 6
Tantalum caps

Solder in all the tantalum caps. Make sure that you have every one soldered in the correct way round. The PCB is marked with a + next to the positive leg of each tantalum. The tantalum will usually have a (very small) + next to one of its legs.

If you have been supplied with electrolytic capacitors, some brands have the negative leg marked with a -. In this case, the - leg goes in the PCB hole away from the + marking.

Plug the SSDCC into 1616 mother board. Connect your power supply and switch it on.

Measure all the supplies for the correct voltages. Verify your 1616 is still working.

PROBLEM

Double check orientation of caps, your soldering and PCB. If the fault cannot be found cut one leg of each cap (on the top side of board) until the faulty cap is located.

Step 7
Connectors, links

Solder in the floppy disk power connector (also the hard disk power connector, which is just next to it, if applicable). This is a small, 4 pin, right angle connector. The pins face the top (closest) edge of the PCB, and the plastic tab (if it has one) is between the legs and the board.

DIL switch. This should be oriented so that the switch numbers read from the top edge of the board down. They will usually read 1 to 4, but some switches may be marked 0 to 3. We have assumed they are marked 1 to 4. Only the first two are used.

The 34 way right angle floppy disk data connector (also Serial and SCSI connectors if you have them) should be soldered so that the pins face the closest edge of the PCB.

Check the power supply voltages again.

Note: The DIL switch should be soldered in so that SW1 is towards the top of the SSDCC board. Also note that the two serial connectors are males (pins).

Next solder in the strapping blocks (a 2 by 3 set of metal pins) as follows:

‘Z80 INT’ strapping block 2 x 3 pins (below U11)

Also, if you have the I/O kit, solder in SJA-0, SJA-1, SJB-0 and SJB-1 (all 2x9 pins).

Note: Do not install any straps on the blocks as yet.

Plug the SSDCC into 1616 mother board. Connect your power supply and switch it on.

Measure all the supplies for the correct voltages. Verify your 1616 is still working.

PROBLEM

Give the PCB a good visual inspection and double check for shorts, solder-bridges etc.

7
Construction: Part 2

Guide to testing, and inserting ICs.

From here on the format of construction changes. The following six instructions detail how to proceed through each step and should be used with reference to the circuit diagram and PCB component overlay.

POWER OFF

You must always turn the power off and wait about 30 seconds before unplugging the SSDCC from your 1616 mother board and before the insertion or removal of any components. 'Power off' may be followed by a few additional instructions to be completed after the board has been turned off.

INSERT

This is always followed by a list of ICs to be inserted. Great care must be taken when handling ICs. The Z80H, EPROM, 5380, SCC and RAM are particularly sensitive to static electricity. ICs must be inserted the correct way and in the correct place. Check the following every time you are about to insert an IC:

- You have turned the power off to your 1616 and removed the SSDCC.
- You have the correct IC.
- You are plugging it into the right socket.
- You are plugging it in the right way round.
- You haven't bent any pins under or out.
- All of the above again!

It is a good idea to mark pin 1 of each IC with a small drop of liquid paper or to a make a little stick-on.

VERIFY

This could be a list of tests to complete and tick off or a list of instructions to complete and verify. It is assumed that the constructor has a multimeter, logic probe and/or an oscilloscope for such tests. It is safe to assume that you must plug the SSDCC into your 1616 and then turn the power on at the start of 'verify'.

PROBLEM

This instruction always follows 'Verify'. If you can not verify a test 'Problem' suggests possible causes and/or fixes.

SWITCH

Here you should press your 1616's reset switch, and dial up the correct 'test' on part of the DIL switch (SW2). There are four tests, (labelled 0 to 3) however as there is only one section of DIP switch available, the tests are selected by toggling the switch after a reset.

The switch should be 'on' (closed) at reset to start test 0. Switching it 'off' then starts test 1, 'on' next starts test 2, while 'off' again starts test 3. Full details of the switch settings and their functions are given in appendix B. An assembly source listing for the Z80's test code is given in appendix C.

Step 8
Z80 CPU timing

POWER OFF

Solder in the 16 MHz oscillator (OSC), being careful to get it the correct way round. Match the small dot identifying pin 1 of the oscillator with the corresponding white dot on the PCB.

	Socket	Part	Function
INSERT	U7	74F74	Dual D flip flop (marked 74LS74 on some boards)
VERIFY	( )	U7/3	16MHz
	( )	U7/5	8MHz

Notes This verify instruction means you should verify pins 3 and 5, of the 74F74 integrated circuit you just plugged in to position U7, are oscillating. The pins are numbered anti-clockwise from the notch, starting at pin 1, which is to the left of the notch (we mention this in case all your clocks are digital!)

The 16MHz and 8MHz indicate which signal you are checking. The name used will usually correspond with that used in the schematic, and in the earlier explanatory notes about the circuit. In all cases, the name will be derived from the function of the circuit being tested.

Verification can only be done while power is on. The signals you are verifying in this test can be best examined using an oscilloscope with a 20 MHz bandwidth. If you don't have easy access to one, continue to the next test.

PROBLEM

If something is not working suspect U7 or OSC.

Step 9
Processor and decoding

POWER OFF

Set SW1 for the size of the EPROM (U3) being inserted, as tabled in appendix B. This will almost always be a 27256, in which case SW1 must be ON (closed).
Ensure that SSDCC SW2 is ON, or in test mode.
This section includes **serveral** changes to the test switch settings, followed by checking of the SSDCC LED, or of signal levels. Be careful you don't skip any of the sections labelled SWITCH or VERIFY.

	Socket	Part	Function
INSERT	U3	27256	EPROM
	U4	Z80H	CPU
	U6	74LS74	Dual D flip flop
	U8	16L8	ZPAL Z80 decode (may be tagged as ZPAL or U8)
	U11	74LS04	Hex inverters
	U13	74LS138	3 to 8 line decode
	U15	74LS244	Tri state octal buffer
	U16	74LS273	Octal latch
	U22	74LS244	Octal buffer

VERIFY ( ) LED1 on the disk controller board should glow dimly, indicating that SSDCC test 0 is running. This tests the CPU, EPROM, address decode and support circuitry. U15 isn't needed at this point, except for allowing the test results to be indicated on the LED (early manuals inserted it too late in the test proceedure, leading to failure of the tests).

PROBLEM

If the LED does not glow (or for peace of mind, even if it does):
Check the orientation of the LED, and that it is functional.
Check that /NMI (U4/17), /INT (U4/16) and /BUSRQ (U4/25) are high.
Check that /RESET (U4/26) goes low when the 1616's reset switch is pressed.
Check that /MREQ (U4/19) is pulsing.
Check that /IORQ (U4/20) is pulsing.
Check that /WAIT (U4/24) pulses low during I/O reads and writes, and during ROM reads.
Double check that SW1 is correctly set for your type of EPROM (ON for 27256).
Suspect U3, U4, U6, U8, U11, U13, U16.
Check for opens, shorts, interconnections, etc on the Z80's control, data and address buses.

VERIFY	( )	U8/18	/SCCCS	Pulsing low
	( )	U8/19	/SCSICS	Pulsing low
	( )	U8/17	/FDCCS	Pulsing low
	( )	U8/16	/ROMSEL	Pulsing low
	( )	U13/12	/ZWRDAT	Pulsing low
	( )	U13/13	/ZINT	Pulsing low
	( )	U13/14	/LATCHW	Pulsing low
	( )	U13/11	/PORTR	Pulsing low

Some logic probes may simply show these lines pulsing, and not indicate them pulsing low. U8/16 /ROMSEL may only show low. You probably should obtain a better quality logic probe!

PROBLEM

Suspect U8, U13. Trace back to U4.

SWITCH Select SSDCC test '1', by switching SW2 OFF (that is, SW2 was ON at reset or power up, and is now switched OFF). This test makes the LED pulse irregularly, at about 2 to 3 times a second.

VERIFY	( )	U4/22	/WR	Pulsing low
	( )	U1/20	/SRAM0SEL	Pulsing low

Check that the Z80's (U4) address lines (A0-A15) are all pulsing low. These are pins 30 to 40, and pins 1 to 5.

PROBLEM

Suspect U4, U8.

Step 10
Z80 RAM

POWER OFF

Ensure switch SSDCC SW2 is ON for test mode.
This section tests RAM, and other Z80 functions. Ensure that you do not skip any sectioned labelled SWITCH or VERIFY.

INSERT	Socket	Part	Function
	U1	6264	8k RAM (part number will often vary)

If you have additional memory chips, or 32k RAM, these may be added once the SSDCC has been built and completely tested. There is no additional jumpering or software required when adding extra RAM.

VERIFY ( ) LED1 on the disk controller board should glow dimly, indicating that SSDCC test 0 is running.

PROBLEM
Suspect U1.

SWITCH Select SSDCC test ‘2’, by switching SW2 OFF, then ON. Remember that SW2 should have been ON at reset or power up, and is now turned OFF, then ON, to cycle through to test ‘2’.

VERIFY ( ) LED1 on the disk controller board should flash, somewhat irregularly, at about 2 to 3 times a second, indicating that SSDCC test 2 is running. This test writes $5A to RAM location $6000, then reads it back and echoes the value to the latch (U16, 74LS273).

PROBLEM
Suspect U1.
Check for opens, shorts, interconnections, etc on the Z80’s control, data and address buses.

SWITCH Select SSDCC test ‘3’, by setting SW2 OFF. That is, from a reset or power up while SW2 is ON, switch it OFF, ON, OFF.

VERIFY ( ) LED1 on the disk controller board should flash faster than test ‘2’, indicating that SSDCC test 3 is running. Flash rate is about 6 per second.

PROBLEM
Suspect U1.
Check for opens, shorts, interconnections, etc on the Z80’s control, data and address buses.

Step 11
1616 interface

POWER OFF
Ensure switch SSDCC SW1-2 is ON for test mode.
This section tests the interface to the 1616 motherboard. Again ensure that you do not skip any section marked SWITCH or VERIFY.

INSERT

Socket	Part	Function
U14	74LS74	Dual D flip flop with clear
U17	74LS74	Dual D flip flop with clear
U18	74LS74	Dual D flip flop with clear
U24	16L8	SPAL 68000 decoding PAL (may be marked SPAL or U24)
U25	74LS30	8 input NAND gate
U26	74LS30	8 input NAND gate
U28	74LS374	Tri state octal latch
U29	74LS374	Tri state octal latch

VERIFY ( ) LED1 on the disk controller board should glow dimly, indicating that SSDCC test 0 is running.

Since enough of the PCB is now running for the Applix 1616 to identify it, you will probably get error messages such as `Disk controller timeout`, and/or `No TXRDY: Command $0B` on your display. This is perfectly normal at this stage.

PROBLEM

Suspect U14, U17, U18, U24, U25, U26, U28, U29.
Check for opens, shorts, interconnections, etc on the Z80's control, data and address buses.

VERIFY ( ) Type the 1616/OS command `MDB FFFFC1 FFFFC1`.
This should produce a line like

```
-00FFFFC1 -E2                              ->_<-
```

The value -E2 will probably be different each time.

PROBLEM

If this generates a bus error then suspect U25, U26 or U24.

SWITCH Select SSDCC test '0' by closing SW1-2 and resetting the 1616. Amongst other things this test writes bytes to the ports which the SSDCC's Z80 uses to communicate with the 1616.

VERIFY ( ) Enter the 1616/OS command `MRDB FFFFC1`. This continually dumps the contents of the data latch, U28, which is being written by the Z80; the displayed 2 digit number should be rapidly changing. Use Ctrl c to stop the test.

PROBLEM

If something is amiss, verify that the Z80 is running test 0. If so, then edit, assemble and execute the following 1616 assembler program:

```
                org       $4000
004000  1038FFC1 loop:    move.b    D0,$FFFFC1
004004  60FA               bra       loop
                           end
```

When this program is running, verify that /SRDDAT (U24/14) is pulsing low, enabling U28 onto the 1616 data bus.
Suspect U18, U24, U25, U26, U28.

Step 12
Disk interface

POWER OFF

Switch SSDCC SW1-2 OFF. This time you are **not** running a test. This time you try to get the disk drive working.

Note that more things can go wrong at this stage than at any other. Most of the things that go wrong are due to the disk drive not being set up correctly. Unfortunately, since almost any disk drive can be used, this is an area in which this manual is of least help to you.

INSERT

Socket	Part	Function
U19	74LS38	Quad 2 input NAND
U20	74LS32	Quad 2 input OR gate
U21	74LS06	Hex inverters (open collector) (may be 7406)
U23	WD1772	Disk controller

Set up your disk drive for use as drive 0 (usually by jumper to D0, as per your disk drive manual).

If you do not have a disk drive manual, look on the drive for a small switch, or set of jumpers marked with some indication like "D0 D1 D2 D3" (or possibly "D1 D2 D3 D4"). The first disk drive in your system should have a header across the jumpers marked "D0". The second disk drive should have the header across the jumper marked "D1".

Many drives have a "Ready" line jumper, which sometimes must be moved.

The last (or only) drive should have a termination resistor in place. Look for a header marked with something like 'Term', and ensure it has a jumper. This is almost always present on 5.25 inch drives, and is essential. It is often not present in 3.5 inch drives, and does not appear to matter as much here. If you can't find it, try powering up without it.

Connect up the disk signal and power cables to your disk drive. You must be careful to connect these cables the correct way round. The four wire power cable should only fit one way. The 34 way data cable connectors usually have a plastic step in the middle, that fits in a corresponding cutout in the connectors on your PCB and your drive. If not, one edge of the cable should have a wire marked differently to the rest (usually a red or blue stripe). This edge should go to pin 1 of each connector, so look on your PCB and your drive for markers indicating pins 1 and/or 34.

Write protect the Applix 1616/OS disk supplied with your SSDCC, by opening its write protect hole if it is a 3.5" disk (or taping over the write protect notch if it is a 5.25" disk).
Note that you must use a bootable Applix disk (such as the User Disk supplied with your SSDCC) for tests.
Put the disk into the disk drive which is set up as drive 0. (Incidently, there is a very slight possibility of accidently damaging a disk if it is completely in the drive at the moment you power up, so the *Users Manual* recommends waiting a moment after power up before inserting the disk ... for these tests, take a chance, and put the disk in before powering up.)

VERIFY ( ) A sign-on message should indicate that the 1616 - Z80 communications are correctly established; the disk controller ROM version is displayed. If the disk drive usage light comes on and no horrifying messages emerge, then all is probably well.
( ) Exercise your disk system by formatting blank disks, copy files to /RD and making a backup copy of the 1616/OS disk provided.
If you happen to have extra RAM chips, or wish to use 32k RAM, this can be added once the SSDCC has passed these tests.

PROBLEM
If an error such as 'NO RDY' comes out then a probable culprit is the disk drive strapping (drive number, or termination usually).
Check also the disk drive power supply and signal cabling.
Suspect U19, U20, U21, U22, U23.

8
SSDCC Software

Interprocessor Communication

So far this manual has dealt with the internal actions of the SSDCC disk controller. Normally you will use the high level commands provided by 1616/OS, or the various ***syscalls***. However, to ease testing of the controller, the 1616 can send a number of middle level commands. These commands will be recognised by the SSDCC. The results of the command (or an error code) will be returned to the 1616.

The command port is located in the 1616 address space at $FFFFD1. Send the command byte to $FFFFD1. Then write the rest of the command parameters to the data port at $FFFFC1. Error messages or results appear on the data port.

The interprocessor communication commands are detailed below. Additional commands are available when using SSDCC cards equiped with the SCSI hard disk upgrade ROMS. In this description data going from the Z80 to the 1616 is represented in angled brackets: < >.

Abort 00

This command should not be used, and generally merely aborts.

Block read command 01

unit, blockhigh, blocklow,
<errorcode or 0> <1024 bytes>

The 1616 writes a $01 to the command port at $FFFFD1, after which it writes the following bytes to the data port at $FFFFC1: The unit number (0 for drive 0, 1 for drive 1), the block number (high byte first). After this the physical read is performed by the Z80 and an error code is returned to the 1616. If the errorcode is zero, the 1616 may read out the 1024 bytes of data.

A non-zero error code indicates that something went astray; an interpretation of the error code may be obtained with command 3 (see below).

Block write command 02

unit, blockhigh, blocklow, data
<errorcode or 0>

Similar to the block read command, except that the 1024 data bytes are sent to the Z80 and then the 1616 must wait for the physical write to complete before an error code is returned.

Display error message command 03

errorcode
<string> <0>

The Z80 is sent the error code byte. The Z80 then sends down a string for human interpretation, terminated by a zero byte. The various strings have differing lengths.

Format command 04

unit $B5 $7E ntracks skewtable
<errorcode or 0>

To physically format a disk, the 1616 selects command 4. After this send the unit number (0 or 1), and write the values $B5 and then $7E to the data latch as a check against accidental issuing of format commands. Follow this with the number of tracks on the disk (usually 80 or 40), then the 10 byte sector skew table. Wait for an error code.

The sector skew table consists of two consecutive 5 byte tables. The first is for side 0 of the disk, the second for side 1. All sectors receive the same interleave pattern with this format command. The Z80 driver code expects sectors to be numbered from 1, not from 0. The skew table sent by the `blockdev.xrel` utility program is:

Side 0 1, 4, 2, 5, 3
Side 1 3, 1, 4, 2, 5

Format type 2 command 05

unit number

This format command permits interactive track-by-track formatting and sector skewing. Its use is not documented by Applix, and it can be considered abandoned by them for the moment. However it is used by Greyham Stoney's software for special formats.

Flush buffers command 06

Flushes the disk buffers, if any. No action if there are no buffers. If there is no action for a few seconds, check for a changed disk.

Read Z80 RAM command 07

Z80addrh, Z80addrl, lengthh, lengthl
<data>

This command permits the 1616 to read 'LENGTH' bytes of the Z80's memory, starting from the specified Z80 address.

Write Z80 RAM command 08

Z80addrh, Z80addrl, lengthh, lengthl, data

This command permits the 1616 to write 'LENGTH' bytes to the Z80's memory, starting at the specified Z80 address.

Call a Z80 program command 09

Z80addrh, Z80addrl

This command causes the Z80 to perform a 'CALL' instruction to the supplied address. Upon return from the called program the Z80 resumes normal operation.

Read Z80 ROM version command 0A

<ROMversion>

The Z80 returns a version byte for its current ROM.

O/S version byte 0B

Version byte

1616/OS tells the disk controller which version it is.

Set floppy disk step rate command 0C

unit, rate

This command is used to change the floppy disk(s) head step rate, to suit slower drives, as follows:

Value	Step rate used
0	2 ms.
1	3 ms.
2	6 ms. (default)
3	12 ms.

You may notice that this does not correspond with the step rates quoted in some WD1772 specification sheets. The specification sheets are wrong.

Disk change detect 40

unit number

The 1616 sends this command ($40, or 64) when it detects a change of disk in the unit. This is sent only if the disk controller EPROM > version 1.4.

Error messages produced by the disk controller software

The disk controller software produces various error messages. Some are first detected by the 1616's processor and the error messages for these are produced by the 1616/OS block driver software. Other errors are detected by the Z80 and are reported to the 1616 by means of an error code; when the 1616 encounters such an error code it fetches an English language interpretation of the error code from the Z80 and displays it. For test purposes, you may note that Greyham Stoney's shareware software includes very extensive error testing and error messages.

Error messages produced by the Z80

These refer to disk I/O failures of various forms.

Seek failure

A particular cylinder cannot be found; possibly because the disk drive has the incorrect number of tracks, or the disk is incorrectly formatted or poorly calibrated.

Drive busy(1)
Drive busy(2)

The WD1772 is permanently busy and cannot be interrupted out.

Write error(n) NN
Read error(n) NN

These refer to errors detected during physical reading and writing. The second number NN is a copy of the WD1772 status register at the time of detection of the error.

Format error

The WD1772 reported an error during disk formatting

No RDY signal

The /RDY signal from the disk drive (pin 34 of the 34 way signal cable) does not go low. This may be due to a cabling problem, incorrect disk drive strap setting or the absence of the selected drive.

Bad error number

This is the message string which the Z80 returns when asked to interpret an unimplemented error code. This usually means the disk controller was producing garbage.

Error messages produced by 1616/OS

These messages are produced by that section of 1616/OS which handles the communication with the disk controller - the /F0 and /F1 block device drivers. They generally indicate that something untoward has happened in the communication between the two processors, such as the Z80 coming unstuck.

Disk controller timeout

This means that the Z80 failed to respond in any way to a command which was sent to it.

No TXRDY: Command = $NN

When attempting to send the command NN to the controller's command port the 1616 could not detect a true level on the STXRDY handshake signal, meaning that the Z80 is probably ignoring its command port.

Using 1616/OS with multiple block devices

1616/OS is designed to support multiple block devices. Until Version 2 the only block device was the RAM disk, so the problem of specifying separate devices did not arise. With Version 4, you have available sixteen different block devices, normally:

/RD	The RAM disk
/F0	Floppy drive 0
/F1	Floppy drive 1
/H0	Hard drive 0
/H1	Hard drive 1
	etc.

Size limits are set for hard disks, mostly due to the RAM memory required by the bit maps. /H0 can be up to 40 megabyte, /H1 and others, up to 8 megabyte. An MRD is provided on the hard drive users disk for /H2, and /H3. If you require different size facilities, they should be provided by altering the MRDs.

The concept of a filename is extended to include a specification of the block device upon which the file is saved. This is done by prepending the block device driver's name onto the normal filename. For example:

`/F0/myfile` is a pathname for a file on floppy 0.
`/rd/MF2` is a pathname for a file on the RAM disk.

You may specify a pathname of this type wherever a normal filename is expected. Files whose names are given without a block device identifier are assumed to reside on the currently logged device.

The currently logged device is identified in the 1616/OS command line prompt. Initially this is the RAM disk (/RD). You can change the default block device by typing `cd` followed by its identifier. For example, typing `cd /rd` logs onto the RAM disk, `cd /f1` logs onto floppy disk drive 1, etc. Floppies may be changed without any need to inform the system (it will detect the change). Swapping floppies when files are still open will have predictably disastrous results.

9
Disk Organisation

A block device such as a floppy disk consists of the
- root block (block 0)
- block usage bitmap
- boot block
- root directory
- actual storage area.

Note that a ‘block’ is a 1024 byte unit of data.

The root block

The organisation of data on 1616/OS block device volumes such as the RAM disk and the floppies is determined by fields in the device’s block zero, referred to as the ‘root block’.

The root block is always at block zero of a device. The root block structure defines the structure of the disk. It contains the following fields:

Offset	Size	Name	Usage
0	ushort	NBLOCKS	Number of 1024 byte blocks on device (normally 800 for floppies).
2	ushort	SSOSVER	Version of 1616/OS under which the device was initialised ($30) for Version 3.
4	ushort	BITMAPSTART	The block at which the device block usage bitmap starts (usually $01).
6	ushort	DIRSTART	The block where the root directory starts (usually $03).
8	ushort	NDIRBLOCKS	The number of blocks in the root directory (usually $07).
10	ushort	REMOVABLE	Non-zero (usually $01) if the media is removable (that is, a floppy).
12	ushort	BOOTBLOCK	Number of block which contains boot code, to be loaded into memory and executed at reset time (usually $02).
14	ulong	SPECIAL	Randomised number for distinguishing disks one from another.
18	64 bytes	ROOTDIR	A directory entry which describes the root directory.

In detail:

NBLOCKS

The number of blocks on the device. This is read when the disk is logged. If a disk with a certain number of blocks is inserted into a drive which previously contained a disk which had a different number of blocks, the change is detected.

SSOSVER

This is the 1616/OS version number of the disk. The disk structure has changed slightly in the change from 1616/OS V2.4 (version = $24) to 1616/OS version 3.0 (version = $30).

BITMAPSTART

The number of the block at which the block usage bitmap commences.

DIRSTART

The number of the block at which the root directory commences. This field duplicates the BLKMAPBLK entry in the root directory entry ROOTDIR, but has been kept so that 1616/OS V3.0 disks may be read under 1616/OS V2.4

NDIRBLOCKS

The number of blocks in the root directory. This field duplicates the FILE_SIZE entry in the root directory entry ROOTDIR, but has been kept for 1616/OS V2.4 compatibility.

REMOVABLE

This flag is non-zero if the media is removable and the system has to check for swapped disks. There is a performance benefit when using non-removable media; in fact a floppy may be defined to be non-removable. If this is done the system must be reset when the floppy is replaced.

BOOTBLOCK

Contains the number of the block which is read in to memory at $3C00 when booting from disk. If this field is zero then no boot code exists.

SPECIAL

A randomised number which is (hopefully) unique for every disk. The system inspects this field and the DATE field in the root directory entry ROOTDIR for a change which indicates a disk swap.

ROOTDIR

This is a standard 64 byte directory entry. It describes the root directory of the disk, so bit 1 of the STATBITS field is set. The DATE field in this directory entry represents the date of creation of the volume and is also used for detecting disk swaps.

The block usage bitmap

Every block device has a bitmap associated with it. This records which blocks are currently used, and which are free to be used.

The bitmap is a sequence of bits. A one means that the corresponding block is used; a zero indicates that it is free. The most significant bit (bit 7) of the first byte in the bitmap corresponds to block 0; bit 6 of byte 0 corresponds to block 1, etc. The bitmap may extend over more than one block: one block contains 8192 bits, so one block of bitmap is needed for every 8 megabytes in the disk.

The BITMAPSTART field in the root block identifies the block at which the bitmap starts. If the bitmap is more than one block long, all the blocks must be contiguous, starting with the block identified by BITMAPSTART. The system uses the NBLOCKS field to calculate how many blocks are contained in the bitmap.

The bitmap usually starts at block 1 of a disk. It occupies however many contiguous blocks are required, based upon the number of blocks on the device.

The boot block

The boot block contains the boot program which the system executes at address $3C00 every time the system is reset.

The BOOTBLOCK field indicates whether or not the disk contains a boot block to be loaded at reset time and, if so, what block it is. Whenever the 1616 is reset (by powering on, pressing the reset switch or by ALT-control-R) the operating system performs all initialisation and then goes through all the block drivers in order (/RD first, then /F0 then /F1 then any others) looking for a device with its BOOTBLOCK field in the root block non-zero. When such a device is found the indicated block is loaded into memory and executed at address $3C00 in the 1616.

The current level of the reset (0, 1 or 2) and the block driver number from which the system is booting are passed on the stack at 4(sp) and 8(sp) respectively. This allows the boot code to perform whatever level of initialisation is needed.

Note that the ram disk may be used in this manner as a boot device. In the ram disk, Block 1 is the bit map block (rdbitmapblock), while Block 2 (rdcksmblk) contains the ram disk checksums. Block 4 is normally the first directory block in a ram disk, and four kbytes (enough for 64 files) are normally defined. Eight blocks are reserved when a ram disk is established.

Block 3 in the ram disk is reserved for booting, however it is not normally used. To use it, read in the root block, set the BOOTBLOCK field to 3, write out the root block again and then put your boot code in block 3. Remember that the 1616/OS OPTION 8 must be set to write enable the system tracks of a block device.

If the disk in drive 0 (/F0) does not contain a boot program then the system will attempt to boot from disk 1 (/F1). If there is no second disk or if the second drive has no disk in it then the attempt to read from the second disk will fail, taking a few seconds to time out. Look in the `/sys` directory of the 1616/OS user release disk for the file `bootv3.exec` or `bootv3`. This is the standard boot code. By changing the `bootv3.s` code, and assembling, you can readily alter what happens upon boot. An easy example would be changing the startup file name from `autoexec.shell` to `hello.shell`.

You can arrange to boot from the hard disk before searching for disks in drives /F0 and /F1. If DIP switch 2 (counting from 0) on the 1616 motherboard is open (off), and you have Version 4 ROMS, the 1616 will attempt to boot in the order /H0 /RD /F0 /F1 /H0 /H1 (this simply adds /H0 in front of the standard boot order).

The root directory

The disk root directory is a number of blocks reserved for the home directory of the disk. On floppies the `blockdev.xrel` program sets this up. The root directory should start after the root block, bitmap block and boot block.

The directory

The directory lies from the block indicated by ‘DIRSTART’ up to ‘DIRSTART + NDIRB-LOCKS - 1’. The 64 byte directory entry is as follows:

Offset	Size	Name	Usage
0	32 chars	FILE_NAME	32 byte null-terminated name
32	8 char	DATE	File creation date (yy/mm/dd/hh/mm/ss).
40	ushort	UID	User ID code
42	long	LOAD_ADDR	Load address of `.exec` file
46	long	FILE_SIZE	Length of file/No of dir blocks
50	word	STATBITS	Status bits, file attribute bits
52	short	BLKMAPBLK	The block which contains the file's block map block / directory start block
54	short	MAGIC	$d742 (42 is Version 4.2)
56	4 words	FFB	First four blocks of file

FILE_NAME

is the name of the file/directory which this directory entry describes. It contains no "/" characters. It is in upper case and should contain no control characters. The following characters may be used in file names: 0-9 @ A-Z. This is only enforced upon creation of the file (in the ***creat*** and ***rename*** system calls) so that any existing files which do not conform to this standard may be read, copied, executed or renamed.

If the first entry in this field is zero then the directory entry is not used. When a file is deleted by the ***unlink*** system call, the file's blocks are released and the first character of its filename is copied to the end of the file name (31 bytes in, which usually contains 0) before the first character is set to zero. This permits files to be un-deleted - the results may not be good if the disk has been written to since the file was deleted. Run `fscheck.xrel` across a disk after un-deleting a file.

DATE

is the date of modification of the file/directory which this directory entry describes. Adding new files to a directory does not affect the date in the directory's directory entry.

UID

one of 64k users. User 0 has all rights; any other user can be restricted.

LOAD_ADDR

is the address at which `.exec` files are to be loaded and executed. It is set to zero for other types of file.

FILE_SIZE

has a different meaning depending upon whether the directory entry describes a file or a directory. If it describes a file, the FILE_SIZE field contains the length of it in bytes. If the directory entry describes a directory (sub-directory), then the FILE_SIZE field contains the number of 1024 byte blocks which the subdirectory occupies. These blocks are contiguous for all directories.

STATBITS

contains file attribute bits. The following bits are defined:

Bit 0 If set, the file has been backed up somewhere.

Bit 1 If set, this directory entry describes a directory

Bit 2 If set, the file/directory described by this directory entry cannot be modified.

Bit 3 If set, Read permissions are ON for users with ID's other than 0.

Bit 4 If set, Write permissions are ON for users with ID's other than 0.

Bit 5 If set, eXecute permissions are ON for users with ID's other than 0.

Bit 6 If set, a symbolic link has been made.

Bit 7 If set, ffblocks (last 4 words) are valid address of file contents.

BLKMAPBLK

field use depends upon whether the directory entry describes a file or a directory. If it describes a file, then the BLKMAPBLK field contains the number of the block which contains a list of the blocks occupied by this file, in the order used. If the directory entry describes a directory (sub-directory), then the BLKMAPBLK field contains the number of the block at which the sub-directory's contents start.

MAGIC

If the magic number $d742 appears, it indicates that the last four words in the directory entry point to the blocks occupied by the contents of the file. The 1616/OS version number under which the file was created is also indicated (in this case, Version 4.2).

FFBLOCKS

Pointers to the block numbers of up to four blocks containing the contents of the file described by the directory entry. This avoids looking at the disk blockmap, but only works on files less than 4k long.

The 1616 disk drive utility program

Initialising block devices

The program `blockdev.xrel` permits the initialisation of block devices, formatting of floppies and the conversion of 1616/OS V2.X disks to 1616/OS V3.0

To run this program type

```
blockdev devname
```

Where `devname` is the name of the device (/F0, /F1, etc) which you wish to format, initialise, etc.

There are three levels of disk preparation available with this program. The most basic level is to simply alter the disk's boot sector program; you enter the name of the new boot program and this is written onto the disk. If no boot program exists on the disk then the disk is skipped in the booting sequence.

In the next level of disk preparation you may 'initialise' a disk. This recreates the disk's root block, bitmap and directory. All files are lost. The boot block needs to be rewritten after this.

The next level of disk preparation involves a physical format of the disk, followed by initialisation, followed by the boot code setup. This will only work on the /F0 and /F1 devices, since these are the only physical devices which early versions of this program know how to format.

Another option with this program is to upgrade a disk to 1616/OS V3.0. This alters the disk's root block into the correct format. The V2.X directory becomes the V3.0 root directory. The disk is still readable and writeable under 1616/OS V2.X. This option may be used for altering a disk's name, as well as upgrading the version.

10
Appendix A: Connector Pinouts

Floppy disk data connector

(34 way)

Pin#	Name	Description
1	/EJECT	Disk eject signal. (not used but normally still connected) (output).
2	/DISK CHANGE	Disk Change. (not supported on some drives) (output).
4	/INUSE	In Use signal line (output).
6,12	/DS3,1	Drive 1 select signal (output).
8	INDEX	Disk drive index signal (input).
10	/DS0	Drive 0 select signal (output).
14	N/C	Not connected.
16	/MOTORON	Motor on signal (output).
18	/DRC	Direction select signal (output).
20	/STEP	Disk head step signal (output).
22	/WD	Write data signal (output).
24	/WG	Write gate signal (output).
26	/TRK00	Track 0 signal (input).
28	/WRPT	Write protect signal (input).
30	/RD	Read data (input).
32	/SIDE	Side select signal (output).
34	/RDY	Ready signal (input).
3,5,7,9 11,13,15 17,19,21 23,25,27 29,31,33	COM	System common ground.

Hard disk power connector

(closest to C23)

Pin#	Name	Description
1	+5v	+5 volt power supply.
2	COM	System common ground.
3	COM	System common ground.
4	+12v	+12 volt power supply.

Floppy disk power connector

(closest to C21)

Pin#	Name	Description
1	+5v	+5 volt power supply.
2	COM	System common ground.
3	COM	System common ground.
4	+12v	+12 volt power supply.

SCSI connector

(50 way)

Pin#	Name	Description
2	B0	Data bit 0.
4	B1	Data bit 1.
6	B2	Data bit 2.
8	B3	Data bit 3.
10	B4	Data bit 4.
12	B5	Data bit 5.
14	B6	Data bit 6.
16	B7	Data bit 7.
18	PARITY	Data bus parity bit.
20,22,24	COM	System common ground.
25, 26	TERMN PWR	Normally connected to +5v. (as standard linking on SSDCC, may need to be cut for some SCSI drives).
28,30	COM	System common ground.
32	/ATN	Attention signal.
34	COM	System common ground.
36	/BUSY	Busy signal.
38	/ACK	Acknowledge signal.
40	/RESET	SCSI bus reset signal.
42	/MSG	Message signal.
44	/SEL	Select signal.
46	CTRL/DATA’	Control/Data signal.
48	/REQ	Request signal.
50	IN/OUT’	Input/Output signal.

All odd number pins (except pin 25) connect to system common ground.

Serial ‘A’ connector

(default pinouts only, assumes standard strapping)

Pin#	Name	Description
1	COM	System common ground.
2	CTS	Clear To Send (input).
3	RTS	Request To Send (output).
4	RxD	Receive Data (input).
5	TxD	Transmit Data (output).
6	+12v	+12 volt power supply.
7	-12v	-12 volt power supply.
8	DCD	Data Carrier Detect (input).
9	DTR	Data Terminal Ready (output).

Serial ‘B’ connector

(default pinouts only, assumes standard strapping)

Pin#	Name	Description
1	COM	System common ground.
2	CTS	Clear To Send (input).
3	RTS	Request To Send (output).
4	RxD	Receive Data (input).
5	TxD	Transmit Data (output).
6	+12v	+12 volt power supply.
7	-12v	-12 volt power supply.
8	DCD	Data Carrier Detect (input).
9	DTR	Data Terminal Ready (output).

Expansion connector pinout

Pin#	Name	Description
1,2	COM	System common ground.
3,4	+5v	+5 volt power supply.
5,6	+12v	+12 volt power supply.
7	-12v	-12 volt power supply.
8	-5v	-5 volt power supply.
9,24	D0-D15	Data Bus (D0-D15). Bidirectional, three state data bus.
25	/AS	Address strobe. This signal indicates that there is valid data on the address bus.
26	/UDS	Upper Data Strobe. This signal indicates that valid data is available on data bus bits D8-D15.
27	/LDS	Lower Data Strobe. This signal indicates that valid data is available on data bus bits D0-D7.
28	R/W’	Read/Write. This signal defines the data bus transfer as a read or write cycle. It also works in conjunction with /UDS and /LDS.
29	/DTACK	Data Transfer Acknowledge. This pin is used by expansion boards to determine when another board or the main board has responded.
30	/EXTDTACK	External Data Transfer Acknowledge. This open collector signal is used by expansion boards and indicates that the data transfer is completed. When the processor recognises /EXTDTACK during a read cycle data is latched and the bus cycle terminated. When /EXTDTACK is recognised during a write cycle, the bus cycle is terminated.
31	/BG	Bus Grant. This output indicates to all other potential bus master devices that the processor will release bus control at the end of the current bus cycle.
32	/BGACK	Bus Grant Acknowledge. This input indicates that some other device has become the bus master.
33	/BR	Bus Request. This input is wire ORed with all other devices that could become bus masters. It indicates to the processor that some other device desires to become the bus master.

34	/HALT	Halt. When this bidirectional line is driven by an external device, it will cause the processor to stop at the end of the current bus cycle. When the processor has been halted using this input, all control signals are inactive and all three-state lines are high impedance. When the processor has stopped executing instructions, such as in a double bus fault condition, the /HALT line is driven by the processor to indicate to external devices that the processor has stopped.
35	/RESET	Reset. This bidirectional line acts to reset the processor in response to an external reset signal. An internally generated reset (result of a RESET instruction) causes all external devices to be reset with the internal state of the processor unaffected. A total system reset results from external /HALT and /RESET signals being applied at the same time.
36	/VMA	Valid Memory Address.This output is used to indicate to M6800 devices that there is a valid address on the address bus and the processor is synchronised to enable (E) signal.
37	E	Enable. This signal is the standard enable signal common to all M6800 devices. The period for this output is ten MC68000 clock periods (six low; four high).
38	/VPA	Valid Peripheral Address. This signal indicates to expansion boards that the device or region currently being addressed is a M6800 family device.
39	/BERR	Bus Error. This input informs the processor that there is a problem with the cycle currently being executed.
40 41 42	/IPL2 /IPL1 /IPL0	Interrupt Control (/IPL2,/IPL1,/IPL0). These input pins indicate the encoded priority level of the device requesting an interrupt.
43 44 45	/FC2 /FC1 /FC0	These function code outputs indicate the state and cycle type currently been executed and are valid whenever /AS is active.
46-68	A23-A1	Address Bus (A23-A1). This 23 bit, unidirectional, three state bus is capable of addressing 8 megawords of data. It provides the address for bus operation during all cycles except interrupts.
69	/STARTUP	Startup. Power on jump signal for use with expansion cards.
70	/EXTVPA	External Valid Peripheral Address. This input indicates that the device or region addressed is a M6800 family device and that the data transfer should be synchronised with the enable (E) signal.
71 72 73 74	/EIRQ0 /EIRQ1 /EIRQ2 /EIRQ3	External Interrupt Request. These decoded inputs are used by expansion boards to generate interrupts. The level of the interrupt is determined by the setting of the 'INT LEVEL' on the main board. Only autovectored interrupts are supported on Rev B 1616 PCBs.

75,76	+5v	+5 volt power supply.
77,78	COM	System common ground.
79	CLK	Clock. 7.5 MHz system clock.
80	30M	30 MHz clock.

11
Appendix B: Switch and Link Settings

Switches

The SSDCC four-way switch is used to select

- The size of the EPROM used.
- Normal operation, or to test construction of the board.

These functions are assigned as follows:

Switch #	Function
SW1	EPROM type
SW2	Normal/Test mode
SW3	Not used
SW4	Not used.

Eprom settings

The SSDCC can be used with 28 pin EPROMS, in sizes ranging from 8k by 8, to 32k by 8. As the pinouts differ, provision is made to customise the EPROM socket to the EPROM being used by means of switch SW1-1.

SW1	off (open)	2764
SW1	off (open)	27128
SW1	on (closed)	27256

Normal mode

If, at reset time, SW2 is off (open), then the Z80 drops into its normal operating mode, ready for disk I/O under 1616/OS.

Test mode switch settings

The construction of the 1616 is aided by its inbuilt diagnostic and test programs. There are four tests, selected via the input switches. As the disk controller only has one available input bit (SW2), this is used to select between the inbuilt tests in the following manner:

Selecting the tests

If switch 2 is on (closed) at reset time, then diagnostic test 0 is executed. If the switch is now turned off, the Z80 drops into test 1. Turning the switch on again invokes test 2, etc. In this manner any test may be selected by closing the switch, resetting the Z80 and toggling the switch until the desired test is running. A listing of the Z80 diagnostic code is included as Appendix C. The tests within the ROM are as follows.

Test 0 - Incrementing loop

Loop around writing an incrementing value to the SCC, SCSI controller, WD1772, interrupt latch, data latch, command latch and disk select latch. Flashes the LED very quickly - too quickly in fact. The following signals should be pulsing low during execution of test 0:

/SCCCS	(U8/18)	
/SCSICS	(U8/19)	
/FDCCS	(U8/17)	
/ROMSEL		(U8/16)
/ZWRDAT		(U13/12)
/ZINT	(U13/13)	
/LATCHW		(U13/14)
/PORTR	(U13/11)	

Test 1 - Write to memory

Write one byte value into every location in memory ($0000 to $FFFF). Increment the byte value and repeat. Flashes LED. Signals which should strobe low during this test include:

A0-A15	(U4)	
/WR	(U4/22)	
/SRAM0SEL		(U1/20)

Test 2 - Write to $6000

Continuously write the value $5A to the RAM location $6000, read back and echo to the data latch. Flashes LED.

This test may be used to debug memory errors. The contents of the data latch may be read from the 1616 using the command ‘mdb FFFFC1 FFFFC1’

Test 3 - Test RAM

Copy the first 8k of ROM to RAM at $6000, then verify the copy. Flash the LED if all checks out. If an error is detected, halt tests.

Link (interrupt) settings

The set of links on the SSDCC allow the use of Z80 interrupts. They are normally not connected, when using floppy disks. If using Version 2.1 of the SCSI drive, pins 3 and 5 are linked. The pin have the following connections:

Pin 1	Z80 Interrupt pin 16
Pin 2	SD0 Byte from 1616 in SINTZ port has D0 bit 0 low.
Pin 3	Z80 Non-maskable interrupt pin 17
Pin 4	DRQ from FDC WD1772 pin 27
Pin 5	SCSIIRQ from NCR5380 pin 23
Pin 6	/SCCIRQ from Z8530 pin 5

12
Appendix C: EPROM startup code

```
;
; Applix SSDCC ROM assembler startup code
;
; Copyright (C) 1987 by Applix pty limited
;
; Programmer: Andrew K.P. Morton
;
; Look at the 'testmode' switch.  If on, loop around reading and writing
; everything.  If the switch is off, jump to the normal operational code.
; Starts at $0000

; I/O addresses

port        equ    0           ;Input port
latch       equ    8           ;Disk select latch
int         equ    10H         ;Write: interrupt 1616
clrint      equ    10H         ;Read: clear pending Z80 interrupt
data        equ    18H         ;Data I/O port
command     equ    1CH         ;Command I/O port

scsibase    equ    20H         ;SCSI controller
fdcbase     equ    40H
sccbase     equ    60H

;Define bits in the port
testmode    equ    8           ;Testmode bit: low = test mode
txrdy       equ    4           ;Ready for transmit
rxrdy       equ    2           ;Ready to receive
scommand    equ    1           ;Command from 1616

;Define bits in the latch
map         equ    80H
bank        equ    40H
side        equ    20H
fmo         equ    10H
led         equ    fmo
ds1         equ    8
ds0         equ    4
eject       equ    2
inuse       equ    1

diskchng    equ    2
rdy         equ    1

;Define SCC registers
sccbcont    equ    sccbase
sccbdata    equ    sccbase+1
sccacont    equ    sccbase+2
sccadata    equ    sccbase+3

;Bits in SCC control register
sccrxrdy    equ    1
scctxrdy    equ    4

;Define registers in the FDC
fdcstatus   equ    fdcbase
fdccommand  equ    fdcbase
fdctrack    equ    fdcbase+1
fdcsector   equ    fdcbase+2
fdcdata     equ    fdcbase+3

drq         equ    2

            .Z80
            .phase             0
```

```
start:      in     a,(port)
            and    testmode
            jp     nz,100H      ;Off to the C code if the switch is off

;Now loop, writing incrementing value to everything

loop1:      ld     a,b
            out    (sccbase),a  ;The SCC
            out    (scsibase),a       ;The SCSI controller
            out    (fdcbase),a  ;The FD controller
            out    (int),a      ;68k interrupt
            out    (data),a     ;Data latch (clear command bit)
            out    (command),a  ;Set command bit
            and    03fh         ;Keep BANK & MAP low
            out    (latch),a

            inc    b            ;New test pattern

            in     a,(port)
            and    testmode
            jr     z,loop1      ;Loop until switch is off

; Drop into the first memory test: write & read every location in memory

            ld     hl,0
loop2:      ld     (hl),b       ;Write B reg to memory
            ld     a,(hl)       ;Read back into A
            inc    hl           ;Do it for all memory
            ld     a,h
            or     l
            jr     nz,loop2

            inc    b            ;New value

            in     a,(latch)    ;Toggle LED
            xor    led
            out    (latch),a

            in     a,(port)
            and    testmode
            jr     nz,loop2     ;Loop until switch is turned on

; New memory test: write repetitively to memory, read back

            ld     hl,6000h
            ld     d,5ah
loop3:      ld     bc,0         ;Loop counter
loop4:      ld     (hl),d
            ld     a,(hl)       ;Read back from RAM
            out    (data),a     ;Echo to data latch
            dec    bc
            ld     a,b
            or     c
            jr     nz,loop4

            in     a,(latch)
            xor    led
            out    (latch),a

            in     a,(port)
            and    testmode
            jr     z,loop3      ;Loop until switch is turned off

;
; Memory pattern test: Copy ROM contents to RAM, verify
;
loop5:      ld     hl,0         ;ROM pointer
            ld     de,6000H     ;RAM pointer
            ld     bc,2000H     ;Counter
            ldir                ;Bang
```

```
            ld      hl,0
            ld      de,6000H
            ld      bc,2000H
check:      ld      a,(de)
            cp      (hl)        ;Check the memory again.
            jr      nz,err
            inc     hl
            inc     de
            dec     bc
            ld      a,b
            or      c
            jr      nz,check

            in      a,(latch)
            xor     led
            out     (latch),a
            jr      loop5

;Come here on error
err:        jp      err         ;Die off

            rept    256-$       ;Pad out to address 100h
            db      0
            endm

            end
```

Index

Table of Contents