APPLICATIONS OF MICROPROCESSOR TO INTERFACING TECHNOLOGY (SCSI)

Elizabeth Ann McKinnon

Declaration for the Degree of M.Sc.

I, Elizabeth Ann McKinnon hereby certify that this thesis has been composed by myself, that it is a record of my own work, and that it has not been accepted in partial or complete fulfilment of any other degree or professional qualification.

Signed: Date: 4.10.88

ABSTRACT

A study of the S-100 interface and the Small Computer System Interface (SCSI) was carried out. An interface board was built to communicate between the S-100 interface and a hard disk drive supporting SCSI. Two methods of communication were investigated. The first interface board used only standard TTL logic and the second interface board used TTL logic and a VLSI SCSI controller (NCR53C80).

ACKNOWLEDGEMENTS

I would like to acknowledge the support and guidance given to me by Dr Reg C G Killean throughout this project.

I would like to thank Professor Wilson Sibbett and the University of St Andrews for giving me the opportunity to undertake a Master of Science degree.

I would also like to thank the everyone in the department who gave me access to the computers and printers necessary to compile and print this thesis.

CONTENTS

CHAPTER 1

S-100 BUS ARCHITECTURE

The industry standard S-100 bus was originally designed at Stanford University for use with the Intel 8080 microprocessor. As a result, the bus signal definitions closely follow those of the 8080 system. The S-100 bus physically consists of 100 parallel lines (50 on each side of the connector), either etched onto a printed circuit board or, in its origination, hard wired. The printed circuit board with the S-100 edge connectors on it is referred to as a "motherboard". Table 1.1 shows the S-100 bus signals.

Table 1.1 : S-100 Bus Signals

1	+8V	26	pHLDA	51	+8V	76	pSYNC
2	+18V	27	- - - -	52	-18V	77	$\overline{\text{pWR}}$
3	XRDY	28	- - - -	53	- - - -	78	pDBIN
4	$\overline{\text{VI0}}$	29	A5	54	- - - -	79	A0
5	$\overline{\text{VI1}}$	30	A4	55	$\overline{\text{DMA0}}$	80	A1
6	$\overline{\text{VI2}}$	31	A3	56	$\overline{\text{DMA1}}$	81	A2
7	$\overline{\text{VI3}}$	32	A15	57	$\overline{\text{DMA2}}$	82	A6
8	$\overline{\text{VI4}}$	33	A12	58	$\overline{\text{sXTRQ}}$	83	A7
9	$\overline{\text{VI5}}$	34	A9	59	A19	84	A8
10	$\overline{\text{VI6}}$	35	DO1/DATA1	60	$\overline{\text{SIXTN}}$	85	A13
11	$\overline{\text{VI7}}$	36	DO0/DATA0	61	A20	86	A14
12	$\overline{\text{NMI}}$	37	A10	62	A21	87	A11
13	- - - -	38	DO4/DATA4	63	A22	88	DO2/DATA2
14	$\overline{\text{DMA3}}$	39	DO5/DATA5	64	A23	89	DO3/DATA3
15	A18	40	DO6/DATA6	65	$\overline{\text{MREQ}}$	90	DO7/DATA7
16	A16	41	DI2/DATA10	66	$\overline{\text{RFSH}}$	91	DI4/DATA12
17	A17	42	DI3/DATA11	67	- - - -	92	DI5/DATA13
18	$\overline{\text{SDSB}}$	43	DI7/DATA15	68	MWRT	93	DI6/DATA14
19	$\overline{\text{CDSB}}$	44	sM1	69	- - - -	94	DI1/DATA9
20	- - - -	45	sOUT	70	- - - -	95	DI0/DATA8
21	- - - -	46	sINP	71	- - - -	96	sINTA
22	$\overline{\text{ADSB}}$	47	sMEMR	72	pRDY	97	$\overline{\text{sWO}}$
23	$\overline{\text{DODSB}}$	48	sHLTA	73	$\overline{\text{pINT}}$	98	$\overline{\text{ERROR}}$
24	ϕ	49	CLOCK	74	$\overline{\text{pHOLD}}$	99	$\overline{\text{POC}}$
25	- - - -	50	GND	75	$\overline{\text{pRESET}}$	100	GND

The signals in Table 1.1 which have a bar across them, (ie. $\overline{\text{SIGNAL}}$), are negative logic and are asserted/active when they are 0V or deasserted/inactive when they are +5V. Signals with no bar, (ie. SIGNAL), are positive logic and are asserted/active when they are +5V or deasserted/inactive when they are 0V.

The S-100 bus allows interaction between a device operating as a bus master and a device operating as a bus slave. Briefly, the bus master, (whether temporary or permanent), initiates all bus cycles, the result of which may be to transfer data to and from the addressed bus slave. The bus slave monitors all bus cycles and, if addressed, will input or output data as required.

Signals not implemented on the S-100 bus using the Cromemco Single Card Computer (SCC) are:

$\overline{\text{SIXTN}}$ and $\overline{\text{sXTRQ}}$	These signals control 16 bit data transfer.
$\overline{\text{DMA0}}$-$\overline{\text{DMA3}}$	These signal are Direct Memory Access control signals which arbitrate between simultaneous requests, by temporary bus masters, for control of the bus.
$\overline{\text{VI0}}$-$\overline{\text{VI7}}$	These are the eight Vectored Interrupt lines, which control the eight levels of interrupt request priority.
$\overline{\text{ERROR}}$	This signal is used to indicate an error during the current bus cycle. On the Cromemco SCC S-100 bus this is used to indicate speed of operation. See Table 2.1 and chapter 2.2.

A brief description of the SCC implemented S-100 bus signals follows, they are also discussed fully in subsequent chapters. The signals present on the S-100 bus can be grouped into categories:

Power supply lines
Address bus
Data bus
Clock and control signals

1.1 POWER SUPPLY

Unregulated DC power supply voltages (+8 Volts, +18 Volts, -18 Volts and Ground) are supplied from a central power supply and must be regulated on each individual board.

This method of supply has certain advantages over a single, centrally regulated supply. Firstly, every card is individually protected from voltage overload, secondly, any heat

produced by voltage regulation is thermally distributed through a larger area and finally, the expansion of a computer system by the addition of cards is made easier, as voltage drop through loading of the power supply is no longer critical.

One disadvantage is that there is a danger of short circuiting the supply lines, (pins 1, 2, 51 and 52), together if a card is inadvertently moved while the voltage is on.

1.2 ADDRESS BUS

The S-100 bus address lines are used to select specific memory locations or specific input/output devices. There are two types of addressing, standard addressing using A0-A15 and extended addressing using A0-A23. The Cromemco Single Card Computer uses standard addressing.

	Memory Locations	Input/Output Devices
Standard Addressing	Up to 65,536 bytes (64 Kbytes) Using A0-A15	Up to 256 Devices Using A0-A7
Extended Addressing	Up to 16,777,216 bytes (16 MBytes) Using A0-A23	Up to 65,536 Devices Using A0-A15

1.3 DATA BUS

There are 16 lines used for data transfer on the S-100 bus. These can be two unidirectional, 8 bit, data buses (DO0-DO7 and DI0-DI7) or one bidirectional, 16 bit, data bus (DATA0-DATA15). The Cromemco Single Card Computer board uses the two, 8 bit bus type.

1.4 CLOCK AND CONTROL SIGNALS

CLOCK SIGNALS

There are two clock signals present on the S-100 bus system:

ϕ, (4MHz)	This is the system clock, which is generated by the permanent master, and controls the timing for all the bus cycles.
CLOCK, (2MHz)	The CLOCK line is not synchronous with any other bus signal and can be used by counters, timers etc.

CONTROL SIGNALS

The status signals identify the current bus cycle:

sMEMR	Memory read
sM1	Op-code fetch
sINP	Input
sOUT	Output
$\overline{\text{sWO}}$	Write cycle
sINTA	Interrupt acknowledge
sHLTA	Halt acknowledge

The control output signals determine the movement and timing of data during bus cycles:

pSYNC	Start of new bus cycle
pDBIN	Gate data onto data bus
$\overline{\text{pWR}}$	Write from data bus
pHLDA	Data and address bus in high impedance state

The following signals are used primarily with the Z80A microprocessor:

$\overline{\text{pINT}}$ and $\overline{\text{NMI}}$	Interrupt lines used to request servicing from the permanent bus master.
$\overline{\text{pHOLD}}$	This signal is used to request control of the bus from a permanent bus master.
$\overline{\text{MREQ}}$	This is a control signal which indicates that the address bus holds a valid address for a memory read/write operation.
$\overline{\text{RFSH}}$	This is a control signal which indicates that the lower seven bits of the address bus can be used as a refresh address for dynamic memories.

The following signals are primarily used on systems which also have front panel control (allowing the operator to interrupt, single step, perform read/write operations etc. on the bus master):

XRDY and pRDY	These signals allow bus slaves to synchronise with bus masters and request operations of the permanent master.
MWRT	This is a control signal used to indicate a memory write operation.

There are four lines which are used to tri-state the bus drivers, for example, during DMA operations:

$\overline{\text{SDSB}}$	Status disable
$\overline{\text{CDSB}}$	Control disable
$\overline{\text{ADSB}}$	Address disable
$\overline{\text{DODSB}}$	Data out disable

System reset signals:

$\overline{\text{pRESET}}$	This is the reset signal for all bus masters.
$\overline{\text{POC}}$	$\overline{\text{Power-On-Clear}}$, this is only asserted when the power is switched on and it also asserts $\overline{\text{pRESET}}$.

CHAPTER 2

CROMEMCO SINGLE CARD COMPUTER

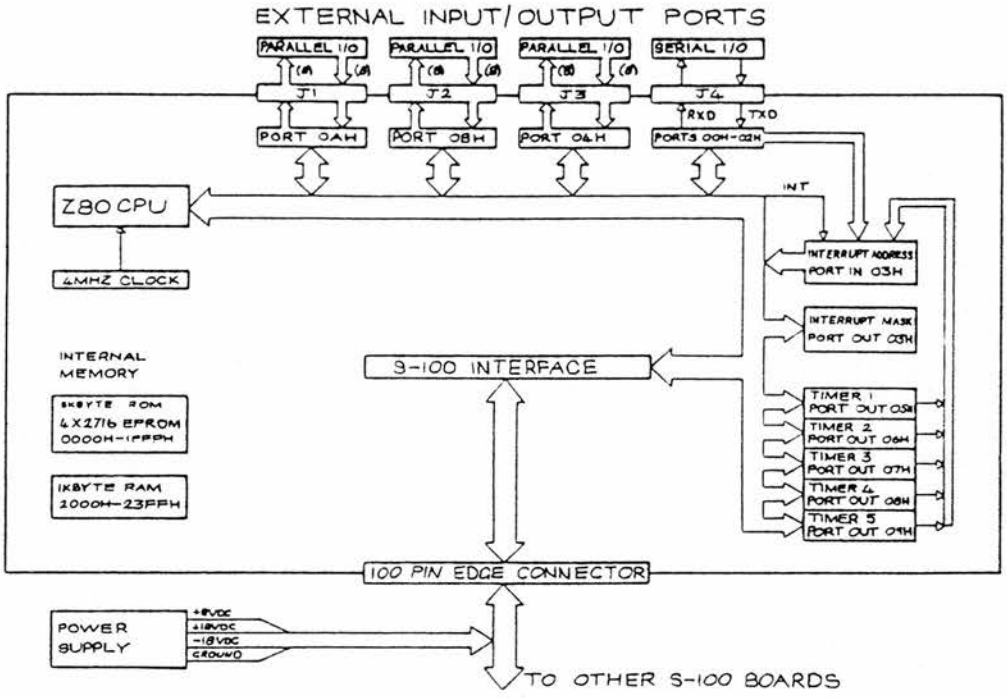
The Cromemco Single Card Computer (SCC) is a Zilog Z80A based S-100 interface board and can be used as a self contained development system. It allows parallel and serial input/output and provides on board sockets for eight kilobytes of user programmable EPROM memory. Figure 2.1 shows the Single Card Computer block diagram. Table 2.1, which is a subset of Table 1.1, shows S-100 bus connections of the Single Card Computer.

Table 2.1 : SCC S-100 Bus Signals

1	+8V	26	pHLDA	51	+8V	76	pSYNC
2	+18V	27	- - - -	52	-18V	77	$\overline{\text{pWR}}$
3	XRDY	28	- - - -	53	- - - -	78	pDBIN
4	- - - -	29	A5	54	- - - -	79	A0
5	- - - -	30	A4	55	- - - -	80	A1
6	- - - -	31	A3	56	- - - -	81	A2
7	- - - -	32	A15	57	- - - -	82	A6
8	- - - -	33	A12	58	- - - -	83	A7
9	- - - -	34	A9	59	- - - -	84	A8
10	- - - -	35	DO1	60	- - - -	85	A13
11	- - - -	36	DO0	61	- - - -	86	A14
12	$\overline{\text{NMI}}$	37	A10	62	- - - -	87	A11
13	- - - -	38	DO4	63	- - - -	88	DO2
14	- - - -	39	DO5	64	- - - -	89	DO3
15	- - - -	40	DO6	65	$\overline{\text{MREQ}}$	90	DO7
16	- - - -	41	DI2	66	$\overline{\text{RFSH}}$	91	DI4
17	- - - -	42	DI3	67	- - - -	92	DI5
18	$\overline{\text{SDSB}}$	43	DI7	68	MWRT	93	DI6
19	$\overline{\text{CDSB}}$	44	sM1	69	- - - -	94	DI1
20	- - - -	45	sOUT	70	- - - -	95	DI0
21	- - - -	46	sINP	71	- - - -	96	sINTA
22	$\overline{\text{ADSB}}$	47	sMEMR	72	pRDY	97	$\overline{\text{sWO}}$
23	$\overline{\text{DODSB}}$	48	sHLTA	73	$\overline{\text{pINT}}$	98	4MHz
24	ϕ	49	CLOCK	74	$\overline{\text{pHOLD}}$	99	$\overline{\text{POC}}$
25	- - - -	50	GND	75	$\overline{\text{pRESET}}$	100	GND

As mentioned in Chapter 1, the S-100 bus signals were designed for use with the Intel 8080 microprocessor. There is a substantial difference between the 8080 and Z80A control lines, but the SCC board is designed to interpret the important S-100 "8080-like" bus functions.

Figure 2.1 : Single Card Computer Block Diagram



The SCC can be grouped into eight parts:

Power supply
Clocks
Z80A Central Processing Unit
System reset
Memory
Input/output ports
Interrupts
Timers

2.1 POWER SUPPLY

The SCC board requires three unregulated voltages:

+8VDC @ 1.75A
+18VDC @ 100mA
-18VDC @ 50mA

These voltages are regulated on the SCC board to +5VDC, +12VDC and -5VDC respectively. The +5VDC provides power to the TTL logic and the 5501 UART. The +12VDC and -5VDC voltages provide power to the UART.

2.2 CLOCKS

There is an 8MHz crystal on the SCC board which is used to control the internal SCC timing functions. This frequency is then halved to provide the 4MHz to the Z80A CPU clock input, and to the S-100 ϕ bus line. Other SCC devices use the complement of the 4MHz signal as a reference. The 4MHz frequency is halved again to supply a 2MHz signal to the S-100 CLOCK bus line, and to both of the 5501 UART clock inputs (ϕ_1 and ϕ_2).

The S-100 bus 4MHz line (pin 98) is a Cromemco Z80A system function. This signal is used to show whether the system is running at 4MHz (logic 1) or 2MHz (logic 0). This line is pulled high on the SCC board.

2.3 Z80A CPU

The Z80A Central Processing Unit allows direct addressing of up to 64 Kilobytes of memory, 256 input ports and 256 output ports. The Z80A instruction set contains 158 instructions, including the 78 instructions of the 8080A.

Z80A REGISTERS

The Z80A contains eighteen 8 bit registers, four 16 bit registers and two interrupt status flip-flops. These can be divided into groups as shown in the following tables.

Main 8 Bit Register Set		**Alternate 8 Bit Register Set**	
A (Accumulator)	F (Flag)	A' (Accumulator)	F' (Flag)
B (General)	C (General)	B' (General)	C' (General)
D (General)	E (General)	D' (General)	E' (General)
H (General)	L (General)	H' (General)	L' (General)

A, A'	The accumulator holds the results of logical and arithmetic operations.
F,F'	The flag register indicates the conditions of the last operation (eg. if the result was zero).
B,B',C,C',D,D', E,E',H,H',L,L'	These remaining general purpose registers can either be used as twelve 8 bit registers, or as six 16 bit register pairs (BC, DE etc.).

Special Purpose 16 Bit Registers

IX (Index Register) SP (Stack Pointer)	IY (Index Register) PC (Program Counter)

IX, IY The index registers are used to hold the base address when using indexed addressing mode.

SP The stack pointer register holds the address of the current top of the stack. The stack is a temporary data storage area.

PC The program counter is the register which holds the address of the instruction which is being fetched from memory. After the contents of the counter are transferred to the address bus, the program counter is automatically incremented. If the instruction being executed causes a program jump, the new address is written over the address in the counter register.

Special Purpose 8 Bit Registers

I (Interrupt Register)	R (Refresh Register)

I The interrupt register is used when the Z80A CPU is operated in a mode which will respond to an interrupt with an indirect call to any memory location. Servicing interrupt requests in this way means that the time taken to access the interrupt routine can be minimised, as the routine can be stored anywhere in memory.

R The refresh register is used to generate the memory refresh address when dynamic memories are being used in the system.

Interrupt Status Flip-Flops

IFF1	IFF2

Interrupt Mode Flip-Flops

IMFa	IMFb

IFF1, IFF2
IMFa, IMFb The interrupt status flip-flops and the interrupt mode flip-flops are the registers which help to ascertain the current interrupt mode of the Z80A CPU, (Mode 0, Mode 1 or Mode 2).

Z80A ADDRESS BUS AND DATA BUS

The Z80A has a 16 bit, (tri-state output), address bus and an 8 bit, (tri-state input/output), data bus. The lines of these buses can be in three different states, logic 1, logic 0 or in a high impedance state. When the buses of the Z80A CPU are in the high impedance state they appear to be disconnected from the other devices which utilise the bus. This allows the other logic circuitry to use the bus without any confusion of the signals.

The S-100 address bus, A0-A15, is driven from the Z80A address bus, A0-A15. The two buses are connected together, via two tri-state drivers. The S-100 address bus drivers can be put into a high impedance state during direct memory access operations by asserting the S-100 bus signal $\overline{\text{ADSB}}$. The Z80A address bus controls the SCC memory and input/output ports, as described in Chapters 2.6 and 2.7.

The S-100 data out bus, DO0-DO7, is driven from the Z80A data bus, D0-D7, via a tri-state driver. Similarly, the Z80A data bus receives data from the S-100 data in bus, DI0-DI7, via a tri-state buffer. The data output driver can be disabled (ie. tri-stated) during direct memory access operations by asserting the S-100 bus signal $\overline{\text{DODSB}}$.

Z80A CONTROL INPUT SIGNALS

$\overline{\text{NMI}}$ (non-maskable interrupt) is a buffered version of the S-100 bus $\overline{\text{NMI}}$ signal. Similarly, $\overline{\text{BUSRQ}}$ (bus request) is the buffered version of S-100 bus signal $\overline{\text{pHOLD}}$. The Z80A $\overline{\text{WAIT}}$ (wait) input is used to indicate to the CPU that the addressed memory or input/output devices are not ready for a data transfer. The CPU stays in the wait state as long as this signal is asserted. $\overline{\text{INT}}$ (interrupt request) is generated by input/output devices. The CPU will service the interrupt at the end of the current instruction if the internal software controlled interrupt enable flip-flop (IFF) is enabled. $\overline{\text{RESET}}$ is used to initialise the CPU, (see Z80A Reset section for more information).

Z80A CONTROL OUTPUT SIGNALS

The Z80A control output lines are logically combined to generate S-100 bus control signals. S-100 signals sM1, pHLDA and sHLTA are the logical inversion of $\overline{\text{MI}}$ (machine cycle one), $\overline{\text{BUSAK}}$ (bus acknowledge) and $\overline{\text{HALT}}$ (halt state) respectively. S-100 $\overline{\text{MREQ}}$ is a buffered version of the Z80A $\overline{\text{MREQ}}$ (memory request) signal. Similarly, S-100 $\overline{\text{RFSH}}$ is the buffered version of Z80A signal $\overline{\text{RFSH}}$ (refresh). S-100 pSYNC which signals the beginning of a new

machine cycle is clocked high by a falling edge of either $\overline{MREQ}$ or $\overline{IORQ}$ and is clocked back low by the next rising edge of phi. The remaining S-100 control lines are derived from the Z80A outputs as shown:

$$\begin{aligned} sINP &= \overline{IORQ}+\overline{RD} \\ sOUT &= \overline{IORQ}+\overline{WR} \\ \overline{sWO} &= sOUT+(\overline{RFSH}\cdot\overline{RD}\cdot MREQ) \\ sMEMR &= \overline{MREQ}+\overline{RD} \\ sINTA &= (\text{External Priority})\cdot(\overline{M1}+\overline{IORQ}) \\ pDBIN &= RD+sINTA \\ \overline{pWR} &= \overline{WR}(\text{Delayed}) \end{aligned}$$

$\overline{pWR}$ is delayed so sOUT has time to stabilise before the pWR low pulse is used as a data strobe. All of these lines can be disabled during DMA by asserting S-100 bus lines $\overline{SDSB}$ and $\overline{CDSB}$.

Z80A RESET

The Z80A CPU is reset when the $\overline{RESET}$ input is asserted (0V) for at least three clock cycles (600nS). This resets the interrupt enable flip-flop, clears the program counter and registers I and R and sets the interrupt status flip-flops to Mode 0. During the reset time the address bus and the data bus go into a high impedance state and all control output signals go to the inactive state. When the Z80A is reset the program counter is set to 0000H, and program execution is started from memory location 0000H.

2.4 SYSTEM RESET

On power up the SCC circuitry automatically asserts the S-100 $\overline{POC}$ (Power-On-Clear) line, this also asserts the S-100 $\overline{pRESET}$ line. This resets the Z80A, by asserting $\overline{RESET}$, and puts the SCC into a known state with the on board memory enabled. The same resetting of the SCC and Z80A occurs when the S-100 $\overline{pRESET}$ signal is asserted by an external source. Any other device can use the S-100 $\overline{pRESET}$ bus line to reset to a known state.

2.5 SCC MEMORY

The SCC has space for 8 kilobytes of 2716 EPROM memory (in sockets ROM0, ROM1, ROM2 and ROM3) and 1 kilobyte of 4045 static RAM. The memory address map is shown in the following table.

SCC Memory Map

ROM0 (2K) ROM1 (2K) ROM2 (2K) ROM3 (2K)	0000H - 07FFH 8000H - 0FFFH 1000H - 17FFH 1800H - 1FFFH
RAM	2000H - 23FFH
External Memory	2400H - FFFFH

Additional memory can be installed in the SCC controlled system by plugging ROM or RAM boards into the S-100 bus.

ROM0 contains the assembled SCSI INTERFACE program (see Appendices A and B) which is automatically entered when the SCC is powered up or when the S-100 $\overline{\text{pRESET}}$ line is pulled low, causing the Z80A to reset the program counter to 0000H and start program execution.

MEMORY SELECTION

The memory select signals are controlled by a standard 4 line to 10 line (BCD to DECIMAL) decoder chip as shown by the table below.

ROM Memory Select Logic

INPUTS				OUTPUTS					
A A11	B A12	C A13	D	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	DEVICE SELECTED
0	0	0	0	0	1	1	1	1	ROM0
1	0	0	0	1	0	1	1	1	ROM1
0	1	0	0	1	1	0	1	1	ROM2
1	1	0	0	1	1	1	0	1	ROM3

RAM Memory Select Logic

INPUTS				OUTPUTS						
A A11	B A12	C A13	D	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{EN}$	DEVICE SELECTED
0	0	1	0	1	1	1	1	0	0	RAM

In both of the above memory select logic tables, decoder inputs A, B and C are A11, A12 and A13 respectively. Decoder input D is used as an active low enable signal. D is low when the SCC memory disable option is inactive, $\overline{MI}$ or $\overline{MREQ}$ is active (logic 0), $\overline{IORQ}$ is not active (logic 1), A14 and A15 are low.

In the RAM select logic table, $\overline{EN}$ is the logical OR of A10 and the decoder output $\overline{4}$.

ROM0-ROM3 are selected with the appropriate address on A11-A13 and decoder input D active (logic 0). The SCC RAM is selected with the appropriate address on A10-A13 and decoder input D active (logic 0).

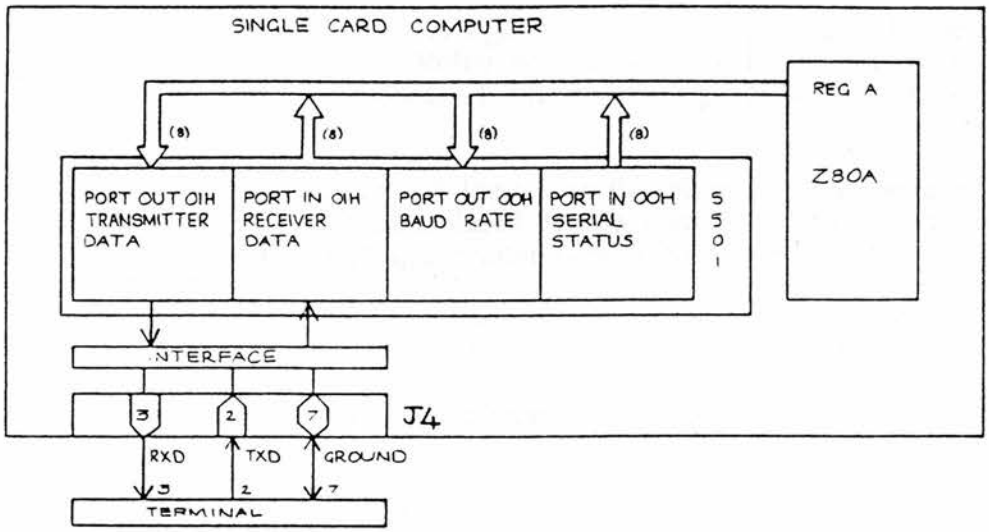
2.6 SCC INPUT/OUTPUT PORTS

The SCC has three 8 bit parallel input/output ports, with separate input data lines and output data lines, on connectors J1, J2 and J3 (addressed at 0AH, 0BH and 04H respectively). Port 04H is an integral part of the 5501 UART and ports 0AH and 0BH are driven from the Z80A data bus via latches. The parallel input/output ports are not used in this project and are only mentioned briefly. The serial input/output port found on connector J4, addressed at 00H-02H, is used to connect a VDU terminal to the SCC. The VDU terminal is used to input data into the SCC and display the output from the SCC.

Data is received from and transmitted to the terminal, (via connector J4), from the Z80A (register A). The parallel CPU data is converted to the serial terminal data, and vice versa, by the 5501 Universal Asynchronous Receiver Transmitter (UART). The UART also supplies the serial status data and the serial control data.

The SCC TTL signals (0V or 5V) are converted to the terminal RS232 (±12V) signals, and from RS232 to TTL, by the interface on the SCC. See figure 2.2 which shows the serial signal paths of the SCC and the terminal connection.

Figure 2.2 : Single Card Computer Serial Signal Paths



The input/output select logic, for the ports and the UART, is controlled by a Cromemco custom PROM as shown by the tables below. The inputs to the PROM are, from the address bus, A0-A3 and, from the Z80A CPU, $\overline{WR}$. The $\overline{CS}$ input is low when A4-A7 are low, $\overline{IORQ}$ is active, and $\overline{MI}$ is inactive.

There are eight output lines from the PROM, Y1-Y8. Y1 and Y2 are used as an active high strobe for 0BH and 0AH respectively. Y3 is an active low enable for the 5501 UART. Y4-Y7 select a 5501 UART function by driving its A0-A3 address lines as shown in the table below.

Receiver Data Port

D7	D6	D5	D4	D3	D2	D1	D0
DATA BYTE FROM TERMINAL SHIFT ---->							

When an IN A,(01H) instruction is executed, an assembled data byte (D0 - D7) is read into the Z80A (register A), from the the UART receiver data port (ie. the terminal). The Z80A knows when there is valid data available by testing the serial status port, bit RDA.

All bits are used in relation to this project.

Transmitter Data Port

D7	D6	D5	D4	D3	D2	D1	D0
DATA BYTE TO TERMINAL SHIFT ---->							

Data is written from the Z80A (register A) into the transmitter data port (D0 - D7) when the Z80A executes an OUT (01H),A. The Z80A is programmed to recognise when to send the data byte, by testing the serial status port, bit TBE.

All bits are used in relation to this project.

Command Register Port

D7	D6	D5	D4	D3	D2	D1	D0
NOT USED	NOT USED	TB5	HBD	INE	RS7	BRK	RES

The UART is configured when an OUT (02H),A instruction is executed (register A containing the data). Only four of the command register port bits have any effect on the serial port - HBD, INE, BRK and RES.

HBD (D4) High Baud. Setting this bit to logic 1 multiplies the UART clock frequency by eight, which causes the serial port baud rate to multiply by eight (eg. 1200 baud becomes 9600 baud).

INE (D3) INTA Enable. When this bit is reset to logic 0 the UART ignores the INTA (interrupt acknowledge) cycles. The UART will recognise the INTA cycles and service the interrupt when this bit is set to logic 1.

BRK (D1) Break. When this is reset to logic 0 the serial transmitter operates normally however, when set to logic 1 the serial transmitter output is latched in the low (spacing) state.

RES (D0) UART Reset. When this bit is set to logic 1 the serial status port bit, RDA, is reset to logic 0 and the serial receiver goes into the search for a new character mode (not affecting the contents of the receiver data port). Also, the serial status register port, bit TBE, is set to logic 1 and the serial transmitter output goes high (marking). Once this happens RES will reset to logic 0 and the UART will be ready to perform serial input/output operations.

Only bits D4 (HBD) and D0 (RES) are used in relation to this project.

Interrupt Address Port

D7	D6	D5	D4	D3	D2	D1	D0
1	1	L_2	L_1	L_0	1	1	1

When the UART interrupt address port is read by execution of an IN A,(03H) instruction, the Z80A (register A) contains the coding of the source of the interrupt request (eg. if the contents are 07EH then is RDA requesting service).

This port is not implemented in relation to this project.

Interrupt Mask Port

D7	D6	D5	D4	D3	D2	D1	D0
TIMER 5	TIMER 4	TBE	RDA	TIMER 3	INT	TIMER 2	TIMER 1

The UART interrupt mask port is configured an OUT (03H),A instruction is executed by the Z80A (register A containing the data).

When the bit is reset to logic 0, the source cannot issue an interrupt request. When the bit is set to logic 1 the source can issue an interrupt request. TBE (D5) and RDA (D4) cannot be disabled in this way.

This port is not implemented in relation to this project.

2.7 SCC INTERRUPTS

The SCC has ten interrupt sources which can be grouped as follows:

S-100 Interrupts	$\overline{\text{NMI}}$ is connected directly to the Z80A $\overline{\text{NMI}}$ input and is the only interrupt which cannot be software disabled. $\overline{\text{pINT}}$ is channelled into the Z80A $\overline{\text{INT}}$ input and is only enabled after the Z80A executes an EI instruction.
UART Interrupts	TIMER1-TIMER5, $\overline{\text{INT}}$, RDA and TBE are all channelled into the Z80A $\overline{\text{INT}}$ input. For any of these interrupts to be enabled the correct interrupt mask port code must be present in addition to the interrupts being enabled by execution of the Z80A EI instruction.

The interrupts are not utilised in relation to this project.

2.8 SCC TIMERS

The UART contains five timers, TIMER1-TIMER5, which can be enabled and disabled by the UART interrupt mask port. There is a port assigned to each of the timers (05H - 09H) which can be loaded with a delay count (maximum value of 0FFH). This delay count is then decremented either every 64 micro-seconds (if command register port HBD is reset) or every 8 micro-seconds (if HBD is set).

If the interrupts are disabled when the delay count reaches zero the UART serial status port, bit IPG, is set to logic 1. The Z80A can determine the timer which is requesting service and reset IPG to logic 0 by reading the UART interrupt address port. This method does not enter an INTA cycle as only the SCC input ports are being used, not the Z80A $\overline{\text{INT}}$ input.

If the interrupts are enabled and UART command register port, bit INE, is set to logic 1 when the delay count reaches zero, the Z80A is interrupted. The Z80A will then execute the timer unique RST instruction in response to the Z80A INTA cycle.

If the interrupts are enabled and UART command register port, bit INE, is reset to logic 0 when the delay count reaches zero, the Z80A is interrupted. The Z80A will initiate an interrupt service routine which will read the interrupt address port to identify the source.

The timers are not utilised in relation to this project.

2.9 UART CONFIGURATION

Before the serial input/output port is used the UART must be configured by writing into register A and issuing an OUT (02H),A instruction. The correct baud rate must also be set by writing into register A and issuing an OUT (00H),A instruction.

The following shows the part of the main program which is used to configure the UART. Refer to the UART ports descriptions for full details on the ports mentioned in the description of the program.

BAUD RATE TABLE

This shows the table of bytes for the different baud rates. When the program is assembled and written into the EPROM, the binary number are stored as hex bytes in consecutive locations with a label (baudrs).

```
baudrs  db   10010000b      ; 2400 / 19,200
        db   11000000b      ; 9600 / 76,800
        db   10100000b      ; 4800 / 38,400
        db   10010000b      ; 2400 / 19,200
        db   10001000b      ; 1200 / 9,600
        db   10000100b      ;  300 / 2,400
        db   10000010b      ;  150 / 1,200
        db   00000001b      ;  110 / 880
```

The terminal can be operating at any one of a number of baud rates, as shown by the table above. The UART command register port, bit HBD determines which range of baud rates are valid. When HBD is set to 0, the range is 110 baud to 9600 baud. When HBD is set to 1, the range is 880 baud to 76,800 baud. The first value in the table, 2400/19,200, is the normal setting for the terminal, if this is not correct, the other values will be tried.

When the boards are powered up, the user must press the carriage return key (ASCII value - 0DH) several times to allow the baud rate of the terminal and the baud rate of the SCC to be matched. The program does this as shown below.

PROGRAM LISTING

Main Program

```
Line 1                  ld    hl,baudrs
Line 2                  ld    c,00H
Line 3                  ld    a,11h
Line 4        baud1     out   (02H),a
Line 5                  outi
Line 6                  call  gbyte
Line 7                  call  gbyte
Line 8                  and   7fh
Line 9                  cp    0dh
Line 10                 ld    a,1
Line 11                 jr    nz,baud1
```

Gbyte Subroutine

```
Line S1       gbyte     in    a,(00H)
Line S2                 bit   6,a
Line S3                 jr    z,gbyte
Line S4                 in    a,(01H)
Line S5                 ret
```

PROGRAM DESCRIPTION

Line 1 to Line 3 sets up the contents of the HL, C and A registers.

Register pair HL is pointing to the first baud rate in the table.

Register C contains the address of the UART baud rate port.

Register A contains reset bit and high baud for sending to the UART command register port.

Line 4 outputs the contents of register A, (00010001 in binary notation), to the UART command register port (02H). This configures and resets the UART. Note that this line has a label (baud1). If the correct baud rate is not found the first time, the program will jump back to here and reset the UART. The program will continue looping round to Line 4 until the correct baud rate is found. The UART command register will contain the following.

D7	D6	D5	D4	D3	D2	D1	D0
NOT USED	NOT USED	TB5	HBD	INE	RS7	BRK	RES
0	0	0	1	0	0	0	1

Four of these bits have an effect on the serial input/output port: HBD, INE, BRK and RES.

HBD=1, serial port baud rate to multiplies by eight (eg. 2400 baud becomes 19,200 baud).

INE=0, interrupt acknowledge (INTA) cycles ignored by UART.

BRK=0, serial transmitter operates normally.

RES=1, reset UART.

Line 5 outputs the contents of the memory location addressed by HL (first baud rate), to the address contained in register C, (UART baud rate port) and then increments HL. HL is then pointing to the next baud rate in the table. For the first baud rate in the table, the UART baud rate port contains the following.

D7	D6	D5	D4	D3	D2	D1	D0
STOP BITS	9600/ 76,800	4800/ 38,400	2400/ 19,200	1200/ 9,600	300/ 2,400	150/ 1,200	110/ 880
1	0	0	1	0	0	0	0

This sets the baud rate port to a baud rate of 19,200, (HBD=1 UART command register port), with 1 stop bits. For subsequent baud rates, HBD=0 in the UART command register port, therefore the lower values are active.

Lines 6 and 7 call the gbyte subroutine. This subroutine is called twice to stop any phase match error. The first call of the gbyte subroutine is being ignored in this description as explained above. The second call of the gbyte subroutine performs a read from the terminal, via the SCC serial port, using the carriage return characters which the user is sending to determine the baud rate setting of the terminal.

Line S1 of the gbyte subroutine inputs a data byte from the UART serial status port, (00H), into register A. Note that this line has a label (gbyte). This label is not only used for the call instruction from the main program, it is also used by the subroutine itself.

Line S2 tests bit 6 of register A, UART serial status port bit RDA. A logic 0 in this bit, (no data byte available), sets the zero flag. A logic 1 in this bit, (data byte available), does not set the zero flag. The UART serial status port is shown below.

D7	D6	D5	D4	D3	D2	D1	D0
TBE	RDA	IPG	TBE	RDA	SRV	ORE	FME
X	0/1	X	X	X	X	X	X

Line S3 causes the program to jump back to Line S1 if there is no data byte available. The program will continue looping in this manner until there is a data byte available.

When there is a data byte available, Line S4 reads it from the receiver data port, (address 01H), and stores it in register A. Line S5 returns to the main program, (Line 8).

Line 8 performs a logical AND between register A and 7FH, this masks off bit D7 of the receiver data port because only bits D0 to D6 are valid.

Line 9 compares the contents of register A, (receiver data port), with 0DH, (carriage return character). If the data byte in register A is 0DH the zero flag is set, otherwise the zero flag is not set. The setting of the zero flag signifies that the baud rate of the terminal and the baud rate of the SCC are matched.

Line 10 loads register A with 1, (00000001 in binary notation), which is used to reset the UART command register port if a 0DH is not recognised, (incorrect baud rate).

Line 11 tests the zero flag and depending on its value, either continues with the program or jumps back to Line 4. If the baud rates match, the zero flag will be set and the main program will continue. If the baud rates do not match, the zero flag will not be set and there will be a jump back to Line 4, (reset UART command register port), where the next value in baud rate table will be tried. The program continues in this manner until the baud rate of the terminal and the baud rate of the SCC are matched.

CHAPTER 3

THE SMALL COMPUTER SYSTEM INTERFACE (SCSI)

In this chapter SCSI is discussed with reference to the RODIME RO652 hard disk drive which was used in this project. SCSI supports both block and character transfer, the RO652 uses the block method. For more detailed information on SCSI, the American National Standards Institute document X3T9 should be referred to.

The theory of bus structure is based on a system originally developed in the 1960's and 1970's for use in mainframe computers. This concept was improved on and was introduced as the Shugart Associated Standard Interface (SASI). SASI was then offered to the American National Standards Institute (ANSI) as the basis for a new industry standard. The Small Computer System Interface (SCSI) was the name chosen by ANSI and the X3T9 specification was issued in the early 1980's.

The SCSI interface is an eight bit, eight port intelligent bus. RO652 bus transfers are asynchronous and follow a defined $\overline{\text{REQ}}/\overline{\text{ACK}}$ handshake protocol. The input/output bus structure enables intelligent peripherals to communicate with multiple hosts over a single bus and allows these peripherals to be used by one or more host computers attached to the same bus. Communication on the SCSI bus uses the initiator/target framework, where the initiator is the device which requests that an operation be performed by another device and the target is the device which performs an operation when requested to do so by an initiator. Up to eight SCSI devices can be supported in any combination on the bus. There are three principal SCSI system configurations: single initiator/single target (figure 3.1); single initiator/multi target (figure 3.2) and multi initiator/multi target (figure 3.3). The main advantage of the SCSI bus system is that several tasks can be interleaved at the same time (figure 3.4).

Figures 3.1-3.3 : SCSI System Configurations

FIGURE 3.1

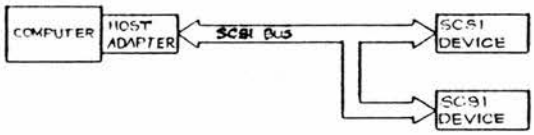
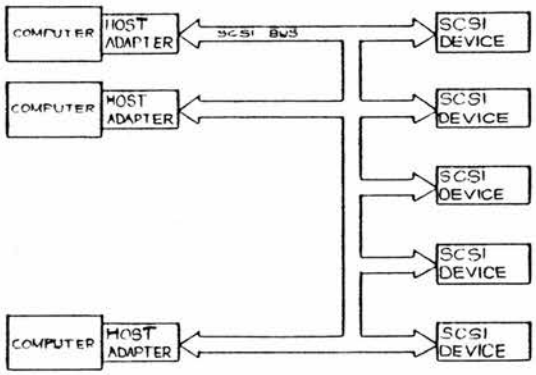
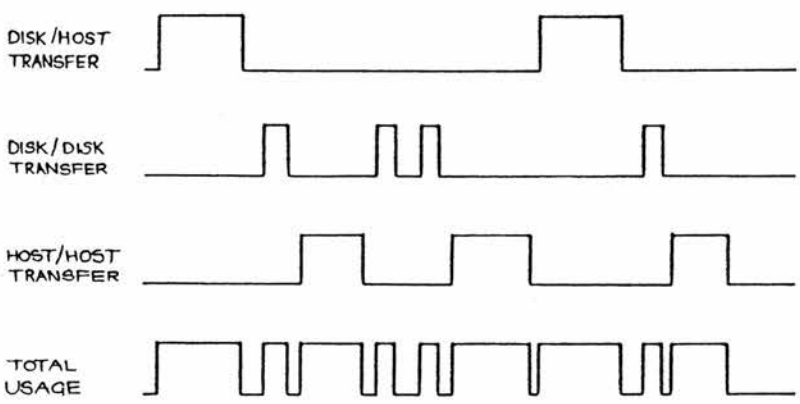


FIGURE 3.2



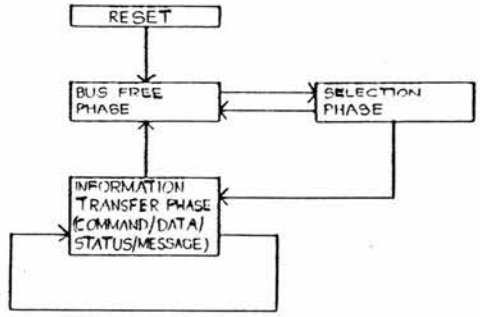
SCSI can also operate in a synchronous data transfer mode, which is faster than the asynchronous mode, due to reduced handshaking. Synchronous data bus transfers can only be used if previously agreed by both the initiator and the target, using a Synchronous Data Transfer Request command.

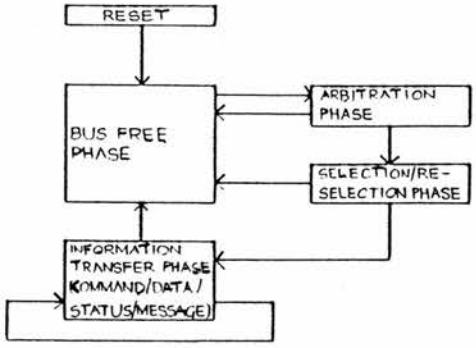
Figure 3.4 : SCSI Task Interleaving



Arbitration allows a number of peripheral devices to be attached to the bus and makes it possible for one SCSI device to gain control of the bus and assume the role of an initiator or a target. To gain control of the bus, a device waits for a bus free state and asserts its own SCSI ID. After waiting for at least an arbitration delay the SCSI device examines the data bus, if there is a higher SCSI ID present the device has lost control, if no higher SCSI ID is present the device has won control of the bus. Figure 3.5 and figure 3.6 show the sequence of bus phases for arbitrating and non-arbitrating devices.

Figures 3.5-3.6 : SCSI Bus Phases



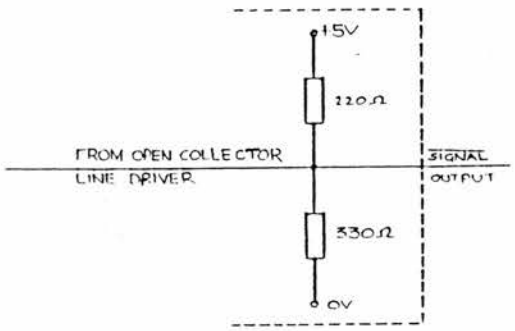


3.1 HARDWARE

SCSI devices are daisy chained together using a 50 way cable. The RO652 hard disk drive uses the single ended driver alternative, which gives a maximum cable length of six metres. The differential pair driver alternative has a maximum cable length of 25 metres and is normally used where the SCSI device has to be some distance from the host. Both ends of the bus cable must be terminated correctly, with 220 ohms to +5V and 330 ohms to ground,

on all of the signal lines. Figure 3.7 shows an example of correct signal termination. Single ended drivers and differential pair drivers cannot be used on the same bus. All signals on the SCSI bus are active low and use open collector drivers. SCSI bus signals use negative logic, this means that a false/deasserted signal on the SCSI bus is +5 volts and a true/asserted signal is 0 volts.

Figure 3.7 : SCSI Bus Termination



The SCSI interface is a bidirectional bus interface which transfers data asynchronously. The fifty way SCSI bus consists of eighteen signal lines: nine control signals used to co-ordinate data transfer between the host system and the disk drive and nine data bus signals used as an eight-bit bidirectional data bus with parity. The following table shows the SCSI single ended option pin numbers and names for the RO652.

SCSI BUS SIGNALS

1	GND	27	GND	2	$\overline{DB0}$	28	NC
3	GND	29	GND	4	$\overline{DB1}$	30	NC
5	GND	31	GND	6	$\overline{DB2}$	32	$\overline{ATN}$
7	GND	33	GND	8	$\overline{DB3}$	34	NC
9	GND	35	GND	10	$\overline{DB4}$	36	$\overline{BSY}$
11	GND	37	GND	12	$\overline{DB5}$	38	$\overline{ACK}$
13	GND	39	GND	14	$\overline{DB6}$	40	$\overline{RST}$
15	GND	41	GND	16	$\overline{DB7}$	42	$\overline{MSG}$
17	GND	43	GND	18	$\overline{DBP}$	44	$\overline{SEL}$
19	GND	45	GND	20	NC	46	$\overline{C/D}$
21	GND	47	GND	22	NC	48	$\overline{REQ}$
23	GND	49	GND	24	NC	50	$\overline{I/O}$
25	NC			26	TERMPWR		

TERMPWR	Terminator power is a +4.0VDC - +5.25VDC signal which is provided on the SCSI bus by the RO652.
GND	GROUND
NC	No Connection

Data Bus Lines

$\overline{DB0}$-$\overline{DB7}$,$\overline{DBP}$ These nine lines, data bit 0-data bit 7, and a data parity bit form the data bus. Data bit 7 is the most significant bit and has the highest priority during an arbitration phase. Data bit 0 is the least significant bit and has the lowest priority during an arbitration phase. Parity is odd in the SCSI system and can either be enabled on all devices connected to the SCSI bus or disabled on all devices. The RO652 does not support parity.

Control Lines

$\overline{ATN}$ Attention indicates to the target that the initiator has a message to send.

$\overline{BSY}$ Busy indicates that the bus is in use.

$\overline{ACK}$ Acknowledge is used with $\overline{REQ}$ to indicate an acknowledgement of a bus transfer handshake.

$\overline{RST}$ Reset is used to clear all activity on the bus.

$\overline{MSG}$ Message indicates that the bus is in a message phase.

$\overline{SEL}$ Select is used during device-selection phase.

$\overline{C}$/D Control/Data indicates whether control or data information is on the data bus. When this signal is true control information is on the bus.

$\overline{REQ}$ Request is used with $\overline{ACK}$ to request a bus transfer handshake.

I/O Input/Output indicates direction of the data flow on data bus, with respect to the initiator. When this signal is true, data is being inputted to the initiator.

3.2 SCSI BUS PHASES

The SCSI bus can be in one of eight phases:

Bus Free Phase
Arbitration Phase
Selection Phase
Reselection Phase
Command Phase
Data Exchange Phase
Status Phase
Message Phase

Phases 5, 6, 7 and 8 are collectively known as the Information Transfer Phase.

The following table shows which of the SCSI bus signals the initiator or the target is permitted to source during the specific phase. The SCSI bus signal RST may be sourced by both initiator and target during any phase. No attempt is made here to show if the source is driving the SCSI bus signal asserted, negated, or is passive. All SCSI device drivers that are not active sources are in the passive state.

Signal Sources

Bus Phase	BSY	SEL	C/D, I/O MSG, REQ	ACK, ATN	DB(7-0,P)
Bus Free	None	None	None	None	None
Arbitration	All	Winner	None	None	SCSI ID
Selection	I+T	Initiator	None	Initiator	Initiator
Reselection	I+T	Target	Target	Initiator	Target
Command	Target	None	Target	Initiator	Initiator
Data In	Target	None	Target	Initiator	Target
Data Out	Target	None	Target	Initiator	Initiator
Status	Target	None	Target	Initiator	Target
Message In	Target	None	Target	Initiator	Target
Message Out	Target	None	Target	Initiator	Initiator

All — The signal is driven by all SCSI devices that are actively arbitrating.

SCSI ID — A unique data bit, the SCSI ID, is driven by each SCSI device that is actively arbitrating; the other 7 data bits are deasserted (ie. not driven) by that device. DBP may be undriven or driven true but can never be driven false during this phase.

I+T — The signal is driven by the initiator, the target or both as specified in the Selection and Reselection Phases.

Initiator — If the signal is driven, it is only driven by the active initiator.

None — The signal is deasserted; that is not be driven by any SCSI device. The bias circuitry of the bus terminators pulls the signal to a false state.

Winner — The signal is driven by the SCSI device that wins arbitration.

Target — If the signal is driven, it is driven only by the active target.

The following descriptions of the phases should be read in conjunction with appendix C.

BUS FREE PHASE

The Bus Free Phase is used to indicate that no device is using the bus. It is caused by $\overline{BSY}$ and $\overline{SEL}$ being false, for at least a bus settle delay (400 nanoseconds), and all connected devices releasing control of the bus within a bus clear delay (800 nanoseconds).

ARBITRATION PHASE

The Arbitration Phase is only used in multiple host configurations and is not implemented in relation to this project. The Arbitration Phase is used to avoid bus conflicts. During arbitration, devices try to gain control of the bus by asserting a unique SCSI ID onto one bit of the data bus and releasing the other seven data bits. The device asserting DB7 has the highest priority and the device asserting $\overline{DB0}$ has the lowest priority. To arbitrate for control of the bus, the device has to:

Test for a Bus Free Phase, ie. $\overline{BSY}$ and $\overline{SEL}$ false for at least a bus settle delay.

After detection of a Bus Free Phase, the device waits for at least a bus free delay.

The device then asserts $\overline{BSY}$ and its own SCSI ID.

The device then waits for an arbitration delay and examines the SCSI bus. If there is a higher priority SCSI ID bit on the data bus then the device has lost arbitration and has to release $\overline{BSY}$ and its SCSI ID from the data bus, then wait for the next Bus Free Phase. If there is no higher SCSI ID bit on the data bus the device has won arbitration and asserts $\overline{SEL}$ to claim the bus and enters the Selection Phase.

SELECTION PHASE

For single initiator systems the Selection Phase is entered after the Bus Free Phase. During this phase:

The initiator selects the SCSI target to communicate with by asserting its own address and the address of the target onto the bus.

After a delay the initiator asserts $\overline{\text{SEL}}$.

The target will then assert $\overline{\text{BSY}}$.

Then the initiator releases $\overline{\text{SEL}}$.

$\overline{\text{BSY}}$ remains asserted until the next Bus Free Phase, when the target will release it.

RESELECTION PHASE

The Reselection Phase allows the target to reconnect to an initiator to continue an operation that was previously started by the initiator, but was suspended by the target, ie. the target disconnected by allowing a Bus Free Phase to occur before the operation was finished. Reselection can only be used in systems that have an Arbitration Phase implemented. The Reselection Phase is not supported by the RO652 and is not used in this project.

INFORMATION TRANSFER PHASE

There are four information phases, as shown below. Note that the Data Phase is subdivided into a Data In Phase and a Data Out Phase, and that the Message Phase is subdivided into a Message In Phase and a Message Out Phase.

Command Phase	
Status Phase	
Data Phase:	Data In Phase
	Data Out Phase
Message Phase:	Message In Phase
	Message Out Phase

The SCSI bus signals $\overline{MSG}$, $\overline{C}/D$ and I/O are used to distinguish between the different Information Transfer Phases as shown in the table below. These three signals are all driven by the target device, which therefore controls the SCSI bus and all phase changes. The Target can also cause a Bus Free Phase by releasing $\overline{MSG}$, $\overline{C}/D$, I/O and $\overline{BSY}$.

$\overline{MSG}$	$\overline{C}/D$	I/O	Phase Name	Direction Of Transfer
0	0	0	Data Out	Initiator -> Target
0	0	1	Data In	Target -> Initiator
0	1	0	Command	Initiator -> Target
0	1	1	Status	Target -> Initiator
1	0	0	Reserved	
1	0	1	Reserved	
1	1	0	Message Out	Initiator -> Target
1	1	1	Message In	Target -> Initiator

Once the Information Transfer Phase is complete the bus returns to the Bus Free Phase.

Command Phase

This phase follows the Selection Phase and allows the target to obtain the command information from the initiator.

The Command Phase is entered when $\overline{BSY}$ and $\overline{C}/D$ are asserted and $\overline{MSG}$, I/O, $\overline{SEL}$ and $\overline{ATN}$ are deasserted.

The target waits for a bus settle delay and asserts $\overline{REQ}$ to request the first byte of the Command Descriptor Block (See Chapter 3.4).

The initiator places the first byte on the bus and asserts $\overline{ACK}$.

The Target reads the byte, then releases $\overline{REQ}$.

The initiator will then release $\overline{ACK}$.

The first byte is now transferred and the target will continue to request additional bytes until the complete Command Descriptor Block has been transferred and the Command Phase has ended.

Data Phase

This phase allows the exchange of data between the target and the initiator.

Data In Phase

The Data In Phase allows data to be transferred from the initiator to the target. The Data In Phase is entered when $\overline{\text{BSY}}$ is asserted and $\overline{\text{MSG}}$, $\overline{\text{C}}$/D and I/O are deasserted.

The target asserts $\overline{\text{REQ}}$ to request the first data byte.

The initiator places the first data byte on the bus and asserts $\overline{\text{ACK}}$.

The Target reads the byte, then releases $\overline{\text{REQ}}$.

The initiator will then release $\overline{\text{ACK}}$.

The first data byte is now transferred and the target will continue to request additional data bytes until the last data byte has been transferred.

Data Out Phase

The Data Out Phase allows data to be transferred to the initiator from the target. The Data Out Phase is entered when $\overline{\text{BSY}}$ and I/O are asserted and $\overline{\text{MSG}}$ and $\overline{\text{C}}$/D are deasserted.

The target asserts $\overline{\text{REQ}}$ and the initiator reads the first data byte from the target.

The initiator asserts $\overline{\text{ACK}}$.

The Target releases the data bus and $\overline{\text{REQ}}$.

The initiator will then release $\overline{ACK}$.

The first data byte is now transferred and the target will continue to request that the initiator reads the data until the last data byte has been transferred.

Status Phase

The Status Phase allows the target to request that the initiator reads its Status information. (Chapter 3.4)

The Status Phase is entered when command execution is complete or an error that cannot be recovered from occurs. $\overline{BSY}$, $\overline{C}$/D and I/O are asserted and $\overline{SEL}$ and $\overline{MSG}$ are deasserted.

The target puts the Status Byte on the bus and asserts $\overline{REQ}$.

The initiator reads the byte and asserts $\overline{ACK}$.

This causes the target to release $\overline{REQ}$.

The initiator will then release $\overline{ACK}$.

Message Phase

This phase allows the transfer of a Message information. The first byte transferred can be a single Message Byte (Chapter 3.4) or the first byte of a multiple-byte message.

Message In Phase

The Message In Phase allows a Message Byte to be read from the target by the initiator. The Message In Phase signals the end of an operation and is entered when $\overline{BSY}$, $\overline{MSG}$, $\overline{C}$/D and I/O are are asserted.

The target puts the Message Byte on the bus and asserts $\overline{REQ}$.

The initiator reads the byte and asserts $\overline{ACK}$.

This causes the target to release $\overline{REQ}$.

The initiator will then release $\overline{ACK}$.

The Message In Phase terminates when $\overline{MSG}$ is deasserted.

Message Out Phase

The Message Out Phase is not supported by the RO652 and is not used in this project.

3.3 SCSI BUS CONDITIONS

The SCSI bus has two asynchronous conditions, the Attention Condition and the Reset Condition. These cause the SCSI device to perform certain actions and can alter the phase sequence.

ATTENTION CONDITION

The Attention Condition allows an initiator to inform a target that the initiator has a message ready, (eg. parity error, whereby the target will send the byte again). The target may get this message by performing a Message Out Phase. The initiator creates the Attention Condition by asserting $\overline{ATN}$ during the Arbitration Phase or Bus Free Phase. This condition is not supported by the R0652 and is not used in this project.

RESET CONDITION

The Reset Condition is used to immediately clear all SCSI devices from the bus. When the $\overline{RST}$ signal is received by the SCSI device, the device removes all signals that it is currently asserting from the SCSI bus and clears any current commands. A Bus Free Phase always follows a Reset Condition.

3.4 SCSI COMMANDS

The RO652 hard disk drive which was used in this project is always a target. To execute commands the initiator sends a command, using the appropriate Command Descriptor Block, to the target via the host adaptor. The target performs the command and reports its status, (Message Byte and Status Byte), to the initiator.

OPERATION CODES

The first byte of the Command Descriptor Block contains the Operation Code of the SCSI command. The Operation Code has two parts as shown below, firstly the Group Code and secondly the Command Code. The Group Code is a three bit code, therefore there are eight command groups. The Command Code is a five bit code, therefore there are thirty-two commands within each group. This gives a total of 256 available command operation codes.

Operation Code Command Descriptor Block

Bit	7	6	5	4	3	2	1	0
Byte 0	Group Code			Command Code				

Group Code

Group 0 6 Byte Commands
Group 1 10 Byte Commands
Group 2 Reserved
Group 3 Reserved
Group 4 Reserved
Group 5 12 Byte Commands
Group 6 Vendor Unique
Group 7 Vendor Unique (6 Byte Commands)

Of the eight possible command groups, only group 0, group 1 and group 7 are supported by the RO652. Only the Command Codes implemented by the RO652 are discussed. See ANSI specification X3T9 and common command set book for full description of all of the commands available.

Command Codes

	Hexadecimal Operation Code	Command Name
Group 0	00	Test Unit Ready
	03	Request Sense
	04	Format Unit
	07	Reassign Blocks
	08	Read
	0A	Write
	0B	Seek
	12	Inquiry
	15	Mode Select
	1A	Mode Sense
	1D	Send Diagnostic
Group 1	25	Read Capacity
	28	Read Extended
	2A	Write Extended
	2F	Verify
	37	Read Defect Data
	3C	Read Data Buffer
	3B	Write Data Buffer
Group 7	E0	Maintenance Seek
	E1	Format Maintenance Tracks
	E2	Certify
	E8	Fast Read
	EA	Fast Write

COMMAND DESCRIPTOR BLOCKS FORMAT

The number of bytes contained by the Command Descriptor Block (CDB) is dependent on the group of the command. Group 0 and Group 7 commands have a 6 byte CDB and Group 1 commands have 10 byte CDB. The Command Descriptor Block has an Operation Code as the first byte, as shown above, followed by a Logical Unit Number (LUN), command parameters (if any) and a Control Byte. If there is an invalid parameter in the CDB, the target will stop executing the command without altering the medium. Regardless of command completion, the target will return a status byte and a message byte to the initiator. The RO652 only supports Status Byte 02H (Check Condition) and Message Byte 00H (command complete), see ANSI spec for full description of other status and message bytes. Tables 3.1 and 3.2 show the typical command descriptor blocks for 6 and 10 byte commands. Tables 3.3 and 3.4 show the status and message byte command descriptor blocks.

Table 3.1 : Typical 6 Byte Command Descriptor Block (Group 0 and Group 7)

Bit	7	6	5	4	3	2	1	0
Byte 0	Operation Code							
Byte 1	Logical Unit Number			Logical Block Address MSB (if required)				
Byte 2	Logical Block Address (if required)							
Byte 3	Logical Block Address LSB (if required)							
Byte 4	Transfer Length (if required)							
Byte 5	Control Byte							

Table 3.2 : Typical 10 Byte Command Descriptor Block (Group 1)

Bit	7	6	5	4	3	2	1	0
Byte 0	Operation Code							
Byte 1	Logical Unit Number			Reserved				RelAdr
Byte 2	Logical Block Address MSB (if required)							
Byte 3	Logical Block Address (if required)							
Byte 4	Logical Block Address (if required)							
Byte 5	Logical Block Address LSB (if required)							
Byte 6	Reserved							
Byte 7	Transfer Length MSB (if required)							
Byte 8	Transfer Length LSB (if required)							
Byte 9	Control Byte							

Logical Unit Number — The LUN addresses one of eight devices attached to the target.

Logical Block Address — The logical block address on logical units begins with block 0 and is contigous up to the last logical block on the unit. The logical block concept implies that the initiator and target have previously established the number of data bytes per logical block. This can be done using the Read Capacity command or the Mode Sense command. Group 0 and Group 7 Command Descriptor Blocks have a 21-bit logical block address. Group 1 Command Descriptor Blocks have a 32-bit logical block address.

Relative Address (Rel Adr) — The relative address of Group 1 Command Descriptor Blocks is set to one to indicate that the logical block address is a two's complement displacement. This negative/positive displacement is added to the logical block address last accessed on the unit to form the logical block address for that command. This is only used when linking commands and is not supported by the RO652.

Transfer Length — The transfer length specifies the amount of data to be transferred, usually the number of blocks.

Control Byte — The control byte is the very last byte of every Command Descriptor Block. A typical Control Byte is shown below. For the R0652 Hard disk drive, all bits of the Control Byte are set to zero.

Control Byte

Bit	7	6	5	4	3	2	1	0
	Vendor Unique		Reserved				Flag	Link

Link Bit — If the Link Bit is set to one the initiator wants an automatic link to the next command when the current command is successfully completed. This is not implemented in the RO652.

Flag Bit — If the Link Bit is set to zero, then the Flag Bit is set to zero. If the Link Bit is set to one and the command is successfully completed, the target sends a linked command complete message (if Flag Bit=0) or a linked command complete message with flag (if Flag Bit=1). This bit is typically used to cause an interrupt in the initiator between linked commands.

STATUS

The status returned on completion of a command contains a Status Byte and a Message Byte.

Only the status supported by the RO652 is discussed here.

Table 3.3 : Status Byte

Bit	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	0	0	ERR	0

ERR=0 — no error occurred during command execution.
ERR=1 — error occurred during command execution.

Most error conditions cannot be explained with a single status code and a Request Sense Command should be issued when ERR=1 to determine nature of the error.

Table 3.4 : Message Byte

Bit	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	0	0	0	0

00H Command Complete is the only message supported by the RO652. This message is sent from a target to an initiator to indicate that the execution of a command has terminated and that a valid status has been sent to the initiator. This does not indicate correct or incorrect execution of the command. After this message is sent a Bus Free Phase is entered.

CHAPTER 4

ADVANTAGES OF SMALL COMPUTER SYSTEM INTERFACE (SCSI)

Before the SCSI bus standard emerged there were many different types of devices, most with different interfaces, which would typically be connected to a computer system. For example, a Winchester disk drive would have an ST506 interface, a tape streamer would have a QIC36 interface and a printer would have an RS232 interface. Therefore, it was necessary to have a different hardware controller for each of these devices. Although each of these were already industry standard interfaces, the introduction of SCSI as the new industry standard meant that the Winchester disk drive, the tape streamer and the printer would all have the same interface and could be controlled by the same hardware.

SCSI has a common command set, which allows the loading to be taken off the host system CPU as more tasks are performed by the SCSI controller. For example, Copy is a function of the common command set which allows transfer of data from one device in the system to another without host CPU monitoring. The common command set supports present and future peripherals, so new software drivers do not need to be written for upgraded devices. This makes system design independent of advances in device capacity and performance. Although the same SCSI hardware is able to control printers, hard disk drives, floppy disk drives etc. a different software driver is needed for each of them. The software has to take into account whether the connected device is a block device (eg. a hard disk drive) or a character device (eg. a printer).

SCSI also supports arbitration between connected devices, allowing several tasks from different devices to be interleaved on the bus at the same time as shown in Chapter 3.

4.1 ADVANTAGES FOR HARD DISKS

SCSI supersedes the ST506 interface as the new industry standard for hard disk drives and is supported by most major device manufacturers. The SCSI common command set ensures

compatibility between similar devices from different manufacturers. Previously, a separate input/output driver was needed for each hard disk drive type as they were manufactured with different numbers of heads, cylinders and sectors. SCSI overcomes this by viewing any hard disk drive, regardless of manufacturer, as a string of random logical blocks. As a result, the only difference between drive types is the maximum logical block number which can be addressed. The logical block number is converted to the physical block number by the disk drive alone and a single input/output driver can be developed to run any SCSI disk drive directly. Device specific problems, such as bad blocks on the hard disk media are handled entirely within the hard disk thus reducing the host system CPU involvement as the hard disk appears to be error free media to the operating system.

With current computers tending to be 16 bit and 32 bit microprocessor based, rather than 8 bit microprocessor based, they need more disk storage capacity. There are three methods of increasing hard disk capacity:

Increase the available recording area by adding disks.

The ST506 is limited to 8 heads maximum. SCSI does not have this limitation as it sees only logical blocks when communicating with a hard disk drive and lets the hard disk electronics convert it into physical blocks.

Increase track density.

To increase the track density on an ST506 interface a closed loop servo system (feedback) is needed to drive the read/write heads. This is due to the type of data coding that the ST506 uses. SCSI uses a different type of data coding, which will give a 50% increase in disk capacity, without the need for a closed loop servo system driving the read/write heads.

Increase Flux Changes per Inch (FCI).

ST506 is limited to a maximum transfer rate of 5 Mbits/s. SCSI has a much higher maximum synchronous transfer rate of 32 Mbits/s.

CHAPTER 5

TTL LOGIC INTERFACE BOARD

This board translates between the SCSI bus and the S-100 bus by using 74 series TTL logic gates. The Single Card Computer can be said to be the host and this interface board the host adapter. Figures 5.1 and 5.2 which show the circuit diagram and parts list for the board, Appendix A which contains the software listings and Appendix C which shows the bus phase sequences, should be referred to while reading the explanations in this chapter. The connections on the left hand side of the circuit diagram go to the S-100 bus, ie. to the Single Card Computer (SCC). The connections on the right hand side of the circuit diagram go to the SCSI bus ie. the RO652 hard disk drive. In this case the SCC is the initiator and the RO652 hard disk drive, the target. The circuit can be grouped into three functional parts:

Input/Output Buffers
Input/Output Buffer Enabling
General Components

5.1 INPUT/OUTPUT BUFFERS

The following are the input/output buffers:

IC1 - 74LS240 octal inverting buffer
IC2 - 74LS373 D-type latch
IC3 - 74LS240 octal inverting buffer
IC9 - 74LS240 octal inverting buffer
IC11 - 74LS373 D-type latch
IC12 - 74LS38 2-input NAND

IC1, IC3 AND IC9

IC1, IC3 and IC9 are 74LS240 octal buffer chips, with three-state outputs. The output is the logical inversion of the input and can be in a logic 1 state, a logic 0 state or a high impedance state. When the buffer is in the high impedance state its output appears to be disconnected from the bus. There are two enable inputs ($1\overline{G}$ and $2\overline{G}$) on the 74LS240, on pins 1 and 19. One enable is used to control four of the buffers and the other enable is used to control the

other four buffers. In the circuit these are connected together, to give one enable control input for all eight buffers. The truth table below shows the enable logic of the 74LS240, with $\overline{G}$ being the common enable input, A being the buffer input and Y being the buffer output.

$\overline{G}$	A	Y
0	0	1
0	1	0
1	X	Z

0-Logic 0 (0 Volts), 1-Logic 1 (+5 Volts), X-irrelevant, Z-high impedance (disconnected)

IC2 AND IC11

IC2 and IC11 are 74LS373 octal D-type latches, with three-state outputs. There are two controlling inputs on the 74LS373, enable ($\overline{G}$) and output control (OC). When the OC input is logic 1 the chip goes into its high impedance state. This state does not occur in the circuit as the OC input is connected to 0 volts. The truth table below shows the enable logic of the 74LS373 with $\overline{G}$ being the enable input, D being the latch input and Q being the latch output.

$\overline{G}$	D	Q
1	1	1
1	0	0
0	X	Qo

0-Logic 0 (0 Volts), 1-Logic 1 (+5 Volts), X-irrelevant, Qo-previous Q

IC12

IC12 is a 74LS38 two-input NAND chip with open collector outputs. In open collector circuits the final "pull up" resistor is missing and must be provided by the user. The missing final resistor allows the user to connect more than one open collector output together through the same "pull up" resistor. By doing this the outputs of the gates which are connected through the same "pull up" resistor are OR-ed together.

5.2 INPUT/OUTPUT BUFFER ENABLING

The following are all part of the buffer enable circuitry:

IC4 - 74LS11 three-input AND
IC5 - 74LS00 two-input NAND
IC6 - 74LS04 inverter
IC7 - 74LS244 octal buffer
IC8 - 74LS08 two-input AND
IC10 - 74LS682 8-bit magnitude comparator
SW1 - 8 way single pole/single throw switch

IC4, IC5, IC6 AND IC8

IC4, IC5, IC6 and IC8 function as standard logic gates, as shown in the circuit diagram.

IC7

IC7 the 74LS244 non-inverting octal buffer with three-state outputs is used to buffer A0. A0 has to be buffered so there is only one TTL load on the S100 bus line.

IC10 AND SW1

IC10 the 74LS682 8-bit magnitude comparator with totem pole output is used to decode the base address on A7 through A1. A7 through A1 are on the P inputs and the Q inputs are connected to SW1 which is an 8 way single pole/single throw switch. SW1 is set for a base address of 20H. The 74LS682 has two control outputs, P=Q and P>Q. Only the P=Q output is used in this circuit. The truth table below shows the decode logic of the 74LS682 with P and Q being the address inputs, and P=Q being the decoder output.

P,Q	P=Q
P=Q	0
P>Q	1
P<Q	1

0-Logic 0 (0 Volts), 1-Logic 1 (+5 Volts)

5.3 GENERAL COMPONENTS

The following are all general components:

IC13 - 7805 voltage regulator
RN1 - 220/330 ohm resistor network
RN2 - 220/330 ohm resistor network
C1-C12 - 0.01 micro-farad capacitors

IC13

IC13 is a 7805 voltage regulator which regulates the +8 volts on the S100 bus (pin 51) to a +5 volts supply for the interface board.

RN1 AND RN2

RN1 and RN2 are 220ohm/330ohm resistor networks used to terminate the SCSI bus lines in the correct manner, for single ended drivers. With 220 ohms to +5 volts and 330 ohms to 0 volts.

C1-C12

C1 through C12 are 0.01 micro-farad capacitors which are placed between 0 volts and +5 volts next to the IC packages. These are de-coupling capacitors which are used to smooth out any spikes appearing on the voltage rails.

5.4 SELECTING THE INPUT/OUTPUT BUFFERS

Data/Command/Status and Message byte transfers are controlled by writing to and reading from interface board address 20H. Direction/Information type (ie. data byte, command byte, status byte or message byte) are controlled by reading from and writing to interface board address 21H. The address which is written to/read from controls the buffer selection/disconnection.

FUNCTION	sINP	pDBIN	sOUT	$\overline{pWR}$	I/O	DEVICE(S) SELECTED				
						IC1	IC2	IC3	IC9	IC11
read datap	A	A	R	R	A	S	D	D	D	D
write datap	R	R	A	A	R	D	S	S	D	D
read statp	A	A	R	R	A	D	D	D	S	D
write contp	R	R	A	A	R	D	D	S	D	S

datap (data port)=20H, statp (status port)=21H, contp (control port)=21H
A=Asserted, R=Deasserted, S=Selected, D=Disconnected

IC1

IC1 is selected when there is a 0V signal on the enable input (pin 1 and pin 19). NAND gate 1C outputs a 0V signal to IC1 when the following conditions are satisfied:

(1) Address lines A7-A0 hold address 20H
(2) sINP and pDBIN are asserted

A read from the interface board, address 20H, causes sINP and pDBIN to be asserted which selects IC1. During a read from the interface board sOUT and $\overline{pWR}$ are deasserted and I/O is asserted which disconnects the other buffers. This allows the transfer of data from the RO652 to the SCC data in bus.

IC2

IC2 is selected when there is a +5V signal on the enable input (pin 11). AND gate 2C outputs a +5V signal to IC2 when the following conditions are satisfied:

(1) Address lines A7-A0 hold address 20H
(2) sOUT and $\overline{pWR}$ are asserted

A write to the interface board, address 20H, causes sOUT and $\overline{pWR}$ to be asserted, and I/O to be deasserted which selects IC2 and IC3. During a write to the interface board sINP and pDBIN are deasserted which disconnects the other buffers. This allows the transfer of data from the SCC data out bus to the RO652.

IC3

IC3 is selected when there is a 0V signal on the enable input (pin 1 and pin 19). INVERTER gate 1E outputs a 0V signal to IC3 when I/O from the SCSI bus is inactive.

A write to the interface board, address 20H or address 21H, causes sOUT and $\overline{\text{pWR}}$ to be asserted, and I/O to be deasserted which selects IC3. During a write to the interface board sINP and pDBIN are deasserted which disconnects the other buffers. This allows the transfer of data and control information from the SCC data out bus to the RO652.

IC9

IC9 is selected when there is a 0V signal on the enable input (pin 1 and pin 19). NAND gate 3C outputs a 0V signal to IC9 when the following conditions are satisfied:

(1) Address lines A7-A0 hold address 21H
(2) sINP and pDBIN are asserted

A read from the interface board, address 21H, causes sINP and pDBIN to be asserted which selects IC9. During a read from the interface board sOUT and $\overline{\text{pWR}}$ are deasserted and I/O is asserted which disconnects the other buffers. This allows transfer of status information from the RO652 to the SCC data in bus.

IC11

IC11 is selected when there is a +5V signal on the enable input (pin 11). AND gate 4C outputs a +5V signal to IC11 when the following conditions are satisfied:

(1) Address lines A7-A0 hold address 21H
(2) sOUT and $\overline{\text{pWR}}$ are asserted

A write to the interface board, address 21H, causes sOUT and $\overline{\text{pWR}}$ to be asserted, and I/O to be deasserted which selects IC11 and IC3. During a write to the interface board sINP and pDBIN are deasserted which disconnects the other buffers. This allows the transfer of control information from the SCC data out bus to the RO652.

5.5 CONTROL SIGNALS

Certain SCC and SCSI signals are used to control the transfer of data, commands, status and message signals between the data buses. The SCC control signals are used to select/disconnect the buffers during different phases. The SCSI control lines are the lines used to control what is being transferred on the data bus.

SCC CONTROL SIGNALS

sINP (output from SCC) — This signal is connected to the input/output buffer enabling circuitry. sINP is asserted (+5V) during a data input to the SCC, deasserted (0V) at all other times.

pDBIN (output from SCC) — This signal is connected to the input/output buffer enabling circuitry. pDBIN is asserted (+5V) when data is latched onto onto data bus, deasserted (0V) at all other times.

sOUT (output from SCC) — This signal is connected to the input/output buffer enabling circuitry. sOUT is asserted (+5V) during a data output from the SCC, deasserted (0V) at all other times.

$\overline{\text{pWR}}$ (output from SCC) — This signal is connected to the input/output buffer enabling circuitry. $\overline{\text{pWR}}$ is asserted (0V) during a data write from the SCC, deasserted (+5V) at all other times.

A0 (output from SCC) — This signal is connected to the input/output buffer enabling circuitry. When A0 is asserted (+5V) transfer is between the RO652 control bus and the SCC data in or data out bus. When A0 is deasserted (0V) transfer is between the RO652 data bus and the SCC data in or data out bus.

SCSI CONTROL SIGNALS

$\overline{\text{ACK}}$ (output from SCC) — This signal is connected, via IC11, from the SCC data out line DO0 to the RO652 control bus. $\overline{\text{ACK}}$ is asserted (0V) to acknowledge a request ($\overline{\text{REQ}}$) from the RO652 hard disk drive. $\overline{\text{ACK}}$ is deasserted (+5V) at other times.

$\overline{\text{RST}}$ (output from SCC) — This signal is connected, via IC11, from the SCC data out line DO1 to the RO652 control bus. $\overline{\text{RST}}$ is asserted (0V) to clear all bus activity. $\overline{\text{RST}}$ is deasserted (+5V) at other times.

$\overline{\text{SEL}}$ (output from SCC) — This signal is connected, via IC11, from the SCC data out line DO2 to the RO652 control bus. $\overline{\text{SEL}}$ is asserted (0V) during selection of the RO652 hard disk drive. $\overline{\text{SEL}}$ is deasserted (+5V) at other times.

$\overline{\text{BSY}}$ (input to SCC) — This signal is connected, via IC9, from the RO652 control bus to the SCC data in line DI0. $\overline{\text{BSY}}$ is asserted (0V) when the SCSI bus is busy. $\overline{\text{BSY}}$ is deasserted (+5V) at other times.

$\overline{MSG}$ (input to SCC) This signal is connected, via IC9, from the RO652 control bus to the SCC data in line DI1. $\overline{MSG}$ is asserted (0V) when the SCSI bus is in a message phase. $\overline{MSG}$ is deasserted (+5V) at other times.

C/D (input to SCC) This signal is connected, via IC9, from the RO652 control bus to the SCC data in line DI2. C/D is asserted (0V) when control information is on the SCSI bus. C/D is deasserted (+5V) when data information is on the SCSI bus.

$\overline{REQ}$ (input to SCC) This signal is connected, via IC9, from the RO652 control bus to the SCC data in line DI3. $\overline{REQ}$ is asserted (0V) when the RO652 is requesting something from the SCC. $\overline{REQ}$ is deasserted (+5V) at other times.

I/O (input to SCC) This signal is connected, via IC9, from the RO652 control bus to the SCC data in line DI4. I/O is asserted (0V) when the direction of transfer on the SCSI bus is from the RO652 to the SCC. I/O is deasserted (+5V) when the direction of transfer on the SCSI bus is from the SCC to the RO652.

5.6 BUS PHASES

This section shows the relevant circuitry active during each of the phases with reference to the subroutines in the SCSI.Z80 software (Appendix A.5).

SELECTION PHASE

During the Selection Phase the RO652 hard disk drive is selected by the SCC. The SELECT subroutine of SCSI.Z80 controls the selection of the RO652 hard disk as follows.

The address of the RO652 hard disk drive is written from the SCC to the RO652 data port, IC2 and IC3 selected.

$\overline{SEL}$ is asserted by writing to the RO652 control port, IC3 and IC11 selected.

The RO652 status port is read until $\overline{BSY}$ is asserted, IC9 selected.

$\overline{SEL}$ is deasserted by writing to the RO652 control port, IC3 and IC11 selected.

This completes the Selection Phase.

DATA IN PHASE

During the Data In Phase data is sent from the RO652 hard disk drive to the VDU, via the SCC. The SCSIRD subroutine of SCSI.Z80 controls the transfer of a data byte as follows.

The RO652 status port is read, IC9 selected, until $\overline{REQ}$ and $\overline{BSY}$ are asserted and $\overline{MSG}$, $\overline{C}$/D and I/O are deasserted.

The RO652 data port is read, IC1 selected, to transfer the data byte into the SCC.

The SCC then outputs the data byte, which was read from the drive, to the VDU.

$\overline{ACK}$ is asserted by writing to the RO652 control port, IC3 and IC11 selected. (SCC has read the data byte).

The RO652 status port is read, IC9 selected, until $\overline{REQ}$ is false.

$\overline{ACK}$ is deasserted by writing to the RO652 control port, IC3 and IC11 selected.

The data byte transfer is complete.

DATA OUT PHASE

During the Data Out Phase data is sent from the VDU, via the SCC, to the RO652 hard disk drive. The SCSIWR subroutine of SCSI.Z80 controls the transfer of a data byte as follows.

The data byte, to be written to the RO652, is read from the VDU into the SCC.

The RO652 status port is read, IC9 selected, until $\overline{REQ}$, $\overline{BSY}$, I/O are asserted and $\overline{MSG}$ and $\overline{C}$/D are deasserted.

The SCC writes the data byte to the RO652 data port, IC2 and IC3 selected, to transfer the data byte into the RO652.

$\overline{ACK}$ is asserted by writing to the RO652 control port, IC3 and IC11 selected. (SCC has written the data byte).

The RO652 status port is read, IC9 selected, until $\overline{REQ}$ is false.

$\overline{ACK}$ is deasserted by writing to the RO652 control port, IC3 and IC11 selected.

The data byte transfer is complete.

COMMAND PHASE

During the Command Phase data is sent from the VDU, via the SCC, to the RO652 hard disk drive. The SEND subroutine of SCSI.Z80 controls the transfer of a command byte as follows.

The data byte, to be written to the RO652, is read from the VDU into the SCC.

The RO652 status port is read, IC9 selected, until $\overline{REQ}$, $\overline{BSY}$ and $\overline{C}$/D are asserted and $\overline{MSG}$ and I/O are deasserted.

The SCC writes the command byte to the RO652 data port, IC2 and IC3 selected, to transfer the data byte into the RO652.

$\overline{ACK}$ is asserted by writing to the RO652 control port, IC3 and IC11 selected. (SCC has read the data byte).

The RO652 status port is read, IC9 selected, until $\overline{REQ}$ is false.

$\overline{ACK}$ is deasserted by writing to the RO652 control port, IC3 and IC11 selected.

The command byte transfer is complete.

STATUS PHASE

During the Status Phase status information is sent from the RO652 hard disk drive to the VDU, via the SCC. The STATUS subroutine of SCSI.Z80 controls the transfer of a status byte as follows.

The RO652 status port is read, IC9 selected, until $\overline{\text{REQ}}$, C/D, I/O and $\overline{\text{BSY}}$ are asserted and $\overline{\text{MSG}}$ is deasserted.

The RO652 data port is read, IC1 selected, to transfer the status byte into the SCC.

The SCC then outputs the status byte, which was read from the drive, to the VDU.

$\overline{\text{ACK}}$ is asserted by writing to the RO652 control port, IC3 and IC11 selected. (SCC has read the data byte).

The RO652 status port is read, IC9 selected, until $\overline{\text{REQ}}$ is false.

$\overline{\text{ACK}}$ is deasserted by writing to the RO652 control port, IC3 and IC11 selected.

The status byte transfer is complete.

MESSAGE IN PHASE

During the Message In Phase message information is sent from the RO652 hard disk drive to the VDU, via the SCC. The MESSAGE subroutine of SCSI.Z80 controls the transfer of a message byte as follows.

The RO652 status port is read, IC9 selected, until $\overline{\text{REQ}}$, C/D, I/O, $\overline{\text{BSY}}$ and $\overline{\text{MSG}}$ are asserted.

The RO652 data port is read, IC1 selected, to transfer the message byte into the SCC.

The SCC then outputs the message byte, which was read from the drive, to the VDU.

$\overline{ACK}$ is asserted by writing to the RO652 control port, IC3 and IC11 selected. (SCC has read the data byte).

The RO652 status port is read, IC9 selected, until $\overline{REQ}$ is false.

$\overline{ACK}$ is deasserted by writing to the RO652 control port, IC3 and IC11 selected.

The message byte transfer is complete.

5.7 SOFTWARE

The listings of the programs are in Appendix A. There are five programs which control the TTL logic interface board:

MAIN.Z80
EQU.Z80
MSG.Z80
VDU.Z80
SCSI.Z80

MAIN.Z80 is the main program, the other four are programs which are included into the assembled firmware.

MAIN.Z80

This is the main program which controls VDU, SCC and RO652 input/output, by utilising the subroutines, equates etc. contained in the other programs.

EQU.Z80

This file contains:

Equates for the VDU control
VDU ASCII equates
SCSI equates
SCSI status bits equates
SCSI control bits equates
SCC RAM storage size

MSG.Z80

This file has the messages which are sent to the VDU during program operation.

VDU.Z80

In this file are the subroutines which control VDU input/output. (Baud rate selection and reading from and writing to the VDU screen.)

SCSI.Z80

As shown in Chapter 5.6, this is file which has the subroutines used to control the RO652 hard disk drive phases.

Figure 5.1

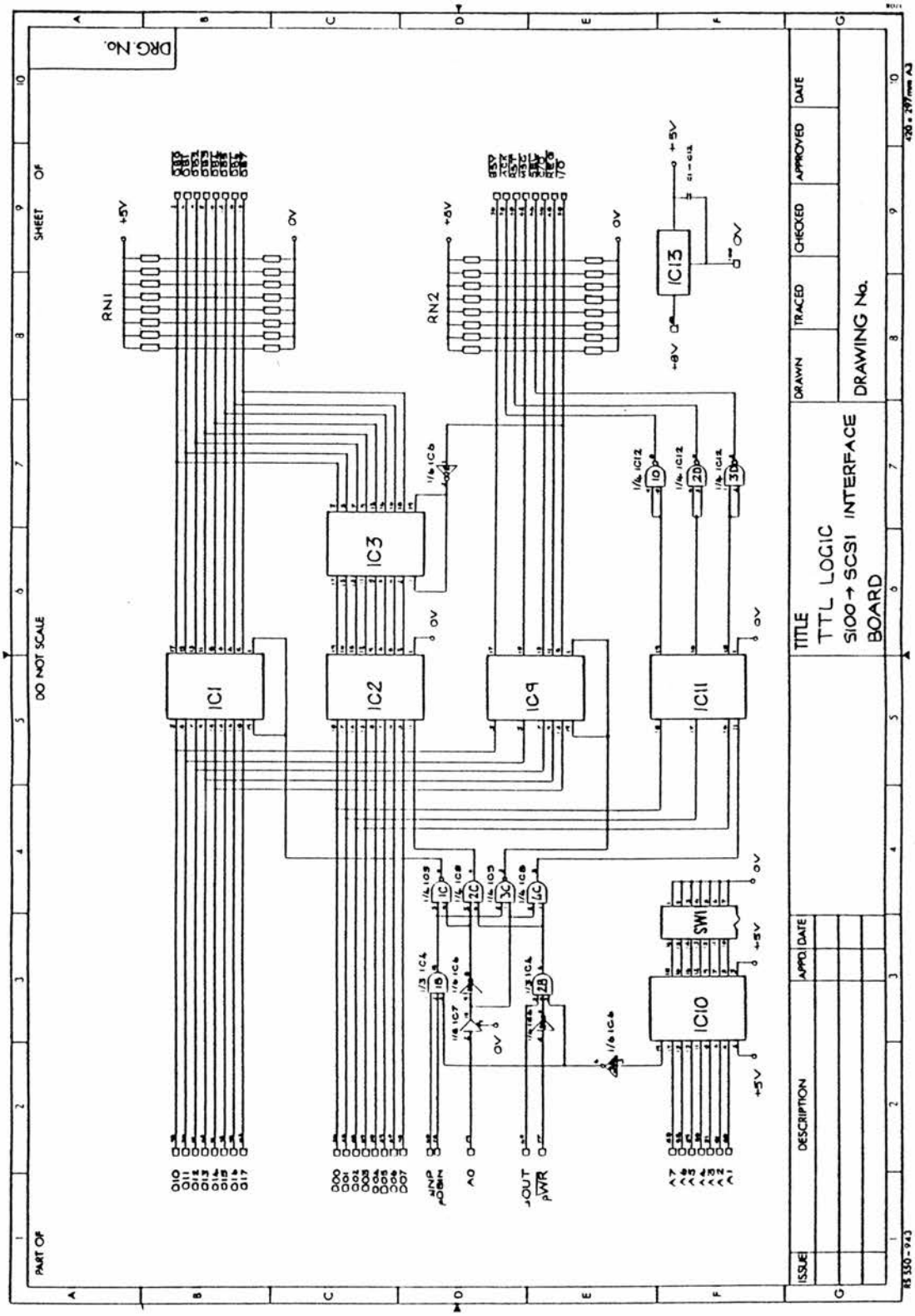
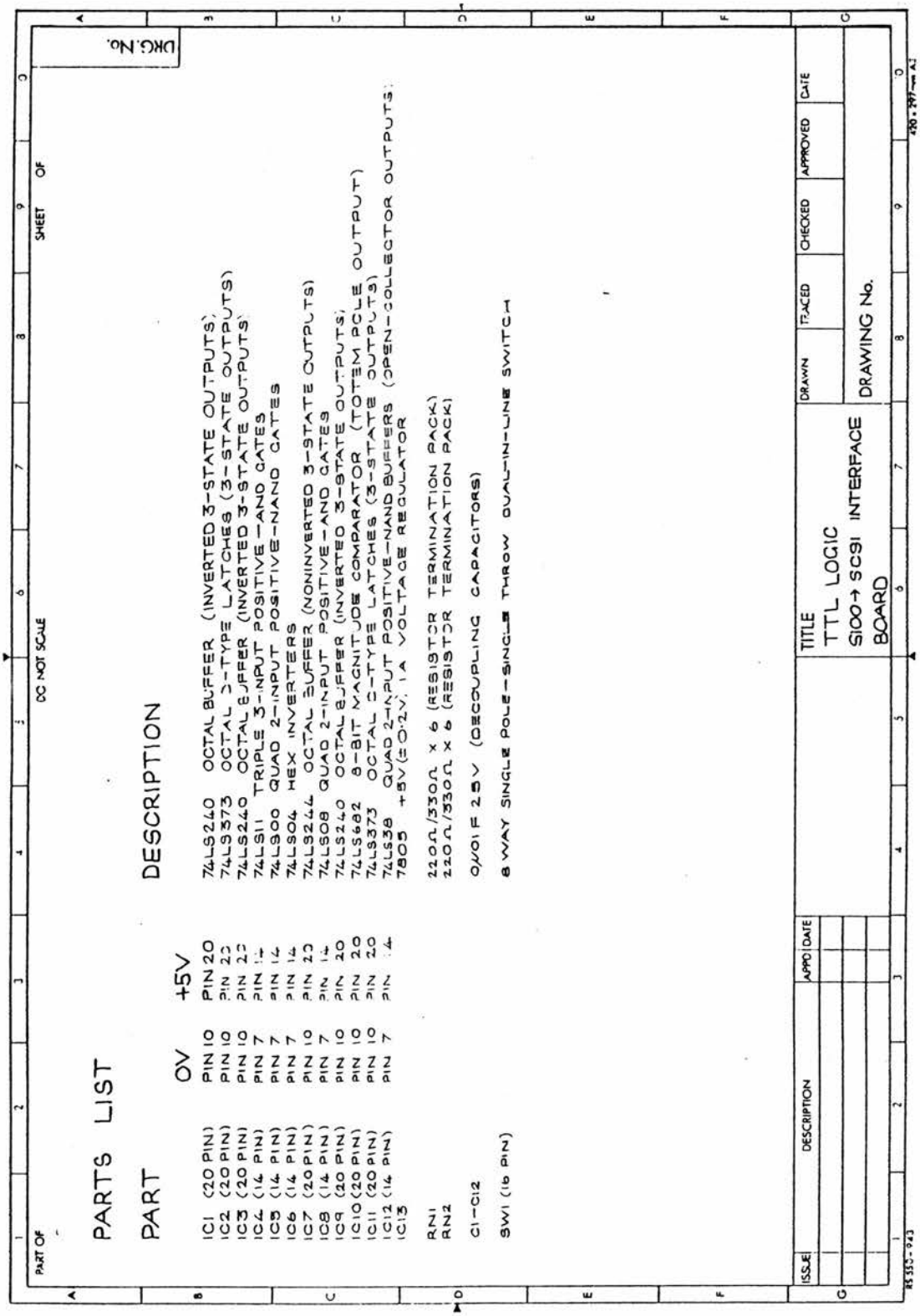

DRG No.
SHEET OF
DO NOT SCALE
PART OF
TITLE
TTL LOGIC
S100→SCSI INTERFACE
BOARD
DRAWING No.
DRAWN
TRACED
CHECKED
APPROVED
DATE
ISSUE
DESCRIPTION
APPD
DATE
RN1
RN2
IC1
IC2
IC3
IC9
IC10
IC11
IC13
SW1
+5V
0V
-5V
A0
JOUT
PWR

Figure 5.2


PART OF
DO NOT SCALE
SHEET OF
DRG. No.
PARTS LIST
PART
0V
+5V
DESCRIPTION
IC1 (20 PIN) PIN 10 PIN 20 74LS240 OCTAL BUFFER (INVERTED 3-STATE OUTPUTS)
IC2 (20 PIN) PIN 10 PIN 20 74LS373 OCTAL D-TYPE LATCHES (3-STATE OUTPUTS)
IC3 (20 PIN) PIN 10 PIN 20 74LS240 OCTAL BUFFER (INVERTED 3-STATE OUTPUTS)
IC4 (14 PIN) PIN 7 PIN 14 74LS11 TRIPLE 3-INPUT POSITIVE-AND GATES
IC5 (14 PIN) PIN 7 PIN 14 74LS00 QUAD 2-INPUT POSITIVE-NAND GATES
IC6 (14 PIN) PIN 7 PIN 14 74LS04 HEX INVERTERS
IC7 (20 PIN) PIN 10 PIN 20 74LS244 OCTAL BUFFER (NONINVERTED 3-STATE OUTPUTS)
IC8 (14 PIN) PIN 7 PIN 14 74LS08 QUAD 2-INPUT POSITIVE-AND GATES
IC9 (20 PIN) PIN 10 PIN 20 74LS240 OCTAL BUFFER (INVERTED 3-STATE OUTPUTS)
IC10 (20 PIN) PIN 10 PIN 20 74LS682 8-BIT MAGNITUDE COMPARATOR (TOTEM POLE OUTPUT)
IC11 (20 PIN) PIN 10 PIN 20 74LS373 OCTAL D-TYPE LATCHES (3-STATE OUTPUTS)
IC12 (14 PIN) PIN 7 PIN 14 74LS38 QUAD 2-INPUT POSITIVE-NAND BUFFERS (OPEN-COLLECTOR OUTPUTS)
IC13 7805 +5V(±0.2V) 1A VOLTAGE REGULATOR
RN1 220Ω/330Ω x 6 (RESISTOR TERMINATION PACK)
RN2 220Ω/330Ω x 6 (RESISTOR TERMINATION PACK)
C1-C12 0.01 F 25V (DECOUPLING CAPACITORS)
SW1 (16 PIN) 8 WAY SINGLE POLE-SINGLE THROW DUAL-IN-LINE SWITCH
ISSUE
DESCRIPTION
APPD
DATE
TITLE
TTL LOGIC
SIOO→SCSI INTERFACE
BOARD
DRAWN
TRACED
CHECKED
APPROVED
DATE
DRAWING No.
RS 335-923
210 x 297 mm A4

CHAPTER 6

INTELLIGENT INTERFACE BOARD

This board translates between the SCSI bus and the S-100 bus using TTL logic and a NCR5380 SCSI interface chip. As in Chapter 5, the Single Card Computer is said to be the host (initiator) and the RO652 hard disk drive the target. Appendix B, containing the software used to control transfers, Appendix C which shows the bus phase sequences and Figures 6.1 and 6.2 which show the circuit diagram and the parts list, should be referred to while reading this chapter. The connections on the left hand side of the circuit diagram go to the S-100 bus, ie. to the Single Card Computer (SCC). The connections on the right hand side of the circuit diagram go to the SCSI bus ie. the RO652 hard disk drive. The circuit can be grouped into three functional parts:

Input/Output
Input/Output Selection
General Components

6.1 INPUT/OUTPUT

The following are the input/output ICs

IC1 - 74LS244 octal buffer
IC2 - 74LS244 octal buffer
IC5 - NCR53C80 SCSI interface chip

IC1 AND IC2

IC1 and IC2 are 74LS244 non-inverting octal buffer chips, with three state outputs. As before, the output can be in a logic 1 state, a logic 0 state or a high impedance state. There are two enable inputs ($1\overline{G}$ and $2\overline{G}$) on the 74LS244, on pins 1 and 19. One enable is used to control four of the buffers and the other enable is used to control the other four buffers. These two inputs are connected together, to give one enable control input for all eight buffers. The truth table below shows the enable logic of the 74LS244, with $\overline{G}$ being the common enable input, A being the buffer input and Y being the buffer output.

$\overline{G}$	A	Y
0	0	0
0	1	1
1	X	Z

0-Logic 0 (0 Volts), 1-Logic 1 (+5 Volts), X-irrelevant, Z-high impedance (disconnected)

IC5

IC5 is a VLSI chip, the NCR53C80. This is a CMOS SCSI interface chip designed to support the ANSI specification for Small Computer Systems Interfaces, X3T9.2. In this project the chip is operated in the initiator mode but it can also be operated in the target mode. The chip is controlled by reading from and writing to several internal registers, Chapter 6.6 discusses these registers. The NCR53C80 can be grouped into seven functional parts:

SCSI Bus Signals
DMA Control Signals
Data Bus Signals
Reset Conditions
Interrupts
Mode of Operation
Internal Register Signals

SCSI Bus Signals

$\overline{DB0}$-$\overline{DB7}$	$\overline{RST}$	$\overline{MSG}$
$\overline{DBP}$	$\overline{ATN}$	$\overline{C}$/D
$\overline{BSY}$	$\overline{ACK}$	I/O
$\overline{SEL}$	$\overline{REQ}$	

These eighteen signals are identical to those shown in Chapter 3. SCSI bus signal $\overline{ATN}$ is not used in this project because the message out phase is not supported.

DMA Control Signals

$\overline{EOP}$	DRQ
READY	$\overline{DACK}$

The NCR53C80 is not used in a DMA mode in relation to this project. Input signals $\overline{EOP}$ and $\overline{DACK}$ are held at +5V, deasserted state (ie. not active). Output signals READY and DRQ are not used.

Data Bus Signals

D0-D7	Data Bus (D0-D7) is the bidirectional, tri-state bus connected to the host computer.

Reset Conditions

There are three possible reset conditions within the NCR53C80:

Hardware Reset	Input signal $\overline{RESET}$ is used to clear all internal registers, it does not reset the SCSI bus.
SCSI $\overline{RST}$ Received	This performs a chip reset, clearing all internal registers except port, 1 bit 7 which is the assert $\overline{RST}$ bit. $\overline{RST}$ is used to clear all SCSI bus activity.
SCSI $\overline{RST}$ Issued	If the host sets the assert $\overline{RST}$ bit (bit 7, port 1) all internal registers are cleared apart from port 1, bit 7. $\overline{RST}$ will be active, clearing all SCSI bus activity, until port 1, bit 7 is deasserted or the NCR53C80 $\overline{RESET}$ input is active.

Interrupts

Output signal IRQ is not used in relation to this project. IRQ is used to inform the host of an error or an event completion.

Mode of Operation

The NCR53C80 supports four modes of operation: programmed I/O transfers; normal DMA mode; block DMA mode and pseudo DMA mode. Programmed I/O is used in this project, this uses the $\overline{REQ}/\overline{ACK}$ handshake method of controlling data transfers.

Internal Registers Signals

Signals $\overline{CS}$, $\overline{IOR}$, $\overline{IOW}$ and A0-A2 are used to address all internal registers. $\overline{CS}$ enables a read or a write of one of the eight internal registers selected. All data transfers between the SCC and the RO652 are controlled by these registers. See Chapter 6.6 for explanation of the registers used in this project.

6.2 INPUT/OUTPUT SELECTION

The following are all part of the input/output select circuitry:

IC4 - 74LS08 2-input AND
IC6 - 74LS04 inverter
IC7 - 74LS10 3-input NAND
IC3 - 74LS682 8-bit magnitude comparator
SW1 - 8 way single pole/single throw switch

IC4, IC6 AND IC7

IC4, IC6 and IC7 function as standard logic gates, as shown in the circuit diagram.

IC3 AND SW1

IC3 the 74LS682 8-bit magnitude comparator with totem pole output is used to decode the base address on A7 through A3. A7 through A3 are on the P inputs and the Q inputs are connected to SW1 which is an 8 way single pole/single throw switch. SW1 is set for a base address of 20H. The 74LS682 has two control outputs, P=Q and P>Q. Only the P=Q output is used in this circuit. The truth table below shows the decode logic of the 74LS682 with P and Q being the address inputs, and P=Q being the decoder output.

P,Q	P=Q
P=Q	0
P>Q	1
P<Q	1

0-Logic 0 (0 Volts), 1-Logic 1 (+5 Volts)

6.3 GENERAL COMPONENTS

The following are all general components:

IC8 - 7805 voltage regulator
RN1 - 220/330 ohm resistor network
RN2 - 220/330 ohm resistor network
R1,R2 - 1K ohm resistors
C1-C7 - 0.01 micro-farad capacitors

IC8

IC8 is a 7805 voltage regulator which regulates the +8 volts on the S100 bus (pin 51) to a +5 volts supply for the interface board.

RN1 AND RN2

RN1 and RN2 are 220ohm/330ohm resistor networks used to terminate the SCSI bus lines in the correct manner, for single ended drivers. With 220 ohms to +5 volts and 330 ohms to 0 volts.

R1 AND R2

R1 and R2 are 1K ohm resistors used as pull-up resistors on the NCR53C80 inputs $\overline{\text{EOP}}$ and $\overline{\text{DACK}}$, preventing them from being asserted.

C1-C7

C1 through C7 are 0.01 micro-farad capacitors which are placed between 0 volts and +5 volts next to the IC packages. These are de-coupling capacitors which are used to smooth out any spikes appearing on the voltage rails.

6.4 SELECTING THE INPUT/OUTPUT

Data/Command/Status and Message byte transfers are controlled by writing to and reading from the interface board, base address 20H. The direction of the information on the data bus controls the buffer selection/disconnection.

The following tables shows the chips that are selected and disconnected with each read/write function.

FUNCTION	sINP	pDBIN	sOUT	$\overline{pWR}$	DEVICE(S) SELECTED		
					IC1	IC2	IC5
read datap	A	A	R	R	D	S	S
write datap	R	R	A	A	S	D	S
write icmdp	R	R	A	A	S	D	S
write modep	R	R	A	A	S	D	S
write tcmdp	R	R	A	A	S	D	S
read stat1p	A	A	R	R	D	S	S
read prstp	A	A	R	R	D	S	S

datap (data register)=20H, icmdp (initiator command register)=21H, modep (mode register=22H), tcmdp (target command register)=23H, stat1p (bus status register)=24H, prstp (reset parity/interrupt register)=27H
A=Asserted, R=Deasserted, S=Selected, D=Disconnected

IC1

IC1 is selected when there is a 0V signal on the enable input (pin 1 and pin 19). NAND gate 2B outputs a 0V signal to IC1 when the following conditions are satisfied:

(1) Address lines A7-A3 hold base address 20H
(2) sOUT and $\overline{pWR}$ are asserted

A write to the interface board, base address 20H, causes sOUT and $\overline{pWR}$ to be asserted which selects IC1. This allows the transfer of information from the SCC data out bus to the NCR53C80 (IC5). During a write to the interface board sINP and pDBIN are deasserted which disconnects IC2.

IC2

IC2 is selected when there is a 0V signal on the enable input (pin 1 and pin 19). NAND gate 1B outputs a 0V signal to IC2 when the following conditions are satisfied:

(1) Address lines A7-A3 hold base address 20H
(2) sINP and pDBIN are asserted

A read from the interface board, base address 20H, causes sINP and pDBIN to be asserted which selects IC2. This allows the transfer of information from the NCR53C80 (IC5) to the SCC data in bus. During a read from the interface board sOUT and $\overline{\text{pWR}}$ are deasserted which disconnects IC1.

IC5

IC5 is selected when there is a 0V signal on the $\overline{\text{CS}}$ input (pin 19). AND gate 3C outputs a 0V signal to IC5 when the output of 1B or 2B is 0V. This happens when there is a read or a write to base address 20H. When there is a read from base address 20H, $\overline{\text{IOR}}$ is asserted (0V) and when there is a write to base address 20H, $\overline{\text{IOW}}$ is asserted (0V). This allows transfers to take place between the RO652 and the SCC.

6.5 CONTROL SIGNALS

Certain SCC control signals are used to control the NCR53C80. The control signals are used to select/disconnect IC1 and IC2, enable or disable the NCR53C80 and set up the correct conditions on the NCR53C80 for communication between the SCC and the RO652. The NCR53C80 control signals are used to address the eight internal registers which control data transfers between the SCC and the RO652.

SCC CONTROL SIGNALS

sINP (output from SCC)	This signal is connected to the input/output buffer enabling circuitry. sINP is asserted (+5V) during a data input to the SCC, deasserted (0V) at all other times.

pDBIN (output from SCC) — This signal is connected to the input/output buffer enabling circuitry. pDBIN is asserted (+5V) when data is latched onto onto data bus, deasserted (0V) at all other times.

sOUT (output from SCC) — This signal is connected to the input/output buffer enabling circuitry. sOUT is asserted (+5V) during a data output from the SCC, deasserted (0V) at all other times.

$\overline{pWR}$ (output from SCC) — This signal is connected to the input/output buffer enabling circuitry. $\overline{pWR}$ is asserted (0V) during a data write from the SCC, deasserted (+5V) at all other times.

A0-A2 (output from SCC) — A0 through A2 are connected to the NCR53C80 A0-A2 input. These are used to control the NCR53C80 internal register selection.

$\overline{pRESET}$ (output from SCC) — This signal is connected to the NCR53C80 $\overline{RESET}$ input. When $\overline{pRESET}$ is asserted (0V) the NCR53C80 clears all its internal registers. This does not force the SCSI bus signal $\overline{RST}$ low and therefore, it does not reset the SCSI bus.

NCR53C80 CONTROL SIGNALS

$\overline{CS}$ — This signal comes from the input/output selection circuitry. When $\overline{CS}$ is asserted (0V), a read/write of the internal register addressed by A0-A2 can take place.

$\overline{IOR}$ — This signal comes from the input/output selection circuitry. When $\overline{IOR}$ is asserted (0V), a read of the internal register selected by $\overline{CS}$, A0-A2 takes place.

$\overline{IOW}$ — This signal comes from the input/output selection circuitry. When $\overline{IOW}$ is asserted (0V), a write to the internal register selected by $\overline{CS}$, A0-A2 takes place.

A0-A2 — These signals come directly from the SCC. The address selects a specific register.

6.6 NCR53C80 INTERNAL REGISTERS

The table below shows a summary of the NCR53C80 internal register selection.

$\overline{CS}$	$\overline{IOR}$	$\overline{IOW}$	A2	A1	A0	REGISTER SELECTED	USED
R	R	A	R	R	R	Output Data	YES
R	A	R	R	R	R	Current SCSI Data	YES
R	R	A	R	R	A	Initiator Command	YES
R	A	R	R	R	A	Initiator Command	NO
R	R	A	R	A	R	Mode	YES
R	A	R	R	A	R	Mode	NO
R	R	A	R	A	A	Target Command	YES
R	A	R	R	A	A	Target Command	NO
R	R	A	A	R	R	Select Enable	NO
R	A	R	A	R	R	Current SCSI Bus Status	YES
R	R	A	A	R	A	Start DMA send	NO
R	A	R	A	R	A	Bus and Status	NO
R	R	A	A	A	R	Start DMA Target Receive	NO
R	A	R	A	A	R	Input Data	NO
R	R	A	A	A	A	Start DMA Initiator Receive	NO
R	A	R	A	A	A	Reset Parity/Interrupts	YES

A-asserted, R-deasserted
Only the registers used in relation to this project are discussed. See the NCR53C80 Design Manual for descriptions of unused registers.

Output Data Register/Current Scsi Data Register

Address 0 - The Output Data Register and Current SCSI Data Register are referred to in the phase explanations (Chapter 6.7) as the data register.

Write datap - This register is used to send data to the SCSI bus and to assert SCSI ID bits during selection phase.

7	6	5	4	3	2	1	0
$\overline{DB7}$	$\overline{DB6}$	$\overline{DB5}$	$\overline{DB4}$	$\overline{DB3}$	$\overline{DB2}$	$\overline{DB1}$	$\overline{DB0}$

$\overline{DB0}$-$\overline{DB7}$ are as the SCSI bus signals of the same name.

Read datap - This register allows the host to read the active SCSI data bus.

7	6	5	4	3	2	1	0
$\overline{DB7}$	$\overline{DB6}$	$\overline{DB5}$	$\overline{DB4}$	$\overline{DB3}$	$\overline{DB2}$	$\overline{DB1}$	$\overline{DB0}$

$\overline{DB0}$-$\overline{DB7}$ are as the SCSI bus signals of the same name.

Initiator Command Register

Address 1 - The Initiator Command Register.

Write icmdp - This register is used to assert some SCSI bus signals.

7	6	5	4	3	2	1	0
Assert RST	TM	DE	Assert $\overline{ACK}$	Assert $\overline{BSY}$	Assert $\overline{SEL}$	Assert $\overline{ATN}$	Assert D BUS

Assert SIGNAL/D BUS — Reading these bits shows the status of the signal/data bus. If the bit is one (1) then the signal/data bus is active. When a one is written to any of these bytes the signal/data bus is asserted.

TM — Test Mode. When a one is written to this bit the NCR53C80 output drivers are disabled.

DE — Differential Enable. When a one is written to this bit the external differential pair driver option is enabled.

Only bits 7 (assert $\overline{RST}$), 4 (assert $\overline{ACK}$), 2 (assert $\overline{SEL}$) and 0 (assert DATA BUS) are used in relation to this project.

Mode Register

Address 2 - The Mode Register.

Write modep - This register controls the operational mode of the chip.

7	6	5	4	3	2	1	0
B Mode DMA	Target Mode	Enable P Chk	Enable P Int	Enable EOP Int	Monitor Busy	DMA Mode	Arb

B Mode DMA — Block Mode DMA. This controls the characteristics of DMA handshakes. Set to 0 during selection phase.

Target Mode — Sets up NCR53C80 as an initiator or a target. This bit is set to 0 (NCR53C80 is an initiator) during the selection phase.

Enable P Chk — Enable Parity Checking. This bit is set to 0 (ignore parity) during the selection phase.

Enable P Int — Enable Parity Interrupt. This bit is set to 0 (interrupt disabled) during the selection phase.

Enable EOP Int — Enable EOP interrupt. This bit is set to 0 (interrupt disabled) during the selection phase.

Monitor Busy — This bit is set to 0 (no interrupt when $\overline{BSY}$ lost) during the selection phase.

DMA Mode — This bit is set to 0 (no DMA) during the selection phase.

Arb — Arbitrate. This bit is set to 0 (no arbitration) during the selection phase.

Target Command Register

Address 3 - The Target Command Register.

Write tcmdp - This register mode register allows the host to control the SCSI bus Information Transfer Phase and assertion of $\overline{REQ}$.

7	6	5	4	3	2	1	0
LBS	X	X	X	Assert $\overline{REQ}$	Assert $\overline{MSG}$	Assert $\overline{C}$/D	Assert I/O

LBS — Last Byte Sent. This is used by the NCR53C80 to indicate, during DMA, when the last byte has been sent to the SCSI bus.

Assert SIGNAL — Assert $\overline{REQ}$ has no meaning when operating as an initiator. When a one is written to any of these bytes the signal is asserted.

Current Scsi Bus Status Register

Address 4 - The Current SCSI Bus Status Register.

Read stat1p - This register is used to monitor seven SCSI bus control signals and the data parity bit.

7	6	5	4	3	2	1	0
$\overline{RST}$	$\overline{BSY}$	$\overline{REQ}$	$\overline{MSG}$	$\overline{C}$/D	I/O	$\overline{SEL}$	$\overline{DBP}$

The signals are as the SCSI bus signals of the same name.

Start Dma Send Register/Bus And Status Register

Address 5 - This register is not used in this project.

Input Data Register/Start Dma Target Receive Register

Address 6 - This register is not used in this project.

Reset Parity/Interrupts Register

Address 7 - Reset Parity/Interrupts Register.

Read prstp - This register is reset when it is read. The three bits reset when this register is read are: parity error bit (bit 5); interrupt request bit (bit 4) and busy error bit (bit 2) in the Bus and Status Register.

6.7 BUS PHASES

This section shows the relevant circuitry active during each of the phases with reference to the subroutines in the NCR.Z80 software (Appendix B).

Here is the sequence of events leading to the phase explanations.

The SCSI bus is reset by asserting, then releasing, $\overline{\text{RST}}$ in the NCR53C80 (IC5) initiator command register.

Class byte and command byte requested, via VDU.

The Selection Phase is entered.

After selecting the hard disk drive, the Command Phase is entered.

After sending the command to the drive the status of the bus is checked by reading the RO652 control port, via the NCR53C80 current SCSI bus status register.

If $\overline{BSY}$ is deasserted, command execution is complete and the program starts again. If $\overline{BSY}$ is asserted, command execution is incomplete and $\overline{REQ}$ is tested. If $\overline{REQ}$ is deasserted the status of the bus is checked again. If $\overline{REQ}$ is asserted, all bits except $\overline{C}$/D , I/O and $\overline{MSG}$ are masked out from the current SCSI bus status register.

These bits are used to determine what phase the bus is in and what subroutine is entered.

$\overline{MSG}$	$\overline{C}$/D	I/O	Phase	Subroutine Entered
R	R	A	Data In Phase	DATAIN
R	R	R	Data Out Phase	SCSIWR
R	A	A	Status Phase	STATUS
A	A	A	Message Phase	MESSAGE

A=asserted, R=deasserted

SELECTION PHASE

During the Selection Phase the RO652 hard disk drive is selected by the SCC, via the NCR53C80 (IC5). The SELECT subroutine of NCR.Z80 controls the selection of the RO652 hard disk as follows.

The SCC sets up IC5 as an initiator with no interrupts and sets the initiator command register, IC1 and IC5 selected.

The SCC resets the IC5 parity/interrupts register, IC2 and IC5 selected.

The address of the RO652 hard disk drive is written from the SCC to the IC5 (RO652 data port), IC1 and IC5 selected.

Information is sent to the IC5 target command register to release C/D and I/O, (data being transferred into RO652), IC1 and IC5 selected.

SEL is asserted by writing to the IC5 initiator command register (RO652 control port), IC1 and IC5 selected.

The RO652 status port, via the IC5 bus status register, is read until BSY is asserted, IC2 and IC5 selected.

SEL is deasserted by writing to the IC5 initiator command register (RO652 control port), IC1 and IC5 selected.

This completes the Selection Phase.

DATA IN PHASE

During the Data In Phase data is sent from the RO652 hard disk drive, via the NCR53C80 (IC5) and the SCC, to the VDU. The DATAIN subroutine of NCR.Z80 controls the transfer of a data byte as follows.

The RO652 data port is read, via the NCR53C80 data register, to transfer the data byte into the SCC, IC2 and IC5 selected.

The SCC then outputs the data byte, which was read from the drive, to the VDU.

ACK is asserted by writing to the IC5 initiator command register (RO652 control port), IC1 and IC5 selected.

The RO652 status port, via the IC5 bus status register, is read until REQ is deasserted, IC2 and IC5 selected.

ACK is then deasserted by writing to the IC5 initiator command register (RO652 control port), IC1 and IC5 selected.

The data byte transfer is complete.

DATA OUT PHASE

During the Data Out Phase data is sent from the VDU, via the SCC and the NCR53C80 (IC5), to the RO652 hard disk drive. The SCSIWR subroutine of NCR.Z80 controls the transfer of a data byte as follows.

The data byte, to be written to the RO652, is read from the VDU into the SCC.

Information is sent to the IC5 target command register to release $\overline{C}/D$ and I/O, (data being transferred into RO652), IC1 and IC5 selected.

The data byte is written to the RO652 by writing to the IC5 data register, IC1 and IC5 selected.

$\overline{ACK}$ is asserted by writing to the IC5 initiator command register (RO652 control port), IC1 and IC5 selected.

The RO652 status port, via the IC5 bus status register, is read until $\overline{REQ}$ is deasserted, IC2 and IC5 selected.

$\overline{ACK}$ is then deasserted by writing to the IC5 initiator command register (RO652 control port), IC1 and IC5 selected.

The data byte transfer is complete.

COMMAND PHASE

During the Command Phase a command byte is sent from the VDU, via the SCC and the NCR53C80 (IC5), to the RO652 hard disk drive. The SEND6 subroutine of NCR.Z80 controls the transfer of a command byte as follows.

The command byte, to be written to the RO652, is read from the VDU into the SCC.

Information is sent to the IC5 target command register to assert C/D and release I/O, (command being transferred into RO652), IC1 and IC5 selected.

The RO652 status port, via the IC5 bus status register, is read until REQ is asserted, IC2 and IC5 selected.

The command byte is written to the RO652 by writing to the IC5 data register, IC1 and IC5 selected.

ACK is asserted by writing to the IC5 initiator command register (RO652 control port), IC1 and IC5 selected.

The RO652 status port, via the IC5 bus status register, is read until REQ is deasserted, IC2 and IC5 selected.

ACK is then deasserted by writing to the IC5 initiator command register (RO652 control port), IC1 and IC5 selected.

The command byte transfer is complete.

STATUS PHASE

During the Status Phase status information is sent from the RO652 hard disk drive, via the NCR53C80 (IC5) and the SCC, to the VDU. The STATUS subroutine of NCR.Z80 controls the transfer of a data byte as follows.

The RO652 data port is read, via the NCR53C80 data register, to transfer the status byte into the SCC, IC2 and IC5 selected.

The SCC then outputs the status byte, which was read from the drive, to the VDU.

$\overline{ACK}$ is asserted by writing to the IC5 initiator command register (RO652 control port), IC1 and IC5 selected.

The RO652 status port, via the IC5 bus status register, is read until $\overline{REQ}$ is deasserted, IC2 and IC5 selected.

$\overline{ACK}$ is then deasserted by writing to the IC5 initiator command register (RO652 control port), IC1 and IC5 selected.

The status byte transfer is complete.

MESSAGE IN PHASE

During the Message In Phase message information is sent from the RO652 hard disk drive, via the NCR53C80 (IC5) and the SCC, to the VDU. The MESSAGE subroutine of NCR.Z80 controls the transfer of a data byte as follows.

The RO652 data port is read, via the NCR53C80 data register, to transfer the message byte into the SCC, IC2 and IC5 selected.

The SCC then outputs the message byte, which was read from the drive, to the VDU.

$\overline{ACK}$ is asserted by writing to the IC5 initiator command register (RO652 control port), IC1 and IC5 selected.

The RO652 status port, via the IC5 bus status register, is read until $\overline{REQ}$ is deasserted, IC2 and IC5 selected.

$\overline{ACK}$ is then deasserted by writing to the IC5 initiator command register (RO652 control port), IC1 and IC5 selected.

The message byte transfer is complete.

6.8 SOFTWARE

NCR.Z80 controls VDU, SCC and RO652 input/output. The program contains:

Equates for VDU control, VDU ASCII characters, NCR53C80 SCSI, NCR53C80 SCSI status/control, SCC RAM storage size.

Messages which are sent to the VDU during operation.

Subroutines which control VDU input/output, baud rate selection and reading from and writing to the VDU.

Subroutines controlling the RO652 hard disk drive phases.

Figure 6.1

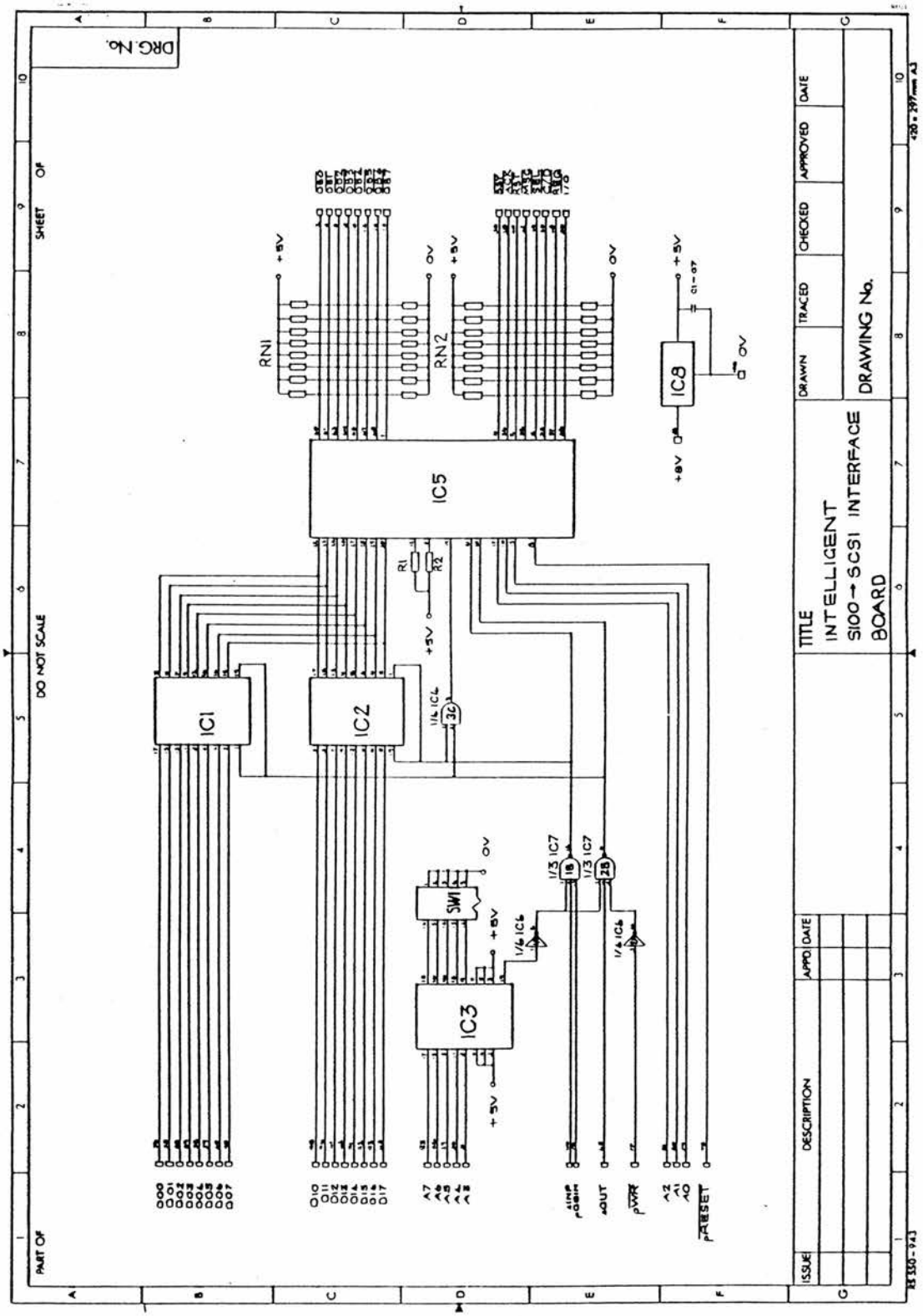
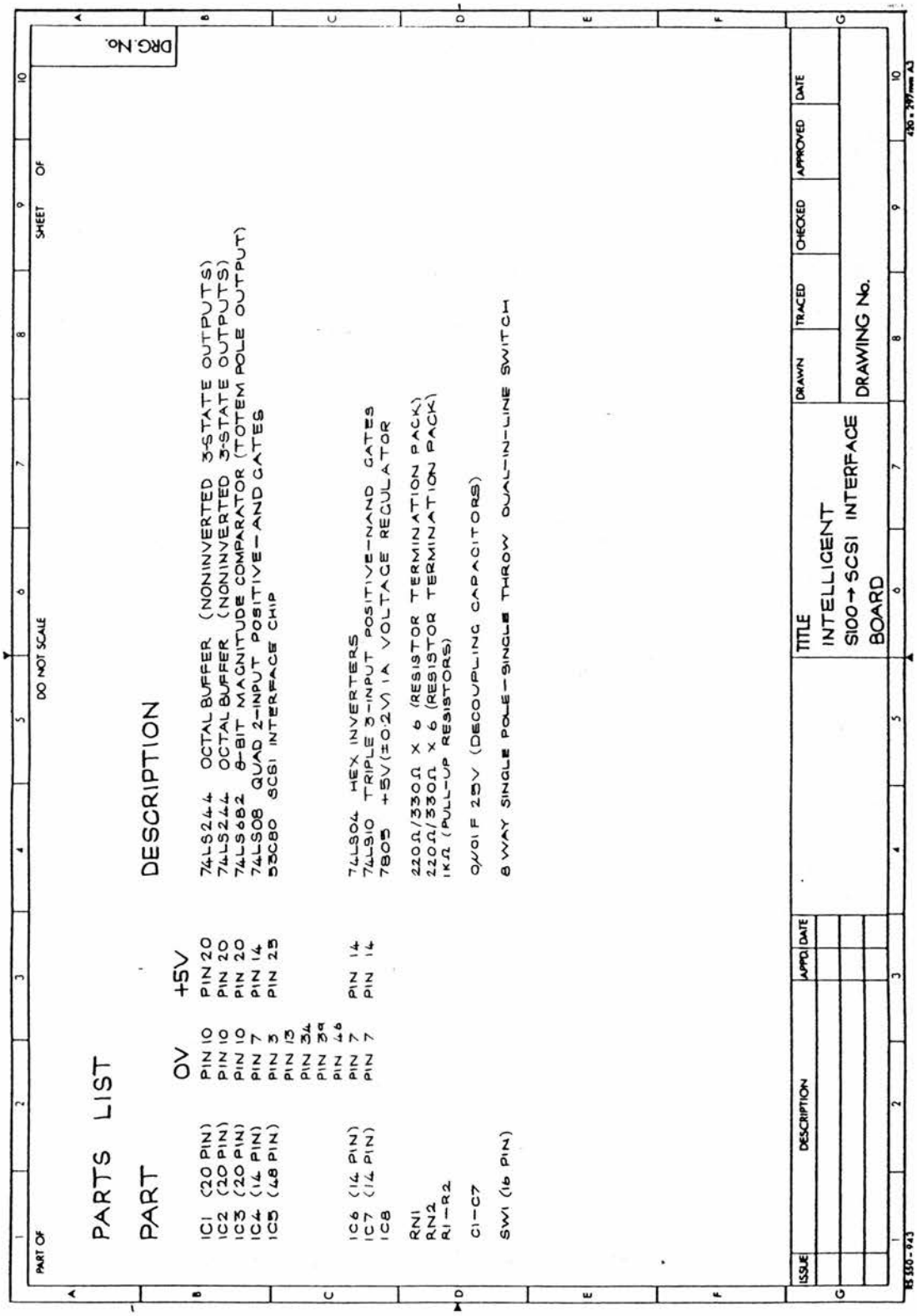

DRG. No.
SHEET OF
DO NOT SCALE
PART OF
RN1
RN2
IC1
IC2
IC3
IC5
IC8
SW1
R1
R2
+5V
0V
+8V
TITLE
INTELLIGENT
SI00→SCSI INTERFACE
BOARD
DRAWING No.
DRAWN
TRACED
CHECKED
APPROVED
DATE
ISSUE
DESCRIPTION
APPD
DATE

Figure 6.2


PART OF
DRG. No.
DO NOT SCALE
SHEET OF
PARTS LIST
PART
0V
+5V
DESCRIPTION
IC1 (20 PIN) PIN 10 PIN 20 74LS244 OCTAL BUFFER (NONINVERTED 3-STATE OUTPUTS)
IC2 (20 PIN) PIN 10 PIN 20 74LS244 OCTAL BUFFER (NONINVERTED 3-STATE OUTPUTS)
IC3 (20 PIN) PIN 10 PIN 20 74LS682 8-BIT MAGNITUDE COMPARATOR (TOTEM POLE OUTPUT)
IC4 (14 PIN) PIN 7 PIN 14 74LS08 QUAD 2-INPUT POSITIVE-AND GATES
IC5 (48 PIN) PIN 3 PIN 25 53C80 SCSI INTERFACE CHIP
PIN 13
PIN 34
PIN 36
PIN 46
IC6 (14 PIN) PIN 7 PIN 14 74LS04 HEX INVERTERS
IC7 (14 PIN) PIN 7 PIN 14 74LS10 TRIPLE 3-INPUT POSITIVE-NAND GATES
IC8 7805 +5V(±0.2V) 1A VOLTAGE REGULATOR
RN1 220Ω/330Ω X 6 (RESISTOR TERMINATION PACK)
RN2 220Ω/330Ω X 6 (RESISTOR TERMINATION PACK)
R1-R2 1KΩ (PULL-UP RESISTORS)
C1-C7 0.01 F 25V (DECOUPLING CAPACITORS)
SW1 (16 PIN) 8 WAY SINGLE POLE-SINGLE THROW DUAL-IN-LINE SWITCH
ISSUE
DESCRIPTION
APPD
DATE
TITLE
INTELLIGENT
SI00→SCSI INTERFACE
BOARD
DRAWN
TRACED
CHECKED
APPROVED
DATE
DRAWING No.

CHAPTER 7

USES OF THE SCSI SYSTEM

The SCSI system, as described in this thesis, can be used to exercise any peripheral device eg. printers, hard disk drives, tape streamers etc. which support the Small Computer System Interface (SCSI). It is a very useful tool for service engineers, because it can communicate directly with a wide range of peripheral devices and is manufacturer independent. It can be used to determine whether a peripheral or a host computer is the cause of failure in a non-working computer system.

The SCSI system can be used to exercise a hard disk drive, for example, by seeking, writing and reading from specified tracks. The SCSI system can also be used to change the format parameters of a hard disk drive (sectors per track, bytes per sector, interleave etc.) and then low level format the drive using these new parameters.

Custom software can be written for the SCSI system and programmed into the SCC resident EPROM. This software can be used to exercise any SCSI device, with or without user input. Some uses of such software are:

Quality Control. Manufacturers can use the SCSI system as part of a "quality control" system to test their SCSI peripherals before they are sent out to customers.

Goods In. OEMs can use the SCSI system as part of a "goods in" test to check out incoming SCSI peripherals before they are used as part of computer systems.

7.1 EXAMPLE FUNCTIONS

The following examples are functions which cannot normally be carried out from a standard operating system. All have been carried out using the SCSI system, which consists of the following:

S-100 bus system (motherboard and power supply)
VDU
Cromemco Single Card Computer (SCC)
Intelligent SCSI adapter card (See Chapter 6)
RO652 hard disk drive

EXAMPLE 1 : EXAMINE DRIVE FORMAT PARAMETERS

When the SCSI system is powered up the prompt for the Class of command appears on the VDU. The class (0) and the command (1A) of the Mode Sense op-code are entered, then the page code for the drive format parameters (03). The allocate length is then prompted for, this specifies the number of bytes that has been allocated for the returned data. The allocate length is set to the maximum value (FF), as the drive will terminate the command when all data has been transferred. When this command is issued, the hard disk drive will respond by returning the bytes shown. Then the status byte and message byte are returned, indicating command completion. The following shows the sequence of entries, with user responses in bold type.

SCSI Tester V 01.02 July 1986
Class: **0**
Command: **1A**

Page Code: **03**
Allocate length: **FF**

18 00 00 00 03 13 01 32 01 32 00 06 00 06 00 22 02 00 00 02 00 00 00 00 80

Status: 00 Message: 00

The 25 bytes returned by the mode sense command are split into two different parts as shown below, the far left column shows the returned value:

Mode sense header (4 bytes)
Format parameters page (21 bytes)

Mode Sense Header (4 bytes)

	Bit	7	6	5	4	3	2	1	0
18	Byte 0	Sense Data Length							
00	Byte 1	00H							
00	Byte 2	00H							
00	Byte 3	00H							

Sense Data Length — This is the number of bytes, in hexadecimal, returned from the drive during the data in phase. The sense data length does not include itself.

21 Byte Format Parameters Page

	Bit	7	6	5	4	3	2	1	0
03	Byte 0	SDP	0	0	0	0	0	1	1
13	Byte 1	Page Length (00...13H)							
01	Byte 2	Tracks per Zone MSB							
32	Byte 3	Tracks per Zone LSB							
01	Byte 4	Alt Sectors per Zone MSB							
32	Byte 5	Alt Sectors per Zone LSB							
00	Byte 6	Alt Tracks per Zone MSB							
06	Byte 7	Alt Tracks per Zone LSB							
00	Byte 8	Alt Tracks per Volume MSB							
06	Byte 9	Alt Tracks per Volume LSB							
00	Byte 10	00H							
22	Byte 11	Sectors per Track							
02	Byte 12	Bytes per Sector MSB							
00	Byte 13	Bytes per Sector LSB							
00	Byte 14	00H							
02	Byte 15	Interleave							
00	Byte 16	00H							
00	Byte 17	00H							
00	Byte 18	00H							
00	Byte 19	00H							
80	Byte 20	80H							

SDP — The SDP bit allows the user to Set Default Parameters. If this bit is set the user implements the factory defined default parameters for this page.

The bytes returned translate as follows:

The Page Length is 13H, 19 decimal.
The number of Tracks per Zone is 132H, 306 decimal.
The number of Alternate Sectors per Zone is 132H, 306 decimal.
The number of Alternate Tracks per Zone is 6.
The number of Alternate Tracks per Volume is 6.
The number of Sectors per Track is 22H, 34 decimal.
The number of Bytes per Sector is 200H, 512 decimal.
The Interleave is 2.

EXAMPLE 2 : LOW-LEVEL FORMAT

When the SCSI system is powered up the prompt for the Class of command appears on the VDU. The class (0) and the command (04) of the Format op-code are entered, then the format control code (00). The interleave is then prompted for, which is set to 03. When this command is issued, the hard disk drive will execute a low-level format. Then the status byte and message byte are returned, indicating command completion. The following shows the sequence of entries, with user responses in bold type.

SCSI Tester V 01.02 July 1986
Class: **0**
Command: **04**

Format Control: **00**
Interleave: **03**

Status: 00 Message: 00

With the format control code set to 00, the drive is formatted using both Primary Defect List (P-LIST) which was written on the drive during manufacture and Growing Defect List (G-LIST) which the user writes to the drive.

The Interleave has been changed from 02, as shown in the previous example to be 03. If the drive format parameters are re-examined after the format is complete, byte 15 of the format parameters page will be 03.

REFERENCES

Draft Proposed American National Standard for Information Systems - Small Computer System Interface (SCSI) X3T9.2 Revision 17

American National Standard Common Command Set (CCS) of the Small Computer System Interface (SCSI) X3T9.2 Revision 4

RODIME 650 Series User Maunual (Product Number: USM 00090)

CROMEMCO Single Card Computer (SCC) Instruction Manual

ZILOG Z80A CPU Data Sheets

The S-100 Bus Handbook, Dave Bursky 1980

TEXAS INTRUMENTS TTL Data Book for Design Engineers Volume 1

NCR53C80 SCSI Interface Chip Design Manual

Programming the Z80, Rodnay Zaks 1980

APPENDIX A

TTL LOGIC INTERFACE BOARD SOFTWARE

A.1 MAIN.Z80

```
; Development program SCSI device tester

*include scsiequ.z80
*include scsimsg.z80
*include scsivdu.z80
*include scsiscsi.z80

; Main Program start

start   ld      a,rst           ; reset drive
        out     contp,a
        ld      a,00h
        out     contp,a
        ld      hl,stack        ; set up stack pointer register
        ld      sp,hl
        call    getbaud         ; set the baud rate
        jp      prog

prog    ld      hl,stack        ; set up stack pointer register
        ld      sp,hl
        ld      hl,banner1      ; output Class request to VDU
        call    outst
        ld      b,2             ; one byte only
        call    inhex
        ld      hl,inhexs + 1   ; hl -> second byte
        ld      a,(hl)          ; test only one byte
        cp      null
        jr      nz,cmderr
        dec     hl
        ld      a,(hl)          ; get the byte
        push    af              ; save Class
        ld      hl,msgcmd
        call    outst           ; ask for command hex bytes
        ld      b,3             ; possibly two bytes
        call    inhex
        ld      hl,inhexs       ; hl -> first byte
        ld      a,(hl)          ; get first byte
        and     0fh
        ld      e,a             ; keep it in e
        inc     hl
        ld      a,(hl)
        cp      null
        jr      z,cmdok         ; only one byte
        and     0fh
        ld      d,a             ; save second byte
        ld      a,e             ; get back first byte
        cp      1               ; check the value is one
        jr      nz,cmderr
        ld      a,10h
        add     a,d             ; form command
        ld      e,a             ; keep it in e
        inc     hl
        ld      a,(hl)
        cp      null
        jr      nz,cmderr       ; only two bytes allowed
cmdok   pop     af              ; restore Class
        cp      '0'             ; test valid Class
        jr      z,class0
        cp      '1'
        jp      z,class1
```

```
            cp      '7'
            jp      z,class7
cmderr      ld      hl,msgwcl       ; command error exit
            call    outst
            jr      prog
```

; Class-0 commands

```
class0      ld      a,e
            ld      hl,buff         ; hl -> command buffer
            ld      (hl),a          ; byte-0
            inc     hl
            cp      00h             ; test drive ready
            jr      z,null5
            cp      01h             ; recalibrate
            jr      z,null5
            cp      03h             ; request sense
            jr      z,sens5
            cp      04h             ; format unit
            jr      z,form5
            cp      07h             ; reassign blocks
            jp      z,reas5
            cp      08h             ; read
            jp      z,read5
            cp      0ah             ; write
            jp      z,writ5
            cp      0bh             ; seek
            jp      z,seek5
            cp      12h             ; inquiry
            jp      z,inqy5
            cp      15h             ; mode select
            jp      z,msel5
            cp      1ah             ; mode sense
            jp      z,msen5
            cp      1dh             ; send diagnostics
            jp      z,diag5
            jp      prog
```

; Send a null byte into command buffer and increment pointer

```
zero        ld      (hl),null
            inc     hl
            ret
```

; Get a hex byte and convert to binary byte

```
gethex2     call    outst
            ld      b,3             ; maximum of two bytes
            call    inhex
            call    binary
            ret
```

; Byte-1 followed by 5 null bytes
; used by cmd000 and cmd001

```
null5       ld      b,5
null5a      call    zero            ; byte-1 to byte-5 are null
            djnz    null5a
            jp      cl0send
```

```
; Byte-1 followed by 3 nulls, allocation length (18), 1 null
; used by cmd003

sens5     ld    b,3
sens5a    call  zero          ; byte-1 to byte-3 are null
          djnz  sens5a
          ld    (hl),12h      ; allocation length
          inc   hl
          ld    (hl),null
          call  select
          ld    hl,buff
          call  send6
          ld    hl,msgsens
          call  outst         ; sense byte banner
          call  scsird        ; read data bytes and output to VDU
          call  status
          call  message
          jp    prog

; Byte-0 followed by format control code, 2 null bytes, interleave, 1
; null byte used by cmd004

form5     push  hl            ; save pointer
          ld    hl,msgfmc     ; format control requested
          call  gethex2
          pop   hl            ; restore pointer
          ld    (hl),a        ; byte-1
          inc   hl
          ld    b,2
form5a    call  zero          ; byte-2 and byte-3 are null
          djnz  form5a
          push  hl            ; save pointer
          ld    hl,msgfmi     ; format interleave requested
          call  gethex2
          pop   hl            ; restore pointer
          ld    (hl),a        ; byte-4
          inc   hl
          ld    (hl),null     ; byte-5 null
          call  select
          ld    hl,buff
          call  send6
          call  scsiwr
          call  status
          call  message
          jp    prog

; Byte-1 followed by 5 null bytes
; used by cmd007

reas5     ld    b,5
reas5a    call  zero          ; byte-1 to byte-5 are null
          djnz  reas5a
          call  select
          ld    hl,buff
          call  send6
          call  scsiwr
          call  status
          call  message
          jp    prog
```

```
; Byte-0 followed by track address and one null byte
; used by cmd008

read5   call    setup5
        call    select
        ld      hl,buff
        call    send6
        call    scsird
        call    status
        call    message
        jp      prog

; Byte-0 followed by track address and one null byte
; requests write byte value once only
; used by cmd00a

writ5   call    setup5
        call    select
        ld      hl,buff
        call    send6
        call    scsiwb
        call    status
        call    message
        jp      prog

; Byte-0 followed by track address and one null byte
; used by cmd00b

seek5   call    setup5
        jp      cl0send

; setup5 - provides byte-1 to byte-5 for read5, writ5 and seek5
; setup5a       byte-4 and byte5 for frd5 and fwr5

setup5  push    hl              ; save pointer
        ld      hl,msgtckh      ; high track address requested
        call    gethex2
        pop     hl              ; restore pointer
        ld      (hl),a          ; byte-1
        inc     hl
        push    hl              ; save pointer
        ld      hl,msgtckm      ; middle track address requested
        call    gethex2
        pop     hl              ; restore pointer
        ld      (hl),a          ; byte-2
        inc     hl
        push    hl              ; save pointer
        ld      hl,msgtckl      ; low track address requested
        call    gethex2
        pop     hl              ; restore pointer
        ld      (hl),a          ; byte-3
        inc     hl
setup5a push    hl              ; save pointer
        ld      hl,msgbcnt      ; block count
        call    gethex2
        pop     hl              ; restore pointer
        ld      (hl),a          ; byte-4
        inc     hl
        ld      (hl),null       ; byte-5 null
        ret
```

```
; Byte-0 followed by 3 null bytes, allocation length, 1 null byte
; used by cmd012

inqy5     ld    b,3
inqy5a    call  zero          ; bytes 1to 3 are null
          djnz  inqy5a
          push  hl            ; save pointer
          ld    hl,msgalen    ; allocation length requested
          call  gethex2
          pop   hl            ; restore pointer
          ld    (hl),a        ; byte-4 is allocation length
          inc   hl
          ld    (hl),null     ; byte-5 null
          call  select
          ld    hl,buff
          call  send6
          call  scsird
          call  status
          call  message
          jp    prog

; Byte-0 followed by 3 null bytes, parameter list length, 1 null byte
; used by cmd015

msel5     ld    b,3
msel5a    call  zero          ; bytes 1 to 3 are null
          djnz  msel5a
          push  hl            ; save pointer
          ld    hl,msgplst    ; allocation length requested
          call  gethex2
          pop   hl            ; restore pointer
          ld    (hl),a        ; byte-4 is parameter list length
          inc   hl
          ld    (hl),null     ; byte-5 null
          call  select
          ld    hl,buff
          call  send6
          call  scsiwr
          call  status
          call  message
          jp    prog

; Byte-0 followed by null, page code, null, allocation length, null
; used by cmd01a

msen5     call  zero          ; byte-1
          push  hl            ; save pointer
          ld    hl,msgpcod    ; page code requested
          call  gethex2
          pop   hl            ; restore pointer
          ld    (hl),a        ; byte-2 is page code
          inc   hl
          call  zero          ; byte-3 is null
          push  hl            ; save pointer
          ld    hl,msgalen    ; allocation length requested
          call  gethex2
          pop   hl            ; restore pointer
          ld    (hl),a        ; byte-4 is allocation length
          inc   hl
          ld    (hl),null     ; byte-5 null
          call  select
          ld    hl,buff
```

```
            call    send6
            call    scsird
            call    status
            call    message
            jp      prog

; Byte-1 followed by 04h followed by 4 null bytes
; cmd01d

diag5       ld      (hl),04h        ; byte-1
            inc     hl
            ld      b,4
diag5a      call    zero            ; byte-2 to byte-5 are null
            djnz    diag5a
            jp      cl0send
```

; Class-1 commands

```
class1      ld      a,e
            add     20h             ; add class-1 for byte-0
            ld      hl,buff         ; hl -> command buffer
            ld      (hl),a          ; byte-0
            inc     hl
            call    zero            ; byte-1
            cp      25h             ; read capacity
            jp      z,capy9
            cp      28h             ; read extended
            jp      z,rdex9
            cp      2ah             ; write extended
            jp      z,wrex9
            cp      2fh             ; verify
            jp      z,verf9
            cp      37h             ; read defect data
            jp      z,deft9
            cp      3ch             ; read data buffer
            jp      z,rddb9
            cp      3bh             ; write data buffer
            jp      z,wrdb9
            jp      prog

; Byte-0 and byte-1 followed by 4 byte logical block address, 2 null bytes
; PMI byte and null byte
; used by cmd105

capy9       push    hl              ; save pointer
            ld      hl,msglba1
            call    gethex2
            pop     hl              ; restore pointer
            ld      (hl),a          ; byte-2 is LBA high byte
            inc     hl
            push    hl              ; save pointer
            ld      hl,msglba2
            call    gethex2
            pop     hl              ; restore pointer
            ld      (hl),a          ; byte-3 is LBA byte 2
            inc     hl
            push    hl              ; save pointer
            ld      hl,msglba3
            call    gethex2
            pop     hl              ; restore pointer
            ld      (hl),a          ; byte-4 is LBA byte 3
```

```
        inc     hl
        push    hl              ; save pointer
        ld      hl,msglba4
        call    gethex2
        pop     hl              ; restore pointer
        ld      (hl),a          ; byte-5 is LBA low byte
        inc     hl
        call    zero            ; byte-6 null
        call    zero            ; byte-7 null
        push    hl              ; save pointer
        ld      hl,msgpmi
        call    gethex2
        pop     hl              ; restore pointer
        ld      (hl),a          ; byte-8 is PMI
        inc     hl
        ld      (hl),null       ; byte-9
        call    select
        ld      hl,buff
        call    send10
        call    scsird
        call    status
        call    message
        jp      prog

; Byte-0 and byte-1 followed by 2 nulls,3 byte logical block address,
; 1 null, 2 byte block count and null byte
; used by cmd108, cmd10a and cmd10f

rdex9
wrex9
verf9   call    zero            ; byte-2
        push    hl              ; save pointer
        ld      hl,msglba1
        call    gethex2
        pop     hl              ; restore pointer
        ld      (hl),a          ; byte-3 is LBA high byte
        inc     hl
        push    hl              ; save pointer
        ld      hl,msglba2
        call    gethex2
        pop     hl              ; restore pointer
        ld      (hl),a          ; byte-4 is LBA byte 2
        inc     hl
        push    hl              ; save pointer
        ld      hl,msglba3
        call    gethex2
        pop     hl              ; restore pointer
        ld      (hl),a          ; byte-5 is LBA low byte
        inc     hl
        call    zero            ; byte-6 null
        push    hl              ; save pointer
        ld      hl,msgbcnt
        call    gethex2
        pop     hl              ; restore pointer
        ld      (hl),a          ; byte-7 is block count MSB
        inc     hl
        push    hl              ; save pointer
        ld      hl,msgbct2
        call    gethex2
        pop     hl              ; restore pointer
        ld      (hl),a          ; byte-8 is block count LSB
        inc     hl
```

```
                ld      (hl),null       ; byte-9
                call    select
                ld      hl,buff
                call    send10
                call    scsird
                ld      hl,buff
                ld      a,2ah
                cp      (hl)
                jr      nz,rdex9a
                call    scsiwb
rdex9a          call    status
                call    message
                jp      prog

; Byte-0 and byte-1 followed by P/G/CODE byte, 4 nulls,
; 2 byte allocation length and null byte
; used by cmd117

deft9           push    hl              ; save pointer
                ld      hl,msgdeft
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-2 is P/G/Code
                inc     hl
                ld      b,4
deft9a          call    zero            ; bytes 3 to 6 are null
                djnz    deft9a
                push    hl              ; save pointer
                ld      hl,msgalen
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-7 is MSB allocation length
                inc     hl
                push    hl              ; save pointer
                ld      hl,msgaln2
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-8 is LSB allocation length
                inc     hl
                ld      (hl),null       ; byte-9
                call    select
                ld      hl,buff
                call    send10
                call    scsird
                call    status
                call    message
                jp      prog

; Byte-0 and byte-1 followed by 6 nulls,
; 2 byte allocation length and null byte
; used by cmd117 and cmd11c

rddb9
wrdb9           ld      b,6
rddb9a          call    zero            ; bytes 3 to 6 are null
                djnz    rddb9a
                push    hl              ; save pointer
                ld      hl,msgalen
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-7 is MSB allocation length
                inc     hl
```

```
            push    hl              ; save pointer
            ld      hl,msgaln2
            call    gethex2
            pop     hl              ; restore pointer
            ld      (hl),a          ; byte-8 is LSB allocation length
            inc     hl
            ld      (hl),null       ; byte-9
            call    select
            ld      hl,buff
            call    send10
            call    scsird
            ld      hl,buff
            ld      a,3bh
            cp      (hl)
            jr      nz,rddb9b
            call    scsiwb
rddb9b      call    status
            call    message
            jp      prog
```

; Class-7 commands

```
class7      ld      a,e
            add     0e0h            ; add class-7 for byte-0
            ld      hl,buff         ; hl -> command buffer
            ld      (hl),a          ; byte-0 to command buffer
            inc     hl
            cp      0e0h            ; maintenance seek
            jp      z,mseek5
            cp      0e1h            ; format maintenance tracks
            jp      z,mform5
            cp      0e2h            ; certify
            jp      z,cert5
            cp      0e8h            ; fast read
            jp      z,frd5
            cp      0eah            ; fast write
            jp      z,fwr5
            jp      prog

; Byte-0 followed by 1 null, 2 byte cylinder number, 1 byte head number
; and null byte
; used by cmd700

mseek5      call    zero            ; byte-1
            push    hl              ; save pointer
            ld      hl,msgcylh
            call    gethex2
            pop     hl              ; restore pointer
            ld      (hl),a          ; byte-2 is cylinder number high byte
            inc     hl
            push    hl              ; save pointer
            ld      hl,msgcyll
            call    gethex2
            pop     hl              ; restore pointer
            ld      (hl),a          ; byte-3 is cylinder number low byte
            inc     hl
            push    hl              ; save pointer
            ld      hl,msghead
            call    gethex2
            pop     hl              ; restore pointer
            ld      (hl),a          ; byte-4 is head number
```

```
            inc     hl
            ld      (hl),null           ; byte-5 null
            jr      cl7send

; Set pattern of bytes
; used by cmd701

mform5      ld      (hl),00000000b      ; byte-1
            inc     hl
            ld      (hl),01010010b      ; byte-2
            inc     hl
            ld      (hl),01001111b      ; byte-3
            inc     hl
            ld      (hl),00000000b      ; byte-4
            inc     hl
            ld      (hl),00000000b      ; byte-5 null
            jr      cl7send

; Byte-0 followed by 2 null bytes, pass count, 2 null bytes
; used by cmd702

cert5       call    zero                ; byte-1
            call    zero                ; byte-2 is null
            push    hl                  ; save pointer
            ld      hl,msgpcnt          ; allocation length requested
            call    gethex2
            pop     hl                  ; restore pointer
            ld      (hl),a              ; byte-3 is pass count
            inc     hl
            call    zero                ; byte-4 null
            ld      (hl),null           ; byte-5 null
            jr      cl7send

; Byte-0 followed by 3 null bytes, block count, 1 null byte
; used by cmd708 and cmd70a

frd5
fwr5        ld      b,3
fwr5a       call    zero                ; bytes 1 to 3 are null
            djnz    fwr5a
            call    setup5a
            jr      cl7send

; Class-7 send control bytes

cl0send
cl7send     call    select
            ld      hl,buff
            call    send6
            call    status
            call    message
            jp      prog

            end     start

; end of file
```

A.2 EQU.Z80

```
; Development program SCSI device tester

; General equates

; VDU equates
vdustat     equ     00h             ; VDU status register
vdubaud     equ     00h             ; VDU baud rate register
vdudata     equ     01h             ; VDU data register
vducmd      equ     02h             ; VDU command register
rda         equ     6               ; RDA bit
tbe         equ     7               ; TBE bit

; ASCII equates
null        equ     00h
cr          equ     0dh
lf          equ     0ah

; SCSI equates
datap       equ     20h
contp       equ     21h
statp       equ     21h

; Status bit positions
bsy         equ     0
msg         equ     1
cd          equ     2
req         equ     3
io          equ     4

; Control values
ack         equ     1
rst         equ     2
sel         equ     4

; Storage allocation in SCC RAM
; 2000h to 23ffh
stack       equ     2100h           ; reserved for stack
inhexs      equ     2200h           ; ASCII bytes and trap for last byte
buff        equ     2220h           ; working buffer
buff1       equ     2240h           ; working buffer

; end of file
```

A.3 MSG.Z80

```
; Development program SCSI device tester

banner1   db    cr,lf,lf,'SCSI Tester V 00.02 June 1986',cr,lf
          db    'Class: ',null
crlf      db    cr,lf,null
msgalen   db    cr,lf,'Allocate length 1: ',null
msgaln2   db    cr,lf,'Allocate length 2: ',null
msgbcnt   db    cr,lf,'Block Count 1: ',null
msgbct2   db    cr,lf,'Block Count 2: ',null
msgcmd    db    cr,lf,'Command: ',null
msgcylh   db    cr,lf,'Cylinder 1: ',null
msgcyll   db    cr,lf,'Cylinder 2: ',null
msgdeft   db    cr,lf,'P/G/Code: ',null
msgfmc    db    cr,lf,'Format Control: ',null
msgfmi    db    cr,lf,'Interleave: ',null
msghead   db    cr,lf,'Head: ',null
msghexe   db    cr,lf,'? HEX ?',cr,lf,lf,null
msglba1   db    cr,lf,'LBA 1: ',null
msglba2   db    cr,lf,'LBA 2: ',null
msglba3   db    cr,lf,'LBA 3: ',null
msglba4   db    cr,lf,'LBA 4: ',null
msgmsg    db    ' Message: ',null
msgnot    db    cr,lf,'????',null
msgpcnt   db    cr,lf,'Pass Count: ',null
msgpcod   db    cr,lf,'Page Code: ',null
msgpmi    db    cr,lf,'PMI: ',null
msgplst   db    cr,lf,'Parameter List Length: ',null
msgsens   db    cr,lf,'Report:',cr,lf,null
msgspac   db    ' ',null
msgstat   db    cr,lf,'Status: ',null
msgtckh   db    cr,lf,'Address 1: ',null
msgtckm   db    cr,lf,'Address 2: ',null
msgtckl   db    cr,lf,'Address 3: ',null
msgwcl    db    cr,lf,'INVALID',null

; end of file
```

A.4 VDU.Z80

```
; Development program SCSI device tester

; VDU I/0 routines

; Output a null terminated string
; hl -> string
; preserves all registers

outst   push  af          ; save registers
        push  hl
outst1  call  tbemp
        ld    a,(hl)      ; get the byte
        cp    null
        jr    z,outst2    ; a null, so all done
        out   vdudata,a
        inc   hl          ; point to next byte
        jr    outst1
outst2  pop   hl          ; restore registers
        pop   af
        ret

; Wait for transmitter bufer to empty
; preserves all registers

tbemp   push  af
tbemp1  in    a,vdustat   ; wait for TBE
        bit   tbe,a
        jr    z,tbemp1
        pop   af
        ret

; Wait for read data available and read byte into a
; preserves all registers except af

gbyte   in    a,vdustat   ; wait for a digit available
        bit   rda,a
        jr    z,gbyte
        in    a,vdudata
        ret

; Output a null terminated byte to VDU followed by a space
; (a) = byte
; preserves all registers

byteout push  af          ; output hex byte to VDU
        and   0f0h
        rrca
        rrca
        rrca
        rrca
        cp    0ah         ; test for hex adjustment
        jr    c,byteou1
        add   a,7
byteou1 add   30h
        call  tbemp
        out   vdudata,a
        call  tbemp
        pop   af
        push  af
        and   0fh
```

```
                cp      0ah             ; test for hex adjustment
                jr      c,byteou2
                add     a,7
byteou2         add     30h
                out     vdudata,a
                call    tbemp
                ld      a,' '           ; output a space
                out     vdudata,a
                pop     af
                ret

; Convert one/two hex values (high/low) to binary
; high hex is stored in inhexs
; (a) = binary byte
; preserves all registers except af

binary          push    hl
                ld      hl,inhexs + 1
                ld      a,(hl)          ; if null only one byte input
                cp      null
                jp      nz,binary1
                dec     hl
                ld      a,(hl)          ; get byte
                and     0fh
                ld      e,a             ; save hex value in e
                jr      binary2
binary1         dec     hl              ; form both nibbles
                rld
                ld      a,(hl)
                ld      e,a             ; save hex value in e
binary2         ld      a,e             ; get byte and return it in a
                pop     hl
                ret

; Input a hex string of numbers up to inhexm long terminated with cr
; input is stored in inhexs
; on input b = maximum number of hex digits required
; preserves all registers

inhexm          equ     0ah
inhex           push    af              ; save registers
                push    bc
                push    hl
                ld      a,inhexm -2
                cp      b
                jr      c,inhexe
                ld      hl,inhexs
                ld      b,inhexm        ; load count
inhex0          ld      (hl),null       ; nulls to storage area
                inc     hl
                djnz    inhex0
                ld      hl,inhexs       ; get back pointer
                ld      b,inhexm        ; load count
inhex1          call    gbyte           ; get a byte from VDU
                and     7fh             ; strip parity bit
                cp      cr              ; is number finished
                jr      z,inhex2
                ld      (hl),a
                call    outst           ; echo to VDU
                cp      '0'
                jr      c,inhexe        ; abandon if not hex
                cp      '9'+1
```

```
                jr      c,inhex3
                cp      'a'
                jr      c,inhex4
                sub     a,20h
inhex4          cp      'A'
                jr      c,inhexe        ; abandon if not hex
                cp      'F'+1
                jr      nc,inhexe       ; abandon if not hex
                sub     a,7             ; adjust byte
inhex3          ld      (hl),a          ; store digit
                inc     hl              ; increment pointer
                djnz    inhex1
                jr      inhexe          ; number too long
inhex2          ld      a,inhexm
                sub     b
                ld      b,a             ; b = no of hex digits
                cp      null
                jr      z,inhexe        ; abandon as no input
                pop     hl
                pop     bc
                pop     af
                ret
inhexe          ld      hl,msghexe      ; point to error message
                call    outst
                jp      prog

; Baud rate selection routine

baudrs          db      10010000b
                db      11000000b       ; 9600
                db      10100000b       ; 4800
                db      10010000b       ; 2400
                db      10001000b       ; 1200
                db      10000100b       ;  300
                db      10000010b       ;  150
                db      00000001b       ;  110

getbaud         ld      hl,baudrs       ; hl -> baud rate table
                ld      c,vdubaud       ; c = baud port
                ld      a,11h           ; reset bit and high baud
baud1           out     vducmd,a        ; reset
                outi
                call    gbyte
                call    gbyte
                and     7fh
                cp      0dh             ; recognise a cr?
                ld      a,1             ; reset bit only
                jr      nz,baud1
                ret

; end of file
```

A.5 SCSI.Z80

```
; Development program SCSI device tester

; SCSI Subroutines

; Read status byte from drive and output to VDU
; preserves all registers

status      push  af
            push  hl
status0     in    a,statp          ; test for status byte
            cp    0fdh
            jr    nz,status0
            ld    hl,msgstat       ; hl -> banner
            call  outst
            in    a,datap          ; read drive status byte
            call  byteout
            ld    a,ack            ; acknowledge drive
            out   contp,a
status1     in    a,statp          ; wait for request false
            bit   req,a
            jr    nz,status1
            ld    a,00h            ; reset acknowledge false
            out   contp,a
            pop   hl
            pop   af
            ret

; Read message byte from drive and output to VDU
; preserves all registers

message     push  af
            push  hl
messag0     in    a,statp          ; test for status byte
            cp    0ffh
            jr    nz,messag0
            ld    hl,msgmsg        ; hl -> banner
            call  outst
            in    a,datap          ; read drive status byte
            call  byteout
            ld    a,ack            ; acknowledge drive
            out   contp,a
messag1     in    a,statp          ; wait for request false
            bit   req,a
            jr    nz,messag1
            ld    a,00h            ; put acknowledge false
            out   contp,a
            pop   hl
            pop   af
            ret

; Read data bytes from drive and output to VDU
; preserves all registers

scsird      push  af
scsird0     in    a,statp          ; test for request
            bit   req,a
            jr    z,scsird0
            cp    0f9h             ; is it read in data?
            jr    nz,scsird2       ; return as all done
            in    a,datap          ; read data byte
```

```
                call    byteout
                ld      a,ack           ; acknowledge drive
                out     contp,a
scsird1         in      a,statp         ; wait for request false
                bit     req,a
                jr      nz,scsird1
                ld      a,00h           ; put acknowledge false
                out     contp,a
                jr      scsird0         ; look for next data in byte
scsird2         pop     af
                ret

; Write data bytes to drive from VDU
; preserves all registers

scsiwrb         db      cr,lf,'B: ',null
scsiwr          push    af
                push    hl
scsiwr0         in      a,statp         ; test for request
                bit     req,a
                jr      z,scsiwr0
                cp      0e9h            ; is it requesting data?
                jr      nz,scsiwr2      ; return as all done
                ld      hl,scsiwrb
                call    outst
                ld      hl,inhexs
                call    inhex
                call    binary
                out     datap,a         ; write data byte
                ld      a,ack           ; acknowledge drive
                out     contp,a
scsiwr1         in      a,statp         ; wait for request false
                bit     req,a
                jr      nz,scsiwr1
                ld      a,00h           ; put acknowledge false
                out     contp,a
                jr      scsiwr0         ; look for next data in byte
scsiwr2         pop     hl
                pop     af
                ret

; Write block of bytes to drive from VDU
; preserves all registers

scsiwb          push    af
                push    hl
                ld      hl,scsiwrb      ; ask for byte
                call    outst
                ld      hl,inhexs
                call    inhex
                call    binary
                ld      h,a             ; save the byte
scsiwb0         in      a,statp         ; test for request
                bit     req,a
                jr      z,scsiwb0
                cp      0e9h            ; is it requesting data?
                jr      nz,scsiwb2      ; return as all done
                ld      a,h             ; get the byte back
                out     datap,a         ; write data byte
                ld      a,ack           ; acknowledge drive
                out     contp,a
scsiwb1         in      a,statp         ; wait for request false
```

```
                bit     req,a
                jr      nz,scsiwb1
                ld      a,00h           ; put acknowledge false
                out     contp,a
                jr      scsiwb0         ; look for next data in byte
scsiwb2         pop     hl
                pop     af
                ret

; Select drive
; preserves all registers

select          push    af
                ld      a,00000001b     ; select drive
                out     datap,a
                ld      a,sel
                out     contp,a
select1         in      a,statp
                bit     bsy,a
                jr      z,select1       ; wait for bsy true
                ld      a,00h
                out     contp,a         ; reset select false
                pop     af
                ret

; Send ten or six control bytes to drive
; hl -> buffer containing the six bytes
; preserves all registers

send6           push    af
                push    bc
                push    hl
                ld      c,datap
                ld      b,6
                jr      send6a
send10          push    af
                push    bc
                push    hl
                ld      c,datap
                ld      b,10
send6a          in      a,statp         ; wait for request
                cp      0edh
                jr      nz,send6a
                outi                    ; send byte
                ld      a,ack           ; acknowledge byte ready
                out     contp,a
send6b          in      a,statp         ; wait for drive to read byte
                cp      0e5h
                jr      nz,send6b
                ld      a,0             ; set acknowledge false
                out     contp,a
                cp      b
                jr      nz,send6a
                ld      hl,crlf
                call    outst           ; cr/lf to tidy display
                pop     hl
                pop     bc
                pop     af
                ret

; end of file
```

APPENDIX B

INTELLIGENT INTERFACE BOARD SOFTWARE

B.1 NCR.Z80

B.1 NCR.Z80

```
; Development program SCSI device tester

; ********Equates********

; VDU equates

vdustat     equ   00h          ; VDU status register
vdubaud     equ   00h          ; VDU baud rate register
vdudata     equ   01h          ; VDU data register
vducmd      equ   02h          ; VDU command register
rda         equ   6            ; RDA bit
tbe         equ   7            ; TBE bit

; ASCII equates

null        equ   00h
cr          equ   0dh
lf          equ   0ah

; NCR53C80 SCSI equates

base        equ   20h          ; Base port
datap       equ   base + 0     ; data port
icmdp       equ   base + 1     ; initiator command port
modep       equ   base + 2     ; mode port
tcmdp       equ   base + 3     ; target command port
stat1p      equ   base + 4     ; bus status port
prstp       equ   base + 7     ; reset parity/interrupts port

; Initiator Command Register commands - icmdp

rst         equ   10000000b    ; reset SCSI bus
sel         equ   00000101b    ; select device
ack         equ   00010001b    ; acknowledge device from host
ackin       equ   00010000b    ; acknowledge device to host

; Storage allocation in SCC RAM
; 2000h to 23ffh

stack       equ   2100h        ; reserved for stack
inhexs      equ   2200h        ; ASCII bytes and trap for last byte
buff        equ   2220h        ; working buffer
buff1       equ   2240h        ; working buffer
scsiwrf     equ   2260h        ; zero for single byte, otherwise block write

; ******** Main Program Start********

start       ld    hl,stack     ; set up stack pointer register
            ld    sp,hl
            ld    hl,baudrs    ; hl -> baud rate table
            ld    c,vdubaud    ; c = baud port
            ld    a,11h        ; reset bit and high baud
baud1       out   vducmd,a     ; reset
            outi
            call  gbyte
            call  gbyte
            and   7fh
            cp    0dh          ; recognise a cr?
            ld    a,1          ; reset bit only
            jr    nz,baud1
```

```
start1      ld      a,rst           ; reset SCSI
            out     icmdp,a
            ld      b,null
waitrst     djnz    waitrst         ; wait a moment
            ld      a,null
            out     icmdp,a         ; finish SCSI reset
            jp      prog

; ********Baud rate selection routine values********

baudrs      db      10010000b
            db      11000000b       ; 9600
            db      10100000b       ; 4800
            db      10010000b       ; 2400
            db      10001000b       ; 1200
            db      10000100b       ;  300
            db      10000010b       ;  150
            db      00000001b       ;  110

; ********Messages to VDU********

banner1     db      cr,lf,lf,'SCSI Tester V 01.02 July 1986',cr,lf
            db      'Class: ',null
crlf        db      cr,lf,null
msgalen     db      cr,lf,'Allocate length 1: ',null
msgaln2     db      cr,lf,'Allocate length 2: ',null
msgbcnt     db      cr,lf,'Block Count 1: ',null
msgbct2     db      cr,lf,'Block Count 2: ',null
msgcmd      db      cr,lf,'Command: ',null
msgcylh     db      cr,lf,'Cylinder 1: ',null
msgcyll     db      cr,lf,'Cylinder 2: ',null
msgdeft     db      cr,lf,'P/G/Code: ',null
msgfmc      db      cr,lf,'Format Control: ',null
msgfmi      db      cr,lf,'Interleave: ',null
msghead     db      cr,lf,'Head: ',null
msghexe     db      cr,lf,'? HEX ?',cr,lf,lf,null
msglba1     db      cr,lf,'LBA 1: ',null
msglba2     db      cr,lf,'LBA 2: ',null
msglba3     db      cr,lf,'LBA 3: ',null
msglba4     db      cr,lf,'LBA 4: ',null
msgmsg      db      '  Message: ',null
msgnot      db      cr,lf,'????',null
msgpcnt     db      cr,lf,'Pass Count: ',null
msgpcod     db      cr,lf,'Page Code: ',null
msgpmi      db      cr,lf,'PMI: ',null
msgplst     db      cr,lf,'Parameter List Length: ',null
msgspac     db      '  ',null
msgstat     db      cr,lf,'Status: ',null
msgtckh     db      cr,lf,'Address 1: ',null
msgtckm     db      cr,lf,'Address 2: ',null
msgtckl     db      cr,lf,'Address 3: ',null
msgwcl      db      cr,lf,'INVALID',null

; ********VDU I/0 Routines********
; Output a null terminated string
; hl -> string
; preserves all registers
```

```
outst       push    af              ; save registers
            push    hl
outst1      call    tbemp
            ld      a,(hl)          ; get the byte
            cp      null
            jr      z,outst2        ; a null, so all done
            out     vdudata,a
            inc     hl              ; point to next byte
            jr      outst1
outst2      pop     hl              ; restore registers
            pop     af
            ret

; Wait for transmitter bufer to empty
; preserves all registers

tbemp       push    af
tbemp1      in      a,vdustat       ; wait for TBE
            bit     tbe,a
            jr      z,tbemp1
            pop     af
            ret

; Wait for read data available and read byte into a
; preserves all registers except af

gbyte       in      a,vdustat       ; wait for a digit available
            bit     rda,a
            jr      z,gbyte
            in      a,vdudata
            ret

; Output a hexadecimal byte to VDU followed by a space
; (a) = binary value of byte
; preserves all registers

byteout     push    af              ; output hex byte to VDU
            and     0f0h            ; mask for upper hex value
            rrca                    ; shift down
            rrca
            rrca
            rrca
            call    byteou0         ; convert and output to VDU
            pop     af              ; get the binary value back
            push    af
            and     0fh             ; mask for lower hex value
            call    byteou0         ; convert and output to VDU
            call    tbemp
            ld      a,' '           ; output a space
            out     vdudata,a
            pop     af
            ret
byteou0     cp      0ah             ; test for hex adjustment
            jr      c,byteou1
            add     a,7
byteou1     add     30h             ; convert to ASCII
            call    tbemp
            out     vdudata,a
            ret
```

```
; Get a hex byte and convert to binary byte

gethex2   call   outst
          ld     b,3              ; maximum of two bytes
          call   inhex
          call   binary
          ret

; Convert one/two hex values (high/low) to binary
; high hex is stored in inhexs
; (a) = binary byte
; preserves all registers except af

binary    push   hl
          ld     hl,inhexs + 1
          ld     a,(hl)           ; if null only one byte input
          cp     null
          jp     nz,binary1
          dec    hl
          ld     a,(hl)           ; get byte
          and    0fh
          pop    hl
          ret
binary1   dec    hl               ; form both nibbles into binary value
          rld
          ld     a,(hl)
          pop    hl
          ret

; Input a hex string of numbers up to inhexm long terminated with cr
; input is stored in inhexs
; on input b = maximum number of hex digits required
; preserves all registers

inhexm    equ    0ah
inhex     push   af               ; save registers
          push   bc
          push   hl
          ld     a,inhexm -2
          cp     b
          jr     c,inhexe
          ld     hl,inhexs
          ld     b,inhexm         ; load count
inhex0    ld     (hl),null        ; nulls to storage area
          inc    hl
          djnz   inhex0
          ld     hl,inhexs        ; get back pointer
          ld     b,inhexm         ; load count
inhex1    call   gbyte            ; get a byte from VDU
          and    7fh              ; strip parity bit
          cp     cr               ; is number finished
          jr     z,inhex2
          ld     (hl),a
          call   outst            ; echo to VDU
          cp     '0'
          jr     c,inhexe         ; abandon if not hex
          cp     '9'+1
          jr     c,inhex3
          cp     'a'
          jr     c,inhex4
          sub    a,20h
inhex4    cp     'A'
```

```
                jr      c,inhexe        ; abandon if not hex
                cp      'F'+1
                jr      nc,inhexe       ; abandon if not hex
                sub     a,7             ; adjust byte
inhex3          ld      (hl),a          ; store digit
                inc     hl              ; increment pointer
                djnz    inhex1
                jr      inhexe          ; number too long
inhex2          ld      a,inhexm
                sub     b
                ld      b,a             ; b = no of hex digits
                cp      null
                jr      z,inhexe        ; abandon as no input
                pop     hl
                pop     bc
                pop     af
                ret
inhexe          ld      hl,msghexe      ; point to error message
                call    outst
                jp      prog

; ******** Main Program Continues********

prog            ld      hl,stack        ; set up stack pointer register
                ld      sp,hl
                ld      hl,banner1      ; output Class request to VDU
                call    outst
                ld      b,2             ; one byte only
                call    inhex
                ld      hl,inhexs + 1   ; hl -> second byte
                ld      a,(hl)          ; test only one byte
                cp      null
                jr      nz,cmderr
                dec     hl
                ld      a,(hl)          ; get the byte
                push    af              ; save Class
                ld      hl,msgcmd
                call    outst           ; ask for command hex bytes
                ld      b,3             ; possibly two bytes
                call    inhex
                ld      hl,inhexs       ; hl -> first byte
                ld      a,(hl)          ; get first byte
                and     0fh
                ld      e,a             ; keep it in e
                inc     hl
                ld      a,(hl)
                cp      null
                jr      z,cmdok         ; only one byte
                and     0fh
                ld      d,a             ; save second byte
                ld      a,e             ; get back first byte
                cp      1               ; check the value is one
                jr      nz,cmderr
                ld      a,10h
                add     a,d             ; form command
                ld      e,a             ; keep it in e
                inc     hl
                ld      a,(hl)
                cp      null
                jr      nz,cmderr       ; only two bytes allowed
cmdok           pop     af              ; restore Class
                cp      '0'             ; test valid Class
```

```
                jr      z,class0
                cp      '1'
                jp      z,class1
                cp      '7'
                jp      z,class7
cmderr          ld      hl,msgwcl       ; command error exit
                call    outst
                jp      prog

; ********Class-0 Commands********

class0          ld      a,e
                ld      hl,buff         ; hl -> command buffer
                ld      (hl),a          ; byte-0
                inc     hl
                cp      00h             ; test drive ready
                jp      z,null5
                cp      01h             ; recalibrate
                jp      z,null5
                cp      03h             ; request sense
                jp      z,sens5
                cp      04h             ; format unit
                jp      z,form5
                cp      07h             ; reassign blocks
                jp      z,reas5
                cp      08h             ; read
                jp      z,read5
                cp      0ah             ; write
                jp      z,writ5
                cp      0bh             ; seek
                jp      z,seek5
                cp      12h             ; inquiry
                jp      z,inqy5
                cp      15h             ; mode select
                jp      z,msel5
                cp      1ah             ; mode sense
                jp      z,msen5
                cp      1dh             ; send diagnostics
                jp      z,diag5
                jp      prog

; Class-0 and Class-7 send control bytes

cl0send
cl7send         call    select
                ld      hl,buff
                call    send6
                jp      phaser

; Class-1 send control bytes

cl1send
                call    select
                ld      hl,buff
                call    send10
                jp      phaser

; Send a null byte into command buffer and increment pointer

zero            ld      (hl),null
                inc     hl
                ret
```

```
; Byte-1 followed by 5 null bytes
; used by cmd000 and cmd001

null5       ld      b,5
null5a      call    zero            ; byte-1 to byte-5 are null
            djnz    null5a
            jp      cl0send

; Byte-1 followed by 3 nulls, allocation length (18), 1 null
; used by cmd003

sens5       ld      b,3
sens5a      call    zero            ; byte-1 to byte-3 are null
            djnz    sens5a
            ld      (hl),12h        ; allocation length
            inc     hl
            ld      (hl),null
            jp      cl0send

; Byte-0 followed by format control code, 2 null bytes, interleave, 1
; null byte used by cmd004

form5       push    hl              ; save pointer
            ld      hl,msgfmc       ; format control requested
            call    gethex2
            pop     hl              ; restore pointer
            ld      (hl),a          ; byte-1
            inc     hl
            ld      b,2
form5a      call    zero            ; byte-2 and byte-3 are null
            djnz    form5a
            push    hl              ; save pointer
            ld      hl,msgfmi       ; format interleave requested
            call    gethex2
            pop     hl              ; restore pointer
            ld      (hl),a          ; byte-4
            inc     hl
            ld      (hl),null       ; byte-5 null
            ld      a,null
            ld      (scsiwrf),a     ; single byte write to drive
            jp      cl0send

; Byte-1 followed by 5 null bytes
; used by cmd007

reas5       ld      b,5
reas5a      call    zero            ; byte-1 to byte-5 are null
            djnz    reas5a
            ld      a,null
            ld      (scsiwrf),a     ; single byte write to drive
            jp      cl0send

; Byte-0 followed by track address and one null byte
; used by cmd008

read5       call    setup5
            jp      cl0send
```

```
; setup5 - provides byte-1 to byte-5 for read5, writ5 and seek5
; setup5a        byte-4 and byte5 for frd5 and fwr5

setup5      push  hl              ; save pointer
            ld    hl,msgtckh      ; high track address requested
            call  gethex2
            pop   hl              ; restore pointer
            ld    (hl),a          ; byte-1
            inc   hl
            push  hl              ; save pointer
            ld    hl,msgtckm      ; middle track address requested
            call  gethex2
            pop   hl              ; restore pointer
            ld    (hl),a          ; byte-2
            inc   hl
            push  hl              ; save pointer
            ld    hl,msgtckl      ; low track address requested
            call  gethex2
            pop   hl              ; restore pointer
            ld    (hl),a          ; byte-3
            inc   hl
setup5a     push  hl              ; save pointer
            ld    hl,msgbcnt      ; block count
            call  gethex2
            pop   hl              ; restore pointer
            ld    (hl),a          ; byte-4
            inc   hl
            ld    (hl),null       ; byte-5 null
            ret

; Byte-0 followed by track address and one null byte
; requests write byte value once only
; used by cmd00a

writ5       call  setup5
            ld    a,-1
            ld    (scsiwrf),a     ; multi-byte write to drive
            jp    cl0send

; Byte-0 followed by track address and one null byte
; used by cmd00b

seek5       call  setup5
            jp    cl0send

; Byte-0 followed by 3 null bytes, allocation length, 1 null byte
; used by cmd012

inqy5       ld    b,3
inqy5a      call  zero            ; bytes 1 to 3 are null
            djnz  inqy5a
            push  hl              ; save pointer
            ld    hl,msgalen      ; allocation length requested
            call  gethex2
            pop   hl              ; restore pointer
            ld    (hl),a          ; byte-4 is allocation length
            inc   hl
            ld    (hl),null       ; byte-5 null
            jp    cl0send
```

```
; Byte-0 followed by 3 null bytes, parameter list length, 1 null byte
; used by cmd015

msel5     ld    b,3
msel5a    call  zero          ; bytes 1 to 3 are null
          djnz  msel5a
          push  hl            ; save pointer
          ld    hl,msgplst    ; allocation length requested
          call  gethex2
          pop   hl            ; restore pointer
          ld    (hl),a        ; byte-4 is parameter list length
          inc   hl
          ld    (hl),null     ; byte-5 null
          ld    a,null
          ld    (scsiwrf),a   ; single byte write to drive
          jp    cl0send

; Byte-0 followed by null, page code, null, allocation length, null
; used by cmd01a

msen5     call  zero          ; byte-1
          push  hl            ; save pointer
          ld    hl,msgpcod    ; page code requested
          call  gethex2
          pop   hl            ; restore pointer
          ld    (hl),a        ; byte-2 is page code
          inc   hl
          call  zero          ; byte-3 is null
          push  hl            ; save pointer
          ld    hl,msgalen    ; allocation length requested
          call  gethex2
          pop   hl            ; restore pointer
          ld    (hl),a        ; byte-4 is allocation length
          inc   hl
          ld    (hl),null     ; byte-5 null
          jp    cl0send

; Byte-1 followed by 04h followed by 4 null bytes
; cmd01d

diag5     ld    (hl),04h      ; byte-1
          inc   hl
          ld    b,4
diag5a    call  zero          ; byte-2 to byte-5 are null
          djnz  diag5a
          jp    cl0send

; ********Class-1 Commands********

class1    ld    a,e
          add   20h           ; add class-1 for byte-0
          ld    hl,buff       ; hl -> command buffer
          ld    (hl),a        ; byte-0
          inc   hl
          call  zero          ; byte-1
          cp    25h           ; read capacity
          jp    z,capy9
          cp    28h           ; read extended
          jp    z,rdex9
          cp    2ah           ; write extended
          jp    z,wrex9
          cp    2fh           ; verify
```

```
                jp      z,verf9
                cp      37h             ; read defect data
                jp      z,deft9
                cp      3ch             ; read data buffer
                jp      z,rddb9
                cp      3bh             ; write data buffer
                jp      z,wrdb9
                jp      prog

; Byte-0 and byte-1 followed by 4 byte logical block address, 2 null bytes
; PMI byte and null byte
; used by cmd105

capy9           push    hl              ; save pointer
                ld      hl,msglba1
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-2 is LBA high byte
                inc     hl
                push    hl              ; save pointer
                ld      hl,msglba2
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-3 is LBA byte 2
                inc     hl
                push    hl              ; save pointer
                ld      hl,msglba3
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-4 is LBA byte 3
                inc     hl
                push    hl              ; save pointer
                ld      hl,msglba4
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-5 is LBA low byte
                inc     hl
                call    zero            ; byte-6 null
                call    zero            ; byte-7 null
                push    hl              ; save pointer
                ld      hl,msgpmi
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-8 is PMI
                inc     hl
                ld      (hl),null       ; byte-9
                jp      cl1send

; Byte-0 and byte-1 followed by 2 nulls,3 byte logical block address,
; 1 null, 2 byte block count and null byte
; used by cmd108, cmd10a and cmd10f

rdex9
wrex9
verf9           call    zero            ; byte-2
                push    hl              ; save pointer
                ld      hl,msglba1
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-3 is LBA high byte
                inc     hl
                push    hl              ; save pointer
```

```
                ld      hl,msglba2
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-4 is LBA byte 2
                inc     hl
                push    hl              ; save pointer
                ld      hl,msglba3
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-5 is LBA low byte
                inc     hl
                call    zero            ; byte-6 null
                push    hl              ; save pointer
                ld      hl,msgbcnt
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-7 is block count MSB
                inc     hl
                push    hl              ; save pointer
                ld      hl,msgbct2
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-8 is block count LSB
                inc     hl
                ld      (hl),null       ; byte-9
                ld      a,-1
                ld      (scsiwrf),a     ; multi-byte write to drive
                jp      cl1send

; Byte-0 and byte-1 followed by P/G/CODE byte, 4 nulls,
; 2 byte allocation length and null byte
; used by cmd117

deft9           push    hl              ; save pointer
                ld      hl,msgdeft
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-2 is P/G/Code
                inc     hl
                ld      b,4
deft9a          call    zero            ; bytes 3 to 6 are null
                djnz    deft9a
                push    hl              ; save pointer
                ld      hl,msgalen
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-7 is MSB allocation length
                inc     hl
                push    hl              ; save pointer
                ld      hl,msgaln2
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-8 is LSB allocation length
                inc     hl
                ld      (hl),null       ; byte-9
                jp      cl1send

; Byte-0 and byte-1 followed by 6 nulls,
; 2 byte allocation length and null byte
; used by cmd11b and cmd11c
```

```
rddb9
wrdb9     ld    b,6
rddb9a    call  zero          ; bytes 3 to 6 are null
          djnz  rddb9a
          push  hl            ; save pointer
          ld    hl,msgalen
          call  gethex2
          pop   hl            ; restore pointer
          ld    (hl),a        ; byte-7 is MSB allocation length
          inc   hl
          push  hl            ; save pointer
          ld    hl,msgaln2
          call  gethex2
          pop   hl            ; restore pointer
          ld    (hl),a        ; byte-8 is LSB allocation length
          inc   hl
          ld    (hl),null     ; byte-9
          ld    a,null
          ld    (scsiwrf),a   ; single byte write to drive
          jp    cl1send

; ********Class-7 Commands********

class7    ld    a,e
          add   0e0h          ; add class-7 for byte-0
          ld    hl,buff       ; hl -> command buffer
          ld    (hl),a        ; byte-0 to command buffer
          inc   hl
          cp    0e0h          ; maintenance seek
          jp    z,mseek5
          cp    0e1h          ; format maintenance tracks
          jp    z,mform5
          cp    0e2h          ; certify
          jp    z,cert5
          cp    0e8h          ; fast read
          jp    z,frd5
          cp    0eah          ; fast write
          jp    z,fwr5
          jp    prog

; Byte-0 followed by 1 null, 2 byte cylinder number, 1 byte head number
; and null byte
; used by cmd700

mseek5    call  zero          ; byte-1
          push  hl            ; save pointer
          ld    hl,msgcylh
          call  gethex2
          pop   hl            ; restore pointer
          ld    (hl),a        ; byte-2 is cylinder number high byte
          inc   hl
          push  hl            ; save pointer
          ld    hl,msgcyll
          call  gethex2
          pop   hl            ; restore pointer
          ld    (hl),a        ; byte-3 is cylinder number low byte
          inc   hl
          push  hl            ; save pointer
          ld    hl,msghead
          call  gethex2
          pop   hl            ; restore pointer
          ld    (hl),a        ; byte-4 is head number
```

```
                inc     hl
                ld      (hl),null       ; byte-5 null
                jp      cl7send

; Set pattern of bytes
; used by cmd701

mform5          ld      (hl),00000000b  ; byte-1
                inc     hl
                ld      (hl),01010010b  ; byte-2
                inc     hl
                ld      (hl),01001111b  ; byte-3
                inc     hl
                ld      (hl),00000000b  ; byte-4
                inc     hl
                ld      (hl),00000000b  ; byte-5 null
                jp      cl7send

; Byte-0 followed by 2 null bytes, pass count, 2 null bytes
; used by cmd702

cert5           call    zero            ; byte-1
                call    zero            ; byte-2 is null
                push    hl              ; save pointer
                ld      hl,msgpcnt      ; allocation length requested
                call    gethex2
                pop     hl              ; restore pointer
                ld      (hl),a          ; byte-3 is pass count
                inc     hl
                call    zero            ; byte-4 null
                ld      (hl),null       ; byte-5 null
                jp      cl7send

; Byte-0 followed by 3 null bytes, block count, 1 null byte
; used by cmd708 and cmd70a

frd5
fwr5            ld      b,3
fwr5a           call    zero            ; bytes 1 to 3 are null
                djnz    fwr5a
                call    setup5a
                ld      a,null
                ld      (scsiwrf),a     ; single byte write to drive
                jp      cl7send

;********SCSI Subroutines********

; Read Current SCSI Bus Status Register and react to contents

; Status bit positions - stat1p

phasem          equ     00011100b       ; phase mask
datoutp         equ     00000000b       ; data out phase
cmdp            equ     00001000b       ; command phase
datinp          equ     00000100b       ; data in phase
statusp         equ     00001100b       ; status phase
msginp          equ     00011100b       ; message in phase

; Status for Target Command Register - tcmdp

datoutt         equ     00000000b       ; data out target
cmdt            equ     00000010b       ; command target
```

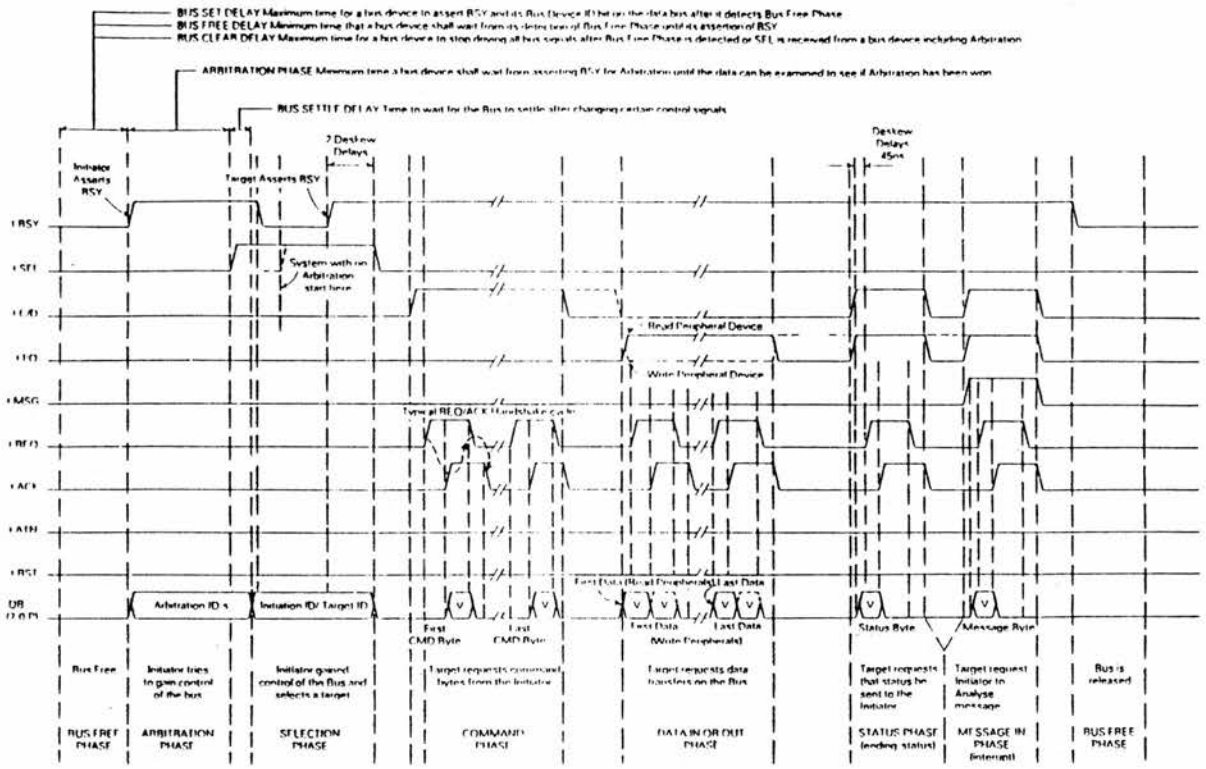
```
phaser      in      a,stat1p        ; read register
            bit     6,a             ; test busy
            jp      z,start1        ; all done?
            bit     5,a             ; test for request
            jr      z,phaser        ; not got one so try again
            and     phasem          ; mask out all but phase
            cp      datoutp         ; data out phase
            jp      z,scsiwr
            cp      datinp          ; data in phase
            jp      z,datain
            cp      statusp         ; status phase
            jp      z,status
            cp      msginp          ; message in phase
            jp      z,message
            jp      phaser          ; do not recognize it

; Acknowledge Drive

ackdrv      ld      a,ack           ; acknowledge drive
            out     icmdp,a
statms1     in      a,stat1p        ; wait for request false
            bit     5,a
            jr      nz,statms1
            ld      a,null          ; reset acknowledge false
            out     icmdp,a
            ret

; Read data byte from drive and output to VDU

datain      in      a,datap         ; read data byte
            call    byteout
            call    ackdrv          ; acknowledge drive
            jr      phaser

; Read status byte from drive and output to VDU

status      ld      hl,msgstat      ; hl -> banner
            call    outst
            in      a,datap         ; read drive status byte
            call    byteout
            call    ackdrv          ; acknowledge drive
            jr      phaser

; Read message byte from drive and output to VDU

message     ld      hl,msgmsg       ; hl -> banner
            call    outst
            in      a,datap         ; read drive status byte
            call    byteout
            call    ackdrv          ; acknowledge drive
            jp      prog

; Read data bytes from drive and output to VDU

scsird      in      a,datap         ; read data byte
            call    byteout
            call    ackdrv          ; acknowledge drive
            ret
```

```
; Select drive

select      ld      a,null
            out     modep,a         ; set as initiator and no interrupts
            out     icmdp,a         ; set initiator command register
            in      a,prstp         ; reset parity/interrupts port
            ld      a,00000001b     ; select drive address
            out     datap,a
            ld      a,datoutt       ; data out target
            out     tcmdp,a
            ld      a,sel
            out     icmdp,a
select1     in      a,stat1p
            bit     6,a
            jr      z,select1       ; wait for bsy true
            ld      a,null
            out     icmdp,a         ; reset select false
            ret

; Send ten or six control bytes to drive
; hl -> buffer containing the six or ten bytes

send6       ld      c,datap
            ld      b,6*2
            jr      send6c
send10      ld      c,datap
            ld      b,10*2
send6c      ld      a,cmdt          ; command target
            out     tcmdp,a
send6a      in      a,stat1p        ; wait for request
            bit     5,a
            jr      z,send6a
            outi                    ; send byte
            call    ackdrv          ; acknowledge byte ready
            djnz    send6a
            ld      hl,crlf
            call    outst           ; cr/lf to tidy display
            ret

; Write data byte to drive from VDU
; single/block depends on flag in (scsiwrf) - null for single byte

scsiwrb     db      cr,lf,'B: ',null
scsiwr      ld      hl,scsiwrb      ; ask for byte
            call    outst
            ld      hl,inhexs
            call    inhex
            call    binary
            push    af              ; save the byte
            ld      a,datoutt       ; data out target
            out     tcmdp,a
            ld      a,(scsiwrf)
            cp      null
            jr      nz,scsiwb       ; is it a block write?
            pop     af              ; get the byte back
            out     datap,a         ; write data byte
            call    ackdrv          ; acknowledge drive
            jp      phaser
scsiwb      in      a,stat1p        ; read register
            bit     5,a             ; test for request
            jr      z,scsiwb        ; not got one so try again
            and     phasem          ; mask out all but phase
```

```
                cp      datoutp         ; data out phase
                jr      nz,scsiwb0      ; return as all done
                pop     af              ; get the byte back
                out     datap,a         ; write data byte
                push    af              ; save the byte
                call    ackdrv          ; acknowledge drive
                jr      scsiwb          ; look for next data in byte
scsiwb0         pop     af              ; clean up the stack
                jp      phaser

                rept    [800h - $]
                db      -1
                mend

                end     start

; end of file
```

APPENDIX C

BUS PHASE SEQUENCES



BUS FREE PHASE

SEL and BSY are both false for at least one Bus Settle Delay

ARBITRATION PHASE

Implementation of this phase is a system option

At least one Bus Free Delay but no more than one Bus Set Delay after Bus Free Phase has been detected the INITIATOR asserts BSY and its own Bus Device ID bit on the data bus

The INITIATOR waits an Arbitration Delay then examines the Data Bus if a higher priority Bus Device ID is true on the Data Bus (DB7 is the highest) the INITIATOR loses Arbitration and releases BSY

SELECTION PHASE

During this phase the I/O signal is deasserted to distinguish this phase from the Re selection Phase

NON ARBITRATING SYSTEMS In systems with the Arbitration Phase not implemented after detecting the Bus Free phase the INITIATOR waits a minimum of one Bus Clear Delay then it asserts the Data Bus with both the desired INITIATOR ID bit and the TARGETS ID bit after two deskew delays the INITIATOR asserts SEL

ARBITRATING SYSTEMS In systems with the Arbitration Phase implemented the INITIATOR that won Arbitration has both BSY and SEL asserted and changes the Data Bus after two Bus Settle delays The Data Bus is then asserted with both the desired INITIATOR ID bit and the TARGETS ID bit Two deskew delays later BSY is released

IN ALL SYSTEMS the TARGET determines that it is selected when SEL and its Bus Device ID bit are true and BSY and I/O are false for at least a Bus Settle Delay The TARGET then asserts BSY within a Selection Abort Time

Two deskew delays after the INITIATOR detects BSY true it releases SEL and may change Data Bus signals

COMMAND PHASE

The TARGET asserts C/D and deasserts I/O and MSG for all of the handshakes of this Phase

The transfer is from INITIATOR to TARGET

HANDSHAKE PROCEDURE

The TARGET asserts REQ The INITIATOR drives data (7 0 P) to their desired values waits at least one deskew delay plus a cable skew delay and asserts ACK the INITIATOR continues to drive data (7 0 P) then asserts REQ

When REQ becomes false at the INITIATOR the INITIATOR may change or release data (7 0 P) and deassert ACK

The TARGET continues requesting command bytes The number of bytes depends on the command group code detected from the first command byte received

DATA IN OR OUT PHASE

DATA IN PHASE Read Peripheral

Data to be sent from TARGET to INITIATOR

TARGET asserts I/O deasserts C/D and MSG during the REQ/ ACK handshake of this phase

HANDSHAKE PROCEDURE

The TARGET first drives data (7 0 P) to their desired values waits at least one deskew delay plus a cable skew delay, then asserts REQ Data (7 0 P) shall remain valid until ACK is true at the TARGET

The INITIATOR shall read data (7 0 P) after REQ is true then asserts ACK

When ACK becomes true at the TARGET the TARGET may change or release data (7 0 P) and deassert REQ

After REQ is false the INITIATOR deasserts ACK after ACK is false the TARGET may continue the transfer by driving data (7 0 P) and asserting REQ

DATA OUT PHASE Write Peripheral

Data is to be sent from INITIATOR to TARGET

TARGET deasserts C D I O and MSG during the REQ/ACK handshake of this phase refer to the handshake procedure of the Command Phase

STATUS PHASE (Ending Status)

TARGET requests that a status byte be sent to the INITIATOR

TARGET asserts C/D and I/O and deasserts MSG during the handshake of this phase

Only one byte of status is transferred

See handshake procedure under Data in Phase

MESSAGE IN PHASE (Interrupt)

TARGET requests message byte(s) to be sent from TARGET to INITIATOR

TARGET asserts C/D, I/O and MSG during the REQ/ACK handshake(s) of this Phase

The message byte could indicate Command Complete (00) indicating interrupt

See handshake procedure under Data in Phase

BUS FREE PHASE

TARGET deasserts BSY to indicate that the bus is available for subsequent users

TARGET releases all bus signals within a Bus Clear Delay after BSY becomes continuously false for a Bus Settle Delay

The following Phase will be an Arbitration Phase

APPENDIX D

COMPARISON OF PERFORMANCE OF TTL BOARD AND NCR 53C80 BOARD

When this research work commenced, there was no VLSI SCSI interface chip (NCR 53C80) available for use. The Rodime RO652 hard disk drive interface was proprietary to Rodime PLC and could not be utilised. As a result, there was no alternative but to produce a prototype using purely TTL logic.

By using a TTL logic only interface, it was found that none of the SCSI maximum delay times could be realised and as such the TTL board does not conform to the American National Standards Institute (ANSI) X3T9 SCSI specification. The data transfer rate was limited to such an extent on the TTL board, that it was realised that only when a VLSI chip became available, a true to SCSI specification interface could be constructed.

The TTL interface board, allowed a full understanding of SCSI handshaking to be developed. It also allowed the majority of the firmware to be written and tested. This firmware was subsequently used as the basis of the firmware for the Intelligent Interface board.

The Intelligent Interface board was developed when the VLSI chip, NCR53C80, became available and unlike the TTL board, conformed to the ANSI SCSI specification data transfer rates.