A SCSI Printer Controller Using Either the DP8490 EASI or DP5380 ASI and Users Guide

National Semiconductor
Application Note 563
Andrew M. Davidson
June 1989

The DP8490 Enhanced Asynchronous SCSI Interface and DP5380 Asynchronous SCSI Interface are CMOS devices, which offer a low cost high performance Small Computer Systems Interface. These devices are pin compatible, and software compatible until an enhanced mode bit is set in the DP8490. This enhanced mode offers many new features which can yield increases in system performance through software, in addition to the improvements in speed and power shown by both devices over existing NMOS devices.

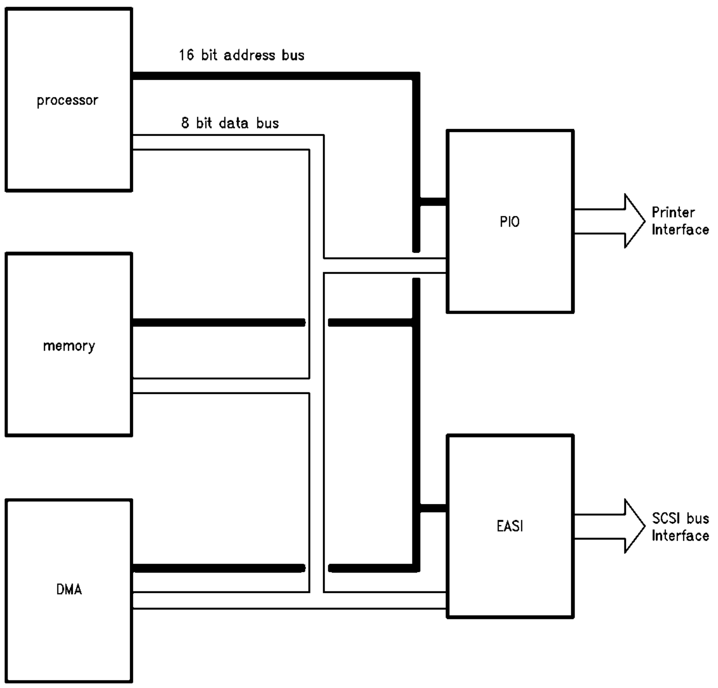
This application note shows how the hardware and software can be designed for a SCSI Printer Controller (SPC) so that it can incorporate either the DP5380 or DP8490. Since the software automatically detects which device is inserted either can be used, although the enhanced mode of the DP8490 offers a better system in terms of throughput and error tolerance. All of the software discussed is available on a floppy disk.

1.0 Introduction

The SCSI Printer Controller *(Figure 1.1)* consists of five main parts:

1. Microprocessor NSC800™
2. Printer Interface NSC831 PIO (Parallel Input/Output)
3. SCSI Interface DP8490 or DP5380
4. Memory
   CMOS EPROM NMC27C256
   CMOS Static RAM 62256
5. DMA Controller 9517 or 8237

The NSC800 is an eight bit CMOS microprocessor which is the central processing unit of the National Semiconductor



TL/F/10082–1

FIGURE 1.1

1.0 Introduction (Continued)

NSC800 microcomputer family. It is capable of addressing 64 kbytes of memory and 256 I/O devices using a multiplexed address and data bus. The instruction set is fully compatible with that of the Z80.

The NSC831 is the parallel input/output (PIO) device of the NSC800 family. This provides 20 I/O bits which can be individually programmed to be inputs or outputs. These are configured as two eight bit ports and one four bit port. The PIO is used as the interface between the microprocessor and the printer.

The DP5380 and DP8490 comply with the ANS X3.131-1986 SCSI standard as defined by the ANSI X3T9.2 committee. They can act as both Initiator and Target supporting all bus phases. Due to the on-chip high-current open-drain drivers the devices interface directly to the SCSI bus.

The 64 kbyte memory space is split between a 32 kbyte EPROM, containing the run-time software, and a 32 kbyte static RAM.

Data transfers between memory and the EASI can be controlled by the DMA controller, which supplies all read and write strobes for a transfer in either direction. This is a synchronous device which uses the 4 MHz clock output from the microprocessor.

The microprocessor is normally in control of the internal busses, giving it the ability to read or write memory or I/O devices. During a DMA transfer the DMA takes control of these busses and can pass data between the EASI and memory. As the microprocessor is the only device with 'intelligence' it must control these transfers. It commences and controls operations by setting registers in the DMA and EASI.

2.0 Hardware

2.1 DEVICES

2.1.1 Introduction

This section is intended to explain the hardware of the SPC referring to the circuit diagram given in Appendix A. This will describe the devices used and the signals generated but not the way in which they are programmed. What will be discussed is how the devices relate to each other, their relative timings, and any extra hardware required. A more exact description of the internal registers of the EASI, DMA and PIO, and their operation, will be given in the diagnostics section.

2.1.2 Microprocessor

The NSC800 is an eight bit microprocessor which multiplexes the eight bit data bus with the lower half of the address bus to create a sixteen bit address bus. An octal latch, MM74HCT373, is required to hold the address on the bus during a memory read or write, with the ALE (Address Latch Enable) output from the NSC800 strobing the latch.

Since the bus has devices which use both TTL and CMOS levels pull-up resistors are required. The $\overline{\text{IOM}}$ output signifies whether the processor is on a memory or an I/O cycle and is used along with the $\overline{\text{RD}}$ and $\overline{\text{WR}}$ outputs to strobe bus data.

This processor allows control of the bus to be passed to an I/O device, using the $\overline{\text{BRQ}}$ (bus request) input and $\overline{\text{BACK}}$ (bus acknowledge) output. On this board the DMA is the only device which may request control of the bus.

When the EASI issues a DMA request the DMA signals that it needs control of the bus by issuing a $\overline{\text{BRQ}}$. The processor acknowledges this by issuing $\overline{\text{BACK}}$, and then drives the bus and all related control signals to their high impedance state. The MM74HCT373 must TRI-STATE® its outputs; the output enable is driven by the DMA signal AEN (address enable), which it asserts when it has control of the bus. At the end of its operations the DMA releases AEN (enabling the latch outputs) and negates $\overline{\text{BRQ}}$. The processor de-asserts $\overline{\text{BACK}}$ and retakes control of the bus by enabling its outputs. To prevent spurious signals being generated when the bus is not driven the control signals must have pull-up resistors.

The NSC800 provides five hardware interrupts of which only $\overline{\text{RSTA}}$ is used. The others are tied inactive. $\overline{\text{RSTA}}$ is driven by the EASI interrupt output, with an inversion between them to allow for the difference in active levels. An interrupt on this pin causes a software reset to a particular address in memory (more details of interrupt servicing will be given in the diagnostic software section).

The system clock is generated using an 8 MHz crystal, with the frequency divided by two for use by the processor and the DMA. There is also power-on reset circuitry, which resets the processor, and causes a reset output to the other devices when power is first applied.

2.1.3 DMA Controller

The 9517 and 8237 are compatible direct memory access devices, controlled by setting internal registers. These registers are selected using a chip select, with the particular register selected by the lower nibble of the address bus, and data strobed by $\overline{\text{IOR}}$ (I/O read) or $\overline{\text{IOW}}$ (I/O write). These registers control the type of data transfer, the number of bytes transferred and the memory address the transfer is to/from.

The two modes of transfer used are single mode and block mode. In single mode the DRQ (DMA request) input and the $\overline{\text{DACK}}$ (DMA acknowledge) output are used to "handshake" every byte of data transferred. In block mode, after an initial DRQ, $\overline{\text{DACK}}$ must be active until after the last byte has been transferred. The rate of transfer is controlled by the READY input.

Another device required in conjunction with the DMA is an MM74HCT74 'D' type flip-flop. This is used to overcome the metastability problem introduced by the asynchronous READY signal driving a synchronous device. The flip-flop synchronizes the input with the system clock.

The DMA supplies all the necessary control signals to move data from memory to EASI or vice-versa. When all data has been transferred the DMA drives $\overline{\text{EOP}}$ (end of process) active. $\overline{\text{EOP}}$ can also be used as an input to the DMA, prematurely terminating any transfer. It is configured as an open-drain driver, so requires a pull-up resistor, of an advised value of 4.7 kΩ.

As in the processor the DMA has a multiplexed address and data bus, also requiring an MM74HCT373 to latch the address. In this case it is the upper byte of the address bus which is multiplexed with the data bus. The strobe signal is similar to ALE but is generated in the DMA and called ADSTB (address strobe). The TRI-STATE for this latch is driven by the inverse of AEN, since the device must drive the bus only when the DMA has control.

The 9517 and 8237 have four DMA channels, of which only one is used. The DRQ inputs for the other three channels are tied inactive. The RESET input causes the control registers to be cleared and the DRQ inputs to be masked.

2.1.4 EASI

The EASI is controlled by setting internal registers, written by an $\overline{IOW}$ and chip select, read by an $\overline{IOR}$ and chip select. The particular register selected is determined by the lowest three bytes of the address bus. The EASI has an eight bit microprocessor data bus, and on the PLCC part a microprocessor bus parity pin. However the NSC800 does not support parity checking so this bit is tied low. The other microprocessor controlled pin is the $\overline{RESET}$. This causes the EASI to clear all registers and therefore reset all logic.

The DRQ output and $\overline{DACK}$ input "handshake" single mode DMA, while the READY output is used to control the speed of a block mode transfer. $\overline{EOP}$ is an input which terminates DMA, and can be used to cause an interrupt. This interrupt output INT can interrupt the microprocessor if the EASI detects an error, or has completed some task.

SCSI uses an eight bit data bus with a parity bit; which must support odd parity during all bus transactions except arbitration. Other SCSI signals are the bus selection control signals $\overline{SEL}$ and $\overline{BSY}$, phase control signals $\overline{MSG}$, $\overline{C/D}$ and $\overline{I/O}$, data transfer handshakes $\overline{REQ}$ and $\overline{ACK}$ and the message flag $\overline{ATN}$. Further details on the use of these signals will be given in the software sections of this document. The SCSI reset $\overline{RST}$ is similar to a chip reset, but generates an interrupt.

The EASI interfaces directly to the SCSI bus using high-current open-drain drivers. This bus is a 50-way ribbon cable (maximum length 6.0 meters) on which all SCSI devices are daisy-chained. The devices at the end of this cable must terminate the SCSI signals i.e., a 220Ω resistor to power, and a 330Ω to ground. This power can be V_{CC} or TERMPWR (terminator power).

TERMPWR is V_{CC} fed through a Schottky barrier diode. This can be fed to pin 26 of the connector allowing the SCSI device at one end of the bus to be powered down without affecting the bus. Since the terminators are receiving power the bus can operate effectively. To make this optional TERMPWR is fed to the connector through a jumper link. The Schottky diode must be included to prevent backflow of power into the printer controller board.

Other manufacturers CMOS devices can not be used in a configuration like this, as they pull the bus low when not powered. The DP5380 and DP8490 have a special input protection to overcome this problem.

All other pins on the connector should be tied to ground, except pin 25 which must be left floating.

2.1.5 PIO

The printer controller uses a NSC831 PIO to interface between the processor and the printer. The NSC831 has 20 individually programmable I/O bits which are arranged as three ports; A, B and C. As it is part of the NSC800 family the NSC831 has a compatible multiplexed address and data bus with an ALE input, to strobe the address into an internal latch. This eight bit address can then be used to access an internal register, in conjunction with an $\overline{IOR}$ or $\overline{IOW}$ and chip select. The device has two chip selects, one of which is tied active (low), with the other coming from the address decode PAL®. The RESET input causes the three ports to become all inputs.

In this application the three ports are set up so that each port is all input or all output. Port A is an output, which drives the printer data bus through an MM74HCT241 octal buffer. Port B is an input, with the lower nibble coming from the printer outputs, driven by half an MM74HCT240 octal inverting buffer. The pull-down resistors on the $\overline{ERROR}$ and PE (paper error) inputs give a proper error signal if the printer is unconnected i.e., the same as when the printer is off-line. The upper three bytes of Port B are connected to switches, which are used to set the board's SCSI I.D. This will be further explained in the SCSI diagnostics section. Only three bytes of Port C are used, all as outputs, driven by the same device used by Port B. Bits 0 and 2 are printer outputs, while bit 3 drives a LED. This displays the 'health' of the board i.e., when the board is operational the LED is on, when non-operational it is off and when there is a known hardware error a message is 'flashed'.

The printer connector is a standard IBM® 25 way 'D' range, with all unused pins grounded, except 13.

2.2 PAL EQUATIONS

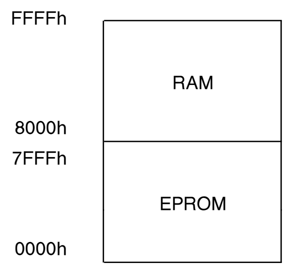
2.2.1 Introduction

PAL's are used in this board to generate chip selects, overcome potential timing problems and to invert signals for devices with different active levels. The PAL's used are National Semiconductor's PAL16L8. The equations are written in a form compatible with PLAN (Programmable Logic Analysis by National) Ver. 2.00.

2.2.2 Decode PAL

The decode PAL supplies the five main devices with their chip selects.

Memory is split into two 32k byte blocks, each of which represents one device. These devices are an EPROM, at the base of memory, and RAM in the upper half. Thus the memory map and chip select equations are as follows:



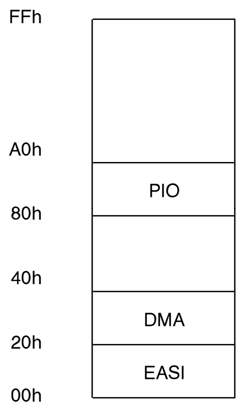
/EPROMZ = /A15*BACKZ*/IOMZ

/RAMZ = A15*BACKZ*/IOMZ + A15 * AEN

EPROMZ (the Z at the end of the name shows it is active low) can only be selected by the processor, while RAMZ is available to the DMA.

2.0 Hardware (Continued)

The DMA, PIO and EASI fit into the I/O map shown, which results in the equations following. These equations show that the devices can only be selected by the microprocessor on an I/O cycle.



/EASIZ = /A7 * /A6 * /A5 * IOMZ * BACKZ

/DMAZ = /A7 * /A6 * A5 * IOMZ * BACKZ

/PIOZ = A7 * /A6 * /A5 * IOMZ * BACKZ

The other three outputs are used as inverters:

/BRQZ = HRQ

/AENOZ = AEN

/HLDA = BACKZ

The two spare inputs are tied low.

2.2.3 Control PAL

The NSC800 uses $\overline{\text{IOM}}$ to distinguish between memory and I/O cycles. This allows $\overline{\text{IOR}}$ and $\overline{\text{IOW}}$, for processor cycles to be generated.

/IORZ = /RDZ * IOMZ
IORZ.TRST = BACKZ

/IOWZ = /WRZ * IOMZ
IOWZ.TRST = BACKZ

The second line of these equations shows that the output is TRI-STATE when the processor relinquishes control of the bus.

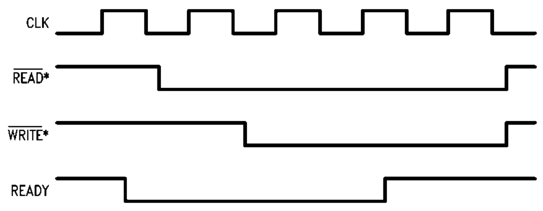
Before examining the next five equations it is important to fully understand the relative signal sequencing involved in a DMA transfer. When the transfer is initiated the EASI issues a DRQ, causing the DMA to respond with $\overline{\text{DACK}}$, and take control of the bus by asserting $\overline{\text{BRQ}}$. The processor asserts $\overline{\text{BACK}}$ and allows the bus and relevant control signals to go TRI-STATE. The DMA then asserts AEN, to show it is in control of the bus, causing the microprocessor address latch to TRI-STATE its outputs and the DMA latch to enable its outputs.

For single mode each byte transferred must have a DRQ and a $\overline{\text{DACK}}$. For block mode DRQ returns inactive after the DMA responds with a $\overline{\text{DACK}}$, but the $\overline{\text{DACK}}$ must remain active until the transfer is complete. The READY line controls the rate of block mode DMA transfer i.e., when READY is low the byte transfer is 'frozen', until READY returns high (*Figures 2.1a* and *2.1b*).

The READY signal is asynchronous but the DMA is synchronous, testing the READY signal on the negative edge of each clock. For this reason the READY input must be synchronized, using a flip-flop which updates its output on the positive edge of the clock.

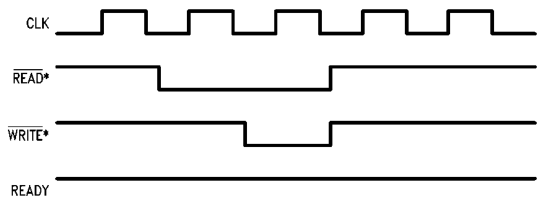
When READY is detected as being inactive on the negative edge of the clock an extra cycle is inserted to the read and write.

While data is being transferred a register keeps track of how many bytes are left. When this reaches zero an $\overline{\text{EOP}}$ signal is generated, concurrent with the last read and write, causing the EASI to set an EOP flag.



TL/F/10082–2

FIGURE 2.1a



TL/F/10082–3

FIGURE 2.1b

***Note:**

Of these read and write signals, one will refer to memory the other I/O, depending on the direction of data transfer.

2.0 Hardware (Continued)

The tranfer is now complete so $\overline{\text{BRQ}}$, AEN and $\overline{\text{DACK}}$ are all driven inactive. AEN going inactive causes the address latches to swop back over; the microprocessor latch outputs are driven, the DMA latch outputs are TRI-STATE. When the microprocessor detects that $\overline{\text{BRQ}}$ is inactive it deasserts $\overline{\text{BACK}}$ and retakes control of the bus. The microprocessor then clecks flags in the EASI to determine the success, or otherwise, of the transfer.

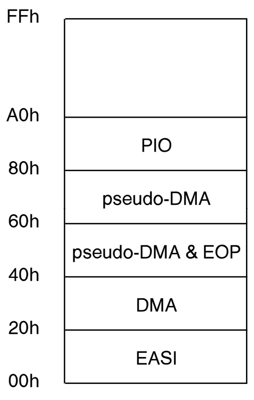
The other form of DMA available is pseudo-DMA; that is the EASI 'thinks' it is a proper DMA transfer but the processor is still in control of the bus. To do this the microprocessor must set the DMA to mask off any DRQ, initialize the EASI for single mode DMA and monitor the EASI flags for DRQ going active. At this point the processor must generate an $\overline{\text{IOR}}$ or an $\overline{\text{IOW}}$ and a $\overline{\text{DACK}}$, to properly simulate DMA. This can be done by causing an I/O read or write at a certain address, to generate a $\overline{\text{DACK}}$, and if wanted an $\overline{\text{EOP}}$. The particular address does not matter since during DMA the EASI ignores the address bus.

The following equations generate pseudo-DMA signals, while allowing the true DMA signals to operate normally:

```
/DACKOZ = /IORZ * BACKZ * /A7 * A6
          + /IOWZ * BACKZ * /A7 * A6
          + /DACKIZ
  /EOP2 = /IORZ * BACKZ * /A7 * A6 * /A5
          +/IOWZ * BACKZ * /A7 * A6 * /A5
          +/EOP1 * EREADY
```

DACKIZ is the $\overline{\text{DACK}}$ from the DMA, which the output normally follows. The rest of the equation generates the pseudo-DMA $\overline{\text{DACK}}$.

$\overline{\text{EOP2}}$ is the input to the EASI, $\overline{\text{EOP1}}$ is the output from the DMA. The first two lines of this equation generate an $\overline{\text{EOP}}$ for the pseudo-DMA cycle, on the lower half of the address space that also generates the pseudo DACK. This means the final I/O map now is as follows:



The third part of the $\overline{\text{EOP1}}$ equation is to overcome a problem that occurs during a block mode DMA transfer. On the last byte the READY signal from the EASI (this will be called EREADY to distinguish it from the READY into the DMA, DREADY) may be driven low to freeze the transfer. However $\overline{\text{DACK}}$, $\overline{\text{EOP}}$ and $\overline{\text{IOR}}$ or $\overline{\text{IOW}}$ will all be active causing a valid EOP condition. This is not a problem until we consider the next equation.

```
/DREADY = /EREADY * /INT * /DACKIZ
```

This equation causes the DMA READY input to go high if any of EREADY, SCSI interrupt INT or DACKIZ go high. This overcomes a potential bus lockup, caused by an error occurring during a block mode DMA transfer. On a phase or parity error the EASI will stop the transfer and generate an interrupt. If DREADY is left inactive the DMA will keep control of the microprocessor bus. This equation holds DREADY high on interrupt, allowing the DMA to pass control of the bus back to the microprocessor.

Although this equation prevents an error it also introduces a problem in the $\overline{\text{EOP2}}$ equation. As explained previously, during the last byte of a block mode transfer although READY is low a valid EOP condition may exist, which will cause an interrupt. This interrupt will then cause DREADY to be driven high, allowing the DMA to finish the transfer, but lose the last byte since the EASI is not ready. To overcome this the $\overline{\text{EOP2}}$ equation gates $\overline{\text{EOP1}}$ with EREADY, so the EASI does not see a valid EOP condition until it is able to transfer the last byte. **This problem does not occur in MODE E since it generates a true end of DMA interrupt.**

Another problem is introduced by the DREADY equation. If an error occurs during a DMA transfer the EASI will generate an interrupt and DREADY will be forced high. This allows the DMA to 'run free', writing garbage into MEMORY, and wasting SCSI bus time. To prevent this an external $\overline{\text{EOP}}$ must be applied to the DMA on error. The next equation does this:

```
/EOP1 = /DACKIZ * /EREADY * INT
EOP1.TRST = /DACKIZ * /EREADY * INT
```

$\overline{\text{EOP1}}$ is an I/O pin which normally acts as input from the DMA, so the output is TRI-STATE unless DACKIZ and EREADY are low, with INT active. When this error condition occurs an $\overline{\text{EOP}}$ is output to the DMA, terminating the data transfer. A prerequisite of this equation working properly is the existence of the DREADY equation, since an externally applied $\overline{\text{EOP}}$ will have no effect on the DMA if READY is held low.

The next equation is also required to prevent a fault occurring during a block mode DMA transfer:

```
/EASIWRZ = /IOWZ * BACKZ
           +/IOWZ * EREADY * /BACKZ
```

When the processor is in control of the bus a straightforward $\overline{\text{IOW}}$ can write to the EASI, but if the DMA is in control EREADY must be high to allow the write. The inclusion of EREADY is required because during DMA the EASI is a flowthrough latch; an $\overline{\text{IOW}}$ passes the data from the processor bus onto the SCSI bus. Therefore if the DMA reaches its next byte before the data on the SCSI bus has been transferred, the DMA will overwrite the data. To prevent this $\overline{\text{EASIWR}}$ can only be allowed when EREADY is high.

The final PAL output is used to invert the SCSI interrupt signal, to make this active high output compatible with the processor's active low input.

```
/SCSIINTZ = INT
```

3.0 Diagnostic Software

3.1 INTRODUCTION

The diagnostic software resides at the base of RAM, with the purpose of checking and initializing the printer controller board after power up and every hard reset. It must be at location zero, since after any hardware reset the program counter is cleared. The interrupt service routine is included here, since it must exist at an exact location in memory.

The software uses the NSC800 instruction set (fully Z80® compatible) which makes the required low level board operations faster, and allows the exact positioning of code in memory. The assembler and linking loader are from Microtec Research.

3.2 DEVICE CHECKING AND INITIALIZATION

This section will refer to the diagnostic software PRINTER.SRC and the files PRNSYM.SRC and EASISYM.SRC which contain the constants used. A full listing of all programs described in this document are available on floppy disk.

3.2.1 Memory Check and Interrupt Servicing

An 'org' statement can be used to position this program at the base of ROM, address 0. After a jump to the ROM check, the next byte stores a label defining which version of the software is installed. On an EPROM the version number can be read from address 0002h. The upper and lower nibbles should be considered as two numbers i.e., '10' defines version 1.0.

The interrupts are disabled, since the service routines are not initialized, and the EPROM is checked; simply consisting of reading the same address twice and ensuring the same value returns both times. If this test fails the system halts since a drastic error must have occurred. If the EPROM passes the test the program jumps to the RAM check.

On interrupt the NSC800 stops before its next instruction, pushes the program counter onto the stack and loads it with a new address, depending on the highest priority interrupt channel active. Interrupt A, the only channel used, causes a jump to location 003Ch. The interrupt service routine at this location pushes all of the processor registers onto the stack, and tests for a SCSI reset. SCSI reset must be checked, as this is a special condition, causing the board to be reset by a general restart routine (this routine is in EASIO.SRC and explained in section 3.5).

If the interrupt is not a SCSI reset the software makes a call to the bottom of RAM, where a jump command should be followed by a public variable RESETA. This variable can be loaded with the starting address of the routine to service the interrupt. On return from this routine the processor registers are popped back off the stack, and program control returns from the interrupt. Since RESETA is public any external software may use it, and therefore control which routine services an interrupt. RAM must be verified before this jump table is set up.

The RAM test simply checks each BIT can contain a zero and a one, then clears every byte. Once RAM has been verified the stack pointer may be initialized, allowing call statements to be included. It should be remembered that although the interrupt routine is between the ROM and RAM tests, when the program runs it will jump straight from the ROM test to the RAM test. Since the interrupt jump table is in volatile RAM the code for a jump instruction must be written into RAM after every reset.

3.2.2 PIO Initialization

The file PRNSYM.SRC contains the port addresses for all of the I/O devices. The upper nibble contains the location the device takes within the I/O map, the lower nibble selects the register within the device to be accessed.

The PIO has five types of registers which control data transfers through the device:

1. **Data Register**—Each port has an eight bit data register containing the data passed between the PIO and the processor. These are either read or write registers, depending on whether the port bits are inputs or outputs.
2. **Data Direction Registers**—Each port also has a data direction register which controls whether each of the 20 bits is an input or an output (an input is defined by a zero, an output by a one).
3. **Mode Register**—There is one three bit mode register which selects which of the four modes the device is in. This board will always require the PIO to be in Mode 0 which is the basic I/O mode. The alternatives use Port C for handshaking.
4. **Bit Clear Register**—Each port has an eight bit, bit clear register which clears any output bit whose corresponding bit in this register is high.
5. **Bit Set Register**—These are similar to the above, but allow the output bits to be set.

The PIO is set for Mode 0, with Ports A and C outputs and Port B an input. Port A drives the printer data bus, while Ports B and C read and write the printer control signals. The four printer outputs, to the Port B input, are $\overline{\text{ACK}}$, BUSY, PE and $\overline{\text{ERROR}}$. $\overline{\text{ERROR}}$ goes low when there is no paper, the printer is off-line or an error occurs; PE goes high when the printer is out of paper; BUSY goes high to show the printer can accept data; $\overline{\text{ACK}}$ pulses low to acknowledge data after a byte has been transferred. Since BUSY and $\overline{\text{ACK}}$ are both handshake signals the user must decide which to use. This software will use BUSY.

The top three bytes of Port B are used to read in the SCSIID from a block of switches. The SCSIID is the boards identifier on the SCSI bus, used in selection and arbitration, consisting of a single active bit. This is read in as a three bit number, converted to the correct bit pattern and stored in a public variable called SCSIID.

Port C is an output, driving the printer signals $\overline{\text{INIT}}$ and $\overline{\text{STROBE}}$. A low pulse on $\overline{\text{INIT}}$ of 50 μs causes the printer to be initialized; a 500 ns low pulse on $\overline{\text{STROBE}}$ causes the printer to read the data on the bus. The highest bit of Port C drives a LED, which is on during board operation, and can display an error message by occulting a fixed code. Errors are displayed by the routine *ERROR,* a public function which displays the error number as a four bit code. It does this by occulting the LED for 1 second to show a 1, ½ a second for a 0.

Since none of the PIO registers are true read/write the device cannot be tested, only initialized.

3.2.3 DMA Test and Initialization

The DMA consists of address, word-count and control registers for four channels, of which only one is used. There follows a description of the registers used, with the addresses shown in PRNSYM.SRC.

1. **Word-Count Registers**—There are two word-count registers; a sixteen bit write only base word-count register and a sixteen bit read only current word-count register. As with all sixteen bit registers in this device the two bytes of data are accessed by two successive selections of the same address. An internal flip-flop determines which byte is read or written, with the lower byte selected first after a reset. To transfer the correct number of bytes the word-count register must be written with the number of bytes to be transferred minus one.

3.0 Diagnostic Software (Continued)

2. **Address Registers**—The address registers are also sixteen bit, and called base and current. The base address register is written with the address the transfer must start at, while the current address register contains the next address to be written or read. *The DMA is tested by writing a value to the base address and reading it back from the current address*.
3. **Control Registers**—There are three types of control register which will be discussed only in the way they are used by this software. The master clear register (DMAMCL) is the software equivalent of a hardware reset; all registers are cleared and DRQ is masked off. Masking of DRQ is controlled by the parallel mask register (DMAMSK), in this case always used to allow DRQ on Channel 0 and mask off the others. The final type of register used is the mode register (DMAMOD) which controls whether a transfer is block or single mode, memory read or memory write. Each of the four channels has a six bit mode register, all at the same address, with the bottom two bits of the data bus determining which is selected.

The DMA test program initializes the device for a one byte, block mode, memory write, to an address TSTBYT. The DMA is initialized in this particular manner for use in the EASI loopback DMA test routine.

An error in DMA test will cause error signal 0.

3.3 EASI TEST AND INITIALIZATION

3.3.1 Introduction

This document will show how the registers within the DP5380 and DP8490 are used in an application. For a full register description refer to the DP5380 and DP8490 datasheets. Register names and their locations are given in PRNSYM.SRC.

3.3.2 DP8490 and DP5380 Test

The DP5380 and DP8490 are completely pin and hardware compatible, and also software compatible until the enhanced mode bit is set in the DP8490. The DP8490 powers up in normal mode (MODE N which is software compatible with the 5380) and is in MODE N after any chip reset. The initial test is therefore the same for both devices, writing to the Mode Register 2 (EASIMR2) and reading back the data. The EASIMR2 is selected since it has the most read/write bits that do not directly affect the bus.

An error in EASI test will cause error signal 1.

For the DP5380 testing is now complete, but the DP8490 enhanced mode offers a loopback test facility, where the SCSI drivers are disabled and the SCSI I/O's looped back inside the EASI. **Using this feature the user can fully check the device, and by doing a DMA transfer fully check the board.**

At this stage it is therefore neccessary to determine which device is inserted. In the DP5380 bit 6 of the Initiator Command Register (EASIICR) selects the 'test mode', which disables all output drivers on the device, making it invisible to the system. Although the device can still be written to no data can be read back, making applications very limited. In the DP8490 this bit selects the enhanced mode (MODE E) of the device.

In MODE E addresses 0–6 access the same registers as in MODE N, but address 7 is different. Instead of accessing the Reset Parity/Interrupt (EASIRPI) and Start DMA Initiator receive (EASISDI) registers, address 7 directly accesses the Enhanced Mode Register (EASIEMR) and indirectly accesses the Interrupt Mask Register (EASIIMR) and Interrupt Status Register (EASIISR). The only other difference to registers occurs in the Target Command Register (EASITCR) where one of the previously unused bits becomes a flag (explained in section 3.3.4).

To test which device is inserted EASI test sets the enhanced/test mode bit, writes data to address 7 and reads back from the same address. If the device is a DP5380 it will be in 'test mode' and the data read will be 0FFh due to the pull-up resistors on the data bus. If it is a DP8490 the data read back will be the data written, providing the only bits set are read/write. For a DP5380 the program jumps to initialization for selection; if it is a DP8490 loopback testing follows.

3.3.3 DP5380 Initialization for Selection

The DP5380 initialization involves programming the device to respond to a selection. First, the public variable DP8490 (which other programs should treat as a constant) is set FALSE to indicate that a DP8490 is not inserted. For the DP5380 to be selected it must have the SCSIID of the device in its Select Enable Register (EASISER) and parity must be enabled. The processor must enable its interrupts and set up the jump table; *intA* is an external routine which sets the processor's interrupt mask to only allow interrupts on A, and enables the interrupts (see EASIO.SRC); *main__* is an external program which responds to a SCSI selection (further explained in the Run-time software section).

If BSY is inactive for a bus settle delay (400 ns), SEL is true and the bit on the data bus corresponding to the SCSIID is active, a SCSI selection interrupt is generated. This causes a processor interrupt, taking the program from the continuous 'jr Now' instruction to the interrupt service routine, and through the jump table to *main__*. When *main__* has finished servicing the interrupt it will return to the interrupt servicing routine and then return from the interrupt. Thus the board's 'idle' state is to continually execute the 'jr Now' instruction.

3.3.4 DP8490 Loopback Test

The loopback test mode of the DP8490 allows all signals to be fully tested, including a DMA transfer, without affecting the SCSI bus. To allow this DMA transfer loopback allows the user to drive both initiator and target signals simultaneously.

After setting the DP8490 flag true all initiator signals, and then target signals, are asserted and checked. This includes a check on the data bus by writing a test value to it and reading it back. The data bus test value has an odd number of bits, and since the specified SCSI parity is ODD, the parity bit must be inactive. One of the new features offered in MODE E is programmable SCSI parity, which is tested by ensuring the parity bit becomes active when EVEN parity is enabled, and inactive when a test value with an even number of active bits is written. Parity is then returned ODD, and the parity bit should become active.

An error in Loopback testing causes error signal 2.

The next test in loopback mode is a DMA target receive transfer to a location in memory, TSTBYT. This not only tests the EASI, but also the interrupt servicing, the DMA and memory. Although the device is in loopback the software must carry out the transfer as it would normally i.e., the bus phase must be correct, $\overline{\text{BSY}}$ must be active and the SCSI bus must be asserted. The EASIMR2 must be properly set up for block mode DMA, with interrupt on $\overline{\text{EOP}}$ or parity error. Since this test is interrupt driven the interrupt jump table must be loaded with the address of the routine which will service the DMA loopback test, and the interrupts must be enabled. The DMA has already been initialized for this.

Before enabling interrupts the SCSI interrupts should be reset, which in MODE N would involve reading address 7. Since the EASIEMR is now at this address the resetting of interrupts, and the start of DMA initiator receive are initiated by writing to function bits of the EASIEMR.

When the Start DMA Target receive (EASISDT) register is written the EASI will issue a $\overline{\text{REQ}}$ to show it is ready to receive data on the SCSI bus. At this point the device at the other end of the bus would normally assert $\overline{\text{ACK}}$ to show it has data available. In this case there is no other device so the user must wait for $\overline{\text{REQ}}$ to go active and then assert $\overline{\text{ACK}}$. Thus both initiator and target signals must be asserted simultaneously. The user waits for $\overline{\text{REQ}}$ to go inactive, and deasserts $\overline{\text{ACK}}$ to show the bus transfer is complete.

The program then goes into a continuous loop awaiting a SCSI interrupt. This interrupt will occur because the EASI will have issued a DRQ, when $\overline{\text{ACK}}$ went active, and the DMA wil have transferred the last test byte written to the EASI to TSTBYT, finishing with an $\overline{\text{EOP}}$. On interrupt the program will jump to address 003Ch, where all the processor registers are pushed onto the stack, a call is made to the base of RAM, and from there it will jump to the subroutine *DIAGA*.

One of the problems with DMA in a DP5380 is that end of DMA is flagged when $\overline{\text{DACK}}$, $\overline{\text{EOP}}$ and $\overline{\text{IOR}}$ or $\overline{\text{IOW}}$ are simultaneously active; although the data may not yet have been transferred on the SCSI bus. To overcome this the software must examine the SCSI handshake signals $\overline{\text{REQ}}$ and $\overline{\text{ACK}}$, both of which must be inactive on three successive samples for a true end of DMA. This is more fully explained in the DP5380 and DP8490 datasheets. **MODE E of the DP8490 detects true end of DMA, after $\overline{\text{ACK}}$ goes inactive, before generating an interrupt.**

DIAGA responds to the interrupt after the loopback DMA by checking all the correct flags have been set. The DP5380 only uses four bits of EASITCR so MODE E uses the free bit 7 as a flag, to show true end of DMA. This is the first flag checked. Address 7 not only directly addresses the EASIEMR it also indirectly addresses the EASIIMR and EASIISR. After writing the correct code to the function bits of the EASIEMR the next access of address 7 will be to the EASIISR, if it is a read, or the EASIIMR, if it is a write. **The advantage of the EASIISR over the DP5380 registers is that all interrupt information is available in one register, and every interrupt is flagged.** To check DMA the user need only read the EASIISR and ensure that the only flag active is end of DMA. The user should note that SCSI reset causes the device to revert to MODE N, from which the EASIISR can not be read, so is not flagged.

The final EASI flag test is the 'conventional' end of DMA flag in the Bus and Status Register (EASIBSR). This flag, the interrupt flag and the flag to show no phase mismatch has occurred must be the only bits active. The final loopback DMA test is to ensure that memory location TSTBYT contains the correct data.

An error in Loopback DMA test causes error signal 3.

3.3.5 DP8490 Initialization for Selection

The DP8490 initialization is very similar to that of the 5380, even calling the same routine, *main__* , to respond to selection. However, this device stays in MODE E and uses these enhancements to only allow interrupts on selection and parity by setting the EASIIMR. Selection and parity are the only valid interrupts at this point.

At the end of the DIAGA routine program control returns to the 'jr Here' instruction, which it will continually enact until a parity or selection interrupt causes a jump to *main__*.

3.4 ERROR HANDLING

Throughout all software for this board the error handling is the same for errors considered non-recoverable, which includes all errors in diagnostics. On error the board continually displays an error number, as a four bit binary code, using the LED. The subroutine *ERROR* carries out this function.

On error register 'l' should be loaded with the error number and routine *ERROR* called. This routine then occults the LED for ½ second to display a zero, 1 second to display a one, with the LED on for ½ second between flashes. The four bits, most significant bit first, are repeatedly displayed between 2 second intervals, during which time the LED is on. The timing delays are generated using a routine *DELAY*, which gives a number of ¼ second delays, the number of delays being determined by the value in the 'l' register.

The following list shows the possible errors in diagnostics. The run-time software section contains a similar listing of its error codes.

Error 0
  The DMA can not be accessed.

Error 1
  The EASI can not be accessed.

All other diagnostic errors concern a DP8490

Error 2
  An error has occurred in the assertion of SCSI signals in loopback test mode.

Error 3
  An error has occurred during the loopback DMA transfer.

3.5 EASIO.SRC

This file contains public assembly language routines, which can be called by either the diagnostics or run-time software, to implement low level commands. As these routines may need to be called by routines written in 'C' any variables passed to the routines are passed in the 'hl' register pair, then in 'de', then 'bc' and then on the stack. *ERROR* and *DELAY* both use the 'l' register to pass a variable. Data passed back to the calling routine is returned in the accumulator.

Functions *'read'* and *'write'* implement general purpose I/O register accesses, while *'dmaread'* and *'dmawrite'* handle the special case of the 16-bit DMA registers. These require two accesses of the same address. *'intA'* initializes the processor interrupt mask, with a 'pseudo' I/O write, which selects a register internal to the processor, setting the mask to only allow interrupts on RSTA. This function also enables the interrupts, which can be done by *'eni',* with *'dsi'* disabling interrupts.

'IDtest' is used by the run-time software, during selection, to read the number of bits active on the bus. This routine checks the number of high bits in the byte passed in the 'l' register, returning the value in the accumulator.

3.0 Diagnostic Software (Continued)

Function *'restrt'* is a general purpose reset routine, which can be called on an error condition. This causes every device on the board to be reset, and the diagnostics to be rerun, thus clearing all memory and reinitializing the stack.

4.0 Run-Time Software

4.1 INTRODUCTION

Following the diagnostic software must be a program which will control all SCSI bus transfers, beginning with selection. This program is written in 'C', using a PARAGON 'C' cross compiler for an NSC800. The relevant files for this section are SCSI.C, COMMAND.LIB, PROCESS.LIB, DMA.LIB, PRINT.LIB, ARBITRAT.LIB, SYM.H, CONST.H and COMMANDS.H, all of which are on the supplied floppy disk.

SCSI.C contains the main program, called *'main'* by 'C' and *'main__'* by assembler. The '.LIB' files contain the functions called by *main* and the '.H' files contain the constant values used by all of these files.

4.2 MANDATORY PHASES

This section outlines all the bus phases and transfers which a target must support. It would be perfectly legal for a target to respond to a selection, fetch a command block from the initiator, and then returns status and message bytes before releasing the bus. Although no process would be actuated this would be a legal sucession of events.

4.2.1 Selection Response

main( ) is the program jumped to when an interrupt is generated as the program circles the continuous loop at the end of diagnostics. This routine should only be entered after a selection interrupt, if any other interrupt is active it is considered an error. The function *select( )* checks an interrupt to ensure it is valid.

The type of device installed is checked by reading the 'DP8490' flag, and this determines how the interrupt is verified. In a DP5380 a selection interrupt is determined by the absence of any other interrupt, with $\overline{\text{SEL}}$ active. The user must check the EASIBSR to ensure that no error flags are active, only the INT and PHSM (phase match) bits.

Any other flag will cause error number 4 to be displayed.

After reading the EASIBSR and finding no interrupt flags the user knows the interrupt should be a selection. The only other unflagged interrupt is a reset, which would have been handled by the low level interrupt service routine. Therefore the EASICSB register must also be read, and if $\overline{\text{SEL}}$ is inactive an error condition exists.

This causes error number 5 to be displayed.

In MODE E the user need only read the EASIISR to determine which interrupt is active, including selection. All interrupts, other than parity and selection, should be masked off, so any error concerns only these two flags.

A SCSI parity error is displayed as error number 8, if the selection flag is not active error number 9 is displayed.

The common tests for MODES E and N both concern the EASICSD; error 6 shows that the correct SCSIID bit was not active on the bus; error 7 shows there was more than two bits active on the bus. During selection an initiator is only allowed to assert two bits on the bus, its own ID and the target ID.

The target must assert $\overline{\text{BSY}}$ to show it has recognized the selection, then when the initiator deasserts $\overline{\text{SEL}}$ the selection phase is complete.

4.2.2 Command Phase

A selection phase is followed by a command phase, where the target reads a command block from the initiator. This command block specifies the actions the initiator requires the target to execute, plus the length of any data transfers requested. This board only allows six byte command blocks, which are transferred into the target by function *fetch__cmd( )*.

The command block is transferred using programmed I/O; that is each byte is individually handshaken under processor control. The bus phase must be Command Out (out and in always refer to the initiator, so Command Out is a command block sent by the initiator), bus phase being set in the EASITCR. To transfer a byte (see *Figure 4.1*) the user must assert $\overline{\text{REQ}}$, then wait for $\overline{\text{ACK}}$ to go active to show data is available. On $\overline{\text{ACK}}$ the target can read the data on the bus, then deassert $\overline{\text{REQ}}$ to show it has received the data. When the initiator deasserts $\overline{\text{ACK}}$ the byte transfer is complete.

4.2.3 Status Phase

The target must send a status byte to the initiator during the status phase, at the termination of each command. A list of status codes is given in COMMANDS.H. *status( )* is the function which enters a Status In phase, and transfers the code.

The EASITCR must be written with the Status In phase and the EASIODR with the status code. This code should be asserted onto the bus, remembering to keep $\overline{\text{BSY}}$ asserted, and if neccessary the MODE E bit. $\overline{\text{REQ}}$ is then asserted to show data is available. The initiator should assert $\overline{\text{ACK}}$ when it has read the data, allowing the target to deassert $\overline{\text{REQ}}$ and take the data off the bus. The transfer is complete when the initiator deasserts $\overline{\text{ACK}}$ (See *Figure 4.2*).

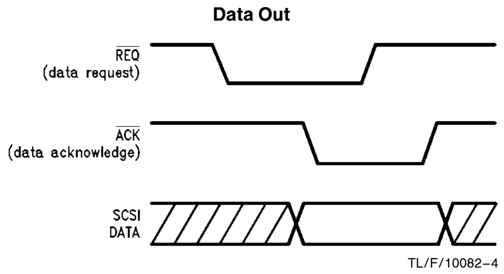

Data Out
$\overline{\text{REQ}}$ (data request)
$\overline{\text{ACK}}$ (data acknowledge)
SCSI DATA
TL/F/10082–4

FIGURE 4.1

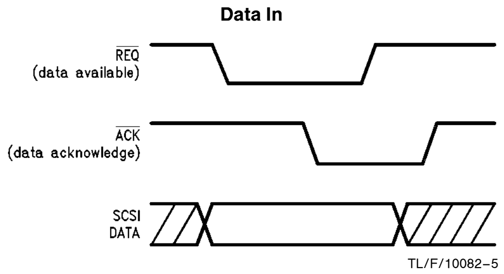

Data In
$\overline{\text{REQ}}$ (data available)
$\overline{\text{ACK}}$ (data acknowledge)
SCSI DATA
TL/F/10082–5

FIGURE 4.2

4.0 Run-Time Software (Continued)

4.2.4 Message Phases

A selection must be terminated by the target entering the Message In phase, and sending a relevant message code (as listed in COMMANDS.H). The target follows the Status phase with the Message In phase, usually to send the COMMAND COMPLETE message. This indicates valid status exists, so the target can release $\overline{\text{BSY}}$ to free the bus. *messin( )* enters the Message In phase and sends the message using programmed I/O, as in *status( )*.

The only message which must be supported is COMMAND COMPLETE. If a device on the bus supports other messages it indicates this by responding to or asserting $\overline{\text{ATN}}$. An initiator asserts $\overline{\text{ATN}}$ if it has a message for the target. It is common for an initiator to assert $\overline{\text{ATN}}$ during selection and, if the target responds by entering a Message Out phase, it sends the IDENTIFY message. This establishes whether the target can respond to a greater set of messages, and whether the initiator supports disconnection. It shows this by setting bit 6 of the message. In a disk controller IDENTIFY would also establish the path by including a Logical Unit Number (LUN), but the printer controller uses all LUN's at this SCSIID.

getmes( ) enters a Message Out phase and fetches a message using programmed I/O. Although this software only supports single byte messages it must be prepared to accept messages from the initiator of up to the maximum length, 256 bytes. If an initiator wishes to send further bytes after the first it must keep $\overline{\text{ATN}}$ asserted, only deasserting after the final byte has been transferred. *getmes( )* will handshake up to 256 bytes, after which if $\overline{\text{ATN}}$ is asserted it will be considered an error. Only the first byte is used in determining the message sent.

The use of messages during disconnection, arbitration, reselection and in error handling will be discussed in those sections of this document.

4.2.5 Command Termination

After sending status and message codes to the initiator, the target should then release $\overline{\text{BSY}}$ to free the SCSI bus. However, before allowing a bus free phase the target must initialize itself for the next selection, as in function *reset( )*.

The interrupt jump table is loaded with *main( )* and the EASISER with the SCSIID. For a MODE E device the interrupt mask is set. For both types of device the interrupts must be reset, and $\overline{\text{BSY}}$ deasserted. Interrupts are disabled at the start of this function, so must be enabled after calling *reset( )*.

4.3 COMMAND PROCESSING

After the command block has been fetched the command has to be determined and executed. The printer command set is shown in COMMANDS.H.

process_cmd( ) reads the first byte of the command block which contains the operation code. This determines what actions are taken. If a command has been sent which this software does not recognize, a status of CHECK CONDITION is returned, with sense set to ILLEGAL REQUEST.

4.3.1 Test Unit Ready

TEST UNIT READY will be sent by an initiator before a print to ensure the printer is on, on-line, has paper and is not in an error condition. On this command the function *ck_printer( )* is called to read the printer signals through Port B of the PIO and check for errors.

If the error line is not high the printer is operational, so the status is GOOD, and the sense is set to NO SENSE (no error). If there is an error the paper error line must be checked. If this is low (active low input) the printer is out of paper and status of CHECK CONDITION is returned, with sense set to MEDIUM ERROR. If it is not a paper error the printer is assumed to be off or off-line, so status is CHECK CONDITION with sense UNIT ATTENTION.

4.3.2 Request Sense

An initiator will send a REQUEST SENSE command after the target has returned a status of CHECK CONDITION. The sense data is sent to the initiator in an effort to understand an error condition, and if possible recover from it. Byte 4 of this command block contains the length of an extended sense message, which must not be greater than 4, or sense is set to ILLEGAL REQUEST and CHECK CONDITION status returned. This software only supports four bytes of sense data.

Sense data is sent to the initiator using single mode DMA. The DMA is initialized by *send_sense( )*; it is reset by a master clear, the mode set, the DMA mask written and the address and word-count loaded. The function *single_dma_in( )* (explained in section 4.4.1) enters the correct phase (Data In) and transfers the data.

4.3.3 Reserve Unit

In a multi-initiator system a printer controller must be reserved before a print commences, or the data from two different initiators may be mixed. This command stores the initiator's ID in a variable called 'reserved', and on subsequent selections only this initiator may execute commands.

Any initiator wishing to reserve the unit must put its own ID on the bus during selection, along with the target's ID, so this software knows which initiator is reserving the unit. If the initiator's ID is not on the bus it can not reserve the unit, and since this is a prerequisite of printing, it cannot use the printer.

If an initiator attempts to reserve this unit without making its ID available sense is set to ILLEGAL REQUEST and status of CHECK CONDITION returned. If another device attempts to reserve the board when it is already reserved status returned is RESERVATION CONFLICT.

4.3.4 Release Unit

This is the reciprocal command to the previous, freeing the printer for other initiators after a print has been completed. If an initiator other than the reserver attempts this command a status of RESERVATION CONFLICT is returned; if this is attempted when there is no reservation current the returned status is CHECK CONDITION with sense set to ILLEGAL REQUEST.

4.3.5 Print

Since transferring data to a printer is a very slow process, typical Epson Fx range 80 cps–160 cps, the print command transfers the data to a buffer, leaving it to be printed later. Bytes 2, 3 and 4 of the command block contain the data length, byte 2 the most significant. The use of three bytes to define the size means that in theory block transfers of up to 16 MB are allowed. This software could support blocks of up to 64 kB, bytes 3 and 4 are read, but blocks are limited to a size BUFFLIM, which will be determined in section 4.4.3.

Possible errors in a print occur when the unit is not reserved, the data length is set to zero, or the block size is too large. The block size is too large if byte 2 of the command block is active or if the data length is greater than BUFFLIM. In response to an error the transfer is cancelled and status of CHECK CONDITION is returned with sense set to ILLEGAL REQUEST.

The data is not transferred unless sense is set to NO SENSE. If an error has occurred in the printer sense will have been set to indicate the error source. The error condition must be rectified and TEST UNIT READY sent to reset the sense data.

print__cmd( ) is the function which transfers data into a buffer using block mode DMA. The print buffer is a circular queue, allowing the user to take data off the front and put data on the rear (see *Figure 4.3*).

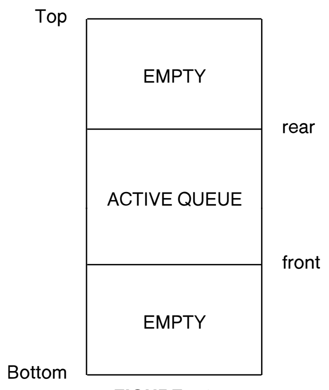


FIGURE 4.3

front points to the next byte to be printed, rear points to the next available byte for entering data. When front equals rear the queue is empty, when rear is one less than front the queue is full.

The buffer limits are called top and bottom. When either the front or rear pointers reach the top the next increment takes them to the bottom. Thus the queue may look like *Figure 4.4.*

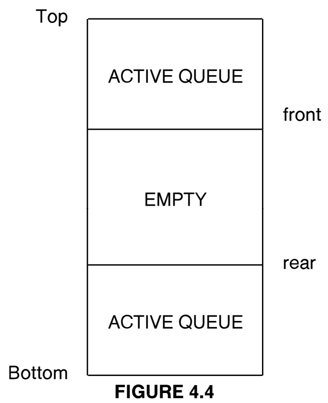


FIGURE 4.4

When *print__cmd( )* is first called after a reset it must set up the queue, defining top and bottom, and setting front equal to rear equal to bottom. Before any transfer the data length must be checked to ensure that the DMA will not take rear past top. If it would the data must be transferred in two blocks; one to the top of the queue, and one starting at the bottom of the queue. This routine should not be called unless there is sufficient free space for the size of transfer. Function *dma__data( )* sets up the EASI and DMA for a block mode transfer, enters the Data Out phase and calls *dma( )* to handle the transfer. This will be more fully explained in section 4.4.

4.3.6 Flush Buffer

The purpose of this command is to allow an initiator to terminate an unwanted print. An initialization pulse is sent to the printer, in case it has its own buffer, and the SPC buffer is cleared. This is done by setting front equal to rear equal to bottom.

4.4 DATA TRANSFERS

This section is concerted with the way in which software controls DMA transfers, both block and single mode, and how the print data block length is determined.

4.4.1 Single Mode DMA

The four bytes of sense data are sent to the initiator using single mode DMA. *send__sense( )* sets up the DMA to transfer four bytes of data from the sense buffer to the EASI, leaving function *single__dma__in( )* to complete the transfer.

The phase, which in this case is Data In, is determined by the calling function, and the EASIMR2 initialized for a single mode transfer. Parity checking is enabled with interrupt on $\overline{\text{EOP}}$ or parity error. The data bus is asserted and the routine *dma( )* is called to handle the transfer.

Since both single and block mode DMA transfers, either in or out of the initiator, have large portions of code that are common, a function can be written for general purpose DMA handling. This function, *dma( )*, sets up the interrupt response and makes a write to the register which initiates the required type of transfer, which could be target receive or target send etc. *dma( )* checks the success of the transfer when the transfer is complete.

4.0 Run-Time Software (Continued)

In MODE E the interrupt mask can be set to only allow parity, $\overline{\text{EOP}}$ and DMA phase mismatch interrupts. The interrupt jump table is loaded with the starting address of a service routine, which sets a flag to show an interrupt has occurred. The program can then sit in a loop waiting for this flag to go active.

After the interrupt the cause has to be checked to ensure the transfer was successful, but this checking depends on the device installed. For a DP8490 the user can read the EASIISR and if any flag other than end of DMA is active an error has occurred. On such an error, status is set to CHECK CONDITION with a sense of ABORTED COMMAND. The interrupt mask can be reset, masking off end of DMA and DMA phase mismatch, and the DMA mode in the EASIMR2 disabled. The interrupt service routine is set to jump to the general interrupt handler *gen__int( )*. The interrupt must then be reset and the processor's interrupts enabled.

In MODE E true end of DMA is detected, but in MODE N the end of DMA interrupt is generated when $\overline{\text{EOP}}$, $\overline{\text{DACK}}$ and $\overline{\text{IOR}}$ or $\overline{\text{IOW}}$ are active concurrently. True end of DMA, after the transfer is complete, must be detected by software. $\overline{\text{REQ}}$ and $\overline{\text{ACK}}$ must both be inactive on three successive samples for true end of DMA. **This additional code makes the MODE N DMA routine slower**. After detecting this the user can check the end of DMA flag is active in the EASIBSR, giving status CHECK CONDITION and setting sense ABORTED COMMAND if it is not. The DMA bit is then reset and *gen__int( )* called to ensure there was no parity or phase errors, and to reset the interrupt service routine.

On return to the function *single__dma__in( )* it will deassert the SCSI bus and return to the calling routine.

4.4.2 Block Mode DMA

blk__dma__in( ) is the equivalent routine to *single__dma__in( )* initiating a block mode transfer to the initiator. *send__sense( )* could call this routine, after setting the DMA for block mode, and transfer the data using block mode, with no difference to the other software. The only difference between these two routines is the setting of the EASIMR2 BLK mode bit.

print__cmd( ) is used to transfer the data to be printed from the initiator. This function checks that the transfer will not exceed the constraints of the queue and calls *dma__data( )* to initiate a block mode transfer. *dma__data( )* sets the EASI and DMA registers for a block mode target receive, with the data size set by *print__cmd( )* and the destination starting address equal to rear. *dma( )* is again used to handle the interrupts.

4.4.3 Determination of Data Block Length

The block of data to be transferred for a print has been specified as a maximum length of BUFFLIM, but the value of this constant has to be evaluated. The two main considerations in this are the latencies of the bus and the SPC.

Since this board is a printer controller it will be of a very low priority. Thus the device should not hold the bus for too great a time per transfer, as this would slow down initiator's accesses of a high priority peripheral. The second consideration is the time that the target takes from selection to entering a data phase, the data block must take longer than this to transfer, or the command is inefficient. Measurements taken on various models of personal computers showed that, with an Advanced Storage Concepts ASC-88™ SCSI Host Adaptor, the time taken from the initiator asserting $\overline{\text{SEL}}$ to the target commencing the data transfer is approximately 3 ms.

The block mode DMA rate was measured as between 200 kB/s and 500 kB/s limited by the DMA in the PC. A block length of 2 kB was selected, since this would take between 4 ms and 10 ms to transfer. Thus the data transfer time is greater than the selection to data phase time, but the overall time on the bus in not too long.

4.5 PRINTING DATA

Due to the inherent slowness of printers data can not be printed while the controller is in command of the SCSI bus. To prevent tying up the bus data is stored in a buffer and printed after $\overline{\text{BSY}}$ has been released. The function *printit( )* handles the transfer of data to the printer.

printit( ) checks the printer is not in an error condition, and transfers the byte of data at address front to the printer. The data is transferred through Port A of the PIO with BUSY and $\overline{\text{STROBE}}$ used to handshake the data. BUSY must be high and $\overline{\text{STROBE}}$ pulsed low a minimum of 0.5 μs for the printer to accept data. front is incremented, to point to the next byte to be transferred. To print out the queue *printit( )* is called until front equals rear.

If the function does detect an error it sets sense to the appropriate value, and implements any outstanding reconnection. The printer is continually polled to determine when it comes back on-line, at which time the print continues.

While the board is printing it must still be available for selection. An initiator can reset every device on the SCSI bus if a target does not respond to a selection within a Selection Timeout, normally 250 ms. The interrupt service routine has been set to jump to *main( )*, but the program is currently in *main( )*, so this function must be re-entrant.

main( ) is written in such a way that it will only process commands if it is unreserved and not printing, or reserved by the current initiator but not printing. If the board is reserved, but not by the current initiator, a status of RESERVATION CONFLICT is returned. If the board is printing, and the reserving initiator attempts to send it another command a status of BUSY can be returned.

However, the controller has a 30 kB print buffer, of which only 2 kB would be used at a time. It would be much more efficient to continue executing commands until the buffer is full. One method of achieving this can be seen in *outbuf( )*. In this function a flag, called next, is set if the free space in the queue is greater than BUFFLIM. If a selection occurs while a print is on, this flag is checked and the command processed if it is active. If it is inactive a status of BUSY is returned. This method has the disadvantage that the initiator must continually poll the SPC to determine when it is ready to accept data. This 'loads' the SCSI bus and slows down the print, since the SPC will be responding to selection. It also restricts other device's use of the bus. The alternative method of utilizing the buffer space is to use disconnection and reconnection.

4.0 Run-Time Software (Continued)

4.6 DISCONNECTION AND RECONNECTION

4.6.1 Disconnection

If an initiator sends the IDENTIFY message to the target, it can indicate that it supports disconnection by setting bit 6 of this message. If this bit is not set, or the message not sent, the initiator is assumed not to support disconnection, and no disconnection is attempted. This software uses disconnection in two places; 1) the board is reserved, currently printing and selected by the reserving initiator; 2) the board has been released, but not finished printing.

If the board is busy printing it can be advantageous to disconnect from the reserved initiator after reading in the command block, and then reconnect having ensured that the command can be implemented i.e., there is enough free space in the queue. For any other initiator the response is the same as before, the status of RESERVATION CONFLICT is returned until the unit is released. After this the controller will disconnect from any initiator attempting to select it, until it has enough free space in its buffer.

disconnect( ) is the routine which carries out the disconnection procedure. This can only take place after the command phase and before the data phase, with an initiator that sent the IDENTIFY message with the disconnection bit set. The target follows the Command Out phase with a Message In phase and sends the DISCONNECT message, after which it can drop $\overline{\text{BSY}}$, to release the bus. If the DISCONNECT message is not sent the initiator will treat the deasserting of $\overline{\text{BSY}}$ as an illegal termination of command. Before releasing $\overline{\text{BSY}}$ *exchange( )* is called to store the ID of the initiator and the command it wants to execute. *reset( )* is again used to initialize the interrupts and release $\overline{\text{BSY}}$.

disconnect( ) stores in a variable, recon__data, the amount of print buffer which will be required to execute the command. For a print, this depends on the length of data to be transferred, for any other command the amount is zero. recon__data is used to determine when the target can reconnect to the initiator i.e., once there is enough space free in the buffer.

After disconnection the program will return to printing, at which point it may be selected by another initiator. As it can only disconnect from one initiator at a time it must return a status of BUSY.

In *outbuf( )* it can be seen that reconnection takes place when the free space in the print buffer is greater than the number of bytes to be transferred. *reconnect( )* calls the functions which action the correct reconnection procedure, beginning with arbitration.

4.6.2 Arbitration

Arbitration requires two functions, one for MODE E another for MODE N, due to the fundamental difference in their methods of arbitration i.e., **MODE E is interrupt driven, MODE N is polled**. These two functions, *Narbitrate( )* and *Earbitrate( )*, carry out the same basic operation, arbitrate for the bus until successful, but do so in distinctly different ways.

To arbitrate for the bus a device must wait for a Bus Free Phase, when $\overline{\text{BSY}}$ is continuously inactive for 400 ns with $\overline{\text{SEL}}$ inactive. After a Bus Free Delay of 800 ns the SCSI device should assert $\overline{\text{BSY}}$, and assert its SCSIID bit on the bus. After a further 2.2 μs Arbitration Delay the data bus should be examined, and the device with the highest priority SCSI ID bit asserted wins arbitration.

In MODE N the EASIODR should be written with the SCSIID and the arbitration bit in the EASIMR2 set. The interrupts must be initialized to cause a jump to routine *servn( )*. The EASI will wait for a Bus Free Phase then, after a Bus Free Delay, it will assert $\overline{\text{BSY}}$, and put the contents of the EASIODR onto the bus. The user must poll the AIP bit of the EASIICR to determine when arbitration has begun and check the LA bit to ensure that arbitration has not been lost. The LA bit is set if another initiator asserts $\overline{\text{SEL}}$ during arbitration. If AIP is active and LA inactive the user must examine the EASICSD to determine whether it is the highest priority device arbitrating. If arbitration is lost the arbitration bit in the EASIMR2 must be reset and the whole procedure begun again. The device shows it has won arbitration by asserting $\overline{\text{SEL}}$. An interrupt during MODE N arbitration is treated as a selection, so *servn( )* enables parity, since there is no parity checking during arbitration, and calls *main( )*.

In MODE E arbitration is interrupt driven, allowing the board to continue printing while waiting to arbitrate. For a busy SCSI bus typically many milliseconds, and potentially many seconds, can be taken up arbitrating. In MODE E this time can be utilized, thus increasing the system throughput. For this application the time gained is used to continue printing, in other applications, such as a disk controller, it could be used in data cacheing, overlapped seeks, etc.

In MODE E arbitration the EASIODR must be written with the SCSIID and parity checking cancelled in the EASIMR2. The interrupts are set to allow selection and arbitration interrupts, with the jump table loaded with *serva( )*. This sets a flag to show an interrupt has occurred. The enhanced arbitration is initiated, with a write to the arbitration bit of the EASIEMR. This causes the EASI to wait for a Bus Free Phase; delay a Bus Free Delay; assert $\overline{\text{BSY}}$ and the EASIODR; delay an Arbitration Delay then interrupt the processor. The interrupt causes the flag to be set that shows the state of arbitration should be examined. While waiting for this interrupt the printer carries on printing out data, until front equals rear.

After the interrupt is detected the EASIISR can be read to determine the cause. If it is not an arbitration interrupt parity checking is enabled and the interrupt treated as a selection, by calling *main( )*. If the interrupt is signalling the commencement of arbitration, the procedure is the same as in MODE N, with the LA bit of the EASIICR being examined, and priority determined by reading the EASICSD. If arbitration is lost, interrupts must be reset, the arbitration bit in the EASIEMR reset, and the whole procedure begun again. If arbitration is successful the user asserts $\overline{\text{SEL}}$.

Function *printarb( )* is used to print data during arbitration in the same way as *printit( )* does normally. The difference here is that the arbitration interrupt should be serviced as quickly as possible.

This routine must be left if an interrupt occurs. Instead of waiting for the printer to come on-line if an error occurs this routine simply does not send the character.

4.6.3 Reselection

When arbitration has been won the disconnected initiators ID and command block must be restored, and the initiator reselected. This reselection is carried out by function *reselect( )*.

The EASISER is cleared to stop it responding to the reselection and the initiator ID bit written into the EASIODR along with the SCSIID. I/O must be asserted to show this is a reselection. The EASIODR is asserted onto the bus and the relevant arbitration bit, depending on MODE reset. This deasserts $\overline{\text{BSY}}$. Parity checking is enabled, and the board waits a selection timeout delay of 250 ms for the initiator to respond, by asserting $\overline{\text{BSY}}$. If it does not, $\overline{\text{RST}}$ is asserted, resetting the whole SCSI bus.

If the initiator does respond, the target must assert $\overline{\text{BSY}}$, then deassert $\overline{\text{SEL}}$ and take the EASIODR off the bus. For MODE E the interrupt mask can be set, and the arbitration interrupt reset. Reselection is now complete.

4.6.4 Reconnection

After *reselect( )* the IDENTIFY message is sent by the target with the disconnect bit set. The command that was previously sent is processed, if this message is received successfully. After processing the command status and the COMMAND COMPLETE message are returned. *reset( )* is again used to set the interrupts and release $\overline{\text{BSY}}$.

If a print was not current at the time of reconnection, and the reconnected command was PRINT, this is handled within *reconnect( )*. As long as the status is GOOD the print is carried out. However, *reconnect( )* will be mostly called from *outbuf( )*, when the queue has enough free space to process the outstanding command. Any reconnection missed by *outbuf( )* is captured in *main( )*.

4.7 ERROR HANDLING

Throughout this document the handling of errors has been discussed as the errors arose, but there are some remaining to be discussed. These are phase or parity errors during command transfers and message errors.

4.7.1 General Error Handling

While this board is connected to the SCSI bus, with $\overline{\text{BSY}}$ active, *gen_int( )* is generally used to handle interrupts. This routine responds to an unexpected interrupt by checking the parity and phase flags, in the EASIBSR for MODE N, the EASIISR for MODE E. It then resets the interrupts, before setting appropriate flags. Sense is set to ABORTED COMMAND with a status of CHECK CONDITION if an error has occurred.

After calling *select( )*, to respond to selection, *main( )* then calls function *set_up( )*, which sets the mask and jump table to respond to an interrupt. This is also the routine which checks to see if $\overline{\text{ATN}}$ is asserted. If it is *getmes( )* is called to enter the Message Out phase and fetch the IDENTIFY message. *messout( )* processes the message. This is where the target determines if the initiator supports disconnection.

If an error occurs during *select( )* or *set-up( )*, detected by *gen_int( )*, the board sends relevant status and a message of COMMAND COMPLETE, before releasing the bus. No attempt is made to recover from the error. Similarly, if a phase error occurs during the command phase, status and sense are returned. However, if a parity error occurs during a command phase, the target can attempt a recovery by sending the RESTORE POINTERS message. This message instructs the initiator to reset the command pointer to the beginning of the command block, allowing the target to re-enter the Command Out phase and re-transfer the command block. If there is a parity error again, status is returned, along with the COMMAND COMPLETE message. If the second transfer is successful, the command execution continues as normal.

4.7.2 Message Errors

If an initiator wishes to respond to a message sent by the target it indicates this by asserting $\overline{\text{ATN}}$, before releasing $\overline{\text{ACK}}$ to finish the transfer. The target should then enter the Message In phase, and transfer message bytes until $\overline{\text{ATN}}$ goes inactive, up to 256 bytes. If the initiator attempts to send more than this the target will send the MESSAGE REJECT message and terminate the command with status of CHECK CONDITION and sense set to ABORTED COMMAND. If a parity error occurs during the message transfer, the target must wait until the transfer is complete, then instruct the initiator to resend all previous message bytes, by asserting $\overline{\text{REQ}}$ before changing phase. If the parity error occurs again the command will be terminated with status of CHECK CONDITION and sense ABORTED COMMAND.

The relevant functions for message phases are *messin( )*, *getmes( )* and *messout( )*. *messin( )* is used to send a message to the initiator, while *getmes( )* fetches a message from the initiator. *messout()* processes the message sent by the initiator. The supported messages are now explained.

IDENTIFY: establishes the use of a greater message set, indicates the ability to support, or not support, disconnection and gives a LUN number, if necessary.

ABORT: if the reserving initiator sends this message to the target it causes the board to be reset.

BUS DEVICE RESET: This is similar to ABORT, except any initiator can implement it, resetting the board.

MESSAGE PARITY ERROR: On receiving this message the target attempts to resend the last message sent, and if this fails, terminates the command with status of CHECK CONDITION and sense set to HARDWARE ERROR.

MESSAGE REJECT: the targets response to this message is determined by the message being rejected. If the last message sent was COMMAND COMPLETE, the MESSAGE REJECT is ignored, since the initiator must support the mandatory message. A MESSAGE REJECT in reply to a DISCONNECT causes the disconnection to be cancelled. If it was in reply to a MESSAGE REJECT sent, the command is terminated with a status of CHECK CONDITION and sense set to HARDWARE ERROR. If the message sent was IDENTIFY (during reconnection) the target immediately goes to the Bus Free Phase and aborts the command. No Status or Message In phases are attempted, though sense is set to HARDWARE ERROR.

4.0 Run-Time Software (Continued)

It should be noted that *messin( )* has been written to only allow *getmes( )* to be called, from inside *messin( )*, once. Alternatively on parity error the target and initiator could eternally cycle, sending each other the MESSAGE PARITY ERROR message.

4.7.3 Non-Recoverable Errors

The following are errors which occur **during selection**, so severe that system operation is terminated, with an error code displayed on the LED.

The first two concern a DP5380.

Error 4
  Wrong interrupt flags active.

Error 5
  SEL inactive.

These errors concern either a DP5380 or a DP8490.

Error 6
  SCSIID bit not active on bus.

Error 7
  More than two bits active on bus.

The final errors are for a DP8490 only.

Error 8
  SCSI parity error.

Error 9
  Select flag inactive.

5.0 User Guide

The SCSI Printer Controller (SPC) is a Small Computer System Interface target board, which can use either the DP5380 Asynchronous SCSI interface (ASI) or DP8490 Enhanced Asynchronous SCSI Interface (EASI). It can be selected and used by an initiator as described in the ANSX3.131-1986 SCSI standard as defined by the ANSI X3T9.2 committee. This document will explain the installation of SPC, and show how it can be used in a system. By way of example a description is given of how the SPC can be used with an Advanced Storage Concepts ASC-88 IBM PC-SCSI Manager II™ host adaptor.

5.1 INSTALLATION

This section explains how the board must be set up before use, and the type of connectors required to interface to it. Any reference made to an EASI also applies for an ASI.

5.1.1 POWER

The SPC can be installed in a Personal Computer (PC) where it takes power from the backplane. There are two connections to +5V and two to ground. These are the only connections made to the backplane. Alternatively power can be taken from the available connector block *(Figure 5.1)*.

When the board receives power the LED will come on, and stay on if the board passes its self diagnostics. If it fails an error message will be displayed by the LED, indicating the source of error. This will be further explained in section 5.1.5. If the LED does not come on some fatal error has occurred.

5.1.2 SCSI CONNECTOR

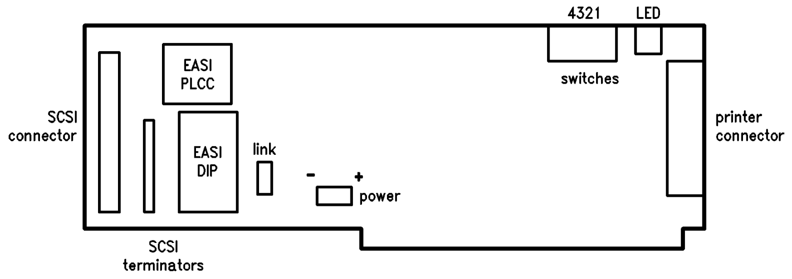
The SCSI bus should consist of a 50 way flat ribbon cable a maximum of 6.0 meters long. Both ends of this cable should have all SCSI lines terminated, with a 330Ω resistor to ground and a 220Ω resistor to power. SCSI devices are daisy chained along this cable, with SCSI signals common to all devices.

The SPC contains sockets for two DP8490 or DP5380 devices, one a PLCC socket the other DIL. **Only one of these sockets should contain a device**.

To comply with the regulations on terminating resistors six SIL resistor packs are available between the SCSI connector and the EASI DIL package *(Figure 5.1)*. If this board is not to be used at the end of the SCSI cable all of these six packs must be removed. The resistors have been socketed for this purpose.

An option available in the SCSI standard is to supply terminator power on the cable, so terminators at either end of the bus can use the same power. If the device at one end of the bus is unpowered its terminators can receive power from the cable, and the bus will operate correctly. It should be noted that other manufacturer's CMOS devices will not work in this configuration since if not powered they pull the SCSI lines low. National Semiconductor's DP5380 and DP8490 have special input protection to prevent this.

Pin 26 on the SCSI connector is the terminator power pin, which can be left floating by leaving the jumper in position 2, see *Figure 5.2*. By moving the jumper to position 1 the terminator power is supplied to this pin. *Figure 5.3* shows the two possible configurations. Terminator power is fed through a Schottky barrier diode to prevent a backflow of power into the board.



TL/F/10082–6

FIGURE 5.1

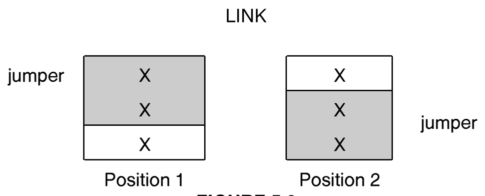


FIGURE 5.2

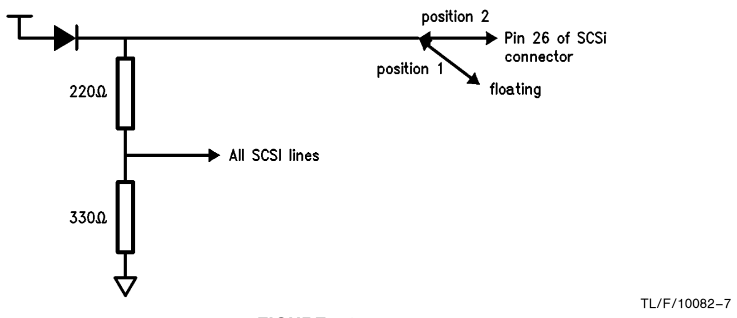


FIGURE 5.3

5.1.3 SWITCH BLOCK SETTINGS

The switch block is included to allow the user to select the SCSI ID of the SPC. This is used to identify the board during selection phases and determine its priority in arbitration. The SCSI ID is read in as a three bit binary number and converted in software to an eight bit pattern, with one bit active. For an ID of 0 the least significant bit is active, for an ID of 7 the most significant bit is active.

The three bit number is taken from the switch block (see *Figure 5.1*), using switches 1, 2 and 3. 1 is the most significant bit, 4 is unused. These switches should be set to give a unique ID for this bus. An ID of 0 is suggested, making the printer the lowest priority device on the bus. These switches must be set up before power is applied, as they are only checked during the board diagnostics.

5.1.4 PRINTER CONTROLLER

The printer connector is a standard IBM 25 way 'D' type connector. The SPC software controls data transfers to the printer using the BUSY and $\overline{\text{STROBE}}$ signals.

5.1.5 ERROR MESSAGES

Non-recoverable errors cause a four bit binary number to be displayed by the LED. This indicates the source of error.

The LED displays the number by occulting for 1 second to show a 1, ½ second to show a 0. The code is displayed most significant bit first, with the LED on for 1 second between bits. The error number is repeatedly displayed, between breaks of 2 seconds when the LED is on.

The error numbers, and their cause, are shown below. The first four errors are generated during the board diagnostics, the others occur when a selection fails. The SCSI controller will be referred to as an EASI, unless the error is specific to either a DP5380 or DP8490. These possible differences are due to the DP8490's more extensive testing, and because it handles a selection differently.

Error 0
The DMA could not be accessed. This is possibly a damaged device.

Error 1
EASI can not be accessed. Device could be damaged, or not properly terminated.

Error 2
DP8490 has failed loopback test. Device could be damaged.

Error 3
DP8490 has failed the loopback DMA test. In this test a DMA transfer to memory from the DP8490 is attempted. Failure here could indicate an error in memory, processor, DMA, DP8490 or PALs.

These first four errors concern problems internal to the board. The following errors indicate a bus problem that occurred while the board was waiting for selection.

Error 4
DP5380 error flag active.

Error 5
DP5380 found select line inactive. Possible selection timeout or bus error.

Error 6
EASI ID bit not active on bus. Possible selection timeout or bus error.

Error 7
EASI detected more than two bits active on bus. During a selection phase the initiator can only assert its own ID and the targets ID on the bus. This indicates a bus error.

Error 8
DP8490 SCSI parity error. Error on bus.

Error 9
DP8490 select flag inactive. This is board hardware error, possibly in DP8490 or PAL, possibly interrupt line.

For any of these problems the user should try switching the device on and off again. For bus errors the cable and terminators should be checked along with any other devices on the bus.

5.2 DRIVER SOFTWARE

By way of example it will be shown how software can be written to drive the SPC from an ASC-88 host adaptor, installed in a PC. The ASC-88, like many commercially available host adaptors, handles all low level SCSI signal controls. The user controls the command to be implemented by means of a Job Control Block, JCB, which is passed to it.

The software required to use an ASC-88 is on the supplied floppy disk. This disk contains both the source code, explained later, and the executable code. This is called ***printout.exe*** and can be implemented in the form:

printout *filename*

The *filename* supports the MS-DOS use of directories and paths:

e.g., printout a: /ASC /ASC.C

The second executable command, called ***stoprint.exe***, can be used to terminate a print. When this command is executed the SPC flushes the print queue and initializes the printer.

5.2..1 SEQUENCE OF COMMANDS

Although the SPC requires no prescribed sequence of SCSI commands **an initiator must reserve the unit before a print can take place**. Otherwise the sequence of commands specified here indicate how the SPC could be used.

Any print should begin with a TEST UNIT READY command. On receiving this command the SPC tests the printer to ensure it has paper and is not in an error condition. If the status of GOOD is returned the user should then attempt RESERVE UNIT. If the command is successful the SPC will only execute commands from this initiator. This prevents print data from two or more sources being mixed.

The user then sends PRINT commands with a maximum length of data per transfer of 2 kB. If the file to be printed is larger than 2 kB the data must be sent in several blocks. This limit, set to minimize bus latency, is more fully explained in Section 4.4.3.

These PRINT commands transfer the data to the SPC print buffer. To check if an error occurred when the SPC attempted to transfer the data to the printer the user can send a REQUEST SENSE command. If the returned data is NO SENSE the printer is still operational. Any other sense indicates an error.

The final command sent is RELEASE UNIT. This allows the SPC to be used by other initiators.

5.2.2 ASC-88 SOFTWARE

All ASC-88 software is written for a Microsoft® C compiler. File ASC.C contains the main run-time software. ASC STRUC.LIB sets up the structure which is used as a Job Control Block, while ASCCOM.LIB defines the JCBs for particular SCSI commands. The routines used in ASC.C, other than SCSI command calls, are in ASCROT.LIB and UTIL.LIB. CONSTANT.H contains the constants used throughout these files. If the user wishes the SCSI ID of the target board to be anything other than zero they should change the value of *TARGET__ID* in this file, and recompile the code for both commands.

STDIO.H is a Microsoft library containing constants, macro definitions and function declarations for I/O stream operations. This controls opening files, reading files, detecting file ends and closing files. DOS.H, also a Microsoft library, handles the interface to MS-DOS®. This allows the user to set up registers and execute interrupts. The final Microsoft library CONIO.H allows the user to fetch information from the keyboard.

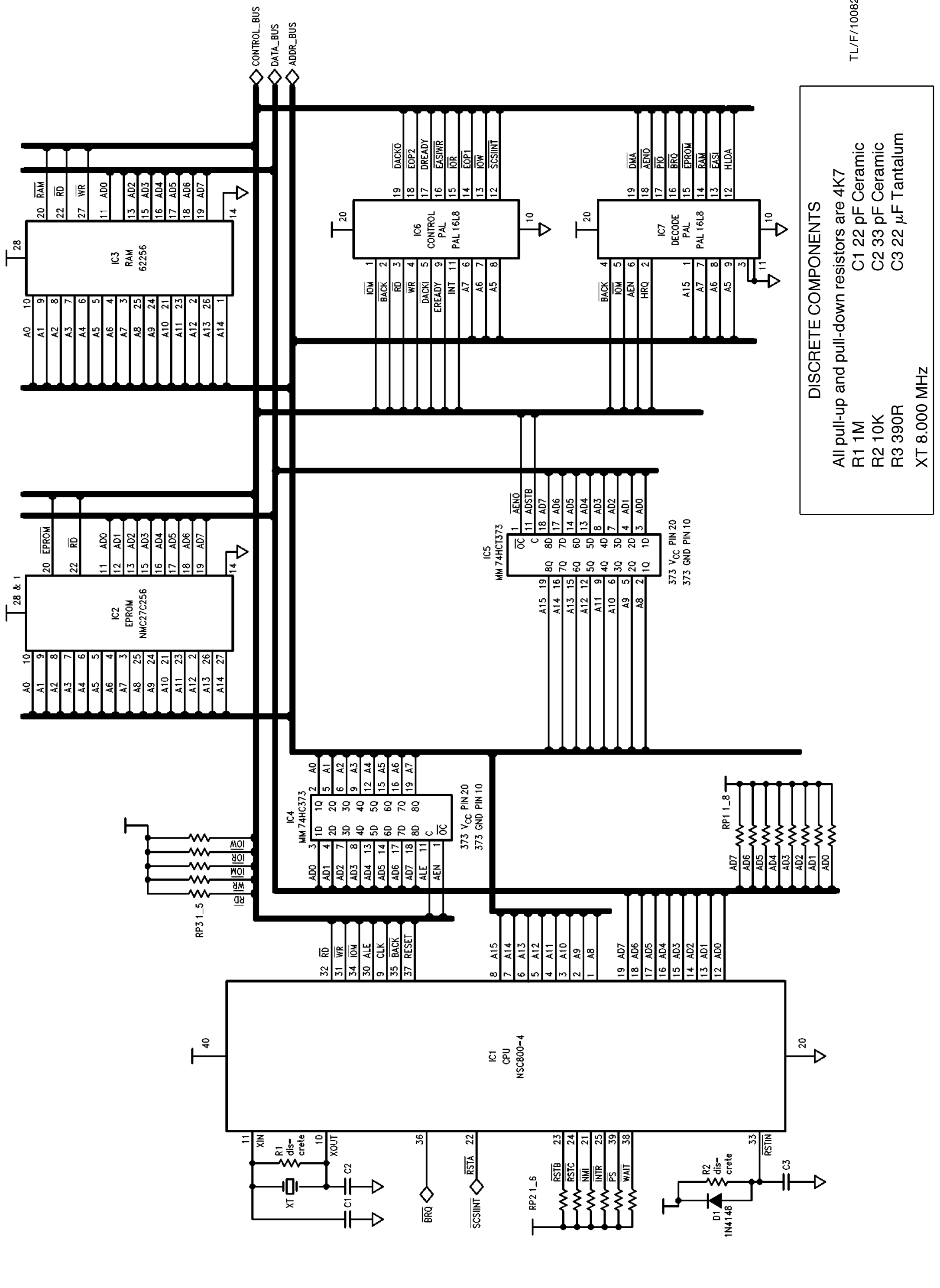
The SCSI-BIOS™ EPROM in the ASC-88 is accessed by generating interrupt number forty. The value in the 'AH' register determines what SCSI-BIOS software is used. SCSI-PRO™ implements the SCSI command specified in the JCB whose starting address is passed in the 'BX' and 'ES' registers.

ASC.C must be compiled and loaded to produce a file printout.exe. When executed this code opens the file specified in the command line and instigates the sequence of commands, outlined in the previous section, to print it out. After these commands the file is closed.

If the printer enters an error state while communicating with the initiator a message will be displayed on screen and the user is given the option of terminating the print. If the print is to be continued the user must correct the printer error and press a key to restart the print. If an error occurs when the SPC has already received all the data from the PC the board will wait until the print error is corrected and continue printing.

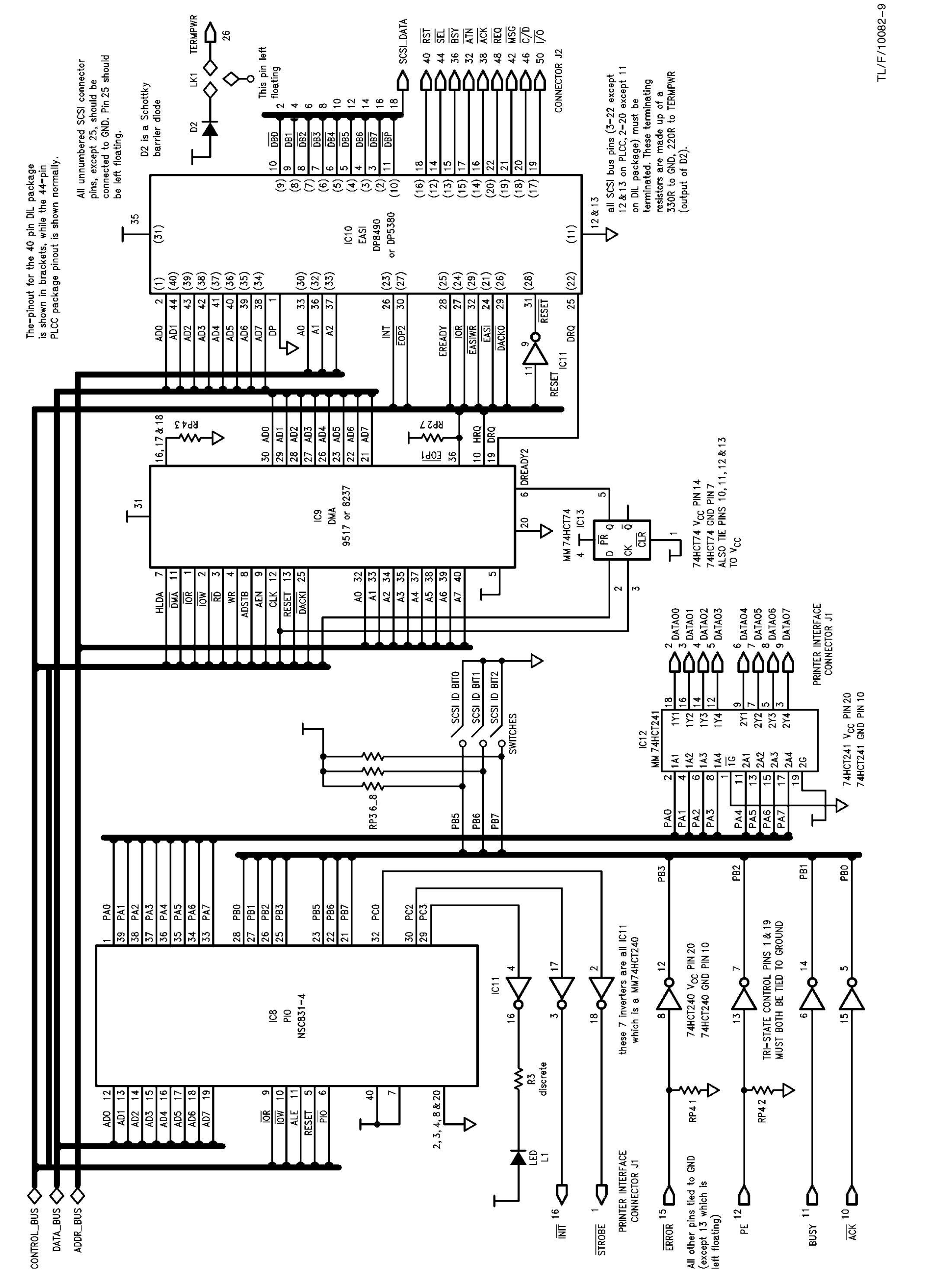
STOPRINT.C contains the run-time software required to terminate a print. This uses the library CHECK.LIB, which checks the success of a FLUSH BUFFER command. To terminate an unwanted long print the user can: take the printer off-line; press escape to leave *printout*; send the *stoprint* command.

Appendix A



DISCRETE COMPONENTS
All pull-up and pull-down resistors are 4K7

R1 1M	C1 22 pF Ceramic
R2 10K	C2 33 pF Ceramic
R3 390R	C3 22 μF Tantalum
XT 8.000 MHz	


The pinout for the 40 pin DIL package is shown in brackets, while the 44-pin PLCC package pinout is shown normally.
All unnumbered SCSI connector pins, except 25, should be connected to GND. Pin 25 should be left floating.
D2 is a Schottky barrier diode
TERMPWR
LK1
D2
This pin left floating
SCSI_DATA
CONNECTOR J2
IC10 EASI DP8490 or DP5380
all SCSI bus pins (3–22 except 12 & 13 on PLCC, 2–20 except 11 on DIL package) must be terminated. These terminating resistors are made up of a 330R to GND, 220R to TERMPWR (output of D2).
IC9 DMA 9517 or 8237
MM74HCT74 IC13
74HCT74 V$_{CC}$ PIN 14
74HCT74 GND PIN 7
ALSO TIE PINS 10, 11, 12 & 13 TO V$_{CC}$
IC12 MM74HCT241
74HCT241 V$_{CC}$ PIN 20
74HCT241 GND PIN 10
PRINTER INTERFACE CONNECTOR J1
SCSI ID BIT0
SCSI ID BIT1
SCSI ID BIT2
SWITCHES
IC8 PIO NSC831-4
IC11
R3 discrete
LED L1
these 7 inverters are all IC11 which is a MM74HCT240
74HCT240 V$_{CC}$ PIN 20
74HCT240 GND PIN 10
TRI-STATE CONTROL PINS 1 & 19 MUST BOTH BE TIED TO GROUND
CONTROL_BUS
DATA_BUS
ADDR_BUS
INIT
STROBE
PRINTER INTERFACE CONNECTOR J1
ERROR
PE
BUSY
ACK
All other pins tied to GND (except 13 which is left floating)

TL/F/10082–9

A SCSI Printer Controller Using Either the DP8490 EASI or DP5380 ASI and Users Guide

AN-563

Lit. # 100563

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: cnjwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408