MANUAL CC-74 DMA-SCSI INTERFACE MODULE

VERSION 1.3 January 1986

Copyright

Disclaimer

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsebility is assumed for inaccuracies. Compcontrol B.V. makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Furthermore, Compcontrol B.V. reserves the right to make changes to any product herein to improve reliability, function or design, without obligation of Compcontrol B.V. to notify any person of such revision or changes. Compcontrol B.V. does not assume any liability arising out of applications or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others.

CC-74 DMA-SCSI INTERFACE MODULE

TABLE OF CONTENTS

PAGE

CHAPTER 1

GENERAL INFORMATION

1.1 Introduction

The CC74 module interfaces the VMEbus and Small Computer System Interface (SCSI) with each other. This manual gives a full description of the hardware and software for users and system programmers. For specific details about the VMEbus, SCSI bus, SCSI protocol Controller or DMA controller, the following documents may also be consulted.

- VMEbus specification manual
- SCSI specification manual (ANSC X3T9.2)
- SCSI protocol controller user's guide (see appendix L)
- SCSI protocol controller data sheet (see appendix L)
- DMA controller data sheet (see appendix K)

1.2 Features

The CC74 VME-SCSI interface module features the high-performance 68450 DMA Controller and the NCR 5386 SCSI Protocol Controller. The SCSI Protocol Controller conforms to the ANSC X3T9.2 Small Computer System Interface standard.

SCSI features:

- Asynchronous data transfer up to 1.5 MByte/s
- Supports both Initiator and Target role
- Parity generation with optional checking
- Supports Arbitration and Reselection
- Controls all bus signals, including Reset
- DMA or programmed I/O transfers
- Block transfers up to 16 MByte

VME features:

- DMA Controller supports I/O to memory, memory to I/O and memory to memory transfers
- Optimal bus width utilization for mixed byte and word transfers
- Programmable Interrupt and Bus Request levels
- Programmable Interrupt vectors
- Interrupts VMEbus when SCSI bus requires service
- 23 bit addresses, 16 data lines and 6 AM code lines
- Highly reliable data transfer by error detect, error interrupt vector and exception features

All SCSI I/O signals are available on the I/O pins of the P2 connector and also on a SCSI-compatible 50-pin flat cable connector.

1.3 General Description

The Small Computer System Interface (SCSI) is a de facto industry standard and is used to interconnect small computers with each other and with intelligent peripherals such as hard disks, flexible disks, magnetic tape devices etc. The standard defines the bus protocol, the bus drivers, cables and connectors and the command set. The CC74 module is a full implementation of the SCSI standard and may act as an Initiator or Target on the SCSI bus. The 8-bit data transfers between the SCSI bus and the VMEbus can be handled by any DTB master or by the local DMA controller. The DMA controller can also be used for memory to memory transfers on the VMEbus and may be used by a RAM disk routine. The CC74 has a four-level Bus Requester and a seven-level Interrupter, which are both software programmable. The normal and error interrupt vector can also be dynamically installed.

CHAPTER 2

SPECIFICATION

2.1 VMEbus Options

Data transfer options:

- DTB MASTER A24,A16; D16,D8
- DTB SLAVE   A24,A16; D16,D8

Requester options:

- Any one of R(0) R(1) R(2) R(3) (DYN)
- RWD

Interrupter options:

- Any one of I(1) I(2) I(3) I(4) I(5) I(6) I(7) (DYN)
- Normal interrupt vector (DYN)
- Error interrupt vector (DYN)

Environmental conditions:

- operating temperature 0-70 degrees C
- max operating humidity 90 %

Power supply requirements:

- 3.0 A max (2.7 A typ) at 5 VDC

Physical configuration options:

- NEXP

2.2 SCSI Bus Options

The following options are implemented on the CC74 module.

- Supports ANSC X3T9.2 SCSI standard
- Supports Arbitration and Reselection
- Single-ended drivers and receivers
- Non-shielded cable option
- Performs both Initiator and Target role
- Parity generation with optional checking
- Controls all SCSI bus signals

CHAPTER 3

INSTALLATION INSTRUCTIONS

3.1 Introduction

This chapter gives all necessary preparation and installation instructions for the CC74 VME-SCSI interface module. The module can be used in VMEbus systems and configuration options are selected by jumpers and switches. All settings are illustrated as seen from the component side with both VMEbus connectors downwards. Jumper blocks are drawn using 'o' for each pin except pin 1 which is identified as '*'.

3.2 Address Selection

Jumper blocks JB1 and JB2 are used for the address modifier selection. JB1 when installed will make the CC74 module respond to supervisory access only.

```
                    *
     JB1:                      supervisory or
                    o          non-privileged access


                    *
     JB1:           |          supervisory-only access
                    o
```

JB2 is used to select standard or short address decoding.

```
     JB2:     *---o   o---o     standard addressing


     JB2:     *   o---o   o     short addressing
```

Switches S1-S4 are hexadecimal switches used to select the base address of the module. S1 selects the most significant nibble, so the address lines A23-A20 are selected by S1, A19-A16 by S2, A15-A12 by S3 and A11-A9 are selected by S4. Note that S4 selects 3 address lines and thus has only 8 significant positions (only even numbers). Also note that S1 and S2 are not significant when JB2 is installed for short addressing.

3.3 DMA Clock Selection

Jumper JB3 is used to select the DMA clock input. When an 8 MHz DMAC is used, a derivative of the system clock can be used. When a 10 MHz DMAC is used, the SCSI Controller clock may be used and for other clock rates the optional oscillator U18 must be installed. Jumper JB3 must be set according to the installed DMAC type.

```
                 *---o

        JB3:     o   o      DMA clock rate defined by U18

                 o   o


                 *   o

        JB3:     o---o      8 MHz DMA clock rate

                 o   o


                 *   o

        JB3:     o   o      10 MHz DMA clock rate

                 o---o
```

3.4 SCSI ID Address Selection

Any Initiator or Target module on the SCSI bus must have a unique ID-bit. Eight ID bits are defined and can be selected with jumper JB4. Only one jumper must be placed at JB4.

```
        ID bit    7   6   5   4   3   2   1   0

                  o   o   o   o   o   o   o   *
        JB4:                          |
                  o   o   o   o   o   o   o   o

                             ID bit 2  selected
```

3.5 External DMA Devices

The jumpers involved with external DMA devices are JB5, JB6 and JB7. The programmable control lines can be used for input or output. JB5 is used to select the direction of the PCL2 control line.

```
                 *   o
      JB5:       |   |     PCL2 configured as input
                 o   o


                 *---o
      JB5:                 PCL2 configured as output
                 o---o
```

JB6 selects the direction of the PCL3 control line.

```
                 *   o
      JB6:       |   |     PCL3 configured as input
                 o   o


                 *---o
      JB6:                 PCL3 configured as output
                 o---o
```

JB7 is used to select the direction of the DONE signal.

```
                 *   o
      JB7:       |   |     DONE configured as input
                 o   o


                 *---o
      JB7:                 DONE configured as output
                 o---o
```

Jumpers JB5, JB6 and JB7 can be removed when not using the option for external DMA devices. When using this option it is necessary that there shall be no conflicts between the programmed direction and the installed jumper settings of the control lines.

3.6 SCSI Bus Termination

The SCSI bus consists of a 50-pole flat cable which may be 'daisy chained' to a maximum of eight Initiator and/or Target devices. Both devices at the ends of the cable should have installed terminating networks and all other devices must not have these networks. The CC74 module has the two relevant resistor networks, RN1 and RN2, installed in sockets, and these networks must be removed when the module is not at the end of the daisy chain.
In some SCSI bus systems, the power of the terminating networks is supplied by a 'Terminating Power Supply' via pole 26 of the SCSI cable. This pin is referred to as 'TRMPWR'. Jumper JB8 is used to select the power source for the on-board resistor networks.

```
              *
              |
JB8:          o       VMEbus power connected to on-board
                      resistor networks
              o
```

```
              *

JB8:          o       SCSI bus 'TRMPWR' connected to on-board
              |       resistor networks
              o
```

CHAPTER 4

THEORY OF OPERATION

4.1 Introduction

This chapter gives a global explanation of the functional blocks as shown in appendix A. The schematic diagrams are given in appendix B. The main functions of the CC74 module are performed by the 68450 DMAC and the NCR 5386 SCSI Protocol Controller. These two parts are fully described in the manufacturers documentation included in appendix K and L.

4.2 DMA Controller

The 68450 DMAC has three modes of operation.
In the MPU mode, the DMAC is selected by an external bus master, through a chip select or interrupt acknowledge. The bus master is writing or reading the contents of the DMAC internal registers.
In the DMA mode, the DMAC is the current bus master and is transferring data or preparing for the data transfer.
In the IDLE mode, the DMAC is in a state other than MPU or DMA mode. A Read/Write access or transfer request will change the mode into MPU or DMA respectively.

4.2.1 MPU Mode

In MPU mode, an access is made to one of the DMAC internal registers. The registers can be accessed by byte or word. There are 64 bytes defined per channel and each channel has 17 registers. There are four channels available, so the DMAC takes 256 bytes in the memory map. The address lines A1 through A7 and the data strobes determine which register will be selected.
When an interrupt acknowledge cycle is performed the DMA controller puts the contents of the normal or error interrupt vector register of the highest channel requesting an interrupt on the data bus. The DMAC uses a multiplexed address/data bus, demultiplexing is done by U7-U10 with the control signals DDIR (Data DIRection), DBEN (Data Bus ENable) and UAS (Upper Address Strobe).

4.2.2 DMA Mode

When in DMA mode, the DMAC is the current VMEbus master and activates the OWN line; this will enable the buffers of the VMEbus interface. Transfer modes used for DMA are defined as follows:

Dual addressing with auto request:

- transfer byte or word from memory to holding register.
- transfer byte or word from holding register to memory.

Single addressing with request acknowledge handshake:

- transfer byte from SCSI PC to memory (using D0-D7).
- transfer byte from SCSI PC to memory (using D8-D15).
- transfer byte from memory to SCSI PC (using D0-D7).
- transfer byte from memory to SCSI PC (using D8-D15).

Timing and control signals are generated by U15, U16 and U29.

4.3 VMEbus Interface

The main part of the interface to the VMEbus is given in the schematic diagram (Appendix B, sheet 1). The address and data lines are shown with their respective buffers to the VMEbus. The control signals for these buffers are; GVME to enable the data buffer on the VMEbus, FROMVME to control the data direction of the data buffers, and OWNL wich is used to enable the address buffers. These control signals are generated by U29 (sheet 7). The multiplexing of the address and data lines is performed by the DMAC using the signals UAS (Upper Address Strobe), DDIR (Data DIRection) and DBEN (Data Bus ENable). The control lines on the VMEbus are buffered by U14, U26 and U27 (sheet 4). The generation of DTACK (when in MPU mode) is performed by U15 (sheet 7) and buffered by U24 and U30. A shift register U52 is used for the DTACK timing.

4.4 Address Decoding

The address decoder is given in the schematic diagram of sheet 2 and uses the address lines A9-A23 and AM code lines AM0-AM5. The LIACK signal is used to inhibit address decoding when an IACK cycle is performed. Jumper JB2 is used to enable standard or short addressing, and JB1 to select supervisory-only or non-privileged access. The remaining modifier codes select data I/O access.

4.5 DTB Requester

The onboard DTB requester consists of two parts, the requester logic U20 and the chaining logic U21. The bus request logic, when activated by LDBR (Local DMA Bus Request) will assert one of the BRx outputs depending on the installed request level (REQLV0, REQLV1). Now it will wait for a Bus Grant (BGT) from the chaining logic. The chaining logic checks the incoming BGxIN signals from the VMEbus and determines if the incoming signals must be chained to BGxOUT or should be used locally. Latch U19 is used to guarantee a valid BGxIN signal for the chaining logic. The level of bus request is software-selectable and is discussed in chapter 5.5.

4.6 Interrupter

The Interrupter U22 (sheet 3) uses the latched VMEbus signals. This circuit asserts one of the seven VMEbus interrupt signals (IRQ1-IRQ7) when a local DMA interrupt (LIRQDMA) is received. The interrupt level is determined by IRQLV0-IRQLV2 and is software selectable (see chapter 5.4). When an interrupt has been asserted the interrupter waits for an Interrupt acknowledge cycle on the VMEbus. Then the interrupt level will be checked and when a match is found IACKSEL will be asserted, otherwise IACKOUT is asserted. The DMA controller will generate the proper interrupt vector, when necessary. When an interrupt acknowledge cycle for the DMAC is performed, the interrupter releases the VMEbus IRQ line and will only respond to another IRQDMA, after negation of LIRQDMA for at least two clock cycles (125 ns).

4.7 External DMA Devices

The 68450 is a four channel DMAC and only two channels are used by the SCSI interface. The control lines of the two unused channels are connected to the P2 connector. The direction of these signals can be selected by jumpers. The two channels may therefore be used by other devices on other modules via control lines on the P2 connector. Care should be taken here to prevent conflicts with the VMEbus specification; however, relatively simple I/O modules may take advantage of these DMA channels.

4.8 SCSI Interface

The SCSI protocol controller used is the NCR 5385 (or NCR 5386). Sheet 5 of the schematic diagrams shows the complete interface. SCSI bus buffering is performed by U42 through U47. Units U40 and U41 are used for driving one of the eight data lines on the SCSI bus during arbitration. JB4 selects which data line will be driven during arbitration, the so called ID address. U36 encodes the JB4 setting into three ID signals ID0, ID1 and ID2, used by the SCSI PC. U39 (sheet 6) is a 10 MHz clock oscillator for the SCSI PC clock. Connector P3 is a flat cable connector which has the proper SCSI bus pin definitions. Connector P2 is the standard VMEbus I/O connector which has also all SCSI bus signals connected. An optional module CC85 is available to interface a 50-pin SCSI bus flat cable to the P2 connector.

CHAPTER 5

PROGRAMMING CONSIDERATIONS

5.1 Introduction

This section contains all necessary information for system programmers to take full advantage of the features of the CC74 module. Additional information may be found in the respective data sheets of the SCSI protocol controller (NCR 5385/6) and the DMA controller (68450). For system programmers who want to write their own CC74 driver software, a full understanding and experience is required. The source code of a sample driver routine for this module is found in Appendix J.

5.2 Memory Map

The memory map is given in Appendix F. The module takes $200 (=512) byte locations, starting at the selected base address. The first 256 locations are used for the DMA controller and the second block is partially used by the SCSI controller (16 bytes) and the local control register (1 byte). Both the SCSI controller and the control register are 8 bits wide and are accessed at odd memory locations. Both are duplicated an arbitrary number of times in the memory map. The recommended address locations for use by system programmers are given below.

```
        B_DMAC    equ   CC74_BASE+0        base address for DMAC
        CNTREG    equ   CC74_BASE+$100     address of control reg.
        B_SCSI    equ   CC74_BASE+$120     base address for SCSI PC
```

CC74_BASE depends on the address select switches S1-S4. Note that the byte wide registers respond at odd addresses only. The registers within the DMAC and SCSI PC can be accessed by using B_DMAC and B_SCSI as the respective register base addresses and declaring each register as an offset from these addresses (see Appendices G and H)

5.3 Reset

The SYSRES* signal from the VMEbus will reset the DMAC, SCSI PC, DTB Requester and the local Control Register. When the DMAC recognizes Reset, it relinquishes the bus, clears the GCR and resets the DCR, OCR, SCR, CCR, CSR, CPR and CER of all channels. The interrupt vector registers are set to $0F (uninitialized interrupt vector number). When SYSRES* is active, the SCSI Protocol Controller is forced into a reset state. All current operations are terminated and internal storage elements (registers, counters etc.) are cleared. A 'chip reset command' loaded into the SCSI PC performs the same operation as the hardware reset. The DTB Requester falls into the idle state after reset, which means that all output signals are negated (no bus requests active). The local control register will be cleared after reset and all output lines will be low. This disables the interrupt request level, the bus request level will be 0 and the SCSI reset signal will NOT be activated. The SCSI reset signal when activated (by a write instruction to the Control register) will reset the SCSI PC, which will then release all bus signals within a 'bus clear delay'. The Interrupt handler attached to the CC74 module can be informed about the SCSI Reset condition with an interrupt request. The PCL1 line of the DMAC is used for this purpose.

5.4 Interrupts

All interrupts from the CC74 module are generated by the DMAC. These interrupts can be caused by several conditions such as channel operation complete, PCL transition or Bus error. Each channel may generate its own normal interrupt vector or an error interrupt vector. An error interrupt vector is generated when a DMA transfer is terminated by a bus error response or when an address error occurs. In the case of an error, the present values of the Memory Address, Device Address and Base Address registers, the Memory Transfer and Base Transfer counters, and Control, Status and Error registers will be available. The interrupt signal of the SCSI PC is connected to the PCL0 input of the DMAC and may also cause an interrupt on the VMEbus. The interrupt signal from the DMAC is sent to the Interrupter which will assert the proper interrupt request line. The interrupt level is selected by the Control Register outputs IRQLV0, IRQLV1 and IRQLV2 (3 binary encoded level outputs).

IRQLV2	IRQLV1	IRQLV0	INTERRUPT LEVEL
0	0	0	(disabled)
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Bit 3 of the Control Register may be used to disable the DMAC IRQ line. This bit is used by the interrupt routine as follows.
-disable IRQ line (set bit 3).
-process the normal interrupt routine.
-disable IRQ line (clear bit 3).
This procedure is necessary to guarantee the proper working of the Interrupter.

5.5 Bus Request

When the DMA controller needs control of the bus it will generate a DMA Bus Request to the DTB requester. The level on which the DTB requester will generate a Bus Request on the VMEbus depends on the REQLV0, REQLV1 outputs of the control register (2 binary encoded level outputs).

REQLV1	REQLV0	BUS REQUEST LEVEL
0	0	0
0	1	1
1	0	2
1	1	3

The DMAC controls VMEbus utilization and request interval timing. The Burst Transfer Mode, Cycle Steal Mode with or without hold, Burst time and Bandwidth Ratio are also under software control.

5.6 Control Register

The Control Register performs several functions. It is used to select the levels for the DTB Requester and the Interrupter and to enable/disable the DMA IRQ line. It is also used for monitoring the SCSI Reset line (LRSTI) and may be used to activate the SCSI Reset line (LRSTO). Note that when LRSTI is '1' and LRSTO was written as '0', means that another device on the SCSI bus is asserting the reset line. The following table shows the bit assignments of the Control Register.

BIT	READ	WRITE	FUNCTION
0	IRQLV0	IRQLV0	
1	IRQLV1	IRQLV1	Interrupt Request Level
2	IRQLV2	IRQLV2	
3	ENIRQ	ENIRQ	Enable DMA IRQ
4	REQLV0	REQLV0	DTB Request Level
5	REQLV1	REQLV1	
6	Spare	Spare	Not used
7	LRSTI	LRSTO	SCSI Reset

5.7 AM Code Generation

When the DMA Controller is in the DMA mode, it will generate Function Codes on its FC0-FC2 lines. These lines are software programmable for source, destination and base address access (see chapter 5.8.1). A translation of these lines is made to generate the six Address Modifier code lines on the VMEbus. The following table shows the translation and function of the respective lines.

FC2	FC1	FC0	AM5	AM4	AM3	AM2	AM1	AM0	FUNCTION
0	0	0	1	0	1	0	0	1	short n.p. I/O
0	0	1	1	1	1	0	0	1	stand n.p. data
0	1	0	1	1	1	0	1	0	stand n.p. prog
0	1	1	1	1	1	0	1	0	stand n.p. prog
1	0	0	1	0	1	1	0	1	short priv I/O
1	0	1	1	1	1	1	0	1	stand priv data
1	1	0	1	1	1	1	1	0	stand priv prog
1	1	1	1	1	1	1	1	0	stand priv prog

5.8 Programming the CC74 Module

Programming the CC74 module can be divided into two main sections: programming the DMAC, and programming the SCSI Protocol Controller. It is also necessary to program the desired interrupt and bus request level as discussed in chapter 5.4 and 5.5 respectively. In chapter 5.8.1 the programming of the DMAC is discussed. Chapter 5.8.2 discusses the programming of the SCSI PC. A programming example is given in Appendix J.

5.8.1 Programming The DMA Controller

The 68450 DMAC has internal control registers and performs the required operation by means of control words written in these registers by the MPU. A normal programming sequence can be divided into three phases.

The Initiation phase: MPU sets up control registers, transfer address and transfer counts.

The Transfer Phase: DMAC receives requests and transfers data. The DMAC writes the transfer status into the error register and the internal status register after the completion of the transfer.

The Termination Phase: The MPU checks the post-transfer status.

The internal registers are shown in Appendix G and a short description is given below.

The Device Control Register designates an external I/O device. It will set the external request generation method, the device type, the device port size, and PCL line operation.

The Operation Control Register designates the transfer operation. It designates the data transfer direction, the operand size, the chain operation types, and the request generation method.

The Sequence Control Register designates the increment/decrement sequence of both memory and device (source and destination) addresses.

The Channel Control Register designates the channel operation. It designates operation start, the continuous-operation setting, HALT, abort, and the interrupt enable/disable.

The Channel Status Register contains the channel status. It shows channel operation completion, block transfer completion, normal termination, the error status, the channel active state, and PCL signal line information.

The Channel Error Register indicates what error types have occurred.

The Channel Priority Register determines the priority of the channel.

The Memory Transfer Counter is a 16-bit register to hold transfer counts. The block size (transfer count) is written when one data block is transferred. When multiple blocks are transferred in Continuous Mode or Chaining Modes, the next block size is automatically loaded in the MTC after completion of the previous block transfer.

The Base Transfer Counter is used in Continuous Mode and Array Chaining Mode. In Continuous Mode the first block size is stored in the MTC and the second block size in the BTC. The content of the BTC is copied into the MTC after completion of the first block transfer. When more than two blocks are transferred in this mode, the BTC and BAR (described later) are rewritten and the CNT bit in the CSR is set again during the second (or third) block transfer. In Array Chaining Mode, the BTC holds the number of blocks being transferred.

The Memory Address Register contains the memory address being output for each transfer cycle. In block transfer, the beginning address of the block is written in the MAR as an initial value. And the content of the MAR varies according to the contents of the OCR and the SIZE bits in the SCR after one operand transfer. In Continuous Mode and Chain Modes, the MAR is rewritten according to the BAR or the array information in memory when a block transfer completes.

The Device Address Register is used to address an I/O device (or to address memory, in memory-to-memory transfer). The DAR is used only in Dual Addressing Mode, and changes its content according to the SCR and according to the SIZE bits in the OCR.

The Base Address Register is used in Continuous Mode and Chain Modes. In Continuous Mode, the start address of the second block is written in the BAR. This BAR is used in the same way as the BTC. In Chain Modes, it keeps the address where the information of the next block is contained.

The Memory Function Code, Device Function Code, and Base Function Code Registers are used together with the MAR, DAR, and BAR respectively. The MFC, DFC, and BFC are used with the same purpose as the FC outputs from the MPU. This makes it possible to transfer data between supervisor program area and user data area, for example. A translation is made to generate VMEbus AM codes from the Function Codes, this is discussed in section 5.7.

The Normal and Error Interrupt Vectors keep the vector numbers outputted in the vector number fetch cycle (Interrupt Acknowledge Cycle). When no error has occurred (ERR bit of CSR is not set), the DMAC outputs the NIV contents. When an error has occured (ERR=1), the DMAC outputs the EIV contents.

The General Control Register is common to all four channels and determines the DMAC's bus use ratio and sample interval in limited Rate Auto-Request Mode. In Maximum Rate Auto-Request Mode, the DMAC takes the bus mastership and transfers all operands until they are exhausted. In this mode, when the higher priority channels request transfer, the channel with the Maxixmum Rate Auto-Request stops its transfer temporarily and the higher priority channel is serviced. The Maximum Rate channel resumes its transfer after the priority channel has been serviced.

After the DMAC is properly initialized, a transfer is started by setting the STR bit in the Channel Control Register. When the DMAC completes a transfer operation, the COC (Channel Operation Complete) bit in the CSR is set. If an error occurs during the transfer, the ERR bit is also set. If the INT (Interrupt Enable) bit has been set, the DMAC will issue an interrupt request when the COC bit is set. If Interrupts are disabled, the MPU should poll the COC bit. The transfer termination routine should check for errors. Error routines should be programmed case by case according to their applications. For bus error and address error, the CER (Channel Error Register) can show which address register caused the error and the address where the error occurred is kept in the address register. The CER also shows which of the transfer counters between the MTC and BTC caused the error.

5.8.2 Programming The SCSI Protocol Controller

The SCSI Protocol Controller has a set of 13 internal 8 bit registers, which are used to read or write data, status and control information. A summary of the SCSI registers is given in appendix H. Note that some registers are read only. A normal programming sequence where the CC74 module act as an Initiator on the SCSI bus is as follows.

1. Arbitration phase; Initiator arbitrates for the bus.
2. Selection phase; Initiator selects a Target.
3. Prepare SCSI chip and DMAC for data transfer.
4. Message-In phase (optional).
5. Command phase; give command to Target.
6. Data phase; transfer data (optional).
7. Status phase; get status from Target.
8. Receive command complete message.
9. Wait for disconnection.
10. Check for errors.

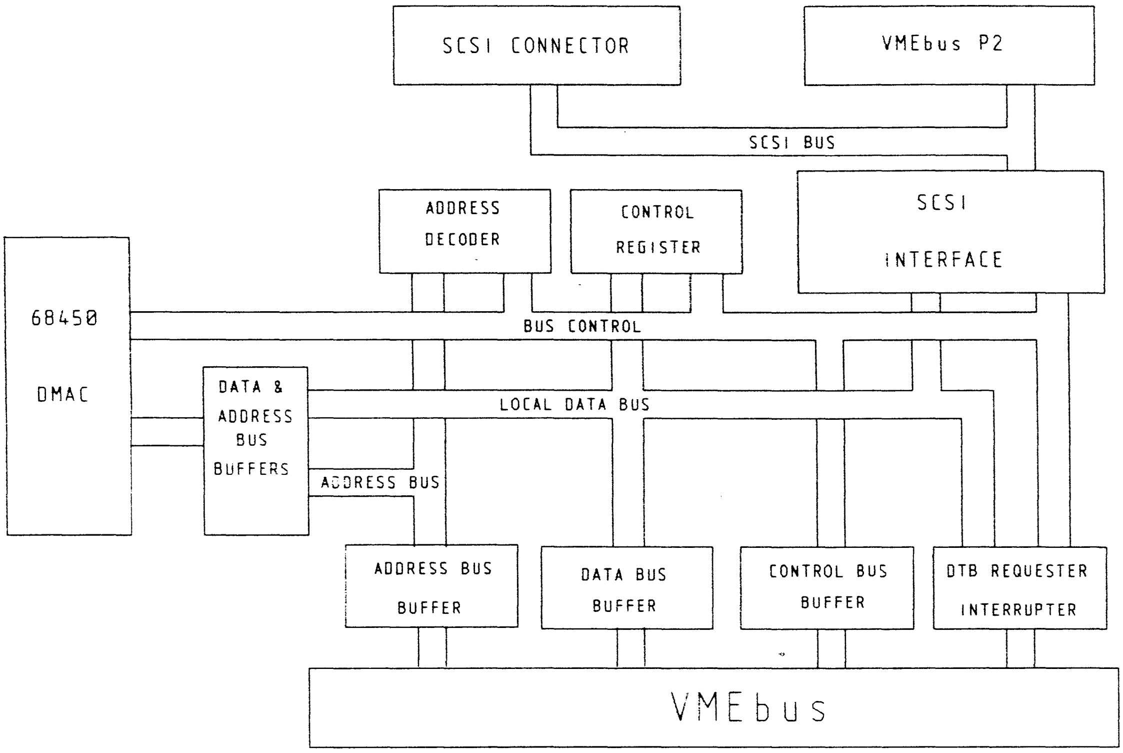
When the CC74 module acts as a Target, the programming sequence can be as given below.

1. Wait for selection.
2. Initiate a Message-In phase (optional).
3. Command phase; get command from Initiator.
4. perform command.
5. Data phase; transfer data (optional).
6. Status phase; send status to Initiator.
7. Message phase; send message to Initiator.
8. Disconnect initiator.
9. Idle mode.

Before any operations are performed, an Internal Diagnostic Command should be given and status information checked for succesful completion. After power-on Reset, the Internal Diagnostic Command is automatically executed.

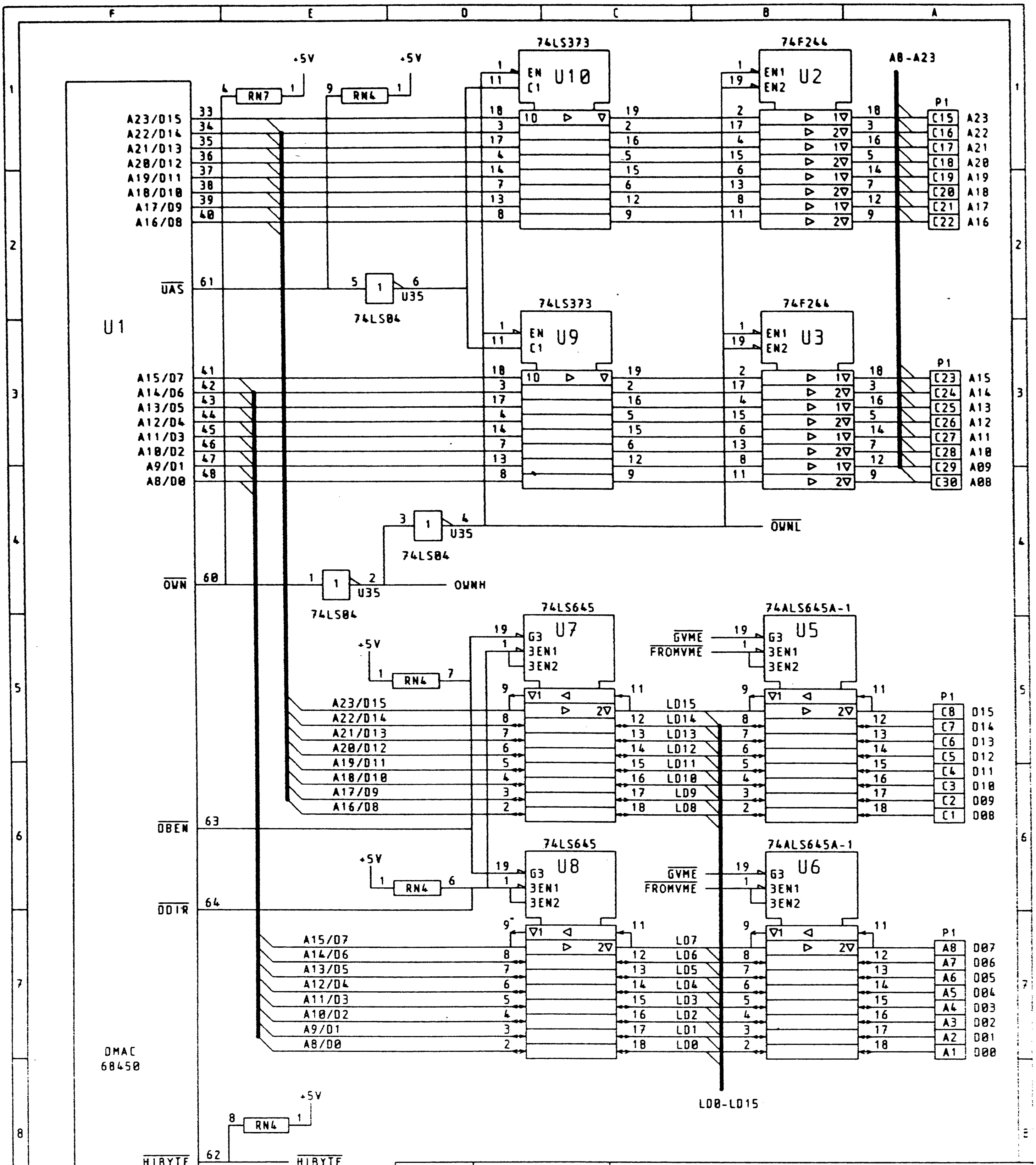
APPENDIX A

BLOCK DIAGRAM



APPENDIX B

SCHEMATIC DIAGRAM



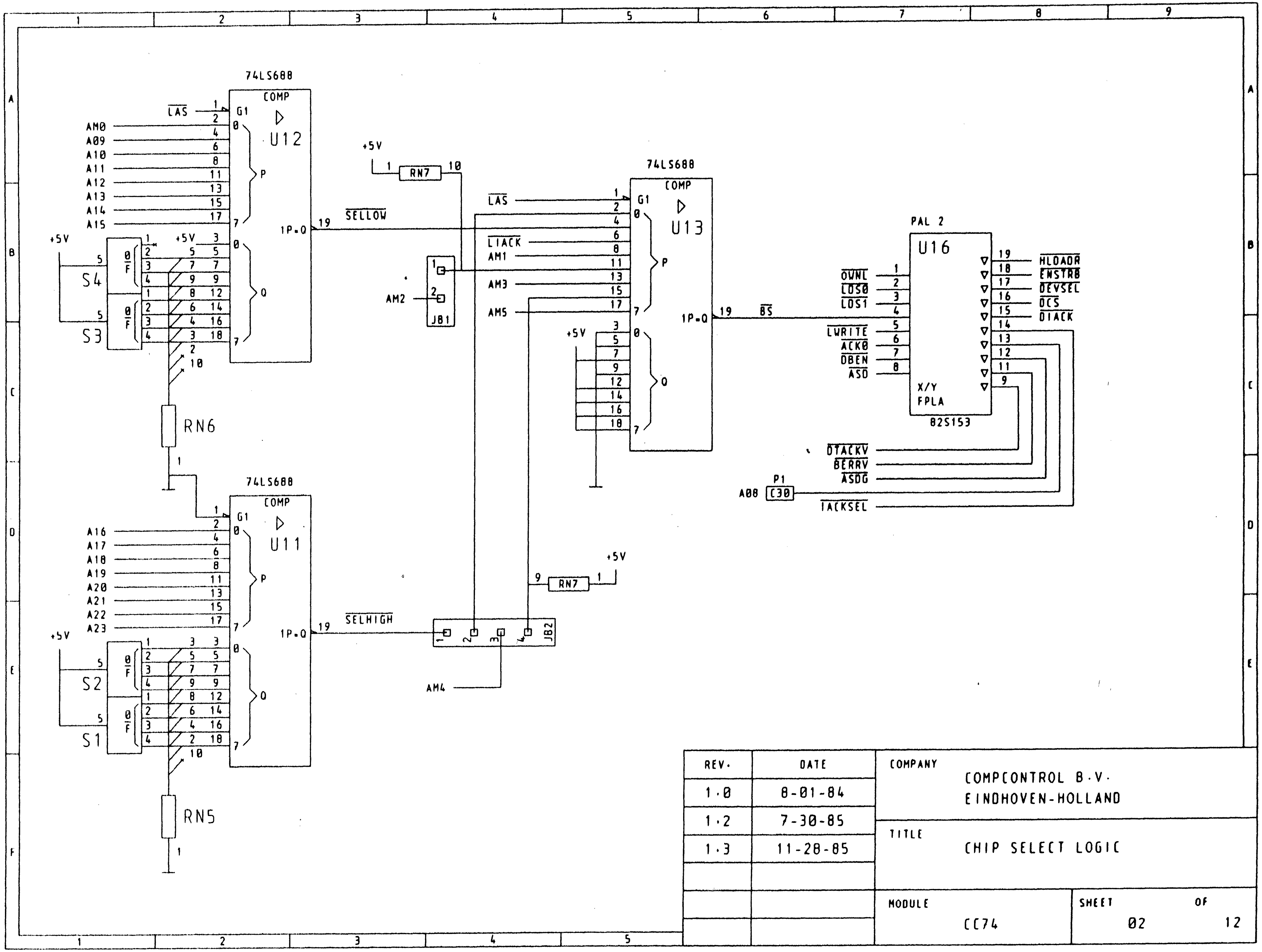
REV.	DATE
1.0	1-05-84
1.2	7-30-85
1.3	11-28-85

COMPANY
COMPCONTROL B.V.
EINDHOVEN-HOLLAND

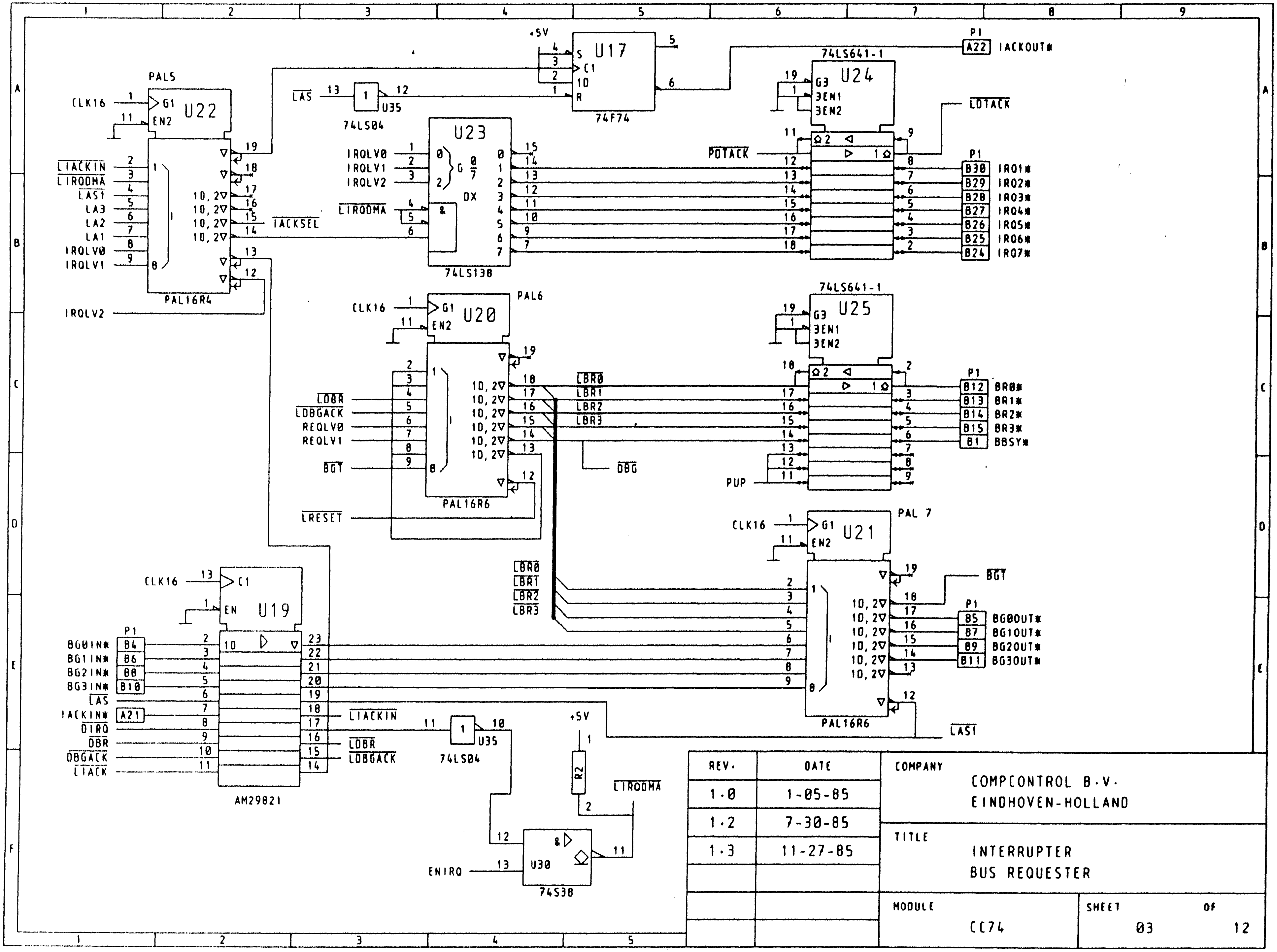
TITLE
DMAC-VMEbus
INTERFACE

MODULE
CC74

SHEET 01 OF 12



REV.	DATE	COMPANY
1.0	8-01-84	COMPCONTROL B.V. EINDHOVEN-HOLLAND
1.2	7-30-85	TITLE CHIP SELECT LOGIC
1.3	11-28-85	
		MODULE CC74
		SHEET 02 OF 12

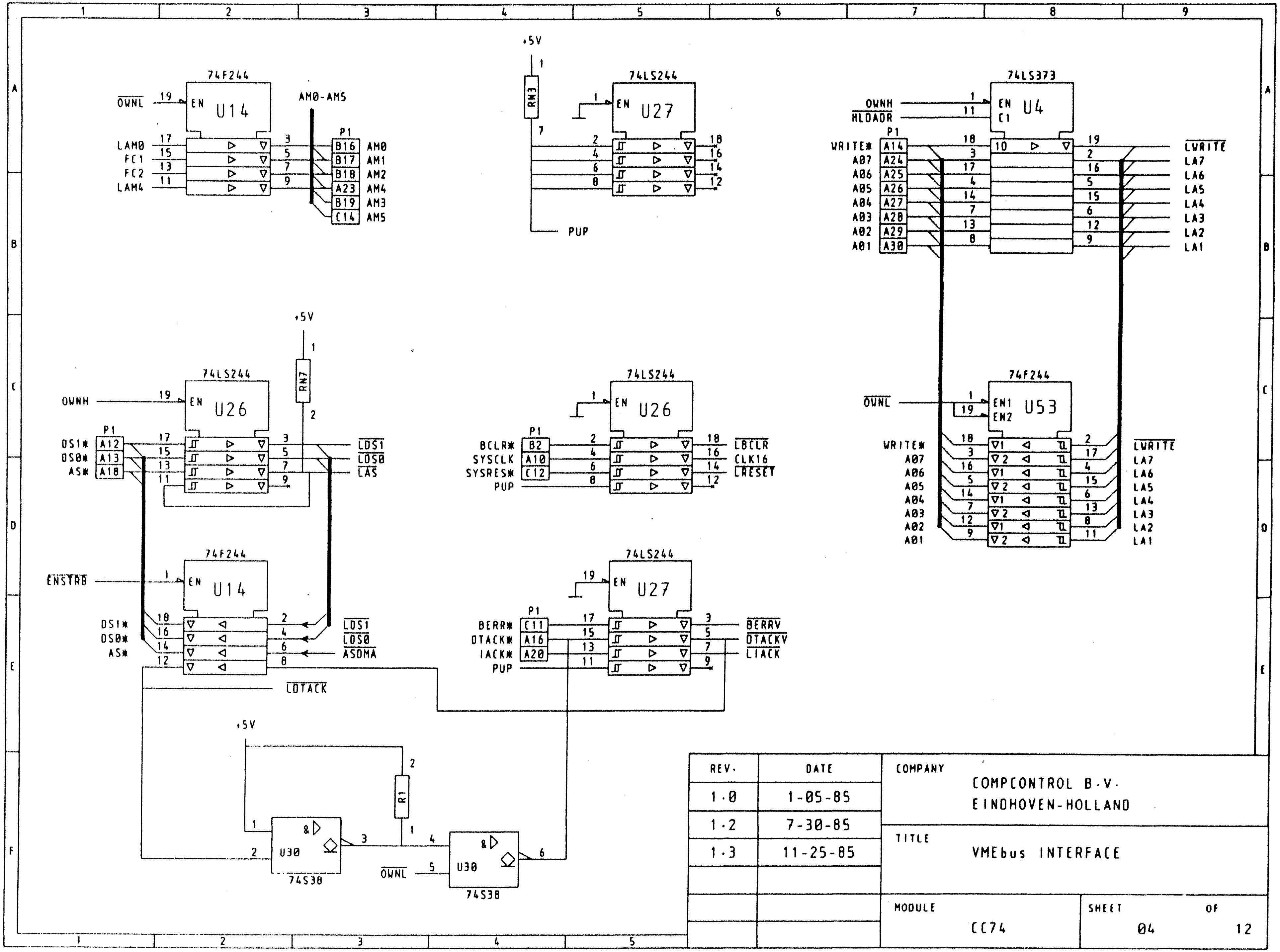


REV.	DATE
1.0	1-05-85
1.2	7-30-85
1.3	11-27-85

COMPANY
COMPCONTROL B.V.
EINDHOVEN-HOLLAND

TITLE
INTERRUPTER
BUS REQUESTER

MODULE	SHEET	OF
CC74	03	12



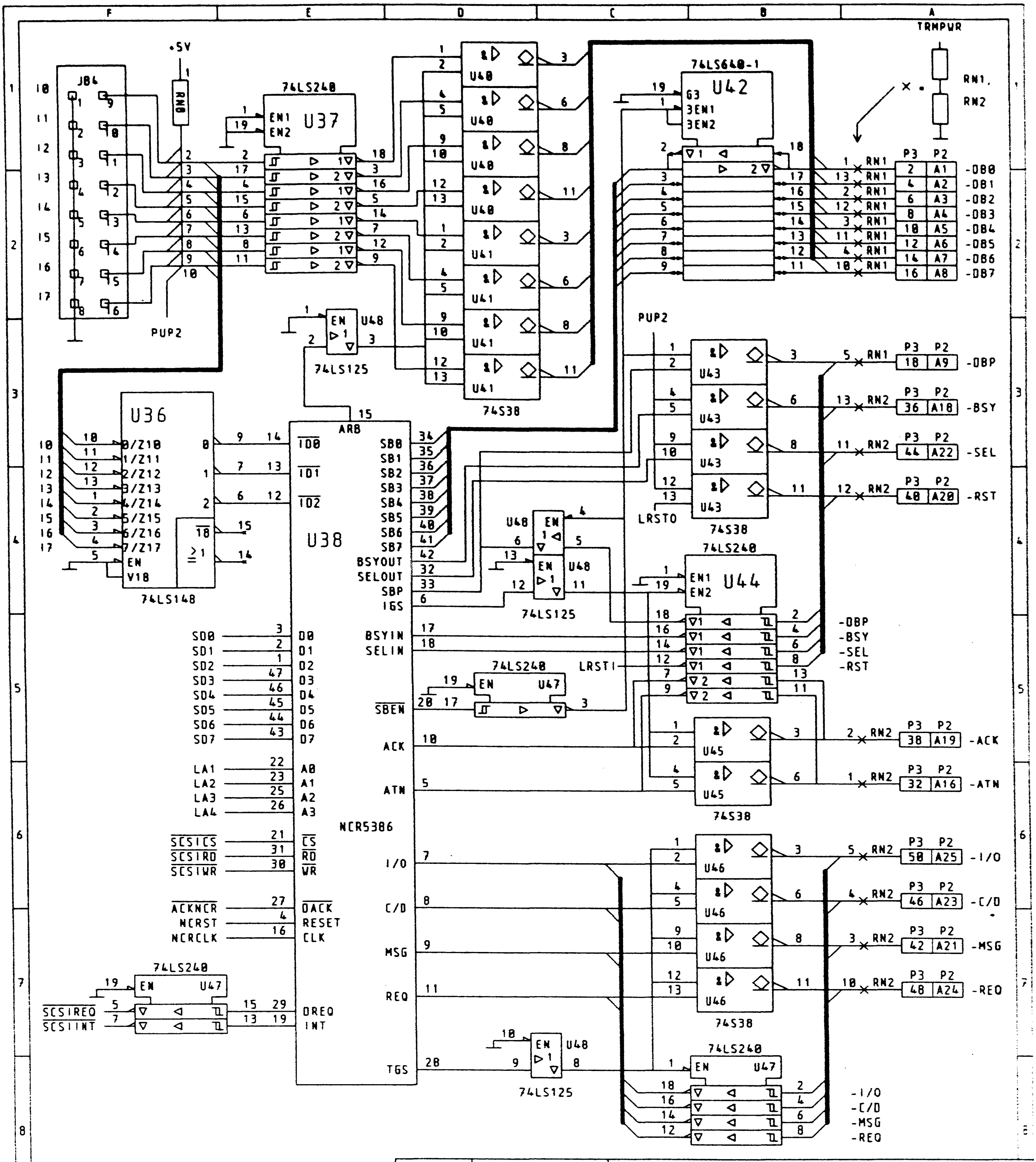
REV.	DATE
1.0	1-05-85
1.2	7-30-85
1.3	11-25-85

COMPANY
COMPCONTROL B.V.
EINDHOVEN-HOLLAND

TITLE
VMEbus INTERFACE

MODULE
CC74

SHEET 04 OF 12



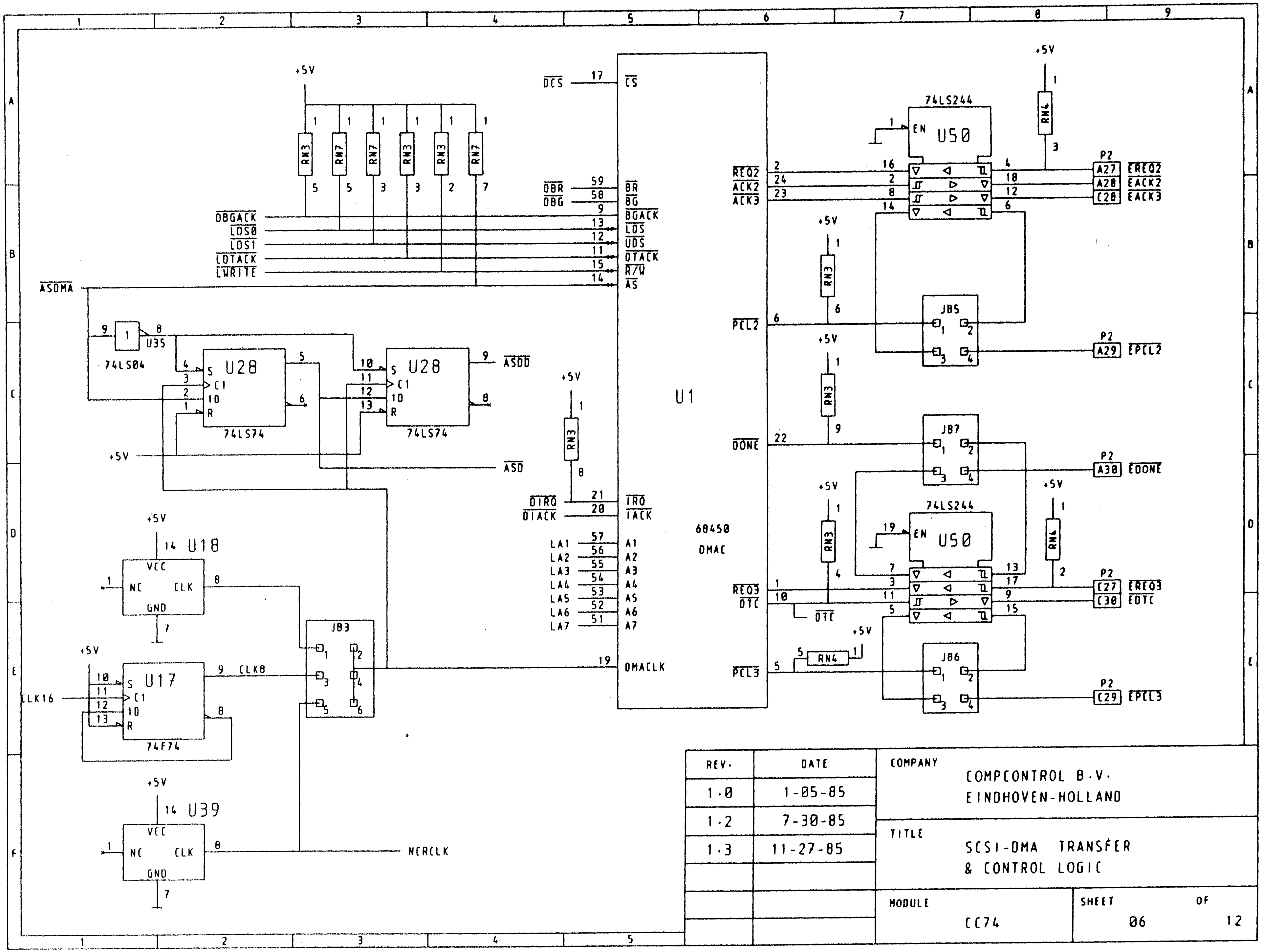
REV.	DATE
1.0	1-05-85
1.2	7-30-85
1.3	11-27-85

COMPANY
COMPCONTROL B.V.
EINDHOVEN-HOLLAND

TITLE
SCSI INTERFACE

MODULE
CC74

SHEET 05 OF 12



REV.	DATE
1.0	1-05-85
1.2	7-30-85
1.3	11-27-85

COMPANY

COMPCONTROL B.V.
EINDHOVEN-HOLLAND

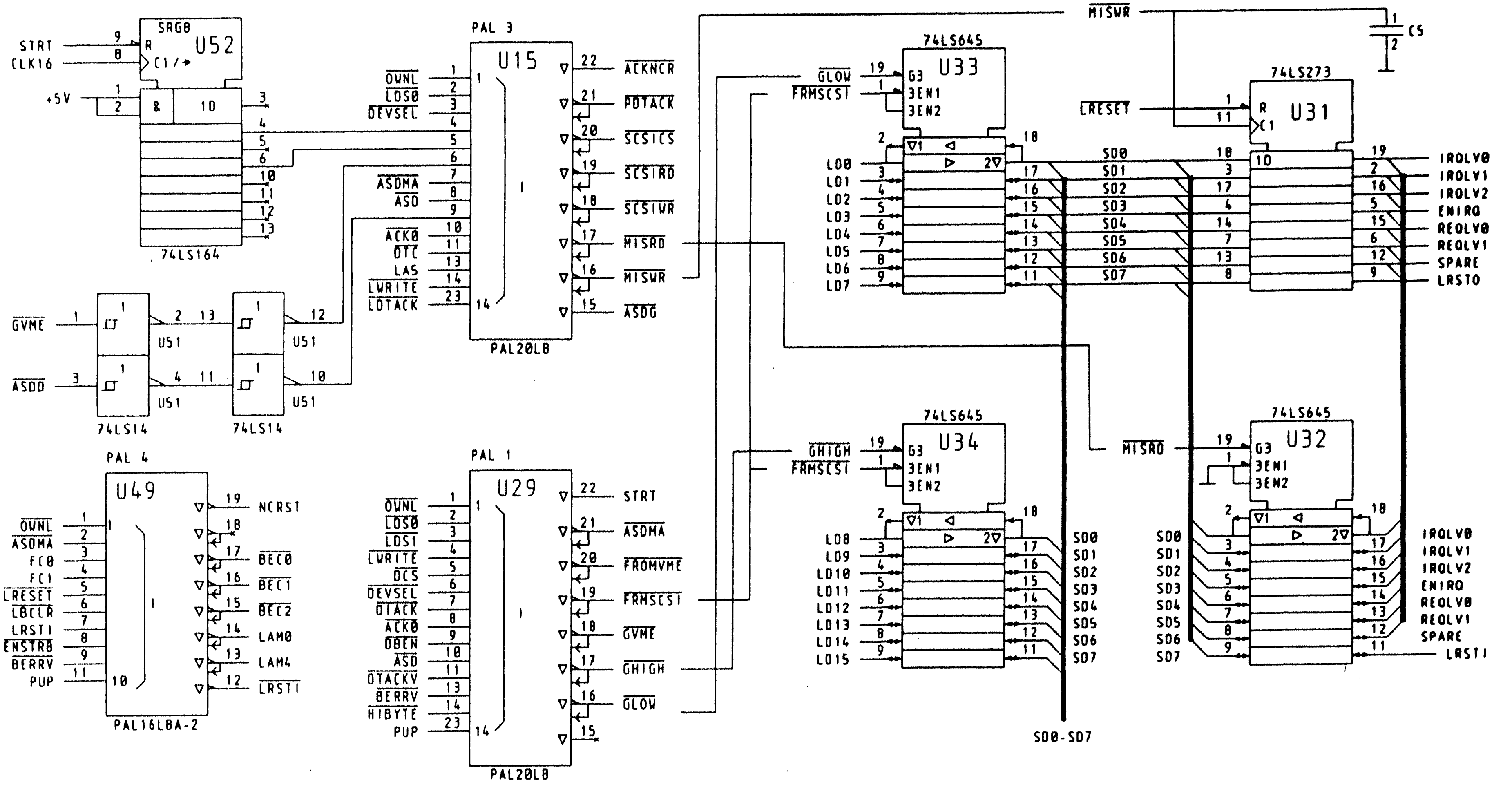
TITLE

SCSI-DMA TRANSFER
& CONTROL LOGIC

MODULE

CC74

SHEET 06 OF 12



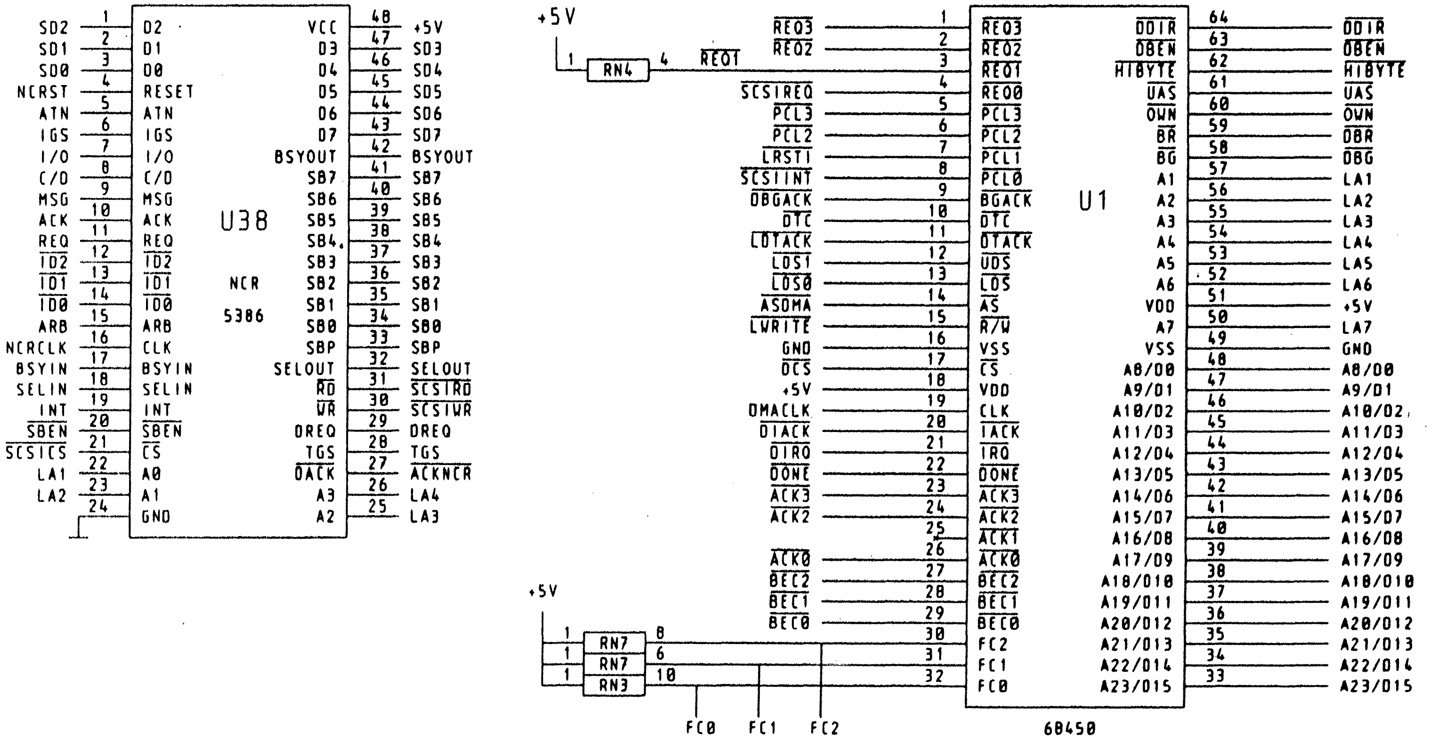
REV.	DATE
1.0	1-05-85
1.2	7-30-85
1.3	11-27-85

COMPANY
COMPCONTROL B.V.
EINDHOVEN-HOLLAND

TITLE
LOCAL DATA PATH.

MODULE
CC74

SHEET 07 OF 12



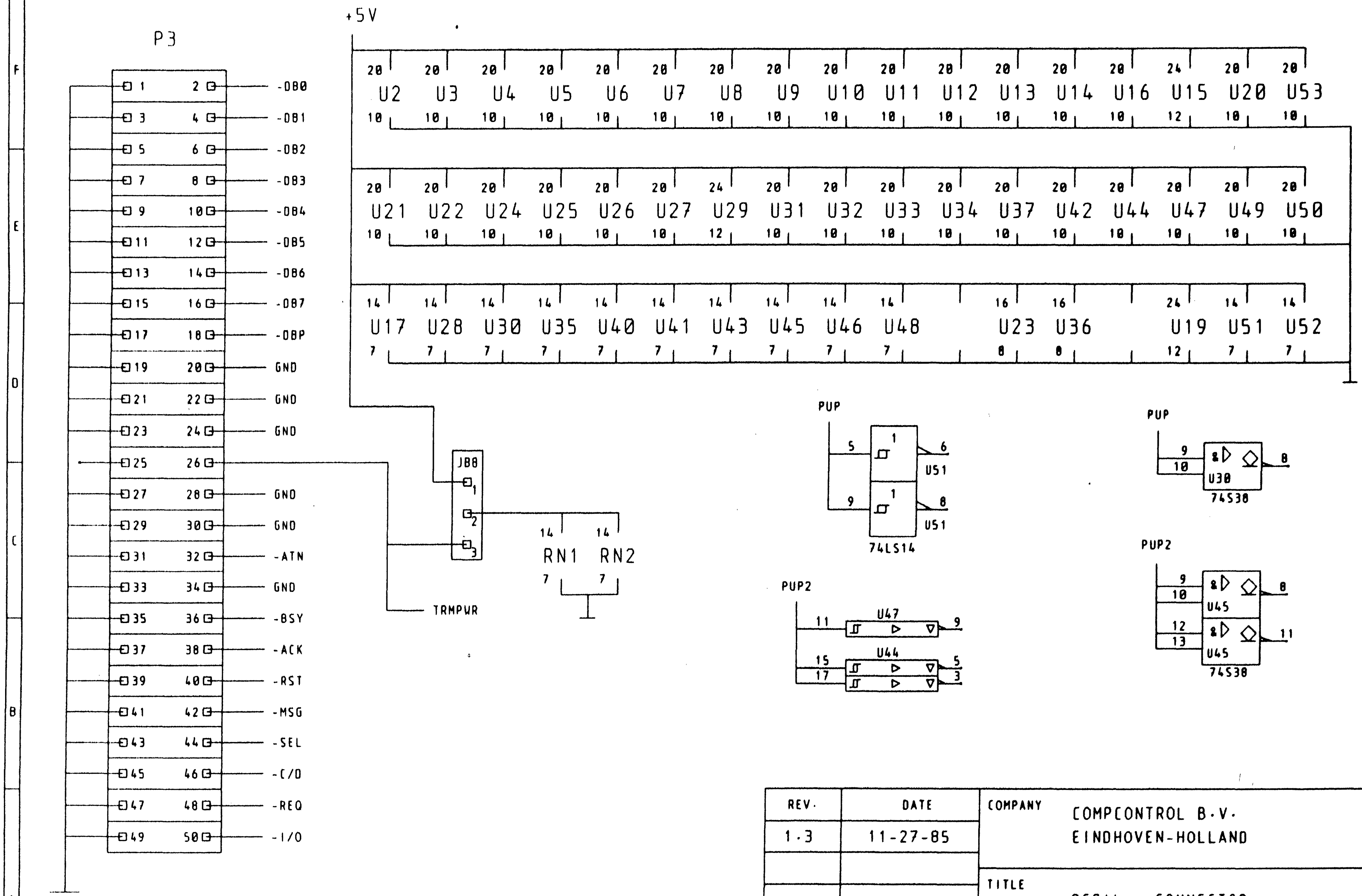
REV.	DATE
1.0	1-05-85
1.2	7-30-85
1.3	11-28-85

COMPANY
COMPCONTROL B.V.
EINDHOVEN-HOLLAND

TITLE
68450 DMAC
NCR 5386

MODULE
CC74

SHEET 08 OF 12



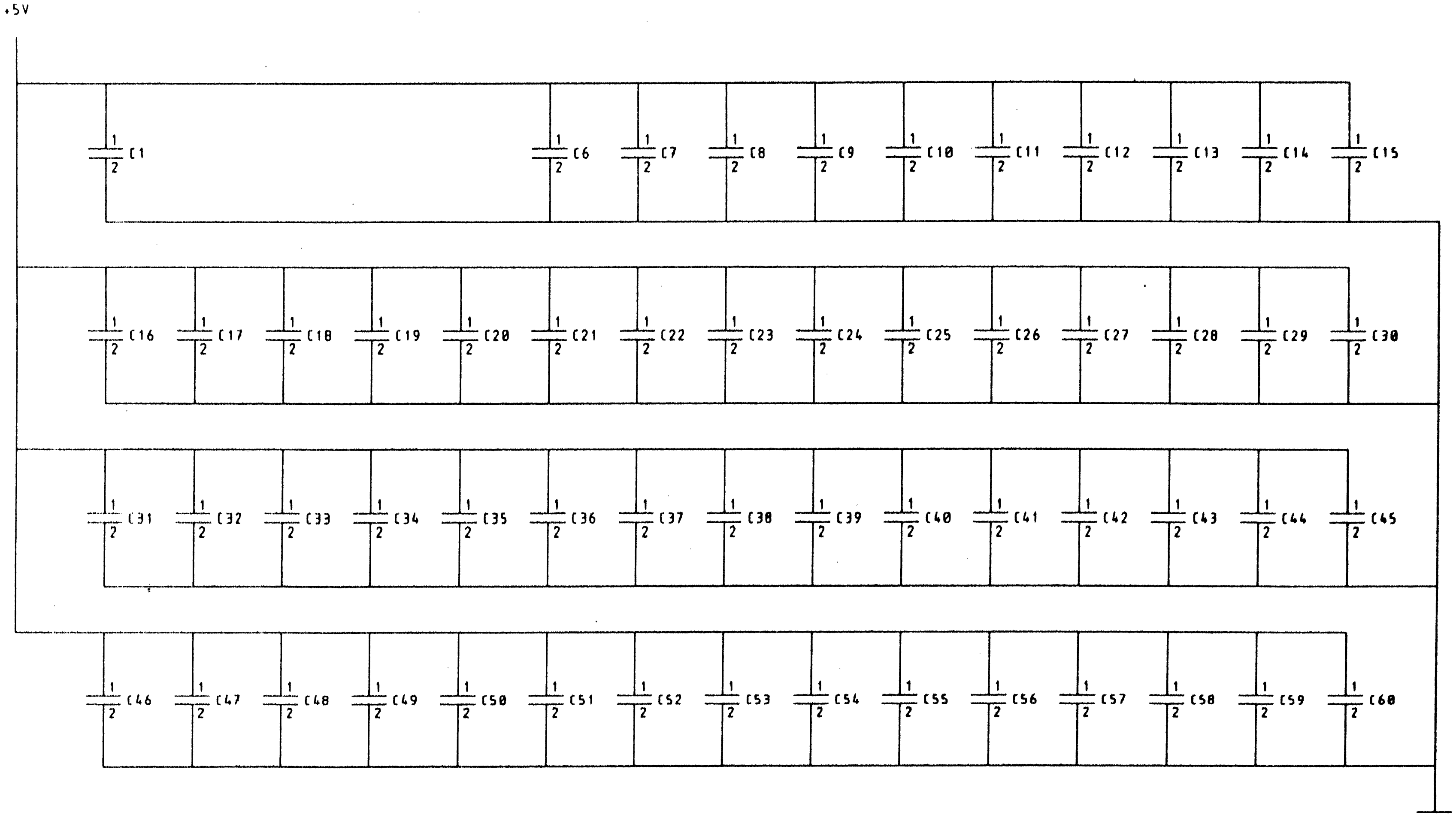
REV.	DATE
1.3	11-27-85

COMPANY
COMPCONTROL B.V.
EINDHOVEN-HOLLAND

TITLE
SCSIbus CONNECTOR
POWER SUPPLY's

MODULE
CC74

SHEET 09 OF 12



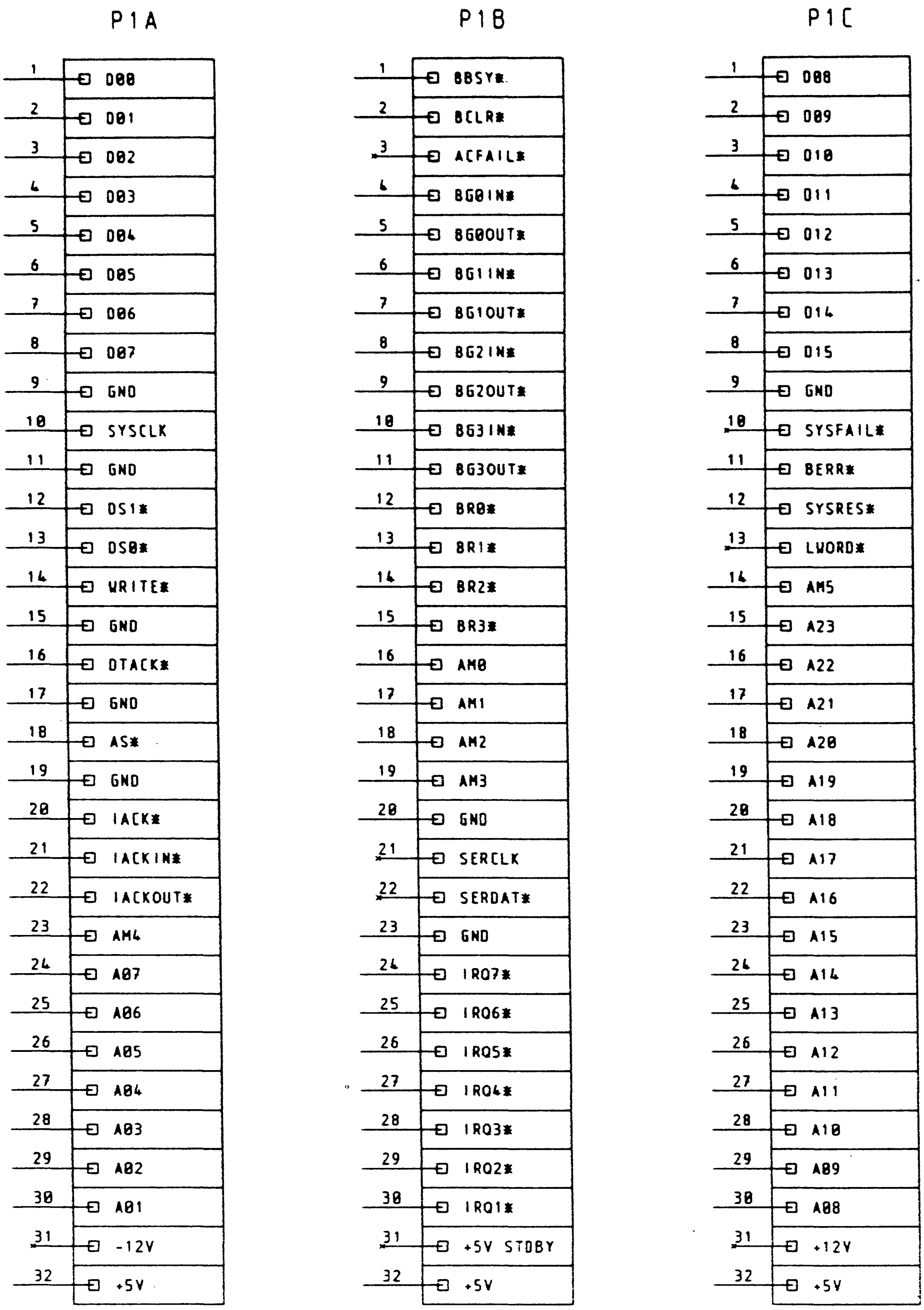
REV.	DATE
1.0	1-05-85
1.2	7-30-85
1.3	11-29-85

COMPANY
COMPCONTROL B.V.
EINDHOVEN-HOLLAND

TITLE
DECOUPLING

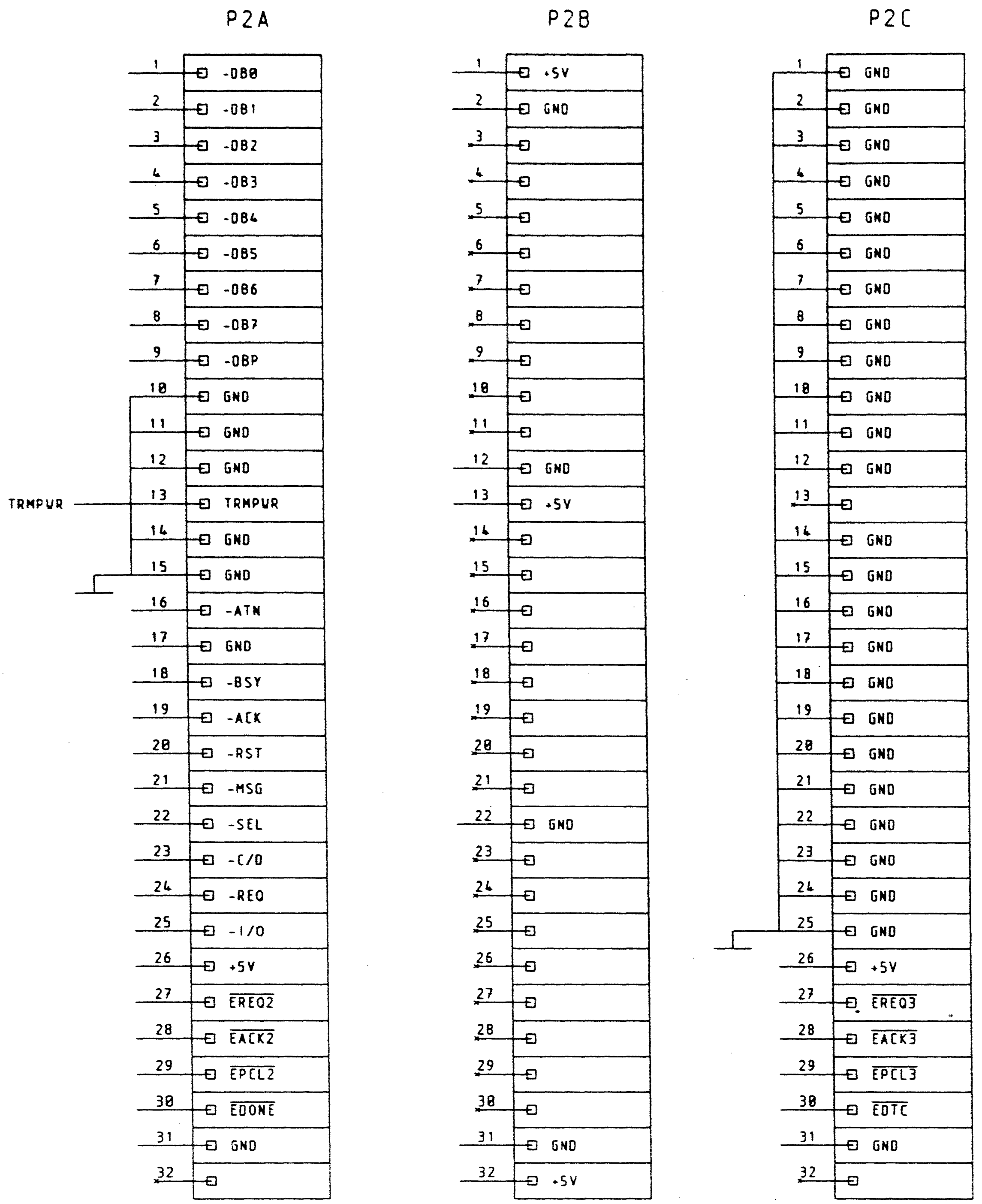
MODULE
CC74

SHEET 10 OF 12



Pin	P1A	P1B	P1C
1	D00	BBSY*	D08
2	D01	BCLR*	D09
3	D02	ACFAIL*	D10
4	D03	BG0IN*	D11
5	D04	BG0OUT*	D12
6	D05	BG1IN*	D13
7	D06	BG1OUT*	D14
8	D07	BG2IN*	D15
9	GND	BG2OUT*	GND
10	SYSCLK	BG3IN*	SYSFAIL*
11	GND	BG3OUT*	BERR*
12	DS1*	BR0*	SYSRES*
13	DS0*	BR1*	LWORD*
14	WRITE*	BR2*	AM5
15	GND	BR3*	A23
16	DTACK*	AM0	A22
17	GND	AM1	A21
18	AS*	AM2	A20
19	GND	AM3	A19
20	IACK*	GND	A18
21	IACKIN*	SERCLK	A17
22	IACKOUT*	SERDAT*	A16
23	AM4	GND	A15
24	A07	IRQ7*	A14
25	A06	IRQ6*	A13
26	A05	IRQ5*	A12
27	A04	IRQ4*	A11
28	A03	IRQ3*	A10
29	A02	IRQ2*	A09
30	A01	IRQ1*	A08
31	-12V	+5V STDBY	+12V
32	+5V	+5V	+5V

REV.	DATE	COMPANY	
1.0	2-8-85	COMPCONTROL B.V. EINDHOVEN-HOLLAND	
		TITLE VMEbus CONNECTOR P1	
		MODULE CC74	SHEET 11 OF 12



Pin	P2A	P2B	P2C
1	-DB0	+5V	GND
2	-DB1	GND	GND
3	-DB2		GND
4	-DB3		GND
5	-DB4		GND
6	-DB5		GND
7	-DB6		GND
8	-DB7		GND
9	-DBP		GND
10	GND		GND
11	GND		GND
12	GND	GND	GND
13	TRMPWR	+5V	
14	GND		GND
15	GND		GND
16	-ATN		GND
17	GND		GND
18	-BSY		GND
19	-ACK		GND
20	-RST		GND
21	-MSG		GND
22	-SEL	GND	GND
23	-C/D		GND
24	-REQ		GND
25	-I/O		GND
26	+5V		+5V
27	$\overline{EREQ2}$		$\overline{EREQ3}$
28	$\overline{EACK2}$		$\overline{EACK3}$
29	$\overline{EPCL2}$		$\overline{EPCL3}$
30	$\overline{EDONE}$		$\overline{EDTC}$
31	GND	GND	GND
32		+5V	

REV.	DATE
1.0	2-8-85
1.3	11-27-85

COMPANY
COMPCONTROL B.V.
EINDHOVEN-HOLLAND

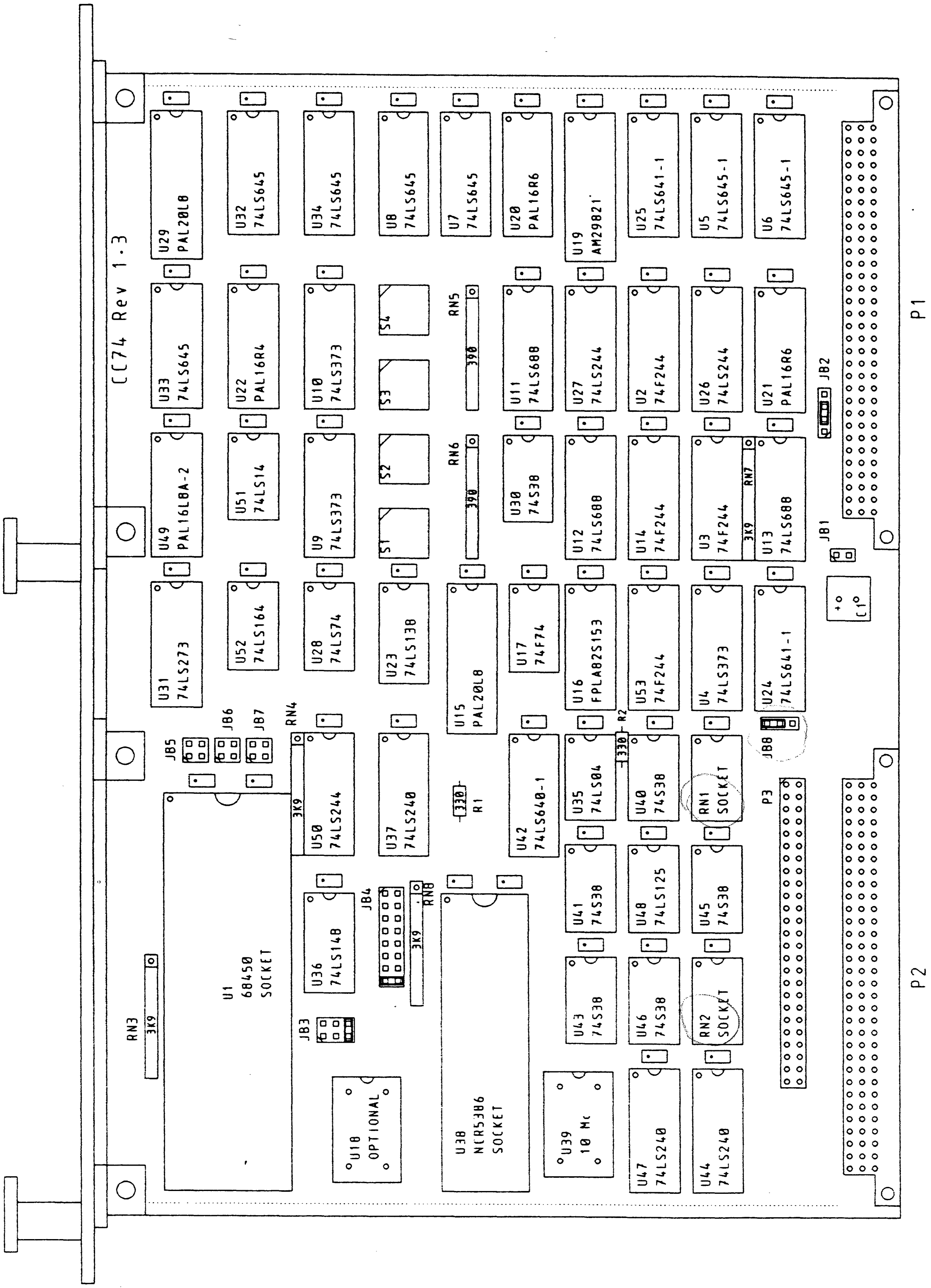
TITLE
VMEbus CONNECTOR P2

MODULE CC74

SHEET 12 OF 12

APPENDIX C

COMPONENT LAYOUT



APPENDIX D

LIST OF COMPONENTS

INTEGRATED CIRCUITS

U1	HD68450-8
U2	74S244, AS244, F244
U3	74S244, AS244, F244
U4	74LS373
U5	74ALS645A-1
U6	74ALS645A-1
U7	74ALS645
U8	74ALS645
U9	74LS373
U10	74LS373
U11	74LS688, AMD2521
U12	74LS688, AMD2521
U13	74LS688, AMD2521
U14	74S244, AS244, F244
U15	PAL20L8A-2
U16	FPLA82S153
U17	74F74
U18	XTAL Oscillator (optional)
U19	AM29821, 74AS821
U20	PAL16R6A-2
U21	PAL16R6A-2
U22	PAL16R4A-2
U23	74LS138
U24	74LS641-1
U25	74LS641-1
U26	74LS244, ALS244, AS244
U27	74LS244, ALS244, AS244
U28	74LS74
U29	PAL20L8A-2
U30	74S38
U31	74LS273
U32	74LS645
U33	74LS645
U34	74LS645
U35	74LS04
U36	74LS148
U37	74LS240
U38	NCR5386
U39	XTAL Oscillator (10 Mc)
U40	74S38
U41	74S38
U42	74LS640-1
U43	74S38
U44	74LS240
U45	74S38
U46	74S38
U47	74LS240

U48	74LS125
U49	PAL16L8A-2
U50	74LS244, ALS244, AS244
U51	74LS14
U52	74LS164
U53	74F244, AS244, S244

RESISTOR NETWORKS

RN1	220/330 Ohm
RN2	220/330 Ohm
RN3	3.9 K 10 Pin SIP
RN4	3.9 K 10 Pin SIP
RN5	390 10 Pin SIP
RN6	390 10 Pin SIP
RN7	3.9 K 10 Pin SIP
RN8	3.9 K 10 Pin SIP

MISCELLANEOUS

R1	Resistor 330 Ohm
R2	Resistor 330 Ohm
C1	Elco 47uF 16V
C5	Capacitor 220 pF
O2-O56	Decoupling Capacitor 0.1 uF
S1	Binary Coded Hex Switch
S2	Binary Coded Hex Switch
S3	Binary Coded Hex Switch
S4	Binary Coded Hex Switch
P1	96 Pin DIN41612 Connector
P2	96 Pin DIN41612 Connector
P3	50 Pin Jumper Block (2 X 25)
JB1	2 Pin Jumper Block (1 X 2)
JB2	4 Pin Jumper Block (1 X 4)
JB3	3 Pin Jumper Block (1 X 3)
JB4	16 Pin Jumper Block (2 X 8)
JB5	4 Pin Jumper Block (2 X 2)
JB6	4 Pin Jumper Block (2 X 2)
JB7	4 Pin Jumper Block (2 X 2)

NOTE: Some parts may have been replaced by their equivalent types.

APPENDIX E

CONNECTOR PIN ASSIGNMENTS

Pin Assignments P1 CC74

PIN NUMBER	ROW A SIGNAL MNEMONIC	ROW B SIGNAL MNEMONIC	ROW C SIGNAL MNEMONIC
1	D00	BBSY*	D08
2	D01	BCLR*	D09
3	D02		D10
4	D03	BG0IN*	D11
5	D04	BG0OUT*	D12
6	D05	BG1IN*	D13
7	D06	BG1OUT*	D14
8	D07	BG2IN*	D15
9	GND	BG2OUT*	GND
10	SYSCLK	BG3IN*	
11	GND	BG3OUT*	BERR*
12	DS1*	BR0*	SYSRESET*
13	DS0*	BR1*	
14	WRITE*	BR2*	AM5
15	GND	BR3*	A23
16	DTACK*	AM0	A22
17	GND	AM1	A21
18	AS*	AM2	A20
19	GND	AM3	A19
20	IACK*	GND	A18
21	IACKIN*		A17
22	IACKOUT*		A16
23	AM4	GND	A15
24	A07	IRQ7*	A14
25	A06	IRQ6*	A13
26	A05	IRQ5*	A12
27	A04	IRQ4*	A11
28	A03	IRQ3*	A10
29	A02	IRQ2*	A09
30	A01	IRQ1*	A08
31			
32	+5 Volts	+5 Volts	+5 Volts

Pin Assignments P2 CC74

PIN NUMBER	ROW A SIGNAL MNEMONIC	ROW B SIGNAL MNEMONIC	ROW C SIGNAL MNEMONIC
1	-DB0	+5 Volts	GND
2	-DB1	GND	GND
3	-DB2		GND
4	-DB3		GND
5	-DB4		GND
6	-DB5		GND
7	-DB6		GND
8	-DB7		GND
9	-DBP		GND
10	GND		GND
11	GND		GND
12	GND	GND	GND
13	TRMPWR	+5 Volts	
14	GND		GND
15	GND		GND
16	-ATN		GND
17	GND		GND
18	-BSY		GND
19	-ACK		GND
20	-RST		GND
21	-MSG		GND
22	-SEL	GND	GND
23	-C/D		GND
24	-REQ		GND
25	-I/O		GND
26	+5 Volts		+5 volts
27	REQ2		REQ3
28	ACK2		ACK3
29	PCL2		PCL3
30	DONE		DTCP2
31	GND	GND	GND
32		+5 Volts	

Pin Assignments P3   CC74

PIN NR.	Description
2	-DB0
4	-DB1
6	-DB2
8	-DB3
10	-DB4
12	-DB5
14	-DB6
16	-DB7
18	-DBP
20	GROUND
22	GROUND
24	GROUND
26	TRMPWR
28	GROUND
30	GROUND
32	-ATN
34	GROUND
36	-BSY
38	-ACK
40	-RST
42	-MSG
44	-SEL
46	-C/D
48	-REQ
50	-I/O

Note : All odd pins are connected to ground, except pin 25.
Pin 25 is not connected.

APPENDIX F

MEMORY MAP

All addresses are offset from the hardware installed base adrress of the CC74 module.

$000 $0FF	DMA CONTROLLER REGISTERS	
$100 $11F	CONTROL REGISTER	one 8 bit register, 16 times duplicated at odd addresses.
$120 $13F	SCSI PC REGISTERS	sixteen 8 bit registers at odd addresses.
$140 $15F	CONTROL REGISTER	
$160 $17F	SCSI PC REGISTERS	
$180 $19F	CONTROL REGISTER	
$1A0 $1BF	SCSI PC REGISTER	
$1C0 $1DF	CONTROL REGSITER	
$1E0 $1FF	SCSI PC REGISTER	

APPENDIX G

DMA CONTROLLER REGISTERS

$00	CHANNEL 0		$01
$3E			$3F
$40	CHANNEL 1		$41
$7E			$7F
$80	CHANNEL 2		$81
$BE			$BF
$C0	CHANNEL 3		$C1
$FE		GCR	$FF

Note: The General Control Register only resides at address $FF

The following table shows the register arrangement of channel 0. Each register can be accessed by byte or word. However when STR bit in CCR is set, only byte access is possible.

	high	low	
$00	CSR 0	CER 0	$01
$02			$03
$04	DCR 0	OCR 0	$05
$06	SCR 0	CCR 0	$07
$08			$09
$0A	MTC 0		$0B

Table continued on next page

DMAC Register Table continued

$0C	MAR 0 (H)		$0D
$0E	MAR 0 (L)		$0F
$10			$11
$12			$13
$14	DAR 0 (H)		$15
$16	DAR 0 (L)		$17
$18			$19
$1A	BTC 0		$1B
$1C	BAR 0 (H)		$1D
$1E	BAR 0 (L)		$1F
$20			$21
$22			$23
$24		NIV 0	$25
$26		EIV 0	$27
$28		MFC 0	$29
$2A			$2B
$2C		CPR 0	$2D
$2E			$2F
$30		DFC 0	$31
$32			$33
$34			$35
$36			$37
$38		BFC 0	$39
$3A			$3B
$3C			$3D
$3E			$3F

DMA CONTROLLER REGISTER DEFINITIONS

CSR	Channel Status Register
CER	Channel Error Register
DCR	Device Control Register
OCR	Operation Control Register
SCR	Sequence Control Register
CCR	Channel Control Register
NIV	Normal Interrupt Vector
EIV	Error Interrupt vector
CPR	Channel Priority Register
MFC	Memory Function Codes
DFC	Device Function Codes
BFC	Base Function Codes
MTC	Memory Transfer Counter
BTC	Base Transfer Counter
MAR	Memory Address Register
DAR	Device Address Register
BAR	Base Address Register
GCR	General Control Register

APPENDIX H

SCSI CONTROLLER REGISTERS

All addresses are offset from the base address of the CC74 module.

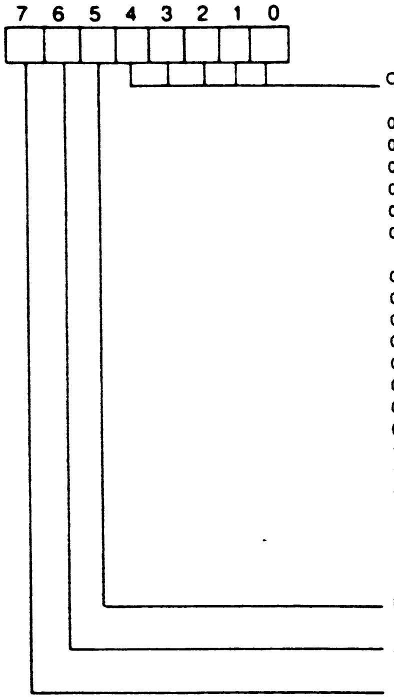
$120		DATNCR1	$121
		COMNCR	$123
		CNTNCR	$125
		DESIDNCR	$127
		AUXNCR	$129
		IDNCR	$12B
		IRQNCR	$12D
		SRCIDNCR	$12F
		DATNCR2	$131
		DIAGNCR	$133
			$135
			$137
		TCHINCR	$139
		TCMINCR	$13B
		TCLONCR	$13D
$13E		RESERVED	$13F

NCR 5385/6 REGISTER DEFINITIONS

DATNCR1	DATA REGISTER 1
COMNCR	COMMAND REGISTER
CNTNCR	CONTROL REGISTER
DESIDNCR	DESTINATION ID REGISTER
AUXNCR	AUXILIARY STATUS
IDNCR	ID REGISTER
IRQNCR	INTERRUPT REGISTER
SRCIDNCR	SOURCE ID REGISTER
DATNCR2	DATA REGISTER 2 (NCR 5386 only)
DIAGNCR	DIAGNOSTIC STATUS
TCHINCR	TRANSFER COUNTER HIGH BYTE (MSB)
TCMINCR	TRANSFER COUNTER MIDDLE BYTE
TCLONCR	TRANSFER COUNTER LOW BYTE (LSB)

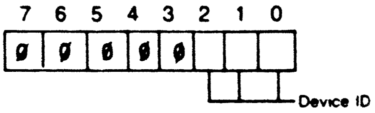
INTERNAL REGISTERS

COMMAND REGISTER

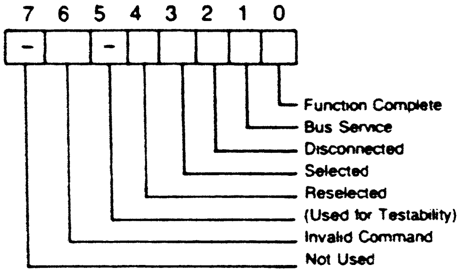


00000	Chip Reset
00001	Disconnect
00010	Pause
00011	Set ATN
00100	Message Accepted
00101	Chip Disabled
01000	Select w/ATN
01001	Select w/o ATN
01010	Reselect
01011	Diagnostic Data Turnaround
01100	Receive Command
01101	Receive Data
01110	Receive Message Out
01111	Received Unspecified Info Out
10000	Send Status
10001	Send Data
10010	Send Message In
10011	Send Unspecified Info In
10100	Transfer Info
10101	Transfer Pad

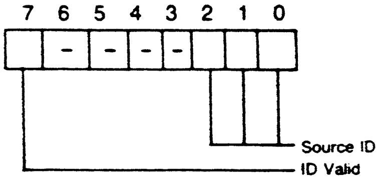
ID REGISTER



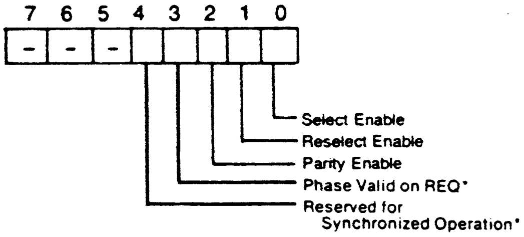
INTERRUPT REGISTER



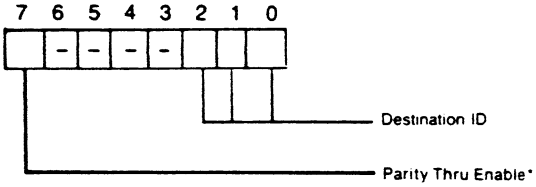
SOURCE ID REGISTER



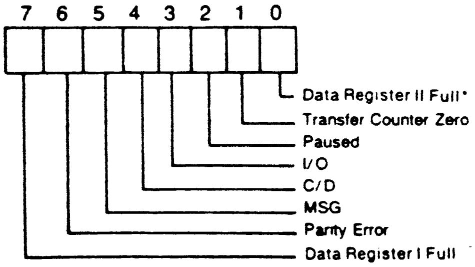
CONTROL REGISTER



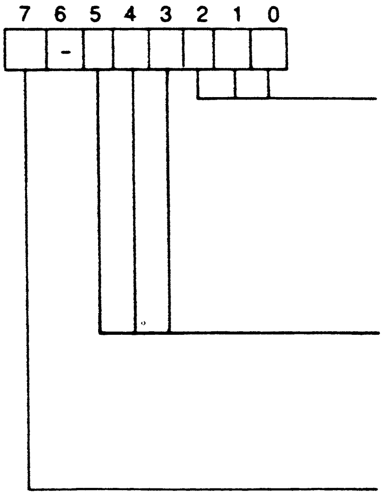
DESTINATION ID REGISTER



AUXILIARY STATUS REGISTER



DIAGNOSTIC STATUS REGISTER



Self-diagnostic Status

000	Successful Completion
001	Unconditional Branch Fail
010	Data Reg. Full Failed
011	Initial Conditions Incorrect
100	Initial Command Bits Incorrect
101	Diagnostic Flag Failed
110	Data Turnaround Failed
111	Not Used

Diagnostic Command Status

001	Turnaround Miscompare (Initial)
010	Turnaround Miscompare (Final)
011	Turnaround Good Parity
100	Turnaround Bad Parity

Self-diagnostic Complete

TRANSFER COUNTER

A3	A2	A1	A0	SELECTED BYTE
1	1	0	0	Most Significant Byte
1	1	0	1	Middle Byte
1	1	1	0	Least Significant Byte

*NCR 5386 ONLY

APPENDIX I

CONTROL REGISTER

```
             CONTROL REGISTER (bit)

             7  6  5  4  3  2  1  0
             -----------------------
     $101   |  |  |  |  |  |  |  |  |
             -----------------------
             |  |  |  |  |  |  |  |
             |  |  |  |  |  |  |   ---  IRQLV0
             |  |  |  |  |  |   ------  IRQLV1    IRQ LEVEL
             |  |  |  |  |   ---------  IRQLV2
             |  |  |  |   ------------  ENIRQ     IRQ ENABLE
             |  |  |   ---------------  REQLV0    REQUEST LEVEL
             |  |   ------------------  REQLV1
             |   ---------------------  spare
              ------------------------  LRST      SCSI RESET
```

APPENDIX J

CC74 PROGRAM EXAMPLE

PROGRAM EXAMPLE FOR CC-74

DMA-SCSI INTERFACE MODULE

The programs included in this manual are provided for users and system programmers who want to write their own SCSI driver routines for the CC-74 module. The two program listings which are given in this document are; a program with test routines, and a complete device driver for the OS-9/68K* operating system. Both programs are written in assembler and use the systype.d file for the global system definitions.

Users are free to use this information as it is, or make changes to the programs as seem appropriate to fit their application.

Compcontrol disclaims any implied warranties and assumes no responsibility for inaccuracies.

* - OS-9/68K is a registered trademark of Microware Systems Corporation.

```
* System definition for CompControl CC74   SCSI interface board (DMA)
   opt -l

* * * * * * * * * *
* Edition History
*    date     comments                                          by
* --------- ------------------------------------------------ -------
*  01/04/85   first implementation CC74 system equates         DJB
*

* board base address
CC74_BASE equ $fffe00

* hardware control register cc-74
CONTROL74 equ $101 offset from cc-74 base

* NCR 5385 register offset def's
DATNCR   equ $121  data register to scsi bus
COMNCR   equ $123  command register
CNTNCR   equ $125  control register
DESIDNCR equ $127 destination id register
AUXNCR   equ $129 auxiliary status register
IDNCR    equ $12B id register
IRQNCR   equ $12D interrupt register
SRCIDNCR equ $12F source id register
DIAGNCR  equ $133 diagnostic status register
TFRNCR   equ $139 transfer counter register
TCMONCR  equ $139 transfer counter most sign. byte
TCMINCR  equ $13B transfer counter middle byte
TCLENCR  equ $13D transfer counter least sign. byte

* DMA 68450 base address
CHNL0 equ $00
CHNL1 equ $40
CHNL2 equ $80
CHNL3 equ $C0
GENCR equ CHNL0+$FF

* DMA 68450 device definitions
CSR equ 0 channel status register
CER equ 1 channel error register
DCR equ 4 device control register
OCR equ 5 operation control register
SCR equ 6 sequence control register
CHCR equ 7 channel control register
MTC equ $A memory transfer counter
MAR equ $C memory address register
DAR equ $14 device address register
BTR equ $1A base transfer register
BAR equ $1C base address register

NIV equ $25 normal interrupt vector
EIV equ $27 error interrupt vector
CPR equ $2D channel priority register
```

```
MFC equ $29 memory function codes
DFC equ $31 device function codes
BFC equ $39 base function codes

*
* device control register (R/W)
*

BurstMod equ $00 burst transfer mode
StealMod equ $80 cycle steal mode without hold
StealHld equ $C0 cycle steal mode with hold

Dev68000 equ $00 68000 compatible device, explicitly addressed
Devc6800 equ $10 6800 compatible device, explicitly addressed
DevAck   equ $20 device with *ACK, implicitly addressed
DevAckRy equ $30 device with *ACK and *READY, implicitly addressed

Dev8Bit  equ $00 device port 8 bit
Dev16Bit equ $08 device port 16 bit
DevSiz_B equ 3 bit number of device port size

StatInp  equ 0 status input - peripheral ctl line
StatInpI equ 1 status input with interrupt
StartPls equ 2 start pulse, negative 1/8 clk
AbortInp equ 3 abort input

 page
*
* Operation control register (R/W)
*

MemToDev equ $00 transfer from memory to device
DevToMem equ $80 transfer from device to memory
XfrDir_B equ 7 transfer direction bit number

ByteSize equ $00 operation size = byte
WordSize equ $10 operation size = word
LongSize equ $20 operation size = long

ChainDis equ $0 chain operation disabled
ChainArr equ $8 array chaining enabled
ChainLnk equ $C linked chaining enabled

AuReqLim equ 0 auto request at rate set by GCR
AuReqMax equ 1 auto request at maximum rate
ReqInit  equ 2 *REQ line intitiates all operand transfers
ReqInitA equ 3 auto request first xfr, *REQ for all others


*
* Sequence Control Register (R/W)
*

MemNoCnt equ 0 memory address register does not count
MemCntUp equ 4 memory address register counts up
```

```
MemCntDn equ 8 memory address register counts down

DevNoCnt equ 0 device address register does not count
DevCntUp equ 1 device address register counts up
DevCntDn equ 2 device address register counts down


*
* Channel Control Register (R/W)
*

NoOpPend equ $00 no operation is pending
StartOp  equ $80 start operation
Start_B  equ 7 bit number of start operation bit

NoContin equ $00 no continue operation is pending
ContinOp equ $40 continue operation
Contin_B equ 6 bit number of continue op bit

OpNoHalt equ $00 operation not halted
OpHalted equ $20 operation halted
Halted_B equ 5 bit number of halted op bit

NoAbort  equ $00 operation not aborted
OpAbort  equ $10 operation aborted
Abort_B  equ 4 bit number of abort op bit

IntrptDi equ 0 interrupts disabled
IntrptEn equ 8 interrupts enabled
Intrpt_B equ 3 bit number of interrupt enable

 page
*
* Channel Status Register (R/W)
*
*     writing a one into any bit clears that status
*     any written zero bits do not affect the status
*

OpNoComp equ $00 operation incomplete
OperComp equ $80 operation complete
OpComp_B equ 7 bit number of operation complete bit

BlkNoCmp equ $00 block transfer incomplete
BlkComp  equ $40 block transfer complete
BlkCmp_B equ 6 bit number of block transfer complete bit

DevTrmAb equ $00 device termination abnormal
DevTrmNo equ $20 device termination normal
DevTrm_B equ 5 bit number of device termination status

ErrorSet equ $10 error occurred and is noted in CER
Error_B  equ 4 bit number of error flag bit

ActiveCh equ 8 channel considered active
```

```
Active_B equ 3 bit number of active channel flag bit

PCLTrans equ 2 transition occurred on *PCL
PCLTrn_B equ 1 bit number of PCL transition flag bit

PCLLow   equ 0 *PCL line low
PCLHigh  equ 1 *PCL line high
PCLSts_B equ 0 bit number of *PCL status bit


*
* Channel Error Register (R only)
*

ErConfig equ $01 configuration error
ErOpTimg equ $02 operation timing error

ErAdrMem equ $05 memory address error
ErAdrDev equ $06 device address error
ErAdrBas equ $07 base address error

ErBusMem equ $09 memory bus error
ErBusDev equ $0A device bus error
ErBusBas equ $0B base bus error

ErCntMem equ $0D memory count error
ErCntDev equ $0E device count error
ErCntBas equ $0F base count error

ErEAbort equ $10 external abort
ErSAbort equ $11 software abort

 page
*
* Channel Priority Register (R/W)
*

ChPrior0 equ 0 channel priority of zero
ChPrior1 equ 1 channel priority of one
ChPrior2 equ 2 channel priority of two
ChPrior3 equ 3 channel priority of three


*
* Function Code Registers (R/W)
*

UserData equ 1 user data address access
UserProg equ 2 user program address access
SupvData equ 5 supervisor data address access
SupvProg equ 6 supervisor program address access



*
```

```
* General Control Register (R/W)
*

BurstTim equ $C mask for burst time
BandwRat equ $3 mask for bandwidth ratio

 opt 1
* end of file
```

```
00001                          nam      tstncr
00002                          ttl      OS-9/68000     CC-74 Test Routines
00003 *
00004 * Testing CC-74 with NCR 5385 with Xebec controller S 1410-A
00005 *  using DMA controller 68450 and non standard Irq routine
00006 *
00007 * Created at:  08 january 1985
00008 * Created by:  D. J. Bosma and N. Noordam
00009 *
00010
00011                          use      defsfile
00012
00013  00000001 Edition    set      1
00014  00000101 Typ_Lang   set      (Prgrm<<8)+Objct
00015  00008000 Attr_Rev   set      (ReEnt<<8)+0
00016
00017                          psect    tstncr,Typ_Lang,Attr_Rev,Edition,100,Start
00018
00019  00000020 SPACE      equ      $20
00020  00000020 space      equ      $20
00021  0000000d CR         equ      $0d
00022  0000000d cr         equ      $0d
00023  0000000a LF         equ      $0a
00024  0000000a lf         equ      $0a
00025  00000000 StdIn      equ      0
00026  00000001 StdOut     equ      1
00027  00000002 StdErr     equ      2
00028  0000004a EscFlg     equ      CHNL1+MTC      Use this registers for irq communication
00029  0000004c SavSts     equ      CHNL1+MAR
00030
00031                          vsect
00032
00033  00000000 Char_Buf:  ds.b     10
00034  0000000a HD_cmd:    ds.b     1
00035  0000000b HD_psn:    ds.b     3
00036  0000000e HD_blk:    ds.b     1
00037  0000000f HD_ctrl:   ds.b     1
00038  00000010 HD_status: ds.b     1
00039  00000011 HD_msg:    ds.b     1
00040  00000012 HD_buffer: ds.b     256
00041  00000112 HD_sense:  ds.b     4
00042  00000116 Drive_no:  ds.b     1
00043  00000000            ends
00044 *
00045 * Start of program, display menu
00046 *
00047 0000 2a7c Start:     movea.l  #CC74_BASE,a5  get base addres of CC-74
           00fffe00
00048 0006 6100            bsr      Init74         init cc-74
           05d0
00049 000a 6400            bcc      Menu           no error's
           001a
00050 000e 41fa            lea      msg2(pc),a0
           0cfa
00051 0012 303c            move.w   #StdOut,d0
           0001
00052 0016 223c            move.l   #szmsg2,d1
           00000019
00053 001c=4e40            os9      I$Write        print "init cc-74 error"
           0000
00054 0020 4281            clr.l    d1
```

```
00055 0022=4e40            os9      F$Exit        stop test prog
           0000
00056 0026 41fa Menu:      lea      msg1(pc),a0
           0b36
00057 002a 303c            move.w   #StdOut,d0
           0001
00058 002e 223c            move.l   #szmsg1,d1
           000001ac
00059 0034=4e40            os9      I$Write       print menu
           0000
00060 0038 303c            move.w   #StdIn,d0
           0000
00061 003c 7202            moveq    #2,d1
00062 003e 41ee            lea      Char_Buf(a6),a0
           0000
00063 0042=4e40            os9      I$ReadLn      get one character
           0000
00064 0046 102e            move.b   Char_Buf(a6),d0
           0000
00065 004a b03c            cmp.b    #'1',d0
           0031
00066 004e 6606            bne.s    Menu1
00067 0050 6100            bsr      Test1
           0176
00068 0054 60d0            bra.s    Menu
00069 0056 b03c Menu1      cmp.b    #'2',d0
           0032
00070 005a 6606            bne.s    Menu2
00071 005c 6100            bsr      Test2
           026c
00072 0060 60c4            bra.s    Menu
00073 0062 b03c Menu2      cmp.b    #'3',d0
           0033
00074 0066 6606            bne.s    Menu3
00075 0068 6100            bsr      Test3
           02ca
00076 006c 60b8            bra.s    Menu
00077 006e b03c Menu3      cmp.b    #'4',d0
           0034
00078 0072 6606            bne.s    Menu4
00079 0074 6100            bsr      Test4
           0328
00080 0078 60ac            bra.s    Menu
00081 007a b03c Menu4      cmp.b    #'5',d0
           0035
00082 007e 6606            bne.s    Menu5
00083 0080 6100            bsr      Test5
           0386
00084 0084 60a0            bra.s    Menu
00085 0086 b03c Menu5      cmp.b    #'6',d0
           0036
00086 008a 6606            bne.s    Menu6
00087 008c 6100            bsr      Test6
           03f2
00088 0090 6094            bra.s    Menu
00089 0092 b03c Menu6      cmp.b    #'7',d0
           0037
00090 0096 6606            bne.s    Menu7
00091 0098 6100            bsr      Test7
           045e
00092 009c 6088            bra.s    Menu
```

```
00093 009e b03c Menu7      cmp.b    #'8',d0
           0038
00094 00a2 6608            bne.s    Menu8
00095 00a4 6100            bsr      Test8
           04bc
00096 00a8 6000            bra      Menu
           ff7c
00097 00ac b03c Menu8      cmp.b    #'9',d0
           0039
00098 00b0 6608            bne.s    Menu9
00099 00b2 6100            bsr      Test9
           0518
00100 00b6 6000            bra      Menu
           ff6e
00101 00ba 0200 Menu9      andi.b   #$df,d0         make upper case
           00df
00102 00be b03c            cmp.b    #'A',d0
           0041
00103 00c2 6608            bne.s    MenuA
00104 00c4 6100            bsr      Buff1
           0046
00105 00c8 6000            bra      Menu
           ff5c
00106 00cc b03c MenuA      cmp.b    #'B',d0
           0042
00107 00d0 6608            bne.s    MenuB
00108 00d2 6100            bsr      Buff2
           00be
00109 00d6 6000            bra      Menu
           ff4e
00110 00da b03c MenuB      cmp.b    #'C',d0
           0043
00111 00de 6608            bne.s    MenuC
00112 00e0 6100            bsr      Askid
           0818
00113 00e4 6000            bra      Menu
           ff40
00114 00e8 b03c MenuC      cmp.b    #'D',d0
           0044
00115 00ec 6608            bne.s    MenuD
00116 00ee 6100            bsr      Askdn
           0856
00117 00f2 6000            bra      Menu
           ff32
00118 00f6 41fa MenuD      lea      msg4(pc),a0
           0c52
00119 00fa 303c            move.w   #StdOut,d0
           0001
00120 00fe 223c            move.l   #szmsg4,d1
           00000009
00121 0104=4e40            os9      I$Write         print "What?"
           0000
00122 0108 6000            bra      Menu
           ff1c
00123 *
00124 * Buff1
00125 *     Show the HD_buffer
00126 *
00127 010c 6100 Buff1:     bsr      wrcrlf
           009c
00128 0110 43ee            lea      HD_buffer(a6),a1 HD_buffer start buffer
```

```
           0012
00129 0114 243c           move.l   #15,d2         Number of line's (16)
           0000000f
00130 011a 263c buff11    move.l   #15,d3         Number of bytes/line
           0000000f
00131 0120 41ee buff12    lea      Char_Buf(a6),a0 start output buffer
           0000
00132 0124 5688           addq.l   #3,a0
00133 0126 10bc           move.b   #$20,(a0)
           0020
00134 012a 113c           move.b   #$20,-(a0)     put two spaces in outputbuffer
           0020
00135 012e 1a19           move.b   (a1)+,d5       get first byte
00136 0130 283c           move.l   #1,d4          number of nibbles
           00000001
00137 0136 1c05 buff13    move.b   d5,d6          use d3 as scratch
00138 0138 e88d           lsr.l    #4,d5          get nibble for next loop
00139 013a 0206           andi.b   #$f,d6
           000f
00140 013e dc3c           add.b    #$30,d6
           0030
00141 0142 bc3c           cmp.b    #$39,d6
           0039
00142 0146 6304           bls.s    buff14
00143 0148 dc3c           add.b    #7,d6
           0007
00144 014c 1106 buff14    move.b   d6,-(a0)
00145 014e 51cc           dbra     d4,buff13      next nibble
           ffe6
00146 0152 303c           move.w   #StdOut,d0
           0001
00147 0156 223c           move.l   #4,d1          number of bytes
           00000004
00148 015c=4e40           os9      I$Write
           0000
00149 0160 51cb           dbra     d3,buff12      next byte
           ffbe
00150 0164 6100           bsr      wrcrlf
           0044
00151 0168 51ca           dbra     d2,buff11      next line
           ffb0
00152 016c 6100           bsr      wrcrlf
           003c
00153 0170 41fa           lea      msg7(pc),a0
           0c29
00154 0174 303c           move.w   #StdOut,d0
           0001
00155 0178 223c           move.l   #szmsg7,d1
           0000000e
00156 017e=4e40           os9      I$Write
           0000
00157 0182 303c           move.w   #StdOut,d0
           0001
00158 0186 7201           moveq.l  #1,d1
00159 0188 41ee           lea      Char_Buf(a6),a0
           0000
00160 018c=4e40           os9      I$ReadLn
           0000
00161 0190 4e75           rts
00162 *
00163 * Buff2
```

```
00164 *   Fill HD_buffer with constant
00165 *
00166 0192 6100 Buff2:     bsr      AskCon         get constant, return in d0
           0724
00167 0196 6510            bcs.s    buff22         error, leave
00168 0198 41ee            lea      HD_buffer(a6),a0
           0012
00169 019c 223c            move.l   #63,d1         HD_buffer size divided by 4
           0000003f
00170 01a2 20c0 buff21     move.l   d0,(a0)+       fill with constant
00171 01a4 51c9            dbra     d1,buff21
           fffc
00172 01a8 4e75 buff22     rts
00173 *
00174 * CR/LF
00175 *   print cr/lf
00176 *
00177 01aa 41ee wrcrlf     lea      Char_Buf(a6),a0
           0000
00178 01ae 5288            addq.l   #1,a0
00179 01b0 10bc            move.b   #$0d,(a0)
           000d
00180 01b4 113c            move.b   #$0a,-(a0)
           000a
00181 01b8 303c            move.w   #StdOut,d0
           0001
00182 01bc 223c            move.l   #2,d1
           00000002
00183 01c2=4e40            os9      I$Write        write CR/LF
           0000
00184 01c6 4e75            rts
00185 *
00186 * TEST1 : internal diagnostic
00187 *         ram test
00188 *         init drive characteristics
00189 *
00190 01c8 2d7c Test1:     move.l   #0,HD_cmd(a6)  clr psn
           00000000
           000a
00191 01d0 102e            move.b   Drive_no(a6),d0
           0116
00192 01d4 812e            or.b     d0,HD_psn(a6)
           000b
00193 01d8 1d7c            move.b   #$e4,HD_cmd(a6) diagn cmd
           00e4000a
00194 01de 43ee            lea      HD_cmd(a6),a1  cmd ptr
           000a
00195 01e2 323c            move.w   #6,d1          get count
           0006
00196 01e6 6100            bsr      NCRcmd
           045a
00197 01ea 6530            bcs.s    test1lex
00198 01ec 47ee            lea      HD_status(a6),a3
           0010
00199 01f0 6100            bsr      NCRstatus
           04ea
00200 01f4 47ee            lea      HD_msg(a6),a3
           0011
00201 01f8 6100            bsr      NCRmsg
           0500
00202 01fc 082e            btst     #1,HD_status(a6)
```

```
           00010010
00203 0202 6700              beq      test12         skip if no error
           001a
00204 0206 41fa              lea      emsg1(pc),a0
           0bf9
00205 020a 303c              move.w   #StdOut,d0
           0001
00206 020e 223c              move.l   #szemsg1,d1
           00000025
00207 0214=4e40              os9      I$Write        write error message
           0000
00208 0218 6100              bsr      senstat
           060a
00209 021c 4e75 test11ex     rts
00210
00211 * start of ram test
00212 021e 1d7c test12:     move.b   #$e0,HD_cmd(a6)
           00e0000a
00213 0224 43ee              lea      HD_cmd(a6),a1  cmd ptr
           000a
00214 0228 323c              move.w   #6,d1          count
           0006
00215 022c 6100              bsr      NCRcmd
           0414
00216 0230 6530              bcs.s    test12ex
00217 0232 47ee              lea      HD_status(a6),a3
           0010
00218 0236 6100              bsr      NCRstatus
           04a4
00219 023a 47ee              lea      HD_msg(a6),a3
           0011
00220 023e 6100              bsr      NCRmsg
           04ba
00221 0242 082e              btst     #1,HD_status(a6)
           00010010
00222 0248 6700              beq      test13         skip if no error
           001a
00223 024c 41fa              lea      emsg2(pc),a0
           0bd8
00224 0250 303c              move.w   #StdOut,d0
           0001
00225 0254 2239              move.l   szemsg2,d1
           0000001a
00226 025a=4e40              os9      I$Write
           0000
00227 025e 6100              bsr      senstat
           05c4
00228 0262 4e75 test12ex     rts
00229
00230 * start init drive parameters
00231 0264 1d7c test13:     move.b   #$0c,HD_cmd(a6) init drive param
           000c000a
00232 026a 43ee              lea      HD_cmd(a6),a1
           000a
00233 026e 323c              move.w   #6,d1
           0006
00234 0272 6100              bsr      NCRcmd
           03ce
00235 0276 653c              bcs.s    test13ex
00236 0278 45fa              lea      HD_par88(pc),a2 get param table
           08dc
```

```
00237 027c 343c           move.w   #8,d2
           0008
00238 0280 6100           bsr      NCRdatwr
           052a
00239 0284 47ee           lea      HD_status(a6),a3
           0010
00240 0288 6100           bsr      NCRstatus
           0452
00241 028c 47ee           lea      HD_msg(a6),a3
           0011
00242 0290 6100           bsr      NCRmsg
           0468
00243 0294 082e           btst     #1,HD_status(a6)
           00010010
00244 029a 6700           beq      test14
           001a
00245 029e 41fa           lea      emsg3(pc),a0
           0ba0
00246 02a2 303c           move.w   #StdOut,d0
           0001
00247 02a6 223c           move.l   #szemsg3,d1
           00000027
00248 02ac=4e40           os9      I$Write
           0000
00249 02b0 6100           bsr      senstat
           0572
00250 02b4 4e75 test13ex  rts
00251
00252 02b6 41fa test14    lea      msg3(pc),a0
           0a6b
00253 02ba 303c           move.w   #StdOut,d0
           0001
00254 02be 223c           move.l   #szmsg3,d1
           00000027
00255 02c4=4e40           os9      I$Write
           0000
00256 02c8 4e75           rts
00257
00258
00259 *
00260 * Test2
00261 *       Test drive ready command
00262 *
00263 * Input :
00264 *
00265 * Exit  :
00266 *
00267 02ca 2d7c Test2:    move.l   #0,HD_cmd(a6)  clr psn
           00000000
           000a
00268 02d2 102e           move.b   Drive_no(a6),d0
           0116
00269 02d6 812e           or.b     d0,HD_psn(a6)
           000b
00270 02da 1d7c           move.b   #$00,HD_cmd(a6) Test drive ready cmd
           0000000a
00271 02e0 43ee           lea      HD_cmd(a6),a1  cmd ptr
           000a
00272 02e4 323c           move.w   #6,d1          get count
           0006
00273 02e8 6100           bsr      NCRcmd
```

```
           0358
00274 02ec 6530              bcs.s    test2ex
00275 02ee 47ee              lea      HD_status(a6),a3
           0010
00276 02f2 6100              bsr      NCRstatus
           03e8
00277 02f6 47ee              lea      HD_msg(a6),a3
           0011
00278 02fa 6100              bsr      NCRmsg
           03fe
00279 02fe 082e              btst     #1,HD_status(a6)
           00010010
00280 0304 6700              beq      test21         skip if no error
           001a
00281 0308 41fa              lea      emsg4(pc),a0
           0b5d
00282 030c 303c              move.w   #StdOut,d0
           0001
00283 0310 223c              move.l   #szemsg4,d1
           00000022
00284 0316=4e40              os9      I$Write        write error message
           0000
00285 031a 6100              bsr      senstat
           0508
00286 031e 4e75 test2ex      rts
00287 0320 41fa test21       lea      msg3(pc),a0
           0a01
00288 0324 303c              move.w   #StdOut,d0
           0001
00289 0328 223c              move.l   #szmsg3,d1
           00000027
00290 032e=4e40              os9      I$Write
           0000
00291 0332 4e75              rts
00292 *
00293 * Test3
00294 *     Format Drive
00295 * Input :
00296 *
00297 * Exit  :
00298 *
00299 0334 2d7c Test3:       move.l   #0,HD_cmd(a6)  clr psn
           00000000
           000a
00300 033c 102e              move.b   Drive_no(a6),d0
           0116
00301 0340 812e              or.b     d0,HD_psn(a6)
           000b
00302 0344 1d7c              move.b   #$04,HD_cmd(a6) Format Drive cmd
           0004000a
00303 034a 43ee              lea      HD_cmd(a6),a1  cmd ptr
           000a
00304 034e 323c              move.w   #6,d1          get count
           0006
00305 0352 6100              bsr      NCRcmd
           02ee
00306 0356 6530              bcs.s    test3ex
00307 0358 47ee              lea      HD_status(a6),a3
           0010
00308 035c 6100              bsr      NCRstatus
           037e
```

```
00309 0360 47ee            lea      HD_msg(a6),a3
           0011
00310 0364 6100            bsr      NCRmsg
           0394
00311 0368 082e            btst     #1,HD_status(a6)
           00010010
00312 036e 6700            beq      test31         skip if no error
           001a
00313 0372 41fa            lea      emsg5(pc),a0
           0b15
00314 0376 303c            move.w   #StdOut,d0
           0001
00315 037a 223c            move.l   #szemsg5,d1
           0000001f
00316 0380=4e40            os9      I$Write        write error message
           0000
00317 0384 6100            bsr      senstat
           049e
00318 0388 4e75 test3ex    rts
00319 038a 41fa test31     lea      msg3(pc),a0
           0997
00320 038e 303c            move.w   #StdOut,d0
           0001
00321 0392 223c            move.l   #szmsg3,d1
           00000027
00322 0398=4e40            os9      I$Write
           0000
00323 039c 4e75            rts
00324 *
00325 * Test4
00326 *     Drive Diagnostic
00327 * Input :
00328 *
00329 * Exit  :
00330 *
00331 039e 2d7c Test4:     move.l   #0,HD_cmd(a6)  clr psn
           00000000
           000a
00332 03a6 102e            move.b   Drive_no(a6),d0
           0116
00333 03aa 812e            or.b     d0,HD_psn(a6)
           000b
00334 03ae 1d7c            move.b   #$e3,HD_cmd(a6) Drive Diagnostic cmd
           00e3000a
00335 03b4 43ee            lea      HD_cmd(a6),a1  cmd ptr
           000a
00336 03b8 323c            move.w   #6,d1          get count
           0006
00337 03bc 6100            bsr      NCRcmd
           0284
00338 03c0 6530            bcs.s    test4ex
00339 03c2 47ee            lea      HD_status(a6),a3
           0010
00340 03c6 6100            bsr      NCRstatus
           0314
00341 03ca 47ee            lea      HD_msg(a6),a3
           0011
00342 03ce 6100            bsr      NCRmsg
           032a
00343 03d2 082e            btst     #1,HD_status(a6)
           00010010
```

```
00344 03d8 6700            beq      test31          skip if no error
           ffb0
00345 03dc 41fa            lea      emsg6(pc),a0
           0aca
00346 03e0 303c            move.w   #StdOut,d0
           0001
00347 03e4 223c            move.l   #szemsg6,d1
           00000022
00348 03ea=4e40            os9      I$Write         write error message
           0000
00349 03ee 6100            bsr      senstat
           0434
00350 03f2 4e75 test4ex    rts
00351 03f4 41fa test41     lea      msg3(pc),a0
           092d
00352 03f8 303c            move.w   #StdOut,d0
           0001
00353 03fc 223c            move.l   #szmsg3,d1
           00000027
00354 0402=4e40            os9      I$Write
           0000
00355 0406 4e75            rts
00356 *
00357 * Test5
00358 *     Read One Sector
00359 * Input :
00360 *
00361 * Exit  :
00362 *
00363 0408 6100 Test5:     bsr      AskPsn          return in d0
           058a
00364 040c 6570            bcs.s    test52          error in PSN
00365 040e 2d40            move.l   d0,HD_cmd(a6)   store psn
           000a
00366 0412 102e            move.b   Drive_no(a6),d0
           0116
00367 0416 812e            or.b     d0,HD_psn(a6)
           000b
00368 041a 1d7c            move.b   #$08,HD_cmd(a6) Read sector(s) cmd
           0008000a
00369 0420 43ee            lea      HD_cmd(a6),a1   cmd ptr
           000a
00370 0424 323c            move.w   #6,d1           get count
           0006
00371 0428 6100            bsr      NCRcmd
           0218
00372 042c 653c            bcs.s    test5ex
00373 042e 343c            move.w   #$100,d2
           0100
00374 0432 45ee            lea      HD_buffer(a6),a2
           0012
00375 0436 6100            bsr      NCRdatrd
           02fc
00376 043a 47ee            lea      HD_status(a6),a3
           0010
00377 043e 6100            bsr      NCRstatus
           029c
00378 0442 47ee            lea      HD_msg(a6),a3
           0011
00379 0446 6100            bsr      NCRmsg
           02b2
```

```
00380 044a 082e            btst     #1,HD_status(a6)
           00010010
00381 0450 6700            beq      test51         skip if no error
           001a
00382 0454 41fa            lea      emsg7(pc),a0
           0a74
00383 0458 303c            move.w   #StdOut,d0
           0001
00384 045c 223c            move.l   #szemsg7,d1
           00000021
00385 0462=4e40            os9      I$Write        write error message
           0000
00386 0466 6100            bsr      senstat
           03bc
00387 046a 4e75 test5ex    rts
00388 046c 41fa test51     lea      msg3(pc),a0
           08b5
00389 0470 303c            move.w   #StdOut,d0
           0001
00390 0474 223c            move.l   #szmsg3,d1
           00000027
00391 047a=4e40            os9      I$Write
           0000
00392 047e 4e75 test52     rts
00393 *
00394 * Test6
00395 *     Write One Sector
00396 * Input :
00397 *
00398 * Exit  :
00399 *
00400 0480 6100 Test6:     bsr      AskPsn         return in d0
           0512
00401 0484 6570            bcs.s    test62         error in PSN
00402 0486 2d40            move.l   d0,HD_cmd(a6)  store psn
           000a
00403 048a 102e            move.b   Drive_no(a6),d0
           0116
00404 048e 812e            or.b     d0,HD_psn(a6)
           000b
00405 0492 1d7c            move.b   #$0a,HD_cmd(a6) Write sector(s) cmd
           000a000a
00406 0498 43ee            lea      HD_cmd(a6),a1  cmd ptr
           000a
00407 049c 323c            move.w   #6,d1          get count
           0006
00408 04a0 6100            bsr      NCRcmd
           01a0
00409 04a4 653c            bcs.s    test6ex
00410 04a6 343c            move.w   #$100,d2
           0100
00411 04aa 45ee            lea      HD_buffer(a6),a2
           0012
00412 04ae 6100            bsr      NCRdatwr
           02fc
00413 04b2 47ee            lea      HD_status(a6),a3
           0010
00414 04b6 6100            bsr      NCRstatus
           0224
00415 04ba 47ee            lea      HD_msg(a6),a3
           0011
```

```
00416 04be 6100             bsr      NCRmsg
           023a
00417 04c2 082e             btst     #1,HD_status(a6)
           00010010
00418 04c8 6700             beq      test61          skip if no error
           001a
00419 04cc 41fa             lea      emsg8(pc),a0
           0a1d
00420 04d0 303c             move.w   #StdOut,d0
           0001
00421 04d4 223c             move.l   #szemsg8,d1
           00000022
00422 04da=4e40             os9      I$Write         write error message
           0000
00423 04de 6100             bsr      senstat
           0344
00424 04e2 4e75 test6ex     rts
00425 04e4 41fa test61      lea      msg3(pc),a0
           083d
00426 04e8 303c             move.w   #StdOut,d0
           0001
00427 04ec 223c             move.l   #szmsg3,d1
           00000027
00428 04f2=4e40             os9      I$Write
           0000
00429 04f6 4e75 test62      rts
00430 *
00431 * Test7
00432 *     Seek to Sector command
00433 * Input :
00434 *
00435 * Exit  :
00436 *
00437 04f8 6100 Test7:      bsr      AskPsn          return in d0
           049a
00438 04fc 6562             bcs.s    test72          error in PSN
00439 04fe 2d40             move.l   d0,HD_cmd(a6)   store psn
           000a
00440 0502 102e             move.b   Drive_no(a6),d0
           0116
00441 0506 812e             or.b     d0,HD_psn(a6)
           000b
00442 050a 1d7c             move.b   #$0b,HD_cmd(a6) Seek sector cmd
           000b000a
00443 0510 43ee             lea      HD_cmd(a6),a1   cmd ptr
           000a
00444 0514 323c             move.w   #6,d1           get count
           0006
00445 0518 6100             bsr      NCRcmd
           0128
00446 051c 652e             bcs.s    test7ex
00447 051e 47ee             lea      HD_status(a6),a3
           0010
00448 0522 6100             bsr      NCRstatus
           01b8
00449 0526 47ee             lea      HD_msg(a6),a3
           0011
00450 052a 6100             bsr      NCRmsg
           01ce
00451 052e 082e             btst     #1,HD_status(a6)
           00010010
```

```
00452 0534 6718           beq.s    test71         skip if no error
00453 0536 41fa           lea      emsg9(pc),a0
           09d5
00454 053a 303c           move.w   #StdOut,d0
           0001
00455 053e 223c           move.l   #szemsg9,d1
           00000020
00456 0544=4e40           os9      I$Write        write error message
           0000
00457 0548 6100           bsr      senstat
           02da
00458 054c 4e75 test7ex   rts
00459 054e 41fa test71    lea      msg3(pc),a0
           07d3
00460 0552 303c           move.w   #StdOut,d0
           0001
00461 0556 223c           move.l   #szmsg3,d1
           00000027
00462 055c=4e40           os9      I$Write
           0000
00463 0560 4e75 test72    rts
00464 *
00465 * Test8
00466 *    Recalibrate command
00467 * Input :
00468 *
00469 * Exit  :
00470 *
00471 0562 2d7c Test8:    move.l   #0,HD_cmd(a6)  clr psn
           00000000
           000a
00472 056a 102e           move.b   Drive_no(a6),d0
           0116
00473 056e 812e           or.b     d0,HD_psn(a6)
           000b
00474 0572 1d7c           move.b   #$01,HD_cmd(a6) Recalibrate cmd
           0001000a
00475 0578 43ee           lea      HD_cmd(a6),a1  cmd ptr
           000a
00476 057c 323c           move.w   #6,d1          get count
           0006
00477 0580 6100           bsr      NCRcmd
           00c0
00478 0584 6530           bcs.s    test8ex
00479 0586 47ee           lea      HD_status(a6),a3
           0010
00480 058a 6100           bsr      NCRstatus
           0150
00481 058e 47ee           lea      HD_msg(a6),a3
           0011
00482 0592 6100           bsr      NCRmsg
           0166
00483 0596 082e           btst     #1,HD_status(a6)
           00010010
00484 059c 6700           beq      test81         skip if no error
           001a
00485 05a0 41fa           lea      emsg10(pc),a0
           098b
00486 05a4 303c           move.w   #StdOut,d0
           0001
00487 05a8 223c           move.l   #szemsg10,d1
```

```
          0000001d
00488 05ae=4e40            os9      I$Write        write error message
          0000
00489 05b2 6100            bsr      senstat
          0270
00490 05b6 4e75 test8ex    rts
00491 05b8 41fa test81     lea      msg3(pc),a0
          0769
00492 05bc 303c            move.w   #StdOut,d0
          0001
00493 05c0 223c            move.l   #szmsg3,d1
          00000027
00494 05c6=4e40            os9      I$Write
          0000
00495 05ca 4e75            rts
00496 *
00497 * exit routine test9
00498 *
00499  000005cc Test9      equ      *
00500 05cc 08ad            bclr     #3,CONTROL74(a5) disable irq
          00030101
00501 05d2 4281            clr.l    d1
00502 05d4=4e40            os9      F$Exit         return to os9
          0000
00503
00504 *
00505 * Init74:
00506 *
00507 * Following is the Initialise routine
00508 *
00509 * The NCR chip and the SCSI bus are reset
00510 *
00511 * entry: a5 = base address of CC-74
00512 *        a6 = static storage pointer
00513 * exit : carry set on error
00514 *      NCR diagnostic register in d3
00515 *
00516 05d8 08ed Init74:    bset     #7,CONTROL74(a5) Reset SCSI bus
          00070101
00517 05de 08ad            bclr     #7,CONTROL74(a5) clr reset bit
          00070101
00518 05e4 1b7c            move.b   #0,COMNCR(a5)  NCR chip reset
          00000123
00519 05ea 203c            move.l   #350,d0        Init timout
          0000015e
00520 05f0 51c8 Init741    dbra     d0,Init744     Time Out?
          0004
00521 05f4 6046            bra.s    Init742        yes
00522 05f6 082d Init744    btst     #7,DIAGNCR(a5) wait for command ready
          00070133
00523 05fc 67f2            beq.s    Init741
00524 05fe 162d            move.b   DIAGNCR(a5),d3 get diagnostic results
          0133
00525 0602 b63c            cmp.b    #$80,d3
          0080
00526 0606 6634            bne.s    Init742
00527 0608 41fa            lea      IrqEntry(pc),a0
          049a
00528 060c 23c8            move.l   a0,$200
          00000200
00529 0612 1b7c            move.b   #0,DESIDNCR(a5) set default target id
```

```
           00000127
00530 0618 1d7c            move.b   #0,Drive_no(a6) set default drive nr.
           00000116
00531 061e 1d7c            move.b   #$c5,HD_ctrl(a6) init control field, retries and step-ra
te
           00c5000f
00532 0624 1d7c            move.b   #1,HD_blk(a6)  init block count, always one block
           0001000e
00533 062a 1b7c            move.b   #$36,CONTROL74(a5) set bus request level 3, irq level 6
           00360101
00534 0630 08ed            bset     #3,CONTROL74(a5) enable irq
           00030101
00535 0636 023c            andi.b   #$fe,ccr        clear carry no error
           00fe
00536 063a 4e75            rts
00537
00538 063c 003c Init742    ori.b    #1,ccr          set carry, error occured
           0001
00539 0640 4e75            rts
00540
00541
00542 *
00543 * start of subroutines
00544 *
00545 * NCRcmd : give the NCR5385 a command
00546 * entry : a1  cmd pointer
00547 *         a5  base address CC74
00548 *         a6  static storage
00549 *         d1  byte count (word)
00550 *
00551 0642 3f01 NCRcmd:    move.w   d1,-(sp)        save byte count
00552 0644 422d            clr.b    TFRNCR(a5)
           0139
00553 0648 422d            clr.b    TFRNCR+2(a5)
           013b
00554 064c 1b7c            move.b   #$ff,TFRNCR+4(a5) arbitration time out
           00ff013d
00555 0652 6100 ncmd0      bsr      TestIrq         is there an interrupt pending
           0246
00556 0656 66fa            bne.s    ncmd0           yes reset by reading irq again
00557           ncmd1
00558 *   move.b #$04,COMNCR(a5) 'Soft reset ??'
00559 0658 1b7c            move.b   #$09,COMNCR(a5) select XEBEC
           00090123
00560 065e 6100            bsr      WaitIrq         bit #0 'function complete' should be set
           0244
00561 0662 0c01            cmpi.b   #1,d1
           0001
00562 0666 670c            beq.s    ncmdc           'function complete' was set
00563 0668 0c01            cmpi.b   #4,d1
           0004
00564 066c 6700            beq      Timouterr       'disconnected set' no response
           003a
00565 0670 6000            bra      Invalirq        'some bit set'
           0050
00566 0674 6100 ncmdc      bsr      WaitIrq         bit #1 'bus service' should be set
           022e
00567 0678 321f            move.w   (sp)+,d1        restore byte count
00568 067a 1b41            move.b   d1,TFRNCR+4(a5) init transfer count
           013d
00569 067e 6100 ncmd2      bsr      TestIrq         is there an interrupt pending
           021a
```

```
00570 0682 66fa            bne.s    ncmd2          yes reset by reading irq again
00571 0684 1b7c            move.b   #$14,COMNCR(a5) transfer info cmd
           00140123
00572 068a 102d ncmd3      move.b   AUXNCR(a5),d0
           0129
00573 068e 0800            btst     #7,d0
           0007
00574 0692 66f6            bne.s    ncmd3          wait for data reg full = 0
00575 0694 0800            btst     #1,d0
           0001
00576 0698 6608            bne.s    ncmd4          transfer count zero?
00577 069a 1019            move.b   (a1)+,d0       no, do another byte
00578 069c 1b40            move.b   d0,DATNCR(a5)
           0121
00579 06a0 60e8            bra.s    ncmd3
00580 06a2 303c ncmd4      move.w   #0,d0
           0000
00581 06a6 4e75            rts
00582
00583           Timouterr:
00584 06a8 41fa            lea      emsg14(pc),a0
           08f8
00585 06ac 303c            move.w   #StdOut,d0
           0001
00586 06b0 223c            move.l   #szemsg14,d1
           0000001a
00587 06b6=4e40            os9      I$Write
           0000
00588 06ba 321f            move.w   (sp)+,d1       restore byte count
00589 06bc=003c            ori.b    #Carry,ccr
           0000
00590 06c0 4e75            rts
00591
00592           Invalirq:
00593 06c2 41fa            lea      emsg15(pc),a0
           08f8
00594 06c6 303c            move.w   #StdOut,d0
           0001
00595 06ca 223c            move.l   #szemsg15,d1
           00000025
00596 06d0 321f            move.w   (sp)+,d1       restore byte count
00597 06d2=4e40            os9      I$Write
           0000
00598 06d6=003c            ori.b    #Carry,ccr
           0000
00599 06da 4e75            rts
00600 *
00601 * NCRstatus : get status byte
00602 * entry : a3  status ptr
00603 *         a5  base address
00604 *         a6  static storage
00605 *
00606 06dc 1b7c NCRstatus: move.b   #1,TFRNCR+4(a5) transfer count is 1
           0001013d
00607           Nstatus1
00608 * bsr TestIrq Is there an interrupt
00609 06e2 1b7c            move.b   #$14,COMNCR(a5) transfer info cmd
           00140123
00610 06e8 6100            bsr      TestIrq        Is there an interrupt
           01b0
00611 06ec 0800            btst     #7,d0          test if data full ncr
```

```
          0007
00612 06f0 67f0           beq.s    Nstatus1       No data received
00613 06f2 102d           move.b   DATNCR(a5),d0
          0121
00614 06f6 1680           move.b   d0,(a3)
00615 06f8 4e75           rts
00616
00617 *
00618 * NCRmsg : get message byte
00619 * entry : a3  message ptr
00620 *         a5  base address
00621 *         a6  static storage
00622 *
00623 06fa 1b7c NCRmsg:   move.b   #1,TFRNCR+4(a5) transfer count is 1
          0001013d
00624          NCRmsg1
00625 0700 102d           move.b   COMNCR(a5),d0  test if command register is 00
          0123
00626 0704 6706           beq.s    NCRmsg2
00627 0706 1b7c           move.b   #$04,COMNCR(a5) message accepted cmd
          00040123
00628          NCRmsg2
00629 * bsr TestIrq Is there an interrupt
00630 070c 1b7c           move.b   #$14,COMNCR(a5) transfer info cmd
          00140123
00631 0712 6100           bsr      TestIrq        Is there an interrupt
          0186
00632 0716 0800           btst     #7,d0          test if data full ncr
          0007
00633 071a 67e4           beq.s    NCRmsg1        No data received, try to give command aga
in
00634 071c 102d           move.b   DATNCR(a5),d0
          0121
00635 0720 1680           move.b   d0,(a3)
00636 0722 1b7c           move.b   #$04,COMNCR(a5) message accepted cmd
          00040123
00637          NCRmsg3
00638 0728 6100           bsr      WaitIrq        SCSI should be disconnected
          017a
00639 072c 0c01           cmpi.b   #4,d1
          0004
00640 0730 66f6           bne.s    NCRmsg3        Irq not from disconnect
00641 0732 4e75           rts
00642
00643 *
00644 * NCRdatrd : get data bytes
00645 * entry : a2 data pointer
00646 *         a5 base address
00647 *         a6 static storage
00648 *         d2.w count
00649 *
00650 0734 3602 NCRdatrd: move.w   d2,d3          save for later
00651 0736 1b42           move.b   d2,TFRNCR+4(a5) least sign. byte transfer count
          013d
00652 073a e04a           lsr.w    #8,d2          most sign. byte
00653 073c 1b42           move.b   d2,TFRNCR+2(a5) store it
          013b
00654 0740 3b7c           move.w   #$00,EscFlg(a5) clear escflag
          0000004a
00655 0746 1b7c           move.b   #(StealHld+DevAck+Dev8Bit+StatInpI),DCR+CHNL0(a5)
          00e10004
00656 074c 1b7c           move.b   #(DevToMem+ByteSize+ChainDis+ReqInit),OCR+CHNL0(a5)
```

```
          00820005
00657 0752 1b7c           move.b   #(MemCntUp+DevNoCnt),SCR+CHNL0(a5)
          00040006
00658 0758 1b7c           move.b   #$80,NIV+CHNL0(a5) User vector $80
          00800025
00659 075e 1b7c           move.b   #$80,EIV+CHNL0(a5) User vector $81
          00800027
00660 0764 1b7c           move.b   #UserData,DFC+CHNL0(a5)
          00010031
00661 076a 1b7c           move.b   #UserData,MFC+CHNL0(a5)
          00010029
00662 0770 1b7c           move.b   #ChPrior0,CPR+CHNL0(a5)
          0000002d
00663 0776 1b7c           move.b   #$ff,CSR+CHNL0(a5) clear all bits
          00ff0000
00664 077c 2b4a           move.l   a2,MAR+CHNL0(a5)
          000c
00665 0780 2b7c           move.l   #(CC74_BASE+DATNCR),DAR+CHNL0(a5)
          00ffff21
          0014
00666 0788 3b43           move.w   d3,MTC+CHNL0(a5)
          000a
00667 078c 1b7c           move.b   #(StartOp+IntrptEn),CHCR+CHNL0(a5) don't start, enable i
rq
          00880007
00668 0792 6100           bsr      TestIrq         read irq
          0106
00669 0796 1b7c           move.b   #$94,COMNCR(a5) give command, irq occurs
          00940123
00670 079c 082d datrd1    btst     #OpComp_B,CSR+CHNL0(a5) operation completed
          00070000
00671 07a2 6606           bne.s    datrd2          yes, branch
00672 07a4 302d           move.w   EscFlg(a5),d0  set in Irq routine if no data phase occur
ed
          004a
00673 07a8 67f2           beq.s    datrd1          no error
00674 07aa 4e75 datrd2    rts
00675
00676 *
00677 * NCRdatwr : send data bytes
00678 * entry : a2 data pointer
00679 *         a5 base address
00680 *         a6 static storage
00681 *         d2.w count
00682 *
00683 07ac 3602 NCRdatwr:  move.w   d2,d3           save for later
00684 07ae 1b42           move.b   d2,TFRNCR+4(a5) least sign. byte transfer count
          013d
00685 07b2 e04a           lsr.w    #8,d2           most sign. byte
00686 07b4 1b42           move.b   d2,TFRNCR+2(a5) store it
          013b
00687 07b8 3b7c           move.w   #$00,EscFlg(a5) clear escflag
          0000004a
00688 07be 1b7c           move.b   #(StealHld+DevAck+Dev8Bit+StatInpl),DCR+CHNL0(a5)
          00e10004
00689 07c4 1b7c           move.b   #(MemToDev+ByteSize+ChainDis+ReqInit),OCR+CHNL0(a5)
          00020005
00690 07ca 1b7c           move.b   #(MemCntUp+DevNoCnt),SCR+CHNL0(a5)
          00040006
00691 07d0 1b7c           move.b   #$80,NIV+CHNL0(a5) User vector $80
          00800025
00692 07d6 1b7c           move.b   #$80,EIV+CHNL0(a5) User vector $81
          00800027
```

```
00693 07dc 1b7c            move.b   #UserData,DFC+CHNL0(a5)
           00010031
00694 07e2 1b7c            move.b   #UserData,MFC+CHNL0(a5)
           00010029
00695 07e8 1b7c            move.b   #ChPrior0,CPR+CHNL0(a5)
           0000002d
00696 07ee 1b7c            move.b   #$ff,CSR+CHNL0(a5) clear all bits
           00ff0000
00697 07f4 2b4a            move.l   a2,MAR+CHNL0(a5)
           000c
00698 07f8 2b7c            move.l   #(CC74_BASE+DATNCR),DAR+CHNL0(a5)
           00ffff21
           0014
00699 0800 3b43            move.w   d3,MTC+CHNL0(a5)
           000a
00700 0804 1b7c            move.b   #(StartOp+IntrptEn),CHCR+CHNL0(a5) don't start, enable i
rq
           00880007
00701 080a 6100            bsr      TestIrq        read irq
           008e
00702 080e 1b7c            move.b   #$94,COMNCR(a5) give command, irq occurs
           00940123
00703 0814 082d datwr1     btst     #OpComp_B,CSR+CHNL0(a5) operation completed
           00070000
00704 081a 6606            bne.s    datwr2         yes, branch
00705 081c 302d            move.w   EscFlg(a5),d0  set in Irq routine if no data phase occur
ed
           004a
00706 0820 67f2            beq.s    datwr1         no error
00707 0822 4e75 datwr2     rts
00708
00709
00710 *
00711 * Sense Status command
00712 *       Should be called if another command returns with an error
00713 * Entry:  a5  base address
00714 *         a6  static storage
00715 *
00716 0824 1d7c senstat:   move.b   #$03,HD_cmd(a6) Set command byte
           0003000a
00717 082a 022e            andi.b   #$1f,HD_psn(a6) clear drive bit
           001f000b
00718 0830 43ee            lea      HD_cmd(a6),a1
           000a
00719 0834 323c            move.w   #6,d1
           0006
00720 0838 6100            bsr      NCRcmd         Execute command
           fe08
00721 083c 655a            bcs.s    sens2
00722 083e 343c            move.w   #4,d2
           0004
00723 0842 45ee            lea      HD_sense(a6),a2
           0112
00724 0846 6100            bsr      NCRdatrd
           feec
00725 084a 47ee            lea      HD_status(a6),a3
           0010
00726 084e 6100            bsr      NCRstatus
           fe8c
00727 0852 47ee            lea      HD_msg(a6),a3
           0011
00728 0856 6100            bsr      NCRmsg
           fea2
```

```
00729 085a 082e            btst     #1,HD_status(a6)
           00010010
00730 0860 6624            bne.s    sens1
00731 0862 41fa            lea      emsg0(pc),a0   start of "Error code"
           057c
00732 0866 303c            move.w   #StdOut,d0
           0001
00733 086a 223c            move.l   #szemsg0,d1
           00000021
00734 0870=4e40            os9      I$Write
           0000
00735 0874 222e            move.l   HD_sense(a6),d1
           0112
00736 0878 303c            move.w   #StdOut,d0
           0001
00737 087c 41ee            lea      Char_Buf(a6),a0
           0000
00738 0880 6100            bsr      Outhex
           01ea
00739 0884 6012            bra.s    sens2
00740 0886 41fa sens1      lea      emsg11(pc),a0
           06c2
00741 088a 303c            move.w   #StdOut,d0
           0001
00742 088e 223c            move.l   #szemsg11,d1
           0000001e
00743 0894=4e40            os9      I$Write
           0000
00744 0898 4e75 sens2      rts
00745
00746 *
00747 * TestIrq
00748 *
00749 089a 102d TestIrq:   move.b   AUXNCR(a5),d0  get auxiliary register NCR
           0129
00750 089e 122d            move.b   IRQNCR(a5),d1  get interrup register NCR
           012d
00751 08a2 4e75            rts
00752
00753 *
00754 * WaitIrq
00755 *
00756 * Poll IRQ line NCR chip via DMA controller
00757 *
00758 08a4 102d WaitIrq:   move.b   CSR(a5),d0     get status register dma channel 0
           0000
00759 08a8 0800            btst     #0,d0          irq NCR occurred
           0000
00760 08ac 66f6            bne.s    WaitIrq        no, try again
00761 08ae 102d            move.b   AUXNCR(a5),d0  get auxiliary register NCR
           0129
00762 08b2 122d            move.b   IRQNCR(a5),d1  get interrup register NCR
           012d
00763 08b6 4e75            rts
00764 *
00765 * AskCon
00766 *    Ask for the fill constant
00767 * Input : none
00768 * Exit  : d0.1 fill constant
00769 *        Carry clr if no error
00770 08b8 41fa AskCon:    lea      msg6(pc),a0
```

```
           04ba
00771 08bc 303c            move.w   #StdOut,d0
           0001
00772 08c0 223c            move.l   #szmsg6,d1
           00000027
00773 08c6=4e40            os9      I$Write        Print 'give con'
           0000
00774 08ca 303c            move.w   #StdIn,d0
           0000
00775 08ce 7209            moveq    #9,d1
00776 08d0 41ee            lea      Char_Buf(a6),a0
           0000
00777 08d4 6100            bsr      Inhex
           0106
00778 08d8 6400            bcc      askcon1
           001a
00779 08dc 41fa            lea      emsg13(pc),a0
           06a2
00780 08e0 303c            move.w   #StdOut,d0
           0001
00781 08e4 223c            move.l   #szemsg13,d1
           00000022
00782 08ea=4e40            os9      I$Write        Print error message
           0000
00783 08ee 003c            ori.b    #1,ccr
           0001
00784 08f2 4e75            rts
00785 08f4 023c askcon1    andi.b   #$fe,ccr
           00fe
00786 08f8 4e75            rts
00787 *
00788 * Askid
00789 *
00790 08fa 41fa Askid:     lea      msg8(pc),a0
           04ad
00791 08fe 303c            move.w   #StdOut,d0
           0001
00792 0902 223c            move.l   #szmsg8,d1
           0000001a
00793 0908=4e40            os9      I$Write        Print 'give target id'
           0000
00794 090c 303c            move.w   #StdIn,d0
           0000
00795 0910 7202            moveq    #2,d1
00796 0912 41ee            lea      Char_Buf(a6),a0
           0000
00797 0916=4e40            os9      I$ReadLn       get idnr
           0000
00798 091a 6516            bcs.s    Askid3
00799 091c 102e            move.b   Char_Buf(a6),d0
           0000
00800 0920 0400            subi.b   #$30,d0
           0030
00801 0924 650c            bcs.s    Askid3         negative result <0
00802 0926 0c00            cmpi.b   #$07,d0
           0007
00803 092a 6206            bhi.s    Askid3
00804 092c 1b40            move.b   d0,DESIDNCR(a5)
           0127
00805 0930 4e75            rts
00806 0932 41fa Askid3     lea      emsg16(pc),a0
```

```
          06ad
00807 0936 303c            move.w   #StdOut,d0
          0001
00808 093a 223c            move.l   #szemsg16,d1
          0000001e
00809 0940=4e40            os9      I$Write
          0000
00810 0944 4e75            rts
00811 *
00812 * Ask Drive no
00813 *
00814 0946 41fa Askdn:     lea      msg9(pc),a0
          047b
00815 094a 303c            move.w   #StdOut,d0
          0001
00816 094e 223c            move.l   #szmsg9,d1
          0000001d
00817 0954=4e40            os9      I$Write        Print 'give drive number: '
          0000
00818 0958 303c            move.w   #StdIn,d0
          0000
00819 095c 7202            moveq    #2,d1
00820 095e 41ee            lea      Char_Buf(a6),a0
          0000
00821 0962=4e40            os9      I$ReadLn       get drivenr
          0000
00822 0966 6518            bcs.s    Askdn3
00823 0968 102e            move.b   Char_Buf(a6),d0
          0000
00824 096c 0400            subi.b   #$30,d0
          0030
00825 0970 650e            bcs.s    Askdn3         negative result <0
00826 0972 0c00            cmpi.b   #$01,d0
          0001
00827 0976 6208            bhi.s    Askdn3
00828 0978 eb08            lsl.b    #5,d0
00829 097a 1d40            move.b   d0,Drive_no(a6)
          0116
00830 097e 4e75            rts
00831 0980 41fa Askdn3     lea      emsg17(pc),a0
          067d
00832 0984 303c            move.w   #StdOut,d0
          0001
00833 0988 223c            move.l   #szemsg17,d1
          00000021
00834 098e=4e40            os9      I$Write
          0000
00835 0992 4e75            rts
00836 *
00837 * AskPsn
00838 *    Ask for the Physical Sector number
00839 * Input : none
00840 * Exit  : d0.l 000xxxxx.xxxxxxxx.xxxxxxxx.xxxxxxxx PSN
00841 *         Carry clr if no error
00842 0994 41fa AskPsn:    lea      msg5(pc),a0
          03bd
00843 0998 303c            move.w   #StdOut,d0
          0001
00844 099c 223c            move.l   #szmsg5,d1
          00000021
00845 09a2=4e40            os9      I$Write        Print 'give psn'
```

```
           0000
00846 09a6 303c            move.w   #StdIn,d0
           0000
00847 09aa 7209            moveq    #9,d1
00848 09ac 41ee            lea      Char_Buf(a6),a0
           0000
00849 09b0 6100            bsr      Inhex
           002a
00850 09b4 6400            bcc      askpsnl
           001a
00851 09b8 41fa            lea      emsg12(pc),a0
           05ae
00852 09bc 303c            move.w   #StdOut,d0
           0001
00853 09c0 223c            move.l   #szemsg12,d1
           00000018
00854 09c6=4e40            os9      I$Write        Print error message
           0000
00855 09ca 003c            ori.b    #1,ccr
           0001
00856 09ce 4e75            rts
00857 09d0 0280 askpsnl    andi.l   #$1fffff,d0    clear drive nr
           001fffff
00858 09d6 023c            andi.b   #$fe,ccr
           00fe
00859 09da 4e75            rts
00860 *
00861 * Following are the special hex input and output routines not found in OS9
00862 *
00863
00864 *
00865 * Inhex:
00866 *
00867 * input : d0.w path number
00868 *         d1.l max number of bytes to read inclusief <cr>
00869 *         (a0) start address of input buffer
00870 * exit  : d0.l hex long word padded with zero's
00871 *         carry set on error
00872 *
00873 09dc 48e7 Inhex:     movem.l  a1,-(sp)       save used registers
           0040
00874 09e0=4e40            os9      I$ReadLn       get ascii string
           0000
00875 09e4 6402            bcc.s    Inhex1         see if error
00876 09e6 603c            bra.s    Inhex5         yes, leave routine with carry set
00877 09e8 2248 Inhex1     move.l   a0,a1          a0 ptr to start buffer
00878 09ea 5381            subi.l   #1,d1          discard <cr>
00879 09ec d3c1            add.l    d1,a1          a1 ptr to end buffer
00880 09ee 4281            clr.l    d1             set hex output to zero
00881 09f0 0c10 Inhex2     cmpi.b   #space,(a0)    skip leading zero's
           0020
00882 09f4 6608            bne.s    Inhex3         was it space
00883 09f6 5288            addq.l   #1,a0          bump input pointer
00884 09f8 b3c8            cmpa.l   a0,a1          last?
00885 09fa 671c            beq.s    Inhex4         yes, leave
00886 09fc 60f2            bra.s    Inhex2         get next
00887 09fe 1010 Inhex3     move.b   (a0),d0        we got some character
00888 0a00 6100            bsr      Aschex         convert ascii to hex
           002e
00889 0a04 651e            bcs.s    Inhex5         error occured if set
00890 0a06 e989            lsl.l    #4,d1          shift d1 for next nibble
```

```
00891 0a08 c0bc             and.l    #$0f,d0         mask to be sure
           0000000f
00892 0a0e d280             add.l    d0,d1           set lowest nibble
00893 0a10 5288             addq.l   #1,a0           bump pointer
00894 0a12 b3c8             cmpa.l   a0,a1           last?
00895 0a14 6702             beq.s    Inhex4          yes, leave
00896 0a16 60e6             bra.s    Inhex3          get next
00897 0a18 2001 Inhex4      move.l   d1,d0
00898 0a1a 4cdf             movem.l  (sp)+,a1        leave with no error
           0200
00899 0a1e 023c             andi.b   #$fe,ccr
           00fe
00900 0a22 4e75             rts
00901 0a24 2001 Inhex5      move.l   d1,d0
00902 0a26 4cdf             movem.l  (sp)+,a1        leave with error
           0200
00903 0a2a 003c             ori.b    #1,ccr
           0001
00904 0a2e 4e75             rts
00905 *
00906 * Aschex:
00907 *
00908 * input : d0 ascii character
00909 * exit  : d0 converted to hex if no error
00910 *         carry set if error
00911 *
00912 0a30 2f00 Aschex:     move.l   d0,-(sp)        save for a while
00913 0a32 b03c             cmp.b    #'a',d0         check for lower case
           0061
00914 0a36 6d0a             blt.s    Aschex1
00915 0a38 b03c             cmp.b    #'f',d0
           0066
00916 0a3c 6204             bhi.s    Aschex1
00917 0a3e 0200             andi.b   #$df,d0         was lower case, make upper case
           00df
00918 0a42 0400 Aschex1     subi.b   #$30,d0         first step to hex value
           0030
00919 0a46 651c             bcs.s    Aschex3         negative result, is not numerical hex
00920 0a48 0c00             cmpi.b   #$09,d0
           0009
00921 0a4c 630e             bls.s    Aschex2         test if in range 0-9
00922 0a4e 0c00             cmpi.b   #$11,d0
           0011
00923 0a52 6510             bcs.s    Aschex3         test if >9 en <A
00924 0a54 0c00             cmpi.b   #$16,d0
           0016
00925 0a58 620a             bhi.s    Aschex3         test if in range A-F
00926 0a5a 5f00             subi.b   #$7,d0
00927 0a5c 588f Aschex2     addq.l   #4,sp           discard saved d0 on stack
00928 0a5e 023c             andi.b   #$fe,ccr        no error, clear carry
           00fe
00929 0a62 4e75             rts
00930 0a64 201f Aschex3     move.l   (sp)+,d0        get original character
00931 0a66 003c             ori.b    #1,ccr          error, not hex, set carry
           0001
00932 0a6a 4e75             rts
00933 *
00934 * Outhex
00935 *
00936 * input: d0.w path number
00937 *       (a0) start address of buffer
```

```
00938 *       d1   long word to output
00939 * exit : d1   number of bytes written
00940 *       carry set if error
00941 *
00942 0a6c 48e7 Outhex:    movem.l  d2/d3,-(sp)    save used registers
          3000
00943 0a70 5e88            addq.l   #7,a0          set to end of buffer to write
00944 0a72 243c            move.l   #7,d2          set loop count is eigth
          00000007
00945 0a78 2601 Outhex1    move.l   d1,d3          use d3 as scratch
00946 0a7a e889            lsr.l    #4,d1          get nibble for next loop in position
00947 0a7c 0203            andi.b   #$f,d3         mask lowes nibble
          000f
00948 0a80 d63c            add.b    #$30,d3        get to range 0-9
          0030
00949 0a84 b63c            cmp.b    #$39,d3        is it higher
          0039
00950 0a88 6304            bls.s    Outhex2        no
00951 0a8a d63c            add.b    #7,d3          get to range A-F
          0007
00952 0a8e 1103 Outhex2    move.b   d3,-(a0)       store byte in write buffer
00953 0a90 51ca            dbra     d2,Outhex1     get next until all eigth done
          ffe6
00954 0a94 223c            move.l   #$8,d1         number of bytes
          00000008
00955 0a9a=4e40            os9      I$Write        a0 set to start
          0000
00956 0a9e 4cdf            movem.l  (sp)+,d2/d3
          000c
00957 0aa2 4e75            rts
00958 *
00959 * Irq routine
00960 * This routine becomes enabled in the NCRdatrd and NCRdatwr
00961 * It checks the NCR chip for the 'Bus Phase' and initiate
00962 * the 'Transfer Info command' until the bus phase is the
00963 * data phase. Then the DMA is enabled and we wait for the
00964 * 'status phase'
00965 *
00966 0aa4 48e7 IrqEntry:  movem.l  d0-d4/a0-a5,-(sp) save registers
          f8fc
00967 0aa8 2a7c            movea.l  #CC74_BASE,a5
          00fffe00
00968 0aae 082d            btst     #3,CONTROL74(a5)
          00030101
00969 0ab4 6710            beq.s    irqen0
00970 0ab6 102d            move.b   CHNL0+CSR(a5),d0
          0000
00971 0aba 0800            btst     #OpComp_B,d0
          0007
00972 0abe 6676            bne.s    irqen3
00973 0ac0 0800            btst     #PCLSts_B,d0
          0000
00974 0ac4 6710            beq.s    irqen1         Irq from dma pcl line occured
00975           irqen0
00976 0ac6 4cdf            movem.l  (sp)+,d0-d4/a0-a5
          3f1f
00977 0aca 4879            pea.l    $200           do the same as os9 does for unexpected ir
q's
          00000200
00978 0ad0 4ef9            jmp      $83230
          00083230
00979           irqen1
```

```
00980 0ad6 08ad             bclr      #3,CONTROL74(a5) toggle ENIRQ
           00030101
00981 0adc 1b7c             move.b    #$ff,CHNL0+CSR(a5) clear interrupt cause
           00ff0000
00982 0ae2 102d             move.b    AUXNCR(a5),d0
           0129
00983 0ae6 122d             move.b    IRQNCR(a5),d1
           012d
00984 0aea 08ed             bset      #3,CONTROL74(a5)
           00030101
00985 0af0 1600             move.b    d0,d3
00986 0af2 e18b             lsl.l     #8,d3
00987 0af4 d600             add.b     d0,d3
00988 0af6 3b43             move.w    d3,SavSts(a5)
           004c
00989 0afa 0200             andi.b    #$38,d0         mask off msg, c/d and i/o line
           0038
00990 0afe b03c             cmp.b     #$00,d0         test Data Out phase
           0000
00991 0b02 6718             beq.s     irqen2
00992 0b04 b03c             cmp.b     #$08,d0         test Data In phase
           0008
00993 0b08 6712             beq.s     irqen2
00994 0b0a b03c             cmp.b     #$18,d0         test Status phase
           0018
00995 0b0e 6726             beq.s     irqen3
00996 0b10 1b7c             move.b    #$94,COMNCR(a5) give transfer info command, dma
           00940123
00997 0b16 4cdf             movem.l   (sp)+,d0-d4/a0-a5
           3f1f
00998 0b1a 4e73             rte
00999 0b1c 1b7c irqen2      move.b    #$94,COMNCR(a5) give transfer info command, dma
           00940123
01000 0b22 082d             btst      #Active_B,CHNL0+CSR(a5) test if channel already started
           00030000
01001 0b28 6606             bne.s     irqen4          yes, started
01002 0b2a 08ed             bset      #Start_B,CHNL0+CHCR(a5) enable dma controller
           00070007
01003 0b30 4cdf irqen4      movem.l   (sp)+,d0-d4/a0-a5
           3f1f
01004 0b34 4e73             rte
01005 0b36 1b7c irqen3      move.b    #(StealHld+DevAck+Dev8Bit+StatInp),CHNL0+DCR(a5)
           00e00004
01006 0b3c 08ad             bclr      #Intrpt_B,CHNL0+CHCR(a5)
           00030007
01007 0b42 1b7c             move.b    #$ff,EscFlg(a5)
           00ff004a
01008 0b48 4cdf             movem.l   (sp)+,d0-d4/a0-a5
           3f1f
01009 0b4c 4e73             rte
01010 *
01011 * parameter table
01012 *
01013                HD_par85
01014 0b4e   01             dc.b      $01
01015 0b4f   b8             dc.b      $B8
01016 0b50   06             dc.b      $06
01017 0b51   00             dc.b      $00
01018 0b52   10             dc.b      $10
01019 0b53   00             dc.b      $00
01020 0b54   80             dc.b      $80
```

```
01021 0b55   0b              dc.b     $0B
01022
01023            HD_par88
01024 0b56   02              dc.b     $02
01025 0b57   64              dc.b     $64
01026 0b58   04              dc.b     $04
01027 0b59   02              dc.b     $02
01028 0b5a   64              dc.b     $64
01029 0b5b   00              dc.b     $00
01030 0b5c   80              dc.b     $80
01031 0b5d   0b              dc.b     $0B
01032
01033
01034 *
01035 * messages
01036 *
01037 0b5e 0d0a msg1        dc.b     CR,LF
01038 0b60 0d0a             dc.b     CR,LF
01039 0b62 5465             dc.b     "Testing CC-74 module with NCR5385 and XEBEC S1410A"
           7374696e
           67204343
           2d373420
           6d6f6475
           6c652077
           69746820
           4e435235
           33383520
           616e6420
           58454245
           43205331
           34313041
01040 0b94 0d0a             dc.b     CR,LF
01041 0b96 0d0a             dc.b     CR,LF
01042 0b98 2020             dc.b     "     1 : initialize controller and drive"
           20202031
           203a2069
           6e697469
           616c697a
           6520636f
           6e74726f
           6c6c6572
           20616e64
           20647269
           7665
01043 0bc0 0d0a             dc.b     CR,LF
01044 0bc2 2020             dc.b     "     2 : test drive ready"
           20202032
           203a2074
           65737420
           64726976
           65207265
           616479
01045 0bdb 0d0a             dc.b     CR,LF
01046 0bdd 2020             dc.b     "     3 : format drive"
           20202033
           203a2066
           6f726d61
           74206472
           697665
01047 0bf2 0d0a             dc.b     CR,LF
01048 0bf4 2020             dc.b     "     4 : drive diagnostic"
```

```
           20202034
           203a2064
           72697665
           20646961
           676e6f73
           746963
01049 0c0d 0d0a                dc.b     CR,LF
01050 0c0f 2020                dc.b     "     5 : read one sector"
           20202035
           203a2072
           65616420
           6f6e6520
           73656374
           6f72
01051 0c27 0d0a                dc.b     CR,LF
01052 0c29 2020                dc.b     "     6 : write one sector"
           20202036
           203a2077
           72697465
           206f6e65
           20736563
           746f72
01053 0c42 0d0a                dc.b     CR,LF
01054 0c44 2020                dc.b     "     7 : seek to sector"
           20202037
           203a2073
           65656b20
           746f2073
           6563746f
           72
01055 0c5b 0d0a                dc.b     CR,LF
01056 0c5d 2020                dc.b     "     8 : recalibrate"
           20202038
           203a2072
           6563616c
           69627261
           7465
01057 0c71 0d0a                dc.b     CR,LF
01058 0c73 2020                dc.b     "     9 : exit"
           20202039
           203a2065
           786974
01059 0c80 0d0a                dc.b     CR,LF
01060 0c82 2020                dc.b     "     a : show buffer"
           20202061
           203a2073
           686f7720
           62756666
           6572
01061 0c96 0d0a                dc.b     CR,LF
01062 0c98 2020                dc.b     "     b : fill buffer with constant"
           20202062
           203a2066
           696c6c20
           62756666
           65722077
           69746820
           636f6e73
           74616e74
01063 0cba 0d0a                dc.b     CR,LF
01064 0cbc 2020                dc.b     "     c : change target id"
```

```
           20202063
           203a2063
           68616e67
           65207461
           72676574
           206964
01065 0cd5 0d0a           dc.b    CR,LF
01066 0cd7 2020           dc.b    "     d : change drive number"
           20202064
           203a2063
           68616e67
           65206472
           69766520
           6e756d62
           6572
01067 0cf3 0d0a           dc.b    CR,LF
01068 0cf5 0d0a           dc.b    CR,LF
01069 0cf7 2020           dc.b    "     test number ? "
           20202074
           65737420
           6e756d62
           6572203f
           20
01070  000001ac szmsg1    equ     *-msg1
01071
01072 0d0a 0d0a msg2      dc.b    CR,LF
01073 0d0c 2a2a           dc.b    "** CC74 init error **"
           20434337
           3420696e
           69742065
           72726f72
           202a2a
01074 0d21 0d0a           dc.b    CR,LF
01075  00000019 szmsg2    equ     *-msg2
01076
01077 0d23 0d0a msg3      dc.b    CR,LF
01078 0d25 5375           dc.b    "Successfull completion of this test"
           63636573
           7366756c
           6c20636f
           6d706c65
           74696f6e
           206f6620
           74686973
           20746573
           74
01079 0d48 0d0a           dc.b    CR,LF
01080  00000027 szmsg3    equ     *-msg3
01081
01082 0d4a 0d0a msg4      dc.b    CR,LF
01083 0d4c 5768           dc.b    "What?"
           61743f
01084 0d51 0d0a           dc.b    CR,LF
01085  00000009 szmsg4    equ     *-msg4
01086
01087 0d53 0d0a msg5      dc.b    CR,LF
01088 0d55 0d0a           dc.b    CR,LF
01089 0d57 4769           dc.b    "Give Physical Sector Number: "
           76652050
           68797369
           63616c20
           53656374
           6f72204e
           756d6265
           723a20
01090  00000021 szmsg5     equ      *-msg5
01091
01092 0d74 0d0a msg6       dc.b     CR,LF
01093 0d76 0d0a            dc.b     CR,LF
01094 0d78 4769            dc.b     "Give constant to fill buffer with: "
           76652063
           6f6e7374
           616e7420
           746f2066
           696c6c20
           62756666
           65722077
           6974683a
           20
01095  00000027 szmsg6     equ      *-msg6
01096
01097 0d9b 0d0a msg7       dc.b     CR,LF
01098 0d9d 0d0a            dc.b     CR,LF
01099 0d9f 436f            dc.b     "Continue: "
           6e74696e
           75653a20
01100  0000000e szmsg7     equ      *-msg7
01101
01102 0da9 0d0a msg8       dc.b     CR,LF
01103 0dab 0d0a            dc.b     CR,LF
01104 0dad 4769            dc.b     "Give target id (0-7): "
           76652074
           61726765
           74206964
           2028302d
           37293a20
01105  0000001a szmsg8     equ      *-msg8
01106
01107 0dc3 0d0a msg9       dc.b     CR,LF
01108 0dc5 0d0a            dc.b     CR,LF
01109 0dc7 4769            dc.b     "Give drive number (0-1): "
           76652064
           72697665
           206e756d
           62657220
           28302d31
           293a20
01110  0000001d szmsg9     equ      *-msg9
01111 *
01112 * error messages
01113 *
01114 0de0 0d0a emsg0      dc.b     CR,LF
01115 0de2 0d0a            dc.b     CR,LF
01116 0de4 2a2a            dc.b     "** Sense Status error code : "
           2053656e
           73652053
           74617475
           73206572
           726f7220
           636f6465
           203a20
01117  00000021 szemsg0    equ      *-emsg0
```

```
01118
01119 0e01 0d0a emsgl       dc.b     CR,LF
01120 0e03 0d0a             dc.b     CR,LF
01121 0e05 2a2a             dc.b     "** Internal diagnostic error **"
           20496e74
           65726e61
           6c206469
           61676e6f
           73746963
           20657272
           6f72202a
           2a
01122 0e24 0d0a             dc.b     CR,LF
01123  00000025 szemsgl     equ      *-emsgl
01124
01125 0e26 0d0a emsg2       dc.b     CR,LF
01126 0e28 0d0a             dc.b     CR,LF
01127 0e2a 2a2a             dc.b     "** Ram test error **"
           2052616d
           20746573
           74206572
           726f7220
           2a2a
01128 0e3e 0d0a             dc.b     CR,LF
01129  0000001a szemsg2     equ      *-emsg2
01130
01131 0e40 0d0a emsg3       dc.b     CR,LF
01132 0e42 0d0a             dc.b     CR,LF
01133 0e44 2a2a             dc.b     "** Init drive parameters error **"
           20496e69
           74206472
           69766520
           70617261
           6d657465
           72732065
           72726f72
           202a2a
01134 0e65 0d0a             dc.b     CR,LF
01135  00000027 szemsg3     equ      *-emsg3
01136
01137 0e67 0d0a emsg4       dc.b     CR,LF
01138 0e69 0d0a             dc.b     CR,LF
01139 0e6b 2a2a             dc.b     "** Test Drive Ready error **"
           20546573
           74204472
           69766520
           52656164
           79206572
           726f7220
           2a2a
01140 0e87 0d0a             dc.b     CR,LF
01141  00000022 szemsg4     equ      *-emsg4
01142
01143 0e89 0d0a emsg5       dc.b     CR,LF
01144 0e8b 0d0a             dc.b     CR,LF
01145 0e8d 2a2a             dc.b     "** Format Drive error  **"
           20466f72
           6d617420
           44726976
           65206572
           726f7220
           202a2a
01146 0ea6 0d0a             dc.b     CR,LF
01147  0000001f szemsg5     equ      *-emsg5
01148
01149 0ea8 0d0a emsg6       dc.b     CR,LF
01150 0eaa 0d0a             dc.b     CR,LF
01151 0eac 2a2a             dc.b     "** Drive Diagnostic error **"
           20447269
           76652044
           6961676e
           6f737469
           63206572
           726f7220
           2a2a
01152 0ec8 0d0a             dc.b     CR,LF
01153  00000022 szemsg6     equ      *-emsg6
01154
01155 0eca 0d0a emsg7       dc.b     CR,LF
01156 0ecc 0d0a             dc.b     CR,LF
01157 0ece 2a2a             dc.b     "** Read One Sector error **"
           20526561
           64204f6e
           65205365
           63746f72
           20657272
           6f72202a
           2a
01158 0ee9 0d0a             dc.b     CR,LF
01159  00000021 szemsg7     equ      *-emsg7
01160
01161 0eeb 0d0a emsg8       dc.b     CR,LF
01162 0eed 0d0a             dc.b     CR,LF
01163 0eef 2a2a             dc.b     "** Write One Sector error **"
           20577269
           7465204f
           6e652053
           6563746f
           72206572
           726f7220
           2a2a
01164 0f0b 0d0a             dc.b     CR,LF
01165  00000022 szemsg8     equ      *-emsg8
01166
01167 0f0d 0d0a emsg9       dc.b     CR,LF
01168 0f0f 0d0a             dc.b     CR,LF
01169 0f11 2a2a             dc.b     "** Seek to Sector error **"
           20536565
           6b20746f
           20536563
           746f7220
           6572726f
           72202a2a
01170 0f2b 0d0a             dc.b     CR,LF
01171  00000020 szemsg9     equ      *-emsg9
01172
01173 0f2d 0d0a emsg10      dc.b     CR,LF
01174 0f2f 0d0a             dc.b     CR,LF
01175 0f31 2a2a             dc.b     "** Recalibrate error **"
           20526563
           616c6962
           72617465
```

```
                20657272
                6f72202a
                2a
01176 0f48 0d0a              dc.b     CR,LF
01177  0000001d szemsg10     equ      *-emsg10
01178
01179 0f4a 0d0a emsg11       dc.b     CR,LF
01180 0f4c 0d0a              dc.b     CR,LF
01181 0f4e 2a2a              dc.b     "** Sense Status Error **"
                2053656e
                73652053
                74617475
                73204572
                726f7220
                2a2a
01182 0f66 0d0a              dc.b     CR,LF
01183  0000001e szemsg11     equ      *-emsg11
01184
01185 0f68 0d0a emsg12       dc.b     CR,LF
01186 0f6a 0d0a              dc.b     CR,LF
01187 0f6c 2a2a              dc.b     "** Error in PSN **"
                20457272
                6f722069
                6e205053
                4e202a2a
01188 0f7e 0d0a              dc.b     CR,LF
01189  00000018 szemsg12     equ      *-emsg12
01190
01191 0f80 0d0a emsg13       dc.b     CR,LF
01192 0f82 0d0a              dc.b     CR,LF
01193 0f84 2a2a              dc.b     "** Error in fill constant **"
                20457272
                6f722069
                6e206669
                6c6c2063
                6f6e7374
                616e7420
                2a2a
01194 0fa0 0d0a              dc.b     CR,LF
01195  00000022 szemsg13     equ      *-emsg13
01196
01197 0fa2 0d0a emsg14       dc.b     CR,LF
01198 0fa4 0d0a              dc.b     CR,LF
01199 0fa6 2a2a              dc.b     "** Time out error **"
                2054696d
                65206f75
                74206572
                726f7220
                2a2a
01200 0fba 0d0a              dc.b     CR,LF
01201  0000001a szemsg14     equ      *-emsg14
01202
01203 0fbc 0d0a emsg15       dc.b     CR,LF
01204 0fbe 0d0a              dc.b     CR,LF
01205 0fc0 2a2a              dc.b     "** Illegal interrupt occured **"
                20496c6c
                6567616c
                20696e74
                65727275
                7074206f
                63637572
```

```
           6564202a
           2a
01206 0fdf 0d0a             dc.b     CR,LF
01207  00000025 szemsg15    equ      *-emsg15
01208
01209 0fe1 0d0a emsg16      dc.b     CR,LF
01210 0fe3 0d0a             dc.b     CR,LF
01211 0fe5 2a2a             dc.b     "** Error in target id **"
           20457272
           6f722069
           6e207461
           72676574
           20696420
           2a2a
01212 0ffd 0d0a             dc.b     CR,LF
01213  0000001e szemsg16    equ      *-emsg16
01214
01215 0fff 0d0a emsg17      dc.b     CR,LF
01216 1001 0d0a             dc.b     CR,LF
01217 1003 2a2a             dc.b     "** Error in drive number **"
           20457272
           6f722069
           6e206472
           69766520
           6e756d62
           6572202a
           2a
01218 101e 0d0a             dc.b     CR,LF
01219  00000021 szemsg17    equ      *-emsg17
01220
01221  00001020             ends
```

```
00001                         nam      Disk           Driver
00002                         ttl      Device         Driver For CC74 VME scsi controller
00003 * Editon History
00004
00005 *  #    Date      Comments                                          By
00006 * -- --------- ------------------------------------------------- ---
00007 * 00  85/01/21 Fist attempt to complete new set up              N.N.
00008 *              Uses scsi NCR 5385 and dmac MC68450.
00009 * 01  85/05/22 Added code to use Xebec 1401 controller          wvv
00010 * 02  85/07/30 Added some error exits, some cleanup.
00011 *              added drive # init at 'init drive' subr.
00012 *              added second error table for 1401.
00013 *              added direct command in 'PutStat' subr.
00014 *              Tested with xebeq 1410, 1410a, 1401 and SCSI
00015 *              chip NCR 5385(E) as target.                      N.N.
00016 * 03  85/10/11 Changed code for 1401 to CC-80                   NN
00017
00018  00000003 Edition    equ      3              current edition number
00019
00020  00000e01 Typ_Lang   set      (Drivr<<8)+Objct Device Driver In Assembly Language
00021  00008000 Attr_Rev   set      (ReEnt<<8)+0
00022
00023                      psect    Ncr,Typ_Lang,Attr_Rev,Edition,0,DiskEnt
00024
00025                      use      defsfile
00007
00008
00026 ** Edition 2 changes
00027 *
00028 * these equates are usefull in de direct command mode
00029 * they refer to the block structure used, if this
00030 * block structure is changed, be sure these are also
00031 * updated
00032 *
00033  00000000 CMD_PTR    equ      00             4 bytes
00034  00000004 CMD_SIZ    equ      04             2 bytes
00035  00000006 DATA_PTR   equ      06             4 bytes
00036  0000000a DATA_SIZ   equ      10             2 bytes
00037  0000000c STAT_PTR   equ      12             4 bytes
00038  00000010 STAT_SIZ   equ      16             2 bytes
00039  00000012 MSG_PTR    equ      18             4 bytes
00040  00000016 MSG_SIZ    equ      22             2 bytes
00041  00000018 ERR_PTR    equ      24             4 bytes
00042  0000001c ERR_SIZ    equ      28             2 bytes
00043  0000001e CNT_DAT    equ      30             2 bytes
00044
00045 ** 3NN start change
00046 *
00047 * Mode offset equates, useful in accessing Mode byte's
00048 * for descripter cc-80
00049 * Offset in initialization tables for floppy's
00050
00051  00000000 M_Res0     equ      0              reserved
00052  00000001 M_Type     equ      1              media type 00=01=5"ss 02=5"ds 80=81=8"ss 82=8"ds
00053  00000002 M_Res2     equ      2              reserved
00054  00000003 M_BlkL     equ      3              length of blok list (= 16)
00055
00056  00000004 M_Dens0    equ      4              density code track 0 00=01=sd 02=dd
00057  00000005 M_NoB0     equ      5              number of blocks track 0
00058  00000008 M_Res8     equ      8              reserved
00059  00000009 M_BlkS0    equ      9              block size track 0
```

```
00060
00061  0000000c M_Densl    equ      12               density code track !0
00062  0000000d M_NoB1     equ      13               number of blocks track !0
00063  00000010 M_Res16    equ      16               reserved
00064  00000011 M_BlkS1    equ      17               block size track !0
00065
00066  00000014 M_Gap0     equ      20               gap length track 0
00067  00000015 M_FGap0    equ      21               gap length format track 0
00068  00000016 M_Gap1     equ      22               gap length track !0
00069  00000017 M_Fgap1    equ      23               gap length format track !0
00070  00000018 M_Step     equ      24               step rate
00071  00000019 M_Hlt      equ      25               head load time
00072  0000001a M_Hut      equ      26               head unload time
00073  0000001b M_MotOn    equ      27               motor on timing
00074  0000001d M_MotOff   equ      29               motor off timing
00075  0000001f M_Sect     equ      31               starting sector number
00076  00000020 M_Cil      equ      32               starting cilinder number
00077  00000021 M_MaxBlk   equ      33               highest block address
00078  00000024 M_RetryR   equ      36               retry count for read
00079  00000025 M_RetryW   equ      37               retry count for write
00080  00000026 M_RetryF   equ      38               retry count for format
00081  00000027 M_Res39    equ      39               reserved
00082
00083 ** Edition 2 end changes
00084
00085 *********************************************************************
00086 * CC74 Definitions
00087 *
00088 * board base address
00089  00fffe00 CC74_BASE  equ      $fffe00
00090
00091 * hardware control register cc-74
00092  00000101 CONTROL74  equ      $101             offset from cc-74 base
00093
00094 * NCR 5385 register offset def's
00095  00000121 DATNCR     equ      $121             data register to scsi bus
00096  00000123 COMNCR     equ      $123             command register
00097  00000125 CNTNCR     equ      $125             control register
00098  00000127 DESIDNCR   equ      $127             destination id register
00099  00000129 AUXNCR     equ      $129             auxiliary status register
00100  0000012b IDNCR      equ      $12B             id register
00101  0000012d IRQNCR     equ      $12D             interrupt register
00102  0000012f SRCIDNCR   equ      $12F             source id register
00103  00000133 DIAGNCR    equ      $133             diagnostic status register
00104  00000139 TFRNCR     equ      $139             transfer counter register
00105  00000139 TCMONCR    equ      $139             transfer counter most sign. byte
00106  0000013b TCMINCR    equ      $13B             transfer counter middle byte
00107  0000013d TCLENCR    equ      $13D             transfer counter least sign. byte
00108
00109 * DMA 68450 base address
00110  00000000 CHNL0      equ      $00
00111  00000040 CHNL1      equ      $40
00112  00000080 CHNL2      equ      $80
00113  000000c0 CHNL3      equ      $C0
00114  000000ff GENCR      equ      CHNL0+$FF
00115
00116 * DMA 68450 device definitions
00117  00000000 CSR        equ      0                channel status register
00118  00000001 CER        equ      1                channel error register
00119  00000004 DCR        equ      4                device control register
00120  00000005 OCR        equ      5                operation control register
```

```
00121  00000006 SCR        equ      6               sequence control register
00122  00000007 CHCR       equ      7               channel control register
00123  0000000a MTC        equ      $A              memory transfer counter
00124  0000000c MAR        equ      $C              memory address register
00125  00000014 DAR        equ      $14             device address register
00126  0000001a BTR        equ      $1A             base transfer register
00127  0000001c BAR        equ      $1C             base address register
00128
00129  00000025 NIV        equ      $25             normal interrupt vector
00130  00000027 EIV        equ      $27             error interrupt vector
00131  0000002d CPR        equ      $2D             channel priority register
00132
00133  00000029 MFC        equ      $29             memory function codes
00134  00000031 DFC        equ      $31             device function codes
00135  00000039 BFC        equ      $39             base function codes
00136
00137 *
00138 * device control register (R/W)
00139 *
00140
00141  00000000 BurstMod   equ      $00             burst transfer mode
00142  00000080 StealMod   equ      $80             cycle steal mode without hold
00143  000000c0 StealHld   equ      $C0             cycle steal mode with hold
00144
00145  00000000 Dev68000   equ      $00             68000 compatible device, explicitly addressed
00146  00000010 Devc6800   equ      $10             6800 compatible device, explicitly addressed
00147  00000020 DevAck     equ      $20             device with *ACK, implicitly addressed
00148  00000030 DevAckRy   equ      $30             device with *ACK and *READY, implicitly addressed
00149
00150  00000000 Dev8Bit    equ      $00             device port 8 bit
00151  00000008 Dev16Bit   equ      $08             device port 16 bit
00152  00000003 DevSiz_B   equ      3               bit number of device port size
00153
00154  00000000 StatInp    equ      0               status input - peripheral ctl line
00155  00000001 StatInpI   equ      1               status input with interrupt
00156  00000002 StartPls   equ      2               start pulse, negative 1/8 clk
00157  00000003 AbortInp   equ      3               abort input
00158
```

```
00160 *
00161 * Operation control register (R/W)
00162 *
00163
00164  00000000 MemToDev   equu     $00             transfer from memory to device
00165  00000080 DevToMem   equ      $80             transfer from device to memory
00166  00000007 XfrDir_B   equ      7               transfer direction bit number
00167
00168  00000000 ByteSize   equ      $00             operation size = byte
00169  00000010 WordSize   equ      $10             operation size = word
00170  00000020 LongSize   equ      $20             operation size = long
00171
00172  00000000 ChainDis   equ      $0              chain operation disabled
00173  00000008 ChainArr   equ      $8              array chaining enabled
00174  0000000c ChainLnk   equ      $C              linked chaining enabled
00175
00176  00000000 AuReqLim   equ      0               auto request at rate set by GCR
00177  00000001 AuReqMax   equ      1               auto request at maximum rate
00178  00000002 ReqInit    equ      2               *REQ line intitiates all operand transfers
00179  00000003 ReqInitA   equ      3               auto request first xfr, *REQ for all others
00180
00181
00182 *
00183 * Sequence Control Register (R/W)
00184 *
00185
00186  00000000 MemNoCnt   equ      0               memory address register does not count
00187  00000004 MemCntUp   equ      4               memory address register counts up
00188  00000008 MemCntDn   equ      8               memory address register counts down
00189
00190  00000000 DevNoCnt   equ      0               device address register does not count
00191  00000001 DevCntUp   equ      1               device address register counts up
00192  00000002 DevCntDn   equ      2               device address register counts down
00193
00194
00195 *
00196 * Channel Control Register (R/W)
00197 *
00198
00199  00000000 NoOpPend   equ      $00             no operation is pending
00200  00000080 StartOp    equ      $80             start operation
00201  00000007 Start_B    equ      7               bit number of start operation bit
00202
00203  00000000 NoContin   equ      $00             no continue operation is pending
00204  00000040 ContinOp   equ      $40             continue operation
00205  00000006 Contin_B   equ      6               bit number of continue op bit
00206
00207  00000000 OpNoHalt   equ      $00             operation not halted
00208  00000020 OpHalted   equ      $20             operation halted
00209  00000005 Halted_B   equ      5               bit number of halted op bit
00210
00211  00000000 NoAbort    equ      $00             operation not aborted
00212  00000010 OpAbort    equ      $10             operation aborted
00213  00000004 Abort_B    equ      4               bit number of abort op bit
00214
00215  00000000 IntrptDi   equ      0               interrupts disabled
00216  00000008 IntrptEn   equ      8               interrupts enabled
00217  00000003 Intrpt_B   equ      3               bit number of interrupt enable
00218
```

```
00220 *
00221 * Channel Status Register (R/W)
00222 *
00223 *     writing a one into any bit clears that status
00224 *     any written zero bits do not affect the status
00225 *
00226
00227  00000000 OpNoComp    equ      $00             operation incomplete
00228  00000080 OperComp    equ      $80             operation complete
00229  00000007 OpComp_B    equ      7               bit number of operation complete bit
00230
00231  00000000 BlkNoCmp    equ      $00             block transfer incomplete
00232  00000040 BlkComp     equ      $40             block transfer complete
00233  00000006 BlkCmp_B    equ      6               bit number of block transfer complete bit
00234
00235  00000000 DevTrmAb    equ      $00             device termination abnormal
00236  00000020 DevTrmNo    equ      $20             device termination normal
00237  00000005 DevTrm_B    equ      5               bit number of device termination status
00238
00239  00000010 ErrorSet    equ      $10             error occurred and is noted in CER
00240  00000004 Error_B     equ      4               bit number of error flag bit
00241
00242  00000008 ActiveCh    equ      8               channel considered active
00243  00000003 Active_B    equ      3               bit number of active channel flag bit
00244
00245  00000002 PCLTrans    equ      2               transition occurred on *PCL
00246  00000001 PCLTrn_B    equ      1               bit number of PCL transition flag bit
00247
00248  00000000 PCLLow      equ      0               *PCL line low
00249  00000001 PCLHigh     equ      1               *PCL line high
00250  00000000 PCLSts_B    equ      0               bit number of *PCL status bit
00251
00252
00253 *
00254 * Channel Error Register (R only)
00255 *
00256
00257  00000001 ErConfig    equ      $01             configuration error
00258  00000002 ErOpTimg    equ      $02             operation timing error
00259
00260  00000005 ErAdrMem    equ      $05             memory address error
00261  00000006 ErAdrDev    equ      $06             device address error
00262  00000007 ErAdrBas    equ      $07             base address error
00263
00264  00000009 ErBusMem    equ      $09             memory bus error
00265  0000000a ErBusDev    equ      $0A             device bus error
00266  0000000b ErBusBas    equ      $0B             base bus error
00267
00268  0000000d ErCntMem    equ      $0D             memory count error
00269  0000000e ErCntDev    equ      $0E             device count error
00270  0000000f ErCntBas    equ      $0F             base count error
00271
00272  00000010 ErEAbort    equ      $10             external abort
00273  00000011 ErSAbort    equ      $11             software abort
00274
```

```
00276 *
00277 * Channel Priority Register (R/W)
00278 *
00279
00280  00000000 ChPrior0   equ     0              channel priority of zero
00281  00000001 ChPrior1   equ     1              channel priority of one
00282  00000002 ChPrior2   equ     2              channel priority of two
00283  00000003 ChPrior3   equ     3              channel priority of three
00284
00285
00286 *
00287 * Function Code Registers (R/W)
00288 *
00289
00290  00000001 UserData   equ     1              user data address access
00291  00000002 UserProg   equ     2              user program address access
00292  00000005 SupvData   equ     5              supervisor data address access
00293  00000006 SupvProg   equ     6              supervisor program address access
00294
00295
00296
00297 *
00298 * General Control Register (R/W)
00299 *
00300
00301  0000000c BurstTim   equ     $C             mask for burst time
00302  00000003 BandwRat   equ     $3             mask for bandwidth ratio
00303
00304 * end CC74 definitions
00305 *******************************************************************
00306
00307  00000000 True       equ     0
00308  00000001 False      equ     1
00309 *ListExec set True list only Exec
00310  00000001 ListExec   set     False          list entire file
00311
00315
```

```
00317 * Controller Commands
00318 *
00319 * Class 0 commands
00320 *
00321  00000000 C$TRDY     equu      0                   Test For Drive Ready
00322  00000001 C$RSTR     equ       1                   Restore Head To Track 00
00323  00000003 C$RDET     equ       3                   Request Error Detail (Request sense status)
00324  00000004 C$FRMT     equ       4                   Format All Blocks
00325  00000005 C$CHKF     equ       5                   Check Track Format
00326  00000006 C$FTRK     equ       6                   Format Track
00327  00000007 C$FBAD     equ       7                   Format Bad Track
00328  00000008 C$RBLK     equ       8                   Read Block
00329  0000000a C$WBLK     equ       10                  Write A Block
00330  0000000b C$SEEK     equ       11                  Seek To Track Containing Block
00331  0000000c C$MINI     equ       12                  Set Mini Format (Initialize drive parameters)
00332  00000015 C$MODE     equ       21                  Set Mode Format (Initialize drive parameters) CC-
80
00333  0000000d C$RDLN     equ       13                  Read Length Of Error Burst
00334
00335  00000001 C$BLKCNT   equ       1                   Number of blocks to read
00336  00000030 C$BRQLVL   equ       (3<<4)              DMA bus request level 0 - 3.
00337
00338 * Bit equates DD_FMT
00339  00000000 Side_Bit   equ       0                   Side bit
00340  00000001 Dens_Bit   equ       1                   Density bit
00341
00342  00000001 ErrStat    equ       1                   bit 1 of status byte form controller
00343 * Number Of Drives Supported
00344  00000008 NumDriv    equ       8
00345  00000100 BuffSize   equ       256                 size of sector 0 buffer
00346
```

```
00348 ***********************************************************
00349 *                                                         *
00350 *     Static Storage Definitions                          *
00351 *                                                         *
00352 ***********************************************************
00353
00354                      vsect
00355  00000000 VBuffer    ds.l     64            Verify Buffer
00356  00000100 V_BufRsv   ds.l     (BuffSize/4)*NumDriv Total reserved for Sector 0's
00357  00000900 V_VBuff    ds.l     1             Pointer to verify buffer
00358  00000904 HD_cmd     ds.b     1
00359  00000905 HD_psn     ds.b     3
00360  00000908 HD_blk     ds.b     1
00361  00000909 HD_ctrl    ds.b     1
00362  0000090a HD_status  ds.b     1
00363  0000090b HD_msg     ds.b     1
00364  0000090c HD_sense   ds.b     4
00365  00000910 V_CurDrv   ds.b     1             current drive
00366 * 3NN change
00367  00000908 V_All      equ      HD_blk
00368  00000912 V_IntBuf   ds.l     10            Initialize buffer for floppy
00369  0000093a V_Tr0      ds.b     1             'off' track 0 flags
00370  0000093b V_Frmt     ds.b     1             ready to format flag
00371  0000093c V_Initflg  ds.b     1             global init flag, indicate drive par's init.
00372  00000000            ends
00373
```

```
00375 ************************************************************
00376 *                                                          *
00377 *     Long Branch And IRQ Polling Table's                  *
00378 *                                                          *
00379 ************************************************************
00380
00381 0000 000c DiskEnt    dc.w     Init           Initialize Storage
00382 0002 00ec            dc.w     Read           Read A Sector
00383 0004 0152            dc.w     Write          Write A Sector
00384 0006 01b6            dc.w     GetStat        Return No Get Status Supported
00385 0008 01b8            dc.w     PutStat        Set Status (Format And Restore)
00386 000a 01fc            dc.w     Term           Shut Down Device
00387
00388
```

```
00390 *
00391 * Init:
00392 *    Initialize device and its static storage area
00393 *    note: Prior to being called, the device permanent storage will be
00394 *          cleared (set to zero) except for V_PORT which will contain the
00395 *          device address. The driver should initialize each drive table
00396 *          appropriately for the type of disk the driver expects to be
00397 *          used on the corresponding drive.
00398 *
00399 * Input : (a1) = Address Of Device Descriptor Module
00400 *         (a2) = Address Of Statics Storage
00401 *         (a6) = system global data pointer
00402 * Output: none
00403 * Error output :(cc)   = Carry Set If Error
00404 *               (d1.w) = Error Code If Error
00405 *
00406
00407 *
00408 * Init Static Storage
00409 *
00410            Init
00411 000c 7008           moveq    #NumDriv,d0    Number Of Drives
00412 000e=1540           move.b   d0,V_NDRV(a2)
           0000
00413 0012 72ff           moveq    #$FF,d1        Init Fake Media Size
00414 0014 1541           move.b   d1,V_CurDrv(a2) Init high drive #
           0910
00415 0018=41ea           lea      DRVBEG(a2),a0  Point at first table
           0000
00416 001c 47ea           lea      V_BufRsv(a2),a3 Point at sector zero buffer
           0100
00417 0020=1141 Init10    move.b   d1,DD_TOT(a0)  Init to non zero
           0000
00418 0024=1141           move.b   d1,V_TRAK(a0)
           0000
00419 0028=214b           move.l   a3,V_ScZero(a0)
           0000
00420 002c 47eb           lea      BuffSize(a3),a3 point to next buffer
           0100
00421 0030=41e8           lea      DRVMEM(a0),a0  Move To Next Table
           0000
00422 0034 5300           subq.b   #1,d0          last drive?
00423 0036 66e8           bne.s    Init10         branch if not
00424 *
00425 * get verify buffer
00426 *
00427 0038 41ea           lea      VBuffer(a2),a0
           0000
00428 003c 2548           move.l   a0,V_VBuff(a2) save pointer for later use
           0900
00429 *
00430 * Reset controller and interface
00431 *
00432 0040=266a           movea.l  V_PORT(a2),a3  point to port
           0000
00433 0044 08eb           bset     #7,CONTROL74(a3) Reset SCSI bus
           00070101
00434 * Edition 2 change
00435 004a 303c           move.w   #$ffff,d0
           ffff
00436 004e 51c8 Init15    dbra     d0,Init15
```

```
            fffe
00437 * Edition 2 change end
00438 0052 08ab            bclr     #7,CONTROL74(a3) clr reset bit
            00070101
00439 0058 177c            move.b   #0,COMNCR(a3)  NCR chip reset
            00000123
00440 005e 203c            move.l   #350,d0        Init timout
            0000015e
00441 0064 51c8 Init20     dbra     d0,Init30      Time Out?
            0004
00442 0068 6072            bra.s    Init40         yes Something wrong
00443 006a 082b Init30     btst     #7,DIAGNCR(a3) wait for command ready
            00070133
00444 0070 67f2            beq.s    Init20
00445 0072 162b            move.b   DIAGNCR(a3),d3 get diagnostic results
            0133
00446 0076 b63c            cmp.b    #$80,d3
            0080
00447 007a 6660            bne.s    Init40         Something wrong with NCR chip
00448 007c=1029            move.b   M$IRQLvl(a1),d0 get irq level
            0000
00449 0080 0000            ori.b    #C$BRQLVL,d0   get bus request level
            0030
00450 0084 1740            move.b   d0,CONTROL74(a3) set bus request level & irq level
            0101
00451 0088 08eb            bset     #3,CONTROL74(a3) enable irq
            00030101
00452 * Init DMA Controller
00453 * 1 Sequense control register : Memory counts up ; device does not count
00454 008e 177c            move.b   #(MemCntUp+DevNoCnt),SCR+CHNL0(a3)
            00040006
00455 * 2 Device function code : User data
00456 0094 177c            move.b   #UserData,DFC+CHNL0(a3)
            00010031
00457 * 3 Memory function code : User Data
00458 009a 177c            move.b   #UserData,MFC+CHNL0(a3)
            00010029
00459 * 4 channel priority register : channel 0 highest priority
00460 00a0 177c            move.b   #ChPrior0,CPR+CHNL0(a3)
            0000002d
00461 * 5 Device address register : Data register from ncr chip
00462 00a6 4beb            lea      DATNCR(a3),a5
            0121
00463 00aa 274d            move.l   a5,DAR+CHNL0(a3)
            0014
00464 * 6 Normal interrupt vector : from device descriptor
00465 00ae=1769            move.b   M$Vector(a1),NIV+CHNL0(a3) Normal vector
            00000025
00466 * 7 Error Interrupt vector : from device descriptor
00467 00b4=1769            move.b   M$Vector(a1),EIV+CHNL0(a3) Error vector
            00000027
00468 * Edition 2 change
00469 * 8 Device Control Register : Steal Hold , 8 bits device with ack , pcl inp
00470 00ba 177c            move.b   #(StealHld+DevAck+Dev8Bit+StatInpI),DCR+CHNL0(a3)
            00e10004
00471 * Edition 2 end change
00472 *
00473 *
00474 * Put Device on The Polling Table
00475 *
00476 00c0=1029            move.b   M$Vector(a1),d0
```

```
           0000
00477 00c4 0c00               cmpi.b   #64,d0         is it a legal vector
           0040
00478 00c8 6518               blo.s    Init50         branch if not
00479 00ca=1229               move.b   M$Prior(a1),d1 get priority level
           0000
00480 00ce 41fa               lea      IrqEntry(pcr),a0 Point To IRQ Routine
           014a
00481 00d2=4e40               os9      F$IRQ          Get On The Table
           0000
00482 00d6 650e               bcs.s    Init60         Error
00483 00d8 7200               moveq    #0,d1          clear carry
00484 00da 4e75               rts
00485
00486 00dc=323c Init40        move.w   #E$NotRdy,d1
           0000
00487 00e0 6004               bra.s    Init60
00488 00e2=323c Init50        move.w   #E$Unit,d1
           0000
00489
00490           Init60
00491 00e6 003c               ori      #1,ccr         set carry
           0001
00492 00ea 4e75               rts                     exit with error
00493
```

```
00495 *
00496 * Read:
00497 *    Read a 256 byte sector
00498 *    note: Whenever logical sector zero is read, the first part is
00499 *          copied into the appropriate drive table. PD_DTB contains a
00500 *          pointer to the proper drive table entry. The number of bytes
00501 *          to copy is DD_SIZ.
00502 *
00503 * Input: (a1) = Address Of The Path Descriptor
00504 *        (a2) = Address Of The Device Static Storage
00505 *        (a4) = Process descriptor pointer
00506 *        (a5) = caller's register stack pointer
00507 *        (a6) = system global data storage pointer
00508 *        (d2.1) = Disk logical sector number to read
00509 *
00510 * Output: sector in sector buffer
00511 *
00512 * Error output: (CC) = Carry Set If Error
00513 *               (d1.w) = Error Code If Error
00514 *
00515 *
00516           Read
00517 00ec 6100           bsr      InitDriv
          03ca
00518 00f0 655e           bcs.s    Read90         error occured
00519
00520           Read10
00521 00f2 4a82           tst.1    d2             reading sector zero?
00522 00f4 670a           beq.s    Read20         branch if so
00523 00f6=2a69           movea.1  PD_BUF(a1),a5
          0000
00524 00fa 6100           bsr      RdSector       Execute Read sector and exit
          0536
00525 00fe 6050           bra.s    Read90
00526
00527 * Here If Sector 0 Being Read
00528
00529 0100=2069 Read20    movea.1  PD_DTB(a1),a0  get drive table pointer
          0000
00530 0104 7800           moveq    #0,d4          read once on floppy
00531 0106=4a29           tst.b    PD_TYP(a1)     hard disk ?
          0000
00532 010a 6a0a           bpl.s    Read25         ..no, don't use sector 0 buffer
00533 010c=4a28           tst.b    V_ZeroRd(a0)   sector zero been read?
          0000
00534 0110 6628           bne.s    Read60         branch if so
00535 0112 383c           move.w   #20000,d4      try several times (in case not spun up)
          4e20
00536 0116=2a68 Read25    move.l   V_ScZero(a0),a5 point to buffer
          0000
00537 011a 6100 Read30    bsr      RdSector       Execute do the read
          0516
00538 011e 6406           bcc.s    Read40         if no error move data
00539 0120 51cc           dbra     d4,Read30      try again
          fff8
00540 0124 602a           bra.s    Read90         exit with error
00541 0126=117c Read40    move.b   #1,V_ZeroRd(a0) flag sector zero has been read
          00010000
00542 012c=323c           move.w   #DD_SIZ-1,d1   UpDate Drive Table
          ffff
00543 0130 11b5 Read50    move.b   (a5,d1.w),(a0,d1.w)
```

```
        10001000
00544 0136 51c9           dbra     d1,Read50       branch if not
        fff8
00545
00546 013a=2068 Read60     movea.l  V_ScZero(a0),a0
        0000
00547 013e=2a69            movea.l  PD_BUF(a1),a5
        0000
00548 0142 323c           move.w   #63,d1          move 256 bytes
        003f
00549 0146 2ad8 Read70     move.l   (a0)+,(a5)+
00550 0148 51c9           dbra     d1,Read70
        fffc
00551 014c 7200 Read80     moveq    #0,d1           No Errors
00552 014e 4e75           rts
00553
00554 0150 4e75 Read90     rts                      Leave with carry set
00555
```

```
00557 *
00558 * Write:
00559 *    Write a 256 byte sector
00560 *
00561 * Input: (a1) = Address Of The Path Descriptor
00562 *        (a2) = Address Of The Device Static Storage
00563 *        (a4) = Process descriptor pointer
00564 *        (a5) = caller's register stack pointer
00565 *        (a6) = system global data storage pointer
00566 *        (d2.l) = Disk logical sector number to write
00567 *
00568 * Output: sector buffer is written on disk
00569 *
00570 * Error output: (CC) = Carry Set If Error
00571 *               (d1.w) = Error Code If Error
00572 *
00573           Write
00574 0152 6100           bsr      InitDriv
           0364
00575 0156 6500           bcs      Write90        error occurred
           0058
00576 015a=2a69           movea.l  PD_BUF(a1),a5
           0000
00577 015e 2f02           move.l   d2,-(a7)       save sector number
00578 0160 6100           bsr      WrSector       Execute The Command
           052e
00579 0164 4cdf           movem.l  (a7)+,d2       retreive sector number
           0004
00580 0168 6546           bcs.s    Write90        Leave If Error
00581 016a 4a82           tst.l    d2             was it sector 0
00582 016c 6608           bne.s    Write10        branch if not
00583 016e=2069           movea.l  PD_DTB(a1),a0  get drive table pointer
           0000
00584 0172=4228           clr.b    V_ZeroRd(a0)   flag sector zero writen
           0000
00585 0176=4a29 Write10   tst.b    PD_VFY(a1)     Verify ?
           0000
00586 017a 662c           bne.s    Write20        No, Leave
00587 017c=2f29           move.l   PD_BUF(a1),-(a7) save buffer pointer
           0000
00588 0180=236a           move.l   V_VBuff(a2),PD_BUF(a1) substitute verify buffer
           09000000
00589 0186 6100           bsr      Read           Re-Read The Written Block
           ff64
00590 018a 4cdf           movem.l  (a7)+,a0       recover buffer pointer
           0100
00591 018e 40e7           move.w   sr,-(a7)       save cc
00592 0190=2348           move.l   a0,PD_BUF(a1)
           0000
00593 0194 46df           move.w   (a7)+,sr       restore cc
00594 0196 6514           bcs.s    Write80        exit with error
00595 0198 266a           movea.l  V_VBuff(a2),a3
           0900
00596 019c 303c           move.w   #(BuffSize/4)-1,d0 get # of long words to check
           003f
00597 01a0 b788 Verify10  cmpm.l   (a0)+,(a3)+    is data the same?
00598 01a2 6608           bne.s    Write80        branch if not
00599 01a4 51c8           dbra     d0,Verify10    last word?,branch if not
           fffa
00600 01a8 4241 Write20   clr.w    d1             No Errors
00601 01aa 4e75           rts
```

```
00602
00603            Write80
00604 01ac=323c             move.w   #E$Write,d1    flag write error
           0000
00605 01b0=003c Write90     ori      #Carry,ccr     flag error
           0000
00606 01b4 4e75             rts
00607
00608
```

```
*
* GetStat:
*       Get device Status
*
* Input: (a1) = Address Of The Path Descriptor
*        (a2) = Address Of The Device Static Storage
*        (a4) = Process descriptor pointer
*        (a5) = caller's register stack pointer
*        (a6) = system global data storage pointer
*        (d0.w) = status code
*
* Output: Depends on status code
* Error Output: (CC) = C bit set if error
*               (d1.w) = Apropriate error code
*
          GetStat
01b6 4e75            rts                     Not supported

```

```
00629 *
00630 * PutStat:
00631 *       Set device Status
00632 *
00633 * Input: (a1) = Address Of The Path Descriptor
00634 *        (a2) = Address Of The Device Static Storage
00635 *        (a4) = Process descriptor pointer
00636 *        (a5) = caller's register stack pointer
00637 *        (a6) = system global data storage pointer
00638 *        (d0.w) = status code
00639 *
00640 * Output: Depends on status code
00641 * Error Output: (CC) = C bit set if error
00642 *               (d1.w) = Apropriate error code
00643 *
00644                PutStat
00645 01b8=266a                movea.l  V_PORT(a2),a3
           0000
00646 01bc=0c40                cmpi.w   #SS_WTrk,d0    is it a Write Track call?
           0000
00647 01c0 6608                bne.s    PutStat10
00648 01c2 6100                bsr      WriteTrk       branch if so
           0176
00649 01c6 6428                bcc.s    PutStat80      no error
00650 01c8 602c                bra.s    PutStat90
00651 01ca=0c40 PutStat10      cmpi.w   #SS_Reset,d0   is it a restore call?
           0000
00652 01ce 6608                bne.s    PutStat20
00653 01d0 6100                bsr      Restore        branch if so
           010a
00654 01d4 641a                bcc.s    PutStat80      no error
00655 01d6 601e                bra.s    PutStat90
00656 01d8=0c40 PutStat20      cmpi.w   #SS_DCmd,d0    is it a direct command?
           0000
00657 01dc 66fa                bne.s    PutStat20
00658 01de 6100                bsr      DirectCmd      branch if so
           024a
00659 01e2 640c                bcc.s    PutStat80      no error
00660 01e4 6010                bra.s    PutStat90
00661          PutStat30
00662 01e6=323c                move.w   #E$UnkSvc,d1   flag unknown service code
           0000
00663 01ea=003c                ori      #Carry,ccr     flag error
           0000
00664 01ee 4e75                rts
00665
00666 01f0 323c PutStat80      move.w   #0,d1
           0000
00667 01f4 4e75                rts
00668
00669 01f6=003c PutStat90      ori      #Carry,ccr
           0000
00670 01fa 4e75                rts
00671
```

```
00673 *
00674 * Terminate:
00675 *      Terminate Device
00676 *
00677 * Input: (a1) - Adress of the device descripter module
00678 *        (a2) - Address Of Device Static Storage
00679 *        (a6) - System global static storage
00680 *
00681 * Output: none
00682 * Error Output: (CC) - Carry Set If Error
00683 *               (d1.w) - Error Code If Error
00684 *
00685 *
00686                  Term
00687 01fc=4a6a             tst.w    V_WAKE(a2)      See if I/O busy
           0000
00688 0200 6708             beq.s    Term20          Yes
00689 0202 7001 Term10      moveq    #1,d0           give up slice
00690 0204=4e40             os9      F$Sleep         I/O should be done within 1 slice
           0000
00691 0208 60f2             bra.s    Term
00692 020a=1029 Term20      move.b   M$Vector(a1),d0
           0000
00693 020e=1229             move.b   M$Prior(a1),d1
           0000
00694 0212 91c8             suba.l   a0,a0
00695 0214=4e40             os9      F$IRQ           Delete Entry
           0000
00696 0218 4e75             rts
00697
```

```
00699 *
00700 * IRQ service routine
00701 *      Service device interrupts
00702 *
00703 * Inputs: (a2) static storage address
00704 *         (a3) port address
00705 *         (a6) system global static storage
00706 * Output: (CC) = cleared
00707 *
00708 * It checks the NCR chip for the 'Bus Phase' and initiate
00709 * the 'Transfer Info command' until the bus phase is the
00710 * data phase. Then the DMA is enabled and we wait for the
00711 * 'status phase'
00712 *
00713 * Note: Although this is an IRQ routine it's called as a subroutine
00714 *        so we leave with RTS
00715 *
00716               IrqEntry
00717 021a 082b             btst     #3,CONTROL74(a3) Check if irq enabled
           00030101
00718 0220 6710             beq.s    irqen05
00719 0222 102b             move.b   CHNL0+CSR(a3),d0
           0000
00720 0226 0800             btst     #OpComp_B,d0
           0007
00721 022a 665e             bne.s    irqen50        Operation completed
00722 022c 0800             btst     #PCLSts_B,d0
           0000
00723 0230 6706             beq.s    irqen10        Irq from dma pcl line occured
00724               irqen05
00725 0232 003c             ori.b    #1,ccr
           0001
00726 0236 4e75             rts                     Return as quick as possible
00727               irqen10
00728 0238 08ab             bclr     #3,CONTROL74(a3) disable irq
           00030101
00729 * Edition 2 changes
00730 023e 177c             move.b   #$03,CSR+CHNL0(a3) clear irq cause
           00030000
00731 0244 08eb             bset     #3,CONTROL74(a3) enable irq
           00030101
00732 * Edition 2 end changes
00733               irqen15
00734 024a 102b             move.b   AUXNCR(a3),d0
           0129
00735 024e 122b             move.b   IRQNCR(a3),d1
           012d
00736 *   bne.s irqen15 still irq's
00737 0252 0200             andi.b   #$38,d0        mask off msg, c/d and i/o line
           0038
00738 0256 b03c             cmp.b    #$00,d0        test Data Out phase
           0000
00739 025a 6716             beq.s    irqen20
00740 025c b03c             cmp.b    #$08,d0        test Data In phase
           0008
00741 0260 6710             beq.s    irqen20
00742 0262 b03c             cmp.b    #$18,d0        test Status phase
           0018
00743 0266 6754             beq.s    irqen30
00744 0268 177c             move.b   #$94,COMNCR(a3) give transfer info command, dma
           00940123
```

```
00745 * Edition 2 change
00746 * bset #3,CONTROL74(a3) enable irq
00747 * Edition 2 end changes
00748 026e 7200            moveq    #0,d1           no error
00749 0270 4e75            rts
00750          irqen20
00751 * Edition 2 change
00752 *   move.b #$03,CSR+CHNL0(a3) clear irq cause
00753 *   bset #3,CONTROL74(a3) enable irq
00754 * Edition 2 end changes
00755 0272 082b            btst     #Active_B,CHNL0+CSR(a3) test if channel already started
           00030000
00756 0278 6606            bne.s    irqen40         yes, started
00757 027a 08eb            bset     #Start_B,CHNL0+CHCR(a3) enable dma controller
           00070007
00758          irqen40
00759 0280 177c            move.b   #$94,COMNCR(a3) give transfer info command, dma
           00940123
00760 0286 7200            moveq    #0,d1           no error
00761 0288 4e75            rts
00762          irqen50
00763 028a 08ab            bclr     #3,CONTROL74(a3) disable irq
           00030101
00764          irqen55
00765 0290 102b            move.b   AUXNCR(a3),d0  read until no more irq's pending
           0129
00766 0294 122b            move.b   IRQNCR(a3),d1
           012d
00767 0298 66f6            bne.s    irqen55
00768 029a 082b            btst     #XfrDir_B,OCR+CHNL0(a3) Are we reading?
           00070005
00769 02a0 661a            bne.s    irqen30         ..yes
00770 02a2 0200            andi.b   #$38,d0         mask off msg, c/d and i/o line
           0038
00771 02a6 b03c            cmp.b    #$18,d0         test Status phase
           0018
00772 02aa 6710            beq.s    irqen30
00773 02ac 177c            move.b   #OperComp,CSR+CHNL0(a3) clear irq COC bit.
           00800000
00774 02b2 08eb            bset     #3,CONTROL74(a3) enable irq
           00030101
00775 02b8 7200            moveq    #0,d1           Irq serviced
00776 02ba 4e75            rts
00777          irqen30
00778 02bc 08ab            bclr     #Intrpt_B,CHNL0+CHCR(a3) disable interrupts
           00030007
00779 02c2 08eb            bset     #3,CONTROL74(a3) enable irq
           00030101
00780 02c8=302a            move.w   V_WAKE(a2),d0  waiting for irq
           0000
00781 02cc 670a            beq.s    irqen90         branch if not
00782 02ce=426a            clr.w    V_WAKE(a2)      flag irq serviced
           0000
00783 02d2=7200            moveq    #S$Wake,d1      wake up waiting process
00784 02d4=4e40            os9      F$Send          send signal
           0000
00785
00786 02d8 7200 irqen90    moveq    #0,d1           Interrupt Serviced
00787 02da 4e75            rts
00788
00789
```

```
*
* Restore:
*    Called from PutStat
* Input:
*        (a1) = Address Of The Path Descriptor
*        (a2) = Address Of The Device Static Storage
*        (a4) = Process descriptor pointer
*        (a5) = caller's register stack pointer
*        (a6) = system global data storage pointer
*        (d0.w) = status code
*
* Output: head assembly restored
* Error Output : (CC) = carry set if error
*                (d1.w) = Error code if error
*
          Restore
* 3NN change
02dc=1229            move.b   PD_DRV(a1),d1  get drive number
     0000
02e0 e409            lsr.b    #2,d1          make controller number
02e2 660e            bne.s    Rest80         it is CC-80
02e4 7400            moveq    #0,d2          take sector 0
02e6 6100            bsr      InitDriv       Init Controller
     01d0
02ea 6508            bcs.s    Rest90         error
02ec 6100            bsr      Recal          execute recalibrate
     0008
02f0 6002            bra.s    Rest90
02f2 7000 Rest80     moveq    #0,d0
02f4 4e75 Rest90     rts

```

```
00820 *
00821 * Recalibrate
00822 * Input:
00823 *     (a2) = static storage
00824           Recal
00825 02f6 123c            move.b  #C$RSTR,d1     Recalibrate cmd
           0001
00826 02fa 243c            move.l  #0,d2          fake Psn
           00000000
00827 0300 163c            move.b  #C$BLKCNT,d3
           0001
00828 0304 6100            bsr     SetUp
           03e2
00829 0308 323c            move.w  #6,d1          get count
           0006
00830 030c 4bea            lea     HD_cmd(a2),a5  cmd ptr
           0904
00831 0310 6100            bsr     NCRcmd
           045a
00832 * Edition 2 change
00833 0314 6522            bcs.s   Recal15
00834 0316 4bea            lea     HD_status(a2),a5
           090a
00835 031a 6100            bsr     NCRstatus
           04c2
00836 031e 4bea            lea     HD_msg(a2),a5
           090b
00837 0322 6100            bsr     NCRmsg
           04d6
00838 * Edition 2 change
00839 0326 082a            btst    #ErrStat,HD_status(a2)
           0001090a
00840 032c 6706            beq.s   Recal10        skip if no error
00841 032e 6100            bsr     senstat
           05e6
00842 0332 4e75            rts
00843 0334 323c Recal10    move.w  #0,d1
           0000
00844 0338 4e75 Recal15    rts
00845
```

```
*
* WriteTrk:
*     Format a track
*     note: Currently no format track is supported, instead a compleet
*           drive format is done. For definition of R$d# see I$SetStt call
* Input: (a1) = Address Of The Path Descriptor
*        (a2) = Address Of The Device Static Storage
*        (a4) = Process descriptor pointer
*        (a5) = caller's register stack pointer
*        (a6) = system global data storage pointer
*        (d0.w) = status code
*
                   WriteTrk
033a 4a2a                   tst.b     V_Frmt(a2)      ready to format ?
     093b
033e 6600                   bne       WritTk20        ..yes
     0020
0342=2869                   movea.l   PD_RGS(a1),a4   point to users reg stack
     0000
0346=242c                   move.l    R$d2(a4),d2     track zero get track #
     0000
034a 6600                   bne       WrtTrkEx        exit if not
     00a0
034e=122c                   move.b    R$d3+2(a4),d1   is it side 0
     0002
0352 6600                   bne       WrtTrkEx        exit if not
     0098
0356 157c                   move.b    #1,V_Frmt(a2)   ready to format next time
     0001093b
035c 6000                   bra       WrtTrkEx
     008e

                   WritTk20
0360 422a                   clr.b     V_Frmt(a2)
     093b
0364 7400                   moveq     #0,d2           track 0
0366=2069                   move.l    PD_DTB(a1),a0   get drive table pointer
     0000
036a=2869                   movea.l   PD_RGS(a1),a4   point to users reg stack
     0000
036e=116c                   move.b    R$d3+3(a4),DD_FMT(a0)
     00030000
0374 6100                   bsr       InitDriv
     0142
0378 6500                   bcs       WrtTrkEr        ..error
     0076
037c 48e7                   movem.l   d2/a4,-(a7)
     2008
0380=266a                   move.l    V_PORT(a2),a3   get Port Address
     0000
0384 2542                   move.l    d2,HD_cmd(a2)   store logical sector number
     0904
0388 157c                   move.b    #C$FRMT,HD_cmd(a2) store command
     00040904
038e=1429                   move.b    PD_DRV(a1),d2   get drive #
     0000
0392 e40a                   lsr.b     #2,d2           make it controller number
0394 670c                   beq.s     WritTk22        it is 1410
0396 157c                   move.b    #$e5,HD_cmd+2(a2) set filler byte
     00e50906
039c 157c                   move.b    #0,HD_cmd+3(a2) clr msb byte interleave
```

```
           00000907
00887 03a2=156c WritTk22   move.b   R$d4+3(a4),HD_blk(a2) store interleave
           00030908
00888 03a8=1429            move.b   PD_DRV(a1),d2  get drive #
           0000
00889 03ac 0202            andi.b   #$03,d2        isolate drive number
           0003
00890 03b0 eb0a            lsl.b    #5,d2          adjust it
00891 03b2 842a            or.b     HD_psn(a2),d2  or into MSB's of address
           0905
00892 03b6 1542            move.b   d2,HD_psn(a2)  store it
           0905
00893 03ba=1569            move.b   PD_STP(a1),HD_ctrl(a2) load the option byte
           00000909
00894 03c0 022a            andi.b   #$BF,HD_ctrl(a2) clear 'a' retry bit
           00bf0909
00895 03c6 6100            bsr      Form           execute the command
           002e
00896 03ca 4cdf            movem.l  (a7)+,d2/a4
           1004
00897 03ce 641c            bcc.s    WrtTrkEx       exit if no errors
00898 03d0=b23c            cmp.b    #E$Seek,d1     can we continue?
           0000
00899 03d4 661a            bne.s    WrtTrkEr       branch if not
00900 03d6 242a            move.l   HD_sense(a2),d2 valid controller address?
           090c
00901 03da 6a14            bpl.s    WrtTrkEr       branch if not
00902 03dc 0282            andi.l   #$001FFFE0,d2  get even track #
           001fffe0
00903 03e2 0682            addi.l   #32,d2         continue with next track
           00000020
00904 03e8 6000            bra      WritTk20       do command again
           ff76
00905
00906 03ec 7200 WrtTrkEx   moveq    #0,d1          clear error
00907 03ee 4e75            rts
00908
00909 03f0=003c WrtTrkEr   ori      #Carry,ccr     flag error
           0000
00910 03f4 4e75            rts
00911
00912
```

```
00914 *
00915 *  Format Drive
00916 *
00917 *  Input:
00918 *       (a2) = static storage
00919              Form
00920 03f6 323c            move.w   #6,d1          get count
           0006
00921 03fa 4bea            lea      HD_cmd(a2),a5  cmd ptr
           0904
00922 03fe 6100            bsr      NCRcmd
           036c
00923 * Edition 2 change
00924 0402 6524            bcs.s    Form15
00925 0404 4bea            lea      HD_status(a2),a5
           090a
00926 0408 6100            bsr      NCRstatus
           03d4
00927 040c 4bea            lea      HD_msg(a2),a5
           090b
00928 0410 6100            bsr      NCRmsg
           03e8
00929 * Edition 2 change
00930 0414 082a            btst     #ErrStat,HD_status(a2)
           0001090a
00931 041a 6700            beq      Form10         skip if no error
           0008
00932 041e 6100            bsr      senstat
           04f6
00933 0422 4e75            rts
00934 0424 323c Form10     move.w   #0,d1
           0000
00935 0428 4e75 Form15     rts
00936
```

```
00938 * Edition 2 changes
00939 **************************
00940 *
00941 * Direct command
00942 *
00943 *  R$a0(PD_RGS(a1)) = ptr to sasi cmd block
00944 *
00945 *  (a0) ->00 command block ptr     (4 bytes)
00946 *         04 size of command block (2 bytes) *
00947 *         06 data block ptr        (4 bytes)
00948 *         10 size of data block    (2 bytes)
00949 *         12 status block ptr      (4 bytes) *
00950 *         16 size of status block  (2 bytes) *
00951 *         18 message block ptr     (4 bytes) *
00952 *         22 size of message block (2 bytes) *
00953 *         24 error block ptr       (4 bytes)
00954 *         28 size of error block   (2 bytes) *
00955 *         30 control word          (2 bytes)
00956 *            bit 0 = 0 from <mem> to <dev>
00957 *                    1 from <dev> to <mem>
00958 *            bit 1 = 0 no data phase expected
00959 *                    1 data phase expected
00960 *
00961 *   (*)  not used in this implementation of direct cmd.
00962 *
00963 * return: if ERROR then CC=set and (d1.w)=Error-code
00964 *         else CC=clr
00965           DirectCmd
00966 042a=2869           movea.l  PD_RGS(a1),a4  point to user reg. stack.
           0000
00967 042e=42ac           clr.l    R$d0(a4)       assume no error.
           0000
00968 0432=286c           movea.l  R$a0(a4),a4    point to command block struct
           0000
00969 0436 41ea           lea      HD_cmd(a2),a0
           0904
00970 043a 266c           movea.l  CMD_PTR(a4),a3 point to command block
           0000
00971 043e 303c           move.w   #5,d0
           0005
00972 0442 10db Dcmnd05   move.b   (a3)+,(a0)+    copy command to static storage
00973 0444 51c8           dbra     d0,Dcmnd05
           fffc
00974 0448 323c           move.w   #$6,d1
           0006
00975 044c 4bea           lea      HD_cmd(a2),a5
           0904
00976 0450=266a           movea.l  V_PORT(a2),a3
           0000
00977 0454 6100           bsr      NCRcmd
           0316
00978 0458 6500           bcs      Dcmnd15
           0058
00979 045c 342c           move.w   CNT_DAT(a4),d2
           001e
00980 0460 0802           btst     #1,d2
           0001
00981 0464 671a           beq.s    Dcmnd25        no data phase
00982 0466 2a6c           movea.l  DATA_PTR(a4),a5 get data block pointer
           0006
00983 046a 322c           move.w   DATA_SIZ(a4),d1 get size
```

```
           000a
00984 046e 103c             move.b   #MemToDev.d0   assume <mem> to <dev>
           0000
00985 0472 0802             btst     #0.d2
           0000
00986 0476 6704             beq.s    Dcmnd20
00987 0478 103c             move.b   #DevToMem.d0
           0080
00988 047c 6100 Dcmnd20     bsr      NCRdma
           03b4
00989 0480 4bea Dcmnd25     lea      HD_status(a2),a5
           090a
00990 0484 6100             bsr      NCRstatus
           0358
00991 0488 4bea             lea      HD_msg(a2),a5
           090b
00992 048c 6100             bsr      NCRmsg
           036c
00993 * Edition 2 change
00994 0490 082a             btst     #ErrStat,HD_status(a2)
           0001090a
00995 0496 671c             beq.s    Dcmnd90
00996 0498 6100             bsr      senstat
           047c
00997 049c 266c             movea.l  ERR_PTR(a4),a3 point to error block
           0018
00998 04a0 303c             move.w   #$3,d0
           0003
00999 04a4 41ea             lea      HD_sense(a2),a0
           090c
01000 04a8 16d8 Dcmnd10     move.b   (a0)+,(a3)+    copy error details
01001 04aa 51c8             dbra     d0,Dcmnd10
           fffc
01002 04ae=003c             ori      #Carry,ccr
           0000
01003 04b2 4e75 Dcmnd15     rts                     leave with carry set and D1 with error code
01004 04b4 7000 Dcmnd90     moveq.l  #0,d0
01005 04b6 4e75             rts                     leave with carry clear, d0 also clr.
01006 ** Edition 2 end change
01007
```

```
01009 *
01010 ******************************************
01011 *
01012 *INIDRV
01013 *
01014 *Initialise controller from drive parameters in path descriptor
01015 *
01016 *Input: (a1) = path descriptor
01017 *       (a2) = Static storage
01018 *
01019 * 3NN changes
01020            InitDriv
01021 04b8 7200              moveq    #0,d1          clear all of d1
01022 04ba=1229              move.b   PD_DRV(a1),d1  get drive number
           0000
01023 04be 4a82              tst.l    d2             sector 0?
01024 04c0 6716              beq.s    IniDrv10       ..yes
01025 04c2=4a29              tst.b    PD_TYP(a1)     hard disk?
           0000
01026 04c6 6b00              bmi      IniDrv90       ..yes, no action
           00e4
01027 04ca=b469              cmp.w    PD_TOS(a1),d2  in track 0?
           0000
01028 04ce 6446              bcc.s    IniDrv20       ..no
01029 04d0 032a              btst     d1,V_Tr0(a2)   already on track 0?
           093a
01030 04d4 6700              beq      IniDrv90       ..yes, no action
           00d6
01031
01032            IniDrv10
01033 * Going to track 0 or going to sector 0 of hard disk
01034 04d8=4a29              tst.b    PD_TYP(a1)     hard disk ?
           0000
01035 04dc 6a00              bpl      IniDrv15       ..no
           0030
01036 04e0 48e7              movem.l  d0/d2-d7/a0-a6,-(a7)
           bffe
01037 04e4=2a69              movea.l  PD_DEV(a1),a5  get device table pointer
           0000
01038 04e8=2a6d              movea.l  V$DESC(a5),a5  point to descriptor
           0000
01039 04ec 7000              moveq    #0,d0
01040 04ee=302d              move.w   M$DevCon(a5),d0 get offset to init bytes
           0000
01041 04f2 5440              addq     #2,d0          point past byte count
01042 04f4 dbc0              adda.l   d0,a5          set up pointer to init bytes
01043 04f6 49ea              lea      V_IntBuf(a2),a4 point to buffer
           0912
01044 04fa 7007              moveq    #7,d0          number of bytes - 1
01045 04fc 18dd IniDrv12     move.b   (a5)+,(a4)+
01046 04fe 51c8              dbra     d0,IniDrv12    do all bytes
           fffc
01047 0502 6100              bsr      Doinit         do the command
           00ac
01048 0506 4cdf              movem.l  (a7)+,d0/d2-d7/a0-a6
           7ffd
01049 050a 6000              bra      IniDrv95
           00a2
01050 050e 03aa IniDrv15     bclr     d1,V_Tr0(a2)   say on track 0
           093a
01051 0512 6000              bra      IniDrv25
```

```
            000a
01052
01053              IniDrv20
01054 0516 03ea                   bset     d1,V_Tr0(a2)   already off track 0? (set flag if not)
            093a
01055 051a 6600                   bne      IniDrv90       ..yes, no action
            0090
01056 051e 48e7 IniDrv25   movem.l  d0/d2-d7/a0-a6,-(a7)
            bffe
01057 0522=2069                   movea.l  PD_DTB(a1),a0  point to current drive table
            0000
01058 0526=2a69                   movea.l  PD_DEV(a1),a5  get device table pointer
            0000
01059 052a=2a6d                   movea.l  V$DESC(a5),a5  point to descriptor
            0000
01060 052e 7000                   moveq    #0,d0
01061 0530=302d                   move.w   M$DevCon(a5),d0 get offset to init bytes
            0000
01062 0534 5440                   addq     #2,d0          point past byte count
01063 0536 dbc0                   adda.l   d0,a5          set up pointer to init bytes
01064 0538 49ea                   lea      V_IntBuf(a2),a4 point to buffer
            0912
01065 053c 103c                   move.b   #39,d0         number of bytes
            0027
01066 0540 18dd IniDrv30   move.b   (a5)+,(a4)+    copy descriptor to buffer
01067 0542 51c8                   dbra     d0,IniDrv30
            fffc
01068 0546 4bea                   lea      V_IntBuf(a2),a5 point to Init buffer again
            0912
01069 054a 102d                   move.b   M_Type(a5),d0  get number of sides from table
            0001
01070 054e 0200                   andi.b   #$f0,d0        clear number of heads
            00f0
01071 0552=0828                   btst     #Side_Bit,DD_FMT(a0) double sided?
            00000000
01072 0558 6706                   beq.s    IniDrv40       ..no
01073 055a 0000                   ori.b    #$02,d0        double sided
            0002
01074 055e 6004                   bra.s    IniDrv50
01075 0560 0000 IniDrv40   ori.b    #$01,d0        single sided
            0001
01076 0564 1b40 IniDrv50   move.b   d0,M_Type(a5)  restore number of heads in table
            0001
01077 0568=0828                   btst     #Dens_Bit,DD_FMT(a0) double density?
            00010000
01078 056e 6612                   bne.s    IniDrv60       ..yes
01079 0570 2b6d                   move.l   M_Dens0(a5),M_Dens1(a5) Rest of media is like track 0
            0004000c
01080 0576 2b6d                   move.l   M_Res8(a5),M_Res16(a5) Copy param's
            00080010
01081 057c 3b6d                   move.w   M_Gap0(a5),M_Gap1(a5)
            00140016
01082              IniDrv60
01083 0582 7c00                   moveq    #0,d6          calculate number of sectors
01084 0584=3c29                   move.w   PD_CYL(a1),d6  get number of cyl's
            0000
01085 0588=0828                   btst     #Side_Bit,DD_FMT(a0) double sided?
            00000000
01086 058e 6702                   beq.s    IniDrv61       no
01087 0590 e34e                   lsl.w    #1,d6          double cyl's
01088              IniDrv61
```

```
01089 0592 5386           subq.l   #1,d6          ignore track 0
01090 0594 cced           mulu.w   M_NoB1+1(a5),d6 we have it almost
           000e
01091 0598 dc6d           add.w    M_NoB0+1(a5),d6 add count from  track 0
           0006
01092 059c 3b46           move.w   d6,M_MaxBlk+1(a5) save on max blocks
           0022
01093 05a0 6100           bsr      Doinit         do the command
           000e
01094 05a4 4cdf           movem.l  (a7)+,d0/d2-d7/a0-a6
           7ffd
01095 05a8 6000           bra      IniDrv95
           0004
01096           IniDrv90
01097 05ac 7200           moveq    #0,d1          clear carry
01098 05ae 4e75 IniDrv95  rts
01099 * 3NN end changes
01100
01101 * Init Drive
01102 *
01103 * 3NN changes
01104           Doinit
01105 05b0 48e7           movem.l  d0/d2-d7/a0-a6,-(a7)
           bffe
01106 05b4=266a           movea.l  V_PORT(a2),a3  point to port
           0000
01107 05b8 157c           move.b   #C$MINI,HD_cmd(a2) init drive params
           000c0904
01108 05be=1029           move.b   PD_DRV(a1),d0
           0000
01109 05c2 e408           lsr.b    #2,d0          make it controller number
01110 05c4 670c           beq.s    Doin10         was 1410
01111 05c6 157c           move.b   #C$MODE,HD_cmd(a2) is CC-80
           00150904
01112 05cc 157c           move.b   #40,V_All(a2)
           00280908
01113           Doin10
01114 * Edition 2 change
01115 05d2=1029           move.b   PD_DRV(a1),d0
           0000
01116 05d6 0200           andi.b   #$03,d0
           0003
01117 05da eb08           lsl.b    #5,d0
01118 05dc 1540           move.b   d0,HD_psn(a2)
           0905
01119 * Edition 2 end change
01120 05e0 4bea           lea      HD_cmd(a2),a5
           0904
01121 05e4 323c           move.w   #6,d1          byte count
           0006
01122 05e8 6100           bsr      NCRcmd
           0182
01123 05ec 6500           bcs      Doinit20
           003e
01124 05f0 4bea           lea      V_IntBuf(a2),a5
           0912
01125 05f4 323c           move.w   #8,d1          byte count
           0008
01126 05f8=1029           move.b   PD_DRV(a1),d0
           0000
01127 05fc e408           lsr.b    #2,d0          make it controller number
```

```
01128 05fe 6704             beq.s    Doin20         was 1410
01129 0600 323c             move.w   #40,d1         byte count
           0028
01130           Doin20
01131 0604 6100             bsr      NCRwr
           02d8
01132 0608 4bea             lea      HD_status(a2),a5
           090a
01133 060c 6100             bsr      NCRstatus
           01d0
01134 0610 4bea             lea      HD_msg(a2),a5
           090b
01135 0614 6100             bsr      NCRmsg
           01e4
01136 * Edition 2 change
01137 0618 082a             btst     #ErrStat,HD_status(a2)
           0001090a
01138 061e 670a             beq.s    Doinit10        ..no error
01139 0620 6100             bsr      senstat
           02f4
01140 0624 4cdf             movem.l  (a7)+,d0/d2-d7/a0-a6
           7ffd
01141 0628 4e75             rts
01142
01143 062a 7200 Doinit10    moveq    #0,d1
01144 062c 4cdf Doinit20    movem.l  (a7)+,d0/d2-d7/a0-a6
           7ffd
01145 0630 4e75             rts
01146 * 3NN end changes
01147
01148
```

```
01150 *
01151 * Read One Sector
01152 *
01153 * Input:  (a1.l) = path descriptor
01154 *         (a2.l) = device static storage
01155 *         (a5.l) = buffer pointer
01156 *         (d2.l) = Logical Sector Number
01157 *
01158 * Exit  : (cc) = Carry set on error else cleared
01159 *         (d1.w) = Error code if error else cleared
01160 *
01161           RdSector
01162 0632 48e7             movem.l  a5,-(sp)
           0004
01163 0636 123c             move.b   #C$RBLK,d1     Read sector(s) cmd
           0008
01164 063a 6100             bsr      SetUp
           00ac
01165 063e=266a             move.l   V_PORT(a2),a3
           0000
01166 0642 323c             move.w   #6,d1          get count
           0006
01167 0646 4bea             lea      HD_cmd(a2),a5  cmd ptr
           0904
01168 064a 6100             bsr      NCRcmd
           0120
01169 * Edition 2 change
01170 064e 653a             bcs.s    RdSec90
01171 0650 323c             move.w   #$100,d1       should calculate number
           0100
01172 0654 2a57             move.l   (sp),a5
01173 0656 103c             move.b   #DevToMem,d0
           0080
01174 065a 6100             bsr      NCRdma
           01d6
01175 * bsr NCRrd
01176 065e 4bea             lea      HD_status(a2),a5
           090a
01177 0662 6100             bsr      NCRstatus
           017a
01178 0666 4bea             lea      HD_msg(a2),a5
           090b
01179 066a 6100             bsr      NCRmsg
           018e
01180 * Edition 2 change
01181 066e 082a             btst     #ErrStat,HD_status(a2)
           0001090a
01182 0674 670a             beq.s    RdSec10
01183 0676 6100             bsr      senstat
           029e
01184 067a 4cdf             movem.l  (sp)+,a5
           2000
01185 067e 4e75             rts
01186 0680 323c RdSec10     move.w   #0,d1
           0000
01187 0684 4cdf             movem.l  (sp)+,a5
           2000
01188 0688 4e75             rts
01189 068a 4cdf RdSec90     movem.l  (sp)+,a5
           2000
01190 068e 4e75             rts
```

```
01191
```

```
01193 *
01194 * Write One Sector
01195 *
01196 * Input:  (a1.l) = path descriptor
01197 *         (a2.l) = device static storage
01198 *         (a5.l) = buffer pointer
01199 *         (d2.l) = Logical Sector Number
01200 *
01201 * Exit  : (cc) = Carry set on error else cleared
01202 *         (d1.w) = Error code if error else cleared
01203 *
01204           WrSector
01205 0690 48e7           movem.l  a5,-(sp)
           0004
01206 * Edition 2 change
01207 * bcs.s WrSec90 error
01208 0694 123c           move.b   #C$WBLK,d1     Write sector(s) cmd
           000a
01209 0698 6100           bsr      SetUp
           004e
01210 069c=266a           move.l   V_PORT(a2),a3
           0000
01211 06a0 323c           move.w   #6,d1          get count
           0006
01212 06a4 4bea           lea      HD_cmd(a2),a5  cmd ptr
           0904
01213 06a8 6100           bsr      NCRcmd
           00c2
01214 * Edition 2 change
01215 06ac 6534           bcs.s    WrSec90
01216 06ae 323c           move.w   #$100,d1
           0100
01217 06b2 2a57           move.l   (sp),a5
01218 06b4 103c           move.b   #MemToDev,d0
           0000
01219 06b8 6100           bsr      NCRdma
           0178
01220 * bsr NCRwr
01221 06bc 4bea           lea      HD_status(a2),a5
           090a
01222 06c0 6100           bsr      NCRstatus
           011c
01223 06c4 4bea           lea      HD_msg(a2),a5
           090b
01224 06c8 6100           bsr      NCRmsg
           0130
01225 * Edition 2 change
01226 06cc 082a           btst     #ErrStat,HD_status(a2)
           0001090a
01227 06d2 670a           beq.s    WrSec10
01228 06d4 6100           bsr      senstat
           0240
01229 06d8 4cdf           movem.l  (sp)+,a5
           2000
01230 06dc 4e75           rts
01231 06de 323c WrSec10   move.w   #0,d1
           0000
01232 06e2 4cdf WrSec90   movem.l  (sp)+,a5
           2000
01233 06e6 4e75           rts
01234
```

```
01236 *
01237 *     Set Up Command Buffer
01238 *
01239 *     This subroutine sets up the command buffer using the
01240 * regs passed by the caller.
01241 *
01242 * Input:
01243 *         (a2) = address of static storage
01244 *         (d1.b) = Command Code
01245 *         (d2.l) = Physical Sector #
01246 *         (d3.b) = Block count / Interleave
01247 * Edition 2 change
01248 *         (d3.b) = no longer needed, no more interleave.
01249 * Returns: Nothing
01250 *
01251            SetUp
01252 06e8 163c              move.b   #1,d3          assume one block to read
         0001
01253 06ec=4a29              tst.b    PD_TYP(a1)     Hard Disk ?
         0000
01254 * 3NN change
01255 06f0 6b48              bmi.s    SetUp25        ..yes
01256 06f2 48e7              movem.l  d6/a5,-(a7)
         0204
01257 06f6=2a69              movea.l  PD_DEV(a1),a5  get device table pointer
         0000
01258 06fa=2a6d              movea.l  V$DESC(a5),a5  get descriptor
         0000
01259 06fe 7c00              moveq    #0,d6
01260 0700=3c2d              move.w   M$DevCon(a5),d6 offset to init bytes
         0000
01261 0704 5446              addq     #2,d6          past byte count
01262 0706 dbc6              adda.l   d6,a5          point to mode bytes
01263 0708=b469              cmp.w    PD_TOS(a1),d2  in Track 0 ?
         0000
01264 070c 651a              bcs.s    SetUp10        ..yes,always single Density
01265 070e=2069              move.l   PD_DTB(a1),a0  get drive table
         0000
01266 0712=0828              btst     #Dens_Bit,DD_FMT(a0) double Density ?
         00010000
01267 0718 670e              beq.s    SetUp10        ..no
01268 071a 0c6d              cmpi.w   #256,M_BlkS0+1(a5) IBM ??
         0100000a
01269 0720 6714              beq.s    SetUp20        no
01270 0722=d469              add.w    PD_TOS(a1),d2  add Track 0 Sectors
         0000
01271 0726 600e              bra.s    SetUp20
01272            SetUp10
01273 0728 0c6d              cmpi.w   #256,M_BlkS0+1(a5) IBM ??
         0100000a
01274 072e 6706              beq.s    SetUp20        no
01275 0730 e38a              lsl.l    #1,d2          double sector number
01276 0732 163c              move.b   #2,d3          two sectors/block
         0002
01277 0736 4cdf SetUp20      movem.l  (a7)+,d6/a5
         2040
01278 * 3NN end changes
01279 073a 2542 SetUp25      move.l   d2,HD_cmd(a2)  buffer the logical sector #
         0904
01280 073e 1541              move.b   d1,HD_cmd(a2)  buffer command
         0904
```

```
01281 0742=1569            move.b   PD_STP(a1),HD_ctrl(a2) Load The Standard Option Byte
           00000909
01282 0748 0c01            cmpi.b   #C$RBLK,d1     is it a read command ?
           0008
01283 074c 6706            beq.s    SetUp30        ..yes
01284 074e 022a            andi.b   #$BF,HD_ctrl(a2) clear 'a' retry bit on other than read
           00bf0909
01285           SetUp30
01286 0754=1229            move.b   PD_DRV(a1),d1  get drive #
           0000
01287 0758 0201            andi.b   #$03,d1        clear higher bits
           0003
01288 075c eb09            lsl.b    #5,d1          adjust drive #
01289 075e 822a            or.b     HD_psn(a2),d1  Or Into MSB's Of Address
           0905
01290 0762 1541            move.b   d1,HD_psn(a2)
           0905
01291 0766 1543            move.b   d3,HD_blk(a2)  Number of blocks/ Interleave
           0908
01292           SetUpEx
01293 076a 4e75            rts                     Return
01294 * 3NN end change
01295
01296
```

```
01298 *
01299 *
01300 * NCRcmd : give the NCR5385 a command
01301 * entry : a5  cmd pointer
01302 *         a3  base address CC74
01303 *         a2  static storage
01304 *         d1.w  byte count
01305 *
01306 076c 3f01 NCRcmd:    move.w   d1,-(sp)        save byte count
01307 076e 422b           clr.b    TFRNCR(a3)      timout = 10000 * 1024 * Clock period NCR
           0139
01308 0772 323c           move.w   #10000,d1       (Which is 100 nsec)
           2710
01309 0776 038b           movep.w  d1,TFRNCR+2(a3) Timout = 1 sec.
           013b
01310 077a 6100 ncmd0     bsr      TestIrq         is there an interrupt pending
           0282
01311 077e 66fa           bne.s    ncmd0           yes reset by reading irq again
01312 * Edition 2 change
01313 *ncmd1 move.b #$04,COMNCR(a3) 'Soft reset ??'
01314 0780-1029           move.b   PD_DRV(a1),d0   get drive nr.
           0000
01315 0784 e408           lsr.b    #2,d0           make it controller number
01316 0786 1740           move.b   d0,DESIDNCR(a3) set target id
           0127
01317 078a 177c           move.b   #$09,COMNCR(a3) select XEBEC
           00090123
01318 0790 6100           bsr      WaitIrq         bit #0 'function complete' should be set
           0288
01319 0794 0801           btst     #0,d1           is it set ?
           0000
01320 0798 6738           beq.s    nccmd9          ..no, not ready
01321 079a 6100           bsr      WaitIrq         bit #1 'bus service' should be set
           027e
01322 079e 321f           move.w   (sp)+,d1        restore byte count
01323 07a0 422b           clr.b    TFRNCR(a3)
           0139
01324 07a4 422b           clr.b    TFRNCR+2(a3)
           013b
01325 07a8 1741           move.b   d1,TFRNCR+4(a3) init transfer count
           013d
01326 07ac 6100 ncmd2     bsr      TestIrq         is there an interrupt pending
           0250
01327 07b0 66fa           bne.s    ncmd2           yes reset by reading irq again
01328 07b2 177c           move.b   #$14,COMNCR(a3) transfer info cmd
           00140123
01329 07b8 102b ncmd3     move.b   AUXNCR(a3),d0
           0129
01330 07bc 0800           btst     #7,d0
           0007
01331 07c0 66f6           bne.s    ncmd3           wait for data reg full = 0
01332 07c2 0800           btst     #1,d0
           0001
01333 07c6 6606           bne.s    ncmd4           transfer count zero?
01334 07c8 175d           move.b   (a5)+,DATNCR(a3) no, do another byte
           0121
01335 07cc 60ea           bra.s    ncmd3
01336 07ce 7200 ncmd4     moveq    #0,d1
01337 07d0 4e75           rts
01338                nccmd9
01339 07d2 321f           move.w   (sp)+,d1        restore stack
```

```
01340 07d4=323c            move.w   #E$NotRdy,d1   flag unit not ready
           0000
01341 07d8=003c            ori      #Carry,ccr
           0000
01342 07dc 4e75            rts
01343
01344
```

```
01346 *
01347 * NCRstatus : get status byte
01348 * entry : a5  status ptr
01349 *         a3  base address
01350 *         a2  static storage
01351 *
01352 07de 177c NCRstatus: move.b   #1,TFRNCR+4(a3) transfer count is 1
            0001013d
01353 07e4 177c Nstatus1   move.b   #$14,COMNCR(a3) give transfer command
            00140123
01354 07ea 6100            bsr      TestIrq         testif irq pending
            0212
01355 07ee 0800            btst     #7,d0           test if data full ncr
            0007
01356 07f2 67f0            beq.s    Nstatus1        No data received
01357 07f4 1aab            move.b   DATNCR(a3),(a5) read status
            0121
01358 07f8 4e75            rts
01359
01360
```

```
01362 *
01363 * NCRmsg : get message byte
01364 * entry : a5  message ptr
01365 *         a3  base address
01366 *         a2  static storage
01367 *
01368 07fa 177c NCRmsg:    move.b   #1,TFRNCR+4(a3) transfer count is 1
           0001013d
01369 0800 102b NCRmsg1    move.b   COMNCR(a3),d0  test if command register is 00
           0123
01370 0804 6706            beq.s    NCRmsg2
01371 0806 177c            move.b   #$04,COMNCR(a3) message accepted cmd
           00040123
01372 080c 177c NCRmsg2    move.b   #$14,COMNCR(a3) transfer info cmd
           00140123
01373 0812 6100            bsr      TestIrq        Is there an interrupt
           01ea
01374 0816 0800            btst     #7,d0          test if data full ncr
           0007
01375 081a 67e4            beq.s    NCRmsg1        No data received, try to give command again
01376 081c 1aab            move.b   DATNCR(a3),(a5)
           0121
01377 * Edition 2 change
01378 0820 177c            move.b   #$04,COMNCR(a3) message accepted cmd
           00040123
01379           NCRmsg3
01380 0826 6100            bsr      WaitIrq        SCSI should be disconnected
           01f2
01381 082a 0c01            cmpi.b   #$04,d1
           0004
01382 082e 66f6            bne.s    NCRmsg3
01383 0830 4e75            rts
01384 * Edition 2 end change
01385
01386
```

```
01388 *
01389 * NCRdma : get data bytes
01390 * entry : a5 data pointer
01391 *         a3 base address
01392 *         a1 path descriptor
01393 *         a2 static storage
01394 *         d0.b transfer direction
01395 *         d1.w count
01396 *
01397           NCRdma
01398 0832 038b               movep.w  d1,TFRNCR+2(a3) store tansfer count
           013b
01399 0836 0000               ori.b    #(ByteSize+ChainDis+ReqInit),d0 Output control register
           0002
01400 083a 1740               move.b   d0,OCR+CHNL0(a3)
           0005
01401 083e 274d               move.l   a5,MAR+CHNL0(a3) store memory address
           000c
01402 0842 3741               move.w   d1,MTC+CHNL0(a3) store number of bytes to transfer
           000a
01403 0846 6100 dma05         bsr      TestIrq        clear hanging interrupts
           01b6
01404 084a 6600               bne      dma05
           fffa
01405 084e 177c dma06         move.b   #$94,COMNCR(a3) give tranfer info command (with DMA) to NCR chip
           00940123
01406 *    move.w #100,d0
01407 *dma07 dbra d0,dma07
01408 0854 6100               bsr      TestIrq        clear possible interrupt
           01a8
01409 0858 6600               bne      dma06
           fff4
01410 085c 177c               move.b   #$ff,CSR+CHNL0(a3) clear status register
           00ff0000
01411 * Edition 2 change
01412 0862=356a               move.w   V_BUSY(a2),V_WAKE(a2) init sleep
           00000000
01413 0868 177c               move.b   #(IntrptEn+StartOp),CHCR+CHNL0(a3) start dma, enable irq
           00880007
01414 086e 7000 dma10         moveq    #0,d0          sleep forever
01415 0870=4e40               os9      F$Sleep        wait until V_WAKE
           0000
01416 0874=4a6a               tst.w    V_WAKE(a2)     valid?
           0000
01417 0878 66f4               bne.s    dma10
01418 087a 082b               btst     #Error_B,CSR+CHNL0(a3) error occured
           00040000
01419 0880 660c               bne.s    dma90          yes
01420 0882 177c               move.b   #$ff,CSR+CHNL0(a3) clear irq cause
           00ff0000
01421 0888 323c               move.w   #0,d1
           0000
01422 088c 4e75               rts
01423 088e 003c dma90         ori.b    #1,ccr
           0001
01424 0892 4e75               rts
01425
```

```
01427 *
01428 * NCRrd : get data bytes
01429 * entry : a5 data pointer
01430 *         a3 base address
01431 *         a2 static storage
01432 *         d1.w count
01433 *
01434 0894 038b NCRrd      movep.w  d1,TFRNCR+2(a3) store transfer count
           013b
01435 * Edition 2 change
01436 0898 6100 datrd0     bsr      WaitIrq        Clear pending interrupts
           0180
01437 *           bne.s    datrd0
01438 089c 177c datrd1     move.b   #$14,COMNCR(a3)
           00140123
01439 08a2 6100 datrd2     bsr      TestIrq        Is there an Irq?
           015a
01440 08a6 6700            beq      ncrrd40        ..no
           0010
01441 08aa 1600            move.b   d0,d3          save aux. status
01442 08ac 0203            andi.b   #$38,d3        isolate msg c/d and i/o
           0038
01443 08b0 0c03            cmpi.b   #$18,d3        status phase ?
           0018
01444 08b4 6700            beq      ncrrd90        ..yes, exit
           0026
01445           ncrrd40
01446 08b8 0800            btst     #1,d0          test if last byte
           0001
01447 08bc 6610            bne.s    datrd3         was last
01448 08be 4a01            tst.b    d1             See if command still accepted
01449 08c0 66da            bne.s    datrd1         Interrupt occured, restore command
01450 08c2 0800            btst     #7,d0          test if data register full
           0007
01451 08c6 67da            beq.s    datrd2
01452 08c8 1aeb            move.b   DATNCR(a3),(a5)+
           0121
01453 08cc 60d4            bra.s    datrd2
01454 08ce 1aeb datrd3     move.b   DATNCR(a3),(a5)+ move last byte too
           0121
01455 08d2 102b            move.b   AUXNCR(a3),d0
           0129
01456 08d6 0800            btst     #7,d0
           0007
01457 08da 66f2            bne.s    datrd3
01458 08dc 4e75 ncrrd90    rts
01459
```

```
01461 *
01462 * NCRwr : send data bytes
01463 * entry : a5 data pointer
01464 *         a3 base address
01465 *         a2 static storage
01466 *         d1.w count
01467 *
01468 08de 038b NCRwr      movep.w  d1,TFRNCR+2(a3) store transfer count
          013b
01469 * Edition 2 change
01470 08e2 6100 datwr0     bsr      WaitIrq         Is there an Irq pending
          0136
01471 *          bne.s    datwr0          Irq register not clear
01472 08e6 177c datwr1     move.b   #$14,COMNCR(a3)
          00140123
01473 08ec 6100 datwr2     bsr      TestIrq         Is there an Irq?
          0110
01474 08f0 670c            beq.s    ncrwr40         ..no
01475 08f2 1600            move.b   d0,d3           save aux. status
01476 08f4 0203            andi.b   #$38,d3         isolate msg c/d and i/o
          0038
01477 08f8 0c03            cmpi.b   #$18,d3         status phase ?
          0018
01478 08fc 6716            beq.s    datwr3          ..yes, exit
01479           ncrwr40
01480 08fe 0800            btst     #1,d0           test if last byte
          0001
01481 0902 6610            bne.s    datwr3          was last
01482 0904 4a01            tst.b    d1              See if command still accepted
01483 0906 66de            bne.s    datwr1          Interrupt, restore command
01484 0908 0800            btst     #7,d0           test if data register empty
          0007
01485 090c 66de            bne.s    datwr2
01486 090e 175d            move.b   (a5)+,DATNCR(a3)
          0121
01487 0912 60d8            bra.s    datwr2
01488 0914 4e75 datwr3     rts
01489
```

```
01491 *
01492 * Sense Status command
01493 *       Should be called if another command returns with an error
01494 * Entry:  (a2) static storage
01495 *           HD_psn, HD_blk, and HD_cntl should not have changed
01496 *
01497 0916 157c senstat    move.b   #C$RDET,HD_cmd(a2) Set command byte
           00030904
01498 091c 4bea            lea      HD_cmd(a2),a5
           0904
01499 0920 323c            move.w   #6,d1
           0006
01500 0924 6100            bsr      NCRcmd         Execute command
           fe46
01501 * Edition 2 change
01502 0928 6568            bcs.s    sens95
01503 092a 323c            move.w   #4,d1
           0004
01504 092e 4bea            lea      HD_sense(a2),a5
           090c
01505 0932 6100            bsr      NCRrd
           ff60
01506 0936 4bea            lea      HD_status(a2),a5
           090a
01507 093a 6100            bsr      NCRstatus
           fea2
01508 093e 4bea            lea      HD_msg(a2),a5
           090b
01509 0942 6100            bsr      NCRmsg
           feb6
01510 * Edition 2 change
01511 0946 082a            btst     #ErrStat,HD_status(a2)
           0001090a
01512 094c 663c            bne.s    sens90
01513 094e 102a            move.b   HD_sense(a2),d0 Get The Error Code
           090c
01514 0952 0200            andi.b   #$3F,d0        Mask Out Address Valid
           003f
01515 0956 b03c            cmp.b    #$18,d0        correctable error?
           0018
01516 095a 672c            beq.s    sens40         branch if so
01517 * Edition 2 changes
01518 095c=1229            move.b   PD_DRV(a1),d1  get drive nr.
           0000
01519 0960 e409            lsr.b    #2,d1          make it controller nr
01520 0962 6606            bne.s    sens05         it is CC-80 controller
01521 0964 4bfa            lea      ErrorTbl(pcr),a5 Point At The Error Table
           002e
01522 0968 600e            bra.s    sens20
01523                sens05
01524 096a 4bfa            lea      Error2Tbl(pcr),a5
           0058
01525 096e 6008            bra.s    sens20
01526 * Edition 2 end changes
01527 0970 b001 sens10     cmp.b    d1,d0          do error codes match?
01528 0972 6708            beq.s    sens30         branch if so
01529 0974 4bed            lea      2(a5),a5       skip to next entry
           0002
01530 0978 1215 sens20     move.b   (a5),d1        get controller error #
01531 097a 6af4            bpl.s    sens10         branch if not end of table
01532 097c 122d sens30     move.b   1(a5),d1       move errorcode to d1
```

```
          0001
01533 0980 0241            andi.w   #$ff,d1        clear msb of error word
          00ff
01534 0984=003c            ori      #Carry,ccr     flag error
          0000
01535 0988 4e75 sens40     rts
01536
01537 098a=323c sens90     move.w   #E$NotRdy,d1   What to do if Senstat returns with error?
          0000
01538 098e=003c            ori.b    #Carry,ccr
          0000
01539 0992 4e75 sens95     rts
01540
```

```
***************
* Error translation table xebeq 1410
*  The first byte being the SASI 1410 controller code & the
*  second byte is the corresponding OS-9 error code.

 00000994 ErrorTbl   equ      *
0994=0000            dc.b     $00,E$NotRdy
0996=0100            dc.b     $01,E$NotRdy
0998=0200            dc.b     $02,E$Seek     no seek complete from dsk drv
099a=0300            dc.b     $03,E$WP       write fault from dsk drv
099c=0400            dc.b     $04,E$NotRdy   drive not ready
099e=0600            dc.b     $06,E$Seek     track 00 not found
09a0=0800            dc.b     $08,E$Seek     drive still seeking
09a2=1000            dc.b     $10,E$Read     ID field read error
09a4=1100            dc.b     $11,E$Read     uncorrectable data error
09a6=1200            dc.b     $12,E$Seek     address mark not found
09a8=1400            dc.b     $14,E$Seek     not found
09aa=1500            dc.b     $15,E$Seek     seek error
09ac=1900            dc.b     $19,E$Seek     bad track flag detected
09ae=1a00            dc.b     $1A,E$BTyp     format error
09b0=1c00            dc.b     $1C,E$Seek     illegal alt. tk access
09b2=1d00            dc.b     $1D,E$Seek     bad alternate track
09b4=1e00            dc.b     $1E,E$Seek     not an alternate track
09b6=1f00            dc.b     $1F,E$Seek     alt. tk same as bad
09b8=2000            dc.b     $20,E$Unit     invalid command
09ba=2100            dc.b     $21,E$Sect     illegal disk address
09bc=3000            dc.b     $30,E$DevBsy   ram diagnostic failure
09be=3100            dc.b     $31,E$DevBsy   program memory checksum err
09c0=3200            dc.b     $32,E$DevBsy   ecc diagnostic failure
09c2=ff00            dc.b     $FF,E$WP       flag for end of table

** 2NN start changes changes
** 3NN changes
****************
* Error translation table CC-80
*  The first byte being the SASI CC-80 controller code & the
*  second byte is the corresponding OS-9 error code.

 000009c4 Error2Tbl  equ      *
09c4=0000            dc.b     $00,E$NotRdy
09c6=0200            dc.b     $02,E$Seek     Seek timeout
09c8=0400            dc.b     $04,E$NotRdy   drive not ready
09ca=0600            dc.b     $06,E$Seek     track 00 not found
09cc=0700            dc.b     $07,E$NotRdy   Door open
09ce=0800            dc.b     $08,E$NotRdy   No head loaded
09d0=1000            dc.b     $10,E$Read     ID field read error
09d2=1100            dc.b     $11,E$Read     DataEr
09d4=1200            dc.b     $12,E$WP       write fault from dsk drv
09d6=1300            dc.b     $13,E$Read     ID missing
09d8=1400            dc.b     $14,E$Seek     sector not found
09da=1500            dc.b     $15,E$Seek     seek error
09dc=1600            dc.b     $16,E$NotRdy   fault
09de=1700            dc.b     $17,E$WP       read only
09e0=1900            dc.b     $19,E$BTyp     format error
09e2=1a00            dc.b     $1A,E$Read     wrong data mark
09e4=1b00            dc.b     $1B,E$Read     Transfer length error
09e6=1d00            dc.b     $1D,E$Read     Lost data in fdc
09e8=1e00            dc.b     $1E,E$Read     Data CRC error
09ea=1a00            dc.b     $1A,E$Read     wrong data mark
09ec=1f00            dc.b     $1F,E$NotRdy   Fdc busy error
09ee=2000            dc.b     $20,E$BTyp     invalid command
```

```
01603 09f0=2100            dc.b     $21,E$Sect      illegal block address
01604 09f2=2200            dc.b     $22,E$BTyp      invalid drive init.
01605 09f4=2300            dc.b     $23,E$BTyp      invalid drive number
01606 09f6=3000            dc.b     $30,E$DevBsy    ram diagnostic failure
01607 09f8=3100            dc.b     $31,E$DevBsy    program memory checksum err
01608 09fa=3200            dc.b     $32,E$DevBsy    ecc diagnostic failure
01609 09fc=ff00            dc.b     $FF,E$WP        flag for end of table
01610
01611 ** 2NN And 3NN end changes
01612
01613
01614
```

```
01616 *
01617 * TestIrq
01618 * Input:
01619 *       (a3) = base address of cc-74
01620 *
01621                TestIrq
01622 09fe 102b               move.b   CSR(a3),d0      get status register dma channel 0
           0000
01623 0a02 0800               btst     #0,d0           irq NCR occurred ?
           0000
01624 0a06 660a               bne.s    TstIrq10        ..no, just read auxilary status
01625 0a08 102b               move.b   AUXNCR(a3),d0   get auxiliary register NCR
           0129
01626 0a0c 122b               move.b   IRQNCR(a3),d1   get interrupt register NCR
           012d
01627 0a10 4e75               rts
01628                TstIrq10
01629 0a12 102b               move.b   AUXNCR(a3),d0   get auxiliary register NCR
           0129
01630 0a16 7200               moveq    #0,d1           no interrupt
01631 0a18 4e75               rts
01632
01633
01634 *
01635 * WaitIrq
01636 *
01637 * Poll IRQ line NCR chip via DMA controller
01638 * Input:
01639 *       (a3) = base address cc-74
01640 *
01641 0a1a 102b WaitIrq       move.b   CSR(a3),d0      get status register dma channel 0
           0000
01642 0a1e 0800               btst     #0,d0           irq NCR occurred
           0000
01643 0a22 66f6               bne.s    WaitIrq         no, try again
01644 0a24 102b               move.b   AUXNCR(a3),d0   get auxiliary register NCR
           0129
01645 0a28 122b               move.b   IRQNCR(a3),d1   get interrup register NCR
           012d
01646 0a2c 4e75               rts
01647
01648  00000a2e               ends
```

APPENDIX K

DATA SHEET 68450 DMAC

HITACHI MICROCOMPUTER SYSTEM
HD68450-4, HD68450-6, HD68450-8
DMAC (Direct Memory Access Controller)

HITACHI

DIRECT MEMORY ACCESS CONTROLLER

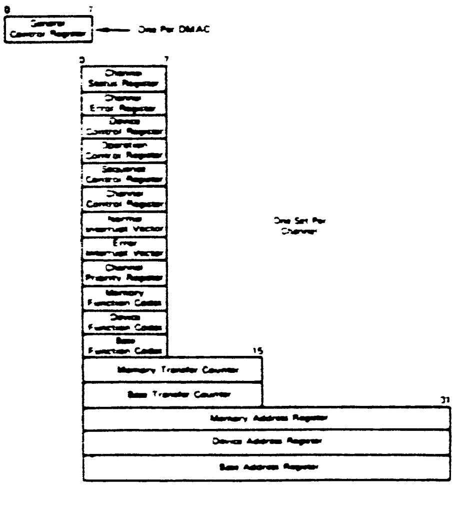
Microprocessor implemented systems are becoming increasingly complex, particularly with the advent of high-performance 16-bit MPU devices with large memory addressing capability. In order to maintain high throughput, large blocks of data must be moved within these systems in a quick, efficient manner with minimum intervention by the MPU itself.

The HD68450 Direct Memory Access Controller (DMAC) is designed specifically to complement the performance and architectural capabilities of the HD68000 MPU by providing the following features:

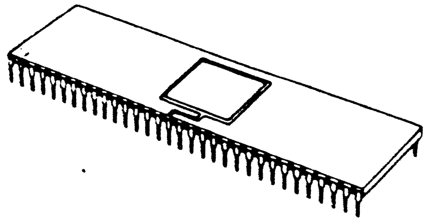
- HMCS68000 Bus Compatible
- 4 independent DMA Channels
- Memory-to-Memory, Memory-to-Device, Device-to-Memory Transfers
- MMU Compatible
- Array-Chained and Linked-Array-Chained Operations
- On-Chip Registers that allow Complete Software Control by the System MPU
- Interface Lines that Provide for Requesting, Acknowledging, and Incidental Control of the Peripheral Devices
- Transfers to/from HMCS68000 or HMCS6800 Peripherals
- Variable System Bus Bandwidth Utilization
- Programmable Channel Prioritization
- 2 Vectored interrupts for each Channel
- Auto-Request and External-Request Transfer Modes
- Up to 4 Megabytes/Second Transfer Rates
- +5 Volt Operation

The DMAC functions by transferring a series of operands (data) between memory and device; operand sizes can be byte, word, or long word. A block is a sequence of operations; the number of operands in a block is determined by a transfer count. A single-channel operation may involve the transfer of several blocks of data between memory and device.

■ DMAC ACCESSIBLE REGISTERS

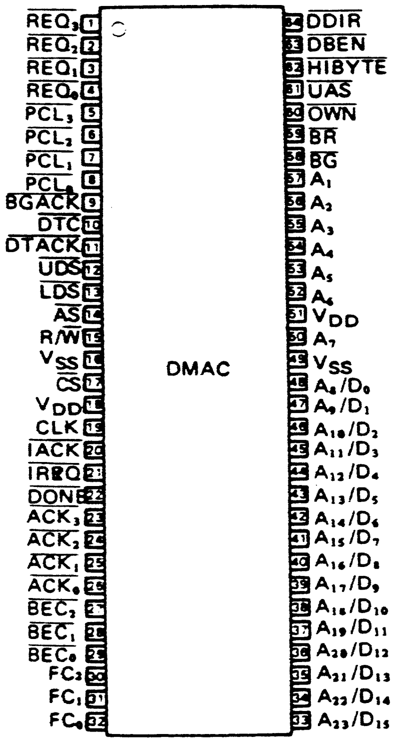


HD68450-4, HD68450-6, HD68450-8



(DC-64)

■ PIN ARRANGEMENT



(Top View)

■ ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Value	Unit
Supply Voltage	V_{CC}*	-0.3 ~ +7.0	V
Input Voltage	V_{in}*	-0.3 ~ +7.0	V
Operating Temperature Range	T_{opr}	0 ~ +70	°C
Storage Temperature	T_{stg}	-55 ~ +150	°C

* With respect to V_{SS} (SYSTEM GND)

(NOTE) Permanent LSI damage may occur if maximum ratings are exceeded. Normal operation should be under recommended operating conditions. If these conditions are exceeded, it could affect reliability of LSI.

■ RECOMMENDED OPERATING CONDITIONS

Item	Symbol	min	typ	max	Unit
Supply Voltage	V_{CC}*	4.75	5.0	5.25	V
Input Voltage	V_{IH}*	2.0	–	V_{CC}	V
	V_{IL}*	-0.3	–	0.8	V
Operating Temperature	T_{opr}	0	25	70	°C

* With respect to V_{SS} (SYSTEM GND)

■ ELECTRICAL CHARACTERISTICS

● DC CHARACTERISTICS (V_{CC} = 5V ±5%, V_{SS} = 0V, Ta = 0 ~ +70°C, unless otherwise noted.)

Item		Symbol	Test Condition	min	typ	max	Unit
Input "High" Voltage		V_{IH}		2.0	–	V_{CC}	V
Input "Low" Voltage		V_{IL}		V_{SS}-0.3	–	0.8	V
Input Leakage Current	$\overline{CS}$, $\overline{IACK}$, $\overline{BG}$, CLK, $\overline{BEC_0}$ ~ $\overline{BEC_2}$ $\overline{REQ_0}$ ~ $\overline{REQ_3}$	I_{in}		–	–	10	μA
Three-State (Off State) Input Current	A_1 ~ A_7, D_0 ~ D_{15}/A_8 ~ A_{23}, $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, $R/\overline{W}$, $\overline{UAS}$, $\overline{DTACK}$, $\overline{BGACK}$, $\overline{OWN}$, $\overline{DTC}$, $\overline{HIBYTE}$, $\overline{DDIR}$, $\overline{DBEN}$, FC_0 ~ FC_2	I_{TSI}		–	–	10	μA
Open Drain (Off State) Input Current	$\overline{IREQ}$, $\overline{DONE}$	I_{ODI}		–	–	20	μA
Output "High" Voltage	A_1 ~ A_7, D_0 ~ D_{15}/A_8 ~ A_{23}, $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, $R/\overline{W}$, $\overline{UAS}$, $\overline{DTACK}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{OWN}$, $\overline{DTC}$, $\overline{HIBYTE}$, $\overline{DDIR}$, $\overline{DBEN}$, $\overline{ACK_0}$ ~ $\overline{ACK_3}$, $\overline{PCL_0}$ ~ $\overline{PCL_3}$, FC_0 ~ FC_2	V_{OH}	I_{OH} = -400μA	2.4	–	–	V
Output "Low" Voltage	A_1 ~ A_7, FC_0 ~ FC_2	V_{OL}	I_{OL} = 3.2mA	–	–	0.5	V
	D_0 ~ D_{15}/A_8 ~ A_{23}, $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$, $R/\overline{W}$, $\overline{DTACK}$, $\overline{BR}$. $\overline{OWN}$, $\overline{DTC}$, $\overline{HIBYTE}$, $\overline{DDIR}$, $\overline{DBEN}$, $\overline{ACK_0}$ ~ $\overline{ACK_3}$, $\overline{UAS}$, $\overline{PCL_0}$ ~ $\overline{PCL_3}$, $\overline{BGACK}$,	V_{OL}	I_{OL} = 5.3mA	–	–	0.5	
	$\overline{IREQ}$, $\overline{DONE}$	V_{OL}	I_{OL} = 8.9mA	–	–	0.5	
Power Dissination		P_D	f = 8MHz	–	1.0	1.75	W
Capacitance		C_{in}	V_{in} = 0V, Ta = 25°C, f = 1MHz	–	–	15	pF

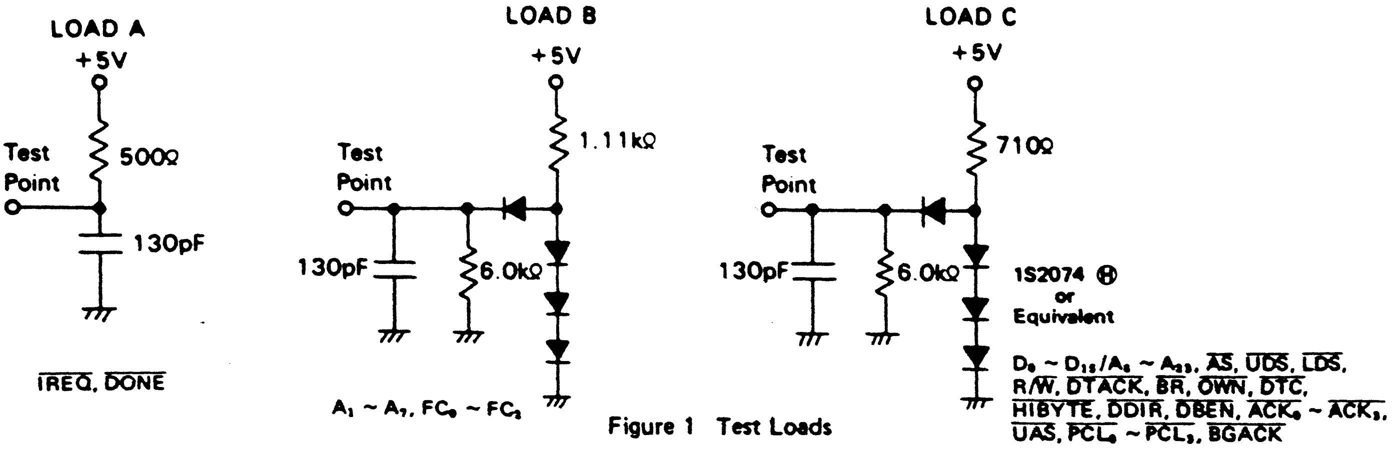

LOAD A
+5V
Test Point
500Ω
130pF
$\overline{IREQ}$, $\overline{DONE}$
LOAD B
+5V
1.11kΩ
Test Point
130pF
6.0kΩ
$A_1 \sim A_7$, $FC_0 \sim FC_2$
LOAD C
+5V
710Ω
Test Point
130pF
6.0kΩ
1S2074 Ⓗ or Equivalent
$D_0 \sim D_{15}/A_8 \sim A_{23}$, $\overline{AS}$, $\overline{UDS}$, $\overline{LDS}$,
$R/\overline{W}$, $\overline{DTACK}$, $\overline{BR}$, $\overline{OWN}$, $\overline{DTC}$,
$\overline{HIBYTE}$, $\overline{DDIR}$, $\overline{DBEN}$, $\overline{ACK_0} \sim \overline{ACK_3}$,
$\overline{UAS}$, $\overline{PCL_0} \sim \overline{PCL_3}$, $\overline{BGACK}$

Figure 1 Test Loads

- **AC ELECTRICAL SPECIFICATION (V_{CC} = 5.0V ±5%, V_{SS} = 0V, T_a = 0 ~ +70°C)**

No.	Item	Symbol	Test Condition	4 MHz		6 MHz		8 MHz		Unit
				min	max	min	max	min	max	
	Frequency of Operation	f		2	4	2	6	2	8	MHz
1	Clock Period	t_{cyc}		250	500	167	500	125	500	ns
2	Clock Width "Low"	t_{CL}		115	250	75	250	55	250	ns
3	Clock Width "High"	t_{CH}		115	250	75	250	55	250	ns
4	Clock Fall Time	t_{Cf}		–	10	–	10	–	10	ns
5	Clock Rise Time	t_{Cr}		–	10	–	10	–	10	ns
6	Asynchronous Input Setup Time	t_{ASI}		30	–	25	–	20	–	ns
7	Data In to $\overline{DS}$ In "Low"	t_{DIDSL}		0	–	0	–	0	–	ns
8	Data In to Clock "Low" (Setup Time)	t_{DICL}		30	–	25	–	15	–	ns
9	$\overline{DS}$ In "High" to Data In Valid	t_{DSHDI}		0	–	0	–	0	–	ns
10	Clock "High" to $\overline{DDIR}$ "High" Impedance Off	t_{CHDRZO}		–	120	–	100	–	80	ns
11	Clock "High" to $\overline{DBEN}$ "High" Impedance Off	t_{CHDBZO}		–	120	–	100	–	80	ns
12	Clock "High" to $\overline{DDIR}$ "Low" (MPU Write)	t_{CHDRLM}		–	90	–	80	–	70	ns
13	Clock "High" to $\overline{DDIR}$ "High" (MPU Write)	t_{CHDRHM}		–	90	–	80	–	70	ns
14	Clock "Low" to $\overline{DBEN}$ "Low (MPU Cycle)	t_{CLDBLM}		–	90	–	80	–	70	ns
15	Clock "Low" to $\overline{DBEN}$ "High" (MPU Cycle)	t_{CLDBHM}		–	90	–	80	–	70	ns
16	$\overline{DS}$ In "High" to $\overline{DDIR}$ "High" Impedance	t_{DSHDRZ}		–	120	–	100	–	80	ns
17	$\overline{DS}$ In "High" to $\overline{DBEN}$ "High" Impedance	t_{DSHDBZ}		–	120	–	100	–	80	ns
18	Clock "High" to Data Out Valid (MPU Read)	t_{CHDVM}		–	140	–	120	–	100	ns
19	$\overline{DS}$ In "High" to Data "High" Impedance	t_{DSHDZ}	Fig. 2 ~ 6	–	160	–	140	–	120	ns
20	Clock "Low" to $\overline{DTACK}$ "Low"	t_{CLDTL}		–	90	–	80	–	70	ns
21	$\overline{DS}$ In "High" to $\overline{DTACK}$ "High"	t_{DSHDTH}		–	90	–	80	–	70	ns
22	$\overline{DS}$ In "High" to $\overline{DTACK}$ "High" Impedance	t_{DSHDTZ}		–	220	–	200	–	180	ns
23	$\overline{DTACK}$ "Low" to $\overline{DS}$ In "High"	t_{DTLDSH}		0	–	0	–	0	–	ns
24	$\overline{REQ}$ Width "Low"	t_{REQL}		2.0	–	2.0	–	2.0	–	Clk. Per.
25	Clock "High" to $\overline{BR}$ "Low"	t_{CHBRL}		–	90	–	80	–	70	ns
26	Clock "High" to $\overline{BR}$ "High"	t_{CHBRH}		–	90	–	80	–	70	ns
27	$\overline{BR}$ "Low" to $\overline{BG}$ "Low"	t_{BRLBGL}		0	–	0	–	0	–	ns
28	$\overline{BR}$ "Low" to MPU Cycle End ($\overline{AS}$ In "High")	t_{BRLASH}		0	–	0	–	0	–	ns
29	MPU Cycle End ($\overline{AS}$ In "High") to $\overline{BGACK}$ "Low"	t_{ASHBL}		4.5	5.5	4.5	5.5	4.5	5.5	Clk. Per.
30	Clock "High" to $\overline{BGACK}$ "Low"	t_{CHBL}		–	90	–	80	–	70	ns
31	Clock "High" to $\overline{BGACK}$ "High"	t_{CHBH}		–	90	–	80	–	70	ns
32	$\overline{REQ}$ "Low" to $\overline{BGACK}$ "Low"	t_{REQLBL}		12.0	–	12.0	–	12.0	–	Clk. Per.
33	Clock "Low" to $\overline{BGACK}$ "High" Impedance	t_{CLBZ}		–	120	–	100	–	80	ns
34	Clock "High" to Address/FC Valid	t_{CHAV}		–	140	–	120	–	120	ns
35	Clock "High" to Address/FC/Data "High" Impedance	t_{CHAZx}		–	120	–	100	–	80	ns
36	Clock "High" to Address/FC/ Data Invalid	t_{CHAZn}		0	–	0	–	0	–	ns
37	Clock "Low" to Address "High" Impedance	t_{CLAZ}		–	120	–	100	–	80	ns
38	$\overline{UAS}$ "High" to Address Invalid	t_{UHAI}		50	–	40	–	30	–	ns

(to be continued)

No.	Item	Symbol	Test Condition	4 MHz		6 MHz		8 MHz		Unit
				min	max	min	max	min	max	
39	$\overline{\mathrm{AS}}$, $\overline{\mathrm{DS}}$ "High" to Address/FC/Data Invalid	t_{SHAZ}	Fig. 2 ~ 6	50	–	40	–	30	–	ns
40	Address/FC Valid to $\overline{\mathrm{AS}}$, $\overline{\mathrm{DS}}$ "Low" (Read)	t_{AVSL}		50	–	40	–	30	–	ns
41	Clock "High" to $\overline{\mathrm{UAS}}$ "Low"	t_{CHUL}		–	90	–	80	–	70	ns
42	Clock "High" to $\overline{\mathrm{UAS}}$ "High"	t_{CHUH}		–	90	–	80	–	70	ns
43	Clock "Low" to $\overline{\mathrm{UAS}}$ "High" Impedance	t_{CLUZ}		–	120	–	100	–	80	ns
44	Clock "High" to $\overline{\mathrm{AS}}$, $\overline{\mathrm{DS}}$ "Low"	t_{CHSL}		–	80	–	70	–	60	ns
45	Clock "Low" to $\overline{\mathrm{AS}}$, $\overline{\mathrm{DS}}$ "High"	t_{CLSH}		–	90	–	80	–	70	ns
46	Clock "Low" to $\overline{\mathrm{AS}}$, $\overline{\mathrm{DS}}$ "High" Impedance	t_{CLSZ}		–	120	–	100	–	80	ns
47	Clock "Low" to $\overline{\mathrm{DS}}$ "Low" (Write)	t_{CLDSL}		–	80	–	70	–	60	ns
48	$\overline{\mathrm{AS}}$ Width "Low"	t_{ASL}		545	–	350	–	255	–	ns
49	$\overline{\mathrm{DS}}$ Width "Low"	t_{DSL}		420	–	265	–	190	–	ns
50	$\overline{\mathrm{DS}}$ "High" to R/$\overline{\mathrm{W}}$ "High"	t_{SHRH}		60	–	50	–	40	–	ns
51	Clock "High" to R/$\overline{\mathrm{W}}$ "Low"	t_{CHRL}		–	90	–	80	–	70	ns
52	Clock "High" to R/$\overline{\mathrm{W}}$ "High"	t_{CHRH}		–	90	–	80	–	70	ns
53	Clock "Low" to R/$\overline{\mathrm{W}}$ "High" Impedance	t_{CLRZ}		–	120	–	100	–	80	ns
54	Address/FC Valid to R/$\overline{\mathrm{W}}$ "Low"	t_{AVRL}		110	–	50	–	25	–	ns
55	R/$\overline{\mathrm{W}}$ "Low" to $\overline{\mathrm{DS}}$ "Low" (Write)	t_{RLSL}		285	–	170	–	120	–	ns
56	Clock "Low" to $\overline{\mathrm{OWN}}$ "Low"	t_{CLOL}		–	90	–	80	–	70	ns
57	Clock "Low" to $\overline{\mathrm{OWN}}$ "High"	t_{CLOH}		–	90	–	80	–	70	ns
58	Clock "High" to $\overline{\mathrm{OWN}}$ "High" Impedance	t_{CHOZ}		–	120	–	100	–	80	ns
59	Clock "High to $\overline{\mathrm{DDIR}}$ "Low" (Read)	t_{CHDRL}		–	90	–	80	–	70	ns
60	Clock "High" to $\overline{\mathrm{DDIR}}$ "High"	t_{CHDRH}		–	90	–	80	–	70	ns
61	Clock "Low" to $\overline{\mathrm{DDIR}}$ "High" Impedance	t_{CLDRZ}		–	120	–	100	–	80	ns
62	Clock "Low" to $\overline{\mathrm{DBEN}}$ "Low"	t_{CLDBL}		–	90	–	80	–	70	ns
63	Clock "Low" to $\overline{\mathrm{DBEN}}$ "High"	t_{CLDBH}		–	90	–	80	–	70	ns
64	Clock "Low" to $\overline{\mathrm{DBEN}}$ "High" Impedance	t_{CLDBZ}		–	120	–	100	–	80	ns
65	Clock "High" to $\overline{\mathrm{HIBYTE}}$ "Low"	t_{CHHIL}		–	90	–	80	–	70	ns
66	Clock "High" to $\overline{\mathrm{HIBYTE}}$ "High"	t_{CHHIH}		–	90	–	80	–	70	ns
67	Clock "Low" to $\overline{\mathrm{HIBYTE}}$ "Low"	t_{CLHIL}		–	90	–	80	–	70	ns
68	Clock "Low" to $\overline{\mathrm{HIBYTE}}$ "High" Impedance	t_{CLHIZ}		–	120	–	100	–	80	ns
69	Clock "High" to $\overline{\mathrm{ACK}}$ "Low"	t_{CHACL}		–	90	–	80	–	70	ns
70	Clock "High" to $\overline{\mathrm{ACK}}$ "High"	t_{CHACH}		–	90	–	80	–	70	ns
71	Clock "Low" to $\overline{\mathrm{ACK}}$ "Low"	t_{CLACL}		–	90	–	80	–	70	ns
75	Clock "High" to $\overline{\mathrm{DTC}}$ "Low"	t_{CHDTL}		–	90	–	80	–	70	ns
76	Clock "High" to $\overline{\mathrm{DTC}}$ "High"	t_{CHDTH}		–	90	–	80	–	70	ns
77	Clock "Low" to $\overline{\mathrm{DTC}}$ "High" Impedance	t_{CLDTZ}		–	120	–	100	–	80	ns
78	$\overline{\mathrm{DTC}}$ Width "Low"	t_{DTCL}		1.0	–	1.0	–	1.0	–	Clk. Per.
81	Clock "Low" to $\overline{\mathrm{PCL}}$ "Low" (1/8 Clock)	t_{CLPL}		–	90	–	80	–	70	ns
82	Clock "Low" to $\overline{\mathrm{PCL}}$ "High" (1/8 Clock)	t_{CLPH}		–	90	–	80	–	70	ns
83	$\overline{\mathrm{PCL}}$ Width "Low" (1/8 Clock)	t_{PCLL}		4.0	–	4.0	–	4.0	–	Clk. Per.
84	$\overline{\mathrm{DTACK}}$ "Low" to Data In (Setup Time)	t_{DALDI}		–	180	–	120	–	90	ns
85	Data In to Clock "Low" (Setup Time)	t_{DICL}		30	–	25	–	15	–	ns
86	$\overline{\mathrm{DS}}$ "High" to Data Invalid (Hold Time)	t_{SHDI}		0	–	0	–	0	–	ns
87	$\overline{\mathrm{DS}}$ "High" to $\overline{\mathrm{DTACK}}$ "High"	t_{SHDAH}		0	240	0	160	0	120	ns
88	$\overline{\mathrm{BEC}}$ "Low" to $\overline{\mathrm{DTACK}}$ "Low"	t_{BECDAL}		50	–	50	–	50	–	ns
89	$\overline{\mathrm{BEC}}$ Width "Low"	t_{BECL}		2.0	–	2.0	–	2.0	–	Clk. Per.
90	$\overline{\mathrm{OWN}}$ "Low" to $\overline{\mathrm{UAS}}$ "Low"			50	–	40	–	30	–	ns
91	$\overline{\mathrm{DDIR}}$ "Low" to $\overline{\mathrm{DBEN}}$ "Low"			50	–	40	–	30	–	ns
92	$\overline{\mathrm{DBEN}}$ "High" to $\overline{\mathrm{DDIR}}$ "High"			50	–	40	–	30	–	ns
93	$\overline{\mathrm{DTACK}}$ Width "High"	t_{DTH}		10	–	10	–	10	–	ns

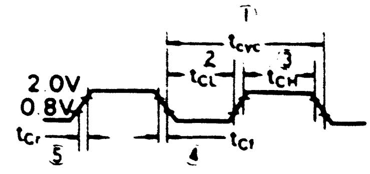


Figure 2 Input Clock Waveform

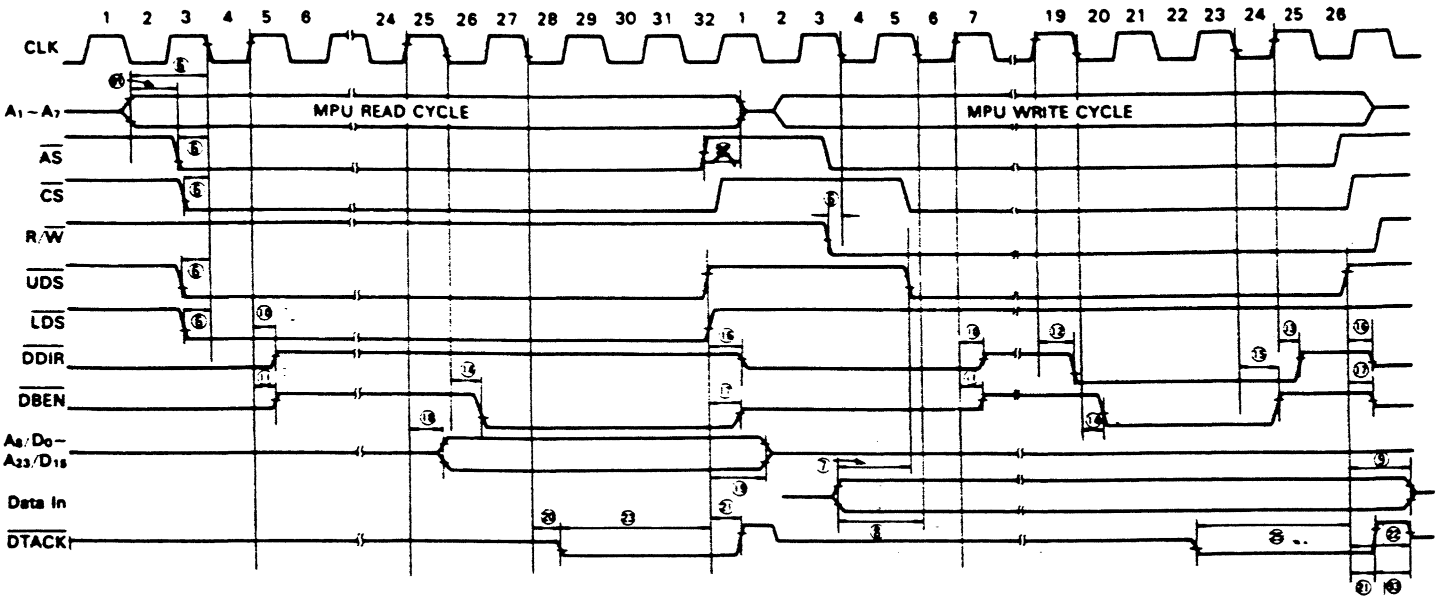
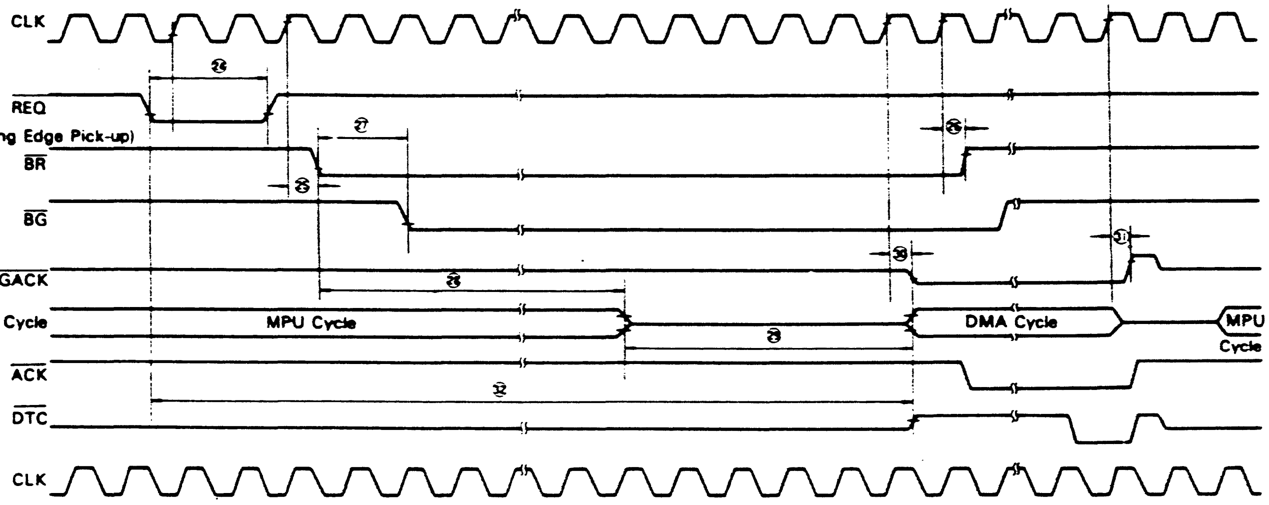


Figure 3 AC Electrical Waveforms — MPU Read/Write



(NOTES)

1) Setup time for the asynchronous inputs $\overline{\text{BG}}$, $\overline{\text{BGACK}}$, $\overline{\text{BEC}_0}$ ~ $\overline{\text{BEC}_2}$, $\overline{\text{CS}}$, $\overline{\text{IACK}}$, $\overline{\text{AS}}$, $\overline{\text{UDS}}$, $\overline{\text{LDS}}$, and R/$\overline{\text{W}}$ guarantees their recognition at the next falling edge of the clock. Setup time for $\overline{\text{REQ}_0}$ ~ $\overline{\text{REQ}_3}$, $\overline{\text{PCL}_0}$ ~ $\overline{\text{PCL}_3}$, $\overline{\text{DTACK}}$, and $\overline{\text{DONE}}$ guarantees their recognition at the next rising edge of the clock.

2) Timing measurements for Input pins are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts. Timing measurements for Output pins are referenced to and from a low voltage of 0.5 volts and a high voltage of 2.4 volts.

3) These waveforms should only be referenced in regard to the edge-to-edge measurement of the timing specifications. They are not intended as a functional description of the input and output signals. Refer to other functional descriptions and their related diagrams for device operation.

Figure 4 AC Electrical Waveforms — Bus Arbitration

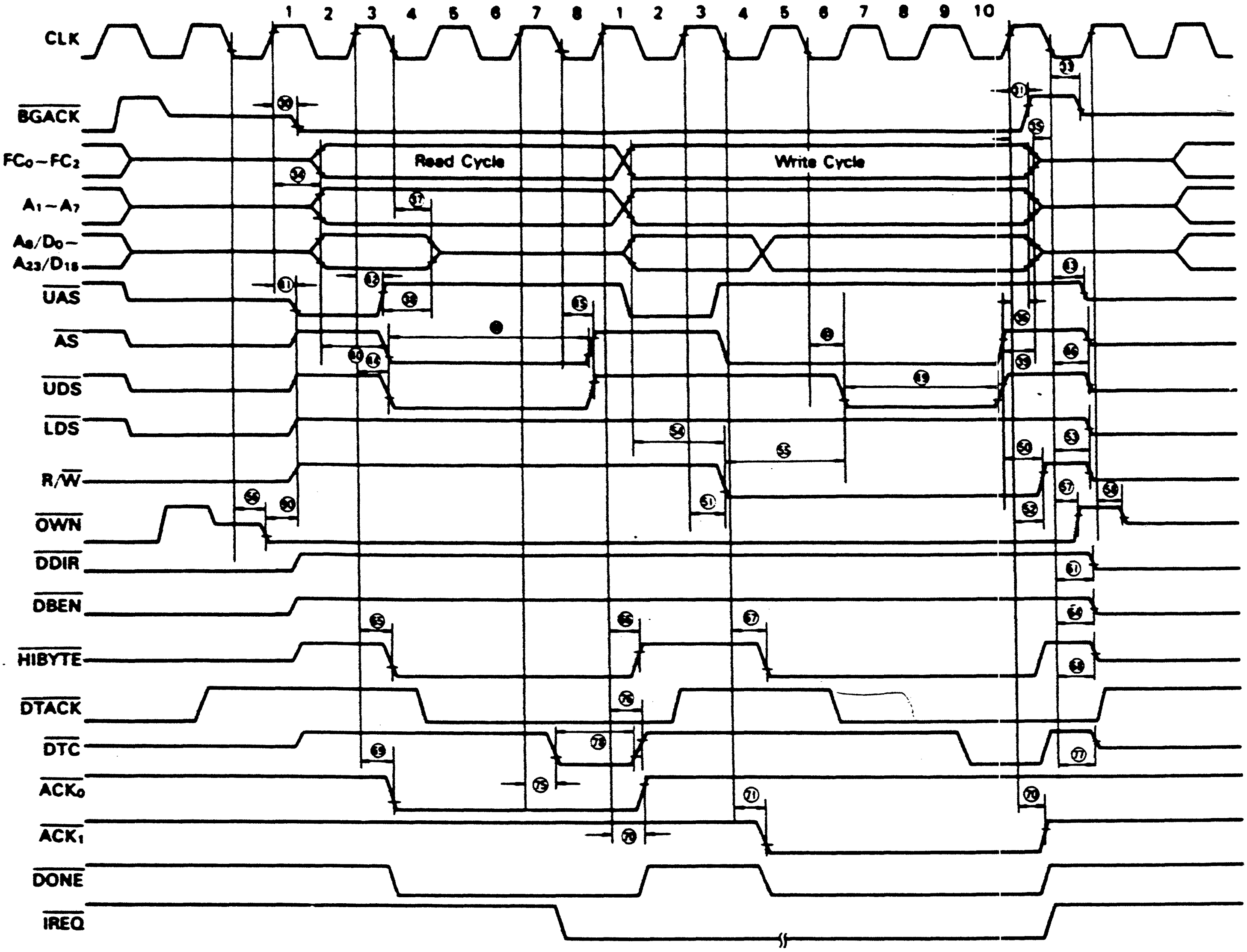


Figure 5 AC Electrical Waveforms – DMA Read/Write (Single Cycle)

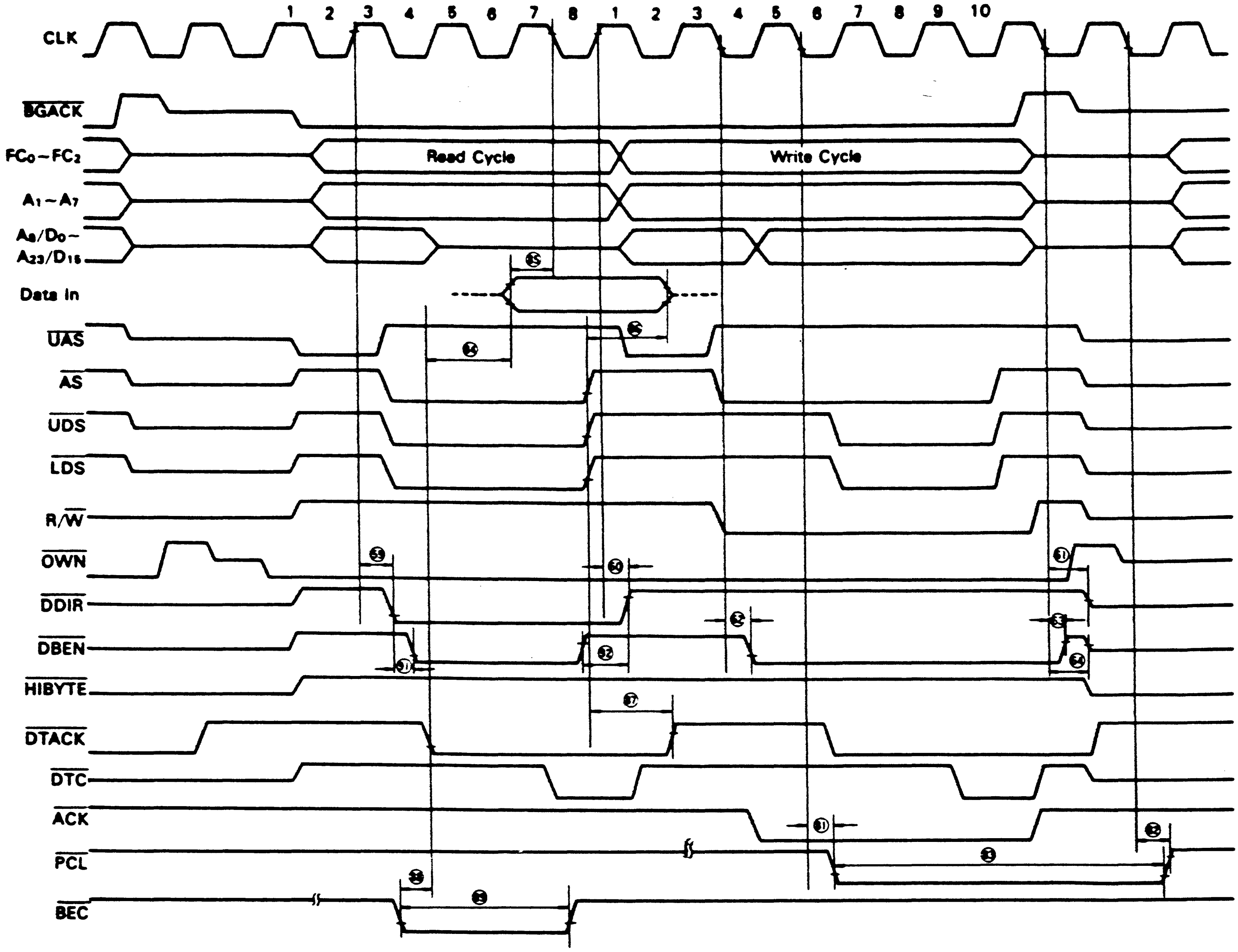


Figure 6 Electrical Waveforms – DMA Read/Write (Dual Cycle)

■ GENERAL DESCRIPTION

This document defines the HD68450, a four channel DMA Controller. The operation of each channel is independent of the other channels. The controller supports single-address or dual-address transfers. The controller supports unchained, array chained, or link chained operations. The device interface includes lines for requesting, acknowledging, and providing incidental control for the device.

The DMAC functions by transferring a series of operands between memory and device; operand sizes can be byte, word, or long word. A block is a sequence of operands; the number of operands in a block is determined by the transfer count. A single channel operation may involve the transfer of several blocks of data between memory and device.

NOTE:

Throughout the specification, signals are discussed using the terms active and inactive or asserted or negated independent of whether the signal is active in the logic one state or the logic zero state.

■ SIGNAL DESCRIPTION

The following section identifies the signals used in connecting to the HMCS68000 bus and peripherals using the DMA controller. Each signal has a basic definition of its use, a detailed description of the operation of each signal is contained in subsequent sections. Specific timing information is also contained in subsequent sections.

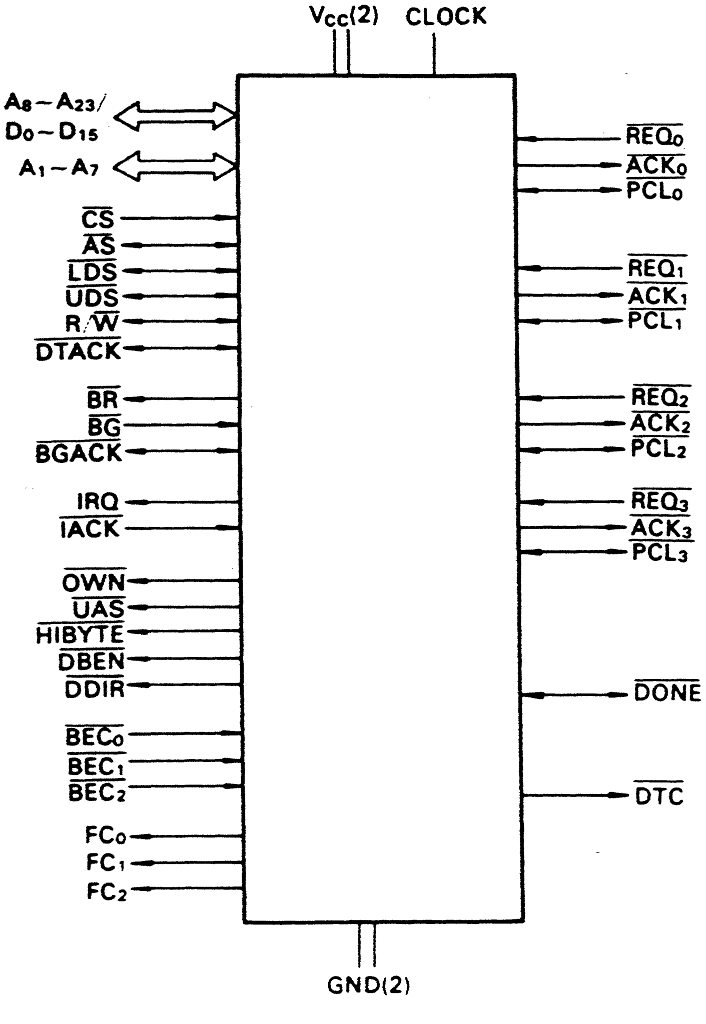


Figure 7 Input and Output Signals

In the following definitions, "MPU mode" refers to the state when the DMAC is chip selected. The term "DMA mode" refers to the state when the DMAC assumes ownership of the bus. The DMAC is in the "IDLE mode" at all other times.

● ADDRESS DATA BUS (A_8/D_0 through A_{23}/D_{15})

Input/Output Three-statable
Active high

These lines are time multiplexed for data and address leads. The lines $\overline{DDIR}$, $\overline{DBEN}$, $\overline{UAS}$ and $\overline{OWN}$ are used to control the demultiplexing of the data/address lines with external gating. This is explained in a later section.

The bi-directional data lines ($D_0 \sim D_{15}$) are used to transfer data between the MPU, DMAC, memory and peripheral devices. Address lines are outputs to address memory and peripheral devices.

● ADDRESS BUS (A_1 through A_7)

Input/Output Three-statable
Active high

In the MPU mode, the low order seven address lines specify which of the internal registers is accessed. The address map for these registers is shown in Table 1. During a DMA bus cycle, $A_1 \sim A_7$ are outputs containing the low order address bits of the location being accessed.

● FUNCTION CODES (FC_0 through FC_2)

Output Three-statable
Active high

These output signals provide the function codes during DMA bus cycles. They are three-stated while in MPU mode or IDLE mode.

● CLOCK (CLK)

Input

This is the HMCS68000 system clock and must not be gated off at any time. Transferring to or from the DMAC registers, sampling of channel request lines, and gating of all control lines are done internally in conjunction with the CLK input.

● CHIP SELECT ($\overline{CS}$)

Input
Active low

This input signal is used to select the DMAC for programmed transfers to and from the DMAC. The DMAC is deselected when the $\overline{CS}$ input is inactive. If the $\overline{CS}$ input is asserted during a bus cycle which is generated by the DMAC, the DMAC internally terminates the bus cycle, signals an address error, but does not perform an operation. This protects DMAC registers during bus cycles which are generated by itself. However, bus cycles generated by any other bus masters, including other DMACs, may address and change the DMAC's internal registers and, consequently, the operation of the DMAC.

- **ADDRESS STROBE ($\overline{AS}$)**

Input/Output Active low	Three-statable

In the MPU or IDLE modes, this signal is monitored by the DMAC if it is requesting, and has been granted, permission to become bus master. In the DMA mode, this signal is an output indicating that the DMAC has placed a valid address on the bus.

- **UPPER ADDRESS STROBE ($\overline{UAS}$)**

Output Active low	Three-statable

This line is an output to latch the upper address bits on the multiplexed data/address lines. Further explanation is given in later sections and diagrams. It is three-stated during the MPU mode and the IDLE mode.

- **OWN ($\overline{OWN}$)**

Output Active low	Three-statable

This line is asserted by the DMAC during DMA mode. It is used to control the output of the transparent latch used to latch the address lines. This line may also be used to control the direction of bi-directional buffers when the loads on $\overline{AS}$, $\overline{LDS}$, $\overline{UDS}$, R/$\overline{W}$ and other signals exceed the drive of the DMAC pins. It is three-stated during the MPU mode and the IDLE mode.

- **DATA DIRECTION ($\overline{DDIR}$)**

Output	Three-statable

This line controls the direction of data through a bidirectional buffer on the data bus. It is three-stated during the IDLE mode.

- **DATA BUS ENABLE ($\overline{DBEN}$)**

Output Active low	Three-statable

This line controls the output of bidirectional buffers on the multiplexed data/address bus. It is three-stated during the IDLE mode.

- **HIGH BYTE ($\overline{HIBYTE}$)**

Output Active low	Three-statable

This line is used when the operand size is byte in the implicit addressing operation. It is asserted when data is present on the upper eight bits of the data bus. It is three-stated during the MPU mode and the IDLE mode.

- **READ/WRITE (R/$\overline{W}$)**

Input/Output Active low	Three-statable

Read/Write (R/$\overline{W}$) is an input in the MPU mode and an output during the DMA mode. In the MPU mode, it is used to control the direction of data flow through the DMAC's input/output data bus interface. In the DMA mode, R/$\overline{W}$ is an output to memory and I/O controllers. It is held three-stated during IDLE mode.

- **UPPER DATA STROBE ($\overline{UDS}$)**

Input/Output Active low	Three-statable

- **LOWER DATA STROBE ($\overline{LDS}$)**

Input/Output Active low	Three-statable

These lines are extensions of the address lines indicating which byte or bytes of data (LSB, MSB) of the addressed word are being addressed

- **DATA TRANSFER ACKNOWLEDGE ($\overline{DTACK}$)**

Input/Output Active low	Three-statable

In the MPU mode $\overline{DTACK}$ is an output indicating that the DMAC has completed the requested data transfer (read or write).

In the DMA mode, the DMAC monitors $\overline{DTACK}$ to determine when a data transfer has completed. In the event that a preemptory bus exception occurs prior to or concurrent with $\overline{DTACK}$, the $\overline{DTACK}$ response is ignored and the bus exception honored. In the IDLE mode, this signal is held in three-state.

- **BUS EXCEPTION CONTROLS ($\overline{BEC_0}$ through $\overline{BEC_2}$)**

Input
Active low

These lines provide an encoded signal indicating some exceptional bus condition. See Page 35 for details on bus exceptions.

- **BUS REQUEST ($\overline{BR}$)**

Output
Active low

The Bus Request ($\overline{BR}$) output is generated by the DMAC to request ownership of the bus.

- **BUS GRANT ($\overline{BG}$)**

Input
Active low

The Bus Grant ($\overline{BG}$) input indicates to the DMAC that it is to be the next bus master. This signal is originated by the MPU and propagated via a daisy chain or other arbitration mechanism. The DMAC cannot assume ownership until both

$\overline{\mathrm{AS}}$ and $\overline{\mathrm{BGACK}}$ become inactive. Once the DMAC acquires the bus, it does not continue to monitor the $\overline{\mathrm{BG}}$ input.

- **BUS GRANT ACKNOWLEDGE ($\overline{\mathrm{BGACK}}$)**

Input/Output	Three-statable
Active low	

Bus Grant Acknowledge ($\overline{\mathrm{BGACK}}$) is a bidirectional control line. As an output, it is generated by the DMAC to indicate that it is the bus master.

As an input, $\overline{\mathrm{BGACK}}$ is monitored by the DMAC in order to determine whether or not the current bus master is a DMA device or not. $\overline{\mathrm{BGACK}}$ must be inactive before the DMAC may assume ownership of the bus.

- **INTERRUPT REQUEST ($\overline{\mathrm{IRQ}}$)**

Output	Open drain
Active low	

Interrupt Request ($\overline{\mathrm{IRQ}}$) is used to interrupt the MPU.

- **INTERRUPT ACKNOWLEDGE ($\overline{\mathrm{IACK}}$)**

Input
Active low

Interrupt acknowledge ($\overline{\mathrm{IACK}}$) is an input to the DMAC indicating that the current bus cycle is an interrupt acknowledge cycle. By the MPU, the DMAC responds with the contents of the normal or exception interrupt vector register of the highest priority channel requesting an interrupt. $\overline{\mathrm{IACK}}$ is not serviced if the DMAC has not generated $\overline{\mathrm{IRQ}}$.

- **CHANNEL REQUEST ($\overline{\mathrm{REQ}_0}$ through $\overline{\mathrm{REQ}_3}$)**

Input
Falling edge or active low

The four $\overline{\mathrm{REQx}}$ inputs ($\overline{\mathrm{REQ}_0}$ ~$\overline{\mathrm{REQ}_3}$) are falling edge sensitive inputs when the request mode is cycle steal. The $\overline{\mathrm{REQx}}$ inputs are low level sensitive when the request mode is burst.

- **CHANNEL ACKNOWLEDGE ($\overline{\mathrm{ACK}_0}$ through $\overline{\mathrm{ACK}_3}$)**

Output	Non-three-statable
Active low	

The four $\overline{\mathrm{ACKx}}$ lines ($\overline{\mathrm{ACK}_0}$ ~ $\overline{\mathrm{ACK}_3}$) indicate to a requesting peripheral device that the bus has been acquired and that the requested bus cycle is beginning. The $\overline{\mathrm{ACKx}}$ line may be used as part of the enable circuit for bus interface to the peripheral.

- **PERIPHERAL CONTROL ($\overline{\mathrm{PCL}_0}$ through $\overline{\mathrm{PCL}_3}$)**

Input/Output	Three-statable
Active low	

The four $\overline{\mathrm{PCLx}}$ lines ($\overline{\mathrm{PCL}_0}$ ~ $\overline{\mathrm{PCL}_3}$) are multi-purpose lines which may be individually programmed to be a START output, an Enable Clock, READY, ABORT, STATUS, or INTERRUPT input.

- **DONE ($\overline{\mathrm{DONE}}$)**

Input/Output	Open drain
Active low	

The one $\overline{\mathrm{DONE}}$ output is normally high and goes low concurrent with $\overline{\mathrm{ACKx}}$ if that channel's operation is completed as a result of that transfer. As an input, it allows the device to indicate a normal termination of the operation.

- **DEVICE TRANSFER COMPLETE ($\overline{\mathrm{DTC}}$)**

Output	Three-statable
Active low	

The single device transfer complete output is normally high and goes low to signal to the device that the data transfer is complete. On a write to memory operation, it indicates that the data has been successfully stored. On a read from memory operation, it indicates that the data is present at the device and should be latched.

■ INTERNAL ORGANIZATION

The DMAC has four largely independent DMA channels. Each channel has its own set of channel registers. These registers define and control the activity of the DMAC in processing a channel operation.

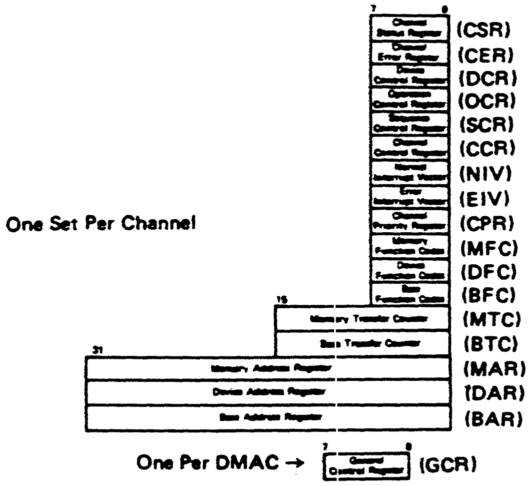


Figure 8 Internal Registers

- **REGISTER ORGANIZATION**

The internal accessible register organization is represented in Table 1. Address space not used within the address map is reserved for future expansion. A read from a reserved location in the map results in a read from the "null register". The null register returns all ones for data and results in a normal bus cycle. A write to one of these locations results in a normal bus cycle but no write occurs. Unused bits of a defined register read as zeros.

Table 1 DMAC Register Addressing Assignments

Register	7	6	5	4	3	2	1	0	Mode
	Address Bits								
Channel Status Register	c	c	0	0	0	0	0	0	R/$\overline{W}$*
Channel Error Register	c	c	0	0	0	0	0	1	R
Device Control Register	c	c	0	0	0	1	0	0	R/$\overline{W}$
Operation Control Register	c	c	0	0	0	1	0	1	R/$\overline{W}$
Sequence Control Register	c	c	0	0	0	1	1	0	R/$\overline{W}$
Channel Control Register	c	c	0	0	0	1	1	1	R/$\overline{W}$
Memory Transfer Counter	c	c	0	0	1	0	1	b	R/$\overline{W}$
Memory Address Register	c	c	0	0	1	1	s	s	R/$\overline{W}$
Device Address Register	c	c	0	1	0	1	s	s	R/$\overline{W}$
Base Transfer Counter	c	c	0	1	1	0	1	b	R/$\overline{W}$
Base Address Register	c	c	0	1	1	1	s	s	R/$\overline{W}$
Normal Interrupt Vector	c	c	1	0	0	1	0	1	R/$\overline{W}$
Error Interrupt Vector	c	c	1	0	0	1	1	1	R/$\overline{W}$
Channel Priority Register	c	c	1	0	1	1	0	1	R/$\overline{W}$
Memory Function Codes	c	c	1	0	1	0	0	1	R/$\overline{W}$
Device Function Codes	c	c	1	1	0	0	0	1	R/$\overline{W}$
Base Function Codes	c	c	1	1	1	0	0	1	R/$\overline{W}$
General Control Register	1	1	1	1	1	1	1	1	R/$\overline{W}$

cc: 00 – Channel #0, 01 – Channel #1
10 – Channel #2, 11 – Channel #3
ss: 00 – High-order, 01 – Upper middle,
10 – Lower middle, 11 – Low-order
b: 0 – High-order, 1 – Low-order
* See Channel Status Register on Page 31.

• DEVICE CONTROL REGISTER (DCR)

The DCR is a device oriented control register. The XRM bits specifies whether the channel is in burst or cycle steal request mode. The DTYP bits define what type of device is on the channel. If the DTYP bits are programmed to be a HMCS6800 device the PCL definition is ignored and the $\overline{\text{PCL}}$ line is an Enable clock input. If the DTYP bits are programmed to be a device with $\overline{\text{READY}}$, the PCL definition is ignored and the $\overline{\text{PCL}}$ line is a ready input (active low). The DPS bit defines what port size the device has. The PCL bits define the function of the $\overline{\text{PCL}}$ line. When the content of the DTYP bits implies HMCS6800 compatible device, or Device with $\overline{\text{ACK}}$ and $\overline{\text{READY}}$, the content of the PCL bits is disregarded. The XRM bits are ignored if an auto request mode in the OCR is selected.

7	6	5	4	3	2	1	0
XRM		DTYP		DPS	0	PCL	

XRM External Request Mode
00 Burst Transfer Mode
01 (undefined, reserved)
10 Cycle Steal Mode without Hold
11 Cycle Steal Mode with Hold

DTYP Device Type
00 HMCS68000 compatible device, explicitly addressed
01 HMCS6800 compatible device, explicitly addressed
10 Device with $\overline{\text{ACK}}$, implicitly addressed
11 Device with $\overline{\text{ACK}}$ and $\overline{\text{READY}}$, implicitly address

DPS Device Port Size
0 8 Bit Port
1 16 Bit Port

PCL Peripheral Control Line
00 Status Input (can be read by reading CSR)
01 Status Input with Interrupt
10 Start Pulse, Negative 1/8 CLK
11 Abort Input

0 Bit 2 Not Used

• OPERATION CONTROL REGISTER (OCR)

The OCR is an operation oriented register. The DIR bit defines the direction of the transfer, to or from memory. The SIZE bits define the size of the operand and how the transfer count and address registers are to be handled. The CHAIN bits tells the DMAC if any or what type of chaining is to be performed. The REQG bits define how requests for transfers are generated. Chaining and requests are discussed in a later section.

7	6	5	4	3	2	1	0
DIR	0	SIZE		CHAIN		REQG	

DIR Direction
0 Transfer from memory to device
1 Transfer from device to memory

0 Bit 6 Unused

SIZE Operation Size
00 Byte
01 Word
10 Long word
11 (undefined, reserved)

CHAIN Chaining Operation
00 Chain operation is disabled
01 (undefined, reserved)
10 Array chaining
11 Linked chaining

REQG DMA Request Generation Method
00 Auto-request at rate limited by General Control Register (GCR)
01 Auto-request at maximum rate
10 $\overline{\text{REQ}}$ line initiates an operand transfer
11 Auto-request the first operand, external request for subsequent operands

• SEQUENCE CONTROL REGISTER (SCR)

The SCR is used to define the sequencing of memory device addresses.

7	6	5	4	3	2	1	0
0	0	0	0	MAC		DAC	

0 Bits 7, 6, 5, 4 Not Used

MAC Memory Address Count
00 Memory address register does not count
01 Memory address register counts up
10 Memory address register counts down
11 (undefined, reserved)

DAC Device Address Register Count
00 Device address register does not count
01 Device address register counts up
10 Device address register counts down
11 (undefined, reserved)

• CHANNEL CONTROL REGISTER (CCR)

The CCR is used to start or terminate the operation of a channel. The register also determines if an interrupt is to be generated at the termination of an operation (normal or error termination). Setting the STR bit causes immediate activation of the channel; the channel will be ready to accept requests immediately. The STR and CNT bits of the register may not be reset by a write to the register. The software abort bit (SAB) may be used to terminate the operation. Setting the SAB bit will reset STR and CNT. Setting the HLT bit will halt the channel, and resetting the HLT bit will resume the operation.

7	6	5	4	3	2	1	0
STR	CNT	HLT	SAB	INT	0	0	0

STR **Start Operation**
0 No operation is pending
1 Start operation

CNT **Continue Operation**
0 No continuation is pending
1 Continue operation

HLT **Halt Operation**
0 Operation not halted
1 Operation halted

SAB **Software Abort**
0 Channel operation not aborted
1 Abort channel operation

INT **Interrupt Enable**
0 No interrupts enabled
1 Interrupts enabled

0 **Bits 2, 1, 0 Not Used**

• CHANNEL STATUS REGISTER (CSR)

The CSR is a register containing the status of the channel.

7	6	5	4	3	2	1	0
COC	BTC	NDT	ERR	ACT	0	PCT	PCS

COC **Channel Operation Complete**
0 Channel operation incomplete
1 Channel operation complete

BTC **Block Transfer Complete**
0 Block transfer incomplete
1 Block transfer complete

NDT **Normal Device Termination**
0 No normal device termination
1 Device terminated operation normally

ERR **Error Bit**
0 No errors
1 Error as coded in CER

ACT **Channel Active**
0 Channel not active
1 Channel active

PCT **$\overline{\text{PCL}}$ Transition**
0 No $\overline{\text{PCL}}$ transition occurred
1 $\overline{\text{PCL}}$ transition occurred

PCS **The State of the $\overline{\text{PCL}}$ Input Line**
0 $\overline{\text{PCL}}$ low
1 $\overline{\text{PCL}}$ high

0 **Bit 2 Not Used**

• CHANNEL ERROR REGISTER (CER)

The CER is an error condition status register. The ERR bit of CSR indicates if there is an error or not. Bits 0–4 indicate what type of error occurred.

7	6	5	4	3	2	1	0
0	0	0	ERROR CODE				

0 **Bits 7, 6, 5 Not Used**

ERROR CODE

00000	No error
00001	Configuration error
00010	Operation timing error
00011	(undefined, reserved)
001rr	Address error
010rr	Bus error
011rr	Count error
10000	External abort
10001	Software abort

rr register or counter code

01	Memory address or memory counter
10	Device address
11	Base address or base counter

• CHANNEL PRIORITY REGISTER (CPR)

The CPR is used to define the priority level for each channel.

7	6	5	4	3	2	1	0
0	0	0	0	0	0	CP	

CP **Channel Priority**
These two bits determine the priority (0–3) of the channel.

0 **Bits 7, 6, 5, 4, 3, 2 Not Used**

• GENERAL CONTROL REGISTER (GCR)

The GCR is used to define what portion of the bus cycles is available to the DMAC for limited rate auto-request generation.

7	6	5	4	3	2	1	0
0	0	0	0	BT		BR	

BT **Burst Time**
The number of DMA clock cycles per burst that the DMAC allows in the auto-request at a limited rate of transfer is controlled by these two bits. The number is 2** (BT+4) (two to the BT+4 power).

BR **Bandwidth Ratio**
The amount of the bus bandwidth utilized by the auto-request at a limited rate transfer is controlled by these two bits. The ratio is 2** (BR+1) (two to the BR+1 power).

0 **Bits 7, 6, 5, 4 Not Used**

• ADDRESS REGISTERS

Three 32-bit registers are utilized to implement the Memory Address Register, Device Address Register, and the Base Address Register. Due to packaging limitations, only the least significant twenty-four bits are connected to the address output pins.

• FUNCTION CODE REGISTERS

Each address register has a function code register associated with it. The function code registers are three bits wide and are loaded via the lowest three bits of the data bus.

• TRANSFER COUNT REGISTERS

Two sixteen bit counters per channel are provided to implement the Memory Transfer Counter and the Base Transfer Counter.

• INTERRUPT VECTOR REGISTERS

Each channel has an interrupt vector register and an error interrupt vector register, each consisting of eight bits.

■ OPERATION DESCRIPTION

• GENERAL DESCRIPTION

A DMAC channel operation proceeds in three principal phases. During the initialization phase, the MPU configures the channel control registers, sets initial addresses, and starts the channel. During the transfer phase, the DMAC accepts requests for data operand transfers, and provides addressing and bus controls for the transfers. The termination phase occurs after the operation is completed when the DMAC reports the status of the operation. Refer to Figure 9 through Figure 11 for MPU DMAC communication timings.

This section describes DMAC operation. A brief description of the device/DMAC communication is given first. Next, the transfer phase is covered, including how the DMAC recognizes requests and how the DMAC arranges for data transfer. Following this, the initialization phase is described. The termination phase is covered, introducing chaining, error signaling, and bus exceptions. A description of the channel priority scheme rounds out the section.

The MPU reads and writes the DMAC internal registers to control the operations. Figure 9 indicates the timing diagram when the MPU reads the contents of the DMAC internal register. the MPU outputs $A_1 \sim A_{23}$, $\overline{AS}$, R/$\overline{W}$, $\overline{UDS}$, and $\overline{LDS}$ and accesses the internal register. The DMAC outputs data on the bus, the buffer control signals ($\overline{DDIR}$ and $\overline{DBEN}$), and $\overline{DTACK}$. Read cycle consists of sixteen clocks. Figure 10 shows the timing of the MPU writes to the DMAC. Write cycle consists of thirteen clocks.

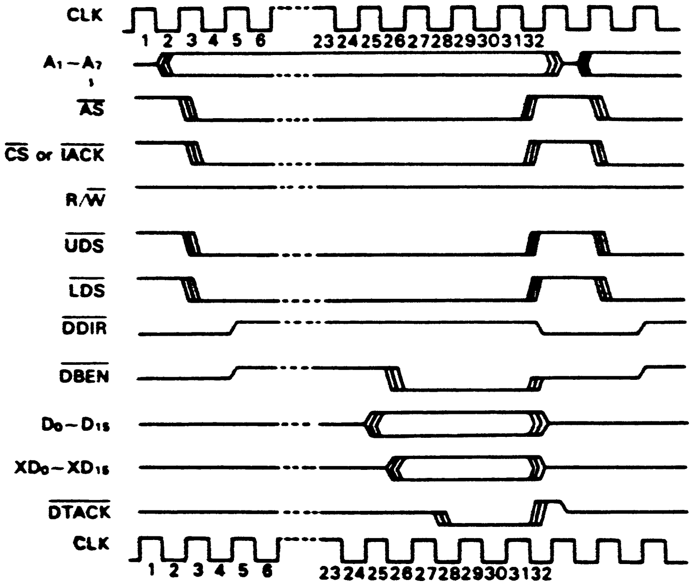


Figure 9 MPU Read from DMAC – Word

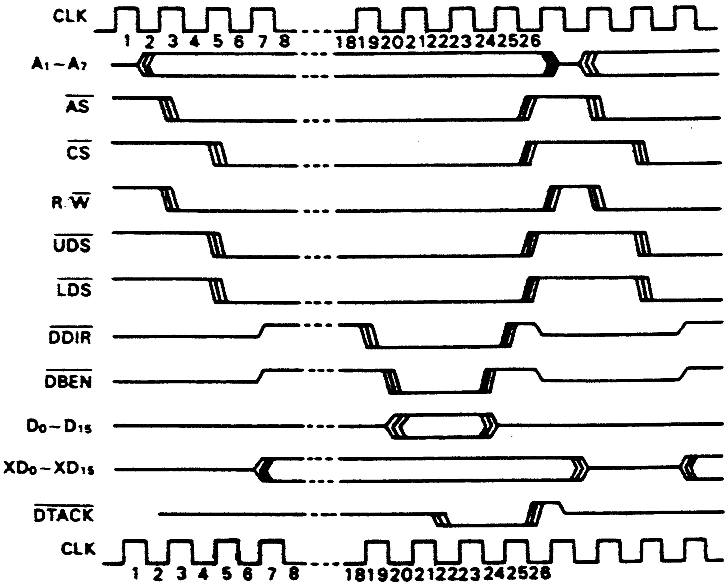
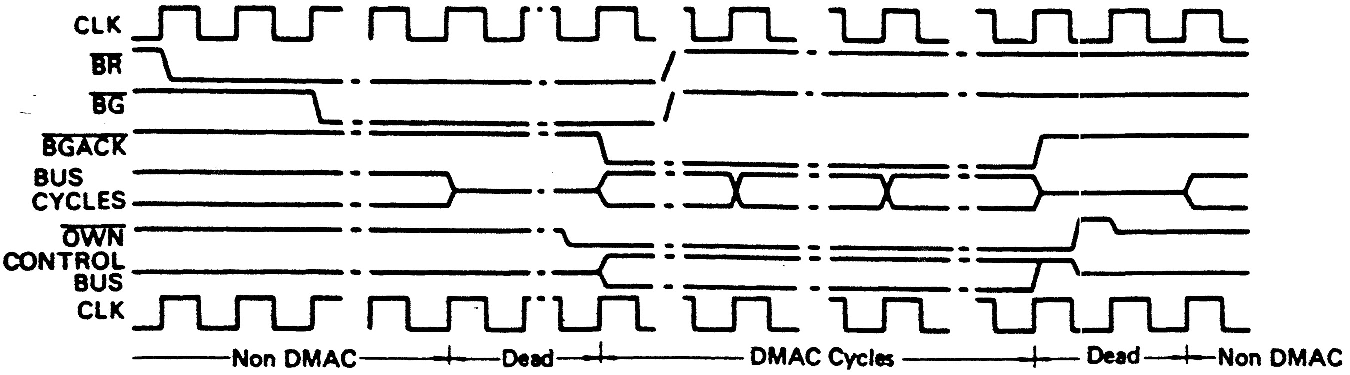


Figure 10 MPU Write to DMAC – Word

(NOTES)

1) The $\overline{CS}$ equation includes data strobe.
2) $D_0 \sim D_{15}$ represent the multiplexed address/data pins which are used for data only during MPU mode.
3) $XD_0 \sim XD_{15}$ is the external or 68000 system data bus, i.e. on the system side of the data buffers.
4) Cycle lengths reflect the response of the current HD68000 MPU.
5) In the MPU read from DMAC mode, the DMAC will not give $\overline{DTACK}$ until the data is guaranteed valid on the system data bus for one half clock.
6) During the MPU read, the DMAC must remove signals within one clock after $\overline{AS}$ is negated.
7) The DMAC will negate $\overline{DTACK}$ within one clock after $\overline{AS}$ is negated.
8) During the MPU write to DMAC, the $\overline{DDIR}$ line will be driven low to direct the data buffers toward to DMAC before the buffers are enabled.
9) The DMAC will latch the data (MPU write to DMAC) before asserting $\overline{DTACK}$. Once the data is latched the DMAC will negate $\overline{DBEN}$ and $\overline{DDIR}$ in the proper order.
10) $\overline{CS}$ will be removed within one clock after $\overline{AS}$ is negated.
11) Note that $\overline{DDIR}$ and $\overline{DBEN}$ must drive out of tristate when $\overline{CS}$ is detected and then must be re-tristated at the end of the cycle.
12) The clock reference shown in this diagram is the CPU clock.

Figure 11 indicates the DMAC bus arbitration timing. The DMAC asserts $\overline{BR}$ to request the bus mastership. The MPU issues $\overline{BG}$ to grant the ownership in the next bus cycle. After the end of the current cycle, the DMAC starts its own bus cycle accompanied with the dead cycles.



(NOTES) 1) Note the timing of the $\overline{OWN}$ signal. It will drive active one half clock prior to the start of the first DMAC cycle. It will drive inactive one half clock after the end of the last DMAC cycle. At this same time, all other control signals will tristate. One half clock after this, the $\overline{OWN}$ signal will tristate.

2) CONTROL BUS refers to the control pins such as $\overline{DBEN}$, $\overline{AS}$, $\overline{ACK}$, etc. on the DMAC.

3) $\overline{BR}$ signal will be negated one clock after $\overline{BGACK}$ signal is asserted.

Figure 11 DMAC Bus Arbitration Timing

- **DEVICE/DMAC COMMUNICATION**

Communication between peripheral devices and the DMAC is accommodated by five signal lines. Each channel has a request ($\overline{REQ}$), an acknowledge ($\overline{ACK}$), and a peripheral control line ($\overline{PCL}$). The last two lines, the $\overline{DONE}$ and $\overline{DTC}$ lines, are shared among the four channels.

(1) REQUEST ($\overline{REQ}$)

A channel can make a request for service by asserting the individual channel request line.

(2) ACKNOWLEDGE ($\overline{ACK}$)

Each channel has an acknowledge line which is activated during transfers to or from the device. This line is used to implicitly address the device which is transferring the data. It may also be used to control the buffering circuits between the device and the HMCS68000 bus.

(3) PERIPHERAL CONTROL LINE ($\overline{PCL}$)

Each channel has a peripheral control line. The function of this line is quite flexible, and is determined by the programmed state of DCR.

The DTYP bits of the DCR define what type of device is on the channel. If the DTYP bits are programmed to be a HMCS6800 device, the PCL definition is ignored and the $\overline{PCL}$ line is an Enable clock input. If the DTYP bits are programmed to be a device with $\overline{READY}$, the PCL definition is ignored and the $\overline{PCL}$ line is a ready input.

The $\overline{PCL}$ line is active at all times when the $\overline{PCL}$ line is programmed as a Status input, Interrupt input, a Ready input, or an Enable input. When programmed to be an Abort input it is only active after the channel has been started.

$\overline{PCL}$ AS A STATUS INPUT

The $\overline{PCL}$ line may be programmed as a status input. The status level can be determined by reading the PCS bit in the CSR. If a negative transition occurs and remains stable for two DMAC clock cycles on the $\overline{PCL}$ line, the PCT bit of the CSR is set. This bit is cleared by resetting the DMAC or writing to the PCT bit of the Channel Status Register.

$\overline{PCL}$ AS AN INTERRUPT

The $\overline{PCL}$ line may also be programmed to generate an interrupt on a negative transition. This enables an interrupt which is requested if the PCT bit of the CSR is set.

$\overline{PCL}$ AS A STARTING CLOCK PULSE

The $\overline{PCL}$ line may be programmed to output a single pulse. The duration of the active low pulse is eight clock cycles, and starts when the channel is activated.

$\overline{PCL}$ AS AN ABORT INPUT

The $\overline{PCL}$ line may be programmed to be an negative transition abort input which terminates an operation by signaling the abort error. When this function has been programmed, the $\overline{PCL}$ line is only active after the channel has been started. The negative transition must remain stable in a low level for a minimum of two DMAC clock cycles.

$\overline{PCL}$ AS AN ENABLE INPUT

If the DTYP bits are programmed to be a HMCS6800 device, the PCL definition is ignored and the $\overline{PCL}$ line is an Enable clock input. The Enable clock downtime must be as long as five clock cycles, and must be high for a minimum of three DMAC clock cycles, but need not be synchronous with the clock.

$\overline{PCL}$ AS A READY INPUT

If the DTYP bits are programmed to be a device with $\overline{READY}$, the PCL definition is ignored and the $\overline{PCL}$ line is a $\overline{READY}$ input. The $\overline{READY}$ is an active low input.

(4) $\overline{DONE}$

$\overline{DONE}$ is an active low signal which is asserted when the memory transfer count is exhausted, and there are no more links to pick up in a chaining operation or the continue bit is not set. It is asserted and negated coincident with the acknowledge signal of the last operand part.

The DMAC also monitors the state of the $\overline{\text{DONE}}$ line while acknowledging a device. If the device asserts $\overline{\text{DONE}}$, the DMAC will terminate the operation after the transfer of the current operand. The DMAC terminates the operation by clearning the ACT bit of the CSR, and setting the COC and NDT bits of the CSR. If both the DMAC and the device asserts $\overline{\text{DONE}}$, the device termination is not recognized, but the channel operation does terminate.

(5) DATA TRANSFER COMPLETE

$\overline{\text{DTC}}$ is an active low signal which is asserted when the actual data transfer is accomplished. If data is being transferred from the device, $\overline{\text{DTC}}$ is asserted to indicate that the data is valid at the device, and should be latched. If a preemptory bus exception terminates the bus cycle, $\overline{\text{DTC}}$ is not asserted. $\overline{\text{DTC}}$ is an active signal whenever the DMAC is a bus master. It is asserted for both memory and peripheral DMAC initiated transfers.

• REQUESTS

Requests activate the DMAC to transfer an operand. The REQG bits of the OCR determine the manner in which requests are generated. Requests may be externally generated by circuitry in the device, or internally generated by the auto-request mechanism. Usually a single operation uses only one method of request generation, but an operation can auto-request the first transfer and then wait for the device to request further transfers.

(1) AUTO-REQUEST TRANSFERS

The auto-request mechanism provides generation of requests within the DMAC. These requests can be generated at either of two rates: maximum-rate, so that the channel always has a request pending, or limited-rate. The limited rate auto-request feature functions by monitoring the bus utilization.

AUTO-REQUEST BUS UTILIZATION

The DMAC monitors bus utilization to control the limited-rate auto-request (LRAR) feature. This monitoring is also used to determine when an external request device has paused.

The DMAC divides time into equal length sample intervals by counting clock cycles. The end of one sample interval marks the beginning of the next. During a sample interval, the DMAC notes bus and channel activity. At the end of the interval, decisions are made which affect channel operations during the next sample interval, as shown in Figure 12.

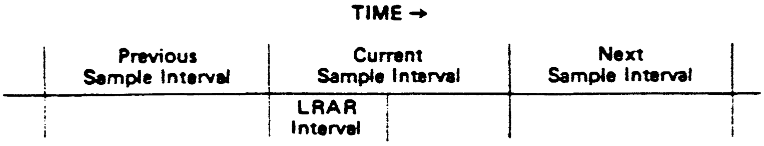


Figure 12 DMAC Sample Intervals

Based on the DMA activity during a sample interval, the DMAC allows limited-rate auto-requests for some initial portion of the next sample interval. The length of the sample interval, and the portion of the sample interval during which limited rate auto-requests can be made are controlled by the BT and BR parameters in the GCR. The length in clock cycles of the subinterval during which the DMAC allows limited-rate auto-requests is controlled by the BT. The number is 2** (BT+4). For example, if BT equals TWO and the DMA utilization of the bus was low during the previous sample interval, then the DMAC generates as many auto-request transfers as is possible during the first 64 clock cycles of the current sample interval.

The ratio of the length of the sample interval to the length of the limited-rate auto-request interval is controlled by the BR bits. This same parameter is used to determine the level of DMA bus utilization during the sample interval. If the fraction of DMA clock cycles during a sample interval exceeds the programmed utilization level, the DMAC will not allow limited-rate auto-requests during the next sample interval. Either ratio is 2** (BR+1) (2 raised to the BR+1 power). For example, if BR equals THREE, then at most one out of 16 clock cycles during a sample interval can be a DMA cycle, and still the DMAC would allow limited-rate auto-requests during the next sample interval. The DMAC monitors $\overline{\text{BGACK}}$ during each clock cycle to determine whether or not that clock cycle is used by a DMA device. If the $\overline{\text{BGACK}}$ input is active, the DMAC assumes that that clock cycle is for a DMA device. If it is inactive, the DMAC assumes that it is not a DMA cycle.

The sample interval length is not a direct parameter, but is equal to 2** (BT+BR+5) clock cycles. Thus the sample interval can vary from 32 to 2048 clock cycles.

AUTO-REQUEST

If the REQG bits in the OCR indicate auto-request at the maximum rate, the DMAC acquires the bus after the operation is started and transfers data until channel termination. The DMAC does not relinquish the bus until termination. If a request is made by another channel of equal or higher priority, the DMAC services that channel and then resumes the auto-request sequence.

If the REQG bits indicate auto-request at a limited rate, the channel generates requests only during the limited rate auto-request interval and then only when the bus utilization was below the required threshould during the previous sample interval. As a consequence, if an auto-request at maximum rate transfer is started, no limited rate auto-requests are generated before the termination of the maximum rate auto-request operation.

The $\overline{\text{ACK}}$, $\overline{\text{PCL}}$ and $\overline{\text{DTC}}$ lines are held inactive during an auto-request operation if the device type is HMCS68000 compatible. Consequently, any channel may be used for the auto-request function in addition to its normal application without disturbing any peripheral devices connected to that channel.

Refer to Figure 13 for more specific timing diagrams.

(2) EXTERNAL REQUESTS

If the REQG bits of the OCR indicate that the $\overline{\text{REQ}}$ line generates requests, the transfer requests are generated externally. The request line associated with each channel allows the device to externally generate requests for DMA transfers. When the device wants an operand transferred, it makes a request by asserting the request line. The external request mode is determined by the XRM bits of the DCR, which allows both burst and cycle steal request modes. The burst request mode allows a channel to request the transfer of multiple operands using consecutive bus cycles. The cycle steal request mode allows a channel to request the transfer of a single operand.

BURST REQUEST RECOGNITION

In the burst request mode, the $\overline{\text{REQ}}$ line is an active low input. The device requests an operand transfer by asserting $\overline{\text{REQ}}$. The DMAC services the request by arbitrating for the HMCS68000 bus, obtaining the bus, and notifying the peripheral by asserting the acknowledge line. If the request line is active when the DMAC asserts acknowledge, and remains active at least until the DMAC asserts device transfer complete, the DMAC recognizes a valid request for another operand, which will be transferred during the next bus cycle if the channel has priority. If the request line is negated before the DMAC asserts device complete, the DMAC determines there is no valid request for an operand transfer, and no transfers are generated for that channel. Channels of the same or higher priority within the same DMA Controller may have DMA operand transfer requests serviced during this mode.

If the request is negated before the first transfer cycle has started, the cycle will terminate with the DMAC returning the bus.

Refer to Figure 14 for more specific timing diagrams.

CYCLE STEAL REQUEST RECOGNITION

In the cycle steal request mode, the device requests an operand transfer by generating a falling edge on the $\overline{\text{REQ}}$ line. The DMAC services a request by arbitrating for the bus, obtaining the bus and notifying the peripheral by asserting the acknowledge line.

After an request edge has been asserted it must remain at the assertion level at least two clock cycles. The request line must be inactive at least one clock cycle before a request is made. If another request from the channel is received before the first operand part of a former request is acknowledged, the second request is not recognized.

After the DMAC completes the transfer, it may service another channel, relinquish the bus, or hold the bus and wait for another request. If there are pending requests from other channels, one of the requesting channels is serviced. If there are no requests, the XRM bits determine whether the DMAC will relinquish the bus, or retain ownership.

If the XRM bits specify cycle steal with hold, the DMAC will retain ownership. The bus is not given up for arbitration until the channel operation terminates or until the device pauses. The device is determined to have paused if it does not make any requests during the next full sample interval. The sample interval counter is free running and is not reset or modified by this mode of operation. The sample interval counter is the same counter that is used for Limited Rate Auto Request and is programmed via the GCR.

If the XRM bits specify cycle steal without hold, the DMAC will relinquish the bus. If the device generates a request before DMAC asserts $\overline{\text{DTC}}$ for the last operand part, the DMAC will retain ownership of the bus, and that request will be serviced before the DMAC relinquishes the bus.

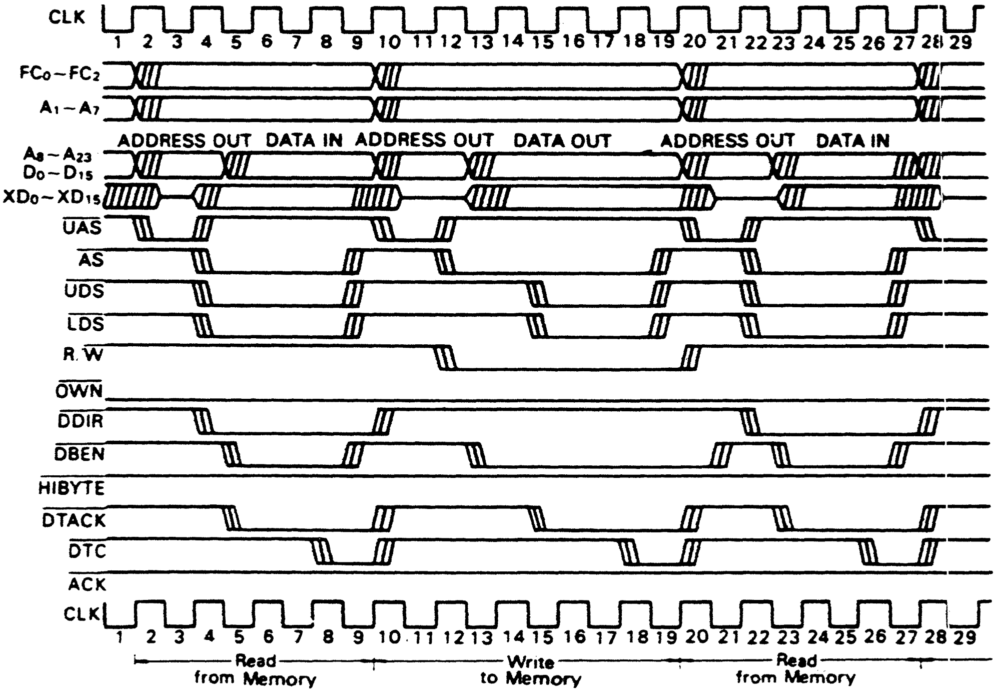
Refer to Figure 15 and Figure 16 for more specific timing diagrams.

REQUEST RECOGNITION IN DUAL-ADDRESS TRANSFERS

In a following section dual-address transfers are defined. Dual address transfer is an exception to the request recognition rules in the previous paragraphs. Refer to the Explicitly Addressed Device section for information.

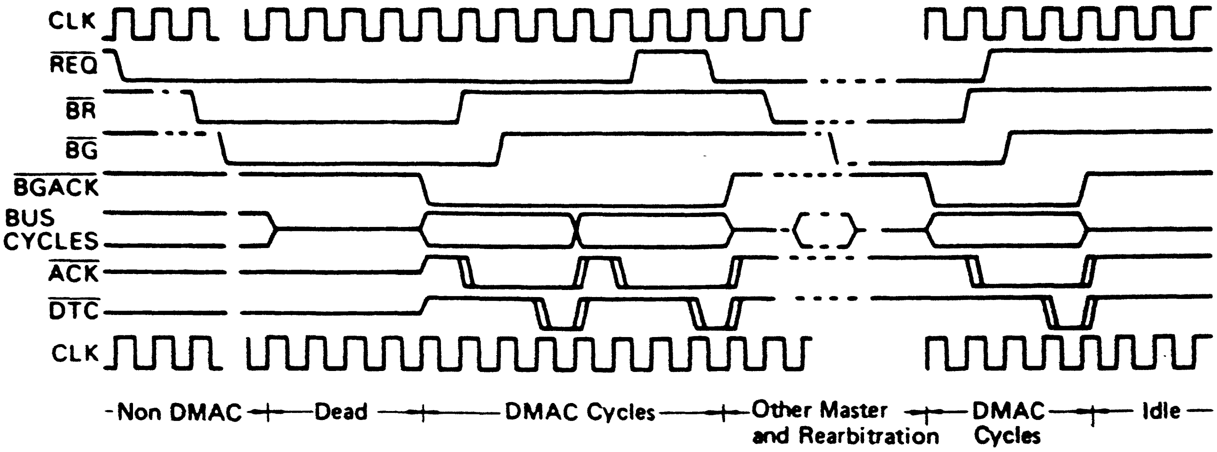
(3) MIXED REQUEST GENERATION

A single channel can mix the two request generation methods. By appropriately programming the REQG bits of the OCR, when the channel is started, the DMAC auto-requests the first transfer. Subsequent requests are then generated externally by the device. The $\overline{\text{ACK}}$ and $\overline{\text{PCL}}$ lines perform their normal functions in this operation.



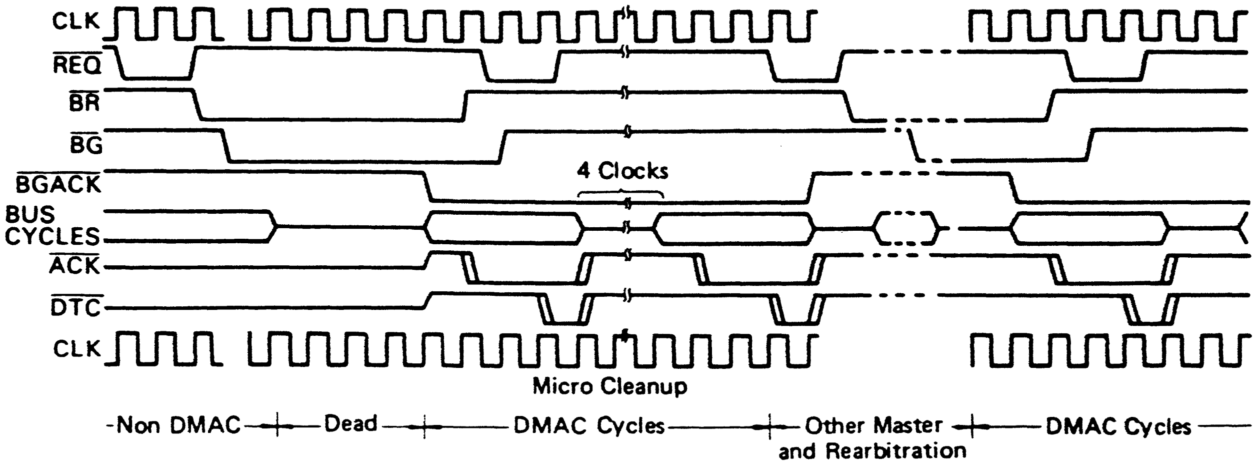
(NOTE) 1) Note that $\overline{\text{ACK}}$, $\overline{\text{DONE}}$, $\overline{\text{DTC}}$, and $\overline{\text{HIBYTE}}$ are always inactive in this mode. For comments on the other signals, see notes on the dual addressing mode with 8 bit device as source.

Figure 13 DMAC Auto Request Read – Write – Read Cycles



(NOTE) 1) Note that in the diagrams showing request timing it is assumed that only one channel is active.

Figure 14 DMAC Burst Mode Request Timing



(NOTES) 1) In this mode the device must re-assert $\overline{\text{REQ}}$ one clock before the assertion edge of $\overline{\text{DTC}}$ of the last bus cycle or lose the bus. The $\overline{\text{REQ}}$ signal is edge triggered.

2) The time labeled "micro cleanup" is the time it takes for the internal sequencer to start another bus cycle if no other channel has requests pending.

Figure 15 DMAC Cycle Steal Mode Request Timing

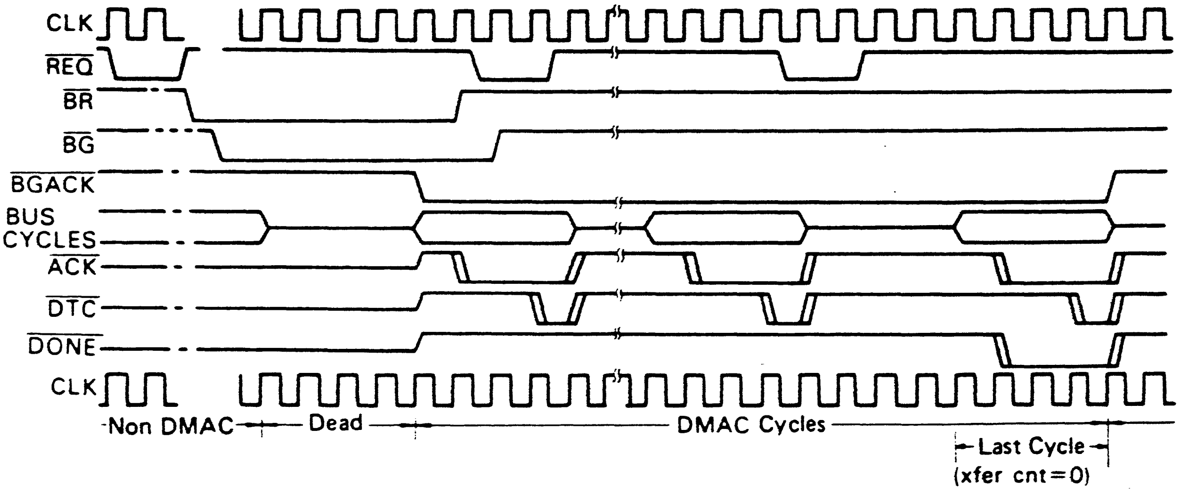


Figure 16 DMAC Cycle Steal-Hold Mode Request Timing

• DATA TRANSFERS

(1) DEVICE PROTOCOLS

All DMAC data transfers are assumed to be between memory and another device. The word "memory" means a 16-bit HMCS68000 bus compatible device. By programming the DCR, the characteristics of the device may be assigned. Each channel can communicate using any of the following protocols.

DTYP	Device Type	
00	HMCS68000 compatible device	Dual Addressing
01	HMCS6800 compatible device	
10	Device with $\overline{ACK}$	Single Addressing
11	Device with $\overline{ACK}$ and $\overline{READY}$	

DUAL ADDRESSING

HMCS68000 and HMCS6800 compatible devices may be explicitly addressed. This means that before the peripheral transfers data, a data register within the device must be addressed. Because the address bus is used to address the peripheral, the data cannot be directly transferred to/from the memory because the memory also requires addressing. Instead, the data is transferred from the source to the DMAC and held in an internal DMAC holding register. A second bus transfer between the DMAC and the destination is then required to complete the operation. Because both the source and destination of the transfer are explicitly addressed, this protocol is also called dual-addressed.

Request Recognition in Dual-Address Transfers

The request recognition protocols defined in a previous section apply to dual-address operations. Requests are recognized during the transfer to/from the DMAC holding register and the peripheral as described in the request protocol section. This protocol requires the request to be asserted before the signal $\overline{DTC}$ is asserted, to have request recognition for the next cycle.

During the portion of the operation when the operand or operand part is transferred between the DMAC holding register and memory, requests are also recognized. During the transfer between memory and the holding register, $\overline{DTC}$ is not asserted, so it may not be used as reference point for request recognition during this portion of the operation. However, requests will be recognized if they are asserted prior to the portion of the cycle where $\overline{DTC}$ would have been asserted. This point is one half clock cycle before the upper and lower data strobes are negated.

HMCS68000 Compatible Device Transfers

In this operation, when a request is received, the bus is obtained and the transfer completed using the HMCS68000 bus protocol as shown in Figures 17 and 18. Refer to Figures 19 through 22 for timing information.

HMCS6800 Compatible Device Transfers

When a channel is programmed to perform HMCS6800 compatible transfers, the $\overline{PCL}$ line for that channel is defined as an Enable clock input. The DMAC performs data transfers between itself and the device using the HMCS6800 bus protocol, with the $\overline{ACK}$ output providing the valid memory address signal. This operation is necessary since the HMCS6800 bus is synchronous and the HMCS68000 bus is asynchronous. Figure 23 illustrates this protocol. This operation provides DMAC compatibility with existing HMCS6800 and other synchronous devices. Refer to Figures 24 and 25 for timing information. Figure 44 illustrates a sample circuit diagram of a two-6800 device system.

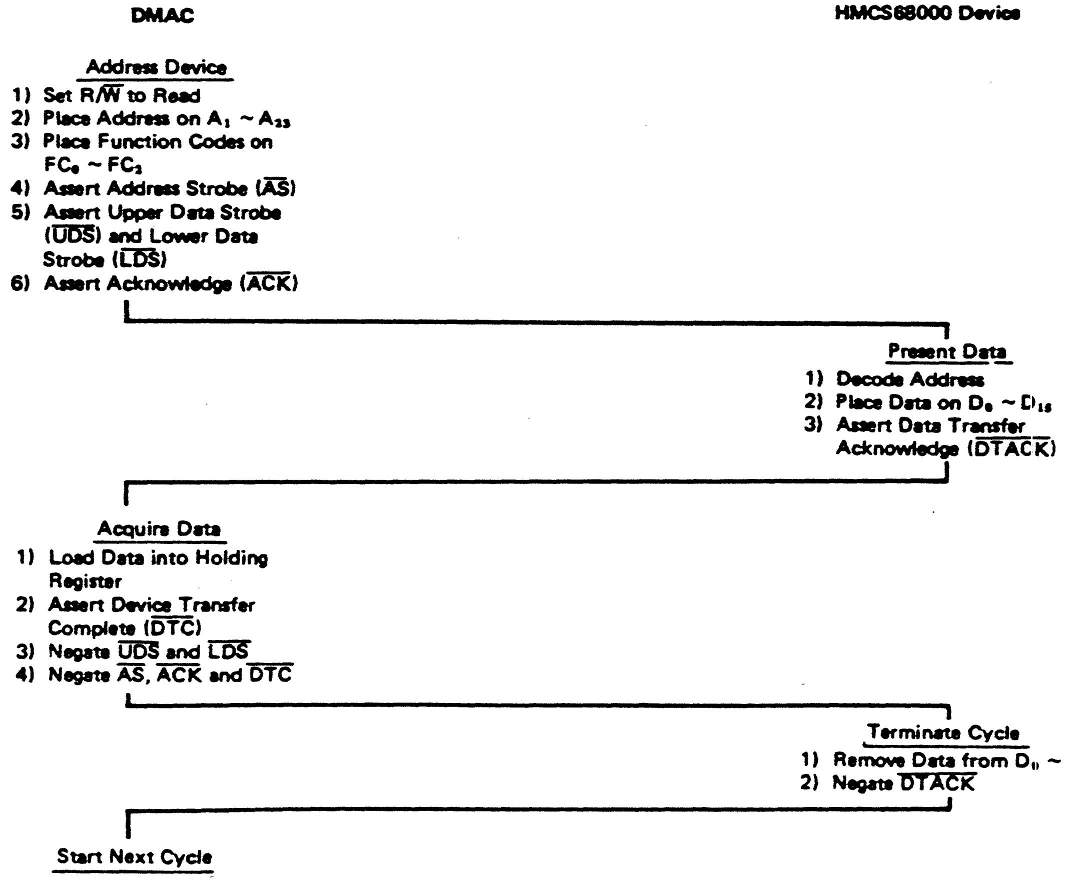

DMAC
HMCS68000 Device
Address Device
1) Set R/$\overline{W}$ to Read
2) Place Address on $A_1 \sim A_{23}$
3) Place Function Codes on $FC_0 \sim FC_2$
4) Assert Address Strobe ($\overline{AS}$)
5) Assert Upper Data Strobe ($\overline{UDS}$) and Lower Data Strobe ($\overline{LDS}$)
6) Assert Acknowledge ($\overline{ACK}$)
Present Data
1) Decode Address
2) Place Data on $D_0 \sim D_{15}$
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)
Acquire Data
1) Load Data into Holding Register
2) Assert Device Transfer Complete ($\overline{DTC}$)
3) Negate $\overline{UDS}$ and $\overline{LDS}$
4) Negate $\overline{AS}$, $\overline{ACK}$ and $\overline{DTC}$
Terminate Cycle
1) Remove Data from $D_0 \sim D_{15}$
2) Negate $\overline{DTACK}$
Start Next Cycle

Figure 17 Word Read Cycle Flowchart HMCS68000 Type Device

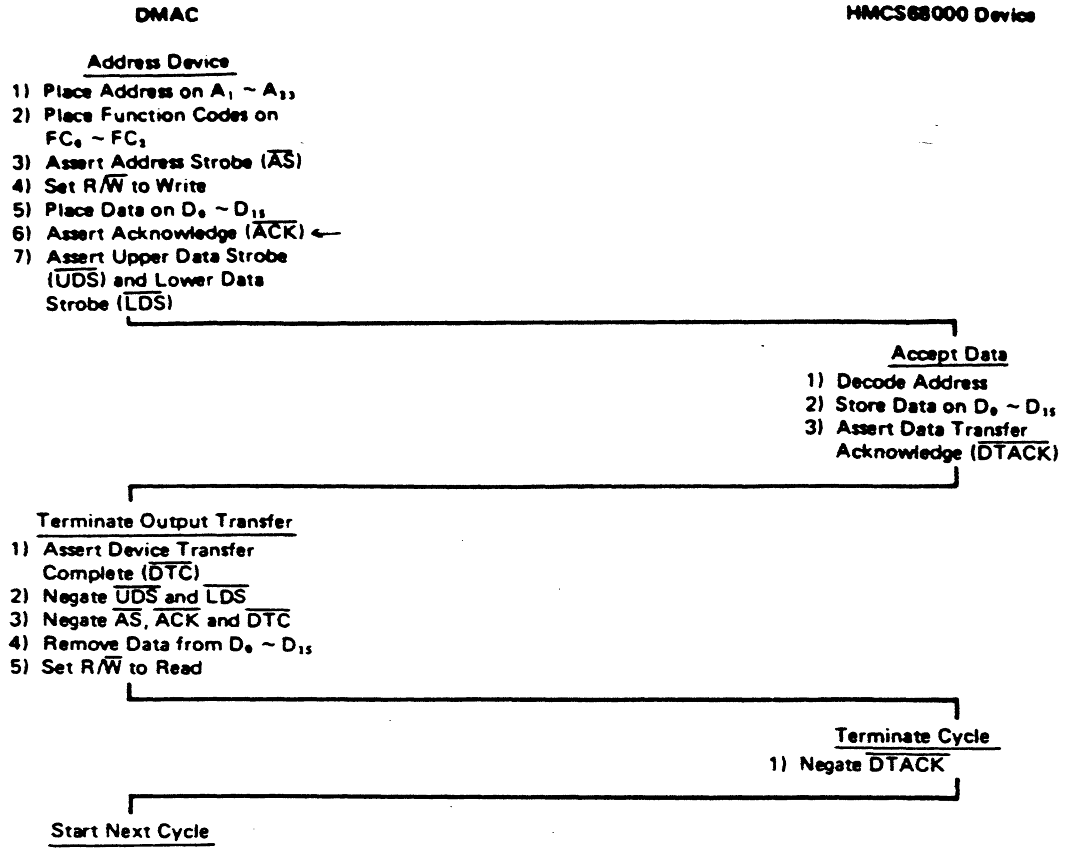
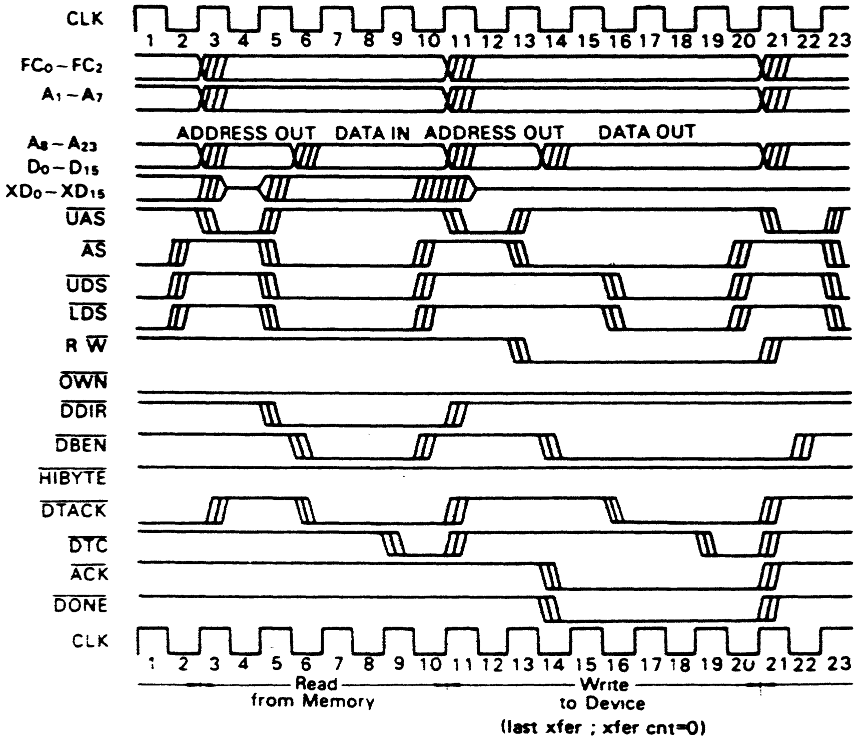
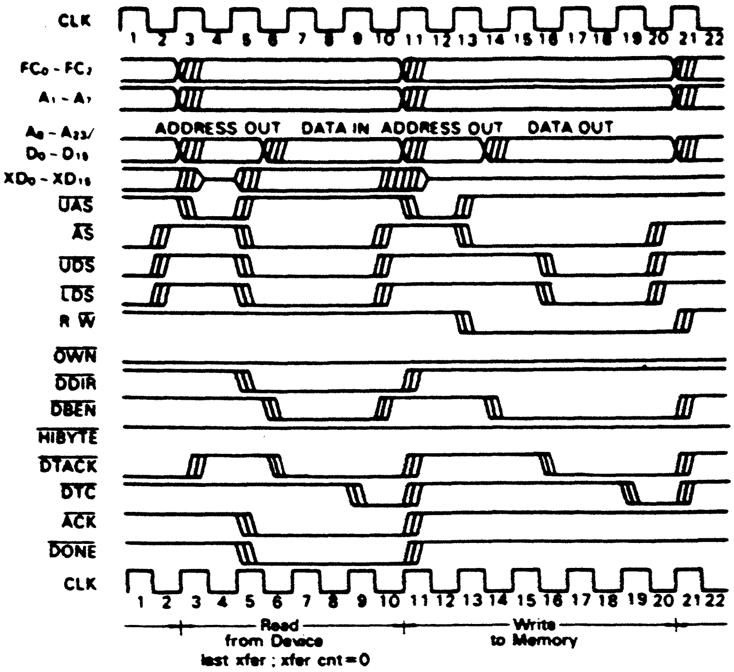

DMAC
HMCS68000 Device
Address Device
1) Place Address on $A_1 \sim A_{23}$
2) Place Function Codes on $FC_0 \sim FC_2$
3) Assert Address Strobe ($\overline{AS}$)
4) Set R/$\overline{W}$ to Write
5) Place Data on $D_0 \sim D_{15}$
6) Assert Acknowledge ($\overline{ACK}$)
7) Assert Upper Data Strobe ($\overline{UDS}$) and Lower Data Strobe ($\overline{LDS}$)
Accept Data
1) Decode Address
2) Store Data on $D_0 \sim D_{15}$
3) Assert Data Transfer Acknowledge ($\overline{DTACK}$)
Terminate Output Transfer
1) Assert Device Transfer Complete ($\overline{DTC}$)
2) Negate $\overline{UDS}$ and $\overline{LDS}$
3) Negate $\overline{AS}$, $\overline{ACK}$ and $\overline{DTC}$
4) Remove Data from $D_0 \sim D_{15}$
5) Set R/$\overline{W}$ to Read
Terminate Cycle
1) Negate $\overline{DTACK}$
Start Next Cycle

Figure 18 Word Write Cycle Flowchart HMCS68000 Type Device



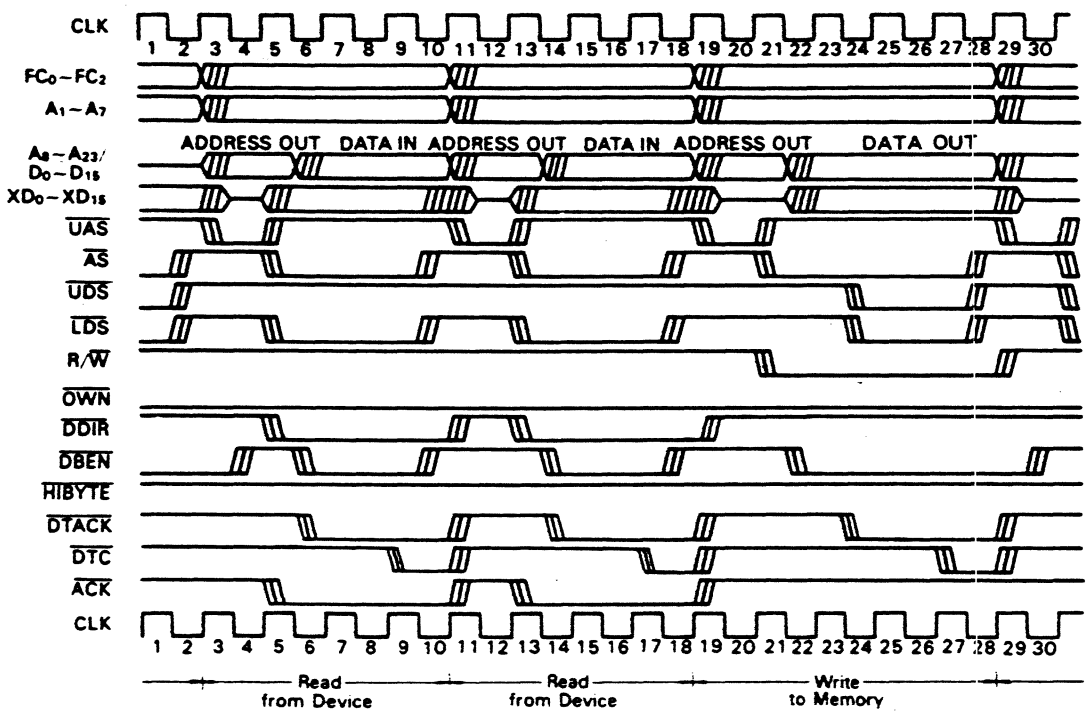
(NOTE) 1) This mode is identical to the dual address with 8 bit device as destination except that the device has a 16 bit port; i.e. both data strobes are asserted and data is on both halves of the bus. See the notes on the dual addressing mode with 8 bit device as source.

Figure 19 Dual Addressing Mode with 16 Bit Device as Destination (Read-Write Cycles)



(NOTE) 1) This mode is identical to the dual address with 8 bit device as source except that the device has a 16 bit port; i.e. both data strobes are asserted and data is on both halves of the bus. See the notes on the dual addressing mode with 8 bit device as source.

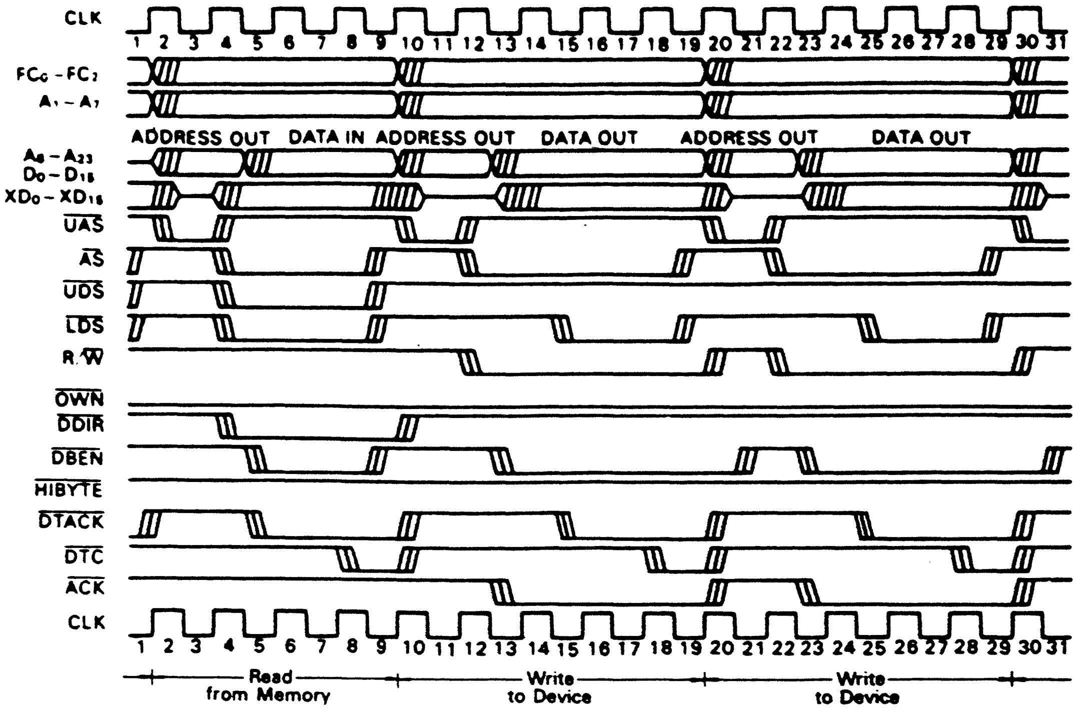
Figure 20 Dual Addressing Mode with 16 Bit Device as Source (Read-Write Cycles)



(NOTES) 1) In this mode the device is the source and memory is the destination.
2) In the dual addressing mode $\overline{\text{HIBYTE}}$ is not used since the DMAC will put the data on the correct half of the date bus.
3) The $\overline{\text{ACK}}$ timing is similar to the single addressing mode $\overline{\text{ACK}}$. The Read from Device $\overline{\text{ACK}}$ is asserted one clock into the cycle, and negated during the first CLK after the cycle has terminated. The Write to Device $\overline{\text{ACK}}$ is asserted one and one half clocks into the cycle (to avoid buffer conflicts) and removed the first CLK after the cycle has terminated. (See Figure 22)
4) Note that where Data Out is shown (clock 22) the data must be valid on the external data bus by the end of clock 23.

General Notes on Dual Addressing Mode
1) $\overline{\text{DBEN}}$ and $\overline{\text{DDIR}}$ are used in this mode with $\overline{\text{DBEN}}$ always changing on falling edge of CLK and $\overline{\text{DDIR}}$ always changing on rising edge of CLK.
2) Note that for consecutive reads from device or memory, $\overline{\text{DDIR}}$ need not be returned to the inactive (high) state between cycles.
3) Data In should be latched on the last CLK in the cycle (17 above).

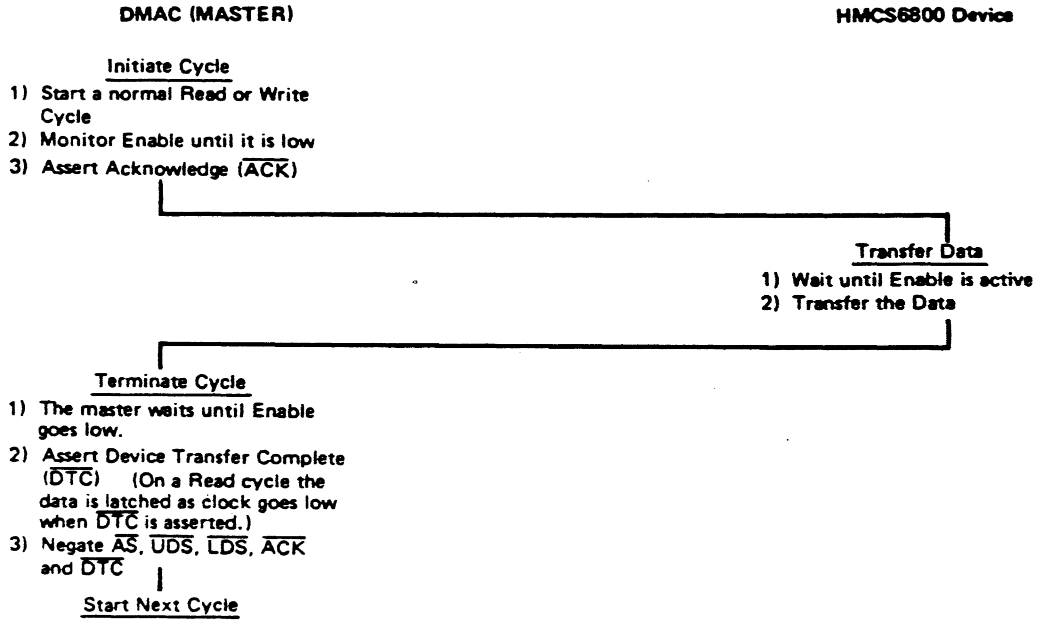
Figure 21 Dual Addressing Mode with 8 Bit Device as Source (Read-Write Cycles)



(NOTES) 1) These cycles are similar to the dual addressing mode 8 bit device as source with the exception that the device is now the destination. See the comments that refer to the dual mode 8 bit source.

2) Shown above in the Write to Device, the data strobes are not asserted until clock Δ15 (and Δ25) to allow data setup and travel time.

Figure 22 Dual Addressing Mode with 8 Bit Device as Destination (Read-Write Cycles)


DMAC (MASTER)

HMCS6800 Device

Initiate Cycle

1) Start a normal Read or Write Cycle
2) Monitor Enable until it is low
3) Assert Acknowledge ($\overline{\text{ACK}}$)

Transfer Data

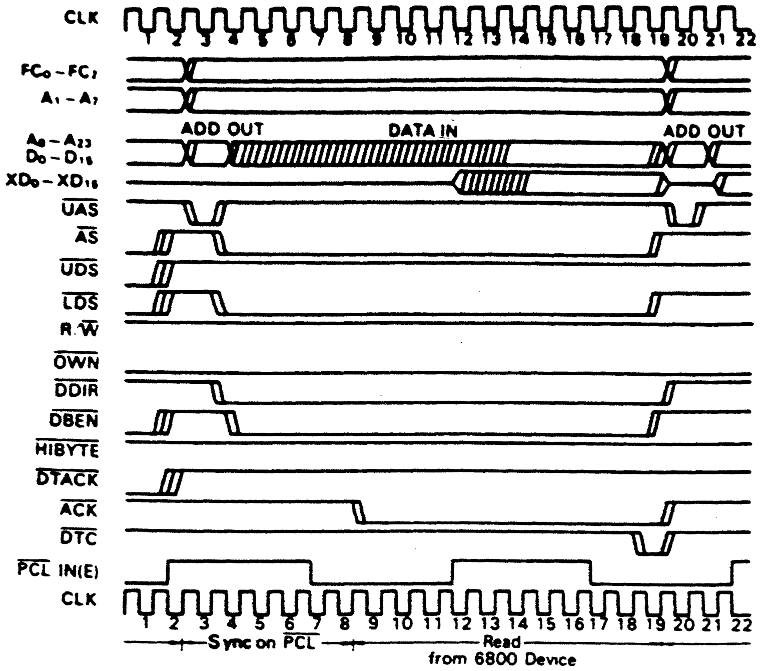
1) Wait until Enable is active
2) Transfer the Data

Terminate Cycle

1) The master waits until Enable goes low.
2) Assert Device Transfer Complete ($\overline{\text{DTC}}$) (On a Read cycle the data is latched as clock goes low when $\overline{\text{DTC}}$ is asserted.)
3) Negate $\overline{\text{AS}}$, $\overline{\text{UDS}}$, $\overline{\text{LDS}}$, $\overline{\text{ACK}}$ and $\overline{\text{DTC}}$

Start Next Cycle

Figure 23 HMCS6800 Cycle Flowchart



(NOTES) 1) The DMAC should latch the data during clock 19.
2) The logic should allow back to back operations on successive E pulses is possible.
3) The ACK low to E high time should be at least 250 ns worst case.
4) The clock reference shown above is the DMAC clock.
5) The E clock duty cycle shown above is an example. A 40% duty cycle is acceptable (4 up, 6 down like HD68000). The E clock must be low for a minimum of 5 clock cycles.

Figure 24 6800 Compatible Dual Addressing Mode (Read Cycle)

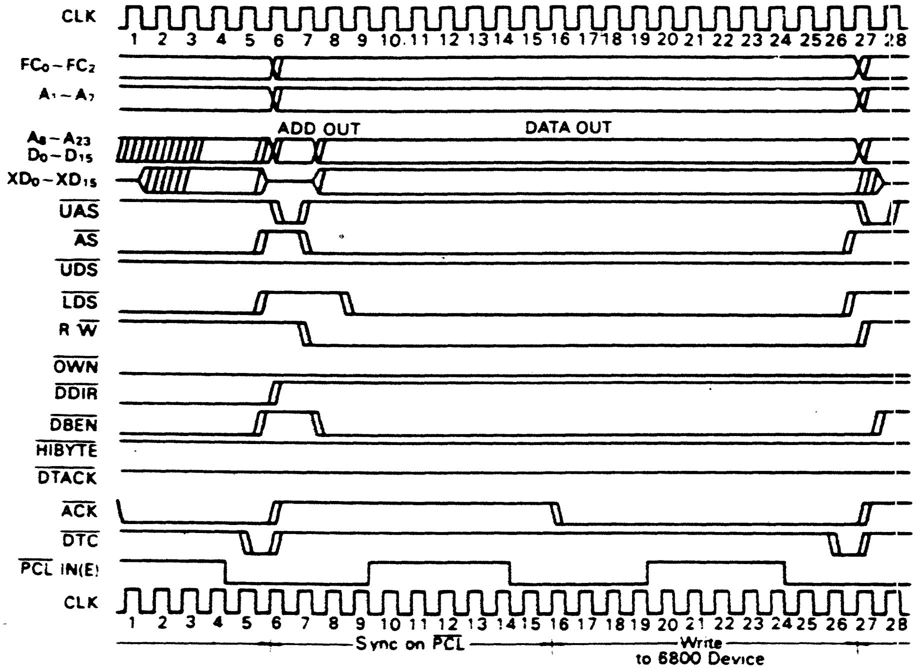


Figure 25 6800 Compatible Dual Addressing Mode (Write Cycle)

An Example of a Dual Address Transfer

This section contains an example of a dual address transfer using Table 7 of Dual-Address Sequencing. The table is reproduced here as Table 2. The transfer mode of this example is the following:

1. Device Port size = 8 bits
2. Operand size = Long Word (32 bits)
3. Memory to Device Transfer
4. Source (Memory) Counts up, Destination (Device) Counts Down
5. Memory Transfer Counter = 2

In this mode, a data transfer from the source (memory) is done according to the 6th row of Table 2, since the port size of the memory is always 16-bits. A data transfer to the destination (device) is done according to the 3rd row of Table 2. Table 3 shows the data transfer sequence.

The memory map of this example is shown in Table 4. The operand consists of BYTE A through BYTE D in memory of Table 4. Prior to the transfer, MAR and DAR are set to 00000012 and 00000108 respectively. The operand is transferred to the 8 bit port device according to the order of transfer number in Table 3.

Table 2 Dual-Address Sequencing (Table 7)

Row No.	Port Size	Operand Size	Part Size	Operand Part Addresses	Address Increment		
					+	=	-
1	8	BYTE	BYTE	A	+2	0	-2
2	8	WORD	BYTE	A, A+2	+4	0	-2
③	8	LONG	BYTE *4	A, A+2, A+4, A+6 *3 *5 *7 *8	+8	0	-8 *10
4	16	BYTE	PACK	A	+P	0	-P
5	16	WORD	WORD	A	+2	0	-2
⑥	16	LONG	WORD *2	A, A+2 *1 *6	+4 *9	0	-4

Table 3 An Example of a Data Transfer for One Operand

SRC: Source (Memory), DST Destination (Device), HR: Holding Register (DMAC Internal Reg.)

Transfer No.	Data Transfer	Address Output	Data Size on Bus	DMAC Registers after Transfer		Comment
				MAR	DAR	
0	–	–	–	00000012	00000108	Initial Register Setting
1	SRC → HR	00000012 *1	WORD *2	00000014	00000108	Higher order 16 bits of operand is fetched.
2	HR → DST	00000108 *3	BYTE *4	00000014	0000010A	Higher order 16 bits of operand is transferred.
3	HR → DST	0000010A *5	BYTE *4	00000014	0000010C *10	
4	SRC → HR	00000014 *6	WORD *2	00000016 *9	0000010C	Lower order 16 bits of operand is fetched
5	HR → DST	0000010C *7	BYTE *4	00000016	0000010E	Lower order 16 bits of operand is transferred.
6	HR → DST	0000010E *8	BYTE *4	00000016	00000110 *10	
6'	–	–	–	00000016	00000110	MAR, DAR are pointing the next operand addresses when the transfer is complete.

Mode: Port size = 8, Operand size = Long Word, Memory to Device, Source (Memory) Counts Up, Destination (Device) Counts Down

Table 4 Memory Map for the Example of the Data Transfer

ADDRESS			ADDRESS
00000010			00000011
00000012	BYTE A *1	BYTE B *1	00000013
00000014	BYTE C *6	BYTE D *6	00000015
00000016			00000017

Source (Memory)

ADDRESS			
00000106			00000107
00000108	BYTE A *3		00000109
0000010A	BYTE B *5		0000010B
0000010C	BYTE C *7		0000010D
0000010E	BYTE D *8		0000010F
00000110			00000111

Destination (Device)

SINGLE ADDRESSING MODE

Implicitly addressed devices do not require addressing of data register before data may be transferred. Transfers between memory and these devices are controlled by the request/acknowledge protocol. Such peripherals require only one bus cycle to transfer data between themselves and memory, and the DMAC internal holding register is not used. Because only the memory is addressed during a data transfer, this protocol is also called single-address.

Device with $\overline{\text{ACK}}$ Transfers

Under this protocol, the device is not explicitly addressed and communication is performed with a two signal request/acknowledge handshake. When a request is generated using the request method programmed in the control registers, the DMAC obtains the bus and responds with acknowledge. The DMAC asserts all HMCS68000 bus control signals needed for the transfer.

When the transfer is from memory to a device, data is valid when $\overline{\text{DTACK}}$ is asserted and remains valid until the data strobes are negated. The assertion of $\overline{\text{DTC}}$ from the DMAC may be used to latch the data, as the data strobes are not removed until 1/2 clock after the assertion of $\overline{\text{DTC}}$.

When the transfer is from device to memory, data must be valid on the HMCS68000 bus before the DMAC asserts the data strobes. The data strobes are asserted one clock period after $\overline{\text{ACK}}$ is asserted. Further definition of this protocol is explained in Figures 26, 27 and timing diagrams in Figures 28 and 29.

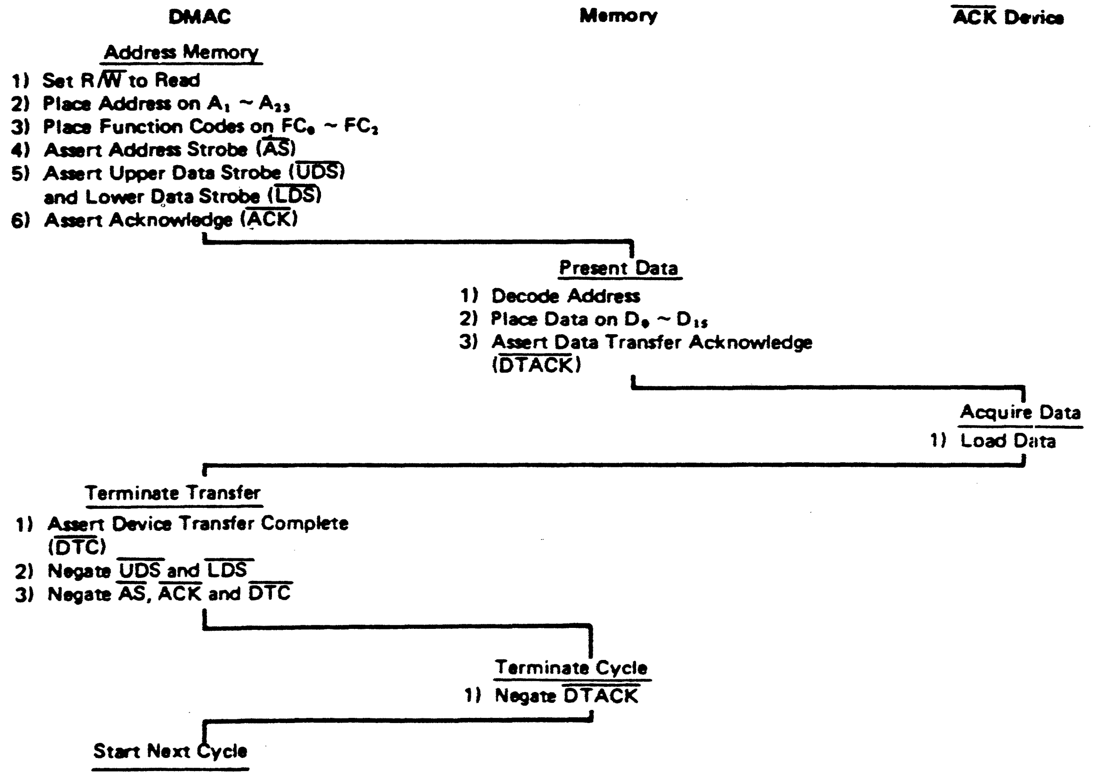

DMAC
Memory
$\overline{\text{ACK}}$ Device
Address Memory
1) Set R/$\overline{\text{W}}$ to Read
2) Place Address on $A_1 \sim A_{23}$
3) Place Function Codes on $FC_0 \sim FC_2$
4) Assert Address Strobe ($\overline{\text{AS}}$)
5) Assert Upper Data Strobe ($\overline{\text{UDS}}$) and Lower Data Strobe ($\overline{\text{LDS}}$)
6) Assert Acknowledge ($\overline{\text{ACK}}$)
Present Data
1) Decode Address
2) Place Data on $D_0 \sim D_{15}$
3) Assert Data Transfer Acknowledge ($\overline{\text{DTACK}}$)
Acquire Data
1) Load Data
Terminate Transfer
1) Assert Device Transfer Complete ($\overline{\text{DTC}}$)
2) Negate $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$
3) Negate $\overline{\text{AS}}$, $\overline{\text{ACK}}$ and $\overline{\text{DTC}}$
Terminate Cycle
1) Negate $\overline{\text{DTACK}}$
Start Next Cycle

Figure 26 Word from Memory to Device with $\overline{\text{ACK}}$

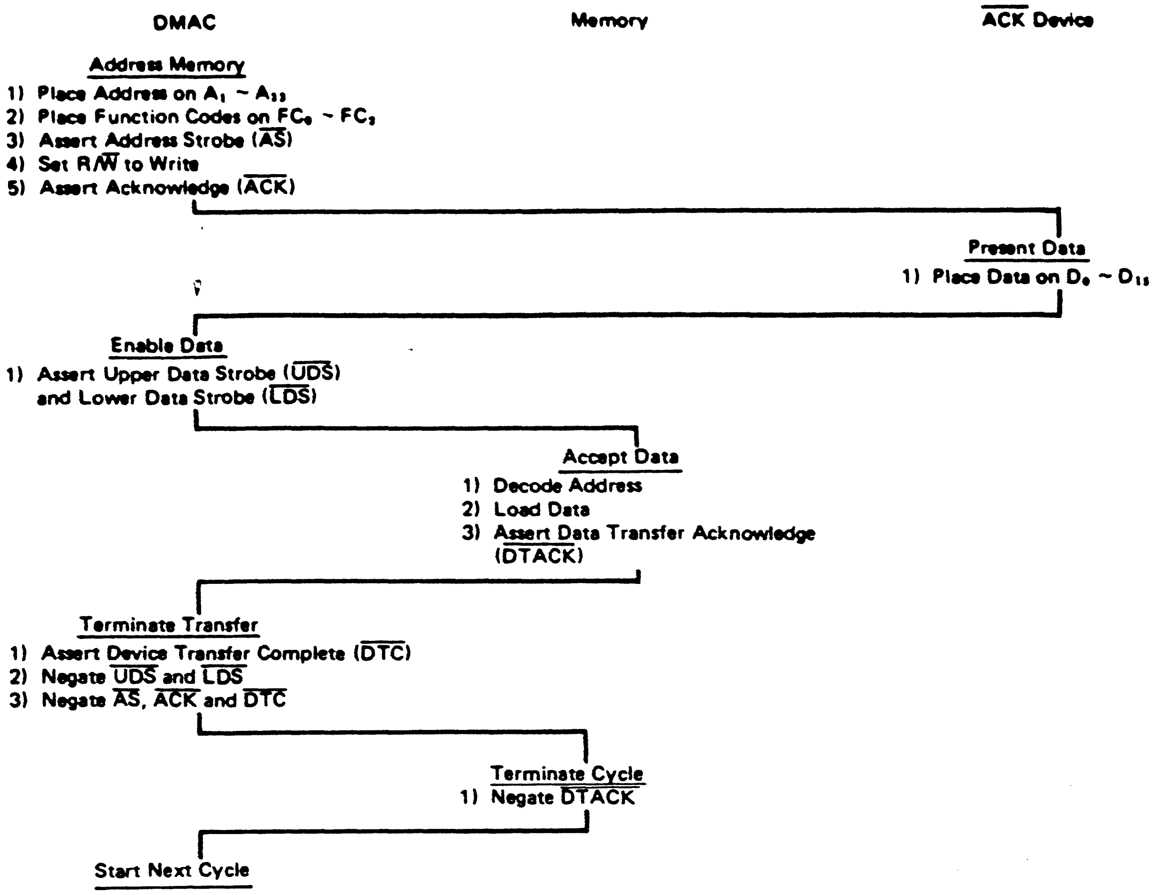
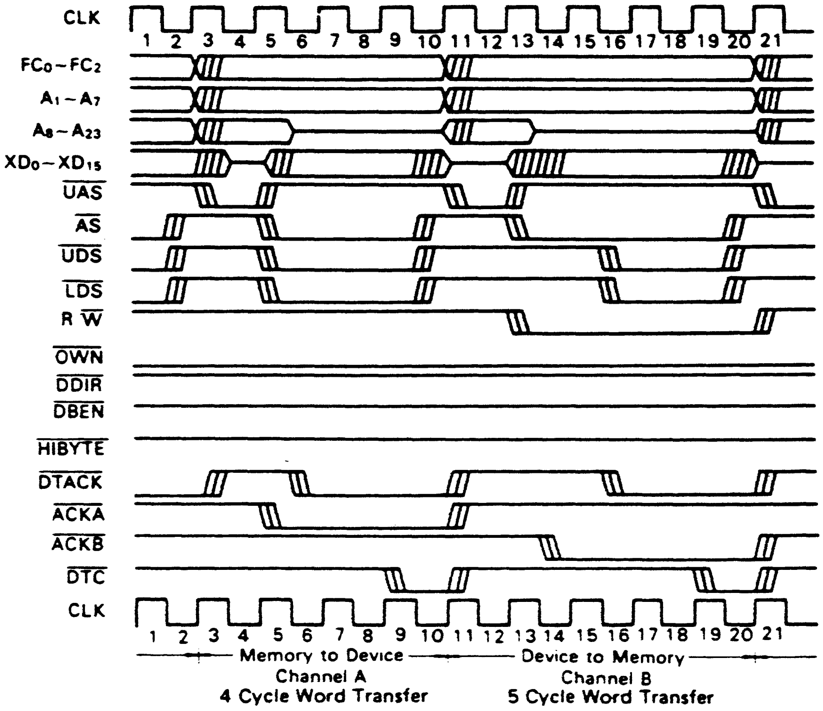
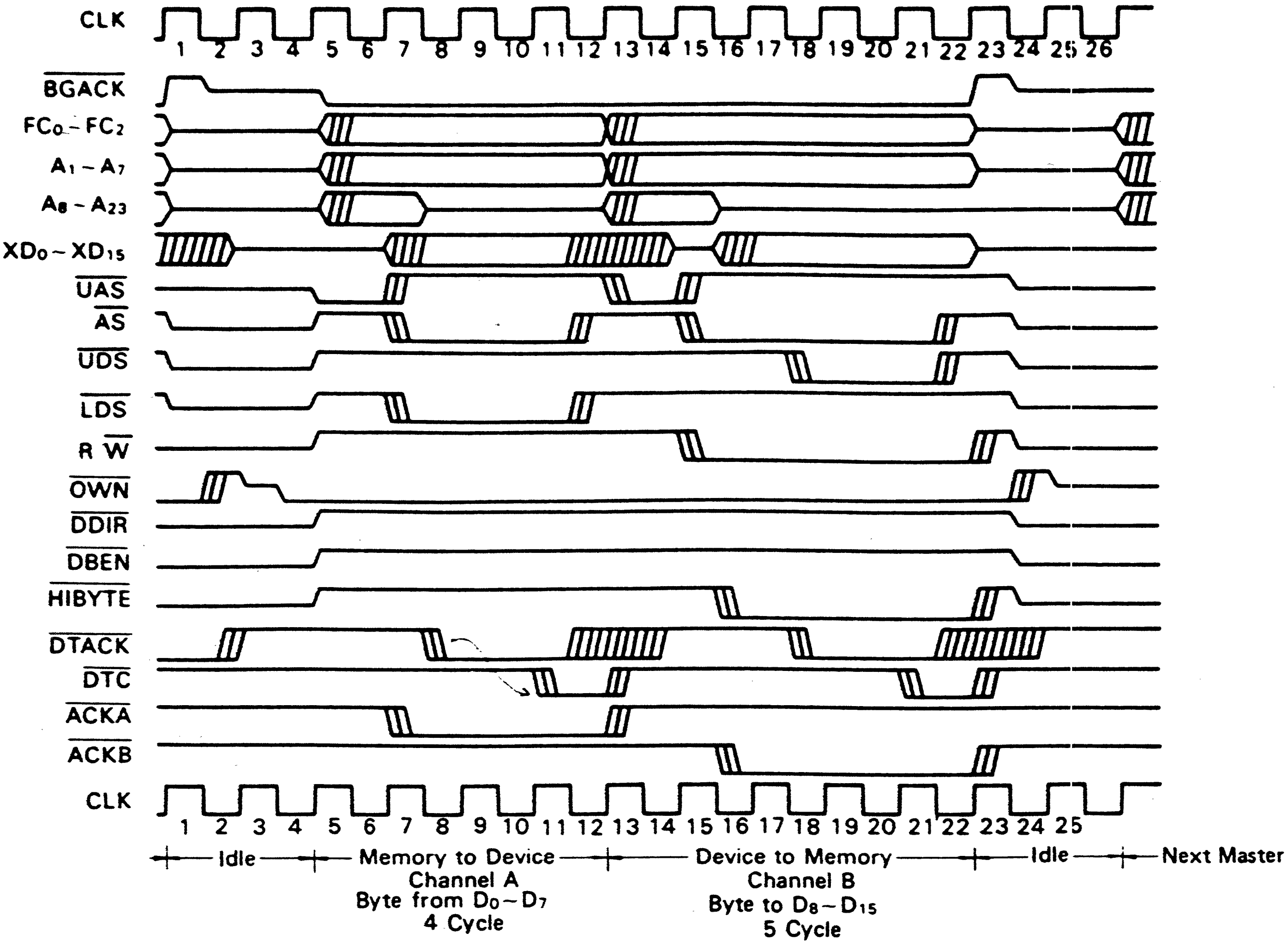

DMAC
Memory
$\overline{\text{ACK}}$ Device
Address Memory
1) Place Address on $A_1 \sim A_{23}$
2) Place Function Codes on $FC_0 \sim FC_2$
3) Assert Address Strobe ($\overline{\text{AS}}$)
4) Set R/$\overline{\text{W}}$ to Write
5) Assert Acknowledge ($\overline{\text{ACK}}$)
Present Data
1) Place Data on $D_0 \sim D_{15}$
Enable Data
1) Assert Upper Data Strobe ($\overline{\text{UDS}}$) and Lower Data Strobe ($\overline{\text{LDS}}$)
Accept Data
1) Decode Address
2) Load Data
3) Assert Data Transfer Acknowledge ($\overline{\text{DTACK}}$)
Terminate Transfer
1) Assert Device Transfer Complete ($\overline{\text{DTC}}$)
2) Negate $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$
3) Negate $\overline{\text{AS}}$, $\overline{\text{ACK}}$ and $\overline{\text{DTC}}$
Terminate Cycle
1) Negate $\overline{\text{DTACK}}$
Start Next Cycle

Figure 27 Word from Device with $\overline{\text{ACK}}$ to Memory



(NOTES) 1) These cycles are identical to the 8 bit transfers with the exception that all transfers are 16 bits (word) so that both data strobes are always asserted and $\overline{\text{HIBYTE}}$ is always inactive. See the comments on single addressing mode with 8 bit devices.
2) The A and B on the $\overline{\text{ACK}}$ signals are there to distinguish two different channels. The actual channel numbers could be any of 0 to 3.

Figure 28 Single Addressing Mode with 16 Bit Devices as Sources and Destinations (Read-Write Cycles)



(NOTES)
1) Any signal change shown before clock 4 is due to the previous master (MPU or other DMAC).
2) All signals except $\overline{\text{OWN}}$ tristate during the clock 2 after the end of the last DMAC cycle. $\overline{\text{OWN}}$ tristates one half clock later.
3) $\overline{\text{DDIR}}$ and $\overline{\text{DBEN}}$ remain in the inactive state during single address mode cycles.
4) A four (4) cycle transfer is the minimum cycle length for a Memory to Device transfer.
5) A five (5) cycle transfer is the minimum cycle length for a Device to Memory transfer.
6) $\overline{\text{HIBYTE}}$ (when used as fold signal) is used to gate the high bus data to the low bus during a Memory to Device transfer, and low bus data to the high bus during a Device to Memory transfer.
7) Note that if the transfer is 8 bits wide, only one data strobe (and possibly $\overline{\text{HIBYTE}}$) is asserted.
8) Address bits 8 through 23 are shown in tristate beginning clock 8 (and 16). In this mode only they can be driven similar to A_1 through A_7 if need be.

The followings are notes on the Memory to Device transfer.
9) $\overline{\text{ACK}}$ is not asserted until one clock into the cycle (clock 7) due to the fact that it may have just been negated (clock 5). It must remain negated for at least one clock.
10) DTC is asserted one clock before the end of the cycle to signal the peripheral that the data is valid and should be latched at this time. It also indicates that a successful transfer is being completed (no Berr, Retry, etc.).

The followings are notes on the Device to Memory transfer.
11) $\overline{\text{ACK}}$ cannot be asserted until one and one half clocks into the cycle (clock 16) to allow the R/$\overline{\text{W}}$ signal to settle in the write mode (clock 15). R/$\overline{\text{W}}$ cannot be asserted until one clock into the cycle (clock 15) because of buffer collisions.
12) The data strobes cannot be asserted until clock 18 to allow the data to become valid at the memory (data setup).
13) The A and B on the $\overline{\text{ACK}}$ signals are there to distinguish two different channels. The actual channel numbers could be any of 0 to 3.

Figure 29 Single Addressing Mode with 8 Bit Devices as Sources and Destinations

Device with $\overline{\text{ACK}}$ and $\overline{\text{READY}}$ Transfers

Under this protocol, the device is not explicitly addressed and communication is performed using a three signal request/acknowledge/ready handshake. The ready input to the DMAC is provided by the $\overline{\text{PCL}}$ line, the use of this protocol forces it to be an active low input. When a request is generated using the request method programmed in the control registers, the DMAC obtains the bus and asserts acknowledge to notify the device that the transfer is to take place. The DMAC asserts all HMCS68000 bus control signals needed for the transfer and holds them until the device responds with $\overline{\text{READY}}$. After $\overline{\text{READY}}$ is received the bus cycle terminates normally.

When the transfer is from memory to a device, data is valid when $\overline{\text{DTACK}}$ is asserted and is valid until the data strobes are negated. The assertion of $\overline{\text{DTC}}$ from the DMAC may be used to latch the data, as the data strobes are not removed until 1/2 clock after the assertion of $\overline{\text{DTC}}$.

When the transfer is device to memory, data must be valid on the HMCS68000 bus before the DMAC asserts the data strobes. The data strobes are held asserted until the device asserts $\overline{\text{READY}}$. Further definition of this protocol is explained in Figures 30, 31 and the timing diagrams in Figure 32.

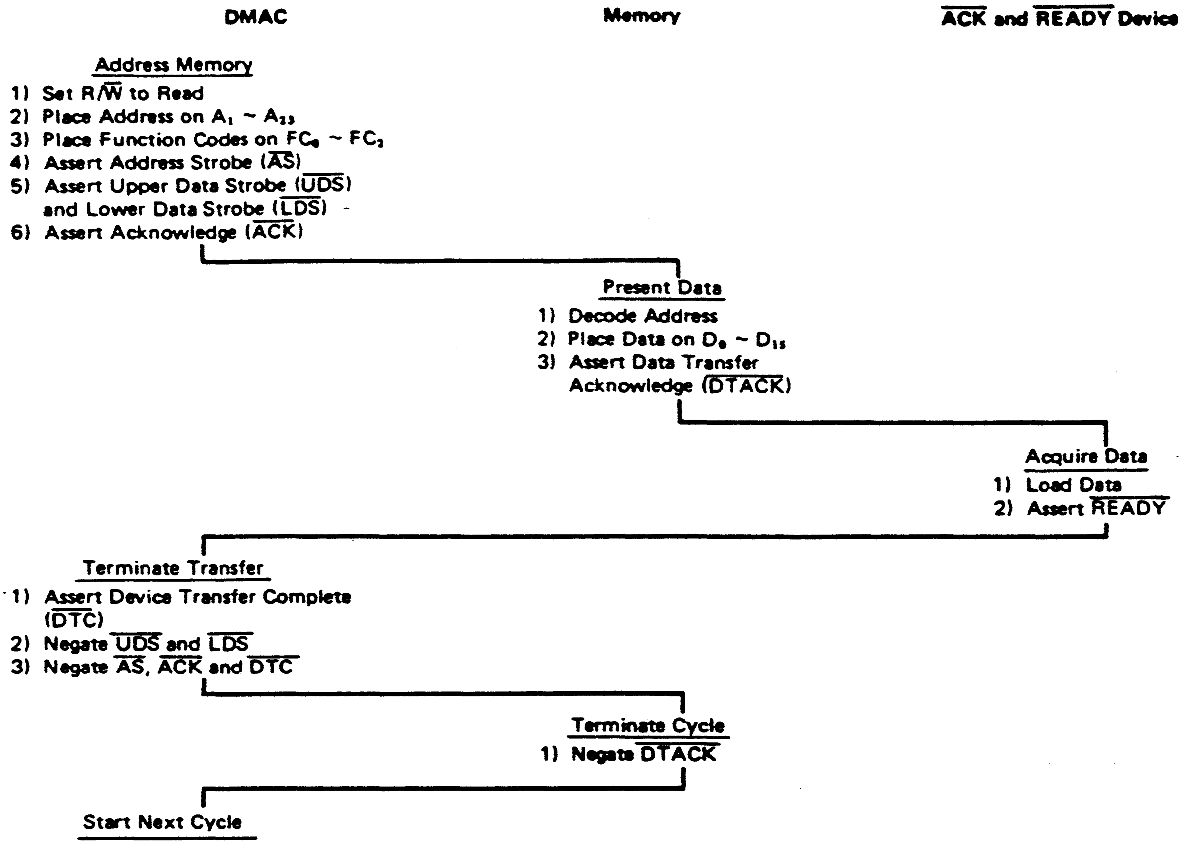
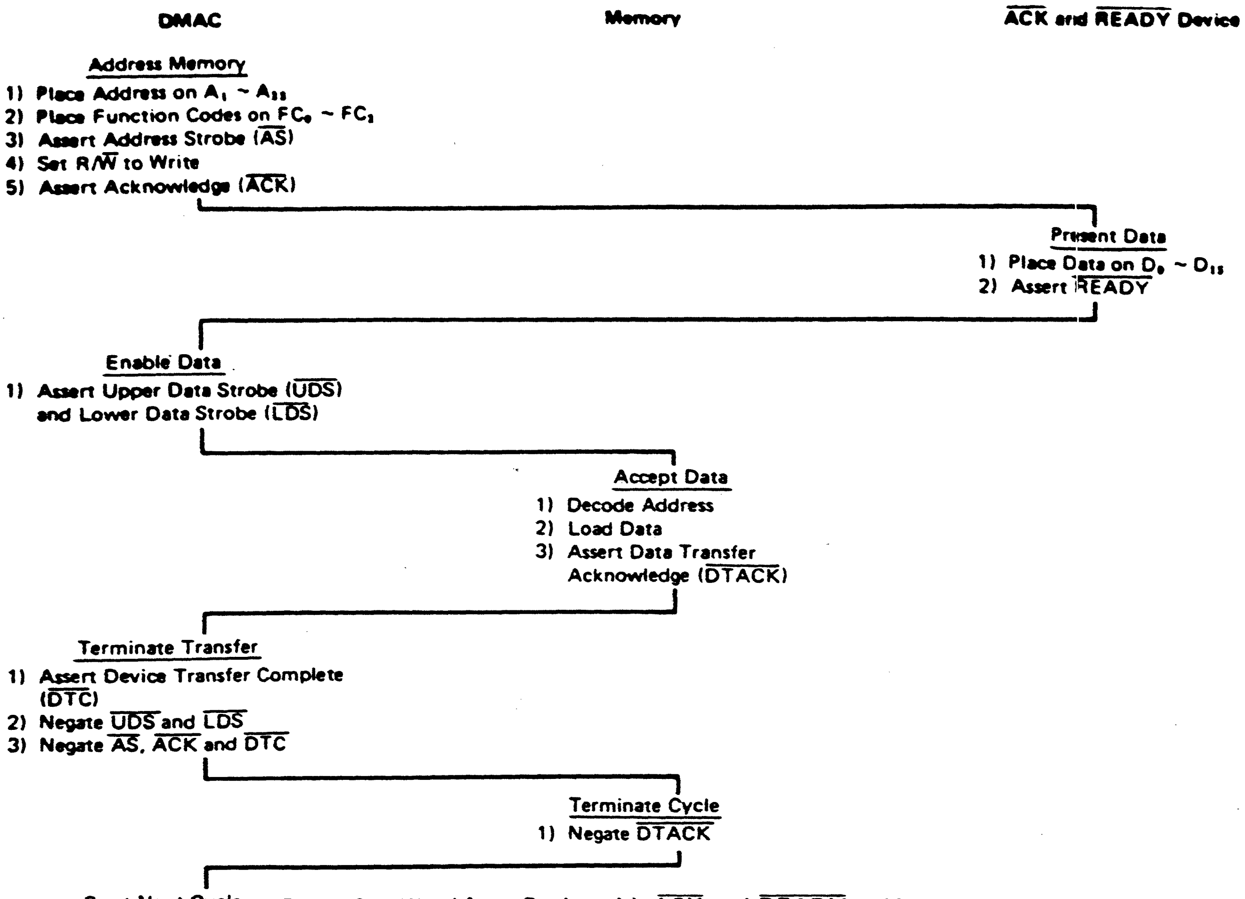

DMAC
Memory
$\overline{\text{ACK}}$ and $\overline{\text{READY}}$ Device
Address Memory
1) Set R/$\overline{\text{W}}$ to Read
2) Place Address on A_1 ~ A_{23}
3) Place Function Codes on FC_0 ~ FC_2
4) Assert Address Strobe ($\overline{\text{AS}}$)
5) Assert Upper Data Strobe ($\overline{\text{UDS}}$) and Lower Data Strobe ($\overline{\text{LDS}}$)
6) Assert Acknowledge ($\overline{\text{ACK}}$)
Present Data
1) Decode Address
2) Place Data on D_0 ~ D_{15}
3) Assert Data Transfer Acknowledge ($\overline{\text{DTACK}}$)
Acquire Data
1) Load Data
2) Assert $\overline{\text{READY}}$
Terminate Transfer
1) Assert Device Transfer Complete ($\overline{\text{DTC}}$)
2) Negate $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$
3) Negate $\overline{\text{AS}}$, $\overline{\text{ACK}}$ and $\overline{\text{DTC}}$
Terminate Cycle
1) Negate $\overline{\text{DTACK}}$
Start Next Cycle

Figure 30 Word from Memory to Device with $\overline{\text{ACK}}$ and $\overline{\text{READY}}$



DMAC

Memory

$\overline{\text{ACK}}$ and $\overline{\text{READY}}$ Device

Address Memory

1) Place Address on $A_1 \sim A_{23}$
2) Place Function Codes on $FC_0 \sim FC_2$
3) Assert Address Strobe ($\overline{\text{AS}}$)
4) Set R/$\overline{\text{W}}$ to Write
5) Assert Acknowledge ($\overline{\text{ACK}}$)

Present Data

1) Place Data on $D_0 \sim D_{15}$
2) Assert $\overline{\text{READY}}$

Enable Data

1) Assert Upper Data Strobe ($\overline{\text{UDS}}$) and Lower Data Strobe ($\overline{\text{LDS}}$)

Accept Data

1) Decode Address
2) Load Data
3) Assert Data Transfer Acknowledge ($\overline{\text{DTACK}}$)

Terminate Transfer

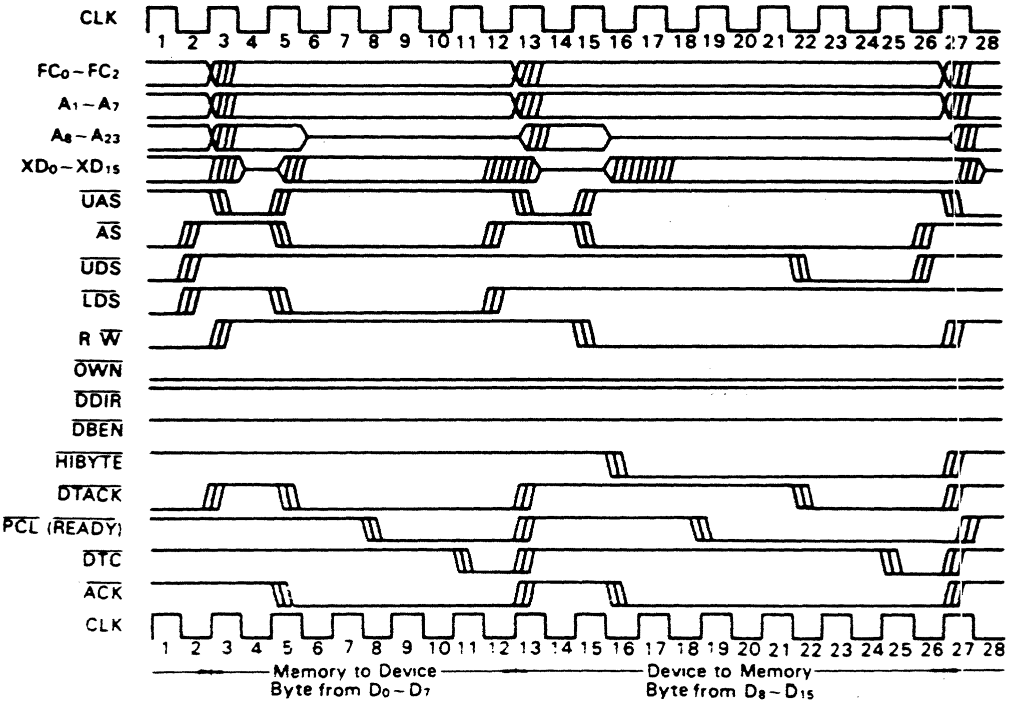
1) Assert Device Transfer Complete ($\overline{\text{DTC}}$)
2) Negate $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$
3) Negate $\overline{\text{AS}}$, $\overline{\text{ACK}}$ and $\overline{\text{DTC}}$

Terminate Cycle

1) Negate $\overline{\text{DTACK}}$

Start Next Cycle

Figure 31 Word from Device with $\overline{\text{ACK}}$ and $\overline{\text{READY}}$ to Memory



(NOTES)
1) With the exception of the notes below, these cycles are identical to the normal single addressing mode cycles. See the comments on the 8 bit single addressing mode transfer.
2) In the Memory to Device transfer only, the $\overline{\text{READY}}$ ($\overline{\text{PCL}}$) line is used as a "second $\overline{\text{DTACK}}$" i.e. both $\overline{\text{READY}}$ and $\overline{\text{DTACK}}$ are required to terminate the cycle.
3) In the Device to Memory transfer, the $\overline{\text{READY}}$ input is used to delay the assertion of the data strobes. Once $\overline{\text{READY}}$ is detected, the data strobes are asserted and $\overline{\text{DTACK}}$ is sampled to terminate the cycle. $\overline{\text{AS}}$ is asserted at the beginning of the cycle as usual.

Figure 32 Single Addressing Mode with 8 Bit Devices as Sources and Destinations with $\overline{\text{PCL}}$ Used as a $\overline{\text{READY}}$ Input (Read-Write Cycles)

(2) OPERANDS AND ADDRESSING

Three factors enter into how the actual data is handled: port size, operand size and address sequencing.

PORT SIZE

The DCR is used to program the device port size

DPS Device Port Size
0 8 bit port
1 16 bit port

The port size is the number of bits of data which the device can transfer in a single bus cycle. During a DMAC bus cycle, a 16-bit port transfers 16 bits of data on $D_0 \sim D_{15}$, while an 8-bit port transfers 8 bits of data, either on $D_0 \sim D_7$ or on $D_8 \sim D_{15}$. The memory is always assumed to have a port size of 16.

OPERAND SIZE

OCR is used to program the operand size.

SIZE Operand Size
00 Byte
01 Word
10 Long word
11 (undefined, reserved)

The operand size is the number of bits of data to be transferred to honor a single request. Multiple bus cycles may be required to transfer the operand through the device port. A byte operand consists of 8 bits of data, a word operand consists of 16 bits of data, a long word operand consists of 32 bits of data. The transfer counter counts the number of operands transferred.

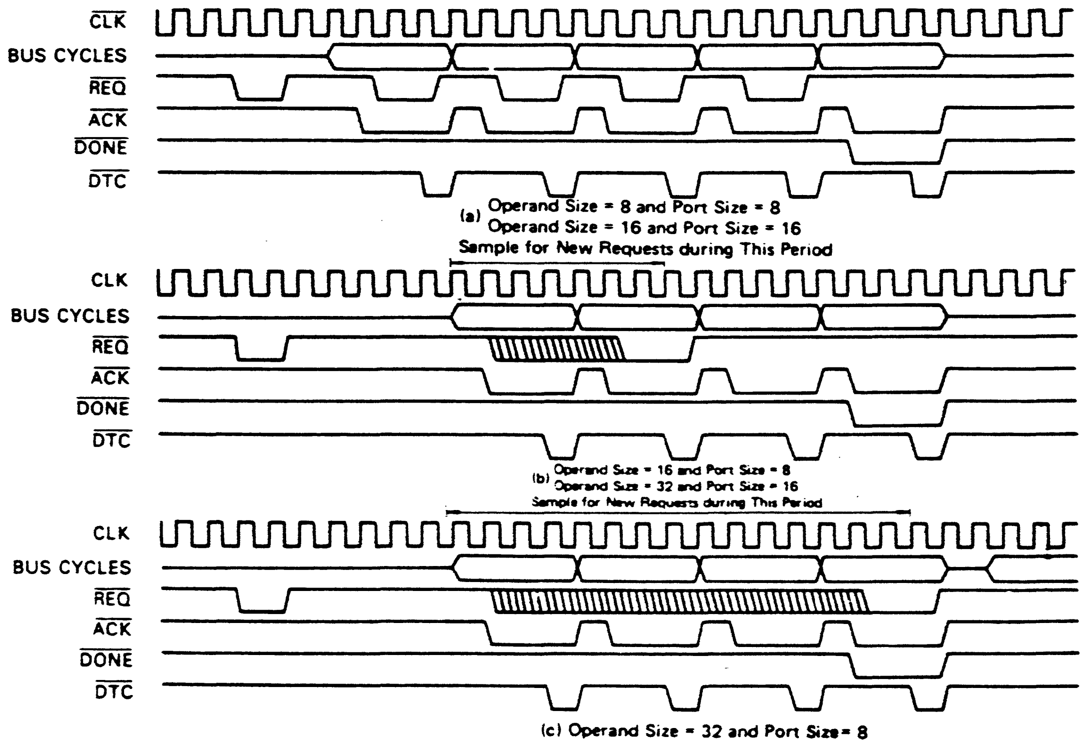
For single-address operations, the port size and the operand size must be the same. 68000 and 6800 type devices may not use byte operands when the port size is 16 bits and the request generation method is the request pin. (REQG = 10 or 11)

Table 5 Operation Combinations

Addressing	Device Type	Port	Operand			REQG*
			Byte	Word	LW	
Dual	68000, 6800	8	Yes	Yes	Yes	00, 01, 10, 11
Dual	68000, 6800	16	Yes	Yes	Yes	00, 01
Dual	68000, 6800	16	No	Yes	Yes	10, 11
Single	with $\overline{\text{ACK}}$ or $\overline{\text{ACK}}$ & $\overline{\text{READY}}$	8	Yes	No	No	00, 01, 10, 11
		16	No	Yes	No	00, 01, 10, 11

*Refer to Page 11.

ONLY DEVICE BUS CYCLES ARE SHOWN IN THESE TIMING DIAGRAMS


CLK
BUS CYCLES
REQ
ACK
DONE
DTC
Sample for New Requests during This Period
CLK
BUS CYCLES
REQ
ACK
DONE
DTC
Sample for New Requests during This Period
CLK
BUS CYCLES
REQ
ACK
DONE
DTC

(a) Operand Size = 8 and Port Size = 8
Operand Size = 16 and Port Size = 16

(b) Operand Size = 16 and Port Size = 8
Operand Size = 32 and Port Size = 16

(c) Operand Size = 32 and Port Size = 8

(NOTES)
1) The above cycles reflect both dual and single addressing mode.
2) In the dual addressing case, the memory references are not shown above, only the device references. Therefore all cycles have $\overline{\text{ACK}}$ asserted.
3) Note that when the operand size exceeds the port size, the DMAC should sample the request pins (new priority check) until the $\overline{\text{DTC}}$ of the last bus cycle has occurred.

Figure 33 Request and Acknowledge Generation vs. Operand Size

ADDRESS SEQUENCING

The sequence of addresses generated depends upon the port size, operand size, whether the addresses are to count up, down, or not change and whether the transfer is explicitly or implicitly addressed. The Sequence Control Register is used to program the memory address count method and the device address count method.

MAC Memory address count
- 00 Memory address register does not count
- 01 Memory address register counts up
- 10 Memory address register counts down
- 11 (undefined, reserved)

DAC Device address register count
- 00 Device address register does not count
- 01 Device address register counts up
- 10 Device address register counts down
- 11 (undefined, reserved)

Single-Address Transfers

Single-address transfers require the device port size and the operand size to be equal. Address sequencing is determined by the port size and the sequence control register as shown in Table 6. If the operand size is byte, the memory address increment is one (1). If the operand size is word, the memory address increment is two (2). If the memory address register does not count, the memory address is unchanged after the transfer. If the memory address counts up, the increment is added to the memory address; if the memory address counts down, the increment is subtracted from the memory address. The memory address is changed after the operand is transferred.

Table 6 Single Address Sequencing

Port Size	Operand Size	Memory Address Increment		
		+	=	-
8	Byte	+1	0	-1
16	Word	+2	0	-2

Dual-Address Transfers

In dual-address operations, the operand size need no match the port size: Thus the transfer of an operand may require several transfers between device and memory. Each pair of transfers, between memory and DMAC and between DMAC and device, transfers a portion of the operand, called the operand part.

The addresses of the operand parts are in a linear increasing sequence. The step between the addresses of parts is two (2). The size of the parts is the minimum of the port size and operand size. The number of parts is the operand size divided by the port size. The address increment is added or subtracted after the operand is transferred.

If the port size is 16 bits, the operand size is byte, and the request generation method is auto request or auto request at a limited rate, the DMAC packs consecutive transfers. This means that word transfers are made from the associated address with an address increment of two (2). If the initial source address location contains a single byte, the first transfer is a byte transfer to the internal DMAC holding register, and subsequent transfers from the source are word transfers. If the initial destination location contains a single byte, the first transfer is a byte transfer from the internal DMAC holding register, and any remaining byte remains in the holding register. Likewise, if either the final source or destination location contains a single byte, only a byte transfer is done. Packing is not performed if the address does not count; each byte is transferred by a separate access to the same location.

(3) ADDRESS REGISTER OPERATION

The DMAC has three 32-bit address registers per channel: the memory address register (MAR), the device address register (DAR), and the base address register (BAR).

The MAR is used in all operations because of the assumption that all operations are between memory and a device. The MAR is sequenced as previously described. This register is either initialized before the channel operation is started, or is loaded during chaining or continue operations which are defined in a later section.

Table 7 Dual-Address Sequencing

Port Size	Operand Size	Part Size	Operand Part Address	Address Increment		
				+	=	-
8	Byte	Byte	A	+2	0	-2
8	Word	Byte	A, A+2	+4	0	-4
8	Long	Byte	A, A+2, A+4, A+6	+8	0	-8
16	Byte	Pack	A	+P	0	-P
16	Word	Word	A	+2	0	-2
16	Long	Word	A, A+2	+4	0	-4

P = 1 if packing is not done
 = 2 if packing is done

The DAR is used to address devices or memory in dual-address operations. It is initiated before starting the channel operation, and is sequenced as previously described.

The BAR register is used only in chaining or continue operations. It is sequenced only in regard to chaining operations.

(4) FUNCTION CODE REGISTER OPERATIONS

There are three function code registers per channel: the memory function code register, the device function code register, and the base function code register. The function code registers correspond to the address registers and are output on $FC_0 \sim FC_2$ when the corresponding address register provides the address for a DMA bus cycle.

(5) TRANSFER COUNT REGISTER OPERATION

The DMAC has two 16-bit transfer counter registers per channel: the memory transfer counter, and the base transfer counter.

The memory transfer counter is used in all operations to count the number of operands transferred in a block. The memory transfer counter is decremented by one as each operand is transferred. This register is either initialized before the channel operation is started, or is loaded during chaining or continue operations.

Both the memory transfer counter and the base transfer counter have a terminal count of zero (0). If either register is initialized or loaded with a terminal count when the channel is configured to use that register, a count error is signaled.

- **INITIATION AND CONTROL OF CHANNEL OPERATION**

The Channel Control Register provides mechanisms for starting, continuing, halting, or aborting an operation. It also controls the enabling of interrupts from a channel.

(1) OPERATION INITIATION

To initiate the operation of a channel the STR bit of the CCR is set to start the operation. Setting the STR bit causes the immediate activation of the channel, the channel will be ready to accept requests immediately. The channel initiates the operation by cleaning the STR bit and setting the channel active bit in the CSR. Any pending requests are cleared, and the channel is then ready to receive requests for the new operation. If the channel is configured for an illegal operation, the configuration error is signaled, and no channel operation is run. The illegal operations include the selection of any of the options marked "(undefined, reserved)". If the operation is dual-address, the device address register should have been previously initialized. The channel cannot be started if any of the ACT, COC, BTC, NDT or ERR bits is set in the CSR. In this case, the channel signals the operation timing error.

If the operation is unchained, the memory address register and the memory transfer counter should have been previously initialized.

If the operation is chained, the base address register, and the base transfer counter should have been previously initialized.

(2) OPERATION CONTINUATION

The continue bit (CNT) allows multiple blocks to be transferred in unchained operations. The CNT bit is set in order to continue the current channel operation. If an attempt is made to continue a chained operation, a configuration error is signaled. The base address register and base transfer counter should have been previously initialized.

The continue bit may be set as the channel is started or while the channel is still active. The operation timing error bit is signaled if a continuation is otherwise attempted.

(3) HALT

The CCR has a halt bit which allows suspension of the operation of the channel. If this bit is set, a request may still be generated and recognized, but the DMAC does not attempt to acquire the bus or to make transfers for the halted channel. When this bit is reset, the channel resumes operation and services any request that may have been received while the channel was halted.

(4) SOFTWARE ABORT

The CCR has a software abort bit (SAB) which allows the current operation of the channel to be aborted. The writing of a one (1) into the SAB bit causes a channel abort error to be signaled. When the CCR is read, the SAB always reads as zero (0).

(5) INTERRUPT ENABLE

The CCR has an interrupt enable bit (INT) which allows the channel to request interrupts on the completion of block transfers or on the termination of channel operations. If INT is set, the channel can request interrupts. If it is clear, the channel may not request interrupts.

- **BLOCK TERMINATION**

As part of the transfer of an operand, the DMAC decrements the memory transfer counter. If this counter is decremented to the terminal count, the transfer counter is exhausted and the operand is the last operand of the block. The channel operation is complete if the operation is unchained and there is no continuation, or if the operation is chained and the chain is exhaused. The DMAC notifies the device of channel completion via the $\overline{\text{DONE}}$ output. When the transfer has been completed, the ACT bit of the CSR is cleared, and the COC bit is set.

The occurrence of a bus exception during a bus cycle being run for a channel, or the occurrence of some error in the channel terminates the block transfer and the channel operation. The bit of the CER corresponding to the error being signal is set. The ACT of the CSR is cleared and the COC and ERR bits are set.

(1) CHANNEL STATUS REGISTER

The channel status register contains the status of the channel. The register is cleared by writing a one (1) into each bit of the register to be cleared. Those bits positions which contain a zero (0) in the write data remain unaffected.

COC

The channel operation complete bit is set if the DMA transfer has completed. The COC bit is set following the termination, whether successful or not, of any DMA operation. This bit must be cleared in order to start another channel operation. This bit is cleared only by writing the channel status register or resetting the DMAC.

PCS

The peripheral status bit reflects the state of the $\overline{\text{PCL}}$ I/O line regardless of its programmed function. This bit is unaffected by write operations.

PCT

The peripheral control transition bit is set if an falling edge transition has occurred on the $\overline{\text{PCL}}$ line. This bit is cleard only by writing the channel status register or resetting the DMAC.

BTC

Block transfer complete is set when the memory transfer count is exhausted, the operation is unchained, and the continue bit is set. This bit must be cleared before another continuation is attempted, otherwise an operation timing error is signaled. This bit is cleared only by writing the channel status register or resetting the DMAC.

NDT

Normal device termination is set when the device terminates the channel operation by asserting the $\overline{\text{DONE}}$ line while the device was being acknowledged. This bit is cleared only by writing the channel status register or resetting the DMAC.

ERR

This bit is used to report the occurrence of error conditions. It is set if any errors have been signaled. This bit is cleared only by writing the channel status register or resetting the DMAC.

ACT

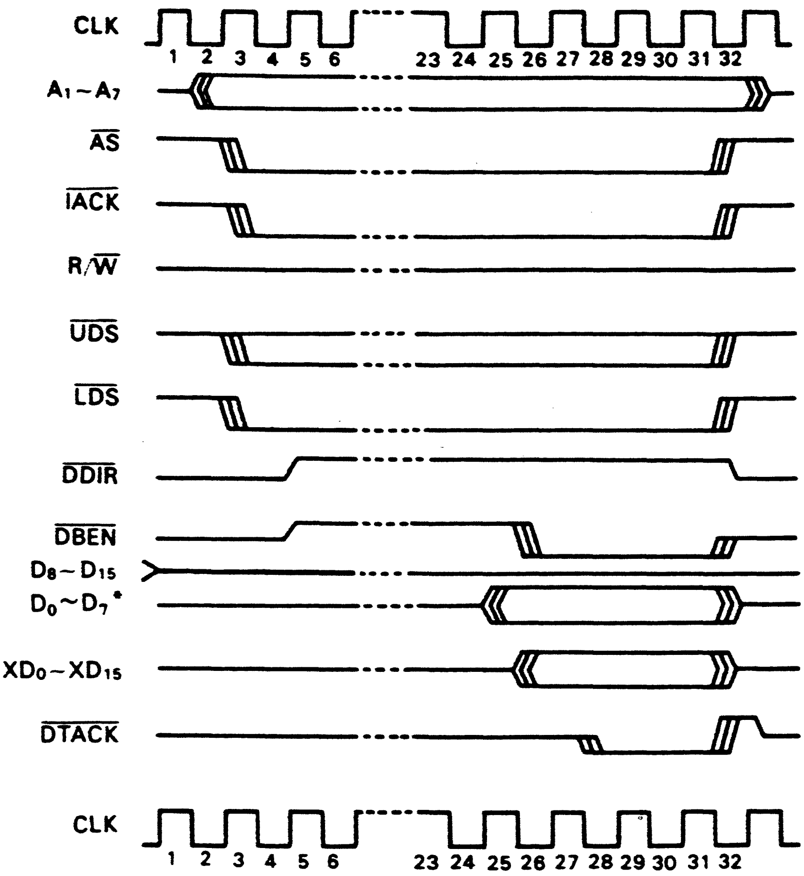
This is the channel active bit. It is asserted after the channel has been started. The bit remains set until the channel operation terminates. This bit is unaffected by write operations.

(2) INTERRUPTS

The INT bit of the CCR determines if an interrupt can be generated. The interrupt request is generated if INT is set and the bits COC or BTC are set in the CSR or the PCT bit is set and the $\overline{PCL}$ line is programmed to be an interrupt input.

If a channel has an interrupt request, the DMAC makes an interrupt request by asserting the $\overline{IRQ}$ output. If the DMAC has an interrupt request pending, and receives an $\overline{IACK}$ from the MPU the DMAC provides an interrupt vector. If multiple channels have interrupt requests pending, the determination of which channel presents its interrupt vector is made using the same priority scheme defined for channel operations.

The interrupt vector returned to the MPU comes from either the normal or the error interrupt vector register. The normal interrupt register is used unless the ERR bit of CSR is set, in which case the error interrupt vector register is used. The content of the interrupt vector register is placed on $D_0 \sim D_7$, and $\overline{DTACK}$ is asserted to indicate that the vector is on the data bus. If a reset bus exception occurs, all interrupt vector registers are set to $0F (binary 00001111), the value of the uninitialized interrupt vector.



* MPX $A_8 \sim A_{23}/D_8 \sim D_{15}$ pins
Interrupt Vector Register is Output.

(NOTES) 1) This cycle is similar to the chip select cycle except it is triggered by $\overline{IACK}$. See the notes on the chip select cycle.
2) $\overline{IACK}$ will be negated within one clock after $\overline{AS}$ is negated.
3) The clock referenced above is the CPU clock.

Figure 34 MPU $\overline{IACK}$ Cycle to DMAC

(3) MULTIPLE BLOCK OPERATION

When the memory transfer counter is exhausted, there are further blocks to be transferred if the channel is chained and the chain is not exhausted. The DMAC provides the reinitialization of the memory address register and the memory transfer counter in these cases.

CONTINUED OPERATIONS

When the memory transfer counter is exhausted and the continue bit of the CCR is set, the DMAC performs a continuation of the channel operation. The base address, base function code, and base transfer count registers are copied into the memory address, memory function code, and memory transfer count registers. The block transfer complete (BTC) bit of the CSR is set, the continue bit is reset, and the channel begins a new block transfer. If the memory transfer counter is loaded with a terminal count, the count error is signaled.

ARRAY CHAINING

This type of chaining uses an array in memory consisting of memory addresses and transfer counts. Each entry in the array is six bytes long and, consists of four bytes of address followed by two bytes of transfer count. The beginning address of this array is in the base address register, and the number of entries in the array is in the base transfer counter. Before starting any block transfers, the DMAC fetches the entry currently pointed to by the base address register. The address information is placed in the memory address register, and the count information is placed in the memory transfer counter. As each chaining entry is fetched, the base transfer counter is decremented by one. After the chaining entry is fetched, the base address register is incremented to point the next entry. When the base transfer counter reaches a terminal count, the chain is exhausted, and the entry just fetched determines the last block of the channel operation.

The memory format for supporting the Array Chaining is shown in Figure 35. The array must start at an even address, or the entry fetch results is an address error. If a terminal count is loaded into the memory transfer counter, the count error is signaled. Since the base registers may be read by the MPU, appropriate error recovery information is available should the DMAC encounter an error anywhere in the chain.

LINKED CHAINING

This type of chaining uses a list in memory consisting of memory address, transfer counts, and link addresses. Each entry in the chain list is ten bytes long, and consists of four bytes of memory address, two bytes of transfer count and four bytes of link address. The address of the first entry in the list is in the base address register, and the base transfer counter is unused. Before starting any block transfers, the DMAC fetches the entry currently pointed to by the base address register. The address information is placed in the memory address register, the count information is placed in the memory transfer counter, and the link address replaces the current contents of the base address register. The channel then begins a new block transfer. As each chaining entry is fetched, the update base address register is examined for the terminal link which has all 32 bits equal to zero. When the new base address is the terminal address, the chain is exhausted, and the entry just fetched determines the last block of the channel operation.

The memory format for this type of chaining is shown in Figure 36. This type of chaining allows entries to be easily removed or inserted without having to reorganize data within the chain. Since the end of the chain is indicated by a terminal link, the number of entries in the array need not be specified to the DMAC. All entries in the array must start at even address, or the entry fetch results in an address error. If a terminal count is loaded into the memory transfer counter, the count error is signaled. Becaused the MPU can read all of the DMAC registers, all necessary error recovery information is available to the operating system.

Table 8 Chaining Mode Address/Count Information

Chaining Mode	Base Addr. Register	Base TC	Completed When
Array Chaining	BA of Array	No. of Entries In Array	Base Transfer Counter – 0
Linked Chaining	BA of Array	–	Pointer = 0

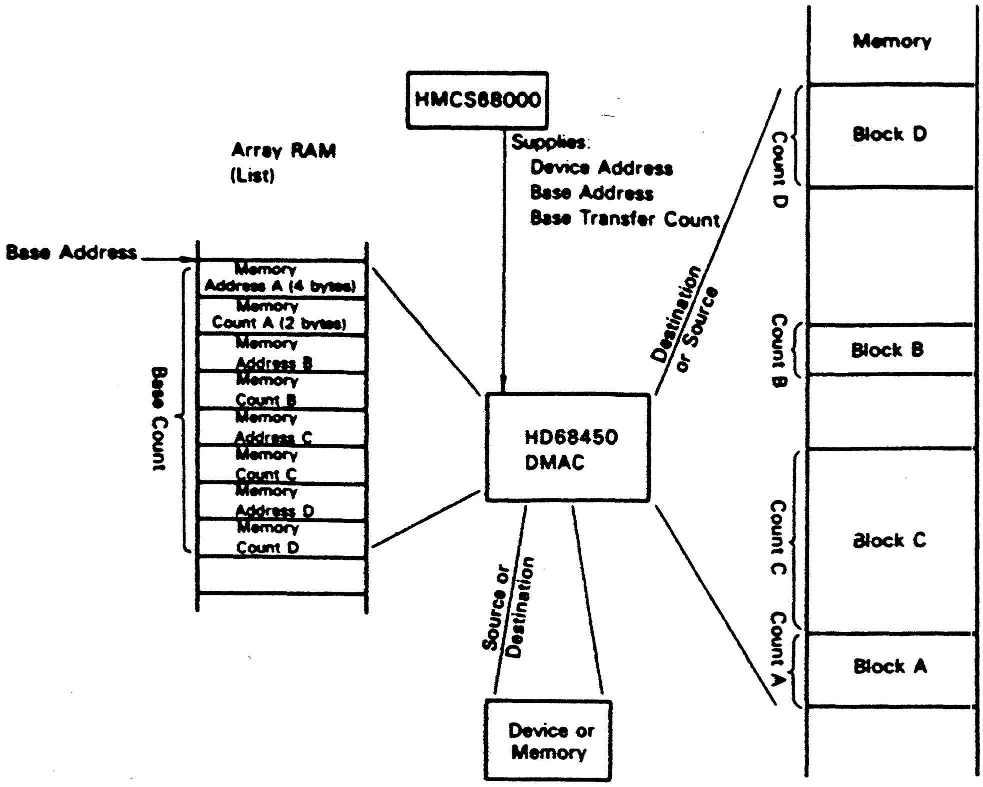


Figure 35 Array Chain Transfer

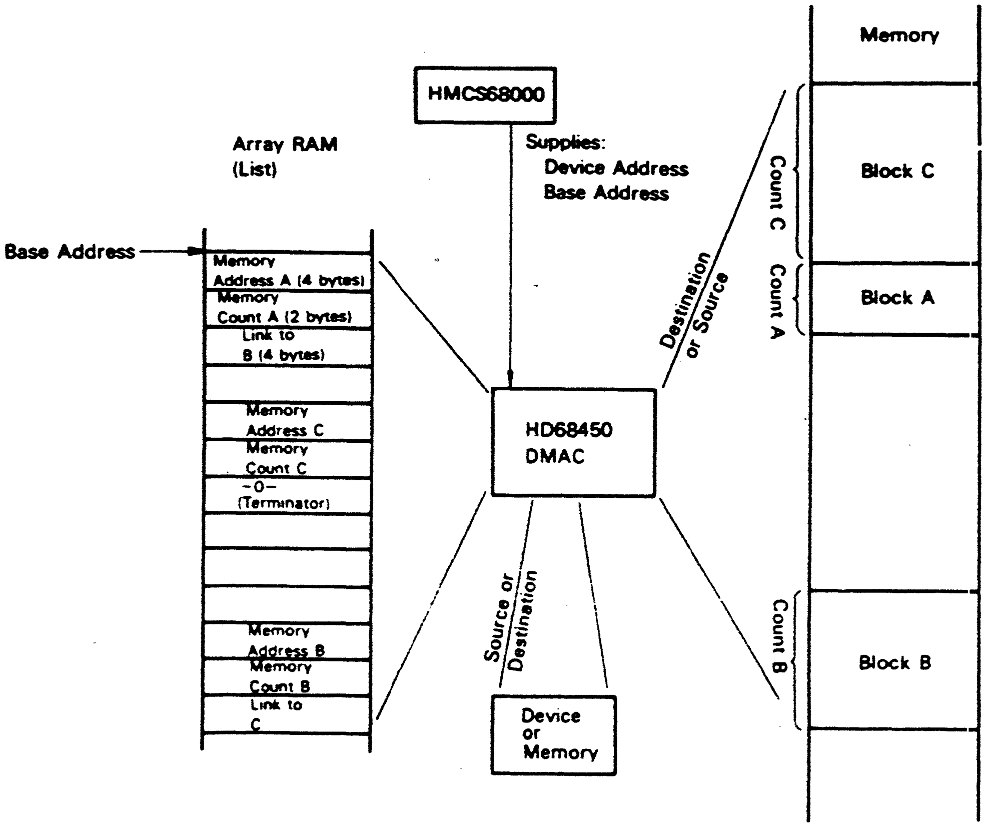


Figure 36 Linked Array Chain Transfer

(4) BUS EXCEPTION CONDITIONS

The DMAC has three lines for bus exception conditions. A priority encoder can be used to generate these signals. In order to guarantee reliable decoding, the DMAC verifies that the incoming code has been stable for two DMAC clock cycles before acting on it. The lines are encoded in the following manner (0 = active).

$\overline{\text{BEC}}$ 2	1	0	
1	1	1	– No exception condition
1	1	0	– Halt
1	0	1	– Bus error
1	0	0	– Retry
0	1	1	– Relinquish bus and retry
0	1	0	– (undefined, reserved)
0	0	1	– (undefined, reserved)
0	0	0	– Reset

These signals indicate the presence of bus exceptions. All bus exceptions except halt are preemptory. The occurrence of a preemptory bus exception during a DMAC bus cycle forces the DMAC to terminate the bus cycle in an orderly manner. The preemptory bus exception must arrive prior to or in coincidence with $\overline{\text{DTACK}}$ in order to be recognized as an abnormal bus termination. Here coincident means meeting the same set up requirements for the same sampling edge of the clock. The DMAC does not generate any bus cycles if a bus exception condition exists, and thus will not honor any requests until it is removed. However, the DMAC still recognizes requests. The reserved bus exceptions are not used by the DMAC, they should not be asserted during DMAC operations, as the result may be unpredictable.

HALT

The halt exception causes the DMAC to complete the operation in progress and three-state the bus. It does not rearbitrate for the bus until this exception is removed. When halt is negated, the DMAC resumes normal operation.

Refer to Figure 38 for more specific timing diagram.

BUS ERROR

The bus error exception is generated by external circuitry to indicate the current transfer cannot be successfully completed and is to be aborted. The recognition of this exception during a DMAC bus cycle signals the internal bus error condition for the channel for which the current bus cycle is being run.

Refer to Figure 39 for more specific timing diagram.

RETRY

The retry exception causes the DMAC to terminate the present operation and retry that operation when retry is re-

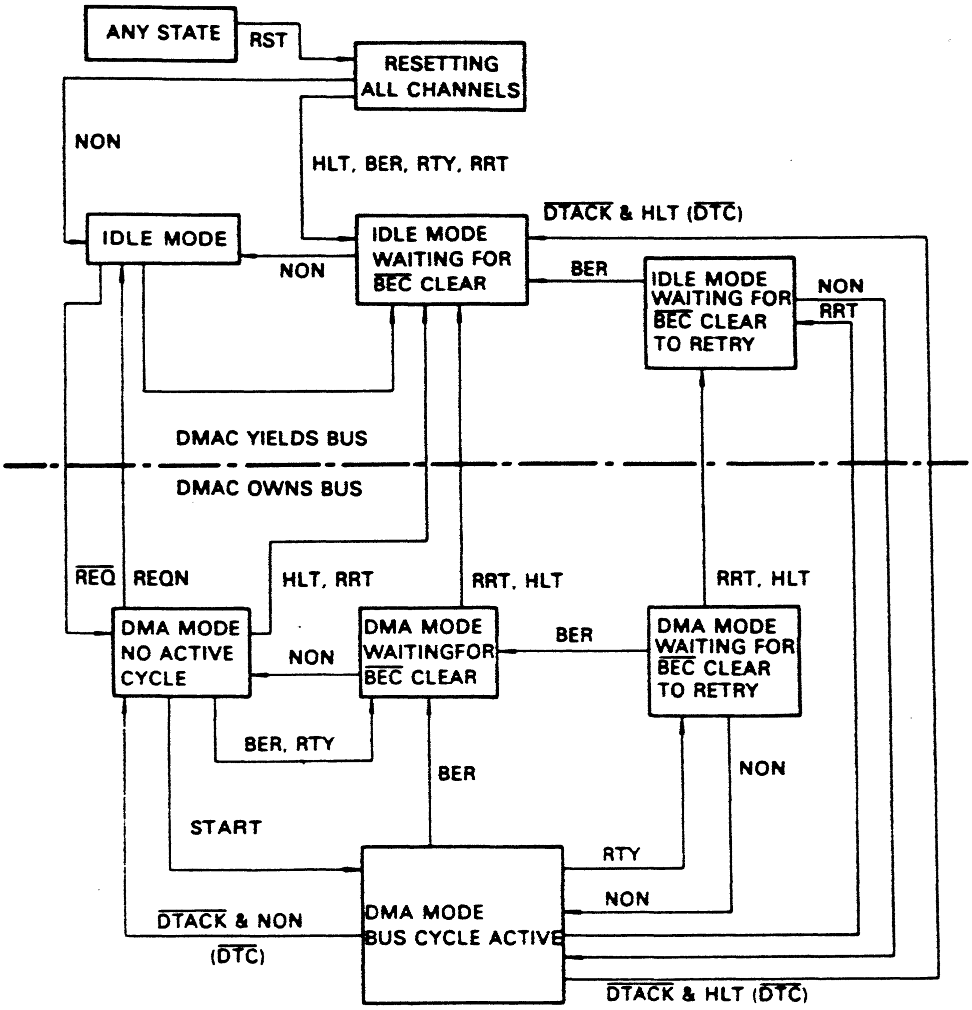


Figure 37 Bus Exception Flow Diagram

moved. The bus is not relinquished for rearbitration and the operation is reinitiated when retry is removed.

Refer to Figure 40 for more specific timing diagram.

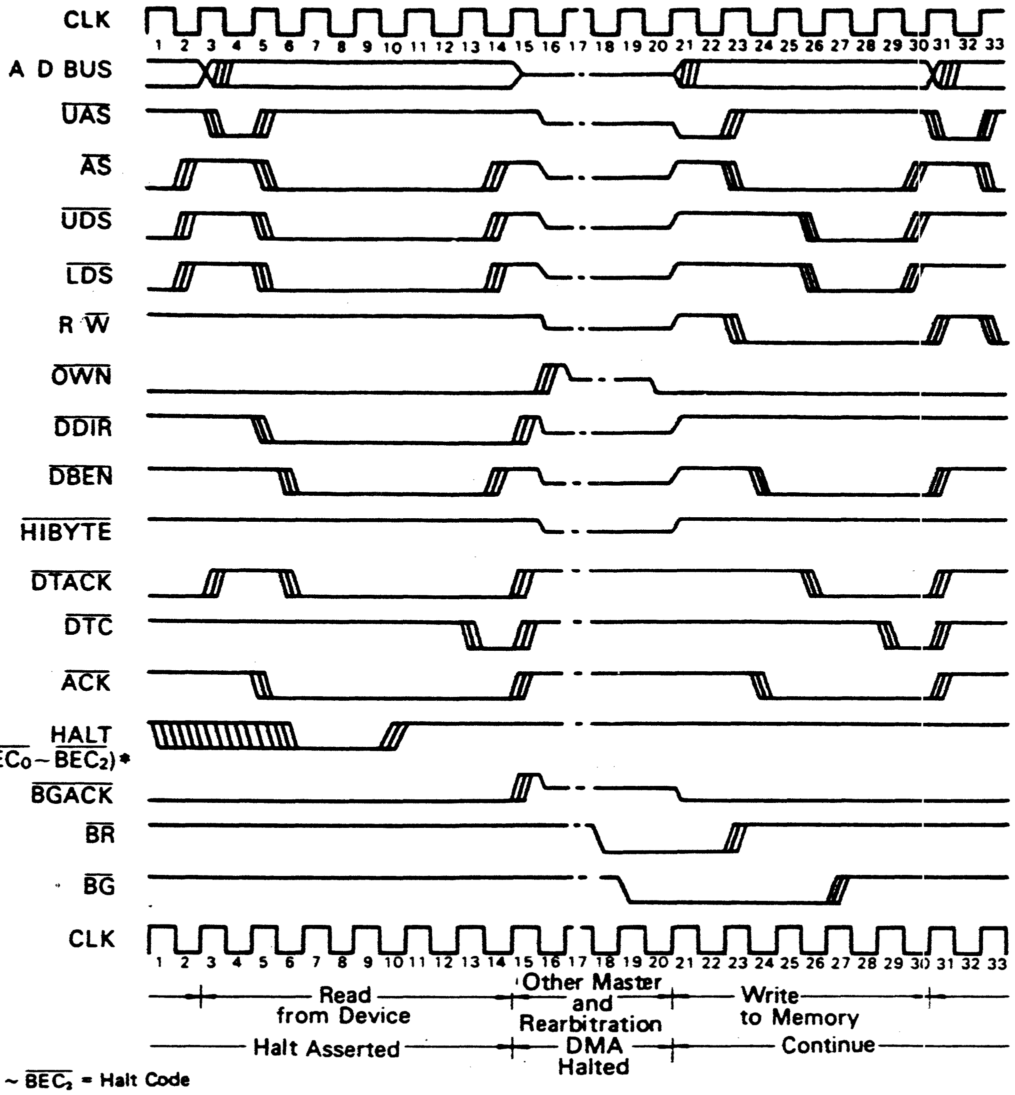
RELINQUISH AND RETRY

The relinquish and retry exception causes the DMAC to three-state all bus master controls and when the exception is removed, rearbitrate for the bus to retry the previous operation.

Refer to Figure 41 for more specific timing diagram.

RESET

The reset exception provides a means of resetting and initializing the DMAC from an external source. If the DMAC is bus master when the reset is received, the DMAC relinquishes the bus. Reset clears GCR, DCR, OCR, SCR, CCR, CSR, CPR, and CER for all channels. This resets STR, CNT, ACT and the interrupt generation bits and clears the status and error registers. The interrupt vector registers are set to $0F, the HD68000 uninitialized interrupt vector number.



* BEC0 ~ BEC2 = Halt Code

(NOTES) The following notes refer to all bus exceptions.

1) The Bus Exception will be acted upon if it is detected INTERNALLY before or at the same time as DTACK.

2) The Bus Exception pins must be stable for two clocks before the DMAC will take any action. In addition, if the BEC pins are moving but are not stable, and a DTACK is also received, the DMAC will wait for the BEC pins to resolve (remain stable for two clocks) before terminating the cycle. If the BEC pins resolve to an exception code, the DMAC will act accordingly. If they resolve to the normal mode, the DMAC will terminate if a DTACK is received and will continue normal operation.
NOTE: As long as the BEC pins are moving the DMAC will not start another cycle.

3) If possible, the DMAC should allow exceptions to be honored if they are asserted after DTACK is asserted but before the cycle has finished terminating.

4) If a cycle is not running, the Retry and Berr exceptions will be ignored except that no bus cycles are started as long as anything is detected on the BEC pins.

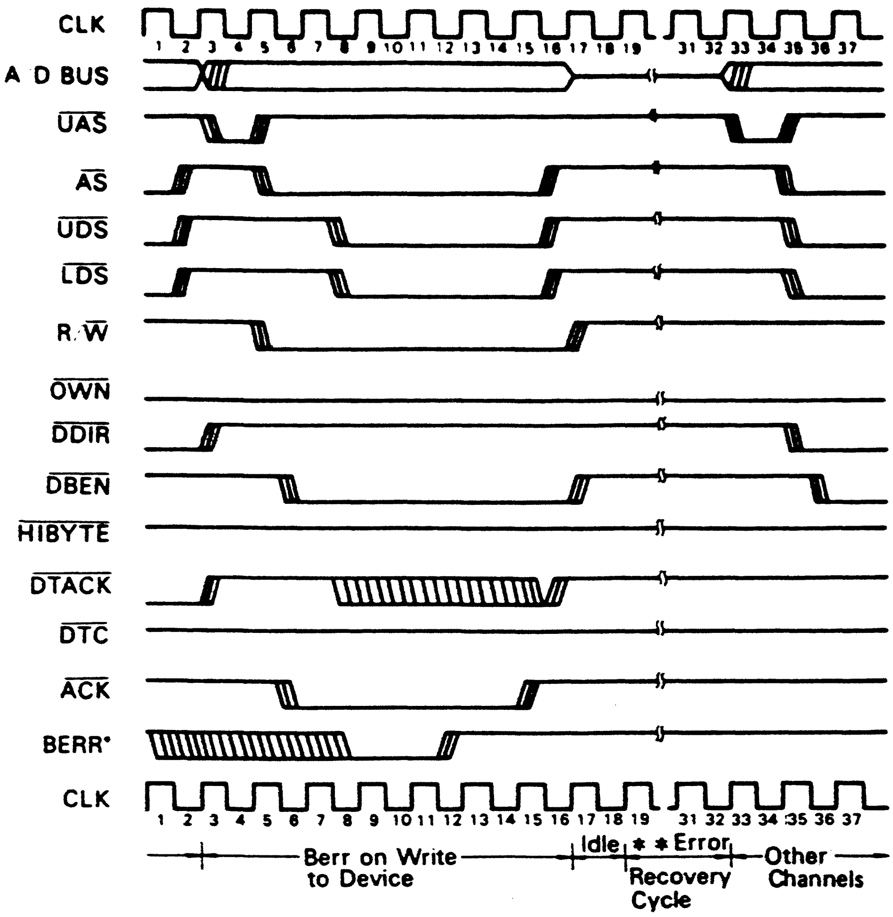
The following refer to the Halt exception only.

5) If halt is received during a cycle, it does not terminate the operation. DTACK is still required.

6) If halt is received when no bus cycle is running, the DMAC will simply give up the bus if it owns it.

7) The DMAC will not attempt to re-acquire the bus until HALT has been negated.

Figure 38 Halt Operation

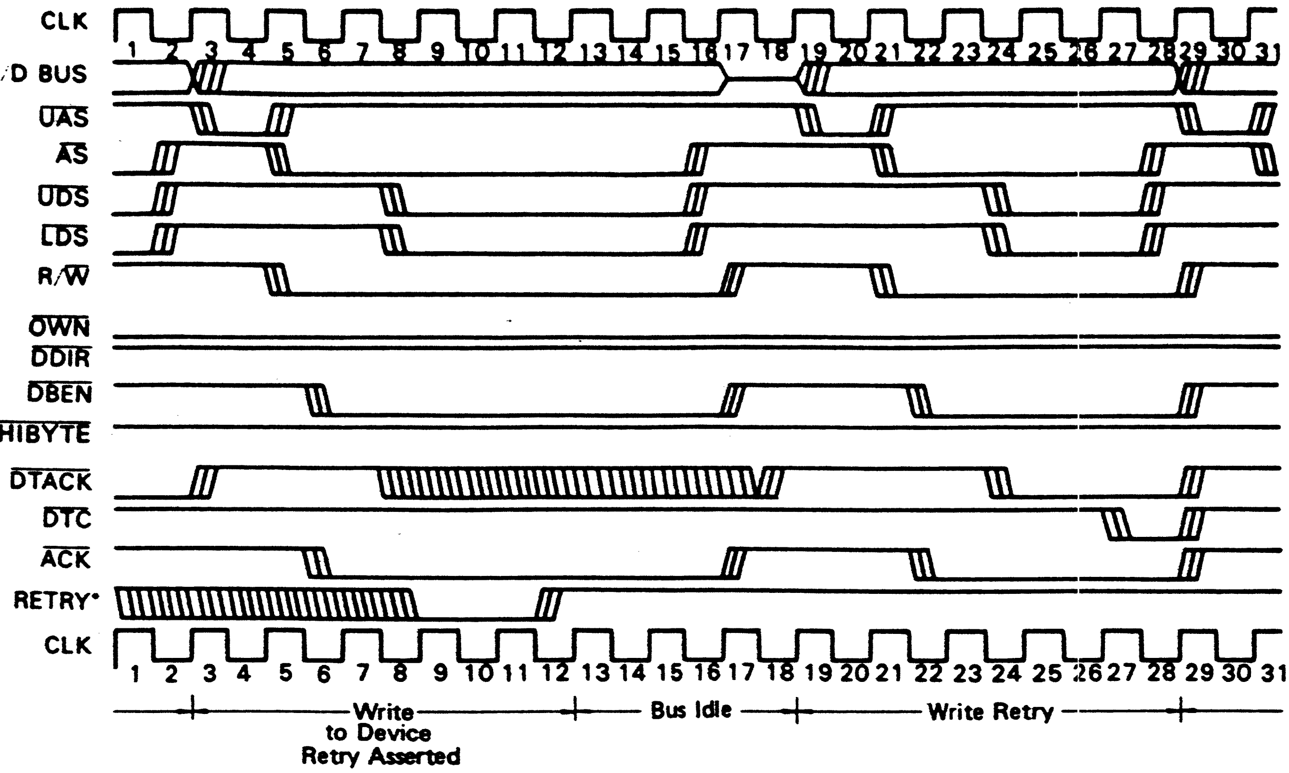


* $\overline{BEC_0}$ ~ $\overline{BEC_2}$ = Bus Error Code

** Single and Dual Cycle Source Address Error – 24 clocks
Dual Cycle Destination Address Error – 28 clocks

(NOTES)
1) In the case of preemptory bus exception, the bus cycle will always terminate immediately, but normally; i.e. it will sequence off as if a $\overline{DTACK}$ had been received. $\overline{DTC}$ will not be asserted.
2) In the case of a Berr, the DMAC will not terminate the cycle until the $\overline{BEC}$ pins have been stable in the Berr code for at least two clocks, even if a $\overline{DTACK}$ is also received.
3) See the bus exception comments below the Halt diagram.

Figure 39 Berr Operation

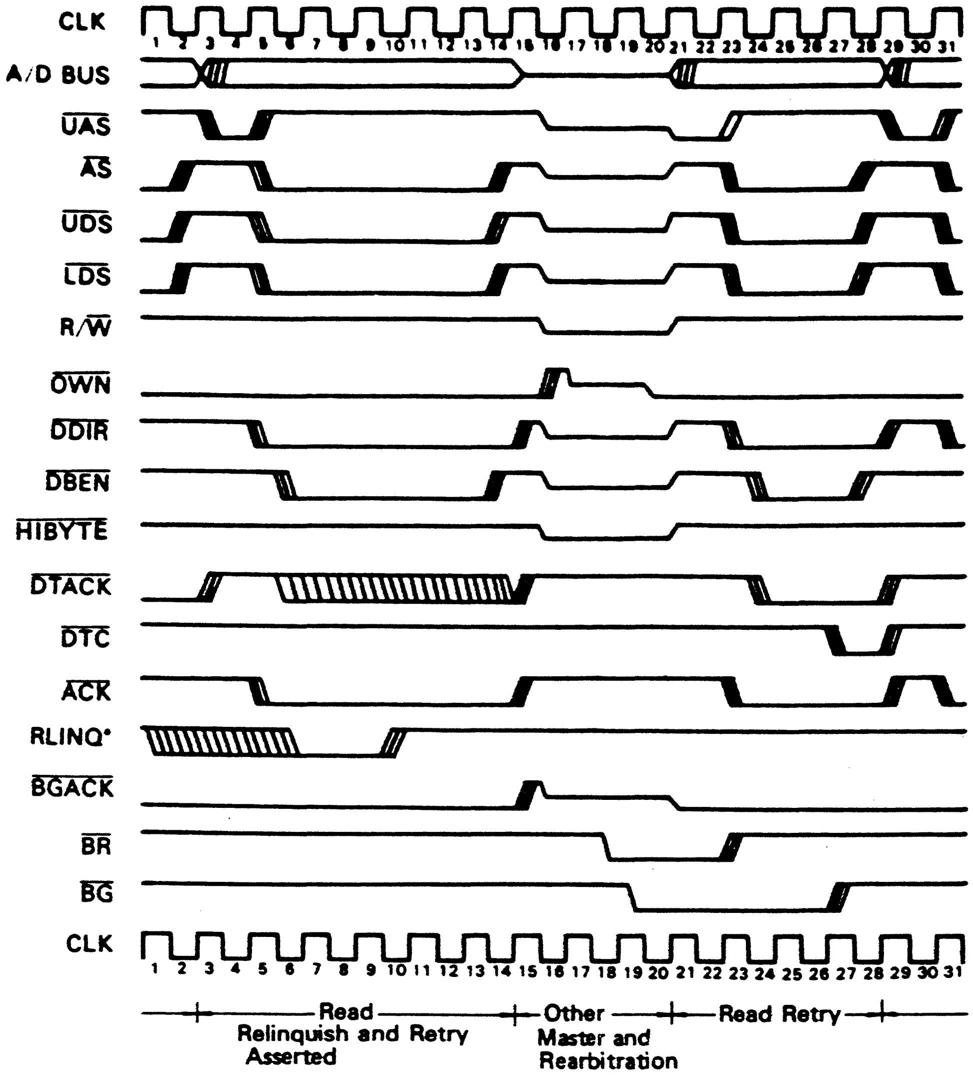


(NOTES) 1) In the case of a preemptory bus exception, the bus cycle should always terminate immediately, but normally; i.e. it should sequence off as if a $\overline{DTACK}$ had been received.

2) In the case of a Retry, the DMAC should not terminate the cycle until the $\overline{BEC}$ pins have been stable in the Retry code for at least one clock, even if a $\overline{DTACK}$ is also received.

3) See the bus exception comments below the Halt diagram.

Figure 40 Retry Operation



• $\overline{BEC_0}$~$\overline{BEC_2}$ = Relinquish and Retry Code

(NOTES)
1) In the case of a preemptory bus exception, the bus cycle should always terminate immediately, but normally; i.e. it should sequence off as if a $\overline{DTACK}$ had been received.
2) In the case of a Relinquish and Retry, the DMAC should not terminate the cycle until the $\overline{BEC}$ pins have been stable in the Relinquish and Retry code for at least two clocks, even if a $\overline{DTACK}$ is also received.
3) See the bus exception comments below the Halt diagram.

Figure 41 Relinquish and Retry Operation

$\overline{\text{BEC}}$ CONTROLS

If the $\overline{\text{BEC}}$ controls are asserted to a state that is undefined/reserved, this version of the DMAC will enter a wait state and resume operation when the exception is removed.

(3) ERROR CONDITIONS

When an error is signaled on a channel, all activity on that channel is stopped. The ACT bit of the CSR is cleared, and the COC bit is set. The ERR bit of the CSR is set, and the error code indicated in the CER All pending operations are cleared, so that both the STR and CNT bits of CCR are cleared.

SOURCES OF ERRORS

Enumerated below are the error signals and their sources.

Configuration Error

A configuration error is signaled if chaining is programmed and the continue bit is also set. Configuration error is signaled if DTYP specifies a single-address transfer, and the device port size is not the same as the operand size. Configuration error is signaled if DTYP is 68000 or 6800, DPS is 16 bits, SIZE is 8 bits, and REQG is 10 or 11 (request pin). Setting an undefined configuration will signal a configuration error. The undefined configurations are: XPM = 01, MAC = 11, DAC = 11, CHAIN = 01, SIZE = 11.

Operation Timing Error

An operation timing error is signaled if an attempt is made to continue an operation without STR being simultaneously set or if the channel is not active. Operation timing error is signaled if an attempt to set STR is made with ACT, COC, BTC, NPT, or ERR asserted. Operation timing error is signaled if an attempt to write to the DCR, OCR, SCR, GCR, MAR, DAR or MTC is made with STR or ACT asserted. Operation timing error is signaled if an attempt to assert CNT is made when CHAIN is 10 or 11 (chaining modes). Operation timing error is signaled if an attempt to assert CNT is made when BTC and ACT are asserted.

Address Error

Address error is signaled if an odd address operation is attempted with word or long word operands or if $\overline{\text{CS}}$ or $\overline{\text{IACK}}$ is asserted while the DMAC is bus master. The address error is asserted after the odd address is encountered, this is consistent with the processor operation.

Bus Error

A Bus error occurred during the last bus cycle generated by the channel.

Count Error

A count error is signaled if the memory or base transfer count registers are initialized with terminal count. A count error is signaled if a terminal count is encountered during continue or chain processing.

Abort

An abort error is signaled if the $\overline{\text{PCL}}$ line was configured as an abort input and made an active transition, or if the channel operation was aborted by the SAB bit of the CCR.

Note: When the $\overline{\text{PCL}}$ line is used as an abort input, the PCT bit should be cleared prior to starting the channel. If the PCT bit is set prior to the channel being started, the DMAC will recognize this as an external abort when the channel is started.

When the transfer mode is set to dual addressing mode, the transfer direction is set to I/O device to memory, and $\overline{\text{PCL}}$ signal is set to external about input mode, the external abort for that channel will be ignored after a $\overline{\text{DONE}}$ input from the I/O device is received during the channel's I/O device-to-memory data transfer cycle. After the $\overline{\text{DONE}}$ input, the channel will accept external abort when the channel is properly reinitialized and is restarted.

ERROR RECOVERY PROCEDURES

If an error occurs during a DMA transfer, appropriate information is available to the operating system to allow a "soft failure" operation. The operating system must be able to determine how much data was transferred, where the data was transferred to, and what type of error occurred.

The information available to the operating system consists of the present values of the Memory Address, Device Address and Base Address Registers, the Memory Transfer and Base Transfer Counters, and control, status, and error registers. After the successful completion of any transfer, the memory and device address registers points to the location of the next operand to be transferred and the memory transfer counter contains the number of operands yet to be transferred. If an error occurs during a transfer, that transfer has not completed and the registers contain the values they had before the transfer was attempted. If the channel operation uses chaining, the Base Address Register points to the next chain entry to be serviced, unless the termination occurred while attempting to fetch an entry in the chain. In that case, the Base Address Register points to the entry being fetched.

MULTIPLE ERRORS

The DMAC will log the first error encountered in the channel error resister. If an error is pending in the error register and another error is encountered the second error will not be logged. This is true in the case of an error pending and an attempt is made to set the STR bit. In this case the Operation timing error is not logged in the error register but the error is recognized internally and the channel is not started.

● CHANNEL PRIORITIES

Each channel has a priority level, determined by the contents of the Channel Priority Register (CPR). The priority of a channel is a number from 0 to 3, with 0 being the highest priority level. When multiple requests are pending at the DMAC, the channel with the highest priority receives first service. The priority of a channel is independent of the device protocol or the request mechanism for that channel. If there are several requesting channels at the highest priority level, a round-robin resolution is used, that is, as long as these channels continue to have requests, the DMAC does operand transfers in rotation.

■ APPLICATIONS INFORMATION

This section contains examples of how to interface various I/O devices to a HMCS68000/DMAC based system.

Figure 42 shows an example of how to demultiplex the address/data bus.

Figure 43 indicates the example of how to latch the data, when the DMAC has two channels which operate in 6800 mode.

Figure 44 indicates the example of inter-device connection in the HMCS68000 system.

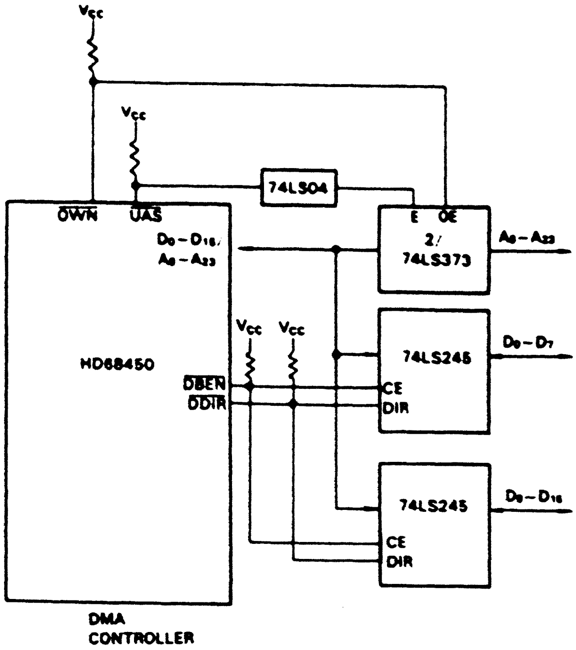


Figure 42 Required Multiplexed Data/Address Hardware for the Bus Control Logic

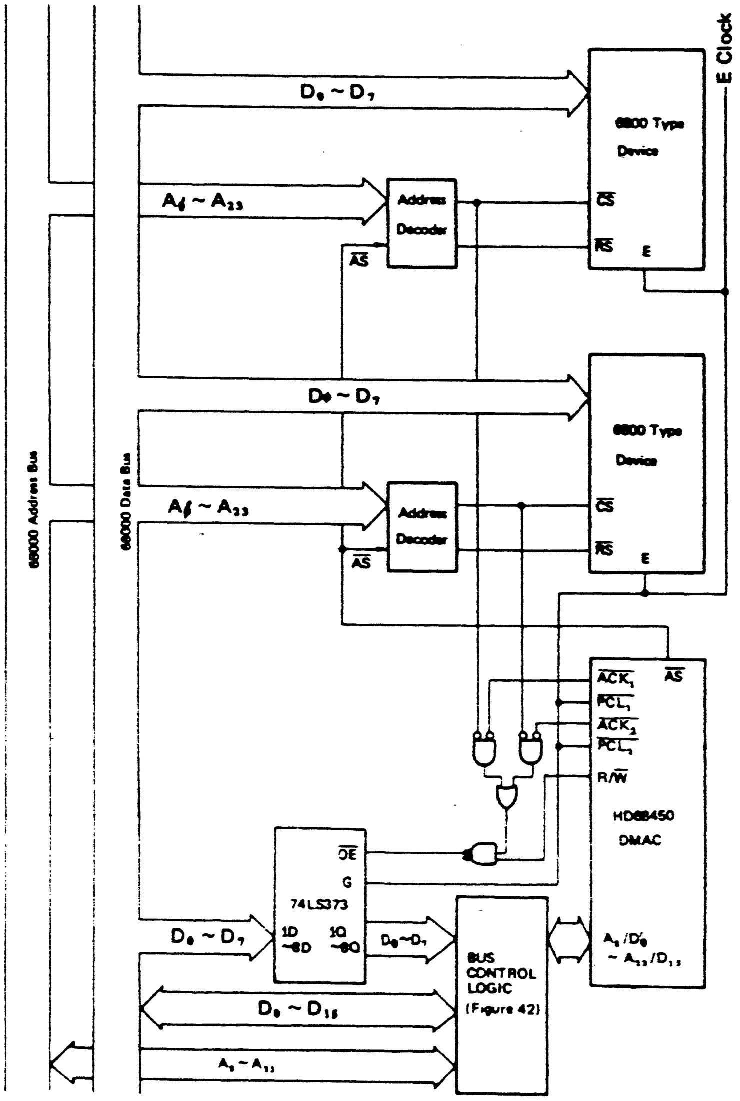


Figure 43 An Example of Connection with Peripheral Devices in 6800 Mode

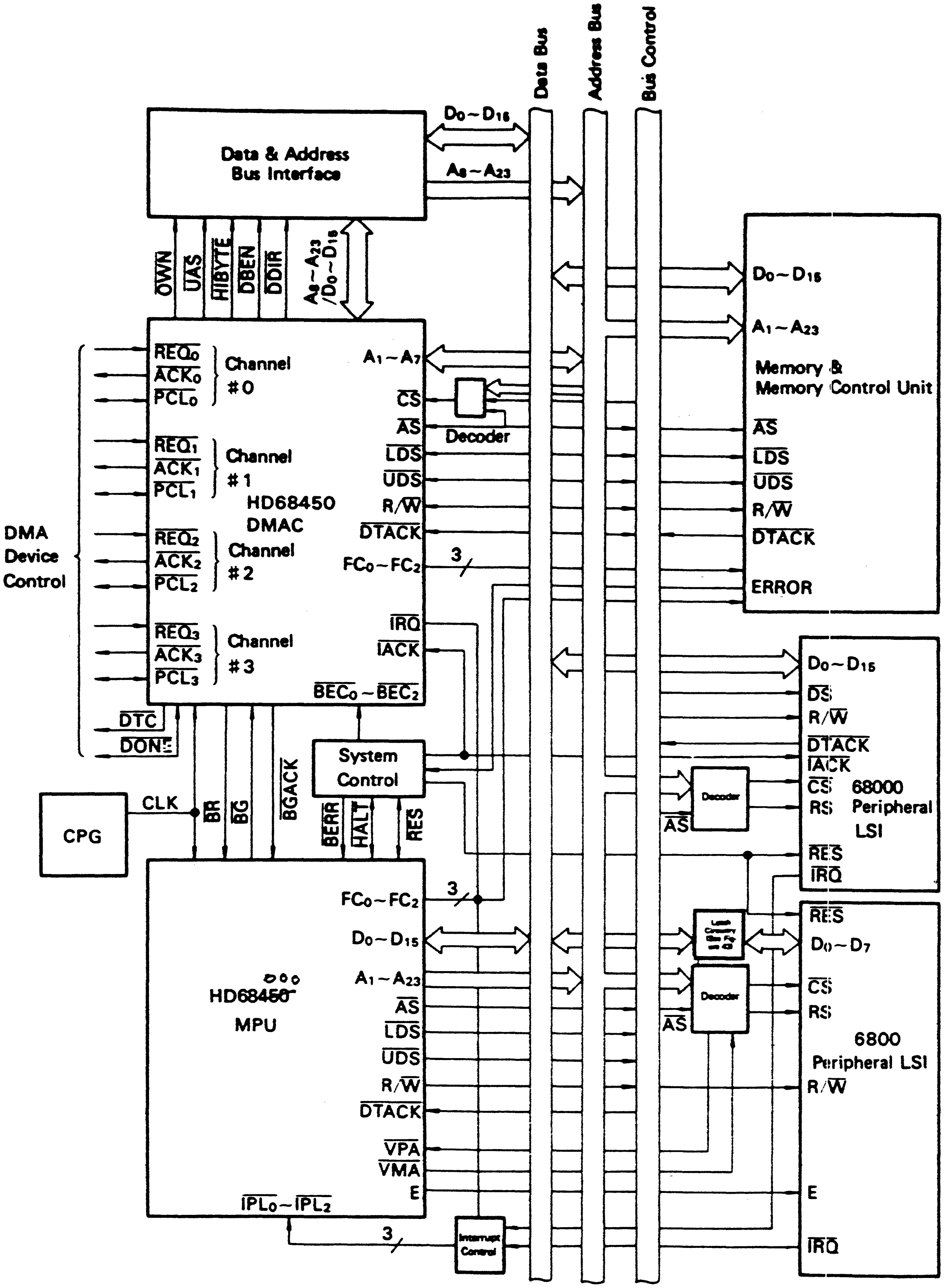


Figure 44 An Example of Inter-device Connection in the HMCS68000 System

Hitachi Microcomputer Device Technical Information
September, 1982

Revision of HD68450 Data Sheet

Content

Page

1 Replace the mnemonic for pin 21, "$\overline{\mathrm{IREQ}}$" with "$\overline{\mathrm{IRQ}}$".

2 Revise the value of "Power Dissipation".

Before:	Revised:
1.0W typ., 1.75W max.	1.4W typ., 2.0W max. at Ta = 25 degrees C and V_{CC} = 5.0V

3 Revise the following items of "AC Electrical Specification". No. 94 and 95 are newly added.

No.	Item	Symbol	Revised Value (ns)					
			4MHz		6MHz		8MHz	
			min	max	min	max	min	max
16	$\overline{\mathrm{DS}}$ In "High" to $\overline{\mathrm{DDIR}}$ "High" Impedance	t DSHDRZ		160		140		120
17	$\overline{\mathrm{DS}}$ In "High" to $\overline{\mathrm{DBEN}}$ "High" Impedance	t DSHDBZ		160		140		120
18	Clock "High" to Data Out Valid (MPU Read)	t CHDVM		240		170		130
21	$\overline{\mathrm{DS}}$ In "High" to $\overline{\mathrm{DTACK}}$ "High"	t DSHDTH		140		130		110
34	Clock "High" to Address/FC Valid	t CHAV		160		140		120
35	Clock "High" to Address/FC/ Data "High" Impedance	t CHAZx		140		120		100
94	Address In to $\overline{\mathrm{AS}}$ In "Low"	t AIASL	0		0		0	
95	$\overline{\mathrm{AS}}$, $\overline{\mathrm{DS}}$ In "High" to Address Invalid	t SIHAIV	0		0		0	

5 Ignore indices "91" and "92" in "Figure 3, AC Electrical Waveforms-MPU Read/Write".

5 Replace index "25" with "23" in "Figure 3, AC Electrical Waveforms-MPU Read/Write".

14 Delete the following sentence from the paragraph titled "$\overline{\mathrm{PCL}}$ AS AN ABORT INPUT":

When this function has been programmed, the $\overline{\mathrm{PCL}}$ line is only active after the channel has been started.

35 Add the following comment to "Figure 37, Bus Exception Flow Diagram":

When the DMAC is in "IDLE MODE WAITING FOR $\overline{\mathrm{BEC}}$ CLEAR", (BER or RTY is asserted in this state) the MPU cannot access the DMAC registers. BER or RTY must be negated before the MPU accesses the DMAC. The MPU can access the DMAC registers while HLT or RRT is asserted.

40 Add the following sentence after the 6th line from the top of the right column (ending with"... and is restarted."):

In order to detect an external abort of the above kind, the user is advised to examine the PCT bit of the channel status register (CSR) along with the ERR bit. If the PCT bit is set and ERR bit is reset, then he can conclude that an external abort has occured in the DONE cycle and take an appropreate action.

HITACHI reserves the right to make changes to any products herein to improve functioning or design. Although the information in this document has been carefully reviewed and is believed to be reliable, HITACHI does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights or others.

40 Replace the content of the section titled "MULTIPLE ERRORS" with the following sentence:
The DMAC detects and services multiple errors, however the content of CER (channel error register) retains the first error that the channel has encountered regardless of what type of errors occur after the first one.

41 The following change has to be made on "Figure 43 An Example of Connection with Peripheral Devices in 6800 Mode":
The AND gate input to $\overline{OE}$ of 74LS373 should be changed to a NAND gate.
Change "$A_0 \sim A_{23}$" to "$A_1 \sim A_{23}$"
Change "$D_8 \sim D_7$" to "$D_0 \sim D_7$"

42 Replace HD 68450 with HD 68000
MPU MPU
(Lower left block of page)

HITACHI

HITACHI-EUROPE
MÜNCHEN, HEADQUARTER
Hitachi Electronic Components
Europe GmbH
Hans-Pinsel-Straße 3
8013 Haar b. München
☎ (089) 4614-0
Telex: 5-22593
Telefax: (089) 463151

HITACHI-EUROPE
DÜSSELDORF
Hitachi Electronic Components
Europe GmbH
Königsallee 6
4000 Düsseldorf 1
☎ (0211) 84995
Telex: 8-584536
Telefax: (0211) 324612

HITACHI-EUROPE
STUTTGART
Hitachi Electronic Components
Europe GmbH
Fabrikstraße 17
7024 Filderstadt
☎ (0711) 772011
Telex: 7-255267

HITACHI-EUROPE
PARIS
Hitachi Electronic Components
Europe GmbH
Bureau de Représentation en France
95-101, Rue Charles-Michels
F-93200 Saint Denis
☎ 01-8216015
Télex: 611387
Téléfax: 01-2436997

HITACHI-EUROPE
MILANO
Hitachi Electronic Compo
Europe GmbH
Via B. Davanzati, 27
I-20158 Milano
☎ 02-3763024
Telex: 320343
Telefax: 02-683730

HITACHI-UNITED KINGDOM
LONDON, HEADQUARTER
Hitachi Electronic Components
(UK) Ltd.
Hitec House,
221-225 Station-Road
Harrow, Middlesex, HA1 2XL
☎ 01-8611414
Telex: 936293
Telefax: 01-8636646

HITACHI-UNITED KINGDOM
STOCKHOLM
Hitachi Electronic Components
(UK) Ltd.
Box 1602, 16311 Spanga
Hankadalsgatan 10 Kista
☎ 0046-87510035
Telex: 14106 (Hitecst S)
Telefax: 08-7515073

A-12-82-06.UP450 Printed in West-Germany

APPENDIX L

DATA SHEET 5385/6 SCSI CONTROLLER

NCR 5385E SCSI

Protocol Controller

Data Sheet

NCR

Microelectronics Division, Colorado Springs

This document contains the latest information available at the time of publication. However, NCR reserves the right to modify the contents of this material at any time. Also, all features. functions and operations described herein may not be marketed by NCR in all parts of the world. Therefore, before using this document, consult your NCR representative or NCR office for the information that is applicable and current.

TABLE OF CONTENTS

SECTION PAGE

SECTION 1
GENERAL DESCRIPTION

The NCR SCSI Protocol Controller (SPC) is designed to accomodate the Small Computer Systems Interface (SCSI) as defined by the ANSI X3T9.2 committee. The SPC operates in both the Initiator and Target roles and can therefore be used in host adapter and control unit designs. This device supports arbitration, including reselection, and is intended to be used in systems that require either open collector or differential pair transceivers.

The NCR 5385E SCSI Protocol Controller communicates with the system microprocessor as a peripheral device. The chip is controlled by reading and writing several internal registers which may be addressed as standard or memory mapped I/O. A 24-bit Transfer Counter and the appropriate handshake signals accommodate large DMA transfers with minimal processor intervention. Since the NCR 5385E interrupts the MPU when it detects a bus condition that requires servicing, the MPU is freed from polling or controlling any of the SCSI bus signals.

Below is a list of important features:

SCSI INTERFACE

*Supports ANSI X3T9.2 SCSI Standard

- Asynchronous data transfers to 1.5 MBPS
- Supports both Initiator and Target roles
- Parity generation with optional checking
- Supports arbitration
- Controls all bus signals except Reset
- Doubly-buffered Data Register

MPU INTERFACE

*Versatile MPU Bus Interface

- Memory or I/O mapped MPU interface
- DMA or programmed I/O transfers
- 24-bit Internal Transfer Counter
- Programmable (Re)Selection timeouts
- Interrupts MPU on all bus conditions requiring service

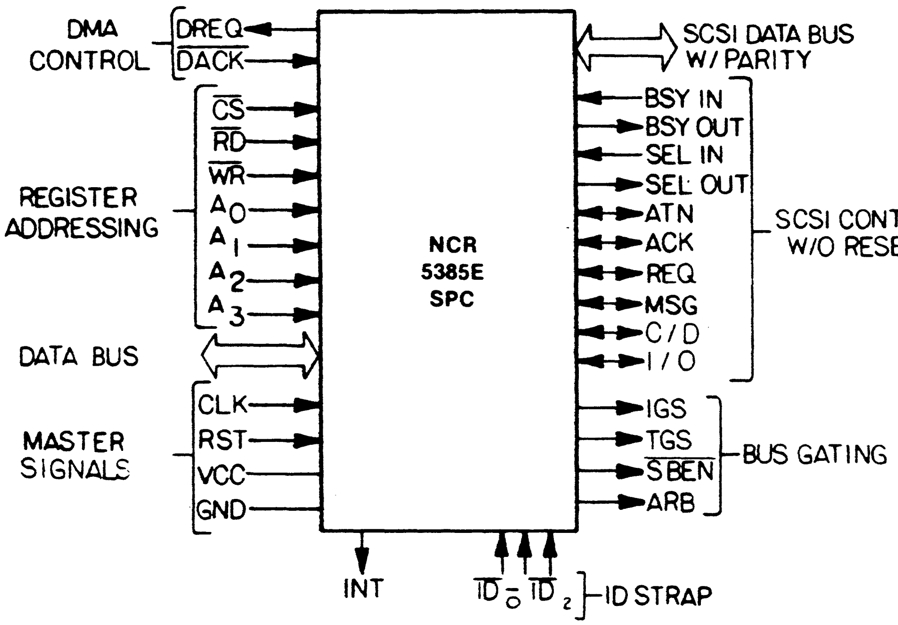


FIG. 1.1 FUNCTIONAL PIN GROUPING

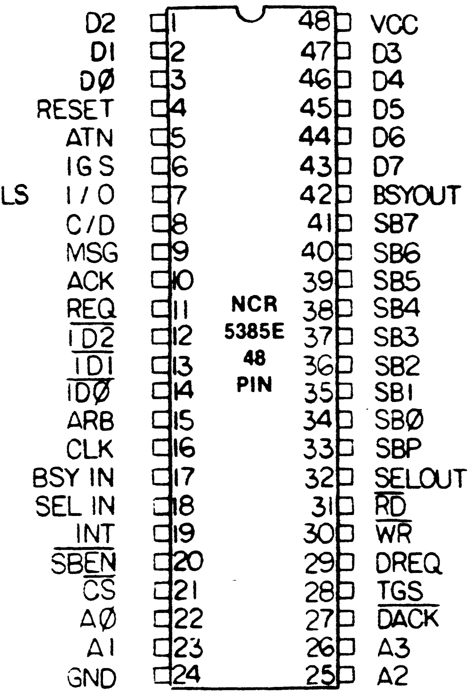


FIG. 1.2 PINOUT

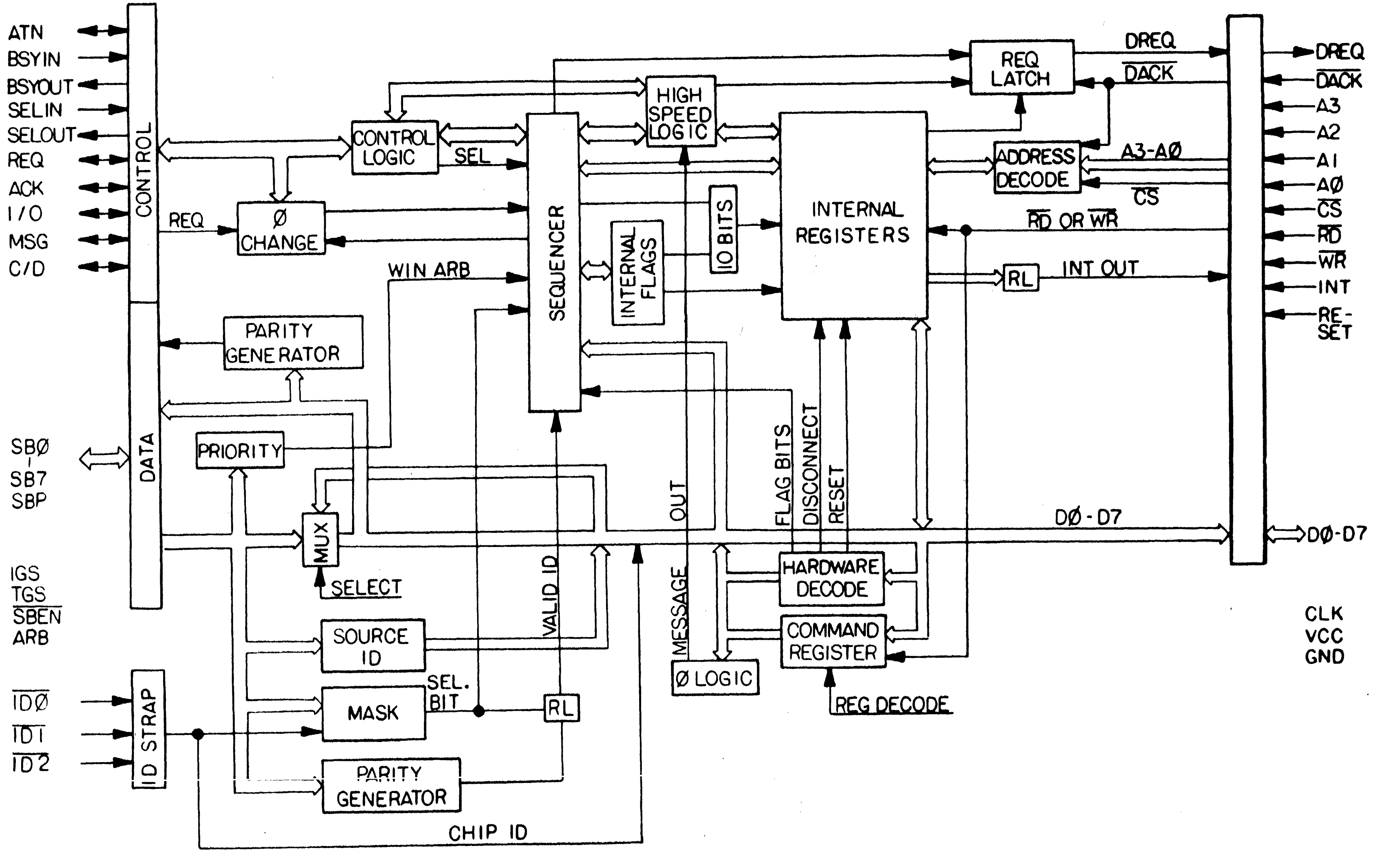


FIG. 1.3 NCR 5385E BLOCK DIAGRAM

SECTION 2
PIN DESCRIPTION

2.1 MICROPROCESSOR INTERFACE SIGNALS

CLK	16	Symmetrical square wave signal which generates internal chip timing. Maximum frequency is 10 MHz.
RESET	4	When high (1), this signal forces the chip into a reset state. All current operations are terminated. Internal storage elements are cleared and self-diagnostics are performed.
D0-D7	3-1 47-43	These signals comprise an active high data bus. It is intended that these signals be connected to the microprocessor data bus.
INT	19	This signal is used to interrupt the microprocessor for various bus conditions that require service. INT is set high for request and cleared when the chip is reset or the Interrupt Register is read.
$\overline{WR}$	30	Write pulse (active low) is used to strobe data from the data bus into an internal register which has been selected.
$\overline{RD}$	31	Read pulse (active low) is used to read data from an internal register that has been selected. The contents of the register is strobed onto the data bus.
$\overline{CS}$	21	When low (0), this signal enables reading from or writing to the internal register which has been selected.
A0-A3	22, 23, 25, 26	These signals are used in conjunction with $\overline{CS}$, to address all the internal registers.
DREQ	29	Data request. When high (1), this signal indicates that the internal Data Register has a byte to transfer (inputting from the SCSI bus) or needs a byte to transfer (outputting to the SCSI bus). This signal becomes active only if the DMA mode bit in the Command Register is on. It is cleared when $\overline{DACK}$ becomes active.
$\overline{DACK}$	27	Data acknowledge. When low (0), this signal resets DREQ and selects the Data Register for input or output. $\overline{DACK}$ acts as a chip select for the Data Register when in the DMA mode. $\overline{DACK}$ and $\overline{CS}$ must never be active at the same time.

2.2 SCSI INTERFACE SIGNALS

$\overline{\text{ID0}}$-$\overline{\text{ID2}}$	14-12	These active low signals determine the three-bit code of the SCSI bus ID assigned to the chip. External pullup resistors are required only if tied to switches or straps.
SB0-SB7, SBP	34-41, 33	Active high data bus. These signals comprise the SCSI data bus and are intended to be connected to the external SCSI bus transceivers.
BSYIN	17	When high (1), this signal indicates to the chip that the SCSI BSY signal is active.
BSYOUT	42	When high (1), the chip is asserting the BSY signal to the SCSI bus.
SELIN	18	When high (1), this signal indicates to the chip that the SCSI SEL signal is active.
SELOUT	32	When high (1), the chip is asserting the SEL signal to the SCSI bus.
ATN	5	INITIATOR ROLE: The chip asserts this signal when the microprocessor requests the attention condition or a parity error has been detected in a byte received from the SCSI bus. TARGET ROLE: This signal is an input which indicates the state of the ATN signal on the SCSI bus.
ACK	10	INITIATOR ROLE: The chip asserts this signal in response to REQ for a byte transfer on the SCSI bus. TARGET ROLE: This signal is an input which, when active, indicates a response to the REQ signal.
REQ	11	INITIATOR ROLE: This signal is an input which, when active, indicates that the Target is requesting a byte transfer on the SCSI bus. TARGET ROLE: Asserted by the chip to request a byte transfer on the SCSI bus.
MSG, C/D, I/O	9, 8, 7	INITIATOR ROLE: These signals are inputs which indicate the current SCSI bus phase. TARGET ROLE: The chip drives these signals to indicate the current bus phase.
IGS	6	Initiator Group Select. When high (1), this signal indicates to the external SCSI drivers that the chip is controlling in the Initiator role. Its purpose is to enable the external drivers for ATN and ACK.
TGS	28	Target Group Select. When high (1), this signal indicates to the external SCSI drivers that the chip is controlling in the Target role. Its purpose is to enable the external drivers for REQ, MSG, C/D, and I/O.

$\overline{\text{SBEN}}$	20	SCSI data Bus Enable. When low (0), this signal directly enables the external SCSI data bus drivers.
ARB	15	Arbitration phase. When high (1), this signal enables the external circuitry to place the ID bit on the SCSI bus for the Arbitration phase.

POWER SIGNALS

VCC	48	+5 V input
GND	24	Signal reference input

SECTION 3
ELECTRICAL CHARACTERISTICS

OPERATING CONDITIONS

PARAMETER	SYMBOL	MIN	MAX	UNITS
Supply Voltage	V_{DD}	4.75	5.25	VDC
Supply Current	I_{DD}		300	mA
Ambient Temp.	T_A	0	70	°C

INPUT SIGNAL REQUIREMENTS

PARAMETER	CONDITIONS	MIN	MAX	UNITS
High-level Input, V_{IH}		2.0	5.25	VDC
Low-level Input, V_{IL}		-0.3	0.8	VDC
High-level Input Current, I_{IH}	$V_{IH} = 5.25V$		10	μA
Low-level Input Current, I_{IL}	$V_{IL} = 0V$		-10	μA

OUTPUT SIGNAL REQUIREMENTS (Except $\overline{\text{SBEN}}$, IGS, and TGS)

PARAMETER	CONDITIONS	MIN	MAX	
High-level Output Voltage, V_{OH}	$V_{DD} = 4.75V$ @ $I_{OH} = -400\mu A$	2.4	—	VDC
Low-level Output Voltage, V_{OL}	$V_{DD} = 4.75V$ @ $I_{OL} = 2.0mA$	—	0.4	VDC

$\overline{\text{SBEN}}$, IGS, and TGS SIGNALS

PARAMETER	CONDITIONS	MIN	MAX	UNITS
High-level Output Voltage, V_{OH}	$V_{DD} = 4.75V$ @ $I_{OH} = -400\mu A$	2.4	—	VDC
Low-level Output Voltage, V_{OL}	$V_{DD} = 4.75V$ @ $I_{OL} = 4.0mA$	—	0.4	VDC

PRELIMINARY

Notice: This is not a final specification.
Some parametric limits are subject to change.

SECTION 4
INTERNAL REGISTERS

4.0 GENERAL

The NCR SCSI Protocol Controller has a set of internal registers which are used by the microprocessor to direct the operation of the SCSI bus. These registers are read (written) by activating $\overline{\text{CS}}$ with an address on A3-A0 and then issuing a $\overline{\text{RD}}$/($\overline{\text{WR}}$) pulse. They can be made to appear to a microprocessor as standard I/O ports or as memory-mapped I/O ports depending on the external circuitry that controls $\overline{\text{CS}}$. The following sections describe the operation of these internal registers.

REGISTER SUMMARY

A3	A2	A1	A0	R/W	REGISTER NAME
0	0	0	0	R/W	Data Register
0	0	0	1	R/W	Command Register
o	0	1	0	R/W	Control Register
0	0	1	1	R/W	Destination ID
0	1	0	0	R	Auxiliary Status
0	1	0	1	R	ID Register
0	1	1	0	R	Interrupt Register
0	1	1	1	R	Source ID
1	0	0	1	R	Diagnostic Status
1	1	0	0	R/W	Transfer Counter (MSB)
1	1	0	1	R/W	Transfer Counter (2nd BYTE)
1	1	1	0	R/W	Transfer Counter (LSB)
1	1	1	1	R/W	Reserved for Testability

4.1 DATA REGISTER

The Data Register is used to transfer SCSI commands, data, status and message bytes between the microprocessor data bus and the SCSI bus. This is an eight-bit register which is doubly-buffered in order to support maximum throughput. In the non-DMA mode, the microprocessor reads from (writes to) the Data Register by activating $\overline{\text{CS}}$ with A3-A0 = 0000 and issuing a $\overline{\text{RD}}$/($\overline{\text{WR}}$) pulse. A bit has been included in the Auxiliary Status Register to indicate when the Data Register is full. In the DMA mode, the DMA logic reads from (writes to) the Data Register by responding to DREQ with $\overline{\text{DACK}}$ and issuing a $\overline{\text{RD}}$/($\overline{\text{WR}}$) pulse. The SCSI bus reads from or writes to the Data Register when the chip is connected as an Initiator or Target and the bus is in one of the Information Transfer Phases.

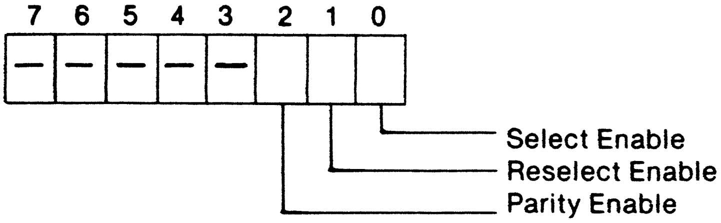
4.2 COMMAND REGISTER

The Command Register is an eight-bit register used to give commands to the SCSI chip. The microprocessor can write to (read from) the Command Register by activating $\overline{\text{CS}}$ with A3-A0 = 0001 and issuing a $\overline{\text{WR}}$/($\overline{\text{RD}}$) pulse. Writing to the Command Register causes the chip to execute the command that is written. The Command Register can be read; however, the chip resets the Command Register when it sets an Interrupt. Therefore, one cannot guarantee that the data in the register will be correct after loading an interrupting command or enabling selection or reselection. To be safe, a copy of the last command issued should be stored in the microprocessor's memory. Immediate commands are not stored.

The contents of the Command Register are described in a later section (See page 19, COMMANDS).

4.3 CONTROL REGISTER

This eight-bit read/write register is used for enabling certain modes of operation for the SCSI Protocol Controller. The microprocessor reads from (writes to) the Control Register by activating $\overline{CS}$ with A3-A0 =0010 and issuing a $\overline{RD}$ ($\overline{WR}$) pulse.



BIT 7-3 Reserved

BIT 2 Parity Enable — When the parity enable bit is a "1", the chip generates and checks parity on all transfers on the SCSI bus. When the parity enable bit is a "0", the chip generates but does not check parity on bus transfers.

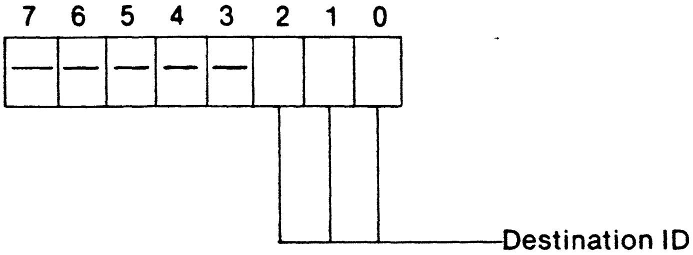
BIT 1 Reselect Enable — When this bit is a "1", the chip will respond to any attempt by a Target to reselect it. When the bit is a "0", the chip will ignore all attempts to reselect it.

BIT 0 Select Enable — When this bit is a "1", the chip will respond to attempt to select it as a Target. When it is a "0", the chip will ignore all selections.

NOTE: After being reset and completing self-diagnostics, the control register will contain all zeros.

4.4 DESTINATION ID REGISTER

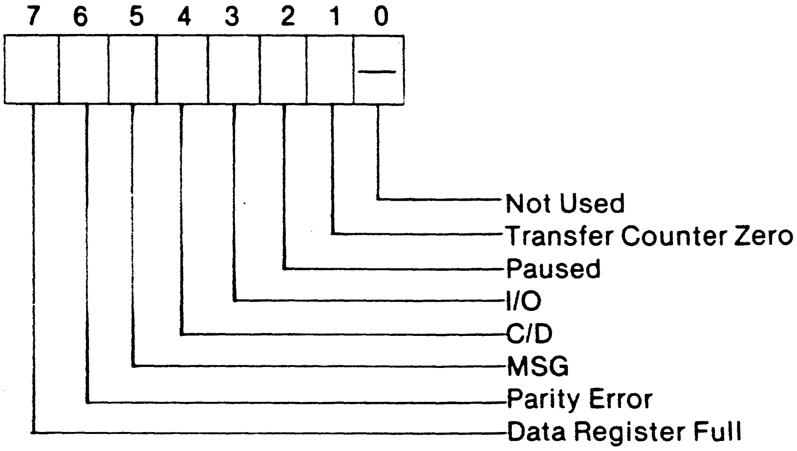
The Destination ID Register is an eight-bit register that is used to program the SCSI bus address of the destination device prior to issuing a Select or Reselect command to the chip. Bits 0-2 specify the address and bits 3-7 are always zeroes. The ID register is written (read) by activating $\overline{CS}$ with A3-A0 equal to "0011" and then pulsing $\overline{WR}$ ($\overline{RD}$).



4.5 AUXILIARY STATUS REGISTER

The Auxiliary Status Register is an eight-bit read-only register. It contains bits which indicate the status of the chip's operational condition. Some of these bits are used to determine the reason for interrupts. Therefore, the Auxiliary Status Register should always be read prior to reading the Interrupt Register when servicing interrupts. After the Interrupt Register is read, the Auxiliary Status Register bits needed to service the interrupt may change.

The Auxiliary Status Register is read by activating $\overline{CS}$ with A3-A0 = 0100 and then pulsing $\overline{RD}$. The individual bits of the Auxiliary Status Register are defined below.



BIT 7 Data Register Full

This bit indicates the status of the Data Register and must be monitored by the microprocessor during non-DMA mode commands that use the Data Register. When the DMA mode bit in the Command Register is off (0) and the command being executed is one of Send, Receive or Transfer Info commands (refer to Section 5.0 page 19, COMMANDS), data is transferred to (from) the chip by writing (reading) the Data Register. Data Register Full is set on (1) when data is written and turned off (0) when data is read. Therefore, Data Register Full should be on before taking data from the chip, and off when sending data to the chip.

The Data Register Full bit is always reset (to 0) at the time an interrupting type command is loaded into the Command Register. Therefore, when issuing such commands, the Command Register should be loaded prior to loading the Data Register and monitoring the Data Register Full flag.

BIT 6 Parity Error

When this bit is one, it indicates that the chip has detected a parity error on a byte of data received across the SCSI bus. It can be set when the chip is executing one of the Receive commands or the Transfer Info command (when the transfer is an input). This bit is reset after the Interrupt Register is read.

BIT 3-5 I/O, C/D, MSG

These bits indicate the status of the SCSI I/O, C/D, and MSG signals at all times. They define the Information Phase type being requested by the Target. These signals are significant when servicing interrupts and the chip is logically connected to the bus in the Initiator role. An interrupt will occur with any phase change. This allows the Initiator to prepare for the next phase of data transfer. These bits are only held while INT is active. The bits are coded as follows:

I/O	C/D	MSG	BUS PHASE
0	0	0	Data Out
0	0	1	Unspecified Info Out
0	1	0	Command
0	1	1	Message Out
1	0	0	Data In
1	0	1	Unspecified Info In
1	1	0	Status
1	1	1	Message In

BIT 2 Paused

When on (1), this bit indicates that the chip has aborted the command being executed in response to the Pause command. It is turned off when the interrupting type command code is loaded into the Command Register.

BIT 1 Transfer Counter Zero

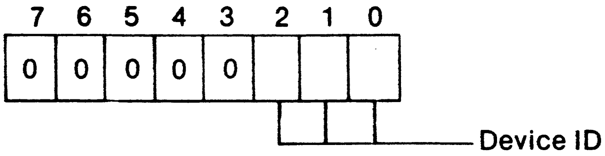
This bit is provided to indicate the status of the 24-bit Transfer Counter. When on (1), it indicates that the Transfer Counter is equal to zero. It is intended to facilitate interrupt servicing.

BIT 0 Not Used

NOTE: The Auxiliary Status Register will contain the following pattern after a Reset and self-diagnostics: 00xxx010.

4.6 ID REGISTER

The ID Register is an eight-bit read-only register that indicates the logical SCSI bus address occupied by the chip. Bit 0-2 directly reflect the logical inversion of the chip ID input signals $\overline{\text{ID0}}$-$\overline{\text{ID2}}$. The ID Register is active high whereas the ID input signals are active low. The ID Register allows the microprocessor to read the chip's SCSI bus address which would normally be strapped in hardware. Bits 3-7 of the ID Register will always be zeroes. The ID Register is read by activating $\overline{\text{CS}}$ with A3-A0 = 0101 and then pulsing $\overline{\text{RD}}$.

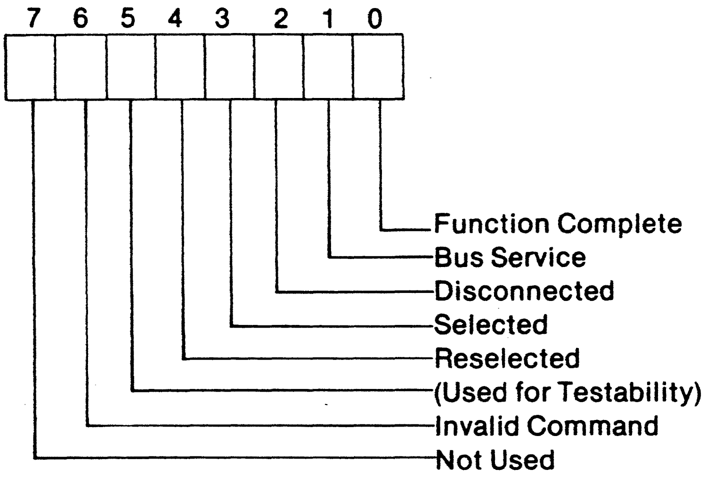


4.7 INTERRUPT REGISTER

The Interrupt Register is an eight-bit read-only register. It is used in conjunction with the Auxiliary Status Register to determine the reason for an interrupt condition. This register is read by activating $\overline{CS}$ with A3-A0 = 0110 and then pulsing $\overline{RD}$. When the Interrupt Register is read, it automatically resets itself (after the read is complete) and enables the chip for a new interrupt condition. Since the Parity Error bit in the Auxiliary Status Register is reset after a read of the Interrupt Register, and since I/O, C/D, and MSG are only held while INT is active, the Auxiliary Status Register should always be read prior to reading the Interrupt Register.

If a Selected or Reselected interrupt occurs after issuing a command that would normally cause an interrupt, the chip will ignore the last command issued. This allows the microprocessor to service the Selected or Reselected interrupt prior to proceeding with the other operation. An example of this situation is when the microprocessor issues a command to select a Target at about the same time another Target reselects the chip. If the chip sees the reselection first, the microprocessor will receive an interrupt for the reselection, and the chip will ignore the Select command, which would now be invalid since the chip is now logically connected on the SCSI bus to another device.

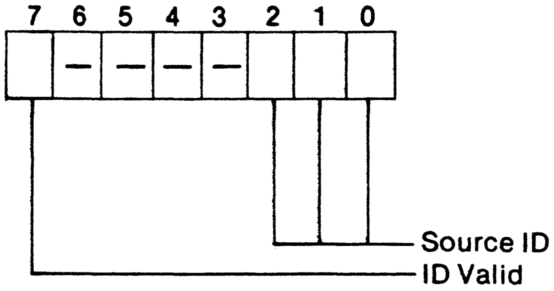
Individual interrupt conditions are described below. (Note: that for all cases, an interrupt condition is on, when the corresponding bit is a one (1), and off when zero (0).)



BIT	7	Not Used	May be either (1) or (0).
BIT	6	Invalid Command	When on (1), this bit indicates that the last command loaded into the Command Register is not valid.
BIT	5	Not Used	(Reserved for testability)
BIT	4	Reselected *	This interrupt will be on (1) when the chip has been reselected by another SCSI device. After setting this interrupt, the chip is logically connected to the bus in an Initiator role and is waiting for the Target to send REQ or disconnect from the bus.
BIT	3	Selected *	This interrupt will be on (1) whenever the chip has been selected by another SCSI device. After setting this interrupt, the chip is logically connected to the bus in the Target role and is waiting for a command to be loaded into the Command Register. * The chip will become selected (reselected) only if the ID data byte put on the SCSI bus during the Selection (Reselection) Phase has good parity and not more than one ID other than the chip's own ID is on.
BIT	2	Disconnected	This interrupt will be set on (1) when the chip is connected to the bus in the Initiator role and the Target disconnects or when the chip is executing a Select or Reselect command and the destination device does not respond before the Transfer Counter times out.
BIT	1	Bus Service	When the chip is logically connected to the bus in the Initiator role, this bit will be set on (1) whenever the Target sends a REQ which the chip cannot automatically handle. This happens when the first REQ for connection is received or when the chip is executing a Transfer Info or Transfer Pad command and either the Transfer Counter is zero or the Target changes the Information Phase type. A Bus Service interrupt may also be set if a phase change occurs before REQ is seen. This early notification will allow the Initiator extra time to prepare for a phase change in some unbuffered systems. (Note: that the chip may generate Bus Service Interrupts for phases that never request transfers. This is not an error condition, merely transitional status of I/O, C/D, and MSG.) If the chip is logically connected in the Target role, this bit will be set on (1) whenever the Initiator asserts ATN. When indicating ATN the Bus Service interrupt may occur by itself, with a Selected interrupt, or with a Function Complete interrupt.
BIT	0	Function Complete	When this bit is on (1), it indicates that the last interrupting command has completed. It is the normal successful completion interrupt for Select, Reselect, Send and Receive commands (Refer to Section 5.0 page 19, COMMANDS). During any of the Receive commands, it is set on (1) along with the parity error bit as soon as a parity error is detected. A Bus Service Interrupt may also occur simultaneously with the Function Complete if an ATN signal was activated during a Send or a Receive command. The Function Complete interrupt is also generated at the end of a Message In phase for a Transfer Info command. (See TRANSFER INFO command, page 29 for details.)

4.8 SOURCE ID REGISTER

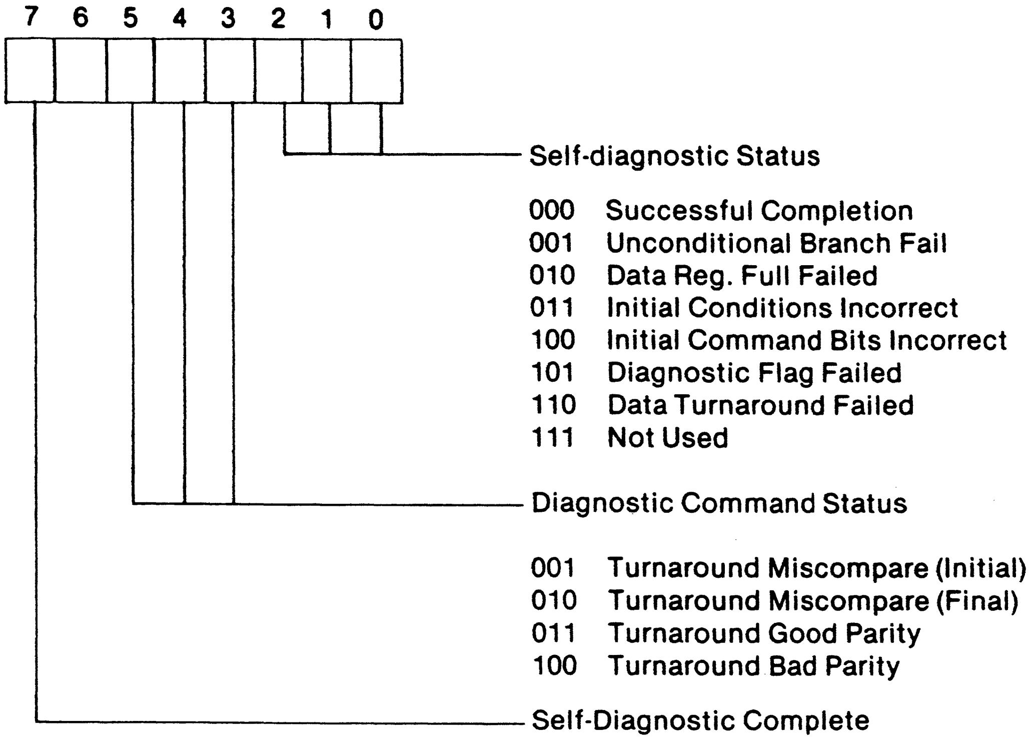
The Source ID Register is an eight-bit read-only register which contains the three-bit encoded ID of the last device which Selected or Reselected the chip. The following is the format of the Source ID Register.



The ID Valid bit indicates that the source device placed its own ID bit on the SCSI bus during the Selection Phase. The SPC chip has encoded the source ID and placed it in bits 2-0. This information remains valid until the chip disconnects from the SCSI bus, at this time the ID Valid bit is reset.

4.9 DIAGNOSTIC STATUS REGISTER

The Diagnostic Status Register is an eight-bit read-only register which indicates the result of self-diagnostics and the last diagnostic command issued to the chip. The format of the Diagnostic Status Register is shown below.



Self-diagnostic Status

000	Successful Completion
001	Unconditional Branch Fail
010	Data Reg. Full Failed
011	Initial Conditions Incorrect
100	Initial Command Bits Incorrect
101	Diagnostic Flag Failed
110	Data Turnaround Failed
111	Not Used

Diagnostic Command Status

001	Turnaround Miscompare (Initial)
010	Turnaround Miscompare (Final)
011	Turnaround Good Parity
100	Turnaround Bad Parity

Self-Diagnostic Complete

Bit 7 = 1 indicates that self-diagnostics have been completed. (NOTE: A reset will clear bits 6-3 if possible). After a reset to the chip, the microprocessor should make sure that the Diagnostic Status Register contains the following pattern before attempting any commands: 10000000. This code indicates self-diagnostics are complete and no errors were detected. After a diagnostic command has been executed, bits 6-3 will contain the resulting status, but bit 7 and bits 2-0 are not affected.

The microprocessor may read the Diagnostic Status Register by activating $\overline{CS}$ with A3-A0 = 1001 and issuing a $\overline{RD}$ pulse.

If an error is detected during self-diagnostics, the proper status is loaded into the Diagnostic Status Register and the chip halts until a Reset command or a Reset signal is asserted. Refer to the Self-Diagnostic Status Code Summary for an explanation of the individual codes.

When a diagnostic command is issued to the chip, the chip will attempt to perform the function, load a status into bits 6-3, and initiate a Function Complete Interrupt.

4.9.1 SELF-DIAGNOSTIC STATUS CODE SUMMARY

000 - Successful Completion. The chip executed all self-diagnostics following a reset and detected no errors.

001 - Unconditional Branch Failed. The chip's internal sequencer attempted an unconditional branch and failed to reach the desired location.

010 - Data Register Full Failed. The chip attempted to set and reset the Data Registet Full status bit in the Interrupt Register and failed.

011 - Initial Conditions Incorrect. The chip detected one of its internal initial conditions in the wrong state.

100 - Initial Command Bits Incorrect. The chip tested bits 6,4,2,1 and 0 of the Command Register and found at least one was not zero.

101 - Diagnostic Flag Failed. The chip failed in its attempt to set and reset its internal diagnostic flag.

110 - Data Turnaround Failed. During self-diagnostics the chip attempts to flush several bytes of data through its internal data paths. It also attempts to set and reset the Parity Error bit in the Interrupt Status Register. This status indicates that one of these operations failed.

4.10 TRANSFER COUNTER (THREE EIGHT-BIT COUNTERS)

The Transfer Counter is comprised of three, eight-bit register/counters. It is used by the chip for Send, Receive and Transfer commands that require more than a single byte of data to be transferred. It may also be used with Select and Reselect commands to set a timeout for no response. To write to (read from) the Transfer Counter, $\overline{CS}$ is activated with A3-A0 selecting a byte and then pulsing $\overline{WR}$ ($\overline{RD}$). The Transfer Counter is addressed as shown below.

A3	A2	A1	A0	SELECTED BYTE
1	1	0	0	Most Significant Byte
1	1	0	1	Middle Byte
1	1	1	0	Least Significant Byte

For Send, Receive and Transfer commands with single-byte not specified, the Transfer Counter specifies to the chip the maximum number of bytes to be sent or received before interrupting. The Transfer Counter must be loaded prior to issuing the command. When single-byte is specified, the chip neither uses nor alters the Transfer Counter. To facilitate servicing interrupts for commands that use the Transfer Counter, a bit is provided in the Auxiliary Status Register to indicate when the Transfer Counter is zero.

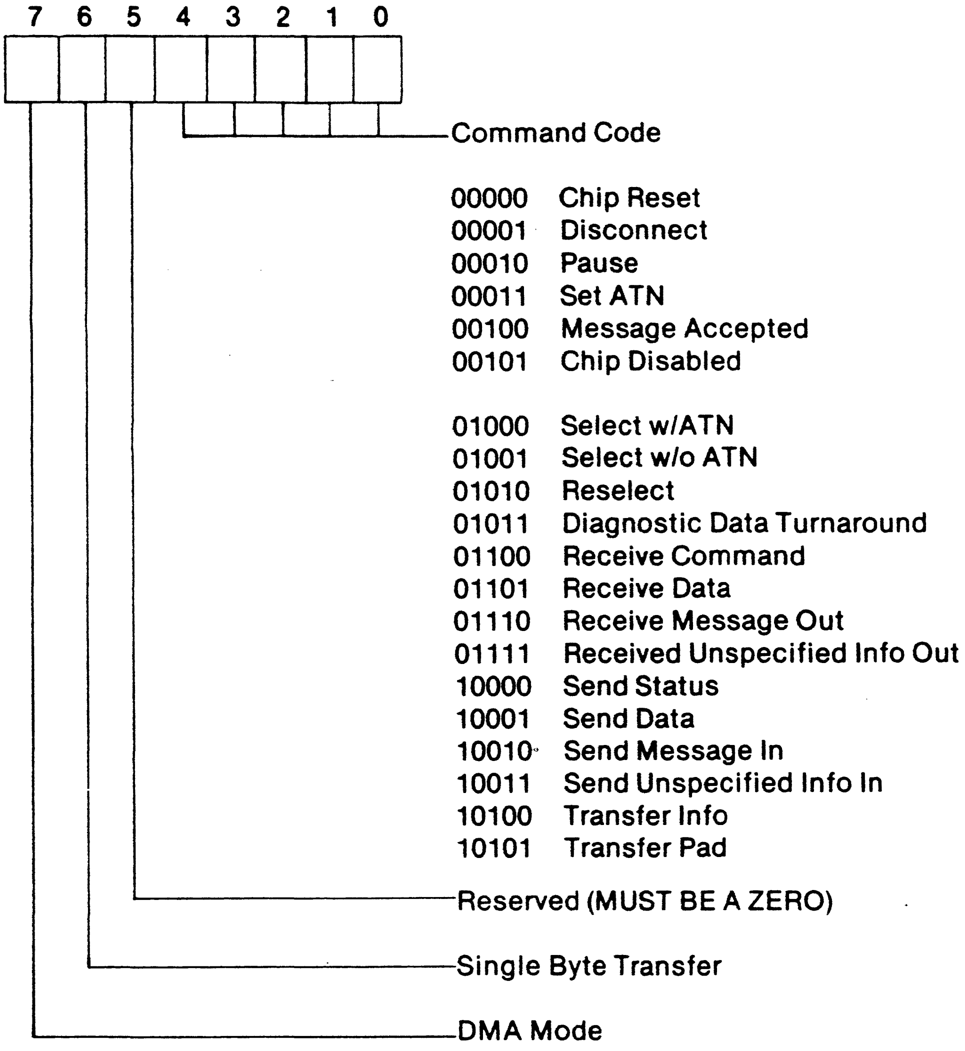
For Select and Reselect commands, the Transfer Counter specifies the number of time intervals (1024 CLK periods) that the chip will wait before automatically aborting the command due to no response (BSY) from the destination device. The Transfer Counter must be loaded prior to issuing the command. If the Transfer Counter is loaded with all zeroes, the timeout logic in the chip will be disabled, and the chip will not automatically abort the command due to no response.

SECTION 5
COMMANDS

This section defines command format, types, codes and operation. Commands are given to the chip by loading the Command Register.

5.1 COMMAND FORMAT

The bits in the Command Register are defined as follows.



Command Code

00000	Chip Reset
00001	Disconnect
00010	Pause
00011	Set ATN
00100	Message Accepted
00101	Chip Disabled
01000	Select w/ATN
01001	Select w/o ATN
01010	Reselect
01011	Diagnostic Data Turnaround
01100	Receive Command
01101	Receive Data
01110	Receive Message Out
01111	Received Unspecified Info Out
10000	Send Status
10001	Send Data
10010	Send Message In
10011	Send Unspecified Info In
10100	Transfer Info
10101	Transfer Pad

Reserved (MUST BE A ZERO)

Single Byte Transfer

DMA Mode

BIT	7	DMA Mode	This bit is applicable only for commands that use the Data Register. When this bit is on (1), it indicates that data will be transferred to (from) the Data Register using the DMA signals DREQ and $\overline{\text{DACK}}$. When it is off (0), the microprocessor must monitor the state of the Data Register Full flag in the Auxiliary Status Register. Data is then transferred by using the appropriate input/output command.
BIT	6	Single Byte Transfer	When on (1), this bit indicates that only one byte of data is to be transferred for this command. The Transfer Counter will not be used or altered by the chip. Therefore, for common single byte message and status transfers, the Transfer Counter does not need to be loaded prior to issuing a command with this bit set. When this bit is off (0). the Transfer Counter is used by the chip to determine the length of the transfer for the command.
BIT	5	Reserved	This bit is not used and should always be programmed off (0).
BIT	4-0	Command Code	These bits are used to specify the command to be executed.

5.2 COMMAND TYPES

There are two types of commands; Immediate and Interrupting. All of the Immediate commands, except for Pause, cause immediate results within three clock cycles from the time the Command Register is loaded. The Pause command is explained in a later section (See page 22, PAUSE). Interrupting commands do not result in immediate action. Their completion is always flagged by an interrupt.

Command codes 00000-00111 specify Immediate commands. Immediate commands that are listed as reserved, will be ignored if issued to the SPC chip. Command codes 01000-10101 specify Interrupting commands. When one of these codes is loaded into the Command Register, a second Interrupting command code should not be loaded until after the interrupt has occurred for the first command. However, an Immediate type command may be loaded before the interrupt for an Interrupting command occurs. If a reserved Interrupting command code is issued, the chip will respond with an Invalid Command interrupt.

5.3 INVALID COMMANDS

The user of the chip can be in one of three states at any particular time: Disconnected, connected as an Initiator, or connected as a Target. Commands are valid only in specified states. If an invalid Immediate command is issued, the chip will ignore the command. If an Interrupting command is issued in an invalid state, or a reserved Interrupting command code is issued, an Invalid Command interrupt will result. The exceptions are described below:

The microprocessor must never issue any interrupting type command when the chip is not expecting such a command. Unpredictable results will occur in this case. The following is a list of user states in which the chip is not expecting an interrupting command:

1. The chip is currently processing an Interrupting type command and has not yet set the interrupt to signal the completion.
2. The chip is currently processing an Interrupting type command, a Pause command has been issued but the Paused bit in the Auxiliary Status Register has not been set.
3. The chip is connected as an Initiator, but the Target has not yet requested an Information Transfer.
4. The chip has completed a Transfer Info or Transfer Pad command and the Target has not requested additional information or has not changed the Information Phase.

In user states three and four, described above, the microprocessor must wait for a Bus Service, Disconnected, or Function Complete interrupt.

If an interrupting command is illegitimately issued in these states, no interrupt will occur for it, and it is likely that the current function will be altered.

5.4 COMMAND SUMMARY

Below is a summary that lists all commands. In the table the following abbreviations are used.

INT = INTERRUPTING   D = DISCONNECTED   I = CONNECTED AS AN INITIATOR
IMM = IMMEDIATE   T = CONNECTED AS A TARGET

COMMAND CODE	COMMAND	TYPE	VALID STATES
00000	Chip Reset	IMM	D,I,T
00001	Disconnect	IMM	I,T
00010	Paused	IMM	D,T
00011	Set ATN	IMM	I
00100	Message Accepted	IMM	I
00101	Chip Disable	IMM	D,I,T
00110-00111	Reserved	IMM	
01000	Select w/ATN	INT	D
01001	Select w/o ATN	INT	D
01010	Reselect	INT	D
01011	Diagnostic	INT	D
01100	Receive Command	INT	T
01101	Receive Data	INT	T
01110	Receive Message Out	INT	T
01111	Receive Unspecified Info Out	INT	T
10000	Send Status	INT	T
10001	Send Data	INT	T
10010	Send Message In	INT	T
10011	Send Unspecified Info In	INT	T
10100	Transfer Info	INT	I
10101	Transfer Pad	INT	I
10110-11111	Reserved	INT	

5.5 COMMAND DEFINITIONS

5.5.1 CHIP RESET

Chip Reset immediately stops any chip operation and resets all registers, counters, etc. on the chip. It performs the same operation as the hardware "reset" input.

5.5.2 DISCONNECT

Upon receipt of this command, the chip immediately releases all SCSI bus signals and returns to a Disconnected idle state. For the Target role, this is the normal method of disconnecting from the bus when a transfer is complete. For the Initiator role, Disconnect may be used to release the bus signals as a result of a timeout condition. In this case, the chip ignores the Target and is left in the Disconnected state. For the Disconnected state, it is not valid to issue a Disconnect command. If issued, the chip will ignore this command.

5.5.3 PAUSE

Pause is an Immediate command that is valid in the Disconnected state or when logically connected to the bus as a Target device. Pause is not valid when connected as an Initiator.

When connected as a Target, the Pause command provides a means of halting a Send or Receive command without having to wait for the transfer to complete. When Pause is issued, it immediately sets a flag in the chip. Within one byte transfer cycle, the chip recognizes the flag, aborts the Send or Receive operation, and then sets the Paused status bit in the Auxiliary Status Register. At this time, the chip is still connected to the bus in the Target role, and it is waiting for another command.

The Pause command stops the Send or Receive command in an orderly manner leaving the Transfer Counter in a valid state that indicates the remaining number of bytes to be transferred. Also no REQ or ACK is asserted on the bus and no data is left in the chip waiting to be transferred. An operation that is paused may be resumed, if desired, simply by reloading the original command into the Command Register. (Note: after issuing the Pause while executing Send or Receive, it is necessary to continue transferring data with the chip (due to double-buffering) until the Paused status bit is set or an interrupt occurs.)

When in the disconnected state, Pause may be issued to abort a Select or Reselect command. After a Select or Reselect command is issued and before an interrupt occurs, a Pause command may be issued to abort the operation. The Pause command immediately sets an internal flag. If the chip has not yet won arbitration, it sets the Paused bit in the Auxiliary Status Register and waits in the disconnected state for another command. If the chip has won arbitration, it releases the bus by dropping the two ID bits with SELOUT on for a minimum of 100μs, checks for no BSYIN, and then releases the bus. After this procedure, it sets the Paused bit in the Auxiliary Status Register and waits for another command in the Disconnected state.

Since Pause is an Immediate command, it does not cause an interrupt. As previously noted, the chip sets the Paused status bit to indicate that is has been executed. If an interrupt-causing event occurs before the chip sees the pause flag set, the chip will set the interrupt. In this case, the Paused status bit is not set by the chip either before or after the interrupt. In all cases, an interrupt-causing event will take precedence over Pause. For example, in the Target role if ATN is on when Pause is issued, a Bus Service interrupt will occur and the Paused status bit will not be set.

If the Pause command is issued when the chip is Disconnected, the Paused status bit will be set by the chip, provided it has not already detected a Selection or Reselection.

5.5.4 SET ATN

The Set ATN command causes ATN to be asserted immediately if the chip is connected as an Initiator. This command is invalid and ignored if issued when the chip is Disconnected or is operating in a Target role. The ATN signal is de-asserted in a Message Out phase when the transfer count becomes zero or one byte has been transferred (in a one-byte transfer command) during the execution of a Transfer Info command.

The chip automatically sets ATN in two cases:

1. If a Select w/ATN command is issued and arbitration is won.

2. If a parity error is detected on an input byte during execution of a Transfer Info command.

5.5.5 MESSAGE ACCEPTED

The Message Accepted command is an Immediate command that is valid only when connected as an Initiator. It is used after a Transfer Info or Pad command (See pages 29,30 TRANSFER INFO and TRANSFER PAD) to indicate to the chip that ACK can be de-asserted for the last byte.

When an Initiator receives a message, a Transfer command is used. If the transfer is an input (I/O = 1) and the information is a message (MSG = 1, C/D = 1), the chip interrupts after receiving the last byte with a Function Complete interrupt. For this one special case, the chip also leaves ACK asserted on the bus. By interrupting and leaving ACK asserted, the chip gives the microprocessor a chance to interpret the message and set ATN, prior to ACK being de-asserted. This allows the chip to properly request a Message Out phase if the Initiator wants to send a "Reject Message" to the Target.

Message Accepted must always be issued after a Transfer Info for a Message In phase, whether or not Set ATN is issued, in order to have the chip de-assert ACK. If the Initiator wants to reject the message, Set ATN would be issued first followed by Message Accepted. If the message is not to be rejected, only Message Accepted is issued. (Note: until Message Accepted is issued, the Target will not send another REQ since ACK is still asserted.)

5.5.6 CHIP DISABLE

Chip Disable immediately stops all chip operations and logically disconnects it from the circuit. All outputs will be placed in a high impedance state and the chip will not respond to any commands (other than chip reset). The chip will also not respond to any activity on the SCSI bus. The only way to exit this condition is to activate the "reset" input or issue a Reset command.

5.5.7 SELECT w/ATN

This command causes the chip to attempt to select a Target. It may only be used if the microprocessor is in the Disconnected state. Any attempt to issue this command in another state will result in an Invalid Command interrupt. Before issuing this command, the microprocessor must load the Transfer Counter for a timeout on the Target's response. This value is computed according to the following formula:

Transfer Counter = Desired Timeout/ (1024 x Clock Period)

If the Transfer Counter is loaded with the value zero, the chip will wait indefinitely for a response from the Target being selected.

The microprocessor must also load the Destination ID Register with the three-bit code of the Target to be selected before issuing the Select w/ATN command.

When the chip detects the Select w/ATN command, it begins by attempting to arbitrate for control of the SCSI bus. If, at any time during arbitration the chip becomes selected or reselected, the Select w/ATN is aborted and forgotten and the chip will interrupt with one of the following conditions:

1. Selected
2. Selected and Bus Service
3. Reselected

If arbitration is won, the chip places the SCSI bus in the Selection phase with ATN asserted, and uses the Destination ID Register to identify the desired Target. At the same time, the chip begins a timer based on the value computed above. If the Target does not respond within the timeout period, the chip will disconnect from the bus and interrupt with the Disconnected flag set in the Interrupt Register. (Note: The microprocessor should never monitor the Transfer Counter Zero flag in the Auxiliary Status Register to determine when a timeout has occurred.) If the Target responds within the allotted time, the chip will interrupt with a Function Complete status. Control of the SCSI bus then belongs to the selected Target and after the interrupt status has been read, another interrupt may occur indicating either that the Target has disconnected or is requesting a transfer.

If the timeout is disabled and the Target does not respond, or if arbitration is not won, the only way to abort the Select w/ATN command is to issue the Pause command. After the Pause command is issued, it is still possible that the Function Complete or Disconnect interrupts may occur. This happens if one of the interrupts get set before the chip detects the Pause command, or if the Target responds while the chip is sequencing off the SCSI bus in a timeout condition. If the chip does not set either interrupt, it will set the Paused bit in the Auxiliary Status Register. If the microprocessor detects this bit after issuing the Pause command, then it is assured that the chip aborted the Select w/ATN command and no connection exists.

5.5.8 SELECT w/0 ATN

The Select w/o ATN is identical to the Select w/ATN command except that the ATN signal is not asserted during the Selection phase.

5.5.9 RESELECT

This command causes the chip to attempt to reselect an Initiator. It may only be used if the microprocessor is in the Disconnected state. Any attempt to issue this command in another state will result in an Invalid Command interrupt. Before issuing this command, the microprocessor must load the Transfer Counter for a timeout on the Initiator's response. This value is computed according to the following formula:

Transfer Counter = Desired Timeout/ (1024 x Clock Period)

If the Transfer Counter is loaded with the value zero, the chip will wait indefinitely for a response from the Initiator being reselected.

The microprocessor must also load the Destination ID Register with the three-bit code of the Initiator to be reselected before issuing the Reselect command.

When the chip detects the Reselect command, it begins by attempting to arbitrate for control of the SCSI bus. If, at any time during arbitration, the chip becomes selected or reselected, the Reselect is aborted and forgotten and the chip will interrupt with one of the following conditions:

1. Selected
2. Selected and Bus Service
3. Reselected

If arbitration is won, the chip places the SCSI bus in the Reselection phase using the Destination ID Register to identify the desired Initiator. At the same time, the chip begins a timer based on the value computed above. If the Initiator does not respond within the timeout period, the chip will disconnect from the bus and interrupt with the Disconnected flag set in the Interrupt Register. (Note: The microprocessor should never monitor the Transfer Counter Zero flag in the Auxiliary Status Register to determine when a timeout has occurred.) If the Initiator responds within the allotted time, the chip will interrupt with a Function Complete status. The chip (acting as the Target) is then in control of the SCSI bus, and waits for the Interrupt Register to be read by the microprocessor. After it has been read, the chip waits for a command from the microprocessor or ATN from the Initiator. If the ATN occurs, the chip will set the Bus Service interrupt. This interrupt may happen immediately after a command has been issued due to internal timing. In this case, the chip waits for the Interrupt Register to be read and the command is ignored. The chip then waits for a new command.

If the timeout is disabled and the Initiator does not respond, or if arbitration is not won, the only way to abort the Reselect command is to issue the Pause command. After the Pause command is issued, it is still possible that the Function Complete or Disconnected interrupts may occur. This happens if one of the interrupts get set before the chip detects the Pause command, or if the Initiator responds while the chip is sequencing off the SCSI bus in a timeout condition. If the chip does not set either interrupt, it will set the Paused bit in the Auxiliary Status Register. If the microprocessor detects this bit after issuing the Pause command, then it is assured that the chip aborted the Reselect command and no connection exists.

5.5.10 DIAGNOSTIC (DATA TURNAROUND)

This Interrupting command causes the chip to attempt to turn a data byte around through its internal data paths. When the command is loaded into the Command Register the Data Register Full bit is reset. The microprocessor then writes one byte into the Data Register. The chip moves the byte to another register and compares the contents of the Data Register. The byte is then moved to a third register (the SCSI output register) and good parity is generated if bit 6 of the command is off (0); bad parity is generated if bit 6 is on (1). Finally, the chip moves the byte back to the Data Register and compares it with the contents of the second register. Based on these comparisons and parity checking, the chip stores a result into the Diagnostic Status Register and sets the Function Complete interrupt. After reading the Interrupt Register, the microprocessor should make sure the Data Register Full bit is on (1) and read the contents of the Data Register. If the Data Register Full bit is not on (0), then an error has occurred. The following is a list of codes which are loaded into bits 6-3 of the Diagnostic Status Register as a result of this command.

BIT 6543	RESULT
0001	Data Miscompare (INITIAL)
0010	Data Miscompare (FINAL)
0011	Good Parity Detected
0100	Bad Parity Detected

5.5.11 RECEIVE COMMANDS

The Receive commands are Interrupting commands that are valid only when connected as a Target device. They are used by the Target to receive commands, data, and message information from an Initiator.

The Receive commands transfer data; therefore, the Single Byte Transfer and DMA mode bits in the Command Register are valid for these commands. If the Single Byte Transfer bit is off (0), the Transfer Counter must be loaded before a Receive command is issued to the chip. In this case, the chip uses the Transfer Counter to determine the number of bytes to receive.

When a Receive command is issued, the chip immediately resets the Data Register Full bit in the Auxiliary Status Register. The chip then drives the I/O, C/D, and MSG outputs for the proper information phase as follows.

COMMAND NAME	I/O	C/D	MSG
Receive Command	0	1	0
Receive Data	0	0	0
Receive Message Out	0	1	1
Receive Unspecified Info Out	0	0	1

The chip then proceeds to request and receive the specified number of information bytes. The DMA mode bit in the Command Register determines how the chip transfers these bytes from its Data Register to the microprocessor.

When a Receive command is terminated, the chip generates an interrupt. The following two events can cause termination:

1. The operation completes successfully; the Transfer Counter is zero. This event results in a Function Complete interrupt with the Parity Error bit in the Auxiliary Status Register off (0). If the initiator activated ATN during the operation, the Bus Service bit will also be on.
2. A Parity Error occurs. The last byte transferred is the byte that caused the error. This event causes a Function Complete interrupt with the Parity Error bit in the Auxiliary Status Register on (1). If the Initiator activated ATN during the operation, the Bus Service bit will also be on.

After any of the interrupts, the chip is always left in the connected Target state. The Transfer Counter indicates the number of bytes remaining to be transferred (zero if completed successfully, and the Data Register is empty (the last byte received is sent to the microprocessor). Also, ACK and REQ are inactive on the bus.

(Note: if a Bus Service interrupt alone occurs after issuing a Receive command, the Initiator activated ATN before the chip began executing the command. In this case, the command is ignored by the chip.)

A Receive command may be stopped prior to an interrupt causing event by issuing a Pause command. Operation of the Pause command is explained in an earlier section (See page 22, PAUSE). In the event the Initiator does not respond, or stops responding, the chip is left in a state where it cannot respond to a Pause command. For this case, a Disconnect command can be used to abort the command and the connection. The Disconnect command is explained in an earlier section (See page 22, DISCONNECT).

5.5.12 SEND COMMANDS

The Send commands are Interrupting commands that are valid only when connected to the bus in the Target role. They are used by a Target to send status, data, and message information to an Initiator.

The Send commands transfer data, and therefore, the Single Byte Transfer and DMA mode bits in the Command Register are valid for these commands. If the Single Byte Transfer bit is off (0), the Transfer Counter must be loaded before a Send command is issued to the chip. In this case, the chip uses the Transfer Counter to determine the number of bytes to send.

When a Send command is issued, the chip immediately resets the Data Register Full bit in the Auxiliary Status Register. Therefore, the first byte of data for the transfer cannot be put into the Data Register until after a Send command is loaded into the Command Register.

In executing a Send command, the chip drives the I/O, C/D, and MSG outputs for the proper information phase. These lines are logically driven for each Send command as shown below.

COMMAND NAME	I/O	C/D	MSG
Send Status	1	1	0
Send Data	1	0	0
Send Message In	1	1	1
Send Unspecified Info In	1	0	1

After resetting Data Register Full and driving I/O, C/D, and MSG, the chip then proceeds to monitor Data Register Full, take the data from the Data Register, and send it to the Initiator. The DMA mode bit in the Command Register specifies how the data is loaded into the chip.

After interrupting, the chip is left in the connected Target state, and ACK and REQ are inactive on the bus. When the transfer is complete, the chip interrupts with a Function Complete Interrupt. If the Initiator activated ATN during the transfer, a Bus Service bit will also be set by the chip.

(Note: if a Bus Service interrupt alone occurs after issuing a Send command, the Initiator activated ATN before the chip began executing the command. In this case, the command is ignored by the chip.)

A Send command may be stopped prior to an interrupt causing event by issuing a Pause command. Operation of the Pause command is explained in an earlier section (See page 22, PAUSE). In the event the Initiator does not, or stops responding, the chip is left in a state where it cannot respond to a Pause command. For this case, a Disconnect command can be used to abort the command and the connection. The Disconnect command is explained in an earlier section (See page 22, DISCONNECT).

5.5.13 TRANSFER INFO

The Transfer Info command is an Interrupting command that is valid only when connected to the bus in the Initiator role. It is used by the Initiator for all information transfers across the SCSI bus.

A Transfer Info command is issued by an Initiator in response to a Bus Service interrupt. The Bus Service interrupt, as explained in a previous section (See page 14, INTERRUPT REGISTER), is received by the connected Initiator upon the following conditions: receiving the first REQ from a Target, a previous command has completed and the Target changes phases, the Target changes phases before termination, or when a previous command has completed and the Target is requesting more information. It is not valid to issue a Transfer Info command without having a Bus Service interrupt, because the Target requests and controls all transfers. The chip will only permit one Transfer Info or Transfer Pad per Bus Service interrupt.

After an Initiator receives a Bus Service interrupt, and prior to issuing a Transfer Info command, the I/O, C/D, and MSG bits from the Auxiliary Status Register (read prior to reading the Interrupt) should be examined to determine the type of information phase and the direction of transfer requested by the Target. The Initiator then prepares for the transfer. If the Single Byte Transfer bit is not going to be set in the Command Register, the Transfer Counter must be loaded prior to issuing the Transfer Info command. This is done in order to specify to the chip the maximum number of bytes to be transferred.

When a Transfer Info is issued, the chip immediately resets the Data Register Full bit in the Auxiliary Status Register. For this reason, the first byte of data for an output operation cannot be loaded into the Data Register until after the command is loaded into the Command Register. The chip then proceeds with the transfer, expecting data to be read from (input), or written to (output), its Data Register as indicated by the DMA Mode bit in the Command Register. The chip automatically detects the direction of the transfer from the I/O bit which is stored in the Auxiliary Status Register.

The chip continues a transfer until an interrupt causing event occurs. The following four events will cause the chip to terminate and interrupt.

1. The maximum number of bytes specified have been transferred and the Target activated REQ or the Information Phase changed. This event results in a Bus Service Interrupt. Either single byte transfer was specified or the Transfer Counter is zero as indicated by a bit in the Auxiliary Status Register. The Target may or may not have changed the information phase type. The I/O, C/D, and MSG bits in the Auxiliary Status Register need to be examined at the time of the interrupt to determine what phase the Target is requesting.

   (Note: due to early notification of the phase change, a phase may be selected spuriously and not transfer any data. The microprocessor should not consider this an error condition.)

2. The Target changes the information phase type before the maximum number of bytes are transferred. This event also causes a Bus Service interrupt. The new information phase may be determined by examining the I/O, C/D, and MSG bits in the Auxiliary Status Register. The Transfer Counter may be read at the time of the interrupt to determine the number of bytes remaining to be transferred. When this interrupt occurs for an output transfer, the chip may take one more byte from the microprocessor than it transfers, because of pre-fetching. However, the Transfer Counter still reflects the number of bytes remaining to be transferred.

3. The Target releases the bus by dropping BSY. This event results in a Disconnected interrupt. Following this interrupt, the chip is no longer in the Initiator role. It now remains in the Disconnected state.

4. The last byte of a Message Input phase has been received. This event results in a Function Complete interrupt. For this case, ACK is left active on the bus to allow the microprocessor to Set ATN for the purpose of rejecting the message. After this interrupt is received and a Set ATN is issued (if desired), a Message Accepted must be issued to turn off ACK for the last byte of the Message In phase.

For input transfers (I/O = 1), the chip checks parity for each byte received if the Parity Enable bit in the Control Register is on. When checking parity and the parity error occurs, the chip activates ATN prior to deactivating ACK for the byte that causes the error. It also turns on the Parity Error bit in the Auxiliary Status Register. The parity error, however, does not result in an interrupt. The chip waits for one of the four events listed above before interrupting. Therefore, the Parity Error bit should be examined when servicing any interrupt after issuing Transfer Info command for an input transfer.

If ATN is asserted by the chip, either because of a parity error or because a SET ATN command is issued, the ATN will remain asserted until the end of the connection, or until a Message Out is transferred. Therefore, during each cycle of a Transfer Info operation for output, the chip checks for a message phase (C/D = 1, MSG = 1) and also either a single byte transfer or the Transfer Counter set at zero. If these conditions exist, the chip turns off ATN prior to activating ACK for the last byte of the message.

As previously stated, a Transfer Info normally terminates with an interrupt. If a Transfer Info command must be aborted, possibly because of a timeout violation, either a Chip Reset or a Disconnect command can be used. It is noted, however, that although these commands will force the chip into a disconnected state, the Target device is left on the bus. A SCSI bus reset, which is not a chip function, is the only way an Initiator can force a Target to disconnect.

5.5.14 TRANSFER PAD

The Transfer Pad command is an Interrupting command that is valid only when connected to the bus as an Initiator. It is similiar to the Transfer Info command except that the data transfer between the chip and the microprocessor bus will be different.

Transfer Pad can be used by an Initiator to continue handshaking with a Target without giving data to, or taking data from, the chip. This may be useful if the Target requests an invalid Information Transfer Phase. The chip operates in the same manner as it does for a Transfer Info command, except that for output transfers it takes only one byte of data from the microprocessor and sends the same byte repeatedly until the transfer terminates. For input transfers, it accepts data from the SCSI bus but does not check parity or send it to the microprocessor. Though data is not exchanged with the microprocessor bus, the Transfer Counter is still used by the chip so that a maximum number of pad bytes can be specified.

Protocol for using a Transfer Pad command is the same as the Transfer Info except that the DMA Mode bit has significance only for output transfers. The Transfer Pad terminates because of the same four events that cause a Transfer Info command to terminate. Also, similar to the Transfer Info command, Chip Reset and Disconnect can be used to abort the command.

SECTION 6
BUS INITIATED FUNCTIONS

6.1 SELECTION

If the Select Enable bit in the Control Register is on, the chip may be selected by another SCSI device to be a TARGET for an I/O operation. Selection occurs in the chip only if all the following conditions exist: SELOUT = 0, BSYIN = 0, SELIN = 1, I/O = 0, the chip's ID bit is asserted by the selecting device on the data bus, no more than one other ID bit (the Initiator's) is asserted on the data bus and data bus parity is good.

When all of these conditions exist, the chip is selected. It then encodes the Initiator's ID and loads it into bits 2-0 of the Source ID Register. The chip also detects whether or not the Initiator asserted its ID during selection, and either sets or resets the ID Valid bit in the Source ID Register.

The chip then asserts BSYOUT, waits for SELIN to turn off, and proceeds to take one of the following actions as a result of being selected:

1. If ATN is not asserted by the Initiator during selection, the chip generates a Selected interrupt indicating that the chip is connected as a Target.
2. If ATN is asserted, the chip simultaneously generates Selected, and Bus Service interrupts, indicating that the chip is connected as a Target and ATN is asserted.

6.2 RESELECTION

If the Reselect Enable bit in the Control Register is on, the chip may be reselected by a SCSI Target device. Reselection occurs only if SELOUT = 0, SELIN = 1, BSYIN = 0, I/O = 1, the chip's ID bit and the Target's ID bit are asserted on the data bus, no other ID bits are asserted, and data bus parity is good.

When all of these conditions exist, the chip is reselected. It then encodes the Target's ID and loads it into the Source ID Register. The chip also sets the ID Valid bit in the Source ID Register.

The chip then asserts BSYOUT and waits for SELIN to be released by the Target. When the chip detects SELIN = 0, it de-asserts BSYOUT and then generates a Reselected interrupt.

Reselection is now complete and the chip is in the connected Initiator state.

SECTION 7
INITIALIZATION

The SCSI device may be initialized by asserting RST for a period of at least 100ns, or by issuing a Chip Reset command to the device. The NCR 5385E will respond to the RST pulse or the Chip Reset command, by immediately disconnecting from the SCSI bus, initializing all storage elements and executing an internal self-diagnostic program. The self-diagnostic is explained in a previous section (See page 17, Diagnostic Status Register). The following table lists the status of all registers after the initialization procedure.

	7	6	5	4	3	2	1	0
Data Register	x	x	x	x	x	x	x	x
Command Register	0	0	0	0	0	0	0	0
Control Register	0	0	0	0	0	0	0	0
Destination ID Register	0	0	0	0	0	0	0	0
Auxiliary Status Register	0	0	x	x	x	0	1	0
ID Register	0	0	0	0	0	x	x	x
Interrupt Register	0	0	0	0	0	0	0	0
Source Register	0	0	0	0	0	1	1	1
Diagnostic Status Register	1	x	x	x	x	x	x	x
Transfer Counter (MSB)	0	0	0	0	0	0	0	0
Transfer Counter (2nd)	0	0	0	0	0	0	0	0
Transfer Counter (LSB)	0	0	0	0	0	0	0	0

x = Unknown

TABLE 7.1 REGISTER INITIALIZATION

The controlling processor should loop on reading the Diagnostic Status Register until the Self-Diagnostic Complete bit (bit 7) is on (1). This should take approximately 350 clock cycles after reset occurs. The processor should then check the remaining bits in this register for all zeroes (no errors), and then load the Control Register enabling the proper bits to begin operation. The SCSI Protocol Controller is now connected to the SCSI bus in a disconnected state. It is ready to receive commands from the controlling processor or respond to (re) selection attempts.

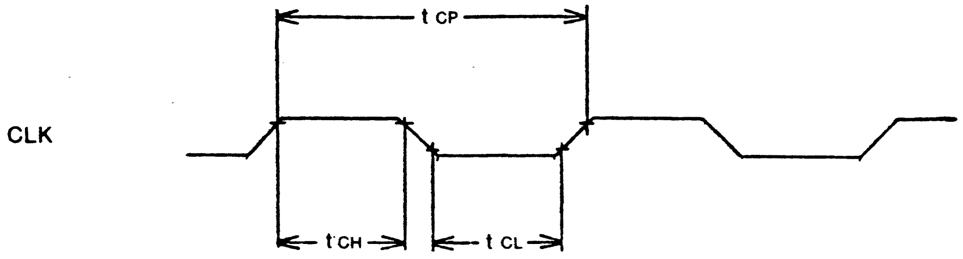
SECTION 8
EXTERNAL CHIP TIMING

Timing requirements must be over the operating temperature (0-70°C) and voltage (4.75 to 5.25V) ranges. Loading for all output signals, except $\overline{\text{SBEN}}$, is assumed to be four low-power Schottky inputs, including 50 pF capacitance. Loading for $\overline{\text{SBEN}}$ is assumed to be ten low-power Schottky inputs, including 100 pF capacitance.

8.1 MICROPROCESSOR INTERFACE

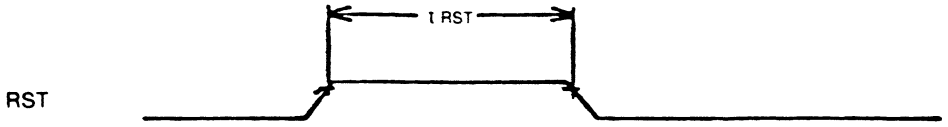
8.1.1 CLK

NAME	DESCRIPTION	MIN	MAX	UNITS
t_{CP}	Clock Period	100	200	ns
t_{CH}	Clock High	.45 t_{CP}	.55 t_{CP}	
t_{CL}	Clock Low	.45 t_{CP}	.55 t_{CP}	



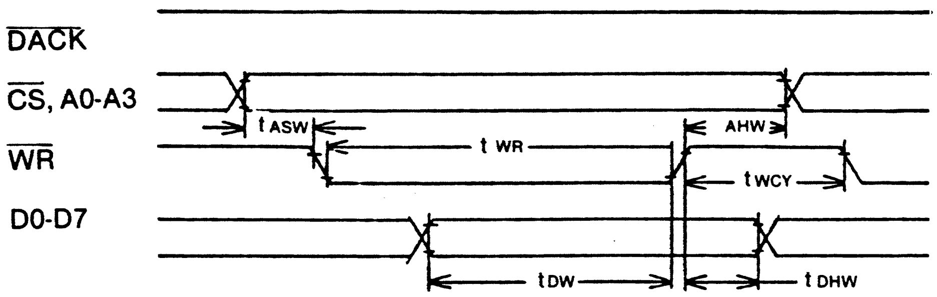
8.1.2 RESET

NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{RST}	Reset Pulse Width	100			ns



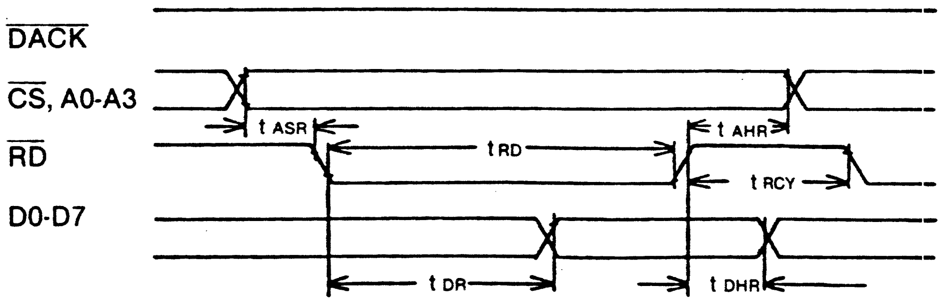
8.1.3 MPU WRITE

NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{ASW}	Address Set-up Time	0			ns
t_{WR}	$\overline{\text{WR}}$ Pulse Width	95			ns
t_{DW}	Data-to $\overline{\text{WR}}$ High	50			ns
t_{AHW}	Address Hold Time	0			ns
t_{DHW}	Data Hold Time	20			ns
t_{WCY}	$\overline{\text{WR}}$ Off to $\overline{\text{WR}}$ or $\overline{\text{RD}}$ On	125			ns



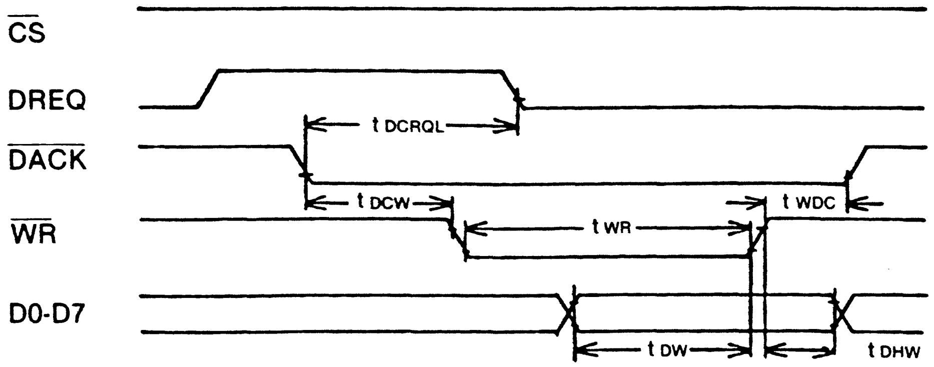
8.1.4 MPU READ

NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{ASR}	Address Set-up Time to $\overline{\text{RD}}$	0			ns
t_{RD}	$\overline{\text{RD}}$ Pulse Width	125			ns
t_{DR}	$\overline{\text{RD}}$ to Data			90	ns
t_{AHR}	Address Hold Time	0			ns
t_{DHR}	Data Hold Time	10			ns
t_{RCY}	$\overline{\text{RD}}$ Off to $\overline{\text{WR}}$ or $\overline{\text{RD}}$ On	125			ns



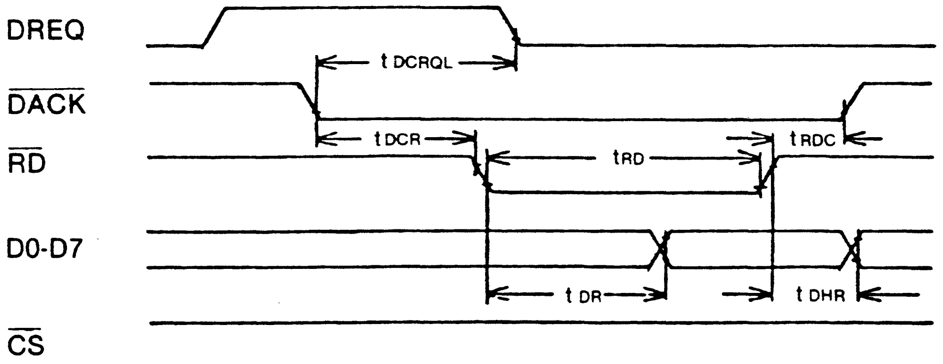
8.1.5 DMA WRITE

NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{DCRQL}	$\overline{\text{DACK}}$ to DREQ Low	0		40	ns
t_{DCW}	$\overline{\text{DACK}}$ to $\overline{\text{WR}}$	0			ns
t_{WR}	$\overline{\text{WR}}$ Pulse Width	95			ns
t_{WDC}	$\overline{\text{WR}}$ High to $\overline{\text{DACK}}$ High	0			ns
t_{DHW}	Data Hold Time	20			ns
t_{DW}	Data to $\overline{\text{WR}}$ High	50			ns



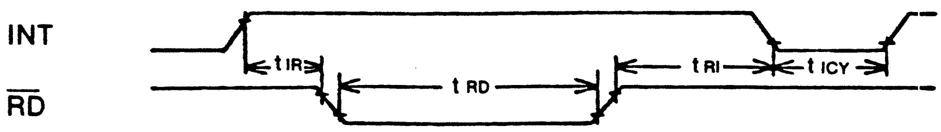
8.1.6 DMA READ

NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{DCRQL}	$\overline{\text{DACK}}$ to DREQ Low	0		40	ns
t_{DCR}	$\overline{\text{DACK}}$ to $\overline{\text{RD}}$	0			ns
t_{RD}	$\overline{\text{RD}}$ Pulse Width	95			ns
t_{RDC}	$\overline{\text{RD}}$ High to $\overline{\text{DACK}}$ High	0			ns
t_{DHR}	Data Hold Time	10			ns
t_{DR}	$\overline{\text{RD}}$ to Data			80	ns



8.1.7 INTERRUPT

NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{IR}	INT to $\overline{RD}$	0			ns
t_{RD}	$\overline{RD}$ Pulse Width	95			ns
t_{RI}	$\overline{RD}$ High to INT Low			125	ns
t_{ICY}	INT Off to INT On	125			ns



8.2 SCSI INTERFACE

8.2.1 SELECTION (INITIATOR)

NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
tBF	Bus Free	385			ns
tBIA(5)	BSYIN low to ARB high	1.2		2.6	us
tSLA	SELOUT high to ARB low & ID bit Disabled	3.2			us
tBIBO (5)	BSYIN low to BSYOUT high	1.2		2.8	us
tBCD	Bus Clear Delay			225	ns
tAD	Arbitration Delay	3.0			us
tPC	Priority check to SELOUT	0			ns
tBID (5)	BSYIN low to ID bit high	1.2		2.9	us
tADV	Arbitration Data Valid to Priority Check	0			ns
tSI	SELOUT to IGS	2.0			us
tIDBL	Target ID high to BSYOUT low	1.1			us
tBOBI	BSYOUT low to BSYIN low	0		400	ns
tBSL	BSYIN high to SELOUT low	800			ns
tDID	SBEN active to Bus enabled	150			ns

NOTES:

1. The chip ensures that the bus remains free (BSYIN and SELIN inactive) for tBF before attempting arbitration.

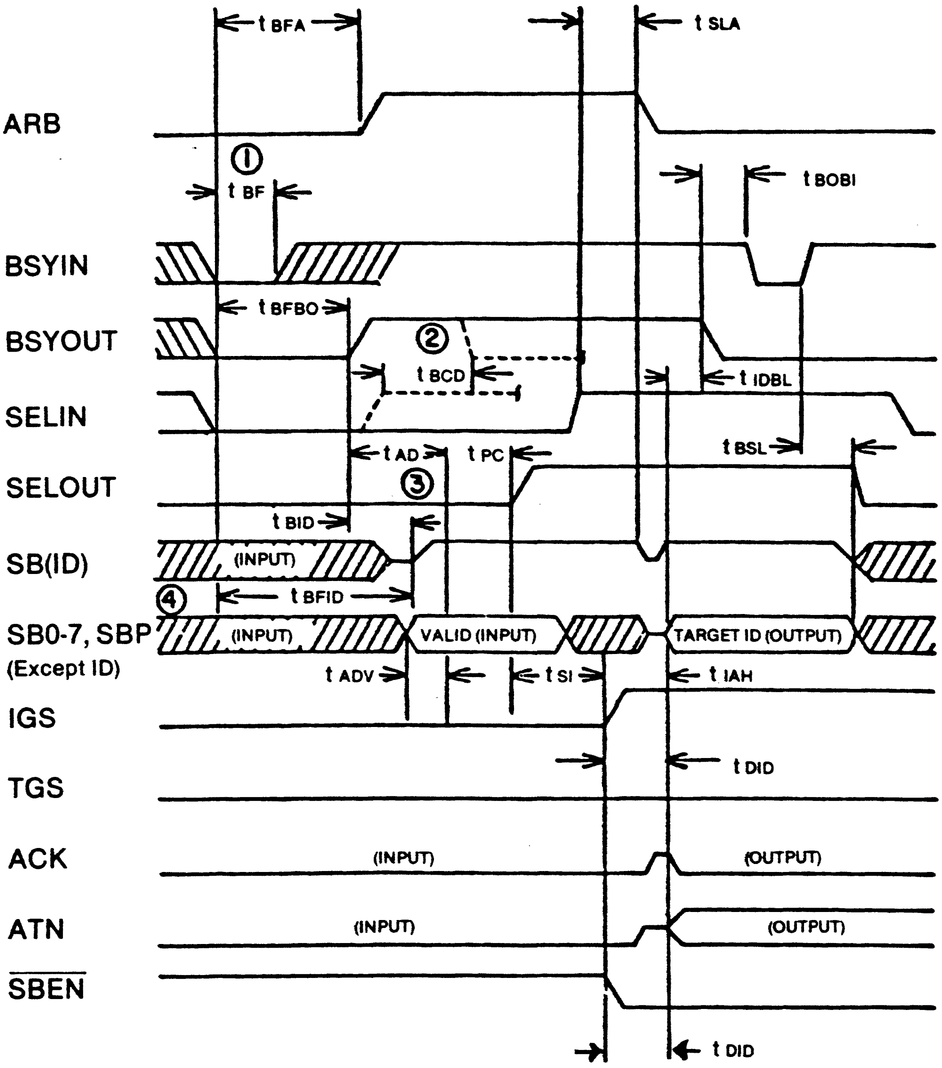
2. If SELIN becomes active at any time during arbitration, the chip must deassert BSYOUT within tBCD.

3. The chip waits (tAD), and then checks to see if arbitration is won (tPC). The chip then asserts SELOUT if arbitration is won.

4. One of the data bits is assigned as an ID bit by the IDO-ID2 signals. During Bus Free, the chip places all of the data bits, including ID, in a high impedance state. During arbitration the chip enables its ID bit and drives it high, but the remainder of the data bits remain in the high impedance state for reading.

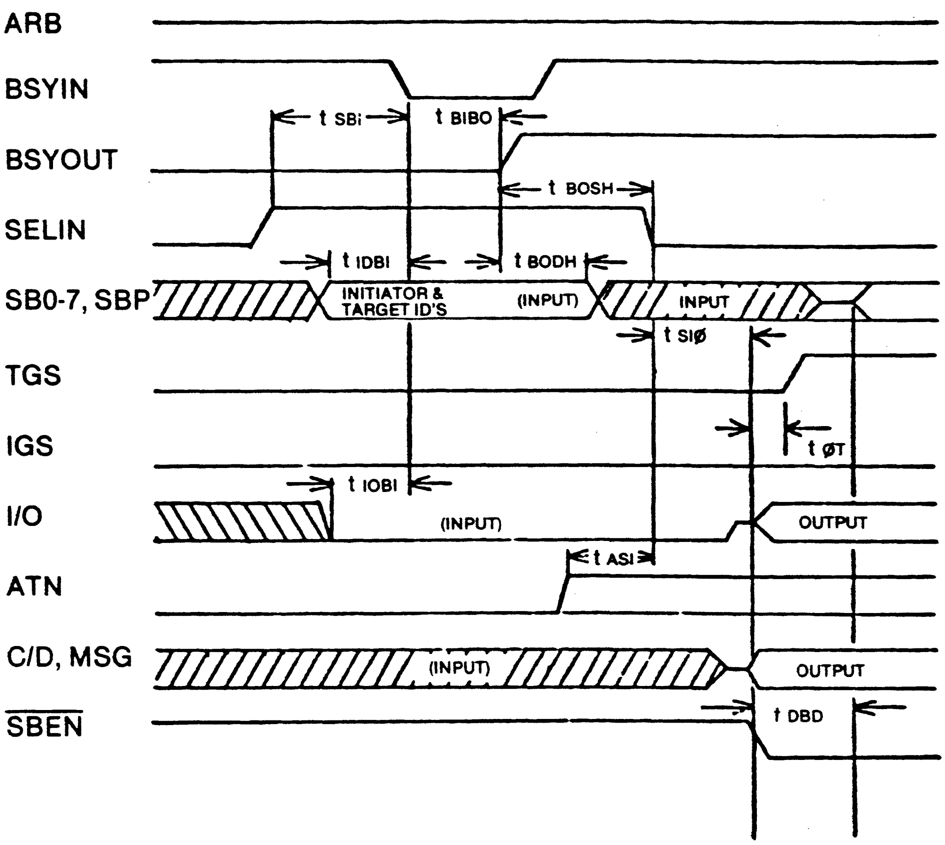
5. To verify these timings in a test environment, the user must allow a minimum of 45 clock cycles after the select command has been issued before the device begins to check for BSYIN low.

8.2.1 SELECTION (INITIATOR)



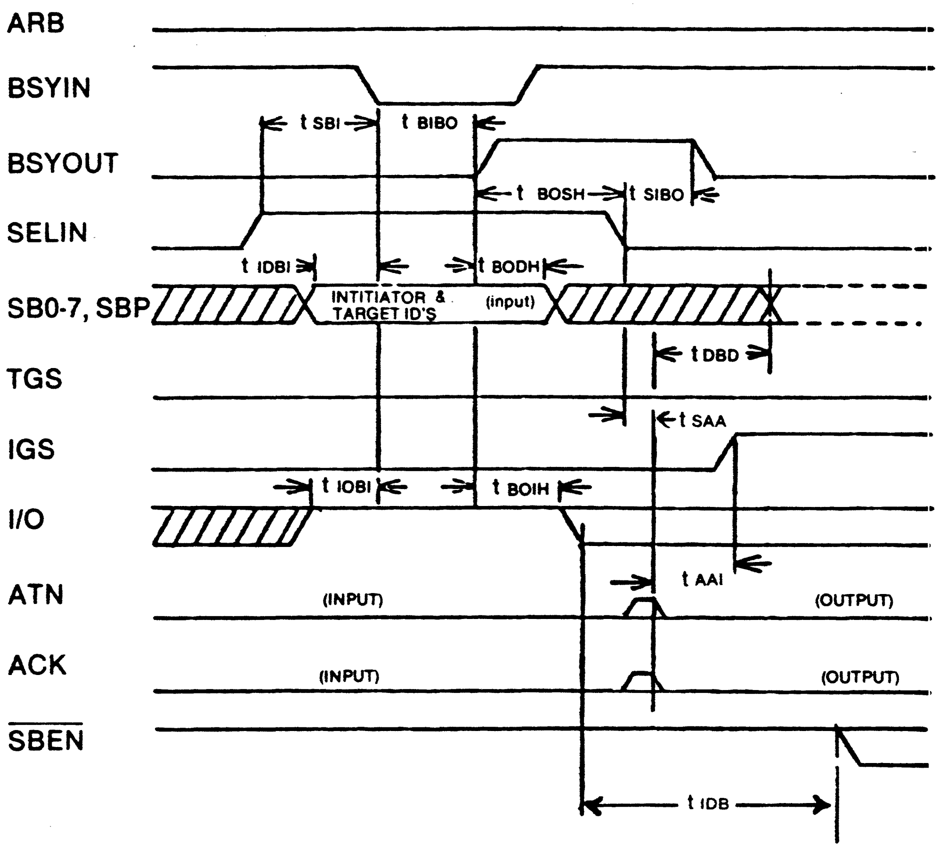
8.2.2 SELECTION (TARGET)

NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
T_{SBI}	SELIN high to BSYIN low	50			ns
t_{IDBI}	ID's valid to BSYIN low	0			ns
t_{IOBI}	I/O low to BSYIN low	0			ns
t_{BIBO}	BSYIN low to BSYOUT high	0		2.0	us
t_{BODH}	BSYOUT high Data Hold	0			ns
t_{BOSH}	BSYOUT high SELIN Hold	0			ns
t_{ASI}	ATN high to SELIN low	0			ns
t_{SIO}	SELIN low to Phase signals Enabled	150			ns
t_{OT}	Phase signals enabled to TGS High	150			ns
t_{DBD}	SBEN low to Data Bus Enabled	150			ns



8.2.3 RESELECTION (INITIATOR)

NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{SBI}	SELIN high to BSYIN low	50			ns
t_{IDBI}	ID's valid to BSYIN low	0			ns
t_{IOBI}	I/O high to BSYIN low	0			ns
t_{BIBO}	BSYIN low to BSYOUT high	0		2.0	µs
t_{BODH}	BSYOUT high Data Hold	0			ns
t_{BOSH}	BSYOUT high SELIN Hold	0			ns
t_{BOIH}	BSYOUT high I/O hold	0			ns
t_{SIBO}	SELIN low to BSYOUT low	0			ns
t_{SAA}	SELIN low to ACK & ATN enabled	750			ns
t_{AAI}	ACK & ATN enabled to IGS high	150			ns
t_{IDB}	I/O low to SBEN low	0			ns
t_{DBD}	SBEN low to Data Bus Enabled	150			ns



8.2.4 RESELECTION (TARGET)

NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{BF}	Bus Free	385			ns
t_{BIA} (5)	BSYIN low to ARB high	1.2		2.6	us
t_{SLA}	SELOUT high to ARB low & ID bit Disabled	3.2			us
t_{BIBO} (5)	BSYIN low to BSYOUT high	1.2		2.8	us
t_{BCD}	Bus Clear Delay			225	ns
t_{AD}	Arbitration Delay	3.0			us
t_{PC}	Priority check to SELOUT	0			ns
t_{BID} (5)	BSYIN low to ID bit high	1.2		2.9	us
t_{ADV}	Arbitration Data Valid to Priority Check	0			ns
t_{SO}	SELOUT Phase signals Enabled & SBEN Low	2.4			us
t_{OT}	Phase Signals Enabled to TGS High	150			ns
t_{DID}	SBEN low to Bus Enabled	150			ns
t_{IDBL}	INITIATOR ID high to BSYOUT low	2.7			us
t_{BOBI}	BSYOUT low to BSYIN	0		400	ns
t_{BIBO}	BSYIN high to BSYOUT high	0.7		2.0	us
t_{BSL}	BSYOUT high to SELOUT low	450			ns

NOTES:

1. The chip ensures that the bus remains free (BSYIN and SELIN inactive) for TBF before attempting arbitration.

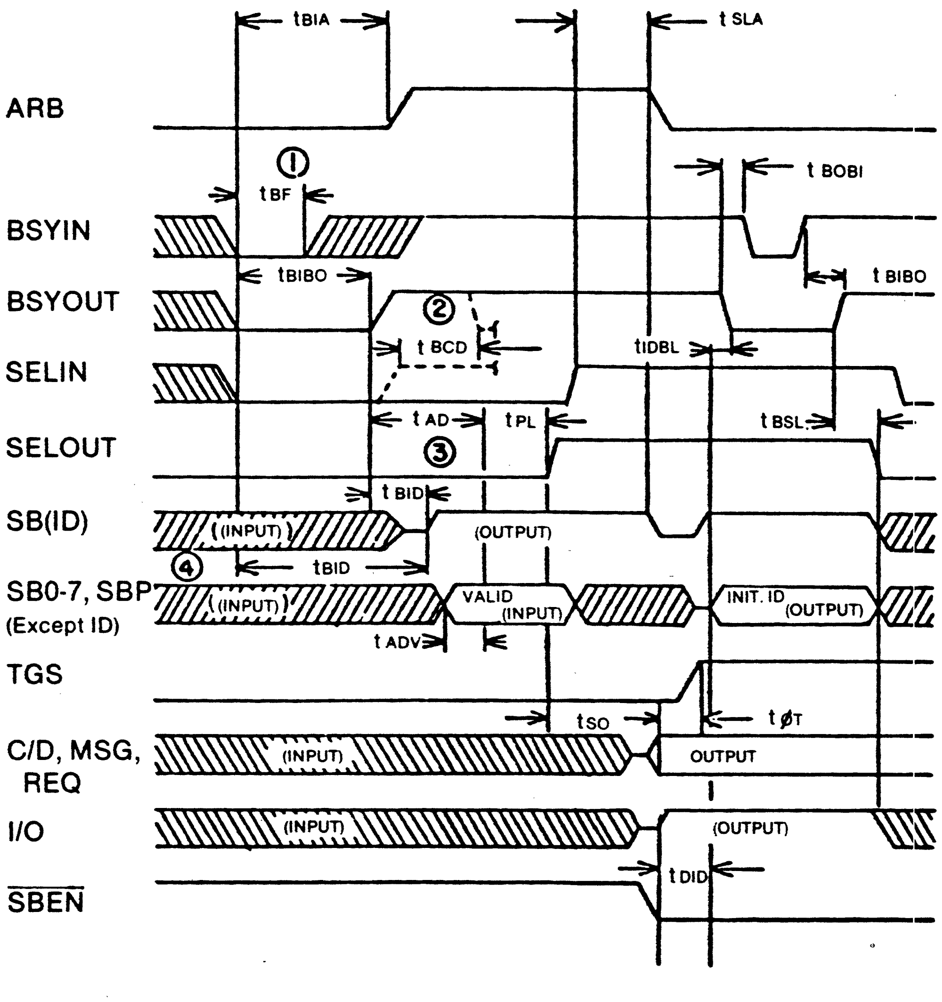
2. If SELIN becomes active at any time during arbitration, the chip must deassert BSYOUT within t_{BCD}.

3. The chip waits (t_{AD}), and then checks to see if arbitration is won (t_{PC}). The chip then asserts SELOUT if arbitration is won.

4. One of the data bits is assigned as an ID bit by the IDO-ID2 signals. During Bus Free, the chip places all of the data bits, including ID, in a high impedance state. During arbitration the chip enables its ID bit and drives it high, but the remainder of the data bits remain in the high impedance state for reading.

5. To verify these timings in a test environment, the user must allow a minimum of 45 clock cycles after the select command has been issued before the device begins to check for BSYIN low.

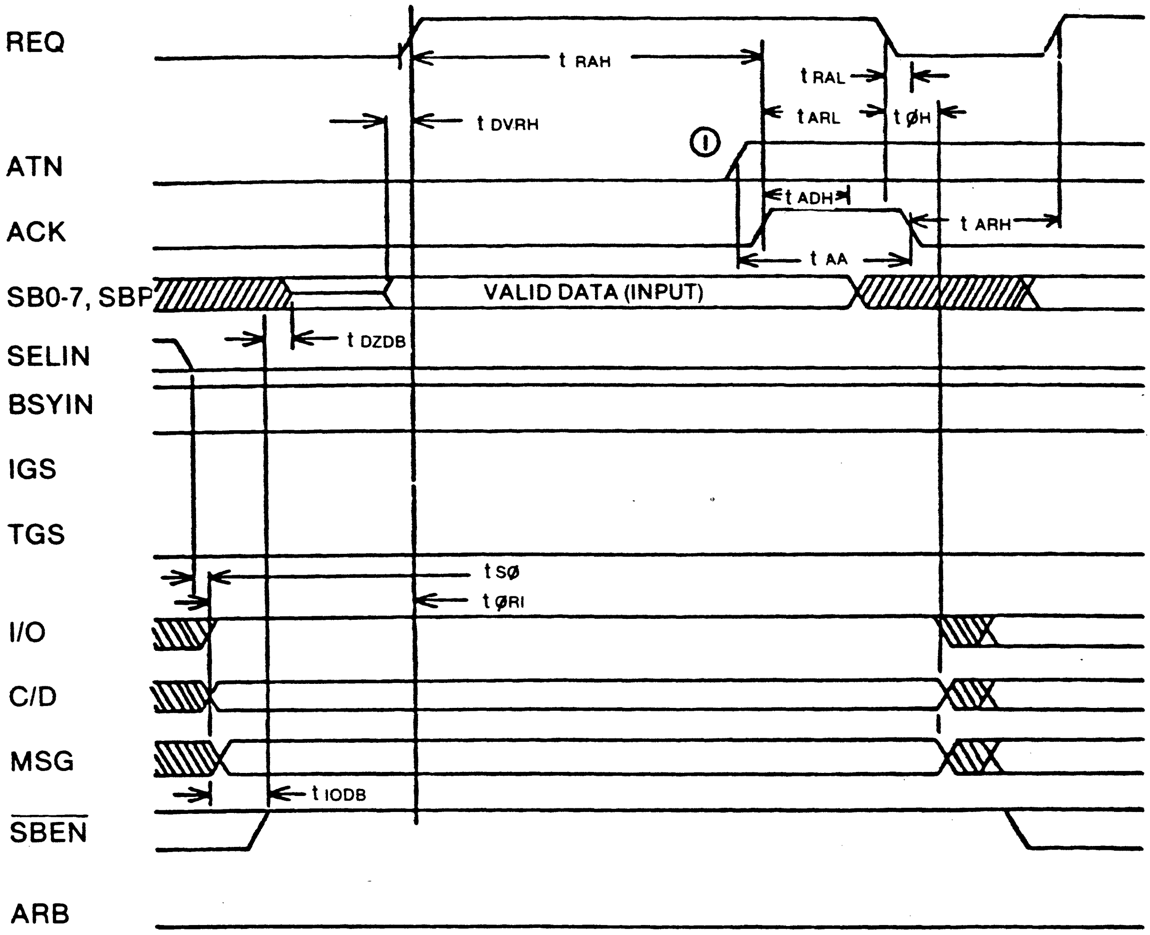
8.2.4 RESELECTION (TARGET)



8.2.5 INFORMATION TRANSFER PHASE INPUT (INITIATOR)

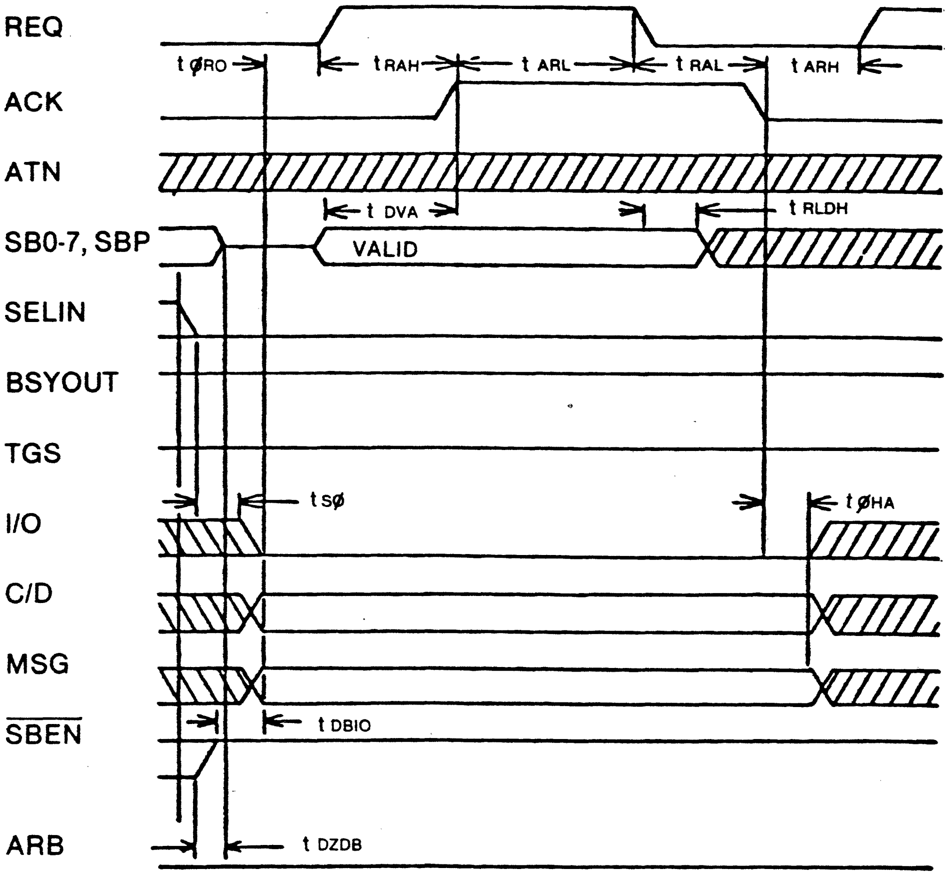
NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{DVRH}	Data Valid to REQ high	0			ns
$t_{\emptyset RI}$	Phase Valid to REQ high	100			ns
t_{RAH}	REQ high to ACK high	0			ns
t_{RAL}	REQ low to ACK low	0			ns
t_{AA}	ATN high to ACK low	100			ns
$t_{S\emptyset}$	SELIN low to Phase change	0			ns
$t_{\emptyset H}$	Phase hold from ACK low	20			ns
t_{ADH}	Data hold from ACK high	0			ns
t_{ARL}	ACK high to $\underline{\text{REQ}}$ low	35			ns
t_{IODB}	I/O high to SBEN high			50	ns
t_{DZDB}	Data Bus disable from $\overline{\text{SBEN}}$ high			10	ns
t_{ARH}	ACK Low to REQ High	35			ns

NOTE 1: If the chip detects a parity error it must assert ATN at least t_{AA}before it de-asserts ACK.



8.2.6 INFORMATION TRANSFER PHASE INPUT (TARGET)

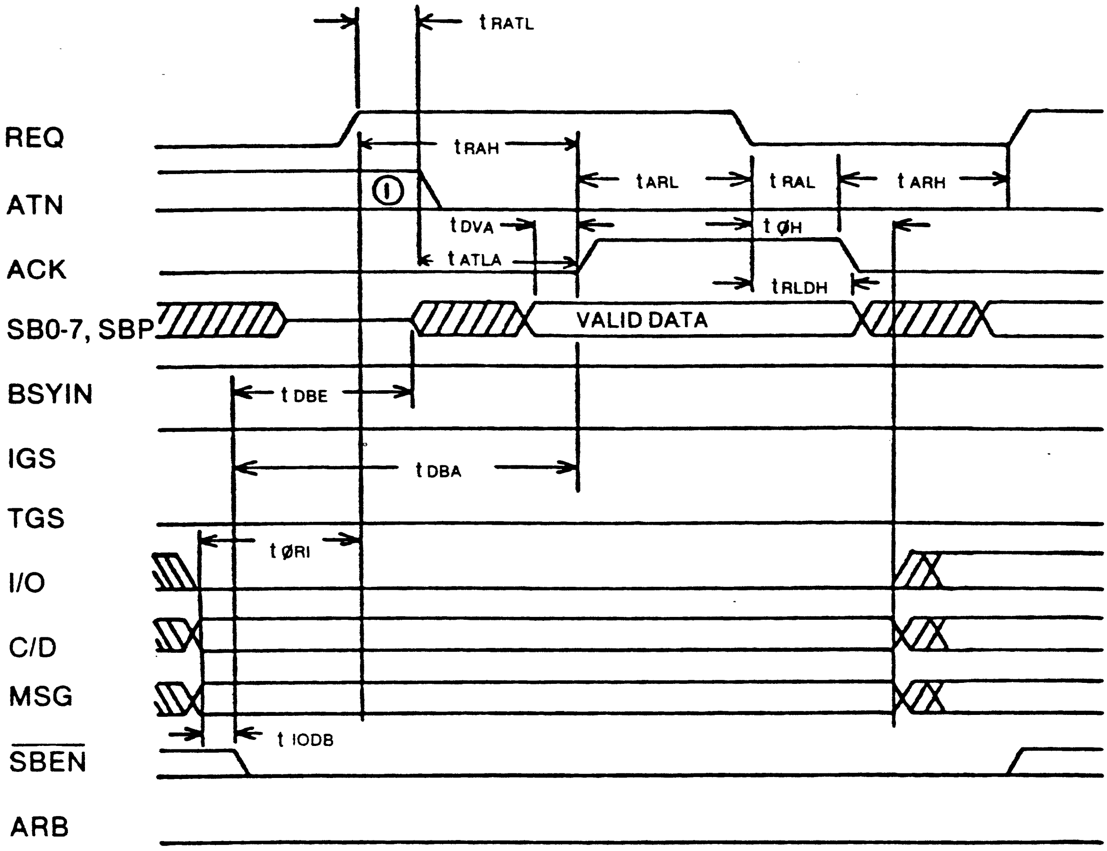
NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
$t_{S\emptyset}$	SELIN low to Phase Change	0			ns
$t_{\emptyset RO}$	Phase Change to REQ out	500			ns
t_{RAH}	REQ high to ACK high	35			ns
t_{ARL}	ACK high to REQ low	0			ns
t_{DVA}	Data Valid to ACK high	0			ns
t_{RAL}	REQ low to ACK low	35			ns
t_{ARH}	ACK low to REQ high	0			ns
t_{RLDH}	REQ low Data Hold	0			ns
$t_{\emptyset HA}$	Phase Hold from ACK low	0			ns
t_{DBIO}	SBEN high to I/O low	0			ns
t_{DZDB}	Data Bus disable to SBEN high	0			ns



8.2.7 INFORMATION TRANSFER PHASE OUTPUT (INITIATOR)

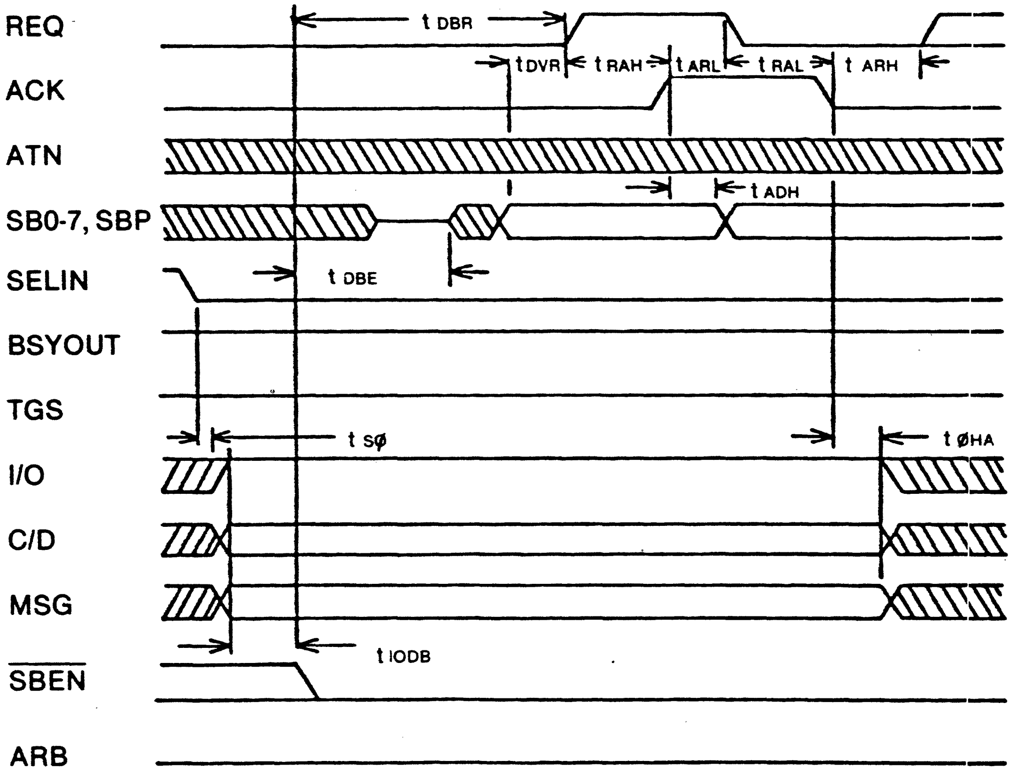
NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
$t_{\phi RI}$	Phase Valid to REQ high	100			ns
t_{RAH}	REQ high to ACK high	35			ns
t_{RAL}	REQ low to ACK low	0			ns
t_{DVA}	Data Valid to ACK high	100			ns
t_{RLDH}	REQ low Data hold	0			ns
$t_{\phi H}$	Phase hold from ACK low	20			ns
t_{ARL}	ACK high to REQ low	0			ns
t_{IODB}	I/O low to $\overline{\text{SBEN}}$ low	0			ns
t_{DBE}	$\overline{\text{SBEN}}$ low to Data Bus Enable	85			ns
t_{DBA}	$\overline{\text{SBEN}}$ low to ACK high	185			ns
t_{RATL}	REQ High to ATN low	0			ns
t_{ATLA}	ATN Low to ACK High	25			ns
t_{ARH}	ACK Low to REQ High	35			ns

NOTE 1: ATN is only de-asserted in this manner during the last byte of a Message Out Phase.



8.2.8 INFORMATION TRANSFER PHASE OUTPUT (TARGET)

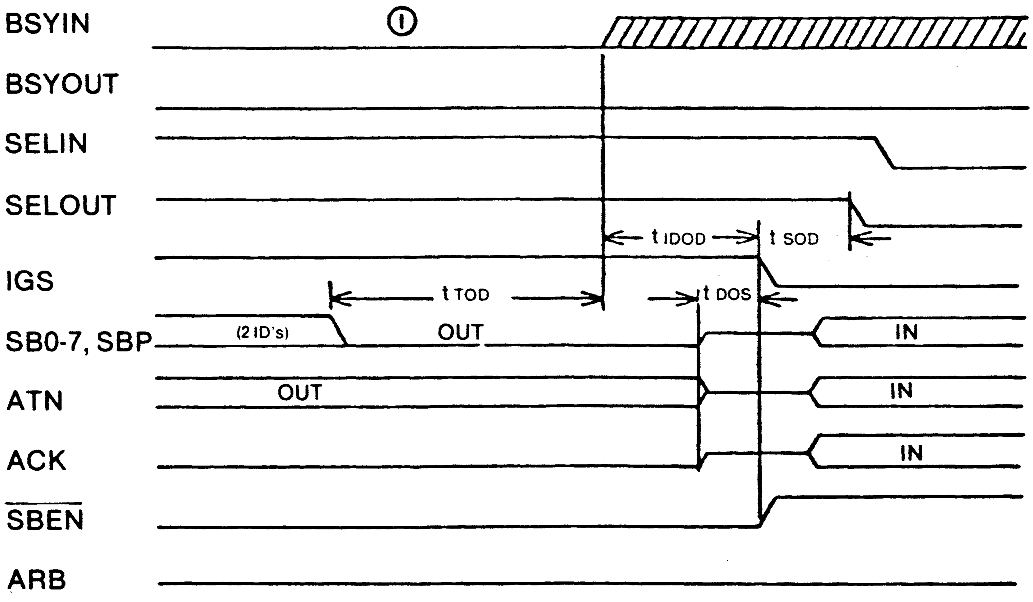
NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
$t_{S\emptyset}$	SELIN low to Phase Change	0			ns
t_{IODB}	I/O high to $\overline{\text{SBEN}}$ low	500			ns
t_{DBR}	$\overline{\text{SBEN}}$ low to REQ out	185			ns
t_{DVA}	Data Valid to REQ high	100			ns
t_{RAH}	REQ high to ACK high	0			ns
t_{ARL}	ACK high to REQ low	0			ns
t_{RAL}	REQ low to ACK low	0			ns
t_{ARH}	ACK low to REQ high	0			ns
$t_{\emptyset HA}$	Phase hold from ACK low	0			ns
t_{ADH}	Data hold from ACK low	0			ns
t_{DBE}	SBEN low to Data Bus Enabled	85			ns



8.2.9 BUS RELEASE FROM SELECTION (INITIATOR)

NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{TOD}	Bus Release Timeout Delay	100			µs
t_{IDOD}	IGS & $\overline{\text{SBEN}}$ Turn-off Delay	0			ns
t_{SOD}	SELOUT Turn-off Delay	0			ns
t_{DOS}	Driver Turn-off set-up to IGS & $\overline{\text{SBEN}}$ off	0			ns

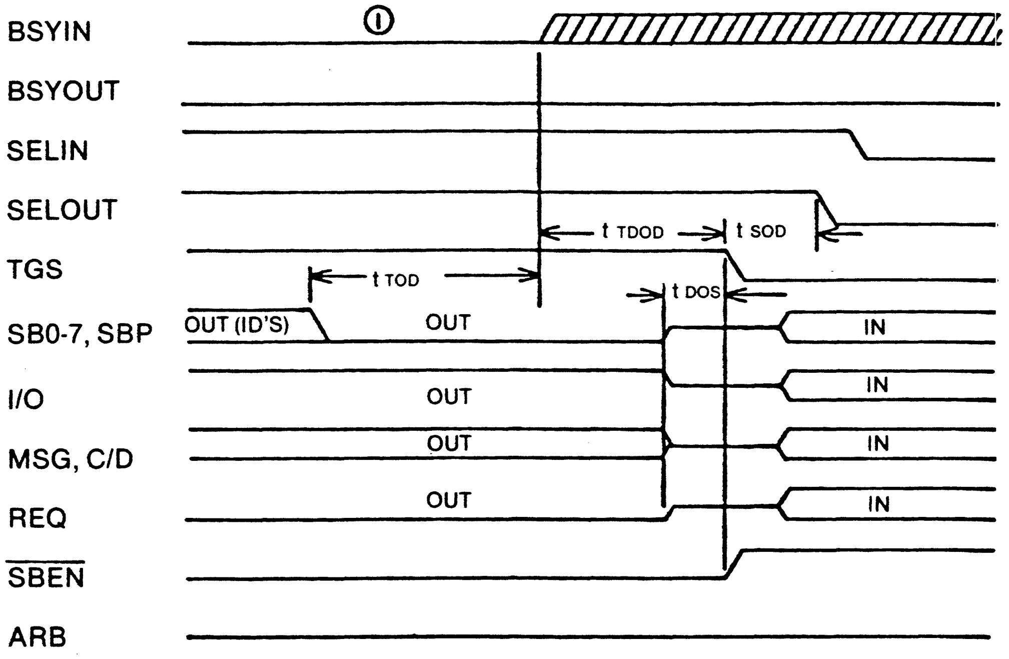
NOTE 1: If the chip detects BSYIN active by the end of the timeout delay, the bus release sequence shall be aborted since selection has been successful.



8.2.10 BUS RELEASE FROM RESELECTION (TARGET)

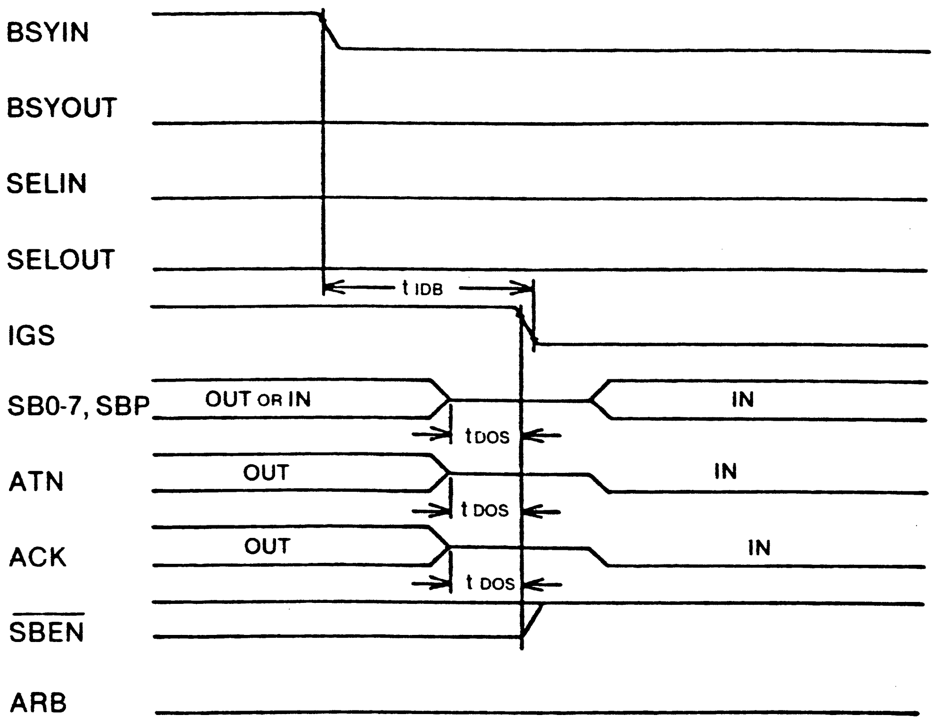
NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{TOD}	Bus Release Timeout Delay	100			μs
t_{DOD}	TGS & $\overline{\text{SBEN}}$ Turn-off Delay	0			ns
t_{SOD}	SELOUT Turn-off Delay	0			ns
t_{DOS}	Driver Turn-off set-up to TGS & $\overline{\text{SBEN}}$ off	0			ns

NOTE 1: If the chip detects BSYIN active by the end of the timeout delay, the bus release sequence shall be aborted since reselection has been successful.



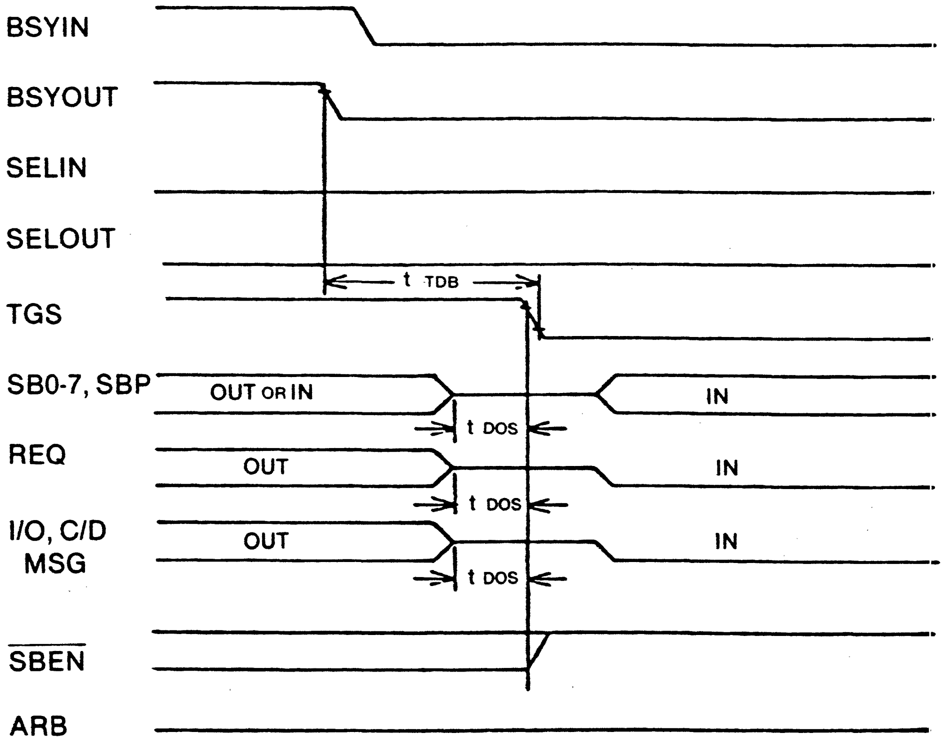
8.2.11 BUS RELEASE FROM INFORMATION PHASE (INITIATOR)

NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{IDB}	IGS & $\overline{\text{SBEN}}$ Turn-off Delay from BSYIN off			225	ns
t_{DOS}	Driver Turn-off set-up to IGS off	0			ns



8.2.12 BUS RELEASE FROM INFORMATION PHASE (TARGET)

NAME	DESCRIPTION	MIN	TYP	MAX	UNITS
t_{TDB}	TGS & $\overline{\text{SBEN}}$ Turn-off from BSYOUT off			225	ns
t_{DOS}	Driver Turn-off set-up to TGS off	0			ns



NCR 5385E/5386 SCSI

Protocol Controller

User's Guide

NCR

Microelectronics Division, Colorado Springs

This document contains the latest information available at the time of publication. However, NCR reserves the right to modify the contents of this material at any time. Also, all features, functions and operations described herein may not be marketed by NCR in all parts of the world. Therefore, before using this document, consult your NCR representative or NCR office for the information that is applicable and current.

TABLE OF CONTENTS

APPENDICES

APPENDIX PAGE

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

SECTION 1
GENERAL INFORMATION

The NCR SCSI Protocol Controller is capable of operating as either an Initiator or a Target, and can therefore be used in host adapter and control unit designs. The purpose of this manual is to assist the user in the design of these SCSI bus devices.

Sections 2 through 5 discuss the software required to control the device. Sample flowcharts and step-by-step walk-through's demonstrate the required routines for both the Initiator and Target roles. Additionally, the interrupt service routines presented in Section 5 cover all possible interrupting conditions for the Connected as Initiator, Connected as Target and Disconnected states.

Section 6 describes the inteface required between the NCR 5385E/86 and SCSI bus for both Single-Ended and Differential-Pair operation, and provides sample schematics for each.

This design manual is not an SCSI specification and assumes some prior knowledge of the SCSI proposed standard. Copies of the proposed standard may be obtained, with a pre-payment of $20, from:

X3 Secretariat, Computer and Business Equipment
Manufacturers Association
311 First Street, NW, Suite 500
Washington, D.C. 20001

Please include a self-addressed mailing label.

1.1 ADDITIONAL DOCUMENTATION

Other documents which may be useful are:

- NCR 5385 SCSI Protocol Controller Data Sheet (MC-704)
- NCR 5380 SCSI Interface Chip Design Manual
- SCSI Engineering Notebook

These documents may be obtained from your local NCR Microelectronics sales representative or from:

NCR Microelectronics
Logic Products Marketing
1635 Aeroplaza Drive
Colorado Springs, CO 80916
(800) 525-2252 or
(303) 596-5612

SECTION 2
INITIALIZATION

The three steps typically performed after the NCR SCSI Protocol Controller is reset are presented below. It is assumed that no errors occur.

1. **Loop on reading the Diagnostic Status Register until the Self-Diagnostic Complete bit is on. (This should occur within 350 clock cycles after the reset pulse goes inactive or after the write pulse for a Chip Reset command.) Never attempt to read the Diagnostic Status Register while Reset is active, as the data bus is in an unknown state and the Self-Diagnostic Complete bit may appear asserted.**

2. **Check the Diagnostic Command Status and Self-Diagnostic Status bits of the Diagnostic Status Register for all zeros (no errors).**

   (At this point, the user may wish to perform other tests such as loading an invalid command, performing the Diagnostic Data Turnaround, etc.)

3. **Load the Control Register with the desired information (Parity Enable, Reselect Enable, Select Enable).**

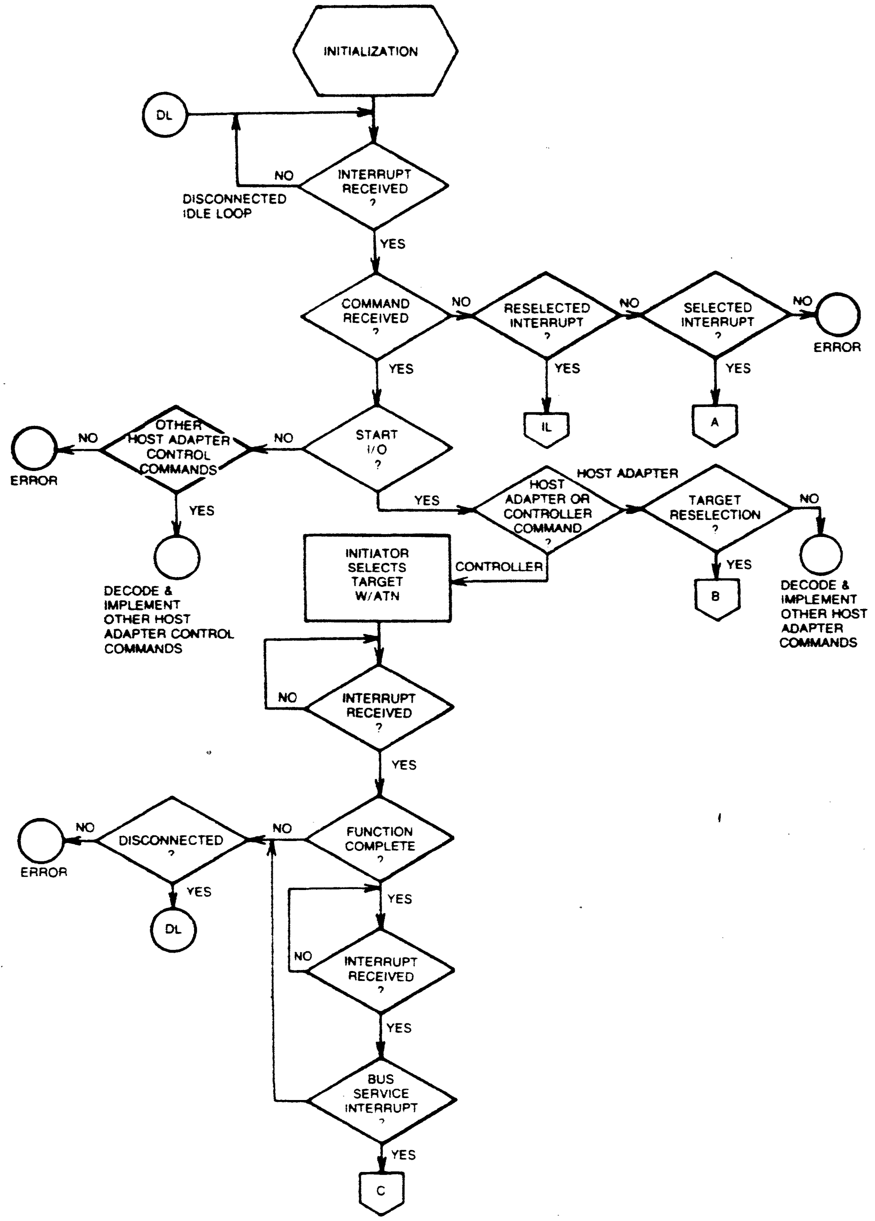
It should be noted that immediately following step 3, the chip is in the Disconnected state. If the Reselect Enable or Select Enable bits are enabled, an interrupt can occur prior to issuing any commands to the NCR SCSI Protocol Controller. A Reselection or Selection interrupt may also occur even after issuing a Select or Reselect command to the chip. In this case, a higher priority device wins arbitration and selects or reselects the chip, generating a Selected or Reselected interrupt rather than a Function Complete Interrupt. The user must wait until the chip is in the Disconnected state before reissuing the Select or Reselect command.

For more information concerning Initialization, please refer to Section 7 of the NCR 5385 SCSI Protocol Controller Data Sheet (Publication # MC-704), and to Section 7, Device Note 5 of this document.

SECTION 3
INITIATOR ROLE

The Initiator Role is normally associated with the host adapter, but may also be assumed by a tape controller performing a disk back-up operation. After selecting a Target device, the Initiator must respond to the Information Transfer Phases controlled by the Target. Please refer to the latest revision of the draft proposed SCSI standard for a complete description of the Initiator Role.

The following partial flowchart illustrates the role of the SCSI bus Initiator. (Note that the Target portion of this flowchart appears in Section 4, "Target Role," and that the flow chart is presented in its entirety in Appendix C.)



3.1 INITIATOR ROLE FLOWCHART

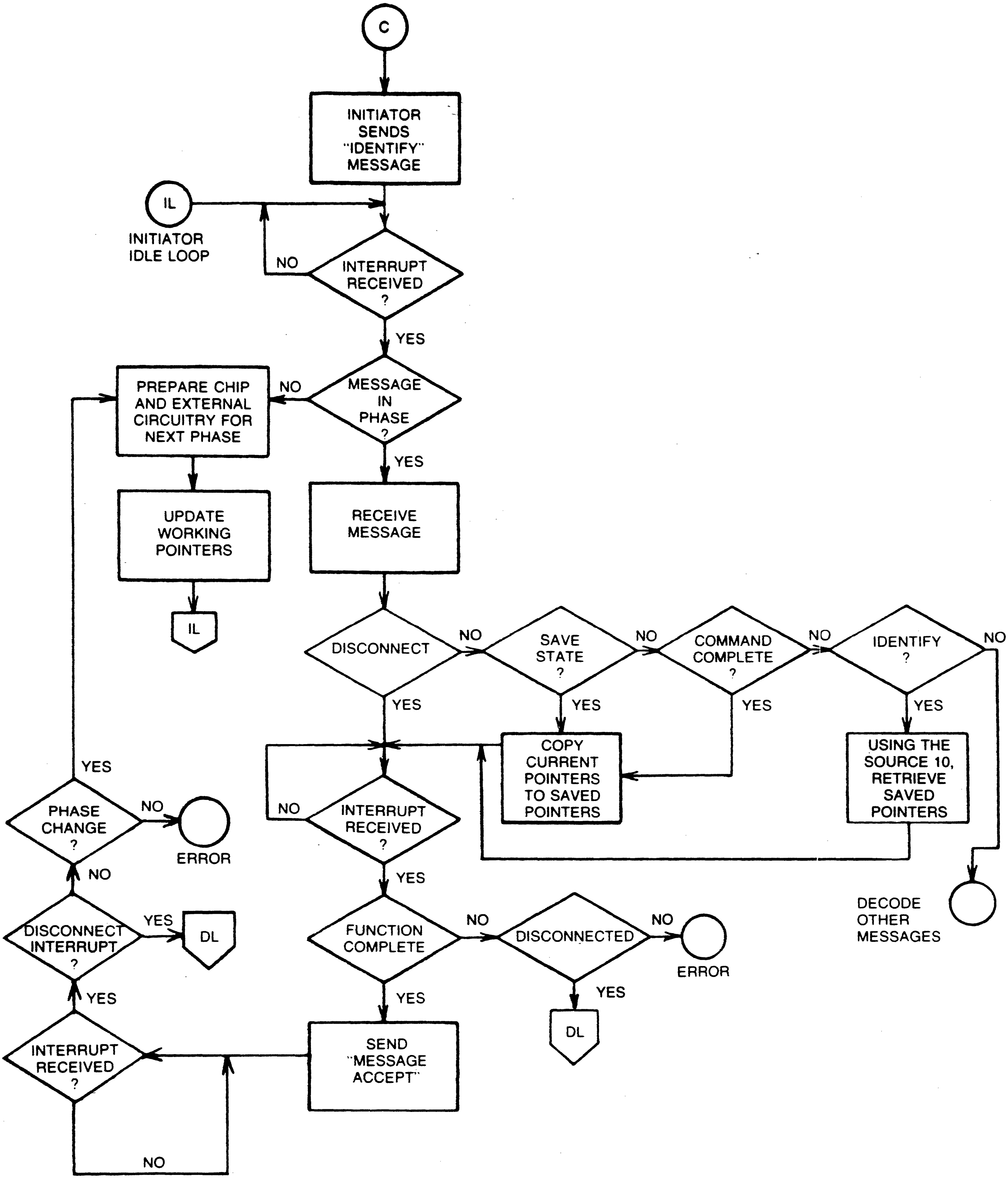


FIGURE 3.1 INITIATOR ROLE (Concluded)

3.1 INITIATOR ROLE WALKTHROUGH

This sample walkthough outlines the steps typically required to perform a complete I/O function as an Initiator. It is assumed that both the Initiator and Target can handle messages and are able to disconnect and reconnect during the function. To simplify this example, it is further assumed that no errors or exceptions occur during the entire operation.

Note that the steps are grouped under headings that describe the function each group accomplishes.

INITIATOR SELECTS TARGET

1. Load the Destination ID Register with the Target's ID.
2. Load the Transfer Counter to program the selection timeout. Write to each of the three 8-bit registers.
3. Load the Command Register with SELECT W/ATN.
4. Wait for an interrupt.
5. Read the Auxiliary Status Register.
6. Read the Interrupt Register.
7. Check for a Function Complete interrupt. (This indicates the SELECT W/ATN was successful.)

(The NCR 5385 SCSI Protocol Controller is now in the Connected as Initiator state.)

INITIATOR SENDS "IDENTIFY" MESSAGE

8. Wait for an interrupt.
9. Read the Auxiliary Status Register.
10. Read the Interrupt Register.
11. Check the interrupt. A Bus Service interrupt should have occurred, indicating that the Target has initiated an information transfer.
12. Check the I/O, C/D, and MSG bits read from the Auxiliary Status Register. The Target should be requesting a Message Out phase to receive the "Identify" message.
13. Load the Command Register with a Transfer Info command. Since the "Identify" message is a single byte, program the Single Byte Transfer bit ON ("1") and the DMA Mode bit OFF ("0").
14. Read the Auxiliary Status Register.
15. Check the Data Register Full bit.
16. Repeat steps (14) and (15) until Data Register Full is OFF ("0").
17. Write the "Identify" message into the Data Register.
18. Wait for an interrupt.
19. Read the Auxiliary Status Register.
20. Read the Interrupt Register.
21. Check the interrupt. Another Bus Service interrupt should have occurred, indicating that the Target has again changed the bus phase.

INITIATOR RECEIVES "DISCONNECT" MESSAGE

(Note that the Target is not required to send a "Disconnect" message and disconnect at this point. It may request the command before disconnecting, or not disconnect at all. If the Target does not issue the message, proceed to step 65.

22. Check the I/O, C/D, and MSG bits read from the Auxiliary Status Register. The Target should be requesting a Message In phase.

23. Load the Command Register with a Transfer Info command. The Single Byte Transfer bit should be ON ("1"), and the DMA Mode bit OFF ("0").

24. Read the Auxiliary Status Register.

25. Check the Data Register Full bit.

26. Repeat 24 and 25 until Data Register Full is on.

27. Read the Data Register.

28. Check the message. The Target should have sent a "Disconnect" message, indicating that it will reconnect later to complete the I/O function.

29. Wait for an interrupt. (Note that a Function Complete interrupt may occur at any time after the Transfer Info command is loaded (step 23). To provide for its occurrence during steps 24 through 28, it is suggested that the user set an interrupt flag in the interrupt service routine, mask any other interrupts, and then complete steps 24 through 28.) If a Disconnect interrupt occurs, it must be serviced immediately.

30. Read the Auxiliary Status Register.

31. Read the Interrupt Register.

32. Check the interrupt. A Function Complete should have occurred, indicating that the last byte of the message has been received. ACK is left active so that ATN may be asserted if the message needs to be rejected.

33. Load the Command Register with a Message Accepted command.

INITIATOR AWAITS DISCONNECTION

34. Wait for an interrupt.

35. Read the Auxiliary Status Register.

36. Read the Interrupt Register.

37. Check the interrupt. The Target should have disconnected, causing a Disconnect interrupt.

(The NCR 5385 SCSI Protocol Controller is now in the Disconnected state and may start or handle I/O functions for any other logical unit. For this I/O function to continue, the user must wait until the chip is reselected by the Target while in the disconnected state. Step 38 continues the flow.)

INITIATOR IS RESELECTED

38. Wait for an interrupt.

39. Read Auxiliary Status Register.

40. Read Interrupt Register.

41. Check the interrupt. Assuming the Target has reselected the NCR SCSI Protocol Controller to continue the function, a Reselect Interrupt should have occurred.

(The user is now in the Connected as Initiator state.)

INITIATOR RECEIVES "IDENTIFY" MESSAGE

42. Wait for an interrupt.

43. Read the Auxiliary Status Register.

44. Read the Interrupt Register.

45. Check the interrupt. A Bus Service interrupt should have occurred, indicating that the Target has activated REQ for an information transfer.

46. Check the I/O, C/D, and MSG bits read from the Auxiliary Status Register. The Target should be requesting a Message In phase to identify the I/O.

47. Load the Command Register with a Transfer Info command (Single Byte Transfer = 1, DMA Mode = 0.)

48. Read the Auxiliary Status Register.

49. Check the Data Register Full bit.

50. Repeat 48 and 49 until Data Register Full is ON ("1").

51. Read the Data Register.

52. Check the message. The Target should have sent an "Identify" message which contains the logical unit number for the I/O.

53. Read the Source ID Register.

54. Check the contents of the Source ID Register to determine which device did the reselection.

55. Having identified the device and logical unit number, retrieve the command, data and status pointers for this I/O, and store them in a working pointer area outside the NCR 5385 chip.

56. Wait for an interrupt. (Note that a Function Complete interrupt may occur at any time after the Transfer Info command (step 47). To provide for its occurrence during steps 48 through 55, it is suggested that the user set an interrupt flag in the interrupt service routine, mask any further interrupts, and then complete steps 48 through 55.) If a Disconnect interrupt occurs, it must be serviced immediately.

57. Read the Auxiliary Status Register.

58. Read the Interrupt Register.

59. Check the interrupt. A Function Complete should have occurred, indicating that the last byte of the message has been received. ACK is left active so that ATN may be asserted if the message needs to be rejected.

60. Load the Command Register with a Message Accepted command.

INITIATOR TRANSFERS COMMAND, DATA, OR STATUS

61. Wait for an interrupt.

62. Read the Auxiliary Status Register.

63. Read the Interrupt Register.

64. Check the interrupt. A Bus Service interrupt should have occurred, indicating that the Target has initiated another information phase.

65. Check the I/O, C/D, and MSG bits read from the Auxiliary Status Register. The Target should be requesting a Command, Data, or Status phase.

66. Prepare circuitry external to the chip for the

requested transfer by using the appropriate working pointer.

67. Load the Transfer Counter for the maximum number of bytes to be transferred. Write to each of the three 8-bit registers. (This step is omitted for a Single Byte Transfer.)

68. Load the Command Register with a Transfer Info command. Normally, for command or data transfers, Single Byte Transfer = 0 and DMA Mode = 1. (For status, these bits might be 1 and 0, respectively.)

69. If Single Byte Transfer = 0, go to step 74.

70. Read the Auxiliary Status Register.

71. Check the Data Register Full bit.

72. Repeat 70 and 71 until the Data Register Full bit if OFF ("0").

73. Read the Data Register (with status byte).

74. Wait for an interrupt.

75. Read the Auxiliary Status Register.

76. Read the Interrupt Register.

77. Check the interrupt. A Bus Service interrupt should have occurred, indicating that the Target has initiated a different information phase.

INITIATOR UPDATES WORKING POINTER FOR LAST TRANSFER

78. If the last transfer was a single byte, go to step 82.

79. Check the Transfer Counter Zero bit in the Auxiliary Status Register.

80. If Transfer Counter = 0, go to step 82.

81. Read the Transfer Counter.

82. Update the *working* pointer for the last information phase. Note that the stored pointer is not updated at this time. Stored pointers are updated ONLY when a "Save State" or "Command Complete" message is received.

INITIATOR CHECKS NEW PHASE TYPE

83. Check the I/O, C/D, and MSG bits read from the Auxiliary Status Register.

84. If the Target is requesting a Command, Data or Status phase, go back to step 66.

INITIATOR RECEIVES MESSAGE

85. If the Target is requesting a Message In phase, load the Command Register with a Transfer Info command (Single Byte Transfer = 1, DMA Mode = 0).

86. Read the Auxiliary Status Register.

87. Check the Data Register Full bit.

88. Repeat steps 86 and 87 until the Data Register Full is set.

89. Read the message in the Data Register.

90. If the message is "Command Complete," go to step 113.

91. If the message is "Disconnect," go to step 103.

INITIATOR HANDLES "SAVE STATE" MESSAGE

92. In normal operation, the message referred to in step 89 should be "Save State." In this case, the user saves the state of the working pointers by moving them to the stored pointer area.

93. Wait for an interrupt. (Note that a Function Complete interrupt may occur at any time after step 85. To provide for its occurrence during steps 86 through 92, it is suggested that user set an interrupt flag in the interrupt service routine, mask any further interrupts, and proceed to complete steps 86 through 92.) If a Disconnect interrupt occurs, it must be serviced immediately.

94. Read the Auxiliary Status Register.

95. Read the Interrupt Register.

96. Check the interrupt. A Function Complete should have occurred, indicating that the last byte of the message was received. ACK is left active so that ATN may be asserted if the message needs to be rejected.

97. Load the Command Register with a Message Accepted command.

98. Wait for an interrupt.

99. Read the Auxiliary Status Register.

100. Read the Interrupt Register.

101. Check the interrupt. A Bus Service interrupt should have occurred, indicating that the Target has initiated another information phase.

102. Go to step 83.

INITIATOR HANDLES "DISCONNECT" MESSAGE

103. Wait for an interrupt. (Note that a Function Complete interrupt may occur at any time after step 85. To provide for its occurrence during steps 86 through 102, it is suggested that the user set an interrupt flag in the interrupt service routine, mask any further interrupts, and proceed to complete steps 86 through 102.) If a Disconnect interrupt occurs, it must be serviced immediately.

104. Read the Auxiliary Status Register.

105. Read the Interrupt Register.

106. Check the interrupt. A Function Complete should have occurred, indicating that the last byte of the message has been received. ACK is left active so that ATN may be asserted if the message needs to be rejected.

107. Load the Command Register with a Message Accepted command.

INITIATOR AWAITS DISCONNECTION

108. Wait for an interrupt.

109. Read the Auxiliary Status Register.

110. Read the Interrupt Register.

111. Check the interrupt. After sending the "Disconnect" message, the Target should have disconnected, resulting in a Disconnected interrupt.

112. Go to step 38. (The note prior to 38 applies.)

INITIATOR HANDLES "COMMAND COMPLETE" MESSAGE

113. Save the state of the working pointers by moving them to the stored pointer area.

114. Wait for an interrupt. (Note that a Function Complete interrupt may occur at any time after step 85. To provide for its occurrence during steps 86 through 113, it is suggested that the user set an interrupt flag in the interrupt service routine, mask any further interrupts, and complete steps 86 through 113.) If a Disconnect interrupt occurs, it must be serviced immediately.

115. Read the Auxiliary Status Register.

116. Read the Interrupt Register.

117. Check the interrupt. A Function Complete should have occurred, indicating that the last byte of the message has been received. ACK is left active so that ATN may be asserted if the message needs to be rejected.

118. Load the Command Register with a Message Accepted command.

INITIATOR AWAITS DISCONNECTION

119. Wait for an interrupt.

120. Read the Auxiliary Status Register.

121. Read the Interrupt Register.

122. Check the interrupt. After sending the "Command Complete" message, the Target should have disconnected, resulting in a Disconnected interrupt.

(The I/O function is now complete. The user is back in the Disconnected state.)

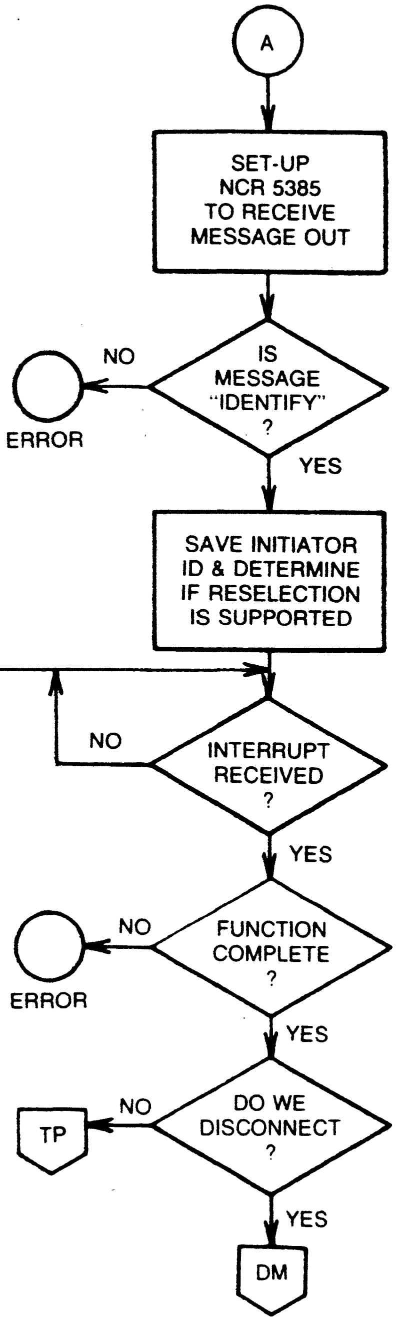
3.2 NOTES

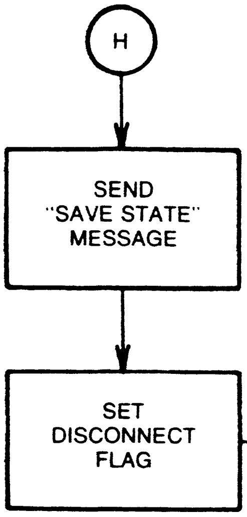
1. Steps 14 through 16, and 70 through 72 can be omitted if the microprocessor guarantees that one full clock cycle elapses between loading the Command Register and loading the Data Register. The act of loading an Interrupting command resets the Data Register Full Status Bit in the Auxiliary Status Register. Therefore, when a command is issued that requires data to be put into the Data Register, data may not be loaded until the Data Register Full Status Bit is allowed to reset.
2. If a Disconnect Command is issued when connected as an Initiator, the Target is left hanging on the bus. A bus reset may be required to free the bus.

SECTION 4
TARGET ROLE

The Target Role, though normally performed by a peripheral, may also be assumed by the host adapter during host-to-host communications. When selected, the Target controls the bus activity by driving the C/D, I/O, and MSG signals. Please refer to the latest draft proposed SCSI standard for a complete description of the Target Role.

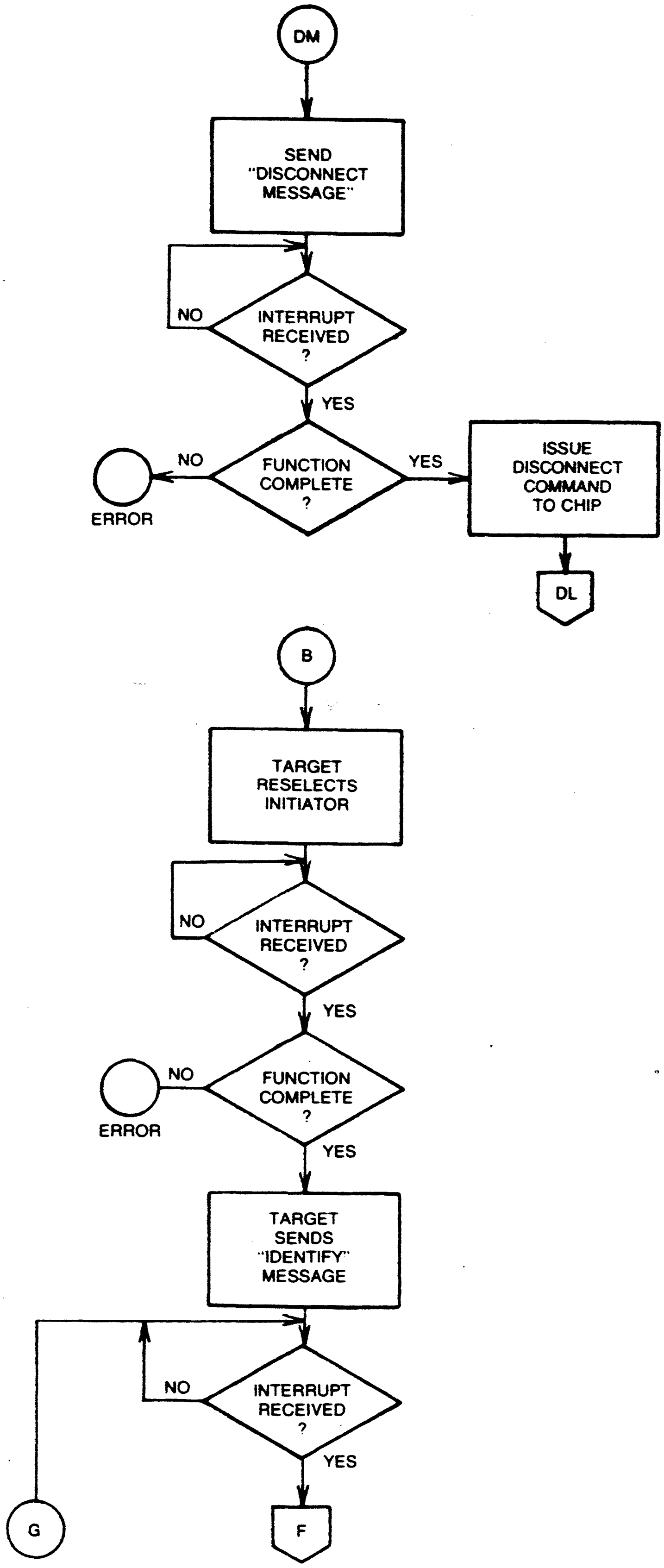
The following partial flowchart illustrates the role of the Target on the SCSI bus. The flowchart in Fig. 4.1 is a continuation of Fig. 3.1 and resumes after the target has been selected. The flowchart of the Initiator and Target Roles is shown in its entirety in Appendix C.



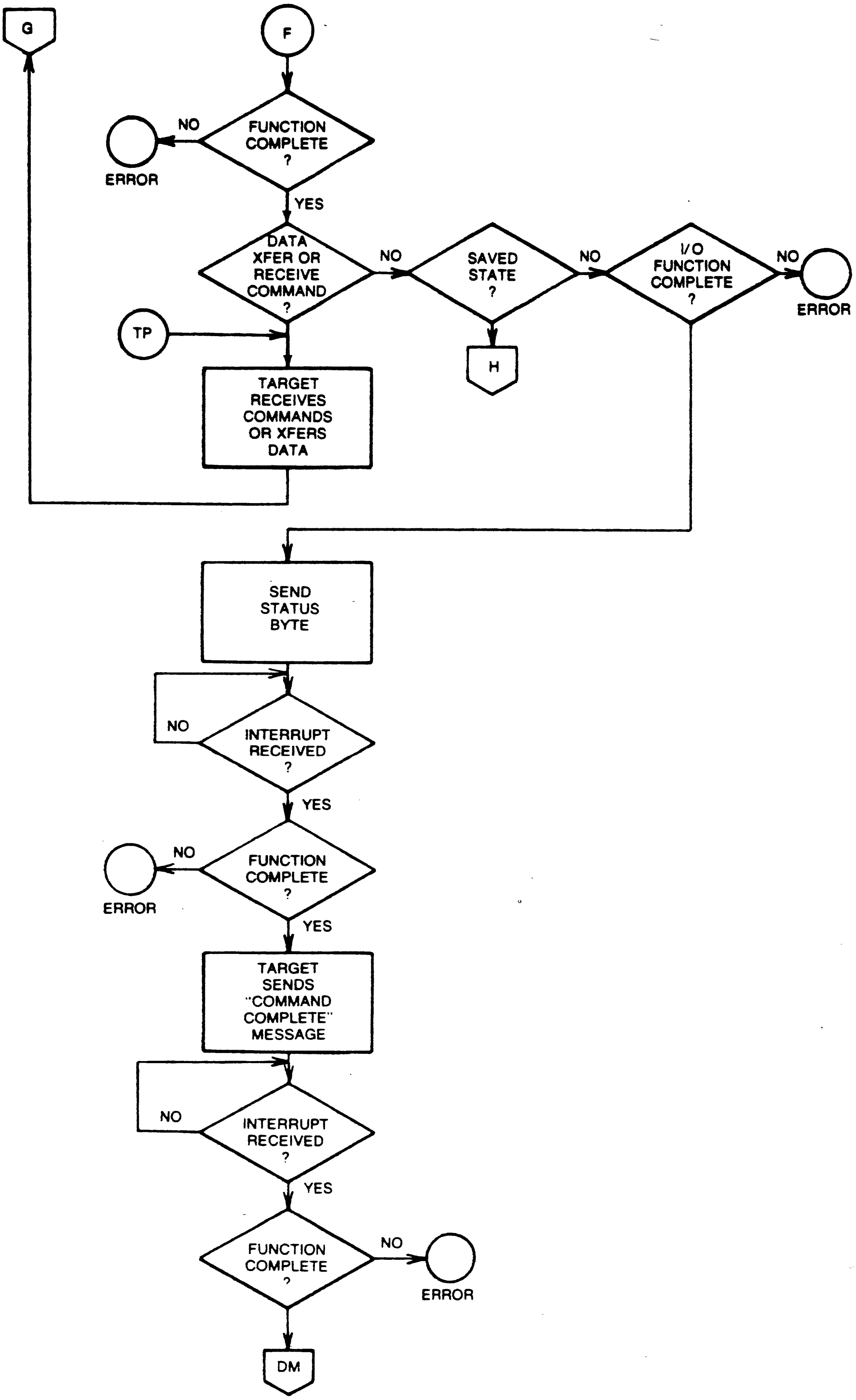


4.1 TARGET ROLE FLOWCHART

4.1 TARGET ROLE FLOWCHART (Continued)



4.1 TARGET ROLE FLOWCHART (Continued)



4.1 TARGET ROLE WALKTHROUGH

This sample walkthrough outlines the steps typically required to perform a complete I/O function as a Target. It is assumed that both the Target and Initiator can handle messages and are able to disconnect. It is also assumed that no errors or exceptions occur during the entire operation. The sequence of steps begins after the chip has been selected as a Target.

Note that the steps are grouped under headings that describe the function each group accomplishes.

TARGET IS SELECTED

1. Wait for an interrupt.
2. Read the Auxiliary Status Register.
3. Read the Interrupt Register.
4. Check the interrupt. Selected and Bus Service interrupts should have occurred, indicating that the chip has been selected as a Target, and the Initiator has asserted the ATN signal, respectively.

(The user is now in the Connected as Target state.)

TARGET RECEIVES "IDENTIFY MESSAGE"

5. Load the Command Register with a Receive Message Out command (Single Byte Transfer = 1, DMA Mode = 0).
6. Read the Auxiliary Status Register.
7. Check the Data Register Full bit.
8. Repeat steps 6 and 7 until Data Register Full bit is ON ("1").
9. Read the Data Register.
10. Check the message. The Initiator should have sent an "Identify" message which indicates whether he can disconnect. The message also contains the logical unit number for the I/O.
11. Read the Source ID Register.
12. Check the ID Valid bit. If the Initiator has the ability to disconnect, it will be ON. (The user now has the Initiator ID and the logical unit number, which uniquely defines an I/O. The user may record this information and disconnect.)
13. Wait for an interrupt. (Note that a Function Complete may occur at any time after step 9. To provide for its occurrence during steps 10 through 12, it is suggested that the user set an interrupt flag in the interrupt service routine, mask any further interrupts, and complete steps 10 through 12.) If a Disconnect interrupt occurs, it must be serviced immediately.
14. Read the Auxiliary Status Register.
15. Read the Interrupt Register.
16. Check the interrupt. A Function Complete should have occurred, indicating that the ID message has completed.

(The user is back in the Connected as Target state. If it is desired not to disconnect at this point, go to step 43.)

TARGET SENDS DISCONNECT MESSAGE AND DISCONNECTS

17. Load the Command Register with a Send

Message In command, (Single Byte Transfer = 1, DMA Mode = 0).

18. Read the Auxiliary Status Register.

19. Check the Data Register Full bit.

20. Repeat steps 18 and 19 until Data Register Full bit is OFF ("0").

21. Write the "Disconnect" message into the Data Register.

22. Wait for an interrupt.

23. Read the Auxiliary Status Register.

24. Read the Interrupt Register.

25. Check the interrupt. A Function Complete interrupt should have occurred, indicating the message was sent successfully.

26. Load the Command Register with a Disconnect command.

(DISCONNECT immediately breaks the connection, and the user is in the Disconnected state. When ready to continue the I/O operation, go to step 27.)

TARGET RESELECTS INITIATOR

27. Load the Destination ID Register with the Initiator's ID.

28. Load the Transfer Counter to program in the reselection timeout. Write to each of the three 8-bit registers.

29. Load the Command Register with a Reselect command.

30. Wait for an interrupt.

31. Read the Auxiliary Status Register.

32. Read the Interrupt Register.

33. Check the interrupt. A Function Complete interrupt should have occurred, indicating that the Reselect was successful.

(The user is now in the Connected as Target state. Note that SCSI protocol requires that the "Identify" Message be sent immediately following the reselection. Therefore, the user should continue with steps 34 through 42.)

TARGET SENDS "IDENTIFY" MESSAGE

34. Load the Command Register with a Send Message In command, (Single Byte Transfer = 1, DMA Mode = 0).

35. Read the Auxiliary Status Register.

36. Check the Data Register Full bit.

37. Repeat steps 35 and 36 until Data Register Full bit is OFF ("0").

38. Write the "Identify" message into the Data Register.

39. Wait for an interrupt.

40. Read the Auxiliary Status Register.

41. Read the Interrupt Register.

42. Check the interrupt. A Function Complete interrupt should have occurred, indicating that the message was sent successfully.

TARGET RECEIVES COMMAND OR TRANSFERS DATA

43. Load the Transfer Counter for a command or data transfer.

44. Load the Command Register with a Receive Command, Receive Data, or Send Data command (Single Byte Transfer = 0, DMA Mode = 1).

45. Wait for an interrupt.

46. Read the Auxiliary Status Register.

47. Read the Interrupt Register.

48. Check the interrupt. A Function Complete interrupt should have occurred, indicating that the transfer was successful.

49. To do a data transfer, go back to step 43.

50. If the I/O function is complete, go to step 62 and continue through step 80. Otherwise, proceed to steps 51 through 61 with the intent of reconnecting later.

TARGET SENDS "SAVE STATE" AND "DISCONNECT" MESSAGES, AND DISCONNECTS

51. Load the Command Register with a Send Message In command, (Single Byte Transfer = 1, DMA Mode = 0).

52. Read the Auxiliary Status Register.

53. Check the Data Register Full bit.

54. Repeat steps 52 and 53 until Data Register Full is OFF ("0").

55. Write the "Save State" message into the Data Register.

56. Wait for an interrupt.

57. Read the Auxiliary Status Register.

58. Read the Interrupt Register.

59. Check the interrupt. A Function Complete interrupt should have occurred, indicating the message was sent successfully.

60. Repeat steps 51 through 59 for a "Disconnect" message.

61. Load the Command Register with a Disconnect command.

(A Disconnect command immediately breaks the connection, and the NCR SCSI Protocol Controller is in the Disconnected state. When ready to continue I/O operation, go to step 27.)

TARGET SENDS STATUS BYTE

62. Load the Command Register with a Send Status command, (Single Byte Transfer = 1 and DMA Mode = 0).

63. Read the Auxiliary Status Register.

64. Check the Data Register Full bit.

65. Repeat steps 63 and 64 until Data Register Full bit is OFF ("0").

66. Write the status byte into the Data Register.

67. Wait for an interrupt.

68. Read the Auxiliary Status Register.

69. Read the Interrupt Register.

70. Check the interrupt. A Function Complete interrupt should have occurred, indicating that the status byte was sent successfully.

TARGET SENDS COMMAND COMPLETE MESSAGE

71. Load the Command Register with a Send Message in command, (Single Byte Transfer = 1, DMA Mode = 0).

72. Read the Auxiliary Status Register.

73. Check the Data Register Full bit.

74. Repeat steps 72 and 73 until Data Register Full bit is OFF ("0").

75. Write the "Command Complete" message into the Data Register.

76. Wait for an interrupt.

77. Read the Auxiliary Status Register.

78. Read the Interrupt Register.

79. Check the interrupt. A Function Complete interrupt should have occurred, indicating that the message was sent successfully.

80. Load the Command Register with a Disconnect command.

(The I/O function is now complete. The user is back in the disconnected state.)

4.2 NOTES

Steps 18 through 20, 35 through 37, 52 through 54, 63 through 65, and 72 through 74 can be omitted if the microprocessor guarantees that one full clock cycle elapses between loading the Command Register and loading the Data Register.

The act of loading an interrupting command resets the Data Register Full status bit in the Auxiliary Status Register. Therefore, when a command is issued that requires data to be put into the Data Register, the data may not be loaded until the Data Register Full bit is allowed to reset.

SECTION 5
INTERRUPT SERVICE ROUTINES (ISR)

This section defines all possible interrupt conditions, and provides suggested responses.

5.1 GENERAL

When interrupted by the NCR SCSI Protocol Controller, the users should read and save the Auxiliary Status Register and the Interrupt Register. (Note that this is not a requirement for servicing the interrupt for the Diagnostic command.)

Figures 5.1 and 5.2 depict the Auxiliary Status and Interrupt Registers.

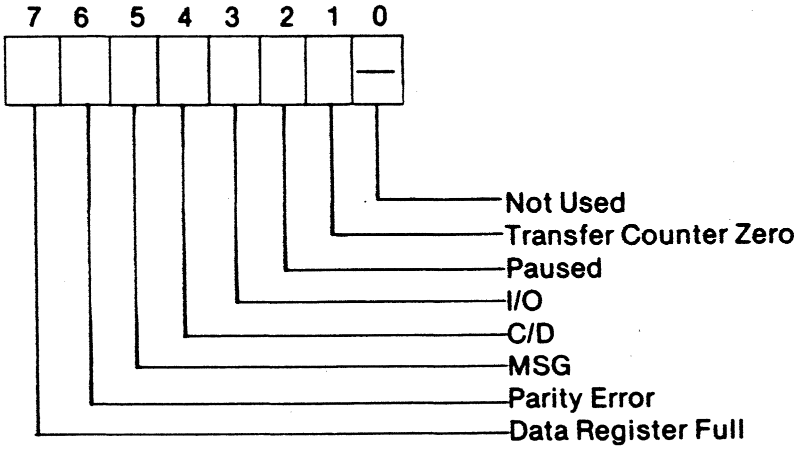


Figure 5.1 Auxiliary Status Register

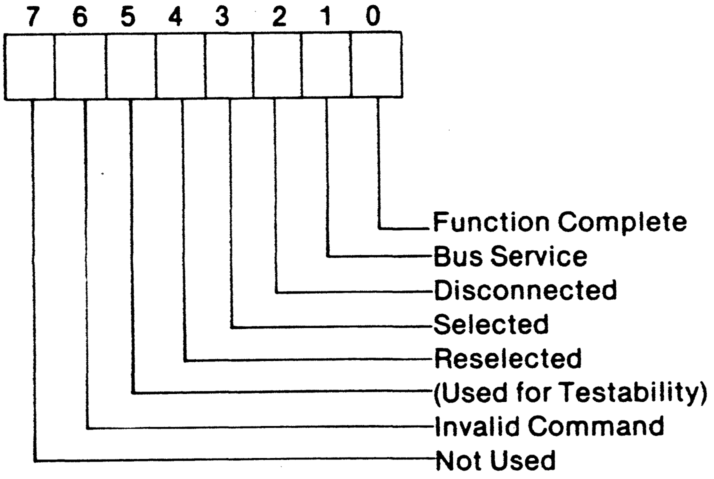


Figure 5.2 Interrupt Register

The act of reading the Interrupt Register disables the interrupt signal (INT), resets the Interrupt Register, and resets an internal flag that indicates an Interrupting command is in progress. For this reason, if a bus event which also causes an interrupt occurs at the same time an Interrupting command is loaded into the Command Register, the command in the Command Register is not executed. Rather, the bus event is honored.

The user must always know in which state the device is operating: Disconnected, Connected as Initiator, or Connected as Target. (This determines which commands are valid for that state and what the interrupt service routine should contain.) Interrupt information is summarized in Table 5.1. The paragraphs following the table describe the suggested interrupt service routines for each state.

5.2 USER DISCONNECTED ISR

In the Disconnected state, the user is either currently logically disconnected from the SCSI bus, or was disconnected at the time the last command to the chip was issued. Valid commands that may have been issued in this state are:

Interrupting	SELECT W/ATN
	SELECT W/O ATN
	RESELECT

Immediate	PAUSE CHIP RESET

The seven interrupts that may occur in the Disconnected state are numbered and described below.

1. Interrupt Register (7-0): 0000 1000

   Auxiliary Status Register (7-0): xxxx xxxx

   Reason for Interrupt: The user has been selected as the Target. ATN was not enabled by the Initiator; therefore, the Target is not capable of using messages (with the exception of "Command Complete"), and cannot disconnect prior to completing the function.

   Suggested Response: Set-up the Transfer Counter, issue a Receive command, and proceed with the function through status phase and "Command Complete" message.

2. Interrupt Register (7-0): 0000 1010

   Auxiliary Status Register (7-0): xxxx xxxx

   Reason For Interrupt: The user has been selected as the Target. ATN was asserted by the Initiator; therefore, the Target is capable of using messages.

   Suggested Response: Read the Source ID Register. If the ID Valid bit is not enabled, the Initiator cannot be disconnected until the function is completed. Set the Transfer Counter and issue a Receive Message Out command. The Identify message can be used to determine whether the Initiator can disconnect.

3. Interrupt Register (7-0): 0001 0000

   Auxiliary Status Register (7-0): xxxx xxxx

   Reason for Interrupt: The user has been reselected as the Initiator. It is implied that the Target has disconnect and control message capability.

   Suggested Response: Wait for another interrupt (either Bus Service or Disconnect).

Normally, a Bus Service interrupt is generated for a Message In phase, allowing an "Identify" message to be sent.

4. Interrupt Register (7-0): 0000 0001

   Auxiliary Status Register (7-0): xxxx xxxx

   Reason for Interrupt: The user command has been successfully completed. The Select commands (SELECT W/ATN, SELECT W/O ATN) imply that the user is connected to a Target and is acting as an Initiator, while the Reselect command implies reconnection to an Initiator and action is as a Target.

   Suggested Response: Proceed with intended command sequence. After either Select command, wait for a Bus Service or Disconnect interrupt. After a Reselect command, issue a Send Message In to transmit an "Identify" message.

5. Interrupt Register (7-0): 0000 0100

   Auxiliary Status Register (7-0): xxxx xxxx

   Reason For Interrupt: While executing a Select or Reselect command, no response (BSY) was received from the destination device within the specified timeout. The operation was aborted.

   Suggested Response: Retry a limited number of times.

6. Interrupt Register (7-0): 0100 0000

   Auxiliary Status Register (7-0): xxxx xxxx

   Reason For Interrupt: The user issued a command that is not valid in the Disconnected state.

   Suggested Response: If the command is valid, either retry or issue a Chip Reset command and retry.

7. Interrupt Register (7-0): All Others

   Auxiliary Status Register (7-0): xxxx xxxx

   Reason for Interrupt: Chip malfunction.

   Suggested Response: Issue chip reset and retry operation.

5.3 USER CONNECTED AS TARGET ISR

In this state, the user is logically connected on the SCSI bus in the Target role. Commands that may be issued in this state are:

Interrupting
- RECEIVE COMMAND
- RECEIVE DATA
- RECEIVE MESSAGE OUT
- RECEIVE UNSPECIFIED OUTPUT
- SEND STATUS
- SEND DATA
- SEND MESSAGE IN
- SEND UNSPECIFIED INPUT

Immediate
- PAUSE
- DISCONNECT
- CHIP RESET

In order to service an interrupt in the Connected as Target state, the user should know the current command and if the pending command will result in an interrupt.

The seven interrupts that may occur in this state are numbered and described below.

1. Interrupt Register
   (7-0): 0000 0010

   Auxiliary Status
   Register (7-0): xxxx xxxx

   Reason for Interrupt: ATN was received from the Initiator. If this interrupt was received after issuing an interrupting command, the command was not and will not be executed by the chip.

   Suggested Response: Issue a Receive Message Out command to determine why the Initiator enabled ATN. If a command was aborted, it should be reissued.

2. Interrupt Register
   (7-0): 0000 0001

   Auxiliary Status
   Register (7-0): x0xx xxxx

   Reason for Interrupt: A Send or Receive command completed successfully.

   Suggested Response: Proceed with function by issuing any other valid command.

3. Interrupt Register
   (0-7): 0000 0011

   Auxiliary Status
   Register (0-7): x0xx xxxx

   Reason for Interrupt: A Send or Receive command has completed. ATN was enabled by the Initiator during the transfer.

   Suggested Response: Issue a Receive Message Out to determine why the Initiator set ATN.

4. Interrupt Register
   (0-7): 0000 0001

   Auxiliary Status
   Register (0-7): x1xx xxxx

   Reason for Interrupt: A Receive command terminated due to a bus parity error. (ATN is not enabled.)

   Suggested Response: If the error occurred during a Receive Message Out command, issue a Send Message In, "Message Parity Error" followed by a Receive Message Out in order to retry the message. If the error occurred during another Receive command, issue a Send Message In, "Restore State," and retry the entire transmission. In either case, the number of retries should be limited.

5. Interrupt Register
   (7-0): 0000 0011

   Auxiliary Status
   Register (7-0): x1xx xxxx

   Reason for Interrupt: A Receive command terminated due to a bus parity error, and the Initiator is asserting ATN.

   Suggested Response: Similar to previous interrupt, with an exception: if the error did not occur on a message, the ATN should be serviced first by issuing a Receive Message Out.

6. Interrupt Register
   (7-0): 0100 0000

   Auxiliary Status
   Register (0-7): xxxx xxxx

   Reason for Interrupt: The user issued a command that is not valid in the Connected as Target state.

Suggested Response: If the command is valid, retry or issue CHIP RESET and retry the entire operation.

7. Interrupt Register (7-0): All Others

   Auxiliary Status Register (0-7): xxxx xxxx

   Reason for Interrupt: Chip malfunction.

   Suggested Response: Issue chip reset and retry operation.

5.4 USER CONNECTED AS INITIATOR ISR

In the Connected as Initiator state, the user is logically connected on the SCSI bus in the Initiator role. Commands that may be issued in this state are:

Interrupting	TRANSFER INFO TRANSFER PAD
Immediate	MESSAGE ACCEPTED SET ATN DISCONNECT CHIP RESET

In order to service an interrupt, the user should know the current command and if the pending command will result in an interrupt. The information phase during the last Transfer command should also be noted.

The seven interrupts that may occur in the Connected as Initiator state are numbered and described below.

1. Interrupt Register (7-0): 0000 0010

   Auxiliary Status Register (0-7): xxxx xxxx

   Reason for Interrupt: A REQ has been received from a Target that the chip cannot service automatically. This may occur prior to issuing a Transfer command when a REQ is received after TC = 0 during a Transfer command, or when an information phase change is detected by the chip during a Transfer command.

   Suggested Response: Compare I/O, C/D, and MSG in the Auxiliary Status Register with the previous information phase to determine if an information phase change has occurred. If the phase type changed, read the Transfer counter and update working pointers for the old phase, and proceed to set-up for the new transfer (Refer to section 5.41 Bus Service Interrupt.) If the phase did not change, a buffer overflow has occurred, and the Transfer Counter Zero bit will have been set.

2. Interrupt Register (0-7): 0000 0100

   Auxiliary Status Register (0-7): xxxx xxxx

   Reason for Interrupt: The Target disconnected from the bus. The disconnection may or may not be expected, depending upon the previous sequence of events.

   Suggested Response: Do housekeeping to complete Initiator role.

3. Interrupt Register (0-7): 0000 0001

   Auxiliary Status Register (0-7): xxxx xxxx

   Reason for Interrupt: A Transfer command for a Message In phase has completed. ACK is left active on the bus.

Suggested Response: Examine the message. To reject the message, issue a Set ATN followed by a Message Accepted command. To accept the message, issue only the Message Accepted command.

4. Interrupt Register
   (0-7): 0000 0100

   Auxiliary Status
   Register (0-7): X1XX XXXX

   Reason for Interrupt: The Target disconnected from the bus when ATN was on due to a parity error.

   Suggested Response: Consider this I/O invalid since the Target never sent a Message Out to check the parity error. Abort the I/O.

5. Interrupt Register
   (0-7): 0000 0010

   Auxiliary Status
   Register (0-7): X1XX XXXX

   Reason for Interrupt: A REQ from the Target cannot be serviced automatically by the chip. Also, a parity error occurred during the last Transfer Info command.

   The interrupt does not occur at the time of the parity error, but when TC = 0 or the Target changes information phases. The chip automatically sets ATN when the parity error occurs.

   Suggested Response: Use I/O, C/D, and MSG to determine if a phase change occurred. If so, and the new phase is a Message Out, send either a "Message Parity Error" or an "Initiator Detected Error" message. (The choice depends on whether the last phase was a message phase.) If the new phase is not a Message Out, service the new phase and issue a Transfer command. (The chip will keep ATN on until a Message Out is sent with TC = 0.)

   If the phase did not change and the TC = 0, a buffer overflow occurred in addition to the parity error.

6. Interrupt Register
   (7-0): 0100 0000

   Auxiliary Status
   Register (0-7): XXXX XXXX

   Reason for Interrupt: The user issued a command that is not valid in the Connected as Initiator state.

   Suggested Response: If the command is valid, retry or issue a Chip Reset command, and retry the entire operation.

7. Interrupt Register
   (7-0): All Others

   Auxiliary Status
   Register (7-0): XXXX XXXX

   Reason for Interrupt: Chip malfunction.

   Suggested Response: Issue a Chip Reset command, and retry the operation.

5.4.1 Bus Service Interrupt

The NCR 5385E is designed to interrupt the user for a detected phase change, even when REQ is not active. This offers two advantages:

1. Provides early notification to the Initiator for unbuffered target devices, allowing the Initiator to prepare for the next information phase before it occurs.

2. In high performance systems, this early notification allows the Initiator to prepare the chip

for a requested phase change and increases the overall system performance.

When a phase change is detected by the chip, the phase lines are monitored for 12 clock periods. If the phase lines have indeed changed, the chip monitors the BSY line for an additional 12 clock periods to determine if the chip is still connected. If so, a Bus Service interrupt occurs, indicating a phase change. The user must respond to this phase change by issuing either a Transfer Info or Transfer Pad Command even if this is an unexpected bus phase.

One possible way to handle an invalid or unexpected bus phase is to program the Transfer Counter to a value of "1", and program the Command Register with a Transfer Pad command. If the Target requests data, the Transfer Counter goes to zero and the user receives an interrupt indicating that a REQ has occurred. The important point is that the Initiator must respond to all Bus Service Interrupts by issuing either a Transfer Info or Transfer Pad command to the chip.

The NCR 5386 defaults to NCR 5385E type operation but may be optionally programmed to ignore phase changes except when REQ is active. This is accomplished by setting Bit 3 (phase valid on REQ) in the control register.

5.5 INTERRUPT SUMMARY

The information provided in this section is summarized in the following table.

Table 5.1 Interrupt Summary

User State	Interrupt Register (7-0)	Auxillary Status Register (7-0)	Event
Disconnected	0000 1000	XXXX XXXX	Selected as Target, ATN off.
	0000 1010	XXXX XXXX	Selected as Target, ATN on.
	0001 0000	XXXX XXXX	Reselected as Initiator.
	0000 0001	XXXX XXXX	Select W/ATN, Select W/O ATN, or Reselect command completed successfully.
	0000 0100	XXXX XXXX	No response from Destination while executing a Select or Reselect command.
	0100 0000	XXXX XXXX	Invalid command issued.
	All Others	XXXX XXXX	Hardware Error - should not occur.
Connected as Target	0000 0010	XXXX XXXX	ATN received.
	0000 0001	X0XX XXXX	Send or Receive command successfully completed.
	0000 0011	X0XX XXXX	Send or Receive command completed: ATN was turned on during the transfer.
	0000 0001	X1XX XXXX	Receive command terminated due to bus parity error.
	0000 0011	X1XX XXXX	Receive command terminated due to bus parity error. ATN is on.
	0100 0000	XXXX XXXX	Invalid command issued.
	All Others	XXXX XXXX	Hardware Error - should not occur.
Connected as Initiator	0000 0010	XXXX XXXX	Service Target request.
	0000 0100	X0XX XXXX	Message In transfer completed.
	0000 0001	XXXX XXXX	Transfer for Message In has completed.
	0000 0100	X1XX XXXX	Target disconnected from bus. Did not respond to ATN due to parity error.
	0000 0010	X1XX XXXX	Service Target request. A parity error was previously detected and ATN turned on.
	0100 0000	XXXX XXXX	Invalid command issued.
	All Others	XXXX XXXX	Hardware error - should not occur.

SECTION 6
SCSI BUS INTERFACE

The NCR SCSI Protocol Controller supports either differential pair or open-collector operation. Differential pair operation allows bus devices to be spaced up to 25 meters apart and offers better noise immunity than the more prominent open-collector interface.

The open-collector or single-ended interface is recommended for in cabinet use and limits bus device spacing to 6 meters.

Figure 6.1. shows the suggested interface between the SCSI Protocol Controller and the differential pair transceivers. A 3-to-8 decoder, gated by the ARB signal, is used to enable the driver for the device ID used during arbitration. At this time, all other data bit receivers are enabled for reading and the Protocol Controller drives the appropriate device ID data bit high.

The single-ended interface may be simply implemented using the NCR 8310 General Purpose 48 ma Driver/Receiver Chip. The equivalent circuit for the NCR 8310 is shown in Figure 6.2. Aside from providing 48 ma sink capability for the SCSI bus, this device may be used with other common device interfaces that require 48 ma operation. The interface to the NCR SCSI Protocol Controller is shown in Figure 6.3.

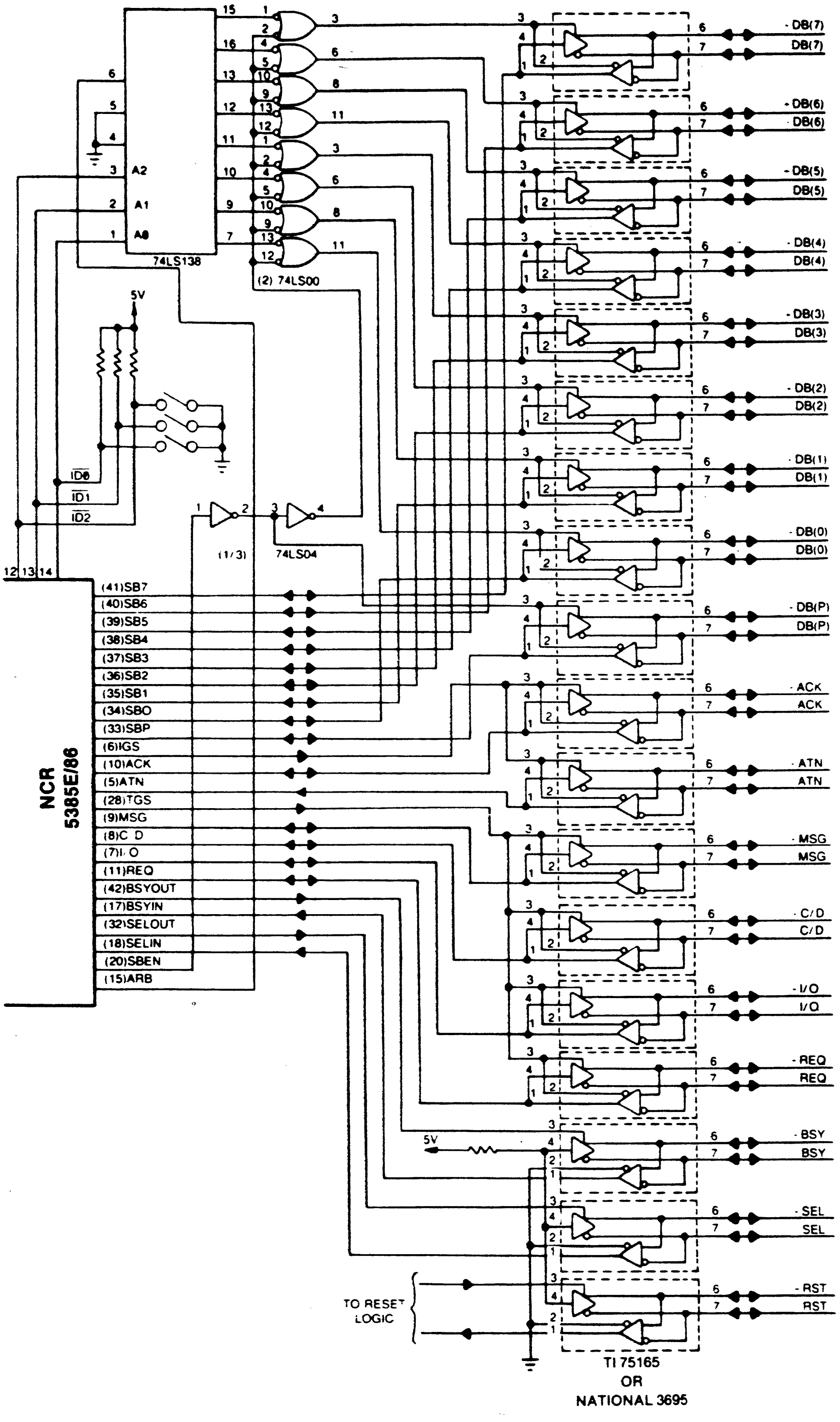


Figure 6.1 Suggested Interface to SCSI Differential Transceivers

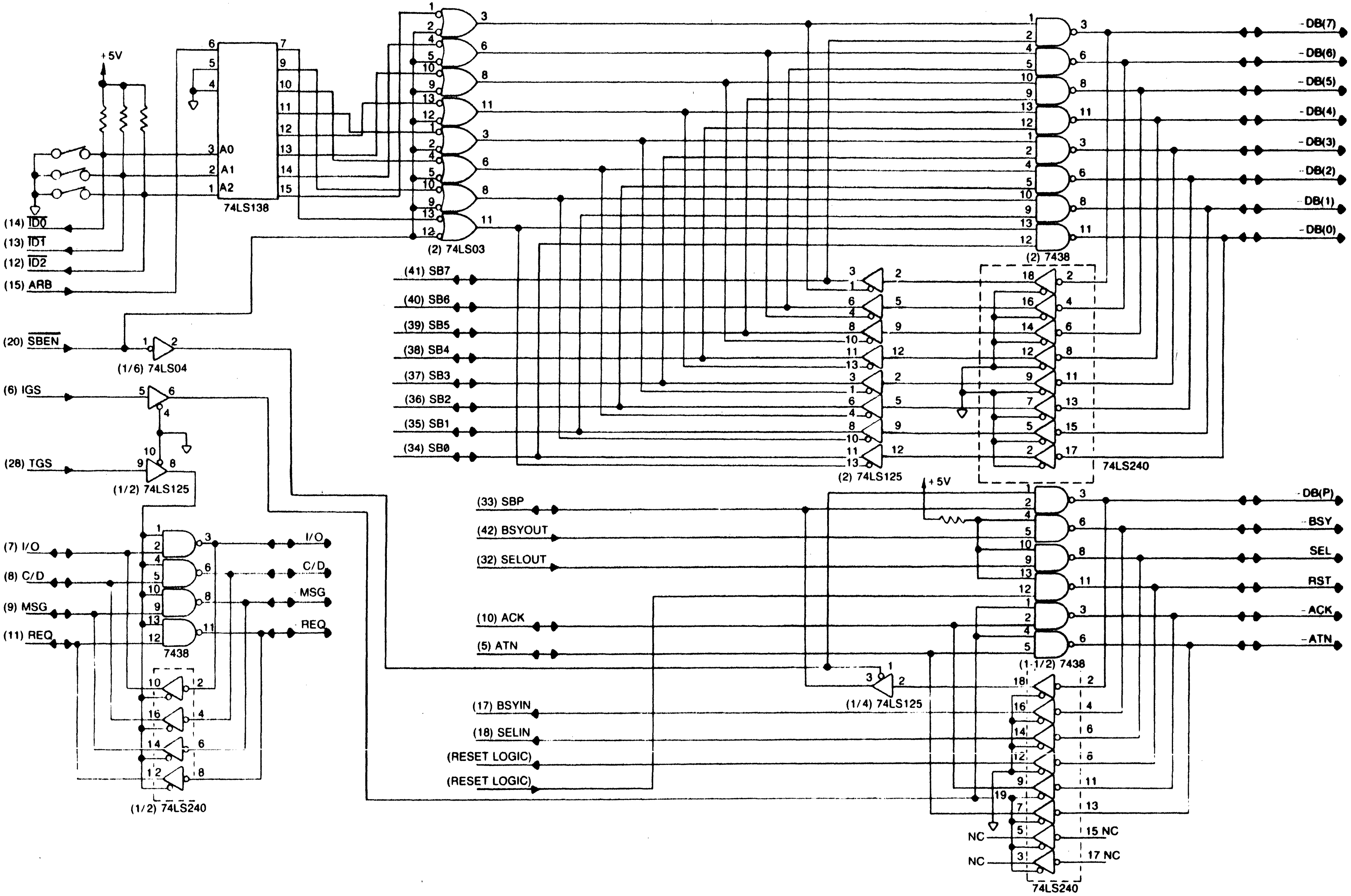


Figure 6.2 NCR 8310 Equivalent Circuit

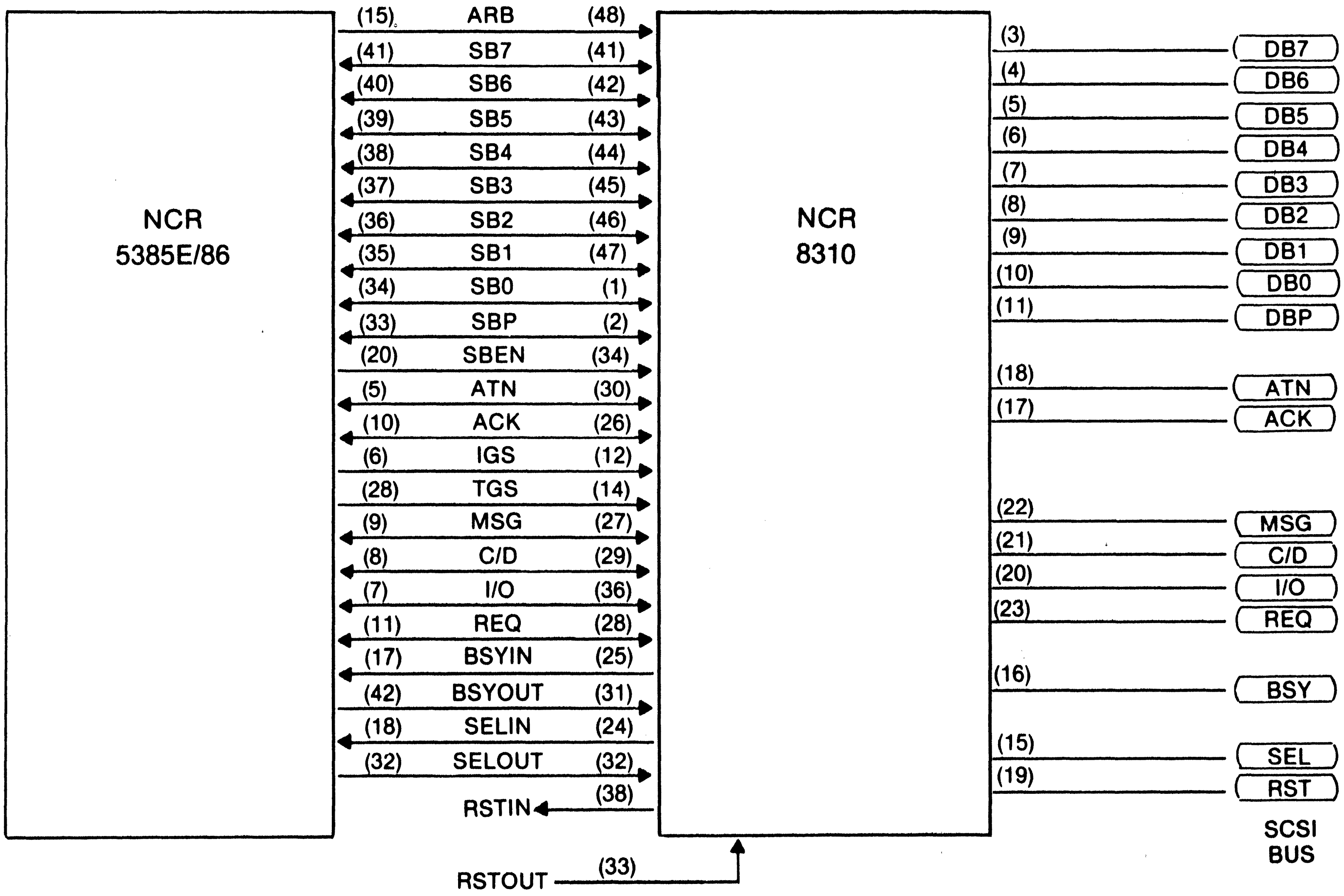


Figure 6.3 Single-Ended Interface Using the NCR 8310 Driver/Receiver Chip

APPENDIX A
NCR 5385E/86 SCSI PROTOCOL CONTROLLER REGISTER AND COMMAND SUMMARY

REGISTER SUMMARY

A3	A2	A1	A0	R/W	REGISTER NAME
0	0	0	0	R/W	Data Register I
0	0	0	1	R/W	Command Register
0	0	1	0	R/W	Control Register
0	0	1	1	R/W	Destination ID
0	1	0	0	R	Auxiliary Status
0	1	0	1	R	ID Register
0	1	1	0	R	Interrupt Register
0	1	1	1	R	Source ID
1	0	0	0	R	Data Register II*
1	0	0	1	R	Diagnostic Status
1	1	0	0	R/W	Transfer Counter (MSB)
1	1	0	1	R/W	Transfer Counter (2nd BYTE)
1	1	1	0	R/W	Transfer Counter (LSB)
1	1	1	1	R/W	Reserved for Testability

*NCR 5386 ONLY

COMMAND SUMMARY

INT = INTERRUPTING
IMM = IMMEDIATE

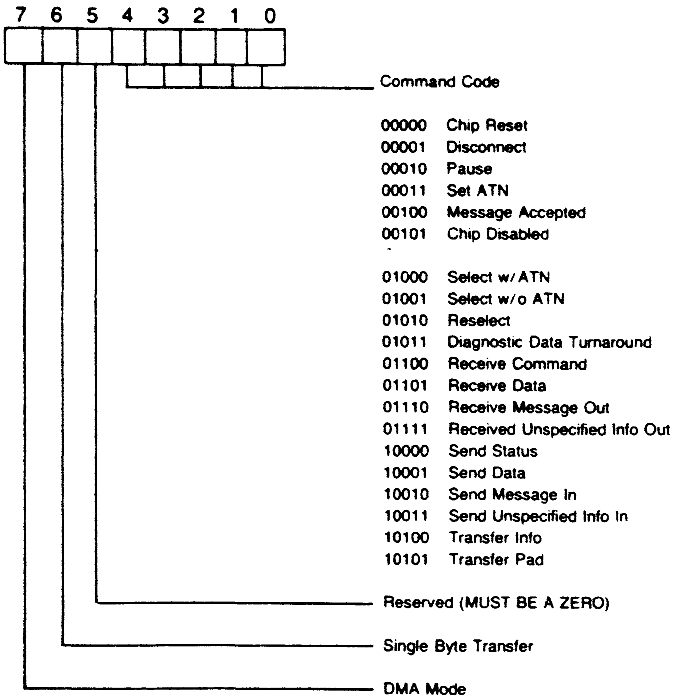
D = DISCONNECTED
T = CONNECTED AS A TARGET

I = CONNECTED AS AN INITIATOR

COMMAND CODE	COMMAND	TYPE	VALID STATES
00000	Chip Reset	IMM	D,I,T
00001	Disconnect	IMM	I,T
00010	Paused	IMM	D,T
00011	Set ATN	IMM	I
00100	Message Accepted	IMM	I
00101	Chip Disable	IMM	D,I,T
00110-00111	Reserved	IMM	
01000	Select w/ATN	INT	D
01001	Select w/o ATN	INT	D
01010	Reselect	INT	D
01011	Diagnostic	INT	D
01100	Receive Command	INT	T
01101	Receive Data	INT	T
01110	Receive Message Out	INT	T
01111	Receive Unspecified Info Out	INT	T
10000	Send Status	INT	T
10001	Send Data	INT	T
10010	Send Message Out	INT	T
10011	Send Unspecified Info In	INT	T
10100	Transfer Info	INT	I
10101	Transfer Pad	INT	I
10110-11111	Reserved	INT	

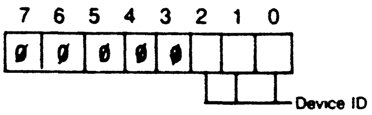
APPENDIX B
INTERNAL REGISTERS

COMMAND REGISTER

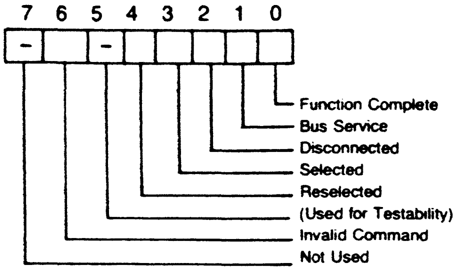


00000	Chip Reset
00001	Disconnect
00010	Pause
00011	Set ATN
00100	Message Accepted
00101	Chip Disabled
01000	Select w/ATN
01001	Select w/o ATN
01010	Reselect
01011	Diagnostic Data Turnaround
01100	Receive Command
01101	Receive Data
01110	Receive Message Out
01111	Received Unspecified Info Out
10000	Send Status
10001	Send Data
10010	Send Message In
10011	Send Unspecified Info In
10100	Transfer Info
10101	Transfer Pad

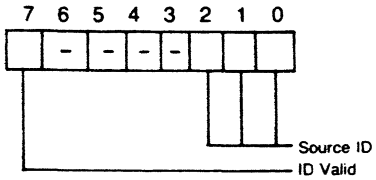
ID REGISTER



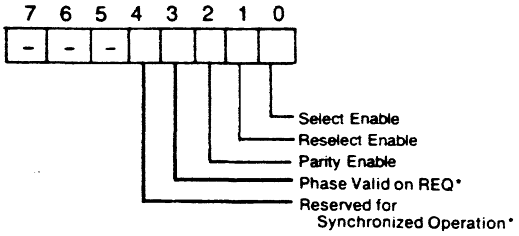
INTERRUPT REGISTER



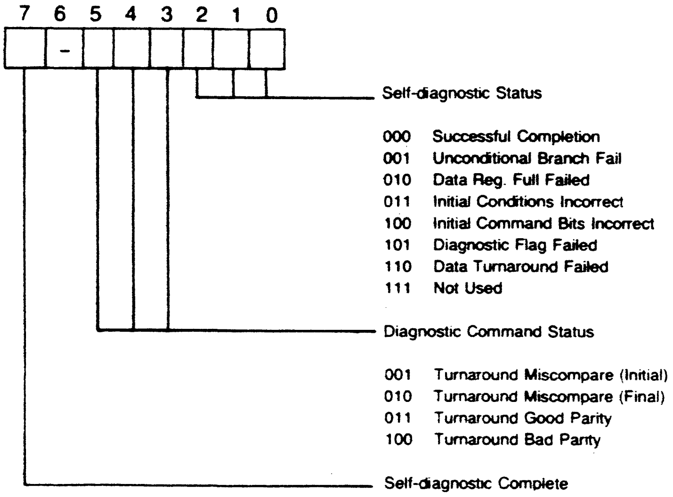
SOURCE ID REGISTER



CONTROL REGISTER



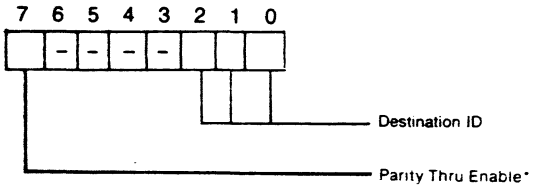
DIAGNOSTIC STATUS REGISTER



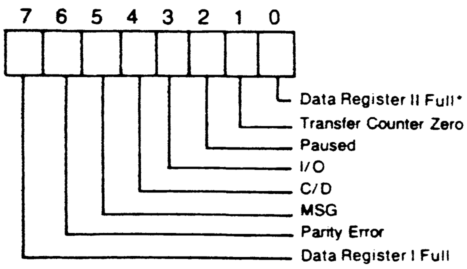
000	Successful Completion
001	Unconditional Branch Fail
010	Data Reg. Full Failed
011	Initial Conditions Incorrect
100	Initial Command Bits Incorrect
101	Diagnostic Flag Failed
110	Data Turnaround Failed
111	Not Used

001	Turnaround Miscompare (Initial)
010	Turnaround Miscompare (Final)
011	Turnaround Good Parity
100	Turnaround Bad Parity

DESTINATION ID REGISTER



AUXILIARY STATUS REGISTER

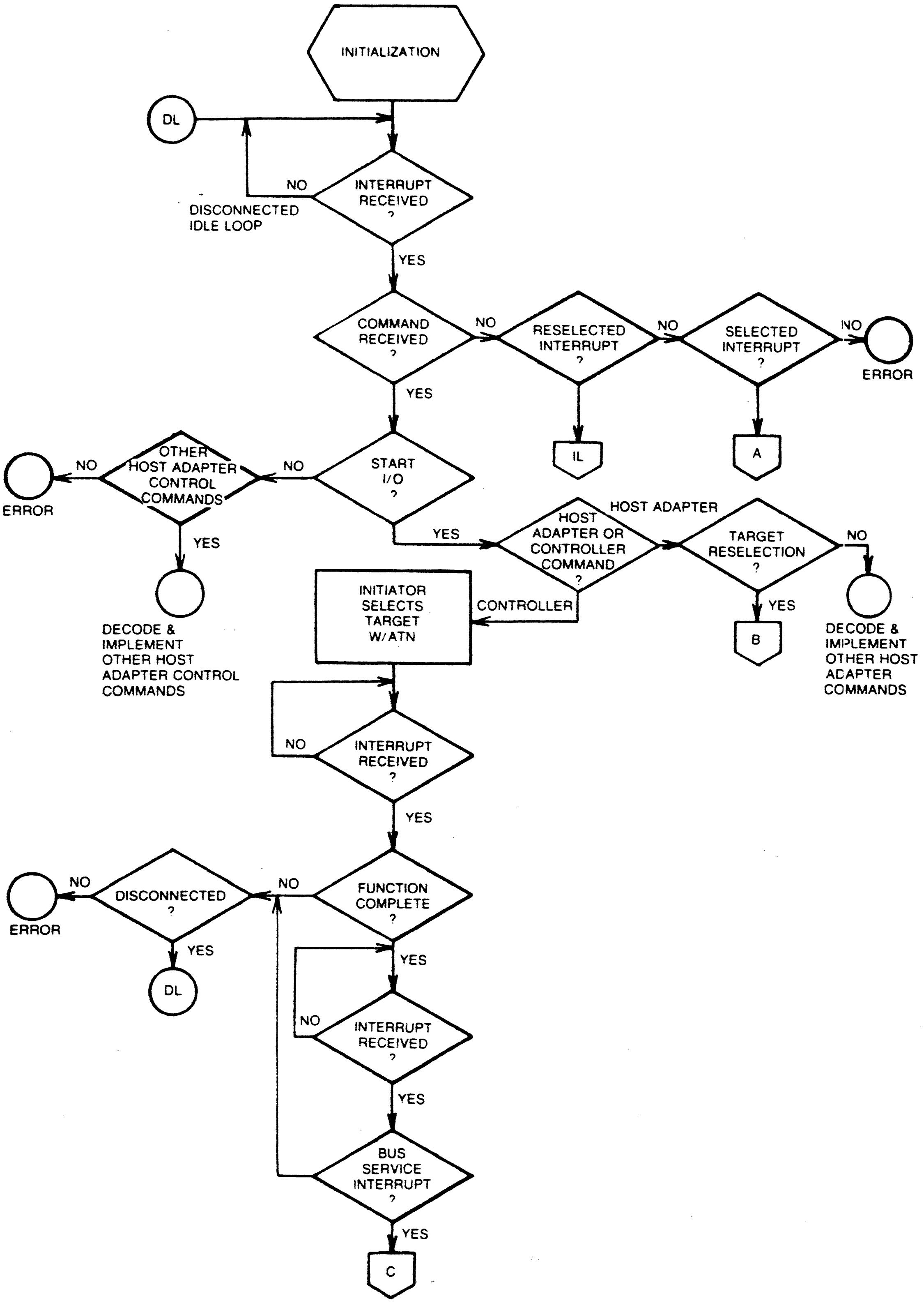


*NCR 5386 ONLY

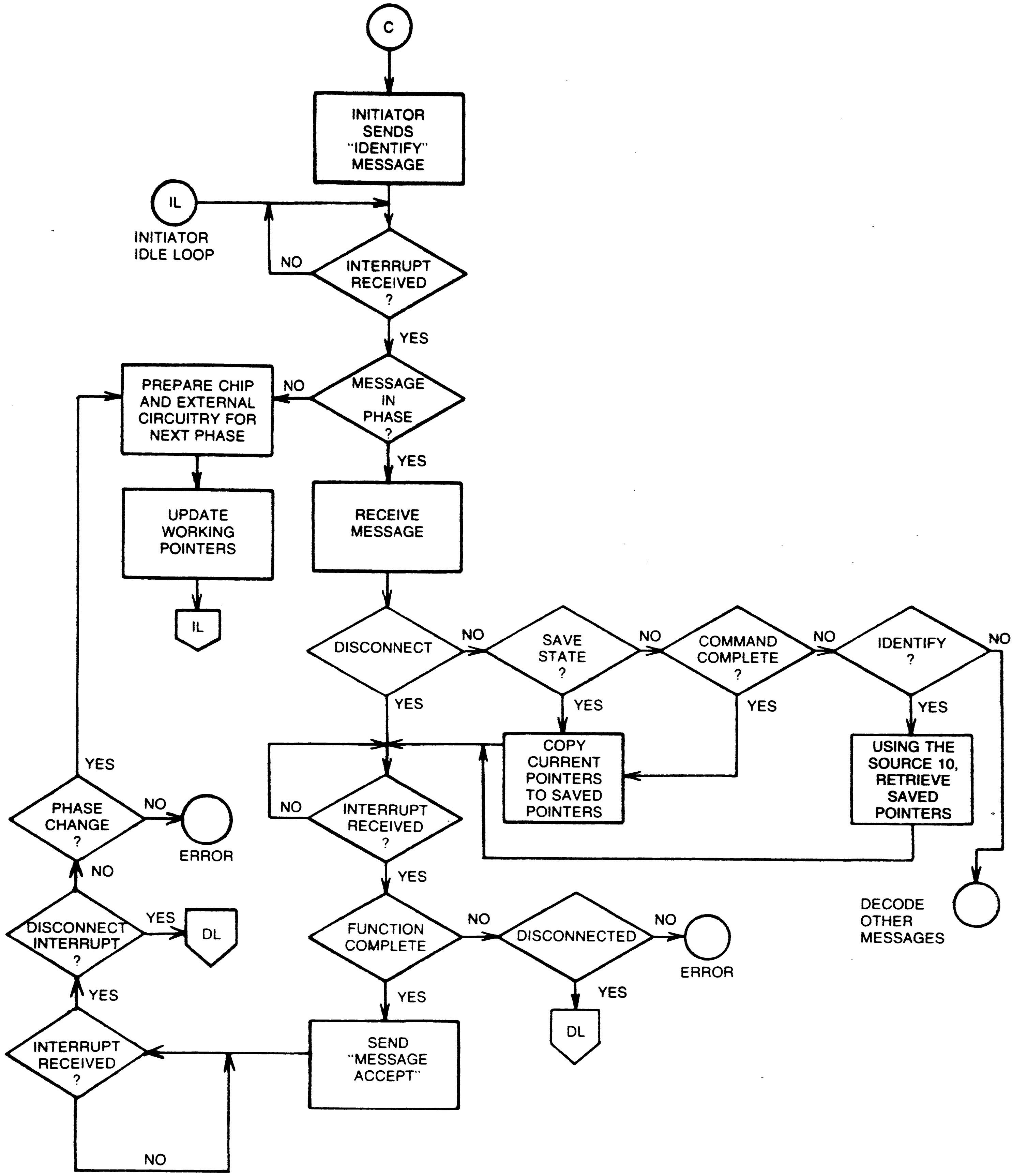
TRANSFER COUNTER

A3	A2	A1	A0	SELECTED BYTE
1	1	0	0	Most Significant Byte
1	1	0	1	Middle Byte
1	1	1	0	Least Significant Byte

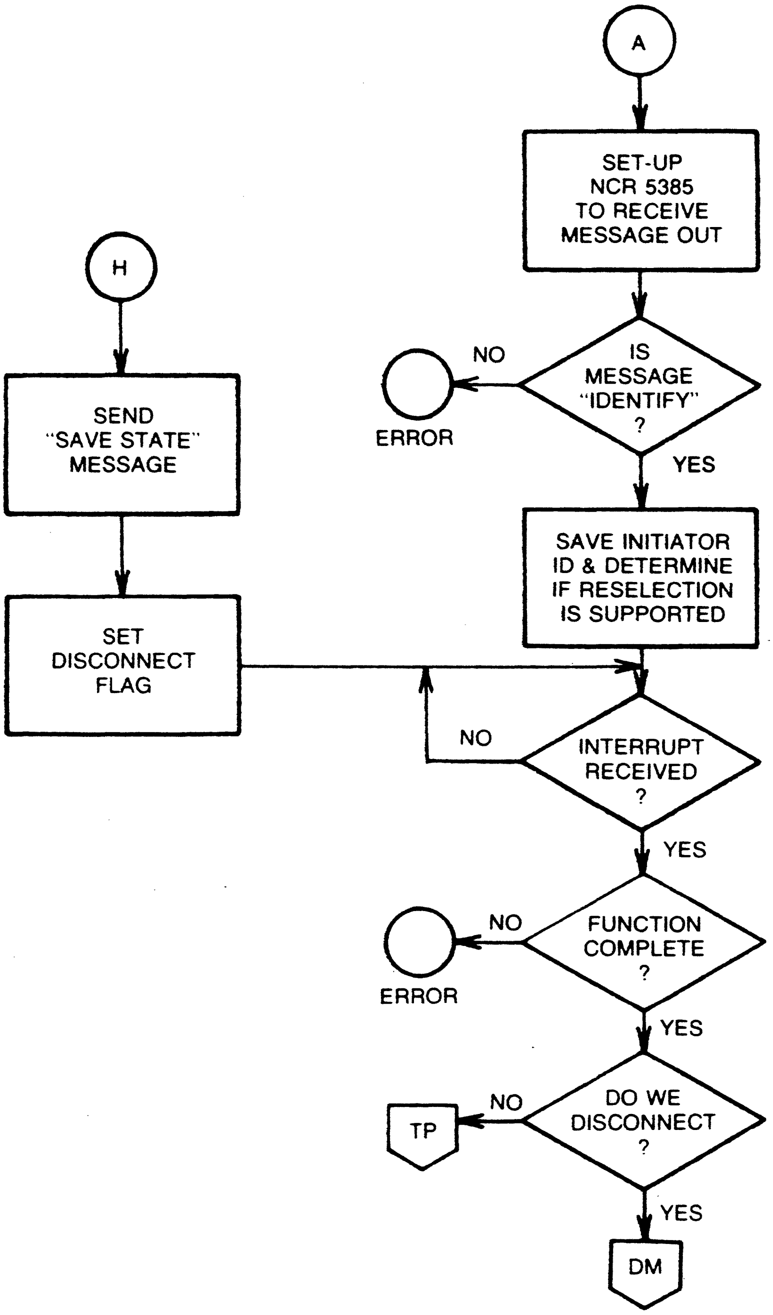
APPENDIX C
INITIATOR/TARGET ROLE FLOWCHART



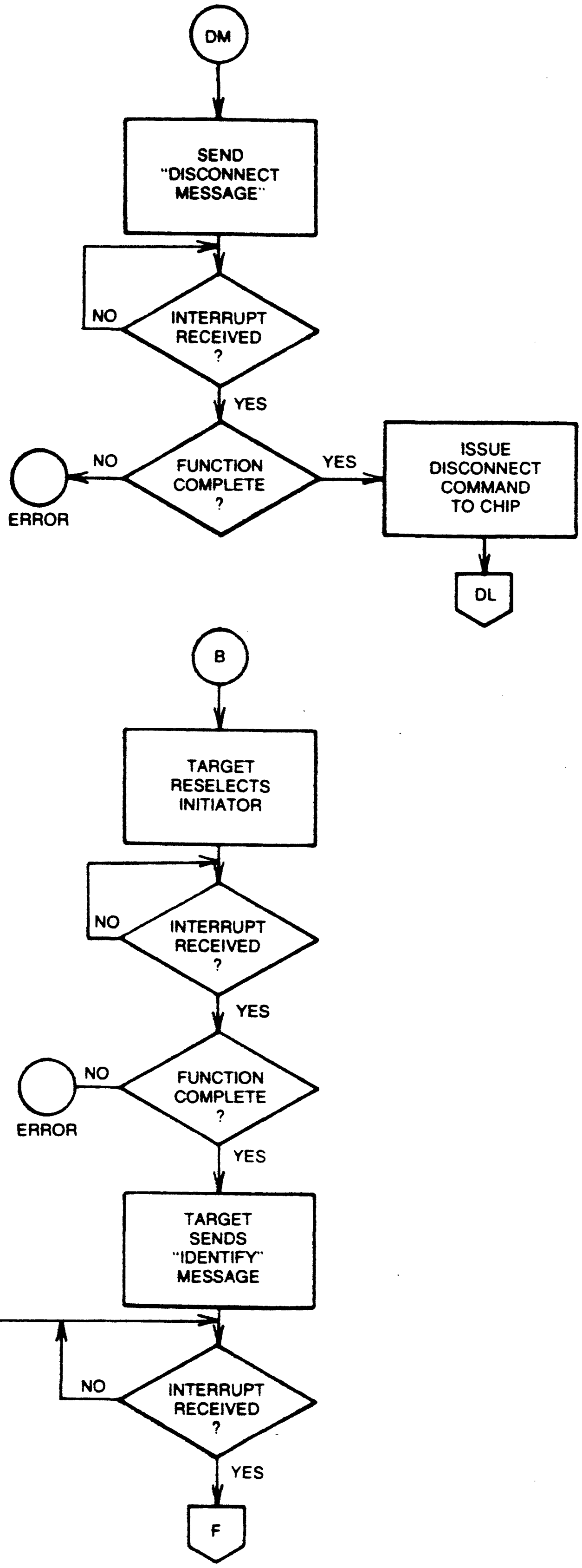
INITIATOR/TARGET ROLE FLOWCHART



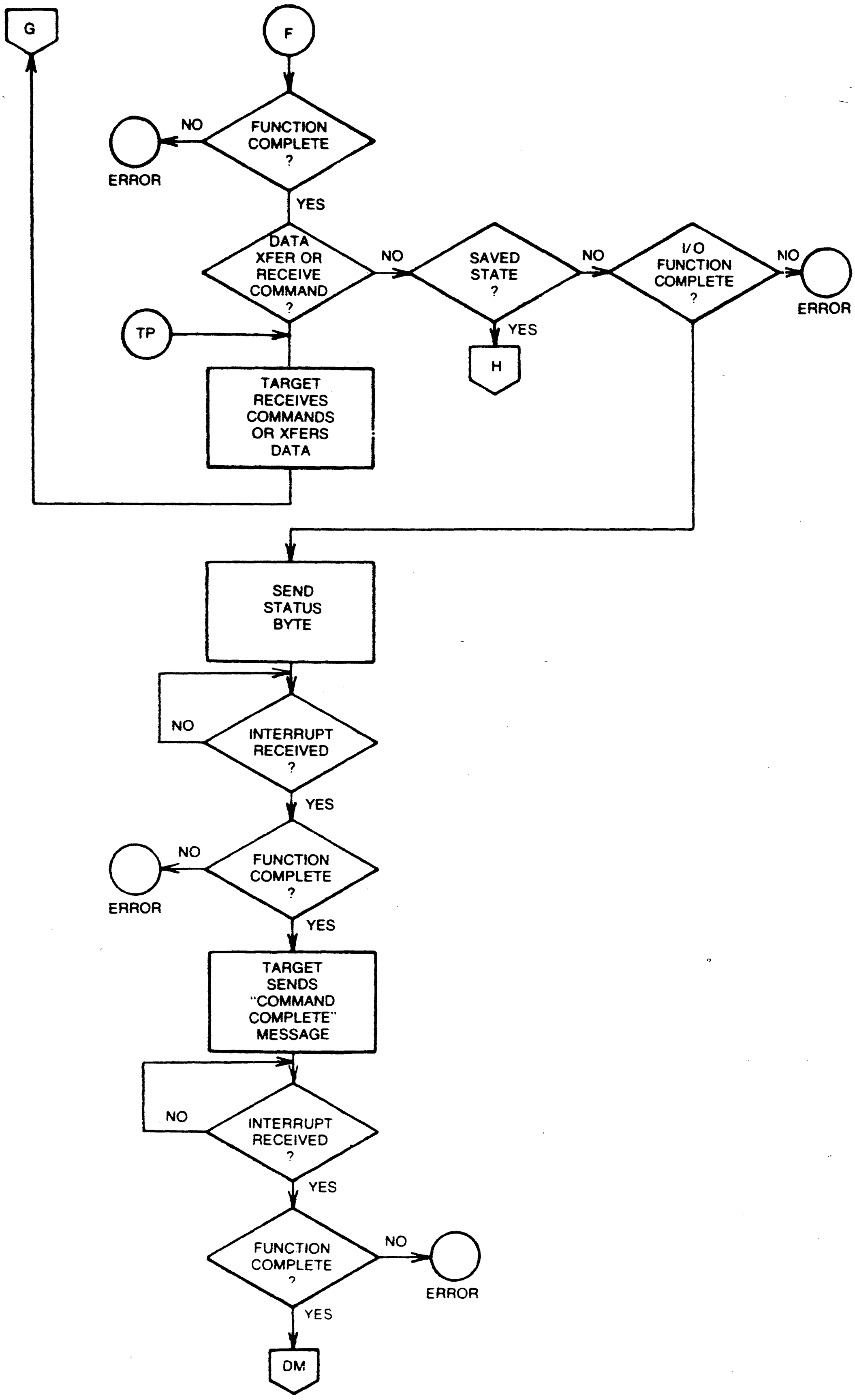
INITIATOR/TARGET ROLE FLOWCHART



INITIATOR/TARGET ROLE FLOWCHART (Continued)



INITIATOR/TARGET ROLE FLOWCHART (Continued)


G
F
FUNCTION COMPLETE ?
NO
ERROR
YES
DATA XFER OR RECEIVE COMMAND ?
NO
SAVED STATE ?
NO
I/O FUNCTION COMPLETE ?
NO
ERROR
YES
H
TP
TARGET RECEIVES COMMANDS OR XFERS DATA
SEND STATUS BYTE
NO
INTERRUPT RECEIVED ?
YES
NO
FUNCTION COMPLETE ?
ERROR
YES
TARGET SENDS "COMMAND COMPLETE" MESSAGE
NO
INTERRUPT RECEIVED ?
YES
FUNCTION COMPLETE ?
NO
ERROR
YES
DM