MVME101/D2

VMEmodules

MC68000 Monoboard Microcomputer User's Manual

MOTOROLA

M V M E 1 0 1

M C 6 8 0 0 0 M O N O B O A R D C O M P U T E R

U S E R ´ S M A N U A L

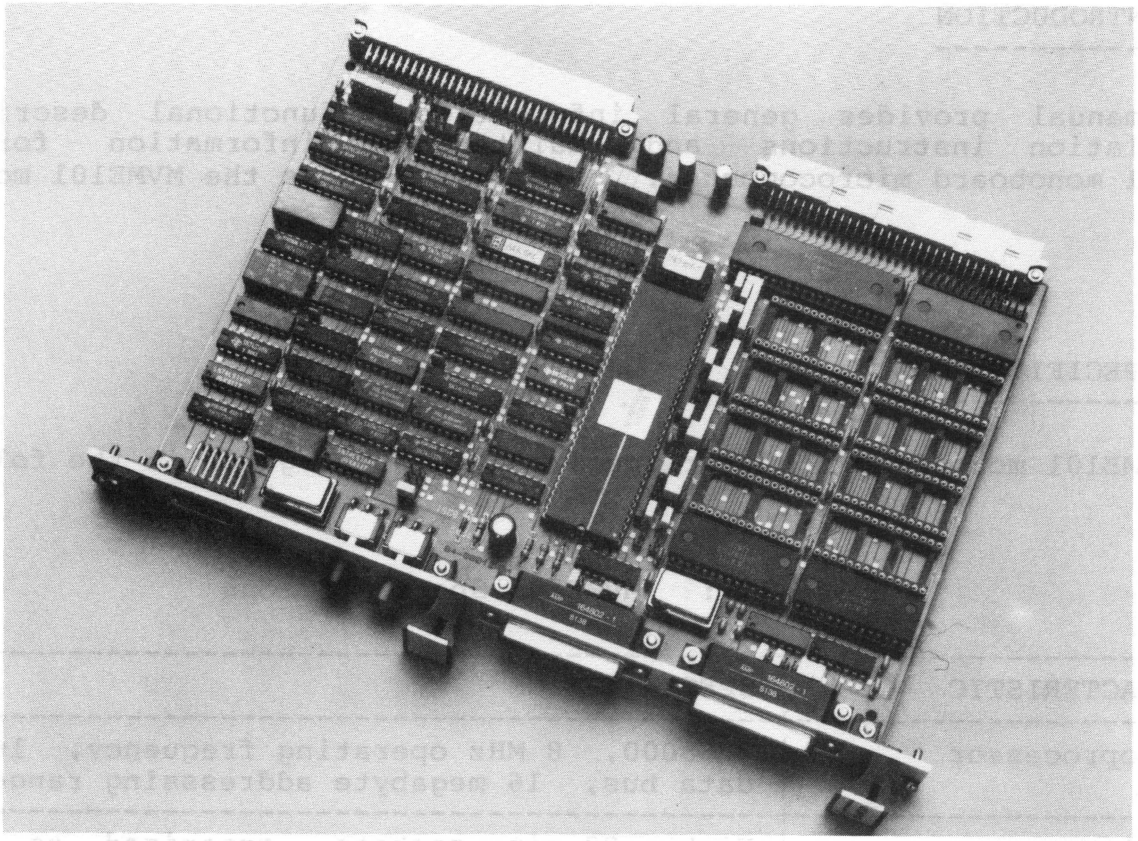
Second Edition, July 1983

TABLE OF CONTENTS Page

LIST OF TABLES

LIST OF FIGURES

Figure 1.1: The MVME101 Monoboard Computer



CHAPTER 1

GENERAL INFORMATION

1.1. INTRODUCTION

This manual provides general information, functional description, installation instructions and maintenance information for the MVME101 monoboard microcomputer. Figure 1.1 shows the MVME101 module.

1.2. SPECIFICATIONS

The MVME101 monoboard computer specifications are given in the following table.

Table 1.1: MVME101 Specifications

CHARACTERISTIC	SPECIFICATION
Microprocessor	MC68000, 8 MHz operating frequency, 16-bit data bus, 16 megabyte addresssing range
Local Memory	Eight 28-pin sockets, organized as four pairs, for user-provided memory. Each pair is individually configurable to accept any JEDEC-standard compatible byte-wide static RAM or ROM devices, ranging from 2K to 32K bytes each. Local RAM is accessed without wait cycles, local ROM access time is selectable.
Serial I/O-Ports	Two Motorola MC68661 Enhanced Programmable Communication Interfaces, featuring several synchronous and asynchronous protocols and software selectable baud rates from 50 to 19200 baud. Both ports are RS232C standard compatible, may be configured as data set or data terminal, and are available at 25-pole connectors at the front panel.
Parallel I/O-Ports	A Motorola MC6821 Parallel Interface Adapter provides two independent programmable 8-bit I/O ports with two handshake lines and one interrupt output to the MPU. All peripheral I/O signals are available at the lower rear connector.

Figure 4.9: Schematic Diagram Sheet 8/11

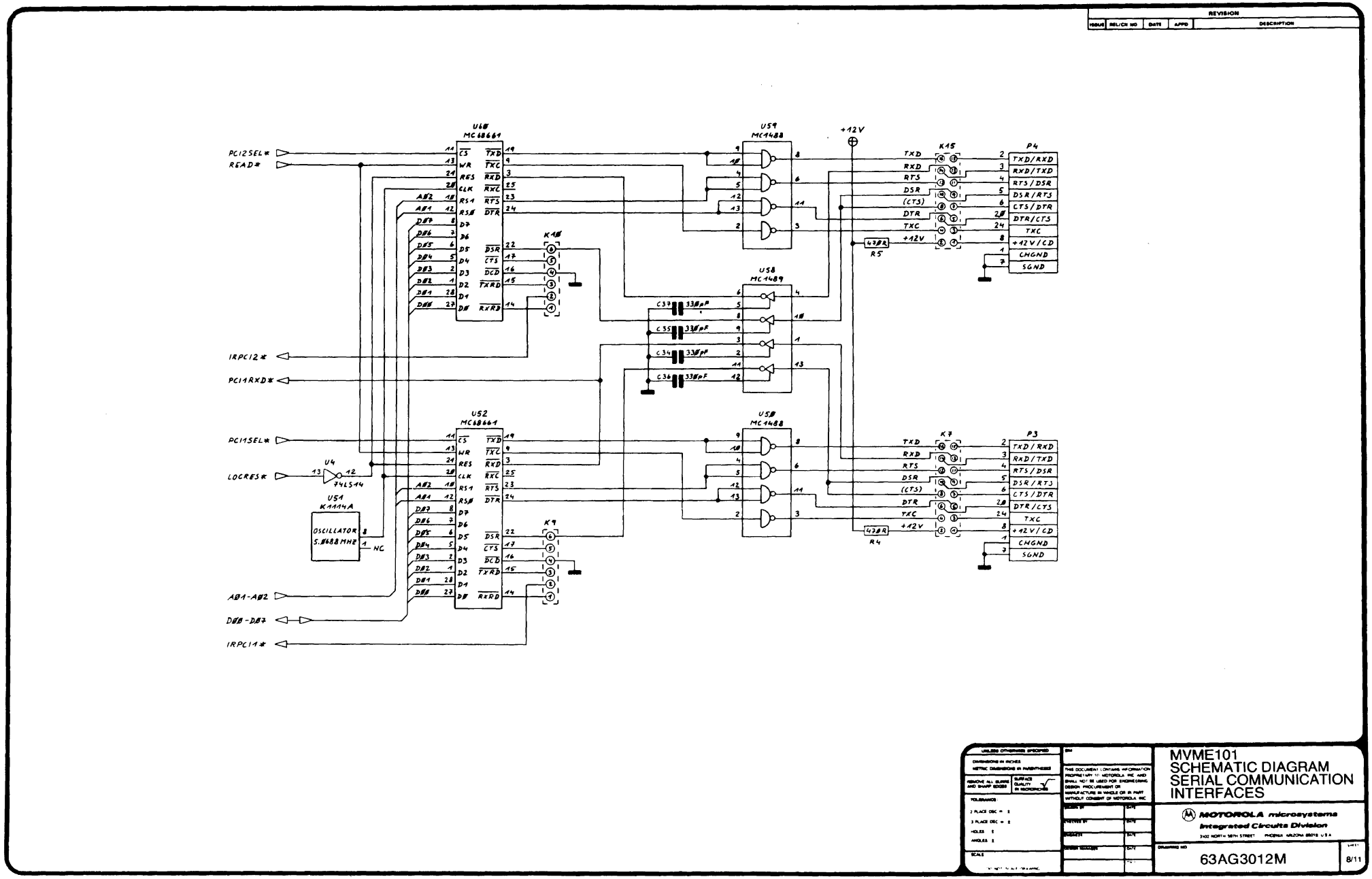


Table 1.1: MVME101 Specifications (cont'd)

CHARACTERISTIC	SPECIFICATION
Timer/Counter	A Motorola MC6840 Programmable Timer Module contains three independent 16-bit counters. All peripheral clock, gate and output lines are available at the lower rear connector. A jumper area provides gate enabling, real time counting, bus cycle counting and timer cascading.
Address Map	Decoder logic devides a 2 Megabyte address range (000000 - 0FFFFF and F00000 - FFFFFF) into 512 segments, each covering 4K bytes. An address decoder PROM assigns each of these segments to one of the four on-board memory pairs, to the on-board I/O devices, or to off-board resources on the VMEbus. All addresses from 100000 to EFFFFF are assumed to be off board and directed to the VMEbus.
VMEbus Interface	The private bus interconnecting all on-board devices is connected to the VMEbus through a VMEbus interface when off-board resources are to be accessed. This interface is fully compatible with the VMEbus Specification Rev.B.
VMEbus Requester	For implementation in multiprocessor systems the module contains a VMEbus requester which requests and releases the bus either under direct software control, or indirectly upon decoding off-board and on-board addresses. The bus requester is selectable to operate on one of four prioritized bus arbitration levels.
VMEbus Arbiter	For use as the system controller in a VMEbus system, the module contains an option ONE bus arbiter, supporting daisy-chained bus arbitration on a single level.
Interrupt Handler	Any or all of the seven VMEbus interrupt request lines can be strapped to generate prioritized and user-vectorized interrupts. The interrupt outputs of the on-board I/O devices and the VMEbus signals BCLR* and SYSFAIL* can be jumpered to any of six prioritized and auto-vectorized interrupts. The ABORT pushbutton and the VMEbus signal ACFAIL* generate a non-maskable auto-vectorized interrupt.
Time Out Counters	Two software controlled time-out counters supervise VMEbus operations. A bus error can be generated if a bus request is not granted within 128 microseconds, or if a bus data transfer is not acknowledged within 8 microseconds.

Table 1.1: MVME101 Specifications (cont'd)

CHARACTERISTIC	SPECIFICATION
Display	Programmable hexadecimal LED display at the front panel for status indication.
Front Panel Controls	Two pushbutton switches at the front panel for System Reset and Software Abort.
System Control	For use as the system controller in a VMEbus system, the module can be configured to drive the bus signals SYSCLK and SYSRESET*.
Control Register	Through an 8-bit Module Control Register the MPU controls the status display, the VMEbus output SYSFAIL*, the VMEbus requester, and the time-out counters.
Status Register	Through an 8-bit Module Status Register the MPU can monitor the VMEbus signals ACFAIL*, SYSFAIL* and BCLR*, the VMEbus availability, the activation of the Software Abort switch, the data input of Serial Port 1, and the occurence of a time-out condition.
Mechan. Dimensions	Double height VME board with front panel Board Size: 233 mm x 160 mm Front Panel Size: 262 mm x 20 mm
Connectors	One 96 pole DIN 41612 connector for VMEbus, one 64 pole DIN 41612 connector for parallel I/O and timer signals, two 25 pole D-Subminiatur connectors for the serial ports.
Power Requirements (See Note)	+ 5 V DC (+/- 5%), 2.0 A (typ), 3.0 A (max) +12 V DC (+/- 5%), 25 mA (typ), 50 mA (max) -12 V DC (+/- 5%), 25 mA (typ), 50 mA (max)
Temperature Range	Operating temperature: 0 to 55 C Storage temperature: -40 to 100 C
Rel. Humidity Range	Operating humidity: 0% to 90% non-condensing

Note: The current at +5 V DC is specified for the MVME101 module without any local memory. To calculate the actual required value, add the supply current of the memory devices used.

The currents at +12 V and -12 V DC are specified for the MVME101 module with the serial port connectors open. The actual required values depend on the load of the RS232C ports. All serial port outputs are current-limited to sink or source 12 mA (max) each.

1.3. REFERENCE MANUALS

The following manuals may be used for further information about the MC68000 microprocessor, the MC6840 timer module, the VMEbus system and the VMEbug debugger/monitor:

* MC68000UM MC68000 16-bit Microprocessor User's Manual
* MC6840UM MC6840 Programmable Timer Fundamentals and Applications
* MVMEBS VMEbus Specification Manual
* MVME101BUG MVME101bug Debug Package User's Manual

1.4. MANUAL TERMINOLOGY

1.4.1. Address and Data Format

Throughout this manual, unless otherwise noted, all address and data values are given in hexadecimal format.

1.4.2. Electrical Signal Levels

A signal line is always assumed to be in one of two levels, or in transition between these levels. Whenever the term "high" is used, it refers to a high TTL voltage level (> +2.0 V). The term "low" refers to a low TTL voltage level (< +0.8 V). There are two possible transitions which can appear on a signal line, and these will be referred to as "edges". A "rising edge" is defined as the time period during which a signal line makes its transition from a low level to a high level. The "falling edge" is defined as the time period during which a signal line makes its transition from a high level to a low level.

A signal is defined as "active low", if the function associated with the signal line is valid or initiated by either a low level or a falling edge on the signal line. The mnemonics of active low signals are marked with the suffix "*".

A signal is defined as "active high", if the function associated with the signal line is valid or initiated by either a high level or a rising edge on the signal line.

1.4.3. Logic Signal States

The terms "assert" and "negate" describe the logic state of a signal without indicating the associated voltage level. An active low signal is asserted when its voltage level is low, it is negated when its voltage level is high. An active high signal is asserted when its voltage level is high, it is negated when its voltage level is low.

For signals which are driven by three-state or open-collector outputs, the term "release" describes the high impedance state of the corresponding driver. Typically these signal lines are driven to a high voltage level by pull-up resistors when all drivers on the line are turned off.

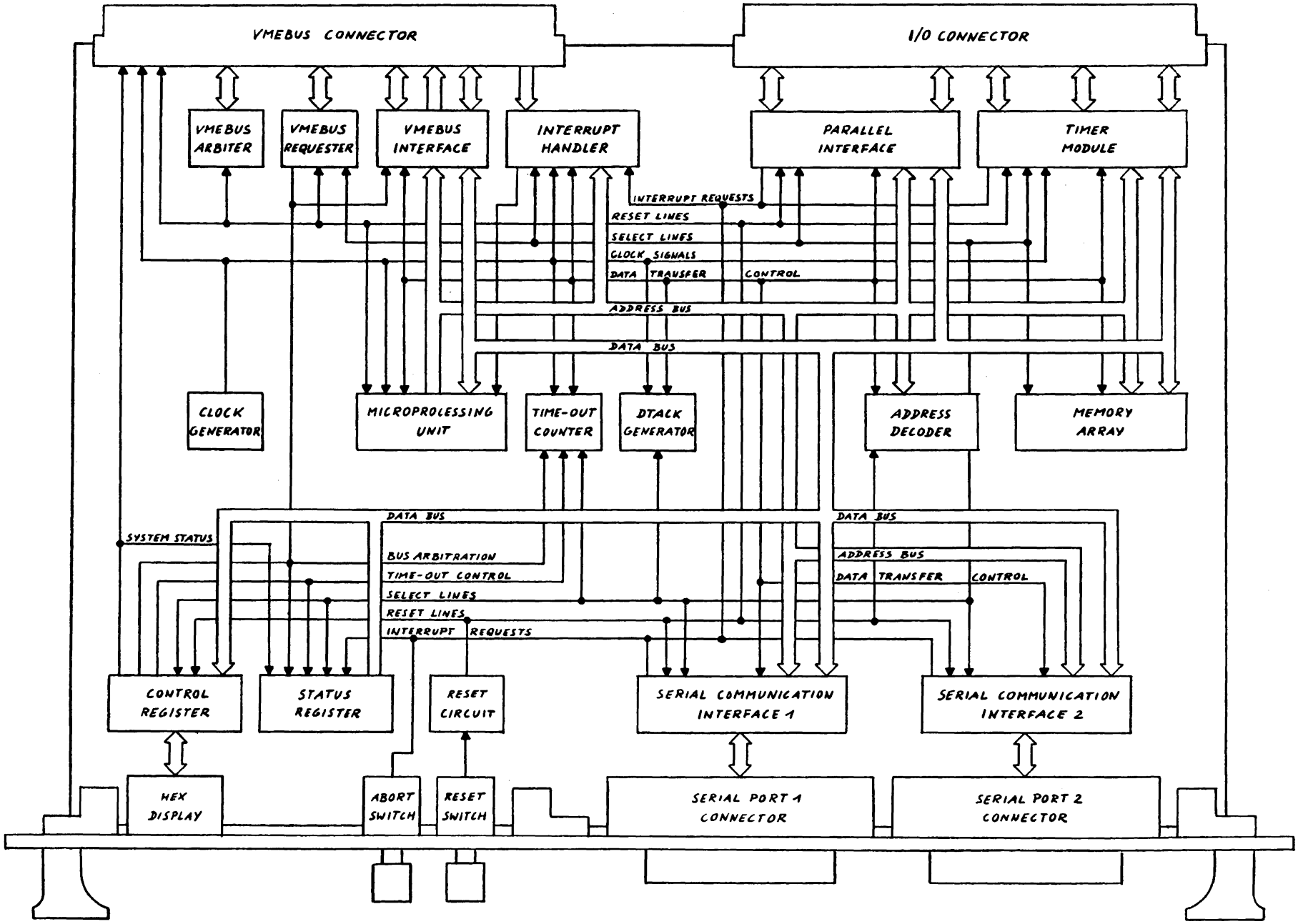


Figure 2.1: MVME101 Block Diagram

CHAPTER 2

FUNCTIONAL DESCRIPTION

2.1. INTRODUCTION

This chapter provides a detailed description of the MVME101 monoboard computer and its various modes of operation. The module can be regarded as consisting of functional blocks, as shown in Figure 2.1. Each block is described in a seperate paragraph in this chapter. For hardware details, Chapter 4 includes the schematic diagrams and an assembly drawing.

The MVME101 is designed to operate either as a monoboard system, as a single MPU controller in a VMEbus system, or as a MPU element in a multiprocessor configuration. Hardware and software application hints for each of these modes are given in this chapter. Detailed electrical and timing specifications of the VMEbus connector signals allow the user to design peripheral modules and his target hardware around the monoboard computer without requiring measurements on the board.

2.2. MICROPROCESSING UNIT

The microprocessing unit of the MVME101 consists of the Motorola MC68000 MPU and some interfacing hardware for other functional blocks. The microprocessor runs at 8 MHz clock frequency.

A detailed description of the microprocessor is given in the Motorola MC68000 Data Sheet in Appendix A of this User's Guide.

2.3 MEMORY

2.3.1. Data Organization In Memory

The 16-bit data word of the MC68000 MPU is separated into a lower data byte (D00-D07) and an upper data byte (D08-D15), corresponding to a given memory address (A01-A23). The address line A00 is only internal to the MPU and externally replaced by the data strobe signals LDS* and UDS*. A detailed description of the data organization in memory can be found in the MC68000 Data Sheet in Appendix A.

Accordingly any memory block for the MC68000 must be made up of two identical halves, one of them connected to the lower order data lines D00-D07 and activated by LDS*, the other half connected to the upper order data lines D08-D15 and activated by UDS*.

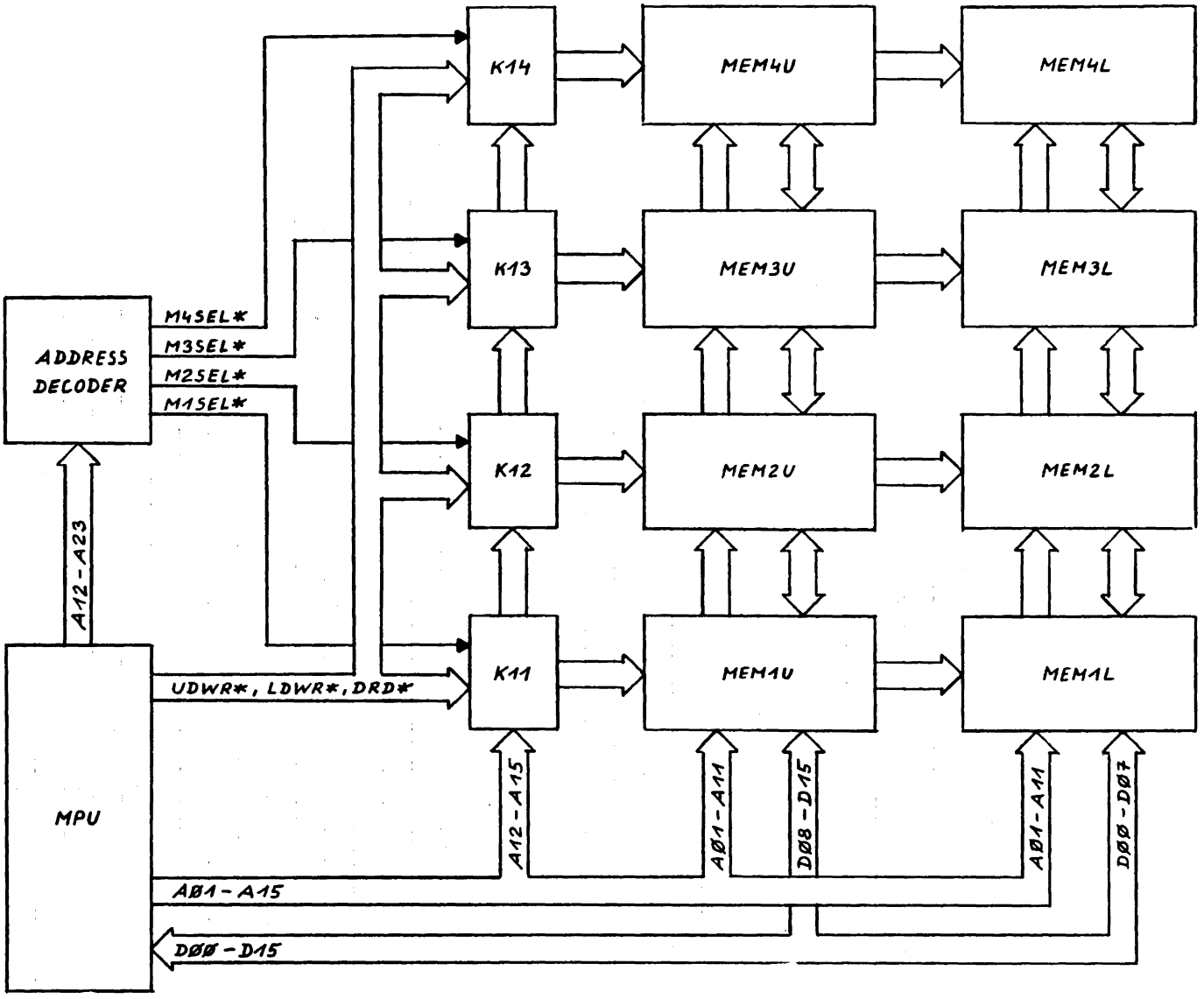
2.3.2. Memory Array

As shown in Figure 2.2, the memory array of the MVME101 consists of eight 28-pin sockets, organized as four pairs, for user-provided memory. These sockets accept any RAM or ROM devices which meet the following specifications:

- 24-pin or 28-pin dual-in-line package compatible with the JEDEC standard pin-out for byte-wide memories,
- memory size 2K, 4K, 8K, 16K, or 32K bytes per device,
- static operation,
- single + 5 V power supply,
- high impedance inputs (MOS characteristic), three-state outputs,
- timing requirements accordant with the specifications given in Paragraph 2.12.

A jumper area is associated with each memory pair to support different device sizes and pin-outs. Paragraph 3.4.8 describes the configuration of these jumpers.

Figure 2.2: Memory Array



2.3.3. Memory Map

For the first four MPU cycles after a board reset, data is fetched from the memory devices located in the socket pair 4, regardless of the addresses assigned. Therefore, the sockets MEM4L and MEM4U must be populated with ROM, and the first eight bytes of this ROM must contain the initial supervisor stack pointer and program counter values.

For the socket pairs 1, 2, and 3, the user is free to install either ROM or RAM or to leave them open. Each memory pair may be placed anywhere in a 2 Megabyte address range (000000 - 0FFFFF and F00000 - FFFFFF) by programming an address decoder PROM according to the desired memory map. Paragraph 2.7 gives a detailed description of the Address Decoder.

As socket pair 4 must contain ROM in any case, it is preferable that this firmware includes at least the board initialization, system monitoring, and failure servicing routines, to ensure their proper execution with a minimum of hardware involved. For the same reason the exception vector table and the stack should reside in on-board RAM.

2.3.4. Memory Access Time

Data transfers between MPU and memory are performed in an asynchronous manner. Having asserted address, data, and strobe signals, the MPU inserts wait states until it receives the data transfer acknowledge signal, and then terminates the transfer. A detailed description of the data transfer protocol can be found in the Motorola MC68000 Data Sheet in Appendix A.

On the MVME101 monoboard computer, data to and from the on-board RAM is transferred without inserting wait states. For read operations from the on-board ROM, the configuration of jumper area K4 determines the number of wait states inserted by the MPU. The jumper must be positioned in accordance with the access time requirements of the installed memory devices. Paragraph 2.12 specifies the on-board memory timing. The configuration of jumper area K4 is described in Paragraph 3.4.9.

2.4. INPUT/OUTPUT-DEVICES

The following input/output-devices are provided on the MVME101 monoboard computer:

* two programmable serial communication interfaces
* a programmable parallel peripheral interface adapter
* a programmable triple timer module

The serial ports are RS232C standard compatible, may be configured as data set or data terminal, and are available at two 25-pole connectors on the front panel. The peripheral I/O signals of the parallel interface adapter and of the timer module are fed to the lower rear DIN 41612 64-pin connector.

2.4.1. Local I/O Access

All on-board I/O-devices, including the Module Control and Status Registers, are memory-mapped and occupy a 4 Kilobyte address segment. This segment may be placed anywhere in a 2 Megabyte address range (000000 - 0FFFFF and F00000 - FFFFFF) by programming the address decoder PROM according to the desired memory map. Paragraph 2.7 gives a detailed description of the Address Decoder.

Data transfers between MPU and on-board I/O-devices are performed in a synchronous manner. When the address decoder detects an address in the local I/O address segment, it asserts the valid peripheral address signal VPA*. This causes the MPU to terminate the current cycle after internal synchronization with the peripheral clock signal E. A detailed description of the synchronous data transfer protocol can be found in the Motorola MC68000 Data Sheet in Appendix A.

2.4.2. Enhanced Programmable Communication Interfaces

2.4.2.1. General Information

On the MVME101 monoboard computer two serial I/O-channels are installed, each of them controlled by a Motorola MC68661C Enhanced Programmable Communication Interface (EPCI). The EPCIs support several synchronous and asynchronous protocols in full or half duplex mode, and software selectable baud rates ranging from 50 to 19200 baud. Both ports are RS232C standard compatible, may be configured as data set or data terminal, and are available at 25-pole connectors on the front panel.

2.4.2.2. Features

Features, common to synchronous and asynchronous operation:

* 5 to 8 bit characters
* odd, even or no parity
* local or remote maintenance loop back mode
* 16 programmable baud rates
* double buffered transmitter and receiver
* dynamic character length switching
* half or full duplex operation

Additional features in synchronous operation:

* internal or external character synchronization
* transparent or non-transparent mode
* transparent mode DLE stuffing and detection
* single or double SYN operation
* automatic SYN or DLE-SYN insertion
* SYN, DLE, and DLE-SYN stripping

Additional features in asynchronous operation:

* parity, overrun and framing error detection
* line break detection and generation
* false start bit detection
* automatic serial echo mode

2.4.2.3. EPCI Device Description

A detailed description of the Enhanced Peripheral Communications Interface is given in the Motorola MC68661 Data Sheet in Appendix B.

2.4.2.4. Hardware Configuration

Both serial ports may be configured independently as data terminal or as data set on the jumper areas K7 and K15. The EPCI input CTS* can either be constantly enabled or shortened with the input DSR* on the jumper areas K9 and K10. The same jumper areas are used to connect the EPCI outputs TXRDY* and RXRDY* with the interrupt handler. Paragraph 3.4 includes detailed instructions how to configure the jumper areas for the various modes of operation.

2.4.2.5. Programming Information

Prior to initiating data communications, the EPCI registers must be loaded with a set of mode and command bytes. Detailed programming instructions are given in the Motorola MC68661 Data Sheet in Appendix B. The addresses of the EPCI registers are listed in Paragraph 2.7.

The serial data input of SP1 can be monitored through the Module Status Register. This feature supports the automatic detection of a terminal's baud rate: After hitting a specified character on the keyboard, the width of the first serial data bit is measured with the Programmable Timer Module. The result then is compared with a list of values in a lookup table to determine the transmitter's baud rate. (The automatic baud rate detection in MVME101bug is implemented in this way.)

2.4.3. Peripheral Interface Adapter

2.4.3.1. General Information

The MC6821 Peripheral Interface Adapter (PIA) provides the universal means of interfacing peripheral equipment to the MVME101 monoboard computer. The PIA can interface the MPU to peripherals through two 8-bit bidirectional peripheral data buses and four control lines.

2.4.3.2. Features

* two bidirectional 8-bit buses for interface to peripherals
* each peripheral line individually programmable as input or output
* four individually controlled interrupt input lines; two usable as peripheral control outputs
* handshake control logic for input and output peripheral operation
* high-impedance 3-state and direct transistor drive peripheral lines
* program controlled interrupt and interrupt disable capability
* CMOS drive capability on side A peripheral lines
* two TTL drive capability on all A and B side buffers

2.4.3.3. PIA Device Description

A detailed description of the Peripheral Interface Adapter is given in the Motorola MC6821 Data Sheet in Appendix C.

2.4.3.4. Hardware Configuration

All peripheral data and control lines are fed to the DIN 41612 C 96 rear connector P2. A description of the input/output signals is given in Table 2.1. Their locations at P2 are shown in Table 2.2.

Note that the peripheral input/output lines are not buffered between the PIA and the connector P2. Therefore, the electrical characteristics of the signals at P2 are equivalent with the values given in the MC6821 Data Sheet.

The interrupt outputs of the PIA may be wired to one of the Auto-Vectorized Interrupt Request lines on the jumper area K6. Paragraph 3.4.4 describes the configuration of K6.

2.4.3.5. Programming Information

The functional configuration of the PIA is programmed by the MPU during system initialization. Each of the peripheral data lines can be programmed to act as an input or output, and each of the four control/interrupt lines may be programmed for one of several control modes. Detailed programming instructions are given in the Motorola MC6821 Data Sheet in Appendix C. The addresses of the PIA registers are listed in Paragraph 2.7.

2.4.4. Programmable Timer Module

2.4.4.1. General Information

The MC6840 Programmable Timer Module (PTM) contains three 16-bit binary counters, three corresponding control registers, and a status register. The counters are under software control and may be programmed to generate module interrupts and/or output signals. The PTM can be used for frequency measurements, event counting, interval measuring, and similar tasks. It can generate square waves, gated delay signals, and single pulses of controlled or modulated duration.

2.4.4.2. Features

* selectable prescaler on timer 3
* programmable interrupt output to MPU
* readable down counter indicates counts to go to time-out
* selectable gating for frequency or pulse-width comparison
* three asynchronous external clock and gate/trigger inputs internally synchronized
* three maskable outputs
* peripheral inputs/outputs fully TTL compatible

2.4.4.3. PTM Device Description

A detailed description of the Programmable Timer Module is given in the Motorola MC6840 Data Sheet in Appendix D.

2.4.4.4. Hardware Configuration

All peripheral clock,gate and output lines are fed to the DIN 41612 C 64 rear connector P2. A description of the input/output signals is given in Table 2.1. Their locations at P2 are shown in Table 2.2.

Note that the peripheral input/output lines are not buffered between the PTM and the connector P2. Therefore, the electrical characteristics of the signals at P2 are equivalent with the values given in the MC6840 Data Sheet.

The gate inputs of the counters can be constantly enabled by setting jumpers on jumper area K16. Also, K16 provides the hardware connections for cascading the PTM's counters, for real time counting, MPU cycle counting, or VMEbus cycle counting. Paragraph 3.4.7 gives a detailed description of jumper area K16.

The interrupt output of the PTM may be wired to one of the Auto-Vectorized Interrupt Request lines on the jumper area K6. Paragraph 3.4.4 describes the configuration of K6.

2.4.4.5. Programming Information

The functional configuration of the PTM is programmed by the MPU during system initialization. Detailed programming instructions are given in the Motorola MC6840 Data Sheet in Appendix D. The addresses of the PTM registers are listed in Paragraph 2.7.

2.4.5. Connector P2 Signals

Table 2.1 identifies the peripheral input/output signals by signal mnemonic, connector pin number and signal characteristics, Table 2.2 shows the signal locations at connector P2.

Table 2.1: Connector P2 Signal Description

SIGNAL	PIN NO.	SIGNAL DESCRIPTION
PA0...PA7	C12...C19	PIA SECTION A PERIPHERAL DATA Eight TTL compatible peripheral data lines. Each line can be programmed to act as an output or input by setting the corresponding bit in the PIA Data Direction Register A to "0" or "1".

Table 2.1: Connector P2 Signal Description (cont'd)

SIGNAL	PIN NO.	SIGNAL DESCRIPTION
CA1	C21	PIA SECTION A INTERRUPT A TTL compatible clock input line that sets the interrupt flag of the PIA Control Register A. The active transition of this signal is programmed by the PIA Control Register A.
CA2	C20	PIA SECTION A PERIPHERAL CONTROL A TTL compatible line that can be programmed by the PIA Control Register A to act as a peripheral control output or an interrupt input.
PB0...PB7	C4...C11	PIA SECTION B PERIPHERAL DATA Eight TTL compatible peripheral data lines. Each line can be programmed to act as an output or high impedance input by setting the corresponding bit in the PIA Data Direction Register B to "0" or "1".
CB1	C3	PIA SECTION B INTERRUPT A TTL compatible clock input line that sets the interrupt flag of the PIA Control Register B. The active transition of this signal is programmed by the PIA Control Register B.
CB2	C2	PIA SECTION B PERIPHERAL CONTROL A TTL compatible line that can be programmed by the PIA Control Register B to act as a peripheral control output or a high impedance interrupt input.
C1*...C3*	C23, C26 C29	PTM CLOCK INPUTS 1...3 Three active low TTL compatible high impedance clock inputs that can be used to decrement Timers 1...3, respectively.
G1*...G3*	C25, C28, C31	PTM GATE INPUTS 1...3 Three active low TTL compatible high impedance inputs that can be programmed to act as triggers or clock gating functions to Timers 1...3, respectively.
O1...O3	C24, C27 C30	PTM OUTPUTS 1...3 Three active high TTL compatible outputs of Timers 1..3, respectively. The output waveform is defined by the contents of the PTM Control Registers 1...3, respectively.

Table 2.1: Connector P2 Signal Description (cont'd)

SIGNAL	PIN NO.	SIGNAL DESCRIPTION
+5V	C1, C22, C32	+ 5 VOLTS + 5 Volts power supply output
GND	A1...A32	GROUND Power supply ground lines

Table 2.2: Connector P2 Signal Locations

PIN NO.	ROW A SIGNALS	ROW C SIGNALS	PIN NO.
1	GND	+5V	1
2	GND	CB2	2
3	GND	CB1	3
4	GND	PB7	4
5	GND	PB6	5
6	GND	PB5	6
7	GND	PB4	7
8	GND	PB3	8
9	GND	PB2	9
10	GND	PB1	10
11	GND	PB0	11
12	GND	PA7	12
13	GND	PA6	13
14	GND	PA5	14
15	GND	PA4	15
16	GND	PA3	16
17	GND	PA2	17
18	GND	PA1	18
19	GND	PA0	19
20	GND	CA2	20
21	GND	CA1	21
22	GND	+5V	22
23	GND	C3*	23
24	GND	O3	24
25	GND	G3*	25
26	GND	C2*	26
27	GND	O2	27
28	GND	G2*	28
29	GND	C1*	29
30	GND	O1	30
31	GND	G1*	31
32	GND	+5V	32

2.5. MODULE STATUS REGISTER

Through the Module Status Register (MSR) the current status of several on-board signals and VMEbus lines can be monitored. By that the MPU can detect certain system conditions and branch to the appropriate servicing routines.

The MSR appears as an 8-bit register in the on-board I/O-devices address segment. Paragraph 2.7 gives more detailed addressing information.

Figure 2.3 shows how the MSR is interconnected with VMEbus signals and with other functional blocks on the MVME101. During a read operation, the outputs of the MSR are enabled and put on the lower order data lines D00-D07. The outputs MSR0-MSR5 represent the current states of the signals ACFAIL*, SYSFAIL*, ABORT*, BCLR*, BAV* and PCI1RXD*. MSR6 and MSR7 are Flip-Flop outputs which are set to 0 when a bus request time-out (MSR6) or a data transfer time-out (MSR7) occured. Any write operation to the MSR clears MSR6 and MSR7 to 1, regardless of the data transferred.

All signals represented in the MSR are active low. A bit value of 0 indicates that the corresponding signal is asserted, a value of 1 means that it is negated.

Figure 2.3: Module Status Register

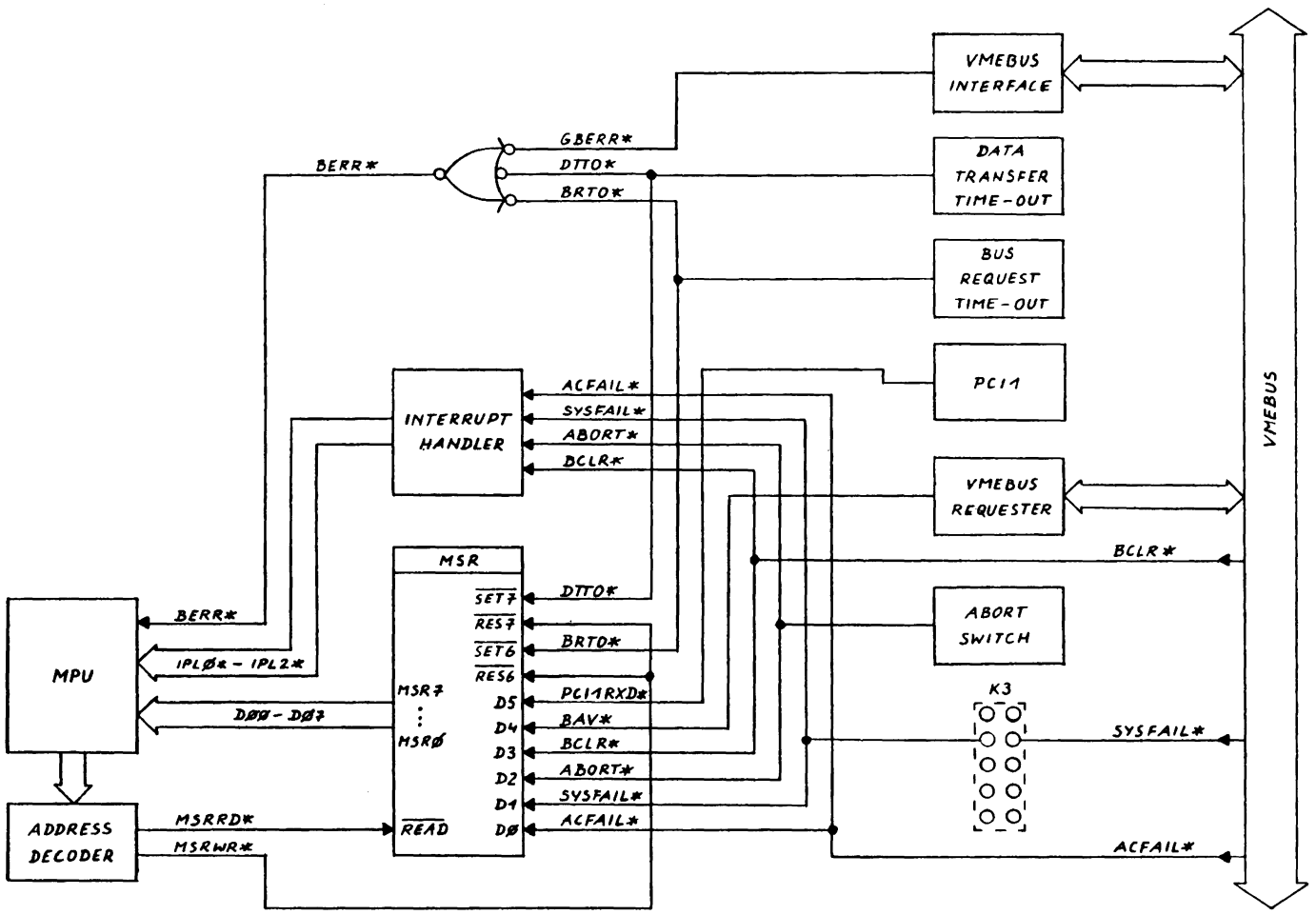


Table 2.3 shows the allocation of signals in the MSR and explaines the information contained in each bit.

Table 2.3: Module Status Register

BIT	SIGNAL	DESCRIPTION
MSR7	DTTO*	MSR7 = 0: A Data Transfer Time-Out occured. MSR7 = 1: A Data Transfer Time-Out did not occur. Note: Paragraph 2.9.4 describes the Data Transfer Time Out counter in detail.
MSR6	BRTO*	MSR6 = 0: A Bus Request Time-Out occured. MSR6 = 1: A Bus Request Time-Out did not occur. Note: Paragraph 2.9.4 describes the Bus Request Time Out counter in detail.
MSR5	PCI1RXD*	MSR5 reflects the current state of the data input of Serial Port 1. Note: Paragraph 2.4.2.5 describes how MSR5 can be used for automatic baud rate detection.
MSR4	BAV*	MSR4 = 0: The VMEbus is available. MSR4 = 1: The VMEbus is not available. Note: Paragraph 2.8.2 describes how the BAV* signal is used for bus arbitration.
MSR3	BCLR*	MSR3 = 0: The VMEbus signal BCLR* is asserted. MSR3 = 1: The VMEbus signal BCLR* is negated. Note: Paragraph 2.8.2 describes how the BCLR* signal is used for bus arbitration.
MSR2	ABORT*	MSR2 = 0: The ABORT switch is pressed. MSR2 = 1: The ABORT switch is released. Note: Paragraph 2.11.1 describes the ABORT function.
MSR1	SYSFAIL*	MSR1 = 0: The VMEbus signal SYSFAIL* is asserted. MSR1 = 1: The VMEbus signal SYSFAIL* is negated. Note: Paragraph 2.11.2 describes the SYSFAIL function.
MSR0	ACFAIL*	MSR0 = 0: The VMEbus signal ACFAIL* is asserted. MSR0 = 1: The VMEbus signal ACFAIL* is negated. Note: Paragraph 2.11.1 describes the ACFAIL function.

2.6. MODULE CONTROL REGISTER

The Module Control Register (MCR) contains eight bits for controlling various module functions and the hexadecimal STATUS display. To support single bit manipulations, the data byte in the MCR can be both written and read.

The MCR appears as an 8-bit register in the on-board I/O-devices address segment. Paragraph 2.7 gives more detailed addressing information.

Figure 2.4 shows how the MCR is interconnected with other functional blocks on the MVME101. During a write operation, the bit pattern on the lower order data lines D00-D07 is stored in the MCR. The four bits MCR0-MCR3 represent the hex number to be shown on the STATUS display in binary data format. In addition, when MCR0-MCR3 all are set to 1, i.e. when the hex number F is displayed, the VMEbus signal SYSFAIL* is asserted. MCR4 is used to switch the display on and off. MCR5 controls the bus block transfer mode of the VMEbus Requester. The bits MCR6 and MCR7 are used to enable or disable the time-out counters.

After a system reset all bits in the MCR are cleared to 0. Also, when the MPU has halted due to a double bus error, the MCR is cleared, and both decimal points on the STATUS display are lit.

All signals controlled by the MCR are active high. A bit value of 1 causes the assertion of the corresponding signal, a value of 0 causes its negation.

Figure 2.4: Module Control Register

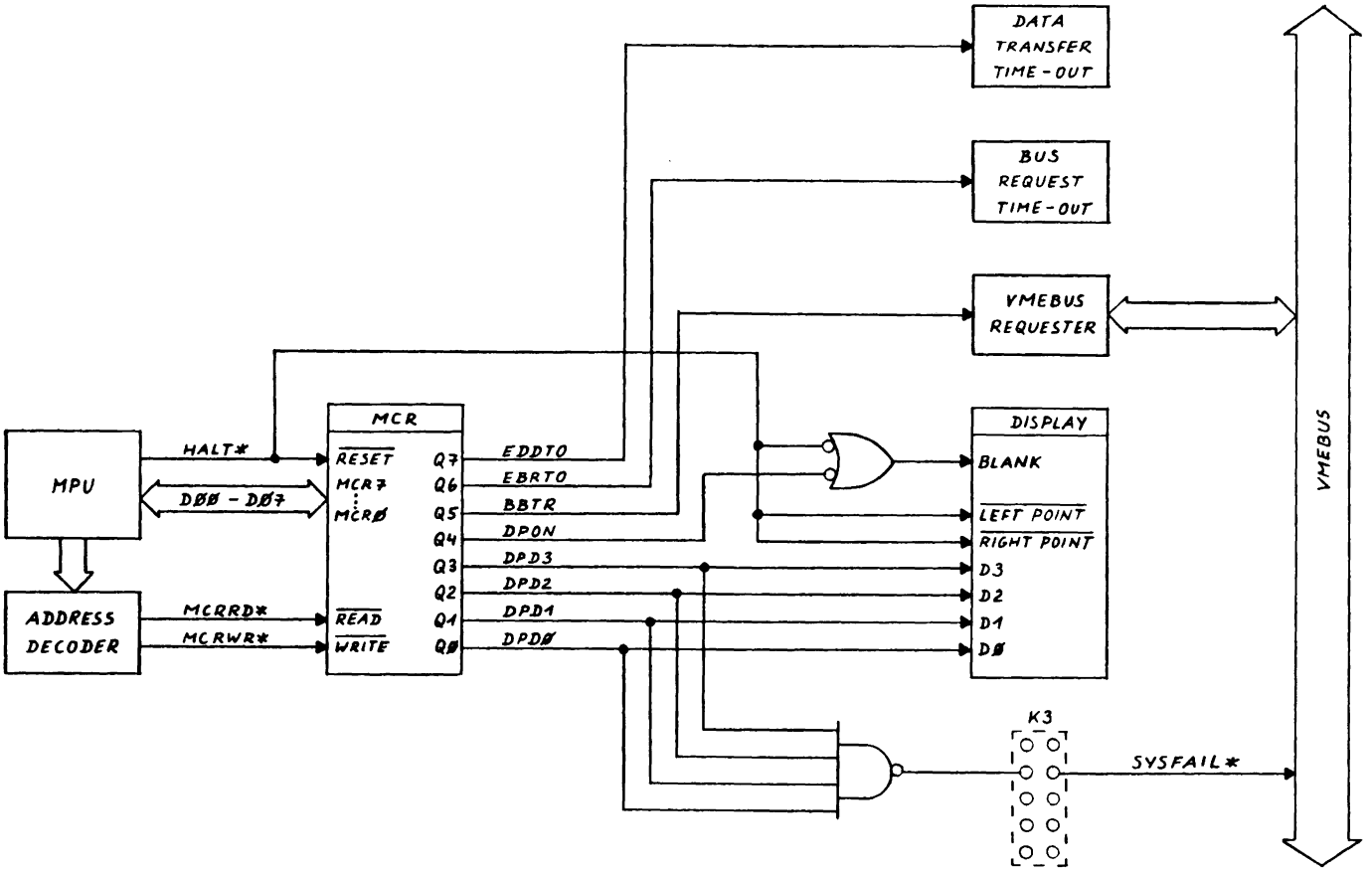


Table 2.4 shows the allocation of signals in the MCR and explaines the function of each bit.

Table 2.4: Module Control Register

BIT	SIGNAL	DESCRIPTION
MCR7	EDDTO	MCR7 = 0: Disable Data Transfer Time-Out counter. MCR7 = 1: Enable Data Transfer Time-Out counter. Note: Paragraph 2.9.4 describes the Data Transfer Time-Out counter in detail.
MCR6	EBRTO	MCR6 = 0: Disable Bus Request Time-Out counter. MCR6 = 1: Enable Bus Request Time-Out counter. Note: Paragraph 2.9.4 describes the Bus Request Time-Out counter in detail.
MCR5	BBTR	MCR5 = 0: Negate Bus Block Transfer Request. MCR5 = 1: Assert Bus Block Transfer Request. Note: Paragraph 2.8.2 describes the function of the BBTR signal.
MCR4	SDON	MCR4 = 0: Blank STATUS Display. MCR4 = 1: Lit STATUS Display. Note: The STATUS Display is also blanked after system reset and when the MPU has halted.
MCR3 MCR2 MCR1 MCR0	SDD3 SDD2 SDD1 SDD0	SDD3,SDD2,SDD1,SDD0 = 0,0,0,0: Display "0" SDD3,SDD2,SDD1,SDD0 = 0,0,0,1: Display "1" : : : : SDD3,SDD2,SDD1,SDD0 = 1,1,1,0: Display "E" SDD3,SDD2,SDD1,SDD0 = 1,1,1,1: Display "F" and assert SYSFAIL* The bits SDD0-SDD3 are the binary equivalent of the hexadecimal number on the STATUS display. Also, these bits are used to assert the SYSFAIL* signal on the VMEbus by setting them all to 1, i.e. by writing "F" into the STATUS display. Note: Paragraph 2.11.2 describes the SYSFAIL function.

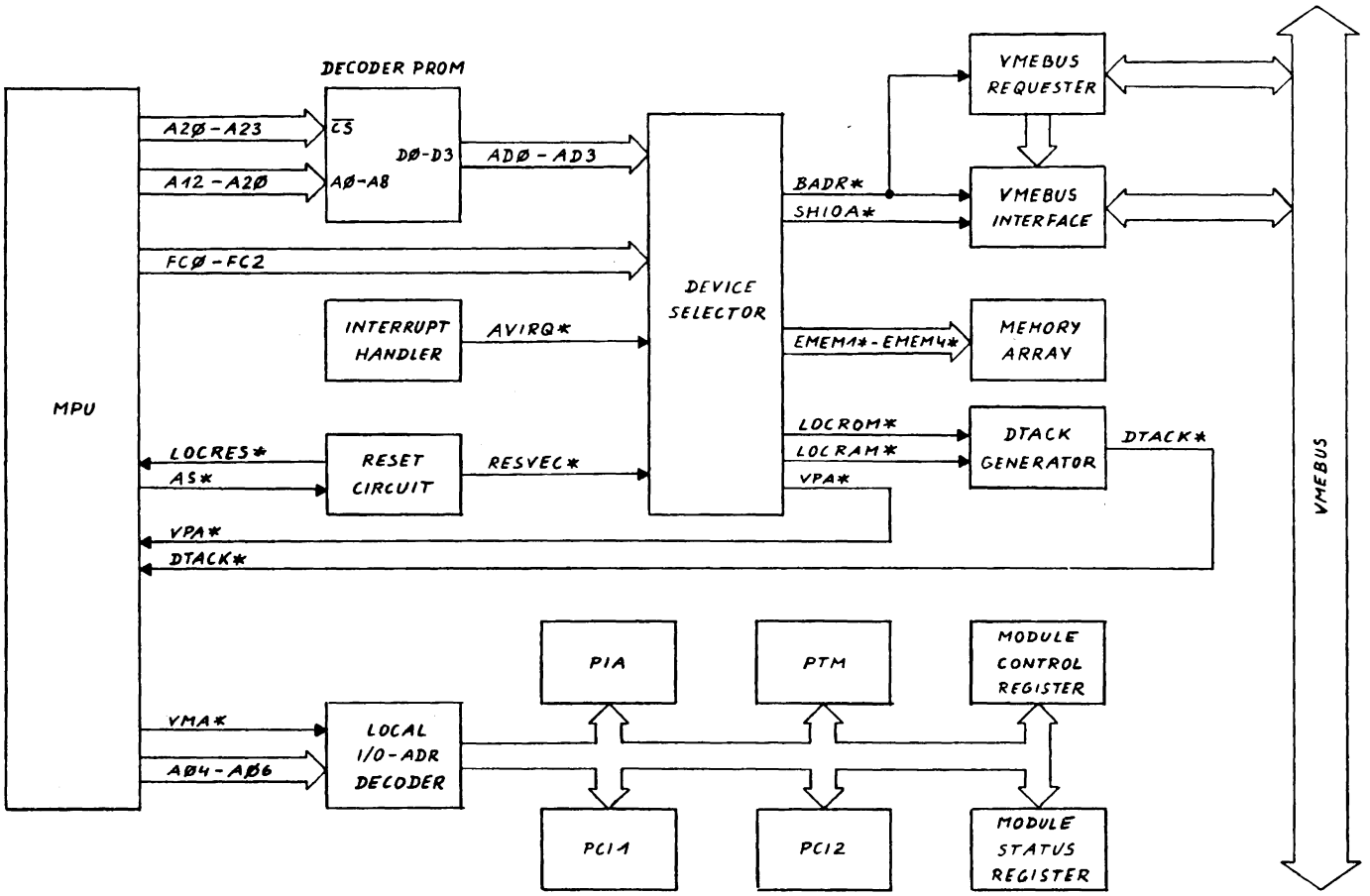
2.7. ADDRESS DECODER

2.7.1. Circuit Description

The Address Decoder logic is responsible for selecting the various on-board devices or the VMEbus Interface, depending on the address asserted by the MPU. Also, it contains circuitry to generate the data transfer handshake signals for on-board operations.

Figure 2.5 shows how the Address Decoder is interconnected with on-board devices and other functional blocks on the MVME101. The data contained in the Decoder PROM determines the address map configuration and assigns each address either to one of the on-board devices or to the VMEbus. The Device Selector receives signals from the Decoder PROM, the MPU, the Interrupt Handler and the Reset Circuit, and determines the current cycle to be either a VMEbus data transfer, a data transfer to or from one of the on-board ROM or RAM devices, an access to the on-board I/O-devices, a VMEbus interrupt acknowledge cycle, an auto-vectorized interrupt acknowledge cycle, or a reset vector fetch. For VMEbus operations, the Device Selector enables the VMEbus Requester and the VMEbus Interface. When on-board memory is accessed, the Device Selector enables the addressed memory pair and causes the DTACK Generator to assert the data transfer acknowledge signal. When one of the on-board I/O-devices is accessed, or in case of an auto-vectorized interrupt acknowledge cycle, the Device Selector asserts the VPA* signal. After receiving VPA*, the MPU synchronizes internally with the peripheral clock signal and then asserts VMA*. This enables the Local I/O-Address Decoder, which selects the addressed I/O-device.

Figure 2.5: Address Decoder



2.7.2. Address Map Configuration

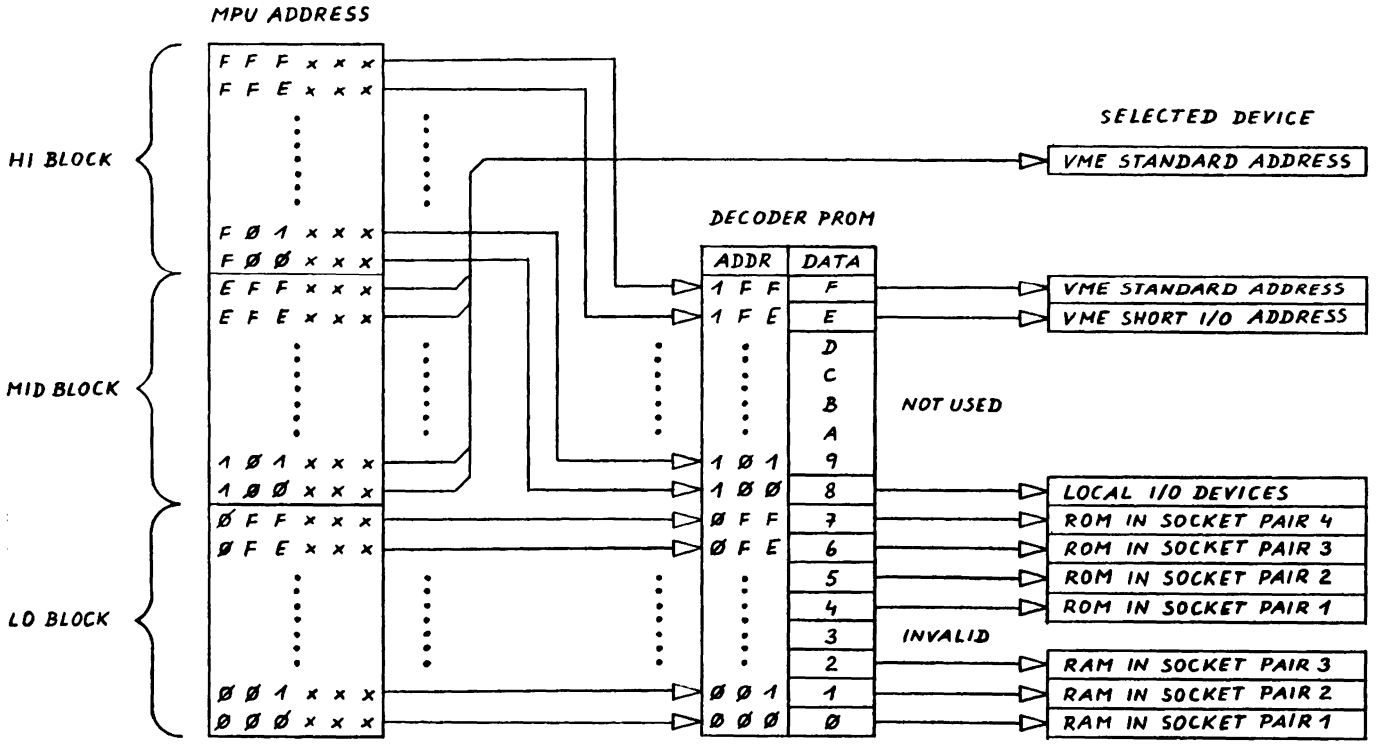
The Address Decoder logic first devides the 16M byte address map of the MPU into three blocks, as shown in Figure 2.6. The Lo Block covers the lower 1M bytes address range (000000-0FFFFF), the Mid Block comprises the following 14M bytes (100000-EFFFFF), and the Hi Block covers the uppermost 1M bytes address range (F00000-FFFFFF).

All addresses in the Mid Block are supposed to be off-board. When the MPU asserts an address in the range 100000-EFFFFF, the Device Selector initiates a VMEbus data transfer by enabling the VMEbus Requester and the VMEbus Interface.

The Lo Block and the Hi Block are further subdivided into address segments of 4K bytes. Each of these 512 segments corresponds to one location of the Decoder PROM. This PROM, organized as 512 x 4 bits, assigns each address segment either to one of the on-board memory pairs, to the on-board I/O-devices, or to the VMEbus. When on-board memory is addressed, the PROM also determines whether the ROM or the RAM access time is used by the DTACK generator. For transferring data to or from global I/O-modules on the VMEbus, the address modifier code for Short I/O Address may be specified for a 64K address field.

Figure 2.6 illustrates how the MPU address map is divided into blocks and segments, and how the segments are represented in the Decoder PROM. The figure also specifies the data to be programmed in the PROM for appointing the devices to the address segments.

Figure 2.6: Address Map Configuration



The address decoding scheme of the MVME101 allows the user to place each memory pair and the on-board I/O-devices anywhere in the Lo Block or the Hi Block of the memory map. All address segments that are not occupied by on-board devices can be defined to be either standard addresses or short I/O addresses on the VMEbus. By that the user may create independent areas for ROM, RAM and I/O-devices, with contiguous on-board and off-board allocation for each area.

The MVME101 module is delivered with a Decoder PROM which contains the address map configuration shown in Table 2.5. This address map is designed to accomodate the MVME101bug Debug Package firmware and in addition 10K bytes RAM for user programs. The addresses 000000-002FFF are assigned to RAM in the memory socket pairs 1-3, where the addresses 000000-0003FF are occupied by the MPU exception vector table, and the addresses 000400-0007FF are used as a temporary data storage area for the MVME101bug parameters. The addresses 000800-002FFF are available for user programs and data. The addresses F00000-F03FFF are assigned to ROM in memory socket pair 4, which may be the MVME101bug package or, after the dubugging phase, any user-provided firmware-resident program. The on-board I/O-devices are located in the address segment FE0000-FE0FFF. The upper 64K bytes in the address map are dedicated to I/O-devices on the VMEbus which are accessed using Short I/O Address encoding in the address modifiers. All other addresses in the map are decoded as VMEbus Standard Addresses for access to off-board memory or memory-mapped devices.

The registers of the on-board I/O-devices occupy a 4K bytes segment in the address map. The register addresses are listed in Table 2.6. As the data width of all I/O-devices is 8 bits, their registers are located on odd addresses, and data transfers to and from the MPU are performed via the lower order data lines D00-D07. The even address locations in the local I/O address segment are redundant and should not be accessed.

The addresses of the on-board I/O-registers are not fully decoded. The upper order 3 digits of the 6-digit address indicate the 4K bytes address segment that is reserved for the local I/O-devices. Then the address lines A04-A06 are decoded to determine the specific device to be selected. As the Local I/O-Address Decoder does not care about the address lines A07-A11, the I/O-registers appear virtually multiplied in address increments of hex 80 in the local I/O-address segment. Thus the I/O-register listing in Table 2.6 can be regarded to be one of 32 possible sets of addresses.

If the original address map configuration, as described above, does not meet the user's requirements, he may specifv any other configuration, and program the Decoder PROM accordingly. A detailed step-by-step description of this procedure is given in Paragraph 3.4.10.

Table 2.5: Original Address Map

ADDRESS	CONTENTS	SELECTED DEVICES
FFFFFF : FF0000	VMEbus Short I/O Addresses	Global I/O-devices
FEFFFF : FE1000	VMEbus Standard Addresses	Global Memory or Memory-mapped Devices
FE0FFF : FE0000	On-board I/O Registers (Only odd addresses used)	Local I/O-devices
FDFFFF : F04000	VMEbus Standard Addresses	Global Memory or Memory-mapped Devices
F03FFF : F00000	MVME101bug Debug Package or User-provided Program	2 x 8K bytes Local ROM in Memory Socket Pair 4
EFFFFF : : : : : : : : : : : 003000	VMEbus Standard Addresses	Global Memory or Memory-mapped Devices
002FFF : 002000	User Program/Data	2 x 2K bytes Local RAM in Memory Socket Pair 3
001FFF : 001000	User Program/Data	2 x 2K bytes Local RAM in Memory Socket Pair 2
000FFF : 000800	User Program/Data	2 x 2K bytes Local RAM in Memory Socket Pair 1
0007FF : 000400	MVME101bug Data/Stack	
0003FF : 000000	MPU Exception Vectors	

Table 2.6: Original I/O-Register Address Map

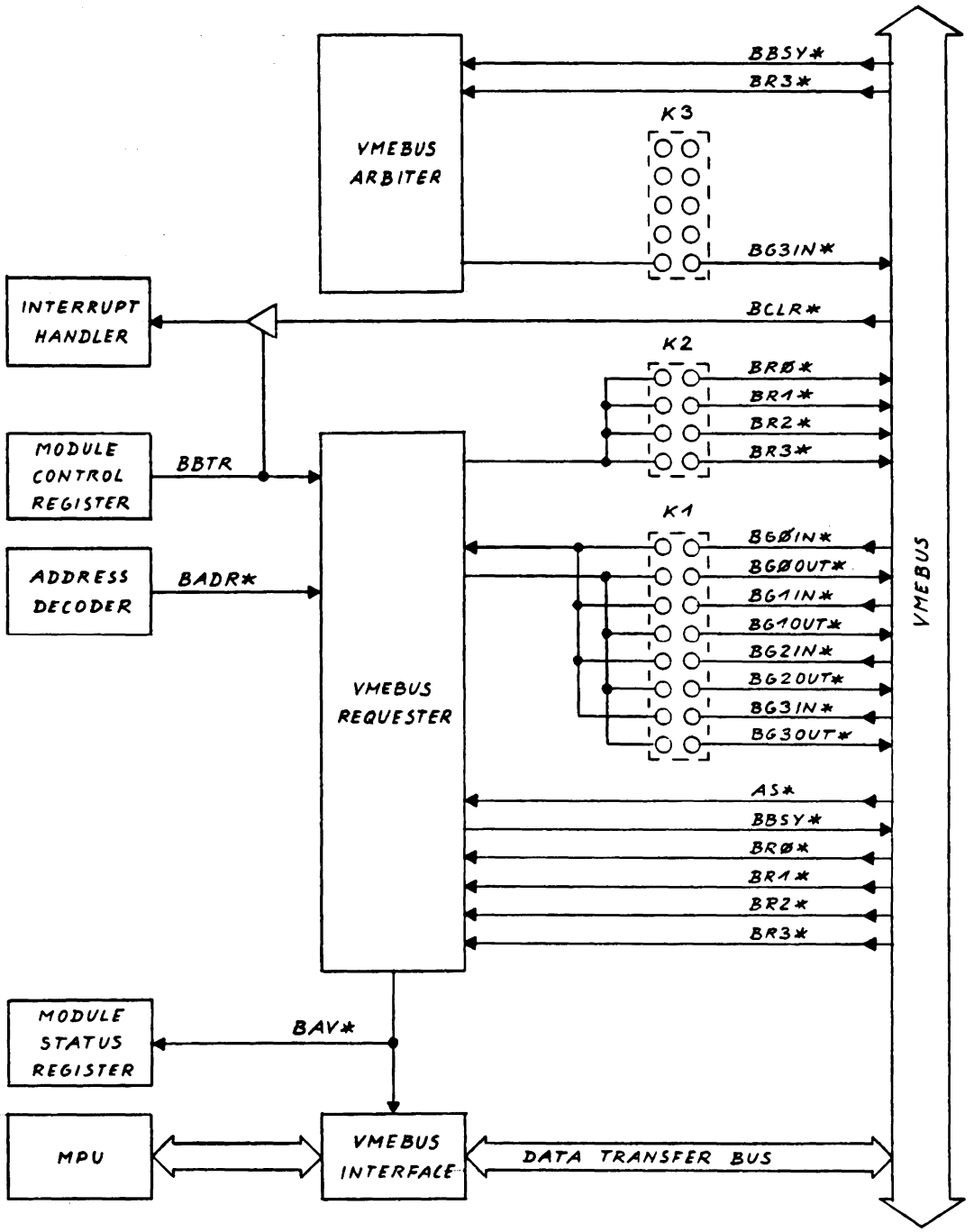
DEVICE	ADDRESS	MODE	REGISTER
MCR	FE00F1	r/w	Module Control Register
MSR	FE00E1	r/w	Module Status Register
PTM	FE00DF	read	LSB buffer register
	FE00DF	write	Timer #3 latches
	FE00DD	read	Timer #3 counter
	FE00DD	write	MSB buffer register
	FE00DB	read	LSB buffer register
	FE00DB	write	Timer #2 latches
	FE00D9	read	Timer #2 counter
	FE00D9	write	MSB buffer register
	FE00D7	read	LSB buffer register
	FE00D7	write	Timer #1 latches
	FE00D5	read	Timer #1 counter
	FE00D5	write	MSB buffer register
	FE00D3	read	status register
	FE00D3	write	control register #2
	FE00D1	read	no operation
	FE00D1	write	CR20 = 1: control register #1
	FE00D1	write	CR20 = 0: control register #3
PIA	FE00C7	r/w	Section B control register
	FE00C5	r/w	CRB-2 = 1: Section B peripheral register
	FE00C5	r/w	CRB-2 = 0: Section B data direction register
	FE00C3	r/w	Section A control register
	FE00C1	r/w	CRA-2 = 1: Section A peripheral register
	FE00C1	r/w	CRA-2 = 0: Section A data direction register
PCI2	FE00B7	r/w	command register
	FE00B5	r/w	mode register #1 / mode register #2
	FE00B3	read	status register
	FE00B3	write	SYN1 register / SYN2 register / DLE register
	FE00B1	read	receive holding register
	FE00B1	write	transmit holding register
PCI1	FE00A7	r/w	command register
	FE00A5	r/w	mode register #1 / mode register #2
	FE00A3	read	status register
	FE00A3	write	SYN1 register / SYN2 register / DLE register
	FE00A1	read	receive holding register
	FE00A1	write	transmit holding register

2.8. VMEbus ARBITER AND REQUESTER

Bus arbitration is a technique to request, be granted, and acknowledge bus mastership in a system, where multiple master-type modules share common resources on the bus. For that purpose the MVME101 monoboard computer contains a VMEbus Arbiter and a VMEbus Requester. Most of the logic is included in the BAR101 Bus Arbiter/Requester device, which is described in Appendix E.

On the MVME101, all on-board devices are interconnected by a local bus which is connected to the VME data transfer bus only when off-board devices are to be accessed. This feature allows on-board processing at full speed, while another module transfers data on the VMEbus.

Figure 2.7: VMEbus Arbiter and Requester



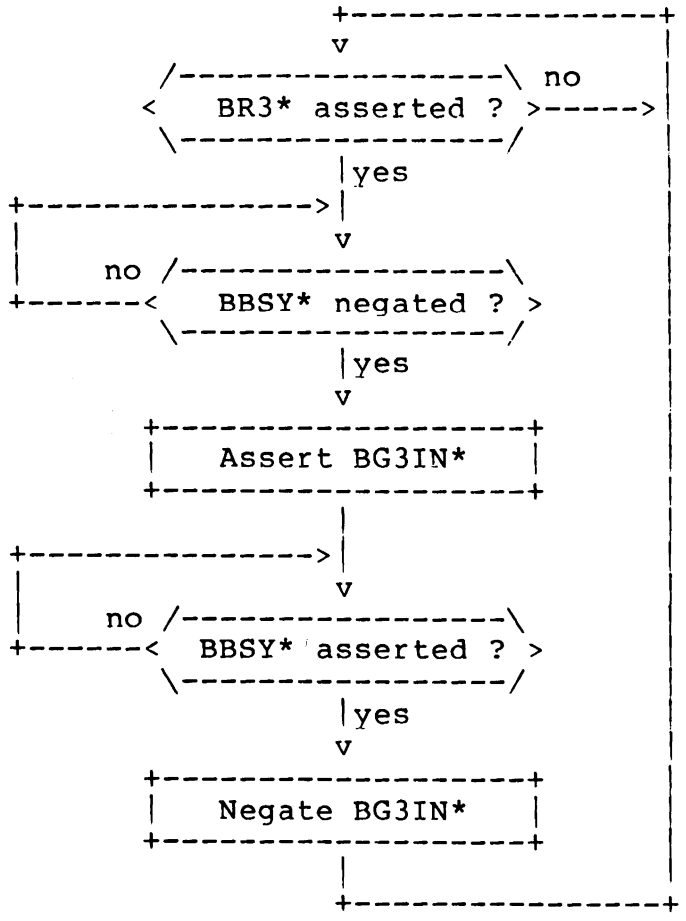
2.8.1. VMEbus Arbiter

For use as the System Controller in a VMEbus System, the MVME101 module contains an option ONE single level arbiter which arbitrates bus requests on level 3. Figure 2.7 shows the interconnections of the bus signals with the Arbiter, and the flow chart in Figure 2.8 illustrates its operation. When the VMEbus Arbiter receives a bus request at the input BR3*, it monitors the BBSY* line. A low level on BBSY* indicates that another master module is currently using the bus, and the bus request is made pending. When BBSY* is high, the VMEbus Arbiter grants the request by asserting BG3IN*. This signal is propagated along the bus grant daisy-chain through all modules participating in the bus arbitration until the first bus requester is reached which has asserted BR3*. This requester acknowledges the bus grant by asserting BBSY* and negating BR3*. Upon detecting that, the VMEbus Arbiter negates BG3IN* and is ready for another arbitration sequence.

When the MVME101 is used as the System Controller, it must be located in slot 1 of the VMEbus backplane to ensure that the VMEbus Arbiter resides to the left of all bus requesters. In this configuration, the VMEbus Requester on the MVME101 is the first in the daisy-chain, and therefore has the highest priority.

When installed on lower bus priorities in a multi-processor system, the VMEbus Arbiter on the MVME101 must be disabled by removing the according jumper from jumper area K3, as described in Paragraph 3.4.2.

Figure 2.8: VMEbus Arbiter Operation Flow Chart



2.8.2. VMEbus Requester

The VMEbus Requester on the MVME101 is responsible for performing the following tasks:

- Assert a bus request when the MPU needs access to off-board devices,
- Acquire bus mastership when the bus request is granted,
- Release the bus upon another request when it is no longer needed,
- Propagate not requested bus grants to the next bus requester.

Each of these functions is described in detail in the following paragraphs. Figure 2.7 shows how the VMEbus Requester is interconnected with the bus signals and with other functional blocks on the MVME101. The flow chart in Figure 2.9 illustrates the operation sequence.

The VMEbus Requester can be configured to operate on anyone of the four bus arbitration levels. This is done by setting the appropriate jumpers on the jumper areas K1 and K2, as described in Paragraph 3.4.1.

2.8.2.1. Bus Request Assertion

There are two methods by which the MVME101 can request the VMEbus: The indirect, or software transparent method, and the direct method, by which a specific request can be programmed.

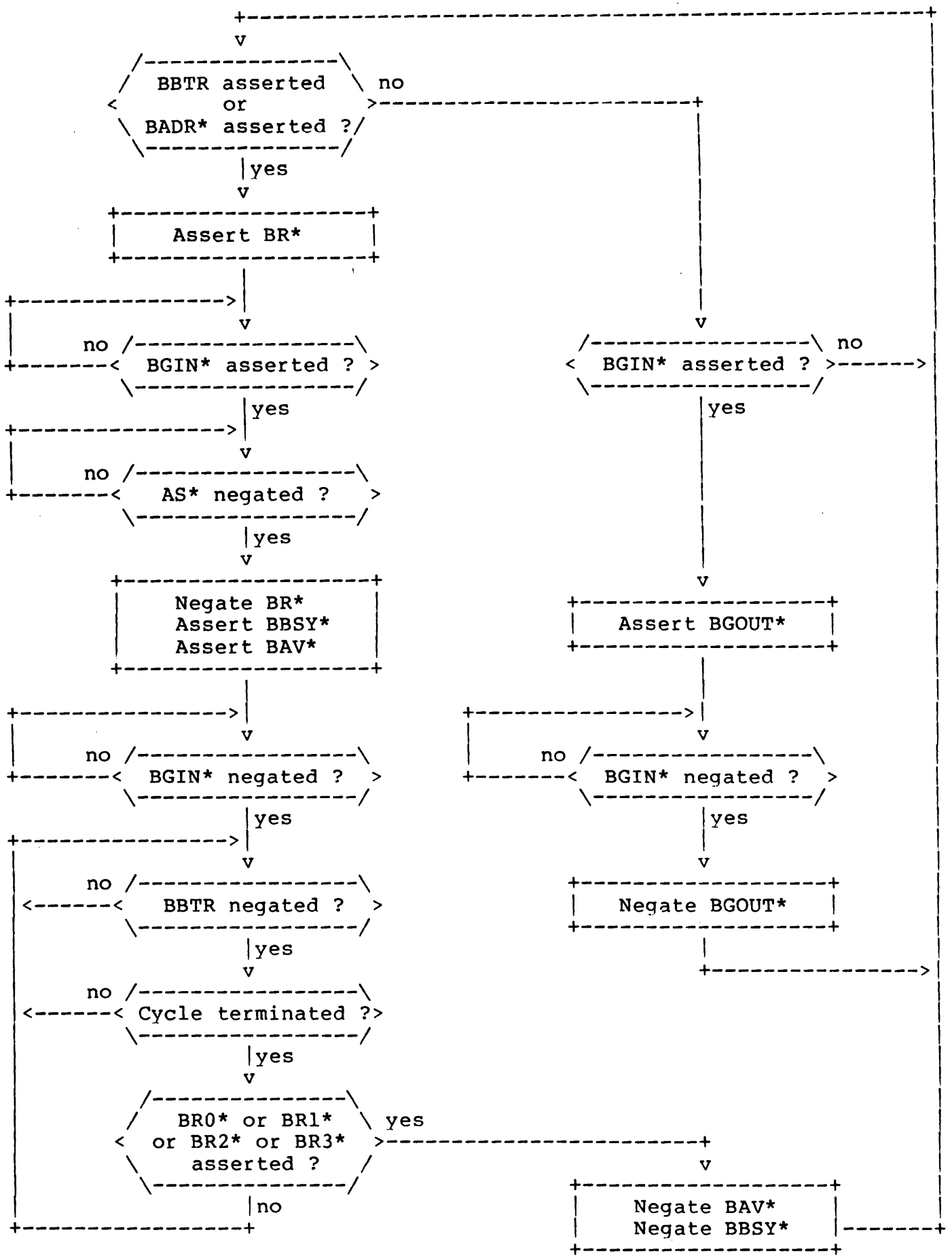
An indirect bus request is initiated by the Address Decoder. When the MPU starts either a VMEbus data transfer cycle, or a VMEbus interrupt vector fetch, the Address Decoder asserts the signal BADR*. If the MVME101 is not the current bus master, the VMEbus Requester then asserts BR* on the jumpered level. The MPU inserts wait states until the bus arbitration is performed and the addressed slave module acknowledges the data transfer.

In applications where a long idle state of the MPU is not acceptable, the Bus Request Time-Out counter can be enabled through the Module Control Register. This counter supervises the bus request and aborts the started MPU cycle if it is not acknowledged within 128 microseconds. The Bus Request Time-Out counter is described in Paragraph 2.9.4.

The direct bus request is initiated under program control by setting the Bus Block Transfer Request bit (BBTR) in the Module Control Register. This causes the VMEbus requester to assert BR* and, after being granted, to retain bus mastership as long as BBTR is set. This method protects routines against interruption by other bus requests, and therefore is useful for tasks such as data block transfers, system control, or emergency servicing.

Once having requested the bus, the VMEbus Requester keeps BR* asserted until it receives a bus grant on the same priority level, regardless of further transitions on the signals BBTR and BADR*. This is necessary to obey the bus arbitration protocol as specified for the VMEbus.

Figure 2.9: VMEbus Requester Operation Flow Chart



2.8.2.2. Bus Mastership Acquisition

When the VMEbus Requester has a bus request pending and it receives a bus grant on the same priority level, it acquires bus mastership. After the previous bus master has finished its last VMEbus cycle and negated AS*, the VMEbus Requester acknowledges the bus grant by negating BR* and asserting BBSY*, and it enables the VMEbus Interface.

The availability of the VMEbus can be checked by the MPU in the Module Status Register. This feature is useful in programs where the execution of on-board and off-board tasks does not require a fixed sequence. For example, the MPU can prepare an off-board task by setting the BBTR bit in the Module Control Register, and then execute on-board tasks. Now and then it tests the BAV* bit in the Module Status Register, and when it detects that the bus is available, it starts executing the off-board task.

2.8.2.3. Bus Release

As long as the BBTR bit in the Module Control Register is set, the MVME101 never releases the bus. However, in a system with a multilevel bus arbitration scheme, the VMEbus Arbiter can interrupt the current program with the BCLR* signal upon a higher level bus request. When the MVME101 operates as bus master in the block-transfer mode, the assertion of BCLR* causes a maskable auto-vectorized interrupt request at the MPU, and it depends on the executed software, whether and when the BBTR bit is cleared and the VMEbus is released.

The BBTR bit may be set or cleared at any time. If it is set although the MVME101 is already the bus master, it has no effect on the VMEbus Requester, but it protects the execution of the following task.

When BBTR is negated, the VMEbus Requester operates in the release-on-request mode. In this mode, it retains bus mastership until another module asserts a bus request on any of the four levels. When that happens, the VMEbus Requester waits until the MPU has terminated the current cycle, and then releases the bus by disabling the VMEbus Interface and negating BBSY*.

This release-on-request scheme provides maximum system efficiency, as unnecessary bus arbitration cycles are avoided.

2.8.2.4. Bus Grant Propagation

When the VMEbus Requester receives a bus grant on its priority level, and it has no bus request pending, the corresponding bus grant output signal is produced and propagated to the next bus requester. BGOUT* keeps asserted as long as BGIN* is low.

The BGIN* and BGOUT* lines which are not on the module's priority level are connected directly on the jumper area K1.

2.9. VMEbus INTERFACE

The VMEbus Interface provides the signal path between the local bus of the MVME101 computer and the VMEbus backplane. The interface complies with all requirements for the signal driver/receiver characteristics, and for the bus operation protocol timings, as specified in the VMEbus Specification Manual Rev.B. Any VMEmodule which is designed according to these specifications will run with the MVME101 Monoboard Computer without restrictions.

This chapter gives detailed functional descriptions of all VMEbus signals that are handled by the MVME101, and explains the available hardware options. The timing specifications for the VMEbus Interface are included in Paragraph 2.12.

2.9.1. VMEbus Signals

All VMEbus signals are available at the upper rear connector P1. Table 2.8 identifies all these signals by mnemonics, pin numbers at P1, and electrical characteristics, and it describes the signal functions on the MVME101. The abbreviations used in Table 2.8 are explained in Table 2.7. The locations of the VMEbus signals at connector P1 are shown in Table 2.9.

VMEbus signals that are not driven by the MVME101 module appear as being high at other modules on the bus, due to the termination resistors on the VMEbus backplane. Such signals are put in parantheses in the following tables.

For some VMEbus signals the user can choose whether the MVME101 handles or ignores them, by setting or removing jumpers. Such signals are termed optional signals.

Table 2.7: Symbol Definitions

SYMBOL	DEFINITION
TP	totem-pole bus driver output
TS	three-state bus driver output
OC	open-collector bus driver output
ST	schmitt-trigger bus receiver input with hysteresis
IOH	minimum high-level output current at 2.4 V
IOL	minimum low-level output current at 0.5 V
IOZH	maximum off-state output current at 2.7 V
IOZL	maximum off-state output current at 0.4 V
IIH	maximum high-level input current at 2.7 V
IIL	maximum low-level input current at 0.4 V

Note: The values for the input/output currents listed in Table 2.8 result in the sum of the driver-, receiver-, and pull-up-resistor-currents for each signal.

Table 2.8: VMEbus Signal Description

SIGNAL	PIN NO.	SIGNAL DESCRIPTION	ELEC. SPEC.
D00..D07 D08..D15	A1..A8 C1..C8	DATA BUS 16-bit TS-output/ST-input bidirectional data bus for transferring data to and from slave modules.	IOH -3 mA IOL 48 mA IOZH 20 uA IOZL -400 uA IIH 20 uA IIL -400 uA
A01..A07 A08..A23	A30..A24 C30..C15	ADDRESS BUS 23-bit TS-output address bus capable of addressing up to 16M bytes directly.	IOH -3 mA IOL 48 mA IOZH 20 uA IOZL -400 uA
AM0..AM2 (AM3) AM4 (AM5)	B16..B18 (B19) A23 (C14)	ADDRESS MODIFIERS Six TS-output signals providing additional address information. AM3 and AM5 are not connected.	IOH -3 mA IOL 64 mA IOZH 50 uA IOZL -50 uA
(LWORD*)	(C13)	LONG WORD LWORD* is not connected.	
WRITE*	A14	WRITE An active-low TS-output that specifies the direction of a data transfer: A high level indicates a read operation, a low level indicates a write operation.	IOH -3 mA IOL 64 mA IOZH 50 uA IOZL -50 uA
AS*	A18	ADDRESS STROBE An active low bidirectional TS-output/ST-input signal. During a data transfer the falling edge indicates a valid address on the bus. During bus arbitration the rising edge indicates the end of the last cycle.	IOH -3 mA IOL 64 mA IOZH -250 uA IOZL -750 uA IIH -250 uA IIL -750 uA
DS0*	A13	DATA STROBE 0 An active low TS-output that indicates a data transfer on the data lines D00-D07.	IOH -3 mA IOL 64 mA IOZH 50 uA IOZL -50 uA
DS1*	A12	DATA STROBE 1 An active low TS-output that indicates a data transfer on the data lines D08-D15.	IOH -3 mA IOL 64 mA IOZH 50 uA IOZL -50 uA

Table 2.8: VMEbus Signal Description (cont'd)

SIGNAL	PIN NO.	SIGNAL DESCRIPTION	ELEC. SPEC.
DTACK*	A16	DATA TRANSFER ACKNOWLEDGE An active low ST-input that indicates the successful completion of a data transfer.	IIH -250 uA IIL -700 uA
BERR*	C11	BUS ERROR An active low ST-input that indicates that an unrecoverable error has occured during a data transfer.	IIH -250 uA IIL -700 uA
BR0* BR1* BR2* BR3*	B12 B13 B14 B15	BUS REQUEST LEVEL 0-3 One of these active low signals is an optional OC-output at the jumpered bus priority level and indicates that the MVME101 module requests bus mastership. All four signals are inputs at the VMEbus Requester to support the release-on-request mode. BR3* is also an input at the MVME101 VMEbus Arbiter.	IOL 48 mA IOZH -250 uA IOZL -750 uA IIH -250 uA IIL -750 uA
BG0IN* BG1IN* BG2IN* BG3IN*	B4 B6 B8 B10	BUS GRANT INPUTS LEVEL 0-3 One of these active low signals is an optional ST-input at the jumpered bus priority level. It indicates to the MVME101 VMEbus Requester that a bus request on the same level has been granted by the bus arbiter. The remaining three bus grant inputs are jumpered directly to the respective bus grant outputs. BG3IN* is also an optional TP-output of the MVME101 VMEbus Arbiter.	IIH -250 uA IIL -700 uA
BG0OUT* BG1OUT* BG2OUT* BG3OUT*	B5 B7 B9 B11	BUS GRANT OUTPUTS LEVEL 0-3 One of these active low signals is an optional TP-output at the jumpered bus priority level. It indicates to the next module in the bus grant daisy-chain that it may become bus master. The remaining three bus grant outputs are jumpered directly to the respective bus grant inputs.	IOH -800 uA IOL 16 mA

Table 2.8: VMEbus Signal Description (cont'd)

SIGNAL	PIN NO.	SIGNAL DESCRIPTION	ELEC. SPEC.
BBSY*	B1	BUS BUSY This active low bidirectional signal indicates that a master module is using the data transfer bus. It is an OC-output of the VMEbus Requester and an ST-input at the VMEbus Arbiter.	IOL 48 mA IOZH -250 uA IOZL -900 uA IIH -250 uA IIL -900 uA
BCLR*	B2	BUS CLEAR This active low ST-input signal is driven by a multilevel bus arbiter when a bus request of a higher than the current bus master's level is pending. On the MVME101 BCLR* can be used for generating an interrupt in this event.	IIH -250 uA IIL -700 uA
IRQ1*... ...IRQ7*	B30... ...B24	INTERRUPT REQUEST LEVEL 1-7 Seven optional active low input signals that generate a prioritized interrupt request at the MPU. Level seven is the highest priority.	IIH -250 uA IIL -900 uA
IACK*	A20	INTERRUPT ACKNOWLEDGE An active low TS-output that indicates an interrupt vector fetch on the data transfer bus.	IOH -3 mA IOL 48 mA IOZH 20 uA IOZL -400 uA
IACKIN* IACKOUT*	A21 A22	INTERRUPT ACKNOWLEDGE INPUT INTERRUPT ACKNOWLEDGE OUTPUT These signals form an interrupt acknowledge daisy-chain through the interrupt requesters. On the MVME101 module IACKIN* and IACKOUT* are directly connected.	
ACFAIL*	B3	AC POWER FAILURE An active low ST-input that is driven by the power supply module. It indicates that the DC supply voltages may be out of the specified limits after 10 milliseconds and generates a non-maskable interrupt.	IIH -250 uA IIL -700 uA

Table 2.8: VMEbus Signal Description (cont'd)

SIGNAL	PIN NO.	SIGNAL DESCRIPTION	ELEC. SPEC.
SYSFAIL*	C10	SYSTEM FAILURE This active low signal indicates that a failure has occured in the system. On the MVME101 it is an optional bidirectional OC-output/ST-input, and can be jumpered to generate an interrupt at the MPU.	IOL 48 mA IOZH -250 uA IOZL -900 uA IIH -250 uA IIL -900 uA
SYSRESET*	C12	SYSTEM RESET This active low signal causes a complete VME system reset. On the MVME101 it is an optional OC-output that is activated by the Reset Switch and upon power up. Also, it is an optional ST-input that causes a board reset when asserted by another module.	IOL 48 mA IOZH -250 uA IOZL -900 uA IIH -250 uA IIL -900 uA
SYSCLK	A10	SYSTEM CLOCK An optional TP-output that deliveres the 16 MHz system clock signal.	IOH -3 mA IOL 60 mA
(SERCLK) (SERDAT)	(B21) (B22)	SERIAL COMMUNICATION BUS CLOCK SERIAL COMMUNICATION BUS DATA SERCLK and SERDAT are not connected.	
GND	A9, A11, A15, A17, A19, B20, B23, C9	GROUND	
+5V	A32, B32, C32	+ 5 VOLTS POWER	
(+5VSTB)	(B31)	+ 5 VOLTS STAND BY POWER +5VSTB is not connected.	
+12V	C31	+ 12 VOLTS POWER	
-12V	A31	- 12 VOLTS POWER	

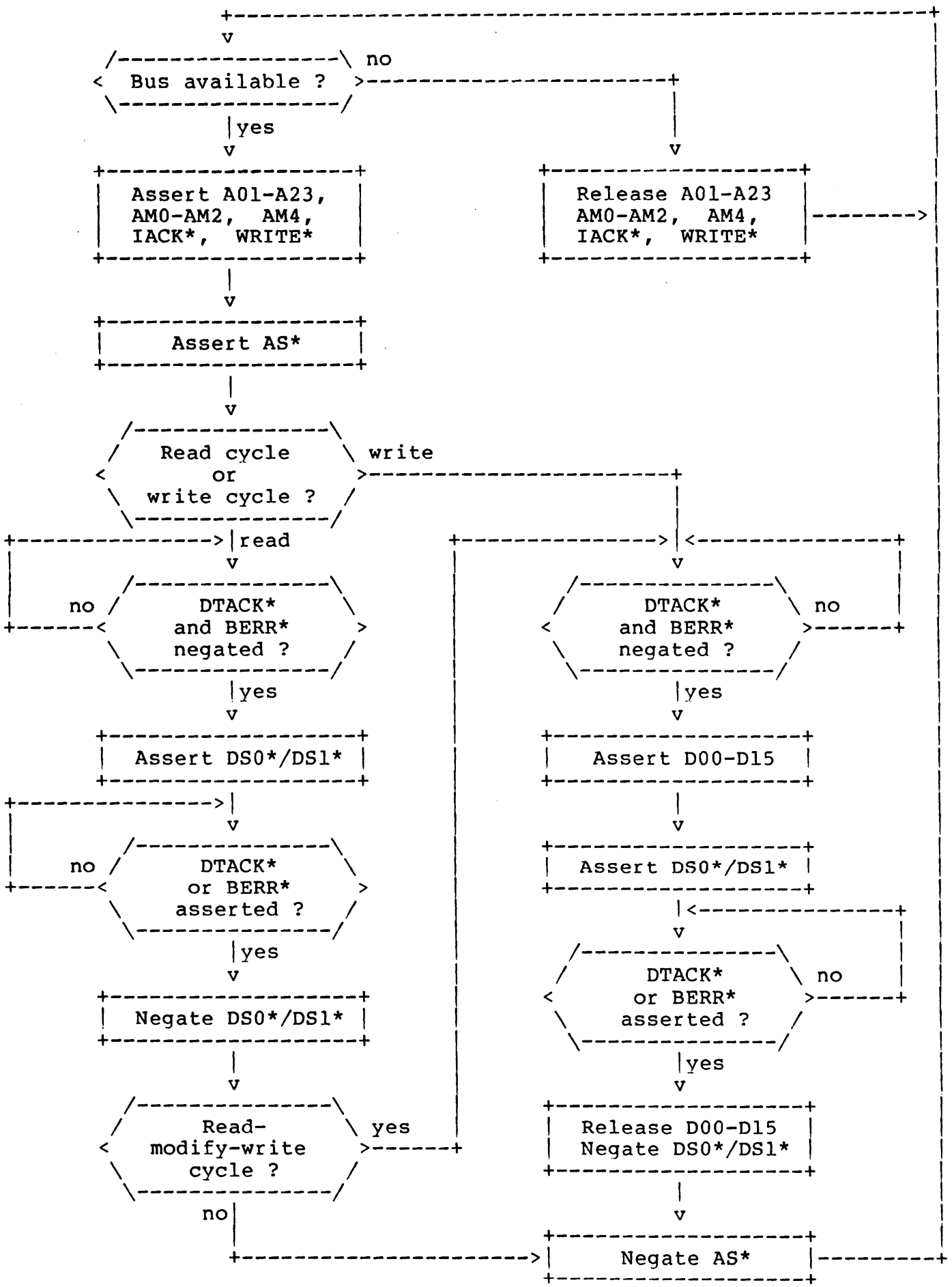
Table 2.9: Connector P1 Signal Locations

PIN NO.	ROW A SIGNALS	ROW B SIGNALS	ROW C SIGNALS	PIN NO.
1	D00	BBSY*	D08	1
2	D01	BCLR*	D09	2
3	D02	ACFAIL*	D10	3
4	D03	BG0IN*	D11	4
5	D04	BG0OUT*	D12	5
6	D05	BG1IN*	D13	6
7	D06	BG1OUT*	D14	7
8	D07	BG2IN*	D15	8
9	GND	BG2OUT*	GND	9
10	SYSCLK	BG3IN*	SYSFAIL*	10
11	GND	BG3OUT*	BERR*	11
12	DS1*	BR0*	SYSRESET*	12
13	DS0*	BR1*	(LWORD*)	13
14	WRITE*	BR2*	(AM5)	14
15	GND	BR3*	A23	15
16	DTACK*	AM0	A22	16
17	GND	AM1	A21	17
18	AS*	AM2	A20	18
19	GND	(AM3)	A19	19
20	IACK*	GND	A18	20
21	IACKIN*	(SERCLK)	A17	21
22	IACKOUT*	(SERDAT)	A16	22
23	AM4	GND	A15	23
24	A07	IRQ7*	A14	24
25	A06	IRQ6*	A13	25
26	A05	IRQ5*	A12	26
27	A04	IRQ4*	A11	27
28	A03	IRQ3*	A10	28
29	A02	IRQ2*	A09	29
30	A01	IRQ1*	A08	30
31	-12V	(+5VSTB)	+12V	31
32	+5V	+5V	+5V	32

2.9.2. VMEbus Data Transfer

The category of VMEbus signals which is responsible for transferring data between master and slave modules is termed the Data Transfer Bus (DTB). On the MVME101 module, the DTB drivers and receivers are enabled and disabled by the VMEbus Requester upon acknowledging and releasing bus mastership. For meeting the data transfer protocol and timing requirements of the VMEbus Specification, the VMEbus Interface logic generates its own signal handshaking and timing, independently of the MPU. Figure 2.10 shows a flow chart of the DTB interface operation.

Figure 2.10: VMEbus Data Transfer Flow Chart



2.9.3. Address Modifiers

The address modifier signals are used to provide slave modules on the VMEbus with additional addressing information, as defined in the VMEbus Specification Manual. The VME101 supports a subset of the specified address modifier codes which is listed in Table 2.10. Note that the signals AM3 and AM5 are not driven by the VMEbus Interface, but kept in the high state by the termination resistors on the VMEbus backplane.

Table 2.10: Address Modifier Codes

AM CODE	ADDRESS MODIFIER 5	4	3	2	1	0	FUNCTION
3E	1	1	1	1	1	0	standard supervisory program access
3D	1	1	1	1	0	1	standard supervisory data access
3A	1	1	1	0	1	0	standard non-privileged program access
39	1	1	1	0	0	1	standard non-privileged data access
2D	1	0	1	1	0	1	short supervisory I/O data access
29	1	0	1	0	0	1	short non-privileged I/O data access

2.9.4. Time-Out Counters

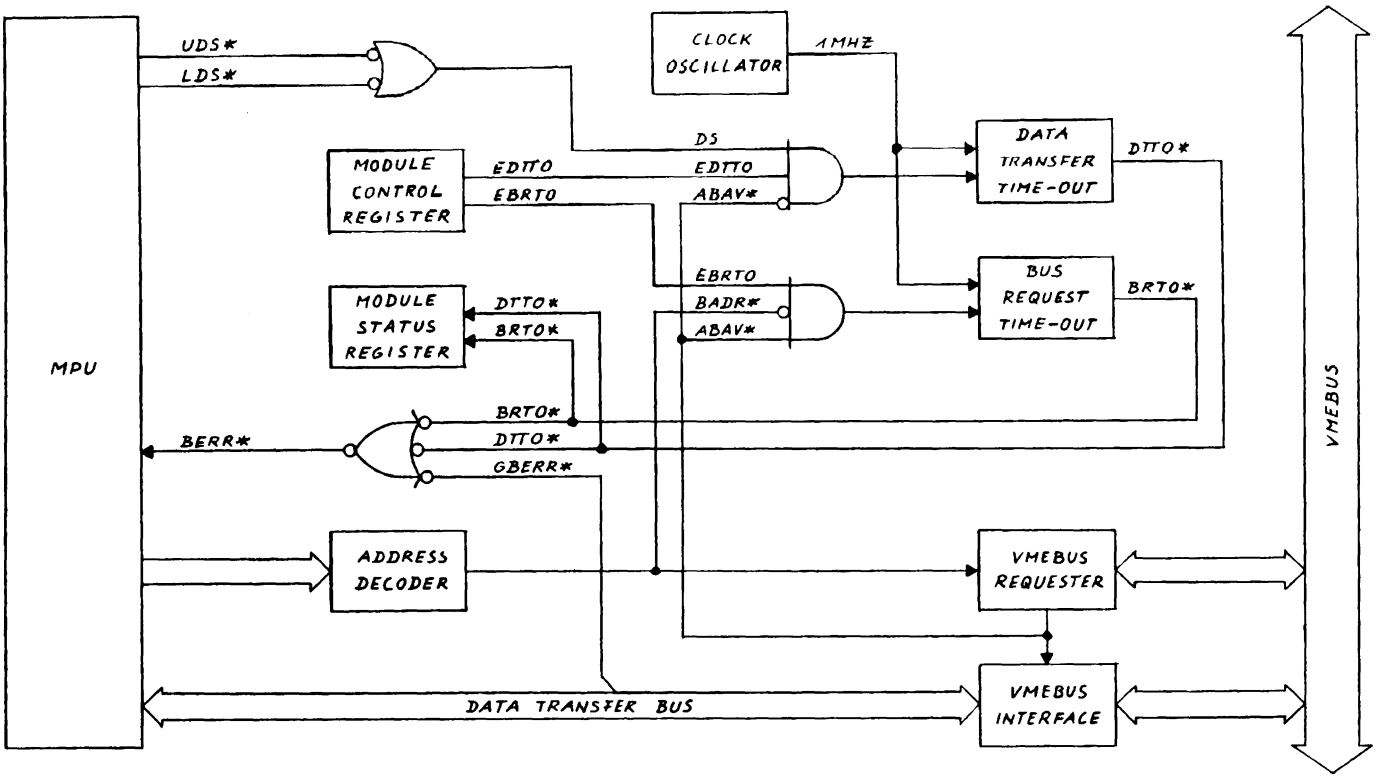
The MVME101 contains two time-out counters for supervising VMEbus accesses: the Bus Request Time-Out Counter, and the Data Transfer Time-Out Counter. Both counters can be independently enabled and disabled under software control. Figure 2.11 shows how the time out counters are interconnected with other functional blocks on the MVME101.

The Bus Request Time-Out (BRTO) Counter is enabled when bit 6 in the Module Control Register is set. It starts running when the MPU has asserted an off-board address, and the module is not currently the bus master, i. e. when a bus request is initiated by the Address Decoder. Note that if a Bus Block Transfer Request was asserted, the BRTO Counter is not started until the first off-board access. Once being started, the BRTO Counter asserts a Bus Error at the MPU and sets bit 6 in the Module Status Register, if the bus does not become available within 128 microseconds.

The Data Transfer Time-Out (DTTO) Counter is enabled when bit 7 in the Module Control Register is set. It starts running when the MPU has asserted a data strobe signal, and the bus is available, i. e. at the beginning of an off-board data transfer cycle. The DTTO Counter asserts a Bus Error at the MPU and sets bit 7 in the Module Status Register, if the data transfer is not acknowledged within 8 microseconds.

Any assertion of BERR* causes the MPU to abort the started cycle and to enter an exception routine. In this routine the MPU should test bit 6 and bit 7 in the Module Status Register to determine the source of the Bus Error to be either the BRTO Counter, or the DTTO Counter, or the addressed device, and then branch to the appropriate service routine. After testing the bits, the MPU should perform a dummy write into the Module Status Register to clear the bits 6 and 7.

Figure 2.11: Time Out Counters



The Bus Request Time-Out Counter should be used with care. If the MVME101 module does not reside on the highest bus arbitration level, and if another module occupies the bus for transferring large blocks of data, it may often take more than 128 microseconds until a bus request of the MVME101 is granted. As software recovery from a Bus Error is a problematic task under certain conditions, it might sometimes not be allowed to abort an off-board cycle. In such cases the BRTO Counter must be disabled, or the off-board access must be embedded in a routine that is protected by a Bus Block Transfer Request.

The Data Transfer Time-Out Counter should be constantly enabled in all systems that do not contain very slow slave modules with access times of more than 8 microseconds. This prevents the MPU from being hung up in case of system malfunctioning, such as addressing defect devices or non existant locations.

2.9.5. Interface Options

Several VMEbus signals are optionally used by the MVME101 Monoboard Computer. By setting or removing jumpers, the module can be configured either as the system controller module in a VMEbus system (System Controller Configuration), or as an MPU module on a selectable priority in a multiprocessor VMEbus system (Standard Configuration), or as an isolated monoboard system that resides only physically on a VMEbus backplane (Isolated Configuration). The jumper configurations for these different modes of operation are described in Chapter 3.

2.9.5.1. System Controller Configuration

When the MVME101 is configured as the VMEbus system controller, the following options are selected:

VMEbus Arbiter: The VMEbus Arbiter arbitrates bus requests on level 3 and drives BG3IN*.

VMEbus Requester: The VMEbus Requester operates on level 3. It receives BG3IN* from the VMEbus Arbiter and drives BG3OUT*. The remaining three bus grant outputs are not connected.

Interrupt Handler: The Interrupt Handler receives interrupt requests on the jumpered levels.

System Utilities: SYSFAIL* is both driven through the Module Control Register and received through the Module Status Register. Also, it may be jumpered to generate an interrupt.

SYSRESET* is bidirectional. It is driven by the Reset Switch and upon power-up. When received from a power supply module, it causes a board reset.

SYSCLK is driven by the on-board clock oscillator.

2.9.5.2. Standard Configuration

When the MVME101 is configured as a non-controller MPU module in a VMEbus system, the following options are selected:

VMEbus Arbiter: The VMEbus Arbiter is disconnected from BG3IN* and thus disabled.

VMEbus Requester: The VMEbus Requester receives BGIN* and drives BGOUT* on the selected priority level. The remaining three bus grant inputs are jumpered directly to the respective bus grant outputs.

Interrupt Handler: The Interrupt Handler receives interrupt requests on the jumpered levels.

System Utilities: SYSFAIL* is both driven through the Module Control Register and received through the Module Status Register. Also, it may be jumpered to generate an interrupt.

SYSRESET* is an input only. When received from another module, it causes a board reset. The Reset Switch has no effect on the bus.

SYSCLK is not connected.

2.9.5.3. Isolated Configuration

When the MVME101 is configured as an isolated monoboard computer, it can be placed on a VMEbus backplane without effecting other modules in the system. In this configuration, the MVME101 takes its power supply from the VMEbus, but neither drives nor responds to any bus signal, with the exception of ACFAIL*.

VMEbus Arbiter: The VMEbus Arbiter is disconnected from BG3IN* and thus disabled.

VMEbus Requester: The VMEbus Requester is disconnected from the bus. Thus the DTB drivers remain constantly in the high-impedance state. All bus grant inputs are jumpered directly to the respective bus grant outputs.

Interrupt Handler: The Interrupt Handler does not receive any interrupt requests from the VMEbus.

System Utilities: SYSFAIL* is not connected.

SYSRESET* is not connected. The board is reset by Reset Switch and upon power-up.

SYSCLK is not connected.

2.10. RESET AND HALT FUNCTIONS

The reset structure of the MVME101 is shown in Figure 2.12. There are four sources on the module which perform reset functions: The Power-Up Reset cicuit, the Reset Switch, the MPU executing a RESET operation, and the MPU being halted. The interaction between the on-board reset signals and the VMEbus depends on the configuration of jumper area K3.

For the three selectable VMEbus Interface options that are described in Paragraph 2.9.5, the effects of all reset sources on the on-board devices and on the VMEbus signals SYSRESET* and SYSFAIL* are listed in Table 2.11.

The Power-Up Reset cicuit and the Reset Switch have identical functions: MPU, PCI1, PCI2, PIA, PTM, MCR, and VMEbus Requester are reset, and the Address Decoder is initialized for the reset vector fetch. When configured as System Controller, the VMEbus SYSRESET* is asserted.

In the System Controller and in the Standard Configuration the assertion of SYSRESET* on the VMEbus produces the same effects as the Reset Switch. In the Isolated Configuration a VMEbus reset is ignored.

When the MPU executes a RESET instruction, only the on-board I/O-devices (PCI1, PCI2, PIA, PTM) are reset. No other devices are affected.

When the MPU is halted because of a double bus fault, the MCR is reset for negating an eventual Bus Block Transfer Request, and the decimal points on the Display are lit to indicate the halted state. In the System Controller and in the Standard Configuration also the VMEbus signal SYSFAIL* is asserted.

Figure 2.12: Reset Structure

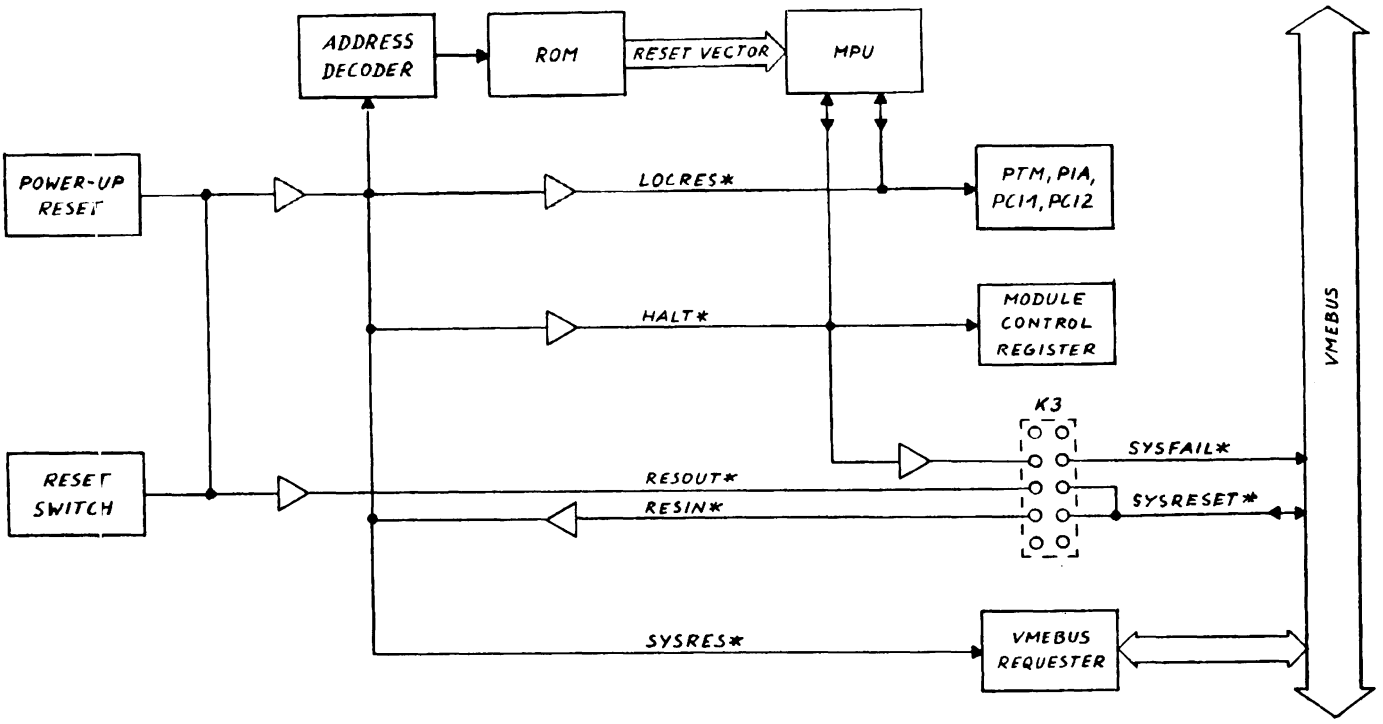


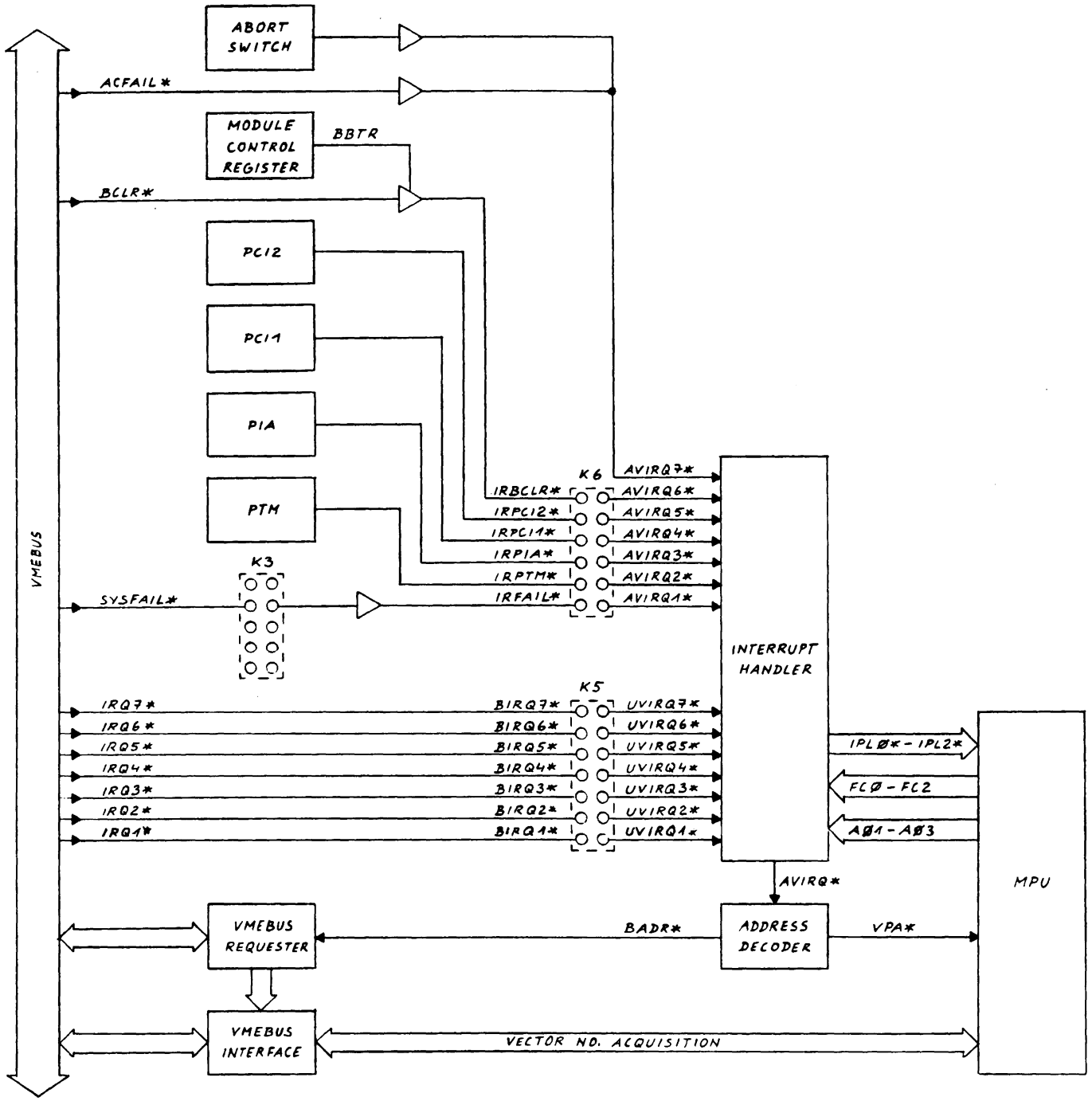
Table 2.11: Reset and Halt Functions

RESET SOURCE	MODULE CONFIGURATION	AFFECTED DEVICES
Power-Up Reset, Reset Switch	System Controller	MPU, PCI1, PCI2, PIA, PTM, MCR, Addr.Decoder, VMEbus Requester, VME SYSRESET*
	Standard Config.	MPU, PCI1, PCI2, PIA, PTM, MCR, Addr.Decoder, VMEbus Requester
	Isolated Config.	MPU, PCI1, PCI2, PIA, PTM, MCR, Address Decoder
VMEbus SYSRESET*	System Controller, Standard Config.	MPU, PCI1, PCI2, PIA, PTM, MCR, Addr.Decoder, VMEbus Requester
	Isolated Config.	none
RESET Instruction	any	PCI1, PCI2, PIA, PTM
MPU Halted	System Controller Standard Config.	MCR, VMEbus SYSFAIL*
	Isolated Config.	MCR

2.11. INTERRUPT HANDLER

The Interrupt Handler is responsible for encoding interrupt requests coming from on-board devices or from the VMEbus, for asserting the highest pending interrupt request at the MPU, and for managing the interrupt acknowledge cycle. The MC68000 Data Sheet in Appendix A gives a detailed description how the MPU processes interrupts. A block diagram of the Interrupt Handler and its interconnections with the VMEbus and the on-board devices is shown in Figure 2.13.

Figure 2.13: Interrupt Handler



Interrupt requests are categorized into two groups: seven prioritized Auto-Vectorized Interrupt Requests (AVIRQ1* - AVIRQ7*), which are acknowledged in the automatic vectoring mode, and seven prioritized User-Vectorized Interrupt Requests (UVIRQ1* - UVIRQ7*), where the interrupt vector number is supplied by the interrupting device. Auto-Vectorized Interrupt Requests may be caused by the on-board I/O-devices, by the Abort switch, or by the VMEbus signals ACFAIL*, SYSFAIL* and BCLR*. The User-Vectorized Interrupt Requests represent the VMEbus interrupt signals IRQ1* - IRQ7*.

The Interrupt Handler arbitrates incoming interrupt requests according to their priority levels, and encodes the highest pending request on the interrupt inputs of the MPU. When the interrupt is acknowledged, the Interrupt Handler decodes the priority level of the MPU and compares it with the interrupt requests. If an Auto-Vectorized Interrupt Request is pending on the acknowledged priority, the Interrupt Handler asserts the VPA* signal, and the MPU uses the interrupt autovector of this priority level as a pointer to the exception routine. Otherwise, the Interrupt Handler assumes an off-board interrupter, and initiates an interrupt acknowledge cycle on the VMEbus for acquiring the user-vector number.

The priority sequence of the fourteen available interrupt requests can be represented as follows:

Highest level:	AVIRQ7*	(non-maskable)
	UVIRQ7*	(non-maskable)
	AVIRQ6*	
	UVIRQ6*	
	AVIRQ5*	
	UVIRQ5*	
	AVIRQ4*	
	UVIRQ4*	
	AVIRQ3*	
	UVIRQ3*	
	AVIRQ2*	
	UVIRQ2*	
	AVIRQ1*	
Lowest level:	UVIRQ1*	

The devices capable of asserting Auto-Vectorized Interrupt Requests are not fixed at appointed priority levels, with exception of the Abort switch and the ACFAIL* signal, which are hard-wired to AVIRQ7*. All other devices may be jumpered to any of the six interrupt request inputs AVIRQ1* - AVIRQ6* on the jumper area K6. Also, two or more of the interrupt outputs of these devices may be connected in a wired-or configuration on one common priority level.

VMEbus interrupt requests used by the Interrupt Handler must be jumpered to the according User-Vectorized Interrupt Requests of the same priority level in a one-to-one configuration on the jumper area K5. Also, they must not be wire-or connected, as that would short the interrupt request signals on the VMEbus.

The configuration of the jumper areas K5 and K6 is described in Paragraphs 3.4.3 and 3.4.4.

2.11.1. Software Abort and AC Failure

The Abort switch on the front panel and the VMEbus signal ACFAIL* are both connected with the Auto-Vectorized Interrupt Request AVIRQ7*, thus causing a non-maskable interrupt of the highest priority. To determine the appropriate service routine, the status of the ABORT* and ACFAIL* signals can be read in the Module Status Register.

2.11.2. System Failure

When the MVME101 is configured as the System Controller, the VMEbus signal SYSFAIL* can be jumpered to generate an Auto-Vectorized Interrupt Request on a selectable priority. In case of a system failure, lower priority programs would then be interrupted, and the MVME101 enters a service routine. The status of SYSFAIL* can be read in the Module Status Register.

2.11.3. Bus Clear

If the MVME101 resides in a system that contains other modules with a higher bus priority, the VMEbus signal BCLR* should be jumpered to an Auto-Vectorized Interrupt Request. This provides the bus arbiter with the means to interrupt lower priority programs on the MVME101 that are executed in the block-transfer mode, when another module has a bus request of a higher priority pending. BCLR* can only cause an interrupt when the Bus Block Transfer Bit in the Module Control Register is set. Otherwise BCLR* is ignored, as the VMEbus Requester then operates in the release-on-request mode. However, the status of BCLR* can be read at any time in the Module Status register.

2.11.4. On-Board I/O Interrupts

All interrupt request outputs of the on-board I/O-devices can be jumpered to generate Auto-Vectorized Interrupt Requests on selectable levels.

2.11.5. VMEbus Interrupts

Any or all of the VMEbus interrupt request signals IRQ1* - IRQ6* may be jumpered to generate User-Vectorized Interrupt Requests on the according priorities. The appropriate interrupt vector numbers are fetched from the interrupter in a VMEbus interrupt acknowledge cycle.

2.12. TIMING SPECIFICATIONS

This paragraph provides detailed timing specifications of the MVME101 module for local memory access and for VMEbus operations. The tabulated maximum and minimum times are guaranteed over the recommended operating conditions, as specified in Table 1.1. Whenever possible, typical times for operation at 25 C temperature and 5.00 V supply voltage are given.

The following list summarizes the operations described in this paragraph and the respective figures and tables:

For local memory accesses, the specifications include both the timing supplied by the MPU and the Address Decoder, and the timing requirements for the installed memory devices. As the number of wait cycles inserted by the MPU during local ROM accesses is selectable, the specification of the local memory read cycle timing includes all available options from 0 to 3 wait cycles (in the tables abbreviated W.C.). For read operations from local RAM, the times specified for 0 W.C. are valid.

For VMEbus operations, this paragraph specifies the timings that are supplied by the MVME101 module for interactions with other modules on the bus. No timing requirements for these modules are given, but it is assumed that they comply with the VMEbus Specification Rev.B. Whenever possible, the timing relations between MPU signals and VMEbus signals are specified for bus operations.

The signal mnemonics used in the following figures and tables are identical with the signal names used in the schematic diagrams in Chapter 4. To distinguish between on-board and off-board signals, the mnemonics of all on-board signals are put in parantheses.

Figure 2.14: Local Memory Read Cycle

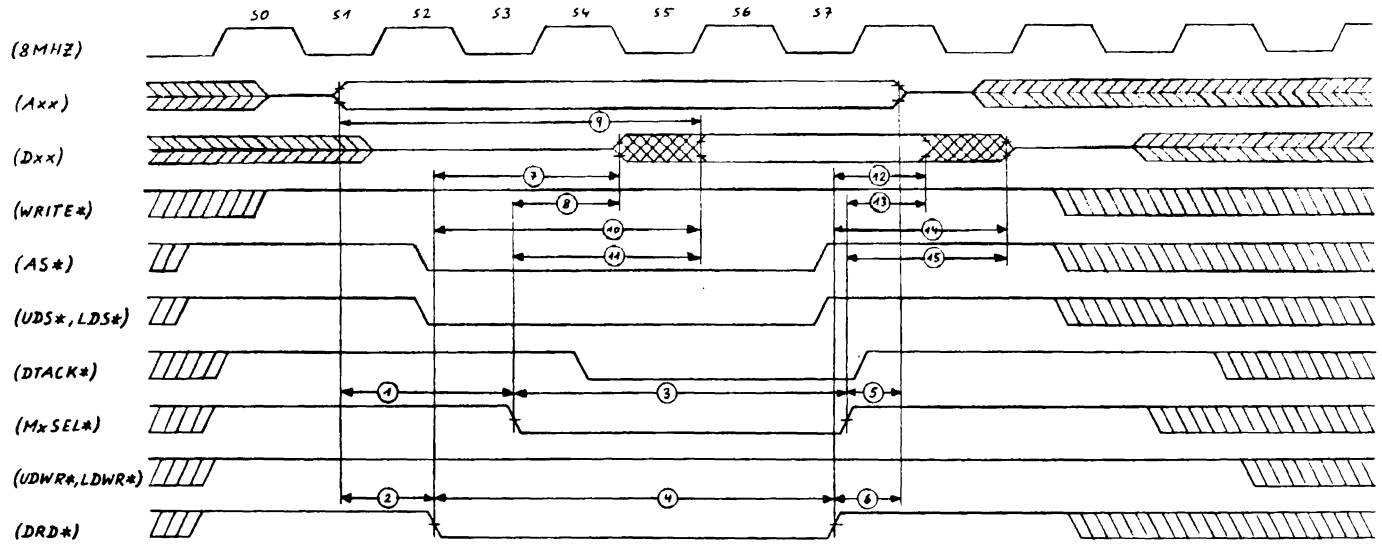


Table 2.12: Local Memory Read Cycle Timing

NO	PARAMETER	W.C.	MIN	TYP	MAX	UNIT
	Supplied Memory Access Timing:					
1	(Axx) Valid to (MxSEL*) Low	0-3	90			ns
2	(Axx) Valid to (DRD*) Low	0-3	35			ns
3	(MxSEL*) Width Low	0	190	260		ns
		1	315	385		ns
		2	440	510		ns
		3	565	635		ns
4	(DRD*) Width Low	0	240	310		ns
		1	365	435		ns
		2	490	560		ns
		3	615	685		ns
5	(MxSEL*) High to (Axx) Invalid	0-3	10			ns
6	(DRD*) High to (Axx) Invalid	0-3	25			ns
	Memory Response Requirements:					
7	(DRD*) Low to (Dxx) Low Impedance	0-3	0			ns
8	(MxSEL*) Low to (Dxx) Low Imped.	0-3	0			ns
9	(Axx) Valid to (Dxx) Valid	0			290	ns
		1			415	ns
		2			540	ns
		3			665	ns
10	(DRD*) Low to (Dxx) Valid	0			225	ns
		1			350	ns
		2			475	ns
		3			600	ns
11	(MxSEL*) Low to (Dxx) Valid	0			155	ns
		1			280	ns
		2			405	ns
		3			530	ns
12	(DRD*) High to (Dxx) Invalid	0-3	0			ns
13	(MxSEL*) High to (Dxx) Invalid	0-3	0			ns
14	(DRD*) High to (Dxx) High Imped.	0-3			140	ns
15	(MxSEL*) High to (Dxx) High Imped.	0-3			125	ns

Figure 2.15: Local Memory Write Cycle

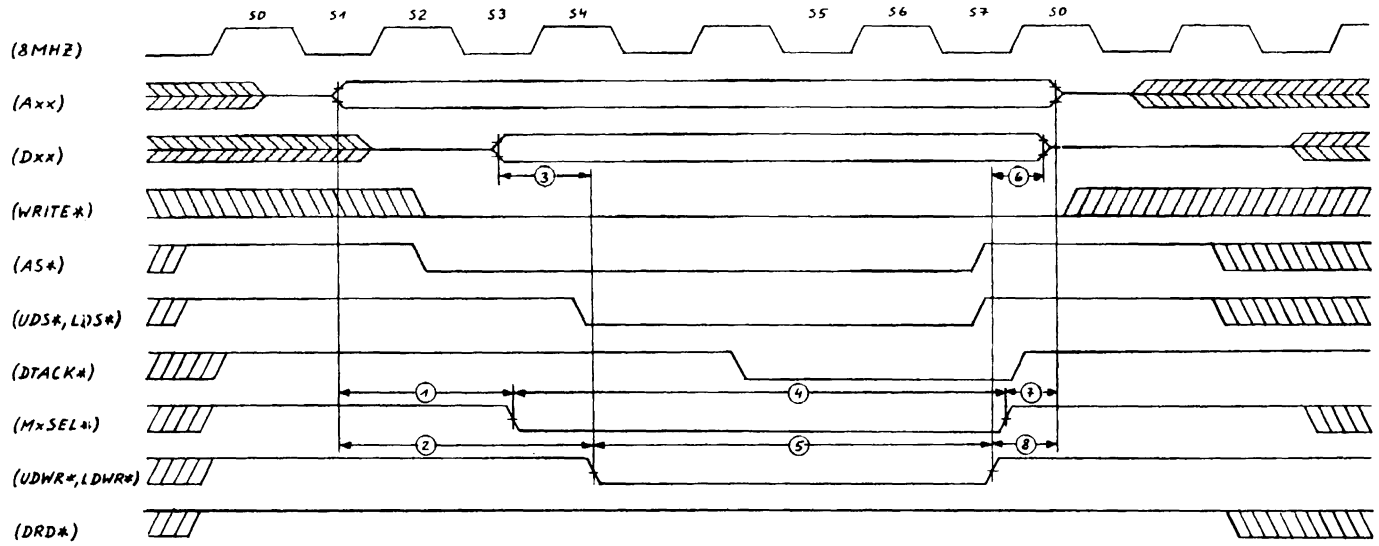


Table 2.13: Local Memory Write Cycle Timing

NO	PARAMETER	MIN	TYP	MAX	UNIT
1	(Axx) Valid to (MxSEL*) Low	90			ns
2	(Axx) Valid to (LDWR*), (UDWR*) Low	115			ns
3	(Dxx) Valid to (LDWR*), (UDWR*) Low	35			ns
4	(MxSEL*) Width Low	315	385		ns
5	(LDWR*), (UDWR*) Width Low	240	310		ns
6	(LDWR*), (UDWR*) High to (Dxx) Invalid	10			ns
7	(MxSEL*) High to (Axx) Invalid	10			ns
8	(LDWR*), (UDWR*) High to (Axx) Invalid	10			ns

Figure 2.16: VMEbus Read Cycle

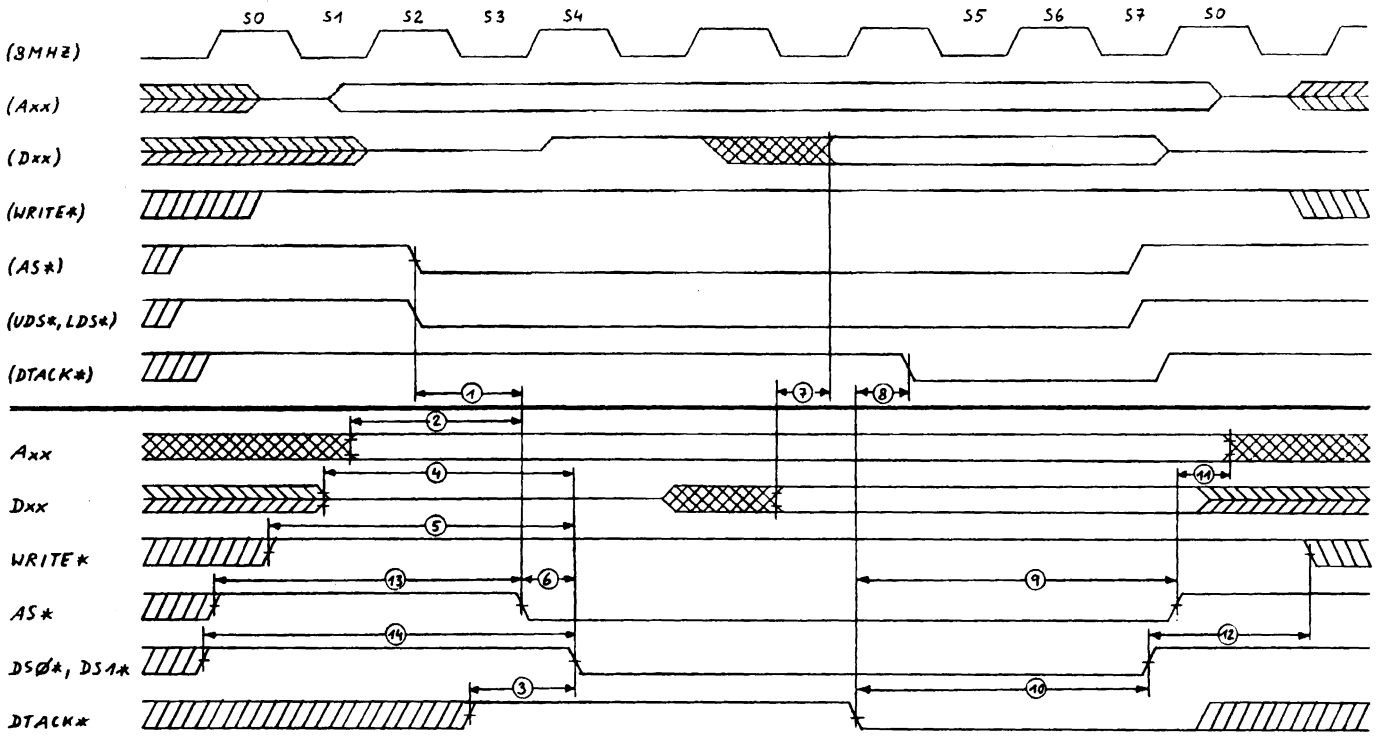


Table 2.14: VMEbus Read Cycle Timing

NO	PARAMETER	NOTES	MIN	TYP	MAX	UNIT
1	(AS*) Low to AS* Low		60	75	90	ns
2	Axx Valid to AS* Low		40			ns
3	DTACK* High to DS0*, DS1* Low	1	20	45	75	ns
4	Dxx High Imped. to DS0*, DS1* Low		180			ns
5	WRITE* High to DS0*, DS1* Low		120			ns
6	AS* Low to DS0*, DS1* Low	2	5	15	35	ns
7	Dxx Valid to (Dxx) Valid		5	10	15	ns
8	DTACK* Low to (DTACK*) Low		10	20	35	ns
9	DTACK* Low to AS* High		10			ns
10	DTACK* Low to DS0*, DS1* High		10		250	ns
11	AS* High to Axx Invalid		0			ns
12	DS0*, DS1* High to WRITE* Invalid		65			ns
13	AS* Width High		195			ns
14	DS0*, DS1* Width High		210			ns

Note 1 : Provided that AS* is low.
Note 2 : Provided that DTACK* is high.

Figure 2.17: VMEbus Write Cycle

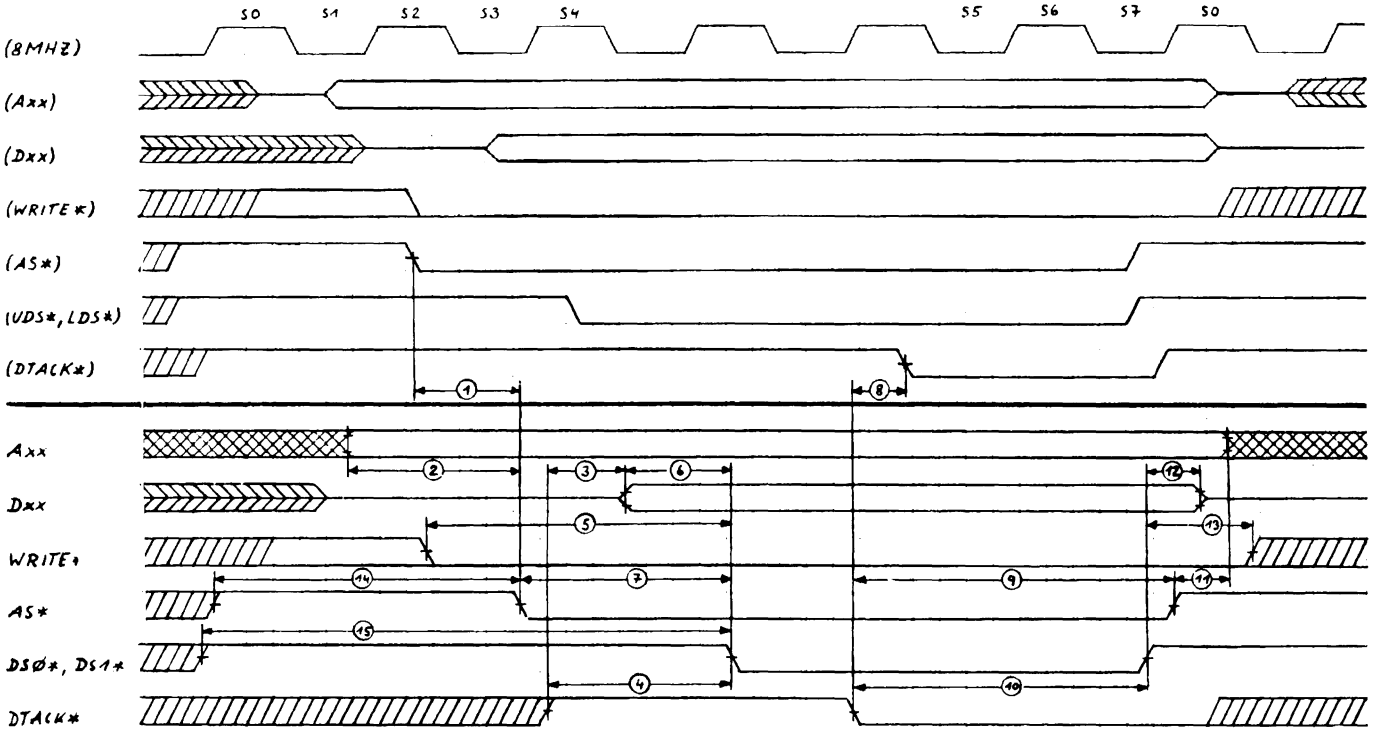


Table 2.15: VMEbus Write Cycle Timing

NO	PARAMETER	NOTES	MIN	TYP	MAX	UNIT
1	(AS*) Low to AS* Low		60	75	90	ns
2	Axx Valid to AS* Low		40			ns
3	DTACK* High to Dxx Low Impedance	1	25	50	75	ns
4	DTACK* High to DS0*, DS1* Low	1	95	125	160	ns
5	WRITE* Low to DS0*, DS1* Low		165			ns
6	Dxx Valid to DS0*, DS1* Low		45			ns
7	AS* Low to DS0*, DS1* Low	2	65	160	260	ns
8	DTACK* Low to (DTACK*) Low		10	20	35	ns
9	DTACK* Low to AS* High		10			ns
10	DTACK* Low to DS0*, DS1* High		10		250	ns
11	AS* High to Axx Invalid		0			ns
12	DS0*, DS1* High to Dxx Invalid		-20			ns
13	DS0*, DS1* High to WRITE* Invalid		15			ns
14	AS* Width High		195			ns
15	DS0*, DS1* Width High		340			ns

Note 1 : Provided that AS* is low.
Note 2 : Provided that DTACK* is high.

Figure 2.18: VMEbus Request and Acquisition

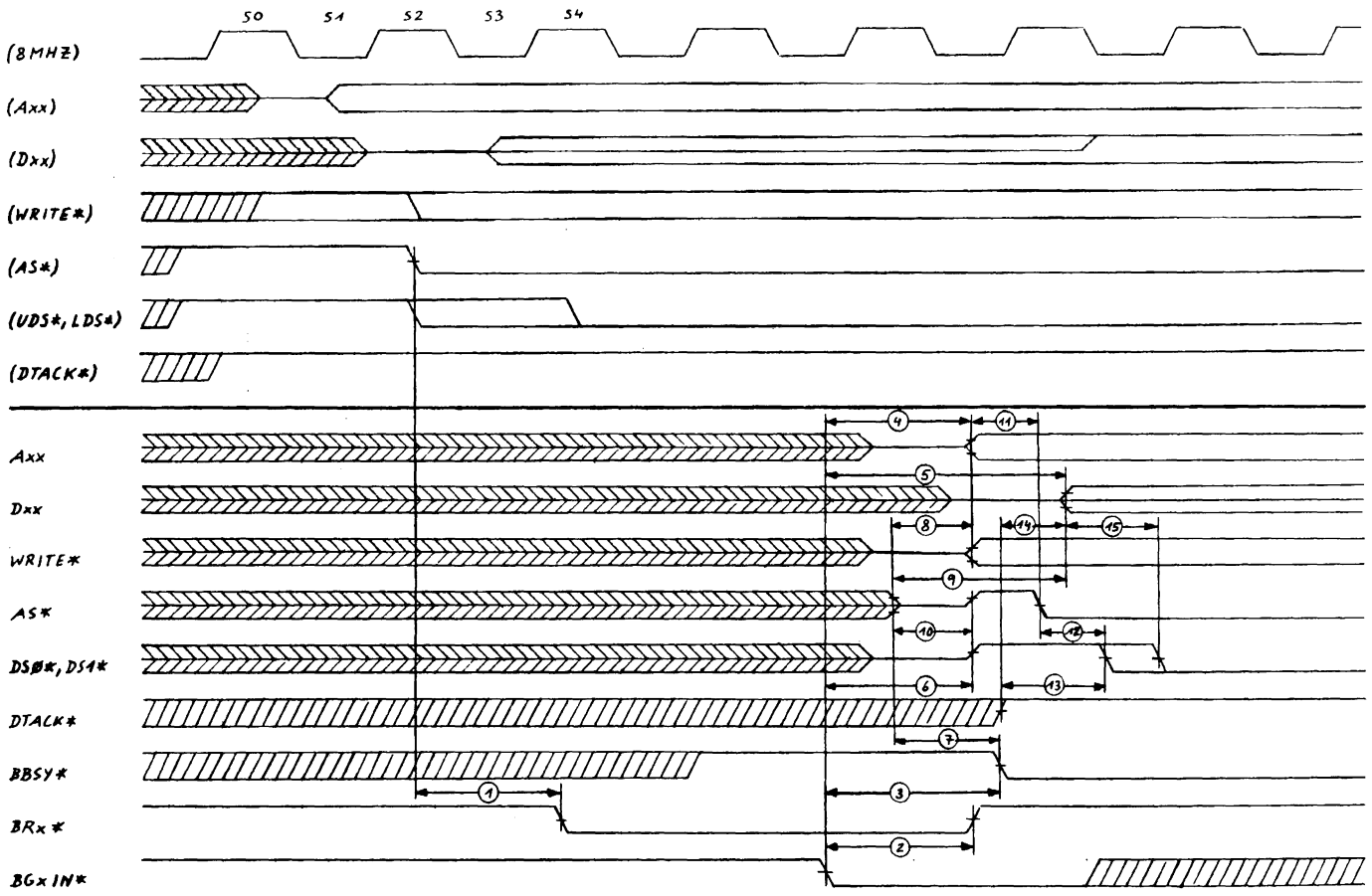


Table 2.16: VMEbus Request and Acquisition Timing

NO	PARAMETER	NOTES	MIN	TYP	MAX	UNIT
1	(AS*) Low to BRx* Low		80		265	ns
2	BGxIN* Low to BRx* High	1	140		335	ns
3	BGxIN* Low to BBSY* Low	1	55		250	ns
4	BGxIN* Low to Axx, WRITE* Valid	1	25		225	ns
5	BGxIN* Low to Dxx Valid (write)	1,3	30		235	ns
6	BGxIN* Low to Strobes Low Imped.	1	15		200	ns
7	AS* High to BBSY* Low	2	55		250	ns
8	AS* High to Axx, WRITE* Valid	2	25		225	ns
9	AS* High to Dxx Valid (write)	2,3	30		235	ns
10	AS* High to Strobes Low Imped.	2	15		200	ns
11	Axx, WRITE* Valid to AS* Low		40	60	90	ns
12	AS* Low to DS0*, DS1* Low (read)	3	5	15	35	ns
13	DTACK* High to DS0*,DS1* Low (read)	4	20	45	75	ns
14	DTACK* High to Dxx Valid (write)	4	25	50	75	ns
15	Dxx Valid to DS0*,DS1* Low (write)		45	80	120	ns

Note 1 : Provided that the previous master has released the bus.
Note 2 : Provided that the bus request has been granted.
Note 3 : Provided that DTACK* is high.
Note 4 : Provided that AS* is low.

Figure 2.19: VMEbus Release and Bus Grant Propagation

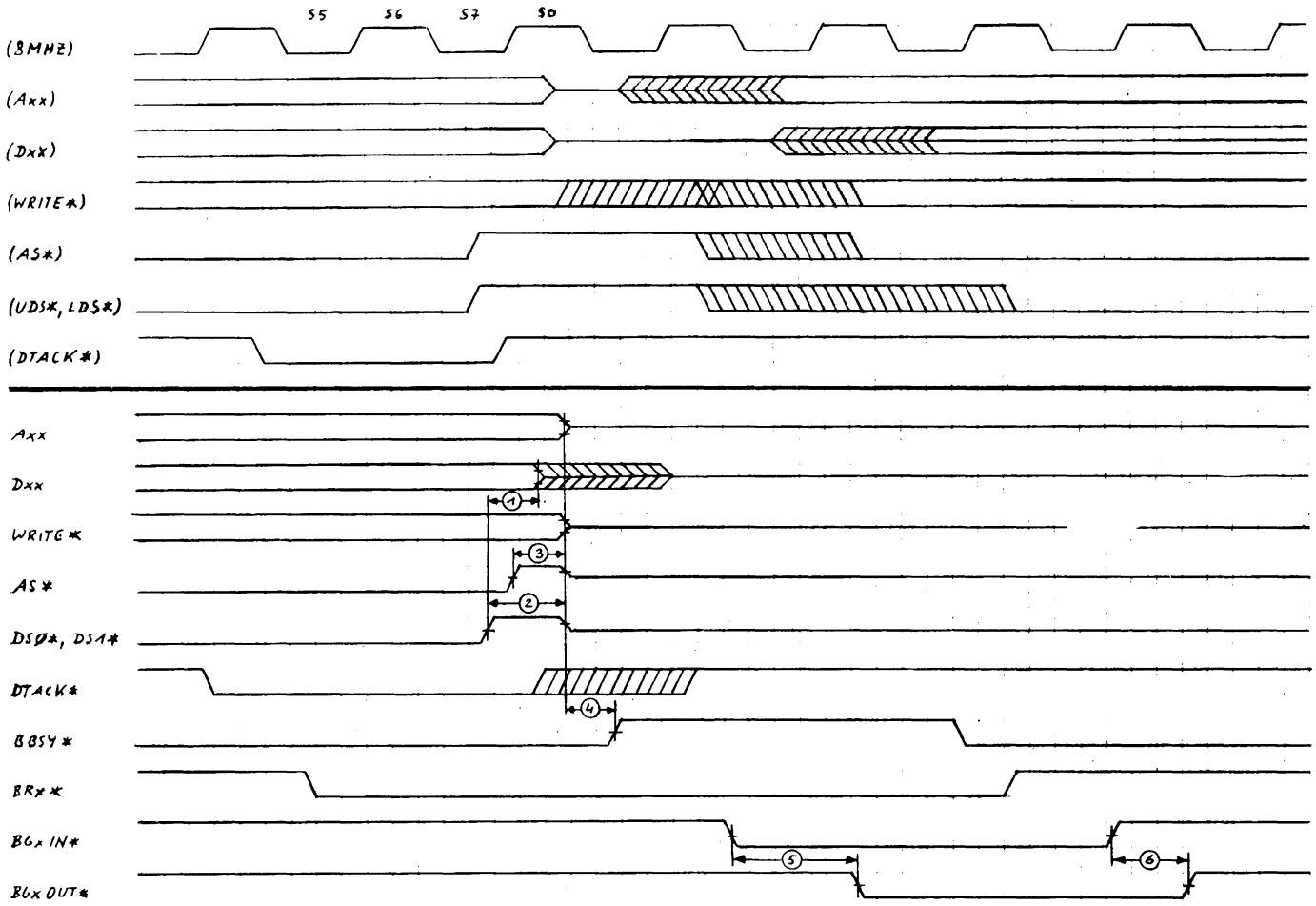


Table 2.17: VMEbus Release and Bus Grant Propagation Timing

NO	PARAMETER	MIN	TYP	MAX	UNIT
1	DS0*, DS1* High to Dxx Invalid (write)	-20			ns
2	DS0*, DS1* High to DTB High Impedance	5	35	80	ns
3	AS* High to DTB High Impedance	5		220	ns
4	DTB High Impedance to BBSY* High	20	40	60	ns
5	BGxIN* Low to BGxOUT* Low	45		225	ns
6	BGxIN* High to BGxOUT* High	45		225	ns

CHAPTER 3

INSTALLATION

3.1. INTRODUCTION

This chapter provides the user of the MVME101 monoboard computer with the unpacking, inspection, hardware preparation and installation procedures.

3.2. UNPACKING INSTRUCTIONS

IF THE SHIPPING CARTON IS DAMAGED UPON RECEIPT, REQUEST THAT CARRIER'S AGENT BE PRESENT DURING UNPACKING AND INSPECTION OF THE MODULE.

Unpack the MVME101 monoboard computer from its shipping carton. Refer to the packing list and verify that all items are present. Save the packing material for storing or reshipping the module.

AVOID TOUCHING AREAS OF MOS CIRCUITRY. STATIC DISCHARGE CAN DAMAGE INTEGRATED CIRCUITS.

3.3. INSPECTION

The module should be inspected upon receipt for broken, damaged or missing parts and for physical damage to the printed circuit board.

3.4. HARDWARE PREPARATION

This paragraph describes the hardware preparation of the MVME101 module prior to system installation. That includes configuring the jumper areas to select the various optional functions of the module, and programming the Address Decoder PROM according to the desired address map.

Figure 3.1 illustrates the physical location of each jumper area on the module. Table 3.1 lists the function of each jumper area and refers to the detailed descriptions in Paragraphs 3.4.1 through 3.4.9.

Figure 3.1: MVME101 Jumper Area Locations

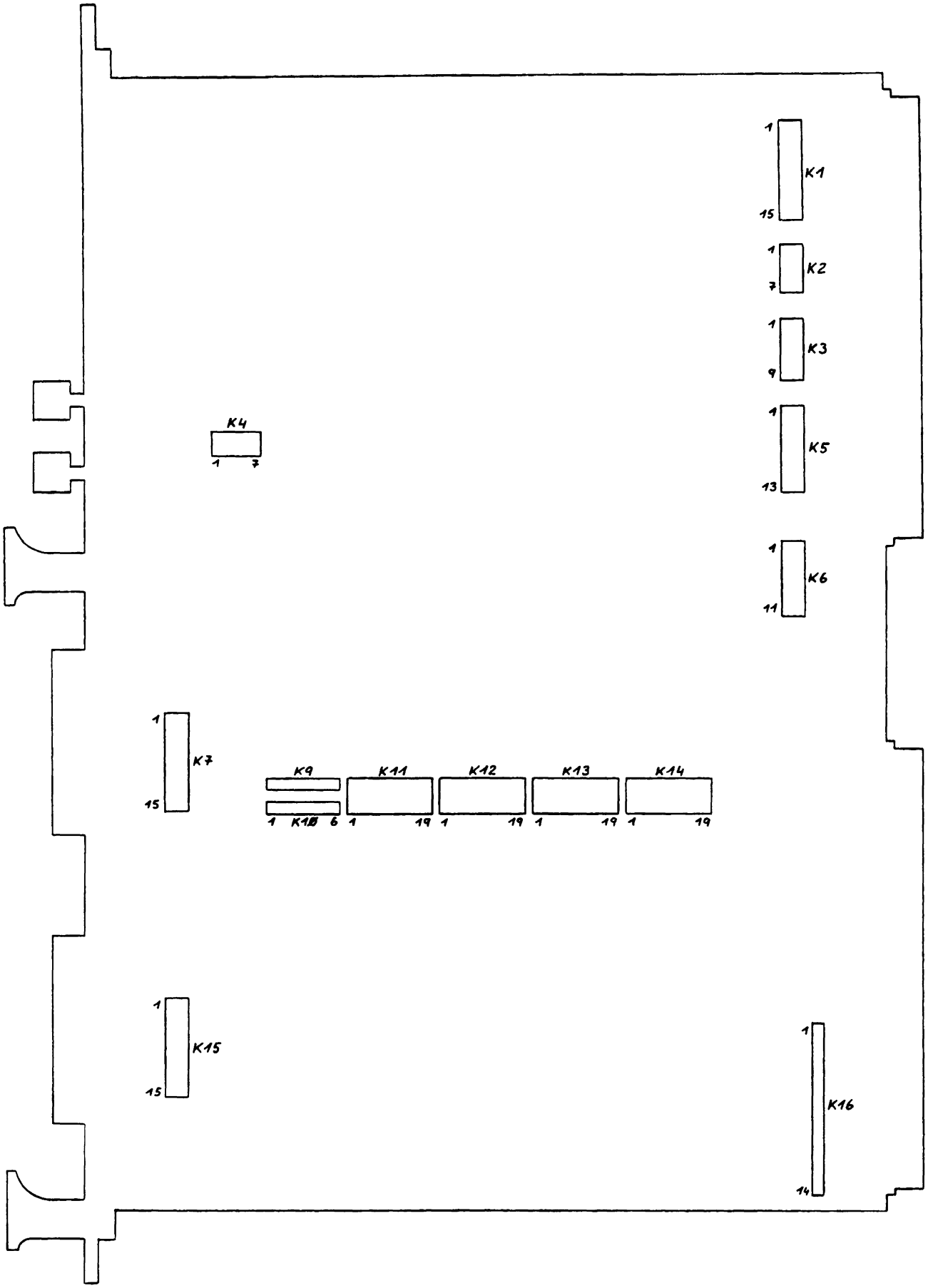


Table 3.1: MVME101 Jumper Areas

JUMPER	FUNCTION	OPTIONS	PARAGR.
K1, K2	VMEbus Requester Priority Level	Select level 0, 1, 2, or 3, or isolated configuration.	3.4.1
K3	VMEbus System Control Functions	Enable/disable SYSCLK output, enable/disable SYSFAIL*, enable/disable RESET* output, enable/disable RESET* input, enable/disable VMEbus Arbiter.	3.4.2
K4	Local ROM Access Time	Insert 0, 1, 2, or 3 wait cycles.	3.4.9
K5	User-Vectorized Interrupt Requests	Enable/disable VMEbus interrupt request inputs IRQ1*, IRQ2*, IRQ3*, IRQ4*, IRQ5*, IRQ6*, IRQ7*.	3.4.3
K6	Auto-vectorized Interrupt Requests	Enable/disable interrupt request inputs from PCI1, PCI2, PIA, PTM, SYSFAIL*, BCLR*, and select their priorities.	3.4.4
K7, K15	Serial Ports Configuration	Configure SP1 and SP2 as Data Set or Data Terminal, and for synchronous or asynchronous operation.	3.4.5
K9, K10	Serial Interface Control	Configure interrupt outputs and control inputs of PCI1 and PCI2.	3.4.6
K11, K12, K13, K14	Memory Sockets Configuration	Configure signal locations at the memory socket pairs MEM1, MEM2, MEM3 and MEM4 according to the used devices.	3.4.8
K16	PTM Connections	Configure PTM clock and gate inputs.	3.4.7

3.4.1. VMEbus Requester Priority

The jumper areas K1 and K2 determine the priority level on which the VMEbus Requester will operate. On K2 the bus request output signal of the VMEbus Requester is connected with the appropriate bus request line, on K1 it is placed in the corresponding bus grant daisy-chain. Also, on K1 the unused bus grant inputs are jumpered to the respective bus grant outputs.

When the MVME101 is used as the VMEbus system controller, and in any system containing an option ONE single level VMEbus arbiter, the VMEbus Requester must be placed on level 3. For use with a multilevel arbiter, any one of the four priority levels may be selected. When configured as isolated module, the VMEbus Requester is disconnected from the bus.

Original configuration: VMEbus Requester on level 3

Figure 3.2: Jumper Area K1

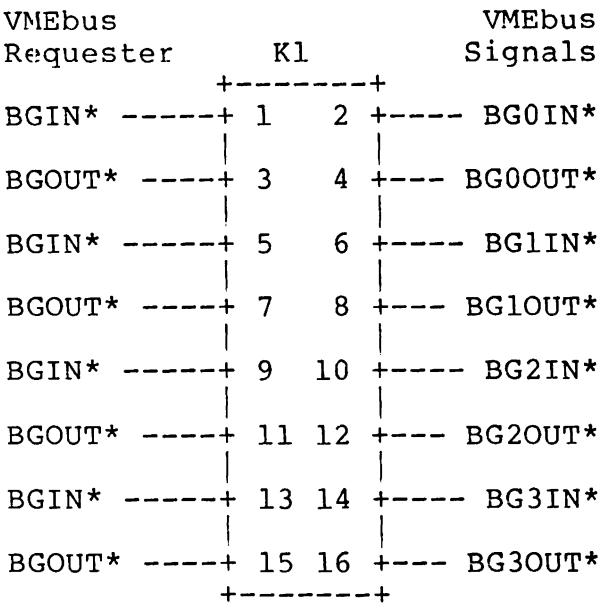


Figure 3.3: Jumper Area K2

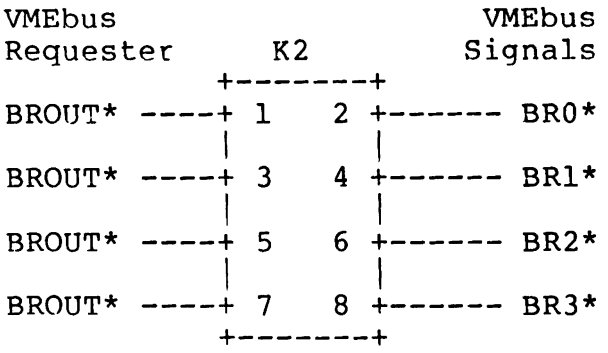


Table 3.2: VMEbus Requester Priority Selection

K1 and K2 CONNECTIONS	SELECTED PRIORITY LEVEL
K1: 1-2, 3-4, 6-8, 10-12, 14-16 K2: 1-2	VMEbus Requester on level 0
K1: 2-4, 5-6, 7-8, 10-12, 14-16 K2: 3-4	VMEbus Requester on level 1
K1: 2-4, 6-8, 9-10, 11-12, 14-16 K2: 5-6	VMEbus Requester on level 2
K1: 2-4, 6-8, 10-12, 13-14, 15-16 K2: 7-8	VMEbus Requester on level 3, System Controller Configuration
K1: 2-4, 6-8, 10-12, 14-16 K2: none	Module isolated from VMEbus

3.4.2. VMEbus System Control Functions

The jumper area K3 is used to enable or disable various VMEbus system control functions. The VMEbus signals SYSCLK, SYSFAIL*, RESET*, and the VMEbus Arbiter output are fed through K3, and can be independently selected to be handled by the MVME101 module.

When the MVME101 is used as the VMEbus system controller, all optional system control outputs must be enabled to provide the system clock, system failure, system reset, and bus arbiter functions. When the board is a non-controller MPU module in the standard configuration, the system clock, system reset, and bus arbiter outputs must be disabled. In the isolated configuration, all system control signals must be disconnected from the VMEbus, to ensure proper stand-alone operation.

Original configuration: VMEbus System Controller

Figure 3.4: Jumper Area K3

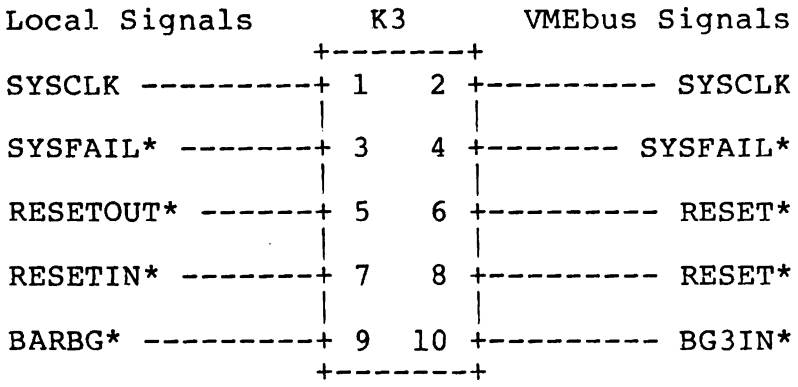


Table 3.3: VMEbus System Control Configuration

K3 CONNECTIONS	ENABLED FUNCTIONS	MODULE CONFIGURATION
1-2, 3-4, 5-6, 7-8, 9-10	SYSCLK output, SYSFAIL* in/out, RESET* output, RESET* input, VMEbus Arbiter	VMEbus System Controller
3-4, 7-8	SYSFAIL* in/out, RESET* input	Standard Configuration
none	none	Isolated Configuration

3.4.3. User-Vectorized Interrupt Requests

The jumper area K5 determines which of the seven interrupt request lines on the VMEbus may interrupt the on-board MPU. Originally, all interrupt levels are enabled on K5 and handled by the MVME101. If any other modules capable of handling VMEbus interrupts are present in the system, the user must assign the interrupt levels to the interrupt handlers such that not more than one MPU responds to a given VMEbus interrupt. If a VMEbus interrupt is not to be received by the MVME101, the corresponding jumper must be removed from K5.

VMEbus interrupt requests cannot be wired to a different on-board interrupt level. The jumpers must be installed straight across the pins on jumper area K5.

Original configuration: All VMEbus interrupt requests enabled

Figure 3.5: Jumper Area K5

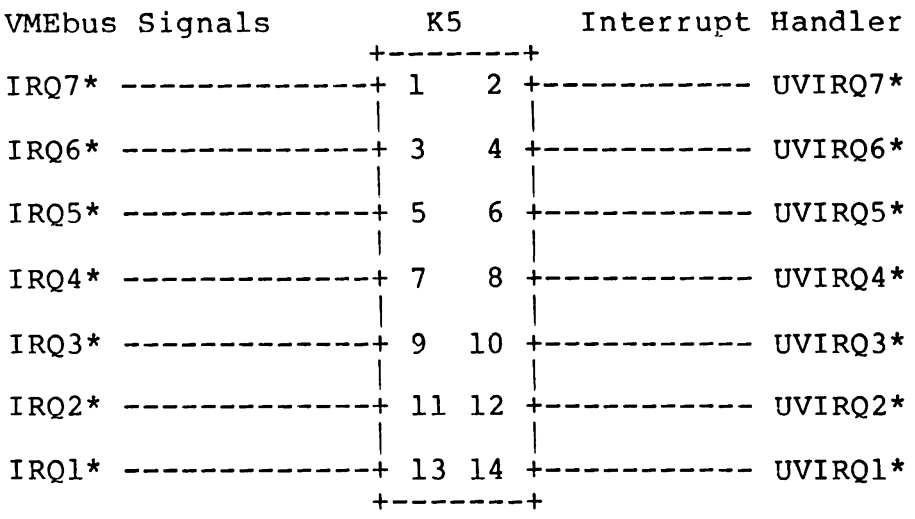


Table 3.4: User-Vectorized Interrupt Selection

K5 CONNECTIONS	ENABLED INTERRUPTS
1-2	VMEbus Interrupt Request level 7
3-4	VMEbus Interrupt Request level 6
5-6	VMEbus Interrupt Request level 5
7-8	VMEbus Interrupt Request level 4
9-10	VMEbus Interrupt Request level 3
11-12	VMEbus Interrupt Request level 2
13-14	VMEbus Interrupt Request level 1

3.4.4. Auto-Vectorized Interrupt Requests

On the jumper area K6 the interrupt request outputs of the on-board I/O-devices and the VMEbus signals SYSFAIL* and BCLR* may be jumpered to interrupt the MPU in the auto-vectorized mode. Any of these interrupters may be connected with any of the six lower MPU interrupt levels. Also, two or more interrupters may be jumpered in a wired-or configuration on one common interrupt request level.

The non-maskable auto-vectorized interrupt on level 7 is not available for the user. Instead, it is reserved for software abort and AC power failure.

Original configuration: SYSFAIL* on interrupt level 1,
PTM on interrupt level 2,
PIA on interrupt level 3,
PCI1 on interrupt level 4,
PCI2 on interrupt level 5,
BCLR* on interrupt level 6

Figure 3.6: Jumper Area K6

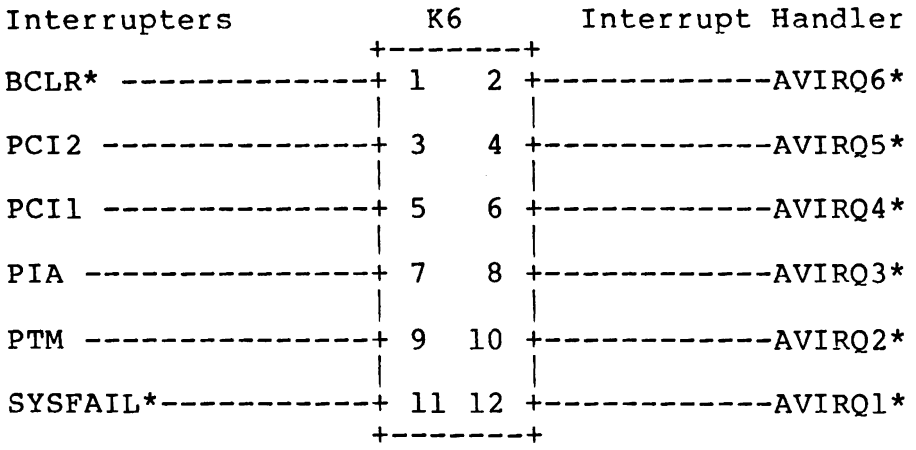


Table 3.5: Auto-Vectorized Interrupt Selection

K6 PIN	INTERRUPTER	K6 PIN	INTERRUPT REQUEST
1	VMEbus signal BCLR*	2	Interrupt level 6
3	PCI2 interrupt output	4	Interrupt level 5
5	PCI1 interrupt output	6	Interrupt level 4
7	PIA interrupt output	8	Interrupt level 3
9	PTM interrupt output	10	Interrupt level 2
11	VMEbus signal SYSFAIL*	12	Interrupt level 1

3.4.5. Serial Ports Configuration

The peripheral input/output signals of the Programmable Communication Interfaces are fed to the connectors on the front panel through the jumper areas K7 and K15. The jumpers on K7 determine the pin assignment of SP1, the jumpers on K15 that of SP2. Both ports may be configured independently as Data Terminal or Data Set, and for asynchronous or synchronous data transmission. Also, the DSR and CTS inputs may optionally be supported.

Original configuration: SP1 configured as asynchronous Data Set,
SP2 configured as asynchronous Data Terminal.

Figure 3.7: Jumper Area K7

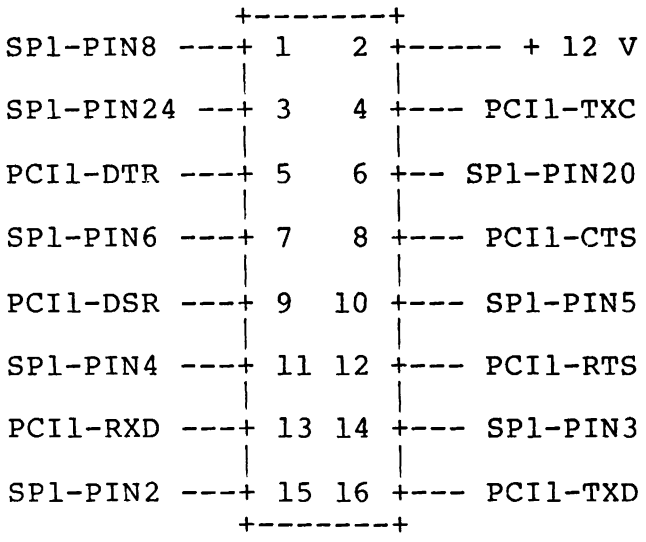


Figure 3.8: Jumper Area K15

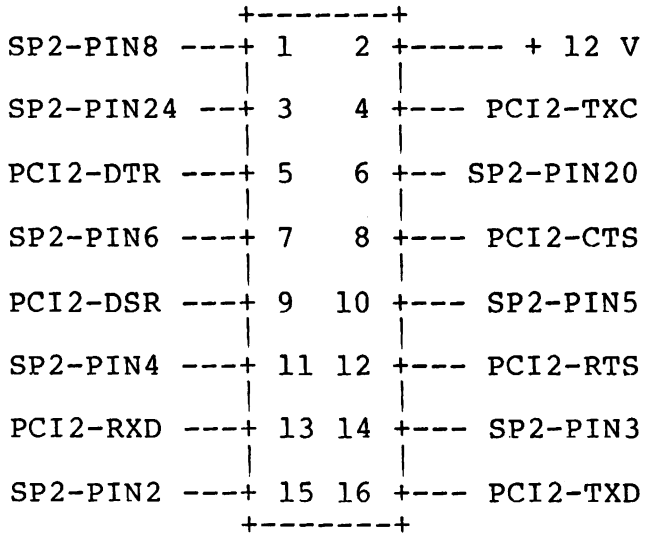


Table 3.6: Serial Ports Configuration

K7 / K15 CONNECTIONS	SP1 / SP2 CONFIGURATION
1-2, 5-7, 10-12, 13-15, 14-16	Port configured as Data Set
6-8	DSR/CTS controlled by Data Terminal
9-11	DSR/CTS controlled by Data Terminal
1-2, 5-6, 11-12, 13-14, 15-16	Port configured as Data Terminal
7-8	DSR/CTS controlled by Data Set
9-10	DSR/CTS controlled by Data Set
3 and 4 open	Asynchronous data transmission
3-4	Synchronous data transmission

3.4.6. Serial Interface Control

On the jumper areas K9 and K10 the interrupt outputs and the CTS* inputs of the Programmable Communication Interfaces may be configured for different modes of operation. K9 belongs to PCI1, K10 belongs to PCI2. For each interface, either one of the PCI interrupt outputs TXRDY* and RXRDY*, or both can be connected with the interrupt request line which is fed to the jumper area K6. There it may be jumpered on any auto-vectorized interrupt request. The CTS* inputs of the PCIs can be either constantly enabled, or shorted with the DSR* inputs, to support control from peripherals.

Original configuration: Interrupt outputs open,
CTS* inputs enabled

Figure 3.9: Jumper Area K9

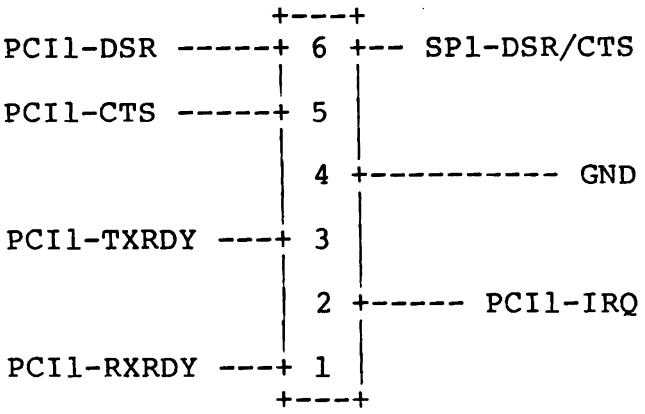


Figure 3.10: Jumper Area K10

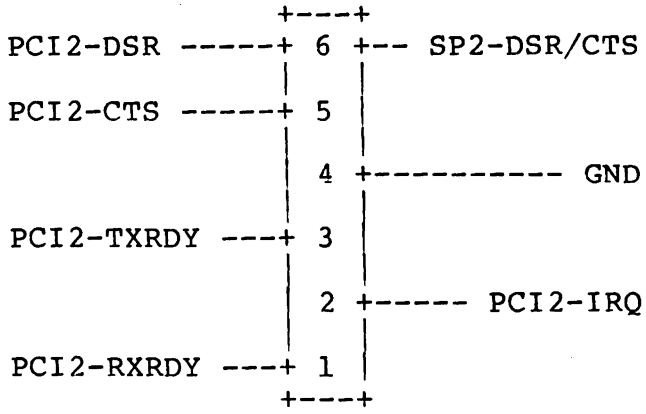


Table 3.7: Serial Interface Control

K9 / K10 CONNECTIONS	PCI1 / PCI2 CONFIGURATION
1-2	Interrupt asserted by RXRDY
2-3	Interrupt asserted by TXRDY
1-2-3	Interrupt asserted by RXRDY and TXRDY
4-5	CTS input constantly enabled
5-6	CTS input enabled by peripheral device

3.4.7. Programmable Timer Configuration

The peripheral clock, gate, and output signals of the Programmable Timer Module may be configured for several modes of operation on the jumper area K16. The gate inputs can be connected with ground and thus be constantly enabled. For real time counting, the clock input of counter 3 can be connected with the 2 MHz free running clock signal. VMEbus cycles can be counted by connecting the VMEbus address strobe with the clock input of counter 2. MPU cycles can be counted by connecting the MPU address strobe with the clock input of counter 1. Counters can be cascaded by connecting a counter's input with the output of the previous counter.

For other applications of the Programmable Timer Module, all peripheral clock, gate, and output signals are also available at connector P2.

Original configuration: No jumpers set

Figure 3.11: Jumper Area K16

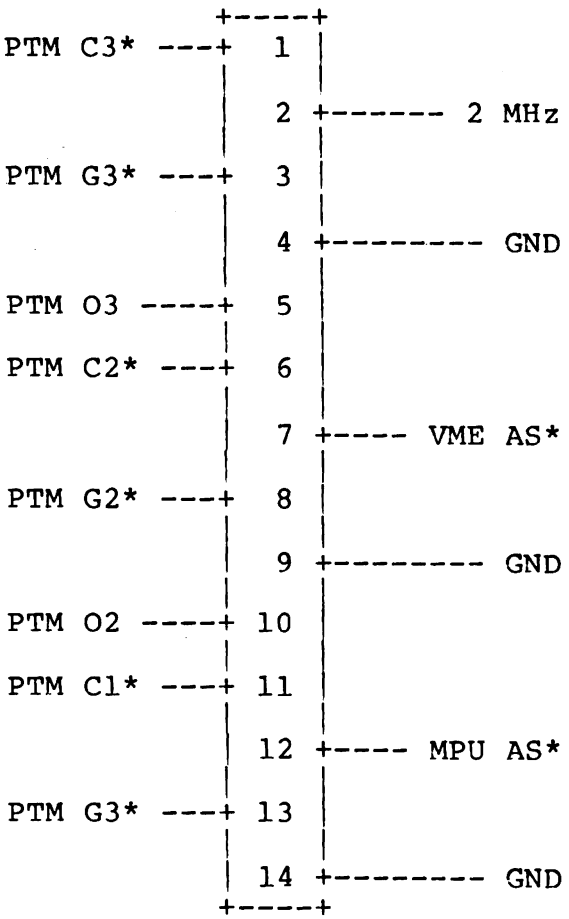


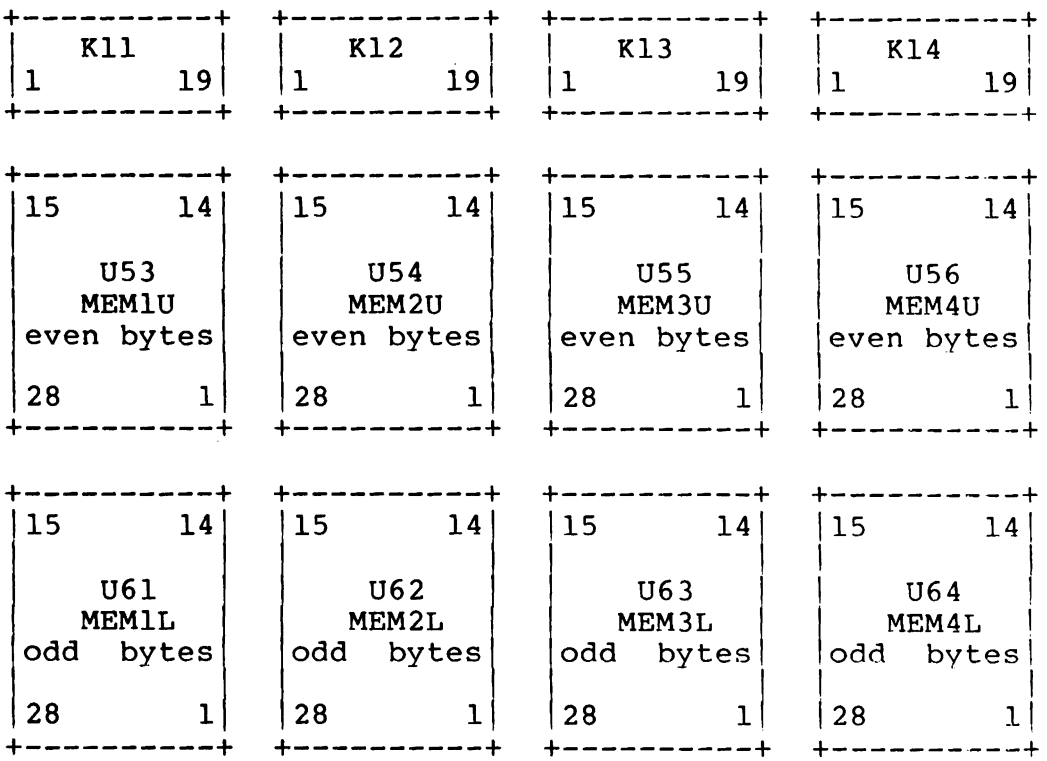
Table 3.8: Programmable Timer Configurations

K16 CONNECTIONS	PTM CONFIGURATION
3-4	Gate input of counter 3 is constantly enabled
8-9	Gate input of counter 2 is constantly enabled
13-14	Gate input of counter 1 is constantly enabled
1-2	Counter 3 is clocked with 2 Mhz real time clock
6-7	Counter 2 is clocked with VMEbus address strobe
11-12	Counter 1 is clocked with MPU address strobe
5-6	Counter 3 and counter 2 are cascaded
10-11	Counter 2 and counter 1 are cascaded

3.4.8. Memory Sockets Configuration

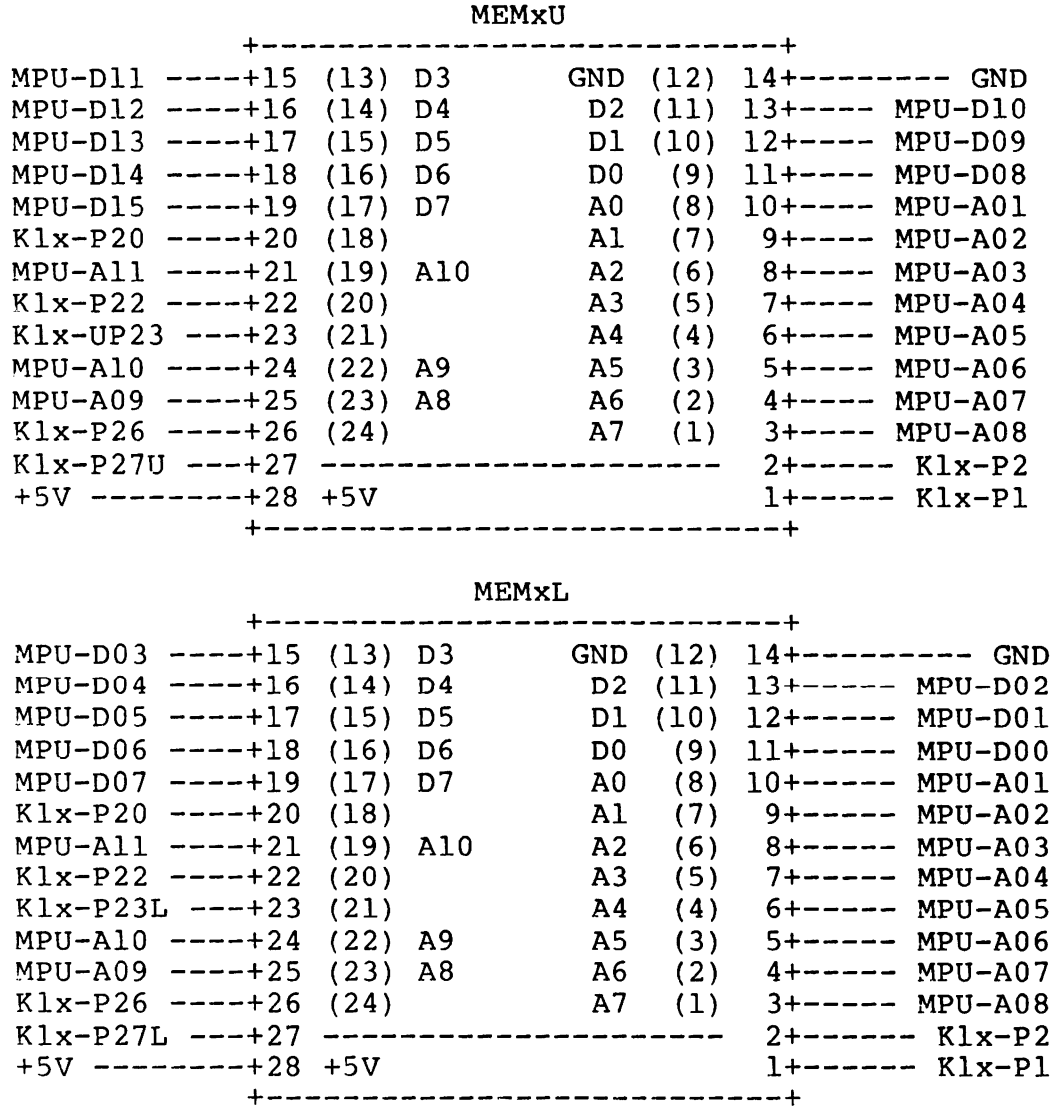
The jumper areas K11, K12, K13, and K14 are used to configure the memory sockets on the MVME101 for the various types of memory devices which may be installed. The memory array consists of eight 28-pin sockets, organized as four pairs. Each memory pair is configured individually on its associated jumper area.

Figure 3.12: Local Memory Organization



The memory sockets accept 24-pin dual-in-line packages as well as 28-pin packages, provided the devices are compatible with the JEDEC standard pin-out for byte-wide memories. 28-pin devices are inserted with pins 1 - 28 of the device matching pins 1 - 28 of the socket, 24-pin devices are inserted with pins 1 - 24 of the device matching pins 3 - 26 of the socket. By that the memory address inputs A0 - A10 are connected with the MPU address ouputs A01 - A11, the lower order (odd bytes) memory data lines D0 - D7 are connected with the MPU data lines D00 - D07, and the upper order (even bytes) memory data lines D0 - D7 are connected with the MPU data lines D08 - D15. For supporting different device sizes and pin-outs, the signals at pins 18, 20, and 21 of 24-pin memories, and the signals at pins 1, 2, 20, 22, 23, 26, and 27 of 28-pin memories are fed to the configuration jumper areas, where they have to be connected with the appropriate address and control signals. Figure 3.13 illustrates a memory socket pair and the signal connections for 28-pin and 24-pin devices.

Figure 3.13: Memory Pin Assignment



Note: The letter "x" reflects the number of the socket pair, and may have a value of 1, 2, 3, or 4.

Figure 3.14 shows the signal assignment on the jumper areas K11 - K14. On the local bus side, these signals are the address lines A12 - A15, the memory select signal MxS*, the output enable signal OE*, the upper byte and lower byte write pulses WRU* and WRL*, and the +5V power supply voltage. On the memory side, the socket pins 1, 2, 20, 22, 23U, 23L, 26, 27U, and 27L are fed to the jumper areas.

Figure 3.14: Jumper Areas K11 - K14

Pin Numbers:			Signals:		
21	20	19	A13	P23L	A12
18	17	16	P2	+5V	P20
15	14	13	A14	P1	MxS*
12	11	10	P26	A15	P22
9	8	7	+5V	P27L	OE*
6	5	4	P27L	WRL*	P23L
3	2	1	P27U	WRU*	P23U

After having selected the devices to be installed in a memory socket pair, the user has to configure the according jumper area. Table 3.9 and Table 3.10 list which signals must be connected on the jumper areas for RAMs and ROMs of different sizes.

Table 3.9: Signal Connections for RAM Devices

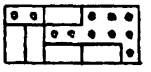
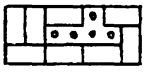
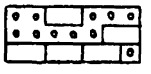
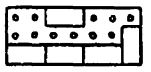
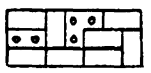
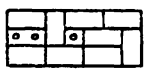
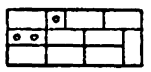
MEMORY TYPE	SIGNAL CONNECTIONS ON CONFIGURATION JUMPER AREA
2K x 8 RAM	MxS* to CS* of both RAMs, OE* to OE* of both RAMs, WRU* to WR* of upper RAM, WRL* to WR* of lower RAM
4K x 8 RAM	MxS* to CS* of both RAMs, OE* to OE* of both RAMs, WRU* to WR* of upper RAM, WRL* to WR* of lower RAM, A12 to A11 of both RAMs
8K x 8 RAM	MxS* to CS* of both RAMs, OE* to OE* of both RAMs, WRU* to WR* of upper RAM, WRL* to WR* of lower RAM, A12 to A11 of both RAMs, A13 to A12 of both RAMs
16K x 8 RAM	MxS* to CS* of both RAMs, OE* to OE* of both RAMs, WRU* to WR* of upper RAM, WRL* to WR* of lower RAM, A12 to A11 of both RAMs, A13 to A12 of both RAMs, A14 to A13 of both RAMs
32K x 8 RAM	MxS* to CS* of both RAMs, OE* to OE* of both RAMs, WRU* to WR* of upper RAM, WRL* to WR* of lower RAM, A12 to A11 of both RAMs, A13 to A12 of both RAMs, A14 to A13 of both RAMs, A15 to A14 of both RAMs

Table 3.10: Signal Connections for ROM Devices

MEMORY TYPE	SIGNAL CONNECTIONS ON CONFIGURATION JUMPER AREA
2K x 8 ROM	MxS* to CS* of both ROMs, OE* to OE* of both ROMs,
4K x 8 ROM	MxS* to CS* of both ROMs, OE* to OE* of both ROMs, A12 to A11 of both ROMs
8K x 8 ROM	MxS* to CS* of both ROMs, OE* to OE* of both ROMs, A12 to A11 of both ROMs, A13 to A12 of both ROMs
16K x 8 ROM	MxS* to CS* of both ROMs, OE* to OE* of both ROMs, A12 to A11 of both ROMs, A13 to A12 of both ROMs, A14 to A13 of both ROMs
32K x 8 ROM	MxS* to CS* of both ROMs, OE* to OE* of both ROMs, A12 to A11 of both ROMs, A13 to A12 of both ROMs, A14 to A13 of both ROMs, A15 to A14 of both ROMs

Table 3.11 lists several popular RAM and EPROM devices that may be installed in the local memory sockets. If any of these devices, or device types having identical pin-outs, are selected for use, the specified connections must be made on the according jumper areas. When devices with different pin-outs are installed, the user should refer to Tables 3.9 and 3.10 to determine the appropriate jumper configuration.

Table 3.11: Configurations for Popular Memories

K11, K12, K13, K14 CONNECTIONS		MEMORY DEVICE	
1-2, 4-5, 7-10, 9-12, 13-16		2128 5128	2K x 8 RAM 2K x 8 RAM
1-4, 2-3, 5-6, 7-10, 9-12, 13-16, 18-21, 19-20		5188	8K x 8 RAM
1-4, 7-10, 9-12, 13-16, 17-20		2516 2716	2K x 8 EPROM 2K x 8 EPROM
1-4, 9-12, 10-13, 16-19, 17-20		2532	4K x 8 EPROM
1-4, 7-10, 9-12, 13-16, 19-20		2732	4K x 8 EPROM
1-4, 3-6, 7-10, 8-9, 13-16, 14-17, 18-21, 19-20		2764	8K x 8 EPROM
1-4, 3-6, 7-10, 8-9, 12-15, 13-16, 14-17, 18-21, 19-20		27128	16K x 8 EPROM
1-4, 3-6, 7-10, 8-11, 12-15, 13-16, 14-17, 18-21, 19-20		27256	32K x 8 EPROM

3.4.9. Local ROM Access Time

The jumper area K4 is used to select the number of wait cycles inserted by the MPU when accessing local ROM. K4 must be configured such that the timing of a read operation from local ROM meets the requirements of the slowest ROM device installed in the memory sockets. For each jumper position on K4, Table 3.12 lists the maximum output delay times of the ROM devices that can be tolerated for proper operation.

More detailed timing specifications of the local memory access are given in Paragraph 2.12.

Original configuration: 3 wait cycles inserted

Figure 3.15: Jumper Area K4

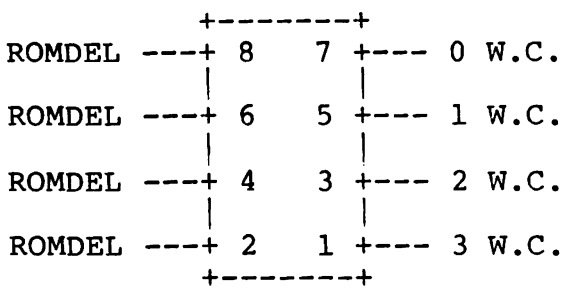


Table 3.12: Local ROM Access Time Selection

K4 CONNECTIONS	WAIT CYCLES	MAXIMUM ROM DELAY TIMES
1-2	3	Addr. Valid to Data Valid max. 665 ns OE* Low to Data Valid max. 600 ns CS* Low to Data Valid max. 530 ns
3-4	2	Addr. Valid to Data Valid max. 540 ns OE* Low to Data Valid max. 475 ns CS* Low to Data Valid max. 405 ns
5-6	1	Addr. Valid to Data Valid max. 415 ns OE* Low to Data Valid max. 350 ns CS* Low to Data Valid max. 280 ns
7-8	0	Addr. Valid to Data Valid max. 290 ns OE* Low to Data Valid max. 225 ns CS* Low to Data Valid max. 155 ns

3.4.10. Address Map Configuration

The original configuration of the MVME101 address map, as shipped from the factory, is shown in Table 2.5. If this map does not meet the requirements of the actual application, a new Address Decoder PROM must be programmed according to the demands. For a good comprehension of the following procedure, the user should be familiar with the functional description of the Address Decoder in Paragraph 2.7.

3.4.10.1. Local Memory Addresses

After the user has selected the RAM and ROM devices to be used for local memory, the addresses to be contained within each memory socket pair must be specified. To avoid address swapping, the base address of each memory pair must reside on the correct boundary. These boundaries are integer multiples of the memory pair size. Each memory pair occupies an address range of twice the size of a single device. Table 3.14 can assist in the selection of local memory base addresses for the various sizes of memory devices. The position of the local memories may be registered in the personal address map in Table 3.16.

3.4.10.2. Local I/O Addresses

The local I/O-devices occupy one 4K bytes segment in the address map. Any 4K boundary in Table 3.14 may be specified as the base address of the I/O-registers. After the user has selected this base address, he may obtain his personal I/O-register address map by using Table 3.17 and adding the chosen base address to the values listed in the ADDRESS column. The position of the local I/O-devices segment may be registered in the personal address map in Table 3.16.

3.4.10.3. VMEbus Short I/O Addresses

When I/O-modules using the address modifier code for Short I/O Address are installed in the system, an address field of 64K bytes must be reserved in the address map for accessing them. Such modules decode only the address lines A01 - A15 on the VMEbus, i.e. a 64K address range, when they are enabled by the address modifier lines. The user may specify any 64K boundary in Table 3.14 as the base address of these global I/O devices. Their addresses in the MVME101 memory map, as seen from the MPU, can then be calculated by adding the selected base address as an offset to their 16-bit addresses. The position of the Short I/O Address field may be registered in the personal address map in Table 3.16.

3.4.10.4. VMEbus Standard Addresses

All address segments in the Lo Block and in the Hi Block which are not selected as local memory, local I/O, or VMEbus short I/O addresses, should be specified as VMEbus Standard Addresses in the personal address map. By that, on-board and off-board address fields for RAM, ROM, and memory-mapped I/O-devices may be contiguously allocated.

3.4.10.5. Address Decoder PROM Programming

After the user has configured his personal address map, he must specify the contents of the Address Decoder PROM. This PROM is organized as 512 x 4 bits. The PROM locations 000 - 0FF represent the Lo Block, the locations 100 - 1FF represent the Hi Block of the address map. Each PROM location corresponds to one 4K bytes address segment.

For each of these 512 address segments, the Address Decoder PROM must be programmed to define which device is selected. These devices may be either local RAM or local ROM in one of the four memory socket pairs, or local I/O-devices, or VMEbus I/O modules responding to short I/O addresses, or VMEbus modules responding to standard addresses. The PROM defines the devices using the encoding scheme shown in Table 3.13.

To determine the data to be recorded in each PROM location, the user refers to his personal address map in Table 3.16 and specifies for each MPU address segment in Table 3.15 the device to be selected by entering the appropriate hexadecimal code number of Table 3.13.

The Address Decoder PROM may be a Signetics N82S130 or any electrically and physically compatible bipolar PROM. For proper operation, the maximum address access time of the used PROM must not exceed 50 ns, the maximum chip select access time must not exceed 30 ns.

Table 3.13: Address Decoder PROM Data Definition

PROM DATA	SELECTED DEVICES
0	Local RAM in socket pair 1
1	Local RAM in socket pair 2
2	Local RAM in socket pair 3
3	Invalid
4	Local ROM in socket pair 1
5	Local ROM in socket pair 2
6	Local ROM in socket pair 3
7	Local ROM in socket pair 4
8	Local I/O-devices
9	Invalid
A	Invalid
B	Invalid
C	Invalid
D	Invalid
E	VMEbus Short I/O Address
F	VMEbus Standard Address

Table 3.14: Address Boundaries

ADDRESS FIELD SIZE					BOUNDARY
				4K bytes	FxF000
			8K bytes	4K bytes	FxE000
				4K bytes	FxD000
		16K bytes	8K bytes	4K bytes	FxC000
				4K bytes	FxB000
			8K bytes	4K bytes	FxA000
				4K bytes	Fx9000
	32K bytes	16K bytes	8K bytes	4K bytes	Fx8000
				4K bytes	Fx7000
			8K bytes	4K bytes	Fx6000
				4K bytes	Fx5000
		16K bytes	8K bytes	4K bytes	Fx4000
				4K bytes	Fx3000
			8K bytes	4K bytes	Fx2000
				4K bytes	Fx1000
64K bytes	32K bytes	16K bytes	8K bytes	4K bytes	Fx0000
				4K bytes	0xF000
			8K bytes	4K bytes	0xE000
				4K bytes	0xD000
		16K bytes	8K bytes	4K bytes	0xC000
				4K bytes	0xB000
			8K bytes	4K bytes	0xA000
				4K bytes	0x9000
	32K bytes	16K bytes	8K bytes	4K bytes	0x8000
				4K bytes	0x7000
			8K bytes	4K bytes	0x6000
				4K bytes	0x5000
		16K bytes	8K bytes	4K bytes	0x4000
				4K bytes	0x3000
			8K bytes	4K bytes	0x2000
				4K bytes	0x1000
64K bytes	32K bytes	16K bytes	8K bytes	4K bytes	0x0000

Note: The letter "x" in the BOUNDARY column may have any hex value.

Table 3.15: Address Decoder PROM Specification

MPU ADDR	PROM A	PROM D	!
FFFxxx	1FF		
FFExxx	1FE		
FFDxxx	1FD		
FFCxxx	1FC		
FFBxxx	1FB		
FFAxxx	1FA		
FF9xxx	1F9		
FF8xxx	1F8		
FF7xxx	1F7		
FF6xxx	1F6		
FF5xxx	1F5		
FF4xxx	1F4		
FF3xxx	1F3		
FF2xxx	1F2		
FF1xxx	1F1		
FF0xxx	1F0		
FEFxxx	1EF		
FEExxx	1EE		
FEDxxx	1ED		
FECxxx	1EC		
FEBxxx	1EB		
FEAxxx	1EA		
FE9xxx	1E9		
FE8xxx	1E8		
FE7xxx	1E7		
FE6xxx	1E6		
FE5xxx	1E5		
FE4xxx	1E4		
FE3xxx	1E3		
FE2xxx	1E2		
FE1xxx	1E1		
FE0xxx	1E0		

MPU ADDR	PROM A	PROM D
FDFxxx	1DF	
FDExxx	1DE	
FDDxxx	1DD	
FDCxxx	1DC	
FDBxxx	1DB	
FDAxxx	1DA	
FD9xxx	1D9	
FD8xxx	1D8	
FD7xxx	1D7	
FD6xxx	1D6	
FD5xxx	1D5	
FD4xxx	1D4	
FD3xxx	1D3	
FD2xxx	1D2	
FD1xxx	1D1	
FD0xxx	1D0	
FCFxxx	1CF	
FCExxx	1CE	
FCDxxx	1CD	
FCCxxx	1CC	
FCBxxx	1CB	
FCAxxx	1CA	
FC9xxx	1C9	
FC8xxx	1C8	
FC7xxx	1C7	
FC6xxx	1C6	
FC5xxx	1C5	
FC4xxx	1C4	
FC3xxx	1C3	
FC2xxx	1C2	
FC1xxx	1C1	
FC0xxx	1C0	

MPU ADDR	PROM A	PROM D
FBFxxx	1BF	
FBExxx	1BE	
FBDxxx	1BD	
FBCxxx	1BC	
FBBxxx	1BB	
FBAxxx	1BA	
FB9xxx	1B9	
FB8xxx	1B8	
FB7xxx	1B7	
FB6xxx	1B6	
FB5xxx	1B5	
FB4xxx	1B4	
FB3xxx	1B3	
FB2xxx	1B2	
FB1xxx	1B1	
FB0xxx	1B0	
FAFxxx	1AF	
FAExxx	1AE	
FADxxx	1AD	
FACxxx	1AC	
FABxxx	1AB	
FAAxxx	1AA	
FA9xxx	1A9	
FA8xxx	1A8	
FA7xxx	1A7	
FA6xxx	1A6	
FA5xxx	1A5	
FA4xxx	1A4	
FA3xxx	1A3	
FA2xxx	1A2	
FA1xxx	1A1	
FA0xxx	1A0	

MPU ADDR	PROM A	PROM D
F9Fxxx	19F	
F9Exxx	19E	
F9Dxxx	19D	
F9Cxxx	19C	
F9Bxxx	19B	
F9Axxx	19A	
F99xxx	199	
F98xxx	198	
F97xxx	197	
F96xxx	196	
F95xxx	195	
F94xxx	194	
F93xxx	193	
F92xxx	192	
F91xxx	191	
F90xxx	190	
F8Fxxx	18F	
F8Exxx	18E	
F8Dxxx	18D	
F8Cxxx	18C	
F8Bxxx	18B	
F8Axxx	18A	
F89xxx	189	
F88xxx	188	
F87xxx	187	
F86xxx	186	
F85xxx	185	
F84xxx	184	
F83xxx	183	
F82xxx	182	
F81xxx	181	
F80xxx	180	

Note: The letter "x" in the MPU ADDR column represents any hex value.

Table 3.15: Address Decoder PROM Specification (cont'd)

MPU ADDR	PROM A	PROM D	MPU ADDR	PROM A	PROM D	MPU ADDR	PROM A	PROM D	MPU ADDR	PROM A	PROM D
F7Fxxx	17F		F5Fxxx	15F		F3Fxxx	13F		F1Fxxx	11F	
F7Exxx	17E		F5Exxx	15E		F3Exxx	13E		F1Exxx	11E	
F7Dxxx	17D		F5Dxxx	15D		F3Dxxx	13D		F1Dxxx	11D	
F7Cxxx	17C		F5Cxxx	15C		F3Cxxx	13C		F1Cxxx	11C	
F7Bxxx	17B		F5Bxxx	15B		F3Bxxx	13B		F1Bxxx	11B	
F7Axxx	17A		F5Axxx	15A		F3Axxx	13A		F1Axxx	11A	
F79xxx	179		F59xxx	159		F39xxx	139		F19xxx	119	
F78xxx	178		F58xxx	158		F38xxx	138		F18xxx	118	
F77xxx	177		F57xxx	157		F37xxx	137		F17xxx	117	
F76xxx	176		F56xxx	156		F36xxx	136		F16xxx	116	
F75xxx	175		F55xxx	155		F35xxx	135		F15xxx	115	
F74xxx	174		F54xxx	154		F34xxx	134		F14xxx	114	
F73xxx	173		F53xxx	153		F33xxx	133		F13xxx	113	
F72xxx	172		F52xxx	152		F32xxx	132		F12xxx	112	
F71xxx	171		F51xxx	151		F31xxx	131		F11xxx	111	
F70xxx	170		F50xxx	150		F30xxx	130		F10xxx	110	
F6Fxxx	16F		F4Fxxx	14F		F2Fxxx	12F		F0Fxxx	10F	
F6Exxx	16E		F4Exxx	14E		F2Exxx	12E		F0Exxx	10E	
F6Dxxx	16D		F4Dxxx	14D		F2Dxxx	12D		F0Dxxx	10D	
F6Cxxx	16C		F4Cxxx	14C		F2Cxxx	12C		F0Cxxx	10C	
F6Bxxx	16B		F4Bxxx	14B		F2Bxxx	12B		F0Bxxx	10B	
F6Axxx	16A		F4Axxx	14A		F2Axxx	12A		F0Axxx	10A	
F69xxx	169		F49xxx	149		F29xxx	129		F09xxx	109	
F68xxx	168		F48xxx	148		F28xxx	128		F08xxx	108	
F67xxx	167		F47xxx	147		F27xxx	127		F07xxx	107	
F66xxx	166		F46xxx	146		F26xxx	126		F06xxx	106	
F65xxx	165		F45xxx	145		F25xxx	125		F05xxx	105	
F64xxx	164		F44xxx	144		F24xxx	124		F04xxx	104	
F63xxx	163		F43xxx	143		F23xxx	123		F03xxx	103	
F62xxx	162		F42xxx	142		F22xxx	122		F02xxx	102	
F61xxx	161		F41xxx	141		F21xxx	121		F01xxx	101	
F60xxx	160		F40xxx	140		F20xxx	120		F00xxx	100	

Note: The letter "x" in the MPU ADDR column represents any hex value.

Table 3.15: Address Decoder PROM Specification (cont'd)

MPU ADDR	PROM A	PROM D	!
0FFxxx	0FF		
0FExxx	0FE		
0FDxxx	0FD		
0FCxxx	0FC		
0FBxxx	0FB		
0FAxxx	0FA		
0F9xxx	0F9		
0F8xxx	0F8		
0F7xxx	0F7		
0F6xxx	0F6		
0F5xxx	0F5		
0F4xxx	0F4		
0F3xxx	0F3		
0F2xxx	0F2		
0F1xxx	0F1		
0F0xxx	0F0		
0EFxxx	0EF		
0EExxx	0EE		
0EDxxx	0ED		
0ECxxx	0EC		
0EBxxx	0EB		
0EAxxx	0EA		
0E9xxx	0E9		
0E8xxx	0E8		
0E7xxx	0E7		
0E6xxx	0E6		
0E5xxx	0E5		
0E4xxx	0E4		
0E3xxx	0E3		
0E2xxx	0E2		
0E1xxx	0E1		
0E0xxx	0E0		

MPU ADDR	PROM A	PROM D
0DFxxx	0DF	
0DExxx	0DE	
0DDxxx	0DD	
0DCxxx	0DC	
0DBxxx	0DB	
0DAxxx	0DA	
0D9xxx	0D9	
0D8xxx	0D8	
0D7xxx	0D7	
0D6xxx	0D6	
0D5xxx	0D5	
0D4xxx	0D4	
0D3xxx	0D3	
0D2xxx	0D2	
0D1xxx	0D1	
0D0xxx	0D0	
0CFxxx	0CF	
0CExxx	0CE	
0CDxxx	0CD	
0CCxxx	0CC	
0CBxxx	0CB	
0CAxxx	0CA	
0C9xxx	0C9	
0C8xxx	0C8	
0C7xxx	0C7	
0C6xxx	0C6	
0C5xxx	0C5	
0C4xxx	0C4	
0C3xxx	0C3	
0C2xxx	0C2	
0C1xxx	0C1	
0C0xxx	0C0	

MPU ADDR	PROM A	PROM D
0BFxxx	0BF	
0BExxx	0BE	
0BDxxx	0BD	
0BCxxx	0BC	
0BBxxx	0BB	
0BAxxx	0BA	
0B9xxx	0B9	
0B8xxx	0B8	
0B7xxx	0B7	
0B6xxx	0B6	
0B5xxx	0B5	
0B4xxx	0B4	
0B3xxx	0B3	
0B2xxx	0B2	
0B1xxx	0B1	
0B0xxx	0B0	
0AFxxx	0AF	
0AExxx	0AE	
0ADxxx	0AD	
0ACxxx	0AC	
0ABxxx	0AB	
0AAxxx	0AA	
0A9xxx	0A9	
0A8xxx	0A8	
0A7xxx	0A7	
0A6xxx	0A6	
0A5xxx	0A5	
0A4xxx	0A4	
0A3xxx	0A3	
0A2xxx	0A2	
0A1xxx	0A1	
0A0xxx	0A0	

MPU ADDR	PROM A	PROM D
09Fxxx	09F	
09Exxx	09E	
09Dxxx	09D	
09Cxxx	09C	
09Bxxx	09B	
09Axxx	09A	
099xxx	099	
098xxx	098	
097xxx	097	
096xxx	096	
095xxx	095	
094xxx	094	
093xxx	093	
092xxx	092	
091xxx	091	
090xxx	090	
08Fxxx	08F	
08Exxx	08E	
08Dxxx	08D	
08Cxxx	08C	
08Bxxx	08B	
08Axxx	08A	
089xxx	089	
088xxx	088	
087xxx	087	
086xxx	086	
085xxx	085	
084xxx	084	
083xxx	083	
082xxx	082	
081xxx	081	
080xxx	080	

Note: The letter "x" in the MPU ADDR column represents any hex value.

Table 3.15: Address Decoder PROM Specification (cont'd)

MPU ADDR	PROM A	PROM D	!
07Fxxx	07F		
07Exxx	07E		
07Dxxx	07D		
07Cxxx	07C		
07Bxxx	07B		
07Axxx	07A		
079xxx	079		
078xxx	078		
077xxx	077		
076xxx	076		
075xxx	075		
074xxx	074		
073xxx	073		
072xxx	072		
071xxx	071		
070xxx	070		
06Fxxx	06F		
06Exxx	06E		
06Dxxx	06D		
06Cxxx	06C		
06Bxxx	06B		
06Axxx	06A		
069xxx	069		
068xxx	068		
067xxx	067		
066xxx	066		
065xxx	065		
064xxx	064		
063xxx	063		
062xxx	062		
061xxx	061		
060xxx	060		

MPU ADDR	PROM A	PROM D
05Fxxx	05F	
05Exxx	05E	
05Dxxx	05D	
05Cxxx	05C	
05Bxxx	05B	
05Axxx	05A	
059xxx	059	
058xxx	058	
057xxx	057	
056xxx	056	
055xxx	055	
054xxx	054	
053xxx	053	
052xxx	052	
051xxx	051	
050xxx	050	
04Fxxx	04F	
04Exxx	04E	
04Dxxx	04D	
04Cxxx	04C	
04Bxxx	04B	
04Axxx	04A	
049xxx	049	
048xxx	048	
047xxx	047	
046xxx	046	
045xxx	045	
044xxx	044	
043xxx	043	
042xxx	042	
041xxx	041	
040xxx	040	

MPU ADDR	PROM A	PROM D
03Fxxx	03F	
03Exxx	03E	
03Dxxx	03D	
03Cxxx	03C	
03Bxxx	03B	
03Axxx	03A	
039xxx	039	
038xxx	038	
037xxx	037	
036xxx	036	
035xxx	035	
034xxx	034	
033xxx	033	
032xxx	032	
031xxx	031	
030xxx	030	
02Fxxx	02F	
02Exxx	02E	
02Dxxx	02D	
02Cxxx	02C	
02Bxxx	02B	
02Axxx	02A	
029xxx	029	
028xxx	028	
027xxx	027	
026xxx	026	
025xxx	025	
024xxx	024	
023xxx	023	
022xxx	022	
021xxx	021	
020xxx	020	

MPU ADDR	PROM A	PROM D
01Fxxx	01F	
01Exxx	01E	
01Dxxx	01D	
01Cxxx	01C	
01Bxxx	01B	
01Axxx	01A	
019xxx	019	
018xxx	018	
017xxx	017	
016xxx	016	
015xxx	015	
014xxx	014	
013xxx	013	
012xxx	012	
011xxx	011	
010xxx	010	
00Fxxx	00F	
00Exxx	00E	
00Dxxx	00D	
00Cxxx	00C	
00Bxxx	00B	
00Axxx	00A	
009xxx	009	
008xxx	008	
007xxx	007	
006xxx	006	
005xxx	005	
004xxx	004	
003xxx	003	
002xxx	002	
001xxx	001	
000xxx	000	

Note: The letter "x" in the MPU ADDR column represents any hex value.

Table 3.16: Personal Address Map

ADDRESS	CONTENTS	SELECTED DEVICES
FFFFFF : F.....		
F..... : F.....		
F..... : F.....		
F..... : F.....		
F..... : F.....		
F..... : F00000		
EFFFFF : : : 100000	VMEbus Standard Addresses	Global Memory or Memory-mapped Devices
0FFFFF : 0.....		
0..... : 0.....		
0..... : 0.....		
0..... : 0.....		
0..... : 000400		
0003FF : 000000	MPU Exception Vectors	

Table 3.17: Personal I/O-Register Address Map

DEVICE	ADDRESS	MODE	REGISTER
MCR	...0F1	r/w	Module Control Register
MSR	...0E1	r/w	Module Status Register
PTM	...0DF	read	LSB buffer register
	...0DF	write	Timer #3 latches
	...0DD	read	Timer #3 counter
	...0DD	write	MSB buffer register
	...0DB	read	LSB buffer register
	...0DB	write	Timer #2 latches
	...0D9	read	Timer #2 counter
	...0D9	write	MSB buffer register
	...0D7	read	LSB buffer register
	...0D7	write	Timer #1 latches
	...0D5	read	Timer #1 counter
	...0D5	write	MSB buffer register
	...0D3	read	status register
	...0D3	write	control register #2
	...0D1	read	no operation
	...0D1	write	CR20 = 1: control register #1
	...0D1	write	CR20 = 0: control register #3
PIA	...0C7	r/w	Section B control register
	...0C5	r/w	CRB-2 = 1: Section B peripheral register
	...0C5	r/w	CRB-2 = 0: Section B data direction register
	...0C3	r/w	Section A control register
	...0C1	r/w	CRA-2 = 1: Section A peripheral register
	...0C1	r/w	CRA-2 = 0: Section A data direction register
PCI2	...0B7	r/w	command register
	...0B5	r/w	mode register #1 / mode register #2
	...0B3	read	status register
	...0B3	write	SYN1 register / SYN2 register / DLE register
	...0B1	read	receive holding register
	...0B1	write	transmit holding register
PCI1	...0A7	r/w	command register
	...0A5	r/w	mode register #1 / mode register #2
	...0A3	read	status register
	...0A3	write	SYN1 register / SYN2 register / DLE register
	...0A1	read	receive holding register
	...0A1	write	transmit holding register

3.5. SOFTWARE INITIALIZATION

In the reset routine, the user has to provide routines for initializing the Serial Communication Interfaces, the Peripheral Interface Adapter, the Programmable Timer Module and the Module Control Register of the MVME101 monoboard computer.

3.5.1. Serial Communication Interface Initialization

Prior to transmitting data via the serial ports, the user has to program the MC68661 devices by initializing their mode and command registers and, for synchronous operation, their SYN1, SYN2 and DLE registers. The Motorola MC68661 EPCI Data Sheet in Appendix B provides detailed programming instructions. The original addresses of the EPCI registers are listed in Table 2.6.

3.5.2. Peripheral Interface Adapter Initialization

The direction of the peripheral input/output lines and the function of the peripheral control lines at the connector P2 are controlled by the data direction and control registers of the MC6821 device. The Motorola MC6821 PIA Data Sheet in Appendix C provides detailed programming instructions. The original addresses of the PIA registers are listed in Table 2.6.

3.5.3. Programmable Timer Module Initialization

The initialization of the MC6840 Programmable Timer Module is described in the Motorola MC6840 PTM Data Sheet in Appendix D. The original addresses of the PTM registers are listed in Table 2.6.

3.5.4. Module Control Register Initialization

The Module Control Register controls the hexadecimal status display, the bus block transfer request, and the time out counters. Paragraph 2.6 gives a detailed description of the MCR functions. The device is originally located at address FE00F1.

3.6. INSTALLATION

The MVME101 may be used either as the system controller module in a VMEbus system (System Controller Configuration), or as an MPU module on a selectable priority in a multiprocessor VMEbus system (Standard Configuration), or as an isolated monoboard system that resides only physically on the VMEbus backplane (Isolated Configuration). The hardware preparation for these different modes of operation is described in Paragraph 3.4.

In the Isolated Configuration, the MVME101 module may also be used as a monoboard computer system without a VMEbus backplane. In this case, the power supply voltages must be connected to the respective terminals of P1 by a female DIN 41612 C 96 connector. The location of the power supply inputs at P1 is outlined in Table 2.9.

PRIOR TO INSERTING OR REMOVING THE MVME101 MODULE, ENSURE THAT SYSTEM POWER IS SWITCHED OFF, AS COMPONENTS COULD BE DAMAGED.

At the connector P2 the peripheral input/output signals of the PIA and the PTM are available. Note that these lines are not buffered and do not have any overvoltage protection. Therefore the characteristics of the signals applied at P2 must meet the specifications of the MC6821 and MC6840 devices. In addition to the I/O signals, the +5V power voltage is available at P2 and may be used to supply interface buffers. As the maximum +5V input current at P1 is limited to 1.5 Ampere per terminal by the DIN 41612 connector specifications, the maximum +5V output current at P2 must not exceed 4.5 Amperes minus the MVME101 supply current.

CHAPTER 4

MAINTENANCE INFORMATION

4.1. INTRODUCTION

This chapter provides the parts list, the assembly drawing, and the schematic diagrams for the MVME101 monoboard computer.

4.2. PARTS LIST

Table 4.1 reflects the latest issue of hardware for the MVME101 at the time of printing. The parts locations are shown in Figure 4.1.

Table 4.1: MVME101 Parts List

QU	DESIGNATION	PART NUMBER	DESCRIPTION
2	C1,C2	23-G9618M05	100 uF/ 10 V Electrolytic Capacitor
2	C3,C4	23-G9618M03	22 uF / 35 V Electrolytic Capacitor
4	C34-C37	21NW9604A58	330 pF / 50 V Ceramic Capacitor
37	C5-C33, C38-C45	21NW9702A09	0.1 uF / 50 V Ceramic Capacitor
1	CR	48NW9616A03	1N4148 Rectifier
2	K9,K10	28NW9802D58	Header Single Row / 1 x 6 Pins
1	K16	28NW9802F52	Header Single Row / 1 x 14 Pins
2	K2,K4	28NW9802C43	Header Double Row / 2 x 4 Pins
1	K3	28NW9802C52	Header Double Row / 2 x 5 Pins
1	K6	28NW9802C63	Header Double Row / 2 x 6 Pins
1	K5	28NW9802C36	Header Double Row / 2 x 7 Pins
3	K1,K7,K15	28-G9802M01	Header Double Row / 2 x 8 Pins
4	K11-K14	28NW9802C36 +28NW9802F51	Header Triple Row / 3 x 7 Pins (2 x 7 Pins + 1 x 7 Pins)
1	P1	28-G9802M03	DIN 41612 C 96 Male Connector
1	P2	28-G9802M04	DIN 41612 C 64 Male Connector
2	P3,P4	28-G9802M05	Sub-D 25-pole Female Connector
1	R1	06SW-124A17	47 ohm / 0.25 W Carbon Resistor

Table 4.1: MVME101 Parts List (cont'd)

QU	DESIGNATION	PART NUMBER	DESCRIPTION
2	R4,R5	06SW-124A41	470 ohm / 0.25 W Carbon Resistor
2	R2,R3	06SW-124A43	560 ohm / 0.25 W Carbon Resistor
1	RP4	51NW9626A69	7 x 1.0 kohm SIL Resistor Network
1	RP7	51NW9626A47	7 x 4.7 kohm SIL Resistor Network
5	RP1-RP3, RP5,RP6	51NW9626A49	7 x 10 kohm SIL Resistor Network
2	SW1,SW2	40NW9801B27	Momentary Action Pushbutton Switch
1	at SW1	38NW9404A56	Switch Cap Black
1	at SW2	38NW9404B96	Switch Cap Red
1	U19	48NW9606A33	K1114A 16.000 MHz Crystal Osc.
1	U51	48AW1014B06	K1114A 5.0688 MHz Crystal Osc.
1	U36	51-G5017M01	BAR101B Bus Arbiter/Requester
1	U49	51-G5008M01	MAD101 Address Decoder PROM
1	U48	51NW9615G97	MC68000L8 Microprocessor
2	U52,U60	51NW9615H19	MC68661PC Progr. Comm. Interface
1	U57	51NW9615B27	MC6821P Periph. Interf. Adapter
1	U65	51NW9615D81	MC6840P Programmable Timer Module
2	U50,U59	51NW9615B29	MC1488P Quad RS232 Driver
1	U58	51NW9615B30	MC1489AP Quad RS232 Receiver
1	U12	01NW9804B83	PE21198 Delay Module 50 ns
1	U7	01NW9804B35	PE21199 Delay Module 100 ns
1	U5	01NW9804C33	PE21264 Delay Module 3 x 40 ns
1	U24	51NW9615E91	SN74LS00N Quad 2-NAND Gate
2	U45,U46	51NW9615C22	SN74LS08N Quad 2-AND Gate
1	U29	51NW9615H53	SN74LS09N Quad 2-AND Gate OC
1	U26	51NW9615E88	SN74LS10N Triple 3-NAND Gate
1	U4	51NW9615E93	SN74LS14N Hex Schmitt-Trigger Inv.

Table 4.1: MVME101 Parts List (cont'd)

QU	DESIGNATION	PART NUMBER	DESCRIPTION
3	U18,U27,U30	51NW9615C24	SN74LS32N Quad 2-OR Gate
1	U39	51NW9615C69	SN74LS138N 3-Bit Binary Decoder
1	U44	51NW9615G10	SN74LS148N 8-Bit Priority Encoder
1	U47	51NW9615E86	SN74LS151N 8-Input Multiplexer
2	U2,U23	51NW9615F41	SN74LS164N 8-Bit Shift Register
3	U8,U32,U33	51NW9615F02	SN74LS244N Octal Bus Driver TS
1	U43	51NW9615F09	SN74LS266N Quad 2-EXNOR Gate OC
2	U34,U35	51NW9615F52	SN74LS273N 8-Bit D-Register
2	U25,U31	51NW9615F38	SN74LS393N Dual 4-Bit Bin.Counter
1	U20	51NW9615H83	SN74LS641-1N Octal Bus Transc. OC
5	U9,U10,U22, U37,U38	51NW9615H89	SN74LS645-1N Octal Bus Transc. TS
3	U13,U28,U40	51NW9615C94	SN74S00N Quad 2-NAND Gate
1	U16	51NW9615D32	SN74S02N Quad 2-NOR Gate
1	U14	51NW9615C96	SN74S04N Hex Inverter
1	U17	51NW9615E27	SN74S10N Triple 3-NAND Gate
1	U15	51NW9615D90	SN74S11N Triple 3-AND Gate
1	U42	51NW9615F15	SN74S15N Triple 3-AND Gate OC
1	U6	51NW9615D27	SN74S32N Quad 2-OR Gate
1	U1	51NW9615C95	SN74S74N Dual D-Flip-Flop
1	U41	51NW9615K80	SN74S139N Dual 2-bit Binary Dec.
1	U3	51NW9615J11	SN74S140N Dual 4-NAND Driver
1	U21	51NW9615F65	SN74S241N Octal Bus Driver TS
1	U11	72NW9624A03	TIL311 Hexadecimal LED Display
1	at U48	09NW9811A30	64-Pin DIL IC Socket
1	at U57	09NW9811A22	40-Pin DIL IC Socket
11	at U52-U56, U60-U65	09NW9811A21	28-Pin DIL IC Socket

Table 4.1: MVME101 Parts List (cont'd)

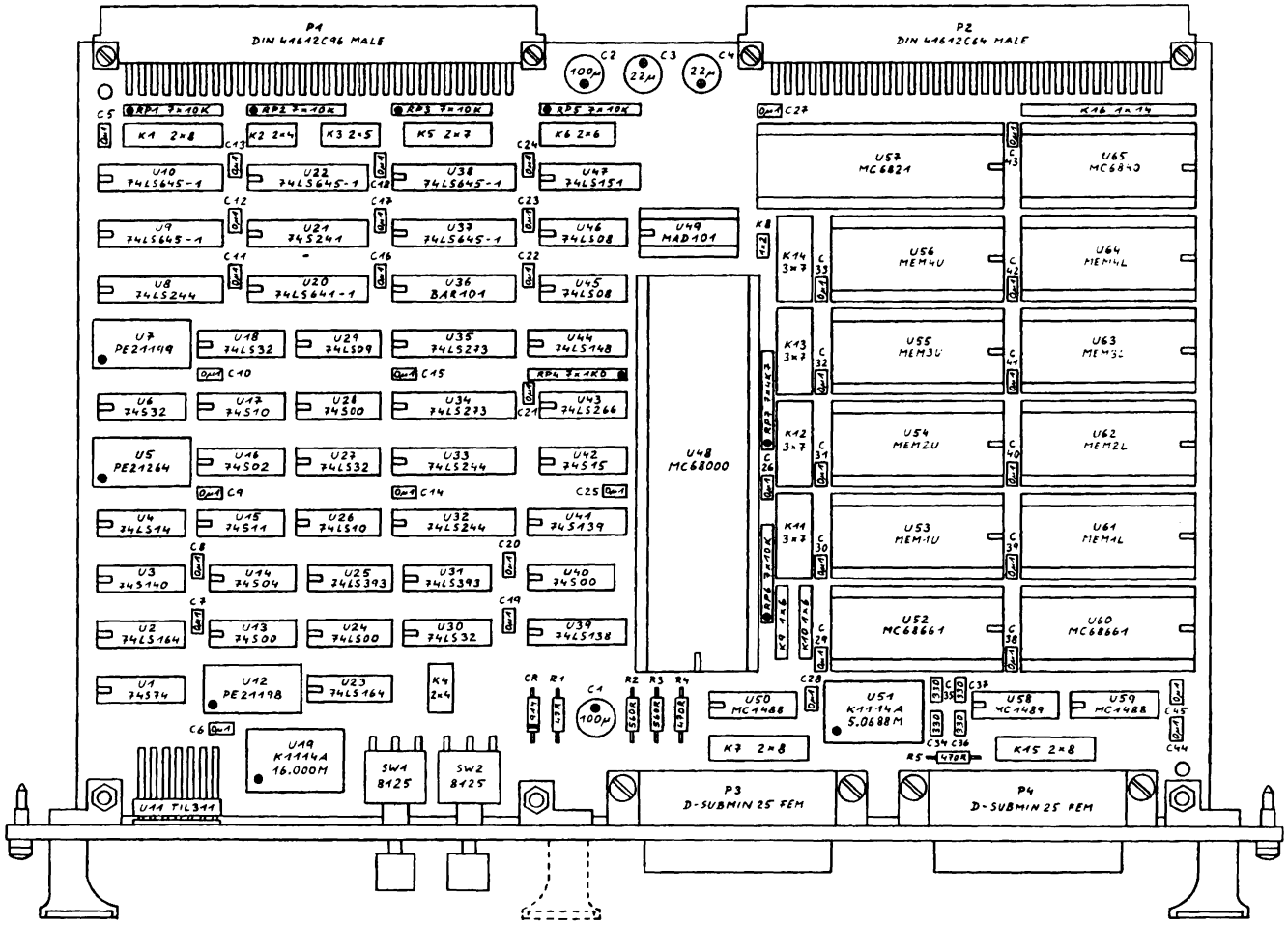
QU	DESIGNATION	PART NUMBER	DESCRIPTION
1	at U49	09NW9811A04	16-Pin DIL IC Socket
1	at U11	09-G9811M01	14-Pin DIL Display Socket
1	at U19,U51	09NW9811A46	4-Pin Oscillator Socket
7	at P1,P2, Front Panel	03SW993D210	DIN 84 M 2.5 x 10 Flat Head Srew
4	at P3,P4	03SW993D310	DIN 84 M 3 X 10 Flat Head Screw
7	at P1,P2, Front Panel	02SW990D001	DIN 934 M 2.5 Hexagonal Nut
4	at P3,P4	02SW990D002	DIN 934 M 3 Hexagonal Nut
1		84-G8012M01	MVME101 Printed Circuit Board
1		64-G4073M01	MVME101 Front Panel
80		29NW9805B17	Jumper

4.3. ASSEMBLY DRAWING, SCHEMATIC DIAGRAMS

The Assembly Drawing in Figure 4.1 shows all part locations on the MVME101 monoboard computer.

The Figures 4.2 to 4.12 show the schematic diagram sheets 1/11 to 11/11.

Figure 4.1: Assembly Drawing



ASSEMBLY DRAWING

MOTOROLA microsystems
Integrated Circuits Division

01AG3012M

Figure 4.2: Schematic Diagram Sheet 1/11

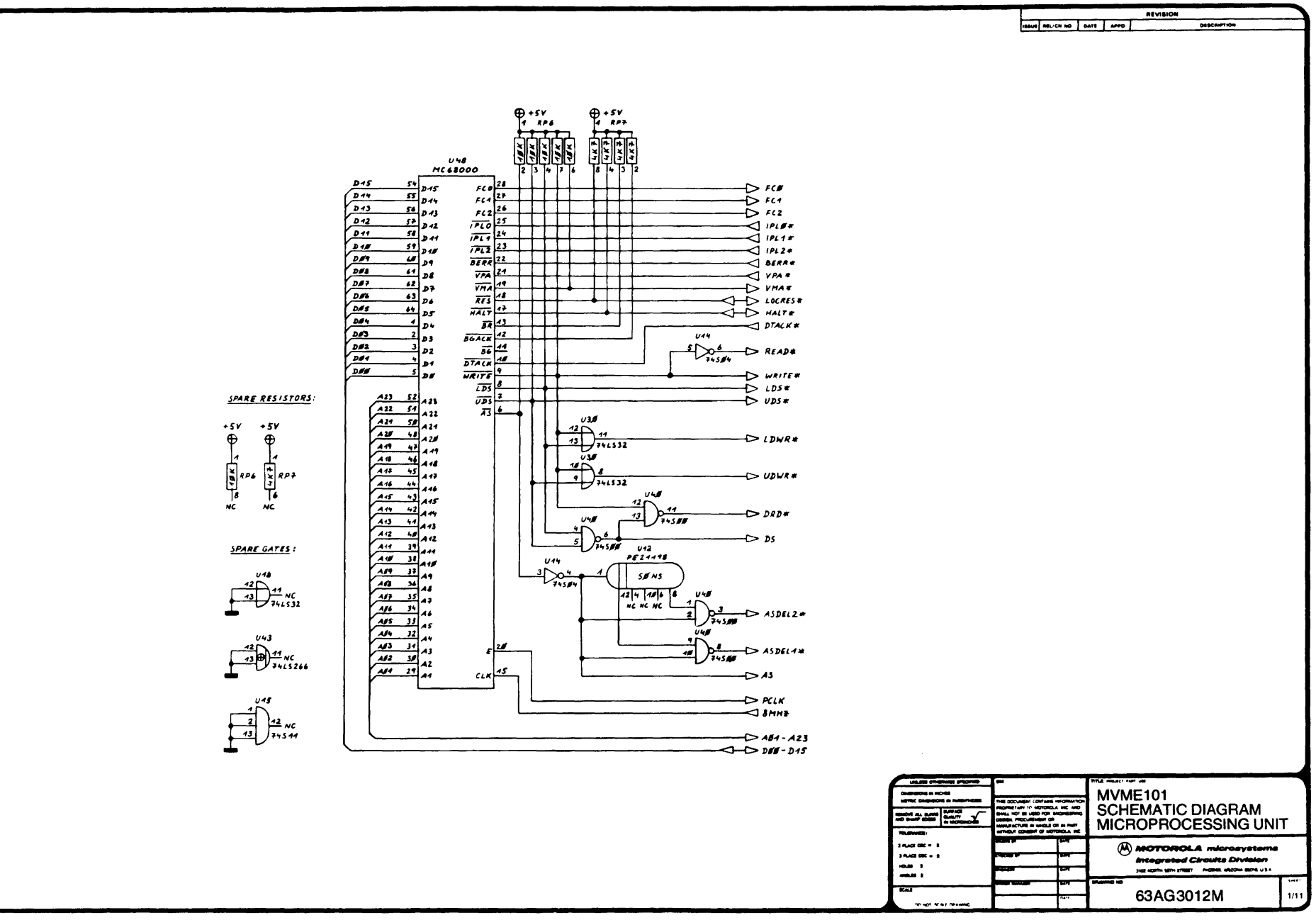


Figure 4.3: Schematic Diagram Sheet 2/11

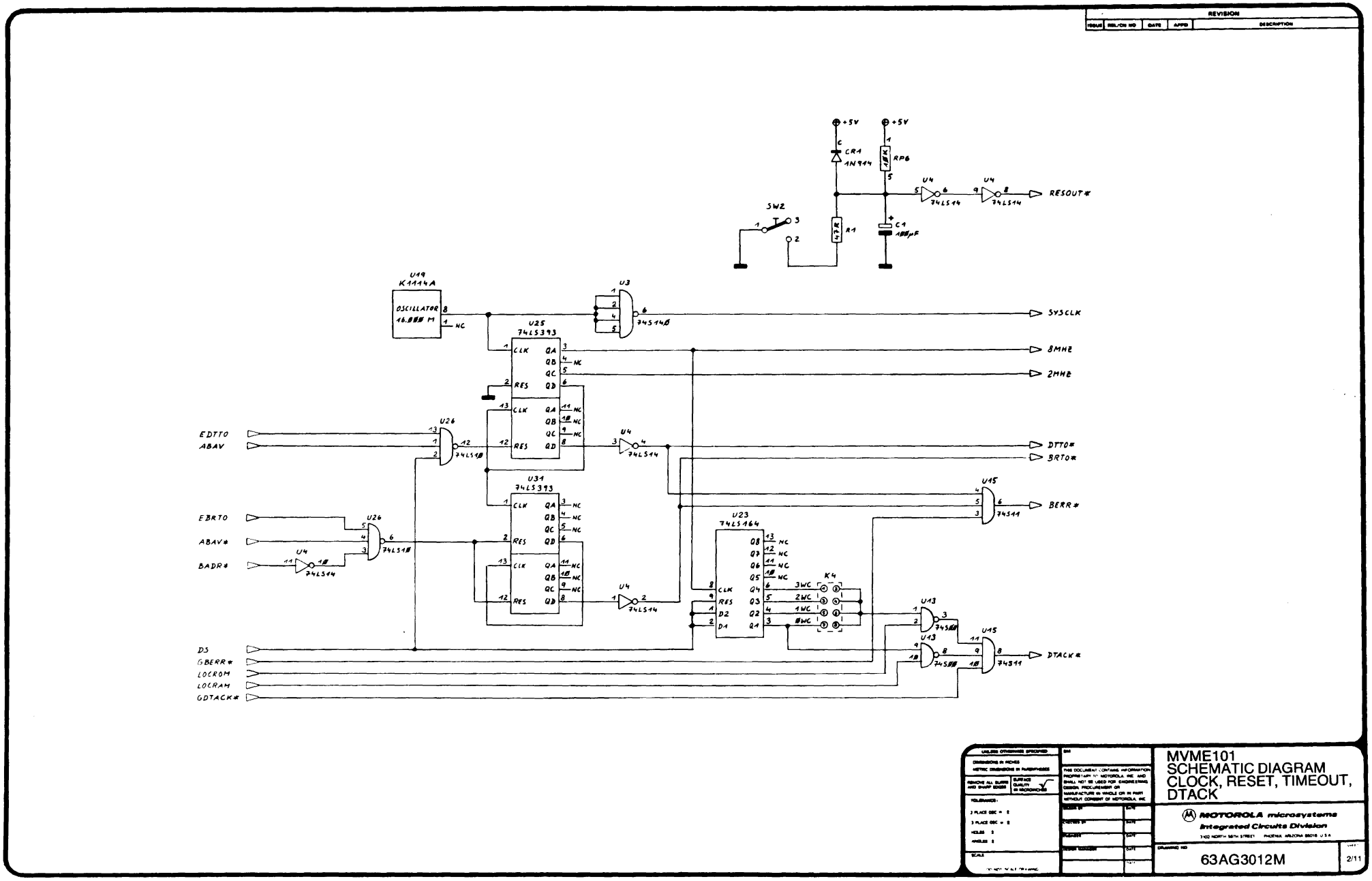


Figure 4.4: Schematic Diagram Sheet 3/11

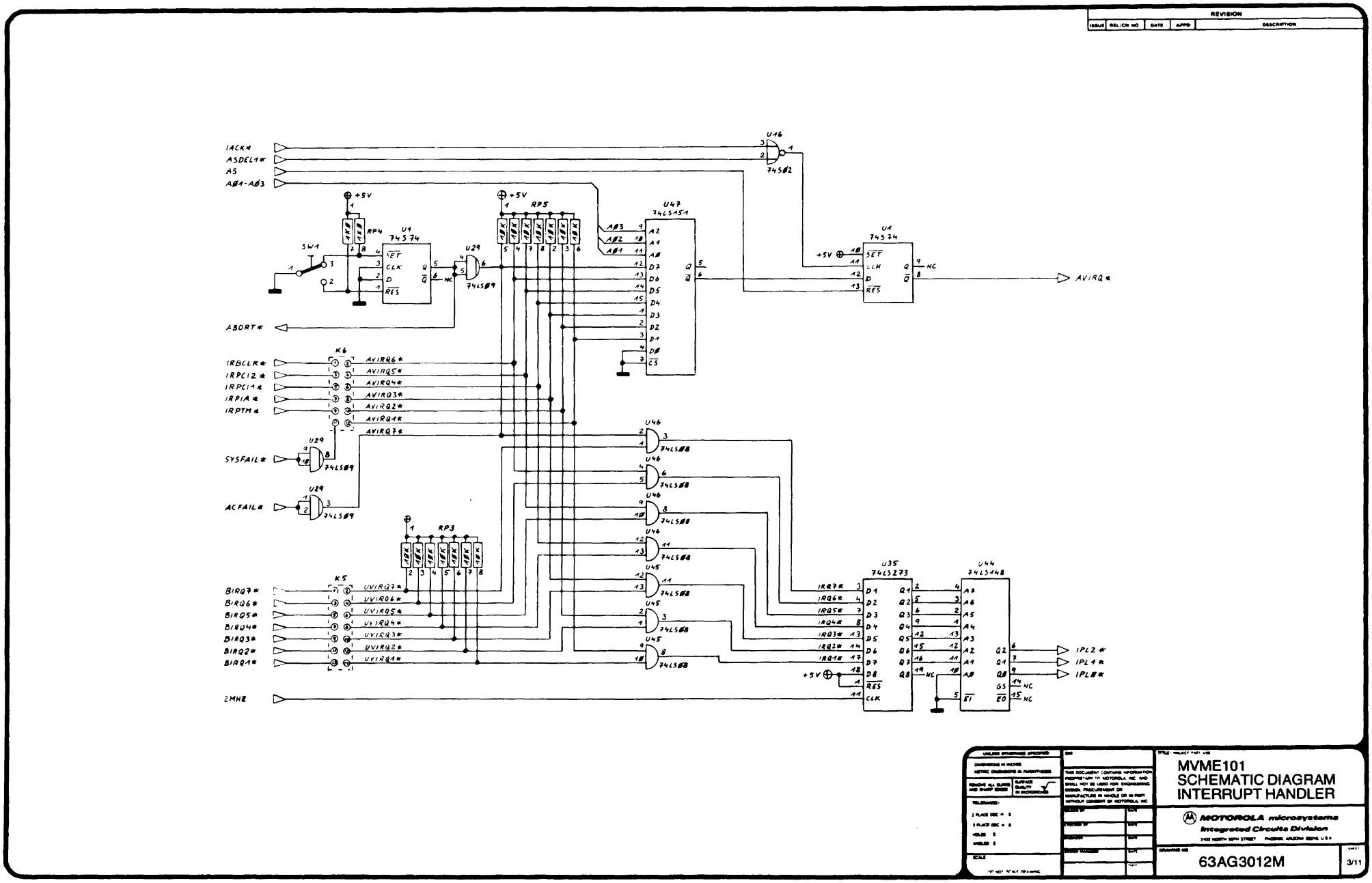


Figure 4.5: Schematic Diagram Sheet 4/11

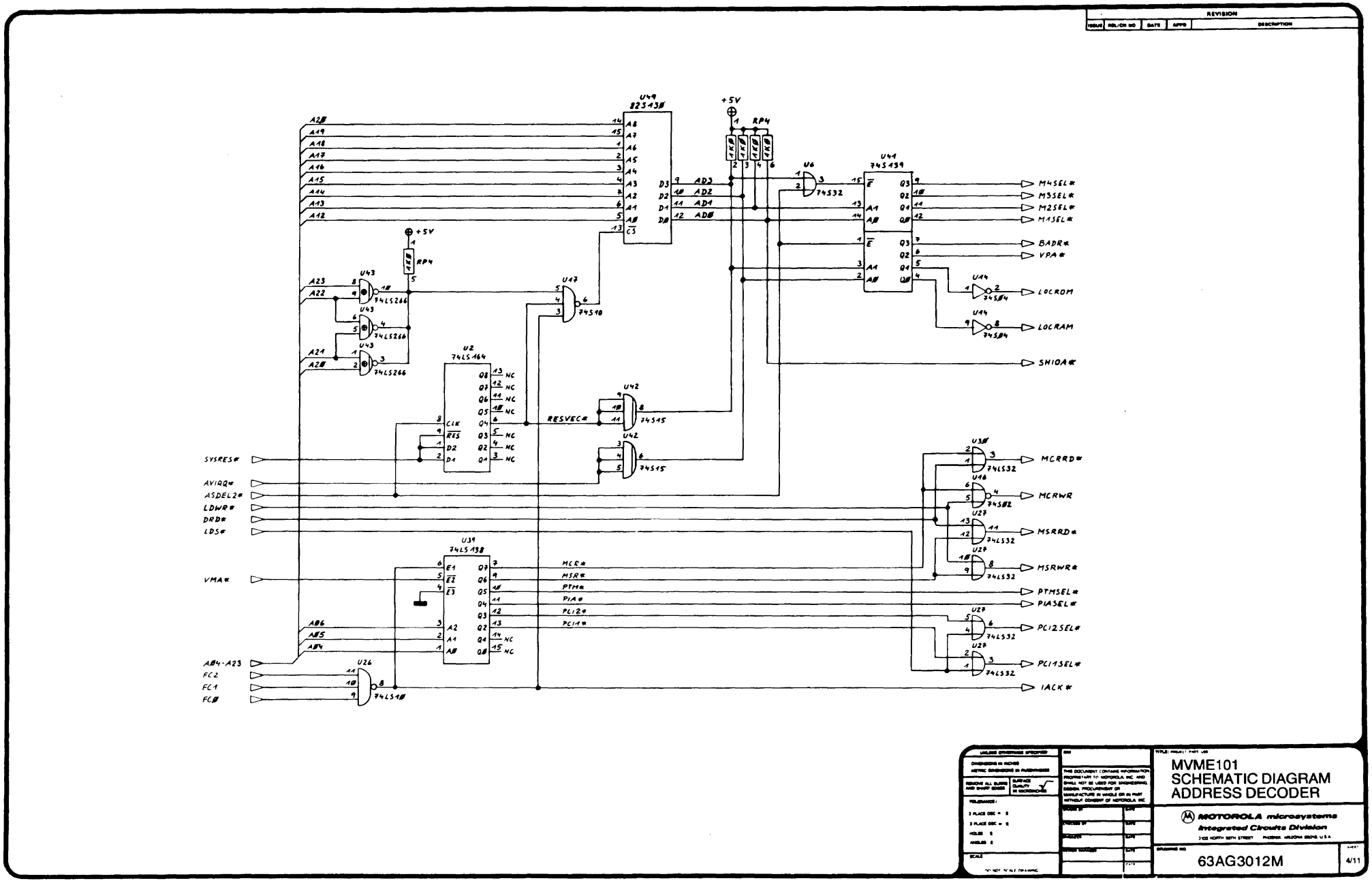


Figure 4.6: Schematic Diagram Sheet 5/11

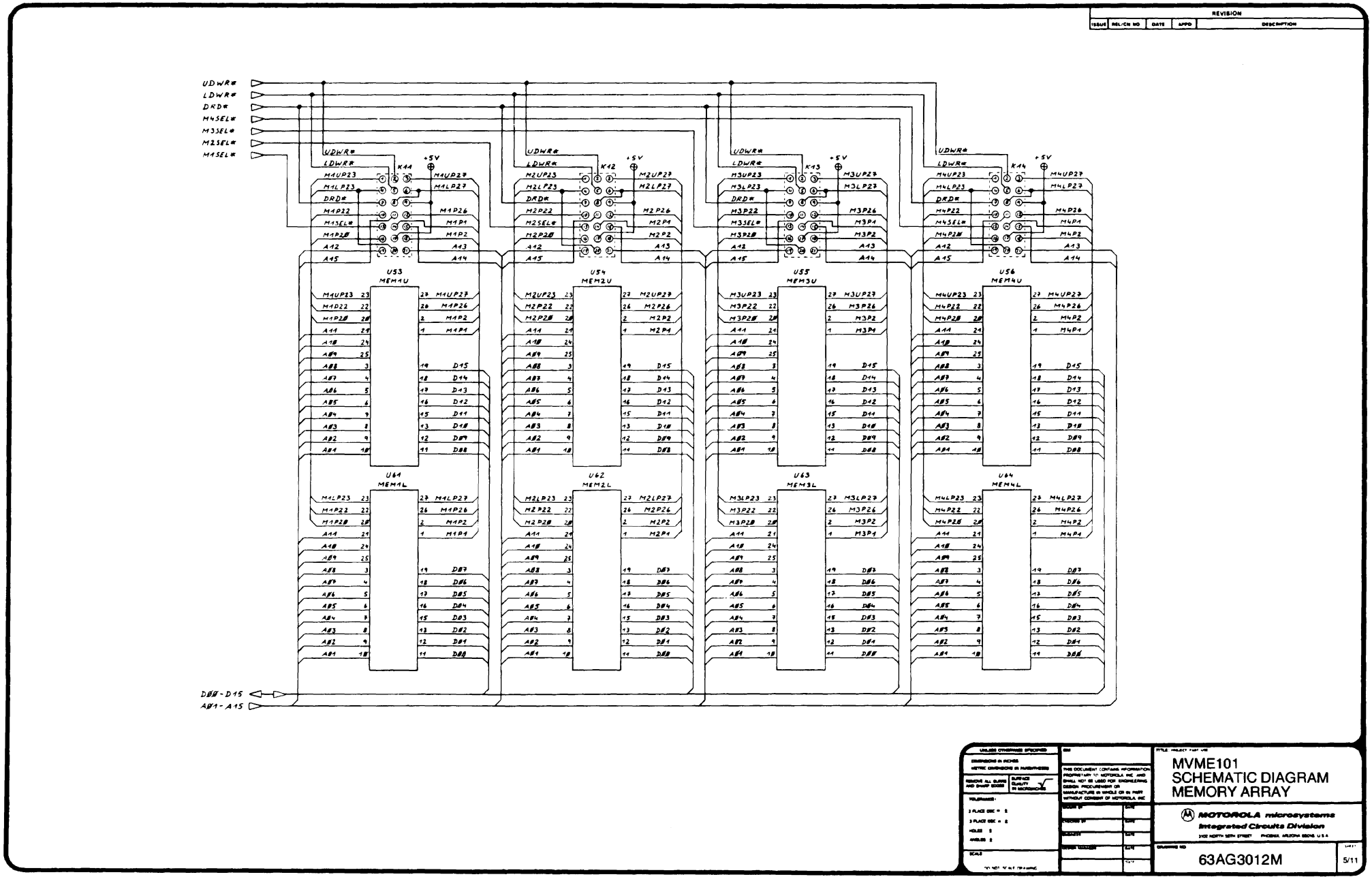


Figure 4.7: Schematic Diagram Sheet 6/11

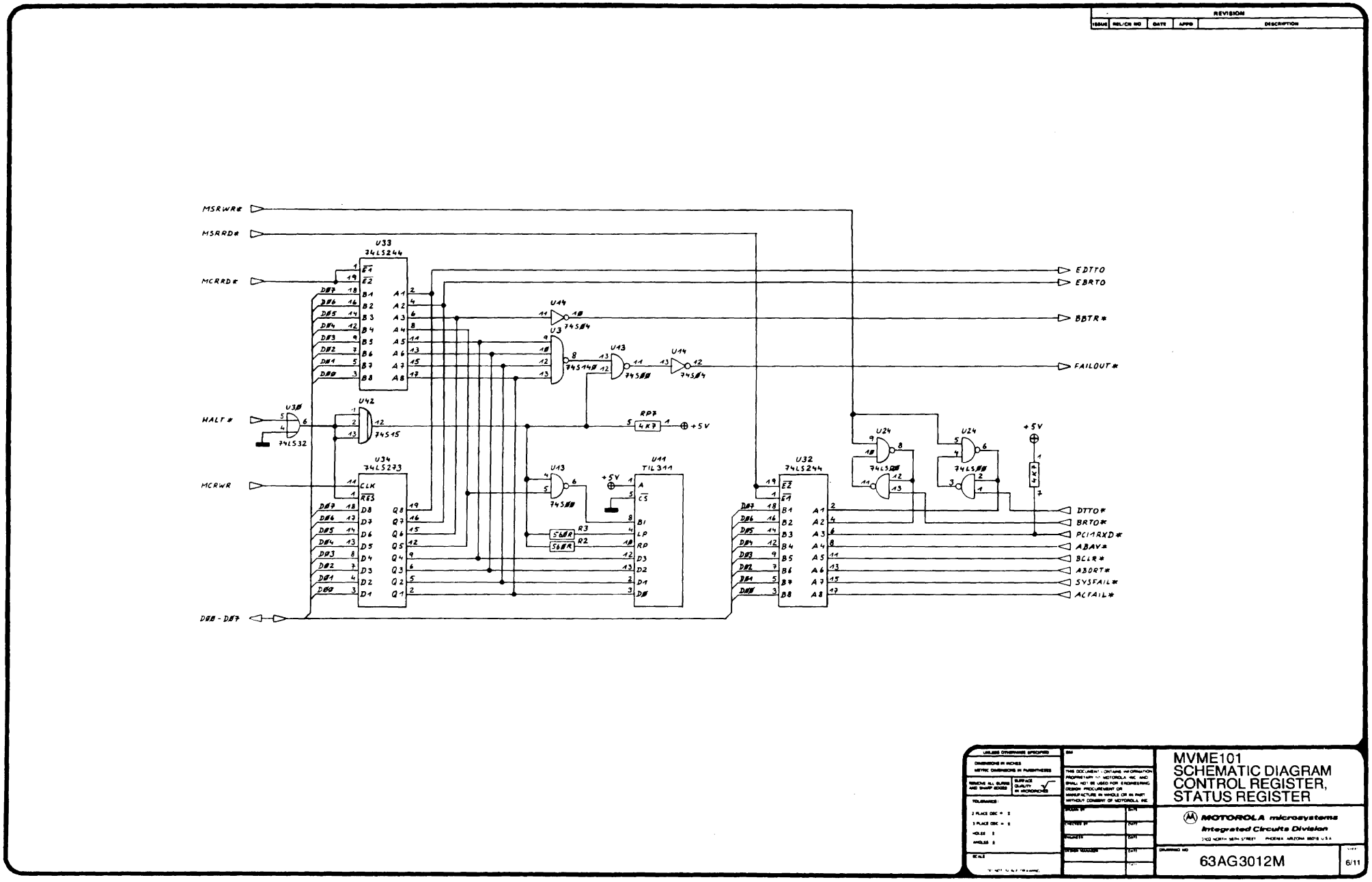


Figure 4.8: Schematic Diagram Sheet 7/11

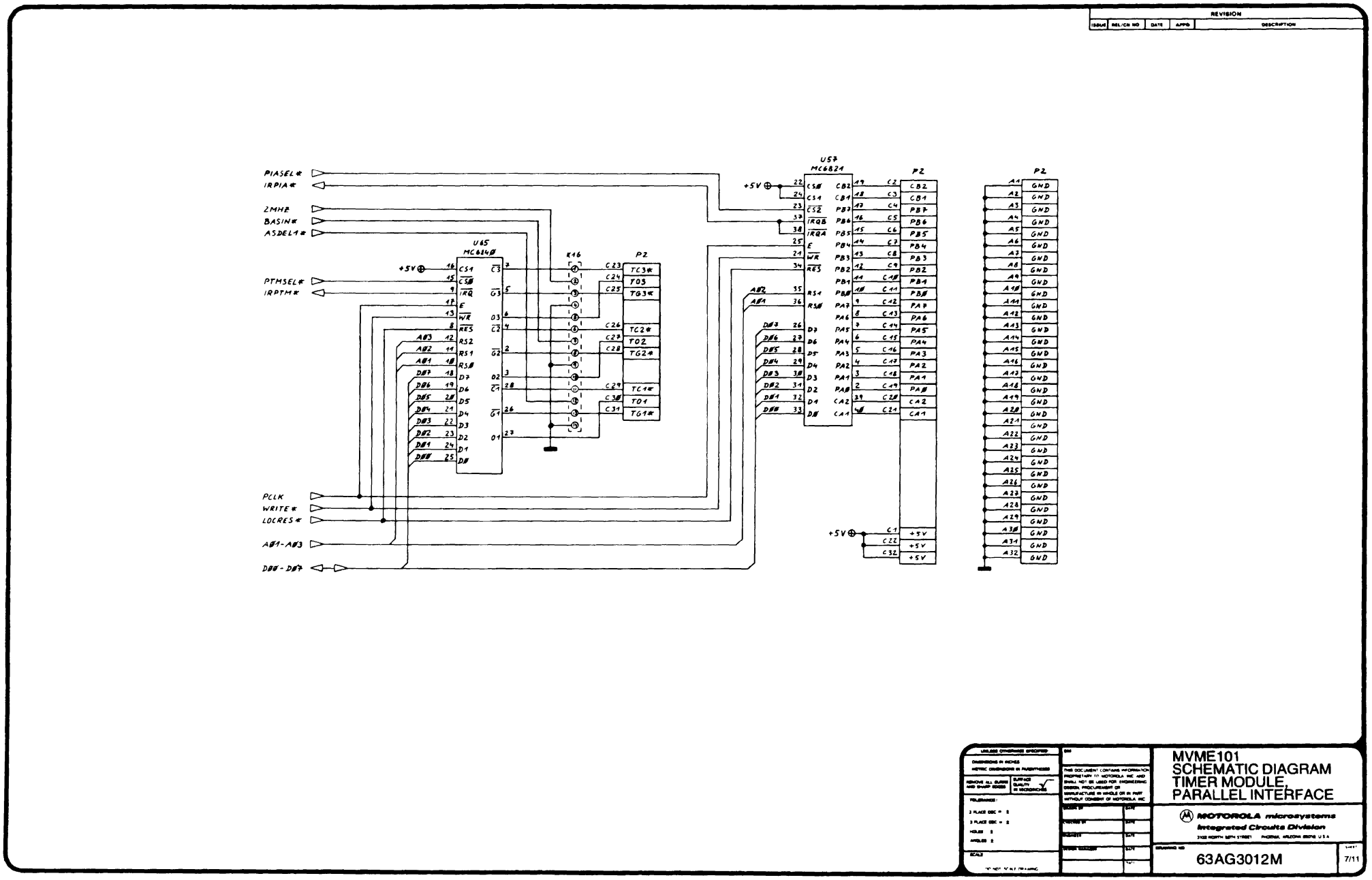


Figure 4.9: Schematic Diagram Sheet 8/11

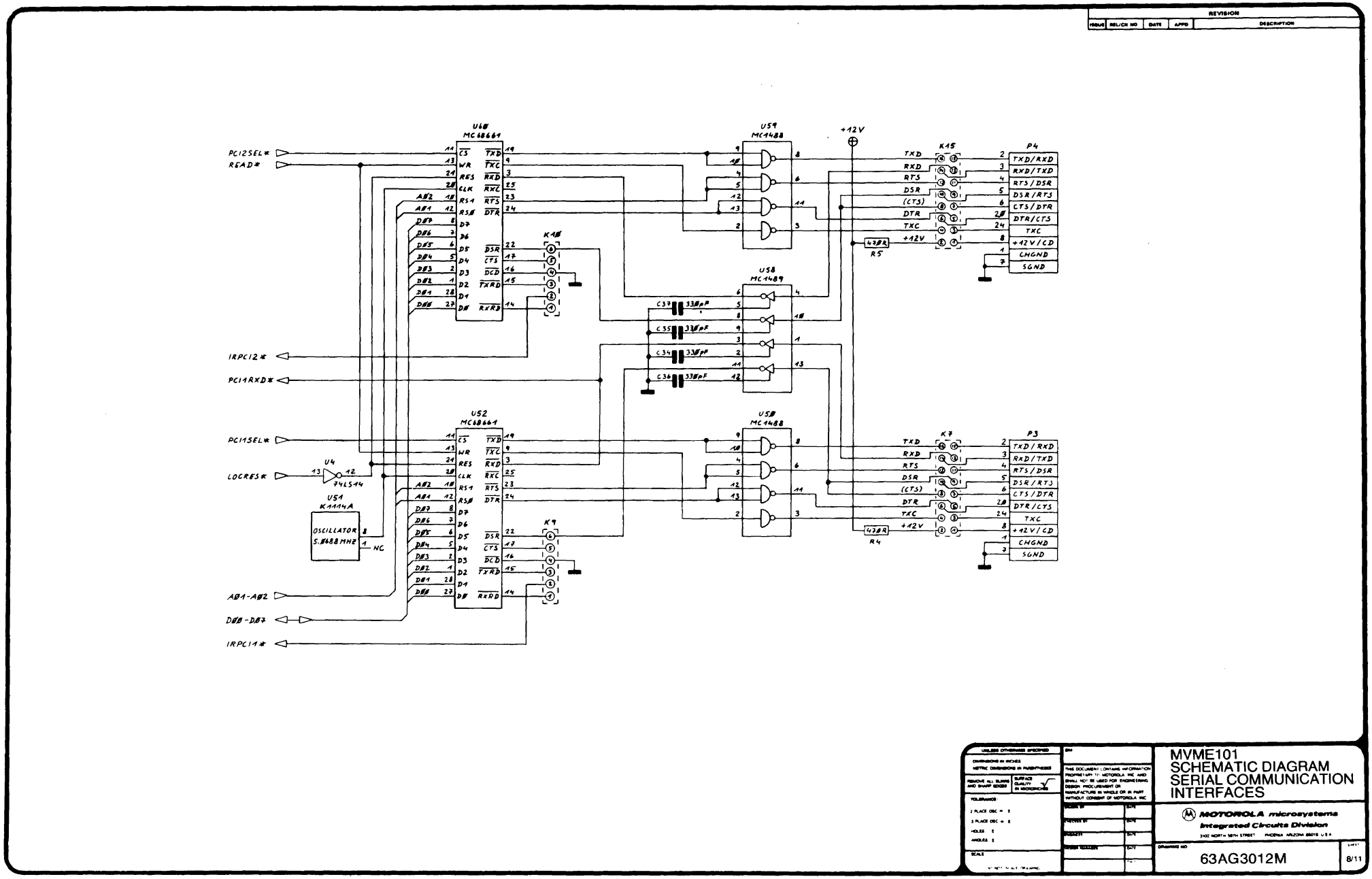


Figure 4.10: Schematic Diagram Sheet 9/11

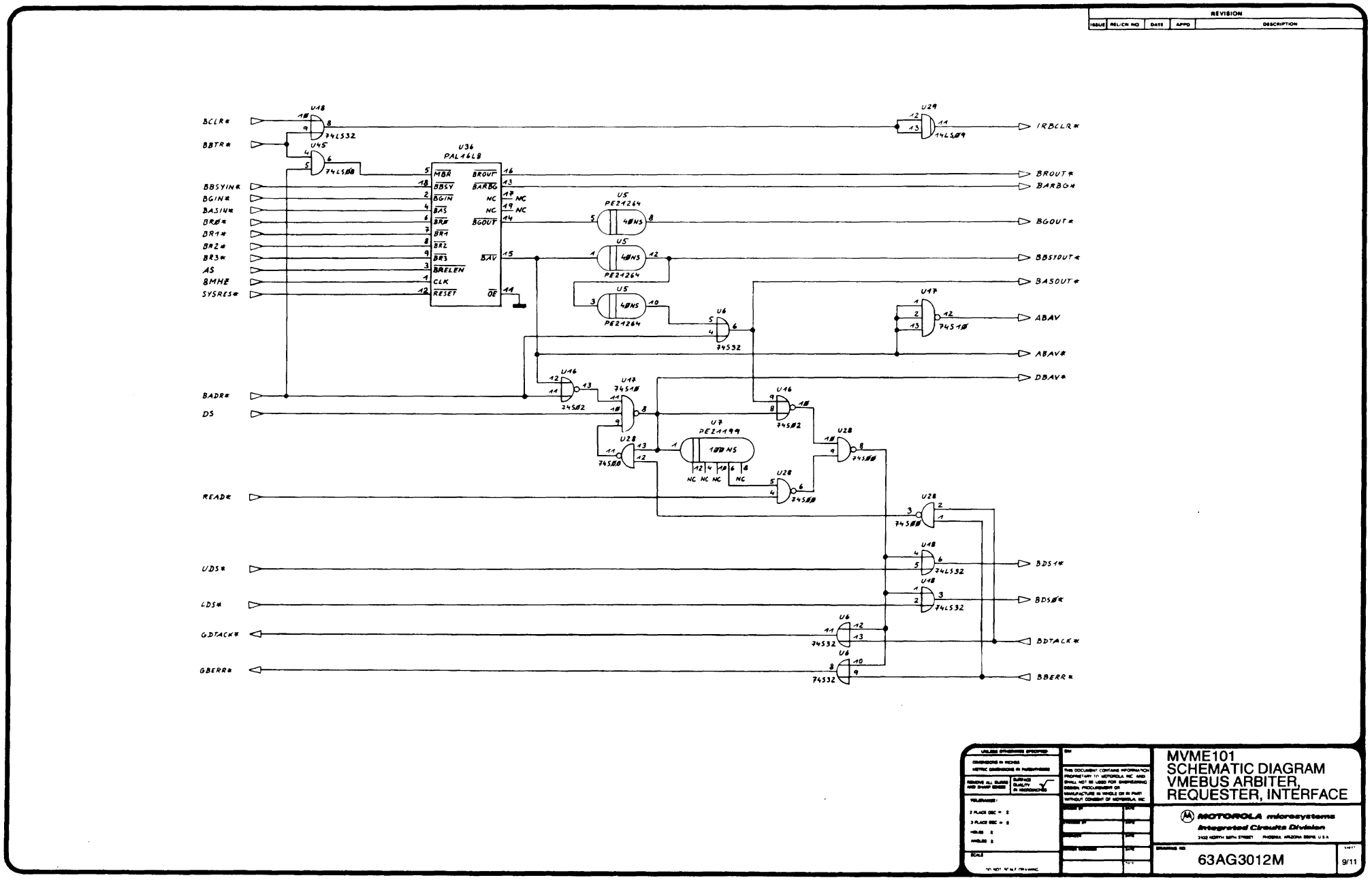


Figure 4.11: Schematic Diagram Sheet 10/11

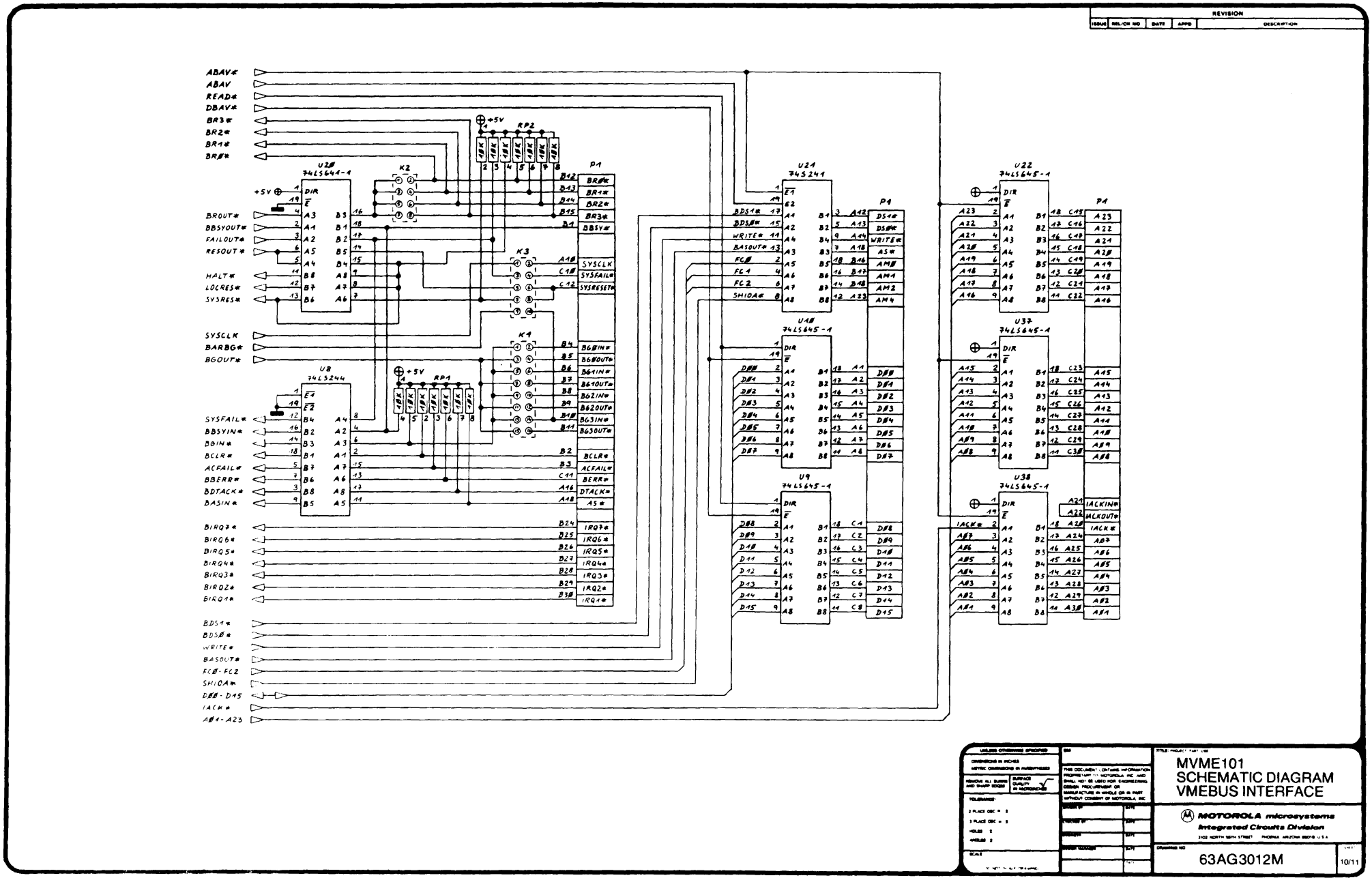
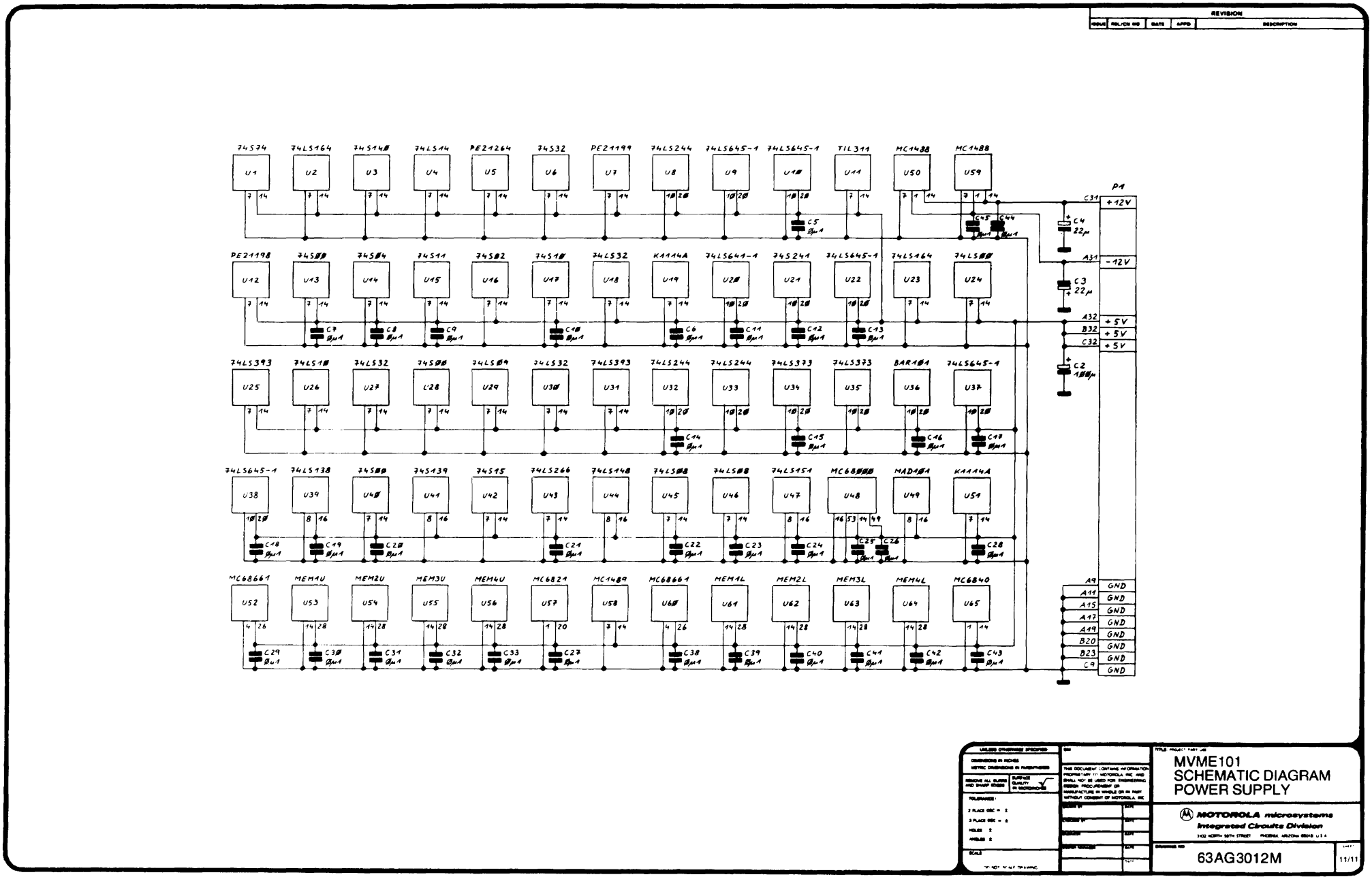


Figure 4.12: Schematic Diagram Sheet 11/11



A P P E N D I X A

MC68000 16-BIT MICROPROCESSING UNIT

MOTOROLA
SEMICONDUCTORS
3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

Advance Information

MC68000L4
(4 MHz)
MC68000L6
(6 MHz)
MC68000L8
(8 MHz)
MC68000L10
(10 MHz)

HMOS
(HIGH-DENSITY, N-CHANNEL, SILICON-GATE DEPLETION LOAD)

16-BIT MICROPROCESSOR

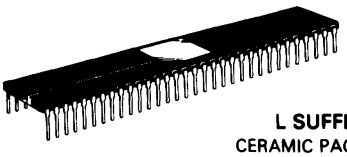
16-BIT MICROPROCESSING UNIT

Advances in semiconductor technology have provided the capability to place on a single silicon chip a microprocessor at least an order of magnitude higher in performance and circuit complexity than has been previously available. The MC68000 is the first of a family of such VLSI microprocessors from Motorola. It combines state-of-the-art technology and advanced circuit design techniques with computer sciences to achieve an architecturally advanced 16-bit microprocessor.

The resources available to the MC68000 user consist of the following:

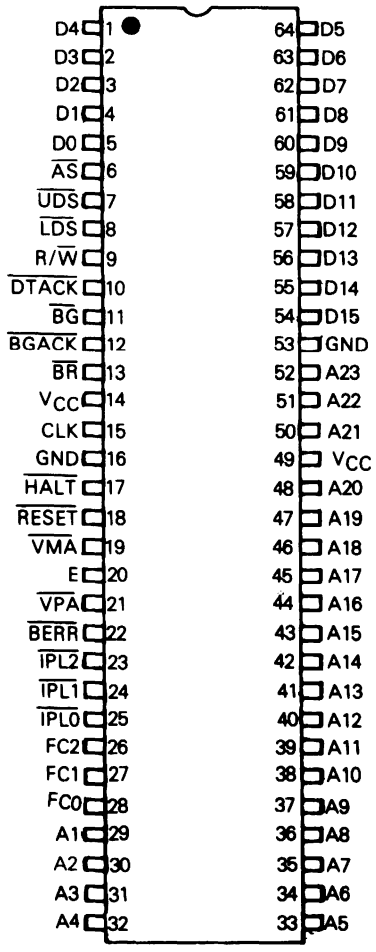
- 32-Bit Data and Address Registers
- 16 Megabyte Direct Addressing Range
- 56 Powerful Instruction Types
- Operations on Five Main Data Types
- Memory Mapped I/O
- 14 Addressing Modes

As shown in the programming model, the MC68000 offers seventeen 32-bit registers in addition to the 32-bit program counter and a 16-bit status register. The first eight registers (D0-D7) are used as data registers for byte (8-bit), word (16-bit), and long word (32-bit) data operations. The second set of seven registers (A0-A6) and the system stack pointer may be used as software stack pointers and base address registers. In addition, these registers may be used for word and long word address operations. All seventeen registers may be used as index registers.

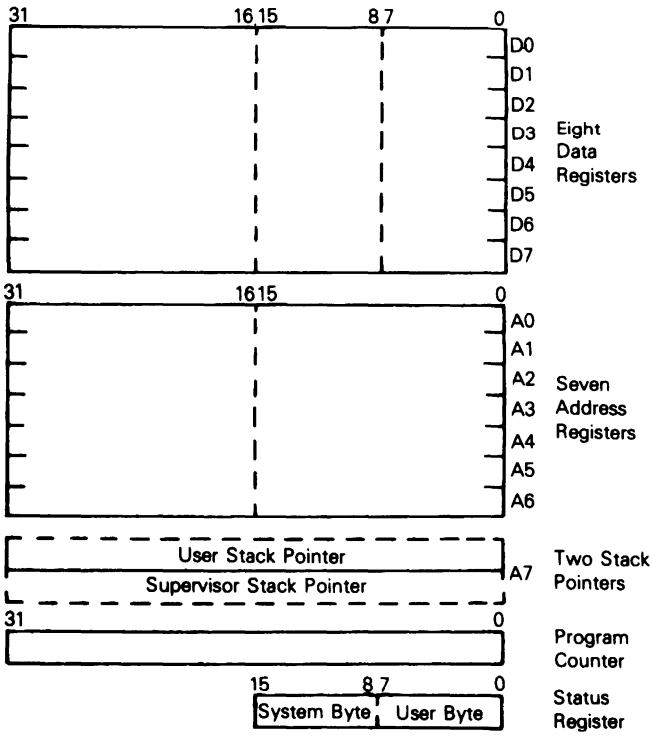


L SUFFIX
CERAMIC PACKAGE
CASE 746

PIN ASSIGNMENT



PROGRAMMING MODEL



This document contains information on a new product. Specifications and information herein are subject to change without notice.

ADI-814R2

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}	−0.3 to +7.0	V
Input Voltage	V_{in}	−0.3 to +7.0	V
Operating Temperature Range	T_A	0 to 70	°C
Storage Temperature	T_{stg}	−55 to 150	°C

THERMAL CHARACTERISTICS

Characteristic	Symbol	Value	Unit
Thermal Resistance Ceramic Package	θ_{JA}	30	°C/W

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum-rated voltages to this high-impedance circuit. Reliability of operation is enhanced if unused inputs are tied to an appropriate logic voltage level (e.g., either V_{SS} or V_{CC}.

POWER CONSIDERATIONS

The average chip-junction temperature, T_J, in °C can be obtained from:

$$T_J = T_A + (P_D \bullet \theta_{JA}) \quad (1)$$

Where:

T_A ≡ Ambient Temperature, °C

θ_{JA} ≡ Package Thermal Resistance, Junction-to-Ambient, °C/W

$P_D \equiv P_{INT} + P_{I/O}$

$P_{INT} \equiv I_{CC} \times V_{CC}$, Watts — Chip Internal Power

$P_{I/O}$ ≡ Power Dissipation on Input and Output Pins — User Determined

For most applications $P_{I/O} \ll P_{INT}$ and can be neglected.

An approximate relationship between P_D and T_J (if $P_{I/O}$ is neglected) is:

$$P_D = K \div (T_J + 273°C) \quad (2)$$

Solving equations 1 and 2 for K gives:

$$K = P_D \bullet (T_A + 273°C) + \theta_{JA} \bullet P_D^2 \quad (3)$$

Where K is a constant pertaining to the particular part. K can be determined from equation 3 by measuring P_D (at equilibrium) for a known T_A. Using this value of K the values of P_D and T_J can be obtained by solving equations (1) and (2) iteratively for any value of T_A.

DC ELECTRICAL CHARACTERISTICS (V_{CC} = 5.0 Vdc ±5%, V_{SS} = 0 Vdc; T_A = 0°C to 70°C, See Figures 1, 2, and 3)

Characteristic		Symbol	Min	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 @ 5.25 V	$\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$, CLK, $\overline{IPL0}$-$\overline{IPL2}$, VPA $\overline{HALT}$, $\overline{RESET}$	I_{in}	− −	2.5 20	μA
Three-State (Off State) Input Current @ 2.4 V/0.4 V	$\overline{AS}$, A1-A23, D0-D15 FC0-FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$	I_{TSI}	−	20	μA
Output High Voltage (I_{OH} = −400 μA)	E* $\overline{AS}$, A1-A23, BG, D0-D15 FC0-FC2, $\overline{LDS}$, R/$\overline{W}$, $\overline{UDS}$, $\overline{VMA}$	V_{OH}	V_{CC} − 0.75 2.4	− −	V
Output Low Voltage (I_{OL} = 1.6 mA) (I_{OL} = 3.2 mA) (I_{OL} = 35.0 mA) (I_{OL} = 5.3 mA)	 $\overline{HALT}$ A1-A23, $\overline{BG}$, FC0-FC2 $\overline{RESET}$ E, $\overline{AS}$, D0-D15, $\overline{LDS}$, R/$\overline{W}$ $\overline{UDS}$, $\overline{VMA}$	V_{OL}	 − − − −	 0.5 0.5 0.5 0.5	V
Power Dissipation (Clock Frequency = 8 MHz)		P_D	−	1.5	W
Capacitance (V_{in} = 0 V, T_A = 25°C; Frequency = 1 MHz)		C_{in}	−	10.0	pF

* With external pullup resistor of 470 Ω

FIGURE 1 — RESET TEST LOAD

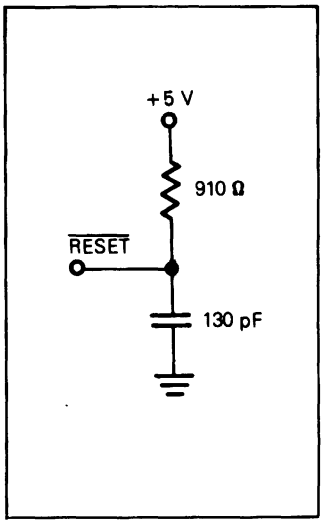


FIGURE 2 — HALT TEST LOAD

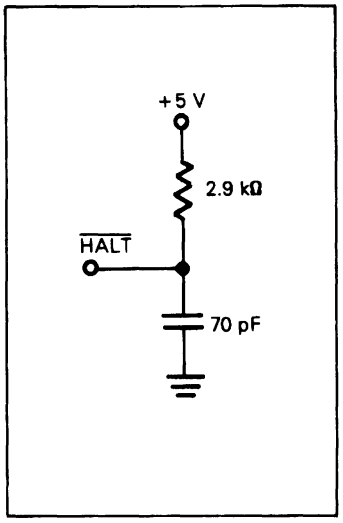
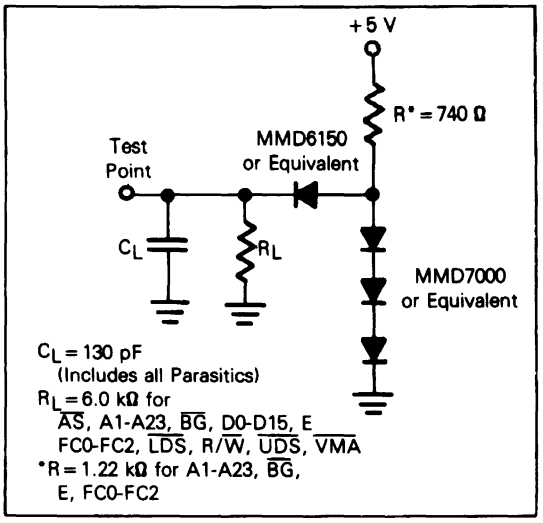


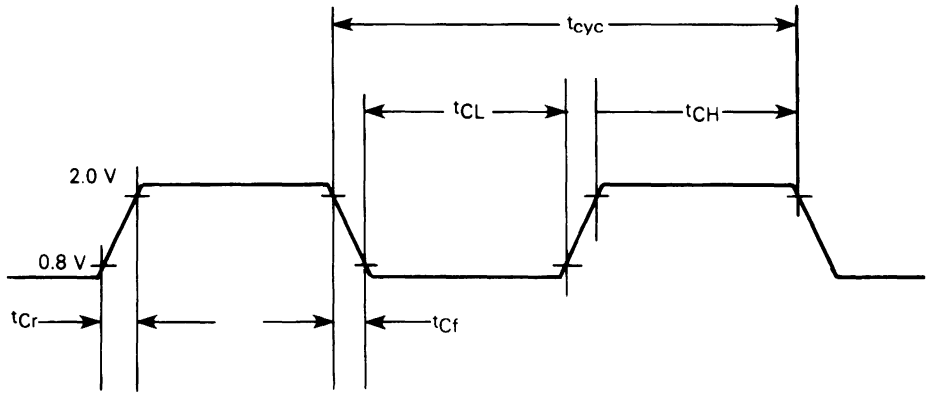
FIGURE 3 — TEST LOADS



CLOCK TIMING (See Figure 4)

Characteristic	Symbol	4 MHz MC68000L4		6 MHz MC68000L6		8 MHz MC68000L8		10 MHz MC68000L10		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
Frequency of Operation	F	2.0	4.0	2.0	6.0	2.0	8.0	2.0	10.0	MHz
Cycle Time	t_{cyc}	250	500	167	500	125	500	100	500	ns
Clock Pulse Width	t_{CL} t_{CH}	115 115	250 250	75 75	250 250	55 55	250 250	45 45	250 250	ns
Rise and Fall Times	t_{Cr} t_{Cf}	– –	10 10	– –	10 10	–	10 10	– –	10 10	ns

FIGURE 4 — INPUT CLOCK WAVEFORM



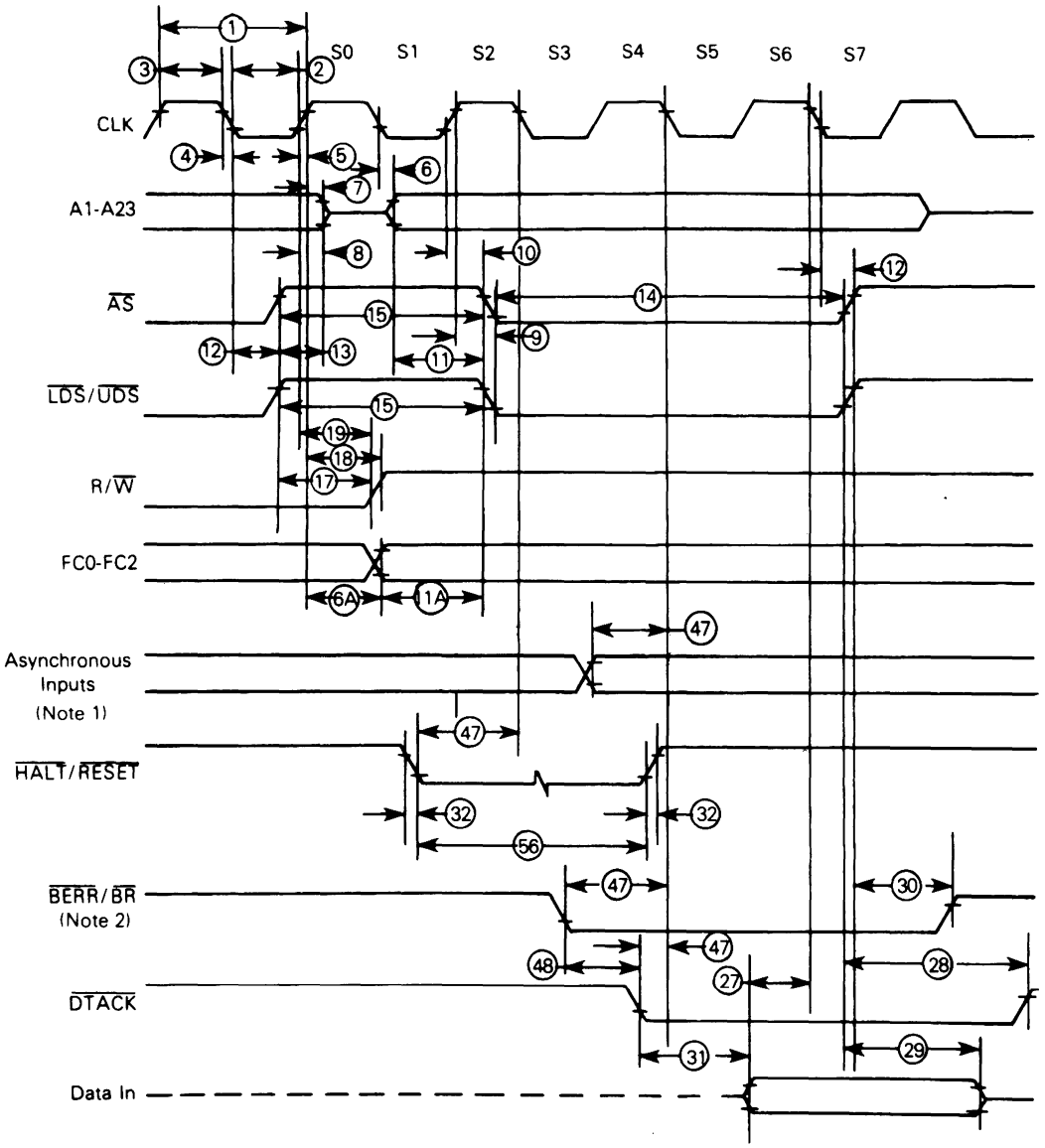
AC ELECTRICAL SPECIFICATIONS (V_{CC} = 5.0 Vdc ±5%, V_{SS} = 0 Vdc; T_A = 0°C to 70°C, See Figures 5 and 6)

Number	Characteristic	Symbol	4 MHz MC68000L4		6 MHz MC68000L6		8 MHz MC68000L8		10 MHz MC68000L10		Unit
			Min	Max	Min	Max	Min	Max	Min	Max	
1	Clock Period	t_{cyc}	250	500	167	500	125	500	100	500	ns
2	Clock Width Low	t_{CL}	115	250	75	250	55	250	45	250	ns
3	Clock Width High	t_{CH}	115	250	75	250	55	250	45	250	ns
4	Clock Fall Time	t_{Cf}	–	10	–	10	–	10	–	10	ns
5	Clock Rise Time	t_{Cr}	–	10	–	10	–	10	–	10	ns
6	Clock Low to Address	t_{CLAV}	–	90	–	80	–	70	–	55	ns
6A	Clock High to FC Valid	t_{CHFCV}	–	90	–	80	–	70	–	60	ns
7	Clock High to Address Data High Impedance (Maximum)	t_{CHAZx}	–	120	–	100	–	80	–	70	ns
8	Clock High to Address/FC Invalid (Minimum)	t_{CHAZn}	0	–	0	–	0	–	0	–	ns
9[1]	Clock High to $\overline{AS}$, $\overline{DS}$ Low (Maximum)	t_{CHSLx}	–	80	–	70	–	60	–	55	ns
10	Clock High to $\overline{AS}$, $\overline{DS}$ Low (Minimum)	t_{CHSLn}	0	–	0	–	0	–	0	–	ns
11[2]	Address to $\overline{AS}$, $\overline{DS}$ (Read) Low/$\overline{AS}$ Write	t_{AVSL}	55	–	35	–	30	–	20	–	ns
11A[2]	FC Valid to $\overline{AS}$, $\overline{DS}$, (Read) Low/$\overline{AS}$ Write	t_{FCVSL}	80	–	70	–	60	–	50	–	ns
12[1]	Clock Low to $\overline{AS}$, $\overline{DS}$ High	t_{CLSH}	–	90	–	80	–	70	–	55	ns
13[2]	$\overline{AS}$, $\overline{DS}$ High to Address/FC Invalid	t_{SHAZ}	60	–	40	–	30	–	20	–	ns
14[2, 5]	$\overline{AS}$, $\overline{DS}$ Width Low (Read)/$\overline{AS}$ Write	t_{SL}	535	–	337	–	240	–	195	–	ns
14A[2]	$\overline{DS}$ Width Low (Write)	–	285	–	170	–	115	–	95	–	ns
15[2]	$\overline{AS}$, $\overline{DS}$ Width High	t_{SH}	285	–	180	–	150	–	105	–	ns
16	Clock High to $\overline{AS}$, $\overline{DS}$ High Impedance	t_{CHSZ}	–	120	–	100	–	80	–	70	ns
17[2]	$\overline{AS}$, $\overline{DS}$ High to R/$\overline{W}$ High	t_{SHRH}	60	–	50	–	40	–	20	–	ns
18[1]	Clock High to R/$\overline{W}$ High (Maximum)	t_{CHRHx}	–	90	–	80	–	70	–	60	ns
19	Clock High to R/$\overline{W}$ High (Minimum)	t_{CHRHn}	0	–	0	–	0	–	0	–	ns
20[1]	Clock High to R/$\overline{W}$ Low	t_{CHRL}	–	90	–	80	–	70	–	60	ns
21[2]	Address Valid to R/$\overline{W}$ Low	t_{AVRL}	45	–	25	–	20	–	0	–	ns
21A[2]	FC Valid to R/$\overline{W}$ Low	t_{FCVRL}	80	–	70	–	60	–	50	–	ns
22[2]	R/$\overline{W}$ Low to $\overline{DS}$ Low (Write)	t_{RLSL}	200	–	140	–	80	–	50	–	ns
23	Clock Low to Data Out Valid	t_{CLDO}	–	90	–	80	–	70	–	55	ns
25[2]	$\overline{DS}$ High to Data Out Invalid	t_{SHDO}	60	–	40	–	30	–	20	–	ns
26[2]	Data Out Valid to $\overline{DS}$ Low (Write)	t_{DOSL}	55	–	35	–	30	–	20	–	ns
27[6]	Data In to Clock Low (Setup Time)	t_{DICL}	30	–	25	–	15	–	15	–	ns
28[2]	$\overline{AS}$, $\overline{DS}$ High to $\overline{DTACK}$ High	t_{SHDAH}	0	240	0	160	0	120	0	90	ns
29	$\overline{DS}$ High to Data Invalid (Hold Time)	t_{SHDI}	0	–	0	–	0	–	0	–	ns
30	$\overline{AS}$, $\overline{DS}$ High to $\overline{BERR}$ High	t_{SHBEH}	0	–	0	–	0	–	0	–	ns
31[2, 6]	$\overline{DTACK}$ Low to Data In (Setup Time)	t_{DALDI}	–	180	–	120	–	90	–	65	ns
32	$\overline{HALT}$ and $\overline{RESET}$ Input Transition Time	t_{RHrf}	0	200	0	200	0	200	0	200	ns
33	Clock High to $\overline{BG}$ Low	t_{CHGL}	–	90	–	80	–	70	–	60	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}	–	90	–	80	–	70	–	60	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk. Per.
36	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk. Per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	t_{GALGH}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk Per.
38	$\overline{BG}$ Low to Bus High Impedance (With $\overline{AS}$ High)	t_{GLZ}	–	120	–	100	–	80	–	70	ns
39	$\overline{BG}$ Width High	t_{GH}	1.5	–	1.5	–	1.5	–	1.5	–	Clk. Per.
46	$\overline{BGACK}$ Width	t_{BGL}	1.5	–	1.5	–	1.5	–	1.5	–	Clk. Per.
47[6]	Asynchronous Input Setup Time	t_{ASI}	30	–	25	–	20	–	20	–	ns
48	$\overline{BERR}$ Low to $\overline{DTACK}$ Low (Note 3)	t_{BELDAL}	50	–	50	–	50	–	50	–	ns
53	Data Hold from Clock High	t_{CHDO}	0	–	0	–	0	–	0	–	ns
55	R/$\overline{W}$ to Data Bus Impedance Change	t_{RLDO}	55	–	35	–	30	–	20	–	ns
56	Halt/$\overline{RESET}$ Pulse Width (Note 4)	t_{HRPW}	10	–	10	–	10	–	10	–	Clk. Per.

NOTES:

1. For a loading capacitance of less than or equal to 500 picofarads, subtract 5 nanoseconds from the values given in these columns.
2. Actual value depends on clock period.
3. If #47 is satisfied for both $\overline{DTACK}$ and $\overline{BERR}$, #48 may be 0 ns.
4. After V_{CC} has been applied for 100 ms.
5. For T6E, BF4, and R9M mask sets #14 and #14A are one clock period less than the given number.
6. If the asynchronous setup time (#47) requirements are satisfied, the $\overline{DTACK}$ low-to-data setup time (#31) requirement can be ignored. The data must only satisfy the data-in to clock-low setup time (#27) for the following cycle.

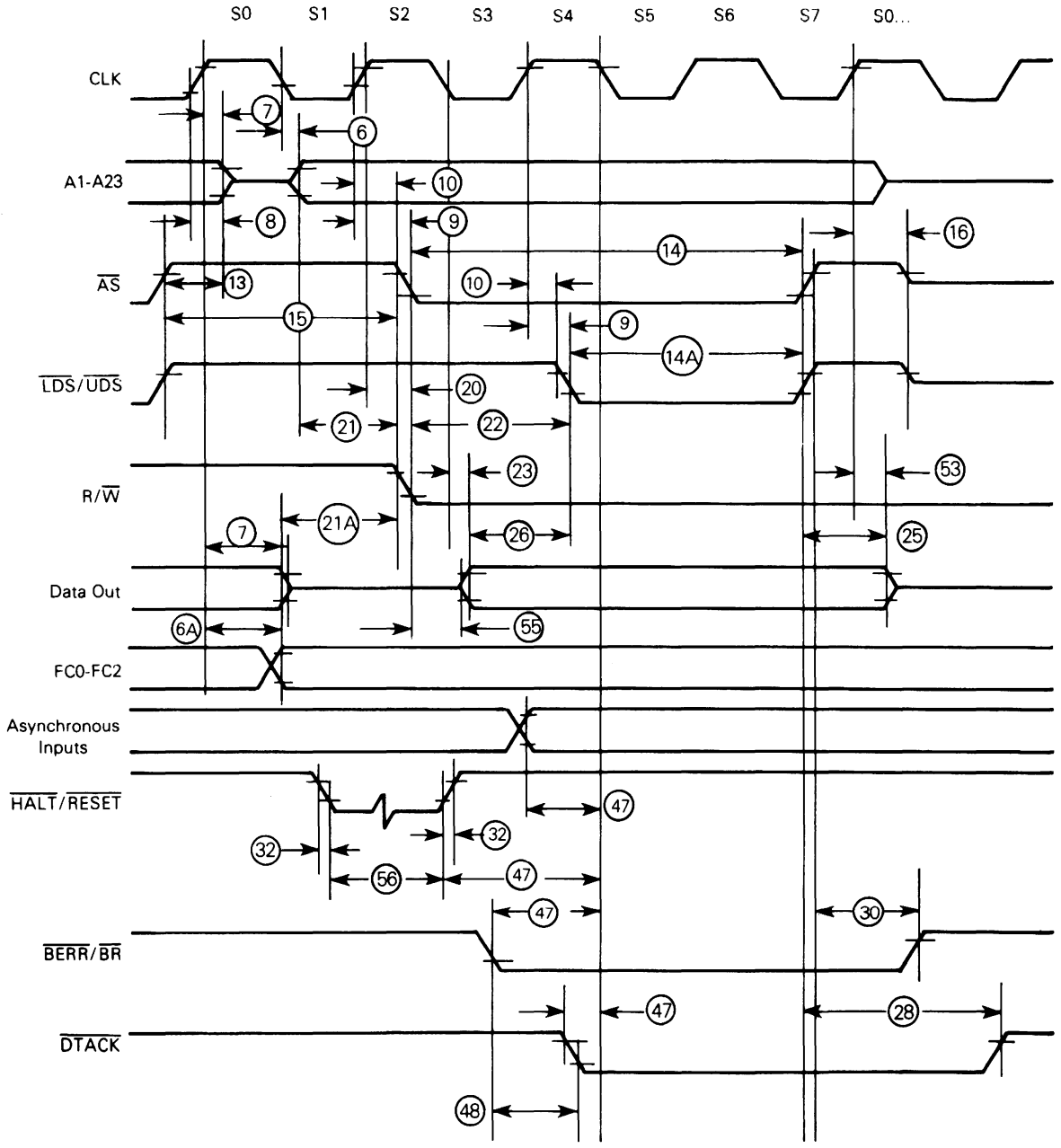
FIGURE 5 — READ CYCLE TIMING



NOTES:

1. Setup time for the asynchronous inputs BGACK, IPL0-IPL2, and VPA guarantees their recognition at the next falling edge of the clock.
2. BR need fall at this time only in order to insure being recognized at the end of this bus cycle.
3. Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted.

FIGURE 6 — WRITE CYCLE TIMING



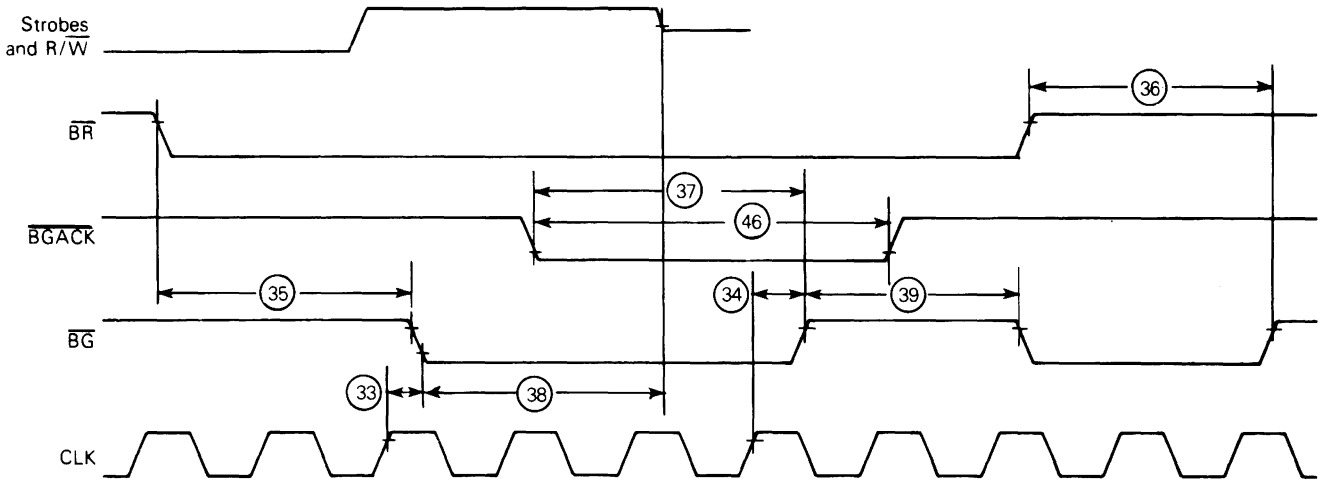
NOTE: Timing measurements are referenced to and from a low voltage of 0.8 volts and a high voltage of 2.0 volts, unless otherwise noted.

AC ELECTRICAL SPECIFICATIONS — BUS ARBITRATION (V_{CC} = 5.0 Vdc ±5%, V_{SS} = 0 Vdc; T_A = 0°C to 70°C, See Figure 7)

Number	Characteristic	Symbol	4 MHz MC68000L4		6 MHz MC68000L6		8 MHz MC68000L8		10 MHz MC68000L10		Unit
			Min	Max	Min	Max	Min	Max	Min	Max	
33	Clock High to $\overline{BG}$ Low	t_{CHGL}	–	90	–	80	–	70	–	60	ns
34	Clock High to $\overline{BG}$ High	t_{CHGH}	–	90	–	80	–	70	–	60	ns
35	$\overline{BR}$ Low to $\overline{BG}$ Low	t_{BRLGL}	1.5	3.5	1.5	3.5	1.5	3.5	1.5	3.5	Clk. Per.
36	$\overline{BR}$ High to $\overline{BG}$ High	t_{BRHGH}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk. Per.
37	$\overline{BGACK}$ Low to $\overline{BG}$ High	t_{GALGH}	1.5	3.0	1.5	3.0	1.5	3.0	1.5	3.0	Clk. Per.
38	$\overline{BG}$ Low to Bus High Impedance (with $\overline{AS}$ High)	t_{GLZ}	–	120	–	100	–	80	–	70	ns
39	$\overline{BG}$ Width High	t_{GH}	1.5	–	1.5	–	1.5	–	1.5	–	Clk. Per.
46	$\overline{BGACK}$ Width	t_{BGL}	1.5	–	1.5	–	1.5	–	1.5	–	Clk. Per.

FIGURE 7 — AC ELECTRICAL WAVEFORMS — BUS ARBITRATION

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.



NOTES:

1. Setup time for the asynchronous inputs $\overline{BERR}$, $\overline{BGACK}$, $\overline{BR}$, $\overline{DTACK}$, $\overline{IPL0}$-$\overline{IPL2}$, and $\overline{VPA}$ guarantees their recognition at the next falling edge of the clock.
2. Waveform measurements for all inputs and outputs are specified at: logic high = 2.0 volts, logic low = 0.8 volts.

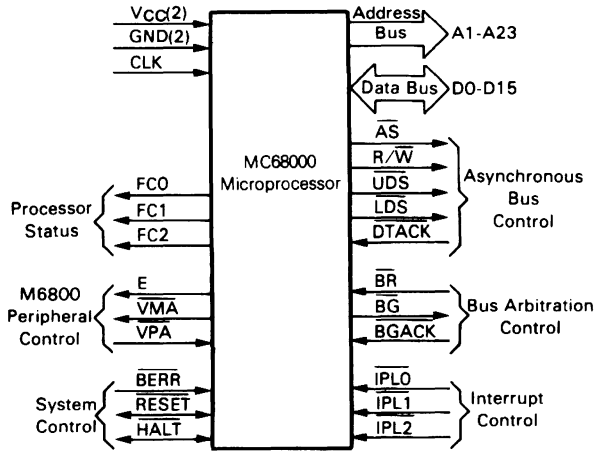
SIGNAL DESCRIPTION

The following paragraphs contain a brief description of the input and output signals. A discussion of bus operation during the various machine cycles and operations is also given.

SIGNAL DESCRIPTION

The input and output signals can be functionally organized into the groups shown in Figure 8. The following paragraphs provide a brief description of the signals and also a reference (if applicable) to other paragraphs that contain more detail about the function being performed.

FIGURE 8 — INPUT AND OUTPUT SIGNALS



ADDRESS BUS (A1 THROUGH A23). This 23-bit, unidirectional, three-state bus is capable of addressing 8 megawords of data. It provides the address for bus operation during all cycles except interrupt cycles. During interrupt cycles, address lines A1, A2, and A3 provide information about what level interrupt is being serviced while address lines A4 through A23 are all set to a logic high.

DATA BUS (D0 THROUGH D15). This 16-bit, bidirectional, three-state bus is the general purpose data path. It can transfer and accept data in either word or byte length. During an interrupt acknowledge cycle, an external device supplies the vector number on data lines D0-D7.

ASYNCHRONOUS BUS CONTROL. Asynchronous data transfers are handled using the following control signals: address strobe, read/write, upper and lower data strobes, and data transfer acknowledge. These signals are explained in the following paragraphs.

Address Strobe ($\overline{AS}$). This signal indicates that there is a valid address on the address bus.

Read/Write (R/$\overline{W}$). This signal defines the data bus transfer as a read or write cycle. The R/$\overline{W}$ signal also works in conjunction with the upper and lower data strobes as explained in the following paragraph.

Upper And Lower Data Strobes ($\overline{UDS}$, $\overline{LDS}$). These signals control the data on the data bus, as shown in Table 1. When the R/$\overline{W}$ line is high, the processor will read from the data bus as indicated. When the R/$\overline{W}$ line is low, the processor will write to the data bus as shown.

TABLE 1 — DATA STROBE CONTROL OF DATA BUS

$\overline{UDS}$	$\overline{LDS}$	R/$\overline{W}$	D8-D15	D0-D7
High	High	–	No valid data	No valid data
Low	Low	High	Valid data bits 8-15	Valid data bits 0-7
High	Low	High	No valid data	Valid data bits 0-7
Low	High	High	Valid data bits 8-15	No valid data
Low	Low	Low	Valid data bits 8-15	Valid data bits 0-7
High	Low	Low	Valid data bits 0-7*	Valid data bits 0-7
Low	High	Low	Valid data bits 8-15	Valid data bits 8-15*

*These conditions are a result of current implementation and may not appear on future devices.

Data Transfer Acknowledge ($\overline{DTACK}$). This input indicates that the data transfer is completed. When the processor recognizes $\overline{DTACK}$ during a read cycle, data is latched and the bus cycle terminated. When $\overline{DTACK}$ is recognized during a write cycle, the bus cycle is terminated. An active transition of data transfer acknowledge, $\overline{DTACK}$, indicates the termination of a data transfer on the bus.

If the system must run at a maximum rate determined by RAM access times, the relationship between the times at which $\overline{DTACK}$ and DATA are sampled are important.

All control and data lines are sampled during the MC68000's clock high time. The clock is internally buffered, which results in some slight differences in the sampling and recognition of various signals. MC68000 mask sets prior to CC1 (R9M and T6E), allowed $\overline{DTACK}$ to be recognized as early as S2 (bus state 2), and all devices allow $\overline{BERR}$ or $\overline{DTACK}$ to be recognized in S4, S6, etc., which terminates the cycle. The $\overline{DTACK}$ signal, like other control signals, is internally synchronized to allow for valid operation in an asynchronous system. If the required setup time (#47) is met during S4, $\overline{DTACK}$ will be recognized during S5 and S6, and data will be captured during S6. The data must meet the required setup time (#27).

If an asynchronous control signal does not meet the required setup time, it is possible that it may not be recognized during that cycle. Because of this, asynchronous systems must not allow $\overline{DTACK}$ to precede data by more than parameter #31.

Asserting $\overline{DTACK}$ (or $\overline{BERR}$) on the rising edge of a clock (such as S4) after the assertion of address strobe will allow a MC68000 system to run at its maximum bus rate. If setup times #27 and #47 are guaranteed, #31 may be ignored.

BUS ARBITRATION CONTROL. These three signals form a bus arbitration circuit to determine which device will be the bus master device.

Bus Request ($\overline{BR}$). This input is wire ORed with all other devices that could be bus masters. This input indicates to the processor that some other device desires to become the bus master.

Bus Grant ($\overline{BG}$). This output indicates to all other potential bus master devices that the processor will release bus control at the end of the current bus cycle.

Bus Grant Acknowledge ($\overline{BGACK}$). This input indicates that some other device has become the bus master. This signal cannot be asserted until the following four conditions are met:

1. a Bus Grant has been received
2. Address Strobe is inactive which indicates that the microprocessor is not using the bus
3. Data Transfer Acknowledge is inactive which indicates that neither memory nor peripherals are using the bus
4. Bus Grant Acknowledge is inactive which indicates that no other device is still claiming bus mastership.

INTERRUPT CONTROL ($\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$). These input pins indicate the encoded priority level of the device requesting an interrupt. Level seven is the highest priority while level zero indicates that no interrupts are requested. The least significant bit is given in $\overline{IPL0}$ and the most significant bit is contained in $\overline{IPL2}$.

SYSTEM CONTROL. The system control inputs are used to either reset or halt the processor and to indicate to the processor that bus errors have occurred. The three system control inputs are explained in the following paragraphs.

Bus Error ($\overline{BERR}$). This input informs the processor that there is a problem with the cycle currently being executed. Problems may be a result of:

1. nonresponding devices
2. interrupt vector number acquisition failure
3. illegal access request as determined by a memory management unit
4. other application dependent errors.

The bus error signal interacts with the halt signal to determine if exception processing should be performed or the current bus cycle should be retried.

Refer to **BUS ERROR AND HALT OPERATION** paragraph for additional information about the interaction of the bus error and halt signals.

Reset ($\overline{RESET}$). This bidirectional signal line acts to reset (initiate a system initialization sequence) the processor in response to an external reset signal. An internally generated reset (result of a RESET instruction) causes all external devices to be reset and the internal state of the processor is not affected. A total system reset (processor and external devices) is the result of external HALT and RESET signals applied at the same time. Refer to **RESET OPERATION** paragraph for additional information about reset operation.

Halt ($\overline{HALT}$). When this bidirectional line is driven by an external device, it will cause the processor to stop at the completion of the current bus cycle. When the processor has been halted using this input, all control signals are inactive and all three-state lines are put in their high-impedance state. Refer to **BUS ERROR AND HALT OPERATION** paragraph for additional information about the interaction between the halt and bus error signals.

When the processor has stopped executing instructions, such as in a double bus fault condition, the halt line is driven by the processor to indicate to external devices that the processor has stopped.

M6800 PERIPHERAL CONTROL. These control signals are used to allow the interfacing of synchronous M6800 peripheral devices with the asynchronous MC68000. These signals are explained in the following paragraphs.

Enable (E). This signal is the standard enable signal common to all M6800 type peripheral devices. The period for this output is ten MC68000 clock periods (six clocks low; four clocks high).

Valid Peripheral Address ($\overline{VPA}$). This input indicates that the device or region addressed is a M6800 family device and that data transfer should be synchronized with the enable (E) signal. This input also indicates that the processor should use automatic vectoring for an interrupt. Refer to **INTERFACE WITH M6800 PERIPHERALS**.

Valid Memory Address ($\overline{VMA}$). This output is used to indicate to M6800 peripheral devices that there is a valid address on the address bus and the processor is synchronized to enable. This signal only responds to a valid peripheral address ($\overline{VPA}$) input which indicates that the peripheral is a M6800 family device.

PROCESSOR STATUS (FC0, FC1, FC2). These function code outputs indicate the state (user or supervisor) and the cycle type currently being executed, as shown in Table 2. The information indicated by the function code outputs is valid whenever address strobe ($\overline{AS}$) is active.

TABLE 2 — FUNCTION CODE OUTPUTS

FC2	FC1	FC0	Cycle Type
Low	Low	Low	(Undefined, Reserved)
Low	Low	High	User Data
Low	High	Low	User Program
Low	High	High	(Undefined, Reserved)
High	Low	Low	(Undefined, Reserved)
High	Low	High	Supervisor Data
High	High	Low	Supervisor Program
High	High	High	Interrupt Acknowledge

CLOCK (CLK). The clock input is a TTL-compatible signal that is internally buffered for development of the internal clocks needed by the processor. The clock input shall be a constant frequency.

SIGNAL SUMMARY. Table 3 is a summary of all the signals discussed in the previous paragraphs.

TABLE 3 — SIGNAL SUMMARY

Signal Name	Mnemonic	Input/Output	Active State	Three State
Address Bus	A1-A23	output	high	yes
Data Bus	D0-D15	input/output	high	yes
Address Strobe	$\overline{AS}$	output	low	yes
Read/Write	R/$\overline{W}$	output	read-high write-low	yes
Upper and Lower Data Strobes	$\overline{UDS}$, $\overline{LDS}$	output	low	yes
Data Transfer Acknowledge	$\overline{DTACK}$	input	low	no
Bus Request	$\overline{BR}$	input	low	no
Bus Grant	$\overline{BG}$	output	low	no
Bus Grant Acknowledge	$\overline{BGACK}$	input	low	no
Interrupt Priority Level	$\overline{IPL0}$, $\overline{IPL1}$, $\overline{IPL2}$	input	low	no
Bus Error	$\overline{BERR}$	input	low	no
Reset	$\overline{RESET}$	input/output	low	no*
Halt	$\overline{HALT}$	input/output	low	no*
Enable	E	output	high	no
Valid Memory Address	$\overline{VMA}$	output	low	yes
Valid Peripheral Address	$\overline{VPA}$	input	low	no
Function Code Output	FC0, FC1, FC2	output	high	yes
Clock	CLK	input	high	no
Power Input	V_{CC}	input	—	—
Ground	GND	input	—	—

*open drain

REGISTER DESCRIPTION AND DATA ORGANIZATION

The following paragraphs describe the registers and data organization of the MC68000.

OPERAND SIZE

Operand sizes are defined as follows: a byte equals 8 bits, a word equals 16 bits, and a long word equals 32 bits. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation. All explicit instructions support byte, word or long word operands. Implicit instructions support some subset of all three sizes.

DATA ORGANIZATION IN REGISTERS

The eight data registers support data operands of 1, 8, 16, or 32 bits. The seven address registers together with the active stack pointer support address operands of 32 bits.

DATA REGISTERS. Each data register is 32 bits wide. Byte operands occupy the low order 8 bits, word operands the low order 16 bits, and long word operands the entire 32 bits. The least significant bit is addressed as bit zero; the most significant bit is addressed as bit 31.

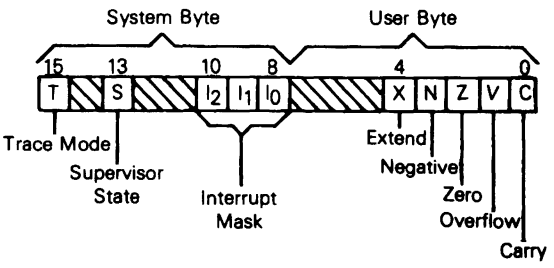
When a data register is used as either a source or destination operand, only the appropriate low-order portion is changed; the remaining high-order portion is neither used nor changed.

ADDRESS REGISTERS. Each address register and the stack pointer is 32 bits wide and holds a full 32 bit address. Address registers do not support byte sized operands. Therefore, when an address register is used as a source operand, either the low order word or the entire long word operand is used depending upon the operation size. When an address register is used as the destination operand, the entire register is affected regardless of the operation size. If the operation size is word, any other operands are sign extended to 32 bits before the operation is performed.

STATUS REGISTER

The status register contains the interrupt mask (eight levels available) as well as the condition codes; extend (X), negative (N), zero (Z), overflow (V), and carry (C). Additional status bits indicate that the processor is in a trace (T) mode and/or in a supervisor (S) state.

STATUS REGISTER



DATA ORGANIZATION IN MEMORY

Bytes are individually addressable with the high order byte having an even address the same as the word, as shown in Figure 9. The low order byte has an odd address that is one count higher than the word address. Instructions and multibyte data are accessed only on word (even byte) boundaries. If a long word datum is located at address n (n even), then the second word of that datum is located at address n+2.

The data types supported by the MC68000 are: bit data, integer data of 8, 16, or 32 bits, 32-bit addresses and binary coded decimal data. Each of these data types is put in memory, as shown in Figure 10.

BUS OPERATION

The following paragraphs explain control signal and bus operation during data transfer operations, bus arbitration, bus error and halt conditions, and reset operation.

DATA TRANSFER OPERATIONS. Transfer of data between devices involves the following leads:

- Address Bus A1 through A23
- Data Bus D0 through D15
- Control Signals

The address and data buses are separate parallel buses used to transfer data using an asynchronous bus structure. In all cycles, the bus master assumes responsibility for deskewing all signals it issues at both the start and end of a cycle. In addition, the bus master is responsible for deskewing the acknowledge and data signals from the slave device.

The following paragraphs explain the read, write, and read-modify-write cycles. The indivisible read-modify-write cycle is the method used by the MC68000 for interlocked multiprocessor communications.

NOTE

The terms **assertion** and **negation** will be used extensively. This is done to avoid confusion when dealing with a mixture of "active-low" and "active-high" signals. The term assert or assertion is used to indicate that a signal is active or true independent of whether that voltage is low or high. The term negate or negation is used to indicate that a signal is inactive or false.

Read Cycle. During a read cycle, the processor receives data from memory or a peripheral device. The processor reads bytes of data in all cases. If the instruction specifies a word (or double word) operation, the processor reads both bytes. When the instruction specifies byte operation, the processor uses an internal A0 bit to determine which byte to read and then issues the data strobe required for that byte. For byte operations, when the A0 bit equals zero, the upper data strobe is issued. When the A0 bit equals one, the lower data strobe is issued. When the data is received, the processor correctly positions it internally.

A word read cycle flow chart is given in Figure 11. A byte read cycle flow chart is given in Figure 12. Read cycle timing is given in Figure 13. Figure 14 details word and byte read cycle operations.

FIGURE 9 — WORD ORGANIZATION IN MEMORY

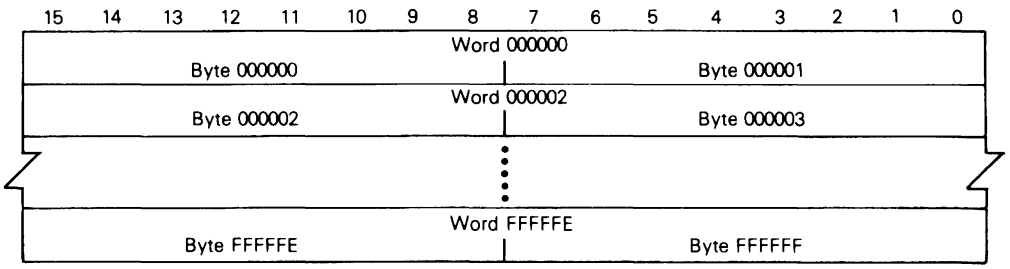
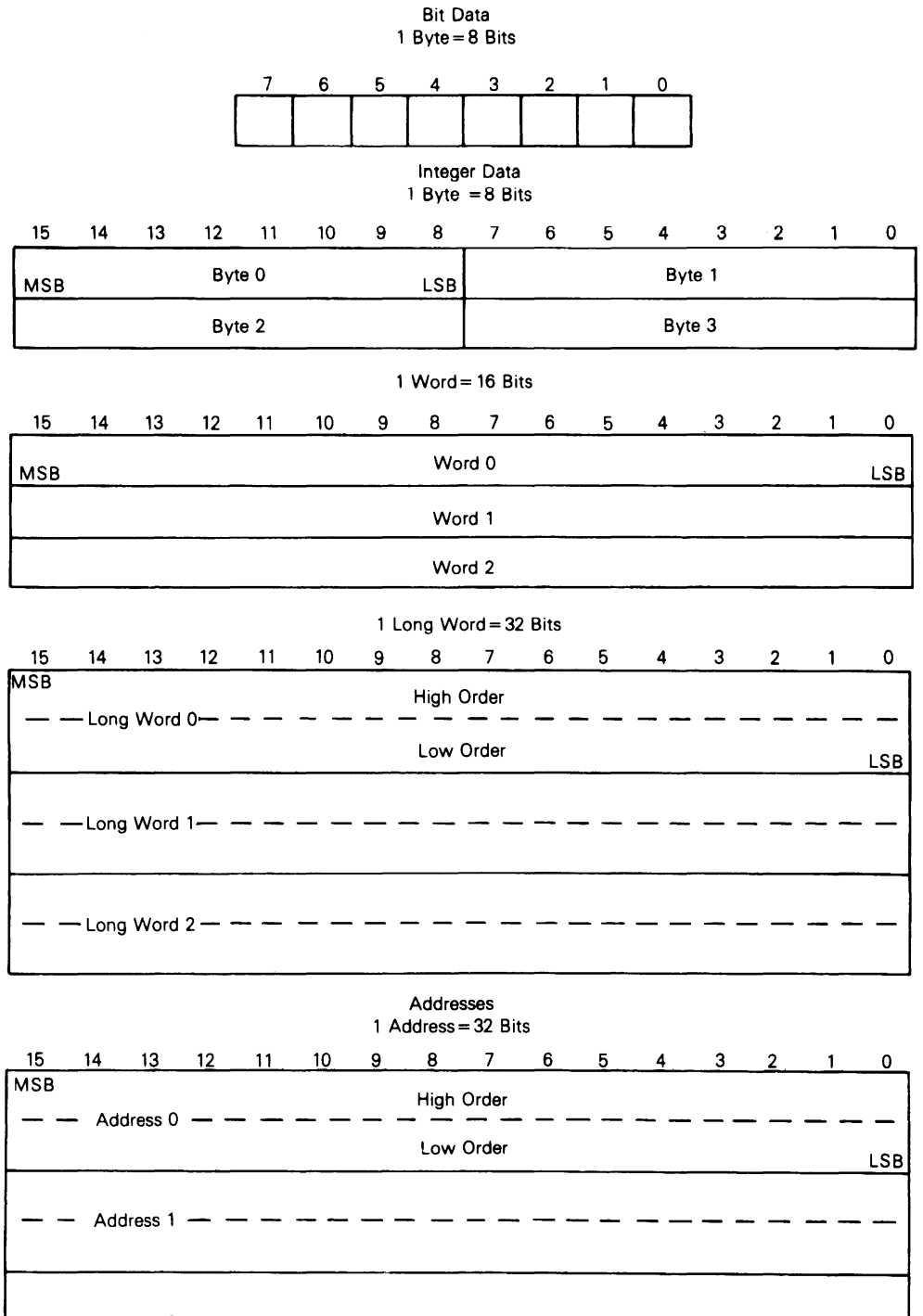
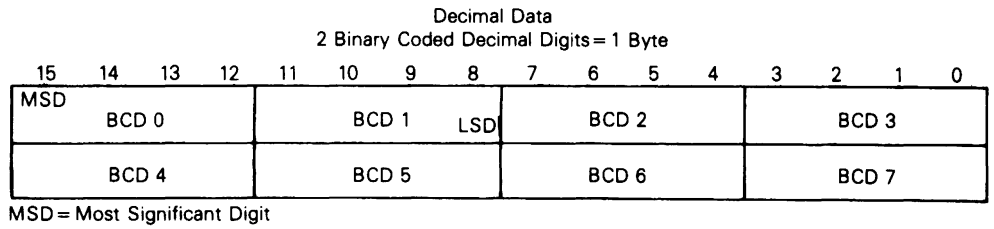


FIGURE 10 — DATA ORGANIZATION IN MEMORY



MSB = Most Significant Bit
LSB = Least Significant Bit



MSD = Most Significant Digit
LSD = Least Significant Digit

FIGURE 11 — WORD READ CYCLE FLOW CHART

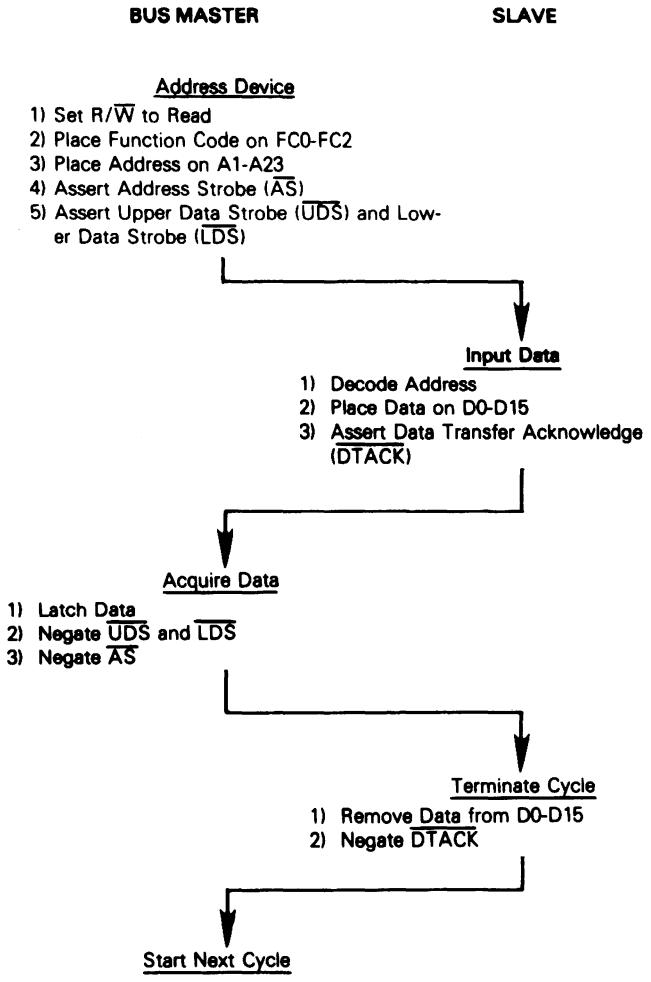


FIGURE 12 — BYTE READ CYCLE FLOW CHART

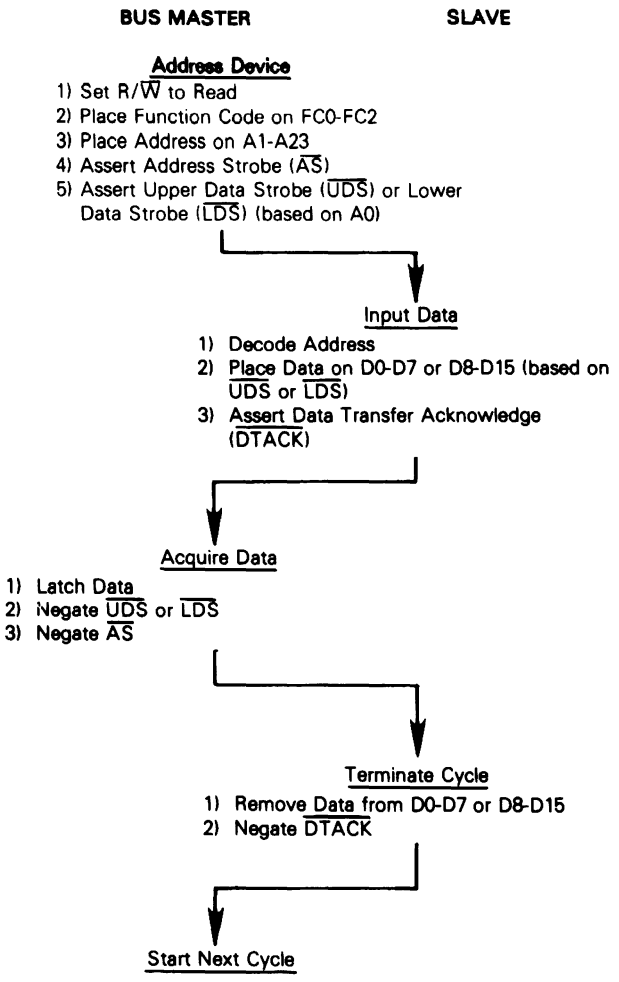


FIGURE 13 — READ AND WRITE CYCLE TIMING DIAGRAM

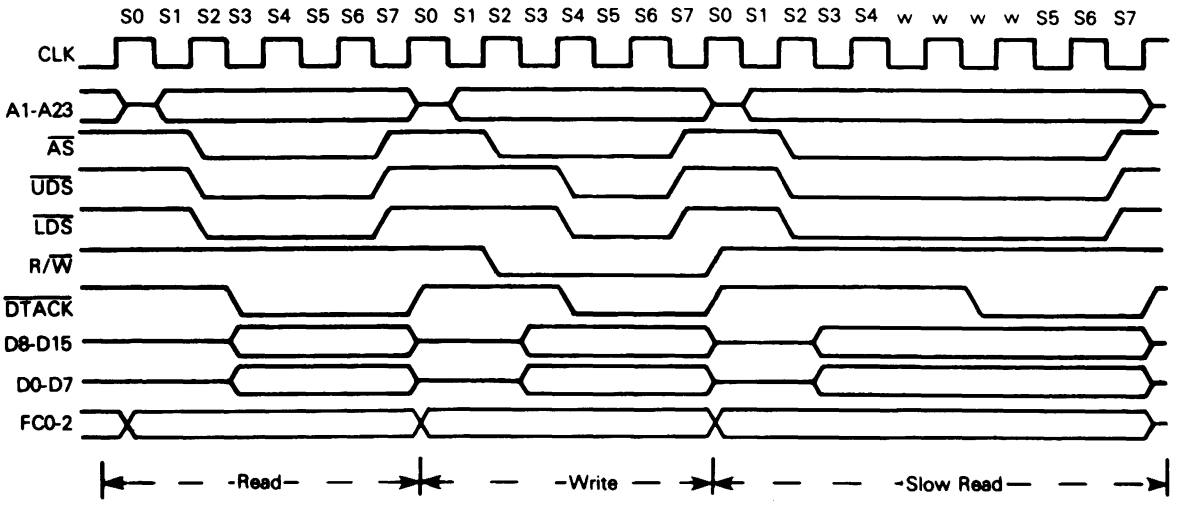
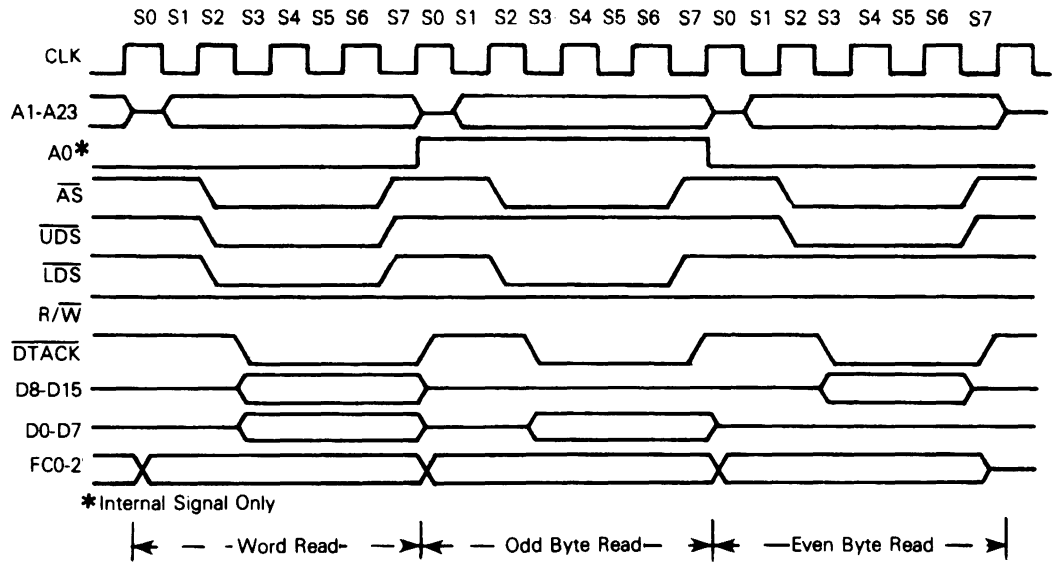


FIGURE 14 — WORD AND BYTE READ CYCLE TIMING DIAGRAM



Write Cycle. During a write cycle, the processor sends data to memory or a peripheral device. The processor writes bytes of data in all cases. If the instruction specifies a word operation, the processor writes both bytes. When the instruction specifies a byte operation, the processor uses an internal A0 bit to determine which byte to write and then issues the data strobe required for that byte. For byte operations, when the A0 bit equals zero, the upper data strobe is issued. When the A0 bit equals one, the lower data strobe is issued. A word write cycle flow chart is given in Figure 15. A byte write cycle flow chart is given in Figure 16. Write cycle timing is given in Figure 13. Figure 17 details word and byte write cycle operation.

Read-Modify-Write Cycle. The read-modify-write cycle performs a read, modifies the data in the arithmetic-logic unit, and writes the data back to the same address. In the MC68000 this cycle is indivisible in that the address strobe is asserted throughout the entire cycle. The test and set (TAS) instruction uses this cycle to provide meaningful communication between processors in a multiple processor environment. This instruction is the only instruction that uses the read-modify-write cycles and since the test and set instruction only operates on bytes, all read-modify-write cycles are byte operations. A read-modify-write cycle flow chart is given in Figure 18 and a timing diagram is given in Figure 19.

BUS ARBITRATION. Bus arbitration is a technique used by master-type devices to request, be granted, and acknowledge bus mastership. In its simplest form, it consists of:

1. Asserting a bus mastership request.
2. Receiving a grant that the bus is available at the end of the current cycle.
3. Acknowledging that mastership has been assumed.

Figure 20 is a flow chart showing the detail involved in a request from a single device. Figure 21 is a timing diagram for the same operations. This technique allows processing of bus requests during data transfer cycles.

The timing diagram shows that the bus request is negated at the time that an acknowledge is asserted. This type of operation would be true for a system consisting of the processor and one device capable of bus mastership. In systems having a number of devices capable of bus mastership, the bus request line from each device is wire ORed to the processor. In this system, it is easy to see that there could be more than one bus request being made. The timing diagram shows that the bus grant signal is negated a few clock cycles after the transition of the acknowledge ($\overline{BGACK}$) signal.

However, if the bus requests are still pending, the processor will assert another bus grant within a few clock cycles after it was negated. This additional assertion of bus grant allows external arbitration circuitry to select the next bus master before the current bus master has completed its requirements. The following paragraphs provide additional information about the three steps in the arbitration process.

FIGURE 15 — WORD WRITE CYCLE FLOW CHART

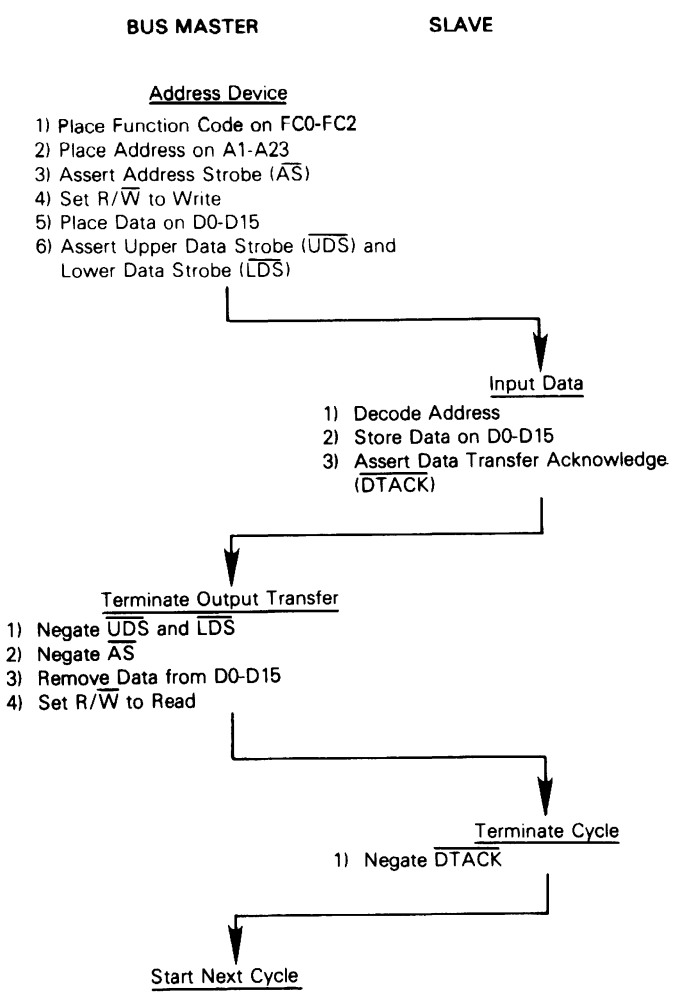


FIGURE 16 — BYTE WRITE CYCLE FLOW CHART

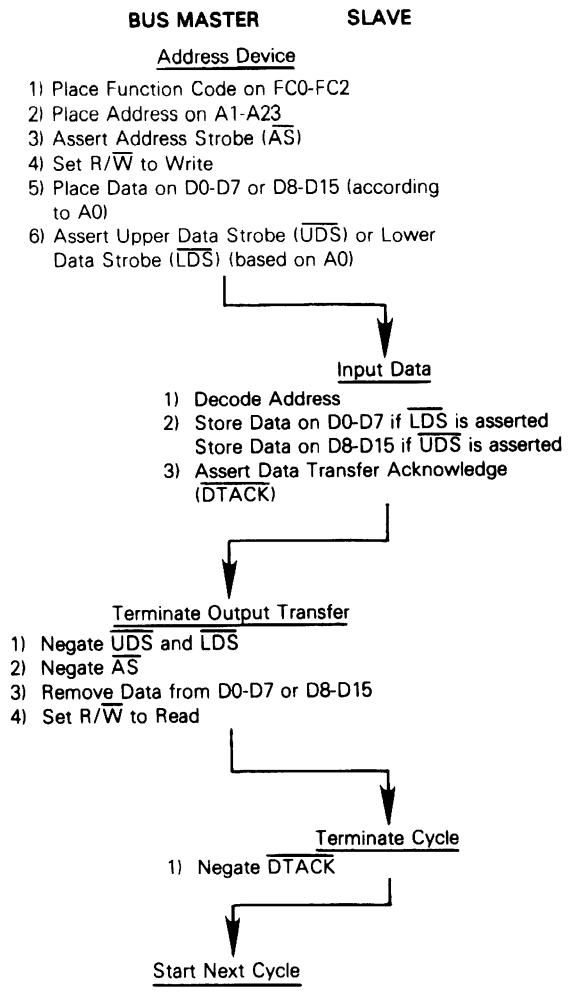


FIGURE 17 — WORD AND BYTE WRITE CYCLE TIMING DIAGRAM

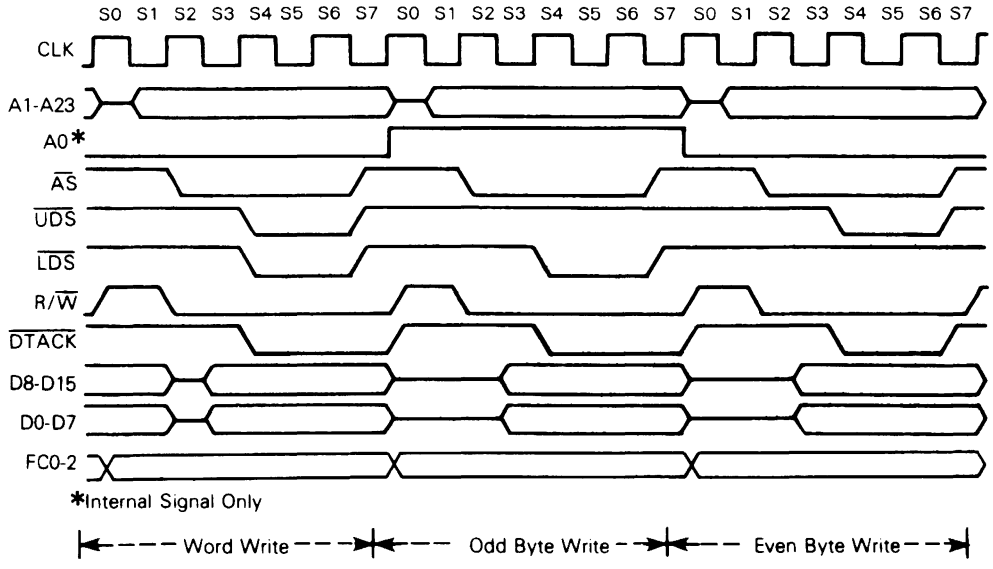


FIGURE 18 — READ-MODIFY-WRITE CYCLE FLOW CHART

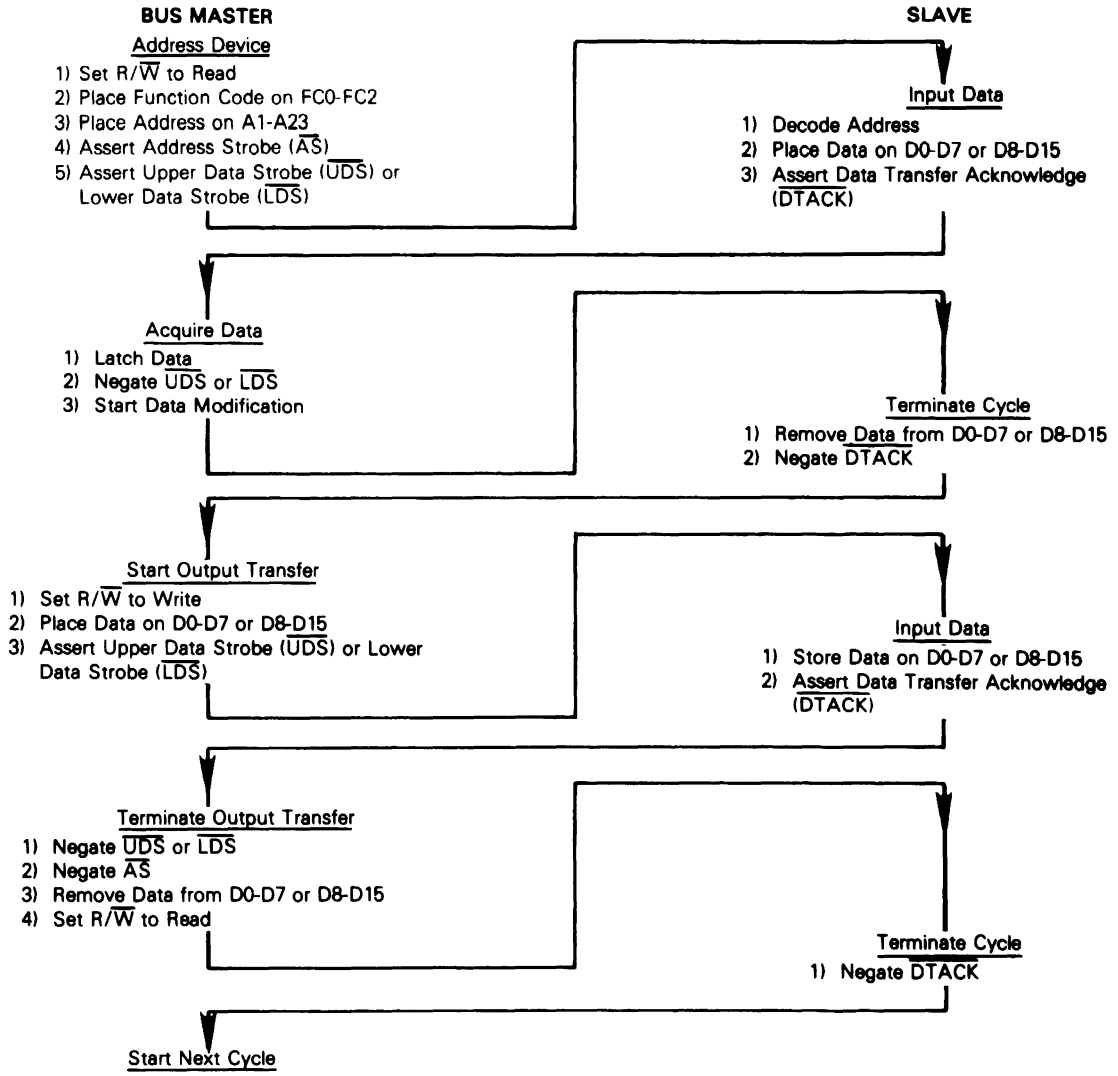


FIGURE 19 — READ-MODIFY-WRITE CYCLE TIMING DIAGRAM

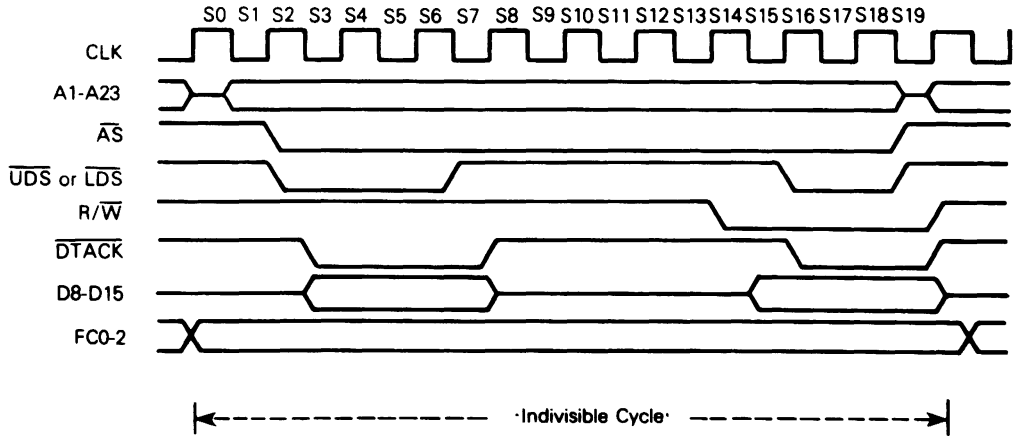
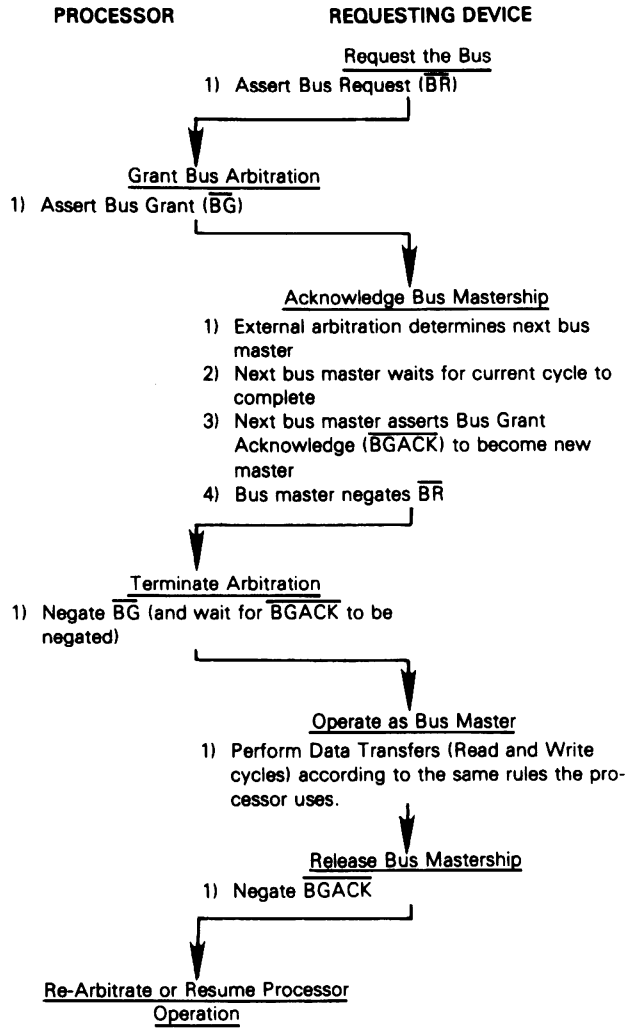


FIGURE 20 — BUS ARBITRATION CYCLE FLOW CHART



Requesting the Bus. External devices capable of becoming bus masters request the bus by asserting the bus request ($\overline{BR}$) signal. This is a wire ORed signal (although it need not be constructed from open collector devices) that indicates to the processor that some external device requires control of the external bus. The processor is effectively at a lower bus priority level than the external device and will relinquish the bus after it has completed the last bus cycle it has started.

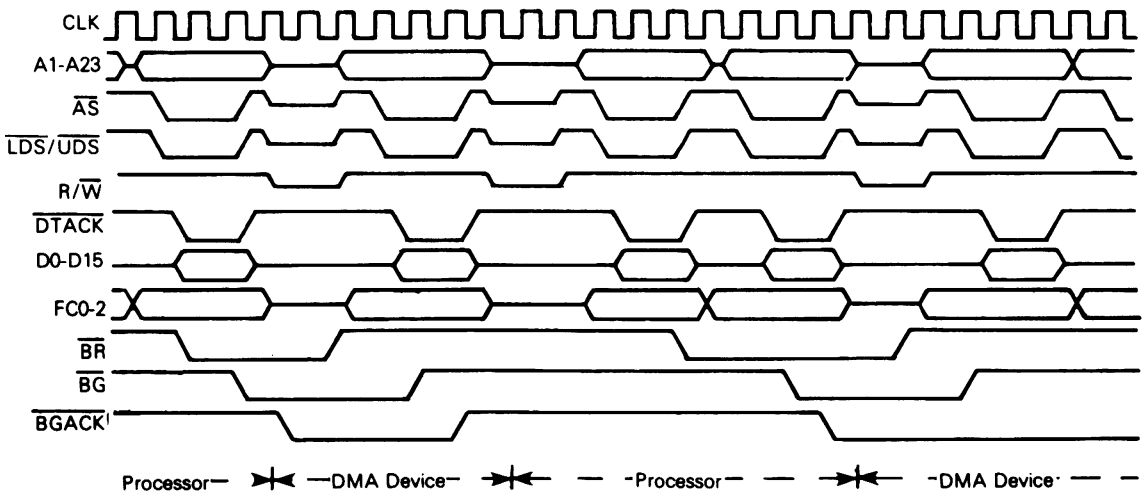
When no acknowledge is received before the bus request signal goes inactive, the processor will continue processing when it detects that the bus request is inactive. This allows ordinary processing to continue if the arbitration circuitry responded to noise inadvertently.

Receiving the Bus Grant. The processor asserts bus grant ($\overline{BG}$) as soon as possible. Normally this is immediately after internal synchronization. The only exception to this occurs when the processor has made an internal decision to execute the next bus cycle but has not progressed far enough into the cycle to have asserted the address strobe ($\overline{AS}$) signal. In this case, bus grant will not be asserted until one clock after address strobe is asserted to indicate to external devices that a bus cycle is being executed.

The bus grant signal may be routed through a daisy-chained network or through a specific priority-encoded network. The processor is not affected by the external method of arbitration as long as the protocol is obeyed.

Acknowledgement of Mastership. Upon receiving a bus grant, the requesting device waits until address strobe, data transfer acknowledge, and bus grant acknowledge are negated before issuing its own $\overline{BGACK}$. The negation of the address strobe indicates that the previous master has completed its cycle, the negation of bus grant acknowledge indicates that the previous master has released the bus. (While address strobe is asserted no device is allowed to "break into" a cycle.) The negation of data transfer acknowledge indicates the previous slave has terminated its connection to the previous master. Note that in some applications data transfer acknowledge might not enter into this function. General purpose devices would then be connected such that

FIGURE 21 — BUS ARBITRATION CYCLE TIMING DIAGRAM



they were only dependent on address strobe. When bus grant acknowledge is issued the device is bus master until it negates bus grant acknowledge. Bus grant acknowledge should not be negated until after the bus cycle(s) is (are) completed. Bus mastership is terminated at the negation of bus grant acknowledge.

The bus request from the granted device should be dropped after bus grant acknowledge is asserted. If a bus request is still pending, another bus grant will be asserted within a few clocks of the negation of bus grant. Refer to Bus Arbitration Control section. Note that the processor does not perform any external bus cycles before it re-asserts bus grant.

BUS ARBITRATION CONTROL. The bus arbitration control unit in the MC68000 is implemented with a finite state machine. A state diagram of this machine is shown in Figure 22. All asynchronous signals to the MC68000 are synchronized before being used internally. This synchronization is accomplished in a maximum of one cycle of the system clock, assuming that the asynchronous input setup time (#47) has been met (see Figure 23). The input signal is sampled on the falling edge of the clock and is valid internally after the next falling edge.

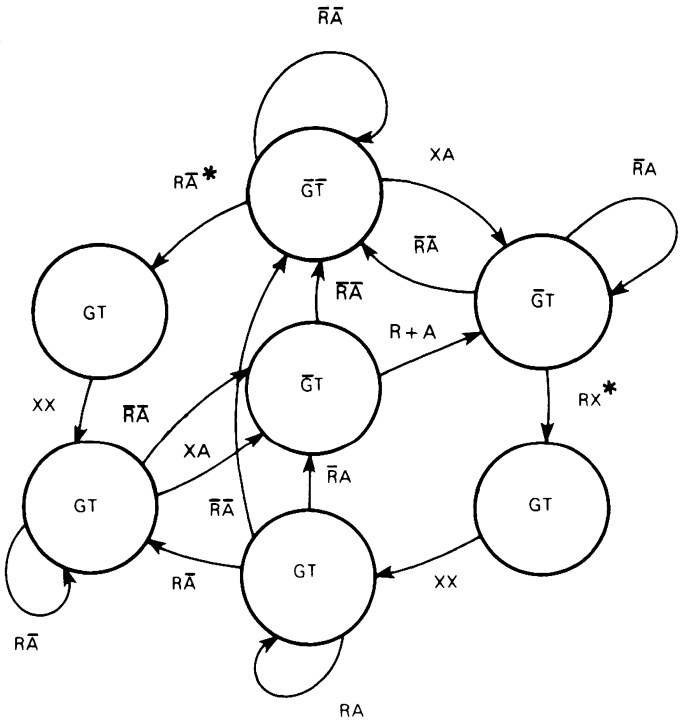
As shown in Figure 22, input signals labeled R and A are internally synchronized on the bus request and bus grant acknowledge pins respectively. The bus grant output is labeled G and the internal three-state control signal T. If T is true, the address, data, and control buses are placed in a high-impedance state when $\overline{AS}$ is negated. All signals are shown in positive logic (active high) regardless of their true active voltage level.

State changes (valid outputs) occur on the next rising edge after the internal signal is valid.

A timing diagram of the bus arbitration sequence during a processor bus cycle is shown in Figure 24. The bus arbitration sequence while the bus is inactive (i.e., executing internal operations such as a multiply instruction) is shown in Figure 25.

If a bus request is made at a time when the MPU has already begun a bus cycle but $\overline{AS}$ has not been asserted (bus state S0), $\overline{BG}$ will not be asserted on the next rising edge. Instead, $\overline{BG}$ will be delayed until the second rising edge following it's internal assertion. This sequence is shown in Figure 26.

FIGURE 22 — STATE DIAGRAM OF MC68000 BUS ARBITRATION UNIT

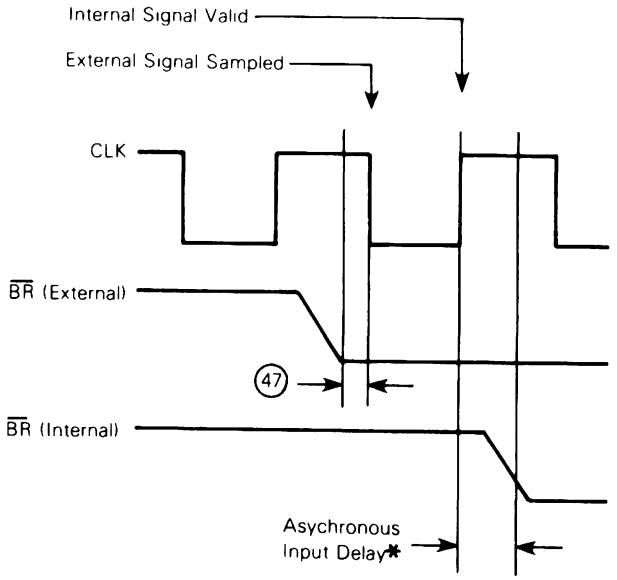


R = Bus Request Internal
A = Bus Grant Acknowledge Internal
G = Bus Grant
T = Three-State Control to Bus Control Logic
X = Don't Care

* State machine will not change state if bus is in S0. Refer to BUS ARBITRATION CONTROL for additional information.

BUS ERROR AND HALT OPERATION. In a bus architecture that requires a handshake from an external device, the possibility exists that the handshake might not occur. Since different systems will require a different maximum response time, a bus error input is provided. External circuitry must be used to determine the duration between address strobe and data transfer acknowledge before issuing a bus error signal. When a bus error signal is received, the processor has two options: initiate a bus error exception sequence or try running the bus cycle again.

FIGURE 23 — TIMING RELATIONSHIP OF EXTERNAL ASYNCHRONOUS INPUTS TO INTERNAL SIGNALS



* This delay time is equal to parameter #33, t_{CHGL}.

FIGURE 24 — BUS ARBITRATION DURING PROCESSOR BUS CYCLE

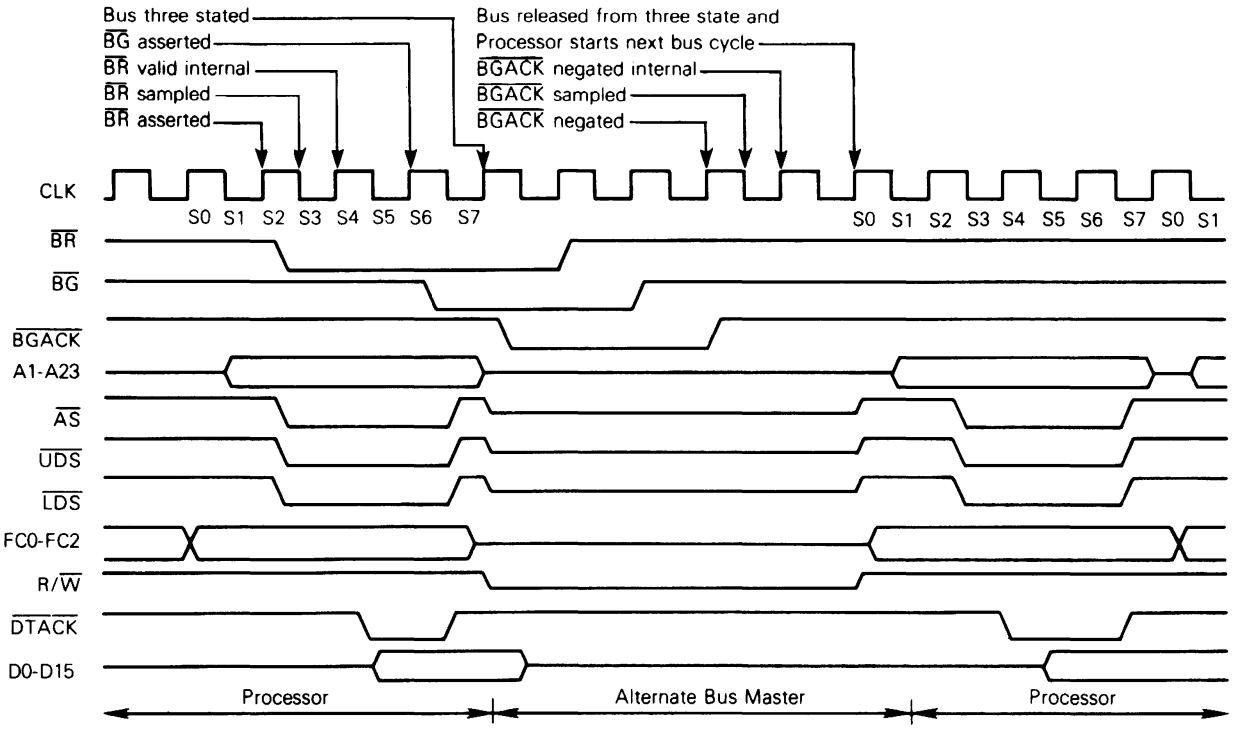


FIGURE 25 — BUS ARBITRATION WITH BUS INACTIVE

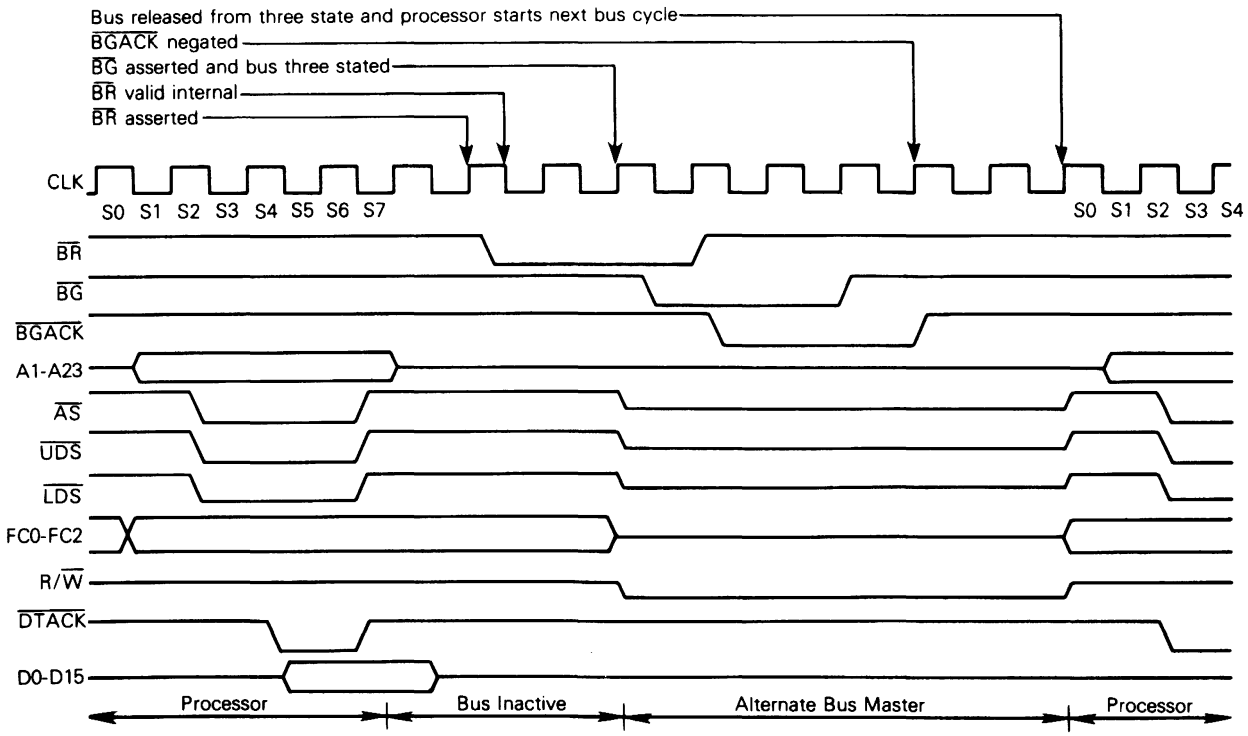
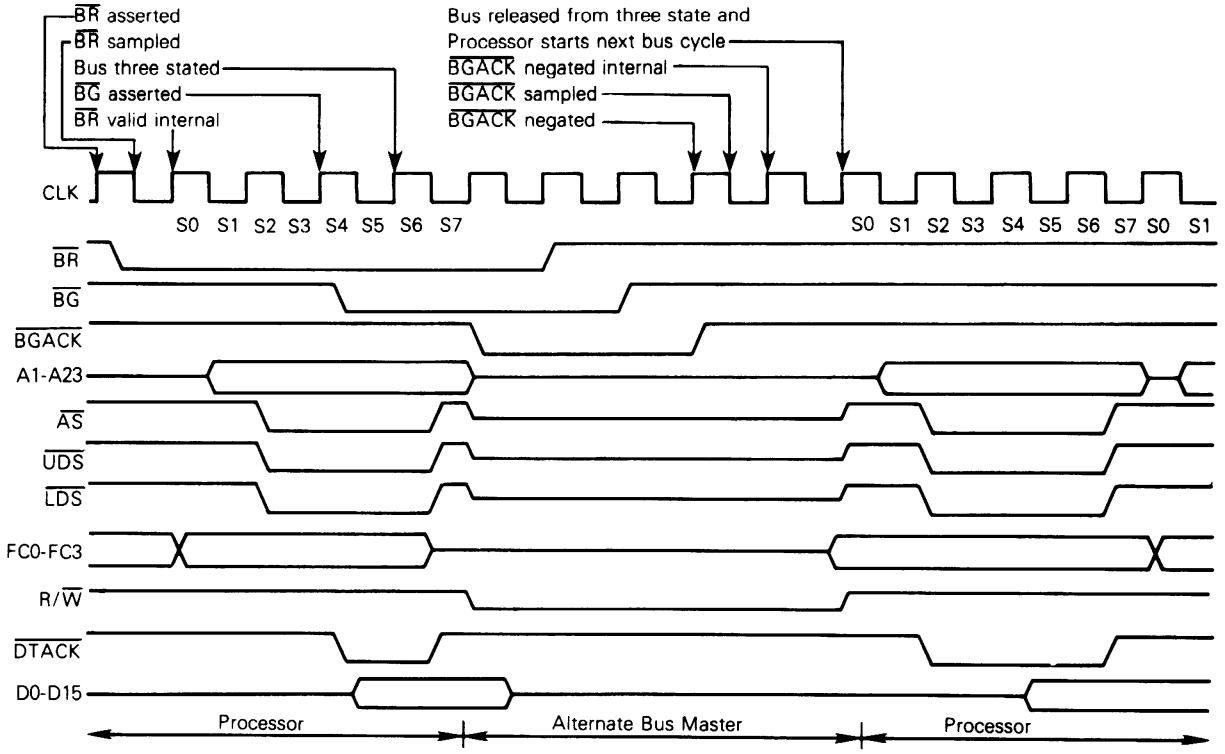


FIGURE 26 — BUS ARBITRATION DURING PROCESSOR BUS CYCLE SPECIAL CASE



Exception Sequence. When the bus error signal is asserted, the current bus cycle is terminated. If BERR is asserted before the falling edge of S4, AS will be negated in S7 in either a read or write cycle. As long as BERR remains asserted, the data and address buses will be in the high-impedance state. When BERR is negated, the processor will begin stacking for exception processing. Figure 27 is a timing diagram for the exception sequence. The sequence is composed of the following elements.

1. Stacking the program counter and status register
2. Stacking the error information
3. Reading the bus error vector table entry
4. Executing the bus error handler routine

The stacking of the program counter and the status register is the same as if an interrupt had occurred. Several additional items are stacked when a bus error occurs. These items are used to determine the nature of the error and correct it, if possible. The bus error vector is vector number two located at address $000008. The processor loads the new program counter from this location. A software bus error handler routine is then executed by the processor. Refer to **EXCEPTION PROCESSING** for additional information.

Re-Running the Bus Cycle. When, during a bus cycle, the processor receives a bus error signal and the halt pin is being driven by an external device, the processor enters the re-run sequence. Figure 28 is a timing diagram for re-running the bus cycle.

The processor terminates the bus cycle, then puts the address and data output lines in the high-impedance state. The processor remains "halted," and will not run another bus cycle until the halt signal is removed by external logic. Then the processor will re-run the previous bus cycle using the same address, the same function codes, the same data (for a write operation), and the same controls. The bus error signal should be removed at least one clock cycle before the halt signal is removed.

NOTE

The processor will not re-run a read-modify-write cycle. This restriction is made to guarantee that the entire cycle runs correctly and that the write operation of a Test-and-Set operation is performed without ever releasing $\overline{AS}$. If $\overline{BERR}$ and $\overline{HALT}$ are asserted during a read-modify-write bus cycle, a bus error operation results.

FIGURE 27 — BUS ERROR TIMING DIAGRAM

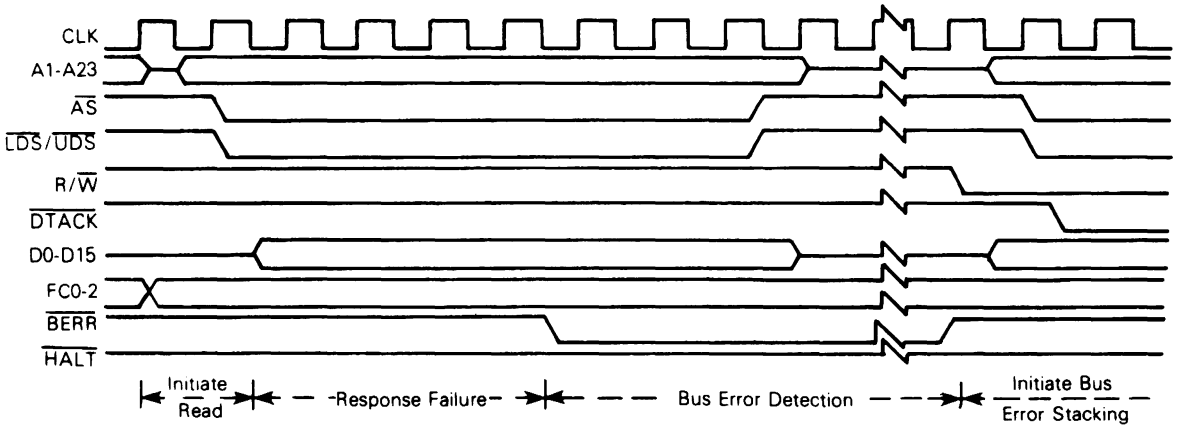
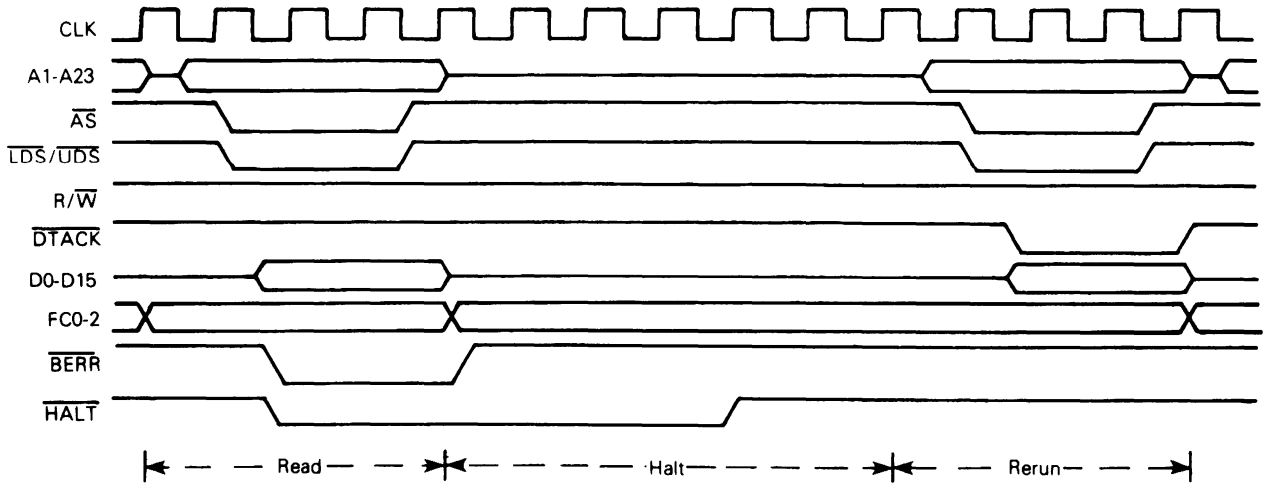


FIGURE 28 — RE-RUN BUS CYCLE TIMING INFORMATION



The processor terminates the bus cycle, then puts the address, data and function code output lines in the high-impedance state. The processor remains "halted," and will not run another bus cycle until the halt signal is removed by external logic. Then the processor will re-run the previous bus cycle using the same address, the same function codes, the same data (for a write operation), and the same controls. The bus error signal should be removed before the halt signal is removed.

Halt Operation with No Bus Error. The halt input signal to the MC68000 performs a Halt/Run/Single-Step function in a similar fashion to the M6800 halt function. The halt and run modes are somewhat self explanatory in that when the halt signal is constantly active the processor "halts" (does nothing) and when the halt signal is constantly inactive the processor "runs" (does something).

The single-step mode is derived from correctly timed transitions on the halt signal input. It forces the processor to execute a single bus cycle by entering the "run" mode until the processor starts a bus cycle then changing to the "halt" mode. Thus, the single-step mode allows the user to proceed through (and therefore debug) processor operations one bus cycle at a time.

Figure 29 details the timing required for correct single-step operations. Some care must be exercised to avoid harmful interactions between the bus error signal and the halt pin when using the single cycle mode as a debugging tool. This is also true of interactions between the halt and reset lines since these can reset the machine.

When the processor completes a bus cycle after recognizing that the halt signal is active, most three-state signals are put in the high-impedance state. These include:

1. address lines
2. data lines

This is required for correct performance of the re-run bus cycle operation.

While the processor is honoring the halt request, bus arbitration performs as usual. That is, halting has no effect on bus arbitration. It is the bus arbitration function that removes the control signals from the bus.

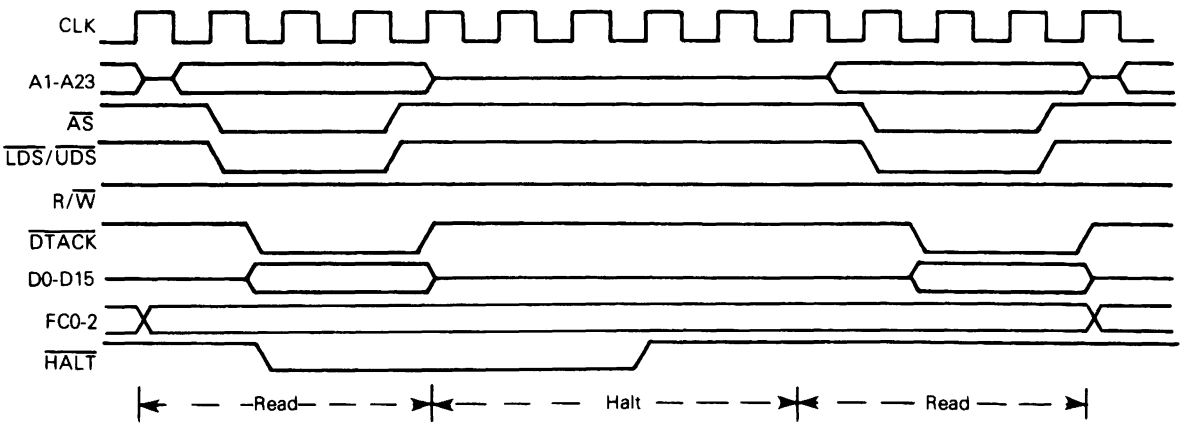
The halt function and the hardware trace capability allow the hardware debugger to trace single bus cycles or single instructions at a time. These processor capabilities, along with a software debugging package, give total debugging flexibility.

Double Bus Faults. When a bus error exception occurs, the processor will attempt to stack several words containing information about the state of the machine. If a bus error exception occurs during the stacking operation, there have been two bus errors in a row. This is commonly referred to as a double bus fault. When a double bus fault occurs, the processor will halt. Once a bus error exception has occurred, any bus error exception occurring before the execution of the next instruction constitutes a double bus fault.

Note that a bus cycle which is re-run does not constitute a bus error exception, and does not contribute to a double bus fault. Note also that this means that as long as the external hardware requests it, the processor will continue to re-run the same bus cycle.

The bus error pin also has an effect on processor operation after the processor receives an external reset input. The processor reads the vector table after a reset to determine the address to start program execution. If a bus error occurs while reading the vector table (or at any time before the first instruction is executed), the processor reacts as if a double bus fault has occurred and it halts. Only an external reset will start a halted processor.

FIGURE 29 — HALT SIGNAL TIMING CHARACTERISTICS



THE RELATIONSHIP OF $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, AND $\overline{\text{HALT}}$

In order to properly control termination of a bus cycle for a re-run or a bus error condition, $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, and $\overline{\text{HALT}}$ should be asserted and negated on the rising edge of the MC68000 clock. This will assure that when two signals are asserted simultaneously, the required setup time (#47) for both of them will be met during the same bus state.

This, or some equivalent precaution, should be designed external to the MC68000. Parameter #48 is intended to ensure this operation in a totally asynchronous system, and may be ignored if the above conditions are met.

The preferred bus cycle terminations may be summarized as follows (case numbers refer to Table 4):

Normal Termination: $\overline{\text{DTACK}}$ occurs first (case 1).

Halt Termination: $\overline{\text{HALT}}$ is asserted at same time, or precedes $\overline{\text{DTACK}}$ (no $\overline{\text{BERR}}$) cases 2 and 3.

Bus Error Termination: $\overline{\text{BERR}}$ is asserted in lieu of, at same time, or preceding $\overline{\text{DTACK}}$ (case 4); BERR negated at same time, or after $\overline{\text{DTACK}}$.

Re-Run Termination: $\overline{\text{HALT}}$ and $\overline{\text{BERR}}$ asserted at the same time, or before $\overline{\text{DTACK}}$ (cases 6 and 7); $\overline{\text{HALT}}$ must be negated at least 1 cycle after $\overline{\text{BERR}}$. (Case 5 indicates $\overline{\text{BERR}}$ may precede $\overline{\text{HALT}}$ on all except R9M and T6E <early mask sets> which allows fully asynchronous assertion).

Table 4 details the resulting bus cycle termination under various combinations of control signal sequences. The negation of these same control signals under several conditions is shown in Table 5 ($\overline{\text{DTACK}}$ is assumed to be negated normally in all cases; for best results, both $\overline{\text{DTACK}}$ and $\overline{\text{BERR}}$ should be negated when address strobe is negated.)

Example A: A system uses a watch-dog timer to terminate accesses to un-populated address space. The timer asserts $\overline{\text{DTACK}}$ and $\overline{\text{BERR}}$ simultaneously after time-out. (case 4)

Example B: A system uses error detection on RAM contents. Designer may (a) delay $\overline{\text{DTACK}}$ until data verified, and return $\overline{\text{BERR}}$ and $\overline{\text{HALT}}$ simultaneously to re-run error cycle (case 6), or if valid, return $\overline{\text{DTACK}}$; (b) delay $\overline{\text{DTACK}}$ until data verified, and return $\overline{\text{BERR}}$ at same time as $\overline{\text{DTACK}}$ if data in error (case 4); (c) return $\overline{\text{DTACK}}$ prior to data verification, as described in previous section. If data invalid, $\overline{\text{BERR}}$ is asserted (case 1) in next cycle. Error-handling software must know how to recover error cycle.

TABLE 4 — $\overline{\text{DTACK}}$, $\overline{\text{BERR}}$, $\overline{\text{HALT}}$ ASSERTION RESULTS

Case No.	Control Signal	Asserted on Rising Edge of State N	Asserted on Rising Edge of State N+2	Result
1	$\overline{\text{DTACK}}$	A	S	Normal cycle terminate and continue.
	$\overline{\text{BERR}}$	NA	X	
	$\overline{\text{HALT}}$	NA	X	
2	$\overline{\text{DTACK}}$	A	S	Normal cycle terminate and halt. Continue when HALT removed.
	$\overline{\text{BERR}}$	NA	X	
	$\overline{\text{HALT}}$	A	S	
3	$\overline{\text{DTACK}}$	NA	A	Normal cycle terminate and halt. Continue when HALT removed.
	$\overline{\text{BERR}}$	NA	NA	
	$\overline{\text{HALT}}$	A	S	
4	$\overline{\text{DTACK}}$	X	X	Terminate and take bus error trap.
	$\overline{\text{BERR}}$	A	S	
	$\overline{\text{HALT}}$	NA	NA	
5	$\overline{\text{DTACK}}$	NA	X	**R9M, T6E, BF4:** Unpredictable results, no re-run, no error trap; usually traps to vector number 0. **All others:** terminate and re-run.
	$\overline{\text{BERR}}$	A	S	
	$\overline{\text{HALT}}$	NA	A	
6	$\overline{\text{DTACK}}$	X	X	Terminate and re-run.
	$\overline{\text{BERR}}$	A	S	
	$\overline{\text{HALT}}$	A	S	
7	$\overline{\text{DTACK}}$	NA	X	Terminate and re-run when HALT removed.
	$\overline{\text{BERR}}$	NA	A	
	$\overline{\text{HALT}}$	A	S	

Legend:
- N — the number of the current even bus state (e.g., S4, S6, etc.)
- A — signal is asserted in this bus state
- NA — signal is not asserted in this state
- X — don't care
- S — signal was asserted in previous state and remains asserted in this state

TABLE 5 — $\overline{\text{BERR}}$ AND $\overline{\text{HALT}}$ NEGATION RESULTS

Conditions of Termination in Table A	Control Signal	Negated on Rising Edge of State N		Negated on Rising Edge of State N+2	Results — Next Cycle
Bus Error	$\overline{\text{BERR}}$	●	or	●	Takes bus error trap.
	$\overline{\text{HALT}}$	●	or	●	
Re-run	$\overline{\text{BERR}}$	●	or	●	Illegal sequence; usually traps to vector number 0.
	$\overline{\text{HALT}}$	●			
Re-run	$\overline{\text{BERR}}$	●			Re-runs the bus cycle.
	$\overline{\text{HALT}}$			●	
Normal	$\overline{\text{BERR}}$	●			May lengthen next cycle.
	$\overline{\text{HALT}}$	●	or	●	
Normal	$\overline{\text{BERR}}$			●	If next cycle is started it will be terminated as a bus error.
	$\overline{\text{HALT}}$	●	or	none	

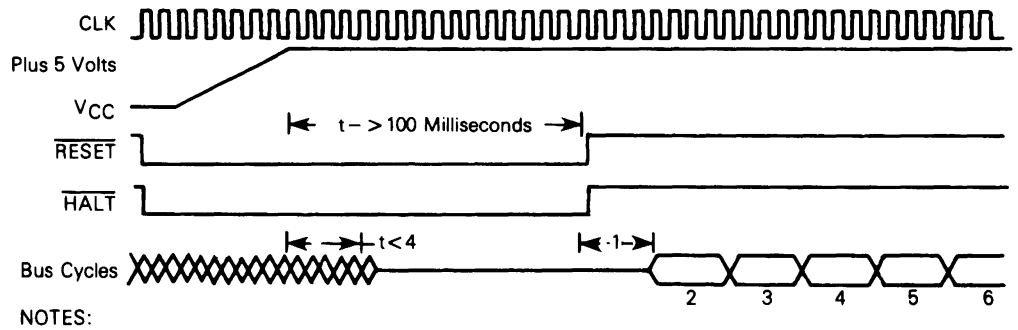
RESET OPERATION. The reset signal is a bidirectional signal that allows either the processor or an external signal to reset the system. Figure 30 is a timing diagram for reset operations. Both the halt and reset lines must be applied to ensure total reset of the processor.

When the reset and halt lines are driven by an external device, it is recognized as an entire system reset, including the processor. The processor responds by reading the reset vector table entry (vector number zero, address $000000) and loads it into the supervisor stack pointer (SSP). Vector table entry number one at address $000004 is read next and loaded into the program counter. The processor initializes the status register to an interrupt level of seven. No other registers are affected by the reset sequence.

When a RESET sequence is executed, the processor drives the reset pin for 124 clock pulses. In this case, the processor is trying to reset the rest of the system. Therefore, there is no effect on the internal state of the processor. All of the processor's internal registers and the status register are unaffected by the execution of a RESET instruction. All external devices connected to the reset line should be reset at the completion of the RESET instruction.

Asserting the Reset and Halt pins for 10 clock cycles will cause a processor reset, except when V_{CC} is initially applied to the processor. In this case, an external reset must be applied for 100 milliseconds.

FIGURE 30 — RESET OPERATION TIMING DIAGRAM



NOTES:

1) Internal start-up time
2) SSP High read in here
3) SSP Low read in here
4) PC High read in here
5) PC Low read in here
6) First instruction fetched here.

Bus State Unknown:

All Control Signals Inactive.
Data Bus In Read Mode:

PROCESSING STATES

The MC68000 is always in one of three processing states: normal, exception, or halted. The normal processing state is that associated with instruction execution; the memory of the bits in the supervisor portion of the status register are covered: the supervisor/user bit, the trace enable bit, and the processor interrupt priority mask. Finally, the sequence of memory references and actions taken by the processor on exception conditions is detailed.

The MC68000 is always in one of three processing states: normal, exception, or halted. The normal processing state is that associated with instruction execution; the memory references are to fetch instructions and operands, and to store results. A special case of the normal state is the stopped state which the processor enters when a STOP instruction is executed. In this state, no further memory references are made.

The exception processing state is associated with interrupts, trap instructions, tracing and other exceptional conditions. The exception may be internally generated by an instruction or by an unusual condition arising during the execution of an instruction. Externally, exception processing can be forced by an interrupt, by a bus error, or by a reset. Exception processing is designed to provide an efficient context switch so that the processor may handle unusual conditions.

The halted processing state is an indication of catastrophic hardware failure. For example, if during the exception processing of a bus error another bus error occurs, the processor assumes that the system is unusable and halts. Only an external reset can restart a halted processor. Note that a processor in the stopped state is not in the halted state, nor vice versa.

PRIVILEGE STATES

The processor operates in one of two states of privilege: the "user" state or the "supervisor" state. The privilege state determines which operations are legal, is used by the external memory management device to control and translate accesses, and is used to choose between the supervisor stack pointer and the user stack pointer in instruction references.

The privilege state is a mechanism for providing security in a computer system. Programs should access only their own code and data areas, and ought to be restricted from accessing information which they do not need and must not modify.

The privilege mechanism provides security by allowing most programs to execute in user state. In this state, the accesses are controlled, and the effects on other parts of the system are limited. The operating system executes in the supervisor state, has access to all resources, and performs the overhead tasks for the user state programs.

SUPERVISOR STATE. The supervisor state is the higher state of privilege. For instruction execution, the supervisor state is determined by the S-bit of the status register; if the S-bit is asserted (high), the processor is in the supervisor state. All instructions can be executed in the supervisor state. The bus cycles generated by instructions executed in the supervisor state are classified as supervisor references. While the processor is in the supervisor privilege state, those instructions which use either the system stack pointer implicitly or address register seven explicitly access the supervisor stack pointer.

All exception processing is done in the supervisor state, regardless of the setting of the S-bit. The bus cycles generated during exception processing are classified as supervisor references. All stacking operations during exception processing use the supervisor stack pointer.

USER STATE. The user state is the lower state of privilege. For instruction execution, the user state is determined by the S-bit of the status register; if the S-bit is negated (low), the processor is executing instructions in the user state.

Most instructions execute the same in user state as in the supervisor state. However, some instructions which have important system effects are made privileged. User programs are not permitted to execute the STOP instruction, or the RESET instruction. To ensure that a user program cannot enter the supervisor state except in a controlled manner, the instructions which modify the whole status register are privileged. To aid in debugging programs which are to be used as operating systems, the move to user stack pointer (MOVE USP) and move from user stack pointer (MOVE from USP) instructions are also privileged.

The bus cycles generated by an instruction executed in user state are classified as user state references. This allows an external memory management device to translate the address and to control access to protected portions of the address space. While the processor is in the user privilege state, those instructions which use either the system stack pointer implicitly, or address register seven explicitly, access the user stack pointer.

PRIVILEGE STATE CHANGES. Once the processor is in the user state and executing instructions, only exception processing can change the privilege state. During exception processing, the current setting of the S-bit of the status register is saved and the S-bit is asserted, putting the processing in the supervisor state. Therefore, when instruction execution resumes at the address specified to process the exception, the processor is in the supervisor privilege state.

REFERENCE CLASSIFICATION. When the processor makes a reference, it classifies the kind of reference being made, using the encoding on the three function code output lines. This allows external translation of addresses, control of access, and differentiation of special processor states, such as interrupt acknowledge. Table 6 lists the classification of references.

TABLE 6 – REFERENCE CLASSIFICATION

Function Code Output			Reference Class
FC2	FC1	FC0	
0	0	0	(Unassigned)
0	0	1	User Data
0	1	0	User Program
0	1	1	(Unassigned)
1	0	0	(Unassigned)
1	0	1	Supervisor Data
1	1	0	Supervisor Program
1	1	1	Interrupt Acknowledge

EXCEPTION PROCESSING

Before discussing the details of interrupts, traps, and tracing, a general description of exception processing is in order. The processing of an exception occurs in four steps, with variations for different exception causes. During the first step, a temporary copy of the status register is made, and the status register is set for exception processing. In the second step the exception vector is determined, and the third step is the saving of the current processor context. In the fourth step a new context is obtained, and the processor switches to instruction processing.

EXCEPTION VECTORS. Exception vectors are memory locations from which the processor fetches the address of a routine which will handle that exception. All exception vectors are two words in length (Figure 31), except for the reset vector, which is four words. All exception vectors lie in the supervisor data space, except for the reset vector which is in the supervisor program space. A vector number is an eight-bit number which, when multiplied by four, gives the address of an exception vector. Vector numbers are generated internally or externally, depending on the cause of the exception. In the case of interrupts, during the interrupt acknowledge bus cycle, a peripheral provides an 8-bit vector number (Figure 32) to the processor on data bus lines D0 through D7. The processor translates the vector number into a full 24-bit address, as shown in Figure 33. The memory layout for exception vectors is given in Table 7.

As shown in Table 7, the memory layout is 512 words long (1024 bytes). It starts at address 0 and proceeds through address 1023. This provides 255 unique vectors; some of these are reserved for TRAPS and other system functions. Of the 255, there are 192 reserved for user interrupt vectors. However, there is no protection on the first 64 entries, so user interrupt vectors may overlap at the discretion of the systems designer.

KINDS OF EXCEPTIONS. Exceptions can be generated by either internal or external causes. The externally generated exceptions are the interrupts and the bus error and reset requests. The interrupts are requests from peripheral devices for processor action while the bus error and reset inputs are used for access control and processor restart. The internally generated exceptions come from instructions, or from ad-

FIGURE 31 — EXCEPTION VECTOR FORMAT

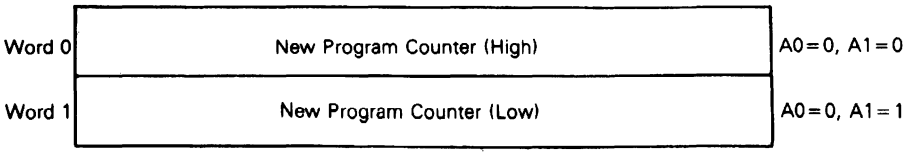
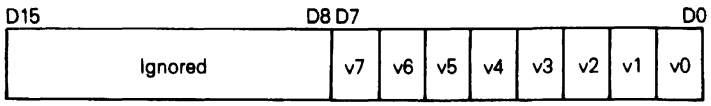


FIGURE 32 — PERIPHERAL VECTOR NUMBER FORMAT



Where:
v7 is the MSB of the Vector Number
v0 is the LSB of the Vector Number

FIGURE 33 — ADDRESS TRANSLATED FROM 8-BIT VECTOR NUMBER

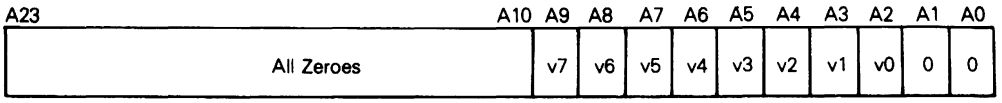


TABLE 7 — EXCEPTION VECTOR ASSIGNMENT

Vector Number(s)	Address			Assignment
	Dec	Hex	Space	
0	0	000	SP	Reset: Initial SSP
–	4	004	SP	Reset: Initial PC
2	8	008	SD	Bus Error
3	12	00C	SD	Address Error
4	16	010	SD	Illegal Instruction
5	20	014	SD	Zero Divide
6	24	018	SD	CHK Instruction
7	28	01C	SD	TRAPV Instruction
8	32	020	SD	Privilege Violation
9	36	024	SD	Trace
10	40	028	SD	Line 1010 Emulator
11	44	02C	SD	Line 1111 Emulator
12*	48	030	SD	(Unassigned, reserved)
13*	52	034	SD	(Unassigned, reserved)
14*	56	038	SD	(Unassigned, reserved)
15	60	03C	SD	Uninitialized Interrupt Vector
16-23*	64	04C	SD	(Unassigned, reserved)
	95	05F		–
24	96	060	SD	Spurious Interrupt
25	100	064	SD	Level 1 Interrupt Autovector
26	104	068	SD	Level 2 Interrupt Autovector
27	108	06C	SD	Level 3 Interrupt Autovector
28	112	070	SD	Level 4 Interrupt Autovector
29	116	074	SD	Level 5 Interrupt Autovector
30	120	078	SD	Level 6 Interrupt Autovector
31	124	07C	SD	Level 7 Interrupt Autovector
32-47	128	080	SD	TRAP Instruction Vectors
	191	0BF		–
48-63*	192	0C0	SD	(Unassigned, reserved)
	255	0FF		–
64-255	256	100	SD	User Interrupt Vectors
	1023	3FF		–

*Vector numbers 12, 13, 14, 16 through 23 and 48 through 63 are reserved for future enhancements by Motorola. No user peripheral devices should be assigned these numbers.

dress errors or tracing. The trap (TRAP), trap on overflow (TRAPV), check register against bounds (CHK) and divide (DIV) instructions all can generate exceptions as part of their instruction execution. In addition, illegal instructions, word fetches from odd addresses and privilege violations cause exceptions. Tracing behaves like a very high priority, internally generated interrupt after each instruction execution.

EXCEPTION PROCESSING SEQUENCE. Exception processing occurs in four identifiable steps. In the first step, an internal copy is made of the status register. After the copy is made, the S-bit is asserted, putting the processor into the supervisor privilege state. Also, the T-bit is negated which will allow the exception handler to execute unhindered by tracing. For the reset and interrupt exceptions, the interrupt priority mask is also updated.

In the second step, the vector number of the exception is determined. For interrupts, the vector number is obtained by a processor fetch, classified as an interrupt acknowledge. For all other exceptions, internal logic provides the vector number. This vector number is then used to generate the address of the exception vector.

The third step is to save the current processor status, except for the reset exception. The current program counter value and the saved copy of the status register are stacked using the supervisor stack pointer. The program counter value stacked usually points to the next unexecuted instruction, however for bus error and address error, the value stacked for the program counter is unpredictable, and may be incremented from the address of the instruction which caused the error. Additional information defining the current context is stacked for the bus error and address error exceptions.

The last step is the same for all exceptions. The new program counter value is fetched from the exception vector. The processor then resumes instruction execution. The instruction at the address given in the exception vector is fetched, and normal instruction decoding and execution is started.

MULTIPLE EXCEPTIONS. These paragraphs describe the processing which occurs when multiple exceptions arise simultaneously. Exceptions can be grouped according to their occurrence and priority. The Group 0 exceptions are reset, bus error, and address error. These exceptions cause the instruction currently being executed to be aborted, and the exeception processing to commence within two clock cycles. The Group 1 exceptions are trace and interrupt, as well as the privilege violations and illegal instructions. These exceptions allow the current instruction to execute to completion, but preempt the execution of the next instruction by forcing exception processing to occur (privilege violations and illegal instructions are detected when they are the next instruction to be executed). The Group 2 exceptions occur as part of the normal processing of instructions. The TRAP, TRAPV, CHK, and zero divide exceptions are in this group. For these exceptions, the normal execution of an instruction may lead to exception processing.

Group 0 exceptions have highest priority, while Group 2 exceptions have lowest priority. Within Group 0, reset has highest priority, followed by bus error and then address error. Within Group 1, trace has priority over external interrupts, which in turn takes priority over illegal instruction and privilege violation. Since only one instruction can be executed at a time, there is no priority relation within Group 2.

The priority relation between two exceptions determines which is taken, or taken first, if the conditions for both arise simultaneously. Therefore, if a bus error occurs during a TRAP instruction, the bus error takes precedence, and the TRAP instruction processing is aborted. In another example, if an interrupt request occurs during the execution of an instruction while the T-bit is asserted, the trace exception has priority, and is processed first. Before instruction processing resumes, however, the interrupt exception is also processed, and instruction processing commences finally in the interrupt handler routine. A summary of exception grouping and priority is given in Table 8.

TABLE 8 — EXCEPTION GROUPING AND PRIORITY

Group	Exception	Processing
0	Reset Bus Error Address Error	Exception processing begins within two clock cycles.
1	Trace Interrupt Illegal Privilege	Exception processing begins before the next instruction
2	TRAP, TRAPV, CHK, Zero Divide	Exception processing is started by normal instruction execution

EXCEPTION PROCESSING DETAILED DISCUSSION

Exceptions have a number of sources, and each exception has processing which is peculiar to it. The following paragraphs detail the sources of exceptions, how each arises, and how each is processed.

RESET. The reset input provides the highest exception level. The processing of the reset signal is designed for system initiation, and recovery from catastrophic failure. Any processing in progress at the time of the reset is aborted and cannot be recovered. The processor is forced into the supervisor state, and the trace state is forced off. The processor interrupt priority mask is set at level seven. The vector number is internally generated to reference the reset exception vector at location 0 in the supervisor program space. Because no assumptions can be made about the validity of register contents, in particular the supervisor stack pointer, neither the program counter nor the status register is saved. The address contained in the first two words of the reset exception vector is fetched as the initial supervisor stack pointer, and the address in the last two words of the reset exception vector is fetched as the initial program counter. Finally, instruction execution is started at the address in the program counter. The power-up/restart code should be pointed to by the initial program counter.

The RESET instruction does not cause loading of the reset vector, but does assert the reset line to reset external devices. This allows the software to reset the system to a known state and then continue processing with the next instruction.

INTERRUPTS. Seven levels of interrupt priorities are provided. Devices may be chained externally within interrupt priority levels, allowing an unlimited number of peripheral devices to interrupt the processor. Interrupt priority levels

are numbered from one to seven, level seven being the highest priority. The status register contains a three-bit mask which indicates the current processor priority, and interrupts are inhibited for all priority levels less than or equal to the current processor priority.

An interrupt request is made to the processor by encoding the interrupt request level on the interrupt request lines; a zero indicates no interrupt request. Interrupt requests arriving at the processor do not force immediate exception processing, but are made pending. Pending interrupts are detected between instruction executions. If the priority of the pending interrupt is lower than or equal to the current processor priority, execution continues with the next instruction and the interrupt exception processing is postponed. (The recognition of level seven is slightly different, as explained in a following paragraph.)

If the priority of the pending interrupt is greater than the current processor priority, the exception processing sequence is started. First a copy of the status register is saved, and the privilege state is set to supervisor, tracing is suppressed, and the processor priority level is set to the level of the interrupt being acknowledged. The processor fetches the vector number from the interrupting device, classifying the reference as an interrupt acknowledge and displaying the level number of the interrupt being acknowledged on the address bus. If external logic requests an automatic vectoring, the processor internally generates a vector number which is determined by the interrupt level number. If external logic indicates a bus error, the interrupt is taken to be spurious, and the generated vector number references the spurious interrupt vector. The processor then proceeds with the usual exception processing, saving the program counter and status register on the supervisor stack. The saved value of the program counter is the address of the instruction which would have been executed had the interrupt not been present. The content of the interrupt vector whose vector number was previously obtained is fetched and loaded into the program counter, and normal instruction execution commences in the interrupt handling routine. A flow chart for the interrupt acknowledge sequence is given in Figure 34, a timing diagram is given in Figure 35, and the interrupt exception timing sequence is shown in Figure 36.

FIGURE 34 — INTERRUPT ACKNOWLEDGE SEQUENCE FLOW CHART

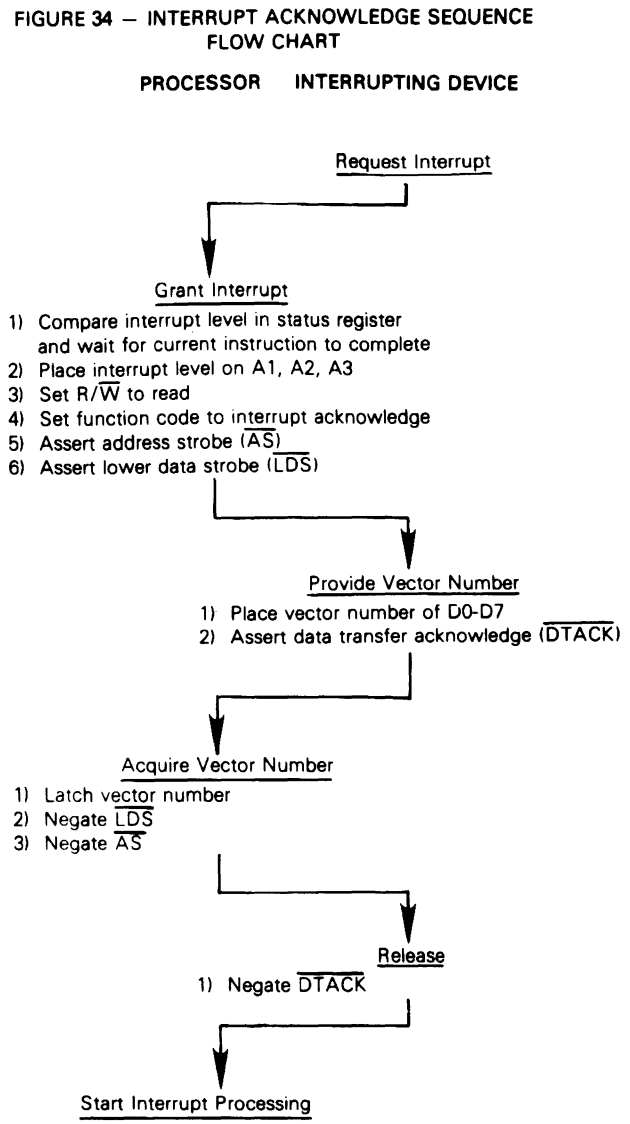


FIGURE 35 — INTERRUPT ACKNOWLEDGE SEQUENCE TIMING DIAGRAM

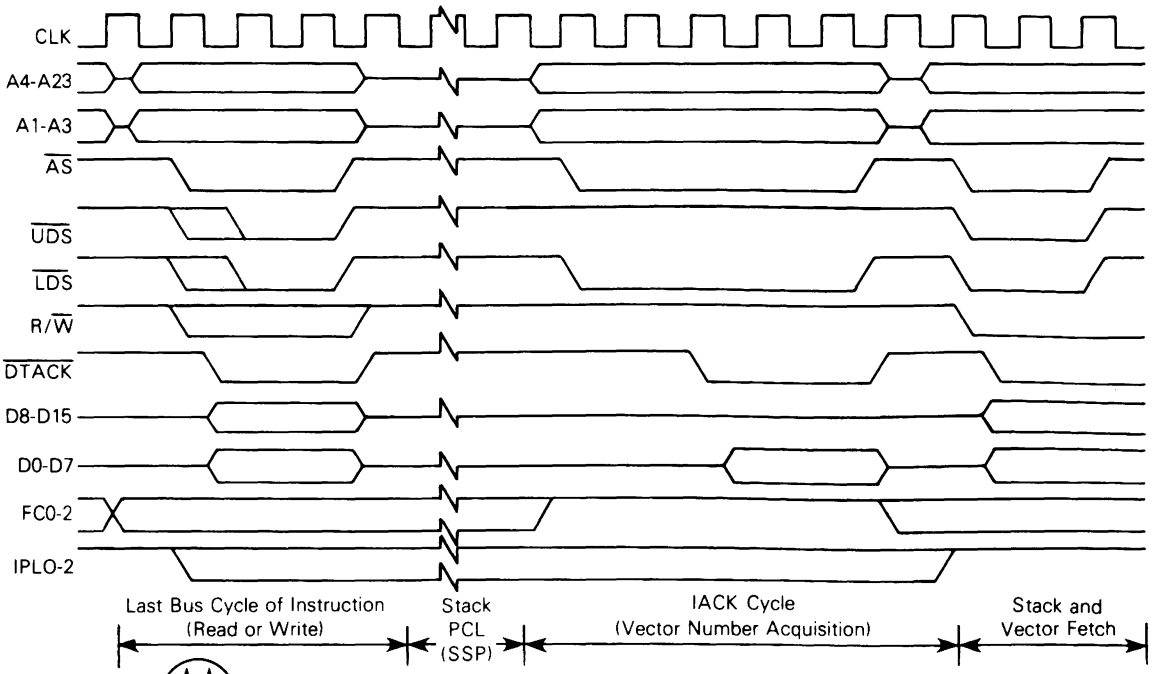
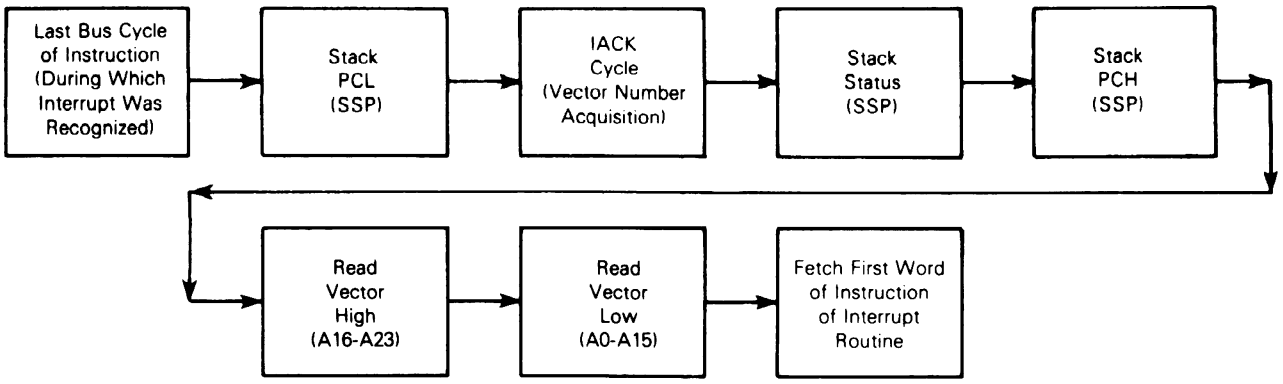


FIGURE 36 — INTERRUPT EXCEPTION TIMING SEQUENCE



Priority level seven is a special case. Level seven interrupts cannot be inhibited by the interrupt priority mask, thus providing a "non-maskable interrupt" capability. An interrupt is generated each time the interrupt request level changes from some lower level to level seven. Note that a level seven interrupt may still be caused by the level comparison if the request level is a seven and the processor priority is set to a lower level by an instruction.

UNINITIALIZED INTERRUPT. An interrupting device asserts $\overline{VPA}$ or provides an interrupt vector during an interrupt acknowledge cycle to the MC68000. If the vector register has not been initialized, the responding M68000 Family peripheral will provide vector 15, the unitialized interrupt vector. This provides a uniform way to recover from a programming error.

SPURIOUS INTERRUPT. If during the interrupt acknowledge cycle no device responds by asserting $\overline{DTACK}$ or $\overline{VPA}$, the bus error line should be asserted to terminate the vector acquisition. The processor separates the processing of this error from bus error by fetching the spurious interrupt vector instead of the bus error vector. The processor then proceeds with the usual exception processing.

INSTRUCTION TRAPS. Traps are exceptions caused by instructions. They arise either from processor recognition of abnormal conditions during instruction execution, or from use of instructions whose normal behavior is trapping.

Some instructions are used specifically to generate traps. The TRAP instruction always forces an exception, and is useful for implementing system calls for user programs. The TRAPV and CHK instructions force an exception if the user program detects a runtime error, which may be an arithmetic overflow or a subscript out of bounds.

The signed divide (DIVS) and unsigned divide (DIVU) instructions will force an exception if a division operation is attempted with a divisor of zero.

ILLEGAL AND UNIMPLEMENTED INSTRUCTIONS. Illegal instruction is the term used to refer to any of the word bit patterns which are not the bit pattern of the first word of a legal instruction. During instruction execution, if such an instruction is fetched, an illegal instruction exception occurs.

Word patterns with bits 15 through 12 equaling 1010 or 1111 are distinguished as unimplemented instructions and separate exception vectors are given to these patterns to permit efficient emulation. This facility allows the operating system to detect program errors, or to emulate unimplemented instructions in software.

PRIVILEGE VIOLATIONS. In order to provide system security, various instructions are privileged. An attempt to execute one of the privileged instructions while in the user state will cause an exception. The privileged instructions are:

STOP	AND (word) Immediate to SR
RESET	EOR (word) Immediate to SR
RTE	OR (word) Immediate to SR
MOVE to SR	MOVE USP

TRACING. To aid in program development, the MC68000 includes a facility to allow instruction by instruction tracing. In the trace state, after each instruction is executed an exception is forced, allowing a debugging program to monitor the execution of the program under test.

The trace facility uses the T-bit in the supervisor portion of the status register. If the T-bit is negated (off), tracing is disabled, and instruction execution proceeds from instruction to instruction as normal. If the T-bit is asserted (on) at the beginning of the execution of an instruction, a trace exception will be generated after the execution of that instruction is completed. If the instruction is not executed, either because an interrupt is taken, or the instruction is illegal or privileged, the trace exception does not occur. The trace exception also does not occur if the instruction is aborted by a reset, bus error, or address error exception. If the instruction is indeed executed and an interrupt is pending on completion, the trace exception is processed before the interrupt exception. If, during the execution of the instruction, an exception is forced by that instruction, the forced exception is processed before the trace exception.

As an extreme illustration of the above rules, consider the arrival of an interrupt during the execution of a TRAP instruction while tracing is enabled. First the trap exception is processed, then the trace exception, and finally the interrupt exception. Instruction execution resumes in the interrupt handler routine.

BUS ERROR. Bus error exceptions occur when the external logic requests that a bus error be processed by an exception. The current bus cycle which the processor is making is then aborted. Whether the processor was doing instruction or exception processing, that processing is terminated, and the processor immediately begins exception processing.

Exception processing for bus error follows the usual sequence of steps. The status register is copied, the supervisor state is entered, and the trace state is turned off. The vector number is generated to refer to the bus error vector. Since the processor was not between instructions when the bus er-

ror exception request was made, the context of the processor is more detailed. To save more of this context, additional information is saved on the supervisor stack. The program counter and the copy of the status register are of course saved. The value saved for the program counter is advanced by some amount, two to ten bytes beyond the address of the first word of the instruction which made the reference causing the bus error. If the bus error occurred during the fetch of the next instruction, the saved program counter has a value in the vicinity of the current instruction, even if the current instruction is a branch, a jump, or a return instruction. Besides the usual information, the processor saves its internal copy of the first word of the instruction being processed, and the address which was being accessed by the aborted bus cycle. Specific information about the access is also saved: whether it was a read or a write, whether the processor was processing an instruction or not, and the classification displayed on the function code outputs when the bus error occurred. The processor is processing an instruction if it is in the normal state or processing a Group 2 exception; the processor is not processing an instruction if it is processing a Group 0 or a Group 1 exception. Figure 37 illustrates how this information is organized on the supervisor stack. Although this information is not sufficient in general to effect full recovery from the bus error, it does allow software diagnosis. Finally, the processor commences instruction processing at the address contained in the vector. It is the responsibility of the error handler routine to clean up the stack and determine where to continue execution.

If a bus error occurs during the exception processing for a bus error, address error, or reset, the processor is halted, and all processing ceases. This simplifies the detection of catastrophic system failure, since the processor removes itself from the system rather than destroy all memory contents. Only the RESET pin can restart a halted processor.

ADDRESS ERROR. Address error exceptions occur when the processor attempts to access a word or a long word operand or an instruction at an odd address. The effect is much like an internally generated bus error, so that the bus cycle is aborted, and the processor ceases whatever processing it is currently doing and begins exception processing. After exception processing commences, the sequence is the same as that for bus error including the information that is stacked, except that the vector number refers to the address error vector instead. Likewise, if an address error occurs during the exception processing for a bus error, address error, or reset, the processor is halted. As shown in Figure 38, an address error will execute a short bus cycle followed by exception processing.

INTERFACE WITH M6800 PERIPHERALS

Motorola's extensive line of M6800 peripherals are directly compatible with the MC68000. Some of these devices that are particularly useful are:

MC6821 Peripheral Interface Adapter
MC6840 Programmable Timer Module
MC6843 Floppy Disk Controller
MC6845 CRT Controller
MC6850 Asynchronous Communication Interface Adapter
MC6852 Synchronous Serial Data Adapter
MC6854 Advanced Data Link Controller
MC68488 General Purpose Interface Adapter

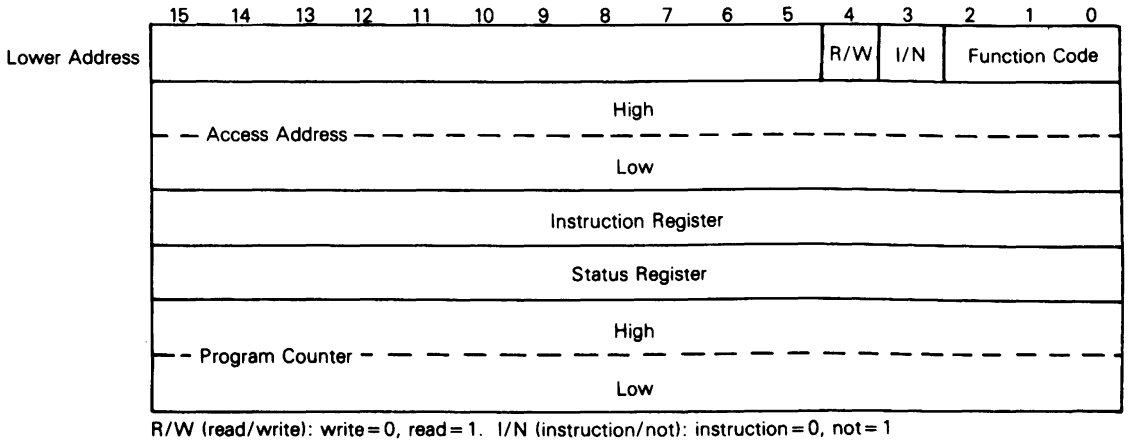
To interface the synchronous M6800 peripherals with the asynchronous MC68000, the processor modifies its bus cycle to meet the M6800 cycle requirements whenever an M6800 device address is detected. This is possible since both processors use memory mapped I/O. Figure 39 is a flow chart of the interface operation between the processor and M6800 devices.

DATA TRANSFER OPERATION

Three signals on the processor provide the M6800 interface. They are: enable (E), valid memory address ($\overline{VMA}$), and valid peripheral address ($\overline{VPA}$). Enable corresponds to the E or $\phi 2$ signal in existing M6800 systems. The bus frequency is one tenth of the incoming MC68000 clock frequency. The timing of E allows 1 MHz peripherals to be used with an 8 MHz MC68000. Enable has a 60/40 duty cycle; that is, it is low for six input clocks and high for four input clocks. This duty cycle allows the processor to do successive VPA accesses on successive E pulses.

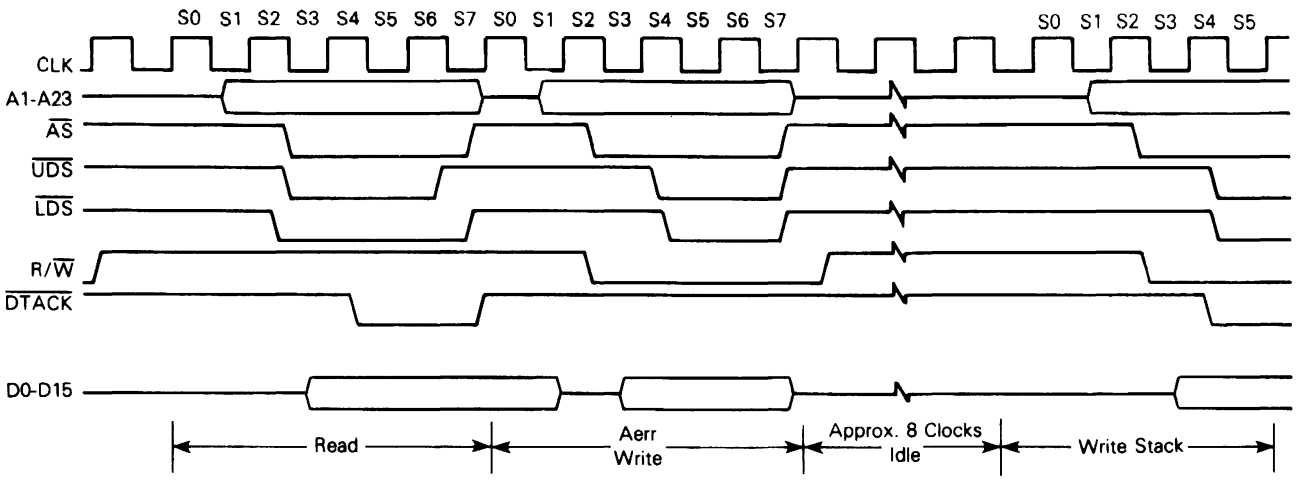
M6800 cycle timing is given in Figures 40 and 41. At state zero (S0) in the cycle, the address bus is in the high-impedance state. A function code is asserted on the function code output lines. One-half clock later, in state 1, the address bus is released from the high-impedance state.

FIGURE 37 – SUPERVISOR STACK ORDER (GROUP 0)



R/W (read/write): write = 0, read = 1. I/N (instruction/not): instruction = 0, not = 1

FIGURE 38 — ADDRESS ERROR TIMING



During state 2, the address strobe ($\overline{AS}$) is asserted to indicate that there is a valid address on the address bus. If the bus cycle is a read cycle, the upper and/or lower data strobes are also asserted in state 2. If the bus cycle is a write cycle, the read/write (R/$\overline{W}$) signal is switched to low (write) during state 2. One half clock later, in state 3, the write data is placed on the data bus, and in state 4 the data strobes are issued to indicate valid data on the data bus. The processor now inserts wait states until it recognizes the assertion of $\overline{VPA}$.

The $\overline{VPA}$ input signals the processor that the address on the bus is the address of an M6800 device (or an area reserved for M6800 devices) and that the bus should conform to the $\phi 2$ transfer characteristics of the M6800 bus. Valid peripheral address is derived by decoding the address bus, conditioned by address strobe.

After the recognition of $\overline{VPA}$, the processor assures that the Enable (E) is low, by waiting if necessary, and subsequently asserts $\overline{VMA}$. Valid memory address is then used as part of the chip select equation of the peripheral. This ensures that the M6800 peripherals are selected and deselected at the correct time. The peripheral now runs its cycle during the high portion of the E signal. Figures 40 and 41 depict the best and worst case M6800 cycle timing. This cycle length is dependent strictly upon when $\overline{VPA}$ is asserted in relationship to the E clock.

During a read cycle, the processor latches the peripheral data in state 6. For all cycles, the processor negates the address and data strobes one half clock cycle later in state 7, and the Enable signal goes low at this time. Another half clock later, the address bus is put in the high-impedance state. During a write cycle, the data bus is put in the high-impedance state and the read/write signal is switched high The peripheral logic must remove $\overline{VPA}$ within one clock after address strobe is negated.

$\overline{DTACK}$ should not be asserted while $\overline{VPA}$ is asserted. Notice that the MC68000 $\overline{VMA}$ is active low, contrasted with the active high M6800 VMA. This allows the processor to put its buses in the high-impedance state on DMA requests without inadvertently selecting peripherals.

FIGURE 39 — M6800 INTERFACING FLOW CHART

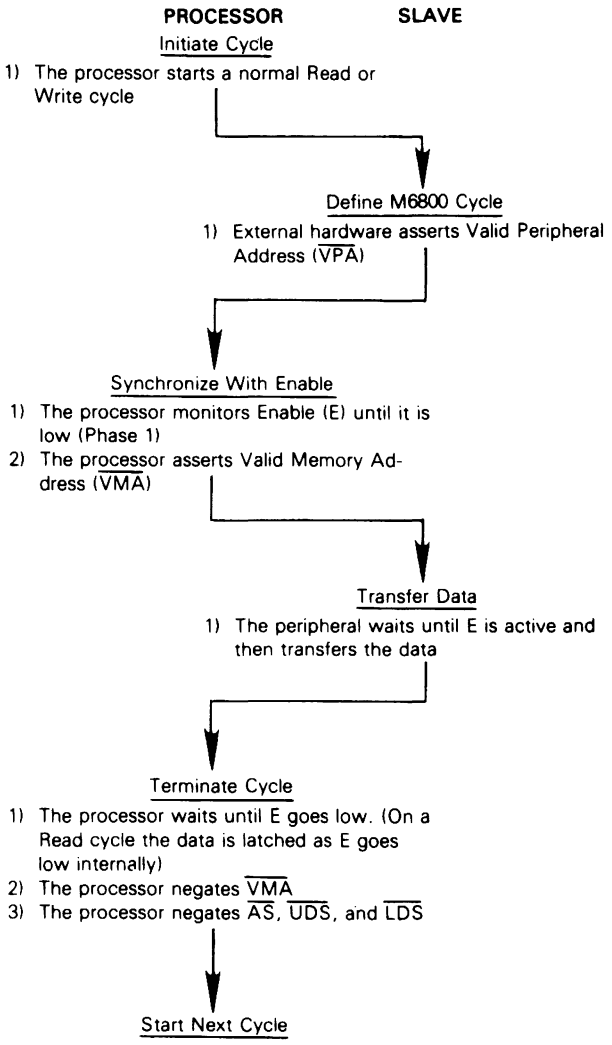
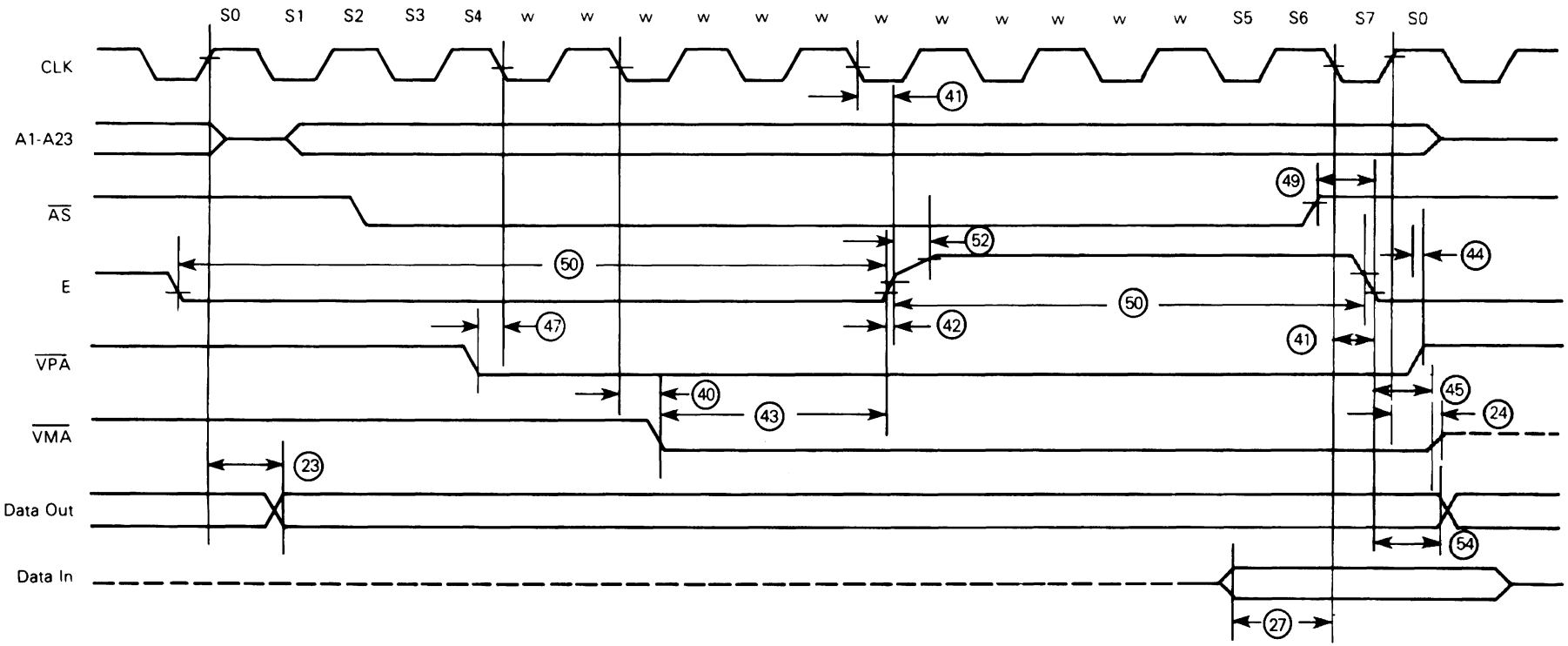
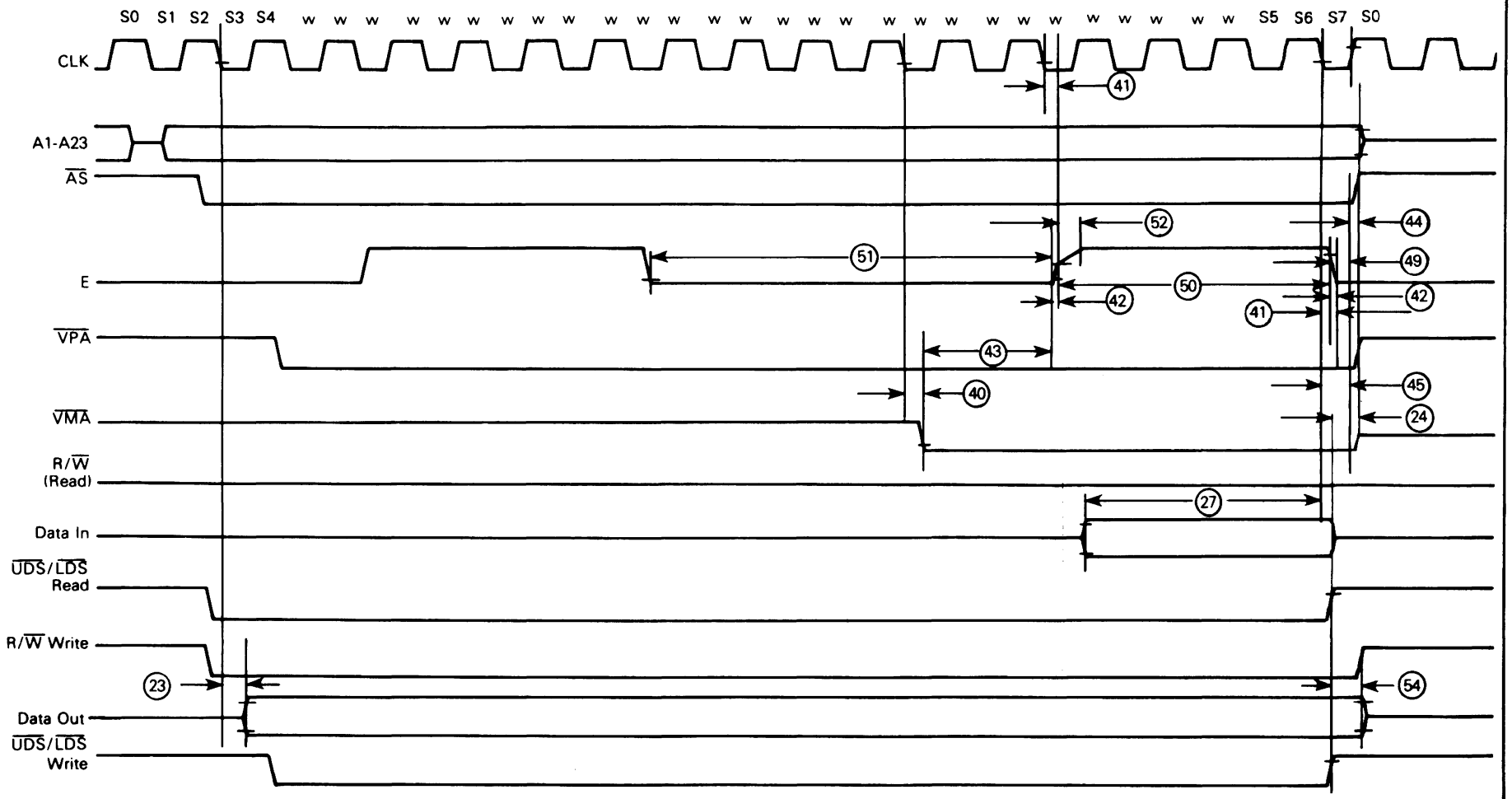


FIGURE 40 – M6800 TIMING – BEST CASE



NOTE: This figure represents the best case M6800 timing where $\overline{VPA}$ falls before the third system clock cycle after the falling edge of E.

FIGURE 41 — MC6800 TIMING — WORST CASE



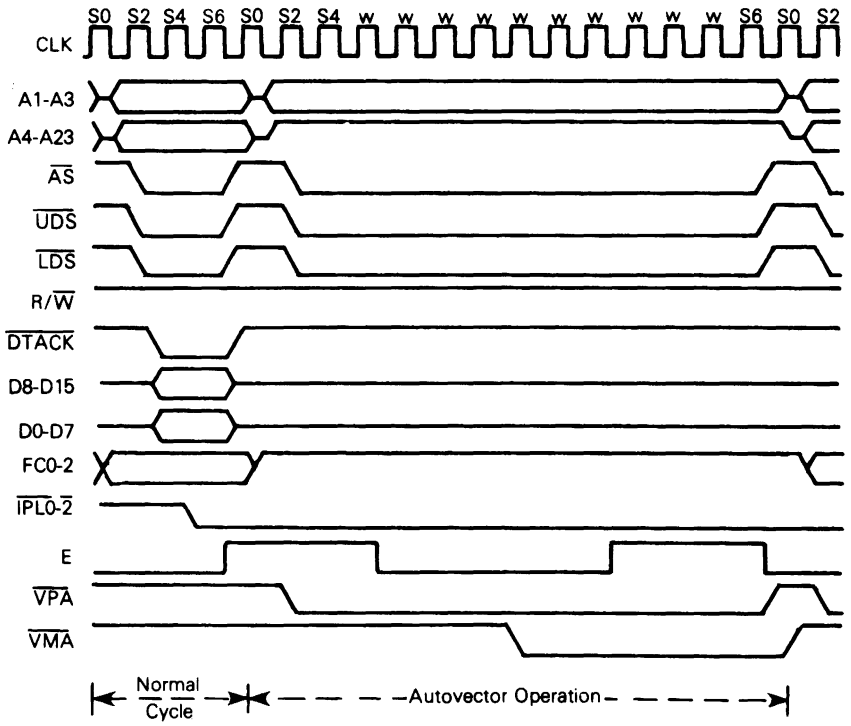
INTERRUPT OPERATION

During an interrupt acknowledge cycle while the processor is fetching the vector, if $\overline{VPA}$ is asserted, the MC68000 will assert $\overline{VMA}$ and complete a normal M6800 read cycle as shown in Figure 42. The processor will then use an internally generated vector that is a function of the interrupt being serviced. This process is known as autovectoring. The seven autovectors are vector numbers 25 through 31 (decimal).

This operates in the same fashion (but is not restricted to) the M6800 interrupt sequence. The basic difference is that there are six normal interrupt vectors and one NMI type vector. As with both the M6800 and the MC68000's normal vectored interrupt, the interrupt service routine can be located anywhere in the address space. This is due to the fact that while the vector numbers are fixed, the contents of the vector table entries are assigned by the user.

Since $\overline{VMA}$ is asserted during autovectoring, the M6800 peripheral address decoding should prevent unintended accesses.

FIGURE 42 — AUTOVECTOR OPERATION TIMING DIAGRAM



AC ELECTRICAL SPECIFICATIONS (V_{CC} = 5.0 Vdc ±5%, V_{SS} = 0 Vdc, T_A = 0°C to 70°C, refer to Figures 30 and 31)

Number	Characteristic	Symbol	4 MHz MC68000L4		6 MHz MC68000L6		8 MHz MC68000L8		10 MHz MC68000L10		Unit
			Min	Max	Min	Max	Min	Max	Min	Max	
24	Clock High to R/$\overline{W}$, $\overline{VMA}$ High Impedance	t_{CHRZ}	–	120	–	100	–	80	–	70	ns
40	Clock Low to $\overline{VMA}$ Low	t_{CLVML}	–	90	–	80	–	70	–	70	ns
41	Clock Low to E Transition	t_{CLC}	–	100	–	85	–	70	–	55	ns
42	E Output Rise and Fall Time	t_{Erf}	–	25	–	25	–	25	–	25	ns
43	$\overline{VMA}$ Low to E High	t_{VMLEH}	325	–	240	–	200	–	150	–	ns
44	$\overline{AS}$, $\overline{DS}$ High to VPA High	t_{SHVPH}	0	240	0	160	0	120	0	90	ns
45	E Low to Address/$\overline{VMA}$/FC Invalid	t_{ELAI}	55	–	35	–	30	–	10	–	ns
49	E Low to $\overline{AS}$, $\overline{DS}$ Invalid	t_{ELSI}	−80	–	−80	–	−80	–	−80	–	ns
50	E Width High	t_{EH}	900	–	600	–	450	–	350	–	ns
51	E Width Low	t_{EL}	1400	–	900	–	700	–	550	–	ns
52	E Extended Rise Time	t_{CIEHX}	80	–	80	–	80	–	80	–	ns
54	Data Hold from E Low (Write)	t_{ELDOZ}	60	–	40	–	30	–	20	–	ns
23	Clock Low to Data Out Valid	t_{CLDO}	–	90	–	80	–	70	–	55	ns
27	Data In to Clock Low (Setup Time)	t_{DICL}	30	–	25	–	15	–	15	–	ns
47	Asynchronous Input Setup Time	t_{AS1}	30	–	25	–	20	–	20	–	ns

DATA TYPES AND ADDRESSING MODES

Five basic data types are supported. These data types are:

- Bits
- BCD Digits (4-bits)
- Bytes (8-bits)
- Word (16-bits)
- Long Words (32-bits)

In addition, operations on other data types such as memory addresses, status word data, etc., are provided for in the instruction set.

The 14 addressing modes, shown in Table 9, include six basic types:

- Register Direct
- Register Indirect
- Absolute
- Immediate
- Program Counter Relative
- Implied

Included in the register indirect addressing modes is the capability to do postincrementing, predecrementing, offsetting and indexing. Program counter relative mode can also be modified via indexing and offsetting.

TABLE 9 — ADDRESSING MODES

Mode	Generation
Register Direct Addressing	
Data Register Direct	EA = Dn
Address Register Direct	EA = An
Absolute Data Addressing	
Absolute Short	EA = (Next Word)
Absolute Long	EA = (Next Two Words)
Program Counter Relative Addressing	
Relative with Offset	EA = (PC) + d_{16}
Relative with Index and Offset	EA = (PC) + (Xn) + d_8
Register Indirect Addressing	
Register Indirect	EA = (An)
Postincrement Register Indirect	EA = (An), An ← An + N
Predecrement Register Indirect	An ← An − N, EA = (An)
Register Indirect with Offset	EA = (An) + d_{16}
Indexed Register Indirect with Offset	EA = (An) + (Xn) + d_8
Immediate Data Addressing	
Immediate	DATA = Next Word(s)
Quick Immediate	Inherent Data
Implied Addressing	
Implied Register	EA = SR, USP, SP, PC

NOTES:

EA = Effective Address
An = Address Register
Dn = Data Register
Xn = Address or Data Register used as Index Register
SR = Status Register
PC = Program Counter
( ) = Contents of
d_8 = Eight-bit Offset (displacement)
d_{16} = Sixteen-bit Offset (displacement)
N = 1 for Byte, 2 for Words and 4 for Long Words
← = Replaces

INSTRUCTION SET OVERVIEW

The MC68000 instruction set is shown in Table 10. Some additional instructions are variations, or subsets, of these and they appear in Table 11. Special emphasis has been given to the instruction set's support of structured high-level languages to facilitate ease of programming. Each instruction, with few exceptions, operates on bytes, words, and long words and most instructions can use any of the 14 addressing modes. Combining instruction types, data types, and addressing modes, over 1000 useful instructions are provided. These instructions include signed and unsigned multiply and divide, "quick" arithmetic operations, BCD arithmetic and expanded operations (through traps).

TABLE 10 — INSTRUCTION SET

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
AND	Logical And
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
B_{CC}	Branch Conditionally
BCHG	Bit Test and Change
BCLR	Bit Test and Clear
BRA	Branch Always
BSET	Bit Test and Set
BSR	Branch to Subroutine
BTST	Bit Test
CHK	Check Register Against Bounds
CLR	Clear Operand
CMP	Compare
DB_{CC}	Test Condition, Decrement and Branch
DIVS	Signed Divide
DIVU	Unsigned Divide
EOR	Exclusive Or
EXG	Exchange Registers
EXT	Sign Extend
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link Stack
LSL	Logical Shift Left
LSR	Logical Shift Right
MOVE	Move
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral Data
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NOP	No Operation
NOT	One's Complement
OR	Logical Or
PEA	Push Effective Address
RESET	Reset External Devices
ROL	Rotate Left without Extend
ROR	Rotate Right without Extend
ROXL	Rotate Left with Extend
ROXR	Rotate Right with Extend
RTE	Return from Exception
RTR	Return and Restore
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
S_{CC}	Set Conditional
STOP	Stop
SUB	Subtract
SWAP	Swap Data Register Halves
TAS	Test and Set Operand
TRAP	Trap
TRAPV	Trap on Overflow
TST	Test
UNLK	Unlink

TABLE 11 — VARIATIONS OF INSTRUCTION TYPES

Instruction Type	Variation	Description
ADD	**ADD**	Add
	ADDA	Add Address
	ADDQ	Add Quick
	ADDI	Add Immediate
	ADDX	Add with Extend
AND	**AND**	Logical And
	ANDI	And Immediate
CMP	**CMP**	Compare
	CMPA	Compare Address
	CMPM	Compare Memory
	CMPI	Compare Immediate
EOR	**EOR**	Exclusive Or
	EORI	Exclusive Or Immediate
MOVE	**MOVE**	Move
	MOVEA	Move Address
	MOVEQ	Move Quick
	MOVE from SR	Move from Status Register
	MOVE to SR	Move to Status Register
	MOVE to CCR	Move to Condition Codes
	MOVE USP	Move User Stack Pointer
NEG	**NEG**	Negate
	NEGX	Negate with Extend
OR	**OR**	Logical Or
	ORI	Or Immediate'
SUB	**SUB**	Subtract
	SUBA	Subtract Address
	SUBI	Subtract Immediate
	SUBQ	Subtract Quick
	SUBX	Subtract with Extend

The following paragraphs contain an overview of the form and structure of the MC68000 instruction set. The instructions form a set of tools that include all the machine functions to perform the following operations:

Data Movement
Integer Arithmetic
Logical
Shift and Rotate
Bit Manipulation
Binary Coded Decimal
Program Control
System Control

The complete range of instruction capabilities combined with the flexible addressing modes described previously provide a very flexible base for program development.

ADDRESSING

Instructions for the MC68000 contain two kinds of information: the type of function to be performed, and the location of the operand(s) on which to perform that function. The methods used to locate (address) the operand(s) are explained in the following paragraphs.

Instructions specify an operand location in one of three ways:

Register Specification – the number of the register is given in the register field of the instruction.
Effective Address – use of the different effective address modes.
Implicit Reference – the definition of certain instructions implies the use of specific registers.

DATA MOVEMENT OPERATIONS

The basic method of data acquisition (transfer and storage) is provided by the move (MOVE) instruction. The move instruction and the effective addressing modes allow both address and data manipulation. Data move instructions allow byte, word, and long word operands to be transferred from memory to memory, memory to register, register to memory, and register to register. Address move instructions allow word and long word operand transfers and ensure that only legal address manipulations are executed. In addition to the general move instruction there are several special data movement instructions: move multiple registers (MOVEM), move peripheral data (MOVEP), exchange registers (EXG), load effective address (LEA), push effective address (PEA), link stack (LINK), unlink stack (UNLK), and move quick (MOVEQ). Table 12 is a summary of the data movement operations.

INTEGER ARITHMETIC OPERATIONS

The arithmetic operations include the four basic operations of add (ADD), subtract (SUB), multiply (MUL), and divide (DIV) as well as arithmetic compare (CMP), clear (CLR), and negate (NEG). The add and subtract instructions are available for both address and data operations, with data operations accepting all operand sizes. Address operations are limited to legal address size operands (16 or 32 bits). Data, address, and memory compare operations are also available. The clear and negate instructions may be used on all sizes of data operands.

The multiply and divide operations are available for signed and unsigned operands using word multiply to produce a long word product, and a long word dividend with word divisor to produce a word quotient with a word remainder.

Multiprecision and mixed size arithmetic can be accomplished using a set of extended instructions. These instructions are: add extended (ADDX), subtract extended (SUBX), sign extend (EXT), and negate binary with extend (NEGX).

A test operand (TST) instruction that will set the condition codes as a result of a compare of the operand with zero is also available. Test and set (TAS) is a synchronization instruction useful in multiprocessor systems. Table 13 is a summary of the integer arithmetic operations.

TABLE 12 – DATA MOVEMENT OPERATIONS

Instruction	Operand Size	Operation
EXG	32	Rx ↔ Ry
LEA	32	EA → An
LINK	–	An → SP@ – SP → An SP + d → SP
MOVE	8, 16, 32	(EA)s → EAd
MOVEM	16, 32	(EA) → An, Dn An, Dn → EA
MOVEP	16, 32	(EA) → Dn Dn → EA
MOVEQ	8	#xxx → Dn
PEA	32	EA → SP@ –
SWAP	32	Dn[31:16] ↔ Dn[15:0]
UNLK	–	An → Sp SP@ + → An

NOTES:
s = source
d = destination
[ ] = bit numbers
@ – = indirect with predecrement
@ + = indirect with postdecrement

INSTRUCTION FORMAT

Instructions are from one to five words in length, as shown in Figure 43. The length of the instruction and the operation to be performed is specified by the first word of the instruction which is called the operation word. The remaining words further specify the operands. These words are either immediate operands or extensions to the effective address mode specified in the operation word.

PROGRAM/DATA REFERENCES

The MC68000 separates memory references into two classes: program references, and data references. Program references, as the name implies, are references to that section of memory that contains the program being executed. Data references refer to that section of memory that contains data. Generally, operand reads are from the data space. All operand writes are to the data space.

REGISTER SPECIFICATION

The register field within an instruction specifies the register to be used. Other fields within the instruction specify whether the register selected is an address or data register and how the register is to be used.

TABLE 13 — INTEGER ARITHMETIC OPERATIONS

Instruction	Operand Size	Operation
ADD	8, 16, 32	Dn + (EA) → Dn
		(EA) + Dn → EA
		(EA) + #xxx → EA
	16, 32	An + (EA) → An
ADDX	8, 16, 32	Dx + Dy + X → Dx
	16, 32	Ax@ – Ay@ – + X → Ax@
CLR	8, 16, 32	0 → EA
CMP	8, 16, 32	Dn – (EA)
		(EA) – #xxx
		Ax@ + – Ay@ +
	16, 32	An – (EA)
DIVS	32 ÷ 16	Dn/(EA) → Dn
DIVU	32 ÷ 16	Dn/(EA) → Dn
EXT	8 → 16	$(Dn)_8 \rightarrow Dn_{16}$
	16 → 32	$(Dn)_{16} \rightarrow Dn_{32}$
MULS	16*16 → 32	Dn*(EA) → Dn
MULU	16*16 → 32	Dn*(EA) → Dn
NEG	8, 16, 32	0 – (EA) → EA
NEGX	8, 16, 32	0 – (EA) – X – EA
SUB	8, 16, 32	Dn – (EA) → Dn
		(EA) – Dn → EA
		(EA) – #xxx → EA
	16, 32	An – (EA) → An
SUBX	8, 16, 32	Dx – Dy – X → Dx
		Ax@ – – Ay@ – – X → Ax@
TAS	8	(EA) – 0, 1 → EA[7]
TST	8, 16, 32	(EA) – 0

NOTE: [ ] = bit number

EFFECTIVE ADDRESS

Most instructions specify the location of an operand by using the effective address field in the operation word. For example, Figure 44 shows the general format of the single effective address instruction operation word. The effective address is composed of two 3-bit fields: the mode field, and the register field. The value in the mode field selects the different address modes. The register field contains the number of a register.

The effective address field may require additional information to fully specify the operand. This additional information, called the effective address extension, is contained in the following word or words and is considered part of the instruction, as shown in Figure 43. The effective address modes are grouped into three categories: register direct, memory addressing, and special.

REGISTER DIRECT MODES. These effective addressing modes specify that the operand is in one of the 16 multifunction registers.

Data Register Direct. The operand is in the data register specified by the effective address register field.

Address Register Direct. The operand is in the address register specified by the effective address register field.

MEMORY ADDRESS MODES. These effective addressing modes specify that the operand is in memory and provide the specific address of the operand.

Address Register Indirect. The address of the operand is in the address register specified by the register field. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

FIGURE 43 — INSTRUCTION FORMAT

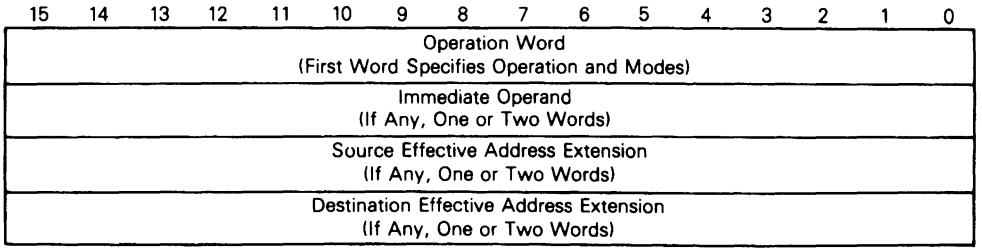
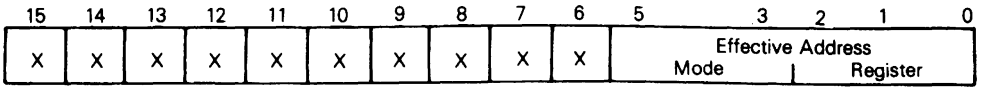


FIGURE 44 — SINGLE-EFFECTIVE-ADDRESS INSTRUCTION OPERATION WORD GENERAL FORMAT



Address Register Indirect With Postincrement. The address of the operand is in the address register specified by the register field. After the operand address is used, it is incremented by one, two, or four depending upon whether the size of the operand is byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is incremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.

Address Register Indirect With Predecrement. The address of the operand is in the address register specified by the register field. Before the operand address is used, it is decremented by one, two, or four depending upon whether the operand size is byte, word, or long word. If the address register is the stack pointer and the operand size is byte, the address is decremented by two rather than one to keep the stack pointer on a word boundary. The reference is classified as a data reference.

Address Register Indirect With Displacement. This address mode requires one word of extension. The address of the operand is the sum of the address in the address register and the sign-extended 16-bit displacement integer in the extension word. The reference is classified as a data reference with the exception of the jump to subroutine instructions.

Address Register Indirect With Index. This address mode requires one word of extension. The address of the operand is the sum of the address in the address register, the sign-extended displacement integer in the low order eight bits of the extension word, and the contents of the index register. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

SPECIAL ADDRESS MODE. The special address modes use the effective address register field to specify the special addressing mode instead of a register number.

Absolute Short Address. This address mode requires one word of extension. The address of the operand is the extension word. The 16-bit address is sign extended before it is used. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

Absolute Long Address. This address mode requires two words of extension. The address of the operand is developed by the concatenation of the extension words. The high-order part of the address is the first extension word; the low-order part of the address is the second extension word. The reference is classified as a data reference with the exception of the jump and jump to subroutine instructions.

Program Counter With Displacement. This address mode requires one word of extension. The address of the operand is the sum of the address in the program counter and the sign-extended 16-bit displacement integer in the extension word. The value in the program counter is the address of the extension word. The reference is classified as a program reference.

Program Counter With Index. This address mode requires one word of extension. This address is the sum of the address in the program counter, the sign-extended displacement integer in the lower eight bits of the extension word, and the contents of the index register. The value in the program counter is the address of the extension word. This reference is classified as a program reference.

Immediate Data. This address mode requires either one or two words of extension depending on the size of the operation.

Byte operation – operand is low order byte of extension word

Word operation – operand is extension word

Long word operation – operand is in the two extension words, high-order 16 bits are in the first extension word, low-order 16 bits are in the second extension word.

Condition Codes or Status Register. A selected set of instructions may reference the status register by means of the effective address field. These are:

ANDI to CCR
ANDI to SR
EORI to CCR
EORI to SR
ORI to CCR
ORI to SR

EFFECTIVE ADDRESS ENCODING SUMMARY

Table 14 is a summary of the effective addressing modes discussed in the previous paragraphs.

TABLE 14 – EFFECTIVE ADDRESS ENCODING SUMMARY

Addressing Mode	Mode	Register
Data Register Direct	000	register number
Address Register Direct	001	register number
Address Register Indirect	010	register number
Address Register Indirect with Postincrement	011	register number
Address Register Indirect with Predecrement	100	register number
Address Register Indirect with Displacement	101	register number
Address Register Indirect with Index	110	register number
Absolute Short	111	000
Absolute Long	111	001
Program Counter with Displacement	111	010
Program Counter with Index	111	011
Immediate	111	100

IMPLICIT REFERENCE

Some instructions make implicit reference to the program counter (PC), the system stack pointer (SP), the supervisor stack pointer (SSP), the user stack pointer (USP), or the status register (SR).

SYSTEM STACK. The system stack is used implicitly by many instructions; user stacks and queues may be created and maintained through the addressing modes. Address register seven (A7) is the system stack pointer (SP). The system stack pointer is either the supervisor stack pointer (SSP) or the user stack pointer (USP), depending on the state of the S-bit in the status register. If the S-bit indicates supervisor state, SSP is the active system stack pointer, and the USP cannot be referenced as an address register. If the S-bit indicates user state, the USP is the active system stack pointer, and the SSP cannot be referenced. Each system stack fills from high memory to low memory.

LOGICAL OPERATIONS

Logical operation instructions AND, OR, EOR, and NOT are available for all sizes of integer data operands. A similar set of immediate instructions (ANDI, ORI, and EORI) provide these logical operations with all sizes of immediate data. Table 15 is a summary of the logical operations.

TABLE 15 — LOGICAL OPERATIONS

Instruction	Operand Size	Operation
AND	8, 16, 32	Dn∧(EA) → Dn (EA)∧Dn → EA (EA)∧#xxx → EA
OR	8, 16, 32	Dn v (EA) → Dn (EA) v Dn → EA (EA) v #xxx → EA
EOR	8, 16, 32	(EA) • Dy → EA (EA) • #xxx → EA
NOT	8, 16, 32	~(EA) → EA

NOTE: ~ = invert

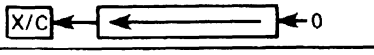
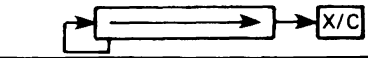
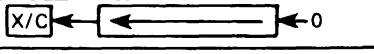
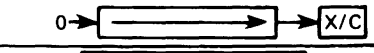
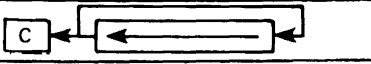
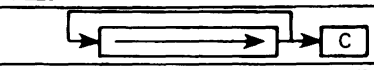
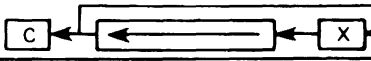
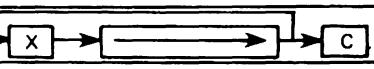
SHIFT AND ROTATE OPERATIONS

Shift operations in both directions are provided by the arithmetic instructions ASR and ASL and logical shift instructions LSR and LSL. The rotate instructions (with and without extend) available are ROXR, ROXL, ROR, and ROL. All shift and rotate operations can be performed in either registers or memory. Register shifts and rotates support all operand sizes and allow a shift count specified in the instruction of one to eight bits, or 0 to 63 specified in a data register.

Memory shifts and rotates are for word operands only and allow only single-bit shifts or rotates.

Table 16 is a summary of the shift and rotate operations.

TABLE 16 — SHIFT AND ROTATE OPERATIONS

Instruction	Operand Size	Operation
ASL	8, 16, 32	
ASR	8, 16, 32	
LSL	8, 16, 32	
LSR	8, 16, 32	
ROL	8, 16, 32	
ROR	8, 16, 32	
ROXL	8, 16, 32	
ROXR	8, 16, 32	

BIT MANIPULATION OPERATIONS

Bit manipulation operations are accomplished using the following instructions: bit test (BTST), bit test and set (BSET), bit test and clear (BCLR), and bit test and change (BCHG). Table 17 is a summary of the bit manipulation operations. (Bit 2 of the status register is Z.)

TABLE 17 — BIT MANIPULATION OPERATIONS

Instruction	Operand Size	Operation
BTST	8, 32	~bit of (EA) → Z
BSET	8, 32	~bit of (EA) → Z 1 → bit of EA
BCLR	8, 32	~bit of (EA) → Z 0 → bit of EA
BCHG	8, 32	~bit of (EA) → Z ~bit of (EA) → bit of EA

BINARY CODED DECIMAL OPERATIONS

Multiprecision arithmetic operations on binary coded decimal numbers are accomplished using the following instructions: add decimal with extend (ABCD), subtract decimal with extend (SBCD), and negate decimal with extend (NBCD). Table 18 is a summary of the binary coded decimal operations.

TABLE 18 — BINARY CODED DECIMAL OPERATIONS

Instruction	Operand Size	Operation
ABCD	8	$Dx_{10} + Dy_{10} + X \rightarrow Dx$ $Ax@_{-10} + Ay@_{-10} + X \rightarrow Ax@$
SBCD	8	$Dx_{10} - Dy_{10} - X \rightarrow Dx$ $Ax@_{-10} - Ay@_{-10} - X \rightarrow Ax@$
NBCD	8	$0 - (EA)_{10} - X \rightarrow EA$

PROGRAM CONTROL OPERATIONS

Program control operations are accomplished using a series of conditional and unconditional branch instructions and return instructions. These instructions are summarized in Table 19.

The conditional instructions provide setting and branching for the following conditions:

CC	— carry clear	LS	— low or same
CS	— carry set	LT	— less than
EQ	— equal	MI	— minus
F	— never true	NE	— not equal
GE	— greater or equal	PL	— plus
GT	— greater than	T	— always true
HI	— high	VC	— no overflow
LE	— less or equal	VS	— overflow

TABLE 19 — PROGRAM CONTROL OPERATIONS

Instruction	Operation
Conditional	
B_{CC}	Branch conditionally (14 conditions) 8- and 16-bit displacement
DB_{CC}	Test condition, decrement, and branch 16-bit displacement
S_{CC}	Set byte conditionally (16 conditions)
Unconditional	
BRA	Branch always 8- and 16-bit displacement
BSR	Branch to subroutine 8- and 16-bit displacement
JMP	Jump
JSR	Jump to subroutine
Returns	
RTR	Return and restore condition codes
RTS	Return from subroutine

SYSTEM CONTROL OPERATIONS

System control operations are accomplished by using privileged instructions, trap generating instructions, and instructions that use or modify the status register. These instructions are summarized in Table 20.

TABLE 20 — SYSTEM CONTROL OPERATIONS

Instruction	Operation
Privileged	
RESET	Reset external devices
RTE	Return from exception
STOP	Stop program execution
ORI to SR	Logical OR to status register
MOVE USP	Move user stack pointer
ANDI to SR	Logical AND to status register
EORI to SR	Logical EOR to status register
MOVE EA to SR	Load new status register
Trap Generating	
TRAP	Trap
TRAPV	Trap on overflow
CHK	Check register against bounds
Status Register	
ANDI to CCR	Logical AND to condition codes
EORI to CCR	Logical EOR to condition codes
MOVE EA to CCR	Load new condition codes
ORI to CCR	Logical OR to condition codes
MOVE SR to EA	Store status register