MC5600/5700 SYSTEM REFERENCE MANUAL

Order Number: M-SRM-567
Part Number: 075-04020-00-0

Revision A

Concurrent Computer Corporation

MC5600/5700 SYSTEM REFERENCE MANUAL

Order Number: M-SRM-567
Part Number: 075-04020-00-0

Revision A

Written By Stephen Gilbane

Revision A, September 1983
Revision B, March 1984

The postpaid READER'S COMMENTS form on the last page of this document requests your critical evaluation to assist in preparing future documentation.

Illustrated by Stephen Gilbane

Edited by Wesley Frost

CONTENTS

Chapter 8 Physical Address Structure

Chapter 9 The MULTIBUS

Chapter 10 The MULTIBUS Adaptor

Chapter 11 System Configuration

Chapter 12 The EPROM Bootstrap

Chapter 13 The Console

ILLUSTRATIONS

TABLES

Table No. **Page**

Preface

This manual describes the MASSCOMP MC5600 and MC5700 high performance 32-bit computer systems. These two systems are, in general, similar enough to be described as a single machine, and throughout this book, the term **MC5600/5700** is used to denote either system.

The MC5600/5700 is capable of high speed data acquisition, data analysis, graphics presentation, and general timesharing functions. The system integrates the processing power of up to 2 (in the MC5600) or 4 (in the MC5700) processor modules, each based on the MC68020 32-bit processor. The Triple Bus architecture allows for efficient delegation of processing tasks, as well as flexible configuration capabilities.

What This Manual Covers

This manual contains the necessary information to allow you to understand the internal hardware of the MC5600/5700. The manual assumes previous knowledge of general system hardware terms and concepts (such as bus, CPU, protocol, and the like) but otherwise assumes no specific knowledge of MASSCOMP products. The manual attempts to give a programmer's model of the hardware. It is designed to help anyone interfacing new hardware devices, writing operating systems or device drivers, maintaining system hardware, or performing any other activity that requires a detailed understanding of the machine's internal structure.

The manual consists of the following chapters and topics:

Chapter 1. System Overview. This chapter discusses the overall 5600/5700 System and MASSCOMP-specific system concepts. General system architecture illustrations are given. The processor, memory, and Auxiliary Function modules are discussed in the context of the system structure.

Chapter 2. Memory Management. This chapter discusses the memory management architecture, the high-speed address translation buffer, the virtual and secondary address spaces, paging algorithms, and page table bit field definitions.

Chapter 3. The Cache. This chapter discusses the operating modes and the flush and control registers of the cache. The invalidation logic is also discussed.

Chapter 4. Local CPU Devices. This chapter discusses devices local to each processor module: the 2 DUARTs and their serial communication ports, the PCRA & PCRB, and the CMPU write buffer.

Chapter 5. Interrupts and Exceptions. This chapter discusses how the CMPU hardware handles interrupts. It lists the assigned interrupt vectors for each MASSCOMP device, and the causes and responses to interrupts and bus errors. The system initialization circuitry on the AFM also is described in this chapter.

Chapter 6. The Memory Module. This chapter discusses the memory part of the CPU/Memory Subsystem. This includes memory exceptions and interrupts, error checking and handling, initialization information, and CSR bit assignments.

Chapter 7. The MASSCOMP Synchronous Memory Interconnect (SMI). This chapter discusses the high speed MASSCOMP SMI, the central bus linking all processors, memories and MULTIBUS Adaptors. The chapter discusses its role in the system, its capabilities, its arbitration, and, briefly, its protocol. A bus specification is not given in this document.

Chapter 8. Physical Address Space. This chapter gives the address range assignments within the SMI physical space for all devices in the system.

Chapter 9. The MULTIBUS. This chapter discusses the MC5600/5700 MULTIBUS and the differences between it and the standard MULTIBUS. The MASSCOMP block mode protocol is described in detail.

Chapter 10. The MULTIBUS Adaptor. This chapter discusses the MBA and its role in allowing communication between the SMI and the MULTIBUS. It describes the I/O Map, interbus byte swapping, and error handling.

Chapter 11. System Configuration. This chapter shows how to configure the CMPU, CMM, AFM, and backplane modules to change a basic system configuration. The chapter describes how to add multiple CPUs, additional memory modules, and multiple MULTIBUSs.

Chapter 12. The EPROM Bootstrap. This chapter discusses the MASSCOMP bootstrap code, powerup hardware initialization, diagnostic testing, and reading the boot block during powerup. It also includes the descriptions and physical addresses of the entry points of callable subroutines. The communication protocol between the CMPU and AFM is described to allow customers to access the NVRAM.

Chapter 13. The Console. This chapter discusses the console command set, syntax, and machine environment, with examples of the console commands included.

Appendix A. Pinouts. This appendix lists the pinouts for cables and primary module connectors in the system.

Appendix B. Specifications. This appendix provides the mechanical, electrical, environment, and expected performance specifications for all packages.

Appendix C. DUART Specification This appendix reproduces the specification sheet for the DUART chip, which has various programmable features.

Index.

Related Information

- The *System Management Guide* supplied with your system explains the range of system managerial responsibilities.
- The *Diagnostic Monitor User's Manual* describes how to use the Diagnostic Monitor program with your system.
- The *MC68020 32 Bit Microprocessor User's Manual* published by Motorola explains how to program the microprocessor used on the MC5600/5700 processor module.
- The *Bipolar Microprocessor Logic and Interface Data Book* describes the operation of the EDC chip on the CMM module. It is available upon request from Advanced Micro Devices, 901 Thompson Place, P.O. Box 3453, Sunnyvale, CA

94088 TEL:(408)732-2400.

- The *Motorola MC146818A Specification Sheet* describes the operation of the Time of Day chip used on the AFM. It is available upon request from Motorola Semiconductor Products, 3501 Ed Bluestein Blvd., Austin, TX 78721.

A complete listing of manuals available from MASSCOMP is contained in the *Guide to MASSCOMP Documentation.* For a copy of this document, see your MASSCOMP sales representative.

Conventions Used in this Manual

Fonts and Characters

Italics are for file and directory names, command names, and variable parameters in command lines. The parameters are described in the text following the command lines.

`Constant Width`
is for computer generated output, constant fields in command lines, and programming examples.

`Constant Width Bold`
is for literal user input.

Bold is used within the text for referring to command options or arguments, or for introducing new terms.

Terms

System refers to a computer system and is equivalent to *computer* or *host.* It does not refer to a specific MASSCOMP computer.

Programmer is a person writing low-level programs, such as operating system or device driver code.

System Manager
is the person with overall responsibility for the installation and maintenance of the MC5600/5700 at a particular installation.

User is a person who uses the MASSCOMP-supplied software applications only and does not perform system management.

Data sizes

Nibble	4 bits.
Byte	8 bits.
Word	16 bits.
Longword	32 bits.
Quadword	64 bits.

All hexadecimal numbers are written using the standard C prefix **0x**. The hexadecimal number F70000 is referred to as `0xF70000`.

Chapter 1
System Overview

ILLUSTRATIONS

Chapter 1
System Overview

This chapter gives an overview of the MC5600/5700 architecture. Most elements of the system are only briefly outlined and are developed in detail in later chapters that are referenced where appropriate. This chapter serves as both an introduction to the machine and as an index to topics covered in later chapters.

1.1 Introduction

The MC5600/5700 system is a high performance computer designed around the 32-bit 68020 microprocessor architecture. It is designed with multiprocessing capabilities that distribute tasks using symmetric processing techniques.

The system achieves high I/O throughput by using multiple I/O buses. The proprietary high-bandwidth system memory bus also facilitates this high throughput, as well as supporting tightly-coupled multiprocessing.

The CPU module architecture extracts the maximum performance from the 68020 by implementing various performance enhancements, such as a data cache, memory management hardware, and a write buffer between the processor and memory bus. The system also has integrated floating point arithmetic capability through the use of a floating point coprocessor (MC68881) or floating point accelerator module.

Finally, the system packages are flexible enough to allow peripheral expansion that is commensurate with the processing power. Systems may be configured to fit a wide variety of applications and processing requirements.

Figure 1-1 shows a typical MC5600 package (pedestal shown).

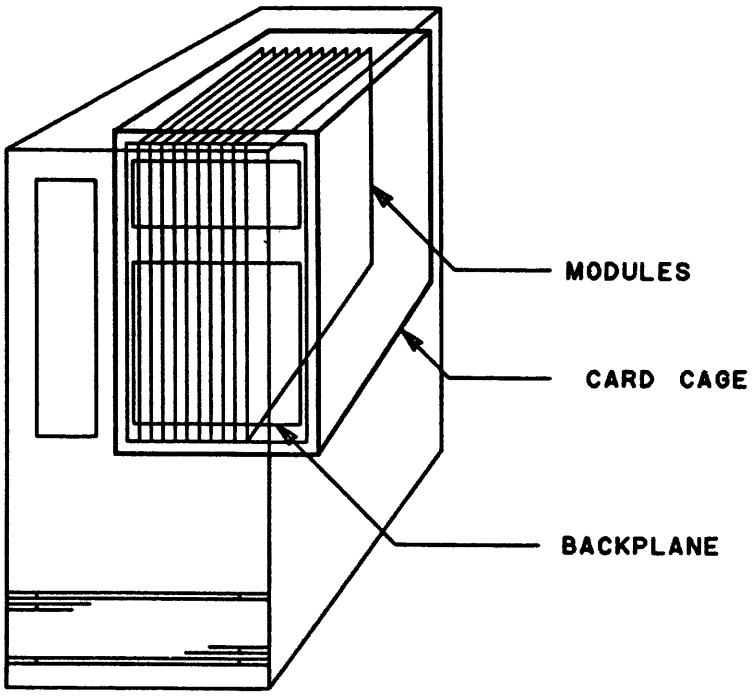


Figure 1-1. An MC5600 Pedestal Package

The heart of the system is the group of printed circuit boards, or modules, that slide into the system's card cage. The back of the card cage is called the **backplane**, which is itself a printed circuit board holding several of the system buses. All the modules communicate information over these buses on the backplane.

MC5600 systems can be housed in one of three types of packages (pedestal, tabletop, and rack mount). Each MC5600 package uses one card cage containing two or three system buses.

MC5700 systems are housed in one type of package (wide cabinet) and typically use two card cages, forming a backplane with between three and five system buses. Although understanding a large MC5700 configuration can appear prohibitively complex, its architectural building blocks are the same as those in the smaller MC5600 systems. The only difference between the MC5600 and MC5700 is the expandability of the package.

This chapter describes the major elements of a single processor MC5600/5700 system first. Section 1.6 describes the elements that are used to expand a system to a multiple bus and/or multiple processor system.

Figure 1-2 shows the logical block diagram for a typical (single processor) configuration for an MC5600/5700 system.

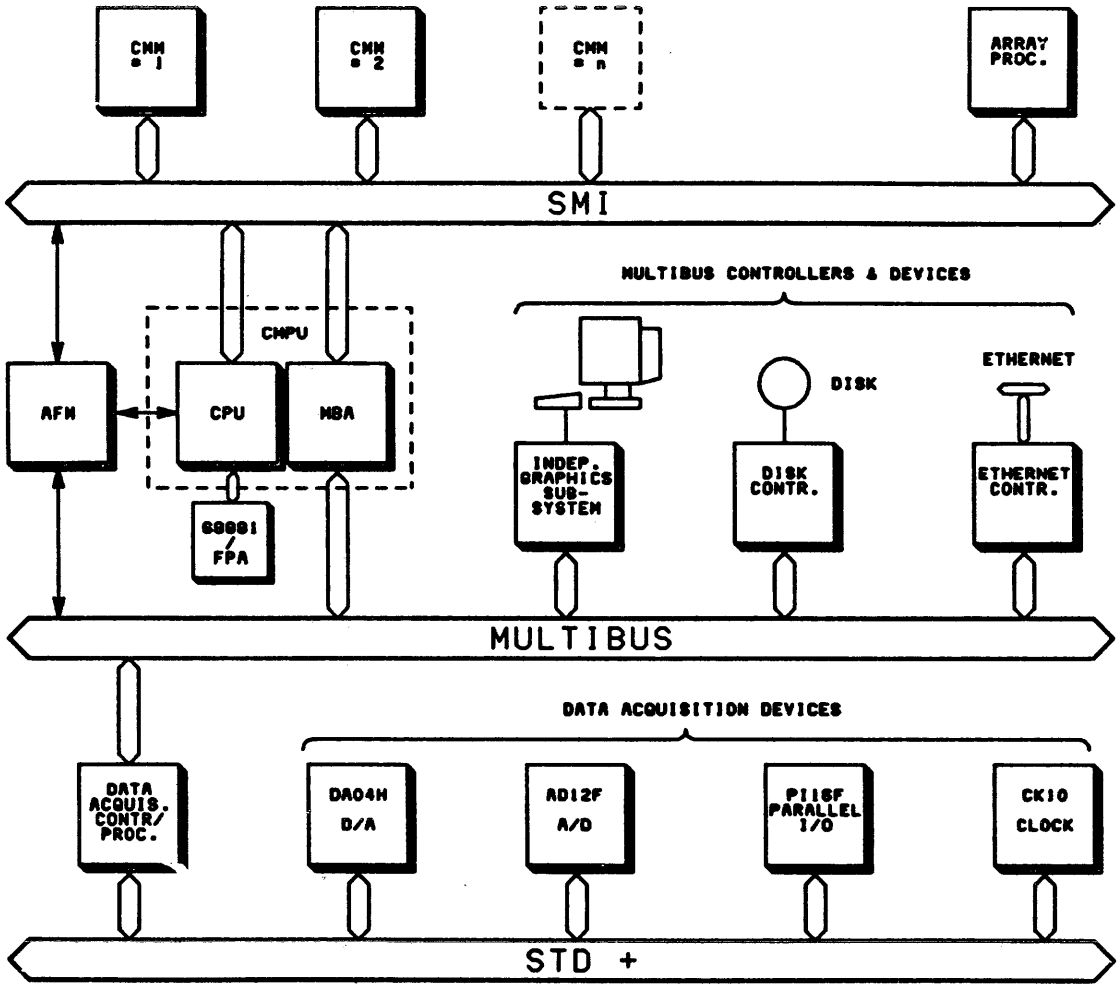


Figure 1-2. The MC5600/5700 System

The MC5600/5700 system architecture is made up of the following elements:

- Three buses (SMI, MULTIBUS, and STD+)
- One central processor module (CMPU), including a MULTIBUS Adaptor (MBA)
- One or more memory modules (CMM)
- An Auxiliary Function module (AFM)
- MULTIBUS controller modules (ETHERNET, disk, graphics)
- Data Acquisition / Control Processor (DACP)
- STD+ Data Acquisition Devices (DA04H, AD12F, and so on)

The following sections describe each of these system elements. The overall MC5600/5700 architecture is designed around the three buses, so they are described first.

1.2 System Buses

One of the central design elements of a computer system is its bus structure. A **bus** is defined as a collection of interface signals that allow the system's hardware elements to interact with each other. This interaction includes memory and Input/Output (I/O) data transfers, direct memory access (DMA) transfers, generation of interrupts, and so forth.

Most bus structures are built on the **master-slave** concept, where a master device in the system initiates bus activity by taking control of the bus and broadcasting the address of another device on the bus. The device that detects its own address on the bus becomes the slave and acts upon the command supplied by the master. This master-slave relationship (or **handshake**) allows modules of different speeds to interface over the same bus. Different buses implement this master-slave concept in various ways, each using a protocol that strictly defines the activity between master and slave.

The **physical address space** of a bus refers to the range of logical address values a given bus can support. The space actually supported by physical memory devices on the bus may be smaller than the physical address space of the system. Many buses support two independent, unique address spaces: **memory space** and **I/O space**. A bus often has separate control signals to select each type of space. Memory space is generally used for devices with large arrays of contiguous memory locations. I/O space is usually reserved for ports, control and status registers, and other miscellaneous devices.

The MC5600/5700 system uses three different, independent buses, with various conduits that facilitate inter-bus communication. These buses are:

- **Synchronous Memory Interconnect (SMI)** - This MASSCOMP proprietary bus is used to link all central processors with main system memory devices over a dedicated high-speed path. The bus is designed to support a multiprocessing environment, with physical memory uniformly accessible to all processors. The bus uses a proprietary split transaction protocol that frees up the use of the bus during memory access time. The SMI is described in Chapter 7.
- **MASSCOMP MULTIBUS** - This is an industry standard bus with several MASSCOMP enhancements. This bus is used as the entry channel into the system for intelligent I/O controllers handling graphics, mass storage, communication protocol and also Data Acquisition with the MASSCOMP Data Acquisition Controller Processor (DACP). One major enhancement is a 32-bit data transfer mode, added as a superset of the standard MULTIBUS protocol. The MASSCOMP MULTIBUS is described in Chapter 9.
- **STD +** - This is another industry standard bus enhanced by MASSCOMP. This bus is used on the MC5600/5700 exclusively for MASSCOMP data acquisition devices such as digital-to-analog, analog-to-digital, and parallel I/O modules. The DACP acts as both the STD+ bus master and the interface device between the STD+ bus and the MULTIBUS. Note that the STD+ bus uses a separate backplane not shown in Figure 1-1. The MASSCOMP STD+ bus is fully described in the *DACP System Programming Manual.*

These three buses are integrated into the Triple Bus Architecture of the system by means of two dedicated MASSCOMP devices that serve as inter-bus conduits: the MULTIBUS Adaptor handles communication between the SMI and the MULTIBUS, and the Data Acquisition Controller Processor handles communication between the STD+ and the MULTIBUS.

1.3 CMPU Module

The MC5600/5700 **Multi-Processing Unit (CMPU)** Module holds the central processing resources for the system. As shown in Figure 1-2, this module communicates directly both with memory devices on the SMI and with peripheral controllers on the MULTIBUS. The module contains both the central processor circuitry, with its associated performance enhancements, and the MULTIBUS Adaptor circuitry. The MULTIBUS Adaptor (MBA) acts as a separate SMI device through which all MULTIBUS communication passes. The system architecture allows multiple CMPU modules to operate simultaneously, sharing the SMI bus and its memory devices as well as the use of any MULTIBUS Adaptor on the system.

Figure 1-3 shows the functional block diagram of a CMPU module.

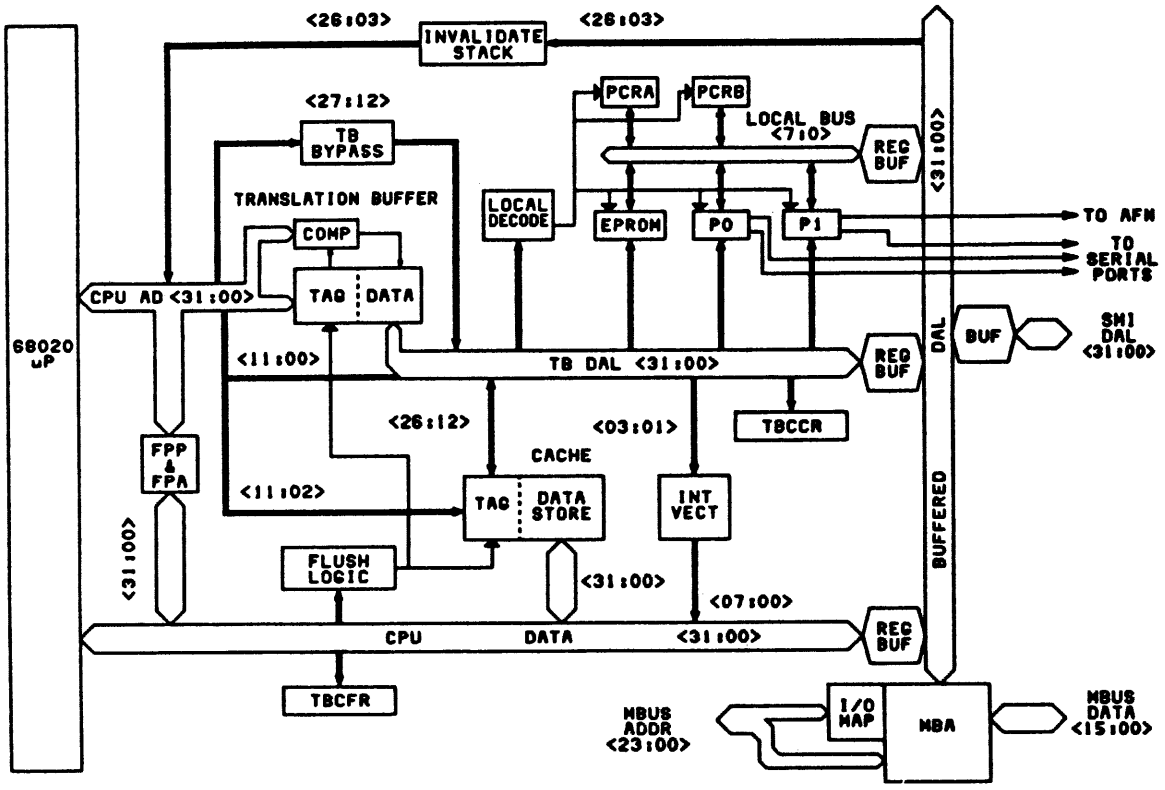


Figure 1-3. The CMPU Module

Each part of the CMPU is described in the following section. Figure 1-3 is referenced throughout this section.

1.3.1 68020 Microprocessor

The **68020 microprocessor** uses a non-multiplexed bus with 32 bits of address (CPU AD <31:00>) and 32 bits of data (CPU DATA <31:00>). It uses 7 levels of interrupts, and, in the MC5600/5700, can vector to a maximum of 55 unique interrupt or trap handlers. The processor operates in one of two levels of privilege: user mode or supervisor mode. User mode uses a subset of the supervisor's instruction set, and the two modes use two different stack pointers during operand references. The privilege level is used to control access of user programs to supervisor-only (usually operating system) areas in memory.

The 68020 is fully described in the Motorola *MC68020 32-bit Microprocessor User's Manual*, referenced in the preface of this manual. This manual references the Motorola text throughout when dealing with the details of 68020 operation.

1.3.2 Memory Management

Memory management hardware is provided to support the virtual memory operating system and manage the 4 GByte virtual address space. All 32-bit virtual addresses generated by the 68020 are first translated into 28-bit physical addresses used by the devices on the SMI. The memory management hardware constructs these translations using a fixed algorithm and 2 levels of **page tables** that are set up in main memory by the operating system. Once created, these address translations are loaded into the **Translation Buffer** (TB) on a demand basis. A part of the virtual address indexes into the TB, and this indexed location stores the translation along with an accompanying **tag** (a slice of the virtual address). Subsequent addresses similarly index into the TB, and the tag bits are compared with the contents of the **TB Tag Store** at the indexed location. If the two tags match, the corresponding translation in the **TB Data Store** is used as the physical address. If the tags are not the same, a new translation is constructed, put into the TB, and used as the new address.

Physical addresses from the TB are put onto a local **TB Data-Address Line** (TB DAL) bus. Normally, this bus is used as an intermediate address bus, but it also serves as a data bus when the TB is being loaded with translations.

The TB may be turned off by writing to the **Translation Buffer / Cache Control Register (TBCCR)**, a local register used for program control over both the TB and the cache. When the TB is disabled, the upper 4 bits of the CPU address bus are ignored, causing all 68020 virtual addresses to be treated as physical addresses. The 68020 addresses are then routed through the **TB Bypass** buffers, instead of through the memory management circuitry. Another local register, the **Translation Buffer / Cache Flush Register (TBCFR)** allows programmer to flush the TB and/or the cache of data.

Chapter 2 describes in detail the operation of the memory management hardware.

1.3.3 The Cache

The CMPU module uses a high-speed, two-way associative **Cache** that minimizes the number of wait states incurred by the 68020 whenever it accesses memory. The CPU may incur a number of wait states whenever information is read from main memory by way of the SMI. When an instruction or data is found to be in the cache, the CPU does not need to access the SMI, and the instruction incurs no wait states. The cache also improves multiprocessor system performance since it filters out a large fraction of processor memory cycles before they reach the SMI.

The cache uses a read-allocate, write-through algorithm. Data read from main memory fills the cache. Data written to memory is checked for its presence in the cache and updated only if already present. The cache contents are organized as two sets of 512 8-byte entries, for a total cache size of 8 KBytes. As the CPU processes the retrieved longword (4 bytes), the cache automatically receives the next longword from main memory. Thus, if the CPU requests the next sequential word, it is available without delay in the cache.

Since multiple processors may use the SMI and share the same SMI memory resources, memory locations modified by one processor must be invalidated in the caches of other processors that may be caching the same data. The **Invalidation Stack** holds SMI addresses generated by other processors or I/O devices that are to be checked against the addresses currently held in the cache. Each of the 1024 cache entries has a status bit used to track whether or not the

associated entry has been invalidated. This tracking feature substantially simplifies the software that deals with processes that are running concurrently on different CPUs.

The cache is described in Chapter 3.

1.3.4 Interrupt Processing

The MC5600/5700 uses vectored interrupts. The **Interrupt Vector Decode** circuitry (INT VECT) is enabled whenever the 68020 begins exception processing. This decode logic produces a unique 8-bit vector indicating which device or group of devices has generated the interrupt. Eighteen out of the 256 possible vectors are assigned to MASSCOMP-specific devices or groups of devices. The interrupt handling hardware is described in Chapter 5.

1.3.5 Local Devices

Each CMPU module has a group of local devices that are accessed privately without using the SMI bus. **Local Decode** circuitry checks the TB DAL bus for a physical address referencing one of the local devices. If the CPU addresses one of these devices, the requested data is sent to or received from a private 8-bit **Local Data Bus**.

The local devices include two **Processor Control Registers** (PCRA & PCRB), which control various elements of the CMPU, and two **Dual Universal Asynchronous Receiver/Transmitters** (DUARTs P0 & P1) configured as 4 serial ports (described in Chapter 4). Three of these ports communicate with external devices such as terminals, printers, and modems. The fourth port is used to communicate internally with the Auxiliary Function Module (AFM). Also, a 64 KByte **EPROM (Erasable Programmable Read Only Memory)** contains the system bootstrap code (described in Chapter 12) and the **Console** code, used for manual bootstrap and diagnostics (described in Chapter 13).

1.3.6 The Buffered DAL Bus

All addresses not selecting a local device are routed to a 32-bit **Buffered Data Address Line** (DAL) bus. This bus is isolated from the neighboring buses with **Registered Buffers** (REG BUF).

One of these buffers is a **write buffer** that enables the 68020 processor to write to devices in SMI memory space without incurring a wait state. This buffered write feature insures the 68020 is kept running at full speed even during write operations.

Programming control over the write buffer is explained in Chapter 4.

1.3.7 MULTIBUS Adaptor

Each MULTIBUS on a system requires an associated **MULTIBUS Adaptor** (MBA). This circuitry on the CMPU module acts as the interface between the SMI and that particular MULTIBUS, accommodating differences in control lines, data sizes, and address translations between the two buses.

One crucial performance consideration in a virtual memory system is the way direct memory access (DMA) devices transfer data to memory. In a virtual memory system, the largest number of bytes in physical memory contiguously addressable in a DMA operation is one 4 KBtye page. If each DMA operation were to be limited to transferring only one page, the mass storage performance would be greatly impaired. Unless special hardware is provided for moving noncontiguous physical pages in a single transfer, I/O-intensive applications operate poorly.

The MBA includes an **I/O Map** that deals with this DMA transfer problem (among other issues). The I/O Map allows the transfer, in a single DMA operation, of multiple pages that are not contiguous in physical memory. It thereby provides for efficient I/O between main memory and peripherals by managing multiple-page transfers.

The I/O Map has 4096 entries, each mapping one page (4 KBytes). The Map is loaded and updated by the operating system. The I/O Map may also be changed by intelligent MULTIBUS devices to allow them to do their own mapping.

The MULTIBUS Adaptor is described in Chapter 10.

1.3.8 Floating Point Interface

Without the use of extra hardware, the performance of the 68020 degrades considerably when performing floating point operations as compared with integer operations. The MC5600/5700 system has two floating point devices available to speed up floating point operations. The Motorola MC68881 **Floating Point Co-processor** chip is standard on the MC5600/5700 and improves performance by about a factor of 10 over software floating point routines. The **Floating Point Accelerator** (FPA) module supplements the 68881 and improves on 68881 floating point performance by a factor of about three. Both floating point devices physically mount on the same connector and programs compiled for use with an MC68881 can still be run with an FPA installed (since the FPA also has an MC68881 onboard). Programs compiled for an FPA require the presence of an FPA to run on the system.

You can access the MC68881 with coprocessor instructions, as explained in the *MC68020 User's Manual.* You can access the FPA by setting a bit in the page tables used by the memory management hardware. Chapter 2 explains how to use this floating point interface.

1.3.9 Flush Logic

The cache and TB may be flushed of their contents by setting bits in a control register. The **Flush Logic** interprets the data from this register and invalidates the entries in the cache and TB as appropriate. This flush facility and the control registers are described in Chapter 2.

1.4 Memory Module

The MC5600/5700 memory module (CMM) is the memory array device used for system main memory. The modules use 256K Dynamic RAM chips and provide a total of two or four MBytes, with error detection and correction (EDC). The modules may be configured using a technique called interleaving to achieve maximum throughput. The CMM module is described in Chapter 6.

1.5 Auxiliary Function Module

The Auxiliary Function Module (AFM) that provides a number of system features, many of which would be redundant if on every CMPU module in a multiprocessing system. As a result, the functions performed on the AFM are described in various places in this manual. This section gives a brief description of the board and references the chapters where each of the module functions are more thoroughly described in their appropriate context.

The AFM is not an SMI or MULTIBUS device, but provides general services for the bus and CPU. The devices on the AFM are not assigned any physical address space or interrupt level. The AFM communicates directly with the boot CMPU module only (see Section 1.6), using that module's serial port P1B.

Figure 1-4 shows a block diagram of the AFM.

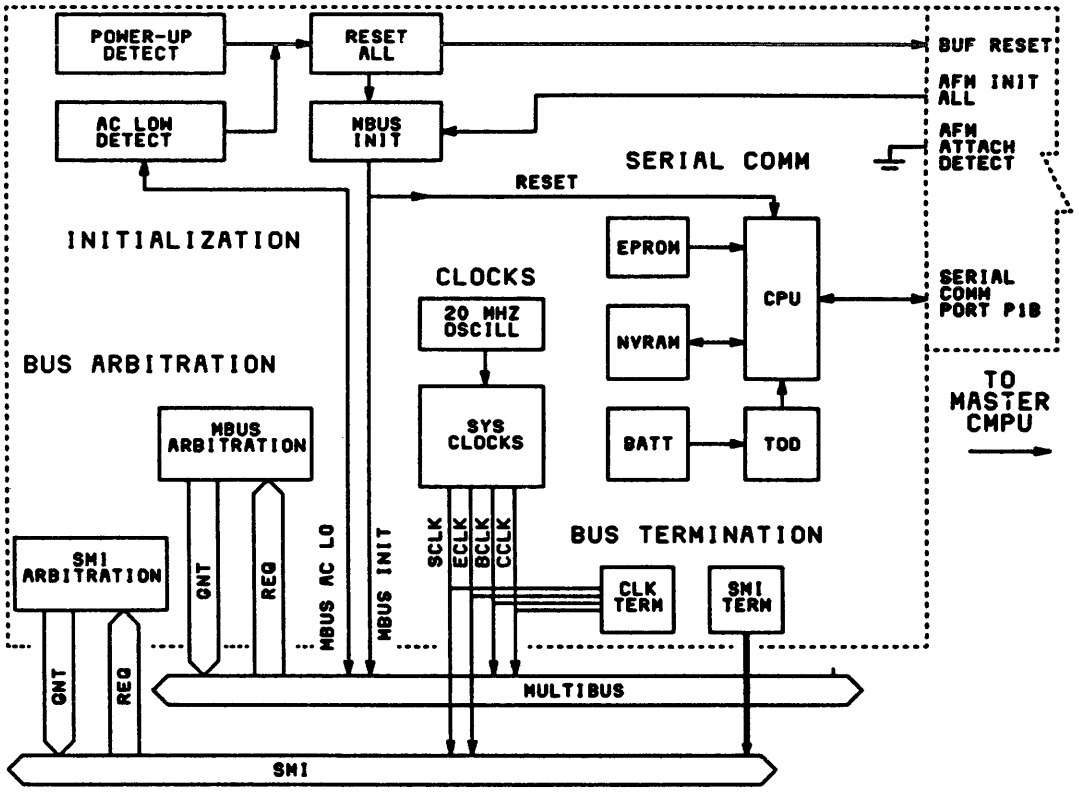


Figure 1-4. Auxiliary Function Module

The module is separated into five functional sections: Bus arbitration, CMPU serial communication interface, bus clocks, termination networks, and reset circuitry. Each of these parts is briefly described in the following sections.

Note that this section, unless otherwise stated, describes the operation of a fully populated AFM. A partially populated version of the AFM called the **Arbitration Module** (ARB) is also used on MC5700 systems with two 15-slot backplanes. The ARB contains only the MULTIBUS arbitration and termination circuitry. One fully populated AFM is required in every system, while one ARB is required for the additional 15-slot cardcage. Chapter 11 describes the AFM/ARB configuration issues in a system.

1.5.1 Bus Arbitration

The AFM arbitrates both the SMI and the MULTIBUS, granting the use of each bus to the device issuing the highest priority request. The MC5600/5700 system uses a parallel priority scheme for the arbitration of both the MULTIBUS and the SMI buses.

In parallel arbitration, each module on each bus has one request line and one grant line associated with it. All request and grant lines enter the arbitration circuitry of the AFM. The AFM issues a grant to the device with the highest priority level request. When this request is removed, the next highest priority request is granted, until all pending requests have been granted.

Specific bus arbitration issues are further described in Chapters 7 and 9.

1.5.2 CMPU Serial Interface

The AFM contains the Time of Day Clock (TOD) for the system. This clock is continuously powered, using its own battery backup when system power is unavailable. The device uses internal 8-bit registers for data storage. The accuracy of the real time clock is a approximately ± 18-20 seconds per month.

The AFM has a dedicated microprocessor for transmitting its data to the boot CMPU and, through this, to the system. This microprocessor monitors the registers on the TOD to determine the time of day and day of week, and it also reads and writes to sections of the NVRAM on command. The NVRAM is used to store default bootstrap variables used in powerup, the system's serial number I.D., and space for optional customer-designed boot code. An on-board EPROM contains code to run the AFM microprocessor.

Chapter 12 contains programming instructions for accessing the NVRAM during a bootstrap program.

1.5.3 Initialization Circuit

The AFM contains initialization circuitry that detects system powerup, the RESET switch on the front panel, power outage, or a RESET instruction from the boot CMPU. The circuit responds differently in each of these conditions. This circuitry is explained in Chapter 5.

1.5.4 Termination Networks

Termination is provided on the AFM for bus signals that require it (such as the SMI DAL lines and the system clock signals). At the high data rate of the buses, a delayed signal reflection may be interpreted as response to the original bus operation signals. To attenuate this reflection, the signals are terminated with a network of resistors. Termination issues are a consideration only in system configuration, as explained in Chapter 11.

1.5.5 Bus Clocks

The SMI bus clocks SCLK and ECLK and MULTIBUS clocks BCLK and CCLK are generated on the AFM. The SMI bus clocks are covered in Chapter 7 and the MULTIBUS bus clocks are covered in Chapter 9.

1.6 Multiprocessor Systems

The CMPU and SMI bus have been designed to allow two or more CMPU modules to share the SMI bus and its memory devices. The SMI protocol minimizes the bandwidth used by each access, allowing for an efficient, tightly coupled multiprocessing environment. A multiprocessor system may have one MULTIBUS per central processor, with up to 2 MULTIBUSs per MC5600 and 4 MULTIBUSs per MC5700 system.

Figure 1-5 shows a typical multiprocessor MC5700 configuration, with two processors and two associated MULTIBUSs.

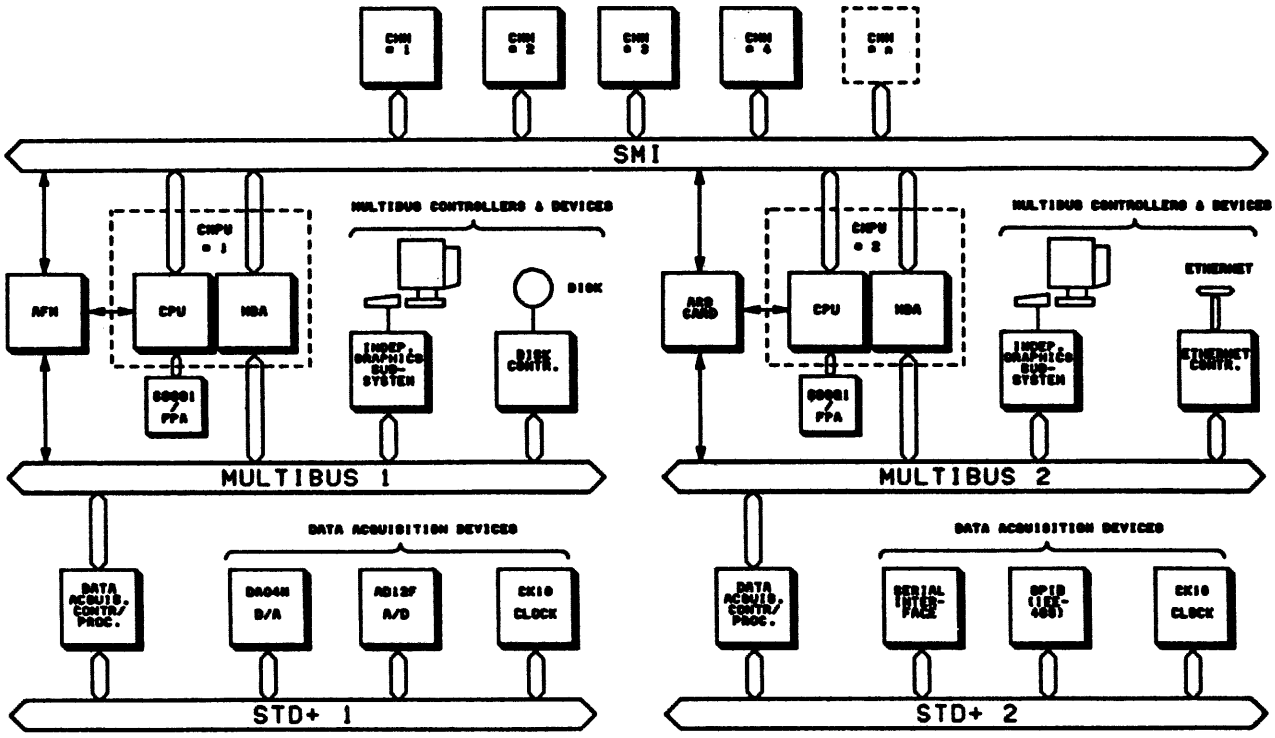


Figure 1-5. A Multiprocessor MC5700 System

Each CMPU in the system is assigned a **Processor I.D.** value of **0** through **7**, as set on the CMPU module's switchpack. These switches set the node I.D. used in SMI transfers for both the MBA and CPU, and also set the I.D. number for that MBA's MULTIBUS. This manual refers to the MULTIBUS with the same I.D. as a given CMPU as that CMPU's **associated** MULTIBUS. Note that this term does not imply any limitation to access by other processors.

Each Processor I.D. value (except **0**) has been assigned a unique SMI address space for that processor's associated MULTIBUS memory and I/O devices, its MULTIBUS I/O Map, and its **Inter-Processor Interrupt Register (IPIR)**, used for delegating tasks between processors. A CMPU module may or may not have an associated MULTIBUS and may or may not use any or all of the address space it has been assigned. Chapter 11 describes how to disable the MBA (and its associated MULTIBUS) on a CMPU module.

In a multiprocessor system, one processor must be assigned the task of performing the system bootstrap, executing the on-board console code if necessary, and loading the operating system into memory. Since this processor must be able to initialize the entire system during the bootstrap, it also must have sole authority over the system's RESET signal. In the MC5600/5700, this is called the **Boot Processor** and all other processors in the system are termed **Non_boot processors**.

The boot processor has the following properties:

- Its processor I.D. is **1**. The code resident in the EPROM on Processor **1** is always used to boot the system (see Chapter 12). All systems require one CMPU module to have Processor I.D. **1**.
- The boot processor's fourth serial port (P1B) is attached to the AFM with a private cable. This communication link gives the boot processor exclusive access to the following AFM functions:
    1. System reset priority. Only the boot processor is able to reset the system by asserting its RESET line. This capability is necessary during system bootstrap. Non-boot CMPUs cannot reset the system.
    2. The Time of Day Clock. This clock is used for time of day operations and has a battery backup on-board the AFM.
    3. The system boot environment. Certain flags used during the bootstrap are stored on the AFM in Non-Volatile RAM (NVRAM). These flags are explained in Chapter 13.
    4. The Customer Bootstrap Code. The NVRAM on the AFM can also store customized device drivers for non-MASSCOMP boot devices (see Chapter 12).

With the exceptions noted above of system reset priority and AFM accessibility, all CMPU modules are equal processors in a MC5600/5700 multiprocessing environment, in that they have equal access to all devices on the SMI, including all MULTIBUS Adaptors. Each CMPU module operates independently and, unless noted, uses the same hardware and algorithms described in the following chapters.

1.7 MULTIBUS Controllers

Because MASSCOMP uses the industry standard MULTIBUS, there are many devices available on the market that can be easily interfaced to an MC5600/5700 system. The *MASSCOMP MC5600/5700 System Configuration Guide* describes how to interface MULTIBUS devices both supported and unsupported by MASSCOMP.

1.8 STD+ Data Acquisition Devices

The *Data Acquisition Application Programming Manual* describes both the hardware and software considerations in using the DACP and STD+ data acquisition devices. The STD+ bus is not covered in this manual.

Chapter 2
Memory Management Hardware

ILLUSTRATIONS

TABLES

Chapter 2
Memory Management Hardware

This chapter describes the memory management hardware on the CMPU module and its implications for the system programmer. It describes how addresses are translated before being sent out on the SMI bus, and how these address translations are cached. The division of virtual address space and the secondary address spaces used by the system are also covered. These spaces include two control/status registers that control both the memory management hardware and the cache.

2.1 Introduction

The MC5600/5700 is a virtual memory machine. Addresses sent out by the 68020 refer to the virtual address space perceived by the programmer, which for that microprocessor is 4 GBytes. However, the MC5600/5700 system's physical memory space (described in Chapter 8) is only 256 MBytes or less, and can hold only a fraction of the virtual address space at any instant. Programs, of course, are usually much smaller than the virtual address space, but they may be larger than the physical memory in the system due to large data arrays, or spread throughout the virtual space due to conventions of the operating system (OS). Thus, as a program runs there must be a way to dynamically manage the physical memory space so it *appears* to be the same size as the virtual space.

A second consideration in managing memory space is that the system must support a multiprogram environment. The OS must be able to timeslice among many ongoing processes, all coexisting with the OS code in main memory (given small enough programs). So, memory management must be able to protect the operating system from being overwritten or tampered with by any resident user programs. Each user program also must be able to perceive its own unique 4 GByte virtual space without seeing any other program's space. For example, the physical location that Program A sees when it sends out virtual address 0x0 must be different from the location Programs B or C see when accessing the same virtual address. This is, of course, unless Programs A and B are sharing the data in a given location. The capability for allowing multiple access to data must also be accommodated by the memory management.

All of these considerations have been integrated in the MC5600/5700 memory management hardware design:

- **Managing Memory Size.** All memory is managed using a technique called **demand paging**, where one 4 KByte **page** of contiguous address space is handled at a time. Pages of program data and text are copied into main memory only when explicitly referenced by a running process.

  When a program is read into physical memory one page at a time, each page's virtual address is first **mapped** to the physical address that is actually used. Mapping is done by the OS, by dynamically setting up **page tables** in main memory. These page tables keep track of the physical location of each virtual page, with a unique set of page tables kept for each process. Then, each virtual

address referenced by the running program is **translated** into a physical address, using the page tables, and then sent out over the SMI bus. The hardware that performs this virtual-to-physical translation is called the **page table engine**.

- **Performance**. The memory management hardware uses a performance enhancement called the **Translation Buffer (TB)**, a high-speed RAM that stores virtual-to-physical translations. The TB keeps close at hand the address translations that have most recently been constructed by the page table engine. Thus, the most frequently used translations do not have to be fetched from memory and reconstructed on every consecutive access by a running program.
- **Multi-program Support**. The page tables contain read and write access information about each page that is being mapped. The access code is compared with the permissions given to the running process on every virtual address reference. Thus, supervisor-owned pages may be protected from access by user programs.

  Also, the TB has been divided into **System** and **Program** sections. By writing to a control register, all TB entries in either section can be invalidated (or **flushed**) independently. This allows the OS to divide the virtual address space into corresponding sections. The translations for the OS can remain in the TB while different user programs alternately execute on a given CMPU.
- **Shared Access to Memory**. A separate type of page table (called **Indirect**) can be set up to allow different programs to share pages in main memory using the same page table entries.

2.2 The Translation Buffer

The central element of the memory management hardware is the Translation Buffer (TB), a 1024 x 32 bit high-speed RAM. Each 32-bit **Translation Buffer Entry (TBE)** contains a virtual-to-physical translation for a 4 KByte page that has previously been translated and copied into main memory. Whenever the 68020 puts an address onto its internal bus, the memory management hardware first checks to see if a corresponding TBE is already at hand in the TB to translate that virtual address.

- If an entry corresponding to the address is present, a **TB Hit** occurs. The translation in the TB is used immediately to convert the virtual address to the proper physical address. This process is described in this section.
- If an entry corresponding to the address is NOT present, a **TB Miss** occurs. The translation is not stored in the TB and must be first constructed from data fetched from page tables in main memory that have typically been set up by the operating system. This process is done by the page table engine and is described in Section 2.3.

The operation of the TB and its associated hardware is shown in Figure 2-1.

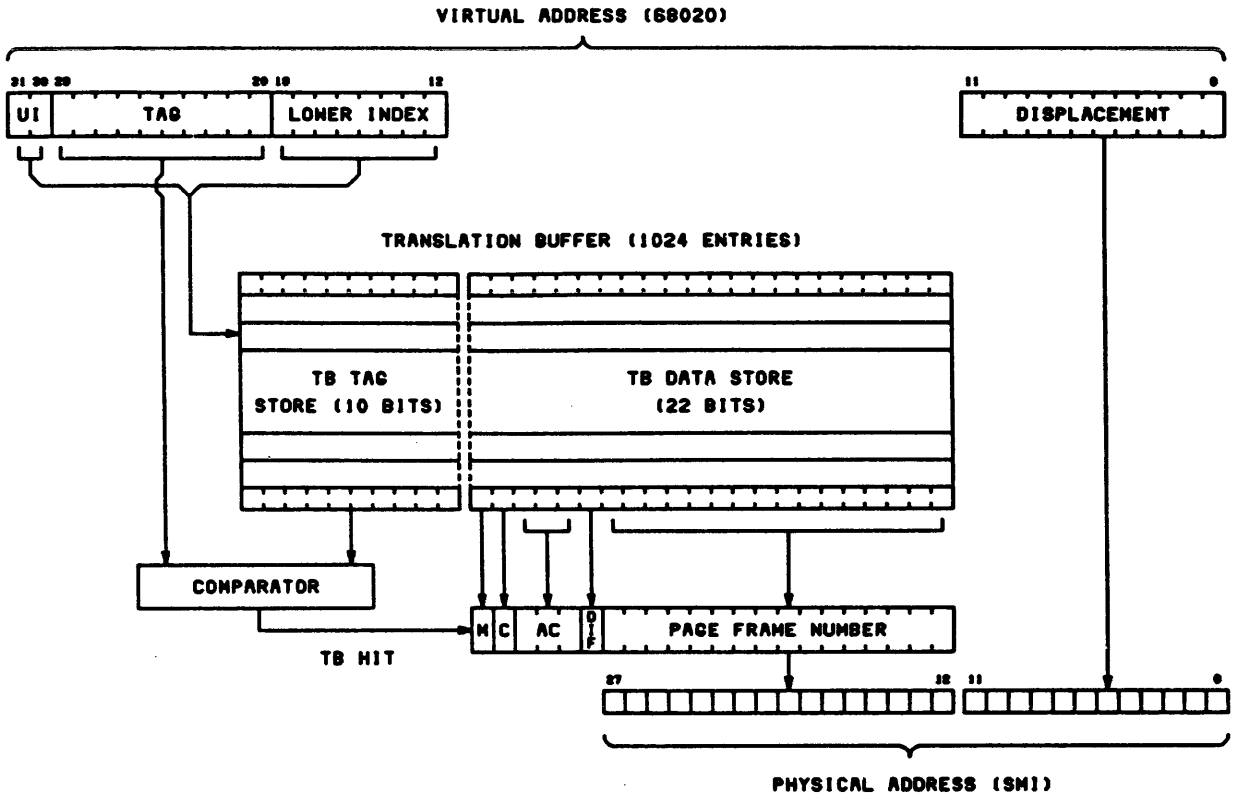


Figure 2-1. Translation Buffer

Each of the 1024 Translation Buffer Entries consists of two parts, the **TB Tag Store** and the **TB Data Store**. The TB Data Store is the address translation, along with various control bits, that has previously been constructed from software page tables during a previous TB miss. The TB Tag Store holds the corresponding **tag** for each translation. The tag is the 10-bit slice of the virtual address which, along with the index, guarantees an exact identification of the address. If the tag portion of the virtual address and the Tag Store entry at the indexed location are the same, the addresses are the same (a **TB match**).

A virtual address coming from the 68020 is split into four parts by the TB hardware, as shown at the top of Figure 2-1:

- **UI (Upper Index)** <31:30> - These bits are used as the upper two address bits of the TB RAM to index into the 1024-item list. Notice that the upper index effectively splits the TB into four quadrants. This may be used by software for separating the virtual space into system and program areas, which may be independently flushed (see the description of the TBCCR in Section 2.5.3 and the description of system and program sections in Section 2.4).
- **Tag** <20:29>- The tag is compared with the tag portion of the virtual address to determine a TB match. The 16 bit Page Frame Number field in the TB Data

Store at the indexed location replaces the virtual address' upper 20 bits to form the complete physical address.

- **Lower Index** <19:12> - These bits are used as the lower eight bits of the TB RAM to index into the 1024-item list.
- **Displacement** <11:00> - This part of the address specifies the location within the 4 KByte page. These bits are passed through the memory management hardware untouched and used as the lower part of the final SMI physical address.

Each TB Data Store entry is composed of the actual translation (or **Page Frame Number (PFN)**) and six status bits. On a TB match, the hardware checks the **AC, DIF,** and **M** status bits (described in Section 2.3.6), and if these are in order, a TB hit occurs. The physical address is put onto the SMI bus and the 68020 instruction cycle continues from the point where it left off. The **C** bit is used by the cache as explained in Chapter 3.

Each TB Data Store entry can be accessed directly by the 68020, as described in Section 2.5.1.

2.3 Page Table Engine (TB Miss)

If the TB fails to come up with the physical translation for an address for any reason, the **Page Table Engine** is started. The page table engine is the hardware used to construct the translation for the virtual address requested by the 68020, using **Page Table Entries (PTEs)** in main memory. This hardware is not directly accessible to the programmer (except for the fact that memory management can be disabled as a whole) and so is not shown here in any of the hardware block diagrams. The page table engine remains in an idle loop as long as each 68020 address cycle generates a TB hit. It will break out of this loop for any of the following reasons:

- **TB Miss** - The translation is not present in the TB
- **M bit update** - The page table engine updates the **Modify (M)** bit in the PTE in main memory and in the TBE whenever a previously unmodified page is written (see Section 2.3.6).
- **Access violation** - The access privilege of the 68020 bus cycle does not match the access code for the page
- **A move to or from Diagnostic Space** (described in Section 2.5.2)

In any of these cases, the page table engine first inhibits the acknowledge signal to the 68020 read or write cycle. The engine then attempts to execute the page table algorithm (described in the next section) and retrieve the new translation. If the algorithm is successful, the page table engine loads the translation into the TB and allows the 68020 to continue with its instruction cycle from the point where it left off. If the algorithm is unsuccessful, the engine restarts the 68020 by generating one of the following signals:

1. **BERR** - The engine forces a bus error to indicate that a problem has occurred that the 68020 must handle directly. Any of the following situations generates a bus error and clears bit <5> in the PCRB:
    - **Access protection violation** - The PTE AC bits deny access to a page (see Section 2.3.6)

- **Invalid PTE** - The **I** bit in the PTE (described in Section 2.3.6) indicates that the PTE copied from main memory is invalid
- **Nested Indirect** - The **D** bit of an indirect PTE is **0**, implying that the PTE points to another indirect PTE (an address) rather than an actual table (see Section 2.3.3).
- **SMI NACK response to a PTE fetch** - A memory device is not responding correctly (Chapter 7 describes the SMI NACK response)
- **SMI RERR response to a PTE fetch** - Data in a memory device is corrupted (Chapter 7 describes the SMI RERR command)
- **PTE consistency error** - The AC bits are **0** (no access or invalid) and the **I** bit is **0** (valid).

2. **DSACK (Forced)** - This is generated in response to a 68020 MOVES instruction to or from Diagnostic Space (described in Section 2.5.2). If the page table engine successfully executes the translation algorithm and retrieves the PTEs, this signal prevents the address from being sent out over the SMI bus.

2.3.1 Page Table Algorithm

The page table engine uses an algorithm that involves two levels of page tables initially set up by the operating system to map each page in system memory:

- **Second Level Page Tables (SLPTs)** contain the base addresses to the physical pages being mapped
- **First Level Page Tables (FLPTs)** contain the base addresses to the SLPTs

This two level algorithm minimizes the amount of physical memory needed to keep page tables for the entire virtual memory space.

Figure 2-2 shows how the page tables are used.

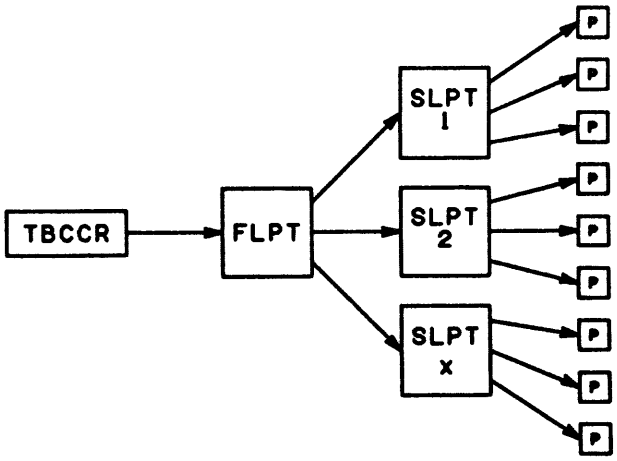


Figure 2-2. Page Table Levels

Typically, each time a user process begins execution on a CPU (that is, the context is switched by the OS), the virtual address associated with the process is first loaded into the Base Page Frame Number field in the **TB / Cache Control Register (TBCCR)** (described in Section 2.5.3). This process address points to the FLPT for the running process. A portion of each virtual address of the running program is used to index into this FLPT. Each entry in the FLPT, in turn, contains a base address that points to an SLPT. A second portion of the virtual address is used to index into this SLPT. The indexed SLPT entry contains the base physical address for the actual page in memory. This address, concatenated with the page displacement from the virtual address, is the final physical address that replaces the original 68020 virtual address. This two-stage method allows the most efficient access to the 4 GBytes of virtual memory space required by each program.

Also, to allow different processes to share portions of the same page tables, each level of the page table algorithm may be executed as a **direct** or **indirect** algorithm:

- A **direct** Page Table entry contains the physical address of the next table.
- An **indirect** Page Table entry contains the physical address of the *address* of the next table.

Figure 2-3 shows the direct page table algorithm in detail, and Figure 2-4 shows the full two-level indirect page table algorithm. Sections 2.3.2 through 2.3.5 describe the algorithms, and Section 2-5 describes each bit field in each entry. Note that first direct/second indirect and first indirect/second direct combinations are also allowed.

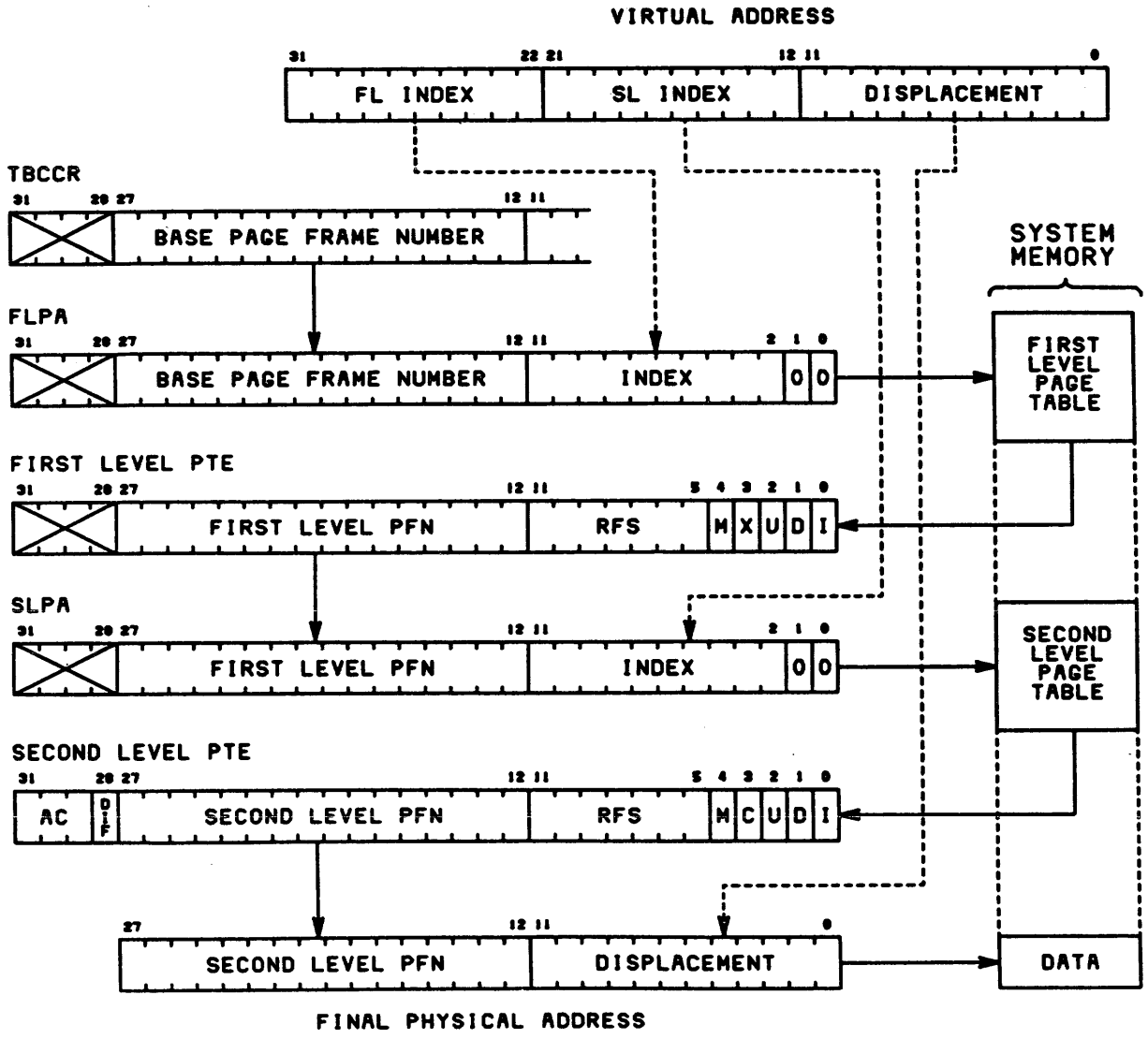


Figure 2-3. Page Table Algorithm (Direct)

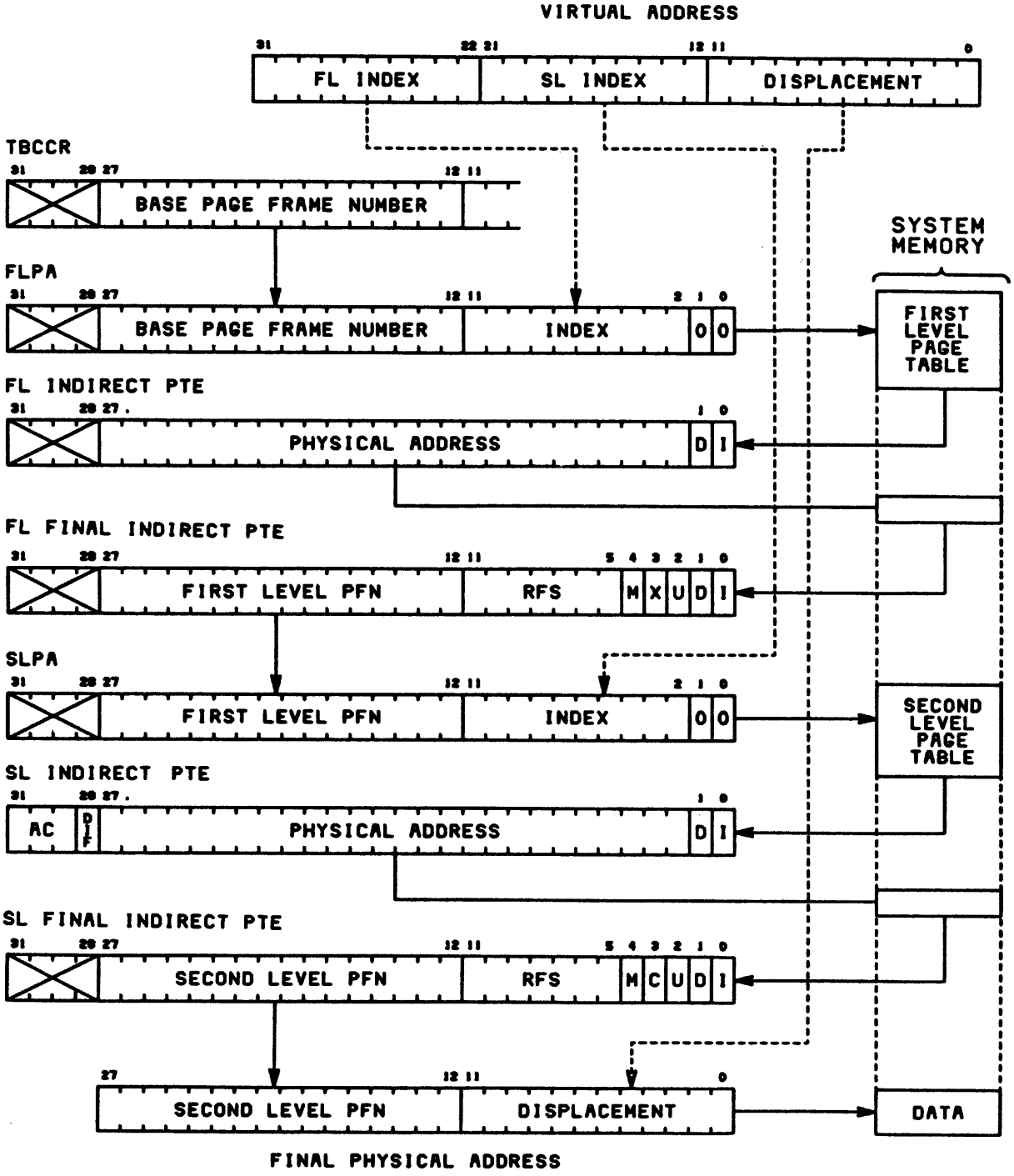


Figure 2-4. Page Table Algorithm (Indirect)

The page table engine divides the virtual address from the 68020 into three fields: **First Level Index, Second Level Index**, and the **displacement**. These fields are used throughout the translation algorithm. The stages of the page table algorithm (direct and indirect) are

described in the next four sections, followed by a field-by-field description of each PTE.

2.3.2 First Level Direct

In the first step of the translation, the First Level Index (10 bits) and the contents of the BFPN field (16 bits) in the TBCCR are used to form the **First Level Physical Address (FLPA)**. The FLPA gives the physical address of the First Level PTE. The lower two bits of the address are forced to be **0**, since each entry in the FLPT is an aligned longword.

The First Level PTE is fetched from system memory and stored in a temporary register. Each entry is made up of a PFN, a group of status bits, and bits reserved for software, as shown in Figure 2-5 and described later. These bits are checked in the following order:

- If the **I** bit indicates that an entry is invalid (that is, the associated page is not presently in memory), the logic stops and a BERR is generated.
- If the **D** bit indicates the entry is an indirect entry, the logic jumps to the first level indirect algorithm (described in the next section).
- The **M** and **U** bits are both set to **1** and the modified PTE is written back into memory. This read/modify/write is an atomic cycle.

Bits <27:12> of the entry (the PFN) are retained for the second level stage.

2.3.3 First Level Indirect

If the **D** bit in the first level PTE is **0**, the entry is used as the physical address in system memory space to another PTE. The format of a First Level Indirect PTE is shown in Figure 2-5. Bits <27:00> are used as the physical address and bits <01:00> are zero to provide a longword-aligned address.

The data pointed to by the indirect PTE is fetched and has the same format as the First Level Direct PTE. The engine now continues the First Level algorithm normally, with one caveat. If the **D** bit in this First Level PTE is found to be **0** (indirect), the process is aborted and a BERR is generated. This error is called a **nested indirect**.

2.3.4 Second Level Direct

The PFN from the First Level PTE is concatenated with the Second Level Index from the original 68020 virtual address to form the **Second Level Physical Address (SLPA)**. The SLPA gives the physical location of the Second Level PTE. Again, the lower two bits of the address are **0**, since each entry in the SLPT is an aligned longword.

The Second Level PTE is fetched from system memory space and stored in a temporary register. The format for a Second Level Direct PTE is shown in Figure 2-5. Each entry is made up of a PFN, a group of status bits, and a section reserved for software, as well as access (**AC**) bits that determines if the page is accessible by for this particular process. These bits are checked in the following order:

- If the **I** bit indicates an invalid entry, the page table engine stops and a BERR is generated.
- If the **D** bit indicates the entry is an indirect entry, the engine jumps to the second level indirect algorithm . (described in the next section)
- The **AC** bits are checked against the current bus cycle type of the 68020, and against the **I** bit for consistency. If the page's access code or a PTE inconsistency prohibits the operation, the engine stops and a BERR is generated.
- The **U** bit is set to **1** and the **M** bit is also set if the 68020 is executing a write cycle. The modified PTE is written back into memory. This read/modify/write is an atomic cycle.

The PFN from the SLPT entry is combined with the displacement field from the original virtual address to form the physical address. The SLPT entry is stored in the TB Data Store in the format shown in Figure 2-7. The tag is stored in the corresponding index in the TB Tag store. The 68020 instruction cycle is allowed to continue, now generating a TB hit, and the page table engine returns to its idle state.

2.3.5 Second Level Indirect

If the **D** bit in the second level PTE is **0**, the entry is used as the address to another PTE. The format of a Second Level Indirect PTE is shown in Figure 2-5. Bits <01:00> in the indirect entry are **0** since the address is longword aligned.

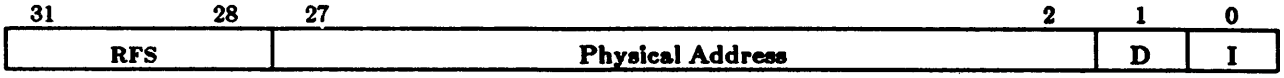
The data pointed to by the indirect PTE is fetched from system memory. This table entry is called **Second Level PTE (Final Indirect)** and is slightly different from the Second Level Direct PTE format. Bits <31:28> (**AC** and **DIF**) are derived from bits <31:28> in the indirect entry, rather than the entry from the page table. Again, if the **D** bit in this Second Level PTE is found to be **0** (nested indirect), the process is aborted and a BERR is generated.

The engine now follows the second level direct algorithm normally.

2.3.6 Page Table Formats

Figure 2-5 shows the bit fields for First and Second Level Table entries for both direct and indirect addressing.

First Level Page Table Entry (Indirect)



First Level Page Table Entry (Direct/Final Indirect)

Second Level Page Table Entry (Direct)

Second Level Page Table Entry (Indirect)

Second Level Page Table Entry (Final Indirect)

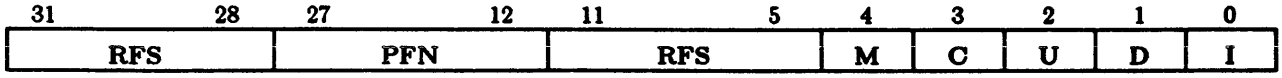


Figure 2-5. PTE Bit Field Formats

The fields descriptions of all PTEs are as follows:

AC **Access Code** (3 bits). The **Access** bits indicate the read/write access privilege for supervisor and user programs to the associated page. Memory management hardware compares the **AC** field with the **FC** and **R/W** signals of the 68020 to determine the type of access being attempted. If the running bus cycle does not have access permission to the page, it causes is called an **access violation**. Table 2-1 shows the access code assignments.

Table 2-1
Access Codes

AC			Supervisor	User
MSB		LSB		
0	0	0	No Access	No Access
0	0	1	Execute	No Access
0	1	0	Read/Execute	No Access
0	1	1	Read/Write/Execute	No Access
1	0	0	Execute	Execute
1	0	1	Read/Execute	Read/Execute
1	1	0	Read/Execute	Read/Execute
1	1	1	Read/Write/Execute	Read/Write/Execute

Note that the access codes are not implemented in the First Level PTE.

C **Cache** (1 bit). If this bit is a **0**, the data in the associated page is not cached.

D **Direct/Indirect** (1 bit). This bit indicates whether the addressing algorithm used is direct (D=**1**) or indirect (D=**0**).

DIF **Don't Invalidate / Floating Point** (1 bit). This bit has a dual function:

1. If the physical address is in SMI memory space (above 0x8000000), setting this bit to a **1** causes the **DI** signal on the SMI bus to be a **1** on the transfer. On write transfers with the **DI** bit set to a **1**, entries in other processors' caches are not invalidated against the associated address. The write transaction will still be used to update the contents of the local cache on a tag store match. In this case, the invalidation logic on other processors ignores the transfer and the address is not pushed onto their respective invalidation stacks. Chapter 3 describes the cache and the invalidation process.
2. If the physical address is in SMI I/O space (below 0x8000000), setting this bit to a **1** causes the transfer to be made to/from the Floating Point Accelerator Module. In this case, no access is made to the SMI.

I **Invalid** (1 bit). This bit indicates that the PTE is valid (I=**0**) or invalid (I=**1**). If this bit is found to be set in a PTE fetched from memory, the engine restarts the 68020 with a BERR signal.

M **Modify** (1 bit). This bit is set whenever any data in the associated page is modified. The page table engine automatically keeps this bit properly updated in both the TB and the main memory copies of every PTE.

PFN **Page Frame Number** (16 bits). This field is used to create the upper 16 bits of the associated physical address.

RFS **Reserved for Software.** The translation hardware maintains the integrity of these (and all PTE) bits while manipulating the PTEs.

U

Used (1 bit). This bit is set to a **1** whenever the PTE has been used. The hardware logic keeps the PTE in main memory properly updated.

X **Don't care**. These bits are ignored.

2.4 Virtual Address Space

Since the 68020 has 32 address lines, the virtual address space that the processor sees is 4 GBytes. The virtual address space on the MC5600/5700 has been divided into two sections shown in Figure 2-6, called System space and Program space. The 1024 entries of the TB have also been partitioned into two sections, with 256 translation entries allocated to the 1 GBtye System space and 768 entries allocated to the 3 GBtye Program space. This is done by using the upper two bits of the virtual address as the upper bits of the 10-bit TB index, as described in Section 2.2.

The advantage of this division from the programmer's standpoint involves the TBCFR (see Section 2.5.3). Two bits in the TBCFR allow the Program section (0x0 to 0xBFFFFFFF) or the System section (0xC0000000 to 0xFFFFFFFF) of the TB to be flushed independently if needed. while keeping the translations for the system OS code intact. When all PTEs in the Program section of the TB are invalidated in this way, the page table engine is forced to construct new translations, but only for those addresses in Program space. This feature is typically used to flush the address translations for user programs on an context switch, saving the OS software considerable overhead.

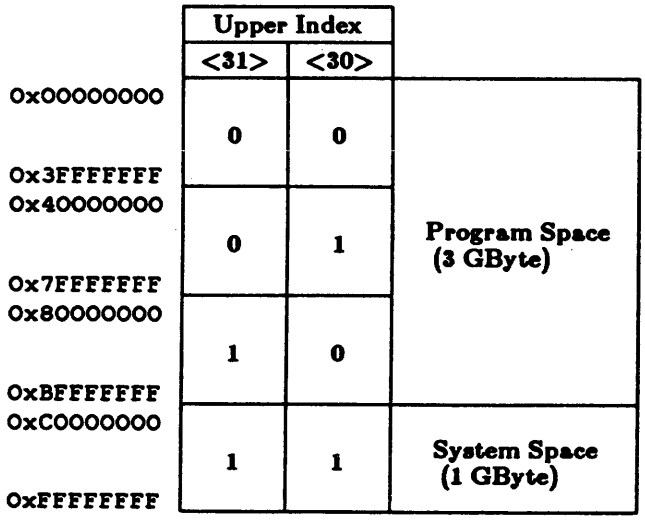


Figure 2-6. Division of Virtual Address Space

2.5 Secondary Address Spaces

There are three secondary address spaces reserved by MASSCOMP in the MC5600/5700: **Diagnostic**, **Processor Register**, and **Translation Buffer** space. Each is the same size as the standard virtual address space. These have been created to allow the programmer to access special registers without having to use the normal transfer path. The spaces allow the memory management, cache and SMI decode logic to be bypassed in these transactions.

The secondary address spaces are accessed using the 68020 MOVES instruction, with the 68020 DFC or SFC register (depending on the transfer direction) loaded with the binary code **011**. The **Space Modify** bits <1:0> in the PCRA register (see Chapter 4) determine which of the three address spaces is being accessed, as shown in Table 2-2.

Table 2-2
Secondary Address Spaces

PCRA Bits		Space
1	0	
0	0	Translation Buffer
0	1	Processor Register
1	0	Diagnostic
1	1	Illegal

Each of these address spaces is described in the following sections.

2.5.1 Translation Buffer Space

The Translation Buffer Address Space is used to directly access entries in the Data Store portion of the TB, essentially treating the TB as a standard RAM device. It is a unique 4 GByte virtual space that contains only TB entries.

Reads and writes to TB space simulate the normal operation of the TB. The upper and lower index of the operation's virtual address are used to index into the 1024 entry TB RAM. A translation is then directly read from or written to the indexed TB location using the 68020 data bus, in a data format unique to TB space operations.

The TB space entry format is shown in Figure 2-7.

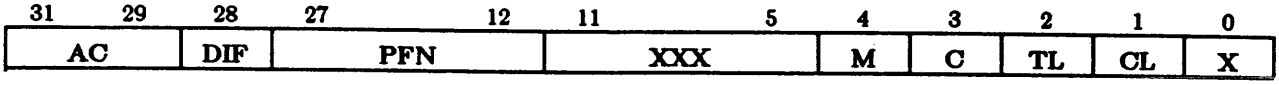


Figure 2-7. TB Space Entry Format

The fields in the TB Data Store entry have the same definition as the Page Table Entries shown in Figure 2-5, except for:

TL **TB Hit** (bit <2>). This bit is a read-only bit used only when the 68020 is accessing TB address space. This bit is a **0** when the contents of the TB Tag store matches bits <29:20> of the referenced virtual address. This bit allows the TB tag stores and comparators to be tested by diagnostics.

CL **Cache Hit** (bit <1>). This bit is a read-only bit also used only when the 68020 is accessing TB address space. The bit is a **0** when any access through TB space generates a hit in one of the cache tag stores (described in Chapter 3). The address presented to the cache tag comparators is the PFN field concatenated with displacement bits <11:03> of the virtual address. The **CL** bit allows the cache tag stores and comparators to be tested by diagnostics.

On a write, bits <29:20> of the virtual address (the tag) are written into the indexed location of the TB tag store. The data on the 68020 data bus is written into the indexed location of the TB data store, in the format shown in Figure 2-7. The MOVES instruction *must* specify a longword operand. Note that only the **AC, DIF, PFN, M,** and **C** fields in the TBE are implemented on a write. The data in all other fields (including **TL** and **CL** bits) are ignored.

On a read, the tag in the indexed TB location is compared with the tag portion of the virtual address. If they are the same, the **TL** bit in the TB space entry is a **0**. If they are not the same, the **TL** bit is a **1**. In either case, the entry is put on the 68020 data bus in the format shown in Figure 2-7, and read using the MOVES instruction.

2.5.2 Diagnostic Space

Diagnostic space is used to verify that the page table engine is working properly. Memory management hardware should be disabled when accessing this space by writing a **1** to bit <0> in the TBCCR (see Section 2.5.3). Accessing a virtual address in diagnostic space causes a translation to be constructed without actually completing the 68020 bus cycle. The page table engine loads the new translation into the indexed location in the TB data store. The TB hardware then generates a forced DSACK and no physical address is presented to the SMI. If an error occurs, the BERR is asserted as usual.

2.5.3 Processor Register Space

A separate space has been assigned exclusively for two registers used in conjunction with the cache and TB. These registers are the **Translation Buffer / Cache Control Register (TBCCR)** and the **Translation Buffer / Cache Flush Register (TBCFR)**. These 32-bit registers are located within this space as follows:

- **TBCFR** - The lower half of processor register space (bit <31> = 0)
- **TBCCR** - The upper part of processor register space (bit <31> = 1)

The fields in each of these registers are shown in Figure 2-8.

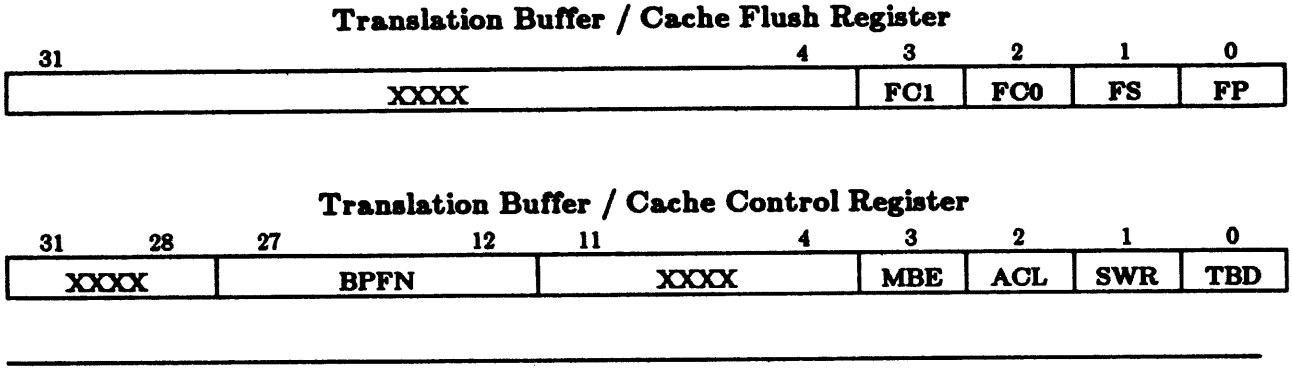


Figure 2-8. TBCCR and TBCFR Entry Formats

The TBCFR is a write-only register. The following is a list of the TBCFR fields:

FC1 **Flush Cache 1** (bit <03>). Setting this bit to 1 flushes the cache Set 1 (see Chapter 3)

FC0 **Flush Cache 0** (bit <02>). Setting this bit to 1 flushes the cache Set 0 (see Chapter 3)

FS **Flush System section** (bit <01>). Setting this bit to 1 flushes the System section (upper 256 entries) of the TB

FP **Flush Program section** (bit <00>). Setting this bit to 1 flushes the Program section (lower 768 entries) of the TB

XXXX Not used

The TBCCR is a read/write register. The following is a list of the TBCCR fields:

BPFN **Base Frame Page Number** (bits <27:12>). This field is used in the page table algorithm to locate the First Level Page Table (see Figure 2-3). This number is typically loaded into the TBCCR by the OS on context switch.

MBE **SYSTEM ERROR** (bit <03>). This read-only bit is **0** when the system error line (labeled MBUS ERROR and described in Chapter 9) has been asserted. MBUS ERROR is asserted whenever any processor on the SMI is

halted.

ACL **AC Low** (bit <02>). This read-only bit is **0** when the AC LO signal on the MULTIBUS is asserted.

SWR **SW Reset** (bit <01>). This read-only bit is a **0** when the INTERRUPT switch is pressed. The bit is reset to **1** when the switch is released.

TBD **Translation Buffer Disable** (bit <00>). When this bit is cleared, the Translation Buffer, page table engine, and memory management logic become enabled. Clearing this bit also enables the cache if the cache set bits in the PCRA (see Chapter 3) are enabled. When this bit is set to a **1**, memory management and cache become disabled. In this case, no translation takes place, effectively causing the machine to treat the lower 28 bits of all virtual addresses as physical addresses.

Chapter 3
The Cache

ILLUSTRATIONS

TABLES

Chapter 3
The Cache

The cache is a high-speed memory buffer that enhances system performance. Studies have shown that typical programs spend most of their time in a few small routines or tight loops. Once this code is captured in the high-speed cache, these active code segments can execute significantly faster than if each instruction had to incur wait states while being fetched from memory.

This chapter explains in detail the cache, its operating modes, its control/status registers, and the process of invalidating cached data that has become stale.

3.1 General Operation

The cache is structured similarly to the TB, and operates on physical addresses taken from the TB. When the 68020 does a read operation, a part of the physical address is used to index into a list of cache address **tags**. If the indexed tag matches the remaining portion of the address, it is called a **cache hit**. The data cached in the indexed location is put on the internal 68020 data bus, and no fetch operation is initiated on the SMI bus. An acknowledge is returned to the 68020 to end its cycle.

The MC5600/5700 uses an 8 KByte two-way associative cache structure. A **one-way associative** cache structure guarantees that a given memory location is always cached in one unique slot. Each time a cache slot is filled, the new data always overwrites the previous value cached in that slot. This cache memory is a single space that wraps around *n* times, where *n* is the cache size divided into the size of the physical address space. The TB is an example of a one-way associative structure.

Two-way associative mapping uses two sets of memory locations that share the same address lines. In two-way mapping, a given memory location can be cached in either set, allowing some flexibility in deciding which set's old cache value is overwritten. It allows two memory locations with the same index bits to be simultaneously resident in the cache.

When the requested data value is not found to be in the cache (called a **cache miss**), the processor accesses main memory for the data. Following a cache miss, 8 bytes are brought in from memory in two successive 32-bit words (the cache uses a 8-byte block size). The portion of the data requested by the 68020 is presented to its data bus, and all 8 bytes of data are copied into the cache. Cache logic determines which set of cache locations to use. This process of adding a new cache entry is called a **cache fill**.

Data in system I/O address space (the lower 128 MBytes at physical addresses 0x0000000 to 0x7FFFFFF) are never cached. Data in the system memory address space (the upper 128 MBytes at physical addresses 0x8000000 to 0xFFFFFFF) may be cached, depending on the setting of the Cache bit (or C bit) in the TB of the data's associated page, as described later.

On 68020 write operations, if the physical address generates a cache hit, the indexed cache location is updated, as well as the copy in main memory. This process is called **write through**

and ensures that the cache is kept current with memory. Writes that do not generate a cache hit only update memory. Table 3-1 summarizes the cache operation.

Table 3-1
Cache Operation

Operation	Hit	Miss
Read	Data in cache	Fill
Write	Write-Through	Update Memory Only

The local CPU is not the only device in the system able to write into main memory. Other SMI devices, such as a MULTIBUS device, a second CPU, or an array processor, may access the same space. Thus, when a given data value residing both in main memory and the local cache is changed, the system must modify both copies. The memory location is always updated, but whether the cache is updated or not depends on what device is writing to the SMI:

- When the local CPU writes to the SMI, both the cache value and the value in main memory are updated.
- When any non-local SMI device (a MULTIBUS device or remote CPU, for example) writes data over the SMI, the corresponding physical address value is loaded into an **invalidation stack**. When the 68020 internal address bus is free, the local invalidation stack logic checks the cache tag stores for any entries matching the addresses in the stack and marks these entries as being invalid. The invalidation logic then relinquishes control of the internal bus to the 68020. This is called the cache invalidation process.

The details of the cache implementation are described in the next section.

3.2 Cache Structure

The structure of the cache on the MC5600/5700 system is shown in Figure 3-1.

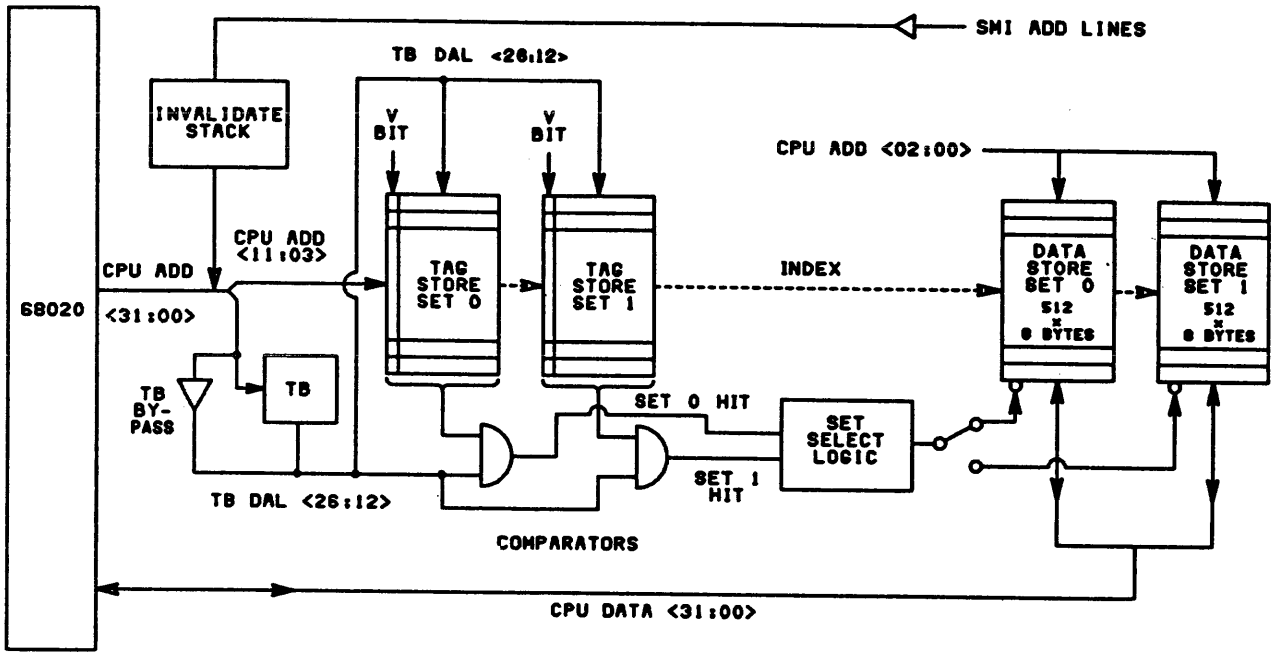


Figure 3-1. The Cache

The cache is implemented with two 512-entry one-way cache sets, called **cache set 0** and **cache set 1**. Each entry in each cache set is comprised of a 64-bit **data store**, a corresponding 15-bit **tag store** and 1-bit **V-bit store**.

Address bits <11:03> from the 68020 are used as the index to select a location in set 0 and in set 1. Address bits <26:12> of the physical address (generated by the translation buffer) are used as the **tag** bits. If the indexed location in the tag store of *either* set contains the same data as bits <26:12> of the physical address and that entry's **V** bit indicates a valid entry, it is a **tag match**. The hardware then checks that:

- The operation is not a Read-Modify-Write cycle
- The address is in the upper (system memory) half of the physical address space
- The C-bit is set in the Translation Buffer entry for that page's translation (see Chapter 2)

If these conditions are all met, it is a **cache hit**, and the data in the appropriate set's data store is immediately sent to the 68020 data bus. Address bits <02:00> are used to select which part of the cached 8-byte data is needed by the 68020.

Since the two cache sets use the same index lines, either set is eligible to contain an entry for a given index. In the case of a cache hit or a write through (where it has been determined that the data has already been cached), the set select logic enables the appropriate set's data store.

In the case of a cache fill, set selection is determined by the current values of the two indexed entries and an internal flip-flop:

- If *one and only one* set's indexed entry has its **V** bit set to **0** (an invalid entry), this location is used to cache the data.
- If the indexed location in *both* sets have their **V** bit set to **1** (valid) or to **0** (invalid), the internal flip flop is used to select the set. This flip flop toggles on every 68020 access to SMI physical memory space (above 0x8000000).

This cache fill logic guarantees that a given physical address cannot be cached in both sets at any one time.

Once the set is selected on a cache fill, bits <26:12> of the physical address are stored in that set's tag store. The cache checks address bit <2> to select the 32-bit portion of the quadword that is requested by the 68020 and puts this data on the internal 68020 bus. The entire quadword is stored in the cache data store at the indexed location. Thus, if the next address reference from the 68020 falls within this quadword (which statistically is often the case), the data is already available in the cache.

Note that the read part of 68020 read-modify-write operations always bypasses the cache logic. In these operations, data is fetched directly from main memory and written back through the cache (that is, the cache is updated if the write generates a cache hit). Also, writes to non-cacheable pages are written through the cache.

3.3 Cache Operating Modes

Two registers control the cache operating modes: **Processor Control Register A (PCRA)** and the **Translation Buffer / Cache Control Register (TBCCR)**. The PCRA contains three bits that control the cache operating mode. The PCRA is located at physical addresses 0x0080000 - 0x009FFFF (in the local CMPU device section of SMI device space) and is fully described in Chapter 4. The TBCCR also contains a control bit that affects cache operation. The TBCCR is located in Processor Register space and is described in Chapter 2. This section describes the PCRA and TBCCR bits that specifically pertain to the operation of the cache.

Table 3-2 shows the operating modes of the cache as set by these four control bits.

Table 3-2
Cache Operating Modes

TBCCR	PCRA			Mode	Description
TBD (L) <0>	CDM (H) <6>	EC1 (L) <5>	EC0 (L) <4>		
X	0	1	1	Cache off	No Read Hits or Fills No Write Throughs No Invalidation
1	0	1	0	S0 Tracking	No Read Hits or Fills Write Through S0 Invalidate S0
1	0	0	1	S1 Tracking	No Read Hits or Fills Write Through S1 Invalidate S1
1	0	0	0	S0 & S1 Tracking	No Read Hits or Fills Write Through S0 & S1 Invalidate S0 & S1
1	1	1	*z*	S0 Diag Mode	Read S0 Data Write S0 Data Update S0 Tag & V Bit on Writes
1	1	0	1	S1 Diag Mode	Read S1 Data Write S1 Data Update S1 Tag & V Bit on Writes
1	1	0	0	S0 & S1 Diag Mode	Read & Write S0 & S1 as specified by Set Select logic No Tag or V Bit updates on Writes
0	0	1	0	S0 On	Read S0 Hits & Fills Write through S0 Invalidate S0
0	0	0	1	S1 On	Read S1 Hits & Fills Write through S1 Invalidate S1
0	0	0	0	S1 & S2 On	Read S0 & S1 Hits & Fills Write through S0 & S1 Invalidate S0 & S1. Set specified by Set Select logic
0	1	*X*	*X*	Prohibited	

X = Either **1** or **0**

The following is a description of the cache operating modes:

- **Cache Off.** When the cache is turned off, all requests for data are brought in from main memory. Hit detection and invalidation functions are not operational.
- **Tracking Mode.** Tracking mode keeps cache tag and data entries valid by writing through, but does not read the cache on hits or fill new entries on cache misses. All accesses are made to and from memory. Either or both sets may be selected for tracking mode.
- **Diagnostic Mode.** Diagnostic mode allows the programmer to read and write the cache data stores directly, effectively treating the cache as a RAM. When S0 or S1 is selected in diagnostic mode, the data store of the selected set is read or written over the 68020 data bus.

  On a write operation, the tag store portion of the physical address is stored in the indexed location and the V bit is set or cleared, depending on the value of the CV bit (bit <7>) in the PCRA (see Chapter 4).

  If both sets are selected, the tag and the V bit are not updated on write operations (as they are when one set is selected). The flip flop toggles the set select on every access into SMI memory space. Thus, writing sequentially through all cache locations with both sets in diagnostic mode results in alternating locations in each set being filled. This feature may be used to test the operation of the flip flop.
- **One-Way Associative Mode.** Either S0 or S1 may be selected (if, for example, a set is determined to be malfunctioning). In this mode, one set is always selected and the other is turned off.
- **Two-Way Associative Mode (normal).** In normal operation, either set may generate a cache hit. Cache fills are put into the first invalid set location or, if neither set's entry is invalid, a flip flop selects the set. All 68020 write operations into SMI physical memory generate write throughs in the processor's own local cache.

3.4 Invalidation

SMI DAL <28> is the Don't Invalidate (DI) line, as described in Chapter 7. All SMI write transaction addresses that are sent with the DI line asserted are entered into the queue for the invalidation process.

In the invalidation process, the SMI physical address is pushed onto the 4-entry cache invalidate stack shown in Figure 3-1. The stack then requests control of the 68020 internal bus. When the 68020 gives up control of its bus, the next address in this stack is put out on 68020 address bits <27:03>. Since the address is physical and needs no translation, the address is sent by the invalidation logic through TB bypass buffers to the TB DAL lines. If this address generates a tag match in either cache set, that set's indexed V bit is cleared. In this case, the 68020 will have to fetch the data from memory on its next access to this address.

If at some point the invalidate stack becomes full, the hardware asserts the invalidate inhibit line on the SMI (IINH), preventing further writes to memory (except from the page table engine) until the stack is cleared of addresses.

SMI write transactions with the **DIF** bit set do not activate the invalidation process. This can be used by the system programmer to save the processing overhead of the invalidation process, which causes the 68020 to stall.

Note that the hardware invalidation implementation does not guarantee that the cache tracks memory updates in real time. The programmer has to carefully coordinate processes using atomic read-modify instructions to assure correct sharing of data. It must be guaranteed that data modified by another processor during a critical section of code will be invalidated in cache before the current CPU gains control of that critical section.

3.5 The C Bit

When the **C** bit in the Translation buffer entry (see Chapter 2) is **0** for a given page, any data accessed from that page is not cached on a write or checked for a cache hit on a read. This is useful if it is not appropriate to cache data on a specific page.

WARNING

> Page Table Entries used to access page tables **must** have their **C** bit cleared. Since the page table engine (described in Chapter 2) does not honor invalidate inhibits on its write operations, cached PTEs can become stale in a multiprocessing system.

3.6 Cache Flushing

Either or both cache sets may be flushed by writing to the Translation Buffer / Cache Flush Register (TBCFR). This register is fully described in Chapter 2 in the processor register space section. Bit <2> of the TBCFR controls cache set 0 and bit <3> controls cache set 1. When the 68020 writes a **1** to a a flush bit in the TBCFR, the corresponding cache set is flushed by clearing the **V** bits of all of its entries.

Chapter 4
CMPU Local Devices

ILLUSTRATIONS

TABLES

Chapter 4
CMPU Local Devices

Five devices on the CMPU module are local to that processor and are not accessible by other processors in the system. These are called **CMPU Local Devices**, and have been collectively assigned their own physical address range, as described in Chapter 8.

When the processor accesses an address in CMPU Local Device space, internal logic decodes this address and selects the appropriate on-board device. These local devices use their own internal byte-wide data path, which is tied to byte 0 of the buffered DAL data path.

The CMPU Local Devices are:

- **EPROM** - This 64 KByte Ultraviolet Erasable Programmable Read Only Memory holds the bootstrap (see Chapter 12) and console code (see Chapter 13).
- **PCRA & PCRB** - Processor Control Registers A & B are two 8-bit registers used for various control and status functions on the CMPU module. These registers are explained in Section 4.1.
- **P0 & P1** - Two Dual Universal Asynchronous Receiver/Transmitters (DUARTs) are used for serial ports for communication with terminals, modems, and the AFM module. These ports are explained in Section 4.2.

The CMPU module also uses a write buffer as a performance enhancement to its SMI interface. This buffer can be enabled or disabled by writing a control bit in the PCRB, but otherwise its operation is transparent to the programmer. The programming considerations of the write buffer are explained in Section 4.3. Figure 4-1 shows the local device section of the CMPU.

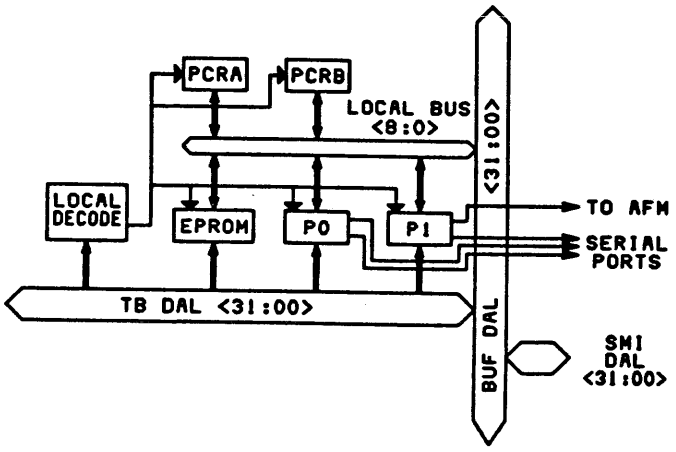


Figure 4-1. CMPU Local Devices

The following sections describe each local device.

4.1 Processor Control Registers

The processor control registers are two 8-bit read-write registers located at physical addresses 0x80000 (PCRA) and 0xA0000 (PCRB).
Figure 4-2 shows the bit fields in each processor control register.

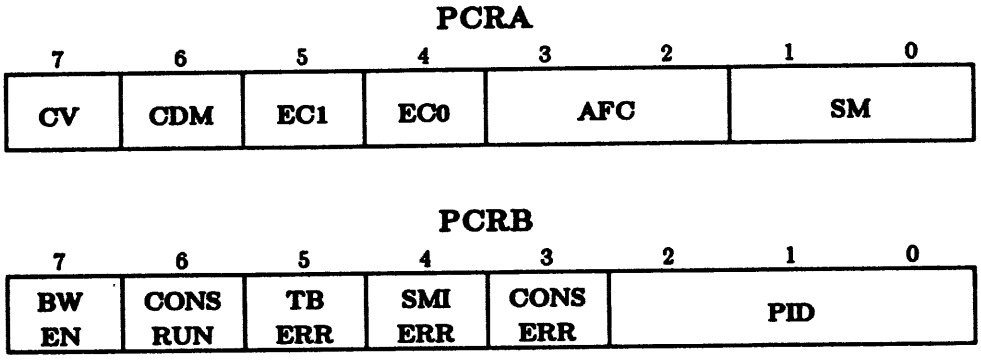


Figure 4-2. PCRA & PCRB Bit Fields

Processor Control Register A contains the following bits:

SM **Space Modify** (R/W). Bits <01:00> are used to select the secondary address spaces (Diagnostic, Processor Register or Translation Buffer space). The use of these spaces and the SM bit decoding are explained in Chapter 2.

AFC **Alternate Function Code** (R/W). Bits <02> and <03> are substituted for the 68020 function code bits **FC1** and **FC2**, respectively, when referencing diagnostic space (see Chapter 2). Table 4-1 shows how these bits are decoded.

Table 4-1
Alternate Function Codes

Bit <3>	Bit <2>	Description
0	0	User Data
0	1	User Program
1	0	Supervisor Data
1	1	Supervisor Program

EC0 **Enable Cache Set 0** (R/W). Clearing bit <4> enables cache set 0 (see Chapter 3).

EC1 **Enable Cache Set 1** (R/W). Clearing bit <5> enables cache set 1 (see Chapter 3).

CDM **Enable Cache Diagnostic Mode** (R/W). Setting bit <6> to a **1** enables the cache to be in diagnostic mode (see Chapter 3).

CV **Cache Valid** (R/W). Bit <7> is used only when the cache is in diagnostic mode (see Chapter 3). When this bit is set, the **V** bit in the cache data store is set on writes into the cache.

Processor Control Register B contains the following bits:

PID **Processor I.D.** (Read Only). Bits <02:00> are set with switch SW1 on the CMPU module (see Chapter 11). Setting the processor I.D. sets up the physical address assignments for that processor's MULTIBUS memory space, I/O space, IPIR, and I/O maps, as described in Chapter 8. Setting the processor I.D. to **1** defines that CMPU as the boot processor in a multiprocessing system. Bit **0** is the LSB of the I.D.

CONS ERR **Console Error** (R/W). Bit <3> controls the (yellow) ERROR LED on the CMPU module. When this bit is cleared, the LED is turned on and MBUS ERROR signal is asserted. Since the MBUS ERROR signal is a wire OR'd (open collector) signal, this signal is asserted when bit <3> in the PCRB on any processor in the system is a **0**. This condition causes the FAULT LED on the front panel to turn on.

SMI ERR **SMI Error** (Read Only). Bit <4> is a **0** when an SMI access by the 68020 receives a NACK or a RETURN ERROR response (see Chapter 7).

TB ERR **TB Error** (Read Only). Bit <5> is a **0** when the page table engine encounters any error (see Chapter 2).

CONS RUN **Console Run** (R/W). Writing to bit <6> controls the (green) run LED on the CMPU module. When a **1** is written to this bit, the LED turns on.

BW EN **Buffered Write Enable** (R/W). When bit <7> is set to a **1** (and memory management is enabled by writing the TBCCR, as explained in Chapter 2), all writes to SMI memory space are buffered by the internal

SMI bus interface logic (see Section 4.3). If a buffered write interrupt is detected, this bit must be cleared to **0** and then set to **1** to clear the interrupt (see Chapter 5).

4.2 Serial Ports

The two DUARTs on the CMPU module provide serial interfaces to:

1. The three RS-232-C ports on the system back panel (P0A, P0B, and P1A) used to communicate with external devices, such as terminals, printers, and modems.
2. The TTL level port used to communicate with devices on the Auxiliary Function Module, including the NVRAM containing bootstrap defaults, customer boot code, system serial number, and time of day clock.

The DUARTs are two Signetics 2681 chips, each of which features two asynchronous serial ports, a programmable baud rate generator and a 16-bit programmable counter/timer. Address bits <03:00> may be used by the programmer when accessing a DUART to use various functions and internal registers on the chip. For more information on the operation of the 2681 chip, see Appendix C.

Port A & B on DUART 0 and Port A on DUART 1 are buffered for standard RS-232-C signal levels. A cable (at CMPU connector **P3**) connects the three ports to the connector distribution board on the system back panel. The fourth port (Port B on DUART 1) is used to communicate with the Auxiliary Function Module and uses unbuffered TTL level signals. A separate cable (at CMPU connector **P4**) ties this port to the AFM. See Chapter 12 for a description of the communication protocol used over this link.

Table 4-2 below gives the pinouts for each of these ports, including the 2681 signal names and the pinouts on both CMPU module connectors and system back panel connectors.

4.3 Buffered Writes

The CMPU module uses a write buffer that enables the 68020 processor to write to SMI memory without incurring a wait state. The SMI interface logic buffers the data and address on 68020 write cycles into SMI memory space. The interface logic asserts DSACK to the 68020 and then handles the SMI protocol involved in completing the transaction. Buffered writes do not occur in write transactions into SMI I/O space.

The buffered write feature is enabled by setting the **BW EN** bit in the PCRB (see Section 4.1). Memory management must be enabled to use the write buffer. This is done by setting the **TBD** bit of the TBCCR (see Chapter 2).

An error generated on the SMI during a buffered write (to a non-existent device, for example) causes a level 7 interrupt (but does not affect the **SMI ERR** bit of the PCRB). Errors incurred when performing non-buffered writes and writes into SMI I/O space by the 68020 cause the 68020 BERR line to be asserted and the **SMI ERR** bit in the PCRB to be cleared.

Table 4-2
Serial Port Pinouts

CMPU Module			External Device			
DUART #	**2681 Signal**	**P3 Pin #**	**Serial Port**	**RS-232-C Signal**	**Connector Pin #**	**UNIX Dev #**
0	TXDA	1	P0A	TxD	2	*tty0*
	RXDA	2		RxD	3	
	OP0	3		RTS	4	
	IP0	4		CTS	5	
	IP4	5		DSR	6	
	GND	6		GND/RTN	7	
	IP2	7		DCD	8	
	OP2	8		DTR	20	
	TXDB	9	P0B	TxD	2	*tty1*
	RXDB	10		RxD	3	
	OP1	11		RTS	4	
	IP1	12		CTS	5	
	IP5	13		DSR	6	
	GND	14		GND/RTN	7	
	IP3	15		DCD	8	
	OP3	16		DTR	20	
1	TXDA	17	P1A	TxD	2	*tty2*
	RXDA	18		RxD	3	
	OP0	19		RTS	4	
	IP0	20		CTS	5	
	IP4	21		DSR	6	
	GND	22		GND/RTN	7	
	IP2	23		DCD	8	
	OP2	24		DTR	20	
DUART #	**2681 Signal**	**P4 Pin #**	**Serial Port**	**TTL Signal**	**AFM Pin #**	**UNIX Dev #**
1	TXDB	2	P1B (AFM)	TxD	2	*clk*
	RXDB	3		RxD	3	
	OP1	4		RTS	4	
	IP1	5		CTS	5	
	IP5	6		DSR	6	
	GND	7		GND/RTN	7	
	IP3	8		DCD	8	
	OP3	9		DTR	9	

Chapter 5
Interrupts and Exceptions

ILLUSTRATIONS

TABLES

Chapter 5
Interrupts and Exceptions

As described in the *MC68020 32 Bit Microprocessor User's Manual*, there are two general types of exceptions that may divert normal program execution:

- **External** - interrupts, bus errors, resets, and coprocessor-detected errors generated by external devices in the system
- **Internal** - traps and other error-returning instructions, address errors, privilege violations, breakpoints, and tracing instructions.

This chapter is concerned primarily with how the system handles external exceptions generated by its various peripheral devices. It also describes how interrupts are handled in a multiprocessor environment, and how bus errors and resets are treated by the system. Internal exceptions and 68020-defined exception vector assignments are described in the 68020 manual.

5.1 External Interrupts

The MC5600/5700 system uses a vectored interrupt scheme. An interrupt (or exception) vector is a pointer to a location in memory that contains the physical address of a routine used to handle an exception produced by a specific situation. There are 7 levels of interrupts, with level 7 having the highest priority, and up to 256 device-assignable interrupt vectors that can be referenced.

The 68020 has an internal register called the Vector Base Register (VBR) whose contents point to the base of the 1 KByte exception vector table in system memory. This table contains the 256 exception vectors used by the system. All exception vectors are one longword in length, except for the RESET vector, which is two longwords in length.

The **vector offset** is the number added to the VBR value to index the exact location in the vector table for an interrupting device handling routine address. A specific vector offset has been assigned to each interrupting device for each interrupt level. The exception to this rule is that all MULTIBUS devices at a given interrupt level are collectively assigned one vector. When a given device generates an interrupt at a given level and the processor enters exception processing, the interrupt vector decode circuitry puts a code corresponding to this unique vector offset (the vector offset divided by 4) onto 68020 data lines <07:00>. The 68020 then adds the offset (the vector code times 4) to the VBR value and fetches the address of the interrupt handler pointed to by the vector. The processor then executes the handler at this address.

The *MC68020 32-Bit Microprocessor User's Manual* has assigned vector offsets internal to the microprocessor. It has also assigned offsets 0x100 - 0x3FC to be used for user-defined interrupts. Within this range, MASSCOMP has defined its own specific vectors. Table 5-1 lists these MASSCOMP-defined interrupts and the corresponding vector offset added to the VBR.

Table 5-1
MASSCOMP Interrupt Vector Assignments

Level	Device	Offset	Device	Offset	Device	Offset	Device	Offset
7	SWI7 or PFI or BWI	3E0	IPI7	3E4	-	3E8	-	3EC
6	MBI6	3C0	IPI6	3C4	DUI6	3C8	DUI6 and IPI6	3CC
5	MBI5	3A0	IPI5	3A4	DUI5	3A8	DUI5 and IPI5	3AC
4	MBI4	380	IPI4	384	-	388	-	38C
3	MBI3	360	IPI3	364	-	368	-	36C
2	MBI2	340	IPI2	344	-	348	-	34C
1	MBI1	320	IPI1	324	-	328	-	32C

The device type abbreviations in Table 5-1 are:

- **BWI** - Buffered Write Interrupt. When the Write buffer write is enabled (see Chapter 4), an error incurred by the SMI bus interface logic generates this interrupt. To clear this interrupt, you must disable and then enable the Write Buffer feature (see the PCRB in Chapter 4).
- **DUI** - DUART Interrupt. DUI5 is DUART 1 and DUI6 is DUART 0.
- **IPI** - Inter-Processor Interrupt. Inter-processor interrupts are described in Section 5.2.
- **MBI** - MULTIBUS Interrupt. Any device on the associated MULTIBUS generates this interrupt. MULTIBUS lines MBUS INT 1-6 generate MBI1 - MBI6 respectively.
- **PFI** - Power Failure Interrupt. This interrupt occurs when MBUS AC LO is asserted by the system's power supply circuitry, indicating that primary power is about to be lost (see Section 5.4 for a description of the AC LO detect circuitry on the AFM). The **ACL** bit of the TBCCR is also cleared when this type of error occurs (see Chapter 2).
- **SWI** - Software Interrupt Switch. This is generated when the INTERRUPT switch on the front panel has been pressed. The **SWR**bit of TBCCR is also cleared when this type of interrupt occurs (see Chapter 2).

Vector addresses with no assigned device (indicated in Table 5-1 by the dash (-) symbol) should never be returned if the CMPU logic is functioning properly. However, if you are writing interrupt handling code, it is good practice to send these vectors to the unexpected interrupt handler.

5.2 Inter-Processor Interrupts

The Inter-Processor Interrupt Register (IPIR) is an internal register on each CMPU module. In a multiprocessor environment, this device allows any processor to interrupt another processor on the SMI. Also, a processor can write to its own IPIR as a means of scheduling software processes.

When a given processor's IPIR has been written, an interrupt to that processor is generated. The level of the interrupt is determined by the new contents of the register. Each processor uses the IPIR vector offsets shown in Table 5-1 (added to the VBR), to fetch the appropriate address of the inter-processor interrupt handler. The interrupt will not be cleared until the appropriate bits in the IPIR are cleared.

The IPIR address space assigned to each processor is described in Chapter 8 and is determined by the processor I.D. set on switch SW1 on each CMPU module. Each processor's IPIR has 8192 redundant locations (the lower 15 address bits of the IPIR are not decoded). The IPIR must be written as a longword operand using the format shown in Figure 5-1.

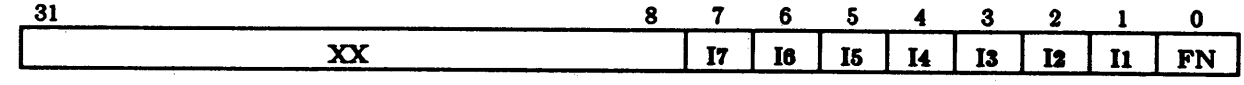


Figure 5-1. IPIR Format

The following are the IPIR field descriptions:

XX = Not Used

I7 = Affect Level **7** IP interrupt

I6 = Affect Level **6** IP interrupt

I5 = Affect Level **5** IP interrupt

I4 = Affect Level **4** IP interrupt

I3 = Affect Level **3** IP interrupt

I2 = Affect Level **2** IP interrupt

I1 = Affect Level **1** IP interrupt

FN = Function (**1** = Sets interrupt; **0** = Clears pending interrupt)

The **FN** bit determines how **I1** through **I7** are interpreted. For example, when **FN** is set to **1**, a **1** in the **I1** position sets a Level 1 inter-processor interrupt. A **0** in the **FN** position and a **1** in the **I1** position clears a pending Level 1 interrupt.

5.3 Bus Errors

Table 5-2 lists the possible causes of bus errors in the MC5600/5700 system, signaled to the 68020 when it is running a bus cycle.

Table 5-2
Bus Error Causes

Type	Description
TB BERR	Page Table engine fault. This error clears the **TB ERR** bit of PCRB (see Chapters 2 & 4).
FPA BERR	Bus Error signal from Floating Point Module
PRE BERR	Parity error or non-existent SMI memory device. This error clears the **SMI ERR** bit of PCRB (see Chapter 4).

As explained in the 68020 Manual, a BERR generates an exception with a vector offset of 0x8.

5.4 Initialization and Reset Circuitry

The Auxiliary Function Module (AFM) contains initialization circuitry (shown in the AFM circuit diagram in Chapter 1) that is used in any of the following circumstances:

- System powerup
- Pressing the RESET switch on the front panel
- Power outage
- Assertion of RESET signal from the boot CMPU's 68020

Since the system bootstrap is invoked in all of these situations, the boot CMPU is given exclusive access to this circuitry. The cable between the boot CMPU module and the AFM has three signals associated with the initialization circuitry:

- **AFM Attach Detect** - This signal is grounded on the AFM. It indicates to the CMPU that an AFM is attached, thus designating it the boot CMPU of a system.
- **BUFF RESET** - This buffered signal drives the 68020 RESET pin on the CMPU and is asserted during a powerup or when the RESET switch on the front panel is pressed.
- **AFM INIT ALL** - This line is driven by the 68020 RESET pin. When the boot CMPU performs a RESET instruction, it causes the AFM initialization circuitry to assert MBUS INIT. This signal is chained to all MULTIBUSs on the system and causes all MULTIBUS devices to be initialized. This signal may also be used to initialize SMI devices. The duration of MBUS INIT L in this case will be 10 milliseconds The AFM INIT ALL line also resets the microcomputer serial interface logic on the AFM card.

The initialization circuit is used during any of the following conditions:

POWER-UP. When the system powers up, the AFM uses an internal RC circuit to delay the assertion of the boot processor's RESET line. This delay is sufficient to assure that the voltage level has reached the minimum for proper circuit operation before the system is reset. After this delay, MBUS INIT and the RESET line to all processors is driven low for 500 ms.

RESET/Power failure. The bidirectional AC LO signal on the MULTIBUS is used both to detect a drop in the power supply voltage and as control signal for the RESET switch on the front panel of the system. The AC input to the power supply must drop below 90 VAC for at least 5 milliseconds to trigger AC LO. The assertion of MBUS AC LO, due to either power loss or the RESET switch being pressed, results in the assertion of the MBUS INIT signal and the RESET signal on all processors for 500 milliseconds after MBUS AC LO has been deasserted. The initial assertion of MBUS INIT is delayed by 5 milliseconds.

68020 RESET instruction. When the 68020 on Processor 1 executes a RESET instruction, the AFM detects the instruction and asserts MBUS INIT for 5 milliseconds. Since the MBUS INIT lines of all non-boot processors are tied together, this causes all non-boot CMPUs and their associated MULTIBUS devices to be reset. There is no effect on the RESET line of the boot CPU. When a non-boot processor executes a RESET instruction, the signal has no effect on the system.

Chapter 6
Memory Module

ILLUSTRATIONS

Chapter 6
Memory Module

The MC5600/5700 Memory Module (CMM) is an SMI device with an array of Dynamic Random Access Memory chips (DRAMs) used for system memory. Each CMM module can be configured to be fully populated (4 MBytes) or partially populated (2 MBytes) using the same physical board. Since there may be more than one processor using the SMI bus, the CMM module may be used as shared memory.

This chapter describes the following CMM features:

- General operation
- Error detection and correction
- Two control/status registers (CSRs), used in conjunction with error detection/correction circuitry and diagnostics
- Initialization requirements
- Interleaving

Configuration information about the CMM module, such as setting interleaving, base address, slot assignments, and size considerations, is given in Chapter 11.

6.1 General Operation

The CMM module is organized with a 32-bit wide data path and 7 additional check bits assigned to each longword of data. These check bits provide the module with single bit error correction and double bit error detection (see Section 6.2). There are 2 or 4 rows (depending on whether the board is fully or partially populated) of 256 KByte x 1 DRAMs, with 32 + 7 chips in each row. Thus, each row constitutes 1 MByte of memory plus the associated check bits.

The CMM handles all SMI cycle types, specifically 8, 16, 24, 32, and 64 bit write accesses and 32 and 64 bit read accesses (see Chapter 7). Memory access time on read operations is two wait states.

Refresh operations on the dynamic RAM chips are performed by the hardware and are transparent to the software.

The first CMM module must have a starting address of 0x8000000. Chapter 11 describes how to set the module address and how to add modules.

6.2 Error Handling

The CMM module is designed to correct all single bit errors and detect all double bit errors. The error correction logic is built around two AMD **Error Detection and Correction Unit (EDC)** chips used on each module. Error correction is based on a 32-bit storage unit, requiring 7 check bits to support the single-error correction and double-error detection.

On read operations, a single bit error is reported by logging the appropriate information in the Memory Control Register (see Section 6.3) and by asserting an interrupt line on the SMI, if the interrupt is enabled. When a double bit error is detected during a 32-bit read, a RETURN ERROR command is transmitted (rather than RETURN DATA) on the SMI. The corrupted data is still put onto SMI bus. When a double bit error is detected in the first word of a 64-bit read, a RETURN ERROR command is transmitted for both words. When a double bit error is detected in the second word of a 64-bit read, valid data and a RETURN LONGWORD command is transmitted with the first word and RETURN ERROR is transmitted with the (corrupted) second word.

Writes that are either 8, 16, or 24 bits require that the memory module do an internal read, followed by a merge of the new data, and finally a 32-bit write back to the memory location (also called **read-and-merge**). For longword and quadword write commands, this internal read-and-merge cycle is not required. Errors that occur during the read of a 68020 read-modify-write operation are not logged. When a single bit error occurs during the read portion, a correction is performed before the write. An uncorrectable error causes the write to be aborted and the contents of memory to be left unchanged. No interrupt is generated by the aborted write cycle. Any subsequent read of this location will detect the uncorrectable error.

For further information on the EDC chip, see the data book referenced in the preface of this manual.

6.3 Module CSRs

Two control/status registers (CSRs) are used on the CMM module that are internal to the Error Correction and Detection chip: the Substitute Check Bits Register (SCBR) and the Memory Control Register (MCR). The SCBR is a write-only register and the MCR is a read/write register. Both CSRs only respond properly to longword accesses, and read or write operations of byte, word, or quadword data sizes produces unpredictable results. The physical address of each register depends directly on the module's base address (setting the CMM base address is explained in Chapter 11). The bit fields in each CSR are shown in Figure 6-1.

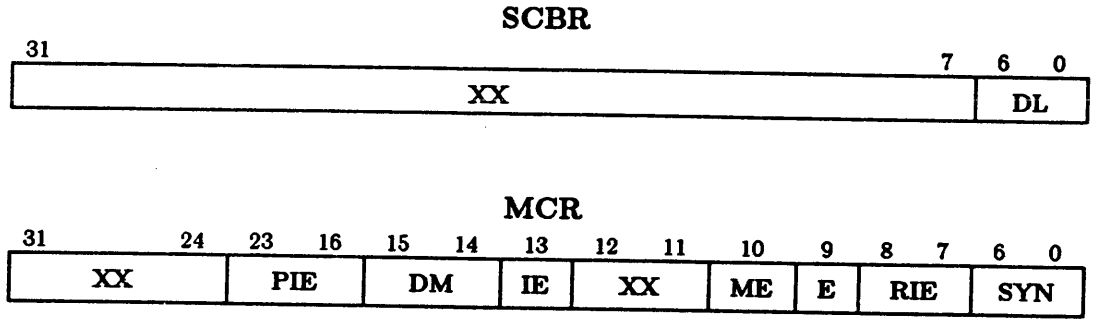


Figure 6-1. SCBR & MCR Bit Field Formats

The following are the field descriptions for the two CSRs:

XX **Not Used**

DL **Diagnostic Latch.** Bits <6:0> correspond to bits <06:00> of the diagnostic latch in the AMD 2960 chip (see the chip specification). These bits are write-only.

DM **Diagnostic Mode.** When address bit <19> is **1**, MCR bits <14> and <15> are loaded as is into bits **DM0** and **DM1** on the AMD 2960 chip, respectively. When SMI address bit <19> is **0**, the code **00** (normal operation) is substituted into the AMD 2960 chip for a **00, 01**, or **10** MCR code. Refer to the AMD catalog referenced in the preface of this manual for more information on how the AMD 2960 chip interprets these codes.

E **Error.** Bit <9> is cleared on any error, correctable or uncorrectable. Once this bit is cleared, further clocking of bits <23:16> and <10:00> is inhibited. This ensures that the status of the first error is always held until the software can read it, even in the event of multiple errors. The bit is set to a **1** on any write to the MCR.

IE **Interrupt Enable.** When bit <13> is set to a **1**, the memory generates a level 6 MULTIBUS interrupt for all errors that occur during 32 and 64 bit reads. Errors during read-and-merge cycles are not posted. Interrupts are inhibited when this bit is set to a **0**.

ME **Multiple (Uncorrectable) Error.** Bit <10> is cleared when an uncorrectable error occurs. The bit is set to a **1** on any write to the MCR. This bit is loaded using the same rules as the **Error** bit.

PIE **Page In Error.** Bits <23:16> contain the value of SMI address bits <19:12> at the time of the first EDC error. These bits define the physical page containing the error and are typically used for error logging by the operating system. These bits are loaded using the same rules as the **Error** bit.

RIE **Row In Error.** Bits <8:7> contain the value of SMI address bits <21:20> (non-interleaved) or <22:21> (interleaved) at the time of an error. These bits give the binary value of the row number that experienced the error. These bits are loaded

using the same rules as the **Error** bit.

SYN **Syndrome**. Bits <6:0> correspond to the AMD 2960 syndrome bits **16, 8,4,2,1,0,** and **X**, respectively (refer to the AMD 2960 specification referenced in preface of this manual). These bits are loaded using the same rules as the **Error** bit.

6.4 Initialization

This section is for those writing their own bootstrap code. At the time of powerup, the check words and CSRs contain random **0**s and **1**s. Before normal use of the memory can begin, the powerup bootstrap must load good check bits and initialize both CSRs. This is done by writing longwords or quadwords of any pattern (the MASSCOMP EPROM bootstrap code writes all zeros) to all CMM memory locations with MCR bits <15:14> = **0**.

6.4.1 Interleaving

Interleaving is a memory configuration technique for improving memory system throughput that allows the module to match the SMI bandwidth. A single memory module is capable of reaching only half the bandwidth of the SMI bus, due to data fetch delays and cycle time. By interleaving modules, one module is presenting data on the bus while the other is fetching data, thus achieving maximum use of the bus bandwidth.

In an interleaved configuration, two memory modules of equal size are given the same base addresses on the SMI bus. The SMI signal ADR <03> is used as the module select line and is swapped with SMI address line <27>, the highest physical address bit. This causes each consecutive quadword address to be physically assigned an alternate module, effectively interleaving the entire available memory space.

Chapter 11 describes how to configure memory modules for interleaving.

Chapter 7
Synchronous Memory Interconnect (SMI) Bus

ILLUSTRATIONS

TABLES

Chapter 7
Synchronous Memory Interconnect (SMI) Bus

The MC5600/5700 Synchronous Memory Interconnect (SMI) bus is a high-speed bus designed for maximum data transfer bandwidth. As described in Chapter 1, the SMI is the central bus connecting all system memory, processors, MULTIBUS Adaptors (described in Chapter 10), and future high-speed devices, such as array processors. Each processor on an SMI bus can directly access any other processor's MULTIBUS Adaptor. A device on a MULTIBUS, by contrast, can directly interface with only one MULTIBUS Adaptor, and cannot access devices on other MULTIBUSs within the system.

The SMI bus uses a total of 50 signals. The data and address lines are multiplexed on 32 tri-state lines. A **bus cycle** is defined as one clock period of 100 nanoseconds. The bus uses two bus cycles for each 32-bit transfer (one address and one data cycle) and three bus cycles for each 64-bit transfer (one address and two data cycles).

This chapter describes the following SMI features:

- Bus Protocol
- Signal descriptions
- SMI Commands
- SMI Address control bits
- SMI Arbitration

Note that this chapter gives only an overview of the SMI bus and is not intended as a design specification. It is provided only as an aid to understanding its role within the entire MC5600/5700 system architecture.

7.1 Split Transaction Protocol

The SMI uses a communication algorithm called **split transaction protocol** to arbitrate between devices on the bus. Any device on the SMI that requires a transfer of data (a CPU, for example) broadcasts a specific command over the bus and then releases the bus. At some later time, the device responding to the command (a memory module, for example) may then itself gain control of the bus in order to complete its part of the transaction. Thus, *all* devices on the SMI are eligible to control the bus at any point in time.

This protocol differs from an **interlocked protocol**, where only certain devices, such as the CPU module, can control the bus and must control the bus at all times during a given transaction. Split transaction protocol allows a higher bus bandwidth than interlocked protocol, since no bus cycles are wasted holding the bus while the slave device fetches its data. This means that this idle bus time is freed up and may be used by other devices that are waiting for the use of the bus, making bus traffic more efficient. This type of protocol is ideal for a tightly-coupled multiprocessor system such as the MC5600/5700, where numerous CPUs must share use of memory resources.

The following definitions are used throughout this chapter:

- A **bus transmitter** is any bus device that is controlling the bus (as granted by the arbitration logic on the AFM described in Section 7.5). In split transaction protocol, *any* device on the SMI can become bus transmitter during normal operation .
- A **bus receiver** is any bus device not in control of the bus that is able to respond to the bus transmitter's command.
- A **bus master** is any bus transmitter issuing a command requesting a transfer of data.
- A **bus slave** is defined as any device that responds to a bus master's command. A slave may or may not need to become bus transmitter to complete a transfer, depending on the type of transfer.

A transaction or data transfer is typically made up of two steps: a master initiates a command, and the slave responds to the command.

When a bus master device gains control of the bus to initiate a data transfer, it transmits a command, a self-identifier (called the **Node I.D.**) and an address over the bus. The master then expects an acknowledgement to be returned on the next cycle indicating that the command was accepted by a bus slave. In the case of a write cycle, this acknowledgement ends the transaction.

As soon as the responding slave device is ready with the data (in the case of a read cycle), the slave requests permission for control of the bus. When the slave has become bus transmitter per the SMI arbitration requirements, it transmits a command, the node I.D. of the master to which it is responding, and the accompanying data in one bus cycle (or, in the case of quadword data, in two cycles).

7.2 SMI Signals

Table 7-1 gives a summary of the SMI signals. All signals are active low. The SMI pinouts are given in Appendix A.

Table 7-1
SMI Signal Descriptions

Group	Mnemonic	Description	# bits
Data/Addr	DAL *xx*	Data / Address Lines	32
Command	CMD *x*	SMI Command	3
	NID *x*	Node Identification	5
	ACK *x*	Acknowledge	2
Arbitration	REQ	Bus Request	1
	GNT	Bus Grant	1
	BUSY	Bus Busy	1
Inhibits	MINH	Memory Inhibit	1
	LINH	Lock Inhibit	1
	IINH	Invalidate Inhibit	1
Clocks	SCLK	System Clock	1
	ECLK	Enable Clock	1

x = a decimal digit

The following describes the SMI signals:

DAL *xx* **Data / Address Lines (32).** These lines multiplex the 32 address bits and 32 data bits during a bus transfer. Thus, at least two cycles are required per transfer.

CMD *x* **SMI Command (3).** The command lines are used to indicate the type of operation and, in some cases, the size of the operand transferred. The commands are explained in Section 7.3.

NID *x* **Node I.D. (5).** The node I.D. is broadcast by master devices initiating read requests or writes. The slave transmits the master's node I.D. along with a Return Longword, Error, or I/O Write complete command (see Section 7.3) to complete the request. Only bus masters are assigned node I.D.s. The node I.D. for both the CPU and the MBA are set on the CMPU module (see Chapter 11) by setting its processor I.D. The logical range of node values is **0** to **31**.

ACK *x* **Acknowledge (2).** The ACK signals are driven by slave devices to indicate to a bus master device that its command has been accepted. There are four possible acknowledge responses to a given command:

- **NACK** (No Acknowledge) - No device responded within the allotted time
- **UACK** (Write Unconditionally Accepted) - This code indicates that a device is responding to a write and no I/O Write Complete cycle (described later) is to follow. All devices in system memory space respond to write commands with this code.
- **CACK** (Write Conditionally Accepted / Read Request Accepted) - This has one of two interpretations. If sent in response to a Read Request, this code indicates that the slave device has accepted the

request and a Return Longword or Quadword or Return Error command is to follow. If sent in response to a Write Request to a slave device in system I/O space, this code indicates that the slave has conditionally accepted the command, and an I/O Write Complete or Return Error command (described later) is to follow.

- **RETRY** (Retry Read/Write (I/O Space only)) - The responding device (usually on the MULTIBUS) is busy and the initiator should retry the transaction. Note that no retries are permitted for devices in SMI memory space.

REQ **Bus Request (1).** The bus master drives this line to request control of the bus. Arbitration logic on the Auxiliary Function Module (AFM) (see Section 7.5) can field up to 30 discrete request lines for bus control.

GNT **Bus Grant (1).** The AFM drives this line to grant control of the bus to a device that has asserted the REQ line. Arbitration on the AFM can grant control of the bus to one of a maximum of 30 devices (see Section 7.5). Note MBUS BUSY must be deasserted as a second condition for becoming bus transmitter.

BUSY **Bus Busy (1).** The BUSY signal is asserted by any master during a multiple cycle transaction and indicates whether the SMI will be busy on the next cycle. In multiple cycle transactions, the BUSY signal is asserted on all but the last cycle. In single cycle transactions, BUSY is not asserted.

MINH **Memory Inhibit (1).** This signal is asserted by a memory device to prevent its command buffer from being overwritten. This signal is unconditional and is honored by all bus masters.

LINH **Lock Inhibit (1).** This signal is asserted by masters who wish to prevent other masters from performing operations with LINH asserted (called **locked** operations). The master must first win control of the bus through the usual arbitration and honoring any other LINH that is asserted. After winning the bus and asserting LINH, the bus master is guaranteed that other devices that honor this line will not use the bus for locked operations as long as it keeps LINH asserted.

The CMPU module asserts LINH while performing read-modify-write cycles, ensuring these operations are atomic by preventing other CMPUs from doing so concurrently.

MULTIBUS devices that generate MBUS LOCK with a transfer to the SMI cause the MBA to assert LINH, as long as LINH is not asserted on the bus by another master. As long as MBUS BUSY is asserted, MBUS LOCK controls SMI LINH. The MBA logic guarantees that SMI LINH remains asserted to the end of a transaction, even if the locked transaction on the MULTIBUS completes early. Local MULTIBUS transactions (between devices on the same MULTIBUS) do not pass MBUS LOCK on to the SMI LINH line.

IINH **Invalidate Inhibit (1).** IINH is asserted by the invalidate sequencer when the cache invalidate stack is about to become full (see Chapter 3). This signal is honored by all bus masters writing to SMI memory space, and ignored by transfers to system I/O space. Note that the CMPU page table engine (described in Chapter 2) also ignores this signal, since Page Table Entries are, by convention, not cached.

SCLK **System Clock (1).** This signal is generated on the Auxiliary Function Module. SMI devices clock and enable their components with the falling edge of this clock. Note the 68020 does not use the SCLK signal, but has its own dedicated clock. The MULTIBUS clock signal BCLK is derived from and in phase with the same 20 MHz oscillator used by SMI SCLK.

ECLK **Enable Clock (1).** The enable clock is used exclusively to enable the SMI data path. The ECLK signal deasserts for approximately 20 nanoseconds as SMI SCLK is being asserted.

7.3 SMI Commands

This section describes the SMI commands. These commands are sent over SMI lines **CMD** <2:0> by the bus transmitter during a transmission cycle. This section is meant only to briefly describe these SMI terms as referenced in other chapters.

Table 7-2 summarizes the SMI commands.

Table 7-2
SMI Commands

Command	Operation
NOP	No operation / Stall
RERR	Return Error
RTNL	Return Longword *or* I/O Write Longword Complete
RTNW	Return Word Data *or* I/O Write Word Complete (SMI to MULTIBUS only)
RR	Read Request (8,16,32, or 64 bits)
WR	Write (8,16,24,32, or 64 bits)

The following are brief descriptions of the SMI commands:

NOP **No Operation / Stall.** The SMI is idle or a bus transmitter is stalled (a memory has detected an EDC error, for example, and needs time to correct the error)

RERR **Return Error.** The transmitting slave device has detected an error in the course of carrying out the bus master's command. The type of error involved depends on the device.

RTNL **Return Longword / I/O Write Complete.** This command is transmitted by a slave responding to a read request or a write to I/O space.

If the transmitting device is responding to a read request, the command indicates to the master-receiver (in this case, the device at the Node I.D.) that the accompanying data is the requested size. The CYC bit (described later) in the SMI address

determines whether the data is 32 or 64 bits.

If the slave is responding to an I/O space write, the command indicates to the master that the transmitter (the MULTIBUS Adaptor, for example) has completed the data transfer that it previously conditionally acknowledged.

RTNW **Return Word / I/O Write Complete.** This command is transmitted by a bus slave responding to a read request for more data than the slave can return in one cycle. The command indicates to the master-receiver (in this case, the device at the node I.D.) that the accompanying data is one word.

If the slave is responding to an I/O space write, the command indicates to the master that the transmitter (the MULTIBUS Adaptor) could not accept all of the data but did successfully write the lower 16 bits. The Return Longword and Return Word commands support dynamic bus sizing of MULTIBUS memory and I/O space for words versus longwords.

RR **Read Request.** This command initiates a read cycle. The command requests data from the slave device at the accompanying address.

WR **Write Request.** This command initiates a write transaction to the slave device at the accompanying address.

7.4 SMI Address Field

The SMI bus uses the lower 28 bits of the 32 DAL lines during its address cycle to specify the physical address. The remaining 4 bits are control bits, as shown in Figure 7-1.

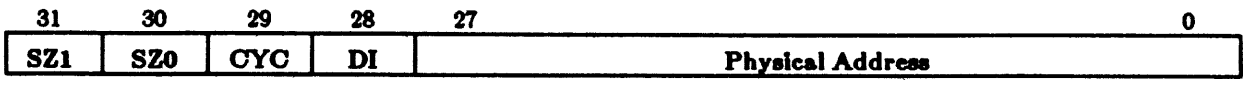


Figure 7-1. SMI Address Field

All 28 bits of each physical address are transmitted on DAL <27:00> during read requests and the address cycle portion of writes. During 64-bit read transactions, DAL <02> specifies which longword of the longword pair to send first. This allows the 68020 request to be satisfied with the first longword while the cache is filled with the next longword.

The address field control bits are explained in the following sections.

7.4.1 The CYCLE code

SMI DAL line <29> is defined as the **CYC** code. This code defines the number of data cycles (in addition to the address cycle) that are associated with the accompanying transaction. When CYC = **0**, one data cycle (32 bits or less) is used. When CYC = **1**, two data cycles (64 bits) are used.

In write transactions, the data cycle(s) follow immediately after the address cycle. In read transactions, the data cycles are associated with the Return Longword transaction. In both Read and Write Requests, the **CYC** bit specifies whether one or two cycles of data is being transferred.

7.4.2 The Size Code

SMI DAL lines <31:30> are defined as the size code field, or **SZ** <1:0> During write transactions with one data cycle, this field specifies which of the four associated data bytes is to be written. The size code and the address allow for 8, 16, 24, and 32 bit transfers. Data must be aligned on word boundaries for 16 and 32 bit writes to MULTIBUS devices. The Size Code is valid only if the CYC bit is **0**. In transactions where the CYC bit is **1**, SZ1 and SZ0 are ignored.

The following data size rules apply to the SMI bus:

- For read transactions in system memory space, transfers may be 32 or 64 bits.
- For read transactions in system I/O space when the master is an SMI device, transfers are either 8, 16, or 32 bits, as specified by the Size Code. The MBA can also use the RTN DATA or RTN WORD command to define size. An I/O port that is only 16 bits wide (the MULTIBUS, for example) accepts for writes and returns for reads only the lower word. It notifies the master of this with the Return Word command.
- For read transactions in I/O Space when the master is a MULTIBUS device, transfers may be 8, 16, or 32 bits.
- For writes to memory space, the data size may be 8, 16, 24, 32, or 64 bits. For the first four of these data sizes, the CYC bit is **0** and the Size Code and the address determine which of the four is selected.
- For writes to I/O space with the master on the SMI, the size may be 8, 16, or 32 bits. Data sizes of 16 and 32 bits must be on word-aligned address boundaries.

All devices in I/O space can transmit an accompanying Return Longword or Return Word I/O Write Complete command. This frees devices on the SMI from having to buffer commands. If a slave in I/O space is able to handle only 8 and 16 bit writes, the size code must be correct (with the exception of the MBA). If the master is on the MULTIBUS, the size may be 8, 16, or 32 bits, with the latter two requiring word and longword aligned address boundaries, respectively.

For non-MULTIBUS I/O writes, the restrictions are again device dependent with software and hardware control.

7.4.3 The Don't Invalidate Bit

SMI DAL <28> is the DI (Don't Invalidate) signal. All SMI write transactions into memory space with this accompanying bit cleared are passed through the cache invalidation process on all CMPUs (see Chapter 3). The operating system controls invalidation via the DIF bit in the second level page table and in the associated TB entries (see Chapter 2). SMI write transactions with the DIF bit set do not activate the invalidation process. This feature allows 68020 cycles normally stolen by the cache invalidation process to be minimized by allowing invalidation to be bypassed.

7.5 Bus Arbitration

The AFM arbitrates the SMI bus traffic, granting the use of the bus to the highest priority request. The MC5600/5700 system uses a parallel priority scheme for SMI arbitration. A module's arbitration level depends on the physical slot in which the module resides. Arbitration levels are given in Chapter 11.

In parallel arbitration, each module on each bus has one request line and one grant line associated with it. All request and grant lines enter the arbitration circuitry of the AFM. The AFM grants control of the bus to the device with the highest priority level request. When this request is removed, the next highest priority request is granted.

The AFM waits for SMI BUSY to deassert before it asserts SMI GNT to give the next level priority module mastership of the bus.

Chapter 8
Physical Address Structure

ILLUSTRATIONS

Chapter 8
Physical Address Structure

The physical address space of a system refers to the assignment of devices, registers, and memory to unique addresses on the system's primary bus. As described in Chapter 1, the MC5600/5700 system uses the Synchronous Memory Interconnect (SMI) bus as its primary high-speed bus to tie together its CPUs, memory, and MULTIBUS Adaptors. The address portion of the SMI bus is 28 bits wide, allowing a total of 256 MBytes that are physically addressable on the bus.

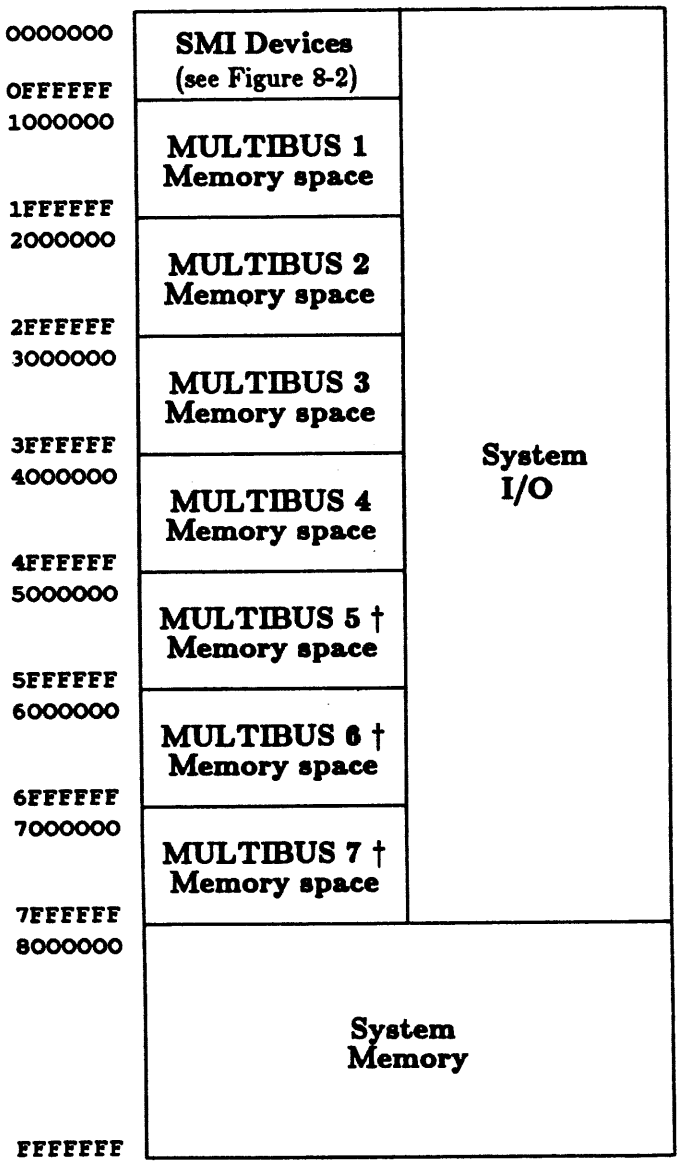
The 256 MByte SMI address space is divided into sections that are allocated to specific groups of devices (all the devices on a MULTIBUS, for example). Since there are no control signals on the SMI that specify separate memory and I/O address spaces as there are on the MULTIBUS, the SMI physical address space contains both types of devices.

Note that each address range may or may not have actual physical devices assigned to every address. For example, there may be only eight 1-byte registers used on a MULTIBUS that has been assigned 16 MBytes of address space.

The SMI physical address space provides distinct addressing ranges for:

- System memory devices (CMM Modules)
- All future SMI modules (array processors, floating point processors)
- Memory space for each MULTIBUS
- I/O space and the I/O map for each MULTIBUS
- The Inter-Processor Interrupt Register (IPIR) associated with each CMPU
- Local devices on the CMPU board (control/status registers and the EPROM)
- Control/Status Registers on the CMM memory module

Figure 8-1 shows how the entire MC5600/5700 SMI physical address space is divided.



* † A maximum of 4 MULTIBUSs are allowed on an MC5600/5700 system

Figure 8-1. The SMI Physical Address Space

The higher half (128 MBytes) of the physical address space is assigned to the **System Memory Space**. Main memory storage devices, such as the CMM module, are located in this part of the address space, starting at location 0x8000000.

The lower half of the physical address space is assigned to **System I/O space**. The System I/O space is further subdivided into seven 16 MByte **MULTIBUS memory** address spaces and one 16 MByte **SMI device** address space that includes address ranges for seven **MULTIBUS I/O spaces, Inter-Processor Interrupt Registers (IPIRs), I/O maps, CMPU Local Devices**, and also for MASSCOMP internal use.

The following sections discuss each of these address space subdivisions.

8.1 System Memory Space

The highest 128 MBytes of physical address space are assigned to **System Memory**, as shown in Figure 8-1. These addresses are occupied by the CMM module(s) on the system (setting the CMM base address is described in Chapter 11). System memory device addresses must start at 0x8000000 and be populated contiguously through the highest address that is used (0xFFFFFFF if all of the space is used). There can be no gaps within the system memory address range.

8.2 MULTIBUS Memory Space

Seven 16 MByte address spaces are reserved for **MULTIBUS memory**, as shown in Figure 8-1. Each address range accesses the memory space on the MULTIBUS of the associated processor. A MULTIBUS memory strobe is generated with each access to this space.

While MASSCOMP supports a maximum of 4 MULTIBUSs on a system, any of the 16 MByte regions not used as MULTIBUS space may be used for raster memory, array processor memory, or other similar uses. If you use this space as MULTIBUS memory space, you can access the entire 16 MByte space without mapping it.

8.3 SMI Device Space

The lowest 16 MBytes of the physical address space (the top section in Figure 8-1) is assigned to **SMI Device addresses**. This is a general category for miscellaneous devices. This group of addresses is subdivided into sixteen 1 MByte address ranges, as shown in Figure 8-2. The SMI device space contains addresses for the following devices:

CMPU local devices - This space is used for devices that reside locally on the CMPU module, such as serial ports and CPU control/status registers. See Section 8.3.1

Memory Module CSRs - This space is used for two control / status registers on the CMM module. These are described in Chapter 6.

Reserved - This 7 x 1 MBtye address range is reserved for future MASSCOMP SMI products.

Customer Devices (6 x 1 MByte) - These six 1 MBtye address ranges may be used for customer-designed SMI products.

MULTIBUS I/O space, I/O Maps and IPIRs - This space is subdivided into the I/O space, I/O map and IPIR for each processor. See Section 8.3.2.

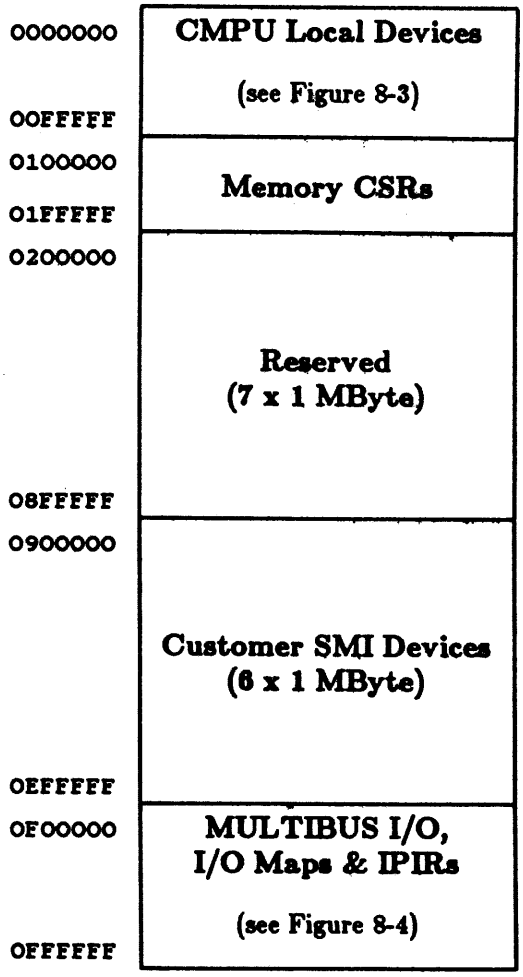


Figure 8-2. SMI Device Space

8.3.1 CMPU Local Devices

A number of addressable devices reside locally on the CMPU module. These as a group have been assigned to the highest 1 MByte address range within the SMI device space that is shown in Figure 8-2. Each CMPU device has been allocated 128K of this 1 MByte address space, as shown in Figure 8-3.

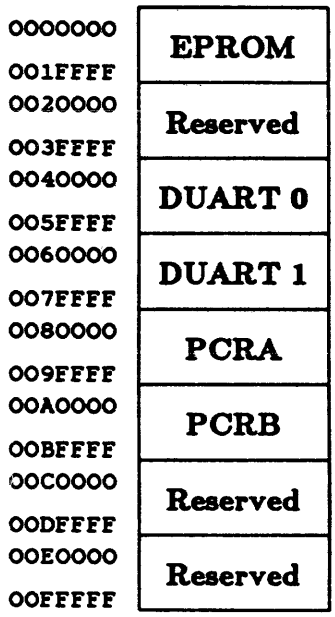


Figure 8-3. CMPU Local Devices

The following are the CMPU Local devices:

EPROM - The Erasable Programmable Read Only Memory stores the Console program (described in Chapter 13) and the system bootstrap program (described in Chapter 12).

DUART 0 & 1 - These are the serial ports on the CMPU used for communicating with the terminals, printers, modems, and the AFM. They are described in Chapter 4.

PCRA & PCRB - Processor Control Registers A & B are 8-bit control/status registers for the Cache, the Translation Buffer and other CPU-related activities. These registers are described in Chapters 2, 3, and 4.

Reserved - This space is reserved for future use by MASSCOMP.

8.3.2 IPIRs and MULTIBUS I/O

One MByte of the SMI Device address range shown in Figure 8-2 is subdivided into spaces for the I/O Map, IPIR and MULTIBUS I/O space for each processor I.D. Figure 8-4 shows how this section of the SMI Device space (0x0F00000 to 0x0FFFFFF) is divided.

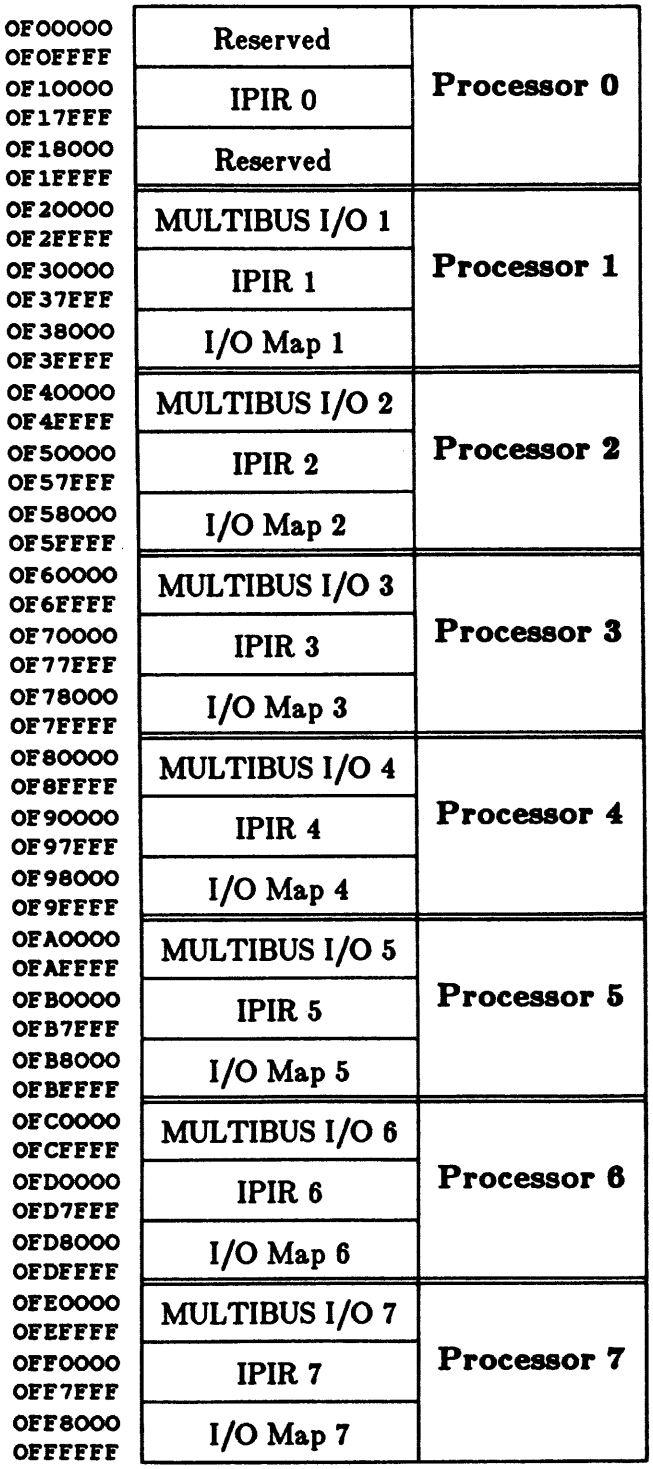


Figure 8-4. I/O Maps, IPIRs, & MULTIBUS I/O Space

Note that only those processors with their corresponding MULTIBUS Adaptor enabled actually use their assigned space. Processor I.D. **0** and those processors with a disabled MULTIBUS Adaptor use only the IPIR section of the space, while the corresponding I/O Map and I/O space becomes unusable. The subdivisions for each processor are:

I/O Map - The I/O map is a translation table on MULTIBUS Adaptor portion of the CMPU module containing 4096 entries. The I/O map is used to translate MULTIBUS-generated addresses to SMI physical addresses, as described in Chapter 10. This address space is used for reading and writing the I/O map entries directly.

MULTIBUS I/O - Seven spaces are reserved for MULTIBUS I/O (MASSCOMP supports a maximum of only four MULTIBUSs). Each MULTIBUS has access to 64 KBytes of I/O address space. A MULTIBUS I/O strobe is generated with each access to this space. This space is generally assigned to control and status registers on MULTIBUS devices. There is no I/O space allocated for Processor I.D. **0**, since that processor is not, by definition, allowed an associated MULTIBUS.

IPIR - The Inter-Processor Interrupt Register is a register on the CMPU module used to coordinate processing tasks in a multiprocessing environment. The IPIR for each processor responds to 4096 redundant locations (the lower 15 address lines are not decoded). The IPIR is described in Chapter 5.

Chapter 9
The MULTIBUS

ILLUSTRATIONS

TABLES

Chapter 9
The MULTIBUS

The MASSCOMP MULTIBUS provides a general purpose I/O bus for the MC5600/MC5700 System. The bus conforms to the industry standard IEEE-796 MULTIBUS, with MASSCOMP enhancements that allow up to a 6 MByte/second transfer rate (compared to the standard 2.5 to 3.5 MByte/second). The MASSCOMP implementation of MULTIBUS is a fully compatible superset of the IEEE-796 standard.

A variety of MULTIBUS processors and controllers are available to support specific I/O devices, such as disk drives, graphics display terminals, and standard communications protocols such as RS-232, Ethernet, and X.25.

MASSCOMP has added a dedicated signal to its MULTIBUS that enables 32-bit transfers using a protocol called **Block Mode**. This feature is designed to support high-performance DMA devices, such as MASSCOMP's Data Acquisition / Control Processor (DACP). When using Block Mode, a DACP, for example, transfers two consecutive words to memory for each address and command strobe asserted on the MULTIBUS . At the rate of 2 MBytes/second for DACP transfers, this mode uses only 35% of the available bus bandwidth. A disk controller with Block Mode capability and a 2 MByte/second transfer rate could store this data in real-time and still leave 30% of the MULTIBUS bandwidth available to the rest of the system.

The interface between the MULTIBUS and the SMI bus is called the **MULTIBUS Adaptor (MBA)** and is explained in Chapter 10. Each MULTIBUS in the system has available two independent address spaces, as described in Chapter 8: a 64 KByte I/O space and a 16 MByte memory space. The MBA handles all address translations between the SMI and the MULTIBUS. The MBA, as part of the CMPU module, also passes all MULTIBUS interrupt lines directly to the CPU. One MULTIBUS is allowed per CPU/MBA, with a maximum per system of 2 MULTIBUSs in an MC5600 and 4 MULTIBUSs in an MC5700.

This chapter covers the following MULTIBUS topics:

- Conformance to the MULTIBUS standard
- Non-supported MULTIBUS options
- MASSCOMP MULTIBUS enhancements
- Protocol and Timing specifications for Block Mode, the MASSCOMP MULTIBUS 32-bit transfer protocol
- Bus Arbitration

The MULTIBUS standard specification referenced throughout this chapter is the Intel publication *MULTIBUS I Architecture Reference Book*, copyright 1983.

9.1 Conformance to the MULTIBUS Standard

The MC5600/5700 CMPU Module is a bus master of the following type:

Master D16 M24 I16 V0 L

as defined in the chapter on Levels of Compliance in the MULTIBUS specification. This compliance designation means that the MC5600/5700 MBA and backplane support 8- and 16-bit data paths, 24-bit addressing, and non-bus-vectored, level-triggered interrupts. Note that the CPU converts MULTIBUS nonvectored interrupts to vectored interrupts (see Chapter 5).

Table 9-1 summarizes the MASSCOMP MULTIBUS signals that deviate from those of the standard MULTIBUS.

Table 9-1
MASSCOMP-specific MULTIBUS Signals

Type	Pin	Standard Signal	MASSCOMP Signal	Actual Usage
Power Supplies	9-10	Reserved	-5 V	Bussed supply
Bus Control	16	BPRO	-	Not supported
	24	INH1	SW RESET	INTERRUPT switch detect
	26	INH2	MC ERROR	System Error Light
	29	CBRQ	MC CBRQ	Slot 1: Private Slot 2-15: Bussed,Grounded
	33	INTA	-	Not supported
Interrupts	36	INT7	MC LOCK2	Identical to MBUS LOCK
	41	INT0	BLOCK MODE	Block Mode control
Power Supplies	77	Reserved	MC Reserved	Reserved
	78	Reserved	AC LO	AC Low / RESET detect

The MBUS signal mnemonics mentioned here and elsewhere are MASSCOMP names for the corresponding MULTIBUS signal mnemonics. All MULTIBUS signals are active low unless otherwise indicated with an **H** suffix.

The MULTIBUS signals are coupled to the CMPU Module at Connectors P1 and P2. The P1 Connector and 4 signals on the P2 connector of the standard MULTIBUS specification are used for the MASSCOMP MULTIBUS signals. The remaining P2 connector signals are used by MASSCOMP for the SMI bus signals (see Chapter 7). The complete list of MULTIBUS signal pin assignments at Connectors P1 and P2 is given in Appendix A.

The MULTIBUS clock line signals, MBUS CCLK and MBUS BCLK, are both generated by a crystal-controlled oscillator circuit on the AFM Module. The clock frequency for both of these signals is:

$$f = 10 \text{ MHz}$$

Note that the BCLK and CCLK are 180 degrees out of phase.

9.1.1 Non-supported Features

The following standard MULTIBUS features are not supported:

- **Vectored Interrupts** - The MC5600/5700 supports only non-vectored interrupts on the MASSCOMP MULTIBUS. The MULTIBUS interrupt signals INT1-6 are sent to the CMPU directly (levels 0 and 7 are not used). Using these, the CMPU module generates a single vector for each MULTIBUS interrupt level. The processor must then determine which of the MULTIBUS devices sharing that level has generated the interrupt. The INTA (Interrupt Acknowledge) signal is not used. See Chapter 5 for the interrupt vectors assigned to each MULTIBUS level.

  The MULTIBUS standard defines INT0 as the highest priority and INT7 as the lowest. The 68020 encodes eight levels of interrupts on its IPL0, IPL1, and IPL2 lines, with level 0 as lowest and level 7 as the highest priority (the opposite of the MULTIBUS convention). The MC5600/5700 uses the 68020 convention to prioritize its MULTIBUS lines.

- **Bus Inhibit** - The standard MULTIBUS signals INH1 and INH2 are not supported.

- **Arbitration** - Because the MULTIBUS can accommodate more than one master, the MULTIBUS requires arbitration circuitry, described in Section 9.3. Since the MASSCOMP arbitration circuitry supports only parallel type (as opposed to serial) arbitration, the MULTIBUS serial arbitration line BPRO (Bus Priority Out) is not used.

- **Common Bus Request** - The standard MULTIBUS signal CBRQ is grounded by the CMPU module. Slot 1 receives a non-bussed signal (MBUS MC CBRQ H) from the AFM/ARB Module. This private signal is the logical OR of MBUS BREQ from slots 2 through 15 and is used by the secondary MULTIBUS repeater module for arbitration. Note that, because the implementation of CBREQ is optional, this feature does not constitute a compromise in functionality or performance.

9.1.2 MULTIBUS Enhancements

The following MULTIBUS signal lines are enhancements of the standard MULTIBUS:

- **BLOCK MODE** - This enhancement allows 32-bit transfers over the MULTIBUS. Pin 41 (standard signal INT0) is used by MASSCOMP as this block mode control signal. Section 9.2 explains the use of this signal.

- **MC LOCK2** - Pin 36 (standard INT7) is used by the MBA in the same way that MBUS LOCK is used (see Chapter 10). This signal allows the GPM & GPH MASSCOMP graphics modules to perform operations requiring a locked bus. This signal on the MC5500 is used for bypassing the I/O Map.

- **AC LO** - Pin 78 (standard RESERVED, bussed) is used by MASSCOMP to detect the failure of the AC power or the RESET switch on the front panel being pressed. The section on AFM initialization circuitry in Chapter 5 explains how this signal is used.

- **SW RESET** - Pin 24 (standard INH1) is used by MASSCOMP to sense the INTERRUPT switch on the front panel.
- **MBUS ERROR** - Pin 26 (standard INH2) is used by MASSCOMP to activate the red front panel FAULT LED. The signal asserted when the **CONS ERR** bit in the PCRB of any CMPU module has been written with a **1**.

9.2 MULTIBUS 32-bit Block Mode

This section describes the MASSCOMP enhancement to the IEEE-796 MULTIBUS called **Block Mode** within the context of the MC5600/5700 system. This enhancement allows the transferring of 32 bits of data per address over the existing MULTIBUS data path.

The motivation underlying this extension is to provide twice the throughput for devices that implement this protocol, while maintaining compatibility with devices that implement only the standard 796 specification. The term **Block Mode transaction** or **protocol** is used when referring to 32-bit operations, while **standard transaction** or protocol is used when referring to the existing 796 byte and word transactions.

The system uses pin 41 (INT0 in the MULTIBUS standard) from the P1 connector as the Block Mode control signal. A standard MULTIBUS device normally ignores this signal, while MASSCOMP devices that have Block Mode capability, such as the DACP, use the signal as the Block Mode control line.

The BLOCK MODE signal is asserted in unison with the address. When the BLOCK MODE control signal is asserted, the current MULTIBUS transaction is extended to 32 bits of data by providing two 16-bit data words per address. Block mode is only supported for memory strobes.

The ADR1 signal has two functions under Block Mode protocol. First, the value of ADR1 at the beginning of a transaction determines which half of the associated longword to send first. Second, ADR1 is toggled by the master in the middle part of the transaction. During read transactions, this transition indicates that the second data word should be sent by the slave. During write transactions, this transition indicates that the second data word should be strobed by the slave.

In the middle of a cycle, the master flips address line ADR1 to command the other half of the longword. In a read, the slave senses the ADR1 flip and switches its data to the other word within the longword. In a write, the master changes ADR1 and the write data simultaneously. The slave, sensing the ADR1 change, obtains the data and performs the write.

Block Mode protocol differs from standard protocol in several ways. During block mode transactions:

- The signals that are involved are synchronous with respect to BCLK (as opposed the normal asynchronous protocol), which introduces some timing relationships not in accordance with the MULTIBUS specification.
- The timing relationships between memory strobes, XACK and data differ significantly from those of standard transactions, as described later in this chapter.
- 32 bits of data are time-multiplexed onto 16 data lines in two 100 nanosecond

(minimum) time slots.

9.2.1 Requirements

At least one Block Mode master device and one Block Mode slave device are required for a system to support Block Mode protocol. Slave devices that implement Block Mode protocol must also support standard word and byte transactions. There is no corresponding requirement for master devices.

In systems in which both standard and Block Mode slave devices are intermixed, the system designer must insure that Block Mode transactions are not inadvertently directed to slave devices that support only standard transactions, since the resultant data transfers will be unpredictable. The possibility for this type of mismatch exists because a standard slave can be selected by a Block Mode master, since device selection is based on address. The resultant handshake appears to complete normally, while actually the proper data transfer does not occur. A similar situation exists between standard byte and word devices if slaves that support only byte transactions are allowed in systems that support only word transactions.

9.2.2 General Protocol

A master device performing a Block Mode transaction uses the following protocol:

- It becomes the bus master using the standard 796 Bus arbitration sequence, then
- It Asserts BUSY, BLOCK MODE, BHEN, and ADR*n* off the next negative edge of BCLK.

It is possible to do Block Mode transactions in burst mode by keeping BUSY asserted while many transactions are issued. It is also possible to mix byte, word, and Block Mode transactions under one continuous assertion of BUSY. However, in all the timing diagrams in this specification, BUSY is shown being asserted and deasserted in unison with BLOCK MODE, BHEN, and ADR*n*, since this is the typical case.

9.2.3 Block Mode Read Protocol

The timing diagrams for a typical Block Mode Read transaction are shown in Figure 9-1 and Figure 9-2. These figures are referenced throughout this section.

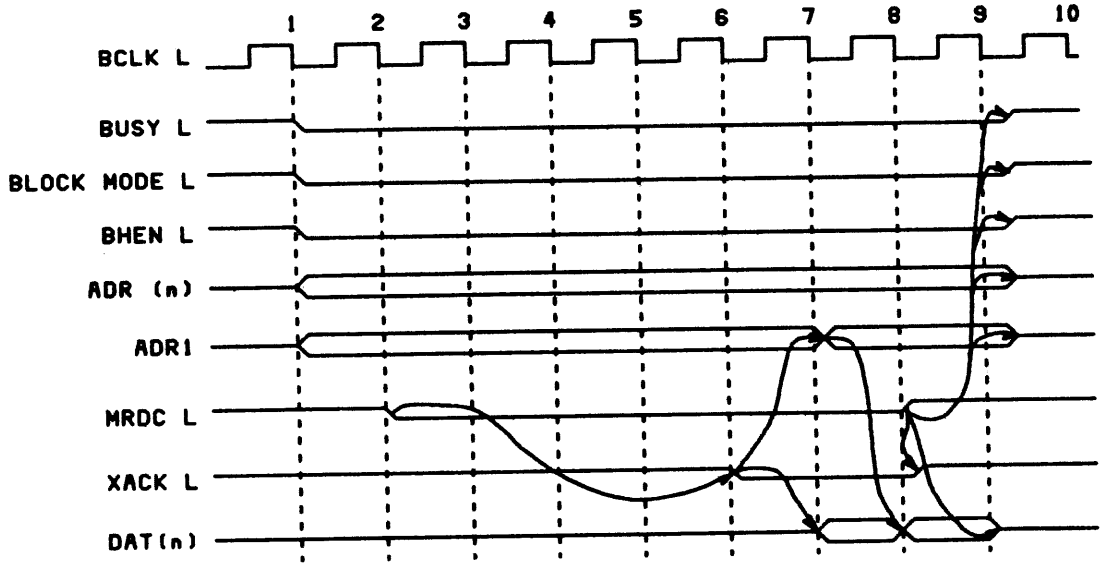


Figure 9-1. Block Mode Read Protocol Example

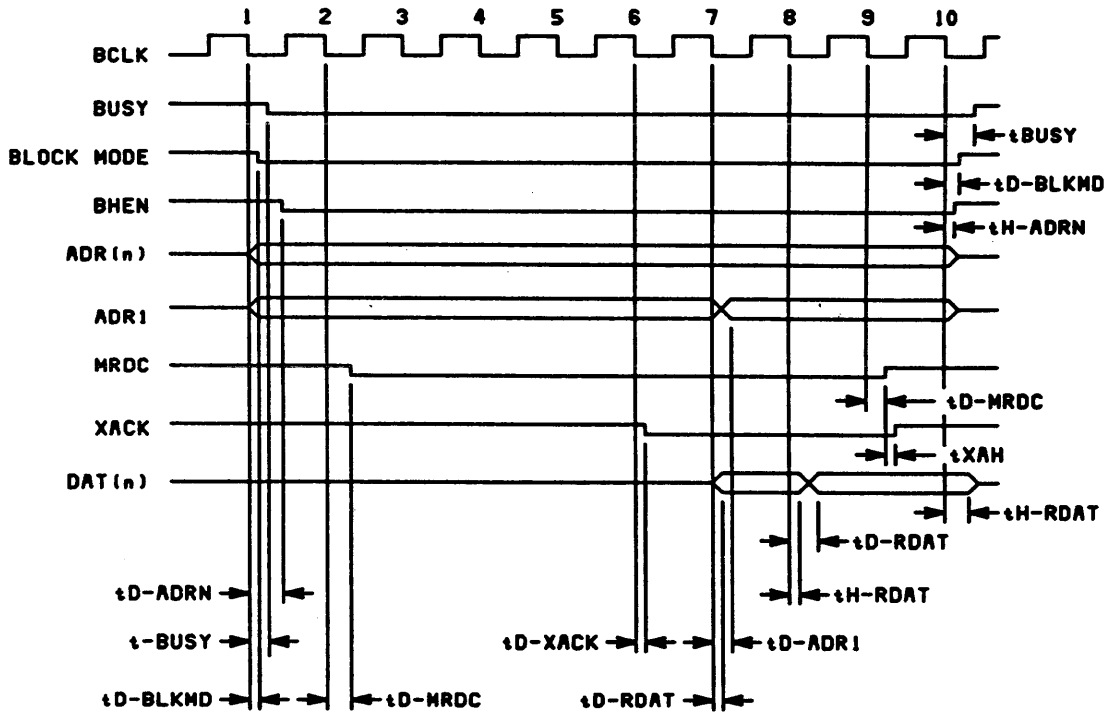


Figure 9-2. Block Mode Read Timing Parameters

Typically, the master asserts the read strobe signal MRDC one clock cycle after the assertion of the BUSY, BLOCK MODE, BHEN, and ADR(*n*) lines to indicate that the transaction is to be a Block Mode read. The master may delay asserting MRDC, but the performance will be degraded.

The selected slave senses this assertion on edge #3 and begins its internal data fetch operation. The slave can return XACK off of edge #3 or any subsequent negative edge, depending on its internal read performance. For the purpose of this discussion, XACK is shown being asserted off of edge #6 in the examples shown in Figure 9-1 and 9-2. The only additional requirement on when XACK can be asserted is that the slave must commit to being able to finish the operation (that is, supply both data words within the timing limitations).

The slave asserts the XACK signal one BCLK cycle ahead of placing valid data on the data bus, to give the master device advance notice of impending data arrival. The master device may strobe the first data word on negative edge #8, which is the second negative edge following the assertion of XACK.

The master device toggles ADR1 on edge #7 to indicate that the second data word should now be transmitted by the slave. The second data word is strobed by the master on edge #9. The master can wait any number of cycles after sensing the assertion of XACK before toggling ADR1. However, by toggling ADR1 in the very next cycle, maximum transfer rates can be achieved.

The master deasserting MRDC controls the deassertion of all other signals at the end of the Block Mode read transaction. In Figure 9-1, MRDC is deasserted off of edge #8, which causes BUSY, BLOCK MODE, BHEN, ADR(*n*), and DAT(*n*) to deassert off of edge #9. If the master needs more time, the termination of the cycle can be stalled by continuing to assert MRDC, as shown in Figure 9-2. The master must hold BUSY, BLOCK MODE, BHEN, and ADR(*n*), and the slave must hold DAT(*n*), for one clock cycle after MRDC is deasserted.

9.2.4 Block Mode Write Protocol

The timing diagrams for a typical Block Mode Write transaction are shown in Figure 9-3 and Figure 9-4. These figures are referenced throughout this section.

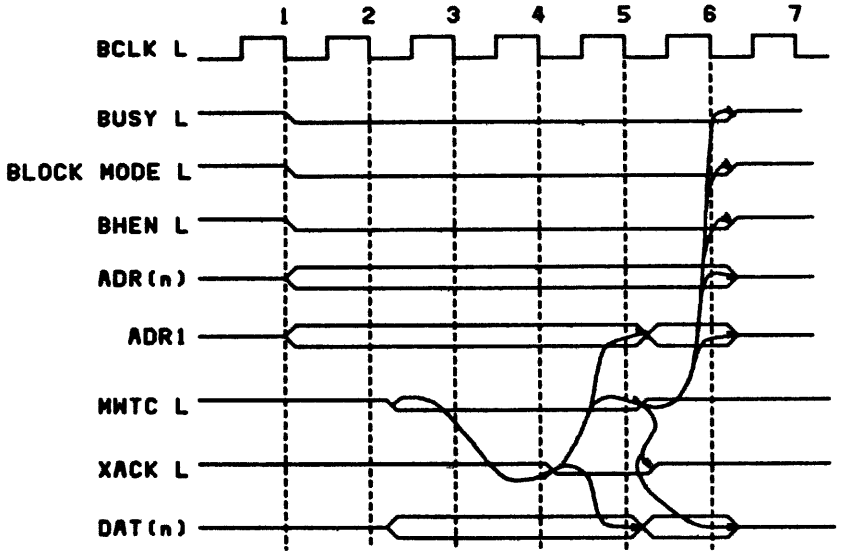


Figure 9-3. Block Mode Write Protocol Example

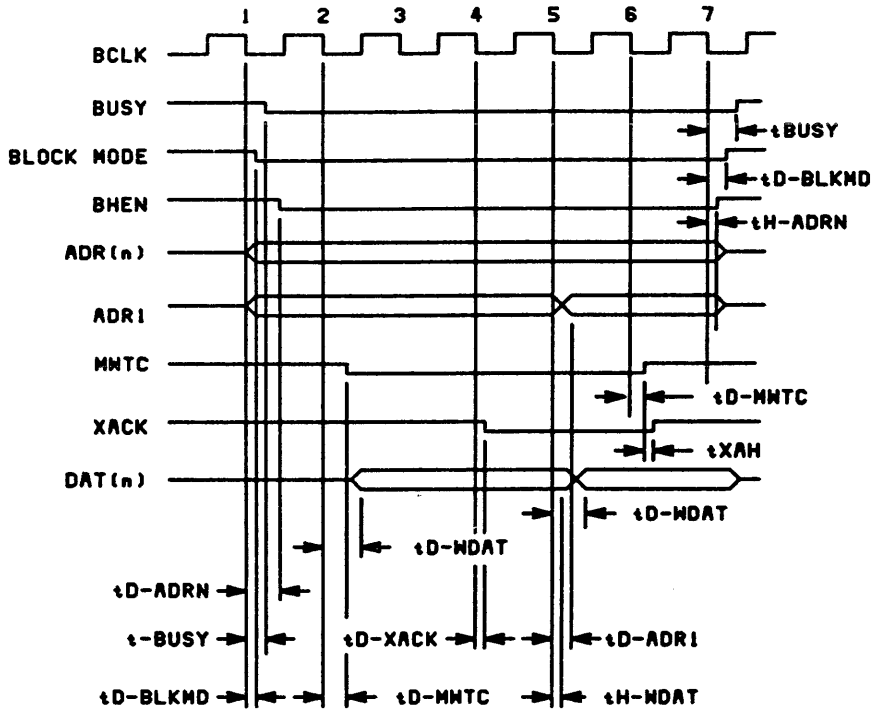


Figure 9-4. Block Mode Write Timing Parameters

Typically, the master asserts the write strobe signal MWTC and valid DAT(*n*) one clock cycle after the assertion of BUSY, BLOCK MODE, BHEN, and ADR(*n*), to indicate that the transaction is to be a Block Mode write. The master may delay asserting MWTC, but performance will be degraded. The selected slave senses this assertion on edge #3 and begins its internal write operation. The slave can return XACK off of edge #3 or any subsequent negative edge, depending on its ability to dispense with the incoming data. For the purpose of this discussion, XACK is shown being asserted off of edge #4, as shown in in Figures 9-3 and 9-4. In this example, the first data word can be strobed by the slave on edge #3, edge #4, or edge #5.

The master device toggles ADR1 off of edge #5 in the examples to indicate to the slave that the second data word is now being transmitted. The master can wait 0,1,or 2 cycles after sensing the assertion of XACK before toggling ADR1. However, by toggling ADR1 in the next cycle, the transaction bandwidth is maximized.

The slave must strobe the second data word on the negative edge after ADR1 has been toggled. The master can deassert MWTC and ADR1 off of the same edge (as shown in Figure 9-3), or it can delay deasserting MWTC after ADR1 has been toggled (as shown in Figure 9-4). The master device must hold BUSY, BLOCK MODE, BHEN, ADR(*n*), and DAT(*n*) for one clock cycle after MWTC is deasserted.

9.2.5 Block Mode Timing Specification

This section defines and specifies all timing parameters that pertain to devices that implement Block Mode protocol. Block Mode devices must be capable of operating correctly in systems where BCLK is running at its maximum rate of 10 MHz.

Both maximum delay times and maximum setup times for signals are specified such that the following relationship is observed:

MIN BCLK CYCLE = MAX DELAY + MAX SETUP + MAX CLOCK SKEW

The BCLK clock cycle is 100 nanoseconds, and the clock skew is fixed at 5 nanoseconds. Then, for any *signal*:

tD_*signal* + tS_*signal* = 95ns

Timing parameters are defined and specified by Table 9-2 in conjunction with Figures 9-2 and 9-4. In Figures 9-2 and 9-4, delay times and hold times are defined relative to the negative edge of BCLK. Setup times are not explicitly defined in Figures 9-2 and 9-4, but are implicitly defined by their associated delay times according to the formula above. For example, tS-BLKMD is the time taken by the master to assert BLOCK MODE after BCLK edge 1. The slave must have BLOCK MODE available tS-BLKLMD before BCLK edge 2. The setup time is always defined relative to the negative edge of BCLK that is subsequent to the edge which defines the delay.

9.3 Bus Arbitration

The AFM arbitrates use of the MULTIBUS, using a parallel arbitration scheme that grants the use of the bus to the device with the highest priority request.

In parallel arbitration, each module on the bus has one request line and one grant line associated with it. All MULTIBUS request and grant lines enter the arbitration circuitry of the AFM. The AFM issues a grant to the device with the highest priority level request.

The arbitration section of the AFM also generates the MULTIBUS signal **MBUS MC CBRQ**. This signal is the logical OR of all but the highest MULTIBUS REQ request line. The **MBUS MC CBRQ** signal is used by the MASSCOMP MULTIBUS repeater module for passing bus mastership between backplanes. Note that this signal differs from the standard MBUS CBRQ signal in that it is not bussed (it only goes to slot 1) and is a high true signal. The MULTIBUS signal MBUS BUSY must be deasserted along with MBUS BPRO for a given module to be granted mastery of the bus.

The AFM accommodates either the 7/8 or 15 slot MULTIBUS configuration (see Chapter 11). On a 30 slot system, the AFM performs the MULTIBUS arbitration for slots **J1** through **J15**. The ARB module performs the MULTIBUS arbitration for slots **J16** through **J30**.

Table 9-2
Extended Protocol Timing Specification

Parameter	Description	Min	Max	Units
tD-ADRN	ADR(*n*) delay time	—	45	ns
tS-ADRN	ADR(*n*) setup time	50	—	ns
tH-ADRN	ADR(*n*) hold time	5	—	ns
tD-ADR1	ADR1 delay time	—	40	ns
tS-ADR1	ADR1 setup time	55	—	ns
tD-BLKMD	Block Mode delay time	—	40	ns
tS-BLKMD	Block Mode setup time	55	—	ns
tD-MRDC	MRDC delay time	—	40	ns
tS-MRDC	MRDC setup time	55	—	ns
tD-MWTC	MWTC delay time	—	40	ns
tS-MWTC	MWTC setup time	55	—	ns
tD-XACK	XACK delay time	—	30	ns
tS-XACK	XACK setup time	65	—	ns
tD-RDAT	Read Data delay time	—	50	ns
tS-RDAT	Read Data setup time	45	—	ns
tH-RDAT	Read Data hold time	13	—	ns
tD-WDAT	Write Data delay time	—	50	ns
tS-WDAT	Write Data setup time	45	—	ns
tH-WDAT	Write Data hold time	20	—	ns

NOTE:

(1) The parameters t_{BUSY} and t_{XAH} are defined in the previously referenced MULTIBUS manual.

(2) XACK is asserted synchronously and deasserted asynchronously.

Chapter 10
The MULTIBUS Adaptor

ILLUSTRATIONS

TABLES

Chapter 10
The MULTIBUS Adaptor

The MASSCOMP MULTIBUS, described in Chapter 9, provides a general purpose I/O interface bus for the MC5600/5700 System. The Synchronous Memory Interconnect (SMI) bus, described in Chapter 13, is a higher speed bus designed for memory devices interacting with the CPU. The interface circuitry between the MULTIBUS and the SMI bus is called the **MULTIBUS Adaptor (MBA)**. The MBA allows MULTIBUS devices (disk controllers, graphics controllers, and so forth) and SMI devices (the CPU, Memory modules, and array processor) to communicate with each other across the two buses.

This chapter explains the operations and functions of the MBA, the differences between the SMI and MULTIBUS that are handled by the MBA, and the address mapping considerations between the two buses.

10.1 General Operation

The MBA performs two functions, corresponding to the two possible directions of data transfer between the SMI and MULTIBUS:

- It enables an SMI device to access a given MULTIBUS device by making the necessary data format translation.
- During a direct memory access (DMA) by MULTIBUS devices into SMI memory space, the MBA translates the MULTIBUS address to an SMI physical address. The MBA also handles the SMI protocol during the DMA transfer on behalf of the MULTIBUS device and makes the necessary data translation.

The MBA circuitry is physically situated on the CMPU board, but it is logically independent from the CPU. The MBA has a unique SMI node I.D. apart from the CPU, and devices that transfer data to and from the MBA treat it as an independent SMI device. The exception to this independence is that interrupt signals from all MULTIBUSs are coupled directly to one CPU, rather than being routed through the MBA. The current MASSCOMP RTU operating system requires that all MULTIBUS interrupts be fielded by the boot processor (processor I.D. **1**).

The MBA always acts as a MULTIBUS master while it is an SMI slave, and MULTIBUS slave while it is an SMI master (see Chapter 7 for SMI master and slave definitions).

The MBA handles the four possible access types between the SMI and MULTIBUS as follows:

- **SMI to MULTIBUS memory.** An SMI master accesses the 16 MByte (24-bit) MULTIBUS memory space through the 28-bit SMI address space. The upper 4 bits of the physical address select the appropriate MULTIBUS (bit<27> must be **0** and bits <26:24> must be **1-7**. The lower 24 bits are passed through the selected MBA unmodified. No address translation is performed in this direction. The MBA, serving as MULTIBUS master, cannot perform MASSCOMP Block Mode transfers (described in Chapter 9).

- **SMI to MULTIBUS I/O.** An SMI master accesses the 64 KByte MULTIBUS I/O space through the 28-bit SMI. The upper 12 bits of the SMI address select the appropriate MULTIBUS I/O space, and the lower 16 bits are passed through the MBA unmodified. Again, no address translation is performed in this direction. The MBA, serving as MULTIBUS master, cannot perform MASSCOMPs Block Mode transfers (described in Chapter 9).
- **MULTIBUS memory to SMI.** The (24-bit) MULTIBUS accesses the 256 MByte (28-bit) SMI address space with a memory strobe (in a DMA reference, for example). Since the MBA has only 24 address bits available to reference into a 28-bit space, all MULTIBUS references into SMI memory space must be mapped using the I/O Map (described in the next section). Without the I/O map, each MULTIBUS would be able to access only the lower 16 MBytes of SMI memory, and transfers would be limited to one page per disk transfer. The map allows dynamic use of the entire SMI memory space by the MULTIBUS. The MBA, when serving as MULTIBUS slave, can perform Block Mode transfers (see Chapter 9).
- **MULTIBUS I/O to SMI.** The MBA does not respond to MULTIBUS I/O strobes.

There are several crucial restrictions to the cases above:

- **I/O Maps** - A MULTIBUS device can read and write only its own I/O map by referencing the appropriate SMI physical space. This process is described in Section 10.2.2 and 10.2.3.
- **MULTIBUS *x* to MULTIBUS *y*** - A MULTIBUS device *cannot* access devices on another MULTIBUS by going through two MBAs.
- **MULTIBUS *x* to MULTIBUS *x*** - A MULTIBUS device can access other devices on its own MULTIBUS directly without going out on the SMI by appropriately setting the I/O map. The device *cannot* access other devices on its own MULTIBUS by going out to the SMI and back through its own MBA.

10.2 The I/O Map

Each MBA uses a 4096 x 17 bit RAM called the **I/O Map** to store MULTIBUS-to-SMI address translations. The I/O map translates MULTIBUS addresses into SMI physical addresses in the same way the Translation Buffer translates virtual addresses. The I/O Map RAM for each MULTIBUS is physically located on its associated CMPU board. Each I/O Map is assigned to the SMI address space shown in Chapter 8. As in the Translation Buffer, each I/O Map entry maps one 4 KByte page of SMI address space.

Figure 10-1 shows the translation process used by the I/O Map.

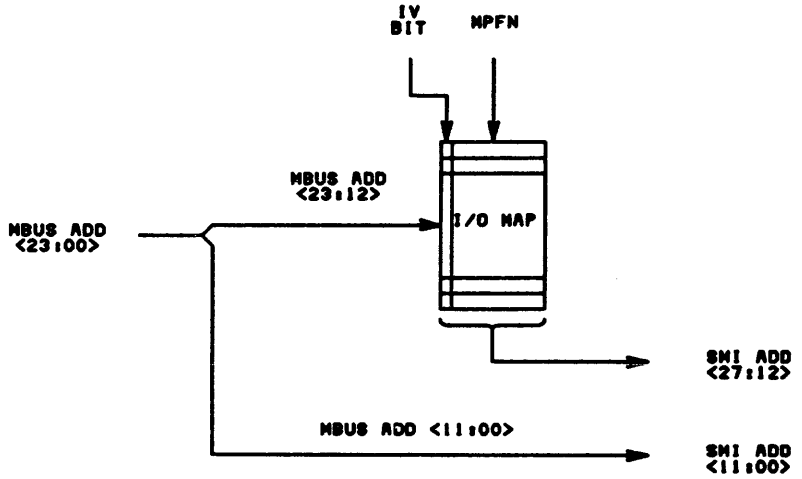


Figure 10-1. I/O Map Translation Algorithm

The **MULTIBUS Page Frame Number (MPFN)** is analogous to the PFN field of the Translation Buffer entry. The MPFN determines which of the 65,536 4-KByte pages of SMI physical address space to access. The most significant bit of the MPFN selects between SMI memory and SMI I/O space.

When the MULTIBUS has MWTC or MRDC asserted to read or write an address, the MBA first uses the upper 12 bits of the address as an index into the mapping RAM. The indexed **Map Table Entry (MTE)** contains the corresponding MPFN and the IV bit. If cleared, the IV bit indicates that the entry is valid. If the entry is valid, the MBA performs the appropriate SMI transaction. If the entry is invalid, the MBA assumes the address targets a local MULTIBUS device and does nothing.

10.2.1 The Map Table Entry Format

Each of the 4096 **I/O Map Table Entries (MTEs)** are accessible to SMI devices as longword data in the format shown in Figure 10-2.

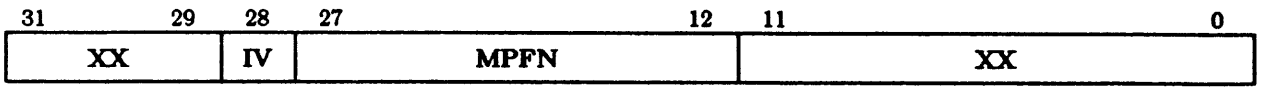


Figure 10-2. I/O Map Table Entry Format

The following are the MTE fields:

MPFN MULTIBUS Page Frame Number (16 bits). This field becomes SMI DAL <27:12> when the MULTIBUS address is translated onto the SMI.

IV Invalid (1 bit). If this bit is a **1**, the associated entry is invalid and no SMI transfer occurs. If this bit is a **0**, the associated entry is valid and the MBA performs the appropriate SMI transfer if either MWTC or MRDC is asserted on its MULTIBUS.

XX Not used

10.2.2 Accessing the I/O Map from the MULTIBUS

It is possible for a MULTIBUS intelligent peripheral to access the Mapping RAM and update MAP Table Entries on its associated MULTIBUS. Devices cannot access the maps of other MBAs. For such devices that require the ability to set up their own translations, MAP locations must be set up by the operating system that point at the mapping RAM itself. This section describes how to do this.

To allow a controller access to the I/O map, the operating system must set up one, two, three, or four locations in the map, called **self-mapping entries**, that point back into the map itself. Each one of these special locations allows the MULTIBUS controller to access 1024 contiguous locations in its MBA's I/O Map. If it is desirable for all 4096 entries to be available to the controller, all four special locations must be set up, since each I/O Map entry translates a 4 KByte page of data and the I/O Map is 4096 entries x 4 bytes.

Table 10-1 shows the entries for the four Map locations and the SMI physical addresses for each.

Table 10-1
Setting Up Self-Mapping I/O Map Entries

SMI Address of Self-Mapping Entry	Self-Mapping Entry Value	Accesses Map Locations:
0xF18000 + *a* * 0x20000 + *d* * 0x10 + 0	0x0F18*zzz* + *a* * 0x20000	0 - 1023
0xF18000 + *a* * 0x20000 + *e* * 0x10 + 4	0x0F19*zzz* + *a* * 0x20000	1024 - 2047
0xF18000 + *a* * 0x20000 + *f* * 0x10 + 8	0x0F1A*zzz* + *a* * 0x20000	2048 - 3071
0xF18000 + *a* * 0x20000 + *g* * 0x10 + C	0x0F1B*zzz* + *a* * 0x20000	3072 - 4095

Table 10-1 uses the following variables:

1. The variable *a* has a range of 1 to 7 and selects the desired processor's I/O Map set, as described in Chapter 8.
2. The variables *d, e, f,* and *g* have a range of 0 to 2^{10} — 1 and select which of the 1024 longword Map locations contains the entry that maps back. It is preferable (though not necessary) that *d, e, f,* and *g* are the same value for ease of understanding
3. The value of bits <14:00> are **0** in a self-mapping entry, since:
   - Only bits <28:12> of any I/O Map Entry are valid
   - Bit <14> is not decoded. The 16 KByte I/O Map is positioned at the bottom of the 32 KByte space
   - Bits <13:12> are taken from the MULTIBUS Address (not from SMI address bits <13:12>) when the read or write operation occurs due to hardware limitations.

CAUTION

You must be *extremely* careful that a write from the MULTIBUS does not overwrite the location through which it was translated. This will jeopardize future attempts to write into SMI space by MULTIBUS devices.

Table 10-2 gives an example of 4 self-mapping entries.

Table 10-2
Example of Self-Mapping I/O Map Entries

Address	Entry Value
0xF18000	0x0F1800
0xF18004	0x0F1900
0xF18008	0x0F1B00
0xF1800C	0x0F1A00

10.2.3 Using the Four Self-Mapping Entries

This section explains how to use the self-mapping I/O Map entries correctly once you have set them up as described in the previous section.

Access of an MTE from the MULTIBUS requires two MULTIBUS word references, and uses the format shown in Table 10-3.

Table 10-3
MTE Addressing

MTE Bits	Word #	MBUS ADD <01>
<31:16>	word 0 (even)	0
<15:00>	word 1 (odd)	1

If the MULTIBUS peripheral is a 68000-based device (such as an Independent Graphics subsystem), then this process is automatic as long as longword operands are specified. In other words, the 68000/10/20 processor takes care of the two word references per longword and sets ADRS <1> as appropriate.

CAUTION

You must treat the MAP Table Entry as a longword operand when accessing it with the 68020 over the SMI. Other MAP operand sizes will produce unpredictable results and should be avoided.

Table 10-4 shows the MULTIBUS address you use to select each of the four special entries.

Table 10-4
Self-Mapping I/O Map Entries

For Map Entries	MULTIBUS Address	SMI Address generated
0 - 1023	$d * 2^{14} + 0 + mmm$	0xF38*mmm*
1024 - 2047	$e * 2^{14} + 0x1000 + nnn$	0xF39*nnn*
2048 - 3071	$f * 2^{14} + 0x2000 + ppp$	0xF3A*ppp*
3072 - 4095	$g * 2^{14} + 0x3000 + qqq$	0xF3B*qqq*

The following rules apply to Table 10-4:

1. The values *mmm, nnn, ppp,* and *qqq* should be located on word boundaries (address bit <0> should always be a **0**). Since bit <1> is used to read/write the upper or lower word, each value allows access to all 1024 entries in the associated group.
2. To prevent a MULTIBUS controller from inadvertently overwriting the entry through which it has gained access to the I/O Map, *mmm, nnn, ppp,* and *qqq* should not equal any of the values shown in Table 10-5. This is the safest method of preventing the corruption of the I/O Map, while taking up only 32 of the 4096 entries.

Table 10-5
MULTIBUS-Unwritable Self-Mapping Entries

Address Values			
$d * 0x10$	$e * 0x10$	$f * 0x10$	$g * 0x10$
$d * 0x10 + 2$	$e * 0x10 + 2$	$f * 0x10 + 2$	$g * 0x10 + 2$
$d * 0x10 + 4$	$e * 0x10 + 4$	$f * 0x10 + 4$	$g * 0x10 + 4$
$d * 0x10 + 6$	$e * 0x10 + 6$	$f * 0x10 + 6$	$g * 0x10 + 6$
$d * 0x10 + 8$	$e * 0x10 + 8$	$f * 0x10 + 8$	$g * 0x10 + 8$
$d * 0x10 + A$	$e * 0x10 + A$	$f * 0x10 + A$	$g * 0x10 + A$
$d * 0x10 + C$	$e * 0x10 + C$	$f * 0x10 + C$	$g * 0x10 + C$
$d * 0x10 + E$	$e * 0x10 + E$	$f * 0x10 + E$	$g * 0x10 + E$

10.3 SMI / MULTIBUS Data Transfers

Another function of the MBA, besides performing address translations, is to translate the data format between the SMI and the MULTIBUS. This translation is necessary because the bytes within a word or longword are numbered differently on the SMI (which follows the 68020 numbering scheme) than on the MULTIBUS. Figure 10-3 shows the byte and word numbering within an SMI longword, compared with the byte numbering within a MULTIBUS word.

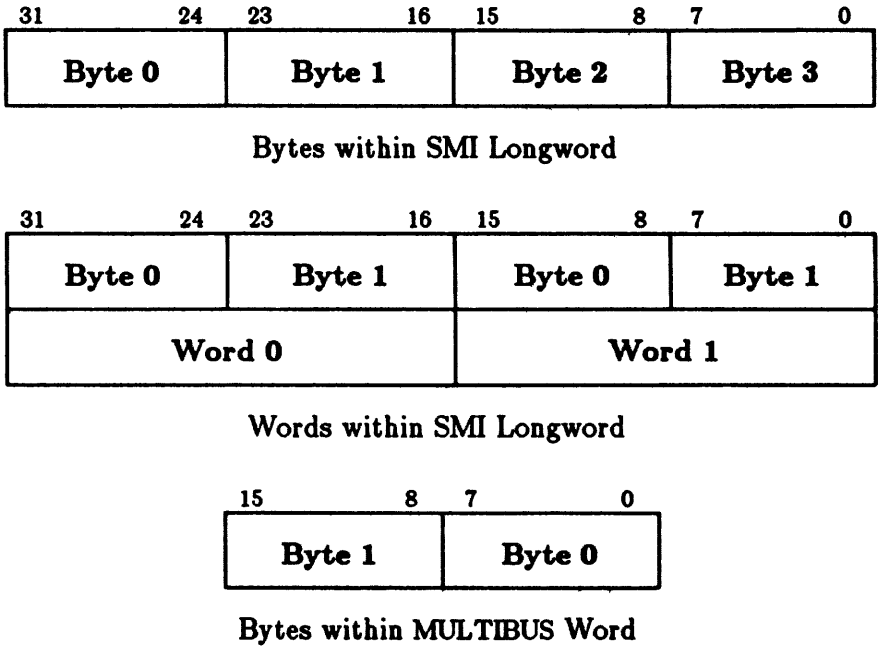


Figure 10-3. SMI and MULTIBUS Byte and Word Numbering

The MBA handles the data position differences whenever transfers between these two buses occur. Each byte has a unique address within each structure. When a byte is transferred between one bus and another, its position within a word or longword may change, but its address remains constant. Therefore, in transfers between the SMI and the MULTIBUS, bytes are placed in accordance with their addresses. This is done using the numbering schemes shown in Figure 10-3.

Tables 10-6, 10-7 and 10-8 give the rules for byte, word, and longword block mode transfers.

Table 10-6
Transfers Between MULTIBUS Byte Devices and SMI DAL

SMI DATA		MBUS ADDR <01>	MBUS ADDR <00>	MBUS DATA
Byte 0	DAL <31:24>	0	0	DAT<07:00>
Byte 1	DAL <23:16>	0	1	DAT<07:00>
Byte 2	DAL <15:08>	1	0	DAT<07:00>
Byte 3	DAL <07:00>	1	1	DAT<07:00>

Table 10-7
Transfers Between MULTIBUS Word Devices and SMI DAL

SMI DATA		MBUS ADDR <01>	MBUS DATA
Word 0	DAL <31:24> DAL <23:16>	0	DAT <07:00> DAT <15:08>
Word 1	DAL <15:08> DAL <07:00>	1	DAT <07:00> DAT <15:08>

Table 10-8
Transfers Between MULTIBUS Block Mode Devices and SMI DAL

SMI DATA		MBUS ADDR <01>	MBUS DATA
Word 0	DAL <31:24> DAL <23:16>	0	DAT <07:00> DAT <15:08>
Word 1	DAL <15:08> DAL <07:00>	1	DAT <07:00> DAT <15:08>

10.3.1 Dynamic Bus Sizing

The 68020 allows operand transfers to and from 8-bit, 16-bit, and 32-bit ports by dynamically determining the port's size during each bus cycle. This size determination feature is called **dynamic bus sizing**. If a port is found to be smaller than the requested data size, the 68020 automatically runs as many cycles as necessary to transfer the data. For example, if the 68020 executes a 32-bit read instruction from a 16-bit device (such as the MULTIBUS), the processor:

- Latches the first 16 bits onto bits <31:16> of its data bus
- Runs another read cycle at the previous address + 2

- Latches the second 16 bits onto bits <15:00> of its data bus, and then continues.

The MBA is seen as a 16-bit port by SMI processors. The MBA supports dynamic bus sizing between the two buses, but *only when the MBA is an SMI slave.* In other words, when the 68020 writes or reads longword data to or from a MULTIBUS through the MBA, the MBA - SMI - CPU interface logic handles the extra cycle needed to access the upper 16 bits. On MC5500 systems, this is automatically handled by the 16-bit 68010. Thus, MC5500 (68010-based) software is compatible in this respect with MC5600/5700 systems.

The MBA does not support dynamic bus sizing when it is an SMI bus master. Transfer sizes must match the port size of the slave device.

Note also that 64-bit transfers to the MBA are not supported and produce unpredictable results.

10.4 MULTIBUS Lock

Each bus has a mechanism used to lock the bus during a read-modify-write operation. On the MULTIBUS it is called **MBUS LOCK** and on the SMI it is called **SMI LINH (Lock Inhibit)**. The MBA implements this locking mechanism only in the case when a MULTIBUS device is accessing the SMI. That is, asserting the **MBUS LOCK** causes the **LINH** to be asserted as long as **MBUS BUSY** remains asserted. If **MBUS BUSY** is deasserted while the MBA is still in the middle of the last SMI write cycle, the MBA circuitry guarantees that **LINH** remains asserted for the remainder of the SMI cycle. However, the **LINH** cannot be routed through the MBA to assert the **MBUS LOCK** signal.

10.5 SMI / MULTIBUS Error Handling

There are two general types of data transfer errors that may occur at the interface between the MULTIBUS and SMI:

1. *The SMI master accesses a non-existent address or malfunctioning MULTIBUS device.* In this case, the MULTIBUS slave device does not respond to the MBA with an XACK signal (see Chapter 9). The MBA logic then times out after approximately 100 microseconds and transmits a RETURN ERROR code on the SMI.
2. *The MULTIBUS accesses a non-existent address or malfunctioning SMI device.* The read and write cases are different:
   - **Read.** When a MULTIBUS device performs a read instruction, the following SMI codes may be returned (see Chapter 7 for further explanation of the codes):
     A. NACK - Due to a bad address (due to a bad I/O map entry, for example) or broken module
     B. RERR - Due to an uncorrectable error on a memory device, for

example (see Chapter 6)

C. RTNW - Due to a broken SMI device

In any of these cases, the MBA never asserts MBUS XACK, and the MULTIBUS master device must timeout and flag the error.

- **Write.** When a MULTIBUS device performs a write instruction, *no error condition is generated and no error flag is set.* The operating system must take the necessary precautions to assure that the SMI address provided by the map is valid and that the SMI device is functioning properly.

10.6 Deadlock Avoidance

The fact that the MC5600/5700 system uses two separate buses, and that these buses operate in parallel, raises the possibility of a deadlock. Deadlocks occur when a process is unable to obtain the use of a resource it needs to continue. Specifically, a deadlock would arise if the SMI interface section of the MBA committed to perform a data transfer to the MULTIBUS, while, at the same time, a MULTIBUS device won arbitration to perform a transfer to SMI space. In this case, the MBA cannot complete the SMI transfer, since the MULTIBUS is busy. Neither can the MBA complete the MULTIBUS data transfer, since it already has one pending from the SMI.

In this situation, the MBA gives MULTIBUS devices priority over SMI devices. Before accepting an SMI read or write request, the MBA checks for the following:

- MBUS GRNT is asserted
- MBUS BUSY is deasserted
- The MBA MULTIBUS slave circuitry is idle

Thus, while the MBA is acting as MULTIBUS slave, the MBA SMI interface returns a RETRY to any SMI read or write requests, and the SMI device must retry the transfer.

Chapter 11
System Configuration

ILLUSTRATIONS

TABLES

Chapter 11
System Configuration

Each module has various switches and jumpers that set that module's physical address and other module-specific features. These must be set before the modules are placed in the system. This chapter discusses how to configure the following modules in an MC5600/5700 system:

1. The CMPU module (Section 11.2) - Adding a processor to the system
2. The CMM module (Section 11.3) - Adding more memory to the system
3. The AFM/ARB module and the Backplane (Section 11.4) - Adding a MULTIBUS to the system

For system configuration information, see the *MC5600/5700 System Configuration Guide.*

11.1 Packages

The MC5600/5700 is available in four packages, shown in Table 11-1 and referenced throughout this chapter.

Table 11-1
MC5600/5700 Packages

System	Package		Slots in Standard Package		Additional Slots with MBUS Expander	
			SMI/MBUS	STD+	MBUS	STD+
MC5600	Pedestal		15	n/a	n/a	n/a
	Table Top		15	9	n/a	n/a
	Rack Mount (40"/70")		15	9	15	9
MC5700	Wide Cabinet	1 BP	15	9	15	9
		2 BP	30	18	15	9

The number of slots refers to the maximum number of printed circuit boards on the SMI/MULTIBUS(s) that can physically fit in the system package. The only differences between the packages, in terms of configuration, is the number of slots and the power supply capacities.

The wide cabinet package is available with one backplane (15-slot) or two backplanes (30-slot). A backplane may be added to the 15-slot wide cabinet package (described in Section 11.4). Also, a MULTIBUS expander box with repeater module is available for the rack mount and wide cabinet packages to add slots to an existing MULTIBUS. Note that the expander box *does not* add slots for SMI devices, and requires one slot itself on the SMI.

Each slot in a backplane consists of two physical edge card connectors, labeled:

- P1 (86 pins) - the module's link into the MULTIBUS
- P2 (60 pins) - the module's link into the SMI bus

The **MULTIBUS** refers to the signals common to connector P1 and 4 signals on P2 (MBUS ADRS <20:23>) of all slots. The **SMI** refers to the remaining signals common to connector P2 of all slots.

11.1.1 The Front Panel

The system's front panel is the same on all packages and is shown in Figure 11-1.

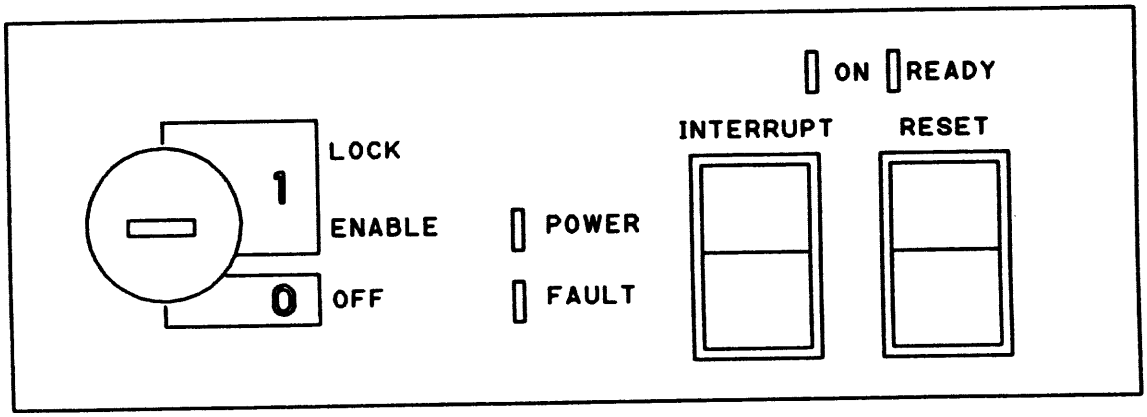


Figure 11-1. MC5600/5700 Front Panel

The following front panel features are used by the system hardware:

- **OFF, LOCK, & ENABLE** - The 3-position key control switch turns the system power supplies on and off. The LOCK position disables the RESET and INTERRUPT switches so that pressing them has no effect. The ENABLE position reenables both switches.
- **POWER LED** - The POWER LED is activated when the system power supply provides power to the +5 Volt signal on the front panel connector (see Appendix A).
- **FAULT LED** - The FAULT LED on the front panel is the logical OR of the ERROR LEDs on all CMPUs in the system. Each CMPU ERROR LED is activated by setting bit <3> to a **1** on that CMPU's PCRB (see Chapter 4).
- **INTERRUPT Switch** - Pressing the INTERRUPT switch generates a Level 7 interrupt (see Chapter 5) and clears bit <1> of the TBCCR on all CMPU modules to a **0** (see Chapter 2).
- **RESET Switch** - Pressing the RESET switch generates a Level 7 interrupt (see

Chapter 5), clears bit <2> in the TBCCR on all CMPU modules (see Chapter 2), and activates the initialization circuitry in the AFM (described in Chapter 5).

For information on how the front panel LEDs and switches are used by MASSCOMP RTU, see the *MC5600/5700 System Management Guide.*

11.2 Configuring the CMPU Module

There are four configuration considerations for installing the CMPU module:

- Setting the Processor I.D.
- Enabling the MBA
- Disabling the MULTIBUS interrupt jumper
- Configuring the MULTIBUS termination

The CMPU module also has five LED indicators that are explained in this section.

Figure 11-2 shows the CMPU module switches, jumpers, and LEDs.

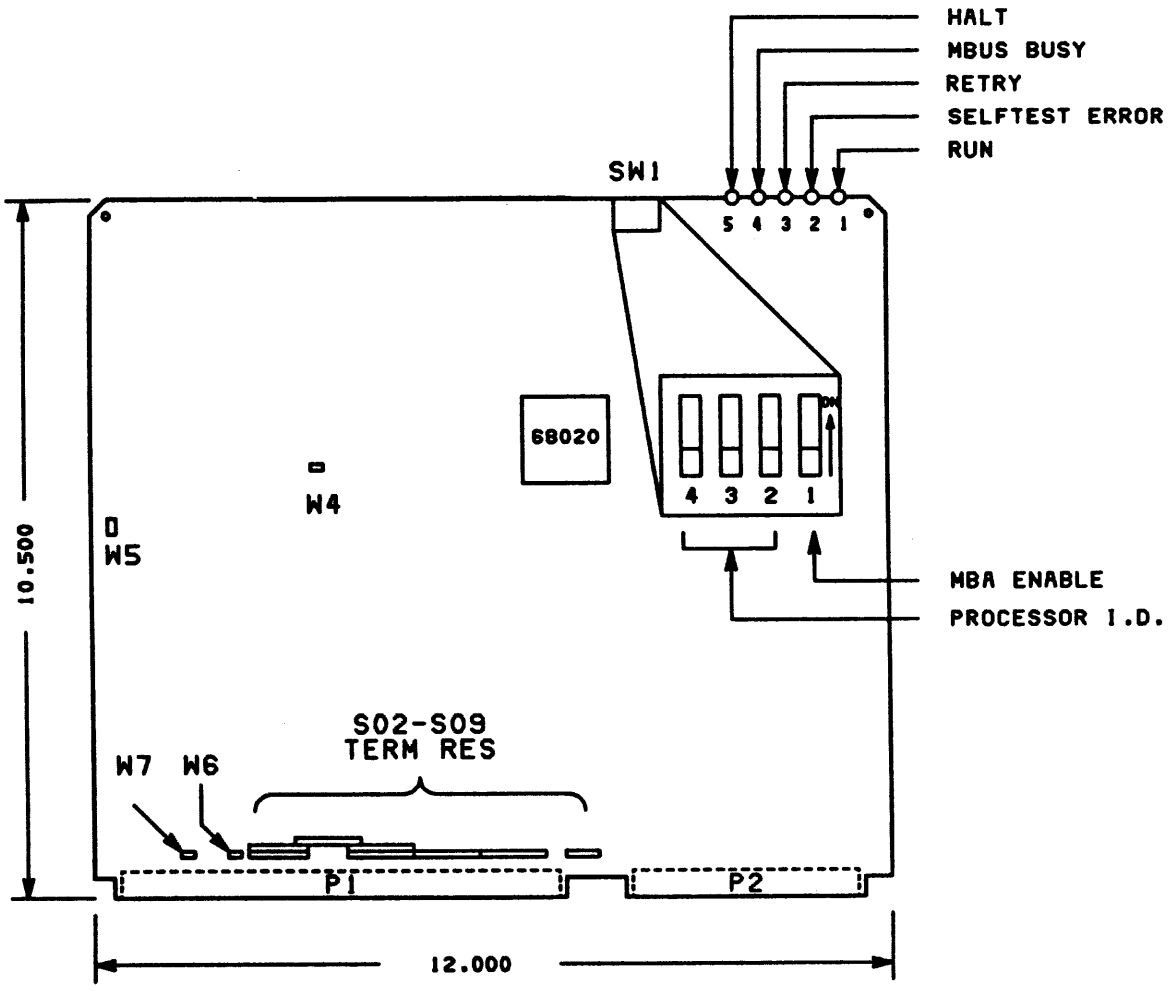


Figure 11-2. The CMPU Module

The switches on switchpack SW1 have the following functions:

Switch 1 - When switch 1 is in the ON position, the MULTIBUS Adaptor circuitry is turned ON. This switch must be ON for each CMPU module that is using an associated MULTIBUS.

Switches 2-4 - Switches 2-4 set the Processor I.D. The Processor I.D. determines the areas in the physical address space used by the associated MULTIBUS (see Chapter 8). The Processor I.D. also sets the upper bits of the SMI node I.D. (described in Chapter 7) of both the CPU and the MBA on that module (the lower bit of each circuit is determined by hardware). One CMPU in every system (the boot processor) must be set with its Processor I.D. to be 1. Table 11-2 shows the switch settings for the eight Processor I.D.'s.

Table 11-2
Setting Processor I.D. on Switchpack SW1

Processor I.D.	2	3	4
0	ON	ON	ON
1	ON	ON	OFF
2	ON	OFF	ON
3	ON	OFF	OFF
4	OFF	ON	ON
5	OFF	ON	OFF
6	OFF	OFF	ON
7	OFF	OFF	OFF

The jumpers and resistor packs are used for the following configuration purposes:

S02-S09 - These 8 resistor packs are the pullup resistors for the MULTIBUS and are factory installed. If you are installing a CMPU module without an associated MULTIBUS, these must be removed.

W1 - When this jumper is in, **MBUS LOCK2** (P1-36) performs the same function as **MBUS LOCK** (P1-25). When the jumper is out, P1-36 is not used.

W4 - This jumper is the MULTIBUS interrupt enable jumper and is factory installed. In a multi-processor system, only the CMPU with I.D. 1 should have this installed. All other processors in the system *must* have this jumper removed.

W5 - This jumper is the Loop-On-Error jumper. When this jumper is installed, any selftest that fails loops indefinitely (instead of flashing an error code on the error LED LD4). This jumper is not installed at the factory.

W6 & W7 - These jumpers are the MULTIBUS enable jumpers. These must be installed when the CMPU is using an associated MULTIBUS. They must be removed when the MBA is disabled.

The five LEDs on the CMPU indicate the following conditions when lit:

LD1 - The CMPU module is running

LD2 - The Selftest error code (see Chapter 13)

LD3 - The CMPU module is attempting a retry on the SMI

LD4 - The CMPU module is BUSY

LD5 - The CMPU module is halted

11.2.1 Adding a CMPU module

If you are adding a CMPU to your system and you also are adding an associated MULTIBUS:

1. Follow the steps in Section 11.4 for configuring the AFM and backplane to add a MULTIBUS to the system
2. Set the Processor I.D. on SW1-2 through SW1-4, as shown in Table 11-2.
3. Remove MULTIBUS interrupt enable jumper W4.
4. Install the CMPU module in the appropriate slot, as described in the *MC5600/5700 System Configuration Guide.*

If you are adding a CMPU to your system without an associated MULTIBUS (that is, with the MBA disabled):

1. Set the Processor I.D. on SW1 as shown in Table 11-2.
2. Remove MULTIBUS interrupt enable jumper W4.
3. Disable the MULTIBUS Adaptor by setting switch SW1-1 to OFF.
4. Remove MULTIBUS termination resistor packs S02 - S09.
5. Remove jumpers W6 and W7.
6. Install the CMPU module in the system, as described in the *MC5600/5700 System Configuration Guide.*

11.3 Configuring the Memory Module

This section describes how to configure a CMM module. This includes setting the base address and configuring the module as interleaved memory.

The CMM module is available in 2 MByte and 4 MByte units. The 2 MByte size is a depopulated version of the 4 MByte board.

The first CMM module in a system must be given a base address of 0x8000000. All memory must be contiguous. Larger modules must be given lower addresses in a system than smaller modules.

Figure 11-3 shows the CMM module switches and dimensions.

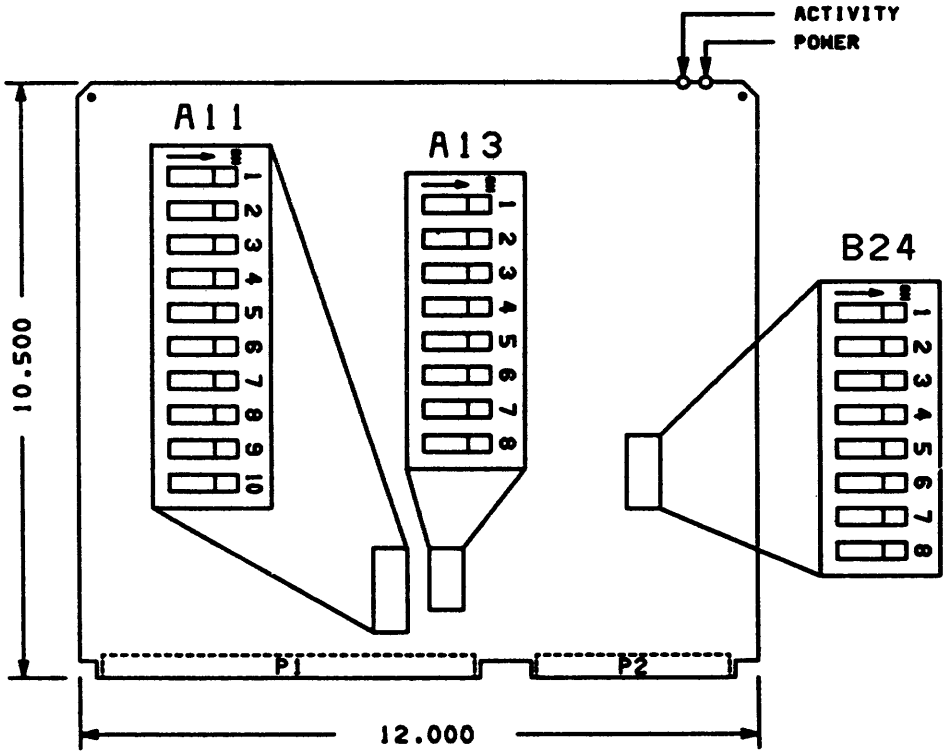


Figure 11-3. The CMM Module

11.3.1 Interleaving

Table 11-3 summarizes all switch settings for each CMM module. Switches A11-1 through A11-10, B24-1 through B24-8, and A13-1 must be set according to the size of the module (2 MB or 4 MB) and whether or not the module is interleaved. Modules that are interleaved should be of equal size.

Table 11-3
CMM Switch Configuration

Switch	Non-Interleaved		Interleaved	
	4MB	2MB	4MB	2MB
A13-1	ON	ON	ON OFF	ON (lower) OFF (upper)
A11-1	OFF	OFF	ON	OFF
A11-2	OFF	OFF	ON	ON
A11-3	OFF	OFF	ON	ON
A11-4	ON	ON	OFF	OFF
A11-5	ON	OFF	OFF	OFF
A11-6	OFF	ON	OFF	ON
A11-7	ON	ON	OFF	OFF
A11-8	OFF	OFF	OFF	OFF
A11-9	OFF	OFF	ON	ON
A11-10	ON	ON	OFF	OFF
B24-1	ON	ON	OFF	ON
B24-2	OFF	OFF	ON	OFF
B24-3	OFF	OFF	OFF	OFF
B24-4	OFF	ON	OFF	OFF
B24-5	ON	OFF	ON	ON
B24-6	OFF	OFF	OFF	OFF
B24-7	OFF	OFF	OFF	OFF
B24-8	ON	ON	ON	ON

11.3.2 Setting Module Base Address

You set the base address for each module in the system at switches A13-2 through A13-8. The memory space is assigned to the upper half of the system physical address space, while the module's CSRs are assigned to a separate range within the SMI Device space (see Chapter 8). Setting the base address for a given module also determines the physical address of its CSRs.

Table 11-4 shows how switch A13 sets the base address for the module and its two CSRs.

Table 11-4
Setting Module Base Address

SMI Address Bit	CMM Base Address Bit	MCR Address Bit	SCBR Address Bit
<27>	1	0	0
<26>	A13-2	0	0
<25>	A13-3	0	0
<24>	A13-4	0	0
<23>	A13-5	0	0
<22>	A13-6	0	0
<21>	A13-7	0	0
<20>	A13-8	1	1
<19>	0	A13-2	
<18>	0	A13-3	
<17>	0	A13-4	
<16>	0	A13-5	
<15>	0	A13-6	
<14>	0	A13-7	
<13>	0	A13-8	
<12>	0	(A13-1)	
<11>	0	1	0
<10:00>	0	*x*	*x*

Switch A13 determines the address of each device using the following algorithms:

CMM Base Address. Address bit <27> for all CMM modules is fixed at **1** (the upper half of the address space). Switches A13-2 through A13-8 set SMI address bits <26:20> respectively, where ON sets the address bit to a **0** and OFF sets the bit to a **1**.

MCR & SCBR addresses. SMI Address bits <27:20> are fixed to select the Memory module CSR address range, described in Chapter 8. Switches A13-2 through A13-8 set SMI address bits <19:13> respectively, to select which module's CSR is addressed. Toggling the switch ON sets the address bit to a **0** and OFF sets the bit to a **1**. Address bit <12> is set to a **1** if the module is interleaved and upper half; otherwise, it is **0** (the inverse of A13-1). Address bit <11> is used to select between the MCR (<11>=1) or the SCBR (<11>=0). Bits <10:00> are ignored when accessing a memory module CSR.

Table 11-5 shows an example of a system with four 4MB interleaved CMM modules.

Table 11-5
Example of Setting CMM Base Address

Module	Base Address	Top Address	MCR Address	SCBR Address
0	0x8000000	0x87FFFF7	0x0100800	0x0100000
1	0x8000000	0x87FFFFF	0x0101800	0x0101000
2	0x8800000	0x8FFFFF7	0x0110800	0x0110000
3	0x8800000	0x8FFFFFF	0x0111800	0x0111000

11.4 Configuring the AFM and Backplane

This section describes how to change the number of MULTIBUSs in a system. The Auxiliary Function / Arbitration Module (AFM/ARB) and the system backplane each hold removable termination resistors that must be configured for proper bus operation. The backplane also has jumpers and cable connectors that allow an existing MULTIBUS configuration to be changed without extensive hardware modifications.

11.4.1 The AFM Module

Figure 11-4 shows the AFM/ARB jumpers and LEDs.

NOTE

Figure 11-4 shows the AFM2/ARB2. Earlier versions of the AFM/ARB use slightly different jumper locations and do not have testpoints.

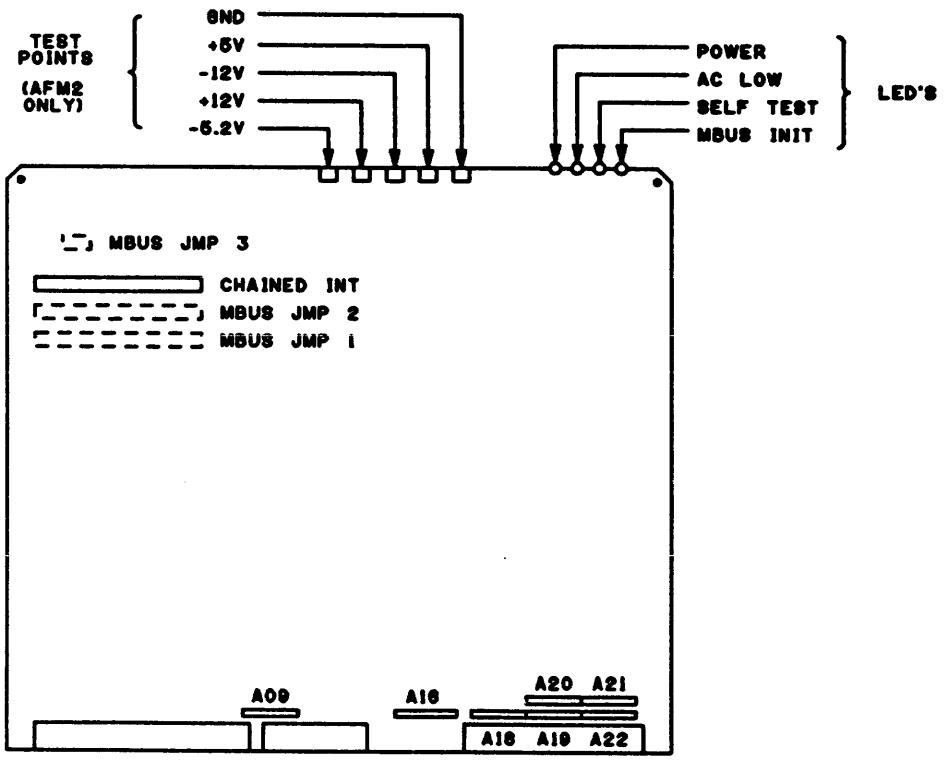


Figure 11-4. The Auxiliary Function Module

The AFM has several locations on the board used for system configuration:

A9, A16, and A18-22 - These are the removable resistor packs used for rear end termination on the SMI bus. The front end SMI termination is located on the backplane (described later in this chapter).

MBUS JMP 1-3 - These pins hold the 15 slot backplane MULTIBUS jumpers when they have been removed from the backplane and are not in use.

CHAINED INT - This location holds the backplane chained interrupt jumper when not in use. This jumper is only used when the backplane is configured as a 7/8 slot backplane.

The four LEDs on the AFM indicate the following conditions when lit:

LD1 - Power is being supplied to the AFM module (and the MULTIBUS)

LD2 - MULTIBUS signal AC LO is asserted

LD3 - This LED is ON until the AFM internal selftest (which is started on every powerup) has finished without errors. If the selftest fails, the LED remains on indefinitely.

LD4 - MULTIBUS signal MBUS INIT is asserted

The five testpoints shown in Figure 11-4 (AFM2/ARB2 only) can be used to verify that D.C. voltage levels are present on the MULTIBUS.

11.4.2 The Backplane

Figure 11-5 shows the backplane jumpers and connectors used in system configuration.

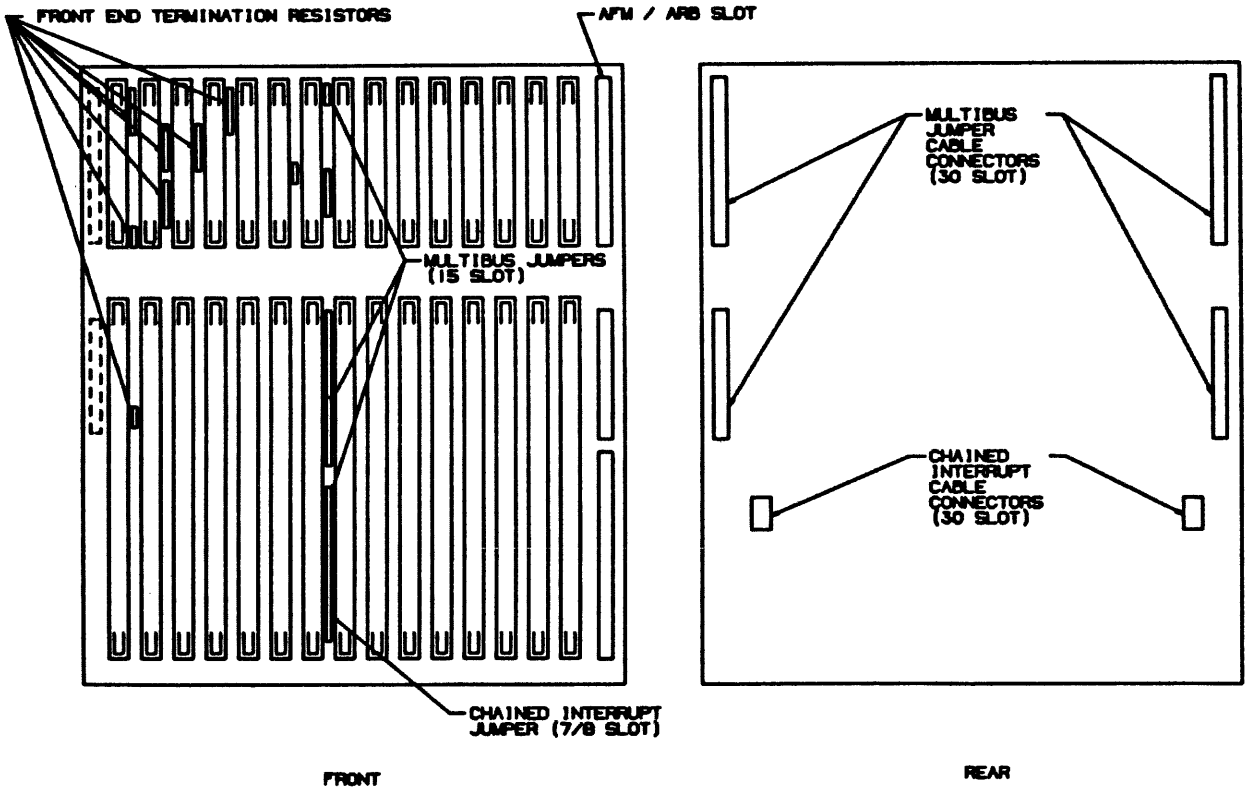


Figure 11-5. The 15 Slot Backplane

The backplane has the following elements that are used for configuration:

Front End Termination Resistors - The backplane has 5 resistor packs used for front end SMI termination and 2 packs used for front end MULTIBUS and SMI clock termination. In the case of a 30-slot system, all front end termination on the second backplane (near slot 16) must be removed.

MULTIBUS Jumpers (15 Slot) - Three jumpers are used to link the MULTIBUS signals across the entire backplane in a 15 slot configuration. In a 7/8 slot configuration, these jumpers are removed to divide the backplane into two separate buses.

MULTIBUS Jumper Cables (30 Slot) - Two cables are used to link the SMI signals across the two 15 slot backplanes in a 30-slot configuration.

Chained Interrupt Jumper (7/8 Slot) - All MULTIBUSs in the system use the same interrupt lines, as explained in Chapter 5. A special jumper links the interrupt lines in a 7/8 slot backplane. This jumper is stored on the AFM when not in use.

Chained Interrupt Cable (30 Slot) - A special cable is used in a 30-slot configuration to link the interrupt lines between the two 15-slot backplanes. This cable must be in whether the configuration uses 2, 3, or 4 MULTIBUSs.

Common Ground Jumper (30 Slot) - A metal jumper is installed to ensure the two backplanes in a 30-slot configuration are at the same ground potential.

Figure 11-6 shows the cable connections used in a 30-slot backplane.

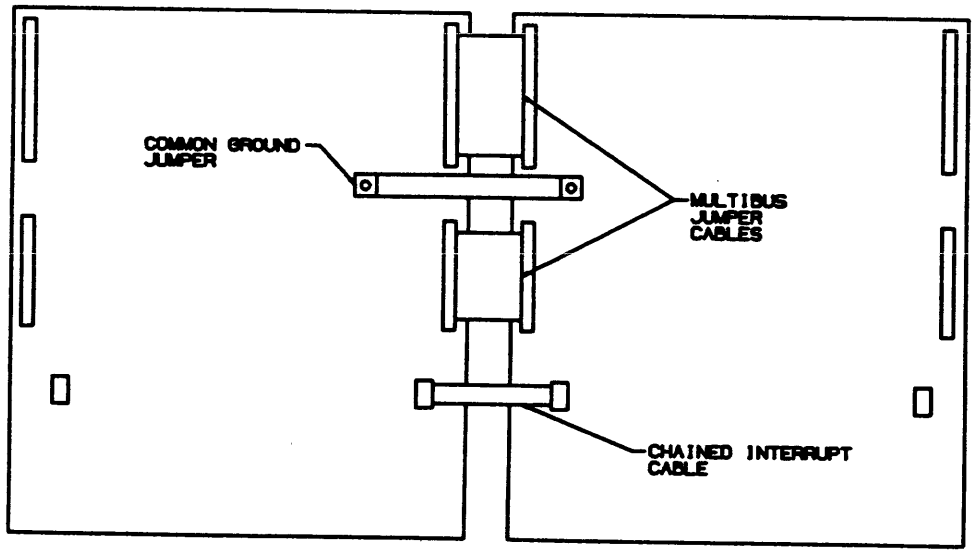


Figure 11-6. 30 Slot Backplane

11.4.3 Adding MULTIBUSs to a System

Table 11-6 summarizes the elements used to properly configure multiple MULTIBUSs in a system.

Table 11-6
Configuring the AFM & Backplane

Config-uration	Rear End Termination (AFM)	Front End Termination (Backplane)	MBUS Jumpers (15 Slot)	MBUS Cables (30 Slot)	Chained Interrupt
15-slot	In	In	In	Not Used	Not Used
15 to 7/8 slot	In	In	Remove & Store	Not Used	Install Jumper
(2) 15-slots to (1) 30-slot	Remove Termination	Remove from 2nd Backplane	In	Install (2) Between Backplanes	Install Cable
(1) 30-slot to (2) 7/8 slots	Remove Termination	Remove from 2nd Backplane	Remove from both Backplanes & Store	Install (2) Between Backplanes	Install Cable & (2) Jumpers

Termination resistors are required to prevent signal reflection on the entire SMI bus and the MULTIBUS clock lines. Two sets of 7 resistor packs reside on the system:

- one set near the first slot (1) on the bus (**front end termination**)
- one set on the last slot (**rear end termination**)

The general rule for configuring termination is that there must be both front and rear end termination on the SMI and on the MULTIBUS BCLK and CCLK signals. The front end termination set in the MC5600/5700 is located on the backplane itself. The rear end termination set is located on the AFM or ARB module, depending on the configuration (as explained below). Note that, since the MULTIBUS BCLK and CCLK clock signals span the entire backplane and are used by all MULTIBUSs, they follow the same configuration rules as the SMI signals.

The Arbitration Module is a depopulated AFM containing only the rear end termination circuits and the arbitration logic for a third and fourth MULTIBUS. The AFM and ARB Modules are used together as follows:

- **One card cage.** In 15 or 7/8 slot systems, only the AFM is used, with all termination in place.
- **Two card cages.** In 30 slot systems, the AFM is used with all termination resistors removed. The AFM is placed at the end of card cage 1 (between slot 15 and 16) and an ARB Module with all termination resistors installed is placed at the end of card cage 2 (after slot 30).

Chapter 12
The EPROM Bootstrap

ILLUSTRATIONS

TABLES

Chapter 12
The EPROM Bootstrap

When your system is powered up, the operating system must first be loaded into main memory from an external device (a disk or tape) before the system can begin normal operation. Thus, a small amount of code must be kept on board in a non-volatile memory space that instructs the CPU how to retrieve the operating system and start it running.

The CMPU module has a 64 KByte Erasable Programmable Read Only Memory (EPROM) which contains four distinct groups of code that are used to execute:

- **Selftest diagnostics.** The selftest code tests essential hardware devices for proper operation after powerup.
- **Hardware initialization.** The hardware initialization code resets appropriate registers to known initial states.
- **Bootstrap code.** The bootstrap code retrieves the operating system code from an external device and then relinquishes control of the CPU to the OS. This code includes subroutines that may be called by programs running in memory.
- **Console mode.** The console code is an optional operating mode that used for debugging, manual control over bootstrap procedures and low-level diagnostics, and is described in Chapter 13.

This chapter describes the selftests and the standard powerup bootstrap sequence. It also describes the programming considerations for writing your own bootstrap device code and the callable subroutines resident in the EPROM. These programming considerations include the method of accessing the NVRAM and Time of Day clock on the AFM.

12.1 Powerup Sequence

This section summarizes the flow of operations during a 5600/5700 powerup routine. The bootstrap flow is shown in Figure 12-1. In a multiprocessor system, the bootstrap code is executed by all processors simultaneously until the boot/non-boot processor determinations are made and synchronization occurs (step 13). The diagnostic selftests are described in detail in the next section.

1. **Power is turned on**, or
   The front panel RESET switch has been pressed
2. **Initialize DUARTs and Processor Control Registers A & B.** The console screen is blank at this point
3. **Check if RESET switch on front panel has been pressed**
   A. If RESET has been pressed and CPU is boot processor, go to step 7
   B. If RESET has been pressed and CPU is not boot processor, go to step 13

   If RESET has not been pressed, continue.

4. **Perform selftest diagnostics 0x1 through 0x9.** Each selftest number appears before the test begins on the console device, if it is on one of the 3 DUART ports (but not if it is a graphics terminal). Numbers 0x1 - 0x4 appear on TTY0 only. In the event of a failure of test 0x5 or higher, the bootstrap halts and flashes an error code on the error LED of the CMPU that failed (see Section 12.2).

5. **Check if current CMPU is boot processor (Processor I.D. #1)**
   If boot processor, continue selftests
   If not boot processor, go to step 12

6. **Complete hardware selftest.** Perform selftest diagnostics 0xA through 0x17 (if GPM or GCM is not present, do not print 0x10 through 0x16). In the event of a failure, the bootstrap will halt and flash an error code on the error LED of the CMPU that failed (see Section 12.2). Also, instruct any other non-boot CMPU modules to perform selftests 0xB-0xE. Wait for these processors to finish step 13. If no other CMPUs are present, continue.

7. **Read and check for valid NVRAM data.**
   If invalid, use the default environment parameters from the EPROM code to deposit into main memory and NVRAM, and go to step 11.
   If valid, continue.

8. **Check that INTERRUPT switch on front panel has been not pressed.**
   If INTERRUPT has been pressed, use all NVRAM parameters except the Console TTY value and baud rate, and go to step 11.
   If INTERRUPT has not been pressed, continue.

9. **Use all NVRAM parameters.**

10. **Check if AUTOBOOT is set.**
    If AUTOBOOT is set, go to BOOT code in boot processor's EPROM
    If AUTOBOOT is not set, go to CONSOLE code in boot processor's EPROM

11. **If INTERRUPT was pressed or invalid NVRAM:** Select console port from operator and the baud rate using autobaud. Go to CONSOLE code in EPROM.

12. **Perform non-boot processor selftests:**
    Wait for boot processor to finish step 6.
    When instructed by boot processor, perform selftests 0xB through 0xE (non-boot processors do not do selftests 0x10 through 0x17). Print the test numbers on non-boot processor DUARTs.
    Go to step 13.

13. **Perform non-boot processor functions:**
    Wait for boot processor to synchronize.
    Turn on access to memory.
    Initialize globals.
    Copy exception vectors from EPROM to main memory.
    Set up Vector Base Register to point to main memory vectors.
    Tell boot processor that selftests are complete.
    Set the Interrupt Process Level to 7 and set IPIR vector 7 to point to the Console code entry point.
    Execute a STOP instruction.

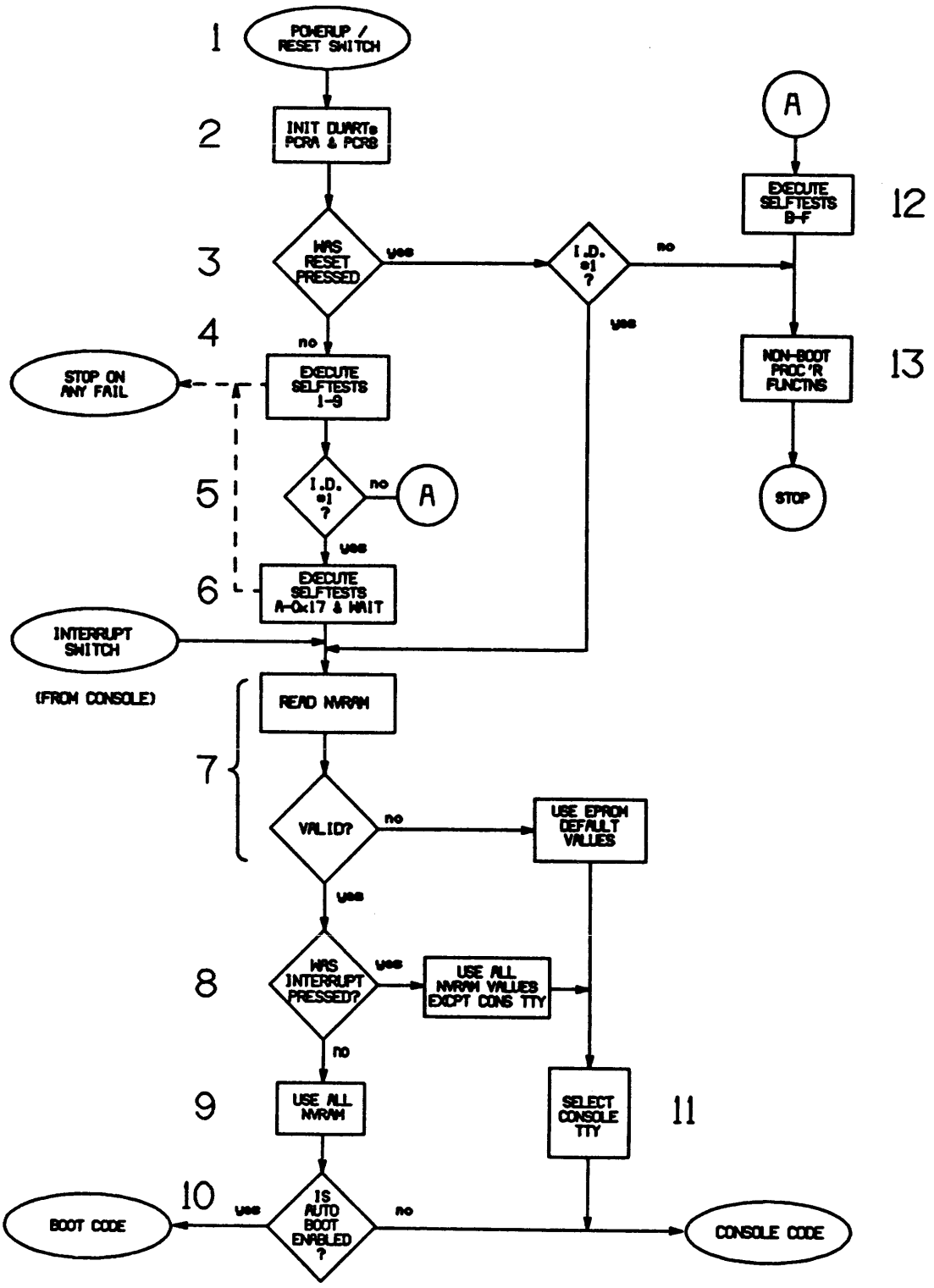


Figure 12-1. Bootstrap Flowchart

12.2 Selftest Diagnostics

This section explains each of the selftest diagnostics and how to interpret diagnostic error LEDs on the CMPU and front panel.

When the CMPU selftest diagnostics are running, a test identification number is printed out on the console *before* each test is run. There are 23 diagnostic tests, numbered 1 through 17 hexadecimal. If the selftest finishes successfully, the TTY0 screen should show the following:

```
1,2,3,4,05,06,07,08,09,0A,0B,...0C,0D,0E,0F,10,11,12,13,14,15,16,17
V x.x mm/dd/yy PROCESSOR ID = n
```

Each dot displayed after test **0B** represents 128 KByte of memory tested. The TTY1 and TTY2 screens should display all but the first four test numbers (0x5-0x16). The TTY4 screen (graphics terminal port) displays only the number 17. If your system does not have a graphics memory board, tests 0x10 through 0x16 are NOT printed. Table 12-1 gives a summary of the tests performed during powerup diagnostics. The version number, release date, and the processor I.D. running the bootstrap follow the selftest printout.

12.2.1 Diagnostic Error Codes

Before beginning selftest 1, the bootstrap verifies the EPROM checksums and checks the read and write operations on DUART 0.

1. If the EPROM checksum is incorrect, the 68020 halts and all error LEDs are turned on indefinitely.
2. If the DUART test fails, the 68020 goes into an infinite loop writing and reading back the DUART register. So, if no numbers at all appear on the TTY0 screen and the error LED is off, this indicates that the DUART test has failed.
3. If one of the selftest diagnostics fails during powerup, the console printout stops at the test that fails. If the failed test is 0x5 or higher, the error LEDs on both the CMPU module and on the system front panel also flash an error code. Each LED flashes *n* times for test number *n*. Each LED then goes out for 5 seconds and repeats the sequence indefinitely.

So, for example, if the console shows `1,2,3,4,5,6` and the CMPU error LED flashes 6 times and pauses, this indicates that Test 6 (MULTIBUS Map Integrity) has failed.

The error code should always be read on each CMPU module in a multiprocessor system (see Chapter 11). The system fault LED on the front panel is derived from the error signals from all CMPU modules OR'd together. This LED is only designed to indicate an error on *at least one* CMPU and does not necessarily flash a readable code.

The Loop On Error jumper, when installed on the CMPU board, forces any failing tests to loop indefinitely instead of flashing the error LED. This jumper is not factory installed. See Chapter 11 for instructions on installing this jumper.

Table 12-1
Powerup Diagnostics

TEST	DESCRIPTION
1	Cache Data Integrity
2	Cache Address Test 1
3	Cache Address Test 2
4	Cache Byte Write
5	MULTIBUS Map Access
6	MULTIBUS Map Integrity
7	MULTIBUS Map Address Integrity 1
8	MULTIBUS Map Address Integrity 2
9	Memory 0 CSR Response
A	SMI Data Integrity & Memory Init
B	Memory Data Integrity 1
C	Memory Address Integrity 1
D	Memory Address Integrity 2
E	Memory Byte Write
F	Check For GPM/GCM On System
	GPM (OR GPM & GCM) PRESENT
10	GPM/GPX Register Data Integrity
11	GPM/GPX Extended Tests
12	Graphics Read From Main Memory
13	Graphics Write To Main Memory
14	Graphics To MULTIBUS Data Integrity
15	Graphics To MULTIBUS Address Integrity
17	Non-boot Processor Memory (Tests B-E by non-boot CPUs)
	GCM ONLY PRESENT
10	MULTIBUS read to word **0**
11	MULTIBUS read to word **1**
12	MULTIBUS Block mode read
13	MULTIBUS write to word **0**
14	MULTIBUS write to word **1**
15	MULTIBUS Block mode write
16	MULTIBUS Address test
17	Non-boot Processor Memory (Tests B-E by non-boot CPUs)

12.3 Boot Sequence

Once the selftest diagnostics have run successfully, and if the powerup sequence has not been diverted to console mode, the **boot device driver** in the EPROM is invoked. The driver has the sole task of reading the **boot block** (or **primary bootstrap**) from the **boot device** (usually a disk or tape) into main memory, and executing this code. The boot block then reads the

target boot file (usually the operating system) into memory and starts executing it. The full boot sequence may be summarized as follows:

1. Read and verify the geometry block from the boot device
2. Read and start the primary bootstrap pointed to by the geometry block
3. Read and start the secondary bootstrap (the target file) as specified by command line arguments or default flags

The MASSCOMP driver in the EPROM performs steps 1 and 2. The MASSCOMP primary bootstrap then performs step 3 using routines in the EPROM or in the customer boot space.

If any of these steps detects an error, a message is printed on the console terminal and the system enters console mode (described in Chapter 13).

12.4 Customer Boot Space

The MC5600/5700 has 8175 bytes of memory space reserved in the NVRAM (on the Auxiliary Function Module) for bootstrap driver code to support non-MASSCOMP boot devices. This section describes the programming considerations when writing your own boot code.

During system bootstrap, the EPROM first determines whether the bootstrap device driver is contained in the customer NVRAM. If so, the EPROM uses that driver to perform the boot sequence, *even if the EPROM also contains a driver for the device.* The NVRAM driver can be disabled by using the **boot** command in console mode with the **/nr** qualifier (see Chapter 13). If a driver is not present in NVRAM, the system boots using the device driver in the EPROM for the device specified in the NVRAM environment or on the **boot** command line.

If the driver is present in NVRAM, it is first copied into main memory at physical address 0x08008000, where it is called by the bootstrap. There are 512 bytes of main memory address space, starting at offset 0xA00 in the boot CMPU's console page (physical address 0x08001A00), which are reserved for the BSS segment of the customer boot driver. This means that the program contained in the customer boot device must be linked at the appropriate base addresses with the *ld* command (from UNIX):

```
% ld -T 08008000 -B 08001A00 file(s)
```

The primary bootstrap is read from the boot device into memory at address 0x0800A000. This bootstrap should not destroy the 8 KBytes of memory that contain the device driver at 0x08008000 if it plans on using the driver.

The geometry block is read into memory reserved for the MASSCOMP ROM at address 0x08001800, keeping it available for the entire boot process.

12.5 General Purpose Callable Subroutines

This section describes the EPROMs callable entry points. These are general purpose subroutines that facilitate interfacing to the hardware. Most of them are used by some part of the EPROM bootstrap code. The information given here allows you as a programmer to have independent access to them. For each subroutine, a brief functional specification and the address of a pointer to it are given.

These subroutines follow the C-language conventions of:

1. Returning values in 68020 register D0
2. Assuming all parameters are four bytes in length
3. Saving the contents of all 68020 registers except D0, D1, A0, and A1

These routines use the stack and various locations in the console's reserved page in main memory (see Chapter 13).

The EPROM code is written to be able to run in a mapped system (that is, a system with memory management hardware enabled and the virtual address space 0xF0000000 — 0xFFFFFFFF mapped to physical address space 0 — 0x0FFFFFFF).

Each subroutine name is first cast to the virtual address of the pointer to the subroutine (F0000*xxx*). The subroutine is then shown in the form used to call it in the program.

12.5.1 check_for_received_char

The **check_for_received_char** subroutine tests the keyboard receiver on the currently active console port for the presence of a character at the keyboard. It is called:

```
#define check_for_received_char(r)  (*(char(**)())0xF0000420)(r)
     int raw;
     check_for_received_char(raw);
```

If no character is present, the function returns a zero (0). If there is a character present and the `raw` argument is non-zero, the character is returned. If the `raw` argument is zero, the function checks if the character is a CTRL-C (3), CTRL-O (15), CTRL-Q (17) or CTRL-S (19), which affect the flow control flags. The function handles these characters as described in Chapter 13. Otherwise, the function returns the character.

12.5.2 console_no_init

The **console_no_init** subroutine is used to return to EPROM console mode without initializing the console variables. This entry point can be used only if the console's reserved memory space has not been modified (see Chapter 13). This entry point remembers any breakpoints that may have been placed into memory. It is called:

```
#define console_no_init()  (*(int(**)())0xF0000414)()
console_no_init();
```

12.5.3 put_char

The **put_char** subroutine prints a character on the active terminal. It is called:

```
#define put_char(d,r)  (*(int(**)())0xF0000424)(d,r)
     char data;
     int raw;
     put_char(data,raw);
```

If the `raw` argument is non-zero, the function ignores the flow control flags set with `check_for_received_char`. Otherwise, the control flags determine how the data argument is handled.

12.5.4 put_number

The **put_number** subroutine converts a number from binary to an ASCII hexadecimal string and prints it on the active terminal using the **put_char** function with the **raw** flag set to **0**. The size of the number is specified in nibbles (1 through 8). It is called:

```
#define put_number(n,s)  (*(int(**)())0xF0000410)(n,s)
     int number, size;
     put_number(number,size);
```

12.5.5 put_string

The **put_string** subroutine prints a zero-terminated character string on the active terminal using the **put_char** function with the `raw` flag set to **0**. It is called:

```
#define put_string(s)  (*(int(**)())0xF000040C)(s)
     0xchar *string;
     put_string(string);
```

12.5.6 reboot

The **reboot** subroutine is used to boot from a device or file specified in *string*. The command can be run *only* on the Boot processor (Processor I.D. 1). It is called:

```
#define reboot(s)  (*(int(**)())0xF0000418)(s)
     char *string;
     reboot (string);
```

The `string` parameter is a virtual pointer to a command line string of 131 characters or less, beginning with the *device-specification* argument (the command line syntax is specified in Chapter 13 under the **boot** command). The EPROM copies this string into its memory space, disables memory management, initializes the system, and performs the boot sequence as if the user had typed the **boot** command from console mode.

If an error occurs during the execution of **reboot**, an error message is printed on the console terminal and the system enters console mode.

12.5.7 non_boot_console

The **non_boot_console** subroutine is used to return to EPROM console mode on a non-boot processor without initializing the console variables. It is called:

```
#define non_boot_console()  (*(int(**)())0xF000041C)()
     non_boot_console();
```

This entry point can only be used if the console's reserved memory space has not been modified (see Chapter 13). This entry point remembers any breakpoints that may have been placed in memory.

12.5.8 setmap

The **setmap** subroutine loads the MULTIBUS I/O maps with a specified mapping. It is called:

```
#define setmap(m,s,i)  (*(char*(**)())0xF0000440)(m,s,i)
     int size, id;
     char *memory;
     setmap(memory,size,id);
```

The `memory` argument is a pointer to the start of the memory buffer to be mapped. The `size` argument is the number of bytes in the buffer. The `id` argument is the MULTIBUS number whose maps are being written.

The routine returns a pointer to the start of the buffer in the MULTIBUS address space. If the CMPU with the specified I.D. does not exist or if the buffer is too large, the routine returns a **-1**.

12.5.9 enable_gcm_terminal

The **enable_gcm_terminal** subroutine is used to test for the presence of a GCM graphics processor on MULTIBUS **1**, or to enable ASCII terminal mode on that processor. It is called:

```
#define enable_gcm_terminal(f)  (*(int(**)())0xF0000444)(f)
     int flag;
     enable_gcm_terminal(flag);
```

The valid values for `flag` are as follows:

0 - Test for presence of a GCM module. If present, test to see if ASCII terminal mode is enabled.

1 - Enable ASCII terminal mode.

2 - Enable ASCII terminal mode only if the active terminal is on the GCM processor.

The routine returns the following values:

1 - A GCM processor is present and ASCII terminal mode is enabled.

0 - A GCM processor is present but ASCII terminal mode is not enabled.

-1 - No GCM processor is present

12.5.10 probe_address

The **probe_address** subroutine is used to determine if an address responds to a read or write operation. It is called:

```
#define probe_address(a,d,f) (*(int(**)())0xF000043C)(a,d,f)
     data_type *address;
     int data_type, function;
     probe_address (address, data_type, function);
```

The `data_type` is encoded as follows:

0 - char
1 - short
2 - long

If `function` is a **0**, a read operation is performed. Otherwise, a write is performed with a data value of **0**. This routine returns the value **0** if the address responds and the value **-1** if there is no response. The routine must be called with the 68020 in supervisor mode.

12.5.11 reset_smi

The **reset_smi** subroutine is used to reset the MC5600/5700 system. The command can *only* be run on the Boot processor (Processor I.D. 1). It is called:

```
#define reset_smi() (*(int(**)())0xF0000428)()
     reset_smi();
```

The subroutine causes the following sequence of events to occur:

1. Execute a 68020 RESET instruction
2. Enable the graphics terminal's ASCII terminal mode
3. Wait for the non-boot processors to initialize

12.6 CMPU-AFM Serial Interface

This section describes the interface and communication protocol between the CMPU and the Auxiliary Function Module (AFM). The AFM has a dedicated microprocessor for transmitting its data to the boot CMPU and, through this, to the system. This microprocessor can read and write the Time of Day Clock and the NVRAM on command. A small on-board EPROM contains code to run the AFM microprocessor. The device used to communicate with the AFM is CMPU serial port **P1B** (described in Chapter 4).

The CMPU can initiate a data transmission between it and the AFM by sending a function code character as the first word of a serial transmission. Each AFM function (time of day, system I.D., user boot code, and flags) is assigned a unique function code. Once this control character is identified, the AFM microprocessor executes the appropriate task and transmits the requested data or writes the transmitted data.

Data being received by the AFM is limited to 64 bytes due to the limited amount of RAM in the AFM's microprocessor. Data transmitted to the CMPU has a 256 byte limit.

12.6.1 AFM Serial Communication Format

The serial line connecting the AFM to the CMPU uses the following communication format:

Baud-Rate	38.4K
Start Bits	1
Stop Bits	1
Data Bits	8
Parity	None

To communicate with the AFM, the CMPU's 68020 sends a packet of 2 or more characters out of its serial port P1B. A packet consists of:

- A Function code (1 byte), followed by
- A Count (1 byte), followed by
- Optional Address and Data characters

When the Function and Count characters are received by the AFM, the packet is checked for validity. If the packet is valid, an acknowledge (ACK) code is returned. An ACK response is the single character 0x41 (the ASCII "A"). This response indicates that the AFM has successfully processed either the Function code, the Count/Address, or the Data.

If the Function character is unrecognizable or the Count/Address field is too large, the AFM returns an ERROR code. The ERROR code is the single character 0x45 (the ASCII "E"). If an ERROR code is retuned, the CPU must inhibit sending the rest of the packet.

The flow of operations for reading and writing AFM functions are essentially the same, regardless of which function is the target.

To write an AFM function, the CPU:

1. Sends a 1-byte Function code, followed by
2. A 1-byte count (*cc*) within the range shown in Table 12-2
3. Waits for an ACK or ERROR code to be returned
4. Sends a 1- or 2-byte starting address (if appropriate)
5. Waits for ACK or ERROR to be returned
   - If an ERROR code is received, then either the starting address or the byte count is too large. Sending the rest of the data must be inhibited at this point.
   - If ACK, sends the AFM *cc* bytes of data
6. Waits for ACK to be returned, indicating that all data has been processed by the AFM processor (this takes approximately 30 milliseconds per character for the NVRAM)

To read an AFM function, the CPU:

1. Sends a 1-byte Function code, followed by

2. A 1-byte count character (*cc*) in the range shown in Table 12-2
3. Waits for ACK or ERROR to be returned
4. Sends a 1- or 2-byte starting address (if appropriate)
5. Waits for ACK or ERROR
6. If not an ERROR, the AFM then returns *cc* bytes of data.

Table 12-2 summarizes the format of each packet used to communicate with the AFM.

Table 12-2
AFM Serial Communications Format

Function		Packet Format			
Target	R/W	Function Code (1 Byte)	Count *cc* Range (1 Byte)	Address Range	Data
Bootstrap	W	0x46	1 - 64	2 Bytes High Byte, Low Byte (0-8174)	*cc* bytes
	R	0x06	1 - 255	2 Bytes High Byte, Low Byte (0-8174)	-
Boot Flags	W	0x42	1 - 8	1 Byte (0 - 7)	*cc* bytes
	R	0x02	1 - 8	0 - 7	-
System I.D.	W*	-	-	-	-
	R	0x04	1 - 8	0 - 7	-
Clock Chip	W	0x40	1 - 64	1 Byte (0 - 63)	*cc* bytes
	R	0x00	1 - 64	1 Byte (0 - 63)	-
Diagnostic Byte	W	0x48	Don't Care	-	1 Byte
	R	0x08	Don't Care	-	-

* The system I.D. is read-only

The following describes the AFM functions that may be accessed using the serial format:

Bootstrap. Customer bootstrap software may be stored in the NVRAM. The size of the bootstrap NVRAM is 8175 bytes and the 2-byte address range is 0x0 - 0x1FEF. Section 12.4 describes how this NVRAM code is used by the MASSCOMP bootstrap.

Boot Flags. The software boot flags are stored in 8 bytes of the NVRAM. The flags are used to control the console on powerup initialization and can be modified by various console commands (see Chapter 13).

System I.D.. The system I.D. is an 8 byte read-only number assigned to the system by MASSCOMP at the factory. The console mode password, if present, is also encrypted into this field. The system I.D. may be read in console mode as part of the environment (see Chapter 13). If a write is attempted, the AFM returns ERROR.

Time of Day Clock. The AFM uses a Motorola MC146818A chip to provide the system's time of day clock. The AFM interface to the clock chip models the chip as a 64 byte RAM.

The operation and programming of this chip is described in the specification sheet referenced in the preface of this manual.

Diagnostic Byte. This is a spare byte on the NVRAM used to verify its reading and writing capabilities.

Chapter 13
The Console

TABLES

Chapter 13
The Console

The console mode is a specialized operating mode in which the processor executes code in the EPROM, rather than code in main memory. The console mode is designed to facilitate hardware debugging and other operations. It may be used for:

- Manual control over the bootstrap process
- Hardware level diagnosis
- Program debugging
- Performing system memory dump
- Performing selftest and initialization as discrete operations

13.1 Entering Console Mode

To enter console mode from the current console device:

- **From UNIX**: Type **`reboot -h`**. This leaves memory mangement enabled. The terminal port specified in the NVRAM on the Auxiliary Function Module is used as the console device (this default may be changed, as described later in this chapter).
- **From powerup**:
  - If the system autoboots to UNIX, wait for it to finish and then type **`reboot -h`**.
  - If the autoboot environment flag is set to be off, the system enters console mode automatically after performing its powerup selftests.

You are in console mode when you see the `>>>` prompt.

13.1.1 Console Mode Password Protection

Some systems are configured with password protection for console mode. In this case, the system issues the prompt before entering console mode:

```
Password:
```

You must type the password matching the one stored in the NVRAM by the system administrator before you can enter console mode. Typed characters are not echoed. If the password is entered incorrectly, the prompt is reissued until the correct one is entered.

For information on changing the console password, contact your MASSCOMP service representative.

13.1.2 Changing the Current Console Device

If you wish to use console mode at a terminal that is not currently the console device, press the **INTERRUPT** switch on the front panel of your system. The **autobaud** feature in the console software then attempts to determine the baud rate of any terminal on which it detects a transmitted character. The system shows the prompt `HIT RETURN TO PROMPT` on all terminals transmitting at the baud rate indicated by the asterisk (*) in Table 13-1.

Table 13-1
Console Baud Rates

Terminal	Baud Rate		
	1 <CR>	2 <CR>	3 <CR>
TTY 0 TTY 2	19200* 9600 7200 4800	2400 1200	300
TTY 1	38400* 9600 7200	4800 2400 1200	

If your terminal is transmitting at one of the baud rates shown in column 1 in Table 13-1, pressing one <CR> at one of the terminals selects that terminal to be the console device. If your terminal is transmitting at one of the baud rates shown in column 2 in Table 13-1, pressing <CR> *twice* at one of the terminals selects that terminal to be the console device. Setting the console device baud rate to 300 requires pressing <CR> *three* times.

13.1.3 Using Console Mode for Debugging

While console mode is most often used in system maintenance, it is also very useful for debugging programs or pinpointing hardware problems. Typically, a console debugging session includes the following steps:

1. Load your diagnostic program into main memory, using either the boot command (to load an entire program) or the deposit command (to manually load the program one instruction at a time).
2. Run the program until it reaches an error or ends successfully.
3. At any point in console mode, you can set breakpoints in the program, examine and change the contents of 68020 registers, and continue program execution from the halt point. A single step and a data copying facility are also available.

13.1.4 Main Memory Pages Reserved for Console

The first ten 4 KByte pages of system memory address space are reserved by convention for use by the code in the CMPU EPROM. The first 32 Kbytes (0x8000000 — 0x8007FFF) are reserved for use by the console code. The second 4 KByte page is reserved for the EPROM on processor I.D. 1 (Boot CPU). The first and third through seventh pages are reserved for non-boot CMPU EPROMs and future expansion. The next two pages (0x8008000 — 0x8009FFF) are reserved for a customer boot driver to be read in from the NVRAM (see Chapter 12). None of these pages are overwritten, even after the MASSCOMP RTU operating system is booted.

13.2 Console Command Syntax

Console mode uses a limited command interpreter that recognizes commands in the following format:

```
keyword[/qualifier[:count]] [arguments...]
```

The fields are:

keyword the command.

/qualifier[:count]
optional **data**, **address**, or **special use** qualifiers, each delimited by a slash (/). A data qualifier specifies the length of the data argument. An address qualifier indicates which register or address space is referred to by the command's address argument. Special use qualifiers are only used in specific situations. Some qualifiers can have a numerical count, delimited by a colon (:).

arguments optional data values or address expressions, separated by spaces.

Table 13-2 lists all console mode commands and qualifiers.

Table 13-2
Console Summary

COMMANDS	
Command	**Description**
b	Boot
br	Breakpoint
co	Continue
cp	Copy
d	Deposit
du	Dump Memory
e	Examine
i	Initialize SMI & MULTIBUS
m	Enable Main Memory
n	Next (Single Step)
r	Repeat Indefinitely
re	Remote Port Enable
s	Start
t	Test
z	Zero Memory

QUALIFIERS		
Qualifier	**Description**	**Type**
/b	Byte (8 bit data length)	**Data**
/l	Longword (32 bit data length)	
/w	Word (16 bit data length)	
/a	Address Registers	**Address (68020)**
/ca	Cache Address Register	
/cc	Cache Control Register	
/d	Data Registers	
/df	Destination Function Code Reg.	
/i	Instruction (PC) Register	
/m	Status Register	
/s	User Stack Pointer	
/sf	Source Function Code	
/v	Vector Base Register	
/dg	Diagnostic Space	**Address (non-68020)**
/e	Environment Field	
/f	Boot Flags	
/p	Physical Address Space	
/pr	Processor Register Space	
/tb	Translation Buffer Space	
/n:	Next Count	**Special Use**
/nr	Ignore Customer Boot ROM	
/r	Relocation Base Address	
/x	Inhibit Config Defaults	

13.2.1 Arguments

Most console commands require an argument. Command arguments may be data expressions or address expressions.

Data expressions may be byte, word or longword in length, specified by the data qualifiers **/b, /w,** and **/l**. Leading zeros do not need to be typed and are ignored. Data expressions use the following form:

nn	one byte expressed in hexadecimal (example: **FF**)
nnnn	one word expressed in hexadecimal (example: **12FF**)
nnnnnnnn	one longword expressed in hexadecimal (example: **AB3412FF**)

Data arguments that are larger than their qualifier are truncated. So, for example, the command **d/b 6 12345** deposits the value **45** at address **6**.

There is no such thing as a default data value. You *must* include a data argument each time you type a command that requires data.

Address expressions specify registers or various system address spaces. Address expressions use the following syntax:

nnnnnnnn Hexadecimal number representing an address within the address space specified by the address qualifier. Leading zeros are ignored.

***** Reuse the last address value from the most recent command that used an address argument. For example, the command **e *** examines the the most recently specified address argument.

+ Increment the most recently specified address value and use it for the current command. The current data type (not the type declared in the command) determines whether the address is incremented by one byte, one word, or one longword. For example, the command **e/p/b** following the command **Be/p/l** examines the address 4 bytes ahead of the previous one in physical memory.

- Decrement the most recently specified address value and use it for the current command. The current data type (not the type declared in the command) determines whether the address is decremented by one byte, one word, or one longword. For example, the command **e/p/b -** following the command **Be/p/l** examines the address 4 bytes before the previous one in physical memory.

$ Indirect addressing. Use the last data value returned from an **examine** command as the address value for the current command. For example, the command **d/p/l $ 12345678** deposits the value **12345678** into the address returned as data from the previous command.

Commands given without address arguments default to the previous address increment size and direction. For example, the command **e/l -** causes subsequent commands to default to decrement the previous address in longword units. The contents of the Console Relocation Register (Section 13.5.3) is added to all address arguments, default or explicit.

13.2.2 Control Characters

Console mode uses its own set of control characters for flow control and command editing. These control characters are:

CTRL-C interrupt the current command and invoke a new prompt

CTRL-U (also **CTRL-X** or @) discard the current input line, and issue a new prompt

CTRL-R repeat the command line in its current form

CTRL-O alternately suppress and resume the display of output to the console device; do not suspend the current activity

CTRL-S (XOFF) suspend output to the console device; suspend activity until paired **CTRL-Q** (XON) is received from the device

CTRL-Q (XON) resume output to the console device; resume any further activity that outputs to that device

RUBOUT (also **DELETE, BACKSPACE,** or #) delete the last character typed

13.3 The Machine Environment

The NVRAM on the Auxiliary Function Module contains a data field called the **machine environment**. When a system is booted, the console program in EPROM first looks at the environment for all default information required to perform a bootstrap. You may examine and change the environment values stored in the NVRAM from console mode.

To examine the current machine environment in console mode, type:

```
>>> e/e
```

The console terminal displays the current environment in the following form:

```
System serial number is: w-xxx-yyy-zzzz

  Function           Address   Value
---------------      -------   ------
Boot Device            0         ca0
Tty a Baud Rate        1         9600
Auto Boot Enable       2         0
Boot Flags             3         0
Processor Select       4         FF
Buffer Wrt Enable      5         1

   Processor Revisions
ID   ROM    mc68020    mc68881
--   ---    -------    -------
1    x.y      A          B
2    x.y      A          B
 .
 .
 .
```

If the NVRAM is invalid or unreadable, the message `NV Ram Invalid -` *error_code* appears on the top line. The system then uses environment defaults set in the console code. Table 13-3 shows the possible error codes and their descriptions.

Table 13-3
NVRAM Error Codes

Error Code	Description	Field in Error
FE	Error on AFM command	Serial I.D.
FD	Error on AFM address	
FC	Timeout while getting AFM data	
FB	Error on AFM command	Flags
FA	Error on AFM address	
F9	Timeout while getting AFM data	

The system serial number and the console password (if used) are not changeable. The first column in the environment table describes each environment parameter. The second column gives the address in NVRAM that controls that parameter. The third column gives the current value in that address. The parameters are described below.

Boot Device The physical device from which to read in the bootstrap program into memory. The string is limited to 5 characters.

TTY *n* Baud Rate The data rate of the console device. The current console TTY device *n* is displayed and may have values **0, 1, 2,** or **4**. The TTY value can be changed only by pressing the INTERRUPT switch on the front panel and then <CR> on the new TTY device

Auto Boot Enable Determines whether system powers up / resets with the BOOT DEVICE and FLAGS in the environment or goes to console mode for manual boot

Boot Flags Default bootstrap program (UNIX, Standalone Shell, or Diagnostic Monitor)

Processor Select An 8-bit hexadecimal value with each bit corresponding to a processor I.D. that the O.S. is permitted to use. The LSB represents processor I.D. **0**. When a bit is set to a **1**, the corresponding processor is ON. The default is all processors to be ON (NVRAM Address 4 = FF).

Buffered Write Enables or disables the buffered write hardware function on bootstrap (described in Chapter 4).

Processor Revisions Lists the software revision of the EPROM and the hardware revision of the MC68020 microprocessor and MC68881 Floating Point chip on each CMPU module in the system. If there is any variance of these revision numbers between CMPU modules, the system bootstrap fails. If this occurs, contact your MASSCOMP service representative.

Table 13-4 lists the possible values for each field.

Table 13-4
Environment Fields

Environment Parameter	Value	Description
BOOT DEVICE	da0	1st disk on 1st XMD
	da1	2nd disk on 1st XMD
	db0	1st disk on 2nd XMD
	db1	2nd disk on 2nd XMD
	ca0	1st disk on 1st XMC
	ca1	2nd disk on 1st XMC
	cb0	1st disk on 2nd XMC
	cb1	2nd disk on 2nd XMC
	flp	floppy on 1st XMC
BAUD RATE	50*	Console Device Baud Rate
	75**	for TTY 0,1, & 2, except:
	110	* TTY 1 only
	150**	** TTY 0 & 2 only
	134.5	
	200*	
	300	
	600	
	1050*	
	1200	
	1800**	
	2000**	
	2400	
	4800	
	7200*	
	9600	
	19200**	
	38400*	
AUTO BOOT	0	Go to console mode on powerup
	1	Boot using above devices and BOOT FLAGS (below) on powerup
BOOT FLAGS	0	UNIX / Multi-user
	1	Stand Alone Shell (SASH)
	2	UNIX / Single User
	4	Reserved for MASSCOMP
	8	Diagnostic Monitor
PROCESSOR SELECT	0x00 - 0xFF	An 8-bit hexadecimal value with one bit set to a 1 for each processor I.D. that the O.S. is permitted to use
BUFFERED WRITE	0	Disabled
	1	Enabled

To change an environment field value, use the **deposit** command (below).

13.4 Command Descriptions

This section gives a detailed description of each console command.

The command formats below are given with verbatim text in bold and variable input in italics. The examples highlight user input in typewriter bold font. The console prompt >>> is never typed.

13.4.1 Boot

The **boot** command (**b**) has many optional parameters and arguments. In general, the command has the form:

b[**/f:***n*][**/x**] [*device_spec*][*/path_name*] [*switch_string*]

If you execute the boot command without giving any qualifiers, the parameters stored in the NVRAM are used to perform the bootstrap.

If you use the **/f:***n* qualifier, the value *n* overrides the boot flag in the machine environment. See Table 13-4 for a list of valid boot flags in the machine environment.

If you use the **/x** qualifier, the main memory, PCRA and PCRB are left in their current state. The **/x** qualifier also inhibits an I/O reset that may leave non-boot processors in a state that inhibits their use by the operating system. The **x** qualifier allows the system to be booted even if the cache is malfunctioning.

The *device_spec* is an optional argument used to override the boot device default in the environment. See Table 13-4 for valid MASSCOMP boot devices in the environment field.

The *path_name* is an optional UNIX pathname on the boot device that identifies the file to be used to boot the system. This pathname may be used to specify a diagnostic program, a non-standard version of UNIX, or some other program to be debugged. If *path_name* is not given, the default file used is */unix* (or *diagm..1.5600*, depending on the boot flags or whether a *switch_string* is specified).

The *switch_string* is an optional string used to override the default boot flag set in the environment on the NVRAM. Possible switch string values are given in Table 13-5.

Table 13-5
Boot Switch String Values

Switch	Description
-stand	Standalone Shell
-single	Single user
-multi	Multiuser
-btcpu	Boot CMPU
-nbtcpu	Non-boot CMPUs
-all	All CMPUs
-debug	Kernel Debugger
-swap	Change swap file
-printer	Enable printer for console

Some examples of the **boot** command are shown in Table 13-6.

Table 13-6
Boot Command Examples

Boot Command	Description
b	Boot using default parameters in the NVRAM
b da0/unix -single	Boot to single user UNIX from da0
b da0/unix	Boot to multiuser UNIX from da0
b/f:8 flp	Boot the diagnostic monitor from floppy
b da0/stand/sash	Boot *sash* (standalone shell) from da0
b/f:1 flp	Boot *sash* from floppy

13.4.2 Breakpoint

The **breakpoint** command (**br**) sets a breakpoint at a specified address in the program. When the program in memory is run, execution is suspended as soon as the specified breakpoint address is reached. The command has the form:

br [*/qualifiers*] *address*

Typing **br** with no arguments displays all currently set breakpoints. A dash before the address clears a breakpoint at that address. Some examples are:

```
>>> br/1 08001234     # sets breakpoint at 08001234
>>> br -08001234      # clears breakpoint at 08001234
```

```
>>> br              # displays breakpoints
>>> br -            # clears all breakpoints
```

13.4.3 Continue

The **continue** command (**co**) resumes program execution from the current program counter. The command takes no arguments.

13.4.4 Copy

The **copy** command (**cp**) copies the contents of one address or group of addresses to another. The command has the form:

```
cp[/qualifiers] source_addr destination_addr
```

For example, to copy 100 longwords at address 1000 to address 2000, the command is:

```
>>> cp/n:100/l 08001000 08002000
```

13.4.5 Deposit

The **deposit** command (**d**) is used both for general purpose diagnostics and for changing default values in the NVRAM. The **d** command writes its data argument to a specified address. The command has the form:

```
d[/qualifiers] address data
```

For example, to deposit the data byte **0xFF** to the first XMD controller on the MULTIBUS 1, the command is:

```
>>> d/p/b F2EE40 FF
```

The deposit command is also used to change NVRAM values in the environment field. For example, to change the boot flag in the environment to a **2** to change the system default powerup to be single-user mode, the command is:

```
>>> d/e 3 2
```

13.4.6 Dump

The **dump** command (**du**) writes the entire contents of memory onto the specified device. If the device is a file system, it uses the default file */stand/crashdump*, which must already exist. The command requires a device spec as an address argument, as shown in Table 13-7.

Table 13-7
Dump Devices

Dump Device	Controller
ca0, ca1, cb0, cb1	XMC
cc0, cc1, cd0, cd1	
da0, da1, da2, da3	XMD
db0, db1, db2, db3	
dc0, dc1, dc2, dc3	
xmt0boot	XMT

Some examples are:

```
>>> du otp      # Dump onto cartridge tape
>>> du mtpboot  # Dump onto 1/2" tape
>>> du ca1      # Dump onto 2nd disk on first XMC
```

The **du** command automatically enables main memory.

13.4.7 Examine

The **examine** command (**e**) is used to display the contents of a specified register or address. The command has the form:

```
e[/qualifiers] address
```

Some examples are:

```
>>> e/p/w 8000000          # examine first word in SMI memory
>>> e/l/n:100 8000000      # examine 100 longwords starting at 08000000
```

13.4.8 Initialize

The **initialize** command (**i**) executes a 68020 RESET instruction. If the CMPU jumpers are configured with the 68020 RESET line tied to the MULTIBUS INIT line (as is normally the case for processor I.D. 1), executing the **i** command on the boot processor causes the MULTIBUS INIT line to be asserted on all MULTIBUSs. Non-boot processors enter the stopped state if memory has been enabled. Otherwise, the non-boot processors wait for a handshake from the boot processor.

13.4.9 Memory Enable

Memory is normally disabled whenever the hardware reset switch is pressed. The **memory-enable** command (**m**) enables main memory and leaves its contents intact for program debugging. It takes no arguments.

13.4.10 Next (Single Step)

The **next** command (**n**) is used to single step through a program. The command executes the next instruction pointed to by the 68020 program counter. The command may be given a numerical count. For example, to execute the next 5 instructions, the command is:

```
>>> n/n:5
```

After each instruction, the Console Relocation Register (described in Section 13.5.3) is subtracted from the current contents of the program counter, and this value is printed.

13.4.11 Repeat

The **repeat** command (**r**) repeats the accompanying command until interrupted by pressing CTRL-C. It has the form:

```
r console_command
```

The **r** command can only be used with the **d, e,** and **cp** commands. For example, to repeatedly examine address **FF**, the command is:

```
>>> r e/b FF
```

The console device repeatedly displays the contents of address **0xFF**.

13.4.12 Remote Port Enable

The **remote port enable** command (**re**) allows a terminal connected to the system by a modem to be the console device. The **re** command ensures that a remote terminal can become console *only* by being first enabled at the local terminal.

Typing the **re** command at the current console terminal first disconnects that terminal device. If **re** is issued through a modem line, the line is disconnected until the modem hangs up and reconnects.

After disconnecting the current console device, the system scans all of its ports for a deasserted DCD (Data Carrier Detect) line to become asserted. Any port at which this occurs is included in the autobaud algorithm described in Section 13.1.2.

Pressing the INTERRUPT switch on the front panel causes the remote enable state to be cleared but does not disconnect a modem that is already connected. Pressing the RESET switch disconnects any modems that are connected.

13.4.13 Start

The **start** command (**s**) initializes the program counter, executes the equivalent of the **initialize** console command, and begins program execution. The **s** command has the form:

s address

where *address* is the address in physical memory of the first instruction to be executed.

13.4.14 Selftest

The **selftest** command (**t**) executes the CPU hardware selftest program in the EPROM that is normally run during powerup. The test displays a character on all three UART ports for each test begun. See Chapter 12 for more information on the CPU selftest.

13.4.15 Zero

The **zero** command **z** enables and clears main memory, thus initializing the memory error checking mechanism. The command takes no arguments. To enable memory without clearing, use the **m** command.

13.5 Qualifiers

This section describes all of the console mode qualifiers. There are three types of qualifiers, as explained below: **address, data**, and **special use qualifiers**.

13.5.1 Address Qualifiers

Address qualifiers specify the the register or memory space referred to by the accompanying command's address argument. There are two groups of address qualifiers: 68020 registers and non-68020 registers. The following qualifiers refer to internal registers on the 68020:

/a	Address registers
/d	Data registers
/df	Destination Function register
/sf	Source Function register
/ca	Cache Address register
/cc	Cache Control register
/i	Instruction (program counter) register

/m Status register

/s User Stack Pointer register

/v Vector Base register

All 68020 registers (except the status register) are 32-bit (longword) registers. The data in these registers are actually stored in main memory locations by the console code, so some registers may not show as expected until the **co** command is given. For more information on these registers, consult the *MC68020 32-Bit Microprocessor User's Manual.*

The following are the non-68020 console address qualifiers:

/p Physical address space (the powerup default if no qualifier is specified)

/tb Translation buffer space (see Chapter 2)

/dg Diagnostic space (see Chapter 2)

/pr Processor Register space. These are the control registers TBCCR and TBCFR (see Chapter 2) on the CMPU board running the Console code

/e Environment Fields. This data is located in the NVRAM on the AFM and determines the type of bootstrap operation.

/f Boot Flags. See Table 13-4 for boot flag values.

Once specified, an address type qualifier overwrites the console-maintained defaults for all subsequent commands. The console applies the current default when an address argument is not given an address type qualifier. So, for example, you do not have to type **/d** for each consecutive command that refers to a 68020 data register.

Section 13.6 describes how to run the console in a multiprocessor system to examine registers on other CMPU modules.

13.5.2 Data Type Qualifiers

Data qualifiers are used with the **examine, deposit,** and **copy** commands to change the default size of a command's data argument. Data may be a byte, a word or a longword in length:

/b byte (8 bits)

/w word (16 bits)

/l longword (32 bits). This is the powerup default

Once specified, a data type qualifier overwrites the console-maintained defaults for all subsequent commands. The console applies the current default when a data argument is not given a data type qualifier. So, for example, you do not have to type /l for each consecutive command that uses a longword data-type.

13.5.3 Special Use Qualifiers

There are four qualifiers that are associated only with specific commands or situations: the **/n:**, **/nr**, **/r**, and **/x** qualifiers.

The **next** qualifier (**/n:***x*) performs the accompanying command beginning at the specified address for *x* iterations. The count *x* is interpreted as a hex value. This qualifier is used only with the **cp, d, e,** and **n** commands.

Some examples are:

```
>>> n/n:9       # execute next 9 instructions
>>> n/n:C       # execute next 12 instructions
```

The **ignore customer boot ROM** qualifier (**/nr**) is used with the **boot** command. If you have loaded your boot code into the NVRAM as described in Chapter 12, the bootstrap defaults to the code in this memory space. The **/nr** qualifier instructs the console to boot using the standard MASSCOMP bootstrap. This qualifier is useful if a device driver in the NVRAM has the same name as a MASSCOMP driver and you wish to execute the MASSCOMP driver.

The **relocation** qualifier (**/r**) stores its data argument in the Console Relocation Register. The value in this register is added as an offset to all address qualifiers given to the **br, cp, d, e,** and **s** commands. A data argument is required with this qualifier. Initialization sets the value of the Relocation Register to zero.

For example, the following command sets the Relocation Register to be 5 and all subsequent addresses to be automatically incremented by 5:

```
>>> d/r 5
```

The **exclude** qualifier (**/x**) excludes PCRA, PCRB, main memory initialization, and I/O reset when executing a **b** or **s** command. Normally, these commands initialize cache and main memory. The **x** qualifier causes the PCRA and PCRB registers and memory to be left in their current state.

13.6 Running Console Mode on Non-boot Processors

If you want to examine or deposit a non-boot CMPU local device, register, or secondary address space in console mode, you must execute the code on that processor. To run console mode on a processor other than Processor I.D. 1 in a multiprocessor system:

1. Enter console mode, as described in the beginning of this chapter.
2. Deposit a value of **0x81** to the IPIR of the CMPU module you wish to run using the console **d** command to set a level 7 interrupt. The EPROM sets the level 7 IPIR interrupt on all non-boot processors to vector to the local EPROM's console code. Chapter 5 explains how to set the IPIR and Chapter 8 describes the physical addresses of the IPIRs in a multiprocessor system.
3. The TTY0 terminal on the selected processor displays console mode.

Appendix A
Pinouts

Appendix B
MC5600/5700 Specifications

Appendix C
DUART Specification

TABLES

Appendix A
Pinouts

This appendix contains the following pinout tables:

- **Table A-1:** The pinout for CMPU backplane connector J36 that connects to the front panel.
- **Table A-2:** The pins on the MULTIBUS common to all 15 backplane connectors. The MULTIBUS pins used by the CMPU and CMM modules are also indicated.
- **Table A-3:** The pins on the MULTIBUS that vary depending on the connector and module.
- **Table A-4:** The pins on the SMI common to all 15 backplane connectors. The SMI pins used by the CMPU and CMM modules are also indicated.
- **Table A-5:** The pins on the SMI that vary depending on the connector and module.
- **Tables A-6 and A-7:** The CMPU Serial Ports pinouts (2 connectors).
- **Tables A-8 through A-11:** The pinouts for the 4 connectors on the Auxiliary Function / Arbitration Module (AFM/ARB).

The pinout tables in this appendix use the following symbols:

Symbol	Description
•	Used by module
-	Not used by module
*	Signal varies per connector; See table of signal exceptions
—	Voltage supply

Table A-1
CMPU To Front Panel Connector (J36) Pinout

Backplane Connector J36			
PIN	SIGNAL NAME	DESCRIPTION	SIGNAL TYPE
P1	GND	Signal Ground	—
P2	GND	Signal Ground	—
P3	+5v	DC +5v Power Supply	—
P4	MBUS SW Reset L	Software Reset	TTL
P5	Power on L	Power Controller	Output
P6	MBUS AC LO L	MBUS AC Low	OC
P7	MBUS Error L	CMPU Error Condition(s)	OC
P8	NC	Not Connected	—
P9	Power on L Return	Power Controller Return	Input
P10	NC	Not Connected	—

Table A-2
MULTIBUS Backplane & Module Pinouts (86 Pins)

Backplane Connectors J01 thru J15				CMPU	CMM
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE		
P1	GND	Ground	—	●	●
P2	GND	Ground	—	●	●
P3	VCC	Power +5v	—	●	●
P4	VCC	Power +5v	—	●	●
P5	VCC	Power +5v	—	●	●
P6	VCC	Power +5v	—	●	●
P7	VDD	Power +12v	—	●	-
P8	VDD	Power +12v	—	●	-
P9	VEE	Power -5.2v	—	-	-
P10	VEE	Power -5.2v	—	-	-
P11	GND	Ground	—	●	●
P12	GND	Ground	—	●	●
P13	MBUS BCLK L	MBus B Clock	TTL	-	-
P14	MBUS INIT L	INIT Signal	OC	●	●
P15	MBUS BPRN L	MBUS Grant	TTL	●	-
P16	MBUS SPARE 1 L	—	—	-	-
P17	MBUS BUSY L	Bus Busy	OC	●	-
P18	MBA Disable	Disable (MBUS BREQ L)	TTL	●	-
P19	MBUS MRDC L	Memory Read Command	TS	●	-
P20	MBUS MWTC L	Memory Write Command	TS	●	-
P21	MBUS IORC L	I/O Read Command	TS	●	-

Backplane Connectors J01 thru J15				CMPU	CMM
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE		
P22	MBUS IOWC L	I/O Write Command	TS	●	-
P23	MBUS XACK L	Transfer Acknowledge	TS	●	-
P24	MBUS SW Reset L	Software Reset	TTL	●	-
P25	MBUS Lock L	Lock	TS	●	-
P26	MBUS Error L	Error	OC	●	-
P27	MBUS BHEN L	Byte High Enable	TS	●	-
P28	MBUS ADRS <16> L	MBUS Address Line	TS	●	-
P29	SIG GND L	Ground	—	●	-
P30	MBUS ADRS <17> L	MBUS Address Line	TS	●	-
P31	MBUS CCLK L	MBUS C Clock	TTL	-	-
P32	MBUS ADRS <18> L	MBUS Address Line	TS	●	-
P33	MBUS SPARE 2 L	—	—	-	-
P34	MBUS ADRS <19> L	MBUS Address Line	TS	●	-
P35	MBUS INT <6> L	MBUS Interrupt	OC	●	●
P36	MBUS MC Lock2 L	MBUS MASSCOMP Lock	TS	●	-
P37	MBUS INT <4> L	MBUS Interrupt	OC	●	-
P38	MBUS INT <5> L	MBUS Interrupt	OC	●	-
P39	MBUS INT <2> L	MBUS Interrupt	OC	●	-
P40	MBUS INT <3> L	MBUS Interrupt	OC	●	-
P41	MBUS Block Mode L	MBUS Block Mode	TS	●	-
P42	MBUS INT <1> L	MBUS Interrupt	OC	●	-
P43	MBUS ADRS <14> L	MBUS Address Line	TS	●	-
P44	MBUS ADRS <15> L	MBUS Address Line	TS	●	-
P45	MBUS ADRS <12> L	MBUS Address Line	TS	●	-
P46	MBUS ADRS <13> L	MBUS Address Line	TS	●	-
P47	MBUS ADRS <10> L	MBUS Address Line	TS	●	-
P48	MBUS ADRS <11> L	MBUS Address Line	TS	●	-
P49	MBUS ADRS <08> L	MBUS Address Line	TS	●	-
P50	MBUS ADRS <09> L	MBUS Address Line	TS	●	-
P51	MBUS ADRS <06> L	MBUS Address Line	TS	●	-
P52	MBUS ADRS <07> L	MBUS Address Line	TS	●	-
P53	MBUS ADRS <04> L	MBUS Address Line	TS	●	-
P54	MBUS ADRS <05> L	MBUS Address Line	TS	●	-
P55	MBUS ADRS <02> L	MBUS Address Line	TS	●	-
P56	MBUS ADRS <03> L	MBUS Address Line	TS	●	-
P57	MBUS ADRS <00> L	MBUS Address Line	TS	●	-
P58	MBUS ADRS <01> L	MBUS Address Line	TS	●	-
P59	MBUS DATA <14> L	MBUS Data Line	TS	●	-
P60	MBUS DATA <15> L	MBUS Data Line	TS	●	-
P61	MBUS DATA <12> L	MBUS Data Line	TS	●	-
P62	MBUS DATA <13> L	MBUS Data Line	TS	●	-

Backplane Connectors J01 thru J15				CMPU	CMM
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE		
P63	MBUS DATA <10> L	MBUS Data Line	TS	●	-
P64	MBUS DATA <11> L	MBUS Data Line	TS	●	-
P65	MBUS DATA <08> L	MBUS Data Line	TS	●	-
P66	MBUS DATA <09> L	MBUS Data Line	TS	●	-
P67	MBUS DATA <06> L	MBUS Data Line	TS	●	-
P68	MBUS DATA <07> L	MBUS Data Line	TS	●	-
P69	MBUS DATA <04> L	MBUS Data Line	TS	●	-
P70	MBUS DATA <05> L	MBUS Data Line	TS	●	-
P71	MBUS DATA <02> L	MBUS Data Line	TS	●	-
P72	MBUS DATA <03> L	MBUS Data Line	TS	●	-
P73	MBUS DATA <00> L	MBUS Data Line	TS	●	-
P74	MBUS DATA <01> L	MBUS Data Line	TS	●	-
P75	GND	Ground	—	●	●
P76	GND	Ground	—	●	●
P77	MBUS SPARE 3 L	—	—	-	-
P78	MBUS AC LO L	MBUS AC Low	OC	●	-
P79	VNN	Power -12v	—	●	-
P80	VNN	Power -12v	—	●	-
P81	VCC	Power +5v	—	●	●
P82	VCC	Power +5v	—	●	●
P83	VCC	Power +5v	—	●	●
P84	VCC	Power +5v	—	●	●
P85	GND	Ground	—	●	●
P86	GND	Ground	—	●	●

Table A-3
MULTIBUS Pinout Exceptions

CONNECTOR	PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE
J01	P15	MBUS BPRN <01> L	Bus Grant	TTL
	P18	MBUS BREQ <01> L	Bus Request	TTL
	P29	MBUS MC CBRQ H	MASSCOMP Com Bus Req	TTL
J02 thru J15	P15	MBUS BPRN <2> thru <15> L	Bus Grant	TTL
	P18	MBUS BREQ <2> thru <15> L	Bus Request	TTL
	P29	MBUS CBRQ H	Com Bus Req	OC
CMPU	P15	MBUS BPRN L	Bus Grant	TTL
	P18	MBA Disable H	Disable MBA (MBUS BREQ L)	TTL
	P29	SIG GND	Ground	—

Jumper headers J38, J39, and J48 on the backplane allow the MULTIBUS to be split into two separate buses, MULTIBUS 1 and MULTIBUS 2. Some signals, however, are not jumpered and span the entire backplane whether or not there are two separate MULTIBUSs. These signals are:

- BCLK (pin 13)
- SW RESET (pin 24)
- ERROR (pin 26)
- CCLK (pin 31)
- SPARE 2 (pin 33)
- SPARE 3 (pin 77)
- AC LO (pin 78)

Table A-4
SMI Backplane & Module Pinouts (60 Pins)

Backplane Connectors J16 thru J31, J34				CMPU	CMM
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE		
P1	GND	Ground	—	●	●
P2	GND	Ground	—	●	●
P3	*	* See Table A-5	*	*	*
P4	*	* See Table A-5	*	*	*
P5	SMI SCLK L	SMI S Clock	TTL	●	●
P6	*	* See Table A-5	*	*	*
P7	SMI ECLK L	SMI E Clock	TTL	●	●
P8	SMI CMD <2> L	SMI Command	TS	●	●

Backplane Connectors J16 thru J31, J34				CMPU	CMM
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE		
P9	SMI CMD <1> L	SMI Command	TS	●	●
P10	SMI CMD <0> L	SMI Command	TS	●	●
P11	SMI NID <4> L	Node ID	TS	●	●
P12	SMI NID <3> L	Node ID	TS	●	●
P13	SMI NID <2> L	Node ID	TS	●	●
P14	SMI NID <1> L	Node ID	TS	●	●
P15	SMI NID <0> L	Node ID	TS	●	●
P16	SMI MEM INH L	Memory Inhibit	OC	●	●
P17	SMI ACK <1> L	Acknowledge	OC	●	●
P18	SMI ACK <0> L	Acknowledge	OC	●	●
P19	SMI LCK INH L	Lock Inhibit	OC	●	-
P20	SMI INV INH L	Invalidate Inhibit	OC	●	-
P21	GND	Ground	—	●	●
P22	GND	Ground	—	●	●
P23	SMI BUSY L	Bus Busy	OC	●	●
P24	*	* See Table A-5	*	*	*
P25	SMI DAL <31> L	SMI Data & Address	TS	●	●
P26	SMI DAL <30> L	SMI Data & Address	TS	●	●
P27	SMI DAL <29> L	SMI Data & Address	TS	●	●
P28	SMI DAL <28> L	SMI Data & Address	TS	●	●
P29	SMI DAL <27> L	SMI Data & Address	TS	●	●
P30	SMI DAL <26> L	SMI Data & Address	TS	●	●
P31	SMI DAL <25> L	SMI Data & Address	TS	●	●
P32	SMI DAL <24> L	SMI Data & Address	TS	●	●
P33	SMI DAL <23> L	SMI Data & Address	TS	●	●
P34	SMI DAL <22> L	SMI Data & Address	TS	●	●
P35	SMI DAL <21> L	SMI Data & Address	TS	●	●
P36	SMI DAL <20> L	SMI Data & Address	TS	●	●
P37	SMI DAL <19> L	SMI Data & Address	TS	●	●
P38	SMI DAL <18> L	SMI Data & Address	TS	●	●
P39	SMI DAL <17> L	SMI Data & Address	TS	●	●
P40	SMI DAL <16> L	SMI Data & Address	TS	●	●
P41	SMI DAL <15> L	SMI Data & Address	TS	●	●
P42	SMI DAL <14> L	SMI Data & Address	TS	●	●
P43	SMI DAL <13> L	SMI Data & Address	TS	●	●

Backplane Connectors J16 thru J31, J34				CMPU	CMM
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE		
P44	SMI DAL <12> L	SMI Data & Address	TS	●	●
P45	SMI DAL <11> L	SMI Data & Address	TS	●	●
P46	SMI DAL <10> L	SMI Data & Address	TS	●	●
P47	SMI DAL <09> L	SMI Data & Address	TS	●	●
P48	SMI DAL <08> L	SMI Data & Address	TS	●	●
P49	SMI DAL <07> L	SMI Data & Address	TS	●	●
P50	SMI DAL <06> L	SMI Data & Address	TS	●	●
P51	SMI DAL <05> L	SMI Data & Address	TS	●	●
P52	SMI DAL <04> L	SMI Data & Address	TS	●	●
P53	SMI DAL <03> L	SMI Data & Address	TS	●	●
P54	SMI DAL <02> L	SMI Data & Address	TS	●	●
P55	*	* See Table A-5	*	*	-
P56	*	* See Table A-5	*	*	-
P57	*	* See Table A-5	*	*	-
P58	*	* See Table A-5	*	*	-
P59	SMI DAL <01> L	SMI Data & Address	TS	●	●
P60	SMI DAL <00> L	SMI Data & Address	TS	●	●

Table A-5
SMI Pinout Exceptions

CONNECTOR	PIN	SIGNAL NAME	DESCRIPTION	SIGNAL TYPE
J16 thru J22	P3	VBB	+5v Battery	—
	P4	VBB	+5v Battery	—
	P6	SMI REQ <1> thru <7> L	SMI Bus Request	TTL
	P24	SMI GNT <1> thru <7> L	SMI Bus Grant	TTL
	P55	MBUS 1 ADRS <22> L	MBUS Address	TS
	P56	MBUS 1 ADRS <23> L	MBUS Address	TS
	P57	MBUS 1 ADRS <20> L	MBUS Address	TS
	P58	MBUS 1 ADRS <21> L	MBUS Address	TS
J23 thru J30	P3	VBB	+5v Battery	—
	P4	VBB	+5v Battery	—
	P6	SMI REQ <8> thru <15> L	SMI Bus Request	TTL
	P24	SMI GNT <8> thru <15> L	SMI Bus Grant	TTL
	P55	MBUS 2 ADRS <22> L	MBUS Address	TS
	P56	MBUS 2 ADRS <23> L	MBUS Address	TS
	P57	MBUS 2 ADRS <20> L	MBUS Address	TS
	P58	MBUS 2 ADRS <21> L	MBUS Address	TS
J31	P3	VBB	+5v Battery	—
	P4	VBB	+5v Battery	—
	P6	SMI REQ <19> L	SMI Bus Request	TTL
	P24	SMI GNT <19> L	SMI Bus Grant	TTL
	P55	SMI REQ <22> L	SMI Bus Request	TTL
	P56	SMI GNT <22> L	SMI Bus Grant	TTL
	P57	SMI REQ <26> L	SMI Bus Request	TTL
	P58	SMI GNT <26> L	SMI Bus Grant	TTL
J34	P3	NC	Not Connected	—
	P4	NC	Not Connected	—
	P6	SMI REQ <04> L	SMI Bus Request	TTL
	P24	SMI GNT <04> L	SMI Bus Grant	TTL
	P55	SMI REQ <07> L	SMI Bus Request	TTL
	P56	SMI GNT <07> L	SMI Bus Grant	TTL
	P57	SMI REQ <11> L	SMI Bus Request	TTL
	P58	SMI GNT <11> L	SMI Bus Grant	TTL

MODULE	PIN	SIGNAL NAME	DESCRIPTION	SIGNAL TYPE
CMPU	P6	SMI REQ L	Bus Request	TTL
	P24	SMI GNT L	Bus Grant	TTL
	P55	MBUS ADRS <22> L	MBUS Address Line	TS
	P56	MBUS ADRS <23> L	MBUS Address Line	TS
	P57	MBUS ADRS <20> L	MBUS Address Line	TS
	P58	MBUS ADRS <21> L	MBUS Address Line	TS
CMM	P3	VBB	+5v Battery	—
	P4	VBB	+5v Battery	—
	P6	SMI REQ L	Bus Request	TTL
	P24	SMI GNT L	Bus Grant	TTL

Table A-6
CMPU Module, Connector P03 Pinout (Ports P0A, P0B, P1A)

CMPU P03 Connector			
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE
P1	P0A TXD L	Transmit Data	RS-232
P2	P0A RXD L	Receive Data	RS-232
P3	P0A RTS H	Request to Send	RS-232
P4	P0A CTS H	Clear to Send	RS-232
P5	P0A DSR H	Data Set Ready	RS-232
P6	GND	Ground	—
P7	P0A DCD H	Data Carrier Detected	RS-232
P8	P0A DTR H	Data Terminal Ready	RS-232
P9	P0B TXD L	Transmit Data	RS-232
P10	P0B RXD L	Receive Data	RS-232
P11	P0B RTS H	Request To Send	RS-232
P12	P0B CTS H	Clear to Send	RS-232
P13	P0B DSR H	Data Set Ready	RS-232
P14	GND	Ground	—
P15	P0B DCD H	Data Carrier Detected	RS-232
P16	P0B DTR H	Data Terminal Ready	RS-232
P17	P1A TXD L	Transmit Data	RS-232
P18	P1A RXD L	Receive Data	RS-232
P19	P1A RTS H	Request To Send	RS-232
P20	P1A CTS H	Clear to Send	RS-232
P21	P1A DSR H	Data Set Ready	RS-232
P22	GND	Ground	—
P23	P1A DCD H	Data Carrier Detected	RS-232
P24	P1A DTR H	Data Terminal Ready	RS-232
P25	VCC	Power +5v	—
P26	BUF CON ERR L	Error Condition	TTL

Table A-7
CMPU Module, Connector P04 Pinout (Port P1B)

CMPU P04 Connector			
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE
P1	SIG GND L	Signal Ground	—
P2	P1B TXD H	Transmit Data	TTL
P3	P1B RXD H	Receive Data	TTL
P4	P1B RTS L	Request to Send	TTL
P5	P1B CTS L	Clear to Send	TTL
P6	P1B DSR L	Data Set Ready	TTL
P7	SIG GND L	Signal Ground	—
P8	P1B DCD L	Data Carrier Detected	TTL
P9	P1B DTR L	Data Terminal Ready	TTL
P10	GND	Ground	—
P11	AFM Attached L	AFM Attached	TTL
P12	AFM INIT ALL L	INIT	TTL
P13	BUF Reset L	Buffer Reset	TTL
P14	VCC	Power +5v	—

Table A-8
AFM/ARB Module, Connector P01 Pinout (70 Pins)

AFM/ARB P01 Connector			
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE
P1	MBUS BREQ 01 L	MBUS Bus Request	TTL
P2	MBUS BPRN 01 L	MBUS Bus Grant	TTL
P3	MBUS BREQ 02 L	MBUS Bus Request	TTL
P4	MBUS BPRN 02 L	MBUS Bus Grant	TTL
P5	MBUS BREQ 03 L	MBUS Bus Request	TTL
P6	MBUS BPRN 03 L	MBUS Bus Grant	TTL
P7	MBUS BREQ 04 L	MBUS Bus Request	TTL
P8	MBUS BPRN 04 L	MBUS Bus Grant	TTL
P9	MBUS BREQ 05 L	MBUS Bus Request	TTL
P10	MBUS BPRN 05 L	MBUS Bus Grant	TTL
P11	MBUS BREQ 06 L	MBUS Bus Request	TTL
P12	MBUS BPRN 06 L	MBUS Bus Grant	TTL
P13	MBUS BREQ 07 L	MBUS Bus Request	TTL
P14	MBUS BPRN 07 L	MBUS Bus Grant	TTL
P15	MBUS BREQ 08 L	MBUS Bus Request	TTL
P16	MBUS BPRN 08 L	MBUS Bus Grant	TTL

AFM/ARB P01 Connector			
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE
P17	MBUS BREQ 09 L	MBUS Bus Request	TTL
P18	MBUS BPRN 09 L	MBUS Bus Grant	TTL
P19	MBUS BREQ 10 L	MBUS Bus Request	TTL
P20	MBUS BPRN 10 L	MBUS Bus Grant	TTL
P21	MBUS BREQ 11 L	MBUS Bus Request	TTL
P22	MBUS BPRN 11 L	MBUS Bus Grant	TTL
P23	MBUS BREQ 12 L	MBUS Bus Request	TTL
P24	MBUS BPRN 12 L	MBUS Bus Grant	TTL
P25	MBUS BREQ 13 L	MBUS Bus Request	TTL
P26	MBUS BPRN 13 L	MBUS Bus Grant	TTL
P27	MBUS BREQ 14 L	MBUS Bus Request	TTL
P28	MBUS BPRN 14 L	MBUS Bus Grant	TTL
P29	MBUS BREQ 15 L	MBUS Bus Request	TTL
P30	MBUS BPRN 15 L	MBUS Bus Grant	TTL
P31	VEE	(AFM2/ARB2 only)	—
P32	VDD	(AFM2/ARB2 only)	—
P33	MBUS MC CBRQ H	MASSCOMP Com MBUS Bus Req	TTL
P34	VCC	Power +5v	—
P35	VCC	Power +5v	—
P36	GND	Ground	—
P37	GND	Ground	—
P38	GND	Ground	—
P39	SMI GNT 15 L	SMI Bus Grant	TTL
P40	SMI REQ 15 L	SMI Bus Request	TTL
P41	SMI GNT 14 L	SMI Bus Grant	TTL
P42	SMI REQ 14 L	SMI Bus Request	TTL
P43	SMI GNT 13 L	SMI Bus Grant	TTL
P44	SMI REQ 13 L	SMI Bus Request	TTL
P45	SMI GNT 12 L	SMI Bus Grant	TTL
P46	SMI REQ 12 L	SMI Bus Request	TTL
P47	SMI GNT 11 L	SMI Bus Grant	TTL
P48	SMI REQ 11 L	SMI Bus Request	TTL
P49	SMI GNT 10 L	SMI Bus Grant	TTL
P50	SMI REQ 10 L	SMI Bus Request	TTL
P51	SMI GNT 09 L	SMI Bus Grant	TTL
P52	SMI REQ 09 L	SMI Bus Request	TTL
P53	SMI GNT 08 L	SMI Bus Grant	TTL
P54	SMI REQ 08 L	SMI Bus Request	TTL
P55	SMI GNT 07 L	SMI Bus Grant	TTL
P56	SMI REQ 07 L	SMI Bus Request	TTL
P57	SMI GNT 06 L	SMI Bus Grant	TTL

AFM/ARB P01 Connector			
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE
P58	SMI REQ 06 L	SMI Bus Request	TTL
P59	SMI GNT 05 L	SMI Bus Grant	TTL
P60	SMI REQ 05 L	SMI Bus Request	TTL
P61	SMI GNT 04 L	SMI Bus Grant	TTL
P62	SMI REQ 04 L	SMI Bus Request	TTL
P63	SMI GNT 03 L	SMI Bus Grant	TTL
P64	SMI REQ 03 L	SMI Bus Request	TTL
P65	SMI GNT 02 L	SMI Bus Grant	TTL
P66	SMI REQ 02 L	SMI Bus Request	TTL
P67	SMI GNT 01 L	SMI Bus Grant	TTL
P68	SMI REQ 01 L	SMI Bus Request	TTL
P69	NC	Not Connected	—
P70	NC	Not Connected	—

Table A-9
AFM/ARB Module, Connector P02 Pinout (40 Pins)

AFM/ARB P02 Connector			
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE
P1	MBUS BCLK L	MBUS B Clock	TTL
P2	MBUS CCLK L	MBUS C Clock	TTL
P3	SMI GNT 30 L	SMI Bus Grant	TTL
P4	SMI REQ 30 L	SMI Bus Request	TTL
P5	SMI GNT 29 L	SMI Bus Grant	TTL
P6	SMI REQ 29 L	SMI Bus Request	TTL
P7	SMI GNT 28 L	SMI Bus Grant	TTL
P8	SMI REQ 28 L	SMI Bus Request	TTL
P9	SMI GNT 27 L	SMI Bus Grant	TTL
P10	SMI REQ 27 L	SMI Bus Request	TTL
P11	NC	Not Connected	—
P12	NC	Not Connected	—
P13	SMI GNT 25 L	SMI Bus Grant	TTL
P14	SMI REQ 25 L	SMI Bus Request	TTL
P15	SMI GNT 24 L	SMI Bus Grant	TTL
P16	SMI REQ 24 L	SMI Bus Request	TTL
P17	SMI GNT 23 L	SMI Bus Grant	TTL
P18	SMI REQ 23 L	SMI Bus Request	TTL
P19	NC	Not Connected	—
P20	NC	Not Connected	—

AFM/ARB P02 Connector			
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE
P21	SMI GNT 21 L	SMI Bus Grant	TTL
P22	SMI REQ 21 L	SMI Bus Request	TTL
P23	SMI GNT 20 L	SMI Bus Grant	TTL
P24	SMI REQ 20 L	SMI Bus Request	TTL
P25	NC	Not Connected	—
P26	VNN	(AFM2/ARB2 only)	—
P27	SMI GNT 18 L	SMI Bus Grant	TTL
P28	SMI REQ 18 L	SMI Bus Request	TTL
P29	SMI GNT 17 L	SMI Bus Grant	TTL
P30	SMI REQ 17 L	SMI Bus Request	TTL
P31	SMI GNT 16 L	SMI Bus Grant	TTL
P32	SMI REQ 16 L	SMI Bus Request	TTL
P33	MBUS ERROR L	Error	OC
P34	MBUS SW RESET L	Software Reset	TTL
P35	MBUS AC LO L	MBUS AC Low	OC
P36	NC	Not Connected	—
P37	15 Slot MBUS L	1=7/8slot;0=15 slot	TTL
P38	MBUS INIT L	INIT	OC
P39	VCC	Power +5v	—
P40	VCC	Power +5v	—

Table A-10
AFM/ARB Module, Connector P03 Pinout (60 Pins)

AFM/ARB P03 Connector			
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE
P1	GND	Ground	—
P2	GND	Ground	—
P3	VBB	+5v	—
P4	VBB	+5v	—
P5	SMI SCLK L	SMI S Clock	TTL
P6	SMI REQ <19> L	SMI Bus Request	TTL
P7	SMI ECLK L	SMI E Clock	TTL
P8	NC	Not Connected	—
P9	NC	Not Connected	—
P10	NC	Not Connected	—
P11	SMI NID <4> L	Node ID	TS
P12	SMI NID <3> L	Node ID	TS
P13	SMI NID <2> L	Node ID	TS

AFM/ARB P03 Connector			
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE
P14	SMI NID <1> L	Node ID	TS
P15	SMI NID <0> L	Node ID	TS
P16	NC	Not Connected	—
P17	NC	Not Connected	—
P18	NC	Not Connected	—
P19	NC	Not Connected	—
P20	NC	Not Connected	—
P21	NC	Not Connected	—
P22	NC	Not Connected	—
P23	NC	Not Connected	—
P24	SMI GNT <19> L	SMI Bus Grant	TTL
P25	SMI DAL <31> L	SMI Data & Address	TS
P26	SMI DAL <30> L	SMI Data & Address	TS
P27	SMI DAL <29> L	SMI Data & Address	TS
P28	SMI DAL <28> L	SMI Data & Address	TS
P29	SMI DAL <27> L	SMI Data & Address	TS
P30	SMI DAL <26> L	SMI Data & Address	TS
P31	SMI DAL <25> L	SMI Data & Address	TS
P32	SMI DAL <24> L	SMI Data & Address	TS
P33	SMI DAL <23> L	SMI Data & Address	TS
P34	SMI DAL <22> L	SMI Data & Address	TS
P35	SMI DAL <21> L	SMI Data & Address	TS
P36	SMI DAL <20> L	SMI Data & Address	TS
P37	SMI DAL <19> L	SMI Data & Address	TS
P38	SMI DAL <18> L	SMI Data & Address	TS
P39	SMI DAL <17> L	SMI Data & Address	TS
P40	SMI DAL <16> L	SMI Data & Address	TS
P41	SMI DAL <15> L	SMI Data & Address	TS
P42	SMI DAL <14> L	SMI Data & Address	TS
P43	SMI DAL <13> L	SMI Data & Address	TS
P44	SMI DAL <12> L	SMI Data & Address	TS
P45	SMI DAL <11> L	SMI Data & Address	TS
P46	SMI DAL <10> L	SMI Data & Address	TS
P47	SMI DAL <09> L	SMI Data & Address	TS
P48	SMI DAL <08> L	SMI Data & Address	TS
P49	SMI DAL <07> L	SMI Data & Address	TS
P50	SMI DAL <06> L	SMI Data & Address	TS
P51	SMI DAL <05> L	SMI Data & Address	TS
P52	SMI DAL <04> L	SMI Data & Address	TS
P53	SMI DAL <03> L	SMI Data & Address	TS
P54	SMI DAL <02> L	SMI Data & Address	TS

AFM/ARB P03 Connector			
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE
P55	SMI REQ <22> L	SMI Bus Request	TTL
P56	SMI GNT <22> L	SMI Bus Grant	TTL
P57	SMI REQ <26> L	SMI Bus Request	TTL
P58	SMI GNT <26> L	SMI Bus Grant	TTL
P59	SMI DAL <01> L	SMI Data & Address	TS
P60	SMI DAL <00> L	SMI Data & Address	TS

Table A-11
AFM/ARB Module, Connector P04 Pinout (14 Pins)

AFM/ARB P04 Connector			
PIN	SIGNAL NAME	DESCRIPTION	SIG TYPE
P1	—	—	—
P2	Serial Data In H	Data In	TTL
P3	Serial Data Out H	Data Out	TTL
P4	—	—	—
P5	GND	Ground	—
P6	—	—	—
P7	—	—	—
P8	GND	Ground	—
P9	—	—	—
P10	—	—	—
P11	GND	Ground	—
P12	INIT All L	INIT All	TTL
P13	BUF Reset L	Buffered Reset	TTL
P14	—	—	—

Appendix B
MC5600/5700 Specifications

Element	Characteristic	Specification
CMPU Processor Module	Type of Microprocessor	68020 (16.667 MHz)
	Number of CPU Data Lines	32
	Number of CPU Address Lines	32
	Virtual Memory Size	3 GBytes program 1 GByte system
	Number of Address Modes	14
	Number of Interrupt Levels	7
	Protection Modes	Execute only Read/Execute Read/Execute/Write No Access
Memory Management	Cache Size	8 KBytes (2 x 512 8-byte entries)
	Cache Data Entry Size	8 Bytes
	Translation Buffer Size	1024 entries
	Page Size	4 KBytes
CMPU Local Devices	Number of RS-232-C Serial Ports	3
	Baud Rate Range Per Port	110 to 19200 (110 to 38400 on Port P1A)
	Supported Bootstrap Devices	Floppy Disk Winchester Disk
	Programmable Bootstrap ROM	8 KBytes
SMI Memory Bus	Number of Address Lines	28
	Number of Data lines	32
	System Memory Physical Address space	128 MBytes
	System I/O Physical Address space (including all MULTIBUSs)	128 MBytes
	Transfer rate	26.6 MBytes/sec (64-bit words) 20 MBytes/sec (32-bit words)
CMM Memory Module	Size	2 MBytes or 4 MBytes
	Minimum Configuration	2 MBytes (MC5600) 4 MBytes (MC5700)
	Maximum User Physical Memory	16 MBytes (MC5600) 32 MBytes (MC5700)
	Read Access time	200 nsec (4 Bytes) 300 nsec (8 Bytes)
	Write cycle	600 nsec (1,2, or 3 Bytes) 400 nsec (4 Bytes)

Element	Characteristic	Specification
MULTIBUS Interface	Number of MULTIBUS Address Lines	24
	MULTIBUS Memory Addressed	16 MBytes
	MULTIBUS I/O Addressed	64 KBytes
	Block Mode transfer rate	4-6 MBytes/sec
Backplane	Total Slots	15 (MC5600 Base System) 30 (MC5700 Base System)
	Maximum MULTIBUS Slots	13 (MC5600 Base System) 13/MULTIBUS (MC5700)
	STD+ Slots	9 (not available on Pedestal)
Electrical	Pedestal	14 amps @ 120 Volts 90-130 VAC, 47-63 Hz 8 amps @ 230 Volt 180-264 VAC, 47-63 Hz
	Tabletop Rack Mount	9 amps @ 120 Volts 105-127 VAC, 47-63 Hz 6 amps @ 230 Volt 210-254 VAC, 47-63 Hz *Japan: 10 amps @ 100 Volts* *90-110 VAC, 47-63 Hz*
	Wide Cabinet	10 amps @ 230 Volts (15-slot) 20 amps @ 230 Volts (30-slot) 180-264 VAC, 47-63 Hz
Environment	Operating temperature	10° to 40° C
	Storage Temperature	-40° to 65° C
	Relative Humidity (operating)	10-80% non-condensing
	Relative Humidity (storage)	10-80% non-condensing
	Noise level	Below NC-45 (Pedestal/Wide Cabinet) Below NC-50 (Tabletop) Below NC-50 (Rack Mount)
Dimensions	Pedestal	29" high x 13" wide x 24" deep
	Tabletop	12.25" high x 19" wide x 28" deep
	Rack Mount	49.5" high x 23.5" wide x 33" deep
	Wide Cabinet	49.5" high x 28.5" wide x 33" deep
Weight	Pedestal	200 lb
	Tabletop	120 lb
	Rack Mount	120 lb (without cabinet) 270 lb (with cabinet)
	Wide Cabinet	800 lb (fully configured)
Regulatory	Safety	UL, VDE, CSA
	EMI/RFI	FCC Class A, VDE Class A

Appendix C
DUART Specification

Signetics

SCN2681
Dual Asynchronous Receiver/ Transmitter (DUART)

Product Specification

Microprocessor Products

DESCRIPTION

The Signetics SCN2681 Dual Universal Asynchronous Receiver/Transmitter (DUART) is a single chip MOS-LSI communications device that provides two independent full-duplex asynchronous receiver/transmitter channels in a single package. It interfaces directly with microprocessors and may be used in a polled or interrupt driven system.

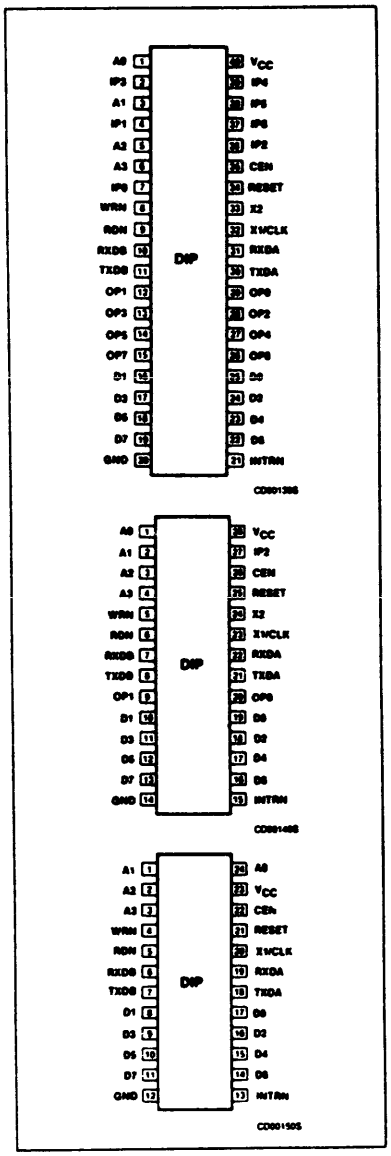
The operating mode and data format of each channel can be programmed independently. Additionally, each receiver and transmitter can select its operating speed as one of eighteen fixed baud rates, a 16x clock derived from a programmable counter/timer, or an external 1x or 16x clock. The baud rate generator and counter/timer can operate directly from a crystal or from external clock inputs. The ability to independently program the operating speed of the receiver and transmitter make the DUART particularly attractive for dual-speed channel applications such as clustered terminal systems.

Each receiver is quadruply buffered to minimize the potential of receiver overrun or to reduce interrupt overhead in interrupt driven systems. In addition, a flow control capability is provided to disable a remote DUART transmitter when the buffer of the receiving device is full.

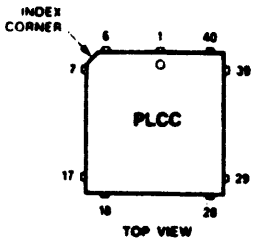
FEATURES

- Dual full-duplex asynchronous receiver/transmitter
- Quadruple buffered receiver data registers
- Programmable data format
  - 5 to 8 data bits plus parity
  - Odd, even, no parity or force parity
  - 1, 1.5 or 2 stop bits programmable in 1/16 bit increments
- Programmable baud rate for each receiver and transmitter selectable from:
  - 18 fixed rates: 50 to 38.4K baud
  - One user defined rate derived from programmable timer/ counter
  - External 1x or 16x clock
- Parity, framing, and overrun error detection
- False start bit detection
- Line break detection and generation
- Programmable channel mode
  - Normal (full duplex)
  - Automatic echo
  - Local loopback
  - Remote loopback
- Multi-function programmable 16-bit counter/timer
- Multi-function 7-bit input port
  - Can serve as clock or control inputs
  - Change of state detection on four inputs
- Multi-function 8-bit output port
  - Individual bit set/reset capability
  - Outputs can be programmed to be status/interrupt signals
- Versatile interrupt system
  - Single interrupt output with eight maskable interrupting conditions
  - Output port can be configured to provide a total of up to six separate wire-OR'able interrupt outputs
- Maximum data transfer: 1X-1MB/ sec, 16X-125KB/sec
- Automatic wake-up mode for multidrop applications
- Start-end break interrupt/status
- Detects break which originates in the middle of a character
- On-chip crystal oscillator
- TTL compatible
- Single +5V power supply

PIN CONFIGURATION



PIN CONFIGURATION (Continued)



Pin	Function	Pin	Function
1	NC	23	NC
2	A0	24	INTRN
3	IP3	25	D6
4	A1	26	D4
5	IP1	27	D2
6	A2	28	D0
7	A3	29	OP6
8	IP0	30	OP4
9	WRN	31	OP2
10	RDN	32	OP0
11	RXDB	33	TXDA
12	NC	34	NC
13	TXDB	35	RXDA
14	OP1	36	X1/CLK
15	OP3	37	X2
16	OP5	38	RESET
17	OP7	39	CEN
18	D1	40	IP2
19	D3	41	IP6
20	D5	42	IP5
21	D7	43	IP4
22	GND	44	V_{CC}

Also provided on the SCN2681 are a multipurpose 7-bit input port and a multipurpose 8-bit output port. These can be used as general purpose I/O ports or can be assigned specific functions (such as clock inputs or status/interrupt outputs) under program control.

The SCN2681 is available in four package versions: 40-pin and 28-pin, both 0.6" wide DIPs; a compact 24-pin 0.4" wide DIP; and a 44-pin PLCC.

ORDERING CODE

PACKAGES	V_{CC} = 5V ± 5%, T_A = 0°C to 70°C			
	24-Pin[1]	28-Pin[2]	40-Pin[2]	44-Pin
Ceramic DIP	Not available	SCN2681AC1I28	SCN2681AC1I40	Not available
Plastic DIP	SCN2681AC1N24	SCN2681AC1N28	SCN2681AC1N40	Not available
Plastic LCC	Not available	Not available	Not available	SCN2681AC1A44

[1]400 mil wide DIP
[2]600 mil wide DIP

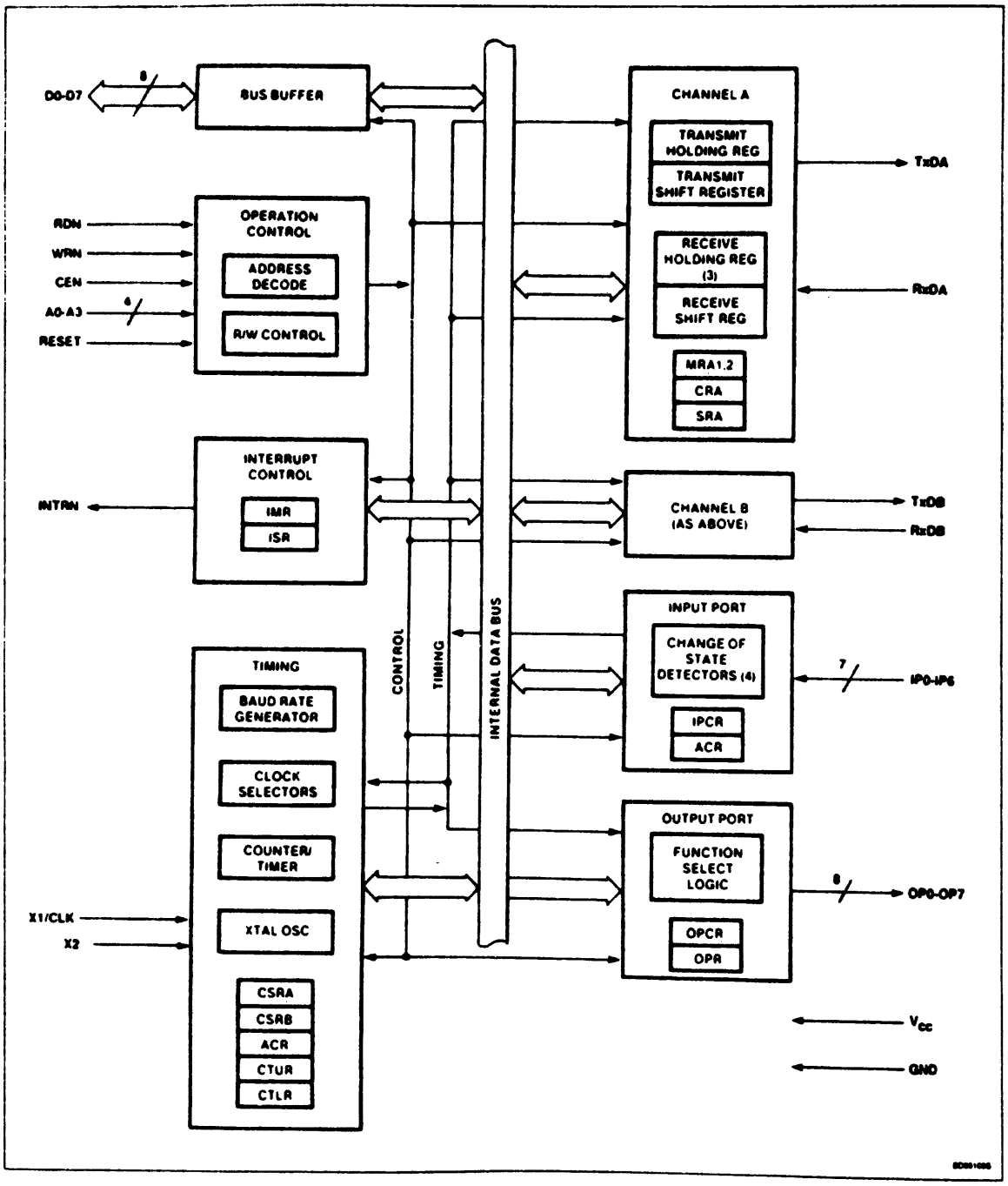
PIN DESCRIPTION

MNEMONIC	APPLICABLE			TYPE	NAME AND FUNCTION
	40	28	24		
D0-D7	X	X	X	I/O	**Data Bus:** Bidirectional 3-state data bus used to transfer commands, data and status between the DUART and the CPU. D0 is the least significant bit.
CEN	X	X	X	I	**Chip Enable:** Active low input signal. When low, data transfers between the CPU and the DUART are enabled on D0-D7 as controlled by the WRN, RDN and A0-A3 inputs. When high, places the D0-D7 lines in the 3-state condition.
WRN	X	X	X	I	**Write Strobe:** When low and CEN is also low, the contents of the data bus is loaded into the addressed register. The transfer occurs on the rising edge of the signal.
RDN	X	X	X	I	**Read Strobe:** When low and CEN is also low, causes the contents of the addressed register to be presented on the data bus. The read cycle begins on the falling edge of RDN.
A0-A3	X	X	X	I	**Address Inputs:** Select the DUART internal registers and ports for read/write operations.
RESET	X	X	X	I	**Reset:** A high level clears internal registers (SRA, SRB, IMR, ISR, OPR, OPCR), puts OP0-OP7 in the high state, stops the counter/timer, and puts channels A and B in the inactive state, with the TxDA and TxDB outputs in the mark (high) state.
INTRN	X	X	X	O	**Interrupt Request:** Active low, open drain, output which signals the CPU that one or more of the eight maskable interrupting conditions are true.
X1/CLK	X	X	X	I	**Crystal 1:** Crystal or external clock input. A crystal or clock of the specified limits must be supplied at all times. When a crystal is used, a capacitor must be connected from this pin to ground (see figure 5).
X2	X	X		I	**Crystal 2:** Connection for other side of the crystal. When a crystal is used, a capacitor must be connected from this pin to ground (see figure 5).
RxDA	X	X	X	I	**Channel A Receiver Serial Data Input:** The least significant bit is received first. 'Mark' is high, 'space' is low.

PIN DESCRIPTION (Continued)

MNEMONIC	APPLICABLE			TYPE	NAME AND FUNCTION
	40	28	24		
RxDB	X	X	X	I	**Channel B Receiver Serial Data Input:** The least significant bit is received first. 'Mark' is high, 'space' is low.
TxDA	X	X	X	O	**Channel A Transmitter Serial Data Output:** The least significant bit is transmitted first. This output is held in the 'mark' condition when the transmitter is disabled, idle, or when operating in local loopback mode. 'Mark' is high, 'space' is low.
TxDB	X	X	X	O	**Channel B Transmitter Serial Data Output:** The least significant bit is transmitted first. This output is held in the 'mark' condition when the transmitter is disabled, idle, or when operating in local loopback mode. 'Mark' is high, 'space' is low.
OP0	X	X		O	**Output 0:** General purpose output, or channel A request to send (RTSAN, active low). Can be deactivated automatically on receive or transmit.
OP1	X	X		O	**Output 1:** General purpose output, or channel B request to send (RTSBN, active low). Can be deactivated automatically on receive or transmit.
OP2	X			O	**Output 2:** General purpose output, or channel A transmitter 1X or 16X clock output, or channel A receiver 1X clock output.
OP3	X			O	**Output 3:** General purpose output, or open drain, active low counter/timer output, or channel B transmitter 1X clock output, or channel B receiver 1X clock output.
OP4	X			O	**Output 4:** General purpose output, or channel A open drain, active low, RxRDYA/FFULLA output.
OP5	X			O	**Output 5:** General purpose output, or channel B open drain, active low, RxRDYB/FFULLB output.
OP6	X			O	**Output 6:** General purpose output, or channel A open drain, active low, TxRDYA output.
OP7	X			O	**Output 7:** General purpose output, or channel B open drain, active low, TxRDYB output.
IP0	X			I	**Input 0:** General purpose input, or channel A clear to send active low input (CTSAN).
IP1	X			I	**Input 1:** General purpose input, or channel B clear to send active low input (CTSBN).
IP2	X	X		I	**Input 2:** General purpose input, or counter/timer external clock input.
IP3	X			I	**Input 3:** General purpose input, or channel A transmitter external clock input (TxCA). When the external clock is used by the transmitter, the transmitted data is clocked on the falling edge of the clock.
IP4	X			I	**Input 4:** General purpose input, or channel A receiver external clock input (RxCA). When the external clock is used by the receiver, the received data is sampled on the rising edge of the clock.
IP5	X			I	**Input 5:** General purpose input, or channel B transmitter external clock input (TxCB). When the external clock is used by the transmitter, the transmitted data is clocked on the falling edge of the clock.
IP6	X			I	**Input 6:** General purpose input or channel B receiver external clock input (RxCB). When the external clock is used by the receiver, the received data is sampled on the rising edge of the clock.
V_{CC}	X	X	X	I	**Power Supply:** +5V supply input
GND	X	X	X	I	**Ground**

BLOCK DIAGRAM



BLOCK DIAGRAM

The 2681 DUART consists of the following eight major sections: data bus buffer, operation control, interrupt control, timing, communications channels A and B, input port and output port. Refer to the block diagram.

Data Bus Buffer

The data bus buffer provides the interface between the external and internal data busses. It is controlled by the operation control block to allow read and write operations to take place between the controlling CPU and the DUART.

Operation Control

The operation control logic receives operation commands from the CPU and generates appropriate signals to internal sections to control device operation. It contains address decoding and read and write circuits to permit communications with the microprocessor via the data bus buffer.

Interrupt Control

A single active low interrupt output (INTRN) is provided which is activated upon the occurrence of any of eight internal events. Associated with the interrupt system are the interrupt mask register (IMR) and the interrupt status register (ISR). The IMR may be programmed to select only certain conditions to cause INTRN to be asserted. The ISR can be read by the CPU to determine all currently active interrupting conditions.

Outputs OP3-OP7 can be programmed to provide discrete interrupt outputs for the transmitters, receivers, and counter/timer.

Timing Circuits

The timing block consists of a crystal oscillator, a baud rate generator, a programmable 16-bit counter/timer, and four clock selectors. The crystal oscillator operates directly from a 3.6864MHz crystal connected across the X1/CLK and X2 inputs. If an external clock of the appropriate frequency is available, it may be connected to X1/CLK. The clock serves as the basic timing reference for the baud rate generator (BRG), the counter/timer, and other internal circuits. A clock signal within the limits specified in the specifications section of this data sheet must always be supplied to the DUART.

If an external is used instead of a crystal, both X1 and X2 (of the 28/40 pin versions) should be driven using a configuration similar to the one in figure 5. For the 24 pin version in which only X1/CLK is available, the input clock must be capable of attaining a V_{IH} of 4.4 volts.

The baud rate generator operates from the oscillator or external clock input and is capable of generating 18 commonly used data communications baud rates ranging from 50 to 38.4K baud. The clock outputs from the BRG are at 16X the actual baud rate. The counter/timer can be used as a timer to produce a 16X clock for any other baud rate by counting down the crystal clock or an external clock. The four clock selectors allow the independent selection, for each receiver and transmitter, of any of these baud rates or an external timing signal.

The counter/timer (C/T) can be programmed to use one of several timing sources as its input. The output of the C/T is available to the clock selectors and can also be programmed to be output at OP3. In the counter mode, the contents of the C/T can be read by the CPU and it can be stopped and started under program control. In the timer mode, the C/T acts as a programmable divider.

Communications Channels A And B

Each communications channel of the 2681 comprises a full duplex asynchronous receiver/transmitter (UART). The operating frequency for each receiver and transmitter can be selected independently from the baud rate generator, the counter timer, or from an external input.

The transmitter accepts parallel data from the CPU, converts it to a serial bit stream, inserts the appropriate start, stop, and optional parity bits and outputs a composite serial stream of data on the TxD output pin. The receiver accepts serial data on the RxD pin, converts this serial input to parallel format, checks for start bit, stop bit, parity bit (if any), or break condition and sends an assembled character to the CPU.

The input port pulse detection circuitry uses a 38.4KHz sampling clock derived from one of the baud rate generator taps. This results in a sampling period of slightly more than 25μsec (this assumes that the clock input is 3.6864MHz). The detection circuitry, in order to guarantee that a true change in level has occurred, requires two successive samples at the new logic level be observed. As a consequence, the minimum duration of the signal change is 25μsec if the transition occurs "coincident with the first sample pulse." The 50μsec time refers to the situation in which the change of state is "just missed" and the first change of state is not detected until 25μsec later.

Input Port

The inputs to this unlatched 7-bit port can be read by the CPU by performing a read operation at address D_{16}. A high input results in a logic 1 while a low input results in a logic 0. D_7 will always be read as a logic 1. The pins of this port can also serve as auxiliary inputs to certain portions of the DUART logic.

Four change-of-state detectors are provided which are associated with inputs IP3, IP2, IP1, and IP0. A high-to-low or low-to-high transition of these inputs, lasting longer than 25-50μs, will set the corresponding bit in the input port change register. The bits are cleared when the register is read by the CPU. Any change of state can also be programmed to generate an interrupt to the CPU.

Output Port

The 8-bit multi-purpose output port can be used as a general purpose output port, in which case the outputs are the complements of the output port register (OPR). OPR[n] = 1 results in OP[n] = low and vice versa. Bits of the OPR can be individually set and reset. A bit is set by performing a write operation at address E_{16} with the accompanying data specifying the bits to be set (1 = set, 0 = no change). Likewise, a bit is reset by a write at address F_{16} with the accompanying data specifying the bits to be reset (1 = reset, 0 = no change).

Outputs can be also individually assigned specific functions by appropriate programming of the channel A mode registers (MR1A, MR2A), the channel B mode registers (MR1B, MR2B), and the output port configuration register (OPCR).

OPERATION

Transmitter

The 2681 is conditioned to transmit data when the transmitter is enabled through the command register. The 2681 indicates to the CPU that it is ready to accept a character by setting the TxRDY bit in the status register. This condition can be programmed to generate an interrupt request at OP6 or OP7 and INTRN. When a character is loaded into the transmit holding register (THR), the above conditions are negated. Data is transferred from the holding register to transmit shift register when it is idle or has completed transmission of the previous character. The TxRDY conditions are then asserted again which means one full character time of buffering is provided. Characters cannot be loaded into the THR while the transmitter is disabled

The transmitter converts the parallel data from the CPU to a serial bit stream on the TxD output pin. It automatically sends a start bit followed by the programmed number of data bits, an optional parity bit, and the programmed number of stop bits. The least significant bit is sent first. Following the transmission of the stop bits, if a new character is not available in the THR, the TxD output remains high and the TxEMT bit in the status register (SR) will be set to 1. Transmission resumes and the TxEMT bit is cleared when the CPU loads a new character into the THR. If the transmitter is disabled, it continues operating until the character currently being

transmitted is completely sent out. The transmitter can be forced to send a continuous low condition by issuing a send break command.

The transmitter can be reset through a software command. If it is reset, operation ceases immediately and the transmitter must be enabled through the command register before resuming operation. If CTS operation is enabled, the CTSN input must be low in order for the character to be transmitted. If it goes high in the middle of a transmission, the character in the shift register is transmitted and TxDA then remains in the marking state until CTSN goes low. The transmitter can also control the deactivation of the RTSN output. If programmed, the RTSN output will be reset one bit time after the character in the transmit shift register and transmit holding register (if any) are completely transmitted, if the transmitter has been disabled.

Receiver

The 2681 is conditioned to receive data when enabled through the command register. The receiver looks for a high to low (mark to space) transition of the start bit on the RxD input pin. If a transition is detected, the state of the RxD pin is sampled each 16X clock for 7 ½ clocks (16X clock mode) or at the next rising edge of the bit time clock (1X clock mode). If RxD is sampled high, the start bit is invalid and the search for a valid start bit begins again. If RxD is still low, a valid start bit is assumed and the receiver continues to sample the input at one bit time intervals at the theoretical center of the bit, until the proper number of data bits and the parity bit (if any) have been assembled, and one stop bit has been detected. The least significant bit is received first. The data is then transferred to the receive holding register (RHR) and the RxRDY bit in the SR is set to a 1. This condition can be programmed to generate an interrupt at OP4 or OP5 and INTRN. If the character length is less than eight bits, the most significant unused bits in the RHR are set to zero.

After the stop bit is detected, the receiver will immediately look for the next start bit. However, if a non-zero character was received without a stop bit (framing error) and RxD remains low for one half of the bit period after the stop bit was sampled, then the receiver operates as if a new start bit transition had been detected at that point (one-half bit time after the stop bit was sampled).

The parity error, framing error, overrun error and received break state (if any) are strobed into the SR at the received character boundary, before the RxRDY status bit is set. If a break condition is detected (RxD is low for the entire character including the stop bit), a character consisting of all zeros will be loaded into the RHR and the received break bit in the SR is set to 1. The RxD input must return to a high condition for at least one-half bit time before a search for the next start bit begins.

The RHR consists of a first-in-first-out (FIFO) stack with a capacity of three characters. Data is loaded from the receive shift register into the topmost empty position of the FIFO. The RxRDY bit in the status register is set whenever one or more characters are available to be read, and a FFULL status bit is set if all three stack positions are filled with data. Either of these bits can be selected to cause an interrupt. A read of the RHR outputs the data at the top of the FIFO. After the read cycle, the data FIFO and its associated status bits (see below) are 'popped' thus emptying a FIFO position for new data.

In addition to the data word, three status bits (parity error, framing error, and received break) are also appended to each data character in the FIFO (overrun is not). Status can be provided in two ways, as programmed by the error mode control bit in the mode register. In the 'character' mode, status is provided on a character-by-character basis: the status applies only to the character at the top of the FIFO. In the 'block' mode, the status provided in the SR for these three bits is the logical OR of the status for all characters coming to the top of the FIFO since the last 'reset error' command was issued. In either mode reading the SR does not affect the FIFO. The FIFO is 'popped' only when the RHR is read. Therefore the status register should be read prior to reading the FIFO.

If the FIFO is full when a new character is received, that character is held in the receive shift register until a FIFO position is available. If an additional character is received while this state exits, the contents of the FIFO are not affected: the character previously in the shift register is lost and the overrun error status bit (SR[4]) will be set-upon receipt of the start bit of the new (overruning) character.

The receiver can control the deactivation of RTS. If programmed to operate in this mode, the RTSN output will be negated when a valid start bit was received and the FIFO is full. When a FIFO position becomes available, the RTSN output will be re-asserted automatically. This feature can be used to prevent an overrun, in the receiver, by connecting the RTSN output to the CTSN input of the transmitting device.

If the receiver is disabled, the FIFO characters can be read. However, no additional characters can be received until the receiver is enabled again. If the receiver is reset, the FIFO and all of the receiver status, and the corresponding output ports and interrupt are reset. No additional characters can be received until the receiver is enabled again.

Multidrop Mode

The DUART is equipped with a wake up mode used for multidrop applications. This mode is selected by programming bits MR1A[4:3] or MR1B[4:3] to '11' for channels A and B respectively. In this mode of operation, a 'master' station transmits an address character followed by data characters for the addressed 'slave' station. The slave stations, with receivers that are normally disabled, examine the received data stream and 'wake-up' the CPU (by setting RxRDY) only upon receipt of an address character. The CPU compares the received address to its station address and enables the receiver if it wishes to receive the subsequent data characters. Upon receipt of another address character, the CPU may disable the receiver to initiate the process again.

A transmitted character consists of a start bit, the programmed number of data bits, an address/data (A/D) bit, and the programmed number of stop bits. The polarity of the transmitted A/D bit is selected by the CPU by programming bit MR1A[2]/MR1B[2]. MR1A[2]/MR1B[2] = 0 transmits a zero in the A/D bit position, which identifies the corresponding data bits as data, while MR1A[2]/MR1B[2] = 1 transmits a one in the A/D bit position, which identifies the corresponding data bits as an address. The CPU should program the mode register prior to loading the corresponding data bits into the THR.

In this mode, the receiver continuously looks at the received data stream, whether it is enabled or disabled. If disabled, it sets the RxRDY status bit and loads the character into the RHR FIFO if the received A/D bit is a one (address tag), but discards the received character if the received A/D bit is a zero (data tag). If enabled, all received characters are transferred to the CPU via the RHR. In either case, the data bits are loaded into the data FIFO while the A/D bit is loaded into the status FIFO position normally used for parity error (SRA[5] or SRB[5]). Framing error, overrun error, and break detect operate normally whether or not the receiver is enabled.

PROGRAMMING

The operation of the DUART is programmed by writing control words into the appropriate registers. Operational feedback is provided via status registers which can be read by the CPU. The addressing of the registers is described in table 1.

The contents of certain control registers are initialized to zero on RESET. Care should be exercised if the contents of a register are changed during operation, since certain changes may cause operational problems.

Table 1. 2681 REGISTER ADDRESSING

A3	A2	A1	A0	READ (RDN = 0)	WRITE (WRN = 0)
0	0	0	0	Mode Register A (MR1A, MR2A)	Mode Register A (MR1A, MR2A)
0	0	0	1	Status Register A (SRA)	Clock Select Reg. A (CSRA)
0	0	1	0	*Reserved*	Command Register A (CRA)
0	0	1	1	RX Holding Register A (RHRA)	TX Holding Register A (THRA)
0	1	0	0	Input Port Change Reg. (IPCR)	Aux. Control Register (ACR)
0	1	0	1	Interrupt Status Reg. (ISR)	Interrupt Mask Reg. (IMR)
0	1	1	0	Counter/Timer Upper (CTU)	C/T Upper Register (CTUR)
0	1	1	1	Counter/Timer Lower (CTL)	C/T Lower Register (CTLR)
1	0	0	0	Mode Register B (MR1B, MR2B)	Mode Register B (MR1B, MR2B)
1	0	0	1	Status Register B (SRB)	Clock Select Reg. B (CSRB)
1	0	1	0	*Reserved*	Command Register B (CRB)
1	0	1	1	RX Holding Register B (RHRB)	TX Holding Register B (THRB)
1	1	0	0	*Reserved*	*Reserved*
1	1	0	1	Input Port	Output Port Conf. Reg. (OPCR)
1	1	1	0	Start Counter Command	Set Output Port Bits Command
1	1	1	1	Stop Counter Command	Reset Output Port Bits Command

For example, changing the number of bits per character while the transmitter is active may cause the transmission of an incorrect character. In general, the contents of the MR, the CSR, and the OPCR should only be changed while the receiver(s) and transmitter(s) are not enabled, and certain changes to the ACR should only be made while the C/T is stopped.

Mode registers 1 and 2 of each channel are accessed via independent auxiliary pointers. The pointer is set to MR1x by RESET or by issuing a 'reset pointer' command via the corresponding command register. Any read or write of the mode register while the pointer is at MR1x, switches the pointer to MR2x. The pointer then remains at MR2x, so that subsequent accesses are always to MR2x unless the pointer is reset to MR1x as described above.

Mode, command, clock select, and status registers are duplicated for each channel to provide total independent operation and control. Refer to table 2 for register bit descriptions. The reserved registers at addresses H'02' and H'0A' should never be read during normal operation since they are reserved for internal diagnostics.

MR1A – channel A Mode Register 1

MR1A is accessed when the channel A MR pointer points to MR1. The pointer is set to MR1 by RESET or by a 'set pointer' command applied via CRA. After reading or writing MR1A, the pointer will point to MR2A.

MR1A[7] – Channel A Receiver Request-to-Send Control

This bit controls the deactivation of the RTSAN output (OP0) by the receiver. This output is normally asserted by setting OPR[0] and negated by resetting OPR[0]. MR 1A[7] = 1 causes RTSAN to be negated upon receipt of a valid start bit if the channel A FIFO is full. However, OPR[0] is not reset and RTSAN will be asserted again when an empty FIFO position is available. This feature can be used for flow control to prevent overrun in the receiver by using the RTSAN output signal to control the CTSN input of the transmitting device.

MR1A[6] – Channel A Receiver Interrupt Select

This bit selects either the channel A receiver ready status (RXRDY) or the channel A FIFO full status (FFULL) to be used for CPU interrupts. It also causes the selected bit to be output on OP4 if it is programmed as an interrupt output via the OPCR.

MR1A[5] – Channel A Error Mode Select

This bit selects the operating mode of the three FIFOed status bits (FE, PE, received break) for channel A. In the 'character' mode, status is provided on a character-by-character basis: the status applies only to the character at the top of the FIFO. In the 'block' mode, the status provided in the SR for these bits is the accumulation (logical OR) of the status for all characters coming to the top of the FIFO since the last 'reset error' command for channel A was issued.

Table 2. REGISTER BIT FORMATS

	BIT7	BIT6	BIT5	BIT4 BIT3	BIT2	BIT1 BIT0
	RX RTS CONTROL	RX INT SELECT	ERROR MODE	PARITY MODE	PARITY TYPE	BITS PER CHAR.
MR1A MR1B	0 = no 1 = yes	0 = RXRDY 1 = FFULL	0 = char 1 = block	00 = with parity 01 = force parity 10 = no parity 11 = multi-drop mode	0 = even 1 = odd	00 = 5 01 = 6 10 = 7 11 = 8

Table 2. REGISTER BIT FORMATS (Continued)

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
MR2A MR2B	CHANNEL MODE		Tx RTS CONTROL	CTS ENABLE Tx	STOP BIT LENGTH*			
	00 = Normal 01 = Auto echo 10 = Local loop 11 = Remote loop		0 = no 1 = yes	0 = no 1 = yes	0 = 0.563 1 = 0.625 2 = 0.688 3 = 0.750	4 = 0.813 5 = 0.875 6 = 0.938 7 = 1.000	8 = 1.563 9 = 1.625 A = 1.688 B = 1.750	C = 1.813 D = 1.875 E = 1.938 F = 2.000

*Add 0.5 to values shown for 0–7 if channel is programmed for 5 bits/char.

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
CSRA CSRB	RECEIVER CLOCK SELECT				TRANSMITTER CLOCK SELECT			
	See Text				See Text			

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
CRA CRB		MISCELLANEOUS COMMANDS			DISABLE Tx	ENABLE Tx	DISABLE Rx	ENABLE Rx
	not used - must be 0	See Text			0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
SRA SRB	RECEIVED BREAK	FRAMING ERROR	PARITY ERROR	OVERRUN ERROR	TxEMT	TxRDY	FFULL	RxRDY
	0 = no 1 = yes *	0 = no 1 = yes *	0 = no 1 = yes *	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes

*These status bits are appended to the corresponding data character in the receive FIFO. A read of the status register provides these bits (7:5) from the top of the FIFO together with bits (4:0) These bits are cleared by a 'reset error status' command. In character mode they are discarded when the corresponding data character is read from the FIFO

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
OPCR	OP7	OP6	OP5	OP4	OP3		OP2	
	0 = OPR[7] 1 = TxRDYB	0 = OPR[6] 1 = TxRDYA	0 = OPR[5] 1 = RxRDY/ FFULLB	0 = OPR[4] 1 = RxRDY/ FFULLA	00 = OPR[3] 01 = C/T OUTPUT 10 = TxCB(1X) 11 = RxCB(1X)		00 = OPR[2] 01 = TxCA(16X) 10 = TxCA(1X) 11 = RxCA(1X)	

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
ACR	BRG SET SELECT	COUNTER/TIMER MODE AND SOURCE			DELTA IP3 INT	DELTA IP2 INT	DELTA IP1 INT	DELTA IP0 INT
	0 = set1 1 = set2	See table 4			0 = off 1 = on	0 = off 1 = on	0 = off 1 = on	0 = off 1 = on

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
IPCR	DELTA IP3	DELTA IP2	DELTA IP1	DELTA IP0	IP3	IP2	IP1	IP0
	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = low 1 = high	0 = low 1 = high	0 = low 1 = high	0 = low 1 = high

Table 2. REGISTER BIT FORMATS (Continued)

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
ISR	INPUT PORT CHANGE	DELTA BREAK B	RxRDY/ FFULLB	TxRDYB	COUNTER READY	DLETA BREAK A	RxRDY/ FFULLA	TxRDYA
	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes	0 = no 1 = yes

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
IMR	IN. PORT CHANGE INT	DELTA BREAK B INT	RxRDY/ FFULLB INT	TxRDYB INT	COUNTER READY INT	DELTA BREAK A INT	RxRDY/ FFULLA INT	TxRDYA INT
	0 = off 1 = on	0 = off 1 = on	0 = off 1 = on	0 = off 1 = on	0 = off 1 = on	0 = off 1 = on	0 = off 1 = on	0 = off 1 = on

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
CTUR	C/T[15]	C/T[14]	C/T[13]	C/T[12]	C/T[11]	C/T[10]	C/T[9]	C/T[8]

	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
CTLR	C/T[7]	C/T[6]	C/T[5]	C/T[4]	C/T[3]	C/T[2]	C/T[1]	C/T[0]

MR1A[4:3] - Channel A Parity Mode Select
If 'with parity' or 'force parity' is selected, a parity bit is added to the transmitted character and the receiver performs a parity check on incoming data. MR1A[4:3] = 11 selects channel A to operate in the special multidrop mode described in the Operation section.

MR1A[2] - Channel A Parity Type Select
This bit selects the parity type (odd or even) if the 'with parity' mode is programmed by MR1A[4:3], and the polarity of the forced parity bit if the 'force parity' mode is programmed. It has no effect if the 'no parity' mode is programmed. In the special multidrop mode it selects the polarity of the A/D bit.

MR1A[1:0] - Channel A Bits Per Character Select
This field selects the number of data bits per character to be transmitted and received. The character length does not include the start, parity, and stop bits.

MR2A - Channel A Mode Register 2

MR2A is accessed when the channel A MR pointer points to MR2, which occurs after any access to MR1A. Accesses to MR2A do not change the pointer.

MR2A[7:6] - Channel A Mode Select
Each channel of the DUART can operate in one of four modes. MR2A[7:6] = 00 is the normal mode, with the transmitter and receiver operating independently. MR2A[7:6] = 01 places the channel in the automatic echo mode, which automatically retransmits the received data. The following conditions are true while in automatic echo mode:

1. Received data is reclocked and retransmitted on the TxDA output.
2. The receive clock is used for the transmitter.
3. The receiver must be enabled, but the transmitter need not be enabled.
4. The channel A TxRDY and TxEMT status bits are inactive.
5. The received parity is checked, but is not regenerated for transmission, i.e., transmitted parity bit is as received.
6. Character framing is checked, but the stop bits are retransmitted as received.
7. A received break is echoed as received until the next valid start bit is detected.
8. CPU to receiver communication continues normally, but the CPU to transmitter link is disabled.

Two diagnostic modes can also be configured. MR2A[7:6] = 10 selects local loopback mode. In this mode:

1. The transmitter output is internally connected to the receiver input.
2. The transmit clock is used for the receiver.
3. The TxDA output is held high.
4. The RxDA input is ignored.
5. The transmitter must be enabled, but the receiver need not be enabled.
6. CPU to transmitter and receiver communications continue normally.

The second diagnostic mode is the remote loopback mode, selected by MR2A[7:6] = 11. In this mode:

1. Received data is relocked and retransmitted on the TxDA output.
2. The receive clock is used for the transmitter.
3. Received data is not sent to the local CPU, and the error status conditions are inactive.
4. The received parity is not checked and is not regenerated for transmission, i.e., transmitted parity bit is as received.
5. The receiver must be enabled.
6. Character framing is not checked, and the stop bits are retransmitted as received.
7. A received break is echoed as received until the next valid start bit is detected.

The user must exercise care when switching into and out of the various modes. The selected mode will be activated immediately upon mode selection, even if this occurs in the middle of a received or transmitted char-

acter. Likewise, if a mode is deselected, the device will switch out of the mode immediately. An exception to this is switching out of autoecho or remote loopback modes: if the de-selection occurs just after the receiver has sampled the stop bit (indicated in autoecho by assertion of RxRDY), and the transmitter is enabled, the transmitter will remain in autoecho mode until the entire stop bit has been retransmitted.

MR2A[5] - Channel A Transmitter Request-to-Send Control
This bit controls the deactivation of the RTSAN output (OP0) by the transmitter. This output is normally asserted by setting OPR[0] and negated by resetting OPR[0]. MR2A[5] = 1 causes OPR[0] to be reset automatically one bit time after the characters in the channel A transmit shift register and in the THR, if any, are completely transmitted, including the programmed number of stop bits, if the transmitter is not enabled. This feature can be used to automatically terminate the transmission of a message as follows:

1. Program auto-reset mode: MR2A[5] = 1.
2. Enable transmitter.
3. Assert RTSAN: OPR[0] = 1.
4. Send message.
5. Verify the message is sent by waiting until the transmit ready status (TxRDY) is asserted. Disable transmitter after the last character is loaded into the channel A THR.
6. The last character will be transmitted and OPR[0] will be reset one bit time after the last stop bit, causing RTSAN to be negated.

MR2A[4] - Channel A Clear-to-send Control
If this bit is 0, CTSAN has no effect on the transmitter. If this bit is a 1, the transmitter checks the state of CTSAN (IP0) each time it is ready to send a character. If IP0 is asserted (low), the character is transmitted. If it is negated (high), the TxDA output remains in the marking state and the transmission is delayed until CTSAN goes low. Changes in CTSAN while a character is being transmitted do not affect the transmission of that character.

MR2A[3:0] - Channel A Stop Bit Length Select
This field programs the length of the stop bit appended to the transmitted character. Stop bit lengths of 9/16 to 1 and 1 9/16 to 2 bits, in increments of 1/16 bit, can be programmed for character lengths of 6, 7, and 8 bits. For a character length of 5 bits, 1 1/16 to 2 stop bits can be programmed in increments of 1/16 bit. The receiver only checks for a 'mark' condition at the center of the first stop bit position (one bit time after the last data bit, or after the parity bit if parity is enabled) in all cases.

If an external 1X clock is used for the transmitter, MR2A[3] = 0 selects one stop bit and MR2A[3] = 1 selects two stop bits to be transmitted.

MR1B - Channel B Mode Register 1

MR1B is accessed when the channel B MR pointer points to MR1. The pointer is set to MR1 by RESET or by a 'set pointer' command applied via CRB. After reading or writing MR1B, the pointer will point to MR2B.

The bit definitions for this register are identical to the bit definitions for MR1A, except that all control actions apply to the channel B receiver and transmitter and the corresponding inputs and outputs.

MR2B - Channel B Mode Register 2

MR2B is accessed when the channel B MR pointer points to MR2, which occurs after any access to MR1B. Accesses to MR2B do not change the pointer.

The bit definitions for this register are identical to the bit definitions for MR2A, except that all control actions apply to the channel B receiver and transmitter and the corresponding inputs and outputs.

CSRA - Channel A Clock Select Register

CSRA[7:4] - Channel A Receiver Clock Select
This field selects the baud rate clock for the channel A receiver as follows:

CSRA[7:4]	Baud Rate ACR[7] = 0	Baud Rate ACR[7] = 1
0 0 0 0	50	75
0 0 0 1	110	110
0 0 1 0	134.5	134.5
0 0 1 1	200	150
0 1 0 0	300	300
0 1 0 1	600	600
0 1 1 0	1,200	1,200
0 1 1 1	1,050	2,000
1 0 0 0	2,400	2,400
1 0 0 1	4,800	4,800
1 0 1 0	7,200	1,800
1 0 1 1	9,600	9,600
1 1 0 0	38.4K	19.2K
1 1 0 1	Timer	Timer
1 1 1 0	IP4-16X	IP4-16X
1 1 1 1	IP4-1X	IP4-1X

The receiver clock is always a 16X clock except for CSRA[7:4] = 1111.

CSRA[3:0] - Channel A Transmitter Clock Select
This field selects the baud rate clock for the channel A transmitter. The field definition is as per CSRA[7:4] except as follows:

CSRA[3:0]	Baud Rate ACR[7] = 0	Baud Rate ACR[7] = 1
1 1 1 0	IP3-16X	IP3-16X
0 1 1 1	IP3-1X	IP3-1X

The transmitter clock is always a 16X clock except for CSRA[3:0] = 1111.

CSRB - Channel B Clock Select Register

CSRB[7:4] - Channel B Receiver Clock Select
This field selects the baud rate clock for the channel B receiver. The field definition is as per CSRA[7:4] except as follows:

CSRB[7:4]	Baud Rate ACR[7] = 0	Baud Rate ACR[7] = 1
1 1 1 0	IP6-16X	IP6-16X
0 1 1 1	IP6-1X	IP6-1X

The receiver clock is always a 16X clock except for CSRB[7:4] = 1111.

CSRB[3:0] - Channel B Transmitter Clock Select
This field selects the baud rate clock for the channel B transmitter. The field definition is as per CSRA[7:4] except as follows:

CSRB[3:4]	Baud Rate ACR[7] = 0	Baud Rate ACR[7] = 1
1 1 1 0	IP5-16X	IP5-16X
1 1 1 1	IP5-1X	IP5-1X

The transmitter clock is always a 16X clock except for CSRB[3:0] = 1111.

CRA - Channel A Command Register

CRA is a register used to supply commands to channel A. Multiple commands can be specified in a single write to CRA as long as the commands are non-conflicting, e.g., the 'enable transmitter' and 'reset transmitter' commands cannot be specified in a single command word.

CRA[6:4] - Channel A Miscellaneous Commands
The encoded value of this field may be used to specify a single command as follows:

CRA[6:4]	COMMAND
0 0 0	No command.
0 0 1	Reset MR pointer. Causes the channel A MR pointer to point to MR1.
0 1 0	Reset receiver. Resets the channel A receiver as if a hardware reset had been applied. The receiver is disabled and the FIFO is flushed.
0 1 1	Reset transmitter. Resets the channel A transmitter as if a hardware reset had been applied.

CRA[6:4]	COMMAND
1 0 0	Reset error status. Clears the channel A Received Break, Parity Error, Framing Error, and Overrun Error bits in the status register (SRA[7:4]). Used in character mode to clear OE status (although RB, PE, and FE bits will also be cleared) and in block mode to clear all error status after a block of data has been received.
1 0 1	Reset channel A break change interrupt. Causes the channel A break detect change bit in the interrupt status register (ISR[2]) to be cleared to zero.
1 1 0	Start break. Forces the TXDA output low (spacing). If the transmitter is empty the start of the break condition will be delayed up to two bit times. If the transmitter is active the break begins when transmission of the character is completed. If a character is in the THR, the start of the break will be delayed until that character, or any others loaded subsequently are transmitted. The transmitter must be enabled for this command to be accepted.
1 1 1	Stop Break. The TXDA line will go high (marking) within two bit times. TXDA will remain high for one bit time before the next character, if any, is transmitted.

CRA[3] - Disable Channel A Transmitter
This command terminates transmitter operation and resets the TxRDY and TxEMT status bits. However, if a character is being transmitted or if a character is in the THR when the transmitter is disabled, the transmission of the character(s) is completed before assuming the inactive state.

CRA[2] - Enable Channel A Transmitter
Enables operation of the channel A transmitter. The TxRDY status bit will be asserted.

CRA[1] - Disable Channel A Receiver
This command terminates operation of the receiver immediately - a character being received will be lost. The command has no effect on the receiver status bits or any other control registers. If the special multidrop mode is programmed, the receiver operates even if it is disabled. See Operation section.

CRA[0] - Enable Channel A Receiver
Enables operation of the channel A receiver. If not in the special wakeup mode, this also forces the receiver into the search for start-bit state.

CRB - Channel B Command Register

CRB is a register used to supply commands to channel B. Multiple commands can be specified in a single write to CRB as long as the commands are non-conflicting, e.g., the 'enable transmitter' and 'reset transmitter' commands cannot be specified in a single command word.

The bit definitions for this register are identical to the bit definitions for CRA, except that all control actions apply to the channel B receiver and transmitter and the corresponding inputs and outputs.

SRA - Channel A Status Register

SRA[7] - Channel A Received Break
This bit indicates that an all zero character of the programmed length has been received without a stop bit. Only a single FIFO position is occupied when a break is received: further entries to the FIFO are inhibited until the RxDA line returns to the marking state for at least one-half a bit time (two successive edges of the internal or external 1x clock).

When this bit is set, the channel A 'change in break' bit in the ISR (ISR[2]) is set. ISR[2] is also set when the end of the break condition, as defined above, is detected.

The break detect circuitry can detect breaks that originate in the middle of a received character. However, if a break begins in the middle of a character, it must persist until at least the end of the next character time in order for it to be detected.

SRA[6] - Channel A Framing Error
This bit, when set, indicates that a stop bit was not detected when the corresponding data character in the FIFO was received. The stop bit check is made in the middle of the first stop bit position.

SRA[5] - Channel A Parity Error
This bit is set when the 'with parity' or 'force parity' mode is programmed and the corresponding character in the FIFO was received with incorrect parity.

In the special multidrop mode the parity error bit stores the received A/D bit.

SRA[4] - Channel A Overrun Error
This bit, when set, indicates that one or more characters in the received data stream have been lost. It is set-upon receipt of a new character when the FIFO is full and a character is already in the receive shift register waiting for an empty FIFO position. When this occurs, the character in the receive shift register (and its break detect, parity error and framing error status, if any) is lost.

This bit is cleared by a 'reset error status' command.

SRA[3] - Channel A Transmitter Empty (TxEMTA)
This bit will be set when the channel A transmitter underruns, i.e., both the transmit holding register (THR) and the transmit shift register are empty. It is set after transmission of the last stop bit of a character if no character is in the THR awaiting transmission. It is reset when the THR is loaded by the CPU or when the transmitter is disabled.

SRA[2] - Channel A Transmitter Ready (TxRDYA)
This bit, when set, indicates that the THR is empty and ready to be loaded with a character. This bit is cleared when the THR is loaded by the CPU and is set when the character is transferred to the transmit shift register. TxRDY is reset when the transmitter is disabled and is set when the transmitter is first enabled, viz., characters loaded into the THR while the transmitter is disabled will not be transmitted.

SRA[1] - Channel A FIFO Full (FFULLA)
This bit is set when a character is transferred from the receive shift register to the receive FIFO and the transfer causes the FIFO to become full, i.e., all three FIFO positions are occupied. It is reset when the CPU reads the RHR. If a character is waiting in the receive shift register because the FIFO is full, FFULL will not be reset when the CPU reads the RHR.

SRA[0] - Channel A Receiver Ready (RxRDYA)
This bit indicates that a character has been received and is waiting in the FIFO to be read by the CPU. It is set when the character is transferred from the receive shift register to the FIFO and reset when the CPU reads the RHR, if after this read there are no more characters still in the FIFO.

SRB - Channel B Status Register

The bit definitions for this register are identical to the bit definitions for SRA, except that all status applies to the channel B receiver and transmitter and the corresponding inputs and outputs.

OPCR - Output Port Configuration Register

OPCR[7] - OP7 Output Select
This bit programs the OP7 output to provide one of the following:

-The complement of OPR[7]

-The channel B transmitter interrupt output, which is the complement of TxRDYB. When in this mode OP7 acts as an open collector

output. Note that this output is not masked by the contents of the IMR.

OPCR[6] - OP6 Output Select
This bit programs the OP6 output to provide one of the following:

-The complement of OPR[6]

-The channel A transmitter interrupt output, which is the complement of TxRDYA. When in this mode OP6 acts as an open collector output. Note that this output is not masked by the contents of the IMR.

OPCR[5] - OP5 Output Select
This bit programs the OP5 output to provide one of the following:

-The complement of OPR[5]

-The channel B receiver interrupt output, which is the complement of ISR[5]. When in this mode OP5 acts as an open collector output. Note that this output is not masked by the contents of the IMR.

OPCR[4] - OP4 Output Select
This bit programs the OP4 output to provide one of the following:

-The complement of OPR[4]

-The channel A receiver interrupt output, which is the complement of ISR[1]. When in this mode OP4 acts as an open collector output. Note that this output is not masked by the contents of the IMR.

OPCR[3:2] - OP3 Output Select
This field programs the OP3 output to provide one of the following:

-The complement of OPR[3]

-The counter/timer output, in which case OP3 acts as an open collector output. In the timer mode, this output is a square wave at the programmed frequency. In the counter mode, the output remains high until terminal count is reached, at which time it goes low. The output returns to the high state when the counter is stopped by a stop counter command. Note that this output is not masked by the contents of the IMR.

-The 1X clock for the channel B transmitter, which is the clock that shifts the transmitted data. If data is not being transmitted, a free running 1X clock is output.

-The 1X clock for the channel B receiver, which is the clock that samples the received data. If data is not being received, a free running 1X clock is output.

OPCR[1:0] - OP2 Output Select
This field programs the OP2 output to provide one of the following:

-The complement of OPR[2]

-The 16X clock for the channel A transmitter. This is the clock selected by CSRA[3:0], and will be a 1X clock if CSRA[3:0] = 1111.

-The 1X clock for the channel A transmitter, which is the clock that shifts the transmitted data. If data is not being transmitted, a free running 1X clock is output.

-The 1X clock for the channel A receiver, which is the clock that samples the received data. If data is not being received, a free running 1X clock is output.

ACR - Auxiliary Control Register

ACR[7] - Baud Rate Generator Set Select
This bit selects one of two sets of baud rates to be generated by the BRG:

Set 1: 50, 110, 134.5, 200, 300, 600, 1.05K, 1.2K, 2.4K, 4.8K, 7.2K, 9.6K, and 38.4K baud.

Set 2: 75, 110, 134.5, 150, 300, 600, 1.2K, 1.8K, 2.0K, 2.4K, 4.8K, 9.6K, and 19.2K baud.

The selected set of rates is available for use by the channel A and B receivers and transmitters as described in CSRA and CSRB. Baud rate generator characteristics are given in table 3.

ACR[6:4] - Counter/Timer Mode And Clock Source Select
This field selects the operating mode of the counter/timer and its clock source as shown in table 4.

ACR[3:0] - IP3, IP2, IP1, IP0 Change Of State Interrupt Enable
This field selects which bits of the input port change register (IPCR) cause the input change bit in the interrupt status register (ISR[7]) to be set. If a bit is in the 'on' state, the setting of the corresponding bit in the IPCR will also result in the setting of ISR[7], which results in the generation of an interrupt output if IMR[7] = 1. If a bit is in the 'off' state, the setting of that bit in the IPCR has no effect on ISR[7].

IPCR - Input Port Change Register

IPCR[7:4] - IP3, IP2, IP1, IP0 Change Of State
These bits are set when a change of state, as defined in the input port section of this data sheet, occurs at the respective input pins. They are cleared when the IPCR is read by the CPU. A read of the IPCR also clears ISR[7], the input change bit in the interrupt status register.

The setting of these bits can be programmed to generate an interrupt to the CPU.

IPCR[3:0] - IP3, IP2, IP1, IP0 Current State
These bits provide the current state of the respective inputs. The information is unlatched and reflects the state of the input pins at the time the IPCR is read.

ISR - Interrupt Status Register

This register provides the status of all potential interrupt sources. The contents of this register are masked by the interrupt mask register (IMR). If a bit in the ISR is a '1' and the corresponding bit in the IMR is also a '1', the INTRN output will be asserted. If the corresponding bit in the IMR is a zero, the state of the bit in the ISR has no effect on the INTRN output. Note that the IMR does not mask the reading of the ISR - the true status will be provided regardless of the contents of the IMR. The contents of this register are initialized to 00_{16} when the DUART is reset.

ISR[7] - Input Port Change Status
This bit is a '1' when a change of state has occurred at the IP0, IP1, IP2, or IP3 inputs and that event has been selected to cause an interrupt by the programming of ACR[3:0]. The bit is cleared when the CPU reads the IPCR.

ISR[6] - Channel B Change In Break
This bit, when set, indicates that the channel B receiver has detected the beginning or the end of a received break. It is reset when the CPU issues a channel B 'reset break change interrupt' command.

ISR[5] - Channel B Receiver Ready Or FIFO Full
The function of this bit is programmed by MR1B[6]. If programmed as receiver ready, it indicates that a character has been received in channel B and is waiting in the FIFO to be read by the CPU. It is set when the character is transferred from the receive shift register to the FIFO and reset when the CPU reads the RHR. If after this read there are more characters still in the FIFO the bit will be set again after the FIFO is 'popped'. If programmed as FIFO full, it is set when a character is transferred from the receive holding register to the receive FIFO and the transfer causes the channel B FIFO to become full, i.e., all three FIFO positions are occupied. It is reset when the CPU reads the RHR. If a character is waiting in the receive shift register because the FIFO is full, the bit will be set again when the waiting character is loaded into the FIFO.

ISR[4] - Channel B Transmitter Ready
This bit is a duplicate of TxRDYB (SRB[2]).

ISR[3] - Counter Ready
In the counter mode, this bit is set when the counter reaches terminal count and is reset when the counter is stopped by a stop counter command.

Table 3. BAUD RATE GENERATOR CHARACTERISTICS CRYSTAL OR CLOCK = 3.6864MHz

NOMINAL RATE (BAUD)	ACTUAL 16X CLOCK (KHz)	ERROR (PERCENT)
50	0.8	0
75	1.2	0
110	1.759	-0.069
134.5	2.153	0.059
150	2.4	0
200	3.2	0
300	4.8	0
600	9.6	0
1050	16.756	-0.260
1200	19.2	0
1800	28.8	0
2000	32.056	0.175
2400	38.4	0
4800	76.8	0
7200	115.2	0
9600	153.6	0
19.2K	307.2	0
38.4K	614.4	0

NOTE:
Duty cycle of 16X clock is 50% ±1%

Table 4. ACR 6:4 FIELD DEFINITION

ACR 6:4	MODE	CLOCK SOURCE
0 0 0	Counter	External (IP2)
0 0 1	Counter	TXCA - 1X clock of channel A transmitter
0 1 0	Counter	TXCB - 1X clock of channel B transmitter
0 1 1	Counter	Crystal or external clock (X1/CLK) divided by 16
1 0 0	Timer	External (IP2)
1 0 1	Timer	External (IP2) divided by 16
1 1 0	Timer	Crystal or external clock (X1/CLK)
1 1 1	Timer	Crystal or external clock (X1/CLK) divided by 16

In the timer mode, this bit is set once each cycle of the generated square wave (every other time that the counter/timer reaches zero count). The bit is reset by a stop counter command. The command, however, does not stop the counter/timer.

ISR[2] - Channel A Change In Break
This bit, when set, indicates that the channel A receiver has detected the beginning or the end of a received break. It is reset when the CPU issues a channel A 'reset break change interrupt' command.

ISR[1] - Channel A Receiver Ready Or FIFO Full
The function of this bit is programmed by MR1A[6]. If programmed as receiver ready, it indicates that a character has been received in channel A and is waiting in the FIFO to be read by the CPU. It is set when the character is transferred from the receive shift register to the FIFO and reset when the CPU reads the RHR. If after this read there are more characters still in the FIFO the bit will be set again after the FIFO is 'popped'. If programmed as FIFO full, it is set when a character is transferred from the receive holding register to the receive FIFO and the transfer causes the channel A FIFO to become full, i.e., all three FIFO positions are occupied. It is reset when the CPU reads the RHR. If a character is waiting in the receive shift register because the FIFO is full, the bit will be set again when the waiting character is loaded into the FIFO.

ISR[0] - Channel A Transmitter Ready
This bit is a duplicate of TxRDYA (SRA[2]).

IMR - Interrupt Mask Register

The programming of this register selects which bits in the ISR cause an interrupt output. If a bit in the ISR is a '1' and the corresponding bit in the IMR is also a '1', the INTRN output will be asserted. If the corresponding bit in the IMR is a zero, the state of the bit in the ISR has no effect on the INTRN output. Note that the IMR does not mask the programmable interrupt outputs OP3-OP7 or the reading of the ISR.

CTUR And CTLR - Counter/ Timer Registers

The CTUR and CTLR hold the eight MSBs and eight LSBs respectively of the value to be used by the counter/timer in either the counter or timer modes of operation. The minimum value which may be loaded into the CTUR/CTLR registers is 0002_{16}. Note that these registers are write-only and cannot be read by the CPU.

In the timer (programmable divider) mode, the C/T generates a square wave with a period of twice the value (in clock periods) of the CTUR and CTLR. If the value in CTUR or CTLR is changed, the current half-period will not be affected, but subsequent half periods will be. In this mode the C/T runs continuously. Receipt of a start counter command (read with A3-A0 = 1110) causes the counter to terminate the current timing cycle and to begin a new cycle using the values in CTUR and CTLR.

The counter ready status bit (ISR[3]) is set once each cycle of the square wave. The bit is reset by a stop counter command (read with A3-A0 = 1111). The command, however, does not stop the C/T. The generated square wave is output on OP3 if it is programmed to be the C/T output.

On power up and after reset, the timer/ counter runs in timer mode and can only be restarted. Because it cannot be shut off or stopped, and runs continuously in timer mode, it is recommended that at initialization, the output port (OP3) should be masked off through the OPCR[3:2] = 00 until the T/C is programmed to the desired operational state.

In the counter mode, the C/T counts down the number of pulses loaded into CTUR and CTLR by the CPU. Counting begins upon receipt of a start counter command. Upon reaching terminal count (0000_{16}), the counter ready interrupt bit (ISR[3]) is set. The counter continues counting past the terminal count until stopped by the CPU. If OP3 is programmed to be the output of the C/T, the output remains high until terminal count is reached, at which time it goes low. The output returns to the high state and ISR[3] is cleared when the counter is stopped by a stop counter command. The CPU may change the values of CTUR and CTLR at any time, but the new count becomes effective only on the next start counter command. If new values have not been loaded, the previous count values are preserved and used for the next count cycle.

In the counter mode, the current value of the upper and lower 8 bits of the counter (CTU, CTL) may be read by the CPU. It is recommended that the counter be stopped when reading to prevent potential problems which may occur if a carry from the lower 8-bits to the upper 8-bits occurs between the times that both halves of the counter are read. However, note that a subsequent start counter command will cause the counter to begin a new count cycle using the values in CTUR and CTLR.

ABSOLUTE MAXIMUM RATINGS[1]

PARAMETER	RATING	UNIT
Operating ambient temperature[2]	0 to +70	°C
Storage temperature	-65 to +150	°C
All voltages with respect to ground[3]	-0.5 to +6.0	V

DC ELECTRICAL CHARACTERISTICS T_A = 0°C to +70°C, V_{CC} = 5.0V ± 5%[4, 5, 6]

PARAMETER		TEST CONDITIONS	LIMITS			UNIT
			Min	Typ	Max	
V_{IL}	Input low voltage				0.8	V
V_{IH}	Input high voltage (except X1/CLK)		2.0			V
V_{IH}	Input high voltage (X1/CLK)		4.0			V
V_{OL}	Output low voltage	I_{OL} = 2.4mA			0.4	V
V_{OH}	Output high voltage (except o.c.outputs)	I_{OH} = -400µA	2.4			V
I_{IL}	Input leakage current	V_{IN} = 0 to V_{CC}	-10		10	µA
I_{LL}	Data bus 3-state leakage current	V_O = 0.4 to V_{CC}	-10		10	µA
I_{X1L}	X1/CLK low input current	V_{IN} = 0, X2 grounded	-4.0	-2.0	0.0	mA
		V_{IN} = 0, X2 floated[7]	-3.0	-1.5	0.0	mA
I_{X1H}	X1/CLK high input current	V_{IN} = V_{CC}, X2 grounded	-1.0	0.2	1.0	mA
		V_{IN} = V_{CC}, X2 floated[7]	0.0	3.5	10.0	mA
I_{X2L}	X2 low input current	V_{IN} = 0, X1/CLK floated	-100	-30	0.0	µA
I_{X2H}	X2 high input current	V_{IN} = V_{CC}, X1/CLK floated	0.0	+30	100	µA
I_{OC}	Open collector output leakage current	V_O = 0.4 to V_{CC}	-10		10	µA
I_{CC}	Power supply current				150	mA

NOTES:

1. Stresses above those listed under Absolute Maximum Ratings may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operation section of this specification is not implied.
2. For operating at elevated temperatures, the device must be derated based on +150°C maximum junction temperature.
3. This product includes circuitry specifically designed for the protection of its internal devices from damaging effects of excessive static charge. Nonetheless, it is suggested that conventional precautions be taken to avoid applying any voltages larger than the rated maxima.
4. Parameters are valid over specified temperature range.
5. All voltage measurements are referenced to ground (GND). For testing, all inputs except X1/CLK swing between 0.4V and 2.4V with a transition time of 20ns maximum. For X1/CLK this swing is between 0.4V and 4.4V. All time measurements are referenced at input voltages of 0.8V and 2.0V as appropriate.
6. Typical values are at +25°C, typical supply voltages, and typical processing parameters.
7. X2 is left internally floating in the 24 pin version.

AC ELECTRICAL CHARACTERISTICS T_A = 0°C to +70°C, V_{CC} = 5.0V ± 5%[4, 5, 6, 7]

PARAMETER		LIMITS			UNIT
		Min	Typ	Max	
Reset Timing (figure 1)					
t_{RES}	RESET pulse width	1.0			µs
Bus Timing (figure 2)[8]					
t_{AS}	A0-A3 set-up time to RDN, WRN low	10			ns
t_{AH}	A0-A3 hold time from RDN, WRN high	0			ns
t_{CS}	CEN set-up time to RDN, WRN low	0			ns
t_{CH}	CEN hold time from RDN, WRN high	0			ns
t_{RW}	WRN, RDN pulse width	225			ns
t_{DD}	Data valid after RDN low			175	ns
t_{DF}	Data bus floating after RDN high			100	ns
t_{DS}	Data setsup time before WRN high	100			ns
t_{DH}	Data hold time after WRN high	20			ns
t_{RWD}	High time between READs and/or WRITEs[9,10]	200			ns

AC ELECTRICAL CHARACTERISTICS T_A = 0°C to +70°C, V_{CC} = 5.0 V ± 5%[4, 5, 6, 7]

PARAMETER		LIMITS			UNIT
		Min	Typ	Max	
Port Timing (figure 3)[8]					
t_{PS}	Port input set-up time before RDN low	0			ns
t_{PH}	Port input hold time after RDN high	0			ns
t_{PD}	Port output valid after WRN high			400	ns
Interrupt Timing (figure 4)					
t_{IR}	INTRN (or OP3-OP7 when used as interrupts) negated from:				
	Read RHR (RXRDY/FFULL interrupt)			300	ns
	Write THR (TXRDY interrupt)			300	ns
	Reset command (delta break interrupt)			300	ns
	Stop C/T command (counter interrupt)			300	ns
	Read IPCR (input port change interrupt)			300	ns
	Write IMR (clear of interrupt mask bit)			300	ns
Clock Timing (figure 5)					
t_{CLK}	X1/CLK high or low time	100			ns
f_{CLK}	X1/CLK frequency	2.0	3.6864	4.0	MHz
t_{CTC}	CTCLK (IP2) high or low time	100			ns
f_{CTC}	CTCLK (IP2) frequency	0		4.0	MHz
t_{RX}	RxC high or low time	220			ns
f_{RX}	RxC frequency (16X)	0		2.0	MHz
	(1X)	0		1.0	MHz
t_{TX}	TxC high or low time	220			ns
f_{TX}	TxC frequency (16X)	0		2.0	MHz
	(1X)	0		1.0	MHz
Transmitter Timing (figure 6)					
t_{TXD}	TxD output delay from TxC low			350	ns
t_{TCS}	Output delay from TxC low to TxD data output	0		150	ns
Receiver Timing (figure 7)					
t_{RXS}	RxD data set-up time to RXC high	240			ns
t_{RXH}	RxD data hold time from RXC high	200			ns

NOTES:

4. Parameters are valid over specified temperature range.
5. All voltage measurements are referenced to ground (GND). For testing, all inputs except X1/CLK swing between 0.4V and 2.4V with a transition time of 20ns maximum. For X1/CLK this swing is between 0.4V and 4.4V. All time measurements are referenced at input voltages of 0.8V and 2.0V as appropriate.
6. Typical values are at +25°C, typical supply voltages, and typical processing parameters.
7. Test condition for outputs: C_L = 150pF, except interrupt outputs. Test condition for interrupt outputs: C_L = 50pF, R_L = 2.7K ohm to V_{CC}.
8. Timing is illustrated and referenced to the WRN and RDN inputs. The device may also be operated with CEN as the 'strobing' input. In this case, all timing specifications apply referenced to the falling and rising edges of CEN. CEN and RDN (also CEN and WRN) are AND'ed internally. As a consequence, the signal asserted last initiates the cycle and the signal negated first terminates the cycle.
9. If CEN is used as the 'strobing' input, the parameter defines the minimum high times between one CEN and the next. The RDN signal must be negated for t_{RWD} to guarantee that any status register changes are valid.
10. Consecutive write operations to the same command register require at least three edges of the X1 clock between writes.

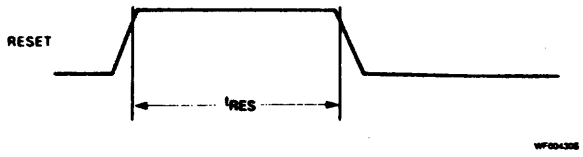


Figure 1. Reset Timing

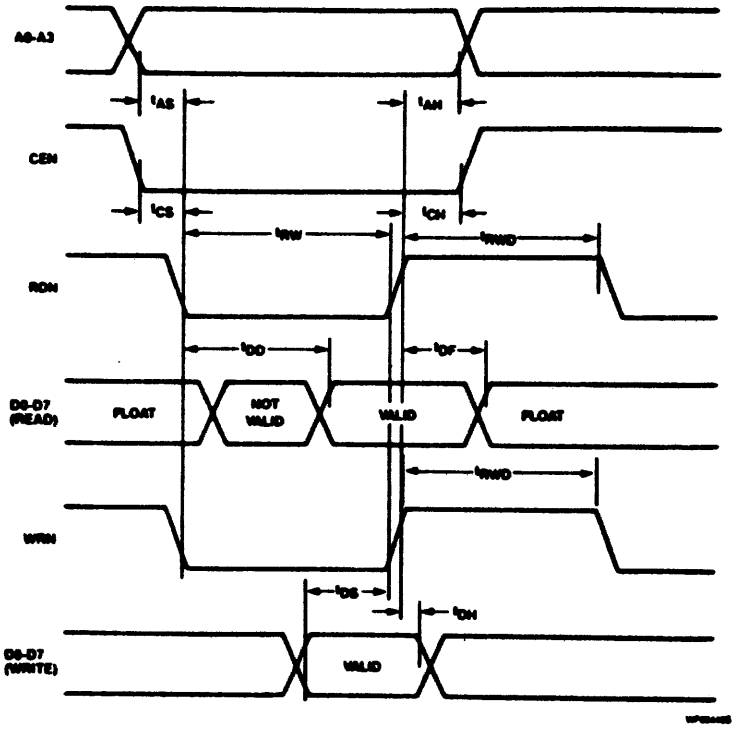


Figure 2. Bus Timing

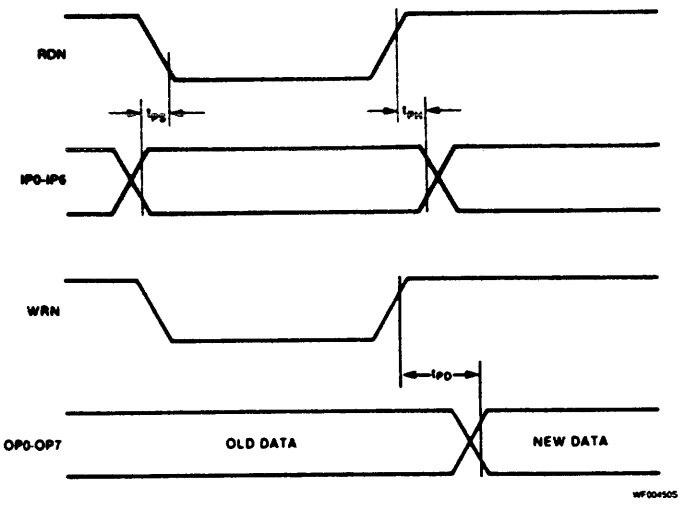
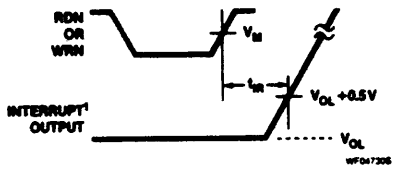


Figure 3. Port Timing



NOTES:
1. INTRN or OP3 - OP7 when used as interrupt outputs.
2. The test for open drain outputs is intended to guarantee switching of the output transistor. Measurement of this response is referenced from the midpoint of the switching signal, V_M, to a point 0.5 volts above V_{OL}. This point represents noise margin that assures true switching has occurred. Beyond this level, the effects of external circuitry and test environment are pronounced and can greatly affect the resultant measurement.

Figure 4. Interrupt Timing

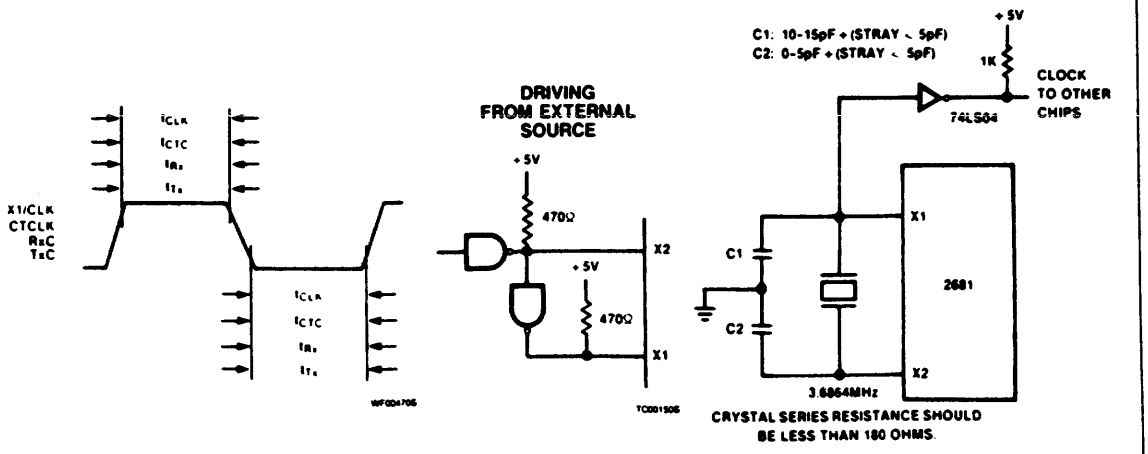


Figure 5. Clock Timing

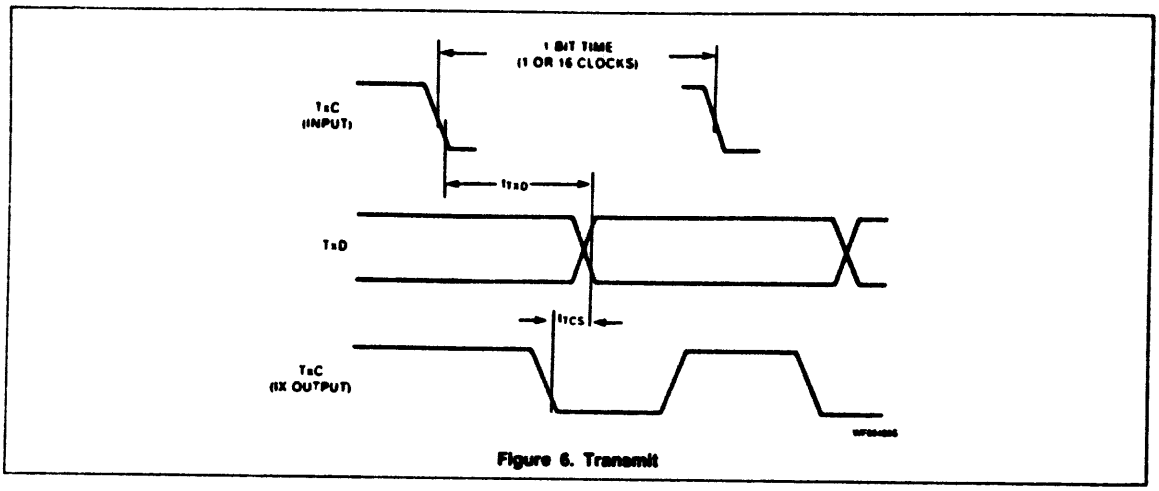


Figure 6. Transmit

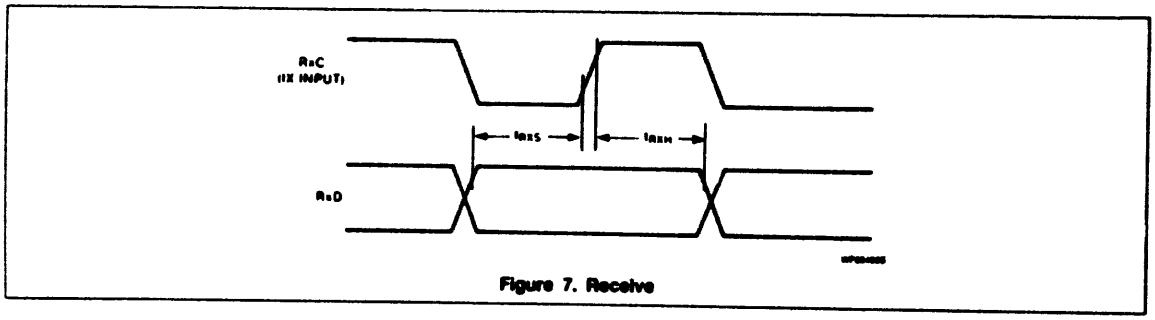


Figure 7. Receive

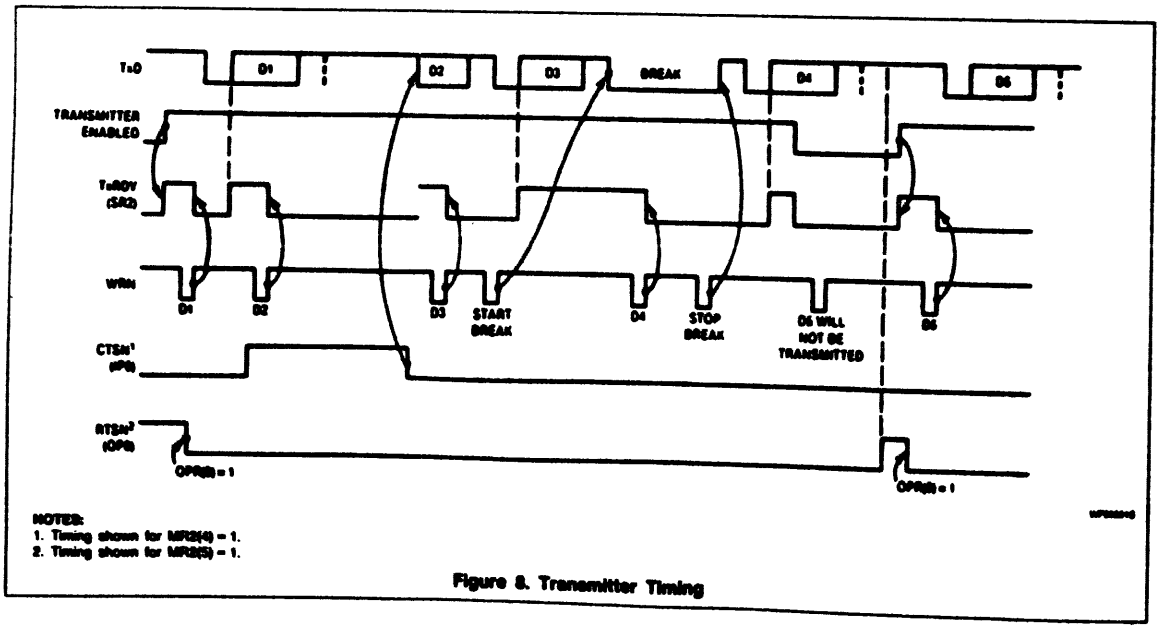
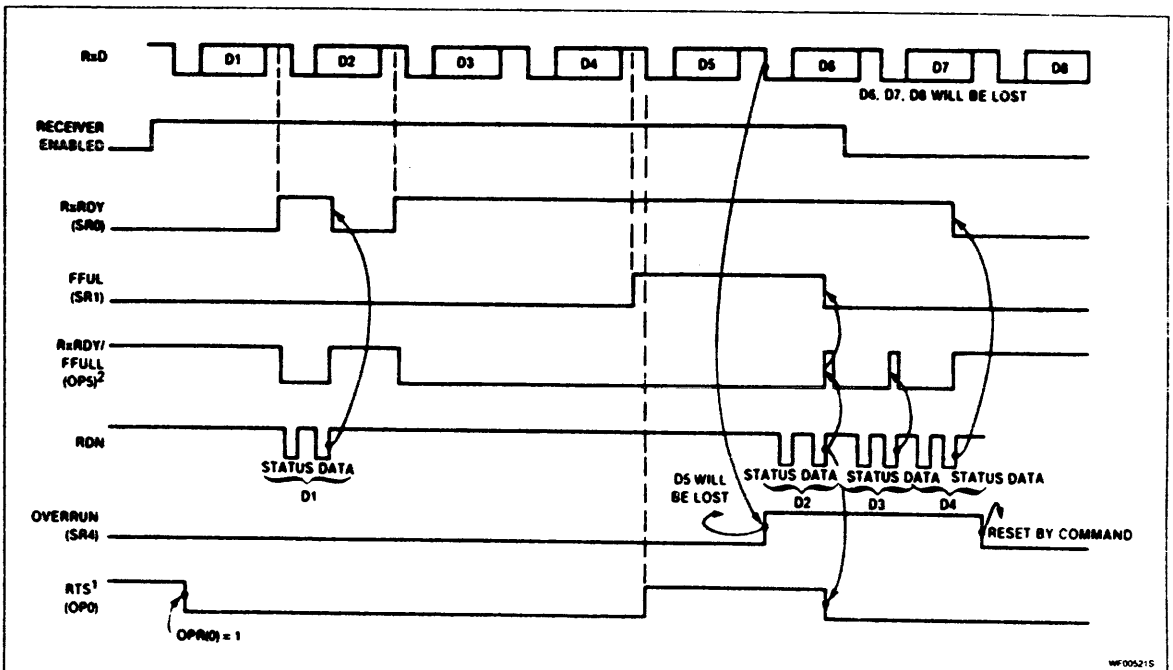


Figure 8. Transmitter Timing



NOTES:
1. Timing shown for MR1(7) = 1.
2. Shown for OPCR(4) = 1 and MR(6) = 0.

Figure 9. Receiver Timing

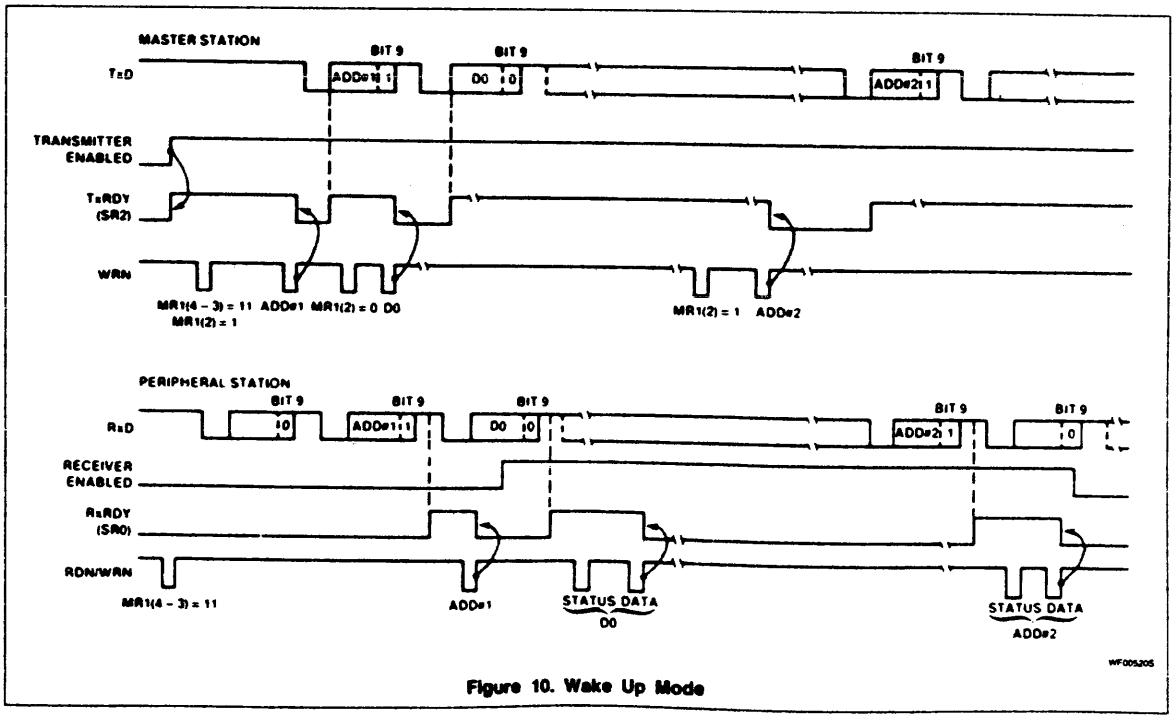


Figure 10. Wake Up Mode

Index

T

Available MASSCOMP Manuals

Available MASSCOMP Manuals		
Title	Revision	Order Number:
AD12F Hardware Manual	A	M-AD12F-HM
AD12FA Hardware Manual	A	M-AD12FA-HM
ALIS, Volume I	A	M-ALIS-01
ALIS, Volume II	A	M-ALIS-02
Array Processor Programmer's Manual	B	M-AP501-LM
The C Programming Language	A	M-SP28-000
DA04H Hardware Manual	A	M-DA04H-HM
DA08F Hardware Manual	A	M-DA08F-HM
DA/CP Assembly Language Reference Manual	A	M-SP55-MS
DA/CP Debugger Reference Manual	A	M-SP55-MS
DA/CP System Programming Manual	A	M-SP55-MS
Data Acquisition Application Programming Manual	E	M-SP50-AP
Data Presentation Application Programming Manual	B	M-SP45-SP
DI-3000 User's Guide and Quick Reference Guide	5	M-SP80-00
Diagnostic Monitor User's Guide	A	M-DIAG-UG
EF12M Hardware Manual	B	M-EF12M-HM
MASSCOMP Ethernet Manager's Guide	C	M-SP70-MS
MASSCOMP Ethernet User's Guide	B	M-SP70-MS
Floating Point Programming Manual	B	M-FP501-SP
FORTRAN Compiler User's Manual	A	M-SP24-002
FORTRAN Language Reference Manual	A	M-SP24-001
The Franz LISP Reference Manual	A	M-SP26-RM
GK-2000 User's Guide	B	M-SP88-00
Graphics Application Programming Manual	H	M-SP40-AP
Graphics System Programming Manual	A	M-SP40-PM
Guide to Writing a UNIX Device Driver	A	M-DRIVER-G
Guide to MASSCOMP Documentation	B	M-GTD-00
Introducing the UNIX System	—	M-RTU-001
LISPcraft	—	M-SP26-00
MC500 CPU Reference Manual	A	M-MC500-RM
MC5300/5400 Installation Guide	A	M-IG-534
MC5300/5400 System Management Guide	A	M-SMG-534
MC5500 System Management Guide	F	M-SMG-013
MC5600/5700 Installation Guide	A	M-IG-567
MC5600/5700 System Management Guide	A	M-SMG-567
MC5600/5700 System Reference Manual	A	M-SRM-567
MC-Windows Programming Manual	A	M-SP43-PM
Oregon Software Pascal-2.1 User's Manual	2	M-SP22-001
PI16F Hardware Manual	B	M-PI16F-HM
Programs for Digital Processing	—	M-SP60-001

Available MASSCOMP Manuals		
Title	Revision	Order Number:
Quick Choice Menu Reference Manual	A	M-SP30-MS
Quick Choice Primer	A	M-SP30-MS
Quick Choice User's Manual	B	M-SP30-MS
RTU Programming Manual	A	M-RTU-PM
UNIX Programmer's Manual, Vol. 1A	H	M-RTU-011
UNIX Programmer's Manual, Vol. 1B	H	M-RTU-011
UNIX Programmer's Manual, Vol. 2A	B	M-RTU-012
UNIX Programmer's Manual, Vol. 2B	—	M-RTU-012
The UNIX Programming Environment	—	M-RTU-002
UNIX Quick Reference Guide	A	M-RTU-QR
Using the EF12M	A	M-EF12M-SM
Using the SC-16F	B	M-SC16F-PRE
Using the SH16F	A	M-SH16F-PRE
Using The TC16F	A	M-SC16F-PRE
UUCP Management Guide	A	M-UUCP-MG
Ved Text Editor Reference Manual	B	M-VED-015
X.25 Manager's Guide	A	M-SCP-MG
X.25 User's Guide	A	M-SCP-UG

MC5600/5700 System Reference Manual
Revision A July 30, 1986

Part Number: 075-04020-00-0
Order Number: M-SRM-567

Reader's Documentation Comments

Please help us improve our manuals by filling out and mailing this form.

If you need a written reply and are under a maintenance contract, please submit your comments on an SQR form. All software comments must be on an SQR form.

What is your application for your MASSCOMP System?

_____

_____

_____

_____

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

Did you find errors in this manual? If you did, please specify the error and the page number.

_____

_____

_____

_____

_____

Please circle the appropriate category in each column.

LANGUAGE	EXPERIENCE
C	Student/novice
F77	2+ years UNIX experience
Other (Which?)	5+ years UNIX experience

Name _____ Phone _____ Date ____/____/____

Organization _____

Street _____

City _____ State _____ Zip code or Country _____

Fold Here

Do Not Tear – Fold Here and Staple

Concurrent Computer Corporation

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO. 12 WESTFORD, MASS

POSTAGE WILL BE PAID BY ADDRESSEE

CONCURRENT COMPUTER CORPORATION
1 TECHNOLOGY WAY – BOX 563
WESTFORD, MA 01886
ATTENTION: TECHNICAL PUBLICATIONS

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES