



NuMachine Technical Summary

LAMBDA

SDU Operating System
SDU General Description

Mouse Manual
LMI Printer Software Manual

VR-Series Monitor
Z29 Monitor

Ethernet Multibus

LMI DOCUMENTATION SYSTEM MAP

** indicates location of tab divider

Old



BASICS:

- **LMI Lambda Technical Summary
- **LMI Lambda Field Service Manual
- **NuMachine Installation and User Manual
- **Introduction to the Lambda
- **Programming on the Lambda



RELEASE NOTES:

- **System 94 Notes
- **System 98 Notes
- **Common LISP Notes



LISP 1: The LISP Machine Manual, Part 1

- **Introduction
 - Primitive Object Types
 - Evaluation
 - Flow of Control
 - Manipulating List Structure
- **Symbols
 - Numbers
 - Arrays
 - Strings
- **Functions
 - Closures
 - Stack Groups
 - Locatives
 - Subprimitives
 - Areas
- **The Compiler
 - Macros
 - The LOOP Iteration Macro
- **Defstruct



LISP 2: The LISP Machine Manual, Part 2

- **Objects, Message Passing, and Flavors
- **The I/O System
 - Naming of Files
 - The Chaosnet
- **Packages
 - Maintaining Large Systems
 - Processes
 - Errors and Debugging
- **How to Read Assembly Language
 - Querying the User
 - Initializations
 - Dates and Times
 - Miscellaneous Useful Functions
- **Indices



LISP 3:

- **Introduction to the Window System
- **The Window System Manual
- **ZMAIL
- **Prolog
- **InterLISP



EDITORS:

- **ZMACS Introductory Manual
- **ZMACS Reference Manual
- **Mince
- **Scribble



UNIX 1:

- **NuMachine Release and Update Information
- **NuMachine Operating System
- **UNIX Programmer's Manual, V. 1: Section 1 Sections 2-8
- **Fortran Installation Memo



UNIX 2: UNIX Programmer's Manual, Vol. 2

- **The UNIX Time-sharing System
 - An Introduction to the UNIX Shell
 - Typing Documents on the UNIX System
 - A Guide to Preparing Documents with -ms
 - Tbl--A Program to Format Tables
 - NROFF/TROFF User's Manual
 - A TROFF Tutorial
- **The C Programming Language Reference Manual
 - Recent Changes to C
 - Lint, A C Program Checker
 - Make--A Program for Maintaining Computer Programs
- **UNIX Programming--Second Edition
 - A Tutorial Introduction to ADB
 - Yacc: Yet Another Compiler-Compiler
 - Lex--A Lexical Analyzer Generator
- **A Portable Fortran 77 Compiler
 - RATFOR--A Preprocessor for a Rational Fortran
 - The M4 Macro Processor
 - SED--A Non-Interactive Text Editor
 - Awk--A Pattern Scanning and Processing Language (2d. ed.)
 - DC--An Interactive Desk Calculator
 - BC--An Arbitrary Precision Desk-Calculator Language
- An Introduction to Display Editing with Vi
- **The UNIX I/O System
 - On the Security of UNIX
 - Password Security: A Case History



HARDWARE 1:

- **NuMachine Technical Summary
- **SDU Operating System
 - SDU General Description
- **Mouse Manual
 - LMI Printer Software Manual
- **VR-Series Monitor
 - Z29 Monitor
- **Ethernet Multibus



HARDWARE 2:

- **SMD 2181 Controller Board
- **Fujitsu Disk Drive (Micro or Mini)
- **Cipher Tape Drive 1/4"



HARDWARE 3:

- **Cipher Tape Drive 1/2"
- **Tapemaster Product Spec
- **Tapemaster Application Note



OPTIONS:

- ** (varies according to options purchased)

Nu Machine Technical Summary

TI-2242820-0001

Distributed by LMI 6033 W. Century Blvd. Los Angeles CA 90045
USA

Information furnished in this document is believed to be accurate and reliable. However, no responsibility is assumed by Texas Instruments for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Texas Instruments. Texas Instruments reserves the right to change product specifications at any time.

UNIX[™] is a trademark of American Telephone & Telegraph.

Multibus[™] is a trademark of Intel Corp.

Ethernet[™] is a trademark of Xerox Corp.

Copyright © 1982 Texas Instruments All rights reserved.

NU MACHINE TECHNICAL SUMMARY

Contents

1	INTRODUCTION	1
	1.1 System Origin	1
	1.2 The System	1
	1.3 Applications	4
2	THE NUBUS	7
	2.1 Overview	7
	2.2 NuBus Signals	7
	2.3 Data Transfer Operations	8
	2.4 Multiprocessor Support	10
	2.4.1 Arbitration	
	2.4.2 Interrupts	
3	68010 PROCESSOR BOARD	11
	3.1 Virtual Address Space	11
	3.2 Main Functions of the CPU Board	12
	3.2.1 68010 Microprocessor	
	3.2.2 Cache Operation	
	3.2.3 Cache Control	
	3.2.4 Cache ID	
	3.2.5 Address Translation	
	3.2.6 Interrupt Mechanism	
	3.2.7 CPU Control	
4	SYSTEM DIAGNOSTIC UNIT	21
5	MULTIBUS INTERFACE	25
	5.1 Conversion	25
	5.1.1 NuBus-to-Multibus Conversion	
	5.1.2 Multibus-to-NuBus Conversion	
	5.2 Interrupt Mapping	28
	5.3 Lockup Prevention	28
6	VIDEO DISPLAY SUBSYSTEM	29
	6.1 Main Functions of the Video Display Subsystem	29
	6.2 Serial Ports	31
7	MAIN MEMORY	33
	7.1 Main Functions of the Memory Board	33
8	SOFTWARE	35
	8.1 SDU Monitor and Diagnostics	35
	8.1.1 SDU Monitor	
	8.1.2 Diagnostics	
	8.2 Nu Machine Operating System	37
	8.2.1 Overview	
	8.2.2 Nu Machine Operating System Kernel	
	8.2.3 Utilities	
	8.3 Future Plans	41
9	NU MACHINE PACKAGING	43
	9.1 Monitor, Keyboard and Mouse	43
	9.2 Office Module	43
	9.3 I/O Interconnection	44
	9.4 Rack Module	44
	9.5 Summary	45

List of Figures

Figure 1-1.	Comparison of Traditional and Nu Machine Architectures	2
Figure 2-1.	Layout of Words, Halfwords, and Bytes	9
Figure 3-1.	Virtual Address Space Concept	12
Figure 3-2.	Block Diagram of CPU Board	13
Figure 3-3.	Data Cache Entry (Parity, Tag Field, and Data)	14
Figure 3-4.	Cache Control	15
Figure 3-5.	Cache Status	15
Figure 3-6.	Virtual-to-Physical Address Translation	17
Figure 3-7.	Representation of Physical Address	18
Figure 3-8.	PTE Description	18
Figure 3-9.	Definitions of Access Bits	19
Figure 4-1.	System Diagnostic Unit Block Diagram	22
Figure 5-1.	NuBus-to-Multibus Memory Space Conversion	25
Figure 5-2.	NuBus-to-Multibus I/O Space Conversion	26
Figure 5-3.	Multibus-to-NuBus Conversion	27
Figure 5-4.	Multibus-to-NuBus Address Conversion	28
Figure 6-1.	Block Diagram — B/W Video Display Subsystem	30
Figure 7-1.	Memory Board Block Diagram	33
Figure 9-1.	Office Module Cabinet and Contents	43
Figure 9-2.	Rack Module and Component Layout	45

1 INTRODUCTION

1.1 System Origins

The Nu Machine is based on the advanced NuBus technology developed at the Laboratory for Computer Science at the Massachusetts Institute of Technology.

The NuBus and Nu Machine architectures were developed to solve computing needs which were not served by commercially available equipment. The goal was to create a workstation-oriented computer with the following features:

- Processor independent
- State-of-the-art backplane bus
- Network and Graphics oriented

In April 1981, after the initial breadboard Nu Machines were constructed at M.I.T. and the UNIX™ operating system was ported to it, the design was licensed to Western Digital Corporation to be re-engineered and marketed as a commercial product. By late 1982, Western Digital had completed the redesign but decided not to enter the workstation business. Texas Instruments acquired the Nu Machine/NuBus technology including the original development group located in Irvine, California.

1.2 The System

Although always conceived of as a workstation, the Nu Machine is designed to be a general purpose, flexible, high-performance computer. The Nu Machine is a communications centered architecture in which the NuBus is a wide and efficiently controlled data freeway, providing exchange of information among processors, memory, mass storage, and remote computers (via networks). This approach is a departure from other computers. Figure 1-1 contrasts the NuBus architecture with a traditional supermini.

**Nu Machine Technical Summary
Texas Instruments**

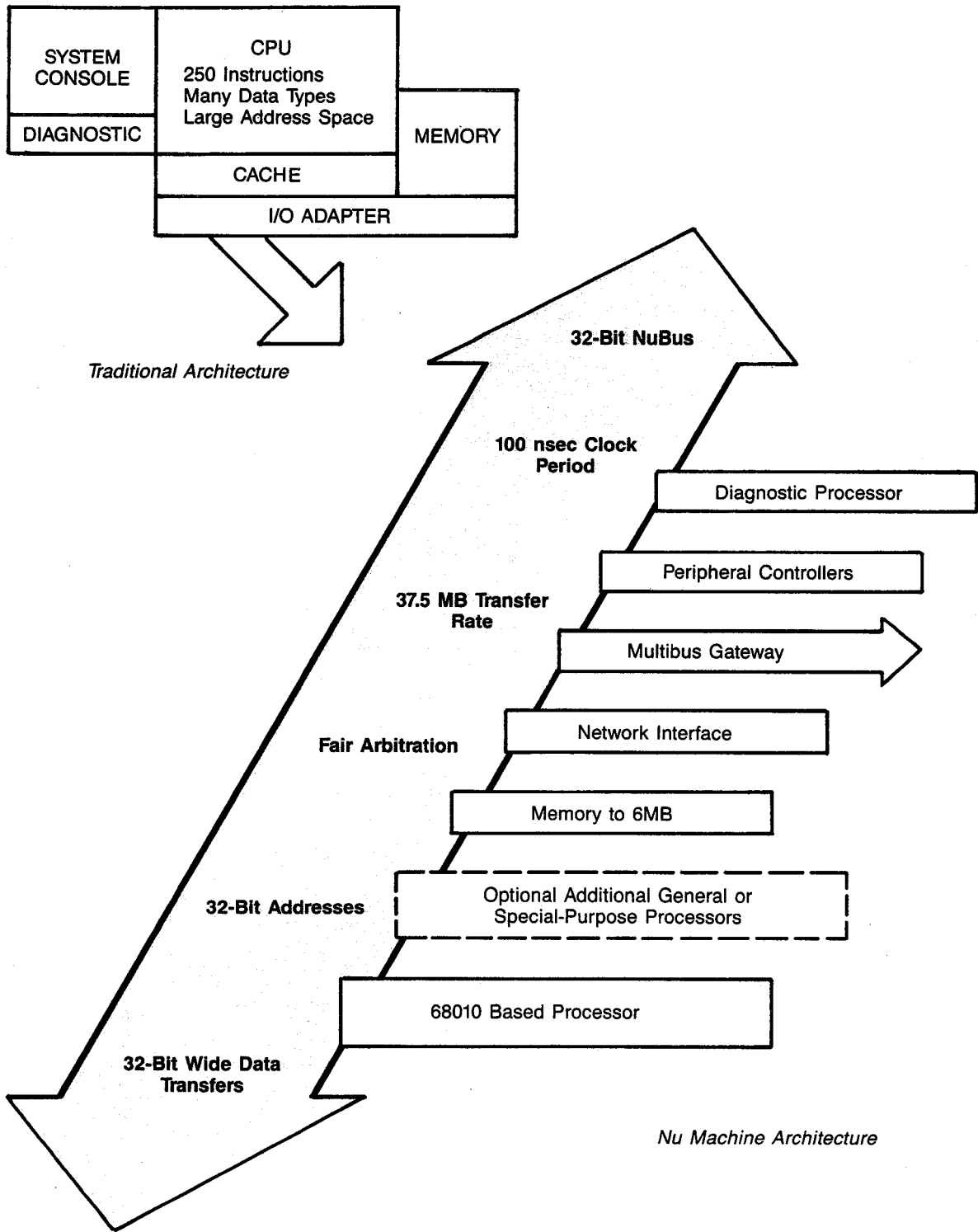


Figure 1-1. Comparison of Traditional and Nu Machine Architectures.

Nu Machine Technical Summary Texas Instruments

The traditional computer system architecture is processor centered, with the major subsystems arrayed around the central processing unit (CPU) and isolated from the outside world by an I/O adapter. The traditional architecture is ill-suited to multiprocessor configurations and special purpose processor configurations. The NuBus, on the other hand, provides a framework for systems that require multiple general purpose processors, graphics processors, signal processors, file management processors, and/or communications processors.

The Nu Machine incorporates these technologies:

NuBus

The NuBus provides a maximum transfer rate of 37.5 Mbytes per second, a bandwidth required for high-end applications involving multiple processors and future high-performance peripherals. OEMs may attach processors and controllers of their own design directly to the NuBus. The system implementor can have each processor running a different operating system, or have multiple processors running one operating system.

The NuBus itself provides many of the key Nu Machine features. Reliability is enhanced by the use of DIN connectors rather than card edge connectors. The bus supports autoconfiguration and eliminates the need for "DIP" switches and jumpers. The features that support multiprocessing are high bandwidth, dynamically redirectable interrupts and "fair" bus arbitration. TI is working with standards organizations, proposing that NuBus technology be adopted as an industry standard.

68010-based CPU with Cache Memory

The central processor of the Nu Machine has a 4K byte 45 nanosecond cache memory and a sophisticated memory management system implemented in hardware. These features allow the 68010 to form the heart of a very high performance card. Because the NuBus supports 32 bits of data and address, it is well suited to take advantage of the forthcoming 68020 processor. The processor independence of the bus allows future processors to be built around other standard microprocessor families or special purpose instruction sets as justified by market needs.

System Diagnostic Unit (SDU)

This multifunction board provides many features previously available only on much larger computers. The SDU is independent from the rest of the computer in that it contains an 8088 processor, RAM, ROM, and serial ports. It can be used under local or remote access to execute bus, board, and system diagnostics.

By reading ID ROMS on individual cards, the SDU performs an autoconfiguration, allowing the system to use a new/replacement card immediately, and then bootstraps the system software. Under program control the SDU can margin both the system clock and +5 volt power supply to test system robustness and aid in fault isolation.

An independent function of the SDU board is the conversion between the NuBus and the Multibus™. This conversion allows masters on either bus to address slaves on the other bus, and is hardware implemented.

Nu Machine Technical Summary

Texas Instruments

Multibus Subsystem

A large selection of third party I/O controllers are accessible through use of the NuBus-to-Multibus converter. The Multibus Subsystem operates independently of the NuBus, except during cycles which use them both.

High Resolution Raster Graphics

The 800 × 1024 60Hz *noninterlaced* display produces the precise drawings and mixed, multifont text and graphics needed by engineering workstation users.

Networking Capabilities

The Nu Machine Ethernet™ networking capabilities provide dedicated computing resources under the direct control of the user, while also facilitating inter-user communication and sharing of resources.

Nu Machine Operating System

The operating system ported from Bell Laboratories UNIX, is very well suited to the technical work station software needs. Adaptations and enhancements have been made to support the high resolution graphics and other Nu Machine hardware features.

Attractive, Functional Packaging.

There are two primary configurations of Nu Machine. The smaller "office" unit is designed to fit under a work surface and be acoustically quiet, and the 19" rack model, which supports larger peripherals, is more suited to computer room operations.

The office unit has a 12 slot card cage and supports a ¼" cartridge tape drive and an 8" 84 megabyte disk. The rack unit has a 21 slot card cage and supports a ½" streaming tape unit and a 474 megabyte disk. In both models, all cables attach to the rear of the card cage rather than to printed circuit cards, thus simplifying maintenance. Careful design of the motherboard and cabling reduces RF radiation at the source. In the office system, peripherals and power supplies are mounted on easy-to-remove plates to simplify maintenance and to easily accommodate the mounting requirements of many different makes of peripherals.

The optional 15" high resolution graphics portrait display used with both Nu Machine configurations is designed to have a very small footprint on the user's work surface. The monitor provides connections for the low profile keyboard and mouse. The monitor/keyboard/mouse assembly can be up to 200 feet from the Nu Machine itself.

1.3 Applications

The long life expectancy of the NuBus and the high degree of modularity and processor independence make the Nu Machine boards and chassis a fundamental system for the OEM or systems integrator. These features make the Nu Machine particularly attractive to OEMs who plan to create special purpose processors, especially if high bandwidth and

**Nu Machine Technical Summary
Texas Instruments**

multiprocessing are required. For example, a current Nu Machine customer is delivering Nu Machines with a proprietary Lisp processor for Artificial Intelligence applications. The resulting combination of the Lisp and 68010 processor allows their users to do development in Lisp and still have access to the growing base of UNIX software.

Other possible applications for the Nu Machine include VLSI circuit design, printed circuit board design, mechanical design, mapping and cartography, civil engineering, architecture, and Computer-Aided Engineering (CAE).

An advantage of the Nu Machine is its ability to support teams. Many of the above applications involve large databases which must be shared by team members. The volume of data may be cumbersome when moved across a local network. With a NuBus, several processors can directly access the same disk and memory, creating what in essence is a 300M bit-per-second very "local" network. For example, several users each with a dedicated processor and operating system could have rapid access to the complete project database within the limits of their access privileges.

The common denominator of all these applications is the requirement for networking in combination with high-performance processing and graphics. The NuBus makes the Nu Machine a uniquely flexible and powerful tool in the hands of a skilled system architect.

Nu Machine Technical Summary
Texas Instruments



2 THE NUBUS

2.1 Overview

The NuBus interface is the flexible bus structure which allows the various system components, including the 68010 CPU board, memory board, video interfaces, network interfaces, and peripheral controllers, to interact with each other. It supports direct addressability of more than four gigabytes of memory through 32-bit byte addressing, and data transactions of 8-, 16-, and 32-bits. Block transfers of multiple words are supported to achieve maximum bandwidth.

The bus structure is built on a master-slave concept. For each transaction, a device takes control of the NuBus interface, thus becoming a "master," and addresses another unit to be a "slave" for that transaction. The slave device, on decoding its address, acts on the command provided by the master. A simple handshake protocol between the master and slave allows modules of different speeds to use the NuBus interface.

The flexibility of the NuBus plus its high bandwidth allow multiple master modules to be connected for multiprocessing configurations. Fair arbitration among the bus masters gives each processor an essentially equal share of the bus bandwidth.

Any module on the NuBus can interrupt a processor module by writing into an area of address space that is monitored by that processor. This "event" mechanism is an important aspect of multiprocessor support. Interrupt lines are eliminated and interrupts may be dynamically reassigned to different processors.

In summary, important NuBus features are:

- All bus signals, power and grounds are contained within a single 96-pin DIN connector;
- 32-bits of data multiplexed with 32-bits of address;
- Distributed bus arbitration that implements fair bus bandwidth sharing. Arbitration may take place simultaneously with data transfer operations;
- Simple handshake protocol synchronized to clock cycles;
- "Memory-mapped" interrupt scheme that supports multiprocessors;
- Increased bus bandwidth through block transfers;
- Board addressing by slot position rather than jumpers or switches; and
- Address and data transfer integrity may be protected by parity logic.

The NuBus supports the unified bus interconnection of up to 16 system modules. The motherboard presents each board location with a unique identification number. Any system module can occupy any board location; thus, jumpers, or switches on individual cards and special backplane wiring are not needed to define address space.

2.2 NuBus Signals

The NuBus signals are grouped into five classes based on the function performed. The five classes are:

Nu Machine Technical Summary
Texas Instruments

Card Slot Identification

4 signals; assigns the physical location to each module.

Control

6 signals; performs all control functions. These functions are as follows:

- RESET/ (where "/" indicates a negative true signal);
- CLK/ (Clock);
- START/ — signals the beginning of data transfer;
- ACK/ (Transfer Acknowledge) — indicates the end of data transfer;
- TM0/and TM1/ (Transfer Mode 0 and 1) — are encoded by the current bus master to indicate type of transfer.

Address/Data

32 signals; carries 32-bit address at beginning of cycle and 32 bits of data within remainder of cycle.

Bus

5 signals; regulates bus arbitration.

Parity

2 signals

- SP/ (System Parity) — transmits parity information between cards implementing NuBus parity checking.
- SPV/ (System Parity Valid) — indicates whether system parity is valid or not for that transaction.

The NuBus also includes conductors that carry power and ground signals. Voltages available are +5, -5, +12, and -12. Critical signals are isolated and surrounded by power and ground lines to minimize crosstalk.

2.3 Data Transfer Operations

Data transfer on the NuBus is accomplished using a synchronous master/slave protocol. Bus transactions are synchronous to the system clock and their durations are multiples of the clock period. The protocol uses two handshake signals to coordinate the transfer: START/ (generated by the current bus master) and either TM0/ or ACK/ (generated by the slave).

Information is placed on the bus synchronous with the rising (assertion) edge and is sampled on the falling (sample) edge of the clock cycle. This technique provides protection from race conditions caused by bus transmission skews.

Read Transactions

Read operations with data widths of 8, 16, and 32 bits are selected by the transfer mode lines (TMx) and the two low-order address lines. The current bus master is responsible for selecting the appropriate byte or halfword for internal use if all 32 bits are not relevant.

Write Transactions

Write operations with data widths of 8, 16, and 32 bits are selected by the transfer mode lines (TMx) and the two low-order address lines. Bytes, halfwords, and words are organized as shown in Figure 2-1.

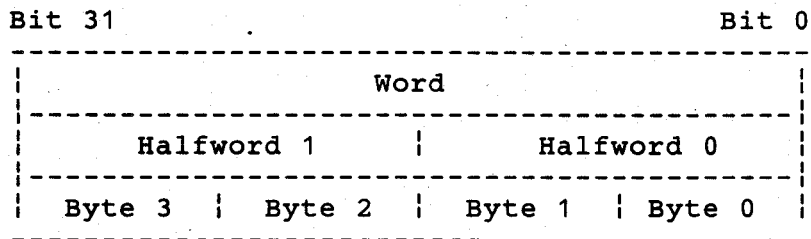


Figure 2-1. Layout of Words, Halfwords, and Bytes.

The current bus master has the responsibility of aligning the data to be written onto the appropriate ADx/ lines for halfword and byte writes. For example, a write byte 3 requires that the data be placed on AD24/ through AD31/; all other ADx/ lines are not defined and can be driven to any state.

Block Read

Block reads on the NuBus consist of a single command/address transfer initiated by the current bus master, followed by "N" 32-bit data outputs from sequential ascending locations of the addressed slave. "N" is equal to 2-, 4-, 8-, or 16-word blocks and is determined by AD2/, AD3/, AD4/, and AD5/ on the START/ cycle.

Block Write

Block writes on the NuBus are similar to block reads except the TM1/ (read/write) is driven to the write state during the START/ cycle, and the current bus master drives the data bus while the slave accepts data.

Error Acknowledgement

The two TMx lines are encoded at ACK time to indicate whether or not the bus transfer was successful. Four states are possible:

ACK

Successful completion of transfer.

NAK

Unsuccessful completion of transfer because of an error condition.

Timeout

Unsuccessful completion of transfer because "N" bus cycles elapsed while the bus was "busy" (between START and ACK).

Try Again Later

Unsuccessful completion of transfer but no error condition exists; the master should rearbtrate for the bus and try again.

2.4 Multiprocessor Support

Advanced arbitration and interrupt techniques provide the primary hardware support for multiprocessing.

2.4.1 Arbitration

Arbitration for the NuBus takes place each time control is transferred between bus masters. The winner of the arbitration contest takes control of the bus and retains control until an arbitration contest is won by another bus master. (Actual transfer of control does not occur until after the current bus master completes the data transfer in progress.)

The NuBus provides fair bandwidth sharing between processors in a multiprocessor system. The sharing is accomplished by a rule which is:

If several devices request the bus at a certain time, they are given the bus in priority order — highest to lowest. Fairness is provided by the rule that no new bus requests can originate from any device, including those in this group, until all devices in that group have acquired the bus.

This rule guarantees that processors of higher physical priority do not starve a processor of lower priority.

Once a bus master has acquired the NuBus, it is, by definition the highest priority bus master of the group of modules still requesting the bus. An undivided set of data transfers, such as in the "test-and-set" operation, can be accomplished by the bus master continually arbitrating for, and winning, the bus.

An arbitration contest takes place only when control of the bus must be transferred between bus masters. Therefore, if no other processor has requested the bus, the current bus master may initiate data transfers without first rearbtrating for use of the bus. This capability relieves the current bus master from the overhead of arbitrating for use of an idle bus and is termed "parking" or "glomming."

2.4.2 Interrupts

A processor module can be interrupted by any module on the NuBus performing a write transaction operation into memory space monitored by the processor. No unique lines or protocols are required. The address used to post the interrupt can be at any location in the address space. This allows interrupts to be posted to individual processors in a multiprocessor system and allows the priority of the interrupt to be software specified by memory-mapping the priority level.

3 68010 PROCESSOR BOARD

The CPU board design incorporates a single Motorola 68010 microprocessor, a high-speed cache memory, and a virtual memory translation unit. The main computational resource of the Nu Machine, the 68010 generates 24-bit virtual addresses, which allow access to 16M bytes per virtual address space. These addresses are termed "virtual" because each address is not the actual NuBus address in physical memory. Instead, a state machine on the board translates the virtual addresses to physical addresses.

Main features of the CPU board are:

- **68010**
 - 32-Bit Arithmetic
 - 17 Registers
 - 24-Bit Virtual Addresses for Each Process
 - 10 MHz Operation
 - Byte Addressability
- **4 KB Data Cache**
 - No Wait States on Cache "Hits"
- **"Demand Paging" Virtual Memory Implementation**
 - Translation Engine to Translate Virtual Addresses to Physical Addresses
 - 512-Entry Translation "Look-Aside" Buffer
 - Sufficient context stacking to permit recovery from page faults.

3.1 Virtual Address Space

In a typical multiprogramming environment, several "tasks" are active at the same time. The Nu Machine CPU provides separate virtual address spaces for each of these tasks.

Because even a single virtual address space may be too large to be contained in the available main memory, the active part of the virtual address space must be mapped to the available physical address space. To affect mapping, the physical and virtual memory are divided into 1K byte units called pages. A page of virtual memory is mapped either to a page in physical memory or to a page in mass storage.

Figure 3-1 is a simple representation of the virtual address space concept. It shows two virtual address spaces with most pages associated with unique physical memory pages. The second page in space "A" and the third page in space "B" are, in fact, the same page of physical memory.

A virtual memory scheme is needed to provide a large address space and yet allow programs to run on hardware with smaller memory size. The virtual memory hardware and software assign virtual addresses to specific physical memory locations in real-time while the process is executing, in some cases, having to bring the required location into memory from the disk. Usually the physical memory assigned to different processes will be non-overlapping. The paging mechanism, however, allows one physical location to be mapped into the virtual address spaces of two or more processes to implement data or code sharing.

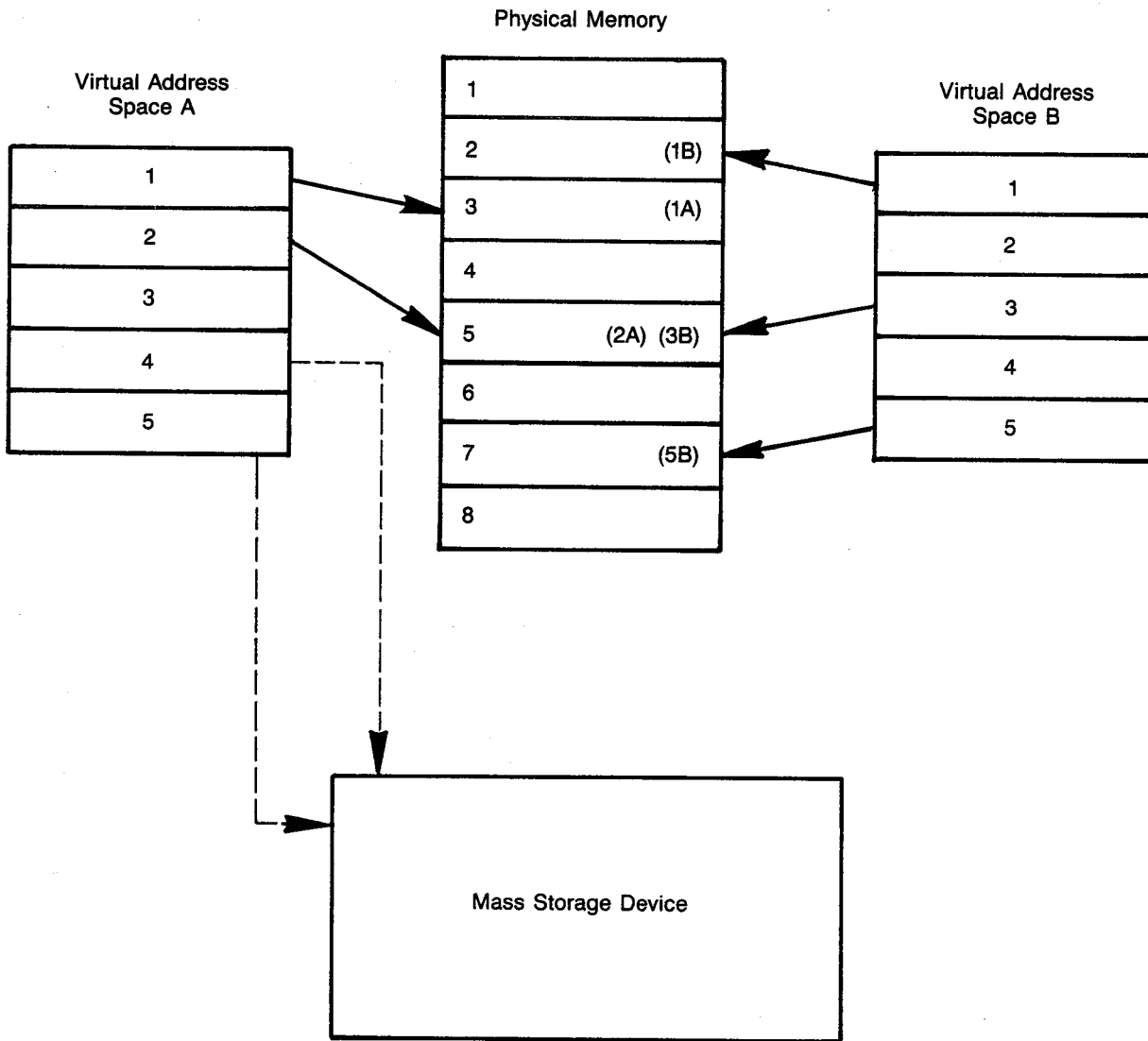


Figure 3-1. Virtual Address Space Concept.

Virtual-to-physical address mapping is only one part of a virtual memory scheme. The other parts provided by the Nu Machine hardware and software are a "fault recoverable" CPU and a page replacement algorithm.

3.2 Main Functions of the CPU Board

Figure 3-2 is a simplified block diagram of the CPU board showing the main functions.

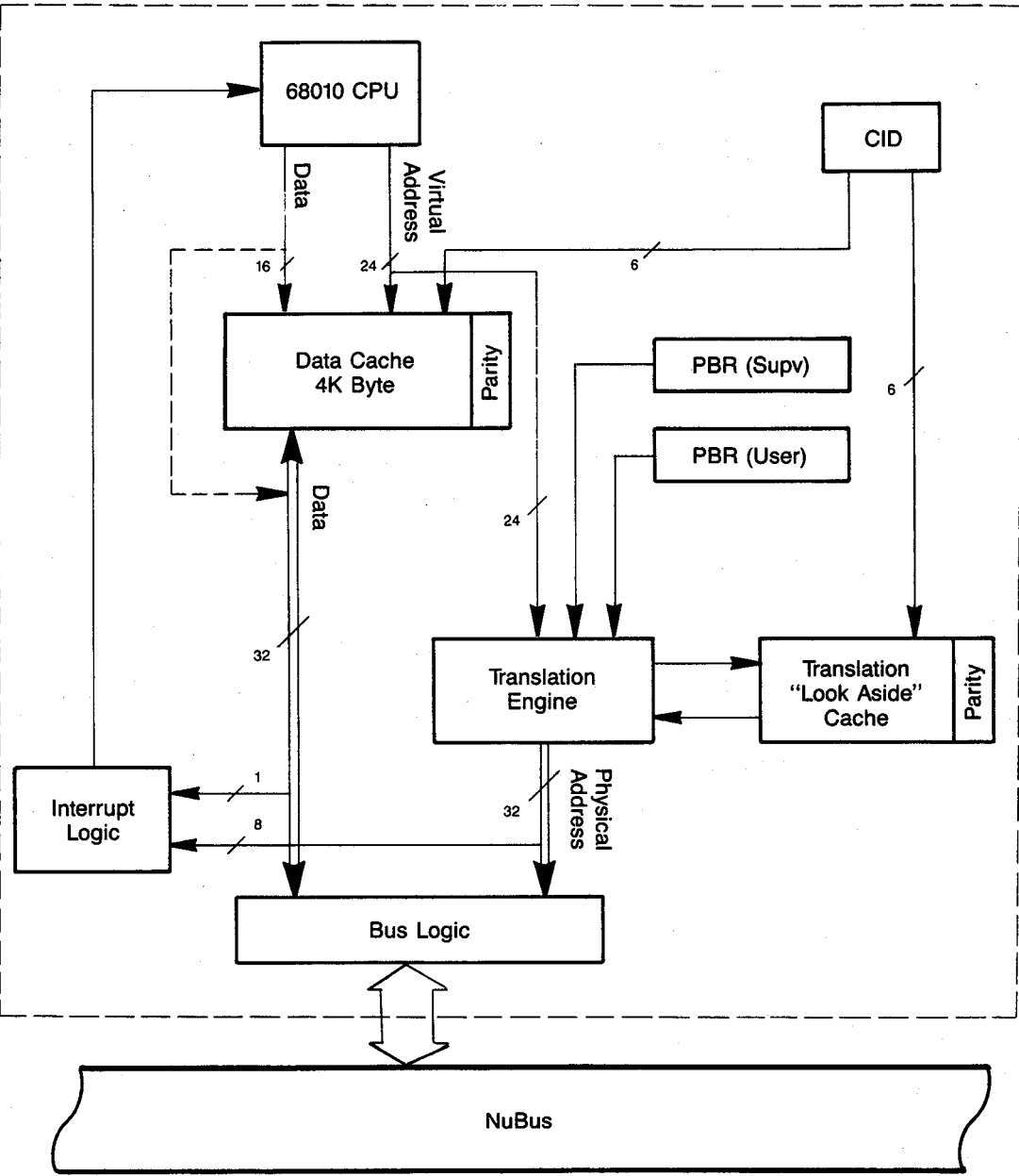


Figure 3-2. Block Diagram of CPU Board.

3.2.1 68010 Microprocessor

The 68010 microprocessor is a version of the 68000 which can correctly recover from a virtual memory page fault. The occurrence of a page fault (the page referenced does not reside in physical main memory) causes the 68010 to stack its internal machine state. At this point, the 68010 executes the software that causes the page to be brought into memory. The 68010 then retrieves its previous internal machine state and picks up execution at the point of the original page fault.

3.2.2 Cache Operation

The 4K byte, write-through data cache provides the CPU with high-speed access to frequently used words of main memory. Both instructions and data are stored in this unit and are treated equivalently. This set-associative cache contains 1024 one-word entries and carries byte parity. If desired, the cache can be disabled for diagnostics or to allow reduced performance providing graceful degradation in the event of cache failure.

The cache operates in a manner similar to many super minicomputers. A search of the cache determines if the desired data item is present. The search is performed for all 68010 memory reference operations as long as the cache is enabled. Each word of virtual memory has a place in the cache where it will be stored, if it is in the cache. The cache must examine only the location in which the word would be found if it were present.

The entry in the cache where a word is stored, if it is stored at all, is determined by the lower 10 bits of the address. That is, data from location 2003(Hex) would be stored in the third location of the cache as would data from location 5003(Hex). Only one of these pieces of data may be cached at any instant in time. The lower 10 bits of the address emitted by the 68010 are used to address the 1024 entry cache in implementing that operation of the CPU. The address tag, or upper 12 bits of the virtual address space of stored information, is compared with the upper 12 bits of the emitted address. Figure 3-3 represents a cache entry.

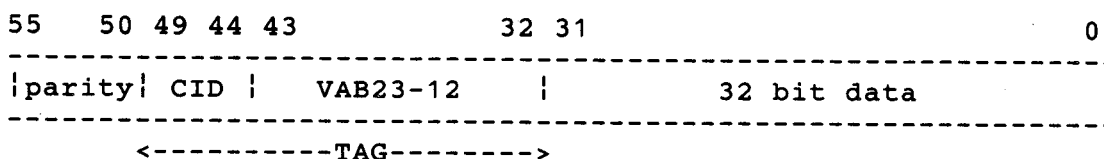


Figure 3-3. Data Cache Entry (Parity, Tag Field, and Data).

A "hit" occurs when the 12-bit address tag field stored at that entry matches the emitted address being sought by that particular 68010 cycle. The "hit" rate for this particular cache is approximately 85%.

A "miss" occurs when the upper 12 bits of the emitted address of the 68010 do not match the 12-bit address tag field of the stored data. The virtual address must then be translated into the physical address so that the bus logic will read the complete 32-bit word over the bus. The desired data item is placed in the cache entry at which the miss occurred, overwriting whatever data and address tag were there.

The write-through feature insures cache integrity because this cache is always a copy of memory. Therefore, on write operations when a hit or miss is made, address translation and bus logic are always invoked causing the written data to go into main memory. Thus, main memory is always a true copy of the state of the virtual address base, and the cache is a copy of that data. The cache is checked for a hit even on a write because if a hit occurs, that cache cell must be updated to the new value.

3.2.3 Cache Control

The cache data integrity is protected by byte parity across the data and tag fields. Any entry showing a parity error does not cause a hit and is treated as a normal miss except that the parity error flag in the error status register is set. Figure 3-4 shows the cache control register, and Figure 3-5 shows the cache "hit" status register.

For diagnostic purposes, the cache may be partially enabled or disabled via bits in the cache control. Partial enabling allows a diagnostic to test the cache without instructing itself to be cached.

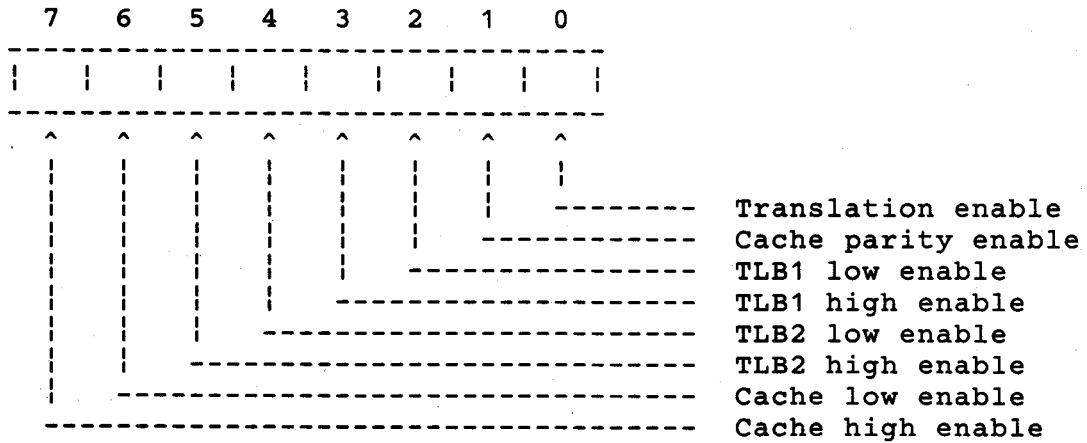


Figure 3-4. Cache Control.

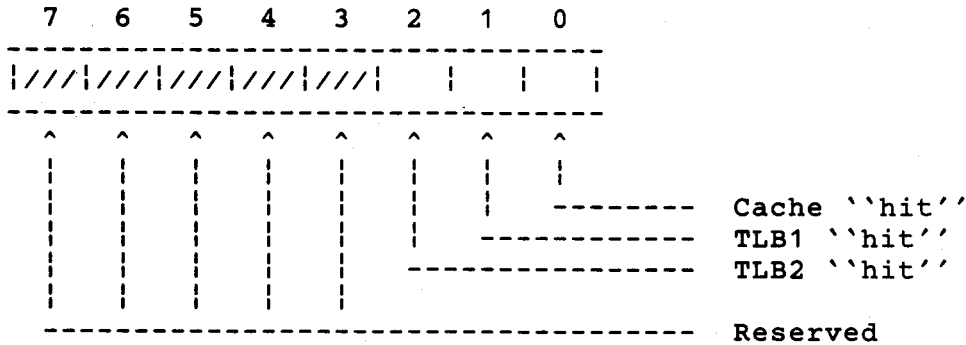


Figure 3-5. Cache Status.

An aspect of cache control that is not directly implemented by the cache logic itself is the ability *not* to cache selected pages of virtual memory. This capability allows support of writable, shared data areas; for instance, semaphores which must be accessed by multiple processes. The selective caching of pages is *not* directly implemented by logic in the caching section. Rather, the determination of which pages are not to be cached is a feature of the translation system.

Nu Machine Technical Summary Texas Instruments

Another application of selected caching is I/O registers. The selective caching of pages is *not* directly implemented by logic in the caching section. Rather, the determination of which pages are not to be cached is a feature of the translation system. When an I/O register is read multiple times, the desired data is *not* multiple copies of the first value but, instead, the current state of the I/O register.

3.2.4 Cache ID

Since the data cache on the Nu Machine is directly connected to the 68010 CPU, it caches data related to virtual address.

The advantages of this approach are obvious. Having the high speed cache (45 nanoseconds RAM) as close in the architecture as possible to the CPU eliminates wait states for the 10MHz 68010. In the future, faster microprocessors could be accommodated by increasing the speed of the cache memory path.

In other systems, this approach would have the disadvantage of taking extra time for cache flushes on every context switch. However, in the Nu Machine, a six-bit register and six extra bits of tag increase the speed at which the cache may be "flushed." The six extra bits of tag are called the cache identification (CID) field. Each active virtual address space has a unique six-bit CID code. When contexts are switched, the six-bit CID register associated with the 68010 CPU is changed. Therefore, cached data from previous virtual address spaces are immediately invalidated (flushed). An actual flush must be performed *only* between every sixty-four context switches.

3.2.5 Address Translation

The CPU card provides hardware to translate 24-bit virtual addresses to 32-bit physical addresses in 1024 byte page increments. Each page can be marked as resident/nonresident by the "valid entry" bit; has read and write access protection for both supervisor and user modes; has a bit to disable data caching on the page; and has both a page-accessed and a page-modified bit.

Address translation of the virtual address is accomplished by the translation engine and is based on two levels of page maps. The level-one page map defines the physical addresses of the level-two map. The level-two map defines the physical addresses of the virtual address space. The level-one map is 64 words long and the level-two map is 256 words long. The first page level table *must* be resident in main memory. Second level page tables may be resident or "paged" out to the disk.

Figure 3-6 illustrates the following explanation of the virtual-to physical address operation.

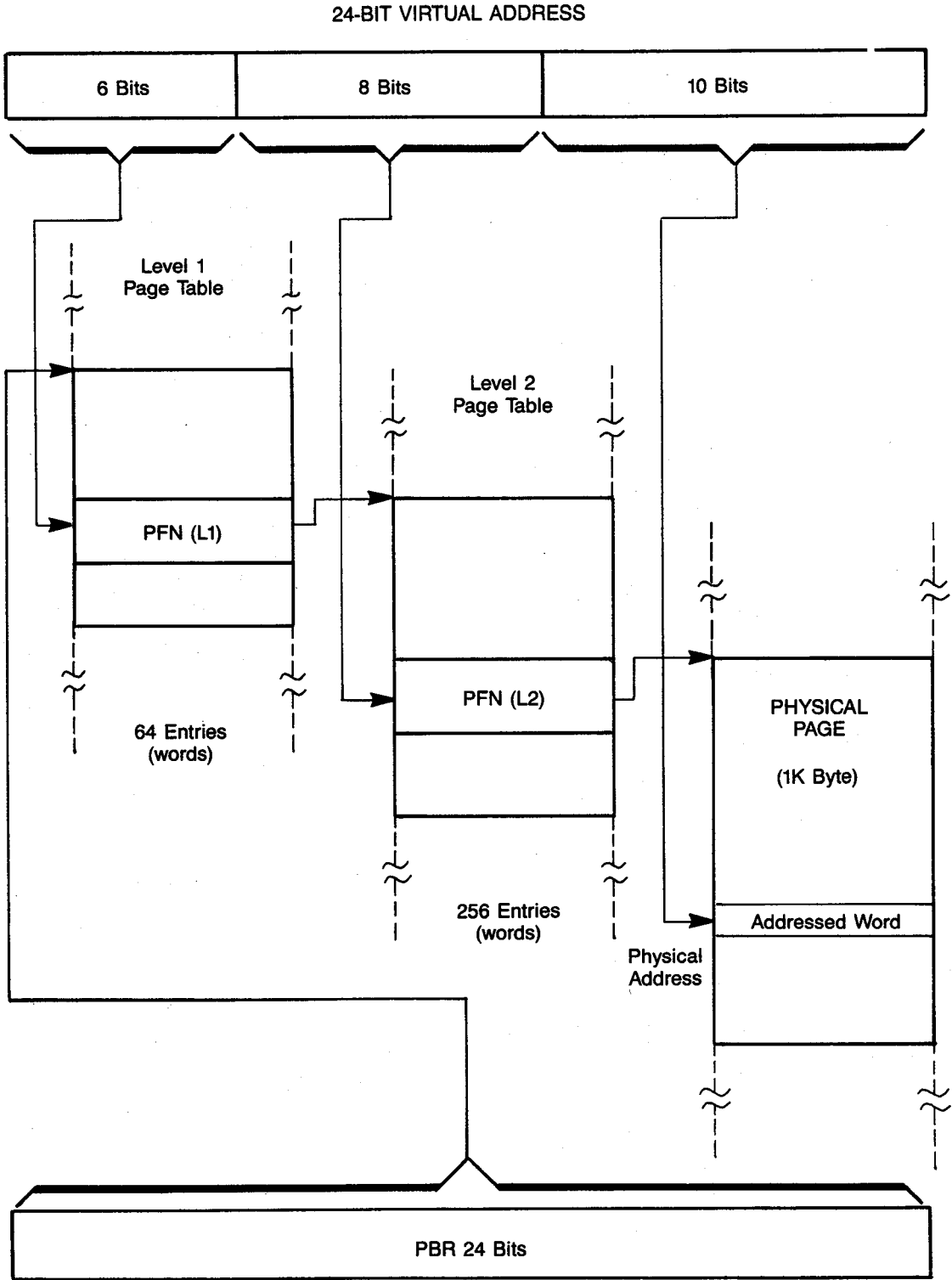


Figure 3-6. Virtual-to-Physical Address Translation.

Nu Machine Technical Summary
Texas Instruments

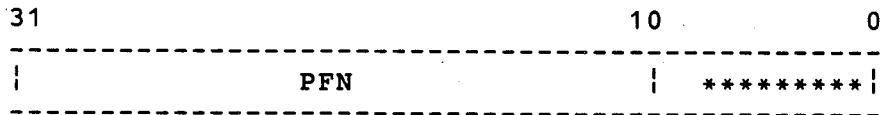
Abbreviations used in this description are:

- CID is a cache ID number.
- PADDR is the physical address.
- PBR is the process base register.
- PFN is a page frame number.
- PTE is a page table entry a 32-bit word that represents the physical mapping for one virtual page.
- PTE1 and PTE2 are level-one and level-two PTEs, respectively.
- VADDR is the virtual address from the CPU.

First, one of two 24-bit PBRs on the CPU card points to the start of the first level page table. (The hardware contains two PBRs — one for supervisor mode and one for user mode.) Then the first six bits of the VADDR (bits 23 through 18) are appended to the PBR to form a physical address for a PTE in the level one page map. This PTE is read from memory.

Next, the PFN of the PTE from the level-one page-table are appended to bits 17 through 10 of the VADDR. This forms the physical address of a PTE in the level-two map. This PTE is read from memory.

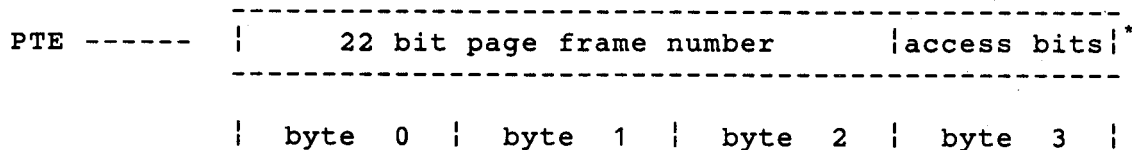
Finally, the PFN of this level-two PTE is appended to bits 9 through 0 of the VADDR. This represents the final physical address. Figure 3-7 represents the PFN from the level-one page table plus the virtual address bits and the data bits.



*****Filled with VADDR 17-10 for level-two PTE fetch.
 Filled with VADDR 9-0 for data fetch.

Figure 3-7. Representation of Physical Address.

PTEs, as shown in Figure 3-8 have a field, the PFN, that contains the high-order 22 bits of the physical address of a page. Pages always begin on multiples of 1024 bytes; therefore, the PFN with 10 bits of 0s concatenated is the address of the page.



(Note byte numbering)
 *See Figure 3-9 below.

Figure 3-8. PTE Description.

Each memory reference is checked for read and write access privileges and for the validity of the PTE. Figure 3-9 shows the definitions of the access bits.

```

Access bits:  BIT 31 - RESERVED
              BIT 30 - RESERVED
              BIT 29 - PRIVILEGE BIT 1 (PV1) \ ACCESS
              BIT 28 - PRIVILEGE BIT 0 (PV0) / PRIVILEGE
              BIT 27 - UPDATE CACHE
              BIT 26 - VALID ENTRY (page resident/nonresident)
              BIT 25 - ACCESSED
              BIT 24 - MODIFIED
    
```

Figure 3-9. Definitions of Access Bits.

The access privileges are further illustrated in Table 3-1.

TABLE 3-1. Privilege Bits

PV1	PV0	ACCESS PRIVILEGE
0	0	Supervisor R only
0	1	Supervisor R/W
1	0	Supervisor R/W, User R only
1	1	Supervisor R/W, User R/W

If a bit is "modified," a memory write sets it in the level-two PTE, which, if not already set, in turn, sets the "modified" bit in the level-one PTE.

Translation 'Look Aside' Buffer If the CPU hardware implemented address translation only as just described, each processor reference would cause two memory accesses to do translation. To speed up these translations, another caching mechanism, the Translation "Look Aside" Buffer (TLAB), holds an additional 512 second-level PTEs and 64 first-level PTEs. All caches are updated as necessary by the translation engine.

Therefore, if a "hit" occurs on a second-level PTE, no memory references are necessary for translation. If a hit occurs on a first-level PTE, one memory reference is required, and in the rare case where a hit does not occur, two memory references are required to translate the address.

3.2.6 Interrupt Mechanism

The 68010 microprocessor supports seven levels of interrupt priorities, labeled one through seven, with seven having the highest priorities. The Nu Machine CPU unit provides 32 unique vectored interrupts for each of these levels, totaling 224 possible cases. As described in Section 2.1 on the NuBus, the bus does not require traditional interrupt signals.

Nu Machine Technical Summary

Texas Instruments

The shortcoming of traditional interrupt approaches is that they do not support multiprocessing. In a multiprocessor environment, interrupts or "events" must be dynamically assignable to any of many processing elements.

Interrupts, like memory and I/O, are memory-mapped in the single NuBus address space. Therefore, computing units can readily interrupt each other or can even interrupt themselves from high-level language programs.

On the NuBus, events are posted by *ordinary* bus write cycles to *special* areas of the address space to which the CPU units respond. Logic on the CPU board translates all writes to that region into 68010 interrupts. These interrupts are latched and presented to the 68010 in priority order.

The priority level of an event is determined by its address within the 224-word area. For example, the top 32 words of the area correspond to level-seven interrupts; the next 32 words correspond to level-six interrupts, and so forth. When the 68010 requests the interrupt vector, it is given the unique address of the highest priority interrupt. This action immediately clears the particular interrupt.

3.2.7 CPU Control

The CPU control logic is primarily a debugging tool. The "Stop CPU" control bit may be used to halt the 68010 microprocessor upon completion of its current cycle. The halt request may then be removed allowing the 68010 to continue or begin instruction execution. Writing the "Single-Step CPU" bit enables the single step function. That is, when the 68010 begins its bus cycle, the stop CPU bit is turned on and, upon completion of that bus cycle, the microprocessor halts. The halt status of the 68010 may be obtained by reading the "Halt Status" bit.

4 SYSTEM DIAGNOSTIC UNIT

The System Diagnostic Unit (SDU) is present in all NuBus systems to provide many important one-per-system functions and "smart" front-end and diagnostic capabilities. By concentrating these functions on the SDU, rather than on separate CPU cards, the system is able to support multiple processors without conflict.

On power-up, the SDU verifies the integrity of the bus through a series of bus transfer tests. The SDU then identifies all boards within its system environment; initiates and monitors diagnostic and self-test routines to ensure that the boards are functioning properly; signals the operator, using an on-board serial I/O facility, if any boards have malfunctioned; and then initiates the system bootstrap program.

The SDU also serves as the NuBus-to-Multibus converter. Facilities are provided for mapping Multibus cycles into NuBus cycles and vice versa; for mapping Multibus interrupts into NuBus events; and for generating Multibus interrupts from the NuBus.

The SDU consists of an Intel 8088 microprocessor with on-board memory, two serial I/O ports for communication with the operator and peripherals, and many other system maintenance and diagnostic features. A functional block diagram is shown in Figure 4-1.

The main features of the SDU are:

System Clock

The SDU is the source of the 75% duty cycle, 10 MHz System Clock (CLK/), to which all bus operations are synchronized. The system clock rate can be programmatically margined up or down for diagnostic purposes.

Timeout Recovery

The SDU provides NuBus timeout recovery by monitoring the time between the START/ and ACK/ control signals. If more than 256 clock cycles occur, the SDU asserts the ACK/ signal with the appropriate TM0 and TM1 code for a timeout.

Nonvolatile Features

The SDU contains 2K bytes of battery backed-up CMOS RAM. This memory is used to store the system configuration information in a nonvolatile manner. The SDU also provides a battery backed-up time-of-day clock.

1/4" Tape Interface

The SDU contains a 1/4" streaming tape drive interface. This feature provides low cost transportable media for the Nu Machine. Since this interface is on the SDU, diagnostic routines can be loaded from this tape and yet the diagnostic unit remains self contained.

Nu Machine Technical Summary
Texas Instruments

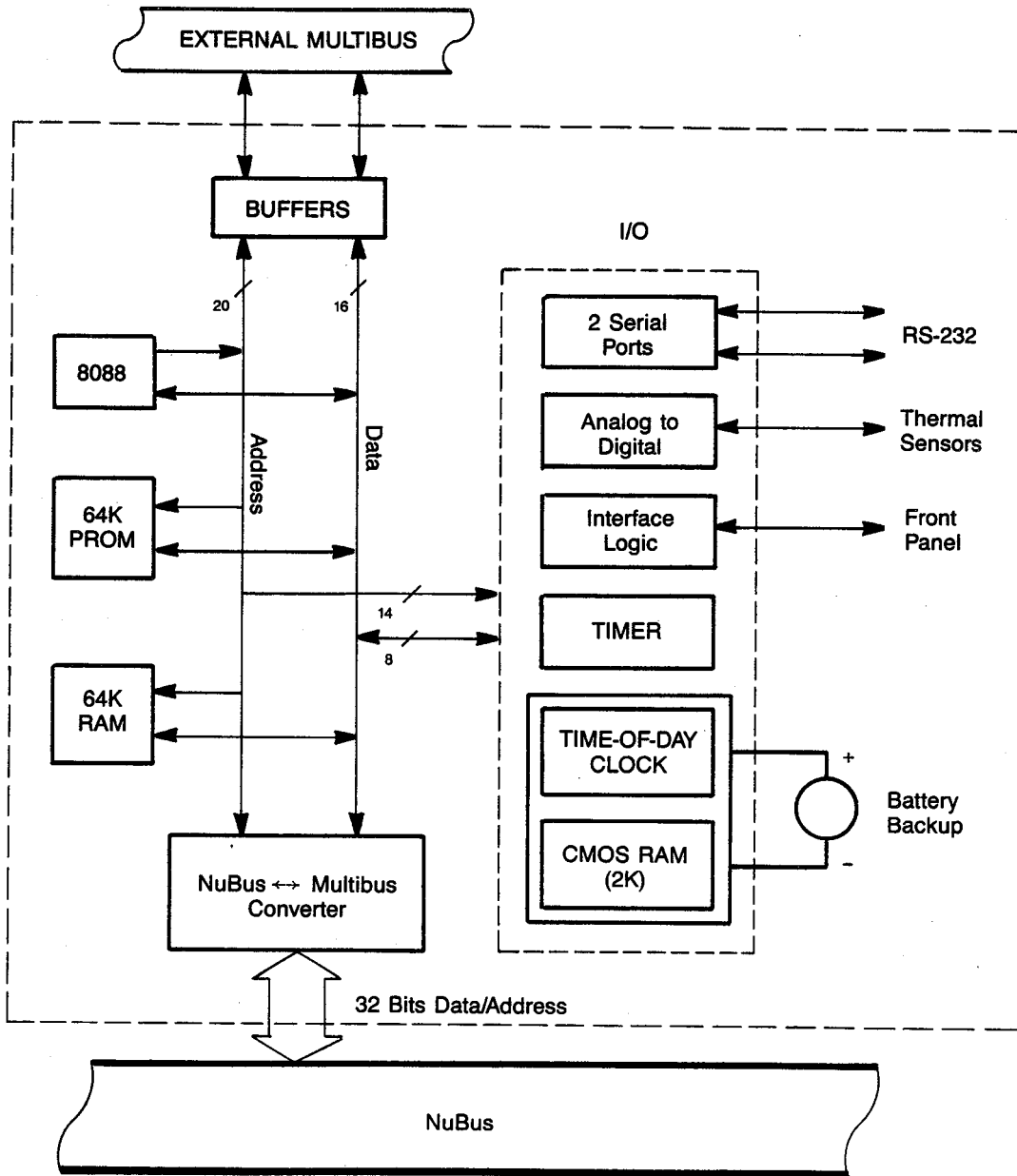


Figure 4-1. System Diagnostic Unit Block Diagram.

Interval Timer

The SDU contains a programmable timer for generating periodic events to specific CPUs. The timer may be used for many important system functions, such as process scheduling.

Debug/Diagnostic Facilities

The SDU provides the operator of the NuBus system several diagnostic tools. A monitor in the SDU's PROMs allows either serial port to be used to read and write bus locations, to initiate SDU self tests, and to execute diagnostics. The NuBus diagnostic hardware verifies the integrity of the bus.

System Status Display

On power-up, the SDU first runs a self test, next a bus test, and then individual board tests. The SDU uses the front panel LEDs to summarize the results of the tests, and the detailed results may be read using the system console. A LED on each board is automatically turned on at power-up, and turned off by the SDU if that board passes power-up diagnostics. A board with its LED on is therefore presumed bad.

Serial Port

The SDU contains two serial communications ports, either of which may be used as the smart front panel/remote diagnostics port, depending on the position of the diagnostic rotary switch. Otherwise, they are both available as general-purpose serial ports.

Power Supply Interface

The interface to the power supply includes several lines in addition to the actual current carrying cables. These lines are ACPF, DCOT, MARGHI, MARGLO, and ACOFF. The SDU provides the system interface to these lines. ACPF (AC power fail) is generated by the power supply. The SDU then posts "events" to the installed CPU's so that they can take appropriate action. DCOT (DC out of tolerance) indicates that the +5 volt supply is within tolerance (+/- 5%). When this signal is active the SDU generates a system reset. MARGHI and MARGLO are signals from the SDU by which the +5 volt supply can be margined +/- 7%. ACOFF is a signal from the SDU to the AC distribution box by which the SDU can shut off the AC power under user control or due to temperature excesses.

System Boot

The NuBus system boots automatically on power-up by action taken by the SDU. However, if for any reason program control is lost or undefined, the system may be reinitialized from the SDU manually. A system halt, reset, and reboot may be initiated from the console serial port.

Nu Machine Technical Summary
Texas Instruments

NuBus/Multibus Translation

The NuBus/Multibus interface is explained in Section 5.

5 MULTIBUS INTERFACE

Although the Nu Machine is designed around the high bandwidth NuBus, the Multibus (IEEE-796) is also part of the system. The two buses operate independently from one another except during a conversion. This combination provides NuBus advantages such as flexibility, speed, expandability, and multiprocessing with Multibus peripheral controller options. Further, Nu Machine users might use any of a large number (approximately 900-1000) of Multibus-compatible, board-level products from over 100 manufacturers.

5.1 Conversion

The NuBus/Multibus converter resides on the System Diagnostic Unit (SDU). Bus conversion in both directions is done by hardware mapping logic and requires no intervention by the 8088 microprocessor.

The Multibus interface is termed transparent because it translates NuBus transactions and Multibus transactions into each other. Further, the NuBus processor can access data or even execute programs out of the Multibus memory while conversion is taking place. Figure 5-1 illustrates how the transparent conversion acts like a "bus window." The entire Multibus memory space appears in the NuBus slot space of the SDU. When an address emitted by a NuBus master falls into this "window," the converter acquires Multibus mastership.

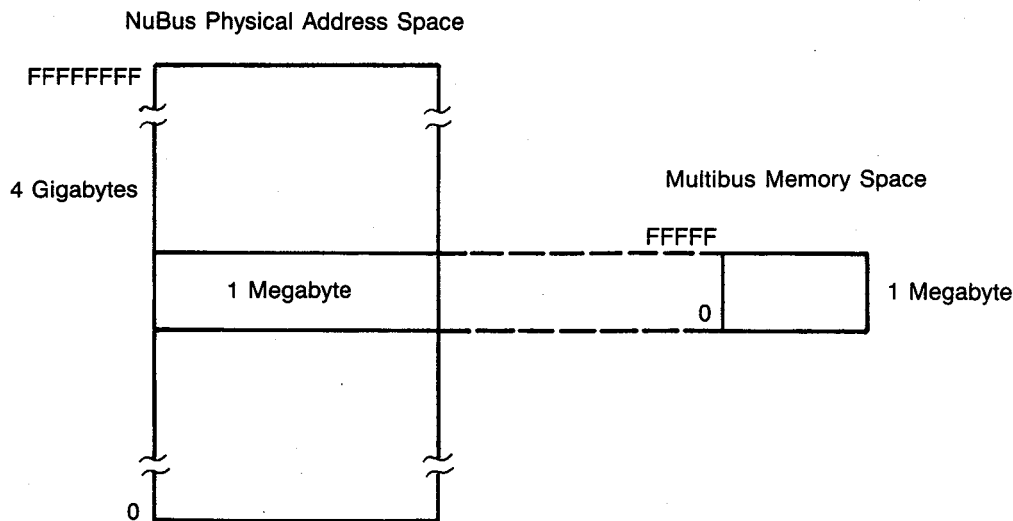


Figure 5-1. NuBus-to-Multibus Memory Space Conversion.

5.1.1 NuBus-to-Multibus Conversion

Memory Space NuBus 8- and 16-bit transactions are converted directly into 8- and 16-bit Multibus transactions. That is, the converter translates a transfer acknowledge on the Multibus to a transfer acknowledge on the NuBus. Bus conversions occur at bus speeds. NuBus 32-bit transactions are turned into two separate Multibus transactions to successive addresses.

I/O Space The 64K byte Multibus I/O space is mapped by the converter to a 64K-word region in the NuBus SDU slot space. Each byte of the Multibus I/O space is mapped to a separate word of the NuBus space. Thus, NuBus masters that are only capable of performing 16- or 32-bit operations are able to access individual byte locations in the Multibus I/O space. Figure 5-2 illustrates the Multibus I/O space mapping.

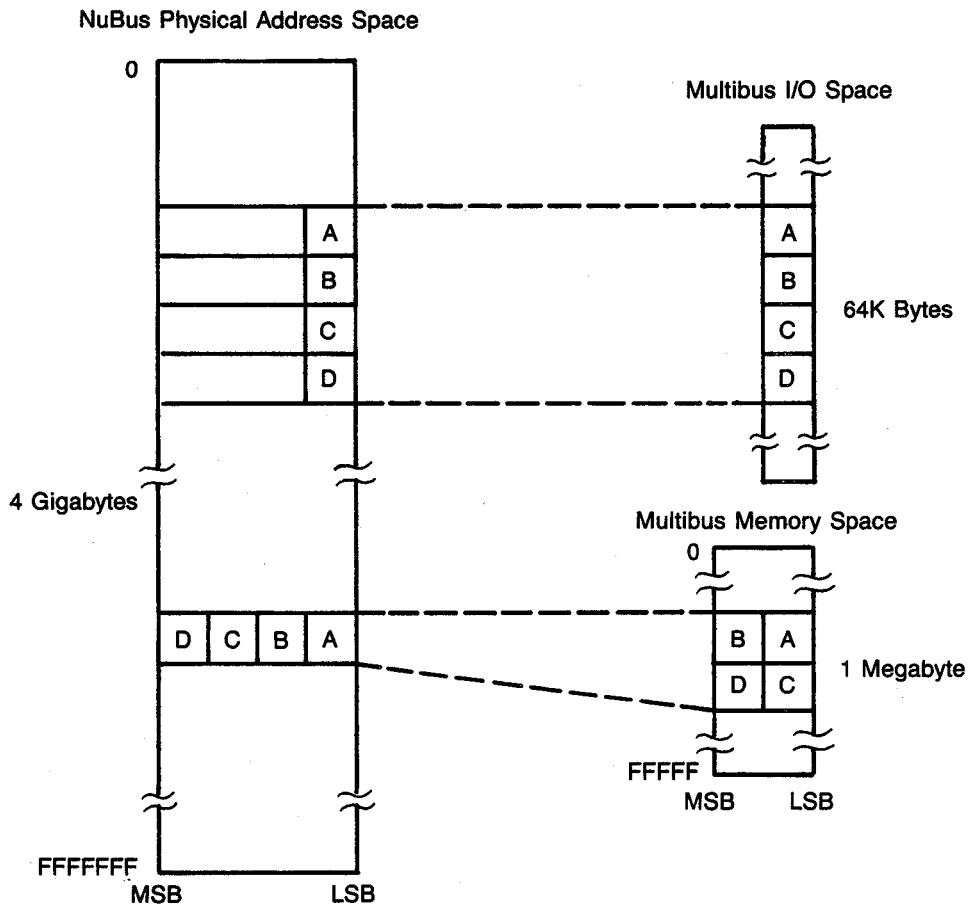


Figure 5-2. NuBus-to-Multibus I/O Space Conversion.

5.1.2 Multibus-to-NuBus Conversion

On every Multibus cycle, the converter (which is a slave device on the Multibus) checks to see if it is being addressed. This test to determine if the Multibus address page may be converted to a NuBus page is accomplished by using the upper 10 bits of the Multibus address to reference the Multi-to-Nu page map. A bit in each entry in this page map, called

the Valid Entry bit, determines if a conversion is to take place. If a conversion is required, the 22-bit PFN (page frame number) field of the PTE (page table entry) determines the page of NuBus memory referenced. The completed translation of the NuBus address is the concatenation of the 22-bit PFN field with the lower 10 bits of the Multibus address. Figures 5-3 and 5-4 show this process from two points of view.

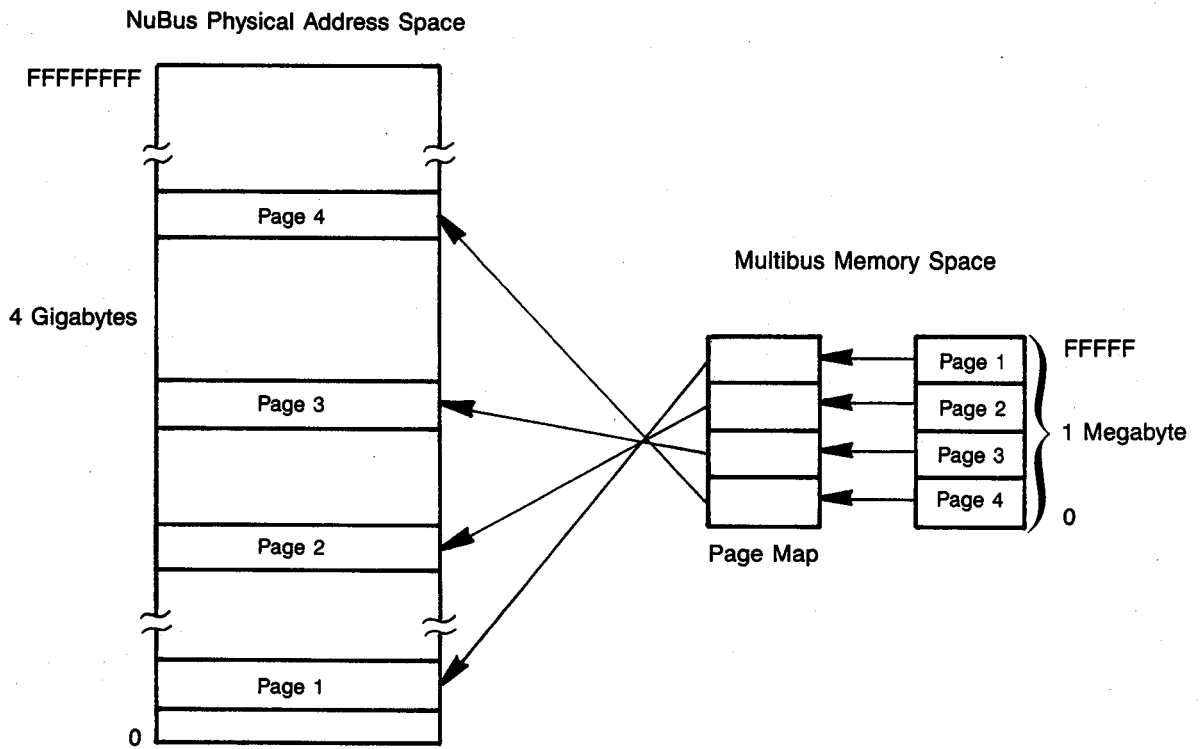


Figure 5-3. Multibus-to-NuBus Conversion.

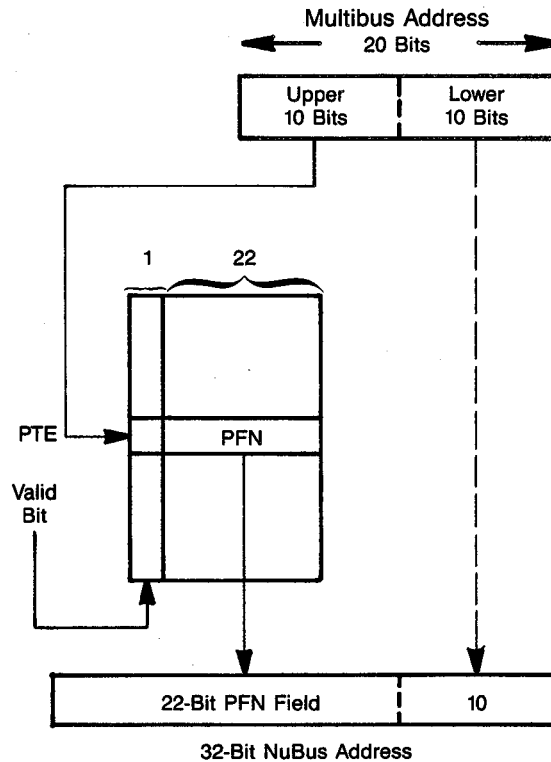


Figure 5-4. Multibus-to-NuBus Address Conversion.

5.2 Interrupt Mapping

Multi-to-Nu The mapping of Multibus interrupts is handled by the 8088 processor on the SDU board. When a Multibus interrupt is received by the 8088, the 8088 uses a table that associates Multibus interrupts with NuBus interrupt address locations to write into the NuBus address space. This causes the appropriate interrupt.

Nu-to-Multi Event cycles on the NuBus are mapped into Multibus interrupts by an addressable latch on the SDU. The NuBus writes to this area of the address space to create Multibus interrupts.

5.3 Lockup Prevention

In bus conversion, the potential exists that when both buses request the other a “deadly embrace” could occur. The Nu Machine protocol, however, provides an elegant solution to this problem.

If a NuBus master requests use of the Multibus converter at the same time that a Multibus master has requested the NuBus, the SDU returns a “Try Again Later” response. This informs the NuBus master that it is unable to respond at that moment. The NuBus master then releases control of the NuBus and must re-arbitrate for the NuBus again at a later time. This scheme frees the conversion path allowing the Multi-to-Nu transfer to finish.

6 VIDEO DISPLAY SUBSYSTEM

The video display subsystem is a single-board text and graphics display system. It utilizes a bit-map memory plane and video control circuitry to drive a high resolution noninterlaced monitor. Two video buffers on the board provide high bandwidth from the NuBus and high bandwidth to the display monitor. This board can be a master on the NuBus in order to generate interrupts.

Features of the video display subsystem are:

- One megabit of dedicated display memory (dual buffered);
- 1/60th of a second per screen update;
- 64K RAMs;
- Programmable number of words per line and lines per frame;
- Arithmetic Logic Unit (ALU) supports on-board logical functions such as XOR;
- Normal/reverse video;
- Horizontal, vertical, and composite synchronous outputs;
- TTL and ECL video outputs available;
- Programmable interrupt generations using vertical blank;
- Maximum pixel display rate of 70 MHz; and
- Two RS-232 serial ports for keyboard/mouse interface.

6.1 Main Functions of the Video Display Subsystem

The video display subsystem uses two dual-ported memory arrays. This unique approach enables the display to be refreshed from the video memory while, at the same time, the display may be changed by writing into the bit-map memory. Figure 6-1 is a functional block diagram of the board.

Nu Machine Technical Summary
Texas Instruments

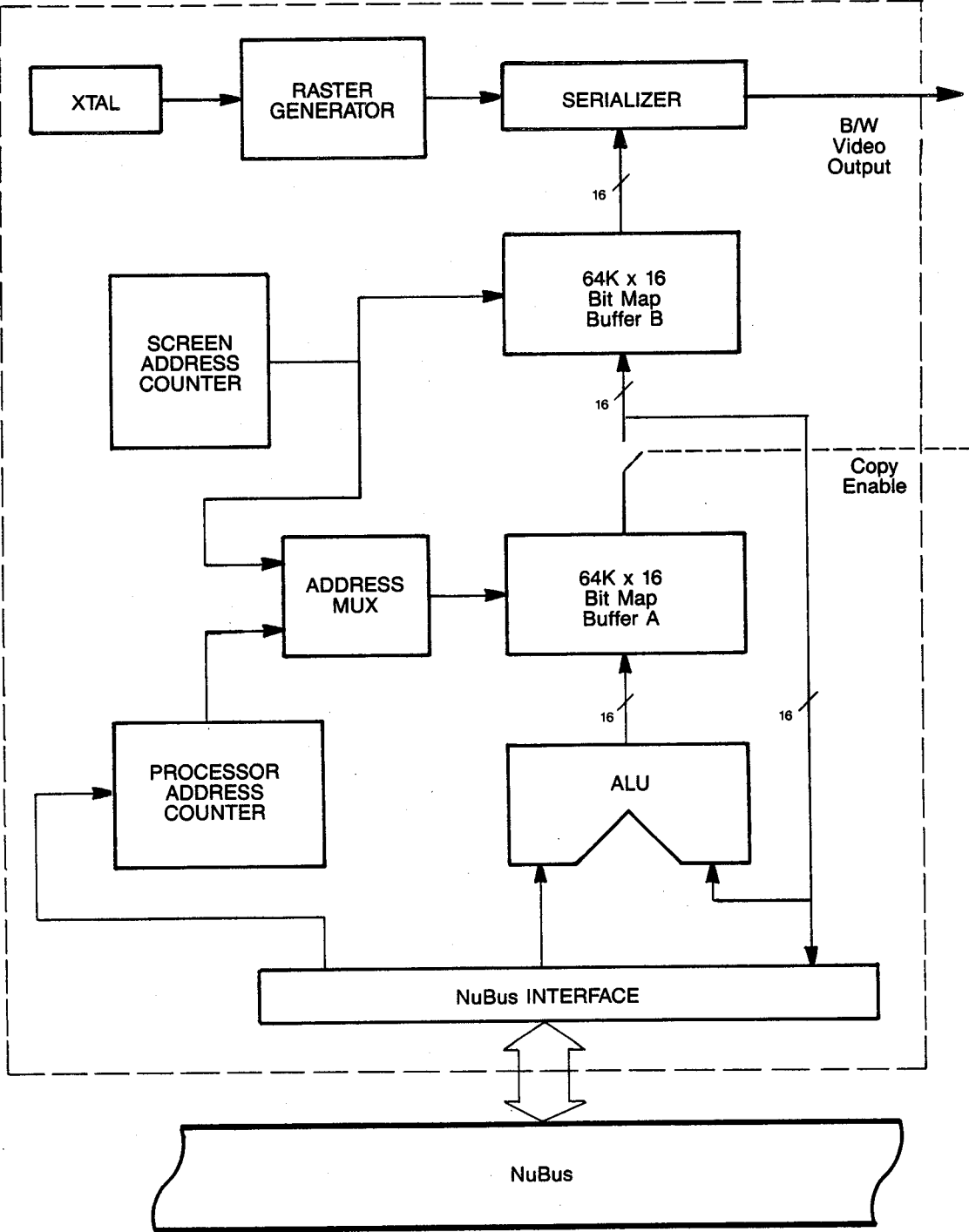


Figure 6-1. Block Diagram — B/W Video Display Subsystem.



The dual-ported arrays, called "A" and "B", are each a 16 × 64K memory. The lower (or B) memory is constantly addressed by the refresh address counters and may continually repaint the screen by providing a 70-Mbyte bit stream. This design leaves the "A" bit map free to accept new data from the processor controlling the graphics.

As seen in the block diagram, not only may data be written from the bus directly into memory "A", but also a read-modify-write cycle using a single ALU on the video board may be performed. This architecture allows information written to the board to be ORed, ANDed, or EXCLUSIVE ORed with values already in the bit map. This read-modify-write and on-board ALU allows many common operations to be performed within the video card itself.

In order for updated information to appear on the screen, bits are copied from the "A" memory to the "B" memory. In fact, when the "A" memory is not being accessed by the NuBus, it is copied into the "B" memory providing a continual update. However, the NuBus has priority over the copy/update operation, which provides the maximum bandwidth for delivering new information to the display.

The screen has the capability to show the old image until the new image is ready to be displayed. For this feature, the copy operation is inhibited by turning off the "COPY" bit in the command register. In this state, new information is written into the "A" memory is *not* displayed on the screen. Instead, the screen keeps refreshing the old information from the "B" memory. When the update is complete, the "COPY" bit is turned on, causing the new information to appear on the screen in one frame time (1/60th of a second). This makes the Nu Machine suitable for animation and other applications where it would be undesirable to watch the screen being gradually updated.

Another advanced feature is the scan line table, a RAM that contains the starting addresses of each scan line in the "B" memory. This mechanism is used for scrolling and making rapid changes in small portions of the screen. It also could enable display screens with different aspect ratios, to be used, thereby enhancing the subsystem's flexibility.

6.2 Serial Ports

Two RS-232 serial ports on the video display board complete the man-machine interface. Although unrelated to the production of video signals, they support the keyboard and mouse which may be associated with each display.

Software requirements for servicing the RS-232 ports are simplified by a small amount of hardware buffering. There are 4 bytes of FIFO on the input side of each port and 2 bytes of FIFO on the output side of each port.

**Nu Machine Technical Summary
Texas Instruments**



7 MAIN MEMORY

The Nu Machine memory board is a self-contained memory controller and 1/2 Mbyte memory array that includes logic to support error correction, block transfers, and error logging. The memory board interfaces to the NuBus as a "slave" device, responding to requests from a bus master. Up to 14 memory boards can be installed in the Nu Machine, yielding a maximum of 7M bytes of available physical memory.

7.1 Main Functions of the Memory Board

The main functions of the memory board reads, writes, and error detection are discussed below. A functional block diagram of the memory board is shown in Figure 7-1.

The memory board uses a 32/39 modified hamming code for error correction and detection; seven ECC (Error Correction Code) bits are appended to each 32-bit data word. This code enables correction of all single-bit errors and allows detection of all double-bit errors within the 39-bit word (32 data bits plus seven ECC bits). Additionally, some multiple-bit errors are detected. A multiple-bit error is one in which three or more bits in a particular word are in error.

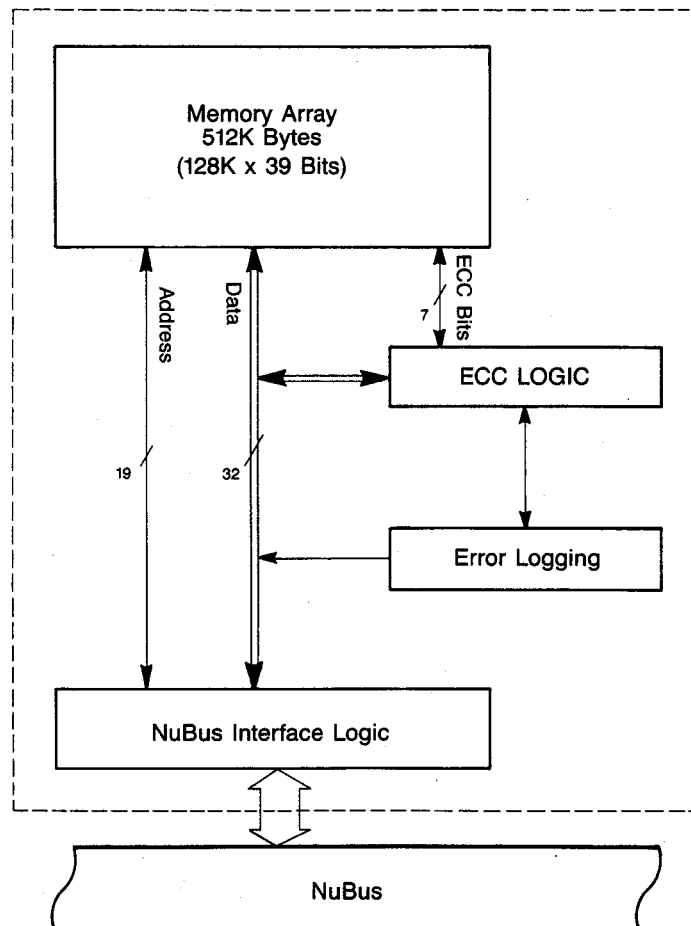


Figure 7-1. Memory Board Block Diagram.

Nu Machine Technical Summary
Texas Instruments

Read Operations — A read cycle always results in a 32-bit array access. However, a full 32-bit read of the array generates the correct data on the bus for all size objects because of the way byte and half-word reads are specified.

The memory board checks the data for errors on all read cycles and then performs one of three operations:

- If no errors are detected, data are placed on the bus, and a bus transfer complete (ACK) response is given.
- If a single-bit error is detected, it is corrected, and the corrected data are placed on the bus with a bus transfer complete (ACK) response. This correction is transparent to the bus master. The corrected data are *not* written back to memory.
- If a double-bit error is detected, an error condition (NAK) response is given, and the data transferred are undefined.

Write Operations — Write cycles may result in 8-, 16-, or 32-bit data transfers. All write operations are carried out "off-line." That is, the data are latched, the bus transfer completed (ACK) signal is given, and the bus is released while the write operation is completed.

A 32-bit transfer is the simplest write operation. The latched data are presented to memory, a corresponding seven-bit ECC pattern is generated, and both are written to the addressed location. On 8- and 16-bit write transfers, a read of the addressed location is performed first. One of three things then occurs.

- If no errors are detected, the new data are substituted for the old data; a new ECC pattern is generated; and the new word and ECC are written in the addressed location.
- If a single-bit error is detected, it is corrected before the new data are substituted. This new word and its ECC pattern are then written in memory.
- If a double-bit error occurs, the write portion of the partial write is not performed. Thus, the data with the error are left in memory so that a subsequent read will detect an error. No indication of the error is immediately returned.

Block Read Operation — Block read cycles result in 2-, 4-, 8-, or 16-word (32-bits/word) block transfers. Each word of data is read and the data are checked for errors. Then one of three operations is performed.

- If no errors are detected, the data are placed on the bus. If that transfer is the last one, a bus transfer complete (ACK) response is given. If that transfer is not the last one, and intermediate "ACK" is given and the transfer continues.
- If a single-bit error is detected, it is corrected, and the corrected data are placed on the bus. If that transfer is the last one, a bus transfer complete (ACK) response is given. If that transfer is not the last one, an "intermediate" ACK is given and the transfer continues.
- If a double-bit error is detected, undefined data are transferred, an error condition (NAK) response is given, and the block transfer terminates.

Block Write Operation — Block write cycles result in 2-, 4-, 8-, or 16-word block transfers. Each word of data are latched and acknowledged. (An "intermediate" ACK response is given for all words except the last word transferred.) When the last word is transferred, a Bus Transfer Complete (ACK) response is given, and the bus transfer is complete. Each word of data is latched and presented to memory; a corresponding seven-bit ECC pattern is generated; and both are written to the addressed location. Then the address is incremented to the next word. This process occurs for each word of data to be written until all words have been transferred. Because complete words are written, no errors can occur on a block write operation.

8 SOFTWARE

Two categories of software are available with the Nu Machine. The first category consists of the SDU Monitor and the diagnostics that run under that monitor.

The second category of software is the Nu Machine Operating System. This is a port of Bell Lab's UNIX Operating System. The Nu Machine Operating System is derived from UNIX, Seventh Edition and contains, in addition to the normal UNIX commands and kernel, enhancements from the Berkeley version of UNIX and a proprietary window system for the Nu Machine video display subsystem. Of course, the Nu Machine Operating System will not run on Nu Machines that do not contain a 68010 processor board.

8.1 SDU Monitor and Diagnostics

8.1.1 SDU Monitor

The SDU Monitor's primary function is to provide device independence for the diagnostics, bootstraps and other commands that run on the SDU. In addition, certain primitive commands built into the Monitor allow the user or service representative to probe the hardware even if the mass storage devices are broken.

A mode switch on the back of the machine selects a console device to access the SDU Monitor. One hardwired position selects a serial port on the SDU and sets it to operate at 300 baud. The functions of the other switch positions are defined by data in a non-volatile RAM.

Primitive Commands Primitive commands available on the SDU Monitor are:

- Reading and writing NuBus and Multibus locations;
- Selectively resetting and enabling the buses;
- Clock and voltage margining to aid in diagnosis;
- Copying data between devices; and
- Downloading commands from the serial ports.

Device Support The Monitor has a uniform device access mechanism similar to that provided by UNIX. In fact, some UNIX commands can be recompiled and run directly under the Monitor. Several device drivers are built into the Monitor. In addition, the Monitor has a facility for installing new device drivers while it is in operation. Device support is provided for:

- Serial ports on the SDU;
- The high resolution display and keyboard;
- The Multibus SMD controller;
- The Multibus 1/2-inch tape controller;
- The 1/4-inch tape controller;
- The non-volatile RAM; and
- The time-of-day clock.

For many of the devices, both raw and structured access is provided. For example, the disk may be accessed as a simple sequential byte stream and may also be accessed via a UNIX file system driver. The file system driver is not built into the Monitor but rather is "booted" from the disk when

Nu Machine Technical Summary

Texas Instruments

it is first accessed. This means that it can be replaced by different format file systems. Similarly, the tapes can be accessed both as a byte stream and as a structured file system in the UNIX "tar" format.

Loadable Commands To expand the command repertoire, the Monitor loads commands from several of the devices into RAM on the SDU, passes on arguments to these commands, and runs the commands. Commands are normally loaded from the disk but can also be loaded from the tapes or via host machine serial ports. Loadable commands perform I/O via standard input and output streams, unaware of which device they are talking to.

More important loadable commands are:

- The "setup" program used to manipulate non-volatile RAM;
- The main system and board diagnostics;
- The Nu Machine Operating System bootstrap program;
- The UNIX file system checker; and
- The UNIX dump program.

8.1.2 Diagnostics

Nu Machine diagnostics conveniently verify the correct functions of the system and aid in the debugging of a downed machine. Two kinds of diagnostics are available: startup diagnostics and loadable diagnostics. Both forms of diagnostics run under the Monitor; however, the startup diagnostics are simpler and return only an error code rather than the more extensive output of the main system diagnostics.

Startup Diagnostics Startup diagnostics are loaded from ROMs on each board as part of the power up sequence. They provide a relatively quick verification of the correct functioning of a board but do not provide much information as to the nature of any errors. Being resident on the boards, these diagnostics do not require any peripherals to be functioning and are always appropriate for the board on which they reside.

SDU Loadable Diagnostics Loadable diagnostics on the mass storage devices verify the correct functioning of system components. These diagnostics take longer to run but provide more information about each test performed and conditions on the device. Loadable diagnostics run on the SDU make only minimal assumptions about how much of the system is functioning. If the SDU and bus are functioning, boards can be independently tested.

All loadable diagnostics are written in a high level language, C, and use a common procedure that specifies the flow control and user interface. Therefore, loadable diagnostics are relatively simple to use and provide highly structured input and output for machine interpretation. Once an operator learns to use one diagnostic he has essentially learned all of them. They have a command-oriented interface, similar to UNIX commands, instead of menus. This interface is more effective for remote diagnostics and lends itself to machine automated diagnostic procedures.

Diagnostic Development Kit The development tools used to create the Nu Machine diagnostics are available as a software option for those who wish to write their own SDU programs and diagnostics for experimental boards.

Included in this kit are:

- a C compiler, assembler and linker for the 8088;
- a standard set of C runtime routines; and
- a library of standard Nu Machine diagnostic routines.

The tools run under the Nu Machine Operating System implementation but since they are written in C, they could also be ported to other machines.

8.2 Nu Machine Operating System

8.2.1 Overview

The Nu Machine Operating System is a port of UNIX Version 7 with extensions that run on the 68010 processor board and make use of the Nu Machine I/O devices. It consists of the kernel, and includes language processors, text processing tools, and system maintenance utilities. To port the kernel, the machine dependent modules of UNIX such as the memory management and the device drivers have been re-written to be appropriate to the Nu Machine. Most of the machine independent modules, as well as the commands were simply recompiled without change.

UNIX was selected as a basis for the Nu Machine Operating System because its applications are broad and its advantages are both practical and technical. UNIX is perhaps the most popular programming environment available. It is capable of supporting the most demanding applications as well as providing an effective base for every-day applications. Many people have been exposed to UNIX at universities, providing a large base of users and programmers familiar with the system. Since UNIX has been readily ported to many machines, there is a wide selection of application software available and a large market ready for further applications.

On the technical side, UNIX offers a variety of advantages:

It is a timesharing system.

Additional terminals and users can be conveniently added to the system at minimal cost. Even in a single user environment the user may have additional activities, such as print spoolers and network demons, operating on his behalf.

Terminals, files and other processes have identical I/O.

This dramatically extends the power of UNIX applications. Since input can come from files as easily as from terminals, output can be saved in a file for later processing or evaluation. Furthermore, input can come directly from other programs and be passed directly to other programs for further processing in any particular run of an application. Programs can be connected together as building blocks combining their effectiveness in a multiplicative way.

A tree-structured file system with access protection.

The hierarchy of directories provides a simple, convenient method for organizing files. Access protection promotes cooperation in a multiuser environment by permitting the user to decide to what extent he should trust others.

Nu Machine Technical Summary Texas Instruments

UNIX is simple.

The kernel is small and easily understood. The commands are easy to learn and operate. Overall, UNIX is remarkably free of annoying restrictions and arbitrary details. Although simple, UNIX has great sophistication able to provide adequate performance while retaining generality.

8.2.2 Nu Machine Operating System Kernel

The Nu Machine Operating System kernel is the control program which implements the system calls that all application programs use and the device drivers that access the hardware devices. The kernel is a recompilation of UNIX, Version 7, with portions changed to adapt UNIX to the Nu Machine. These changes primarily affect the memory management routines and the device drivers.

The Nu Machine 68010 processor uses page-oriented memory management. Within memory, page tables describe the virtual-to-physical address mapping for a particular process. The memory management scheme supports the full 24 bit address space of the 68010 processor. It also takes advantage of the rapid context switch mechanism provided by the cache ID bits in cache entries and the process base register. In the current release, processes are swapped as is done in the standard UNIX implementation. Therefore, the process size is limited to the amount of physical memory installed on a particular machine. Future releases will operate in a demand page mode which will take full advantage of the Nu Machine's hardware capabilities.

Additional and modified drivers are provided for all the standard Nu Machine peripherals. These include:

SMD disk

The disk driver takes advantage of the bad track mapping provided by the controller. It performs an elevator algorithm sort on the disk queue to minimize head movement.

1/2" tape

The 1/2 inch tape streamer operates the tape in start/stop mode for an effective tape speed of 25 inches per second. It performs extensive error checking and automatic retries on error conditions.

1/4" tape

The 1/4 inch tape operates in streaming mode at a peak tape speed of 90 inches per second. Controller limitations force all tape blocks to be multiples of 512 bytes.

SDU USARTs

Two serial ports are provided on the SDU. These ports are operated in asynchronous mode with full baud rate control and limited modem control. Two teletype line disciplines are provided—the standard UNIX, Version 7, discipline and the more advanced Berkeley line discipline. Depending on the contents of the "/etc/ttys" file, these ports may be configured to support either a terminal with a normal login process or a special demon such as a line printer spooler.

VCMEM

This driver, referred to as the Raster Scan Display driver or RSD, implements a powerful window system. Drivers support the window system, including the virtual terminal interface with ANSI X3.64 command sequences, the Nu Machine keyboard, and the mouse.

8.2.3 Utilities

The Nu Machine command environment is the standard set of tools delivered with the operating system. The tools may be divided into three groups: text processing and miscellaneous tools, programming tools, and the window system and graphics.

Text Processing and Miscellaneous Tools Most of the commands provided by UNIX, Version 7, are available on the Nu Machine operating system. These include the Bourne shell, the file system utility commands such as *ls*, *cp*, *mkdir*, and various miscellaneous commands such as the desk calculator *bc*, the online manual *man*, and the electronic mail program *mail*. In addition to the Version 7 commands, commands from other sources, especially the Berkeley C shell, have been included.

Text processing tools available on the Nu Machine operating system include the line editor *ed*, the regular expression search program *grep*, and the text formatter *nroff*. Additional Berkeley tools include the screen editor *vi* and the file perusal program *page*, both of which use the terminal support package *termcaps*.

Programming Tools Two compilers, a C compiler and a Fortran compiler are currently supplied with the Nu Machine operating system. They are both different variations of the same compiler, the *UNIX Portable C Compiler*, with different front ends. The C version embodies all the features of the Kernighan and Ritchie book, *The C Programming Language*, and generates 32 bit ints. The Fortran version implements Fortran 77 and includes a Ratfor front end. Both versions generate assembly code that, as part of the compile, is fed to an assembler for the 68010. This assembler is available for special purpose routines.

Nu Machine compilers support separate compilation and linking which dramatically speeds the building of large programs. Extensive libraries of support routines are provided that exploit the linking mechanism. Those on the Nu Machine include the *standard I/O library*, a library of floating point routines, required since the 68010 does not have a hardware floating point, a mathematics library and the Berkeley terminal independent I/O libraries, *curses* and *termib*.

The Source Code Control System is a collection of programs that implement version control. They keep track of different versions, pull together the various pieces of the code and keep them up to date, and check code for portability to other environments. They can be used to control any sort of text file, not just programs. The program *make* uses the modification date of files to determine which subportions of a program need to be rebuilt to make the program up-to-date and allows the programmer to specify what tools to invoke to create the program. There is a program for C called *lint* that checks for portability across machines and performs a variety of other useful checks to identify errors that the compiler would not catch.

To aid in debugging programs, a powerful UNIX debugging tool, *adb*, is available on the Nu Machine. Similarly, to aid in understanding program behavior a trace package is available that allows the programmer to count how many times each subroutine is called and to compute what percentage of total execution time is spent in each routine.

Nu Machine Technical Summary Texas Instruments

The Window System and Graphics The window system is an extension of the kernel and user commands that provides multiple virtual terminals on the Nu Machine high resolution display. This feature of the Nu Machine Operating System is not a standard part of UNIX, and was derived at M.I.T. The window system implements the following features:

- Multiple windows* A window is a raster of pixels, that is, a rectangular array of dots either on or off. Each window is treated as an independent entity by the window system and may be used to implement a virtual terminal. Logical windows are mapped to physical portions of the high resolution screen. They may be partially or totally occluded, that is, covered up by other windows. The user may change the mapping of windows to the screen, moving windows and causing, for example, an occluded window to be exposed. When a process tries to write to an occluded window, it will block until the window is fully exposed.
- Terminal emulation* One window function is to implement a virtual terminal so that when a user process writes to a window, characters appear on the screen. One window is designated as the keyboard window and can be identified by a blinking cursor. Characters typed on the keyboard appear in this window. The keyboard may be detached from one window and attached to another, effectively switching it between processes. As part of the terminal emulation, ANSI X3.64 control sequences move the cursor, and erase portions of the screen. This emulation is known as *termcaps*.
- Multiple fonts* Another aspect to terminal emulation is that the user can specify the font, ie. the size and shape of characters displayed in the window. Each window can support multiple fonts and the user can switch between them with control commands.
- Screen mapping* The user may map a window directly into the address space of a process. The window simply appears as an array of bits in the address space. In this way, the program can directly control each bit of the window and implement its own graphics routines and character painting for maximum efficiency without any intervening code.
- Mouse support* The window system includes a mouse driver that permits an application program to read the current position and button state of the mouse and optionally gives the program an interrupt when the position or button state changes. The mouse is associated with the current keyboard window and changes between windows as the keyboard moves.

In addition to the code inside the kernel, several commands permit the user to manipulate the screen. A graphics library is available to aid in developing programs that use features of the high resolution screen. User commands are available to create new windows, change the properties of a window, change fonts and to display UNIX plot format data in a window. This last command, *trsd*, can be used in conjunction with standard UNIX commands to create graphs on the screen and draw other kinds of pictures. *Trsd* makes use of the graphics library *libg*, a set of routines to draw vectors, manipulate rasters and paint characters onto the screen.

8.3 Future Plans

The following items describe planned enhancements that will be available in future releases of the Nu Machine Operating System.

- Demand paging* The Nu Machine 68010 processor board has the capability to recover from page faults but the Nu Machine Operating System does not yet take advantage of this capability. When this feature is implemented, it will remove the restriction that programs must fit in the available memory. Furthermore, it should improve the overall efficiency of the system particularly for large programs by making more effective use of the available memory.
- Local networking* Support for the Ethernet with both the XNS and IP/TCP families of protocols is planned for the Nu Machine.
- Multiprocessing* The NuBus and the Nu Machine 68010 processor have been designed so that multiple processors can reside on the same bus. When these hardware features are fully exploited by the software, the Nu Machine operating system will support load sharing between multiple processors to increase the amount of computer power available in a multitasking environment. Owners of Nu Machines will be able to extend the power of their machine by simply adding more processor boards.

**Nu Machine Technical Summary
Texas Instruments**

1

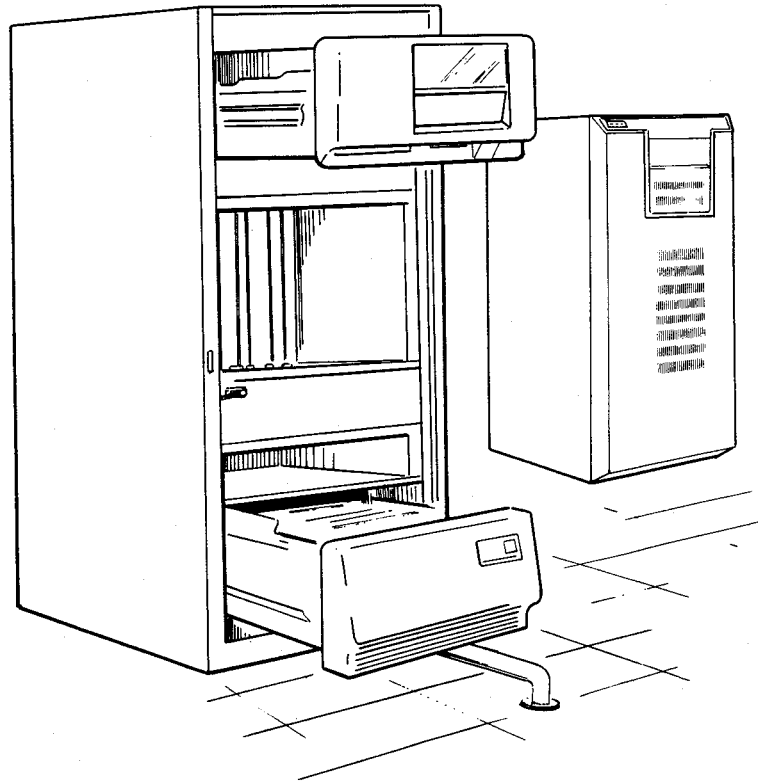


Figure 9-2. Rack Module and Component Layout

9.5 Summary

The Nu Machine is the result of research done at M.I.T. combined with product development by industry. This combination has created a state-of-the-art system which is well thought out and expertly engineered. The system framework provides the performance and flexibility to create a wide range of computing environments.

**Nu Machine Technical Summary
Texas Instruments**

Nu Machine SDU Monitor User's Manual

TI-2242811-0001

November, 1983

**Distributed by LMI 6033 W. Century Blvd. Los Angeles CA 90045
USA**

Copyright © 1983 Texas Instruments All rights reserved.

The information and/or drawings set forth in this document and all rights in and to inventions disclosed herein and patents which might be granted therein disclosing or employing the materials, methods, techniques of apparatus described herein, are the exclusive property of Texas Instruments Incorporated.

INTRODUCTION TO SDU MONITOR SOFTWARE

This manual describes the software used by the SDU. It describes the commands and the diagnostics available both internally in prom and externally on the disk. Also included are sections describing routines useful in writing new diagnostics or other software.

The volume is divided into six sections:

1. Commands
2. System calls
3. Subroutines
4. Devices
5. File formats and conventions
6. Diagnostics

Commands are the programs other than diagnostics that can be run on the SDU. Sections 2 and 3 are useful in writing new programs for the SDU. These sections describe routines contained in libraries that can be linked in with the programmer's new code.

Section 4 describes "devices" which can be "opened" for reading, writing, or IO control. These may be devices such as disk or tape, or they may be parts of the SDU's multibus address space.

Section 5 describes the formats for certain files such as the configuration header.

Section 6 describes the diagnostics available for the multibus IO boards and for the nubus cards.

NAME

addressing - read/write nubus and multibus addresses

SYNOPSIS

r [-b] [-w] [-l] [[first-last] [first,count]] ...

w [-b] [-w] [-l] [[first-last] [first,count]] data

x [-b] [-w] [-l] [[first-last] [first,count]]

DESCRIPTION

The r command prints out the contents of the specified addresses. The w command writes data as the contents of each address in the range. The x (examine) command interactively prints the current contents of each address and waits for new data to be entered. A carriage return without entering new data leaves the old contents unchanged.

Data size options:

-b The data size is bytes.

-w The data size is words (shorts).

-l The data size is longs.

The data size determines the size of object being addressed. The last used data size is the default in subsequent addressing commands unless a new data size is specified.

An address range consists of one or more consecutive addresses. The address increment is implied by the current data size. An address range is specified by giving the first address first and either the last address last or the number of addresses count. If neither the last address nor the number of addresses is given, the address range consists of the single address specified.

All addresses and data are printed in hexadecimal and must be entered in hexadecimal. Data and nubus addresses are specified by from 1 to 8 digits.

Multibus addresses are given in the format

segment:offset

where the segment and offset contain from 1 to 4 digits.

Multibus ports (I/O addresses) are given in the format

phh

where hh is the one or two digit I/O address.

Relative addressing is possible. When an address range is specified, the first address may be referenced by the symbol "." for subsequent use. This symbol may be used as the first address in a subsequent address range specification, (even on the same line). The symbols "+" and "-" may also be used as the first address in an address range. They refer respectively to the address following and the one preceding the . address. It is important to remember that each time a new address range is specified a new current starting address is saved.

Variables consisting of a single printable character may be defined and used as the first address in an address range (see equals(1)). The variable is used in the format \$x where x is the printable character.

SEE ALSO
equals(1)

NAME

cd - change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

directory becomes the new default directory. If no directory is given, the default working directory "/disk/monitor/bin" is used.

Because a new process is created to execute each command, cd would be ineffective if it were written as a normal command. It is therefore recognized and executed by the command monitor itself.

SEE ALSO

pwd(1)

BUGS

Any string is accepted as an argument, whether or not it is a legal directory.

NAME

clock - set the nubus clock

SYNOPSIS

clock [-n] [-s] [-f]

DESCRIPTION

clock sets the nubus clock rate (normal, slow, or fast).

The options are:

n Normal rate.

s Slow rate.

f Fast rate.

If no options are given the current setting of the nubus clock is given.

SEE ALSO

voltage(1)

NAME

conf - Print out configuration rom data by slot number.

SYNOPSIS

conf [slot...]

DESCRIPTION

Conf prints out the data contained in the configuration prom of each card. With no slot number specified, data for every slot will be printed. Otherwise only the requested slots will be printed. Slots are given as decimal numbers in the range 0 to 15.

SEE ALSO

crom(5)

BUGS

Converter cards put out their own configuration rom data on bus timeouts so empty slots appear to have converter cards in them.

conf - interactive system configuration
includes board enable/disable

NAME

copy - copy from one device to another

SYNOPSIS

copy from to [numblks]

DESCRIPTION

copy copies numblks blocks (1024-bytes each) from the device given by from to the device given by to. If numblks is not specified the copy will continue until the end-of-file is reached on either device.

The code for copy is contained in prom, and is most commonly used to back up or restore the root file system.

NAME

cp - copy from one device to another

SYNOPSIS

cp from [offset1] to [offset2] [-numblks]

DESCRIPTION

cp copies numblks blocks (1024-bytes each) from the device given by from to the device given by to. An optional block offset into either device can be specified using offset1 and offset2. If an offset is not specified, 0 is assumed. If numblks is not specified the copy will continue until the end-of-file is reached on either device.

Because cp only reads one block at a time, it can be painfully slow if one of the devices is a tape. In most cases, copy(1) will do the job much more quickly. However, cp is provided for flexibility.

NAME

date - set the SDU battery clock and print its value

SYNOPSIS

date [-dn] [YYMMDDHHMM [.SS]] [gmt]

DESCRIPTION

Date is used to initialize the battery clock on the SDU. The battery clock has three fields: the actual date, a bit saying whether daylight savings applies to this location and a field that specifies the time zone in time minutes west of Greenwich. The date is specified as in the UNIX date command as year, month, day, hour, minute, and optionally, seconds. Only the least significant digits ie minutes field, need be supplied, the higher order digits will be unchanged from the current time. Minutes west of Greenwich would be 480 for pacific time, 300 for eastern time. The daylight saving flag says whether daylight savings applies to this region at all, not whether it is currently in effect.

Option flags:

- d Daylight savings applies.
- n Daylight savings does not apply.

With no arguments, date prints the current date and the value of the other parameters.

NAME

dev - list the current sdu drivers

SYNOPSIS

dev

DESCRIPTION

dev gives a list of the drivers currently present in the system.

SEE ALSO

driver(1)

NAME

driver - load device drivers

SYNOPSIS

driver [-O offset] [-r] name [file]

DESCRIPTION

Driver loads device drivers into the diagnostic monitor, normally from the disk. Many device drivers are not built into the diagnostic monitor, so they must be loaded before they are used. Driver images exist as load modules that are similar to command load modules. When the image is loaded, memory is permanently allocated for the driver load module. Loading device drivers reduces the amount of memory available for running monitor programs.

With one argument, the name, driver looks on the disk in the standard driver directory to find the driver load module. It assumes the load module file name is the same as the driver name. If a second argument is supplied, driver interprets it as the name of the load module file name which may be on the disk or any other device.

Option flags:

O offset The offset is static parameter supplied to the driver when it is loaded. Its meaning depends on the driver, some do not use it at all. A typical use is to specify the starting block number for the disk driver. In these circumstances, the same driver load module may be loaded under several names.

r The r option removes the

FILES

/disk/monitor/drv/* Default directory for devices.

SEE ALSO

dev(1), setdr(2)

NAME

enable - enable or disable the multibus and/or nubus

SYNOPSIS

enable [[-] mnx]

DESCRIPTION

enable enables or disables the multibus and nubus. The options are:

m Enable or disable the multibus.

n Enable or disable the nubus.

x If this option is given the specified bus or busses are disabled; otherwise they are enabled.

If no options are given both busses are enabled, if only the x option is given both busses are disabled.

SEE ALSO

reset(1), init(1)

NAME

equals - assign or read an addressing variable

SYNOPSIS

= [variable [value]]

DESCRIPTION

A variable is a single printable character which may be used to specify the starting address in an addressing range for the r, w, and x commands (see addressing(1)). After a variable has been assigned a value, it is used in the format \$x where x is the printable character.

value is the 1 to 8 hexadecimal digit address that is to be assigned to the variable. If variable and value are both given then the value is assigned to the variable. If only variable is given, the value last assigned to that variable is given. If no argument is given, all of the currently assigned variables and their values are given.

SEE ALSO

addressing(1)

NAME

`fsck` - file system consistency check and interactive repair

SYNOPSIS

`fsck` [`-y`] [`-n`] [`-sX`] [`-SX`] [`-t filename`] [`filesystem`] ...

DESCRIPTION

`Fsck` audits and interactively repairs inconsistent conditions for UNIX file systems. If the file system is consistent then the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**.

`Fsck` has more consistency checks than its predecessors `check`, `dcheck`, `fcheck`, and `icheck` combined.

The following flags are interpreted by `fsck`.

- `-y` Assume a yes response to all questions asked by `fsck`.
- `-n` Assume a no response to all questions asked by `fsck`; do not open the file system for writing.
- `-sX` Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super-block of the file system.

The `-sX` option allows for creating an optimal free-list organization. `-SX` Conditionally reconstruct the free list. This option is like `-sX` above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using `-S` will force a no response to all questions asked by `fsck`. This option is useful for forcing free list reorganization on uncontaminated file systems.

- `-t` If `fsck` cannot obtain enough memory to keep its tables, it uses a scratch file. If the `-t` option is specified, the file named in the next argument is used as the scratch file, if needed. Without the `-t` flag, `fsck` will prompt the operator for the name of the scratch file. The file chosen should not be on the filesystem being checked, and if it is not a special file or did not already exist, it is removed when `fsck` completes.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
 - Incorrect number of blocks.
 - Directory size not 16-byte aligned.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
 - File pointing to unallocated inode.
 - Inode number out of range.
8. Super Block checks:
 - More than 65536 inodes.
 - More blocks for inodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the **lost+found** directory. The name assigned is the inode number. The only restriction is that the directory **lost+found** must preexist in the root of the filesystem being checked and must have empty slots in which entries can be made. This is accomplished by making **lost+found**, copying a number of files to the directory, and then removing them (before fsck is executed).

Checking the raw device is almost always faster.

DIAGNOSTICS

The diagnostics produced by fsck are intended to be self-explanatory.

BUGS

Inode numbers for **.** and **..** in each directory should be checked for validity.

-g and **-b** options from check should be available in fsck.

Fsck should understand about quotas.

NAME

`fsck` - file system consistency check and interactive repair

SYNOPSIS

`fsck [-y] [-n] [-sX] [-SX] [-t filename] [filesystem] ...`

DESCRIPTION

Fsck audits and interactively repairs inconsistent conditions for UNIX file systems. If the file system is consistent then the number of files, number of blocks used, and number of blocks free are reported. If the file system is inconsistent the operator is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions will result in some loss of data. The amount and severity of data lost may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond **yes** or **no**.

Fsck has more consistency checks than its predecessors check, dcheck, fcheck, and icheck combined.

The following flags are interpreted by fsck.

- y** Assume a yes response to all questions asked by fsck.
- n** Assume a no response to all questions asked by fsck; do not open the file system for writing.
- sX** Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super-block of the file system.

The **-sX** option allows for creating an optimal free-list organization. **-SX** Conditionally reconstruct the free list. This option is like **-sX** above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using **-S** will force a no response to all questions asked by fsck. This option is useful for forcing free list reorganization on uncontaminated file systems.

- t** If fsck cannot obtain enough memory to keep its tables, it uses a scratch file. If the **-t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **-t** flag, fsck will prompt the operator for the name of the scratch file. The file chosen should not be on the filesystem being checked, and if it is not a special file or did not already exist, it is removed when fsck completes.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one inode or the free list.
2. Blocks claimed by an inode or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
 - Incorrect number of blocks.
 - Directory size not 16-byte aligned.
5. Bad inode format.
6. Blocks not accounted for anywhere.
7. Directory checks:
 - File pointing to unallocated inode.
 - Inode number out of range.
8. Super Block checks:
 - More than 65536 inodes.
 - More blocks for inodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free inode count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the operator's concurrence, reconnected by placing them in the `lost+found` directory. The name assigned is the inode number. The only restriction is that the directory `lost+found` must preexist in the root of the filesystem being checked and must have empty slots in which entries can be made. This is accomplished by making `lost+found`, copying a number of files to the directory, and then removing them (before `fsck` is executed).

Checking the raw device is almost always faster.

DIAGNOSTICS

The diagnostics produced by `fsck` are intended to be self-explanatory.

BUGS

Inode numbers for `.` and `..` in each directory should be checked for validity.

`-g` and `-b` options from `check` should be available in `fsck`.

`Fsck` should understand about quotas.

NAME

help - help in using monitor commands

SYNOPSIS

help [cmd ...]

DESCRIPTION

help prints information on the usage of the specified internal monitor commands. If no commands are specified, help is given for all monitor commands.

NAME

init - initialize system

SYNOPSIS

init

DESCRIPTION

init resets the multibus and the nu system. It also enables both the nubus and the multibus, and jumps to the beginning of the sdu monitor's code. It is useful in resetting the system, e.g., when "out of memory" occurs when trying to run a program.

SEE ALSO

enable(1), reset(1)

NAME

pmd - post mortem dump

SYNOPSIS

pmd [-adtv] [-C conf] [dumpfile]

DESCRIPTION

Pmd is a utility which will save the contents of CPU and NuBus RAM. Pmd is normally used to save the state of a crashed system for later analysis. The dump file contains a header block (as defined by pmdump.h), followed by CPU cache contents, followed by all NuBus RAM contents. The header block includes sizes of various portions of the dump file, CPU control register contents, system configuration, and a checkword.

The dumpfile can be a disk file, 1/2" tape (/tape), or 1/4" tape (/quart). If dumpfile is a tape, then the -t option must be supplied. If dumpfile is a disk file, then it must already exist in the root file system (/disk) and must be large enough to hold all RAM plus 24K bytes. Most root file systems are not large enough to also contain a dump file so tape is the most common dump medium.

Option flags:

- a Automatic configuration. Contents of CMOS RAM and configuration ROMs on NuBus cards are used to determine system configuration.
- d Print debugging messages.
- t Dump device is tape (also changes default dumpfile to /tape).
- v Causes the dump to be verified (recommended).

Options which expect arguments after the option letter:

- C conf Use file conf as the configuration file that describes which slots contain what boards and other configuration data.

FILES

- /disk/monitor/conf/std Default configuration file
- /disk/pmdump Default dumpfile.

NAME

`pwd` - working directory name

SYNOPSIS

`pwd`

DESCRIPTION

`pwd` prints the pathname of the working (current) directory. Unless changed, the default working directory is `"/disk/monitor/bin"`. The name of the current directory is prepended to any file name used by the sdu monitor that does not begin with a slash.

SEE ALSO

`cd(1)`, `fileSYS(4)`

NAME

reset - reset busses/system

SYNOPSIS

reset [[-] bmn]

DESCRIPTION

reset resets the nubus (short reset), multibus, or nu system. The options are:

- b** Reset the nubus. This is a short reset of the nubus only and does not affect any registers.
- m** Reset the multibus.
- n** Do a nu system reset. This is a hard reset which changes the state of certain registers. If this option is chosen the b option is not needed.

If no options are given both multibus and nu system resets are done.

SEE ALSO

enable(1), init(1)

BUGS

A reset of the NuBus will halt the sdu operating system if it is executed using the rsd device as the command console.

NAME

setup - setup the cmos configuraion ram

SYNOPSIS

setup [clear] [sp] [micro [num]] [eagle [num]] [shell]

setup dskpart num name byteoffset size

setup shell switchnum byteoffset bytesize

DESCRIPTION

setup is used to read or modify the contents of the system configuration ram. Without any arguments setup gives a verbose description of what the configuration is.

In the first usage above, one or more arguments may be passed, but the order is important, since the arguments are handled sequentially.

clear erases all of the configuration information.

sp sets the defaults for the serial ports (e.g. 9600 baud).

micro sets the defaults for systems with the 80 megabyte disk.

eagle sets the defaults for systems with the eagle disk.

A number may be entered after either micro or eagle to specify a non-default interleave factor for disk formatting.

shell sets default limits for the location and size available for the shell script corresponding to a particular switch setting.

The second usage is to allow the allocation of part of the disk for a specific purpose.

The third usage is to allow setting specific parameters for shell scripts corresponding to the switch setting. This usage is dangerous and should be used with great care.

When first initializing the configuration data, a typical sequence (for an eagle disk system) is:

dl setup clear

dl setup sp eagle shell

BUGS

The clear option can not be used with any other option. The

format for using setup is likely to change at a future date.

NAME

ttyset - change the sdu's monitor device

SYNOPSIS

ttyset device

DESCRIPTION

This command determines what device is to be the sdu monitor (e.g. for running diagnostics). device may typically be either "keytty", "ttya", or "ttyb". keytty is the graphics monitor. ttya is the remote serial port on the sdu. ttyb is the local serial port on the sdu. The names ttya and ttyb are used only by ttyset. The actual name used for I/O is still tty, so that the command "ttyset tty" will use the last previously used serial port.

NAME

uboot - Nu Machine UNIX[^] bootstrap loader

SYNOPSIS

uboot [-adnctpiT] [-D dev] [-C conf] [-k console] [sys]

DESCRIPTION

Uboot bootstraps UNIX from the SDU to the 68000 processor. It then establishes a message handler to process messages coming from UNIX running on the 68000. If sys is specified, it will bootstrap that file rather than the default, /uroot/unix.

Option flags:

- a Automatic configuration. Contents of CMOS RAM and configuration ROMs on NuBus cards are used to determine system configuration.
- d Print debugging messages.
- n Do not start processor after booting.
- c Do not enable the cache on the selected processor.
- t Print timing information when program exits.
- p Start primary rather than secondary cpu.
- i Use interrupt handling on messages (normally 'busywaits')
- N Disable the clock.

Options which expect arguments after the option letter:

- D dev Use file dev as the message handling device.
- C conf Use file conf as the configuration file that describes which slots contain what boards and other configuration data.

The k option selects the UNIX console. This option expects a console keyword to follow. If the keyword is not recognized or the k option is not specified, then the UNIX console defaults to SDU serial port A (remote).

- k ttya Use SDU serial port A as the UNIX console.
- k rsd Use the raster scan monitor as the UNIX console.

Several device drivers are used by uboot including the raw disk driver, and the iomsg driver. The disk driver must be loaded before running uboot. Normally, if uboot is run from the disk rather than by downloading, the monitor will have loaded the disk driver for you. Uboot will automatically load the iomsg driver using a default name unless it is overridden with the -D option.

FILES.

/disk/monitor/drv/ioport Default ioport device
/disk/monitor/conf/std Default configuration file
/uroot/unix Default Nu Machine UNIX

SEE ALSO

fileSYS(4)

BUGS

Uboot tends to change frequently hence some of the option flags may not work and others may be added.

You can stop uboot by typing control C to the monitor but unix will not bootstrap properly again unless you reset because uboot steals the clock from the monitor.

^UNIX is a trademark of Bell Laboratories, Incorporated.

NAME

voltage - set voltage margining

SYNOPSIS

voltage [-n] [-l] [-h]

DESCRIPTION

voltage margins the nubus voltage (normal, low, or high).
The options are:

n Normal voltage.

l Low voltage.

h High voltage.

If no options are given the current setting of the nubus voltage is given.

SEE ALSO

clock(1)

NAME

alarm - schedule signal after specified time

SYNOPSIS

```
alarm(ticks)
long ticks;
```

DESCRIPTION

alarm causes signal SIGALARM, see signal(2), to be sent to the invoking process in the number of sdu clock ticks given by the argument. The clock is programmed to interrupt at 50 millisecond intervals. The default action that a process takes, if SIGALARM is not caught, is to ignore the signal.

Alarm requests are not stacked; successive calls reset the alarm clock. If the argument is 0L, any alarm request is cancelled. Because the clock has a 50-millisecond resolution, the signal may occur up to 50 milliseconds early; because of scheduling delays, resumption of execution of when the signal is caught may be delayed an arbitrary amount.

SEE ALSO

pause(2), signal(2), sleep(2), delay(3) The 50-millisecond clock resolution may change at a future date.

NAME

close - close a file

SYNOPSIS

short close(fildes)
short fildes;

DESCRIPTION

Given a file descriptor returned from an open(3) call, close closes the associated file. A close of all files is automatic on exit(2), but since there is a limit on the number of open files per process, close is necessary for programs which deal with many files.

Files are automatically closed upon termination of a process.

SEE ALSO

open(3), mopen(2)

DIAGNOSTICS

-1 is returned for an unknown file descriptor.

NAME

execv - change the code and data for the process

SYNOPSIS

```
short execv(name, argv)
char *name;
char **argv;
```

DESCRIPTION

execv replaces the code and data space of the process with the new code and data given in the file name.

When a C program is executed, it is called as follows:

```
programe(argc, argv)
short argc;
char **argv;
```

where programe is the name of the start of the code, argc is the argument count, and argv is an array of character pointers to the arguments themselves. A null pointer must end the array. argc is conventionally at least one and the first member of the array points to a string containing the name of the file.

SEE ALSO

runv(2)

DIAGNOSTICS

If the file cannot be found, if it does not start with a valid magic number (see a.out(5)), or if maximum memory is exceeded, the return value is -1. Otherwise there is no return.

NAME

exit - terminate process

SYNOPSIS

```
exit(status)
short status;
```

```
exit(status)
short status;
```

DESCRIPTION

exit is the normal means of terminating a process. exit closes all the process's files and notifies the parent process if it is executing a wait. status is made available to the parent process.

This call can never return.

exit may cause cleanup actions before the final "system exit". The function exit circumvents all cleanup.

SEE ALSO

wait(2), fclose(3)

NAME

ioctl - control device

SYNOPSIS

```
short ioctl(fildes, request, argp)
short fildes;
short request;
char *argp;
```

DESCRIPTION

The file descriptor fildes is a value returned from a successful open call. The function performed by ioctl and the significance of the arguments request and argp depend on the particular device that was opened.

SEE ALSO

open(3), read(3), write(3), seek(2), close(2)

DIAGNOSTICS

If the ioctl call was unsuccessful or if fildes did not refer to a previously opened device, the return value is -1; the call returns 0 if it was successful.

NAME

iomov - do I/O on an opened file

SYNOPSIS

```
short iomov(mode, fildes, buffer, nbytes)
short mode;
short fildes;
char *buffer;
short nbytes;
```

DESCRIPTION

mode is either 1 for reading from the file or 2 for writing to the file. The file descriptor fildes is a value returned from a successful open call. buffer is the location of nbytes contiguous bytes into which the input will be placed. The return value is the number of characters read or written.

It is not guaranteed that all nbytes bytes will be read if using the read mode; for example, if the file refers to the tty at most one line will be returned. If the returned value when reading is 0, then end-of-file has been reached.

The subroutines read(3) and write(3) which call iomov, should be used instead of iomov.

SEE ALSO

open(3), ioctl(2), seek(2), close(2), read(3), write(3)

DIAGNOSTICS

If the iomov was unsuccessful the return value is -1. Conditions that can generate an error: physical I/O errors, file descriptor not that of an opened file, or a read or write attempted when the file was not opened for that mode.

NAME

kill - send signal to a process

SYNOPSIS

```
short kill(pid, sig);
short pid, sig;
```

DESCRIPTION

kill sends the signal(s) sig to the process specified by the process number pid. See signal(2) for a list of signals.

If the process number is \emptyset , the signal is sent to all processes which have opened the device tty, possibly including the calling process itself; see tty(4).

Processes may send signals to themselves.

SEE ALSO

signal(2)

DIAGNOSTICS

Zero is returned if the process is killed; -1 is returned if the process does not exist.

NAME

malloc - memory allocator

SYNOPSIS

```
char *malloc(size)
      unsigned short size;
```

```
mfree(ptr)
char *ptr;
```

DESCRIPTION

malloc and mfree provide simple general-purpose memory routines. malloc returns a pointer to a block of at least size bytes beginning on a 16-byte boundary.

The argument to mfree is a pointer to a block previously allocated by malloc; this space is made available for further allocation, but its contents are left undisturbed.

Needless to say, grave disorder will result if the space assigned by malloc is overrun or if some random number is handed to mfree.

DIAGNOSTICS

malloc returns a null pointer ((char *) 0) if there is no available memory.

NAME

mopen - open for reading or writing

SYNOPSIS

```
short mopen(name, mode)
char *name;
short mode;
```

DESCRIPTION

mopen opens the file name for reading (if mode is 1), writing (if mode is 2) or for both reading and writing (if mode is 3). name is the address of a string of ASCII characters representing a path name, terminated by a null character.

The file is positioned at the beginning (byte 0). The returned file descriptor must be used for subsequent calls for other input-output functions on the file.

The subroutine open(3), which calls mopen, should be used instead of mopen.

SEE ALSO

open(3), read(3), write(3), ioctl(2), seek(2), close(2)

DIAGNOSTICS

The value -1 is returned if the file does not exist, if one of the necessary directories does not exist, or if too many files are open.

BUGS

This call should probably be redesigned to look exactly like open(3).

NAME

passint - pass along an interrupt from the SDU

SYNOPSIS

```
#include /usr/monitor/h/interrupt.h
```

```
short passint(n, ptr)
```

```
short n;
```

```
char *ptr;
```

DESCRIPTION

passint installs an interrupt handler on the SDU which writes a 1 to the multibus address specified by ptr. This call is most often useful if the multibus address has been previously mapped into a NuBus address through the numapp call, thereby providing a general interrupt mechanism for the NuBus.

There are three pics on the sdu. The interrupt lines are numbered from 0 to 23, going from line 0 on pic 0 to line 7 on pic 2. The include file gives defines for the various interrupts.

If ptr is null, the specified pic interrupt is disabled.

DIAGNOSTICS

If the interrupt number is larger than the maximum number of pic interrupts, the return value is -1. Otherwise 0 is returned to indicate success.

BUGS

The program or driver that calls passint may go away without resetting the interrupt.

NAME

runv - create a new process

SYNOPSIS

```
short runv(name, argv, mode)
char *name;
char **argv;
short mode;
```

DESCRIPTION

runv creates a new process by allocating code, data, and stack space for the file name. mode is 1 if the new process is to be traced, 2 if tty is to be opened for standard input, standard output, and standard error output, 3 if both options are to be done, or 0 if neither tracing nor opening the tty is to be done.

When a C program is executed, it is called as follows:

```
progname(argc, argv)
short argc;
char **argv;
```

where progname is the name of the start of the code, argc is the argument count, and argv is an array of character pointers to the arguments themselves. A null pointer must end the array. argc is conventionally at least one and the first member of the array points to a string containing the name of the file.

SEE ALSO

execv(2)

DIAGNOSTICS

If no more processes can be created, if the file cannot be found, if it does not start with a valid magic number (see a.out(5)), or if maximum memory is exceeded, the return value is -1. Otherwise the return is the process ID of the new process.

NAME

seek - move read/write pointer

SYNOPSIS

```
short seek(fildes, offset)
short fildes;
long offset;
```

DESCRIPTION

The file descriptor refers to a file opened for reading or writing. The read/write pointer for the file is set to offset bytes from the beginning of the file.

SEE ALSO

open(3), read(3), write(3)

DIAGNOSTICS

-1 is returned for an undefined file descriptor.

BUGS

seek is a no-op on device tty. seek should return the value of the last offset.

NAME

setdr - install a driver

SYNOPSIS

```
short setdr(drname, fname, ptr)
char *drname, *fname;
char *ptr;
```

DESCRIPTION

setdr creates a driver drname by allocating memory and loading code and data from fname. ptr is a value that can be used or ignored by the driver as it wishes. If the driver drname already exists, the old code and data is replaced. If the file name fname is null, the driver is released and no longer can be opened.

SEE ALSO

open(3)

DIAGNOSTICS

If the file cannot be found, if it does not start with a valid magic number (see a.out(5)), if maximum memory is exceeded, or if the maximum number of devices is exceeded, the return value is -1. Otherwise 0 is returned to indicate success.

NAME

setint - install an interrupt handler

SYNOPSIS

```
#include /usr/monitor/h/interrupt.h
```

```
short setint(n, func, ds)
short n;
(*func)();
short ds;
```

DESCRIPTION

setint installs an interrupt handler given by the function func and the data segment ds for the pic interrupt whose number is n. There are three pics on the sdu. The interrupt lines are numbered from 0 to 23, going from line 0 on pic 0 to line 7 on pic 2. The include file gives defines for the various interrupts.

If func is null, the specified pic interrupt is disabled.

DIAGNOSTICS

If the interrupt number is larger than the maximum number of pic interrupts, the return value is -1. Otherwise 0 is returned to indicate success.

BUGS

The program or driver that calls setint may go away without resetting the interrupt.

NAME

signal - catch one or more signals

SYNOPSIS

```
#include /usr/monitor/h/signal.h
```

```
(*signal(signals, func))()
short signals;
(*func)();
```

DESCRIPTION

A signal is generated by some abnormal event, initiated either by the user at the tty (interrupt), or by a request of another program (kill). Normally the signal SIGALARM is ignored and all the other signals cause termination of the receiving process, but a signal call allows one or more signals to cause an interrupt to a specified location. All of the signals that are candidates to be caught must be "or"ed into the signals argument. Here is the list of signals with names as in the include file.

SIGALARM	0x1	alarm from clock has occurred
SIGINTR	0x2	interrupt (control-C) has been sent
SIGDIVERR	0x10	program got a divide error
SIGNOMASK	0x20	program got a non-maskable interrupt
SIGOVFLOW	0x40	program got an overflow error
SIGTIMOUT	0x80	program got a bus error

When one or more of the signals trapped for occurs func will be called with the signal numbers "or"ed as argument. A return from the function will continue the process at the point it was interrupted. Once, the function has run, the signals are reset so that if it is desired to catch any subsequent signal, the catching routine must issue another signal call.

If a signal occurs which was not trapped for, the default action is taken, This is to ignore the signal for SIGALARM and to terminate the process otherwise.

When a caught signal occurs during certain system calls, the call terminates prematurely. Specifically this can occur during a sleep(2) or any system call that itself calls the sleep system call, such as pause, wait, or iomov(2) for interrupt-driven device drivers. When such a signal occurs, the saved user status is arranged in such a way that when return from the signal-catching takes place, it will appear that the system call did not complete properly. The user's program may then, if it wishes, re-execute the call.

SEE ALSO

kill(2), sleep(2)

NAME

sleep, wakeup - event synchronization

SYNOPSIS

short sleep(event)
char *event;

wakeup(event)
char *event;

DESCRIPTION

sleep and wakeup provide a simple event synchronization mechanism. An event is typically described by a memory address associated with that event, such as a buffer address used by a device driver waiting for an interrupt. sleep stops a process from running until either the specified event occurs, as indicated by wakeup, or a signal is sent to the process.

wakeup signifies the occurrence of an event to all processes simply by using the event as an argument.

DIAGNOSTICS

sleep returns the value of the signal received or 0 if the return was due to a wakeup.

NAME

wait - wait for process to terminate

SYNOPSIS

```
short wait(status)
short *status;
```

DESCRIPTION

wait causes its caller to delay until a signal is received or one of its child processes terminates. If there are no children, the value 0 is returned immediately. The normal return yields the process ID of the terminated child. In the case of several children several wait calls are needed to learn of all the deaths.

status receives the termination status of the process. See signal(2) for a list of termination statuses (signals); 0 status indicates normal termination.

SEE ALSO

exit(2), signal(2)

DIAGNOSTICS

Returns 0 if there are no children. Returns -1 if the return was because a signal was received.

NAME

atoi, atol - convert ASCII to numbers

SYNOPSIS

```
atoi(nptr)
char *nptr;
```

```
long atol(nptr)
char *nptr;
```

DESCRIPTION

These functions convert a string pointed to by nptr to integer, and long integer representation respectively. The first unrecognized character ends the string.

Atoi and atol recognize an optional string of tabs and spaces, then an optional sign, then a string of digits.

SEE ALSO

scanf(3)

BUGS

There are no provisions for overflow.

NAME

bitcheck - check a bit field

SYNOPSIS

```
bitcheck(ptr, name, start, width)
union {
    unsigned char *cptr;
    unsigned short *sptr;
    unsigned long *lptr;
} ptr;
char *name;
int start, width;
```

DESCRIPTION

bitcheck writes walking ones and walking zeroes into the bits specified and checks the bits written. start is the starting bit position in the field, with 0 the lowest bit, and width is the width of the bit field. The use of the union passed depends on the sum of start and width. The sum must be less than 32 since the maximum field width is 32. If the sum is more than 16 a pointer to an unsigned long is used. If the sum is between 8 and 16 a pointer to an unsigned short is used. Otherwise ptr points to an unsigned char.

name is a string printed out with the error message if a bit error occurs.

SEE ALSO

mem(3), movi(3), movimask(3)

NAME

blt - high speed byte mover

SYNOPSIS

blt(from, to, nbytes)
char *from, *to;

DESCRIPTION

Blt moves bytes from memory to memory using the most efficient instruction sequence.

NAME

configure - print static configuration information

SYNOPSIS

```
configure(s,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10)
char *s;
```

DESCRIPTION

This routine is like printf except that it only prints if the configuration flag is set. The line output will be in a standard form. The argument s is a printf style format string.

SEE ALSO

printf(3), vprintf(3), diag(3)

NAME

isalpha, isupper, islower, isdigit, isalnum, isspace, ispunct, isprint, iscntrl, isascii - character classification

SYNOPSIS

```
#include <ctype.h>
```

```
isalpha(c)
```

```
...
```

DESCRIPTION

These macros classify ASCII-coded integer values by table lookup. Each is a predicate returning nonzero for true, zero for false. Isascii is defined on all integer values; the rest are defined only where isascii is true and on the single non-ASCII value EOF (see stdio(3)).

<u>isalpha</u>	<u>c</u> is a letter
<u>isupper</u>	<u>c</u> is an upper case letter
<u>islower</u>	<u>c</u> is a lower case letter
<u>isdigit</u>	<u>c</u> is a digit
<u>isalnum</u>	<u>c</u> is an alphanumeric character
<u>isspace</u>	<u>c</u> is a space, tab, carriage return, newline, or formfeed
<u>ispunct</u>	<u>c</u> is a punctuation character (neither control nor alphanumeric)
<u>isprint</u>	<u>c</u> is a printing character, code 040(8) (space) through 0176 (tilde)
<u>iscntrl</u>	<u>c</u> is a delete character (0177) or ordinary control character (less than 040).
<u>isascii</u>	<u>c</u> is an ASCII character, code less than 0200

SEE ALSO

ascii(7)

NAME

delay - delay a specified time

SYNOPSIS

```
delay(ticks)
long ticks;
```

DESCRIPTION

delay causes a delay of ticks time intervals. The basic time interval is 50 milliseconds. The effect of the call is to allow a certain amount of time to pass before the call returns. The actual time delayed may be up to one interval less or an arbitrary time later because of scheduling delays.

SEE ALSO

alarm(2), pause(2), sleep(2)

BUGS

delay uses the alarm(2) system call. The 50-millisecond clock resolution may change at a future date.

NAME

diag - routines for diagnostics support

SYNOPSIS

```
#include <diag.h>
```

DESCRIPTION

The standard way of creating a diagnostic program to take advantage of the diagnostic library support routines is to create individual test routines and arrays of structures relating to the tests. Each test routine performs a specific test and is independent of other test routines.

More specifically, the arrays of structures and the data that must be created are given below. t select is encoded according to the '#define' statements.

```
int ntest;
```

```
struct test
{
```

```
    char *t_name;           /* test name */
    int t_select;          /* bits indicating test selected */
    int (*t_fnc)();        /* test function */
    int t_arg1;           /* arguments passed to test function */
    int t_arg2;
    int t_arg3;
    int t_arg4;
} tsttbl[];
```

```
struct tdata
{
```

```
    struct tdata *td_next; /* pointer to next test to run */
    unsigned td_errcnt; /* number of errors on this test */
} testdata[];
```

```
char *cmdname;
char *version;
char *errors[];
```

```
#define T_DIRTY 1 /* this is a dirty test */
#define T_EXPLICIT 2 /* run selected tests explicitly */
#define T_STANDARD 4 /* this test is part of the standard set */
#define T_CONFIG 8 /* test that finds configuration info */
```

ntest is the number of tests in the diagnostic.

The tsttbl array must contain ntest structures, one for each test. t_name is a short description of a test routine. t_select describes under what circumstances the test will run. T_DIRTY means that the test runs only if the "dirty" flag D is given in the command line when the diagnostic is

run, see diag(6). T EXPLICIT is used internally by the library routines. If the T STANDARD bit is set then the test will run by default if no explicit tests are specified in the command line. T CONFIG means that the test runs if the configuration flag c is specified. t fnc is the name of the test routine, and t arg1 etc. are arguments that are passed to the test.

The testdata array must contain n_{test} structures, one for each test. This array is used by the library routines to link together the tests to be done in a specific running of the diagnostic. It is the only data not initialized by the diagnostic writer.

The routine main, which is standardly the first routine called in a C program, is the routine in the diagnostic library which performs various services and calls the test routines.

There are other features of the diagnostic library which may be used by a diagnostic program, some of which depend on the arguments in the command line, see diag(6). The main routine processes the arguments passed, calls the routine startup() supplied by the user, runs the test programs as specified by the arguments, and calls the routine cleanup(), also supplied by the user. startup, options, and cleanup default to null routines if not specified by the user. startup can be used to do any initializations that should be done before running the tests; cleanup to do any cleanup work after the tests have finished. options is called for each non-standard command line flag found. For these flags to be handled an options routine should be written as one of the diagnostic program's routines. The routine is passed three arguments:

```
options(argc, argv, flagp)
short argc;
char **argv;
char *flagp;
```

The first two arguments, argc and argv are just like those in a normal C main routine. The third one flagp is a character pointer to the command line's current flag (without the dash). A global variable arg indicates the index in argv to the current argument. If the option processor wants additional information such as a numeric argument following the flag it should use argv[+arg] but only if arg is less than argc.

cmdname is the diagnostic writer's name for the program. version is a string which should give a date and version number to the current version, printed out if the V command

line flag is specified. errors is an array of error messages. The error routine, see error(3), will print out the indexed error message. There are various library routines whose behavior depends upon the flags given in the command line.

FILES

/usr/monitor/diag/liblddiag/liblddiag.a

SEE ALSO

diag(6), vprintf(3), error(3), configure(3)

NAME

end, etext, edata - last locations in program

SYNOPSIS

```
extern end;  
extern etext;  
extern edata;
```

DESCRIPTION

These names refer neither to routines nor to locations with interesting contents. The address of etext is the first address above the program text, edata above the initialized data region, and end above the uninitialized data region.

NAME

error - print an error message in standard form

SYNOPSIS

error(errnum,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10)

DESCRIPTION

This routine is like printf except that it takes its format string from a table of standard error messages. The first argument, errnum, specifies the index into an array of character strings that will be the format string and also be a description of the error. When the error string is printed, some additional information such as the test name is also printed on the error line.

SEE ALSO

printf(3)

NAME

fclose, fflush - close or flush a stream

SYNOPSIS

```
#include <stdio.h>
```

```
fclose(stream)  
FILE *stream;
```

```
fflush(stream)  
FILE *stream;
```

DESCRIPTION

Fclose causes any buffers for the named stream to be emptied, and the file to be closed. Buffers allocated by the standard input/output system are freed.

Fclose is performed automatically upon calling exit(3).

Fflush causes any buffered data for the named output stream to be written to that file. The stream remains open.

SEE ALSO

close(2), fopen(3), setbuf(3)

DIAGNOSTICS

These routines return EOF if stream is not associated with an output file, or if buffered data cannot be transferred to that file.

NAME

feof, ferror, clearerr, fileno - stream status inquiries

SYNOPSIS

```
#include <stdio.h>
```

```
feof(stream)  
FILE *stream;
```

```
ferror(stream)  
FILE *stream
```

```
clearerr(stream)  
FILE *stream
```

```
fileno(stream)  
FILE *stream;
```

DESCRIPTION

Feof returns non-zero when end of file is read on the named input stream, otherwise zero.

Ferror returns non-zero when an error has occurred reading or writing the named stream, otherwise zero. Unless cleared by clearerr, the error indication lasts until the stream is closed.

Clrerr resets the error indication on the named stream.

Fileno returns the integer file descriptor associated with the stream, see open(3).

These functions are implemented as macros; they cannot be redeclared.

SEE ALSO

fopen(3), open(3)

NAME

`fill` - fill memory with byte pattern
`fillw` - fill memory with word pattern

SYNOPSIS

```
fill(memory, cnt, pattern)
char *memory;
fillw(memory, cnt, pattern)
char *memory;
```

DESCRIPTION

Fill and fillw copy a byte or word respectively to an area of memory. The cnt argument specifies the number of bytes or words to fill.

NAME

`fopen`, `freopen`, `fdopen` - open a stream

SYNOPSIS

```
#include <stdio.h>
```

```
FILE *fopen(filename, type)
char *filename, *type;
```

```
FILE *freopen(filename, type, stream)
char *filename, *type;
FILE *stream;
```

```
FILE *fdopen(fildes, type)
char *type;
```

DESCRIPTION

Fopen opens the file named by filename and associates a stream with it. Fopen returns a pointer to be used to identify the stream in subsequent operations.

Type is a character string having one of the following values:

"r" open for reading

"w" open for writing at beginning of file

"a" append: open for writing at end of file

Freopen substitutes the named file in place of the open stream. It returns the original value of stream. The original stream is closed.

Freopen is typically used to attach the preopened constant names, `stdin`, `stdout`, `stderr`, to specified files.

Fdopen associates a stream with a file descriptor obtained from open(3). The type of the stream must agree with the mode of the open file.

SEE ALSO

`open(3)`, `fclose(3)`

DIAGNOSTICS

Fopen and freopen return the pointer `NULL` if filename cannot be accessed.

BUGS

Fdopen is not portable to systems other than UNIX.

NAME

`fread`, `fwrite` - buffered binary input/output

SYNOPSIS

```
#include <stdio.h>
```

```
fread(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;
```

```
fwrite(ptr, sizeof(*ptr), nitems, stream)
FILE *stream;
```

DESCRIPTION

Fread reads, into a block beginning at ptr, nitems of data of the type of *ptr from the named input stream. It returns the number of items actually read.

Fwrite appends at most nitems of data of the type of *ptr beginning at ptr to the named output stream. It returns the number of items actually written.

SEE ALSO

`read(2)`, `write(2)`, `fopen(3)`, `getc(3)`, `putc(3)`, `gets(3)`,
`puts(3)`, `printf(3)`, `scanf(3)`

DIAGNOSTICS

Fread and fwrite return 0 upon end of file or error.

NAME

`fseek`, `rewind` - reposition a stream

SYNOPSIS

```
#include <stdio.h>

fseek(stream, offset, 0)
FILE *stream;
long offset;

rewind(stream)
```

DESCRIPTION

Fseek sets the position of the next input or output operation on the stream. The new position is at the signed distance offset bytes from the beginning of the file.

Fseek undoes any effects of ungetc(3).

Rewind(stream) is equivalent to fseek(stream, 0L, 0).

SEE ALSO

`lseek(3)`, `fopen(3)`

DIAGNOSTICS

Fseek returns -1 for improper seeks. The third parameter must be 0.

BUGS

There should be a way for determining the present file offset.

NAME

`getc`, `getchar`, `fgetc`, `getw` - get character or word from stream

SYNOPSIS

```
#include <stdio.h>
```

```
int getc(stream)
FILE *stream;
```

```
int getchar()
```

```
int fgetc(stream)
FILE *stream;
```

```
int getw(stream)
FILE *stream;
```

DESCRIPTION

Getc returns the next character from the named input stream.

Getchar() is identical to getc(stdin).

Fgetc behaves like getc, but is a genuine function, not a macro; it may be used to save object text.

Getw returns the next word from the named input stream. It returns the constant EOF upon end of file or error, but since that is a good integer value, feof and ferror(3) should be used to check the success of getw. Getw assumes no special alignment in the file.

SEE ALSO

`fopen(3)`, `putc(3)`, `gets(3)`, `scanf(3)`, `fread(3)`, `ungetc(3)`

DIAGNOSTICS

These functions return the integer constant EOF at end of file or upon read error.

A stop with message, 'Reading bad file', means an attempt has been made to read from a stream that has not been opened for reading by fopen.

BUGS

The end-of-file return from getchar is incompatible with that in UNIX editions 1-6.

Because it is implemented as a macro, getc treats a stream argument with side effects incorrectly. In particular, '`getc(*f++)`;' doesn't work sensibly.

NAME

getcs - get current control segment
getds - get current data segment
getss - get current stack segment
getes - get current extra segment

SYNOPSIS

getcs()
getds()
getss()
getes()

DESCRIPTION

These routines return the contents of a segment register as an int.

NAME

gets, fgets - get a string from a stream

SYNOPSIS

```
#include <stdio.h>
```

```
char *gets(s)
char *s;
```

```
char *fgets(s, n, stream)
char *s;
FILE *stream;
```

DESCRIPTION

Gets reads a string into s from the standard input stream stdin. The string is terminated by a newline character, which is replaced in s by a null character. Gets returns its argument.

Fgets reads n-1 characters, or up to a newline character, whichever comes first, from the stream into the string s. The last character read into s is followed by a null character. Fgets returns its first argument.

SEE ALSO

puts(3), getc(3), scanf(3), fread(3), ferror(3)

DIAGNOSTICS

Gets and fgets return the constant pointer NULL upon end of file or error.

BUGS

Gets deletes a newline, fgets keeps it, all in the name of backward compatibility.

NAME

input - perform an input operation upon a port
output - perform an output operation upon a port

SYNOPSIS

input(port)
output(data, port)

DESCRIPTION

These routines provide access to input and output instructions to perform these operations on the multibus.

NAME

intro - introduction to library functions

SYNOPSIS

```
#include <stdio.h>
```

```
#include <diag.h>
```

DESCRIPTION

This section describes functions that may be found in various libraries, other than those functions that directly invoke sdu system primitives, which are described in section 2. Functions are divided into various libraries distinguished by the section number at the top of the page:

- (3) These functions, together with those of section 2 and those marked (3S), constitute library libmc, which is automatically loaded by the C compiler cc86. The link editor ld86 searches this library under the '-lmc' option. Declarations for some of these functions may be obtained from include files indicated on the appropriate pages.
- (3D) These functions constitute the diagnostic library. The link editor searches this library under the '-lldiag' option. Declarations for these functions may be obtained from the include file <diag.h>
- (3S) These functions constitute the 'standard I/O package', see stdio(3). These functions are in the library libmc already mentioned. Declarations for these functions may be obtained from the include file <stdio.h>.
- (3X) Various specialized libraries have not been given distinctive captions. The files in which these libraries are found are named on the appropriate pages.

FILES

/usr/86lib/libmc.a /usr/86lib/libldiag.a

SEE ALSO

stdio(3), diag(3)

NAME

lock - disable interrupts
unlock - enable interrupts

SYNOPSIS

lock()
unlock()

DESCRIPTION

The routines simply perform "cli" and "sti" instructions and return.

NAME

lseek - move read/write pointer

SYNOPSIS

```
long lseek(fildes, offset, 0)
long offset;
```

DESCRIPTION

The file descriptor refers to a file open for reading or writing. The read (resp. write) pointer for the file is set to offset bytes.

SEE ALSO

open(3), fseek(3)

DIAGNOSTICS

-1 is returned for an undefined file descriptor.

BUGS

Lseek is a no-op on character special files. The third parameter must be 0.

NAME

fillchk, walk - memory tests

SYNOPSIS

```
fillchk(nuaddr, nbytes, pattern)
long nuaddr, nbytes;
int pattern;
```

```
walk(nuaddr, nbytes, ones)
long nuaddr, nbytes;
int ones;
```

DESCRIPTION

fillchk fills nbytes of NuBus memory starting at address nuaddr with the low order byte of pattern. First, all of the specified memory is written with the given pattern, then the memory is read back and verified against the pattern. The number of errors that occurred is returned.

walk also tests nbytes of NuBus memory starting at address nuaddr. walk actually runs 16 tests, each test using a 16-bit pattern consisting of all ones except for a single bit 0, if ones is 0, or a 16-bit pattern consisting of all zeroes except for a single bit 1, if ones is 1. The 16 tests vary in the bit position of the differing bit, which "walks" from the lowest to the highest bit. For each pattern, all of the specified memory is filled with the pattern two bytes at a time, and then the memory is read back to verify it against the pattern. The number of errors is returned.

NAME

movi - perform a moving inversions test

SYNOPSIS

```
movi(nuaddr, nbytes, width)
long nuaddr, nbytes;
int width;
```

DESCRIPTION

movi runs a moving inversions test on nbytes of NuBus memory starting at address nuaddr. The physical width of the memory is given by width. For example, the width is 1 for a 64K-by-1 RAM. The number of errors that occurred is returned.

MOVI marches through memory with both true and complimented data, backwards and forwards by addresses that are ascending powers of two. It takes a time of order $n \log n$. It is claimed to have a high probability of isolating cell shorts, cell opens and address uniqueness problems. It is designed as a shorter version of the more familiar Galpat test.

SEE ALSO

movimask(3), mem(3)

NAME

movimask - perform moving inversions test with a field mask

SYNOPSIS

```
movimask(nuaddr, nbytes, width, startpos, fldwidth)
long nuaddr, nbytes;
int width, startpos, fldwidth;
```

DESCRIPTION

movimask runs a moving inversions test on nbytes of NuBus memory starting at address nuaddr. The physical width of the memory is given by width. For example, the width is 1 for a 64K-by-1 RAM. fldwidth is the field width of the memory (in bits), and startpos is the starting bit position within the field. startpos is 0 to start with position in the field. The sum of startpos and fldwidth must be less than 32.

movimask is like movi except that the testing is restricted to a specified bit-field.

SEE ALSO

movi(3), mem(3)

NAME

numap, nufree, numapp, nufreep - access NuBus addresses

SYNOPSIS

```
short numap(nuaddr, size)
unsigned long nuaddr;
short size;
```

```
nufree(segment, size)
short segment, size;
```

```
char *numapp(nuaddr, size)
unsigned long nuaddr;
short size;
```

```
nufreep(ptr, size)
char *ptr;
short size;
```

DESCRIPTION

numap allocates part of the nu map to map into NuBus addresses starting with nuaddr and including size pages, a page being 1024 bytes. numap returns the multibus segment value that can be used to access the specified NuBus address range. A return of zero means the mapping of multibus to NuBus addresses could not be done. nuaddr should be a multiple of 1024.

nufree frees the size map entries used by a previous numap call that returned the value segment.

numapp is the same as numap except that a pointer to multibus memory is returned. Also, the nu address nuaddr does not have any restrictions on it. A null pointer returned means that the mapping could not be done.

nufreep is the counterpart to numapp and frees the map entries associated with the pointer ptr.

SEE ALSO

map(4)

NAME

open - open for reading or writing

SYNOPSIS

```
open(name, mode)
char *name;
```

DESCRIPTION

`open` opens the file `name` for reading (if `mode` is `0`), writing (if `mode` is `1`) or for both reading and writing (if `mode` is `2`). `name` is the address of a string of ASCII characters representing a path name, terminated by a null character.

The file is positioned at the beginning (byte `0`). The returned file descriptor must be used for subsequent calls for other input-output functions on the file.

SEE ALSO

`read(3)`, `write(3)`, `mopen(2)` `close(2)`

DIAGNOSTICS

The value `-1` is returned if the file does not exist, if one of the necessary directories does not exist or is unreadable, or if too many files are open.

NAME

short pause - stop until signal

SYNOPSIS

pause()

DESCRIPTION

Pause never returns normally. It is used to give up control while waiting for a signal from kill(2) or alarm(2). The returned value is that of the signal(s) that caused the return.

SEE ALSO

kill(2), alarm(2), signal(2), delay(3)

NAME

printf, fprintf, sprintf - formatted output conversion

SYNOPSIS

```
#include <stdio.h>
```

```
printf(format [, arg ] ... )  
char *format;
```

```
fprintf(stream, format [, arg ] ... )  
FILE *stream;  
char *format;
```

```
sprintf(s, format [, arg ] ... )  
char *s, format;
```

DESCRIPTION

Printf places output on the standard output stream stdout. Fprintf places output on the named output stream. Sprintf places 'output' in the string s, followed by the character '\0'.

Each of these functions converts, formats, and prints its arguments after the first under control of the first argument. The first argument is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive arg printf.

Each conversion specification is introduced by the character %. Following the %, there may be

- an optional minus sign '-' which specifies left adjustment of the converted value in the indicated field;
- an optional digit string specifying a field width; if the converted value has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width; if the field width begins with a zero, zero-padding will be done instead of blank-padding;
- an optional period '.' which serves to separate the field width from the next digit string;
- an optional digit string specifying a precision which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;

- the character `l` specifying that a following `d`, `o`, `x`, or `u` corresponds to a long integer arg. (A capitalized conversion code accomplishes the same thing.)
- a character which indicates the type of conversion to be applied.

A field width or precision may be `*` instead of a digit string. In this case an integer arg supplies the field width or precision.

The conversion characters and their meanings are

- d** **o** **x** The integer arg is converted to decimal, octal, or hexadecimal notation respectively.
- f** The float or double arg is converted to decimal notation in the style `'[-]ddd.ddd'` where the number of `d`'s after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly `0`, no digits and no decimal point are printed.
- e** The float or double arg is converted in the style `'[-]d.ddde+dd'` where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced.
- g** The float or double arg is printed in style `d`, in style `f`, or in style `e`, whichever gives full precision in minimum space.
- c** The character arg is printed. Null characters are ignored.
- s** Arg is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is `0` or missing all characters up to a null are printed.
- u** The unsigned integer arg is converted to decimal and printed (the result will be in the range `0` to `65535`).
- %** Print a `'%'`; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by printf are printed by putc(3).

Examples

To print a date and time in the form `Sunday, July 3, 10:02', where weekday and month are pointers to null-terminated strings:

```
printf("%s, %s %d, %02d:%02d", weekday, month, day,
       hour, min);
```

SEE ALSO

putc(3), scanf(3)

BUGS

Very wide fields (> 128 characters) fail.

NAME

ptr2off - convert pointer to long
off2ptr - convert long to pointer

SYNOPSIS

```
long ptr2off(ptr)
char *ptr;
char *off2ptr(addr)
long addr;
```

DESCRIPTION

These routines convert the C compiler representation of pointers which is a segment offset pair to sensible multibus addresses represented as a long and vice-versa. This is not the same as simply doing a pointer to long type coercion in C. Doing a pointer to long type coercion in C returns a long which contains a segment in one half of the long and and offset in the other. Segments are in units of 16 bytes rather than 2*16 bytes so longs obtained from type coercions are not suitable for general address arithmetic.

NAME

`putc`, `putchar`, `fputc`, `putw` - put character or word on a stream

SYNOPSIS

```
#include <stdio.h>
```

```
int putc(c, stream)
char c;
FILE *stream;
```

```
putchar(c)
```

```
fputc(c, stream)
FILE *stream;
```

```
putw(w, stream)
FILE *stream;
```

DESCRIPTION

`Putc` appends the character `c` to the named output `stream`. It returns the character written.

`Putchar(c)` is defined as `putc(c, stdout)`.

`Fputc` behaves like `putc`, but is a genuine function rather than a macro. It may be used to save on object text.

`Putw` appends word (i.e. int) `w` to the output `stream`. It returns the word written. `Putw` neither assumes nor causes special alignment in the file.

The standard stream `stdout` is normally buffered if and only if the output does not refer to a terminal; this default may be changed by `setbuf(3)`. The standard stream `stderr` is by default unbuffered unconditionally, but use of `freopen` (see `fopen(3)`) will cause it to become buffered; `setbuf`, again, will set the state to whatever is desired. When an output stream is unbuffered information appears on the destination file or terminal as soon as written; when it is buffered many characters are saved up and written as a block. `Fflush` (see `fclose(3)`) may be used to force the block out early.

SEE ALSO

`fopen(3)`, `fclose(3)`, `getc(3)`, `puts(3)`, `printf(3)`, `fread(3)`

DIAGNOSTICS

These functions return the constant EOF upon error. Since this is a good integer, `ferror(3)` should be used to detect `putw` errors.

BUGS

Because it is implemented as a macro, putc treats a stream argument with side effects improperly. In particular `'putc(c, *f++);'` doesn't work sensibly.

NAME

puts, fputs - put a string on a stream

SYNOPSIS

```
#include <stdio.h>
```

```
puts(s)  
char *s;
```

```
fputs(s, stream)  
char *s;  
FILE *stream;
```

DESCRIPTION

Puts copies the null-terminated string s to the standard output stream stdout and appends a newline character.

Fputs copies the null-terminated string s to the named output stream.

Neither routine copies the terminal null character.

SEE ALSO

fopen(3), gets(3), putc(3), printf(3), ferror(3)
fread(3) for fwrite

BUGS

Puts appends a newline, fputs does not, all in the name of backward compatibility.

NAME

read - read from file

SYNOPSIS

```
read(fildes, buffer, nbytes)
char *buffer;
```

DESCRIPTION

A file descriptor is a word returned from a successful open. buffer is the location of nbytes contiguous bytes into which the input will be placed. It is not guaranteed that all nbytes bytes will be read; for example if the file refers to tty at most one line will be returned. In any event the number of characters read is returned.

If the returned value is 0, then end-of-file has been reached.

SEE ALSO

iomov(2), open(3)

DIAGNOSTICS

As mentioned, 0 is returned when the end of the file has been reached. If the read was otherwise unsuccessful the return value is -1. Physical I/O errors or file descriptor not that of an input file can cause an error.

conversion characters `d`, `o` and `x` may be preceded by `h` to indicate a pointer to **short** rather than to `int`.

The `scanf` functions return the number of successfully matched and assigned input items. This can be used to decide how many input items were found. The constant `EOF` is returned upon end of input; note that this is different from `0`, which means that no conversion was done; if conversion was intended, it was frustrated by an inappropriate character in the input.

For example, the call

```
int i; float x; char name[50];
scanf( "%d%f%s", &i, &x, name);
```

with the input line

```
25 54.32E-1 thompson
```

will assign to `i` the value 25, `x` the value 5.432, and `name` will contain `'thompson\0'`. Or,

```
int i; float x; char name[50];
scanf("%2d%f*d%[1234567890]", &i, &x, name);
```

with input

```
56789 0123 56a72
```

will assign 56 to `i`, 789.0 to `x`, skip `'0123'`, and place the string `'56\0'` in `name`. The next call to `getchar` will return `'a'`.

SEE ALSO

`getc(3)`, `printf(3)`

DIAGNOSTICS

The `scanf` functions return `EOF` on end of input, and a short count for missing or illegal data items.

BUGS

The success of literal matches and suppressed assignments is not directly determinable.

- %** a single '%' is expected in the input at this point; no assignment is done.
- d** a decimal integer is expected; the corresponding argument should be an integer pointer.
- o** an octal integer is expected; the corresponding argument should be a integer pointer.
- x** a hexadecimal integer is expected; the corresponding argument should be an integer pointer.
- s** a character string is expected; the corresponding argument should be a character pointer pointing to an array of characters large enough to accept the string and a terminating '\0', which will be added. The input field is terminated by a space character or a newline.
- c** a character is expected; the corresponding argument should be a character pointer. The normal skip over space characters is suppressed in this case; to read the next non-space character, try '%ls'. If a field width is given, the corresponding argument should refer to a character array, and the indicated number of characters is read.
- %9f7** a floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a float. The input format for floating point numbers is an optionally signed string of digits possibly containing a decimal point, followed by an optional exponent field consisting of an E or e followed by an optionally signed integer.
- [** indicates a string not to be delimited by space characters. The left bracket is followed by a set of characters and a right bracket; the characters between the brackets define a set of characters making up the string. If the first character is not circumflex (^), the input field is all characters until the first character not in the set between the brackets; if the first character after the left bracket is ^, the input field is all characters until the first character which is in the remaining set of characters between the brackets. The corresponding argument must point to a character array.

The conversion characters **d**, **o** and **x** may be capitalized or preceded by **l** to indicate that a pointer to long rather than to **int** is in the argument list. Similarly, the conversion characters **e** or **f** may be capitalized or preceded by **l** to indicate a pointer to **double** rather than to **float**. The

NAME

scanf, fscanf, sscanf - formatted input conversion

SYNOPSIS

```
#include <stdio.h>
```

```
scanf(format [ , pointer ] . . . )  
char *format;
```

```
fscanf(stream, format [ , pointer ] . . . )  
FILE *stream;  
char *format;
```

```
sscanf(s, format [ , pointer ] . . . )  
char *s, *format;
```

DESCRIPTION

Scanf reads from the standard input stream stdin. Fscanf reads from the named input stream. Sscanf reads from the character string s. Each function reads characters, interprets them according to a format, and stores the results in its arguments. Each expects as arguments a control string format, described below, and a set of pointer arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. Blanks, tabs or newlines, which match optional white space in the input.
2. An ordinary character (not %) which must match the next character of the input stream.
3. Conversion specifications, consisting of the character %, an optional assignment suppressing character *, an optional numerical maximum field width, and a conversion character.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression was indicated by *. An input field is defined as a string of non-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted.

The conversion character indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. The following conversion characters are legal:

NAME

`scan` - search memory for difference from byte pattern
`scanw` - search memory for difference from word pattern

SYNOPSIS

```
scan(memory, cnt, pattern)  
char *memory;  
scanw(memory, cnt, pattern)  
char *memory;
```

DESCRIPTION

`Scan` and `scanw` search memory comparing each byte or word with the `pattern`. They return the number of the first byte or word that differs. The `cnt` argument specifies the number of bytes or words to scan.

NAME

stdio - standard buffered input/output package

SYNOPSIS

```
#include <stdio.h>
```

```
FILE *stdin;  
FILE *stdout;  
FILE *stderr;
```

DESCRIPTION

The functions described in Sections 3S constitute an efficient user-level buffering scheme. The in-line macros getc and putc(3) handle characters quickly. The higher level routines gets, fgets, scanf, fscanf, fread, puts, fputs, printf, fprintf, fwrite all use getc and putc; they can be freely intermixed.

A file with associated buffering is called a stream, and is declared to be a pointer to a defined type FILE. Fopen(3) creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. There are three normally open streams with constant pointers declared in the include file and associated with the standard open files:

<u>stdin</u>	standard input file
<u>stdout</u>	standard output file
<u>stderr</u>	standard error file

A constant 'pointer' NULL (0) designates no stream at all.

An integer constant EOF (-1) is returned upon end of file or error by integer functions that deal with streams.

Any routine that uses the standard input/output package must include the header file <stdio.h> of pertinent macro definitions. The functions and constants mentioned in sections labeled 3S are declared in the include file and need no further declaration. The constants, and the following 'functions' are implemented as macros; redeclaration of these names is perilous: getc, getchar, putc, putchar, feof, ferror, fileno.

SEE ALSO

open(3), close(2), read(3), write(3)

DIAGNOSTICS

The value EOF is returned uniformly to indicate that a FILE pointer has not been initialized with fopen, input (output) has been attempted on an output (input) stream, or a FILE pointer designates corrupt or otherwise unintelligible FILE

NAME

setbuf - assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

setbuf(stream, buf)
FILE *stream;
char *buf;
```

DESCRIPTION

Setbuf is used after a stream has been opened but before it is read or written. It causes the character array buf to be used instead of an automatically allocated buffer. If buf is the constant pointer NULL, input/output will be completely unbuffered.

A manifest constant BUFSIZ tells how big an array is needed:

```
char buf[BUFSIZ];
```

A buffer is normally obtained from malloc(3) upon the first getc or putc(3) on the file, except that output streams directed to terminals, and the standard error stream stderr are normally not buffered.

SEE ALSO

fopen(3), getc(3), putc(3), malloc(3)

data.

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen,
index, rindex - string operations

SYNOPSIS

```
char *strcat(s1, s2)
char *s1, *s2;
```

```
char *strncat(s1, s2, n)
char *s1, *s2;
```

```
strcmp(s1, s2)
char *s1, *s2;
```

```
strncmp(s1, s2, n)
char *s1, *s2;
```

```
char *strcpy(s1, s2)
char *s1, *s2;
```

```
char *strncpy(s1, s2, n)
char *s1, *s2;
```

```
strlen(s)
char *s;
```

```
char *index(s, c)
char *s, c;
```

```
char *rindex(s, c)
char *s;
```

DESCRIPTION

These functions operate on null-terminated strings. They do not check for overflow of any receiving string.

Strcat appends a copy of string s2 to the end of string s1. Strncat copies at most n characters. Both return a pointer to the null-terminated result.

Strcmp compares its arguments and returns an integer greater than, equal to, or less than 0, according as s1 is lexicographically greater than, equal to, or less than s2. Strncmp makes the same comparison but looks at at most n characters.

Strcpy copies string s2 to s1, stopping after the null character has been moved. Strncpy copies exactly n characters, truncating or null-padding s2; the target may not be null-terminated if the length of s2 is n or more. Both return s1.

Strlen returns the number of non-null characters in s.

Index (rindex) returns a pointer to the first (last) occurrence of character c in string s, or zero if c does not occur in the string.

BUGS

Strcmp uses native character comparison, which is signed on PDP11's, unsigned on other machines.

NAME

ungetc - push character back into input stream

SYNOPSIS

```
#include <stdio.h>
```

```
ungetc(c, stream)  
FILE *stream;
```

DESCRIPTION

Ungetc pushes the character c back on an input stream. That character will be returned by the next getc call on that stream. Ungetc returns c.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. Attempts to push EOF are rejected.

Fseek(3) erases all memory of pushed back characters.

SEE ALSO

getc(3), setbuf(3), fseek(3)

DIAGNOSTICS

Ungetc returns EOF if it can't push a character back.

NAME

vprintf - print a verbose message in standard form

SYNOPSIS

```
vprintf(s,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10)
char *s;
```

DESCRIPTION

This routine is like printf except that it only prints if the verbose flag is set. The line output will be in a standard form. The argument s is a printf style format string.

SEE ALSO

printf(3), diag(3)

NAME

write - write on a file

SYNOPSIS

```
write(fildes, buffer, nbytes)
char *buffer;
```

DESCRIPTION

A file descriptor is a word returned from a successful open call.

buffer is the address of nbytes contiguous bytes which are written on the output file. The number of characters actually written is returned. It should be regarded as an error if this is not the same as requested.

Writes which are multiples of 1024 characters long and begin on a 1024-byte boundary in the file are more efficient than any others.

SEE ALSO

iomov(2), open(3)

DIAGNOSTICS

Returns -1 on error: bad descriptor or physical I/O errors.

NAME

\$disk - raw disk driver

SYNOPSIS

built into monitor

DESCRIPTION

The raw disk driver treats the Fujitsu M2312 disk as a single sequentially-addressed file. It uses the Interphase 2181 disk controller. The disk is formatted in 512-byte sectors, 32 sectors per track, 7 tracks per cylinder, and 589 cylinders. However, the driver treats the disk as a file that is 67,551,232 bytes long. That is, single bytes may be read or written, although I/O is more efficiently done if accesses are done in multiples of the logical block size of 1024 bytes, starting on a logical block boundary.

SEE ALSO

fileys(4)

BUGS

Only one open to \$disk may be done at a time. The actual disk drive used may change at a future date, so naturally the physical parameters would change.

NAME

cal - driver for the SDU calendar chip

SYNOPSIS

built into monitor

DESCRIPTION

cal is the driver for the 64 bytes of ram on the SDU MCI46818 calendar and time-of-day chip. The first 14 bytes are used to keep calendar and time information. The last 50 bytes of this memory is nonvolatile, that is, it does not lose its contents if the Nu Machine is powered down.

BUGS

The 64 bytes are accessed by reads and writes to cal but to program the chip requires writing to the correct registers (bytes 10-13). The seek system call is useful in this regard.

SEE ALSO

seek(2), cmos(4)

NAME

cmos - driver for the SDU cmos configuration ram

SYNOPSIS

built into monitor

DESCRIPTION

cmos is the driver for the 2K bytes of cmos ram on the sdu. This memory is nonvolatile, that is, it does not lose its contents if the Nu Machine is powered down. Since the cmos ram contains information on the system configuration, it should not be written to directly. The driver config(4) should be used instead.

SEE ALSO

config(4)

NAME

config - driver for accessing the SDU cmos information

SYNOPSIS

built into the monitor

DESCRIPTION

The config driver is the interface to the SDU's cmos ram. It interprets the cmos ram as containing a certain structure of information. Using ioctl and write calls on config allows changing the contents of the cmos ram. The structure of information that the cmos ram contains and the ioctl calls for reading/writing this information are given in the /usr/monitor/h/confram.h file.

Defines that are useful for the ioctl call are:

```
#define IDXBITS    0x000F
#define CMDBITS    0xFF0
#define WRITEBIT   0x1000

#define VERSION    0x0010
#define CRC        0x0020
#define SHELL      0x0030
#define NUAVAIL    0x0040
#define MCARD      0x0050
#define SP         0x0060
#define DSKPART    0x0070
#define CLEAR      (0x0080|WRITEBIT)
#define ALL        0x0090
```

As an example, the following code fragment shows one way to change the baud rate of the remote port (port 0) to 2400:

```
#define SP          0x0060
#define WRITEBIT   0x1000
#define REMOTEPORT 0
#define LOCALPORT  1
struct sp {
    char sp_mode;           /* e.g. 8 bits/char, 16X clock, etc
    char sp_cmd;           /* e.g. RTS, ER, RXE, DTR, TXEN */
    short sp_baud;
} sp;
int fd;
...
fd = open("config", 2);
ioctl(fd, SP+REMOTEPORT, (char *) &sp);
sp.sp_baud = 2400;
ioctl(fd, WRITEBIT|(SP+REMOTEPORT), (char *) &sp);
close(fd);
...
```

The shell scripts corresponding to the rotary switch position can be read as the special file:

/config/i

where i is the switch position.

NAME

diag - driver for self-diagnostics on a Nubus board

SYNOPSIS

built into the monitor

DESCRIPTION

Each of the Nubus boards contains a rom with configuration information and a self-diagnostic. (This self-test is smaller than the diagnostics available on disk.) Success of the self-test for a given board is indicated by the turning off of the led for that board. If an error occurred the non-zero exit value is reported by the SDU monitor. To run the diagnostic on the board use:

```
/diag/i i
```

where i is the decimal number of the slot containing the board to be tested. (The number must be between 0 and 15).

The first string /diag/i tells the diag driver the board from which to read the self-test. The parameter i is passed to the self-test to let it know which board it is testing. Clearly, the board containing the self-test and the board to be tested will be the same (except possibly in very unusual circumstances).

NAME

disk, uroot - file system drivers

DESCRIPTION

disk and uroot are drivers for file systems contained on the physical device \$disk. Each file system on the disk must have its own driver. The file system is a Unix file system with byte ordering based on the M68000 processor. disk and uroot use a value that is the logical block offset from the beginning of \$disk at which the file system starts. A logical block contains 1024 bytes.

The very beginning of the raw disk is reserved to contain the file system driver disk. The raw disk also contains the sdu's root file system at a certain logical block offset from the beginning of the disk.

When the sdu monitor first starts it contains the raw disk driver \$disk but not any file system driver. The first time an access is attempted for a file on the root file system the sdu monitor reads the blocks at the beginning of the disk using device \$disk to install the file system driver into the monitor's device table. A block offset value is also stored into the device table. This value is used by the file system driver as the offset from the beginning of the disk at which the file system starts. The name given to this file system driver by the monitor is disk.

For example, to access the file "drv/ttf" in the root directory the pathname "/disk/drv/ttf" should be used. In accessing this (or any) file the sdu monitor uses the first component of the path name to open the driver disk and passes the rest of the path name drv/ttf to that driver. disk then uses the offset value in its table entry to make accesses to the raw device \$disk.

There can be (and usually are) other file systems on the disk. In general, to read files in a particular file system a driver for that file system must be installed, see setdr(2) and driver(1). The file system driver uroot is used to access the file system whose name is uroot. uroot can also be used to access other files systems, if it is installed into the sdu operating system appropriately. That is, the name of this driver as installed into the sdu device table must be the same name as the file system it is to access. uroot will search the configuration cmos ram on the sdu to find a file system with the same name as the file system driver's name, e.g., if called uroot then the uroot file system will be the one used; if called usr then the usr file system will be the one accessed. On the other hand, the other file system driver disk does not make use of its name as installed in the device table. It always looks for

the file system called root in the configuration cmos ram.

As mentioned above, each file system driver has its own offset value to know where the file system it handles starts. The name for the driver becomes the first component in the path name for any file in that file system.

SEE ALSO

\$disk(4), setdr(2), cd(1), pwd(1), config(4)

BUGS

The file system drivers can not create new files or change the block size of existing files, that is, they can not make any changes to the superblock or inodes. The file system drivers can not handle very large files, (greater than 138 blocks) nor directories more than ten blocks large. The locations of the file systems on the disk, in particular the sdu's root file system, may change in the future.

NAME

ioport - Interprocessor message exchange driver.

SYNOPSIS

driver ioport

DESCRIPTION

The ioport driver provides a mechanism for other processors such as the 68000 unix processor to send messages to a program running under the diagnostic monitor. To use the ioport, the monitor program does an open of the port which initializes it, then does an ioctl operation that simply waits until a message is available. When another processor puts a message in the port, the ioctl returns with a pointer to the message received.

The ioport itself is a small data structure in a well-known location in multibus memory that contains the synchronization mechanism that permits multiple processors to share access to the port. The structure looks like:

```
struct ioport
{
    unsigned char busy; /* flag indicating port in use */
    unsigned char valid; /* flag indicating port data valid */
    unsigned :16; /* unused */
    long msg; /* NuBus address of the message */
    long wakeup; /* address of wakeup vector */
};
```

In this data structure, busy and valid are just flags that are true when they contain 0x80 and false when they contain 0. Busy is a lock to acquire use of the port and should be set with a locked bus cycle. When first acquired, the valid flag should be false. It should stay in this state until the other data in the port, specifically the msg address is valid then set true. The wakeup address is a NuBus address that will normally be an interrupt pseudo-memory location. A processor using the port should write a NuBus 1 to this address (unless it is zero) after it has set the valid bit to wakeup the handler in the monitor.

Normally there will be a monitor process waiting to receive the message on the port. As soon as it has made a copy of the message address it will clear the valid and busy flags. This does not necessarily mean that it has fully processed the message, only that it read the data out of the port so that somebody else can use the port.

BUGS

NAME

keytty - driver for the graphics monitor

SYNOPSIS

built into monitor

DESCRIPTION

The keytty driver provides a line at a time i/o mechanism for the graphics monitor.

Input Editing

Normal characters are echoed as they are typed and saved in an input buffer for a process to read. Some control characters have special meanings on input in that they effect keytty processing. They are intercepted by the keytty driver and are not passed to the program:

- ^S** Freeze character. Suspends keytty output until the thaw or interrupt characters are typed.
- ^Q** Thaw character. If keytty output has been suspended by the freeze character, typing this character will start it up without any other effect.
- ^D** End of file character. This character is buffered but returns a when input processing reaches it, the read will return a value of zero signifying end of file.
- ^C** Interrupt character. Typing this character will abort all monitor processes that have the keytty open unless they have a special interrupt handler. Typing this character should return control to the monitor shell.
- ^U** Kill character. Typing this character deletes all characters up to the last carriage return in the input buffer. Each character so delete will be echoed by a backspace-space-backspace sequence which has the effect of erasing it on a display.

DELETE

Erase character. Erases the last character saved in the input buffer up to the last carriage return. Echos as backspace-space-backspace on the display.

CARRIAGE RETURN

Wakeup character. If a monitor process is waiting for keytty input, typing this character will cause as much of the input buffer as the process asked for to be passed to the process, normally the whole line. The carriage return will be passed to the process as a new-line character and echos as the newline character.

Escape character. Disables echoing up to and including the next carriage return, however, two escapes in a row will re-enables echoing and passes a single escape character into the input buffer. Used for file transfer protocols.

Output Processing

Writes to the keytty are normally just output but the new-line character outputs as carriage return, linefeed. Output can be frozen and thawed by the appropriate input characters.

SEE ALSO

ttys(1)

BUGS

The vcmem card must be in slot 8.

NAME

lights - The meaning of the RUN, SET_UP, and ATTN lights.

DESCRIPTION

On the front of the Nu Machine are three lights, two red lights called SET_UP and ATTN, and a green RUN light. If the ATTN light is on it means the self-tests for the sdu have not passed. The SET_UP light on means that the cmos ram on the SDU does not have a valid crc. The RUN light on means that a running system (like unix) is operational.

When the Nu Machine is powered up, or a hard reset occurs, all three lights are on. The SDU software immediately turns on the ATTN and SET_UP lights only. They are then turned off respectively as the self-tests pass and the cmos ram is checked as valid.

NAME

map - nu map driver

SYNOPSIS

built into monitor

DESCRIPTION

In order to access NuBus addresses from the sdu monitor a map is used that maps multibus addresses into NuBus addresses. There are 1K entries in the map, and each entry, if enabled, maps a 1K-byte multibus address range into a 1K-byte NuBus address range. Maintaining this map is the job of the map driver numap.

To map into a range of the NuBus address space, numap is first opened and then an ioctl call is made. The NuBus address size is given in 1K-byte page units by passing it as the request argument to ioctl and the base NuBus address is given as the argp argument. The ten least significant bits of the NuBus address should be zeroes, since they are not used by the map driver. The return value from ioctl is a multibus segment value which can be used to access the desired NuBus addresses. A zero return value indicates a failure to set up the requested mapping. The close system call frees that part of the map previously allocated by the ioctl call.

Multibus addresses from 0x40000 to 0xF7FFF (segments 0x400 to 0xF70) are available for use in the mapping. The other multibus addresses are reserved for strictly multibus addressing.

The driver numap should not be used directly. The subroutines in numap(3) should be used instead.

SEE ALSO

numap(3), open(3), ioctl(2), close(3)

BUGS

Only one ioctl call may be done for each open.

BUGS

The request argument is used in a different way from usual ioctl calls.

NAME

quart - driver for the quarter inch tape drive

SYNOPSIS

(already part of the sdu monitor software)

DESCRIPTION

quart automatically rewinds on opens and closes. When writing or reading data, multiples of 512-byte blocks must be used.

Seeking works with this driver.

To access file 'n' on the tape, the file name used should be "/quart/n". The very first file may be accessed as either "/quart" or as "/quart/1".

SEE ALSO

driver(1), setdr(2), read(3), write(3)

NAME

rotary - How the rotary switch positions are used.

DESCRIPTION

The rotary switch may be set into any of the positions 0 to 4. Each position is intended to serve a different purpose in the running of the Nu Machine. All of the switch positions except for position 0 make use of the contents of the cmos ram. If the cmos ram is invalid (has a bad crc), then the "switch position 0" code is run.

What actually happens to cause the various actions corresponding to the switch position (other than 0) is that an ascii shell script for that switch position is read from cmos ram and executed. The shell script is accessed as the device `/config/i` where `i` is the switch position (1 to 4). These shell scripts are written into the cmos ram at the time of system integration.

If the switch is in position 0, serial port 0 (the remote port) on the SDU is used as the SDU monitor device at 300 baud. The prompt will appear on the device connected to port 0.

Switch position 1 uses the remote port at 9600 baud.

Switch position 2 runs the `uboot` command, causing unix to be automatically booted off the disk.

The SDU hardware also uses switch position 2 to activate the "deadman" feature. Roughly every half second the rotary switch register on the SDU must be read to prevent an automatic system reset. Both the SDU software and the unix code have software in their clock handlers to read this register.

Switch position 3 uses the graphics monitor as the SDU monitor device.

Switch position 4 uses the local port at 9600 baud as the sdu monitor device.

NAME

tape - driver for the Cipher Microstreamer Tape Drive

SYNOPSIS

(already part of the sdu monitor software)

DESCRIPTION

tape is the driver for the TAPEMASTER controller using the Cipher series F880 Microstreamer Tape Drive. The tape automatically rewinds on opens and closes. When writing data to a new tape, or adding data to the end of previously written data, multiples of 1024-byte blocks should be written. However, when doing reads or when replacing existing bytes with new data, the I/O may be done in arbitrary numbers of bytes, although multiples of 1024-bytes are most efficient.

Seeking works with this driver.

To access file 'n' on the tape, the file name used should be "/tape/n". The very first file may be accessed as either "/tape" or as "/tape/1".

SEE ALSO

driver(1), setdr(2)

NAME

tar, tarq - tar driver for half-inch and quarter-inch tapes

DESCRIPTION

tar is the driver for files contained in tar format on half-inch tape. tarq is the driver name for files contained in a tar format on quarter-inch tape. (The tar format is the Unix tar format.) The difference in names tells the driver whether it is to call upon the tape driver or the quart driver to access a particular file.

When the sdu operating system first starts it contains the drivers tape and quart but not the tar driver (under either name). To access the file filename on tape, the pathname /tar/filename should be used if half-inch tape is used, or /tarq/filename if the file is on quarter-inch tape.

The first time an access is attempted for a tar file the sdu operating system reads the first file on the tape to install the tar driver into the operating system's device table. The tar driver will then search the second physical file on the tape to see if it is in tar format. The file filename is then searched for. Actually, the tar driver starts by checking the first file on the tape to see if it is in tar format, and if that fails, checks if the second file on the tape is in tar format. This is done so that not every tar tape for use by the sdu operating system has to have the tar driver as its first file. Obviously, this is only useful once the tar driver has been downloaded into the system by a previous usage.

SEE ALSO

tape(4), quart(4)

BUGS

The tar driver can be used only for reading, not writing.

NAME

ttf - Hex download protocol

SYNOPSIS

(already part of the sdu monitor software)

DESCRIPTION

The ttf driver provides a very general, though slow way to access remote files via serial tty lines. It implements a file transfer protocol on top of the normal, line at a time, tty driver. Bytes of the file are transmitted in hex so any kind of file can be transferred. The normal freeze/thaw mechanism remains in operation to avoid flow control problems.

Assuming that a suitable handler is provided on the other end of the tty line, once the driver is loaded, programs may access arbitrary files using a name like:

```
/ttf/foo/bar
```

where foo/bar is a name to be interpreted on the remote system.

To open files and pass data, the monitor and remote system exchange simple, line oriented messages. To access a file, ttf sends:

```
~Oname\n
```

where name is the name of the remote file to be transferred. If the remote system can access file name it replies to ttf with:

```
~O\n
```

Any other message is interpreted to be no access and ttf returns from the open with a "file not found" type error. Once the connection has been established, the remote system sends data packets of the form:

```
~D XXXXXXXXXXXX\n
```

where X represents a hex digit. Each pair of hex digits describe one byte of the file. The line can contain up to 80 hex digits. After each such line, the monitor will reply with an acknowledge message of the form:

```
~A\n
```

When the end of file is reached, the remote system should send an end of file message such as:

~C\n

SEE ALSO
tty(4)

BUGS

The protocol as currently implemented is not very robust. There are no checksums transferred, hence no error recovery. Writes are not implemented. Only one file at a time can be transferred by ttf. Ttf turns the system clock off to prevent the tty driver from dropping characters.

NAME

tty - serial tty driver

SYNOPSIS

built into monitor

DESCRIPTION

The tty driver provides a line at a time i/o mechanism for serial ports. It is designed for full-duplex ascii terminals.

Input Editing

Normal characters are echoed as they are typed and saved in an input buffer for a process to read. Some control characters have special meanings on input in that they effect tty processing. They are intercepted by the tty driver and are not passed to the program:

- ^S** Freeze character. Suspends tty output until the thaw or interrupt characters are typed.
- ^Q** Thaw character. If tty output has been suspended by the freeze character, typing this character will start it up without any other effect.
- ^D** End of file character. This character is buffered but returns a when input processing reaches it, the read will return a value of zero signifying end of file.
- ^C** Interrupt character. Typing this character will abort all monitor processes that have the tty open unless they have a special interrupt handler. Typing this character should return control to the monitor shell.
- ^U** Kill character. Typing this character deletes all characters up to the last carriage return in the input buffer. Each character so delete will be echoed by a backspace-space-backspace sequence which has the effect of erasing it on a display.

DELETE

Erase character. Erases the last character saved in the input buffer up to the last carriage return. Echos as backspace-space-backspace on the display.

CARRIAGE RETURN

Wakeup character. If a monitor process is waiting for tty input, typing this character will cause as much of the input buffer as the process asked for to be passed to the process, normally the whole line. The carriage return will be passed to the process as a newline character and echos as the newline character.

~ Escape character. Disables echoing up to and including the next carriage return, however, two escapes in a row will re-enables echoing and passes a single escape character into the input buffer. Used for file transfer protocols.

Output Processing

Writes to the tty are normally just output but the newline character outputs as carriage return, linefeed. Output can be frozen and thawed by the appropriate input characters.

SEE ALSO

ttys(1)

BUGS

Very long lines do not appear to delete fully on kill characters.

NAME

cmosram - Contents of cmos ram on the sdu.

SYNOPSIS

```
#include <confram.h>
```

DESCRIPTION

The sdu uses the cmos ram to determine system configuration information, such as the type and layout of the disk, baud rates for the serial ports, etc. The sdu cmos ram memory can be accessed as the file "cmos", but for almost all purposes should be accessed indirectly through the "config" driver.

The following is the structure for the cmos ram contents.

```
struct cmos_ram
{
    unsigned short crc;
    struct sh {
        short sh_off; /* switch script offset */
        short sh_size; /* shell script size */
    } sh[5];

    short nuavail; /* bit=0 if not used */

    /* info on the multibus card slots */
    struct mcard {
        char m_name[8];
        char m_interrupt; /* interrupt (0-7) */
        char m_port; /* multibus IO */
        char m_data[24]; /* misc. info */
    } m[8];

    /* info for the serial port driver */
    struct sp { /* check hardware description */
        char sp_mode;
        char sp_cmd;
        short sp_baud;
    } sp[2];

    unsigned char version; /* software version */

    /* logical disk partitions */
    struct dskpart {
        char d_user[8]; /* name */
        char d_unit; /* unit/device number */
        long d_offset; /* byte offset */
        long d_size; /* size in bytes */
    } dskpart[10];
};
```

NAME

crom - Contents of configuration rom on NuBus cards.

SYNOPSIS

```
#include <conf.h>
```

DESCRIPTION

Each NuBus card contains a configuration rom that contains various information about the card. The address of this rom grows down from the top of the slot so that for example, the last byte of the rom is located at $0xFSFFFFFFC$, where S is the slot number in hex. The rom occupies only the low byte of each word.

The following header is adjusted to be at the highest addresses in the rom.

```
struct confhdr
{
    char    resource;        /* resources for boot procedure */
    char    reserved;       /* reserved for future use */
    char    testtime;       /* log2 of self-test time */
    char    layout;        /* ROM header version number */
    char    flags;         /* board dependent flags */
    char    flgoffs[3];     /* RAM offset of device flag register */
    char    diagoffs[3];   /* ROM offset of diagnostic if any */
    char    drvoffs[3];    /* ROM offset of device driver if any */
    char    cfoffs[3];     /* board offset of config register */
    char    part[16];      /* assembly part number */
    char    boardtype[8];  /* generic name of the board */
    char    vendor[4];     /* vendor id */
    char    romsize;       /* log2 of ROM size in bytes */
#define NONCRC 18        /* number of bytes not used in CRC */
    char    crcsum[2];     /* checksum of bytes in ROM */
    char    rev[8];       /* assembly revision level */
    char    serial[8];    /* serial number */
};
```

NAME

unixconfig - Contents of unix configuration file.

SYNOPSIS**DESCRIPTION**

The configuration file tells Unix where boards are located, how the disk is laid out and certain other information. The file is an ascii file in records of one to a line. Each record describes some object, logical or physical. The first token on the line terminated with a colon gives the type of object and the rest of the items on the line give various properties of the object. Normally they are hex numbers.

object	fields
cpu:	slot addr
ram:	slot addr number of 1k pages
port:	ioport addr interrupt address
disk:	port number channel number start byte size in bytes
logport:	port number
logchannel:	channel number

Objects do not have to be unique although no two lines should be identical. For example, if a system has several ram cards, it should have several "ram:" entries in this file.

SEE ALSO

uboot(1)

BUGS

This format is still subject to change.

NAME

2181 - diagnostic for Fujitsu M2312 disk

SYNOPSIS

2181 [-cstvDFHLQTV] [-A track[-endtrack]] [-R count] [test ...]

DESCRIPTION

In addition to the standard diagnostic options, see diag(6), there are several options specific to this diagnostic:

- F Format the specified area of the disk. (Also requires the D option.)
- H Ask for help in using this diagnostic.
- A Specify the area (tracks) of the disk that are to be tested or formatted. The area may be specified either as a single track or as a range of tracks, using a dash to separate the first from the last.

Since the D option destroys data in the area of the disk being tested, it should be used with great care on a disk that has information that is not to be destroyed. The A option must be given correctly to avoid disaster.

SEE ALSO

diag(6)

BUGS

The write tests running on a large area of the disk take unreasonably long to run.

NAME

cpu - diagnostic for 68000 cpu cards

SYNOPSIS

cpu [-cstvLQTV] [-R count] [-S slot] [test...]

DESCRIPTION

The cpu diagnostic uses the standard command line options, see diag(6). There are no special operating instructions.

SEE ALSO

diag(6)

NAME

diag - standard format for running a diagnostic

SYNOPSIS

diag [-cstvDLQTV] [-R count] [-S slot] [test ...]

DESCRIPTION

The diagnostic programs make use of a common diagnostic library to perform certain common functions, see diag(3). The standard command line options are:

- c Print configuration information about the card.
- s Print out summary information when the diagnostic ends.
- t Print the name of tests as they are executed.
- v Print debugging information verbosely.
- D Run the "dirty" tests. Certain tests such as formatting a disk may be designated as destructive (of information). These tests will not be run unless the D option is given in the command line.
- L Loop on the test set.
- Q Do not print error messages.
- T Print a list of test names but do not execute.
- V Print the version number of the diagnostic.
- R Repeat the test sequence count times, a decimal number.
- S Test the board in slot, a decimal number.

If no test numbers are specified then the entire diagnostic will be run. Otherwise, only the listed tests will be run. Tests can be specified as individual decimal numbers or as numbers separated by dashes, meaning all the tests between the two numbers inclusively.

A diagnostic program may also use other command line options specific to its tests, but these must be given using different characters from the standard options.

NAME

keybd - diagnostic for Pegasus keyboards

SYNOPSIS

keybd [-tvTV] [-S slot]

DESCRIPTION

The keybd diagnostic uses a subset of the standard command line options, see diag(6). There is only one test in this diagnostic. This test will "echo" whatever key is depressed on the keyboard. The echo will be done to the device which ran the test. Striking special keys will cause appropriate messages to be displayed. The HOME key will have the message " HOME " printed, the CTRL key will cause " CTRL " to be displayed, etc. The test will continue until the 'C' key is struck while the CTRL key is held down.

The vcmem board must be operational to run this test.

SEE ALSO

diag(6), vcmem(6)

NAME

quart - diagnostic for Cipher quarter inch tape driver

SYNOPSIS

quart [-cstvDLQTV] [-R count] [test...]

DESCRIPTION

The quarter inch tape (cartridge) drive is part of the office version of the Nu Machine. The command line options are standard, see diag(6). There are no special operating instructions.

SEE ALSO

diag(6)

NAME

ram - diagnostic for NuBus ram cards

SYNOPSIS

ram [-cstvLQTV] [-R count] [-S slot] [test...]

DESCRIPTION

The ram diagnostic uses the standard command line options, see diag(6). There are no special operating instructions.

SEE ALSO

diag(6)

BUGS

The moving inversion tests take unreasonably long to run.

NAME

tape - diagnostic for TAPEMASTER controller

SYNOPSIS

tape [-cstvDLQTV] [-R count] [test...]

DESCRIPTION

The command line options are standard, see diag(6). There are no special operating instructions.

SEE ALSO

diag(6)

NAME

vcmem - diagnostic for NuBus video memory cards

SYNOPSIS

vcmem [-cstvLQTV] [-R count] [-S slot] [test ...]

DESCRIPTION

The vcmem diagnostic uses the standard command line options, see diag(6). There are no special operating instructions.

SEE ALSO

diag(6)

**Nu Machine System Diagnostic Unit
General Description**

2242829-0001

**Distributed by LMI 6033 W. Century Blvd. Los Angeles CA 90045
USA**

Information furnished in this document is believed to be accurate and reliable. However, no responsibility is assumed by Texas Instruments Incorporated for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Texas Instruments Incorporated. Texas Instruments Incorporated reserves the right to change product specifications at any time.

Nu Machine and NuBus are trademarks of Texas Instruments Incorporated.

Multibus™ is a trademark of Intel Corp.

Copyright © 1984 Texas Instruments All rights reserved.

This manual provides detailed information about the Texas Instruments Nu Machine(TM)* System Diagnostic Unit (SDU). It is primarily directed to the system programmer and provides some information for the installation personnel.

Information in this manual is divided into the following sections:

Section

1. General--Contains physical and functional descriptions that acquaint the user with the hardware components and capabilities of the Nu Machine SDU.
2. Installation--Outlines procedures for unpacking the Nu Machine SDU from its shipping container, installing the board in the Nu Machine chassis, and performing diagnostics.
3. Operation--Describes the board and front panel light-emitting diodes (LEDs), the back control panel rotary switch, and the reset switch and signals.
4. Programming--Presents information for use by programmers on the function of the SDU and defines the Multibus(R)** address space and NuBus(TM)* address space reserved for SDU use.

Appendix

- A. P1 Pin Assignments--Shows the standard NuBus pin assignments on P1.
- B. P2 Pin Assignments--Shows the Multibus pin assignments on P2.
- C. P3 Pin Assignments--Shows the input/output (I/O) pin assignments on P3.

* Nu Machine and NuBus are trademarks of Texas Instruments Incorporated.

** Multibus is a registered trademark of Intel Corporation.

- D. Serial Port Pin Assignments--Shows the remote and local serial port pin assignments.
- E. Multibus Compliance Levels--Describes the Multibus attributes supported by the SDU.

Reference Documents

The following documents contain additional information related to SDU design. These documents cover NuBus and Multibus specifications, detailed programming and implementation information on some components, and interface requirements for SDU I/O.

TITLE	PART NUMBER
<u>NuBus Specification</u>	2242825-0001
<u>iAPX 86, 88 User's Manual</u> (Intel)	
<u>Intel Component Data Catalog</u>	
<u>Motorola Microprocessor Data Manual</u>	
<u>Multibus Specification</u> (Intel) or <u>IEEE Standard Microcomputer System</u> <u>Bus Specification</u> (IEEE Std 796-1983)	9800683
<u>AP-28A</u> (Intel)	
<u>EIA-RS-232C</u>	
<u>QIC-02 Interface Specification</u> (Rev. D, 9/23/82)	
<u>Nu Generation Computer System</u> <u>Architecture Specification</u>	2236632-0001
<u>Nu Machine Diagnostic User Manual</u>	2244479-0001
<u>Nu Machine Rack Module,</u> <u>General Description</u>	2242821-0001
<u>Nu Machine Office Module,</u> <u>General Description</u>	2242822-0001
<u>Nu Machine Unpacking and</u> <u>Inventory Guide</u>	2244492-0001

<u>Nu Machine Installation Manual</u>	2242824-0001
<u>Nu Machine SDU Operating System User Manual</u>	2242811-0001
<u>Nu Machine SDU Operating System Implementation Description</u>	2242812-0001
<u>Nu Machine SDU Operating System Driver Design Guide</u>	2242813-0001
<u>Nu Machine SDU Development System, User Guide</u>	2242815-0001
<u>Nu Machine SDU Development System, Assembler Reference Manual</u>	2242816-0001

Notation

The following notational conventions have been used throughout this document.

Ones and Zeros

Signal or bit names that end with an asterisk (*) are active low. Names that do not end with an asterisk (*) are active high. Low means logic 0; high means logic 1.

Reset

The four SDU reset functions are:

- ◆ NuBus RESET*
- ◆ Multibus INIT*
- ◆ an SDU reset signal
- ◆ a tape controller reset signal

Each of these signals will be referred to by its complete name, such as NuBus RESET*, unless the context makes it clear which of the preceding reset signals is intended.

Numbers

Bit 0 of a byte is the least significant bit, and bit 7 is the most significant bit. Byte numbering is similar to bit numbering. For example, in a 16-bit halfword, bit 7 of byte 0 is adjacent to bit 0 of byte 1. Words are 32 bits.

The hexadecimal symbol (0x) precedes each hexadecimal number throughout this document. A bit represented by X within a hexadecimal number can range in value from 0x0 to 0xF.

1	GENERAL DESCRIPTION.....	1-1
1.1	General.....	1-1
1.2	Purpose of Equipment.....	1-1
1.3	Equipment Description.....	1-3
1.3.1	8088 and Related Hardware.....	1-4
1.3.2	Configuration ROM.....	1-5
1.3.3	Internal Multibus.....	1-5
1.3.4	NuBus and Multibus Interface.....	1-5
1.3.5	Front/Back Control Panel Interface.....	1-5
1.3.6	Serial Communications.....	1-5
1.3.7	1/4-Inch Tape Interface.....	1-5
1.3.8	Power Supply Interface.....	1-5
1.3.9	AC Shutdown.....	1-6
1.3.10	Thermal Sensors.....	1-6
1.4	Specifications.....	1-6
1.5	Functional Description.....	1-7
1.5.1	Bus Conversion.....	1-7
1.5.1.1	NuBus-to-Multibus Conversion.....	1-7
1.5.1.2	Multibus-to-NuBus Conversion.....	1-8
1.5.1.3	Interrupt Translation.....	1-8
1.5.2	NuBus Central Features.....	1-9
1.5.2.1	System Clock.....	1-9
1.5.2.2	Time-Out Recovery.....	1-9
1.5.3	Nonvolatile Features.....	1-9
1.5.4	1/4-Inch Tape Interface.....	1-9
1.5.5	Interval Timer.....	1-9
1.5.6	Debug/Diagnostic Facilities.....	1-9
1.5.7	System Status Display.....	1-10
1.5.8	Serial Ports.....	1-10
1.5.9	Power Supply Interface.....	1-10
1.5.10	System Bootstrap.....	1-10
2	INSTALLATION.....	2-1
2.1	General.....	2-1
2.2	Unpacking/Packing the SDU Board.....	2-1
2.3	SDU Installation Procedures.....	2-2
3	OPERATION.....	3-1
3.1	General.....	3-1
3.2	Fault Indication LED.....	3-1

3.3	Front Panel LEDs.....	3-1
3.4	Back Panel Rotary Switch.....	3-1
3.5	Reset Pushbutton and Signals.....	3-1
3.6	Self-Diagnostics.....	3-2
4	PROGRAMMING.....	4-1
4.1	General.....	4-1
4.2	Multibus Address Space Definition.....	4-1
4.2.1	SDU ROM.....	4-3
4.2.2	SDU RAM.....	4-3
4.2.3	Address Map.....	4-3
4.2.4	Front Panel LEDs.....	4-5
4.2.5	Back Panel Switch.....	4-6
4.2.6	Control-Status Register 1.....	4-7
4.2.7	Control-Status Register 0.....	4-9
4.2.8	A/D Converter.....	4-11
4.2.9	CMOS TOD Chip.....	4-12
4.2.10	RS-232C Serial Ports.....	4-12
4.2.11	Programmable Interval Timers.....	4-13
4.2.12	Time-Out Register.....	4-14
4.2.13	1/4-Inch Tape Interface.....	4-14
4.2.14	Bus Integrity Registers.....	4-15
4.2.15	Interrupt Controllers.....	4-18
4.2.16	Multibus Interrupt Register.....	4-21
4.2.17	CMOS RAM.....	4-23
4.3	NuBus Address Space Definition.....	4-24
4.3.1	Multibus Memory Space.....	4-25
4.3.2	Multibus I/O Space.....	4-25
4.3.3	Configuration Space.....	4-25
4.4	Software Development.....	4-26
A	P1 PIN ASSIGNMENTS.....	A-1
B	P2 PIN ASSIGNMENTS.....	B-1
C	P3 PIN ASSIGNMENTS.....	C-1
D	SERIAL PORT PIN ASSIGNMENTS.....	D-1
E	MULTIBUS COMPLIANCE LEVELS.....	E-1

Figure 1-1.	Nu Machine System Diagnostic Unit Board.....	1-2
Figure 1-2.	SDU Block Diagram and System Interfaces.....	1-3
Figure 4-1.	Address Map Format.....	4-4
Figure 4-2.	Translation Definition.....	4-5
Figure 4-3.	LED Register.....	4-6
Figure 4-4.	Switch Register.....	4-7
Figure 4-5.	CSR1 Read Format.....	4-8
Figure 4-6.	CSR1 Write Format.....	4-8
Figure 4-7.	Control-Status Register 0.....	4-10
Figure 4-8.	Integrity Register 0.....	4-16
Figure 4-9.	Integrity Register 1.....	4-16
Figure 4-10.	Integrity Register 2.....	4-17
Figure 4-11.	Integrity Register 3.....	4-17
Figure 4-12.	Integrity Register 4.....	4-18

Figure 4-13.	Integrity Register 5.....	4-18
Figure 4-14.	0x1C1E0 Read Format.....	4-22
Figure 4-15.	CMOS RAM Address Space.....	4-23
Figure 4-16.	SDU NuBus Addresses.....	4-24

Tables

Table 1-1.	SDU Board Specifications.....	1-6
Table 2-1.	Required SDU Cables.....	2-4
Table 3-1.	Self-Diagnostics Results.....	3-3
Table 4-1.	Multibus Address Space.....	4-1
Table 4-2.	SDU Register Space Definition.....	4-2
Table 4-3.	ROM Address Allocation.....	4-3
Table 4-4.	IC084 Bit Patterns.....	4-7
Table 4-5.	A/D Converter Addresses.....	4-12
Table 4-6.	Multibus Addresses to Serial Port Registers.....	4-13
Table 4-7.	PIT Address Map.....	4-14
Table 4-8.	Integrity Registers.....	4-15
Table 4-9.	PIC0 Interrupt Functions.....	4-19
Table 4-10.	PIC1 Interrupt Functions.....	4-20
Table 4-11.	PIC2 Interrupt Functions.....	4-21
Table 4-12.	PIC Addresses.....	4-21

Table 4-13.	Interrupt Addressing.....	4-22
Table A-1.	NuBus Pin Assignments.....	A-1
Table B-1.	SDU Multibus Pin Assignments.....	B-1
Table C-1.	P3 Row A.....	C-1
Table C-2.	P3 Row B.....	C-2
Table C-3.	P3 Row C.....	C-3
Table D-1.	Remote Serial Port Pin Assignments.....	D-1
Table D-2.	Local Serial Port Pin Assignments.....	D-2

1.1 General

This manual provides unpacking, installation, testing, and operating information for the Texas Instruments Nu Machine (TM)* System Diagnostic Unit (SDU) board (see Figure 1-1). This section contains functional and physical descriptions to acquaint the user with the hardware components and capabilities of the Nu Machine SDU.

1.2 Purpose of Equipment

The SDU provides many one-per-system functions and smart front-end and diagnostic capabilities. Since these functions are concentrated on the SDU, rather than on separate central processing unit (CPU) cards, the system is able to support multiple processors without conflict. The SDU Monitor, the PROM-resident portion of the SDU Operating System, provides a flexible system-boot environment.

* Nu Machine is a trademark of Texas Instruments Incorporated.

SDU GENERAL DESCRIPTION
GENERAL DESCRIPTION

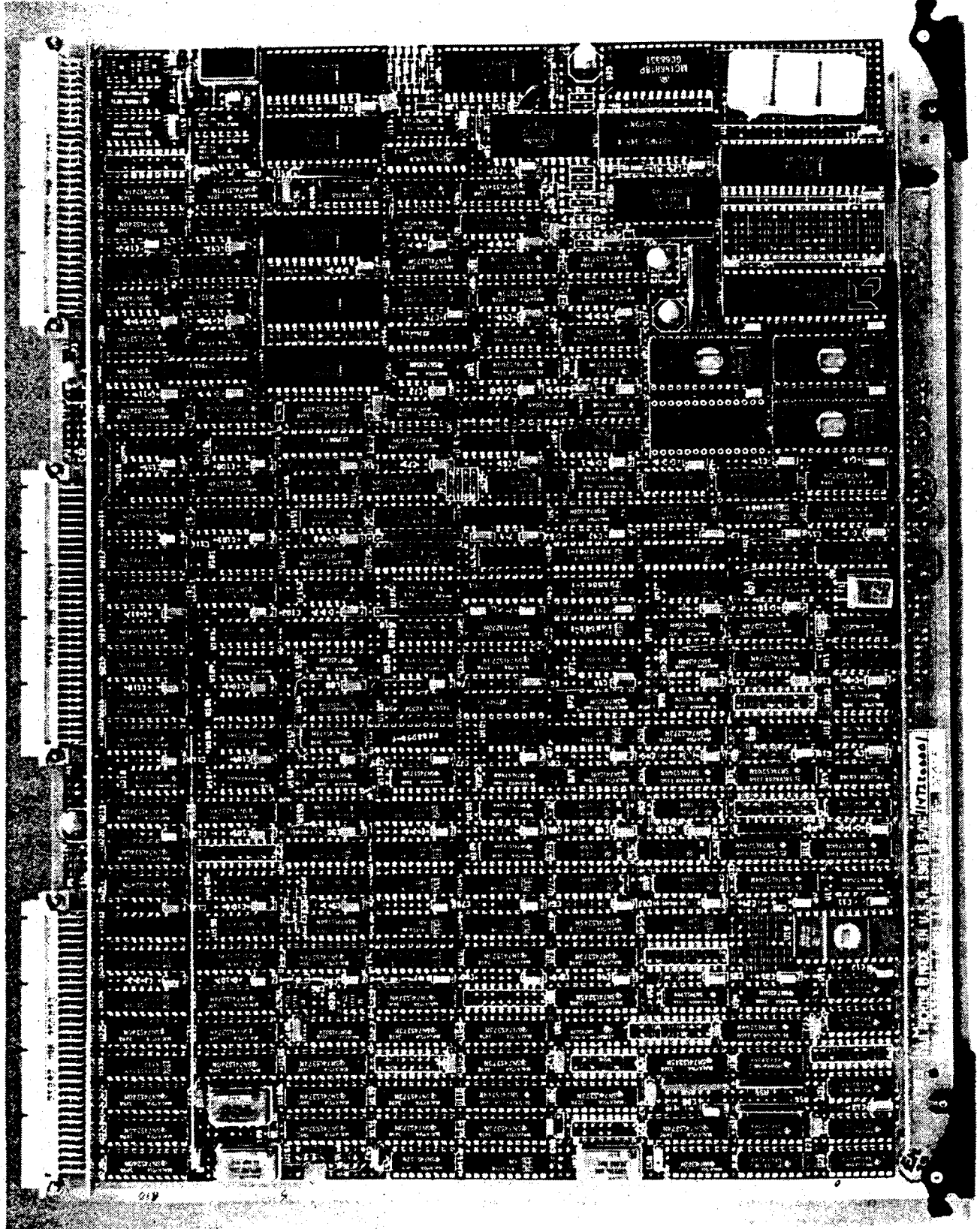


Figure 1-1. Nu Machine System Diagnostic Unit Board.

1.3 Equipment Description

The SDU is an 8088-based, single printed wiring board (PWB) microprocessor system with additional special purpose hardware to interface to the NuBus (TM)* architecture and provide required system utilities. Figure 1-2 shows the major hardware subsections and interfaces with other Nu Machine system components. The following paragraphs describe the major SDU components.

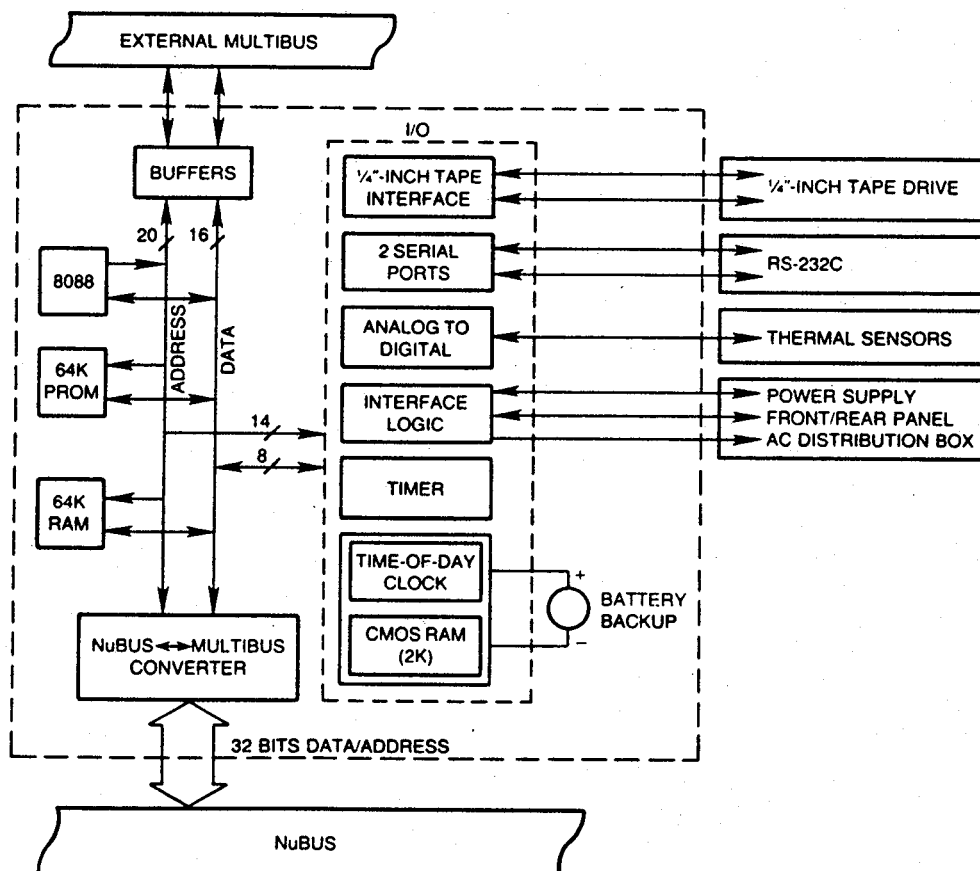


Figure 1-2. SDU Block Diagram and System Interfaces.

* NuBus is a trademark of Texas Instruments Incorporated.

SDU GENERAL DESCRIPTION
GENERAL DESCRIPTION

1.3.1 8088 and Related Hardware

The 8088 microprocessor system is the heart of the SDU. It consists of the following:

- ◆ 8088 microprocessor -- The 8088 specifications and interface are described in the Intel Component Data Catalog.
- ◆ Reset circuits -- The board-level SDU reset signal initializes internal registers to the power-up state and also resets the on-board 8088. The SDU reset signal is driven active by any of the following:
 1. The operator pressing the back panel reset button. A resistor capacitor (RC) circuit then holds the SDU reset signal low for about 3 milliseconds.
 2. The power supply generating an active 5 volts dc out-of-tolerance signal. If the SDU sends a high or low margin signal to the power supply, the dc out-of-tolerance signal is ignored.
 3. The remote serial port detecting a line break character while the back panel switch is at 0, 2, 3, or 4
 4. The on-board deadman timer triggering while the back panel switch is at 2. The timer triggers if the switch panel register, Multibus (R)* address 0x1C084, is not read once every one-half second.
- ◆ Clock generation circuits -- The 8088 clock is derived from a 14.7456 megahertz crystal giving the 8088 an effective clock rate of 4.9152 megahertz.
- ◆ Multibus time-out logic -- The Multibus time-out logic generates a READY signal to the 8088 as well as a time-out interrupt if an operation takes more than 3 milliseconds to complete.
- ◆ 64K bytes of ROM
- ◆ Programmable interrupt controllers -- Three programmable interrupt controllers (PICs) provide a flexible interface between the 8088 and devices

* Multibus is a registered trademark of Intel Corporation.

requiring real-time servicing. The three PICs (PIC0, PIC1, and PIC2) are configured as a master PIC (PIC0) and two slave PICs (PIC1 and PIC2) as described in the 8259A data sheet. Paragraph 4.2.15, Interrupt Controllers, details the operation of the 8259A.

- ◆ 64K bytes of dynamic RAM
- ◆ Multibus interface logic

1.3.2 Configuration ROM

The SDU has a 2K-byte configuration ROM and a configuration register residing on the NuBus. The contents and format of the configuration ROM are described in the Nu Generation Computer System Architecture Specification. The configuration register implements only the on-board LED bit.

1.3.3 Internal Multibus

The SDU is built around an internal Multibus. The I/O/Data Bus and CMOS/Data Bus, extensions of the internal Multibus, reduce drive requirements for the various buses. The external Multibus is also an extension of the internal Multibus. The buffers between the internal Multibus and external Multibus provide isolation and meet the loading requirements of the Multibus. (The relationship between the internal Multibus, external Multibus, and the SDU board is explained in Paragraph 4.2.7, Control-Status Register 0. Multibus attributes supported by the SDU are discussed in Appendix E, Multibus Compliance Levels.)

1.3.4 NuBus and Multibus Interface

The SDU interfaces to and enables two-way conversions between the NuBus and Multibus.

1.3.5 Front/Back Control Panel Interface

Three lines from the SDU drive the front panel LEDs. Four lines into the SDU indicate the mode selector position on the back panel rotary switch. A single input from the back control panel resets the SDU.

1.3.6 Serial Communications

Two asynchronous RS-232C ports support data rates to 19.2K baud.

1.3.7 1/4-Inch Tape Interface

A QIC-02 streaming tape drive interface provides system backup capability and diagnostic loading media.

1.3.8 Power Supply Interface

Two signal lines from the power supply to the SDU indicate the 5 volts and ac power status. A line from the SDU to the

SDU GENERAL DESCRIPTION
GENERAL DESCRIPTION

power supply enables the SDU to margin the +5 volt supply above and below nominal.

1.3.9 AC Shutdown

A line from the SDU to the ac distribution box enables the SDU to shut off the ac power in emergency situations.

1.3.10 Thermal Sensors

Five inputs from the optional external thermal sensors to an analog-to-digital converter are provided.

1.4 Specifications

Table 1-1 lists the physical, electrical, and environmental specifications for the SDU board.

Table 1-1. SDU Board Specifications

CHARACTERISTIC	SPECIFICATION
Ambient Temperature:	
Operating	10 to 35 degrees C
Storage	-40 to 65 degrees C
Ambient Humidity:	
Operating	15 to 80% (noncondensing)
Storage	5 to 95% (noncondensing)
Altitude:	
Operating	-300 to 3,000 meters (-990 to 10,000 feet)
Storage	-300 to 12,000 meters (-990 to 40,000 feet)
Shock:	
Operating	15g for 11 ms
Storage	25g for 11 ms
Vibration:	
Operating	.5g rms, random
Storage	.75g rms, random
Power	48 W
Current:	
+5 V	9 A
+12 V	150 mA
-12 V	120 mA

1.5 Functional Description

The following paragraphs describe the function of the major Nu Machine SDU components.

1.5.1 Bus Conversion

The NuBus/Multibus converter resides on the SDU. Bus conversion in both directions is done by hardware mapping logic and does not require 8088 microprocessor intervention. The two buses run independently until one bus accesses the other. Under certain circumstances, a NuBus card can attempt to access the Multibus at the same time that a Multibus card is accessing the NuBus. The SDU prevents a lockup by giving the NuBus a "Try Again Later" response and allowing the Multibus cycle to complete. The following two paragraphs describe the conversions in more detail.

1.5.1.1 NuBus-to-Multibus Conversion The Multibus memory space and I/O space (as described in Paragraphs 4.3.1 and 4.3.2, respectively) are completely contained within the SDU NuBus slot space. Thus, there is a direct correlation between NuBus addresses and the accessed Multibus addresses.

The Multibus memory space can be accessed with byte, halfword, and word operations. Byte and halfword operations on the NuBus are translated into byte and halfword operations on the Multibus. Word operations on the NuBus are translated into two halfword operations on the Multibus with the lower halfword transfer occurring first.

The Multibus I/O space can be accessed in one of two different address ranges. In one range, byte, halfword, or word transfers can access the Multibus I/O space but only the low byte of each NuBus word is valid data. In the other range, byte, halfword, and word transfers are all supported across the interface.

When a NuBus access of either the Multibus memory or I/O space occurs, the following sequence of events occurs:

1. START* on the NuBus accesses the Multibus.
2. The SDU bus translation logic initiates arbitration for the Multibus.
3. The SDU bus translation logic acquires the Multibus.
4. The Multibus operation (or operations for 32 bits) occurs as per the Multibus Specification.

SDU GENERAL DESCRIPTION

GENERAL DESCRIPTION

5. The SDU returns XACK* on the Multibus.
6. The SDU does an ACK cycle on the NuBus.

The SDU detects bus locks on the NuBus and asserts the Multibus signal LOCK* if a NuBus-to-Multibus translation is in progress. Thus, indivisible transfers are supported across the bus interface.

1.5.1.2 Multibus-to-NuBus Conversion Multibus-to-NuBus conversions are similar in sequence to NuBus-to-Multibus conversions. The following sequence of events occur:

1. The address map is initialized (see Paragraph 4.2.3) to point to the desired NuBus page.
2. Multibus master initiates Multibus transfer (upper ten bits of Multibus address select map entry).
3. Valid bit (bit 23) of map entry indicates a NuBus access.
4. The SDU bus translation logic initiates arbitration for the NuBus.
5. The SDU acquires NuBus mastership.
6. The NuBus operation is initiated by a START cycle as explained in the NuBus Specification.
7. The addressed slave on the NuBus responds with an ACK*.
8. The SDU bus translation logic generates XACK* on the Multibus.

If the Multibus signal LOCK* is active when a Multibus-to-NuBus conversion is performed, the NuBus is locked until LOCK* is inactive.

1.5.1.3 Interrupt Translation The SDU can translate Multibus interrupts into NuBus interrupts. The SDU's 8088 microprocessor receives all Multibus interrupts in a non-bus-vectored fashion (explained in the Multibus Specification). When programmed, the 8088 can respond to Multibus interrupts by writing to an arbitrary NuBus address (or addresses). Since NuBus interrupts are specifically addressed writes with the least significant bit set to 1, the Multibus interrupt can be translated into a NuBus interrupt.

The SDU can also translate NuBus interrupts into Multibus non-bus-vectored interrupts. On the SDU, the Multibus address space is contained within the NuBus address space. The Multibus interrupt register, which lies in this Multibus address space, can drive any of the eight Multibus interrupt lines active. Therefore, NuBus writes to the Multibus interrupt register which have the least significant bit set to 1 are translated into Multibus interrupts.

1.5.2 NuBus Central Features

1.5.2.1 System Clock The SDU is the source of the 75 percent duty cycle, 10 megahertz system clock (CLK*), to which all bus operations are synchronized. Under software control, the system clock rate can be increased or decreased by 7 percent for diagnostic purposes.

1.5.2.2 Time-Out Recovery The SDU optionally provides NuBus time-out recovery by monitoring the time between the START* and ACK* control signals. If more than the programmed number of clock cycles (up to 256) occur, the SDU asserts the ACK* signal with the appropriate TMO and TMI code for a time-out.

1.5.3 Nonvolatile Features

The SDU contains 2K bytes of battery backed-up CMOS RAM. This memory is used to store the system configuration information in a nonvolatile manner. The SDU also provides a battery backed-up time-of-day clock.

1.5.4 1/4-Inch Tape Interface

The SDU contains a 1/4-inch streaming tape drive interface. This feature provides low-cost transportable media for the Nu Machine. Since this interface is on the SDU, only the SDU, power supply, and tape drive must be functioning properly to load the diagnostic routines from the 1/4-inch tape drive.

1.5.5 Interval Timer

The SDU contains a programmable timer for generating periodic events to specific CPUs. The timer can be used for many important system functions, such as process scheduling.

1.5.6 Debug/Diagnostic Facilities

The SDU provides the NuBus system operator with several diagnostic tools. The SDU Monitor of the SDU Operating System allows a terminal on either serial port to read and write bus locations and to initiate and execute SDU

SDU GENERAL DESCRIPTION

GENERAL DESCRIPTION

self-diagnostics. The NuBus diagnostic hardware verifies bus integrity.

1.5.7 System Status Display

On power-up, the SDU first runs a self-test, then a bus test, and finally individual board tests. The SDU uses the front panel LEDs to summarize the results of the tests; the detailed results can be read using the system console. An LED on each board is automatically turned on at power-up and is turned off by the SDU after the board-specific diagnostic passes.

1.5.8 Serial Ports

The SDU contains two serial communications ports, either of which can be used as the smart front panel/remote diagnostics port, depending on the position of the diagnostic rotary switch. Otherwise, they are both available as general-purpose serial ports.

1.5.9 Power Supply Interface

The interface to the power supply includes several lines in addition to the actual current-carrying cables. These lines are ACPF, DCOT, MARGIN, and ACOFF. The SDU provides the system interface to these lines. ACPF (ac power fail) is generated by the power supply. The SDU then posts events to the installed CPUs so that they can take appropriate action. DCOT (dc out of tolerance) indicates that the +5 volt supply is out of tolerance +/-5 percent. When this signal is active, the SDU generates a system reset. MARGIN is the SDU signal by which the +5 volt supply margin can be increased or decreased by 7 percent. ACOFF is a signal from the SDU to the ac distribution box by which the SDU can shut off the ac power.

1.5.10 System Bootstrap

The SDU Monitor of the SDU Operating System boots automatically upon power-up. The SDU then determines the location of the system console, either serial port or high resolution display, from the position of the rotary switch. This allows the system to be manually reset, reinitialized, or rebooted via SDU Monitor control. Refer to the Nu Machine SDU Operating System User Manual for the command descriptions.

The SDU can fully and automatically boot the Nu Machine Operating System when the rotary switch is properly set. The rotary switch positions are explained in both the Nu Machine Rack Module, General Description and the Nu Machine Office Module, General Description.

2.1 General

This section provides information and procedures for unpacking the SDU from its shipping container and installing it in a Nu Machine (TM)* system chassis. When the SDU is shipped in a chassis as part of a complete system, refer to the Nu Machine Unpacking and Inventory Guide for unpacking procedures.

This section covers installation details of the SDU board only. The procedures assume that the user has a fundamental knowledge of basic hand tools and cabling techniques, but they do not require a detailed understanding of computer hardware or software.

2.2 Unpacking/Packing the SDU Board

Upon receipt of the container, inspect it to ensure that no damage has occurred. If any damage is found, note the damage on the bill of lading and file claim against the carrier, if applicable. Photograph any damages to the equipment container.

CAUTION

The Nu Machine systems contain static-sensitive electronic components. To avoid damage to these components, ensure that you are well grounded before removing or handling the printed circuit boards.

Use a static-control system consisting of a static-control floor or table mat and a static-control wrist strap. These are commercially available. If you do not have a static-control system, you can discharge any accumulated static charge by touching a grounded object prior to handling a board.

* Nu Machine is a trademark of Texas Instruments Incorporated.

SDU GENERAL DESCRIPTION
INSTALLATION

It is imperative that you do not place the printed circuit board on top of its shipping bag. The conductive external surface of this bag will short the SDU board's energized pins on contact, resulting in a corrupted CMOS RAM. Always store or transport a printed circuit board inside its protective package.

After completion of the preliminary inspection, perform the following steps to remove the board from its container and prepare the computer for installation.

NOTE

Do not discard any packing materials until unpacking, inspection, and inventory are complete.

1. Remove the top cushion pad or other packing material.
2. Obtain the packing list. Inventory the items received against the packing list.
3. Pack all shipping materials into the original shipping container and store the container for reshipment of the unit.
4. Inspect the SDU board and components for signs of damage that may have occurred during shipment. If damage has occurred, notify the carrier immediately.

To repack the unit, reverse the above procedure using the original packing material.

2.3 SDU Installation Procedures

The following paragraphs describe the preparation and installation of the Nu Machine SDU board.

WARNING

Ensure that the chassis ac power cord is disconnected from ac power during installation. Failure to observe this precaution could result in severe electrical shock.

SDU GENERAL DESCRIPTION
INSTALLATION

1. Select the chassis slot for the SDU board. The SDU board must be installed in either slot 13, 14, or 15, whichever is the highest priority NuBus (TM)* slot in the chassis. Since the SDU generates clock signals for both the NuBus and Multibus (R)** interfaces, the selected slot must have a NuBus in the P1 position and, if the system is equipped with Multibus, a Multibus in the P2 position.
2. Install a 1 high by 1 wide connector plate assembly (TI P/N 2235471-0001) to provide the jack for the P3 board connector to plug into. Lower the I/O back panel to access the back chassis framework. Push the connector plate assembly into the selected chassis slot hole directly below the motherboard. Secure the assembly to the back chassis framework with screws.
3. Returning to the front of the chassis, slide the board into the selected chassis slot. Hold the board so that the inserter/ejector tabs on the front corners of the board are pushed out and the board components are to the right. When the board is fully inserted, the inserter/ejector tabs will snap over the board's locking pins to prevent it from vibrating out of the chassis.
4. For standard Nu Machine systems, install the Multibus priority jumper (TI P/N 2220779-0001) on pin A31 and pin B31 in P2 slot 15. Note that the SDU slot position can affect the jumper setting. As a rule, install the jumper on the first slot (either 14 or 15) which is the Multibus master. If the jumper is not installed in the correct slot, the Multibus cannot be enabled.
5. An SDU paddle board (TI P/N 2235465-0001) must be mounted on the P3 connector plate assembly which was installed in the step 2 above. To connect the SDU paddle board to the other system components, the SDU cables in Table 2-1 are required. Note that the cable length and part number can vary depending on the Nu Machine system. Consult your local sales representative for exact ordering information. Refer

* NuBus is a trademark of Texas Instruments Incorporated.

** Multibus is a registered trademark of Intel Corporation.

SDU GENERAL DESCRIPTION
INSTALLATION

to the Chassis Section of the Nu Machine Installation Manual for the SDU paddle board and interface cable installation procedures.

Table 2-1. Required SDU Cables

CABLE NAME	TI PART NUMBER
SDU Front Panel	2235232
SDU Rear Panel	2235233
SDU Serial Interface	2235416
SDU Power Supply	2235418
SDU Ac Power Control	2235419

3.1 General

This section describes SDU operation after installation in the chassis with appropriate peripherals and mass storage devices.

3.2 Fault Indication LED

The SDU card has a red LED mounted on its front edge. This LED should come on at power-up and go out at the successful completion of the SDU configuration ROM diagnostic.

3.3 Front Panel LEDs

There are three LEDs on the front panel that indicate the results of the SDU self-diagnostics. The green RUN LED, when lit, indicates that the processor has resumed execution after the self-diagnostics are successfully completed. The red ATTN LED goes out when the self-diagnostics are successfully completed. The red SET-UP LED goes out if the battery-powered SDU CMOS RAM is correctly initialized. A full explanation of self-diagnostics is given in Paragraph 3.6.

3.4 Back Panel Rotary Switch

The back panel rotary switch has five mode selector positions labeled 0 through 4. Each switch position selects a preprogrammed boot sequence device, system console, and data rates for the serial ports. If the battery-powered CMOS RAM is not set up, then default boot parameters must be used. Refer to either the Nu Machine (TM)* Rack Module, General Description or the Nu Machine Office Module, General Description for mode selector position information.

3.5 Reset Pushbutton and Signals

A reset pushbutton is located on the back panel of the chassis. This pushbutton resets the board-level SDU reset

* Nu Machine is a trademark of Texas Instruments Incorporated.

SDU GENERAL DESCRIPTION OPERATION

signal which initializes internal registers to the power-up state and also resets the on-board 8088.

NuBus (TM)* RESET* and Multibus (R)** INIT*, standard reset signals, are generated by the SDU but do not reset any SDU functions. These signals are driven active at SDU power-up and by explicitly resetting them through the SDU monitor. Setting the NuBus reset bit and the Multibus reset bit in control-status register 1 (CSR1) will be discussed in Paragraph 4.2.6.

Tape RESET* is the reset signal for the 1/4-inch tape interface. This signal is only driven active through the SDU Monitor. Setting the tape reset bit in CSR1 will be discussed in Paragraph 4.2.6.

3.6 Self-Diagnostics

The SDU automatically performs self-diagnostics at power-up. Manually resetting the system via the INIT command runs the same self-diagnostics. Besides the following self-diagnostics description, additional information can be found in the Nu Machine Diagnostic User Manual.

At reset, the SDU Monitor turns on all three front panel LEDs (ATTN, SET-UP and RUN) and then tests the on-board RAM. The monitor halts if the RAM test fails and leaves all LEDs lighted. If the RAM test passes, the monitor tests the:

- ◊ Programmable interrupt controller (PIC)
- ◊ Bus time-out register (BTO)
- ◊ Interrupt register
- ◊ Map RAM
- ◊ CMOS RAM CRC

Table 3-1 depicts the results of the self-test diagnostics indicated by the front panel LEDs. When the ATTN and SET-UP LEDs are out and the RUN LED is on, a >> prompt will appear

* NuBus is a trademark of Texas Instruments Incorporated.

** Multibus is a registered trademark of Intel Corporation.

on the SDU monitor console device indicating that the SDU and its operating system are operational.

If the ATTN LED remains on, the power-up self-test was not successfully completed. In this case, refer to the troubleshooting flow chart in either Nu Machine Rack Module, Field Theory and Maintenance or Nu Machine Office Module, Field Theory and Maintenance for help.

If the SET-UP LED remains on, the battery-powered SDU RAM was not correctly initialized. Refer to the section on system power-up in either the Nu Machine Rack Module, General Description or the Nu Machine Office Module, General Description for help.

Table 3-1. Self-Diagnostics Results

LEDs ON	CMOS RAM CRC PASSED	ALL OTHER TESTS PASSED
All	No	No
SET-UP only	No	Yes
None	Yes	No
RUN only	Yes	Yes

4.1 General

This section describes the SDU Multibus(R)* and NuBus (TM)** address spaces. If the programmer wishes to alter the 8088 software or investigate 8088 capabilities, begin by referring to the Nu Machine** SDU Development System User Guide.

4.2 Multibus Address Space Definition

The Multibus is a non-multiplexed, asynchronous bus which supports 8-bit and 16-bit data transfers. Multibus addresses are twenty bits long, providing one megabyte of address space. Multibus operation is explained in the IEEE Standard Microcomputer System Bus Specification (IEEE Std 796-1983).

Table 4-1 is a partial breakdown of the Multibus address space reserved for SDU use. The low 64K bytes of the SDU address space are RAM and the high 64K bytes are ROM.

Table 4-1. Multibus Address Space

FUNCTION	ADDRESS RANGE	COMMENTS
SDU RAM	0x00000-0x0FFFF	64K bytes
SDU Register Space	0x10000-0x1FFFF	Sparsely implemented, multifunction area
SDU ROM	0xF0000-0xFFFFF	64K bytes, byte addressable

The SDU register space (0x10000-0x1FFFF) contains the various control registers and special functions implemented

* Multibus is a registered trademark of Intel Corporation.

** NuBus and Nu Machine are trademarks of Texas Instruments Incorporated.

**SDU GENERAL DESCRIPTION
PROGRAMMING**

on the SDU. Table 4-2 details the addresses and related functions in the SDU register space.

Table 4-2. SDU Register Space Definition

FUNCTION	ADDRESS RANGE	COMMENTS
Address map	0x18000-0x18FFF	1024-entry page map
Front panel LEDs/ac shutdown	0x1C080	
Back panel switch	0x1C084	Read only
CSR1	0x1C088	See CSR desc.
CSR0	0x1C08C	See CSR desc.
A/D converter	0x1C100-0x1C11C	Read only
CMOS TOD chip	0x1C120,0x1C124	See M146818 desc.
Remote serial port	0x1C150,0x1C154	See Intel 8251A desc.
Local serial port	0x1C158,0x1C15C	See Intel 8251A desc.
Interval timer #1	0x1C160-0x1C16C	See Intel 8253 desc.
Interval timer #0	0x1C170-0x1C17C	See Intel 8253 desc.
Time-out register	0x1C180	NuBus time-out preset
1/4-inch tape interface	0x1C1A0 and 0x1C600-0x1C7FF	See <u>Cipher Tape Manual</u>
Bus integrity registers	0x1C1A8-0x1C1BC	
Interrupt controller #0	0x1C1C0,0x1C1C4	See Intel 8259A desc.
Interrupt controller #1	0x1C1C8,0x1C1CC	See Intel 8259A desc.
Interrupt controller #2	0x1C1D0,0x1C1D4	See Intel 8259A desc.
Multibus interrupt register	0x1C1E0-0x1C1FC	8-bit addressable latch
CMOS RAM	0x1E000-0x1FFFC	2K byte RAM, low byte only

The following paragraphs describe each address space and corresponding memory-mapped function in detail.

4.2.1 SDU ROM

The Multibus addresses from 0xF0000 to 0xFFFFF access the SDU ROM. This ROM contains the SDU Monitor of the SDU Operating System and certain drivers associated with SDU functions. Refer to the SDU Operating System User Manual for detailed SDU ROM information. The ROM consists of up to four 27128 EPROMS and Multibus interface logic. Each 27128 contains 16K bytes of information. Table 4-3 relates ROM address ranges to board locations. The SDU ROM can only be accessed in bytes; 16-bit accesses will not provide meaningful data.

Table 4-3. ROM Address Allocation

ROM NUMBER	ADDRESS RANGE	REFERENCE LOCATION
1	0xF0000-0xF3FFF	U43
2	0xF4000-0xF7FFF	U6
3	0xF8000-0xFBFFF	U42
4	0xFC000-0xFFFFF	U5

The 27128s have a 200 nanosecond access time but the actual SDU ROM response time is between 300 and 400 nanoseconds due to logic and synchronization delays.

4.2.2 SDU RAM

The SDU RAM is accessed by the Multibus addresses 0x00000 through 0x0FFFF. The RAM consists of eight 4164 64K byte by 1-bit dynamic RAMs, an 8203 dynamic RAM controller, Multibus interface logic, a PAL state machine, and termination resistors. The PAL state machine and the interface logic translate a 16-bit operation into two byte operations (low byte first); so the RAM supports Multibus transfers of both sizes. The 8203 automatically refreshes the DRAMs. The bus cycle time to read or write the RAM is between approximately 500 and 750 nanoseconds. However, if a refresh cycle is in progress, the cycle will be delayed as required. Some areas of the SDU RAM are reserved for the SDU Monitor. The SDU Operating System Implementation Description details RAM allocation.

4.2.3 Address Map

Conversion of Multibus cycles into NuBus cycles requires mapping 20-bit addresses into 32-bit addresses. The SDU translates an arbitrary Multibus address into a NuBus address using a 24-bit by 1024-word static RAM array. Figure 4-1 depicts the format of this array.

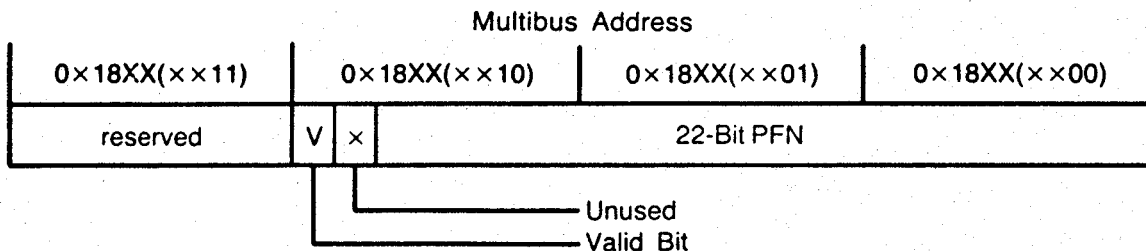
SDU GENERAL DESCRIPTION
PROGRAMMING

The array is accessed by Multibus addresses 0x18000 through 0x18FFF. Each map entry consists of four bytes; the most significant byte is unimplemented.

Bits 0 through 21 are used as page pointers in the NuBus address space during Multibus-to-NuBus translation. Bit 22 is unused. Bit 23, the valid bit, is set when a map entry contains a valid page (1024 bytes) pointer and cleared when the entry does not contain a valid page pointer.

The address map is readable and writable over the NuBus. Only byte transactions are performed; 16-bit operations are not supported.

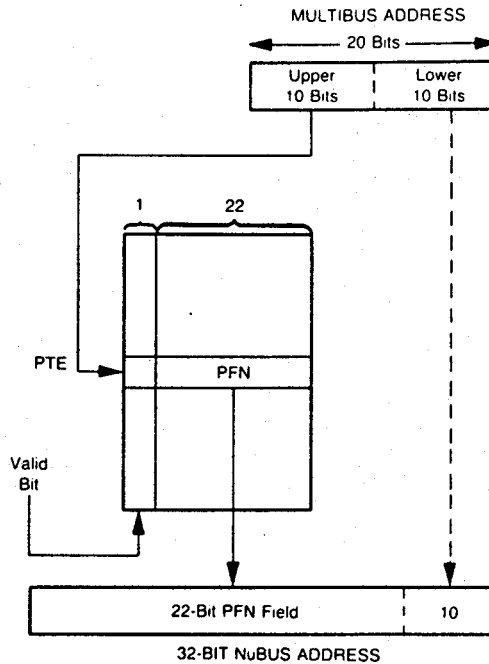
The address map contains undefined data at power-up. Valid bits in the map may or may not be set. All map entries must be set to a known assigned value prior to setting bit 1 in CSR0 for the NuBus interface. Failure to do so will result in spurious accesses to the NuBus during transfers to valid Multibus addresses.



NOTE:
 PFN = Page Frame Number

Figure 4-1. Address Map Format.

Figure 4-2 shows how Multibus addresses are translated into NuBus addresses using the map. The upper 10 bits of the Multibus address select one of the 1024 map entries. If the valid bit is set in the selected map entry, the Multibus operation is translated into a NuBus operation. The lower 22 bits of the selected map entry are used as the most significant bits of the NuBus address. The lower 10 bits of the NuBus address are the same as the lower 10 bits of the Multibus address.



NOTE:
PFN -- Page Frame Number
PTE -- Page Table Entry

Figure 4-2. Translation Definition.

Each map entry corresponds to a fixed page of Multibus addresses. For example, the map entry at locations 0x184C8, 0x184C9, and 0x184CA always corresponds to Multibus addresses 0x4C800 through 0x4CBFF. Care should be taken so that implemented Multibus addresses do not have the valid bit set in the corresponding map entry.

4.2.4 Front Panel LEDs

Multibus address 0x1C080 accesses the register for the front panel LEDs. This is a write only register; reading the register results in undefined data.

This register is one byte wide but only four bits are meaningful. Bits 0, 1, and 2 each correspond to a different front panel LED. The LED bits drive the front panel when they are 0. Thus, writing 1 to a particular bit position turns the corresponding LED off. Bit 7, the ac shutdown bit, turns the power off when it is 1. Thus, writing 0x80 to address 0x1C080 will cause the ac power to go off.

This register is cleared at power-up or SDU reset, enabling ac power and driving all three LEDs on. Figure 4-3 shows each bit position function for this register.

SDU GENERAL DESCRIPTION
PROGRAMMING

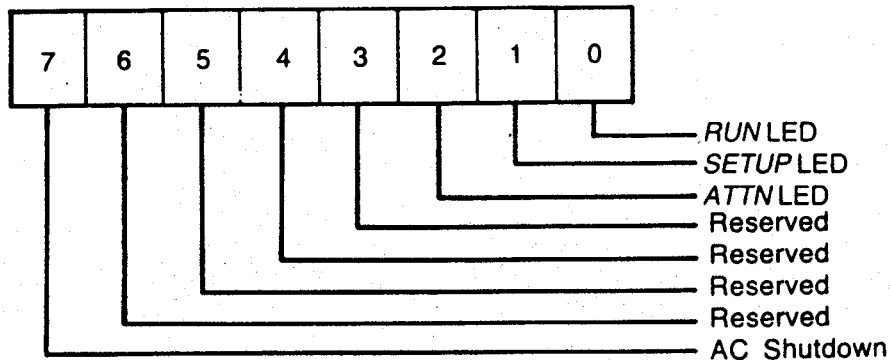


Figure 4-3. LED Register.

4.2.5 Back Panel Switch

The status of the back panel switch positions and the current NuBus slot position of the SDU are determined by reading Multibus address 0x1C084. This register can only be read; writing to this location has no effect.

The lower four bits (bits 0 to 3) convey the position of the back panel rotary switch. Table 4-4 shows the relationship between the five switch positions and bits 0 to 3 of 0x1C084. The upper four bits (bits 4 to 7) correspond to the NuBus slot identification lines ID0 to ID3. Figure 4-4 shows the switch register's bit positions and functions.

The back panel switch positions select both software and hardware options. The software options are described in both the Nu Machine Rack Module, General Description and the Nu Machine Office Module, General Description. Switch positions 0, 2, 3, and 4 enable the remote serial port to generate the SDU reset signal when it receives a break character; switch position 1 disables this function. Switch position 2 enables the on-board deadman timer; all other switch positions disable this function. Refer to Paragraph 1.3.1, 8088 and Related Hardware, for more information on the deadman timer.

Table 4-4. 1C084 Bit Patterns

SWITCH POSITION	SWITCH3	SWITCH2	SWITCH1	SWITCH0
0	1	1	1	1
1	1	1	1	0
2	1	1	0	1
3	1	0	1	1
4	0	1	1	1

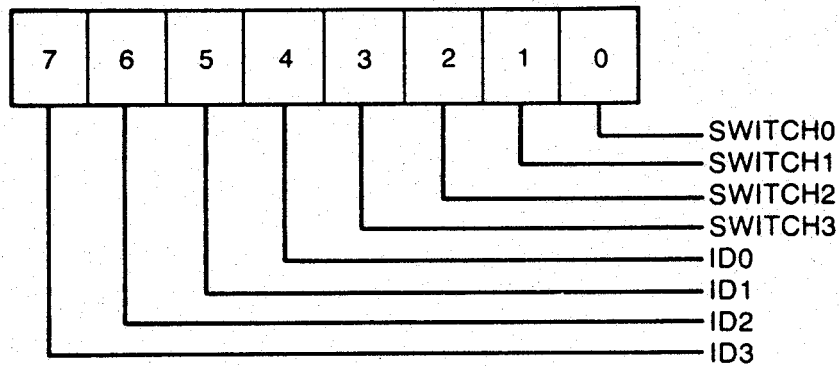


Figure 4-4. Switch Register.

4.2.6 Control-Status Register 1

Multibus address 0x1C088 accesses a multipurpose control and status register. This register is both readable and writable, but the data read is not always the same as the data written. This register is cleared at power-up and at SDU reset. Figures 4-5 and 4-6 show each bit position function in CSR1 for both reads and writes.

**SDU GENERAL DESCRIPTION
PROGRAMMING**

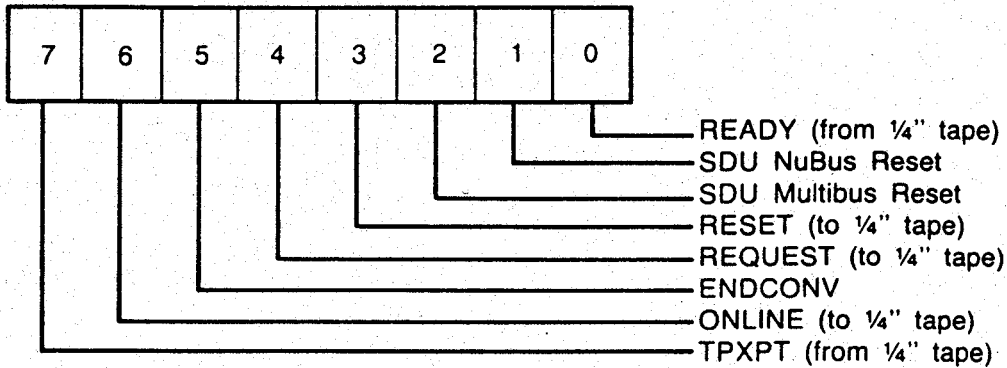


Figure 4-5. CSRI Read Format.

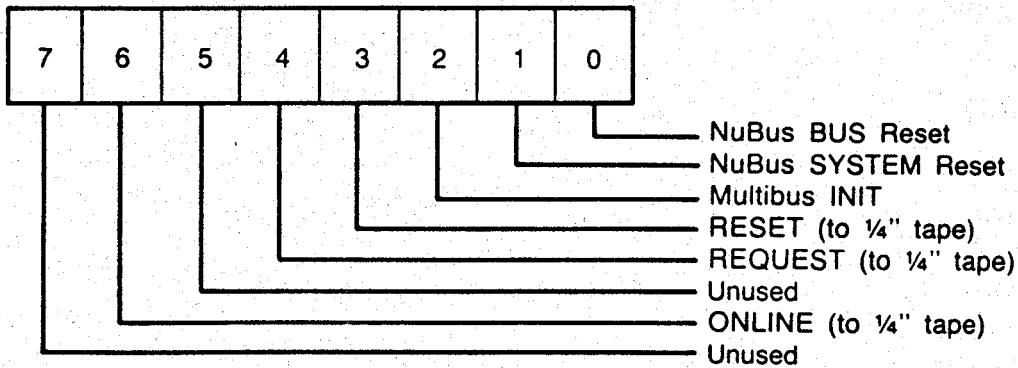


Figure 4-6. CSRI Write Format.

Bits 0, 3, 4, 6, and 7, when read, convey the status of some 1/4-inch tape interface control lines. Bits 3, 4, and 6, when written, drive corresponding bits on the tape interface. These bits are all active high.

The QIC-02 Interface Specification explains the meaning of these tape interface control lines. Note that all SDU tape

control and status bits are inverted from the corresponding bits shown in the QIC-02 Specification. For example, writing 1 to CSR1 bit 6 will cause the QIC-02 signal ONLINE* to go active (low). If CSR1 is read and bit 7 is 1, an exception condition exists and the signal EXCEPTION* from the tape is active (low).

Writing 1 to bit 0 drives the signal NuBus RESET* active for a single bus cycle.

Writing 1 to bit 1 drives NuBus RESET* active; it remains active until 0 is written to this bit. When read, this bit conveys the state of bit 1 but not necessarily the state of NuBus RESET*.

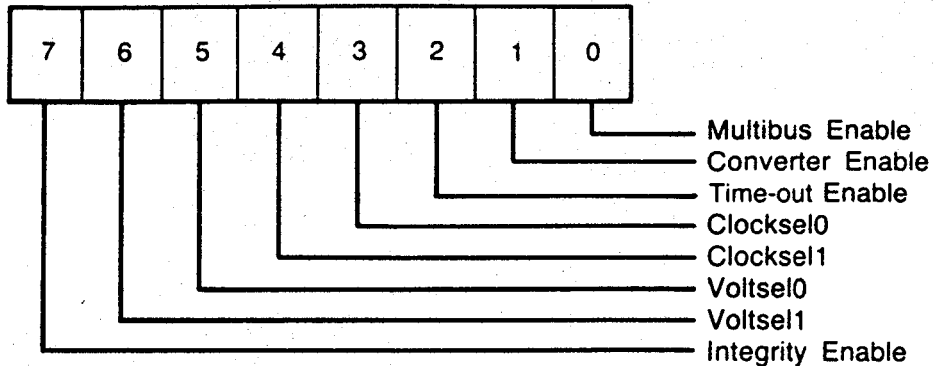
Bit 2 is similar to bit 1, except that it drives Multibus INIT* active. Reading this bit does not necessarily convey the state of INIT*.

Bit 5 is a status flag from the SDU analog-to-digital (A/D) converter which, when active high, indicates that the A/D converter has completed a conversion and the A/D output is valid. Bits 5 and 7 are not connected on the register output; writing data to them has no effect.

4.2.7 Control-Status Register 0

Multibus address 0x1C08C is a second multipurpose control and status register. This register is readable and writable, and the read data indicates the state of each written bit. This register is cleared at power-up and at SDU reset. Figure 4-7 summarizes each bit's function in CSR0.

SDU GENERAL DESCRIPTION
PROGRAMMING



Notes:

1) Clocksel1	Clocksel0	Bus Clock
0	0	10 MHz
0	1	9 MHz
1	0	11 MHz
1	1	Off-board source
2) Voltsel1	Voltsel0	+ 5 V dc status
0	0	Normal
0	1	Margin high
1	0	Margin low
1	1	Transition

Figure 4-7. Control-Status Register 0.

Bit 0 of CSR0 is the Multibus enable bit. When this bit is 1, the interface between the SDU internal Multibus and the external Multibus is enabled. Under these conditions, the internal Multibus and external Multibus are effectively one Multibus, and the interface is transparent for all address, data, control, interrupt, and arbitration lines.

When this bit is 0, the internal Multibus and external Multibus become separated. At this point, activity on the one has no effect on the other. The SDU continues to drive the external Multibus timing signals (CCLK* and BCLK*), although all other interface connections are severed, until the Multibus priority input is forced on. Attempts to access the external Multibus when this interface is disabled result in Multibus time-outs; the target slave is not affected and no transfer acknowledge occurs.

Bit 1 of CSR0 is the converter enable bit. This bit enables conversions in both directions between the Multibus and

NuBus. When this bit is 0, the SDU is effectively isolated from the NuBus. Attempts to access the NuBus when this bit is 0 result in Multibus time-outs; no NuBus transaction can be initiated.

Note that CSR0 is cleared at power-up and at SDU reset. Therefore, the bus interfaces previously discussed are both disabled and must be explicitly enabled.

Bit 2 of CSR0 enables the SDU to generate NuBus time-outs. The time-out period depends on the value in the time-out register. If this bit is 0, time-outs on the NuBus must be generated by some other system component. Refer to Paragraph 4.2.12, Time-Out Register, for more details.

Bits 3 and 4 of CSR0 select the rate at which the SDU drives the NuBus clock. Figure 4-7 shows the clock rate selected by the various bit patterns. At power-up or SDU reset the normal 10 megahertz clock rate is selected. The 9 megahertz and 11 megahertz clock rates are for diagnostic purposes only and should not be used in normal operation. These clock rates are intended to stress the system and demonstrate operational tolerances. If both bit 3 and bit 4 are high, an off-board clock source, if connected, is selected.

Bits 5 and 6 of CSR0 select the voltage level of the system's +5 volts dc supply. Figure 4-7 shows the voltage level selected by each bit pattern. Note that when the voltage is changed from a marginal to normal value, both bits 5 and 6 should be in a transition state of 1 for no less than 500 milliseconds before clearing them. This period allows the power supply to return to its normal output without enabling the dc out-of-tolerance signal. Failure to go through this transition state can result in spurious SDU resets.

Bit 7 of CSR0 enables SDU diagnostic registers to drive and sense the level of NuBus lines. Paragraph 4.2.14, Bus Integrity Registers, details the relationship between the integrity registers and particular NuBus signals. When bit 7 of CSR0 is 1, the contents of the integrity registers are statically and unconditionally enabled onto the NuBus. Therefore, this bit should only be turned on when the NuBus is in a diagnostic state and, preferably, with NuBus RESET* active.

4.2.8 A/D Converter

The SDU board has an 8-channel A/D converter which could be used for sensing levels of external thermal sensors. The reference inputs to the A/D are GND and +5. Thus, 0x00

SDU GENERAL DESCRIPTION PROGRAMMING

indicates that the selected channel is connected to ground or a very low voltage. 0xFF indicates that the channel is connected to a +5 or higher voltage.

To select a channel and read the digitized value:

- ◆ Arbitrary data is written to the Multibus address of the desired channel. This initiates the analog signal digitization corresponding to that channel.
- ◆ When the signal ENDCONV (bit 5 of CSR1) goes active (high), the digitized data is available from the A/D.
- ◆ The digital value is retrieved from the A/D by reading any valid A/D address.

Table 4-5 shows the relationship between Multibus addresses and A/D channels. Note that A/D channel selection occurs on write cycles only, and any A/D address can be used to read the data.

Table 4-5. A/D Converter Addresses

MULTIBUS ADDRESS	A/D CHANNEL	I/O PINS
0x1C100	0	P003 A12
0x1C104	1	P003 B9
0x1C108	2	P003 B10
0x1C10C	3	P003 B11
0x1C110	4	Ground
0x1C114	5	P003 B13
0x1C118	6	P003 B14
0x1C11C	7	+5 volts dc

4.2.9 CMOS TOD Chip

The SDU battery-powered time-of-day chip is a Motorola MC146818. The MC146818 data sheet gives a complete chip description and programming information. The MC146818 can be viewed as an array with 64 byte registers with addresses from 0 to 63. The first 14 bytes (0 to 13) have special purposes, while the remaining 50 bytes are general purpose. Data transfer to this chip requires two bus transactions: writing the register's address to Multibus address 0x1C124, then reading/writing data from/to Multibus address 0x1C120.

4.2.10 RS-232C Serial Ports

The SDU has two RS-232C serial communications ports: the remote serial port and the local serial port. Each serial port consists of an 8251A programmable communications

interface (PCI) and interface logic for receiving and driving RS-232C levels. Both ports operate in asynchronous mode only. The remote serial port drives the SDU reset signal active if a break character is detected on the received data input while the back panel switch is in position 0, 2, 3, or 4. Each serial port is connected to a 25-pin, D-shell connector on the Nu Machine back I/O panel. The remote port is connected to a male connector; the local port is connected to a female connector. Refer to either the Nu Machine Rack Module, General Description or the Nu Machine Office Module, General Description for details on the SDU paddle card and its interface cable to the serial ports.

Appendix D shows the relationship between D-shell pin assignments and 8251A signals. Two Multibus addresses are associated with each port: 1) a command register for initialization, setting control parameters, and reading status, and 2) a data register for reading/writing received/transmitted data. The 8251A data sheet gives detailed programming information. Table 4-6 shows the Multibus addresses for accessing serial port registers.

Table 4-6. Multibus Addresses to Serial Port Registers

MULTIBUS ADDRESS	DESCRIPTION
0x1C150	Remote serial port data register
0x1C154	Remote serial port command/status register
0x1C158	Local serial port data register
0x1C15C	Local serial port command/status register

4.2.11 Programmable Interval Timers

Two Intel 8253 Programmable Interval Timers (PITs) are used on the SDU: PIT0 and PIT1. Each PIT has three timers on it each with an input clock rate of 1.2288 megahertz. The 8253 data sheet gives detailed programming information. Table 4-7 shows the relationship of Multibus addresses to PIT registers and each PIT's function.

SDU GENERAL DESCRIPTION
PROGRAMMING

Table 4-7. PIT Address Map

MULTIBUS ADDRESS	REGISTER DEFINITION	COUNTER FUNCTION
0x1C160	PIT1 counter #0	Periodic interrupt #0
0x1C164	PIT1 counter #1	Periodic interrupt #1
0x1C168	PIT1 counter #2	Periodic interrupt #2
0x1C16C	PIT1 mode	None
0x1C170	PIT0 counter #0	Remote serial port rate generator
0x1C174	PIT0 counter #1	Local serial port rate generator
0x1C178	PIT0 counter #2	Unused
0x1C17C	PIT0 mode	None

PITs operating as periodic interrupts go to interrupt controller #1 at interrupt levels 4, 5, and 6 (refer to Paragraph 4.2.15, Interrupt Controllers). PITs operating as rate generators drive the receive and transmit clocks of the serial ports. Note that the rate generator PITs must be programmed in the square wave rate generator mode.

The serial data rate is determined by both PIT0 and the 8251A programming. For example, suppose a 9600 baud rate is required, and the 8251A is in 16X mode. Since clock input is 16 times the data rate, the 8251A should have a clock rate of 153600 (16 X 9600) hertz, and the PIT should be programmed to divide the input clock rate (1.2288 megahertz) by 8 ($1228800/8 = 153600$).

4.2.12 Time-Out Register

Multibus address 0x1C180 accesses the NuBus time-out register on the SDU. This register is both readable and writable. Setting this register to 0 establishes the maximum NuBus time-out period of 25.6 microseconds. Shorter time-out periods (from 0 to 25.6 microseconds) are achieved by writing numbers greater than 0 into this register. For example, if 0x80 is written into the time-out register, the NuBus time-out is set to 12.8 microseconds. This register is cleared at power-up and on SDU resets. Note that NuBus time-outs are only enabled by setting the time-out bit (bit 2) in CSR0.

4.2.13 1/4-Inch Tape Interface

The SDU 1/4-inch tape interface is memory-mapped in Multibus address space. The tape interface state machine translates reads of Multibus address 0x1C1A0 into tape status reads. Writes to this address are translated into command transfers to the tape drive. Multibus addresses in the range from

0x1C600 to 0x1C7FF are translated into string data transfers to or from the tape drive. Only byte transfers are supported.

The tape interface control signals RESET*, ONLINE*, and REQUEST* are determined by bits in CSR1. Paragraph 4.2.6, CSR1, relates these control signals and bit positions. The control signal XFER* is driven by a PAL state machine, which translates Multibus control lines into XFER* and vice versa. The status lines EXCEPTION* and READY* can be sensed by reading CSR1 or received as interrupts on PIC0. The signals DIRC* and ACK* from the tape are used strictly by the tape interface state machine.

For a complete description of the SDU 1/4-inch tape interface, refer to the QIC-02 Interface Specification Rev. D (9/23/82).

4.2.14 Bus Integrity Registers

The SDU can statically drive NuBus signal lines and then read the status of these lines to determine if any NuBus signals are shorted high, low, or together. The bus integrity registers statically and unconditionally drive the NuBus if the integrity enable bit (bit 7) in CSR0 is 1. Therefore, the integrity logic should only be activated if the NuBus is quiescent, preferably with the SDU holding NuBus RESET* active. The integrity logic is in six Multibus address space registers. Each of these registers is readable and writable. If the NuBus signal lines are not shorted, the data read should be the data written. Table 4-8 shows which Multibus address accesses which integrity register.

Table 4-8. Integrity Registers

MULTIBUS ADDRESS	REGISTER NUMBER
0x1C1A8	0
0x1C1AC	1
0x1C1B0	2
0x1C1B4	3
0x1C1B8	4
0x1C1BC	5

Figures 4-8 through 4-13 show the relationship between NuBus signal names and bit positions for each of the integrity registers.

SDU GENERAL DESCRIPTION
PROGRAMMING

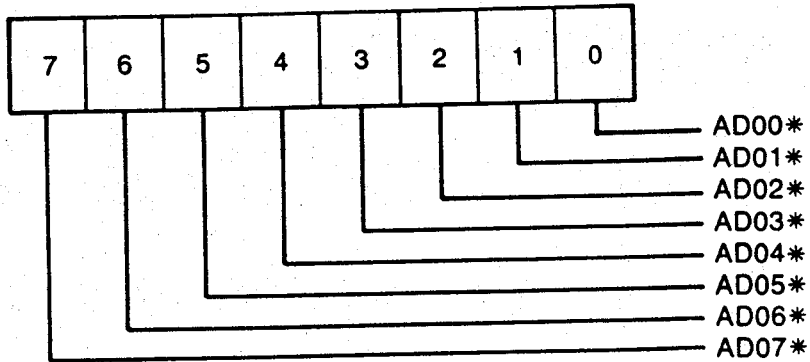


Figure 4-8. Integrity Register 0.

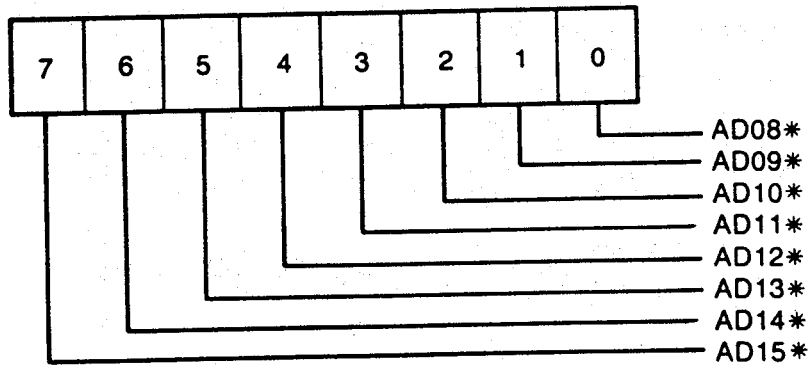


Figure 4-9. Integrity Register 1.

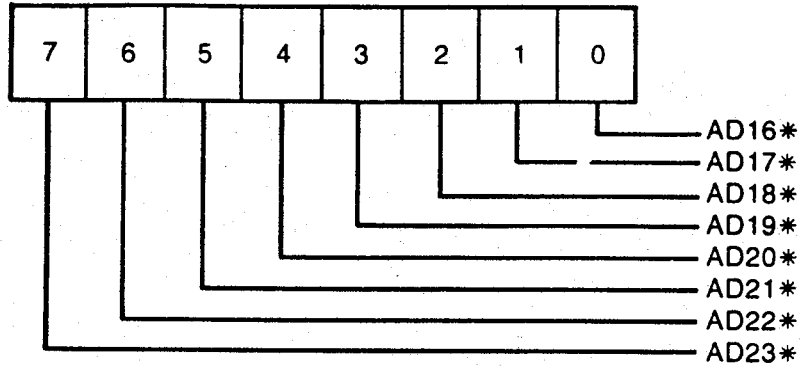


Figure 4-10. Integrity Register 2.

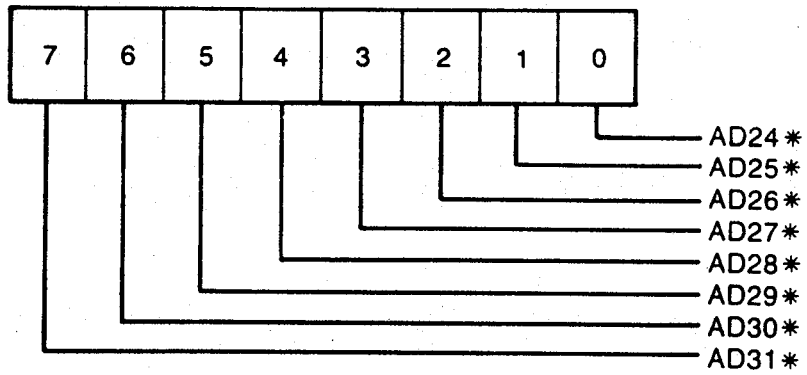


Figure 4-11. Integrity Register 3.

**SDU GENERAL DESCRIPTION
PROGRAMMING**

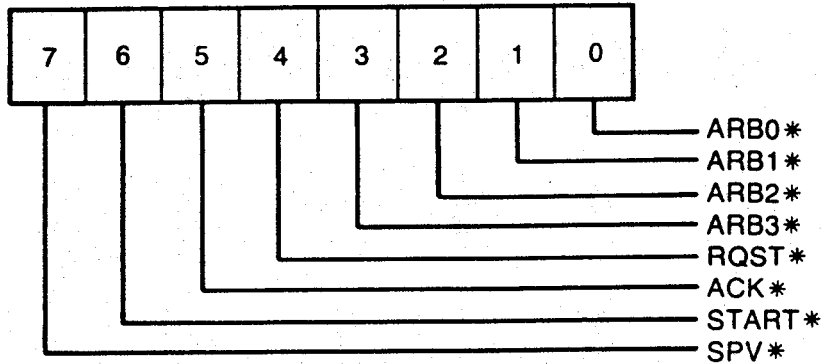


Figure 4-12. Integrity Register 4.

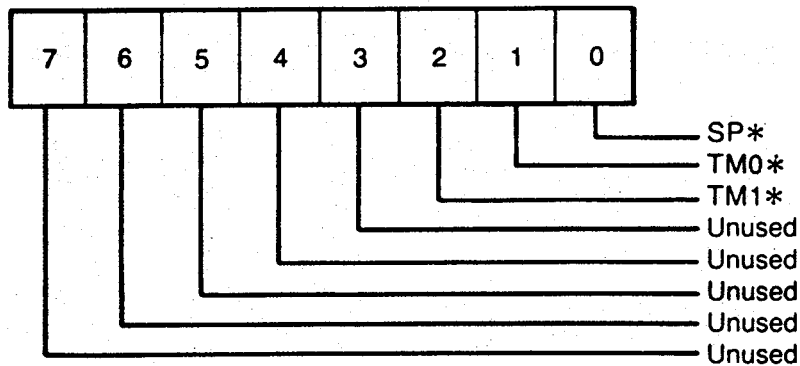


Figure 4-13. Integrity Register 5.

4.2.15 Interrupt Controllers

The SDU has three 8259A programmable interrupt controllers (PICs) which generate interrupts to the 8088. Two of them operate in the slave mode (PIC1, PIC2) and the third (PIC0) operates in the master mode. Detailed programming information is in the 8259A data sheet. The interrupts are

individually maskable, and the priority is also programmable. The interrupt requests can also be programmed to sense either edges (low-to-high transitions) or high levels on the PIC inputs. Table 4-9 through Table 4-11 relate each PIC interrupt number to its interrupt function.

Table 4-9. PIC0 Interrupt Functions

INTERRUPT #	SIGNAL	DESCRIPTION
0	MULTITO*	Multibus time-out
1	TIMEOUT*	NuBus time-out
2	TPXPT*	1/4-inch tape EXCEPTION* input
3	TPRDY*	1/4-inch tape READY* input
4	ACLOINT*	Ac power fail from power supply
5	RSVD*	Reserved
6	PIC2INT*	Interrupt from slave PIC2
7	PIC1INT*	Interrupt from slave PIC1

The MULTITO signal indicates that a Multibus time-out has occurred. Multibus time-outs are generated if the 8088 does not access a bus for longer than 3 milliseconds. Thus, MULTITO will go active if the 8088 executes a halt instruction or if the 8088 is denied Multibus access for longer than the time-out period. Note that Multibus time-outs do not cause any transfer acknowledge on the Multibus. The Multibus does not support error acknowledgment.

The SDU generates a NuBus time-out when the time-out enable bit in CSR0 is 1 and either: 1) A START* cycle occurs on the NuBus and an ACK* cycle has not occurred for longer than the programmed time-out period, or 2) RQST* has been active but a START* has not occurred for longer than the programmed time-out period. In the first situation, the SDU generates an ACK* with a bus time-out code on the TMx* lines. In the second case, the SDU generates an idle cycle to reinitiate arbitration. A time-out interrupt indicates one of these cases.

The TPXPT interrupt indicates an exception condition from the 1/4-inch tape controller. This interrupt is asserted at the high-to-low transition of the EXCEPTION* signal from the tape controller. Refer to the QIC-02 Interface Specification for implications of activity on this line.

TPRDY is similar to TPXPT except that it indicates that the READY* signal on the 1/4-inch tape interface has made a high-to-low transition.

SDU GENERAL DESCRIPTION
PROGRAMMING

ACLOINT indicates that the ac power to the +5 volt power supply is below a specified level. This interrupt is asserted whenever the power supply ACLO line goes from the low-to-high state.

The slave interrupt controllers, PIC1 and PIC2 are cascaded through the master interrupt controller PIC0. The slave interrupt controllers generate PIC1INT and PIC2INT interrupts to the master. These interrupts indicate that there is an active input on one or more of the respective interrupt lines.

Table 4-10. PIC1 Interrupt Functions

INTERRUPT #	SIGNAL	DESCRIPTION
0	RXRDY0*	Remote serial port receive ready
1	TXRDY0*	Remote serial port transmit ready
2	RXRDY1*	Local serial port receive ready
3	TXRDY1*	Local serial port transmit ready
4	PITINT0*	Periodic interrupt 0
5	PITINT1*	Periodic interrupt 1
6	PITINT2*	Periodic interrupt 2
7	Spare	

RXRDY0 is a signal from the remote serial port indicating that a character is in the receive buffer and ready for processing. This signal is connected directly to pin 14 of the 8251A associated with the remote serial port.

TXRDY0 is a signal from the remote serial port indicating that the transmitter is ready to accept a data character. This signal is connected directly to pin 15 of the 8251A associated with the remote serial port.

RXRDY1 and TXRDY1 are similar to RXRDY0 and TXRDY0 except that they pertain to the local serial port rather than the remote serial port.

PITINT0, PITINT1, and PITINT2 are all interrupt inputs from PIT1. These general system timers can generate interrupts from once every 1.62 microseconds to once every 53 milliseconds. Paragraph 4.2.11, Programmable Interval Timers, gives more information on the interrupt timer.

Table 4-11. PIC2 Interrupt Functions

INTERRUPT #	SIGNAL	DESCRIPTION
0	LOCALINT0*	Multibus interrupt 0
1	LOCALINT1*	Multibus interrupt 1
2	LOCALINT2*	Multibus interrupt 2
3	LOCALINT3*	Multibus interrupt 3
4	LOCALINT4*	Multibus interrupt 4
5	LOCALINT5*	Multibus interrupt 5
6	LOCALINT6*	Multibus interrupt 6
7	LOCALINT7*	Multibus interrupt 7

The inputs to PIC2 are driven by a three-state buffer. The inputs to the buffer are the Multibus interrupt lines INT0* through INT7*. The buffer is activated only when the Multibus enable bit (bit 0) of CSR0 is 1.

The PICs are programmed by writing initialization command words (ICWs) and operation control words (OCWs), as required, into the PIC. Details on these control words and the programming sequence are in the 8259A data sheet. Note in the data sheet a signal, A0, which differentiates between the first ICW (or OCW) and subsequent control words. In the SDU implementation, the signal A0 is driven by Multibus address bit #2 (MADDR02). Table 4-12 relates Multibus addresses and each PIC's control word locations.

Table 4-12. PIC Addresses

MULTIBUS ADDRESS	FUNCTION	CONTROLLER NUMBER
0x1C1C0	ICW1, OCW1	PIC0
0x1C1C4	ICW2-ICW4, OCW2-OCW3	PIC0
0x1C1C8	ICW1, OCW1	PIC1
0x1C1CC	ICW2-ICW4, OCW2-OCW3	PIC1
0x1C1D0	ICW1, OCW1	PIC2
0x1C1D4	ICW2-ICW4, OCW2-OCW3	PIC2

4.2.16 Multibus Interrupt Register

The Multibus has eight interrupt lines (INT0* to INT7*). The SDU can sense any of these lines through PIC2 as described in the preceding paragraph and can drive any of these lines. Each interrupt has a corresponding Multibus address as Table 4-13 shows.

SDU GENERAL DESCRIPTION
PROGRAMMING

Table 4-13. Interrupt Addressing

MULTIBUS ADDRESS	MULTIBUS INTERRUPT LEVEL
0x1C1E0	0
0x1C1E4	1
0x1C1E8	2
0x1C1EC	3
0x1C1F0	4
0x1C1F4	5
0x1C1F8	6
0x1C1FC	7

Each Multibus interrupt line can be driven active by writing a byte to the corresponding Multibus address with the least significant data bit set to 1. The interrupt can be cleared by writing 0 to the least significant bit at the same location. The status of the interrupt register can be determined by reading address 0x1C1E0. Each bit of this register corresponds to a particular Multibus interrupt bit. For example, if 0x00 is written to 0x1C1E0 through 0x1C1F0, 0x01 is written to 0x1C1F4, and 0x00 is written to 0x1C1F8 and 0x1C1FC, Multibus interrupt 5 will be active and reading 0x1C1E0 will produce 0x20. This interrupt can be cleared by writing 0x00 to 0x1C1F4. Figure 4-14 shows the format of 0x1C1E0 when it is read.

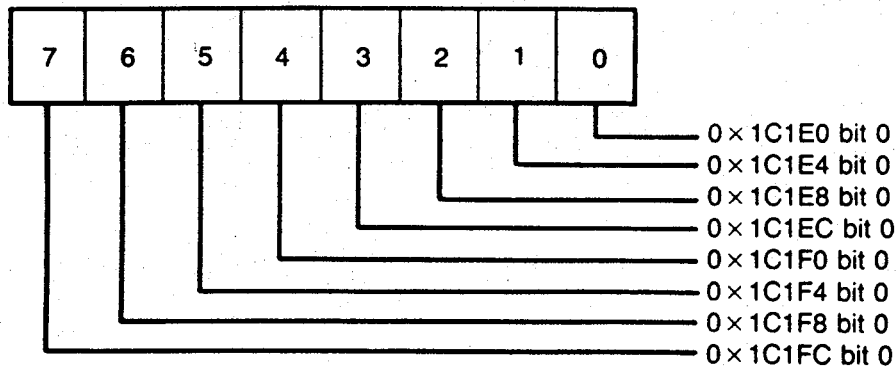


Figure 4-14. 0x1C1E0 Read Format.

4.2.17 CMOS RAM

The SDU battery backed up CMOS RAM is accessed by Multibus addresses 0x1E000 through 0x1FFFC. Only every fourth byte address is a valid RAM address. Thus, byte 0 of the CMOS RAM is at Multibus address 0x1E000, byte 1 is at 0x1E004, and so forth. A total of 2048 byte locations are provided. Figure 4-15 illustrates the CMOS address space. The battery backed up RAM is used to retain system setup and configuration information through power-down cycles. The SDU Operating System User Manual describes the data format stored in the CMOS RAM.

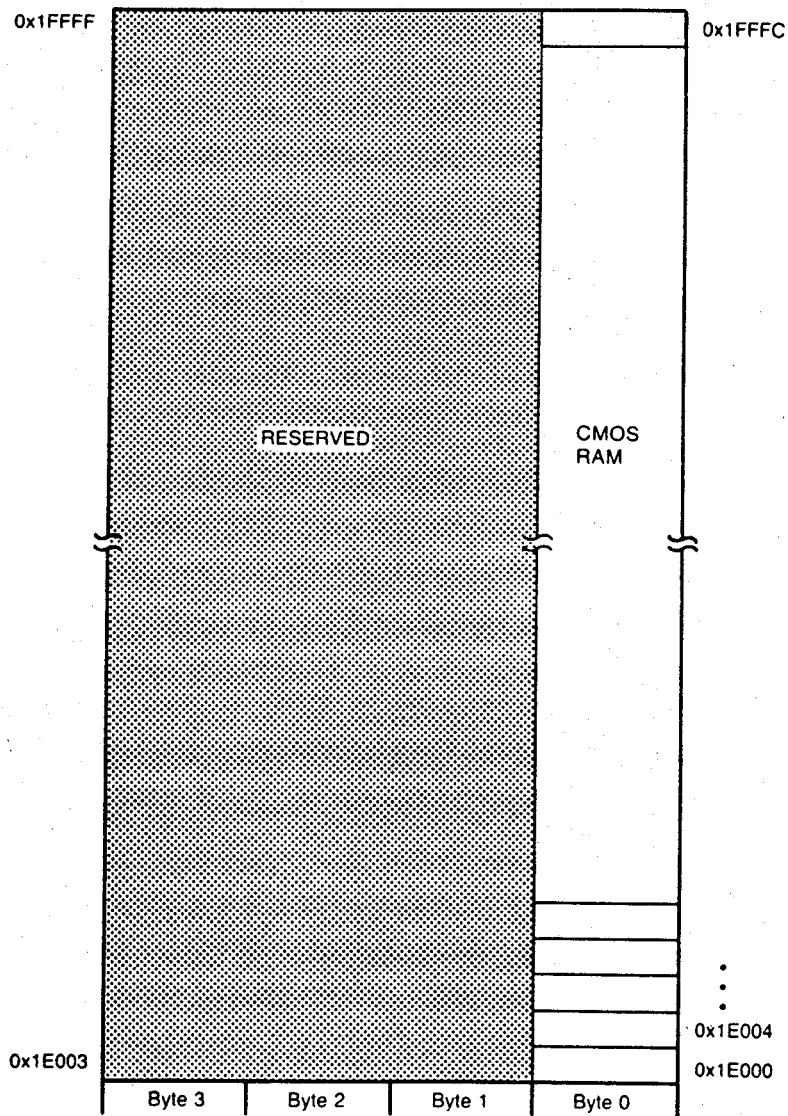


Figure 4-15. CMOS RAM Address Space.

**SDU GENERAL DESCRIPTION
PROGRAMMING**

4.3 NuBus Address Space Definition

The SDU is a NuBus slave. All valid SDU addresses are between NuBus address 0xF(ID)000000 and 0xF(ID)FFFFFF, where ID is the SDU hexadecimal slot identification number. The valid address spaces contained within the SDU's NuBus address space consist of three major blocks: Multibus memory space, Multibus I/O space, and configuration space. Figure 4-16 shows the breakdown of the SDU's NuBus addresses.

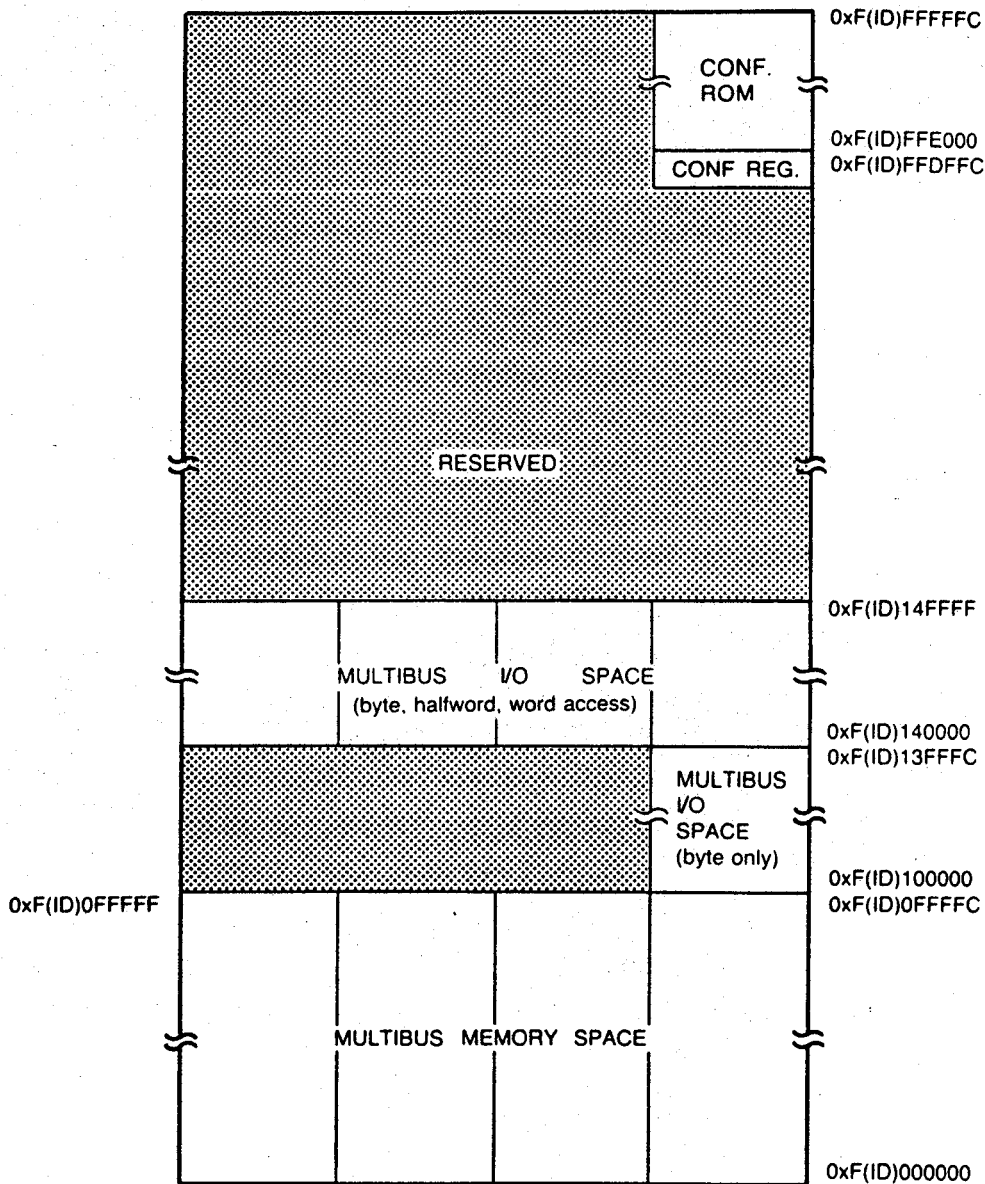


Figure 4-16. SDU NuBus Addresses.

4.3.1 Multibus Memory Space

NuBus addresses 0xF(ID)000000 through 0xF(ID)0FFFFFF directly access the Multibus memory space. The addressing is contiguous (no holes). Byte, halfword, and word transfers are supported across the interface. The NuBus address of a Multibus facility is simply the facility's Multibus address appended to 0xF(ID)0... (with AD0* and AD1* adjusted to indicate transfer size). Thus, SDU facilities on the Multibus are all available through the NuBus and have the same operational characteristics (for example, byte access and holes) when viewed through the bus converter as on the Multibus. Note that for any accesses through the NuBus-to-Multibus interface, the converter enable bit (bit 1) of CSRO must be set to 1--otherwise NuBus time-outs occur.

4.3.2 Multibus I/O Space

Two different mappings access the Multibus I/O space from the NuBus. NuBus addresses 0xF(ID)100000 through 0xF(ID)13FFFC are translated into Multibus I/O operations with only the low byte of each NuBus word mapped into a Multibus I/O port. Thus, the entire Multibus I/O address space is accessible in bytes from the NuBus. Byte, halfword, or word transfers can access the Multibus I/O space in this address range but only the low byte of each NuBus word is valid data. Read operations to this space are mapped into Multibus IORC* operations and writes to this space are mapped into IOWC* operations.

NuBus addresses 0xF(ID)140000 through 0xF(ID)14FFFF are mapped into Multibus I/O operations, as in Multibus memory space mapping, with all byte, halfword, and word transfers supported across the interface. This permits efficient data transfers to I/O locations which support 16-bit operations. Word transfers on the NuBus are translated into two 16-bit Multibus operations. Read operations to this space are mapped into Multibus IORC* operations and writes to this space are mapped into IOWC* operations.

4.3.3 Configuration Space

The configuration space on the SDU is made up of a configuration register and a configuration ROM. The configuration register's standard layout does not provide a reset or enable bit since the SDU performs system boot and is enabled at power-up. Bit 2 is the LED bit. Writing 1 to this bit turns the SDU LED on; writing 0 to this bit turns it off. This bit is turned on at power-up or SDU reset. The configuration register is at address 0xF(ID)FFDFFC.

The 2K-byte configuration ROM is justified to the low byte of each NuBus word. The configuration ROM addresses are 0xF(ID)FFE000 to 0xF(ID)FFFFFC. The configuration ROM

**SDU GENERAL DESCRIPTION
PROGRAMMING**

contains board-dependent information and start-up diagnostics. The Nu Generation Computer System Architecture Specification details the contents of this ROM.

4.4 Software Development

Both object and source licenses for the SDU Development System are available, as well as the SDU Operating System source. Contact your local sales representative for further information.

Software development information can be located in the following documents:

TITLE	PART NUMBER
<u>Nu Machine SDU Operating System User Manual</u>	2242811-0001
<u>Nu Machine SDU Operating System Implementation Description</u>	2242812-0001
<u>Nu Machine SDU Operating System Driver Design Guide</u>	2242813-0001
<u>Nu Machine SDU Development System, User Guide</u>	2242815-0001
<u>Nu Machine SDU Development System, Assembler Reference Manual</u>	2242816-0001

Appendix A PI PIN ASSIGNMENTS

The P1 connector detailed in Table A-1 has standard NuBus pin assignments. Signal line characteristics for the NuBus are described in the NuBus Specification.

Table A-1. NuBus Pin Assignments

PIN/ROW	A	B	C
1	-12	-12	RESET*
2	GND	GND	GND
3	SPV*	GND	+5
4	SP*	+5	+5
5	TM1*	+5	TM0*
6	AD1*	+5	AD0*
7	AD3*	+5	AD2*
8	AD5*	-5	AD4*
9	AD7*	-5	AD6*
10	AD9*	-5	AD8*
11	AD11*	-5	AD10*
12	AD13*	GND	AD12*
13	AD15*	GND	AD14*
14	AD17*	GND	AD16*
15	AD19*	GND	AD18*
16	AD21*	GND	AD20*
17	AD23*	GND	AD22*
18	AD25*	GND	AD24*
19	AD27*	GND	AD26*
20	AD29*	GND	AD28*
21	AD31*	GND	AD30*
22	GND	GND	GND
23	GND	GND	RSVD*
24	ARB1*	-5	ARB0*
25	ARB3*	-5	ARB2*
26	ID1*	-5	ID0*
27	ID3*	-5	ID2*
28	ACK*	+5	START*
29	+5	+5	+5
30	RQST*	GND	+5
31	GND	GND	GND
32	+12	+12	CLK*

Appendix B P2 PIN ASSIGNMENTS

The Multibus pin assignments are mapped into a DIN connector. Table B-1 shows the Multibus pin assignments on the P2 connector as they appear on the SDU. Multibus signal line characteristics are described in the Multibus Specification.

Table B-1. SDU Multibus Pin Assignments

PIN/ROW	A	B	C
1	AD17*	AD16*	AD15*
2	DAT0*	GND	AD14*
3	DAT2*	GND	DAT1*
4	DAT4*	GND	DAT3*
5	DAT6*	+5	DAT5*
6	DAT8*	+5	DAT7*
7	DATA*	+5	DAT9*
8	DATC*	+5	DATB*
9	DATE*	+5	DATD*
10	ADRO*	+5	DATF*
11	ADR2*	-5	ADR1*
12	ADR4*	GND	ADR3*
13	ADR6*	RSVD*	ADR5*
14	ADR8*	RSVD*	ADR7*
15	ADRA*	-12	ADR9*
16	ADRC*	GND	ADRB*
17	ADRE*	GND	ADRD*
18	INT0*	GND	ADRF*
19	INT2*	GND	INT1*
20	INT4*	+12	INT3*
21	INT6*	RSVD*	INT5*
22	INTA*	RSVD*	INT7*
23	CCLK*	GND	AD13*
24	CBRQ*	+5	AD12*
25	BHEN*	+5	AD11*
26	AACK*	+5	AD10*
27	XACK*	+5	INH2*
28	IORC*	+5	INH1*
29	MRDC*	GND	IOWC*
30	BUSY*	GND	MWTC*
31	BPRN*	GND	BREQ*
32	BCLK*	INIT*	BPRO*

Appendix C P3 PIN ASSIGNMENTS

The P3 pin assignments are used for I/O. Tables C-1 through C-3 describe the signals on this connector.

Table C-1. P3 Row A

PIN/ROW	SIGNAL	DESCRIPTION
A01	TPDAT0*	1/4-inch tape data bit 0
A02	TPDAT2*	1/4-inch tape data bit 2
A03	TPDAT4*	1/4-inch tape data bit 4
A04	TPDAT6*	1/4-inch tape data bit 6
A05	TPDIR*	1/4-inch tape DIRC* control line
A06	TPACK*	1/4-inch tape ACK* control line
A07	TPRDY*	1/4-inch tape READY* control line
A08	Open	
A09	SWITCH0	Back panel switch bit 0
A10	SWITCH2	Back panel switch bit 2
A11	RSVD	Reserved
A12	ADCCH0	A/D channel 0
A13	LED0	<u>RUN</u> LED on front panel
A14	LED2	<u>ATTN</u> LED on front panel
A15	GND	Logic ground
A16	Open	
A17	TXD0	Remote serial port transmit data
A18	DTR0	Remote serial port data terminal ready
A19	CTS0	Remote serial port clear to send
A20	TXD1	Local serial port transmit data
A21	DTR1	Local serial port data terminal ready
A22	CTS1	Local serial port clear to send
A23	GND	Logic ground
A24	Open	
A25	ACPF	Ac powerfail signal from power supply
A26	DCOT	Dc out-of-tolerance from power supply
A27	Open	
A28	MARG2	Margin control to power supply
A29	Open	
A30	PWROFF	Ac shutdown to ac distribution box
A31	GND	Logic ground
A32	Open	

SDU GENERAL DESCRIPTION
P3 PIN ASSIGNMENTS

Table C-2. P3 Row B

PIN/ROW	SIGNAL	DESCRIPTION
B01	TPXPT*	1/4-inch Tape EXCEPTION* control line
B02	GND	Logic ground
B03	GND	Logic ground
B04	Open	
B05	Open	
B06	Open	
B07	BFTPRST*	1/4-inch tape RESET* control line
B08	Open	
B09	ADCCH1	A/D channel 1
B10	ADCCH2	A/D channel 2
B11	ADCCH3	A/D channel 3
B12	GND	Logic ground
B13	ADCCH5	A/D channel 5
B14	ADCCH6	A/D channel 6
B15	ADCCH7	A/D channel 7
B16	GND	Logic ground
B17	Open	
B18	Open	
B19	GND	Logic ground
B20	Open	
B21	Open	
B22	Open	
B23	GND	Logic ground
B24	Open	
B25	Open	
B26	Open	
B27	Open	
B28	Open	
B29	Open	
B30	GND	Logic ground
B31	GND	Logic ground
B32	Open	

Table C-3. P3 Row C

PIN/ROW	SIGNAL	DESCRIPTION
C01	TPDAT1*	1/4-inch tape data bit 1
C02	TPDAT3*	1/4-inch tape data bit 3
C03	TPDAT5*	1/4-inch tape data bit 5
C04	TPDAT7*	1/4-inch tape data bit 7
C05	BFTPXFER*	1/4-inch tape XFER* control line
C06	BFTPRQST*	1/4-inch tape REQUEST* control line
C07	BFTPONLINE*	1/4-inch tape ONLINE* control line
C08	Open	
C09	SWITCH1	Back panel switch bit 1
C10	SWITCH3	Back panel switch bit 3
C11	RSVD	Reserved
C12	EXTRST*	Back panel reset switch
C13	LED1	<u>SETUP</u> LED on front panel
C14	RSVD	Reserved
C15	RSVD	Reserved
C16	Open	
C17	RTS0	Remote serial port request to send
C18	RXD0	Remote serial port received data
C19	DSR0	Remote serial port data set ready
C20	RTS1	Local serial port request to send
C21	RXD1	Local serial port received data
C2	DSR1	Local serial port data set ready
C23	GND	Logic ground
C24	Open	
C25	Open	
C26	Open	
C27	Open	
C28	Open	
C29	Open	
C30	Open	
C31	Open	
C32	Open	

The 1/4-inch tape interface signal line characteristics are described in the QIC-02 Interface Specification. The back control panel switch inputs to the SDU are all pulled up to +5 volts through 1K ohm resistors on the SDU. These signals are typically either open or shorted to ground by the external switch. The A/D channel inputs are high impedance inputs to an A/D converter. Locations for resistors in parallel with the A/D are provided on-board to facilitate voltage divider or controlled current circuitry.

The front panel LED outputs are driven by a 74S241 buffer. The LEDs turn on when these signal lines are driven low.

SDU GENERAL DESCRIPTION
P3 PIN ASSIGNMENTS

The LEDs should be connected to +5 volts through a current limiting resistor.

The serial port signal lines are all at standard RS-232C levels. The ac powerfail, dc out-of-tolerance, and margin signals are all determined by the power supply.

Normally, ac powerfail is logic 0. With ac removal, the powerfail signal goes to logic 1 at least 3 milliseconds before loss of dc output. On ac turn-on, this signal remains high until the output is in regulation.

Normally, dc out-of-tolerance is logic 0. When the output is under or over voltage by 5 percent, dc out-of-tolerance goes to logic 1. When the output goes back to within tolerance, this signal remains high for 100 to 500 milliseconds.

The SDU has the capability to margin the power supply by +/-7 percent. The system margin checking driver circuit is on the SDU board.

The ac shutdown signal is an open-collector output capable of sinking 10 milliamps. Signal levels up to 30 volts can be tolerated on this line.

Appendix D SERIAL PORT PIN ASSIGNMENTS

Table D-1. Remote Serial Port Pin Assignments

PIN/ROW	RS-232C DESIGNATION	SDU NAME	8251A PIN
1	Frame ground (AA)	Chassis ground	
2	Transmitted data (BA)	TXD	19
3	Received data (BB)	RXD	3
4	Request to send (CA)	RTS	23
5	Clear to send (CB)	CTS	17
6	Data set ready (CC)	Not used	
7	Signal ground (AB)	Logic ground	
8	Data carrier detect (CF)	DSR	22
9	Not used		
10	Not used		
11	Not used		
12	Not used		
13	Not used		
14	Not used		
15	Transmitter clock (DB)	Not used	
16	Not used		
17	Receiver clock (DD)	Not used	
18	Not used		
19	Not used		
20	Data terminal ready	DTR	24
21	Not used		
22	Not used		
23	Not used		
24	Not used		
25	Not used		

Note:

Connector is a 25-pin, D-shell, male connector.

SDU GENERAL DESCRIPTION
SERIAL PORT PIN ASSIGNMENTS

Table D-2. Local Serial Port Pin Assignments

PIN/ROW	RS-232C DESIGNATION	SDU NAME	8251A PIN
1	Frame ground (AA)	Chassis ground	
2	Received data (BB)	RXD	3
3	Transmitted data (BA)	TXD	19
4	Clear to send (CB)	CTS	17
5	Request to send (CA)	RTS	23
6	Data set ready (CC)	Pull-up	
7	Signal ground (AB)	Logic ground	
8	Data terminal ready	DTR	24
9	Not used		
10	Not used		
11	Not used		
12	Not used		
13	Not used		
14	Not used		
15	Receiver clock (DD)	Not used	
16	Not used		
17	Transmitter clock (DB)	Not used	
18	Not used		
19	Not used		
20	Data carrier detect (CF)	DSR	22
21	Not used		
22	Not used		
23	Not used		
24	Not used		
25	Not used		

Note:

Connector is a 25 pin, D-shell, female connector.

Appendix E MULTIBUS COMPLIANCE LEVELS

The Multibus Specification allows optional levels of compliance in several areas. This section describes Multibus attributes supported by the SDU.

Data Path The SDU supports a 16-bit data path on the Multibus. Byte swapping techniques are used to allow both 8-bit and 16-bit cards to work in the system.

Memory Address Path The SDU supports a 20-bit Multibus memory address path. Multibus cards using 16-bit or 24-bit addressing may not be compatible.

I/O Address Path The SDU supports a 16-bit I/O address path but does not act as an I/O slave. 8-bit I/O address path slaves are supported.

Interrupt Attributes The SDU supports only nonbus-vectorized interrupts over the Multibus.

Multibus Arbitration The SDU supports the Multibus daisy chain priority resolution scheme and drives CCLK* and BCLK* on the Multibus. CCLK* is driven at 10 megahertz but BCLK* (arbitration clock) is driven at 5 megahertz to permit up to seven potential masters on the Multibus. The SDU has two potential masters on it--the 8088 and the NuBus converter.

FOLD



NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 7886 SANTA ANA, CA

POSTAGE WILL BE PAID BY ADDRESSEE



TEXAS INSTRUMENTS

DATA SYSTEMS GROUP

ATTN: CUSTOMER TECHNICAL SUPPORT
17881 Cartwright Road
Irvine, CA 92714



FOLD

is e
manufactured by Mouse Systems Corporation. (The mechanical mice from CADR's will not work with the Lambda.) Setting up the mouse for use is a bit different than it was with the mechanical mice some of you may be used to.

CONTENTS

The mouse box should contain a mouse, a mouse pad, a bag containing extra felt pads and lenses, a technical manual from Mouse Systems, and a copy of this document. The box may have empty spaces for other equipment; these are for accessories which do not need to be used with the Lambda.

The mouse you receive may have either a DB-9P (9-pin D) connector, or an RJ-11C (telephone-type) connector. In either case, it will match the connector on the side of your monitor. (If it doesn't, let us know.)

THE MOUSE PAD

The metal object with blue and green lines printed on its surface is the MOUSE PAD. This is a special surface upon which the mouse must be moved. The mouse will not work correctly unless it is on the pad.

Put the pad on your desk in a comfortable position. Orient it as you would a sheet of ordinary paper, with the long direction going toward and away from you.

CABLE SETUP

The connector on the mouse should be inserted into the matching socket on the right side of the monitor. There is only one appropriate socket there, so you can't go wrong.

CALIBRATING THE MOUSE

Each time the mouse is powered up, it must be CALIBRATED. (The mouse receives its power from the monitor, so the mouse loses power whenever the monitor is turned off, as well as when it is disconnected from the monitor.) This means that the mouse must adjust to the sensitivity of its particular sensors and to the mouse pad in use.

To do this, move the mouse around in large circles (about 6 inches in diameter) at a comfortable speed until the mouse cursor starts moving. This should happen within 10 seconds. If it does not, see the next section.

DIAGNOSING PROBLEMS

If you have a problem with the mouse, first make sure the Lambda software is running. The next thing to try is power-cycling the mouse. Pull the connector out of the

monitor, wait a few seconds, then plug it back in. Then recalibrate the mouse. This will usually clear up any problems.

If trouble persists, you can get some information about the problem by turning the mouse over and looking at the LEDs which shine through holes there. If the mouse is operating normally, only one of the two LEDs will be on, glowing continuously, and pressing the buttons will have no effect on the LEDs. If pushing any of the buttons causes the second LED to light, the calibration failed for some reason (probably because you did not move the mouse smoothly during calibration); power-cycle the mouse and try it again. If no LEDs are lit, try pushing any of the buttons on the front. If both LEDs light, the mouse is not calibrated; try the calibration procedure again. If no LEDs are lit, the mouse probably is not receiving power; make sure the connector is plugged in firmly. If one or more LEDs are flashing, there is some sort of error which was detected by the mouse's internal processor. The Mouse Systems technical manual describes how you can correct most of these. If this does not succeed, contact LMI for a replacement.

If the mouse does not slide easily on the mouse pad, the felt pads on the bottom of the mouse may be dirty or worn. The replacement pads packed with the mouse may be used if needed. The pads have an adhesive on the back which is protected by paper. The replacement lenses are supplied in case you lose one of the original ones while you have the mouse open; they do have an annoying tendency to roll away.

SOME TECHNICAL INFORMATION

Two of the DIP switches inside the mouse are changed from their standard settings. Switch 4 is turned OFF; this sets the pad orientation to suit the shape of our screen. Switch 6 is turned ON; this sets the bit polarity correctly.

Some of the mice supplied by LMI are wired with DB-9P connectors, rather than the RJ-11C connectors shown in the Mouse Systems manual. The pinouts of this connector are shown below:

PIN	FUNCTION	WIRE COLOR
2	Data Out	Red
7	Ground	Black
9	Power (+5)	Brown

Mouse Systems Corporation
M-2 Optical Mouse
Technical Reference Manual

Distributed by LMI 6033 W. Century Blvd. Los Angeles CA 90045
USA

© January 1984 Mouse Systems Corporation

Federal Communications Commission Radio Frequency Interference Statement

Warning: This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures: Reorient the receiving antenna; Relocate the computer with respect to the receiver; Move the computer away from the receiver; Plug the computer into a different outlet so that computer and receiver are on different branch circuits; If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful: "How to Identify and Resolve Radio-TV Interference Problems." This booklet is available from the U.S. Government Printing Office, Washington, DC 20402, Stock No. 004-000-00345-4.

Warranty

Your mouse is warranted against defects in material and workmanship for a period of one year from the date of purchase. The obligation of this warranty shall be limited to repairing or replacing any part of the product which, in the opinion of MSC, shall be proved defective in materials or workmanship under normal use and service during the one-year period commencing with the date of purchase. Contact the factory for a return authorization number. Return postage pre-paid to Mouse Systems Corporation, 2336H Walsh Avenue, Santa Clara, CA 95051.

This one-year warranty is in lieu of all other express warranties, obligations or liabilities. Any implied warranties, obligations or liabilities, including but not limited to any implied warranty of merchantability, shall be limited in duration to the one-year duration of this written limited warranty. Any action for breach of any warranty hereunder, including but not limited to, any implied warranty of merchantability, must be brought within a period of 18 months from the date of purchase. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation may not apply to you. No agent, representative, dealer or employee of MSC has the authority to increase or alter the obligations of this warranty. This warranty shall not apply to any product which, in the opinion of MSC, has been modified, repaired, or altered in any way without the express written consent of MSC. This warranty shall not apply to the felt feet which are expendable. In no case shall MSC be liable for any consequential damages for breach of this or any other warranty expressed or implied whatsoever. Some states do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

1. Introduction

This document explains all about the MSC M-2 optical mouse. MSC has been shipping the M-2 unit since December 1983 and this document is constantly updated with feedback from our customers. If you have any questions, comments, or problems, please feel free to contact us.

The M-2 mouse incorporates a number of features not found on any other mouse. The M-2 requires no calibration and can run on a variety of surfaces (versions of the M-2 will work on blue jeans and newspaper, for example). The M-2 uses analog hysteresis to provide noise-free operation with no loss of resolution. The tracking algorithm never permits the mouse to "fly away" and tracking error is minimal up to the the maximum speed (25 or 50 inches per second depending on the version).

Based on field experience at thousands of installations, the M-2 mouse has proven to be an extremely reliable computer peripheral. The contents of this manual should answer all of your questions in the very remote possibility that something goes wrong.

2. Normal use

The mouse is a fairly simple device. Basically all you do is plug it into your system and go.

3. When not in use

We recommend unplugging the mouse when not in use.

4. Connecting the mouse to your system

Connecting the mouse is straightforward. It will either plug into an existing mouse port in your computer, or into the RS-232 interface box. If you are using the RS-232 interface box, be sure the power supply is plugged into the wall and the mouse is plugged into the interface box. Plug the provided 25-pin connector into your computer and the other end into the interface box. The two RJ11C connectors on the interface box are interchangeable.

Orient the pad so that the blue lines are vertical.

5. Mouse use

The mouse works best when it is within $\pm 45^\circ$ from its nominal (vertical with respect to the selected pad) orientation. Most mouse users place their wrist on the surface and grasp the mouse with their thumb and little finger. Because the mouse can be lifted and re-positioned, the screen cursor can (and often is) *always* be positioned anywhere on the screen without having to move any muscles above the wrist.

One finger may be used for all three buttons, or the mouse buttons may be used one finger per button.

There are about as many different styles of mouse usage and preferences for mouse shape as there are people. The MSC M-2 design attempts to satisfy the broadest range of people.

6. Options

Various options are set at the factory. Most options can be modified in the field in a few minutes without any special parts. The default settings are:

<u>Option</u>	<u>Default</u>	<u>Optional</u>
Baud rate	1200	2400 baud
Polarity	RS-232	TTL
Protocol	MSC standard	Quadrature and others <contact factory>
RS-232 plug	Send on 3 (pin or socket)	<contact factory>
Pad orientation	Landscape	Portrait

For serial protocols, polarity refers to the polarity of the start bit. In the TTL mode, a START bit is a "low" TTL voltage. In the RS-232 mode, a STOP bit is a "low" TTL voltage. In RS-232 mode, the output is open-collector. RS-232 mode allows the mouse to be connected to our RS-232 interface box to be converted to RS-232 compatible levels.

If an RS-232 interface is ordered, we will normally ship a female connector where the mouse transmits on pin 3 (IBM Personal Computer) or a male, pin 3 connector (every other system).

The pad orientation selection determines how the pad is die cut; all mice are the same.

7. Opening up the mouse

The mouse never needs to be opened.

For the curious, to open the mouse, remove the two screws on the bottom of the mouse. Then place the mouse on a table as if you were going to use it. Then remove the top of the mouse.

There are only three things inside the mouse; 2 lenses and a PC board. Please note the placement of these components when you open the mouse. Notice that the lenses fit in the round holes on the bottom of mouse (you wouldn't believe how many mice we get back with the lenses placed on top of the PC board!). DO NOT touch the mirror; it is easily scratched.

When assembling the mouse, do not overtighten the screws.

8. Power-cycling the mouse

It should not be necessary to ever power cycle the mouse. The mouse code constantly checks itself for consistency and will restart itself in the event of a failure.

If it is desired to power-cycle the mouse, wait at least 5 seconds before plugging the mouse back in. This allows sufficient time for the microprocessor reset capacitor to discharge. Pre-mature re-plugging will almost always work and will not harm any of the mouse components.

9. Calibration

The M-2 mouse requires no calibration at any time. It instantaneously adjusts to any surface it is placed upon.

10. What to do if the mouse doesn't work right

There are no user serviceable components inside the mouse. If the mouse does not work or seems to not operate as well as you think it should, first check the following before calling us:

When the mouse is powered on, the red LEDs on the bottom of the mouse will turn on. If this doesn't happen, it probably means the mouse isn't plugged in or there is a bad cord somewhere. Try wiggling the mouse and power supply cords around especially near a connector.

If one of the switches doesn't work, it usually means the case has been seriously overtightened and one of the keytops is actuating one of the switches. Another cause could be a bad switch.

The mouse should not mis-track in normal use. At high speeds and when the mouse is lifted from the surface while moving at high speeds, occasional mis-tracking may occur. If the mouse is exposed to high RF fields or is operated without a proper ground on the host computer, there may be some jitter when the mouse is lifted from the mouse pad.

Erratic tracking of the mouse under normal conditions could be due to improper pad orientation or a bad mouse pad. Although erratic tracking could also be caused by a catastrophic failure of the redundant LEDs, a cracked lens, or an extraordinarily large piece of dust in the optical path, we have never seen any of these cases.

The problem could also be in your computer tracking software. See Section 15.

If you are using the RS-232 interface, you may have ordered the wrong cable. Try reversing pins 2 and 3 (or purchase a "null modem") if you suspect this is the case.

11. Care of the mouse pad

The mouse pad is made out of aluminum and coated with a very hard organic coating. However, the pad is susceptible to scratches and dents. You can wash it and use window glass cleaner on it.

If you have a mouse pad which appears to have only blue lines on it, it should not be exposed to direct sunlight (i.e., taken outdoors). Please note that it is perfectly safe to use these pads indoors in direct sunlight since window glass absorbs UV.

12. Maintenance

The mouse is completely maintenance-free.

13. Unusual applications

The mouse can be used as a digitizer if a transparency of the item to be digitized is placed over the CRT screen.

14. For more information

The mouse is covered by and described in U.S. Patent 4,364,035.

OEM interfacing

15. OEM Software interfacing

For a number of different computers, MSC provides a variety of software support packages. For example on the IBM and TI Personal Computers, MSC provides a software package to add end-user customizable pop-up menus to existing software with no software or hardware modifications.

For other machines, we will provide code samples and suggested high level interfaces on request. If you would like to market your mouse software through MSC, we would be delighted to talk with you.

The MSC mouse protocol is the most bandwidth efficient protocol available today providing more information in a shorter amount of time at any given baud rate than any other protocol.

The protocol is as follows: five byte data blocks are sent when and only when there is a change of mouse state. The start of a data block is indicated by a sync word whose upper 5 bits are 10000. The next 3 bits are the debounced state of the switches (0 means depressed). The next 4 bytes contain two updates of the x and y registers: $\Delta x, \Delta y, \Delta x, \Delta y$ (note that Δx is the horizontal distance moved since the last transmission of Δx relative to the coordinate system of the pad). Each byte is a two's complement packed 8-bit binary number. Positive is RIGHTwards and UPwards motion. After five bytes are sent, other bytes may be sent before a sync word appears. Therefore, most tracking software should scan for a sync word after receiving the 5 bytes.

The following table summaries the protocol:

<u>Byte</u>	<u>Contents</u>
1	1 0 0 0 0 <i>L M R</i>
2	Δx
3	Δy
4	Δx
5	Δy

The mouse only sends out data when it thinks it has changed state (a switch transition or movement).

Mouse tracking software should read bytes until a byte with a 10000 high-order bit pattern is read. The switches are the lower 3 bits of this byte (0 means pressed). Next, read two sets of two bytes. After a total of 5 bytes have been read, the software should ignore bytes until the "start" byte is recognized. This ensures synchronization.

Since extraneous bytes are not sent in the standard M-2 mouse, testing for extraneous bytes is a good way to debug your tracking software (this technique may not be reliable with versions of the M-2 which contain serial numbers since these mice may transmit a serial number as the sixth byte in some data blocks).

After initializing the UART to 8 data bits, no parity, 1200 baud operation, the basic algorithm is as follows:

1. Read a byte
2. If upper bits are 10000, then save this in SWITCH-STATUS. Otherwise, go to 1 (extraneous byte received). XOR'ing the new SWITCH-STATUS and old SWITCH-STATUS will indicate whether a button has changed state.
3. Add the next byte to the X-REGISTER (be sure to do an 8086 CBW instruction first if you are on a 16-bit machine).

4. Add the next byte to the Y-REGISTER.
5. Add the next byte to the X-REGISTER.
6. Add the next byte to the Y-REGISTER.
7. Go To 1.

Note that each data byte should be read using interrupts. Do NOT read the first data byte and then loop inside the interrupt service routine until the other 4 bytes have been read. Instead, read a byte in the interrupt service routine, process it as appropriate, and set a counter to indicate which byte has been read. Then return from the interrupt.

Now, asynchronously with this operation, the computer may LOAD-AND-ZERO these registers. Depending on the program controlling the mouse cursor, these values may or may not be added to the mouse cursor position register (e.g., some software may restrict the mouse cursor to certain areas of the screen at certain times).

There has been some confusion about the protocol: switches, $\Delta x, \Delta y, \Delta x, \Delta y$. The second and third bytes are not the same as the fourth and fifth bytes. They are the same register which is sent and cleared *twice* in the five byte packet. This is because it is a more efficient use of the available bandwidth to send the position information more often than the switch status. The mouse NEVER sends high and low bytes of a sixteen bit word. In other words, treat the second and third bytes just like the fourth and fifth bytes.

For example, say the mouse is at screen position (10,10). The mouse sends the block (hex): 87 01 FF 03 00. This is interpreted as follows: 1) The switches are all up 2) Move the mouse cursor 1 unit to the right and 1 unit down 3) Move the mouse cursor 3 units right. Of course, we don't necessarily redraw the cursor on the screen each time. Rather, we update internal mouse position registers and let our screen refresh routine sample these position registers at the screen refresh rate. Also note that we have simplified things somewhat; what really happens is that the tracking routine updates a "delta position register set." Then it is up to the application software to decide whether and how to update the mouse position registers which correspond to the physical mouse position on the screen. For example, the application software may decide to "trap" the mouse inside a window.

Some systems may update the mouse cursor after reading the first two-byte set. Other systems may wait longer depending on how long it takes to move the mouse cursor on the screen. If you update x and y screen registers at the same time, the screen tracking may look smoother than if you don't.

DO NOT ignore bytes 4 and 5. Those bytes really are significant and you could cause serious mistracking if you ignore them. You will also get (on the average) half the resolution meaning you have to move the mouse twice as far. Remember, you don't have to redraw the cursor every time the mouse position is updated.

Be sure not to let the mouse cursor go off-screen.

Many systems use a 2:1 screen-to-mouse ratio. That is, 1 inch of mouse motion gives 2 inches of screen motion. This usually requires that the mouse position information be multiplied by 2 before being displayed. Hence, every other dot on the screen is addressable. This has never turned out to be a problem; single "pixels" should normally be addressed with the screen "magnified" so that a single pixel is magnified into a square array of pixels. This is because human pointing has a limited dynamic range; it is inefficient to require someone to point with better than 10 mil accuracy.

We strongly urge you to try 2X magnification. Most software engineers are reluctant to do so, but after

they try it, they find the feeling of control and speed far outweighs the inability to choose single pixels. Also be careful on magnification since some screens are not square. The general rule is that if you draw a circle with the mouse, it should look like a circle on the screen.

If you really need single pixel resolution, you might consider using a tracking algorithm such that mouse motion of 0 and ± 1 gives screen motion of 0 and ± 1 and mouse motion of any other value is doubled.

Some people use a non-linear scaling approach based on the velocity of the mouse. That is, if a mouse delta is below a certain absolute value, the delta is scaled with one factor and a different factor is used otherwise. The supposed "advantage" of this technique is that if you move the mouse slowly, you can move in very fine increments, whereas if you move the mouse fast, you can quickly get to the other end of the screen with very little mouse motion. In reality, this technique results in a loss of both speed and accuracy. Human beings are inherently linear and one of the reasons mice are so popular is that using a mouse is very similar to pointing with your arm. With the non-linear scaling approach, you force the user to solve a non-linear control problem. The non-linear technique has been suggested by hundreds of people over the last 10 years, but very few systems use the technique. The reason is simple: the (linear) mouse has been shown to be a near perfect pointing device. If there is a better pointing device, it cannot be perceptibly better.

When using the RS-232C interface, the mouse does not set DSR, CTS, etc. These signals can usually be ignored by software. In some cases, user software must contain code which specifically tells the UART to ignore these signals. In the first 40 msec after the mouse is plugged in, it will send a start bit continuously (BREAK). You may need to reset the UART after the receipt of the BREAK. Also, sometimes initializing the UART is a very tricky procedure; we know of several cases where, because of improperly initializing the UART, you sometimes must run a program twice before the mouse will work.

The mouse sends 8 bit bytes with no parity. All eight bits contain significant data. Operating system features that change a bit (e.g., always clear high order bit), swallow nulls, respond to $\uparrow S$ or $\uparrow Q$, and/or replicate DELETE should be disabled.

16-bit machines should remember to propagate the sign bit (all quantities are two's complement). Use the 8086 instruction CBW (Convert Byte to Word) to do this.

Some users have asked for a poll mode where the mouse can be queried for its position. This is unnecessary overhead for both the mouse and the host system: the host must poll the mouse every refresh to ensure that the mouse is always accurately tracked. Hence, not only is polling redundant, but it is extremely inefficient since most of the time the mouse hasn't moved.

Since 5 bytes are transmitted at 1200 baud, the maximum sustained velocity of the mouse should be under 61 in/sec for real-time tracking. The mouse position is available for refresh 48 times/sec. Velocities seldom exceed 30 in/sec and 24 frames/sec is the frame rate for motion pictures. The mouse can transmit at higher baud rates but the performance advantage is negligible compared with the added burden on the host computer.

If the software interface is correct, the mouse will track reliably and accurately under normal conditions. It will never drift on the screen. No time-delay will be apparent to the user, even at high speeds.

For examples of innovative mouse software, we refer you to Apple's Lisa, Xerox's Star, VisiCorp's VisiOn, or any of the products from Lisp Machines, Inc. or Symbolics (both in Cambridge, MA).


16. OEM Hardware interfacing

If you go direct into a UART you should probably order a TTI mouse. Another way is to use the standard RS-232 polarity and provide a 4.7K pull-up resistor. In this case, a start bit will be a logic high. The mouse requires 5V @ 200mA. The RJ11C connector pinout is GND (1), +5 (2), OUTPUT to UART(3). To determine which is pin 1, unplug the line cord (not the handset cord) from your telephone. Grasp the connector with the "pins" facing you and the cord hanging straight down (or towards you). Then pin 1 is on your left. Note that the connector is a six position connector: the middle four pins are loaded (1-4) and the outer positions (0 and 5) are not.

The levels sent on the RS-232 interface are RS-232 compatible, although they do not meet RS-232 specifications. Specifically, the upper level is +5 V with some ripple and the lower voltage is -5 V (regulated). We used this scheme for increased reliability and low cost: the RS-232 level converter is nothing more than a 1K pull-up resistor.

17. Specifications

- Requires 5V @ 200mA.
- RJ11C connector pinout: GND (1), +5 (2), OUTPUT to UART(3)
- Mouse OUTPUT line requires a 4.7K pull-up resistor at UART input (RS-232 polarity)
- Size: 98 x 66 x 28 (L x W x H in mm)
- 0.25 mm resolution (100 counts/inch)
- Maximum velocity: 25 or 50 inches/sec
- Depth of field >1 mm



**How to Use the
LMI Printer Software
May 1984**

**Published by LMI 6033 W. Century Blvd. Los Angeles CA 90045
USA**

How to use the LMI printer software

This file documents version 6 of the LMI printer software (otherwise known as the TIGER system, named after the IDS Paper Tiger printer, which was the first printer used at LMI, but which is not presently supported). It describes how to request a printout, how to use the special features of the program, and how you must set up your system to get it to work correctly.

The hardcopy of this document that you receive was produced with LMI printer software. At various times, we have used both the Toshiba P1350 and TI 855 printers for the hardcopy; The Toshiba version is produced using bit-graphic fonts; the font list is '(25vg 25vgi 25vgb 25vgbi 25fg 25vrb). The TI855 version uses the font modules *Gothic*, *Courier Italic*, and *Courier* in slots 1, 2, and 3 respectively; the font list is '(3. 2. 35. 34. 1. 17.). This setup uses serifed fonts rather than sans-serif fonts for the main text; this choice is based on the font modules we own.

Obtaining and connecting a printer

Just about any kind of serial ASCII printer will work to some extent with this software. At present, we support the following printer types: **VANILLA**, which is used for printers with no special features (or printers whose special features are not yet supported), **TOSHIBA**, which supports the Toshiba P1350 and P1351 printers, and **TI855**, which supports the Texas Instruments 855 printer. The distribution package also includes software for the **ANADEx** type, which handles the Anadex DP9000 series printers; however, this is not supported by LMI, as we have no ability to test it.

The **TOSHIBA** printer type supports many special features, including multiple-font printing (both with the built-in fonts and with arbitrary fonts which are printed in graphics mode), bit-array printing, setting character and line-per-inch density, and printing Lisp Machine characters using graphics mode.

The **TI855** printer type supports multiple-font printing (using the fonts available in font modules), printing of bit arrays, and setting character and line density. It does not support bit-mode fonts; because of the interchangeable font modules, these are not needed as they are on the Toshiba.

The section "*Printer interfacing*" gives some information on how to connect a printer. Contact LMI Customer Service for more advice on selecting and connecting a printer.

Setting up your site files

To cause printouts to be sent to the correct machine, you have to edit your site file (SYS: SITE; SITE LISP) to make your printer the default printer, and then do (MAKE-SYSTEM 'SITE ':COMPILE) to cause the site information in your system to be updated.

In the site file you will find entries for the variables :DEFAULT-PRINTER and :DEFAULT-BIT-ARRAY-PRINTER. The present value may be NIL, ':TIGER, or ':TOSHIBA, or it may be a list. The new value should be of the form '(:PRINTER-TYPE "HOST-NAME"), where PRINTER-TYPE is the appropriate supported printer type for your printer, and HOST-NAME is the name of the

machine that will have a printer connected. These site options cause the values of the variables `SI:*DEFAULT-PRINTER*` and `SI:*DEFAULT-BIT-ARRAY-PRINTER*`, respectively, to be set; you can change the defaults during a session by setting one or both of these variables.

If your printer supports bit-array printing, you should change both `:DEFAULT-PRINTER` and `:DEFAULT-BIT-ARRAY-PRINTER` to this value. If your printer does not support bit-array printing, `:DEFAULT-BIT-ARRAY-PRINTER` should be `NIL`.

If you have more than one printer on your network, you might want the default printer on some machines to be different from the default on others. You can put a local value for `:DEFAULT-PRINTER`, `:DEFAULT-BIT-ARRAY-PRINTER`, or both into the `lmlocs` file (`SYS: SITE; LMLOCS LISP`). You can also set `:DEFAULT-PRINTER` and `:DEFAULT-BIT-ARRAY-PRINTER` differently; this is useful to distribute the printing load a bit, or if only one of your printers can print graphics. See the *Lisp Machine Manual* for more information.

The program expects to find the `TIGER-QUEUE` directory on the host that contains the printer. The `TIGER-QUEUE` directory is used whenever temporary files need to be created (this includes printing from streams (such as `M-X Print Buffer` and `M-X Print Region` in the editor), printing arrays on machines other than the one with the printer, printing more than one copy of an array, and using the `:ARRAY-FILE` printer type). This means that the Lisp Machine with the printer must have a local file system (all system bands supplied by LMI have one), that the `TIGER-QUEUE` directory must exist, and that the machine must be listed in the `:CHAOS-FILE-SERVER-HOSTS` entry in the site file (again, this should always be true in LMI-supplied site files).

There must be directory translations for `TIGER`; and `TIGER-FONTS`; on your `SYS: host`. The sample site file on this tape (`SYS: TIGER; SITE LISP`) includes an example of the translations you will need. If this software was supplied with your machine, rather than being sent later, these will already be in place.

Finally, your site directory should contain the file `SYS: SITE; TIGER SYSTEM` that is included with this distribution. This file is referenced when you build the `TIGER` system; it tells where to find the system definition. (The actual system definition is in the file `SYS: TIGER; TIGER-SYSTEM LISP`).

How to load the printer software

First, you must restore all the files on the supplied tape. If your file host is a Lisp Machine, (`RESTORE-MAGTAPE`) will do. Remember to use (`FS:TAPEMASTER-INITIALIZE`) first if you are using a Lambda.

Then you will need to load the `TIGER` software into your machine. (`MAKE-SYSTEM 'TIGER`) will do it. Answer `Y` to all the questions it asks. You may want to save a band containing it. Some bands supplied by LMI already contain this system; if you have one of these, you should make sure all the patches are loaded by doing (`LOAD-PATCHES 'TIGER`).

If you want to request printouts remotely (that is, from Lisp Machines other than the one that has the printer), both machines must have this software loaded, and both should have all patches loaded.

The supplied programs will work on both `CADR` and `Lambda` processors. (If you load the supplied files on a `CADR`, a number of warnings of the form "File `FOO`

was compiled with a different version of SELECTQ"; these can be safely ignored.) Hardcopy can be requested on one type of machine and printed on the other.

How to request printouts

To request a printout, you use the functions `HARDCOPY-FILE`, `HARDCOPY-STREAM`, and `HARDCOPY-BIT-ARRAY`. These functions are all defined in the green *Lisp Machine Manual*. They take optional keyword arguments, that are described later. The `HARDCOPY-BIT-ARRAY` function (by default) copies the array into a temporary array before printing it, so you are free to modify or reuse the original array as soon as it returns.

If you are printing arrays, you might find the function `TIGER:TIGER-ARRAY` convenient. Instead of taking required `LEFT`, `TOP`, `RIGHT`, and `BOTTOM` arguments, it takes optional keyword arguments for those numbers. (Any bound that is not specified defaults to the appropriate edge of the array.) This function will also accept a window as its argument; the screen array of the window is printed, and the bounds are defaulted from the inside edges of the window. This function does **NOT** copy the array by default, but you can ask for copying with the `:COPY-ARRAY` keyword argument.

You can also request a printout using the ZMac commands `M-X Print File`, `M-X Print Buffer`, and `M-Sh-P` (Quick Print Buffer). (In System 98 the additional command `M-X Print Region` is available.)

If you have a printer that supports bit-array printing, **Terminal Q** will hardcopy the screen or a portion of it. If you have NewDraw, the `Hardcopy` and `Hardcopy Hierarchy` commands will print your drawings.

There is also a new `Hardcopy` choice in the system menu. If you select this, you get a choose-variable-values menu that lets you select a file to print, and some optional parameters. At present, you will have to edit the sources if you want to change the list of possible printers it presents; this will be changed in the next version.

Optional parameters for printing

- `:COPIES (1)` ALL
The number of copies of the file to print.
- `:DELETE-AFTER (NIL - files, T - arrays and streams)` ALL
If T, the file is deleted after it is printed.
- `:TITLE-PAGE (NIL)` ALL
If T, a page telling who the printout is for is printed before the file is printed.
- `:FILE-HEADING (NIL)` ALL
If T, a heading telling who the printout is for is printed at the top of the first page of the printout.
- `:PAGE-HEADINGS (T - files and streams, NIL - arrays)` ALL
If T, a heading with the page number and filename is printed at the top of every page.

Optional parameters for printing files and streams

- :CHARACTERS-PER-INCH (printer-dependent) TOSHIBA, TI855
 The number of characters per inch to print. This will be rounded to the nearest value that your printer supports.
- :LINES-PER-INCH (printer-dependent) TOSHIBA, TI855
 The number of lines per inch to print. This will be rounded to the nearest value that your printer supports.
- :FONT (0) TOSHIBA, TI855
 The font to print the file in.

On the Toshiba P1350, this can be a fixnum, that refers to a font built into the printer, or it can be a symbol* or string, in which case it refers to a font that will be printed in graphics mode. The user should be aware that graphics-mode printing is slower than character-mode printing. The Toshiba P1350 printer has three built-in fonts. Font 0 is the high-speed font, font 1 is Prestige Elite, and font 2 is Courier. We have a number of bit-graphic fonts for the Toshiba available; they are in the directory SYS:TIGER-FONTS;. The software supports any mixture of these; you can use built-in fonts and bit-graphic fonts in the same document. This is useful, for example, to include some true italics in your printout, while not slowing down printing too much.) The down-loadable font feature of the P1351 is not presently supported.

On the TI855, only fixnum values are allowed. Font 0 is the internal limited character set; note that this works only in draft mode. Fonts 1, 2, and 3 are the fonts contained in the font modules plugged into slots 1, 2, and 3. Font 4 is the font loaded into the Extended Character Set option. (The software does not presently support this option. We will be adding support for it when ours is delivered.) The default font list cyclically uses fonts 1, 2, and 3; fonts 0 and 4 are not used. If you own fewer than 3 font modules, you may want to change the value of the variable TIGER:DEFAULT-TI855-FONT-MAP.

You can also select boldface and shadow printing. Specifying a font number plus 16. in the :FONT or :FONT-LIST argument gets you a boldface version of it. A font number plus 32. get a shadow-printed version. A font number plus 48. gets a boldface shadow-printed version. (I.e.: If font 1 is Gothic, font 17 is Gothic Bold, font 33 is Gothic Shadow, and font 49 is Gothic Bold Shadow.) Boldface printing is simply overstruck. Shadow printing is overstruck with the new set of dots offset 1/120 of an inch; this produces a somewhat different bold effect. The effect of boldfacing does not come out well in copies; use shadow printing in documents that will be reproduced. Which you prefer in originals is a matter of taste.

A warning about selecting fonts on the TI855: Many of the font modules have a modified character set for word processing. The character set in the module is indicated just below the font name. ASCII 96 is a standard ASCII font; WP is a word-processing font. (There are also some foreign-language modules available; these have modified character sets as well.) **Only ASCII fonts should be used for program listings!** (Fonts we know about: Prestige Elite and Gothic (the font module packed with the printer) are ASCII; Orator 85, Courier, Courier Italic, Gothic 15, and Modern PS are WP.) A combination that we find works well for program listings which use Electric Font Lock Mode in the editor is Gothic as font 0 and Courier Italic as font 1. The internal

character set is a standard ASCII character set. It can be used for listings. Note that it only implements draft mode.

The changed characters in the WP fonts are as follows:

ASCII	WP
\	registered trademark
^	copyright
'	degrees
{	
	paragraph
}	dagger
~	trademark

:FONT-LIST (NIL)

TOSHIBA, TI855

A list of fonts to print the file in. Each element of the list can be a valid font specification, as explained above. The first font will be used for text that is in the first font in the attributes list of the file, and so on.

The default for font handling (if neither **:FONT** or **:FONT-LIST** is specified) is to use a default font list containing only hardware fonts (the Toshiba bit-graphics fonts are never used by default). If **:FONT** is specified, that font will be used for the entire document; any font changes contained in the document will not be processed.

:QUALITY (NIL)

TI855

This allows you to select draft or quality printing mode. T means quality; NIL means draft.

Optional parameters for printing arrays

:COPY-ARRAY (HARDCOPY-BIT-ARRAY - T, TIGER-ARRAY - NIL)

TOSHIBA, TI855

T means to copy the array into a temporary array before printing. Note that the default value is different depending on which function you called; these defaults are for compatibility with previous software versions.

:LEFT (0)

TOSHIBA, TI855

The leftmost position of the array to use.

:TOP (0)

TOSHIBA, TI855

The highest position of the array to use.

:RIGHT ((pixel-array-width array))

TOSHIBA, TI855

The rightmost position of the array to use.

:BOTTOM ((pixel-array-height array))

TOSHIBA, TI855

The lowest position of the array to use.

:X-SCALE (printer-dependent)

TOSHIBA, TI855

A factor to multiply all pixels by horizontally. Only small fixnums are permissible values; the exact set is printer-dependent. (The Toshiba and TI855 support any value of this.)

:Y-SCALE (printer-dependent)

TOSHIBA, TI855

A factor to multiply all pixels by vertically. Only small fixnums are permissible values; the exact set is printer-dependent. (The Toshiba supports scale factors 1, 2, 3, and 4; the TI855 supports scale factors 1,

2, and 4. Factors larger than 2 actually produce grotesquely large printouts on these printers; they may be useful someday.)

Cleaning up problems

Occasionally something will go wrong with the printer software, and you will need to prod it a bit. The following functions may be useful. Note that these must be executed on the machine that has the printer connected.

(TIGER:ABORT-FILE)

This stops the printout of the current file and removes it from the queue. Any files that are later on the queue remain there. Since you cut off printing in the middle of a file with this function, you have to power-cycle the printer when you call this, as its state is not known. If the current printout is a temporary array created by the COPY-ARRAY option, the array is returned to the free pool.

(TIGER:UNWEDGE-TIGER)

This kills the TIGER process, deletes the printer streams, and then restarts the TIGER process. The queue is not reset; any file that was in progress when the UNWEDGE-TIGER was done will be restarted. You will have to power-cycle the printer.

(TIGER:RESET-TIGER)

This kills off everything associated with the printer software: the print queue is cleared, the printer streams are deleted, and the TIGER process is killed. Use this when something seems very wrong and you feel you need to start all over. Again, you will have to power-cycle the printer. If any of the queue entries were for arrays created by the COPY-ARRAY option, they are returned to the free pool.

(SEND TIGER:SERIAL-STREAM ':RESET)

If the serial stream seems to get stuck (stays in Serial TYO state forever when the printer is actually ready, for example), resetting the stream sometimes helps. If this fails, you will have to use TIGER:RESET-TIGER.

Extending the software (implementing new printer types)

To find out how to add new printer types, you will have to read the source code. Mostly, you have to create a new flavor of stream for your printer, and give it methods for whatever special features you want to support. Most of these will be self-contained, and therefore simple. The symbol associated with the new printer type must also be given certain properties; see an existing definition for examples.

The TOSHIBA printer is an example of a very complete printer definition. This printer definition is considerably more complex than most are likely to have to be — the interdependence of a lot of the functions is caused by the desire to make the shift between bit-graphics mode and regular printing mode as clean and painless as possible. On the Toshiba, each new bit-graphics command causes the printhead to stop; also, sending regular characters after bit graphics causes the printhead to stop. If the new stuff is too close to the old, the head actually has to back up so it can get into position. Much of the hair is caused by the wish to give all contiguous graphics as one graphics mode command to minimize these problems.

Printer interfacing

The supported port on the LMI Lambda is the B (local) port (the one the SDU console isn't connected to). This port has the following pinouts:

- 1 Protective Ground
- 2 Transmit Data (Lisp Machine receives)
- 3 Receive Data (Lisp Machine sends)
- 4 Ready to Send (Lisp Machine receives)
- 5 Clear to Send (Lisp Machine sends -- always on)
- 6 Data Set Ready (Lisp Machine connects to ground)
- 7 Signal Ground
- 8 Data Carrier Detect (Lisp Machine sends -- always on)

If hardware handshaking is being used, whichever line the other end asserts for Ready/Busy signalling should be connected to pin 4; otherwise, pin 4 should be jumpered to pin 5. Pins 1 and 7 should go straight through; pins 2 and 3 go straight through or reversed, depending on which line the other device asserts. (Most printers will want 2 connected to 2 and 3 connected to 3.)

The CADR serial port is connected to J10 on the IOB. Some LMI-manufactured CADR's have a serial port connector on the bulkhead near the video connector. If you have no such connector, a standard adapter cable that attaches to J10 is available from LMI. (This cable is identical to the cable used to connect to the Chaosnet transceiver.) In either case, the pinouts of the DB25 connector are as follows:

- 1 Protective Ground
- 2 Transmit Data (Lisp Machine sends)
- 3 Receive Data (Lisp Machine receives)
- 4 Ready to Send (Not connected)
- 5 Clear to Send (Lisp Machine receives -- connected to DSR on chip)
- 6 Data Set Ready (Lisp Machine receives -- connected to CTS on chip)
- 7 Signal Ground
- 8 Data Carrier Detect (Lisp Machine receives)
- 20 Data Terminal Ready (Lisp Machine sends -- always on)

Connecting line 6 to whichever line the other device asserts for Ready/Busy should enable hardware handshaking on the CADR. Pins 5, 6, and 8 must be high for output to occur; you should usually jumper 5, 8, and 20 together. Connect pin 6 to these as well if you are not using hardware handshaking. Note that the CADR will usually need a cable that reverses lines 2 and 3 for connection to a printer.

The variable `TIGER:HANDSHAKE-TYPE` controls which type of handshaking the software will attempt to use. It can take three values: `:HARDWARE`, `:SOFTWARE`, and `:DEFAULT` (this uses `HARDWARE` on Lambdas, `SOFTWARE` on CADR's). This is tested only when a new serial stream is made; usually, only one stream is made per warm-boot, but calling `TIGER:ABORT-FILE` or `TIGER:RESET-TIGER` causes a new stream to be created the next time something is printed. Therefore, you should set this variable before you print anything. (You can also patch it or change it in the sources.)

If you have a Toshiba P1350, you will need to wire a cable to connect the printer to your Lisp Machine as follows:

P1350 LMI Lambda CADR

1	1	1
2	2	3
3	3	2
7	7	7

Jumpers:

PI350: connect 5, 6, 8, 20

Lambda: connect 4, 5

CADR: connect 4, 5, 6, 8, 20

The handshake mode should be :SOFTWARE. (Hardware handshaking does not seem to work reliably with the Toshiba PI350 printer. It works fine with the TI855, so we believe this to be a bug in the printer.) Inside the printer, the switches should be set as follows:

S1 Bits 5 and 8 ON, others OFF
 S2 All OFF
 S3 Bit 7 ON, bit 8 OFF, others as preset
 S4 Bits 1, 2, 3, 4, 7, 8 ON, others OFF

If you have a Toshiba PI351, you will need to wire a cable to connect the printer to your Lisp Machine as follows:

PI350	Lambda	CADR
1	1	1
2	2	3
3	3	2
7	7	7
20	4	6

Jumpers:

PI350: connect 5, 6, 8

LMI Lambda: none

CADR: connect 4, 5, 8, 20

The handshake mode should be :HARDWARE. Oddly enough, the PI351 we have only functions correctly in *HARDWARE* handshaking mode -- just the opposite of the PI350! The switches inside the printer should be set as follows:

S1 Bits 5 and 8 ON, others OFF
 S2 All OFF
 S3 Bit 7 OFF, bit 8 ON, others as preset
 S4 Bits 1, 2, 3, 4, 7, 8 ON, others OFF

If you have a TI855 printer, you will need to make a special adapter cable. (Neither of the serial cables offered by TI is exactly correct.) This cable has a male Centronics printer connector at one end, and a male RS-232 connector (DP-25P) at the other end. It is wired as follows:

Centronics (TI855)	DB-25P (LMI Lambda)	(CADR)
15	4	6
16	3	2
17	1	1
19	7	7
33	5	20

Jumpers: None

The handshake mode for the TI855 should be :HARDWARE. (:SOFTWARE also works with TI855 printers; however, we believe that :HARDWARE is the mode of choice whenever feasible; it causes less load on the Lisp Machine processor, and there are no problems with dropped handshake characters.

Moniterm Corporation
Operating Manual
VR-Series

Distributed by LMI 6033 W. Century Blvd. Los Angeles CA 90045
USA

Copyright © 1984 LISP Machine Inc.

<u>A. GENERAL INFORMATION</u>	<u>Page #</u>
I. General	1
II. Power Input	2
III. Power Mating Connector	2
<u>IV.</u> Power Supply Circuit	2
<u>V.</u> Power Dissipation Chart	2
<u>VI.</u> TTL Interface Specifications	3
<u>VII.</u> Separate Sync Specification	4
<u>VIII.</u> ECL Interface Specifications	5
<u>IX.</u> ECL Separate Syncs	6
<u>X.</u> ECL Composite Syncs	6
<u>XI.</u> Two Level Composite Video	6
<u>XII.</u> ECL Board Assembly Drawing	7
<u>B. DISPLAY TIMING</u>	
<u>I.</u> Horizontal	8
<u>II.</u> Vertical	8
<u>III.</u> Two Level Composite Video Option	10
<u>IV.</u> Composite Video SYNC	11
<u>C. THEORY OF OPERATION</u>	
<u>I.</u> Horizontal Section	12
<u>II.</u> Vertical Section	14
<u>III.</u> TTL Video Board	15
<u>IV.</u> ECL Video Board	15
<u>V.</u> Circuit Wave forms	16
<u>D. Schematics, Assembly Drawings, Bill of Materials, Mechanical Drawings, Adjustments</u>	

The Moniterm VR series display monitor utilizes the latest advances in integrated circuits and switching technology teamed with a high performance CRT. Horizontal frequencies are available from 32 KHZ to 68 KHZ and retrace times as low as 2.8 u seconds.

A separate modular high voltage supply allows wide variations in displayed video without changing brightness levels or display blooming, allowing the display designer to use visual attributes such as; reverse video, blink, and reverse blinking video without ill effects. This high voltage supply also allows a wide range of horizontal retrace times. This is very helpful in applications where the display drive logic has bandwidth limitations.

Environmental

Temperature Range: Operating: 10C to 50C (50F to 122F)
Transit storage: -40C to 85C (-40F to 185F)

Humidity: 5% to 90% (non-condensing)

Altitude: Operating: up to 10,000ft (3.0 km)

Transit Altitude: up to 40,000ft. (12.2 km)

X-RADIATION

The monitors comply with DHEW standard 21-CFR-sub chapter J when the monitor is operated within the specified input voltage limits.

WEIGHTS

VR-15-21

VR-17-27

VR-19-33

FULL BODYSHIELD

VR-15 2.5 pounds

VR-17 4.0 pounds

VR-19 5.25 pounds

Low Voltage Power Supply: 6 pounds

Low Voltage Power Supply Shield: 1 pound

Geometric Distortion - sweep non-linearities and pin cushion distortion exceed the requirements of EIA STD RS-375A.

Internal Controls (See Adjustment Section)

Horizontal width

Horizontal Linearity

Vertical Hold

Vertical Top Bottom Linearity

Vertical D.C. Centering

Final Anode Voltage

Brightness

Horizontal Hold

Horizontal Dynamic Focus

Vertical Size

Vertical Linearity

Vertical Dynamic Focus

D.C. Focus

Video Contrast

Optional Controls

Remote Brightness: 100K 1/2watt potentiometer. With the remote brightness option the internal brightness control is a range control.

Remote Contrast: TTL Video 5K ohm 1 watt potentiometer
ECL Video 500 ohm 5 watt potentiometer

The monitor's power input connector is a Molex #22-27-2041 4 pin connector configured as follows:

Pin # 1	+48vDC
Pin # 2	GND
Pin # 3	GND
Pin # 4	+32vDC

*For Power requirements see the power dissipation chart

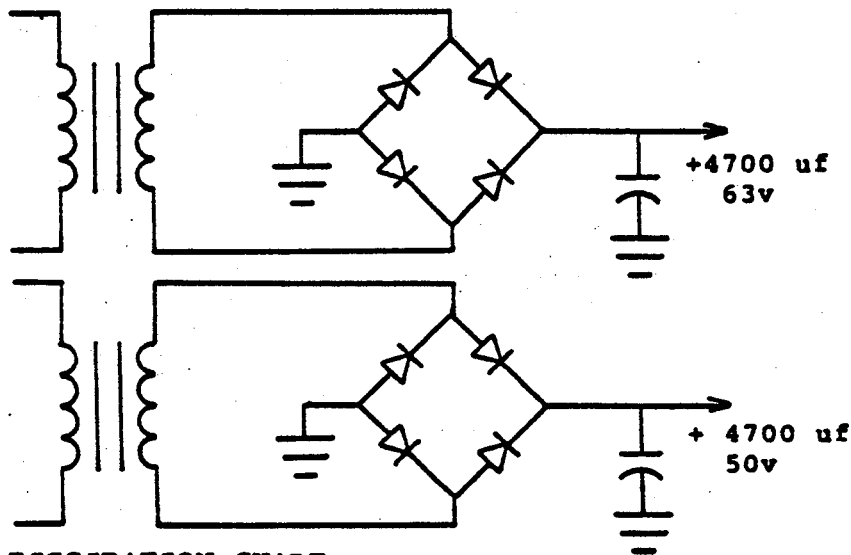
III. MATING CONNECTOR

The power input connector should be mated to Molex #22-01-2045

The Molex pin for this connector is #08-50-0136

IV. POWER SUPPLY CIRCUIT

Since the deflection board has on board regulators, the raw D.C. power circuit shown below is satisfactory.



V. POWER DISSIPATION CHART

Average D.C. Power	15P	15L	17P	17L	20P	20L
+48v \pm 10% (50 KHZ Horizontal)	875ma	1.0a	950ma	1.0a	950ma	1.1a
+32v \pm 10% (50 KHZ Horizontal)	650ma	550ma	750ma	600ma	800ma	650ma
+48v \pm 10% (64 KHZ Horizontal)	875ma	1.1a	950ma	1.1a	950ma	1.1a
+32v \pm 10% (64 KHZ Horizontal)	650ma	550ma	750ma	600ma	800ma	650ma

Moniterm supplied low voltage power supply

Input voltage 100v, 120v, 220v, 240v, RMS 50/60 HZ
programming card selectable

Input power 75w (nominal) See model chart

VI TTL INTERFACE SPECIFICATIONS
 (Connector Molex #09-75-1061)

Pin out
 Vertical Sync 1
 GND 2
 Horizontal Sync 3
 GND 4
 Video (1 Banks) 5
 GND 6

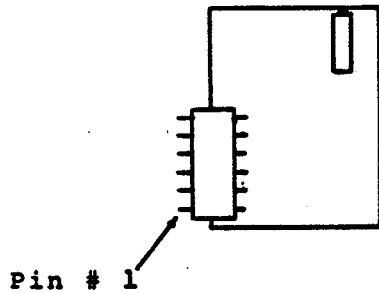
For Sync Specifications see separate Syncs

MATING CONNECTOR

Molex #09-50-3061

Molex Pin # 08-50-0106

Top of the TTL Board



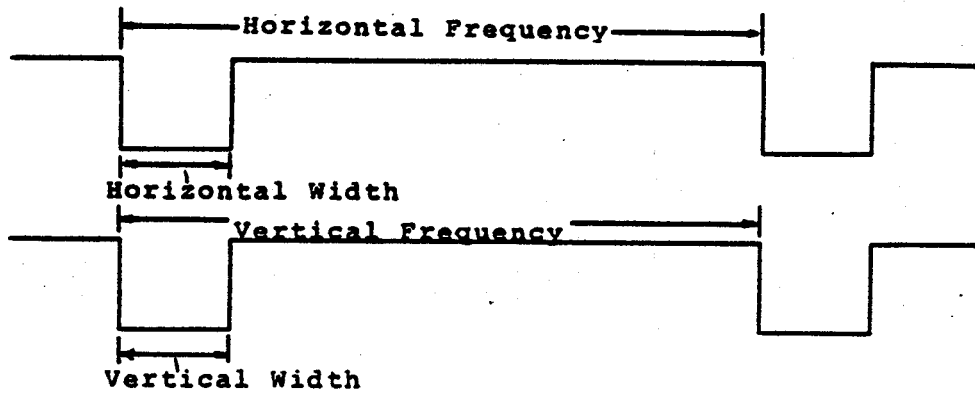
TTL VIDEO

<u>Amplitude</u>	<u>Input Impedance</u>	<u>Video Rise and Fall Time</u>
Low Level (0.0 to 0.8v)=white	220/330ohm Termination to +5v (130ohm)	4 n sec
High Level (+2.0v to +5.2v)=black		

VII SEPARATE SYNC SPECIFICATION

	Amplitude	Input Impedence	Frequency	Width	Rise and Fall Time
Horizontal Sync	TTL compatible phase locks to negative edge LL=0.0 to 0.8v HL=2.0 to 5.2v	220/330ohm termination to +5v (130ohm)		150ns-5us	TTL comp.
Vertical Sync	TTL compatible negative edge Sync LL=0.0 to 0.8v HL=2.0 to 5.2v	220/330ohm termination to +5v (130ohm)	45-65HZ* (other frequencies available as an option)	100ms-300ms	TTL comp.

* If a refresh rate of anything other than 60.0HZ is chosen the low voltage power supply transformer must be shielded with a mumetal shield to prevent a vertical swim problem in the monitor. For countries with 50HZ power, the refresh rate must be 50HZ to prevent the same problem.



ECL INTERFACE SPECIFICATIONS

Specifications: Logic levels shown below gives video on=white,
reverse levels for video off=black

<u>Signal</u>	<u>Connector</u>
Most significant (2^2) bit outer shell is high (-.96v to -.81v) Center is low (-1.85v to -1.65v)	J1
Second most significant (2^1) bit outer shell is High (-.96v to -.81v) Center is low (-1.85v to -1.65v)	J2
Least significant (2^0) bit outer shell is high (-.96v to -.81v) Center is low (-1.85v to -1.65v)	J3

J1,J2,J3, are BNC connectors

ECL VIDEO

<u>Amplitude</u>	<u>Input Impedance</u>	<u>Video Bandwidth</u>	<u>Rise and Fall Time Video Amp</u>
Center conductor (-1.85v to -1.65v)	75ohm without -2v or -5.2v Pulldown	82 MHZ	(10% to 90%) 4.5n sec

Outer shell
(-.96v to -.81v)

Logic levels above video on = white
Reverse levels for video off = black

IX SEPARATE SYNCs - ECL VIDEO BOARD

<u>Signal</u>	<u>Connector Molex (#09-75-1061)J7</u>	<u>Amplitude</u>	<u>Input impedance</u>
Vertical Sync Input	1	TTL compatible negative edge sync	120/180 ohm termination to +5v (72ohm)
GND	2		
Horizontal Sync Input	3	TTL compatible Phase locks to neg. edge	120/180ohm termination to +5v (72ohm)
+5v output (100ma max)	4		
GND	5		
-5v output (100ma max)	6		

J7 Mating connector

Molex # 09-50-3061

Molex Pin # 08-50-0106

See silkscreen drawing for connector layout

See separate syncs page for sync specifications

See ECL interface page for video specifications

X COMPOSITE SYNC - ECL VIDEO BOARD

<u>Signal</u>	<u>Connector</u>	<u>Amplitude</u>	<u>Input impedance</u>
Vertical Sync & Horizontal Sync	(BNC)J4	TTL compatible *LL=0.0 to 0.8v *HL=+2.0 to +5.2v	120/180 ohm termination to +5v (72ohm)

*Low Level *High Level

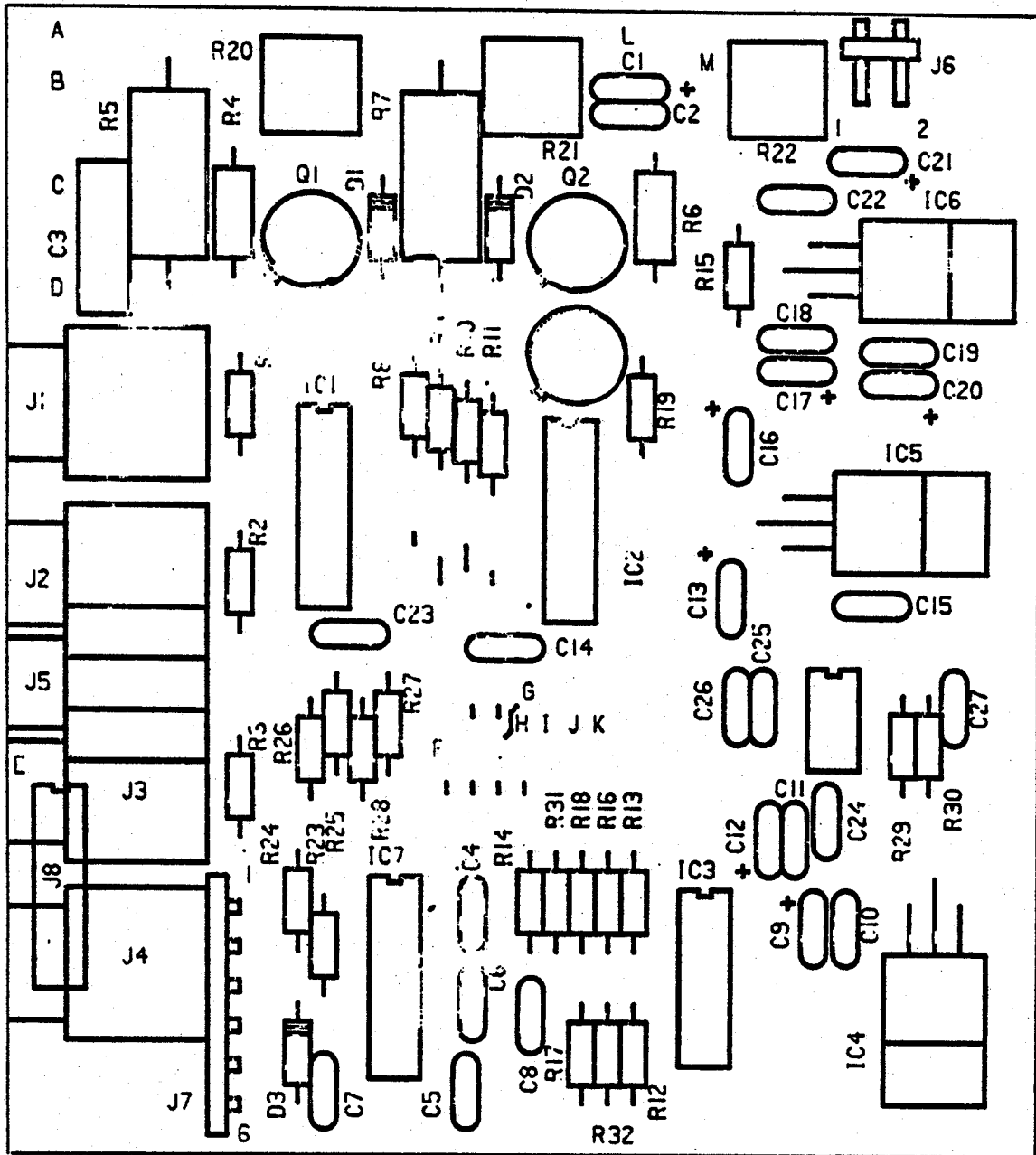
See composite Sync wave form

XI TWO LEVEL COMPOSITE VIDEO

<u>Signal</u>	<u>Connector</u>	<u>Amplitude</u>	<u>Input impedance</u>
Two level composite video	(BNC) J4	Video-Two comparators adjustable from +2.5v to -3.5v Sync-comparator adjustable from +3.5v to -3.5v	75ohm to GND DC coupled

See Two Level Composite Video Option write up

PC BOARD ASSEMBLY



+ DTL820501 SILKSCREEN +

SECTION B DISPLAY TIMING

I Horizontal Timing

The Monitorm Specification includes "back porch" retrace and "front porch" intervals. Since the retrace is phase locked to the falling edge of the sync pulse, and actually starts slightly before it, at least one blank character after the last display character position is recommended. Delaying the horizontal sync additional time causes the display to shift left; thus the user can center the display external to the monitor.

<u>Horizontal Scan</u>	<u>Retrace Time</u>	<u>Video Time</u>
64KHZ + 5%	*3.5 u sec max	11.5 u sec
50 KHZ <u>-</u> 5%	*5 u sec max	15 u sec

*These retrace times are maximum numbers. Since we are using a regulated High Voltage supply, faster retrace times are available. The retrace time and horizontal frequency can be customized to the customer's requirements.

II Vertical Timing

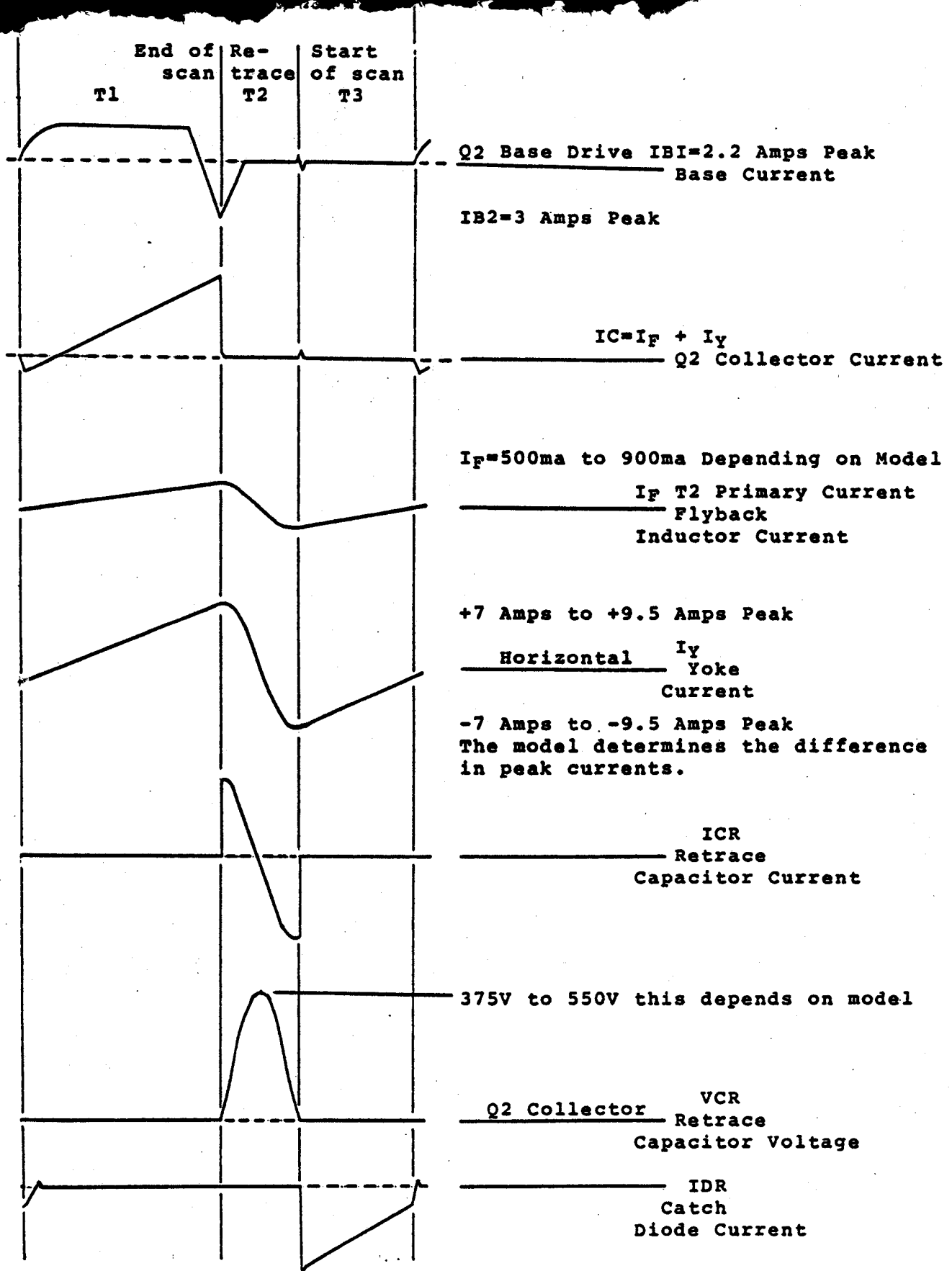
The vertical retrace is initiated on the falling edge of the vertical sync. Best results are obtained if this coincides with the horizontal sync or occurs during horizontal sync. For an interlaced display on alternate frames vertical sync is delayed one half the horizontal time, 7.5us for a 64KHZ horizontal. In any case, total vertical refresh should be a discrete function of the horizontal scan.

The vertical retrace interval is specified at 667us of which approximately 1/2 is beam retrace and 1/2 is settling time. The display is blanked only during the retrace interval. The additional raster lines are available for display although non-linearities are present.

Vertical sync can occur immediately after the last scan of the last display row. Delaying vertical sync additional scan times causes the display to move upward which can facilitate vertical centering or a very smooth scroll, raster by raster (panning).

The vertical oscillator free runs and is factory preset at 7% lower than nominal and will sync to signals initially + 7% from nominal. As with the horizontal setting, any unit for utilization at other than 60HZ should be specified so that vertical lock can be assured.

For the height, sync, and linearity adjustments, see the adjustment section.



End of scan T1 Re-scan trace T2 Start of scan T3

Q2 Base Drive IBI=2.2 Amps Peak
Base Current

IB2=3 Amps Peak

IC=I_F + I_y
Q2 Collector Current

I_F=500ma to 900ma Depending on Model

I_F T2 Primary Current
Flyback
Inductor Current

+7 Amps to +9.5 Amps Peak

Horizontal I_y
Yoke
Current

-7 Amps to -9.5 Amps Peak
The model determines the difference
in peak currents.

ICR
Retrace
Capacitor Current

375V to 550V this depends on model

Q2 Collector VCR
Retrace
Capacitor Voltage

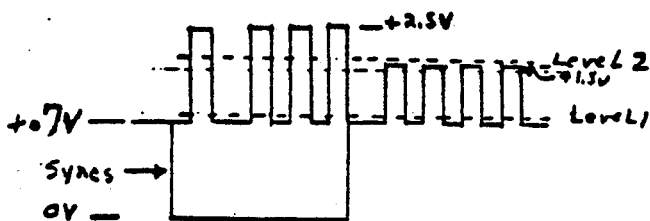
IDR
Catch
Diode Current

III TWO LEVEL COMPOSITE VIDEO OPTION

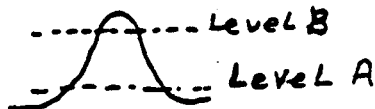
The Two Level Composite Video Interface uses an ECL comparator to sense two discrete video levels. These two levels are set by potentiometers R20 and R21 and can be adjusted between +2.5 to -3.5V.

The Sync is also sensed by a comparator and adjusted by potentiometer R22. The level may be adjusted between +3.5 to -3.5V.

To adjust the Video Comparators, set channel 1 to Video and channel 2 to D.C. potentiometer level. IC7 pin 5 is Level 1 and IC 7 pin 11 is Level 2.



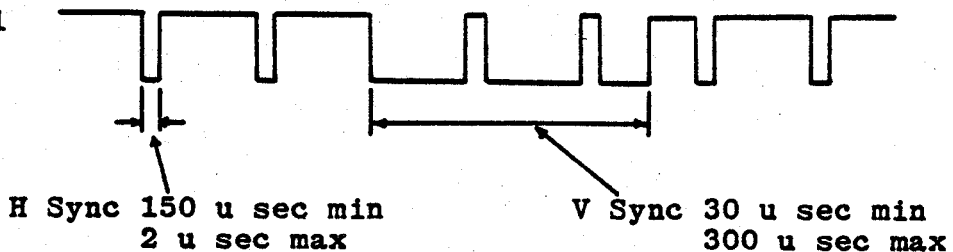
For the example shown, Level 1 would be adjusted to +0.7 V plus the noise level. Level 2 would be +1.5 V plus the noise level. For best rise and fall time of the video the comparators should be adjusted as close to the beginning of the desired video level as possible. An example is shown below.



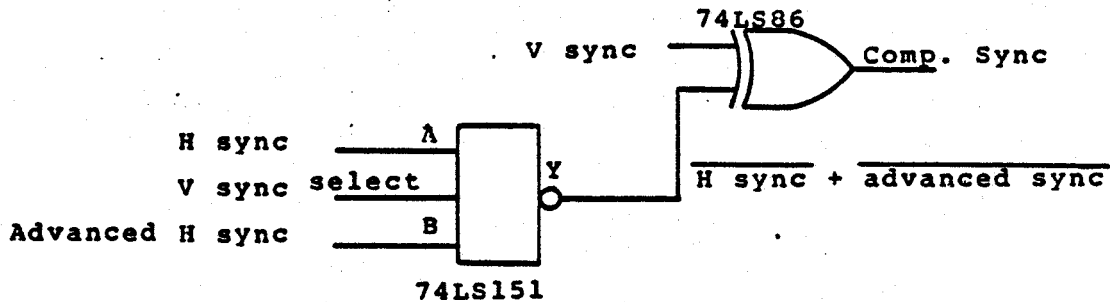
If the video is adjusted to Level A, the single dot characters and the double dot characters will appear the same intensity level. However, if the comparator were set to Level B, the double dot characters would be brighter than the single dot characters.

The Sync should be provided as shown below

TTL High level
2.0v to 5.2v
TTL Low level
0.0 to 0.8v



Note that the Horizontal Sync is advanced by the pulse width of the Horizontal Sync during Vertical Sync. This is done so the Phase Lock isn't out of lock at the end of Vertical Sync. The Phase Lock requires several scan lines to sync up once it is out of lock. A possible circuit is shown below.



C. THEORY OF OPERATION

I. Horizontal Section

IC 3 CD4046 is a phase lock loop (PLL) that drives the horizontal section. The internal oscillator frequency of the PLL is controlled by P2, R9, and C5. The sync input to the PLL is capacitively coupled from Pin G on the video board into Pin 14. The PLL syncs on the positive edge of the H sync pulse. The output of the PLL drives (Pin 4) the gate of the power MOS FET transistor, Q₁.

The drain current of Q₁ is transformer coupled through T1 which provides the base drive for Q₂ (the horizontal output transistor). The horizontal retrace pulse from Q₂ is coupled through the voltage divider of R14 and R11 and is clamped to +12v by Zener diode D4.

This +12v pulse is brought back into the phase comparator of the PLL via Pin 3 of IC3. The output of the phase comparator is low pass filtered at Pin 13 of the PLL by the combination of R6, R10, and C17. The error voltage of the low pass filter is brought into Pin 9, the input to the PLL voltage controlled oscillator (VCO). The VCO sets the frequency of the PLL output (Pin 4). This horizontal drive is directly proportional to the input voltage.

The horizontal yoke has a saw tooth current that swings from +7 amps to -7 amps peak for 15" portrait models, and +9.5 amps to -9.5 amps for the Landscape models. Q₂ clamps the positive yoke voltage to the saturation voltage of the transistor during the positive yoke current. Catch diode D6 clamps the negative yoke voltage during the negative yoke current. When Q₂ is turned off the transition from + to - yoke current C23, 24, and 25 in combination with the horizontal yoke inductance sets the horizontal retrace time. The retrace time voltage wave form is half sine wave called the flyback pulse. The flyback pulse in combination with D5, T2 primary inductance, and C21, determines the boost voltage for the horizontal drive. The boost voltage sets the horizontal energy level and determines the horizontal width. The flyback pulse is stepped down through T2 to provide raw +10v and -10v. The +10v is regulated through IC4 which provides +6v for the CRT filament. The raw +10v and -10v are provided to the video boards via pins I and K respectively. The +10v is regulated on the video board to provide +5v for the TTL logic. The -10v is regulated to -5.2v for the ECL logic.

The horizontal yoke current goes through the linearity coil L1 through S caps C31 and C32 (which help control horizontal linearity) into the horizontal dynamic focus section where the S correction voltage is capacitively coupled through C33 into the primary of T3. The horizontal dynamic focus voltage is stepped up in the secondary of T 3 to approximately 300v and capacitively coupled into the focus grid through C 34 via blue wire 4.

The vertical dynamic focus is brought off C40 and capacitively coupled into the base of the transistor Q3. The collector of Q3 drives producing approximately 250v of vertical dynamic focus.

Power to the horizontal section is provided by the output of IC 1 which provides a maximum of 40v, adjusted by the horizontal width pot P1.

The high voltage power supply provides +1000v and -110v. The 1000v is divided to approximately 500v through P8 and R28 to drive the brightness grid on red wire 3. Also the brightness voltage can be controlled through the brightness transistor Q4, which is controlled by the op amp IC6 and the remote brightness pot. The 1000v is also divided by R27 and P7 to provide approximately 350v of focus voltage on blue wire 4. The -110v goes through D10, R11, and Zener D11 to control grid green wire 2, which is at about -57v at full contrast. The -110v has a "spot killer" circuit consisting of R31, C48, and D10, that holds a negative voltage on the control grid to avoid burning a spot in the CRT after AC power is removed. Power to the high voltage supply is provided by the output of regulator IC2 at approximately 25v.

II VERTICAL SECTION

VERTICAL DEFLECTION CIRCUIT

The heart of the vertical deflection circuit is IC5, the TDA 1170. The IC performs four major functions.

A Power Amplifier and Ramp Generator

Internal Oscillator

Voltage Doubler

Sync Input

The power amplifier provides the power to the vertical yoke from pin 4 of IC5. A current of 1 amp p-p is supplied to the vertical section of the yoke. The yoke current is capacitively coupled through C40 into the sense resistor R21. The sense resistor converts the yoke current into a 1v p-p voltage which is compared against the ramp out of pin 10, and includes the S correction for the vertical axis. This S correction is adjusted by the linearity correction pots P5 and P6.

The Internal Oscillator is set by the RC network R23, C43, and P3. It normally runs in the range from 45-63 Hz.

The input voltage of 25 volts on pin 2 from regulator IC1, is doubled to 50 volts in the doubling circuit D9, C36, and C35.

The 50 volt output on pin 3 is used for the vertical flyback.

Vertical sync input comes in on pin 8 from pin F on the video board connector which is driven by the LS14 on the video board.

This vertical sync input IC4 clamps the sync voltage at .7 volts.

Power to the vertical section is provided by the output of IC2 which generates a voltage of approximately 25 volts.

THEORY OF OPERATION

The TTL video board has a video driver transistor Q1, collector supply voltage regulator IC1, and input buffer IC3, sync buffer IC4, and a +5v regulator (IC2) to drive IC3 & IC4.

The video driver transistor Q1 is a common emitter driver that swings between +30v and +1.8v. The +30v is produced by regulator IC1, TI 783CKC. The regulator is adjustable from 0v to +30v with the contrast Pot P1. This produces the same voltage swing on the cathode (collector of Q1) and also adjusts the control grid G1 from -91v to -61v.

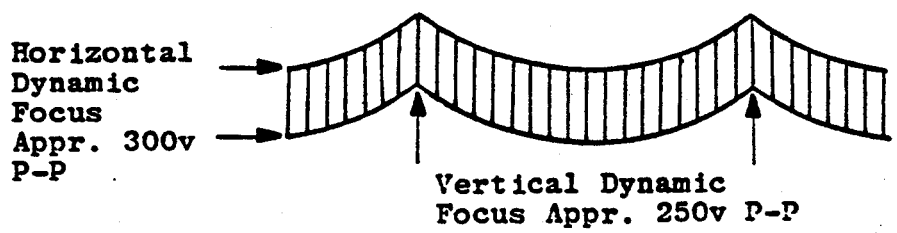
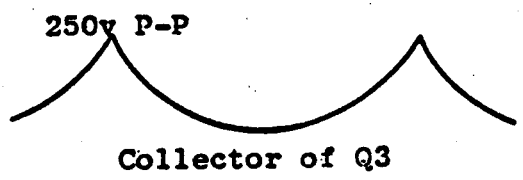
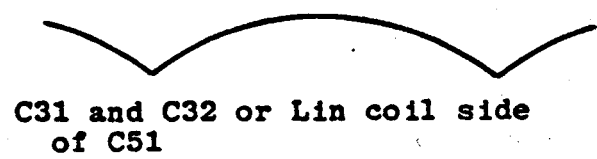
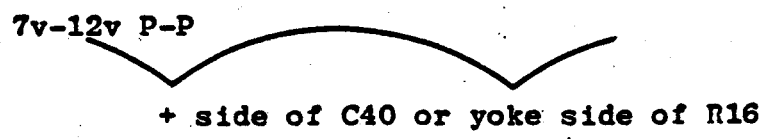
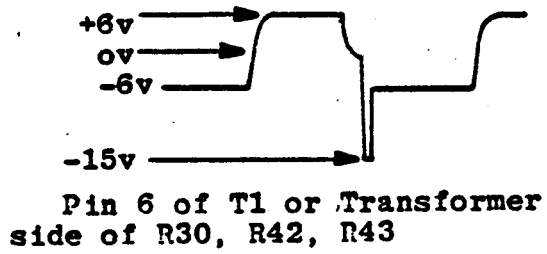
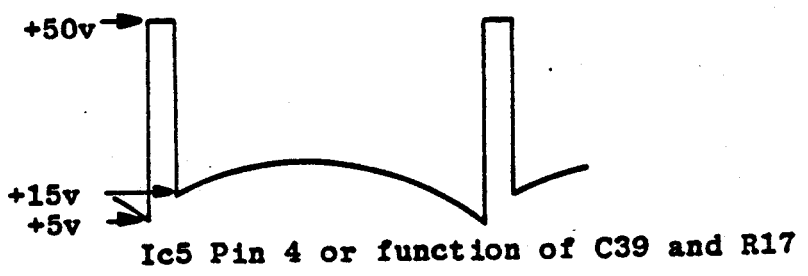
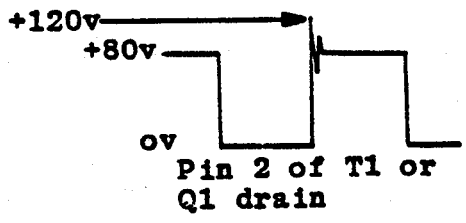
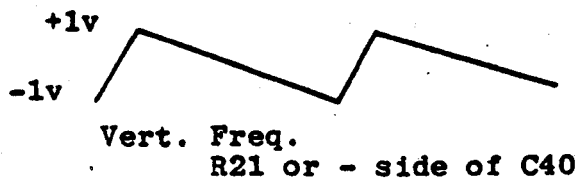
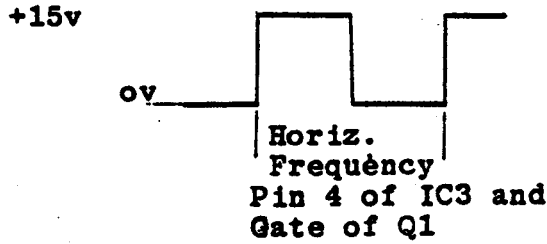
Q1 is kept out of saturation (VCE +1.8v) by the combination of clamp diodes D3 & D4 & the VBE drop of Q1. Peaking inductor L1 speeds up the transistion time from +1.8v to +30v. IC3 (74S04) provides the base drive for Q1.

IC4 (74S14) inverts the horizontal and vertical sync inputs and drives the horizontal phase lock (CD4046) and the vertical deflection IC (TDA1170) on the deflection board. The TDA1170 clamps sync inputs to +.7v and R5 limits the current draw from IC4.

IV ECL VIDEO THEORY OF OPERATION

The ECL video board has a common base video transistor Q1 that drives the cathode and a second common base video transistor Q2 that is capacatively coupled into the control grid (G1). The emitter current of Q1 & Q2 is controlled by IC1 & IC2 (MC10115) defferential input ECL receivers. The emitter follower outputs of IC1 & IC2 are wire-ored, this keeps Q2 off when Q1 is on. Three discrete emitter current levels (60ma, 30ma, 15ma) can be switched into eight different combinations. This emitter current is translated into a voltage change by collector load resistors R4 & R7. As the cathode voltage (Q1 collector) goes from +25v to +9v the control grid voltage (D) goes from -82v to -67v. This collector voltage swing, produced by 100ma of current, gives a differential voltage swing of approximately 30v.

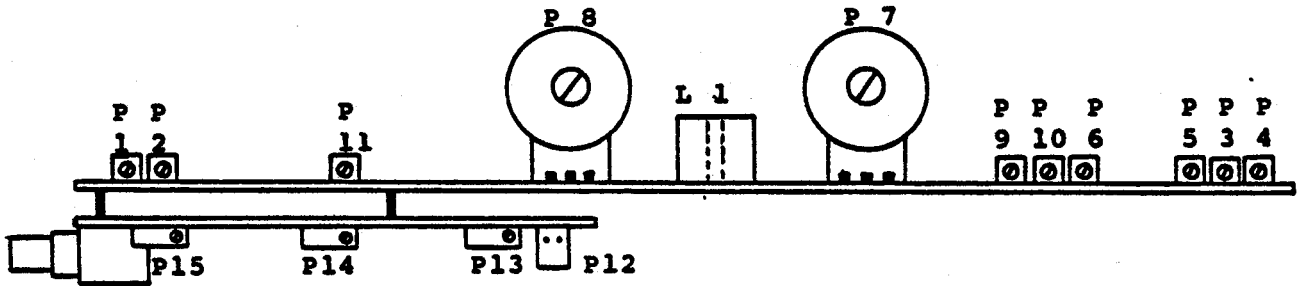
Also on the board are a series of 74LS14 inverters that are used to drive the horizontal and vertical sync inputs.



The dynamic focus voltages vary somewhat from model to model. Wave form at the junction of C34 and R26. (16)

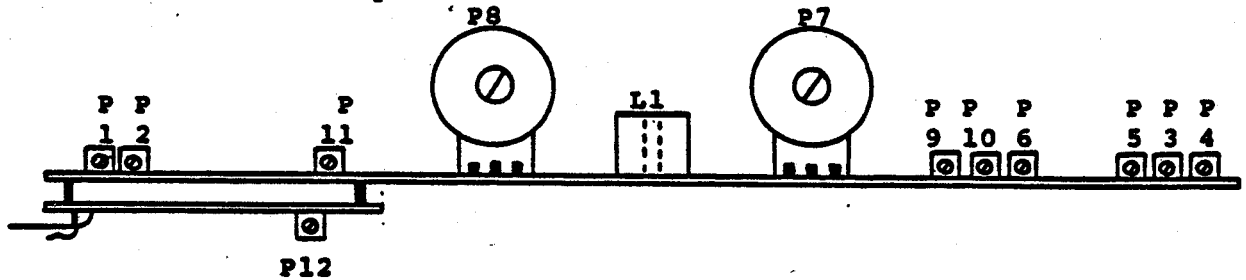
- P1 Horizontal Width
- P2 Horizontal Hold
- P3 Vertical Hold
- P4 Vertical Size
- P5 Vertical Top Bottom Linearity
- P6 Vertical Linearity
- P7 D.C. Focus
- P8 Brightness
- P9 Vertical D.C. Centering
- P10 Vertical Dynamic Focus
- P11 Horizontal Dynamic Focus
- P12 Video Contrast Connector
- P13 Composite Sync Level
- P14 Level 1 Composite Video
- P15 Level 2 Composite Video

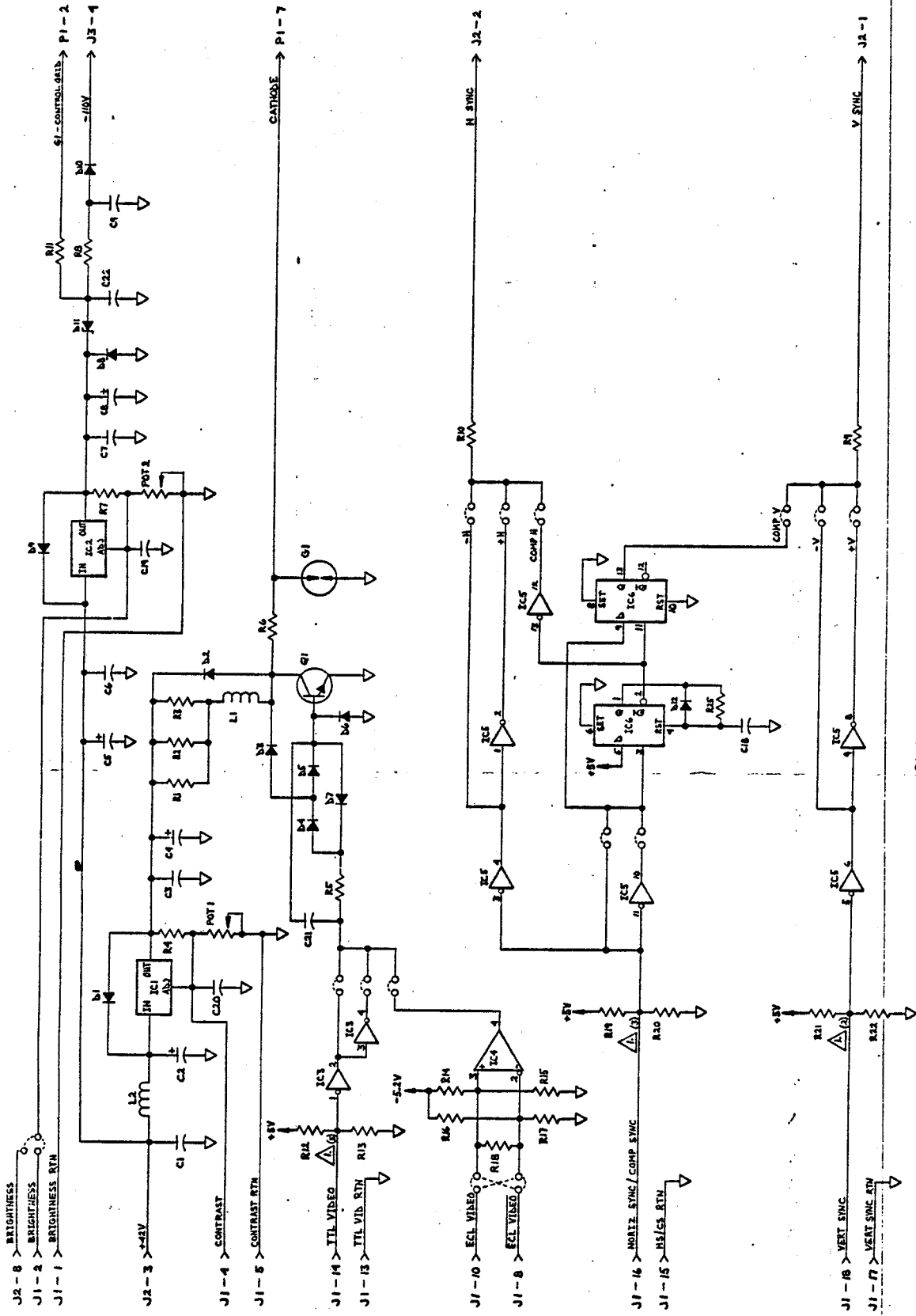
L1 Horizontal Linearity



- P1 Horizontal Width
- P2 Horizontal Hold
- P3 Vertical Hold
- P4 Vertical Size
- P5 Vertical Top Bottom Linearity
- P6 Vertical Linearity
- P7 D.C. Focus
- P8 Brightness
- P9 Vertical D.C. Centering
- P10 Vertical Dynamic Focus
- P11 Horizontal Dynamic Focus
- P12 Video Contrast Connector
- P13 Composite Sync Level
- P14 Level 1 Composite Video
- P15 Level 2 Composite Video

L1 Horizontal Linearity

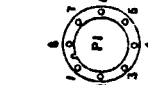




- J3-1 = FILAMENT
- 2 = BRIGHTNESS
- 3 = FOCUS
- 4 = -110V

- J2-8 = BRIGHTNESS
- J1-2 = BRIGHTNESS
- J1-1 = BRIGHTNESS RTN
- J2-3 = +24V
- J1-4 = CONTRAST
- J1-5 = CONTRAST RTN
- J1-14 = TTL VIDEO
- J1-13 = TTL VTB RTN
- J1-10 = ECL VIDEO
- J1-8 = ECL VIDEO
- J1-16 = HORIZ SYNC / COMP SYNC
- J1-15 = HORIZ RTN
- J1-18 = VERT SYNC
- J1-17 = VERT SYNC RTN

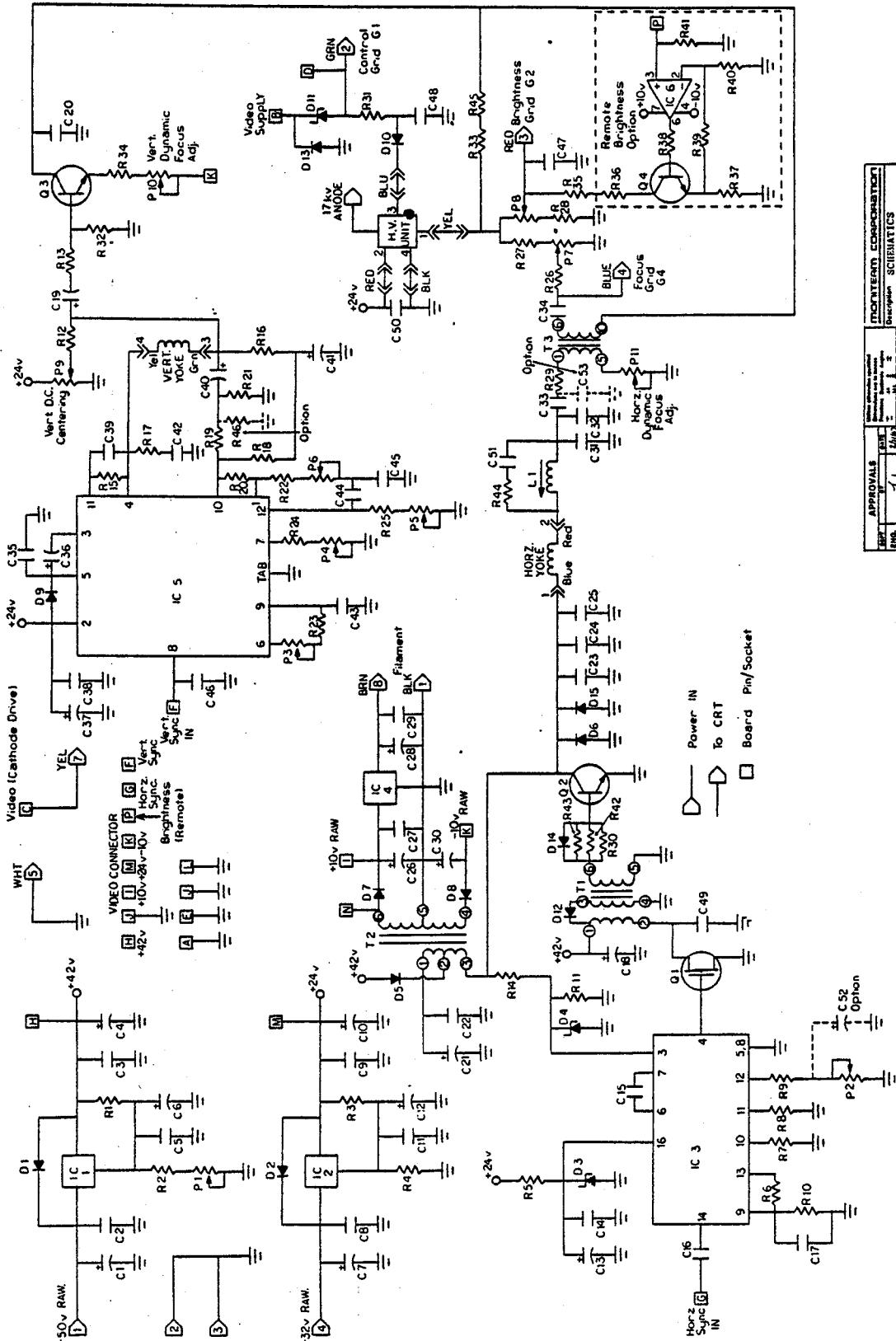
- P1-1 = +5V
- 2 = CONTROL GRID
- 3 = 0V
- 4 = BRIGHTNESS
- 5 = +23 VOLTS
- 6 = GND
- 7 = +10V
- 8 = N/C
- 9 = SPARK GAP RETURN
- 10 = -10V
- 11 = CATHODE
- 12 = GND



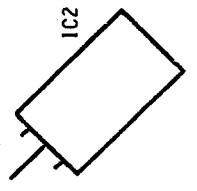
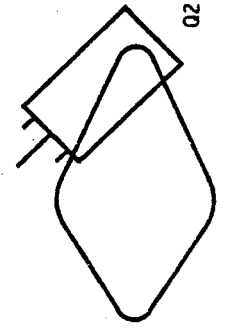
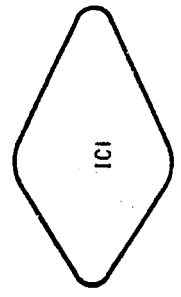
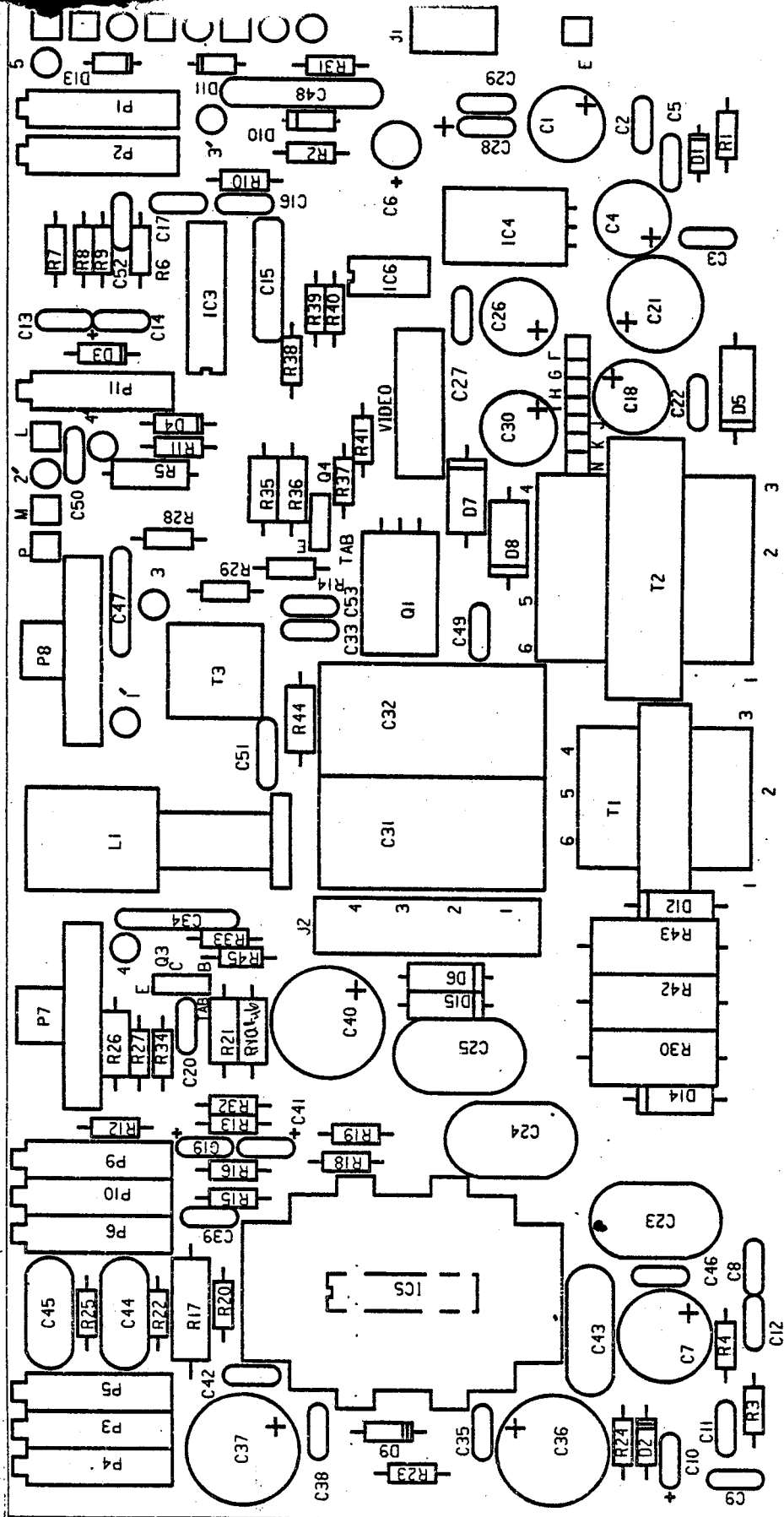
▲ PIN 1'S IN PARENTHESIS INDICATE AP1 (SIP OPTION)

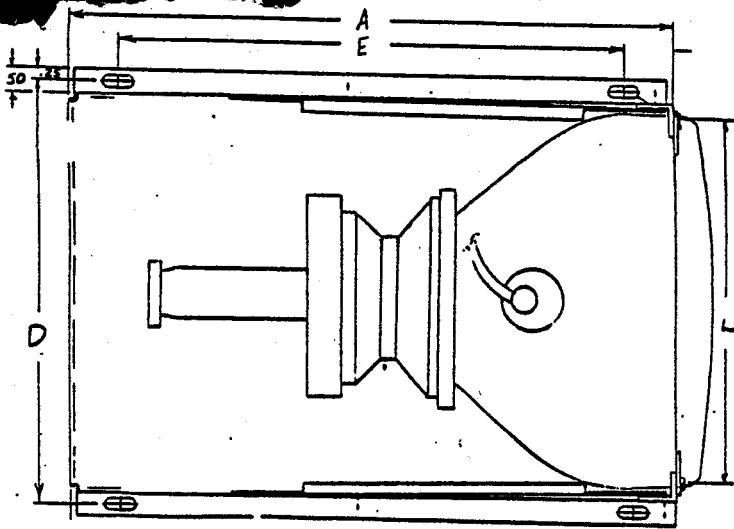
APPROVALS		DATE	
DESIGN	BY		
ENG.	G.A.		
MFG.			
PUR.			
F.S.			

MONITEAM CORPORATION	
Description	SINGLE BIT TTL/ECL
Drawing no.	
Part no.	997-11C5-00
Rev.	REV. 1
Scale	SHEET 1 OF 1

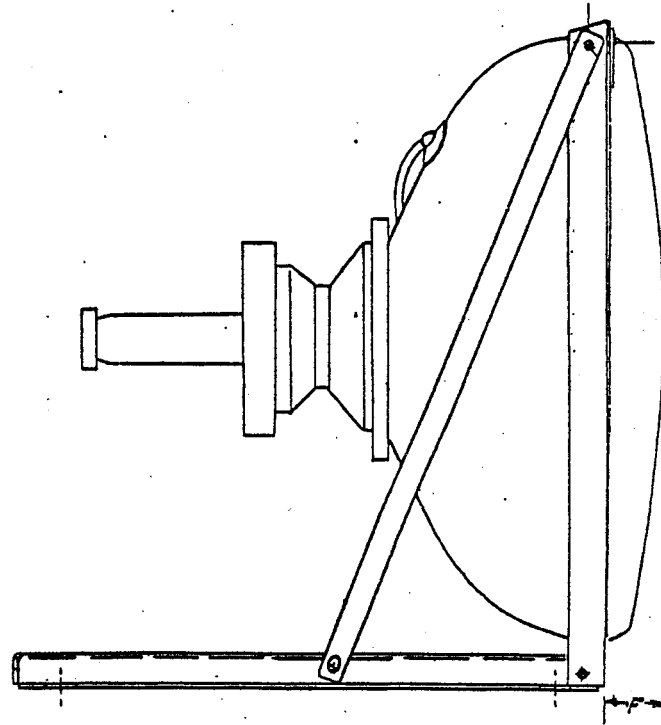
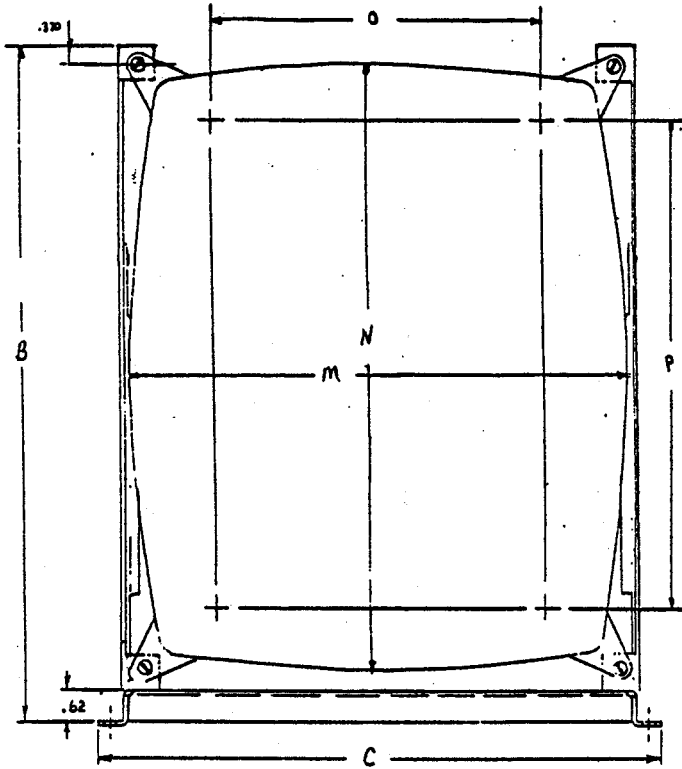


APPROVALS		MONTGOMERY CORPORATION	
DESIGN	DATE	Description	SCHEMATICS
D.A.	1/1	DEFLECTION BOARD	
DRG.		Drawing No.	00112
CHK.		Part No.	870-1100-01
PA.		SCALE	2-1 INCHES UP 1





	15"	17"	20"
A	12.25	13.25	13.25
B	14.00	14.45	18.13
C	11.50	13.00	14.70
D	11.00	12.50	14.20
E	10.25	10.25	10.25
F	1.25	2.25	2.38
L	9.75	11.34	12.87
M	10.00	12.00	13.75
N	13.00	15.00	17.00
O	7.56	9.10	10.5
P	10.14	12.15	13.50



REV. D

DATE OF ISSUE	ISSUED BY	APPROVED BY	DATE
093 H.A.D.			
BASE AND SIZE VR-SERIES		MONTREAU CORPORATION	
CRT MONITOR		80010P	
APPLICATION	DO NOT SCALE DRAWING	SCALE	COPY / 1

TEST PROCEDURE FOR MONITORS

Code Ref. R1

1. Mount the deflection board on the chassis.
2. Connect the high voltage power supply cable to the high voltage power supply.
3. Connect the yoke to the deflection board.
4. Connect the signal inputs to the video board.
5. Power up the pattern generator.
6. Connect the low voltage power supply to the deflection board.
7. Turn on the low voltage power supply.
8. Allow 20 seconds for the display to become operable.
9. Adjust the brightness P8 so the background raster is visible.
10. Adjust the horizontal sync P2 for stable display.
11. Adjust the vertical sync P3 for stable display.
12. Adjust the contrast P9 on the video board to the maximum.
13. Adjust the vertical dynamic focus voltage to a minimum of 200-250V P-P.
14. Adjust the horizontal dynamic focus voltage to a minimum of 300-350V P-P.
15. Make the initial focus adjustment P7 for the best overall focus.
16. Make the initial vertical linearity adjustment by adjusting P5 maximum and the adjusting P5 and P6 in an iterative fashion. Use P4 for the vertical height adjustment. The vertical height should be adjusted to the full display height.
17. Adjust the horizontal linearity coil for maximum width then adjust this for best horizontal linearity. The horizontal linearity should be + or - 5%.

Test Procedures For Monitors Cont.

18. Readjust the vertical linearity by adjusting P5 and P6. Again using P4 for the height adjustment. The vertical linearity should be + or - 5%.
19. Attach the display template to the CRT.
20. Adjust the horizontal size P1 and the vertical size P4 to the boundaries of the template.
21. Adjust the yoke magnets to the boundaries on the template.
22. Install the pincusion magnets on the yoke to the boundaries on the template.
23. Observe that the background raster overscans the displayed area by a minimum of one character time.
24. Adjust brightness P8 so that the background raster is just below the level where it is visible.
25. For units with the remote brightness option check to see that the remote pot will extinguish the raster.
26. Adjust the focus P7 for best overall focus.
27. Adjust the input line voltage to the low voltage supply from 105 vac to 129 vac and observe that the display operates normally.
28. Power down the low voltage power supply and observe that the display doesn't make an extremely bright spot.
29. Stamp tested.
30. 24 Hour burn in.
31. Ship

Code Ref. A3-1

997-1100-01

NEW DEFLECTION BOARD BOM October 12, 1983

<u>BOM</u>	<u>QTY.</u>	<u>PART NUMBER</u>	<u>REF. DESIGNATION</u>	<u>DESCRIPTION</u>
A	1	140-1076-00	C7	CAP 100 uf 63V Aluminum
A	2	140-2271-00	C26,C30	CAP 220 uf 16V Aluminum
A	1	140-2275-00	C18	CAP 22 uf 50V Aluminum
A	1	140-2275-01	C4	CAP 220 uf 50V Aluminum
A	1	140-2761-00	C21	CAP 27 uf 100V Aluminum
A	1	140-3366-00	C1	CAP 33 uf 63V Aluminum
A	2	140-4756-00	C6,C19	CAP 4.7 uf 63V Aluminum
A	3	140-4775-00	C36,C37,C40	CAP 470 uf 50V Aluminum
A	2	142-1021-00	C46,C51	CAP 1K pf 1KV Cer
A	2	142-1032-00	C34,C47	CAP .01 uf 1.4KV Cer
A	1	142-2201-00	C39	CAP 22 pf 1KV Cer
A	2	142-2211-00	C17,C49	CAP 220 pf 1KV Cer
A	1	142-4711-00	C20	CAP 470 pf 1KV
A	1	143-1031-00	C16	CAP .01 uf 100V Cer
A	16	143-1041-00	C2,C3,C5,C8,C9,C11 C14,C22,C27,C29,C33, C35,C38,C42,C50,C52	CAP .1 uf 100V Cer
A	1	143-1045-00	C48	CAP .1 uf 500V Cer
A	2	144-1062-00	C13,C28	CAP 10 uf 25V Tant
A	1	144-2261-00	C41	CAP 22 uf 16V Tant
A	2	144-4755-00	C10,C12	CAP 4.7 uf 50V Tant

Ref. Code A3-2

NEW DEFLECTION BOARD BOM PAGE 2 September 27, 1983 997-1100-01

<u>BOM</u>	<u>QTY</u>	<u>PART NUMBER</u>	<u>REF. DESIGNATION</u>	<u>DESCRIPTION</u>
A	3	146-1041-00	C43,C44,C45	CAP .1 uf 100V 5% Film
A	1	146-1552-00	C31,C32	CAP 1.5 uf 250V Film
A	1	148-1021-00	C15	CAP 1K pf 5% Mica
A	5	160-4004-00	D1,D2,D9,D10,D13	DIODE IN4004
A	1	160-4933-00	D14	DIODE IN4933
A	1	160-4935-00	D12	DIODE IN4935
A	2	160-8500-00	D7,D8	DIODE MR850
A	3	160-8560-00	D5,D6,D15	DIODE MR856
A	1	164-5245-00	D3	DIODE IN5245B
A	1	164-5242-00	D4	DIODE IN5242B
A	1	164-5270-00	D11	DIODE IN5270B
A	1	280-2041-00	J2	CONNECTOR Yoke
A	1	280-5241-00	J1	CONNECTOR P.S.
A	15	282-2202-00		PINS Female
A	1	284-2108-00	IC6	I.C. SOCKET 8-pin
A	1	284-2160-00	IC3	I.C. SOCKET 16-pin
A	1	320-0430-00	L1	LINEARITY COIL
A	1	400-1020-00	P11	POT 1K
A	2	400-1040-00	P3,P4	POT 100K
A	2	400-5020-00	P1,P9	POT 5K

<u>BOM</u>	<u>QTY</u>	<u>PART NUMBER</u>	<u>REF. DESIGNATION</u>	<u>DESCRIPTION</u>
A	4	400-5030-00	P2,P5,P6,P10	POT 50K
A	2	402-2550-00	P7,P8	POT 2.5M 1 Turn
A	1	422-1309-00	Q2	TRANSISTOR MJE 13009
A	1	422-1303-00	Q3	TRANSISTOR MJE 13003
A	1	424-9213-01	Q1	TRANSISTOR RCAREP2N15
A	1	440-3300-15	R17	RES 3.3 ohm 1W 5% C
A	3	440-6800-25	R30,R42,R43	RES 6.8 ohm 2W 5% C
A	1	442-0100-25	R29	RES 10 ohm 1/2W 5% C
A	1	442-0010-25	R21	RES 1 ohm 1/2W 5% C
A	1	442-0102-25	R5	RES 1K ohm 1/2W 5% C
A	4	442-0103-45	R11,R13,R37,R38	RES 10K ohm 1/2W 5% C
A	3	442-0104-45	R32,R39,R40	RES 100K ohm 1/2W 5% C
A	3	442-0105-25	R26,R27,R28	RES 1M ohm 1/2W 5% C
A	3	442-0105-45	R33,R41,R45	RES 1M ohm 1/2W 5% C
A	1	442-0151-25	R44	RES 150 ohm 1/2W 5% C
A	1	442-0181-45	R12	RES 180 ohm 1/2W 5% C
A	2	442-0224-45	R10,R15	RES 220K ohm 1/2W 5% C
A	2	442-0274-25	R35,R36	RES 270K ohm 1/2W 5% C
A	2	442-0333-45	R9,R31	RES 33K ohm 1/2W 5% C
A	2	442-0472-45	R6,R34	RES 4.7K ohm 1/2W 5% C

NEW DEFLECTION BOARD BOM PAGE 4 October 12, 1983 997-1100-01

Code Ref. A3-4

<u>BOM</u>	<u>QTY</u>	<u>PART NUMBER</u>	<u>REF. DESIGNATION</u>	<u>DESCRIPTION</u>
A	2	442-0473-45	R7,R14	RES 47K ohm 1/4W 5% C
A	1	442-0823-45	R24	RES 82K ohm 1/4W 5% C
A	1	444-1002-41	R22	RES 10K ohm 1/4W 1% Film
A	2	444-1913-41	R23,R25	RES 191K ohm 1/4W 1% Film
A	2	444-2002-41	R16,R18	RES 20K ohm 1/4W 1% Film
A	2	444-2430-41	R1,R3	RES 243 ohm 1/4W 1% Film
A	2	444-4531-41	R2,R4	RES 4.53 ohm 1/4W 1% Film
A	1	444-4752-41	R20	RES 47.5K ohm 1/4W 1% Film
A	1	444-4993-41	R8	RES 499K ohm 1/4W 1% Film
A	1	444-5621-41	R19	RES 5.62K ohm 1/4W 1% Film
A	1	480-0210-00	T1	TRANSFORMER T-1
A	1	480-0211-00	T3	TRANSFORMER T-3
A	1	510-4046-00	IC3	I.C. MC14046
A	1	520-0317-01	IC1	I.C. LM317HVK
A	1	520-0317-02	IC2	I.C. LM317T
A	1	520-1170-00	IC5	I.C. TDALL70SH
A	1	520-7806-01	IC4	I.C. MC7806T
A	6	600-0443-01		SCREW 4-40X3/8" S.H.W.H.
A	2	600-0444-02		SCREW 4-40X1/2" S.H.W.H.

NEW DEFLECTION BOARD/OPTION BOM October 12, 1983 997-1100-01

Code Ref. A3-6

<u>BOM</u>	<u>QTY</u>	<u>PART NUMBER</u>	<u>REF. DESIGNATION</u>	<u>DESCRIPTION</u>
A	2	143-1041-00	C52,C53	CAP .1 uf 100V Cer.
A	1	143-8231-00	C53	CAP .082 uf 100V Cer.
A	1	143-6831-00	C53	CAP .068 uf 100V Cer.
A	1	143-5631-00	C53	CAP .056 uf 100V Cer.
A	1	143-4731-00	C53	CAP .047 uf 100V Cer.
A	2	146-1052-00	C31,C32	CAP 1.0 uf 250V 5% Film
A	3	146-6826-00	C23,C24,C25	CAP .0068 uf 600V 5% Film
A	3	146-1036-00	C23,C24,C25	CAP .01 uf 600V 5% Film
A	3	146-1536-00	C23,C24,C25	CAP .015 uf 600V 5% Film
A	3	146-4726-00	C23,C24,C25	CAP .0047 uf 600V 5% Film
A	1		R46	Res. to be determined
A	1	520-0411-00	I.C. 6	I.C. LF411CN
A	1	422-1391-00	Q2	Transistor MJ 13091
A	1	422-1303-00	Q4	TRANSISTOR MJE13003
A	1	480-0213-00	T2	TRANSFORMER T2-1
A	1	480-0212-00	T2	TRANSFORMER T2-2
A	1	480-0214-00	T2	TRANSFORMER T2-3
A	1	480-0237-00	T2	TRANSFORMER T2-4.
A	1		J3	VIDEO CONNECTOR 8-pin
A	2	600-0634-00		SCREW 6-32X $\frac{1}{2}$ P.H.

<u>BOM</u>	<u>QTY</u>	<u>PART NUMBER</u>	<u>REF. DESIGNATION</u>	<u>DESCRIPTION</u>
A	2	602-0632-00		Nut 6-32
A	4	604-0602-00		Lockwasher #6 Internal
A	1	630-4803-00		Heatsink T03
A	2	632-0006-00		Nylon Insert #6
A	1	632-0906-00		TO-3 MICA INSULATOR