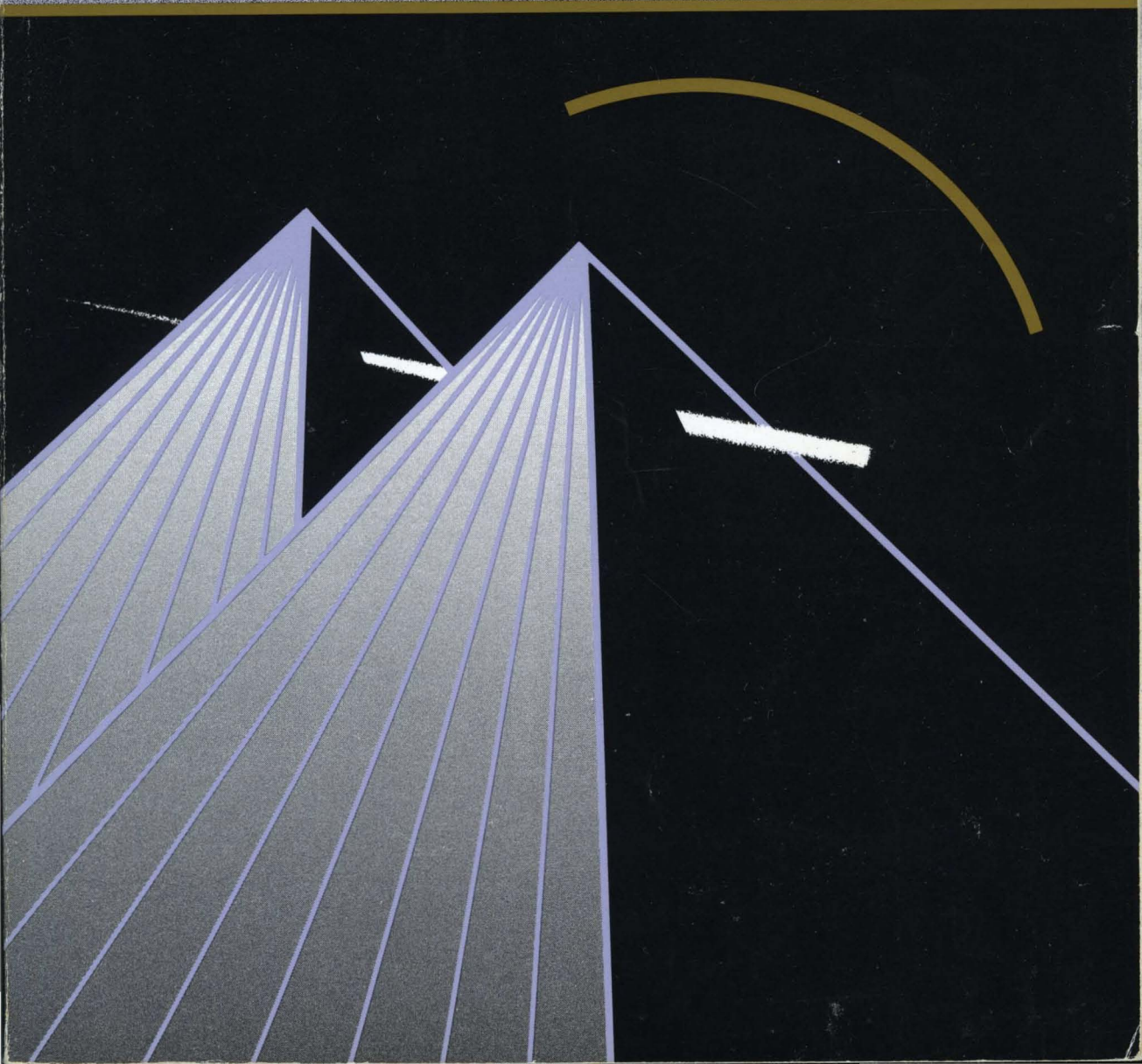
intel

i860™ 64-BIT MICROPROCESSOR
HARDWARE REFERENCE MANUAL

i860™



intel®

LITERATURE

To order Intel Literature or obtain literature pricing information in the U.S. and Canada call or write Intel Literature Sales. In Europe and other international locations, please contact your local sales office or distributor.

INTEL LITERATURE SALES
P.O. BOX 7641
Mt. Prospect, IL 60056-7641

In the U.S. and Canada
call toll free
(800) 548-4725

CURRENT HANDBOOKS

Product line handbooks contain data sheets,application notes, article reprints and other design information.

TITLE	LITERATURE ORDER NUMBER
SET OF 11 HANDBOOKS (Available in U.S. and Canada only)	**231003**
EMBEDDED APPLICATIONS	270648
8-BIT EMBEDDED CONTROLLERS	270645
16-BIT EMBEDDED CONTROLLERS	270646
16/32-BIT EMBEDDED PROCESSORS	270647
MEMORY	210830
MICROCOMMUNICATIONS (2 volume set)	231658
MICROCOMPUTER SYSTEMS	280407
MICROPROCESSORS	230843
PERIPHERALS	296467
PRODUCT GUIDE (Overview of Intel's complete product lines)	210846
PROGRAMMABLE LOGIC	296083
ADDITIONAL LITERATURE (Not included in handbook set)	
AUTOMOTIVE SUPPLEMENT	231792
COMPONENTS QUALITY/RELIABILITY HANDBOOK	210997
INTEL PACKAGING OUTLINES AND DIMENSIONS (Packaging types, number of leads, etc.)	231369
INTERNATIONAL LITERATURE GUIDE	E00029
LITERATURE PRICE LIST (U.S. and Canada) (Comprehensive list of current Intel Literature)	210620
MILITARY (2 volume set)	210461
SYSTEMS QUALITY/RELIABILITY	231762

intel

U.S. and CANADA LITERATURE ORDER FORM

NAME: _____

COMPANY: _____

ADDRESS: _____

CITY: _____ STATE: _____ ZIP: _____

COUNTRY: _____

PHONE NO.: ( ) _____

ORDER NO.	TITLE	QTY.	PRICE	TOTAL

Subtotal _____

Must Add Your Local Sales Tax _____

Postage: add 10% of subtotal → Postage _____

Total _____

Pay by check, money order, or include company purchase order with this form ($100 minimum).We also accept VISA, MasterCard or American Express. Make payment to Intel Literature Sales. Allow 2-4 weeks for delivery.

☐ VISA ☐ MasterCard ☐ American Express Expiration Date _____

Account No. _____

Signature _____

Mail To: Intel Literature Sales
P.O. Box 7641
Mt. Prospect, Il 60056-7641

International Customers outside the U.S. and Canada should use the International order form or contact their local Sales Office or Distributor.

For phone orders in the U.S. and Canada
Call Toll Free: (800) 548-4725

Prices good until 12/31/90.

Source HB

intel

INTERNATIONAL LITERATURE ORDER FORM

NAME: _____

COMPANY: _____

ADDRESS: _____

CITY: _____ STATE: _____ ZIP: _____

COUNTRY: _____

PHONE NO.: ( ) _____

ORDER NO.	TITLE	QTY.	PRICE	TOTAL

Subtotal _____

Must Add Your Local Sales Tax _____

Total _____

PAYMENT

Cheques should be made payable to your local Intel Sales Office (see inside back cover.)

Other forms of payment may be available in your country. Please contact the Literature Coordinator at your local Intel Sales Office for details.

The completed form should be marked to the attention of the LITERATURE COORDINATOR and returned to your local Intel Sales Office.

i860™ 64-BIT MICROPROCESSOR HARDWARE REFERENCE MANUAL

1990

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel products:

376, 386, 387, 486, 4-SITE, Above, ACE51, ACE96, ACE186, ACE196, ACE960, BITBUS, COMMputer, CREDIT, Data Pipeline, DVI, ETOX, FaxBACK, Genius, i, $\overline{\text{i}}$, i486, i750, i860, ICE, iCEL, ICEVIEW, iCS, iDBP, iDIS, I^2ICE, iLBX, iMDDX, iMMX, Inboard, Insite, Intel, int$_e$l, Intel386, int$_e$lBOS, Intel Certified, Intelevision, int$_e$ligent Identifier, int$_e$ligent Programming, Intellec, Intellink, iOSP, iPAT, iPDS, iPSC, iRMK, iRMX, iSBC, iSBX, iSDM, iSXM, Library Manager, MAPNET, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, MultiSERVER, ONCE, OpenNET, OTP, PRO750, PROMPT, Promware, QUEST, QueX, Quick-Erase, Quick-Pulse Programming, Ripplemode, RMX/80, RUPI, Seamless, SLD, SugarCube, ToolTALK, UPI, Visual Edge, VLSiCEL, and ZapCode, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

MULTIBUS is a patented Intel bus.

CHMOS and HMOS are patented processes of Intel Corp.

Intel Corporation and Intel's FASTPATH are not affiliated with Kinetics, a division of Excelan, Inc. or its FASTPATH trademark or products.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

CG-101789

PREFACE

The Intel i860™ 64-bit microprocessor is a general purpose microprocessor integrating an integer RISC core unit, a floating-point unit, a paged memory management unit, instruction and data caches, and 3-D graphics software assist logic in a single VLSI component. The versatile 64-bit design of the i860 microprocessor balances performance across integer, floating point, and graphics processing capability. Its parallel architecture achieves high throughput with RISC design techniques, pipelined processing units, wide data paths, large on-chip caches and fast one micron CHMOS* IV silicon technology.

This manual provides the basic information required to implement an i860 microprocessor based system. It explains the external hardware details of the processor.

Although the main users of this manual are hardware design engineers, it contains basic hardware information which is of value to software engineers and programmers. These readers should reference the first three chapters only.

RELATED PUBLICATIONS

In this manual, the i860 microprocessor is presented from a hardware perspective. Information on the software architecture, instruction set and programming can be found in these related Intel publications:

- *i860™ 64-Bit Microprocessor Programmers Reference Manual*, order number 240329

Information on the device specification for the i860 microprocessor is available in the *i860™ 64-Bit Microprocessor Data Sheet*, order number 240296. Always refer to the most recent version of the device specification.

ORGANIZATION OF THE MANUAL

- Chapter 1, "Introduction to i860™ 64-Bit Microprocessor", provides an overview of the features of the i860 microprocessor and the advantages to system designers. It also provides the insight to i860 microprocessor applications.
- Chapter 2, "Internal Architecture", describes the internal architecture of the i860 microprocessor.
- Chapter 3, "Local Bus Interface", discusses the i860 microprocessor local bus interface. This includes the signal descriptions, bus operation and local bus interface guidelines.
- Chapter 4, "Memory Interfacing", discusses techniques for designing memory subsystems for the i860 microprocessor. The schematics are given in Appendix C.
- Chapter 5, "I/O Interfacing", discusses techniques for connecting I/O devices to an i860 microprocessor system.
- Chapter 6, "Graphics Subsystem Example", discusses a design example for implementing an i860 microprocessor based graphics subsystem.
- Chapter 7, "MULTIBUS® II and the i860™ Microprocessor", provides a design example for the MULTIBUS II board built around the i860 microprocessor.

- Chapter 8, "Physical Design and Debugging", contains recommendations for constructing and debugging the i860 microprocessor based systems.
- Chapter 9, "Testability", covers the testability issues on the i860 microprocessor.
- Appendix A provides tested schematics of an i860 microprocessor Graphics Frame Buffer Board and is to used in conjunction with Chapter 6.
- Appendix B contains an untested example for the i860 microprocessor based MULTIBUS II design that is explained in Chapter 7.
- Appendix C provides the tested schematics and PLD codes for i860 microprocessor memory design.

Signal mnemonics convey whether a signal state is active high or active low. An active-low signal mnemonic is suffixed with a pound character (#); an active-high signal mnemonic does not have this suffix.

TABLE OF CONTENTS

CHAPTER 5
I/O INTERFACING

CHAPTER 6
GRAPHICS SUBSYSTEM EXAMPLE

Figures

Tables

CUSTOMER SUPPORT

INTEL'S COMPLETE SUPPORT SOLUTION WORLDWIDE

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, consulting services and network management services. For detailed information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is quite extensive. It can start with assistance during your development effort to network management. 100 Intel sales and service offices are located worldwide—in the U.S., Canada, Europe and the Far East. So wherever you're using Intel technology, our professional staff is within close reach.

HARDWARE SUPPORT SERVICES

Intel's hardware maintenance service, starting with complete on-site installation will boost your productivity from the start and keep you running at maximum efficiency. Support for system or board level products can be tailored to match your needs, from complete on-site repair and maintenance support to economical carry-in or mail-in factory service.

Intel can provide support service for not only Intel systems and emulators, but also support for equipment in your development lab or provide service on your product to your end-user/customer.

SOFTWARE SUPPORT SERVICES

Software products are supported by our Technical Information Service (TIPS) that has a special toll free number to provide you with direct, ready information on known, documented problems and deficiencies, as well as work-arounds, patches and other solutions.

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and; *COMMENTS Magazine*). Basic support consists of updates and the subscription service. Contracts are sold in environments which represent product groupings (e.g., iRMX® environment).

CONSULTING SERVICES

Intel provides field system engineering consulting services for any phase of your development or application effort. You can use our system engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training and customizing an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, BITBUS™ and LAN applications.

NETWORK MANAGEMENT SERVICES

Today's networking products are powerful and extremely flexible. The return they can provide on your investment via increased productivity and reduced costs can be very substantial.

Intel offers complete network support, from definition of your network's physical and functional design, to implementation, installation and maintenance. Whether installing your first network or adding to an existing one, Intel's Networking Specialists can optimize network performance for you.

CG/CUSTSUPP/100389

Introduction to the i860™ 64-Bit Microprocessor

1

CHAPTER 1
INTRODUCTION TO THE i860™ 64-BIT MICROPROCESSOR

This chapter provides an overview of the i860™ microprocessor, its characteristics, possible system configurations and applications.

The i860 microprocessor uses pipelining, parallelism and reduced instruction set computer (RISC) design techniques for high performance. With over one million transistors, it integrates an integer unit, a floating-point unit, a graphics unit, a memory management unit, and separate data and instruction caches.

The i860 microprocessor executes up to two instructions in parallel and makes extensive use of pipelining. High integration and wide data paths eliminate bottlenecks in fetching instructions or data from its on-chip caches.

Designing with the i860 CPU is like designing with any other microprocessor. Its high-performance can be easily delivered using standard interface logic, DRAMs, EPROMs, and I/O devices. An existing 32-bit microprocessor-based system can be easily upgraded with the 64-bit i860 CPU by widening the memory bus.

1.1 PROCESSOR CHARACTERISTICS

The i860 microprocessor's 1 million transistors offer a high level of integration. A single board design can be completed with four address latches, eight data tranceivers, a clock generator circuit, one EPROM for bootstrapping, a 64-bit wide memory and a few PLDs.

The i860 microprocessor contains the following features:

- Integer processing unit
- Floating-point and graphics unit
- Memory management unit
- 8 Kbyte data cache with 128-bit internal data path
- 4 Kbyte instruction cache with 64-bit internal data path

The i860 microprocessor uses RISC design techniques such as:

- Single clock instruction execution
- Load and store architecture
- 32 general purpose 32-bit registers
- Delayed branching

- Register scoreboarding
- Register bypassing
- Single-cycle loop instruction
- Branch taken/not-taken instruction fetching optimization

The on-chip graphics and floating-point units of the i860 microprocessor support simulation and 3-D displays. The integer instructions can be used to feed data to the graphics and floating-point units. These units are pipelined to produce up to two floating-point results per clock. Some of the features related to the graphics and floating-point unit are:

- Separate adder and multiplier units
- Pipelined/parallel floating-point hardware
- Graphics instructions and hardware optimized for 3-D
- 8-, 16- or 32-bit color or black and white pixel data types
- Single- and double-precision IEEE floating-point standard
- Parallel integer and floating-point operation

The i860 microprocessor's high-performance design uses wide buses and pipelined logic to sustain many parallel operations. The level of performance of the i860 microprocessor is difficult to achieve in multichip systems due to the need for many wide buses. The high-performance design applies to the external bus as well, maximizing the performance that can be obtained from DRAM memory. Some of the performance related features are:

- 64-bit internal and external data bus
- Three operations per clock
- Dual-instruction mode
- 33/40 MHz operating frequency
- 80 million floating-point operations per second peak in single-precision
- 60 million floating-point operations per second peak in double-precision
- 320 Mbyte/sec instruction cache bandwidth at 40 MHz
- 640 Mbyte/sec data cache bandwidth at 40 MHz
- 160 Mbyte/sec external bus bandwidth at 40 MHz
- Fast data movement with 128-bit load and store instructions
- Three pipelined cycles on the external bus

The i860 microprocessor can support virtual memory and multiprocessor systems. All memory references are fully restartable in case of virtual memory faults. Full support is provided for synchronizing operation between multiple CPUs. Features related to virtual memory and multiprocessor support are:

- 32-bit (4 gigabyte) address space with on-chip paging unit
- Support for demand paging
- 4 Kbyte page size
- On-chip 64-entry translation lookaside buffer (TLB)
- User level/supervisor level protection
- Bus locking across multiple instructions
- Cache control
- Trap mechanism for interrupts and faults
- Multiprocessor operating systems

Use of the i860 microprocessor design simplifies system design. It has a conventional microprocessor bus with a standard READY# signal used to allow accesses to slow memories. Software development is assisted by specialized hardware debugging capabilities. Some of the additional implementation features to facilitate hardware design and software development are:

- 1× clock
- Optimizations for use of page mode and static mode dynamic RAMs
- Pin boundary scanning for component or board testing
- CHMOS* IV 1-micron technology, TTL compatible
- 3 watt power dissipation at 40 MHz
- Single 8-bit EPROM can boot system
- On-chip debugging support

1.2 PROCESSOR OVERVIEW

This section provides a quick overview of the i860 microprocessor. The processor architecture will be discussed in greater detail in Chapter 2, and the external interface in Chapter 3.

1.2.1 Pipelining and Parallelism

Pipelining is a technique that divides tasks into a series of smaller subtasks which can be performed quickly and concurrently with one another. By means of pipelining, several data items can be acted on simultaneously. Although several clock cycles are needed to complete an operation, a new result is produced each clock. Pipelining is used throughout the i860 microprocessor to achieve maximum performance. Simple scalar operations can be used to hide details of the pipelining. All scalar floating-point and integer operations are fully interlocked.

Parallelism allows two or more operations to execute simultaneously. The i860 microprocessor supports parallelism, allowing multiplication and addition to execute simultaneously within the floating-point unit. Parallelism also allows the integer and floating-point units to execute simultaneously. The i860 microprocessor can execute an integer instruction and two floating-point operations in the same clock to achieve three operations per clock cycle.

1.2.2 Instruction Set Architecture

The instruction set architecture can implement any portable operating system, language or application. The instruction set includes integer, floating-point, and graphics instructions. All future products of the i860 CPU line will support this architecture.

The core unit executes the basic instruction set including arithmetic, logical, shift and program control instructions. The core unit, floating-point unit and graphics unit are all implemented using RISC principles. All instructions execute in only one clock cycle.

The i860 microprocessor uses a load and store architecture. Only load and store instructions can access memory data. All other instructions use only registers to perform operations. Integer and floating-point instructions use two operand registers and a destination register for the result.

The i860 microprocessor employs delayed branching techniques to avoid interruptions to the flow of data through pipelines. An optimized set of conditional branch and loop instructions minimizes pipeline breaks. Use of a branch-taken or branch-not-taken version of the conditional branch instruction selects whether to pre-fetch the upcoming sequential instruction or the branch target address.

1.2.3 Registers

The i860 microprocessor provides 32 32-bit integer registers for the core unit and another 32 32-bit floating-point registers that are used in the floating-point unit and the graphics unit. The floating-point registers can also be used in pairs as double-precision registers. Quadword memory load instructions load four floating-point registers. Register **r0** of the integer registers and **f0** and **f1** of the floating-point registers return a zero value when read and are treated as a null destination when a value is stored into them.

1.2.4 Address Space

The i860 microprocessor can address up to four gigabytes of memory and memory-mapped I/O locations. Programmers can access memory space as 8-, 16-, 32-, 64- and 128-bit quantities. Operands of 8 and 16 bits are automatically aligned to the low order bits of a 32-bit register on a load. An improperly aligned data access causes a trap.

1.2.5 Floating-Point Operations

The floating-point unit supports the ANSI/IEEE 754-1985 Standard for Binary Floating-point Arithmetic and supports both single- and double-precision operands. Traps detect all floating-point exceptions. Hardware implements all rounding modes in order to support the standard with minimal overhead. The floating-point unit provides a full complement of operations to permit efficient implementation of all low-level and high-level functions defined by the standard.

The parallel floating-point adder unit and multiplication unit have been designed to efficiently perform matrix manipulation, series expansion and signal processing algorithms. The integer unit along with the pipelined and parallel operation of the floating-point unit can emulate the capability of vector processing instructions found in supercomputers, but with added flexibility. The scalar floating-point operations hide all details of the pipeline.

The floating-point load instructions provide 32-, 64- and 128-bit operands for the floating-point unit while it operates in dual-instruction mode. Integer operations can be performed in parallel with all floating point operations.

1.2.6 Graphics Support

The graphics unit supports instructions for 3-D color or black and white algorithms. Pixel formats of 8-, 16- or 32-bit are supported. The i860 microprocessor supports efficient implementation of Phong and Gouraud shading operations. A Z-buffer check instruction is provided to detect the closest surface of a 3-D image. Either 16- or 32-bit Z-buffer can be used. The 3-D graphics unit uses much of the floating-point point hardware.

1.2.7 Caches

Two on-chip caches help to sustain the i860 microprocessor's high performance. The 8 Kbyte data cache is a two-way set-associative memory with a 32-byte block size and a 128-bit data path. It uses a write-back policy on memory write operations. This technique delays the external write operations needed to maintain consistency between the cache and external memory. With this approach, multiple write operations to the same location, do not result in needless multiple bus operations. When write operations to external memory are performed, two 128-bit wide write buffers are used which post the write operations and delay them until the memory subsystem is not in use.

The 4 Kbyte instruction cache is also implemented as a two-way set associative memory. It provides a 64-bit wide internal datapath. The instruction cache is read-only; writes to memory do not update the code cache. A separate flush instruction is available to invalidate the contents of code cache, if the need arises.

External caches are supported via cache control pins, cache control bits in page table entries, and a two clock bus. Multiprocessing systems can be built using external caches and preventing shared data from being cached on-chip.

Since the data and instruction caches map virtual addresses to data, the data and instruction caches can operate in parallel with the translation lookaside buffer (TLB). Data can be obtained in one clock cycle when there is a hit in the cache.

1.2.8 Paging Unit

The on-chip paging unit converts 32-bit virtual memory addresses to 32-bit physical addresses. Each page is 4 Kbytes in size. Supervisor and user level read and write memory protection is provided on a per page basis. Supervisor pages can be write-protected to perform copy-on-write operations for supervisor data.

A translation lookaside buffer (TLB) acts as a 64-entry cache for the virtual memory tables. These tables map virtual page addresses to physical page addresses and provide protection rights. The TLB makes memory management more efficient by operating in parallel with the data and instruction caches. TLB cache misses and updates are handled automatically in hardware.

1.2.9 Debugging Support

The i860 microprocessor provides a debug hardware support trap which can be activated when reading and/or writing at an address stored in the data breakpoint register. The address can refer to data inside the caches or off-chip.

1.2.10 External Interface

This section outlines the functions provided by the i860 microprocessor's external interface. These functions are detailed in Chapter 3.

The external interface pin-out consists of a 29-bit address bus, an 8-bit byte enable control bus, a 64-bit data bus, 19 control signals and 48 power and ground pins. The external bus is timed relative to a clock. All outputs are valid before the end of a clock period. All inputs are synchronous to the clock.

1.2.10.1 TWO-LEVEL BUS PIPELINING OF LOCAL BUS

The i860 microprocessor permits up to two levels of pipelining in external memory operations, providing throughput beyond the cycle or access time of the components used. A two level configuration allows three operations to occur simultaneously and triples memory throughput. While the total cycle time is six clocks, 64 bits of data are transferred every two clocks. Pipelining permits a high-performance memory system while using low-cost DRAMs. Even with a single bank of DRAMs, pipelining allows overlapping of accesses to the same DRAM page.

1.2.10.2 BUS ARBITRATION SUPPORT

The i860 microprocessor provides three signals to control bus and control line arbitration: the input line HOLD (bus hold request) and the output lines HLDA (hold acknowledge) and BREQ (bus request). HOLD and HOLDA provide a handshake with

other processors or an arbitration circuit to allow several processors to share the external buses. The processor can operate out of its internal cache for periods of time and uses the BREQ signal to indicate that it is waiting to use the external bus.

1.2.10.3 DRAM INTERFACE SUPPORT (PAGE-MODE AND STATIC COLUMN)

The i860 microprocessor provides support for page mode and static column DRAMs. These provide faster memory cycles when sequential reads or writes take place from the same row address in a DRAM component. This occurs often, because cache updates typically involve four 64-bit sequential read cycles and the write buffers often group together four sequential write cycles.

The i860 microprocessor provides a next near signal (NENE#) to reduce the amount of external circuitry needed to support page mode and static column DRAMs. The signal is asserted when successive current cycles are in the same DRAM page. The size of the DRAM can be programmed.

1.2.10.4 CACHE CONTROL

The i860 microprocessor supports memory mapped I/O, external caches and multiple CPUs. All of these require correct use of caching. The cache enable input signal (KEN#) and page table bit output signal (PTB) control and monitor the data and instruction caches. The KEN# signal input is used by external logic to tell the processor when read data should not be cached. This prevents the processor from caching shared memory in a multiprocessor system and alleviates inconsistencies when two or more processors are accessing the same area of memory. All read operations from memory mapped I/O locations must deassert KEN#.

The page table bit (PTB) output signal operates in two modes. In one mode it indicates whether the software has disabled updates to the data or code cache during the current read cycle. In its other mode of operation it indicates whether the software has disabled the use of an external cache for the current cycle. This kind of software control of caching is done on a page by page basis, only when paging operation is enabled. The software controls the use of the cache in order to guarantee the consistency of shared data in a multiprocessor environment.

1.2.10.5 LOCKED MEMORY CYCLES

The external LOCK# pin indicates that the i860 microprocessor is performing a set of memory cycles that should not be interrupted. The memory subsystem should be designed so that the memory subsystem cannot be accessed by other processors while this pin is active. Multiple instructions can be executed while the lock signal is active. All semaphore operations, like compare and swap, can be implemented.

1.2.10.6 EIGHT-BIT BUS ACCESS FOR BOOTSTRAPPING OPERATION (CS8)

The INT/CS8 input pin can establish 8-bit code accesses upon hardware reset. This feature allows bootstrapping from an 8-bit external EPROM or ROM device and reduces the number of parts needed to complete a board. Only code accesses are affected in this mode. This mode only works immediately after activation of RESET. A control register allows software to disable this mode, but not to reenable it.

The entire code and initial data of an application or bootup code can be placed into one EPROM and copied to DRAM for execution.

1.2.10.7 BOUNDARY SCAN TO SIMPLIFY BOARD LEVEL TESTING OR BOOTUP CODE

The i860 microprocessor allows all the input pins to be serially sampled. Likewise, the state of all the output pins can assume a value set by serial input data. This precludes the need for complex programs that are normally needed to manipulate pins and permits the testing of board logic circuitry with the i860 microprocessor installed.

1.2.11 Clock Requirements

The i860 microprocessor uses an external clock that runs at 33.3 or 40 MHz. This clock synchronizes the internal functional blocks of the processor, and synchronizes the external signals. Most logic connected to the i860 CPU will also use this clock.

1.2.12 i860™ Microprocessor Packaging and Power Requirements

The i860 microprocessor is available in a 168-pin pin-grid array (PGA) package with 120 signal pins and 48 power and ground pins. All the power and ground pins must be connected. Low-inductance bypass capacitors should be used around the i860 CPU to handle current surges when all the address and data lines change state simultaneously. Direct current power dissipation when running at 33 MHz is two watts at normal operation and three watts at peak operation. A heat sink may be used to keep the case temperature within specification depending on airflow.

1.3 SYSTEM CONFIGURATION

The i860 microprocessor is suitable as a central processor in any high performance microprocessor application. It will usually be configured as a stand-alone processor with private memory and a memory mapped I/O subsystem. The i860 uses standard DRAMs and an EPROM for operation. A small system consists of i860 CPU, clock generator, 4 address latches, 8 data transceivers, 1 EPROM, 64-bit memory of DRAM/SRAM, several PLDs and I/O devices.

Other configurations employ the i860 microprocessor as a dedicated application processor. In this case, a communications bus will exist between the i860 and the host. This bus should allow quick data transfers.

The processor may also be used in a multiprocessor system. The i860 CPU may work in a loosely-coupled fashion, communicating through shared memory or through a link of independent memories. The LOCK# signal must be used to guarantee atomic multiple-cycle memory accesses.

The i860 microprocessor provides several functions for multiprocessing support. Bus granting logic (HOLD, HOLDA, BREQ) eases the interface with other processors and DMAs. The cache enable/disable logic is used to maintain consistency between internal caches and shared memory. The capability for locked external cycles allow implementation of semaphores for use with shared memory.

1.3.1 Private and Shared Memory Configuration

Peak performance for a multiprocessor i860 system requires minimal contention for the use of memory. Typically, this is achieved by providing each i860 microprocessor with cache memory, its own private memory, memory that can be shared, or some combination.

Multiprocessor systems not requiring peak performance can use shared memory only. This results in a simpler and lower-cost implementation. The memory subsystem can prioritize access to the shared memory between the processors to maximize efficiency.

1.4 APPLICATION OVERVIEW

The i860 microprocessor is designed for use in a wide range of applications. The processor's support of demand-paged virtual memory and the IEEE Floating Point Standard makes it especially suitable as a main processing engine for high-performance engineering workstations, mainframe computers and supercomputers. Its high integration allows a desktop form factor for capabilities previously associated with supercomputers and high-end graphics workstations.

The i860 power can support computation intensive applications such as electronics or mechanical system simulations. The power of the hardware 3-D graphics support graphics applications. High-performance 3-D graphics can now become a standard feature of any workstation.

The i860 microprocessor eliminates the need for special-purpose signal, graphics or floating-point coprocessors. The processor can also be used for high-performance embedded controller applications, or as an applications accelerator for existing systems.

Internal Architecture 2

CHAPTER 2
INTERNAL ARCHITECTURE

The i860 microprocessor architecture obtains its performance by a combination of large data paths, high bandwidth data and instruction access (caches and registers), a large number of important functions included on-chip, and high levels of pipelining and parallelism. This architecture allows up to three operations per clock to be executed.

The programmer controls the parallelism to manage data flow to and from the floating-point unit. The on-chip caches provide storage for instructions and data. Wide buses can transfer an instruction pair and two double-precision floating-point operands each clock. A programmer can use the data cache as a large bank of vector floating-point registers.

The i860 microprocessor architecture consists of nine units:

- Core execution unit
- Floating-point control unit
- Floating-point adder unit
- Floating-point multiplier unit
- Paging unit
- Data cache unit
- Instruction cache unit
- Bus and cache control unit
- Graphics unit

The arrangement of these nine units is shown in Figure 2-1. This chapter describes how these nine units interact to interpret the instructions.

2.1 CORE EXECUTION UNIT

The core execution unit is the center of intelligence for the i860 microprocessor and is responsible for its overall operation. It fetches both integer and floating-point instructions. It decodes and executes integer, logical, control-transfer, load/store, exception handling, and cache flushing instructions. It can perform loads and stores to and from the integer register file and the floating-point register file. It also includes a special pixel store instruction that facilitates implementation of the Z-buffer hidden-surface elimination algorithm.

2.1.1 Core Unit Registers

The core execution unit includes a register file containing thirty-two 32-bit integer registers, a 32-bit ALU, a barrel shifter, two 32-bit processor status registers, a data breakpoint register, a fault instruction register and control logic.

The integer registers, labeled **r0** through **r31**, are accessible by arithmetic operations and load/store instructions. These registers are used for address computation and scalar integer computations. All the registers can be read and written except **r0**, which always

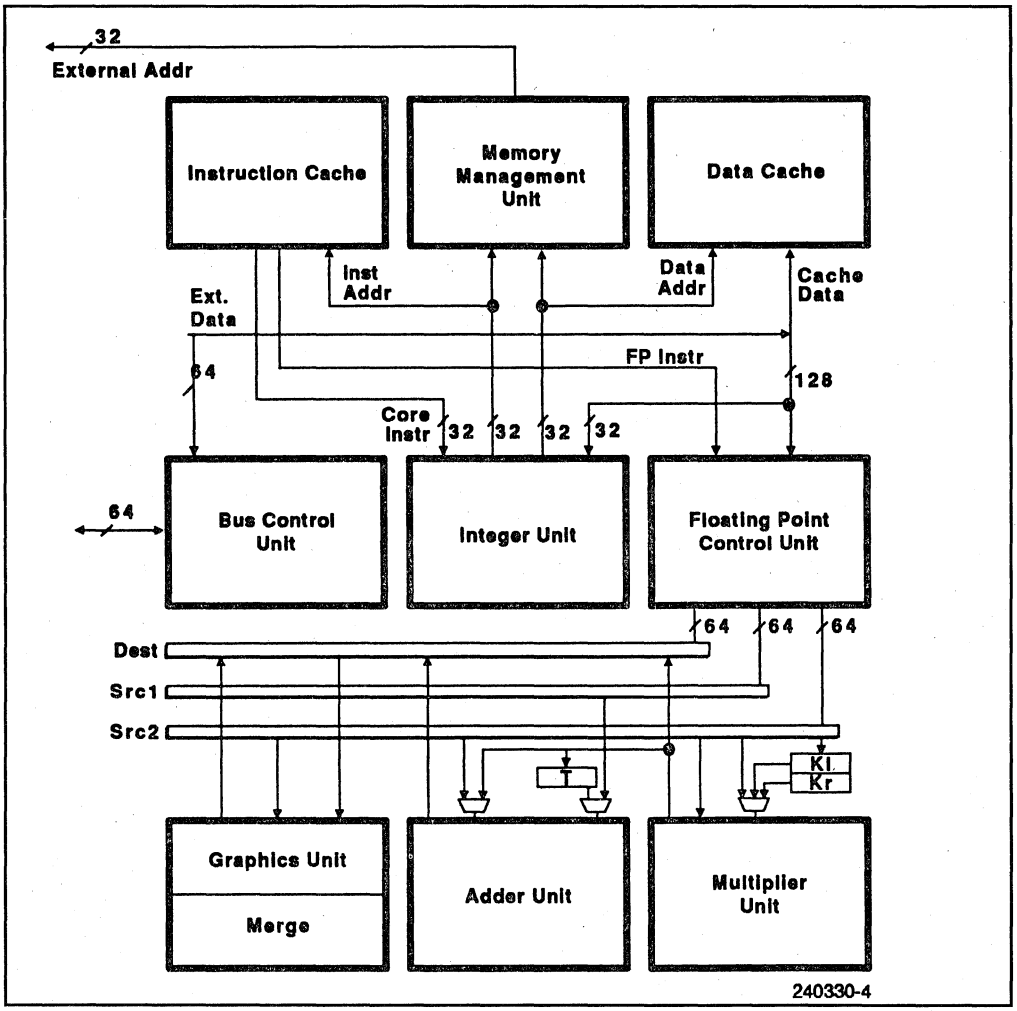


Figure 2-1. Block Diagram of i860™ Microprocessor

reads a value of zero. Writes to **r0** do not store a value in **r0. r0** works in combination with a number of instructions to modify and extend their function. For example, to check if a register contains a zero, the **or** instruction can be used with the other source operand and destination as **r0**. The register bank remains unmodified as a result, and the condition code CC indicates if the given register contains a zero.

The processor status register (**PSR**) and extended processor status register (**EPSR**) are 32-bit read/write registers which contain various information on the status of the current process. They provide information such as condition code bit status, loop condition code, interrupt control and status, trap flags, data breakpoint control, pixel information, processor identification, etc. Refer to the *i860™ 64-Bit Microprocessor Programmer's Reference Manual*, order number 240329, for more information.

The extended processor status register (**epsr**) is a 32-bit read/write register which contains additional state information beyond what is contained in the **psr**. This information includes the processor type (value of one for the i860 microprocessor), step number to distinguish among different revisions, data cache size field, and five flags: the overflow flag, big-endian mode bit, page table bit mode, write-protect mode bit, and the interlock bit. Refer to the *i860™ 64-Bit Microprocessor Programmer's Reference Manual* for more information.

The data breakpoint register (**db**) contains a breakpoint address. It is used to generate traps when loads or stores are made from or to this address, and is thus useful for debugging. Refer to the *i860™ 64-Bit Microprocessor Programmer's Reference Manual* for details.

The core execution unit contains logic for handling exceptions and external interrupts. When an exception condition or an external interrupt occurs, the processor transfers control to the trap handler. Again, refer to *i860™ 64-Bit Microprocessor Programmer's Reference Manual* for details.

2.1.2 Instruction Execution

Instructions are fetched into the core execution unit from the instruction cache. If this address location is not in the cache (a cache miss), the instruction is fed to the core execution unit from the external memory, while the corresponding Instruction Cache block is simultaneously filled.

2.1.2.1 RISC OPTIMIZATIONS

The core execution unit is designed according to RISC principles, as explained in Chapter 1. It uses a pipelined organization that maximizes performance. The instructions are made purposefully simple using a load/store architecture. Emphasis is placed on minimizing circuit delays and economizing chip space in order to include the other processing units that are essential to overall high performance – the floating-point unit, graphics unit, paging unit, caches, and register banks. Pipelining is coupled with register bypassing, scoreboarding, and delayed branching to further enhance performance. Some integer operations are performed by the floating-point unit. Integer multiply and divide are implemented via a code sequence which use floating-point instructions.

Execution pipelining is transparent for arithmetic, logic and shift instructions. Core instructions appear to operate in one clock cycle with the destination register already loaded by the time the next instruction begins executing. However, this is not actually the case. Due to the delay required in storing to and reading from a register, the processor detects if the last instruction's destination is used as an operand in the current instruction. If it is, the operand is returned to the ALU at the same time the register file is updated. This technique, known as *register bypassing*, is invisible to the programmer.

Unlike arithmetic and logic operations, load operations require a minimum of two clock cycles to provide a valid result for the destination register. Because load instructions require a minimum of two cycles, the integer core uses *scoreboarding* to detect if the register operand of the current instruction is the destination of a preceding load.

The use of register bypassing and scoreboarding allow load and store instructions to be executed at an effective rate of 1 instruction per clock cycle, assuming the data and instructions are found in their respective caches. When a cache miss occurs, the hardware will automatically resolve potential problems by freezing execution if the data is needed.

Branch instructions can also have the effect of locking the pipeline for one or more clock cycles. To avoid this waste, the i860 microprocessor uses *delayed branching*. The branch is delayed in the sense that the i860 microprocessor executes one additional instruction following the control-transfer instruction before actually transferring control. During the time used to execute the additional instruction, the i860 microprocessor refills the instruction pipeline by fetching instructions from the new instruction address. This avoids breaks in the instruction execution pipeline.

By using the above techniques, it is possible to execute core unit instructions at the rate of one per clock quite consistently, thus providing a rate of 40 MIPs (40 MHz clock) of native integer operation performance.

2.1.2.2 DUAL-INSTRUCTION MODE

The i860 microprocessor provides a form of parallelism in the ability to execute a core instruction and a floating-point instruction simultaneously. This parallel instruction execution is referred to as dual-instruction mode. When executing in this mode, the instruction sequence consists of 64-bit aligned instruction pairs with a floating-point instruction in the lower 32 bits and a core instruction in the upper 32 bits.

Enabling and disabling dual- and single-instruction mode is controlled by software. The **d.fp-op** in Figure 2-2 indicates the instructions responsible for enabling the dual-instruction mode. As shown in the figure, there is a one-instruction delay between the instruction that does the enabling or disabling and the instruction which performs the operation.

Note that when a 64-bit dual-instruction pair directly follows a delayed branch instruction in dual-instruction mode, both 32-bit instructions are executed.

Further details regarding the use of dual-mode instructions are provided in the *i860™ 64-Bit Microprocessor Programmer's Reference Manual*, order number 240329.

2.2 FLOATING-POINT UNIT

In addition to the core unit, which executes integer instructions, is the floating-point unit, which processes floating-point instructions. The pipelined floating-point unit, along with the on-chip cache, enables the 40 MHz i860 microprocessor to achieve a peak execution rate of up to 80 MFLOPs for single-precision and up to 60 MFLOPs for double-precision operations.

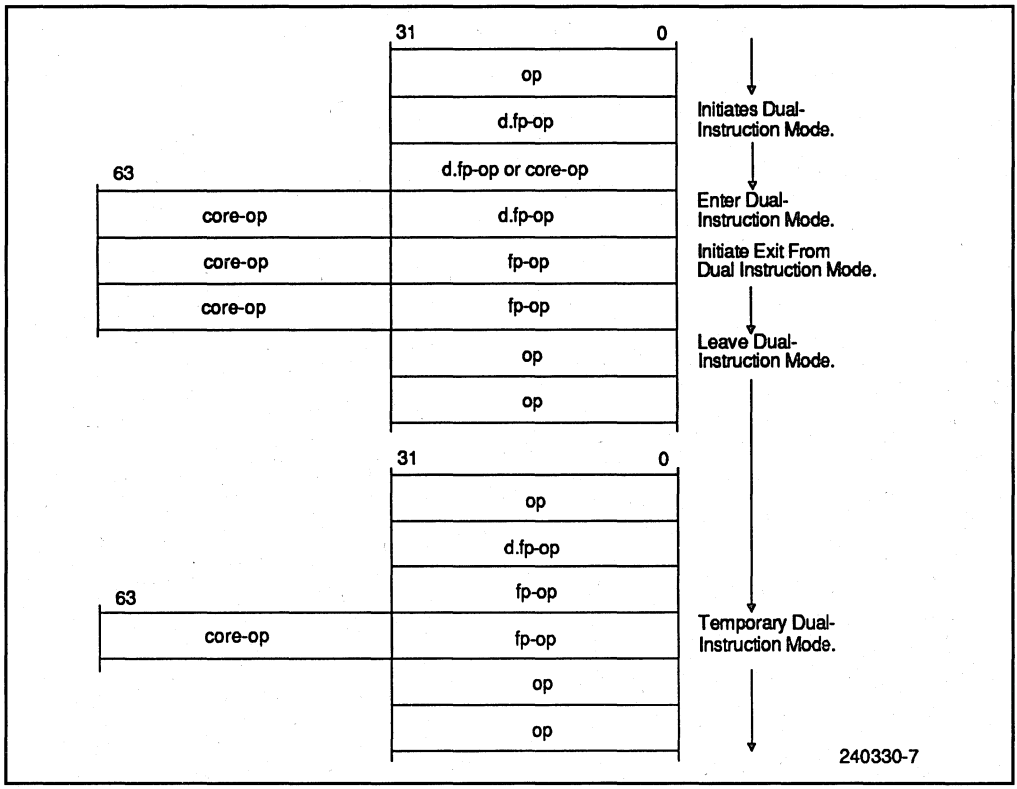


Figure 2-2. Dual-Instruction Mode

The floating-point unit consists of the floating-point register bank, floating-point adder, floating-point multiplier, floating-point status register and floating-point control unit. The floating-point unit has two types of operations: scalar and pipelined. Scalar operations are like those used in most computers. Pipelined operations deliver the highest performance but require more programming expertise. Intel offers a library of common math functions already implemented using pipelined code.

Floating-point data types, floating-point instructions, and exception handling all support the IEEE standard for binary floating-point arithmetic (ANSI/IEEE Std 754-1985) for both single- and double-precision data types. The floating-point status register holds information about the result of the operation. A complete set of traps includes tests for invalid source operands such as NaN (Not a Number), denormalized numbers, and infinities, as well as tests for errors in the result, such as overflow and underflow. The cause of the traps can be determined by examining the value in the floating-point status register. The floating-point traps permit implementation of the IEEE Standard in a very efficient manner. For more information, see the *i860™ 64-Bit Microprocessor Programmer's Reference Manual.*

Due to the low-level instruction set philosophy of the i860 microprocessor architecture, high-level functions defined by the Standard, such as square root, sine, and cosine are

not implemented directly by the hardware. The facilities of the floating-point unit, however, allow for a very efficient implementation of these functions that actually outperforms dedicated floating-point coprocessors.

Intel offers an IEEE trap handler program, as well as a software library, that provides i860 microprocessor programs with the full set of functions supported by the IEEE standard.

2.2.1 Floating-Point Register Bank

The floating-point unit is provided with its own register bank. It contains 32 floating-point registers, each 32-bits wide, labeled **f0** through **f31**. The registers can also be accessed in pairs for 64-bit double-precision values or 64-bit integer values. For this purpose, only even registers are used, e.g. **f2**, **f4**, etc. Load and store instructions also support the transfer of 128-bits worth of data (e.g. two double-precision operands). The registers are used in groups of four for this purpose, e.g. **f4**, **f8**, **f12**, etc. Registers **f0** and **f1** are special in that, when read, they always provide a value of zero, and writing into them has no effect. These registers modify and extend the function of the floating-point instructions. Two null registers are required in order to provide a zero operand and a null destination when using double-precision operations.

The floating-point register bank (refer to Figure 2-1) allows multiple operations to occur in parallel. It contains two read ports, one write port, and two bidirectional ports. All these ports are 64-bits wide and can be used concurrently.

The two 64-bit source operands provided by the floating-point registers are used as data input to the floating-point multiplier unit (FPMU), the floating-point adder unit (FPAU) or the graphics unit. A 64-bit input port to the floating-point registers transfers the result of the operations. The 64-bit integer instructions and graphics instructions also use this register bank for their source and destination operands.

Two 64-bit bidirectional ports between the data cache and the floating-point register bank allow transfers of up to 128 bits. A 64-bit bus can connect either of these two buses to the data bus on the bus cache control unit. This bus allows 64-bit transfers to and from external memory. The transfers are performed by the various floating-point load and store instructions. These transfers are controlled by the core unit and can occur in parallel with the floating-point instructions, as explained in Section 2.1.2.2, "Dual-Instruction Mode".

2.2.2 Pipelined and Scalar Operations

The floating-point unit uses parallelism to increase the rate of operations performed by the unit. One type of parallelism used in the floating-point unit is known as "pipelining". A pipelined architecture treats each operation as a series of more primitive operations called stages. These can be executed in parallel. Consider the floating-point adder unit as an example. Let **a** represent the operation of the adder, and let the stages be represented by A[1], A[2], and A[3]. The stages are designed such that the A[i+1] stage for one add instruction can execute in parallel with the A[i] stage for the next add instruction. Since each A[i] stage can perform its task in a single clock, and three instructions can be in executing in parallel, one add operation per clock is achieved.

Pipelining within the floating-point multiplier unit can be described similarly, except that it requires two clocks per stage on double-precision operations. The resulting status bits in the **fsr** reflect the result of the last completed operation within the pipeline. Pipelined instruction execution is shown in Figure 2-3.

The functions performed by each stage of the pipeline are not documented. Programs should not rely on specific actions, only that the pipeline is of fixed length.

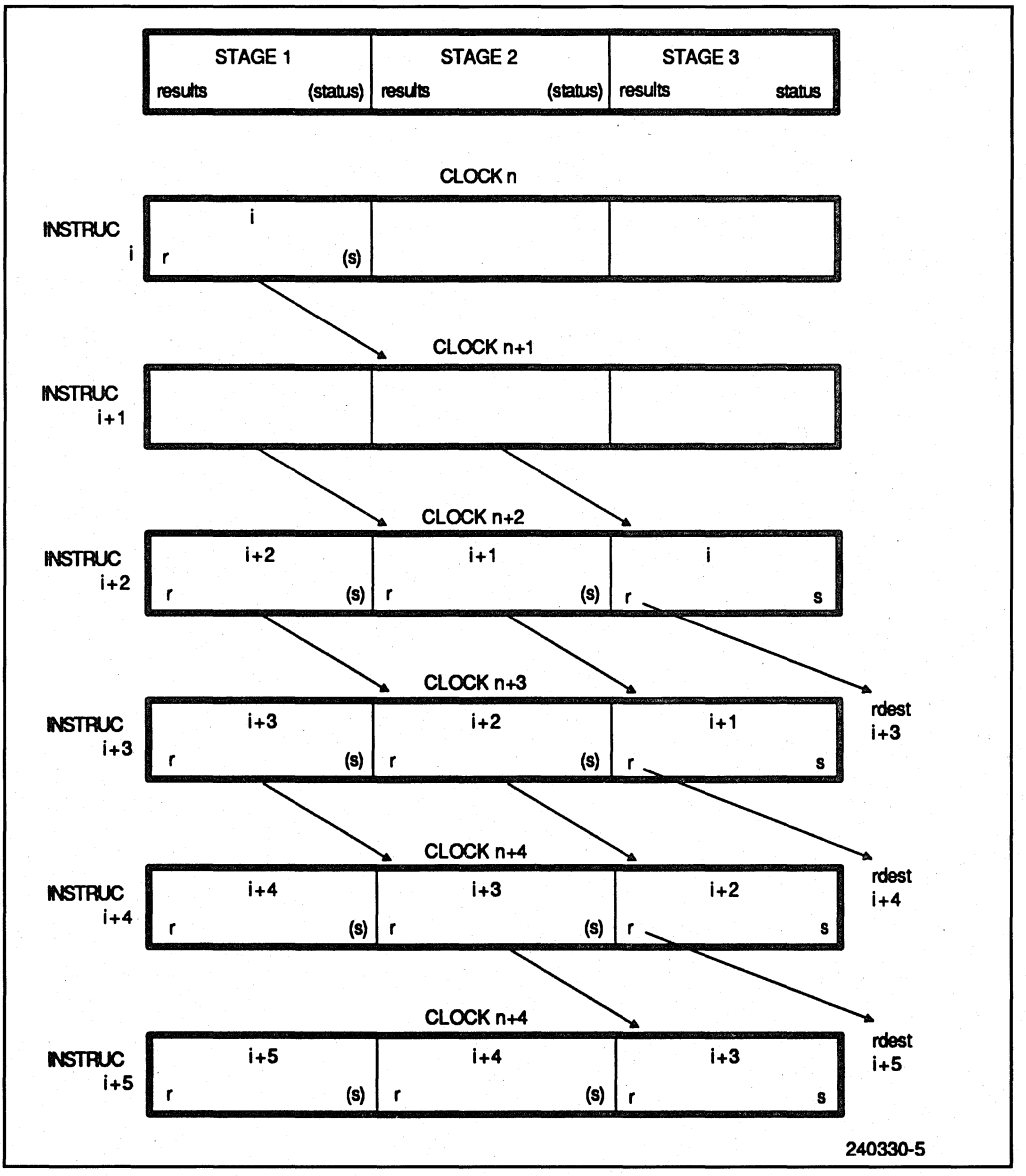


Figure 2-3. Pipelined Instruction Execution

In addition to pipelined execution, the i860 microprocessor can also execute floating-point operations in scalar mode. In this mode, the floating-point unit does not initiate a new operation until the previous floating-point operation is completed, so that the scalar operation passes through all the stages of its pipeline before a new operation is started. Scalar mode is used when the next operation depends upon the result of the previous floating-point operations, or when the compiler or assembly language programmer wishes to avoid the added complexity of pipelining. Integer operations can be performed in parallel to scalar floating-point operations.

2.2.3 Floating-Point Adder Unit

The floating-point adder unit of the i860 microprocessor supports both double- and single-precision IEEE 754 format and operates in two modes: scalar mode and pipelined mode. In scalar mode, three clocks are required to complete an add, subtract or compare operation. In pipelined mode, one result per clock for either a single- or double-precision operation is obtained.

The adder unit supports the following precision combinations between inputs and results: single to single, double to double, and single to double. For this reason, the adder is also used to perform data precision conversions. Some of the instructions executed exclusively by the adder unit are:

- Floating-point add (**fadd**)
- Floating-point subtract (**fsub**)
- Pipelined floating-point comparisons: (**pfgt.p**, **pfeq.p**)

2.2.4 Floating-Point Multiplier Unit

The floating-point multiplier unit performs floating-point multiplication in accordance with the IEEE standard. It is organized as a three-stage pipeline. In pipelined mode, the multiplication throughput is one single-precision instruction per clock and one double-precision instruction each two clocks.

The multiplier unit also supports a reciprocal instruction which is used to implement division and square-root operations by means of an iterative process. A small macro (or function) can be developed based on these instructions to perform the full division or square root.

Certain instructions allow the multiplier unit to operate in parallel with the adder unit in a variety of flexible ways, thereby doubling the number of operations per clock.

2.2.5 Dual Operation Feature

Dual Operation is a special feature of the i860 microprocessor which allows the floating-point adder and multiplier unit to work in parallel, thus doubling the number of floating-point operations performed. Both add-and-multiply and subtract-and-multiply operations are supported.

The instruction formats for add-and-multiply and subtract-and-multiply allow specification of only two source operands and one destination. However, when operating the adder and multiplier in parallel, two pairs of operands and two destinations are needed for the general case. To overcome this limitation, the adder and multiplier can be configured in a variety of ways that are especially suitable for problems such as:

- matrix manipulation (e.g., solving linear equations)
- series calculations (e.g., sine function calculation)
- signal processing applications (e.g., Fast Fourier Transform)
- graphics (e.g., coordinate transformations)

For this purpose, three special registers are used—KR, KI and T. Both KR and KI can be used to hold constants or temporary values. These values can be loaded when used as operand inputs to the multiplier, and can later supply the value, without the need for an explicit instruction operand. T can act as a transfer register to hold the value of the result of a multiplication, which can be passed on as an operand to the adder on a later instruction.

The data paths available are shown in Figure 2-4. Possible configurations can be selected as follows:

- Operand 1 of the multiplier can be KR, KI, or *scrl*.
- Operand 2 of the multiplier can be *scr2* or the last stage of the adder pipeline.
- Operand 1 of the adder can be *scrl*, the T-register, or the last stage result of the adder pipeline.
- Operand 2 of the adder can be *scr2*, the last-stage result of the multiplier pipeline, or the last-stage result of the adder pipeline.

In addition to the selection of operands, the instruction can choose whether to load KI, KR, or T as part of its operation. The possible operand data path selections and loading options allow a large number of possible combinations. Many of these combinations are functionally redundant or of no interest. Each instruction format for add-and-multiply and subtract-and-multiply supports 16 different instructions, and each of these instructions provides a different configuration of operand data path and KI, KR, or T loading selection. The configurations have been specially selected to streamline the implementation of the applications previously mentioned. Refer to the *i860™ 64-Bit Microprocessor Programmer's Reference Manual* for further details.

2.2.6 Floating-Point Computation Throughput

The combination of the dual-instruction mode feature with pipelined dual operation allows the i860 microprocessor to achieve a sustained 80 MFLOPs in single-precision and 60 MFLOPs in double-precision for inner loops of common computations. Assuming the code is in the cache, no visible memory cycles are needed to fetch the instructions.

The dual-instruction mode allows the loading and storing of operands and the updating of array indices and loop control information to be performed in parallel with floating-point execution. A load or a store (core unit instruction) can transfer up to 4 single-precision operands or 2 double-precision operands, assuming these operands are adjacent to each other in memory within some data array. Loads and stores take one

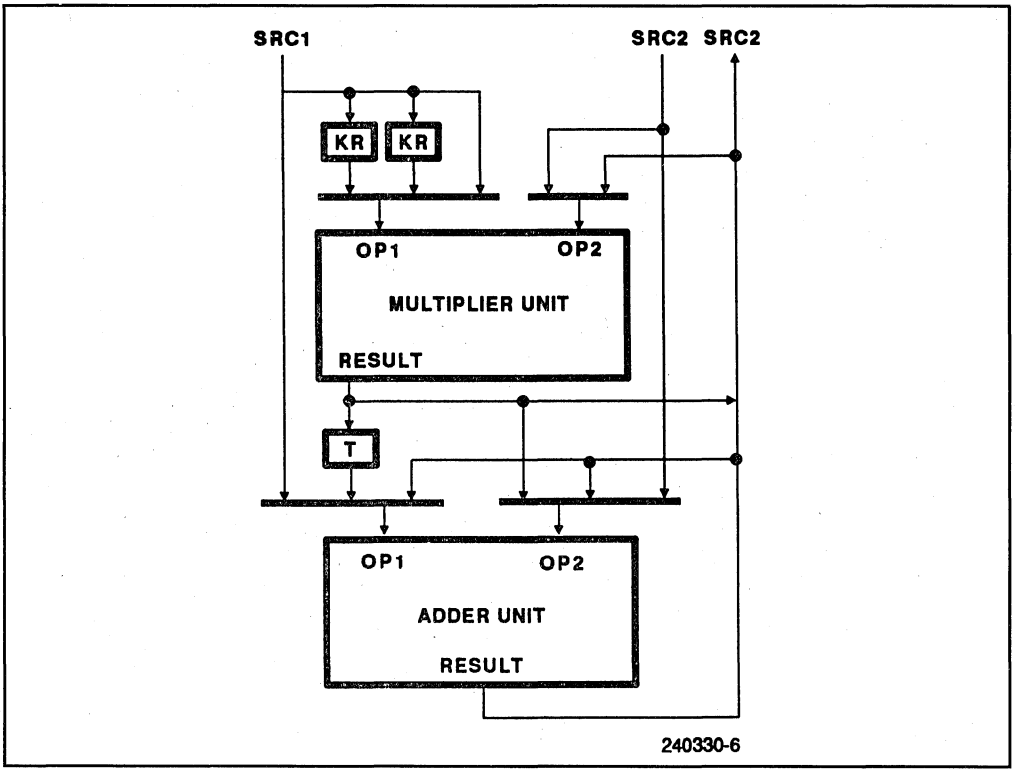


Figure 2-4. Dual-operation Data Paths

clock cycle if the data is in the cache, and two clock cycles if fetched from external memory. If the operands are from memory, instructions can continue to be executed in the pipeline as long as they don't access the registers being loaded. Thus, indexing, loop-control, and operand loading can typically take place in parallel with the floating-point computation, maintaining the sustained rate.

In pipelined mode, two single-precision floating-point operations can be executed per clock cycle resulting in a rate of (2 operations/clock) × (40 MHz/sec) = 80 MFLOPs. For double-precision, addition requires one clock cycle, while multiplication requires two. For algorithms that require two floating-point additions and one multiplication for each iteration, two adds and one multiply can be done in parallel in two clock cycles. This results in three operations in two clock cycles, or (3-operations/2-cycles) × (40 MHz) = 60 MFLOPs. Algorithms requiring a double-precision multiply and add for every iteration execute at two operations per two clock cycles (due to the multiply two-clock rate), resulting in an execution rate of 40 MFLOPs.

2.3 PAGING UNIT

The paging unit provides the i860 microprocessor with the capability of supporting an efficient implementation of demand-paged virtual memory. Demand-paged virtual memory allows programs to use a larger, virtual memory space, which is actually supported by

a smaller real (or physical) memory space. The paging unit, in combination with the appropriate memory management software, automatically allocates physical pages of memory to virtual page addresses as they are needed (on demand). Typically, when all of physical memory is used up, physical memory is swapped out to disk, and pages are reallocated.

The paging unit provides the ability to translate virtual addresses used by the processor to physical addresses that correspond to locations in the external memory. It also provides page-level protection based on access rights, as well as two levels of privilege—user and supervisor.

Address translation and memory protection are optional. They are enabled by the address translation enable (ATE) bit in the directory base register. If this bit is not set, the physical address is the same as the virtual address, and no translation or access-rights checking is performed. The ATE bit is cleared upon reset. The i860 microprocessor paging unit functions the same and uses the same page table entry formats as the Intel 386™ and i486™ microprocessors.

2.3.1 Paging Algorithm

A virtual address is mapped to a physical address according to a set of tables called page tables. The address is divided into a page address and an offset. A page is a collection of data that occupies the space of a page frame in main memory, or some location in secondary storage when there is insufficient space in main memory. A page frame consists of 4K bytes of contiguous physical memory starting on a 4K-byte boundary. The upper 20 bits of the 32-bit address of a page frame is referred to as the page address. In the i860 microprocessor, both virtual and physical addresses are 32 bits wide. They both consist of a 20-bit page address and a 12-bit offset. The concatenation of the two provides a complete 32-bit byte address.

The address translation algorithm uses two levels of page tables. The two levels are referred to as the page directory and the page tables. Both levels of page tables are a page (4096 bytes) in size, consisting of 1024 32-bit entries.

The directory base register, **dirbase**, contains a 20-bit field which points to the page directory. Only one page directory table is active at any given time. The entries of the page directory contain the physical addresses of all the page tables used for the mapping process, or contain entries indicating that the given page tables (corresponding to virtual segments of address space) are not present in physical memory. The page tables themselves contain physical page addresses for all the valid virtual pages, or entries indicating that the given virtual memory address is not present in physical memory.

The algorithm mapping virtual memory to physical memory is fully implemented by the hardware and is depicted in Figure 2-5. The most-significant 10 bits of the virtual address are used as an index into the page directory, which selects a specific page table. The next 10 bits are used as an index into the selected page table, selecting a page frame address. The last 12 bits act as an offset into the page frame address, building up a full 32-bit physical byte address. This completes the virtual to physical address conversion.

If during a memory transfer, the page directory or page table indicates that a selected page table or page frame is not present (by means of a zero in the Present bit of the table entry), a trap occurs, allowing the software to validate the page by reading it from secondary storage. The page table entries also provide the write, user, cache disable, accessed and dirty bits, as well as 3 user-definable bits. These bits along with the write protect bit in the extended processor status register are used to provide page-level protection rights, page cacheability information, and information needed to implement an efficient replacement algorithm for swapping out page frames when main memory is full. More details are provided in the *i860™ 64-Bit Microprocessor Programmer's Reference Manual*, order number 240329.

To avoid accessing the page directory and page table for every address translation, the i860 microprocessor implements an on-chip translation look-aside buffer (TLB), which is a cache that directly translates a virtual page address to a physical address, and provides the additional bits from the page tables needed to provide protection and information for replacement algorithms. TLB translation requires one clock cycle and is typically invisible because of the processor's pipelining.

The TLB is implemented as a 4-way, set-associative cache, mapping a total of 64 page table entries. Because each page table entry maps 4 Kbytes of address space, a total of 4K × 64 or 256 Kbytes of memory are mapped at any one time by the TLB. When there

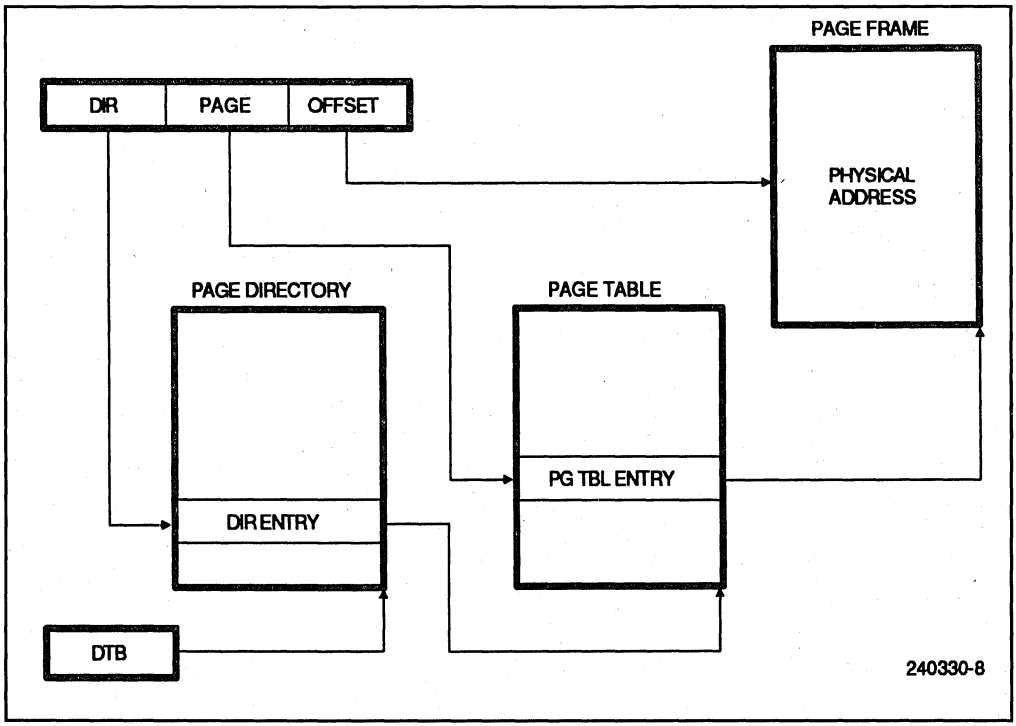


Figure 2-5. Paging Algorithm Implementation

is a miss in the TLB and the page tables in memory are used, an entry in the TLB is automatically replaced by the new mapping.

2.4 ON-CHIP CACHES AND BUS CONTROL

The i860 microprocessor contains both an instruction and data cache. Being integrated so close to the processor, and having been carefully designed for minimum delay, cache storage can operate much faster than external memory. In addition, having both caches on-chip allows them to operate in parallel. Thus, the processor can simultaneously read instructions from the instruction cache, read or write data to and from the data cache, as well as translate virtual addresses through the TLB. The caches also provide wide data paths—the instruction cache is 64-bits wide, and the data cache is 128-bits wide. Also, on-chip caches reduce the need for external caches, which reduces the total system cost and makes available valuable PC-board real estate.

Both the data and instruction caches are virtually addressed. This not only provides for faster operation, but allows the TLB to perform its virtual address to physical address translation in parallel with the operation of the cache. If one of the caches determines that it does not have the contents of the required virtual address (a cache miss), the TLB will be ready with a physical address with which to begin an external memory cycle.

Both the instruction and data caches are implemented as a 2-way set associative memory that maps a virtual address to a 32-byte block of data. These blocks correspond to 32 consecutive bytes loaded from an address having zero for the least-significant 5 bits. When transferring data to or from the cache, the processor uses the desired set of bytes from this 32-byte group. Allocation and replacement for both caches is always performed using blocks of 32 bytes.

The i860 microprocessor uses a wraparound technique which makes the process of filling a cache block more efficient. Since a block consists of four 64-bit (8 bytes) entries, the processor will first read the 64-bit entry that contains the data item, instruction, or instruction pair that is needed by the processor. The processor continues processing while the entry read is simultaneously stored in the cache. Next, the processor sequentially reads the remaining three, 64-bit entries of the block and stores them in the cache. Since the first entry read may lie in the middle of the block, the processor wraps around to read the first 64-bit entry of the block after the last one is read. In this manner, the cache block is loaded and the processor gets its data as quickly as possible.

The use of the caches when accessing memory can be bypassed by means of the CD (Cache Disable) bit in the page tables or the KEN# (Cache Enable) pin, as explained in Chapter 3. This is required for special cases such as I/O references or shared data in a multiprocessor system.

2.4.1 Instruction Cache Unit

The instruction cache size is 4 Kbytes. With a 64-bit wide data path, the cache can provide two 32-bit instructions in each fetch cycle: two core instructions, or one core instruction, and one floating-point instruction. Thus, the transfer rate of the cache is 64 bits/clock (320MB/sec at 40 MHz).

The instruction cache is intended to be used to map read-only memory (i.e., it does not support self-modifying code). System programs that modify code should invalidate the instruction cache via the ITI bit in the directory base register.

2.4.2 Data Cache Unit

The data cache size is 8 Kbytes. Using a 128-bit wide bus for reading data, it can transfer up to 128 bits/clock. Thus with a 40 MHz clock, 640 Mbytes/sec can be transferred. Unlike the instruction cache, the data cache supports both read and write operations (corresponding to load and store instructions).

The data cache with its 128-bit internal bus can supply up to two 64-bit operands to the floating-point unit per cycle. Alternatively, it can supply one 32-bit operand per load cycle, or a 16- or 8-bit, right-aligned, signed-extended value to the core unit.

The data read must be aligned in memory according to its size. The table below shows the restriction on the addresses for different size data items:

Data Size	Number of least-significant bits in byte address that are 0
16-bit value	1
32-bit value	2
64-bit value	3
128-bit value	4

The data cache can be used as a large set of floating-point registers for vector operations. The ICS field of the extended processor status register allows a program to determine the cache size and use it appropriately.

2.4.2.1 WRITE OPERATIONS AND THE DATA CACHE

Writes to memory locations not present in the cache are sent directly to the memory write buffers and do not affect the cache. A write operation to a location that is already in the cache is written to the cache, but is not immediately written to memory. In this scheme, known as *write-back*, the blocks that have been written to the cache but not to memory are marked as "dirty". When the replacement algorithm chooses to replace a block containing a dirty block, or when a cache block is flushed, these dirty blocks are written to memory.

The *write-back* scheme provides better performance than the *write-through* approach, in which data written to the cache is immediately written to memory. This is because accesses and stores to variables or indices that are in the data cache require no external memory cycles and hence reduce bus traffic.

When the data cache writes a block to memory, it uses two 128-bit wide write buffers. These buffers delay the actual memory writes until an opportune time, if possible, as explained in Section 2.4.6.

2.4.3 Bypassing Instruction and Data Caches

There are two pins on the i860 microprocessor that relate to cache enabling during any external memory cycle. They are the Cache Enable input pin (KEN#) and the Page Table Bit (PTB) output pin. They are used for controlling access to shared memory, I/O buffers, and memory-mapped I/O devices.

When the i860 microprocessor detects that the KEN# input is not asserted during a bus cycle, the processor inhibits use of the data cache and instruction cache for this cycle.

When paging is enabled, the CD (cache disable) bit of the secondary page table used during a memory transfer determines whether or not to enable the use of the instruction or data caches. A similar bit, WT (write through) is also available that disables data caching. The value of the OR of these bits is reflected on the PTB output pin, depending on the PBM bit in the extended processor status register. When paging is disabled, the PTB pin remains not asserted.

KEN# is internally NORed with CD OR WT bits to determine whether or not to enable use of the cache, as shown in Table 2-l.

Table 2-1. Cacheability Based on CD, WT and KEN#

CD OR WT	KEN#	Meaning
0	0	Cacheable access
0	1	Noncacheable access
1	0	Noncacheable page
1	1	Noncacheable page

2.4.4 Flushing Instruction Cache, Data Cache, and TLB

Setting the ITI (Instruction TLB Invalidate) bit in the **dirbase** register, invalidates the contents of the instruction cache, as well as the Translation Lookaside Buffer.

The data cache is flushed by software using the cache flush instruction. This instruction flushes one cache block at a time. A loop of code is executed to clear the entire data cache. For a copy of this code, see the *i860™ 64-Bit Microprocessor Programmer's Reference Manual.*

2.4.5 Bus and Cache Control Unit

The bus and cache control unit interfaces to the external bus, performing instruction and data accesses for the execution core unit. The control unit transfers data to and from the external code memory, and controls TLB translation, including normal translation, miss replacement and fault processing. It receives cycle requests and specifications from the execution core unit. It performs instruction or data cache accesses and handles data or instruction cache miss processing (cache block replacement). Its pipelined structure supports up to three outstanding bus cycles. The three-level bus cycle pipelining is explained in Chapter 3.

The bus and cache control unit can fetch one, 64-bit instruction from the instruction cache and 128-bits worth of data from the data cache on every clock cycle, as long as the accessed data resides in the cache.

The bus control unit also performs the physical address comparison for the generation of the Next Near (NENE#) signal. The NENE# signal is asserted by the i860 microprocessor if the currently issued address falls on the same DRAM page as the previously issued address. The NENE# pin allows the external memory system to take advantage of the static column and page-mode DRAMs. The size of the DRAM page is programmable by three bits in the DIRBASE register.

The i860 microprocessor can operate in two instruction fetch modes: normal or CS8 (code size 8) mode. When the CS8 bit in the DIRBASE register is set, external memory cycles are processed as 8-bit cycles. When this bit is clear, instruction cache misses are processed as 64-bit bus cycles. This bit cannot be set by software. To enter the CS8 mode, the INT signal is asserted prior to the falling edge of the RESET signal. This allows the instruction bytes to be fetched on the eight least-significant bits of the external data bus. Data may be transferred as 64, 32, 16, or 8 bits values using all 64 bits of the data bus.

The CS8 mode allows the i860 microprocessor processor to be bootstrapped from an 8-bit EPROM. In the CS8 mode, the signals BE2, BE1 and BE0 are redefined to correspond to the three least-significant bits of the address so that a complete byte address is available (i.e., 32 address pins can be used).

Once the bootstrap code has been loaded into the 64-bit memory, a 64-bit fetch can be initiated. This is accomplished by clearing the CS8 bit in the **dirbase** register via software (one time only). Once this bit is disabled, it can not be enabled until a new hardware reset occurs.

2.4.6 Write Buffers

The bus and cache control unit also supports the use of two 128-bit write buffers. These buffers are designed to delay any write operations to memory until memory is not being used (i.e., instructions and data are being read from the caches). This optimizing delay is not always possible, because the memory operation in question may itself be read. The bus control logic forces memory write operations to insure proper functionality.

The two 128-bit write buffers can operate on independent memory cycles. When write operations of 128 bits are performed, each write buffer is written in two memory cycles. When writes are made that are smaller than 128 bits (64, 32, 16, or 8 bits), the write buffer is written in a single cycle. Proper alignment and the selection of the correct byte enables is made for cycles smaller than 64 bits.

2.5 GRAPHICS UNIT

The graphics unit executes instructions designed to support high-performance 3-D graphics applications. Support for packed pixels of 8-, 16-, and 32-bit data are supported. The precise pixel formats are given in the *i860™ 64-Bit Microprocessor Programmer's Reference Manual*, order number 240329.

The graphics unit executes simple but powerful instructions that can be applied to the following graphics functions:

- Hidden surface elimination
- Distance interpolation
- 3-D shading using intensity interpolation

Based on these instructions, the i860 microprocessor processor can perform real-time, shaded graphics without the need for an external graphics processor.

The instructions operate properly for the various pixel formats. With the 8-bit format the operations only affect the intensity. For 16- and 32-bit color pixels, the operations affect the isolated color vectors, e.g., red, green, or blue. Operations are performed on 64-bit entities, which can contain the values of multiple pixels in parallel. A special Pixel Store instruction implemented by the core unit can work in parallel with the graphics instructions.

The interpolation operations of the processor support graphics applications in which a set of points on the surface of a solid object is represented by polygons. The distance and color intensities of the vertices of the polygons are known, but the distance and intensities of other points must be calculated by interpolation between these points. Graphics instructions, just as floating-point instructions, can be used in dual-instruction mode to achieve greater computation rates.

Local Bus Interface 3

CHAPTER 3
LOCAL BUS INTERFACE

The local bus is designed to provide high data throughput among the processor, memory and I/O subsystems. It provides a flexible interface that is suitable for a wide variety of system environments.

This chapter describes the local bus interface, its basic function, operation, and timing for related signals.

3.1 i860™ MICROPROCESSOR EXTERNAL INTERFACE AND BUS SIGNALS

The external interface of the i860™ microprocessor consists of a 64-bit data bus, 29-bit address bus, eight-bit byte-enable control bus, 19 status and control signals, and 48 power and ground pins. This section provides an overview of the services provided by the external interface of the i860 microprocessor.

3.1.1 i860™ Microprocessor Buses

The i860 microprocessor communicates with external memory and I/O through a synchronous bus interface that includes a separate data and address bus as follows:

D63–D0 — These 64 pins make up the bidirectional data bus external interface. Either 8, 16, 32, or 64 bits of data can be transferred during a bus cycle.

A31–A3 — The address bus consists of 29 address pins which address one of 2^{29} 64-bit memory locations.

BE7#–BE0# — The byte-enable bus consists of eight pins that specify which bytes to access within a 64-bit location. These pins are used to enable writing in one, two, four or eight-bytes of the double-word involved in the current write cycle. Read operations should always return 64-bits of data. See Section 3.3.1.3 for details. BE2#, BE1#, BE0# are used as address bits A2, A1, A0 respectively while in CS8 mode. See Section 3.3.5 for details.

3.1.2 i860™ Microprocessor Output Signals

The i860 microprocessor output signals provide control and status information. The output signals are as follows:

ADS# — The i860 microprocessor asserts the address status signal to indicate the beginning of a bus cycle. It identifies the clock period during which it provides a valid address and the other signals required to perform a memory cycle.

W/R# The write/read# signal indicates whether the current cycle is a write (high state) to or read (low state) from the memory or I/O subsystem.

LOCK# The lock signal is generated by the processor to indicate locked cycles to external circuitry (Section 3.4.3 provides further explanation).

NENE# The next near signal tells the memory subsystem that a cycle is on the same DRAM row as a previous cycle. This allows the memory subsystem to use page mode or static-column mode features of DRAMs (Section 3.4.1 provides further explanation).

PTB The page table bit signal reflects either the value of the cache disable (CD) bit, or the write through (WT) bit of page table entry during the current cycle. The PBM (page-table bit mode) bit of the EPSR indicates which. If PBM is clear, PTB reflects CD; otherwise, it reflects WT (Section 3.5 provides further explanation).

HLDA The hold acknowledge signal indicates that the bus has been released (Section 3.4.2 provides further explanation).

BREQ The bus request signal is asserted when an internal bus request is pending. This signal is used to assist external bus arbitration. Its value is independent of the state of HOLD and HOLDA (Section 3.4.2 provides further explanation).

BREQ is also used as serial output for the boundary scan chain while in boundary scan mode (Section 3.7 provides further explanation).

3.1.3 i860™ Microprocessor Input Signals

Input signals control various i860 microprocessor actions:

CLK The clock input provides basic timing information for the processor to synchronize internal and external operations. All other signals are sampled relative to the rising edge of CLK. The internal operating frequency is the same as the clock frequency.

READY# The ready signal indicates to the processor that a bus cycle is finished. For read cycles, the READY# signal indicates that data being read is valid and that the processor can latch the contents of the data bus. For write cycles, it indicates that the data being output to the data bus is being latched by the memory subsystem and is no longer needed to finish the bus cycle. READY# must be synchronous to CLK; it is sampled on every clock after the clock which follows the sampling of ADS#.

NA# The next address signal allows external data transfers to request pipelining. The signal indicates to the processor that the memory or I/O subsystem is ready to receive a new address and begin a pipelined cycle. NA# is sampled during the second clock after ADS# (Section 3.3.4 provides further explanation).

INT/CS8 — The interrupt and code size signal serves two functions. When the RESET signal is asserted, the CS8 signal can be used to set the code size eight mode to indicate whether the bus performs instruction fetches on the low-order byte of the bus instead of the 64 bit wide bus. This feature allows booting from a single EPROM. Section 3.3.5 provides further details. At all other times, this pin serves as the INT signal and functions as the i860 microprocessor's maskable external interrupt (Section 3.6 provides further explanation). The state of the INT input is sampled on every clock.

KEN# — The cache enable signal enables updates to the processor's instruction and data caches. When paging is enabled, this signal works in combination with the WT, CD, and PTB bits of the current bus cycle. KEN# is sampled on every bus cycle (Section 3.5 provides details).

HOLD — The bus hold signal floats all output signals except HOLDA and BREQ and causes the processor to relinquish control of the bus. The HLDA signal indicates that the bus has been granted. Instruction execution continues unless required instructions and data cannot be read from the on-chip cache (Section 3.4.2. provides further explanation). The state of this pin is sampled every clock.

SHI — The boundary scan shift input signal is used to read boundary scan chain serial data when in boundary scan mode (Section 3.7 provides further explanation).

BSCN — The boundary scan enable signal enables boundary scan mode for board or component testing (Section 3.7 provides further explanation).

SCAN — The shift scan is used in conjunction with boundary scan mode to set normal mode (when SCAN is deasserted) or shift mode (when SCAN is asserted) (Section 3.7 provides further explanation).

CC1, CC0 — These pins are reserved by Intel and must be strapped low.

The i860 microprocessor bus interface pins are summarized in Table 3-1.

Table 3-1. Pin Summary

Pin Name	Function	Active State	Input/Output
Execution Control Pins			
CLK	CLocK		I
RESET	System reset	High	I
HOLD	Bus hold	High	I
HLDA	Bus hold acknowledge	High	O
BREQ	Bus request	High	O
INT/CS8	Interrupt, code-size	High	I

Table 3-1. Pin Summary (continued)

Pin Name	Function	Active State	Input/Output
Bus Interface Pins			
A31–A3	Address bus	High	O
BE7#–BEO#	Byte Enables	Low	O
D63–D0	Data bus	High	I/O
LOCK#	Bus Lock	Low	O
W/R#	Write/Read bus cycle	Hi/Low	O
NENE#	NExt NEar	Low	O
NA#	Next Address request	Low	I
READY#	Transfer Acknowledge	Low	I
ADS#	ADdress Status	Low	O
Cache Interface Pins			
KEN #	Cache ENable	Low	I
PTB	Page Table Bit	High	O
Testability Pins			
SHI	Boundary Scan Shift Input	High	I
BSCN	Boundary Scan Enable	High	I
SCAN	Shift Scan Path	High	I
Intel-Reserved Configuration Pins			
CC1–CC0	Configuration	High	I
Power and Ground Pins			
V_{CC}	System power		
V_{SS}	System ground		

A # after a pin name indicates that the signal is active when at the low voltage level.

3.1.4 Power and Ground Pins

The i860 microprocessor has 24 ground pins and 24 power pins. The *i860™ 64-Bit Microprocessor Data Sheet* provides pin number assignments, detailed electrical characteristics, and decoupling requirements.

3.2 BUS CHARACTERISTICS

The fully-synchronous local bus provides 64-bit data transfers to and from memory or I/O devices. Minimum read and write cycles can be done in two clock cycles. The bus is capable of pipelining bus cycles two levels deep (three stages). I/O is memory mapped. An external address decoder can map address ranges to correspond with the I/O subsystem and the memory subsystem. Also, memory-mapped devices should drive KEN# high during reads to prevent data caching.

To simplify explanation, the term memory subsystem refers to the I/O subsystem and the memory subsystem.

3.3 BUS TRANSFER OPERATIONS

This section discusses all bus transfer operations including data alignment issues, pipelined and nonpipelined bus transfers, and 8-bit mode operation for bootstrapping.

3.3.1 64-bit Bus and Byte Alignment of Data

The i860 microprocessor performs external data transfers by means of its 64-bit data bus. This section discusses the details of how the bus control unit performs bus operations on data of various sizes.

3.3.1.1 MEMORY ADDRESSABILITY AND ALIGNMENT REQUIREMENTS

Memory is addressable down to each byte within a paged virtual address space of 2^{32} bytes. The i860 microprocessor instructions can operate on data of various sizes including bytes (8-bytes), half-words (16-bits), words (32-bits), double-words (64-bits) and quadwords (128-bits).

Data may be located anywhere within the byte-addressable space. However accesses to data not following the alignment requirements given below cause a trap. Load or store operations to unaligned data must be handled by a software routine and are very inefficient. The alignment requirements for data are as follows:

- 128-bit values are aligned on 16-byte boundaries when referenced in memory (the four least-significant address bits must be zero).
- 64-bit values are aligned on 8-byte boundaries when referenced in memory (i.e. the two least-significant address buts must be zero).
- 32-bit values are aligned on 4-byte boundaries when referenced in memory (i.e. the two least-significant address bits must be zero).
- 16-bit values are aligned on 2-byte boundaries when referenced in memory (the least-significant address bit must be zero).

Misaligned instructions are not allowed. Instruction alignment requirements are:

- All instructions are 32-bits long and must be aligned on 4-byte boundaries (i.e. the two least-significant address bits must be zero).
- Dual mode instruction pairs must be aligned on 8-byte boundaries with the floating-point instruction first.

3.3.1.2 DATA ALIGNMENT DURING READ OPERATIONS

The i860 microprocessor performs aligned read operations in the following manner. 64- and 128-bit transfers are handled as one and two 64-bit memory transfers, respectively.

8-, 16-, and 32-bit memory read operations are accomplished by first reading 64-bits of data and extracting the data bytes needed. The 64-bit data is typically stored in the cache in parallel with the extraction of the data. The data is extracted by shifting the 64-bit value so that the least-significant byte of the data item is aligned with bits D7–D0. For 8- and 16-bit integer data, the value is sign-extended to 32 bits, and the value is loaded into the appropriate register.

During read cycles, the byte enable signals BE0#–BE7# reflect the bytes on the data bus that are involved in the read operation. During read operations, however, the byte enable signals are not used to enable memory for specific bytes. Instead, all 8 bytes of the data bus must be read by the processor, if caching is enabled, to properly update the corresponding 64-bit entry in the cache. This type of operation is illustrated by the simplified circuit diagram in Figure 3-1.

3.3.1.3 DATA ALIGNMENT DURING WRITE OPERATIONS

Write operations require the reverse aligning process of a read operation. 64- and 128-bit write operations are handled as one and two 64-bit data transfers, respectively. 8-, 16- and 32-bit memory write operations are performed by properly aligning the data to be written onto the data bus and writing only on the bytes involved within the 64-bit word addressed.

The byte enable lines, BE0#–BE7# are used to determine which bytes of the addressed 64-bit word to write to.

For 8-bit, 16-bit and 32-bit data, the data bus outputs the register data shifted to the left by an appropriate number of bytes. The shift aligns the least-significant byte of the data with the least-significant byte of the destination in memory.

The properly-aligned data bytes in the data bus are written on the bytes of the 64-bit word addressed, as selected by the active BE7#–BE0# lines. The circuit shown in Figure 3-1 illustrates how the byte enable signals operate during a write operation.

The number of bytes by which to left shift the data and the set of byte enable signals that are activated is determined by the least three significant bits of the byte address, the size of the data and the endian mode used, as discussed in Section 3.3.1.4. The 64-bit word involved is selected by the upper 29 bits of the physical address.

3.3.1.4 LITTLE AND BIG ENDIAN MODES AND BUS OPERATION

The i860 microprocessor can store data in memory in one of two formats, little endian or big endian. In little endian, all multibyte data items are stored so that the least significant byte is at the smallest address of the bytes allocated for the data item. In big endian format all multibyte data items are stored so that the most significant byte is at the smallest address. The i860 microprocessor can operate in either mode.

In the i860 microprocessor, multiple-byte data values are normally stored in little endian format (with the least significant byte at the lowest memory address). The processor, however also provides the capability of operating in big endian mode (with the most significant byte is at the lowest address).

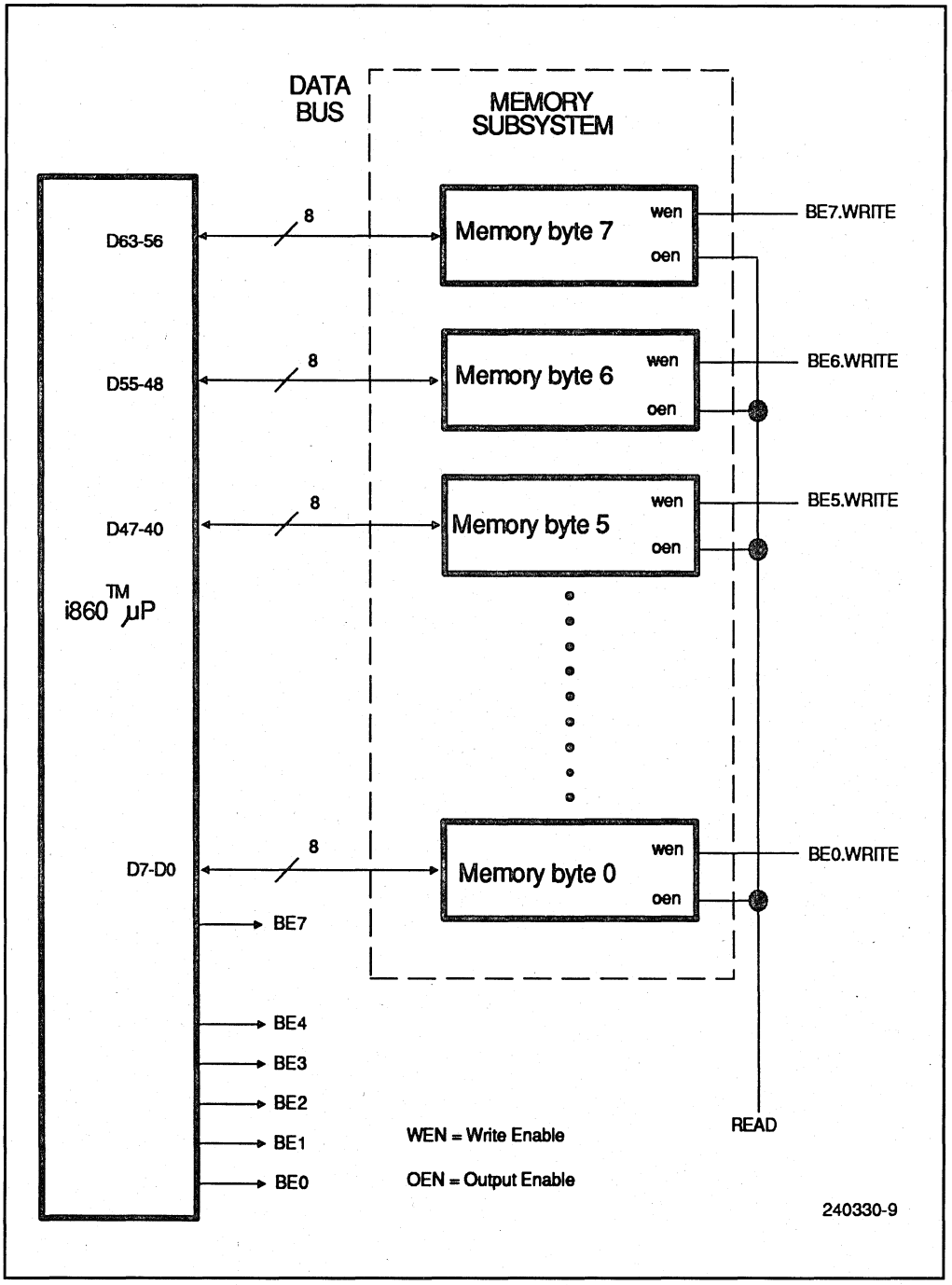


Figure 3-1. Byte Enable Control Signals

The following examples illustrates the use of the two modes of operation. Suppose a C program has the following global declaration for a structure:

```
struct {
        byte A, B;  /* byte */
        short C;    /* 16-bit integer */
        long D;     /* 32-bit integer */
} rec;
```

The fields of rec are addressed as follows: A at rec + Ø, B at rec + 1, C at rec + 2, D at rec + 4. If rec is double word aligned, the rec data structure for big endian mode will have the following form in memory, within the double word address corresponding to the address of rec:

```
Ø     1      2       3      4         5       6      7
A     B    C:msb   C:lsb  D:msb    D:2msb  D:2lsb  D:lsb
```

and in little endian the following form:

```
  7      6       5      4       3      2     1     Ø
D:msb  D:2msb  D:2lsb  D:lsb   C:msb  C:lsb   B     A
```

In the above example msb stands for most-significant byte, lsb for least significant byte, 2msb for second most significant byte, and 2lsb for second least significant byte.

In both cases the addressability of each data item is the same, but the order of the byte addresses for the multibyte data item is different.

The bytes are organized as follows:

Item	Address	Byte addresses (most significant first)	
		Little Endian	Big Endian
A	0	0	0
B	1	1	1
C	2	3,2	2,3
D	4	7,6,5,4	4,5,6,7

The default mode of operation for the i860 microprocessor is little endian. As an option that may be dynamically selected in supervisor mode, the i860 can operate in big endian mode. Figure 3-2 illustrates how the i860 can operate in big endian mode. Figure 3-2 illustrates how the i860 handles little endian and big endian operations. When loading a register from memory, the i860 internally byte shifts the incoming data bus to the right as required to align the least significant byte (right side) of the addressed data with the least significant byte of the register.

When storing register data in memory, the i860 shifts left the register data by the appropriate number of bytes required to align the data with the memory bytes addressed. This data is presented on the bus, and an external write cycle is performed. The bus enable lines that correspond to the data bytes being written are made active.

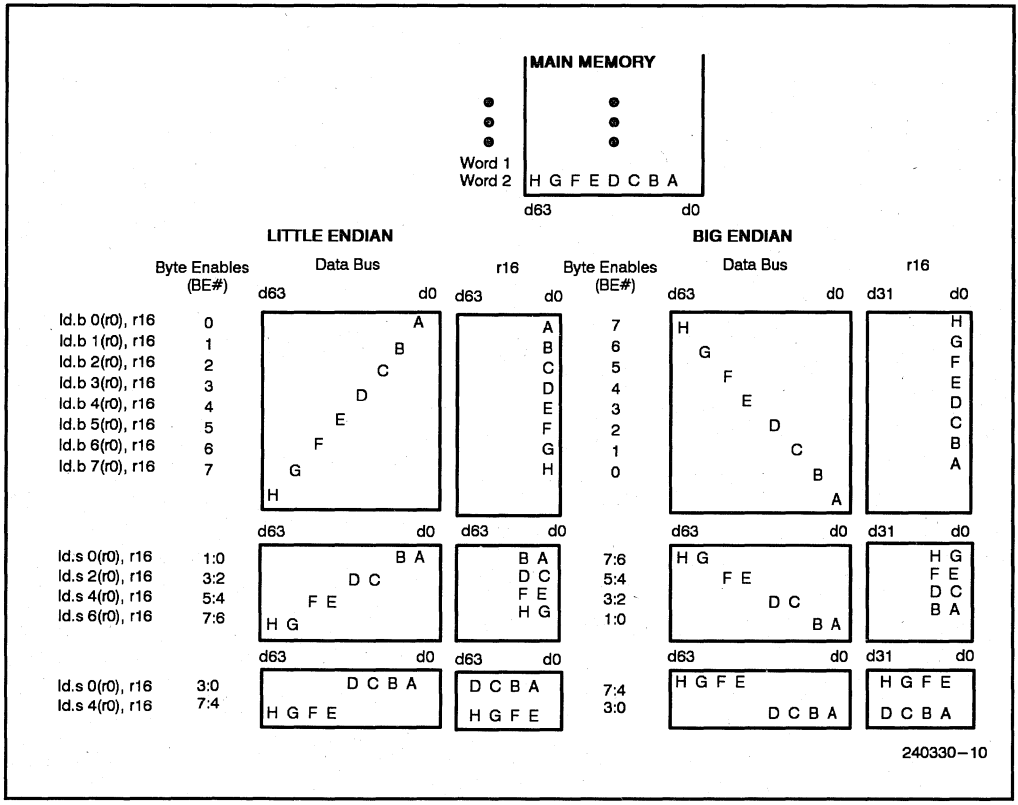


Figure 3-2. Little and Big Endian Data Access

The determination of how many bytes to shift right during the bus operation, or left during a store operation, as well as the selection of the appropriate byte enable signals is determined by the least-significant 3 bits of the memory address involved, by the size of the data, and by the endian-mode selected.

The big-endian little-endian example given earlier and Figure 3-2 illustrate the operation under both endian-modes. The reading or writing the A data byte in little endian shifts the data by zero bytes (no shift). For a write operation, activation of byte enable signal BE0#, stores the memory byte corresponding to bits d7–d0 of the 64-bit memory entry addressed. In big endian, reading A shifts the data bus right by seven bytes, and writing shifts the register data left by seven bytes and activates BE7#, to store it in the byte corresponding to bits d63–d56.

Big endian mode in the i860 essentially inverts the byte offset addresses, converting address 7 into 0, 0 into 7, 6 into 1, 1 into 6 and so on.

3.3.1.5 ENDIAN MODE FOR CODE ACCESSES

Code accesses are always done with little endian addressing. This implies that code will appear differently than documented here when accessed as big endian data. Intel recommends that disassemblers running in a big endian system convert instructions which have been read as data back to little endian form and present them in the format documented here.

3.3.1.6 SYSTEM OPERATION AND ENDIAN MODE

Systems based on the i860 microprocessor can be designed for normal operation in little endian or big endian mode. The natural endian-mode operation is established by the way that external devices address the byte offsets within the 64-bit aligned memory locations. Typically, the system will be used only in the natural mode of operation. The i860, however, allows for dynamically changing the endian-mode by software executing in supervisor mode.

3.3.2 Basic Bus Operation

The i860 microprocessor's fully-synchronous external bus may operate without pipelining or with up to two levels of pipelining to boost memory subsystem throughput. All control signals that affect bus operations are sampled relative to the rising edge of the clock.

A bus cycle begins when ADS# is sampled active and ends when READY# is sampled active. READY# is sampled on every cycle after ADS# is sampled active. New bus cycles can be started on any clock cycle after a one clock cycle delay following the beginning of the prior bus cycle. Thus, new cycles can start as often as every other cycle. Pipelining allows up to three outstanding cycles to exist concurrently. A bus cycle is considered outstanding while its associated READY# has not been sampled active.

The processor can generate pipelined and nonpipelined read and write bus cycles, as requested by the memory subsystem. A pipelined cycle starts while one or two other bus cycles are outstanding. Pipelined cycles are started under control of the NA# signal as explained in Section 3.3.4.

3.3.3 Nonpipelined Bus Operations

Bus cycles require at least two clock cycles to complete. The state diagram in Figure 3-3 illustrates how the bus operates in nonpipelined mode.

The state machine assumes the idle T_I state when there are no processor requests for external bus cycles (indicated by REQ in Figure 3-4). When the processor requests a bus cycle, the bus controller transitions to the T_1 state and the ADS# signal is asserted. ADS# can be sampled by the memory subsystem at the end of the T_1 clock. ADS# assertion indicates the beginning of a bus cycle. The T_1 clock is always followed by the T_{11} clock during which the bus cycle is allowed to complete. The address bus (A32–A3) and the signals W/R#, NENE# and PTB are all made valid and stable prior to the end

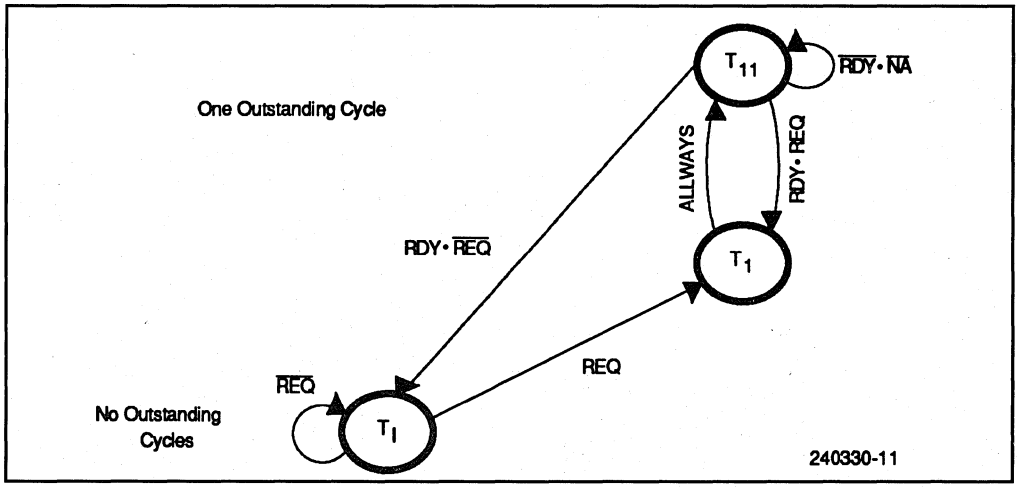


Figure 3-3. Nonpipelined Bus State Machine

of the first T_{11} clock cycle. During read operations, the data bus (D63–D0) is floated to allow the memory subsystem to drive the data. During write operations, the processor drives the data bus and makes it valid and stable before the end of the first T_{11} clock cycle.

The memory subsystem does not assert the READY# signal unless the read and write cycles finish within the first T_{11} clock. The diagram in Figure 3-3 shows that the state machine repeats the T_{11} state to add wait-states as long as NA# is not asserted. Wait-states can be added as needed by leaving READY# unasserted. The signals involved in the memory cycle remain valid during wait-states. Assertion of NA# allows pipelined cycles to take place (Section 3.3.4. provides further explanation). When the processor samples the READY# signal, indicating completion of a bus cycles, the state machine transitions to the T_I state, if there is no new processor bus requests at the time. If new requests are present, the state machine assumes the T_1 state and a new cycle begins.

3.3.3.1 NONPIPELINED READ OPERATIONS

Figure 3-4 shows that read operations can complete at end of the first T_{11} state to produce two-clock read cycle. To achieve this level of efficiency, the memory subsystem must provide read data, assert READY# and allow for processor sampling before the read cycle completes. This calls for an extremely fast address to data access time (the i860 microprocessor Data Sheet provides detailed timing information). Required memory access times can be relaxed by adding wait-states or by using pipelining. Each wait-state eases the access time requirement by one clock period.

3.3.3.2 NONPIPELINED WRITE CYCLES

Figure 3-5 shows a timing diagram of back-to-back write cycles. To perform a nonpipelined write cycle, the processor asserts the ADS# signal during T_1 and drives the signals needed to perform the bus cycle. These include not only the same signals needed for a

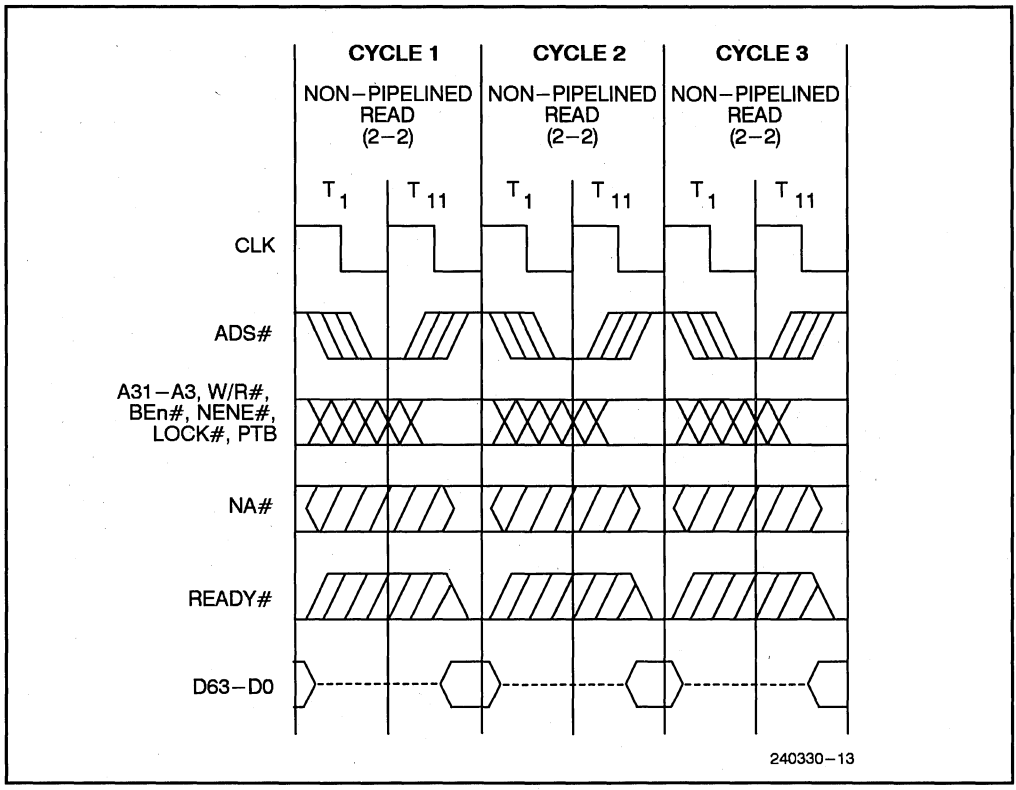


Figure 3-4. Fastest Read Cycles

read, such as the address bus, W/R#, etc; but also the write data on the data bus, and the byte enable lines BE7#–BE0#. Write operations differ from read operations in that the processor does not need to wait for memory to finish its cycle in order to continue computing. The memory subsystem can merely latch the address bus, data bus, byte enable lines and other bus related signals, while simultaneously asserting READY#, allowing the processor to continue computing. The memory subsystem can perform the write operation while the processor is starting a new cycle. Thus, in Figure 3-5, cycle 1 can physically write to memory while T_1, and T_{11} of cycle 2 sets up the next write cycle. In this way, cycle 1 has two full clock cycles to operate and there is no need for wait-states. This situation is different for read cycles, where the required RAM access time needed to perform a two-clock nonpipelined operations is very small.

Unlike cachable read operations, write operations make use of the processor-driven byte enable signals BE7#–BE0# driven by the i860 microprocessor (Section 3.3.1.3 provides further explanation).

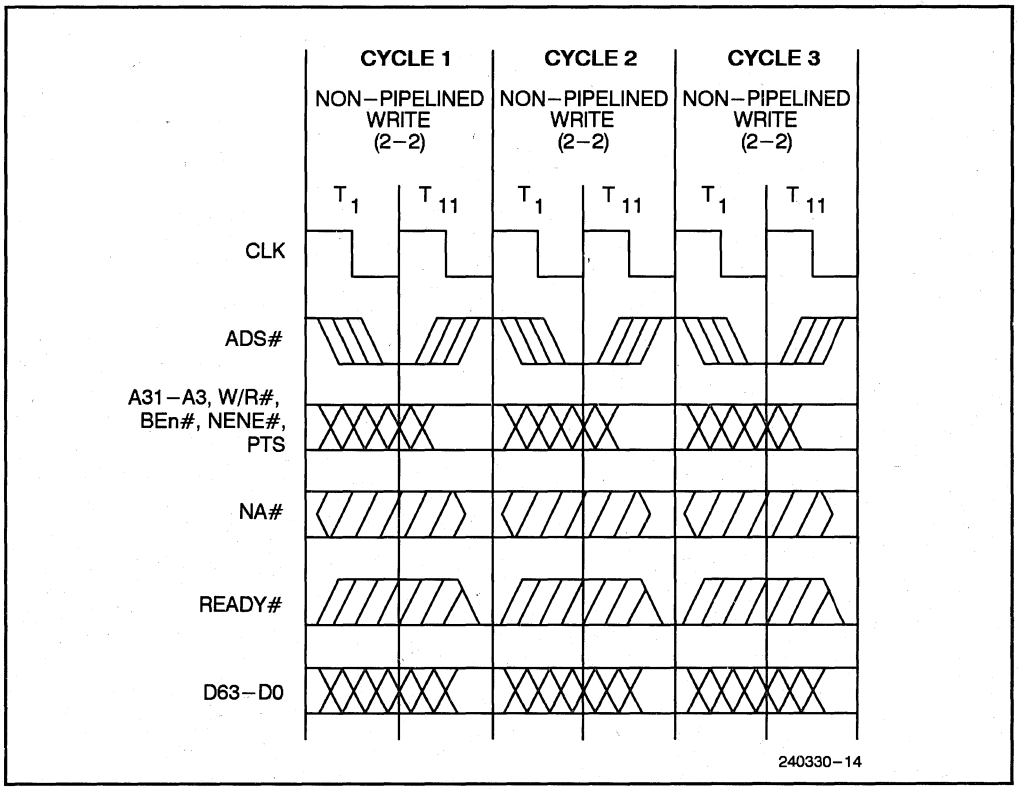


Figure 3-5. Fastest Write Cycles

3.3.3.3 WRITE CYCLES FOLLOWING READ CYCLES

The timing diagram in Figure 3-6 shows that even though the data is not guaranteed valid until the later part of the T_{11} cycle, the processor may begin driving the data bus early in the T_1 cycle. If a read cycle precedes a write cycle, there could be contention for the data bus between read data being held by the memory subsystem past the beginning of the write's T_1 cycle, and write data being driven early in T_1 by the processor. As shown in Figure 3-6, the processor avoids this problem by delaying a write operation by one clock cycle when it follows a read operation. The processor does not start driving the write data for the write cycle (cycle 2 in Figure 3-6) until the beginning of T_{11}, as opposed to the write cycle in Figure 3-5, which starts driving the data as early as T_1.

Memory subsystem design must add a wait-state to accommodate the special handling of write cycles that follow read cycles. Although it is possible for the sake of simplicity to add a wait-state to all write operations, this is undesirable due to the degradation in performance.

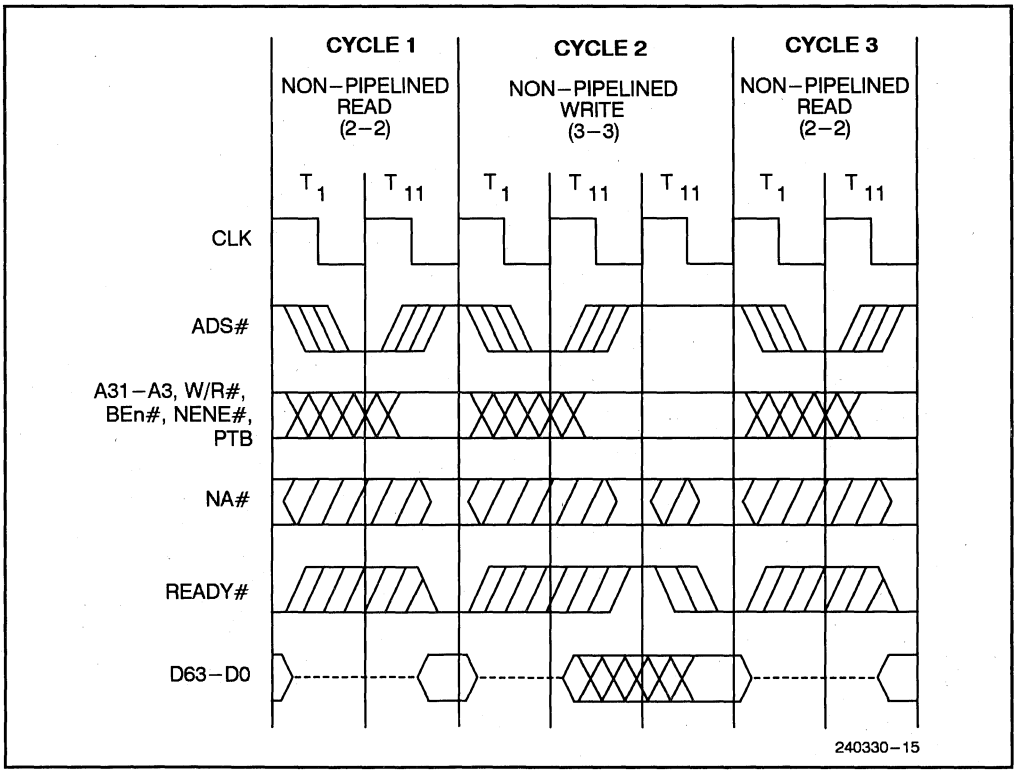


Figure 3-6. Fastest Read/Write Cycles

3.3.4 Pipelined Operations

The i860 microprocessor provides up to two levels of pipelining for 64-bit read and write operations. Two levels of pipelining implies the presence of three outstanding cycles, one level implies two outstanding cycles, and no pipelining implies no more than one outstanding cycle.

Pipelining of the external bus is controlled by the next address signal, NA#. By use of this signal the memory subsystem allows, zero, one, or two levels of pipelining. After a clock during which the processor has asserted ADS# to start a memory cycle, the memory subsystem can assert NA# to indicate that even though the outstanding cycle is not finished, the processor can, if it needs to, start a new bus cycle. The NA# signal needs to be asserted for only one clock cycle, since it is latched internally. Once an asserted ADS# is latched, a new level of pipelining is permitted even if the processor's bus request comes at a later time.

An **m-n** read or write cycle has a cycle time of **m** clocks and a cycle-to-cycle time of **n** clocks (**m** ≥ **n**). Total cycle time is calculated from the clock in which ADS# is asserted to the clock in which READY# becomes active. Cycle-to-cycle time is calculated from the time that READY# is sampled active for the previous cycle to the time that it is sampled active for the current cycle.

Pipelining may begin whenever a bus cycle in progress requires more than two clock cycles to finish (**m** > 2). A new cycle may begin while another cycle is still in progress, if assertion of NA# requests it. NA# is only recognized in a clock where ADS# is inactive.

Figure 3-7 shows how pipelining takes advantage of memory operations with four interleaved memory banks. Pipelined memory reads achieve 6-2 cycles. That is, a total cycle time of six clocks and a cycle to cycle throughput of two clocks. The "A" in the diagram indicates when the address is valid. Total access time in this example is six clock cycles.

3.3.4.1 PIPELINING AND INTERLEAVED MEMORY BANKS

Memory subsystems will typically use as many interleaved memory banks as there are stages in the pipeline. Two levels of pipelining are most effective when a memory subsystem with four interleaved banks. (Using three banks is not feasible because of the difficulty in making an address decoder of this kind.) Interleaved memory banks are designed so that each supports one of several sequential 64-bit addresses. In a two bank system, for example, one memory bank handles odd 64-bit addresses while the other handles the even addresses. In a four bank system, each bank handles one of four consecutive 64-bit addresses. When memory address cycles require a bank currently in use by an outstanding bus cycle, the memory subsystem adds wait-states while the specific outstanding cycle completes. It then resumes with pipelined operations.

In another approach to the use of bus pipelining, the system initiates one more memory operation than the interleaved memory banks can accommodate. When the extra memory cycle is started, the memory subsystem stops issuing cycles until the accessed memory

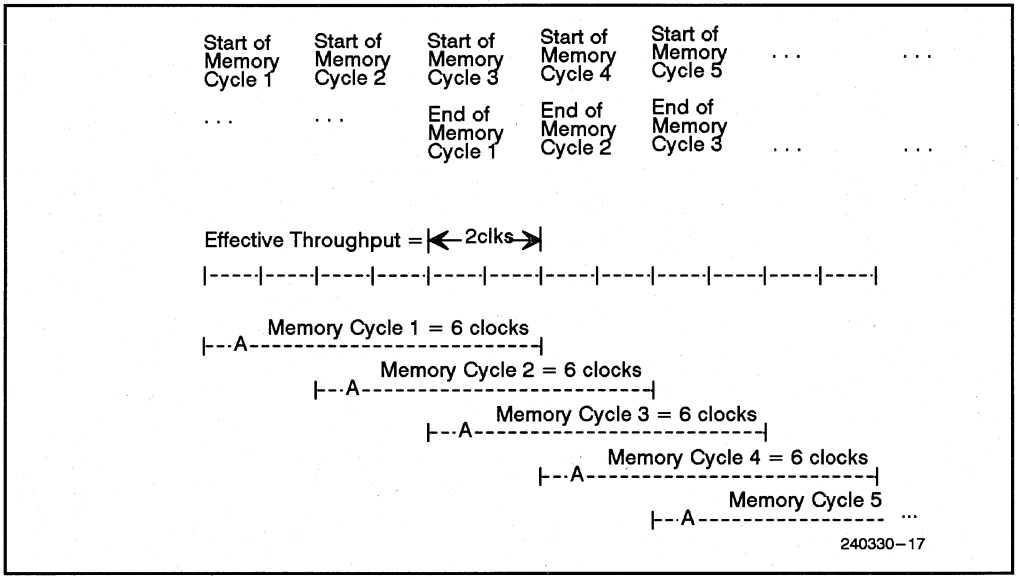


Figure 3-7. Memory Operation Pipelining

bank becomes free. The memory subsystem has all the information needed to start a new cycle. When the new cycle starts, the memory system queues up the next cycle. This eliminates the extra delay required to get a valid address for any queued bus cycle.

3.3.4.2 ORDERING OF DATA DURING PIPELINED OPERATIONS

In typical implementations, the memory subsystem indicates completion of pipelined write operations by asserting READY# when the processor provides valid data for writing. When pipelined bus operations are performed, the ordering of data driven onto the bus during read or write cycles must correspond to the order in which the outstanding cycles were initiated. If two read operations and a write operation are outstanding, the processor will not provide the write data until the outstanding read operations are completed. The memory subsystem must have a state machine that tracks when to drive the read data, and when to latch the write data.

3.3.4.3 BUS STATE MACHINE FOR PIPELINING

Operation of pipelined bus cycles is illustrated by the bus state machine shown in Figure 3-8. Transitions are made on every clock cycle according to the state of the signals provided with the transition paths.

The state machine is divided into sections that indicate zero, one, two and three outstanding cycles. Sections that indicate zero and one outstanding cycle are small extensions of the nonpipelined bus state machine shown in Figure 3-3.

Two three-state machines have been added to the nonpipelined operation of Figure 3-3. They indicate state machines for two and three outstanding pipelined cycles. The states are labeled with subscripts in the form of T_j or T_{ji}. The **j** subscript indicates the number of outstanding cycles, and the **i** subscript indicates substates 1 and 2.

There are three processor states for each of the **j**-states. The states labeled with a single subscript T_j correspond to the initial state entered to begin a new cycle, given $\mathbf{j}-1$ or no outstanding cycles. A T_j state is entered only once for each external bus cycle and asserts the ADS# signal. A T_j state for $\mathbf{j}>1$ is not entered unless there is a request from the processor and the NA# has been or is in the process of being asserted.

Notes:

- RDY# in the figure corresponds to the READY# signal
- NA# is not sampled during ADS# active clock
- ADS# is made active in T_1, T_2 and T_3
- REQ corresponds to an internal processor bus request

T_{j1} is an auxiliary state which helps to complete memory cycles. T_{j1} states operate as the T_{11} state does for in nonpipelined cycles. During the first T_{j1} cycle in a given operation, signals that are relevant to the operation (such as address bus, W/R# and NENE#) are made valid and kept valid during subsequent T_{j1} wait-states. A T_{j1} state can be repeated on subsequent clock cycles as wait-states are introduced (that is, READY# remains unasserted).

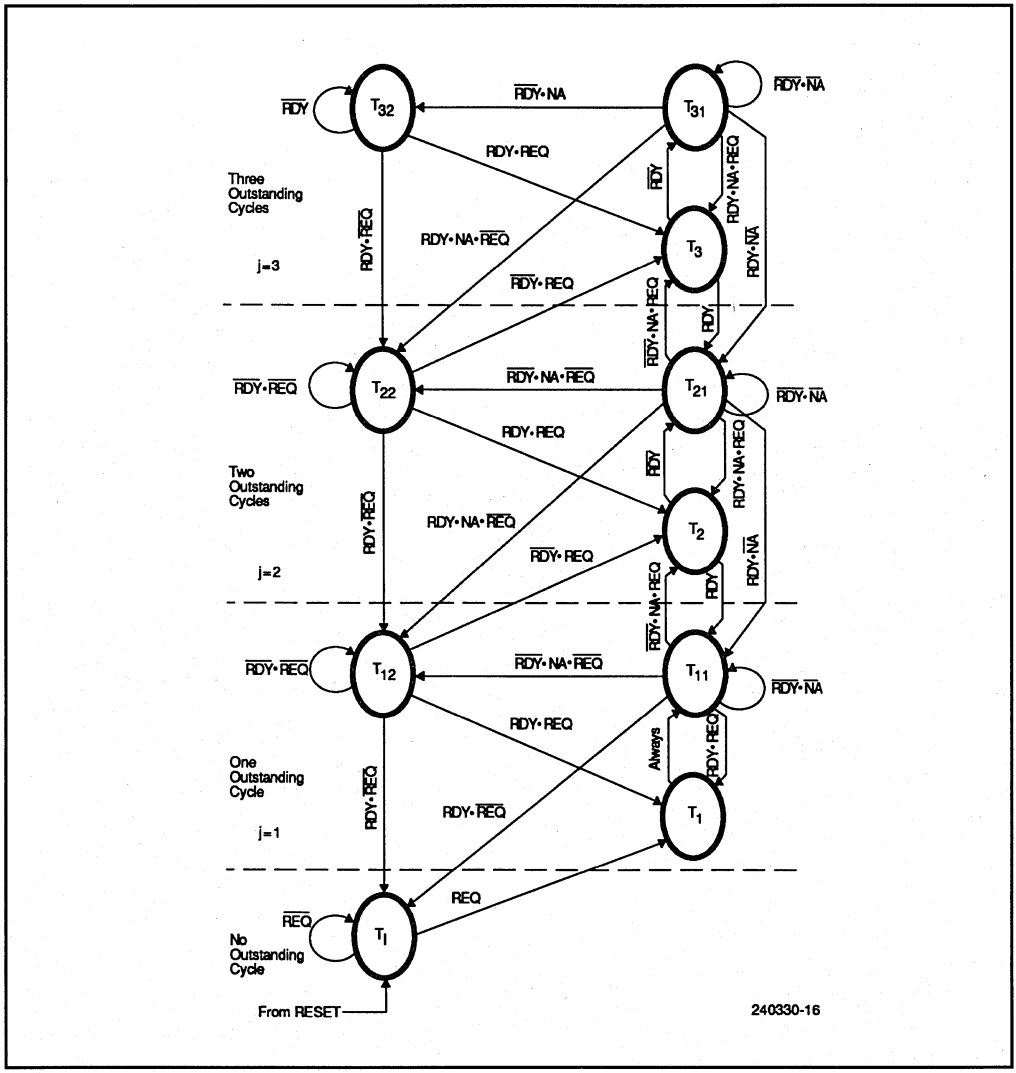


Figure 3-8. Pipelined Bus State Machine

If the processor is not requesting a new cycle, T_{j2} can be entered from T_{j1} when synchronous sampling detects an asserted NA# while a memory cycle is still active. A cycle request is indicated by the REQ internal signal. The T_{j2} state performs like the T_{j1} state but remembers that an NA# signal has been asserted and that a further level of pipelining can be introduced at the processor's request.

As mentioned, pipelined cycles occur when the memory subsystem asserts NA# while the CPU is processing an outstanding cycle. (The memory subsystem should not assert NA# when there are no outstanding cycles.) The NA# signal tells the processor that the

memory subsystem is prepared to receive another cycle while the previous cycle completes. In Figure 3-9, for example, NA# is asserted by the memory subsystem before the end of the T_{11} cycle while the READY# signal remains unasserted. In this case, the memory subsystem has stored all the information it needs to complete the outstanding cycles. As soon as the processor is ready to request a new memory cycle (as indicated by the REQ signal) and starting on the clock following the sampling of NA#, the processor provides a new ADS# which starts the memory cycle.

3.3.4.4 PIPELINED READ AND WRITE CYCLES

Figures 3-9 and 3-10 illustrate pipelined memory cycles. Figure 3-9 shows four memory cycles. Nonpipelined cycles begin when there are no outstanding cycles. The digits in parentheses (such as 5-2 for cycle 2 in Figure 4-5) characterize the number of clocks needed to perform a pipelined cycle. The first digit indicates the total number of clocks between cycle initiation (ADS# assertion) and completion (READY# assertion). The second digit indicates the throughput rate (the number of clocks between READY# signals).

Figure 3-9 illustrates the sequence of events in a group of pipelined cycles. The first operation is the cycle 1 memory read which begins when T_1 ends while asserting ADS#. At the completion of the T_{11} state, address lines A31–A3, W/R#, NENE# and PTB# are valid and latched by the memory subsystem. During this T_{11} clock cycle, the memory subsystem asserts NA# to request a pipelined cycle. The state machine transitions to T_2 initializing the read cycle, cycle 2, by the assertion of ADS#. During T_{21}, the memory subsystem latches the various signals needed to perform the cycle, and asserts NA# to start another pipelined cycle. As a result, the processor requests another cycle. The state

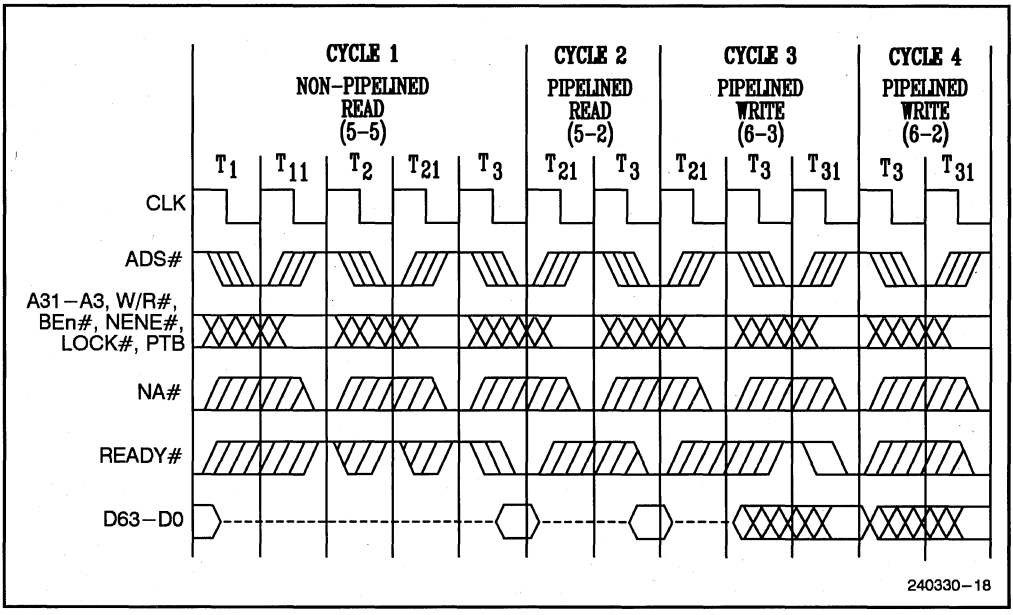


Figure 3-9. Pipelined Read Followed by Pipelined Write

machine transitions to T_3 and a third cycle begins by another assertion of ADS#. During the same T_3 clock, READY# is asserted indicating that cycle 1 has completed and that the data bus has valid data. The state machine transition from T_3 to T_{21} should be understood to gain insight as to how the pipelined bus state machine operates. Cycle 3 starts with a T_3 state and ends with a T_{21} state (instead of T_{31}), since the sampling of an active READY# signal reduces the number of outstanding cycles back to two. Cycle 3 is a write cycle, so during T_{21}, all the previously mentioned bus cycle related signals in addition to the byte enable lines (BE7#–BE0#) and the data bus are made valid and latched by the memory subsystem. During T_{21} a new NA# is asserted, allowing the state machine to start cycle 4 and transition into T_3 again. Another READY# is asserted and cycle 2 completes, leaving cycles 3 and 4 outstanding. For the remaining cycles, the state machine oscillates between T_3 and T_{31} thus maintaining two levels of pipelining (three outstanding bus cycles).

The processor does not drive cycle-3 data onto the Data Bus until two READY# assertions indicate completion of pipelined cycles 1 and 2. Note that since the cycle-3 write operation follows a read operation (cycle 2), the processor waits an additional clock cycle before driving cycle-3 data. The memory subsystem must detect the read/write sequence and delay by one clock cycle the assertion of READY# and the sampling of write data.

Figure 3-10 shows a nonpipelined write operation followed by a pipelined write and two pipelined reads. The write operations are not preceded by a read operation, and the memory subsystem is not required to add a wait-state to complete the cycle.

3.3.5 8-Bit Bus Transfers for Bootstrapping (CS8 Mode)

Eight-bit code size mode is enabled when the INT/CS8 signal is sampled active at the beginning of the period in which RESET is deasserted (Figure 3-11). In eight-bit code size mode, instruction cache misses are transferred as single byte reads instead of 8-byte reads (using bits D7–D0 of the data bus). This allows the i860 microprocessor to be bootstrapped with an 8-bit EPROM. For these single byte code reads, byte enables BE2#–BE0# are redefined to be the three low order bits of the address bus, A2–A0. While KEN# is asserted, these code byte reads are used to update the contents of the instruction cache. (Section 3.5 explains the function of the KEN# signal). Pipelined memory cycles are not started in this mode, even if NA# is asserted.

In CS8 mode, all program code resides in 8-bit memory (ROM) while data are in 64-bit memory (RAM). A reset operation traps to the 0xFFFFFF00 standard trap handler starting address. Hardware must disable RAM address space covered by the 8-bit bootstrap ROM and enable the ROM or EPROM over this address space. When exiting 8-bit code size mode, programs must output to a special I/O port which tells the processor to unmap ROM or EPROM from memory and map the normal 64-bit memory. Once code is loaded in 64-bit memory, initialization code initiates 64-bit code fetches by clearing the CS8 bit in the DIRBASE register. Once 8-bit code-size mode is disabled by software, it cannot be reenabled until the next hardware reset.

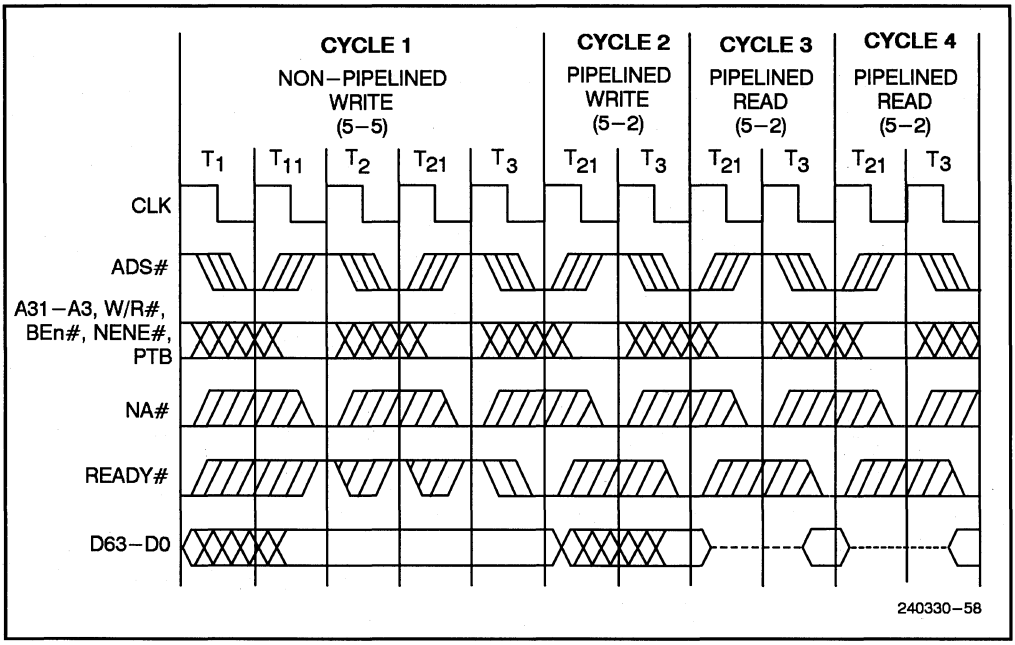


Figure 3-10. Pipelined Write Followed by Pipelined Read

3.4 BUS CONTROL OPERATIONS

This section explains bus control operations including: page-mode and static-column DRAM support, bus arbitration, and bus locking.

3.4.1 Page Mode, Static Column DRAMs and Next Near Operation

The external bus interface facilitates high-performance designs using low-cost DRAMs. The next near signal (NENE#) facilitates designs using page mode and static column DRAMs.

Page mode and static column DRAMs perform best when multiple reads or writes access closely-situated areas of memory (as in the same row address of a given DRAM). This occurs frequently because memory accesses are often sequential.

The NENE# signal indicates that a memory cycle is using the same RAM page address as the previous cycle. The processor ignores a number of lower bits of the address that corresponds to the size of a page in the DRAM. NENE# provides information to the memory subsystem that it can use to enable page mode or the static-column mode operation of the DRAMs. NENE# eliminates the need for external circuitry and eliminates the difficult timing problems that are involved in implementing this capability.

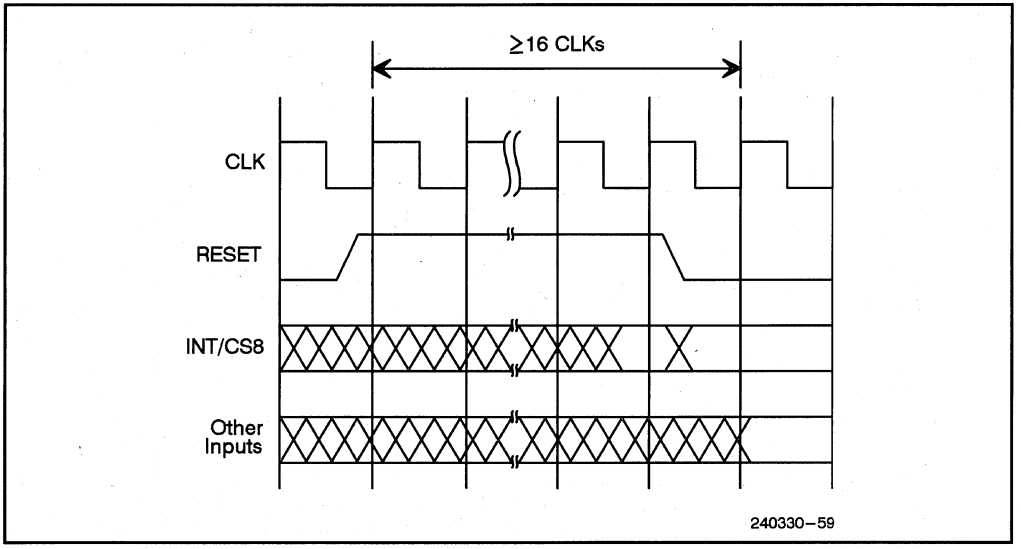


Figure 3-11. CS8 and RESET Activity

The i860 microprocessor determines page size by interpreting the DPS field (bits 3, 2 and 1) of the **dirbase** register. The value in the field indicates the number of least-significant bits to ignore when comparing a cycle address to previous addresses. The number ignored equals 12 plus DPS. Zero is is an appropriate value for 256K × *n* RAMs, 1 for 1M × *n* RAMs, and so on.

NENE# is never asserted on a bus cycle immediately following the deassertion of HLDA. NENE# is not asserted for TLB miss cycles.

3.4.2 Bus Hold, Hold Acknowledge, Bus Request

The i860 microprocessor provides the input line HOLD and the output lines HOLDA (hold acknowledge) and BREQ (bus request) to control arbitration of the buses and control lines.

The hold request signal (HOLD) is driven by an external device or bus arbiter to request control of the bus from the processor. The HOLD signal can be driven asynchronously. The processor has an internal synchronizer to prevent metastable problems when sampling for HOLD. When hold is asserted, the processor blocks any new cycles. Once all outstanding bus cycles are completed, the processor relinquishes bus control by floating all output lines except HOLDA and BREQ. HOLDA indicates to the requesting device that bus control has been relinquished. The device can then use the signals and buses it needs until it relinquishes control to the processor.

The i860 microprocessor can continue executing from the cache once it has relinquished control of the bus. The BREQ signal indicates to any external bus arbiter that the processor is waiting for an external bus cycle. BREQ provides a means to implement a more efficient system for arbitrating between bus contenders such as a DMA device or another processor.

The bus state machine in Figure 3-12 illustrates the process that brings bus activity to a halt when the HOLD input signal is asserted. This diagram is similar to the bus state machine in Figure 3-8, except it includes the hold state and labels the REQ signals as REQNH. REQNH corresponds to REQ ANDed with NOT HOLD. ANDing all REQ signals with NOT HOLD prohibits the bus control unit from initiating new cycles while a hold is pending. Once all outstanding cycles are done, the state machine assumes the T_I idle state where the SHOLD signal (synchronized version of HOLD) causes a transition to the hold state T_H. When the processor detects that the SHOLD signal is no longer active, the state machine transitions back to T_I, if no internal requests are pending (NOT REQ), or to T_1 to start a new bus cycle.

Figure 3-13 timing diagrams illustrate the operation of HOLD, HOLDA, and BREQ. HOLD is asserted within a T_{21} cycle. Since external HOLD must be synchronized internally, SHOLD is asserted one clock cycle after the assertion of HOLD. This explains why the T_2, T_{21} bus cycle was started. This new bus cycle would not have started if HOLD had been asserted one clock earlier. The state machine completes all outstanding cycles and ignores internal bus requests indicated by the assertion of BREQ. Once the T_I idle state is reached, the hold state T_H is entered. HOLDA is then asserted, and the microprocessor floats all output signals except HOLDA and BREQ. Within the second hold cycle, the HOLD signal deasserts. The state machine exits the hold state and transitions to T_1 to perform another bus cycle request. Unlike the assertion of HOLD, deassertion of HOLD resets the SHOLD signal so that, on the next clock cycle, the bus can be recovered and HOLDA made inactive. The recommended set-up and hold times are required to ensure release of the hold state on the clock cycle following the deassertion of HOLD.

BREQ is activated by the processor's internal bus requests independent of the state of HOLD and HOLDA.

Notes:

- RDY# in the table corresponds to the READY# signal
- NA# is not sampled during ADS# active clock
- ADS# is made active in T_1, T_2 and T_3
- REQNH is the internal bus request ANDed with synchronized HOLD
- HLDA is made active in T_H
- HOLD is synchronized internally.

3.4.3 Bus Lock

The bus lock signal (LOCK#) provides indivisible read-modify-write memory operation sequences for use in multiprocessor systems. Multiprocessor systems with an external arbiter should also use the LOCK# signal to prevent granting the bus to other masters.

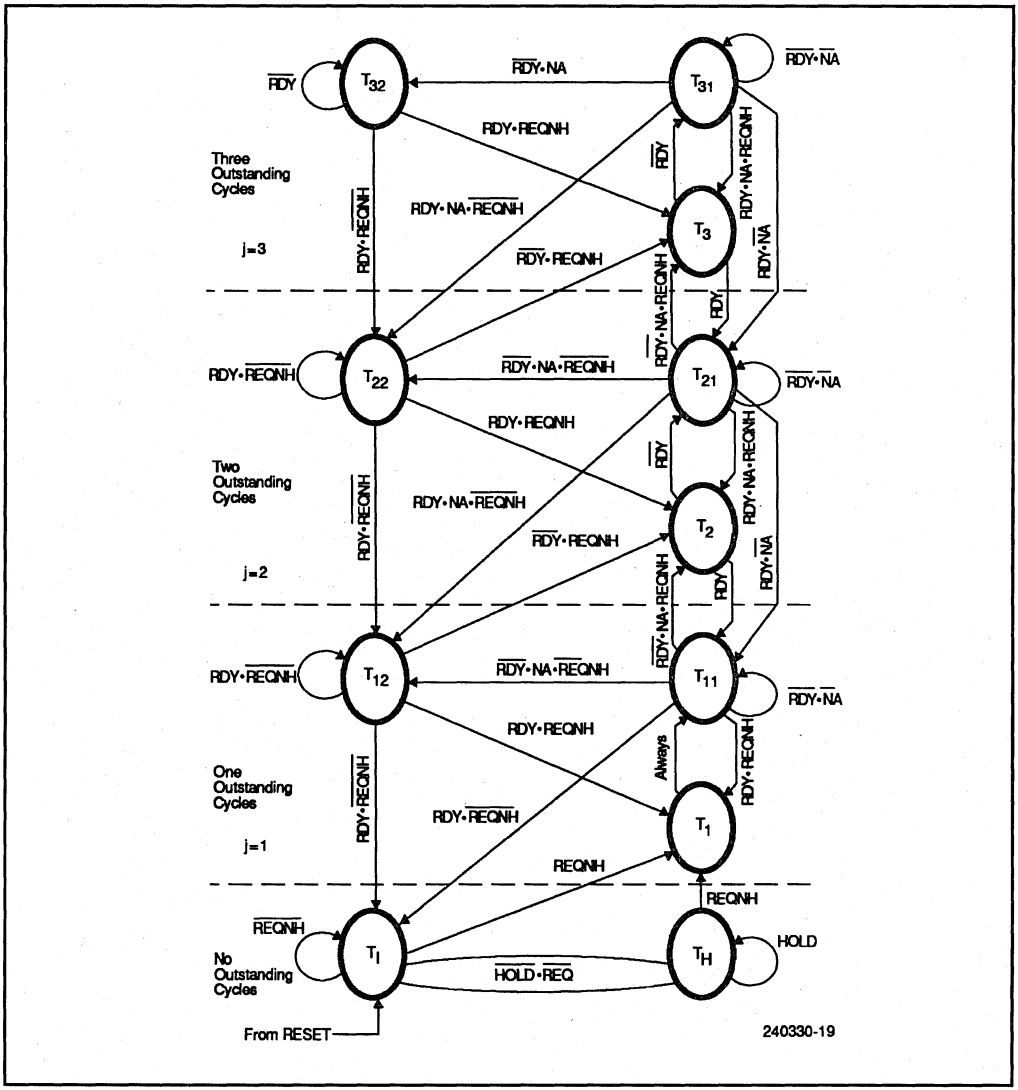


Figure 3-12. Pipelined Bus State Machine Including Hold State

Programmers need not track synchronicity between the bus and the instruction that sets the BL bit. LOCK# is asserted along with ADS# on the first data bus cycle following the setting of the BL bit. LOCK# and ADS# are deasserted on the first data bus cycle after clearing of the BL bit.

The bus is not locked until after the first data access cache miss causes the assertion of the LOCK# signal. Multiprocessor system software should therefore ensure that the first load instruction in a locked sequence references noncachable memory.

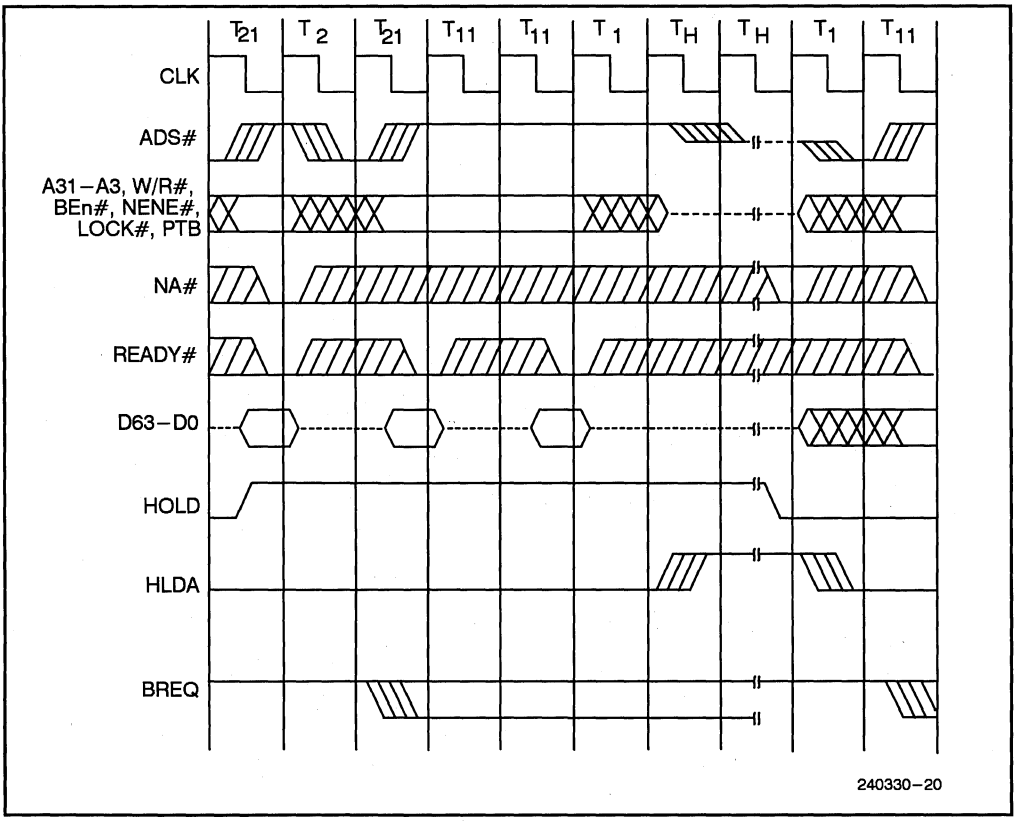


Figure 3-13. HOLD, HLDA, and BREQ

3.4.3.1 SUPERVISOR-MODE ACTIVATION OF LOCK#

Supervisor mode software can set or clear the BL bit in the **dirbase** register directly.

3.4.3.2 USER-MODE ACTIVATION OF LOCK#

User mode software can set or clear the BL in the **dirbase** with the **lock** and **unlock** instructions. These instructions support generalized interlocked sequences. The **lock** instruction sets the BL bit in the **dirbase**, and the next load or store operation locks the bus. Interrupts are disabled until the bus is unlocked. The **unlock** instruction clears the BL bit in the **dirbase**, and the next load or store operation unlocks the bus.

Some restrictions apply to this method. An interlocked instruction sequence must not branch or execute outside the 30 sequential instructions that follow a **lock** instruction. The interlocked sequence must be restartable from the **lock** instruction if a trap occurs, as in read-modify-write sequences. In sequences with more than one store instruction, software must prevent traps following the initial nonrestartable store instruction. This ensures that the sequence will not be restarted beyond this point. To meet this condition, the software must read and write the unmodified values of the locations used by the

instructions following the first store, to prevent the possibility of a data access page fault. Software must also ensure that executed code is within the same page to prevent instruction fetch page faults.

The processor will ignore a second **lock** instruction issued before the bus is unlocked. A trap occurs when from 30 to 33 instructions have executed following a **lock** instruction, if a load or store which follows an **unlock** instruction has not been executed.

If a trap occurs between a **lock** instruction and the first load or store following an **unlock** instruction, the interlock bit (IL) in the **psr** is set and BL is cleared. IL indicates to the trap handler that a **lock** sequence has caused the trap. It searches backward for the **lock** instruction and restarts from that point. The trap handler can scan up to 33 instructions; if no **lock** instruction is found, it signals an error to the user code.

3.4.3.3 BUS LOCK DURING PAGE TABLE UPDATE

The i860 microprocessor also asserts LOCK# in TLB miss processing to update the accessed bit within a page-table entry. The maximum time that LOCK# may be asserted for this case is the time needed for hardware to perform a read-modify-write in the page-table entry.

The LOCK# signal is asserted with an appropriate setup time relative to the clock cycle that samples the assertion of ADS#. This ensures that this bus cycle will be part of a locked sequence.

Figure 3-14 illustrates locked cycles. The LOCK# signal is asserted prior to the begining of the T_{11} state of cycle 1 to initiate a locked cycle. The LOCK# signal is unasserted after the end of T_{11} to indicate that cycle 2 is not locked. This particular example is not of practical use, since it locks a single cycle, but it illustrates the LOCK# signal timing.

3.5 CACHE CONTROL OPERATIONS

The KEN# (cache enable) input signal enables or disables cache updates. The PTB output signal is used when paging is enabled to provide the memory subsystem with cache control page table information relating to the current cycle.

The i860 microprocessor prevents updates of both the data and instruction caches unless KEN# is sampled active. Enabling cache updates for a given cycle requires that KEN# be sampled active on the clock period after ADS# through the clock in which NA# or READY# is asserted. Updating either cache involves the generation of four 64-bit read operations. If KEN# is found deasserted for the first fetch cycle, the data being read is not cached.

Accesses to data or instructions already cached are not inhibited by KEN#. To prevent such accesses the caches must be flushed.

When paging is enabled, the values in the cache disable (CD) and write through (WT) page table bits are used to determine the caching strategy. During memory transfers, the secondary page table entry's CD bit indicates whether accessed data should be stored in an external cache. CD in the first-level page table is ignored. The WT bit allows software to determine the internal caching strategy. If WT is set in a page table entry (PTE), on-chip caching for data is inhibited. If WT is clear, normal write-back policy applies to

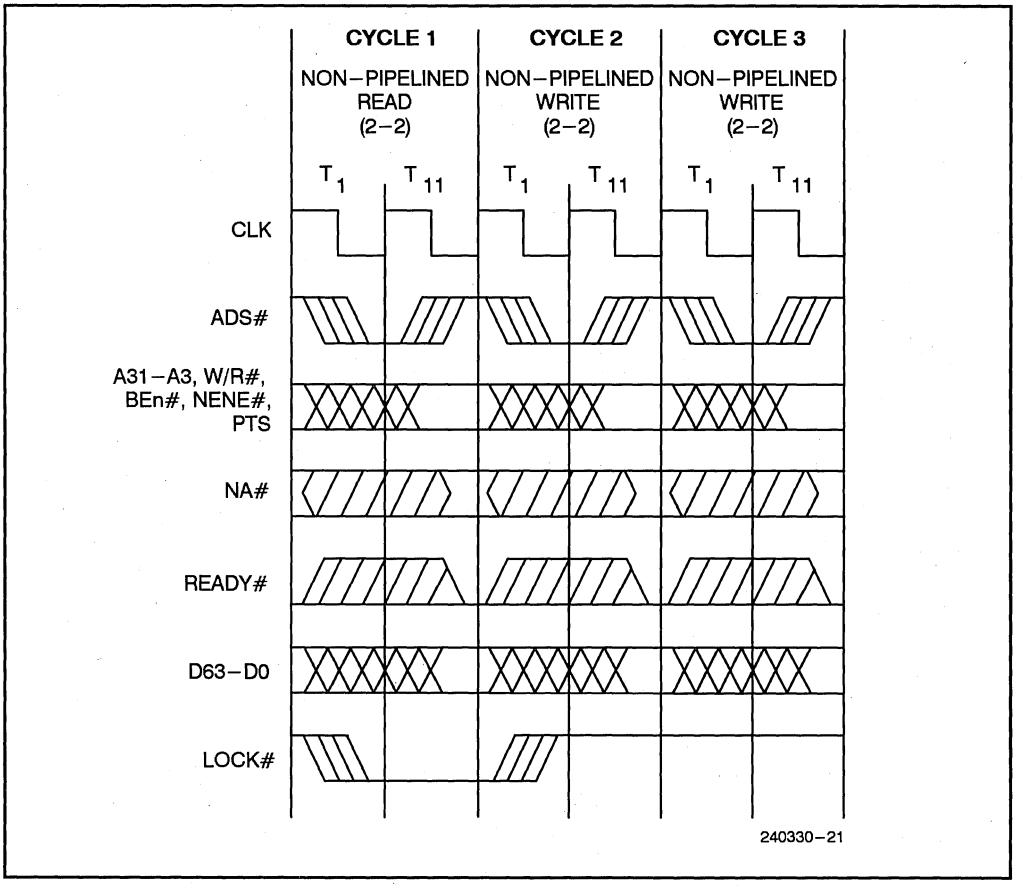


Figure 3-14. Locked Cycles

page table data. The WT bit of the page directory entries is reserved and is not referenced by the processor. The WT bit is independent of the CD bit; data that is not stored in the on-chip caches may be placed in an external cache.

The internal cache is disabled when a CD bit is set or when an external KEN# signal is unasserted. The CD and WT bits do not function when paging is disabled.

When the PBM bit in the **epsr** is set, the PTB output signal reflects the value of the WT bit. When clear, PTB reflects the value of the CD bit. The PTB output signal remains unasserted while paging is disabled.

3.6 TRAPS AND INTERRUPTS

Traps are caused by external conditions or by exceptional conditions detected in the course of a program's execution. A trap causes the instruction being executed to abort and transfers control to a trap handler program stored in the hexadecimal virtual address 0FFFFFF00.

The trap handler reads the **psr** and **fsr** registers to identify the cause of the trap and branches to the portion of code that processes the trap (or traps if more than one occurred simultaneously). Once the trap handler has executed code to service the trap conditions involved, it restores the state of the processor, restarts the aborted instruction and resumes normal program execution. The i860 microprocessor is designed to ensure that all program instructions can be restarted. The *i860™ 64-Bit Programmer's Reference Manual* explains how to write trap handling software. Table 3-2 summarizes the causes and indications of various traps.

The reset trap and interrupt trap are caused by external conditions. Reset traps are caused by hardware resets. (Section 3.8 provides further explanation.) Interrupts are caused when an external source asserts the INT signal while interrupts are enabled (i.e. while the IM interrupt mask bit is set in the **psr**). The trap handler immediately saves and clears the IM to inhibit further incoming interrupts.

The i860 microprocessor does not provide an interrupt acknowledge signal. After the interrupt takes place, the trap handler can read from the external I/O system to determine the source of the interrupt. The processor can then indicate to a port that the interrupt has been processed.

Table 3-2. Types of Traps

Type	Indication			Caused by	
	psr	**epsr**	**fsr**	**Condition**	**Instruction**
Instruction Fault	IT	OF IL		Software traps Missing **unlock**	**trap, intovr** Any
Floating Point Fault	FT		SE AO,MO AU,MU AI,MI	Floating-point source exception Floating-point result exception overflow underflow inexact result	Any M- or A-unit except **fmlow** Any M- or A-unit except **fmlow, pfgt,** any **pfeq**. Reported on any F-P instruction plus **pst, fst,** and sometimes **fld, pfld, lxfr**
Instruction Access Fault	IAT			Address translation exception during instruction fetch	Any
Data Access Fault	DAT			Load/store address translation exception Misaligned operand address Operand address matches **db** register	Any load/store Any load/store Any load/store
Interrupt	IN			External interrupt	
Reset	No trap bits set			Hardware RESET signal	

3.7 TEST SUPPORT FUNCTIONS

The i860 microprocessor has a boundary scan mode for component or board level testing of signals and logic. Special test logic or instrumentation needs only to connect to the CLK, BSCN, SCAN, SHI, BREQ, RESET and HOLD signals to sample and drive the external processor signals. Table 3-3 indicates the signals for the test mode selections.

Sampling of an active boundary scan (BSCN) input signal initiates the boundary scan mode within the following clock cycle. Boundary scan mode may be activated while RESET is active. The processor exits boundary scan mode on the clock cycle following deassertion of BSCN. The internal state is undefined when the processor exits boundary scan mode. RESET should be asserted to reinitialize the processor.

Table 3-3. Test Mode Selection

BSCN	SCAN	Testability Mode
LO	LO	No testability mode selected
LO	HI	(Reserved for Intel)
HI	LO	Boundary scan mode, normal
HI	HI	Boundary scan mode, shift SHI as input; BREQ as output

While in boundary scan mode, the processor may operate in normal mode or shift mode. Shift mode is a submode that is entered on the clock in which SCAN input is asserted. The normal mode is entered on the clock in which SCAN input is deasserted.

For testing purposes, each signed pin has associated with it an internal latch. Table 3-4 identifies these latches by name and classifies them as input, output or control.

Table 3-4. Test Mode Latches

Input Latch	Output Latch	Associated Control Latch
SHI		
BSCN		
SCAN		
RESET		
D0–D63	D0–D63	DATAt
CC1–CC0		
	A31–A3	ADDRt
	NENE#	NENEt
	PTB#	PTBt
	W/R#	W/Rt
	ADS#	ADSt
	HLDA	
	LOCK#	LOCKt
READY#		
KEN#		
NA#		
INT/CS8		
HOLD		
	BE7–BE0#	BEt
	BREQ	

Here is a typical test sequence:

1. Enter shift mode and assign a value to all the latches.
2. Enter normal mode to force values onto desired output pins, and read all pin values.
3. Enter shift mode to read the new values of the input pins.

3.7.1 Normal Mode Operation

Normal mode begins with a clock cycle that samples a deasserted SCAN input signal. Within normal mode, the contents of the output latches are driven onto the output pins or buses if the corresponding control latch bits are set. If the corresponding control latch is not set, the output pins are not driven. RESET and HOLD must be asserted to float the outputs.

3.7.2 Serial Mode Operation

Shift mode begins with a clock cycle that samples the SCAN input in the active state. The value of all pins are immediately loaded into latches. Input pin latches load externally driven values. Floated output or bidirectional pins also read externally driven values. Latches with forced output pin values during normal mode, load their own values and do not change.

Immediately after the clock cycle that sets the shift mode, the output of the BREQ output signal reflects the value of the BREQ latch. On subsequent clocks, the value of all the latches are shifted to the BREQ pin. At the same time, new values are being shifted in via the SHI pin. Figure 3-15 shows the order in which boundary scan latch chains are serially read and written. All outputs except HOLDA and BREQ are floated in shift mode to avoid glitches on the output lines. However, some glitches may be evident in the HOLDA output pin.

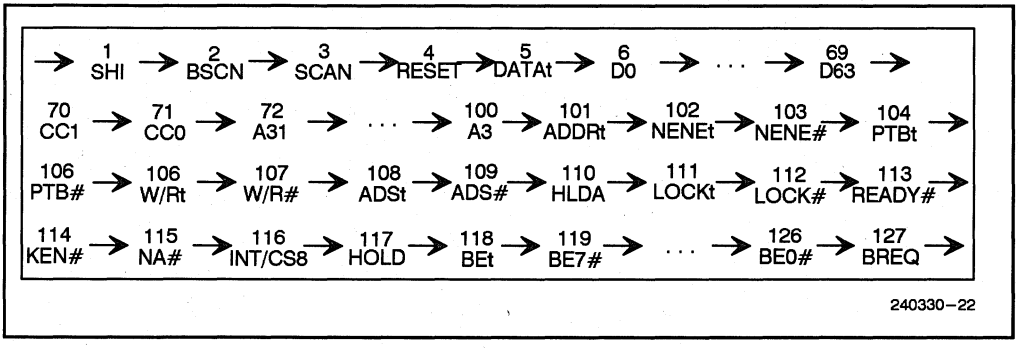


Figure 3-15. Boundary Scan Chain

3.8 RESET AND CLOCK CIRCUIT

i860 microprocessor RESET input must remain high for at least 16 clock periods to initialize the processor. Table 3-5 shows the status of all output pins while RESET is asserted. Once the RESET signal goes low, the processor traps and begins execution at hexadecimal address 0FFFFFF00 to execute the central trap handling routine. The *i860™ 64-Bit Programmer's Reference Manual* details processor status following a reset operation.

The circuit in Figure 3-16 shows the simple system needed to generate the clock and the RESET signal. The clock is a standard TTL circuit rated for the i860 microprocessor configuration used (e.g. 40 MHz).

The RESET signal is triggered when power is first turned on or when the reset button is pushed. Voltage in the capacitor falls to zero, creating a high value for the inverters output. The value is clocked by two D flip-flops to synchronously assert a RESET signal. The RC delay is at least 16 clock periods, raising the inverter input to a voltage level that moves the output to a low state. The low-level signal is clocked through two flip-flops and causes the RESET to deassert. The inverter and the flip-flops are used to clean up the RC signal and synchronize the resulting RESET signal. Two flip-flops help to prevent a metastable state that could be derived when synchronizing the RC network pulse.

Table 3-5. Output Pin Status During Reset

Pin Name	Pin Value	
	HOLD Not Acknowledged	HOLD Acknowledged
ADS#, LOCK#	HIGH	Tri-State OFF
W/R#, PTB	LOW	Tri-State OFF
BREQ	LOW	LOW
HLDA	LOW	HIGH
D63-D0	Tri-State OFF	Tri-State OFF
A31-A3, BE7# - BE0#, NENE#	Undefined	Tri-State OFF

The circuit shown to generate CS8 consists of a driver (e.g. two inverters) to delay RESET and provide the hold time relative to the falling edge of RESET. This signal is ORed with INT. CS8 is inactive after being sampled by the falling edge of RESET, and the INT signal remains unaffected.

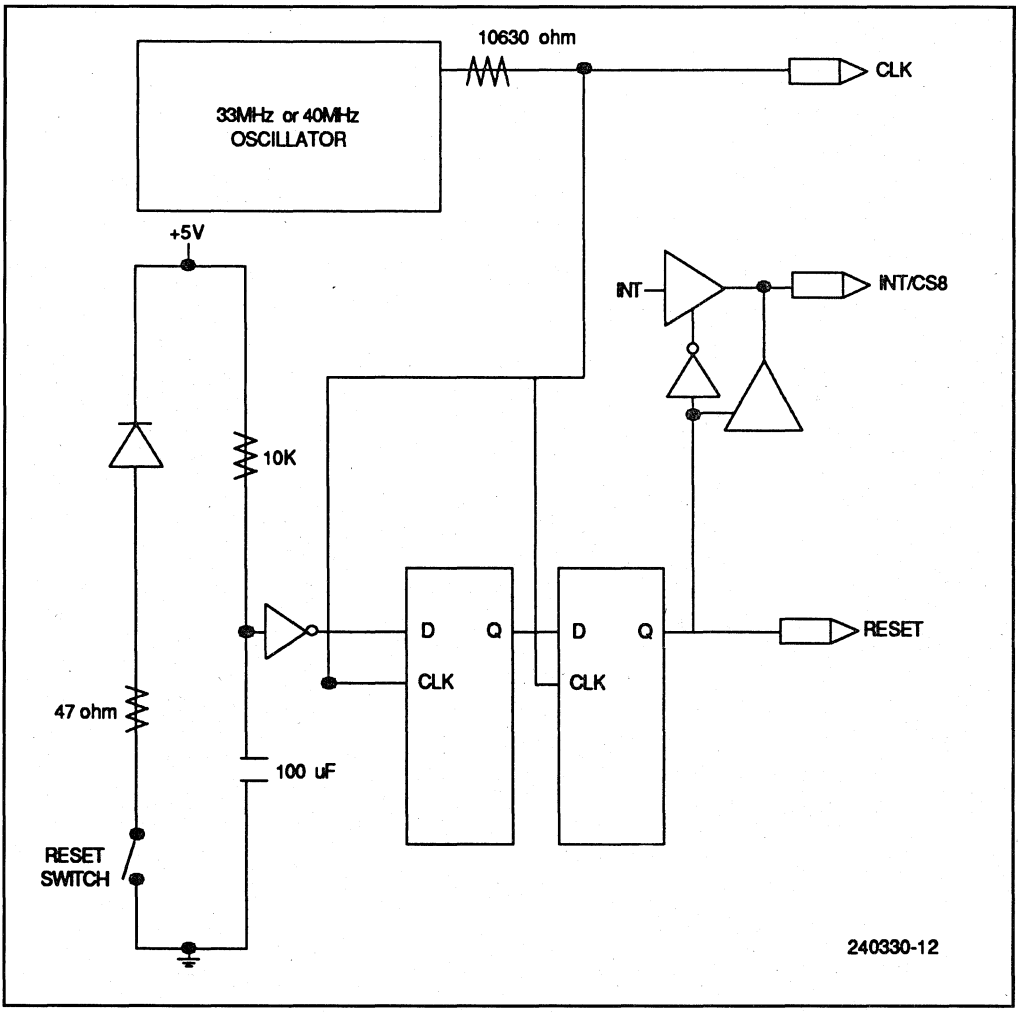


Figure 3-16. Circuit for Clock, RESET and CS8 Generation

Memory Interfacing 4

CHAPTER 4
MEMORY INTERFACING

4.1 INTRODUCTION

The on-chip caches of the i860™ 64-bit microprocessor contribute to its extremely high level of performance. Depending on the locality of memory references, the processor can usually operate from these caches. As a result, fewer accesses are made to the external bus allowing instructions and data to be supplied to the CPU at a very fast rate.

The need for a high-performance cache system stems from the i860 microprocessor's access performance, which is double that of DRAMs. Because of this disparity, the cache is employed to decouple the processor from its external bus. In this way, the effect of a slower DRAM subsystem is greatly reduced.

The locality of memory references is key in determining memory system performance. Software with widely dispersed memory references increase the number of DRAM accesses. The effect of a slower DRAM subsystems on processor performance is much more pronounced in applications characterized by widely dispersed references. These applications, such as Linpack, and graphics matrix operations or file handling applications, such as Troff, warrant the effort and cost needed to design efficient DRAM subsystems. Inoptimal designs can easily reduce performance in these applications by 60 to 70 percent. Poor performance in these applications has a disproportionate detrimental effect on overall system performance, especially if important system functions such as page swapping or display are involved.

This chapter examines a DRAM subsystem design. The example has been optimized to reduce the number of clocks in which the processor waits for the bus. Controller function, processor features, and details such as timing and power consumption are analyzed. The example cannot account for all processor applications, and many applications will implement optimizations not found here. This design has been built and tested, however. If desired, it may be implemented as shown.

The example assumes a working knowledge of computer system design. Items that are discussed but not explained include DRAM operation, PLD programming and operation, worst case analysis techniques, and i860 microprocessor bus operation. The complete schematics and PLD equations are included in Appendix C.

4.2 CPU FEATURES

Certain i860 microprocessor features are specially devised to facilitate optimum DRAM subsystem performance. This section briefly describes these special features.

4.2.1 The KEN# Input

External logic, which is usually part of the DRAM controller, generates the KEN# input. KEN# indicates that the current read cycle is cacheable. As discussed in earlier sections, the processor generates three read cycles (in addition to the current cycle) in

response to KEN#. These cycles provide 32 bytes—enough data to fill one cache line. The handling of these cycles impacts memory system performance and will be discussed in the next section.

KEN# is generated for every cache block fill cycle. It is usually generated by decoding the processor address. Since the processor has no I/O space, memory mapped I/O addresses must be non-cacheable.

Figure 4-1 shows the timing requirements for generating KEN#. The most stringent case is a pipelined zero-wait-state cycle where KEN# must be generated in one clock. Once the address is valid, KEN# must be generated in time to meet the setup time (t_{10}) dictating that KEN# be generated by combinatorial logic.

4.2.2 Bus Pipelining

Obviously, the i860 microprocessor's bus pipelining feature must be incorporated in DRAM subsystem design. This feature allows the processor to overlap bus cycles.

Pipelining is controlled by the processor's NA# input signal. It allows the processor to begin a new bus cycle while another cycle is in progress. It can be activated twice before READY# is activated. In this way, three bus cycles can be activated before the first is completed.

Although the process is not demonstrated in this example, write cycles can be pipelined. In this design, writes are buffered or "posted" and can run without wait-states. As such, write cycle performance does not benefit from the use of pipelining.

Read cycle performance does benefit from pipelining, however, especially if read cycles are due to a floating point vector load. Vector loads can occur in long sequences. Since most of these cycles do not cross a DRAM page boundary, they can be executed with a minimum number of wait-states. Figure 4-2 demonstrates how pipelining facilitates floating-point vector loads.

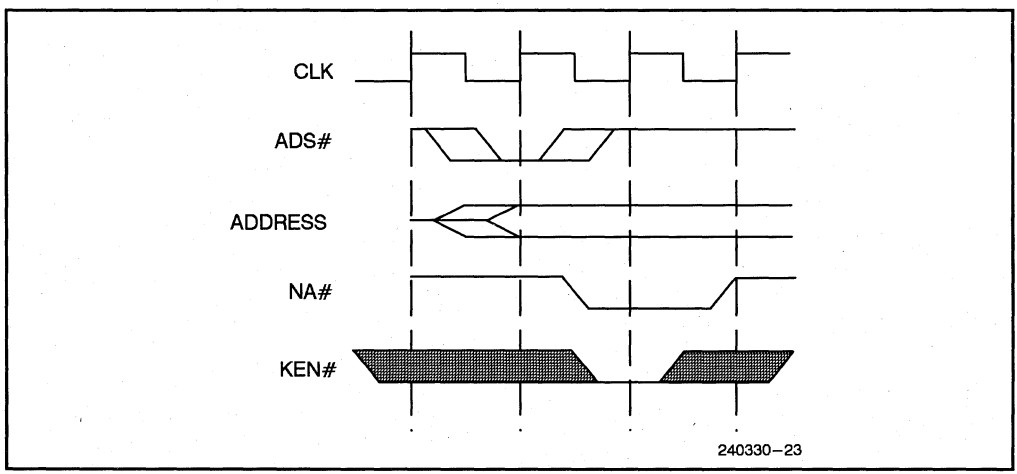


Figure 4-1. KEN# Timing

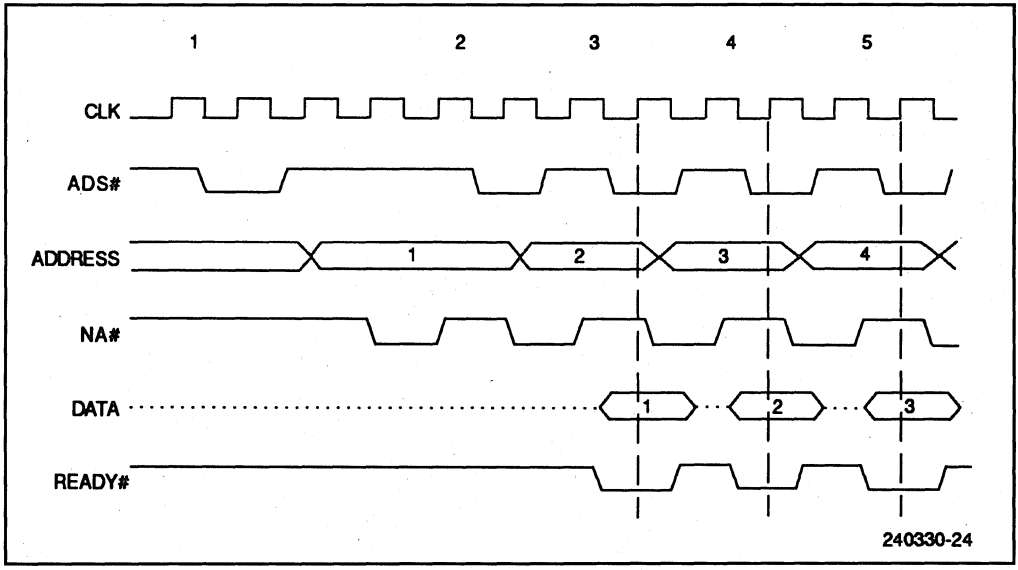


Figure 4-2. Bus Pipelining

The first read cycle in Figure 4-2 occurs from an idle bus state. In this case, the DRAM controller must add five wait-states to the cycle. These wait-states satisfy the RAS access time of the DRAMs. They are added by suppressing READY#. Ordinarily, waiting for READY# is a waste of processor time. Activating NA#, however, allows the next bus cycle to start. NA# is sampled active in the third clock, and ADS# is driven active. The address for the second cycle is valid one clock later.

NA# is activated again before returning READY# to the CPU, and the third vector load is performed. At this point, READY# must be returned to the processor before NA# is activated again. Data for the first read cycle must be valid at the processor data input pins when NA# is activated.

In this example, the pattern can be repeated for an unlimited number of bus cycles. When ADS# is not activated in response to NA#, the pattern is interrupted. Clearly, the benefit of pipelining in this example is that read cycles overlap. Consecutive pipelined read cycles return data to the processor every two clocks.

4.2.3 The Next Near Pin

The next near (NENE#) feature directly supports the DRAM subsystem. (The next near pin is described fully in Chapter 3.) Its function optimizes page or static column mode designs. It indicates that the current row address is the same as the previous address.

This DRAM design example uses static column mode memories. These memory devices contain a row address register. The register simplifies access to memory if the access is within the same DRAM page (row address). In the example, RAS# can remain active from the previous cycle. For read cycles, CS# can also remain active. This type of cycle

saves clocks: the required number of clocks doubles if RAS# must be activated to latch the row address. These extra clocks are required to meet RAS# precharge time. Implementation of this function will be detailed later in this chapter.

The NENE# function optimizes memory subsystem design. If implemented in logic, the function requires at least one register and a comparator. Another benefit is that NENE# is available along with the address, and no additional time is needed to generate this signal.

4.2.4 Write Data Function

Data bus contention could result when a write cycle is immediately preceded by a read cycle. The i860 CPU includes a bus feature to prevent this problem. When a write cycle occurs in the clock following a read cycle, write data valid timing changes. Figure 4-3 illustrates this change.

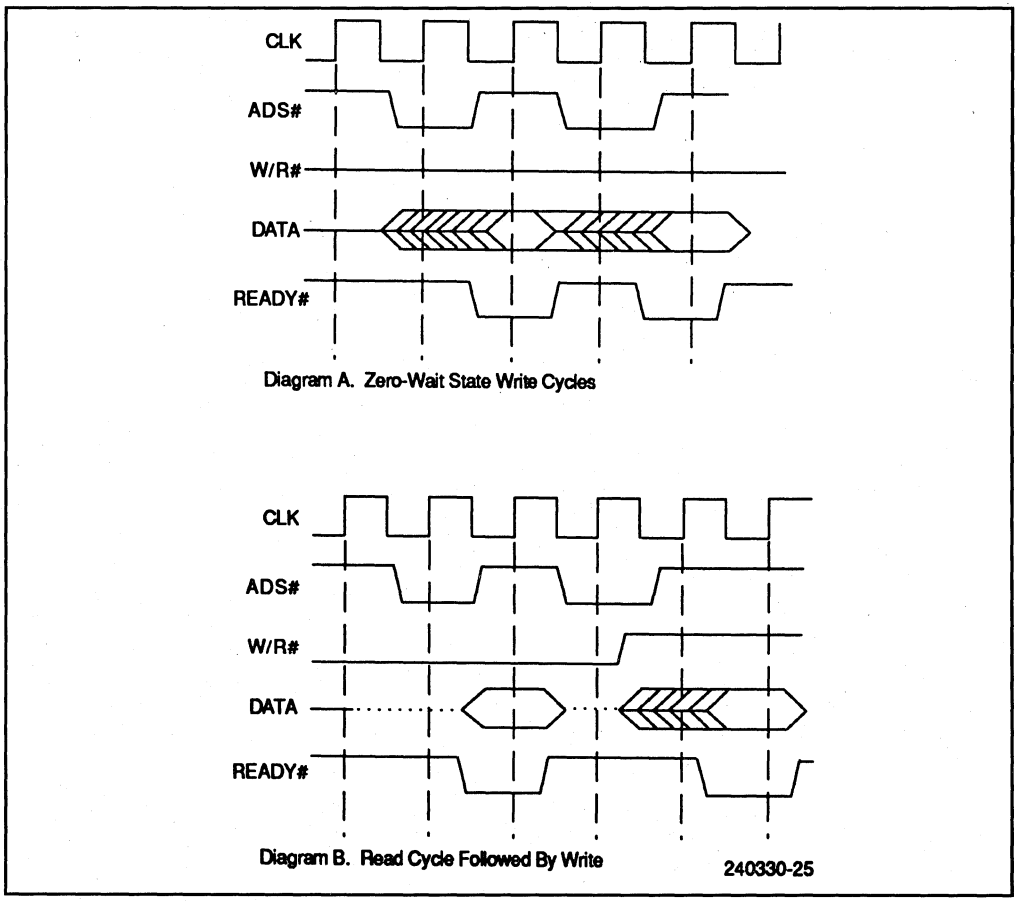


Figure 4-3. Read - Write Timing

Diagram A shows two successive write cycles. Diagram B shows a read cycle followed by a write cycle. The write data in Diagram B is driven one clock later than it is in Diagram A. This feature allows data bus transceivers to tristate the processor side of the bus before the CPU drives the write data.

4.3 DRAM SUBSYSTEM OVERVIEW

The features just described help make the DRAM subsystem more efficient. These functions are included in the processor to save logic needed to implement the DRAM controller. Timing constraints have also been eliminated, allowing fewer clocks per bus cycle.

The DRAM subsystem example described in this chapter illustrates how to use these features and describes the interaction of the processor with the memory system. The subsystem has been designed to use a 33.33 MHz clock. The example uses one Mbit memories which are organized 256K × 4. This memory is selected because of its minimum configuration. With a 64-bit bus, the minimum memory configuration is 2 Mbytes (16 devices). Using 1M × 1 memory devices, the minimum memory configuration is 8 Mbytes (64 devices).

Figure 4-4 is a block diagram of the memory subsystem example. It comprises five parts: address path logic, data path logic, parity logic, controller and DRAMs.

4.3.1 Address Path Logic

Address path logic performs several functions. Its primary function is to drive the processor address to the DRAM bank. To perform properly, it requires two paths for the row and column address.

In this example, the row address consists of processor address bits 12 through 20. These bits are buffered by the 74BCT29827 buffer. The buffer also disables the row address when the column address is selected. The device's OE pin is used for this function.

The output enable of the column address logic is also used for this function, but the column address must be latched for pipelining. For this reason, an AS821 register with output enable function is used.

The register and buffer chosen to implement the address path logic are 10 bits wide. The memory uses nine of these bits. The remaining bit drives the parity DRAMs. The parity logic is described later.

Outputs of row and column address devices drive every DRAM. Restrictions on the drive capabilities of these devices require additional drivers every 4 Mbytes. Two sets of address logic devices are shown in Figure 4-4. This configuration allows up to eight Mbytes of memory. Additional drivers can easily be added for larger configurations.

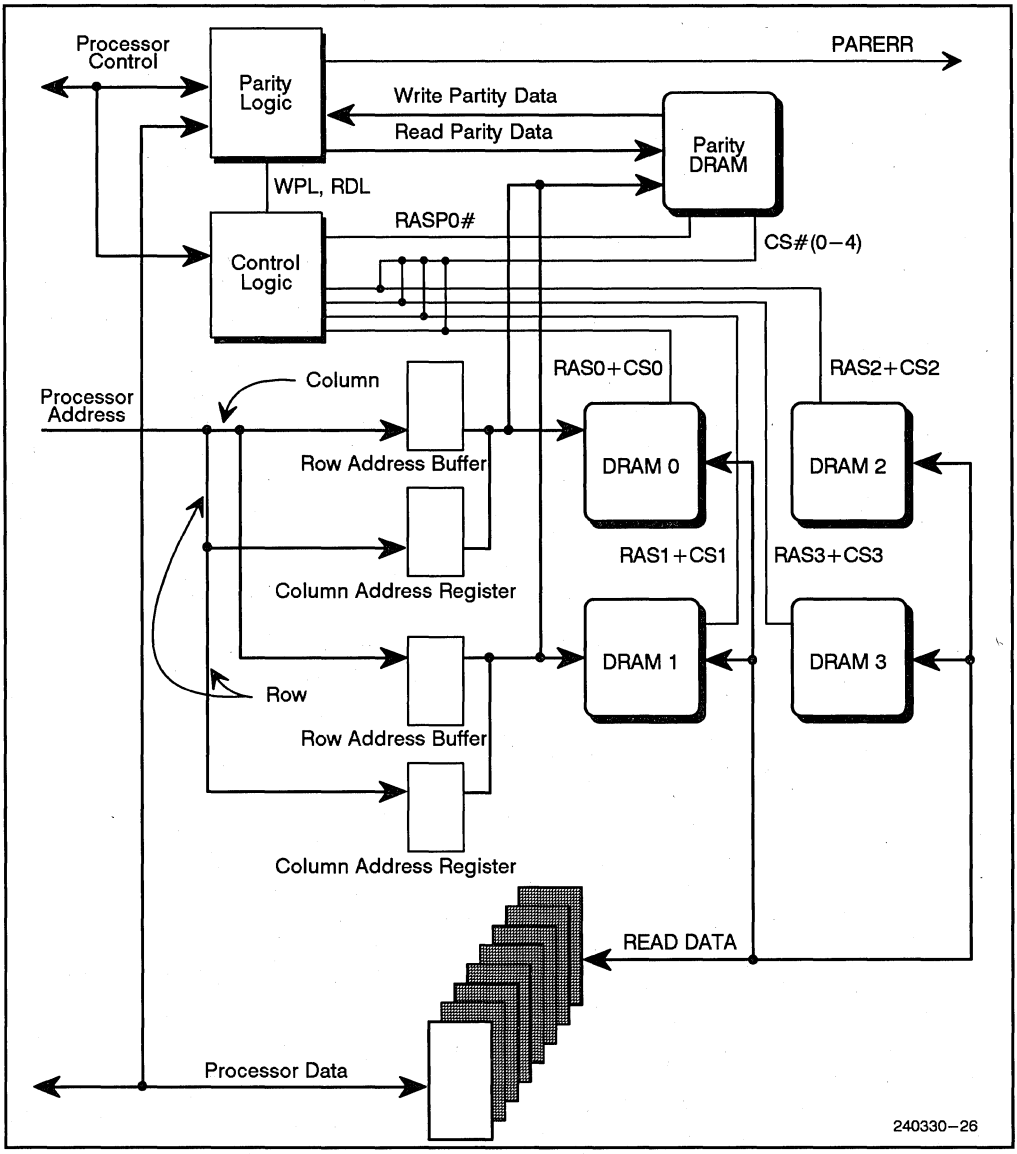


Figure 4-4. Maximum Example Configuration

4.3.2 Data Path Logic

The data path consists of eight 74AS646 bidirectional latching transceivers. These devices fulfill two critical memory system requirements. These transceivers hold read data

when the processor performs pipelined read cycles. Data for two reads has been accessed when the processor begins the third read cycle in a pipelined sequence. The data registers hold the first cycle's data while the second cycle's data is held at the transceiver input.

The register feature of the transceivers is also used during write cycles. The memory system posts all write cycles. During a posted write, NRDY# is returned to the processor before data is written to the DRAM. Without posting, write cycles would require at least two wait-states. Registers in the data transceivers hold write data after NRDY# has been activated. Section 4.4.4 provides a detailed description of the posted write function.

4.3.3 Parity Logic

Parity logic generates parity information and checks for parity errors. Eight 74AS280 9-bit parity generator/checker devices are used.

One parity bit is generated by each device. These bits are either written to the parity DRAMs or used by logic to generate a parity error signal. The parity error (PARERR) signal generates an interrupt which cannot be processed in time to stop the current bus cycle. The bus cycle which caused the error cannot be restarted.

The parity DRAM devices are 1 Mbit chips organized as 1M × 1. These DRAMs are used to store parity information for all eight Mbytes of main memory. The tenth address bit mentioned in Section 3.1 supports these devices. They divide the parity DRAMs into four sections which correspond to appropriate banks of main memory.

Odd-parity was chosen because of implementation restrictions. If even-parity is used, "I" inputs of the parity generators must be pulled down. Pull-ups are preferable in light of noise immunity.

4.3.4 Control Logic

The controller circuit contains the bus tracking state machine which is implemented with one PLD. The output signals of the PLD control three other PLD devices, each of which implements a specific control function.

The state machine PLD generates five state-variable outputs. These outputs define 32 states; designated sequences of these states implement various bus cycles. The state variable signals are decoded by the other control PLDs to create control signals. These signals, such as RAS0#, CAE#, and WEL, are activated to control the DRAMs, addresses and data logic.

Because of timing restraints, the state machine PLD (DSTAT in Appendix C) generates control signals NA# and RDY#. These constraints will be explained in the functional description (Section 4.3).

The DSTAT PLD also generates the BSY# signal. It serves as an arbiter between different system units, allocating processor bus priority to other subsystems that support bus pipelining.

The Control-A PLD generates row address strobe (RAS#) signals for all DRAM banks. Address bits are decoded along with state variable outputs of the DSTAT PLD to generate these signals. Each signal drives every DRAM in a 2 Mbyte array. The PLD is not registered because it primarily performs a decode function.

The RAS# signals select the active DRAM bank. Address inputs of this PLD are A21-A23. Figure 4-5 shows the address map for four banks of memory. The program for this PLD is listed in Appendix C. It shows the states decoded for the write, read and refresh functions. This PLD can be duplicated if a larger main memory is required, in which case address decode must be modified to select more memory banks.

This PLD also generates RAS# for the parity DRAMs. RASP0# enables the 1M × 1 DRAMs containing parity information for 8 Mbytes of main memory. Decode of this signal requires fewer address inputs.

Together, the Control-B and Control-C PLDs generate all address and data path control signals. They both have as inputs the state variables from the DSTAT PLD. Control-B also operates from a delayed clock, DCLK.

The delayed clock allows the Control-B PLD to sample state variables in the clock they are generated. Control-B outputs can be generated in the same clock.

The Control-B PLD controls the column address path. The PLD also generates the column address enable (CAE#) signal, which determines data access time, and the CSX# signal, which generates CS# signals to the DRAMs. Because the early write function is used, CSX# determines when data is written to the DRAMs. Control-B also generates signals which control the transceiver registers. The remaining transceiver control signals are generated by Control-C.

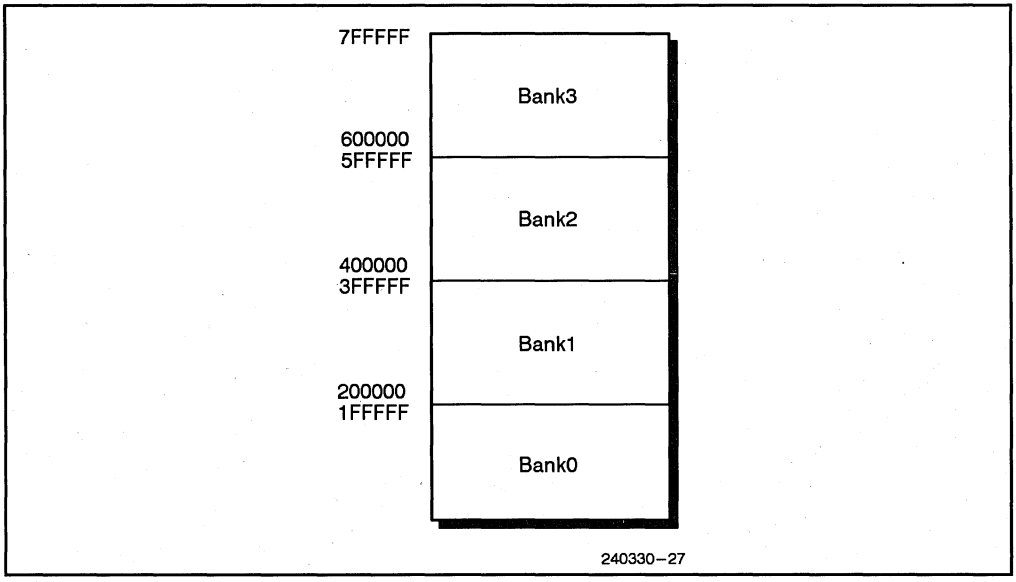


Figure 4-5. Address Map

Control-C also generates parity logic control signals. These signals latch parity data during writes and error indication during reads. Another important Control-C signal is RAE# which controls the row address path. Control-C also generates OEX# which generates output enable signals for the DRAMs.

The Control-D PLD does not use state variable inputs. Its only function is to generate write enable signals for the DRAMs. The WEL signal from Control-B is the clock input for this PLD. The signal is generated at the beginning and end of write cycles. By monitoring LADS# and W/R#, this PLD can determine the type of cycle being run when WEL is activated. The WE# signals are activated during writes and deactivated during reads.

4.4 DRAM SUBSYSTEM FUNCTION

This section defines the function of DRAM subsystem bus cycles.

4.4.1 Signal Description

The following list describes the purpose of all signals generated for the DRAM subsystem. Subsequently, the signals will be implemented in a functional discussion.

4.4.1.1 PROCESSOR INTERFACE

NRDY#	NRDY# is directly connected to the processors READY# input. It signals termination of a bus cycle and can be tristated.
NNA#	NNA# is connected to the processors NA# input. It is activated to enable pipelining and can also be tristated.
KEN#	KEN# indicates when caching can be enabled. It generates a cache line fill when activated in the same CLK as NA# and RDY#.

4.4.1.2 DATA PATH LOGIC CONTROL

RAE#	RAE# enables the row address drivers. It disables the row address before column address is driven.
CAE#	CAE# enables the column address register outputs. It disables the column address when the row address is driven.
CAL#	CAL# drives the clock input of the column address registers.
RDL	RDL activates the transceiver registers during read cycles.

WDL	WDL activates the transceiver registers during write cycles.
DRMDIR	DRMDIR controls the direction of the data transcievers.
DRMEN#	DRMEN# controls the time at which data is driven the the CPU or memory.

4.4.1.3 ADDRESS PATH LOGIC CONTROL

CSX#	CSX# generates CS# signals for four banks of memory and is generated from the rising edge of DCLK.
CS#0–3	These signals are buffered versions of CSX#. Each signal drives one bank of memory.
OEX#	OEX# generates OE# signals for four banks of memory and is generated from the rising edge of CLK.
OE#0–3	These signals are buffered versions of OEX#. Each signal drives one bank of memory.
RAS#0–3	These signals are decoded separately. Each enables access to a specific memory bank. They are decoded from address and state variable outputs.
RASP0#	RASP0# is generated during any access to the first four DRAM banks DRAMs. It latches the row address in the parity DRAMs.
WE0-WE7	These signals are generated by the Control-D PLD. Each signal corresponds to a single byte in each DRAM Bank.

4.4.1.4 CONTROLLER SIGNALS

WEL	WEL is connected to the clock input of the Control-D PLD. It latches the BE#0–7 inputs and is activated only during write cycles.
LADS#	LADS# is used by the state machine to determine when a bus cycle has started. It also indicates that the processor address is valid.
CPEN#	CPEN# indicates that a bus cycle has started and is waiting to be completed. It signals to the state machine that LADS# has been active and that a bus cycle is pending.
DCLK	DCLK drives the clock input of the Control-B pal. It is produced by delaying CLK 20ns.
S0-S4	These signals are the state variable outputs of the state machine. They are decoded by other control PLDs to produce control signals.

BSY#	BSY# is used to prevent other subsystems from driving NA#, READY# or the data bus before the DRAM subsystem has completed pipelined bus cycles.
DRAMSEL	DRAMSEL is a decode output which indicates that the current bus cycle is a DRAM access.
CLRCYC	This signal clears the cycle pending register.

4.4.2 Basic Read Cycle

The basic read cycle is shown in Figure 4-6. This cycle is performed when the processor requests a single read while the bus is idle. To simplify discussion, pipelining is not used in this example, so the NA# signal is not shown.

The read cycle begins when the CPU activates ADS#. ADS# is latched by controller logic and held for one clock. ADS# is latched with discrete logic because of processor bus timing constraints. Note that the processor address is not valid in this clock.

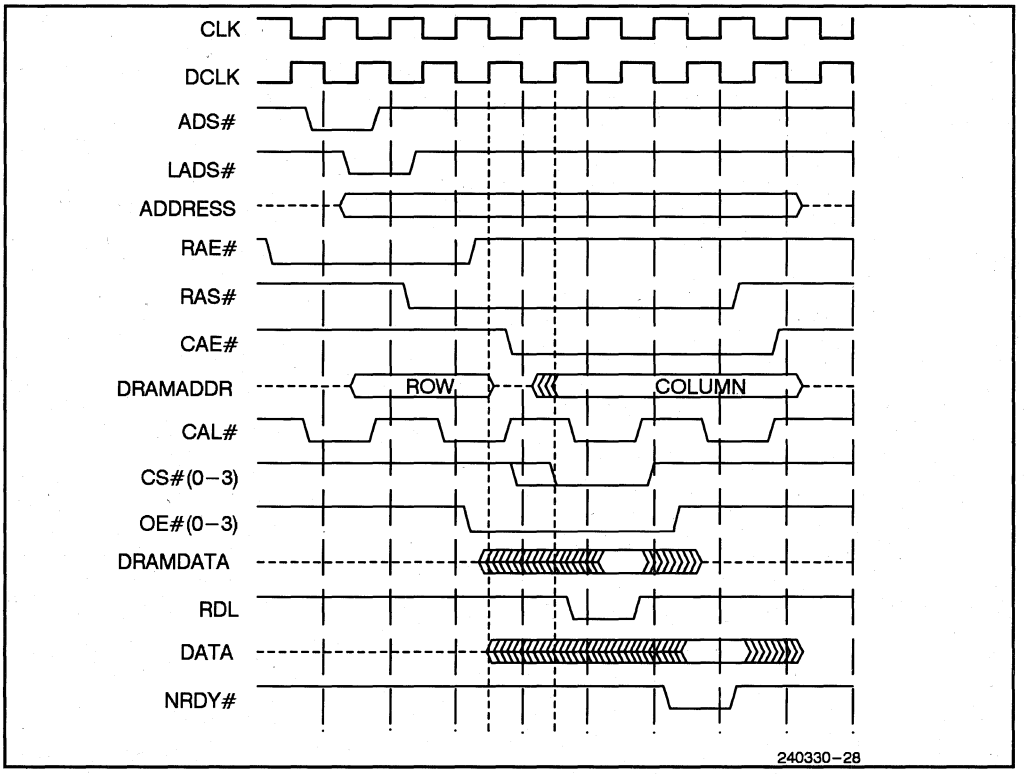


Figure 4-6. Basic Read Cycles

The LADS# register output is sampled active at the next clock edge. The trailing edge of LADS# activates CPEN# if DRAMSEL is active. The row address from the processor becomes valid in this clock. Since RAE# is already active, the row address is driven to the DRAMs and is valid in the following clock. RAS# is driven active following this CLK edge.

The falling edge of RAS# latches the row address in the DRAMs. The row address drivers must be tristated at this point, but not until the row address hold time is met. For this reason, RAE# is not deactivated until one clock after RAS# is activated.

At this point, the column address must be driven as soon as possible. The address valid delay adds directly to data valid time, but row address buffers must be tristated before CAE# is activated. The delayed clock, DCLK, allows CAE# to be activated in the same clock in which RAE# is deactivated. In this way, CAE# is activated in the minimum time possible.

CAL# latches the column address for pipelined reads in the same clock in which CAE# is activated. CSX# is also activated in this clock and is buffered to produce four CS# signals. These signals enable DRAM data buffers for the read cycle while the column address stabilizes.

Data becomes valid two clocks after CAE# is activated. Read data is not valid long enough to meet hold time requirements of CLK or DCLK and must be latched and synchronized to CLK. This function is performed by the RDL signal.

RDL enables the data transciever registers. It is driven from the same PLD as CAE# and is exactly in synch with CAE#. RDL is activated two DCLKs after CAE#, however, to ensure it is active in time to latch the read data.

NRDY# is generated in the next clock. The processor samples read data held in the transceivers at this point. If no other cycle is pending, RAS# and CS# are deactivated in the next clock. The cycle ends when the state machine has waited three clocks and met RAS precharge time.

4.4.3 Pipelined Read Cycles

The basic read cycle just described only occurs if the processor generates a single read cycle to a noncacheable address. Since most read cycles are cacheable, they occur as cache block fills.

A cache block fill generates four consecutive read cycles. Using the pipelining feature of the bus, these cycles may be overlapped as shown in Figure 4-7. Through pipelining, addresses of subsequent read cycles are driven before the previous cycle completes. As a result, the latency of the last three cycles of a fill block is reduced. A cache block fill begins as a basic read cycle. It is converted to a cache block fill by the KEN# signal, which is sampled active when NA# or RDY# is active. Three additional read cycles occur subsequently to fetch the entire cache block, and these cycles can be pipelined.

NA# controls bus pipelining and is generated by DRAM control logic. NA# is driven active by logic during any read cycle. If another bus cycle is pending, the processor activates ADS# in the following clock. Once this bus cycle has started, NA# is activated a second time. The processor then begins the next cycle by issuing ADS#.

Up to three bus cycles may be pending at one time. NRDY# must be returned for the first read cycle before NA# can be activated a third time. This function must be enforced by the DRAM control logic. The state machine PLD is programmed to activate NA# at the appropriate time. During the first read, it activates NA# three clocks prior to the clock in which NRDY# is active. It activates NA# again one clock before the clock in which NRDY# is active. Figure 4-7 shows signal timing for this bus cycle sequence.

Column address and data registers are needed to implement this function. As seen in Figure 4-7, the column address path remains enabled throughout the sequence. CAE#, RAS# and CSX# remain active from the first read. An address driven by the processor in response to NA# must be latched; otherwise, the column address changes before

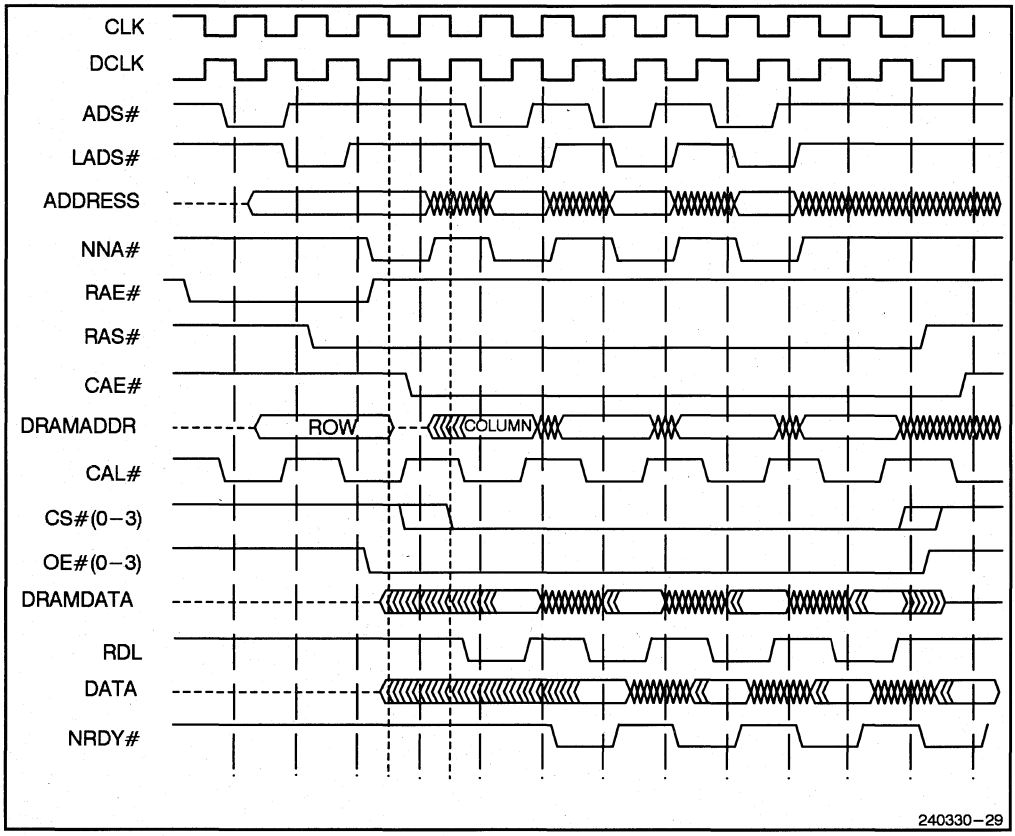


Figure 4-7. Pipelined Read Cycles

DRAM read data is valid. Once the data becomes valid, it must also be latched. Without data registers, the column address would be held for an additional DCLK to allow the processor to sample the data. This requirement would add an additional wait-state.

NA# is activated again once NRDY# is returned to the processor for the first read cycle. As in the previous cycles, ADS# begins the fourth read one clock later. At this time, NRDY# is returned to the processor, completing the second read cycle.

At this point, NA# is activated a fourth time. If another bus cycle is pending, ADS# will be activated. Otherwise, DRAM logic completes the next two bus cycles. If no bus cycles have been started, it then deactivates RAS# and CSX#.

Another bus cycle may begin before the final bus cycles have completed, but not on the clock after which NA# is activated. RAS# and CSX# signals remain active in this case.

Pipelined sequences usually occur as described here. These cycles must all occur within the same DRAM page, however. If the row address changes, RAS# must be deactivated, and the pipelined sequence is interrupted. Read cycles generated for a cache block fill are always within the same page, and the sequence shown in Figure 4-7 illustrates this case. Note that all cycles of subsequent cache block fills can be pipelined.

4.4.4 Basic Write Cycle

In this example, pipelining is not used with write cycles. A method called posting is employed instead.

Posting is similar to pipelining because it improves performance by allowing consecutive bus cycles to overlap. In posting, however, NRDY# is returned to the processor before a cycle is completed at the DRAM. This function is better suited to write cycles because data is not returned from the DRAMs. The processor may complete a cycle and begin another while the DRAM subsystem completes the write cycle.

A write data register is needed to implement posted write cycles. This register holds write data after NRDY# is returned to the processor. In the example, the registers are contained in the data transceivers. This register is controlled by WDL.

The row address function of read and write cycles are the same. Figure 4-8 illustrates the function of a write cycle, which begins from an idle bus state. LADS# starts the cycle one CLK after ADS# is activated. The row address propagates to the DRAM, and RAS# is activated. As with read cycles, CAE# is activated one clock later.

CAL# is activated in this clock to latch the column address. The write data latch signal (WDL) is also activated. It is connected to the transciever clock input. Here, data is latched and driven to memories.

CS#-controlled (early) writes are used in this example. When writes are performed in this way, data must be valid before CS# is activated. The WE# DRAM inputs must also be active prior to the falling edge of CS#.

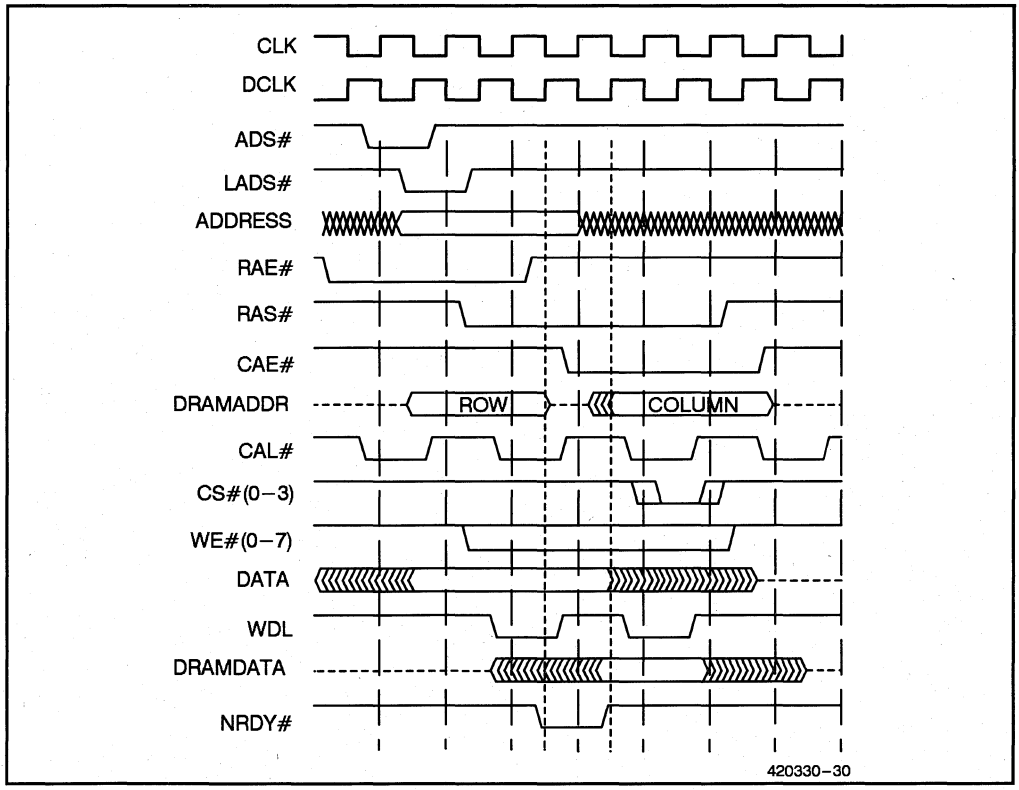


Figure 4-8. Basic Write Cycles

CS# is activated one clock after NRDY# is returned to the processor. The write enable signals are driven in the clock after LADS# is activated. Data becomes valid in the same clock that CS# is activated. If another bus cycle is pending, ADS# is active at the next clock edge.

4.4.4.1 CONSECUTIVE WRITE CYCLES

Posting is most beneficial when write cycles are consecutive. Figure 4-9 shows three consecutive write cycles. The first is identical to that shown in Figure 4-8. The addresses of the following two cycles are in the same DRAM page as the first. These cycles can be completed without wait-states on the processor's bus. Timing becomes more critical at zero wait-states. The WE# signals, for example, are no longer valid three clocks before CS#. They are activated in the same clock as CS#. In addition, write data is only valid in the last clock of the bus cycle.

4.4.5 Consecutive Bus Cycles

Additional wait-states are needed when different types of bus cycles occur consecutively. These are added to the second cycle of the sequence and allow the first bus cycle to complete after the second cycle begins.

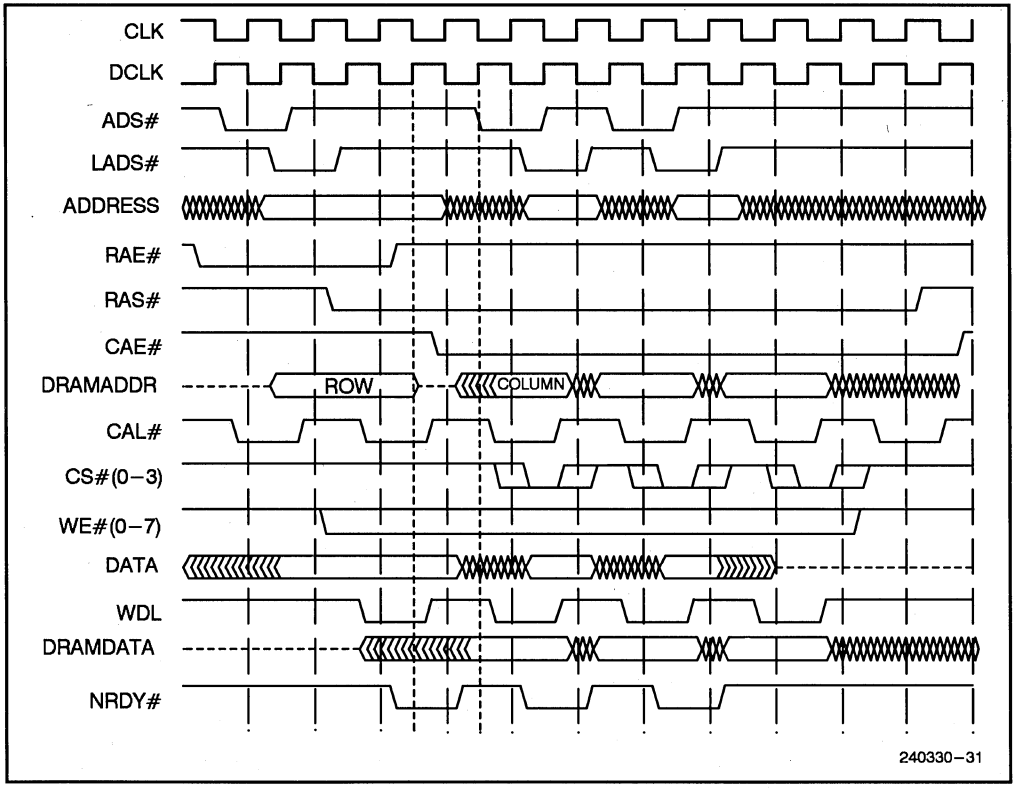


Figure 4-9. Consecutive Write Cycles

The state machine performs this function. It tracks the current bus sequence. If a read cycle immediately follows a write, or if the opposite occurs, the controller adds the needed number of wait-states. Bus cycles proceed normally but are delayed to allow the previous bus cycle to complete and to reverse the direction of the data bus.

These functions are different for read and write cycles. The specific sequence for each is described in the following sections.

4.4.5.1 WRITE FOLLOWED BY READ CYCLES

Figure 4-10 shows a write cycle followed by a read cycle. This sequence occurs when a read cycle is pending before a write cycle has completed. A timing conflict would occur if the read is to the same DRAM page as the write.

The state machine enters a special sequence to handle this event. It begins as if a normal write cycle had occurred. After it returns NRDY# for the write cycle, however, the processor asserts ADS#. This signal starts the read cycle.

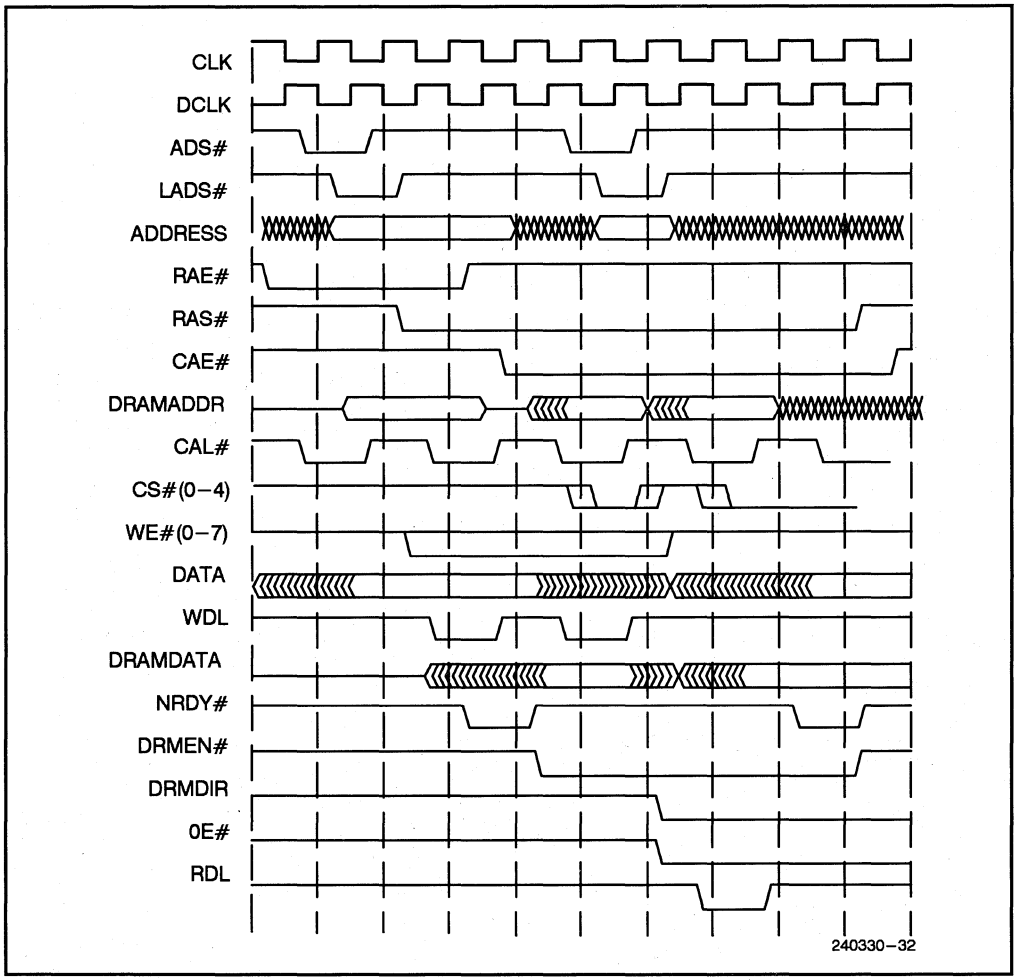


Figure 4-10. Write Followed by Read Cycles

In the first clock, data is written to the DRAM. Next, WDL is deactivated to prepare for another write cycle. Because the state machine has no way to determine the next cycle type, it prepares for the next write.

The process continues into the next clock. Here, CSX# is also deactivated to prepare for the next write. The write enables are still active, and WDL is activated. Control and address signals for the read become valid in this CLK, and the state machine samples them at the next CLK edge.

Once the state machine determines that a read cycle has started, it enters a special state sequence. It activates the OE# signal and activates CSX# to enable the DRAM for a read cycle. The Control-D PLD samples W/R# low and deactivates the WE# signals.

At this point, the DRAM controller has added one extra wait-state to the normal read process. It adds another for the next clock to allow data to propagate to the latching transcievers. RDL is then activated to latch the read data, and the cycle completes normally.

4.4.5.2 READ CYCLES FOLLOWED BY WRITE CYCLES

Figure 4-11 illustrates a read cycle followed by a write cycle. Both cycles access the same DRAM. The write cycle begins while the read cycle is in progress.

Pipelined reads add a level of complexity to the problem. When NA# is activated, a write cycle can start. This cycle can start before NRDY# is returned to the processor.

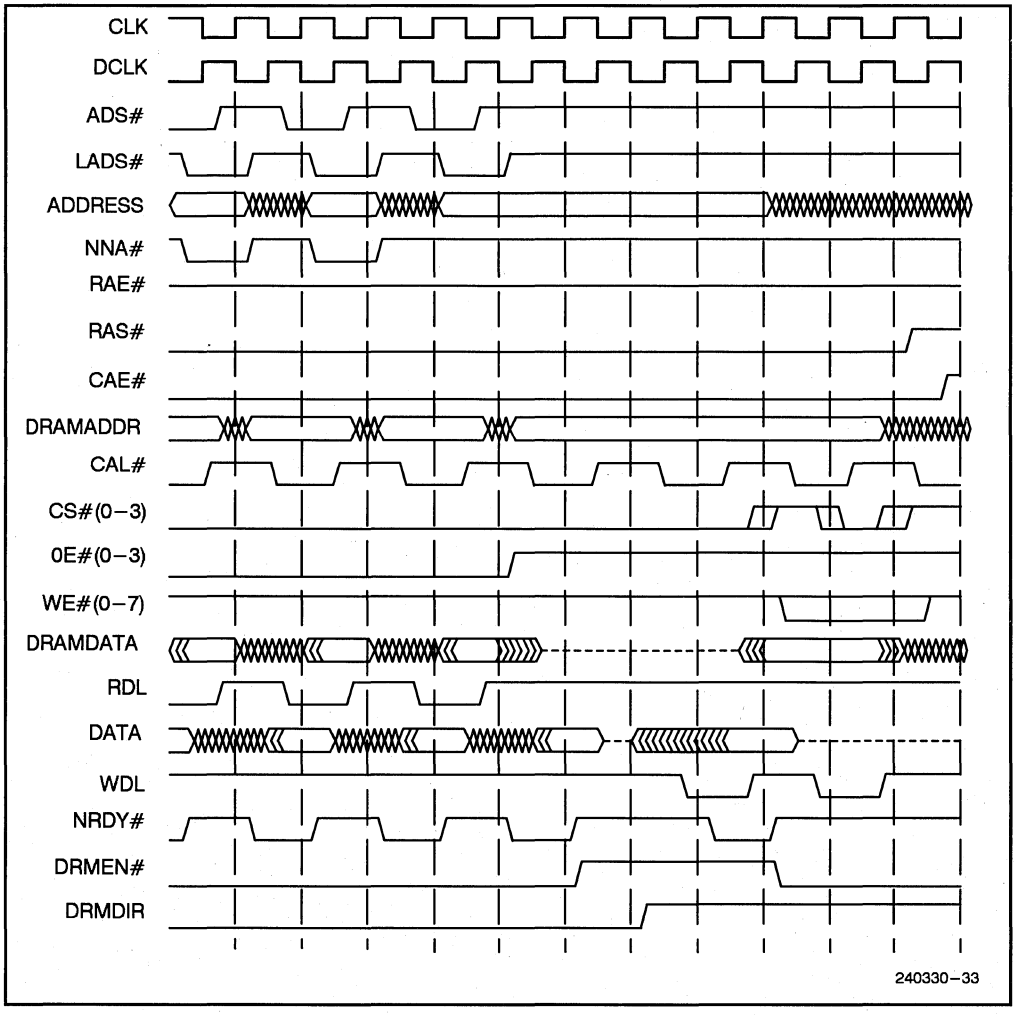


Figure 4-11. Pipelined Read Followed by Write Cycles

Figure 4-11 illustrates a worst-case scenario in which two read cycles must complete after the write cycle has begun.

ADS# for the write cycle is active in the clock in which NRDY# is returned for the first of two read cycles. LADS# is active in the next clock. LADS# and W/R# indicate to the state machine that the cycle is a write. The state machine then enters a two clock sequence to complete the second read.

The cycle pending signal is important in this sequence. LADS# is not active when the second read is finished. The CPEN# signal is the only indication that that another cycle has started. CPEN# is sampled by the state machine in the clock that NRDY# is active for the last read. In response, the state machine enters a special state sequence. Here, it performs the functions necessary to prepare for a write cycle.

Two clocks after NRDY# completes the last read, OE# is deactivated. In the next clock, WDL latches the write data. NRDY# is then returned to complete the write cycle. CSX# is deactivated in this CLK so that data can be written in the next clock. At this point, the cycle continues as described in Section 3.4.

4.4.6 Page Miss Cycles

A page miss cycle is a memory access which changes the row address. This type of access is important in modern memory systems using page mode or static column mode DRAMs. These DRAMs have an internal register which holds the row address. Access can be made by simply changing the column address. This feature reduces the average access time by several clocks.

The DRAM controller must be designed to take advantage of this feature. In doing so, memory system performance is improved dramatically. To use this feature, control logic need only activate RAS when the row address changes.

The NENE# signal indicates when the row address has changed. It is available along with the address during any bus cycle. The state machine samples this pin at the beginning of every bus cycle.

When NENE# is sampled inactive, the state machine enters a special state sequence. Here, it performs the functions needed to latch the new row address. Figure 4-12 shows a typical page miss sequence.

The RAS# signal is immediately deactivated when a page miss is detected and is held inactive for four clocks. The time RAS# is held inactive is determined by the RAS# precharge time. This timing is a requirement of the DRAMs and varies depending on the manufacturer and the speed selection used.

CAE# is deactivated in the same clock as RAS#. The column address register outputs must be tristated before driving the row address. RAE# is activated two clocks later. At this time, the new row address is asserted at the DRAM address inputs. The cycle can then be completed as described earlier.

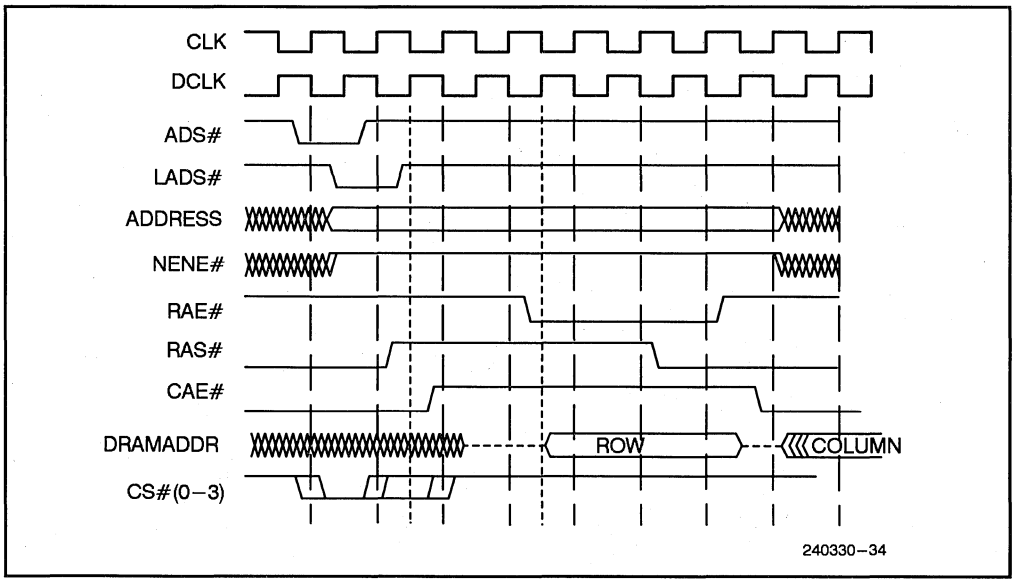


Figure 4-12. Page Miss Cycles

4.4.7 Refresh Cycles

Refresh cycles must be performed at regular intervals of approximately 15 microseconds. Each cycle refreshes one row of DRAM data.

Most DRAMs provide their own refresh address. This address is maintained in an internal counter and is updated if CS# is activated before RAS#. The address from the counter also refreshes the memory array. This function is called CS# before RAS# refresh.

Although not illustrated in this example, the refresh address can be provided by an external counter. This method is called RAS#-only refresh.

The RFRQ signal is generated by another counter and indicates that a refresh cycle must be performed. The counter is set to a count which ensures the proper refresh interval.

The RFRQ signal is sampled by the state machine during any idle cycle and at the end of every bus cycle. If the signal is active, the state machine enters a page miss cycle state sequence. Once the RAS# precharge time is satisfied, the state machine enters the refresh sequence.

A new row address must be latched after a refresh cycle is performed. After RAS# is activated for the refresh cycle, it is again deactivated. Once the precharge time has been met, RAS# can be activated to complete a pending bus cycle.

CPEN# is the only indication that a bus cycle has begun during the refresh cycle. If this signal is active after the second RAS# precharge, a bus cycle begins immediately. Only one bus cycle may be started during the refresh sequence.

4.5 PARITY CIRCUIT

Parity logic provides error detection capability for each byte of data. One bit is stored in the parity memory for each data byte. The value of this bit depends on the number of bits set in the corresponding byte. The parity logic is designed for odd parity. If a byte contains an even number of set bits, the parity bit is also set.

Parity data is generated during write cycles. Some delay is incurred, but because writes are posted, parity data is available in time to be latched by CS# signals. These signals drive parity DRAMs and data memory.

A separate RAS# signal is generated for the parity DRAMs. This signal must be activated for accesses to all four banks of memory. The Control-A PLD generates this signal during any cycle to the banks it controls.

Parity is checked during read cycles. The parity error signal is activated if a zero is detected on any of the parity outputs. Figure 4-13 is a block diagram of parity logic. It shows connections of the control and bus signals.

4.5.1 Dedicated Signals

The following is a summary of the signals that control parity logic.

RASP0#	RASP0# drives parity DRAM RAS# input.
PARERR	PARERR is activated when a parity error is detected. It is held in a register once activated.
PERLTCH	PERLTCH activates the register which samples PARERR during read cycles.
WPL	WPL activates the write parity data register.

4.5.2 Parity Function

Parity data is latched during zero-wait-state writes. It is latched in the clock after write data has been latched. The write data register is clocked by WDL in the same clock in which NRDY# is activated. WPL clocks the parity write data register as shown in Figure 4-13. WPL is activated from the rising edge of CLK. Since CSX# is activated from DCLK, parity data can propagate to the DRAMs before CSX# is active.

Parity data is generated by 74AS280 devices. The I inputs of these devices are not driven during write cycles but are pulled high to generate correct parity output. During read cycles, read parity data is driven to I inputs. Appendix C provides complete DRAM system schematics.

Read parity data is accessed in the same way that read data is accessed. The parity data register is clocked by RDL, and parity data is valid at the parity generators about the same time that read data is valid at the transceivers. The DRAMDIR signal enables output of this register. It disables the register outputs during write cycles.

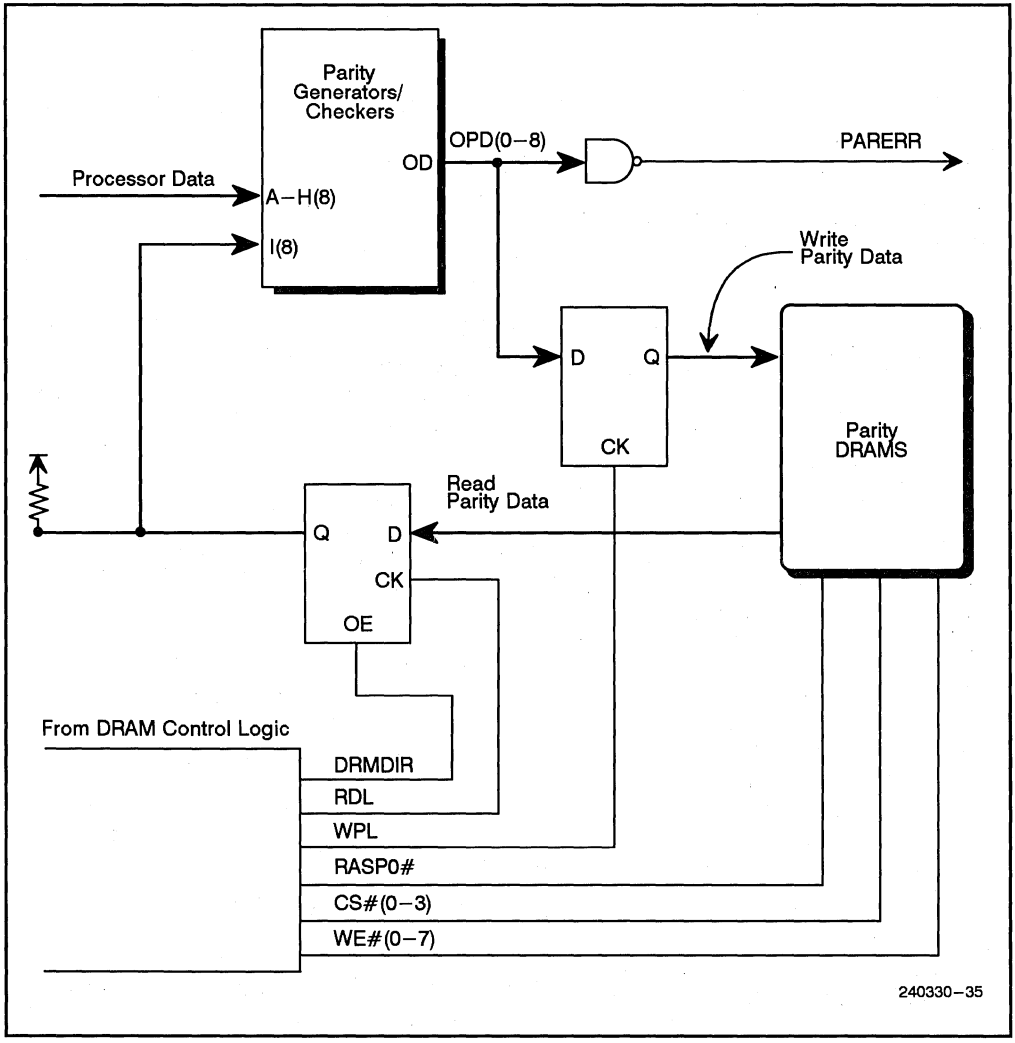


Figure 4-13. Parity Block Diagram

The PARERR signal indicates that a parity error has occurred. It is generated by a 74AS80 nand gate. The inputs of this gate are connected to the OD outputs of the parity generator. If any of these outputs is high during a read cycle, PARERR is activated.

Once it occurs, the PARERR signal is latched. The PERLATCH signal activates the PARERR register. It is active every read cycle. The PARERR register can be cleared in the read cycle following the cycle that caused the parity error.

I/O Interfacing 5

CHAPTER 5
I/O INTERFACING

5.1 OVERVIEW

I/O devices can be mapped anywhere within the i860™ microprocessor's four gigabyte physical address space and can be 64-, 32-, 16- or 8-bits wide. For accesses to 8-, 16- or 32-bit devices, byte-swap logic or 8-, 16- or 32-bit load/store instructions must be used. Although address pipelining can be used for all i860 microprocessor accesses, I/O recovery time and infrequent back-to-back I/O accesses greatly reduce pipelining benefits. The i860 microprocessor does not include I/O instructions, and there is no separate address map. All I/O devices must be mapped into the memory address space. The system distinguishes between memory and I/O accesses by decoding processor addresses. Although all reads can be cached, the I/O reads must not be cacheable and should deassert KEN# to indicate a non-cacheable access.

I/O buffers accessed by DMA devices must be marked non-cacheable or flushed from the data cache before an I/O operation. Two different approaches to I/O are described later.

5.1.1 i860™ Microprocessor I/O Subsystem

Figure 5-1 illustrates the I/O subsystem of an i860 microprocessor based system. The memory address and data are latched to allow pipelined operation. I/O addresses and data can be buffered if drive requirements exceed i860 microprocessor specifications. Processor addresses are decoded to determine whether access is to I/O or to a memory device. Decoder outputs notify control logic of cycles and indicate whether the cycles are to memory or to I/O. Read, write and chip select are generated by control logic. This logic also generates wait-states in i860 microprocessor cycles and generates READY# when control logic is ready to terminate the processor cycle. Interface to all slave devices except DRAMs is very similar and less performance sensitive. Nonvolatile memories such as ROMs and EPROMs are accessed through I/O control logic and share address and data paths with other slave I/O devices. The i860 microprocessor provides a CS/8 bootstrap mode for byte-wide ROMs used only during power-up. The mode is disabled once the system boots.

I/O devices may be mapped anywhere within the processor's four gigabyte physical address space. To minimize the decode logic, devices are placed within one contiguous block of the i860 microprocessor's address map. This minimizes the number of address lines needed to generate I/O select signals. If 8-, 16- or 32-bit I/O devices are used without byte swap logic, I/O ports must be placed at 8-byte increments (i.e addresses must correspond to the data pins the device is hardwired to). This causes the processor to read or write data on the proper bytes of the data bus. For example, an 8-bit access may get data on D7–0; a 16-bit access may get data on D15–0; a 32-bit access may get data on D31–0. Here, addresses are on 64-bit boundaries.

This limitation does not apply if external byte swap logic is used. 8-bit devices can be accessed on continuous byte boundaries; 16-bit devices can be accessed on continuous even byte boundaries; and 32-bit devices can be accessed on continuous, multiple-of-four

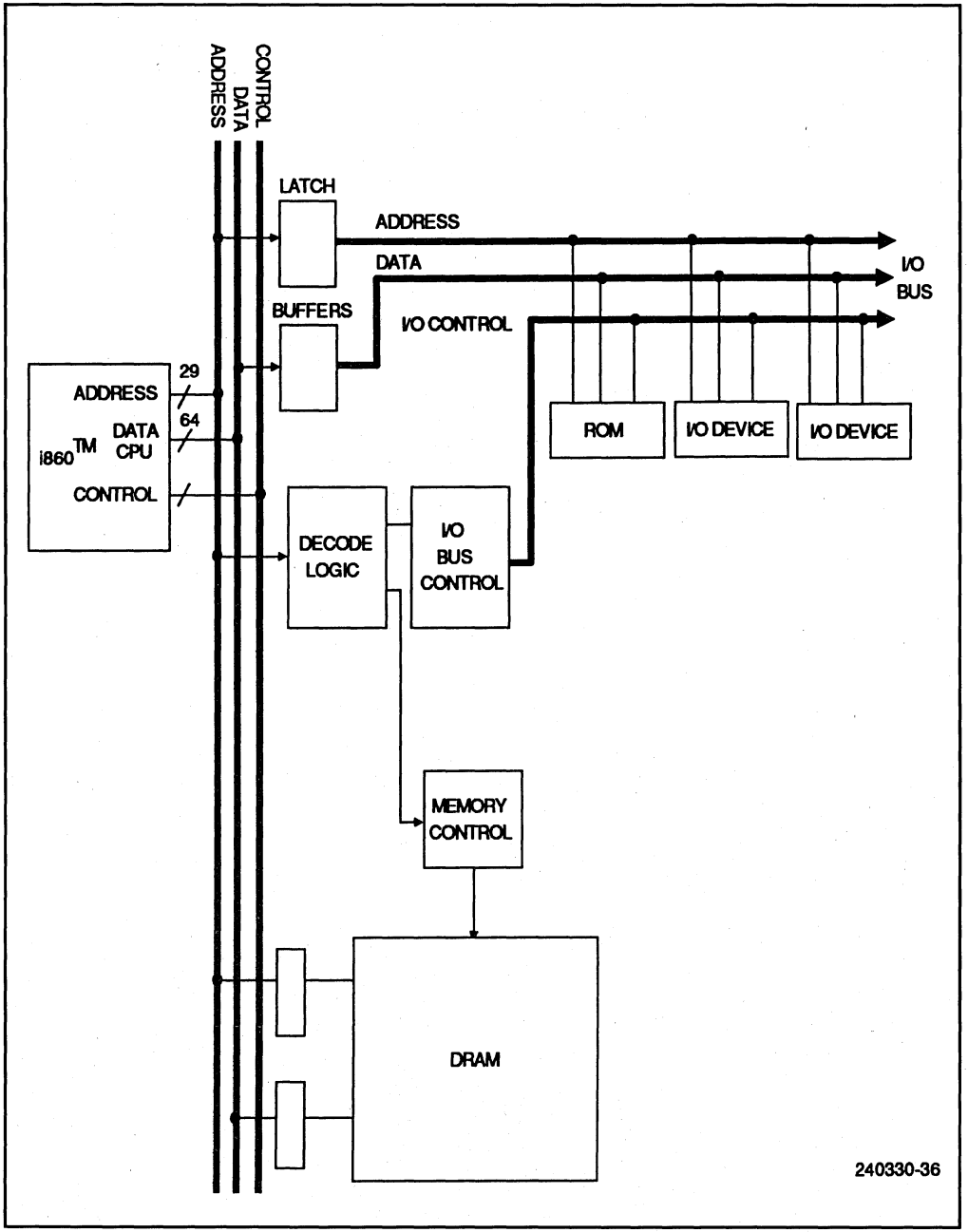


Figure 5-1. i860™ Microprocessor System

byte boundaries. If a 16-bit access is attempted on an odd byte boundary, or if a 32-bit access is attempted on an address boundary that is not a multiple of four, the i860 microprocessor generates an exception.

5.2 GENERATING I/O CONTROL SIGNALS

Control and chip select signals for slave I/O devices are generated by I/O control logic. The i860 microprocessor does not provide a separate I/O space, and the distinction between accesses to memory and to I/O devices is made by decoding addresses. The address decoder typically provides an IOSEL (I/O select) or MEMSEL (memory select) output to indicate the access type. When IOSEL is asserted, I/O control logic generates appropriate control signals, drives KEN# inactive to indicate a non-cacheable access, inserts any needed wait-states and returns READY# to the processor to terminate the cycle. I/O control logic must enable appropriate address and data buffers, and it must eliminate bus contention caused by data float delays. Most I/O devices require a recovery period between back-to-back accesses. This can be enforced either through hardware or software.

5.2.1 I/O Control Logic

Figure 5-2 illustrates i860 microprocessor control logic. I/O select signals are generated from the decode of unlatched addresses. The decoder also generates device select signals which are latched to provide chip selects for specific devices being accessed. ADS# does not ensure a valid address when asserted. LADS#, a latched version of ADS#, is valid in state T_{11} and can be used to generate an address latch enable (ALE) signal. Address is guaranteed to be valid on the clock (rising) edge during which LADS# is active.

When LADS# is active and I/O select is asserted, I/O control logic generates read or write signals based on the state of W/R# input. During pipelined operation, I/O control logic must know of any outstanding DRAM cycles to avoid misordered cycles. BSY indicates that there are outstanding DRAM cycles and that I/O control logic should not start a cycle until BSY is deasserted. I/O control logic inserts needed I/O cycle wait-states according to the number of wait-states required by the device being accessed. In a simple design, I/O accesses can be assigned the number of wait-states required by the slowest device.

Following read cycles, I/O devices may need time to turn off the data bus. During this period, a write or read from another device causes bus contention. To ensure proper operation, I/O devices also require recovery time between consecutive accesses. It may or may not be practical to enforce this recovery with a hardware counter. The advantage of using a hardware enforced recovery mechanism is transparency and reliability. Also, future upgrades to higher CPU clock speeds will not require any changes in the I/O device drivers. If the recovery period is too long to be enforced with hardware, then software timing loops or a timer chip may be used to ensure proper operation. The hardware I/O recovery time is enforced by the control logic. Recovery logic generates a recovery delay (RECOVER) signal which prevents I/O control logic from asserting read or write until the recovery delay signal is deasserted.

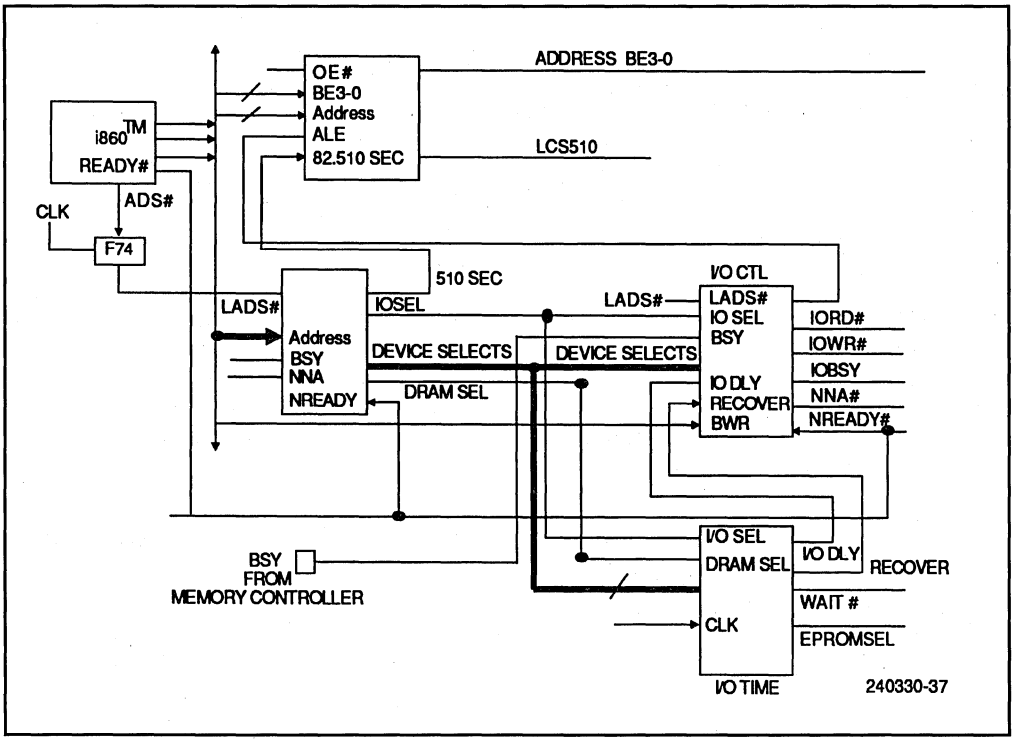


Figure 5-2. I/O Control Logic

5.2.2 Address Decode and Chip Select

Chip selects for various I/O subsystem devices are generated through address decoding. Decodes may be done with a PLD, such as the Intel 85C508, or with a simple one-of-four decoder such as the 74F138. Decodes can be done from latched or unlatched addresses. Unlatched address decodes allow device selects to be generated one propagation delay after an address becomes valid. Some I/O devices need a long chip-select hold time from read or write. This requires that chip selects be latched. Decodes can be done alternately from latched addresses, but this delays the chip select generation and may result in slower cycles. Device selects are also used by I/O control logic to determine the number of wait-states for a device being accessed. If device selects are generated from unlatched addresses, I/O control logic must sample the device select with LADS# to ensure a valid sample of the device select. Latched device selects must be deasserted (unlatched) once READY# is returned to the processor. They are unlatched by activating the latch enable signal which causes the latch to become transparent. The device select is deasserted when the address changes at the beginning of a new cycle.

I/O device addresses should simplify decode logic and reduce the number of address pins required to generate device selects and the I/O select signal. The I/O select signal indicates I/O device accesses to control logic. I/O devices are typically mapped with a contiguous single address space. Address space size is determined by the address lines used in decode.

5.2.3 IORD#/IOWR#

IORD# and IOWR# command signals are generated some time after the processor W/R# signal chip select. I/O devices have command-active width requirements, and wait-states are inserted in I/O cycles to accommodate these. The number of wait-states is determined by the I/O control logic of selected devices. Minimum command-active times range from 100ns to 400ns across various slave devices. I/O devices also have chip select set-up and hold time requirements. Chip select requirements are met by inserting delays in activating commands. If one delay is used across all devices, it must be the worst case delay. In the examples given here, the delay is two clocks. If necessary, hold time is met by latching chip selects and addresses.

It may be necessary to delay driving write data onto the bus for several clocks. This is necessary if the previous cycle was a read to an I/O device with a long data float delay. It may also be necessary to delay the assertion of READY# from the rising edge of IOWR# to ensure sufficient write data hold time.

5.2.4 READY#

Wait-states are introduced into the i860 CPU cycle by not asserting READY#. Since most I/O devices are much slower than the i860 processor bus, all I/O cycles require additional wait states. Since READY# can be driven by multiple PLDs, it must be tri-stated by the PLDs not in use. If several devices reside in the I/O subsystem, the wait-state logic can be simplified by inserting the same number of wait-states in all I/O accesses. This slows accesses to some of the faster I/O devices, but these accesses are infrequent, and impact on system performance is minimal. If I/O speed is critical, a small ROM or PLD can be used to insert wait-states and enforce recovery on a per device basis.

5.2.5 Recovery and Bus Contention

Most I/O devices require a recovery period between back-to-back accesses. At higher i860 CPU clock frequencies, bus contention poses an additional concern. This is because long float delays of the I/O devices can conflict with data driven out in the next cycle by another device or by write data from the CPU. All slave devices stop driving data on the bus on the rising edge of IORD#. Some delay after the rising edge of IORD# the data will float. If, however, another device drives data onto the bus before the data from the previous access floats, bus contention will occur. The i860 CPU has extremely small cycle times (30ns @33 MHz, 25ns @40 MHz), and the possibility of bus contention must be considered. The I/O control logic implements recovery to eliminate bus contention. It asserts the signal RECOVER, which inhibits new I/O cycles from starting until the data from the previous read has floated. I/O control logic can also accomplish I/O recovery using a similar mechanism. The only time hardware enforced I/O recovery may not be

possible is when recovery times are too great for the hardware counter. In this case, recovery time can be enforced with software using NOPs and delay loops or with a programmable timer.

5.3 I/O CYCLES

The I/O read and write cycle timings depend upon the implementation of the I/O control logic. Figures 5-3 and 5-4 illustrate the timings of the I/O read and write cycle for a typical implementation.

5.3.1 Read Cycle Timing

A new i860 microprocessor read cycle is initiated when ADS# is asserted in T_1. The address and status signals become valid in T_{11}. LADS# becomes valid in T_{11}, and the address is latched on the next rising edge of clock (second T_{11}). The I/O select signal is

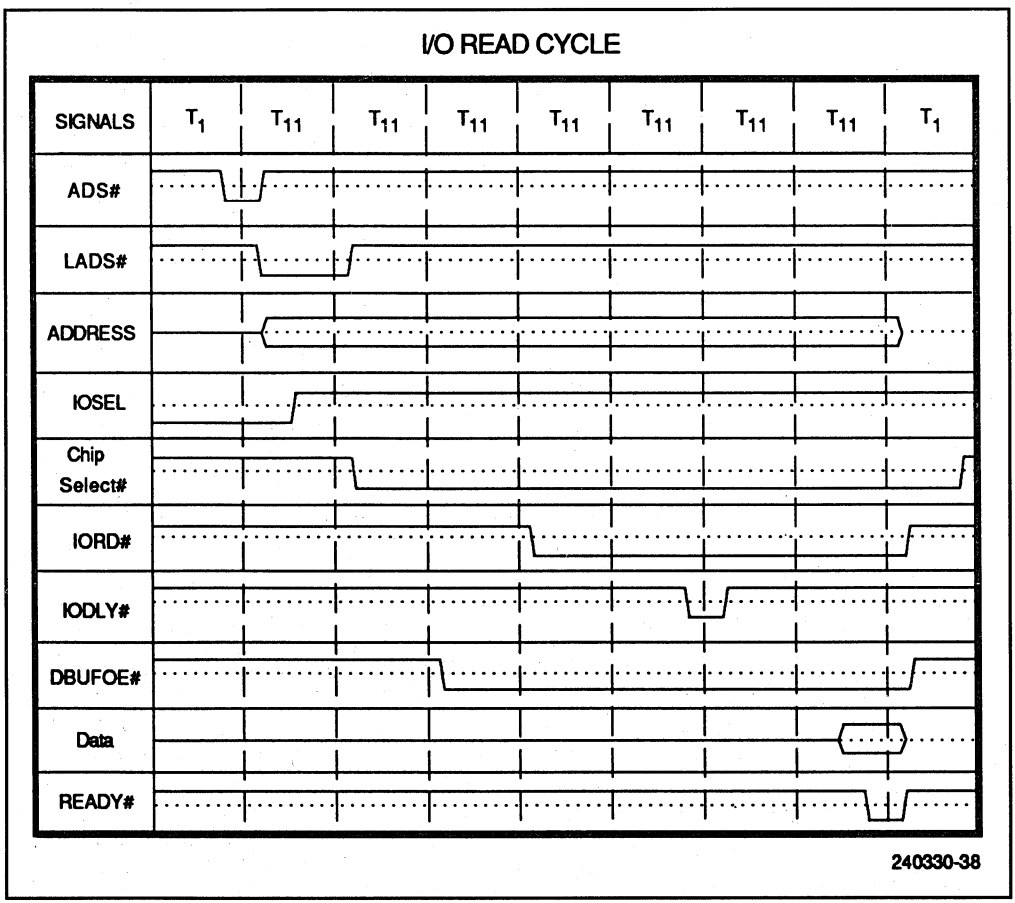


Figure 5-3. Read Cycle Timings

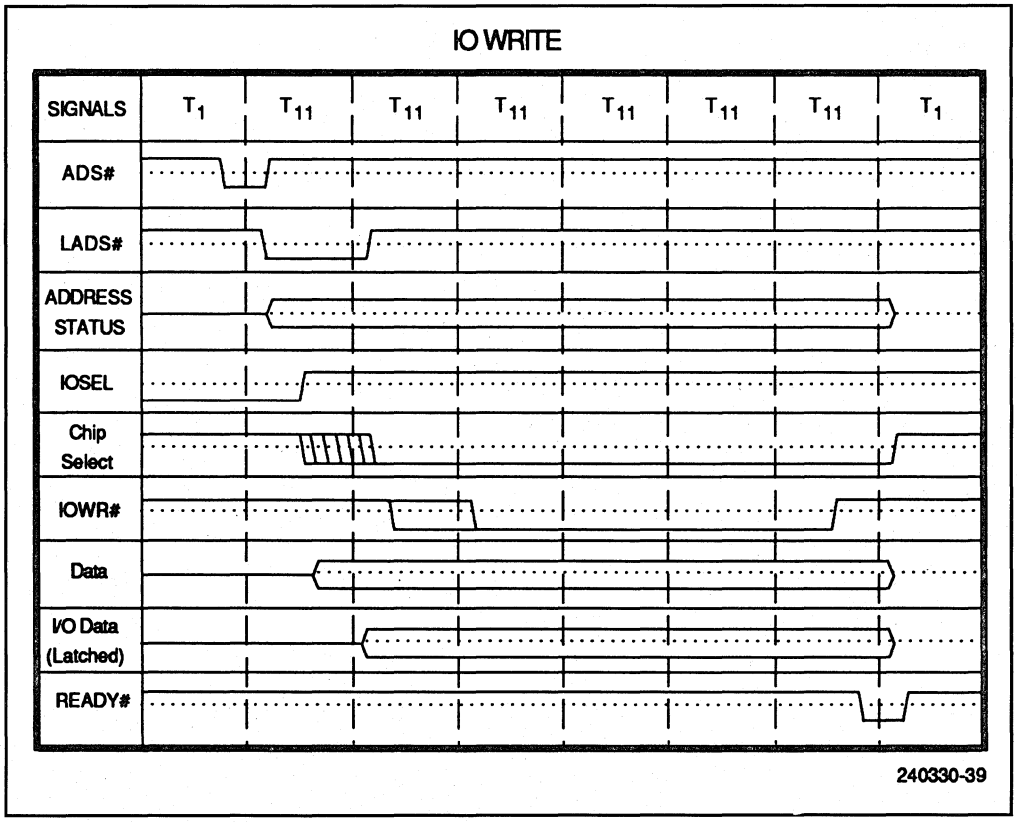


Figure 5-4. I/O Write Timings

generated from a combinatorial decode of the addresses. I/O select, when active with LADS#, indicates an I/O cycle to the control logic. The chip select is either asserted at the same time as I/O select or is latched in the clock following I/O select. The IORD# signal is asserted in the second T_{11} if RECOVER is inactive. RECOVER, if active, indicates that the new cycle must be delayed in order to meet the recovery time of the I/O device or to prevent data bus contention. If RECOVER is active, then the I/O read (IORD#) signal is not asserted until RECOVER is deasserted. I/O data becomes valid on the bus one read delay after IORD# is asserted. The bus control logic will need to keep IORD# active to meet the minimum active time requirements.

The worst-case timings values are calculated by assuming the maximum delay in the address latches and decode logic and the maximum delay through the data transceivers. These formulas will yield the fastest possible cycle. Wait-states must be added to meet the access times of the particular I/O device.

The critical timings for I/O Read cycles are the following:

1. Chip Select/Address Setup to IORD#:

   ≥ 3Tcy + PLDclk-to-output(min) − Decode delaymax
   − Address Valid delaymax − Latch Prop. Delaymax

   Chip select goes active later than I/O address and is latched, so chip select timing is used for the equation.

   Tcy : CLK PERIOD OF i860 Microprocessor. Address Valid Delay : Processor address valid delay in T_{11}.

2. IORD# Minimum Active Width:

   ≥ Tcy + PLDclk-to-output − PLDclk-to-output + *n*Tcy
   ≥ Tcy + *n*Tcy

   n is the number of wait states inserted into the I/O read cycle. *n*Tcy represents the additional time due to wait states.

3. Chip Select/Address Hold after IORD#:

   ≥ Tcy − PLDclk-to-output + Address Validmin + Latch Prop. Delaymin

   Assumes I/O address changes earliest, so hold time calculations are done using the I/O address timings.

4. IORD# Active to Data valid:

   ≥ Tcy − PLDclk-to-output − Data Setup Time − Data buffer Prop. delay.
   Data Setup Time: Processor setup time requirement for data to rising edge of clock.

   This assumes the minimum IORD# active pulse width of 1 clock. For wait states add the appropriate number of clocks to this value.

Other timings which need consideration for read cycles are the read inactive to data float delay and address/chip select hold time after read. The address and chip select hold times from read are not a problem, since both chip select and address are latched. The data float delay in slave devices can be very long (40ns). This can cause data bus contention and is taken care of with the I/O recovery logic. (Section 5.2.5 provides further information.)

5.3.2 Write Cycle Timing

An I/O write cycle starts similarly to an I/O read cycle. The address and status timings are similar to the I/O read. The processor outputs data in T_{11}. I/O write (IOWR#) may be asserted one or two clocks after the chip select (the exact delay between chip select and IOWR# depends upon the chip select/address setup to write requirements of the I/O devices). The use of latching data buffers can improve CPU write performance. Once the data and address of the cycle are latched, READY# can be returned to the processor, and the CPU operation and the write cycle to the device can continue. IOWR# is deasserted only after the data set-up to write specification is met. Data is written into the I/O device on the rising edge of IOWR#, and the processor stops driving data once READY# is sampled active.

The critical timings for the I/O write cycle are the following:

1. IOWR# Active Pulse Width:

   ≥Tcy − PLDclk-to-output − PLDclk-to-output − PLDclk-to-output + *n*Tcy
   ≥ Tcy + *n*Tcy

   n is the number of wait states inserted into the I/O write cycle. NTCY represents the additional time due to wait-states.

2. Address/Chip Select Set-up to Write:

   ≥ 3Tcy + PLDclk-to-output(min) − Decode delaymax
   − Address Valid delaymax − Latch Prop. Delaymax

   Chip select goes active later than I/O address and is latched, so chip select timing is used for the equation.

   Tcy : CLK PERIOD OF i860 Processor.
   Address Valid Delay : Processor address valid delay in T_{11}.

3. Data Setup To Write Inactive:

   ≥ 4Tcy − 860 Data Valid Delaymax − Data Buffer Prop. Delaymax
   + PLDclk-to-output Delay

Formula 3 assumes a zero-wait-state write cycle. Write is not posted. PLDclk-to-output delay is the delay in deasserting IOWR# from the rising edge of the clock.

Other timings in the I/O write cycle are the address/chip select hold time from write (IOWR#) high and data hold time after write (IOWR#) high. Since the address and chip select are latched, hold time is not a problem. However, if data is buffered through a 74F245 type device, the data could float as early as 7ns from READY#. Typical hold times for I/O devices vary from 0ns to 20ns. If data hold time requirements of the slave device are greater than 7ns, the write data must be latched; otherwise, READY# is returned some delay after the rising edge of IOWR#.

5.4 DESIGN EXAMPLES

This section discusses the i860 microprocessor interface to specific slave devices. It describes the basic interface, critical timings and equations.

5.4.1 82510 Interface

82510 is a UART with an asynchronous CPU interface. The basic interface is illustrated in Figure 5-5. The eight I/O ports of the 82510 are mapped to memory locations 0x07000000 through 0x07FF0000. The ports are located on 64-bit boundaries to allow data to be read on D7–D0 without the use of external byte swap logic. The chip select signal is generated from the decode of A20 and A19. By decoding more of the address lines, a smaller memory block can be used to map the device. For example, a full decode could allow the 82510 to be mapped to 0x07000000–0x0700000C. The address decoder generates the IOSEL as well as the 82510SEL signal. The address and chip select are

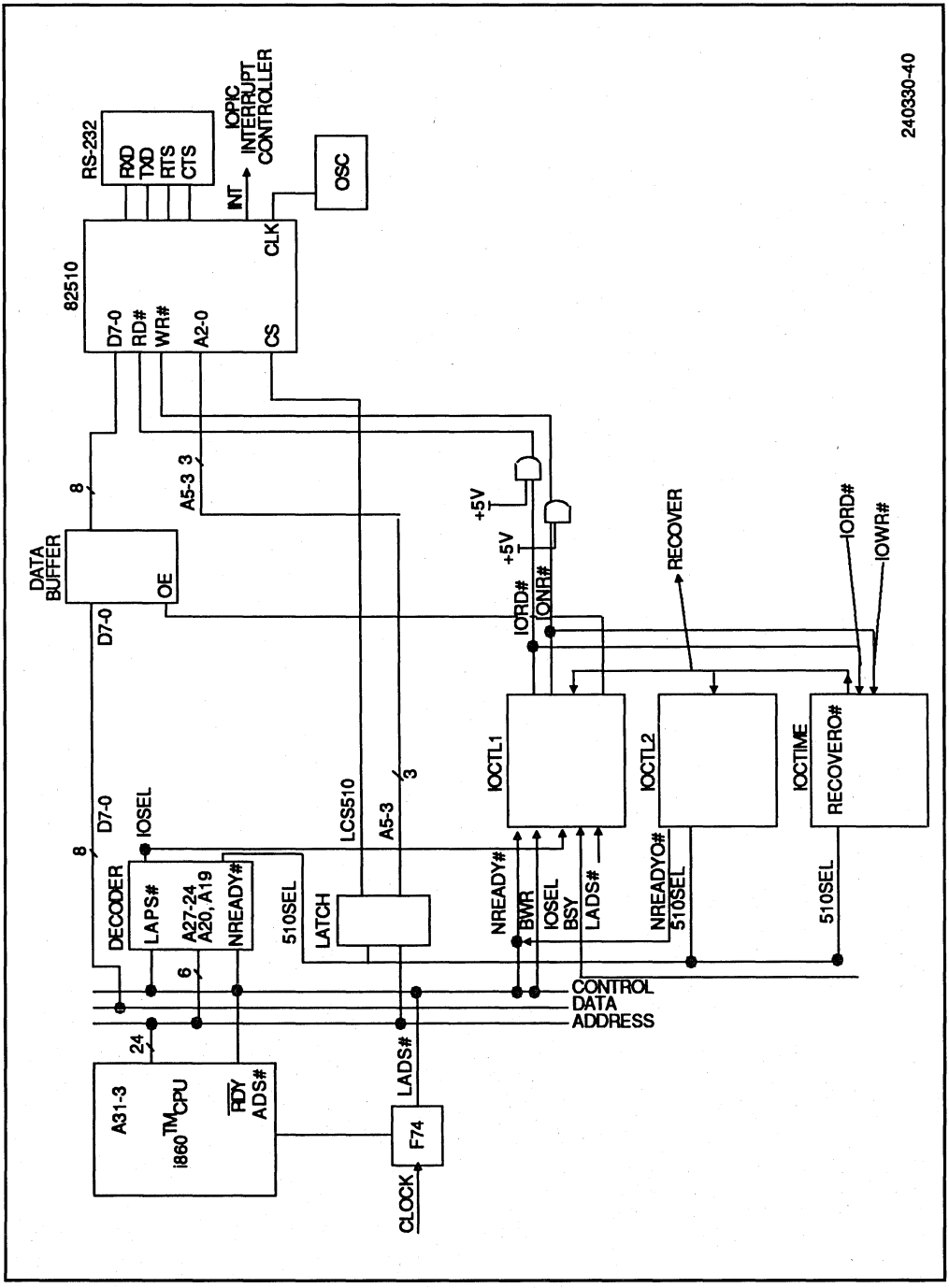


Figure 5-5. i860™ Processor Interface to 82510

latched to meet the minimum chip select active width and hold time. The RD# and WR# signals of the 82510 are generated by the IOCTL1 PLD. Critical timings of the 82510 are the following:

Read Cycle

1. Address Valid to Read Active (Tavrl):

   Tavrl ≤ 3Tcy + PLDclk-to-output(min) − Decode Delaymax
   − Address Valid delaymax − Latch Prop. Delaymax

   7ns ≤ 3Tcy −25
   ≤ 50ns (@40MHz)

2. Command Access Time To Data Valid (Trldv):

   Trldv ≤ Tcy − PLDclk-to-output − Data Setup Time
   − Data buffer Propagation delay + NTcy.

   N: # of Wait States.

   281 ≤ Tcy − 10 − 12 − 6 + NTcy
   ≤ (N + 1)Tcy − 28
   N = 9

3. Command Active Width (Trlrh):

   Trlrh ≤ (N + 1)Tcy
   N = 9

4. Read Inactive to Active (Tciad):

   Tciad ≤ 123ns
   ≤ 4Tcy − PLDclk-to-outputmax + PLDclk-to-outmin
   ≤ 4Tcy − 10 + 2
   ≤ 4*31 − 8
   ≤ 116ns

The control logic can assert IORD# no earlier than 116ns after the previous read. This violates the recovery requirements of the 82510. Control logic can enforce this requirement by delaying the assertion of IORD# by one clock, or software can implement this by using NOPs.

Write Cycle

Address valid to write low and write active width timings are similar to 1 and 2 for read cycles.

3. Data Valid to Write Inactive (Tdvwh):

   Tdvwh ≤ 3Tcy − Data Valid Delay − Buffer Delay + PLDclk-to-outputmin
   ≤ 3Tcy − 50 − 6 + 2
   90 ≤ 3*31 − 50 − 6 + 2 = 39

The IOWR# pulse must be extended by two clocks.

4. Data Hold Time after Write Inactive (Twhdx):

   Twhdx ≤ Data Float Delaymin + Buffer Delaymin
   12 ≤ 3.5 + 2.5 = 6.0

The minimum data float and buffer delays do not meet the 82510 hold time requirements. In this case, READY# can be delayed from IOWR# by one clock, or data can be latched and READY# can be returned early. The second option reduces wait-states but requires latches on the data bus.

5.4.2 Eprom Interface

The i860 CPU supports a special CS8 mode for bootup from 8-bit I/O devices. This allows the processor to bootup from an 8-bit ROM. Once the system boots, the ROM can be copied into memory or can be disabled and replaced by DRAM. In this mode, code fetches (that are misses in the code cache) are eight bits wide. To facilitate 8-bit reads, the BE2#–BE0# pins behave as A2–A0. Once the i860 microprocessor boots, the CS8 mode can be disabled by setting the CS8 bit of the **dirbase** register. If EPROM contents are to be copied into DRAM after the bootup, the EPROM locations need to be remapped to 64-bit boundaries. This allows the i860 microprocessor to access the EPROM bytes on D7–D0 without using byte swap logic. It does, however, require the address multiplexor to use either the processor address bus or the byte enables for A2–A0 inputs of the EPROM. A method which does not require an address mux is the "double copy load" method. This method is more dependent upon software, however. It uses the BE2#–BE0# signals which are connected directly to the EPROM A2–A0 addresses to select the bytes.

Figure 5-6 shows the i860 processor interface to a byte-wide EPROM (27010) and uses an address multiplexor to select between BE2#–BE0# and processor addresses. The chip enable of the EPROM is generated decoding the memory address and the MAP bit. The MAP bit, when set, maps the EPROM addresses into the power-up bootstrap locations. When MAP is low, the EPROM is mapped to another address and DRAM is mapped to the bootstrap locations. The OE# signal is connected to the IORD# signal generated by the I/O Control Logic. The upper 14 address bits of the EPROM are connected to i860 CPU addresses. The lower three addresses A2–A0 are multiplexed between BE2#–BE0# or three address bits of the processor. The multiplexor is controlled by the MAP input which is generated by a bit in an I/O register. The EPROM data bus D7–D0 is connected to the lower byte (D7–D0) of the processor data bus. This requires that the EPROM, when mapped as data, be on 64-bit boundaries.

Use the circuit in Figure 5-7 for i860 microprocessor systems that boot from EPROM and copy their code to DRAM for normal execution. Normally, an EPROM used in CS8 mode cannot be read as data. By adding a triple 2-to-1 multiplexor and using special addressing, the EPROM contents can be read out as data. A special routine is needed to access the EPROM. A portion of the physical address space must be reserved for addressing the EPROM as data. Figure 5-6 illustrates the connection from i860 CPU to EPROM.

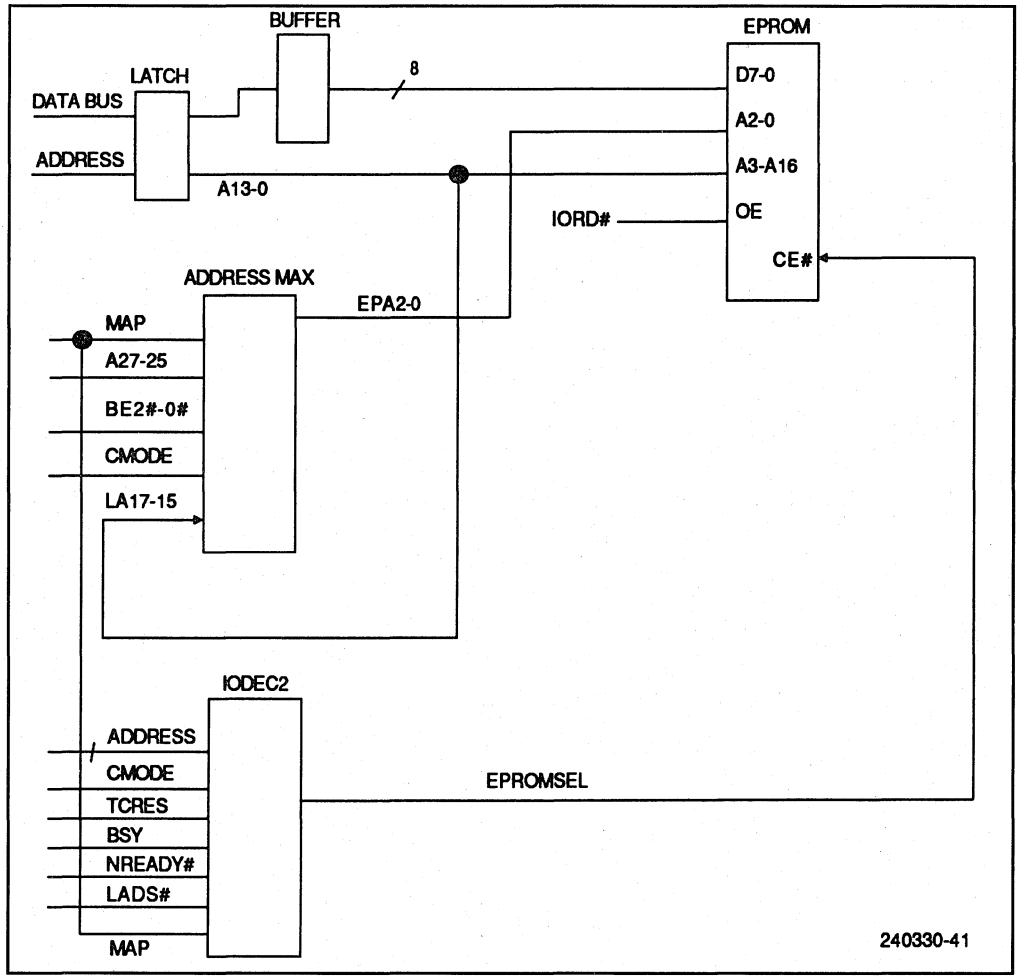


Figure 5-6. i860™ EPROM Interface

The EPROM(s) are connected to the low order 8 data bits of the i860 microprocessor data bus. The top 32 Mbytes of the physical address space is reserved to address up to 2 Mbytes of EPROM code/data. A physical address space sixteen times the size of the EPROM must be used. The EPROM address space must be at the top of physical memory.

Accesses with A24 HIGH are code accesses and should activate KEN# to enable caching of instructions from EPROM. Accesses to EPROM with A24 LOW must not allow caching of the EPROM data. The signals on data bus D63–D8 are undefined and must be ignored by the program. The data multiplexor must select the B inputs when A24 is high.

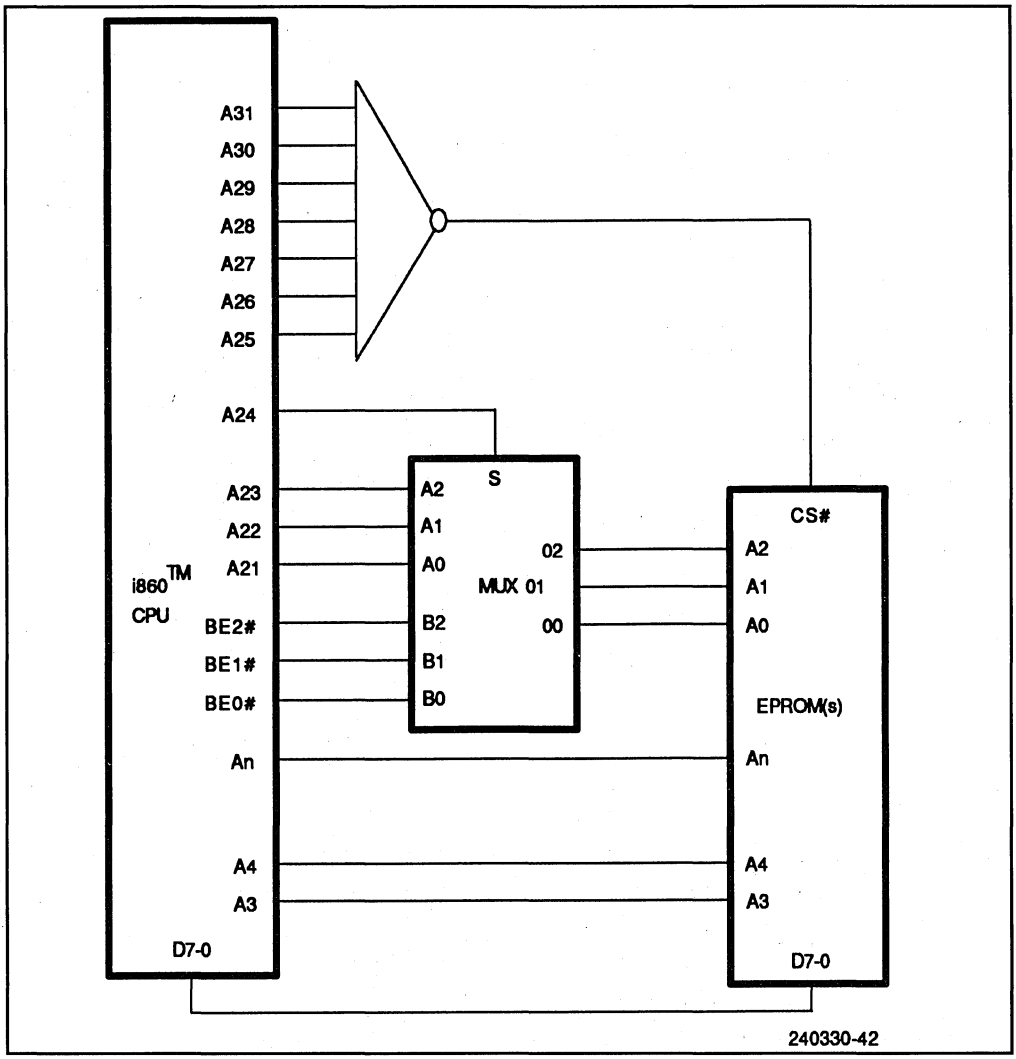


Figure 5-7. Circuit for Booting from EPROM

The EPROM data is copied from EPROM to DRAM via a subroutine. The subroutine is called: **eprom_copy(eprom_base,addr_inc,start,count,destination)**. The subroutine copies count bytes from start offset in the EPROM to destination.

The parameter **eprom_base** points at the beginning of the 16× EPROM space. For the hardware above, **eprom_base** is 0xFE000000. The **eprom_base** is not the same address as would be loaded into the instruction pointer to execute the first four bytes as instructions. The program address of the first instruction at offset 0 in the EPROM in the example above would be 0xFFE00000. In the example above, the EPROM instructions could also be executed at addresses 0xFFC00000, 0xFFA00000, 0xFF800000,

0xFF600000, 0xFF400000, 0xFF200000, and 0xFF000000. An attempt to read these address ranges would not allow access to all bytes of the EPROM since not all combinations of BE2#–BE0# can be generated via reads.

The parameter **addr_inc** is the address pin used to access individual bytes in the EPROM, for the hardware above this is 21. Consecutive bytes in EPROM are spaced 2^{21} bytes apart. At every eight bytes the low order address pins change. The table below illustrates:

EPROM byte offset	Address used by **eprom_copy**
0	0xFE000000
1	0xFE200000
2	0xFE400000
3	0xFE600000
7	0xFEE00000
8	0xFE000008
9	0xFE200008

The parameter **start** is the byte offset in the EPROM to begin the copy. This offset assumes the EPROM appears as a sequence of bytes. The **start** can have any alignment. The **count** is the number of bytes to copy. The **destination** is where to copy the destination data, which can have any alignment.

This subroutine can be called at any time so long as the program can address the lower half of the 16× EPROM address space.

```
//
//  R16 has base address of EPROM block
//  R17 has the address base used to access individual bytes
//  R18 has the starting offset of data in EPROM
//  R19 has the count of bytes to copy
//  R2Ø has the destination address
//

eprom_copy::
    or        1,rØ,r21          // Form address increment
    adds      -1,r19,r19        // See if zero count
    shl       r17,r21,r21       // Form EPROM byte address increment
    bnc       exit              // Exit if zero count

    or        7,rØ,r22          // Form address limit test value
    shl       r17,r22,r22       // Form maximum byte offset
    addu      r16,r22,r25       // Form upper address limit
    and       7,r18,r23         // Form starting byte address
    shl       r17,r23,r23       // Lower byte offset
    andnot    7,r18,r18         // Always address aligned address
    addu      r23,r18,r18       // Form starting read address
    adds      -1,rØ,r24         // Setup decrement value for BLA
    bla       r24,r19,cloop     // Setup LCC for next BLA
    addu      r16,r18,r18       // Form virtual address to read EPROM
cloop:
    ld.b      Ø(r18),r26        // Get value from EPROM
    subu      r18,r25,rØ        // Set carry if we read mod 7 byte
    bnc.t     in_block          // Jump with next instruction if not
    addu      r21,r18,r21       // mod 7 byte, Increment byte address

    andnot    r22,r18,r18       // Go to MOD Ø byte
    adds      8,r18,r18         // Go to next EPROM 8 byte block
in_block:
    st.b      r26,Ø(r2Ø)        // Put data byte into DRAM
    bla       r24,r19,cloop
    adds      1,r2Ø,r2Ø         // Bump destination pointer
exit:
    bri       r1                // Exit
    nop                         // Nothing to do here
```

5.4.2.1 DOUBLE COPY LOAD

An alternative to the address multiplexor is the "double copy load" method. This method requires considerably more software effort than the address mux method and also uses more EPROM space (since two copies of the code are required). In this case the EPROM A2–A0 are directly connected to the BE2#–BE0#. The 8-bit data bus of

the EPROM is connected to D7–D0 of the processor. Unlike the address mux method of copying, this method uses the on-chip data cache; and like the address mux method, the EPROM will need to be remapped from the bootstrap locations into another portion of the memory space of the i860 microprocessor. However, in this case, two copies of the code are needed in the EPROM: one copy with code bytes located on 8-bit boundaries and one copy with code bytes spaced for copying. As Table 5-1 illustrates, in the non-CS8 mode there are only six valid BE2#–BE0# combinations for the eight possible EPROM addresses. Therefore the second copy of the code needs to be spaced so that the first byte is at location 0 and the second byte is at location 3.

Table 5-1. Valid Addresses for SPACED copying

Bus BE# Address	Valid Address 210 T or F Address	EPROM
0 000	T (Load 32-bit value)	0
1 001	F (16-bit access on odd boundary)	Invalid
2 010	F (two noncontiguous bytes)	Invalid
3 011	T (Load 1 byte)	3
4 100	T (Load 16-bit value)	4
5 101	T (Load 1 byte)	5
6 110	T (Load 1 byte)	6
7 111	T (Load 1 byte)	7

The data is first copied into the data cache by using the **ld.l** or **ld.b** instructions to assert the proper BE2#–BE0# values. Once the data is in the cache it is read into a register and saved in the appropriate DRAM location. An untested example of the possible code for the "double copy load" method is illustrated in the following pages.

5.4.2.2 EPROM TIMINGS

The timing analysis assumes that the address multiplexor scheme is being used for the EPROM interface. The OE# input of the EPROM is connected to IORD# output of the I/O Control Logic. The following are the critical timings:

1. Address/CE# to Output (Tacc):

   Assumes that the multiplexed address bits are the last to become valid.

   Tacc ≤ 3Tcy − Latch Propmax − Addr. Muxmax − i860 CPU data Setup − Buffer Delaymax

   Tacc ≤ 3Tcy − 35

   For Tacc of 200ns @33 MHz Two Wait States

   Addr. Mux.: Address Multiplex Delay

2. OE# to Output (Toe):

   Assumes zero-wait-state cycle.

   Toe ≤ Tcy − PLDclk-to-output − Buffer Delay − i860 Microprocessor Setup

   Toe ≤ Tcy − 28

   For Toe of 85ns requires 3 wait states.

3. OE# High to Output Float (Tdf):

   If READY# is asserted to the processor on the clock edge prior to IORD# going inactive, the EPROM data outputs can float as late as 60ns after the rising edge of IORD#. This means that EPROM data may be valid until as late as 10ns into the second T_{11} of the next cycle. The i860 CPU for writes can output data as early as 3.5ns from clock edge of the second T_{11}. If this were allowed to happen it would result in data bus contention and would cause problems in the system. The IOTIME PLD eliminates bus contention by not allowing the next cycle to begin until all data outputs of the slave device float.

5.5 DMA INTERFACE RECOMMENDATIONS

Performing DMA (Direct Memory Access) transfers in an i860 microprocessor system requires deciding between two general approaches. One uses DMA in the conventional mode, performing data transfers directly to DRAM. The other technique uses an intermediate memory area. Each has important issues regarding performance, software complexity, and part count.

Any DMA design must take into account the on-chip caches. The i860 microprocessor caches use a write-back mechanism to minimize external bus traffic and increase performance. The on-chip caches are logical-address caches. External bus snooping is not provided since such cycles would conflict with internal accesses, reducing overall performance.

If DMA is permitted directly to main memory, then the software must insure that no data values associated with that memory area are in the caches. Either the I/O buffer is always marked non-cacheable or the data cache must be flushed before the DMA may read from the memory area. Both the code and instruction caches must be flushed after the DMA has written to memory if the i860 microprocessor might have an old copy of the data.

If DMA transfers directly to main memory and this I/O buffer is used directly by the application, it must also take into account the paging structure used in most systems. Paging breaks up the contiguous logical address space of the application into discontiguous memory addresses. The DMA device needs to transfer a stream of data to discontiguous addresses if that data stream crosses a memory page boundary.

Another approach to direct DMA is to allocate track buffers in memory which are always non-cacheable and occupy consecutive physical memory pages. The operating system can manage these buffers to speed up disk accesses since an entire track can be read in one revolution of the disk and kept for later disk accesses which usually want a

sector in the same track. The OS can copy data between this track buffer and the application. The copy will update all caches and handle discontiguous memory pages. DRAM-to-DRAM copy can be as fast as 56/67 Mbytes/sec.

Based on these issues, two general approaches are possible: low-cost, lower performance design using DMA directly to i860 microprocessor memory and a higher cost, higher performance approach using an intermediate memory area.

An 82380 is used in this example. It provides eight DMA channels, 17 interrupt inputs, and five 16-bit timers. The 82380 has a bus interface identical to that of the 386™ microprocessor. It uses a 2× clock. A 16 MHz 82380 requires the i860 microprocessor to run at 32 MHz. A 20 MHz 82380 can work with a 40 MHz i860 microprocessor.

The first approach is conventional. An 82380 DMA device with four extra data transceivers could perform DMA to memory. It can run directly off the i860 processor clock with special PLDs that convert the 82380 bus signals into DRAM commands. The i860 processor HOLD/HLDA signals are used. Each transfer requires at least four clocks on the i860 processor bus. More will usually be needed to allow access to real memory or I/O devices. For DRAM accesses by the 82380, the DRAM controller must perform them in pairs of clocks to match the 82380 bus timing. If the I/O device is not 32-bits wide, then the 82380 must either perform two-cycle transfers or else extra data transceivers are needed to route the I/O data to the correct part of the 64-bit data bus.

The 82380 can support paged memory systems via a set of second address and count registers per channel. After completing the transfer of one block, the DMA channel automatically switches to the next set of registers. The registers must be reprogrammed during the time of a page transfer. If the operating system can not guarantee quick enough interrupt response, DMA transfers cannot cross a page boundary.

The second approach is to add a special I/O buffer memory between the I/O device and the i860 microprocessor. SRAMs make this easier. The 82380 performs DMA transfers into the SRAMs on an isolated bus. The SRAM area is large enough for all the simultaneous DMA transfers in progress at once.The i860 processor copies data between the SRAM space and DRAM. The copy operation updates the cache and handles page boundary crossing. This approach offers higher performance, because the DMA channels do not tie up the CPU bus when accessing the I/O device. The SRAM space should be 64-bits wide to maximize the copy speed.

The cost of using the I/O buffer memory is the SRAM devices and two or three address latches. Four data transceivers are still needed to buffer the data bus. Reads to the SRAM area must disable KEN#.

When copying data between the application and I/O buffer, use **pfld** instructions to prevent the data from flushing the data cache. Normal writes can be used, because a write miss does not cause a cache load. The copy function shown here achieves 1Mbyte/MHz data rate, assuming 4 wait states on first DRAM access and 1 wait state per SRAM access. At this rate, the time spent in copying data is negligible.

Note: This example has not been tested

```
//
//  Copy a block of data from the source to the destination.
//  Use pipelined floating-point loads to prevent data accesses
//  from flushing out data cache contents.
//
//  bcopy(source_ptr,dest_ptr,count8)
//
//  r16 on entry has pointer to the source, must be 8-byte aligned
//  r17 on entry has pointer to the destination, must be 16-byte aligned
//  r18 on entry has count of 8-byte words to transfer.
//

//  The transfers are performed on source and destination in series of
//  back-to-back operations to allow page mode accesses to DRAMs.
//
//  Assuming 3 wait states on the first access, the transfer speed is
//   38 clocks per 64 bytes transferred for big blocks.
//        56.1 Mbytes/sec at 33.33 Mhz
//        67.4 Mbytes/sec at 40 Mhz
//        0 bytes 8 clocks
//        8-128 bytes 5+10n/8 clocks
//
bcopy::
    addu    -16,r18,r20         // See if at least 16 transfers
    shr     3,r20,r21           // Setup loop counter for 8 per loop
    bc.t    do_long             // Jump if long transfer
    pfld.l  (r16),f0            // Start reads on long transfer
//
//  Copy small block of data 8 bytes at a time
//
    bte     r0,r18,exit         // Jump if 0 count
    adds    -1,r0,r19           // Setup BLA counter
dlast:
    adds    -1,r18,r18          // Setup for BLA
    subs    r17,r16,r17         // Form distance from source to destination
    bla     r19,r18,sloop       // Set LCC for next loop
    adds    -8,r16,r16          // Setup for autoincrement addressing
sloop:
    fld.l   8(r16)++,f16        // Get source and bump address
    bla     r19,r18,sloop       // Loop if more to do
    fst.l   f16,r17(r16)        // Write destination
exit:
    bri     r1                 // Return to caller
    nop                        // Nothing to do here
//
//  Start the pipelined floating-point reads
//
```

```
do_long:
    adds     -1,r0,r19              // Setup BLA counter
    pfld.    8(r16)++,f0            // Read the next value
    bla      r19,r21,bloop          // Set LCC for next loop
    pfld.l   8(r16)++,f0
bloop:
    pfld.l   8(r16)++,f16           // Get value mod 0
    pfld.l   8(r16)++,f18           // Get value mod 1
    pfld.l   8(r16)++,f20           // Get value mod 2
    pfld.l   8(r16)++,f22           // Get value mod 3
    pfld.l   8(r16)++,f24           // Get value mod 4
    pfld.l   8(r16)++,f26           // Get value mod 5
    pfld.l   8(r16)++,f28           // Get value mod 6
    pfld.l   8(r16)++,f30           // Get value mod 7
    fst.q    f16,(r17)              // Store 0-1 pair
    fst.q    f20,16(r17)            // Store 2-3 pair
    adds     64,r17,r17             // Update destination pointer
    fst.q    f24,-32(r17)           // Store 4-5 pair
    bla      r19,r21,bloop          // Loop if more to do
    fst.q    f28,-16(r17)           // Store 6-7 pair

//
//  Finish off last transfers to mod 8
//
    pfld.l    8(r16)++,f16          // Start last 5 reads
    pfld.l    8(r16)++,f18          // Get value mod 1
    pfld.l    8(r16)++,f20          // Get value mod 2
    pfld.l    8(r16)++,f22          // Get value mod 3
    pfld.l    8(r16)++,f24          // Get value mod 4
    pfld.l    (r16),f26             // Get last three data words
    adds      8,r16,r16             // Update source pointer
    pfld.l   -8(r16),f28            // Reread last source
    pfld.l   -8(r16),f30            // Get value mod 7
    fst.q     f16,(r17)             // Put 0-1 pair into memory
    fst.q     f20,16(r17)           // Put 2-3 pair into memory
    fst.q     f24,32(r17)           // Put 4-5 pair into memory
    and       7,r20,r18             // Get remaining count
    fst.q     f20,48(r17)           // Put 6-7 pair into memory
    bnc.t     dlast                 // Jump if still work
    adds      64,r17,r17            // Update destination pointer
    bri       r1                    // Else all done
    nop                             // Nothing to do here
```

Double copy load method:

```
//   This program copies ROM code to DRAM. It assumes that BE2#:BE0# are
//   wired to the ROM address bits 2:0 (least significant bits).

dramstart=  0x7eff0000      // non-cacheable alias for DRAM, where ROM
                            // copied code will go.
epromstart= 0xffff0000
epromend=   0xffffff0
           .atmp r31
           .text
copyrom::                  // start coping EPROM contents to DRAM
    or     l%epromstart,r0,r9
    orh    h%epromstart,r9,r9
    or     l%epromend,r0,r7
    orh    h%epromend,r7,r7
    call   flush_cache          // make sure we get a miss
    nop
    or     l%dramstart,r0,r8
    orh    h%dramstart,r8,r8
loadbytes::
    ld.l   0(r9),r10           // cache data for bytes 0 & 8
    ld.b   0(r9),r10           // load byte 0 from cache
    call   flush_cache         // insure cache miss next time
    ld.b   8(r9),r18           // load byte 8 from cache

    ld.b   2(r9),r13           // cache data for bytes 3 & 11
    ld.b   0(r9),r13           // load byte 3 from cache
    call   flush_cache         // insure cache miss next time
    ld.b   8(r9),r19           // load byte 11 from cache

    ld.s    0(r9),r14          // cache data for bytes 4 & 12
    ld.b    0(r9),r14          // etc.
    call    flush_cache
    ld.b    8(r9),r20

    ld.b    1(r9),r15          // cache data for bytes 5 & 13
    ld.b    0(r9),r15
    call    flush_cache
    ld.b    8(r9),r21

    ld.b    0(r9),r16          // cache data for bytes 6 & 14
    ld.b    0(r9),r16
    call    flush_cache
    ld.b    8(r9),r22

    ld.b    3(r9),r17          // cache data for bytes 7 & 15
    ld.b    0(r9),r17
    call    flush_cache
```

```
    ld.b    8(r9),r23
store::
    st.b    r1Ø,Ø(r8)          // store byte Ø to DRAM
    st.b    r13,1(r8)          //  ''    ''   3 ''
    st.b    r14,2(r8)          //            4
    st.b    r15,3(r8)          //            5
    st.b    r16,4(r8)          //            6
    st.b    r17,5(r8)          //            7
    st.b    r18,6(r8)          //            8
    st.b    r19,7(r8)          //           11

    st.b    r2Ø,8(r8)          //           12
    st.b    r21,9(r8)          //           13
    st.b    r22,Øxa(r8)        //           14
    st.b    r23,Øxb(r8)        //           15
    xor     r7,r9,rØ           // check for end of ROM
    bc      load_reset         // branch if end of ROM
    addu    Øx1Ø,r9,r9         // increment ROM counter
    br      loadbyts           // copy another 12 bytes
    addu    Øxc,r8,r8          // increment DRAM counter

    load_reset::               // Load the reset code to DRAM

    w1=Øxa0000000              // This hex code is for the reset branch
    w2=Øxa0000000              // branch from Øxffffff00 to Øxffff0008
    w3=Øx6bffcØ3f              // (Two nop's, then branch)
    w4=Øxa0000000              // Sequence = nop; nop; br; nop

resxc_start=Øx7efff00          // non-cacheable alias of fefff00
dramc_start=Øxfff0000          // cacheable DRAM that will be mapped
                               // over EPROM space after boot
byte_bucket=Øx7fff00Ø1Ø        // ROM location

    or      l%resxc_start,rØ,r4      // bottom of reset branch code
    orh     h%resxc_start,r4,r4
    or      l%w1,rØ,r5
    orh     h%w1,r5,r5
    st.l    r5,Ø(r4)                  // store word 1
    addu    4,r4,r4                   // increment counter
    or      l%w2,rØ,r5
    orh     h%w2,r5,r5
    st.l    r5,Ø(r4)                 // store word two
    addu    4 ,r4,r4
    or      l%w3,rØ,r5
    orh     h%w3,r5,r5
    st.l      r5,Ø(r4)               // store word three
    addu      4 ,r4,r4
    or        l%w4,rØ,r5
```

```
    orh     h%w4,r5,r5
    st.l    r5,Ø(r4)              // store word four


// The next line must be hand patched after every reassembly

    or      Øxf174,rØ,r1          // fffff174 is mask_cs8
    orh     Øxffff,r1,r1

 // We go through the warp drive section twice to insure that
 // it gets completely cached. We must be executing from
 // cache when we zero CS8 and the boot bit.

    or      l%byte_bucket,rØ,r5
    orh     h%byte_bucket,r5,r5
    ld.c    dirbase,r4                  // get dirbase contents
warp_drive::
    st.b    rØ,Ø(r5)                    // does nothing first time
    st.c    r4,dirbase                  // does nothing first time
    bri     r1                          // go to mask_CS8 the first time
    nop                                 // go to ffffØØØØ second time
mask_cs8::
    or      Øxff7f,rØ,r3                // kill cs8 bit
    orh     Øxffff,r3,r3
    and     r3,r4,r4
    or      Øx2Ø,r4,r4                  // invalidate code cache bit
    xorh    h%byte_bucket,r5,r5         // zero r5's high 16 bits
    orh     ØxØ1ØØ,r5,r5
    or      l%dramc_start,rØ,r1         // r1 now has DRAM start address
    orh     h%dramc_start,r1,r1
    br      warp_drive                  // r5 has the boot port address
    nop                                 // r4 has CS8 bit reset

// The following flush procedure is from the i86Ø™ Programmer's Reference
// manual. Please reference the manual for additional information.
flush_cache::
 FLUSH_P=    Øx7fØØØØØØ-32
 //rw=r24, rx=r25, ry=r26, rz=r27
        mov     r1,r2
        ld.c    dirbase,r27
        or      Øx8ØØ,r27,r27
        adds    -1,rØ,r25
        call    D_FLUSH
```

```
        st.c    r27,dirbase

        or      0x900,r27,r27
        call    D_FLUSH
        st.c    r27,dirbase

        xor     0x900,r27,r27
        mov     r2,r1
        bri     r1
        st.c    r27,dirbase

D_FLUSH::
        or      l%FLUSH_P,r0,r24
        orh     h%FLUSH_P,r24,r24
        or      127,r0,r26
        bla     r25,r26,D_FLUSH_LOOP
        ld.l    32(r24),r0

D_FLUSH_LOOP::
        bla     r25,r26,D_FLUSH_LOOP
        flush   32(r24)++
        bri     r1
        ld.l    -512(r24),r0
.end
```

Graphics Subsystem Example

6

CHAPTER 6
GRAPHICS SUBSYSTEM EXAMPLE

6.1 INTRODUCTION

Computer graphics technology has developed rapidly in recent years and has transformed the computing environment. Graphics provides visualization – the ability of computers to create revealing, life-like images from hard-to-interpret numerical data. Graphics offers user-friendliness to ease the man-machine interface, and it opens new applications in science, engineering and the arts. Advances in graphics hardware technology are catalysts for innovation. Raster display technology, low-cost, high-speed display memory and powerful, intelligent graphics processors are key elements to recent changes. A decade ago, high resolution graphics was reserved to centralized computing facilities. Today, home and office computers provide even better graphics at a fraction of the cost.

The underlying trend in computer graphics development is toward better display and pixel resolution. Display requirements for solids modeling and visualization are stringent because of the need to accurately represent smooth shaded objects. Graphics workstations are a new class of machines for graphic-intensive applications. They combine medium- to high-resolution color graphics with high-speed computational capability. Applications may call for real-time manipulation of complicated images composed of close to a hundred thousand polygons. These 3-D graphics applications demand high MFLOPS (millions of floating-point operations per second) performance. Object modeling, transformation and rendering are common applications that usually require supercomputer power.

6.2 GRAPHICS AND THE i860™ MICROPROCESSOR

The i860™ microprocessor's architectural features provide high performance graphics capability. Floating-point power, parallel execution units, high integration and dedicated 3-D graphics hardware combine to provide supercomputing graphics performance and capability.

The processor provides an integer operation and up to two floating-point operations per clock cycle. Pipelined floating-point multiplication and addition units operate simultaneously using special dual operation instructions. This speeds matrix arithmetic and vector computation to provide a peak performance of 80 single-precision or 60-double precision MFLOPS at 40 MHz.

The i860 microprocessor supports 3-D graphics operations such as hidden surface elimination and Gouraud shading. It operates on 8, 16, or 32-bit pixels. Its high speed, 64-bit data bus delivers a peak 160 megabytes per second with zero-wait-state accesses. Dual instruction mode allows floating-point or graphics operations to execute in parallel with pixel loading and storing. Scoreboarding can provide continuous execution during cache-miss processing of data reads.

6.2.1 Processor Bus Bandwidth

In real-time applications, frames must be refreshed at least 10 times per second. Bandwidth requirements vary according to display and pixel resolution. Nearly 40 Mbytes/sec bandwidth must be dedicated to memory refresh to provide real-time graphics on a 1,280 × 1,024 pixel monitor with 24-bit resolution. Additional data transfers for modeling, transformation, hidden surface elimination and shading increase the bandwidth requirement.

At 40 MHz, the i860 microprocessor provides 160 megabytes per second bus bandwidth with zero-wait-state cycles. This allows for real-time manipulation of high-resolution 3-D images.

6.3 3-D GRAPHICS EXAMPLE

This example outlines a graphics frame buffer daughter card for an i860 microprocessor evaluation vehicle. The example does not represent an ideal design but is instead employed to demonstrate the processor's 3-D graphics capabilities. The concepts employed here can be extended to complete systems where more board space and dedicated hardware/software support permit greater sophistication.

This evaluation vehicle is designed as a frame buffer extension to an i860 microprocessor based system. The processor connects to the frame buffer board through the expansion bus. The frame buffer is mapped into the expansion space as allocated by the CPU core.

6.3.1 Features

The following features are included in the frame buffer example:

- 1,024 × 768 RGB Display with 16-Bit Pixel Resolution
- Double-Buffering at Two Megabytes Per Buffer
- 33 MHz with 40 MHz Upgrade Path
- PLD-Based VRAM Controller
- VRAM Control Options for 100ns and 80ns VRAMs
- PLD-Based Noninterlaced CRT Control

Dedicated to the three colors, red, green and blue, are 6, 6, and 4 bits per pixel respectively. Sixteen-bit pixel resolution allows a double-buffered scheme requiring only four megabytes of total display memory for a 1,024 × 768 pixel display. It also allows four pixels per 64-bit load/store operation and four pixels per computation. Transformation and refresh rates are higher than those obtained with a 24-bit or 32-bit pixel resolution.

6.3.2 Testing

This example has been tested for the operational mode that uses 100ns VRAM. Refer to Section 6.5.1.1. PLD codes are included in Appendix A.

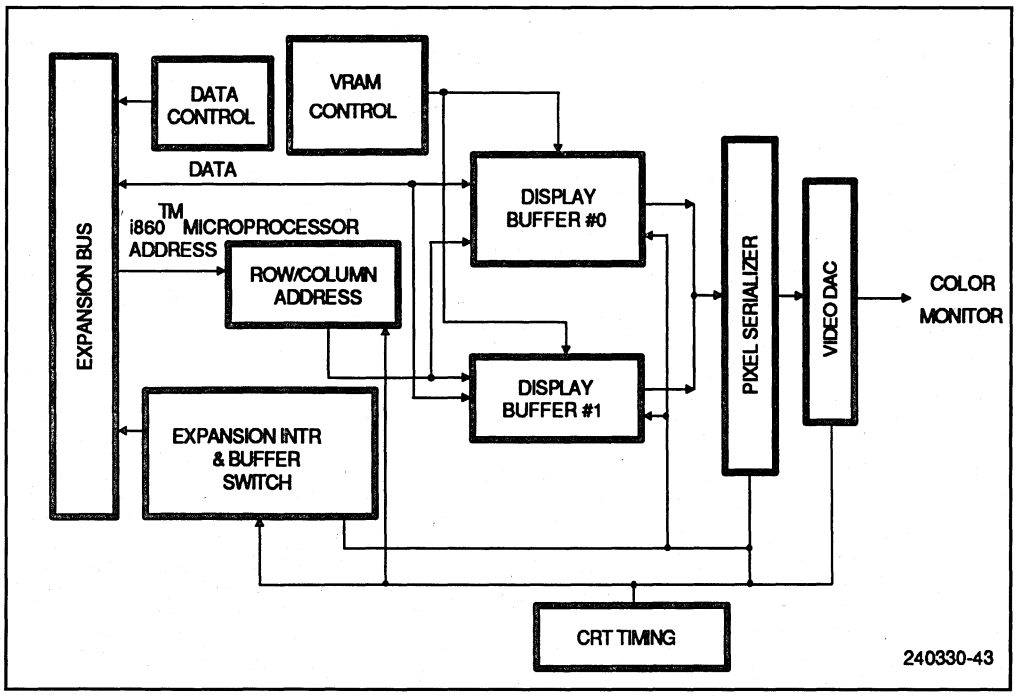


Figure 6-1. Block Diagram of the i860™ Microprocessor Based Graphics Frame Buffer Board

6.4 SYSTEM OVERVIEW

The frame buffer board is composed of three major sections: VRAM control, CRT control and expansion bus interface. The VRAM controller controls four megabytes of video RAMs which are divided into two display buffers. While the i860 processor accesses one buffer, the CRT may display the other buffer. Refer to the block diagram in Figure 6-1.

6.4.1 Expansion Bus Interface

Frame buffer logic resides on the expansion bus. The key signals are listed below:

- 64-bit buffered data bus
- Bits A23-A3 of the i860 microprocessor address bus
- i860 microprocessor control signals such as ADS#, READY# and NA#
- Decoded signals from the baseboard logic such as EXPSEL#
- Power supply and ground signals

Refer to the schematics in Appendix A for the complete list of signals.

6.4.2 Data Transceiver/Latch Control

Data is transferred to and from the i860 microprocessor through data transceivers/ latches on the i860 microprocessor board. Expansion data control signals are generated to control the logic.

6.4.3 Address Transceiver/Latch

Address bits A20-A3 are used to generate row and column addresses during data access cycles. Bits A20-A12 are buffered and then latched as row address by VRAMs. Bits A11-A3 are latched externally as column address when a VRAM cycle is detected. Including bit 21 (which selects display buffer #0 or #1), four megabytes of address space are provided.

6.4.4 VRAM Control

VRAM control provides VRAM control signals for read/write cycles, refresh cycles, and RAM-to-SAM transfer cycles. VRAM read/write cycles are synchronized with pipelined cycles to the DRAM subsystem on the i860 microprocessor board.

6.4.5 Serial Row/Column Address Generation

RAM-to-SAM transfer for the serial port occurs during horizontal blank periods, and row addresses increment accordingly. Each row transfer provides two screen lines, and row addresses increment every other horizontal blank time. A column address latched during SAM transfer indicates the origin of data within the SAM register. All but the most significant bit in the column address are zero. The most significant bit alternates between one and zero. If the bit is a one, the serial shift from the serial port originated from the middle of the SAM register. This occurs during a RAM-to-SAM transfer of display data for the second screen line stored in the second half of the same VRAM row. The row address is not incremented in this case.

6.4.6 Double Buffering

Double buffering reduces flickering and partial image update. Here, each buffer consists of two megabytes of video RAM.

6.4.7 Expansion Interrupt/Buffer Switch

Upon reset, buffer #0 is enabled. When ready to display data in a second buffer, the processor may read or write to a location where A22 is high. Actual buffer switching occurs in the subsequent vertical trace. A low value on A3 indicates that display data comes from buffer #0; a high value indicates that data comes from buffer #1. The processor is interrupted when vertical retrace occurs.

6.4.8 CRT Timing Generation

Blank and sync signals are generated with PLD/TTL logic, and clocking is derived from the 64 MHz pixel clock. The 16 MHz serial clock clocks data out of the VRAM serial ports.

6.4.9 Pixel Serializer

Pixel resolution is 16 bits/pixel, and four pixels are clocked out with each rising edge of the serial clock. Pixels are loaded into shift registers at serial clock rate but shifted out of the registers at pixel clock rate.

6.4.10 Video DACs

Video DACs convert digital video data into analog video data at the pixel clock rate. The sync-on-green mode of the color monitor provides synchronization.

6.5 OPERATION

6.5.1 VRAM Control

VRAM control logic provides all control signals for VRAM operation, including serial port accesses.

6.5.1.1 SPEED MODE

Two operation modes are designed to use 100ns and 80ns VRAMs. They are designated as 1-wait-state-write and 0-wait-state-write modes respectively. The 100ns VRAMs are used and tested in initial prototyping. (PLD code for the 0-wait-state mode is not included in the appendix.) They require one wait-state for writes and two wait-states for reads in page mode. (PLD codes are included in Appendix A.) 80ns VRAMs are needed to support zero-wait-state writes and 1-wait-state reads. Refer to the timing diagrams in Figure 6-2 and 6-3 for one-wait-state-write mode operation.

6.5.1.2 PROCESSOR-INITIATED CYCLES

The frame buffer board shares processor and latched data control buses with the i860 microprocessor board. VRAM cycles may be started by the VRAM controller when the VRAM space is selected and DRAM busy signal (DRMBSY#) is deasserted (that is, pipelined cycles are completed). The expansion busy signal (EXPBSY#) is also asserted to prevent other cycles from starting while VRAM cycles are pending.

Once the first VRAM cycle is started, VRAMs remain in page mode during near (NENE# asserted) cycles. Reads are pipelined with NA#. A new cycle can start when NA# is activated in pipelined mode or when READY# is activated in nonpipelined mode. Idle or far cycles put VRAMs in precharge mode. Row addresses are

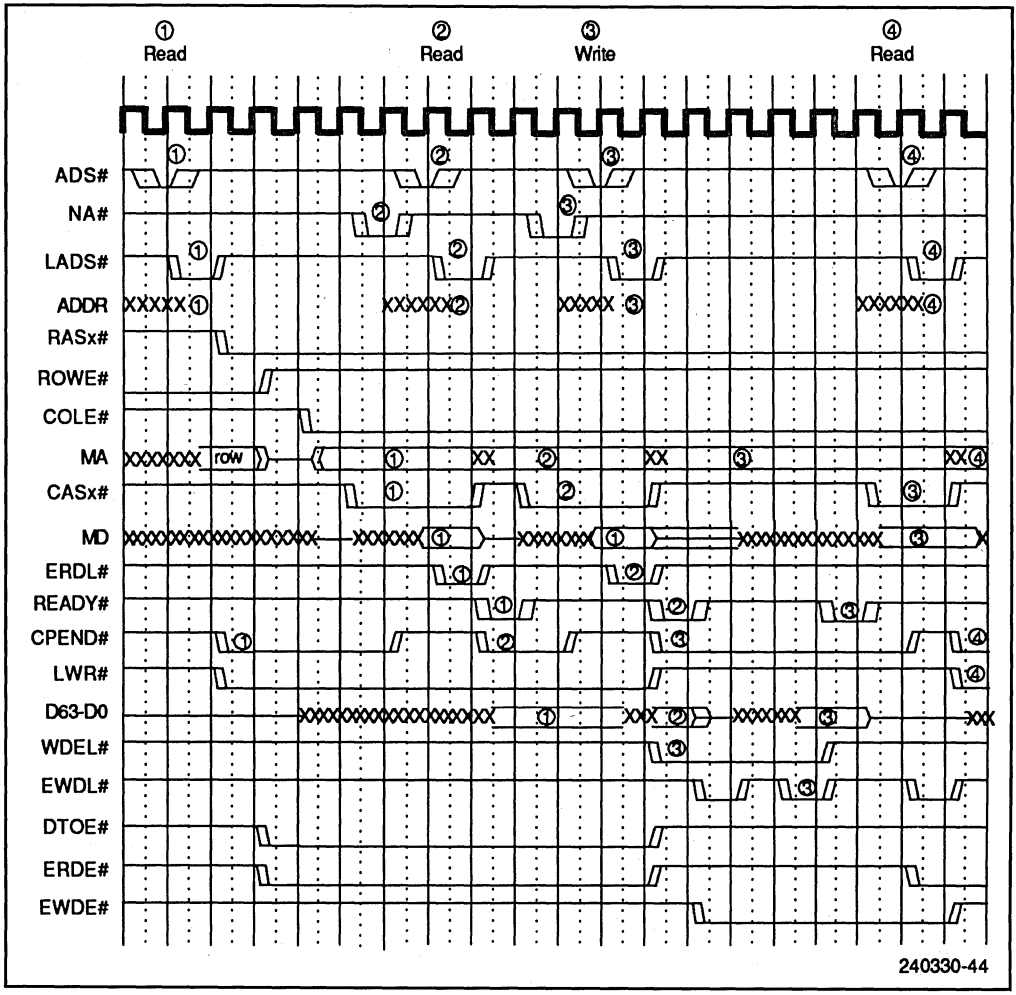


Figure 6-2. Read/Write/Read Operation (1-Wait-State-Write Mode)

latched by VRAMs on the falling edge of RAS#, and column addresses are latched on the falling edge of CAS#. Subsequent near cycles to the VRAM space need to supply column addresses only. Near cycles are indicated by the i860 microprocessor NENE# pin.

In read cycles, data is latched by the data latch on the i860 microprocessor board before being read by the processor. This provides sufficient setup and hold times. In write cycles, data is also latched before being written into VRAMs. The data latch is controlled by expansion bus data control lines. Latch latency requires that write data be latched to maintain one-wait-state operation in successive VRAM write cycles (one-wait-state-write mode operation). Latch signals require at least one clock period to recover.

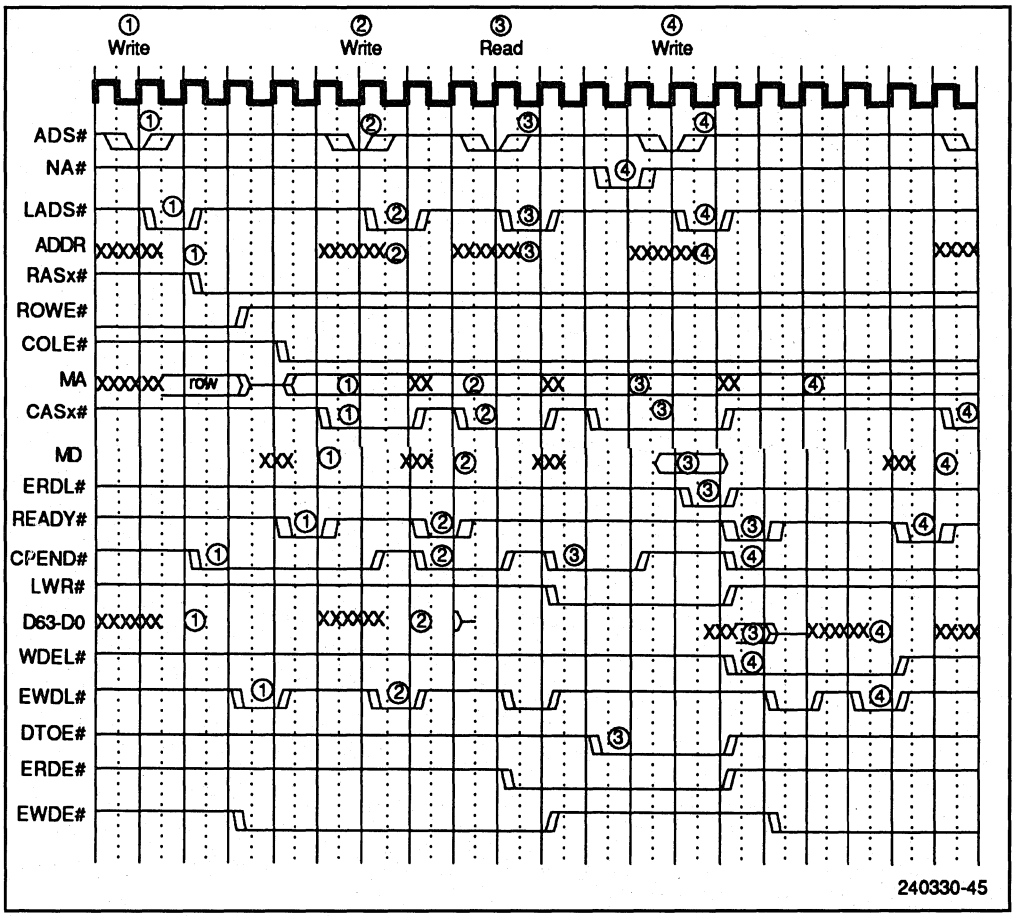


Figure 6-3. Write/Read/Write Operations (1-Wait-State-Write Mode)

When a write cycle is immediately preceded by a read, data is not driven by the processor until one clock after READY# is returned. This is illustrated in Figure 6-2 and 6-3.

6.5.1.3 REFRESH/RAM-TO-SAM TRANSFER CYCLES

The refresh and serial register (SAM) transfer VRAM cycles are not initiated by the processor. The cycles occur simultaneously in both VRAM banks and have priority over processor-initiated cycles. Outstanding processor-initiated cycles resume following refresh/SAM cycles. Refresh cycles have priority over RAM-to-SAM cycles.

Figure 6-4 illustrates a far cycle (NENE# not active) followed by a RAM-to-SAM transfer request and refresh request (REF). The refresh cycle begins as soon as RAS# precharge time has been met. The RAM-to-SAM request (TRQ) arrives one clock before the refresh request. Request lines are not sampled until the end of the precharge period where both lines are active. These cycles are equally important in displaying the correct

images continuously. RAM-to-SAM requests occur during video blanking time and minor delay is tolerable. The RAM-to-SAM cycle begins as soon as RAS# precharge time for the refresh cycle is satisfied. CAS-before-RAS refresh VRAM mode is used to generate a refresh address internally.

A RAM-to-SAM read transfer is requested when DT/OE# is asserted on the falling edge of RAS#. In real-time read transfers without RAM-to-SAM transfers, the transfer must remain active until CAS# has been deactivated. Although not indicated in the diagram, WE# must be deasserted during refresh/RAM-to-SAM transfer cycles. While external address generation is not required in refresh cycles, a scan line address is required for RAM-to-SAM transfers. The address setup and hold mechanisms are similar to those in a regular cycle.

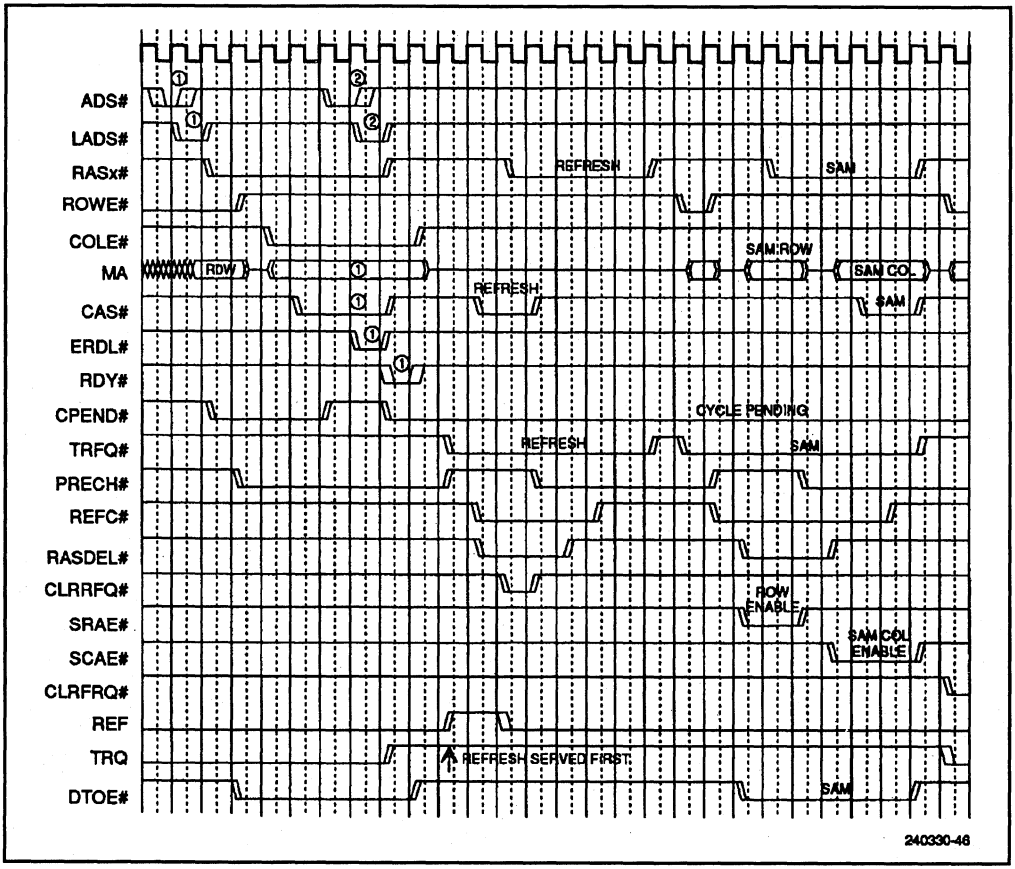


Figure 6-4. Refresh/RAM-to-SAM Transfers

6.5.2 Expansion Bus Interface

The expansion bus allows the i860 microprocessor to access the frame buffer. The schematics provide a complete list of bus signals. Expansion address space spans 8 megabytes. Mapping is illustrated in Figure 6-5.

6.5.2.1 EXPANSION SELECT

The expansion select signal (EXPSEL#) and signals A21 and A22 are needed to decode access to the expansion address space in two megabyte increments (refer to Figure 6-5). Low levels on EXPSEL# and A22 signal VRAM accesses. A21 distinguishes between display buffers #0 and #1 when the random port (processor side) of the VRAMs is accessed. When A22 is high for an access to the expansion space, A3 enables the serial port (CRT side) of buffer #0 or buffer #1. Buffer switching occurs during vertical retrace. Although available, A23 is not used in decode logic; VRAMs and unpopulated space are assigned to two eight megabyte regions.

6.5.2.2 DATA TRANSCEIVER/LATCH SHARING

Four signals control the data transceivers/latches used in VRAM data accesses. The ERDL# and EWDL# signals latch read and write data. ERDE# is used for transceiver enable, and EWDE# is used for transfer direction control. ERDL# and EWDL# must meet the setup time requirement relative to a delay clock (delayed from CLK) on the i860 microprocessor board. Data is latched in the latter part of CLK. EWDL# activates the write data latch signal.

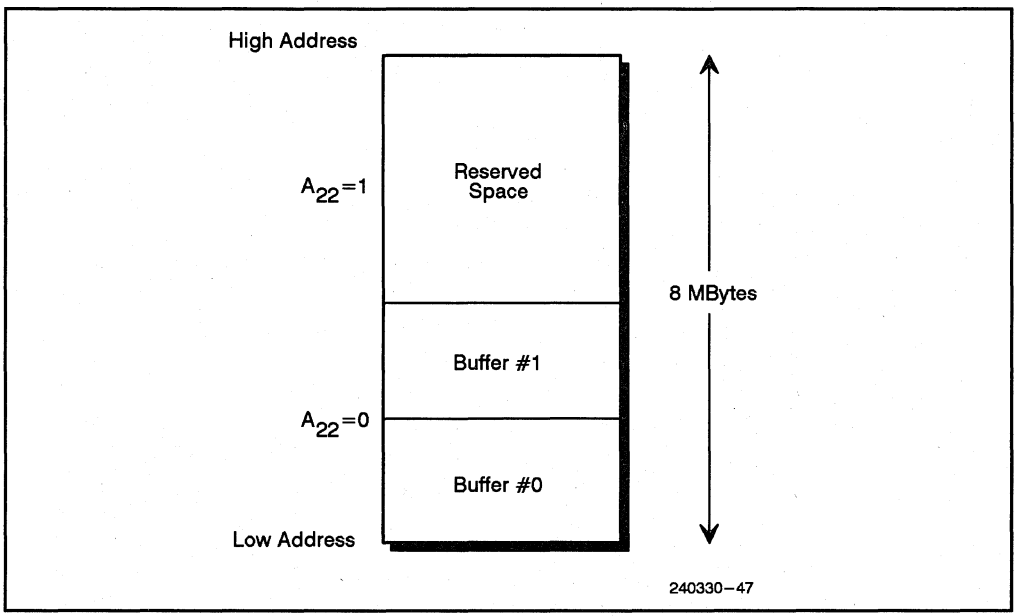


Figure 6-5. Expansion Space

6.5.3 CRT Timing Logic

The 64 MHz pixel clock (PCLK) keys all timing signals. A divide-by-16 clock is generated to clock the horizontal blank signal, HBLANK. As shown in Figure 6-6, each horizontal active period supports the 1,024 pixel horizontal display. Display active times are set at 16.5μs while inactive times are set at 4.5μs. The horizontal sync signal, HSYNC# has a front porch of 0.5μs, an active time of 1.5μs and a back porch of 2.5μs. The horizontal frequency is approximately 48 KHz. These timings are for the Hitachi Superscan (model CM2085MV); they must be adjusted for use with a different monitor.

The vertical blank signal, VBLANK# is active low and remains active for 45 HBLANK periods. To provide a 768 line display (by remaining inactive for 768 HBLANK clock periods), the vertical retrace rate becomes approximately 60 times/sec. The front porch, active time and back porch for the vertical sync signal (VSYNC#) are 6, 6 and 33 HBLANK periods respectively (refer to Figure 6-6).

Zeros are shifted into the shift registers during blanking periods. When blanking ends, display data is shifted out of the VRAMs and loaded into the shift register by the load/shift signal (LDSR#). Undefined data is not displayed. Figure 6-7 provides serial data clocking timing. The serial clock (SC) is held low after all required data has been moved from the shift registers at the end of the display period.

The pixel clock is divided by 16 to generate the CDIV16 clock used in most CRT timing logic. Final timings are synchronized to the pixel clock to ensure correct blanking by the video DACs where digital data, composite sync and blank signals combine to generate RGB signals.

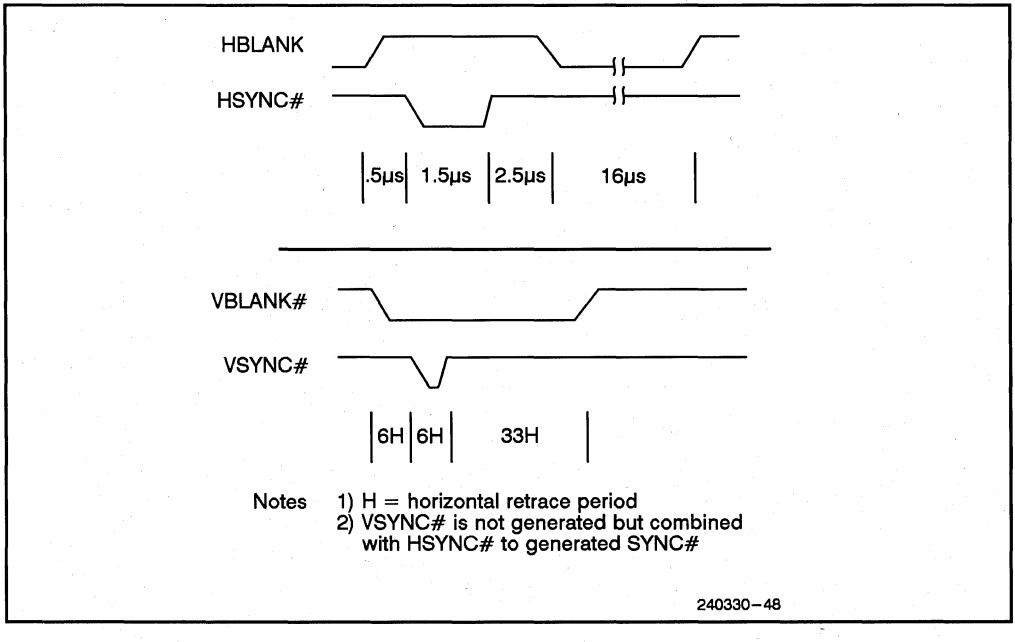


Figure 6-6. Blank and Sync Signals

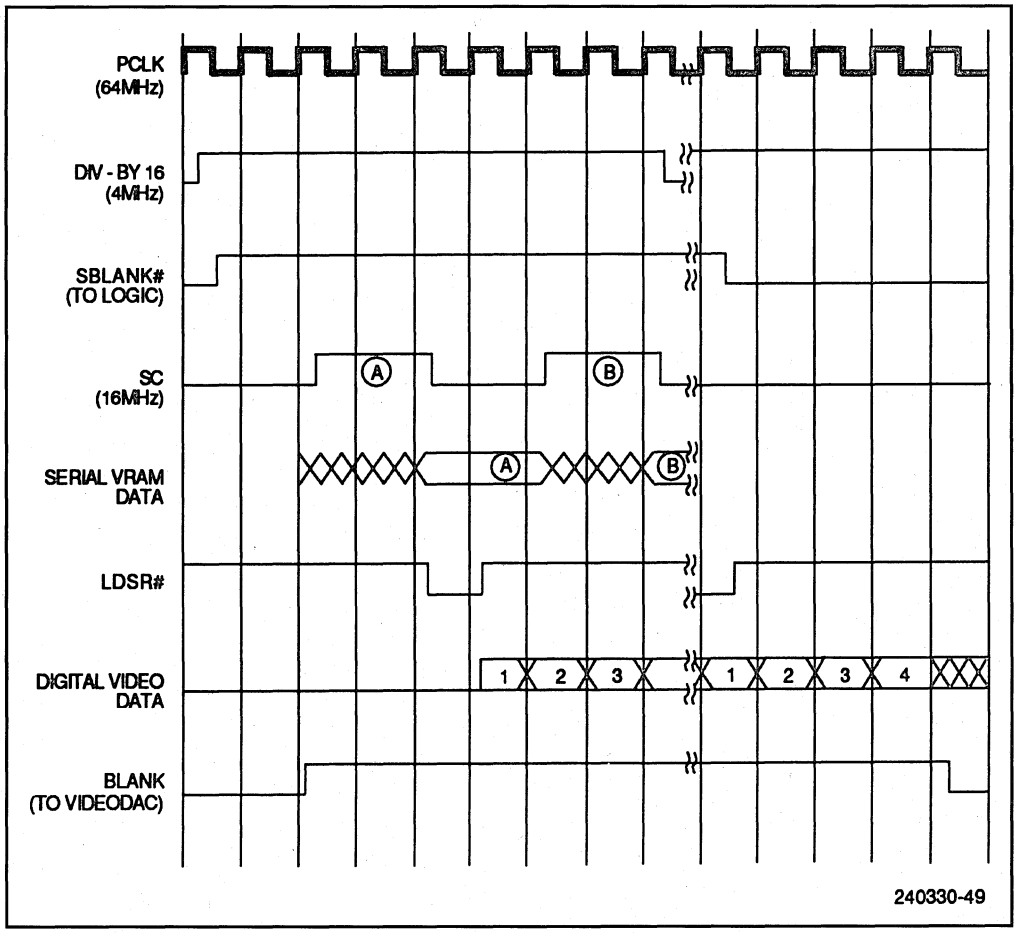


Figure 6-7. Serial Data Clocking

CRT timing relationships are illustrated in Figure 6-8. The horizontal blank (HBLANK) is generated by counting COUNT, and the vertical blank (VBLANK#) is generated by counting HBLANKs. The counters are reset by HCLR# at the end of horizontal display periods and by the VCLR# (not shown) at the end of each display periods.

6.5.4 Schematics Description

Refer to Appendix A, page 24, for the schematics.

Sheet #1

This sheet contains address transceivers and latches.

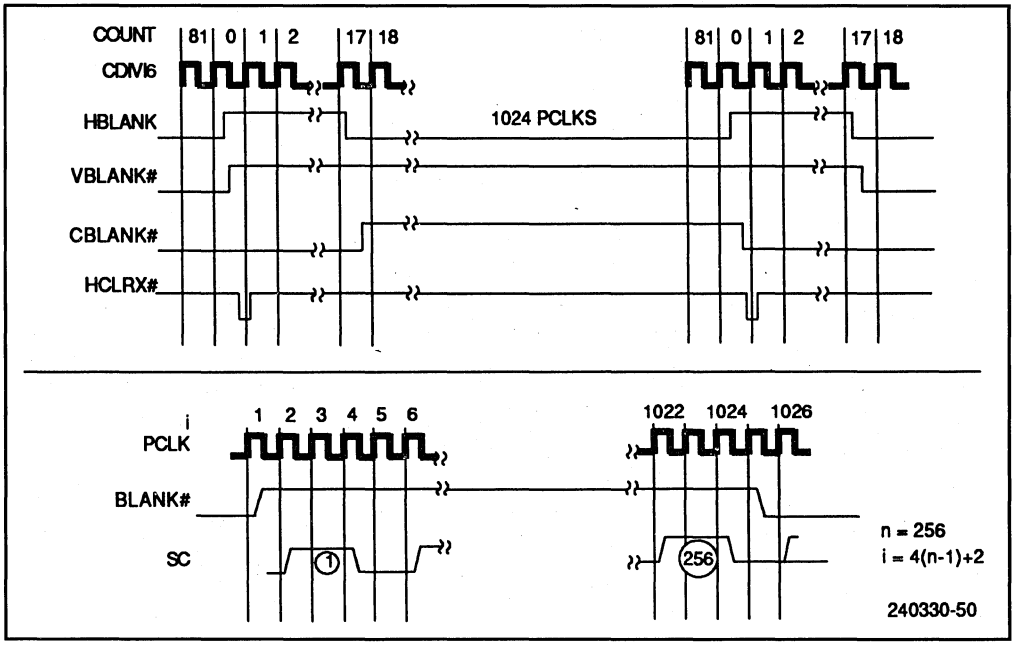


Figure 6-8. Blank Signals

Sheet #2

This sheet contains half of the VRAMs in Buffer #0.

Sheet #3

This sheet contains half of the VRAMs in Buffer #0.

Sheet #4

This sheet contains half of the VRAMs in Buffer #1.

Sheet #5

This sheet contains half of the VRAMs in Buffer #1.

Sheet #6

This sheet contains VRAM control logic. RAS1# and RAS0# control Buffers #1 and #0 respectively. RAS1# is activated when A21 is high; RAS0# is activated when A21 is low. RAS1# and RAS0# are combined to control the remaining logic.

CAS2#, CAS1# and CAS0# are activated together; they are generated to share loading.

PRECH# controls RASx# precharge time: three CLKs for far cycles; four clocks for refresh/SAM to support CAS#/DTOE# before RAS#. REFC# controls RAS# active time (five CLKs) during refresh/SAM. TRFQ# is activated when refresh (REF) or SAM request (TRQ) is active. RASDEL# is to delay RAS# active edge by one clock when TRFQ# is found active.

ROWE#/COLE# and SRAE#/SCAE# are address enabled for regular VRAM and serial port row/column addresses.

CPEND# is activated when VRAM cycles are detected. They are deactivated when CAS# is deactivated and no new VRAM cycle is detected. WDEL# delays logic activation when a write follows a read. NCLK# is added to minimize the number of inputs (related to WDEL#) to the RDY# (processor ready signal) PLD.

ERDL# is activated in the clock that VRAM data is valid. It causes RDL activation in the core logic in the same CLK.

DTOEx# enables VRAM output for read cycles and signals SAM transfer on the RASx# falling edge. CLRTRFQ# clears the refresh request (REF) while CLRTRQ# clears the SAM request (TRQ). ERDE# and EWDE# control direction (DRMDIR#) and enabling (DRMEN#) of data bus transceivers on the i860 microprocessor board. Refer to timing diagrams in the text for details.

Sheet #7

This sheet contains additional VRAM control and miscellaneous logic. DREF, PREF and TREF are decode signals to the product terms and determine the number of inputs required for DTOEx#, CASx# and RASx#, respectively.

EWDL# activates WDL in the core logic in the same CLK to capture processor write data. In the case of zero-wait-state writes, it is activated whenever a VRAM cycle is detected regardless of the read/write or near/far state.

SE1# and SE0# are activated when an access to the upper portion of the expansion space is detected. A low value on A3 activates SE0#; a high value activates SE1#. Upon reset, SE0# is activated. SE1# and SE0# cannot be active at the same time.

RDY# is activated for VRAM cycles and for buffer selection accesses to the upper portion of the expansion space. RDY# is always in the high state when floated. VSELX1 and VSELX0 are state variables for these access cycles.

EXPBSY# is activated when VRAM cycles are in progress. CBUSY# combines the various busy signals.

WEx# are write enable signals for VRAMs. They must be inactive to allow CAS-before-RAS refresh and SAM transfer on the RASx# falling edge. REFC# deactivates WEx#.

Sheet #8

LDSR# loads serial data into the shift registers. Data to be clocked out by PCLK. SCLR# clears register contents once all needed data has been shifted.

Sheet #9

The VOG signal carries composite sync information to support sync-on-green. When BLANK# is active, remaining shift register data is not displayed.

Sheet #10

PCLK is divided by 16 to generate CDIV16 for composite blank and sync clocking. SBLANK# is the composite blank signal synchronized to PCLK. The final blank signal BLANK# is used to clear the screen after remaining pixels are cleared from the shift register. The composite sync (SYNC#) and BLANK# are used by the video DACs.

Sheet #11

REF and TRQ generation. TRQ is activated by HBLANK; REF by REFREQ from the expansion bus.

Sheet #12

Terminations for VRAM control and for the expansion bus CLK signals (CLKC and CLKG).

MULTIBUS® II and the i860™ Microprocessor

7

CHAPTER 7
MULTIBUS® II AND i860™ MICROPROCESSOR

The i860™ microprocessor provides supercomputing performance and capability that can be provided on a standard bus platform. This allows systems integrators to take advantage of rapidly advancing CPU technology while preserving their investment in existing hardware. Boards based on the MULTIBUS® II system bus and on the i860 microprocessor can be added to existing systems or can be the basis for a new system design.

This chapter outlines an example of such a design. Schematics for the design are to be found in Appendix B. **This example has not been tested.**

7.1 i860™ MICROPROCESSOR CPU BOARD

Figure 7-1 shows a block diagram of a board designed around the i860 microprocessor and MULTIBUS II system bus. The main features include the following:

- MULTIBUS II System Bus Interface
- i860 Microprocessor

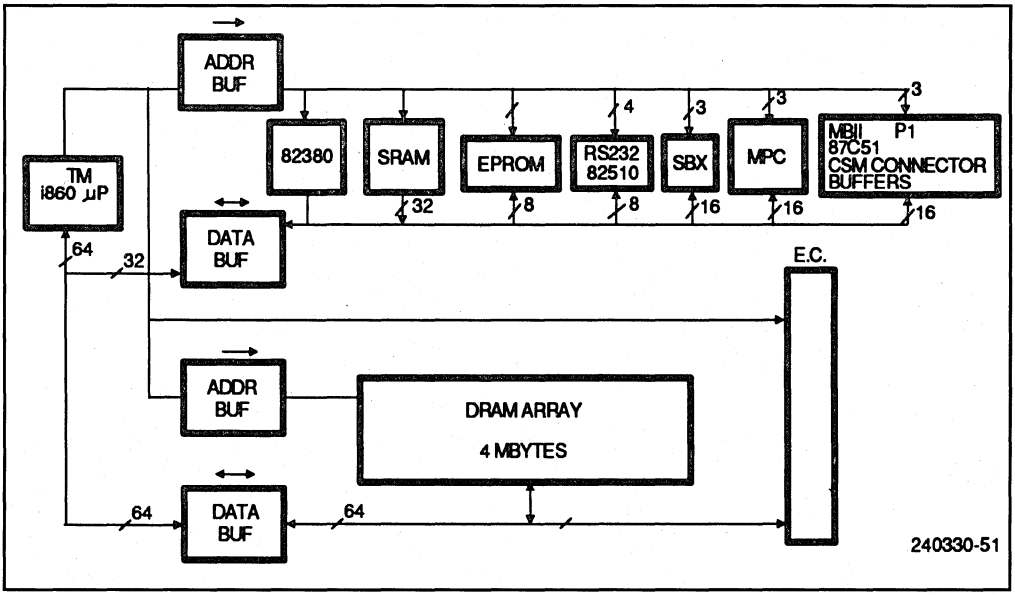


Figure 7-1. System Block Diagram of i860™ Microprocessor Based MULTIBUS® II Board

- DRAM Main Memory
- Local I/O Devices
  - 82380 Integrated Systems Peripheral
  - 82510 UART
  - SBX Connector
  - Single 8-bit Boot EPROM
- DMA Control and SRAM Message Area
- Memory and Graphics Expansion Connector

7.2 MULTIBUS® II SYSTEM BUS STANDARD

The MULTIBUS II system bus standard is a processor-independent bus architecture with full distributed multiprocessor support. The standard defines a 32-bit parallel system bus with a maximum throughput of 40 Megabytes per second. The parallel system bus is isolated from the CPU local bus to allow the i860 microprocessor to access memory on its 64-bit data bus. The parallel system bus handles interprocessor communications and accesses to I/O devices not dedicated to a single CPU board.

The MULTIBUS II system bus architecture also includes the system expansion (iSBX™) I/O bus. Refer to the IEEE 959 specification for a full description of this bus.

7.2.1 Parallel System Bus (PSB)

The parallel system bus (PSB) is optimized for standardized interprocessor data transfer and signalling. Its burst transfer capability provides a maximum sustained bandwidth of 40 megabytes per second for high-performance data transfers. A hardware recognized data type, called a packet, is used to ensure consistent and reliable transfers between different system boards.

The PSB supports four address spaces for each bus agent (board that encompasses a functional subsystem). Conventional I/O and memory address spaces are included, as are two address spaces that support system functions:

- A 255-address message space supports message passing. Microprocessors typically perform interprocessor communication inefficiently. Message passing allows two bus agents to exchange blocks of data at full bus bandwidth without microprocessor supervision. An intelligent bus interface capable of message passing shifts the burden of interprocessor communication away from the processor and into hardware dedicated to this task, thus enhancing overall system performance.
- An interconnect space allows geographic addressing – the identification of any bus agent (board) by slot number. Every system based on the MULTIBUS II system bus contains a central services module (CSM) that provides system services for all bus agents residing on the PSB bus. These services include uniform initialization and bus timeout detection.The CSM may use the interconnect space registers of each bus agent to configure the agent dynamically. Stake pin jumpers, DIP switches and other hardware configuration devices can thus be eliminated.

Three types of bus cycles define activity on the PSB bus:

- Arbitration Cycle – Determines the next owner of the bus. This cycle consists of a resolution phase in which competing bus agents determine priority for bus control. It also includes an acquisition phase in which the bus agent with the highest priority initiates a transfer cycle. This cycle overlaps other cycles allowing agents to transfer bus control on back-to-back cycles.
- Transfer Cycle – Performs a data transfer between the bus owner and another bus agent. In the request phase, address control signals are driven. In the reply phase, two agents perform a handshake to synchronize the data transfer. The reply phase is repeated and data transfers continue until the bus owner ends the transfer cycle.
- Exception Cycle – Indicates that an exception (error) has occurred during a transfer cycle. In the signal phase, an exception signal from one bus agent causes all other bus agents to terminate any arbitration and transfer cycles in progress. In the recovery phase, the exception signals go inactive. A new arbitration cycle can begin on the clock cycle following the recovery phase.

Figure 7-2 illustrates how the timing of these cycles overlap.

7.2.2 Message Passing Coprocessor

The interface from the processor's local bus to the PSB can be simplified with the Intel 82389 message passing coprocessor (MPC). The MPC has been designed for message passing protocols of the MULTIBUS II system bus architecture. It participates in the entire PSB bus protocol and performs bus arbitration, transfer control, error detection and reporting, and parity generation and checking. These functions occur independently of the host CPU.

The MPC decouples local bus activities from interprocessor communications over the PSB bus. The decoupled bus approach has two advantages:

- Resources that would be held in wait-states while dedicated bus access arbitration is underway are instead free. This parallelism increases system performance.
- The bandwidth of one bus does not limit the transfer rate of another. Each bus can perform full-speed, synchronous transfers.

As shown in Figure 7-3, the MPC signals can be divided into three functional groups:

- PSB interface
- Local bus interface
- DMA interface

These signal groups are discussed below.

7.2.2.1 MPC INTERFACE TO PSB

The primary functions of PSB interface signals are arbitration and system control. Five bidirectional arbitration signals (ARB5–ARB0) are used during reset to read card-slot ID and arbitration ID from the CSM. During arbitration, these signals output the arbitration ID for priority resolution. Bus request (BREQ#) is a bidirectional signal.

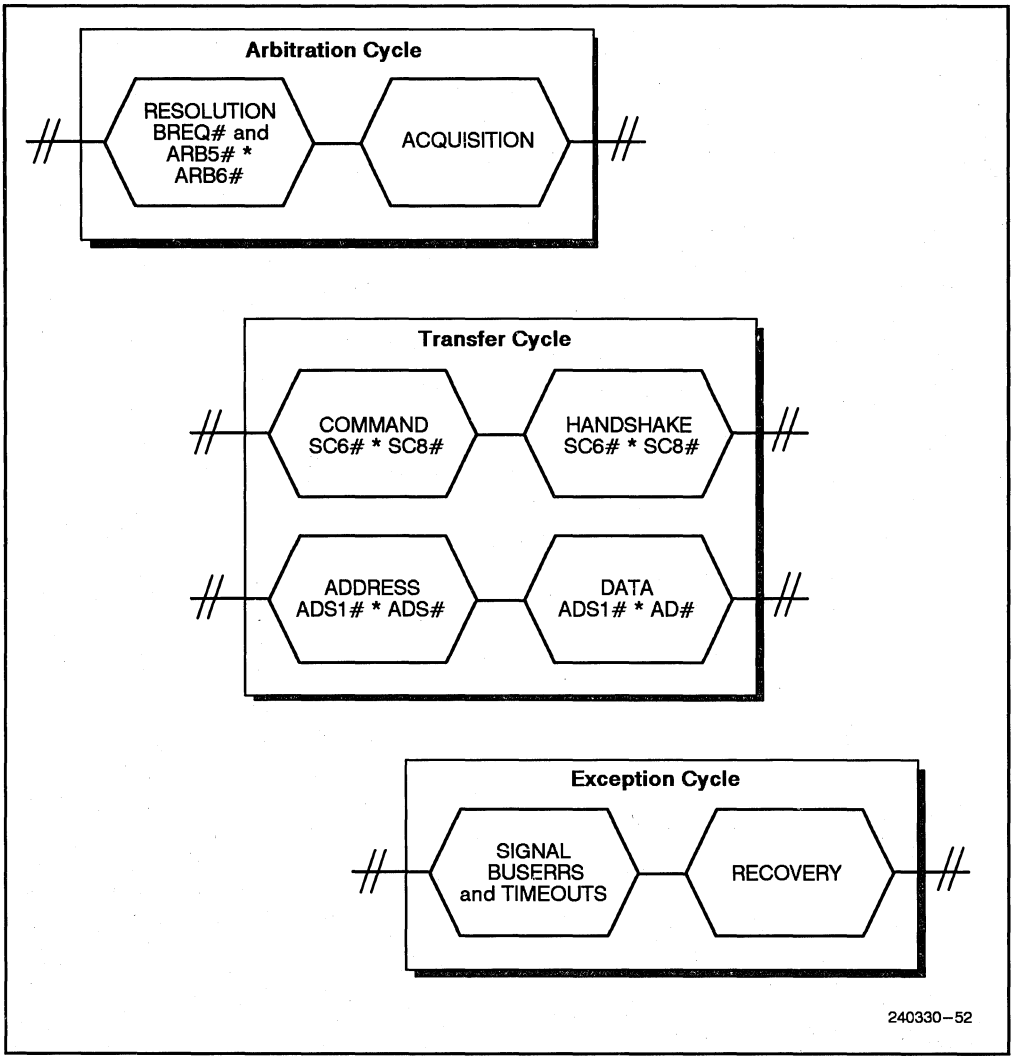


Figure 7-2. PSB Bus Cycle Timing

(It should not be confused with the BREQ# i860 microprocessor signal.) Each bus agent asserts BREQ# to request control of the bus and samples BREQ# to determine if other agents are also contending for bus control.

Bus error (BUSERR#) is a bidirectional signal that a bus agent sends to all other bus agents when it detects a transfer cycle parity error. The CSM sends the bus timeout signal (TIMOUT#) to all bus agents when a bus cycle fails to end within a prescribed time period.

Ten system control signals (SC9#–SC0#) coordinate transfer cycles. The MULTIBUS II Architectural Specification defines each signal. Directional enables (SCOEH and

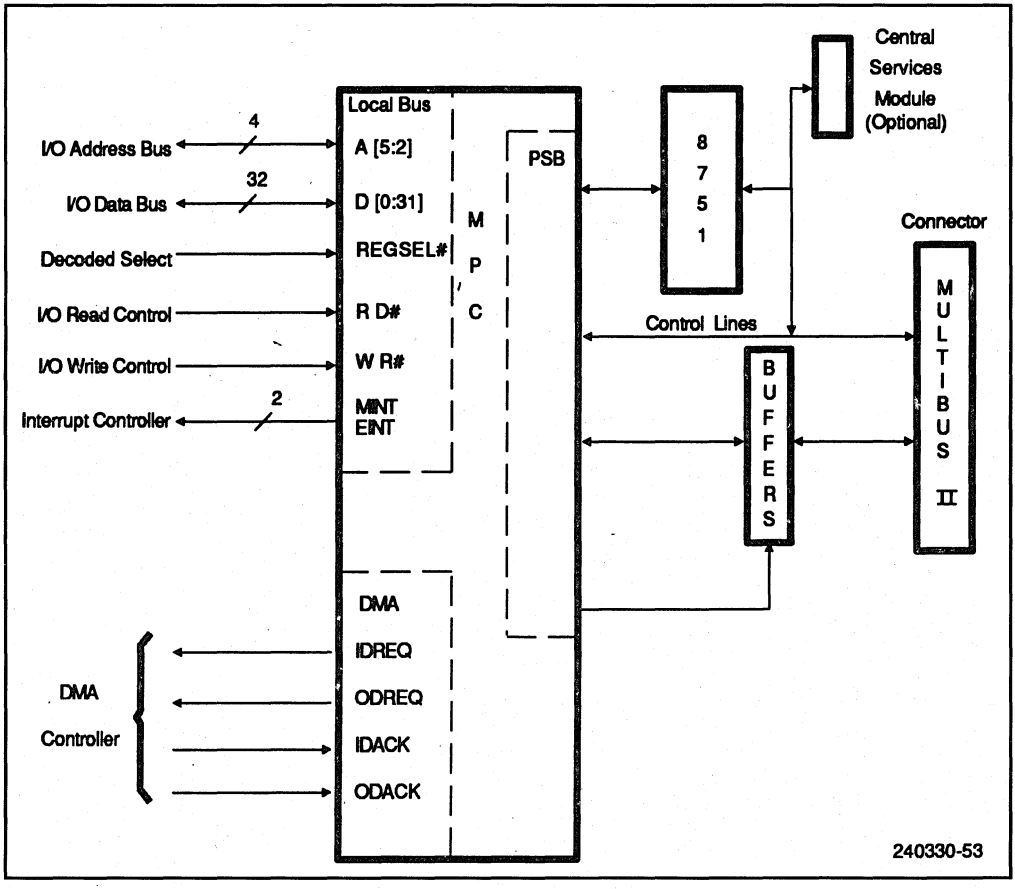


Figure 7-3. MPC Signal Groups

SCOEL) are provided to let transceivers buffer these bidirectional signals. The MPC checks byte parity lines (PAR3–PAR0) are for incoming operations and set the parity lines for outgoing operations.

Other PSB signals are reset (RST#), reset-not-complete (RSTNC#) and ID latch (LACH*n*#, *n* = slot number). These signals are used only during system initialization.

The MPC coordinates interrupt handling for a bus agent on the PSB bus. Interrupts are implemented as virtual interrupts in the message space. To send an interrupt message, the processor writes to the MPC to indicate the source destination and message type. The MPC coordinates the interrupt message transfer.

The PSB interface consists of the multiplexed address/data bus (AD31#–AD0#). The MPC controls external buffers used to drive the PSB. As a requesting agent, the MPC drives addresses and data at appropriate times. As a receiving agent, the MPC monitors the address/data bus for its address. When it recognizes one of its own addresses, the MPC performs the required handshake and reads the message into the message queue.

The MPC then, if necessary, interrupts the i860 microprocessor to indicate that the message is pending in the queue. The processor reads the message and services the interrupt accordingly.

7.2.2.2 MPC LOCAL BUS INTERFACE

The local bus interface of the MPC is like that of any other simple I/O device consisting of select lines, address signals and read and write control lines. To support message passing, MPC registers have various functions. They are accessed by asserting REGSEL# and the appropriate register address while performing a read or write cycle. Among other functions, the registers are used to program data message transfers, to receive and send control transfers and handle errors.

7.2.2.3 MPC DMA INTERFACE

The DMA interface of the MPC has two channels that use the standard DREQ (DMA Request) DACK (DMA Acknowledge) hardware transfer protocol. The two channels are dedicated to the MPC one as the input channel and one as the output channel. Each has its own control lines and operates independently.

MULTIBUS II uses a message passing protocol for data transfer. Control transfers (also called unsolicited messages) are up to 32 bytes long, and data transfers (also called solicited messages) are up to 16 Mbytes long and are split into 32 byte packets by the sending and receiving MPCs. The sending and receiving DMA controllers do not know that the data is being packetized on the system bus. This bus bandwidth preserving feature does not affect local data transfers. The MPC has 32-byte internal FIFOs for packaging data before sending it on the MULTIBUS backplane.

The board includes an SRAM message area to isolate back plane data rates from the processor's local bus. For a solicited outgoing message, the i860 microprocessor transfers messages into the SRAM at a high rate. The DMA channel then transfers messages from the SRAM to the MPC's outgoing message FIFO. For a solicited incoming message the DMA channel transfers the message into the SRAM buffer and signals the processor when it has completed. The processor can then very quickly transfer from the SRAM area into main memory. As discussed later, using the SRAM as a dedicated DMA buffer area has several advantages over DMA directly out of main memory.

7.3 i860™ MICROPROCESSOR BUS INTERFACE

The i860 microprocessor has a synchronous interface with nonmultiplexed address and data buses. The data bus is 64 bits wide and the address bus provides 32-bit addressing. Addressing consists of 29 address lines and separate byte enable for each of eight data bytes. The bidirectional data bus can accept or drive new data on every other clock, yielding a bandwidth of 160 megabytes per second at 40 MHz. The bus allows two levels of pipelining that may be selected on a cycle by cycle basis. In pipelined mode, a new cycle is started before earlier cycles have completed. Chapter 3 provides a complete description of the i860 microprocessor bus interface.

7.4 DRAM SYSTEM

The memory system consists of static column mode, noninterleaved, parity checked DRAMs. The processor's pipelined bus and address and data latches combine to hide DRAM latency for most accesses. Using 80 nanosecond static-column DRAMs, the memory system can supply the maximum data bandwidth required by the processor.

The board allows up to four megabytes of DRAM on the base board. DRAMS are static column, 80 nanosecond, 256K × 4 chips in ZIP sockets. Parity DRAMS are 1M × 1. An additional four megabytes of DRAM can be added to the expansion connector without additional parity DRAMS. Chapter 4 provides more information about the DRAM interface.

7.5 LOCAL I/O SYSTEM

The I/O system consists of an integrated systems peripheral (82380), a serial port (82510), one 8-bit boot EPROM and an iSBX bus connector. The 82380 integrated systems peripheral includes timers (8254), two interrupt controllers (8259 master and slave) and eight DMA channels. The I/O system is memory mapped.

The serial port can be used in a polled or interrupt mode. It also suitable as a console monitor. Timers are clocked from a reduced version of the serial controller's oscillator module. A control port enables and disables the timers. Timer outputs are connected as 8259 interrupts. The master and slave 8259 chips accommodate 15 interrupt sources.

Four of the DMA channels are used. Two are for the MPC interface, and two are for the iSBX bus connector. The DMA channels can transfer into or out of the SRAM message area.

7.5.1 82380 Integrated Systems Peripheral

The 82380 integrated systems peripheral serves both as a slave I/O device and as a bus master DMA controller. The 82380 contains four 16-bit programmable timers and has connections for 15 external interrupts. In addition, there are five internal interrupts that can be used with the DMA channels and the timers.

The 82380 uses a double frequency clock. This allows a 40 MHz CPU to operate synchronously with a 20 MHz 82380. At reset, a phase clock is generated and used by the control logic for 82380 accesses. The port addressing and data bus requirements of interfacing the 64-bit i860 microprocessor to the 32-bit 82380 are discussed in Chapter 5.

The DMA channels are programmed with the 82380 in slave mode. Once programmed, the DMA channels become bus masters to complete transfers. To gain control of the bus, the 82380 asserts HOLD. When HOLDA is returned to the 82380, it initiates the DMA transfer. Transfers continue until completion or until HOLDA is deasserted.

7.5.2 iSBX™ Bus Connector

An iSBX bus connector is also included. The DMA control interface can be configured as defined in the iSBX bus specification. Several jumpers are provided for connection configuring. To access 8- and 16-bit iSBX bus devices, the processor must address them on eight byte boundaries.

DMA transfers with the iSBX bus are supported. Transfers are with the SRAM message area. The DMA controller supports byte assembly and unassembly from 8 or 16 bit to 32 bits.

7.6 DMA CONTROL AND THE SRAM MESSAGE SYSTEM

DMA controllers benefit systems that have peripherals requiring block memory transfers. In this system, DMA channels are used for incoming and outgoing MULTIBUS II system bus messages and for I/O devices on the iSBX bus connector. Once a DMA channel is programmed, the processor remains uninterrupted while block transfers complete.

DMA performance depends on the DMA controller transfer rate and the effect of the DMA controller on processor memory bandwidth. The relative influence of these factors varies according to the task at hand. If the processor waits for DMA data, then transfer rate is most important. In a multitasking environment, the DMA effect on bandwidth is more critical. A multitasking system puts a task waiting for the DMA to sleep and continues to execute other tasks. Overlapping tasks in this way improves overall system performance. Any DMA method must also consider the write-back caching protocols of the CPU and that the physical memory is arranged as pages.

I/O system devices are slow relative to the processor's local bus. The DMA controller is separated from this bus by address and data buffers. This allows DMA memory accesses to occur independently of processor memory accesses. The memory portion of DMA transfers accesses the SRAM message area. Direct transfers to main memory would have to place the processor in a hold state, could not cross page boundaries and would require the operating system to maintain memory consistency between the caches and the DMA areas of memory. Transfers into the SRAM area do not initiate a hold, but they do require the processor to perform a main memory data transfer once the DMA transfer completes. The processor retains full memory bandwidth during DMA transfers to the SRAM area and may continue to execute tasks. Transfers between SRAM and DRAM area execute very quickly because SRAM and DRAM accesses occur at full bus bandwidth. DMA transfers into the SRAM can be programmed to transfer blocks larger than the 4 Kilobyte page size.

7.6.1 DMA Channels

The 82380 is the system DMA controller and contains eight DMA channels. DMA interface devices use the standard DREQ/DACK protocol. Bus control logic of the 82380 is identical to the 386™ microprocessor and similar to the i860 microprocessor. The 82380 can perform data assembly and disassembly for 8- and 16-bit DMA devices. The maximum data width for the device is 32-bits, and it uses address line A2 for accessing within 64-bit processor words.

Fly-by mode may be used for transfers between the I/O and SRAM systems. In this mode, SRAM reads and MPC writes occur together to produce the highest possible DMA transfer rate. Fly-by mode can only be used with 32-bit DMA devices such as the MPC. Figure 7-4 shows two fly-by transfers between the SRAM and the output data channel of the MPC. In the second cycle, the MPC deasserts ODREQ, indicating to the DMA channel that its FIFO is full and that it cannot accommodate another transfer. The DMA channel resumes transfers when ODREQ is again asserted.

The 82380 can assemble and disassemble 8-bit I/O accesses to 32-bit memory accesses during accesses to both main memory and the SRAM message area. Address line A2 of the 82380 determines which half of 64-bit memory data is accessed.

7.6.2 SRAM Message Area

The SRAM message area is ideally suited for isolating DMA traffic from the i860 microprocessor's local bus. DMA transfers may occur in parallel with CPU main memory accesses. Both the DMA controller and the CPU can act as bus masters to access the SRAM and the I/O bus.

The SRAM message area can be designed for various levels of price/performance. Two implementations are shown in Figures 7-5 and 7-6. The first figure illustrates an approach that requires fewer parts and uses four 8-bit SRAMs. In this example, the SRAM

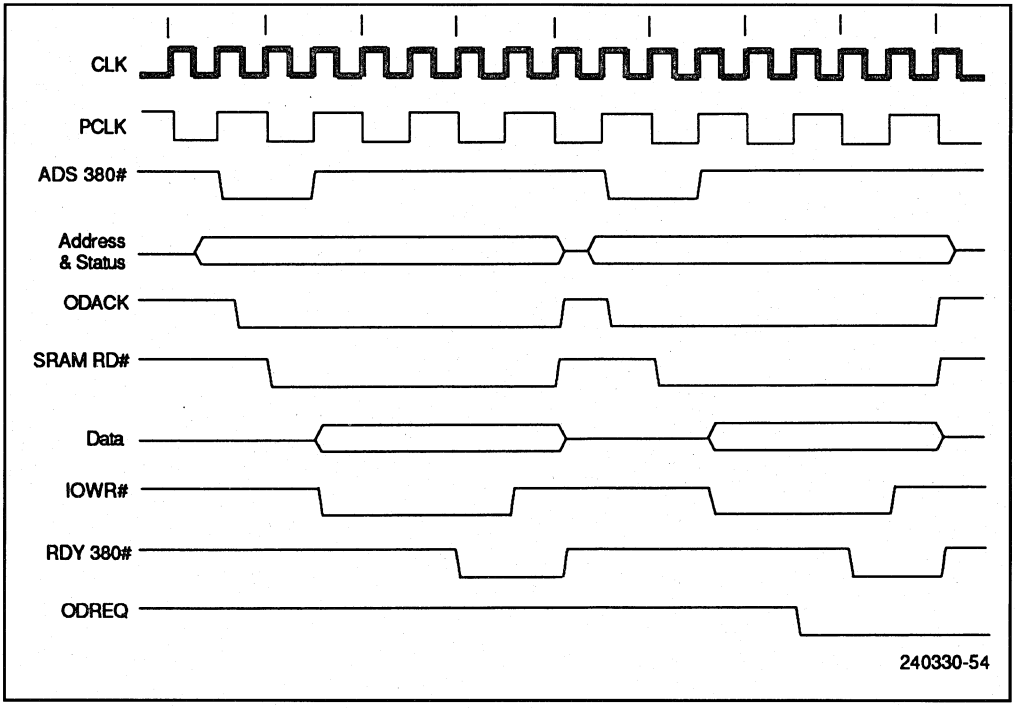


Figure 7-4. Fly-by Transfer with SRAM Read and I/O Write

data bus is directly connected to the I/O data bus. Four data buffers are needed on the processor side. Only the lower 32-bit processor data lines are used. This reduces the processor to memory transfer rate by one-half. The processor may continue to access the SRAM with zero wait-states, but it may only access the lower half of the data bus. The auto-increment addressing mode of the processor is used, and full processor bus bandwidth is maintained.

The second example supports a full 64-bit processor interface. Four additional data buffers are needed on the processor side, the I/O side of the bus requires four additional SRAMS and data buffers. The I/O-side buffers are used to multiplex the halves of the 64-bit wide SRAM to one 32-bit I/O data bus. Eight 256 Kbit SRAMs produce a 256 Kbyte message space. 64 Kbit SRAMs can be used instead if less message space is required.

Arbitration between the two bus masters is performed by a PLD. The PLD can grant control of the DMA message system to the 82380 by asserting HOLD. If the i860 microprocessor begins a bus cycle that requires the DMA message system, the arbitration PLD forces the 82380 from the bus.

Figure 7-7 shows arbitration and DMA control logic blocks. The 82380 asserts HOLD and the arbitration PLD returns HOLDA. The 82380 takes control of the bus and performs DMA cycles until HOLDA is deasserted or until the transfer is completed. If HOLDA is deasserted, the 82380 relinquishes control of the bus. The arbitration PLD

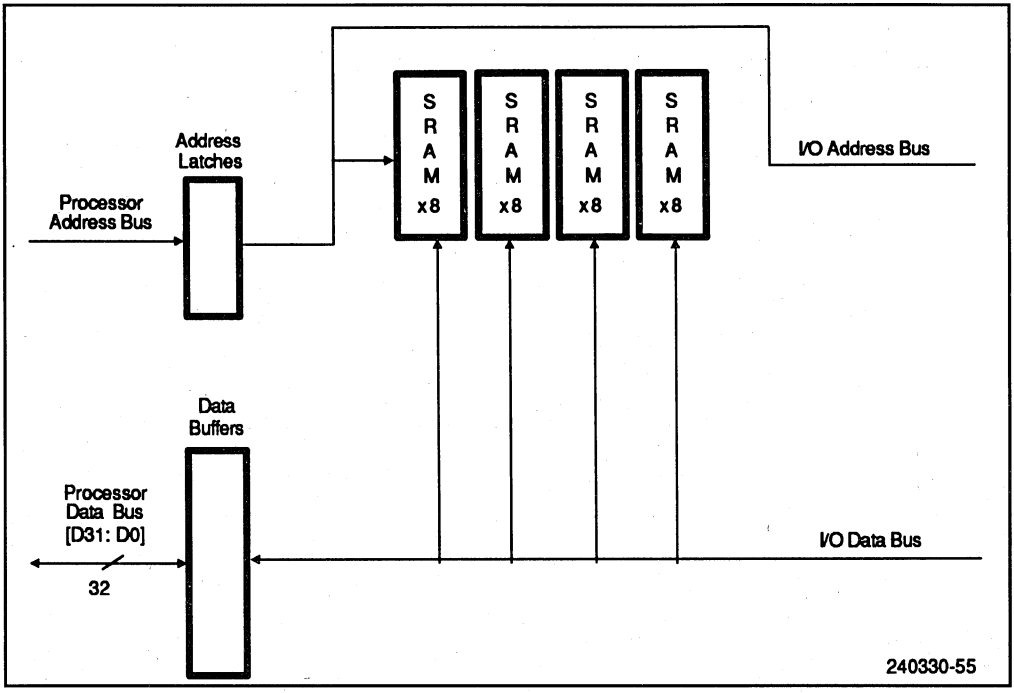


Figure 7-5. SRAM Message Area using 32-bit Bus

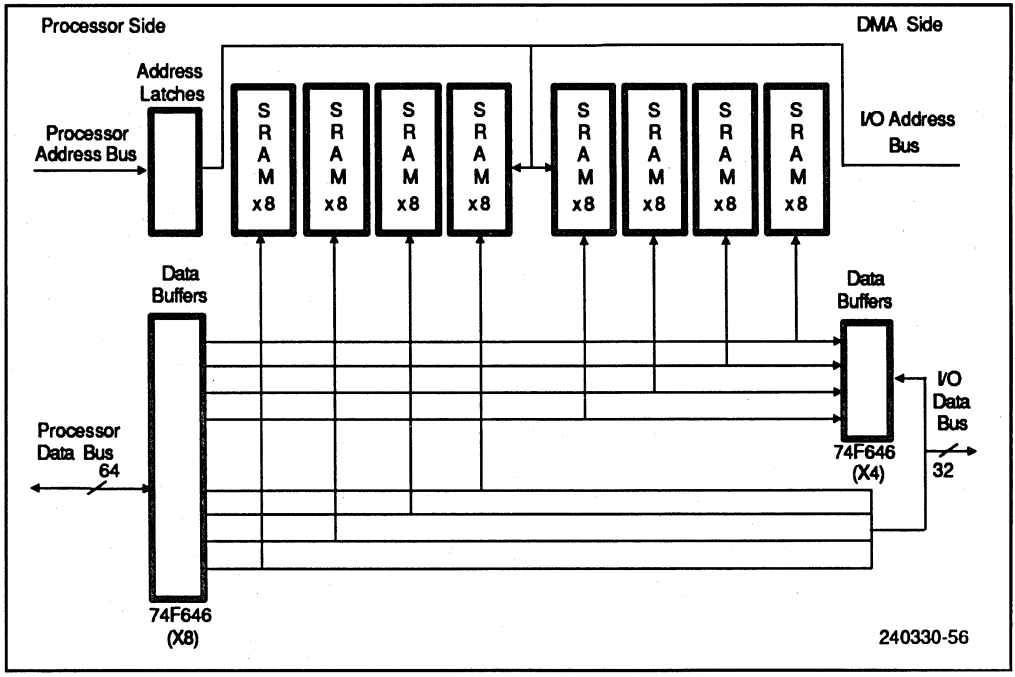


Figure 7-6. SRAM Message Area using 64-bit Bus

deasserts HOLDA to the 82380 and waits for it to deassert HOLD. Although HOLD becomes active again in the next clock, HOLDA is not returned to the 82380 until the processor is finished with the bus. The 82380 is programmed in demand mode to allow this type of arbitration to work. When the processor finishes its I/O cycle, it returns HOLDA to the 82380.

7.7 EXPANSION CONNECTOR

The expansion connector can support memory expansion ranging from simple extensions to complex extensions such as those found in graphics systems. DRAM control lines and processor bus signals are both available on the connector.

7.7.1 Memory Expansion

Additional memory can be added to the expansion board. In this design, the processor address bus must be buffered on the memory board. The address buffers perform address multiplexing and provide DRAM drive capabilities. Control signals are available on the connector, and DRAM and address buffer control signals can be generated using simple decode logic. The data bus uses data buffers from the main board.

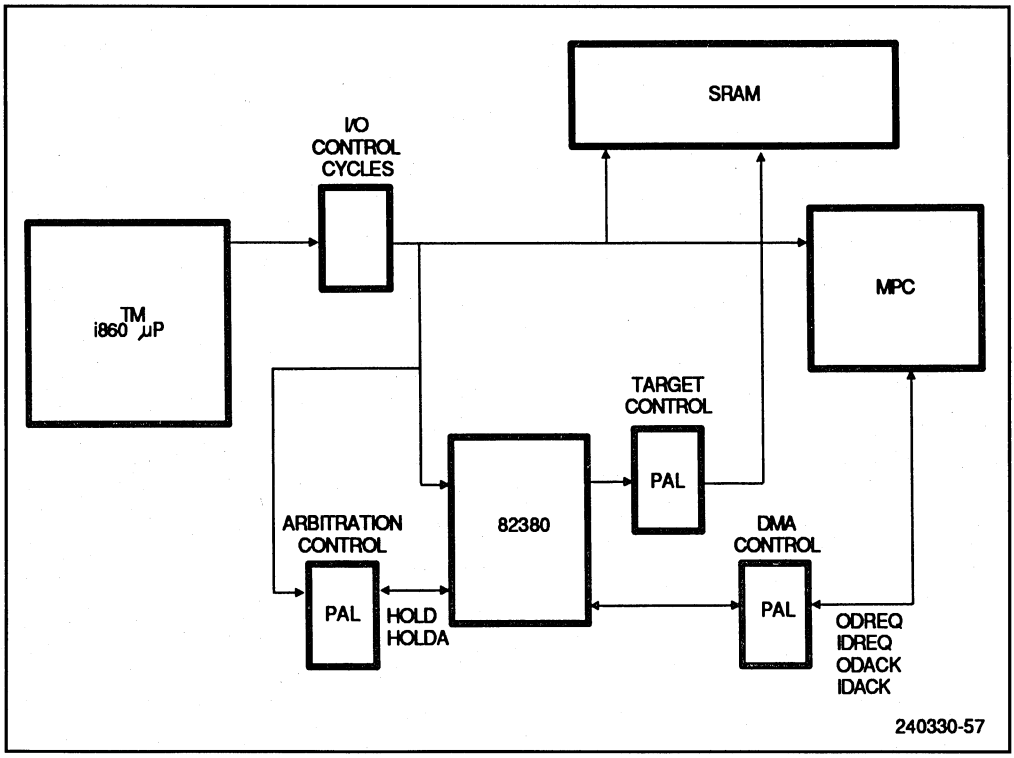


Figure 7-7. DMA System Arbitration and Control

7.7.2 Intelligent Expansion

The connector provides the control signals needed to build an intelligent expansion board. One example of an intelligent board is a frame buffer board. It requires its own VRAM controller and control logic for returning READY# and NA# to the processor. Motherboard data buffer control signals are also available. The controller tracks ADS# and EXPSEL# signals to know when a cycle is intended for the expansion board controller. The controller uses the EXPBSY# signal to indicate that it will be asserting the READY# and NA# control lines. The base controller must disable control of these lines when this occurs.

8 Physical Design and Debugging

CHAPTER 8
PHYSICAL DESIGN AND DEBUGGING

This chapter outlines the basic design issues, ranging from power and ground issues to achieving proper thermal environment for i860™ CPU.

8.1 GENERAL DESIGN GUIDELINES

The performance and proper operation of any high-speed system greatly depends upon proper physical layout. This section gives a brief overview of general issues and design guidelines for layout which are significant to both high- and low-frequency systems.

The ever-increasing improvement of integrated circuit technology has led to an enormous increase in the number of functions that are being implemented on a single chip. Improved technology also allows these functions to be implemented to provide higher performance. The i860 microprocessor, with operating frequencies of 33 MHz/40 MHz and corresponding high edge rates, presents a challenge to the conventional interconnection technologies which to date have been adequate only for interconnecting less sophisticated VLSI devices. This challenge applies especially to system designers who are responsible for providing suitable interconnections at the systems level.

At higher frequencies, the interconnections in a circuit behave like transmission lines which degrade a system's overall speed and distort its output waveforms. In laying out a conventional printed circuit board, there is freedom in defining the length, shape and sequence of interconnections. But with high-speed devices like the i860 microprocessor, this task is carried out with careful planning, evaluation and testing of the wiring patterns. It is also critical to understand the physical properties of transmission lines because interconnection at high edge rates is analogous to a broadcasting transmission line.

8.2 POWER DISSIPATION AND DISTRIBUTION

The i860 microprocessor uses fast one-micron CHMOS* IV process technology. The main difference between the previous HMOS microprocessors and the new ones is that, in the latter, power dissipation is primarily capacitive and there is almost no DC power dissipation. As power dissipation is directly proportional to frequency, accommodating high-speed signals on printed circuit boards and through the interconnections is very critical. The power dissipation of the VLSI device in operation is expressed by the sum of the power dissipation of the circuit elements, which include internal logic gates, I/O buffers and cache RAMs. It is also a function of the operating conditions.

The worst-case power dissipation of any VLSI device is estimated in the following manner:

1. Let the following symbols stand for estimates of typical power dissipation for each circuit element:

   P_G = Typical power dissipation for internal logic gates (mW)

   $P_{I/O}$ = Typical power dissipation for I/O buffers (mW)

   P_{CRAM} = Typical power dissipation for instruction/data cache RAMs (mW)

2. To estimate total typical power dissipation:

   $P_T = P_G + P_{I/O} + P_{CRAM}$ (mW)

   where P_T is the total typical power dissipation in milliwatts.

3. To estimate the worst case power dissipation:

   $P_d = P_T \times C_v$ (mW)

   where P_d is the worst case power dissipation (mW) and C_v is a multiplier that is dependent upon power supply voltage.

Internal logic power dissipation varies with operating frequency and to some extent with wait-states and software. It is directly proportional to the supply voltage. Process variations in manufacturing also affect the internal logic power dissipation, although to a lesser extent than with the NMOS processes.

The I/O buffer power dissipation, which accounts for roughly 10 to 25 percent of the overall power dissipation, varies with the frequency and the supply voltage. It is also affected by the capacitive bus load. The capacitive bus loadings for all output pins is specified in the *i860™ 64-Bit Microprocessor Data Sheet*. The i860 microprocessor's output valid delays will increase if these loadings are exceeded. The addressing pattern of the software can affect I/O buffer power dissipation by changing the effective frequency at the address pins. The frequency variations at the data pins tend to be smaller; thus, a varying data pattern should not cause a significant change in the total power dissipation.

To calculate the total power dissipated by the board, the following formulas can be used.

To calculate the maximum statistical power:

$$P_{typical1} + P_{typical2} + \ldots (P_{max1} - P_{typical1})^2 + (P_{max2} - P_{typical2})^2 + \ldots$$

where $P_{typical_i}$ and P_{max_i} are the typical and maximum power dissipation of each of the integrated circuits on the board. The i860 microprocessor should be placed close to the fan or where the airflow is unrestricted.

8.2.1 Power and Ground Planes

Today's high-speed CMOS logic devices are susceptible to ground noise and the problems that this noise creates in digital system design. This noise is a direct result of the fast switching speed and high drive capability of these devices, which are requisites in high-performance systems. Logic designers can use techniques designed to minimize this problem. One technique is to reduce capacitance loading on signal lines and provide optimum power and ground planes.

Power and ground lines have inherent inductance and capacitance, which affect the total impedance of the entire system. Higher impedances reduce current and therefore offer reduced power consumption, while low impedance (e.g. ground planes) minimize problems like noise and cross talk. Hence, it is very important for a designer to have a controlled impedance design where high speed signals are involved. The formula for impedance Z, given the inductance L and the capacitance C, is as follows:

$Z = (L/C)^{1/2}$.

The total characteristic impedance for the power supply can be reduced by adding more lines. For multilayer boards, power and ground planes must be used in i860 microprocessor designs.

The effect of adding more lines to reduce impedance is illustrated in Figure 8-1, which shows that two lines in parallel have half the impedance of a single line.

To reduce impedance even further, more lines should be added. To lower the impedance a greater number of lines or a plane should be used. Planes also provide the best distribution of power and ground.

The i860 microprocessor has 24 power (V_{cc}) and 24 ground (V_{ss}) pins. All power and ground pins must be connected to their respective planes. Ideally, the i860 microprocessor should be placed at the center of the board to take full advantage of these planes. Although the i860 CPU generally demands less power than conventional devices, the possibility of power surges is increased due to the processor's higher operating frequency and wide address and data buses. Peak-to-peak noise on V_{cc} relative to V_{ss} can be no more than 400 mV, and preferably is no more than 200 mV.

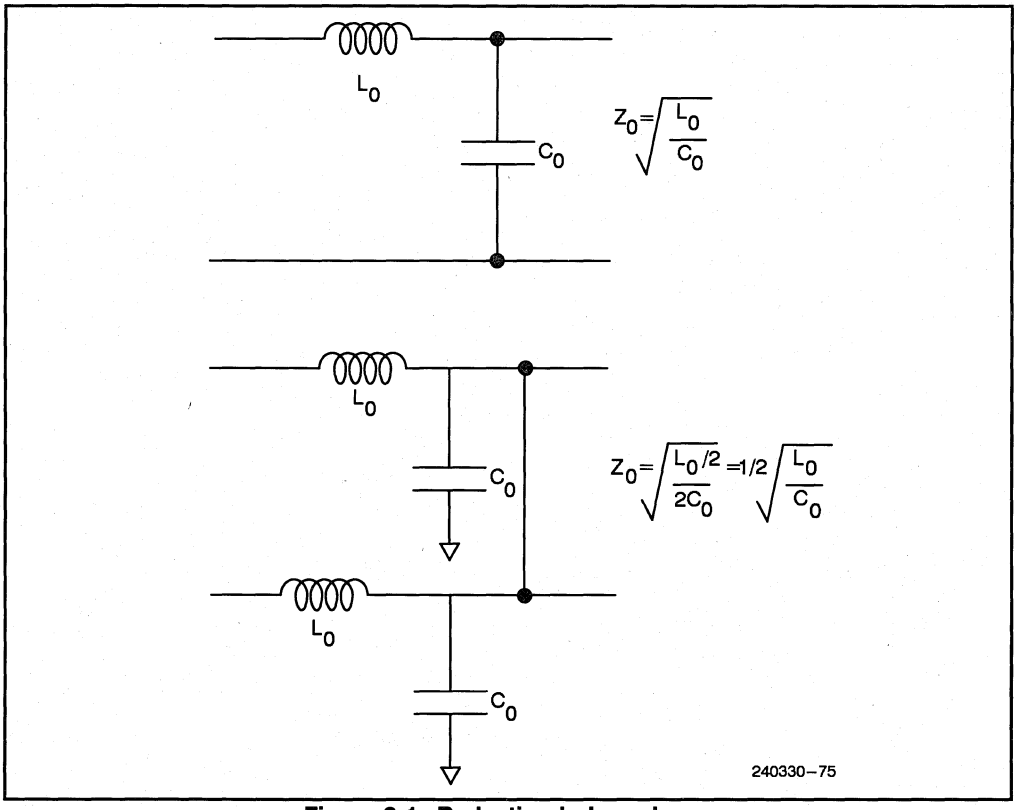


Figure 8-1. Reduction in Impedance

Although power and ground planes are preferable to power and ground traces, double-layer boards present a need for routing of power and ground traces.

The inductive effect of a printed-circuit board (PCB) trace can be reduced by bypassing and careful layout procedures to minimize inductances. Figure 8-2 shows a method for reducing the inductive effects of PCB traces. The power and ground trace layout has a low series inductance, because the loop area between the integrated circuits (ICs) and the decoupling capacitors is small and the power and ground traces are closer. This results in lower characteristic impedance, which in turn reduces the line voltage drop.

Another placement technique is called orthogonal arrangement, which requires more area than the previous technique but produces similar results. This arrangement is shown in Figure 8-3 . These techniques reduce the electromagnetic interference (EMI), which is discussed in section 8.4.3.1.

8.3 DECOUPLING CAPACITORS

The advanced, high-speed CMOS logic families available today have much higher edge rates than do the older, slower logic technologies. The switching speeds and drive capability needed to provide high performance to the systems are also associated with increased noise levels. Some noise levels are inconsequential because they fall within the switching times of the other devices. The switching activity of one device can propagate to other devices through the power supply. For example, in the TTL NAND gate shown in Figure 8-4, both the Q3 and the Q4 transistors are ON for a short time while output is switching. This increased loading causes a negative spike on V_{cc} and a positive spike on V_{ss}.

In synchronous systems where several gates switch simultaneously, the result is a significant amount of noise on the power and ground lines. This noise can be removed by decoupling the power supply. First, it is necessary to match the power supply's impedance to that of the individual components. Any power supply presents a low source impedance to other circuits, whether they are individual components on the same board or on other boards in a multiboard system. It is necessary to match the supply's impedance to that of the components in order to lessen the potential for voltage drops that can be caused by IC edge rates, ground- or signal-level shifting, noise induced currents or voltage reflections.

This mismatch can be minimized by using a suitable high-frequency capacitor for bulk decoupling of major circuitry sections, or for decoupling entire PC boards in multiboard systems. This capacitor is typically placed at the supply's entry point to the board. It should be an aluminum or tantalum-electrolytic type capacitor having a low equivalent series capacitance and low equivalent series inductance. This capacitor's value is typically 10 to 47 μF. Additional 0.1 μF capacitors may be needed if supply noise is still a problem.

A second type of decoupling is used for the rest of the ICs on the board. Additional decoupling capacitors can be used across the devices between V_{cc} and V_{ss} lines. The voltage spikes that occur due to switching of gates are reduced as the extra current

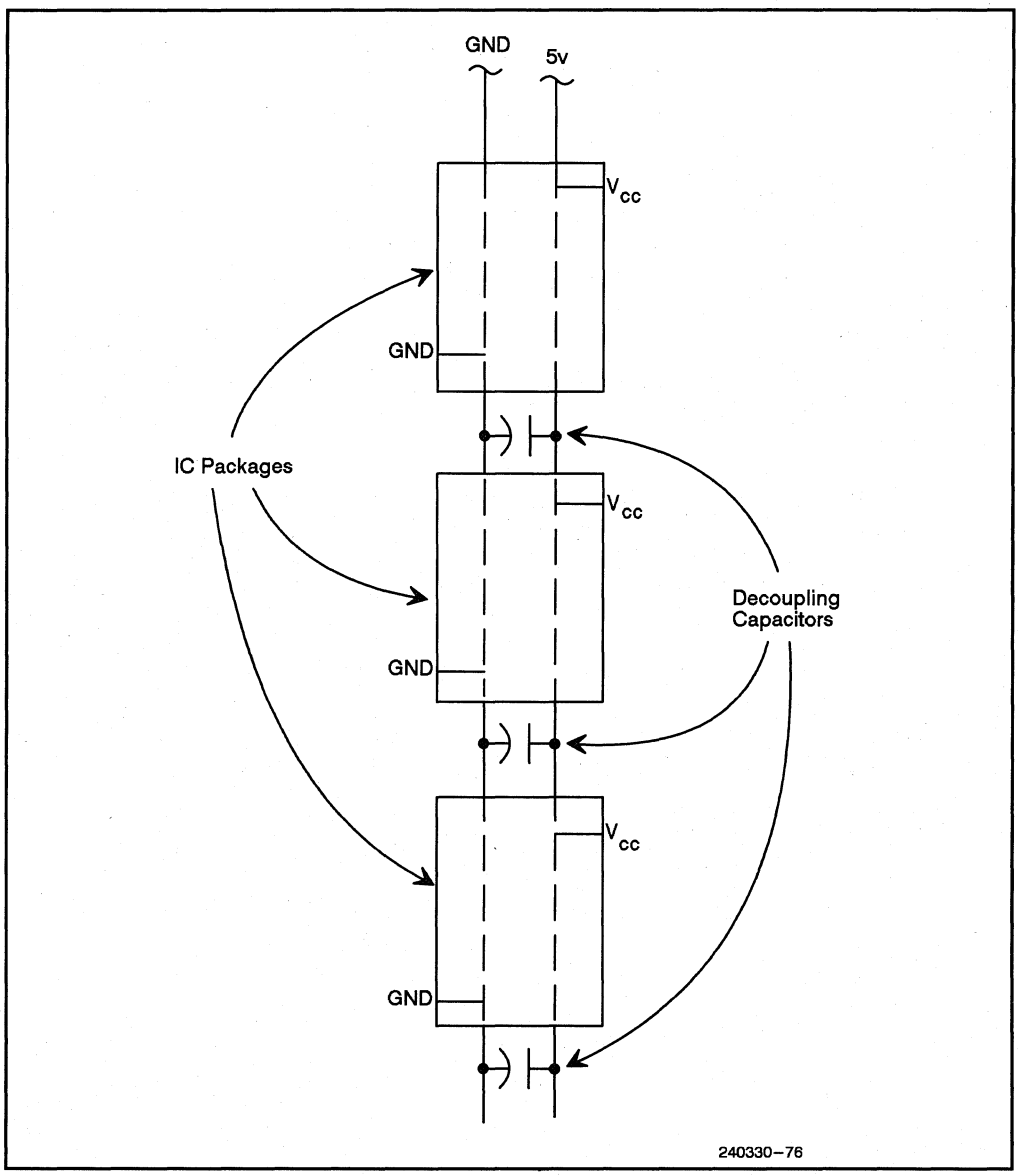


Figure 8-2. Typical Power and Ground Trace Layout for Double-layer Boards

required during switching is supplied by the decoupling capacitors. These capacitors should be placed close to their devices as the inductance of lengthier connection traces reduce their effectiveness.

Most popular logic families require that a capacitor of 0.01 μF to 0.1 μF RF grade be placed between every one to five packages, depending on the exact application. For high-speed CMOS logic, a good rule of thumb is to place one of these bypass capacitors

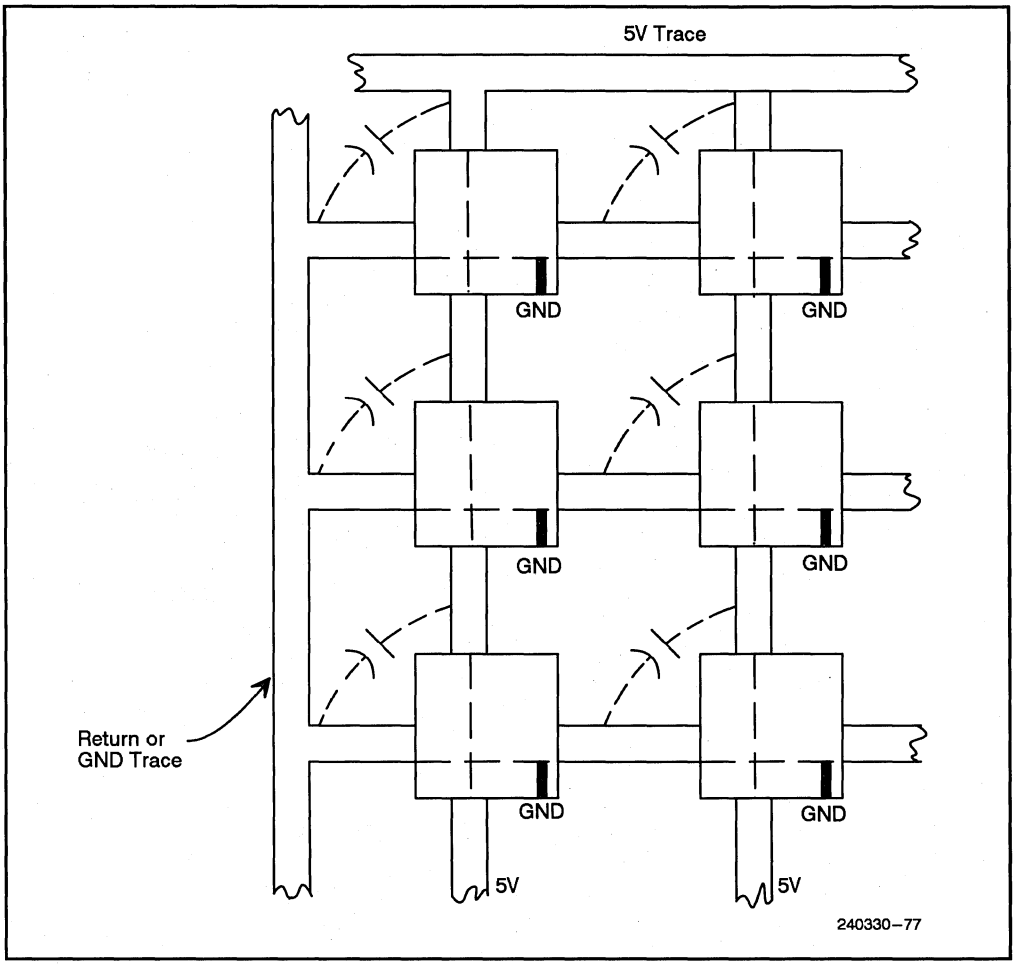


Figure 8-3. Orthogonal Arrangement

between every two to three ICs, depending on the supply voltage, the operating-speed and EMI requirements. The capacitors should be evenly distributed throughout the board to be most effective. In addition, the board should be decoupled from the external supply line with a 10 to 47 pF capacitor. In some cases, moreover, it might be helpful to add a 1 μF tantalum at major supply trace branches, particularly on large PCBs.

Surface mount (chip capacitors) are preferable for decoupling the i860 microprocessor because they exhibit lower inductance and require less total board space than leaded capacitors. They should be connected as shown in the Figure 8-5. These capacitors reduce the inductance, which keeps the voltage spikes to a minimum. They should be used to keep the leads as short as possible.

Inductance is also reduced by the parallel inductor relationships of multiple pins. Six leaded capacitors are required to match the effectiveness of one chip capacitor, but

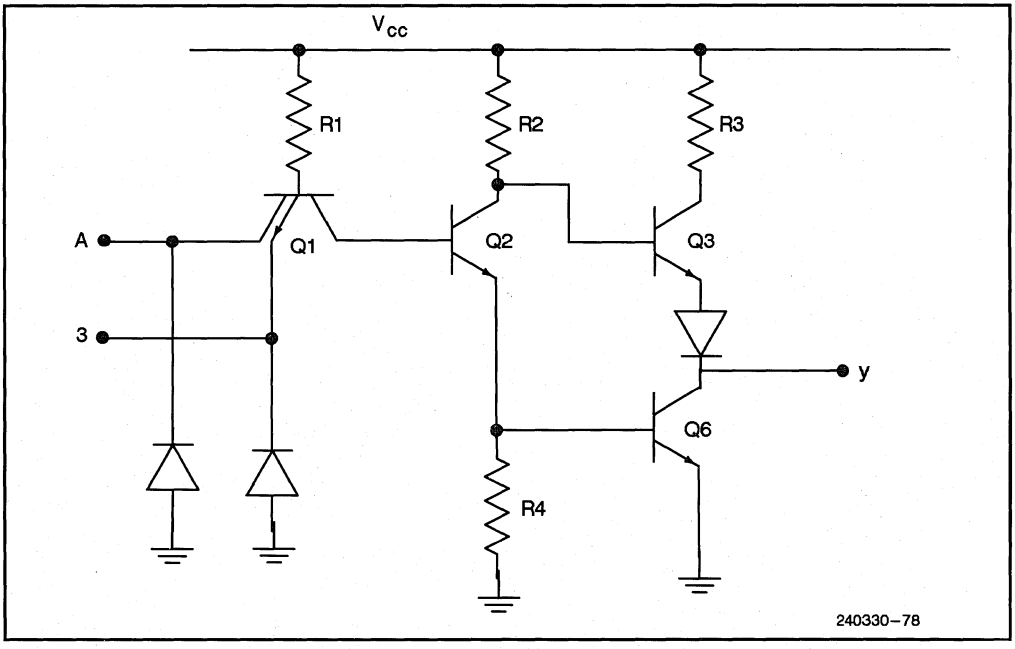


Figure 8-4. Circuit without Decoupling

because only a limited number can fit around i860 CPU, the configuration shown in Figure 8-6 is recommended.

8.4 HIGH FREQUENCY DESIGN CONSIDERATIONS

The overwhelming concern in dealing with high speed technologies is the management of transmission lines. As the edge rates of signals increase, the physical interconnections between devices behave more like transmission lines. Although transmission line theory is straightforward, the difference between ordinary interconnections and transmission line interconnections is fairly complex. Transmission lines have distributed elements which are hard to define, and designers tend to overcompensate for the effects of these elements.

Efficient i860 CPU design requires the identification of transmission lines over back plane wiring, printed circuit board traces, etc. Once this task is accomplished, the designer's next concern should be to deal with three major problems which are associated with electromagnetic propagation: impedance control, propagation delay and coupling.

The following sections discuss the negative effects of a transmission line that occur when operating at higher frequencies. As in higher frequency design, the reflection and crosstalk effects are inevitable. It is impossible to design optimum systems without accounting for these effects. Later sections include a discussion of techniques that can minimize these effects.

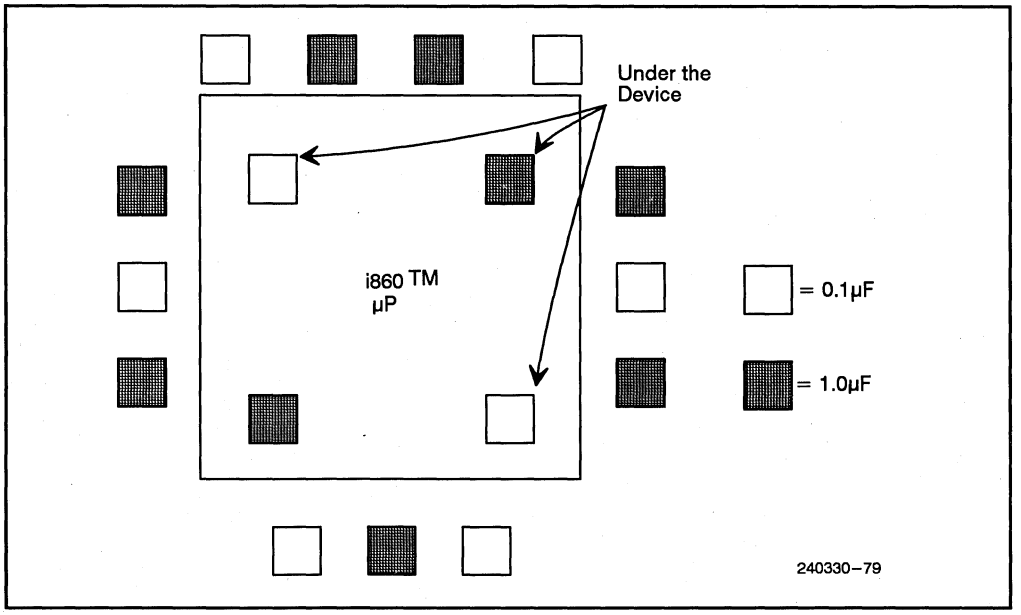


Figure 8-5. Decoupling Chip Capacitors

8.4.1 Transmission Line Effects

As a general rule, any interconnection is considered a transmission line when the time required for the signal to travel the length of the interconnection is greater than one-eighth of the signal rise time. (True K.M., *"Reflection: Computations and Waveforms, The Interface Handbook"*, Fairchild Corp, Mountain View, CA, 1975, Ch. 3). The rise time can be either rise time or fall time, whichever is smaller, and it corresponds to the linear ramp amplitude from 0% to 100%. Normally the rise times are specified between 10% to 90% or 20% to 80% amplitude points. These figures are multiplied by 1.25 and 1.67 to obtain the linear-ramp duration from 0% to 100% amplitude.

For example in a PCB using G-10 and polymide (the two main dielectric systems available for printed circuit boards) signals travel at approximately 5 to 6 inches per nanosecond.

If $t_r / l \times v \leq 8$ then the signal path is not a transmission line but a lumped element,

where

t_r = rise time 0% – 100%
v = speed of propagation (5 to 6 inches/sec)
l = length of interconnection (one-way only)

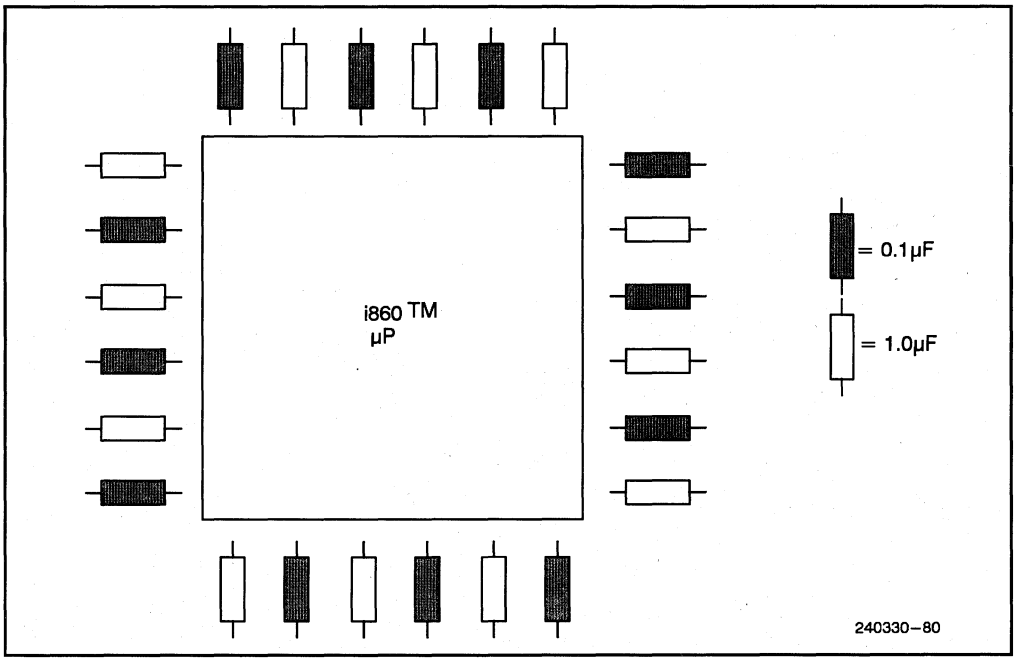


Figure 8-6. Decoupling Leaded Capacitors

The calculation is given by:

$t_r/l \times 6 \leq 8$

so

$l \geq (t_r \times 6)/8 \geq (1.25 \times 4 \times 6)/8 \geq 3.75$ inches.

This calculation is based on the fact that the maximum rise time of the signals for the i860 microprocessor is 4ns. For $l \geq 3.75$ inches interconnections act as transmission lines.

Every conductor that carries an AC signal and acts as a transmission line has a distributed resistance, an inductance and a capacitance which combine to produce the characteristic impedance (Z_0). The value of Z_0 depends upon physical attributes such as cross-sectional area, the distance between the conductors and other ground or signal conductors, and the dielectric constant of the material between them. Because the characteristic impedance is reactive, its effect increases with frequency.

8.4.1.1 TRANSMISSION LINE TYPES

Although many different types of transmission lines (conductors) exist, those most commonly used on the printed circuit boards are microstrip lines, strip lines, printed circuit traces, side-by-side conductors and flat conductors.

8.4.1.1.1 Micro Strip Lines

The micro strip trace consists of a signal plane that is separated from a ground plane by a dielectric as shown in Figure 8-7. G-10 fiber-glass epoxy, which is most common, has an $e_r = 5$ where e_r is the dielectric constant of the insulation. Let:

w = the width of signal line (inches)
t = the thickness of copper (.0015 inches for 1 oz Cu/
.003 inches for 2 oz Cu)
h = the height of dielectric for controlled impedance (inches)

The characteristic impedance Z0, is a function of dielectric constant and the geometry of the board. This is theoretically given by the following formula:

$$Z_0 = (87 / \sqrt{(e_r + 1.41)}) \ln (5.98h/.8w + t) \text{ ohms}$$

where e_r is the relative dielectric constant of the board material and h, w and t are the dimensions of the strip. Knowing the line width, the thickness of Cu and the height of dielectric, the characteristic impedance can be easily calculated.

The propagation delay (t_{pd}) associated with the trace is a function of the dielectric only. This is calculated as follows:

$$t_{pd} = 1.017 \sqrt{(0.475e_r + 0.67)} \text{ ns/ft}$$

For G-10 fiber-glass epoxy boards ($e_r = 5.0$), the propagation delay of microstrip is calculated to be 1.77ns/ft.

8.4.1.1.2 Strip Lines

A strip line is a strip conductor centered in a dielectric medium between two voltage planes. The characteristic impedance is given theoretically by the equation below:

$$Z_0 = 60 / \sqrt{e_r} \ln (5.98b/ \pi (0.8 W + t)) \text{ ohms}$$

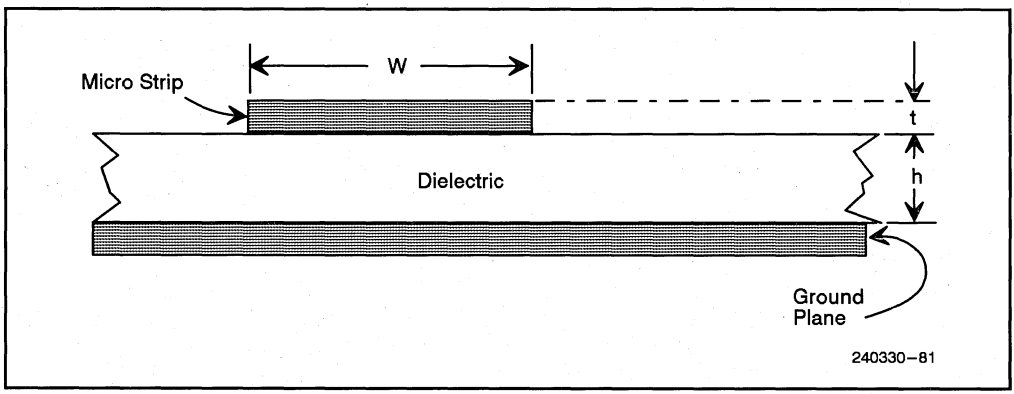


Figure 8-7. Micro Strip Lines

where b = distance between the planes for controlled impedance as shown in Figure 8-8.

The propagation delay is given by the following formula:

$t_{pd} = 1.017 \sqrt{e_r}$ ns/ft

For G-10 fiberglass epoxy boards ($e_r = 5.0$), the propagation delay of the strip lines is 2.26ns/ft.

Typical values of the characteristic impedance and propagation delay of these types of lines are as follows:

$Z_0 = 50$ ohms
$t_{pd} = 2$ns/ft (or 6 in/nsec)

The three major effects of transmission line phenomenon are impedance mismatch, coupling and skew.

The following section will discuss them briefly and provide solutions to minimize their effects.

8.4.2 Impedance Mismatch

As mentioned earlier the impedance of a transmission line is a function of the geometry of the line, its distance from the ground plane, and the loads along the line. Any discontinuity in the impedance will cause reflections.

Impedance mismatch occurs between the transmission line characteristic impedance and the input or output impedances of the devices that are connected to the line. The result is that the signals are reflected back and forth on the line. These reflections can attenuate or reinforce the signal depending upon the phase relationships. The results of these reflections include overshoot, undershoot, ringing and other undesirable effects.

At lower edge rates, the effects of these reflections are not severe. However at higher rates, the rise time of the signal is short with respect to the propagation delay. Thus it can cause problems as shown in Figure 8-9.

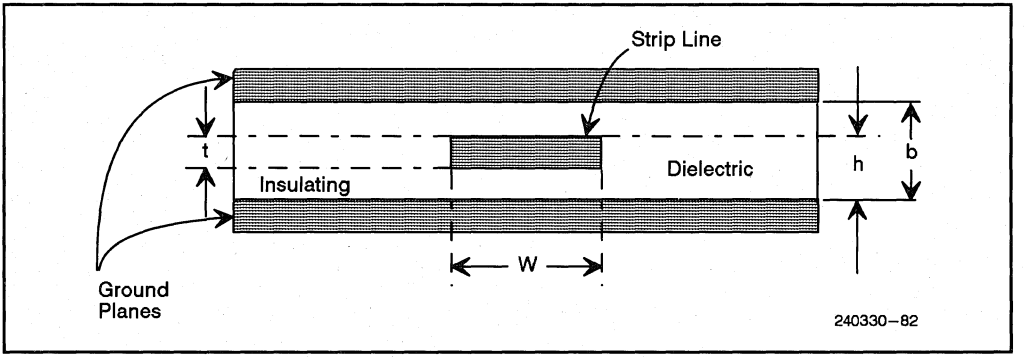


Figure 8-8. Strip Lines

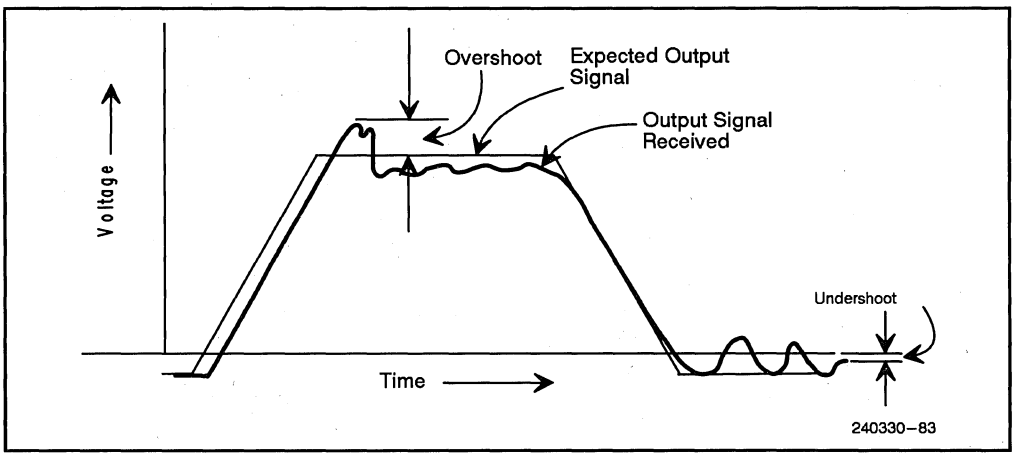


Figure 8-9. Overshoot and Undershoot Effects

Overshoot occurs when the voltage level exceeds the maximum (upper) limit of the output voltage, while undershoot occurs when the level passes below the minimum (lower) limit. These conditions can cause excess current on the input gates which results in permanent damage to the device.

The amount of reflection voltage can be easily calculated. Figure 8-10 shows a system exhibiting reflections.

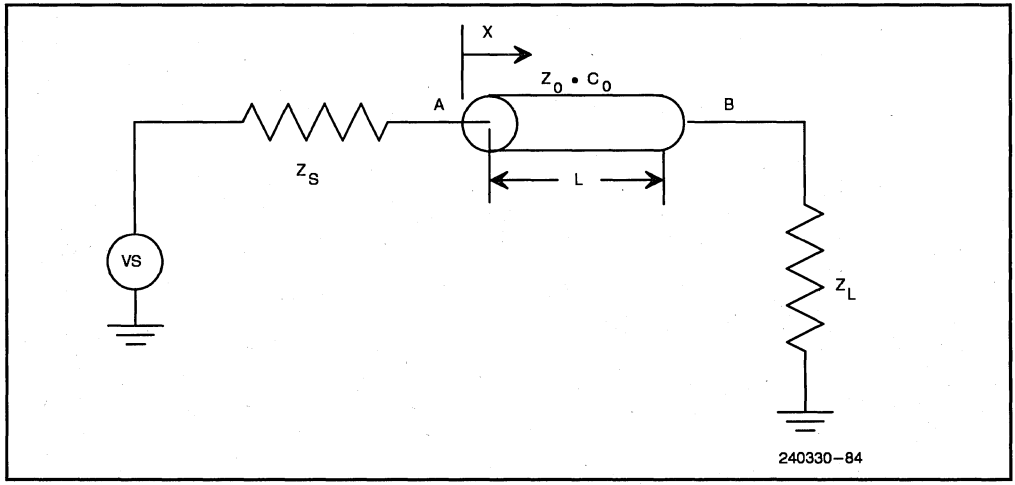


Figure 8-10. Loaded Transmission Line

The magnitude of a reflection is usually represented in terms of a reflection coefficient. This is illustrated in the following equations:

$T = v_r/v_i$ = Reflected voltage/Incident voltage

$T_{load} = (Z_{load} - Z_0)/(Z_{load} + Z_0)$

$T_{source} = (Z_{source} - Z_0)/(Z_{source} + Z_0)$

Reflection voltage V_r is given by V_i, the voltage incident at the point of the reflection, and the reflection coefficient.

The model transmission line can now be completed. In Figure 8-10, the voltage seen at point A is given by the following equation:

$V_a = V_s * Z_0 / (Z_0 + Z_s)$

This voltage V_a enters the transmission line at "A" and appears at "B" delayed by t_{pd}.

$V_b = V_a (t - x/v) H(t - x/v)$

where x = distance along the transmission line from point "A" and H(t) is the unit stop function. The waveform encounters the load Z_L, and this may cause reflection. The reflected wave enters the transmission line at "B" and appears at point "A" after time delay (t_{pd}):

$V_{rl} = T_{load} \bullet V_b$

This phenomenon continues infinitely, but it is negligible after 3 or 4 reflections. Hence:

$V_{r2} = T_{source} \bullet V_{rl}$

Each reflected waveform is treated as a separate source that is independent of the reflection coefficient at that point and the incident waveform. Thus the waveform from any point and on the transmission line and at any given time is as follows:

$$
\begin{aligned}
V(x,t) = \left(\frac{Z_0}{Z_0+Z_s}\right) \Bigg\{ & V_s\left(t-\frac{x}{v}\right) H\left(t-\frac{x}{v}\right) \\
& + T_l \left[V_s\left(t-\left(\frac{2L-x}{v}\right)\right) H\left(t-\left(\frac{2L-x}{v}\right)\right)\right] \\
& + T_l T_s \left[V_s\left(t-\left(\frac{2L+x}{v}\right)\right) H\left(t-\left(\frac{2L+x}{v}\right)\right)\right] \\
& + T_l^2 T_s \left[V_s\left(t-\left(\frac{4L-x}{v}\right)\right) H\left(t-\left(\frac{4L-x}{v}\right)\right)\right] \\
& + T_l^2 T_s^2 \left[V_s\left(t-\left(\frac{4L+x}{v}\right)\right) H\left(t-\left(\frac{4L+x}{v}\right)\right)\right] \\
& + \ldots \Bigg\}
\end{aligned}
$$

Each reflection is added to the total voltage through the unit step function H(t). The above equation can be rewritten as follows:

$$V(x,t) = \left(\frac{Z_0}{Z_0+Z_s}\right)\{ V_s(t-t_{pdx})\ H(t-t_{pdx}) + T_l\ [V_s(t-t_{pd}(2L-x))\ H(t-t_{pd}(2L-x))] + T_lT_s\ [V_s(t-t_{pd}(2L+x))\ H(t-t_{pd}(2L+x))] + \ldots\}$$

Impedance discontinuity problems are managed by imposing limits and control during the routing phase of the design. Design rules must be observed to control trace geometry, including specification of the trace width and spacing for each layer. This is very important because it ensures the traces are smooth and constant without sharp turns.

There are several techniques which can be employed to further minimize the effects caused by an impedance mismatch during the layout process:

1. Impedance matching
2. Daisy chaining
3. Avoidance of 90° corners.
4. Minimization of the number of vias.

8.4.2.1 IMPEDANCE MATCHING

Impedance matching is the process of matching the impedance of the source or load with that of the trace, and it is accomplished with a technique called termination. The reflection, overshoot and undershoot of signals are reduced by terminating the remote end of the transmission line from the source. The terminating impedance combines with the destination input circuitry to produce a load that closely matches the characteristic impedance of the line. (Board traces have characteristic impedances in the range of 30 ohms to 200 ohms.)

The calculation of characteristic impedance was already discussed. Impedance of the printed circuit board backplane connectors have the impedance in the same range as the traces, i.e. 30–200 ohms.

Depending upon the length of the conductors or when using twisted pairs of coaxial cable in place of PC traces, the characteristic impedance of a backplane may change. Backplane impedance is also affected by the number of boards plugged.

8.4.2.1.1 Need for Termination

The transmission line should be terminated when the t_{pd} exceeds one-third of t_r. If $t_{pd} < 1/3\ t_r$, the line can be left unterminated, provided the capacitive coupling between the traces does not cause crosstalk.

Termination thus eliminates impedance mismatches, increases noise immunity, suppresses RFI/EMI and helps to ensure that signals reach their destination with minimum distortion. There are five methods for terminating traces on the board:

1. Series
2. Parallel
3. Thevenin
4. AC
5. Active

Termination requires additional components and power. In case of passive terminations, extra drivers are needed to deliver more current to the line. In case of active terminations extra power is needed which increases the power dissipation of the system.

8.4.2.1.2 Series Termination

One way of controlling ringing on longer lines is with the series termination technique also known as damping. This is accomplished by placing a resistor in series with the transmission line at the driving device end. The receiver has no termination. The value of the impedance looking into the driving device ($R_{driver}+R_L = Z_0$) should approximate the impedance of the line as closely as possible. In this circuit the ringing dampens out when the reflection coefficient goes to zero. Figure 8-11 illustrates the series termination.

One main advantage of series termination is that only logic power dissipation results so that lower overall power is required than other termination techniques. There is one penalty, however, in that the distributed loading along the transmission line cannot be used because only half of the voltage waveform is travelling down the line. The drop in voltage across series terminating resistor limits loading to a maximum of 10 loads.

8.4.2.1.3 Parallel Terminated Lines

Parallel termination is achieved by placing a resistor of an appropriate value between the input of the loading device and the ground as shown in Figure 8-12.

Since the input impedance of the device is high as compared to the characteristic line impedance, the resistor and the line function as a single impedance with a magnitude that is defined by the value of the resistor.

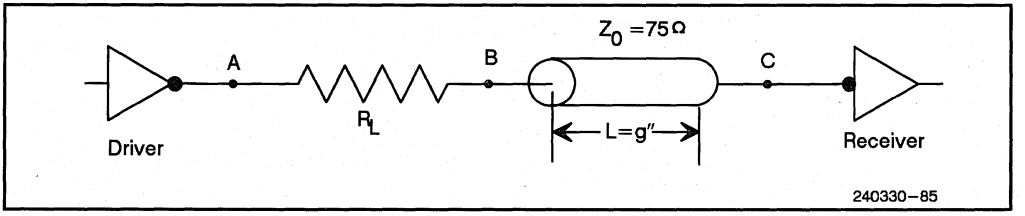


Figure 8-11. Series Termination

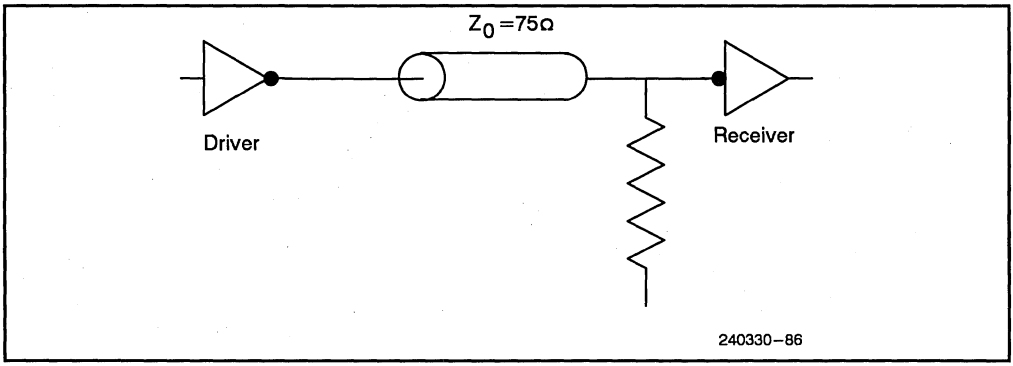


Figure 8-12. Parallel Termination

When the resistor matches the line impedance, the reflection coefficient at the load approaches zero, and no reflection will occur. One useful approach is to place the termination as close to the loading device as possible.

Parallel terminated lines are used to achieve optimum circuit performance and to drive distributed loads—an important benefit of using parallel terminations.

There are two significant advantages of using the parallel termination. First, it provides an undistributed waveform along the entire line. Second, when a long line is loaded in parallel termination, it does not affect the rise and fall time or the propagation delay of the driving device. Note that parallel termination can also be used with wire wrap and backplane wiring where the characteristic impedance is not exactly defined. If the designer approximates the characteristic impedance, the reflection coefficient will be very small. This results in minimum overshoot and ringing. Parallel termination is not recommended for characteristic impedances of less than 100 ohms because of large DC current requirements.

8.4.2.1.4 Thevenin's Equivalent Termination

Thevenin's equivalent termination is an extension of parallel termination technique. It consists of connecting one resistor from the line to the ground and another from the line to V_{cc}. Each resistor has a value of twice the characteristic impedance of the line, so the equivalent resistance matches the line impedance. This scheme is shown in Figure 8-13.

If there were no logic devices present, the line would be placed halfway between the V_{cc} and the V_{ss}. When a logic device is driving the line, a portion of the required current is provided by the resistors, so the drivers can supply less current than needed in parallel termination. The resistor value can be adjusted to bias the line towards V_{cc} or ground V_{ss}. Ordinarily it is adjusted such that the two are equal, providing balanced performance. Thevenin's circuit provides good overshoot suppression and noise immunity.

Due to power dissipation, this technique is best suited for bipolar and mixed MOS/CMOS devices and is not suitable for pure CMOS implementations. There are two reasons for not having Thevenin's equivalent for the pure CMOS system design. First

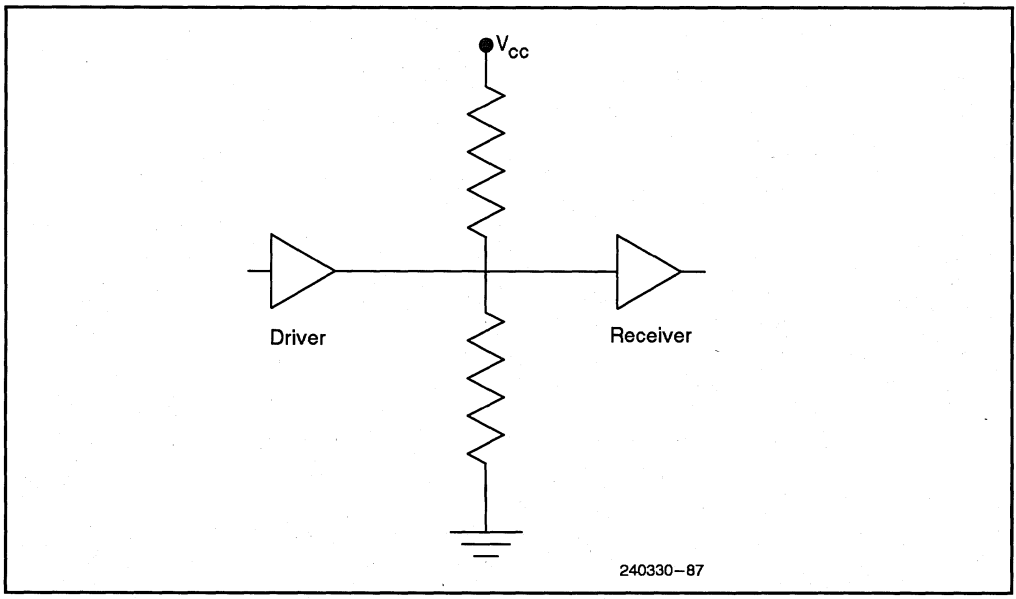


Figure 8-13. Thevenin's Equivalent Circuit

CMOS circuits have very high impedance to ground and V_{cc}, and their switching threshold is 50% of the supply voltage. Second, besides dissipating more power, multiple input crossing may occur which creates output oscillations.

The main problem with Thevenin termination is high power dissipation in the termination resistors in relationship to the total power consumption of all of the CMOS devices on the board. For this reason, most designers prefer series terminations for CMOS to CMOS connections, as this does not introduce any additional impedance from the signal to the ground. The main advantage of the series termination technique, apart from its reduced power consumption, is its flexibility. The received signal amplitude can be adjusted to match the switching threshold of the receiver simply by changing the value of the terminating resistor. Series termination is a very useful technique for interconnecting the logic devices with long lines.

8.4.2.1.5 AC Termination

Another technique for designs that cannot tolerate high power dissipation of parallel termination and delays created by series termination is AC termination. It consists of a resistor and a capacitor connected in series from the line to the ground. It is similar to the parallel termination technique in functionality except that the capacitor blocks the DC component of the signal, and thus reduces power dissipation. This is shown in Figure 8-14.

The main disadvantage of this technique is that it requires two components. Further the optimum value of the RC time constant of the termination network is not easy to calculate. It usually begins as a resistive value which is slightly larger than the characteristic

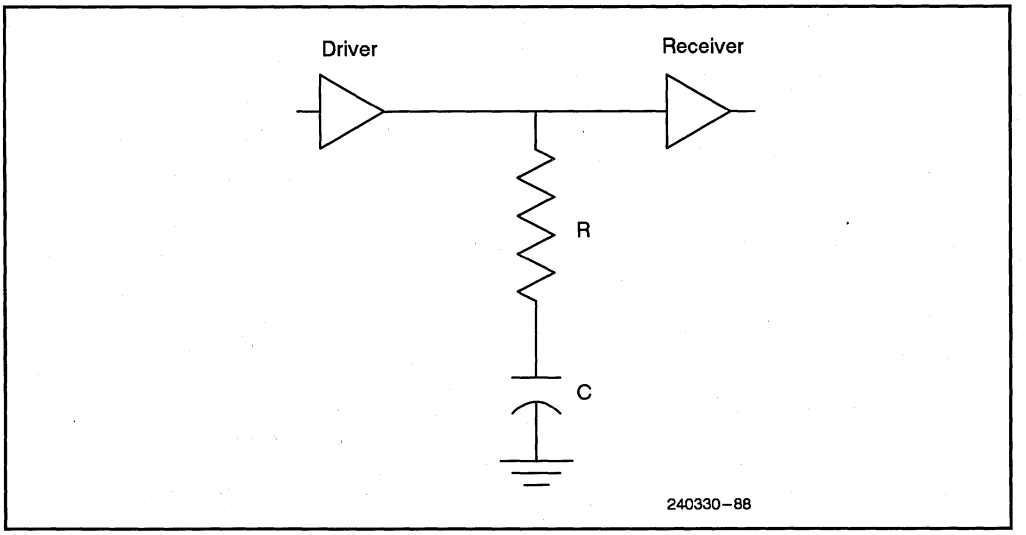


Figure 8-14. A.C. Termination

line impedance. It is critical to determine the capacitor value. If the value of RC time constant is small, the RC circuit will act as an edge generator and will create overshoots and undershoots. Increasing the capacitor value reduces the overshoot and undershoot, but it increases power consumption. As a rule of thumb, the RC time constant should be greater than twice the line delay. The power dissipation of the AC termination is a function of the frequency.

8.4.2.1.6 Active Termination

An active termination consists of a resistor that is connected between the input and output of a buffer driver as shown in Figure 8-15.

The main advantage of this technique is that it can tolerate large impedance variations. This tolerance is valuable when tri-state drivers are connected to backplane busses. However, the terminations are costly, and the signals that are produced are not as clean as other terminations. A common solution is to place active terminations at both ends of the bus. This helps maintain the uniform drive levels along the entire length of the bus, and it reduces crosstalk and ringing.

Table 8-1 shows the comparisons of different termination techniques.

Table 8-1. Comparison of Various Termination Techniques

Termination	# of Extra Components	R_L Power Consumption		Prop Delay
Series	1	Z_0-Z_{OUT}	Low	Yes
Parallel	1	Z_0	High	No
Thevenin	2	$2Z_0$	High	No
AC*	2	$2Z_0$	Medium	No
Active	1	$2Z_0$	Medium	No

* A.C. also uses a capacitor of 200 pf to 500 pf.

Beyond matching impedances, there are other techniques that can help avoid reflections. These are discussed in the following sections.

8.4.2.1.7 Impedance Matching Example

Previous sections discuss the techniques for calculating characteristic impedances (using transmission line theory) and the termination procedures used to match impedances. This section describes an impedance matching example that utilizes these techniques. Figure 8-16 shows a simple interconnection which acts like a transmission line as shown by the calculations.

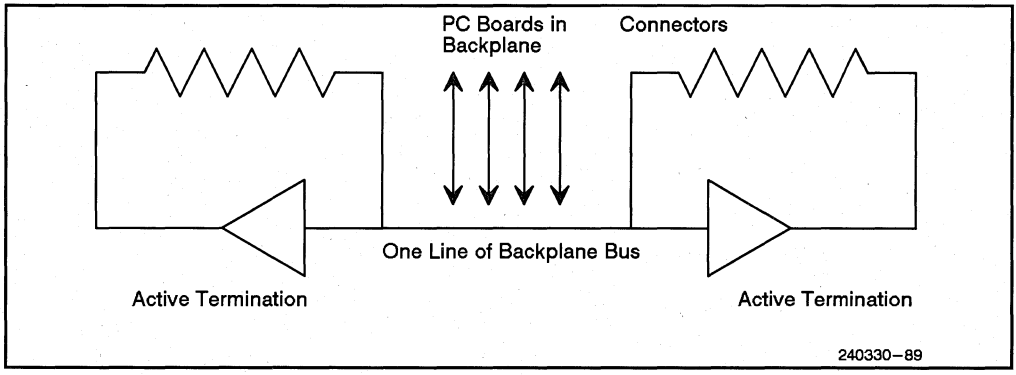


Figure 8-15. Active Termination

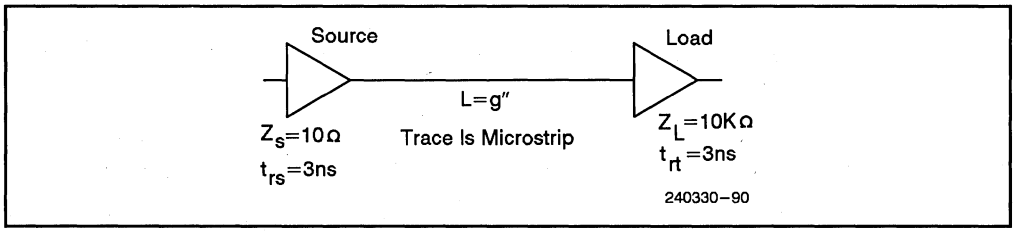


Figure 8-16. Impedance Mismatch Example

In this example the different values are given as follows:

Z_s (source impedance) =10 ohms,
trs (source rise-time) =3nsec (normalized to 0%to 100%)
Z_l (load impedance) =10 Kohms
trl (load rise-time) =3nsec (normalized to 0%to 100%)
l (length of interconnection) =9 in.
trace is micro-strip
e_r (dielectric constant) =5.0
h =.008 in.
w =.01 in.
t =.0015 in.(1 oz. Cu)
v = 6 in./nsec

The interconnection will act as a transmission line if (as was shown in Section 8.4.1)

$l \geq (tr \times v) / 8 \geq (3 \times 6)/8 \geq 3$ in.

The value of l = 9, thus the interconnection acts like a transmission line. The impedance of the transmission line is calculated as follows:

$$Z_0 = 87 / \sqrt{(e_r + 1.41)} \times \ln (5.98h/(.8w + t))$$
$$= 34.39 \ln 5.05 = 55.6 \text{ ohms}$$

Since Z_s = 10 ohms, hence the termination techniques described previously will be needed to match the difference of 45.6 ohms. One method is to use a series terminating resistor of 45.6 ohms or use AC termination where R =55.6 ohms and C =.3μF. The circuit of Figure 8-16 is shown with termination in Figure 8-17.

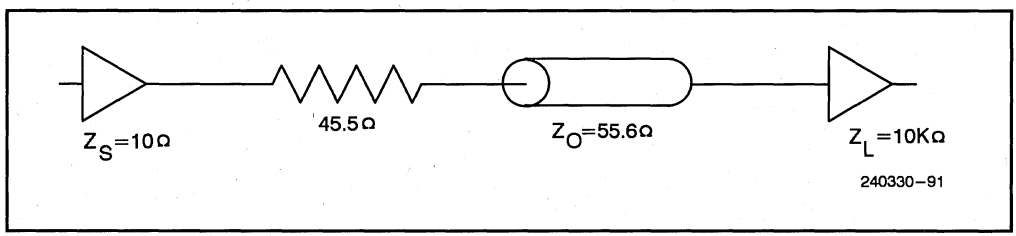


Figure 8-17. Use of Series Termination to Avoid Impedance Mismatch

8.4.2.2 DAISY CHAINING

In laying out PC boards, a stub or T-connection is another source of signal reflection. These types of connections act as inductive loads in the signal path. In daisy chaining, a single trace is run from the source, and the loads are distributed along this trace. This is shown in Figure 8-18.

An alternative to this technique is to run multiple traces from the source to each load. Each trace will have unique reflections, which are then transmitted down other traces when they return to the source. In such cases, a separate termination is required for each branch. To eliminate these multiple terminators from T-connections, high-frequency designs are routed as daisy chains.

Because each gate provides its own impedance load along the chain, it is necessary to distribute these loads evenly along the length of the chain. Hence, the impedance along the chain will change in a series of steps and is easier to match. The overall speed of this line is faster and predictable. Also all loads should be placed at equal distances (regular intervals).

8.4.2.3 90 DEGREE ANGLES

Another major cause of reflections are 90 degree angles in the signal paths, which cause an abrupt change in the signal direction and promote signal reflection. For high-frequency layout of designs, avoid 90 degree angles and use 45 or 135 degree angles, as shown in Figure 8-19.

8.4.2.4 VIAS (FEED-THROUGH CONNECTIONS)

Another impedance source that degrades high-frequency circuit performance is the via. Expert layout techniques can reduce vias to avoid reflection sites on PCBs.

8.4.3 Interference

Previous sections discuss reflections in high-frequency design, their causes, and techniques to minimize them. The following sections discuss additional issues related to high-frequency design, including interference. In general, interference occurs when electrical activity in one conductor causes a transient voltage to appear in another conductor.

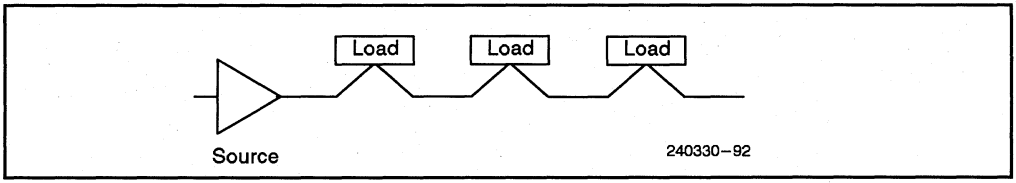


Figure 8-18. Daisy Chaining

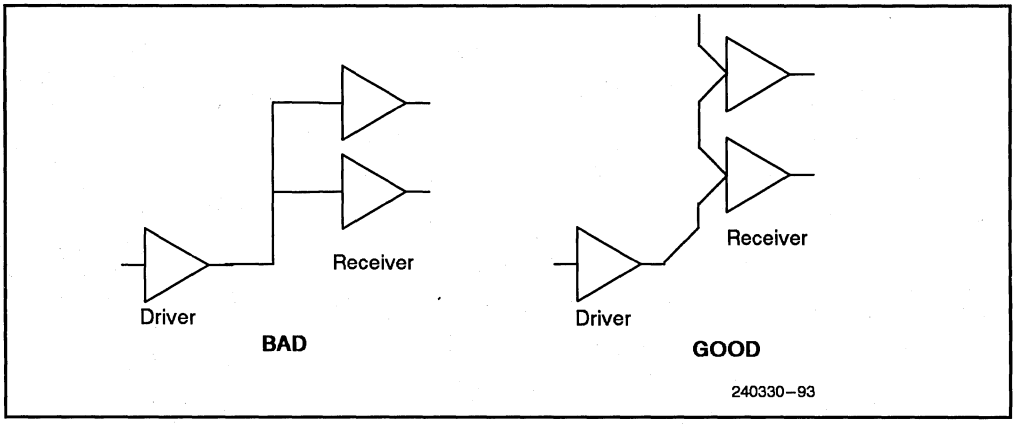


Figure 8-19. Avoiding 90 Degree Angles

Two main factors increase interference in any circuit:

1. Variation of current and voltage in the lines causes frequency interference. This interference increases with increases in frequency.

2. Coupling occurs when conductors are in close proximity.

Two types of interference are observed in high frequency circuits:

1. Electromagnetic Interference (EMI)

2. Electrostatic Interference (ESI)

8.4.3.1 ELECTROMAGNETIC INTERFERENCE (CROSSTALK)

Crosstalk is a problem at high operating frequencies because, as operating frequencies increase, the signal wavelengths become comparable to the length of some of the interconnections on the PC board. Crosstalk is a phenomenon of a signal in one trace inducing another similar signal in an adjacent trace. There are two types of couplings between parallel traces which determine the amount of crosstalk in a circuit: inductive coupling and radiative coupling.

Inductive coupling occurs when a current in one trace produces a current in a parallel trace. This current reduces with the distance between the two traces. Hence, closely spaced wires or traces will incur the greatest degree of inductive coupling. Each of the traces will induce a current in the other.

Radiative coupling occurs when two parallel traces act as a dipole antenna which radiates signals that parallel wires can pick up. This results in the corruption of signal that is already present in the trace. The intensity of this type of coupling is directly proportional to the current present in the trace. However, it is inversely proportional to the square of the distance between the radiator and the receiver.

8.4.3.2 MINIMIZING CROSSTALK

When laying out a board for an i860 microprocessor-based system, several guidelines should be followed to minimize crosstalk.

One source of crosstalk is the presence of a common impedance path. Figure 8-20 shows a typical layout in which earth ground and signal ground are different.

To reduce crosstalk, it is necessary to minimize the common impedance paths, which are shown as the ground impedances Z_2, Z_3 and Z_4. During current switching, the ground line voltage drops causing noise emission. By enlarging the ground conductor (which reduces its effective impedance), this noise can be minimized. This technique also provides a secondary advantage in that it forms a shield which reduces the emissions of other circuit traces, particularly in multilayer circuit boards.

The impedances Z_2 through Z_4 depend upon the thickness of the copper PC-board foil, the circuit switching speeds and the effective lengths of the traces. The current flowing through these common impedance paths radiates more noise as the current increases. The amount of voltage that is generated by these switching currents and multiplied by the impedance is difficult to predict.

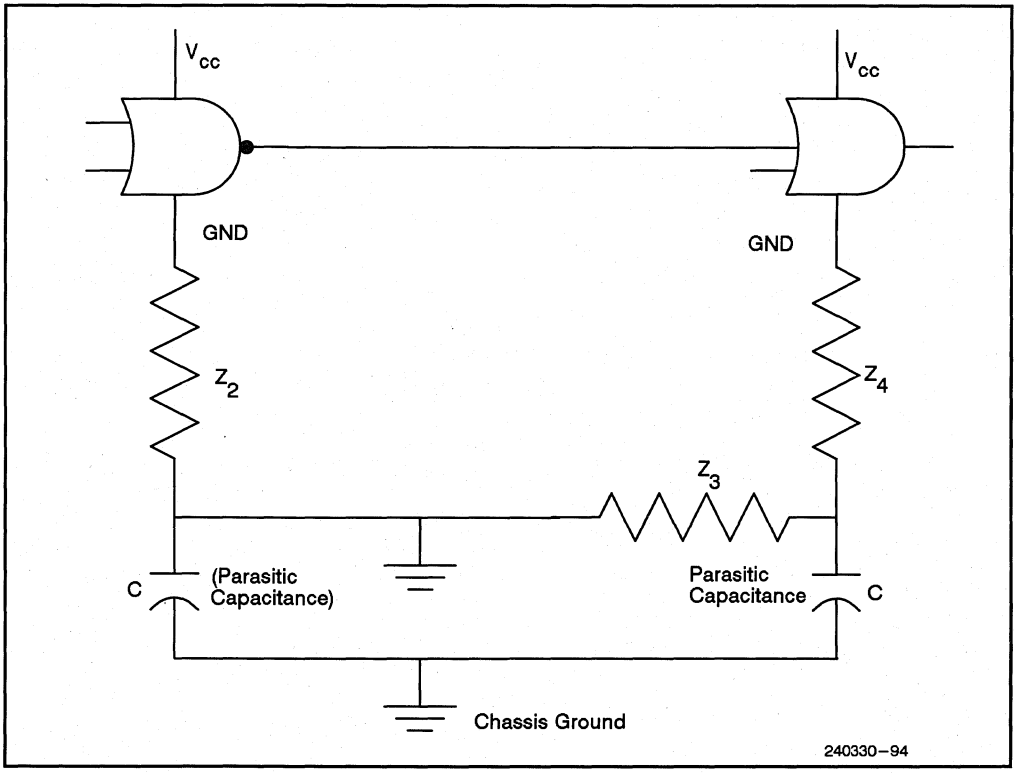


Figure 8-20. Typical Layout

An effective way of reducing EMI is to decouple the power supply by adding bypass capacitors between V_{cc} and ground. This technique is similar to the general technique discussed earlier. (The goal of the previous technique was to maintain correct logic levels.)

The design of effective coupling and bypass schemes centers on maximizing the charge stored in the circuit bypass loops while minimizing the inductances in these loops. Some other precautions that can minimize EMI are as follows:

- Running a ground line between two adjacent lines. The lines should be grounded at both ends.
- Separation of the address and data busses by a ground line. This technique may however be expensive due to large number of address and data lines.
- Removing closed loop signal paths which create inductive noise as shown in Figure 8-21.

Minimizing crosstalk involves first examining the circuit's interconnection with its nearest neighbors since parallel and adjacent lines can interact and cause EMI. It is necessary to maximize the distance between adjacent parallel wires.

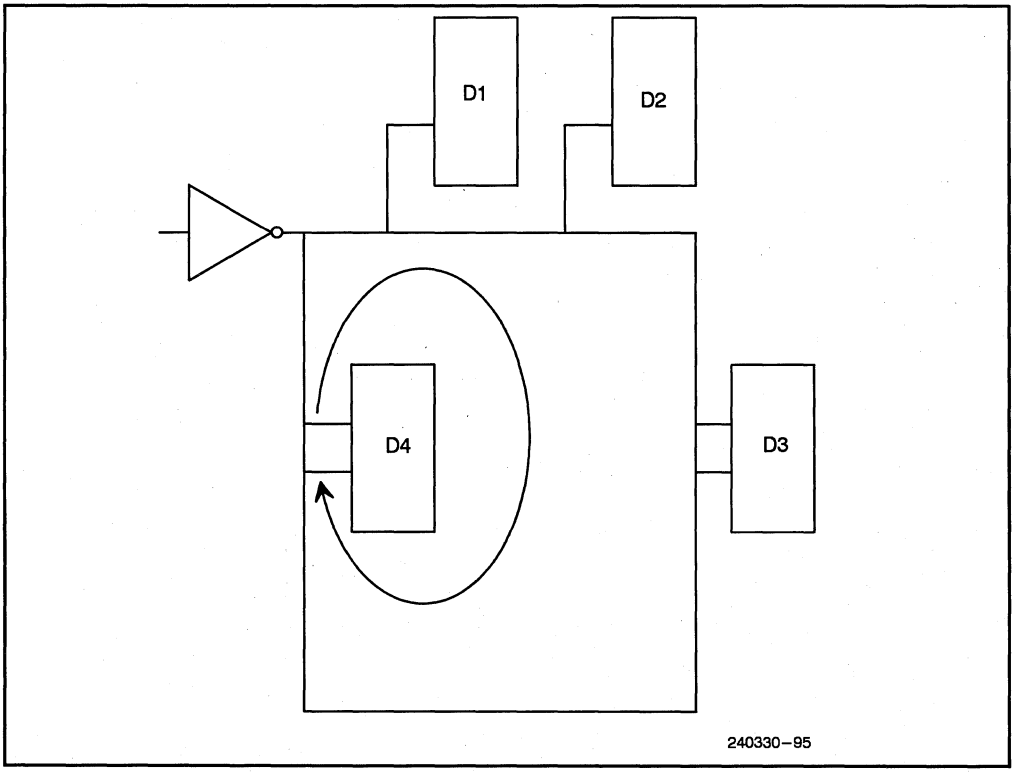


Figure 8-21. Closed Loop Signal Paths are Undesirable

8.4.3.3 ELECTROSTATIC INTERFERENCE

We have discussed two types of coupling, inductive and radiative coupling, which are responsible for creating electromagnetic interference. A third, known as capacitive coupling, occurs when two equipotential parallel traces are separated by a dielectric and act as a capacitor. According to the standard capacitor equation, the electric field between the two capacitor surfaces varies with the permitivity of the dielectric and with the area of the parallel conductors.

Electrostatic interference (ESI) is caused by this type of coupling. The charge built on one plate of the capacitor induces opposite charge on the other. To minimize the ESI, the following steps should be taken.

- Separate the signal lines so that the effect of capacitive coupling is negated.
- Run a ground line between the two lines to cancel the electrostatic fields.

For high-frequency designs, a rule of thumb is to include ground planes under each signal layer. Ground planes limit the crosstalk caused by a capacitive coupling between small sections of adjacent layers that are at equipotentials. Additionally, when the width and thickness of signal lines and their distance from the ground is constant, the effect of capacitive coupling upon impedance remains uniform within ± 5 percent across the board. Using fixed impedance does not reduce capacitive coupling, but it does simplify the modeling of propagation delays and coupling effects. In addition, capacitive coupling can cause interference between layers, so the wires should be routed orthogonally on neighboring board layers.

8.4.4 Propagation Delay

The propagation delay of a circuit is a function of the loads on the line, the impedance, and the line segments. The term propagation delay means the propagation delay in the entire circuit, including the delay in the transmission line (which is a function of the dielectric constant).

Also, the printed circuit interconnections add to the propagation delay of every signal on the wire. These interconnections not only decrease the operating speed of the circuits, but also cause reflection, which produces undershoot and overshoot.

When the propagation delays in the circuit are significant, the design must compensate for the signal skew. Signal skew occurs when the wire lengths (and thus the propagation delays) between each source and each corresponding load are unequal.

Another negative aspect of propagation delay is that it can cause a race condition. This condition occurs when two signals must reach the same destination within one clock pulse of one another. To avoid race conditions, it is necessary to have the signals travel through the same length traces. But if one route is shorter, then the signals will arrive at different timings, causing race conditions.

One way to minimize this is by decreasing the length of the interconnections. Overall route lengths are shorter in multilayer printed circuit boards than in a double layer boards because ground and power traces are not present.

8.5 LATCH-UP

Latch-up is a common condition in the use of CMOS devices which occurs when V_{cc} becomes shorted to V_{ss}. Latch-up is triggered when the voltage limits on the I/O pins are exceeded, causing the internal PN junction to become forward-biased. The following steps ensure the prevention of latch-up.

- Observe the maximum input voltage rating of I/O pins.
- Never apply power to an i860 microprocessor pin or to any device connected to it before applying power to the i860 microprocessor.
- Use good termination techniques to prevent overshoot and undershoot.
- Use a proper layout to minimize reflections and to reduce noise on the signals.

8.6 CLOCK CONSIDERATIONS

8.6.1 Requirements

The i860 microprocessor facilitates an easy to implement 1× clock interface. An external, 33/40 MHz clock synchronizes both the internal functional blocks of the microprocessor and the external signals. Most of the i860 microprocessor's board logic circuitry also uses this clock. A typical i860 microprocessor clock circuit, shown in Figure 8-22, is comprised of a 33/40 MHz oscillator and a buffer.

The clock input requirements for i860 microprocessor systems are more stringent than those for many commonly used TTL devices. A CMOS buffer will meet the clock input requirements as will a TTL buffer with a pullup resistor.

The minimum high and low times are specified as 7ns at 33 MHz and 5ns at 40 MHz. The clock timings are shown in Figure 8-23.

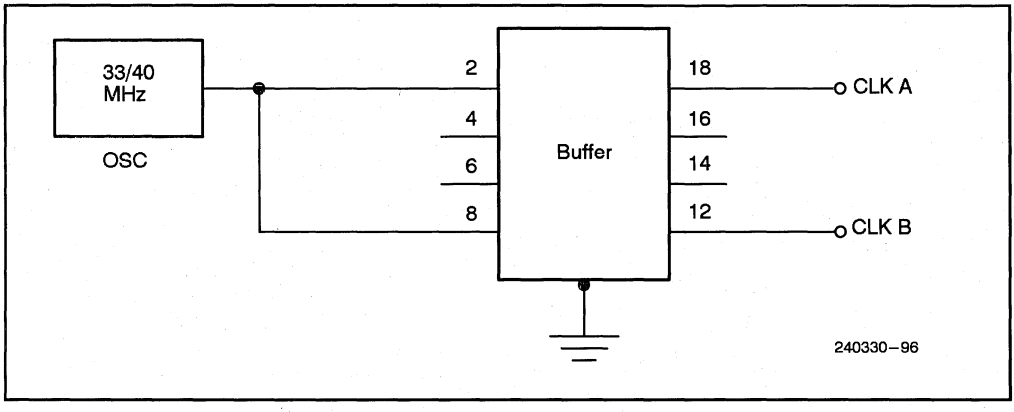


Figure 8-22. Typical Clock Circuit

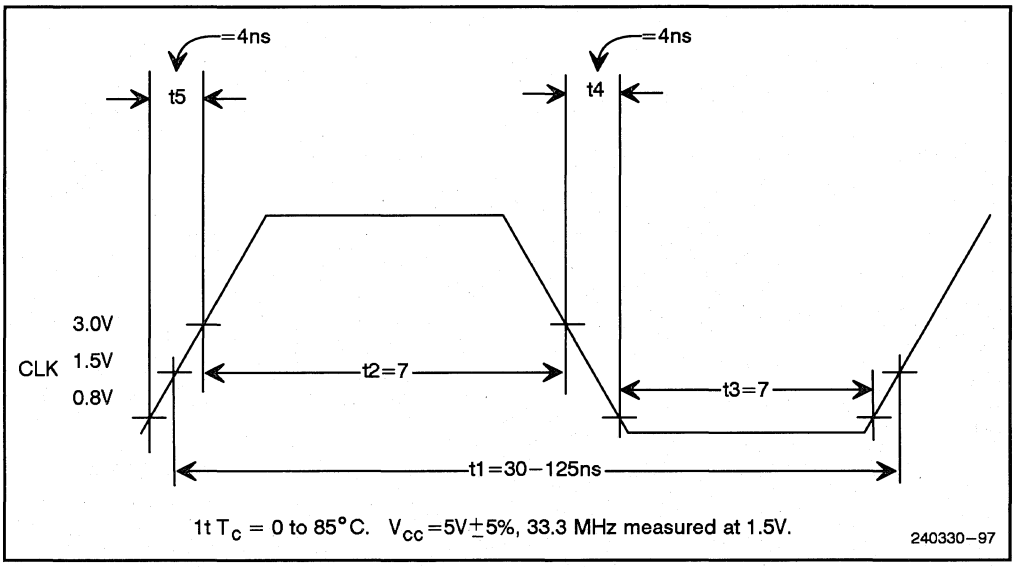


Figure 8-23. Clock Timings

8.6.2 Routing

Achieving the proper clock routing around a 33/40 MHz printed circuit board is delicate because a myriad of problems, some of them subtle, can arise if certain design guidelines are not followed. For example, fast clock edges cause reflections from high impedance terminations. These reflections can cause significant signal degradation in systems operating at 33/40 MHz clock rates. This section covers some design guidelines which should be observed to properly lay out the clock lines for efficient i860 microprocessor operation.

Since the rise/fall time of the clock signal is typically in the range of 2-4ns, the reflections at this speed could result in undesirable noise and unacceptable signal degradation. The degree of reflection depends on the impedance of the traces of the clock connections. These reflections can be optimized by using proper terminations and by keeping the length of the traces as short as possible. The preferred method is to connect all of the loads via a single trace as shown in Figure 8-24, thus avoiding the extra stubs associated with each load. The loads should be as close to one another as possible. Multiple clock sources should be used for distributed loads.

A less desirable method is the star connection layout in which the clock traces branch to the load as closely as possible (Figure 8-25). In this layout, the stubs should be kept as short as possible. The maximum allowable length of the traces depends upon the frequency and the total fanout, but the length of all of the traces in the star connection should be equal. Lengths of less then one inch are recommended.

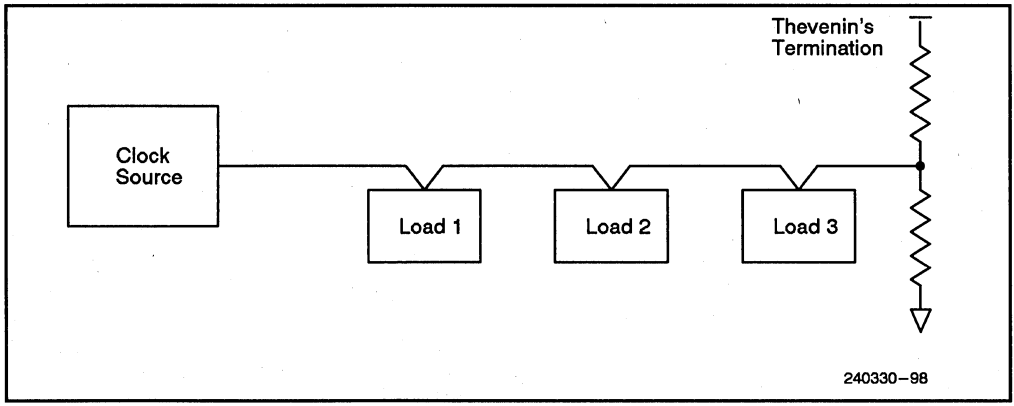


Figure 8-24. Clock Routing

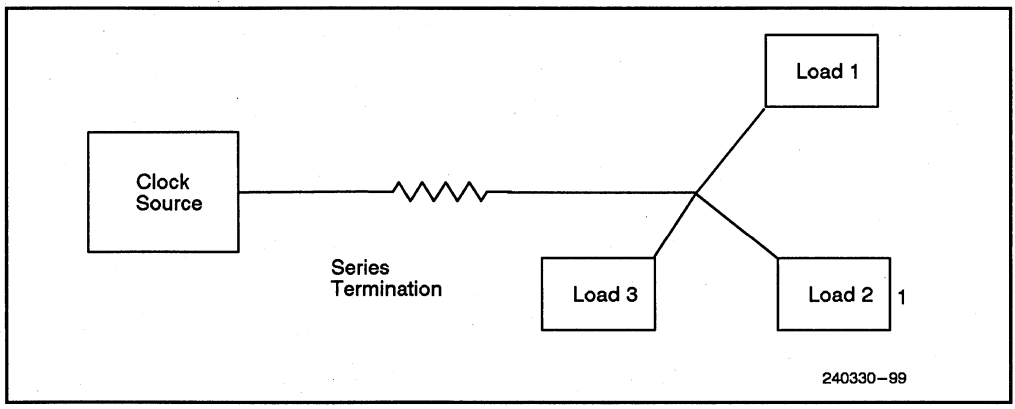


Figure 8-25. Star Connection

8.7 Thermal Characteristics

There are thermal and electrical limitations associated with all operating electronic devices. In an i860 microprocessor-based system, these limitations must be accommodated, due to power dissipation concerns, to achieve proper system performance.

Generally, thermal and electrical characteristics are interrelated, and the actual constraints depend upon the application of a particular device.

Most of the general information on case temperature (T_c), maximum current and voltage ratings, maximum thermal resistance (θ_{ca}) at various airflows and package thermal specifications are given in the *i860™ 64-Bit Microprocessor Data Sheet*. Despite the wealth of information presented in the data sheet, it is impossible to provide graphs and reference tables to cover all applications. The designer must accurately calculate several factors such as junction temperature (T_j) and total power dissipation (P_d) in particular applications. This section explains how to perform these calculations.

The thermal specifications for the i860 microprocessor are designed to ensure a tolerable temperature at the surface of the chip. This temperature, called the junction temperature (T_j), can be determined from external measurements using the known thermal characteristics of the package.

The following two equations facilitate the calculation of the junction temperature (T_j):

$T_j = T_a + (\theta_{ja} * P_d)$ and
$T_j = T_c + (\theta_{jc} * P_d)$

where
- T_j = junction temperature
- T_a = ambient temperature
- T_c = case temperature
- θ_{ja} = junction to ambient temperature coefficient
- θ_{jc} = junction to case temperature coefficient
- P_d = power dissipation (worst case $P_d = I_{cc} * V_{cc}$)

Given a heat sink with a thermal resistance of θ_{sa} (sink to ambient), and given the thermal resistance from the junction to the case θ_{jc}, then the equation for calculating T_j is as follows:

$T_j = P_d(\theta_{jc} + \theta_{cs} + \theta_{sa}) + T_a$
$T_j = P_d(\theta_{jc} + \theta_{cs} + \theta_{sa}) + T_c$

Case temperature calculations offer many advantages over ambient temperature calculations:

- Case temperature is more easily measured compared to ambient temperature because the measurement is localized to a single point (the center of the package).
- The worst case junction temperature (T_j) is lower when calculated with case temperature for two reasons. First, the junction-to-case thermal coefficient (θ_{jc}) is lower than the junction-to-ambient thermal coefficient (θ_{ja}). Therefore, the calculated junction temperature varies less with power dissipation (P_d). Second, the junction-to-case coefficient (θ_{jc}) is not affected by the airflow in the system, while the junction-to-ambient coefficient (θ_{ja}) does vary.

Given the case temperature specification, a designer can either set the ambient temperature or use fans to control the case temperature. Finned heatsinks or conductive cooling may also be used in an environment which prohibits the use of fans.

A designer has considerable freedom in designing the heatsink, and faces only practical and economic limits. Multiple parallel devices may be helpful in reducing θ_{sa}, because, if the heat input to the heat sink is dispersed rather than concentrated, the effective thermal impedance will be lower.

To approximate the case temperature for varying environments, the two equations discussed earlier should be combined by making the junction temperature the same for both, resulting in the following equation:

$T_a = T_c - (\theta_{ja} - \theta_{jc})P_d$

The *i860™ 64-Bit Microprocessor Data Sheet* should be consulted to determine the values of θ_{ja} (per the system's airflow requirement) and the ambient temperature that will yield the desired case temperature. Whatever those conditions are, the proper calculations are very important in achieving an efficient and reliable i860 microprocessor system.

The i860 microprocessor is available in a 168-pin ceramic PGA. The recommended heat sinks for the device are offered in the pin fin design that utilizes air cooling. The heat sink is mounted on the PGA package with a frame and spring. A typical heat sink is shown in Figure 8-26.

8.8 DERATING CURVE AND ITS EFFECTS

A derating curve is a graph that plots the output buffer delay against the capacitive load. The curve is used to analyze a signal delay without necessitating a simulation every time the processor's loading changes. This graph assumes the lumped-sum capacitance model to calculate the total capacitance. The delay in the graph should be added to the specified AC timing value for the device that is driving the load. The derating curve is different for different devices because each device has different output buffers.

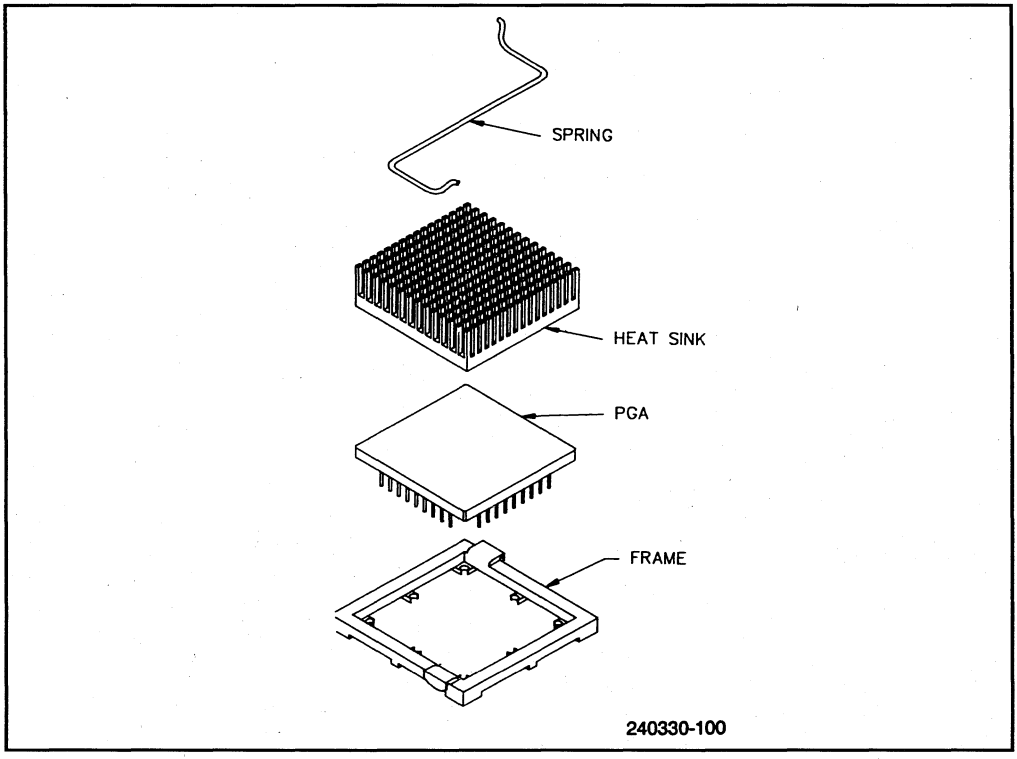


Figure 8-26. Typical Heat Sinks

A derating curve is generated by tying the chip's output buffers to a range of capacitors. The voltage and resistance values chosen for the output buffers are at the highest specified temperature and are rising (worst case) values. The value of the capacitors centers around the AC timing values for the chip. For 33 MHz and above, this is 50 pF. Since the AC timing specifications are measured for a signal reaching 1.5 V, the output buffer delay is the time that it takes for a signal to rise from 0 to 1.5 V. A curve is then drawn from the range of time and capacitance values, with 50 pF representing the average and with nominal or zero derating. These curves are valid only for a 25-150 pF load range. Beyond this range the output buffers are not characterized. The derating curves for the i860 microprocessor are shown in Figure 8-27. These curves use the lumped capacitance model for circuit capacitance measurements and must be modified slightly when doing worst-case calculations that involve transmission line effects.

8.9 BUILDING AND DEBUGGING THE i860™ MICROPROCESSOR-BASED SYSTEM

While an i860 microprocessor based-system designer should plan the entire system, it is necessary to begin building different elements of the core and begin testing them before building the final system. If a printed circuit board layout has to be done, the whole system may be simulated before generating the net list for the layout vendor. It is advisable to work with a preliminary layout to avoid the problems associated with wire wrap boards that operate at high frequencies. A typical i860 microprocessor-based system is shown in Figure 8-28.

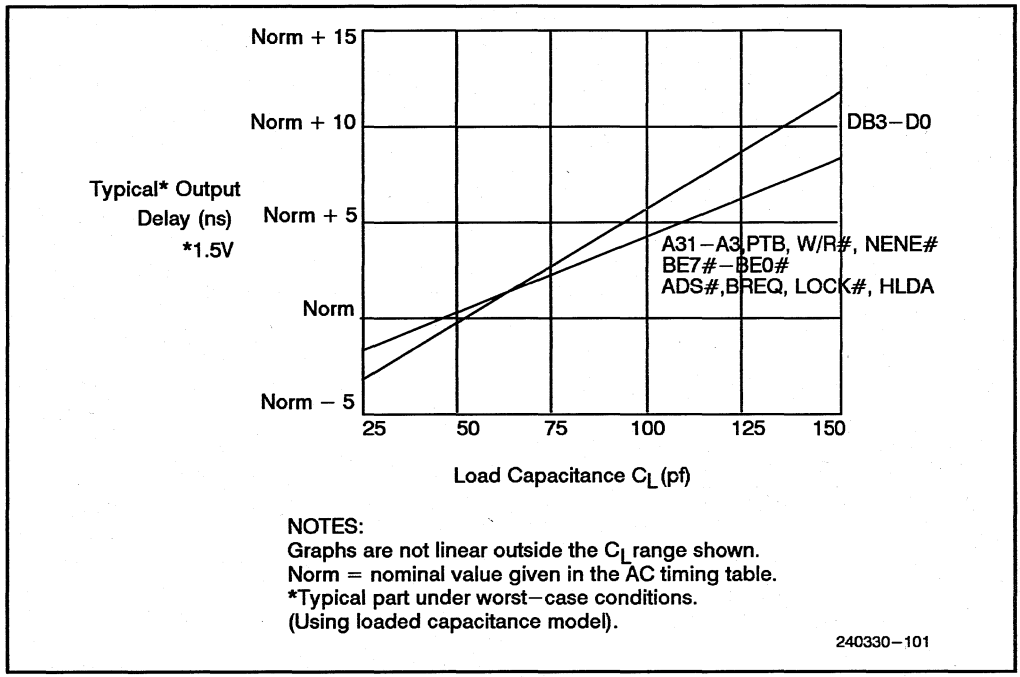


Figure 8-27. Derating Curves for the i860™ Processor

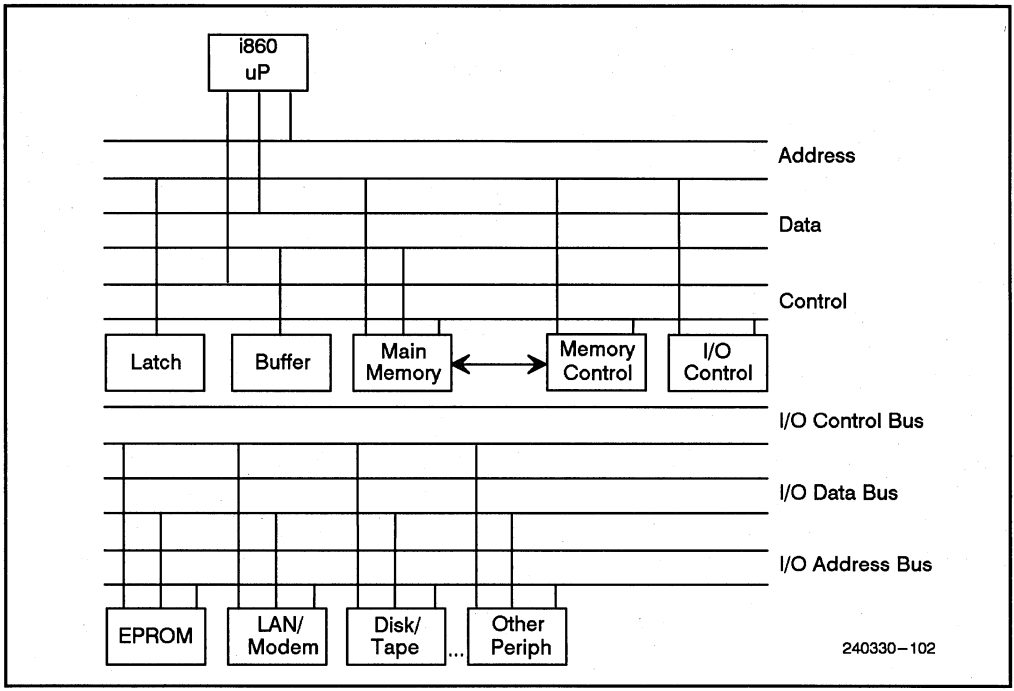


Figure 8-28. Typical i860™ Processor-based System

The following steps are usually carried out in designing with the i860 microprocessor.

1. Clock circuitry should use an oscillator and fast buffer. The CLK signal should be clean, without any overshoots or undershoots.

2. The reset circuitry should be designed as shown in Chapter 3. This circuitry is used to generate the RESET # signal for the i860 microprocessor. The system should be checked during reset for all of the timings. The clock continues to run during these tests.

3. The INT and HOLD pin should be tied to LOW (negated from the active state). The READY# pin is pulled HIGH so as to add additional delays (wait-states) to the first cycle. At this instance, the i860 microprocessor is reset, and the signals emitted from it are checked for the validity of the state. The i860 microprocessor will generate the physical address 0XFFFFFF00. The address latch is connected at this time, and the address is verified.

4. The PLD implementing the address decoder should be connected to the i860 microprocessor. After reset, the i860 microprocessor is checked to find out whether it accesses the EPROM for retrieving the initial code. The i860 microprocessor supports a special CS8 mode for boot-up from eight-bit I/O devices. This allows the processor to boot-up from the eight-bit EPROM. Once the system boots, the ROM can be copied into memory or can be disabled and replaced by DRAMs.

To interface the i860 microprocessor to an eight-bit EPROM, an address multiplexer is used to select between BE2#. . . 0# in CS8 mode and processor addresses in data mode. This mode is discussed in detail in the I/O interfacing chapter.

8.9.1 Debugging Features of the i860™ Microprocessor

The i860 microprocessor supports debugging by providing data and instruction break-points. The various debugging features are discussed below.

- A data break-point register (**db**) allows the specification of the address which can be monitored by the i860 microprocessor.
- The processor status register has break read and break write bits which enable traps. For a detailed discussion, refer to the *i860™ 64-Bit Microprocessor Data Sheet* and the *i860™ 64-Bit Microprocessor Programmer's Reference Manual.*
- A data access trap (DAT) bit in the processor status register helps the trap handler to determine when the data break point cause a trap.
- A special trap instruction sets a breakpoint in the code. Additionally, an instruction trap (IT) bit allows the trap handler to determine when a trap instruction will cause a trap.

Combined with a general understanding of debugging issues, these features are sufficient for debugging an entire i860 microprocessor-based system. These combined with certain hindsight into general debugging issues is sufficient to debug the entire i860 microprocessor based system.

If the initial run of a diagnostic program is not successful, then a logic analyzer can be used to determine the source of a problem.

After an initial debugging the i860 microprocessor should generate a code fetch cycle to the EPROM.

The i860 microprocessor can stop issuing new cycles for the following reasons:

- The READY# signal is never asserted to terminate to the bus cycles.
- There is an infinite loop executing out of the cache. Address lines will toggle but no bus cycles will be present.

8.9.2 Certain Gotchas when Debugging with i860™ Microprocessor

When designing with the i860 microprocessor, there are certain issues of which a designer should be aware:

- After reset, the instruction and data cache may contain data so the cache flush procedure should be used to reset the instruction and data cache.
- In CS8 mode, when i860 microprocessor does a cache block fill, it fetches addresses in decrementing order and executes in incrementing order.

8.9.3 Debugging

Once the i860 microprocessor-based system is designed and the printed circuit board is fabricated and stuffed, the next step is to debug the hardware in increments. The following sections provide valuable debugging concepts and techniques for writing diagnostic software. The *i860™ 64-Bit Microprocessor Data Sheet, i860™ 64-Bit Microprocessor Programmer's Reference Manual* and this manual provide a good start.

8.9.4 Simple Diagnostic Programs

To begin debugging an i860 microprocessor-based system, the designer should utilize a set of EPROMs with simple programs that send a message from the microprocessor signaling that the initial communication channel is working. A code example is shown in Figure 8-29.

```
DATA_PORT_ADDR = 0x10000008

ld$start::
    mov     DATA_PORT_ADDR, r4
loop::
    ld.l    message_ptr,     r6
    ld.l    message_length,  r7
loop1::
    ld.s    0(r6), r5
    st.s     r5,   0(r4)
    addu   2,    r6,    r6
    adds   -2,   r7,     r7
    btne    r7,  r0,     loop1

    br     loop
    nop
.data
 message_ptr::
   .long message
 message_length::
   .long 58 // Message length must be even !
message::
   .byte ''Hello !  (This message has been sent by the i860 CPU)'',13,10
```

Figure 8-29. Simple Diagnostic Program

The loop inside this program can be utilized for various debugging purposes, such as verifying various bus cycles and checking the noise level on the clock and on other critical control signals. The rest of the system can then be exercised and tested.

Once the data transceivers are connected, it is impossible to check the data path from the i860 microprocessor for reading and writing to the external memory subsystem. A special diagnostic program (shown by the example in Figure 8-30) is written to check the read/write function to the external memory. The program should have built in loops so that the i860 microprocessor's behavior can be observed on a logic analyzer.

```
ld$start::
  mov    DATA_PORT_ADDR, r4
  mov    0,    r5

 loop::
  ld.s   0(r4), r10
  ld.s   0(r4), r11
  ld.s   0(r4), r12
  ld.s   0(r4), r13
  ld.s   0(r4), r14

  st.s   r10,     0(r4)
  st.s   r11,     0(r4)
  st.s   r12,     0(r4)
  st.s   r13,     0(r4)
  st.s   r14,     0(r4)

  br     loop
  nop
```

Figure 8-30. Read/Write Diagnostic Program

When the clock generator, i860 microprocessor, address decoder, address latch, data transceivers, READY# generation logic and RESET logic are all functioning, the i860 microprocessor is capable of running the software in the EPROMs.

Once the EPROMs are installed, the READY# line should be added to the bus cycles following reset. During this state, the system is checked using the digital oscilloscope.

Another check should verify that the address latches have latched the first address, and that the address decoder is providing a chip select signal for the EPROMs. The EPROMs should supply the requested data through the data transceivers to the i860 microprocessor's data pins.

Next the READY# input should be connected to the PLD that is generating the ready signal to test the i860 microprocessor while running the simple diagnostic program. The program loops back on itself and the system runs multiple bus cycles. The logic analyzer can be used at this point to observe the dynamic behavior of the system.

8.9.5 Other Simple Diagnostic Software

Additional diagnostic programs can be written to test other system operations such as whether the i860 microprocessor is able to read and write from DRAMs or perform functions like cache flushing. The following are samples of diagnostic programs.

```
//This program writes to the i860 CPU

DATA_PORT_ADDR = 0x1000008
ld$start::
       mov DATA_PORT_ADDR, r4
       mov r0, r5
loop:: st.s r5, 0(r4)
       addu 1, r5, r5
       br loop
       nop
```

```
//This program copies ROM code to DRAM on the i860 uP add-in
//board.

dramstart=    0x7eff0000 //noncacheable alias
epromstart=   0xffff0000
epromend=      0xfffffff0
              .atmp r31
              .text

              // initialize control regs

              // leave all ints disabled
ld$start::    st.c r0,psr
st.c          r0,fsr
ld.c          fir,r0
st.c          r0,epsr

              // clear pipeline
              pfadd.ss        f0,f0,f0
              pfadd.ss        f0,f0,f0
              pfadd.ss        f0,f0,f0
              pfmul.ss        f0,f0,f0
              pfmul.ss        f0,f0,f0
              pfmul.ss        f0,f0,f0

              mov         epromstart,r9
              mov         epromend,r7
              call        flush_cache        //make sure we get a miss
              mov         dramstart,r8

loadbytes::
              ld.l   0(r9),r10             //cache data for bytes 0 & 8
              ld.b   0(r9),r10             //load byte 0 from cache
              call   flush_cache           //insure cache miss next time
              ld.b   8(r9),r18             //load byte 8 from cache

              ld.b   2(r9),r13             //cache data for bytes 3 & 11
              ld.b   0(r9),r13             //load byte 3 from cache
```

```
            call    flush_cache          //insure cache miss next time
            ld.b    8(r9),r19            //load byte 11 from cache

            ld.s    Ø(r9),r14            //cache data for bytes 4 & 12
            ld.b    Ø(r9),r14            //etc.
            call    flush_cache
            ld.b    8(r9),r2Ø

            ld.b    1(r9),r15            //cache data for bytes 5 & 13
            ld.b    Ø(r9),r15
            call    flush_cache
            ld.b    8(r9),r21

            ld.b    Ø(r9),r16            //cache data for bytes 6 & 14
            ld.b    Ø(r9),r16
            call    flush_cache            ld.b   8(r9),r22

            ld.b    3(r9),r17           //cache data for bytes 7 & 15
            ld.b    Ø(r9),r17
            call    flush_cache
            ld.b    8(r9),r23

store::
            st.b    r1Ø,Ø(r8)           //store byte Ø to DRAM
            st.b    r13,1(r8)           //  '' ''    3 ''
            st.b    r14,2(r8)           //           4
            st.b    r15,3(r8)           //           5
            st.b    r16,4(r8)           //           6
            st.b    r17,5(r8)           //           7
            st.b    r18,6(r8)           //           8
            st.b    r19,7(r8)           //          11
            st.b    r2Ø,8(r8)           //          12
            st.b    r21,9(r8)           //          13
            st.b    r22,Øxa(r8)         //          14
            st.b    r23,Øxb(r8)         //          15
            xor     r7,r9,rØ            //check for end of ROM
            bc      load_reset          //branch if end of ROM
            addu    Øx1Ø,r9,r9          //increment ROM counter
            br      loadbytes      //copy another 12 bytes
            addu    Øxc,r8,r8      //increment DRAM counter

load_reset::          //Load the reset code to DRAM

w1=Øxa0000000         //This hex code is for the reset branch
w2=Øxa0000000
w3=Øxe4010000
w4=Øxec21ffff
w5=Øx40000800
w6=Øxa0000000

resxc_start=Øx7effff00    //noncacheable alias of feffff00
dramc_start=Øxffff0000    //cacheable DRAM that will be mapped
```

```
                        //over EPROM space after boot
byte_bucket=0x7fff0010    //ROM location

            mov     resxc_start,r4   //bottom of reset branch code
            mov     w1,r5

            st.l    r5,0(r4)         //store word 1
            addu    4,r4,r4          //increment counter
            mov     w2,r5
            st.l    r5,0(r4)         //store word two
            addu    4,r4,r4
            mov     w3,r5
            st.l    r5,0(r4)         //store word three
            addu    4,r4,r4
            mov     w4,r5
            st.l    r5,0(r4)        //store word four
            addu    4,r4,r4
            mov     w5,r5
            st.l    r5,0(r4)       //store word five
            addu    4,r4,r4
            mov     w6,r5
            st.l    r5,0(r4)       //store word six

//The next line must be hand patched after every reassembly
            mov     0xfffff16c,r1        //fffff16c is mask_cs8
//We go through the warp drive section twice to insure that
//it gets completely cached. We must be executing from
//cache when we kill CS8 and the boot bit.
            mov     byte_bucket,r5
            ld.c    dirbase,r4          //get dirbase contents
            warp_drive::
            st.b    r0,0(r5)            //does nothing first time
            orh     0x100, r0, r20
            ld.b    0(r20),r0           //reset EVAT int
            st.c    r4,dirbase          //does nothing first time
            bri     r1                  //go to mask_CS8 the first time
            nop                         //go to ffff0000 second time
            mask_cs8::
            mov     0xffffff7f,r3 //kill cs8 bit
            and     r3,r4,r4
            or      0x20,r4,r4          //invalidate code cache bit
            mov     0x100001f, r5
            xorh    h%byte_bucket,r5,r5 //zero r5's high 16 bits
            mov     dramc_start,r1      //r1 now has DRAM start address
            br      warp_drive          //r5 has EVAT boot port address
            nop                         //r4 has CS8 bit reset
```

```
//The following cache flush procedure is from the i860™ 64-Bit Microprocessor Pro-
grammer's   Reference Manual
// Please refer to it for additional information.
            flush_cache::
            FLUSH_P=        0x7f000000-32
//rw=r24, rx=r25, ry=r26, rz=r27
            mov     r1,r2
            ld.c    dirbase,r27
            or      0x800,r27,r27
            adds    -1,r0,r25
            call    D_FLUSH
            st.c    r27,dirbase

            or      0x900,r27,r27
            call    D_FLUSH
            st.c    r27,dirbase

            xor     0x900,r27,r27
            mov     r2,r1
            bri     r1
            st.c    r27,dirbase

            D_FLUSH::
            mov     FLUSH_P,r24
            or      127,r0,r26
            bla     r25,r26,D_FLUSH_LOOP
            ld.l    32(r24),r0
            D_FLUSH_LOOP::
            bla     r25,r26,D_FLUSH_LOOP

            flush   32(r24)++
            bri     r1
            ld.l    -512(r24),r0
            .end
```

The diagnostic software verifies the ability of the system to perform the bus cycles. The i860 microprocessor fetches code from the EPROMs, and this implies that the EPROM can read functions correctly. Instructions in the program generate bus cycles to write and read the DRAM. The data value read back is checked for accuracy. The program has built-in loops which allow the designer to observe processor states on a logic analyzer and to monitor the signal level on an oscilloscope.

Testability

9

CHAPTER 9
TESTABILITY

9.1 INTRODUCTION

Testability is a major issue in digital system design. Testing involves two processes, test generation and test verification. Test generation is the process of determining the test sequence for a circuit that will verify its proper operation and developing test vectors. Test verification is the process of proving that the circuit works with the test vectors. Test vectors are generated and introduced into a system, and, by observing the response from the system and comparing it to expected data, error conditions can be detected. Therefore testing the system requires control over the test vectors and a means of observing the response.

Many testing methods are practiced in the industry, including Level-Sensitive Scan Design (LSSD), Scan Path, Scan/Set, and Random Access. These techniques require additional hardware, particularly shift registers, in order to input the test vectors and to observe the critical points in the system.

Various designs for testability have evolved. These designs share the same objective: to control and observe the critical points in a system. As the system becomes more complex it becomes more difficult to control and observe the signal path, and it is therefore essential to plan for testability in the design phase.

The two methods that are most often used to load in test vectors are parallel loading and serial scanning. Parallel loading of the input and output data translates to wide buses and higher costs. Further, it may not be effective in storing and analyzing the results. Serial scanning requires more clock cycles to load in test vectors through a serial channel and to run the diagnostics. The response is then read through a serial channel, whereupon a signature analysis can be performed.

The i860™ microprocessor has a boundary scan mode that involves a simple serial interface that allows the testing of all signal traces with only seven probe connections. These probes allow forcing of all the outputs and sampling of all inputs. This can be used in component testing or board-level testing for the i860 CPU's interface. This chapter discusses the testability features of the i860 CPU and the interface timings in performing board-level testing.

9.2 BOUNDARY SCAN MODE

The boundary scan mode is an elegant testability method that provides serial scan diagnostics. Only seven probes need to be connected: the CLK, BSCN, SCAN, SHI, BREQ, RESET, and HOLD signals. With this configuration the user can apply test vectors to the system via a serial channel and sample the response.

The CLK input determines the execution rate and timings of the i860 microprocessor; the timings of the other signals are specified relative to the rising edge of this signal. In addition, the clock signal determines the shift-in rate of the test vectors and the shift-out rate of the response.

The testability pins are the Boundary Scan Shift Input (SHI), the Boundary Scan Enable (BSCN), the Shift Scan Path (SCAN), and the boundary scan shift output, which is shared with the Bus Request (BREQ) output. The BREQ pin has two functions. In normal processor operations, the BREQ line is asserted when the i860 microprocessor has a pending memory request, even when the HLDA line is asserted. In the boundary scan mode it is the serial shift out pin. The Bus Hold (HOLD) line has a specific function which is described in great detail, along with the above-mentioned signals, later in this chapter. The RESET line causes the processor initialization. More details on RESET are provided in Section 3.2 of the data sheet.

When the Boundary Scan Enable (BSCN) signal is asserted, the i860 mircoprocessor enters the boundary scan mode on the next rising edge of CLK. When BSCN is deasserted while in boundary scan mode, the i860 microprocessor leaves the boundary scan mode on the next rising edge of CLK. After leaving the boundary scan mode the internal state is undefined and therefore RESET must be asserted. The timings to enter and exit the boundary scan mode are shown in Figure 9-1.

The BSCN, signal configures the i860 microprocessor for the test mode. When in test mode, the processor can operate in normal mode or shift mode, and the Shift Scan Path (SCAN) input causes the i860 microprocessor to assume one of the two. The test mode operations are defined in Table 9-1. The normal mode is entered on the rising clock edge when the SCAN line is deasserted; the shift mode is entered on the rising edge of the clock when the SCAN line is asserted. In normal mode, the output signals are driven, and the inputs are sampled simultaneously. In shift mode, the test vectors are shifted in at one end of the chain, and the response is shifted out simultaneously at the other end. In testing operations the user switches from shift mode to normal mode and vice versa until all of the vectors are exercised. Then the user exits from the boundary scan mode.

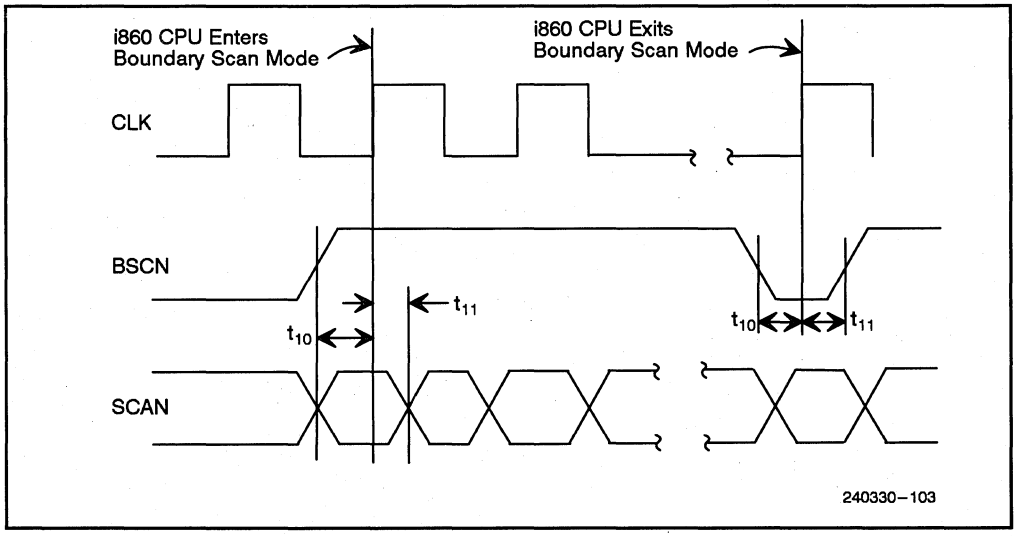


Figure 9-1. Entering and Exiting the Boundary Scan Mode

Table 9-1. Test Mode Selection

BSCN	SCAN	Testability Mode
LO	LO	No testability
LO	HI	Intel Reserved
HI	LO	Boundary Scan Mode, Normal
HI	HI	Boundary Scan Mode, Shift SHI as input, BREQ as output

9.2.1 Shift Mode Operation

Transferring vectors to the system and reading the responses from the system requires a serial path to the i860 microprocessor. This is provided by the shift mode operation. In effect, the shift mode creates a long serial register to and from the i860 microprocessor. Data shifted into the CPU corresponds to new test vectors that will be output to the system. A response from the system can be shifted out while in shift mode. When switching to and from normal mode, the i860 microprocessor performs either a serial-to-parallel operation or a parallel-to-serial operation. This is a simple, yet effective way to test the system.

In shift mode, the pins shown in Table 9-2 are organized as a boundary scan chain. This chain can be thought of as a long shift register that is shifted on the rising edge of CLK. The order of the boundary scan chain is shown in Figure 9-2. The SHI pin receives the input on one end of the boundary scan chain. The other end (the most significant bit) of the boundary scan chain is the BREQ pin, shown here as the 127th position. The bits that are shifted in through the SHI pin in the serial mode are in the sequence shown in Figure 9-2. For shift mode operations, BREQ is shifted in followed in order by BE0#, BE1#, ..., BSCN, SHI. A total of 127 cycles are needed. While these values are being shifted in, serial bits are coming out of the BREQ pin. These bits are in the order BREQ, BE0#, BE1#, ..., BSCN, and SHI, and they correspond to the response of a previously loaded vector.

To avoid glitches that could occur while the values are being shifted out along the chain, the RESET and HOLD pins must be asserted. In this way, all of the tristateable outputs will be disabled. The timings related to serially shifting the data into and out of the processor is shown in Figure 9-3.

The shift mode is used to shift in a new test vector, as well as to read the response to the previous test vector. These operations are performed concurrently.

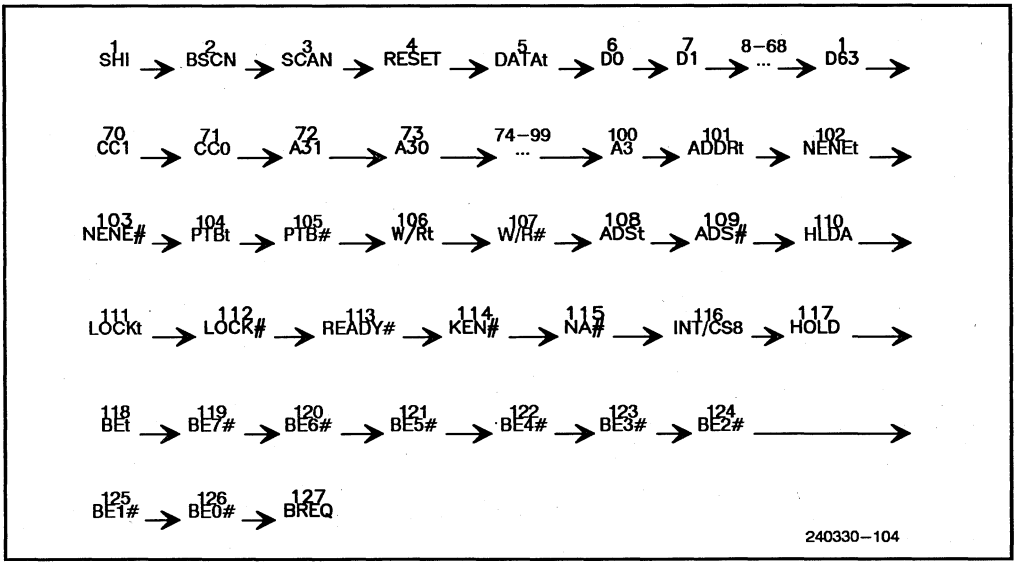


Figure 9-2. Order of Boundary Scan Chain

Table 9-2. Test Mode Latches

Input Latch	Output Latch	Associated Control Latch
SHI		
BSCN		
SCAN		
RESET		
D0–D63	D0–63	DATAt
CC1–CC0		
	A31–A3	ADDRt
	NENE#	NENEt
	PTB#	PTBt
	W/R#	W/Rt
	ADS#	ADSt
	HLDA	
	LOCK#	LOCKt
READY#		
KEN#		
NA#		
INT/CS8		
HOLD		
	BE7#–BE0#	BEt
	BREQ	

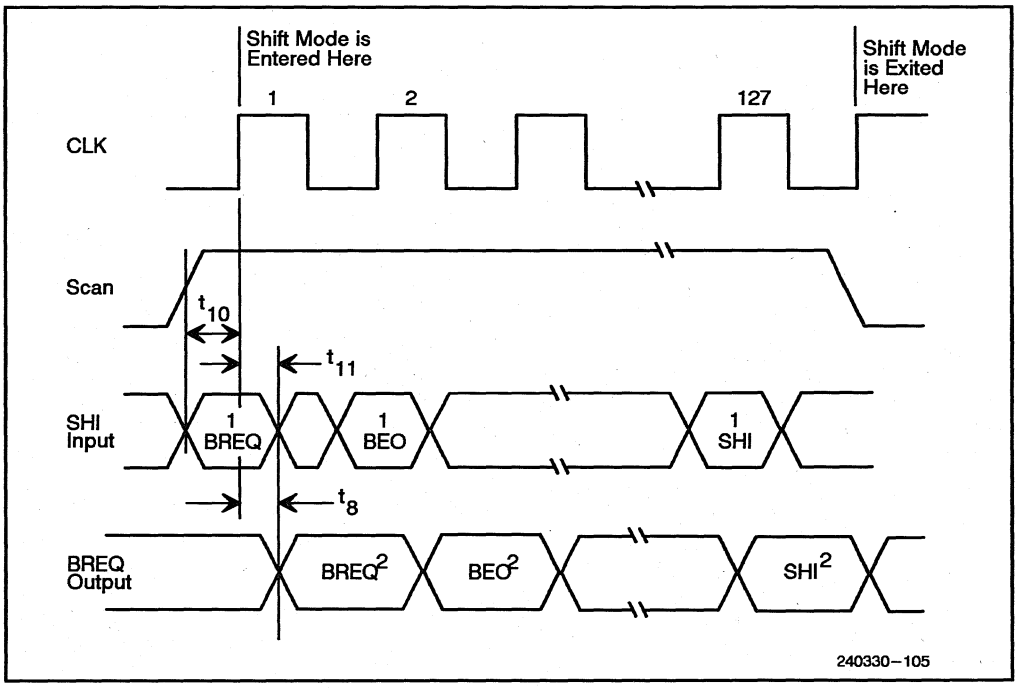


Figure 9-3. The Shift Mode of the Boundary Scan Mode

9.2.2 Normal Mode

During normal mode, two operations can occur simultaneously. The first is the driving of the output lines with the test vector, while the second is the sampling the response on the input lines. The outputs and inputs are shown in Table 9-2.

The three-state output pins A31–A3, BE7#–BE0#, W/R#, NENE#, ADS#, LOCK#, and PTB are enabled by storing a high value in the corresponding control latches ADDRt, BEt,W/Rt, NENEt, ADSt, LOCKt, and PTBt, respectively. If the corresponding output control latch contains a low value, then the pins are tristated.

The data pins D63–D0 are I/O pins and are enabled by the control latch, DATAt, which is similar to the other control latches. In addition when the DATAt contains a low value, then the data pins are configured as inputs and the values on the pins are sampled.

For each of the input pins RESET, HOLD, INT/CS8, NA#, READY#, KEN#, SHI, BSCN, SCAN, and CC1–0, the corresponding latch is loaded with the value that is being driven into the pin.

Normal mode is selected when the SCAN is deasserted while BSCN is asserted. All of the signals shown in Table 9-2 are loaded serially in the shift mode (i.e. while BSCN is high and SCAN is high.) The timings for the normal mode operation are shown in Figure 9-4.

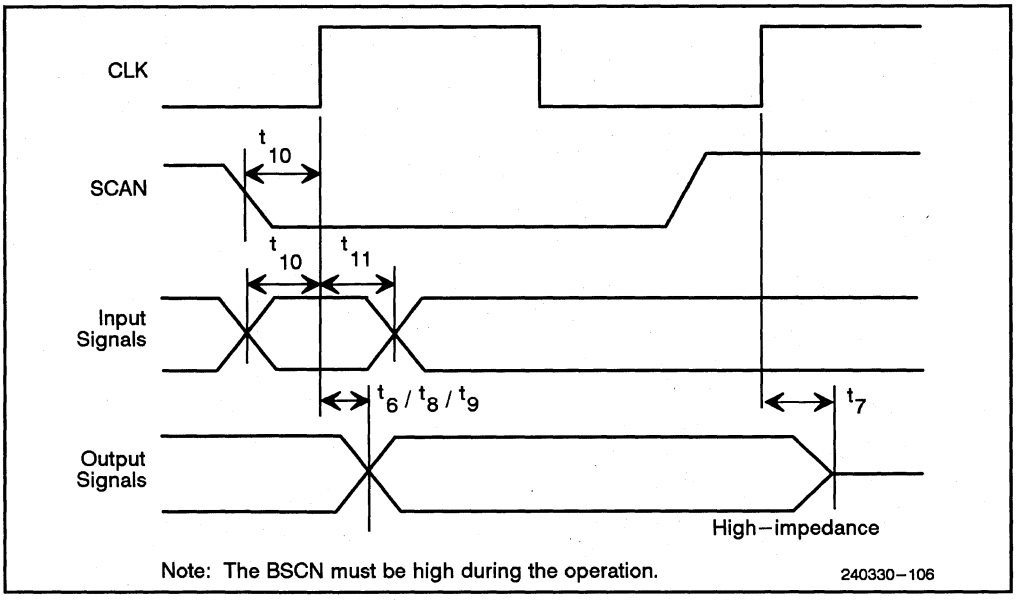


Figure 9-4. The Normal Mode of the Boundary Scan Mode

9.2.3 A Test Sequence Example

The following relates to a typical testing session and addresses controllability and observability issues.

1. Enter boundary scan mode.

2. Establish a serial link via the shift mode to shift in the test vectors and to sample the response. Data can be serially shifted into and out of the i860 CPU. Values are assigned to the latches that correspond with the pins, as shown in Table 9-1. The first test vector is then transferred to the i860 microprocessor.

3. Enter normal mode. The processor drives the output and also samples the inputs. This portion of the sequence sends a test vector to the system and also reads the response synchronously to the rising edge of CLK.

4. Reenter shift mode and read the response from the output of the serial chain, while loading new values for the next test vector at the input of the serial chain. It is possible to loop between steps 3 and 4 until finished.

5. Exit boundary scan mode and reset the processor.

While running the test sequence, the user must be aware that the response being received during the normal mode may correspond to test vector presented in the previous cycle. This is true if a scan path also exists in the peripheral circuits.

9.3 USING THE TESTABILITY FEATURES IN A SYSTEM

The boundary scan mode can help in component-level testing and in board-level testing. These are discussed here, with the emphasis on board level testing.

9.3.1 Component-level Testing

For incoming testing, the serial scan device uses a serial channel to provide a means for causing the output pins to toggle. Values can be assigned to the output latches in the shift mode. During normal mode operation, the processor will drive the output pins. The values of the input pins can also be latched in normal mode and then read out in the shift mode. This provides a mechanism for monitor the inputs and toggling the outputs via a serial channel, and may reduce test program development time.

9.3.2 System-level Testing

The benefits of the boundary scan mode are even more apparent when one is debugging or testing a system, as on a bed of nails in Automatic Test Equipment (ATE). A typical i860 microprocessor system may consist of peripherals and memory elements, as shown in Figure 9-5. If the system is being debugged and brought up, then the serial path can set up test vectors to various memory elements or peripherals. The response from the peripherals can be accessed following the propagation delay while in normal mode. Then, the response is serially shifted out in the shift mode. The received response is compared to the expected response. After the user knows the peripherals are functioning properly, code from an EPROM can be used for more extensive testing.

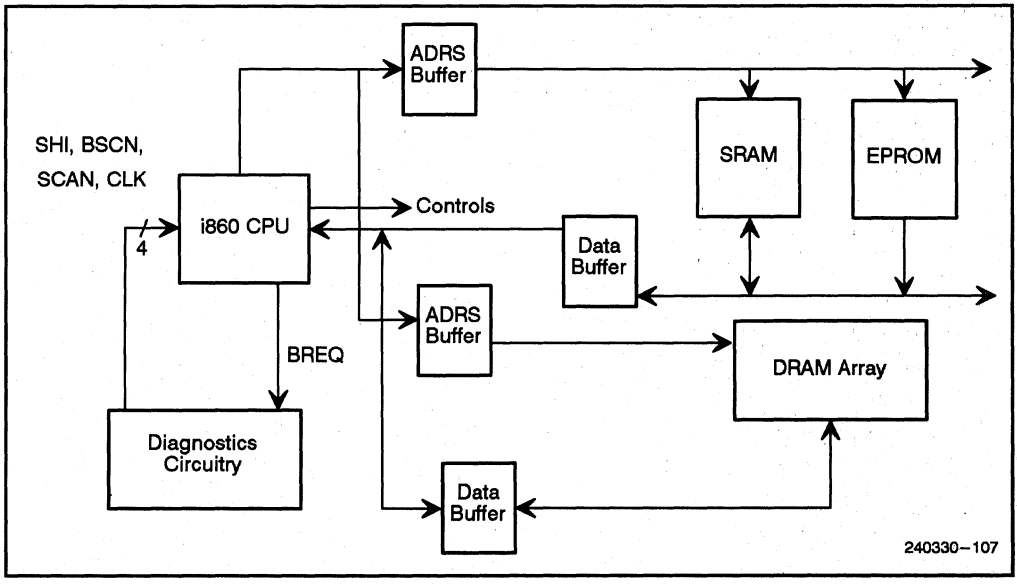
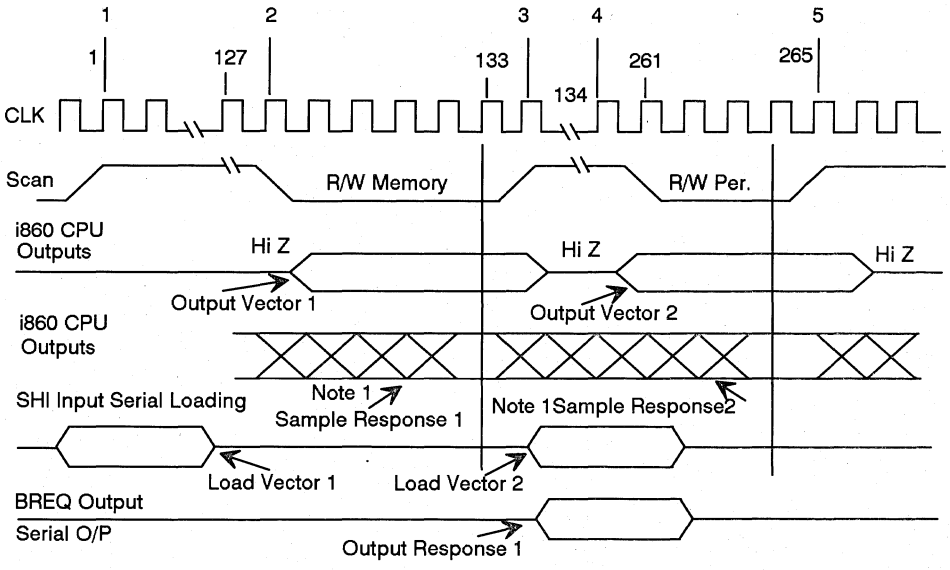


Figure 9-5. A Typical System with Diagnostics Capabilities

The peripherals or the memory elements must observe the set-up and hold times of the i860 microprocessor during normal mode operation. This can be accomplished by giving multiple clocks during normal mode operation to meet the delay timing of the longest path. A typical timing diagram for a memory access is shown in Figure 9-6. (If the longest delay path is four clock cycles, for example, then by asserting the normal mode for five cycles, the address and the other controls are presented to the memory element, after which the data can be read or written.) The latching of the signals in the i860 CPU can be carried out reliably on the last clock edge, before the shift mode operations are performed.

RAM and other circuits external to the i860 microprocessor peripherals may consist of state machines. The user must ensure that the state machines are enabled or disabled at the appropriate time during boundary scan mode. One way to ensure this is to have separate clocks for the on-board state machines.

If the peripheral or memory elements have a serial scanning scheme as well, then the clock can be asserted for as briefly as one cycle during normal mode. This implies that the system will be responding to the test vectors of the previous cycle. The timings would be similar to those shown in Figure 9-6, with a smaller number of clock pulses for timeslots 2-3 and 4-5. Again, this assumes that the peripherals have a serial scan chain as well. This method reduces the number of clock pulses for the normal mode; however, the overhead in the number of external components is larger than in the first method.



TIMESLOT 1–2 LOAD TEST VECTOR SERIALLY–SHIFT.
2–3 OUTPUT THE TEST VECTOR, AND MONITOR THE INPUTS. THE INPUTS ARE LATCHED ON EVERY CLOCK CYCLE FROM CYCLE 128 TO CYCLE 133. THE EXTERNAL DEVICES MUST HAVE THE DATA SET UP AROUND THE RISING EDGE OF CLOCK NO. 133. THE RESPONSE TO TEST VECTOR 1 IS LATCHED FROM THE INPUT PINS.
3–4 LOAD IN TEST VECTOR 2 SERIALLY, AND SERIALLY SHIFT OUT THE RESPONSE FROM TEST VECTOR 1.
4–5 OUTPUT TEST VECTOR 2, AND MONITOR THE INPUTS. THE NUMBER OF CYCLES THAT SCAN IS IN THE LOW STATE IS SHORTER THAN THAT OF TIMESLOT 2–3 AND THIS IS BECAUSE THE EXTERNAL DEVICE HAS A FASTER ACCESS TIME. THE RESPONSE IS LATCHED ON CLOCK CYCLE 261 TO 265.
NOTE 1 INPUTS ARE SAMPLED IN NORMAL SCAN MODE ON EVERY RISING EDGE OF CLK.

240330–108

Figure 9-6. A Typical Timing for Serial Scan Mode

Graphics Frame Buffer Schematics and PLD Code*

A

*This design has been tested.

APPENDIX A
GRAPHICS FRAME BUFFER SCHEMATICS AND PLD CODE

SIGNAL	DESCRIPTION
RASx#	RAS# for buffer #x
RAS#	RAS1# ANDed with RAS0#
LRAS#	Delayed RAS#
CASx#	CAS# for buffer #x
WEx#	Write control for VRAMs
DTOEx#	Output control and SAM transfer request
CPEND#	Cycle pending
ROWE#	Row address enable
COLE#	Column address enable
WDEL#	Write delay
NCLK#	Timing control for WDEL#
REF	Refresh request
TRQ	SAM transfer request
TRFQ#	Conditioned refresh/SAM transfer request
RASDEL#	RAS# delay for CAS-/DTOE-before RAS
PRECH#	Controlling RAS# precharge time
REFC#	Controlling RAS# active time for TRFQ#
ERDL#	Expansion read data latch
LERDL#	Delayed ERDL#
EWLD#	Expansion write data latch
ERDE#	Expansion read data enable
EWDE#	Expansion write data enable
CLRTRQ#	Clearing TRQ
CLRREF#	Clearing REFRESH
SRAE#	Row address enable for SAM transfer
SCAE#	Column address enable for SAM transfer
HBLANK	Horizontal blank
HCOUNT	HBLANK count for generating vertical signal
HCLR#	Clear counter used to generate HBLANK
VCLR#	Clear counter used to generate VBLANK#.
VBLANK#	Vertical blank
SYNC#	Composite sync for video DACs
CBLANK#	Composite blank
SBLANK#	Composite blank synchronized to PCLK
VD Bus	Video data bus
LDSR#	Loading 4 pixels into shift registers
SC	Serial clock for shifting VRAM data out,16MHz
BLANK#	Blank signal for video DACs
VOG/VOR/VOB	RGB signals
PCLK	Pixel clock, 64 MHz

```
module FB_VRM flag '-r3'
title 'Buffer Select'

U1 device 'P16R6';

        VCC,GND,OCn                     pin   2Ø,1Ø,11;
        CLKC,VBFØn,RESET,VSYNC,CLRFRXØ
                                        pin   1,2,4,5,6;
        NC1,NC2,NC3,NC4,NC5,NSS1,NSS2 pin   3,7,8,9,12,13,18;
        SE1n,SEØn,VSYNCS,CLRFRQn,VSYNCn
                                        pin   14,15,16,17,19;

Equations
        CLRFRQn := CLRFRXØ;
        VSYNCS
              := VSYNC;
        VSYNCn = !VSYNC;
state_diagram   [SE1n,SEØn]
state [1,1]:    if(RESET # VSYNCS) then [1,1]
                        else if(!VSYNCS & VBFØn) then [Ø,1]
                        else if(!VSYNCS & !VBFØn) then [1,Ø]
                        else [1,1];
state [Ø,1]:    if(RESET # VSYNCS) then [1,1]
                        else [Ø,1];
state [1,Ø]:    if(RESET # VSYNCS) then [1,1]
                        else [1,Ø];
state [Ø,Ø]:    goto [1,1];

"Description:
"         Buffer switch happens at the time of vertical
"                 sync.
end FB_VRM;
```

```
module FB_VRM flag '-r3'
title 'Buffer Switch'

U2 device 'P16R4';

        VCC,GND,OCn                          pin   2Ø,1Ø,11;
        CLKC,RESET,LADSn,EXPSELn,A22,A3      pin   1,2,3,4,5,6;
        DRMBSYn,BSYn,EXPBSYn,LADSA22n        pin   7,8,9,12;
        CBUSYn,VBFØn,LRDYn,VSELXØ,VSELX1     pin   13,14,15,16,17;
        NC1,RDYn                             pin   18,19;

Equations
        RDYn = !(!VSELX1 & VSELXØ & DRMBSYn & BSYn & EXPBSYn) ;
        enable RDYn = !VSELX1 & DRMBSYn & BSYn & EXPBSYn;
        CBUSYn = BSYn & (DRMBSYn # !EXPBSYn);
        LADS22n = LADSn # A22;
        LRDYn := RDYn;

state_diagram [VSELX1,VSELXØ]
state [1,1]:      if(RESET) then [1,1]
                        else if(!EXPSELn & A22 & !LADSn) then [Ø,1]
                        else [1,1];
state [Ø,1]:      if(RESET # !RDYn & DRMBSYn & BSYn & EXPBSYn) then [Ø,Ø]
                        else [Ø,1];
state [Ø,Ø]:      if(!RESET & !EXPSELn & A22 & !LADSn) then [Ø,1]
                        else [1,1];
state [1,Ø]:      goto [1,1];

state_diagram [VBFØn]
state [1]:        if(RESET # !LADSn & !EXPSELn & A22 & !A3)
                  then [Ø]
             else [1];
state [Ø]:        if(!RESET & !LADSn & !EXPSELn & A22 & A3)
                  then [1]
             else [Ø];

"Description:
"        READYn is asserted when accesses direct to
"              the expansion space when A22 = 1. It is not
"              enabled until no busy signals are active, i.e.,
"              no pipelined cycles are pending.
end FB_VRM;
```

```
module FB_VRM flag '-r3'
title 'Write Enable'

U3 device 'P20L8';

        VCC,GND                                 pin 24,12;
        LWRn,LBE0n,LBE1n,LBE2n,LBE3n,LBE4n      pin 1,2,3,4,5,6;
        LBE5n,LBE6n,LBE7n,PRECHn,CAS0n          pin 7,8,9,10,11;
        RASn,WE0Xn,WE0n,WE1n,WE2n,WE3n          pin 13,14,15,16,17,18;
        WE4n,WE5n,WE6n,WE7n,WE7Xn               pin 19,20,21,22,23;

Equations
        WE0n = (LBE0n # !LWRn) & CAS0n # !PRECHn # RASn # WE0Xn & !CAS0n;
        WE1n = (LBE1n # !LWRn) & CAS0n # !PRECHn # RASn # WE1n & !CAS0n;
        WE2n = (LBE2n # !LWRn) & CAS0n # !PRECHn # RASn # WE2n & !CAS0n;
        WE3n = (LBE3n # !LWRn) & CAS0n # !PRECHn # RASn # WE3n & !CAS0n;
        WE4n = (LBE4n # !LWRn) & CAS0n # !PRECHn # RASn # WE4n & !CAS0n;
        WE5n = (LBE5n # !LWRn) & CAS0n # !PRECHn # RASn # WE5n & !CAS0n;
        WE6n = (LBE6n # !LWRn) & CAS0n # !PRECHn # RASn # WE6n & !CAS0n;
        WE7n = (LBE7n # !LWRn) & CAS0n # !PRECHn # RASn # WE7Xn & !CAS0n;

"Description:
"        The write enables are not asserted when RASn goes active
"        during CAS-Before-RAS refresh and write cycles
"        starting up (to disable write-per-bit).

end FB_VRM;
```

```
module FB_VRM flag '-r3'

title 'CRT Clocks'
U7 device 'P16R8';

        VCC,GND,OCn                         pin   2Ø,1Ø,11;
        PCLK,VCLRn,HCLRn,RESET              pin   1,2,3,4;
        NC1,NC2,NC3,NC4,NC5                 pin   5,6,7,8,9;
        HQØ,VQØ,CDIV16,CDIV8,CDIV4,CDIV2    pin   12,13,14,15,16,17;
        HCLRXn,VCLRXn                       pin   18,19;

        CDIV = [CDIV16,CDIV8,CDIV4,CDIV2];

equations

        CDIV := (CDIV + 1) & !RESET;

state_diagram [HCLRXn,HQØ]
state [1,1]:    if(RESET) then [1,1]
                          else if(!HCLRn) then [Ø,1]
state [Ø,1]:    goto [1,Ø];
state [1,Ø]:     if(RESET # HCLRn) then [1,1]
                          else [1,Ø];
state [Ø,Ø]:    goto [1,1];

state_diagram [VCLRXn,VQØ]
state [1,1]:    if(RESET) then [1,1]
                          else if(!VCLRn) then [Ø,1]
state [Ø,1]:    goto [1,Ø];
state [1,Ø]:    if(RESET # VCLRn) then [1,1]
                          else [1,Ø];
state [Ø,Ø]:    goto [1,1];

"Description:
"       HCLRXn and VCLRXn clear the counters for horizontal and
"       vertical timings.

end FB_VRM;
```

```
module FB_VRM flag '-r3'
title 'Ready Logic'

U8 device 'P20R4';

        VCC,GND,OCn                     pin   24,12,13;
        CLK,RASn,COLEn,LADSn,WRn,TRFQn
                                        pin   1,2,3,4,5,6;
        CPENDn,WDELXØ,WDELn,NC1,EXPBSYn
                                        pin   7,8,9,1Ø,11;
        LERDLn,NC2,WAIT,LRASn,LWRn,NAXØ
                                        pin   14,15,16,17,18,19;
        RDYXØ,RESET,RDYn,NENEn          pin   2Ø,21,22,23;

        RIDLE = 1;
        RACT = Ø;

equations

       LRASn := RASn;
       RDYn = !((WAIT & !RDYXØ & TRFQn & !COLEn
                   & WDELn & !CPENDn & LWRn) # (!LERDLn));
       enable RDYn = !EXPBSYn;

state_diagram    [RDYXØ]

state    RIDLE: if (RASn) then RIDLE
                          else if(!RASn & TRFQn) then RACT
                          else RIDLE;
state    RACT : if(RASn # !RDYn & !RASn # !LADSn & NENEn) then RIDLE
                          else RACT;

state_diagram   [NAXØ]

state   [1]:    if (RASn) then [1]
                          else if(WAIT & !CPENDn & !LWRn & !COLEn & !RASn)
                          then [Ø]
                          else [1];
state   [Ø] :   goto [1];

state_diagram [LWRn]
state [1]:      if(!LADSn & !WRn) then [Ø]
                          else [1];
state [Ø]:      if(!LADSn & WRn) then [1]
                          else [Ø];
End FB_VRM;
```

```
module FB_VRM flag '-r3'
title 'Output Enables'

U9 device 'P20R8';

        VCC,GND,OCn                          pin   24,12,13;
        CLK,DREF,EXPSELn,CPENDn,LADSA22n,TRFQn
                                             pin   1,2,3,4,5,6;
        WRn,LWRn,RASn,NC1,NC2,REF,NC3        pin 7,8,9,10,11,14,15;
        CLRRFX0,DTOE0n,DTOE1n,EWDEn,ERDEX0
                                             pin 16,17,18,19,20;
        CLRTRQn,NC4,PRECHn                   pin 21,22,23;

state_diagram [DTOE1n]
state [1]:        if (!CPENDn & !LWRn & !RASn
                        # RASn & !TRFQn & PRECHn & !REF) then [0]
                  else [1];
state [0]:        if (DREF # !LADSA22n & WRn & !EXPSELn) then [1]
                  else [0];
state_diagram [DTOE0n]
state [1]:        if (!CPENDn & !LWRn & !RASn
                        # RASn & !TRFQn & PRECHn & !REF) then [0]
                  else [1];
state [0]:        if (DREF # !LADSA22n & WRn & !EXPSELn) then [1]
                  else [0];
state_diagram [CLRRFX0]
state [1]:        if (RASn & REF & PRECHn & !TRFQn)
                  then [0]
                  else [1];
state [0]: goto [1];

state_diagram [CLRTRQn]
state [1]:       if(!DTOE1n & PRECHn & !RASn)
                  then [0]
                  else [1];
state [0]:       goto [1];

state_diagram [ERDEX0]
state [1]:       if (RASn) then [1]
                 else if(!RASn & !LWRn & TRFQn
                           # !RASn & !LADSA22n & !WRn & !EXPSELn) then [0]
                  else [1];
state [0]:       if (!LADSA22n & WRn & !EXPSELn # RASn) then [1]
                  else [0];

state_diagram [EWDEn]
state [1]:       if(RASn) then [1]
                  else if(!RASn & LWRn) then [0]
                  else [1];
state [0]:       if (!LADSA22n & !WRn & !EXPSELn # RASn) then [1]
                  else [0];
```

```
"Description:
"         ERDEXØ is an intermediate signal to generate ERDEn.
"         CLRRFXØ is an intermediate signal to generate the
"               CLear_Refresh_ReQuest (CLRRFQ)
"
end FB_VRM;
```

```
module FB_VRM flag '-r3'
title 'Random Logic'

U1Ø device 'P2ØL8';

        VCC,GND                                 pin   24,12;
        RAS1n,RASØn,SCAEn,TRFQn,PRECHn          pin   1,2,3,4,5;
        REF,CAS1n,CASX1,ERDEXØ,WAIT,REFCn       pin   6,7,8,9,1Ø,11;
        LWRn,WRn,DREF,ERDEn,RASDELn,TREF        pin   13,14,15,16,17,18;
        NC1,LADSBn,RASn,PREF,LADSn              pin   19,2Ø,21,22,23;

Equations
       RASn = RAS1n & RASØn;
       TREF = !TRFQn & REFCn & CAS1n
                        # !SCAEn & REFCn
                        # LADSn & REFCn & !CAS1n & !CASX1 & LWRn & WAIT & TRFQn
                        # LADSn & REFCn & !CAS1n & !CASX1 & !LWRn & TRFQn;
       PREF = (!RAS1n # !RASØn) & !SCAEn & RASDELn
                        # !TRFQn & REF & RAS1n & RASØn & PRECHn & RASDELn;
       DREF = RAS1n & RASØn & REFCn
                        # !SCAEn & REFCn;
       ERDEn = ERDEXØ & (LADSn # WRn # RAS1n & RASØn);
       LADSBn = LADSn;

"Description:
"         TREF feeds into the RAS state machine. The !TRFQn term
"                is for CAS-Before-RAS refresh; the !SCAEn term
"                for serial register load inside the VRAMs.
"         PREF feeds into the CAS state machine. The !TRFQn term is
"                for VRAM refresh request. The !SCAEn term is for
"                serial register load inside the VRAMs.
"         DREF feeds into the Output Enable logic.
"         LADSBn is for non-critical logic.
end FB_VRM;
```

```
module FB_VRM flag '-r3'
title 'Composite Sync and Blank'

u11 device 'P22V10';

        VCC,GND                              pin   24,12;
        HBLANK,NC1,HSYNCn,                   pin   1,2,3;
        HB0,HB1,HB2,HB3,RESET                pin   4,5,6,7,8;
        NC2,NC3,NC4,NC5                      pin   9,10,11,13;
        VCLRn,HCNT2,HCNT0,HCNT4,HCNT5        pin   14,15,16,17,18;
        VBLANKn,VSYNC,HCNT1,HCNT3,CSYNCn     pin   19,20,21,22,23;

        HB = [HB3,HB2,HB1,HB0];
        HCNT = [HCNT5,HCNT4,HCNT3,HCNT2,HCNT1,HCNT0];

equations

        WHEN ((HB == 15) & !(HCNT == 50) & !RESET)
                THEN HCNT := (HCNT + 1);
            ELSE HCNT := HCNT & !((HCNT == 50) & (HB == 12)) & !RESET;

        CSYNCn = !VSYNC & HSYNCn;

state_diagram [VBLANKn]
state [0]:       if((HCNT == 2) & (HB ==12) & !RESET) then [1]
                      else [0];                    "Blank for 45 horizontal
                                                          "lines
state [1]:       if((HCNT == 50) & (HB ==12) # RESET) then [0]
                      else [1];                    "Active time = 768
                                                          "horizontal lines

state_diagram [VSYNC]
state [0]:      if((HCNT == 0) & (HB ==5) & !RESET) then [1]
                      else [0];                     "Front Porch = 6 H lines
state [1]:      if((HCNT == 0) & (HB ==11) # RESET) then [0]
                      else [1];                     "Active for 6 H lines

state_diagram [VCLRn]
state [1]:      if((HCNT == 50) & (HB == 12) # RESET) then [0]
                      else [1];
state [0]:      if(!RESET) then [1]
                      else [0];

end FB_VRM;
```

```
module FB_VRM flag '-r3'
title 'Horizontal Sync and Blank'

u14 device 'P16R6';

        VCC,GND,OCn                             pin   2Ø,1Ø,11;
        CDIV16,CC6,CC5,CC4,CC3,CC2,CC1,CCØ      pin   1,2,3,4,5,6,7,8;
        VBLANKn,NC1,HCLRn,CBLANKn,HSYNCn        pin   9,12,13,14,15;
        EXPINTn,EINTXØ,HBLANK,RESET             pin   16,17,18,19;

        CC = [CC6,CC5,CC4,CC3,CC2,CC1,CCØ];

state_diagram [EXPINTn,EINTXØ]
state [1,1]:      if(RESET) then [1,1]
                          else if(!VBLANKn) then [Ø,1]
                          else [1,1];
state [Ø,1]:      goto [1,Ø];
state [1,Ø]:      if(RESET # VBLANKn) then [1,1]
                          else [1,Ø];
state [Ø,Ø]:      goto [1,1];

state_diagram [HBLANK]                          "Horizontal Blank"
state [1]:        if(RESET) then [1]
                          else if(CC == 17) then [Ø]
                          else [1];
state [Ø]:        if((CC == 81) # RESET) then [1]
                          else [Ø];

state_diagram [HCLRn]
state [1]:        if((CC == 81) # RESET) then [Ø]
                          else [1];
state [Ø]:        if(!RESET) then [1]
                          else [Ø];

state_diagram [HSYNCn]                          "Horizontal Sync"

state [1]:        if (RESET) then [1]
                          else if(CC == 1) then [Ø]
                          else [1];
state [Ø]:        if ((CC == 7) # RESET) then [1];
                          else [Ø];

state_diagram [CBLANKn]                         "Composite Blank"
state [1]:        if(HBLANK # !VBLANKn # RESET) then [Ø]
                          else [1];
state [Ø]:        if(!HBLANK & VBLANKn & !RESET) then [1]
                          else [Ø];

"Description:
"         CDIV16 is a 4MHz clock; therefore, horizontal scan period
"             is about 2Ø.5 usec and the active display time is 16.Ø
"         usec (each CC count is Ø.25 usec.).

end FB_VRM;
```

```
module FB_VRM flag '-r3'
title 'Write Data Latch'

U25 device 'P16R4';

        VCC,GND,OCn pin 20,10,11;
        CLK,RESET,TRFQn,COLEn,NAX0,CPENDn pin 1,2,3,4,5,6;
        WDELn,LWRn,RASn,ERDLn,WAIT,LERDLn pin 7,8,9,12,13,14;
        EWDLX0,EWDLn,EXPBSYn,NC1,NAn pin 15,16,17,18,19;

Equations
        NAn = NAX0 # !TRFQn;
        enable NAn = !EXPBSYn;
        LERDLn := ERDLn;

state_diagram [EXPBSYn]
state [1]:      if (RESET) then [1]
                        else if (!RASn & TRFQn) then [0]
                        else [1];
state [0]:      if(RESET # CPENDn & RASn & COLEn & TRFQn) then [1]
                        else [0];

state_diagram [EWDLn,EWDLX0]
state [1,1]:    if (RESET) then [1,1]
                        else if (!RASn & LWRn & TRFQn) then [0,1]
                        else [1,1];
state [0,1]:    if(RESET # RASn # !WAIT) then [1,1]
                        else [1,0];
state [1,0]:    if (WDELn) then [1,1]
                        else [0,1];
state [0,0]:    goto [1,1];

"Description:
"       NAn is driven only when it is a VRAM cycle.
"    EWDLn is asserted whenever, it is a write cycle.

End FB_VRM;
```

```
module FB_VRM flag
'-r3' title 'Cycle Pending'

U26 device 'P16R6';

        VCC,GND,OCn                             pin   2Ø,1Ø,11;
        CLK,LADS22n,RESET,TRFQn,RDYn,CAS1n,RASn
                                                pin   1,2,3,4,5,6,7;
        EXPSELn,WRn,NC1,WDELXØ,WDELn,NCLKXØ
                                                pin   8,9,12,13,14,15;
        NCLKn,NC2,CPENDn,WAIT                   pin   16,17,18,19;

state_diagram [CPENDn]

state [1]:      if (RESET) then [1]
                        else if(!LADS22n & !EXPSELn) then [Ø]
                        else [1];
state [Ø]:      if(LADS22n & !CAS1n & TRFQn
                            # EXPSELn & !CAS1n & TRFQn # RESET) then [1]
                        else [Ø];

state_diagram [WDELn, WDELXØ]
state [1,1]:    if (RESET) then [1,1] "IDLE
                        else if(!LADS22n & !WRn & !EXPSELn
                            # !CPENDn & !WRn & !EXPSELn & !RASn)
                            then [1,Ø] "INACT
                        else [1,1];
state [1,Ø]:    if (RESET # RASn) then [1,1]
                     else if(!LADS22n & WRn & !EXPSELn) then [Ø,1] "ACTIVE1
                        else [1,Ø];
state [Ø,1]:    if (RASn) then [1,1]
                        else if(!NCLKXØ) then [Ø,Ø]
                        else [Ø,1];
state [Ø,Ø]:    goto [1,1];

state_diagram [NCLKn, NCLKXØ]
state [1,1]:    if(RESET) then [1,1] "IDLE
                        else if(!RDYn) then [Ø,1] "INACT
                        else [1,1];
state [Ø,1]:    goto [Ø,Ø];"ACTIVE
state [Ø,Ø]:    if(RESET # RDYn) then [1,1]
                        else if (!RDYn) then [Ø,1]
                        else [Ø,Ø];
state [1,Ø]:    goto [1,1];
```

```
"Description:
"       CPENDn (cycle pending) is deasserted only when CASxn has
"              been asserted for the cycle.
"       WDELn is for freezing the state machines when a write
"              cycle is pipelined right after a read cycle. The
"              i860 processor does not drive the data bus until
"              one clock after READYn is asserted for the
"              previous read cycle.

end FB_VRM;
```

```
module FB_VRM flag '-r3'
title 'Refresh Requests'

U27 device 'P16R8';

        VCC,GND,OCn                             pin 20,10,11;
        CLKG,PRECHn,REF,TRQ,RESET,RASn          pin 1,2,3,4,5,6;
        CLRTRQn,RASDELn,RDYn,ROWEn,SRAEn        pin 7,8,9,12,13;
        TRFQX0,TRFQn,REFCX0,REFCn,SCAEn,COLEn
                                                pin 14,15,16,17,18,19;

        TI = 3;
        TA1 = 1;
        TA2 = 0;
        TI1 = 2;

        RIDLE = 3;
        RINACT1 = 1;
        RINACT2 = 0;
        RINACT3 = 2;

state_diagram [REFCn, REFCX0]
state RIDLE:     if(RESET) then RIDLE
                        else if (RASn & !TRFQn) then RINACT1
                        else RIDLE;
state RINACT1:   if(RESET) then RIDLE
                        else if (!RASn & RASDELn) then RINACT2
                        else RINACT1;
state RINACT2:   goto RIDLE;
state RINACT3:   goto RIDLE;

state_diagram [TRFQn,TRFQX0]
state TI:        if(RESET) then TI
                        else if (REF & RDYn # TRQ & RDYn) then TA1
                        else TI;
state TA1:       if (RESET) then TI
                        else if (!RASn & !REFCn) then TA2
                        else TA1;
state TA2:       if (RESET # REFCn) then TI
                        else TA2;
state TI1:       goto TI;

state_diagram [SRAEn]
state [1]:       if(RESET) then [1]
                        else if(RASn & !TRFQn & !REF & PRECHn) then [0]
                        else [1];
state [0]:       if(!RASn # RESET) then [1]
                        else [0];
```

```
state_diagram [SCAEn]
state [1]:      if(RESET) then [1] else if(!CLRTRQn) then [Ø]
                        else [1];
state [Ø]:      if(REFCn # RESET) then [1]
                        else [Ø];

state_diagram [ROWEn]
state [Ø]:      if(RESET) then [Ø]
                        else if(!RASn # !TRFQn) then [1]
                        else [Ø];
state [1]:      if (RESET # TRFQn & COLEn & RASn) then [Ø]
                        else [1];

state_diagram [COLEn]
state [1]:      if (RESET) then [1]
                        else if(ROWEn & !RASn & TRFQn) then [Ø]
                        else [1];
state [Ø]:      if (RASn) then [1]
                        else [Ø];

"Description:
"       ROWEn enables row address while COLEn enables column
"             address for i86Ø processor cycles. SRAEn and
"       SCAEn enable the serial row and column addresses
"             for serial register load cycles.
"       TRFQn combines the VRAM refresh and serial register load
"             requests.

end FB_VRM;
```

```
module FB_VRM flag '-r3'
title 'CAS Logic'

U28 device 'P16R6';

        VCC,GND,OCn                             pin   2Ø,1Ø,11;
        CLK,CPENDn,RASn,WAIT,LWRn,WDELn         pin   1,2,3,4,5,6;
        COLEn,TRFQn,PREF,RESET,ERDLn            pin   7,8,9,12,13;
        CASXØ,CASX1,CASØn,CAS1n,CAS2n,NC1       pin   14,15,16,17,18,19;

        CIDLE = ^b11111;
        CACTIVE1 = ^bØØØ11;
        CACTIVE2 = ^bØØØ1Ø;
        CACTIVE3 = ^bØØØØ1;
        CNS1 = Ø;
        CNS2 = 4;
        CNS3 = 5;
        CNS4 = 6;
        CNS5 = 7;
        CNS6 = 8;
        CNS7 = 9;
        CNS8 = 1Ø;
        CNS9 = 11;
        CNS1Ø = 12;
        CNS11 = 13;
        CNS12 = 14;
        CNS13 = 15;
        CNS14 = 16;
        CNS15 = 17;
        CNS16 = 18;
        CNS17 = 19;
        CNS18 = 2Ø;
        CNS19 = 21;
        CNS2Ø = 22;
        CNS21 = 23;
        CNS22 = 24;
        CNS23 = 25;
        CNS24 = 26;
        CNS25 = 27;
        CNS26 = 28;
        CNS27 = 29;
        CNS28 = 3Ø;

state_diagram [CAS2n, CAS1n, CASØn, CASX1, CASXØ]

state     CIDLE:     if (RESET) then CIDLE
                        else if(!RASn & !COLEn & !CPENDn & TRFQn & WDELn
                            # PREF) then CACTIVE1
                        else CIDLE;
```

```
state     CACTIVE1 :if(WAIT & LWRn & !RASn & !COLEn
                              # !WAIT & !LWRn & !RASn & !COLEn
                              # !TRFQn & COLEn)
                              then CACTIVE3
                         else if(!RASn & WAIT & !LWRn & !COLEn) then CACTIVE2
                         else if(!RASn & !WAIT & LWRn & !COLEn # RESET) then
CIDLE
                              else CACTIVE1;

state     CACTIVE2: goto CACTIVE3;
state     CACTIVE3: goto CIDLE;
state     CNS1:   goto CIDLE;
state     CNS2:   goto CIDLE;
state     CNS3:   goto CIDLE;
state     CNS4:   goto CIDLE;
state     CNS5:   goto CIDLE;
state     CNS6:   goto CIDLE;
state     CNS7:   goto CIDLE;
state     CNS8:   goto CIDLE;
state     CNS9:   goto CIDLE;
state     CNS1Ø:  goto CIDLE;
state     CNS11:  goto CIDLE;
state     CNS12:  goto CIDLE;
state     CNS13:  goto CIDLE;
state     CNS14:  goto CIDLE;
state     CNS15:  goto CIDLE;
state     CNS16:  goto CIDLE;
state     CNS17:  goto CIDLE;
state     CNS18:  goto CIDLE;
state     CNS19:  goto CIDLE;
state     CNS2Ø:  goto CIDLE;
state     CNS21:  goto CIDLE;
state     CNS21:  goto CIDLE;
state     CNS23:  goto CIDLE;
state     CNS24:  goto CIDLE;
state     CNS25:  goto CIDLE;
state     CNS26:  goto CIDLE;
state     CNS27:  goto CIDLE;
state     CNS28:  goto CIDLE;

state_diagram [ERDLn]

state [1]:        if(RESET) then [1]
                     else if(!WAIT & !COLEn & !LWRn & !CAS1n & CASX1 & CASXØ
                                 # WAIT & !COLEn & !LWRn & !CAS1n & CASX1
                                 & !CASXØ) then [Ø]
                     else [1];
state [Ø]:        goto [1];
```

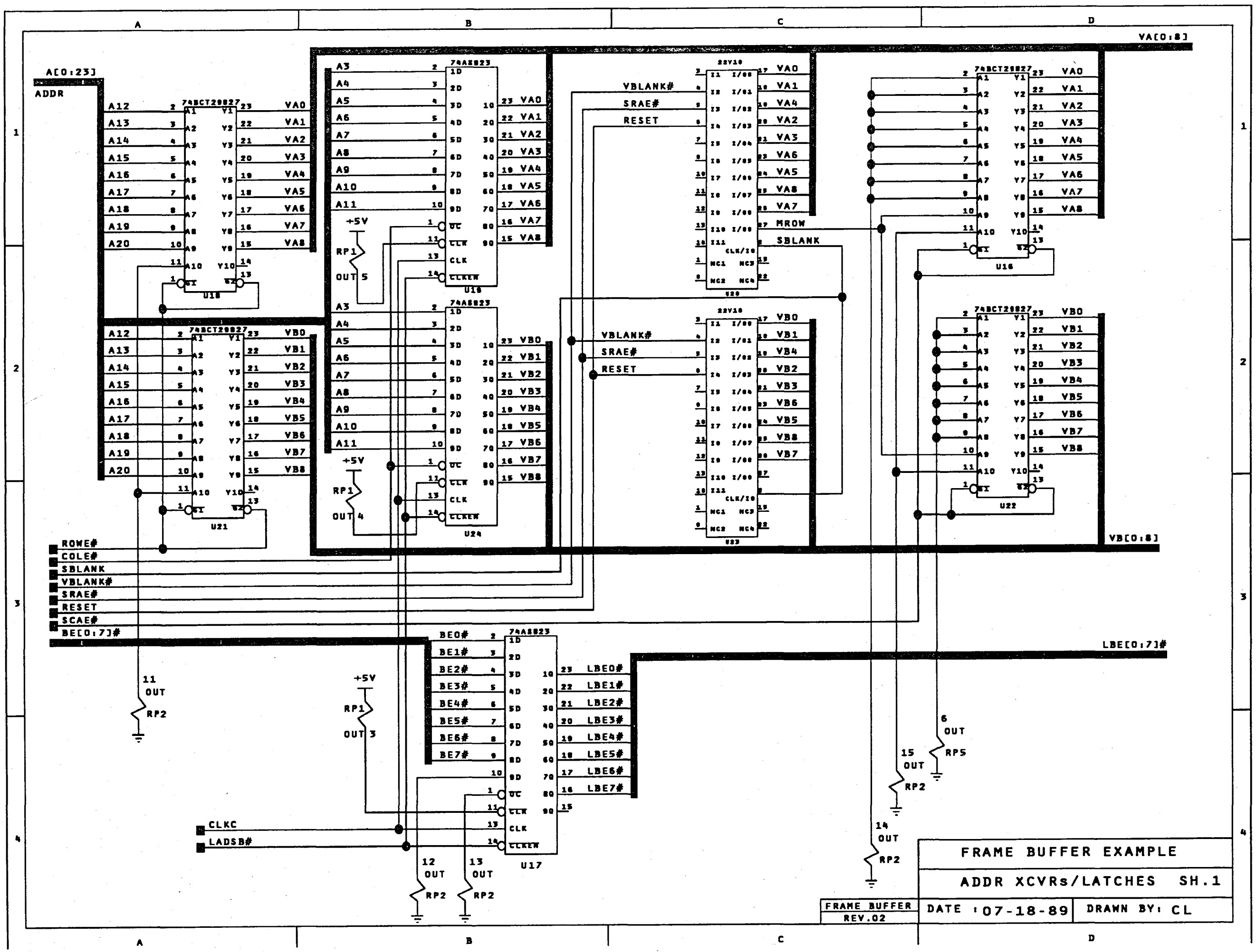
```
"Description:
"        The SCAEn term in the CAS state machine is for serial
"               register load cycle while the REF term for
"              CAS-Before-RAS refresh request.

end FB_VRM;
```

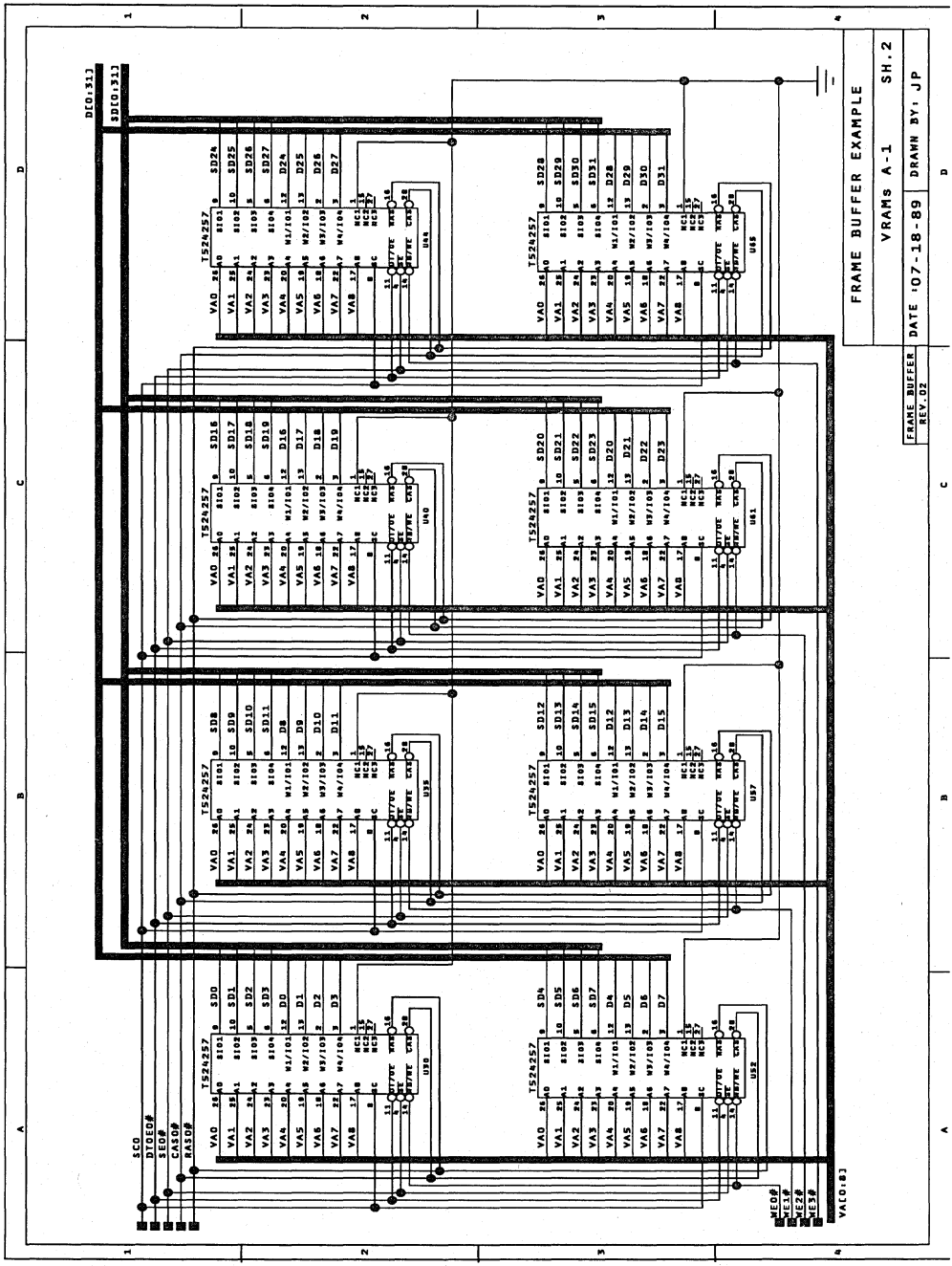
```
module FB_VRM flag '-r3'
title 'Serial Clocks'

U29 device 'P16R8';

        VCC,GND,OCn                     pin   2Ø,1Ø,11;
        PCLK,SBLANKn,SCØ,SC1,LDSRn,SCXØ
                                        pin   1,2,12,13,14,15;
        NC1,NC2,NC3,NC4,NC5,NC6,NC7     pin   3,4,5,6,7,8,9;
        BLANKn,BLANKX1,BLANKXØ,NSS1
                                        pin 16,17,18,19;

        SI  = ^bØØ11;
        SA1 = ^b1111;
        SA2 = ^b111Ø;
        SI1 = ^bØØØØ;
        SNS1 = 5;
        SNS2 = 4;
        SNS3 = 2;
        SNS4 = 1;
        SNS5 = 6;
        SNS6 = 7;
        SNS7 = 8;
        SNS8 = 9;
        SNS9 = 1Ø;
        SNS1Ø = 11;
        SNS11 = 12;
        SNS12 = 13;

        BI  = ^bØ11;
        BI1 = ^b111;
        BI2 = ^b11Ø;
        BI3 = ^b1Ø1;
        BI4 = ^b1ØØ;
        BNS1 = 2;
        BNS2 = 1;
        BNS3 = Ø;

equations

state_diagram [SC1,SCØ,LDSR,SCXØ]
state SI:         if(SBLANKn) then SA1
                  else SI;
state SA1:        goto SA2;
state SA2:        goto SI1;
state SI1:        goto SI;
state SNS1:       goto SI;
state SNS2:       goto SI;
state SNS3:       goto SI;
state SNS4:       goto SI;
state SNS5:       goto SI;
state SNS6:       goto SI;
```

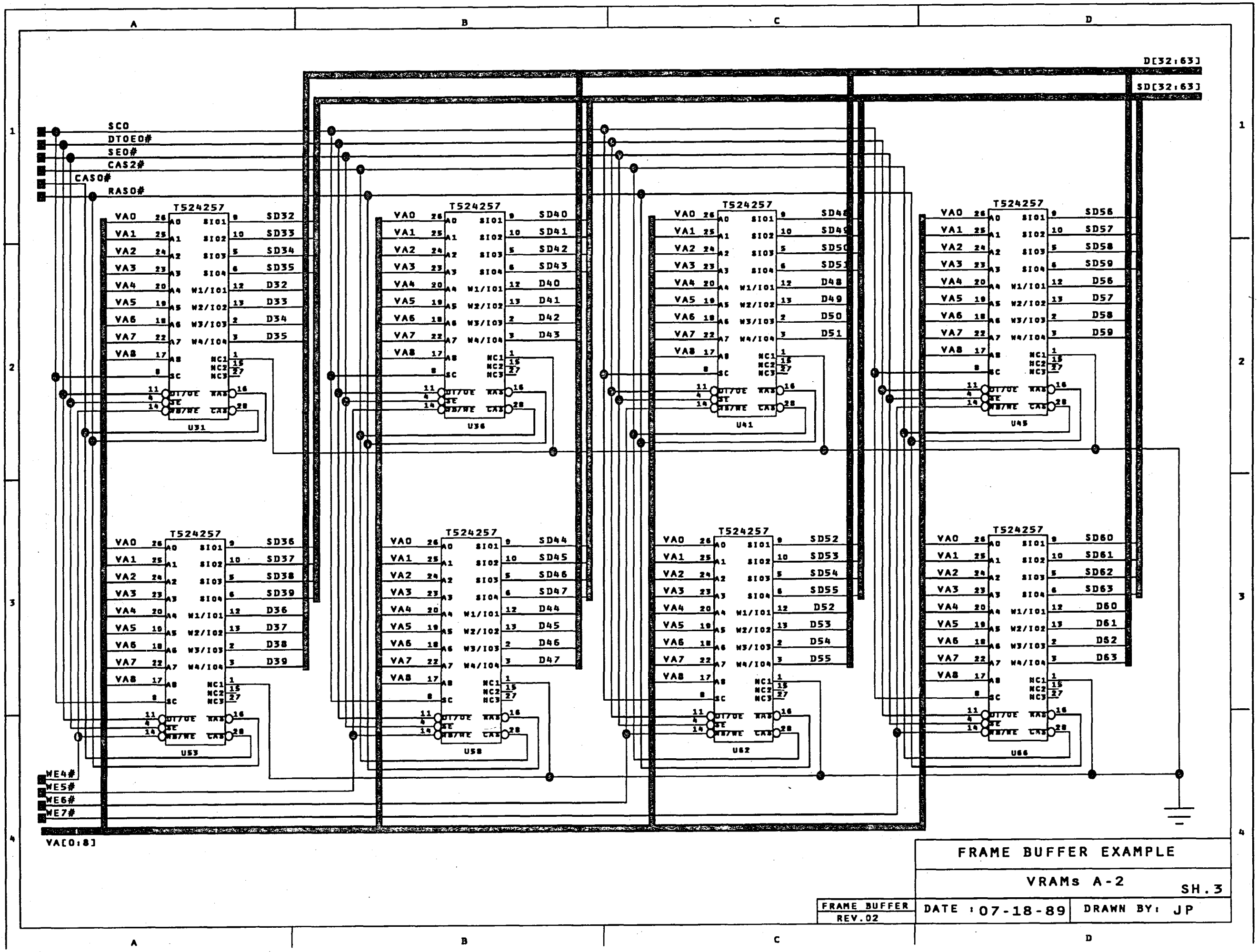
```
state SNS7:       goto SI;
state SNS8:       goto SI;
state SNS9:       goto SI;
state SNS1Ø:      goto SI;
state SNS11:      goto SI;
state SNS12:      goto SI;

state_diagram [BLANKn,BLANKX1,BLANKXØ]
state BI:         if(SBLANKn) then BI1 else BI;
state BI1:        if(!SBLANKn) then BI2 else BI1;
state BI2:        goto BI3;
state BI3:        goto BI4;
state BI4:        goto BI;
state BNS1:       goto BI;
state BNS2:       goto BI;
state BNS3:       goto BI;

"Description:
"        SCØ and SC1 control the serial register loading inside
"               the VRAMs.
"        BLANKn feeds into the video DAC to blank out the monitor
"               screen.

end FB_VRM;
```

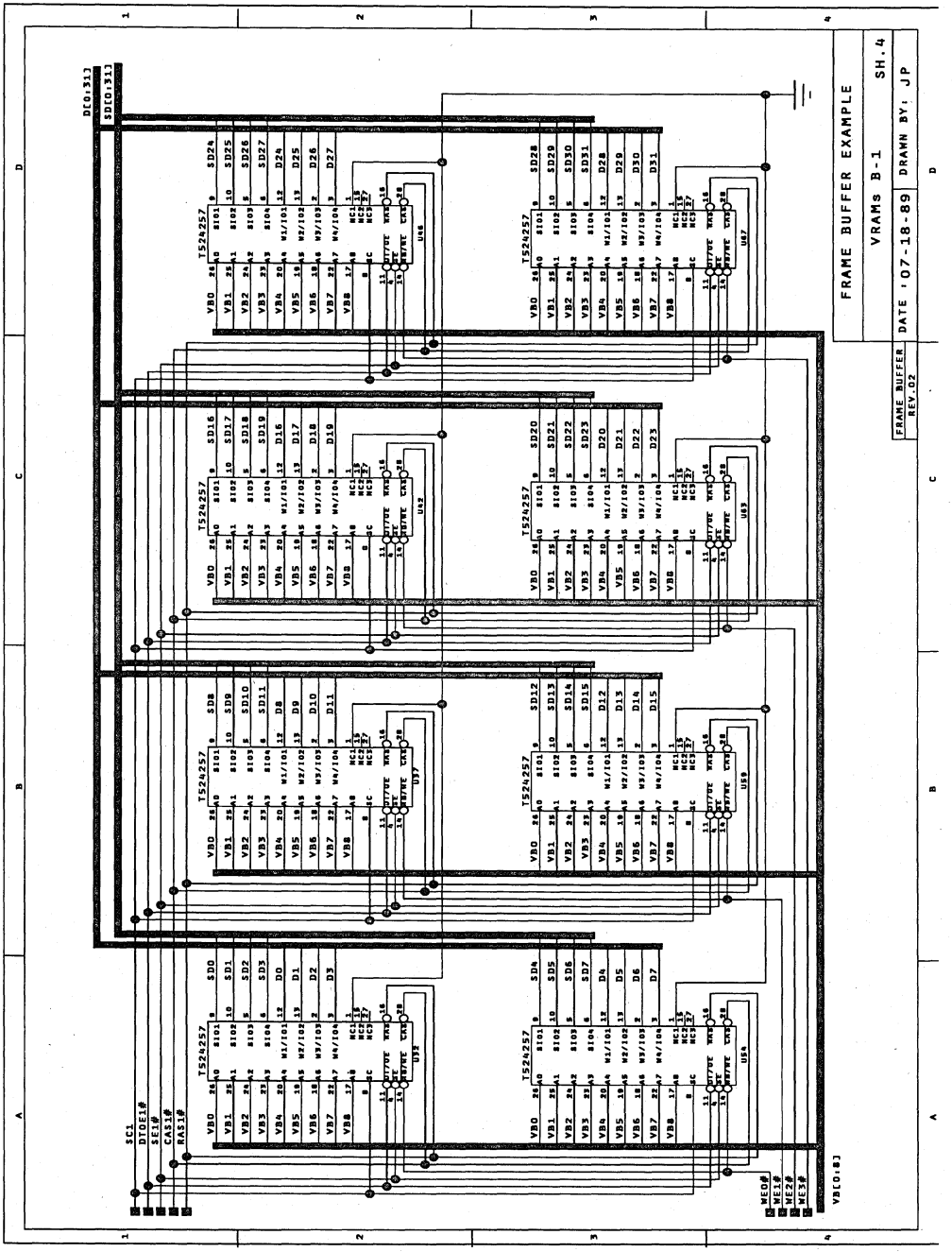
```
module FB_VRM flag '-r3'
title 'RAS Logic'

U34 device 'P2ØR8';

        VCC,GND,OCn                             pin 24,12,13;
        CLK,LADSn,NENEn,A22,A21,CBUSYn,TRFQn
                                                pin 1,2,3,4,5,6,7;
        TREF,RESET,EXPSELn,CPENDn,NC1,NC2       pin 8,9,1Ø,11,14,15;
        PRCHXØ,PRECHn,NC3,RASDELn,RAS1,RASØn
                                                pin 16,17,18,19,2Ø,21;
        NC4,NC5                                 pin 22,23;

        IDLE = 1;
        ACTIVE = Ø;

        PIDLE = 3;
        PACT1 = 1;
        PACT2 = Ø;
        PNS1 = 2;

state_diagram [RASØn]

state    IDLE:    if(!LADSn & !EXPSELn & !A22 & !A21 & CBUSYn
                            & TRFQn & PRECHn
                        # !CPENDn & !EXPSELn & !A22 & !A21 & PRECHn
                            & CBUSYn & TRFQn
                        # !TRFQn & PRECHn & !RASDELn) then ACTIVE
                    else IDLE;
state   ACTIVE :if ( TREF
                        # RESET
                        # !LADSn & EXPSELn & TRFQn
                        # !LADSn & A22 & TRFQn
                        # !LADSn & A21 & TRFQn
                        # !LADSn & !EXPSELn & !A22 & !A21
                                & NENEn & TRFQn)
                        then IDLE
                    else ACTIVE;

state_diagram     [RAS1n]
state     IDLE:   if(!LADSn & !EXPSELn & !A22 & A21 & CBUSYn
                                & TRFQn & PRECHn
                        # !CPENDn & !EXPSELn & !A22 & A21 & PRECHn
                                & CBUSYn & TRFQn
                        # !TRFQn & PRECHn & !RASDELn) then ACTIVE
                    else IDLE;
```

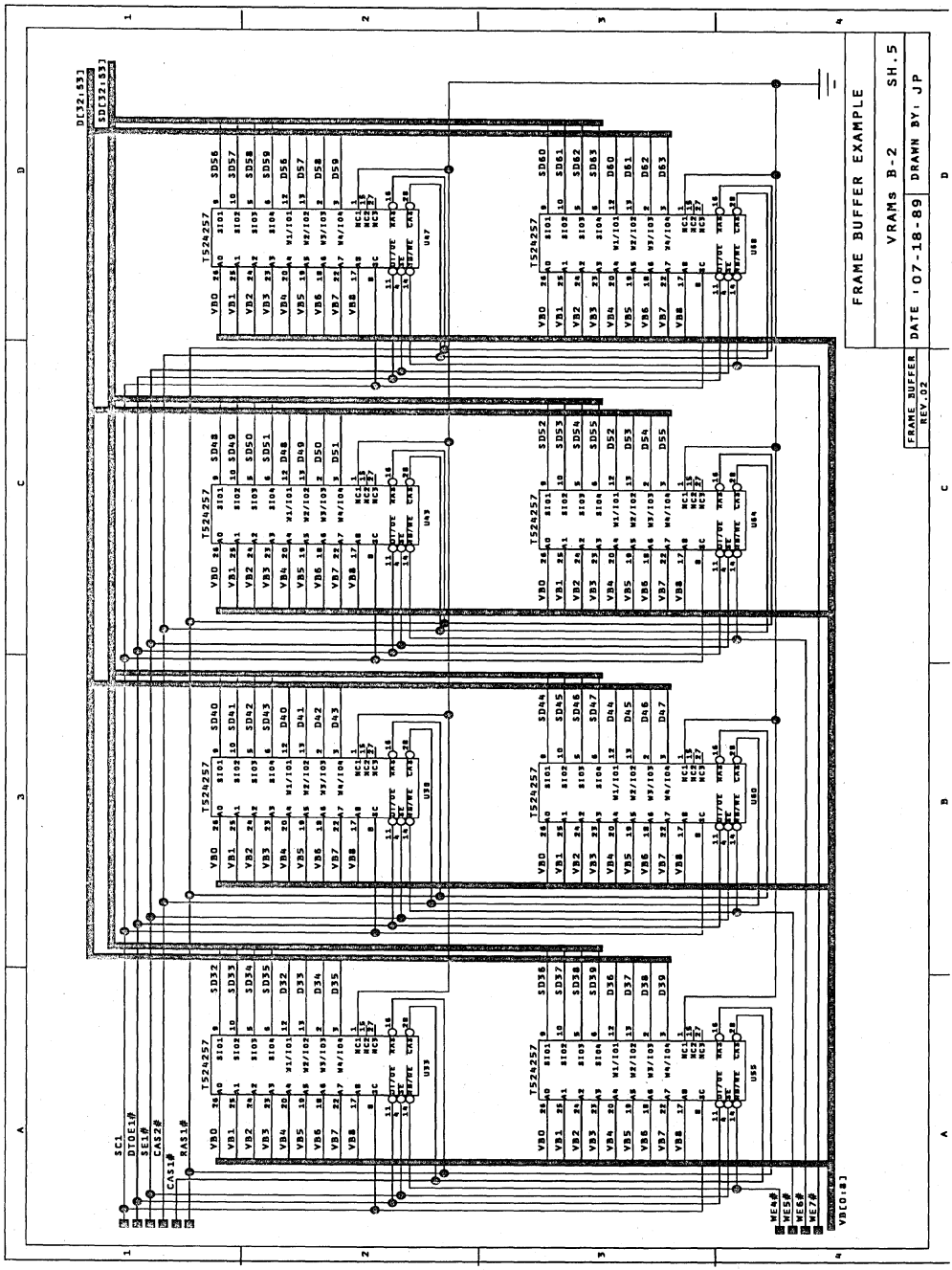
```
state    ACTIVE :if ( TREF
                          # RESET
                       # !LADSn & EXPSELn & TRFQn
                          # !LADSn & A22 & TRFQn
                          # !LADSn & !A21 & TRFQn
                          # !LADSn & !EXPSELn & !A22 & A21
                                   & NENEn & TRFQn)
                          then IDLE
                     else ACTIVE;

state_diagram     [RASDELn]
state [1]:           if (RESET) then [1]
                          else if (!TRFQn & PRECHn) then [Ø]
                          else [1];
state [Ø]:           if(RESET # !PRECHn) then [1]
                          else [Ø];

state_diagram [PRECHn, PRCHXØ]
state PIDLE:         if (RESET) THEN PIDLE
                          else if (!RASØn # !RAS1n) then PACT1
                          else PIDLE;
state PACT1:         if (RESET) THEN PIDLE
                          else if (RAS1n & RASØn) then PACT2
                          else PACT1;
state PACT2:         goto PIDLE;
state PNS1:          goto PIDLE;

"Description:
"         RASxn is asserted only for VRAM cycles and only when no
"                  other non-VRAM cycles are pending.
"         RASDELn delays RASxn activation when refresh and serial
"                  register load requests are detected.
"         PRECHn guarantees minium RASxn precharge time.

end N1Ø_VRM;
```

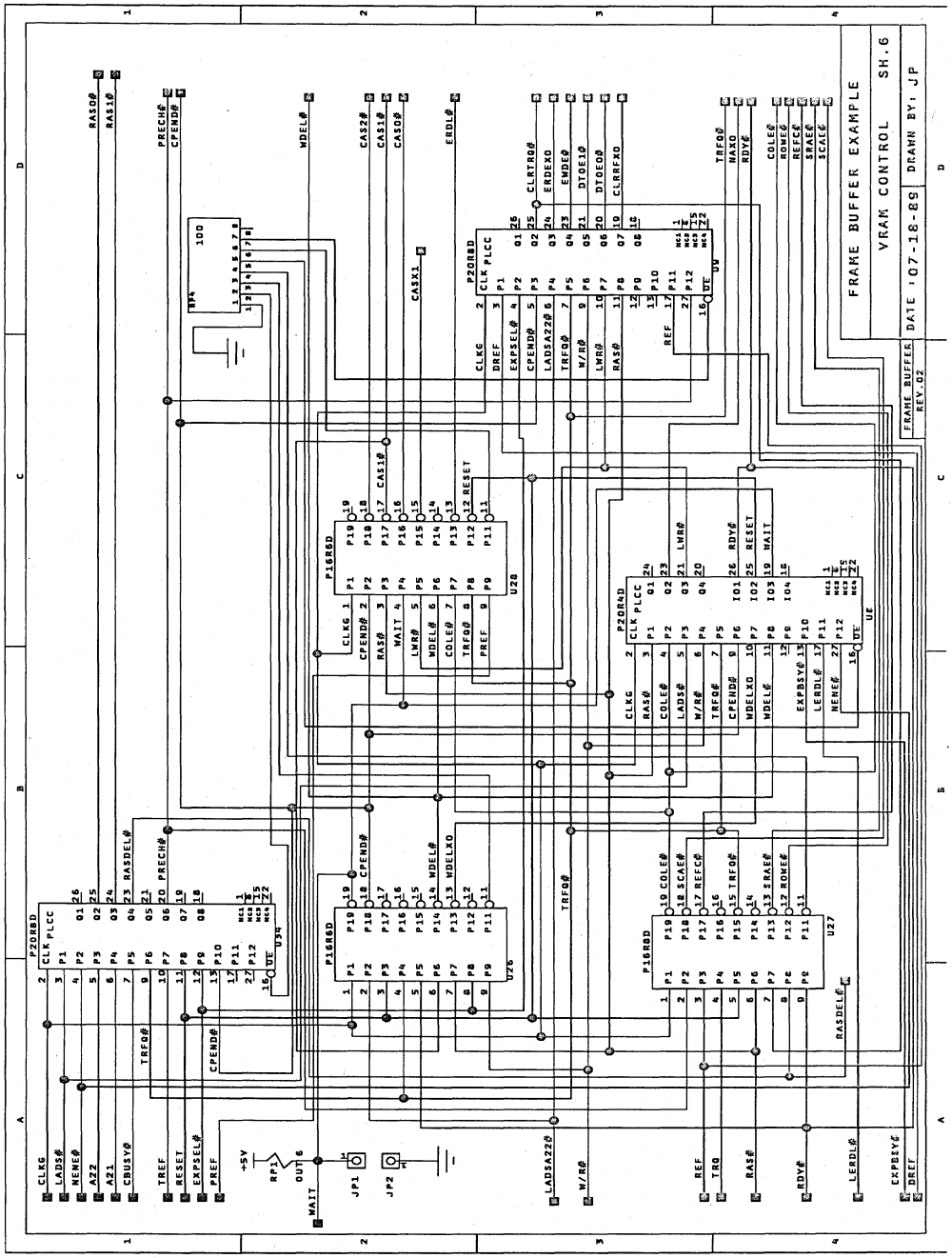

A[0:23]
ADDR
74BCT29827
U18
U21
U16
U22
74AS823
U19
U24
U17
22V10
U20
U23
VA[0:8]
VB[0:8]
LBE[0:7]#
VBLANK#
SRAE#
RESET
MROW
SBLANK
ROWE#
COLE#
SBLANK
VBLANK#
SRAE#
RESET
SCAE#
BE[0:7]#
CLKC
LADSB#
+5V
RP1
RP2
RP5
FRAME BUFFER EXAMPLE
ADDR XCVRs/LATCHES SH.1
DATE :07-18-89
DRAWN BY: CL
FRAME BUFFER
REV.02

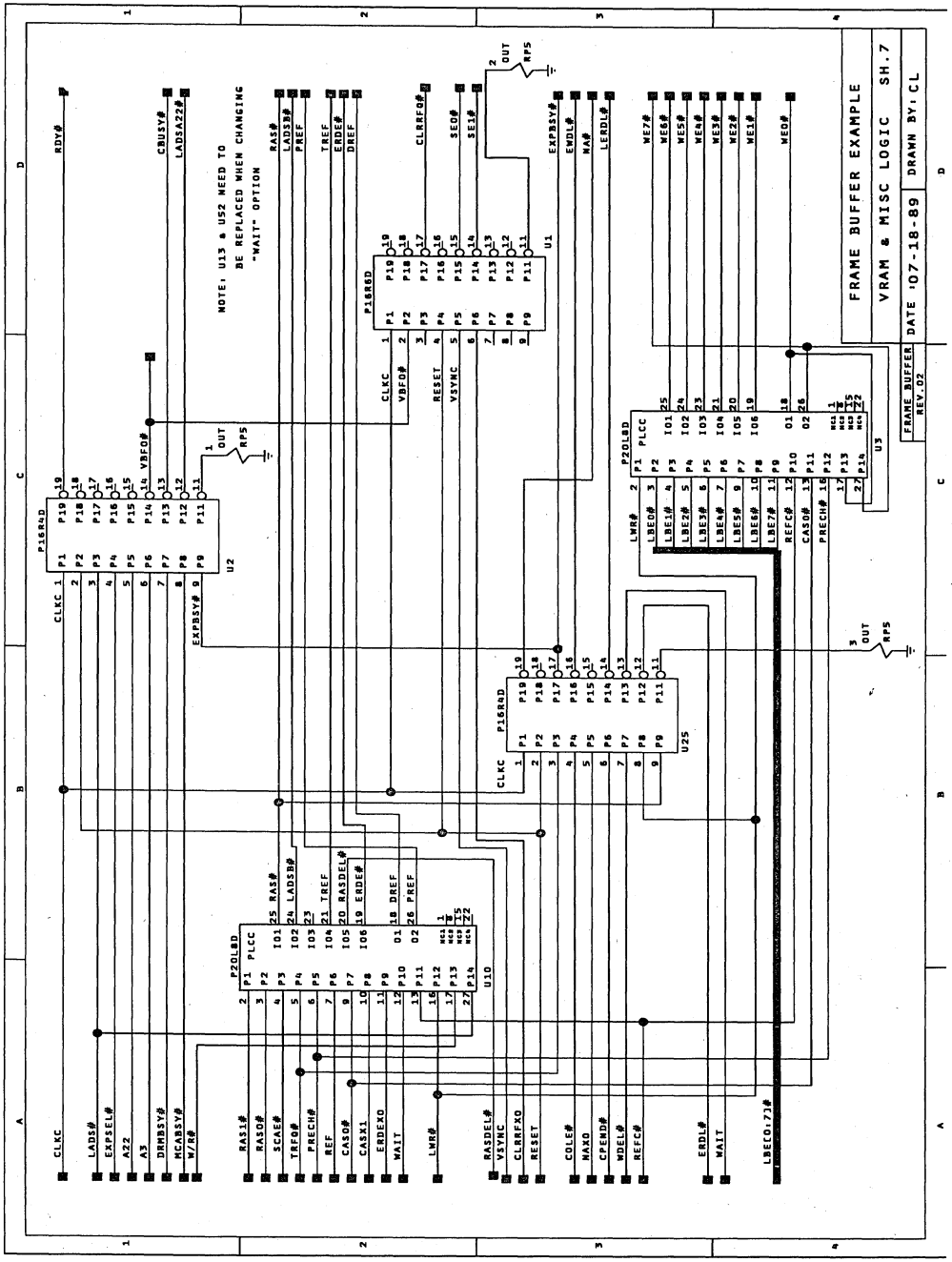

FRAME BUFFER EXAMPLE
VRAMs A-1
SH.2
DATE :07-18-89
DRAWN BY: JP
FRAME BUFFER
REV.02
SD[0:31]
SCO
DTOEO#
SEO#
CASO#
RASO#
VA[0:8]
MEO#
WE1#
WE2#
WE3#
TS24257

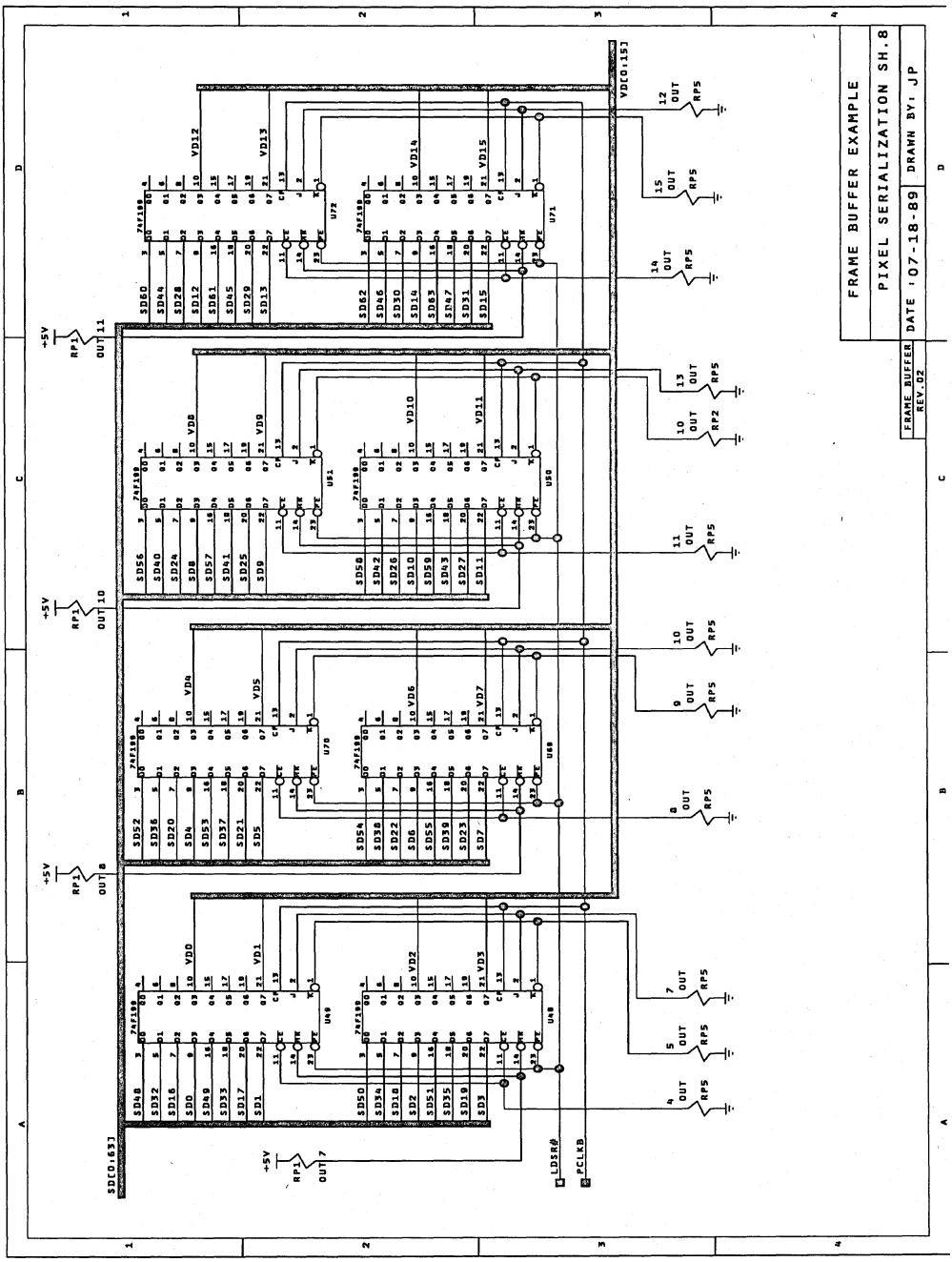

D[32:63]
SD[32:63]
SCO
DTOEO#
SEO#
CAS2#
CASO#
RASO#
WE4#
WE5#
WE6#
WE7#
VA[0:8]
T524257
U31
U36
U41
U45
U53
U58
U62
U66
VA0
VA1
VA2
VA3
VA4
VA5
VA6
VA7
VA8
SIO1
SIO2
SIO3
SIO4
W1/IO1
W2/IO2
W3/IO3
W4/IO4
NC1
NC2
NC3
SC
SD32 SD33 SD34 SD35 D32 D33 D34 D35
SD36 SD37 SD38 SD39 D36 D37 D38 D39
SD40 SD41 SD42 SD43 D40 D41 D42 D43
SD44 SD45 SD46 SD47 D44 D45 D46 D47
SD48 SD49 SD50 SD51 D48 D49 D50 D51
SD52 SD53 SD54 SD55 D52 D53 D54 D55
SD56 SD57 SD58 SD59 D56 D57 D58 D59
SD60 SD61 SD62 SD63 D60 D61 D52 D63
FRAME BUFFER EXAMPLE
VRAMs A-2
SH.3
FRAME BUFFER
REV.02
DATE : 07-18-89
DRAWN BY: JP

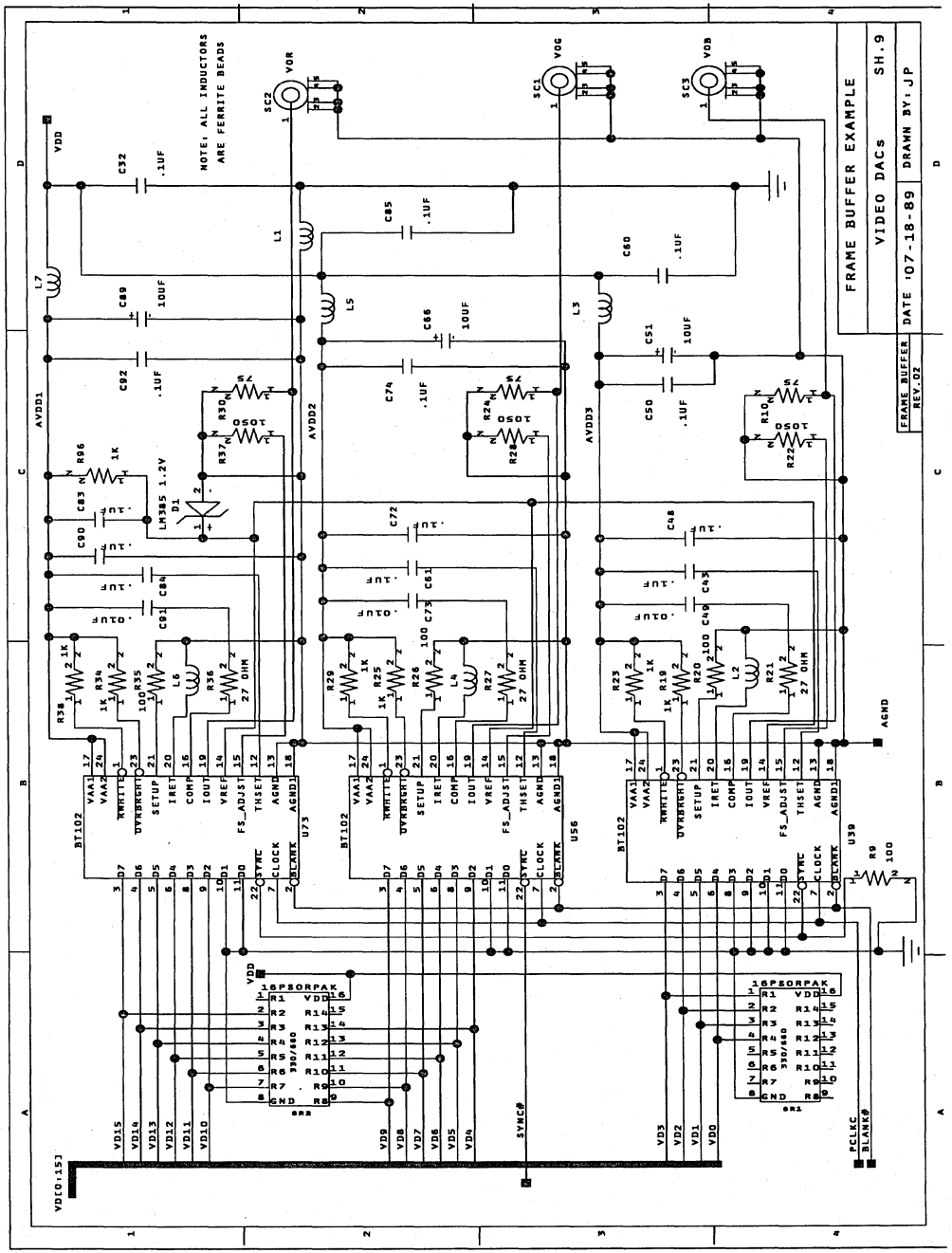

FRAME BUFFER EXAMPLE
VRAMs B-1
SH.4
DATE : 07-18-89
DRAWN BY: JP
FRAME BUFFER REV.02
SC1
DTOE1#
SE1#
CAS1#
RAS1#
SD[0:31]
VB[0:8]
ME0#
ME1#
ME2#
ME3#
TS24257

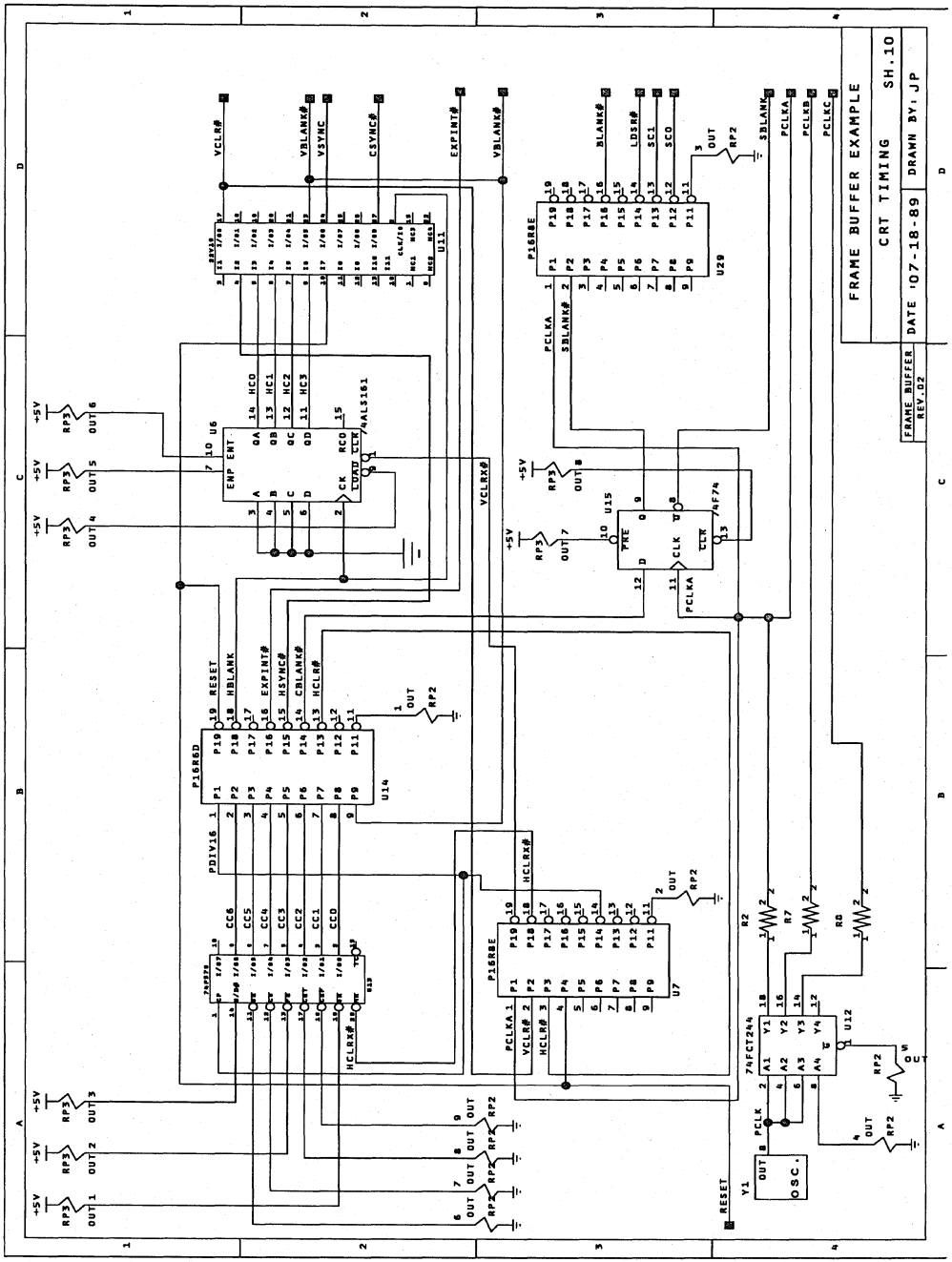

FRAME BUFFER EXAMPLE
VRAMs B-2
SH.5
DATE : 07-18-89
DRAWN BY: JP
FRAME BUFFER
REV.02

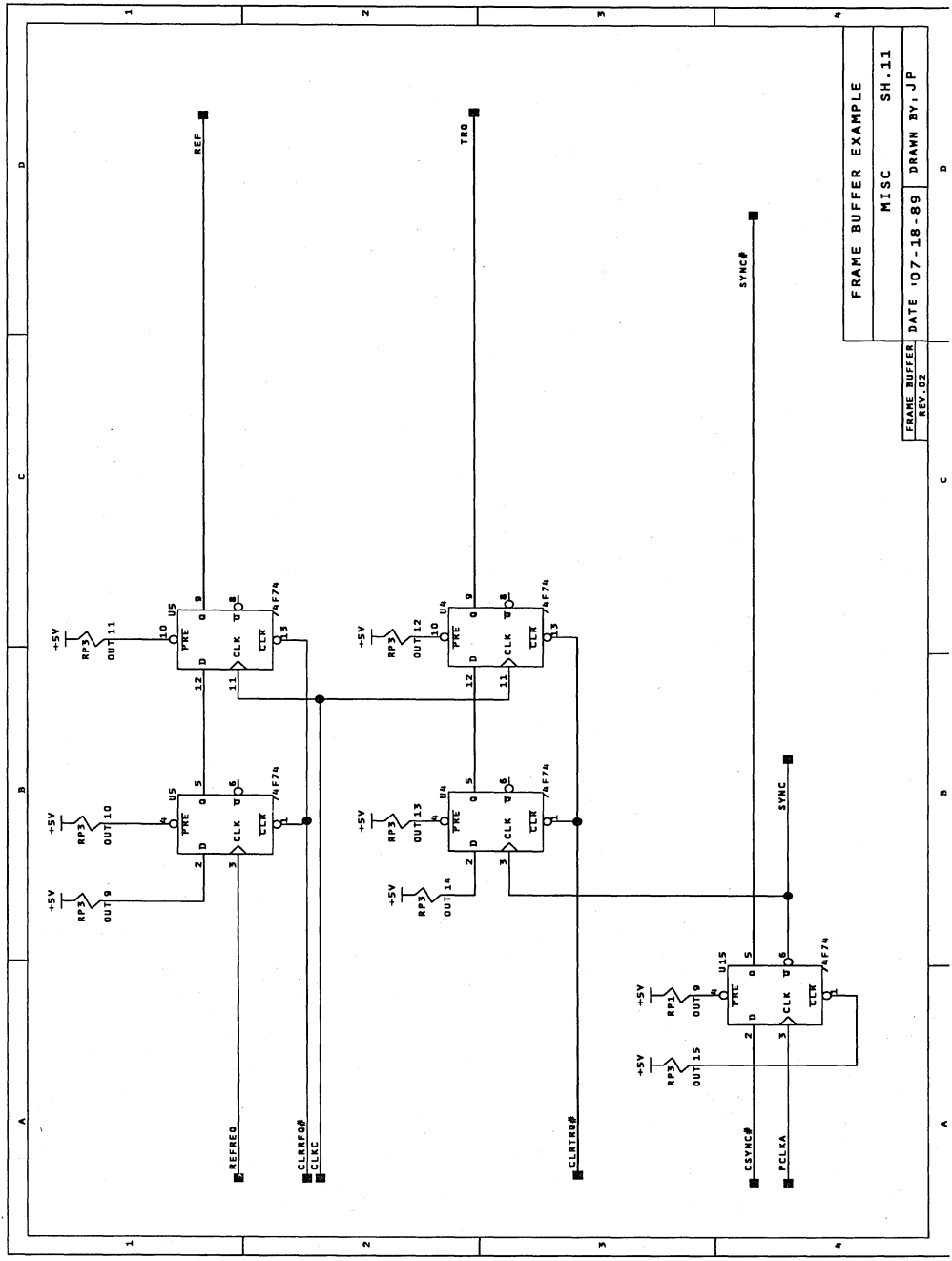

FRAME BUFFER EXAMPLE
VRAM CONTROL SH.6
DATE: 07-18-89 DRAWN BY: JP
FRAME BUFFER REV.02
P20R8D
P16R6D
U26
U28
U27
U34
RP1
JP1
JP2
CLKG
LADS#
MENE#
A22
A21
CBUSY#
TRFQ#
TREF
RESET
EXPSEL#
PREF
CPEND#
WAIT
RASDEL#
PRECH#
WDEL#
WDELXO
COLE#
SCAE#
REFC#
SRAE#
ROWE#
CASX1
CAS2#
CAS1#
CAS0#
ERDL#
RAS0#
RAS1#
CLRTRQ#
ERDEXO
EWDE#
DTOE1#
DTOE0#
CLRRFXO
LWR#
W/R#
RAS#
REF
TRQ
RDY#
LERDL#
EXPBSY#
DREF
LADSA22#
M/R#
HAXO
RDY#
RESET
WAIT
NEWE#
+5V

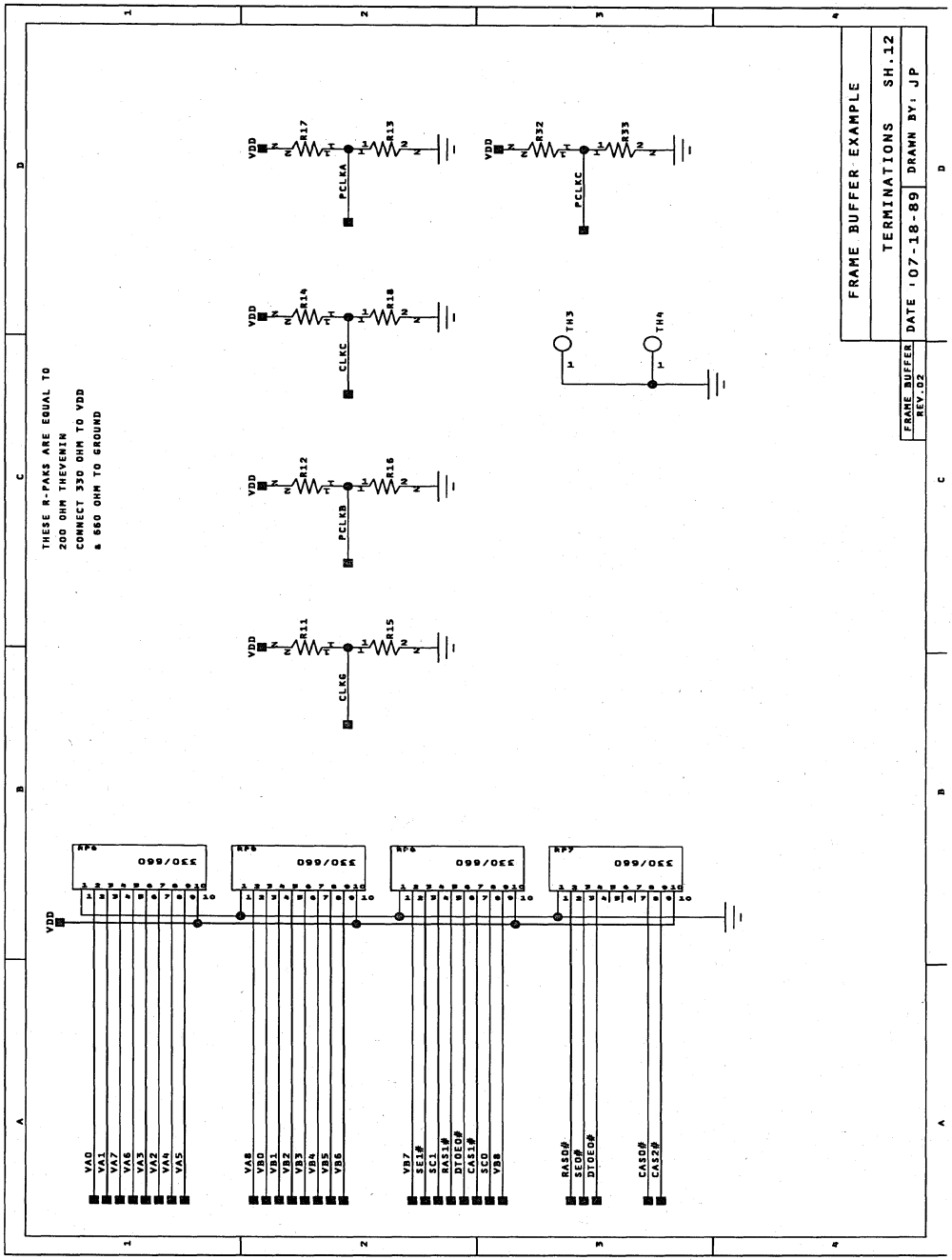

FRAME BUFFER EXAMPLE
VRAM & MISC LOGIC SH.7
DATE :07-18-89
DRAWN BY: CL
FRAME BUFFER REV.02
NOTE: U13 & U52 NEED TO BE REPLACED WHEN CHANGING "WAIT" OPTION
P16R4D
P16R8D
P20L8D
PLCC
RPS
OUT
CLKC
RDY#
LADS#
EXPSEL#
A22
A3
DRMBSY#
MCABSY#
W/R#
CBUSY#
LADSA22#
VBFO#
EXPBSY#
RAS#
LADSB#
PREF
TREF
ERDE#
DREF
CLRREF#
SEO#
SEI#
EXPBSY#
ERDL#
MA#
LERDL#
WE7#
WE6#
WE5#
WE4#
WE3#
WE2#
WE1#
WE0#
RESET
VSYNC
LWR#
REFC#
CAS0#
PRECH#
LBE7:0#
RAS1#
RAS0#
SCAS#
TRFO#
PRECH#
REF
CAS0#
CASX1
ERDEXO
WAIT
RASDEL#
CLRRFXO
COLE#
MAXO
CPEND#
WDEL#
U2
U1
U10
U25
U3

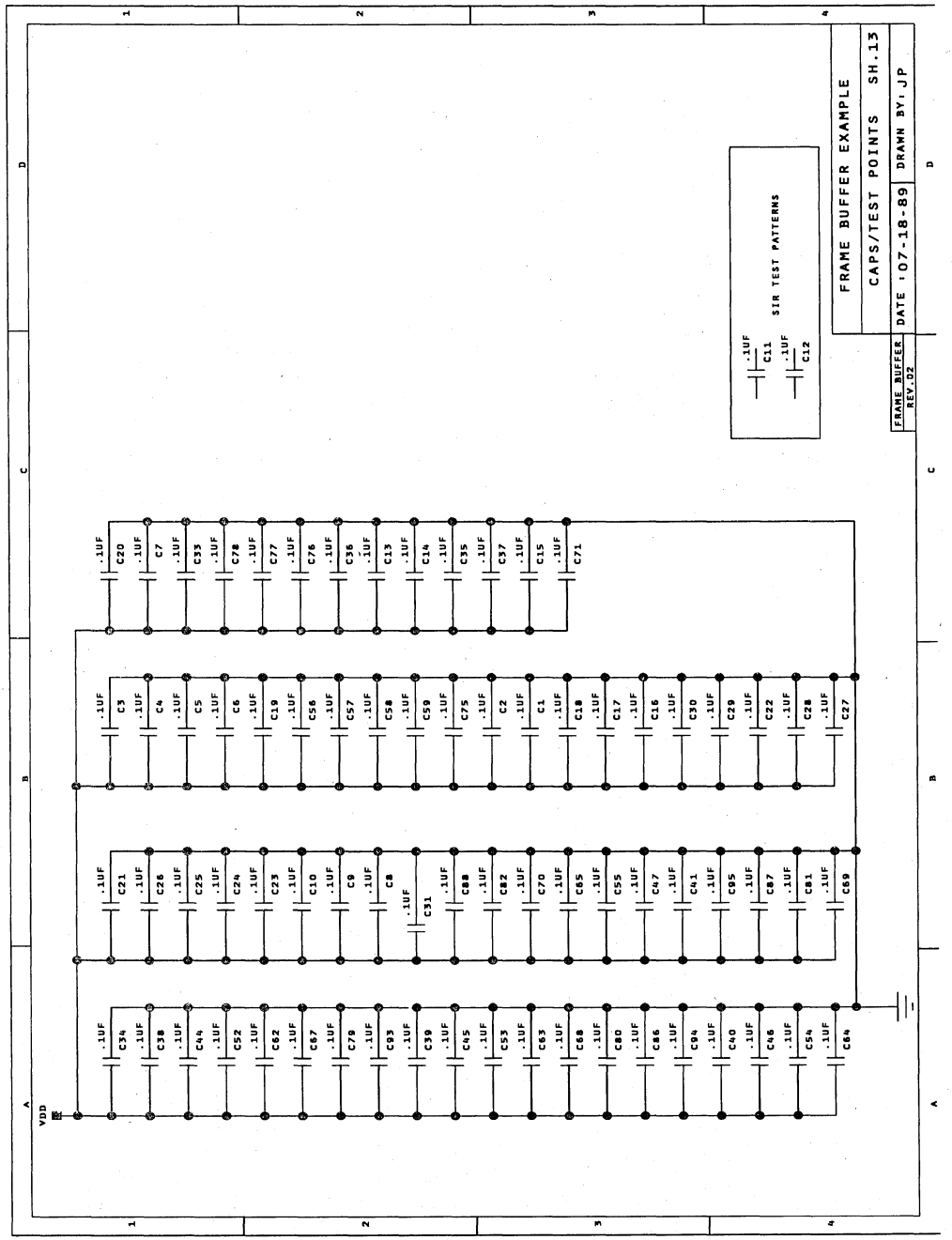

FRAME BUFFER EXAMPLE
PIXEL SERIALIZATION SH.8
DATE : 07-18-89
DRAWN BY: JP
FRAME BUFFER
REV.02
VDIO(15)
SDIO(63)
LDSR#
PCLKB
+5V
OUT


FRAME BUFFER EXAMPLE
VIDEO DACs SH.9
DATE '07-18-89 DRAWN BY:JP
FRAME BUFFER REV.02
NOTE: ALL INDUCTORS ARE FERRITE BEADS
VDD
AVDD1
AVDD2
AVDD3
AGND
VOR
VOG
VOB
LM385 1.2V
BT102
U73
U56
U39
16PSORPAK
VD[0:15]
SYNC#
BLANK#
PCLKC


FRAME BUFFER EXAMPLE
CRT TIMING
SH.10
DATE '07-18-89
DRAWN BY: JP
FRAME BUFFER REV.02
VCLR#
VBLANK#
VSYNC
CSYNC#
EXPINT#
BLANK#
LDSR#
SC1
SCO
SBLANK
PCLKA
PCLKB
PCLKC
RESET
HBLANK
HSYNC#
CBLANK#
HCLR#
HCLRX#
VCLRX#
PDIV16
CC6
CC5
CC4
CC3
CC2
CC1
CC0
PCLK
OSC.
74FCT244
74ALS161
74F74
P16R8D
P16R8E
+5V


FRAME BUFFER EXAMPLE
MISC
SH.11
DATE '07-18-89
DRAWN BY: JP
FRAME BUFFER
REV.02
REF
TRQ
SYNC#
SYNC
REFREQ
CLRREF#
CLKC
CLRTRQ#
CSYNC#
PCLKA
+5V
RP3
RP1
74F74

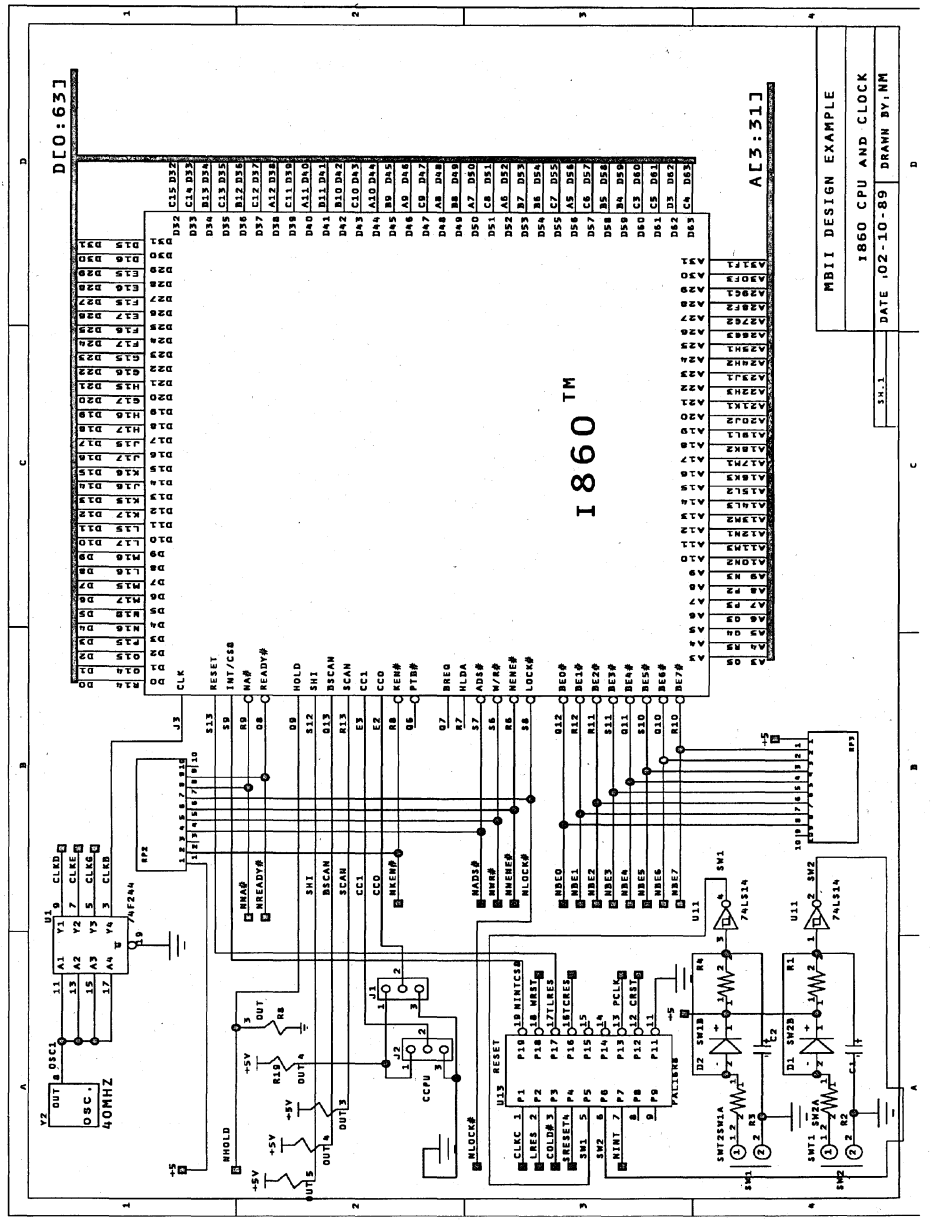

FRAME BUFFER EXAMPLE
TERMINATIONS SH.12
DATE : 07-18-89 DRAWN BY: JP
FRAME BUFFER REV.02
THESE R-PAKS ARE EQUAL TO
200 OHM THEVENIN
CONNECT 330 OHM TO VDD
& 660 OHM TO GROUND
VDD
R17
R13
PCLKA
R32
R33
PCLKC
R14
R18
CLKC
TH3
TH4
R12
R16
PCLKB
R11
R15
CLK6
RP6
RP7
330/660
VA0
VA1
VA7
VA6
VA3
VA2
VA4
VA5
VA8
VB0
VB1
VB2
VB3
VB4
VB5
VB6
VB7
SEI#
SCI
RAS1#
DTOE0#
CAS1#
SCO
VB8
RAS0#
SEO#
DTOE0#
CAS0#
CAS2#

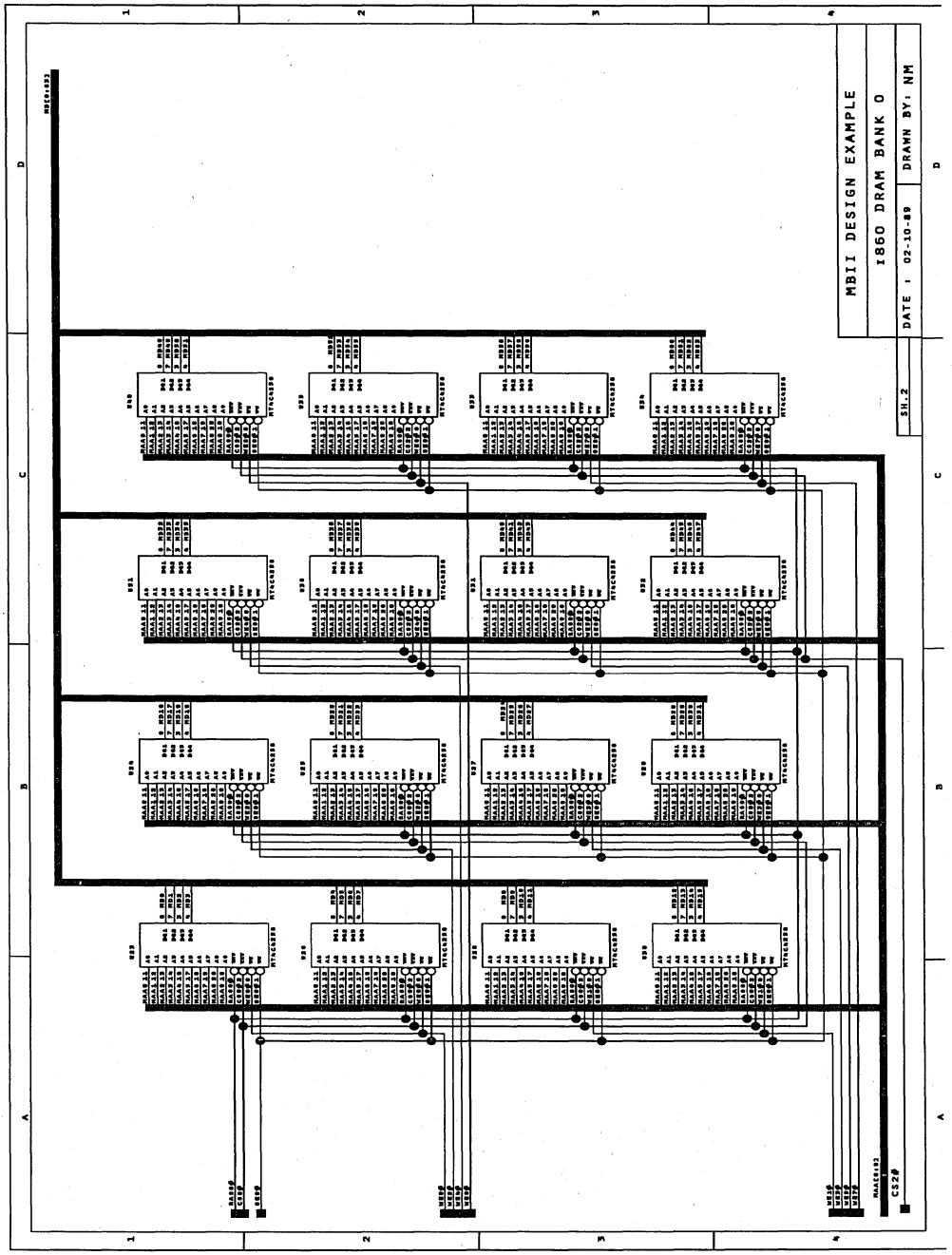

FRAME BUFFER EXAMPLE
CAPS/TEST POINTS SH.13
DATE : 07-18-89 DRAWN BY: JP
FRAME BUFFER REV.02
SIR TEST PATTERNS
.1UF C11
.1UF C12
VDD
.1UF C20 .1UF C7 .1UF C33 .1UF C78 .1UF C77 .1UF C76 .1UF C36 .1UF C13 .1UF C14 .1UF C35 .1UF C37 .1UF C15 .1UF C71
.1UF C3 .1UF C4 .1UF C5 .1UF C6 .1UF C19 .1UF C56 .1UF C57 .1UF C58 .1UF C59 .1UF C75 .1UF C2 .1UF C1 .1UF C18 .1UF C17 .1UF C16 .1UF C30 .1UF C29 .1UF C22 .1UF C28 .1UF C27
.1UF C21 .1UF C26 .1UF C25 .1UF C24 .1UF C23 .1UF C10 .1UF C9 .1UF C8 .1UF C31 .1UF C88 .1UF C82 .1UF C70 .1UF C65 .1UF C55 .1UF C47 .1UF C41 .1UF C95 .1UF C87 .1UF C81 .1UF C69
.1UF C34 .1UF C38 .1UF C44 .1UF C52 .1UF C62 .1UF C67 .1UF C79 .1UF C93 .1UF C39 .1UF C45 .1UF C53 .1UF C63 .1UF C68 .1UF C80 .1UF C86 .1UF C94 .1UF C40 .1UF C46 .1UF C54 .1UF C64

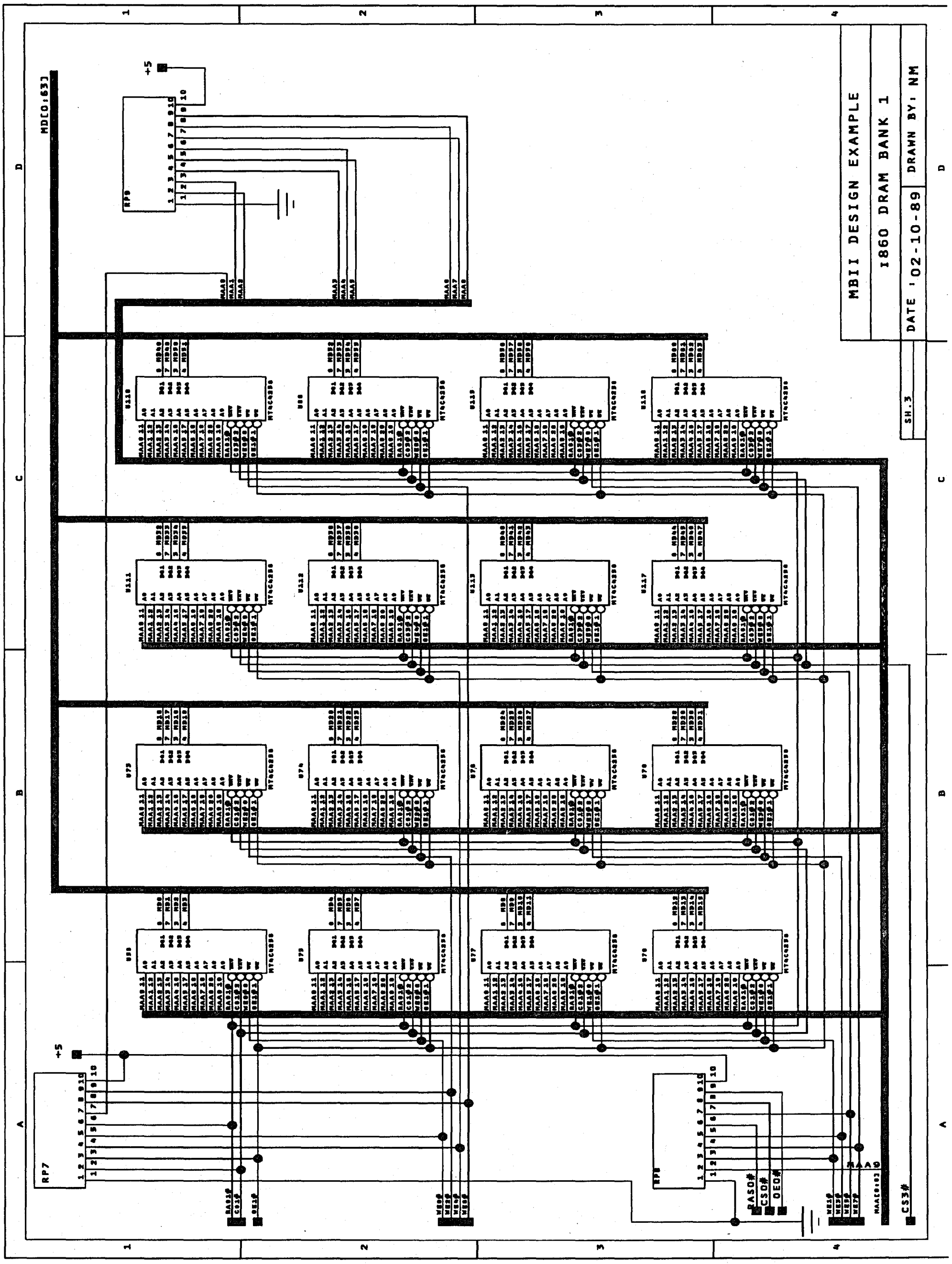
MULTIBUS® II Schematics* B

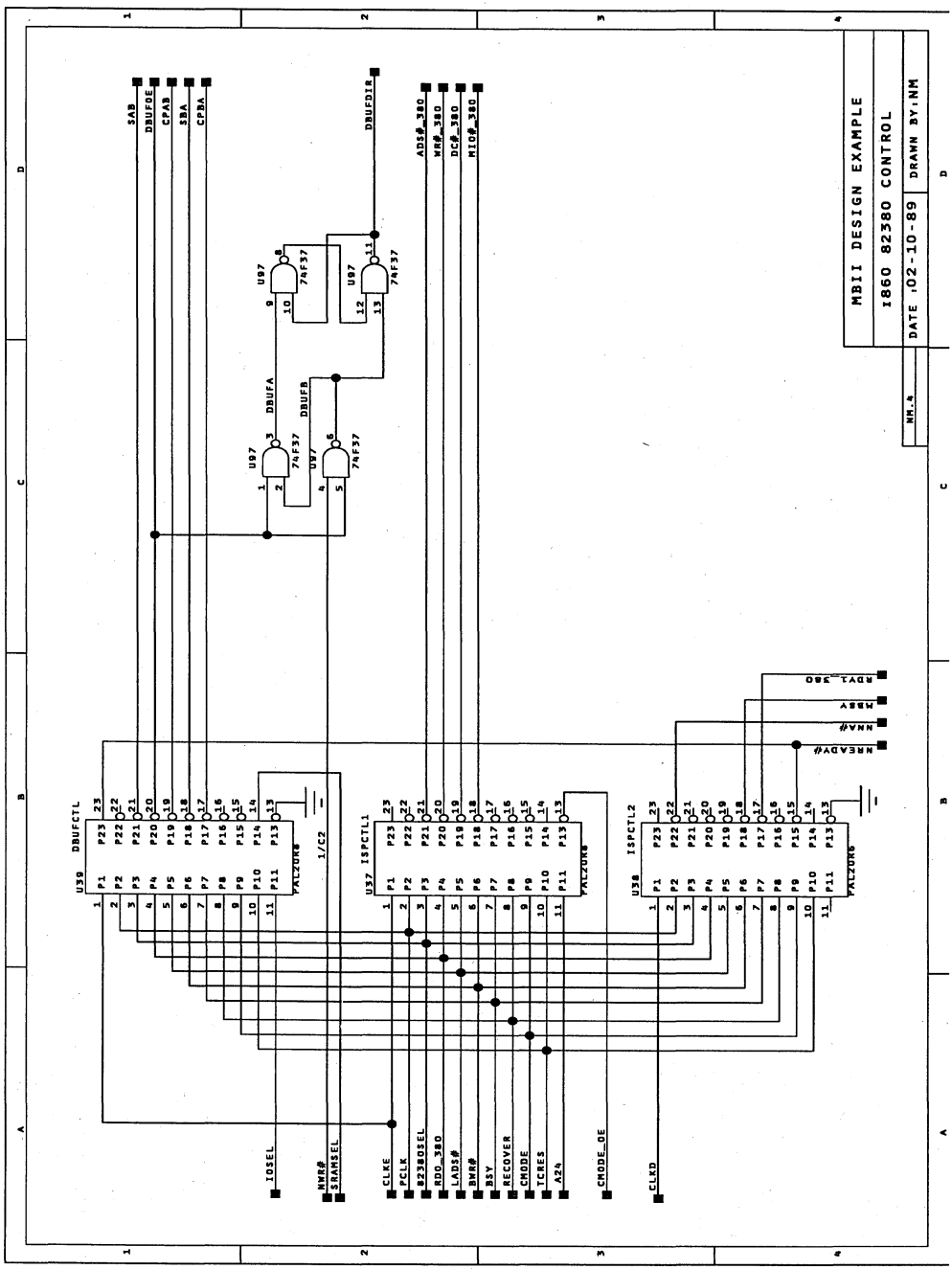
*This design has not been tested.

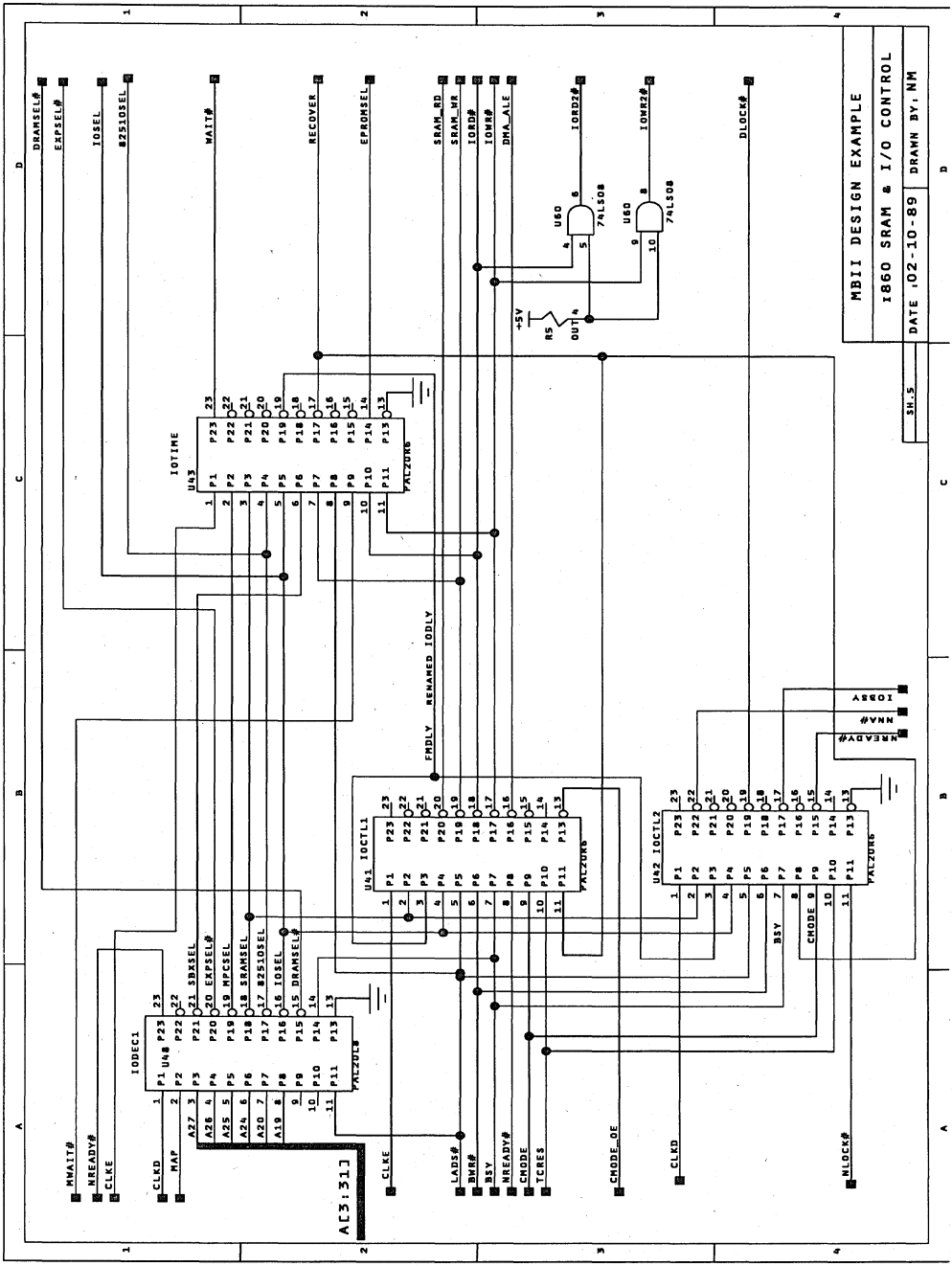
APPENDIX B
MULTIBUS® II
SCHEMATICS

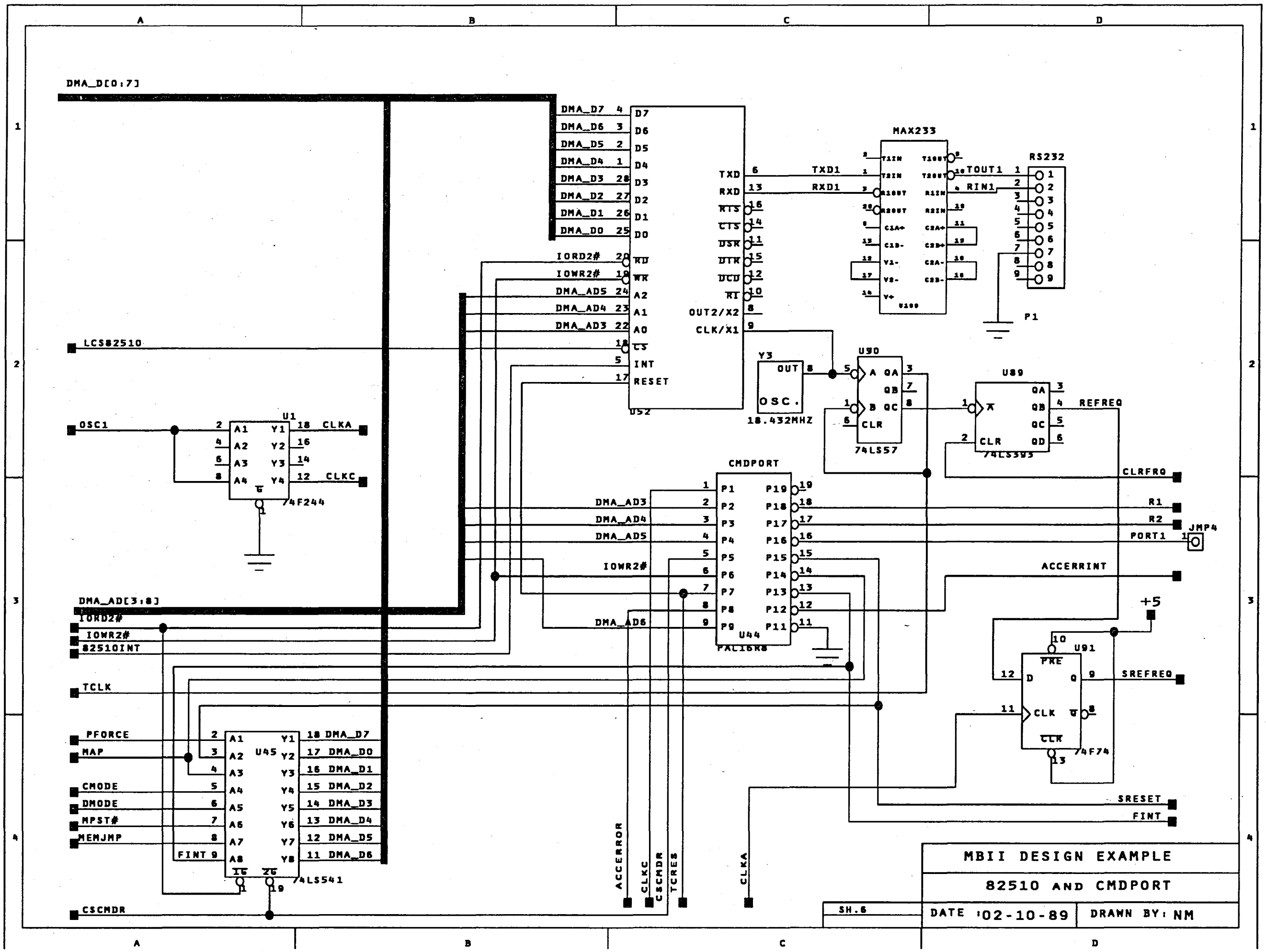


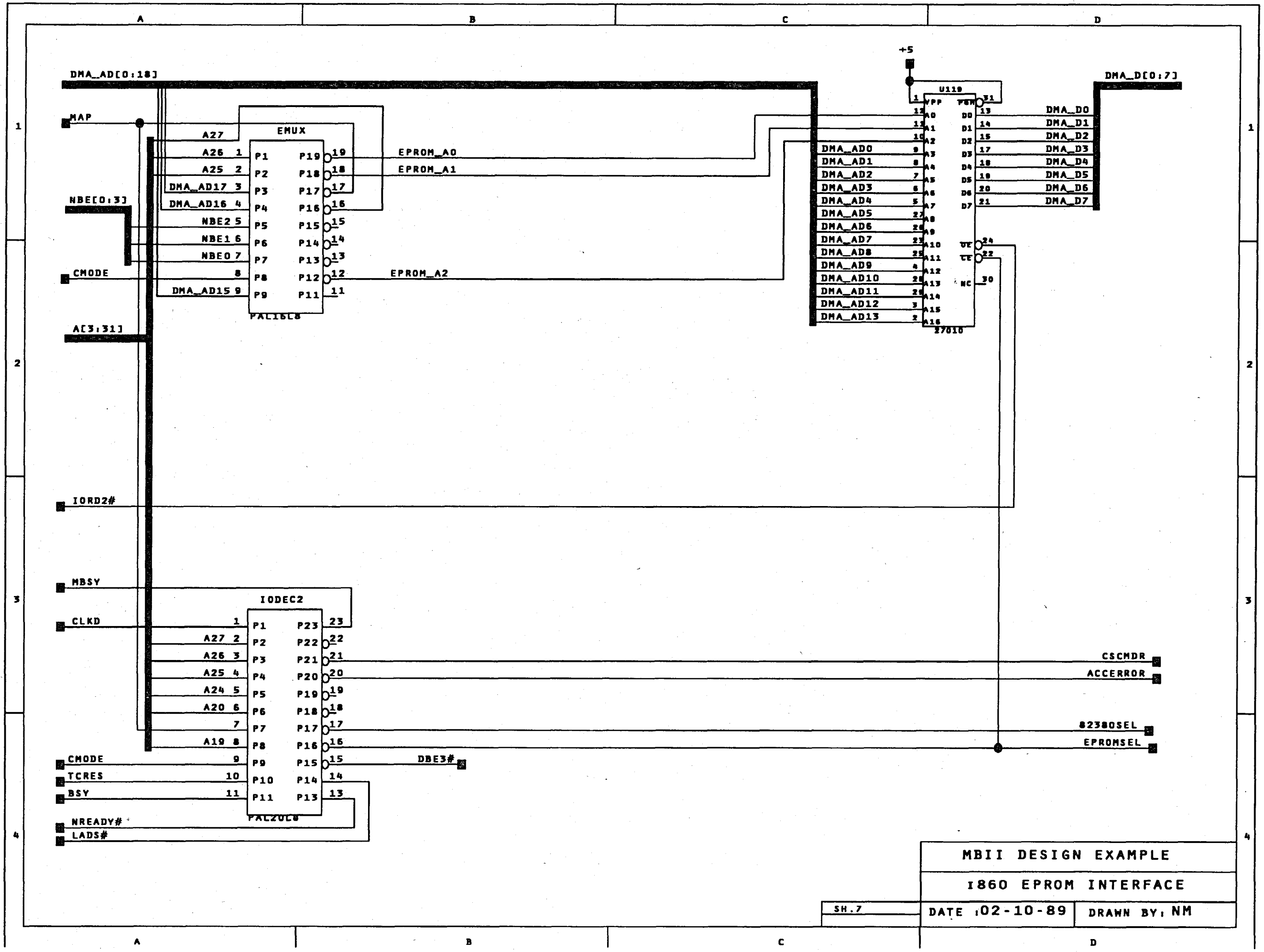

MBII DESIGN EXAMPLE
I860 DRAM BANK 0
SH.2
DATE : 02-10-89
DRAWN BY: NM


MBII DESIGN EXAMPLE
I860 DRAM BANK 1
DATE '02-10-89
DRAWN BY: NM
SH.3
MD[0:63]
+5
RP9
RP7
RP8
RAS0#
CS0#
OE0#
CS3#
MAA9


MBII DESIGN EXAMPLE
1860 82380 CONTROL
DATE :02-10-89
DRAWN BY:NM
NM.A
SAB
DBUFDE
CPAB
SBA
CPBA
DBUFDIR
ADS#_380
MRR#_380
DC#_380
M10#_380
U97
74F37
DBUFA
DBUFB
U39
DBUFCTL
PAL20R6
1/C2
U37
ISPCTL1
U38
ISPCTL2
NRDY1_380
MBSY
NMA#
NMEADY#
IOSEL
MRR#
SRAMSEL
CLKE
PCLK
82389SEL
RDO_380
LADS#
BHR#
BSY
RECOVER
CMODE
TCRES
A24
CMODE_OE
CLKD


MBII DESIGN EXAMPLE
I860 SRAM & I/O CONTROL
DATE :02-10-89
DRAWN BY: NM
SH.5
DRAMSEL#
EXPSEL#
IOSEL
82510SEL
MWAIT#
RECOVER
EPROMSEL
SRAM_BD
SRAM_WR
IORD#
IOWR#
DMA_ALE
IORD2#
IOWR2#
DLOCK#
MWAIT#
MREADY#
CLKE
IODEC1
U48
CLKD
MAP
A27
A26
A25
A24
A20
A19
SBXSEL
EXPSEL#
MPCSEL
SRAMSEL
82510SEL
IOSEL
DRAMSEL#
A[3:31]
IOTIME
U43
74LS08
U60
+5V
R5
OUT
FRDLY
RENAMED IODLY
U41 IOCTL1
U42 IOCTL2
LADS#
BWR#
BSY
MREADY#
CMODE
TCRES
CMODE_OE
CLKD
NLOCK#
IODSEV
NNA#
MREADY#

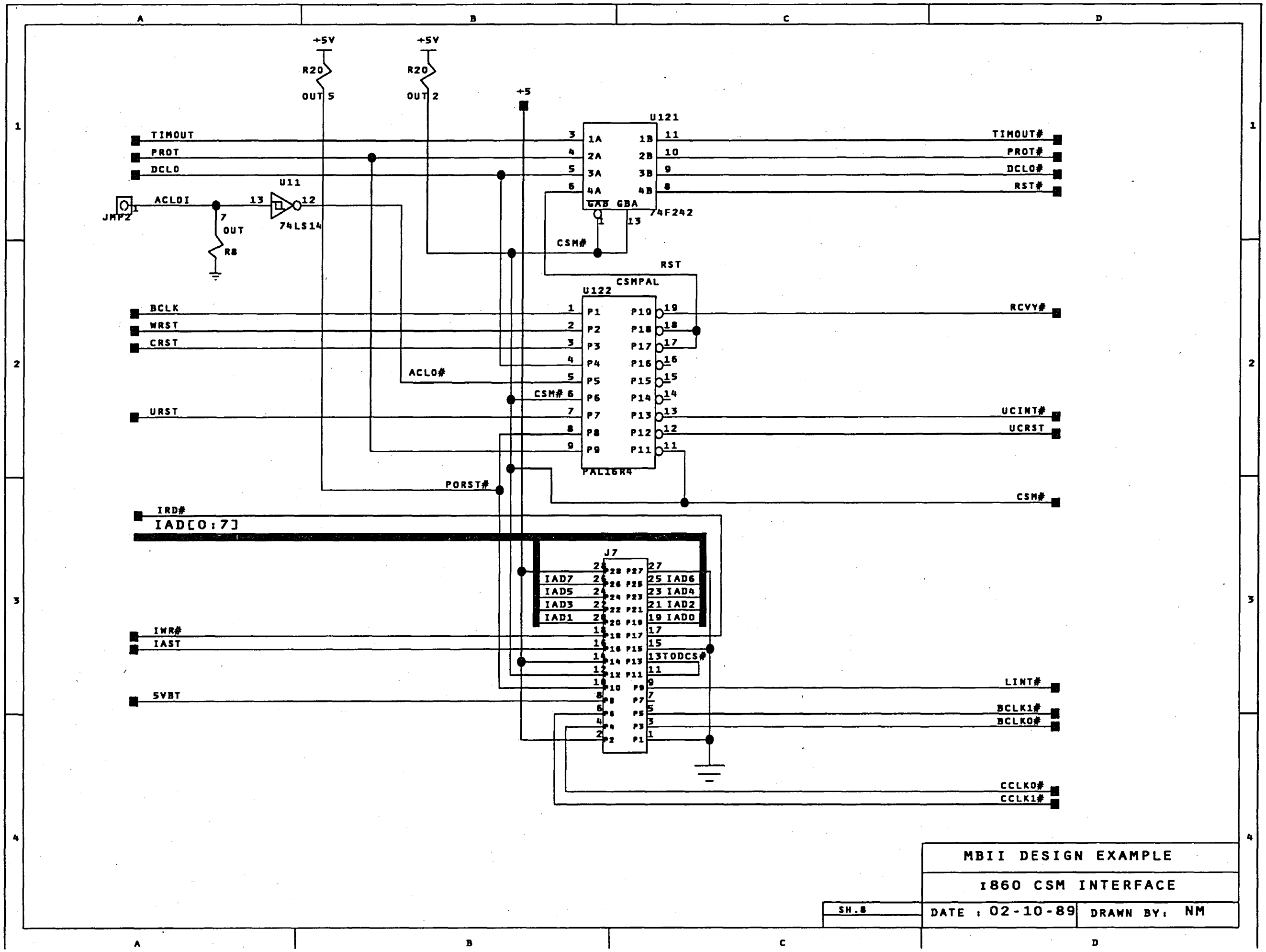

DMA_D[0:7]
DMA_D7 4 D7
DMA_D6 3 D6
DMA_D5 2 D5
DMA_D4 1 D4
DMA_D3 28 D3
DMA_D2 27 D2
DMA_D1 26 D1
DMA_D0 25 D0
IORD2# 20 RD
IOWR2# 19 WR
DMA_AD5 24 A2
DMA_AD4 23 A1
DMA_AD3 22 A0
18 CS
5 INT
17 RESET
TXD 6
RXD 13
RTS 16
CTS 14
DSR 11
DTR 15
DCD 12
RI 10
OUT2/X2 8
CLK/X1 9
U52
TXD1
RXD1
MAX233
TOUT1
RIN1
RS232
P1
LCS82510
OSC1
U1
A1 Y1 18 CLKA
A2 Y2 16
A3 Y3 14
A4 Y4 12 CLKC
74F244
Y3
OUT 8
OSC.
18.432MHZ
U90
A QA
QB
B QC
CLR
74LS57
U89
QA
A QB
QC
CLR QD
74LS393
REFREQ
CLRFRQ
CMDPORT
P1 P19
DMA_AD3 P2 P18
DMA_AD4 P3 P17
DMA_AD5 P4 P16
P5 P15
IOWR2# P6 P14
P7 P13
P8 P12
DMA_AD6 P9 P11
U44
PAL16R8
R1
R2
PORT1
JMP4
ACCERRINT
+5
U91
PRE
D Q
CLK Q
CLR
74F74
SREFREQ
SRESET
FINT
DMA_AD[3:8]
IORD2#
IOWR2#
82510INT
TCLK
PFORCE
MAP
CMODE
DMODE
MPST#
MEMJMP
FINT
A1 Y1 18 DMA_D7
A2 U45 Y2 17 DMA_D0
A3 Y3 16 DMA_D1
A4 Y4 15 DMA_D2
A5 Y5 14 DMA_D3
A6 Y6 13 DMA_D4
A7 Y7 12 DMA_D5
A8 Y8 11 DMA_D6
74LS541
CSCMDR
ACCERROR
CLKC
CSCMDR
TCRES
CLKA
MBII DESIGN EXAMPLE
82510 AND CMDPORT
SH.6
DATE : 02-10-89
DRAWN BY: NM

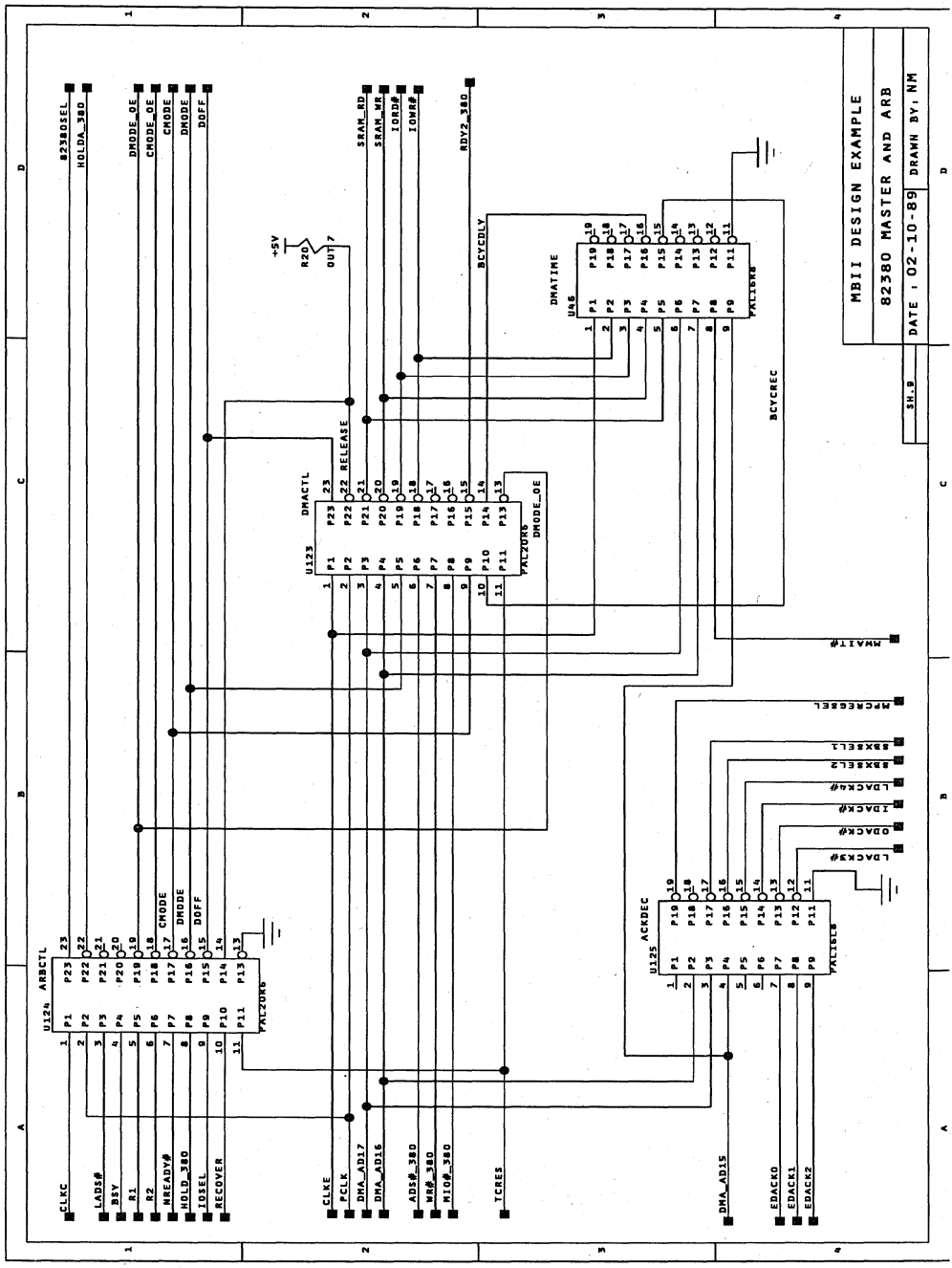


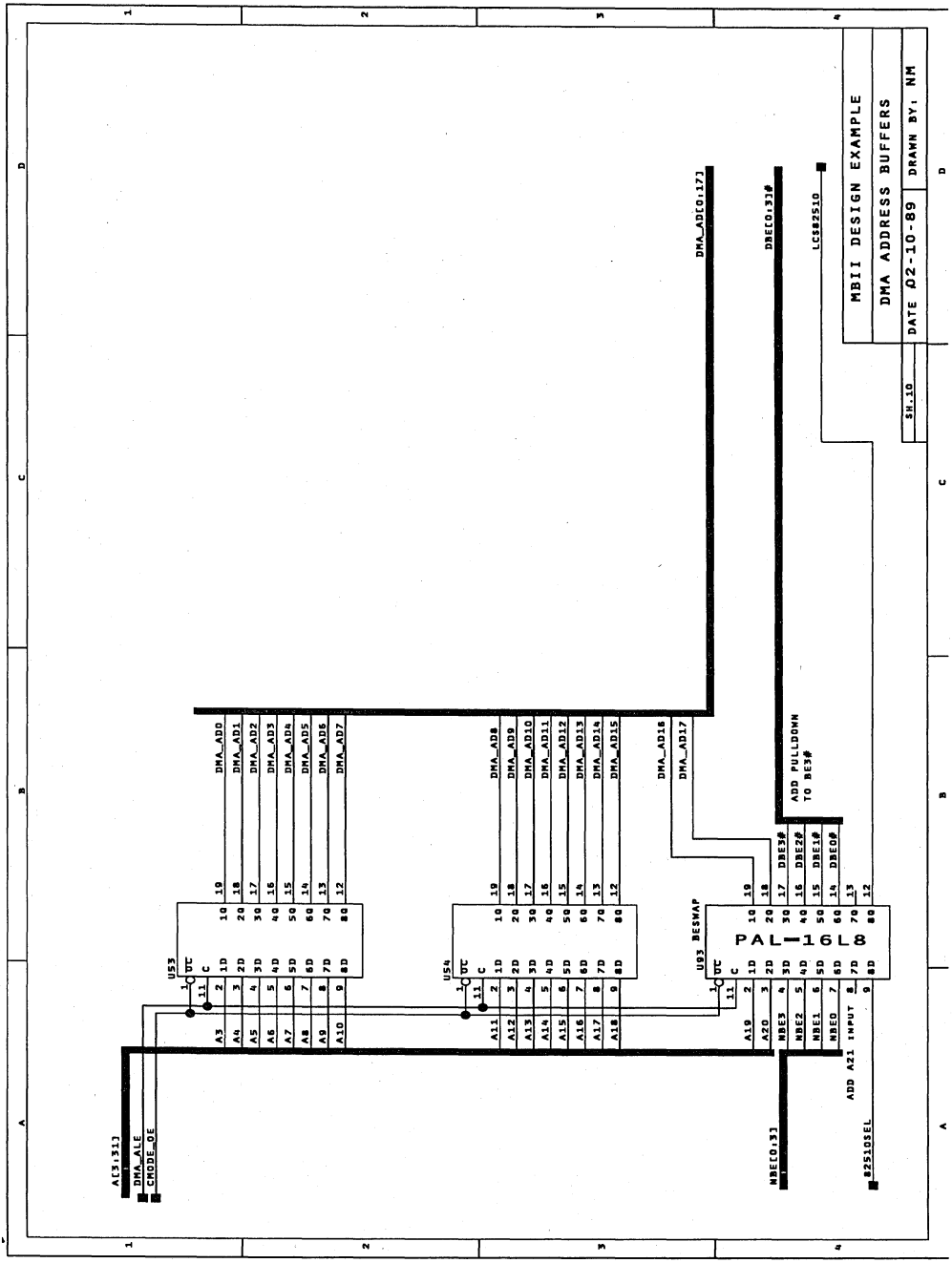
MBII DESIGN EXAMPLE

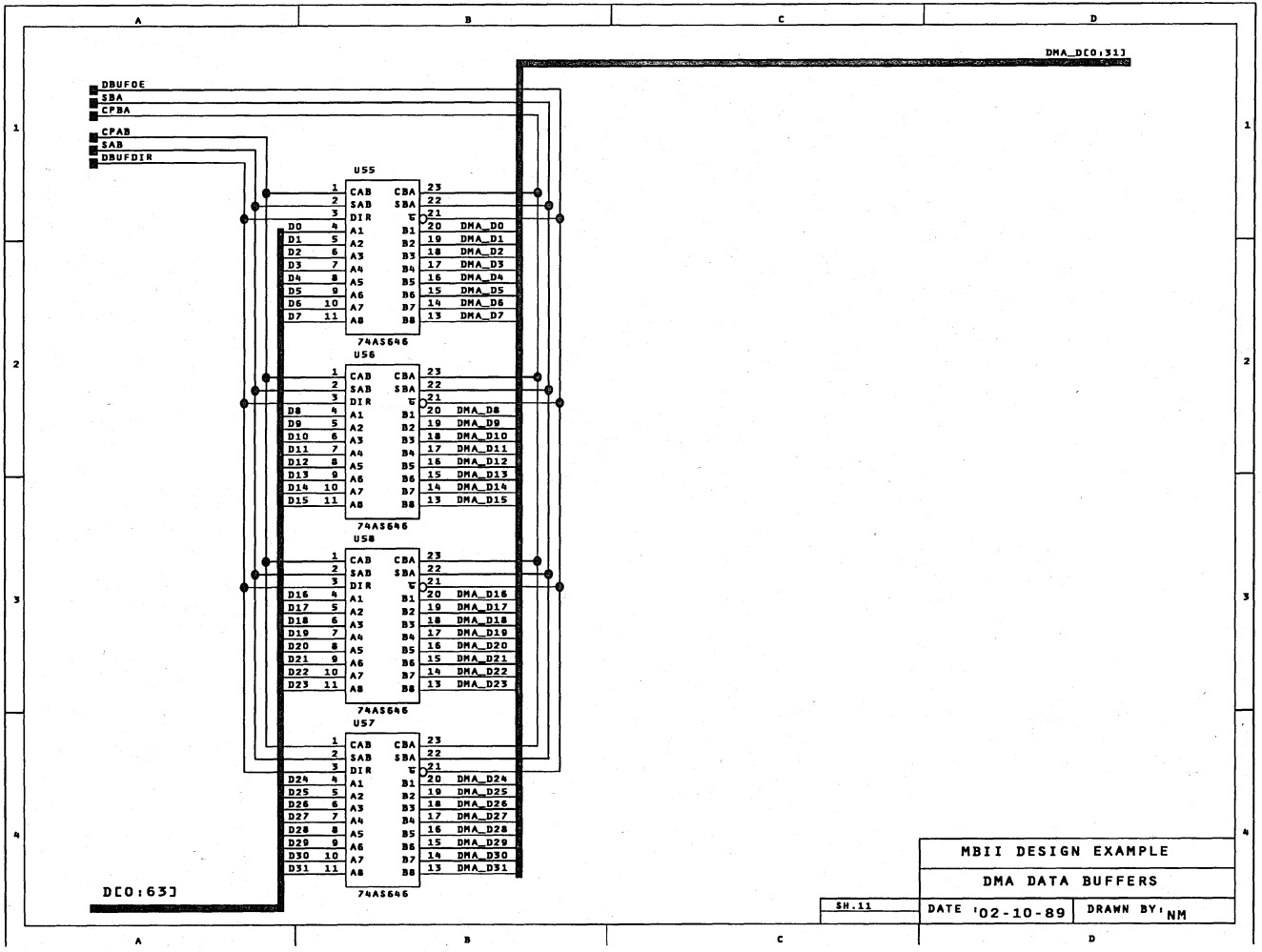
I860 EPROM INTERFACE

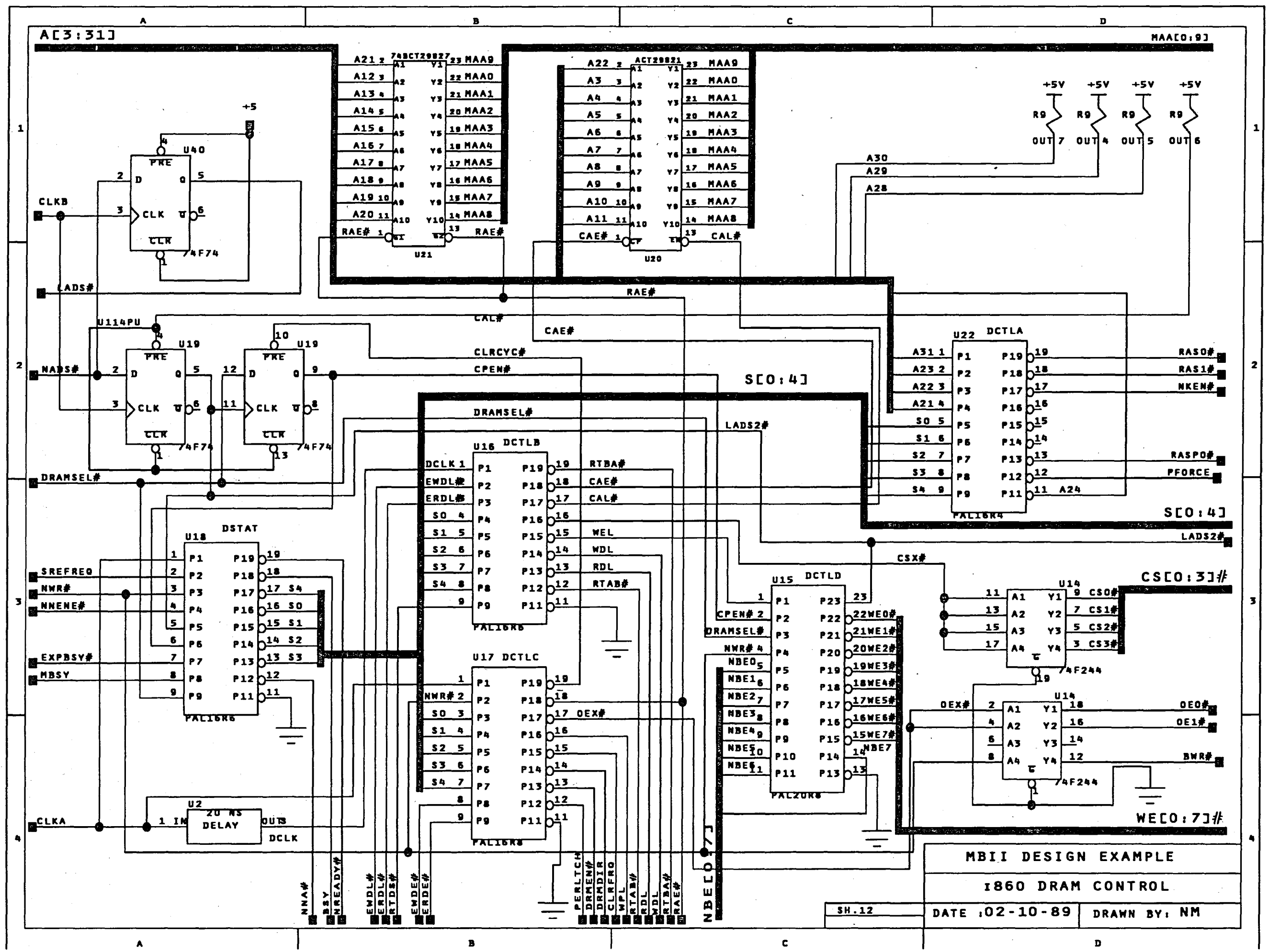
DATE :02-10-89 DRAWN BY: NM

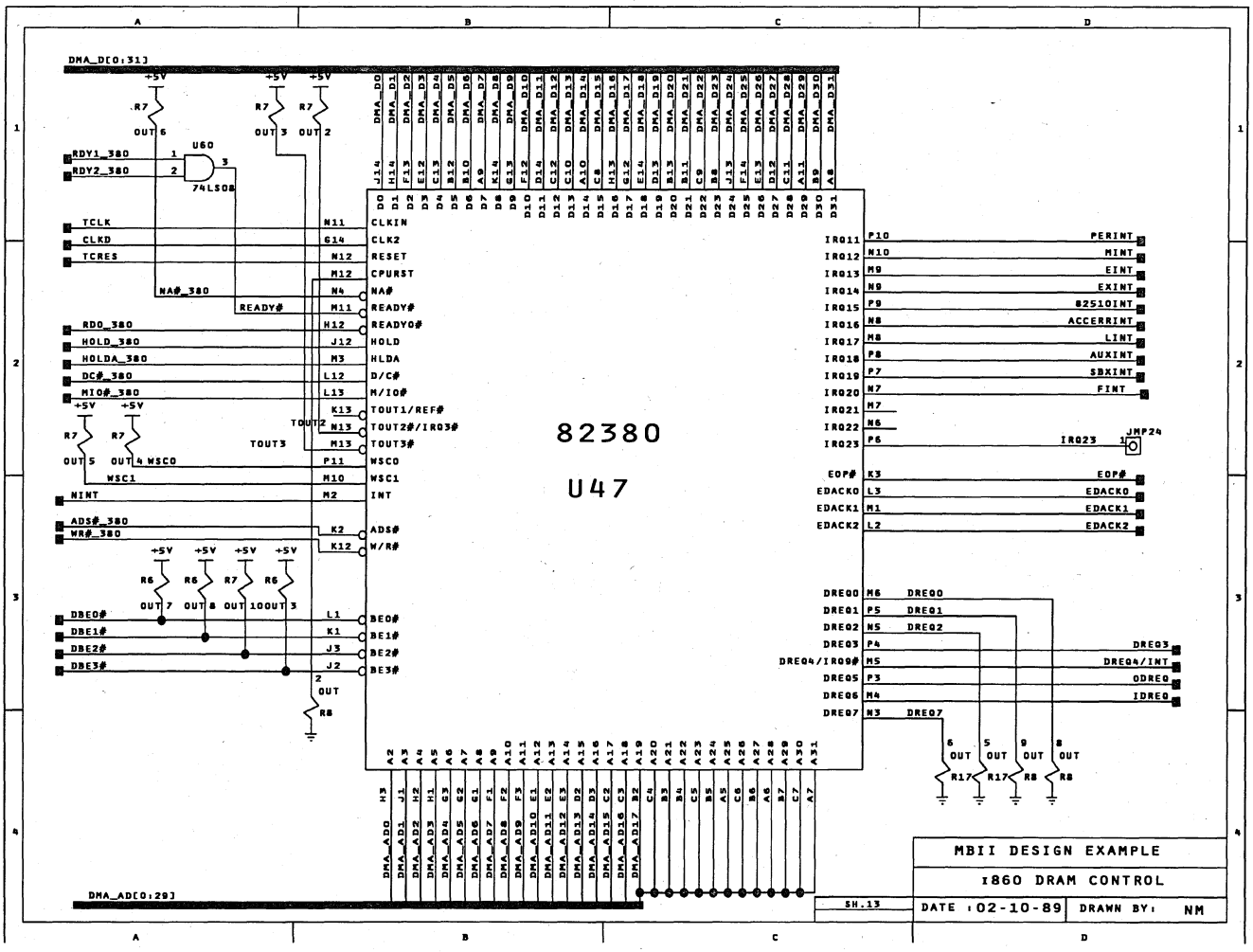

+5V
R20
OUT 5
OUT 2
+5
U121
TIMOUT
PROT
DCLO
ACLDI
JMP2
U11
74LS14
74F242
CSM#
RST
CSMPAL
U122
BCLK
WRST
CRST
ACLO#
URST
PORST#
PAL16R4
IRD#
IAD[0:7]
J7
IAD7
IAD5
IAD3
IAD1
IAD6
IAD4
IAD2
IAD0
IWR#
IAST
TODCS#
5VBT
TIMOUT#
PROT#
DCLO#
RST#
RCVY#
UCINT#
UCRST
CSM#
LINT#
BCLK1#
BCLK0#
CCLK0#
CCLK1#
MBII DESIGN EXAMPLE
I860 CSM INTERFACE
SH.8
DATE : 02-10-89
DRAWN BY: NM

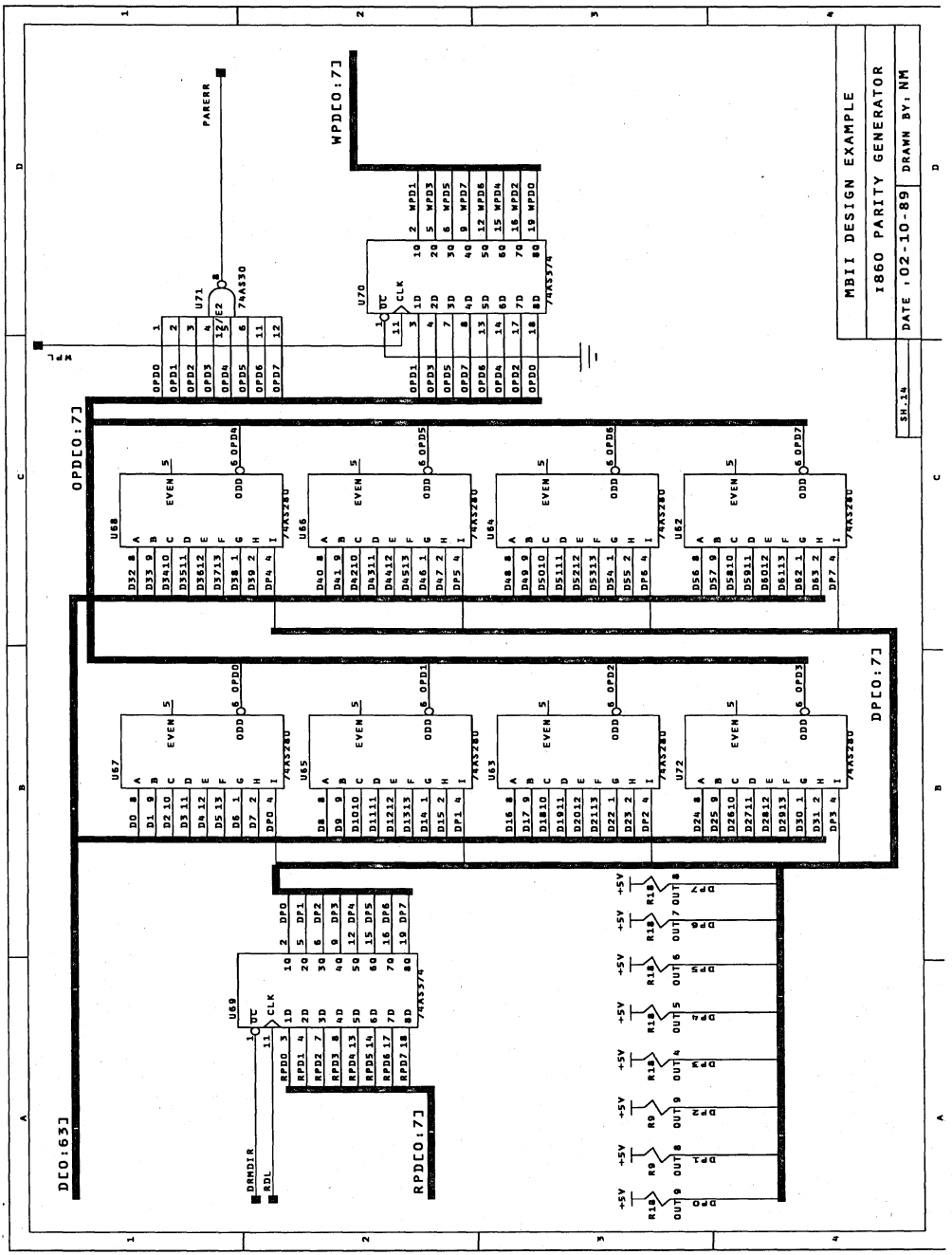

MBII DESIGN EXAMPLE
82380 MASTER AND ARB
SH.0
DATE : 02-10-89
DRAWN BY: NM
ARBCTL
DMACTL
ACKDEC
DMATIME
82380SEL
HOLDA_380
DMODE_OE
CMODE_OE
CMODE
DMODE
DOFF
SRAM_RD
SRAM_WR
IORD#
IOWR#
RDY2_380
BCYCDLY
BCYCREC
MWAIT#
MPCRESSEL
SBXSEL1
SBXSEL2
LDACK4#
IDACK#
ODACK#
LDACK3#
CLKC
LADS#
BSY
A1
R2
NREADY#
HOLD_380
IOSEL
RECOVER
CLKE
PCLK
DMA_AD17
DMA_AD16
ADS#_380
WR#_380
M10#_380
TCRES
DMA_AD15
EDACK0
EDACK1
EDACK2
+5V
R20
OUT

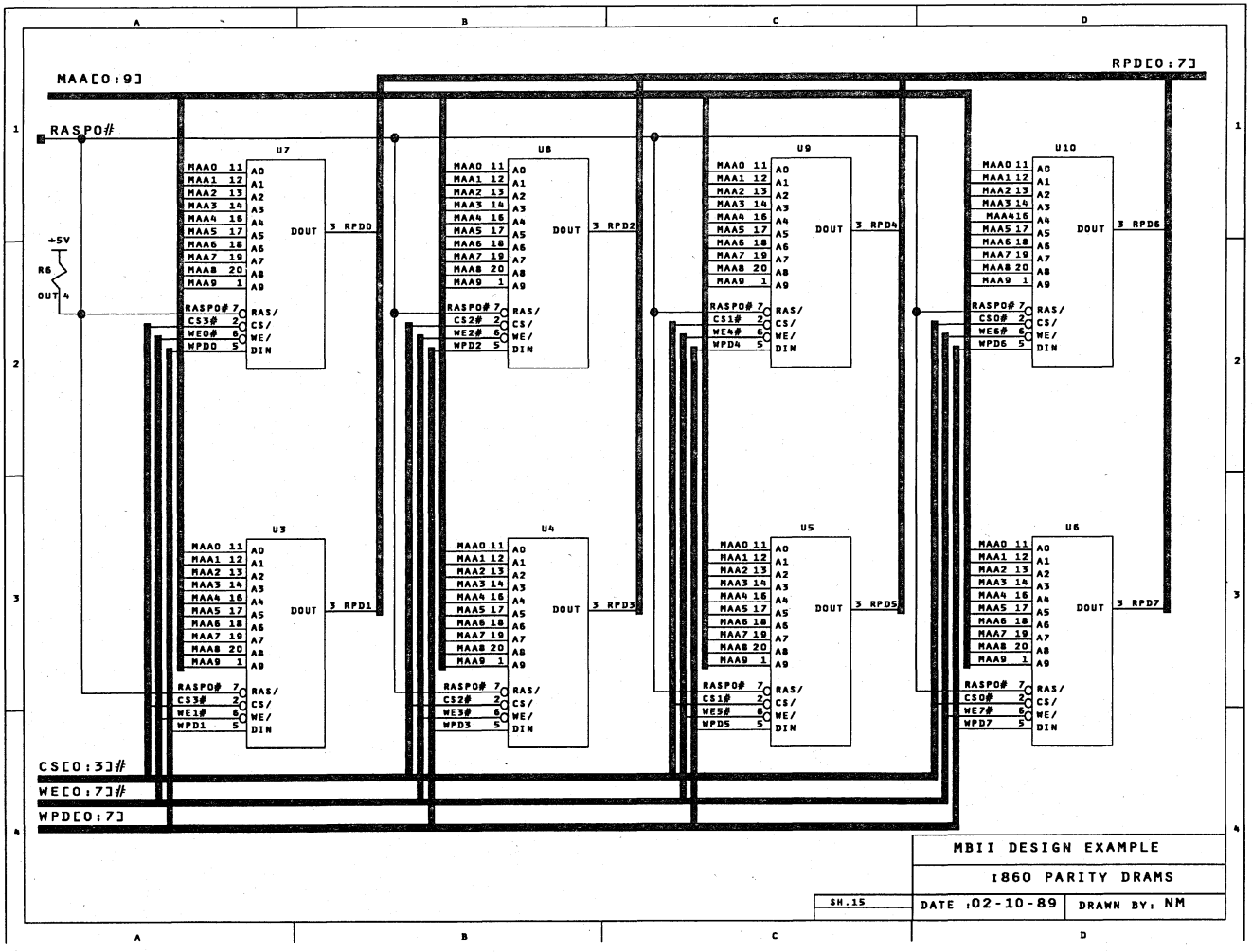

MBII DESIGN EXAMPLE
DMA ADDRESS BUFFERS
DATE 02-10-89
DRAWN BY: NM
SH.10
DMA_ADE0:17]
DBE0:3]#
LCS#2510
U53
U54
U83 BESWAP
PAL-16L8
ADD PULLDOWN TO BE3#
DMA_ALE
CMODE_OE
NBE0:3]
#2510SEL
ADD A21 INPUT

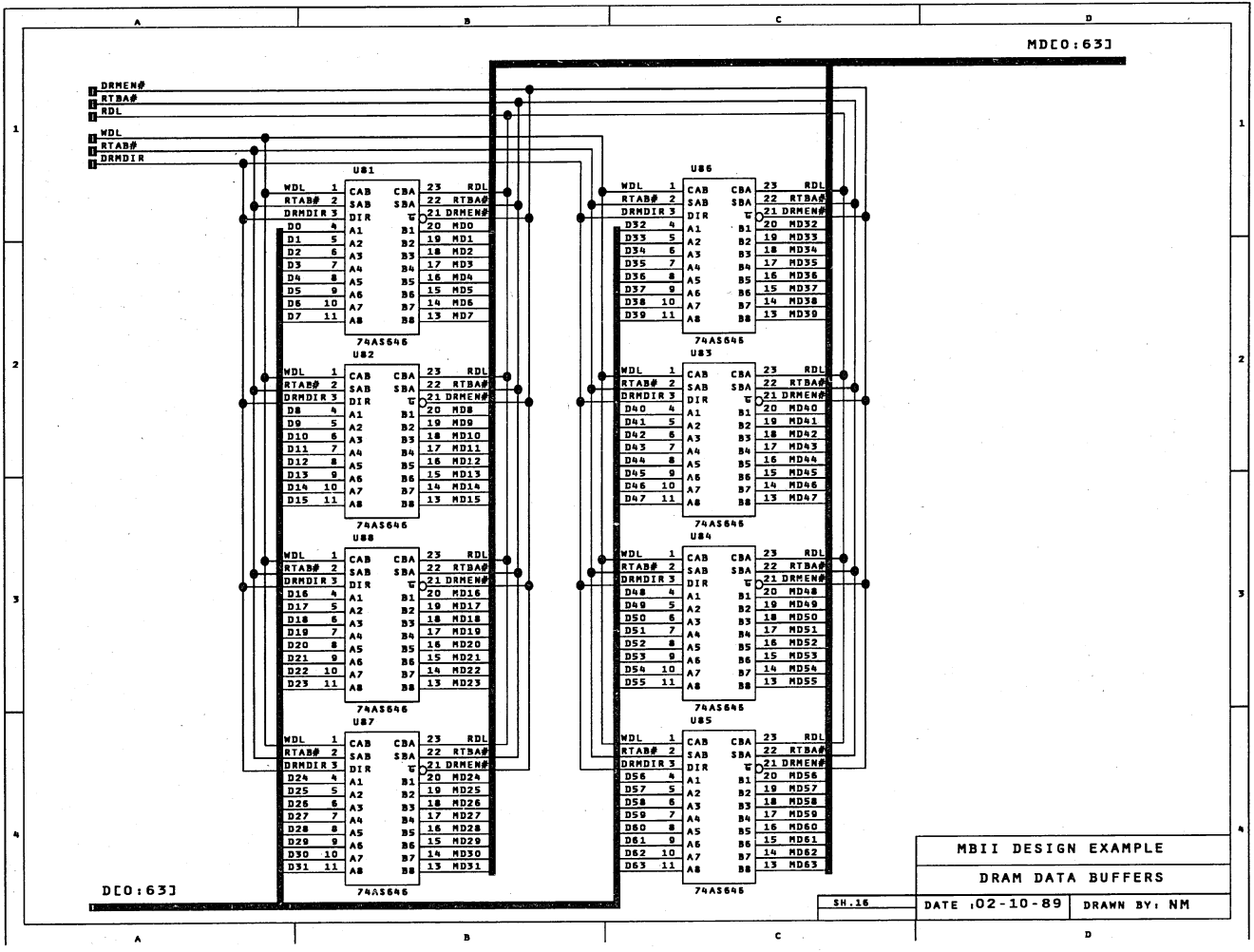

DMA_D[0:31]
DBUFOE
SBA
CPBA
CPAB
SAB
DBUFDIR
U55
U56
U58
U57
74AS646
CAB
SAB
DIR
CBA
SBA
A1
A2
A3
A4
A5
A6
A7
A8
B1
B2
B3
B4
B5
B6
B7
B8
D0
D1
D2
D3
D4
D5
D6
D7
DMA_D0
DMA_D1
DMA_D2
DMA_D3
DMA_D4
DMA_D5
DMA_D6
DMA_D7
D8
D9
D10
D11
D12
D13
D14
D15
DMA_D8
DMA_D9
DMA_D10
DMA_D11
DMA_D12
DMA_D13
DMA_D14
DMA_D15
D16
D17
D18
D19
D20
D21
D22
D23
DMA_D16
DMA_D17
DMA_D18
DMA_D19
DMA_D20
DMA_D21
DMA_D22
DMA_D23
D24
D25
D26
D27
D28
D29
D30
D31
DMA_D24
DMA_D25
DMA_D26
DMA_D27
DMA_D28
DMA_D29
DMA_D30
DMA_D31
D[0:63]
MBII DESIGN EXAMPLE
DMA DATA BUFFERS
SH.11
DATE '02-10-89
DRAWN BY' NM

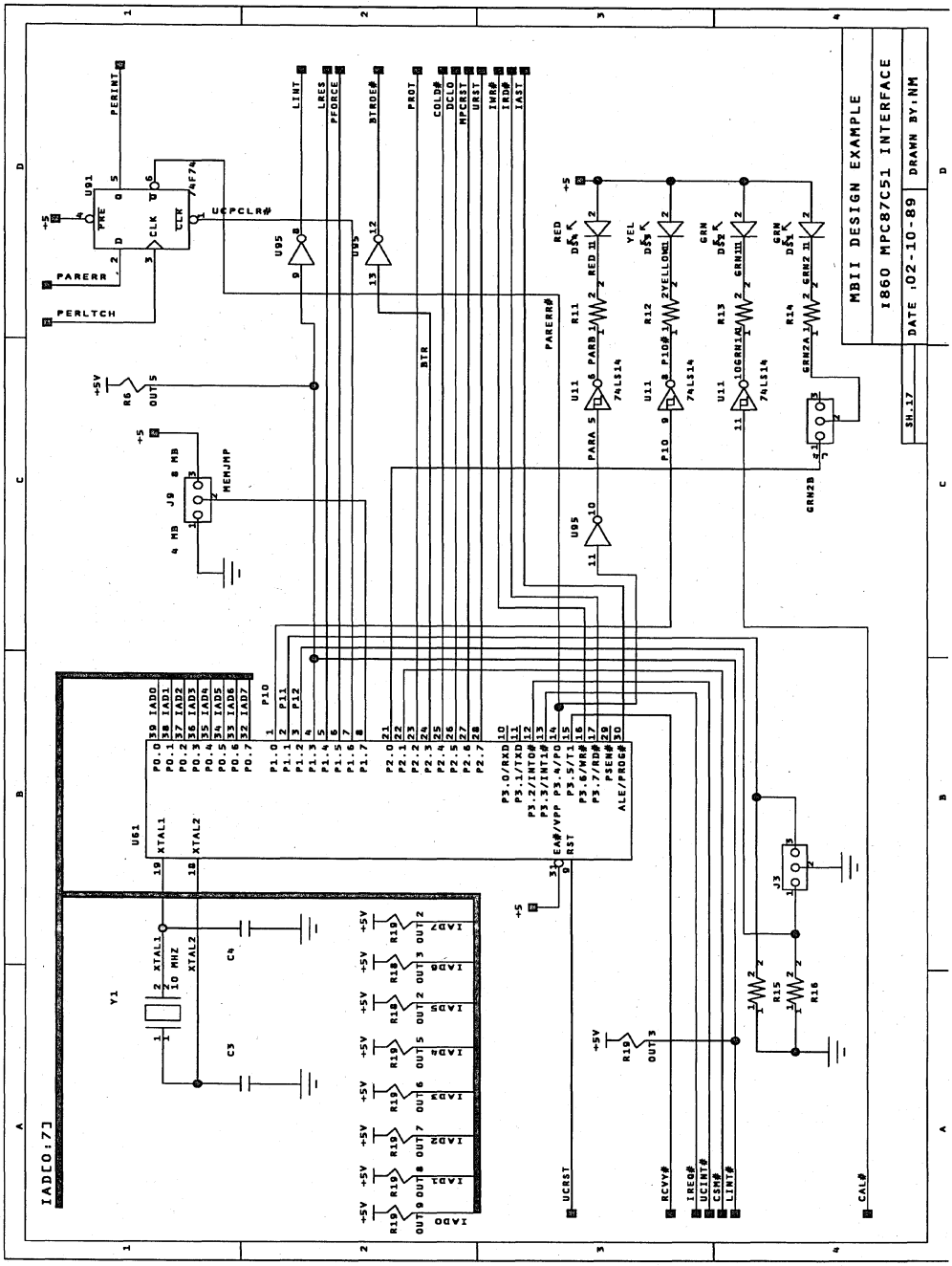

A[3:31]
MAA[0:9]
74BCT29827
U21
ACT29821
U20
RAE#
CAE#
CAL#
U40
74F74
CLKB
LADS#
+5
U114PU
U19
NADS#
DRAMSEL#
CLRCYC#
CPEN#
S[0:4]
LADS2#
U22 DCTLA
PAL16R4
RAS0#
RAS1#
NKEN#
RASP0#
PFORCE
A30
A29
A28
A24
+5V
R9
U18 DSTAT
PAL16R6
SREFREQ
NWR#
NNENE#
EXPBSY#
MBSY
U16 DCTLB
PAL16R6
DCLK
EWDL#
ERDL#
RTBA#
WEL
WDL
RDL
RTAB#
U17 DCTLC
PAL16R8
OEX#
U15 DCTLD
PAL20R8
NBE[0:7]
CSX#
U14
74F244
CS[0:3]#
OE0#
OE1#
BWR#
WE[0:7]#
U2
20 NS DELAY
CLKA
NNA#
BSY
NREADY#
EWDL#
ERDL#
RTDS#
EWDE#
ERDE#
PERLTCH
DRMEN#
DRMDIR
CLRFRQ
WPL
RTAB#
RDL
WDL
RTBA#
RAE#
MBII DESIGN EXAMPLE
I860 DRAM CONTROL
SH.12
DATE :02-10-89
DRAWN BY: NM

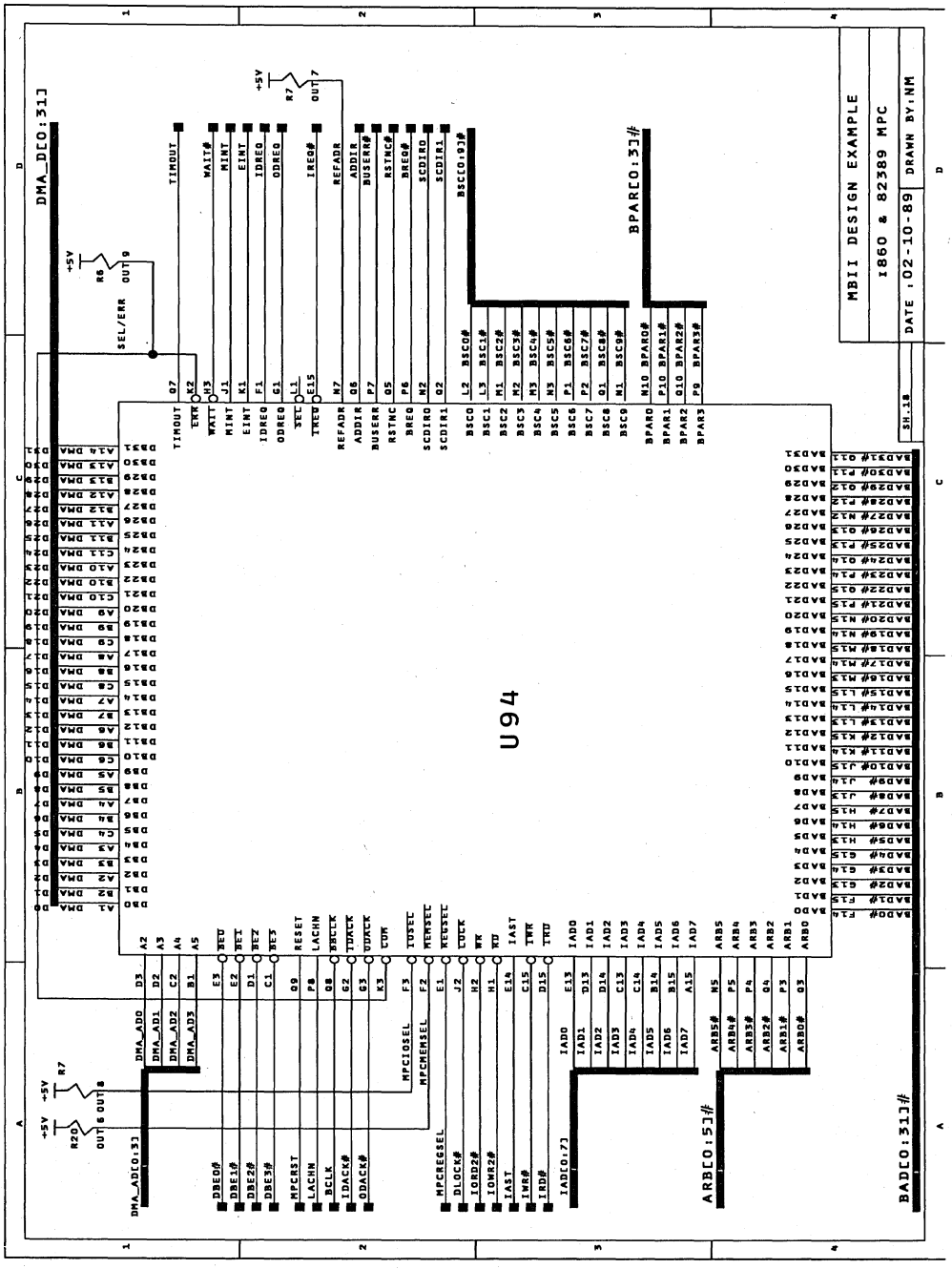

82380
U47
DMA_D[0:31]
DMA_AD[0:29]
MBII DESIGN EXAMPLE
I860 DRAM CONTROL
SH.13
DATE : 02-10-89
DRAWN BY: NM

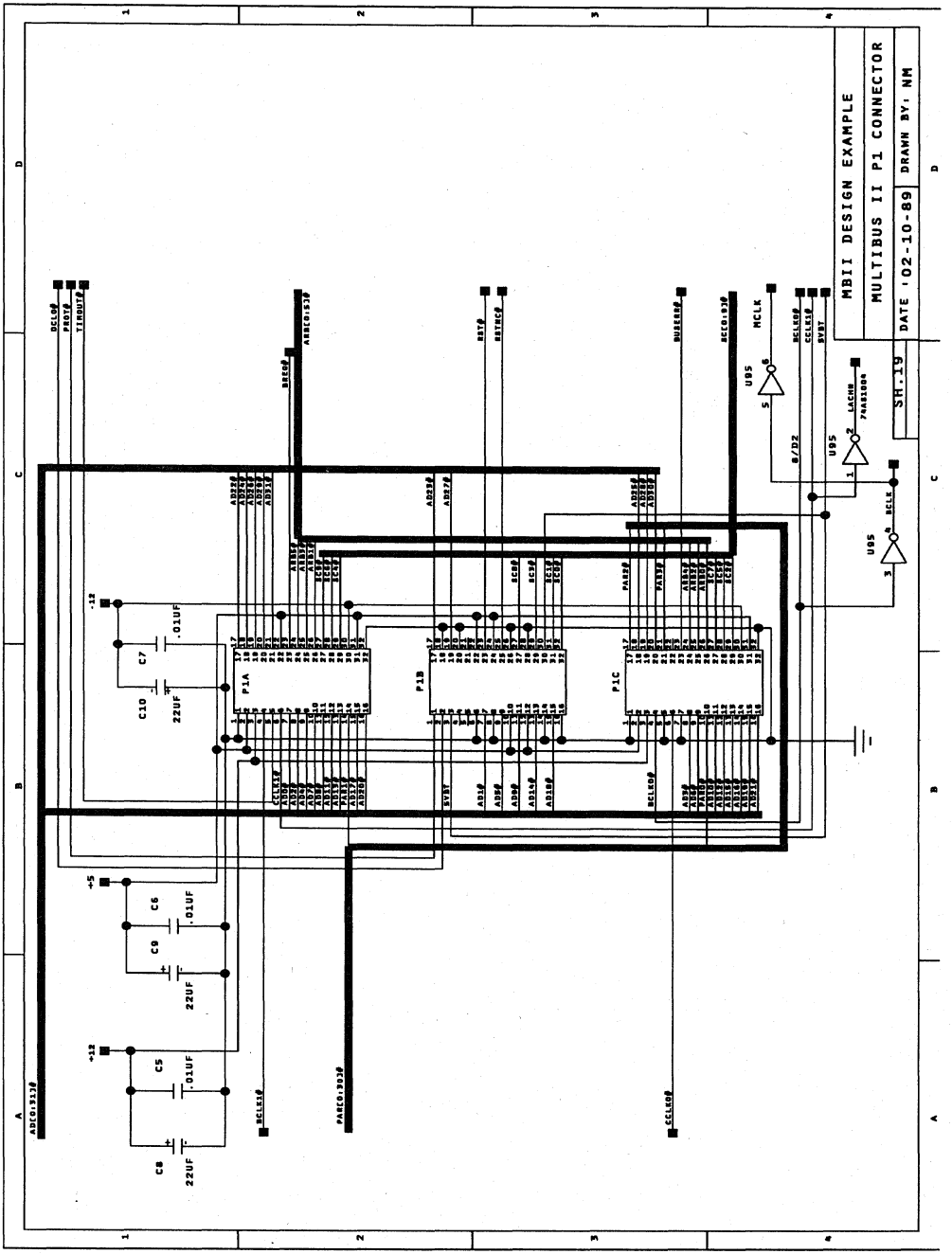

MBII DESIGN EXAMPLE
I860 PARITY GENERATOR
DATE : 02-10-89
DRAWN BY: NM
SH-14
WPDL
PAREER
WPD[0:7]
OPD[0:7]
DPL[0:7]
DC[0:63]
RPD[0:7]
DRMDIR
RDL
74AS374
74AS280
74AS30
+5V

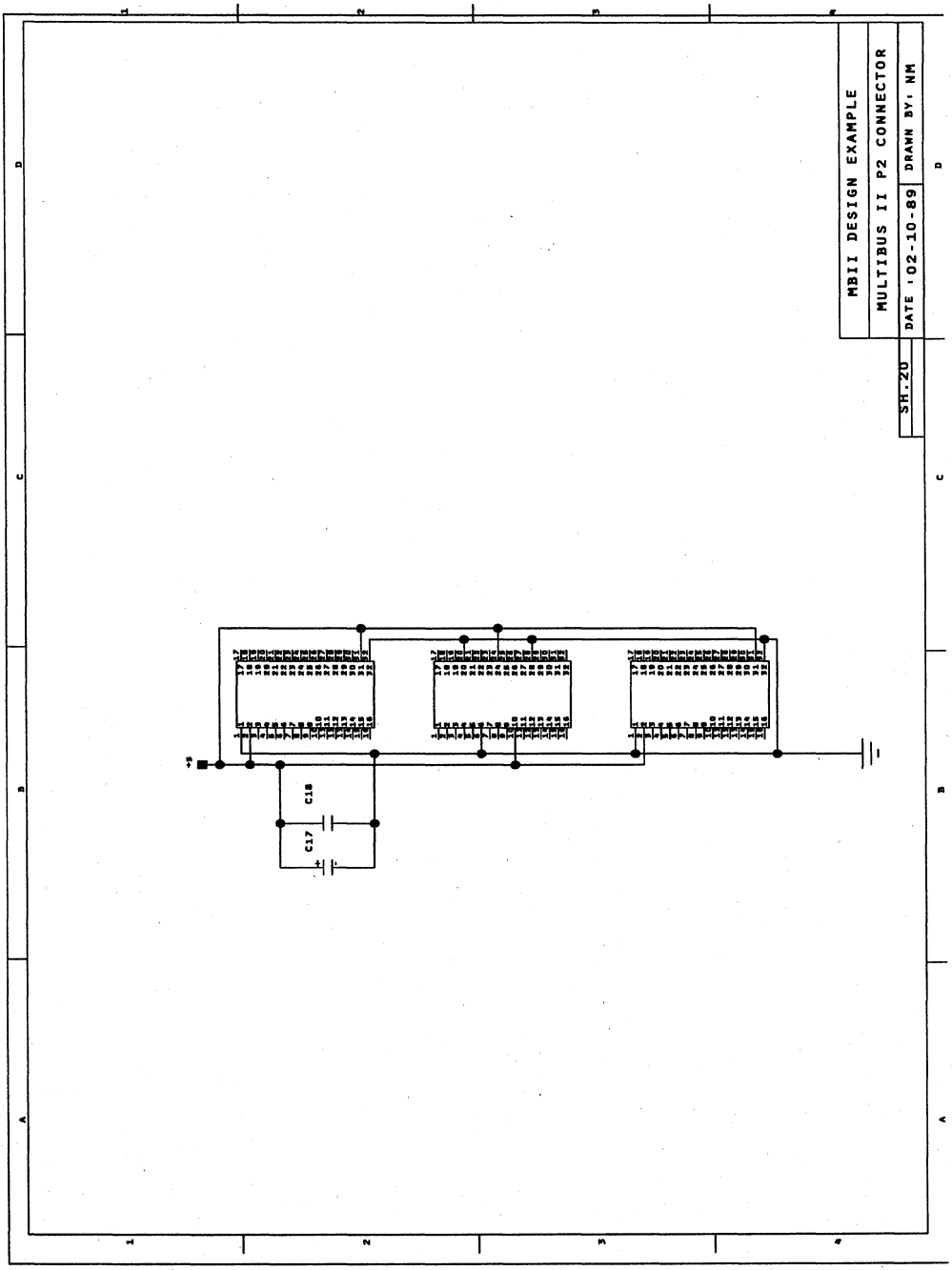

MAA[0:9]
RPD[0:7]
RASP0#
+5V
R6
OUT 4
U7
U8
U9
U10
U3
U4
U5
U6
DOUT
3 RPD0
3 RPD2
3 RPD4
3 RPD6
3 RPD1
3 RPD3
3 RPD5
3 RPD7
CS[0:3]#
WE[0:7]#
WPD[0:7]
MBII DESIGN EXAMPLE
I860 PARITY DRAMS
SH.15
DATE :02-10-89
DRAWN BY: NM

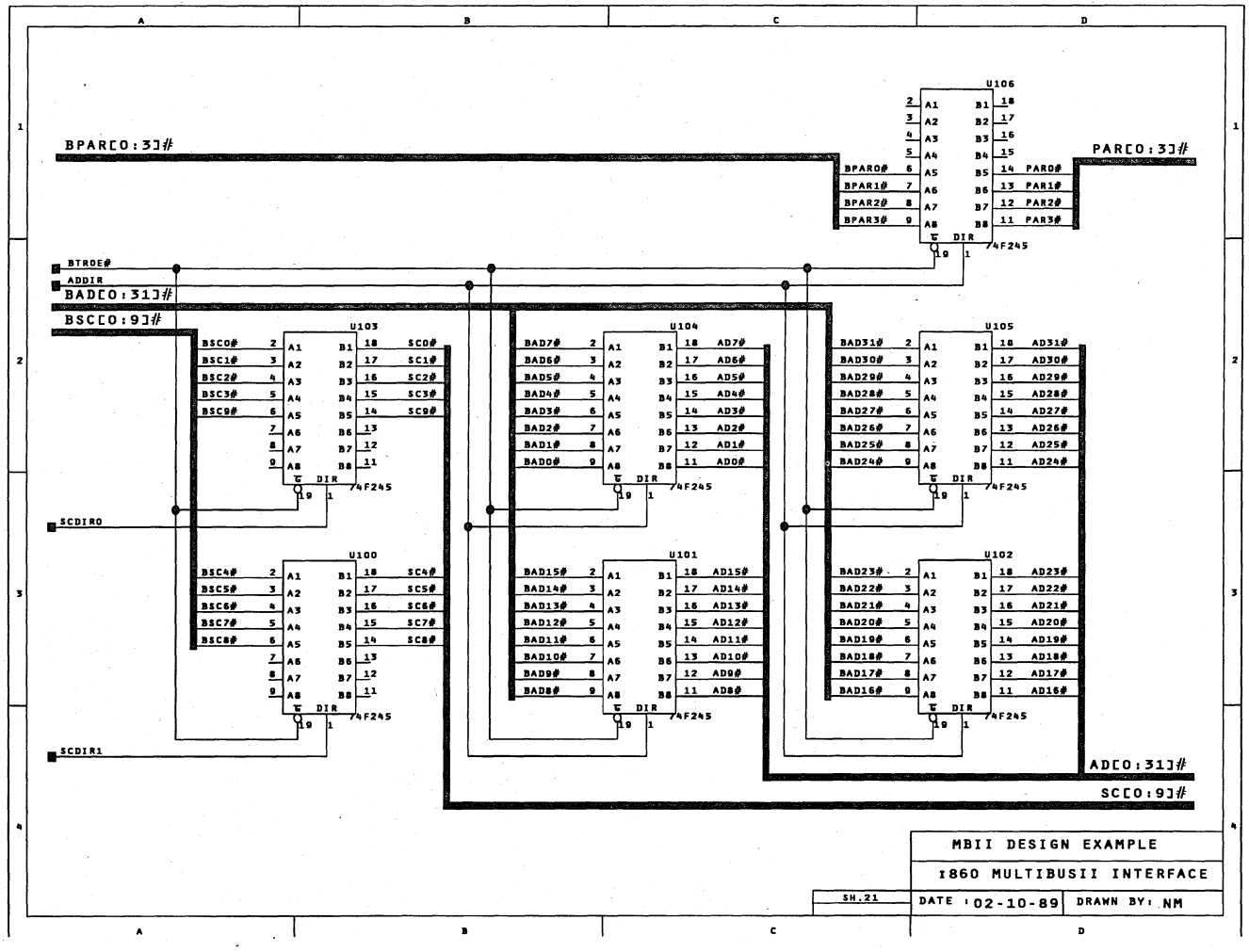

MD[0:63]
DRMEN#
RTBA#
RDL
WDL
RTAB#
DRMDIR
D[0:63]
U81
U82
U86
U83
U88
U84
U87
U85
74AS646
WDL 1
RTAB# 2
DRMDIR 3
CAB CBA
SAB SBA
DIR OE
A1 B1
A2 B2
A3 B3
A4 B4
A5 B5
A6 B6
A7 B7
A8 B8
23 RDL
22 RTBA#
21 DRMEN#
D0 4 20 MD0
D1 5 19 MD1
D2 6 18 MD2
D3 7 17 MD3
D4 8 16 MD4
D5 9 15 MD5
D6 10 14 MD6
D7 11 13 MD7
D8 4 20 MD8
D9 5 19 MD9
D10 6 18 MD10
D11 7 17 MD11
D12 8 16 MD12
D13 9 15 MD13
D14 10 14 MD14
D15 11 13 MD15
D16 4 20 MD16
D17 5 19 MD17
D18 6 18 MD18
D19 7 17 MD19
D20 8 16 MD20
D21 9 15 MD21
D22 10 14 MD22
D23 11 13 MD23
D24 4 20 MD24
D25 5 19 MD25
D26 6 18 MD26
D27 7 17 MD27
D28 8 16 MD28
D29 9 15 MD29
D30 10 14 MD30
D31 11 13 MD31
D32 4 20 MD32
D33 5 19 MD33
D34 6 18 MD34
D35 7 17 MD35
D36 8 16 MD36
D37 9 15 MD37
D38 10 14 MD38
D39 11 13 MD39
D40 4 20 MD40
D41 5 19 MD41
D42 6 18 MD42
D43 7 17 MD43
D44 8 16 MD44
D45 9 15 MD45
D46 10 14 MD46
D47 11 13 MD47
D48 4 20 MD48
D49 5 19 MD49
D50 6 18 MD50
D51 7 17 MD51
D52 8 16 MD52
D53 9 15 MD53
D54 10 14 MD54
D55 11 13 MD55
D56 4 20 MD56
D57 5 19 MD57
D58 6 18 MD58
D59 7 17 MD59
D60 8 16 MD60
D61 9 15 MD61
D62 10 14 MD62
D63 11 13 MD63
MBII DESIGN EXAMPLE
DRAM DATA BUFFERS
SH.16
DATE :02-10-89
DRAWN BY: NM

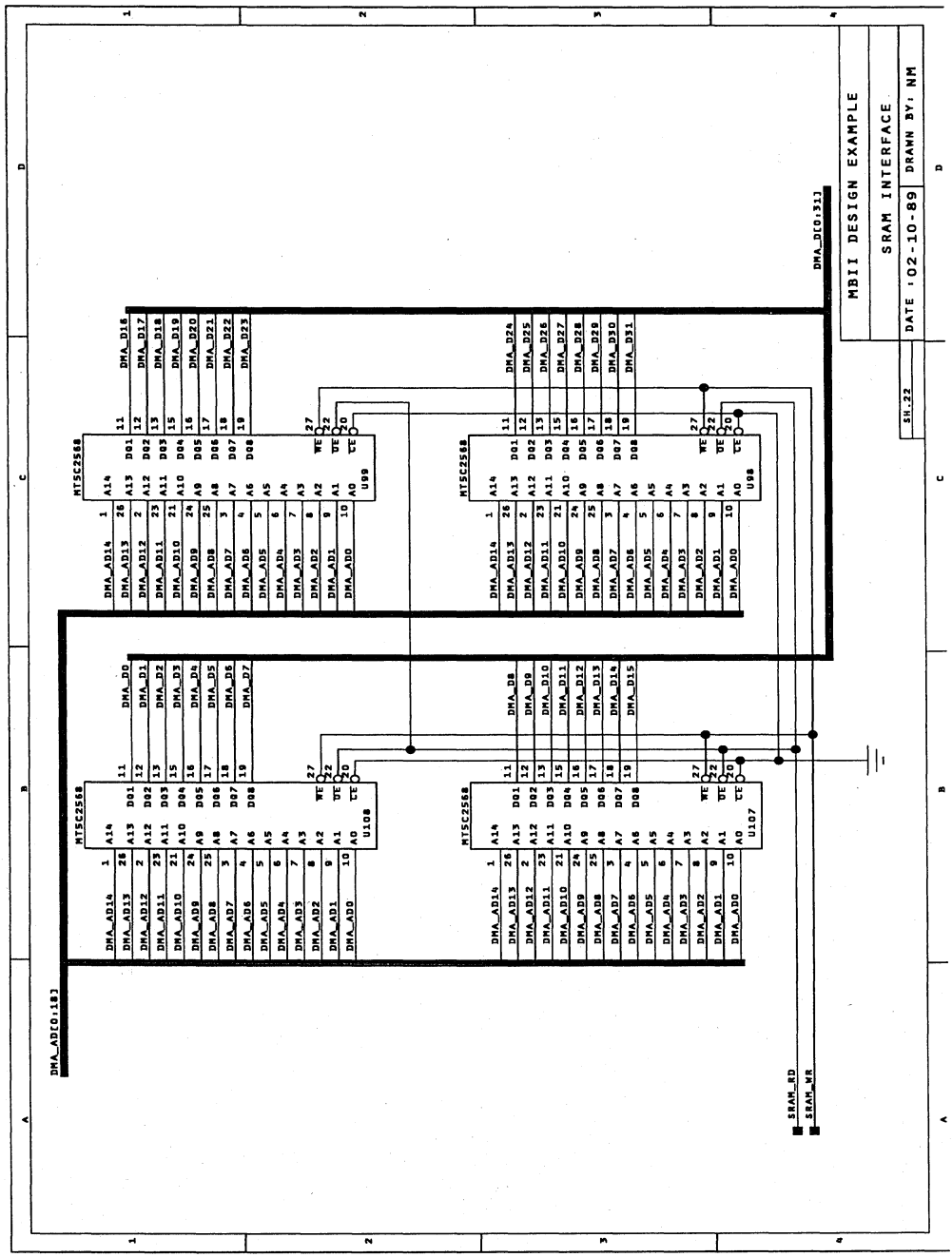

MBII DESIGN EXAMPLE
1860 MPC87C51 INTERFACE
DATE :02-10-89
DRAWN BY:NM
SH.17
PERINT
PARERR
PERLTCH
UCPCLR#
LINT
LRES
PFORCE
BIRQE#
PROT
COLD#
DCLO
MPCRST
URST
INH#
IRU#
LAST
PARERR#
BTR
RED
YEL
GRN
GRN2A
GRN2B
MEMJMP
4 MB
8 MB
XTAL1
XTAL2
10 MHZ
EA#/VPP
RST
ALE/PROG#
PSEN#
UCRST
RCVY#
IRES#
UCINT#
CSM#
LINT#
CAL#
IAD[0:7]

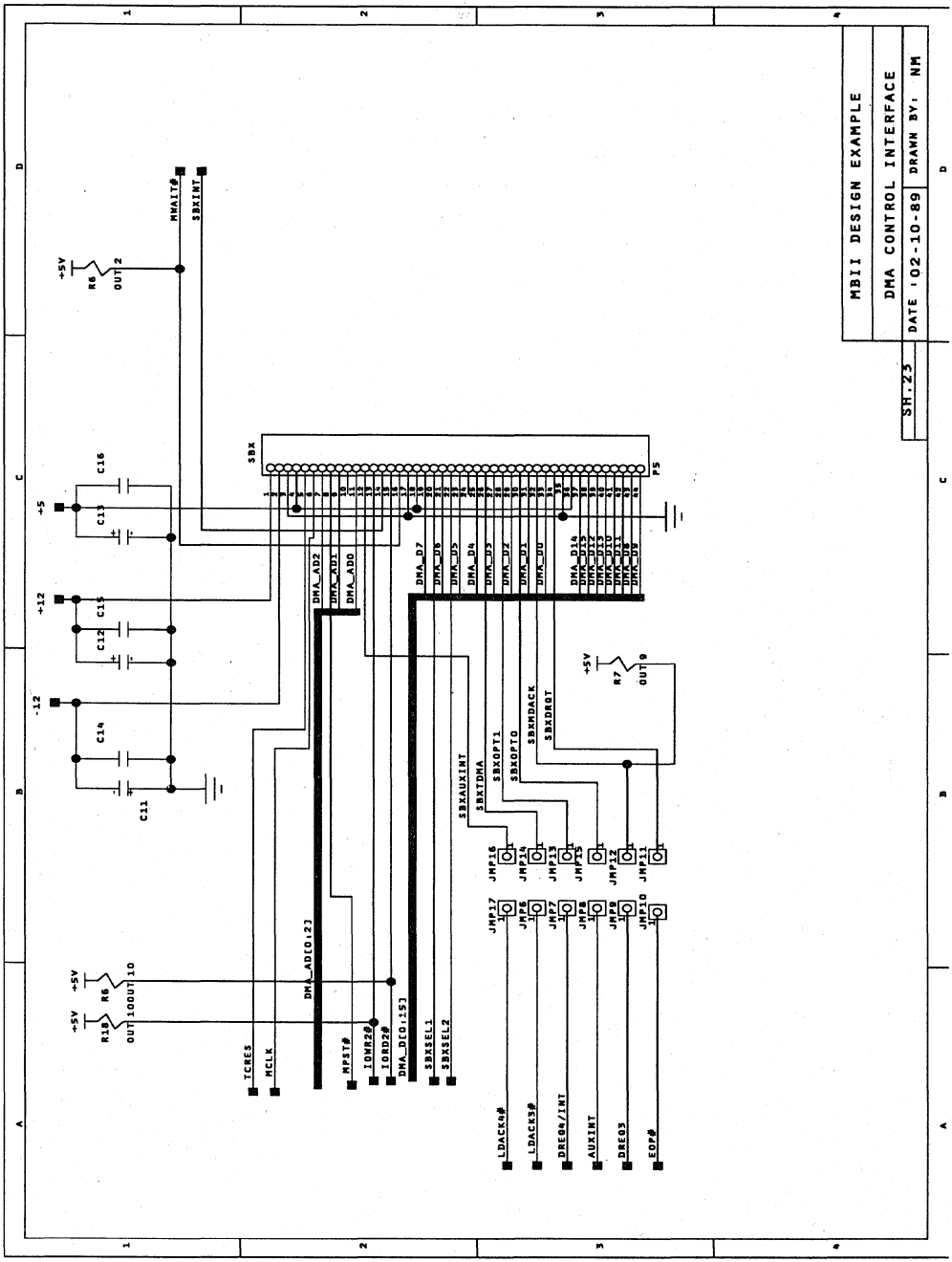

MBII DESIGN EXAMPLE
i860 & 82389 MPC
DATE :02-10-89
DRAWN BY:NM
U94
DMA_DEQ:31]
BPARQ:3]#
BADQ:31]#
ARBQ:5]#
DMA_ADQ:3]
TIMOUT
WAIT#
MINT
EINT
IDREQ
ODREQ
IREQ#
REFADR
ADDIR
BUSERR#
RSTNC#
BREQ#
SCDIRO
SCDIR1
BSCQ:0]#
BPAR0
BPAR1
BPAR2
BPAR3
DBE0#
DBE1#
DBE2#
DBE3#
MPCRST
LACHN
BCLK
IDACK#
ODACK#
MPCREGSEL
DLOCK#
IORD2#
IOWR2#
IAST
IWR#
IRD#
IADQ:7]
ARB5#
ARB4#
ARB3#
ARB2#
ARB1#
ARB0#
SEL/ERR
+5V
R6
R7
R20

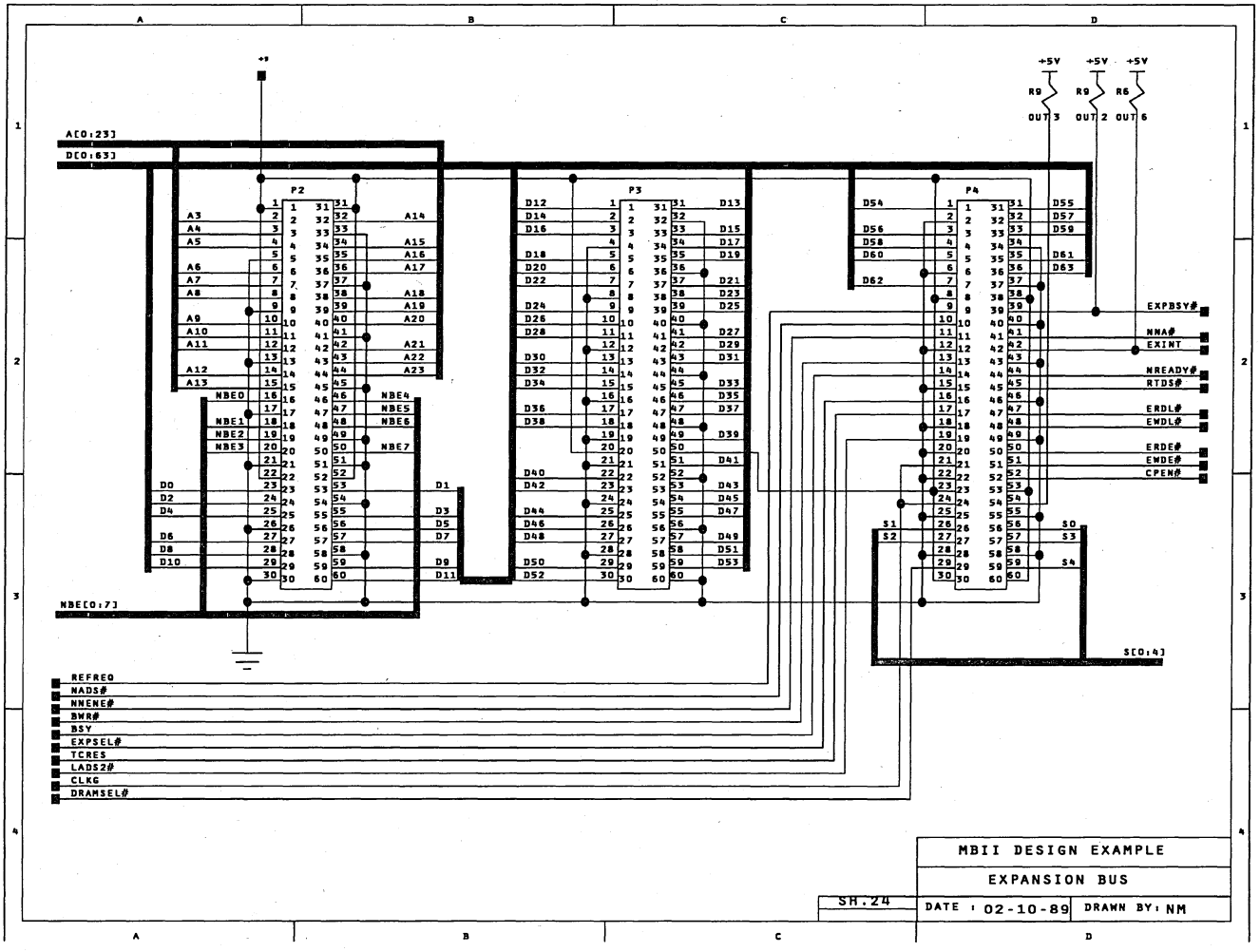

MBII DESIGN EXAMPLE
MULTIBUS II P1 CONNECTOR
DATE: 02-10-89
DRAWN BY: NM
SH.19
P1A
P1B
P1C
C7
.01UF
C10
22UF
C6
C9
C5
C8
+5
+12
U95
MCLK
8/D2


MBII DESIGN EXAMPLE
MULTIBUS II P2 CONNECTOR
DATE : 02-10-89
DRAWN BY: NM
SH-20
C18
C17


BPAR[0:3]#
PAR[0:3]#
BTRDE#
ADDIR
BAD[0:31]#
BSC[0:9]#
SCDIR0
SCDIR1
AD[0:31]#
SC[0:9]#
U106
U103
U104
U105
U100
U101
U102
74F245
DIR
MBII DESIGN EXAMPLE
I860 MULTIBUSII INTERFACE
SH.21
DATE : 02-10-89
DRAWN BY: NM


MBII DESIGN EXAMPLE
SRAM INTERFACE
SH.22
DATE : 02-10-89
DRAWN BY: NM
DMA_AD[0:18]
DMA_D[0:31]
SRAM_RD
SRAM_WR
MTSC2568
U108
U107
U98
U99


MBII DESIGN EXAMPLE
DMA CONTROL INTERFACE
DATE : 02-10-89
DRAWN BY: NM
SH.23
SBX
P5
MWAIT#
SBXINT
DMA_AD2
DMA_AD1
DMA_AD0
DMA_D7
DMA_D6
DMA_D5
DMA_D4
DMA_D3
DMA_D2
DMA_D1
DMA_D0
DMA_AD(0:2)
DMA_DIO(:15)
SBXAUXINT
SBXTDMA
SBXOPT1
SBXOPT0
SBXMDACK
SBXDRQT
TCRES
MCLK
MPST#
IOWR2#
IORD2#
SBXSEL1
SBXSEL2
LDACK4#
LDACK3#
DREQ4/INT
AUXINT
DREQ3
EOP#
JMP16
JMP14
JMP13
JMP15
JMP12
JMP11
JMP17
JMP6
JMP7
JMP5
JMP8
JMP10
C16
C13
C15
C12
C14
C11
R8
R7
R10
OUT2
OUT9
OUT10
+5V
+12
-12

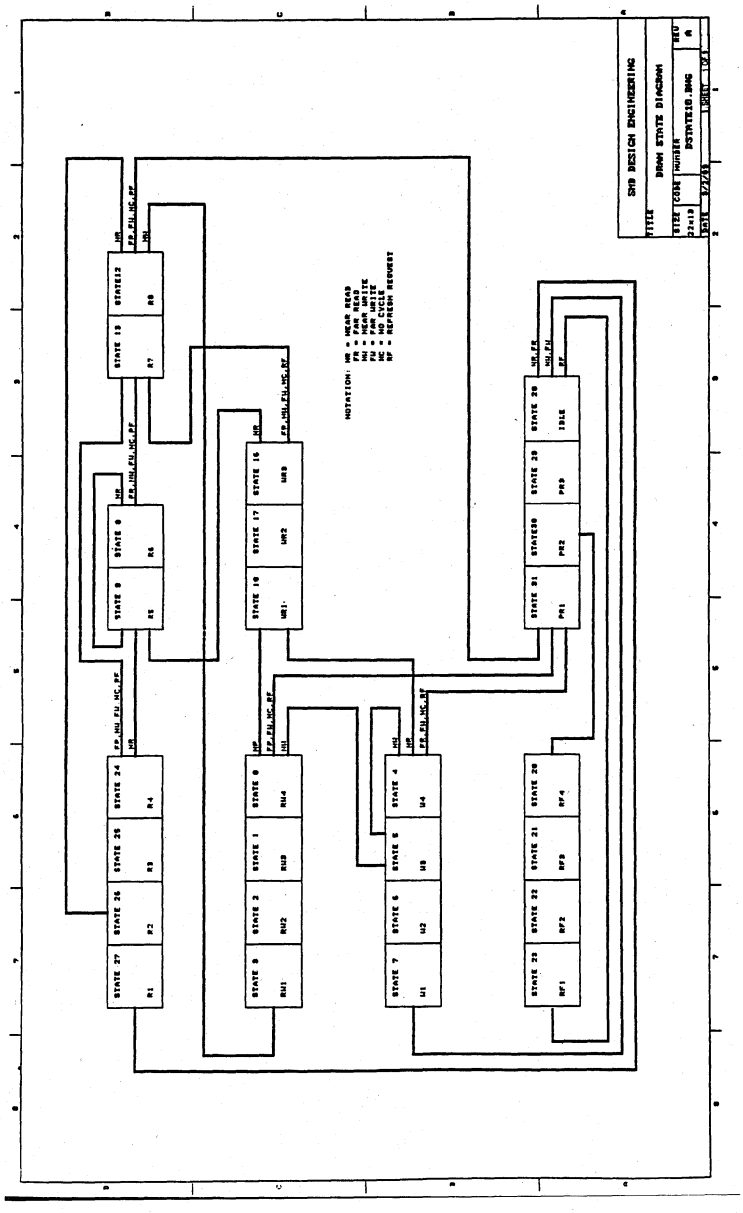

+5V
+5V
+5V
R9
R9
R6
OUT3
OUT2
OUT6
A[0:23]
D[0:63]
P2
P3
P4
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
A16
A17
A18
A19
A20
A21
A22
A23
NBE0
NBE1
NBE2
NBE3
NBE4
NBE5
NBE6
NBE7
NBE[0:7]
D0
D2
D4
D6
D8
D10
D1
D3
D5
D7
D9
D11
D12
D14
D16
D18
D20
D22
D24
D26
D28
D30
D32
D34
D36
D38
D40
D42
D44
D46
D48
D50
D52
D13
D15
D17
D19
D21
D23
D25
D27
D29
D31
D33
D35
D37
D39
D41
D43
D45
D47
D49
D51
D53
D54
D56
D58
D60
D62
D55
D57
D59
D61
D63
EXPBSY#
NNA#
EXINT
NREADY#
RTDS#
ERDL#
EWDL#
ERDE#
EWDE#
CPEN#
S1
S2
S0
S3
S4
S[0:4]
REFREQ
NADS#
NNENE#
BWR#
BSY
EXPSEL#
TCRES
LADS2#
CLKG
DRAMSEL#
MBII DESIGN EXAMPLE
EXPANSION BUS
SH.24
DATE : 02-10-89
DRAWN BY: NM

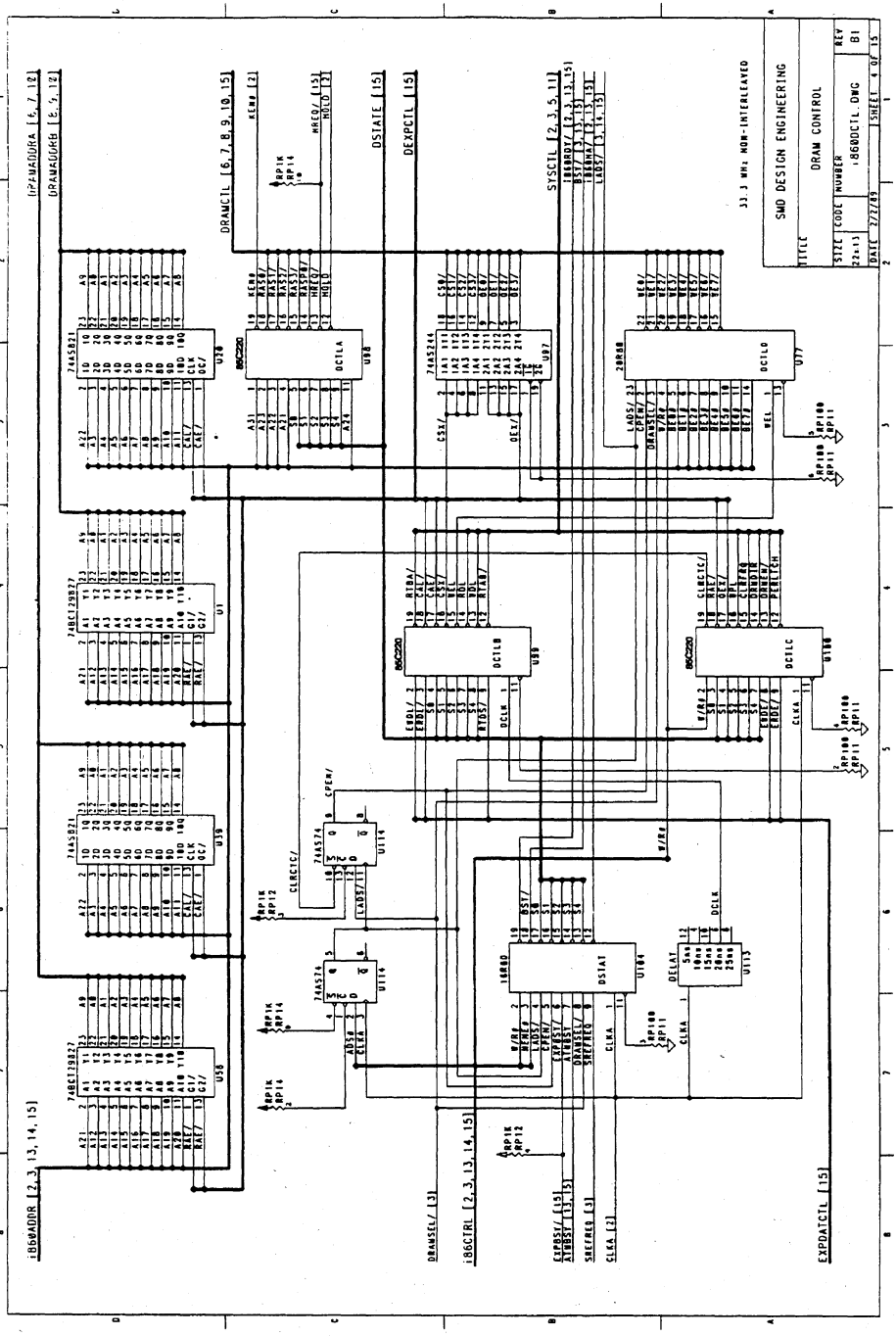
Memory Interface Schematics and PLD Code*

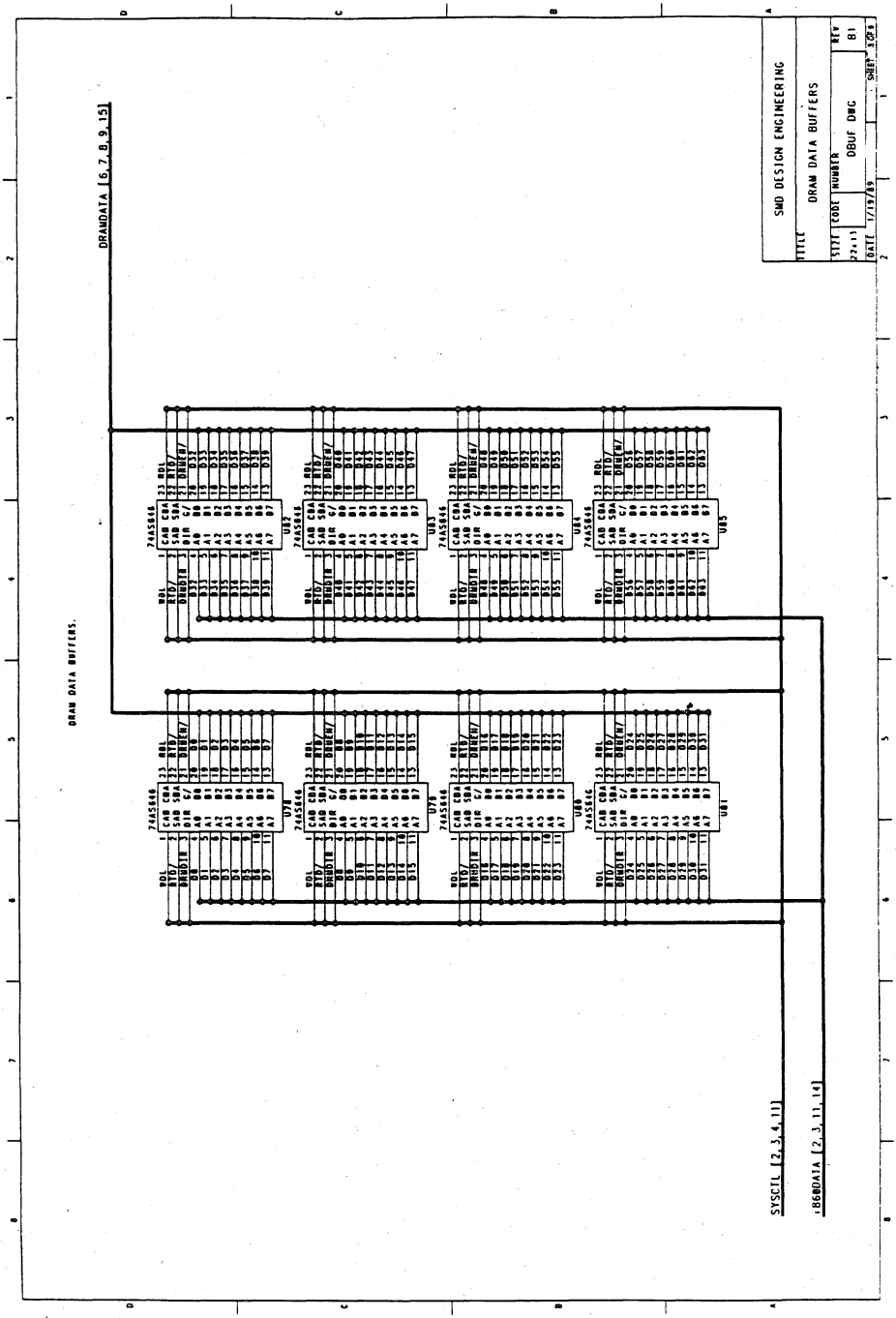
C

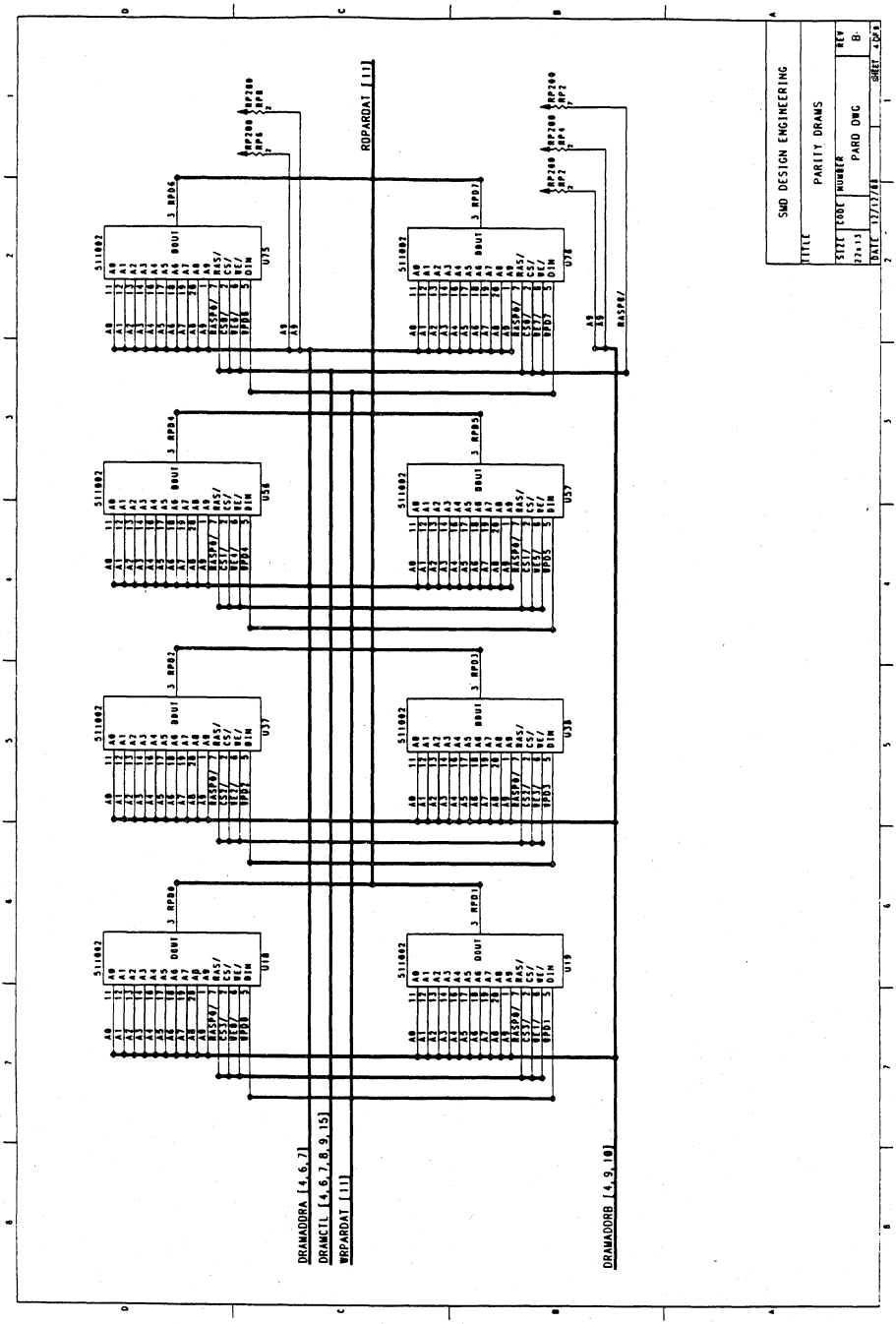
*This is a tested example.

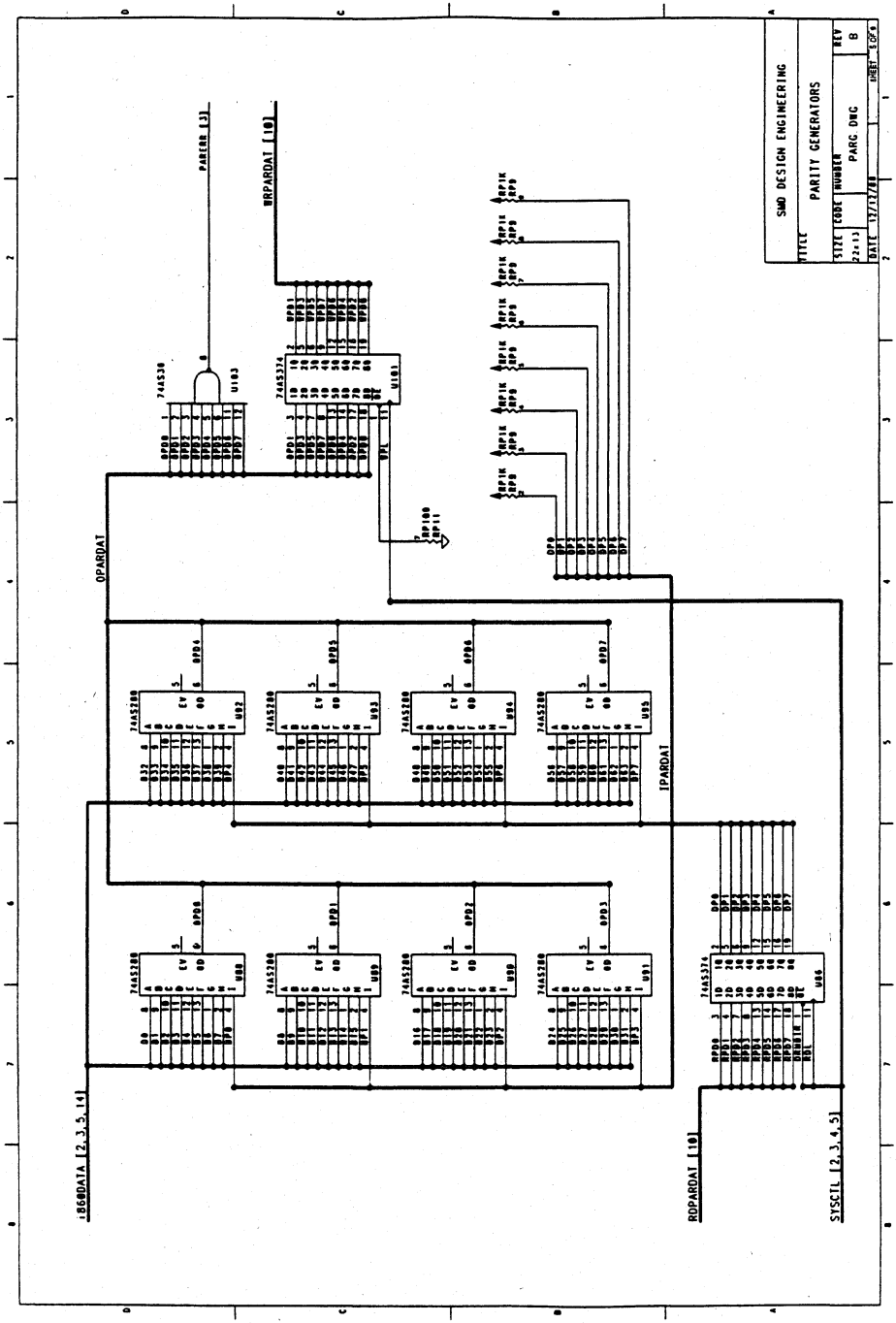
APPENDIX C
MEMORY INTERFACE
SCHEMATICS AND PLD CODE

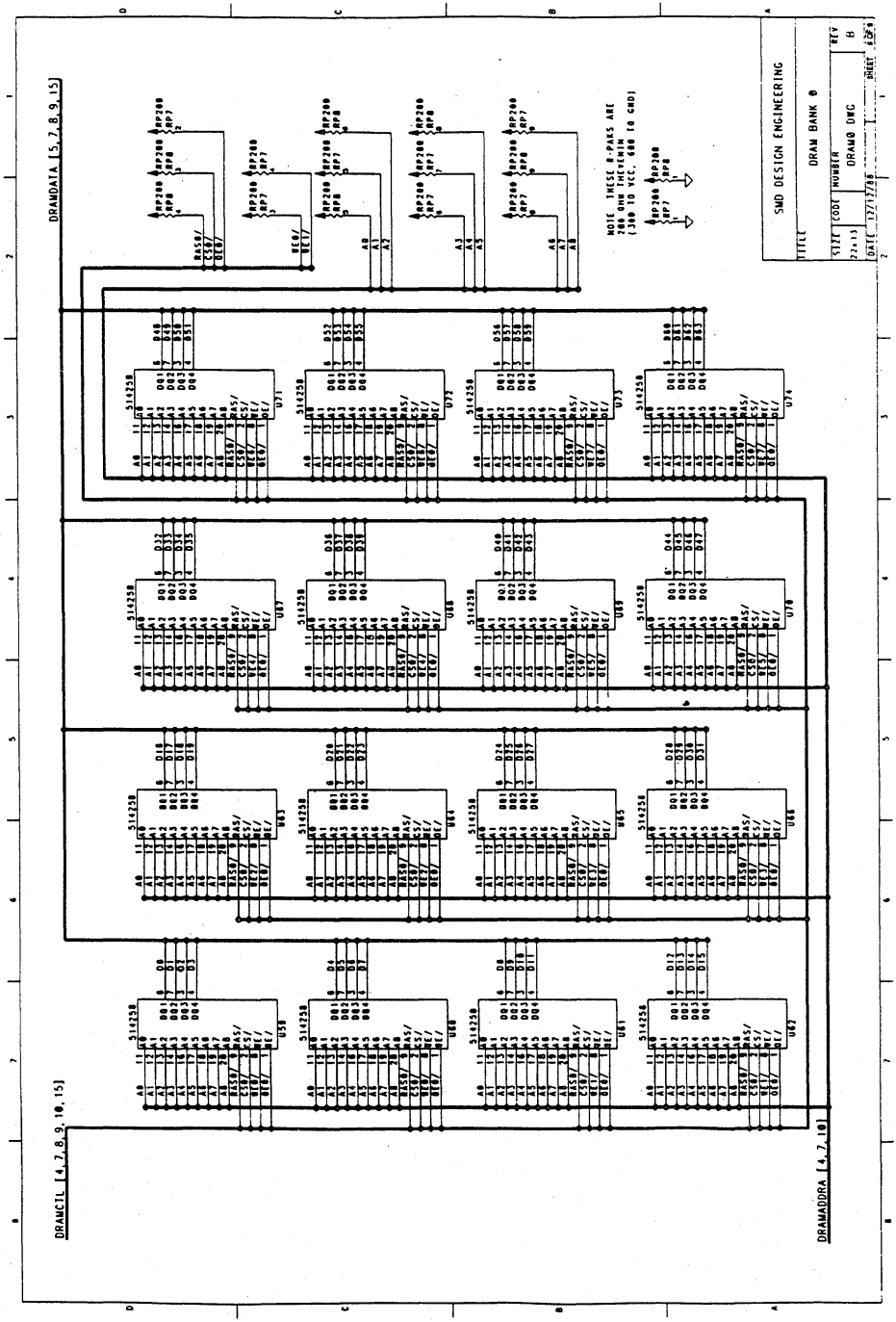


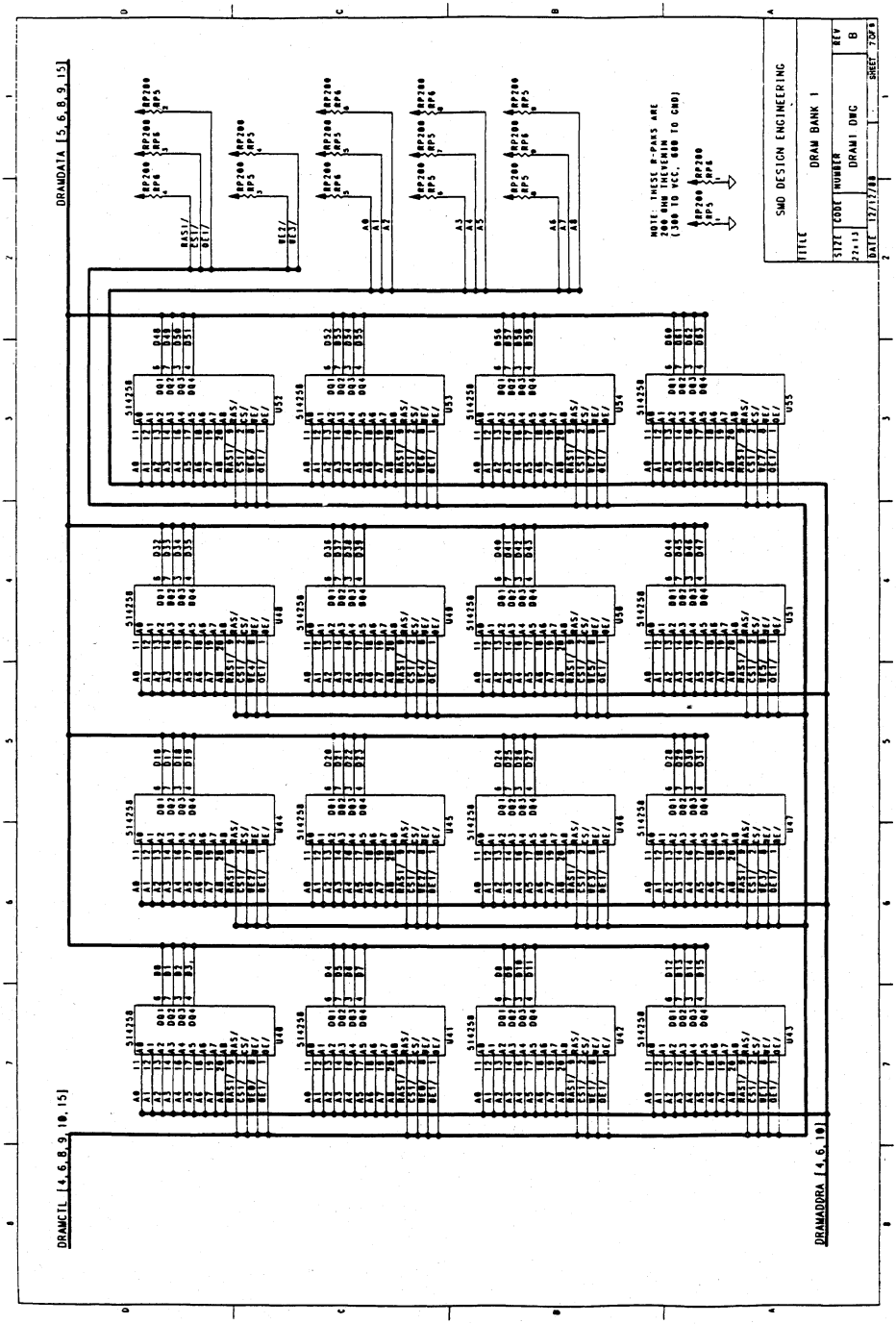

SWD DESIGN ENGINEERING
DRAM CONTROL
i860DCTL.DWG
REV B1
SHEET 4 OF 15
DRAWCTL [6,7,8,9,10,15]
DSTATE [15]
DEXPCTL [15]
SYSCTL [2,3,5,11]
i860ADDR [2,3,13,14,15]
i86CTRL [2,3,13,14,15]
EXPDATCTL [15]
DRAWSEL [13]
SHEFREQ [3]
CLKA [2]
DCLK
DSIAT
DCILA
DCILB
DCILC
DCILO

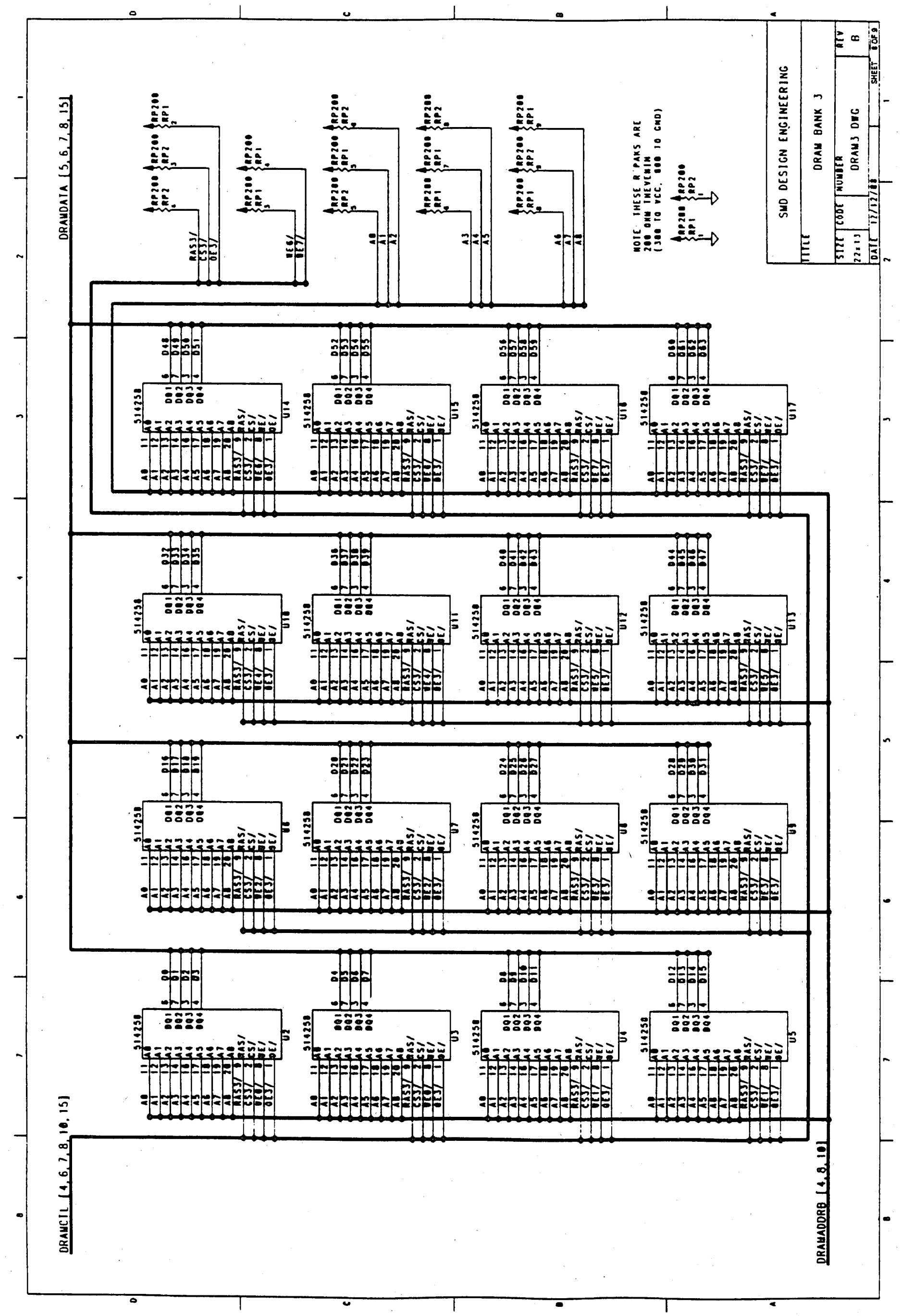

DRAM DATA BUFFERS.
DRAMDATA [6,7,8,9,15]
SYSCTL [2,3,4,11]
i860DATA [2,3,11,14]
SWD DESIGN ENGINEERING
DRAM DATA BUFFERS
DBUF DWG
U1
U2
U3
U4
U5
U6
U7
U8

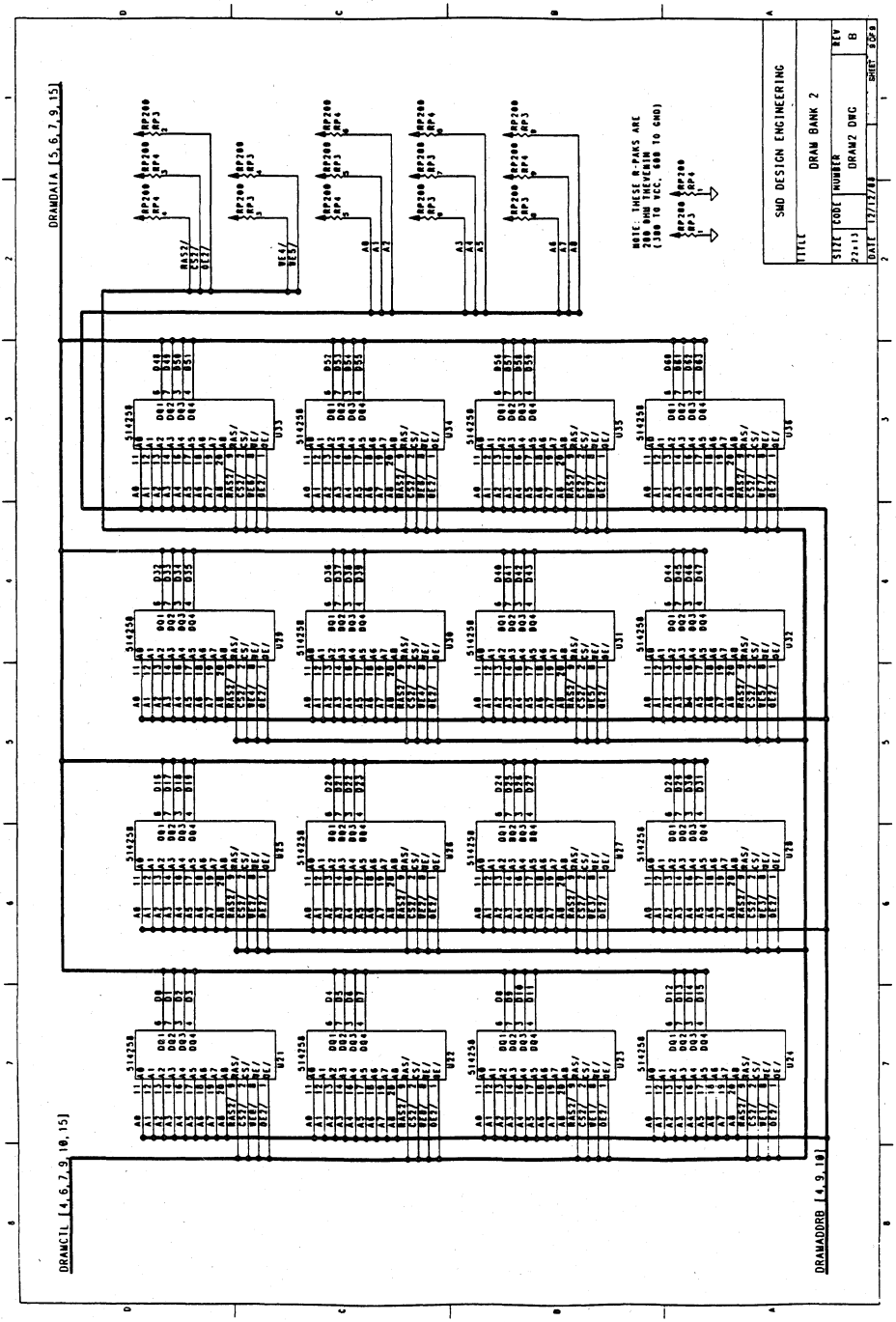

SMD DESIGN ENGINEERING
TITLE
PARITY DRAMS
SIZE
CODE
NUMBER
PARD DWG
REV
B
DATE 11/17/88
SHEET 4 OF 8
RDPARDAT [11]
WRPARDAT [11]
DRAMADDRA [4,6,7]
DRAMCTL [4,6,7,8,9,15]
DRAMADDRB [4,9,10]
RASP0/
511002
3 RPD6
3 RPD4
3 RPD2
3 RPD0
3 RPD3
3 RPD1
U75
U76
U56
U57
U37
U38
U18
U19


SWD DESIGN ENGINEERING
TITLE
PARITY GENERATORS
SIZE
CODE NUMBER
PARC DWG
REV
B
DATE 12/17/88
SHEET 1 OF 5
MRPARDAT [10]
PARERR [2]
OPARDAT
IPARDAT
i960DATA [2,3,5,14]
RDPARDAT [10]
SYSCTL [2,3,4,5]


DRAMDATA [5,7,8,9,15]
DRAMCTL [4,7,8,9,10,15]
DRAMADDRA [4,7,10]
SMD DESIGN ENGINEERING
DRAM BANK 0
DRAM0 DWG
NOTE: THESE 8-PAKS ARE
200 OHM THEVENIN
(300 TO VCC, 400 TO GND)


DRAMDATA [5,6,8,9,15]
DRAMCTL [4,6,8,9,10,15]
DRAMADDRA [4,6,10]
SMD DESIGN ENGINEERING
DRAM BANK 1
DRAM1 DWG
NOTE: THESE R-PAKS ARE
200 OHM THEVENIN
(300 TO VCC, 600 TO GND)


SMD DESIGN ENGINEERING
TITLE
DRAM BANK 3
SIZE
22x11
CODE
NUMBER
DRAM3 DWG
REV
B
DATE 12/12/88
SHEET 8 OF 9
NOTE: THESE R PAKS ARE
200 OHM THEVENIN
(300 TO VCC, 800 TO GND)
DRAMDATA [5, 6, 7, 8, 15]
DRAMCIL [4, 6, 7, 8, 10, 15]
DRAMADDRB [4, 8, 10]
RP200
514258
U2
U3
U4
U5
U6
U7
U8
U9
U10
U11
U12
U13
U14
U15
U16
U17


DRAMDATA [5,6,7,9,15]
DRAMCTL [4,6,7,9,10,15]
DRAMADDRB [4,9,10]
NOTE: THESE R-PAKS ARE
200 OHM THEVENIN
(300 TO VCC, 600 TO GND)
SMD DESIGN ENGINEERING
TITLE
DRAM BANK 2
SIZE
CODE
NUMBER
DRAW2 DWG
REV
B
DATE 12/12/88
SHEET 8 OF 8

```
"TITLE    DRAM Control A.
"PATTERN  DCTLAØ.ABL

" U98
" Reference Sheet 4 of the Schematics.
" This PLD must have a propagation delay of 1Øns or faster

module dctlaØ
U98 device 'EØ32Ø';                    "Intel 85C22Ø PLD

A31, A23, A22, A21, !SØ                pin 1,2,3,4,5;
!S1, !S2, !S3, !S4, GND                pin 6,7,8,9,1Ø;
NCØ, HOLD, !HREQ, !RASPØ, !RAS3        pin 11,12,13,14,15;
!RAS2, !RAS1, !RASØ, !KEN, VCC         pin 16,17,18,19,2Ø;

EQUATIONS

RASØ =  A23 &  A22 &  A21 &  S4 &  S3 & !S2 & !S1 & !SØ      "start write
     #  A23 &  A22 &  A21 & !S4 & !S3 &  S2 & !S1 & !SØ      "start read
     #  !S4 &  S3 & !S2 & !S1 &  SØ                          "start refresh
     # RASØ &  S4                                            "hold
     # RASØ &  S3                                            "hold
     # RASØ &  S2;                                           "hold

RAS1 =  A23 &  A22 & !A21 &  S4 &  S3 & !S2 & !S1 & !SØ      "start write
     #  A23 &  A22 & !A21 & !S4 & !S3 &  S2 & !S1 & !SØ      "start read
     #  !S4 &  S3 & !S2 & !S1 &  SØ                          "start refresh
     # RAS1 &  S4                                            "hold
     # RAS1 &  S3                                            "hold
     # RAS1 &  S2;                                           "hold

RAS2 =  A23 & !A22 &  A21 &  S4 &  S3 & !S2 & !S1 & !SØ      "start write
     #  A23 & !A22 &  A21 & !S4 & !S3 &  S2 & !S1 & !SØ      "start read
     #  !S4 &  S3 & !S2 & !S1 &  SØ                          "start refresh
     # RAS2 &  S4                                            "hold
     # RAS2 &  S3                                            "hold
     # RAS2 &  S2;                                           "hold

RAS3 =  A23 & !A22 & !A21 &  S4 &  S3 & !S2 & !S1 & !SØ      "start write
     #  A23 & !A22 & !A21 & !S4 & !S3 &  S2 & !S1 & !SØ      "start read
     #  !S4 &  S3 & !S2 & !S1 &  SØ                          "start refresh
     # RAS3 &  S4                                            "hold
     # RAS3 &  S3                                            "hold
     # RAS3 &  S2;                                           "hold
```

```
RASPØ =  A23  &  S4 &  S3 & !S2 & !S1 & !SØ                    "start write
      #  A23  & !S4 & !S3 &  S2 & !S1 & !SØ                    "start read
      #  !S4  &  S3 & !S2 & !S1 &  SØ                          "start refresh
      # RASPØ &  S4                                            "hold
      # RASPØ &  S3                                            "hold
      # RASPØ &  S2;                                           "hold

KEN   = A31;

!HOLD =  !HREQ;

end dctlaØ

" Description:
"This PLD decodes and drives the RAS lines for the DRAMs from
"the main state machine and the N1Ø addresses.  It drives the RAS
"lines for the 8 Mbytes on the EV-AT board.
"
"  The addresses (A21-A23) are decoded and activated when the
"DRAM state machine starts a read or a write cycle.  They are held
"active until the state machine enters the Precharge sequence
"(State 31).   All RAS lines are active during a refresh cycle.
"
```

```
"TITLE    DRAM Control B.
"PATTERN  DCTLB1.ABL

" U99
" This PLD decodes and drives numerous DRAM control signals.
" This PLD must have an external clock frequency of 58MHz or faster.

module dctlb1
U99 device 'E0320';                    "Intel 85C220 PLD

@ALTERNATE

DCLK, /EWDL, /ERDL, /S0, /S1           pin 1,2,3,4,5,;
/S2, /S3, /S4, /RTDS, GND              pin 6,7,8,9,10;
/OE, /RTAB, /WDL, /RDL, /WEL           pin 11,12,13,14,15;
/CSX, /CAE, /CAL, /RTBA, VCC           pin 16,17,18,19,20;

EQUATIONS

RTAB    = RTDS;

RTBA    = RTDS;

RDL     :=  /ERDL * /S0                    "/RDL IS HIGH DURING
        +   /ERDL * /S1                    "STATE 8,16,24
        +   /ERDL * /S2                    "
        +   /ERDL *  S3 * S4;              "

WEL     :=  S4 *  S3 *  S2 *  S1 * /S0     "state  1
        +   S4 *  S3 *  S2 * /S1 * /S0     "state  3
        +   S4 *  S3 * /S2 *  S1 * /S0     "state  5
        +   S4 *  S3 * /S2 * /S1 * /S0;    "state  7

WDL     :=  /EWDL * /S0                    "/WDL IS HIGH DURING
        +   /EWDL * /S3                    "STATE 0,2,4,6
        +   /EWDL * /S4;                   "

CSX     :=  S4 *  S3 *  S2       * /S0     "state  1, 3
        +   S4 *  S3 * /S2 *  S1 * /S0     "state  5
        +   S4 * /S3 *  S2 *  S1           "state  8, 9
        +   S4 * /S3 * /S2 *  S1           "state 12,13
        +  /S4 *  S3 *  S2 *  S1           "state 16,17
        +  /S4 *  S3 * /S2 * /S1           "state 22,23
        +  /S4 * /S3 *  S2 *  S1           "state 24,25
        +  /S4       *  S2 * /S1 *  S0;    "state 18,26

CAL     := /S0;                            " all odd states.
```

```
CAE     :=  S4 *  S3 *  S2                      "states Ø,1,2,3
        +   S4 *  S3 * /S2 *  S1                "states 4,5
        +   S4 *  S3 * /S2 * /S1 *  SØ          "state  6
        +   S4 * /S3       *  S1                "states 8,9,12,13
        +  /S4 *  S3 *  S2 *  S1                "states 16,17
        +  /S4 * /S3 *  S2 *  S1                "states 24,25
        +  /S4       *  S2 * /S1 *  SØ;         "state  18,26

end dctlb1
" Description:  Refer to the DRAM State Machine document for an explaination
"               of when these signals are to be active.
"
"       CAE     Column Address Enable.  Turns on the Column address drivers.
"
"       CAL     Column Address Latch.  Latches the N1Ø addresses for the DRAM
"               column address.
"
"       CSX     Chip Select for the DRAM array.
"
"       WEL     Write Enable Latch. Latches the BEN bits to drive the active
"               DRAM WEs.
"
"       WDL     Write Data Latch. For DRAM accesses only.
"
"       RDL     Read Data Latch. For DRAM accesses only.
"
"
"
"
"       CHANGE HISTORY:
"       This PLD adds the state 18 to eliminate the tCAC timing violation
"       for reads immediately following writes.
```

```
"TITLE    DRAM Control C.
"PATTERN  DCTLC3.ABL

" U100
" This PLD decodes and drives numerous DRAM control signals.
" This PLD must have an external count frequency of 58MHz or faster

module dctlc3
U100 device 'E0320';                              "Intel 85C220 PLD

@ALTERNATE

CLKA, WR, /S0, /S1, /S2                           pin 1,2,3,4,5;
/S3, /S4, /EWDE, /ERDE, GND                       pin 6,7,8,9,10;
/OE, /PERLTCH, /DRMEN, /DRMDIR, /CLRFRQ           pin 11,12,13,14,15;
/WPL, /OEX, /RAE, /CLRCYC, VCC                    pin 16,17,18,19,20;

EQUATIONS

PERLTCH  =  S4 * /S3 *  S2 *  S1 * /S0            "state 9
         +  S4 * /S3 * /S2 *  S1 * /S0;           "state 13

CLRCYC   =  S4 *  S3 *  S2 *  S1 * /S0            "state 1
         +  S4 *  S3 * /S2 *  S1 * /S0            "state 5
         +  S4 * /S3 *  S2 *  S1 * /S0            "state 9
         + /S4 *  S3 *  S2 * /S1 *  S0            "state 18
         + /S4 * /S3 *  S2 *  S1 * /S0;           "state 25

OEX      :=  S4 *  S3 *  S2 *  S1 *  S0 * /WR     "state 0  if near read cycle.
         +   S4 *  S3 * /S2 *  S1 *  S0 * /WR     "state 4  if near read cycle.
         +   S4 * /S3 *  S2 *  S1                 "state 8,9
         +  /S4 *  S3 *  S2 *  S1                 "state 16,17
         +  /S4 *  S3 *  S2 * /S1 *  S0           "state 18
         +  /S4 * /S3 *  S2;                      "state 24,25,26,27

RAE      := /S4 * /S3 * /S2 *  S1                 "state 28,29
         +  /S4 * /S3 * /S2 * /S1 *  S0;          "state 30

CLRFRQ   :=  S4                                   "ALL EXCEPT state 23
         +  /S4 *  S3 *  S2
         +  /S4 *  S3 * /S2 *  S1
         +  /S4 *  S3 * /S2 * /S1 *  S0
         +  /S4 * /S3;

WPL      :=  S4 *  S3 *  S2 *  S1 * /S0           "state 1
         +   S4 *  S3 *  S2 * /S1 * /S0           "state 3
         +   S4 *  S3 * /S2 *  S1 * /S0           "state 5
         +   S4 *  S3 * /S2 * /S1 * /S0;          "state 7
```

```
DRMEN  :=  S4 *  S3       * /S1                "state 2,3,6,7
       +   S4 *  S3       *  S1 * /SØ          "state 1,5
       +   S4 * /S3 *  S2 *  S1                "state 8,9
       +  /S4       *  S2 *  S1                "state 16,17,24,25
       +   S4 *  S3       *  S1 *  SØ *  WR    "state Ø,4  if write cycle.
       +   EWDE
       +   ERDE;

DRMDIR :=  S4 * /S3 *  S2 *  S1                "state 8,9
       +   S4 * /S3 * /S2 *  S1 *  /SØ         "state 13
       +  /S4       *  S2 * /S1 *  SØ          "state 18,26
       +  /S4       *  S2 *  S1                "state 16,17,24,25
       +   ERDE;

end dctlc3
" Description:  Refer to the DRAM State Machine document for an explaination
"               of when these signals are to be active.
"
"
"       PERLTCH         Parity Error Latch.  Latches parity errors (if any)
"                       during DRAM read cycles.
"
"       CLRFRQ          Clear Refresh Request.  Restarts the refresh counter.
"
"       WPL             Write Parity Latch.  Latches the parity bits during a
"                       DRAM write.
"
"       OEX             Output Enable.  For the DRAMS.
"
"       RAE             Row Address Enable.  Turns on the row addresses to the
"                       DRAM.
"
"       CLRCYC          Clear Cycle.  Clears queued cycles.

" CHANGE HISTORY
"       REV 1    change to 16R6D to correct for speed path problem, and fix
"                possible buffer contention.
"       REV 2    add state 18 to fix tCAC violation.

"       REV 3    Added state 1 to CLRCYC
```

```
"TITLE    N1Ø EV-AT DRAM Control State Machine.
"PATTERN  DSTAT1.PDS

" U1Ø4
" This PLD contains the state machine for the DRAM controller.
" This PLD must have an external clock frequency of 58MHz or faster.

module dstat1
U1Ø4 device 'EØ32Ø';                    "Intel 85C22Ø PLD

@ALTERNATE

CLKA, WR, /NENE, /LADS, /CPEN           pin 1,2,3,4,5;
/EXPBSY, ATMBSY, /DSEL, REFREQ, GND     pin 6,7,8,9,1Ø;
/OE, /NA, /S4, /S3, /S2                 pin 11,12,13,14,15;
/S1, /SØ, /DRAMBSY, /RDY, VCC           pin 16,17,18,19,2Ø;

EQUATIONS

SØ      :=   S4 *  S3       *  S1 *  SØ *  LADS * /WR *  DSEL *  NENE * /REFREQ
          +   S4 * /S3 * /S2 *  S1 *  SØ *  LADS * /WR *  DSEL *  NENE * /REFREQ
          +   S4 * /S3 * /S2 *  S1 *  SØ *  CPEN * /WR *  DSEL *  NENE * /REFREQ
          +  /S4 *  S3 * /S2 *  S1 *  SØ
          +  /S4 * /S3 * /S2 *  S1 *  SØ * /LADS * /CPEN * /REFREQ
          +  /S4 * /S3 * /S2 *  S1 *  SØ * /DSEL * /REFREQ
          +  /SØ;

S1      :=   S4 *  S3       *  S1 *  SØ *  LADS *  WR *  DSEL *  NENE * /REFREQ
          +   S4 * /S3 *  S2 *  S1 *  SØ
          +  /S4       *  S2 *  S1 *  SØ
          +  /S4 * /S3 * /S2 *  S1 *  SØ * /LADS * /CPEN * /REFREQ
          +  /S4 * /S3 * /S2 *  S1 *  SØ * /DSEL * /REFREQ
          +  /S1 *  SØ
          +   S1 * /SØ;

S2      :=   S4             *  S1 *  SØ *  LADS * /WR *  DSEL *  NENE * /REFREQ
          +   S4 * /S3 * /S2 *  S1 *  SØ *  LADS       *  DSEL *  NENE * /REFREQ
          +   S4 * /S3 * /S2 *  S1 *  SØ *  CPEN       *  DSEL *  NENE * /REFREQ
          +  /S4       *  S2 *  S1 *  SØ *  LADS * /WR *  DSEL *  NENE * /REFREQ
          +  /S4 * /S3 * /S2 *  S1 *  SØ *  LADS * /WR *  DSEL         * /REFREQ
          *  /EXPBSY   * /ATMBSY
          +  /S4 * /S3 * /S2 *  S1 *  SØ *  CPEN * /WR *  DSEL         * /REFREQ
          *  /EXPBSY   * /ATMBSY
          +   S2 * /S1
          +   S2 * /SØ;
```

```
S3      :=   S4 *  S3       *  S1 *  SØ *  LADS       *  DSEL *  NENE * /REFREQ
         +   S4 * /S3 * /S2 *  S1 *  SØ *  LADS *  WR *  DSEL *  NENE * /REFREQ
         +   S4 * /S3 * /S2 *  S1 *  SØ *  CPEN *  WR *  DSEL *  NENE * /REFREQ
         +  /S4 * /S3 * /S2 *  S1 *  SØ *  LADS *  WR *  DSEL         * /REFREQ
         *  /EXPBSY   * /ATMBSY
         +  /S4 * /S3 * /S2 *  S1 *  SØ *  CPEN *  WR *  DSEL         * /REFREQ
         *  /EXPBSY   * /ATMBSY
         +  /S4 * /S3 * /S2 *  S1 *  SØ *  REFREQ
         +   S3 * /S1
         +   S3 * /SØ;

S4      :=   S4             *  S1 *  SØ *  LADS *  WR *  DSEL *  NENE * /REFREQ
         +   S4 * /S3 *  S2 *  S1 *  SØ
         +   S4 * /S3 * /S2 *  S1 *  SØ *  CPEN *  WR *  DSEL *  NENE * /REFREQ
         +  /S4       *  S2 *  S1 *  SØ
         +  /S4 * /S3 * /S2 *  S1 *  SØ *  LADS *  WR *  DSEL         * /REFREQ
         *  /EXPBSY   * /ATMBSY
         +  /S4 * /S3 * /S2 *  S1 *  SØ *  CPEN *  WR *  DSEL         * /REFREQ
         *  /EXPBSY   * /ATMBSY
         +   S4 * /S1
         +   S4 * /SØ;

DRAMBSY :=   S4 *  S3
         +   S4 * /S3
         +  /S4 *  S3 *  S2
         +  /S4 * /S3 *  S2
         +   CPEN
         +   EXPBSY
         +   ATMBSY;

NA      =    S4 *  S3       *  S1 *  SØ *  LADS * /WR *  NENE * /REFREQ
         +   S4 * /S3 *  S2 *  S1 *  SØ *  LADS * /WR *  NENE * /REFREQ
         +  /S4       *  S2 *  S1 *  SØ *  LADS * /WR *  NENE * /REFREQ
         +  /S4 * /S3 *  S2 * /S1 *  SØ;
ENABLE NA  =  /EXPBSY * /ATMBSY * DRAMBSY;

RDY     =    S4 *  S3       *  S1 *  SØ *  LADS *  WR *  NENE * /REFREQ
         +   S4 *  S3       * /S1 *  SØ
         +   S4 * /S3       *  S1 * /SØ;

ENABLE RDY  =  /EXPBSY * /ATMBSY * DRAMBSY;
end dstat1

" Description: This PLD is the state machine for the DRAM interface to the N1Ø.
" There are 32 states some of which are redundant to minimize the number of
" product terms.  This design supports a non-interleaved memory design using
" Static-column 256K x4 DRAMs for the main memory, and 1M x1 DRAMs for the
" parity bits.
"
```

```
" There are six types of cycles possible:  far read, far write, near read,
" near write, refresh, and idle.  Near read and near write cycles imply memory
" accesses with the same row address as the preceeding cycle.  Refresh request
" has the highest priority and will interrupt near memory cycles.
"
" Read cycles will use full pipelining, write cycles are never pipelined.
" Near read cycles will be zero wait-state when following read cycles, and
" near write cycles will be zero wait-state when following write cycles.
"
" DRAMBSY prevents other units from starting a cycle when the DRAM controller
" is busy.
"
```

```
TITLE     DRAM Control State Machine.
PATTERN   DSTAT1.PDS

; U1Ø4
; This PLD contains the state machine for the DRAM controller.

CHIP DSTAT1 PAL16R6

CLKA WR /NENE /LADS /CPEN /EXPBSY ATMBSY /DSEL REFREQ GND
/OE /NNA /S4 /S3 /S2 /S1 /SO /DRAMBSY /NRDY VCC

EQUATIONS

SO     := S4 * S3        * S1 * SO * LADS * /WR * DSEL * NENE * /REFREQ
        + S4 * /S3 * /S2 * S1 * SO * LADS * /WR * DSEL * NENE * /REFREQ

        + S4 * /S3 * /S2 * S1 * SO * CPEN */WR * DSEL * NENE */REFREQ

        + /S4 * S3 */S2 * S1 * SO
        + /S4 */S3 */S2 * S1 * SO */LADS */CPEN */REFREQ
        + /S4 */S3 */S2 * S1 * SO */DSEL */REFREQ
        + /SO

S1     := S4 * S3      * S1 * SO * LADS * WR * DSEL * NENE * /REFREQ
        + S4 */S3 * S2 * S1 * SO
        + /S4      * S2 * S1 * SO
        + /S4 * /S3 */S2 * S1 * SO */LADS */CPEN */REFREQ
        + /S4 * /S3 */S2 * S1 * SO */DSEL * /REFREQ
        + /S1 * SO
        +  S1 */SO

S2     := S4       * S1 * SO * LADS * /WR * DSEL * NENE * /REFREQ
        + S4 */S3 */S2 * S1 * SO * LADS     * DSEL * NENE * /REFREQ

        + S4 */S3 */S2 * S1 * SO * CPEN     * DSEL * NENE * /REFREQ

        + /S4 * S2 * S1 * SO * LADS * /WR   * DSEL * NENE * /REFREQ

        + /S4 */S3 */S2 * S1 * SO * LADS * /WR * DSEL * /REFREQ
        *  /EXPBSY */ATMBSY
        + /S4 */S3 */S2 * S1 * SO * CPEN */WR * DSEL * /REFREQ
        *  /EXPBSY */ATMBSY
        + S2 */S1
        + S2 */SO
```

```
S3      := S4 * S3       * S1 * S0 * LADS         * DSEL * NENE * /REFREQ
         + S4 * /S3 * /S2 * S1 * S0 * LADS * WR * DSEL * NENE * /REFREQ
         + S4 * /S3 * /S2 * S1 * S0 * CPEN * WR * DSEL * NENE * /REFREQ
         + /S4 * /S3 * /S2 * S1 * S0 * LADS * WR * DSEL * /REFREQ
         *  /EXPBSY * /ATMBSY
         + /S4 * /S3 * /S2 * S1 * S0 * CPEN * WR * DSEL * /REFREQ
         *  /EXPBSY */ATMBSY
         + /S4 * /S3 * /S2 * S1 * S0 * REFREQ
         + S3 * /S1
         + S3 * /S0

S4      := S4         * S1 * S0 * LADS * WR * DSEL * NENE * /REFREQ
         + S4 * /S3 * S2 * S1 * S0
         + S4 * /S3 * /S2 * S1 * S0 * CPEN * WR * DSEL * NENE * /REFREQ
         + /S4 * S2 * S1 * S0
         + /S4 */S3 */S2 * S1 * S0 * LADS * WR * DSEL * /REFREQ
         *  /EXPBSY * /ATMBSY
         + /S4 * /S3 * /S2 * S1 * S0 * CPEN * WR * DSEL * /REFREQ
         *  /EXPBSY* /ATMBSY
         + S4 * /S1
         + S4 * /S0

DRAMBSY := S4 * S3
         + S4 * /S3
         + /S4 * S3 * S2
         + /S4 * /S3 * S2
         + CPEN
         + EXPBSY
         + ATMBSY

NNA     = S4 * S3         * S1 * S0 * LADS * /WR * NENE * /REFREQ
         + S4 * /S3 * S2 * S1 * S0 * LADS * /WR * NENE * /REFREQ
         + /S4       * S2 * S1 * S0 * LADS * /WR * NENE * /REFREQ
         + /S4 * /S3 * S2 * /S1 * S0
NNA.TRST = /EXPBSY * /ATMBSY * DRAMBSY

NRDY    = S4 * S3      * S1 * S0 * LADS * WR * NENE * /REFREQ
         + S4 * S3     * /S1 * S0
         + S4 * /S3     * S1 * /S0
NRDY.TRST = /EXPBSY * /ATMBSY * DRAMBSY

; Description: This PLD is the state machine for the i86Ø DRAM interface.
; There are 32 states some of which are redundant to minimize the number of
; product terms. This design supports a non-interleaved memory design using
; Static-column 256K ×4 DRAMs for the main memory, and 1M ×1 DRAMs for the
; parity bits.
```

```
; There are six types of cycles possible: far read, far write, near read,
; near write, refresh, and idle. Near read and near write cycles imply
; memory accesses with the same row address as the preceeding cycle.
; Refresh request has the highest priority and will interrupt near
; memory cycles.

; Read cycles will use full pipelining, write cycles are never pipelined.
; Near read cycles will be zero wait-state when following read cycles, and
; near write cycles wiil be zero wait-state when following write cycles.

; DRAMBSY prevents other units from starting a cycle when the DRAM controller
; is busy.
;


TITLE     DRAM Control A.
PATTERN   DCTLAØ.PDS

;U98
; Reference Sheet 4 of the Schematics.

CHIP DCTLAO PAL16L8

A31 A23 A22 A21 /SØ /S1 /S2 /S3 /S4 GND
NCO HOLD /HREQ /RASPO /RAS3 /RAS2 /RAS1 /RASØ /KEN VCC

EQUATIONS

RASØ  = A23 * A22 * A21 * S4 * S3 * /S2 * /S1 * /SO      ;start write
      + A23 * A22 * A21 * /S4 * /S3 * S2 * /S1 * /SO     ;start read
      + /S4 * S3 * /S2 * /S1 * SO                        ;start refresh
      + RASO * S4                                        ;hold
      + RASO * S3                                        ;hold
      + RASO * S2                                        ;hold

RAS1  = A23 * A22 * /A21 * S4 * S3 * /S2 * /S1 * /SO     ;start write
      + A23 * A22 * /A21 * /S4 * /S3 * S2 * /S1 * /SO    ;start read
      + /S4 * S3 * /S2 * /S1 * SO                        ;start refresh
      + RAS1 * S4                                        ;hold
      + RAS1 * S3                                        ;hold
      + RAS1 * S2                                        ;hold

RAS2  = A23 * /A22 * A21 * S4 * S3 * /S2 * /S1 * /SO     ;start write
      + A23 * /A22 * A21 * /S4 * /S3 * S2 * /S1 * /SO    ;start read
      + /S4 * S3 * /S2 * /S1 * SØ                        ;start refresh
      + RAS2 * S4                                        ;hold
      + RAS2 * S3                                        ;hold
      + RAS2 * S2                                        ;hold
```

```
RAS3  = A23 * /A22 * /A21 * S4 * S3 */S2 * /S1 * /S0     ;start write
      + A23 * /A22 * /A21 * /S4 * /S3 * S2 * /S1 * /S0   ;start read
      + /S4 * S3 * /S2 * /S1 * S0                        ;start refresh
      + RAS3 * S4                                        ;hold
      + RAS3 * S3                                        ;hold
      + RAS3 * S2                                        ;hold

RASP0 = A23 * S4 * S3 * /S2 * /S1 * /S0                  ;start write
      + A23 * /S4 * /S3 * S2 * /S1 * /S0                 ;start read
      + /S4 * S3 * /S2 * /S1 * S0                        ;start refresh
      + RASPØ * S4                                       ;hold
      + RASPØ * S3                                       ;hold
      + RASPØ * S2                                       ;hold

KEN = A31

/HOLD = /HREQ

;Description:
;    This PLD decodes and drives the RAS lines for the DRAMs from
;    the main state machine and the i86Ø addresses. It drives the RAS
;    lines for the 8 Mbytes on the board.

;    The addresses (A21-A23) are decoded and activated when the
;    DRAM state machine starts a read or a write cycle. They are held
;    active until ths state machine enters the Precharge sequence
;    (State 31). All RAS lines are active during a refresh cycle.
;
;    This PLD must be "D" speed.


TITLE     DRAM Control B.
PATTERN   DCTLB1.PDS

; U99
; This PLD decodes and drives numerous DRAM control signals.
; Must be "D" speed.

CHIP DCTLB1 PAL16R6

DCLK /EWDL /ERDL /S0 /S1 /S2 /S3 /S4 /RTDS GND
/OE /RTAB /WDL /RDL /WEL /CSX /CAE /CAL /RTBA VCC

EQUATIONS
RTAB = RTDS

RTBA = RTDS
```

```
RDL  := /ERDL * /S0                    ;/RDL IS HIGH DURING
     + /ERDL * /S1                     ;STATE 8,16,24
     + /ERDL * /S2                     ;
     + /ERDL * S3 * S4                 ;

WEL  := S4 * S3 * S2 * S1 * /S0        ;state 1
     + S4 * S3 * S2 * /S1 * /S0        ;state 3
     + S4 * S3 * /S2 * S1 * /S0        ;state 5
     + S4 * S3 * /S2 * /S1 * /S0       ;state 7

WDL  := /EWDL * /S0                    ;/WDL IS HIGH DURING
     + /EWDL * /S3                     ;STATE Ø,2,4,6
     + /EWDL * /S4                     ;

CSX  := S4 * S3 * S2 * /S0             ;state 1, 3
     + S4 * S3 * /S2 * S1 * /S0        ;state 5
     + S4 * /S3 * S2 * S1              ;state 8, 9
     + S4 * /S3 * /S2 * S1             ;state 12,13
     + /S4 * S3 * S2 * S1              ;state 16,17
     + /S4 * S3 * /S2 * /S1            ;state 22,23
     + /S4 * /S3 * S2 * S1             ;state 24,25
     + /S4      * S2 * /S1 * S0        ;state 18,26

CAL  := /S0                            ;all odd states.

CAE  := S4 * S3 * S2                   ;states Ø,1,2,3
     + S4 * S3 * /S2 * S1              ;states 4,5
     + S4 * S3 * /S2 * /S1 * S0        ;state 6
     + S4 * /S3      * S1              ;states 8,9,12,13
     + /S4 * S3 * S2 * S1              ;states 16,17
     + /S4 * /S3 * S2 * S1             ;states 24,25
     + /S4     * S2 * /S1 * S0         ;state 18,26

; Description:
;
;
;       CAE Column Address Enable. Turns on the Column address drivers.
;
;       CAL Column Address Latch. Latches the N10 addresses for the DRAM
;           column address.
;
;       CSX Chip Select for the DRAM array.
;
;       WEL Write Enable Latch. Latches the BEN bits to drive the active
;           DRAM WEs.
;
;       WDL Write Data Latch. For DRAM accesses only.
;
;       RDL Read Data Latch. For DRAM accesses only.
;
;
;
```

```
TITLE       DRAM Control C.
PATTERN     DCTLC2.PDS

; U100
; This PLD decodes and drives numerous DRAM control signals.
; Must be "D" speed.

CHIP DCTLC2 PAL16R6

CLKA WR /S0 /S1 /S2 /S3 /S4 /EWDE /ERDE GND
/OE /PERLTCH /DRMEN /DRMDIR /CLRFRQ /WPL /OEX /RAE /CLRCYC VCC

EQUATIONS

PERLTCH = S4 * /S3 * S2 * S1 * /S0      ;state 9
        + S4 * /S3 * /S2 * S1 * /S0     ;state 13

CLRCYC = S4 * S3 *S2 * S1 * /S0         ;state 1
       + S4 * S3 * /S2 * S1 * /S0       ;state 5
       + S4 * /S3 * S2 * S1 * /S0       ;state 9
       + /S4 * S3 * S2 * /S1 * S0       ;state 18
       + /S4 * /S3 * S2 * S1 * /S0      ;state 25

OEX := S4 * S3 * S2 * S1, S0 * /WR      ;state Ø if near read cycle.
       + S4 * S3 * /S2 * S1 * S0 * /WR  ;state 4 if near read cycle.
       + S4 * /S3 * S2 * S1             ;state 8,9
       + /S4 * S3 * S2 * S1             ;state 16,17
       + /S4 * S3 * S2 * /S1 * S0       ;state 18
       + /S4 * /S3 * S2                 ;state 24,25,26,27

RAE  := /S4 * /S3 */S2 * S1             ;state 28,29
       + /S4 * /S3 * /S2 * /S1 * S0     ;state 3Ø

CLRFRQ := S4                            ;ALL EXCEPT state 23
      + /S4 * S3 * S2
      + /S4 * S3 * /S2 * S1
      + /S4 * S3 * /S2 * /S1 * SØ
      + /S4 * /S3

WPL := S4 * S3 * S2 * S1 * /S0          ;state 1
     + S4 * S3 * S2 * /S1 * /S0         ;state 3
     + S4 * S3 * /S2 * S1 * /S0         ;state 5
     + S4 * S3 * /S2 * /S1 * /S0        ;state 7

DRMEN:= S4 * S3      */S1               ;state 2,3,6,7
    + S4 * S3      * S1 * /S0           ;state 1,5
    + S4 * /S3 * S2 * S1                ;state 8,9
    + /S4     * S2 * S1                 ;state 16,17,24,25
    + S4 * S3      * S1 * S0 * WR       ;state Ø,4 if write cycle.
    + EWDE
    + ERDE

DRMDIR := S4 * /S3 * S2 * S1            ;state 8,9
    + S4 * /S3 * /S2 * S1 * /S0         ;state 13
    + /S4      * S2 * /S1 * S0          ;state 18,26
```

```
    + /S4       * S2 * S1                ;state 16,17,24,25
    + ERDE

; Description:
;
;
;       PERLTCH     Parity Error Latch. Latches parity errors (if any)
;                   during DRAM read cycles.
;
;       CLRFRQ      Clear Refresh Request. Restarts the refresh counter.
;
;       WPL         Write Parity Latch. Latches the parity bits during a
;                   DRAM write.
;
;       OEX         Output Enable. For the DRAMS.
;
;       RAE         Row Address Enable. Turns on the row addresses to the
;                   DRAM.
;
;       CLRCYC      Clear Cycle. Clears queued cycles.


TITLE       DRAM Control D.
PATTERN     DCTLDØ.PDS

;This PLD decodes and drives the DRAM Write Enable signals.
;Can be "B" speed.

CHIP DCTLDO PAL2ØR8

/WEL /CPEN /DSEL /WR /BEO /BE1 /BE2 /BE3 /BE4 /BE5 /BE6 GND
/OE /BE7 /WE7 /WE6 /WE5 /WE4 /WE3 /WE2 /WE1 /WEO /LADS VCC

EQUATIONS

WE7    := LADS * DSEL * /WR * BE7
       + CPEN * DSEL * /WR * BE7

WE6    := LADS * DSEL * /WR * BE6
       + CPEN * DSEL * /WR * BE6

WE5    := LADS * DSEL * /WR * BE5
       + CPEN * DSEL * /WR * BE5

WE4    := LADS * DSEL * /WR * BE4
       + CPEN * DSEL * /WR * BE4

WE3    := LADS * DSEL * /WR * BE3
       + CPEN * DSEL * /WR * BE3

WE2    := LADS * DSEL * /WR * BE2
       + CPEN * DSEL * /WR * BE2

WE1    := LADS * DSEL * /WR * BE1
```

```
      + CPEN * DSEL * /WR * BE1

WE0   := LADS * DSEL * /WR * BE0
      + CPEN * DSEL * /WR * BE0

; Description: This PLD samples the BEN lines during write cycles and
;              drives the DRAM Write Enables directly.
;
```

DOMESTIC SALES OFFICES

ALABAMA

†Intel Corp.
5015 Bradford Dr., #2
Huntsville 35805
Tel: (205) 830-4010
FAX: (205) 837-2640

ARIZONA

†Intel Corp.
11225 N. 28th Dr.
Suite D-214
Phoenix 85029
Tel: (602) 869-4980
FAX: (602) 869-4294

Intel Corp.
1161 N. El Dorado Place
Suite 301
Tucson 85715
Tel: (602) 299-6815
FAX: (602) 296-8234

CALIFORNIA

†Intel Corp.
21515 Vanowen Street
Suite 116
Canoga Park 91303
Tel: (818) 704-8500
FAX: (818) 340-1144

†Intel Corp.
2250 E. Imperial Highway
Suite 218
El Segundo 90245
Tel: (213) 640-6040
FAX: (213) 640-7133

Intel Corp.
1510 Arden Way
Suite 101
Sacramento 95815
Tel: (916) 920-8096
FAX: (916) 920-8253

†Intel Corp.
9665 Chesapeake Dr.
Suite 325
San Diego 95123
Tel: (619) 292-8086
FAX: (619) 292-0628

†Intel Corp.*
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114
FAX: (714) 541-9157

†Intel Corp.*
San Tomas 4
2700 San Tomas Expressway
2nd Floor
Santa Clara 95051
Tel: (408) 986-8086
TWX: 910-338-0255
FAX: (408) 727-2620

COLORADO

Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (719) 594-6622
FAX: (303) 594-0720

†Intel Corp.*
650 S. Cherry St.
Suite 915
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289
FAX: (303) 322-8670

CONNECTICUT

†Intel Corp.
301 Lee Farm Corporate Park
83 Wooster Heights Rd.
Danbury 06810
Tel: (203) 748-3130
FAX: (203) 794-0339

FLORIDA

†Intel Corp.
6363 N.W. 6th Way
Suite 100
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407
FAX: (305) 772-8193

†Intel Corp.
5850 T.G. Lee Blvd.
Suite 340
Orlando 32822
Tel: (407) 240-8000
FAX: (407) 240-8097

Intel Corp.
11300 4th Street North
Suite 170
St. Petersburg 33716
Tel: (813) 577-2413
FAX: (813) 578-1607

GEORGIA

Intel Corp.
20 Technology Parkway, N.W.
Suite 150
Norcross 30092
Tel: (404) 449-0541
FAX: (404) 605-9762

ILLINOIS

†Intel Corp.*
300 N. Martingale Road
Suite 400
Schaumburg 60173
Tel: (312) 605-8031
FAX: (312) 706-9762

INDIANA

†Intel Corp.
8777 Purdue Road
Suite 125
Indianapolis 46268
Tel: (317) 875-0623
FAX: (317) 875-8938

IOWA

Intel Corp.
1930 St. Andrews Drive N.E.
2nd Floor
Cedar Rapids 52402
Tel: (319) 393-1294

KANSAS

†Intel Corp.
10985 Cody St.
Suite 140, Bldg. D
Overland Park 66210
Tel: (913) 345-2727
FAX: (913) 345-2076

MARYLAND

†Intel Corp.*
10010 Junction Dr.
Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860
FAX: (301) 206-3677
(301) 206-3678

MASSACHUSETTS

†Intel Corp.*
Westford Corp. Center
3 Carlisle Road
2nd Floor
Westford 01886
Tel: (508) 692-3222
TWX: 710-343-6333
FAX: (508) 692-7867

MICHIGAN

†Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48322
Tel: (313) 851-8096
FAX: (313) 851-8770

MINNESOTA

†Intel Corp.
3500 W. 80th St.
Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867
FAX: (612) 831-6497

MISSOURI

†Intel Corp.
4203 Earth City Expressway
Suite 131
Earth City 63045
Tel: (314) 291-1990
FAX: (314) 291-4341

NEW JERSEY

†Intel Corp.*
Parkway 109 Office Center
328 Newman Springs Road
Red Bank 07701
Tel: (201) 747-2233
FAX: (201) 747-0983

†Intel Corp.
280 Corporate Center
75 Livingston Avenue
First Floor
Roseland 07068
Tel: (201) 740-0111
FAX: (201) 740-0626

NEW YORK

Intel Corp.*
850 Cross Keys Office Park
Fairport 14450
Tel: (716) 425-2750
TWX: 510-253-7391
FAX: (716) 223-2561

†Intel Corp.*
2950 Expressway Dr., South
Suite 130
Islandia 11722
Tel: (516) 231-3300
TWX: 510-227-6236
FAX: (516) 348-7939

†Intel Corp.
Westage Business Center
Bldg. 300, Route 9
Fishkill 12524
Tel: (914) 897-3860
FAX: (914) 897-3125

NORTH CAROLINA

†Intel Corp.
5800 Executive Center Dr.
Suite 105
Charlotte 28212
Tel: (704) 568-8966
FAX: (704) 535-2236

Intel Corp.
5540 Centerview Dr.
Suite 215
Raleigh 27606
Tel: (919) 851-9537
FAX: (919) 851-8974

OHIO

†Intel Corp.*
3401 Park Center Drive
Suite 220
Dayton 45414
Tel: (513) 890-5350
TWX: 810-450-2528
FAX: (513) 890-8658

†Intel Corp.*
25700 Science Park Dr.
Suite 100
Beachwood 44122
Tel: (216) 464-2736
TWX: 810-427-9298
FAX: (804) 282-0673

OKLAHOMA

Intel Corp.
6801 N. Broadway
Suite 115
Oklahoma City 73162
Tel: (405) 848-8086
FAX: (405) 840-9819

OREGON

†Intel Corp.
15254 N.W. Greenbrier Parkway
Building B
Beaverton 97005
Tel: (503) 645-8051
TWX: 910-467-8741
FAX: (503) 645-8181

PENNSYLVANIA

†Intel Corp.*
455 Pennsylvania Avenue
Suite 230
Fort Washington 19034
Tel: (215) 641-1000
TWX: 510-661-2077
FAX: (215) 641-0785

†Intel Corp.*
400 Penn Center Blvd.
Suite 610
Pittsburgh 15235
Tel: (412) 823-4970
FAX: (412) 829-7578

PUERTO RICO

†Intel Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

TEXAS

Intel Corp.
8911 Capital of Texas Hwy.
Austin 78759
Tel: (512) 794-8086
FAX: (512) 338-9335

†Intel Corp.*
12000 Ford Road
Suite 400
Dallas 75234
Tel: (214) 241-8087
FAX: (214) 484-1180

†Intel Corp.*
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490
FAX: (713) 988-3660

UTAH

†Intel Corp.
428 East 6400 South
Suite 104
Murray 84107
Tel: (801) 263-8051
FAX: (801) 268-1457

VIRGINIA

†Intel Corp.
1504 Santa Rosa Road
Suite 108
Richmond 23288
Tel: (804) 282-5668
FAX: (216) 464-2270

WASHINGTON

†Intel Corp.
155 108th Avenue N.E.
Suite 386
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002
FAX: (206) 451-9556

Intel Corp.
408 N. Mullan Road
Suite 102
Spokane 99206
Tel: (509) 928-8086
FAX: (509) 928-9467

WISCONSIN

Intel Corp.
330 S. Executive Dr.
Suite 102
Brookfield 53005
Tel: (414) 784-8087
FAX: (414) 796-2115

CANADA

BRITISH COLUMBIA

Intel Semiconductor of
Canada, Ltd.
4585 Canada Way
Suite 202
Burnaby V5G 4L6
Tel: (604) 298-0387
FAX: (604) 298-8234

ONTARIO

†Intel Semiconductor of
Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Tel: (613) 829-9714
FAX: (613) 820-5936

†Intel Semiconductor of
Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel: (416) 675-2105
FAX: (416) 675-2438

QUEBEC

Intel Semiconductor of
Canada, Ltd.
620 St. Jean Boulevard
Pointe Claire H9R 3K2
Tel: (514) 694-9130
FAX: 514-694-0064

†Sales and Service Office
*Field Application Location

DOMESTIC DISTRIBUTORS

ALABAMA

Arrow Electronics, Inc.
1015 Henderson Road
Huntsville 35805
Tel: (205) 837-6955

†Hamilton/Avnet Electronics
4940 Research Drive
Huntsville 35805
Tel: (205) 837-7210
TWX: 810-726-2162

Pioneer/Technologies Group, Inc.
4825 University Square
Huntsville 35805
Tel: (205) 837-9300
TWX: 810-726-2197

ARIZONA

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 231-5140
TWX: 910-950-0077

Hamilton/Avnet Electronics
30 South McKiemy
Chandler 85226
Tel: (602) 961-6669
TWX: 910-950-0077

Arrow Electronics, Inc.
4134 E. Wood Street
Phoenix 85040
Tel: (602) 437-0750
TWX: 910-951-1550

Wyle Distribution Group
17855 N. Black Canyon Hwy.
Phoenix 85023
Tel: (602) 249-2232
TWX: 910-951-4282

CALIFORNIA

Arrow Electronics, Inc.
10824 Hope Street
Cypress 90630
Tel: (714) 220-6300

Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (213) 701-7500
TWX: 910-493-2086

†Arow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94086
Tel: (408) 745-6600
TWX: 910-339-9371

Arrow Electronics, Inc.
9511 Ridgehaven Court
San Diego 92123
Tel: (619) 565-4800
TWX: 888-064

†Arrow Electronics, Inc.
2961 Dow Avenue
Tustin 92680
Tel: (714) 838-5422
TWX: 910-595-2860

†Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel: (714) 754-6071
TWX: 910-595-1928

†Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94086
Tel: (408) 743-3300
TWX: 910-339-9332

†Hamilton/Avnet Electronics
4545 Ridgeview Avenue
San Diego 92123
Tel: (619) 571-7500
TWX: 910-595-2638

†Hamilton/Avnet Electronics
9650 Desoto Avenue
Chatsworth 91311
Tel: (818) 700-1161

†Hamilton Electro Sales
10950 W. Washington Blvd.
Culver City 20230
Tel: (213) 558-2458
TWX: 910-340-6364

Hamilton Electro Sales
1361B West 190th Street
Gardena 90248
Tel: (213) 217-6700

†Hamilton/Avnet Electronics
3002 'G' Street
Ontario 91761
Tel: (714) 989-9411

†Avnet Electronics
20501 Plummer
Chatsworth 91351
Tel: (213) 700-6271
TWX: 910-494-2207

†Hamilton Electro Sales
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4150
TWX: 910-595-2638

†Hamilton/Avnet Electronics
4103 Northgate Blvd.
Sacramento 95834
Tel: (916) 920-3150

Wyle Distribution Group
124 Maryland Street
El Segundo 90254
Tel: (213) 322-8100

Wyle Distribution Group
7382 Lampson Ave.
Garden Grove 92641
Tel: (714) 891-1717
TWX: 910-348-7140 or 7111

Wyle Distribution Group
11151 Sun Center Drive
Rancho Cordova 95670
Tel: (916) 638-5282

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (619) 565-9171
TWX: 910-335-1590

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
TWX: 910-338-0296

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 863-9953
TWX: 910-595-1572

Wyle Distribution Group
26677 W. Agoura Rd.
Calabasas 91302
Tel: (818) 880-9000
TWX: 372-0232

COLORADO

Arrow Electronics, Inc.
7060 South Tucson Way
Englewood 80112
Tel: (303) 790-4444

†Hamilton/Avnet Electronics
8765 E. Orchard Road
Suite 708
Englewood 80111
Tel: (303) 740-1017
TWX: 910-935-0787

†Wyle Distribution Group
451 E. 124th Avenue
Thornton 80241
Tel: (303) 457-9953
TWX: 910-936-0770

CONNECTICUT

†Arrow Electronics, Inc.
12 Beaumont Road
Wallingford 06492
Tel: (203) 265-7741
TWX: 710-476-0162

Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
TWX: 710-456-9974

†Pioneer Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
TWX: 710-468-3373

FLORIDA

†Arrow Electronics, Inc.
400 Fairway Drive
Suite 102
Deerfield Beach 33441
Tel: (305) 429-8200
TWX: 510-955-9456

Arrow Electronics, Inc.
37 Skyline Drive
Suite 3101
Lake Marv 32746
Tel: (407) 323-0252
TWX: 510-959-6337

†Hamilton/Avnet Electronics
6801 N.W. 15th Way
Ft. Lauderdale 33309
Tel: (305) 971-2900
TWX: 510-956-3097

†Hamilton/Avnet Electronics
3197 Tech Drive North
St. Petersburg 33702
Tel: (813) 576-3930
TWX: 810-863-0374

†Hamilton/Avnet Electronics
6947 University Boulevard
Winter Park 32792
Tel: (305) 628-3888
TWX: 810-853-0322

†Pioneer/Technologies Group, Inc.
337 S. Lake Blvd.
Alta Monte Springs 32701
Tel: (407) 834-9090
TWX: 810-853-0284

Pioneer/Technologies Group, Inc.
674 S. Military Trail
Deerfield Beach 33442
Tel: (305) 428-8877
TWX: 510-955-9653

GEORGIA

†Arrow Electronics, Inc.
3155 Northwoods Parkway
Suite A
Norcross 30071
Tel: (404) 449-8252
TWX: 810-766-0439

†Hamilton/Avnet Electronics
5825 D Peachtree Corners
Norcross 30092
Tel: (404) 447-7500
TWX: 810-766-0432

Pioneer/Technologies Group, Inc.
3100 F Northwoods Place
Norcross 30071
Tel: (404) 448-1711
TWX: 810-766-4515

ILLINOIS

Arrow Electronics, Inc.
1140 W. Thorndale
Itasca 60143
Tel: (312) 250-0500
TWX: 312-250-0916

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel: (312) 860-7780
TWX: 910-227-0060

MTI Systems Sales
1100 W. Thorndale
Itasca 60143
Tel: (312) 773-2300

†Pioneer Electronics
1551 Carmen Drive
Elk Grove Village 60007
Tel: (312) 437-9680
TWX: 910-222-1834

INDIANA

†Arrow Electronics, Inc.
2495 Directors Row, Suite H
Indianapolis 46241
Tel: (317) 243-9353
TWX: 810-341-3119

Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
TWX: 810-260-3966

†Pioneer Electronics
6408 Castleplace Drive
Indianapolis 46250
Tel: (317) 849-7300
TWX: 810-260-1794

IOWA

Hamilton/Avnet Electronics
915 33rd Avenue, S.W.
Cedar Rapids 52404
Tel: (319) 362-4757

KANSAS

Arrow Electronics
8208 Melrose Dr., Suite 210
Lenexa 66214
Tel: (913) 541-9542

†Hamilton/Avnet Electronics
9219 Quivera Road
Overland Park 66215
Tel: (913) 888-8900
TWX: 910-743-0005

Pioneer/Tec Gr.
10551 Lockman Rd.
Lenexa 66215
Tel: (913) 492-0500

KENTUCKY

Hamilton/Avnet Electronics
1051 D. Newton Park
Lexington 40511
Tel: (606) 259-1475

MARYLAND

Arrow Electronics, Inc.
8300 Guilford Drive
Suite H, River Center
Columbia 21046
Tel: (301) 995-0003
TWX: 710-236-9005

Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel: (301) 995-3500
TWX: 710-862-1861

†Mesa Technology Corp.
9720 Patuxent Woods Dr.
Columbia 21046
Tel: (301) 290-8150
TWX: 710-828-9702

†Pioneer/Technologies Group, Inc.
9100 Gaither Road
Gaithersburg 20877
Tel: (301) 921-0660
TWX: 710-828-0545

Arrow Electronics, Inc.
7524 Standish Place
Rockville 20855
Tel: 301-424-0244

MASSACHUSETTS

Arrow Electronics, Inc.
25 Upton Dr.
Wilmington 01887
Tel: (617) 935-5134

†Hamilton/Avnet Electronics
10D Centennial Drive
Peabody 01960
Tel: (617) 531-7430
TWX: 710-393-0382

MTI Systems Sales
83 Cambridge St.
Burlington 01813

Pioneer Electronics
44 Hartwell Avenue
Lexington 02173
Tel: (617) 861-9200
TWX: 710-326-6617

MICHIGAN

Arrow Electronics, Inc.
755 Phoenix Drive
Ann Arbor 48104
Tel: (313) 971-8220
TWX: 810-223-6020

Hamilton/Avnet Electronics
2215 29th Street S.E.
Space A5
Grand Rapids 49508
Tel: (616) 243-8805
TWX: 810-274-6921

Pioneer Electronics
4504 Broadmoor S.E.
Grand Rapids 49508
FAX: 616-698-1831

†Hamilton/Avnet Electronics
32487 Schoolcraft Road
Livonia 48150
Tel: (313) 522-4700
TWX: 810-282-8775

†Pioneer/Michigan
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
TWX: 810-242-3271

MINNESOTA

†Arrow Electronics, Inc.
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
TWX: 910-576-3125

†Hamilton/Avnet Electronics
12400 Whitewater Drive
Minnetonka 55434
Tel: (612) 932-0600

†Pioneer Electronics
7625 Golden Triange Dr.
Suite G
Eden Prairi 55343
Tel: (612) 944-3355

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63141
Tel: (314) 567-6888
TWX: 910-764-0882

†Hamilton/Avnet Electronics
13743 Shoreline Court
Earth City 63045
Tel: (314) 344-1200
TWX: 910-762-0684

NEW HAMPSHIRE

†Arrow Electronics, Inc.
3 Perimeter Road
Manchester 03103
Tel: (603) 668-6968
TWX: 710-220-1684

†Hamilton/Avnet Electronics
444 E. Industrial Drive
Manchester 03103
Tel: (603) 624-9400

†Microcomputer System Technical Distributor Center

DOMESTIC DISTRIBUTORS (Contd.)

NEW JERSEY

†Arrow Electronics, Inc.
Four East Stow Road
Unit 11
Marlton 08053
Tel: (609) 596-8000
TWX: 710-897-0829

†Arrow Electronics
6 Century Drive
Parsipanny 07054
Tel: (201) 538-0900

†Hamilton/Avnet Electronics
1 Keystone Ave., Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0110
TWX: 710-940-0262

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-5300
TWX: 710-734-4388

†MTI Systems Sales
37 Kulick Rd.
Fairfield 07006
Tel: (201) 227-5552

†Pioneer Electronics
45 Route 46
Pinebrook 07058
Tel: (201) 575-3510
TWX: 710-734-4382

NEW MEXICO

Alliance Electronics Inc.
11030 Cochiti S.E.
Albuquerque 87123
Tel: (505) 292-3360
TWX: 910-989-1151

Hamilton/Avnet Electronics
2524 Baylor Drive S.E.
Albuquerque 87106
Tel: (505) 765-1500
TWX: 910-989-0614

NEW YORK

†Arrow Electronics, Inc.
3375 Brighton Henrietta
Townline Rd.
Rochester 14623
Tel: (716) 275-0300
TWX: 510-253-4766

Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
TWX: 510-227-6623

Hamilton/Avnet
933 Motor Parkway
Hauppauge 11788
Tel: (516) 231-9800
TWX: 510-224-6166

†Hamilton/Avnet Electronics
333 Metro Park
Rochester 14623
Tel: (716) 475-9130
TWX: 510-253-5470

†Hamilton/Avnet Electronics
103 Twin Oaks Drive
Syracuse 13206
Tel: (315) 437-0288
TWX: 710-541-1560

†MTI Systems Sales
38 Harbor Park Drive
Port Washington 11050
Tel: (516) 621-6200

†Pioneer Electronics
68 Corporate Drive
Binghamton 13904
Tel: (607) 722-9300
TWX: 510-252-0893

Pioneer Electronics
40 Oser Avenue
Hauppauge 11787
Tel: (516) 231-9200

†Pioneer Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
TWX: 510-221-2184

†Pioneer Electronics
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
TWX: 510-253-7001

NORTH CAROLINA

†Arrow Electronics, Inc.
5240 Greensdairy Road
Raleigh 27604
Tel: (919) 876-3132
TWX: 510-928-1856

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27604
Tel: (919) 878-0819
TWX: 510-928-1836

Pioneer/Technologies Group, Inc.
9801 A-Southern Pine Blvd.
Charlotte 28210
Tel: (919) 527-8188
TWX: 810-621-0366

OHIO

Arrow Electronics, Inc.
7620 McEwen Road
Centerville 45459
Tel: (513) 435-5563
TWX: 810-459-1611

†Arrow Electronics, Inc.
6238 Cochran Road
Solon 44139
Tel: (216) 248-3990
TWX: 810-427-9409

†Hamilton/Avnet Electronics
954 Senate Drive
Dayton 45459
Tel: (513) 439-6733
TWX: 810-450-2531

Hamilton/Avnet Electronics
4588 Emery Industrial Pkwy.
Warrensville Heights 44128
Tel: (216) 349-5100
TWX: 810-427-9452

†Hamilton/Avnet Electronics
777 Brooksedge Blvd.
Westerville 43081
Tel: (614) 882-7004

†Pioneer Electronics
4433 Interpoint Boulevard
Dayton 45424
Tel: (513) 236-9900
TWX: 810-459-1622

†Pioneer Electronics
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
TWX: 810-422-2211

OKLAHOMA

Arrow Electronics, Inc.
1211 E. 51st St., Suite 101
Tulsa 74146
Tel: (918) 252-7537

†Hamilton/Avnet Electronics
12121 E. 51st St., Suite 102A
Tulsa 74146
Tel: (918) 252-7297

OREGON

†Almac Electronics Corp.
1885 N.W. 169th Place
Beaverton 97005
Tel: (503) 629-8090
TWX: 910-467-8746

†Hamilton/Avnet Electronics
6024 S.W. Jean Road
Bldg. C, Suite 10
Lake Oswego 97034
Tel: (503) 635-7848
TWX: 910-455-8179

Wyle Distribution Group
5250 N.E. Elam Young Parkway
Suite 600
Hillsboro 97124
Tel: (503) 640-6000
TWX: 910-460-2203

PENNSYLVANIA

Arrow Electronics, Inc.
650 Seco Road
Monroeville 15146
Tel: (412) 856-7000

Hamilton/Avnet Electronics
2800 Liberty Ave.
Pittsburgh 15238
Tel: (412) 281-4150

Pioneer Electronics
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
TWX: 710-795-3122

†Pioneer/Technologies Group, Inc.
Delaware Valley
261 Gibralter Road
Horsham 19044
Tel: (215) 674-4000
TWX: 510-665-6778

TEXAS

†Arrow Electronics, Inc.
3220 Commander Drive
Carrollton 75006
Tel: (214) 380-6464
TWX: 910-860-5377

†Arrow Electronics, Inc.
10899 Kinghurst
Suite 100
Houston 77099
Tel: (713) 530-4700
TWX: 910-880-4439

†Arrow Electronics, Inc.
2227 W. Braker Lane
Austin 78758
Tel: (512) 835-4180
TWX: 910-874-1348

†Hamilton/Avnet Electronics
1807 W. Braker Lane
Austin 78758
Tel: (512) 837-8911
TWX: 910-874-1319

†Hamilton/Avnet Electronics
2111 W. Walnut Hill Lane
Irving 75038
Tel: (214) 550-6111
TWX: 910-860-5929

†Hamilton/Avnet Electronics
4850 Wright Rd., Suite 190
Stafford 77477
Tel: (713) 240-7733
TWX: 910-881-5523

†Pioneer Electronics
18260 Kramer
Austin 78758
Tel: (512) 835-4000
TWX: 910-874-1323

†Pioneer Electronics
13710 Omega Road
Dallas 75234
Tel: (214) 386-7300
TWX: 910-850-5563

†Pioneer Electronics
5853 Point West Drive
Houston 77036
Tel: (713) 988-5555
TWX: 910-881-1606

Wyle Distribution Group
1810 Greenville Avenue
Richardson 75081
Tel: (214) 235-9953

UTAH

Arrow Electronics
1946 Parkway Blvd.
Salt Lake City 84119
Tel: (801) 973-6913

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
TWX: 910-925-4018

Wyle Distribution Group
1325 West 2200 South
Suite E
West Valley 84119
Tel: (801) 974-9953

WASHINGTON

†Almac Electronics Corp.
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
TWX: 910-444-2067

Arrow Electronics, Inc.
19540 68th Ave. South
Kent 98032
Tel: (206) 575-4420

†Hamilton/Avnet Electronics
14212 N.E. 21st Street
Bellevue 98005
Tel: (206) 643-3950
TWX: 910-443-2469

Wyle Distribution Group
15385 N.E. 90th Street
Redmond 98052
Tel: (206) 881-1150

WISCONSIN

Arrow Electronics, Inc.
200 N. Patrick Blvd., Ste. 100
Brookfield 53005
Tel: (414) 767-6600
TWX: 910-262-1193

Hamilton/Avnet Electronics
2975 Moorland Road
New Berlin 53151
Tel: (414) 784-4510
TWX: 910-262-1182

CANADA

ALBERTA

Hamilton/Avnet Electronics
2816 21st Street N.E.
Calgary T2E 6Z3
Tel: (403) 230-3586
TWX: 03-827-642

Zentronics
Bay No. 1
3300 14th Avenue N.E.
Calgary T2A 6J4
Tel: (403) 272-1021

BRITISH COLUMBIA

†Hamilton/Avnet Electronics
105-2550 Boundary
Burmalay V5M 3Z3
Tel: (604) 437-6667

Zentronics
108-11400 Bridgeport Road
Richmond V6X 1T2
Tel: (604) 273-5575
TWX: 04-5077-89

MANITOBA

Zentronics
60-1313 Border Unit 60
Winnipeg R3H 0X4
Tel: (204) 694-1957

ONTARIO

Arrow Electronics, Inc.
36 Antares Dr.
Nepean K2E 7W5
Tel: (613) 226-6903

Arrow Electronics, Inc.
1093 Meyerside
Mississauga L5T 1M4
Tel: (416) 673-7769
TWX: 06-218213

†Hamilton/Avnet Electronics
6845 Rexwood Road
Units 3-4-5
Mississauga L4T 1R2
Tel: (416) 677-7432
TWX: 610-492-8867

Hamilton/Avnet Electronics
6845 Rexwood Rd., Unit 6
Mississauga L4T 1R2
Tel: (416) 277-0484

†Hamilton/Avnet Electronics
190 Colonnade Road South
Nepean K2E 7L5
Tel: (613) 226-1700
TWX: 05-349-71

†Zentronics
8 Tilbury Court
Brampton L6T 3T4
Tel: (416) 451-9600
TWX: 06-976-78

†Zentronics
155 Colonnade Road
Unit 17
Nepean K2E 7K1
Tel: (613) 226-8840

Zentronics
60-1313 Border St.
Winnipeg R3H 0l4
Tel: (204) 694-7957

QUEBEC

†Arrow Electronics Inc.
4050 Jean Talon Quest
Montreal H4P 1W1
Tel: (514) 735-5511
TWX: 05-25590

Arrow Electronics, Inc.
500 Avenue St-Jean Baptiste
Suite 280
Quebec G2E 5R9
Tel: (418) 871-7500
FAX: 418-871-6816

Hamilton/Avnet Electronics
2795 Halpern
St. Laurent H2E 7K1
Tel: (514) 335-1000
TWX: 610-421-3731

Zentronics
817 McCaffrey
St. Laurent H4T 1M3
Tel: (514) 737-9700
TWX: 05-827-535

†Microcomputer System Technical Distributor Center

EUROPEAN SALES OFFICES

DENMARK

Intel Denmark A/S
Glentevej 61, 3rd Floor
2400 Copenhagen NV
Tel: (45) (31) 19 80 33
TLX: 19567

FINLAND

Intel Finland OY
Ruosilantie 2
00390 Helsinki
Tel: (358) 0 544 644
TLX: 123332

FRANCE

Intel Corporation S.A.R.L.
1, Rue Edison-BP 303
78054 St. Quentin-en-Yvelines
Cedex
Tel: (33) (1) 30 57 70 00
TLX: 699016

WEST GERMANY

Intel Semiconductor GmbH*
Dornacher Strasse 1
8016 Feldkirchen bei Muenchen
Tel: (49) 089/90992-0
TLX: 5-23177

Intel Semiconductor GmbH
Hohenzollern Strasse 5
3000 Hannover 1
Tel: (49) 0511/344081
TLX: 9-23625

Intel Semiconductor GmbH
Abraham Lincoln Strasse 16-18
6200 Wiesbaden
Tel: (49) 06121/7605-0
TLX: 4-186183

Intel Semiconductor GmbH
Zettachring 10A
7000 Stuttgart 80
Tel: (49) 0711/7287-280
TLX: 7-254826

ISRAEL

Intel Semiconductor Ltd.*
Atidim Industrial Park-Neve Sharet
P.O. Box 43202
Tel-Aviv 61430
Tel: (972) 03-498080
TLX: 371215

ITALY

Intel Corporation Italia S.p.A.*
Milanofiori Palazzo E
20090 Assago
Milano
Tel: (39) (02) 89200950
TLX: 341286

NETHERLANDS

Intel Semiconductor B.V.*
Postbus 84130
3099 CC Rotterdam
Tel: (31) 10.407.11.11
TLX: 22283

NORWAY

Intel Norway A/S
Hvamveien 4-PO Box 92
2013 Skjetten
Tel: (47) (6) 842 420
TLX: 78018

SPAIN

Intel Iberia S.A.
Zurbaran, 28
28010 Madrid
Tel: (34) (1) 308.25.52
TLX: 46880

SWEDEN

Intel Sweden A.B.*
Dalvagen 24
171 36 Solna
Tel: (46) 8 734 01 00
TLX: 12261

SWITZERLAND

Intel Semiconductor A.G.
Zuerichstrasse
8185 Winkel-Rueti bei Zuerich
Tel: (41) 01/860 62 62
TLX: 825977

UNITED KINGDOM

Intel Corporation (U.K.) Ltd.*
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel: (44) (0793) 696000
TLX: 444447/8

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA

Bacher Electronics G.m.b.H.
Rotenmuehlgasse 26
1120 Wien
Tel: (43) (0222) 83 56 46
TLX: 31532

BELGIUM

Inelco Belgium S.A.
Av. des Croix de Guerre 94
1120 Bruxelles
Oorlogskruisenlaan, 94
1120 Brussel
Tel: (32) (02) 216 01 60
TLX: 64475 or 22090

DENMARK

ITT-Multikomponent
Naverland 29
2600 Glostrup
Tel: (45) (0) 2 45 66 45
TLX: 33 355

FINLAND

OY Fintronic AB
Melkonkatu 24A
00210 Helsinki
Tel: (358) (0) 6926022
TLX: 124224

FRANCE

Almex
Zone industrielle d'Antony
48, rue de l'Aubepine
BP 102
92164 Antony cedex
Tel: (33) (1) 46 66 21 12
TLX: 250067

Jermyn-Generim
60, rue des Gemeaux
Silic 580
94653 Rungis cedex
Tel: (33) (1) 49 78 49 78
TLX: 261585

Metrologie
Tour d'Asnieres
4, av. Laurent-Cely
92606 Asnieres Cedex
Tel: (33) (1) 47 90 62 40
TLX: 611448

Tekelec-Airtronic
Cite des Bruyeres
Rue Carle Vernet - BP 2
92310 Sevres
Tel: (33) (1) 45 34 75 35
TLX: 204552

WEST GERMANY

Electronic 2000 AG
Stahlgruberring 12
8000 Muenchen 82
Tel: (49) 089/42001-0
TLX: 522561

ITT Multikomponent GmbH
Postfach 1265
Bahnhofstrasse 44
7141 Moeglingen
Tel: (49) 07141/4879
TLX: 7264472

Jermyn GmbH
Im Dachsstueck 9
6250 Limburg
Tel: (49) 06431/508-0
TLX: 415257-0

Metrologie GmbH
Meglingerstrasse 49
8000 Muenchen 71
Tel: (49) 089/78042-0
TLX: 5213189

Proelectron Vertriebs GmbH
Max Planck Strasse 1-3
6072 Dreieich
Tel: (49) 06103/30434-3
TLX: 417903

IRELAND

Micro Marketing Ltd.
Glenageary Office Park
Glenageary
Co. Dublin
Tel: (21) (353) (01) 85 63 25
TLX: 31584

ISRAEL

Eastronics Ltd.
11 Rozanis Street
P.O.B. 39300
Tel-Aviv 61392
Tel: (972) 03-475151
TLX: 33638

ITALY

Intesi
Divisione ITT Industries GmbH
Viale Milanofiori
Palazzo E/5
20090 Assago (MI)
Tel: (39) 02/824701
TLX: 311351

Lasi Elettronica S.p.A.
V. le Fulvio Testi, 126
20092 Cinisello Balsamo (MI)
Tel: (39) 02/2440012
TLX: 352040

Telcom S.r.l.
Via M. Civitali 75
20148 Milano
Tel: (39) 02/4049046
TLX: 335654

ITT Multicomponents
Viale Milanofiori E/5
20090 Assago (MI)
Tel: (39) 02/824701
TLX: 311351

Silverstar
Via Dei Gracchi 20
20146 Milano
Tel: (39) 02/49961
TLX: 332189

NETHERLANDS

Koning en Hartman Elektrotechniek B.V.
Energieweg 1
2627 AP Delft
Tel: (31) (0) 15/609906
TLX: 38250

NORWAY

Nordisk Elektronikk (Norge) A/S
Postboks 123
Smedsvingen 4
1364 Hvalstad
Tel: (47) (02) 84 62 10
TLX: 77546

PORTUGAL

ATD Portugal LDA
Rua Dos Lusiados, 5 Sala B
1300 Lisboa
Tel: (35) (1) 64 80 91
TLX: 61562

Ditram
Avenida Miguel Bombarda, 133
1000 Lisboa
Tel: (35) (1) 54 53 13
TLX: 14182

SPAIN

ATD Electronica, S.A.
Plaza Ciudad de Viena, 6
28040 Madrid
Tel: (34) (1) 234 40 00
TLX: 42477

ITT-SESA
Calle Miguel Angel, 21-3
28010 Madrid
Tel: (34) (1) 419 09 57
TLX: 27461

Metrologia Iberica, S.A.
Ctra. de Fuencarral, n.80
28100 Alcobendas (Madrid)
Tel: (34) (1) 653 86 11

SWEDEN

Nordisk Elektronik AB
Torshamnsgatan 39
Box 36
164 93 Kista
Tel: (46) 08-03 46 30
TLX: 105 47

SWITZERLAND

Industrade A.G.
Hertistrasse 31
8304 Wallisellen
Tel: (41) (01) 8328111
TLX: 56788

TURKEY

EMPA Electronic
Lindwurmstrasse 95A
8000 Muenchen 2
Tel: (49) 089/53 80 570
TLX: 528573

UNITED KINGDOM

Accent Electronic Components Ltd.
Jubilee House, Jubilee Road
Letchworth, Herts SG6 1TL
Tel: (44) (0462) 686666
TLX: 826293

Bytech-Comway Systems
3 The Western Centre
Western Road
Bracknell RG12 1RW
Tel: (44) (0344) 55333
TLX: 847201

Jermyn
Vestry Estate
Otford Road
Sevenoaks
Kent TN14 5EU
Tel: (44) (0732) 450144
TLX: 95142

MMD
Unit 8 Southview Park
Caversham
Reading
Berkshire RG4 0AF
Tel: (44) (0734) 481666
TLX: 846669

Rapid Silicon
Rapid House
Denmark Street
High Wycombe
Buckinghamshire HP11 2ER
Tel: (44) (0494) 442266
TLX: 837931

Rapid Systems
Rapid House
Denmark Street
High Wycombe
Buckinghamshire HP11 2ER
Tel: (44) (0494) 450244
TLX: 837931

YUGOSLAVIA

H.R. Microelectronics Corp.
2005 de la Cruz Blvd., Ste. 223
Santa Clara, CA 95050
U.S.A.
Tel: (1) (408) 988-0286
TLX: 387452

Rapido Electronic Components S.p.a.
Via C. Beccaria, 8
34133 Trieste
Italia
Tel: (39) 040/360555
TLX: 460461

*Field Application Location

INTERNATIONAL SALES OFFICES

AUSTRALIA

Intel Australia Pty. Ltd.*
Spectrum Building
200 Pacific Hwy., Level 6
Crows Nest, NSE, 2065
Tel: 612-957-2744
FAX: 612-923-2632

BRAZIL

Intel Semicondutores do Brazil LTDA
Av. Paulista, 1159-CJS 404/405
01311 - Sao Paulo - S.P.
Tel: 55-11-287-5899
TLX: 3911153146 ISDB
FAX: 55-11-287-5119

CHINA/HONG KONG

Intel PRC Corporation
15/F, Office 1, Citic Bldg.
Jian Guo Men Wai Street
Beijing, PRC
Tel: (1) 500-4850
TLX: 22947 INTEL CN
FAX: (1) 500-2953

Intel Semiconductor Ltd.*
10/F East Tower
Bond Center
Queensway, Central
Hong Kong
Tel: (5) 8444-555
TLX: 63869 ISHLHK HX
FAX: (5) 8681-989

INDIA

Intel Asia Electronics, Inc.
4/2, Samrah Plaza
St. Mark's Road
Bangalore 560001
Tel: 011-91-812-215065
TLX: 9538452875 DCBY
FAX: 091-812-215067

JAPAN

Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 300-26
Tel: 0298-47-8511
TLX: 3656-160
FAX: 029747-8450

Intel Japan K.K.*
Daiichi Mitsugi Bldg.
1-8889 Fuchu-cho
Fuchu-shi, Tokyo 183
Tel: 0423-60-7871
FAX: 0423-60-0315

Intel Japan K.K.*
Bldg. Kumagaya
2-69 Hon-cho
Kumagaya-shi, Saitama 360
Tel: 0485-24-6871
FAX: 0485-24-7518

Intel Japan K.K.*
Mitsui-Seimei Musashi-kosugi Bldg.
915 Shinmaruko, Nakahara-ku
Kawasaki-shi, Kanagawa 211
Tel: 044-733-7011
FAX: 044-733-7010

Intel Japan K.K.
Nihon Seimei Atsugi Bldg.
1-2-1 Asahi-machi
Atsugi-shi, Kanagawa 243
Tel: 0462-29-3731
FAX: 0462-29-3781

Intel Japan K.K.*
Ryokuchi-Eki Bldg.
2-4-1 Terauchi
Toyonaka-shi, Osaka 560
Tel: 06-863-1091
FAX: 06-863-1084

Intel Japan K.K.
Shinmaru Bldg.
1-5-1 Marunouchi
Chiyoda-ku, Tokyo 100
Tel: 03-201-3621
FAX: 03-201-6850

Intel Japan K.K.
Green Bldg.
1-16-20 Nishiki
Naka-ku, Nagoya-shi
Aichi 450
Tel: 052-204-1261
FAX: 052-204-1285

KOREA

Intel Technology Asia, Ltd.
16th Floor, Life Bldg.
61 Yoido-dong, Youngdeungpo-Ku
Seoul 150-010
Tel: (2) 784-8186, 8286, 8386
TLX: K29312 INTELKO
FAX: (2) 784-8096

SINGAPORE

Intel Singapore Technology, Ltd.
101 Thomson Road #21-05/06
United Square
Singapore 1130
Tel: 250-7811
TLX: 39921 INTEL
FAX: 250-9256

TAIWAN

Intel Technology Far East Ltd.
8th Floor, No. 205
Bank Tower Bldg.
Tung Hua N. Road
Taipei
Tel: 886-2-716-9660
FAX: 886-2-717-2455

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

ARGENTINA

DAFSYS S.R.L.
Chacabuco, 90-6 PISO
1069-Buenos Aires
Tel: 54-1-334-7726
FAX: 54-1-334-1871

AUSTRALIA

Email Electronics
15-17 Hume Street
Huntingdale, 3166
Tel: 011-61-3-544-8244
TLX: AA 30895
FAX: 011-61-3-543-8179

NSD-Australia
205 Middleborough Rd.
Box Hill, Victoria 3128
Tel: 03 8900970
FAX: 03 8990819

BRAZIL

Elebra Microelectronica S.A.
Rua Geraldo Flausina Gomes, 78
10th Floor
04575 - Sao Paulo - S.P.
Tel: 55-11-534-9641
TLX: 55-11-54593/54591
FAX: 55-11-534-9424

CHILE

DIN Instruments
Suecia 2323
Casilla 6055, Correo 22
Santiago
Tel: 56-2-225-8139
TLX: 240.846 RUD

CHINA/HONG KONG

Novel Precision Machinery Co., Ltd.
Flat D, 20 Kingsford Ind. Bldg.
Phase 1, 26 Kwai Hei Street
N.T., Kowloon
Hong Kong
Tel: 852-0-4223222
TWX: 39114 JINMI HX
FAX: 852-0-4261602

INDIA

Micronic Devices
Arun Complex
No. 65 D.V.G. Road
Basavanagudi
Bangalore 560 004
Tel: 011-91-812-600-631
011-91-812-611-365
TLX: 9538458332 MDBG

Micronic Devices
No. 516 5th Floor
Swastik Chambers
Sion, Trombay Road
Chembur
Bombay 400 071
TLX: 9531 171447 MDEV

Micronic Devices
25/8, 1st Floor
Bada Bazaar Marg
Old Rajinder Nagar
New Delhi 110 060
Tel: 011-91-11-5723509
011-91-11-589771
TLX: 031-63253 MDND IN

Micronic Devices
6-3-348/12A Dwarakapuri Colony
Hyderabad 500 482
Tel: 011-91-842-226748

S&S Corporation
1587 Kooser Road
San Jose, CA 95118
Tel: (408) 978-6216
TLX: 820281
FAX: (408) 978-8635

JAPAN

Asahi Electronics Co. Ltd.
KMM Bldg. 2-14-1 Asano
Kokurakita-ku
Kitakyushu-shi 802
Tel: 093-511-6471
FAX: 093-551-7861

C. Itoh Techno-Science Co., Ltd.
4-8-1 Dobashi, Miyamae-ku
Kawasaki-shi, Kanagawa 213
Tel: 044-852-5121
FAX: 044-877-4268

Dia Semicon Systems, Inc.
Flower Hill Shinmachi Higashi-kan
1-23-9 Shinmachi, Setagaya-ku
Tokyo 154
Tel: 03-439-1600
FAX: 03-439-1601

Okaya Koki
2-4-18 Sakae
Naka-ku, Nagoya-shi 460
Tel: 052-204-2916
FAX: 052-204-2901

Ryoyo Electro Corp.
Konwa Bldg.
1-12-22 Tsukiji
Chuo-ku, Tokyo 104
Tel: 03-546-5011
FAX: 03-546-5044

KOREA

J-Tek Corporation
6th Floor, Government Pension Bldg.
24-3 Yoido-dong
Youngdeungpo-ku
Seoul 150-010
Tel: 82-2-780-8039
TLX: 25299 KODIGIT
FAX: 82-2-784-8391

Samsung Electronics
150 Taepyungro-2 KA
Chungku, Seoul 100-102
Tel: 82-2-751-3985
TLX: 27970 KORSST
FAX: 82-2-753-0967

MEXICO

SSB Electronics, Inc.
675 Palomar Street, Bldg. 4, Suite A
Chula Vista, CA 92011
Tel: (619) 585-3253
TLX: 287751 CBALL UR
FAX: (619) 585-8322

Dicopel S.A.
Tochtli 368 Fracc. Ind. San Antonio
Azcapotzalco
C.P. 02760-Mexico, D.F.
Tel: 52-5-561-3211
TLX: 177 3790 Dicome
FAX: 52-5-561-1279

PSI de Mexico
Francisco Villas Esq. Ajusto
Cuernavaca – Morelos – CEP 62130
Tel: 52-73-13-9412
FAX: 52-73-17-5333

NEW ZEALAND

Email Electronics
36 Olive Road
Penrose, Auckland
Tel: 011-64-9-591-155
FAX: 011-64-9-592-681

SINGAPORE

Electronic Resources Pte, Ltd.
17 Harvey Road #04-01
Singapore 1336
Tel: 283-0888
TWX: 56541 ERS
FAX: 2895327

SOUTH AFRICA

Electronic Building Elements
178 Erasmus Street (off Watermeyet Street)
Meyerspark, Pretoria, 0184
Tel: 011-2712-803-7680
FAX: 011-2712-803-8294

TAIWAN

Micro Electronics Corporation
5/F 587, Ming Shen East Rd.
Taipei, R.O.C.
Tel: 886-2-501-8231
FAX: 886-2-505-6609

Sertek
15/F 135, Section 2
Chien Juo North Rd.
Taipei 10479, R.O.C.
Tel: (02) 5010055
FAX: (02) 5012521
(02) 5058414

VENEZUELA

P. Benavides S.A.
Avilanes a Rio
Residencia Kamarata
Locales 4 AL 7
La Candelaria, Caracas
Tel: 58-2-574-6338
TLX: 28450
FAX: 58-2-572-3321

*Field Application Location

DOMESTIC SERVICE OFFICES

ALABAMA

*Intel Corp.
5015 Bradford Dr., Suite 2
Huntsville 35805
Tel: (205) 830-4010

ALASKA

Intel Corp.
c/o TransAlaska Data Systems
300 Old Steese Hwy.
Fairbanks 99701-3120
Tel: (907) 452-4401

Intel Corp.
c/o TransAlaska Data Systems
1551 Lore Road
Anchorage 99507
Tel: (907) 522-1776

ARIZONA

*Intel Corp.
11225 N. 28th Dr.
Suite D-214
Phoenix 85029
Tel: (602) 869-4980

*Intel Corp.
500 E. Fry Blvd., Suite M-15
Sierra Vista 85635
Tel: (602) 459-5010

CALIFORNIA

†Intel Corp.
21515 Vanowen St., Ste. 116
Canoga Park 91303
Tel: (818) 704-8500

*Intel Corp.
2250 E. Imperial Hwy., Ste. 218
El Segundo 90245
Tel: (213) 640-6040

*Intel Corp.
1900 Prairie City Rd.
Folsom 95630-9597
Tel: (916) 351-6143
1-800-468-3548

Intel Corp.
9665 Cheasapeake Dr., Suite 325
San Diego 92123-1326
Tel: (619) 292-8086

**Intel Corp.
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642

**†Intel Corp.
San Tomas 4
2700 San Tomas Exp., 2nd Floor
Santa Clara 95051
Tel: (408) 986-8086

COLORADO

*Intel Corp.
650 S. Cherry St., Suite 915
Denver 80222
Tel: (303) 321-8086

CONNECTICUT

*Intel Corp.
301 Lee Farm Corporate Park
83 Wooster Heights Rd.
Danbury 06810
Tel: (203) 748-3130

FLORIDA

**Intel Corp.
6363 N.W. 6th Way, Ste. 100
Ft. Lauderdale 33309
Tel: (305) 771-0600

*Intel Corp.
5850 T.G. Lee Blvd., Ste. 340
Orlando 32822
Tel: (407) 240-8000

GEORGIA

*Intel Corp.
3280 Pointe Pkwy., Ste. 200
Norcross 30092
Tel: (404) 449-0541

HAWAII

*Intel Corp.
U.S.I.S.C. Signal Batt.
Building T-1521
Shafter Plats
Shafter 96856

ILLINOIS

**†Intel Corp.
300 N. Martingale Rd., Ste. 400
Schaumburg 60173
Tel: (312) 605-8031

INDIANA

*Intel Corp.
8777 Purdue Rd., Ste. 125
Indianapolis 46268
Tel: (317) 875-0623

KANSAS

*Intel Corp.
10985 Cody, Suite 140
Overland Park 66210
Tel: (913) 345-2727

MARYLAND

**†Intel Corp.
10010 Junction Dr., Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860
FAX: 301-206-3677

MASSACHUSETTS

**†Intel Corp.
3 Carlisle Rd., 2nd Floor
Westford 01886
Tel: (508) 692-1060

MICHIGAN

*†Intel Corp.
7071 Orchard Lake Rd., Ste. 100
West Bloomfield 48322
Tel: (313) 851-8905

MINNESOTA

*†Intel Corp.
3500 W. 80th St., Suite 360
Bloomington 55431
Tel: (612) 835-6722

MISSOURI

*Intel Corp.
4203 Earth City Exp., Ste. 131
Earth City 63045
Tel: (314) 291-1990

NEW JERSEY

**Intel Corp.
300 Sylvan Avenue
Englewood Cliffs 07632
Tel: (201) 567-0821

*Intel Corp.
Parkway 109 Office Center
328 Newman Springs Road
Red Bank 07701
Tel: (201) 747-2233

*Intel Corp.
280 Corporate Center
75 Livingston Ave., 1st Floor
Roseland 07068
Tel: (201) 740-0111

NEW YORK

**†Intel Corp.
2950 Expressway Dr. South
Islandia 11722
Tel: (516) 231-3300

*Intel Corp.
Westage Business Center
Bldg. 300, Route 9
Fishkill 12524
Tel: (914) 897-3860

NORTH CAROLINA

*Intel Corp.
5800 Executive Dr., Ste. 105
Charlotte 28212
Tel: (704) 568-8966

**Intel Corp.
2700 Wycliff Road
Suite 102
Raleigh 27607
Tel: (919) 781-8022

OHIO

**†Intel Corp.
3401 Park Center Dr., Ste. 220
Dayton 45414
Tel: (513) 890-5350

*†Intel Corp.
25700 Science Park Dr., Ste. 100
Beachwood 44122
Tel: (216) 464-2736

OREGON

Intel Corp.
15254 N.W. Greenbrier Parkway
Building B
Beaverton 97005
Tel: (503) 645-8051

*Intel Corp.
5200 N.E. Elam Young Parkway
Hillsboro 97123
Tel: (503) 681-8080

PENNSYLVANIA

*†Intel Corp.
455 Pennsylvania Ave., Ste. 230
Fort Washington 19034
Tel: (215) 641-1000

†Intel Corp.
400 Penn Center Blvd., Ste. 610
Pittsburgh 15235
Tel: (412) 823-4970

Intel Corp.
1513 Cedar Cliff Dr.
Camp Hill 17011
Tel: (717) 761-0860

PUERTO RICO

Intel Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

TEXAS

Intel Corp.
8815 Dyer St., Suite 225
El Paso 79904
Tel: (915) 751-0186

*Intel Corp.
313 E. Anderson Lane, Suite 314
Austin 78752
Tel: (512) 454-3628

**†Intel Corp.
12000 Ford Rd., Suite 401
Dallas 75234
Tel: (214) 241-8087

*Intel Corp.
7322 S.W. Freeway, Ste. 1490
Houston 77074
Tel: (713) 988-8086

UTAH

Intel Corp.
428 East 6400 South, Ste. 104
Murray 84107
Tel: (801) 263-8051

VIRGINIA

*Intel Corp.
1504 Santa Rosa Rd., Ste. 108
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

*Intel Corp.
155 108th Avenue N.E., Ste. 386
Bellevue 98004
Tel: (206) 453-8086

CANADA

ONTARIO

Intel Semiconductor of
Canada, Ltd.
2650 Queensview Dr., Ste. 250
Ottawa K2B 8H6
Tel: (613) 829-9714
FAX: 613-820-5936

Intel Semiconductor of
Canada, Ltd.
190 Attwell Dr., Ste. 102
Rexdale M9W 6H8
Tel: (416) 675-2105
FAX: 416-675-2438

CUSTOMER TRAINING CENTERS

CALIFORNIA

2700 San Tomas Expressway
Santa Clara 95051
Tel: (408) 970-1700
1-800-421-0386

ILLINOIS

300 N. Martingale Road
Suite 300
Schaumburg 60173
Tel: (708) 706-5700
1-800-421-0386

MASSACHUSETTS

3 Carlisle Road, First Floor
Westford 01886
Tel: (301) 220-3380
1-800-328-0386

MARYLAND

10010 Junction Dr.
Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860
1-800-328-0386

SYSTEMS ENGINEERING MANAGERS OFFICES

MINNESOTA

3500 W. 80th Street
Suite 360
Bloomington 55431
Tel: (612) 835-6722

NEW YORK

2950 Expressway Dr., South
Islandia 11722
Tel: (506) 231-3300

†System Engineering locations
*Carry-in locations
**Carry-in/mail-in locations

UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 300-26

FRANCE
Intel Corporation S.A.R.L.
1, Rue Edison, BP 303
78054 Saint-Quentin-en-Yvelines Cedex

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon
Wiltshire, England SN3 1RJ

WEST GERMANY
Intel Semiconductor GmbH
Dornacher Strasse 1
8016 Feldkirchen bei Muenchen

HONG KONG
Intel Semiconductor Ltd.
10/F East Tower
Bond Center
Queensway, Central

CANADA
Intel Semiconductor of Canada, Ltd.
190 Attwell Drive, Suite 500
Rexdale, Ontario M9W 6H8

Order Number: 240330-002

Printed in U.S.A./1289/20K/RRD JM
Microprocessors

ISBN 1-55512-106-3