i860™ 64-BIT MICROPROCESSOR HARDWARE DESIGN GUIDE

FEB. 1989

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, 386SX, 387, 387SX, 486, 4-SITE, Above, BITBUS, COMMputer, CREDIT, Data Pipeline, ETOX, Genius, i, î, i860, ICE, iCEL, iCS, iDBP, iDIS, I^2ICE, iLBX, i_m, iMDDX, iMMX, Inboard, Insite, Intel, int_el, Intel376, Intel386, int_elBOS, Intel Certified, Intelevision, $int_eligent$ Identifier, $int_eligent$ Programming, Intellec, Intellink, iOSP, iPDS, iPSC, iRMK, iRMX, iSBC, iSBX, iSDM, iSXM, KEPROM, Library Manager, MAPNET, MCS, Megachassis, MICROMAINFRAME, MULTIBUS, MULTICHANNEL, MULTIMODULE, ONCE, OpenNET, OTP, PC BUBBLE, Plug-A-Bubble, PROMPT, Promware, QUEST, QueX, Quick-Erase, Quick-Pulse Programming, Ripplemode, RMX/80, RUPI, Seamless, SLD, SugarCube, UPI, and VLSiCEL, 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 58130
Santa Clara, CA 95052-8130

PREFACE

Preface

The Intel i860™ 64-bit Microprocessor combines capabilities of supercomputers and 3D graphics workstations 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 all the hardware details of the processor. **The design examples published in this manual are NOT TESTED.** The tested examples will be published at a later date.

Although the main users of this manual are hardware design engineers. It contains basic hardware information which is of value to the 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™ Microprocessor Programmers Reference Manual,* Order Number 240329

Information on the device specification for the i860 micrprocessor is available in the *i860™ Microrocessor 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 ". This chapter provides an overview of the features of the "i860 Microprocessor" and the advantages to the system designers. It also provides the insight to i860 microprocessor applications.

* Chapter 2, " Internal Architecture ". This chapter describes the internal architecture of the i860 Microprocessor.

* CHMOS is the patented process of Intel Corporation

* Chapter 3, " Local Bus Interface ". This chapter discusses the i860 Microprocessor local bus interface. This includes the signal descriptions, bus operation and local bus interface guidelines.

* Chapter 4, " Memory Interfacing ". This chapter discusses techniques for designing memory subsystems for the i860 microprocessor.

* Chapter 5, " I/O Interface ". This chapter discusses techniques for connecting I/O devices to an i860 microprocessor system.

* Chapter 6, " Graphics Subsystem Example ". This chapter discusses a design example for implementing an i860 microprocessor based graphics subsystem example.

* Chapter 7, " MULTIBUS II and i860 Microprocessor". This chapter provides a design example for the MULTIBUS II board built around the i860 microprocessor.

* Appendix A and B provides the UNTESTED schematics of i860 microprocessor Graphics Frame Buffer Board and the Multibus II board. Appendix A is to be used in conjunction with Chapter 6. Appendix B contains the DRAM interfacing for Chapter 4 and the MULTIBUS II interface for Chapter 7.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

CHAPTER 5

I/O INTERFACING **PAGE**

CHAPTER 6

GRAPHICS SUBSYSTEM

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

Introduction to 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 combines capabilities of supercomputers and 3-D graphics workstations in a single component. Its integer performance makes it ideal for super-minicomputers. 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, memory management 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 reduce bottlenecks in fetching instructions or data from on-chip caches.

1.1 Processor Characteristics

The i860 microprocessor offers a high level of integration with a million transistor design. This allows a single board design. To complete a processor system all that is needed is four address latches, 8 data tranceivers, clock generator circuit, one EPROM for bootstrapping, a 64-bit wide memory and a few PALs. The i860 microprocessor integrates the following features:

. Integer Processing Unit
. On-chip floating-point and graphics unit.
. On-chip 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. These allow the integer logic to match the speed of the floating-point logic. The integer instruction set controls the entire chip. It can execute any language or portable operating system. RISC design features are:

. 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 support 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 hardware. 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

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 possible performance that can be obtained from DRAM memory. Some of the performance related features are:

. 64-bit design
. Sustains three operations per clock
. Dual-instruction mode
. 33/40 MHz operating frequency
. Executes 85K Dhrystones at 40 MHz
. 80 Mflops peak in single-precision
. 60 Mflops 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
. External bus allows up to three pipelined bus cycles.

The i860 microprocessor has virtual memory and multiprocessor support. These support common operating systems like UNIX*. 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

The i860 microprocessor implementation simplifies system design. It has a conventional microprocessor bus. Standard READY# signal is 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:

. 1X 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 which 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.

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.

1.2.1 Instruction Set 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 special 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. Delayed branching is also used to avoid breaks by executing the instruction following the branch.

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, the graphics unit and for extended arithmetic instructions. The floating-point registers can also be used in pairs as double-precision registers. Quadword memory load instructions use four floating-point registers. Register r0 of the integer registers and f0 and f1 of the floating-point registers are special in that they return a zero value when read and are treated as a null destination when stored into.

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. 8- and 16-bit operands are automatically aligned to the low order bits of a 32-bit register on a load and are moved back to appropriate byte address locations on a store. Only the bytes involved in store operations are enabled for writing. 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 are 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 pipelined, parallel operation in dual-instruction mode of the floating-point unit and integer unit can provide or emulate the capability of vector processing instructions found in supercomputers, but with added flexibility.

A set of floating-point register load instructions provides 32-, 64- and 128-bit operands for the floating-point unit while it operates in dual-instruction mode.

1.2.6 Graphics Support

The graphics unit supports instructions for 3-D color or black and white algorithms. 8-,16- or 32-bit pixel formats are supported. It supports efficient implemention of Phong/Gouraud shading operations. A Z-buffer check instruction is provided to detect the closest surface of a 3-D image. The 3-D graphics hardware 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 line size and a 128-bit data path. It uses a write-back technique 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 they use two 128-bit wide write buffers which post the write operations and delays 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.

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. The TLB performs instruction and data address translation simultaneously and in one clock cycle. Translation buffer 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, writing or both 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 other 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.

A. 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.

B. 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.

C. 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 since 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.

D. 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.

E. 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 cannot be accessed by other processors while this pin is active. Multiple instructions can be executed while the lock signal is active. Operations like compare and swap are possible. All semaphore operations can be implemented. HOLD is not recognized during locked operations.

F. 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.

G. Boundary Scan To Simplify Board Level Testing

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 permit the exercise and test of board logic circuitry with the i860 microprocessor installed.

1.2.11 Clock Requirements

The i860 microprocessor uses an external TTL compatible clock that runs at 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.

2. System Configuration

The i860 microprocessor is suitable as a central processor in a mainframe computer or engineering workstation. It will usually be configured as a stand-alone processor with private memory and a memory mapped I/O subsystem.

Other configurations employ the i860 microprocessor as an application accelerator. In this case a communications bus will exist between the i860 and the host. This bus should be 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 with DMA. 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 its own private memory, along with memory that can be shared.

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.

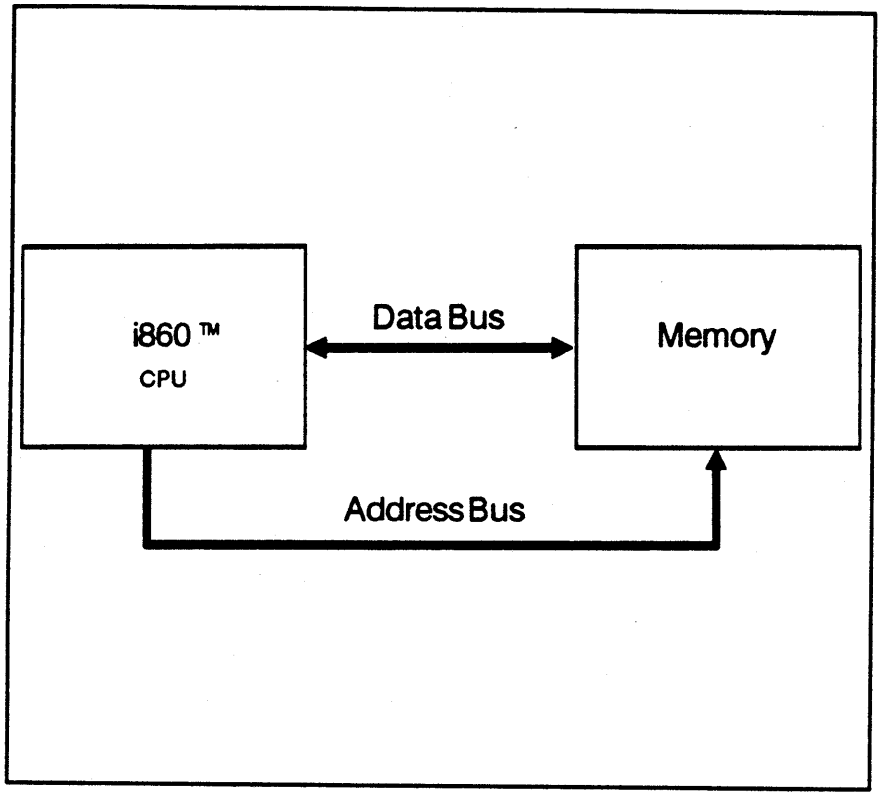


Figure 1.1 Stand-Alone System Configuration

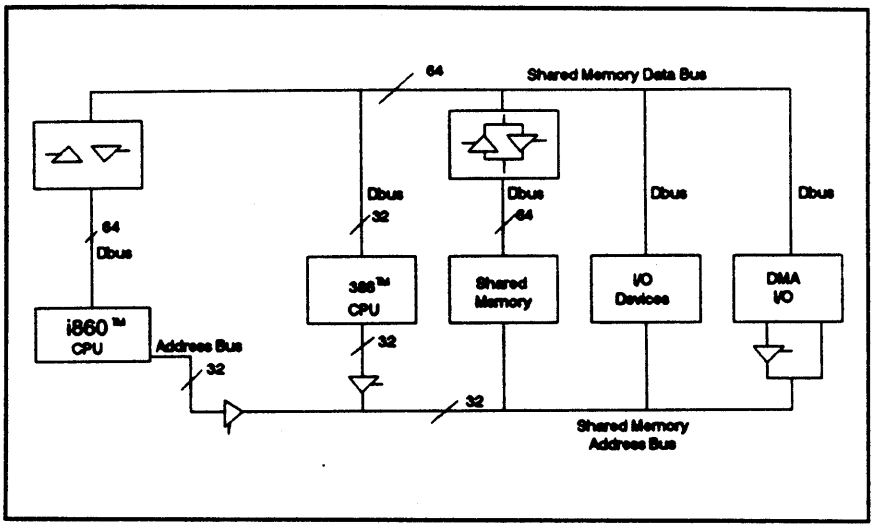


Figure 1.2 Shared Memory Configuration

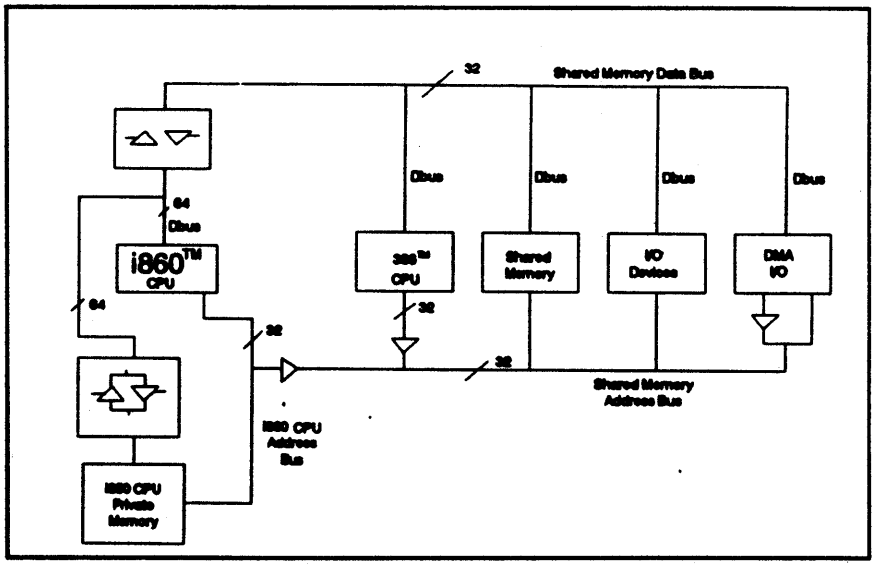


Figure 1.3 Combined Shared and Private Memory Configuration

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 and mainframe computers and supercomputers.

The i860 power can support computational intensive applications such as electronics or mechanical systems simulations. The power of the hardware 3-D graphics support real-time graphics applications of all tupes.

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

* i860, 386 and 486 are trademarks of Intel Corporation.
UNIX is a trademark of AT&T.
OS/2 is a trademark of IBM.

Internal Architecture

2

CHAPTER 2

INTERNAL ARCHITECTURE

The i860™ microprocessor incorporates capabilities of a supercomputer and graphics workstation in its architecture. The i860 microprocessor architecture defines the functional aspects of the target computer system, consisting of registers, arithmetic data paths, memory hierarchy, and control logic. The i860 microprocessor architecture obtains its performance by a combination of larger data paths, increased local access to data and instructions (caches and registers), a larger number of important functions included on-chip, and greater 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 cache provides storage for instructions and data. Wide buses can transfer an instruction pair and two double-precicion floating-point operands each clock. A programmer can use the data cache as 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.

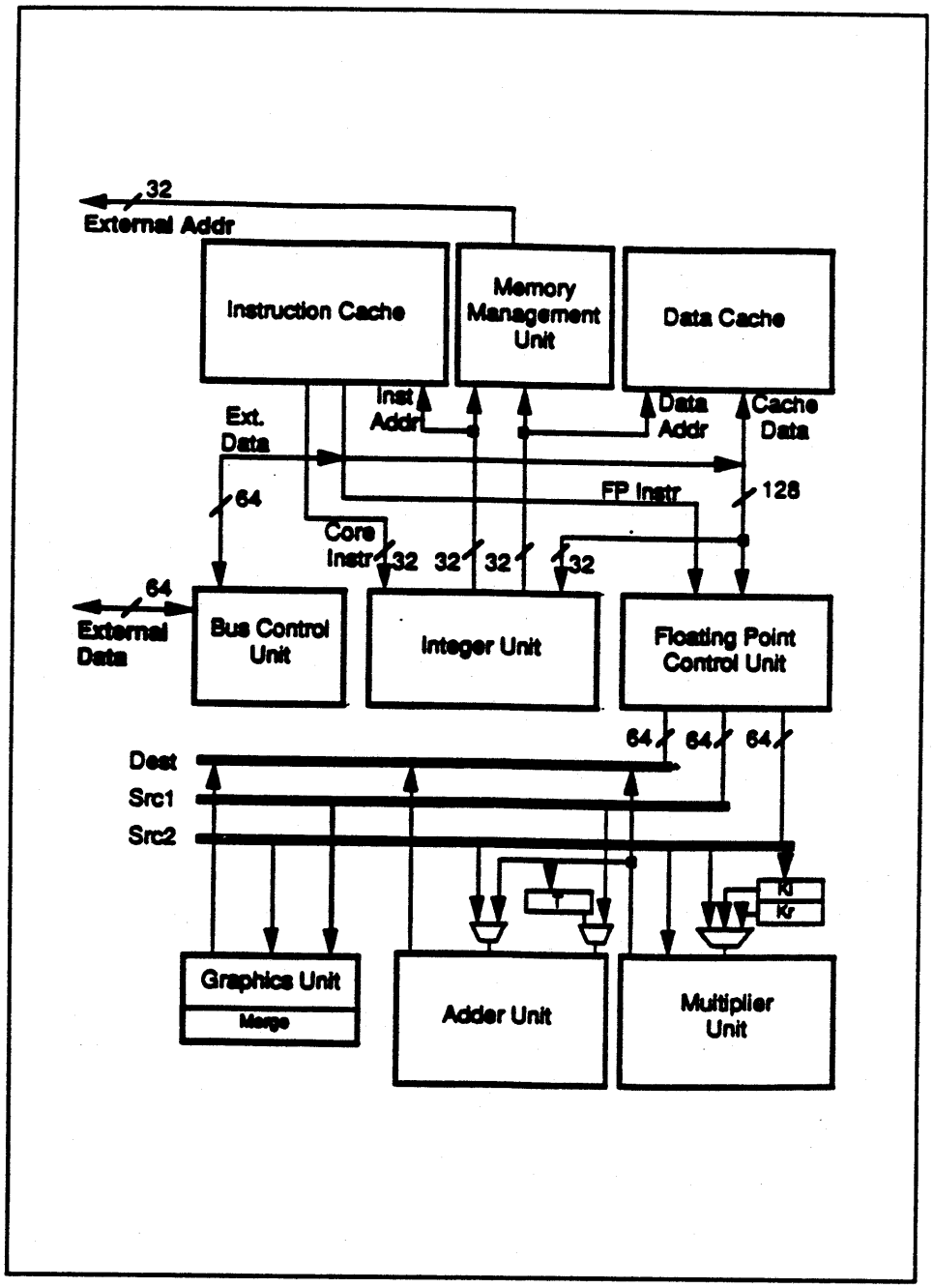


Figure 2.1 Block diagram of i860 microprocessor

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.

The core execution unit includes a register file containing 32, 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 reads a value of zero. Writes to r0 are ignored. 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 microprocessor Programmer's Reference Manual 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 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 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. Refer to i860 Microprocessor Programmer's Reference Manual for details.

The 32-bit instructions are fetched into the core execution unit from the instruction cache given the next instruction's address provided by the core execution unit. If this address location is not in the cache (a cache miss), the 32-bit instruction is fed to the core execution unit from the external memory, while the corresponding Instruction Cache block is simultaneously filled.

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.

Execution pipelining is transparent for arithmetic, logic and shift instructions. These 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, it bypasses the register bank and operand value directly from the bus (in parallel with the previous destination register). This technique is known as register bypassing, and 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 of the extra delay needed to get the valid result, the code should be arranged so that a reference to a register does not directly follow a load instruction to that same register. For this case the instruction directly following the load can execute while the load operation is finishing. However, when the register operand of the current instruction is the same as the destination of an immediately preceding load, the hardware automatically detects this and freezes the clock until the data is present. This technique is called scoreboarding.

The use of register bypassing and scoreboarding allows load and store instructions to be executed at an effective rate of one 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, several techniques are used as described below.

An extra clock cycle is always required on a successful branch in order to refill the instruction pipeline. The i860 microprocessor uses a technique called delayed branching, in which the instruction following the branch is always executed. The system software or the programmer is responsible for reorganizing the program to take advantage of this technique. If no useful instruction can be found for the instruction following the branch, then an operation which does nothing must be used, (e.g. or r0, r0, r0).

A common cause of pipeline breaks is the prefetching of the wrong address on a conditional branch. To avoid this, the i860 ™ Microprocessor provides an optimized set of conditional branch and loop instructions. Conditional branch instructions come in two flavors -- one that prefetches the next sequential instruction, and one that prefetches the branch target address. The compiler or assembler programmer is responsible for using the most desirable flavor of the instruction. Conditional branches at the end of loops, for instance, are usually selected to prefetch the branch destination since code within a loop is executed more often than the code that exits from the loop.

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

The core unit can operate in parallel with the floating point unit. This operation is known as dual instruction mode and is explained in detail in Section 2.2.6.

2.2 FLOATING-POINT UNIT

In addition to the core execution unit, which handles integer instructions, another very important unit is the floating-point control 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.

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.

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.

Due to the low-level instruction set philosophy of the i860 microprocessor architecture, high-level functions defined by the IEEE Standard, such as square root, SIN, and COS 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 processors.

Intel provides 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 0-operand and void 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-poina 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 bi-directional 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 2.2.6.

2.2.2 PIPELINED AND SCALAR OPERATIONS

The floating-point unit uses parallelism to increase the rate of operations performed. 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 2.2.

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.

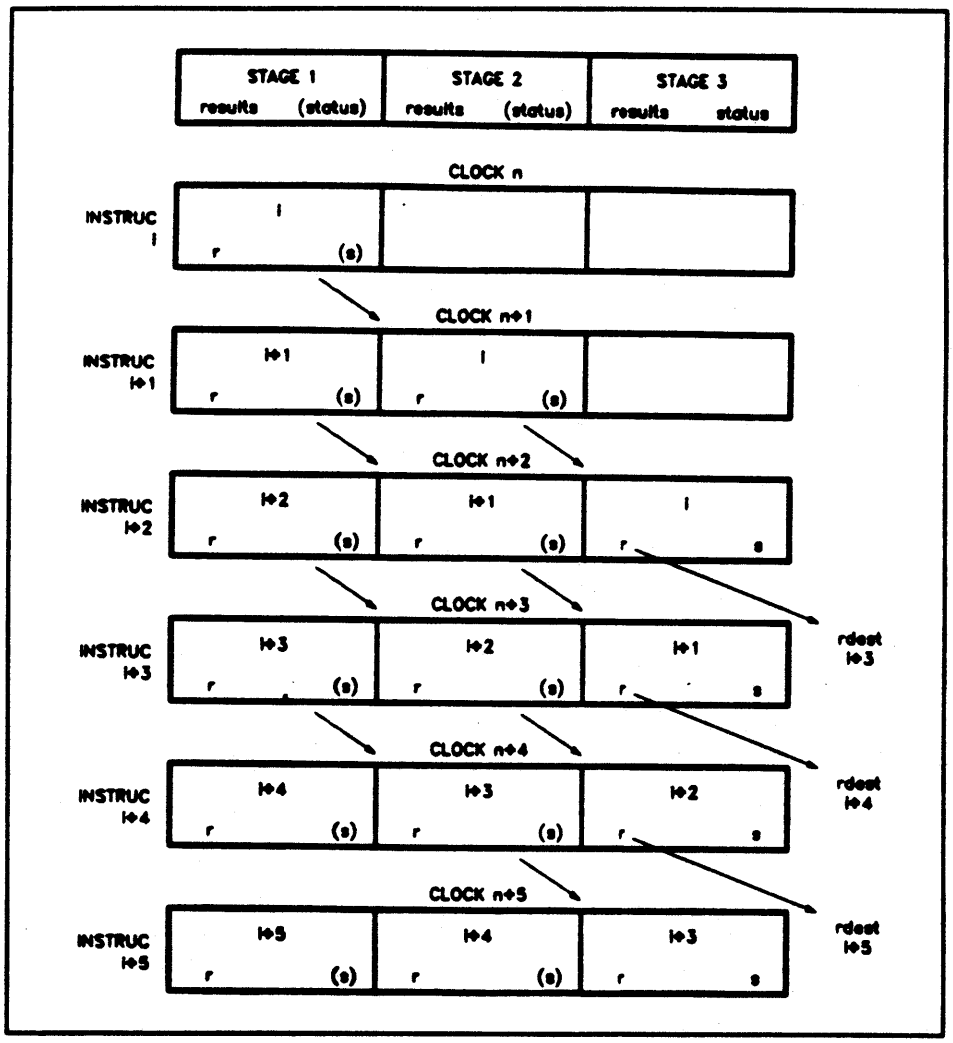


Figure 2.2 Pipelined Instruction Execution.

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.

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 or pipelined mode. In scalar mode, three clocks are required to complete an add, subtract or compare operation. In pipeline 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 double, 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 clock for single-precision and two clocks for double-precision.

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.

Some 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 specially suitable for such problems as:

matrix manipulation (e.g., solving linear equations)
infinite series calculations (e.g., SIN 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 2.3. Possible configurations can be selected as follows:

Operand 1 of the multiplier can be KR, KI, or scr1.

Operand 2 of the multiplier can be scr2 or the last stage of the adder pipeline.

Operand 1 of the adder can be scr1, 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 Microprocessor Programmer's Reference Manual for further details.

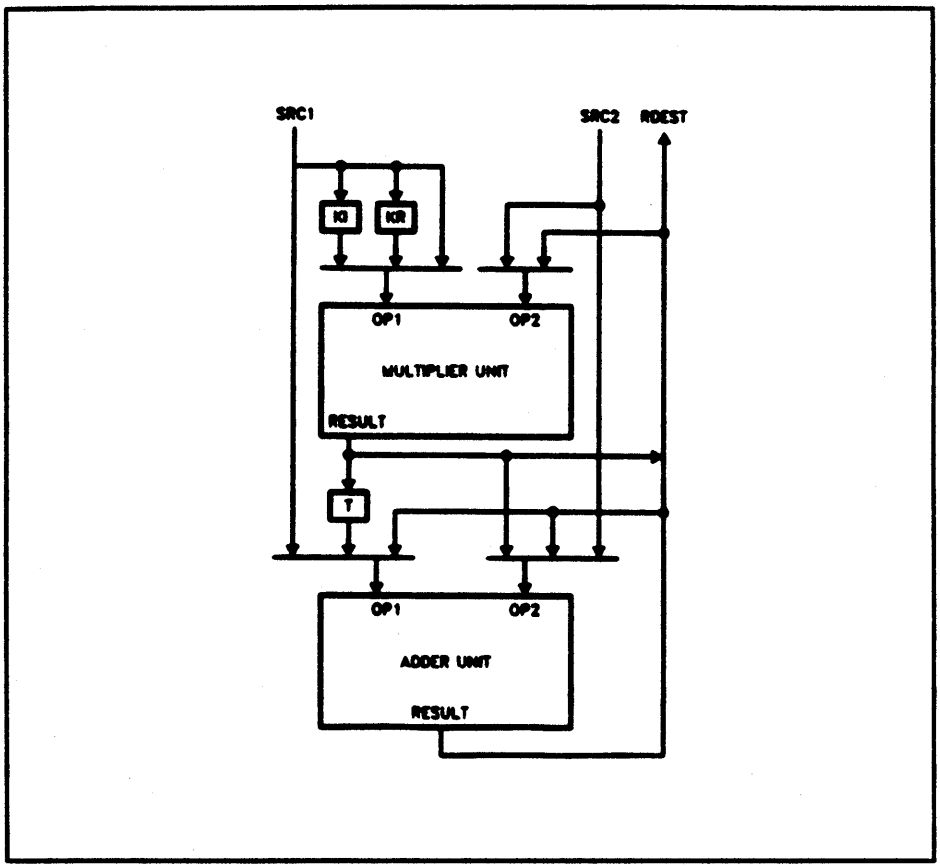


Figure 2.3 Dual-operation Data Paths.

2.2.6 DUAL-INSTRUCTION MODE

The i860 microprocessor also provides an additional form of parallelism, which results from 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. Figure 2.4 identifies the instructions executed by the core unit and those executed by the floating-point unit.

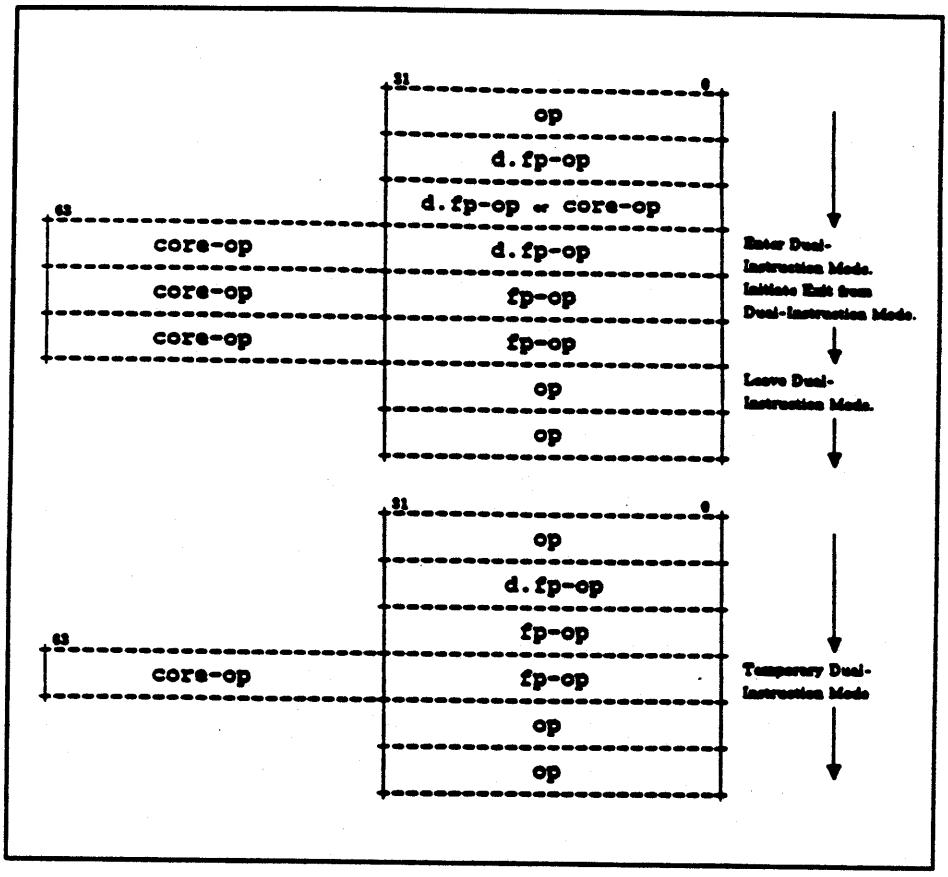


Figure 2.4 Dual-Instruction Mode.

Enabling and disabling dual and single instruction mode is controlled by software. The d.fp-op in Figure 2.4 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 Microprocessor Programmer's Reference Manual.

2.2.7 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 indexes 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 four single-precision operands or two double-precision operands, assuming these operands are adjacent to each other in memory within some data array. Loads and stores take one 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) X(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) X(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 bottleneck), 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 aAddress 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. i860 microprocessor paging unit functions the same and uses the same page table entgry formats as the Intel 386™/486™ 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 address of 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 then, completes the virtual to physical address conversion.

I,f 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 disk. The page table entries also provide the write, user, cache disable, accessed and dirty bits, as well as three user-defined 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 Microprocessor Programmer's Reference Manual.

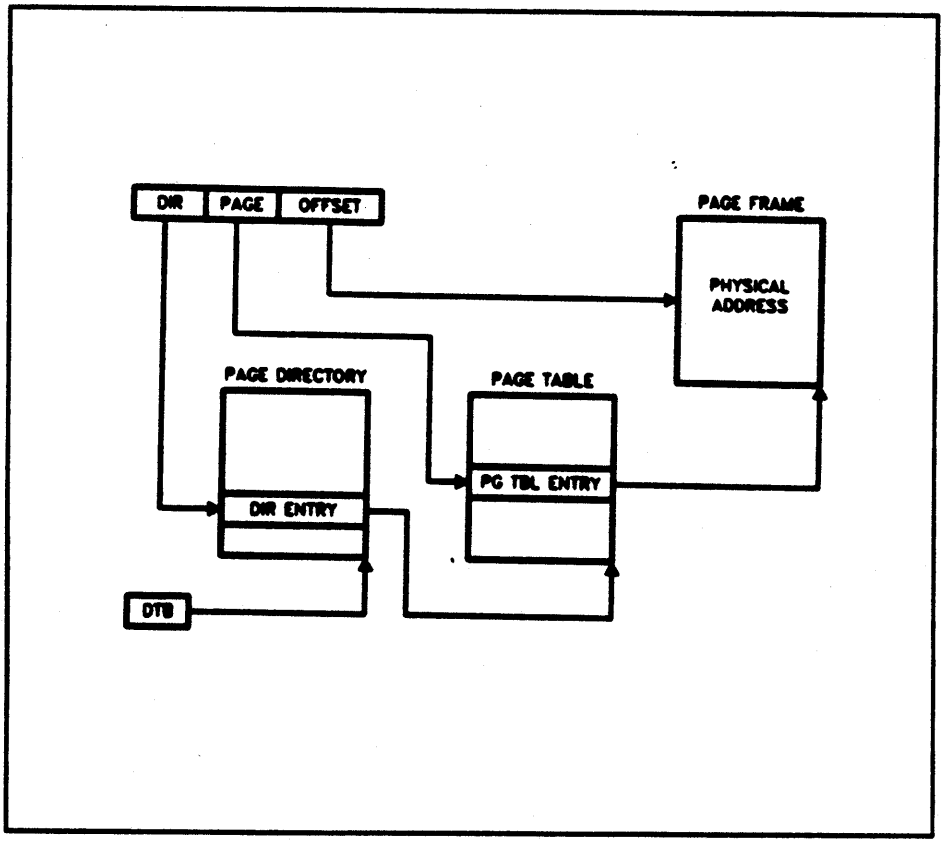


Figure 2.5 Paging Algorithm Implementation.

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 X 64 or 256 Kbytes of memory are mapped at any one time by the TLB. When there 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. Selection of which entry to replace is performed by a proprietary LRU-like (least recently used) algorithm.

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, they can operate in parallel. Thus, the processor can simultaneously read instructions (instruction cache), read or write data (data cache), and translate virtual addresses(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 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 at that point be ready with a physical address with which to begin an external memory cycle.

Both the instruction and data caches are implemented as a two-way set associative memory which 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 five bits. When transferring data to or from the cache, the processor will use 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 wrap-around technique which makes more efficient the process of filling a

cache block and performing the necessary memory reads. Since a block consists of four, 64-bit (8 byte) 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, now having the entry it needs, continues processing. 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 reading 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. If the data item being read is 128 bits, a similar approach is used, but with the two initial reads providing the required data.

The use of the caches when accessing memory can be bypassed by means of the CD (cache disable) bit in the page tables or externally with the KEN# (cache enable) signal 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: one core unit instruction, and one floating-point unit instruction. Thus, the transfer rate of the cache is 64 bits/clock (320mbytes/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 code that requires modification of code memory that might be in the cache should disable the instruction cache and invalidate the contents.

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/clocks. 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-bit, or 8-bit, right-aligned, signed-extended value to the execution core unit.

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

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

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 much 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 indexes that are in the data cache require no external memory cycles.

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 memory cycle. They are the cache enable input pin (KEN#) and the page table bit (PTB) output pin.

When the i860 microprocessor detects that the KEN# input is not asserted prior to

a bus cycle, the processor inhibits use of the data cache and instruction cache for this cycle.

When paging is enabled, the C (cacheable) 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. The value of this bit is reflected on the PTB output pin. When paging is disabled, the PTB pin remains not asserted.

KEN# is internally NORed with the PTB pin to determine whether or not to enable use of the cache, as shown in Table 2.I.

PTB	KEN#	Meaning
0	0	Cacheable access
0	1	Noncacheable access
1	0	Noncacheable page
1	1	Noncacheable page

Table 2.1 Cacheability Based on PTB and KEN #.

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 and 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.

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. It 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 1.

The bus and cache control unit can fetch one 64-bit instruction from the instruction cache and 128-bits 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 (Codesize eight) 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 pin 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 whenever necessary 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 Microprocessor Programmer's Reference Manual.

The graphics unit executes simple but powerful instructions which 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 can perform real-time, shaded graphics without the need for an external graphics processor.

The instructions operate properly for the various pixel formats. The 8-bit format includes three bits of color and five bits of intensity. With this 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 some of 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 such as floating-point instructions can be used in dual-instruction mode, in order to achieve greater computation rates.

Local Bus Interface

3

CHAPTER 3

LOCAL BUS INTERFACE

The local bus is designed to provide high data throughput between 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.

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.

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 bi-directional data bus external interface. Either 8, 16, 32, or 64 bits of data can be transferred during a bus cycle. Pins D7-D0 transfer the least-significant byte. Pins D63 - D56 transfer the most-significant byte.

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 begining 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. This signal is not held active during a pipelined cycle, thus allowing two levels of pipelining.

R/W# — 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 exlanation).

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. The CLK is TTL compatible.

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 eight 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 is useful to allow 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 bit of the page table entry 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. i860 microprocessor bus interface pins are summarized in Table 3-1.

3.1.4. Power and Ground Pins

The i860 microprocessor has 24 GND pins and 24 power pins. The i860 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.

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
Bus Interface Pins			
A31-A3	Address bus	High	O
BE7#-BE0#	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.

TABLE 3.1 Pin Summary

3.3 Bus Transfer Operations

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

3.3.1 64-bit Bus and Byte Alignment of Data

The i860 microprocessor normally performs data transfers on its 64-bit data bus. This section discusses how the bus control unit accomodates data of variable size.

Illustrations in this manual depict the byte furthest to the right as a lowest memory byte address. Bits within data formats are numbered from zero starting with the least-significant bit which is shown at the right of every byte.

3.3.1.1 Memory Addressability and Little and Big Endian Formats

Memory is addressed in byte units within a paged virtual address space of 2^{32} bytes. Data and instructions may be located anywhere within this space.

i860 microprocessor instructions can operate on data of variable size including bytes (8-bits), half-words (16-bits), words (32-bits), double-words (64-bits) and quadwords (128-bits).

Normally, multiple-byte data values are stored in little endian format (with the least significant byte at the lowest memory address). The processor also provides big endian capability whereby the most significant byte is at the lowest address). This feature can be dynamically enabled with software while in supervisor mode, as described in the i860 Programmer's Reference Manual. Big endian and little endian data should not be used together within the same 64-bit data word. Code accesses always use little endian addressing. Figure 3.1 illustrates the two storage modes.

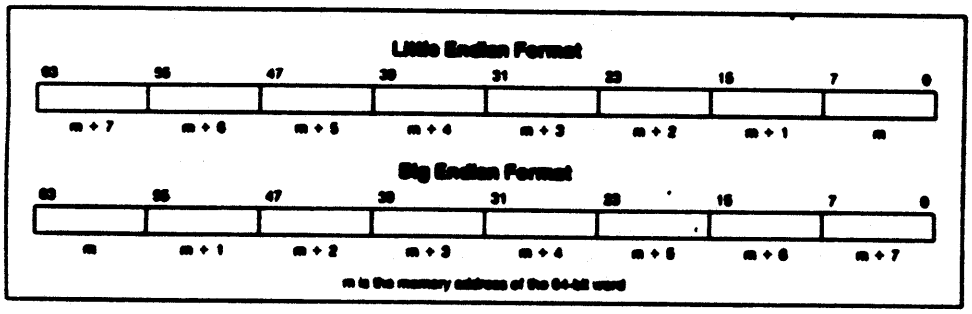


Figure 3.1 Little and Big Endian Memory Format

Data alignment requirements:

* 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 (the three least significant address bits 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).

An instruction cannot access misaligned data items. This causes the instruction to abort and invokes a trap. Trap handling software can perform a load or store at the misaligned address. However, frequent invocation of the trap to handle misaligned data severely degrades performace. Use of misaligned data is not recommended.

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 then processing the data to achieve the format required by processor instructions. The 64-bit data is typically stored in the cache before any processing takes place. The data is prepared by shifting the 64-bit value so that the byte at the lowest address is aligned with data bus D7-D0. Only the numbers of bytes required for the data size are used. The value is then sign extended (for 8- and 16-bit data) if appropriate and loaded into a register.

During read cycles, the byte enable signals reflect the bytes currently needed. These signals should not be used to enable read operations since read operations require all 64-bits to be read.

Enable inputs that correspond to the bytes in a 64-bit word must not be enabled with values of BE7#-BE0#: instead, they must be enabled as shown in the simplified circuit diagram in Figure 3.2

3.3.1.3 Data Alignment During Write Operations

Aligning write data reverses the process used when aligning data for reading. 64- and 128-bit write operations are handled as one and two 64-bit data transfers, respectively. Eight-, 16- and 32-bit memory write operations are performed by aligning the data item to be written and then writing only the bytes involved. For byte operations, the byte enable lines are used to determine what bytes are written into.

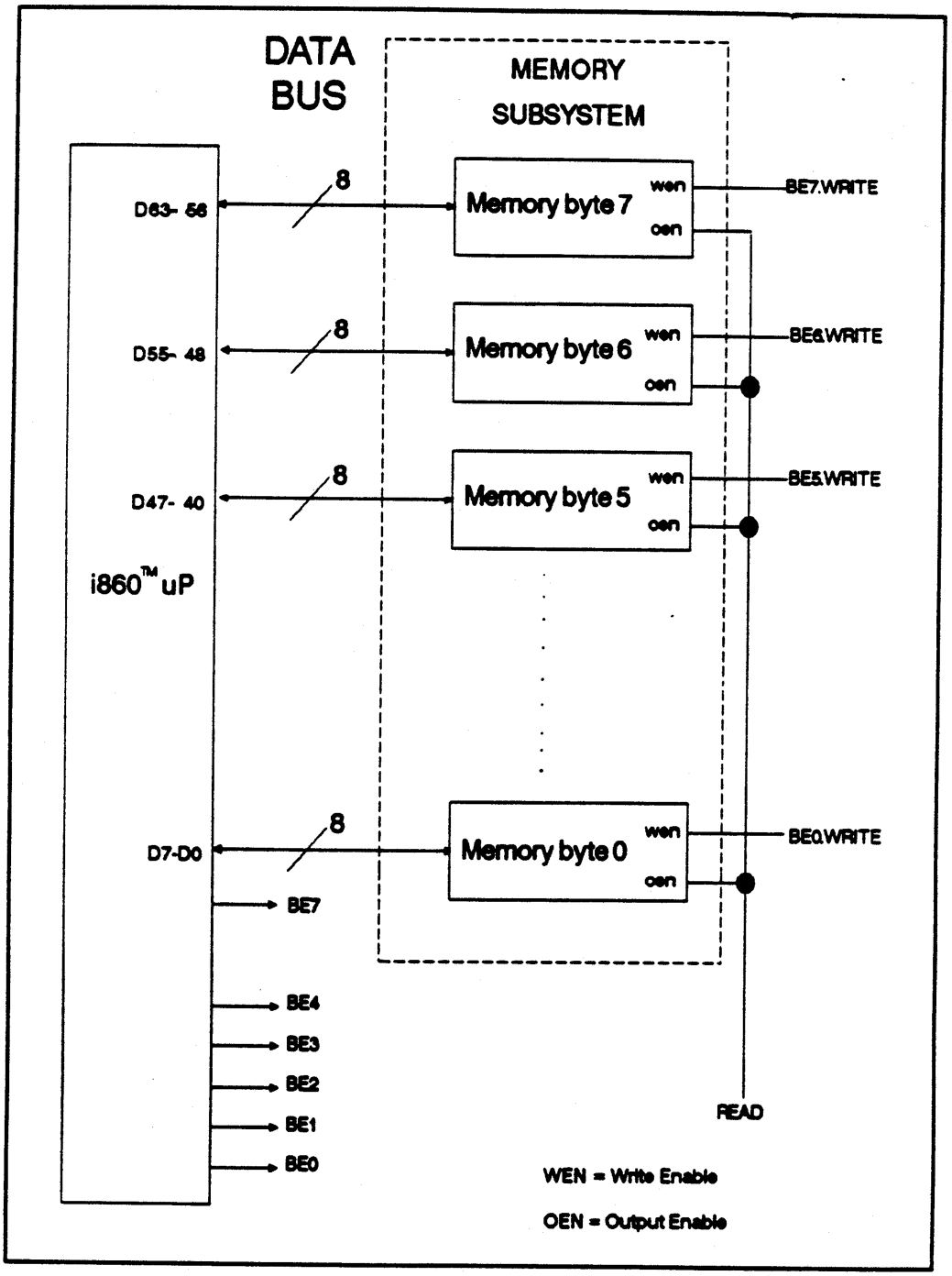


Figure 3.2 Byte Enable Control Signals

For 8-bit, 16-bit and 32-bit data the data being shifted is shifted to the left to properly position the least-significant byte of the data with the corresponding byte address. The amounts of bytes shifted are based on the address and the endian mode.

The write operation is performed on properly-aligned data bytes that are enabled by the proper BE7#-BE0# signals. The circuit shown in Figure 3.3 shows how enable signals operate during a write. Data bytes on the data bus need not be masked because only selected bytes are written while others are ignored. Table 3.2 Shows the corresponding byte enable signals that are asserted for various byte addresses, data sizes and endian modes.

	Byte Enables				
Size	Little Endian (BE7#-BE0#)	Big Endian (BE7#-BE0#)	Internal Bus A2	A1	A0
64-bits	7 to 0	7 to 0	0	0	0
32-bits	7 to 4	3 to 0	1		
	3 to 0	7 to 4	0		
16-bits	7,6	1,0	1	1	
	5,4	3,2	1	0	
	3,2	5,4	0	1	
	1,0	7,6	0	0	
8-bits	7	0	1	1	
	6	1	1	1	0
	5	2	1	0	1
	4	3	1	0	0
	3	4	0	1	1
	2	5	0	1	0
	1	6	0	0	1
	0	7	0	0	0

Table 3.2 Write Byte Enable Combinations

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.

Bus cycles begin when ADS# is sampled active and end 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 non-pipelined 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 Non-Pipelined 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 non-pipelined mode.

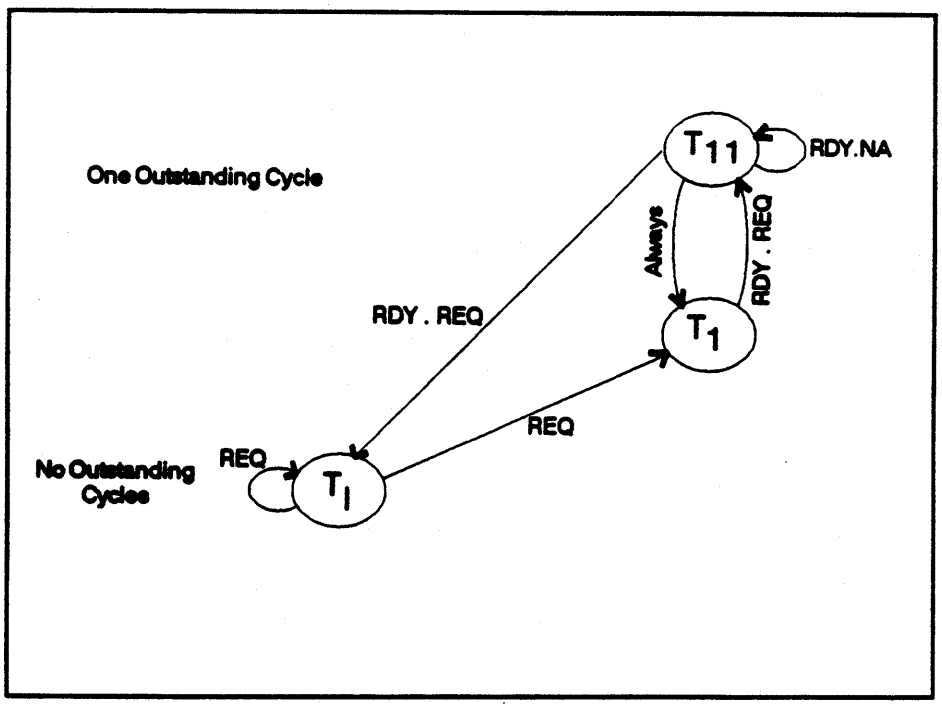


Figure 3.3 Non-Pipelined Bus State Machine

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 of the first T_{11} clock cycle. During read operations, the data bus (D63-D0) is floated to allow the memory sussystem 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 as the state machine repeats the T_{11} state to add a wait-state when 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# allow 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 Non-Pipelined 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 effiiciency, 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 use of pipelining. Each wait-state eases the access time requirement by one clock period.

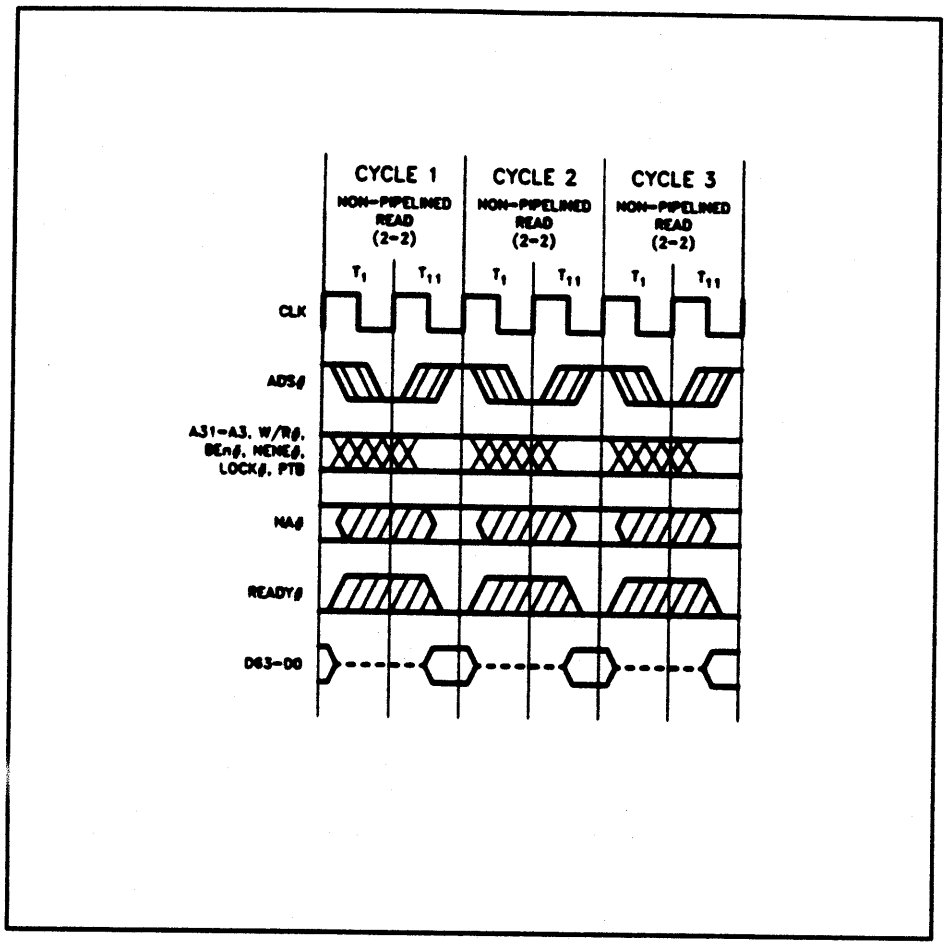


Figure 3.4 Fastest Read Cycles

3.3.3.2 Non-pipelined Write Cycles

Figure 3.5 shows a timing diagram of back to back write cycles. To perform a non-pipelined write cycle, the cycle is started the same as a read cycle, by the processor asserting the ADS# signal during T1. This is followed by the T11 state driving the signals needed to perform the bus cycle. These include the same signals needed for a read, such as the address bus, R/W#, 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 write cycle, Cycle 1 can be physically writing to memory while T1, and T11 of Cycle 2 is setting up the next write cycle. In this way, Cycle 1 has a full two clock cycles to operate without the need for wait-states. This situation is quite different than for read cycles where the required RAM access time needed to perform a two clock non-pipelined operations was 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).

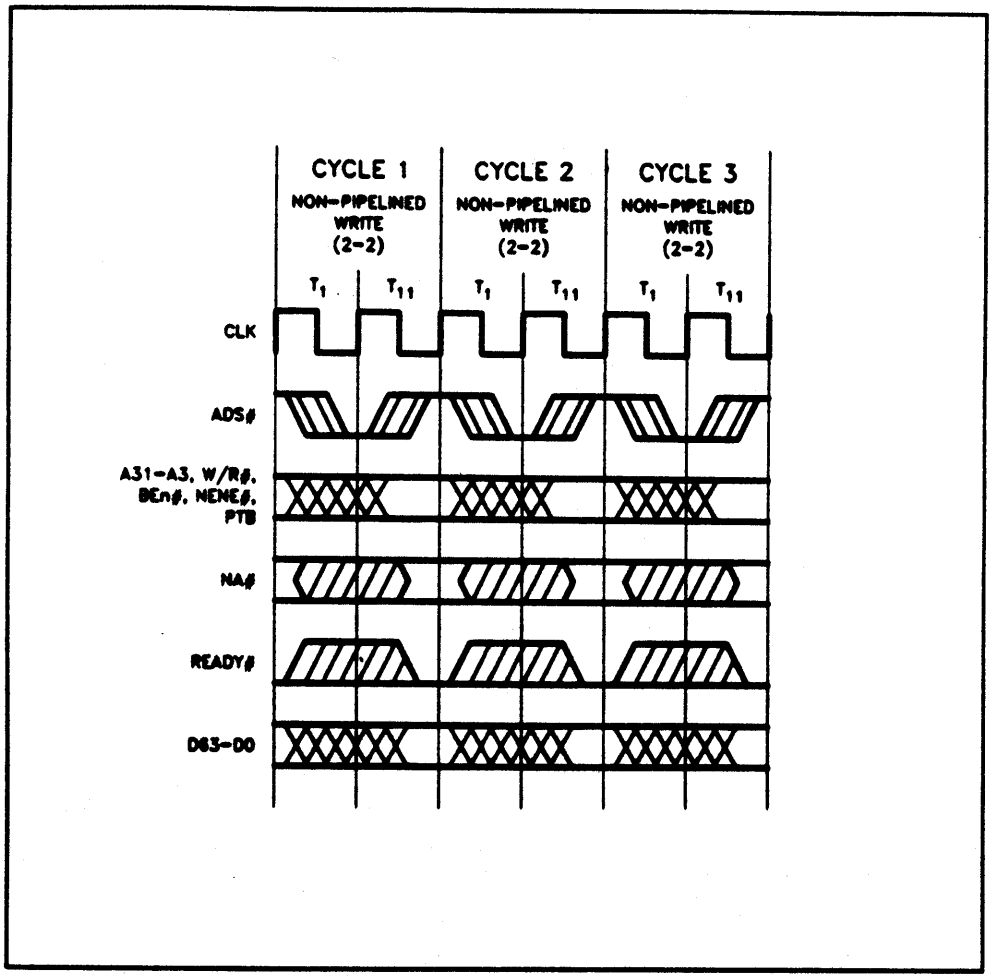


Figure 3.5 Fastest Write Cycles

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. This is demonstrated by the fact that 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 very undesireable due to the degradation in performance.

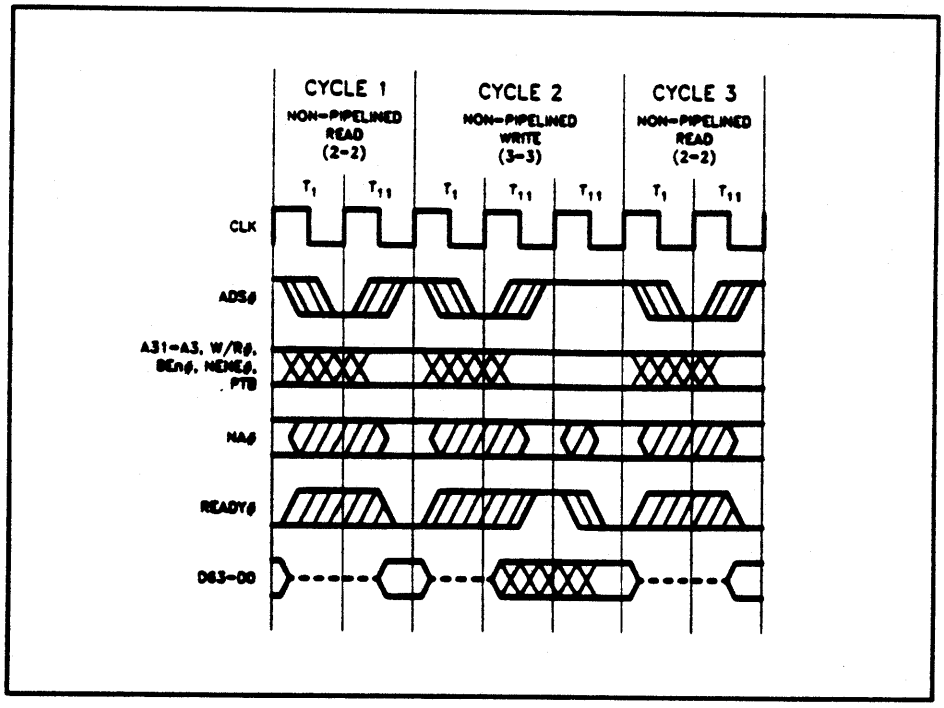


Figure 3.6 Fastest Read/Write Cycles

3.3.4 Pipelined Operations

The i860 microprocessor provides up to two levels of pipelining for 64-bit non-cached read and write operations. Two levels of pipelining implies the presence of three outstanding cycles, one level, 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, one, two, or no 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 only for 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 asserion 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 banks. Pipelined memory reads achieve 6-2 cycles. That is, total cycle times of six clocks and 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.

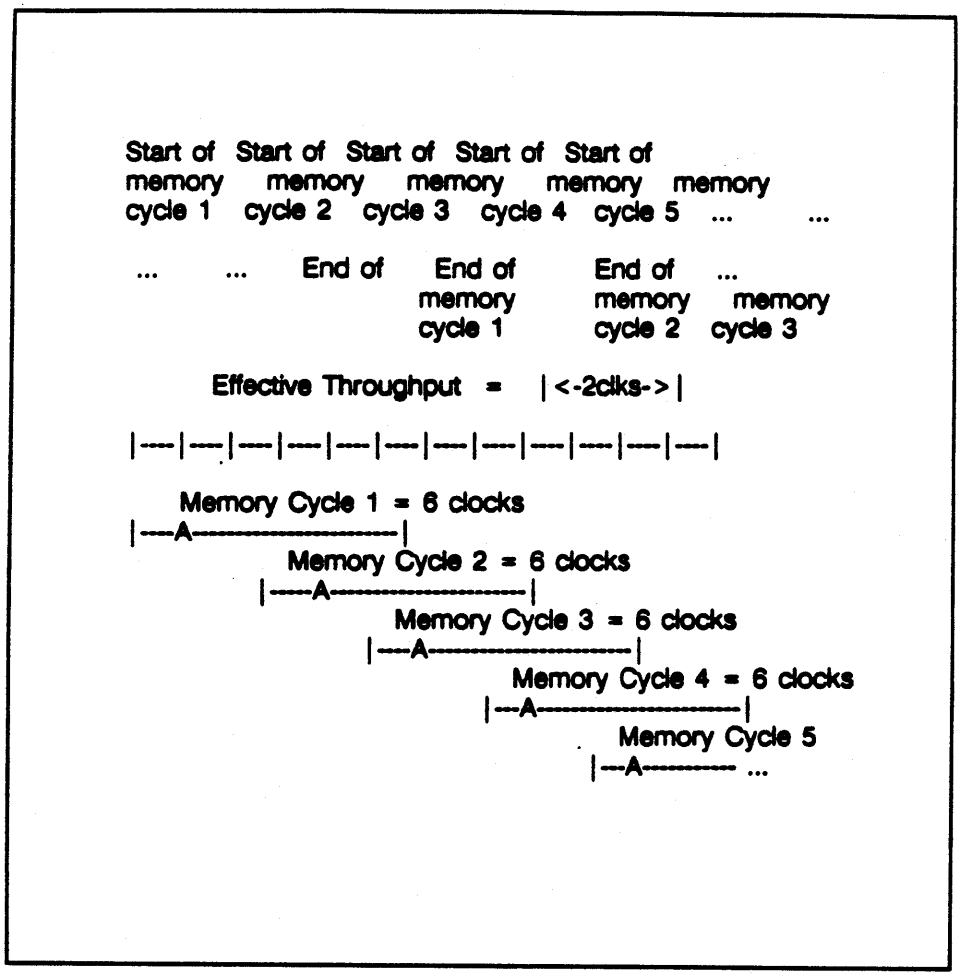


Figure 3.7 Memory Operation Pipelining

3.3.4.1 Pipeline 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 that can work in parallel is used (using three banks is not feasible because of the difficulty in making an address decoder of this kind). Interleaved memory banks are designed to each support 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 then resumes with pipelined operation.
In another approach to the use of bus pipelining, the system initiates one more memory operation than the interleaved memory banks can accomodate. When the extra memory cycle is started, the memory subsystem stops issuing cycles until the accessed memory bank becomes free. The memory subsystem has all the information needed to perform the cycle at this point and while it does it it queues up another cycle. In this manner it eliminates the extra delay that is required to get a valid address and additional signals needed to perform the queued up 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 data 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 in the transition path logic equations.

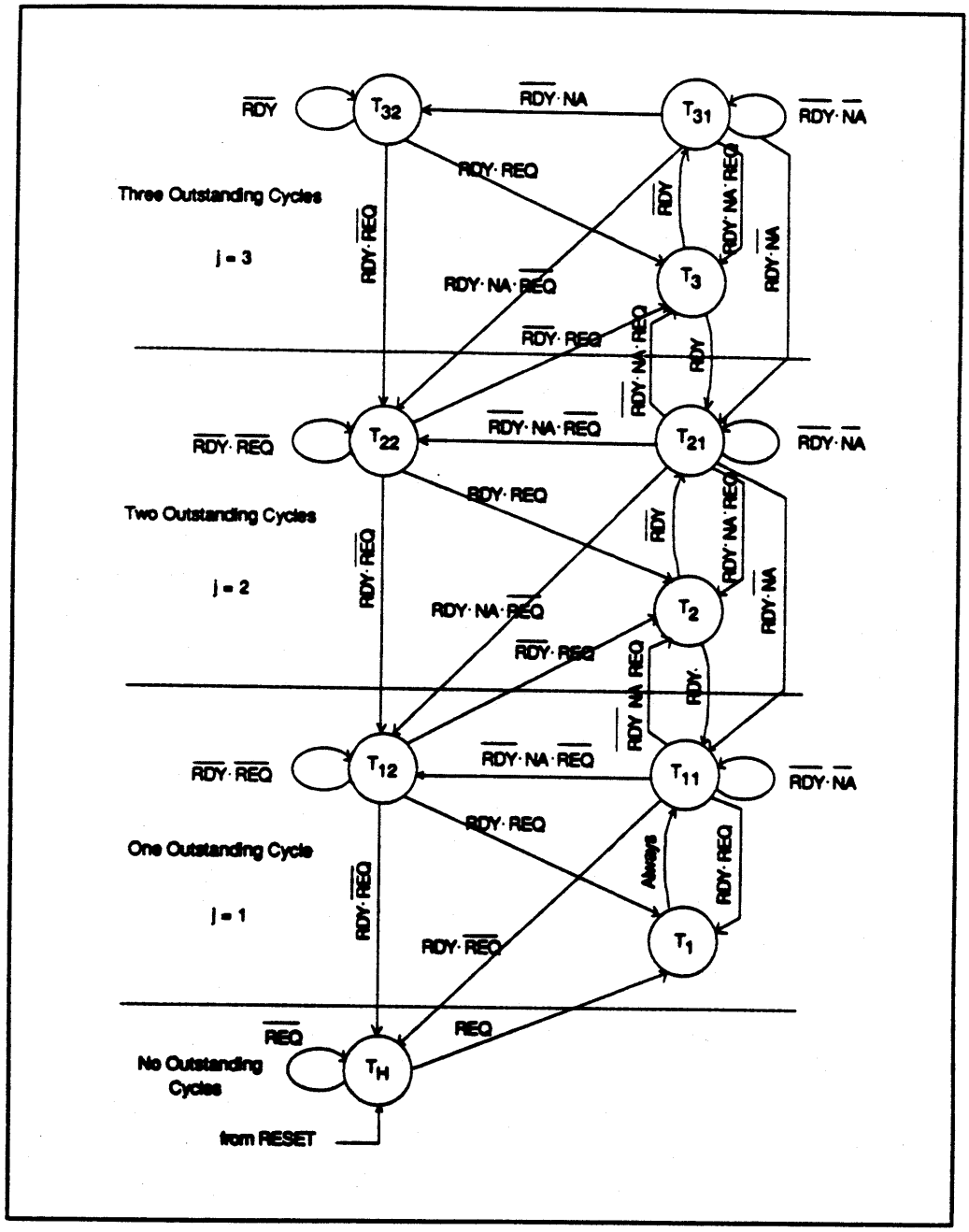


Figure 3.8 Pipelined Bus State Machine

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
* REQ corresponds to an internal processor bus request

The state machine has been 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 non-pipelined bus state machine shown in Figure 3.3.

Two three-state machines have been added to provide non-pipelined operation (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_{j1}. 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 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 $j>1$ is not entered unless there is a request from the processor and the NA# has been or is in the process of being asserted.

T_{j1} is an auxiliary state which helps to complete memory cycles. T_{j1} states operate as the T_{11} state does for in non-pipelined 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).

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. The logic equations for the T_j2 transition paths are like a T_{j1} signal with the NA# asserted.

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 3.10, 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 smapling 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 the bus state machine and pipelined memory cycles. Figures 3.9 shows four memory cycles. Non-pipelined 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).

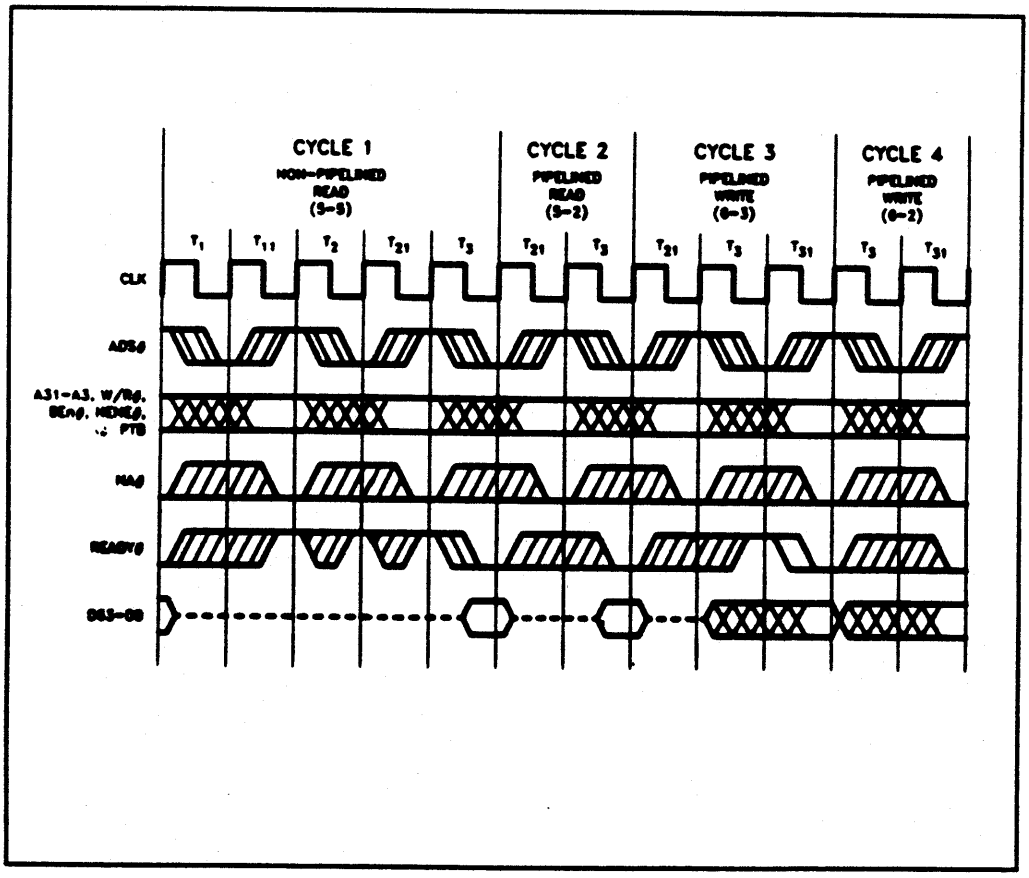


Figure 3.9 Pipelined Read Followed By Pipelined Write

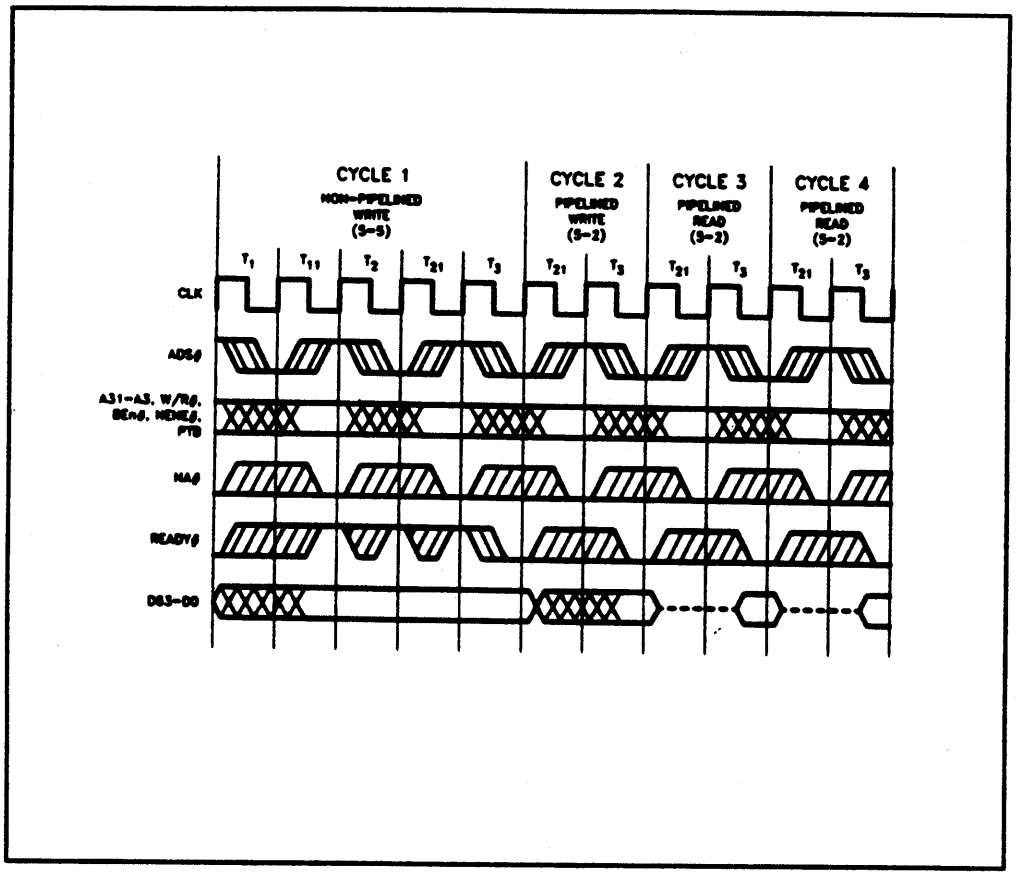


Figure 3.10 Pipelined Write Followed by Pipelined Read

Figure 3.10 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 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

rewduces 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 Cycle 3 and 4 outstanding. For the remaining cycles the state machine oscillates between T_3 and T_{31} through maintaining the bus fully occupied with 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 READY# signal assertion and write data sampling by one clock cycle.

Figure 3.11 shows a non-pipelined 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 by the clock at the time 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 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 (instruction fetches) reside in 8-bit memory (ROM) while data reads and writes are in 64-bit memory (RAM). The processor may be bootstrapped from an 8-bit ROM. A reset operations traps to the 0xFFFFFF00H standard trap handler starting address. Hardware must disable RAM address space covered by the 8-bit bootstrap ROMs 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 INT/CS8 hardware reset.

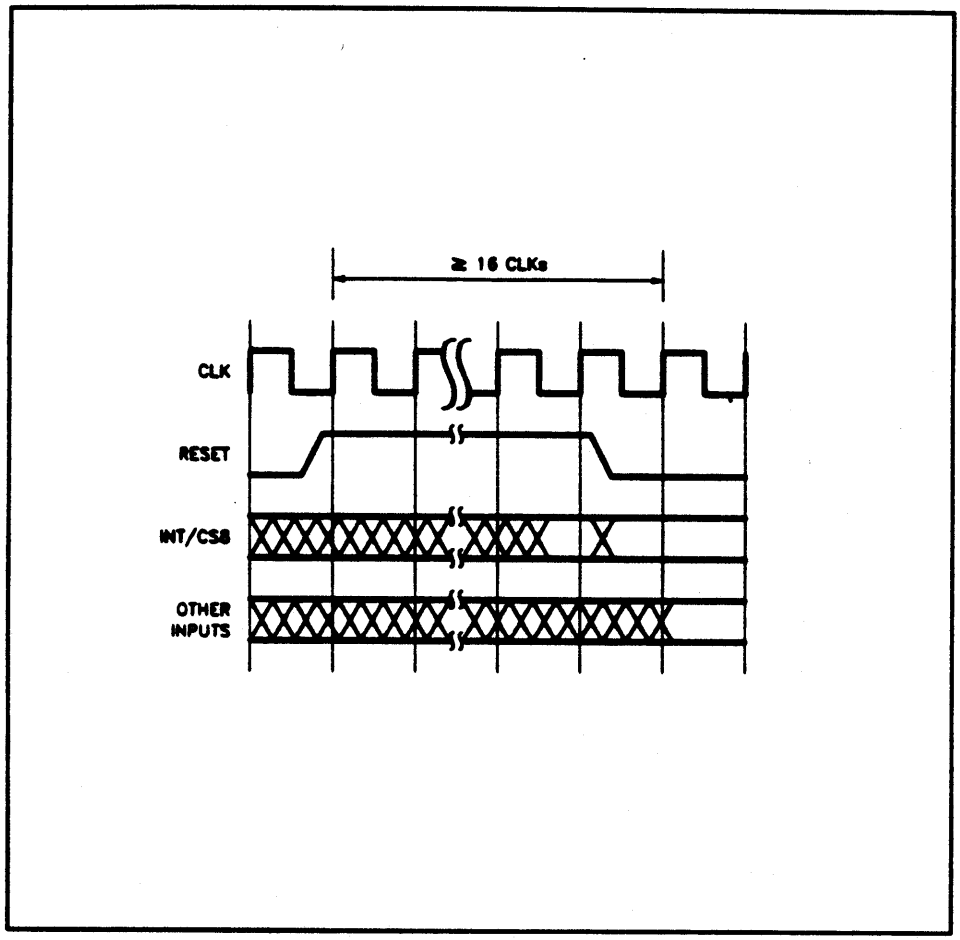


Figure 3.11 CS8 and RESET Activity

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 and

because of the way caches are filled and write buffers are output.

The NENE# signal indicates that a memory cycle is using the RAM page address as the previous cycle. The lower bits of the address corresponding to the size of a page in the DRAM are ignored. 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.

The i860 microprocessor determines page size by interpreting the DPS field in the DIRBASE register. DPS is a 3-bit field corresponding to bits 3, 2 and 1. The value in the field indicates the number of least-significant bits to ignore when comparing a cycle addresses to previous addresses. The number ignored equals 12 plus DPS. Zero is is an appropriate value for 256K X n RAMs, 1 for 1M X 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 processing 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. HOLD is masked out by the bus lock requests (LOCK#) prior to internal synchronization, thereby protecting the bus from being relinquished during a locked sequence. 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 floats all output lines except HOLDA and BREQ to relinquish bus control. 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.

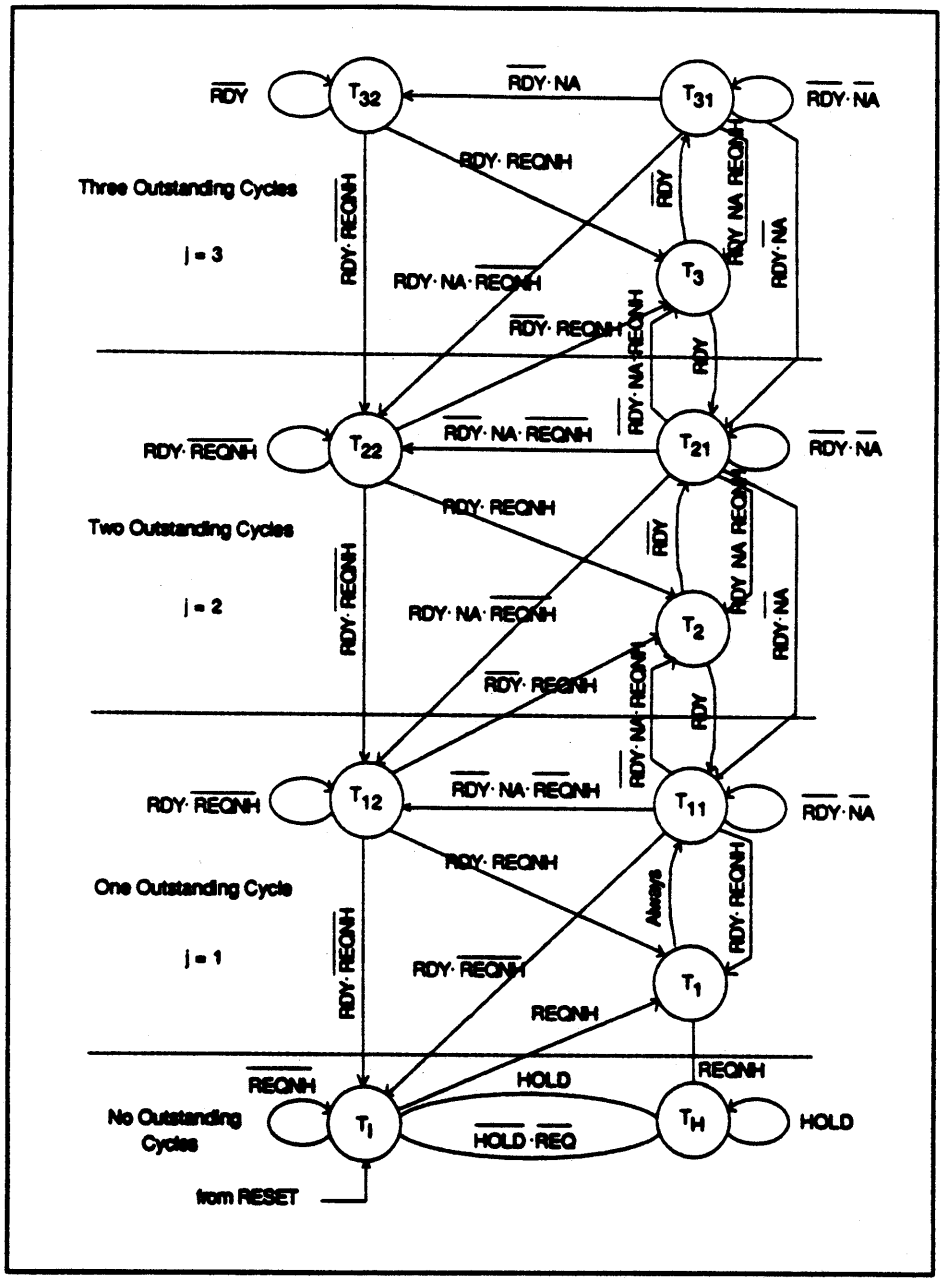


Figure 3.12 Pipelined Bus State Machine Including Hold State

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 and is masked out while a bus lock request is active.

The i860 microprocessor can continue executing from the cache once it has relinquished control of the bus. Execution when the i860 needs to perform an external bus request. 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 brimgs 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 for the fact that 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 or HOLD) transitions 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 T21 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 all output signals except HOLDA and BREQ are floated. 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. 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. BREQ remains active for at least one clock cycle once ADS# has been activated for the requested cycle.

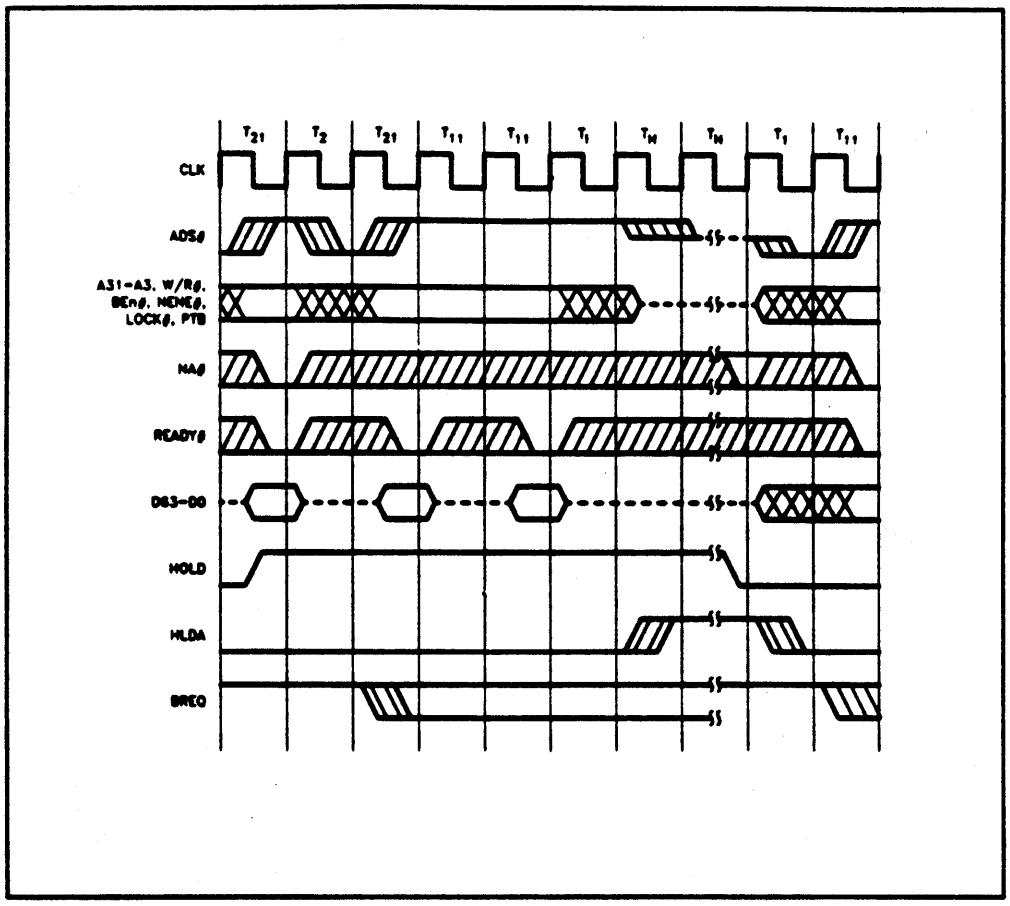


Figure 3.13 HOLD, HLDA, and BREQ

3.4.3 Bus Lock

The bus lock signal (LOCK#) provides indivisible read-modify-write memory operation sequences for use in multiprocessor systems. The internal arbiter (using HOLD and HOLDA signals) will not release the bus to other bus masters until LOCK# is deasserted. Multiprocessor systems with an external arbiter should also use the LOCK# signal to prevent granting the bus to other masters.

Locked sequences begin when the BL bit is set in the DIRBASE register and end

when the BL bit is cleared.

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 non-cachable memory.

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 32 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 non-restartable 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 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 32 instructions have executed following a LOCK instruction, if a load or store not using the cache and 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 32 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 five clocks plus 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. Within the processor, the LOCK# signal masks the HOLD signal before it is synchronized by the processor. In this way, the LOCK# signal prevents the local bus arbiter (HOLD and HOLDA) from relinquishing the bus (See section 3.4.2 provides for more information about the local bus arbiter). The LOCK# output signal internally controls the local bus arbiter. It may also be connected to an external bus arbiter to perform its bus locking function.

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.

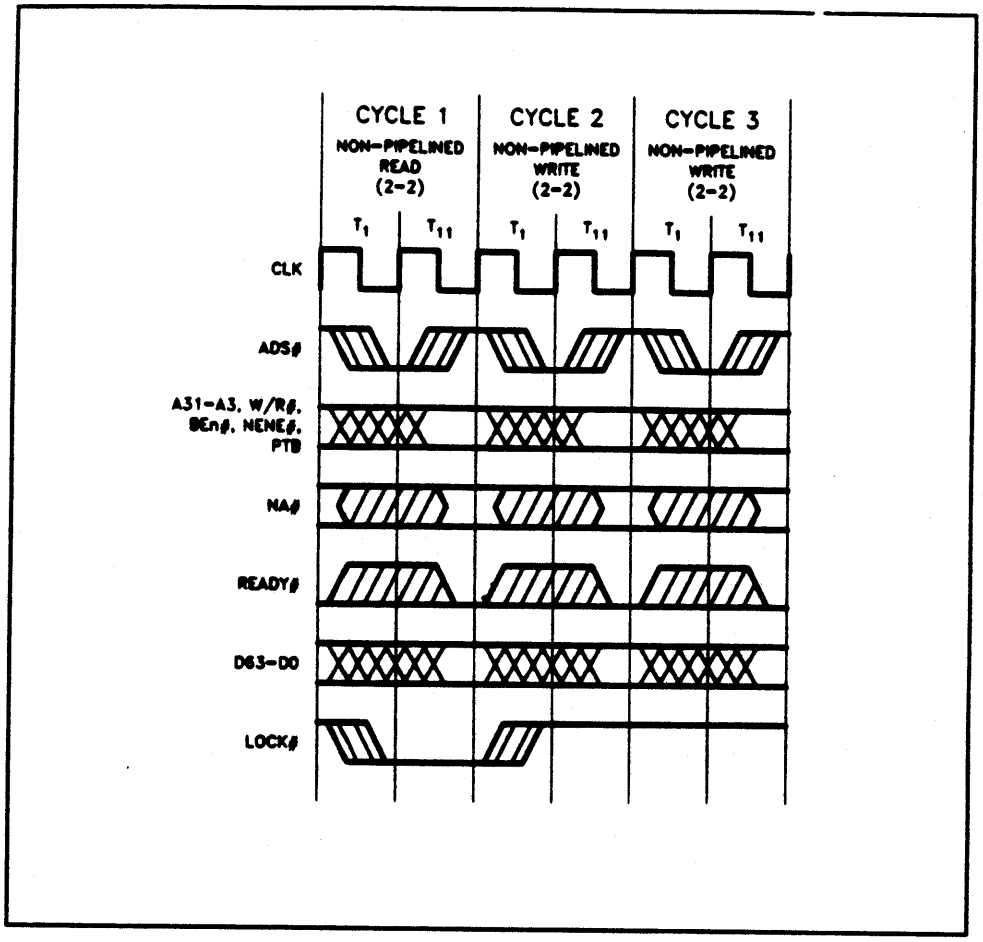


Figure 3.14 Locked Cycles

3.5 CACHE CONTROL OPERATIONS

The KEN# (cache enable) input signal is used to externally 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 the KEN# sampled active. Enabling cache updates for a given cycle requires that the KEN# be sampled on the clock period after ADS# is asserted through the clock in which NA# or READY# is asserted. Updating either cache involves the generation of four 64-bit read operations. KEN# must be asserted through all four cycles. If KEN# is found deasserted at any time during the above described sampling period, the data being read is not cached and two scenarios are possible:

1. If the bus cycle is due to a data-cache miss, no subsequent cache-fill cycles are generated.

2. If the bus cycle is due to an instruction miss, additional cycles are generated until the address reaches a 32-byte boundary.

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 is inhibited. If WT is clear, normal write-back policy applies to page table data. The WT bit of 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 program execution. Traps cause the instruction being executed to abort and transfer control to a trap handler program stored in the hexidecimal virtual address 0FFFFFF00.

The trap handler reads the PSR and FSR register to identify the cause of the trap and branches to a 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 Programmer's Reference Manual explains how to write trap handling software. Table 3.3 summarizes the causes and indications of various traps.

Type	Indication		Caused by	
	PSR	FSR	Condition	Instruction
Instruction Fault	IT		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, and pfeq. Reported on any F-P instruction plus pst, fst, and sometimes fld, pfld, ixfr
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	

*These cases can be distinguished by examining the operand addresses.

Table 3.3 Types of 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). Traps immediately save and clear the IM to inhibit 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 then indicates to a port that the interrupt has been processed.

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 CLK, BSCN, SCAN, SHI and BREQ signals to sample and drive external processor signals.

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.

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

Table 3.4 Test Mode Selection

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

During normal mode, test logic provides a set of latches which can force a value onto the output pins or sample all pin values. Every pin has a latch, and each output signal and output bus (including the bi-directional data bus) has a one-bit control latch.

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	

Table 3.5 Test Mode Latches

Table 3.5 identifies latch names and classifies each as input, output or control. Latches may be loaded and read serially when in serial-mode.

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 new values on all 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. Output pins or buses with the corresponding control latches not set are floated.

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 3.16 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.

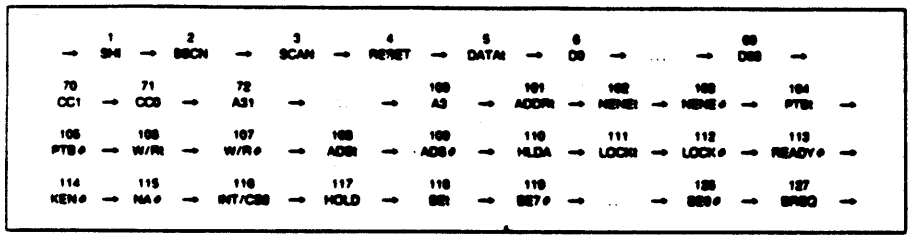


Figure 3.15 Boundary Scan Chain

3.8 RESET AND CLOCK CIRCUIT

i860 microprocessor RESET input must remain high for at least 16 clock periods to initialize the processor. Table 3.6 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 Programmer's Reference Manual details processor status following a reset operation.

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

Table 3.6 Output Pin Status During Reset

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 inverter output. The value is clocked by two D flip-flops to synchronously assert a RESET signal. The 10,000 ohm resistor charges the capacitor and, after a long delay, raises inverter input to a voltage level that moves output switches to a low state. The low-level signal is clocked through two flip-flops and causes the RESET to deassert. The RC delay is long enough to ensure that RESET remains asserted for the needed 16 cycles. 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.

The circuit shown to generate CS8 consists of a driver (e.g. two inverters) to delay RESET and provide 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.

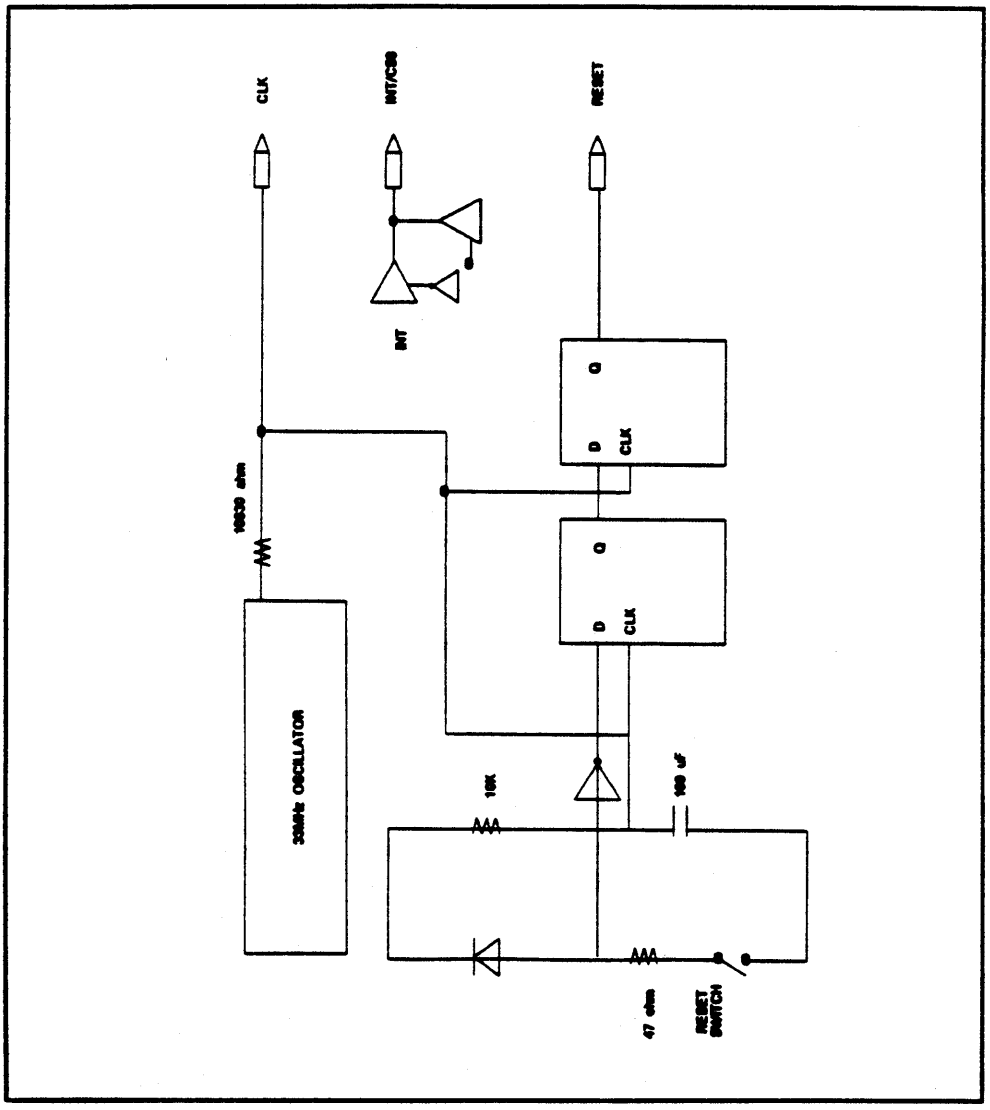


Figure 3.16 Circuit for Clock, RESET and CS8 Generation

Memory Interfacing

4

CHAPTER 4

MEMORY INTERFACING

4.0 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 access are made to the external bus allowing instructions and data to be supplied to the CPU at a very fast rate.

The need for the on-chip cache system stems from the access time performance of the i860 CPU which is double that of current DRAM devices. This disparity is likely to exist for some time. But, the caches reduce the number of accesses to the memory system. Therefore, the memory system's effect on the processor's performance is greatly reduced.

The locality of memory references is key in determining the affect of the memory system on processor performance. Some program algorithms generate widely disbursed memory references. This type of program behavior increases the number of external bus accesses. Obviously, the memory subsystem's effect on processor performance is more pronounced in this case.

Linpack or graphics matrix operations and memory management applications behave in this manner. It is applications such as these that warrant the effort and cost needed to design efficient DRAM subsystems. An inefficient design can reduce the performance of these applications by 60 or 70 percent. Poor performance of these applications can have a disproportionate detrimental effect on overall system performance. This effect is especially noticeable if system functions such as paging 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 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 here.

The example assumes a working knowledge of computer system design. Items that will be discussed but not explained include DRAM operation, PAL programming and operation, worst case analysis techniques and i860 microprocessor bus operation.

4.1 CPU features

Certain i860 microprocessor features are specially devised to facilitate optimum DRAM subsystem performance. The next near pin (NENE#) is one example: it serves to define DRAM page size. This section briefly describes these special features.

4.1.1 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 line 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 11 nanoseconds, dictating that KEN# be generated by combinatorial logic.

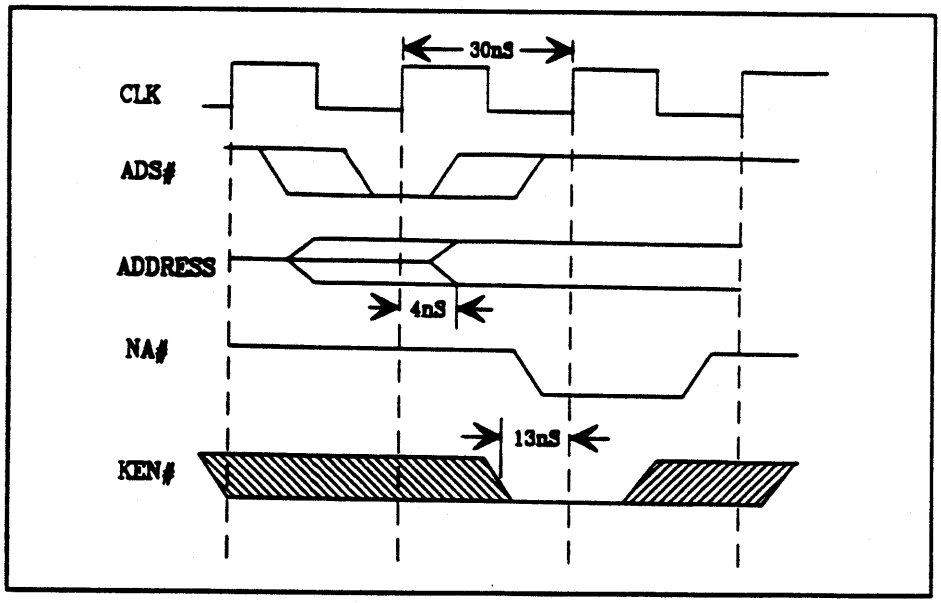


Figure 4.1 KEN# timing

4.1.2 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 RDY# 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 perfomance does benefit from pipelining, however, especially if read cycles are due to a floating point vectore load. Line fills replace that with vector loads. Vector loads can occur in long strings. Since most of these cycles do not cross a DRAM page boundry, they can be executed with a minimum number of wait-states. Figure 4.2 demonstrates how pipelining facilitates cache line fills. Here, two cache line fills occur consecutively.

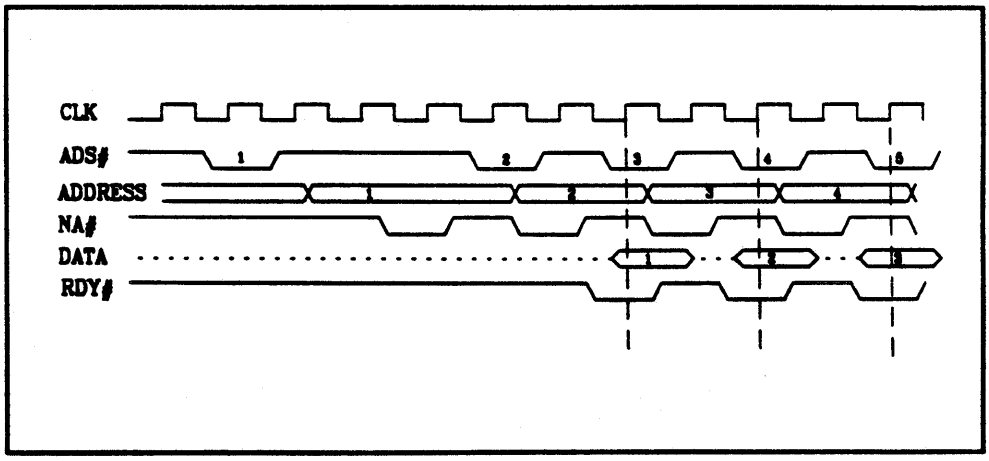


Figure 4.2 Bus Pipelining

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 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 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. In contrast, non-pipelined read cycles return data every four clocks.

4.1.3 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 clock cycles: 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. Zero-wait-state accesses could not be performed without this feature.

4.1.4 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.

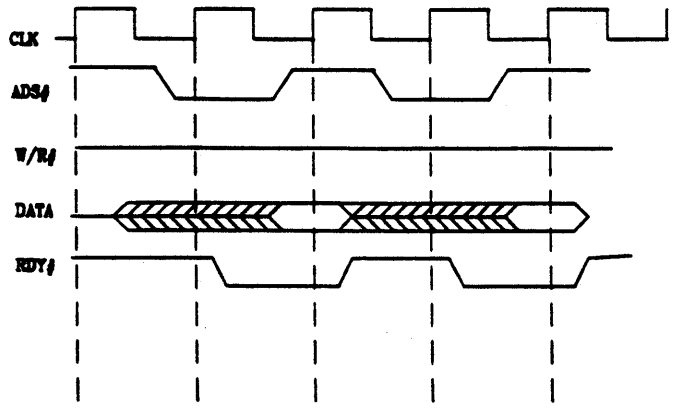


Diagram A. Zero-Wait-State Write Cycles

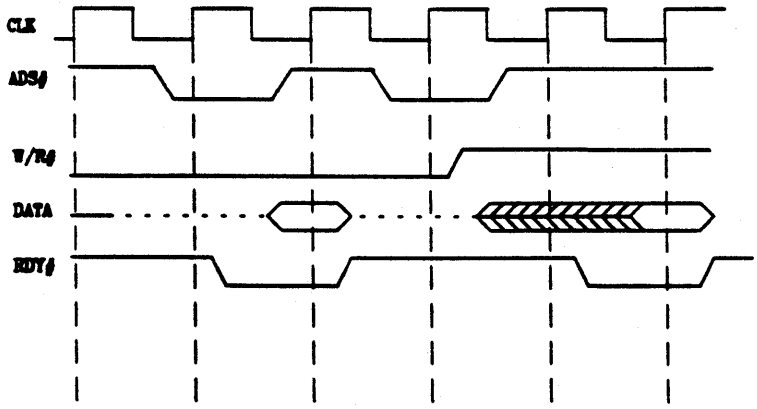


Diagram B. Read cycle Followed by Write

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 transcievers to tristate the processor side of the bus before the CPU drives the write data.

4.2 DRAM Subsystem Overview

The features just described help make the DRAM subsystem more efficient. These functions are included in the processor to save on the 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 x 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 x 1 memory devices, the minimum memory configuration is 8 Mbytes (64 devices).

As defined in this chapter, the system can support up to eight 8 Mbytes. It is easily upgraded, however.

Figure 4.4 is a block diagram of the memory subsystem example. It is comprised of five parts: address path logic, data path logic, parity logic, controller and DRAMs.

4.2.1 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 and uses the device's OE pin 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. 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 x.4. This configuration allows up to eight Mbytes of memory. Additional drivers can easily be added for larger configurations.

4.2.2 Data Path Logic

The data path consists of eight 74AS646 bi-directional latching transcievers. These devices fulfill two critical memory system requirements. These transcievers 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 second cycle's data is held at the transceiver input.

The register feature of the transcievers 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 4.3.4 provides a detailed description of the posted write function.

4.2.3 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 1Mbit chips organized as 1M x 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

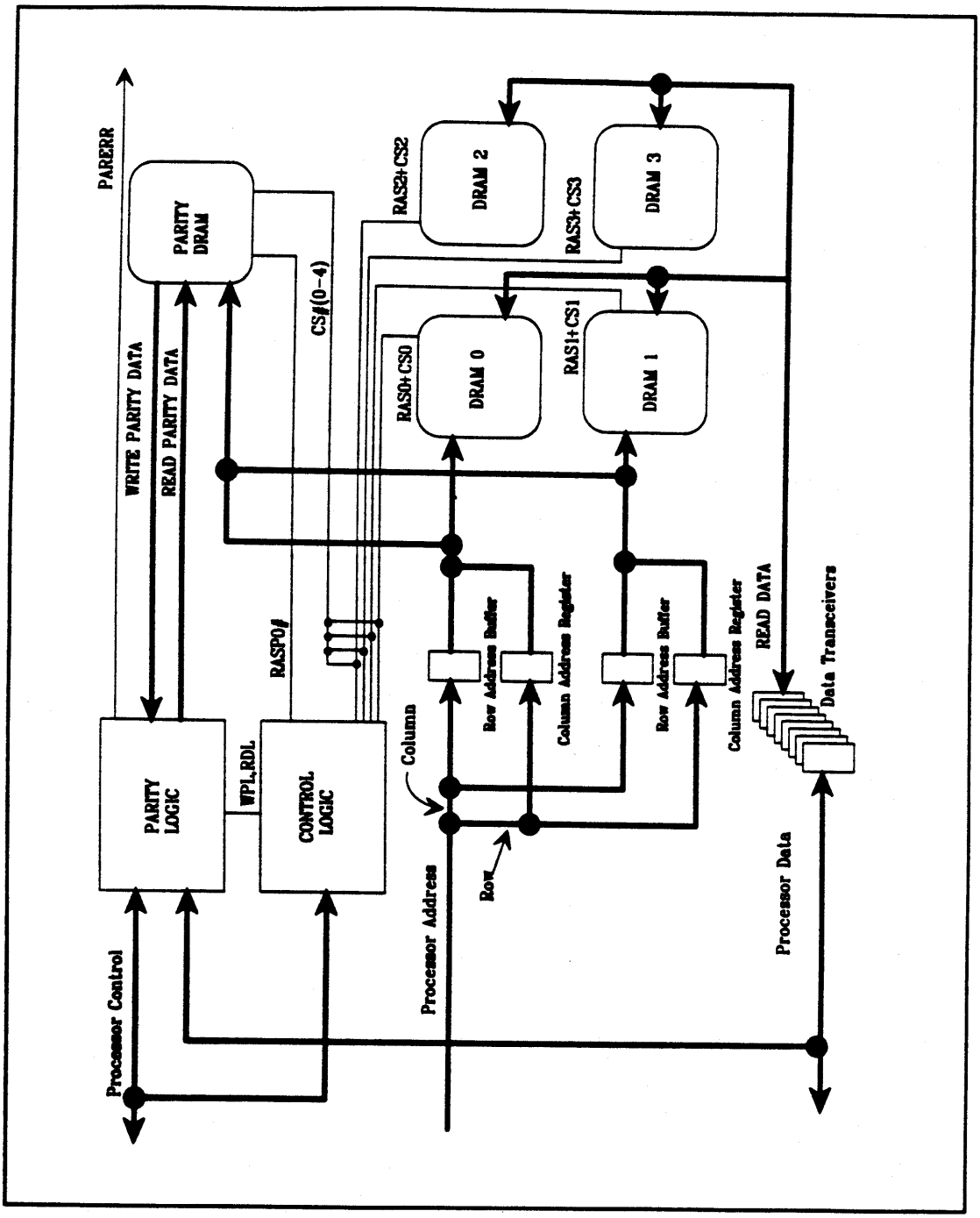


Figure 4.4. Maximum Example Configuration

light of noise immunity.

4.2.4 Control Logic

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

The state machine PAL 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 PALs 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 PAL (DSTAT in Appendix B) generates control signals NA# and RDY#. These constraints will be explained in the functional description (Section 4.3).

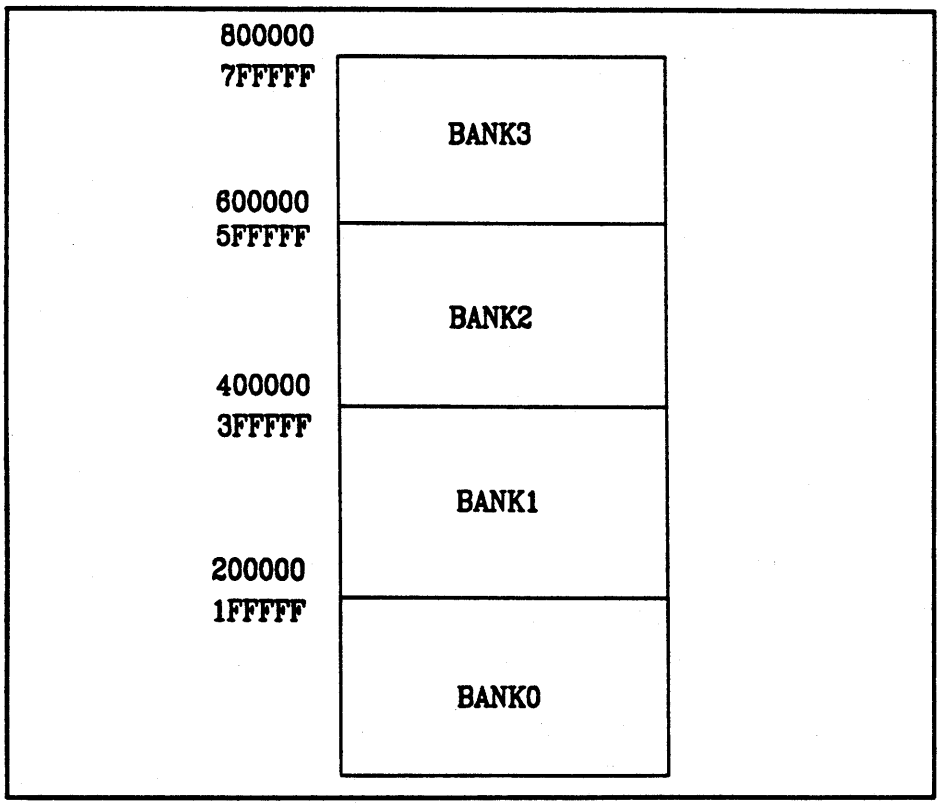


Figure 4.5 Address Map

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

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

This PAL also generates RAS for the parity DRAMs. RASP0 enables the 1M x 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 PALs generate all address and data path control signals. They both have as inputs the state variables from the DSTAT PAL. Control-B also operates from a delayed clock, DCLK.

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

The Control-B PAL controls the column address path. The PAL 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.

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 PAL 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 PAL. The signal is generated at the beginning

and end of write cycles. By monitoring LADS# and W/R#, this PAL can determine the type of cycle being run when WEL is activated. The WE# signals are activated during writes and deactivated during reads.

4.3 DRAM Subsystem Function

This section defines the function of DRAM subsystem bus cycles.

4.3.1 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.

4.3.1.1 Processor Interface

NRDY#
: NRDY# is directly connected to the processors RDY 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#.

4.3.1.2 Data Path Logic Control

RAE#
: RAE# enables the row address drivers. It disables the row address before the 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.

4.3.1.3 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 PAL. Each signal corresponds to a single byte in each DRAM Bank.

4.3.1.4 Controller Signals

WEL
: WEL is connected to the clock input of the control-D PAL. 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 20 ns.

S0-S4
: These signals are the state variable outputs of the state machine. They are decoded by other control PALs to produce cntrl signals.

BSY#
: BSY# is used to prevent other subsystems from driving NA#, RDY# 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.

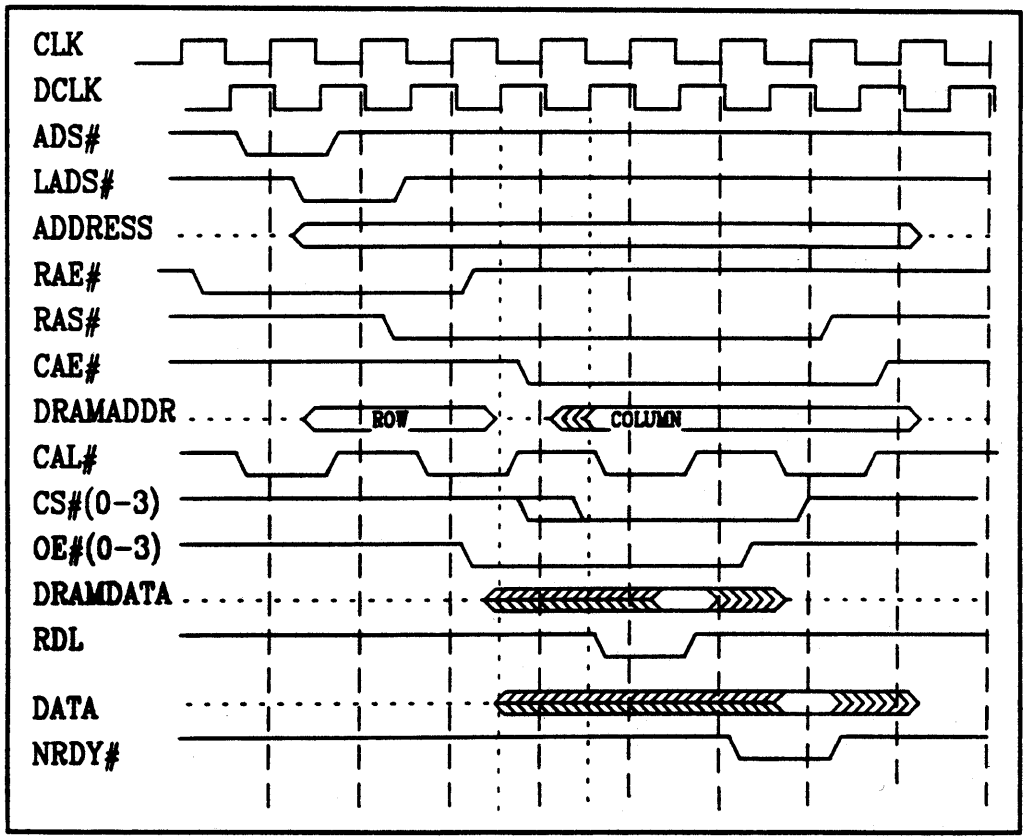


Figure 4.6 Basic Read Cycle

4.3.2 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. Pipelining is not used in this example to simplify discussion, 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.

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 data transciever registers. It is driven from the same PAL as CAE# and is exactly in synch with CAE#. RDL is activated two DCLKs after CAE#, however, to be ensured active in time to latch 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.

4.3.3 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 line fills.

A cache line fill generates four consecutive read cycles. Using the pipelining feature of the bus, these cycles may be overlapped as shown in Figure 4.6. Through pipelining, addresses of subsequent read cycles are driven before the previous cycle completes. As a result, latency of the last three cycles of a fill line is reduced. A cache line fill begins as a basic read cycle. It is converted to a cache line 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 line, 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 PAL 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 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 will remain active in this case.

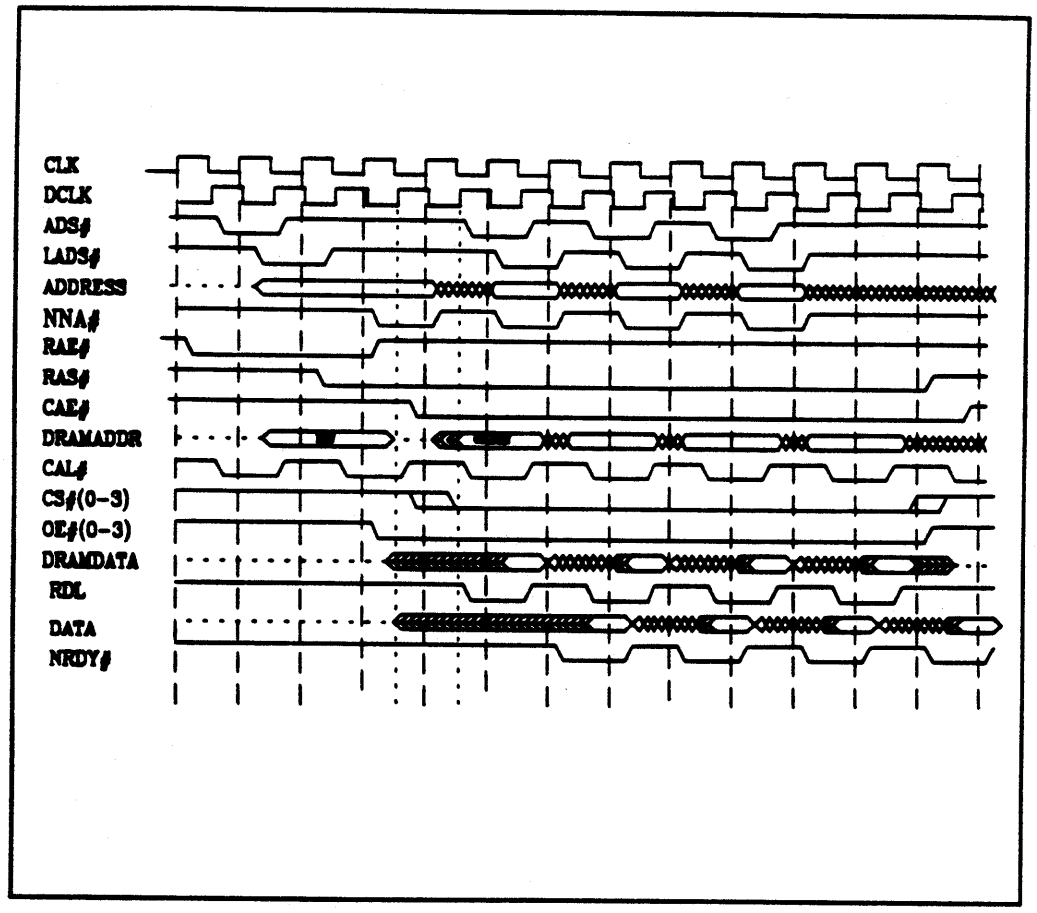


Figure 4.7 Pipelined Read Cycles

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 line fill are always within the same page, and the sequence shown in Figure 4.9 illustrates this case. Note that all cycles of subsequent cache line fills can be pipelined.

4.3.4 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 form the DRAMs. The processor may complete a cycle and begin another while the DRAM subsystem completes the WRITE.

A write data register is needed to implement posted write cycles. This register holds write data after NRDY# is returned to the processor. Write cycles can complete while the next cycle is in progress.

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.

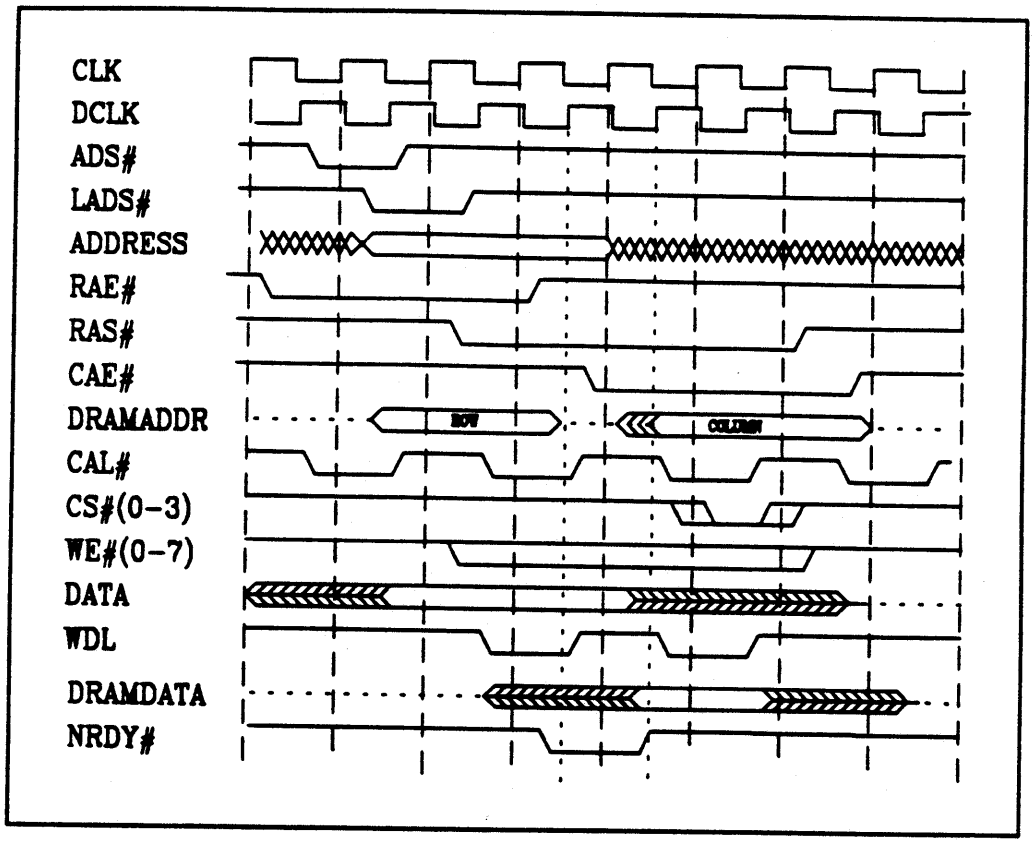


Figure 4.8 Basic Write Cycles

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#.

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.

4.3.4.1 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.

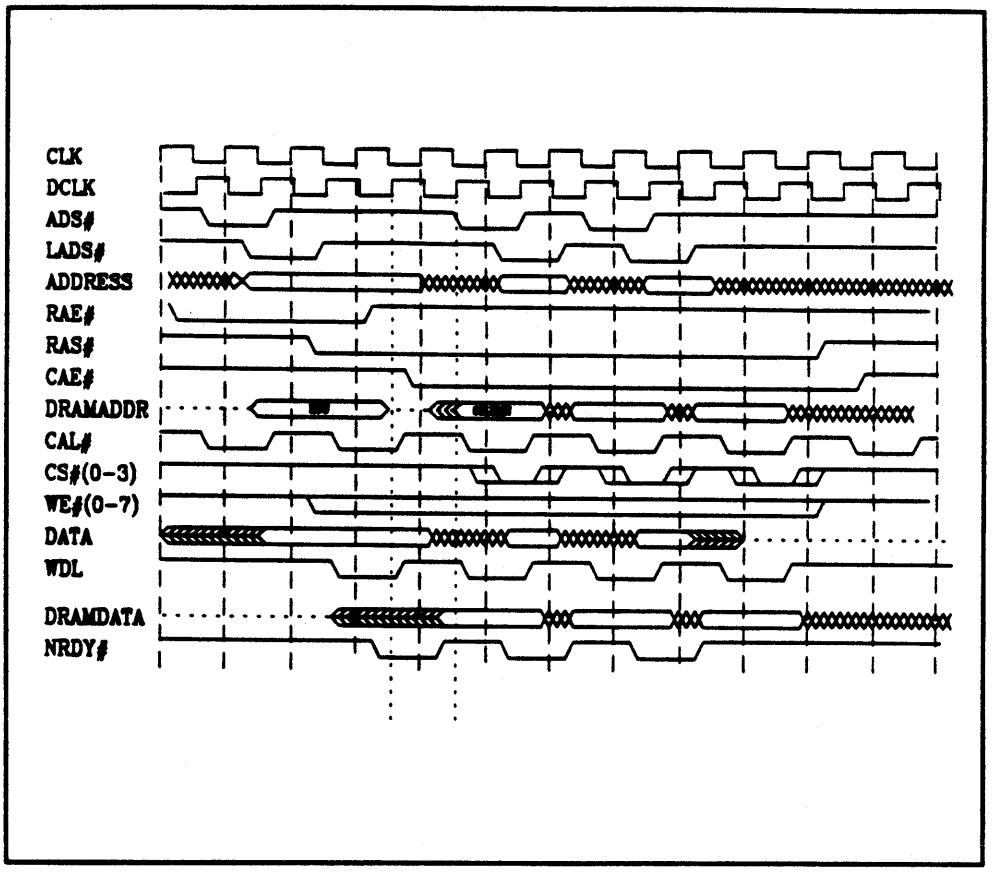


Figure 4.9 Consecutive Write Cycles

4.3.5 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.

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.

4.3.5.1 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 occured. After it returns NRDY# for the write cycle, however, the processor asserts ADS#. This signal starts the read cycle.

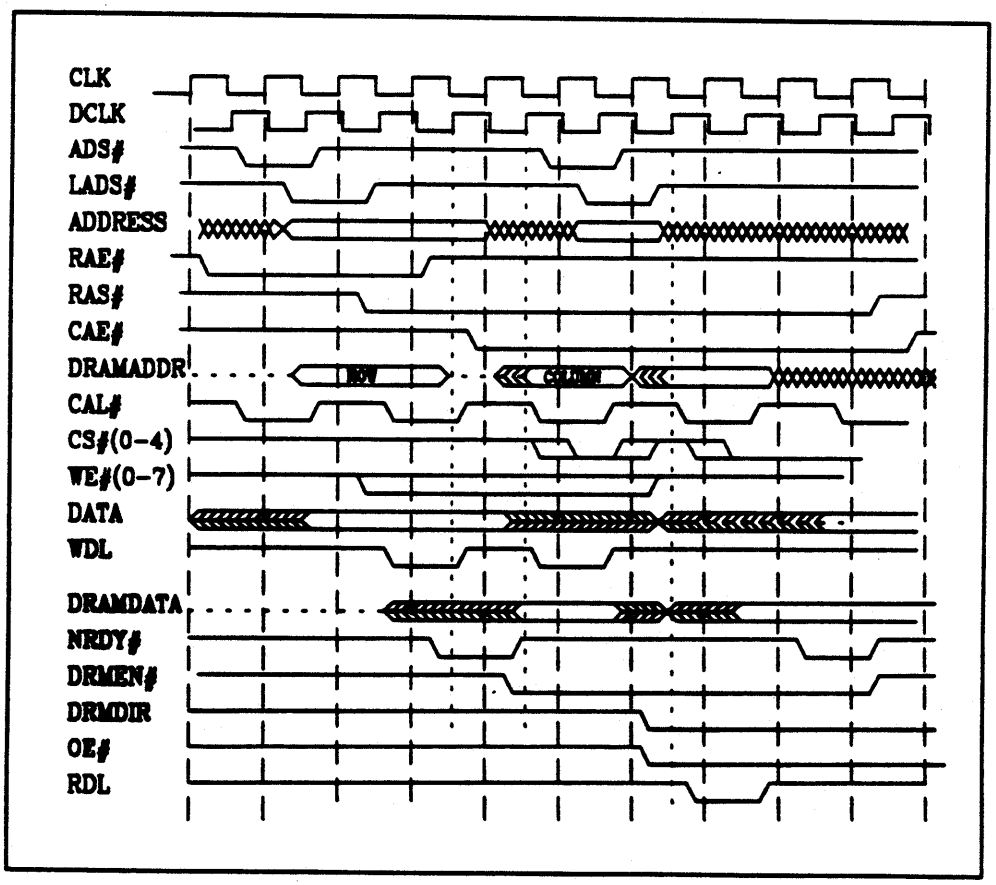


Figure 4.10 Write Followed By Read Cycles

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 PAL 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.

4.3.5.2 Read Cycles Followed by Write Cycles

Figure 4.11 illustates 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 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.

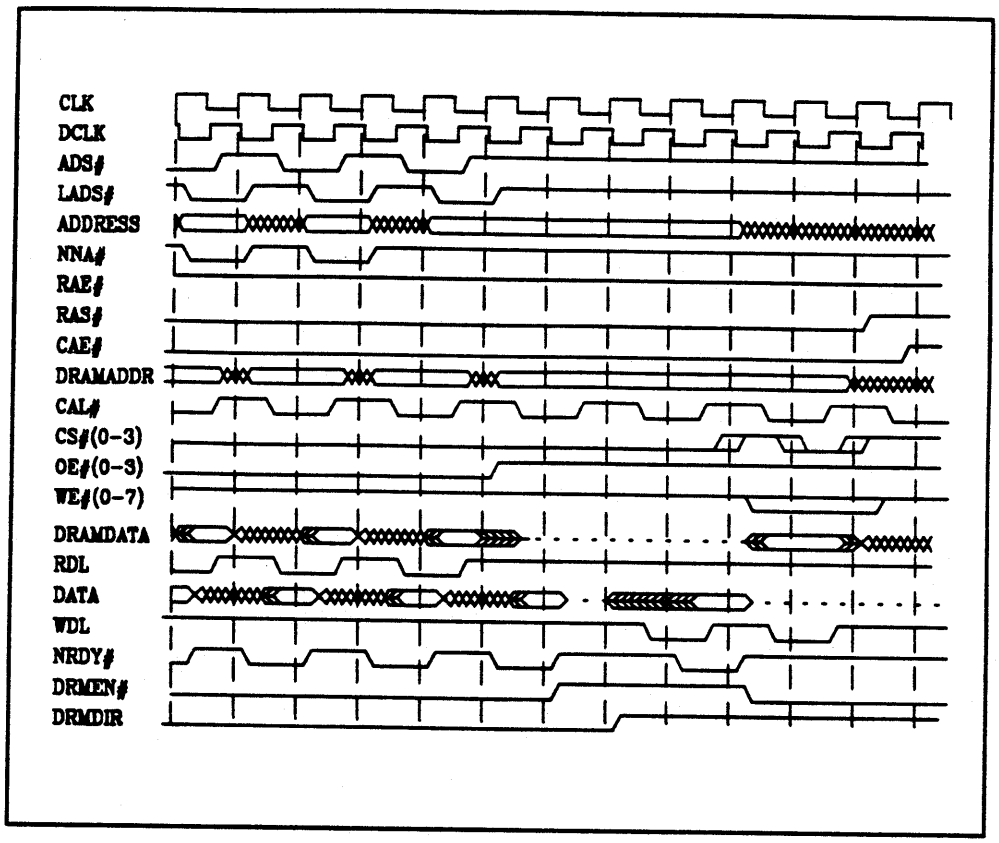


Figure 4.11 Pipelined Read Followed By Write Cycles

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.

4.3.6 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 or static column 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.

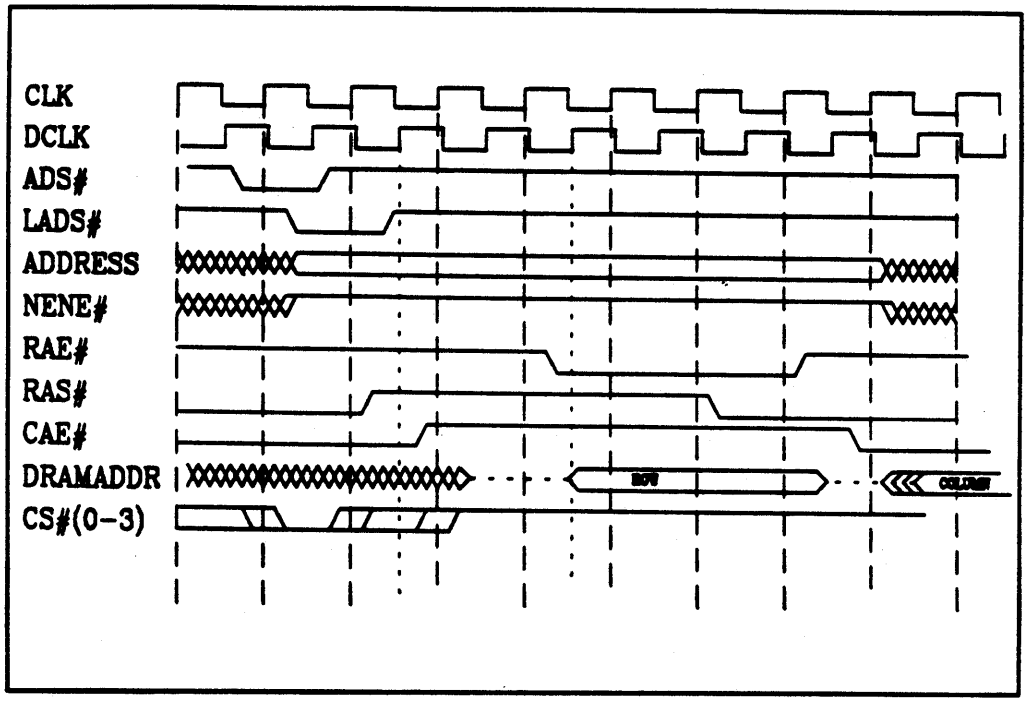


Figure 4.12 Page Miss Cycles

The RAS# signal is immediatly deactivated when a page miss is detected and is 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.

4.3.7 Refresh Cycles

Refresh cycles must be performed at regular intervals of approximately 500 milliseconds. 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.

4.4 Parity Circuit

4.4.1 Circuit Overview

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 PAL 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.

4.4.2 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.

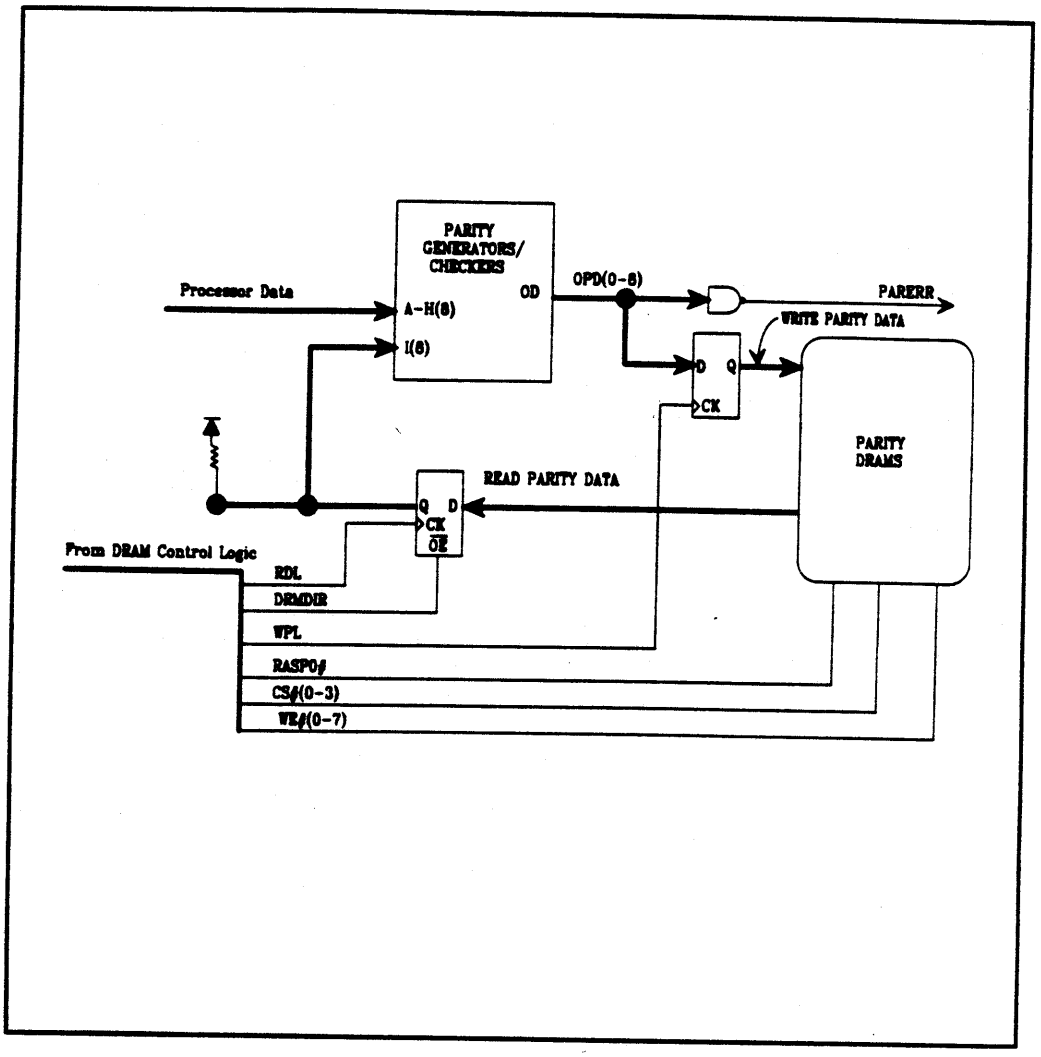


Figure 4.13 Parity Block Diagram

4.4.3 Parity Function

Parity data is latched during zero-wait-state writes. It is latched on the clock after which write data has been latched. The write data register is clocked by WDL in the 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 B 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 at 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.

The PARERR signal indicates that a parity error has occured. 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 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.

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 it is ready to terminate the processor cycle. Interface to all slave devices except DRAMs is very similar and less performance sensitive. Non-volatile 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.

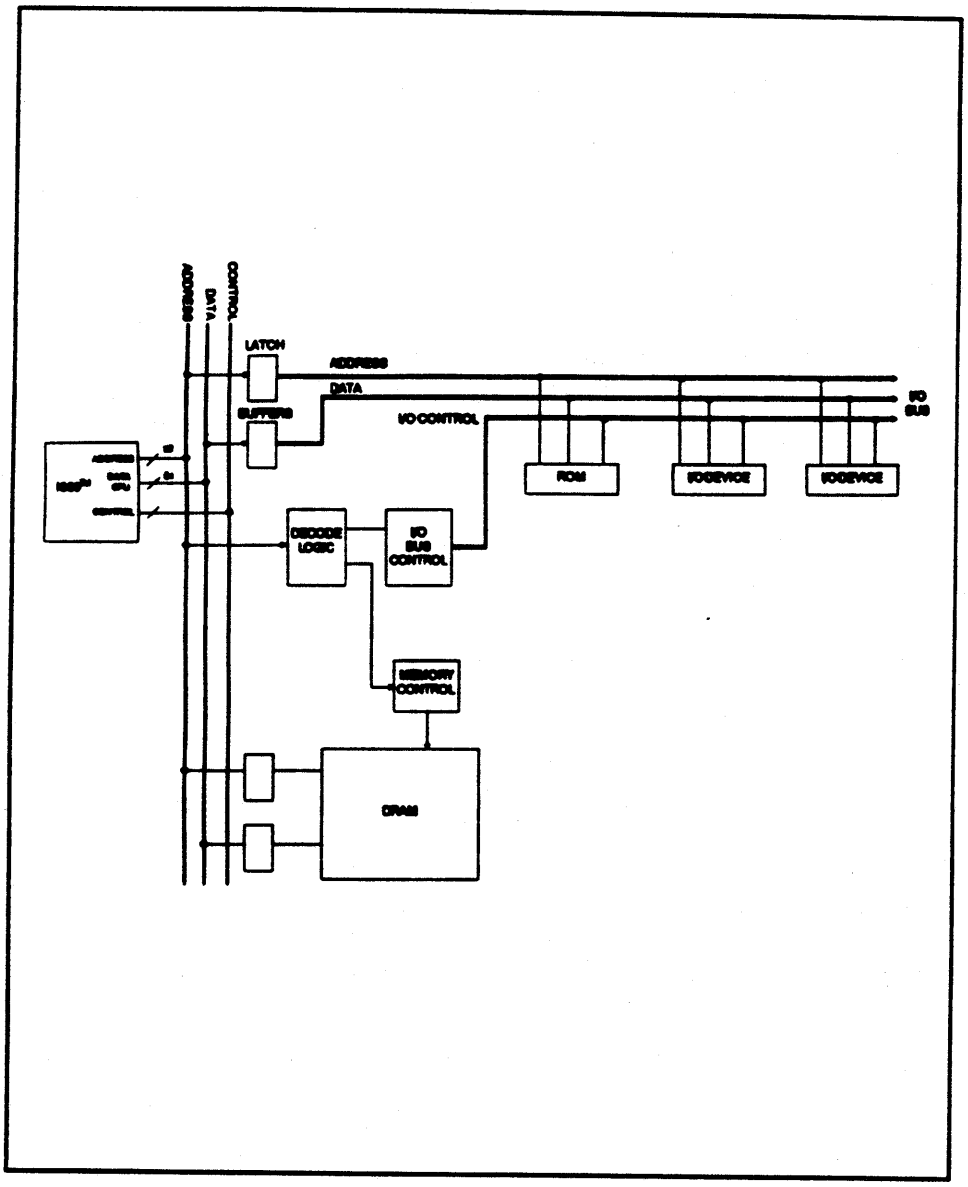


Figure 5.1 I860 Microprocessor System

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 placed on continuous byte boundaries; 16-bit devices can be placed on continuous even byte (word) boundaries; and 32-bit devices can be placed on continuous, multiple-of-four 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 and aborts the cycle.

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. A latched version of ADS# (LADS#) is valid in T11 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.

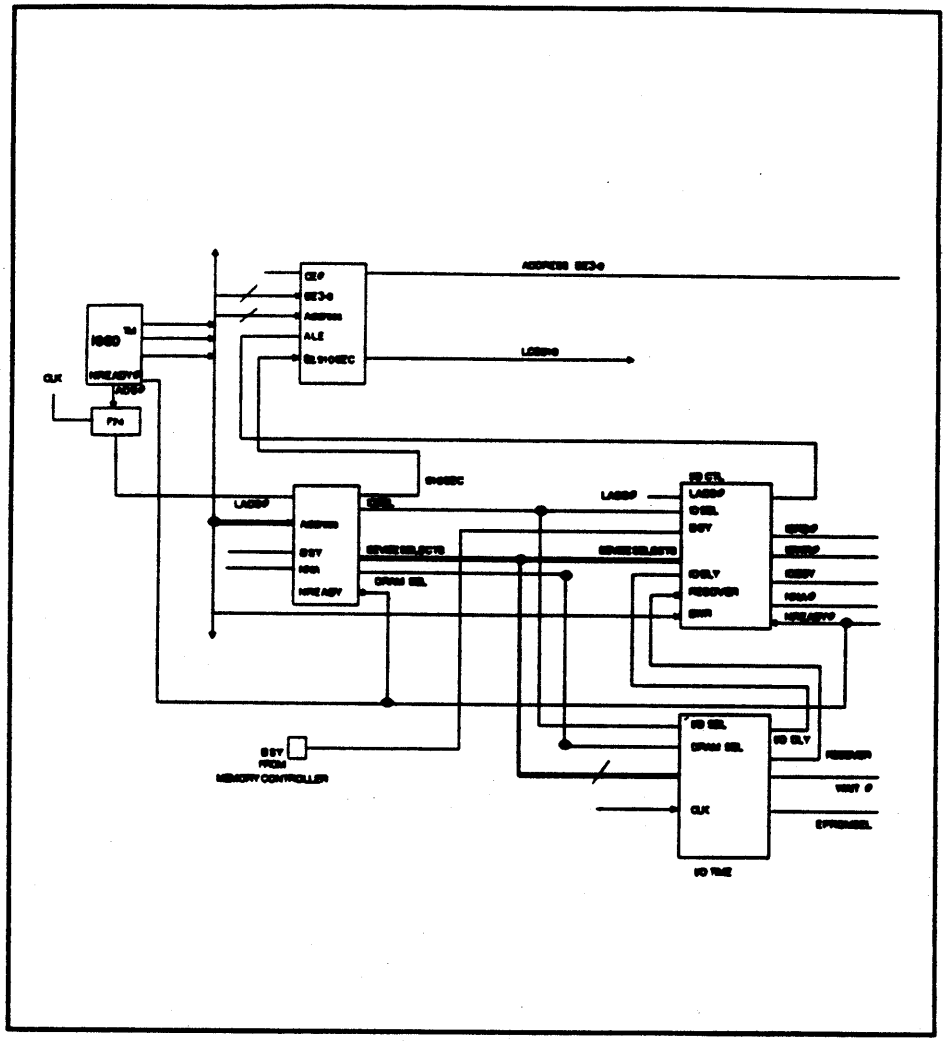


Figure 5.2 - I/O Control Logic

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 PAL 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 address changes and a new cycle begins.

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 accomodate 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 deasserting READY# on rising edge of T11. Since most I/O devices are much slower than the i860 processor processor bus, all I/O cycles require additional wait states. Since READY# can be driven by control PALs, it must be tri-stated by the PALs 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 PAL 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 Read. Some delay after the rising edge of Read 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 addressed. The I/O control logic implements recovery to eliminate bus contention. It asserts a 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 (i.e. when recovery time is in microseconds rather than in nanoseconds. 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 5.2. and 5.3 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 T1. The address and status signals become valid in T11. LADS# becomes valid in T11, and the address is latched on the next rising edge of clock (second T11). The I/O select signal is 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 T11 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 equations 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:
Chip select or address setup to IORD#.
IORD# minimum active width.
Chip select or address hold after IORD#.
IORD# active to data valid.
Data float after IORD# inactive

1. Chip Select/Address Setup to IORD#:
   >= 3Tcy + PALclk-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 T11.

2. IORD# Minimum Active Width:
   >= Tcy + PALclk-to-output - PALclk-to-output + nTcy
   >= Tcy + nTcy

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

3. Chip Select/Address Hold after IORD#:
   >= Tcy - PALclk-to-output + Address Validmin + Latch Prop. Delaymin

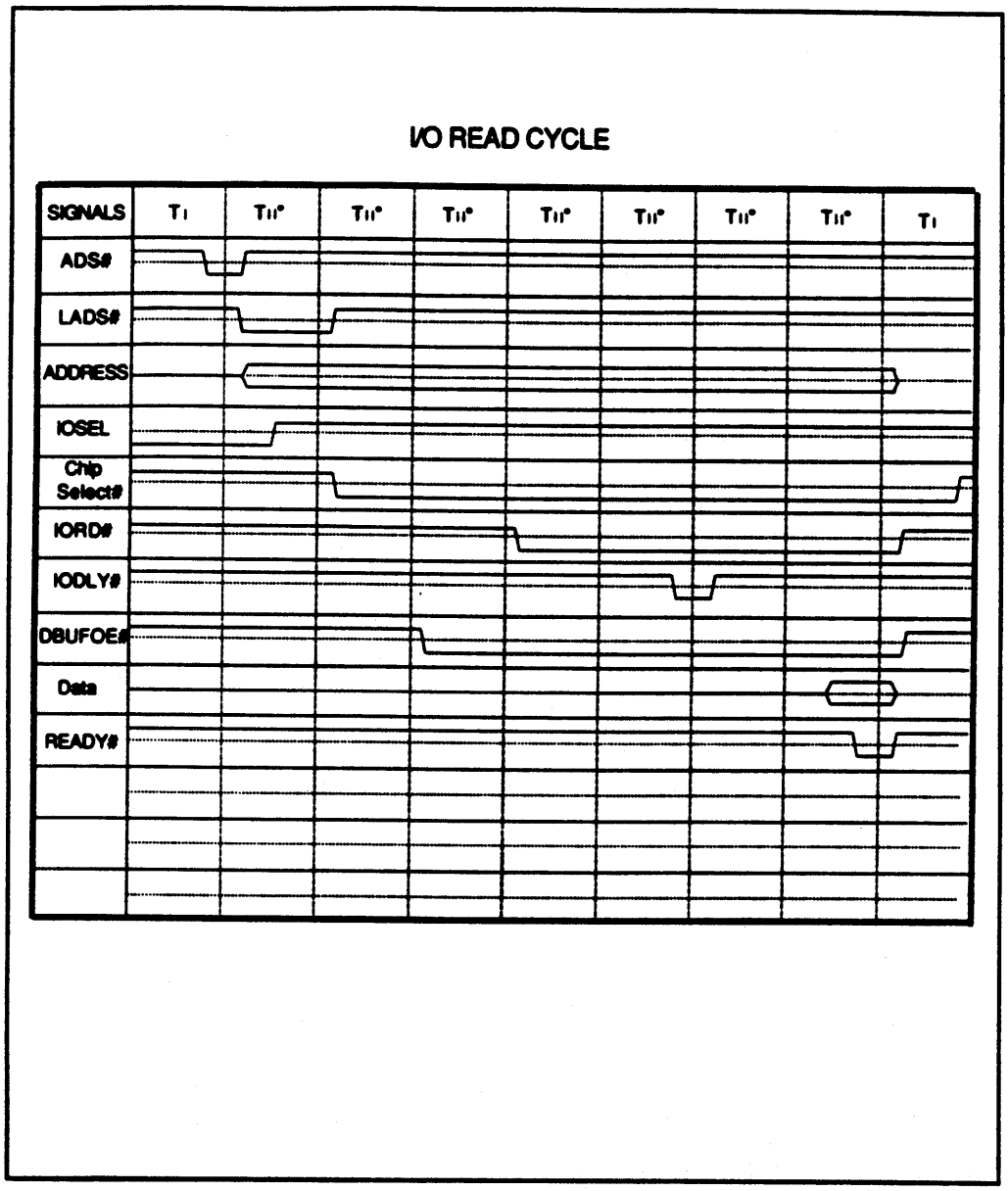


Figure 5.3 Read Cycle Timings

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 - PALclk-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 will not be a problem, since both chip select and address will be latched. The data float delay in slave devices can be very long (40 ns). 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 T11. 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:

Write (IOWR#) Active Pulse Width
Address/Chip Select Set-up to Write (IOWR#) Active
Data Set-up to Write (IOWR#) Inactive
Data Hold Time After IOWR# Inactive

I/O Write Timing - Figure 5.4

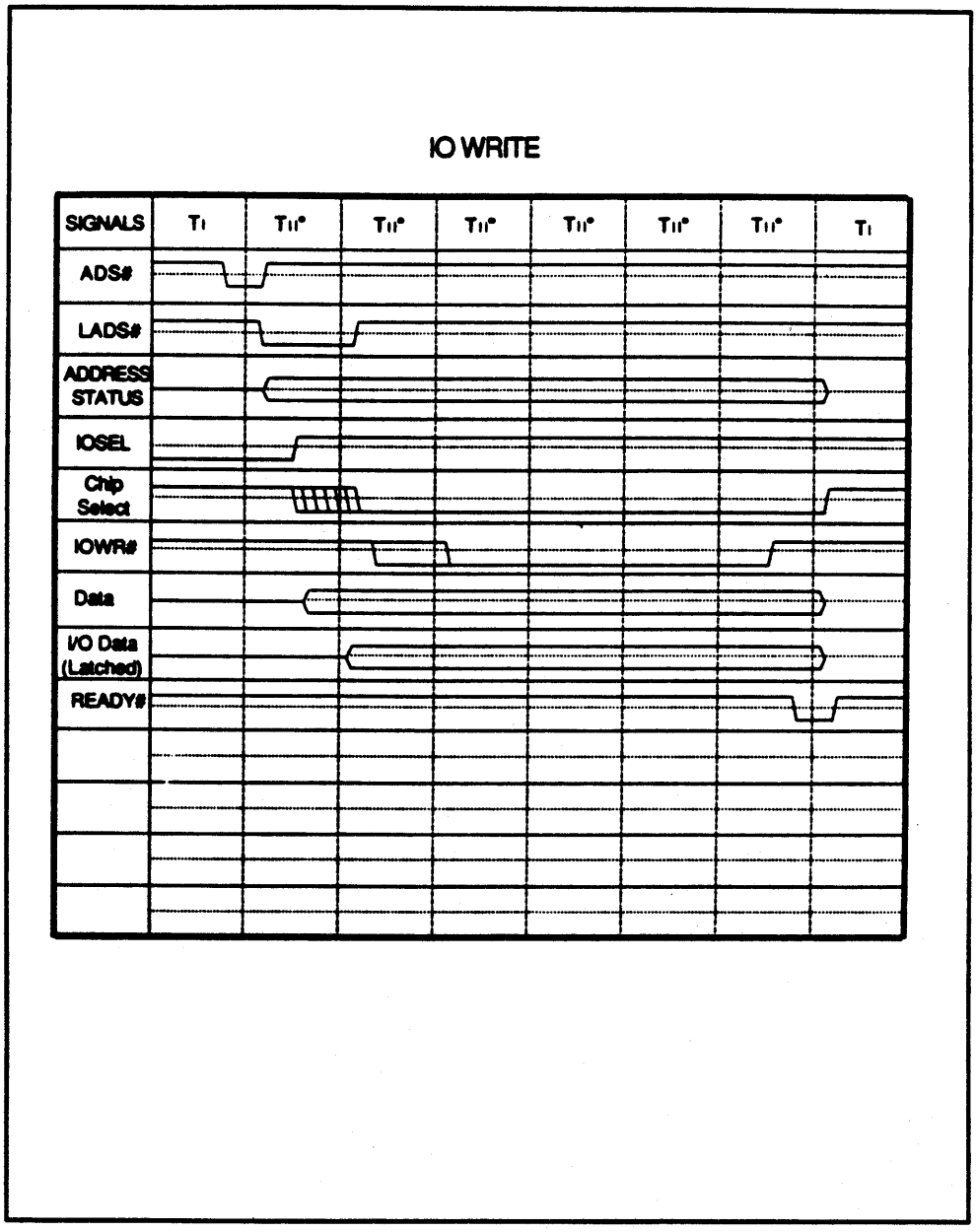


Figure 5.4 I/O Write Timings

1. IOWR# Active Pulse Width:
   >Tcy - PALclk-to-output- PALclk-to-output - PALclk-to-output + nTcy
   > = Tcy + nTcy

   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 + PALclk-to-output(min) - Decode delaymax - Address Valid delay max - 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 T11.

3. Data Setup To Write Inactive:
   > = 4Tcy - 860 Data Valid Delaymax - Data Buffer Prop. Delaymax + PALclk-to-output Delay

This Equation assumes a zero-wait-state write cycle. Write is not posted. PALclk-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 will not be 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 07000000 through 07FF0000. The ports are located on 64-bit boundaries to allow data to be read on D7-0 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 07000000 - 0700000C. The address decoder generates the IOSEL as well as the 82510SEL signal. The address and chip select are 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 PAL. Critical timings of the 82510 are the following:

Read Cycle

1. Address Valid to Read Active (Tavrl)
2. Command (IORD#) Access Time to Data Valid
3. Command (IORD#) Active width.
4. Command (IORD#) Inactive to Active (Recovery Time)
5. Command (IORD#) Inactive to Data Float Time.

1. Address Valid to Read Active (TAVRL):

Tavrl < = 3Tcy + PALclk-to-output(min) - Decode Delaymax - Address Valid delaymax - Latch Prop. Delaymax

7ns < = 3Tcy - 25
< = 50 ns (@40MHz)

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

Trldv < = Tcy - PALclk-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 < = 123 ns.
< = 4Tcy - PALclk-to-outputmax + PALclk-to-outmin
< = 4Tcy - 10 + 2
< = 4*31 - 8
< = 116 ns

The i860 microprocessor 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

1. Address Valid to Write Low (Tavwl)
2. Write Active Width
3. Data valid to Write Inactive. (TDVWH)
4. Data Hold Time after Write Inactive (TWHDX)

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 + PALclk-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.

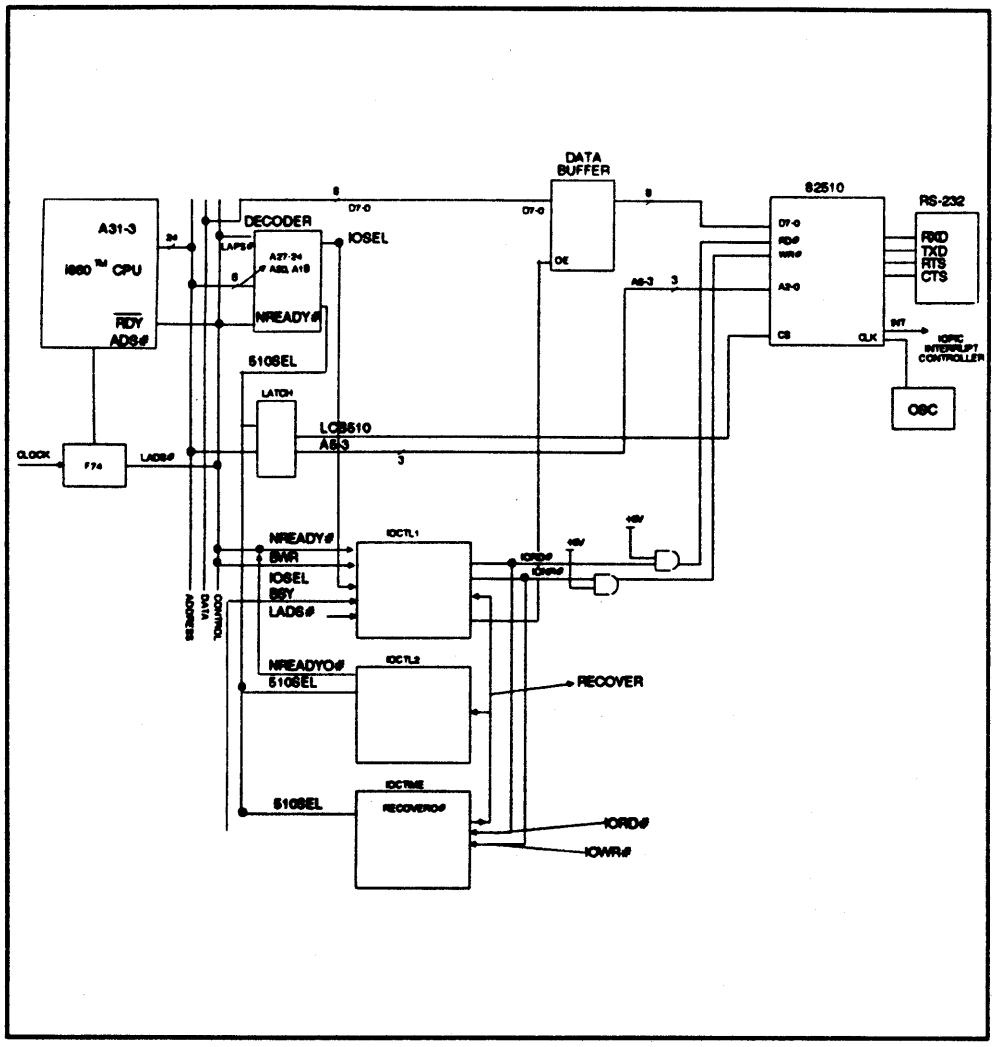


Figure 5.5 - i860 Processor Interface to 82510

5.4.2 EPROM INTERFACE

The i860 CPU supports a special CS/8 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#-0# pins behave as A2-0. Once the i860 microprocessor boots, the CS/8 mode can be disabled by setting the CS/8 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 D7-0 without using byte swap logic. It does, however, require the address multiplexor to use either the processor address bus or the byte enables for A2-0 inputs of the EPROM. A method which does not require an address mux is the double load copy method. This method is more dependent upon software, however. It uses the BE2#-0# signals which are connected directly to the EPROM A2-0 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#-0# in CS8 mode and Processor addresses in data mode. 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 A2-0 are multiplexed between BE2#-0# 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 D7-0 is connected to the lower byte (D7-0) of the processor data bus. This requires that the EPROM when mapped as data must be on 64-bit boundaries.

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 A2-0 are directly connected to the BE2#-0#. The 8-bit data bus of the EPROM is connected to D7-0 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 processor. 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 1 illustrates, in the non-CS8 mode there are only six valid BE2#-0# 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 1: Valid Addresses for SPACED copying

Bus Address		BE#s 210	Valid Address T or F		EPROM Address
0	000	T	(Load 32-bit value		0
1	001	F	(16-bit access on odd boundary)		Invalid
2	010	F	(two non-contiguous bytes)		Invalid
3	011	T	(Load 1 byte)	3	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#-0# 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.

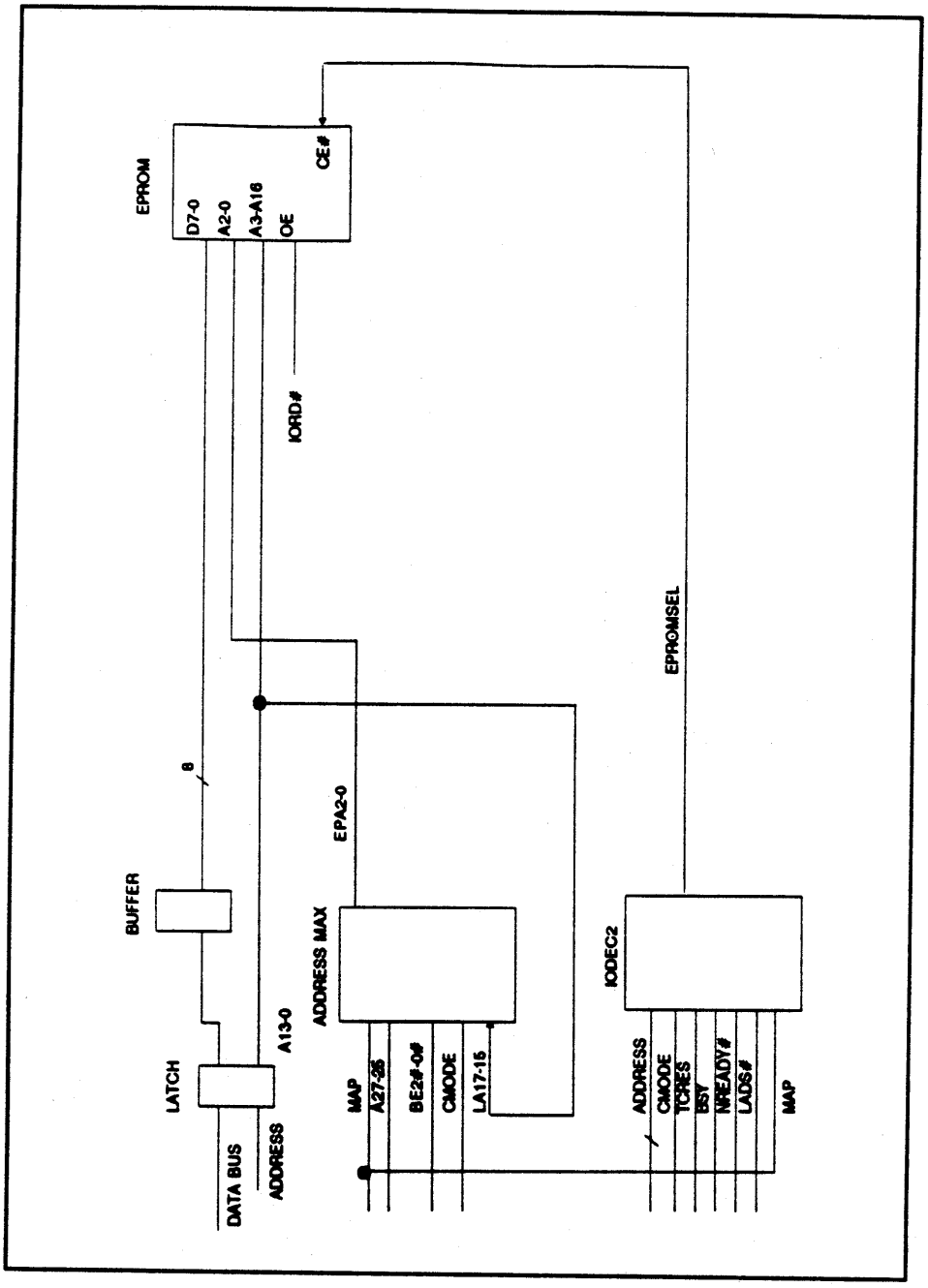


Figure 5.6 - i860 EPROM Interface

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
2. OE# to Output
3. OE# High To Output Float

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 200 ns @33MHz Two Wait States

Addr. Mux. :Address Multiplex Delay.

2. OE# To Output (TOE):

Assumes zero wait state cycle.

Toe < = Tcy - PALclk-to-output - Buffer Delay - i860 uP Setup

Toe < = Tcy - 28

For Toe of 85 ns 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 cam float as late as 60 ns after the rising edge of IORD#. This means that EPROM data may be valid until as late as 10 ns into the second T11 of the next cycle. The i860 CPU for writes can output data as early as 3.5 ns from clock edge of the second T11. If this were allowed to happen

it would result in data bus contention and will cause problems in the system. The IOTIME PAL 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 convential 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 i860 processor 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. 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, 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 address if that data stream crosses a memory page boundary.

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 2x clock. A 16 Mhz 82380 requires the i860 microprocessor 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 PALs used to convert the 82380 bus signals into DRAM commands. The i860 processor HOLD/HLDA signals are used. Each transfer will require 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 will automatically switch to the next set of registers. The registers must be reprogramed during the time of a page transfer. If the operating system can not guarantee quick enough interrupt response, DMA transfers can not cross a page boundary.

The second approach is to add a special I/O buffer memory between the I/O device and the i860 processor. 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 simulataneous 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 for several reasons: the DMA channels do not tie up the CPU bus when accessing the I/O device, The SRAM read/write and DRAM write/read occurs faster than the DMA doing it because the i860 processor does these faster than the 82380 and more DRAM page hits will happen when properly programmed. The DMA transfers can be longer since the I/O buffer is contiguous. An entire disk track could be read into the SRAM area. The i860 processor caches need not be flushed because of an I/O transfer.

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 from DRAM to SRAM, use pfld instructions to prevent the data from flushing the data cache. Normal writes can be used since a write miss does not cause a cache load. The SRAMs will appear on only the lower 16/32 bits of the data bus. The copy routine must increment its address accordingly. The copy function shown here acheive 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.

```
//
//      The following code copies from the I/O buffer to regular memory.
//      The address of the buffer in I/O memory is in r16.
//      The destination address is in r17, it must be 8 byte aligned.
//      The count of 32-bit words to transfer is in r18.
//      Perform four back-to-back DRAM cycles for page mode speedups.
//
getiodata:
        or      r0,-1,r21           // Set count value
        addu    r18,-9,r18          // Setup for BLA
        shr     r18,3,r20           // Divide count by 8, don't change CC
        jnc     gshort              // Jump if short count

        or      r0,8,r19            // Set I/O address increment
        bla     r21,r18,gloop       // Setup for loop
        subu    r16,r19,r16         // Setup for autoincrement loop
gloop:
        ld.l    r19(r16++),f16      // Get an I/O value
        ld.l    r19(r16++),f17      // Get an I/O value
        ld.l    r19(r16++),f18      // Get an I/O value
        ld.l    r19(r16++),f19      // Get an I/O value
        ld.l    r19(r16++),f20      // Get an I/O value
        ld.l    r19(r16++),f21      // Get an I/O value
        ld.l    r19(r16++),f22      // Get an I/O value
        ld.l    r19(r16++),f23
        addu    r17,32,r17          // Update destination address
        st.d    f16,-32(r17)        // Put into DRAM space
        st.d    f18,-24(r17)        // Put into DRAM space
        st.d    f20,-16(r17)        // Put into DRAM space
        bla     r21,r18,gloop       // Loop till done
        st.d    f22,-8(r17)         // Put into DRAM space

        and     r18,7,r18           // Get remaining count
        addu    r18,-9,r18          // Setup for final test
        subu    r16,r19,r16         // Restore source address
gshort:
        addu    r18,8,r18           // Setup for BLA
        jnc     exit                // Jump if zero count

        bla     r21,r18,l1          // Setup for loop
        or      r0,8,r19            // Set I/O address increment
l1:
        ld.l    (r16),f16           // Get an I/O value
        addu    r16,8,r16           // Update source address
        addu    r17,4,r17           // Update destination address
        bla     r21,r18,l1          // Loop till done
        st.d    f16,-4(r17)         // Put into DRAM space
exit:
        bri     r1                  // All done
```

```
        nop                             // Nothing to do here

//
//      The following code transfers data from DRAM space to the I/O space.
//      I/O buffer is only on lower 32-bits of bus.
//      The memory source address is in r16, it must be 8 byte aligned.
//      The destination I/O buffer address is in r17.
//      The count of 32-bit words to transfer is in r18.
//      Use pfld instructions to avoid flushing the data cache.
//      Perform four DRAM reads back to back to run page mode accesses.
//
putiodata:
        or      r0,-1,r21               // Set loop counter
        addu    r18,-9,r18              // See if short count, can't do pflds
        shr     r18,3,r19               // Form 32 byte block count in r19
        jnc     pshort                  // Jump if short count

        or      r0,8,r20                // Set address increment
        subu    r16,r20,r16             // Setup for autoincrement addressing
        pfld.d  r20(r16++),f0           // Start getting source data
        pfld.d  r20(r16++),f0           // Start getting source data
        bla     r21,r19,ploop           // Setup loop counter
        pfld.d  r20(r16++),f0           // Start getting source data
ploop:
        pfld.d  r20(r16++),f16          // Get memory data
        addu    r17,64,r17              // Update destination buffer address
        pfld.d  r20(r16++),f18
        pfld.d  r20(r16++),f20
        pfld.d  r20(r16++),f22
        fst.l   f16,-64(r17)            // Put into I/O buffer
        fst.l   f17,-56(r17)
        fst.l   f18,-48(r17)
        fst.l   f17,-40(r17)
        fst.l   f18,-32(r17)
        fst.l   f17,-24(r17)
        fst.l   f18,-16(r17)
        bla     r21,r19,ploop           // Loop
        fst.l   f19,-8(r17)             // Put into I/O buffer

        pfld.d  -8(r17),f16             // Copy items, read old I/O data
        and     r18,7,r18               // Mask out odd count value
        fst.l   f16,(r17)
        fst.l   f17,8(r17)
        pfld.d  -8(r17),f16             // Copy items, read old I/O data
        addu    r18,-9,r18              // Adjust remaining count for pshort
        fst.l   f16,16(r17)
        fst.l   f17,24(r17)
        addu    r16,8,r16               // Bump source pointer
        pfld.d  -8(r17),f16             // Copy items
        addu    r17,48,r17              // Update destination counter
        fst.l   f16,-16(r17)            // Store items in I/O buffer
```

```
        fst.l   f17,-8(r17)
pshort:
        or      r0,4,r20            // Set source increment
        addu    r18,8,r18           // See if count was zero
        jnc     exit                // Jump if zero count

        bla     r21,r18,psloop      // Setup for inner loop
        subu    r16,r20,r16         // Setup for autoincrement addressing
psloop:
        ld.l    r20(r16++),f16      // Get source
        addu    r17,8,r17           // Bump destination
        bla     r21,r18,psloop      // Loop
        st.l    f16,-8(r17)         // Write to I/O buffer

        bri     r1                  // Return to caller
        nop                         // Nothing to do here
```

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 machine 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 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 Performance

The i860 microprocessor is capable of over 500,000 complete transformations per second at 40 MHz. Each transformation includes translation, rotation, scaling and perspective computation. The processor can generate more than 50,000 Gouraud shaded, 100-pixel triangles per second for an 8-bit pixel resolution.

6.2.2 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 megabytes per second bandwidth must be dedicated to memory refresh to provide real-time graphics on a 1,280 x 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 is 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 processor based system. The processor connects to the 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 x 768 RGB Display with 16-Bit Pixel Resolution
Double-Buffering at Two Megabytes Per Buffer
33 MHz with 40 MHz Upgrade Path
PAL-Based VRAM Controller
VRAM Control Option for 100ns and 80ns VRAMs
PAL-Based Non-interlaced CRT Control

Dedicated to the three colors, red, green and blue are 6, 6, and 4 bits per pixel respectively. 16-bit pixel resolution allows a double-buffered scheme requiring only four megabytes of total display memory for a 1,024 x 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 with 24-bit or 32-bit pixel resolution.

6.3.2 Testing

This example has not been tested, and no PAL codes are included.

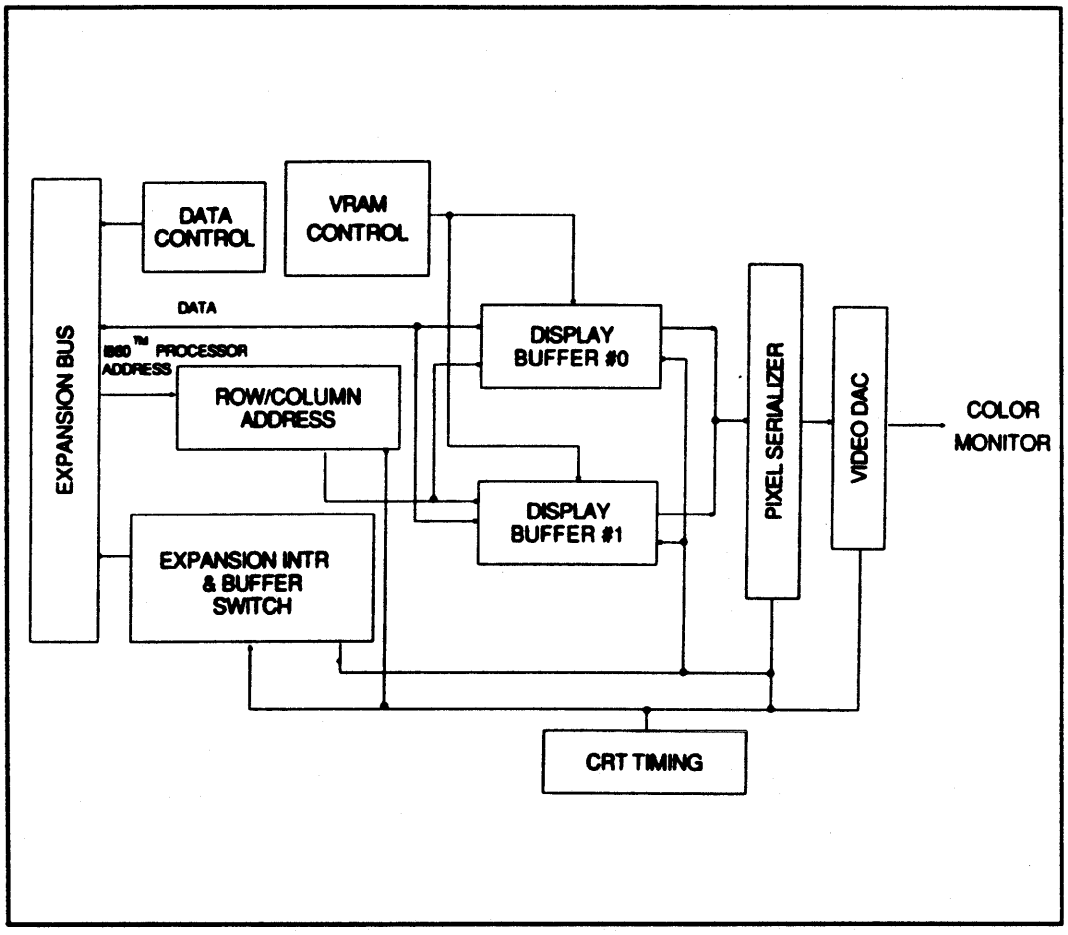


Figure 6.1 Block diagram of the i860 processor based graphics frame buffer board.

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 (please 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 i860 microprocessor 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 CPU core such as EXPSEL#
Power supply and ground signals

Please refer to schematics for the complete list of signals.

6.4.2 Data Transceiver/Latch Control

Data is transferred to and from the i860 microporcessor through data transceivers/latches in the core. 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 CPU core.

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

When ready to display data in a second buffer, the processor may read or write to a corresponding location in the unused area of the expansion space (A22 being 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 PAL/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. Sync information is also embedded in analog output. The sync-on-green mode of the color monitor provides synchronzation.

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 in initial prototyping. They require one wait-state for writes and two wait-states for reads in page mode. 80ns VRAMs zero wait-states for writes and 1 wait-state for reads. Please refer to the timing diagrams in Figure 6.2 and 6.3 for zero wait-state write mode operation.

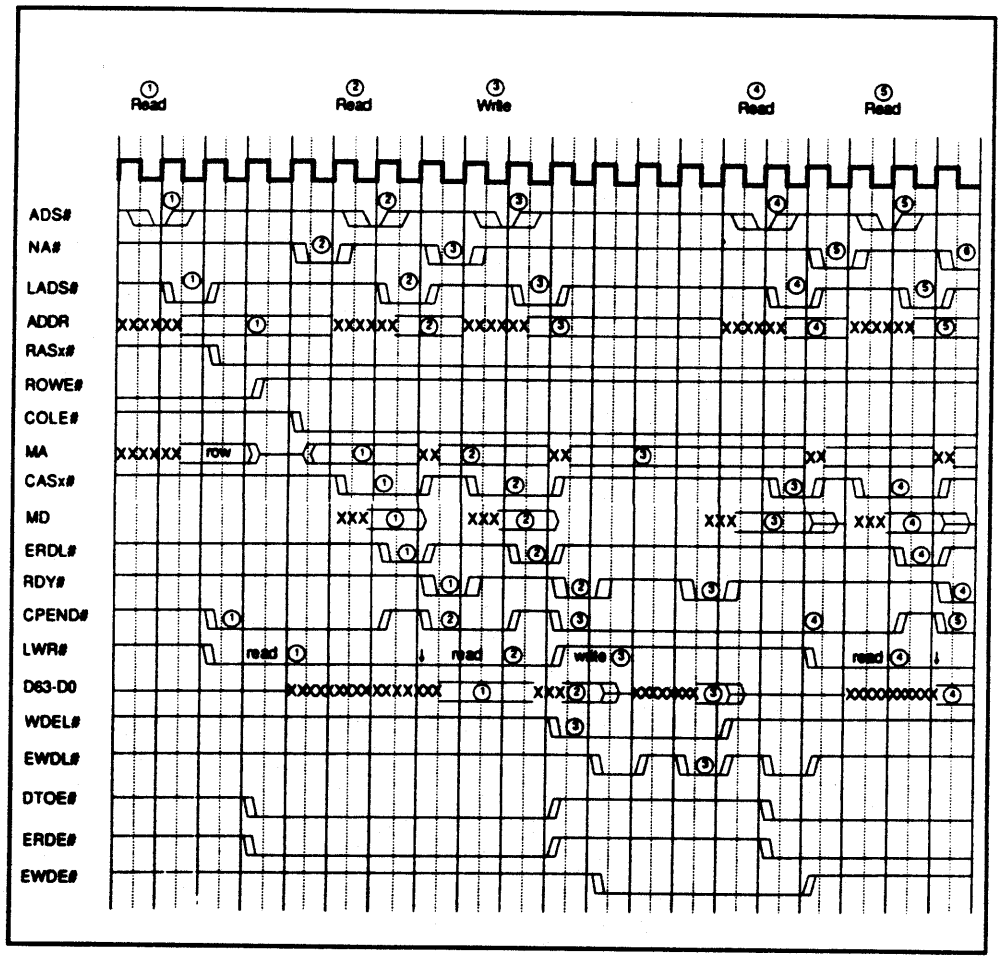


Figure 6.2 Read/Write/Read Operation (0 wait state write)

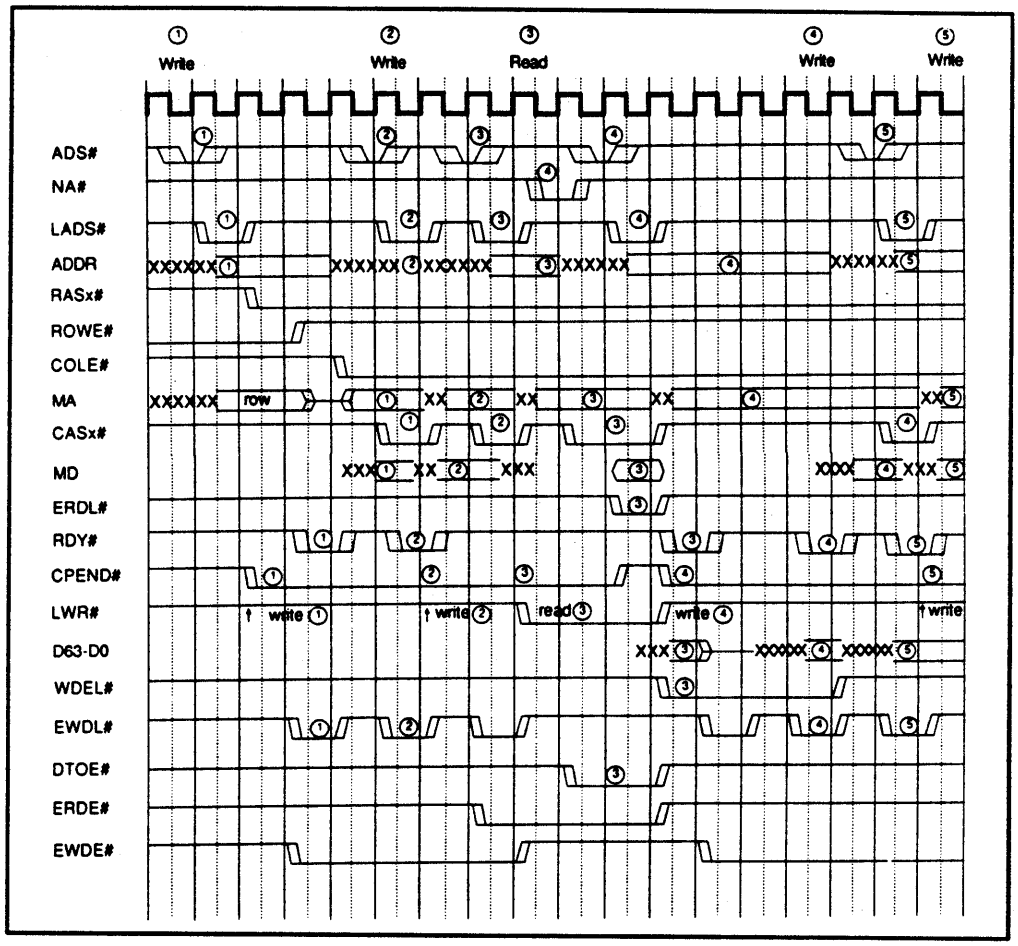


Figure 6.3 Write/Read/Write Operations (0 wait state write)

6.5.1.2 Processor-Initiated Cycles

The frame buffer board shares processor and latched data control buses with the CPU core. 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 non-pipelined mode. Idle

or far cycles put VRAMs in precharge mode. Row addresses are 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 supply column addresses only. Near cycles are indicated by the i860 microprocessor NENE# pin.
In read cycles, data is latched by the data latch in the core 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 control is controlled by expansion bus data control lines. Latch latency requires that write data be latched to maintain zero wait-state operation in successive VRAM write cycles (one wait-state is adequate in slow mode). Latch signals require at least one clock period to recover. This adds complexity in some situations as when a single idle cycle precedes write.

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.

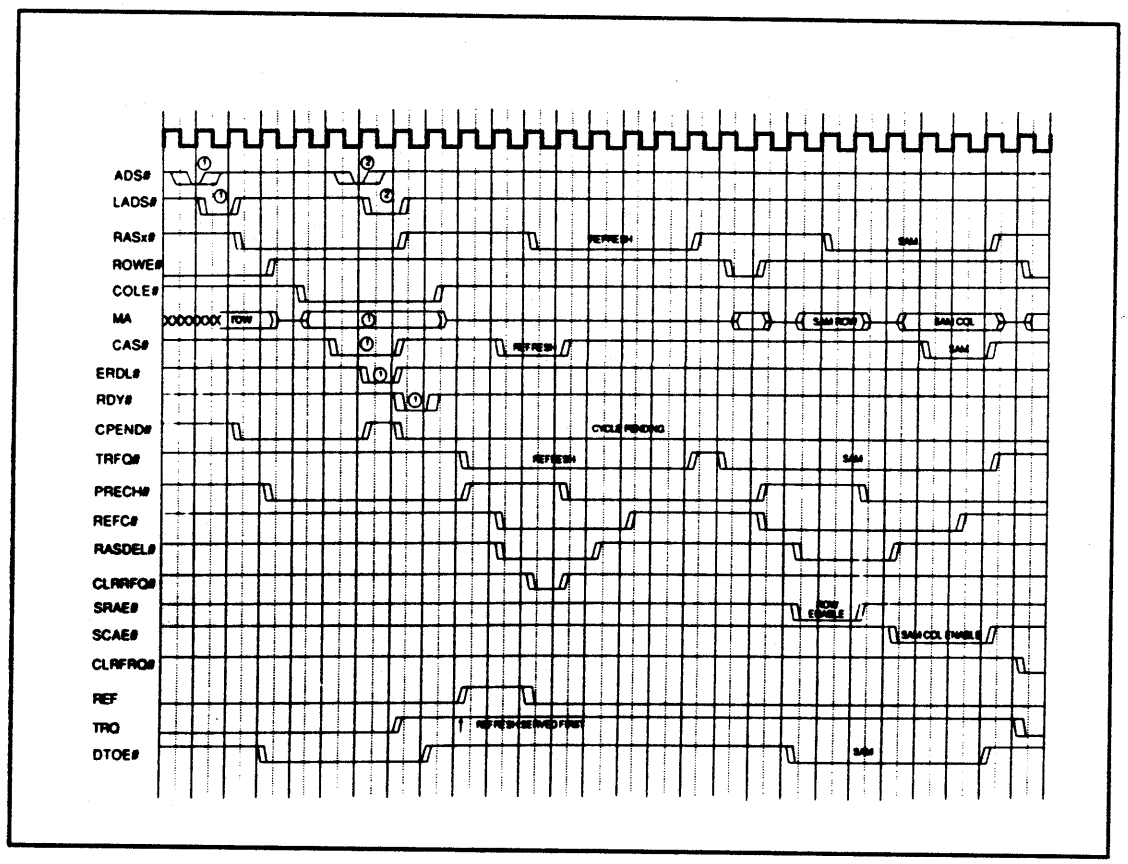


Figure 6.4 Refresh/RAM-to-SAM Transfers

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 when 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.

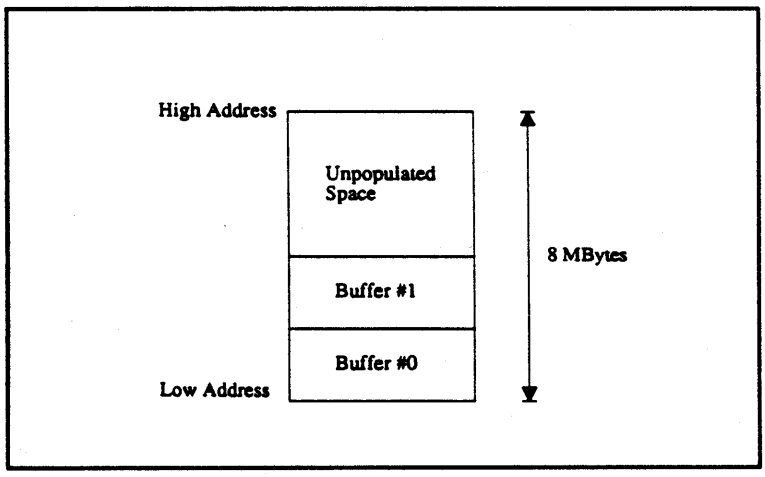


Figure 6.5 Expansion Space

6.5.2 Expansion Bus Interface

The expansion bus allows the i860 processor to access the frame buffer from the core. 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#), A21 and A22 are needed to decode access to the expansion address space in two megabyte increments (please refer to Figure 6.5). Low levels on EXPSEL# and the 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# 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 the delay clock(delayed from CLK) on the CPU core. Data is latched in the latter part of the processor clock period. EWDL# activates the write data latch signal on alternating clocks at the fastest rate. To support zero-wait-state, consecutive cycles in fast mode (low on WAIT), EWDL# is generated blindly. It latches write data on the assumption that the next write is a write to the VRAMs: if not,latched data is not used.

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 16.5us while inactive times are set at 4.5us. The horizontal sync signal, HSYNC# has a front porch of 1.0us, an active time of 2.0us and a back porch of 1.5us. The horizontal frequency is approximately 48 KHz. These timings are adjusted for use with a different monitor.

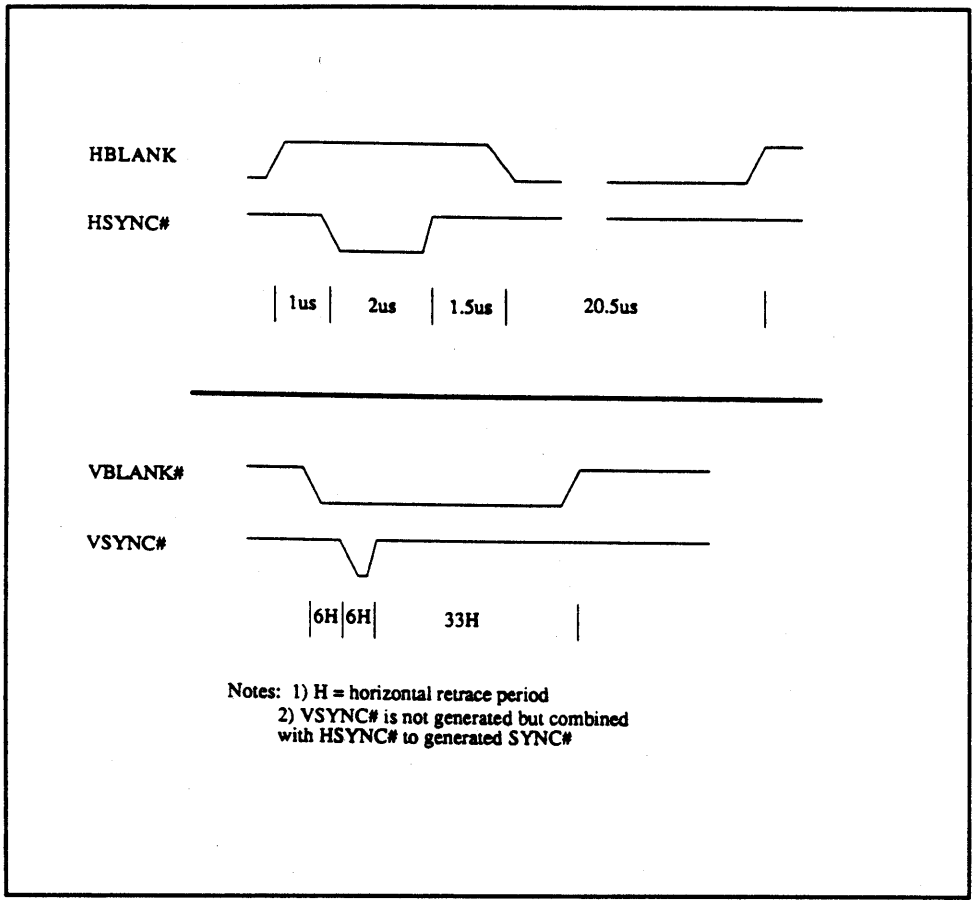


Figure 6.6 Blank and sync signals

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 six, six and 33 HBLANK periods respectively (please 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.

CRT timing relationships are illustrated in Figure 6.8. The horizontal blank (HBLANK) is generated by counting COUNT, and the 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.

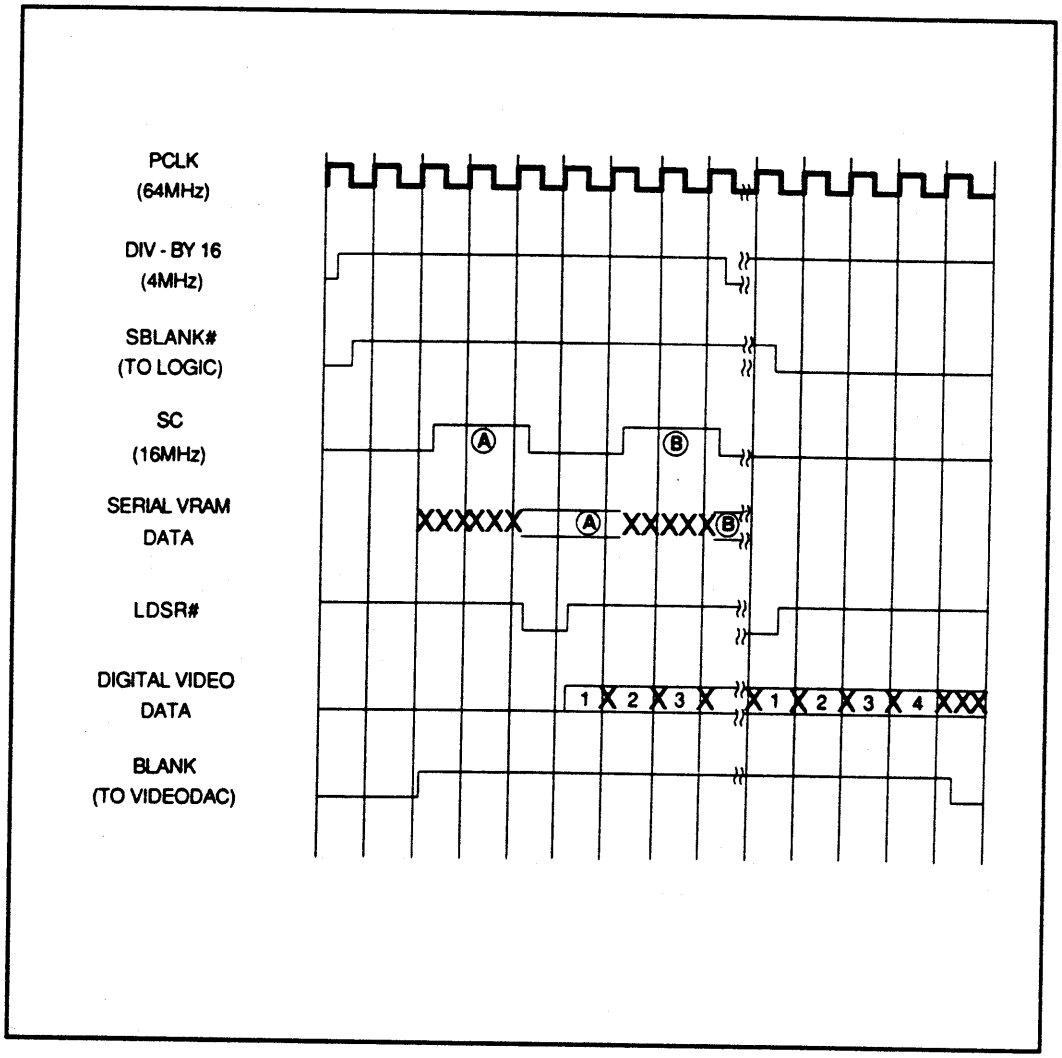


Figure 6.7 Serial Data Clocking

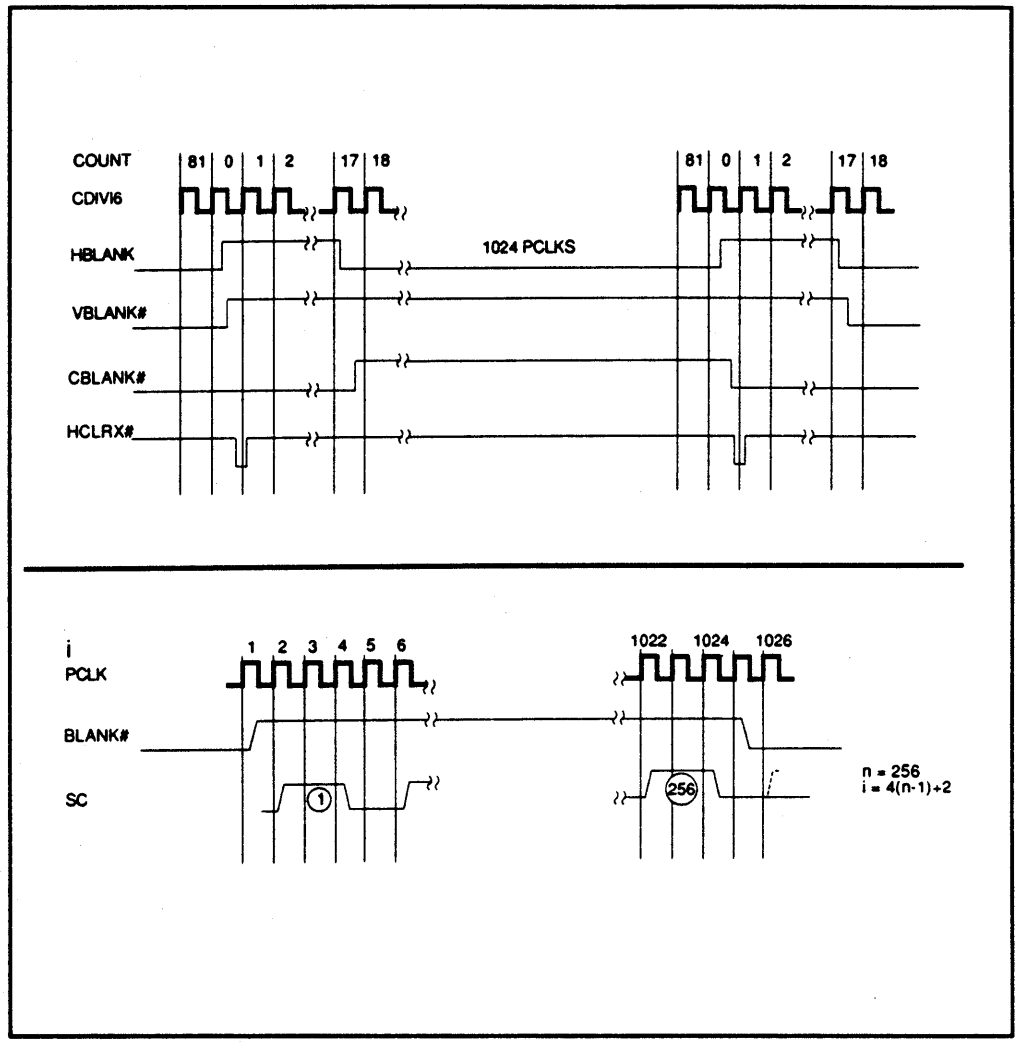


Figure 6.8 Blank Signals

6.6 Appendix

Refer to Appendix A for the schematics.

6.6.1 Schematics Description

Sheet #1
This sheet contains address transceivers and latches.

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 low; RAS0# is activated when A21 is high. 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) PAL.

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 XCVRs in the CPU core. Please refer to timing diagrams in the Appendix 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 in the core logic.

Sheet #12-15:
Terminations for VRAM control and for the expansion bus CLK signals (CLKC and CLKG). Test points are needed for surface-mounted device signals.

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

GRAPHICS SUBSYSTEM EXAMPLE

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

MULTIBUS®II and the i860™ Microprocessor 7

CHAPTER 7

MULTIBUS® II AND THE 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. MULTIBUS II boards based on the i860 microprocessor can be added to existing systems or can be the basis for a new system design.

7.1 i860 MICROPROCESSOR CPU BOARD

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

* MULTIBUS II Interface
* i860 Microprocessor
* 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 STANDARD

The MULTIBUS II standard is a processor-independent bus architecture with full distributed multiprocessor support. The standard defines a 32-bit parallel system bus (PSB) 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 (PSB) handles interprocessor communications and accesses to I/O devices not dedicated to a single CPU board.

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

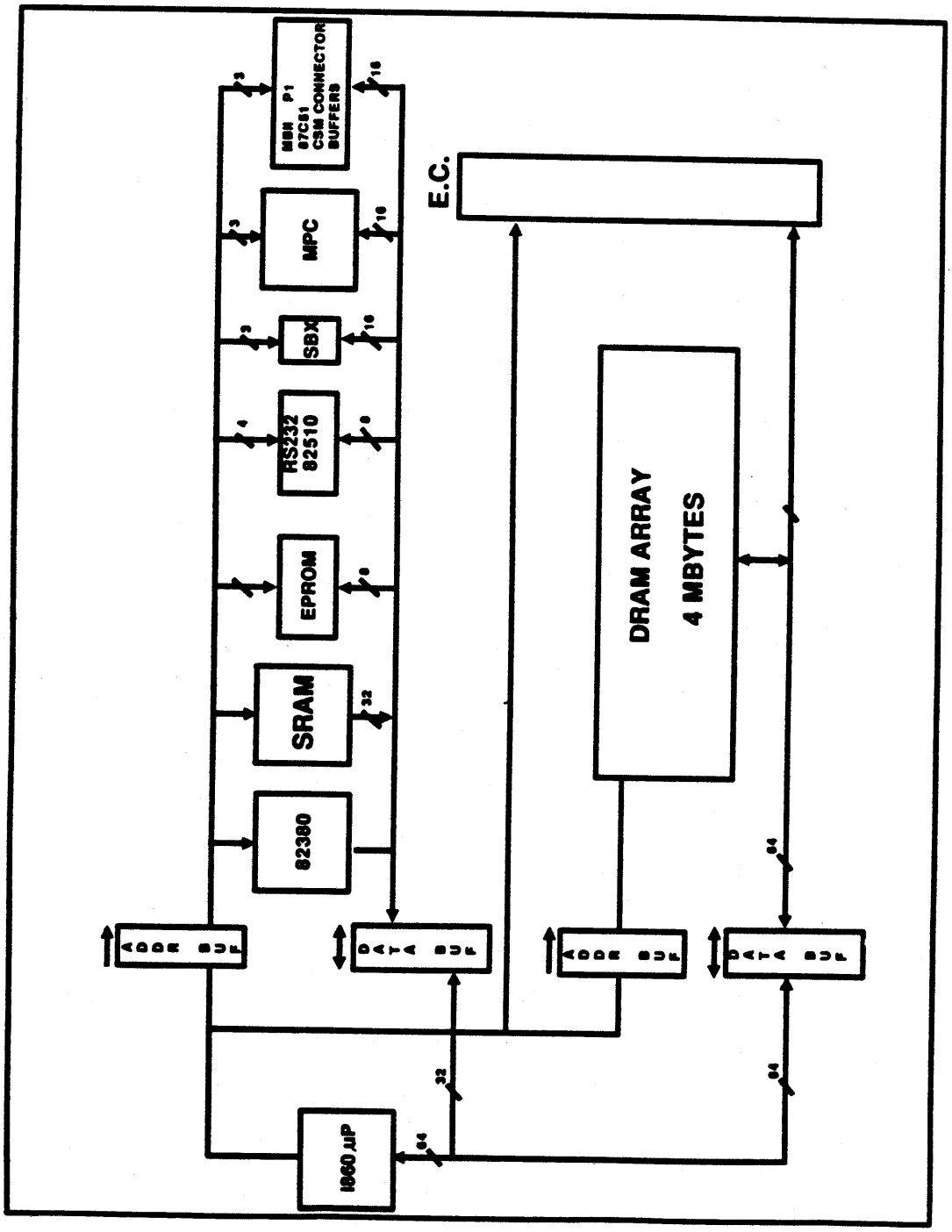


Figure 7.1 System Block Diagram of i860 Microprocessor Based MULTIBUS II Board

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. Microprocessor typically performs 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 dedicated hardware for this task, thus enhancing overall system performance.

* An interconnect space allows geographic addressing, the identification of any bus agent (board) by slot number. Every MULTIBUS II system 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.

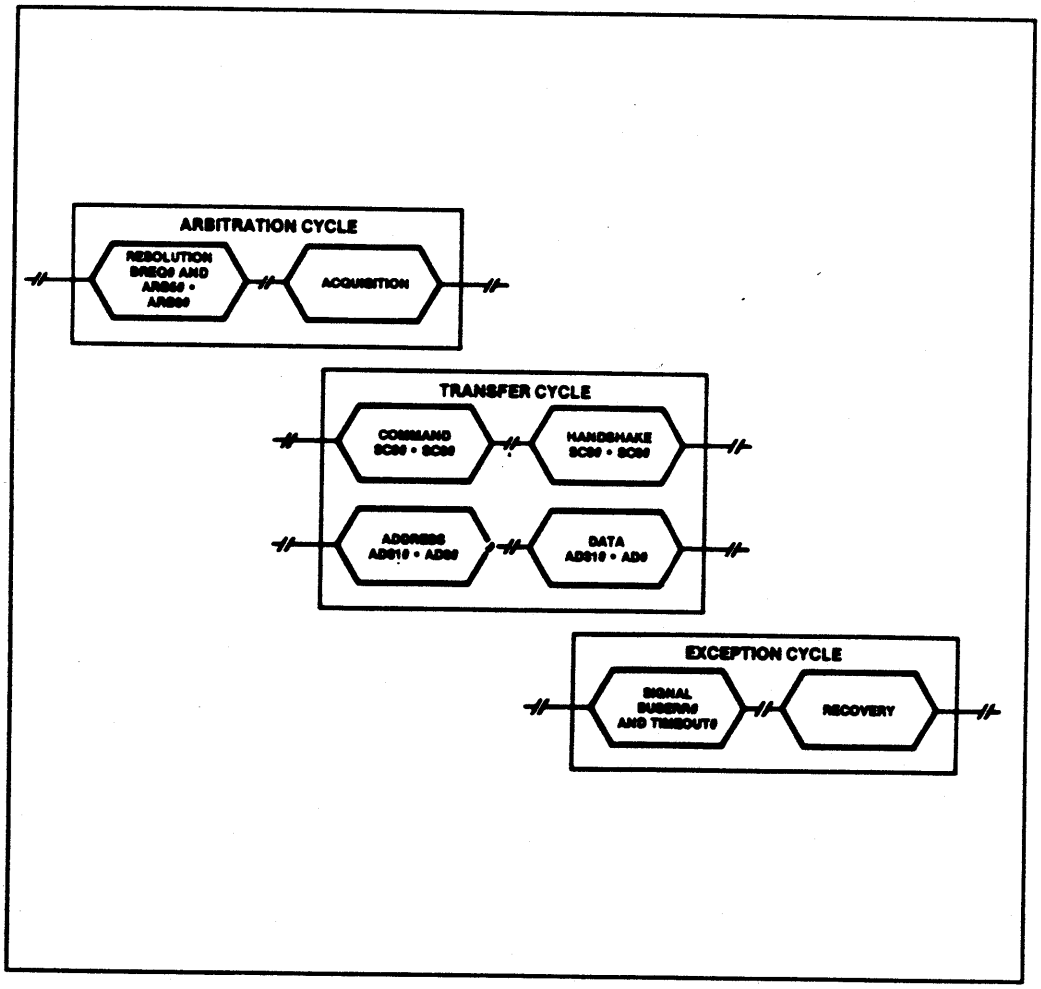


Figure 7.2 PSB Bus Cycle Timing

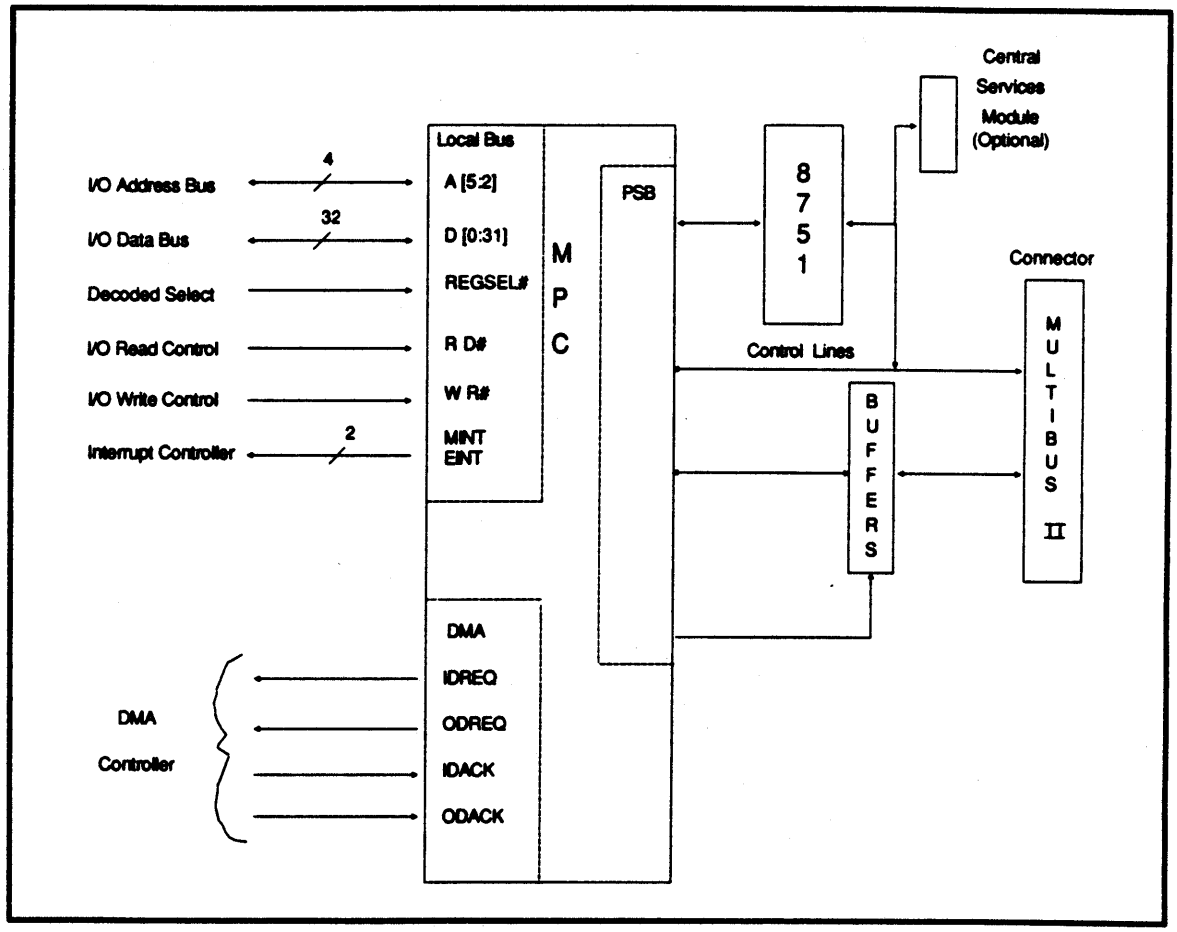


Figure 7.3 MPC Signal Groups

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 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.

* One 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 bi-directional 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 bi-directional signal. (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 bi-directional 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 SCOEL) are provided to let transceivers buffer these bi-directional 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 (LACHn#, 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 (PSB: parallel system bus).

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 signal 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 main memory.

7.3 i860 MICROPROCESSOR BUS INTERFACE

The i860 microprocessor has a synchronous interface with a non-multiplexed address and data bus. 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 bi-directional 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, non-interleaved 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 mode 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 x 4 chips in ZIP sockets. Parity DRAMS are 1M x 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 SBX 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 (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 provide 15 interrupts.

Four of the DMA channels are used. Two are for the MPC interface, and two are for the SBX 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 SBX CONNECTOR

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

DMA transfers with the SBX 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 messages and for I/O devices on the SBX 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, 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 programed 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. 82380 bus control logic 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 both main memory and SRAM message area accesses. Address line A2 of the 82380 determines which half of 64-bit memory data is accessed.

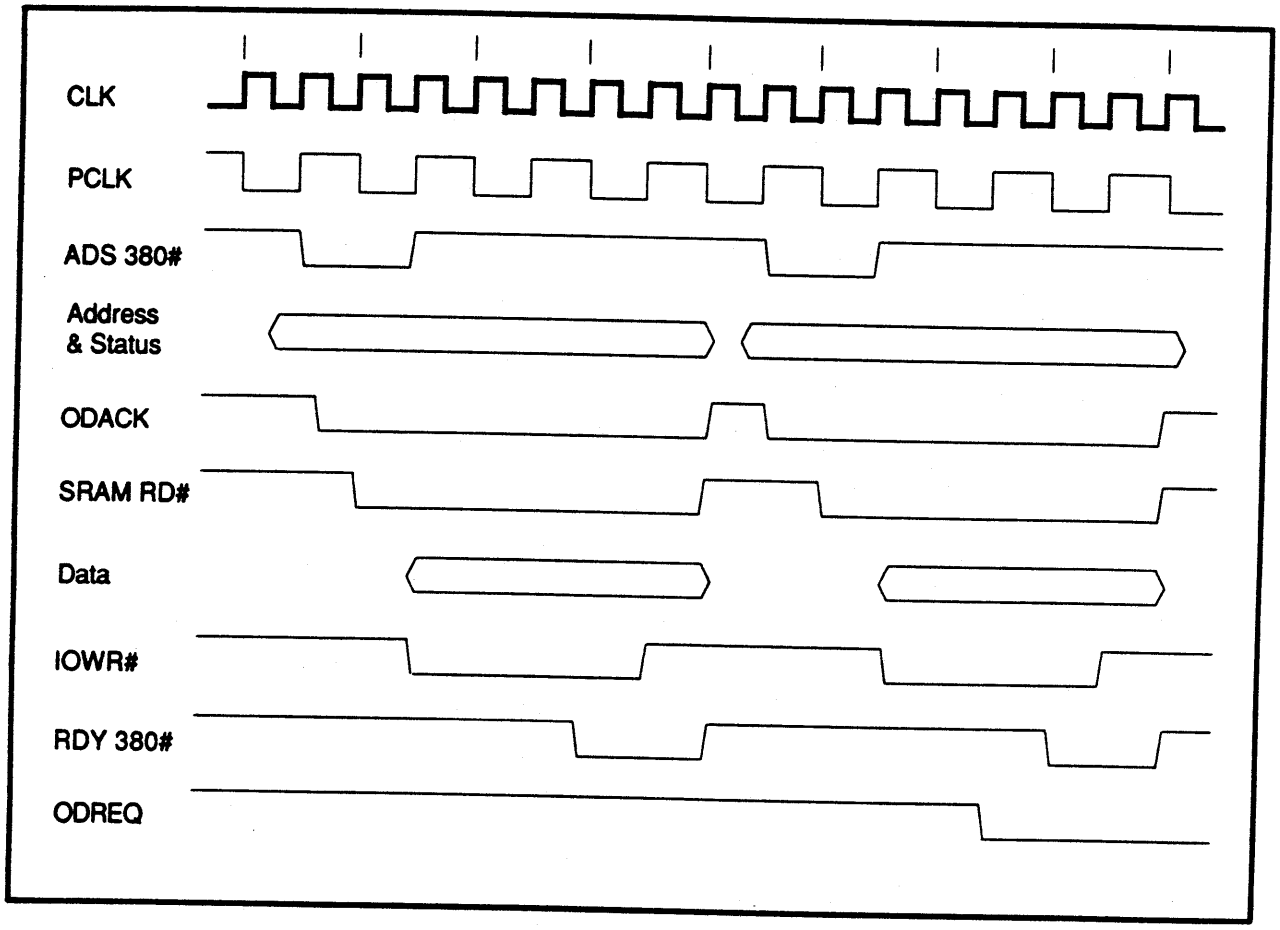


Figure 7.4 Fly-by Transfer with SRAM Read and I/O Write

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 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 even-word addresses. The auto-increment addressing mode of the processor is used, and no full modula two addressing efficiency 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. The previous example employs only half the number of SRAMs.

Arbitration between the two bus masters is performed by a PAL. The PAL 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 PAL forces the 82380 from the bus.

Figure 7-7 shows arbitration and DMA control logic blocks. The 82380 asserts HOLD and the arbitration PAL 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 duration of this process includes 82380 cycle time and data bus recovery time of the device being accessed. The arbitration PAL 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.

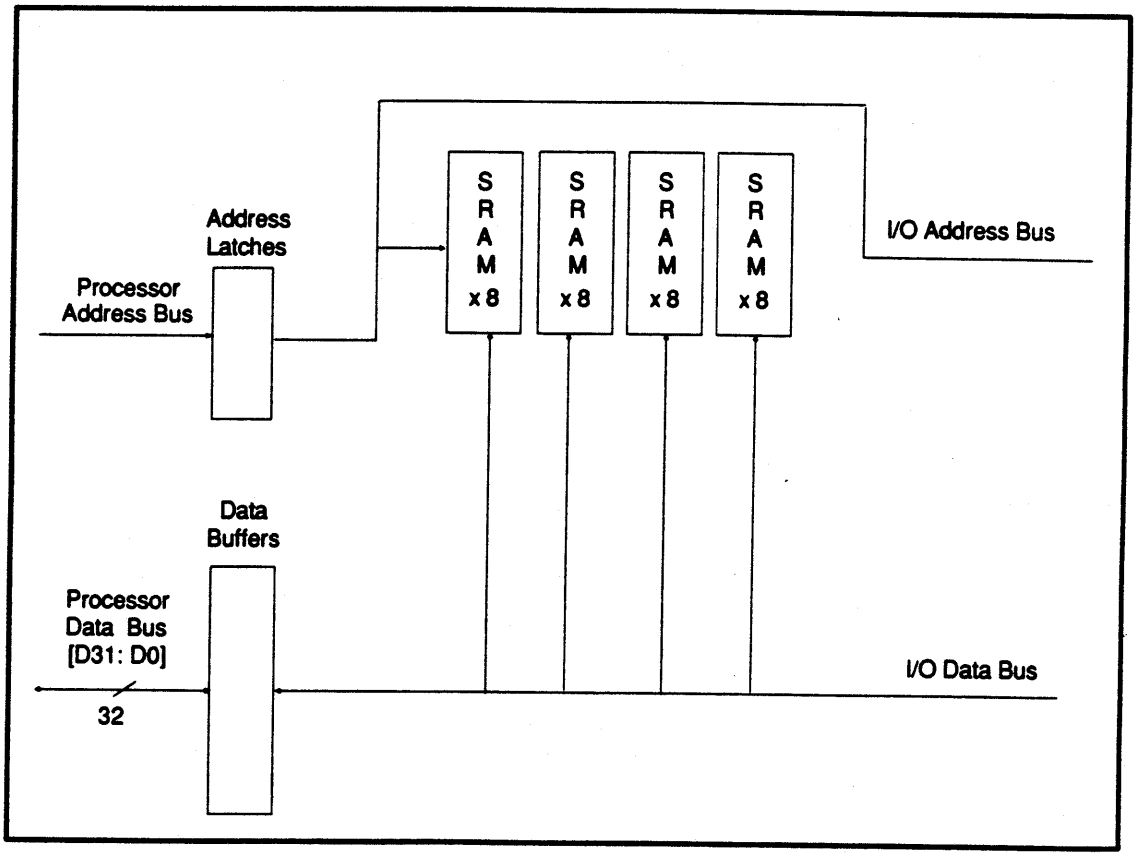


Figure 7.5 SRAM Message Area using 32-bit Bus

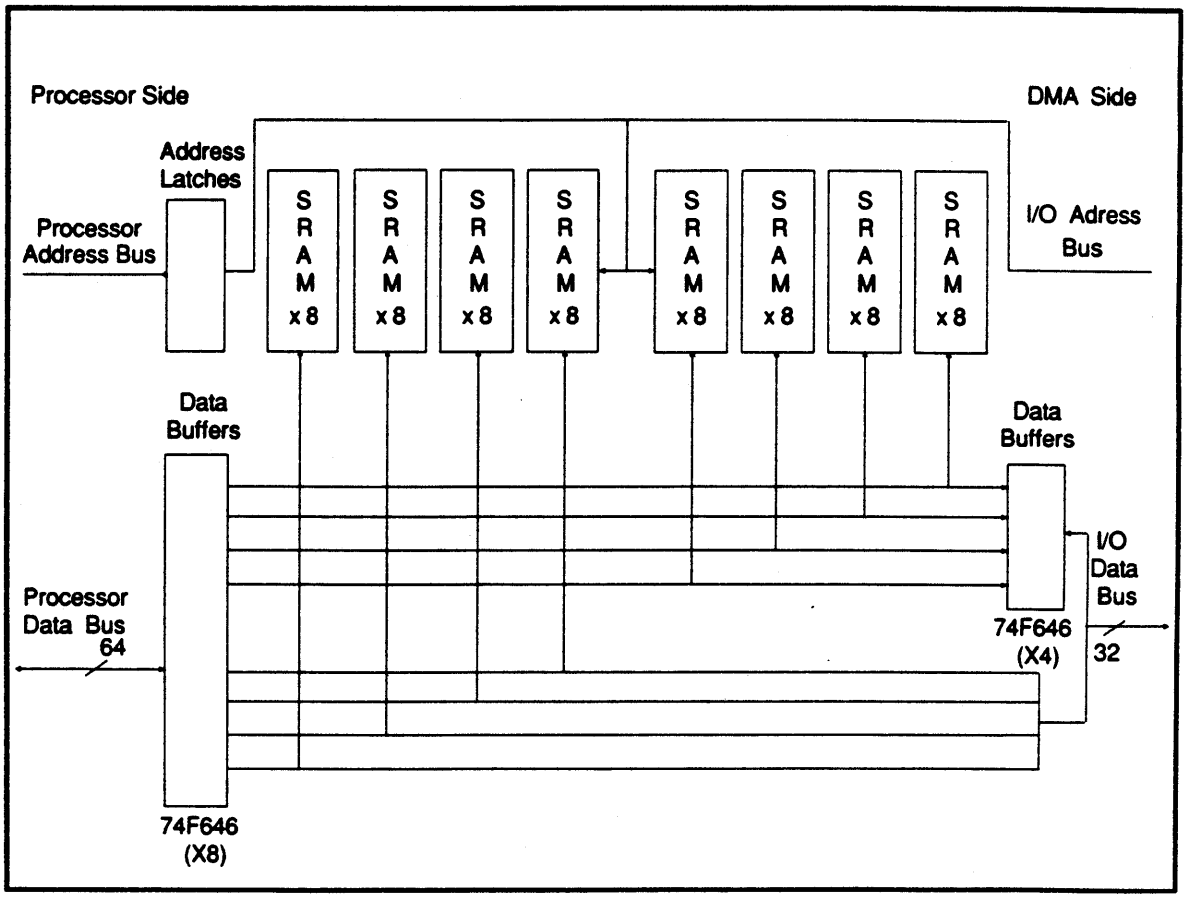


Figure 7.6 SRAM Message Area using 64-bit Bus

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.

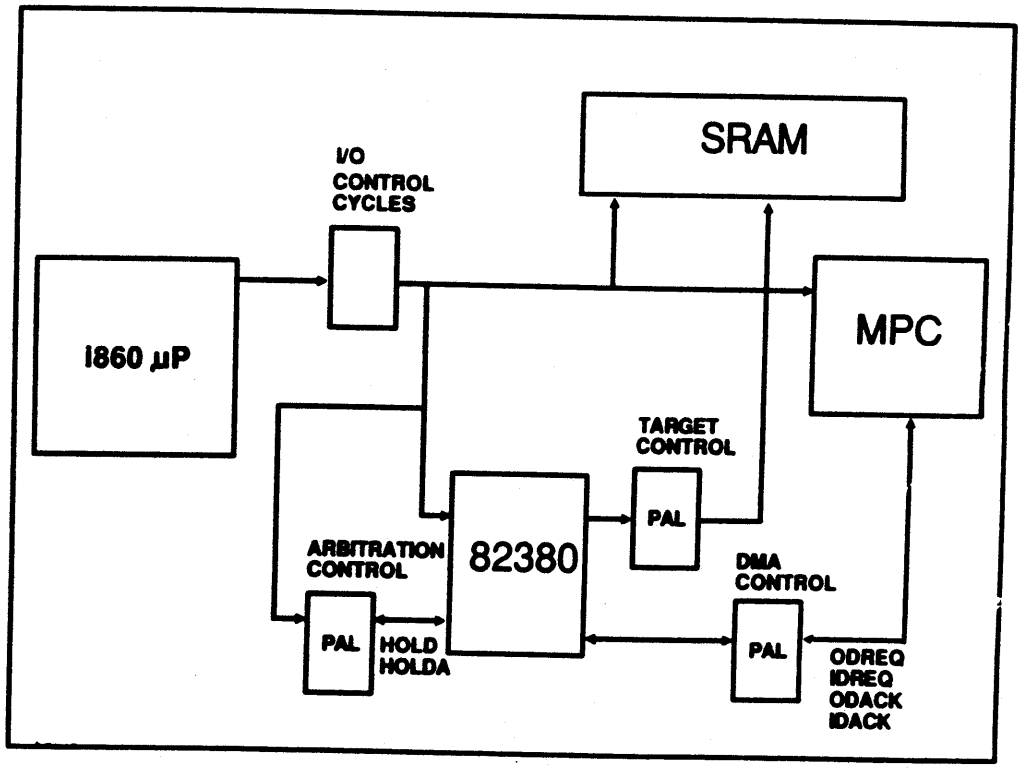


Figure 7.7 DMA System Arbitration and Control

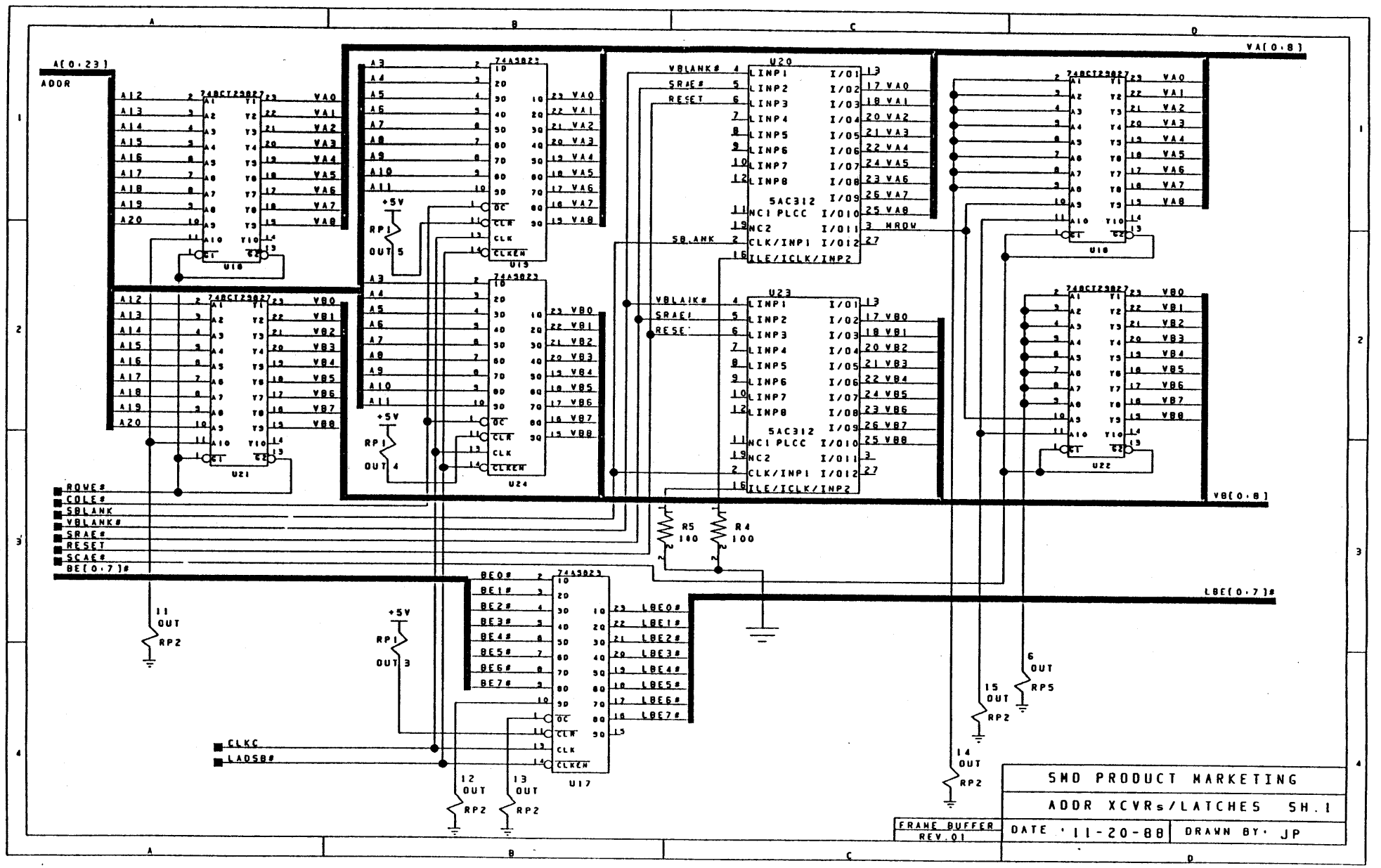
7.7.2 INTELLIGENT EXPANSION

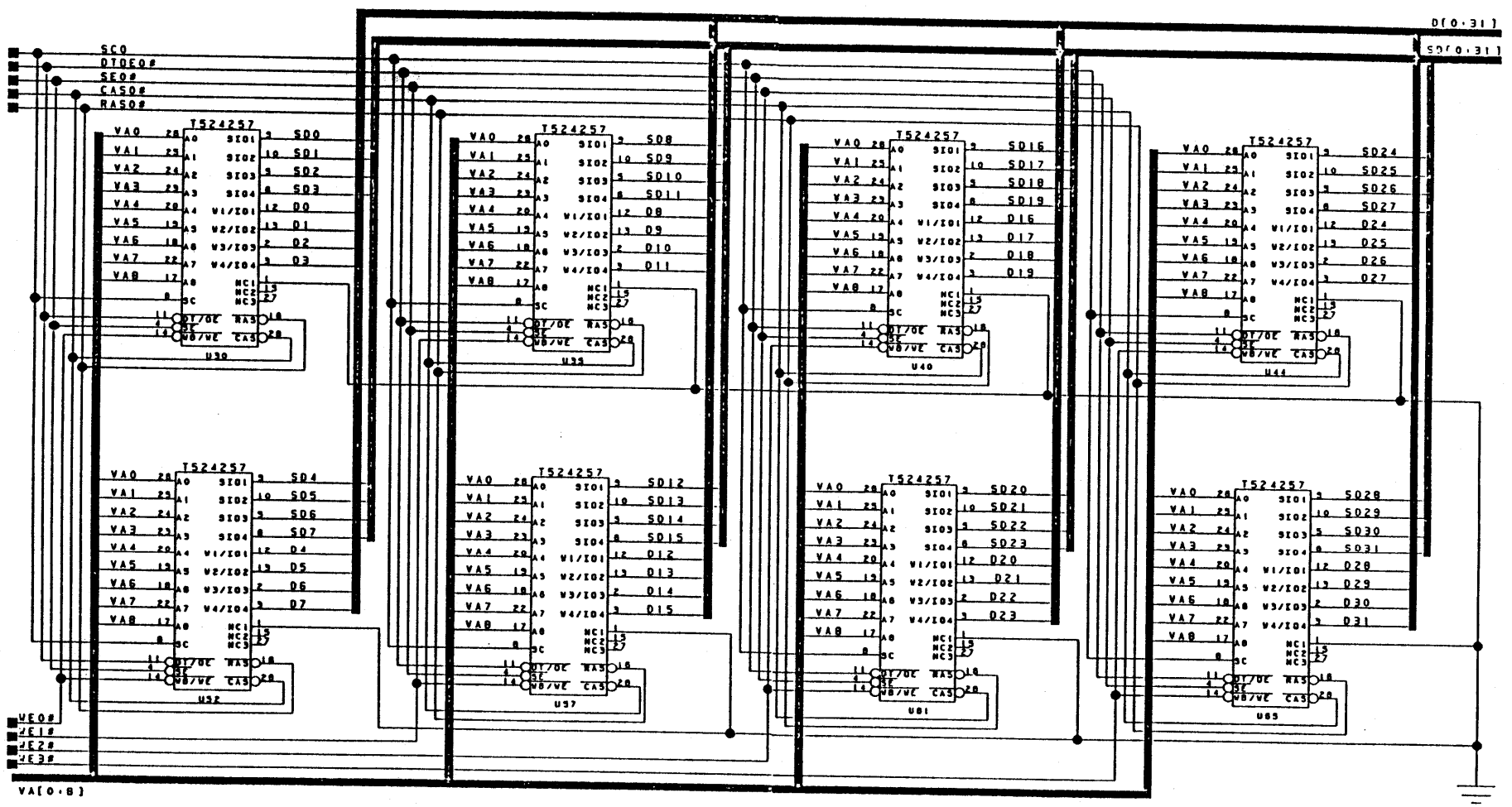
The connector provides the control signals needed to build an intelligent expansion board. One example of an intelligent board is a frame buffer board. It requires its own VRAM controller and control logic for returning READY# and NA# to the processor. Motherboard data buffer control signals are also 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.

*Appendix Graphics Frame Buffer *Schematics**

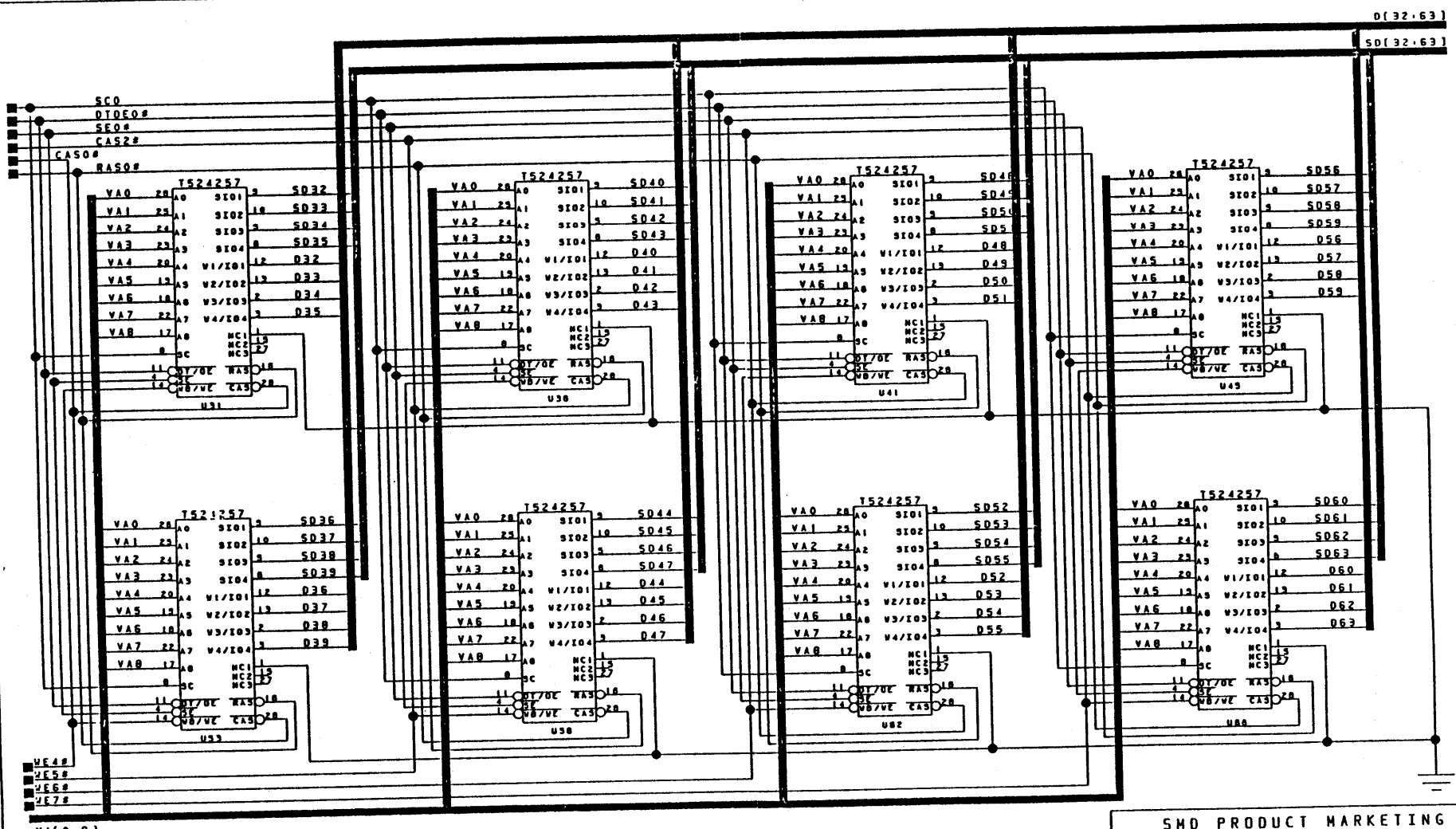
A

* This design has not been tested. The tested design will be published at a later date.

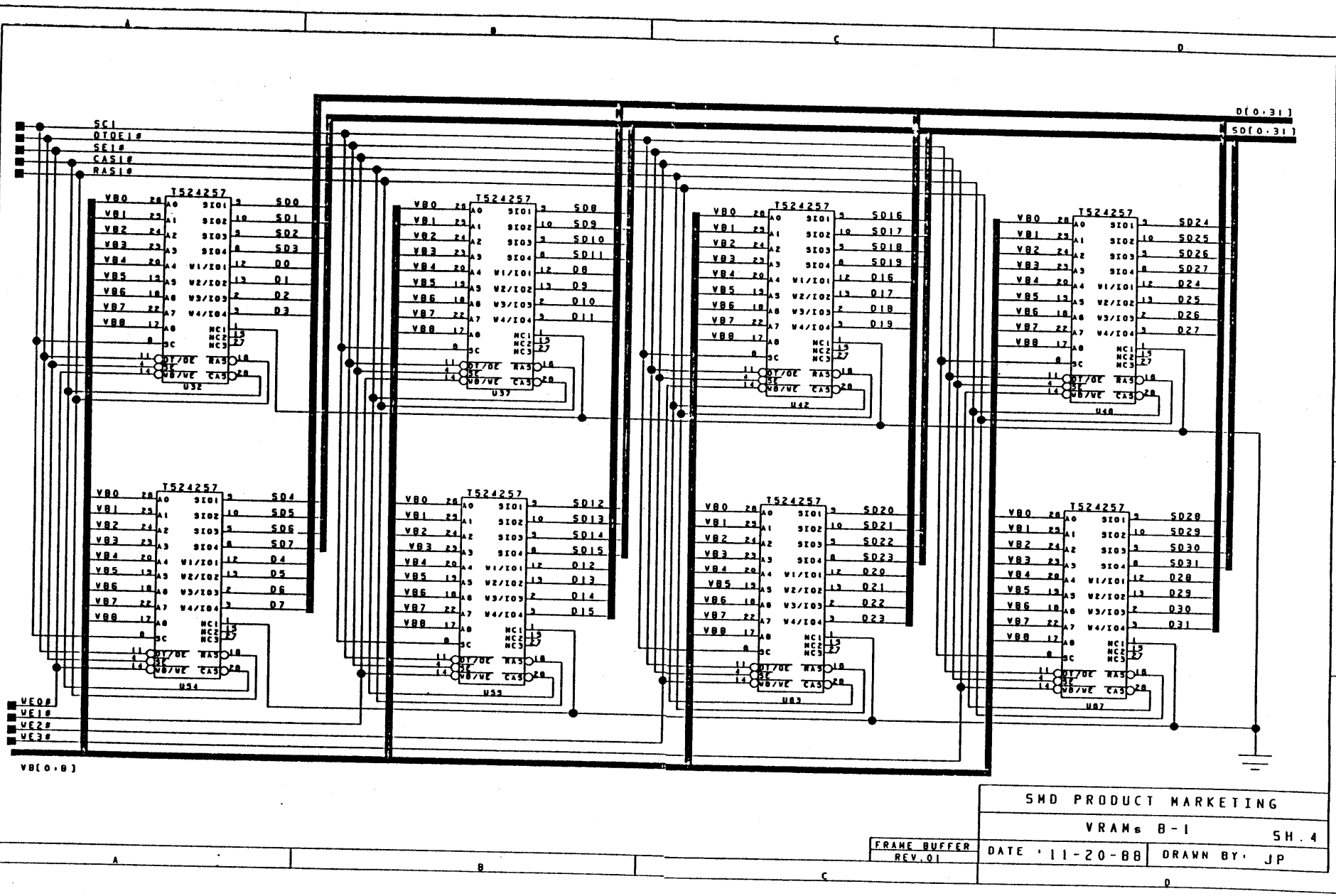

A[0:23]
ADDR
VA[0:8]
VB[0:8]
LBE[0:7]#
U18
U19
U21
U24
U20
U23
U22
U17
74ABCT29827
74A3823
54C312
PLCC
ROVE#
COLE#
SBLANK
VBLANK#
SRAE#
RESET
SCAE#
BE[0:7]#
CLKC
LADSB#
SMD PRODUCT MARKETING
ADDR XCVRs/LATCHES SH.1
FRAME BUFFER
REV.01
DATE 11-20-88
DRAWN BY: JP



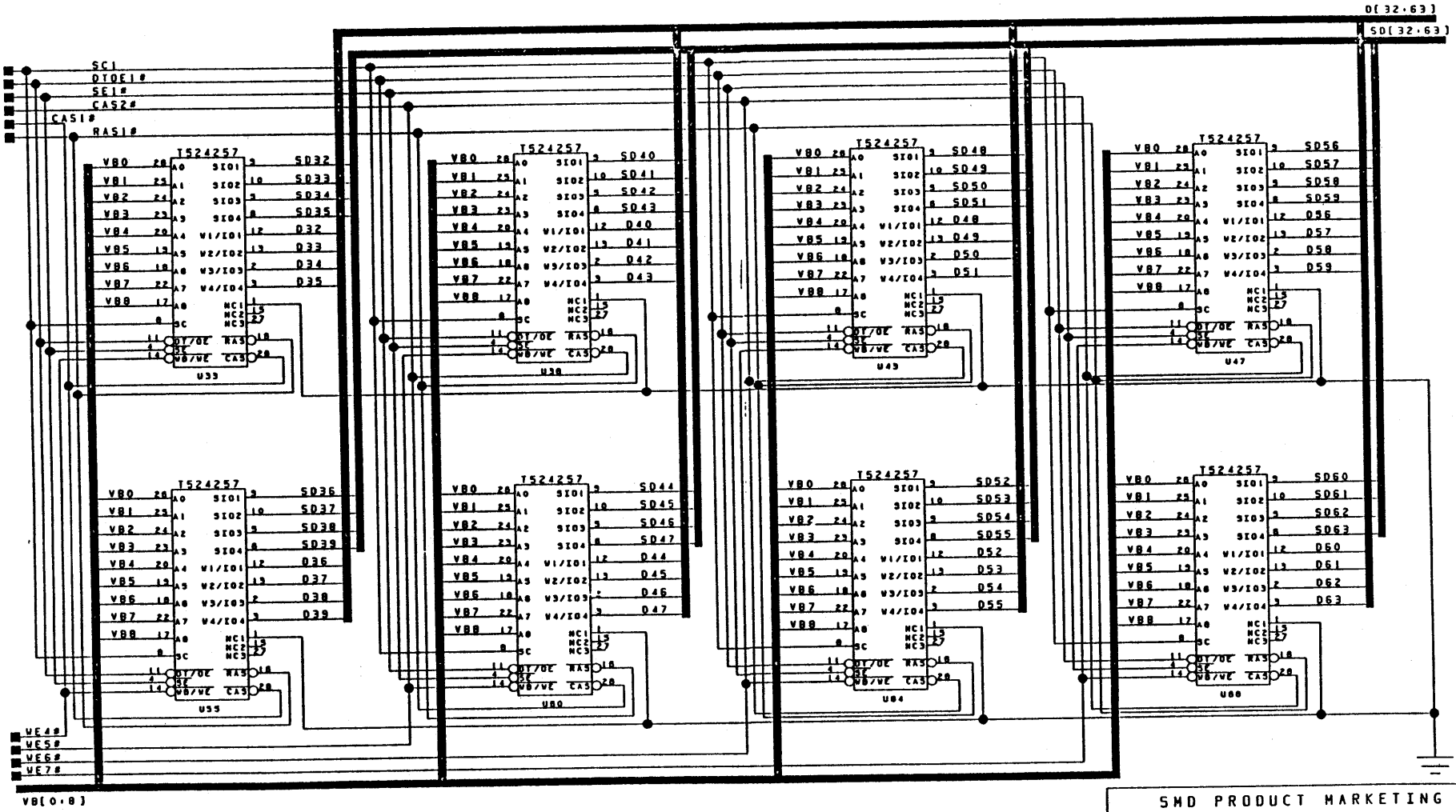
SMD PRODUCT MARKETING	
VRAMs A-1	SH. 2
DATE '11-20-88	DRAWN BY: JP
FRAME BUFFER REV.01	



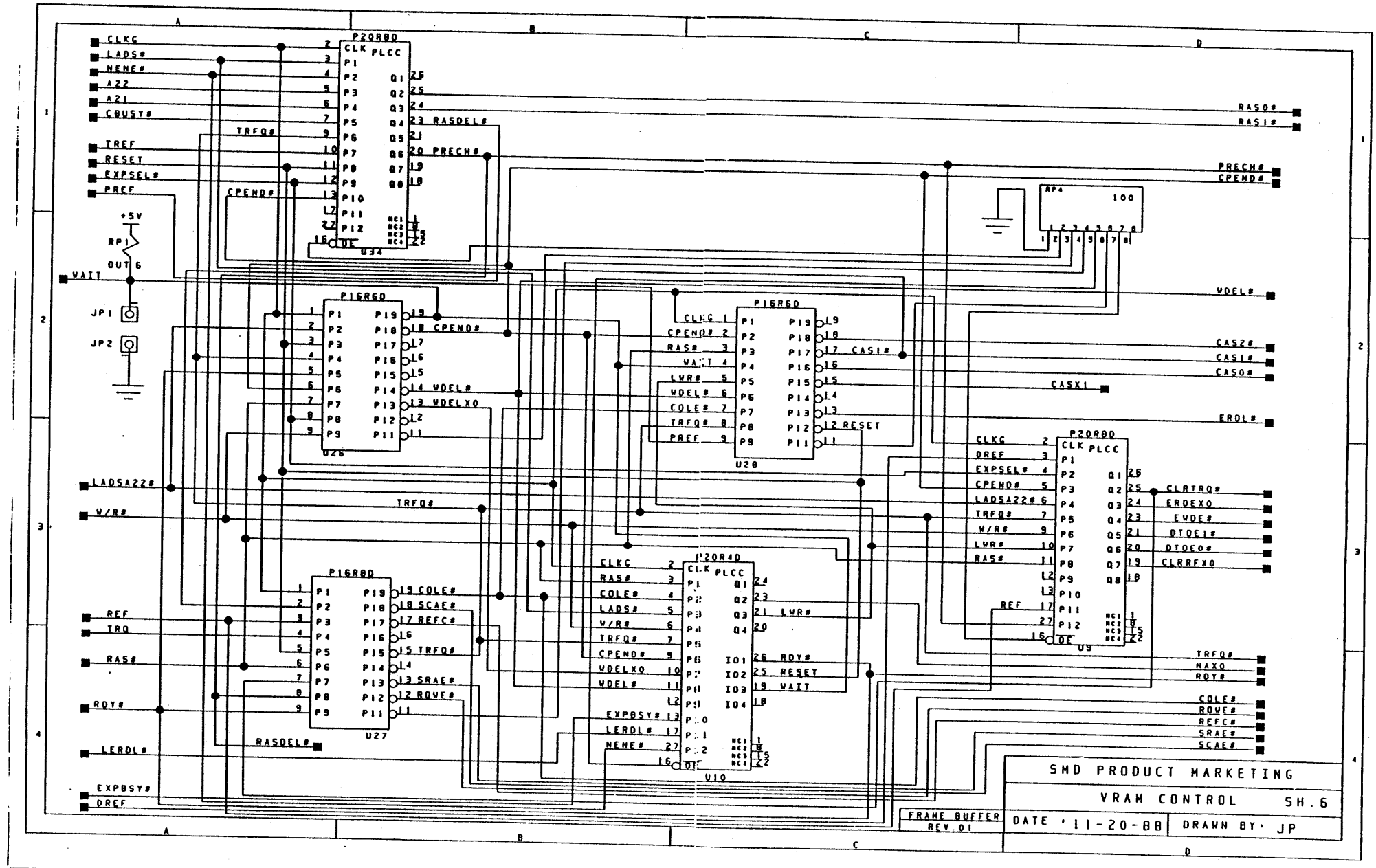
SMD PRODUCT MARKETING		
VRAMs A-2		SH. 3
FRAME BUFFER REV.01	DATE · 11-20-88	DRAWN BY· JP

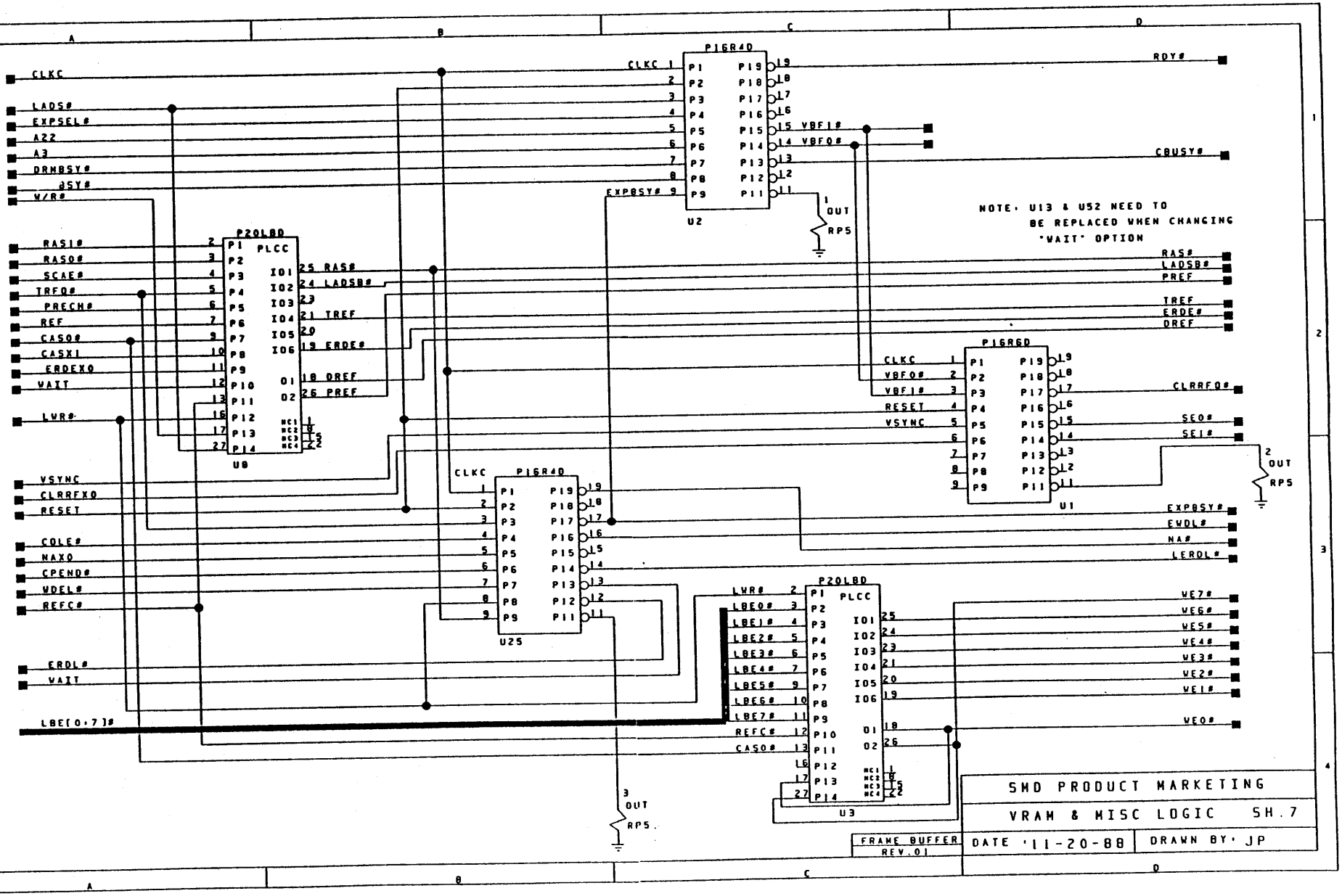


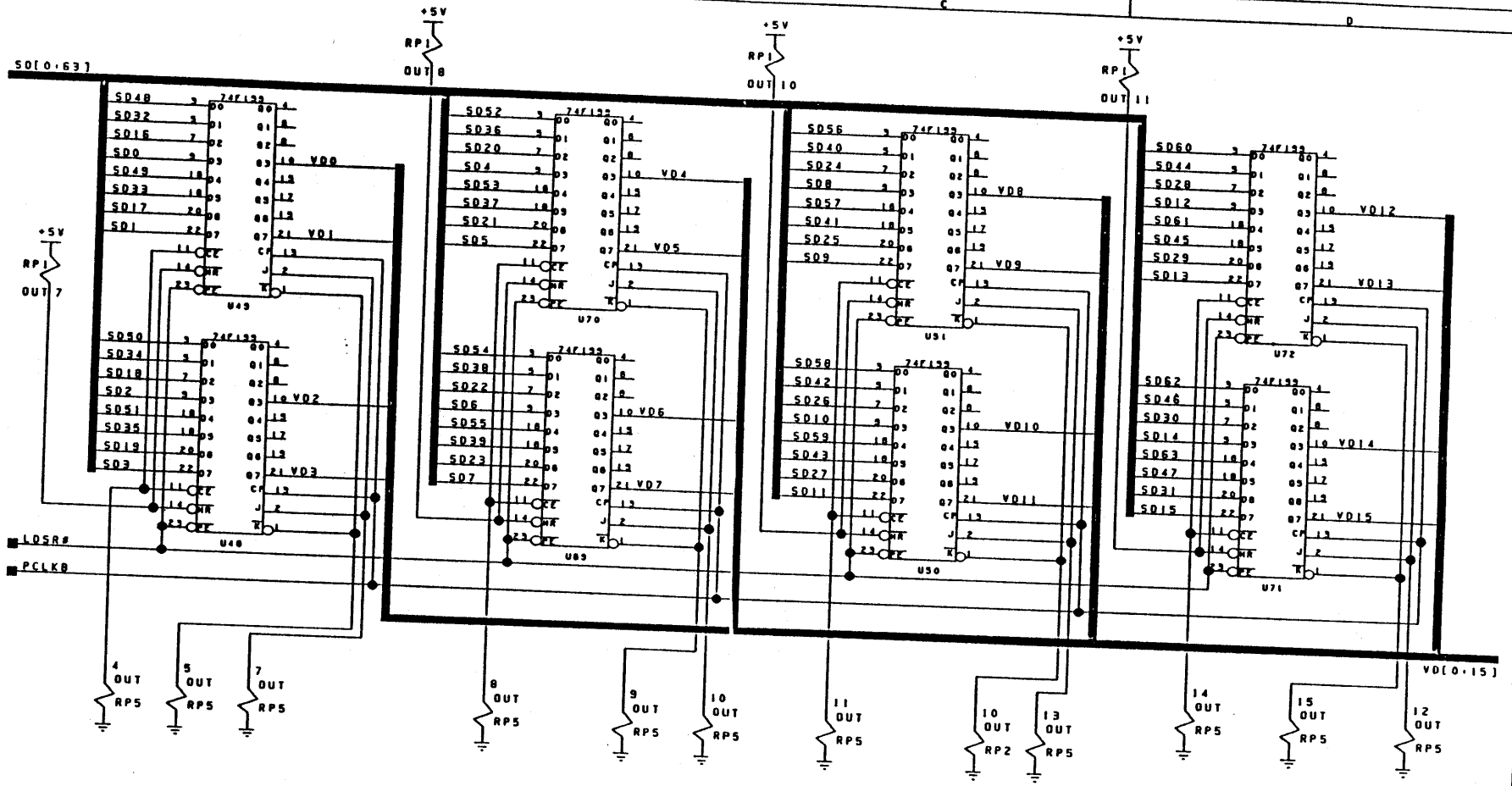
SMD PRODUCT MARKETING		
VRAMs 8-1		SH. 4
FRAME BUFFER REV.01	DATE : 11-20-88	DRAWN BY: JP



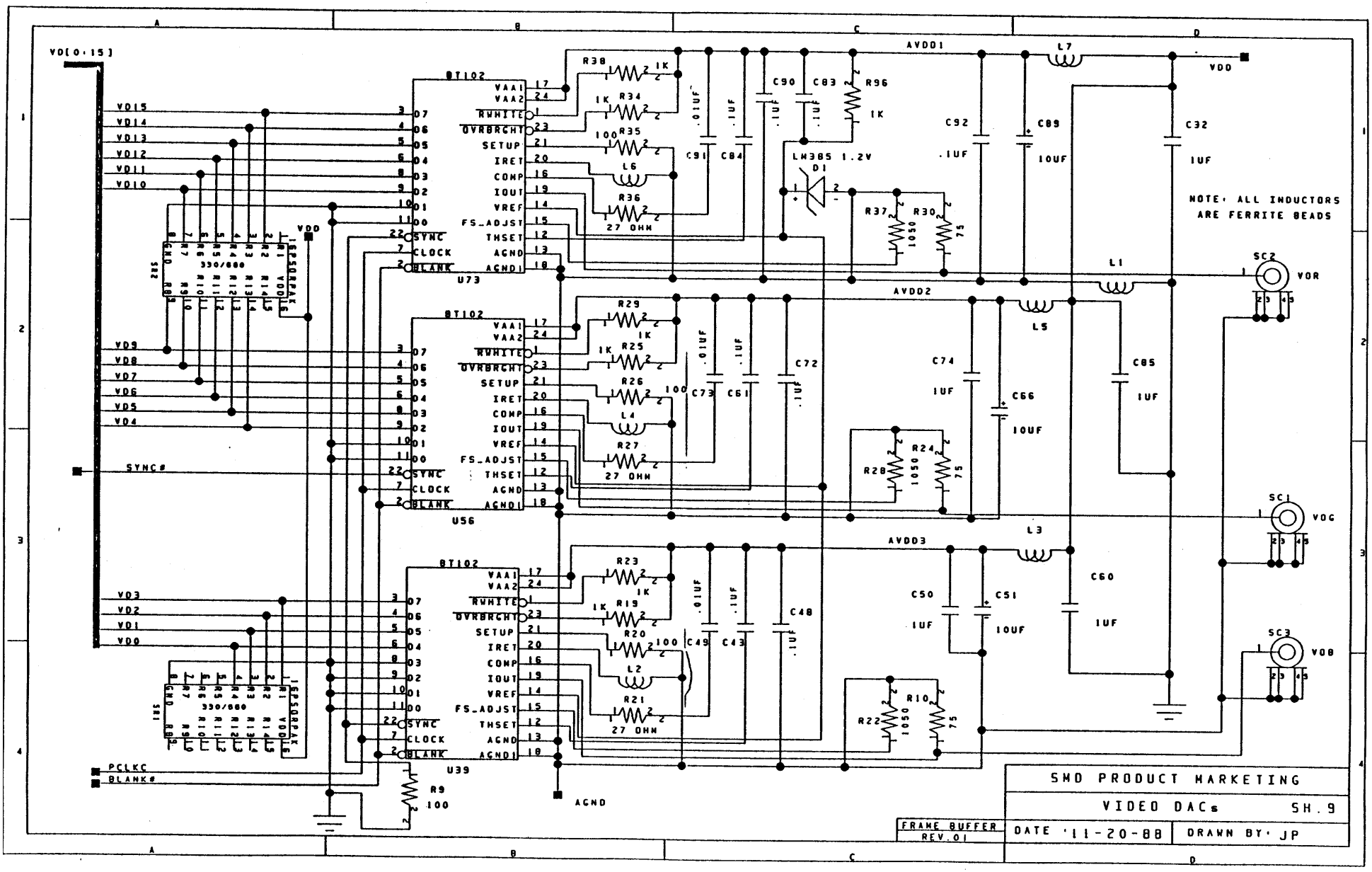
SMD PRODUCT MARKETING	
VRAMs 8-2	SH. 5
DATE : 11-20-88	DRAWN BY: JP
FRAME BUFFER REV. 01	

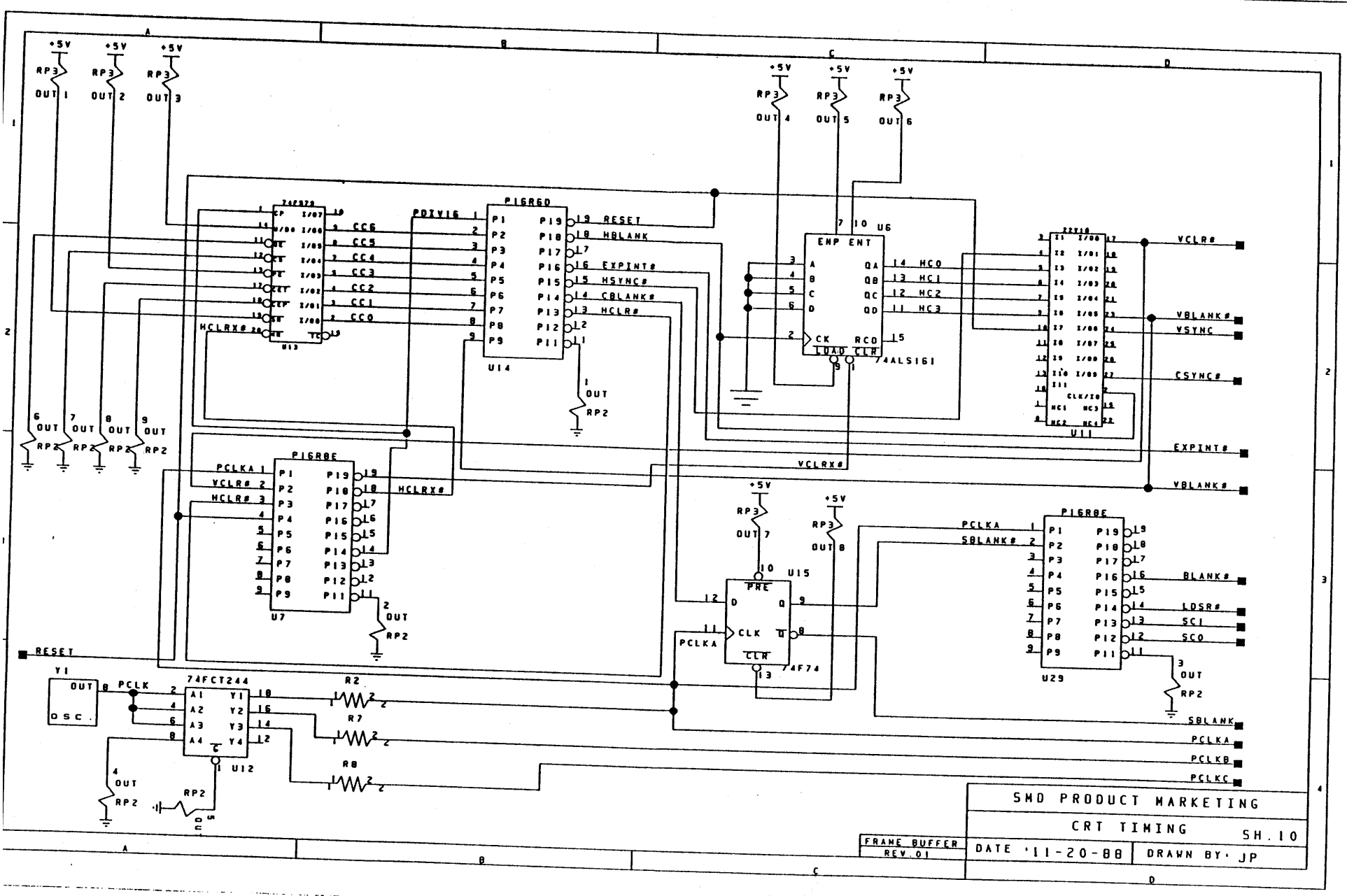

CLKG
LADS#
NENE#
A22
A21
CBUSY#
TREF
RESET
EXPSEL#
PREF
+5V
RP1
OUT 6
WAIT
JP1
JP2
P20R8D
CLK PLCC
TRFQ#
CPEND#
RASDEL#
PRECH#
U34
P16R6D
U26
VDEL#
VDELX0
U28
CAS1#
RESET
RP4
100
RAS0#
RAS1#
CAS2#
CAS1#
CAS0#
CASX1
ERDL#
P16R8D
U27
COLE#
SCAE#
REFC#
TRFQ#
SRAE#
ROVE#
REF
TRQ
RAS#
RDY#
LERDL#
EXPBSY#
DREF
LADSA22#
V/R#
P20R4D
U10
LWR#
RDY#
WAIT
DREF
EXPSEL#
LADSA22#
CLRTRQ#
ERDEX0
EVDE#
DTOE1#
DTOE0#
CLRRFX0
U9
NAX0
COLE#
ROVE#
REFC#
SRAE#
SCAE#
SMD PRODUCT MARKETING
VRAM CONTROL SH.6
FRAME BUFFER
REV.01
DATE : 11-20-88
DRAWN BY: JP

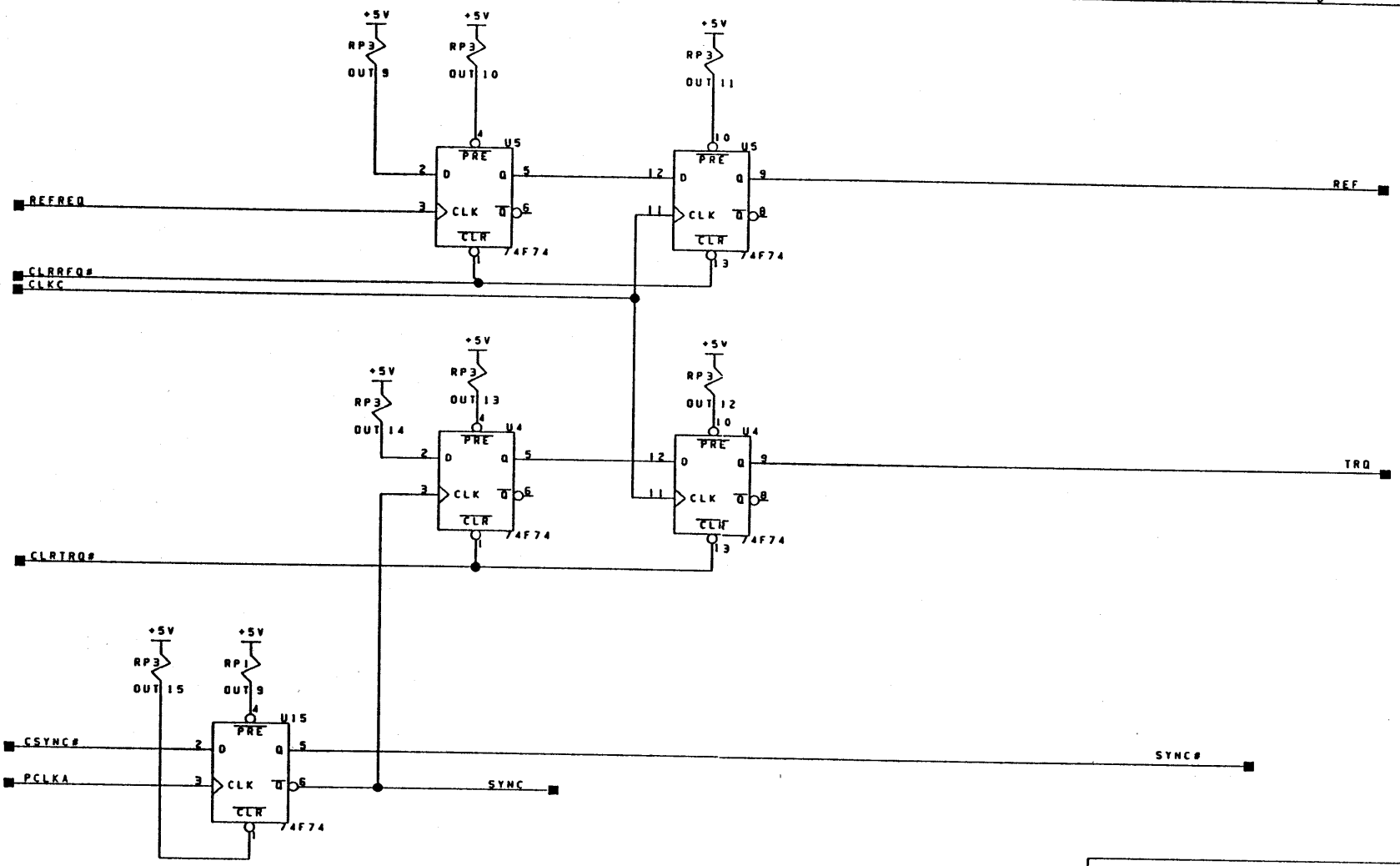

P16R4D
U2
P20LBD
PLCC
U0
P16R6D
U1
U25
U3
CLKC
LADS#
EXPSEL#
A22
A3
DRMBSY#
BSY#
W/R#
RAS1#
RAS0#
SCAE#
TRFQ#
PRECH#
REF
CAS0#
CASX1
ERDEX0
WAIT
LWR#
VSYNC
CLRRFX0
RESET
COLE#
NAX0
CPEND#
VDEL#
REFC#
ERDL#
WAIT
LBE[0:7]#
VBF1#
VBF0#
EXPBSY#
RDY#
CBUSY#
RAS#
LADSB#
PREF
TREF
ERDE#
DREF
CLRRF0#
SE0#
SE1#
EWDL#
NA#
LERDL#
LBE0#
LBE1#
LBE2#
LBE3#
LBE4#
LBE5#
LBE6#
LBE7#
WE7#
WE6#
WE5#
WE4#
WE3#
WE2#
WE1#
WE0#
RP5
NOTE: U13 & U52 NEED TO
BE REPLACED WHEN CHANGING
"WAIT" OPTION
SMD PRODUCT MARKETING
VRAM & MISC LOGIC SH.7
FRAME BUFFER
REV.01
DATE 11-20-88
DRAWN BY: JP



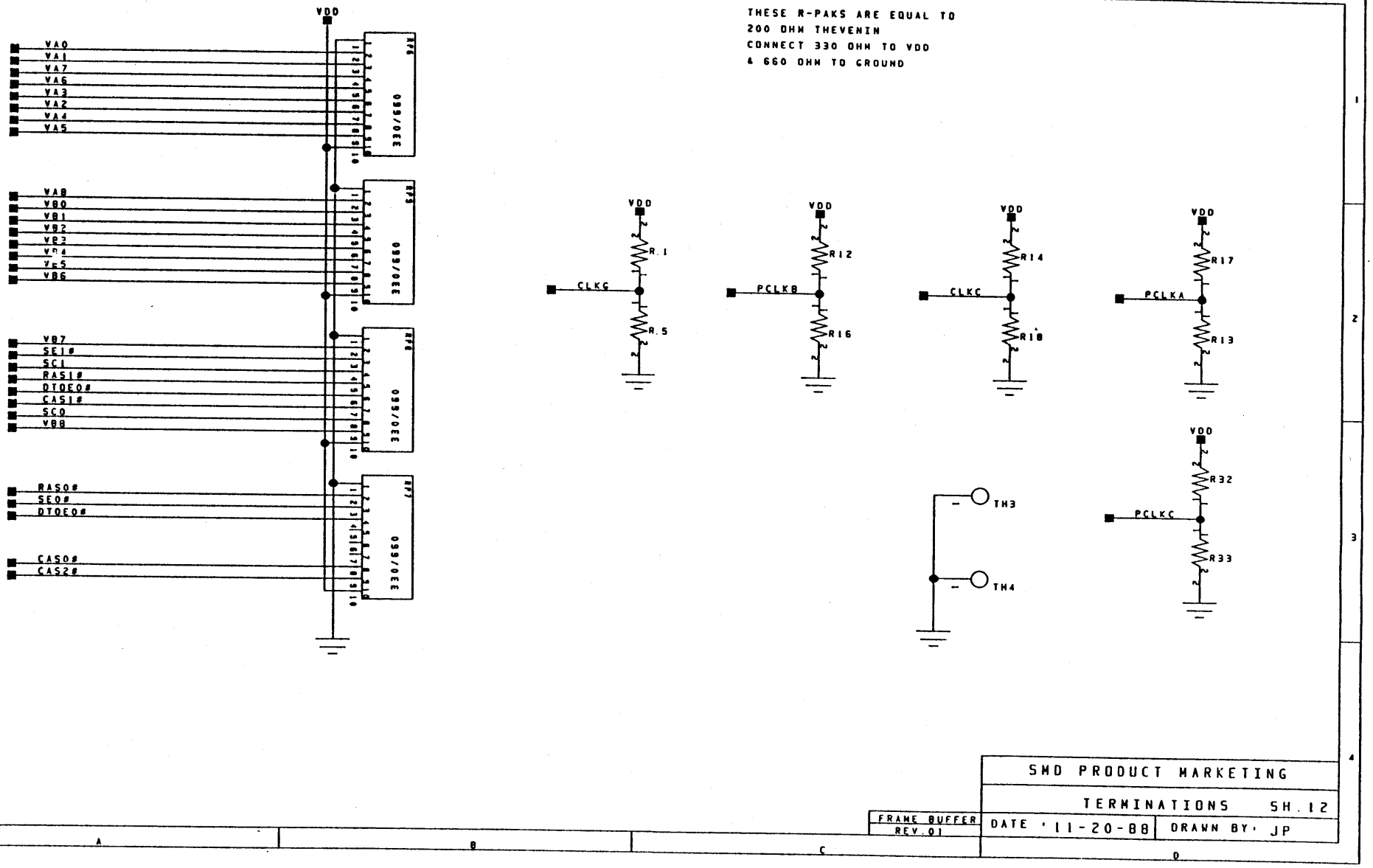
	SMD PRODUCT MARKETING	
	PIXEL SERIALIZATION SH.8	
FRAME BUFFER REV.01	DATE : 11-20-88	DRAWN BY: JP

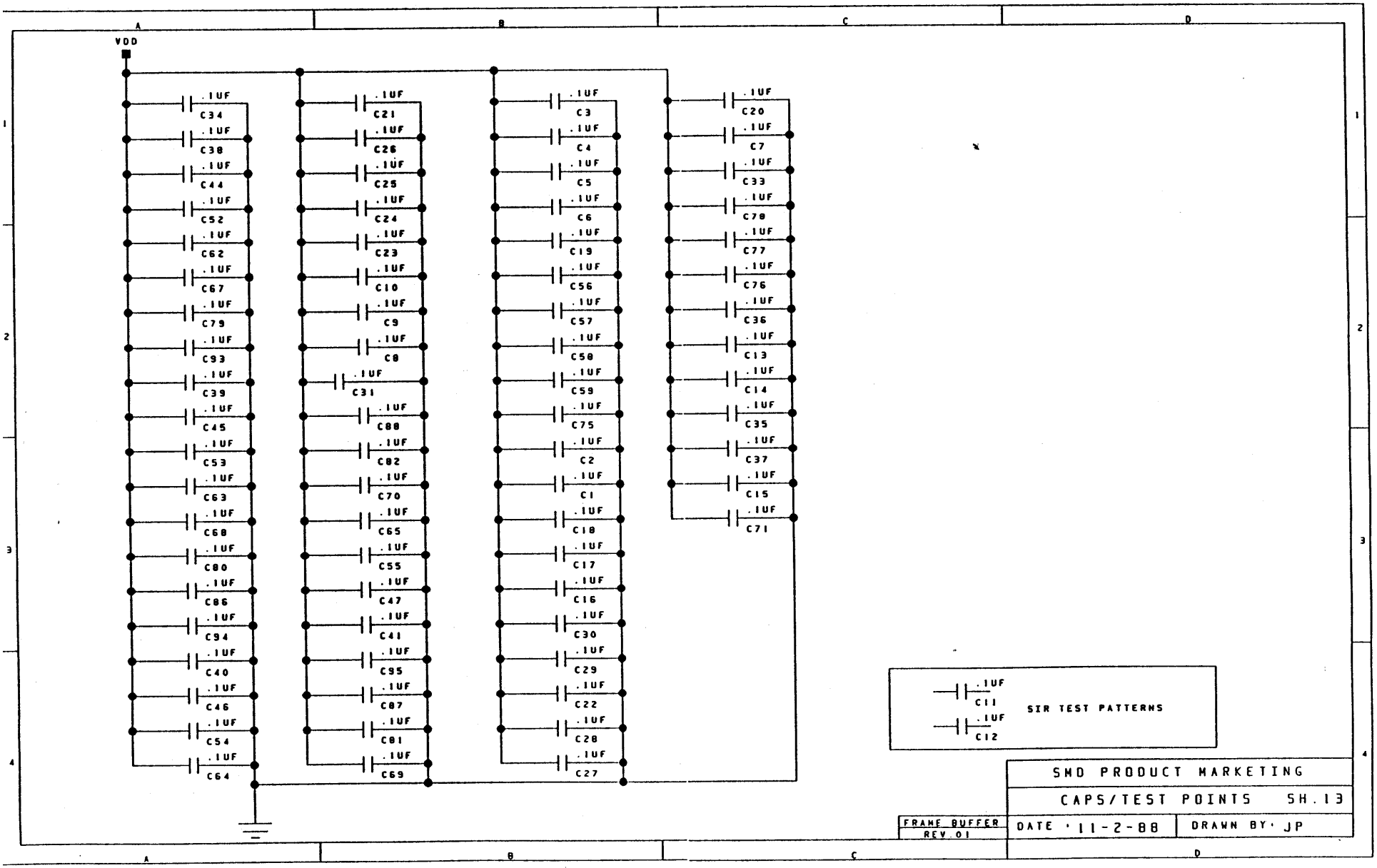

VD[0:15]
BT102
U73
U56
U39
VAA1
VAA2
RWHITE
OVRBRGHT
SETUP
IRET
COMP
IOUT
VREF
FS_ADJST
THSET
AGND
AGND1
SYNC
CLOCK
BLANK
AVDD1
AVDD2
AVDD3
VDD
AGND
R38 1K
R34 1K
R35 100
L6
R36 27 OHM
R29 1K
R25 1K
R26 100
L4
R27 27 OHM
R23 1K
R19 1K
R20 100
L2
R21 27 OHM
R9 100
C91 .01UF
C84 .1UF
C90 .1UF
C83 .1UF
R96 1K
LM385 1.2V
D1
R37 1050
R30 75
C92 .1UF
C89 10UF
C32 1UF
L7
L1
C73 .01UF
C61 .1UF
C72 .1UF
R28 1050
R24 75
C74 1UF
C66 10UF
C85 1UF
L5
C49 .01UF
C43 .1UF
C48 .1UF
R22 1050
R10 75
C50 1UF
C51 10UF
C60 1UF
L3
SC2
SC1
SC3
VDR
VDG
VDB
NOTE: ALL INDUCTORS ARE FERRITE BEADS
SYNC#
PCLKC
BLANK#
SIP8
330/680
ISPSOBPAK16
SMD PRODUCT MARKETING
VIDEO DACs
SH. 9
FRAME BUFFER
REV.01
DATE 11-20-88
DRAWN BY: JP

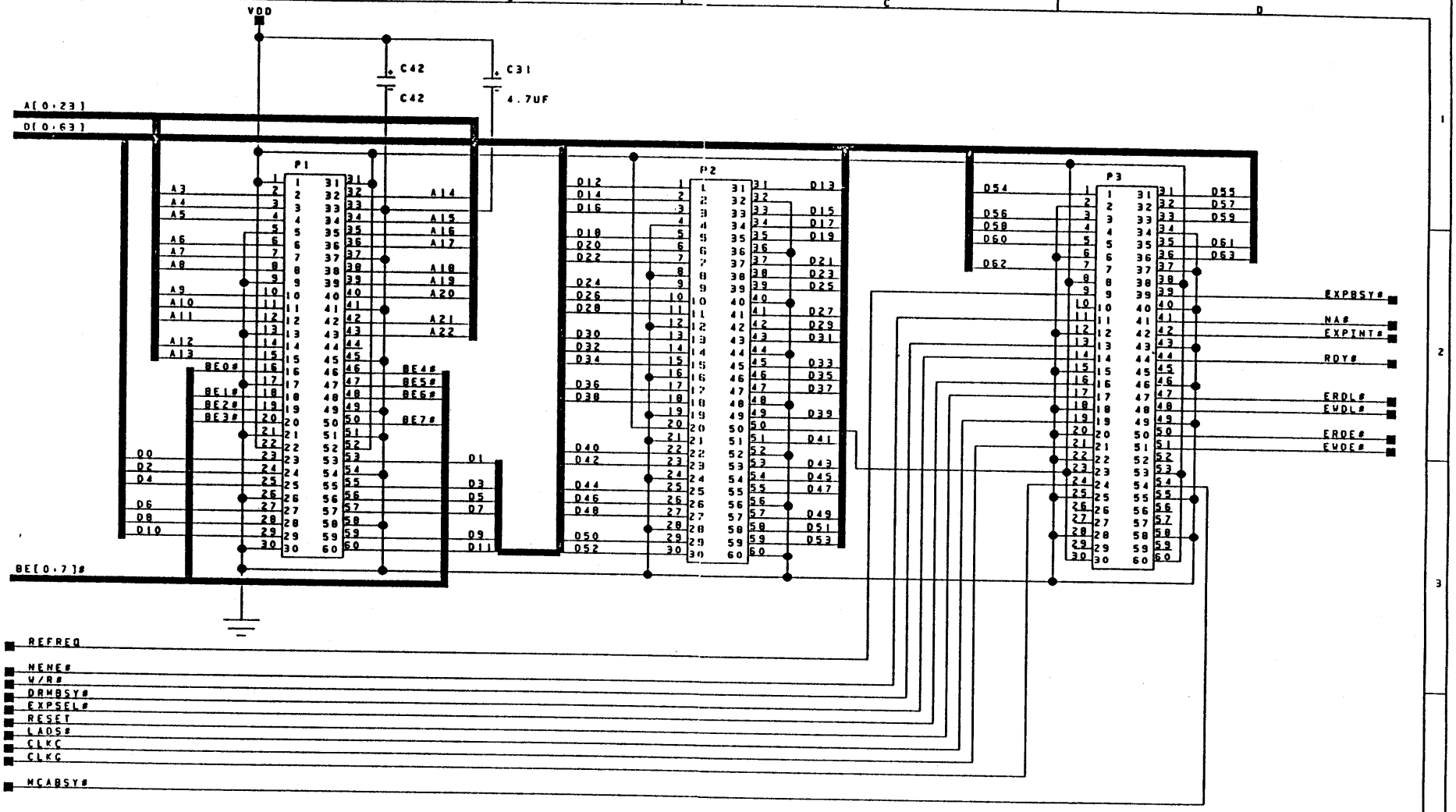

+5V
RP3
OUT 1
OUT 2
OUT 3
OUT 4
OUT 5
OUT 6
OUT 7
OUT 8
OUT 9
RP2
74F823
U13
CC6
CC5
CC4
CC3
CC2
CC1
CC0
HCLRX#
PDIV16
P16R6D
U14
RESET
HBLANK
EXPINT#
HSYNC#
CBLANK#
HCLR#
ENP ENT
74ALS161
U6
HC0
HC1
HC2
HC3
22V10
U11
VCLR#
VBLANK#
VSYNC
CSYNC#
EXPINT#
VCLRX#
P16R8E
U7
PCLKA
VCLR#
HCLR#
HCLRX#
U15
PRE
CLK
CLR
74F74
SBLANK#
BLANK#
LDSR#
SCI
SCO
U29
SBLANK
PCLKB
PCLKC
RESET
Y1
OSC.
PCLK
74FCT244
U12
R2
R7
R8
SMD PRODUCT MARKETING
CRT TIMING
SH. 10
FRAME BUFFER
REV. 01
DATE 11-20-88
DRAWN BY: JP



	SMD PRODUCT MARKETING	
	MISC	SH. 11
FRAME BUFFER REV. 01	DATE '11-20-88	DRAWN BY: JP


THESE R-PAKS ARE EQUAL TO
200 OHM THEVENIN
CONNECT 330 OHM TO VDD
& 660 OHM TO GROUND
VDD
VA0
VA1
VA7
VA6
VA3
VA2
VA4
VA5
VA8
VB0
VB1
VB2
VB3
VB4
VB5
VB6
VB7
SEI#
SCI
RASI#
DTOEO#
CASI#
SCO
VB8
RASO#
SEO#
DTOEO#
CASO#
CAS2#
330/660
CLKG
R.1
R.5
PCLKB
R12
R16
CLKC
R14
R18
PCLKA
R17
R13
TH3
TH4
PCLKC
R32
R33
SMD PRODUCT MARKETING
TERMINATIONS SH.12
FRAME BUFFER
REV.01
DATE : 11-20-88
DRAWN BY: JP

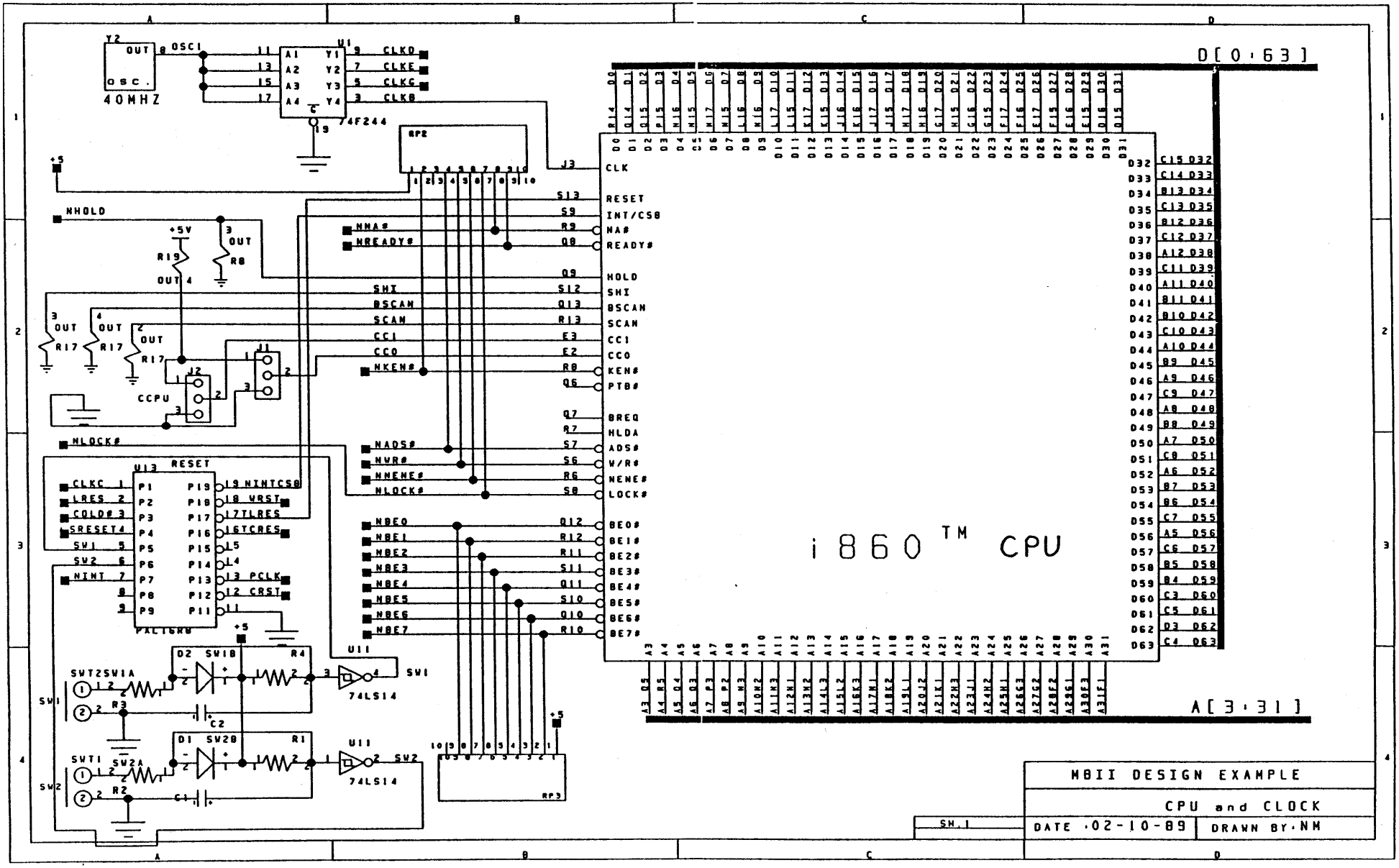

VDD
.1UF C34 .1UF C38 .1UF C44 .1UF C52 .1UF C62 .1UF C67 .1UF C79 .1UF C93 .1UF C39 .1UF C45 .1UF C53 .1UF C63 .1UF C68 .1UF C80 .1UF C86 .1UF C94 .1UF C40 .1UF C46 .1UF C54 .1UF C64
.1UF C21 .1UF C26 .1UF C25 .1UF C24 .1UF C23 .1UF C10 .1UF C9 .1UF C8 .1UF C31 .1UF C88 .1UF C82 .1UF C70 .1UF C65 .1UF C55 .1UF C47 .1UF C41 .1UF C95 .1UF C87 .1UF C81 .1UF C69
.1UF 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 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 C11 .1UF C12 SIR TEST PATTERNS
SMD PRODUCT MARKETING
CAPS/TEST POINTS SH.13
FRAME BUFFER REV.01
DATE: 11-2-88
DRAWN BY: JP

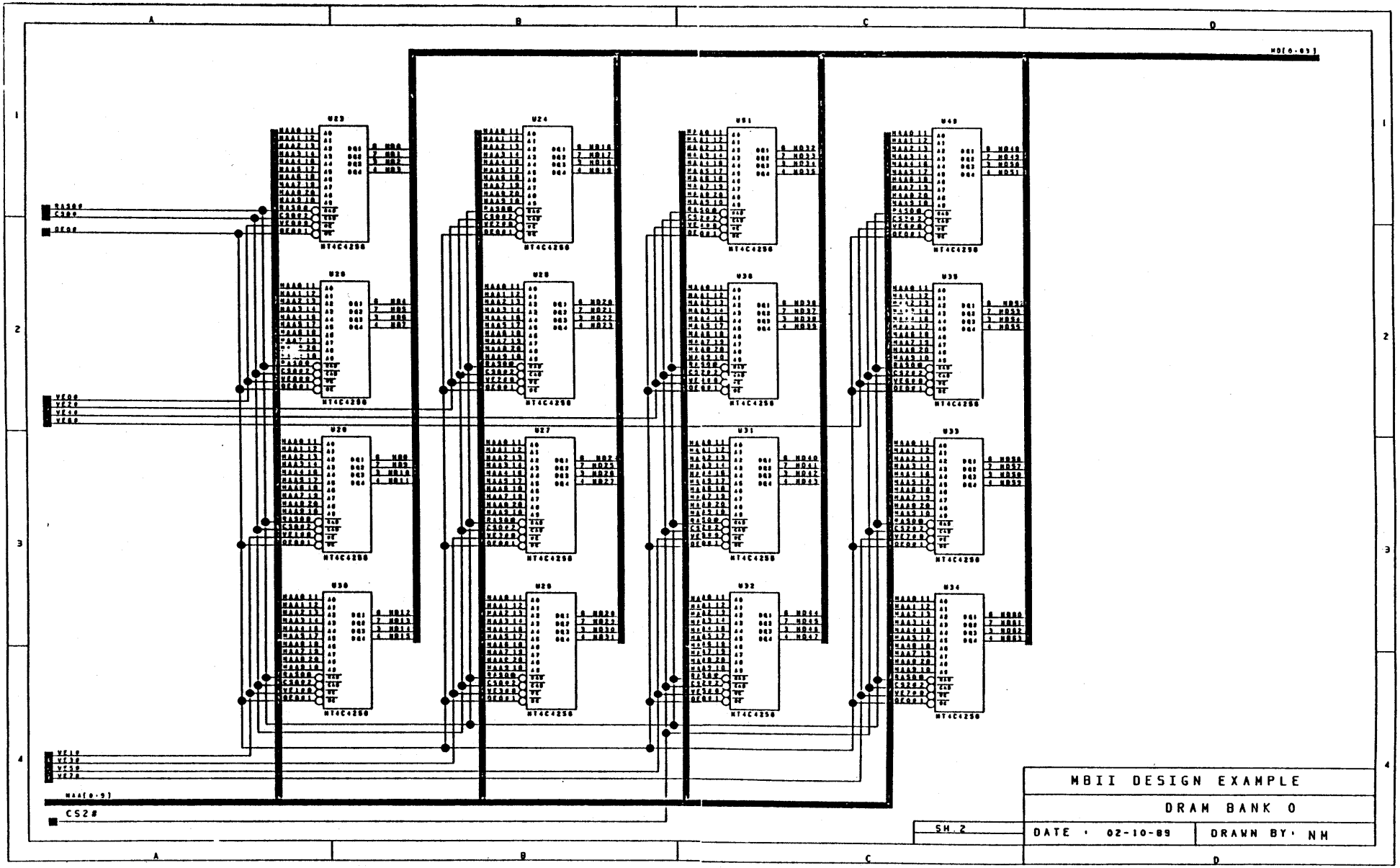


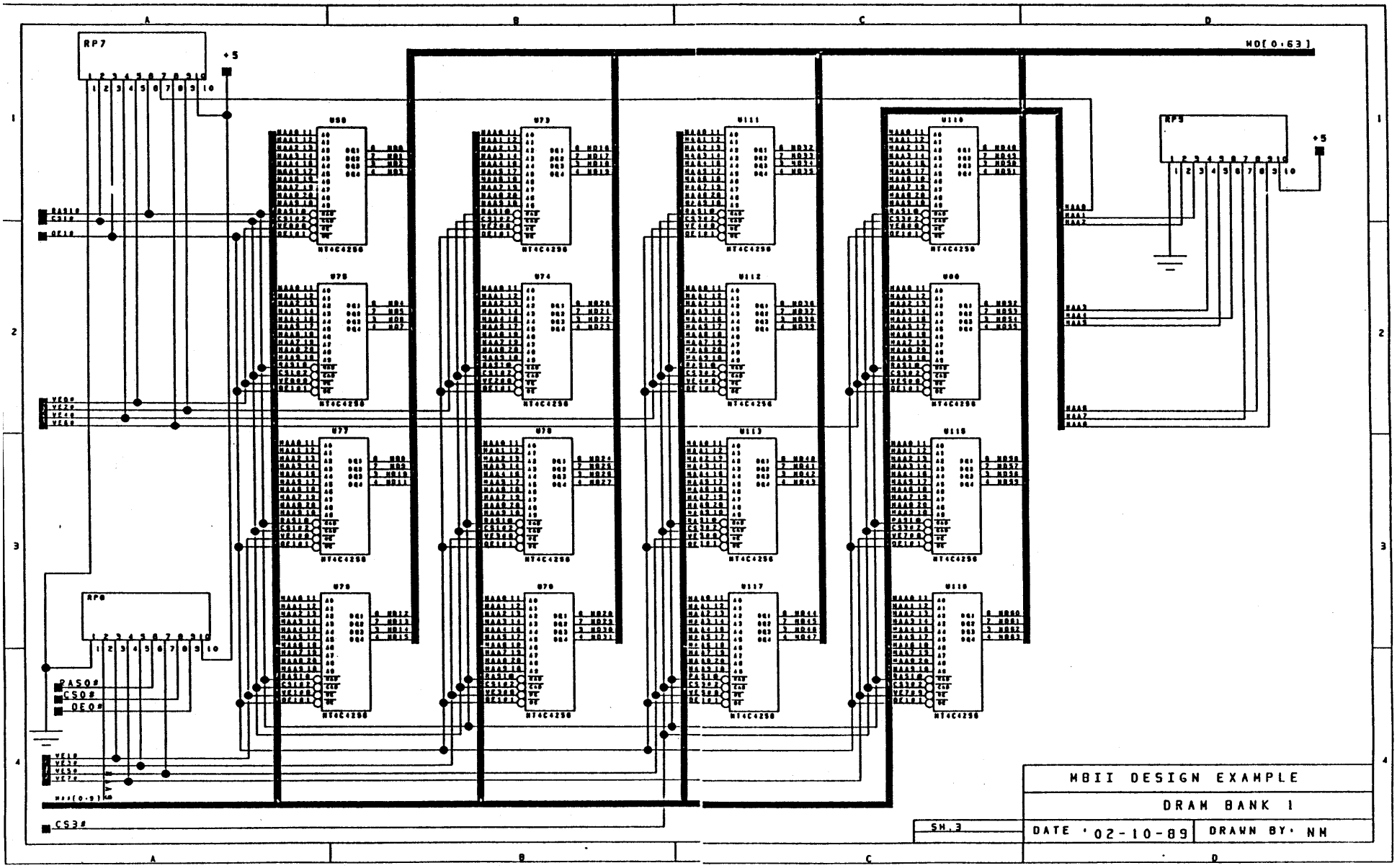
SMD PRODUCT MARKETING	
EXPANSION BUS	SH. 17
FRAME BUFFER REV 01	DATE · 11-20-88 / DRAWN BY · JP

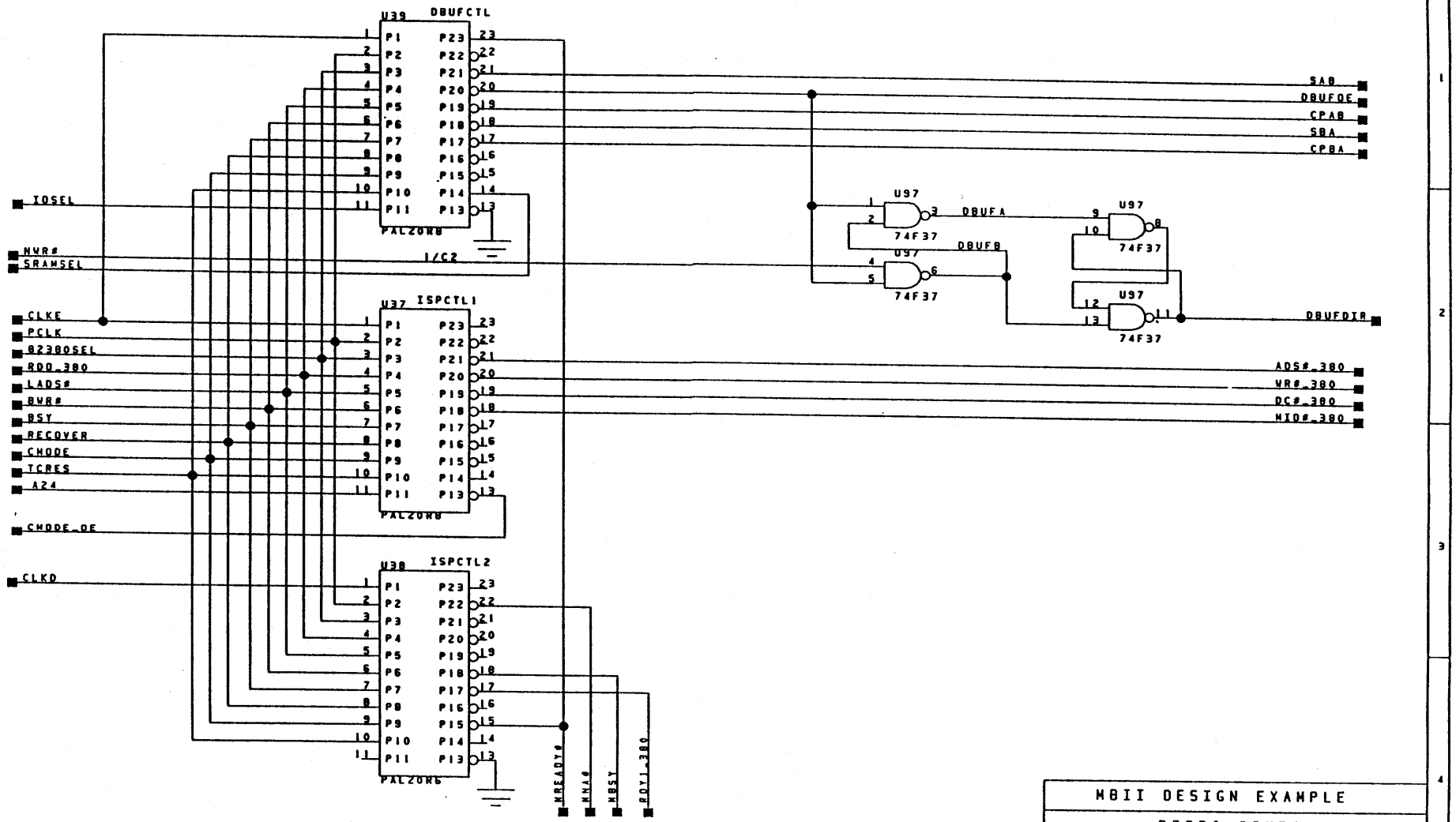
Appendix B
MULTIBUS II *Schematics*

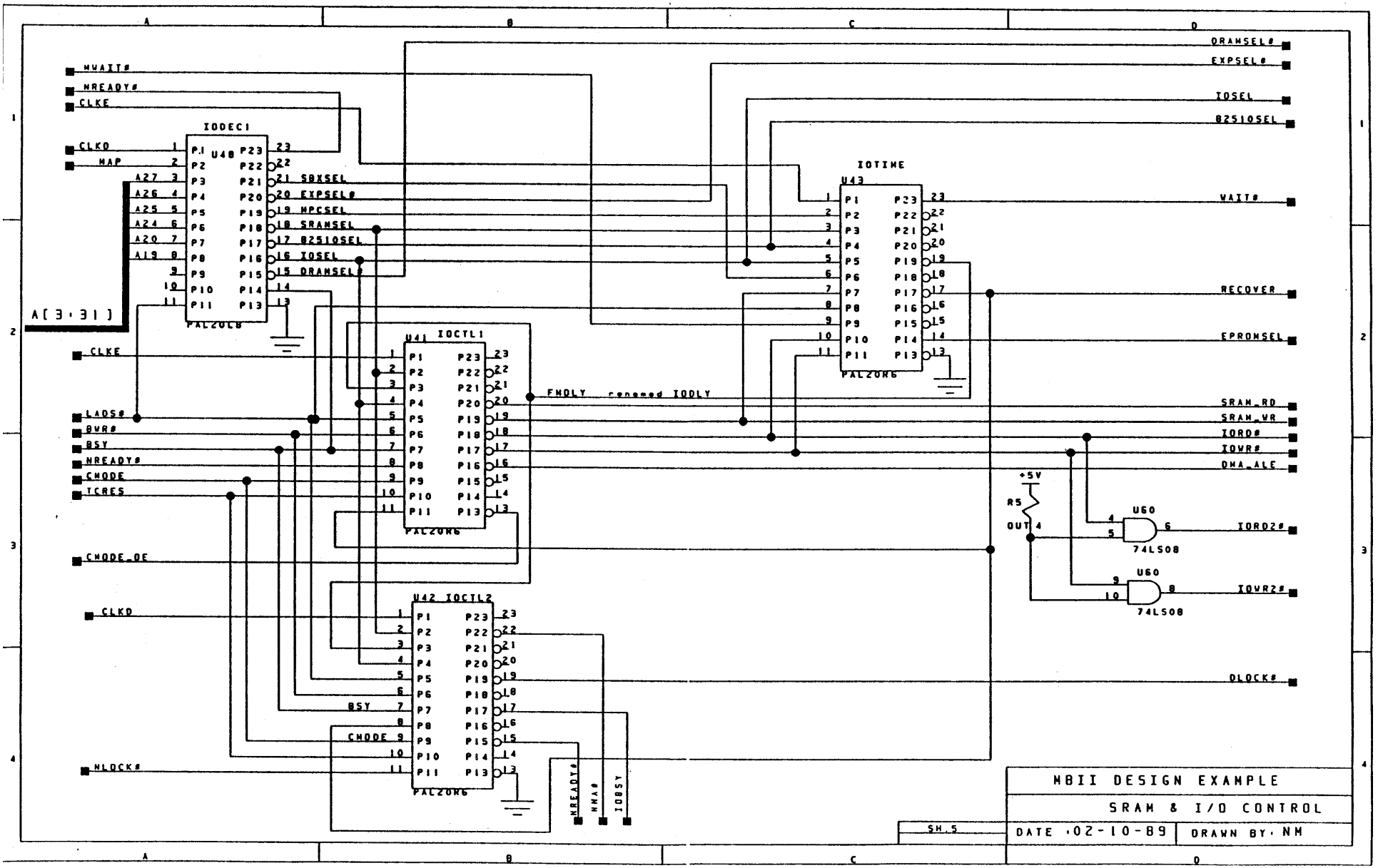
*This design has not been tested. The tested design will be published at a later date.

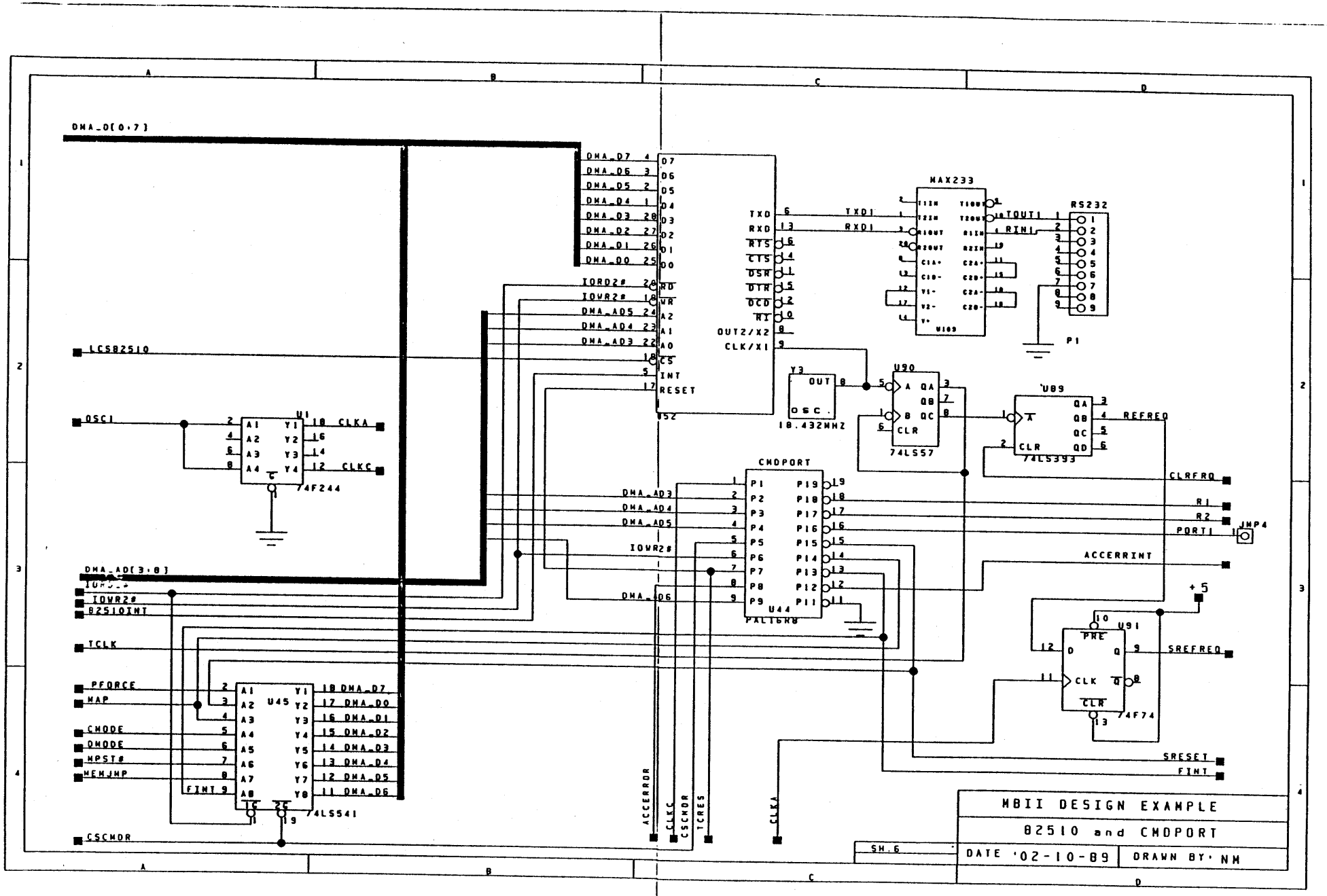

i860 TM CPU
D[0:63]
A[3:31]
40MHZ
74F244
74LS14
PAL16R8
RESET
MBII DESIGN EXAMPLE
CPU and CLOCK
DATE 02-10-89
DRAWN BY NM
SH 1

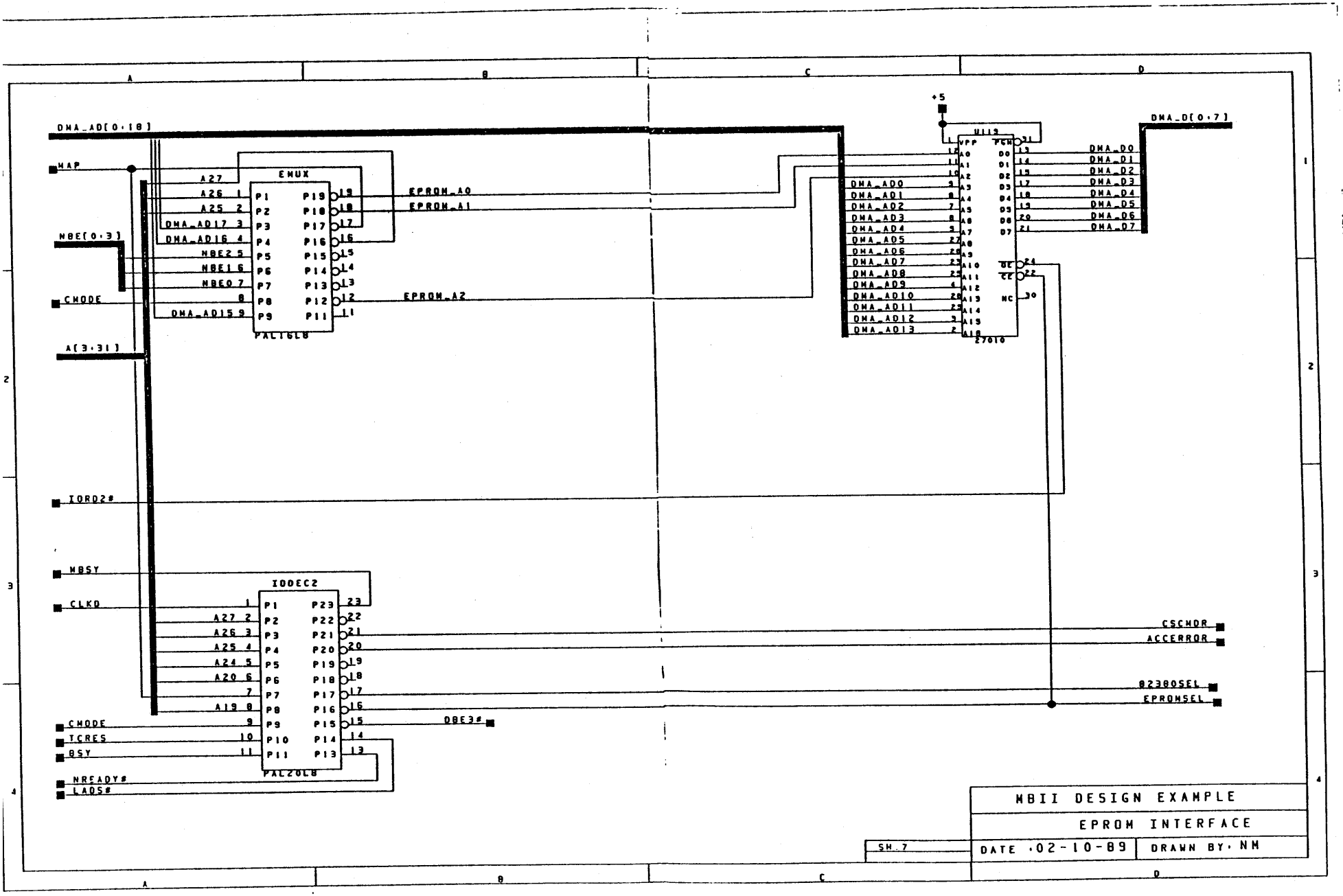

MBII DESIGN EXAMPLE
DRAM BANK 0
SH 2
DATE : 02-10-89
DRAWN BY: NM

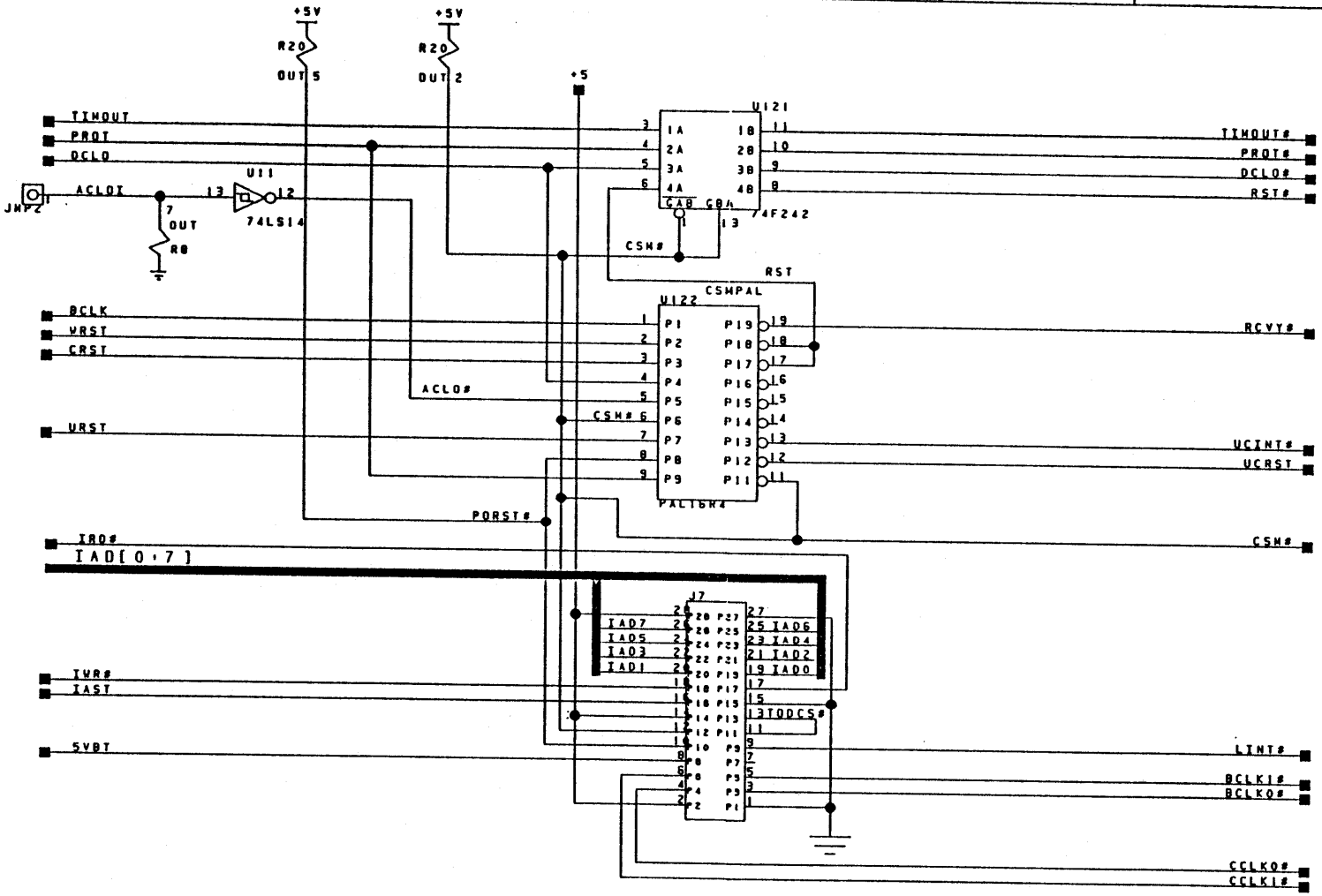

RP7
+5
MD[0:63]
RP5
+5
RP0
PASO#
CSO#
OEO#
CS3#
MBII DESIGN EXAMPLE
DRAM BANK 1
SH.3
DATE '02-10-89
DRAWN BY: NM


U39 DBUFCTL
PALZ0R8
U37 ISPCTL1
PALZ0R8
U38 ISPCTL2
PALZ0R8
U97
74F37
IOSEL
NVR#
SRAMSEL
CLKE
PCLK
82380SEL
RDO_380
LADS#
BVR#
BSY
RECOVER
CMODE
TCRES
A24
CMODE_OE
CLKO
I/C2
DBUFA
DBUFB
SAB
DBUFOE
CPAB
SBA
CPBA
DBUFDIR
ADS#_380
VR#_380
DC#_380
MIO#_380
NMREADY#
NMA#
NMBSY
RDY1_380
MBII DESIGN EXAMPLE
82380 CONTROL
NM.4
DATE ·02-10-89
DRAWN BY·NM


MWAIT#
NREADY#
CLKE
CLKO
MAP
A27
A26
A25
A24
A20
A19
A[3:31]
IODEC1
U48
PAL20L8
SBXSEL
EXPSEL#
MPCSEL
SRAMSEL
82510SEL
IOSEL
DRAMSEL#
IOCTL1
U41
PAL20R6
FMDLY
renamed IODLY
LADS#
BVR#
BSY
NREADY#
CMODE
TCRES
CMODE_OE
IOCTL2
U42
NLOCK#
NREADY#
MWAIT#
IOBSY
IOTIME
U43
DRAMSEL#
EXPSEL#
IOSEL
82510SEL
WAIT#
RECOVER
EPROMSEL
SRAM_RD
SRAM_WR
IORD#
IOWR#
DMA_ALE
+5V
R5
OUT
U60
74LS08
IORD2#
IOWR2#
DLOCK#
MBII DESIGN EXAMPLE
SRAM & I/O CONTROL
SH 5
DATE 02-10-89
DRAWN BY: NM


DMA_D[0:7]
LCS82510
OSC1
DMA_AD[3:8]
IOWR2#
8251OINT
TCLK
PFORCE
MAP
CMODE
DMODE
MPST#
MEMJMP
FINT
CSCMDR
U1
74F244
U45
74LS541
U52
TXD
RXD
RTS
CTS
DSR
DTR
DCD
RI
OUT2/X2
CLK/X1
RESET
INT
IORD2#
IOWR2#
DMA_AD5
DMA_AD4
DMA_AD3
CS
RD
WR
A2
A1
A0
MAX233
TXD1
RXD1
TOUT1
RIN1
RS232
P1
Y3
OSC
18.432MHZ
U90
74LS57
U89
74LS393
REFREQ
CLRFRQ
R1
R2
PORT1
JMP4
ACCERRINT
+5
U91
74F74
SREFREQ
SRESET
FINT
CMDPORT
PAL16R8
U44
DMA_AD6
ACCERROR
CLKC
CSCMDR
TCRES
CLKA
SH.6
MBII DESIGN EXAMPLE
82510 and CMDPORT
DATE '02-10-89
DRAWN BY: NM

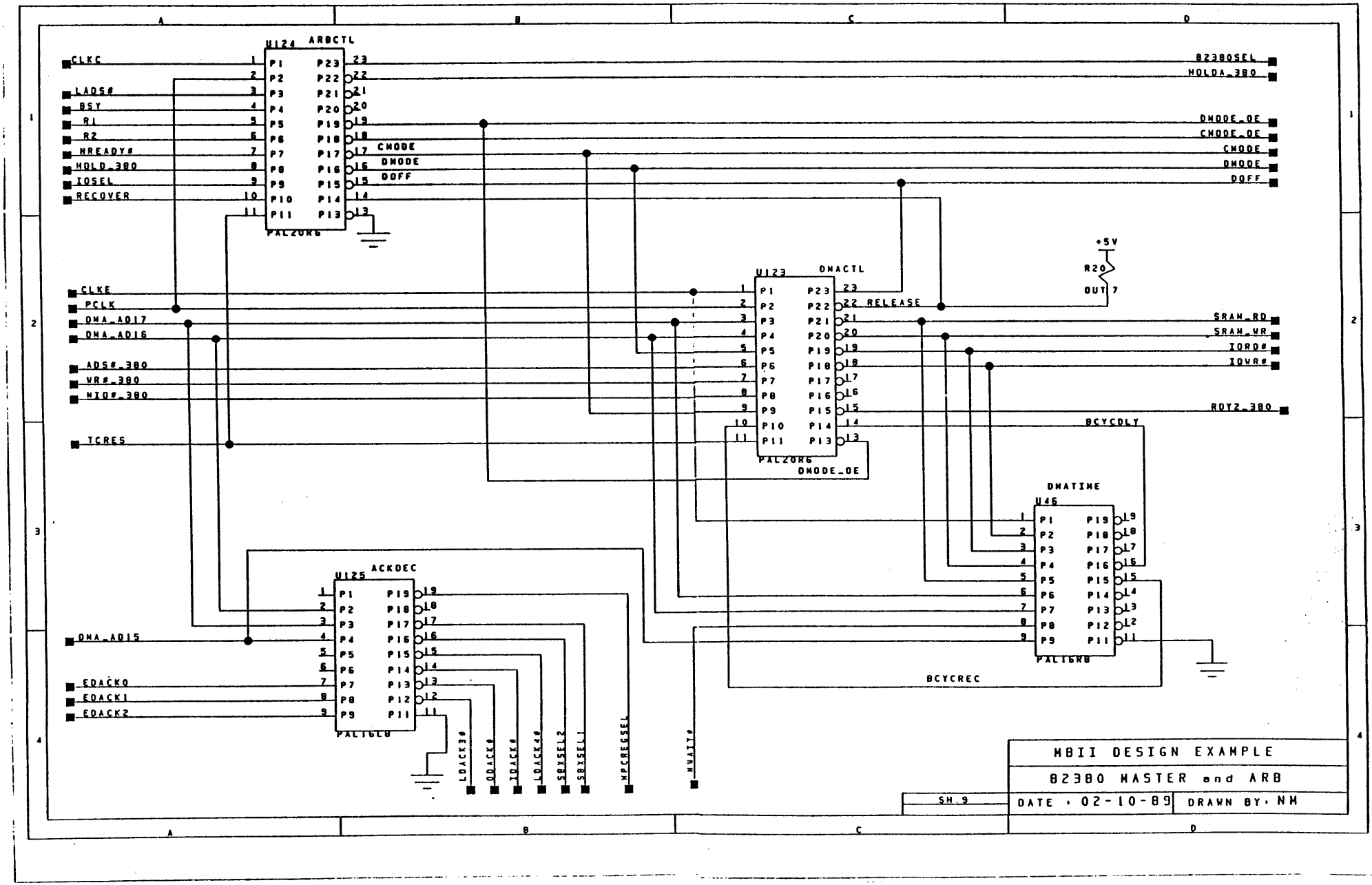

DMA_AD[0:18]
MAP
NBE[0:3]
CMODE
A[3:31]
IORD2#
NBSY
CLKO
CMODE
TCRES
BSY
NREADY#
LADS#
EMUX
PAL16L8
IODEC2
PAL20L8
EPROM_A0
EPROM_A1
EPROM_A2
DBE3#
+5
U113
27010
DMA_D[0:7]
DMA_D0
DMA_D1
DMA_D2
DMA_D3
DMA_D4
DMA_D5
DMA_D6
DMA_D7
CSCMDR
ACCERROR
82380SEL
EPROMSEL
MBII DESIGN EXAMPLE
EPROM INTERFACE
SH 7
DATE 02-10-89
DRAWN BY: NM

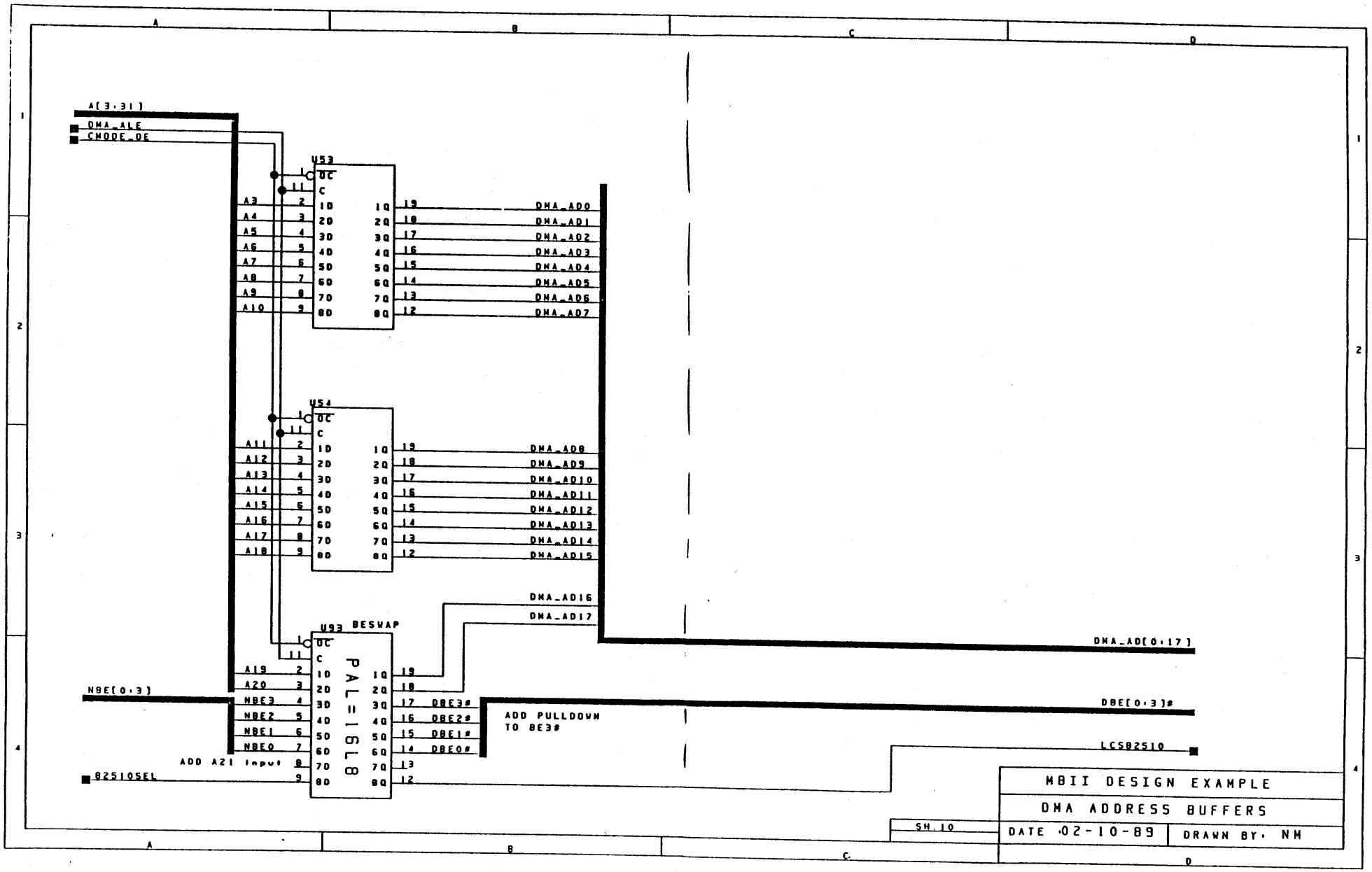


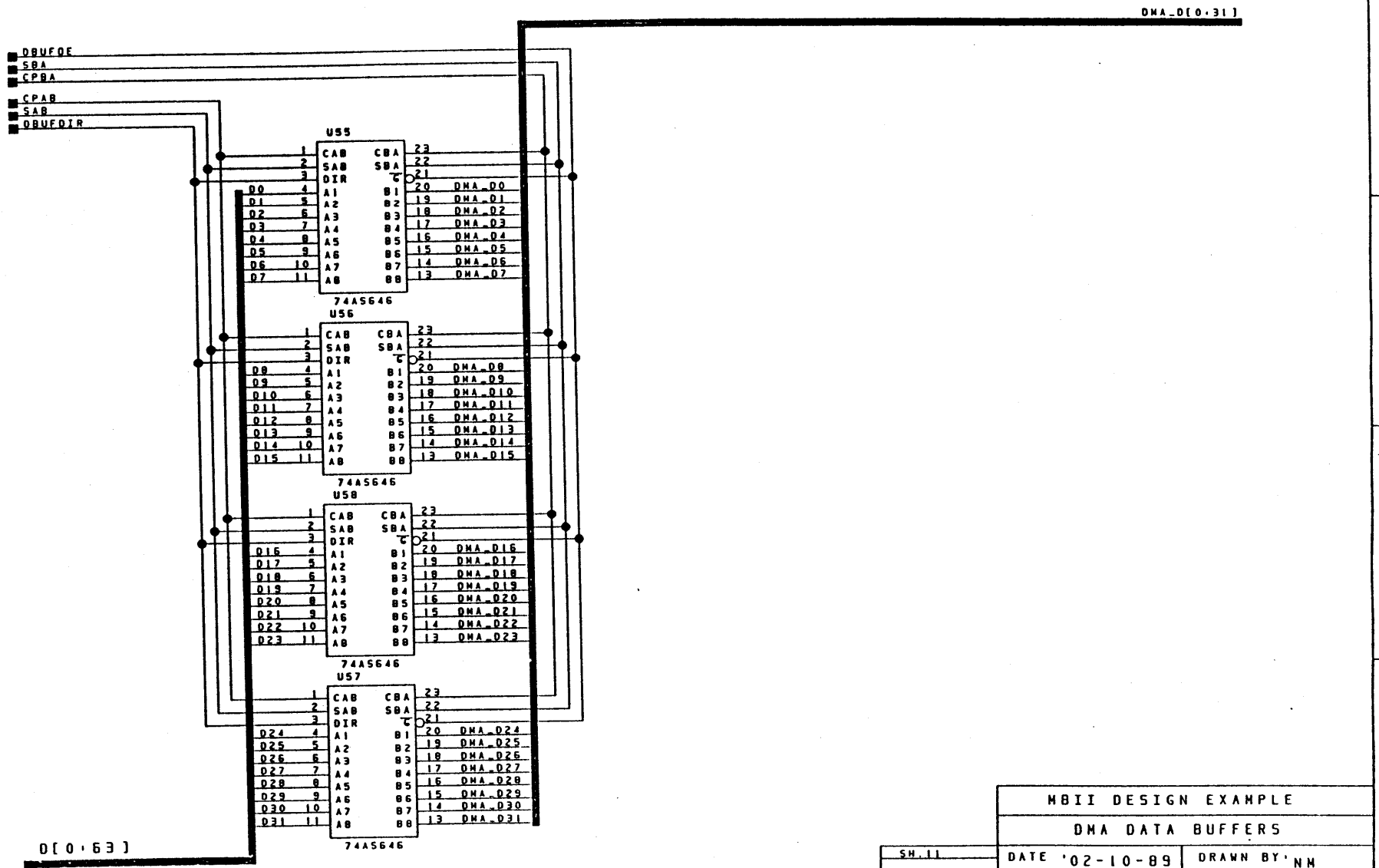
MBII DESIGN EXAMPLE

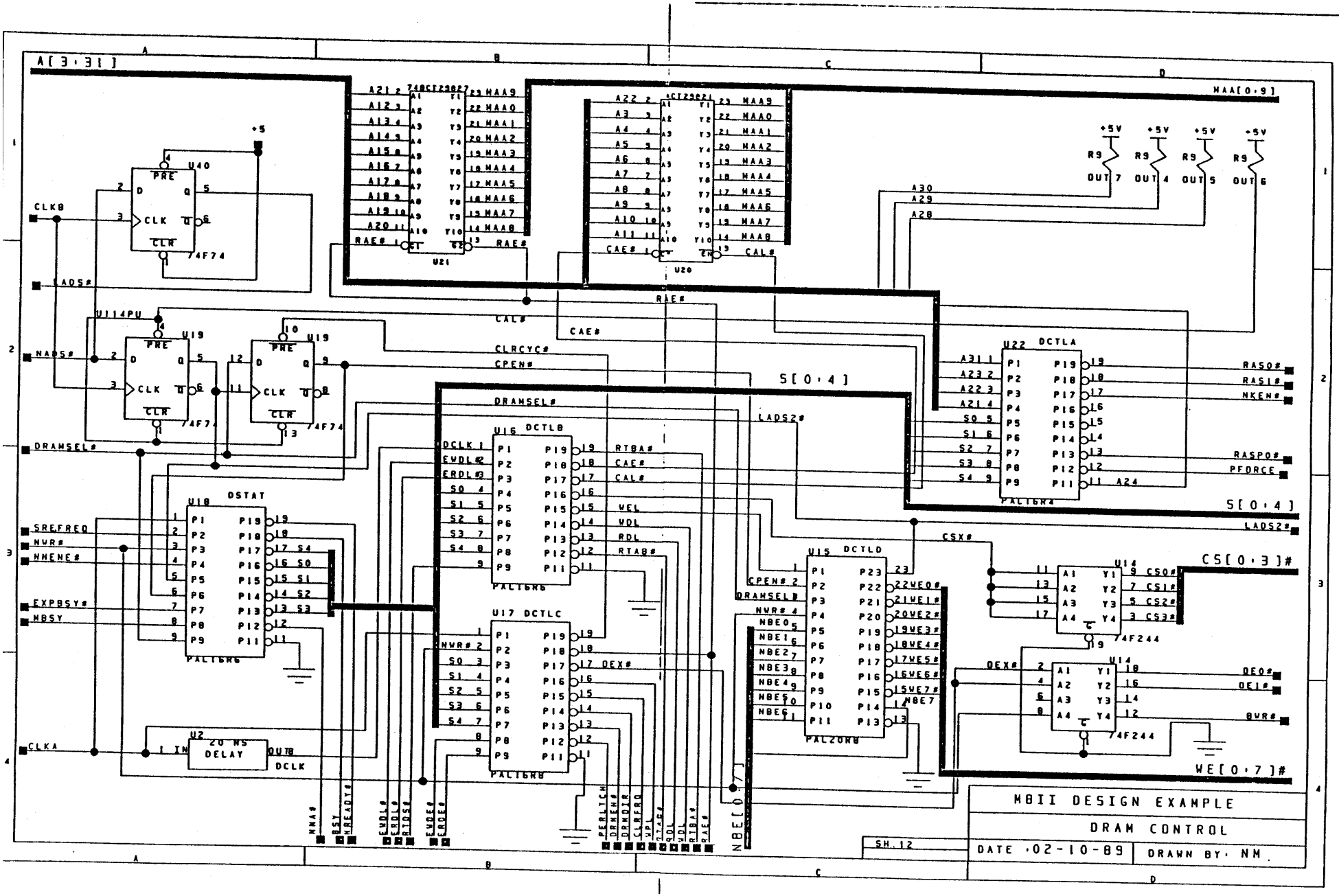
CSM INTERFACE

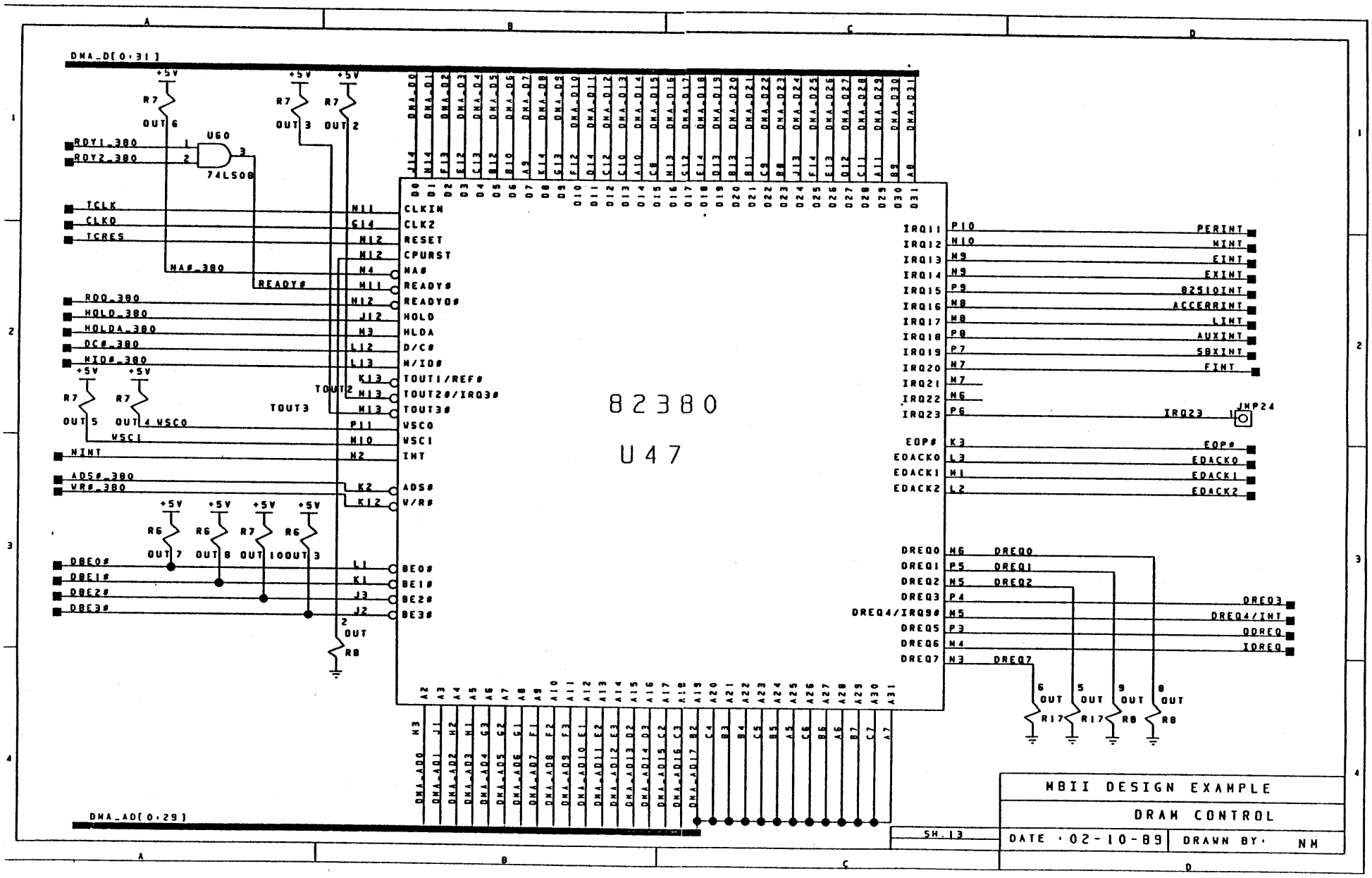
SH.8 | DATE : 02-10-89 | DRAWN BY: NM

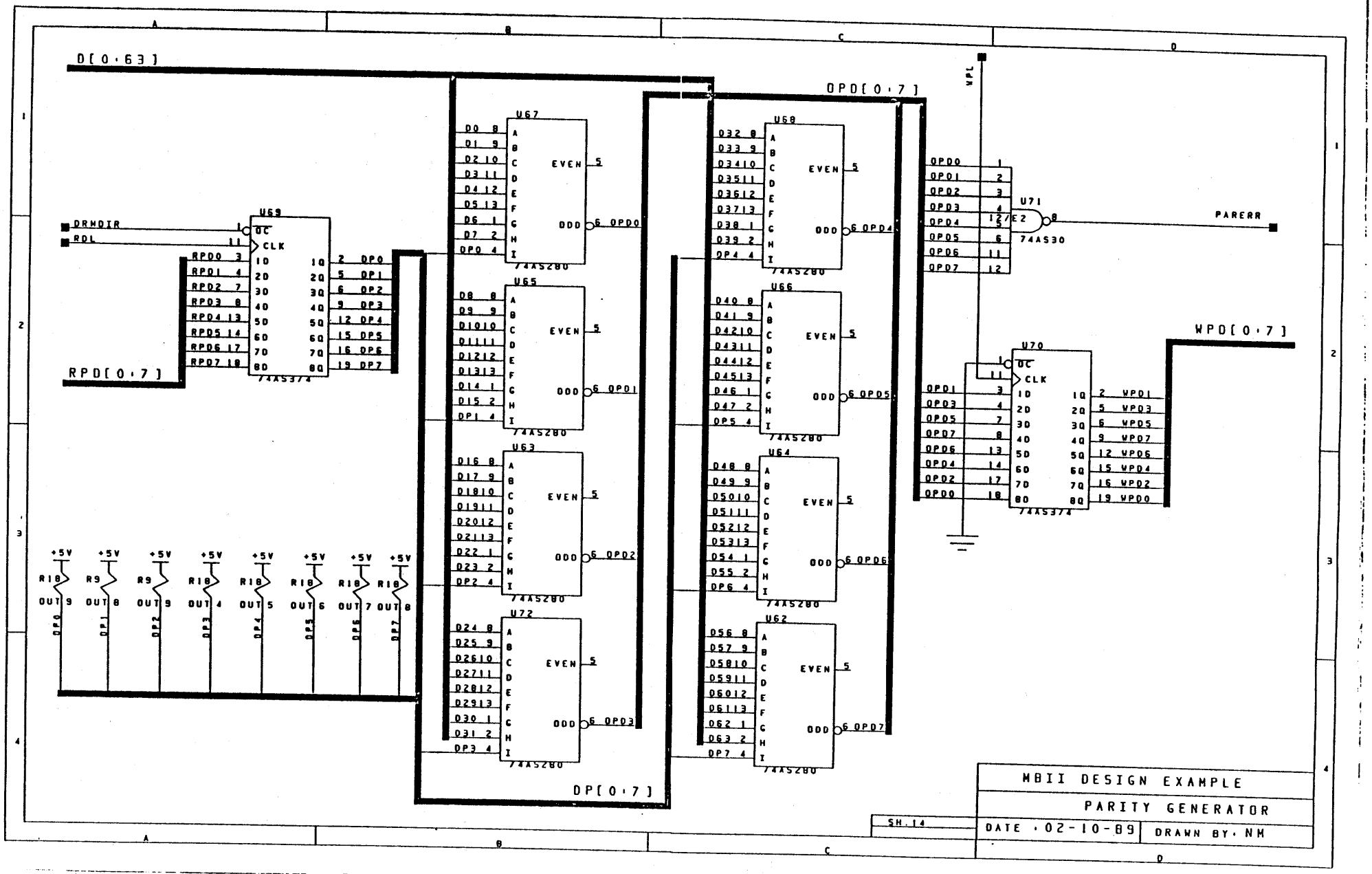

U124 ARBCTL
PAL20R6
CLKC
LADS#
BSY
RI
RZ
NREADY#
HOLD_380
IOSEL
RECOVER
CMODE
DMODE
DOFF
82380SEL
HOLDA_380
DMODE_OE
CMODE_OE
U123 DMACTL
PAL20R6
CLKE
PCLK
DMA_AD17
DMA_AD16
ADS#_380
WR#_380
MIO#_380
TCRES
RELEASE
+5V
R20
OUT 7
SRAM_RD
SRAM_WR
IORD#
IOWR#
RDY2_380
BCYCDLY
DMATIME
U46
PAL16R8
BCYCREC
ACKDEC
U125
PAL16L8
DMA_AD15
EDACK0
EDACK1
EDACK2
LDACK3#
GDACK#
IDACK#
LDACK4#
SRXSEL2
SRXSEL1
MPCREGSEL
MWAIT#
MBII DESIGN EXAMPLE
82380 MASTER and ARB
SH 9
DATE : 02-10-89
DRAWN BY: NM

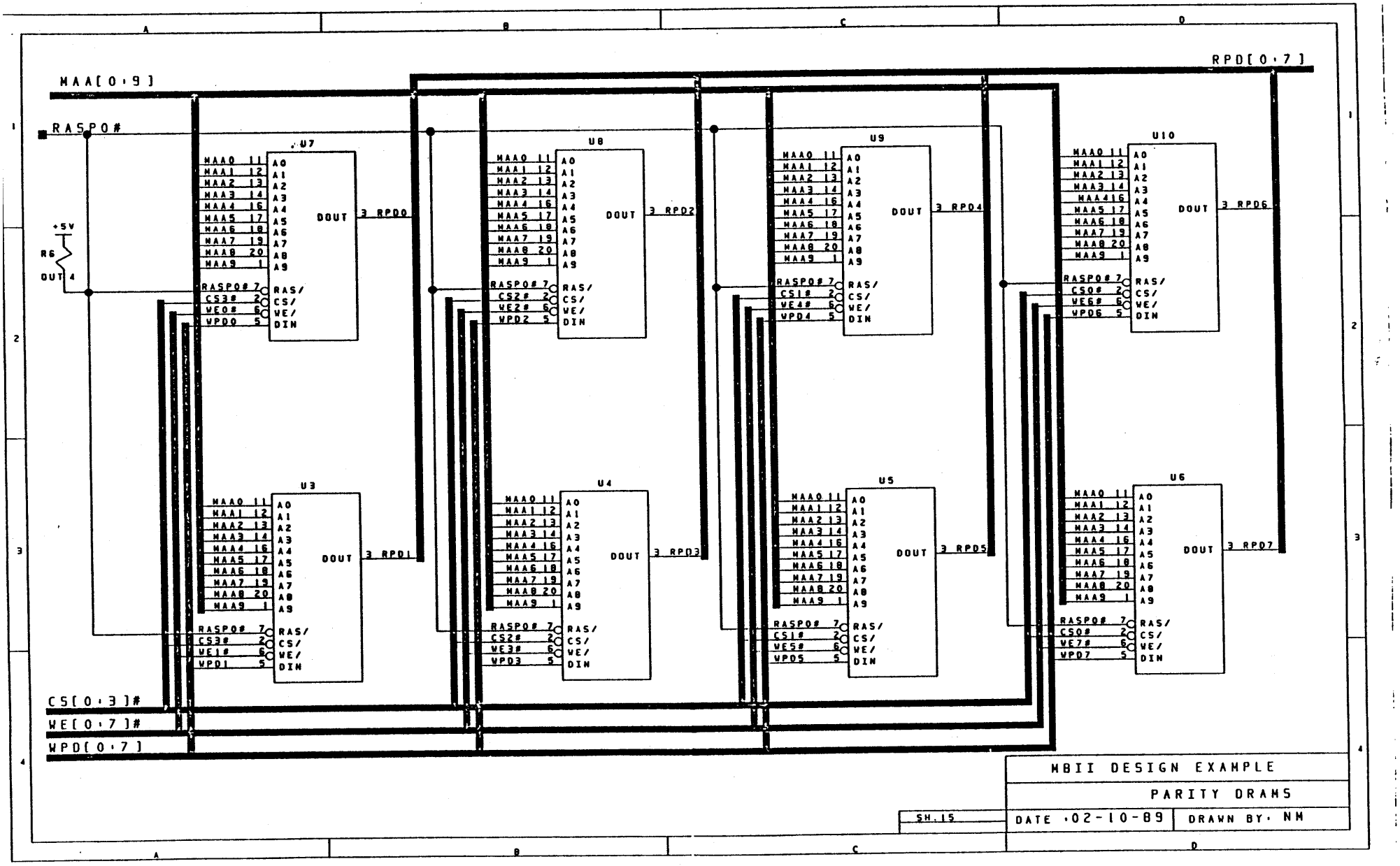

A[3:31]
DMA_ALE
CMODE_OE
U53
U54
U93 BESWAP
PAL=16L8
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
A16
A17
A18
A19
A20
DMA_AD0
DMA_AD1
DMA_AD2
DMA_AD3
DMA_AD4
DMA_AD5
DMA_AD6
DMA_AD7
DMA_AD8
DMA_AD9
DMA_AD10
DMA_AD11
DMA_AD12
DMA_AD13
DMA_AD14
DMA_AD15
DMA_AD16
DMA_AD17
DMA_AD[0:17]
NBE[0:3]
NBE3
NBE2
NBE1
NBE0
BE3#
BE2#
BE1#
BE0#
DBE[0:3]#
ADD PULLDOWN
TO BE3#
ADD A21 Input
82510SEL
LCS82510
SH.10
M8II DESIGN EXAMPLE
DMA ADDRESS BUFFERS
DATE 02-10-89
DRAWN BY: NM

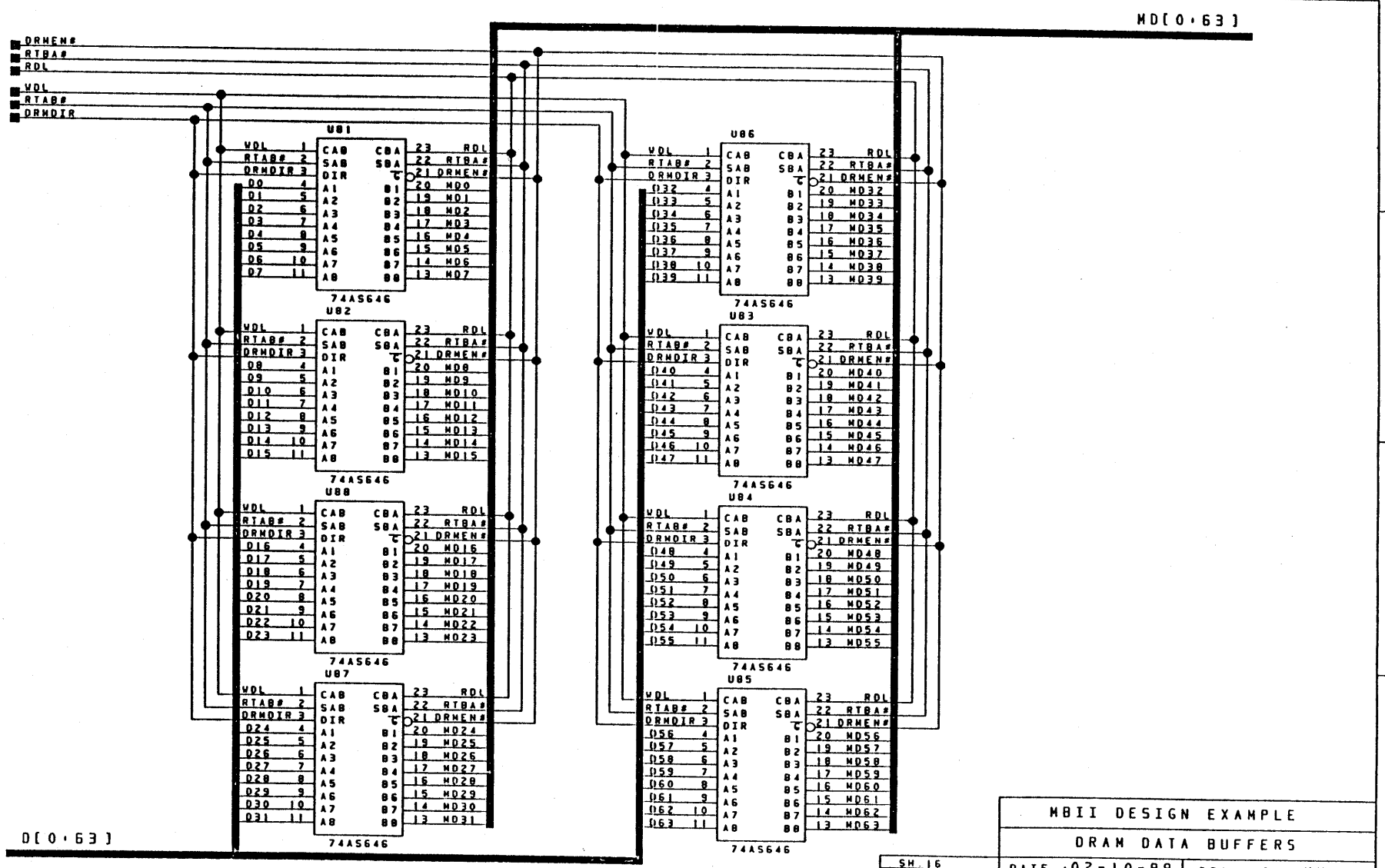

DMA_D[0:31]
DBUFOE
SBA
CPBA
CPAB
SAB
DBUFDIR
U55
U56
U58
U57
74AS646
CAB
SAB
DIR
CBA
SBA
A1
A2
A3
A4
A5
A6
A7
A8
B1
B2
B3
B4
B5
B6
B7
B8
D0
D1
D2
D3
D4
D5
D6
D7
DMA_D0
DMA_D1
DMA_D2
DMA_D3
DMA_D4
DMA_D5
DMA_D6
DMA_D7
D8
D9
D10
D11
D12
D13
D14
D15
DMA_D8
DMA_D9
DMA_D10
DMA_D11
DMA_D12
DMA_D13
DMA_D14
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[3:3]
MAA[0:9]
S[0:4]
CS[0:3]#
WE[0:7]#
NBE[0:7]
U40
U19
U114PU
U21
U20
U16 DCTLB
U17 DCTLC
U18 DSTAT
U15 DCTLD
U22 DCTLA
U14
U2
74F74
74F244
PAL16R6
PAL20R8
20 NS DELAY
+5
CLKB
CLKA
LADS#
NADS#
DRAMSEL#
CLRCYC#
CPEN#
CAE#
CAL#
RAE#
RAE#
CAL#
LADS2#
CSX#
OEX#
RAS0#
RAS1#
NKEN#
RASP0#
PFORCE
A24
A30
A29
A28
SREFREQ
NVR#
NNENE#
EXPBSY#
MBSY
DCLK
RTBA#
CAE#
CAL#
VEL
VDL
RDL
RTAB#
OEX#
NBE0
NBE1
NBE2
NBE3
NBE4
NBE5
NBE6
NBE7
CS0#
CS1#
CS2#
CS3#
OE0#
OE1#
BVR#
R9
OUT
MBII DESIGN EXAMPLE
DRAM CONTROL
DATE 02-10-89
DRAWN BY: NM
SH 12

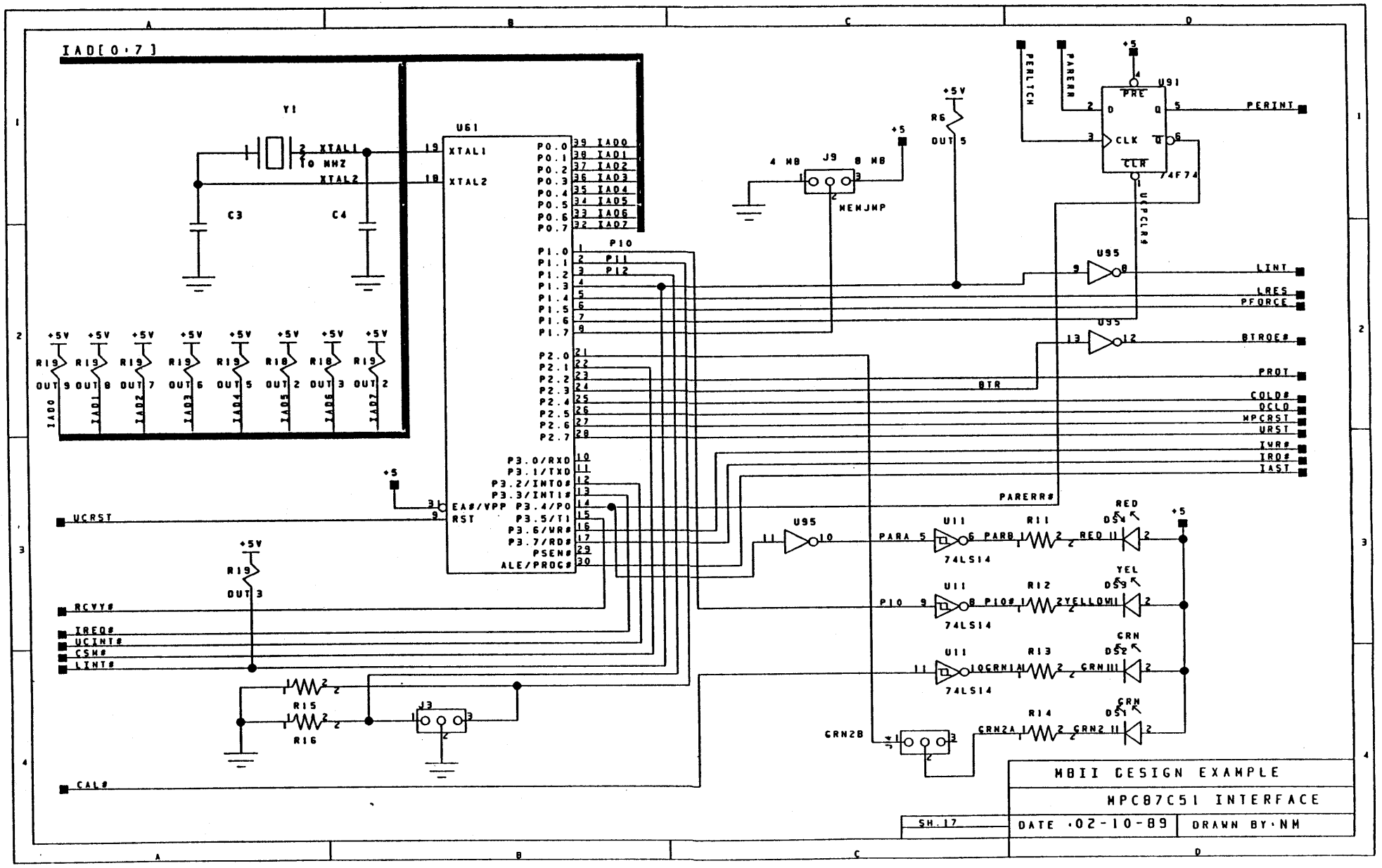

82380
U47
M8II DESIGN EXAMPLE
DRAM CONTROL
DATE · 02-10-89
DRAWN BY· NM
SH.13

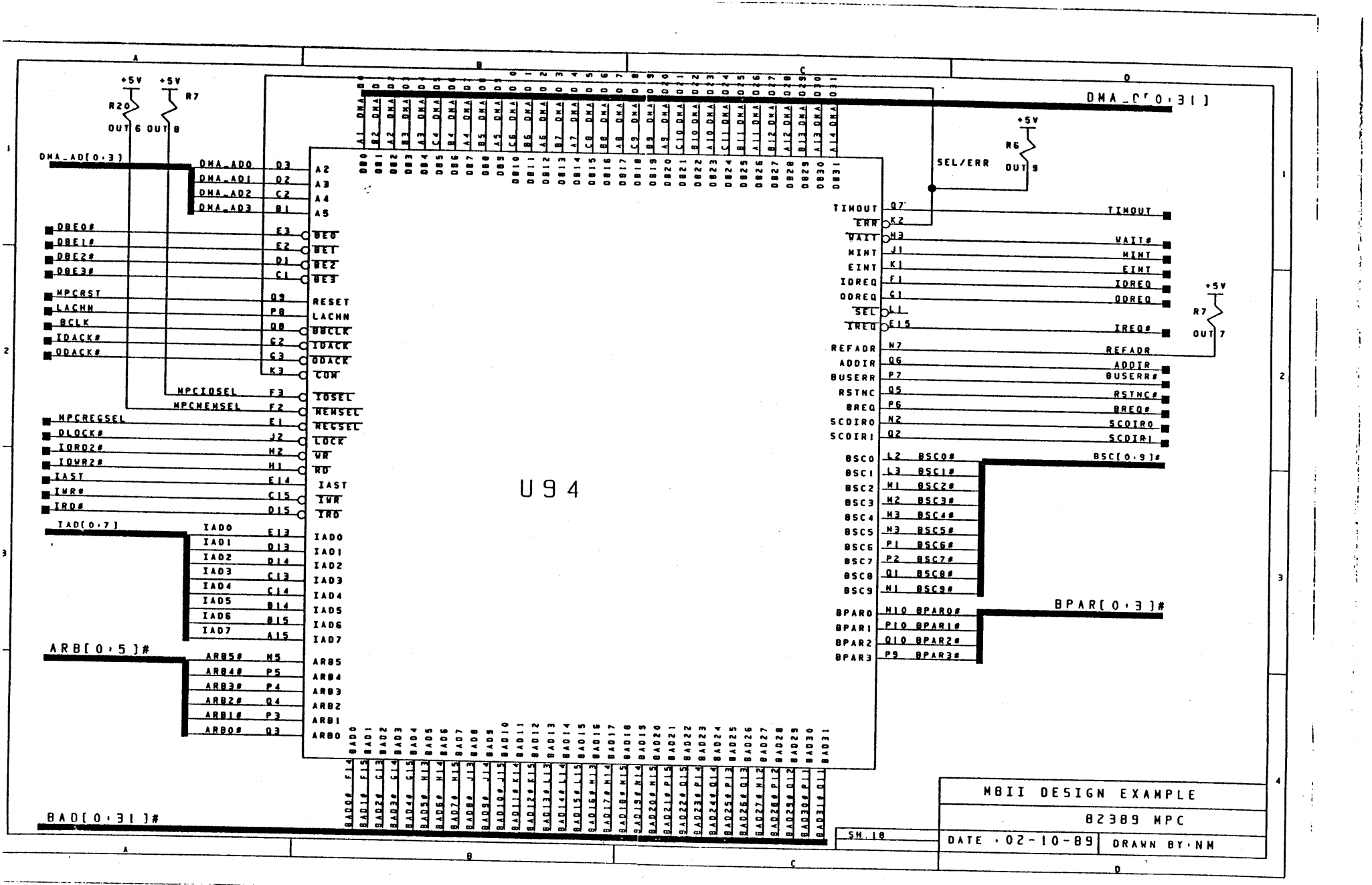

D[0:63]
DRMDIR
RDL
RPD[0:7]
U69
74AS374
U67
U65
U63
U72
U68
U66
U64
U62
74AS280
EVEN
ODD
DP[0:7]
OPD[0:7]
WPL
U71
74AS30
PARERR
U70
WPD[0:7]
+5V
R18
R9
OUT
MBII DESIGN EXAMPLE
PARITY GENERATOR
SH.14
DATE:02-10-89
DRAWN BY:NM

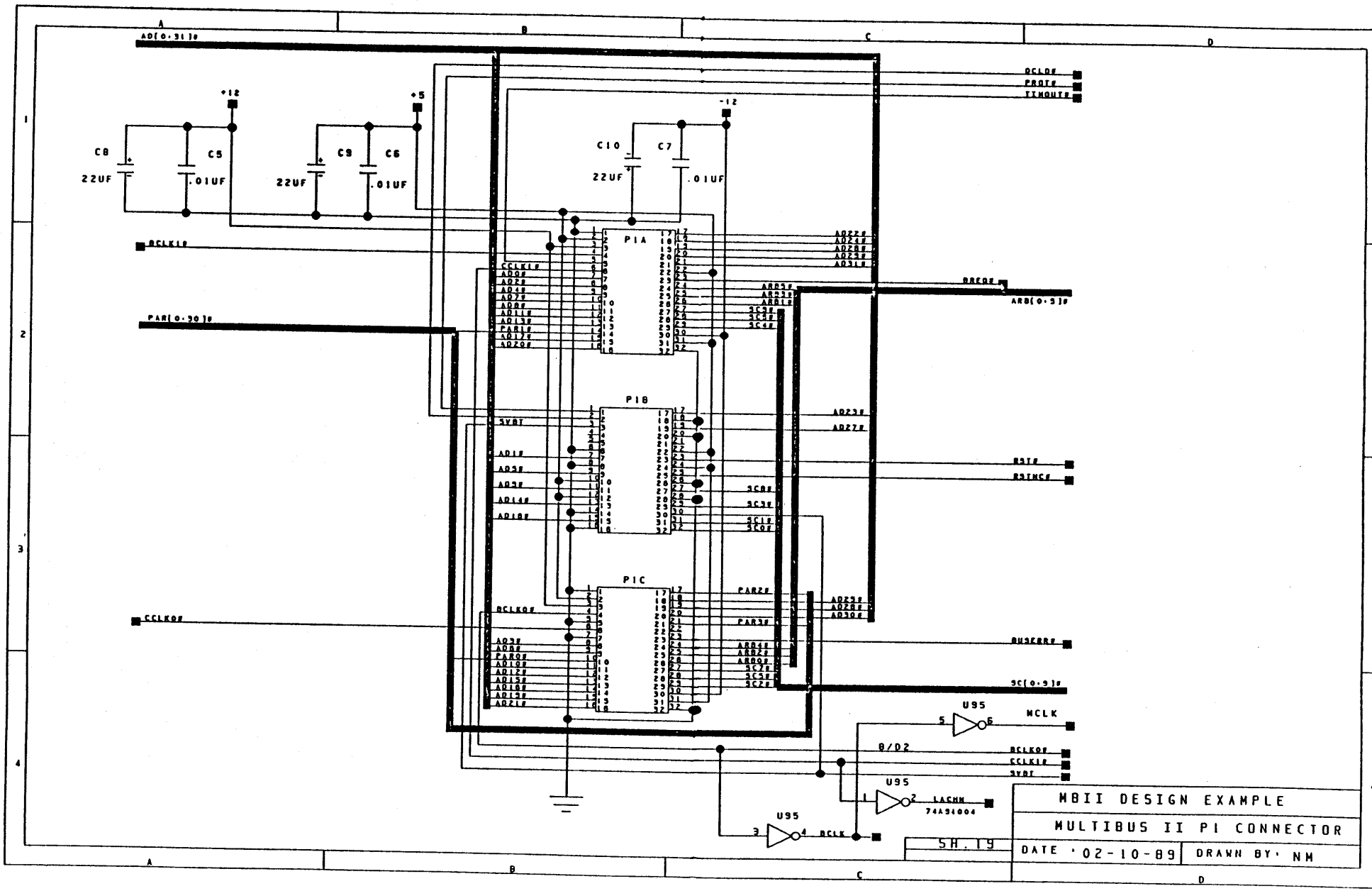

MAA[0:9]
RPD[0:7]
RASP0#
+5V
R6
OUT 4
U7
U8
U9
U10
U3
U4
U5
U6
MAA0 1
MAA1 12
MAA2 13
MAA3 14
MAA4 16
MAA5 17
MAA6 18
MAA7 19
MAA8 20
MAA9 1
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
DOUT
3 RPD0
3 RPD2
3 RPD4
3 RPD6
3 RPD1
3 RPD3
3 RPD5
3 RPD7
RASP0# 7
CS3# 2
WE0# 6
WPD0 5
CS2# 2
WE2# 6
WPD2 5
CS1# 2
WE4# 6
WPD4 5
CS0# 2
WE6# 6
WPD6 5
CS3# 2
WE1# 6
WPD1 5
CS2# 2
WE3# 6
WPD3 5
CS1# 2
WE5# 6
WPD5 5
CS0# 2
WE7# 6
WPD7 5
RAS/
CS/
WE/
DIN
CS[0:3]#
WE[0:7]#
WPD[0:7]
MBII DESIGN EXAMPLE
PARITY DRAMS
SH.15
DATE 02-10-89
DRAWN BY: NM

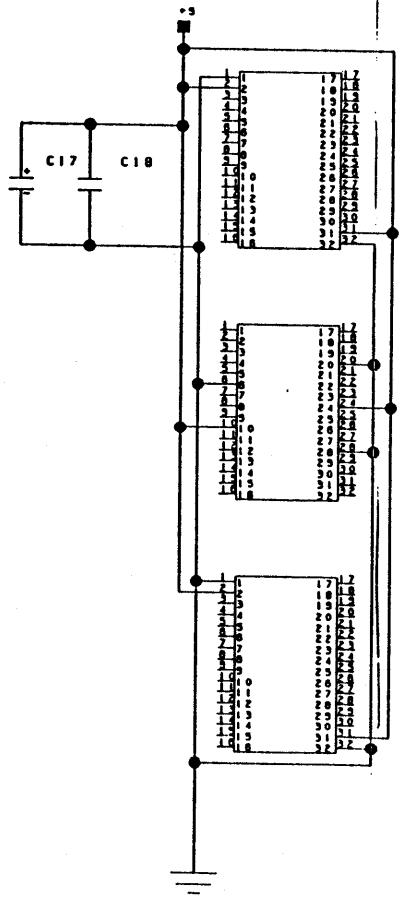


MBII DESIGN EXAMPLE	
DRAM DATA BUFFERS	
DATE :02-10-89	DRAWN BY: NM
SH 16	

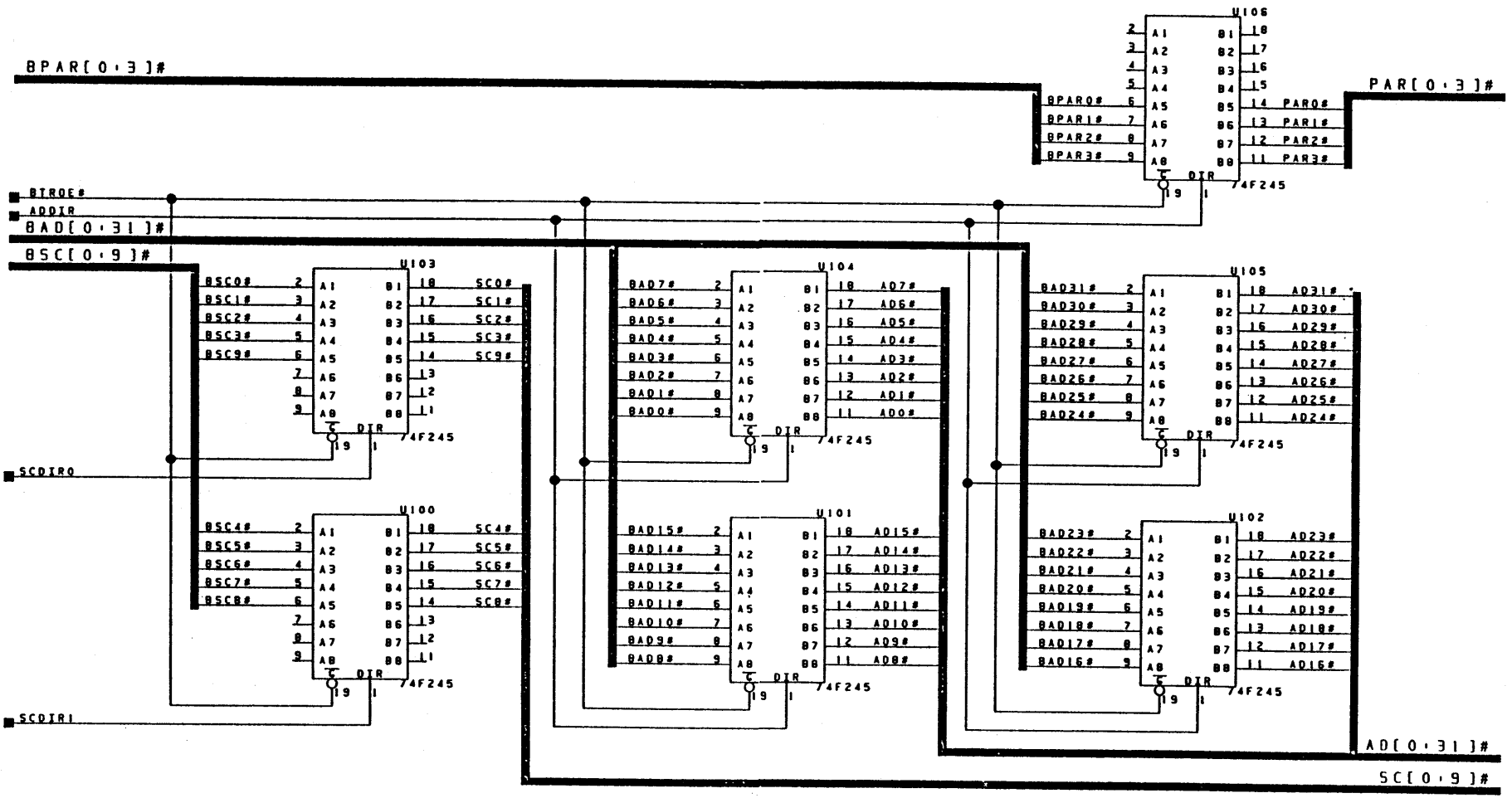

IAD[0:7]
Y1
10 MHZ
XTAL1
XTAL2
C3
C4
U61
P0.0 IAD0
P0.1 IAD1
P0.2 IAD2
P0.3 IAD3
P0.4 IAD4
P0.5 IAD5
P0.6 IAD6
P0.7 IAD7
P1.0 P10
P1.1 P11
P1.2 P12
P1.3
P1.4
P1.5
P1.6
P1.7
P2.0
P2.1
P2.2
P2.3
P2.4
P2.5
P2.6
P2.7
P3.0/RXD
P3.1/TXD
P3.2/INT0#
P3.3/INT1#
P3.4/T0
P3.5/T1
P3.6/WR#
P3.7/RD#
PSEN#
ALE/PROG#
EA#/VPP
RST
R19
R18
J9
4 MB
8 MB
MEMJMP
R6
PERLTCH
PARERR
U91
PRE
CLK
CLR
74F74
PERINT
UCPCLR#
U95
LINT
LRES
PFORCE
BTROE#
PROT
COLD#
DCLD
MPCRST
URST
IVR#
IRD#
IAST
BTR
PARERR#
UCRST
RCVY#
IREQ#
UCINT#
CSN#
LINT#
CAL#
R15
R16
J3
U11
R11
R12
R13
R14
PARA
PARB
RED
P10
P10#
YELLOW
CRN
CRN2A
CRN2B
DS4
DS3
DS2
DS1
74LS14
MBII DESIGN EXAMPLE
MPC87C51 INTERFACE
SH.17
DATE ·02-10-89
DRAWN BY·NM


U94
DMA_AD[0:3]
DBE0#
DBE1#
DBE2#
DBE3#
MPCRST
LACHN
BCLK
IDACK#
ODACK#
MPCIOSEL
MPCMEMSEL
MPCREGSEL
DLOCK#
IORD2#
IOWR2#
IAST
IWR#
IRD#
IAD[0:7]
ARB[0:5]#
BAD[0:31]#
DMA_D[0:31]
SEL/ERR
TIMOUT
WAIT#
MINT
EINT
IDREQ
ODREQ
IREQ#
REFADR
ADDIR
BUSERR#
RSTNC#
BREQ#
SCDIRO
SCDIRI
BSC[0:9]#
BPAR[0:3]#
R20
R7
R6
+5V
OUT 6
OUT 8
OUT 9
OUT 7
SH.18
MBII DESIGN EXAMPLE
82389 MPC
DATE · 02-10-89
DRAWN BY · NM

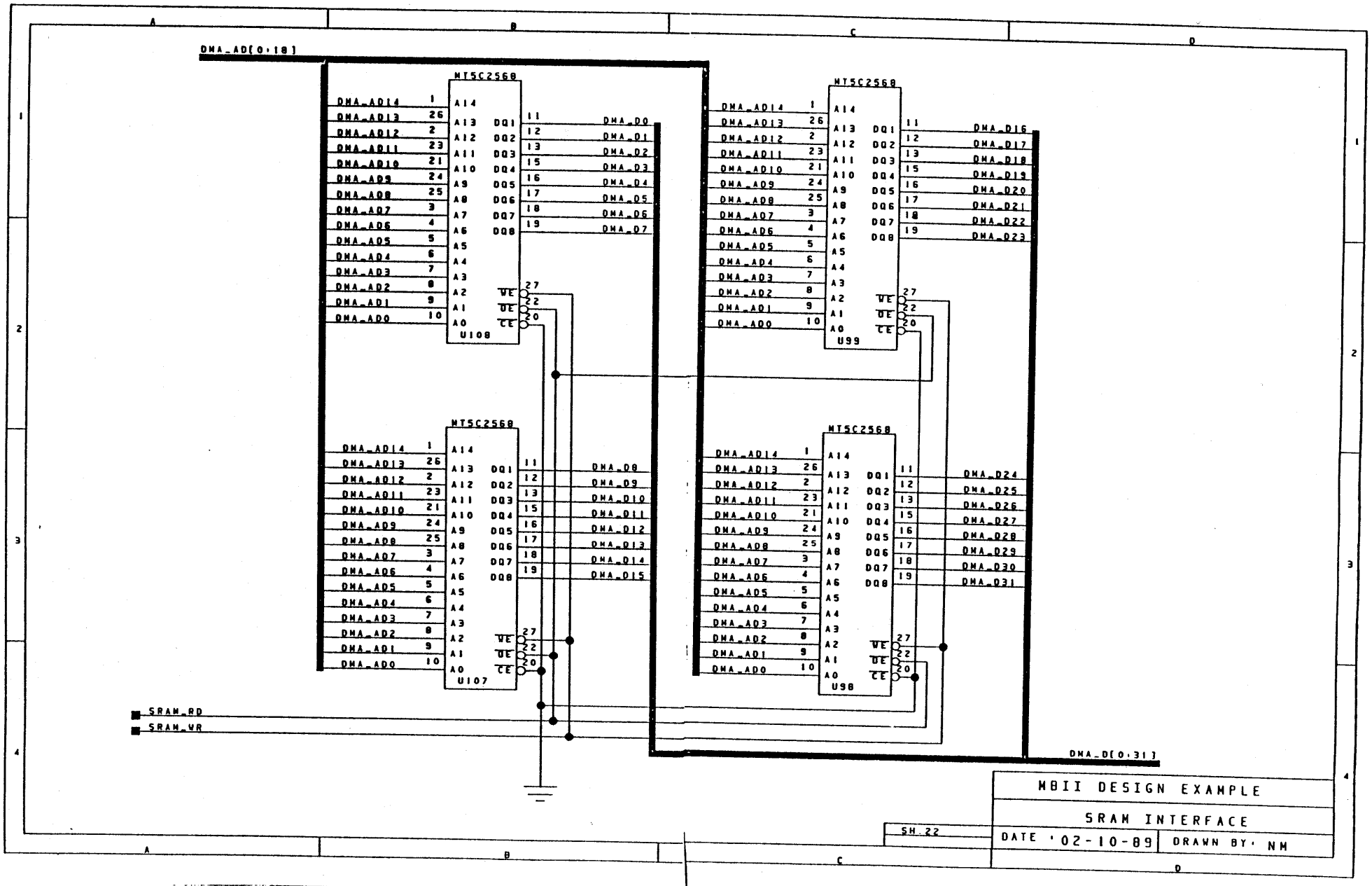

AD[0:31]#
+12
+5
-12
C8
22UF
C5
.01UF
C9
22UF
C6
.01UF
C10
22UF
C7
.01UF
BCLK1#
PAR[0:3]#
CCLK0#
P1A
P1B
P1C
DCLO#
PRO#
TIMOUT#
ARBE#
ARB[0:5]#
RST#
RSINC#
BUSERR#
SC[0:5]#
U95
MCLK
B/D2
BCLK0#
CCLK1#
SYAT
LACHN
74AS1004
BCLK
MBII DESIGN EXAMPLE
MULTIBUS II PI CONNECTOR
SH. 19
DATE '02-10-89
DRAWN BY: NM



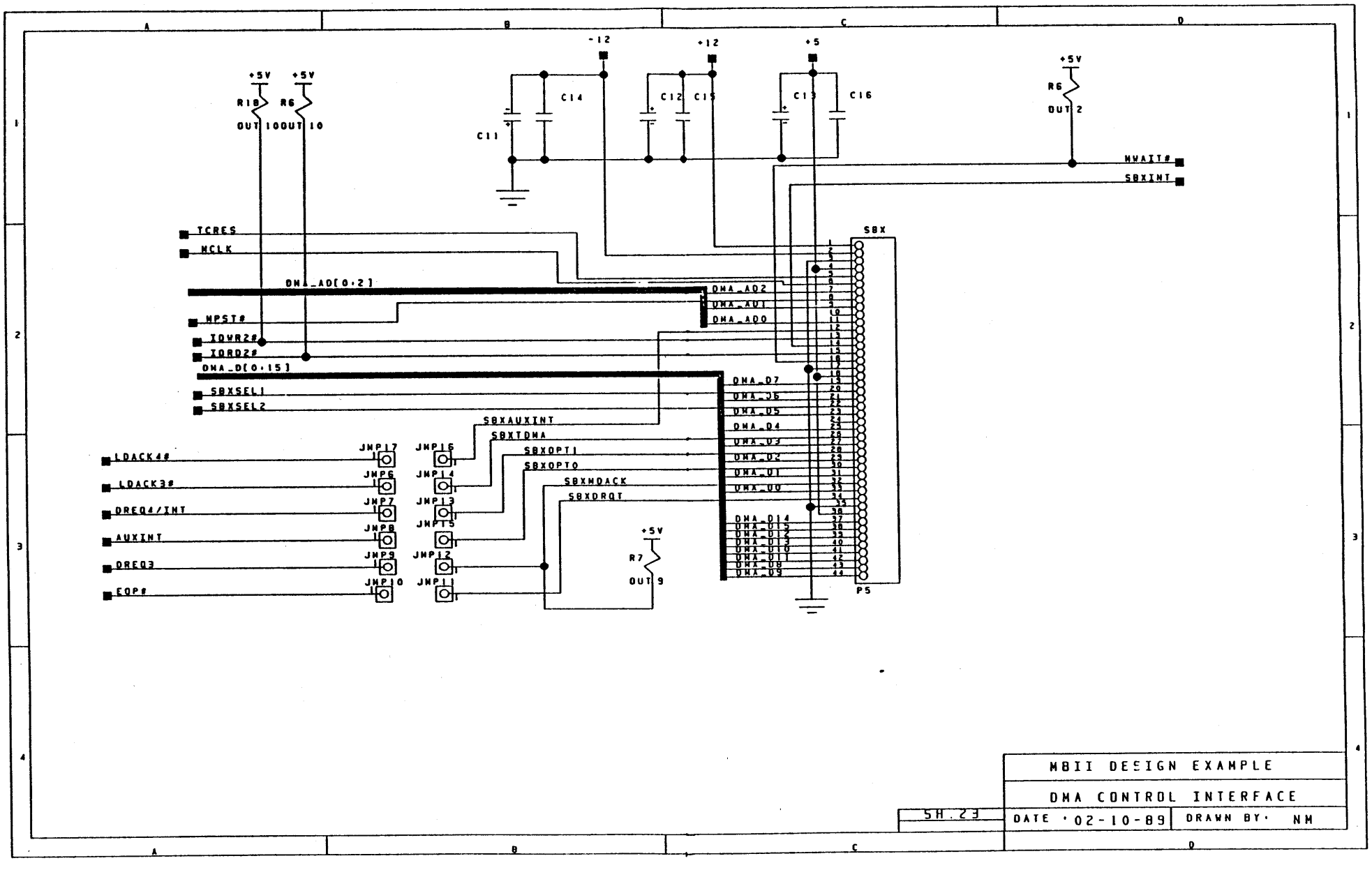
MBII DESIGN EXAMPLE	
MULTIBUS II P2 CONNECTOR	
SH. 20	DATE '02-10-89 DRAWN BY: NM



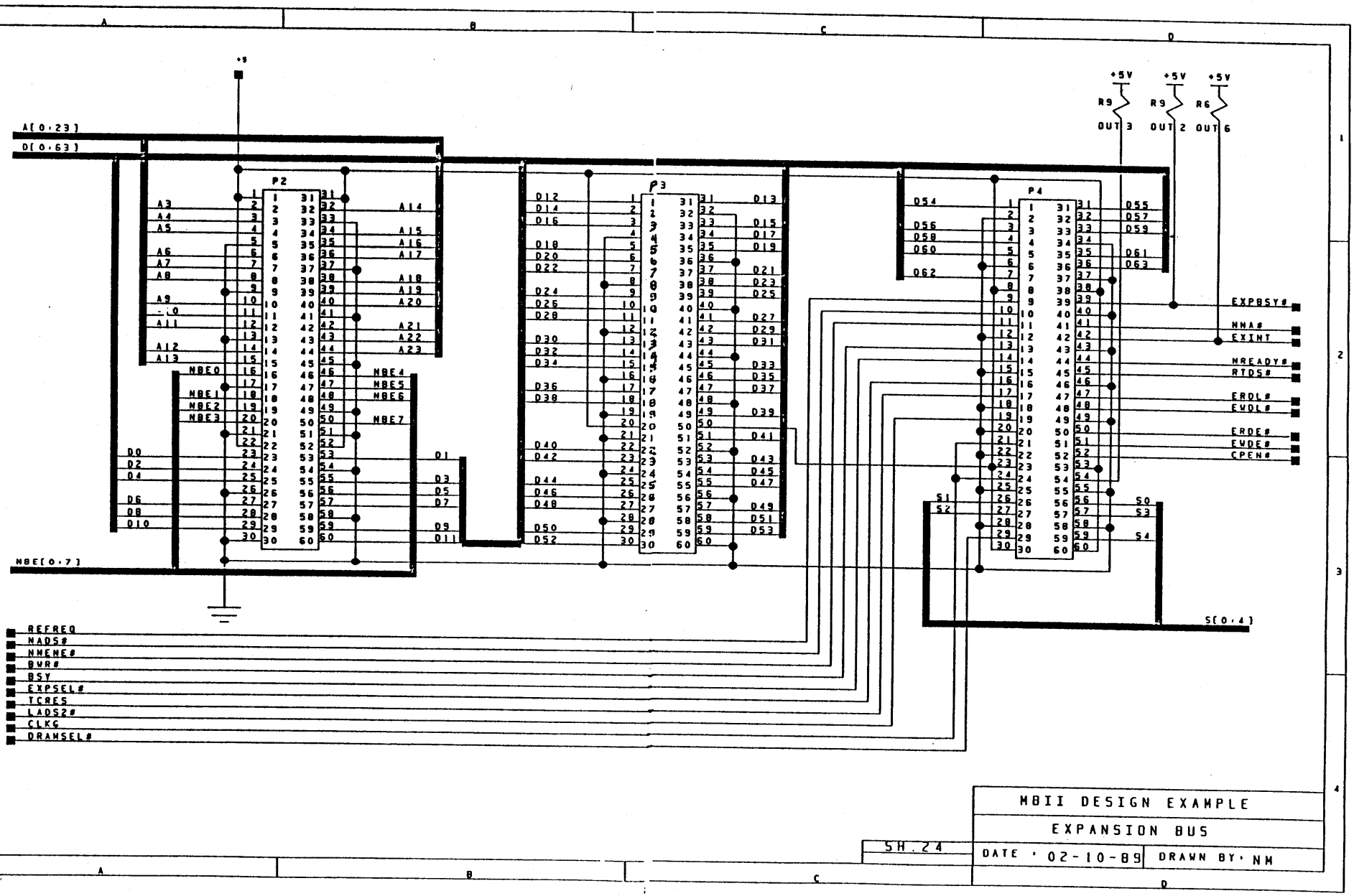
MBII DESIGN EXAMPLE	
MULTIBUSII INTERFACE	
SH.21	DATE ·02-10-89 | DRAWN BY· NM



M8II DESIGN EXAMPLE	
SRAM INTERFACE	
SH 22	DATE : 02-10-89 | DRAWN BY: NM



MBII DESIGN EXAMPLE		
DMA CONTROL INTERFACE		
SH. 23	DATE : 02-10-89	DRAWN BY: NM



MBII DESIGN EXAMPLE	
EXPANSION BUS	
SH. 24	DATE · 02-10-89 / DRAWN BY · NM