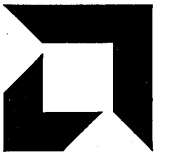
Am29000
32-Bit Streamlined Instruction Processor

Memory Design Handbook

Advanced
Micro
Devices

Advanced Micro Devices



Am29000 32-Bit Streamlined Instruction Processor

Memory Design Handbook

by
Mark McClain, 29K Specialist
Field Applications Engineer
Sherman Lee, 29K Specialist
Field Applications Engineer

PREFACE

The Am29000 changes the meaning of "high performance" for 32-bit CMOS Reduced Instruction Set Computers (RISCs)!

First generation RISCs provided performance in the 4 to 5 million instructions per second (MIPS) range. But, the first member of the Am29000 family of RISC microprocessors can sustain performance in the 10 to 25 MIPS range!

The Am29000 brings high performance to a wide range of cost-sensitive applications ranging from personal computers and embedded controllers using DRAM or VDRAM (10 to 17 MIPS), to extremely high-performance engineering workstations and multi-user systems, using cache or SRAM (17 to over 25 MIPS).

The Am29000 family of microprocessors gives the computer-system designer an entire spectrum of cost-effective system-performance solutions using a single hardware-software platform.

The 29000 provides many features for easing the performance burden placed on system memory so that slower and lower-cost memory systems can be used at any given level of system performance.

This handbook provides Am29000-memory-system design information and specific examples that will be helpful in determining how to design a memory system to give you the best cost/performance ratio available to fit your Am29000 application.

Chapters 1, 2 and 3 review:

- performance of the Am29000 32-bit CMOS microprocessor;
- memory-system architectures, key factors and trade-offs; implementation details;
- important memory-design assumptions and introduction to common notations and conventions.

Chapters 4, 5, 6, and 7 provide detailed memory-design examples:

- high-speed static RAM;
- medium-speed SRAM;
- static-column DRAM;
- video DRAM.

Chapter 8 provides a comparison of features and performance for each example using consistent ground rules.

Chapter 9 provides simulated performance information for different memory speeds and interfaces using the Dhrystone 1.1 benchmark.

Appendix A covers memory-array loading-delay calculations using transmission-line and RLC-circuit analysis.

Appendix B discusses the constraints on a single-cycle memory system with tips on how to build one.

TABLE OF CONTENTS

Chapter 1 Overview 1-1

Chapter 2 Basic Issues For All Am29000 Memory Designs 2-1

Architecture 2-1
Key Memory System Factors Defined 2-1
Trade-offs 2-4
Memory Implementation Issues 2-7
Address-Space and Address-Block Decoding 2-7
System Access to Instruction RAM Memories 2-7
Memory Control Signals and Protocol 2-10
Address and Control Driver Issues 2-16
Speed Limit 2-16
Bank Interleaving 2-19
Speed Emphasis 2-19
Test Hardware Interface 2-20

Chapter 3 Assumptions 3-1

Memory Design Assumptions 3-1
Programmable Array Logic (PAL®) Notations 3-1
Abbreviations and Acronyms 3-2
Notational Conventions 3-2

Chapter 4 High Speed Static Ram 4-1

Overview 4-1
Instruction Memory 4-2
Interface Logic Block Diagram 4-2
Memory Interface Logic Equations 4-5
PAL Definition Files 4-10
Intra-Cycle Timing 4-14
Inter-Cycle Timing 4-15
Parts List 4-19
Data Memory 4-19

Chapter 5 Medium Speed Static Ram With Interleaved Banks 5-1

Overview 5-1
What Is Interleaved Memory 5-1
A Basic Two Bank Design 5-1
Instruction Memory 5-1
Interface Logic Block Diagram 5-2
Memory Interface Logic Equations 5-6
PAL Definition Files 5-16
Intra-Cycle Timing 5-20
Inter-Cycle Timing 5-21
Parts List 5-29
Data Memory 5-29

Chapter 6 Static Column DRAM With Interleaved Banks 6-1

Overview ... 6-1
DRAM Advantages and 29000 DRAM Support ... 6-1
Memory Structure ... 6-1
Instruction Memory ... 6-2
Interface Logic Block Diagram ... 6-2
Memory Interface Logic Equations ... 6-6
PAL Definition Files ... 6-19
Intra-Cycle Timing ... 6-32
Inter-Cycle Timing ... 6-34
Parts List ... 6-35
Data Memory ... 6-35

Chapter 7 Video Dram With Interleaved Banks 7-1

Overview ... 7-1
Video Dram Advantages ... 7-1
Memory Features ... 7-2
Interface Logic Block Diagram ... 7-3
Memory Interface Logic Equations ... 7-6
Pal Definition Files ... 7-18
Intra-Cycle Timing ... 7-24
Inter-Cycle Timing ... 7-30
Parts List ... 7-30

Chapter 8 Memory Example Comparisons 8-1

Memory Block Address Range ... 8-1
Memory Board Space Consumption ... 8-1
Memory Power Consumption ... 8-3
Memory Cost ... 8-4
Memory Access Speed ... 8-4
System Benchmark Performance ... 8-7

Chapter 9 Am29000 Dhrystone 1.1 Memory Benchmarks 9-1

Appendix A Memory Array Loading Delay Calculations A-1

Memory Array Models ... A-1
Determining Memory Load Factors ... A-2
Layout Effects ... A-5
Layout Models ... A-7
Transmission Lines or RLC Circuits? ... A-7
Transmission Line Model ... A-7
Memory Specific Example ... A-9
RLC Model ... A-15
Memory Specific Example ... A-16
Design Example Delay Values ... A-17
Non-Interleaved SRAM Example ... A-17
Bank-Interleaved SRAM Example ... A-18
SCDRAM Example ... A-18
VDRAM Example ... A-18
Damping Resistors ... A-19
Program Listings ... A-20

Appendix B Building a Single-Cycle Memory System B-1

Up Against the Wall ... B-1
The Side Effects, Nothing To Spare ... B-2
System Clock Provided by Processor ... B-2
System Clock Provided by External Oscillator ... B-4
Timing is Everything ... B-5

CHAPTER 1

Overview 1-1

Chapter 1

OVERVIEW

The Am29000 Streamlined Instruction Processor is the first in a new generation of CMOS 32-bit high-performance microprocessors built by Advanced Micro Devices. Based on Reduced Instruction Set Computer (RISC) architecture principles, it provides the following features:

- the ability to execute one instruction virtually every clock cycle;
- a streamlined set of instructions, generally less complex than those of prior-generation processors so that each instruction can complete execution in one clock cycle, while still providing support for all the basic and most frequently needed algorithm steps. These simpler instructions serve to break complex algorithms down into a series of simple steps that are then exposed to powerful optimization techniques embodied in the latest generation of language compilers;
- large on-chip instruction cache and register set, so that accesses to external system memory can be reduced such that the system can take advantage of the fast access speed available with on-chip registers and cache.
- load-store method of access to external resources that separates internal (register-to-register) instructions and memory-I/O (register-to-external) instructions into activities that can often be executed in parallel;
- independent instruction and data buses that provide support for concurrent and continuous accesses of external instruction and data memory, so that instruction memory can feed the processor's voracious appetite for a new instruction execution in each cycle while the data-memory bus still provides access to data operands.

Through the use of the above RISC techniques and the latest in advanced high-speed CMOS technology, the Am29000 is able to sustain performance of 20 to 25 Million Instructions Per Second (MIPS), with a peak of 30 MIPS, when clocked at 30 MHz. This is roughly equivalent to between 19 and 24 times the performance of a VAX 11/780[1].

To sustain the above level of performance, the memory system must be able to supply the microprocessor at a rate of almost one instruction every clock cycle. This instruction-per-cycle rate combined with the fast cycle time of the Am29000 makes the memory-system architecture a critical element in supporting the overall system performance. Indeed, to maintain performance above 20 MIPS with the Am29000 requires very high-speed memories or caches.

However, it is equally important to understand that the Am29000 can also achieve very good performance in the 10 to 17 MIPS range when used in conjunction with inexpensive static-column DRAM or video DRAM, at clock rates from 16 MHz to 25 MHz. DRAM systems have a far lower cost per word than static RAM or caches and, when lower speed versions of the Am29000 are also used, the system cost can be further decreased. Yet in this kind of lower-cost design, the system performance still far exceeds that of comparably priced prior-generation microprocessors, and even that of many current-generation RISC microprocessors.

The Am29000 offers a single hardware platform and an extensive set of software tools for use in a wide spectrum of cost-effective, high-performance systems. It, thus, provides a high performance-to-cost ratio and a clear upgrade path to the best possible performance without requiring a change in processor architecture or software.

Because the Am29000 is designed to minimize internal execution-pipeline latency while allowing the memory system as much latency as possible, slower and lower-cost memory systems can be used without a crippling loss of system performance. In exchange for access latency, the Am29000 demands high information throughput via burst-mode memory access. The memory system is expected to sustain a burst-access rate of one access per cycle, but the memory is permitted to have some initial access latency to begin the burst access. As a result, low-speed memory systems can use techniques like pipelining and bank-interleaving to sustain the burst-access rate required by the Am29000. In addition, burst-mode access is intrinsically supported by modern dynamic-memory devices that have the property of high-speed sequential access after a slower initial random-access time. Examples of these memory devices are: DRAM with page mode, nibble mode, static-column mode, or video (serial output) capability.

The allowance for initial latency is provided via a number of Am29000 features:

- For instruction accesses, the Am29000 contains an on-chip Branch Target Cache (BTC) that provides up to three cycles for the memory to begin supplying a sequential burst of instructions without incurring a performance penalty.
- For data accesses, the Am29000 can overlap memory loads and stores with instruction execution. So, memory latency occurs in parallel with continued instruction execution. The programmer or compiler can schedule a memory access in advance of when the data is required.
- Once data is read from the memory, it is forwarded directly to the execution stage for use in the next cycle. This, again, minimizes the internal pipeline latency to allow additional access time in the memory.
- The large register file (192 registers) of the Am29000 acts as an on-chip stack cache to help reduce the number of off-chip data accesses.
- The on-chip Memory Management Unit (MMU) minimizes pipeline latency by making translated addresses available to the memory early in the cycle following execute. Additionally, the MMU simplifies the memory design by performing the address-translation task on-chip.
- Finally, the Am29000 uses separate non-multiplexed data and instruction buses to simplify the memory interface and maximize the information transfer rate.

This handbook shows how to use the Am29000 in a non-cache memory environment with standard currently available memory devices. Examples of four specific memory systems are shown, each of which is capable of sustaining single-cycle burst access for an Am29000 operating at 25 MHz.

The memory implementations are:

- High-speed static RAM;
- Medium-speed static RAM with interleaved banks;
- Static-column DRAM with interleaved banks;
- Video DRAM with interleaved banks.

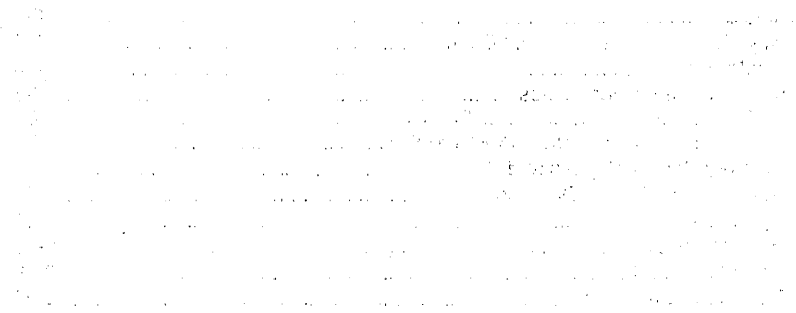
Each implementation explains the trade-offs in system memory size, cost, and the latency associated with initial access. Additionally, the performance of each implementation is simulated and described. Block diagrams, timing diagrams, state-machine diagrams, PAL equations, and component lists are included.

NOTES:

1. The 20 MIPS of sustained performance is based on a system using two Am29062 Integrated Cache Units, one each on the instruction and data buses. These cache units have an initial access time of two cycles (1 wait state) and single-cycle burst-access time (zero wait-state burst mode). Benchmark programs run on this model include: Dhrystone V2.0, grep, diff, and nroff, all of which meet or exceed the sustained-performance quote of 20 MIPS. The 25 MIPS sustained-performance quote is based on using separate static RAMs for instructions and data, able to support single-cycle (zero wait state) access in both initial and burst modes. Most competitive RISC microprocessors claim sustained performance assuming single-cycle (zero wait state) memory or cache units, although some only state peak performance, which for the Am29000 is equal to the 30 MHz clock rate, available since June '88.
2. **Warning:** These are paper designs; they have not been implemented in hardware. The designs are, therefore, subject to the usual number of oversights, mistakes, and outright blunders that lie hidden in the depths of any complex and untried plan. However, the static-column-DRAM and video-DRAM designs have been functionally simulated on an Apollo workstation with Mentor CAD software. Behavioral models for memories, PALs, SSI and MSI logic, and the Am29000 were provided by Logic Automation. Therefore, to the best of our test vectors, we believe the static-column-DRAM and video-DRAM designs work correctly.

CHAPTER 2
Basic Issues For All Am29000 Memory Designs

Architecture 2-1
Key Memory System Factors Defined 2-1
Trade-offs 2-4
Memory Implementation Issues 2-7
Address-Space and Address-Block Decoding 2-7
System Access to Instruction RAM Memories 2-7
Memory Control Signals and Protocol 2-10
Address and Control Driver Issues 2-16
Speed Limit 2-16
Bank Interleaving 2-19
Speed Emphasis 2-19
Test Hardware Interface 2-20



Chapter 2

BASIC ISSUES FOR ALL Am29000 MEMORY DESIGNS

ARCHITECTURE

How you organize a memory system for the Am29000 is driven by a number of factors, and getting the most out of one feature requires trade-offs in other factors. The following discussion will give you guidelines for what you need to be concerned about in a memory system. Additionally, this chapter will show you where you can make some reasonable compromises in the design to get the best of most worlds.

Key Memory System Factors Defined

Access Speed — The whole point of using the Am29000 is to get a three to five times improvement in performance over the "other guy's solution". Memory access speed is the key element in determining the performance of an Am29000 system. But, there are two separate measures of access speed. The balance between them allows the Am29000 a wide range of performance-to-cost trade-offs.

One speed issue is how fast can you get to any random word of memory; this is *initial* access time. The other main issue is how fast can subsequent sequential words of memory be accessed; this is *burst* access time.

Initial access time is different from burst access time because:

- When a new address is supplied by the processor, all bus devices must decode the address to determine whether or not to respond. So, an initial access requires some time to decode the address and begin the access of a memory word. But, a burst access is always to the next word in sequence after either an initial access or a previous burst access. Therefore, the burst access does not require any address decode time; the memory block already knows it is selected and only needs to increment the address from the last access. Further, the memory block does not need any special logic, i.e., added delay, to deal with the possibility of a burst access crossing memory chip or block boundaries because the Am29000 processor will always supply a new address at every 256-word address boundary.

- In the case of a memory block that recognizes its address, the selected word of memory must be accessed. Some memory devices, like DRAMs, require more time to access a random word of memory than to access a sequential word. This is generally due to the upper (row) and lower (column) half of the memory address being time multiplexed to the DRAM. Therefore, a random-word access requires both a row address and a column address to be provided. A burst access needs only a new column address, or in some memories, only a signal to shift out the next sequential word. Thus, access to a random location (new row and column address) takes longer than access to a sequential word.

- Also, when a new row is accessed, DRAM memories require delay time between the end of a previous access and the beginning of the new row access. This time is in addition to the delay time associated with transferring the new row address. This added delay is called *precharge* time. Therefore, when a random access immediately follows a previous access to the same memory, the new initial access incurs the precharge time delay.

- In a bank-interleaved memory system, the first access in a series gains no benefit from the overlapping of access time between memory banks since all the banks must go through a full bank access time before the first (initial) word is available. Therefore, the initial access is always longer than subsequent burst accesses in an interleaved memory architecture. This is covered in more detail later.

Generally, an initial access is slower than a burst access due to the address decode, row-address entry, initial bank access and precharge delays which may be required for an initial access but do not apply to a burst access.

Memory Size — In a dedicated controller application, a few kilobytes of code and data space may be all you need. If so, the speed and simplicity of memory can be maximized by using Static RAMs (SRAM). But, if you need a few megabytes to handle an engineering workstation task, board space, power, and cost considerations will usually drive out SRAMs in favor of DRAMs. With DRAMs, system speed usually drops a little and complexity goes up a little.

Board Space — For a given memory size, the required board area for the memory varies widely depending on memory density, which is technology related. SRAMs provide speed and design simplicity but, they are far less dense than DRAMs and consume a good deal of board space. DRAMs pack the needed memory size into the smallest board space at the cost of initial access speed and design complexity.

Power — Memory speed usually implies high power consumption! SRAMs are generally used for speed and to get large memory size you use a lot of SRAMs. The result is that for a given memory size, SRAMs consume much more power than DRAMs.

Cost — Money always matters! Building your entire 8-Mbyte system memory out of 20-ns SRAM is generally out of the question unless you've just won the lottery. So, cost will generally impact the size, speed, and structure of memory.

Memory Structure — Cost, power, and board-space considerations favor DRAM memory. Speed, and simplicity considerations favor SRAM. Besides the two extremes of using only SRAM or only DRAM, there is also the option of a multi-bank interleave access structure. Bank-interleave schemes allow slower memories to achieve the same performance as a single bank of higher speed memory during the critical burst access mode. In the case of SRAM, it means less costly memories can still provide maximum burst performance. For DRAMs, it means that these slower memories can still give maximum burst performance. Where maximum speed is required along with large size, a compromise structure can be used with a little SRAM and a lot of DRAM. That option is called cache memory and, due to its complexity, is best handled as a topic of its own in a separate discussion.

Complexity — The simplest memory system probably consists of one bank of ROM for instructions and one bank of SRAM for data, with each bank capable only of simple accesses. That way there is virtually no control logic, no address decode logic, no buffers, and no refresh problems to deal with. Of course that structure may not provide enough speed, flexibility, or memory size. The other end of the complexity spectrum would involve something like dual- or quad-interleave DRAM banks with burst access ability. There you get to deal with refresh issues, bank sequencing, address counters, and dual porting of the instruction bank for both instruction and data accesses. The complexity buys memory size, lower power, and burst access speed at the cost of additional control logic and buffering.

Throughput — The Am29000 is a synchronous machine. The timing of all its actions is in relationship to its clock. Information flow to or from the microprocessor must occur in units of time that are integer multiples of the system clock cycle. That means that if the access time of the memory does not fit into a single clock cycle then two cycles will be taken. Even if the access time only misses by a few nanoseconds, a whole cycle of time is lost. Depending on how often that situation comes up, it can be a better deal to slow the system clock down by a few nanoseconds so that most of the memory accesses can occur in a single cycle. Thus the overall throughput of the system can be significantly improved in some cases by *slowing the system down.* Sometimes the the option of slowing down the memory to match a slightly slower system clock can result in significant savings in cost and complexity. The only way to know for sure is to simulate different speed memory configurations with the Am29000 architectural simulator software known as the SIM29K.

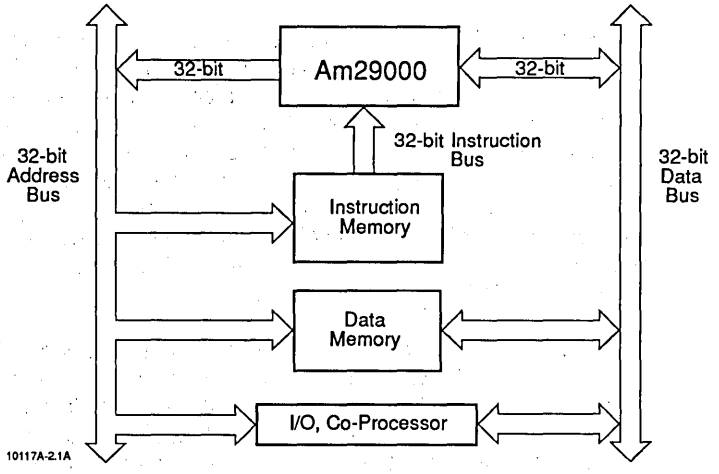
Bus Structure — The Am29000 has three separate buses:

- Address Bus, which is shared between instruction, data, I/O, and co-processor accesses;
- Instruction Bus, which is used to move instructions from the system memory to the processor;
- Data Bus, which is used to move data between the processor, system memory, I/O devices, and co-processors via load and store operations.

Together, these buses and their related control lines are referred to as the channel. This channel allows for concurrent access of instructions and data when the instruction and/or data memories are accessed via pipeline or burst requests. As shown in Figure 2-1, this structure strongly favors memory systems that have separate memory blocks for holding instruction and data so as to allow simultaneous access.

With regard to the Am29000, the data bus is bidirectional, the address bus is "output only", and the instruction bus is "input only". So, by definition the processor cannot write information to the instruction bus. Therefore when separate data and instruction memory blocks are used with the Am29000, the system design must provide a way to load the instruction memory since the processor cannot directly write information into the instruction memory via the instruction bus. This issue is covered in more detail later.

Figure 2-1



Three-Bus Architecture

Trade-offs

What's the best memory architecture? Well, it all depends on your goals for the system as a whole.

- If you are building an *embedded controller* like a network node processor, digital signal processor, or a mainframe-computer I/O processor, the main requirement is system speed. If the memory requirement is small, up to a megabyte or so, then high-speed SRAM works very well.

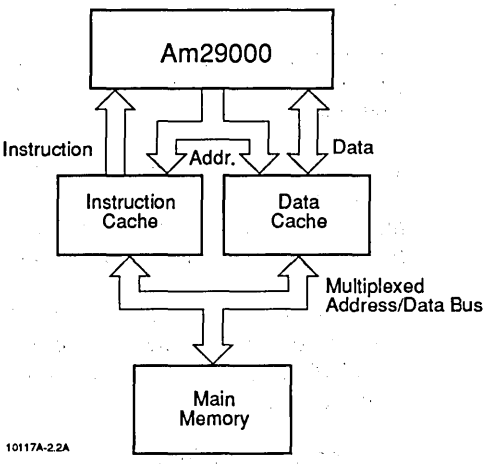
  For small memory systems the cost, power consumption, and board space of SRAM is reasonable and the speed will be the best possible. Initial access time will be one-to-three cycles and burst access speed will be single cycle in a 25 MHz clock-rate system. Average sustained performance will be in the 16 to 18 MIPS range. Peak performance can reach 25 MIPS with any memory system, but its the sustainable performance that counts.

  Note: performance estimates throughout this document are based on the use of a 25-MHz system clock frequency.

- If you are building a *mainframe computer* or high-performance *engineering workstation,* then system speed and large memory capacity are important. Here, a cache memory architecture, such as the one shown in Figure 2-2, provides the best possible performance with access to a large main memory.

  The cache could be built from SRAMs or with the Am29062 Integrated Cache Unit, a single-chip cache controller with an 8K-byte internal cache memory. The main memory can be built from relatively slow and inexpensive DRAMs that provide a main memory as large as needed. The cache memory supports a two-cycle initial access and single-cycle burst access. Performance would again be in the 16-to-18 MIPS range.

Figure 2-2



- Use external I-Cache and D-Cache for maximum performance with:
  - Unlimited memory size
  - Common instruction and data space

Cache Memory for Instructions and Data

- If *system performance* and *memory size* are important, but less important than system *cost* and *complexity*, there is another architecture with cache-like performance but at far less cost and complexity. That is a design using Static Column DRAM (SCDRAM).

  A SCDRAM memory design using interleaved memory banks has an initial row access time of four to six cycles with single-cycle burst accesses. But SCDRAMs also provide a very important caching function. The static column capability of the SCDRAM means that once a given row is addressed for the first time, all subsequent accesses within that row can be made by simply changing the column address. Those accesses within the row may be to any random address and do not incur the timing overhead of multiplexed row and column addresses. Random access within the row can occur in three cycles. Subsequent burst accesses are single cycle.

  In effect, the SCDRAM has a built in "cache" with one row of words in it. The time to do a complete "cache" reload is the initial row access time of four to six cycles.

  This "cache" is put to best use when memory accesses tend to be sequential and localized. When the accesses are sequential the burst mode of access gives excellent performance. Even when the accesses are not sequential, as long as the accesses remain local to one row of the memory the initial access time is held down to three cycles, which is nearly what would be achieved with fast SRAM. Certainly the above access characteristics are typical for instruction memory. Also, many programs have data access patterns that would also benefit from the improved access speed within rows.

  In a dual-bank interleaved SCDRAM memory using sixty-four 1Mbit by 1-bit SCDRAMs, the "cache" size is 2K words (8K bytes) resulting from the two banks of memory each with a 1K-bit row "cache" in each memory. The total memory size is 2M words (8M bytes) resulting from the two 1Mbit by 32-bit memory banks.

The performance of this system would be in the *14 to 16 MIPS range*, which is amazing system performance while using a relatively simple architecture and low-cost memories.

The Am29000's internal Branch Target Cache (BTC), burst-access bus protocols, large register file, independent instruction and data buses, and overlapped load and store operations are all key features that allow the Am29000 to give premium performance with low-cost DRAM memories.

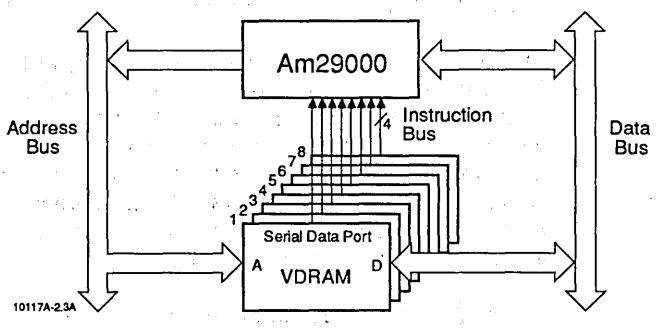
- For a *simpler, lower-cost, medium-speed* application, a Video DRAM (VDRAM) memory architecture may be appropriate. VDRAM does not have quite the same "caching" ability of the SCDRAM but it does provide dual porting of a large common memory array.

  One port of the VDRAM is a serial shift register that holds one row of bits from the internal DRAM array. A by-4 organization memory has four shifters. This row is shifted out providing consecutive memory words. Just what the instruction bus of the Am29000 needs! The other VDRAM port is a bidirectional random-access bus that allows read or write operations on any word of the internal DRAM array. That is just what the data bus of the Am29000 needs!

  The two ports are controlled by a common address input of the VDRAM. As shown in Figure 2-3, that matches nicely with the common address of the Am29000. Once the shifter port is loaded with a row of data, the shifter operation is independent of the internal DRAM array and the random I/O port. This allows simultaneous access to both instructions and data by the Am29000.

  So, the VDRAM allows a single bank of fairly dense memory to serve both the instruction and data buses of the Am29000 in a very simple and efficient manner. The trade-off here is in speed. The initial access time for a VDRAM is four to seven cycles. Its burst access speed for instructions can still be single cycle with a 25 MHz shift rate on the serial port. Its burst access speed on the random I/O port is limited by the speed of page-mode access which requires cycling of a column address strobe; thus data-burst accesses are three to four cycles each. This could be improved by bank interleaving the design.

Figure 2-3



- VDRAMs:
  - Allow simultaneous instruction fetch and data access from a common internal memory array.

VDRAM Main Memory

Even with this slower access time the system performance is still in the 10-to-12 MIPS range. Considering the simplicity and low cost of the design, that is a very respectable performance.

So, whatever the system requirement, the Am29000 has the flexibility to support a wide range of cost-performance trade-offs. And, at whatever cost level, the Am29000 will be at the top of the performance scale against any other monolithic CMOS 32-bit processor.

Just as a point of reference, both the Motorola 68020 and Intel 80386 are at their maximum performance (and cost) of about 5 MIPS when using SRAM cache-memory systems. The Am29000 runs three-to-five times faster in a similar system and, even with the simple VDRAM system described above, the Am29000 is double the performance.

If you think MIPS is not an "apples-to-apples" measure of performance, you're right, so go look at Chapters 8 and 9 on benchmark performance. The Am29000 still beats the competition by three-to-five times on equivalent benchmark programs!

MEMORY IMPLEMENTATION ISSUES

Once you get past the big decision of what the overall architecture will be, you come upon the details. This section discusses several implementation details that are common to nearly all the memory architectures discussed. Thus, each memory design will have to cope with the issues discussed in the following paragraphs.

Address-Space and Address-Block Decoding

The Am29000 distinguishes between multiple address spaces for any given address value. So, in most designs, an instruction/data memory should not respond to instruction ROM address space and vice versa. Similarly, data memories should not respond to I/O or coprocessor address space.

Also, there may be multiple blocks of physical memory in any one address space. Therefore, most memory interfaces will include some degree of block address decoding.

System Access to Instruction RAM Memories

As noted earlier, the Am29000 makes *best* use of memory systems that contain separate instruction and data memories for simultaneous access to instructions and data. In a memory system with separate instruction and data-memory blocks, the data-memory block is straightforward. The memory-data I/O pins are simply connected to the Am29000 data bus. All reading and writing of the data memory is done via the data bus. Access to the data memory can thus be by either the processor or any other bus master.

In the case of the instruction memory block there is an added twist. With respect to the Am29000, the instruction bus is used only for instruction input (fetching) by the processor. The processor thus cannot drive the instruction bus. Therefore the instruction memory cannot be directly loaded (written) with information by the processor via the instruction bus in a manner analogous to the way data memory is loaded via the data bus.

Why is the instruction bus only used for input by the processor?

- In virtually all systems the instruction memory spends the vast majority of its time being read each cycle to fetch instructions for the processor. Very little of the instruction memory bandwidth is needed to load the instruction memory with new instruction information. In fact, many types of systems only need to load the instruction memory during the power-up sequence. And, some store instructions in PROM so the processor never writes instruction words.

- Not putting output drivers on the instruction bus saves silicon area for more valuable functions and simplifies certain electrical design issues for the processor.

- There are other ways for the system design to provide more efficient means to load and perform diagnostics on the instruction memory.

Here are some of the ways to provide system access to the instruction memory:

- The instruction memory may have some additional buffering and control logic so that the memory can read information onto either the instruction or data bus. Also, the data input of the instruction memory would be connected to the Am29000 data bus. This configuration would allow the instruction memory to be both read and written via the data bus by either the Am29000 or another bus master.

- A DMA controller with access to both the instruction and data buses could be used to request the channel from the processor and then access the instruction memory via the instruction bus, in which case, the instruction memory block would be exactly like the data-memory block. The system restriction would be that the Direct Memory Access (DMA) controller would be the only means of writing information into the instruction memory.

- Dual-port memory such as a VDRAM could be used to build the instruction memory. One port, the video shifter port, of the memory would provide read access for the instruction bus and the other port would provide read and write access via the data bus.

  This scheme has an additional benefit: the VDRAMs simplify the whole memory structure. Since the two ports share access to the same internal memory array, there need be no internal distinction between instruction and data information. The VDRAMs can thus be used to serve as *both instruction and data memory* within a single device. As shown in Figure 2-3, VDRAMs thereby support both the simultaneous access of instruction and data from a common memory array, and a data-bus access path to instruction memory.

Simple Dual-Bus-Port Instruction Memory

The first method above would implement a simple dual-port access scheme for the instruction memory via buffers and arbitration logic. The arbitration logic is needed because this multi-port structure for an instruction memory creates a problem for the memory interface logic. That is, whenever instruction and data accesses are addressed to the same block of instruction RAM, the data accesses will contend with instruction accesses. The memory interface logic must, therefore, arbitrate access to the memory.

This situation can occur when either the 29000 processor or a DMA device in the system accesses the instruction RAM via the data bus. In each case, the interface logic is faced with a slightly different set of conditions as outlined below.

- If the 29000 processor is performing the data access, there can be a conflict with the processor's own instruction fetching activity. In this case, the data access is the result of instruction execution and in order for program execution to continue the data access must eventually complete. The data access request can occur during a burst-instruction fetch or an instruction fetch can occur during the data access if the data access is a burst request. If at the time the data access starts, the processor is in the middle of an instruction burst access, it is necessary to preempt the instruction access in order to complete the data access. If an instruction fetch begins during a data burst request, the instruction fetch must be held off until the data access is completed.

- In the case of a DMA device access, the processor will release the bus to the control of the DMA device so it is not possible for the processor to start an instruction fetch during burst-data accesses. But, it is still possible that the DMA access will begin during an already established (but suspended) instruction-burst request. Here again, the memory must be able to preempt the instruction-burst request and proceed with the data access.

Instruction Bus DMA

The second method outlined above requires hardware outside of the memory system. All access to the instruction memory is done for the processor by a Direct Memory Access (DMA) controller, specifically one that can access both the instruction and data buses. A DMA controller with this capability can request the processor to give up all the buses (address, data, and instruction) so that the controller has complete access to all memory and I/O devices.

Once the controller owns the buses, there is no rule that prevents it from both reading and writing information in the instruction memory via the instruction bus. The processor lacks this capability because it was never designed to drive the instruction bus. But, as long as the instruction memory can handle it, there is no problem with a DMA controller doing it. By having access to both the instruction and data buses, the DMA controller can transfer information between I/O devices, instruction memory, data memory, and ROM memory.

In fact, if it can be assumed that the DMA controller will move all the information to and from the instruction memory (including the performance of memory diagnostics), there is no reason for the instruction memory to have a second port for access to the data bus. In this case, the control logic and buffering of the instruction memory can be very simple, in fact, identical to that of the data memory.

True Dual-Port Instruction Memory

True dual-port memory used by the third approach noted above, provides not only dual-bus access but also includes built-in structures that allow simultaneous access to the memory array from both the instruction and data buses. VDRAM is one very elegant and economical means to provide this type of memory. There are of course other true dual-port memories or dual-access memory controllers.

Memory Control Signals and Protocol

The Pipeline Enable Signal

A casual review of the Am29000 bus control lines will show that there are separate but equivalent Request and Response control line sets for instruction and data accesses. The exception to this rule is the Pipeline Enable ($\overline{PEN}$) signal. This response signal must be shared between all instruction and data accesses. Therefore it is important to note that the only device that should drive the $\overline{PEN}$ signal, in a given cycle, is a device being selected by a valid address on the address bus (selected during a primary access). The $\overline{PEN}$ signal should be tied high (or low) only when *all* bus devices will (or will not) handle pipelined accesses.

Request and Burst Acknowledge Signals

When a sequence of consecutive instruction or data words needs to be accessed by the Am29000, a burst access is requested via the Instruction Burst Request ($\overline{IBREQ}$) or Data Burst Request ($\overline{DBREQ}$) signals. The initial address of this burst access is announced by the respective Instruction Request ($\overline{IREQ}$) or Data Request ($\overline{DREQ}$) signal going active. While either $\overline{IREQ}$ or $\overline{DREQ}$ is active, the address bus has a valid address for the access.

The burst request is accepted and a burst transfer is established when the addressed memory responds with the Instruction Burst Acknowledge ($\overline{IBACK}$) or Data Burst Acknowledge ($\overline{DBACK}$). In the cycle following the assertion of the Burst Acknowledge signal, the Am29000 will de-assert the ($\overline{IREQ}$ or $\overline{DREQ}$) signals and remove the initial address of the burst access. This frees the address bus for use in other bus accesses.

The point being emphasized here is that a Burst Acknowledge signal is the cause for a Request signal and its associated address to go invalid immediately after a burst transfer is established.

This distinction is important to understand when implementing a burst memory. It is a common error for a memory designer to assume that the initial access $\overline{IREQ}$ or $\overline{DREQ}$ signal and the initial burst address will remain active and valid until the first Instruction Ready ($\overline{IRDY}$) or Data Ready ($\overline{DRDY}$) response is given by the memory.

A key example that points out the importance of having the correct understanding is the following situation: a burst access is suspended or ended by the processor. Note the memory has no way to tell the difference between suspension or completion of a burst access. A new burst access of the same type (instruction or data) as the previous one is started by the processor.

In this situation, the memory is waiting for a resumption of the first burst access. While waiting, the memory holds either the $\overline{IBACK}$ or $\overline{DBACK}$ signal active. Therefore, when the new burst access begins, the memory Burst Acknowledge signal ($\overline{IBACK}$ or $\overline{DBACK}$) will be active during the initial address cycle of the new burst access. This establishes the new burst access and the processor will remove its Request signal ($\overline{IREQ}$ or $\overline{DREQ}$) and initial address from the bus in the following cycle. That means that the Request signal and address are valid for only one cycle. The memory thus must be able to capture the new address and initiate a new burst access sequence at the end of the first (and only) cycle in which the new burst access appears on the processor bus.

In this situation, the memory control logic does not have any way of making the processor hold the new address and control information valid for more than the first cycle of the new burst access. The memory control logic must be designed to "switch gears" in less than a cycle. The logic must go from "waiting for a burst access to resume" to "starting a new burst access" in one cycle.

As noted above it is a common error for a memory designer to think the memory could use the lack of a Ready response to hold off the beginning of the new burst access for a cycle or two so that the memory control logic would have time to get its state machine turned around. *Wrong!*

Of course, one way to avoid the above problem is to make the $\overline{\text{IBACK}}$ or $\overline{\text{DBACK}}$ signals combinatorial and dependent on the inactive state of the Memory Request signals during the burst phase of a memory access. This causes the $\overline{\text{IBACK}}$ or $\overline{\text{DBACK}}$ signal to go inactive during the first cycle of a new access when the related Memory Request ($\overline{\text{IREQ}}$ or $\overline{\text{DREQ}}$) goes active. This, in turn, holds the address on the bus longer and may eliminate the need for address registers.

Although, in general, for better overall system performance, each memory system should be designed to capture a new address in the minimum time possible so that the address bus can be released for use in a another access.

Burst Preemption — The Last Word

A burst access is preempted by de-asserting the $\overline{\text{IBACK}}$ or $\overline{\text{DBACK}}$ signal. If the related burst request signal ($\overline{\text{IBREQ}}$ or $\overline{\text{DBREQ}}$) was active in the cycle before Burst Acknowledge ($\overline{\text{IBACK}}$ or $\overline{\text{DBACK}}$) was de-asserted, one last word of information must be transferred before the burst access is ended. That word can be transferred in the same cycle that burst acknowledge is de-asserted or in some later cycle but, until it is transferred the burst access is not complete and no new access of the memory may begin.

Burst Access Reactivation

When a burst access is suspended ($\overline{\text{IBREQ}}$ or $\overline{\text{DBREQ}}$ made inactive by the processor) and the access later resumed, it is a requirement of the bus protocol that Memory Ready signal ($\overline{\text{IRDY}}$ or $\overline{\text{DRDY}}$) may not be active in the same cycle that Burst Request ($\overline{\text{IBREQ}}$ or $\overline{\text{DBREQ}}$) is first reasserted. Therefore memory interfaces must de-assert the Ready signal when a burst access is suspended.

Memory Response Control Signals

The $\overline{\text{IRDY}}$ and $\overline{\text{DRDY}}$, the Instruction Error ($\overline{\text{IERR}}$) and Data Error ($\overline{\text{DERR}}$), the $\overline{\text{PEN}}$, and the $\overline{\text{IBACK}}$ and $\overline{\text{DBACK}}$ signals from the memory interface to the processor are critical indicators that must be in a valid state at the end of each clock cycle. In systems with multiple memory control interfaces, each interface must be able to drive these response control signals. Only one memory interface can actively drive these signals in each clock cycle. As different memory interfaces are addressed by the processor, the control over these signals must pass from interface to interface. This transfer of control must be accomplished within a single cycle to ensure that the lines are valid on each cycle.

At a 25 MHz cycle rate, it is nearly impossible to implement the transfer of control by selectively driving the control lines via 3-state buffers as is commonly done in slower memory systems. Wire ORing with open-collector drivers is also impractical.

The solution is to logically OR the respective control lines from each memory interface via an SSI logic gate such as a NOR or AND gate. Where there are several memory interfaces to be logically ORed, a PAL such as the AmPAL16L8 may be used in the place of SSI logic gates.

Write Enable of Memories

For memories that are able to perform data-write operations in a single clock cycle, e.g., CMOS static RAMs, the Write Enable ($\overline{WE}$) signal to these memories must be a pulse that occurs during the latter half of the write cycle. The Am29000 has a data hold time of 4 to 20 ns after the rising edge of System Clock (SYSCLK). If the memory being used has a non-zero data-input hold time relative to the active edge of $\overline{WE}$, then that edge must occur early enough for the Am29000 to satisfy the memory-data-input hold time.

For most single-cycle memories, this situation implies that SYSCLK is a convenient signal to use as a $\overline{WE}$ qualifying signal to ensure that $\overline{WE}$ ends at the rising edge of SYSCLK. The delay of the final write-enable logic gate can then be masked by the propagation delay of a buffer on the data lines so that the $\overline{WE}$ signal, at the memory, ends at or before the time data goes invalid.

Byte and Half-Word Accesses

The Am29000 implements full-word read and write operations on word-address boundaries directly in hardware. Access to a specific byte within a word is provided by instructions for byte extract or insert operations on internal registers. Similarly, access to a half-word located on a half-word address boundary is done via half-word extract or insert instructions. These instructions can be used to manipulate a byte or half-word of interest, with actual memory access occurring via full-word loads and stores.

Word and half-word accesses that are not aligned on respective word or half-word address boundaries can be accomplished via software trap routines executed when a non-aligned access is attempted.

This software approach to byte, half-word, and unaligned accesses provides a general-purpose mechanism for manipulating external byte and half-word quantities, without the requirement for special support hardware. In most cases, this approach produces an overall performance gain by allowing a shorter system cycle time. The shorter cycle time results from the elimination of any requirement for masking, alignment and control hardware in the critial memory-access path.

In cases where it is desired to improve the performance of byte and half-word access via external alignment and control logic, the Am29000 provides a means of controlling the external hardware. Three of the code values on the Option (OPT)0-2 lines are set aside by convention to indicate word, half-word, and byte accesses. These codes can control the alignment and masking of data on load operations and the selection of byte $\overline{WE}$ signals during store operations (the encoding of OPT bits is shown in the Am 29000 Users Manual, Chapter 3).

The decision to add external hardware should be carefully considered to insure that the performance advantage for the byte and half-word accesses justifies the hardware and performance costs.

Compared to the basic processor mechanism for byte and half-word accesses described above, external hardware can reduce the time for byte and half-word loads by zero to 12 or more cycles. In the case of a simple (address boundary aligned) byte or half-word load, there could be zero cycles saved if the added delay of the external hardware increases the memory access path delay to the point that a memory wait state must be added. In the case of an unaligned access, the software approach using a trap routine could incur 12 or more cycles of overhead in the trap execution.

The improvement for byte and half-word stores is more significant, since external hardware can eliminate the extra load (for a load-modify-store sequence) required by the basic processor mechanism.

So, to determine performance and cost effects of external byte and half-word support hardware, the system designer must weigh the cost against the following performance factors for software-vs-hardware approaches:

- Percentage of simple byte and half-word accesses
- Percentage of unaligned accesses
- Performance penalty of hardware in added wait-states multiplied by the number of affected accesses; or performance penalty of hardware in added system cycle time multiplied by the number of cycles executed
- Performance penalty of software overhead from byte and half-word insert and extract instructions or overhead in trap routine execution, multiplied by the number of accesses

If external hardware is used in combination with the OPT0-2 lines, it is very important that the already defined code conventions be followed. Failure to do so will make the non-standard system implementation incompatible with every compiler known to be under development for use with the Am29000. All Am29000 compilers can generate the already defined OPT0-2 codes for use in byte and half-word accesses.

The Late-Late Show Signals

Three memory control signals from the Am29000 arrive rather late in each clock cycle and require some special handling. The signals are $\overline{\text{IBREQ}}$, $\overline{\text{DBREQ}}$ and BusInvalid (BINV).

The first two will, in the worst case, be valid 14 ns after the falling edge of SYSCLK. The falling edge of SYSCLK is defined as occuring at 1/2T ns ± 1 ns into the clock cycle, where T is the total clock-cycle length. That means, in a 40 ns clock cycle, the falling edge of SYSCLK, at worst, occurs 21 ns into the cycle. Therefore $\overline{\text{IBREQ}}$ and $\overline{\text{DBREQ}}$ are valid by 35 ns into the cycle. That leaves a thin 5 ns worth of set-up time for any logic that needs to use those signals. Any good design engineer can subtract another nanosecond or so to account for some clock skew in the system wiring. So that leaves a mere 4 ns of set-up time. So, the most you can hope to do with these signals is to capture their state in a very fast register.

The timing for $\overline{BINV}$ signal is a bit more leisurely. The $\overline{BINV}$ signal is valid 7 ns after the falling edge of SYSCLK, which puts it at 28 ns into the clock cycle. That leaves 12 to 11 ns for set-up time. This is a little better but still, in most cases, this signal is also simply registered and used in the following cycle.

Bus Invalid!? Now What?

First, a little discussion on just what the $\overline{BINV}$ signal is all about.

- $\overline{BINV}$ is involved in the transfer of channel ownership. It goes active during the cycle when the Am29000 releases control of all buses and control lines to another channel master that has requested the channel. It also goes active during the cycle that the Am29000 retakes control over the channel being returned to the processor by another channel master.

  During the cycles that $\overline{BINV}$ is active in this situation, all the channel lines are in a state of transition. One channel master is putting its drivers into a high-impedance state and the other has yet to begin actively driving the channel. Therefore there is no guarantee as to what the logic levels on the channel might be and all control lines and bus lines should simply be ignored while $\overline{BINV}$ is active.

- $\overline{BINV}$ is also used in several situations where the processor has made a Request signal and an address active on the channel but, late in the cycle, the processor recognizes that the Request is incorrect or not necessary.

  In these situations the meaning of $\overline{BINV}$ is only defined as applying to the access being started. Any burst or pipelined access, already in progress, in the unaffected portion of the channel is considered able to continue during the $\overline{BINV}$ cycle.

  One such situation is when a Memory Management Unit (MMU)-translated address is placed on the address bus to begin a new access and the processor recognizes that the address is actually invalid due to a protection violation in the Translation Look-Aside Buffer. The new address is effectively cancelled by $\overline{BINV}$ going active late in the cycle.

  Another situation involves the cancelling of an access because the processor identifies it as no longer needed. This can occur when a jump instruction is immediately followed by another jump. The second jump instruction eliminates the need for any instruction that would have followed the first jump. This recognition causes the processor to cancel the memory access for instructions following the first jump via $\overline{BINV}$ going active.

  Again, in these situations $\overline{BINV}$ is only defined to disrupt the access being started in the cycle that it is active. An access on the alternate bus continues even though $\overline{BINV}$ is active.

Although there are these situations in which an active $\overline{BINV}$ applies to only part of the channel activity, it is recommended that $\overline{BINV}$ always be used to ignore any bus control or data signal during the cycle $\overline{BINV}$ is active.

From the viewpoint of a memory system it is difficult to separate the channel ownership transfer situation from the other situations in which the $\overline{BINV}$ signal goes active. Thus it requires significant extra logic to properly ignore only some signal activity on the channel when $\overline{BINV}$ is active.

The logic to properly do this must monitor the $\overline{BREQ}$, Bus Grant ($\overline{BGRT}$), $\overline{IREQ}$, $\overline{DREQ}$, and $\overline{BINV}$ signals. The logic would follow a sequence like that below.

When $\overline{BGRT}$ first goes active, it indicates a transfer of channel ownership from the processor to another channel master. The first contiguous set of $\overline{BINV}$ active cycles to follow $\overline{BGRT}$ going active identifies a period when all channel signals should be ignored. When $\overline{BINV}$ goes inactive at the end of the channel-transfer sequence, there begins a period during which any further assertions of the $\overline{BINV}$ signal indicates that only the access request being initiated with $\overline{BINV}$ asserted needs to be ignored. The above period ends when $\overline{BREQ}$ first goes inactive, which indicates the return of control over the channel back to the processor. The first contiguous set of $\overline{BINV}$ active cycles to follow $\overline{BREQ}$ going inactive identifies another period during which all channel signals should be ignored. Following this period, any future assertions of $\overline{BINV}$ apply only to the request being started in conjuction with $\overline{BINV}$ going active, until $\overline{BGRT}$ again goes active to start the above cycle over again.

All the above just gets more complicated if there is more than one other channel master in the system which could pass control of the channel on to yet another channel master without first returning control to the processor. In this case $\overline{BINV}$ recognition logic would have to keep track of all channel master $\overline{BREQ}$ and $\overline{BGRT}$ lines.

Now, for all that effort, the savings would be one extra cycle of information transfer on an unaffected bus for each cycle $\overline{BINV}$ is asserted, if the unaffected bus is in fact ready to transfer information during the cycle. This savings would occur less than 0.01% of the time.

Therefore it is best to simply define $\overline{BINV}$ as a signal that defines an idle cycle for the entire channel. Design the memory system so that no action (change of state) occurs as a result of any signal on the channel when $\overline{BINV}$ is active.

Memory Error Signals

The Am29000 has error inputs ($\overline{IERR}$, $\overline{DERR}$) for both instruction and data bus accesses. These signals are only monitored by the Am29000 when an instruction or data access is pending. Therefore, it is required that if an error condition such as a parity error is to be reported, the appropriate error signal must be driven active at or before the time when the memory Ready ($\overline{IRDY}$, $\overline{DRDY}$) signals would normally go active. In some cases this may require that the access time of the memory be increased to allow time for error-detection logic to check the validity of data.

An alternative to requiring memory error signals to be valid with or before memory ready signals would be to use the $\overline{WARN}$, $\overline{TRAP0}$, $\overline{TRAP1}$, or $\overline{INTR0}$-$\overline{INTR3}$ signals in a subsequent cycle to abort the affected process. Another alternative to extending the memory cycle time, to allow time for Error Detection or Correction (EDC), is to add a pipeline stage to the memory access path. This would provide an entire cycle time to perform an EDC function, while increasing only the initial access time by one cycle. Subsequent burst accesses could continue to be single cycle.

Invalid Address Situation

If no valid bus device is addressed by a bus-access attempt, no ready response will ever be provided. This would cause a bus master to hang-up forever waiting for some response. It is therefore advisable to have some kind of timeout mechanism for bus accesses. If an invalid address is accessed by mistake the timeout mechanism can end the access with an error response.

Address and Control Driver Issues

In the high speed memory designs for the Am29000 the emphasis is on using the slowest memory possible while still achieving the necessary speed. This means that control logic and signal drivers must be the fastest available. That means that D-speed (10 ns) PALs are recommened for control logic devices and that these devices directly drive address lines and control lines of the memories.

Directly driving the memories eliminates the added delay of separate buffers often used to drive memory-array signals. But, PAL devices generally have worst-case delay times specified when driving only 50 pF load capacitance. Often a memory array will have 32 or more memory devices, each with an input capacitance of 5 pF to 10 pF. In addition, typical strip-line PC board traces will add an additional 20 pF of capacitance and 100 to 200 nH of inductance per foot of trace length. Such a memory array can easily represent an inductive and capacitive load with 180 pF to ≥ 340 pF of capacitance and ≥ 100 nH of inductance. It is therefore required that the worst-case delay times for the affected PAL outputs be increased to account for the added load.

Appendix A provides an analysis of how to determine the appropriate added delay value.

Speed Limit

It can be useful to determine and analyze the limiting factors for memory speed. For any memory architecture, there are three signal paths with critical timing:

- The address-to-data valid path during a read access.
- The address to end of write path during a write access.
- The channel master control signal active to response signal active path during any access.

There are also two access cycles of interest: the initial access and the burst access. For this analysis the channel master of interest is the Am29000.

Address-to-Data Valid Path

For the address-to-data valid path in an initial access cycle, the memory system is subject to the following key parameters :

- Clock-to-Processor Address, Data and Control Signals Valid,
- Address Control Logic Delay,
- Memory Access Time,

- Data Bus Buffer Delay,
- And Data Set-Up Time.

In a burst-access cycle, the same parameters are used except that the clock-to-address and control signals valid delay and the address and control logic delay are replaced by the clock-to-output delay of the memory address counter.

Clock-to-Processor Address and Control Signals Valid — during the first access to a non-sequential location in memory, the processor must provide a new address and instruction or data-request control signals to indicate a new memory request is being made. This parameter is currently 14 ns for the Am29000.

Address/Control Logic Delay — some memory designs will need to select between the initial address and the output of an address counter used for burst access cycles. The logic to select the address will add some delay. If D-speed PALs are used for this logic, the delay will be 10 ns (assuming only a 50 pF load on the PAL output).

Memory Access Time — this is one factor the memory designer has some control over. The speed limit of the memory system is reached when this delay goes to zero.

Data Bus Buffer Delay — generally a buffer is used to isolate the memory-array outputs from the processor data bus. The propagation delay through the buffer must be considered. One of the fastest buffers available is a 74FCT244A with 4.3 ns propagation delay.

Data Set-Up Time — the Am29000 data input set-up time is 6 ns.

Thus the address-to-data path for an initial access is at best 34.3 ns when the memory access time is zero. This then implies that most memory implementations will have an initial access time of at least two cycles.

In a burst-access cycle the speed limit is set by the clock-to-output time of the address counter (8 ns for a D-speed PAL), data-buffer delay, and the processor set-up time. They total 18.3 ns leaving 21.7 ns for memory access time in a 40 ns cycle time system. Therefore burst accesses can be single cycle with the use of fast SRAMs. Bank interleaved memory can achieve single-cycle burst access even with much slower memory.

Address-to-End of Write Path

For the address-to-end of write path in an initial access cycle the following are key parameters that the memory system is subject to:

- Clock-to-Processor Address, Data and Control Signals Valid;
- Address/Control Logic Delay, in parallel with Data Bus Buffer Delay;
- Memory Address and Data Set-Up Time to Write Enable Active.

In a burst-access cycle, the same parameters are used except that the clock-to-address and control-signals-valid delay and the address and control logic delay are replaced by the clock-to-output delay of the memory address counter. That means the clock-to-data-valid delay may predominate.

Clock-to-Processor Address, Data and Control Signals Valid — during the first access to a non-sequential location in memory, the processor must provide a new address and data request control signals to indicate a new memory request is being made. This parameter is currently 14 ns for the Am29000.

Address/Control Logic Delay — some memory designs will need to select between the initial address and the output of an address counter used for burst access cycles. The logic to select the address will add some delay. If D speed PALs are used for this logic, the delay will be 10 ns (assuming only a 50 pF load on the PAL output).

Data Bus Buffer Delay — generally a buffer is used to isolate the memory-array outputs from the processor data bus. The propagation delay through the buffer must be considered. The fastest buffer available is a 74FCT244A with 4.3 ns propagation delay. During an initial access this delay is masked by the address/contol logic delay. During the burst access this delay adds to the data valid delay.

Memory Address and Data Set-Up Time to Write Enable Active — this is one factor the memory designer has some control over. The speed limit of the memory system is reached when this delay goes to zero.

Thus the address-to-end of write path for an initial access is at best 24 ns when the memory set-up time is zero. This then implies that a write access may be completed within one cycle if the real memory set-up time can be held below 16 ns.

In a burst-access cycle the speed limit is set by the clock-to-data valid delay plus the data bus buffer delay. They total 18.3 ns leaving 21.7 ns for memory set-up time in a 40 ns cycle time system. Therefore, burst accesses can also be single cycle.

Control to Response Path

For the control signal to response signal path the time restrictions are the same in all access cycles. The key parameters are:

- Clock-to-output time of a register;
- Propagation delay of a PAL;
- Propagation delay of a logical OR gate on the response signals from each memory block;
- And control signal set-up time of the processor.

The clock-to-output delays internal to a D-speed PAL are worst-case 8 ns.

The propagation delay of a D-speed PAL is 10 ns.

The propagation delay of the memory response signal OR gate can range from 6 to 10 ns.

The set-up time for control signals to the Am29000 is 12 ns.

All those times total to 40 ns. This makes single-cycle operation possible in a 40 ns cycle-time system.

Exceeding the Limit

It is possible to build specially restricted memories that do not need the address/control logic delay or the data bus buffer delay. This is done by having only a single bank of memory for instructions or data. There is, then, no need for address decode or bus isolation. Such a memory could have single-cycle initial access by using a 13 ns access-time memory. In this type of memory, the worst-case path delay involves the Chip Enable ($\overline{CE}$) signal on memory, which is controlled by the system clock. Using the clock to control the $\overline{CE}$ signal eliminates bus contention between the processor and memory and possible false $\overline{WE}$ signals. The worst-case delay of the clock is 21 ns and the processor set-up time adds an additional 6 ns of delay. That leaves 13 ns for the memory in a 40 ns cycle-time system.

Refer to the description and diagrams in appendix B for more details regarding specially restricted single-cycle-access memory designs.

Bank Interleaving

Memories with 20 ns or faster access times are neither easy to find nor inexpensive to buy. Based on the above timing discussions it is easy to see that it would be very desirable to find a way to use slower memories.

A simple way to reduce the memory-access speed requirement by half or more is to make use of a bank-interleave memory architecture. In bank interleaving, one set of memories contains the even words in memory and another set contains the odd words of memory. The two banks are accessed on alternate clock cycles so that each bank is allowed two cycles of access time. The banks alternately supply data words so that there is one new data word available in each bus cycle. This scheme of course relies on sequential word accesses which is exactly the nature of a burst access by the Am29000. This scheme can be further extended to three, four, five, etc. bank-memory systems in order to further lengthen the allowable memory access time. The penalty is extended initial access time and the complexity of the control logic. Only the initial access requires the full delay of a two-cycle access.

Speed Emphasis

In the discussion of memories, a careful separation of the initial access and burst access times has been made. This is important to help make the trade-off of memory-access speed and initial access time clear. Single-cycle burst-access speed can be maintained even with rather slow memories given that the initial access speed can suffer. Where burst accesses are the predominant mode of memory access and where the bursts are relatively long, the initial access time can be amortized across many accesses. In this case, slow interleaved memory is ideal. But, the more often a non-sequential access is done, the more the initial access time lowers the overall memory system performance.

Instruction accesses are always attempted in burst mode. Statistically "average" instruction streams branch every six to ten instructions. Therefore the initial access time of instruction fetches can be amortized across six to ten cycles of access. Burst access speed is thus important to instruction accesses.

Further, the branch target cache can hide up to three cycles of an instruction memory's initial access time when the target of the branch is in the cache. The hit rate of the branch target cache is application dependent of course but typical hit ratios of 50 to 60% are common in benchmarks that have been run. Thus the importance of burst access time, over initial access time, is further emphasized.

Data accesses are different because most are individual load or store operations. They are more often done as individual non-sequential reads or writes of single words. Burst accesses are done usually only at context switch time and during some procedure entries and exits. This means that over 95% of data accesses are to non-sequential locations. Therefore, the initial access time is a much more important factor for data memories than for instruction memories. Consequently, it is best to emphasize burst access speed in instruction memories and initial access speed in data memories.

Test Hardware Interface

Memory designs must account for the special needs of diagnostics hardware. The key issue here is that development systems will, at times, want to take control of buses and control lines in a system under test. In particular, to perform reads and writes of Am29000 internal registers, a development system may want to masquerade as a system memory device during a diagnostic load or store operation. Doing this allows a development system to directly observe and control register values.

When this is done, the memories in a prototype system need to recognize when the development system takes control of system buses so that the memories will not contend with the development system for control of the buses.

One method for doing this is described. It is the method used by Advanced Micro Devices in its own Am29000 hardware and software development support system, the Advanced Development And Prototyping Tool (ADAPT29K).

The ADAPT29K operates as a system monitor and controller that allows logic-analyzer-like tracking of the Am29000 system activity. It also is able to insert diagnostic instructions into the normal Am29000 instruction stream, read and write processor registers, and read and write system memory.

The ADAPT29K is inserted into a system via the Am29000 socket. An adapter fits into the Am29000 socket and an Am29000 is then plugged into the top of the adapter. This allows the ADAPT29K access to all the signal pins of the Am29000.

At various times the ADAPT29K will drive the following lines: DATA 0-31, INSTRUCTION 0-31, $\overline{\text{RESET}}$, $\overline{\text{DRDY}}$, $\overline{\text{DERR}}$, STAT1, CNTL0, and CNTL1.

The ADAPT29K system must somehow indicate when it will take control of the above lines from the the system under test. Two means of indicating this are provided: use of *pin 169 on the Am29000 socket* and the use of a *special code* on the OPT bits 0-2.

Pin 169 is the device-locator pin that allows chip insertion in only the correct orientation and is the only pin not used by the Am29000. This pin can, therefore, be driven by the ADAPT29K as an indication to the system being debugged that the ADAPT29K is taking control of some of the Am29000 signal lines.

The prototype system under development can simply use the signal on pin 169 as a disable of the selection logic for all system memories. This will ensure that when pin 169 is driven, the ADAPT29K system will be free to take control of the prototype system buses.

This plan is simple but not without problems. Pin 169 may not always be available in future package types for the Am29000. Also, it is an "extra" signal not normally planned

for in the system. Its advantage is that it is a simple, direct, and "pre-decoded" indication that the ADAPT29K is taking control. Its disadvantage is that it is not a consistent and intrinsic part of an Am29000 system. It requires that the system under test be modified to expect this special signal that will only come from specific development hardware.

Recognizing the limitations of pin 169, the ADAPT29K system provides another way to signal its use of system buses.

The ADAPT29K defines one of the reserved codes for the OPT0-OPT2 bits as the equivalent of the pin 169 signal. During a load or store operation, the OPT0-OPT2 bits displaying "110" is defined to mean that the ADAPT29K will control the Instruction bus, Data bus, Ready, and Error lines; even though the address presented would appear to be directed at some other system device (note, OPT0 is the Least Significant Bit (LSB) corresponding to the zero in the "110" code). The ADAPT29K system uses this definition when reading or writing an AM29000 internal register. To do this, a load or store instruction is used with the OPT0-OPT2 bits set to "110". When the load or store is executed, the OPT0-OPT2 code appears on the bus and is used to cause the system memory to not respond while the development system directly moves data to or from the Am29000.

This scheme has the advantage of not requiring any "special" signal connections between the prototype and development systems. All communication is via the standard Am29000 socket. Also, it may be possible to make use of decoding circuits already present for the OPT0-OPT2 bits to decode the needed signal equivalent to the pin 169 indication, thus saving on special-purpose hardware.

The ADAPT29K uses both the pin 169 and OPT0-OPT2 signals, so that allowing the designer of the prototype system can choose which way to support intervention by the development system.

In the case of a read or write of Am29000 registers, the ADAPT29K jams a load or store test instruction with OPT 0-2 bits set to "110" and pin 169 low. At the appropriate moment, the $\overline{\text{DRDY}}$ and $\overline{\text{DERR}}$ pins are driven by the ADAPT29K. It is necessary that memory not respond or drive the instruction or data lines during this operation. It is also required that the $\overline{\text{DRDY}}$ and $\overline{\text{DERR}}$ lines be either open collector or 3-stated by the prototype system when pin 169 is low or the OPT0-OPT2 bits = "110".

In the case of a read or write of memory, the ADAPT29K jams a load or store test instruction with the OPT 0-2 bits set to 000. Pin 169 is driven high when the Am29000 is single stepped. In this case the memory should respond normally when pin 169 is high. Note: This implies that the ADAPT29K requires the ability to read and write the instruction memory via the data bus!

CHAPTER 3
Assumptions

Memory Design Assumptions 3-1
Programmable Array Logic (PAL®) Notations 3-1
Abbreviations and Acronyms 3-2
Notational Conventions 3-2

ASSUMPTIONS

MEMORY DESIGN ASSUMPTIONS

In each of the memory design examples presented in Chapters 4 through 7, the following assumptions were made:

- All designs are intended to operate in a 40 ns clock-cycle system (25 Mhz clock frequency).
- The Am29000 Synchronous Input Setup Time (data sheet parameter 9A) is 6 ns as shown in May '88 data sheet, rather than 8 ns as reflected in February '87 data sheet. Similarly, the Am29000 Synchronous Input Setup Time (data sheet parameter 9) is 12 ns.
- Any other system bus master observes the same bus protocol as the Am29000 processor. Examples: new addresses are provided for each 1K byte boundary crossing; read and write operations may not be mixed within a burst transaction.
- Each memory monitors pin 169 of the Am29000 socket for interface with the Am29000 Advanced Development And Prototyping Tool (ADAPT29K).
- Memories do not drive memory response lines or data lines when not also driving memory Ready or Error signals. This ensures that the memories do not contend with test hardware during diagnostic operations.
- Memories implement only word-write operations.Implementing byte-write control logic is a simple extension to the designs presented here. Byte-write logic will (in those ever famous words) be left as an exercise for the reader.

PROGRAMMABLE ARRAY LOGIC (PAL) NOTATIONS

Depending on the nature of the output signal being described, there are two basic types of PAL-related equations used in this handbook: registered and combinatorial.

The registered equation is for a PAL circuit whose output signal is a function of the inputs that must pass through a register. Thus, the output signal is dependent on a clock (transfer) signal. A registered equation is identified by the special operator ":=". For example:

$$X := A \cdot B + C$$

The combinatorial equation , on the other hand, is for a PAL circuit with an output signal based on only its input signals. That is, the output signal is a propagation-time-delayed function of the inputs without any intervening state elements . A combinatorial equation is identified by operator "=". For example:

$$Z = Q \cdot X + Y$$

ABBREVIATIONS AND ACRONYMS

Abbreviation and acronym definitions are provided on a first-occurrence basis in the text.

NOTATIONAL CONVENTIONS

Chapters 4 through 7 use the notational conventions included in the the following paragraphs.

Boolean Notations

The Boolean equations use the conventional Boolean symbols for identifying logic connectives such as AND and OR. By way of review, the logic connectives for Boolean symbols are:

• = AND

+ = OR

The complement of a variable used in a Boolean equation is represented by an overbar above the variable. For example:

- The complement of X is $\overline{X}$. The complement of a variable is also referred to as the "negation" or "not" operation.

- Double overbar is used over a variable when a complemented variable is nested in brackets and the bracketed expression is also complemented. For example:

$$XX = A \bullet B \bullet \overline{\overline{(C + D)}}$$

CHAPTER 4
High Speed Static Ram

Overview 4-1
Instruction Memory 4-2
Interface Logic Block Diagram 4-2
Memory Interface Logic Equations 4-5
PAL Definition Files 4-10
Intra-Cycle Timing 4-14
Inter-Cycle Timing 4-15
Parts List 4-19
Data Memory 4-19

HIGH SPEED STATIC RAM

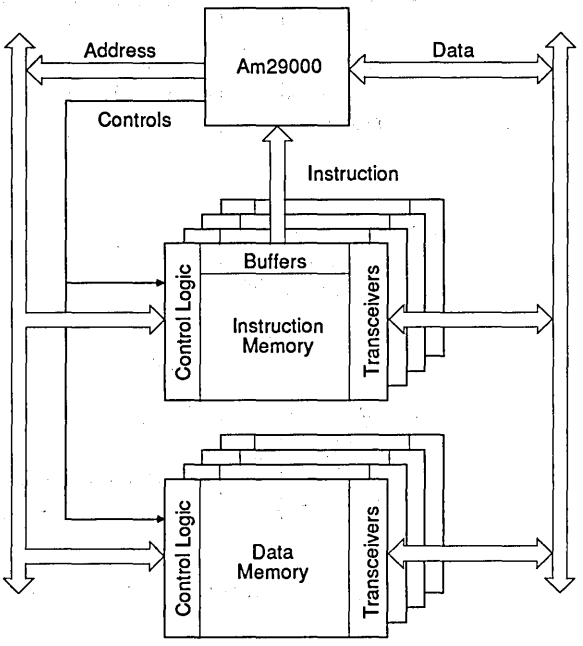
OVERVIEW

Let's start off our memory design examples with the simple "brute force" approach to the architecture shown in Figure 4-1.

We will use one block of Static RAM (SRAM) for instruction memory and one block of SRAM for data memory. The block will contain high speed SRAM that is fast enough to support accessing one word per clock cycle during burst transfers. Each block is 16K words deep and each word is 32 bits wide. The instruction memory block will have a read only port for sending instructions to the Am29000 and a read/write port tied to the Am29000 data bus. The read/write port allows access to the instruction memory via the data bus for instruction loading and memory diagnostics. The data memory will have a single read/write port connection to the Am29000 data bus.

The "brute force" description applied to this architecture refers mainly to the very high speed required of the memories and interface control logic. The memories will need to access data in 20 ns or less and the control logic must be made from Programmable Array Logic (PAL) devices with propagation delays of only 10 ns. At this time, those components are rather expensive and power hungry. But, making use of this raw speed allows the interface logic and overall structure of the memory to be very simple while providing very close to the best achievable memory system access time.

Figure 4-1



Am29000 Memory Interface Overview

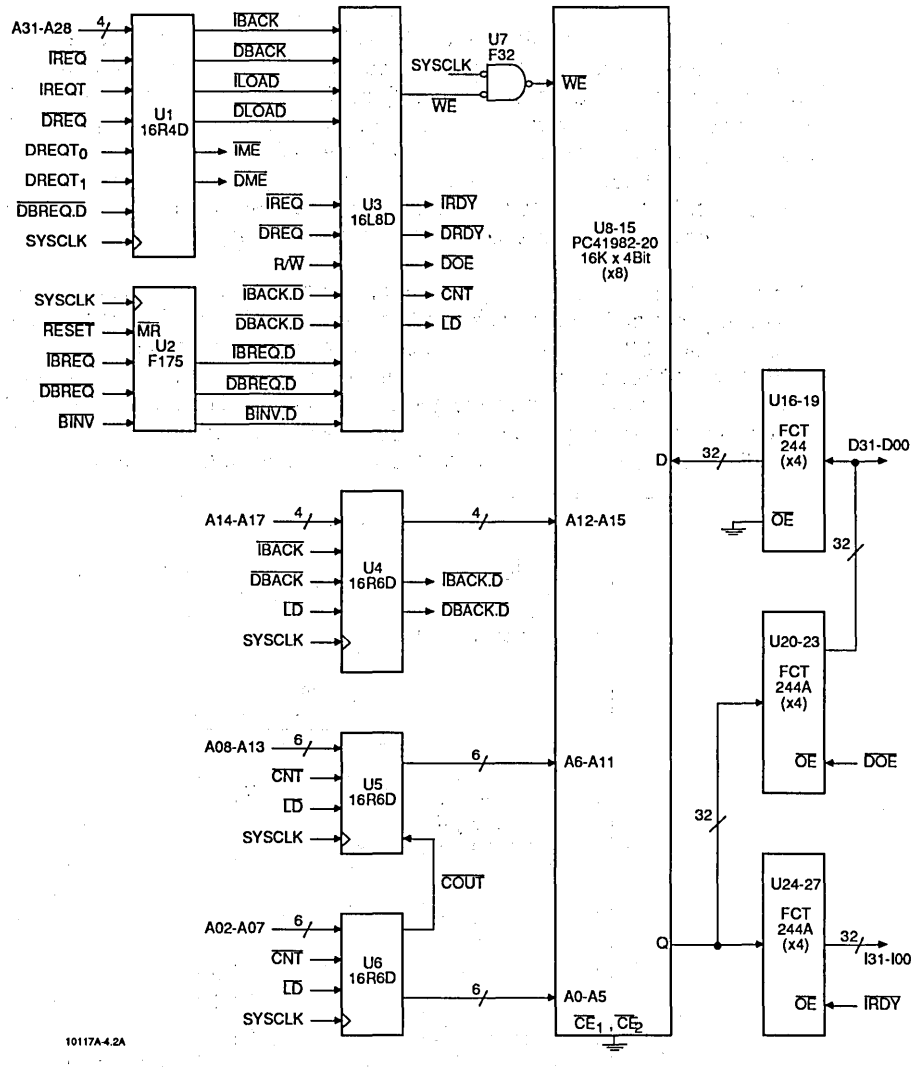
The initial access time will be two clock cycles: one cycle for decode and one for access. For burst accesses, each access beyond the initial access will occur in a single clock cycle.

INSTRUCTION MEMORY

Interface Logic Block Diagram

Refer to the block diagram in Figure 4-2.

Figure 4-2



Interface Logic Block Diagram

Memory

The center of the memory block is of course the memory itself. The memories are 16K x 4-bit SRAMs with separate data in and out lines. The access time is 20 ns and eight devices are required to form the 32-bit wide instruction word for the Am29000.

Bus Buffers

The memory data outputs are connected to the data-bus lines via high speed buffers (U20–U23). These buffers are required to isolate the memory outputs from the data bus whenever the memory is accessing instruction words. This isolation allows another data memory block to use the data lines at the same time that instructions are being fetched from this memory block.

The memory data inputs are also connected to the data bus lines via buffers (U16–U19). These buffers provide delay time to the data lines during write cycles which helps to ensure that data is still valid at the time Write Enable ($\overline{WE}$) goes inactive at the end of each write cycle. As will be shown later the $\overline{WE}$ signal goes inactive one gate delay later than the end of each cycle. Also, note that if this block of memory were made up of multiple banks of memory devices instead of the single bank used in this design, then these buffers might be needed to isolate the heavier capacitive load of multiple memory banks from the data bus lines.

It is worth noting that the memory data I/O connection to the data bus could also be achieved through the use of bidirectional buffers, but doing so would require very careful management of the buffer and memory output enable signals to prevent driver contention. Using separate unidirectional buffers keeps the design simple and robust.

The memory data outputs are also connected to the instruction bus lines via buffers (U24–U27). These buffers serve to isolate the data outputs of this memory blocks from those outputs of other memory blocks which may also drive the instruction bus. Also the buffers would serve to isolate the capacitive load of this memory block from the instruction bus if the block contained a larger number of memory banks.

Address Registers and Counters

To support burst accesses the lower eight address bits to the memory come from a loadable counter. The 8-bit counter is built from two AmPAL16R6 D-speed PALs (U5, U6). The D-speed PALs are used because their clock-to-output delay is significantly less than standard MSI 8-bit counters. Also, the use of PALs allows additional functions to be integrated into the same packages used for the counter function.

The upper eight bits of memory address need not come from a counter since the Am29000 will always output a new address when a 256-word boundary is crossed. The upper eight bits of address are simply registered. The register is built from remaining functions in one of the AmPAL16R6D PALs that form the lower 8-bit counters (U5) and from part of an additional AmPAL16R6D PAL (U4).

Registered Control Signals

As noted earlier, the timing of the $\overline{IBREQ}$, $\overline{DBERQ}$, and $\overline{BINV}$ control signals require that they be registered by a low setup-time register. A 74F175 register is used for this. Also two other signals, $\overline{IBACK}$ and $\overline{DBACK}$, are also registered. Remaining registers in the third AmPAL16R6D PAL (U4) are used for this purpose.

Interface Control Logic

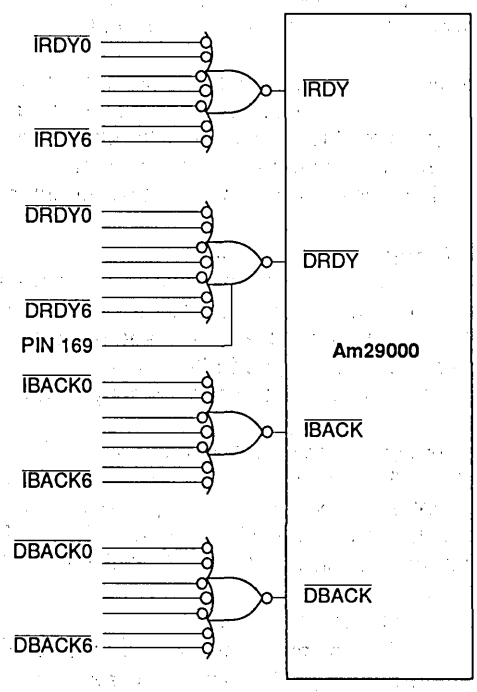
This logic must generate the memory response signals, manage the loading and counting of memory addresses, and control the $\overline{WE}$ signal to the memory. The logic functions needed for this require two D-speed PALs, an AmPAL16R4 and an AmPAL16L8 (U1, U3). Also, the final level of gating on memory $\overline{WE}$ and the memory response lines is shown in Figures 4-2 and 4-3.

In Figure 4-2, the $\overline{WE}$ line of the memory is driven from a 74F32 OR gate which combines the $\overline{WE}$ signal from the Interface Control logic with the SYSCLK signal. The simple OR gate is used to ensure minimum propagation delay so that the memory $\overline{WE}$ signal will go inactive as soon as possible after the rising edge of SYSCLK.

In Figure 4-3, the memory response lines from multiple memory blocks are logically ORed together before being presented to the Am29000. The lines are ORed in AmPAL16L8 D-speed PALs. Each PAL can implement two of the seven input negative logic OR gates that are shown. These final gates are required by the high speed nature of these signals as was explained in Chapter 2, Basic Memory Design Issues. Also note that if the $\overline{IERR}$, $\overline{DERR}$ or $\overline{PEN}$ signals were implemented by this design, those signals would require similar gating to that shown in Figure 4-3.

Again referring to Figure 4-3, note that Pin 169 of the Am29000 is used as an output enable on the $\overline{DRDY}$ OR gate to provide test hardware the ability to take control of this line. This was described in Chapter 2, Test Hardware Interface section.

Figure 4-3



Memory Response Signal OR Gates

Memory Interface Logic Equations

Design Choices

In this memory interface it is assumed that other blocks of instruction or data memory may be added later and that there may be valid addresses in address spaces other than instruction/data space. This means that this memory will only respond with $\overline{IBACK}$ or $\overline{DBACK}$ active when this block has been selected by valid addresses in the instruction/data space. This requires that at least some of the more significant address lines above the address range of this memory block be monitored to determine when this memory block is addressed. Also, it means the IREQT, DREQT0, DREQT1, and Pin 169 lines must be monitored to determine that an address is valid and lies in the instruction/data space.

The support of burst accesses implies the need for a state machine with three states, which will control the transitions between no activity on the burst acknowledge lines and activity on either the $\overline{IBACK}$ or $\overline{DBACK}$ line. This state machine also can ease the management of transitions between instruction and data accesses when preemption is required. The state diagram for this state machine is shown in Figure 4-4.

Another design choice is that when an instruction burst access is in progress and a data access to the same block of instruction memory is attempted, the instruction access will be preempted immediately. The data access will then complete before any further instruction access will be allowed. This approach prevents the processor pipeline from stalling while the instruction prefetch queue fills before instruction access is suspended, as would occur if instruction accesses were given priority.

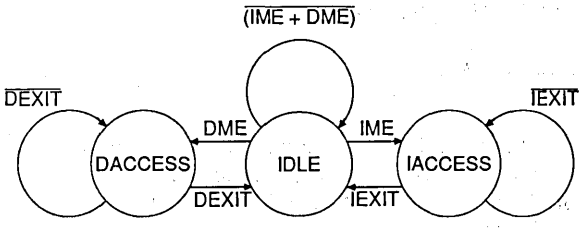
Logic Details — Signal-by-Signal

All signals are described in active-high terms so that the design is a little easier to follow. The signals as implemented in the final PAL outputs will often be active low as required by the actual circuit design. The actual PAL definition files are included in Figures 4-5 through 4-9.

NOTE: All PAL equations in this handbook use the following convention:

- Where a PAL equation uses a colon followed by an equals sign (:=), the equation result is REGISTERED, i.e., registered PAL outputs are used.
- Where a PAL equation uses only an equals sign (=), the equation signals are COMBINATORIAL PAL outputs.

Figure 4-4



Interface Logic State Diagram

IDLE — This is the default state of the interface state machine. It is characterized by Instruction Burst ACKnowledge (IBACK) and the Data Burst ACKnowledge (DBACK) signals both being inactive. This state serves as a way of identifying when the memory is not being accessed and could be placed into a low power mode. It should be noted that the IDLE state is not the sole determiner of when a low power mode can be used. Referring to the explanation of the Chip Enable (CE) signal provides a more complete understanding of low power mode requirements. The more important use of the IDLE state is as a delay cycle in the transition between an active instruction burst access being preempted and the start of the preempting data access. The delay is needed to allow the completion of the final instruction access in the cycle that the IBACK signal is de-asserted and the instruction burst access is pre-empted.

IME — IME is the indication that the address of this memory block is present on the upper address lines, an instruction request is active, Pin 169 is inactive (test hardware has not taken control), and instruction/data address space is indicated. In other words this memory block is receiving a valid instruction access request. This example of a memory system design will assume that the address of this memory block is equal to $\overline{A31} \cdot \overline{A30} \cdot \overline{A29}$. The equation for this signal is:

$$IME = IREQ \cdot \overline{IREQT} \cdot \overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{Pin169}$$

DME — DME is the indication that the address of this memory block is present on the upper address lines, a data request is active, Pin 169 is inactive, and instruction/data address space is indicated. In other words, this memory block is receiving a valid data access request. This example design will assume that the address of this memory block is equal to $A31 \cdot \overline{A30} \cdot \overline{A29}$. Note that for instruction accesses, the memory address for this block is A31 = zero and that for the data accesses, the memory address for this block is A31 = one. This allows instruction memory for instruction accesses to be located at address zero while having the window for data bus access to the instruction memory located at a different base address. This allows the separate data memory block used in this design to have its base address also at zero. Thus both the instruction and data memories are located at address zero in their respective address spaces.

The equation for this signal is:

$$DME = DREQ \cdot \overline{DREQT0} \cdot \overline{DREQT1} \cdot A31 \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{Pin169}$$

IEXIT — Instruction EXIT (IEXIT) is an intermediate equation term not actually implemented as an output of the SRAM State Generator PAL. The logic of the term is used in the generation of IBACK but the name IEXIT is simply a documentation convenience.

The IEXIT equation is:

$$IEXIT = DME + IREQ \cdot \overline{IME}$$

A data request to this memory block for instruction data space will take priority over an instruction fetch in progress. Also, if a new instruction fetch stream is started for either another block of memory or to instruction ROM, this memory interface can return to the IDLE state.

DEXIT — Like IEXIT, Data EXIT (DEXIT) is a term used only for documentation convenience.

The DEXIT equation is:

$$DEXIT = IME \cdot \overline{DBREQ.D} + DREQ \cdot \overline{DME}$$

An instruction request to this memory block for instruction/data space when data burst request was inactive in the last cycle will end any suspended data access. Requiring data burst request to be inactive will hold off instruction fetches until the current data access is complete or suspended. Also, if a new data access stream is started to another block of memory, to I/O space, or to coprocessor space, this memory interface can return to the IDLE state.

IBACK — Instruction Burst ACKnowledge (IBACK) is the indication that the interface state machine is in an active or suspended instruction burst access. The signal is synonymous with the Instruction ACCESS (IACCESS) state in Figure 4-4. The equation is:

$$IBACK := IME \cdot \overline{DBACK} + \overline{IEXIT} \cdot IBACK$$

The IACCESS state is entered when an instruction request to instruction data space with the address of this memory block is active and a data access is not currently active. The $\overline{DBACK}$ term will give an active data access priority by holding off instruction accesses until the data access completes.

Once in the IACCESS state the interface will stay there until one of the IEXIT conditions is satisfied.

DBACK — The Data Burst ACKnowledge (DBACK) is the indication that the interface state machine is in an active or suspended data burst access. The signal is synonymous with the DACCESS state in Figure 4-4. The equation is:

$$DBACK := DME \cdot \overline{IBACK} + \overline{DEXIT} \cdot DBACK$$

The Data ACCESS (DACCESS) state is entered when a data request to instruction/data space with the address of this memory block is active and an instruction access is not currently active. The $\overline{IBACK}$ term will hold off the beginning of a data access until any active instruction access is preempted.

Once in the DACCESS state the interface will stay there until one of the data exit conditions is satisfied.

LD — LoaD (LD) is the signal which enables the lower address bit counter/registers and the upper address bit registers to load a new address on the next rising edge of SYSCLK. The equation is:

$$LD = \overline{DBACK} \cdot IREQ \cdot \overline{ILOAD} + \overline{IBACK} \cdot DREQ \cdot \overline{DLOAD}$$

When an instruction request is active, load is prevented from being active while a data access is active or suspended. In other words, when the state machine is in the DACCESS state a load that would result from an instruction request is suppressed.

Also load is prevented if there was a load in the last cycle. In the case of a burst request this prevents load from being active during the second cycle of a burst request at which time the count signal to the address counters must be active and cause the counters to increment.

Similarly for the case that Data REQuest (DREQ) is active, load is prevented when the state machine is in the IACCESS state or when load was active in the last cycle. The LD signal is combinatorial so that it will be active during the first cycle of a new instruction or data request.

ILOAD — The Instruction LOAD (ILOAD) is a delayed version of the load signal with the qualification that it is active only when a load occurred for an instruction fetch which was addressed to this memory block and the instruction/data space.

$$ILOAD := \overline{DBACK} \cdot IME \cdot \overline{LOAD}$$

ILOAD is used in the generation of the Instruction ReaDY (IRDY) signal.

DLOAD — The Data LOAD (DLOAD) is a delayed version of the load signal with the qualification that it is active only when a load occurred for a data access which was addressed to this memory block and the instruction/data space.

$$DLOAD := \overline{IBACK} \cdot DME \cdot \overline{DLOAD}$$

DLOAD is used in the generation of the Data ReaDy (DRDY) signal.

CNT — The CouNT (CNT) signal causes the address counters to increment on the next rising edge of SYSCLK.

$$CNT = IBREQ.D \cdot \overline{BINV.D} \cdot IBACK + DBREQ.D \cdot DBACK \cdot \overline{BINV.D}$$

CNT is active in the second cycle and beyond of each instruction or data access when the respective burst request was active in the previous cycle. During BINV active cycles no counting is allowed since the Burst Request signals are presumed to be invalid.

IBACK.D — The IBACK Delayed (IBACK.D) is simply a one cycle delayed version of IBACK.

$$IBACK.D := IBACK$$

It is used in the generation of IRDY.

DBACK.D — The DBACK Delayed (DBACK.D) is simply a one cycle delayed version of DBACK.

DBACK.D := DBACK

It is used in the generation of DRDY.

IRDY — Instruction ReaDY (IRDY) indicates that there is valid read data on the instruction bus.

IRDY = ILOAD • $\overline{\text{BINV.D}}$
+ IBREQ.D • $\overline{\text{BINV.D}}$ • IBACK.D

This static memory design will always be ready with data in the cycle after a new instruction request which is implied by ILOAD. But IRDY should never be active if the bus was invalid on the previous cycle when the load of address information occurred. The Bus INValid Delayed (BINV.D) signal must be used to prevent IRDY from going active.

A case that shows the need for this is when control of the bus is transferred between bus masters. When this occurs, the bus is guaranteed to be invalid for at least one cycle. If during the invalid cycle the memory control and address lines were seen as a valid instruction request, then load would go active and ILOAD would be active in the next cycle. This would cause IRDY to be active during the first cycle of the new bus masters first instruction fetch. That would be incorrect since the memory would not have read valid information in time for the first cycle of the instruction fetch. Thus qualification with $\overline{\text{BINV.D}}$ is required.

The memory will also be ready when IBREQ was active with IBACK in the previous cycle. IBACK is required as a qualifier so that when an access is preempted the continued presence of IBREQ will not cause a false ready indication.

Note that $\overline{\text{BINV.D}}$ is again used as a qualifier for the same reasons noted earlier.

The reason that IRDY must be a combinatorial signal is that IBREQ comes very late in the previous cycle and must be registered. There is no time to perform logic on IBREQ in the previous cycle before SYSCLK rises. This means that the information that IBREQ was active in the last cycle is not available until the cycle in which IRDY should go active.

DRDY — Data ReaDY (DRDY) is the equivalent of IRDY for data accesses and therefore uses the same equation with data terms substituted for instruction terms.

DRDY = DLOAD • $\overline{\text{BINV.D}}$
+ DBREQ.D • $\overline{\text{BINV.D}}$ • DBACK.D

DOE — Data Output Enable (DOE) is the same equation as DRDY except that the Read/$\overline{\text{Write}}$ line is added as a qualifier. This prevents the data bus read buffer output enable from going active on a write cycle. Note: the Am29000 Read/$\overline{\text{Write}}$ ($\text{R}/\overline{\text{W}}$) signal has been designated simply as RW in the equation.

DOE = DLOAD • $\overline{\text{BINV.D}}$ • RW
+ DBREQ.D • $\overline{\text{BINV.D}}$ • DBACK.D • RW

WE — Write Enable (WE) has nearly the same equation as for DOE except that it is qualified by the inverse of the read/write line.

$$WE = DLOAD \cdot \overline{BINV.D} \cdot \overline{RW} + DBREQ.D \cdot \overline{BINV.D} \cdot DBACK.D \cdot \overline{RW}$$

In the block diagram of Figure 4-2 you can see that WE is further qualified by SYSCLK. This added qualification will create a pulse that is the result of the overlapped low time of WE and SYSCLK. This means that the pulse is coincident with SYSCLK low time when WE is active.

WE is the result of an 8 ns clock to output delay of PAL registers combined with the propagation delay of a PAL which is 10 ns. The worst-case time is then 18 ns for WE to become valid. The earliest possible occurrence of SYSCLK going low is one half the cycle time plus or minus 1 ns. In this case that is 20 ns – 1 ns = 19 ns. The importance of the timing is that the WE signal must be valid at or before the falling edge of SYSCLK in order to prevent unwanted glitches on the WE line to the memories.

CE — Chip Enable (CE) in this design would only be used to lower the dynamic power of the system by switching off the memories when they are not being accessed. An equation for this would be:

$$CE := \overline{IBACK} \cdot \overline{DBACK} \cdot IME + \overline{IBACK} \cdot \overline{DBACK} \cdot DME + IBACK + DBACK$$

This equation will not allow the memory to go into a deselected or low power mode until the cycle following a transition to the IDLE state. This ensures that the memory is still active on the last access of a preempted instruction burst request.

In this design however there weren't enough outputs on the PALs to add this feature conveniently. So, the CE signal was left out of the design.

Pal Definition Files

The PAL definition equations are provided in Figures 4-5 through 4-9.

NOTE: All PAL equations in this handbook use the following conventions:

- Where a PAL equation uses a colon followed by an equals sign (:=), the equation result is REGISTERED i.e. registered PAL outputs are used.
- Where a PAL equation uses only an equals sign (=), the equation signals are COMBINATORIAL PAL outputs.
- The Device Pin list is shown near the top of each figure as two lines of signal names. The names occur in pin order, numbered from the left to right 1 through 20. The polarity of each name indicates the actual input or output signal polarity. Signals within the equations are shown as active high, e.g., where signal names in the pin list are $\overline{A}$ B $\overline{C}$, the equation is $C = A \cdot \overline{B}$; the inputs are $\overline{A}$ = low, B = low, then the $\overline{C}$ output will be low.

Figure 4-5

AmPAL16R6D SRAM Address Counter—Non-interleaved, Section 0 Device U6

```
CLK CNT LD A02 A03 A04 A05 A06 A07 GND
OE NC12 Q02 Q03 Q04 Q05 Q06 Q07 COUT VCC

Q02   := LD • A02
       + LD • CNT • Q02
       + LD • CNT • Q02

Q03   := LD • A03
       + LD • CNT • Q03
       + LD • CNT • Q02 • Q03
       + LD • CNT • Q02 • Q03

Q04   := LD • A04
       + LD • CNT • Q04
       + LD • CNT • Q02 • Q03 • Q04
       + LD • CNT • Q02 • Q04
       + LD • CNT • Q03 • Q04

Q05   := LD • A05
       + LD • CNT • Q05
       + LD • CNT • Q02 • Q03 • Q04 • Q05
       + LD • CNT • Q02 • Q05
       + LD • CNT • Q03 • Q05
       + LD • CNT • Q04 • Q05

Q06   := LD • A06
       + LD • CNT • Q06
       + LD • CNT • Q02 • Q03 • Q04 • Q05 • Q06
       + LD • CNT • Q02 • Q06
       + LD • CNT • Q03 • Q06
       + LD • CNT • Q04 • Q06
       + LD • CNT • Q05 • Q06

Q07   := LD • A07
       + LD • CNT • Q07
       + LD • CNT • Q02 • Q03 • Q04 • Q05 • Q06 • Q07
       + LD • CNT • Q02 • Q07
       + LD • CNT • Q03 • Q07
       + LD • CNT • Q04 • Q07
       + LD • CNT • Q05 • Q07
       + LD • CNT • Q06 • Q07

COUT = Q02 • Q03 • Q04 • Q05 • Q06 • Q07
```

Figure 4-6

AmPAL16R6D SRAM Address Counter—Non-interleaved, Section 1
Device U5

```
CLK /CNT /LD A08  A09 A10 A11 A12 A13 GND
/OE /CIN /Q08 /Q09 /Q10 /Q11 /Q12 /Q13 NC19 VCC

Q08  := LD * A08
     + /LD * /CNT * Q08
     + /LD * CN T * CIN * /Q08
     + /LD * CNT * /CIN * Q08

Q09  := LD * A09
     + /LD * /CNT * Q09
     + /LD * CNT * CIN * Q08 * /Q09
     + /LD * CNT * /CIN * Q09
     + /LD * CNT * /Q08 * Q09

Q10  := LD * A10
     + /LD * Q10

Q11  := LD * A11
     + /LD * Q11

Q12  := LD * A12
     + /LD * Q12

Q13  := LD * A13
     + /LD * Q13
```

Figure 4-7

AmPAL16R6D SRAM Address Counter—Non-interleaved, Section 2
Device U4

```
CLK NC02 /LD A14 A15 A16 A17 /IBACK /DBACK GND
 /OE NC12 /Q14 /Q15 /Q16 /Q17 /IBACK.D /DBACK.D NC19 VCC

Q14      := LD * A14
         + /LD * Q14

Q15      := LD * A15
         + /LD * Q15

Q16      := LD * A16
         + /LD * Q16

Q17      := LD * A17
         + /LD * Q17

BACK.D   := IBACK

DBACK.D  := DBACK
```

Figure 4-8

AmPAL16L8D SRAM Control Signal Generator—Non-interleaved
Device U3

$\overline{IBACK}$ $\overline{DBACK}$ $\overline{ILOAD}$ $\overline{DLOAD}$ $\overline{IBACK.D}$ $\overline{DBACK.D}$ $\overline{IBREQ.D}$ $\overline{DBREQ.D}$ $\overline{BINV.D}$ GND
$\overline{DREQ}$ $\overline{IRDY}$ $\overline{DRDY}$ $\overline{IREQ}$ RW $\overline{DOE}$ $\overline{WE}$ $\overline{CNT}$ $\overline{LD}$ VCC

$$IRDY = \overline{BINV.D} \cdot ILOAD + \overline{BINV.D} \cdot IBREQ.D \cdot IBACK.D$$

$$DRDY = \overline{BINV.D} \cdot DLOAD + \overline{BINV.D} \cdot DBREQ.D \cdot DBACK.D$$

$$DOE = \overline{BINV.D} \cdot RW \cdot DLOAD + \overline{BINV.D} \cdot RW \cdot DBREQ.D \cdot DBACK.D$$

$$LD = IREQ \cdot \overline{DBACK} \cdot \overline{ILOAD} + DREQ \cdot \overline{IBACK} \cdot \overline{DLOAD}$$

$$CNT = IBREQ.D \cdot IBACK \cdot \overline{BINV.D} + DBREQ.D \cdot DBACK \cdot \overline{BINV.D}$$

$$WE = \overline{BINV.D} \cdot \overline{RW} \cdot DLOAD + \overline{BINV.D} \cdot \overline{RW} \cdot DBACK.D \cdot DBREQ.D$$

Figure 4-9

AmPAL16R4D SRAM State Generator—Non-interleaved
Device U1

CLK $\overline{IREQ}$ IREQT A31 A30 A29 $\overline{Pin169}$ DREQT0 DREQT1 GND
$\overline{OE}$ $\overline{DREQ}$ $\overline{DBREQ.D}$ $\overline{IBACK}$ $\overline{DBACK}$ $\overline{ILOAD}$ $\overline{DLOAD}$ $\overline{IME}$ $\overline{DME}$ VCC

$$IBACK := \overline{DBACK} \cdot IME + \overline{IEXIT} \cdot IBACK$$

$$DBACK := \overline{BACK} \cdot DME + \overline{DEXIT} \cdot DBACK$$

$$ILOAD := \overline{DBACK} \cdot IME \cdot \overline{ILOAD}$$

$$DLOAD := \overline{IBACK} \cdot DME \cdot \overline{DLOAD}$$

$$IME = IREQ \cdot \overline{IREQT} \cdot \overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{Pin169}$$

$$DME = DREQ \cdot \overline{DREQT0} \cdot \overline{DREQT1} \cdot A31 \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{Pin169}$$

NOTE: The terms IEXIT and DEXIT used in the IBACK and DBACK equations are for clarity. Their true representations are as follows:

$$IEXIT = DME + IREQ \cdot \overline{IME}$$

$$DEXIT = IME \cdot \overline{DBREQ.D} + DREQ \cdot \overline{DME}$$

Intra-Cycle Timing

This memory architecture has two basic cycle timings. The first is a cycle used to decode the memory address and control signals from the processor. At the end of this decode cycle, the address is loaded into the address counter and the selected block of memory begins a burst access in the next clock cycle. The second cycle timing is that of a burst access.

The combination of a decode cycle followed by the first burst access cycle defines the two cycle initial access time. Each subsequent burst access requires one cycle.

Within the decode cycle the address timing path is made up of the following.

- The Am29000 clock to address and control signals valid delay of 14 ns,
- Address decode logic PAL delay of 10 ns (device U1),
- And the set-up time of the address counter PALs of 10 ns (devices, U4, U5, U6).

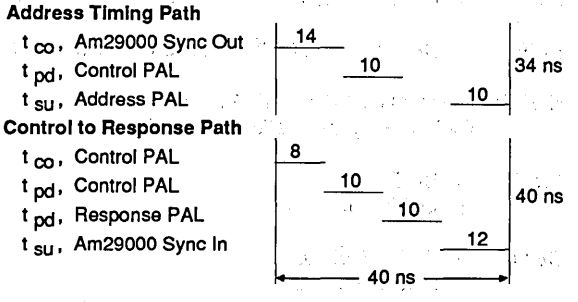
Assuming D-speed PALs those times total 34 ns. See Figure 4-10

Also, within the decode cycle time is the control signal to response signal path. This delay path is made up of the following:

- Clock to output time of registers within the control logic state machine PAL of 8 ns (devices, U1, U4),
- Propagation delay of the control logic PAL, 10 ns (device, U3),
- Propagation delay of a logic OR gate on the response signals from each memory block, 10 ns,
- And control signal set-up time of the processor, 12 ns.

Again assuming D-speed PALs, these times total 40 ns.

Figure 4-10



10117A-4.10

Non-Interleaved SRAM Decode Cycle

Within the burst access cycle the address to data path timing is determined by:

- the clock-to-output time of the address counter (8 ns for a D-speed, PAL) plus added delay due to capacitive and inductive loading by the memory array of the PAL outputs. Since this load exceeds the standard data sheet test loads, the analysis in Appendix A is used to estimate the added delay. The resulting estimated delay is 1.5 ns. The total delay is then an 8 ns clock-to-output time plus 1.5 ns added delay for a grand total of 9.5 ns.
- Memory access time of 20 ns;
- Data buffer delay of 4.3 ns;
- And the processor set-up time of 6 ns;

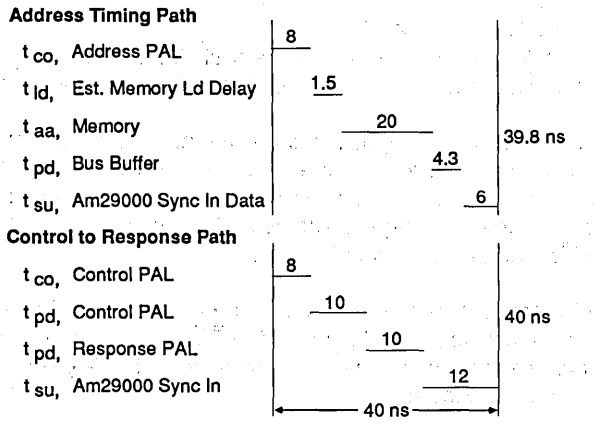
As shown in Figure 4-11, those delays total 39.8 ns worst case.

For the control signal to response signal path the time restrictions are the same in either the initial access or burst access cycles. The total delay is again 40 ns.

Inter-Cycle Timing

This section gives three examples of the cycle-by-cycle interaction between an Am29000 processor and the high-speed static memory system just defined in this Chapter. Each timing diagram includes the Am29000 control and response signals as well as all the internal signals of the memory control logic.

Figure 4-11



10117A-4.11

Non-Interleaved SRAM Burst Access

Instruction Burst Read

The waveform diagram provided in Figure 4-12, shows a burst read of instruction memory. In the first clock cycle the Am29000 initiates a read operation by making $\overline{IREQ}$ and address active. The access is a burst operation since the $\overline{IBREQ}$ signal also goes active late in the cycle. As a result, the address is decoded to signal $\overline{IME}$ indicating that this instruction memory is selected. Also, the $\overline{LD}$ signal goes active causing the memory address counters and latches to capture the address on the bus at the next rising edge of SYSCLK.

In cycle two, the address counters present the first address to the memory. The memory will access the selected data and have it on the bus in time for the Am29000 to receive it at the end of this clock cycle. Since the data is valid, the $\overline{IRDY}$ signal from the memory goes active. The registered value of $\overline{IBREQ}$ from cycle one is now available as the signal $\overline{IBREQ.D}$. This in combination with $\overline{IBACK}$ causes the $\overline{CNT}$ signal to go active. This will increment the address counter at the next rising edge of SYSCLK.

In cycles three and four, the second and third instruction words are read from memory. In cycle four the $\overline{IBREQ}$ signal goes inactive signaling a suspension of the burst access.

In cycle five, the memory control circuits see the absence of $\overline{IBREQ.D}$ and immediately make $\overline{IRDY}$ inactive. $\overline{CNT}$ also goes inactive to hold the address value until the burst is resumed. The suspension of the burst was only one cycle long because $\overline{IBREQ}$ again goes active in this cycle.

In cycle six, $\overline{IBREQ.D}$ is detected and $\overline{IRDY}$ immediately made active. $\overline{CNT}$ goes active again to continue the incrementing of address.

Cycles seven and beyond simply continue the burst access.

Instruction Burst Write

Figure 4-13 shows an example very similar to that of Figure 4-12. The difference is that this access is a burst write operation to the instruction memory.

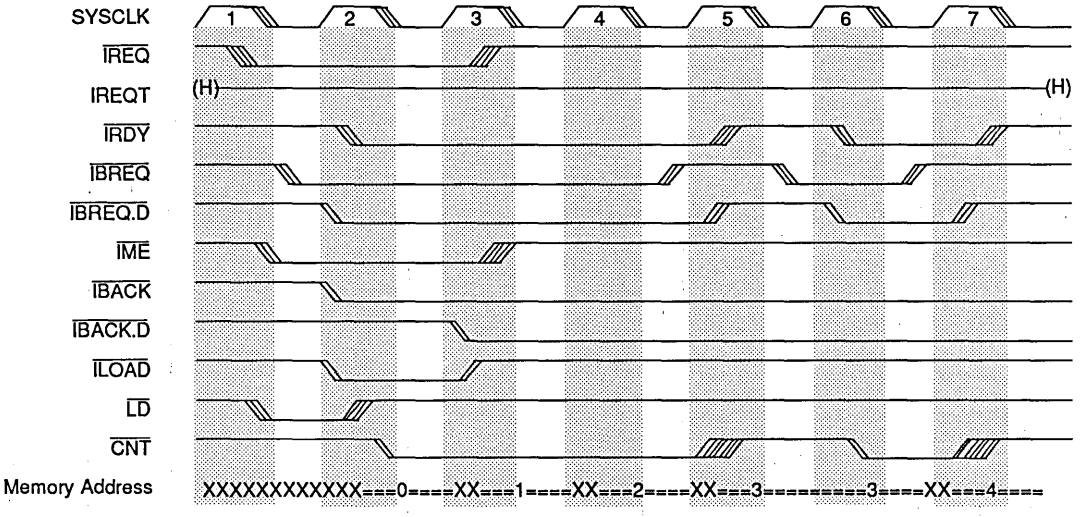
The flow of control signals is the same as for the instruction access just described. The only differences are that data words are now taken from the bus at those times when they would have been supplied during a read; data bus control and response signals are substituted for the equivalent instruction signals, e.g. $\overline{DREQ}$ goes active instead of $\overline{IREQ}$; and the write enable signal is active.

Note that there maybe a glitch on the write enable signal at the beginning of cycle three that is the result of switching on the $\overline{DBACK.D}$ and $\overline{DLOAD}$ lines. This glitch does not reach the memory write enable input since that is gated by SYSCLK via the OR gate (U7) in Figure 4-2.

Instruction Burst Preempt by Data Access

Figure 4-14 shows the interaction of a burst instruction access and a data read access addressed to the same block of memory.

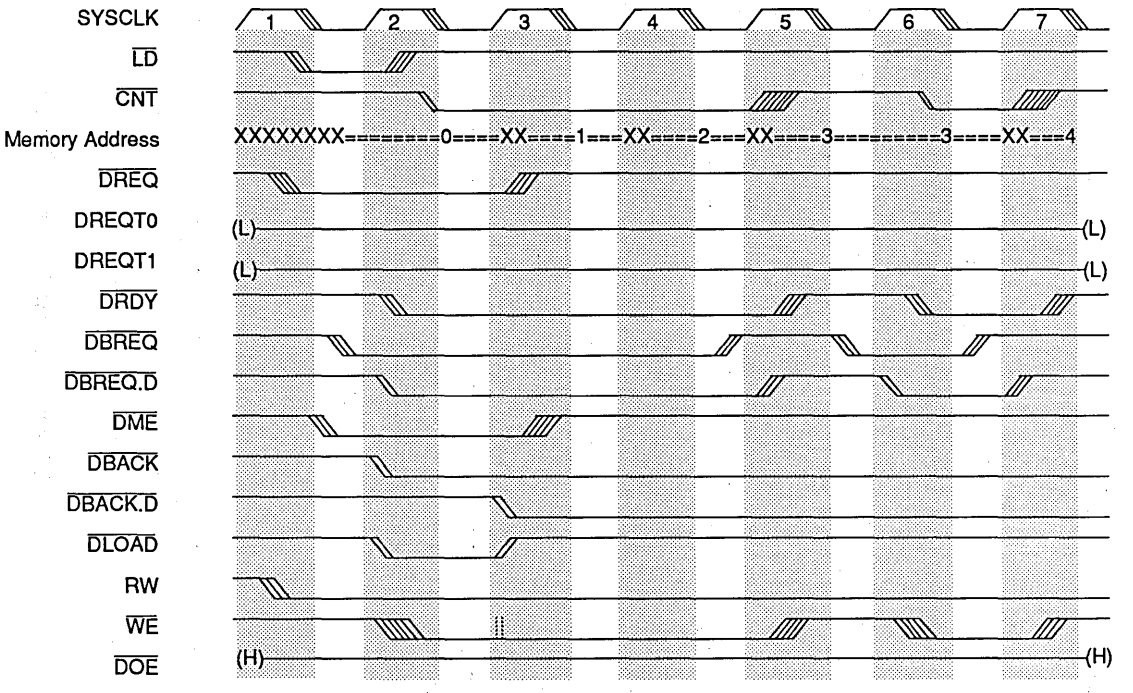
Figure 4-12



10117A-4.12 3/14/88 3/29/88 4/7/88

High-Speed Static RAM Burst Read of Instruction

Figure 4-13



10117A-4.13 3/14/88 3/29/88 4/7/88

High-Speed Static RAM Burst Write of Data

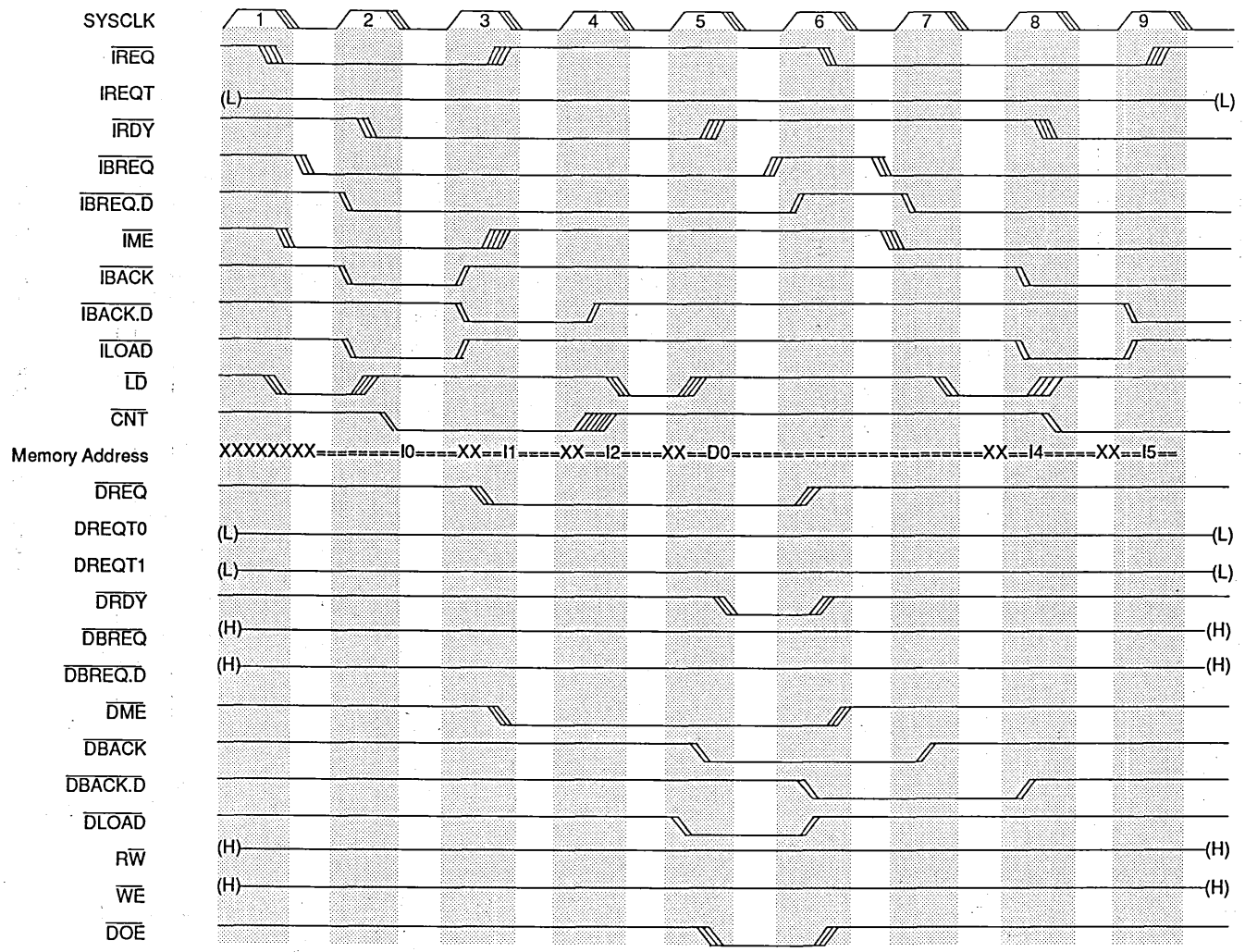


Figure 4-14

High-Speed Static RAM Data Preemption of Instruction Read

The first two cycles occur as previously described for the instruction burst read. In the third cycle, a data access is started by $\overline{DREQ}$ going active. The address is recognized as selecting this block of memory which is signaled by $\overline{DME}$ going active. Since data accesses are given priority over instruction accesses, the instruction access must now be preempted. The memory control state machine exits the IACCESS state and returns to the IDLE state in cycle four. This causes $\overline{IBACK}$ to go inactive, in cycle three, thus preempting the instruction access.

In cycle four, the last word of the instruction burst is supplied by the memory. Also, the $\overline{LD}$ signal goes active to enable the address counters to capture the data access initial address.

In cycle five, the instruction burst request is removed from the bus and the first word of the data access is presented to the bus. Since the $\overline{DBREQ}$ signal has not been active, the data access in this case is a single word rather than a burst.

In cycle six, the $\overline{DREQ}$ signal goes inactive as a result of the $\overline{DRDY}$ in cycle five, which in turn allows $\overline{IREQ}$ to go active to re-establish the preempted burst instruction access. The appearance of $\overline{IREQ}$ and $\overline{IME}$ causes the control state machine to return to the IDLE state in the next cycle.

In cycle seven, the load signal goes active to capture the instruction address.

In cycle eight the control state machine re-enters the IACCESS state with $\overline{IBACK}$ going active. The first word of instruction is placed on the bus with $\overline{IRDY}$. Also, $\overline{CNT}$ goes active to increment the address for the instruction fetch. The instruction burst is thus re-established.

Parts List

The parts list for the Am29000 High-Speed SRAM Interface is provided in Table 4-1.

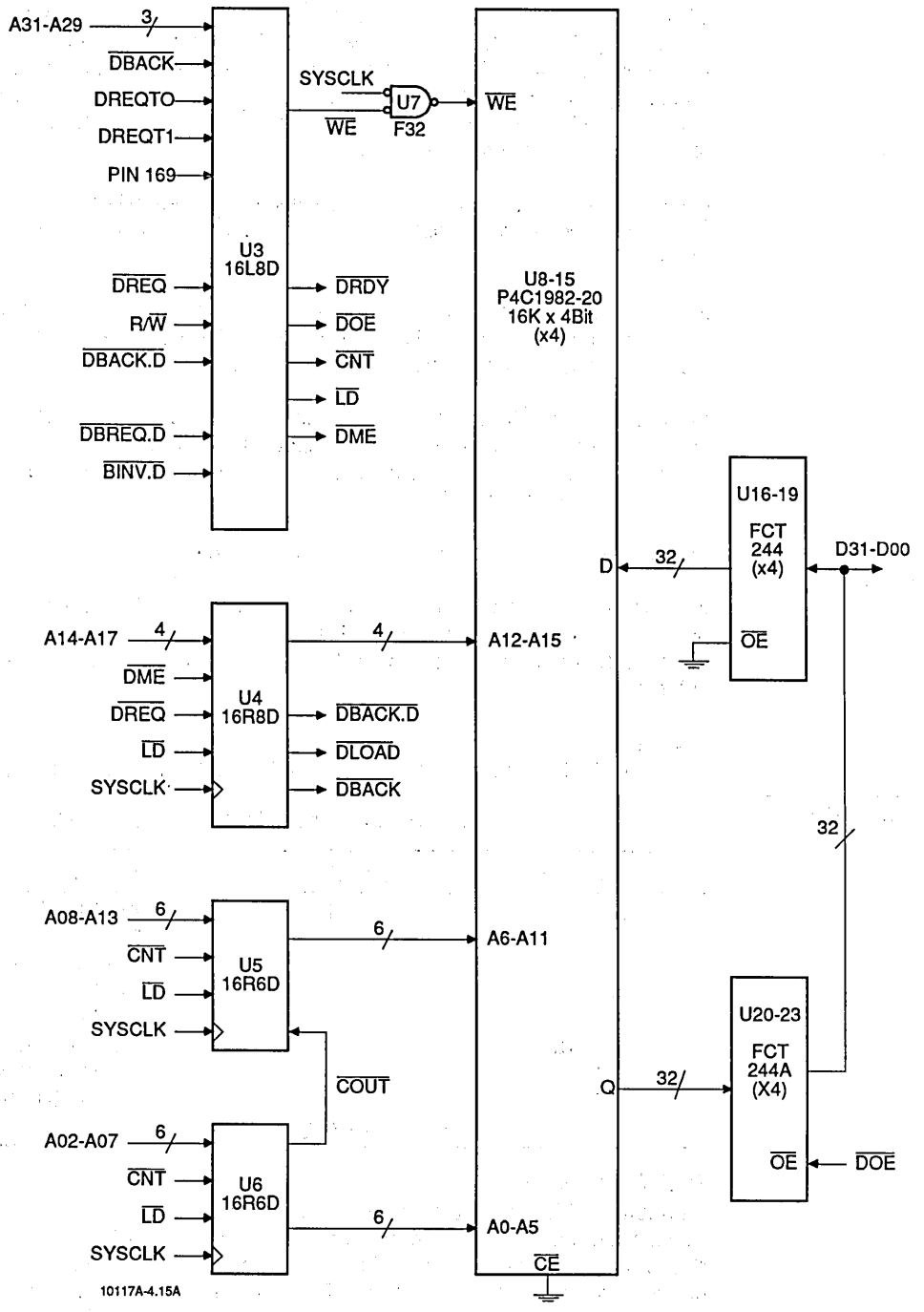
Table 4-1 **Am29000 High-Speed SRAM Interface Parts List**

Item No.	Quantity	Device Description
U1	1	AmPAL16R4D
U2	1	74F175
U3	1	AmPAL16L8D
U4-U6	3	AmPAL16R6D
U7	1	74F32
U8-U15	8	PC41982-20
U16-U19	4	IDT74FCT244
U20-U27	8	IDT74FCT244A
	27 pkgs	

DATA MEMORY

As shown in Figure 4-1 the instruction and data memories for the Am29000 are separate structures. The data memory can be an exact subset of the instruction memory design. In fact the exact same design can be used by tying the instruction related control signals to the inactive state. But, since the data memory is a subset, it is also possible to save a few chips by eliminating the instruction related control signals and rearranging the distribution of logic terms between PALs.

Figure 4-15



Data-Memory Block Diagram

As shown in Figure 4-15 versus Figure 4-2. it is possible to eliminate devices U1, AmPAL16R4D; U2, 74F175; and U24-U27, 74FT244A: a total of 6 chips. The output buffers for the instruction bus are not needed, the 74F175 register in the instruction memory can be shared with the data memory, and by rearranging logic terms as shown in Figures 4-16 and 4-17 the AmPAL16R4D PAL (U1) can be eliminated.

All other aspects of the design are the same as for the instruction memory described in the previous section.

Figure 4-16

AmPAL16L8D SRAM Control Signal Generator—
Non-Interleaved Data Memory Only Version.
Device U3

A31 A30 A29 $\overline{\text{Pin169}}$ $\overline{\text{DBACK}}$ $\overline{\text{DBACK.D}}$ DREQT0 DREQT1 $\overline{\text{DBREQ.D}}$ GND
$\overline{\text{BINV.D}}$ $\overline{\text{DREQ}}$ $\overline{\text{DRDY}}$ $\overline{\text{DME}}$ RW $\overline{\text{DOE}}$ $\overline{\text{WE}}$ $\overline{\text{CNT}}$ $\overline{\text{LD}}$ VCC

$$\text{DRDY} = \overline{\text{BINV.D}} \cdot \text{DLOAD} + \overline{\text{BINV.D}} \cdot \text{DBREQ.D} \cdot \text{DBACK.D}$$

$$\text{DOE} = \overline{\text{BINV.D}} \cdot \text{RW} \cdot \text{DLOAD} + \overline{\text{BINV.D}} \cdot \text{RW} \cdot \text{DBREQ.D} \cdot \text{DBACK.D}$$

$$\text{LD} = \text{DREQ} \cdot \overline{\text{DLOAD}}$$

$$\text{CNT} = \text{DBREQ.D} \cdot \text{DBACK} \cdot \overline{\text{BINV.D}}$$

$$\text{WE} = \overline{\text{BINV.D}} \cdot \overline{\text{RW}} \cdot \text{DLOAD} + \overline{\text{BINV.D}} \cdot \overline{\text{RW}} \cdot \text{DBACK.D} \cdot \text{DBREQ.D}$$

$$\text{DME} = \text{DREQ} \cdot \overline{\text{DREQT0}} \cdot \overline{\text{DREQT1}} \cdot \text{A31} \cdot \text{A30} \cdot \text{A29} \cdot \overline{\text{Pin169}}$$

Figure 4-17

AmPAL16R8D SRAM Address Counter—
Non-Interleaved, Section 2 Data Memory Only Version
Device U4

CLK DREQ $\overline{\text{LD}}$ A14 A15 A16 A17 $\overline{\text{DME}}$ NC07 GND $\overline{\text{OE}}$ NC12 $\overline{\text{Q14}}$ $\overline{\text{Q15}}$
$\overline{\text{Q16}}$ $\overline{\text{Q17}}$ $\overline{\text{DBACK}}$ $\overline{\text{DBACK.D}}$ $\overline{\text{DLOAD}}$ VCC

$$\text{Q14} := \text{LD} \cdot \text{A14} + \overline{\text{LD}} \cdot \text{Q14}$$

$$\text{Q15} := \text{LD} \cdot \text{A15} + \overline{\text{LD}} \cdot \text{Q15}$$

$$\text{Q16} := \text{LD} \cdot \text{A16} + \overline{\text{LD}} \cdot \text{Q16}$$

$$\text{Q17} := \text{LD} \cdot \text{A17} + \overline{\text{LD}} \cdot \text{Q17}$$

$$\text{DBACK.D} := \text{DBACK}$$

$$\text{DLOAD} := \text{LD} \cdot \overline{\text{DLOAD}} + \overline{\text{DREQ}} \cdot \text{DBACK}$$

CHAPTER 5
Medium Speed Static Ram With Interleaved Banks

Overview 5-1
What Is Interleaved Memory 5-1
A Basic Two Bank Design 5-1
Instruction Memory 5-1
Interface Logic Block Diagram 5-2
Memory Interface Logic Equations 5-6
PAL Definition Files 5-16
Intra-Cycle Timing 5-20
Inter-Cycle Timing 5-21
Parts List 5-29
Data Memory 5-29

MEDIUM SPEED STATIC RAM WITH INTERLEAVED BANKS

OVERVIEW

As can be seen from the last chapter, the simple "brute force" approach to memory design has its problems. Even with some of the fastest and most expensive static RAMs available, it is barely possible to meet the timing constraints of a single-cycle burst-access memory in a 25 MHz clock rate system.

Fortunately there is a fairly simple way to ease the timing constraints on the memory while still providing single cycle burst access at 25 MHz. This is called bank interleaving.

What is Interleaved Memory?

In a bank interleaved memory system, two or more separate memory banks are used to split up and overlap the memory-access workload. Each bank is assigned alternate words from the total memory space. In a 2-bank interleaved memory, one bank would contain all the odd words in the memory space and the second bank would contain all the even words. In a 4-bank memory, each bank would contain one out of every four words; the first bank would have words 0, 4, 8,..., the second bank would have words 1, 5, 9,..., the third bank would have words 2, 6, 10,..., the fourth bank would have words 3, 7, 11,..., etc.

For a burst access, the memory block is always used in a fixed sequential order. While one bank is transferring data on the system memory bus, the other bank(s) can be accessing data needed for a subsequent cycle. By staggering and overlapping the access time for each bank, the individual banks are allowed access times equal to one cycle for each bank of interleaved memory. A 2-bank memory allows two cycles of access time for each bank; a 4-bank memory allows four cycles. *While each bank is allowed multiple access cycles, the system memory bus sees a new data transfer on each cycle, thus maintaining single-cycle burst access while using slower memories.*

The trade-off involved is that the access time to the first word of a non-sequential address is determined by the access time of the individual bank selected. In a 2-bank memory this generally means the minimum initial access time is two cycles. It may be more than two cycles depending on how much time is used for address decoding. A 4-bank memory may need at least four cycles, etc. In addition, the control logic is more complex.

A Basic Two-Bank Design

The memory design described in this chapter is a simple extension of the memory design from the last chapter.

There are still separate blocks of memory for instruction and data, as was shown in Figure 4-1. Within each memory block, there are two banks of memory interleaved as odd and even words. Each bank is 64K words deep with each word being 32-bits wide. The total for the instruction memory block is then 128K words. The same is true for the data memory.

It is possible to use "55 ns access time" SRAM memories for all memory banks. The first cycle of a non-sequential access will require one cycle for address decode and two cycles for the first word accessed. Essentially, the inter-cycle timing is the same as for the high-speed SRAM memory of the last chapter except that each burst access is two cycles long. Overlapping the memory bank access time allows this longer access time to be hidden from the system viewpoint except on the first word of a non-sequential access. *The end result is a memory that provides 3-cycle access time for the first word of a non-sequential access and single cycle access for subsequent words in a burst transfer.*

The instruction memory block will have a read only port for sending instructions to the Am29000 and a read/write port tied to the Am29000 data bus. The read/write port provides access to the instruction memory via the data bus to allow instruction loading and memory diagnostics. The data memory will have a single read/write port connection to the Am29000 data bus.

INSTRUCTION MEMORY

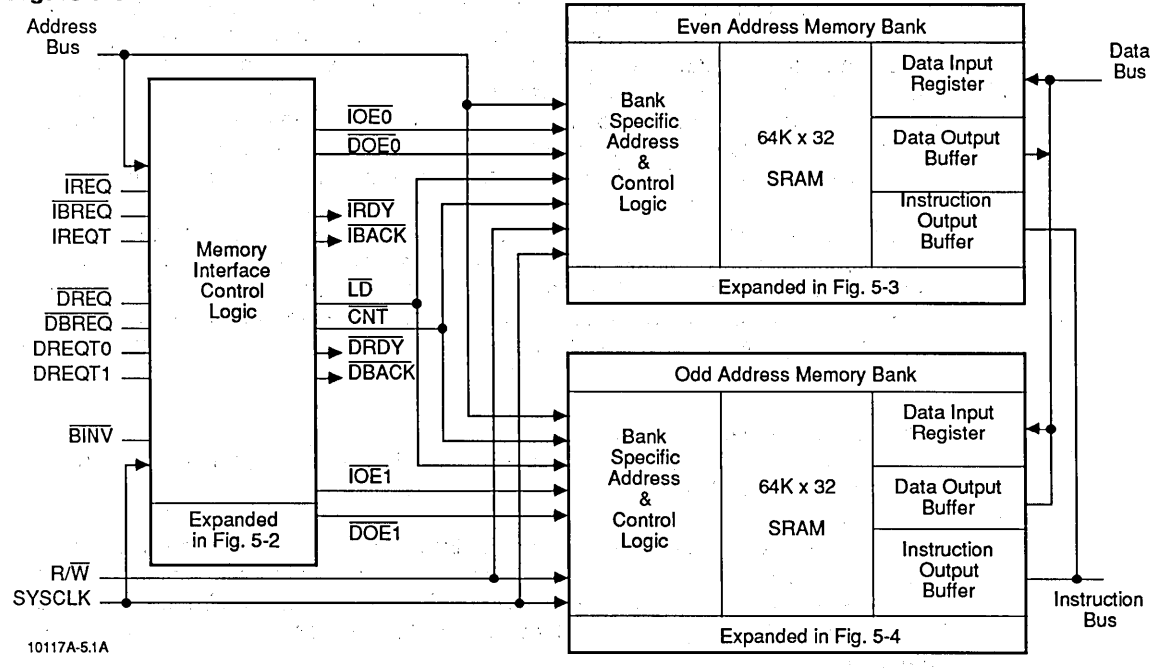
Interface Logic Block Diagram

Refer to the block diagrams in Figures 5-1 through 5-4.

The Memory

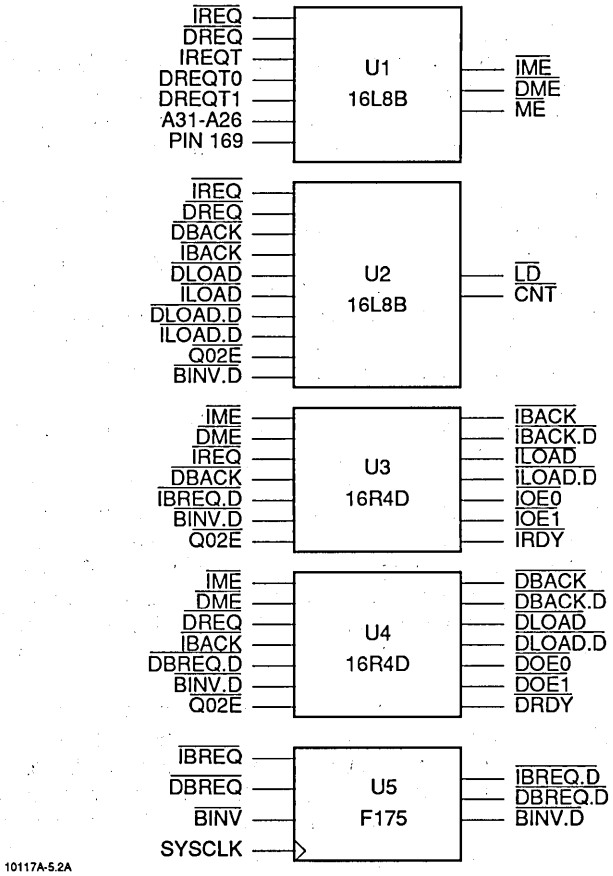
The memories are 64K x 1-bit SRAMs with separate data in and out lines. The access speed is 55 ns. Thirty-two devices are required in each bank to form the 32-bit wide instruction word for the Am29000. The two banks require a total of 64 RAM chips.

Figure 5-1



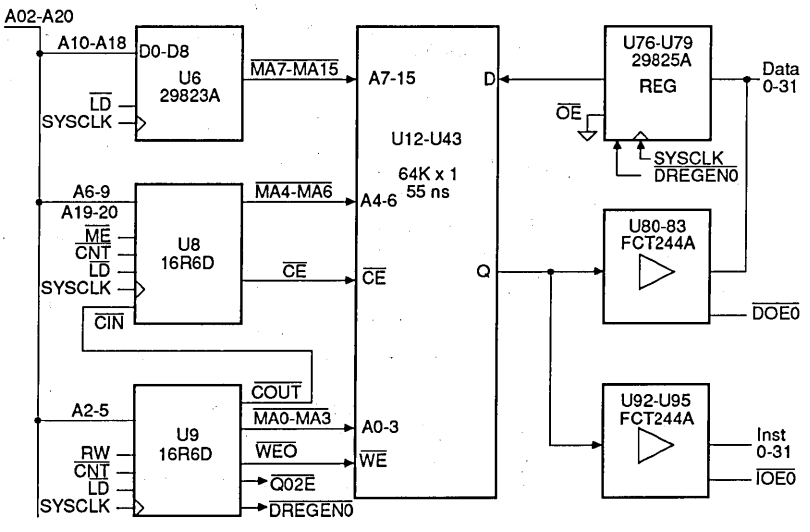
Interface Block Diagram

Figure 5-2



Memory Interface Control Logic

Figure 5-3



Even Address Memory Bank

Bus Buffers

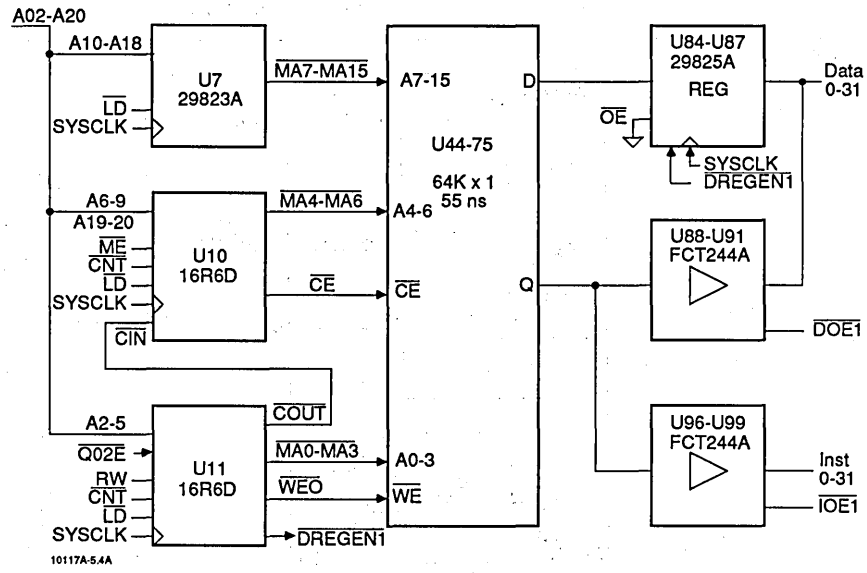
The memory data outputs are connected to the data bus lines via high-speed buffers. These buffers are required to isolate the memory outputs from the data bus whenever the memory is accessing instruction words. This isolation allows another data memory block to use the data lines while the instruction-memory block is fetching instructions.

The memory data inputs are connected to the data bus lines via Am29825A registers. These registers provide two advantages. They have a clock-to-output delay significantly shorter than the clock-to-data output valid time for the Am29000 (10 ns vs 18 ns); this makes it possible to meet the "data setup to end of write time" for 55 ns memories (≥30 ns) within the 40 ns clock cycle time. Also, they allow data to be removed from the bus one cycle earlier than would be the case if simple buffers were used; this makes a write operation one cycle faster than an equivalent read operation.

As will be shown later, the memory Write Enable ($\overline{WE}$) signal goes inactive one D-speed PAL clock-to-output delay later than the end of each cycle. It is therefore necessary to ensure that data at the output of the data registers is held at least until the worst-case clock-to-output time of the PAL to satisfy the memory's zero hold time on data with respect to $\overline{WE}$ signal going inactive. To guarantee this, two separate register banks are used, one for each bank of memory. Each register-bank clock is enabled only on the cycle that data is taken from the bus for the related memory bank. This ensures that the registered data is stable throughout the cycle and that data is being written during the following cycle to satisfy the hold time on data.

The memory data outputs are also connected to the instruction bus lines via buffers. These buffers serve to isolate the data outputs of this memory block from those outputs of other memory blocks which may also drive the instruction bus. Also, the buffers serve to isolate the even and odd banks of this memory block from each other so that simultaneous data access can go on in each bank independently.

Figure 5-4



Odd Address Memory Bank

Address Registers and Counters

To support burst accesses the lower seven address bits to each memory bank come from a loadable counter. An 8-bit counter is used to provide the address so that the least significant bit of the counter can be used to track which memory bank is connected to the data or instruction bus on each cycle. The 8-bit counter is built from one AmPAL16R4 and one AmPAL16R6 D-speed PALs. The D-speed PALs are used because their clock-to-output delay is significantly faster than standard MSI 8-bit counters. Also, the use of PALs allows additional functions to be integrated into the same packages used for the counter function.

The upper nine bits of memory address need not come from a counter since the Am29000 will always output a new address when a 256 word boundary is crossed. The upper nine bits of address are simply registered by an Am29823A 9-bit register.

A separate set of address counter and register logic is used to address each memory bank. This is done for two reasons. One is that when one bank is connected to the data or instruction bus, the other bank will be accessing the next word in sequence. This requires that the two banks have independently incremented addresses. The address for each bank will increment on different cycles. The second reason is that each bank of memory presents a heavy capacitive load to the address counter and register outputs. Giving each bank its own counter and register keeps the capacitive load reasonable and thus maintains system speed.

For these same reasons the memory Chip Enable ($\overline{CE}$) signal, and Data Register Enable ($\overline{DREGEN}$) control logic for each bank is integrated into the same PALs as are used for the address counters.

Registered Control Signals

As noted earlier, the timing of the $\overline{IBREQ}$, $\overline{DBREQ}$, and $\overline{BINV}$ control signals require that they be registered by a low setup time register such as a 74F175 register.

Interface Control Logic

This logic must generate the memory response signals, manage the loading and counting of memory addresses, and control the data buffer output enables. The logic functions needed for this require four PALs, two AmPAL16R4D and two AmPAL16L8B.

In Figure 5-2, device U1 an AmPAL16L8B performs address decode for instruction and data accesses. Its outputs indicate when this memory block has been addressed.

Device U2, also an AmPAL16L8B produces the Load ($\overline{LD}$) and Count enable ($\overline{CNT}$) signals for the address counters.

Device U3 is the instruction portion of the memory interface state machine which manages the Instruction Ready ($\overline{IRDY}$) response signal and the Instruction bus buffer Output Enable ($\overline{IOE}$) signals.

Device U4 performs the same state machine function as in U3 with reference to the Data Ready ($\overline{DRDY}$) and Data bus buffer Output Enable ($\overline{DOE}$) signals.

Response Signal Gating

As noted in the last chapter, the memory response signals from all system bus devices

are logically ORed together before being returned to the Am29000 processor. An example of this circuitry was shown in Figure 4-3. These gates are not counted as part of the components within the memory design since they are shared by all the bus devices in the system and as such are part of the overhead needed in any Am29000 system.

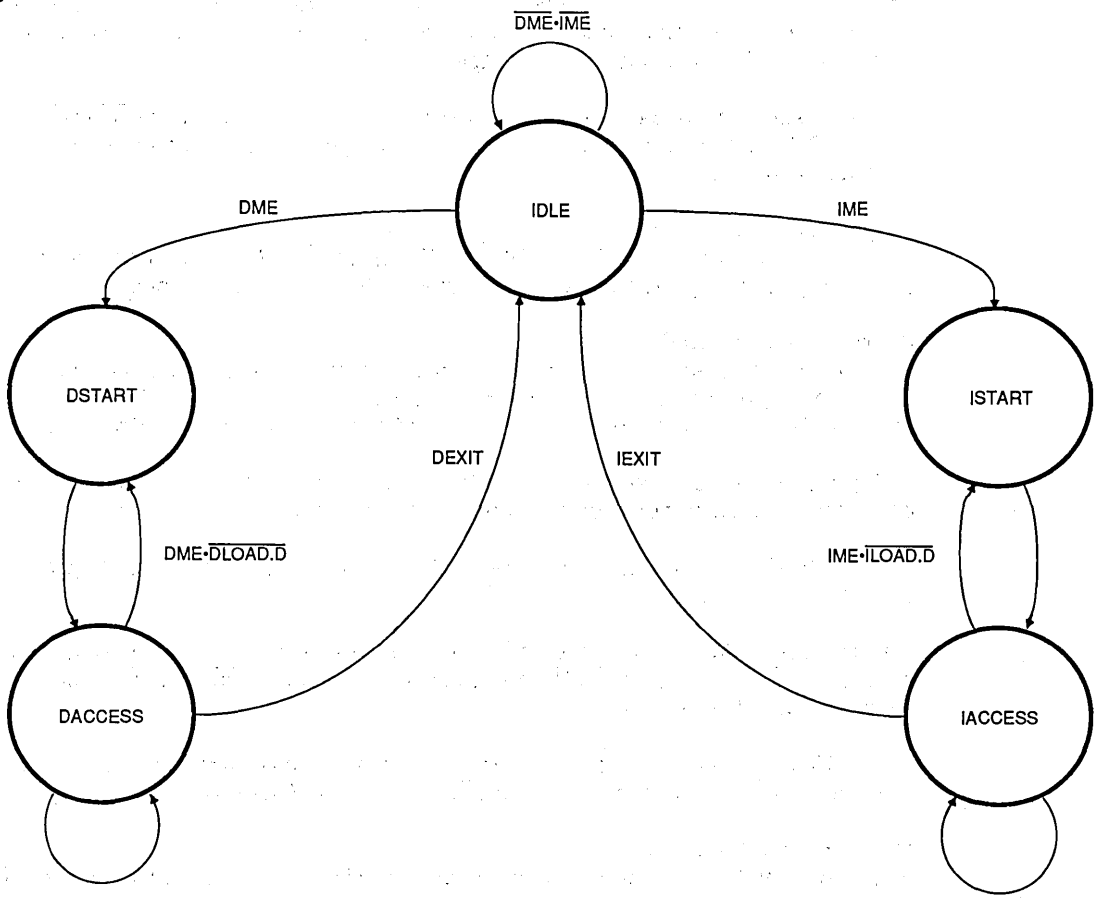
Memory Interface Logic Equations

State Machine

The control logic for this memory (devices, U3 and U4, Figure 5-2) can be thought of as a Mealy-type state machine in which the outputs are a function of the inputs and the present state of the machine. This structure is required since some of the output signals must be based on inputs which are not valid until the same cycle in which the outputs are required to effect control of the memory.

As shown in Figure 5-5, this state machine can be described as having five states. These states control the enabling of activity on the Burst Acknowledge, output buffer

Figure 5-5



10117A-5.5A

Interleaved SRAM Control State Machine

enable, and Ready lines. IDLE is the default state of the interface state machine. It is characterized by Instruction Burst Acknowledge ($\overline{IBACK}$) and Data Burst Acknowledge ($\overline{DBACK}$) both being inactive. This state serves as a way of identifying when the memory is not being accessed and could be placed into a low-power mode. Note: A more detailed explanation of power-mode usage is provided in the discussion of the $\overline{CE}$ signal. The more important use of this state is as a delay cycle in the transition between an active instruction burst access being preempted and the start of the preempting data access. The delay is needed to allow the completion of the final instruction access in the cycle that $\overline{IBACK}$ is deasserted and the instruction burst access is preempted. A transition to either the Instruction Start (ISTART) or Data Start (DSTART) state occurs when an address selecting this memory block is placed on the address bus.

The ISTART state occurs during the first cycle of memory access following a new instruction address being presented on the address bus. During this state the $\overline{IOE}$ and $\overline{IRDY}$ lines are held inactive and the $\overline{IBACK}$ line is active. This state is used as a delay to account for the initial access time of both the even and odd memory banks when a new address is presented on the bus. The transition to the Instruction Access (IACCESS) state is unconditional.

The IACCESS state is used during the second cycle of a new address access and during all subsequent burst access cycles, whether active or suspended. In this state the $\overline{IOE}$ and $\overline{IRDY}$ lines are allowed to be active as required by the active or suspended status of an instruction burst request. When a new instruction address selecting this memory block appears on the bus a transition to the ISTART state will occur. If a new instruction address appears which does not select this memory block then a transition to the IDLE state occurs. Also, if a data address selecting this memory block appears there will be a transition to the IDLE state to force a preemption of the current instruction access. The state machine remains in the IACCESS state as the default if no other state transition condition appears.

The DSTART state is equivalent to the ISTART state but results from a data address which selects this memory block. One other difference is that the $\overline{DRDY}$ line will be active in this cycle during a write operation. The transition to the Data Access (DACCESS) state is unconditional.

The DACCESS state is equivalent to the IACCESS state. Transition from this state is different only in that the transition to the IDLE state will occur only when a data access completed and a new data or instruction access starts. A data access will not be preempted by an instruction access to this memory.

Logic Details—Signal-by-Signal

All signals are described in active high terms so that the design is a little easier to follow. The signals as implemented in the final PAL outputs will often be active low as required by the actual circuit design. The actual PAL Definition files,are included in Figures 5-6 through 5-11.

Note that in the equations, an equal sign indicates a combinatorial signal and a colon followed by an equal sign indicates a registered PAL output.

IME — In this memory interface, it is assumed that other blocks of instruction or data memory may be added later, and that there may be valid addresses in address spaces other than instruction/data space.

This means that this memory will only respond with IBACK or DBACK active when this block has been selected by valid addresses in the instruction/data space. This requires that at least some of the more significant address lines above the address range of this memory block be monitored to determine when this memory block is addressed. Also, it means the IREQT, DREQT0, DREQT1, and Pin 169 lines must be monitored to determine that an address is valid and lies in the instruction/data space.

IME (Instruction for ME) is the indication that the address of this memory block is present on the upper address lines, an instruction request is active, Pin 169 is inactive (test hardware has not taken control), and instruction/data address space is indicated. In other words this memory block is receiving a valid instruction access request. This example design will assume that the address of this memory block is equal to $\overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27}$. The equation for this signal is:

$$IME = IREQ \cdot \overline{IREQT} \cdot \overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27} \cdot \overline{Pin169}$$

DME — DME (Data for ME) is the indication that the address of this memory block is present on the upper address lines, a data request is active, Pin 169 is inactive, and instruction/data address space is indicated. In other words this memory block is receiving a valid data access request. This example design will assume that the address of this memory block is equal to $A31 \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27}$. Note that for instruction accesses the memory address for this block had A31 = zero where the data accesses to this block are valid for A31 = one. This allows instruction memory for instruction accesses to be located at address zero while having the window for data bus access to the instruction memory located at a different base address. This allows the separate data memory block used in this design to have its base address also at zero. Thus both the instruction and data memories are located at address zero in their respective address spaces.

The equation for this signal is:

$$DME = DREQ \cdot \overline{DREQT0} \cdot \overline{DREQT1} \cdot A31 \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27} \cdot \overline{Pin169}$$

ME — The ME (instruction or data for ME) is in effect an OR of the IME and DME signals and is used to indicate when this memory block is addressed for either instruction or data accesses. The ME signal is used to determine when the CE signal for the memory banks will be active. The equation is:

$$\begin{aligned} ME = {} & IREQ \cdot \overline{IREQT} \cdot \overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27} \cdot \overline{Pin169} \\ & + DREQ \cdot \overline{DREQT0} \cdot \overline{DREQT1} \cdot A31 \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27} \cdot \overline{Pin169} \end{aligned}$$

IEXIT — Instruction EXIT (IEXIT) is an intermediate equation term not actually implemented as an output of the SRAM State Generator, Device U3. The logic of the term is used in the generation of IBACK but the name IEXIT is simply a documentation convenience.

The IEXIT equation is:

$$\begin{aligned} IEXIT = {} & DME \\ & + IREQ \cdot \overline{IME} \end{aligned}$$

A data request to this memory block for instruction data space will take priority over an instruction fetch in progress. Also, if a new instruction fetch stream is started for either

another block of memory or to instruction ROM this memory interface can return to the idle state.

DEXIT — Like IEXIT, Data EXIT (DEXIT) is a term used only for documentation convenience.

The DEXIT (DEXIT) equation is:

$$\begin{aligned} DEXIT &= IME \cdot \overline{DBREQ.D} \\ &+ DREQ \cdot \overline{DME} \end{aligned}$$

An instruction request to this memory block for instruction/data space when the DBREQ signal was inactive in the last cycle will end any suspended data access. Requiring DBREQ to be inactive will hold off instruction fetches until the current data access is complete or suspended. Also, if a new data access stream is started for, another block of memory, to I/O space, or to coprocessor space, this memory interface can return to the idle state.

IBACK — The Instruction Burst Acknowledge (IBACK) signal is sent to the Am29000 as an indication that the interface state machine is in an active or suspended instruction access. The equation is:

$$\begin{aligned} IBACK &:= IME \cdot \overline{DBACK} \cdot \overline{BINV} \\ &+ \overline{IEXIT} \cdot IBACK \end{aligned}$$

The IBACK active state is entered when an instruction request to instruction data space with the address of this memory block is active and a data access is not currently active. The $\overline{DBACK}$ term will give an active data access priority by holding off instruction accesses until the data access is completed. The $\overline{BINV}$ input will prevent an access from beginning in the event bus signals are invalid.

Once IBACK is active it will stay active until one of the IEXIT conditions is satisfied.

IBACK.D — The Instruction Burst Acknowledge Delayed (IBACK.D) signal is simply a one cycle delayed version of IBACK.

$$IBACK.D := IBACK$$

It is used in the generation of IRDY, IOE0, and IOE1.

DBACK — The Data Burst Acknowledge (DBACK) signal is sent to the Am29000 as an indication that the interface state machine is in an active or suspended data access. The equation is:

$$\begin{aligned} DBACK &:= DME \cdot \overline{IBACK} \cdot \overline{BINV} \\ &+ \overline{DEXIT} \cdot DBACK \end{aligned}$$

The DBACK active state is entered when a data request-to-instruction/data space with the address of this memory block is active and an instruction access is not currently active. The IBACK term will hold off the beginning of a data access until any active instruction access is preempted. The $\overline{BINV}$ input is used to ignore bus signals during invalid cycles.

Once DBACK is active it will stay active until one of the data exit (DEXIT) conditions is satisfied.

DBACK.D — This is simply a one cycle delayed version of DBACK.

DBACK.D := DBACK

It is used in the generation of DRDY.

LOAD — Load (LD) is the signal which enables the lower address bit counters and the upper address bit registers to load a new address on the next rising edge of SYSCLK. The equation is:

$$LD = IREQ \cdot \overline{DBACK} \cdot \overline{ILOAD} \cdot \overline{ILOAD.D} + DREQ \cdot \overline{IBACK} \cdot \overline{DLOAD} \cdot \overline{DLOAD.D}$$

When an instruction request (IREQ) is active, LD is prevented from being active while a data access is active or suspended. In other words, when the state machine is in the DSTART or DACCESS state, a load which would otherwise result from an IREQ is suppressed. This prevents the changing of the address counter values until the instruction access can be preempted and terminated.

The LD signal is also limited to being one cycle long by suppressing LD when either Instruction LOAD (ILOAD) or Instruction LOAD Delayed (ILOAD.D) is active. These signals are delayed versions of the LD signal and they suppress LD during the two cycles following the initial appearance of IREQ. The LD signal must be suppressed during this time so that the count (CNT) signal to the address counters may be active and cause the counters to increment. Further suppression beyond the cycle that ILOAD.D is active is not needed since IRDY will go active during the ILOAD.D cycle. IRDY going active will cause IREQ to go inactive in the following cycle if no new instruction address is needed. If IREQ is active following the ILOAD.D cycle then a new instruction address is present and a new LD signal pulse will be allowed. Also note that if the instruction access is done in burst mode, the appearance of IBACK during the ILOAD active cycle will cause IREQ to go inactive for the duration of the burst access.

Similarly, for the case that DREQ is active, load is prevented when the IBACK is active or when load was active in the last two cycles.

The LD signal is combinatorial so that it can be active during the first cycle of a new instruction or data request.

ILOAD — The Instruction LOAD (ILOAD) is a delayed version of the LD signal with a qualification. The qualification is that the ILOAD is active when:

- Load occurs for an instruction fetch.
- The bus is valid during the cycle that, IREQ is active.
- The instruction fetch is addressed to this memory block.

This qualification prevents false starts in memory access due to an invalid bus situation.

$$ILOAD := \overline{DBACK} \cdot IME \cdot \overline{ILOAD} \cdot \overline{ILOAD.D} \cdot \overline{BINV}$$

ILOAD is used in the generation of the IRDY, IOE0, IOE1, CNT, and LD signals. Like LD, ILOAD is limited to be a single cycle in duration.

ILOAD.D — The Instruction LOAD.Delayed (ILOAD.D) signal is simply a delayed version of the ILOAD signal. The equation is:

ILOAD.D := ILOAD

DLOAD — Data LOAD (DLOAD) is a delayed version of the LD signal with the qualification that it is active only when a load occurred for a data access which was addressed to this memory block and the instruction/data space.

$$DLOAD := \overline{IBACK} \cdot DME \cdot \overline{DLOAD} \cdot \overline{DLOAD.D} \cdot \overline{BINV.D}$$

DLOAD is used in the generation of the DRDY, DOE0, DOE1, CNT, and LD signals. Like LD, DLOAD is limited to be a single cycle in duration.

DLOAD.D — The Data LOAD.Delayed (DLOAD.D) signal is simply a delayed version of the DLOAD signal in the same way that ILOAD.D is a delayed version of ILOAD. The equation is:

DLOAD.D := DLOAD

CNT — The Count (CNT) signal causes the address counters to increment on the next rising edge of SYSCLK.

$$\begin{aligned} CNT &= ILOAD \\ &+ DLOAD \\ &+ \overline{BINV.D} \cdot IBREQ.D \cdot IBACK \\ &+ \overline{BINV.D} \cdot DBREQ.D \cdot DBACK \end{aligned}$$

The CNT signal will be active when the respective IBREQ or DBREQ and IBACK or DBACK signals are active in the previous cycle, given also that the bus was not invalid.

A CNT signal is forced during the ILOAD or DLOAD cycle to ensure that the LSB of the even counter is pointing to the correct memory bank in the event that no burst request is active. In other words when a single access is requested.

Note that for both the even and odd bank counters, only the upper seven bits are used as the lower address bits to memory. The LSB of the counters serve to cause the memory bank address to increment on every other cycle that the CNT signal is active.

The CNT equation provides a count enable to the even counter during both even and odd word initial address accesses. This would appear to be an extra cycle of counting for the even bank. This is done for the following reason: when a burst access begins on an odd word boundary, it is necessary to have the even bank access the even word that follows the initial odd word. This means that the address going to the even bank will always to be one greater than the address going to the odd bank. This requires that the initial address from the address bus be incremented to point to the next higher even bank memory word. This could be accomplished by placing a combinatorial incrementer in the address path to the even bank address counter, but incrementer logic is already defined as a part of the address counter. When the initial access address is odd, the even bank need not begin its access cycle until the third clock cycle of the

access. This means that the even bank address counter can be loaded with the initial address at the end of the first cycle of the access and incremented in the counter at the end of the second cycle. In effect this makes use of the incrementer logic already in the counter to increment the even address to point to the next even word in sequence.

IRDY —The Instruction Ready (IRDY) indicates that there is valid read data on the instruction bus.

$$\begin{aligned} IRDY &= ILOAD.D \\ &+ \overline{BINV.D} \cdot IBREQ.D \cdot IBACK.D \cdot \overline{ILOAD} \end{aligned}$$

This static memory design will always be ready with data in the second cycle after a new instruction request as implied by ILOAD.D. The memory will also be ready when IBREQ was active with IBACK in the previous cycle. IBACK is required as a qualifier so that when an access is preempted the continued presence of IBREQ will not cause a false ready indication. The $\overline{BINV.D}$ signal is used to prevent false ready indications if the bus was invalid in the previous cycle. Note that situation can occur during a suspended access when the processor grants the bus to another bus master. The $\overline{ILOAD}$ signal prevents IRDY from going active during the ILOAD cycle of a new instruction access when that access immediately follows a previous suspended burst access. In that situation the IBACK signal would already be active during the initial IREQ cycle of the new access. And if the new access is a burst access the IBREQ signal would also go active during the initial IREQ cycle. Without the $\overline{ILOAD}$ signal, that combination of events would cause IRDY to go active one cycle too early for the new access.

The reason that IRDY must be a combinatorial signal is that IBREQ comes very late in the previous cycle and must be registered. There is no time to perform logic on IBREQ in the previous cycle before SYSCLK rises. This means that the information that IBREQ was active in the last cycle is not available until the cycle in which IRDY should go active for a resumption of a suspended burst access.

IOE0 and IOE1 — The Instruction Output Enable (IOE) signal controls for the even and odd memory banks are used to control which bank is allowed to drive the instruction bus during each cycle. The signals use essentially the same logic as IRDY except that each signal is further qualified by the output of the LSB of the even bank counter (Q02E). This bit keeps track of which memory bank is ready to provide data to the instruction bus. The even bank is enabled when IRDY is active and the Q02E bit is one. The odd bank is enabled when IRDY is active and Q02E is zero.

$$\begin{aligned} IOE0 &= Q02E \cdot ILOAD.D \\ &+ \overline{BINV.D} \cdot Q02E \cdot IBREQ.D \cdot IBACK.D \cdot \overline{ILOAD} \end{aligned}$$

$$\begin{aligned} IOE1 &= \overline{Q02E} \cdot ILOAD.D \\ &+ \overline{BINV.D} \cdot \overline{Q02E} \cdot IBREQ.D \cdot IBACK.D \cdot \overline{ILOAD} \end{aligned}$$

DRDY — The Data ReaDY (DRDY) is the equivalent of IRDY for data accesses and therefore uses the same equation with data-respective terms substituted for instruction terms. The one additional change is that a term is added to cause DRDY to occur one cycle early during write operations. This is done because the data to be written is taken from the data bus into a register before actually being stored in the memory. This maintains the same memory timing used during read operations but write data is

removed from the bus one cycle earlier than when DRDY would normally go active during a data read operation.

$$\begin{aligned} DRDY &= \overline{RW} \cdot DLOAD \\ &+ RW \cdot DLOAD.D \\ &+ \overline{BINV.D} \cdot DBREQ.D \cdot DBACK.D \cdot \overline{DLOAD} \end{aligned}$$

DOE0 and DOE1 — The Data Buffer Output Enable (DOE1/DOE2) signals serve the same function for DRDY as does the IOE0 & IOE1 signals do for IRDY. The description for them is the same as for the IOE signals. The only difference being that the DOE signals will be active only during read operations.

$$\begin{aligned} DOE0 &= \overline{Q02E} \cdot RW \cdot DLOAD.D \\ &+ \overline{BINV.D} \cdot Q02E \cdot RW \cdot DBREQ.D \cdot DBACK.D \cdot \overline{DLOAD} \end{aligned}$$

$$\begin{aligned} DOE1 &= \overline{Q02E} \cdot RW \cdot DLOAD.D \\ &+ \overline{BINV.D} \cdot \overline{Q02E} \cdot RW \cdot DBREQ.D \cdot DBACK.D \cdot \overline{DLOAD} \end{aligned}$$

WE — The Write Enable (WE) signal is a registered signal that goes active during the second cycle of each two cycle access period for each word access of a memory bank. The WE signal will go active only during write operations.

Since it is registered, it will stay active throughout the second cycle of each access period in order to satisfy the required WE signal pulse width of 35 ns. The WE signal will go active only if a DRDY signal for the data was active in the previous cycle which indicates that the memory has registered valid data from the data bus ready to be written into the memory bank. The WE signal is also qualified by which bank the signal is being generated for and by the indication of which bank should be written in the second cycle of the access period during a given clock. This last qualifier is effectively the LSB of the even bank counter. In the case of the odd bank counter the value of the LSB output of the even bank counter is brought into the equation via the A02 input of the odd counter (note that since the even bank counter Q02 output is low true, the inverted A02 input is used in the equation). The equation shown here has an input called ODD. That input is strapped high or low depending on which bank counter is being implemented. The reason for this is that the same set of PAL equations that implement the lower even and odd bank counters can be the same given that this ODD input is tied to the appropriate voltage. This allows one equation set to be used for the lower half of both bank counters. Note that the bank WE signal is implemented in the lower of the two bank counter PALs. The equation is as follows:

$$\begin{aligned} WE := &\ ODD \cdot DRDY \cdot A02 \cdot \overline{RW} \\ &+ \overline{ODD} \cdot DRDY \cdot \overline{Q02} \cdot \overline{RW} \end{aligned}$$

DREGEN — Data REGister ENable (DREGEN) is the signal that enables the write data register on the D input of each memory bank to load new data. The equation used is similar to that used for WE except that a combinatorial output is used so that the register will load at the end of the DRDY active cycle. Also the equation is simpler since the register loading only needs to be restricted by the active bank indication served by the LSB bit of the even counter.

$$\begin{aligned} DREGEN &= ODD \cdot \overline{A02} \\ &+ \overline{ODD} \cdot \overline{Q02} \end{aligned}$$

CE — The Chip Enable (CE) signal for this memory block is used to lower the dynamic power of the system by switching off the memories when they are not being accessed. The equation for this is:

$$CE := \overline{LD} \cdot ME + \overline{LD} \cdot CE$$

This block enable is based on the OR of the IME and DME signals. When this block is addressed with either an instruction or data access, the memories receive CE signal on the next cycle. This selection is held until the next time the load signal is active in this memory block.

It is worth noting that this equation will not allow the memory to go into a deselected or low power mode until the cycle following a transition to the IDLE state. This ensures that the memory is still active on the last access of a preempted instruction burst request.

ADDRESS COUNTERS — There is one address counter for each bank of memory. Each is implemented with one AmPAL16R4D and one AmPAL16R6D device (Figures 5-3: U8, U9; Figure 5-4: U10,U11). The counter function is split across two PALs due to the number of product terms required to implement the upper bits of the counter. The lower half of the counter produces a carryout signal to the upper counter half. The equations for the counters are the same except for a difference in treatment of the LSB between banks. This allows the same logic to be used for both bank counters with a single input used to select logic specific to the even or odd bank usage. The selecting input is called $\overline{ODD}$. When the counter PAL is used in the even bank this input is tied high and tied low for use in the odd bank.

The LSB bit of each counter is used as the means to control the timing of when the upper seven bits of each counter will increment. The upper bits of each counter will increment on every cycle that the count signal is active and the LSB is also active.

The value of the LSB in each counter will be different in any given cycle, which will cause the upper bits of the counters to increment on different cycles with regard to each other. In other words, the the upper seven bits of the counters will be out of phase in terms of when they increment. This allows one bank of memory to start the access of the next word in sequence while the other bank completes the access of the current word.

A little added explanation may be in order here. Beyond the first completed access of a burst transfer the counter activity is consistent and mechanical. For every cycle that IBREQ or DBREQ and the appropriate burst acknowledge signal is active, both counters will receive a count enable signal. The LSB of the counters will be of opposite polarity so that the upper seven bits of each counter increment on alternate cycles. The LSB of the counters then act as accurate indicators of when each bank of memory is actively writing data from the bus or providing data to the bus. The difficulty in managing the counters comes during the first access in a burst transfer. At that time, the memory address is the single source for the initial counter value for both counters. Depending on whether the initial address is odd or even, the odd or even bank of memory is accessed; consequently that bank's counter must be incremented first so the address counters can begin the alternating counting scheme needed in all the following

burst transfers. In addition, if the initial address is odd, the even bank memory address must be incremented to point to the next even word in sequence before the even bank can begin a valid access of data.

There are various ways to manipulate the counter values so that the counters have the needed output values and increment in the right sequence. They involve decisions about whether one or two separate count enable signals will be used; whether incrementer logic will be placed in front of the even bank counter or instead the even bank counter will be incremented one extra time before its first use; and whether the LSB of one or both counters will initially be forced to values different from the initial address in order to make the counting sequence begin correctly. The following describes the counter implementation for this particular design, this scheme was chosen because it appeared to minimize the number of required PALs.

The LSB of the even counter is simply treated as the LSB of an 8-bit counter. It is loaded from the memory address at the end of the first cycle in each new memory access. It is incremented (toggled) at the end of each cycle in which the count signal is active. The output of the even bank LSB (Q02E) is used in several other equations where bank selection information is needed. When Q02E is high it indicates that the even memory bank is in the second half of an access sequence (the access sequence is two cycles long). During this second half of the sequence, data will be provided to the bus on a read or data will be written from the data bus registers during a write operation. When Q02E is low it indicates that the odd bank is in the second half of its sequence.

The LSB of the odd counter is handled a little differently. By examining the required counting sequence for the odd counter during both even and odd initial accesses it can be seen that the LSB of the odd counter is almost always a one cycle delayed version of the even bank counter LSB. The only cycle where this might differ would be during the first cycle after the load of a new memory address where the odd counter LSB could be loaded with the LSB of the initial address. If this were done it would be necessary to provide a separate count enable on the odd bank counter to prevent incrementing the odd bank before the first address was used. That count-enable scheme would differ from the one required by the even bank counter which must always increment in the first cycle after the initial address load. By always forcing the odd counter LSB to zero when an initial address is loaded it is possible to have only one count-enable signal. The LSB being zero always prevents the increment of the upper seven bits of the odd counter during the first cycle following an address load. The LSB of the odd counter can then be used to produce the delayed version of the even bank counter LSB by simply loading the odd bank counter LSB from Q02E on each cycle that the count enable is active. The upper seven bits of the odd counter still increment only at the end of cycles in which the odd counter LSB is one and the count enable is active.

This scheme simplifies the counter control logic somewhat and provides that a single control signal (Q02E) is used to manage all bank selection issues throughout the design.

The equation for the LSB of the counter is shown below. The remainder of the counter equations are shown in Figures 5-10 and 5-11:

$$
\begin{aligned}
Q02 \;:=\; & \overline{LD} \cdot \overline{ODD} \cdot A02 \\
+\; & \overline{LD} \cdot ODD \cdot \overline{A02} \\
+\; & \overline{LD} \cdot \overline{ODD} \cdot \overline{CNT} \cdot Q02 \\
+\; & \overline{LD} \cdot \overline{ODD} \cdot CNT \cdot \overline{Q02}
\end{aligned}
$$

PAL Definition Files

The PAL equations are given in Figures 5-6 to 5-11.

Note: All PAL equations in this handbook use the follwing convention:

- Where a PAL equation uses a colon followed by an equals sign (:=), the equation result is REGISTERED (i.e. registered PAL outputs are used).
- Where a PAL equation uses only an equals sign (=), the equation signals are COMBINATORIAL PAL outputs.
- The Device pin list is shown near the top of each figure as two lines of signal names. The names occur in pin order, numbered from left to right 1 through 20. The polarity of each name indicates the actual input or output signal polarity. Signals within the equations are shown as active high, e.g., where signal names in the pin list are: $\overline{A}$ B $\overline{C}$, the equation is C = A • $\overline{B}$; the inputs are: $\overline{A}$ = low, B = low; then the $\overline{C}$ output will be low.

Figure 5-6

AmPAL16L8B SRAM State Decoder—Interleaved Device U1

$\overline{IREQ}$ $\overline{DREQ}$ IREQT A31 A30 A29 A28 A27 $\overline{PIN169}$ GND
DREQT0 $\overline{IME}$ DREQT1 $\overline{ME}$ NC15 NC16 NC17 NC18 $\overline{DME}$ VCC

IME = IREQ • $\overline{IREQT}$ • $\overline{A31}$ • $\overline{A30}$ • $\overline{A29}$ • $\overline{A28}$ • $\overline{A27}$ • $\overline{PIN169}$

DME = DREQ • $\overline{DREQT0}$ • $\overline{DREQT1}$ • A31 • $\overline{A30}$ • $\overline{A29}$ • $\overline{A28}$ • $\overline{A27}$ • $\overline{PIN169}$

ME = IREQ • $\overline{IREQT}$ • $\overline{A31}$ • $\overline{A30}$ • $\overline{A29}$ • $\overline{A28}$ • $\overline{A27}$ • $\overline{PIN169}$
+ DREQ • $\overline{DREQT0}$ • $\overline{DREQT1}$ • A31 • $\overline{A30}$ • $\overline{A29}$ • $\overline{A28}$ • $\overline{A27}$ • $\overline{PIN169}$

Figure 5-7

AmPAL16R4D SRAM State Generator—Interleaved Instruction Section Device U3

CLK $\overline{IME}$ $\overline{DME}$ $\overline{IREQ}$ $\overline{DBACK}$ $\overline{IBREQ.D}$ NC07 $\overline{BINV.D}$ $\overline{Q02E}$ GND
$\overline{OE}$ $\overline{IOE0}$ $\overline{IOE1}$ $\overline{IBACK}$ $\overline{IBACK.D}$ $\overline{ILOAD.D}$ $\overline{ILOAD}$ $\overline{IRDY}$ $\overline{BINV}$ VCC

IBACK := $\overline{DBACK}$ • IME • BINV
+ $\overline{IEXIT}$ • IBACK

IBACK.D := IBACK

ILOAD := $\overline{DBACK}$ • $\overline{ILOAD}$ • $\overline{ILOAD.D}$ • IME • $\overline{BINV}$

ILOAD.D := ILOAD

IOE0 = Q02E • ILOAD.D
+ $\overline{BINV.D}$ • Q02E • IBREQ • IBACK.D • $\overline{ILOAD}$

IOE1 = $\overline{Q02E}$ • ILOAD.D
+ $\overline{BINV.D}$ • Q02E • IBREQ.D • IBACK.D • $\overline{ILOAD}$

IRDY = ILOAD.D
+ $\overline{BINV.D}$ • IBREQ.D • IBACK.D • $\overline{ILOAD}$

NOTE: The term IEXIT used in the IBACK equation is for clarity. Its true representation is as follows:

$$IEXIT = DME + IREQ \cdot \overline{IME}$$

Figure 5-8

AmPAL16R4D SRAM State Generator—Interleaved Data Section Device U4

CLK $\overline{IME}$ $\overline{DME}$ $\overline{DREQ}$ $\overline{IBACK}$ $\overline{DBREQ.D}$ RW $\overline{BINV.D}$ $\overline{Q02E}$ GND
$\overline{OE}$ $\overline{DOE0}$ $\overline{DOE1}$ $\overline{DBACK}$ $\overline{DBACK.D}$ $\overline{DLOAD.D}$ $\overline{DLOAD}$ $\overline{DRDY}$ $\overline{BINV}$ VCC

$$DBACK := \overline{IBACK} \cdot DME \cdot \overline{BINV} + \overline{DEXIT} \cdot DBACK$$

$$DBACK.D := DBACK$$

$$DLOAD := \overline{IBACK} \cdot \overline{DLOAD} \cdot \overline{DLOAD.D} \cdot DME \cdot \overline{BINV}$$

$$DLOAD.D := DLOAD$$

$$DOE0 = Q02E \cdot RW \cdot DLOAD.D + \overline{BINV.D} \cdot Q02E \cdot RW \cdot DBREQ.D \cdot DBACK.D \cdot \overline{DLOAD}$$

$$DOE1 = \overline{Q02E} \cdot RW \cdot DLOAD.D + \overline{BINV.D} \cdot \overline{Q02E} \cdot RW \cdot DBREQ.D \cdot DBACK.D \cdot \overline{DLOAD}$$

$$DRDY = \overline{RW} \cdot DLOAD + RW \cdot DLOAD.D + \overline{BINV.D} \cdot DBREQ.D \cdot DBACK.D \cdot \overline{DLOAD}$$

NOTE: The term DEXIT used in the DBACK equation is for clarity. Its true representation is as follows:

$$DEXIT = ME \cdot \overline{DBREQ.D} + DREQ \cdot \overline{DME}$$

Figure 5-9

AmPAL16L8B SRAM Counter Control—Interleaved Device U2

$\overline{IREQ}$ $\overline{DREQ}$ $\overline{DBACK}$ $\overline{IBACK}$ $\overline{DLOAD}$ $\overline{ILOAD}$ $\overline{DLOAD.D}$ $\overline{ILOAD.D}$ NC09 GND
$\overline{BINV.D}$ $\overline{LD}$ $\overline{IBREQ.D}$ $\overline{DBREQ.D}$ NC15 NC16 NC17 NC18 $\overline{CNT}$ VCC

$$LD = IREQ \cdot \overline{DBACK} \cdot \overline{ILOAD} \cdot \overline{ILOAD} + DREQ \cdot \overline{IBACK} \cdot \overline{DLOAD} \cdot \overline{DLOAD.D}$$

$$CNT = \overline{BINV.D} \cdot IBREQ.D \cdot IBACK + \overline{BINV.D} \cdot DBREQ.D \cdot DBACK + ILOAD + DLOAD$$

Figure 5-10

AmPAL16R4D SRAM Address Counter— Interleaved LSB ODD or Even Bank Devices U9, U11

CLK $\overline{CNT}$ $\overline{LD}$ A02 A03 A04 $\overline{ODD}$ $\overline{DRDY}$ RW GND
$\overline{OE}$ NC12 $\overline{DREGEN}$ $\overline{Q02}$ $\overline{Q03}$ $\overline{Q04}$ $\overline{WE}$ NC18 $\overline{COUT}$ VCC

$$
\begin{aligned}
Q02 &:= ODD \cdot \overline{LD} \cdot \overline{A02} \\
&+ \overline{ODD} \cdot LD \cdot A02 \\
&+ \overline{ODD} \cdot \overline{LD} \cdot \overline{CNT} \cdot Q02 \\
&+ \overline{ODD} \cdot \overline{LD} \cdot CNT \cdot \overline{Q02}
\end{aligned}
$$

$$
\begin{aligned}
Q03 &:= LD \cdot A03 \\
&+ \overline{LD} \cdot \overline{CNT} \cdot Q03 \\
&+ LD \cdot CNT \cdot \overline{Q02} \cdot \overline{Q03} \\
&+ \overline{LD} \cdot CNT \cdot \overline{Q02} \cdot Q03
\end{aligned}
$$

$$
\begin{aligned}
Q04 &:= LD \cdot A04 \\
&+ \overline{LD} \cdot \overline{CNT} \cdot Q04 \\
&+ \overline{LD} \cdot CNT \cdot \overline{Q02} \cdot Q03 \cdot \overline{Q04} \\
&+ \overline{LD} \cdot CNT \cdot \overline{Q02} \cdot Q04 \\
&+ \overline{LD} \cdot CNT \cdot \overline{Q03} \cdot Q04
\end{aligned}
$$

$$COUT = Q02 \cdot Q03 \cdot Q04$$

$$
\begin{aligned}
WE &:= ODD \cdot \overline{RW} \cdot DRDY \cdot \overline{A02} \\
&+ \overline{ODD} \cdot \overline{RW} \cdot DRDY \cdot \overline{Q02}
\end{aligned}
$$

$$
\begin{aligned}
DREGEN &= ODD \cdot \overline{A02} \\
&+ \overline{ODD} \cdot \overline{Q02}
\end{aligned}
$$

Figure 5-11

AmPAL16R6D SRAM Address Counter—Interleaved MSB Even or Odd Bank
Devices U8, U10

CLK $\overline{CNT}$ $\overline{LD}$ A05 A06 A07 A08 A09 $\overline{ME}$ GND
$\overline{OE}$ $\overline{CIN}$ $\overline{Q05}$ $\overline{Q06}$ $\overline{Q07}$ $\overline{Q08}$ $\overline{Q09}$ $\overline{CE}$ NC19 VCC

$$
\begin{aligned}
Q05 &:= \overline{LD} \cdot A05 \\
&+ \overline{LD} \cdot \overline{CNT} \cdot Q05 \\
&+ \overline{LD} \cdot CNT \cdot CIN \cdot \overline{Q05} \\
&+ \overline{LD} \cdot CNT \cdot \overline{CIN} \cdot Q05
\end{aligned}
$$

$$
\begin{aligned}
Q06 &:= \overline{LD} \cdot A06 \\
&+ \overline{LD} \cdot \overline{CNT} \cdot Q06 \\
&+ \overline{LD} \cdot CNT \cdot CIN \cdot Q05 \cdot \overline{Q06} \\
&+ \overline{LD} \cdot CNT \cdot \overline{CIN} \cdot Q06 \\
&+ \overline{LD} \cdot CNT \cdot \overline{Q05} \cdot Q06
\end{aligned}
$$

$$
\begin{aligned}
Q07 &:= \overline{LD} \cdot A07 \\
&+ \overline{LD} \cdot \overline{CNT} \cdot Q07 \\
&+ \overline{LD} \cdot CNT \cdot CIN \cdot Q05 \cdot Q06 \cdot \overline{Q07} \\
&+ \overline{LD} \cdot CNT \cdot \overline{CIN} \cdot Q07 \\
&+ \overline{LD} \cdot CNT \cdot \overline{Q05} \cdot Q07 \\
&+ \overline{LD} \cdot CNT \cdot \overline{Q06} \cdot Q07
\end{aligned}
$$

$$
\begin{aligned}
Q08 &:= \overline{LD} \cdot A08 \\
&+ \overline{LD} \cdot \overline{CNT} \cdot Q08 \\
&+ \overline{LD} \cdot CNT \cdot CIN \cdot Q05 \cdot Q06 \cdot Q07 \cdot \overline{Q08} \\
&+ \overline{LD} \cdot CNT \cdot \overline{CIN} \cdot Q08 \\
&+ \overline{LD} \cdot CNT \cdot \overline{Q05} \cdot Q08 \\
&+ \overline{LD} \cdot CNT \cdot \overline{Q06} \cdot Q08 \\
&+ \overline{LD} \cdot CNT \cdot \overline{Q07} \cdot Q08
\end{aligned}
$$

$$
\begin{aligned}
Q09 &:= \overline{LD} \cdot A09 \\
&+ \overline{LD} \cdot CNT \cdot Q09 \\
&+ \overline{LD} \cdot CNT \cdot CIN \cdot Q05 \cdot Q06 \cdot Q07 \cdot \overline{Q08} \cdot Q09 \\
&+ \overline{LD} \cdot CNT \cdot \overline{CIN} \cdot Q09 \\
&+ \overline{LD} \cdot CNT \cdot \overline{Q05} \cdot Q09 \\
&+ \overline{LD} \cdot CNT \cdot \overline{Q06} \cdot Q09 \\
&+ \overline{LD} \cdot CNT \cdot \overline{Q07} \cdot Q09 \\
&+ \overline{LD} \cdot CNT \cdot \overline{Q08} \cdot Q09
\end{aligned}
$$

$$
\begin{aligned}
CE &:= \overline{LD} \cdot ME \\
&+ \overline{LD} \cdot CE
\end{aligned}
$$

Intra-Cycle Timing

This memory architecture has two basic cycle timings. The first is a cycle used to decode the memory address and control signals from the processor. At the end of this decode cycle the address will be loaded into the address counter and the selected block of memory will begin a burst access in the next clock cycle. The second cycle timing is that of a burst access.

The first burst access time is the time required to access one of the memory banks. This time is designed to fit within two clock cycles. Thus, the initial burst access time will be two cycles.

The combination of a decode cycle followed by the first burst access time defines the three cycle initial access time. Each subsequent burst access requires one cycle due to the interleaving of two memory banks.

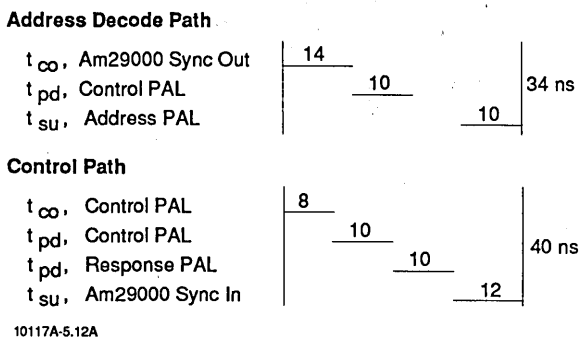
Within the decode cycle the address timing path is made up of:

- The Am29000 clock to address and control valid delay of 14 ns,
- Address decode logic PAL delay of 10 ns,
- And the set-up time of the address counter PAL, 10 ns .

Assuming D-speed PALs those times total 34 ns. See Figure 5-12. Also, within the decode cycle time is the control signal to response signal path. This delay path is made up of:

- Clock-to-output time of registers within the control logic state machine PAL, 8 ns;
- Propagation delay of the control logic PAL, 10 ns;
- Propagation delay of a logical OR gate on the response signals from each memory block, 10 ns;
- And control signal set-up time of the processor, 12 ns.

Figure 5-12



Interleaved Bank SRAM Memory Decode Cycle

Again assuming D-speed PALs, these times total 40 ns as shown in Figure 5-13.

Within the burst access cycle the address to data path timing is determined by:

- The clock-to-output time of the address counter, 8 ns for a D-speed PAL, plus added delay for heavy capacitive and inductive load. The added delay is determined by the method shown in Appendix A.

  The estimated delay is 5 ns. The total delay is then 8 ns, clock to output, plus 5 ns added delay for a total of 13 ns;

- Memory access time (55 ns),

- Data buffer delay (FCT244A = 4.3 ns),

- And the processor set-up time (6 ns).

Those delays total 78.3 ns worst case.

For the control signal-to-response signal path the time restrictions are the same in either the initial access or burst access cycles. The total delay is again 40 ns.

Inter-Cycle Timing

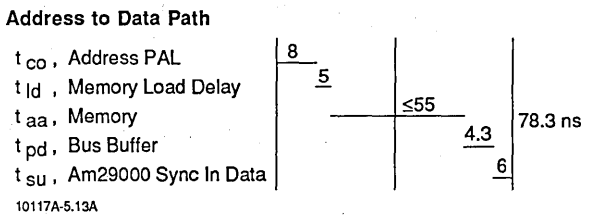
This section gives five examples of the cycle-by-cycle interaction between an Am29000 processor and the Medium Speed Interleaved Bank Static Memory system just defined in this chapter. Each timing diagram includes the Am29000 control and response signals as well as all the internal signals of the memory control logic.

Instruction Burst Read—Even Initial Address

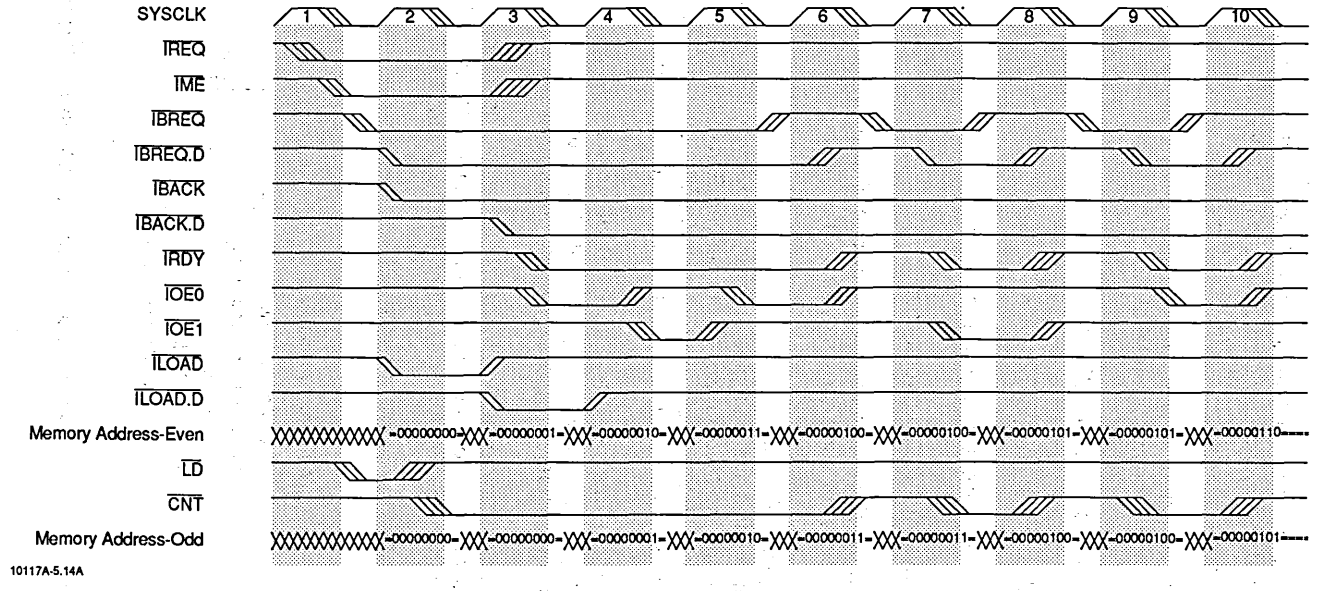
The first example is shown in Figure 5-14. It is a burst read of instruction memory with the initial address beginning at an even address.

In the first clock, cycle the Am29000 initiates a read operation by making $\overline{\text{IREQ}}$ and address active. The access will be a burst operation since the $\overline{\text{IBREQ}}$ signal also goes active late in the cycle. As a result the address is decoded to signal IME indicating that

Figure 5-13



Interleaved Bank SRAM Burst Cycle



These signals are inactive throughout sequence:
$\overline{\text{DLOAD}}$, $\overline{\text{DLOAD.D}}$, $\overline{\text{DREQ}}$, $\overline{\text{DME}}$, $\overline{\text{DBREQ}}$, $\overline{\text{DBREQ.D}}$, $\overline{\text{DBACK}}$, $\overline{\text{DBACK.D}}$, $\overline{\text{DRDY}}$,
$\overline{\text{DOE0}}$, $\overline{\text{DOE1}}$, R/$\overline{\text{W}}$, $\overline{\text{WE-EVEN}}$, $\overline{\text{WE-ODD}}$, $\overline{\text{DREGEN-EVEN}}$, $\overline{\text{DREGEN-ODD}}$

Instruction Burst Read—Even Initial Address

this instruction memory is selected. Also, the $\overline{LD}$ signal goes active causing the memory address counters and latches to capture the address on the bus at the next rising edge of SYSCLK.

In cycle two the address counters present the first address to the memory. The memory accesses the selected data so that it is on the bus in time for the Am29000 to receive it at the end of the third clock cycle. The registered value of $\overline{IBREQ}$ from cycle one is now available as the signal $\overline{IBREQ.D}$. This, in combination with $\overline{IBACK}$, causes the $\overline{CNT}$ signal to go active. When $\overline{CNT}$ goes active, it increments the address counter at the next rising edge of SYSCLK.

In cycles three, four and five, the first, second and third instruction words are read from memory. In each cycle the data is valid and the $\overline{IRDY}$ signal from the memory goes active. The $\overline{IOE0}$ and $\overline{IOE1}$ alternate being active as data from each bank is ready to be placed on the instruction bus. Since the initial address was even, the even bank output enable ($\overline{IOE0}$) goes active first. Note that the memory addresses shown are the output of the 8-bit address counters and only the upper seven bits serve as the lower address bits to the memory. The LSB serves only to control the counters so that the memory addresses increment on every other cycle that $\overline{CNT}$ is active. In cycle five, the $\overline{IBREQ}$ signal goes inactive signaling a suspension of the burst access.

In cycle six, the memory control circuits see the absence of $\overline{IBREQ.D}$ and immediately make $\overline{IRDY}$ inactive. $\overline{CNT}$ also goes inactive to hold the address value until the burst is resumed. The suspension of the burst is only one cycle long because $\overline{IBREQ}$ again goes active in this cycle.

In cycle seven, $\overline{IBREQ.D}$ is detected and $\overline{IRDY}$ immediately made active. $\overline{CNT}$ goes active again to continue the incrementing of address.

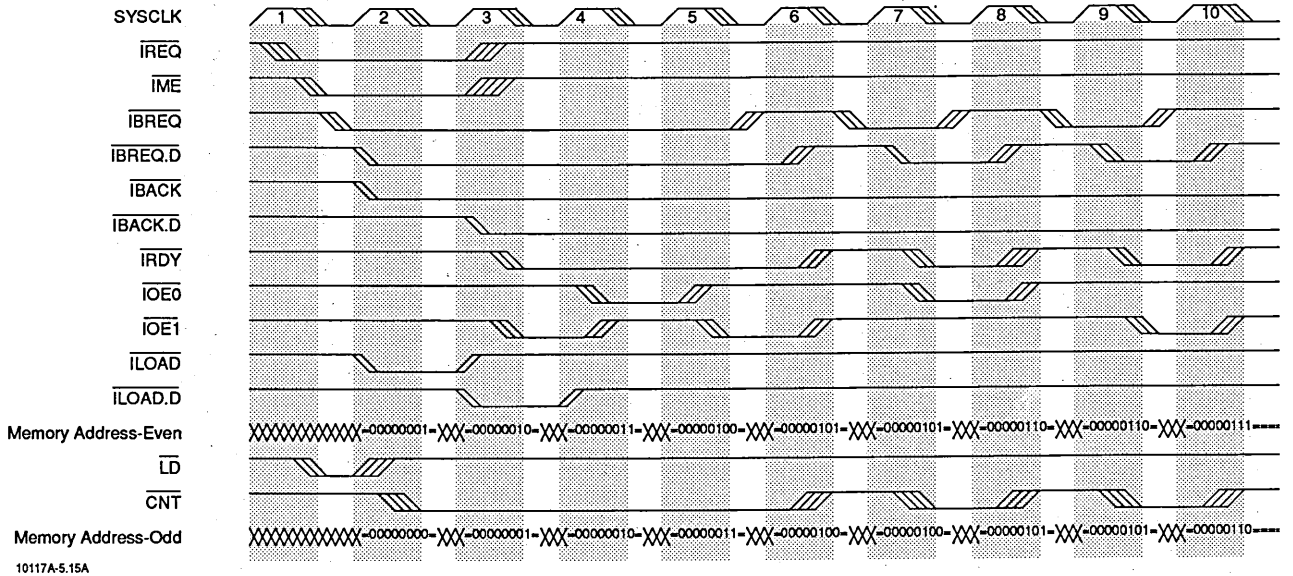
This sequence of $\overline{IBREQ}$ going active every other cycle is repeated through cycles seven, eight, and nine to show how the address counting and instruction output enables behave during repeated suspensions and resumptions.

Instruction Burst Read—Odd Initial Address

This example is the same as the last except that the initial address is odd. This is reflected in $\overline{IOE0}$ and $\overline{IOE1}$ going active in the reverse order from the last example. Also, the memory address for the even memory bank is incremented during cycle two so that the next even word following the initial odd address is accessed as shown in Figure 5-15.

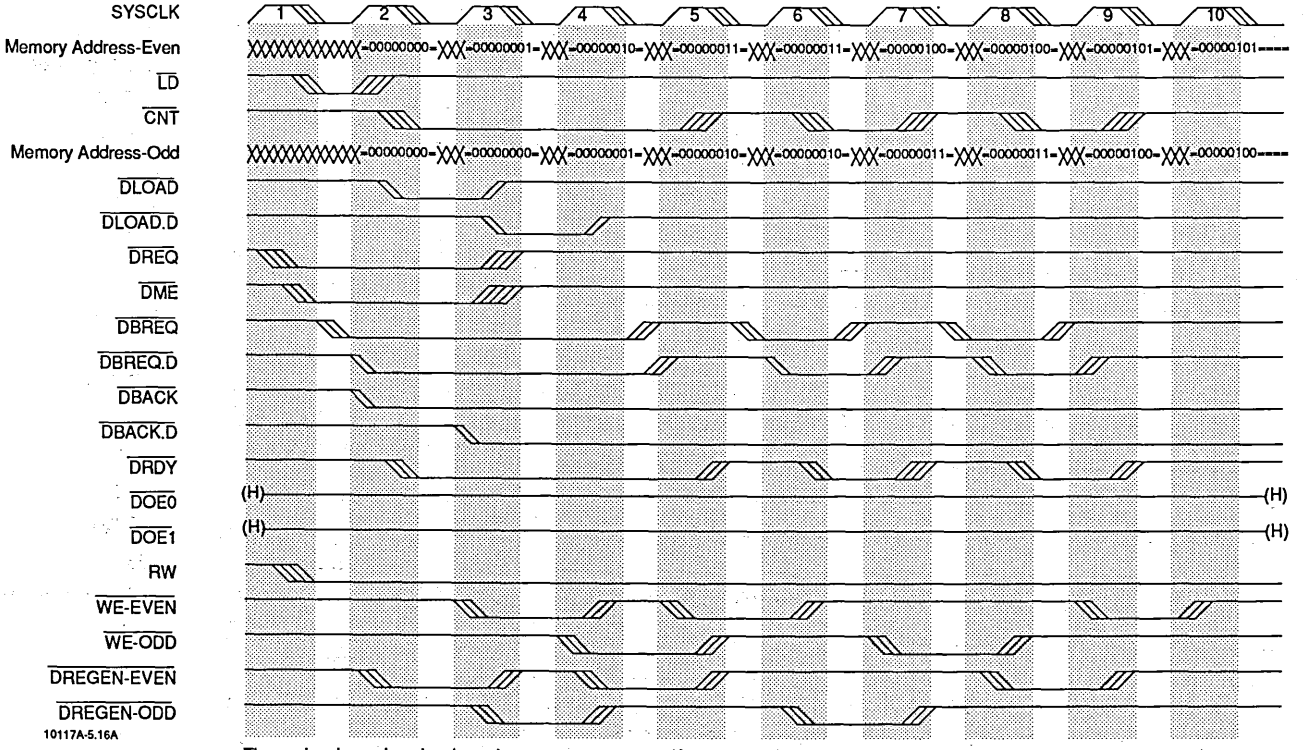
Instruction Burst Write

Figures 5-16 and 5-17 show examples very similar to that of the instruction access figures. The difference is that these accesses are burst-write operations to the instruction memory.



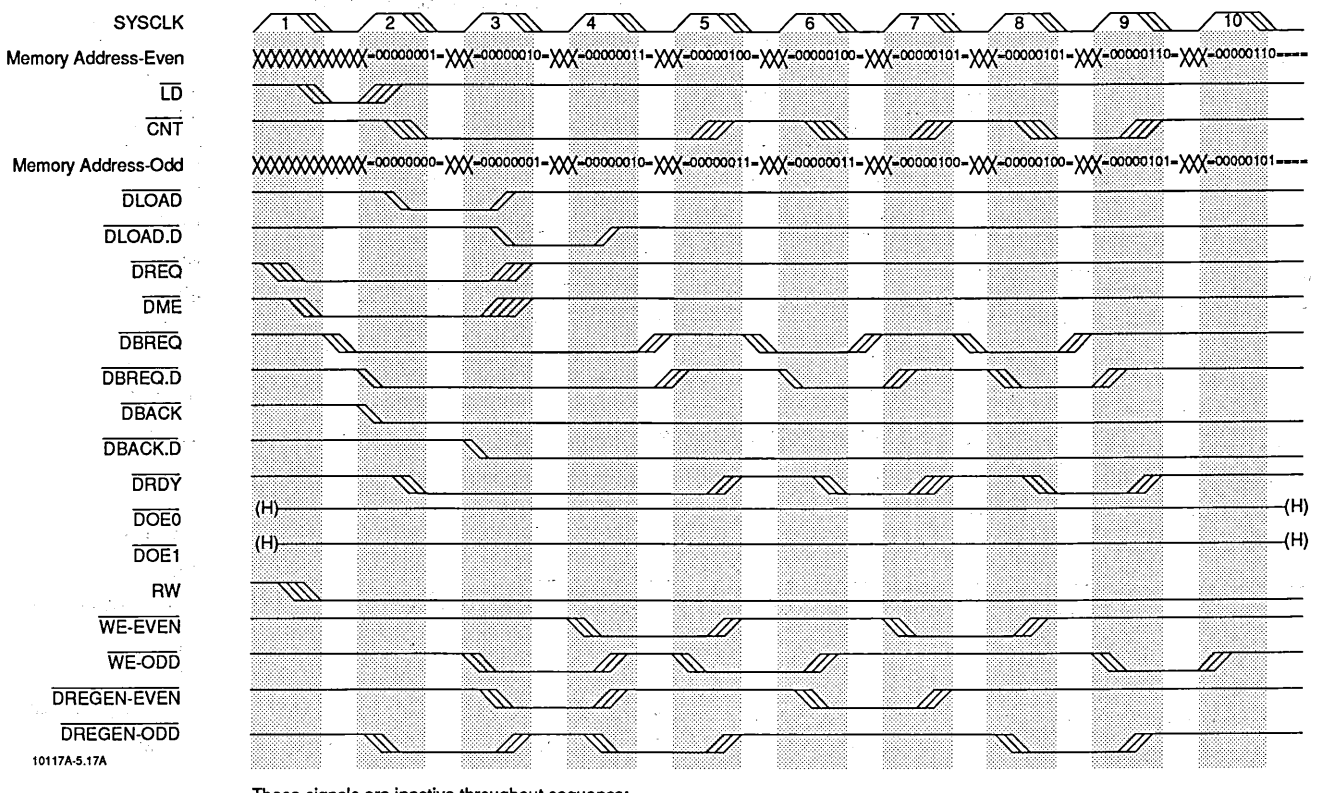
These signals are inactive throughout sequence:
$\overline{\text{DLOAD}}$, $\overline{\text{DLOAD.D}}$, $\overline{\text{DREQ}}$, $\overline{\text{DME}}$, $\overline{\text{DBREQ}}$, $\overline{\text{DBREQ.D}}$, $\overline{\text{DBACK}}$, $\overline{\text{DBACK.D}}$, $\overline{\text{DRDY}}$,
$\overline{\text{DOE0}}$, $\overline{\text{DOE1}}$, R/$\overline{\text{W}}$, $\overline{\text{WE-EVEN}}$, $\overline{\text{WE-ODD}}$, $\overline{\text{DREGEN-EVEN}}$, $\overline{\text{DREGEN-ODD}}$

Instruction Burst Read—Odd Initial Address



These signals are inactive throughout sequence:
$\overline{\text{IREQ}}$, $\overline{\text{IME}}$, $\overline{\text{IBREQ}}$, $\overline{\text{IBREQ.D}}$, $\overline{\text{IBACK}}$, $\overline{\text{IBACK.D}}$, $\overline{\text{IRDY}}$, $\overline{\text{IOE0}}$, $\overline{\text{IOE1}}$, $\overline{\text{ILOAD}}$, $\overline{\text{ILOAD.D}}$

Burst Write of Data—Even Initial Address



These signals are inactive throughout sequence:
$\overline{IREQ}$, $\overline{IME}$, $\overline{IBREQ}$, $\overline{IBREQ.D}$, $\overline{IBACK}$, $\overline{IBACK.D}$, $\overline{IRDY}$, $\overline{IOE0}$, $\overline{IOE1}$, $\overline{ILOAD}$, $\overline{ILOAD.D}$

Burst Write of Data—Odd Initial Address

Instruction Read Preempted by Data Read

Figure 5-18

The flow of control signals is the same as for the instruction accesses just described. The only differences are:

- That data words are now taken from the bus one cycle earlier than those times when they would have been supplied during a read;
- Data bus control and response signals are substituted for the equivalent instruction signals, e.g. $\overline{\text{DREQ}}$ goes active instead of $\overline{\text{IREQ}}$;
- $\overline{\text{DBREQ}}$ goes inactive in cycle 4 rather than cycle 5 as $\overline{\text{IBREQ}}$ did;
- The $\overline{\text{DREGEN}}$ signals enable the write data registers that take data to be written from the bus;
- And the $\overline{\text{WE}}$ signals are active.

Instruction Burst Preempt by Data Access

Figure 5-18 shows the interaction of a burst instruction access and a data read access addressed to the same block of memory.

The first two cycles occur as previously described for the instruction burst read.

In the third cycle, a data access is started by $\overline{\text{DREQ}}$ going active. The address is recognized as selecting this block of memory which is signaled by $\overline{\text{DME}}$ going active.

Since data accesses are given priority over instruction accesses, the instruction access must now be preempted. The memory control state machine exits the IACCESS state and returns to the IDLE state in cycle four. This will cause $\overline{\text{IBACK}}$ to go inactive thus preempting the instruction access. In cycle four the last word of the instruction burst is supplied by the memory. Also, the $\overline{\text{LD}}$ signal goes active to enable the address counters to capture the data access initial address.

In cycle five, $\overline{\text{IBREQ}}$ is removed from the bus.

In cycle six, the $\overline{\text{DREQ}}$ signal goes inactive as a result of the $\overline{\text{DBACK}}$ in cycle five, which in turn allows $\overline{\text{IREQ}}$ to go active to re-establish the preempted burst instruction access. The word resulting from the data access is presented to the bus along with $\overline{\text{DRDY}}$. Since the $\overline{\text{DBREQ}}$ signal has not been active, the data access in this case is a single word rather than a burst. The appearance of $\overline{\text{IREQ}}$, $\overline{\text{IME}}$ and the absence of $\overline{\text{DBREQ}}$ causes the control state machine to return to the IDLE state in the next cycle.

In cycle seven, the load signal goes active to capture the instruction address.

In cycle eight, the control state machine re-enters the IACCESS state with $\overline{\text{IBACK}}$ going active. Also, $\overline{\text{CNT}}$ goes active to increment the LSB of address for the instruction fetch. In cycle nine, the first word of instruction is placed on the bus with $\overline{\text{IRDY}}$. The instruction burst is thus re-established.

Parts List

The part list for the Am29000 Medium-speed Bank Interleaved Static RAM Interface is provided in Table 5-1.

Table 5-1

Am29000 Medium-speed Bank Interleaved Static RAM Interface Parts List

Item No.	Quantity	Device Description
U1-U2	2	AmPAL16L8D
U3-U4,U9,U11	4	AmPAL16R4D
U5	1	74F175
U6,U7	2	Am29823A
U8,U10	2	AmPAL16R6D
U12-U75	64	IDT7187S-55 or CY7C187-55
U76-U79,U84-U87	8	Am29825A
U80-U95,U88-U99	16	74FTC244A
	99 pkgs	

DATA MEMORY

As shown in Chapter 4, Figure 4-1, the instruction and data memories for the Am29000 are separate structures. The data memory can be an exact subset of the instruction-memory design. In fact the exact same design can be used by tying the instruction-related control signals to the inactive state. But, since the data memory is a subset, it is also possible to save a few chips by eliminating the instruction related control signals and rearranging the distribution of logic terms between PALs.

With reference to the instruction-memory design defined in this chapter, the following changes may be made to convert it to a data memory:

- All instruction-related inputs can be removed and all the affected equations simplified;
- U3, the instruction-state machine PAL, can therefore be removed entirely;
- The $\overline{\text{CNT}}$ signal can be moved to U4 and the $\overline{\text{LD}}$ signal can be moved to U1. Therefore U2 can be eliminated;
- The 74F175 from the instruction memory can also be used to supply the delayed control signals to the data memory, thus eliminating the need for U5;
- And finally, the instruction bus output buffers can be eliminated.

In total the design can be reduced by 11 chips. The details of the logic equation simplifications will be left as an exercise for the reader. All other aspects of the design are the same as for the instruction memory described in the previous section.

CHAPTER 6
Static Column Dram With Interleaved Banks

Overview 6-1
Dram Advantages and 29000 Dram Support 6-1
Memory Structure 6-1
Instruction Memory 6-2
Interface Logic Block Diagram 6-2
Memory Interface Logic Equations 6-6
PAL Definition Files 6-19
Intra-Cycle Timing 6-32
Inter-Cycle Timing 6-34
Parts List 6-35
Data Memory 6-35

Chapter 6

STATIC COLUMN DRAM WITH INTERLEAVED BANKS

OVERVIEW

DRAM Advantages and Am29000 DRAM Support

The SRAMs used in the last two designs provide the fastest initial access times. But, SRAMs are not very dense and therefore consume a large amount of board space for a given size memory system. Also, they tend to be expensive and consume a good deal of power for a given size memory.

Dynamic RAMs can provide far more memory at lower cost and power in the available board space than is possible with SRAM. The main penalty in using DRAMs is a loss of speed in the initial memory access time. Burst-access performance can be maintained by the use of bank interleaving and Static Column DRAMs (SCDRAM). Fortunately the Am29000 provides features that help compensate for a slower initial access time of system memory.

The Am29000 branch target cache stores the first four instructions from the 32 most recently accessed branch target addresses. So, when a branch instruction is executed, if the branch target address resides in the branch target cache, the first four instructions after the branch will come from the internal cache. At the same time, the address of the first instruction following those in the cache will be placed on the address bus. In effect, the first three cycles of the memory's initial access time will be hidden by the continued execution of instructions from the branch target cache. Note: three cycles are saved rather than four due to a cycle in which returning instructions must wait in the instruction prefetch buffer.

The Am29000 accesses virtually all its instructions in burst mode. This means that the initial access time of the system memory can be ammoritized over multiple cycles of a burst access. This again lowers the penalty of a slower initial access time.

The large register file of the Am29000 in effect provides a data cache for the most frequently used operands. This significantly reduces the number of times that memory needs to be accessed for data as compared with what is required by most competitive microprocessors. Also, the Am29000 load and store operations may be overlapped with the execution of other instructions, which again reduces the impact of a slower initial access-time memory system.

As a result, DRAMs can significantly increase the size of system memory, while also improving system performance-to-price ratio. The cost per bit of memory in the system drops dramatically while performance is reduced only slightly.

Memory Structure

The memory design described in this chapter is an extension of the memory designs from the previous chapters. There are also separate blocks of memory for instruction and data as was shown in Figure 4-1. Within each memory block, there are two banks of memory interleaved as odd and even words. For a description of interleaved memory architecture, see the overview section of the last chapter.

Each bank is 1M words deep with each word being 32-bits wide. The total for the instruction memory block is then 2M words (8M bytes). The same is true for the data memory.

SCDRAM memories with 85 ns access times are used for all memory banks. A non-sequential access requires one cycle for address decode and three cycles for the first word accessed. The low $\overline{RAS}$ access time allows a 4-cycle intial access time for the memory system; 100 ns $\overline{RAS}$ access time memories may be used if the intial access time is extended to five cycles. Essentially the burst access timing is the same as for the medium speed SRAM of the last chapter, each burst access is two cycles long. Overlapping the memory bank access time allows this longer access time to be hidden from the system viewpoint, except on the first word of a non-sequential access. The result is a memory that provides four cycle access time for the first word of a non-sequential access and single cycle access for subsequent words in a burst transfer.

The instruction memory bank has a read-only port for sending instructions to the Am29000 and a read/write port tied to the Am29000 data bus. This port provides access via the data bus for instruction loading and memory diagnostics. The data memory has a single read/write port connection to the Am29000 data bus.

INSTRUCTION MEMORY

Interface Logic Block Diagram

Refer to the block diagram in Figure 6-1.

The Memory

The memories are 1M x 1-bit SCDRAMs with separate data in and out lines. The access time is 85 ns. Thirty-two devices are required in each bank to form the 32-bit wide instruction word for the Am29000. These are shown as devices U21 through U85.

SCDRAMs are used to provide for access to sequential words within two clock cycles at 25 MHz and to simplify the required logic design. SCDRAMs have an advantage over standard DRAMs in that once a row is accessed, additional accesses within the same row can be done simply by changing the column address and waiting the access time delay of 45 ns. Standard DRAMs with page mode access ability require that the Column Address Strobe (CAS) be cycled for each new word accessed. Eliminating the need to cycle CAS simplifies the logic design and most SCDRAMs have faster access cycle times in static column mode than do equivalent DRAMs in page mode.

One additional "potential" advantage for either Page Mode or SCDRAMs is that the access time to words within an already selected row is much less than that required if the needed word lies in a different row. It is possible to reduce the initial access time of the memory whenever a non-sequential access begins in a row that is already being accessed. This is done by comparing all addresses from the processor with any currently active row address. If a match is identified the memory control logic can simply access the needed word rather than precharging the memory and giving a new row address. This can reduce the initial access time from five to three cycles (precharge time between row addresses adds one clock cycle to the basic 4-cycle initial access time).

This advantage is described above as "potential" because in the interest of keeping the design simple, this memory design does not implement the comparators or control logic needed to utilize the possible improvements from Page or Static Column modes (another exercise for the reader).

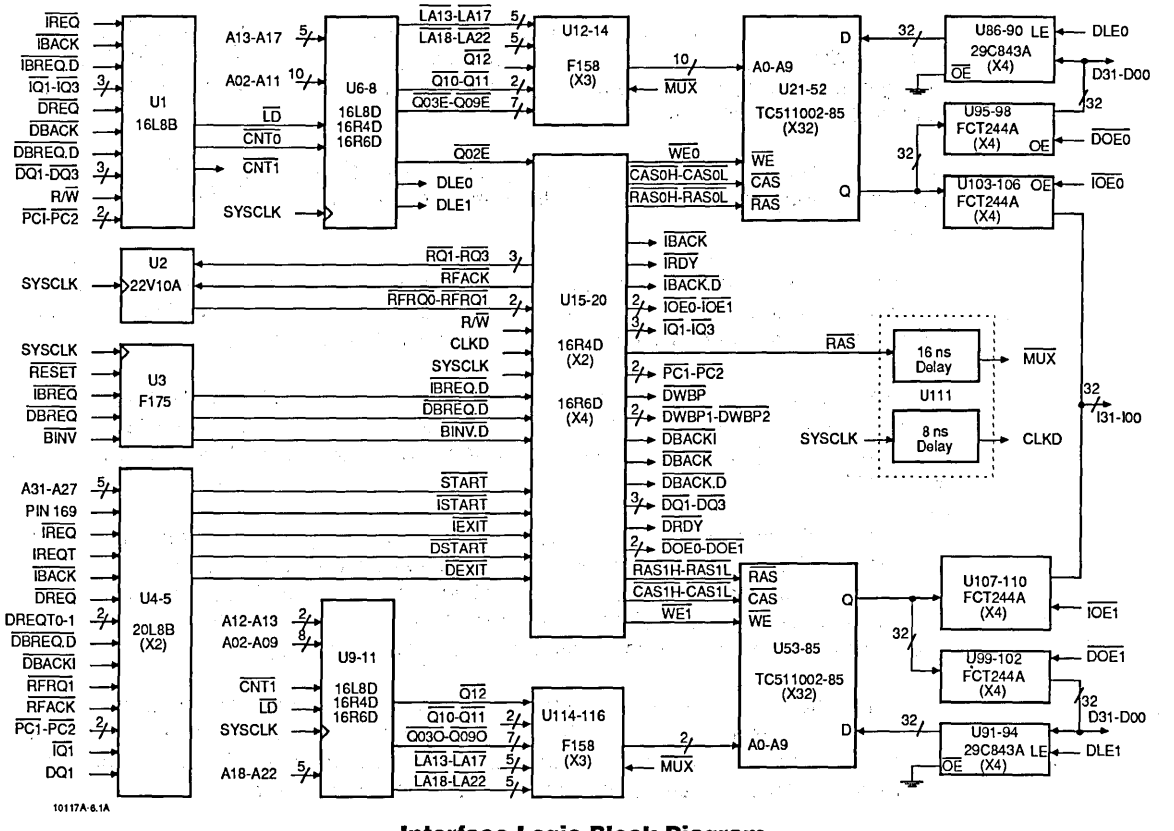
Data Bus Output Buffers

The memory data outputs are connected to the data bus lines via high-speed buffers. These buffers are required to isolate the memory outputs from the data bus whenever the memory is accessing instruction words. This isolation allows another data memory block to use the data lines at the same time that instructions are being fetched from this memory block. These are shown as devices U95 through U102.

Data Bus Input Latches

The memory data inputs are connected to the data bus lines via Am29C843A latches. These are shown as devices U86 through U94.

Figure 6-1



Interface Logic Block Diagram

Latches are used for the following reasons:

1. $\overline{\text{CHIP SELECT}}$ is used as the write-enable qualifier.

2. The $\overline{\text{CHIP SELECT}}$ signal is a registered output of the memory control logic and therefore its edge transitions occur one clock-to-output delay of a D-speed PAL after the system clock time (3 to 8 ns plus memory loading delay).

3. Write data to the memories must be valid at or before the falling edge of the $\overline{\text{CHIP SELECT}}$ signal .

4. Write data must be held valid for at least 20 ns after the falling edge of the $\overline{\text{CHIP SELECT}}$ signal .

5. The $\overline{\text{CHIP SELECT}}$ signal minimum pulse width is 25 ns.

6. The data output valid delay from the Am29000 processor is 18 ns.

Due to the above, it is not possible to write data directly from the processor data bus since the data may not be valid until after the falling edge of the $\overline{\text{CHIP SELECT}}$ signal during burst write cycles where new data is placed on the bus in each cycle (as a result of items 2, 3 and 6 above).

A register clocked by the rising edge of system clock would not have a clock-to-output delay fast enough to ensure meeting the data setup time to the $\overline{\text{CHIP SELECT}}$ signal . (Item 2)

A register clocked by the falling edge of system clock may not satisfy the required hold time relative to the $\overline{\text{CHIP SELECT}}$ signal, assuming a single register set is used and is simply clocked on each falling edge of system clock. (Items 2 and 4)

Dual register sets, one for each bank, clocked on every other falling edge of system clock could work. However, the worst-case timing margin for data setup time to the $\overline{\text{CHIP SELECT}}$ signal is very small, due to clock-gating logic plus clock-to-output time of a register.

Dual latch sets, one for each bank, latch enabled every other cycle by the active bank indicator ($\overline{\text{Q02E}}$) and a delayed system clock, will also work. Latches allow data to flow through to the memory inputs prior to the falling edge of the $\overline{\text{CHIP SELECT}}$ signal. The latches also hold the data valid for the required time after the $\overline{\text{CHIP SELECT}}$ signal. Both functions are accomplished with reasonable timing margins.

So with all the above in mind, data latches were chosen for use in the input data path to the memories. Using this data latching approach means that data is removed from the bus one cycle earlier than would be the case if simple buffers could be used; this makes a write operation one cycle faster than an equivalent read operation.

Instruction Bus Buffers

The memory data outputs are also connected to the instruction bus lines via buffers. These buffers serve to isolate the data outputs of this memory block from those outputs of other memory blocks which may also drive the instruction bus. Also the buffers serve to isolate the even and odd banks of this memory block from each other so that simultaneous data access can go on in each bank independently. These buffers are shown as devices U103 through U110.

Address Registers and Counters

To support burst accesses the lower seven address bits to each memory bank come from a loadable counter. An 8-bit counter is used to provide the address so that the least significant bit of the counter can be used to track which memory bank is connected to the data or instruction bus on each cycle. The upper seven bits of the counter are used as the least significant address bits to each memory bank.

Each 8-bit counter is built from one AmPAL16R4 and one AmPAL16R6 D-speed PALs. The counters for both banks are shown as devices U6, U7, U9, and U10. The D-speed PALs are used because their clock-to-output delay is significantly faster than standard MSI 8-bit counters. Also, the use of PALs allow additional functions to be integrated into the same packages used for the counter function.

The upper 14 bits of memory address need not come from a counter since the Am29000 will always output a new address when a 256 word boundary is crossed. The upper 14 bits of address are simply latched. A latch is used so that the address can flow through to the memories during the decode cycle and be setup before the falling edge of Row Address Strobe (RAS).

Address bits 10 through 12 are latched within the PALs which are used to implement the lower half of each bank address counter.

The upper 10 address bits (address bits 13 through 22) are latched in a pair of AmPAL16L8D PALs which also generate the needed latch-enable term. These are shown as devices U8 and U11.

A separate set of address counter logic is used to address each memory bank. This is done because when one bank is connected to the data or instruction bus, the other bank will be accessing the next word in sequence. This requires that the two banks have independently incremented addresses. The address for each bank will increment on different cycles.

Memory Address Multiplexers

The upper and lower 10 bits of memory address must be multiplexed into the address inputs of the memories. Discrete multiplexers are used rather than simply controlling the output enables of the address counters and latches to form a three-state multiplexer. This was done to provide tighter control over the timing of the multiplexer switching between sources. The input switching delay of the multiplexer is no worse than what the three-state enable delays would be if the three-state multiplexer approach was used, although they do add undesired delay in the burst access address to data timing in read operations. Multiplexing is done via 74F158 multiplexers shown as devices U12–U14 and U114–U116.

Registered Control Signals

As noted earlier, the timing of the Instruction Burst REQuest (IBREQ), Data Burst REQuest (DBERQ) , and Bus INValid (BINV) control signals require that they be registered by a low setup time register. A 74F175 register, U3 shown in Figure 6-1, is used as a low setup time register.

Interface Control Logic

This logic must generate the memory response signals, manage the loading and counting of memory addresses, generate RAS and the CHIP SELECT signals, control the data buffer output enables, and perform memory refresh. The logic functions needed for this require 10 PALs: two AmPAL20L8B, two AmPAL16R4D, four AmPAL16R6D, one AmPAL16L8B, and one AmPAL22V10A.

In Figure 6-1, device U1 an AmPAL16L8B produces the load and count enable signals for the address counters.

Device U2, an AmPAL22V10A provides a refresh interval counter and refresh request logic.

Devices U4 and U5 AmPAL20L8B PALs perform address decode for instruction and data accesses. Their outputs indicate when this memory block has been addressed, when an access is to begin, and when an access is terminated.

Devices U15 through U20, four AmPAL16R6D and two AmPAL16R4D PALs, form a complex state machine that controls the RAS, CHIP SELECT, output buffer enables, write enables, and memory response signals.

Response Signal Gating

As noted in the last chapter, the memory response signals from all system bus devices are logically ORed together before being returned to the Am29000 processor. An example of this circuitry was shown in Figure 4-3. These gates are not counted as part of the components within the memory design since they are shared by all the bus devices in the system and as such are part of the overhead needed in any Am29000 system.

Memory Interface Logic Equations

State Machine

The control logic for this memory can be thought of as a Mealy-type state machine in which the outputs are a function of the inputs and the present state of the machine. This structure is required since some of the output signals must be based on inputs which are not valid until the same cycle in which the outputs are required to effect control of the memory.

As shown in Figure 6-2, this state machine can be described as having 15 states. These states control the enabling of activity on the memory RAS, CHIP SELECT , burst acknowledge, output buffer enable and ready lines.

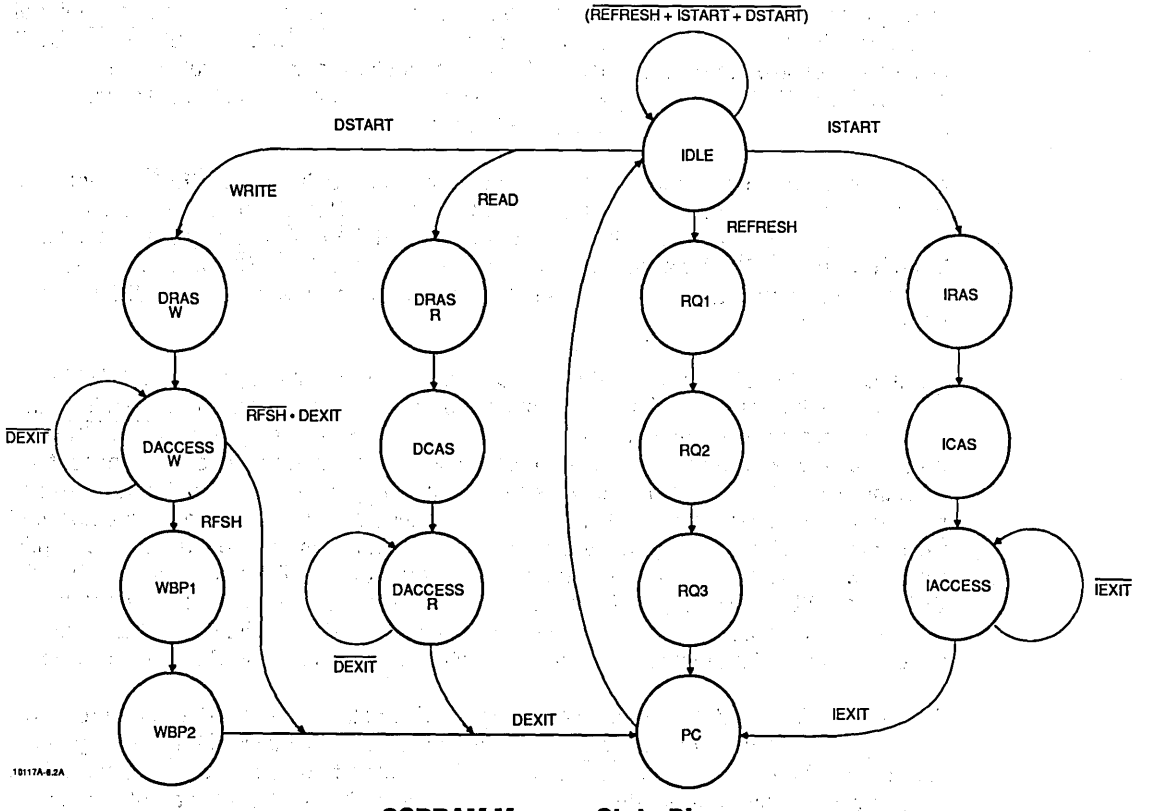
IDLE is the default state of the interface state machine. It is characterized by Instruction Burst ACKnowledge (IBACK) and Data Burst ACKnowledge (DBACK) both being inactive and no refresh activity in progress. This state serves as a way of identifying when the memory is not being accessed and could be placed into a low power mode. This

state also serves as a precharge cycle for the memory when a transition is made between instruction, data, and refresh sequences. A transition to either the Instruction RAS (IRAS) or Data RAS (DRAS) states occurs when an address selecting this memory block is placed on the address bus. A transition to the Refresh Request 1 (RQ1) state occurs when a refresh request is active. Refresh will take priority over any pending instruction or data access request.

The IRAS state occurs during the first cycle of memory access following a new instruction address being presented on the address bus. During this state the instruction output buffer enables and Ready response lines are held inactive and the $\overline{\text{IBACK}}$ and $\overline{\text{RAS}}$ lines go active. The address latches are closed to hold the memory address. $\overline{\text{RAS}}$ is used as the input to a delay line whose output will switch the address mux to the column address after the row address hold time is satisfied. The transition to the Instruction Column Address Strobe (ICAS) state is unconditional.

During the ICAS state the memory $\overline{\text{CHIP SELECT}}$ signal goes active to start the first access cycle. Since the $\overline{\text{CHIP SELECT}}$ access time for the memories used is 45 ns, it will take two cycles to access the memory, propagate data through the data buffers, and meet the setup time of the processor. Therefore the transition to the Instruction ACCESS (IACCESS) state is unconditional.

Figure 6-2



SCDRAM Memory State Diagram

The IACCESS state is used during the third cycle of a new address access and during all subsequent burst access cycles, whether active or suspended. In this state the instruction output buffer enable and ready lines are allowed to be active as required by the active or suspended status of an instruction burst request. When a new instruction address appears on the bus, a transition to the PreCharge (PC) state will occur. Also, if a data address selecting this memory block appears there will be a transition to the PC state to force a preemption of the current instruction access. The same is true when a refresh request is pending. The state machine remains in the IACCESS state as the default if no other state transition condition appears.

During the PC state, both burst acknowledge signals will go inactive along with $\overline{RAS}$. The PC state will preempt any burst access and begin the $\overline{RAS}$ precharge required before any new row address is applied to the memory. The precharge period for the memory used is 80 ns so a second cycle of precharge will be done during the IDLE cycle which unconditionally follows the PC cycle. Another important use of the PC state is as a delay cycle in the transition between an active instruction burst access being preempted and the start of the preempting data access. The delay is needed to allow the completion of the final instruction access in the cycle that IBACK is deasserted and the instruction burst access is preempted.

There are two data access sequences, one for read, and another for write accesses.

During a read access the sequence is the same as for an instruction access except that during the Data ACCESS (DACCESS) cycles the $\overline{DRDY}$ and Data Output Enable ($\overline{DOE}$) signals are allowed to be active instead of the instruction related control signals. The read DACCESS state is exited when a refresh is pending, or when a data access is suspended. The exit transition is to the PC state.

A data write access is a little different in that during a write, the $\overline{CHIP\ SELECT}$ signal is cycled to act as the write enable gate to the memories. This means that data to be written is latched from the bus in the cycle prior to $\overline{CHIP\ SELECT}$ being made active. Therefore the $\overline{DRDY}$ signal will go active one cycle before the $\overline{CHIP\ SELECT}$ goes active. This creates a problem that is solved by the Write Burst Preempt (WBP1 and WBP2) states.

It is important to note that when the $\overline{RFRQ1}$ signal is active, it will preempt a DACCESS and that a write operation is, in effect, pipelined. Data to be written is removed from the bus in the cycle before the write operation is enabled. So in the cycle that $\overline{DBACK}$ is made inactive to preempt the access, there may be one last data word being accepted from the bus. This word must be written in the following cycle. Also, at the point that a refresh request goes active, $\overline{DBACK}$ will still be active and will not be made inactive until the beginning of the next cycle. So, from the time that refresh request goes active until the last write cycle in memory is done, two cycles will occur. These cycles are labeled WBP1 and WBP2. During WBP1 the $\overline{DBACK}$ signal is made inactive to preempt the access, and data from the previous bus cycle is written. During WBP2 the last data word accepted from the bus is written, at which point the exit to the PC state is made.

Finally there is the refresh sequence. Once the IDLE state is reached and a refresh is pending, the refresh sequence will start as the highest priority task of the memory. In fact, during the IDLE cycle, $\overline{CHIP\ SELECT}$ will go active to setup for a $\overline{CAS}$-before-$\overline{RAS}$ refresh cycle. This type of refresh cycle makes use of the SCDRAM internal refresh counters to supply the refresh address. During RQ1, $\overline{RAS}$ is made active as during IRAS and DRAS cycles. The RQ2 and RQ3 cycles are used to supply two additional

wait states to make up the three cycles needed to satisfy the minimum RAS active time of 85 ns.

Logic Details—Signal by Signal
All signals are described in active high terms so that the design is a little easier to follow. The signals as implemented in the final Programmable Array Logic (PAL) outputs will often be active low as required by the actual circuit design. The actual PAL Definition files are included in Figures 6-3 through 6-18 at the end of this chapter.

NOTE: All PAL equations in this handbook use the following convention:

1. Where a PAL equation uses a colon followed by an equals sign (:=), the equation signals are REGISTERED PAL outputs.
2. Where a PAL equation uses only an equals sign (=), the equation signals are COMBINATORIAL PAL outputs

RFREQ (Refresh Request) — Funny thing about dynamic memories, they're very forgetful. They need to be completely refreshed every 4 ms, which translates into at least one row refreshed every 15.6 µs on average. To keep track of this time a counter is used. Once a refresh interval has passed, a latch is used to remember that a refresh is requested while the counter continues to count the next interval. Once the refresh has been performed, the latch is cleared.

The counter and refresh request latch is implemented in an AmPAL22V10A. Nine of the outputs form the counter, which is incremented by the system clock at 25 MHz. This gives up to 512 x 40 ns = 20.48 µs refresh periods. The synchronous preset term for all the registers is programmed to go active on a count value of 389 which will produce a refresh interval of 390 cycles x 40 ns = 15.6 µs. The one remaining output is used to implement the refresh request latch. That latch function (registered output) is also set by the synchronous preset term.

The equations for the counter are shown in Figure 6-3. Below are the preset and refresh latch equation:

$$\text{SYNCHRONOUS PRESET} = RFQ2 \cdot \overline{RFQ3} \cdot RFQ4 \cdot \overline{RFQ5} \cdot \overline{RFQ6} \cdot \overline{RFQ7} \cdot \overline{RFQ8} \cdot RFQ9 \cdot RFQ10$$

$$RFRQ0 := RFRQ0 \cdot (\overline{RFACK \cdot RQ1})$$

Refresh Sequence Equations — A refresh of the memory requires multiple clocks so that the minimum RAS active time of 100 ns can be satisfied. To manage this the following equations are used.

RFACK — The Refresh Acknowledge (RFACK) is used to begin a refresh sequence and to clear the pending refresh request. A refresh may begin when the state machine has returned to the IDLE state indicated by IBACK and DBACKI being inactive. The DBACKI signal is an internal version of DBACK which is active until all data write cycles are completed. RFACK is held active until the end of the sequence, indicated by $\overline{RFRQ1} \cdot RQ3$.

$$RFACK := \overline{DBACKI} \cdot \overline{IBACK} \cdot RFRQ1 + RFACK \cdot (\overline{RFRQ1} \cdot \overline{RQ3})$$

RQ1, RQ2, RQ3 — The three cycles needed for a refresh are tracked by RQ1, RQ2, and RQ3. RQ1 will not go active until the cycle following the IDLE state. This is controlled by $\overline{RQ1} \cdot \overline{PC1} \cdot$ RFACK which is only true during IDLE. The RQ1 signal is held active for all three refresh cycles to provide a single signal to identify when a refresh is in progress. The RQ2 and RQ3 signals simply follow RQ1 with RQ3 signaling the last cycle of the refresh sequence.

$$RQ1 := \overline{RQ1} \cdot \overline{PC1} \cdot RFACK + RQ1 \cdot \overline{RQ3}$$

$$RQ2 := RQ1 \cdot \overline{RQ3}$$

$$RQ3 := RQ2 \cdot \overline{RQ3}$$

REXIT — The Refresh EXIT (REXIT) signal is used to switch off the RAS signal at the end of a refresh sequence. RQ3 causes an exit and the $\overline{RFACK}$ term causes $\overline{REXIT}$ to be active outside of a refresh sequence to disable other equation terms using $\overline{REXIT}$ as a holding input during a refresh sequence.

$$REXIT = \overline{RFACK} + RQ3$$

IME — The use of the Instruction for ME (IME) signal is based on the assumption that other blocks of instruction or data memory may be added later and that there may be valid addresses in address spaces other than instruction/data space.

This means that this memory will only respond with IBACK or DBACK active when this block has been selected by valid addresses in the instruction/data space. This requires that at least some of the more significant address lines above the address range of this memory block be monitored to determine when this memory block is addressed. Also, it means the Instruction Request Type (IREQT) and Pin 169 lines must be monitored to determine that an address is valid and lies in the instruction/data space. Further, when a refresh request is pending the memory will not recognize its address. This will ensure refresh has the highest priority during the IDLE state.

IME is the indication that the address of this memory block is present on the upper address lines, an instruction request is active, Pin 169 is inactive (test hardware has not taken control), no refresh is pending, and instruction/data address space is indicated. In other words this memory block is receiving a valid instruction access request. This example design will assume that the address of this memory block is equal to $\overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27}$. The equation for this signal is:

$$IME = IREQ \cdot \overline{IREQT} \cdot \overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27} \cdot \overline{Pin169} \cdot \overline{RFRQ1}$$

Note that IME is not directly implemented as a PAL output in this design. The terms are used in the generation of the ISTART and IEXIT terms.

DME — The Data ME (DME) signal is the indication that the address of this memory block is present on the upper address lines, a data request is active, Pin 169 is inactive, refresh is not active, and instruction/data address space is indicated. In other words this memory block is receiving a valid data access request. This example design will assume that the address of this memory block is equal to $A31 \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27}$. Note that for instruction accesses the memory address for this block had A31 = zero

where the data accesses to this block are valid for A31 = one. This allows instruction memory for instruction accesses to be located at address zero while having the window for data bus access to the instruction memory located at a different base address. This allows the separate data memory block used in this design to have its base address also at zero. Thus both the instruction and data memories are located at address zero in their respective address spaces.

The equation for this signal is:

$$DME = DREQ \cdot \overline{DREQT0} \cdot \overline{DREQT1} \cdot A31 \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27} \cdot \overline{Pin169} \cdot \overline{REFRQ1}$$

As with IME this term is not directly implemented.

ISTART — The Instruction START (ISTART) signal causes the transition from IDLE to IRAS states. It is valid only in the IDLE or IACESS state with no refresh sequence starting, identified by not being in any other state via $\overline{DBACKI} \cdot \overline{RFACK} \cdot \overline{PC1}$. So when in the IDLE or IACESS state and IME is active, ISTART is active.

$$ISTART = \overline{DBACKI} \cdot \overline{RFACK} \cdot \overline{PC1} \cdot IME$$

DSTART — The Data START (DSTART) signal is similar to ISTART except with DME as the qualifier.

$$DSTART = \overline{IBACK} \cdot \overline{RFACK} \cdot \overline{PC1} \cdot DME$$

START — The START signal is used to restart RAS following precharge when there is still an active access in progress. This condition occurs when an instruction or data access is suspended and a new instruction or data access is started. In that situation the memory must be precharged before the new address is presented along with RAS. During this PC time the appropriate burst acknowledge signal is held active so as not to preempt the new access.

$$START = \overline{PC1} \cdot PC2 \cdot IBACK + \overline{PC1} \cdot PC2 \cdot DBACKI + \overline{PC1} \cdot PC2 \cdot RFACK$$

IEXIT — The Instruction EXIT (IEXIT) equation identifies when it is time to leave the IACCESS state. IEXIT is true if no instruction access is in progress. The $\overline{IBACK}$ input causes this so that other equations that use $\overline{IEXIT}$ to hold a term active will have that holding term made invalid when the IEXIT equation has no valid meaning i.e. when no instruction access is active.

IEXIT is also active when a data access, a refresh, or an instruction access not addressing this memory is pending. But, each of these conditions for IEXIT is restricted in one special situation.

When an instruction access is suspended and a new instruction access begins, IBACK is already active in the first cycle of the new instruction. The $\overline{IBACK}$ signal being active tells the processor that the address has been captured by the memory and a new address may be placed on the bus, perhaps one for a data access.

So, the memory is committed to accessing at least one instruction word for the new instruction access even though the address for the new access may change to begin yet another access.

Therefore any subsequent data access, refresh, or instruction access must be held off until at least one word of the new instruction access can be read. Note that this can take several cycles since, when a new instruction access starts after a previously suspended one, the memory must be precharged followed by the normal sequence of $\overline{RAS}$ and $\overline{CHIP\ SELECT}$ signals before the new instruction access is complete.

This restriction is applied by not allowing an exit until after the PC states and instruction access sequence are complete. These are represented by PC1, PC2, and IQ1 in the final equation.

As noted before, the DME term is a documentation convenience. In the IEXIT equation this term is directly expanded so that all inputs of DME are inputs to IEXIT. This eliminates a level of logic delay that would be needed if DME were implemented as the output of another PAL.

The IEXIT equation is:

$$\begin{aligned} IEXIT = {} & DME \cdot \overline{IQ1} \cdot \overline{PC1} \cdot \overline{PC2} \\ & + IREQ \cdot \overline{IQ1} \cdot \overline{PC1} \cdot \overline{PC2} \\ & + RFRQ1 \cdot \overline{IQ1} \cdot \overline{PC1} \cdot \overline{PC2} \\ & + \overline{IBACK} \end{aligned}$$

A data request to this memory block for instruction data space takes priority over an instruction fetch in progress. Also, if a new instruction fetch stream is started, this memory interface can return to the idle state.

DEXIT — The description of IEXIT applies directly to the Data EXIT (DEXIT) signal; the logic is the same with data respective signals substituted for instruction terms. The only difference is that the first exit term is a little different. A data access terminates when there is no further data burst requested. This approach is an optimization for use with the Am29000. It makes use of the fact that the Am29000 will never suspend a data transfer and burst data transfers will always go to completion in a single contiguous burst access. When a burst simple or piplelined access ends, the memory immediately goes into precharge so the memory will be ready for subsequent accessess with a minium initial access delay.

$$\begin{aligned} DEXIT = {} & \overline{DQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot \overline{DBREQ.D} \\ & + \overline{DQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot RFRQ1 \\ & + \overline{DBACKI} \end{aligned}$$

IBACK — The Instruction Burst ACKnowledge (IBACK) signal is applied to the Am29000 and is in effect the indication that the interface state machine is in an active or suspended instruction access. The equation is:

$$\begin{aligned} IBACK := {} & \overline{BINV} \cdot ISTART \\ & + \overline{IEXIT} \end{aligned}$$

The IBACK active state is entered when ISTART is active and the bus state is valid on the same cycle. Note here that the $\overline{BINV}$ input is used directly rather than the registered

form of $\overline{BINV.D}$. The timing of BINV is such that it will just meet the setup time of a D-speed PAL input. The BINV signal is required as the qualifier since ISTART is a combinatorial signal. IBACK will remain active until one of the IEXIT conditions is active or the bus goes invalid.

IBACK.D — The IBACK Delayed (IBACK.D) signal is simply a one cycle delayed version of IBACK.

IBACK.D := IBACK

It is used in the generation of IRDY, Instruction Output Enable (IOE)0, and IOE1.

DBACK — The Data Burst Acknowledge (DBACK) signal is applied to the Am29000 and is in effect the indication to the processor a burst access is allowed. DBACK is essentially the same as IBACK but with data respective terms substituted.

$$DBACK := \overline{BINV} \cdot DSTART + \overline{DEXIT}$$

DBACK.D — The DBACK Delayed (DBACK.D) signal is simply a one cycle delayed version of DBACK.

DBACK.D := DBACK

It is used in the generation of DRDY.

DBACKI — The DBACK Internal (DBACKI) signal is a memory interface internal version of DBACK to the Am29000 and is in effect the indication that the interface state machine is in an active or suspended data access. This signal will stay active during the DWBP states after DBACK has gone inactive to preempt a data burst write operation. The equation is:

$$DBACKI := \overline{BINV} \cdot DSTART + \overline{DEXIT} + DWBP$$

Instruction Initial Access States — Signals IQ1, IQ2, and IQ3 are used to control the state transitions from IRAS to IACCESS during the first instruction access. IQ1 goes active during IRAS and remains active for two additional cycles. IQ1 will go active when there is a valid ISTART or when there was a previously suspended instruction access and a new instruction access was accepted; indicated by $\overline{PC1} \cdot PC2 \cdot IBACK$. IQ2 and IQ3 follow IQ1 with IQ3 indicating the last cycle of the initial access.

$$IQ: = \overline{BINV} \cdot \overline{IQ1} \cdot ISTART \cdot \overline{IBACK} + \overline{IQ1} \cdot \overline{PC1} \cdot PC2 \cdot IBACK + IQ1 \cdot \overline{IQ3}$$

$$IQ2: = IQ1 \cdot \overline{IQ3}$$

$$IQ3 = IQ2 \cdot \overline{IQ3}$$

Data Initial Access States — These equations are the same as for IQ1–IQ3 with data respective inputs.

$$DQ1 := \overline{BINV} \cdot \overline{DQ1} \cdot DSTART \cdot \overline{DBACK} + \overline{DQ1} \cdot \overline{PC1} \cdot PC2 \cdot DBACK + DQ1 \cdot \overline{DQ3}$$

$$DQ2 := DQ1 \cdot \overline{DQ3}$$

$$DQ3 := DQ2 \cdot \overline{DQ3}$$

Data Write Burst Preempt States — When a data write operation is forced to preempt by a refresh request there are two additional write cycles that must be completed before PC is started. These states are tracked by the Data Write Burst Preempt (DWBP), DWBP1, and DWBP2 signals. DWBP starts the sequence when a data write is in progress, with burst request active, after the initial data write is completed, and a refresh is pending. DWBP1 and DWBP2 simply follow DWBP to indicate those states.

$$DWBP = DBACKI \cdot \overline{RW} \cdot DBREQ.D \cdot RFRQ1 \cdot \overline{DQ1} \cdot \overline{DWBP2}$$

$$DWBP1 := \overline{DWBP1} \cdot DWBP$$

$$DWBP2 := \overline{DWBP2} \cdot DWBP1$$

Precharge States — At the end of any access, the $\overline{RAS}$ lines must be made inactive to precharge internal memory buses before another access with a different row address may begin. Two cycles are needed and are indicated by the signals PC1 and PC2. PC1 is active during the PC state and PC2 is active during the first cycle of the IDLE state. PC1 goes active as the result of an IEXIT condition during instruction access, a DEXIT condition during data access following any Data Write Burst Preempt (DWBP) cycles, and at the end of a refresh sequence. PC2 simply follows PC1.

$$PC1 := \overline{PC1} \cdot IBACK \cdot IEXIT + \overline{PC1} \cdot DBACKI \cdot \overline{DWBP} \cdot DEXIT + \overline{PC1} \cdot RQ3$$

$$PC2 := PC1 \cdot \overline{PC2}$$

LD — The Load (LD) signal enables the lower address bit counters and the upper address bit latches to load a new address on the next rising edge of System CLock (SYSCLK). The equation is:

$$LD = \overline{IQ1} \cdot \overline{PC1} \cdot \overline{DBACKI} \cdot IREQ + \overline{DQ1} \cdot \overline{PC1} \cdot \overline{IBACK} \cdot DREQ$$

When an Instruction Request (IREQ) signal is active, load is prevented from being active while a data access is active or suspended. In other words, when the state machine is in a data access state a load that would result from an instruction request is suppressed. This prevents the changing of the address counter values until the data access ends. Similarly, for the case that Data Request (DREQ) signal is active, load is prevented when IBACK is active.

The LD signal is limited in length to one cycle by $\overline{IQ1}$ or $\overline{DQ1}$ during an initial access. It is limited to one cycle by $\overline{PC1}$ when a new access begins during a previously

suspended access. Limiting the LD signal to one cycle ensures that the correct address is captured and that LD does not interfere with the incrementing of the counters. The LD signal is combinatorial so that it can be active during the first cycle of a new instruction or data request.

Address Counters — There is one address counter for each bank of memory. Each is implemented with one AmPAL16R4D and one AmPAL16R6D device. The counter function is split across two PALs due to the number of product terms required to implement the upper bits of the counter. The lower half of the counter produces a carry out to the upper counter half. The equations for both bank counters are the same. These equations are shown in Figures 6-13 through 6-16.

The LSB bit of each counter is used as the means to control the timing of when the upper seven bits of each counter will increment. Note that only the upper seven bits of the counter are used as the low seven bits of address to the memory in a bank. This is because, with two interleaved banks, the maximum length burst access is split between the banks so each bank counter will never increment more than 128 times.

The upper bits of each counter increment on every cycle that the count signal is active and the LSB is also active. The only exception to the latter condition is during a bus invalid cycle where BINV signal is used to prevent counting when burst request may be invalid.

The value of the LSB bit in each counter is different in any given cycle, which causes the upper bits of the counters to increment on different cycles with regard to each other. In other words, the upper seven bits of the counters will be out of phase in terms of when they increment. This allows one bank of memory to start the access of the next word in sequence while the other bank completes the access of the current word.

Count Signals — There are two Count (CNT) signals defined in this design, CNT0 and CNT1, one for the even bank and one for the odd bank. This is because the even bank always increments one cycle earlier than the odd bank during the initial access of memory. Once the counting is started out of phase between banks, the bank counters are always incremented together to maintain the phase relationship. The CNT signals cause the address counters to increment on the next rising edge of SYSCLK.

The CNT0 controls the even bank counter. During either a data or instruction read operation, the first active cycle of CNT0 is during the DCAS or ICAS states indicated by the first cycle in which DQ2 or IQ2 is active. When the initial address selects an even word of memory, this first count cycle increments only the LSB of the even bank counter. This does not affect the memory address, but it makes the LSB high; this is used as an indication in other equations that data from the even bank is to be placed on the system bus. If the initial address selects an odd word, this first count cycle increments the whole even bank counter to point to the next even word in sequence after the initial odd word that will come from the odd memory bank. In this case, the LSB bit is low and indicates that the word, that is ready to be placed on the system bus, comes from the odd bank.

In the following cycle, IQ2 or DQ2 is still active, which ensures one more cycle of count. Any further count cycles come from burst-request signals being active during IACCESS or DACCESS states.

Note that in case a burst access is suspended and a new access of the same type begins, the address of the new access is loaded into the counter and the memory precharges in preparation for a new RAS cycle. During the precharge cycles, the incrementing of the counter must be inhibited by $\overline{PC1}$ and $\overline{PC2}$ so as not to change the address stored in the counter before the RAS and the CHIP SELECT signal cycles for the new access.

The CNT0 signal is handled differently during a data write in that any increment during IQ3 or DQ3 must be qualified by a burst request in the previous cycle. This is needed because in a write operation, the first Data Ready (DRDY) signal active cycle comes one cycle earlier than in a read operation.

$$
\begin{aligned}
CNT0 = {} & IBACK \cdot IQ2 \\
& + IBACK \cdot \overline{IQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot IBREQ.D \\
& + DBACKI \cdot RW \cdot DQ2 \\
& + DBACKI \cdot RW \cdot \overline{DQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot DBREQ.D \\
& + DBACKI \cdot \overline{RW} \cdot DQ2 \cdot \overline{DQ3} \\
& + DBACKI \cdot \overline{RW} \cdot DQ3 \cdot DBREQ.D \\
& + DBACKI \cdot \overline{RW} \cdot \overline{DQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot DBREQ.D
\end{aligned}
$$

The CNT1 signal controls the odd bank counter. This equation is essentially the same as CNT0 except that the first cycle in which CNT1 is active is always one later than it would have been in CNT0.

$$
\begin{aligned}
CNT1 = {} & IBACK \cdot IQ3 \\
& + IBACK \cdot \overline{IQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot IBREQ.D \\
& + DBACKI \cdot RW \cdot DQ3 \\
& + DBACKI \cdot RW \cdot \overline{DQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot DBREQ.D \\
& + DBACKI \cdot \overline{RW} \cdot DQ3 \cdot DBREQ.D \\
& + DBACKI \cdot \overline{RW} \cdot \overline{DQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot DBREQ.D
\end{aligned}
$$

IRDY — The Instruction Ready (IRDY) signal indicates that there is valid read data on the instruction bus.

$$
\begin{aligned}
IRDY = {} & IQ3 \\
& + \overline{BINV.D} \cdot \overline{IQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot IBREQ.D \cdot IBACK.D
\end{aligned}
$$

This memory design is always ready with data in the IQ3 cycle.

The memory is also ready when IBREQ is active with IBACK in the previous cycle. But, again the special situation of a suspended burst operation followed by a new access of the same type, is handled by adding $\overline{IQ1} \cdot \overline{PC1} \cdot \overline{PC2}$ to the equation. This prevents IRDY from going active until the new access has had time to precharge and readdress the memory. The $\overline{BINV.D}$ input is used to prevent false ready indications due to signals on the bus being invalid.

IBACK.D is required as a qualifier so that when an access is preempted the continued presence of IBREQ will not cause a false ready indication. The $\overline{BINV.D}$ signal is used to prevent false ready indications if the bus was invalid in the previous cycle. Note that situation can occur during a suspended access when the processor grants the bus to another bus master.

The reason that IRDY must be a combinatorial signal is that IBREQ comes very late in the previous cycle and must be registered. There is no time to perform logic on IBREQ in the previous cycle before SYSCLK rises. This means that the information that IBREQ was active in the last cycle is not available until the cycle in which IRDY should go active for a resumption of a suspended burst access.

IOE0 and IOE1 — The Instruction Output Enable (IOE) signals are used to control which bank is allowed to drive the instruction bus during each cycle. The signals use essentially the same logic as IRDY except that each signal is further qualified by the output of the LSB bit of the even bank counter (Q02E). This bit keeps track of which memory bank is ready to provide data to the instruction bus. The even bank is enabled when IRDY is active and the Q02E bit is active. The odd bank is enabled when IRDY is active and Q02E is inactive.

$$\begin{aligned} IOE0 &= Q02E \cdot IQ3 \\ &+ \overline{BINV.D} \cdot Q02E \cdot \overline{IQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot IBREQ.D \cdot IBACK.D \end{aligned}$$

$$\begin{aligned} IOE1 &= \overline{Q02E} \cdot IQ3 \\ &+ \overline{BINV.D} \cdot \overline{Q02E} \cdot \overline{IQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot IBREQ.D \cdot IBACK.D \end{aligned}$$

DRDY — The Data Ready (DRDY) is the equivalent of IRDY for data accesses and therefore uses the same equation with data respective terms substituted for instruction terms. The one additional change is that a term is added to cause DRDY to occur one cycle early during write operations. This is done because the data to be written is taken from the data bus into a latch before actually being stored in the memory. This maintains the same memory timing used during read operations but write data is removed from the bus one cycle earlier than when DRDY would normally go active during a data read operation.

$$\begin{aligned} DRDY &= RW \cdot DQ3 \\ &+ \overline{BINV.D} \cdot RW \cdot \overline{DQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot DBREQ.D \cdot DBACK.D \\ &+ \overline{RW} \cdot DQ2 \cdot \overline{DQ3} \\ &+ \overline{BINV.D} \cdot \overline{RW} \cdot DQ3 \cdot DBREQ.D \cdot DBACK.D \\ &+ \overline{BINV.D} \cdot \overline{RW} \cdot \overline{DQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot DBREQ.D \cdot DBACK.D \end{aligned}$$

DOE0 and DOE1 — The Data Output Enable (DOE) signals serve the same function for DRDY as the IOE0 and IOE1 signals serve for IRDY. Their signal descriptions are the same as for the IOE signals. The only difference is that the DOE signals are active only during read operations.

$$\begin{aligned} DOE0 &= RW \cdot Q02E \cdot DQ3 \\ &+ \overline{BINV.D} \cdot RW \cdot Q02E \cdot \overline{DQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot DBREQ.D \cdot DBACK.D \end{aligned}$$

$$\begin{aligned} DOE1 &= RW \cdot \overline{Q02E} \cdot DQ3 \\ &+ \overline{BINV.D} \cdot RW \cdot \overline{Q02E} \cdot \overline{DQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot DBREQ.D \cdot DBACK.D \end{aligned}$$

WE — Write Enable (WE) is a registered signal that goes active during the first DQ2 active cycle. It stays active throughout the data write operation. The CHIP SELECT signal is used in this design as the actual write gating signal. This was done to reduce the number of write signal outputs. Address, RAS and the CHIP SELECT lines have been duplicated in this design so that only half of each memory bank is driven by a

given output. This reduces the capacitive and inductive loading on each output so as to improve signal speed. Since the CHIP SELECT signal lines have already been doubled they are used as the write gate. The write enable line can thus be made active early in the cycle to have additional time to drive a heavier load.

$$WE0 := DBACKI \cdot \overline{RW}$$
$$WE1 := DBACKI \cdot \overline{RW}$$

Data Latch Enables — Data Latch Enable 0 and 1 (DLE0 and DLE1) are the signals that enable the write data latches on the D input of each memory bank to load new data.

The latches are enabled on every other cycle so that data is held valid long enough to satisfy the hold time after the CHIP SELECT signal goes active. The Q02E counter output is used to control which latch is enabled on a given cycle. A delayed version of the system clock is used to further place a window on the latch enable. This is an 8 ns delay generated in U111. Only during the high time of the delayed clock signal will the data be allowed through the latch. This is done to ensure that data is latched before the end of the system clock cycle when the processor begins changing the data value for the next write cycle. That could not be guaranteed by Q02E alone since it is a registered output with a clock-to-output delay. This is also the reason that the clock used is a delayed version of the system clock. This clock is delayed long enough to ensure that the worst-case clock-to-output time on Q02E has passed before enabling the latch. This ensures that no data is lost by having the latch enabled during the switching transition of Q02E as might happen if simply the system clock were used instead of the delayed clock.

$$\overline{DLE0} = Q02E \cdot CLKD$$
$$\overline{DLE1} = \overline{Q02E} \cdot CLKD$$

Row Address Strobes — There are five duplicated Row Address Strobe (RAS) lines. Four are used to drive the memories and one drives the delay line used to switch the address mux at the appropriate time. Multiple lines are used to split the capacitive and inductive load of the memory array to improve signal speed.

RAS is made active by a valid ISTART, DSTART or START condition. RAS is held active until an exit condition exists for the type of access in progress.

$$\begin{aligned} RAS0H := {} & \overline{BINV} \cdot \overline{RAS0H} \cdot \overline{START} \\ & + \overline{BINV} \cdot \overline{RAS0H} \cdot DSTART \\ & + \overline{BINV} \cdot \overline{RAS0H} \cdot START \\ & + RAS0H \cdot \overline{IEXIT} \\ & + RAS0H \cdot \overline{DEXIT} \\ & + RAS0H \cdot \overline{REXIT} \\ & + RAS0H \cdot DWBP \end{aligned}$$

Chip Select Lines — As with the RAS lines, the CHIP SELECT lines are duplicated to split the memory load.

The CHIP SELECT signal goes active in the cycle after RAS during instruction or data accesses. During a data write access the CHIP SELECT signal is enabled only when the appropriate bank is written with data. This is controlled with the Q02E line from the even bank address counter. CHIP SELECT signal during write is further gated by DRDY being active on the previous cycle which ensures that a write only occurs when

valid data was taken from the bus. Only in the case of a refresh sequence will CHIP SELECT signal be made active prior to RAS. This will initiate a CAS before RAS refresh cycle in the memories. In this case the CHIP SELECT signal is made active during the IDLE state.

$$
\begin{aligned}
\text{CAS0H} :=\ & \text{RAS} \cdot \text{IBACK} \\
& + \text{RAS} \cdot \text{DBACKI} \cdot \overline{\text{RW}} \\
& + \text{RAS} \cdot \text{DBACKI} \cdot \overline{\text{RW}} \cdot \overline{\text{Q02E}} \cdot \text{DRDY} \\
& + \overline{\text{RAS}} \cdot \overline{\text{IBACK}} \cdot \overline{\text{DBACKI}} \cdot \text{RFRQ1}
\end{aligned}
$$

$$
\begin{aligned}
\text{CAS1H} :=\ & \text{RAS} \cdot \text{IBACK} \\
& + \text{RAS} \cdot \text{DBACKI} \cdot \overline{\text{RW}} \\
& + \text{RAS} \cdot \text{DBACKI} \cdot \overline{\text{RW}} \cdot \text{Q02E} \cdot \text{DRDY} \\
& + \overline{\text{RAS}} \cdot \overline{\text{IBACK}} \cdot \overline{\text{DBACKI}} \cdot \text{RFRQ1}
\end{aligned}
$$

Upper Address Bits Latch — The address bits, 13 through 22, are latched by two D-speed PALs. All the bit equations are the same. Data is flow through when the Address Latch Enable (ALE) term is active and latched when ALE is inactive. An additional term ANDs the data input and output to prevent any possible loss of data during the ALE transition that might be caused by timing skew on ALE within the PAL (note the ALE "term" is a documentation convenience only; where ALE is shown, the actual logic definition of ALE is substituted). The ALE term is made active each cycle by a delayed version of the system clock. The delayed clock is used for the same reasons described for the DLE signals. During the initial access of an instruction or data word ALE is prevented from going active by the IQ1 and DQ1 terms. ALE is also held inactive during PC1 and PC2. This is done to preserve the address when a suspended access is followed by another access of the same type. In this case the address must be held while the memory is precharged and during the RAS cycle of the new access.

$$
\begin{aligned}
\text{LA22} =\ & \overline{\text{ALE}} \cdot \text{A22} \\
& + \overline{\text{ALE}} \cdot \text{LA22} \\
& + \text{A22} \cdot \text{LA22}
\end{aligned}
$$

$$
\text{ALE} = \overline{\text{IQ1}} \cdot \overline{\text{DQ1}} \cdot \overline{\text{PC1}} \cdot \overline{\text{PC2}} \cdot \text{CLKD}
$$

PAL Definition Files

The PAL definition files are provided in Figures 6-3 through 6-18.

NOTE: All PAL equations in this Application Note use the following convention:

1. Where a PAL equation uses a colon followed by an equals sign (:=), the equation signals are REGISTERED PAL outputs.

2. Where a PAL equation uses only an equals sign (=), the equation signals are COMBINATORIAL PAL outputs.

3. The device pin list is shown near the top of each figure as two lines of signal names. The names occur in pin order, numbered from left to right 1 through 20. The polarity of each name indicates the actual input or output signal polarity. Signals within the equations are shown as active high, e.g., where signal names in the pin list are: $\overline{\text{A}}$ B $\overline{\text{C}}$; the equation is C = A • $\overline{\text{B}}$; the inputs are $\overline{\text{A}}$ = low, B = low; then the $\overline{\text{C}}$ output will be low.

Figure 6-3

AmPAL22V10A SCDRAM Refresh Counter/Request Generator Device U2

CLK $\overline{RFACK}$ $\overline{RQ1}$ $\overline{RQ2}$ $\overline{RQ3}$ NC6 NC7 NC8 NC9 NC10 NC11 GND
NC13 $\overline{RFRQ0}$ $\overline{RFQ2}$ $\overline{RFQ3}$ $\overline{RFQ4}$ $\overline{RFQ5}$ $\overline{RFQ6}$ $\overline{RFQ7}$ $\overline{RFQ8}$ $\overline{RFQ10}$ $\overline{RFQ9}$ VCC

RFQ2 := $\overline{RFQ2}$

RFQ3 := RFQ2 • $\overline{RFQ3}$
+ $\overline{RFQ2}$ • RFQ3

RFQ4 := RFQ2 • RFQ3 • $\overline{RFQ4}$
+ $\overline{RFQ2}$ • RFQ4
+ RFQ3 • RFQ4

RFQ5 := RFQ2 • RFQ3 • RFQ4 • $\overline{RFQ5}$
+ $\overline{RFQ2}$ • RFQ5
+ $\overline{RFQ3}$ • RFQ5
+ $\overline{RFQ4}$ • RFQ5

RFQ6 := RFQ2 • RFQ3 • RFQ4 • RFQ5 • $\overline{RFQ6}$
+ $\overline{RFQ2}$ • RFQ6
+ $\overline{RFQ3}$ • RFQ6
+ $\overline{RFQ4}$ • RFQ6
+ $\overline{RFQ5}$ • RFQ6

RFQ7 := RFQ2 • RFQ3 • RFQ4 • RFQ5 • RFQ6 • $\overline{RFQ7}$
+ $\overline{RFQ2}$ • RFQ7
+ $\overline{RFQ3}$ • RFQ7
+ $\overline{RFQ4}$ • RFQ7
+ $\overline{RFQ5}$ • RFQ7
+ $\overline{RFQ6}$ • RFQ7

RFQ8 := RFQ2 • RFQ3 • RFQ4 • RFQ5 • RFQ6 • RFQ7 • $\overline{RFQ8}$
+ $\overline{RFQ2}$ • RFQ8
+ $\overline{RFQ3}$ • RFQ8
+ $\overline{RFQ4}$ • RFQ8
+ $\overline{RFQ5}$ • RFQ8
+ $\overline{RFQ6}$ • RFQ8
+ $\overline{RFQ7}$ • RFQ8

RFQ9 := RFQ2 • RFQ3 • RFQ4 • RFQ5 • RFQ6 • RFQ7 • RFQ8 • $\overline{RFQ9}$
+ $\overline{RFQ2}$ • RFQ9
+ $\overline{RFQ3}$ • RFQ9
+ $\overline{RFQ4}$ • RFQ9
+ $\overline{RFQ5}$ • RFQ9
+ RFQ6 • RFQ9
+ $\overline{RFQ7}$ • RFQ9
+ $\overline{RFQ8}$ • RFQ9

Figure 6-3 (Continued)

Device U2 (Continued)

RFQ10 := RFQ2 • RFQ3 • RFQ4 • RFQ5 • RFQ6 • RFQ7 • RFQ8 • RFQ9 • $\overline{\text{RFQ10}}$
+ $\overline{\text{RFQ2}}$ • RFQ10
+ $\overline{\text{RFQ3}}$ • RFQ10
+ $\overline{\text{RFQ4}}$ • RFQ10
+ $\overline{\text{RFQ5}}$ • RFQ10
+ $\overline{\text{RFQ6}}$ • RFQ10
+ RFQ7 • RFQ10
+ $\overline{\text{RFQ8}}$ • RFQ10
+ $\overline{\text{RFQ9}}$ • RFQ10

SYNCHRONOUS PRESET = RFQ2 • $\overline{\text{RFQ3}}$ • RFQ4 • $\overline{\text{RFQ5}}$ • $\overline{\text{RFQ6}}$ • $\overline{\text{RFQ7}}$ • $\overline{\text{RFQ8}}$ • RFQ9 • RFQ10

RFRQ1 := RFRQ1 • ($\overline{\text{RFACK}}$ • $\overline{\text{RQ1}}$)

Figure 6-4

AmPAL16R6D DRAM Refresh State Generator—Interleaved Device U15

CLK $\overline{\text{IBACK}}$ $\overline{\text{DBACKI}}$ $\overline{\text{RFRQ1}}$ $\overline{\text{DBREQ.D}}$ $\overline{\text{DQ1}}$ $\overline{\text{PC1}}$ RW NC9 GND
$\overline{\text{OE}}$ $\overline{\text{DWBP}}$ $\overline{\text{DWBP1}}$ $\overline{\text{DWBP2}}$ $\overline{\text{RFACK}}$ $\overline{\text{RQ1}}$ $\overline{\text{RQ2}}$ $\overline{\text{RQ3}}$ $\overline{\text{REXIT}}$ VCC

RFACK := $\overline{\text{DBACKI}}$ • $\overline{\text{IBACK}}$ • RFRQ1
+ RFACK • ($\overline{\text{RFRQ1}}$ • $\overline{\text{RQ3}}$)

RQ1 := $\overline{\text{RQ1}}$ • $\overline{\text{PC1}}$ • RFACK
+ RQ1 • $\overline{\text{RQ3}}$

RQ2 := RQ1 • $\overline{\text{RQ3}}$

RQ3 := RQ2 • $\overline{\text{RQ3}}$

REXIT = $\overline{\text{RFACK}}$
+ RQ3

DWBP = DBACKI • $\overline{\text{RW}}$ • DBREQ.D • RFRQ1 • $\overline{\text{DQ1}}$ • $\overline{\text{DWBP2}}$

DWBP1 := $\overline{\text{DWBP1}}$ • DWBP

DWBP2 := $\overline{\text{DWBP2}}$ • DWBP1

Figure 6-5

AmPAL16R6D DRAM Precharge State Generator—Interleaved Device U16

CLK $\overline{\text{ISTART}}$ $\overline{\text{DSTART}}$ $\overline{\text{IEXIT}}$ NC5 $\overline{\text{DEXIT}}$ NC7 $\overline{\text{RQ3}}$ $\overline{\text{BINV}}$ GND
$\overline{\text{OE}}$ $\overline{\text{DWBP}}$ $\overline{\text{IBACK}}$ $\overline{\text{DBACK}}$ $\overline{\text{DBACKI}}$ $\overline{\text{PC1}}$ $\overline{\text{PC2}}$ NC18 NC19 VCC

IBACK := $\overline{\text{BINV}}$ • ISTART
+ $\overline{\text{IEXIT}}$

DBACK := $\overline{\text{BINV}}$ • DSTART
+ $\overline{\text{DEXIT}}$

DBACKI := $\overline{\text{BINV}}$ • DSTART
+ $\overline{\text{DEXIT}}$
+ DWBP

PC1 := $\overline{\text{PC1}}$ • IBACK • IEXIT
+ $\overline{\text{PC1}}$ • DBACKI • $\overline{\text{DWBP}}$ • DEXIT
+ $\overline{\text{PC1}}$ • RQ3

PC2 := PC1 • $\overline{\text{PC2}}$

Figure 6-6

AmPAL20L8B DRAM State Decoder—Interleaved Device U4

$\overline{\text{RFRQ1}}$ $\overline{\text{IREQ}}$ DREQT0 DREQT1 IREQT $\overline{\text{PIN169}}$ A31 A30 A29 A28 A27 GND
RFACK $\overline{\text{DREQ}}$ $\overline{\text{ISTART}}$ $\overline{\text{IEXIT}}$ $\overline{\text{IBACK}}$ $\overline{\text{DBACKI}}$ $\overline{\text{PC1}}$ $\overline{\text{PC2}}$ $\overline{\text{START}}$ NC18 $\overline{\text{IQ1}}$ VCC

ISTART = $\overline{\text{DBACKI}}$ • $\overline{\text{RFACK}}$ • $\overline{\text{PC1}}$ • IME

START= $\overline{\text{PC1}}$ • PC2 • IBACK
+ $\overline{\text{PC1}}$ • PC2 • DBACKI
+ $\overline{\text{PC1}}$ • PC2 • RFACK

IEXIT = $\overline{\text{IQ1}}$ • $\overline{\text{PC1}}$ • $\overline{\text{PC2}}$ • DME
+ $\overline{\text{IQ1}}$ • $\overline{\text{PC1}}$ • $\overline{\text{PC2}}$ • IREQ
+ $\overline{\text{IQ1}}$ • $\overline{\text{PC1}}$ • $\overline{\text{PC2}}$ • RFRQ1
+ $\overline{\text{IBACK}}$

NOTE: In the above equations, IME and DME are used only for clarity. The actual input terms should be substituted when compiling this device.

DME = DREQ • $\overline{\text{DREQT0}}$ • $\overline{\text{DREQT1}}$ • A31 • $\overline{\text{A30}}$ • $\overline{\text{A29}}$ • $\overline{\text{A28}}$ • $\overline{\text{A27}}$ • $\overline{\text{PIN169}}$
• $\overline{\text{RFRQ1}}$

IME = IREQ • $\overline{\text{IREQT}}$ • $\overline{\text{A31}}$ • $\overline{\text{A30}}$ • $\overline{\text{A29}}$ • $\overline{\text{A28}}$ • $\overline{\text{A27}}$ • $\overline{\text{PIN169}}$ • $\overline{\text{RFRQ1}}$

Figure 6-7

AmPAL20L8B DRAM State Decoder—Interleaved
Device U5

$\overline{RFRQ1}$ $\overline{IREQ}$ DREQT0 DREQT1 $\overline{PIN169}$ IREQT A31 A30 A29 A28 A27 GND
$\overline{RFACK}$ $\overline{DREQ}$ $\overline{DSTART}$ $\overline{DEXIT}$ $\overline{DBREQ.D}$ $\overline{IBACK}$ $\overline{DBACKI}$ $\overline{PC1}$ $\overline{PC2}$ NC18 $\overline{DQ1}$ VCC

$$DSTART = \overline{IBACK} \cdot \overline{RFACK} \cdot \overline{PC1} \cdot DME$$

$$DEXIT = \overline{DQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot IME \cdot \overline{DBREQ.D} + \overline{DQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot RFRQ1 + \overline{DBACKI}$$

NOTE: In the above equations, IME and DME are used only for clarity. The actual input terms should be substituted when compiling this device.

$$IME = IREQ \cdot \overline{IREQT} \cdot \overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27} \cdot \overline{PIN169} \cdot \overline{RFRQ1}$$

$$DME = DREQ \cdot \overline{DREQT0} \cdot \overline{DREQT1} \cdot A31 \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27} \cdot \overline{PIN169} \cdot \overline{RFRQ1}$$

Figure 6-8

AmPAL16R4D DRAM Instruction State Generator—Interleaved
Device U17

CLK $\overline{IBACK}$ $\overline{ISTART}$ $\overline{IPC1}$ $\overline{IPC2}$ $\overline{IQ02E}$ $\overline{IBREQ.D}$ $\overline{IBINV.D}$ $\overline{IBINV}$ IGND
$\overline{OE}$ $\overline{IOE0}$ $\overline{IOE1}$ $\overline{IQ1}$ $\overline{IQ2}$ $\overline{IQ3}$ $\overline{IBACK.D}$ $\overline{IRDY}$ NC19 VCC

$$IBACK.D := IBACK$$

$$IQ1 := \overline{BINV} \cdot \overline{IQ1} \cdot ISTART \cdot \overline{IBACK} + \overline{IQ1} \cdot \overline{PC1} \cdot PC2 \cdot IBACK + IQ1 \cdot \overline{IQ3}$$

$$IQ2 := IQ1 \cdot \overline{IQ3}$$

$$IQ3 := IQ2 \cdot \overline{IQ3}$$

$$IRDY = IQ3 + \overline{BINV.D} \cdot \overline{IQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot IBREQ.D \cdot IBACK.D$$

$$IOE0 = Q02E \cdot IQ3 + \overline{BINV.D} \cdot Q02E \cdot \overline{IQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot IBREQ.D \cdot IBACK.D$$

$$IOE1 = \overline{Q02E} \cdot IQ3 + \overline{BINV.D} \cdot \overline{Q02E} \cdot \overline{IQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot IBREQ.D \cdot IBACK.D$$

Figure 6-9

AmPAL16R4D DRAM Data State Generator—Interleaved Device U18

```
CLK /DSTART /DBACK /PC1 /PC2 /Q02E /DBREQ.D /BINV RW GND
/OE /DOE0 /DOE1 /DQ1 /DQ2 /DQ3 /DBACK.D /DRDY /BINV.D VCC

DBACK.D  := DBACK

DQ1      := /BINV * /DQ1 * DSTART * /DBACK
          + /DQ1 * /PC1 * PC2 * DBACK
          + DQ1 * /DQ3

DQ2      := DQ1 * /DQ3

DQ3      := DQ2 * /DQ3

DRDY     = RW * DQ3
          + /BINV.D * RW * /DQ1 * /PC1 * /PC2 * DBREQ.D * DBACK.D
          + /RW * DQ2 * /DQ3
          + /BINV.D * /RW * DQ3 * DBREQ.D * DBACK.D
          + /BINV.D * /RW * /DQ1 * /PC1 * /PC2 * DBREQ.D * DBACK.D

DOE0     = RW * Q02E * DQ3
          + /BINV.D * RW * Q02E * /DQ1 * /PC1 * /PC2 * DBREQ.D * DBACK.D

DOE1     = RW * /Q02E * DQ3
          + /BINV.D * RW * /Q02E * /DQ1 * /PC1 * /PC2 * DBREQ.D * DBACK.D
```

Figure 6-10

AmPAL16R6D DRAM RAS Generator—Interleaved Device U19

CLK $\overline{\text{ISTART}}$ $\overline{\text{DSTART}}$ $\overline{\text{IEXIT}}$ NC5 $\overline{\text{DEXIT}}$ NC7 $\overline{\text{REXIT}}$ $\overline{\text{BINV}}$ GND
$\overline{\text{OE}}$ $\overline{\text{START}}$ $\overline{\text{RAS0H}}$ $\overline{\text{RAS0L}}$ $\overline{\text{RAS1H}}$ $\overline{\text{RAS1L}}$ $\overline{\text{RAS}}$ NC18 $\overline{\text{DWBP}}$ VCC

RAS0H := $\overline{\text{BINV}}$ • $\overline{\text{RAS0H}}$ • ISTART
+ $\overline{\text{BINV}}$ • $\overline{\text{RAS0H}}$ • DSTART
+ $\overline{\text{BINV}}$ • $\overline{\text{RAS0H}}$ • START
+ RAS0H • $\overline{\text{IEXIT}}$
+ RAS0H • $\overline{\text{DEXIT}}$
+ RAS0H • $\overline{\text{REXIT}}$
+ RAS0H • DWBP

RAS0L := $\overline{\text{BINV}}$ • $\overline{\text{RAS0L}}$ • ISTART
+ $\overline{\text{BINV}}$ • $\overline{\text{RAS0L}}$ • DSTART
+ $\overline{\text{BINV}}$ • $\overline{\text{RAS0L}}$ • START
+ RAS0L • $\overline{\text{IEXIT}}$
+ RAS0L • $\overline{\text{DEXIT}}$
+ RAS0L • $\overline{\text{REXIT}}$
+ RAS0L • DWBP

RAS1H := $\overline{\text{BINV}}$ • $\overline{\text{RAS1H}}$ • ISTART
+ $\overline{\text{BINV}}$ • $\overline{\text{RAS1H}}$ • DSTART
+ $\overline{\text{BINV}}$ • $\overline{\text{RAS1H}}$ • START
+ RAS1H • $\overline{\text{IEXIT}}$
+ RAS1H • $\overline{\text{DEXIT}}$
+ RAS1H • $\overline{\text{REXIT}}$
+ RAS1H • DWBP

RAS1L := $\overline{\text{BINV}}$ • $\overline{\text{RAS1L}}$ • ISTART
+ $\overline{\text{BINV}}$ • $\overline{\text{RAS1L}}$ • DSTART
+ $\overline{\text{BINV}}$ • $\overline{\text{RAS1L}}$ • START
+ RAS1L • $\overline{\text{IEXIT}}$
+ RAS1L • $\overline{\text{DEXIT}}$
+ RAS1L • $\overline{\text{REXIT}}$
+ RAS1L • DWBP

RAS := $\overline{\text{BINV}}$ • $\overline{\text{RAS}}$ • ISTART
+ $\overline{\text{BINV}}$ • $\overline{\text{RAS}}$ • DSTART
+ $\overline{\text{BINV}}$ • $\overline{\text{RAS}}$ • START
+ RAS • $\overline{\text{IEXIT}}$
+ RAS • $\overline{\text{DEXIT}}$
+ RAS • $\overline{\text{REXIT}}$
+ RAS • DWBP

Figure 6-11

AmPAL16R6D DRAM CAS Generator—Interleaved Device U20

CLK $\overline{Q02E}$ $\overline{IBACK}$ $\overline{DBACKI}$ $\overline{RFACK}$ $\overline{RAS}$ $\overline{RFRQ1}$ RW $\overline{DRDY}$ GND
$\overline{OE}$ NC12 $\overline{CAS0H}$ $\overline{CAS0L}$ $\overline{CAS1H}$ $\overline{CAS1L}$ $\overline{WE0}$ $\overline{WE1}$ NC19 VCC

CAS0H := RAS • IBACK
+ RAS • DBACKI • RW
+ RAS • DBACKI • $\overline{RW}$ • $\overline{Q02E}$ • DRDY
+ $\overline{RAS}$ • $\overline{IBACK}$ • $\overline{DBACKI}$ • RFRQ1

CAS0L := RAS • IBACK
+ RAS • DBACKI • RW
+ RAS • DBACKI • $\overline{RW}$ • $\overline{Q02E}$ • DRDY
+ $\overline{RAS}$ • $\overline{IBACK}$ • $\overline{DBACKI}$ • RFRQ1

CAS1H := RAS • IBACK
+ RAS • DBACKI • RW
+ RAS • DBACKI • $\overline{RW}$ • Q02E • DRDY
+ $\overline{RAS}$ • $\overline{IBACK}$ • $\overline{DBACKI}$ • RFRQ1

CAS1L := RAS • IBACK
+ RAS • DBACKI • RW
+ RAS • DBACKI • $\overline{RW}$ • Q02E • DRDY
+ $\overline{RAS}$ • $\overline{IBACK}$ • $\overline{DBACKI}$ • RFRQ1

WE0 := DBACKI • $\overline{RW}$

WE1 := DBACKI • $\overline{RW}$

Figure 6-12

AmPAL16L8B DRAM Counter Load—Interleaved Device U1

$\overline{IBREQ.D}$ $\overline{DBREQ.D}$ $\overline{IBACK}$ $\overline{DBACKI}$ $\overline{IQ1}$ $\overline{IQ2}$ $\overline{IQ3}$ $\overline{IREQ}$ $\overline{DREQ}$ GND
RW $\overline{CNT0}$ $\overline{LD}$ $\overline{DQ1}$ $\overline{DQ2}$ $\overline{DQ3}$ $\overline{PC1}$ $\overline{PC2}$ $\overline{CNT1}$ VCC

LD = $\overline{IQ1}$ • $\overline{PC1}$ • $\overline{DBACKI}$ • IREQ
+ $\overline{DQ1}$ • $\overline{PC1}$ • $\overline{IBACK}$ • DREQ

CNT0 = IBACK • IQ2
+ IBACK • $\overline{IQ1}$ • $\overline{PC1}$ • $\overline{PC2}$ • IBREQ.D
+ DBACKI • RW • DQ2
+ DBACKI • RW • $\overline{DQ1}$ • $\overline{PC1}$ • $\overline{PC2}$ • DBREQ.D
+ DBACKI • $\overline{RW}$ • DQ2 • $\overline{DQ3}$
+ DBACKI • $\overline{RW}$ • DQ3 • DBREQ.
+ DBACKI • $\overline{RW}$ • $\overline{DQ1}$ • $\overline{PC1}$ • $\overline{PC2}$ • DBREQ.D

CNT1 = IBACK • IQ3
+ IBACK • $\overline{IQ1}$ • $\overline{PC1}$ • $\overline{PC2}$ • IBREQ.D
+ DBACKI • RW • DQ3
+ DBACKI • RW • $\overline{DQ1}$ • $\overline{PC1}$ • $\overline{PC2}$ • DBREQ.D
+ DBACKI • $\overline{RW}$ • DQ3 • DBREQ.D
+ DBACKI • $\overline{RW}$ • $\overline{DQ1}$ • $\overline{PC1}$ • $\overline{PC2}$ • DBREQ.D

Figure 6-13

AmPAL16R4D DRAM Address Counter—Interleaved Section 0—Even Bank Device U6

CLK $\overline{CNT0}$ $\overline{LD}$ A02 A03 A04 A05 NC8 CLKD GND
$\overline{OE}$ DLE0 DLE1 $\overline{Q02E}$ $\overline{Q03E}$ $\overline{Q04E}$ $\overline{Q05E}$ $\overline{BINV}$ $\overline{COUT0}$ VCC

$$
\begin{aligned}
Q02E \;:= {} & LD \cdot A02 \cdot \overline{BINV} \\
& + \overline{LD} \cdot \overline{CNT0} \cdot Q02E \cdot \overline{BINV} \\
& + \overline{LD} \cdot CNT0 \cdot \overline{Q02E} \cdot \overline{BINV} \\
& + BINV \cdot Q02E
\end{aligned}
$$

$$
\begin{aligned}
Q03E \;:= {} & LD \cdot A03 \cdot \overline{BINV} \\
& + \overline{LD} \cdot \overline{CNT0} \cdot Q03E \cdot \overline{BINV} \\
& + \overline{LD} \cdot CNT0 \cdot Q02E \cdot \overline{Q03E} \cdot \overline{BINV} \\
& + \overline{LD} \cdot CNT0 \cdot \overline{Q02E} \cdot Q03E \cdot \overline{BINV} \\
& + BINV \cdot Q03E
\end{aligned}
$$

$$
\begin{aligned}
Q04E \;:= {} & LD \cdot A04 \cdot \overline{BINV} \\
& + \overline{LD} \cdot \overline{CNT0} \cdot Q04E \cdot \overline{BINV} \\
& + \overline{LD} \cdot CNT0 \cdot Q02E \cdot Q03E \cdot \overline{Q04E} \cdot \overline{BINV} \\
& + \overline{LD} \cdot CNT0 \cdot \overline{Q02E} \cdot Q04E \cdot \overline{BINV} \\
& + \overline{LD} \cdot CNT0 \cdot \overline{Q03E} \cdot Q04E \cdot \overline{BINV} \\
& + BINV \cdot Q04E
\end{aligned}
$$

$$
\begin{aligned}
Q05E \;:= {} & LD \cdot A05 \cdot \overline{BINV} \\
& + \overline{LD} \cdot CNT0 \cdot Q05E \cdot \overline{BINV} \\
& + \overline{LD} \cdot CNT0 \cdot Q02E \cdot Q03E \cdot Q04E \cdot \overline{Q05E} \cdot \overline{BINV} \\
& + \overline{LD} \cdot CNT0 \cdot \overline{Q02E} \cdot Q05E \cdot \overline{BINV} \\
& + \overline{LD} \cdot CNT0 \cdot \overline{Q03E} \cdot Q05E \cdot \overline{BINV} \\
& + \overline{LD} \cdot CNT0 \cdot \overline{Q04E} \cdot Q05E \cdot \overline{BINV} \\
& + BINV \cdot Q05E
\end{aligned}
$$

$$COUT0 = Q02E \cdot Q03E \cdot Q04E \cdot Q05E$$

$$\overline{DLE0} = Q02E + \overline{CLKD}$$

$$\overline{DLE1} = \overline{Q02E} + \overline{CLKD}$$

Figure 6-14

AmPAL16R6D DRAM Address Counter—Interleaved Section 1—Even Bank Device U7

CLK $\overline{CNT0}$ $\overline{LD}$ A06 A07 A08 A09 A10 A11 GND
$\overline{OE}$ $\overline{CIN0}$ $\overline{Q06E}$ $\overline{Q07E}$ $\overline{Q08E}$ $\overline{Q09E}$ $\overline{Q10}$ $\overline{Q11}$ $\overline{BINV}$ VCC

$$
\begin{aligned}
Q06E :=\ & LD \cdot A06 \cdot \overline{BINV} \\
&+ \overline{LD} \cdot \overline{CNT0} \cdot Q06E \cdot \overline{BINV} \\
&+ \overline{LD} \cdot CNT0 \cdot \overline{CIN0} \cdot \overline{Q06E} \cdot \overline{BINV} \\
&+ \overline{LD} \cdot CNT0 \cdot \overline{CIN0} \cdot Q06E \cdot \overline{BINV} \\
&+ BINV \cdot Q06E
\end{aligned}
$$

$$
\begin{aligned}
Q07E :=\ & LD \cdot A08 \cdot \overline{BINV} \\
&+ \overline{LD} \cdot \overline{CNT0} \cdot Q07E \cdot \overline{BINV} \\
&+ \overline{LD} \cdot CNT0 \cdot CIN0 \cdot Q06E \cdot \overline{Q07E} \cdot \overline{BINV} \\
&+ \overline{LD} \cdot CNT0 \cdot \overline{CIN0} \cdot Q07E \cdot \overline{BINV} \\
&+ \overline{LD} \cdot CNT0 \cdot \overline{Q06E} \cdot Q07E \cdot \overline{BINV} \\
&+ BINV \cdot Q07E
\end{aligned}
$$

$$
\begin{aligned}
Q08E :=\ & LD \cdot A09 \cdot \overline{BINV} \\
&+ \overline{LD} \cdot \overline{CNT0} \cdot Q08E \cdot \overline{BINV} \\
&+ \overline{LD} \cdot CNT0 \cdot CIN0 \cdot Q06E \cdot Q07E \cdot \overline{Q08E} \cdot \overline{BINV} \\
&+ \overline{LD} \cdot CNT0 \cdot \overline{CIN0} \cdot Q08E \cdot \overline{BINV} \\
&+ \overline{LD} \cdot CNT0 \cdot \overline{Q06E} \cdot Q08E \cdot \overline{BINV} \\
&+ \overline{LD} \cdot CNT0 \cdot \overline{Q07E} \cdot Q08E \cdot \overline{BINV} \\
&+ BINV \cdot Q08E
\end{aligned}
$$

$$
\begin{aligned}
Q09E :=\ & LD \cdot A09 \cdot \overline{BINV} \\
&+ \overline{LD} \cdot \overline{CNT0} \cdot Q09E \cdot \overline{BINV} \\
&+ \overline{LD} \cdot CNT0 \cdot CIN0 \cdot Q06E \cdot Q07E \cdot Q08E \cdot \overline{Q09E} \cdot \overline{BINV} \\
&+ \overline{LD} \cdot CNT0 \cdot \overline{CIN0} \cdot Q09E \cdot \overline{BINV} \\
&+ \overline{LD} \cdot CNT0 \cdot \overline{Q06E} \cdot Q09E \cdot \overline{BINV} \\
&+ \overline{LD} \cdot CNT0 \cdot \overline{Q07E} \cdot Q09E \cdot \overline{BINV} \\
&+ \overline{LD} \cdot CNT0 \cdot \overline{Q08E} \cdot Q09E \cdot \overline{BINV} \\
&+ BINV \cdot Q09E
\end{aligned}
$$

NOTE: Even bank counter holds Q10 and Q11, odd bank counter holds Q12 and Q13.

$$
\begin{aligned}
Q10 &:= LD \cdot A10 + \overline{LD} \cdot Q10 \\
Q11 &:= LD \cdot A11 + \overline{LD} \cdot Q11
\end{aligned}
$$

Figure 6-15

AmPAL16R4D DRAM Address Counter—Interleaved Section 0—Odd Bank Device U9

CLK $\overline{CNT1}$ $\overline{LD}$ A02 A03 A04 A05 NC8 NC9 GND
$\overline{OE}$ NC12 NC13 $\overline{Q02O}$ $\overline{Q03O}$ $\overline{Q04O}$ $\overline{Q05O}$ $\overline{BINV}$ $\overline{COUT1}$ VCC

Q02O := LD • A02 • $\overline{BINV}$
+ $\overline{LD}$ • $\overline{CNT1}$ • Q02O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{Q02O}$ • $\overline{BINV}$
+ BINV • Q02O

Q03O := LD • A03 • $\overline{BINV}$
+ $\overline{LD}$ • $\overline{CNT1}$ • Q03O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • Q02O • $\overline{Q03O}$ • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{Q02O}$ • Q03O • $\overline{BINV}$
+ BINV • Q03O

Q04O := LD • A04 • $\overline{BINV}$
+ $\overline{LD}$ • $\overline{CNT1}$ • Q04O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • Q02O • Q03O • $\overline{Q04O}$ • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{Q02O}$ • Q04O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{Q03O}$ • Q04O • $\overline{BINV}$
+ BINV • Q04O

Q05O := LD • A05 • $\overline{BINV}$
+ $\overline{LD}$ • $\overline{CNT1}$ • Q05O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • Q02O • Q03O • Q04O • $\overline{Q05O}$ • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{Q02O}$ • Q05O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{Q03O}$ • Q05O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{Q04O}$ • Q05O • $\overline{BINV}$
+ BINV • Q05O

COUT1 = Q02O • Q03O • Q04O • Q05O

Figure 6-16

AmPAL16R6D DRAM Address Counter—Interleaved Section 1—Odd Bank Device U16

CLK $\overline{CNT1}$ $\overline{LD}$ A06 A07 A08 A09 A12 A13 GND
$\overline{OE}$ $\overline{CIN1}$ $\overline{Q06O}$ $\overline{Q07O}$ $\overline{Q08O}$ $\overline{Q09O}$ $\overline{Q12}$ $\overline{Q13}$ $\overline{BINV}$ VCC

Q06O := LD • A06 • $\overline{BINV}$
+ $\overline{LD}$ • $\overline{CNT1}$ • Q06O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • CIN1 • $\overline{Q06O}$ • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{CIN1}$ • Q06O • $\overline{BINV}$
+ BINV • Q06O

Q07O := LD • A08 • $\overline{BINV}$
+ $\overline{LD}$ • $\overline{CNT1}$ • Q07O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • CIN1 • Q06O • $\overline{Q07O}$ • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{CIN1}$ • Q07O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{Q06O}$ • Q07O • $\overline{BINV}$
+ BINV • Q07O

Q08O := LD • A09 • $\overline{BINV}$
+ $\overline{LD}$ • $\overline{CNT1}$ • Q08O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • CIN1 • Q06O • Q07O • $\overline{Q08O}$ • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{CIN1}$ • Q08O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{Q06O}$ • Q08O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{Q07O}$ • Q08O • $\overline{BINV}$
+ BINV • Q08O

Q09O := LD • A09 • $\overline{BINV}$
+ $\overline{LD}$ • $\overline{CNT1}$ • Q09O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • CIN1 • Q06O • Q07O • Q08O • $\overline{Q09O}$ • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{CIN1}$ • Q09O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • Q06O • Q09O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{Q07O}$ • Q09O • $\overline{BINV}$
+ $\overline{LD}$ • CNT1 • $\overline{Q08O}$ • Q09O • $\overline{BINV}$
+ BINV • Q09O

NOTE: Even bank counter holds Q10, Q11 and odd bank counter holds Q12 , Q13

Q12 := $\overline{LD}$ • A12
+ $\overline{LD}$ • Q12

Q13 := $\overline{LD}$ • A13
+ $\overline{LD}$ • Q13

Figure 6-17

AmPAL16L8D DRAM Row Address Latch—Interleaved Device U8

CLKD $\overline{IQ1}$ A13 A14 A15 A16 A17 $\overline{PC1}$ $\overline{PC2}$ GND
$\overline{DQ1}$ NC12 $\overline{LA13}$ $\overline{LA14}$ $\overline{LA15}$ $\overline{LA16}$ $\overline{LA17}$ NC18 NC19 VCC

$$LA13 = \overline{ALE} \cdot A13 + ALE \cdot LA13 + A13 \cdot LA13$$

$$LA14 = \overline{ALE} \cdot A14 + ALE \cdot LA14 + A14 \cdot LA14$$

$$LA15 = \overline{ALE} \cdot A15 + ALE \cdot LA15 + A15 \cdot LA15$$

$$LA16 = \overline{ALE} \cdot A16 + ALE \cdot LA16 + A16 \cdot LA16$$

$$LA17 = \overline{ALE} \cdot A17 + ALE \cdot LA17 + A17 \cdot LA17$$

NOTE: The term ALE is used for clarity only. The true form of ALE is:

$$ALE = \overline{IQ1} \cdot \overline{DQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot CLKD$$

Figure 6-18

AmPAL16L8D DRAM Row Address Latch—Interleaved Device U11

CLKD $\overline{IQ1}$ A18 A19 A20 A21 A22 $\overline{PC1}$ $\overline{PC2}$ GND
$\overline{DQ1}$ NC12 $\overline{LA18}$ $\overline{LA19}$ $\overline{LA20}$ $\overline{LA21}$ $\overline{LA22}$ NC18 NC19 VCC

$$LA18 = \overline{ALE} \cdot A18 + ALE \cdot LA18 + A18 \cdot LA18$$

$$LA19 = \overline{ALE} \cdot A19 + ALE \cdot LA19 + A19 \cdot LA19$$

$$LA20 = \overline{ALE} \cdot A20 + ALE \cdot LA20 + A20 \cdot LA20$$

$$LA21 = \overline{ALE} \cdot A21 + ALE \cdot LA21 + A21 \cdot LA21$$

$$LA22 = \overline{ALE} \cdot A22 + ALE \cdot LA22 + A22 \cdot LA22$$

NOTE: The term ALE is used for clarity only. The true form of the ALE signal is:

$$ALE = \overline{IQ1} \cdot \overline{DQ1} \cdot \overline{PC1} \cdot \overline{PC2} \cdot CLKD$$

Intra-Cycle Timing

This memory architecture has three basic cycle timings. The first is a cycle used to decode the memory address and control signals from the processor. At the end of this decode cycle, the address is loaded into the address counter and the selected block of memory begins its initial access in the next clock cycle. Following the decode cycle is the row-address cycle in which the row address is made active at the beginning of the cycle, and in which the address multiplexer is later switched between the row address and the column address.

The third cycle timing is that of a burst access. The first burst access time is the time required to access one of the memory banks. This time is designed to fit within two clock cycles, so the initial burst-access time will be two cycles.

The combination of a decode cycle, followed by the row-address cycle, followed by the first burst-access time defines a 4-cycle initial access time.

After the initial access, all burst accesses use the 2-clock-cycle timing of the initial burst access. Because two memory banks are interleaved, the apparent access time from the viewpoint of the system bus is only one cycle per burst access following the initial access.

Decode Timing

Within the decode cycle the address timing path is made up of:

- The Am29000 clock to address and control valid delay of 14 ns,
- Address decode logic PAL delay of 10 ns, (devices, U4 and U5).
- And the setup time of the address counter PAL, 10 ns (devices, U6-U11).

Assuming D-speed PALs, those times total 34 ns, as shown in Figure 6-19.

Also, within the decode cycle time is the control signal to response signal path. In fact this timing path is present in every cycle in the sense that the memory response signals must be valid in every clock cycle. This delay path is made up of:

- Clock-to-output time of registers within the control logic state machine PAL, 8 ns;
- Propagation delay of the control logic PAL, 10 ns;
- Propagation delay of a logical OR gate on the response signals from each memory block, 10 ns;
- And control signal setup time of the processor, 12 ns.

Again assuming D-speed PALs, these times total 40 ns, as shown in Figure 6-19.

Row Address Timing

Within the row address cycle the RAS line goes low which initiates a time delay signal which later causes the address multiplexer to change from the row to the column address as shown in Figure 6-20.

The RAS delay path is made up of:

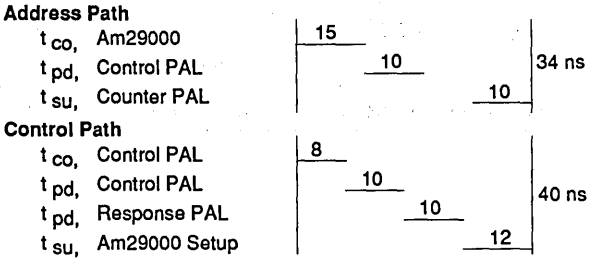
- Clock-to-output time of RAS signal registers within the control logic state machine PAL (8 ns) plus an added delay due to capacitive and inductive loading by the memory array of the PAL outputs. Since this load is in excess of standard data sheet test loads, the equations in appendix A are used to estimate the added delay. That delay estimate is 6.5 ns. This is added to the 8 ns (standard 50 pf load) delay of the RAS line for a total of 14.5 ns worst case.

The Address path is made up of:

- Clock to Output time of RAS output not loaded by memory array, 8 ns.
- Delay line time, 16 ns.
- Minimum and maximum switch time of the multiplexer, 4 ns to 9.5 ns.
- Memory load delay of 6.5 ns.

This works out to satisfy the 15 ns of required hold time of address after RAS goes active. Also the column address is settled by 40 ns into the cycle.

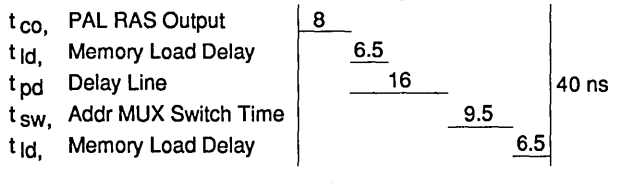
Figure 6-19



10117A-6.19

SCDRAM Interleaved Bank Memory Decode Cycle

Figure 6-20



10117A-6.20A

SCDRAM Interleaved Bank Memory RAS Cycle

Burst Timing

Within the burst access cycle the address to data path timing is determined by:

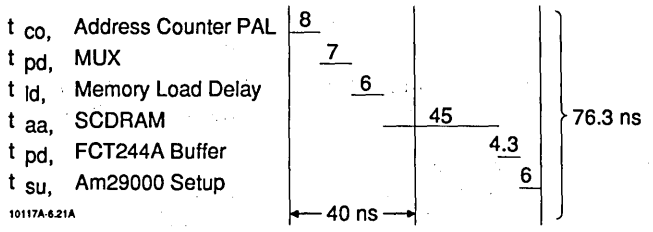
- The clock to output time of the address counter (8 ns for a D-speed, PAL)
- Propagation delay of multiplexer (7 ns) plus added delay for heavy capacitive and inductive load as determined in Appendix A. The added delay is estimated to be 6 ns.
- Memory access time in static column mode, 45 ns),
- Data buffer delay (FCT244A = 4.3 ns),
- And the processor set-up time (6 ns).

Those delays total 76.3 ns worst case as shown in Figure 6-21.

Inter-Cycle Timing

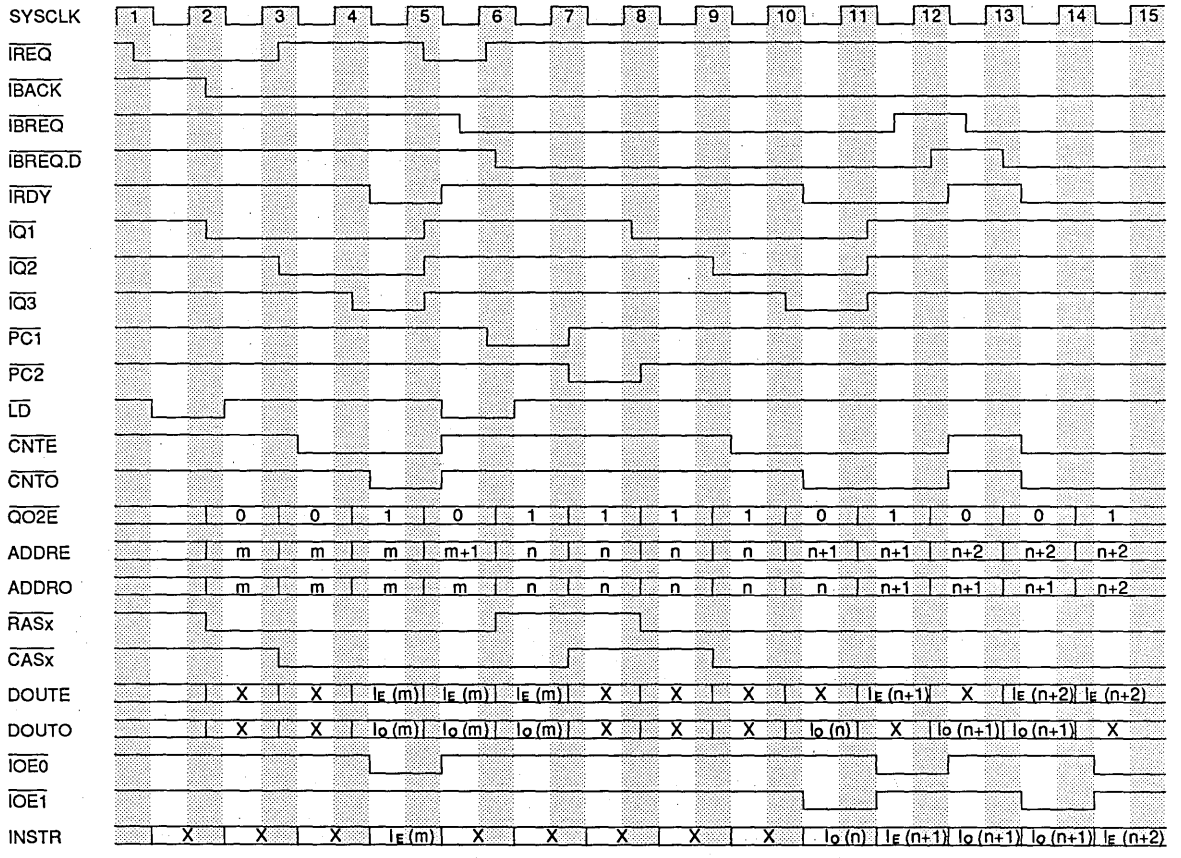
Inter-cycle timing for instruction, data read and data write cycles are provided in Figures 6-22 through 6-24.

Figure 6-21



SCDRAM Interleaved Bank Memory Burst Access

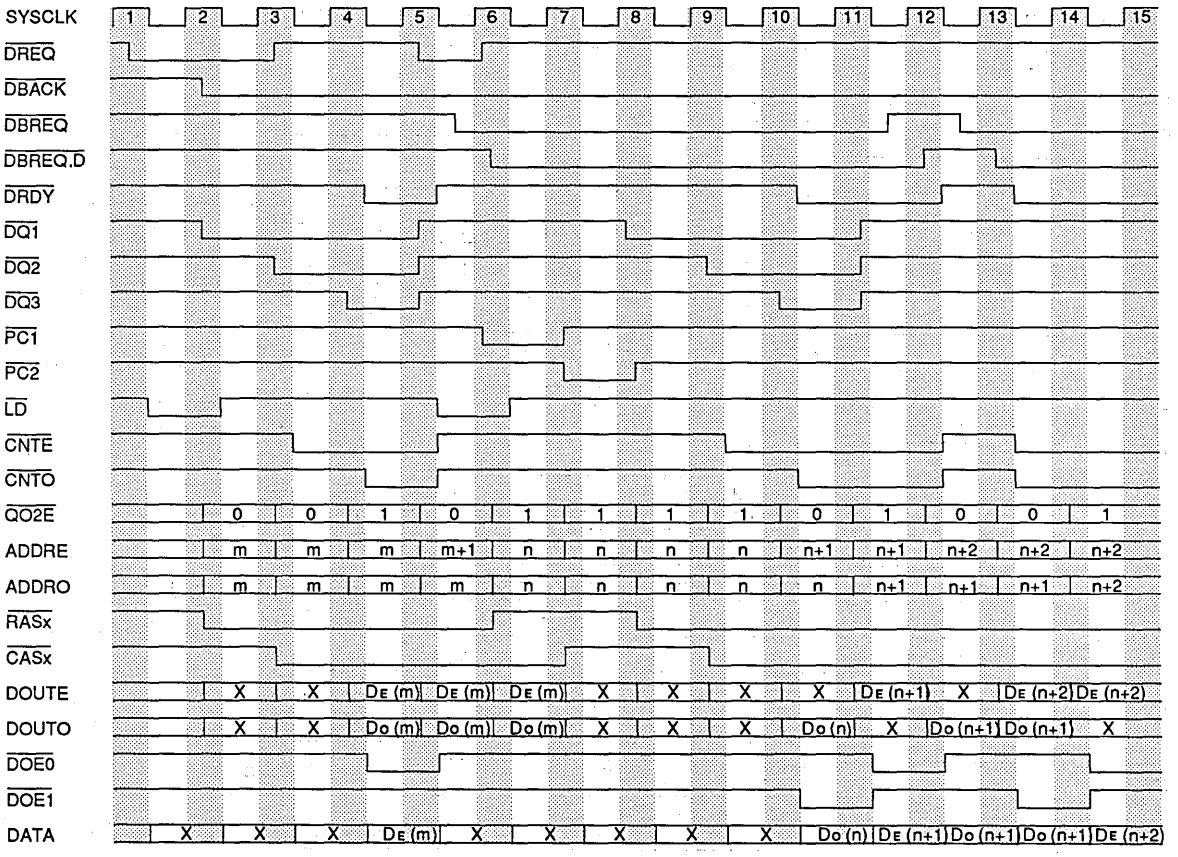
Figure 6-22



10117A-6.22A

DRAM Instruction Timing

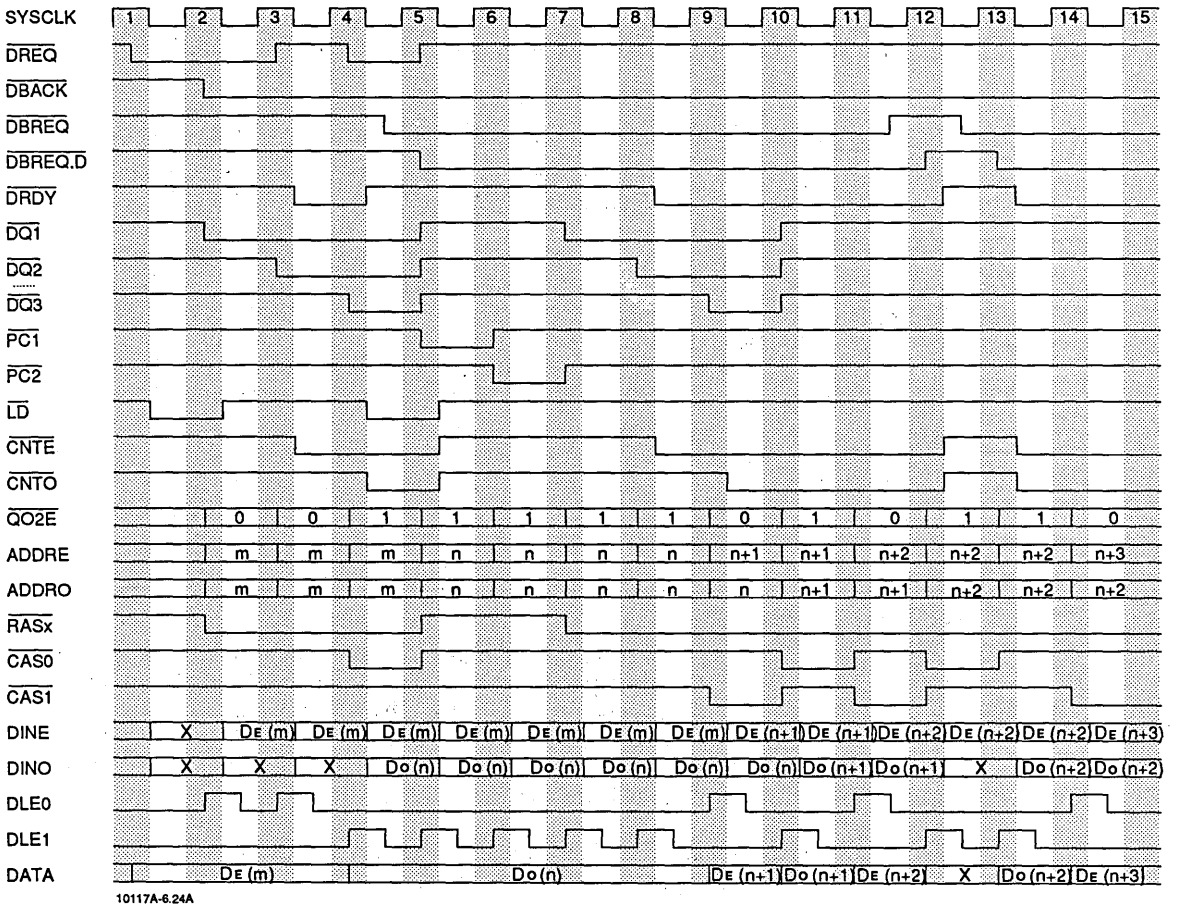
Figure 6-23



10117A-6.23A

DRAM Data Read Timing

Figure 6-24



10117A-6.24A

DRAM Data Write Timing

Parts List

The part list for the Am29000 Interleaved Dynamic RAM Interface is provided in Table 6-1.

Table 6-1 **Am29000 Interleaved Dynamic RAM Interface Parts List**

Item No.	Quantity	Device Description
U1	1	AmPAL16L8B
U2	1	AmPAL22V10A
U4, U5	2	AmPAL20L8B
U6,U9,U17,U18	4	AmPAL16R4D
U7,U10,U15,U16, U19, U20	6	AmPAL16R6D
U8, U11	2	AmPAL16L8D
U21-U85	64	TC511002-85
U3	1	74F175
U12-U14, U114-U116	6	74F158
U86-U94	8	Am29C843A
U95-U110	16	IDT74FCT244A
U111	1	MTTLDL-8
	112 pkgs	

DATA MEMORY

As shown in Figure 4-1 the instruction and data memories for the Am29000 are separate structures. The data memory can be an exact subset of the instruction memory design. In fact the exact same design can be used by tying the instruction-related control signals to the inactive state. But, since the data memory is a subset, it is also possible to save a few chips by eliminating the instruction-related control signals and rearranging the distribution of logic terms between PALs.

With reference to the instruction memory design defined in this chapter, the following changes may be made to convert it to a data memory:

- All instruction related inputs can be removed and all the affected equations simplified;
- U17, the instruction-state machine PAL, can therefore be removed entirely;
- The START signal can be moved to U16; therefore U4 can be eliminated;
- The 74F175 from the instruction-memory can also be used to supply the delayed control signals to the data memory, thus eliminating the need for U3;
- The ALE function from U8 and U11 can be moved to U1. Therefore U8 and U11 could be replaced by a single 10-bit latch such as the 29841A;
- And finally, the instruction-bus output buffers can be eliminated.

In total, the design can be reduced by 12 chips. The details of the logic equation simplifications will be left as an exercize for the reader. All other aspects of the design are the same as for the instruction memory described in the previous section.

CHAPTER 7
Video Dram With Interleaved Banks

Overview 7-1
Video Dram Advantages 7-1
Memory Features 7-2
Interface Logic Block Diagram 7-3
Memory Interface Logic Equations 7-6
Pal Definition Files 7-18
Intra-Cycle Timing 7-24
Inter-Cycle Timing 7-30
Parts List 7-30

Chapter 7

VIDEO DRAM WITH INTERLEAVED BANKS

OVERVIEW

Video DRAM Advantages

Video DRAM (VDRAM) offers an excellent way to reduce the complexity and component count of the memory system. A VDRAM has a dual-ported internal memory array. The first port allows read and write random access to the memory array just as a standard DRAM does. The second port is a serial shift register which is loaded from (and in some cases may be written to) one row of the memory array in a single access cycle. Once the serial shift register is loaded, it may be shifted independently of the random-access port. In effect, a VDRAM provides independent and concurrent access to a common memory array via these two ports. A single address bus provides access to either port.

This memory architecture greatly simplifies the interface to the Am29000. The shifter port can be connected to the instruction bus to provide sequential instruction streams. The random-access port can be connected to the data bus to provide read and write random access to data structures. And, both ports are addressed via the Am29000 address bus.

This nicely places both the instruction and data space in a common memory, thus significantly reducing the complexity of control logic and eliminating the need for many data buffers. Shared instruction and data space in a common memory also results in more efficient use of total memory space. This often results in a significant reduction in required memory size, therefore reduced component count. Due to the ability to concurrently access instructions and data, the VDRAM memory is still able to provide performance near that of the SCDRAM design from the last chapter.

The drawbacks to VDRAM are: a slower initial access time, lower density of currently available memories, and higher per memory cost, although much of the higher cost is offset by the lower cost of control and buffer logic in the system. Soon-to-be-available 1Mbit VDRAMs will remove the density limitation as compared with currently available 1Mbit DRAMs, although their initial cost will be high compared to the same density DRAMs.

Currently available VDRAMs also are unable to provide serial shifter ports fast enough to support a 40 ns instruction access time. To provide single-cycle burst instruction access speed, the current VDRAMs must be dual-bank interleaved. Again, future VDRAM may have the speed needed to eliminate dual-banking requirements. Where lower cost and simplicity is more important than a 20% clock-rate reduction, the system clock can be slowed to 20 MHz so that a single bank of VDRAM can keep up with the demands of the instruction bus.

As was described in the last chapter, the Am29000 provides unique features that allow the use of slower memories such as the VDRAM without the severe performance reductions that plague other high-performance microprocessors when using similar

memory systems As a result, VDRAM memories can significantly reduce system complexity and provide a fairly dense system memory, while also improving system performance-to-price ratio. The cost of the memory system drops while performance is reduced only slightly.

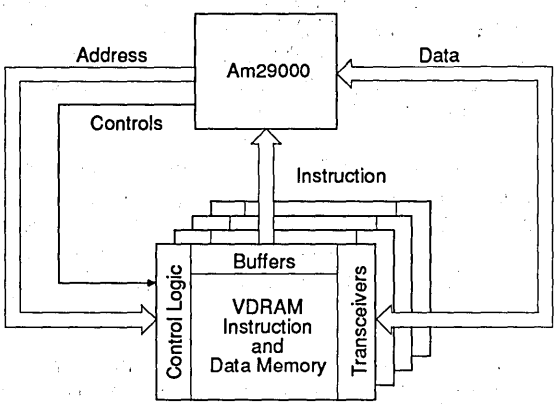
Memory Features

The memory design described in this chapter is an extension of the memory designs from the previous chapters. The first major difference, however, is that there is a single block of memory for instruction and data as shown in Figure 7-1. Within the memory block, there are two banks of memory interleaved as odd and even words. For a description of interleaved memory architecture see the overview section of Chapter 5, which discusses the bank-interleaved-SRAM concept.

Each bank is 64K words deep with each word being 32 bits wide. The total for the whole memory block is then 128K words (512K bytes). It is possible to use 120 ns access-time VDRAMs for both memory banks.

A non-sequential instruction access requires one cycle for address decode plus five additional cycles for the first word accessed. The burst access timing is similar to that used in previous chapters; each burst access is two cycles long. Overlapping the memory bank access time allows this longer access time to be hidden from the system viewpoint except on the first word of a non-sequential instruction access. The end result is a memory that provides 6-cycle access time for the first word of a non-sequential instruction access and single-cycle access for subsequent words in a burst transfer. A data read access requires one cycle for address decode plus four additional cycles to complete the access.

Figure 7-1



AM29000 with VDRAM Memory

A data write access requires one cycle for address decode plus two cycles or three cycles (depending on the memory used) to take data from the bus. The write operation continues internal to the memory for one or two additional cycles but the data bus is released after data is taken from the bus.

No burst accesses are supported for data. So, all data read accesses are five cycles long and all write accesses are three or four cycles long. That is assuming the memory has internally completed a write operation and/or $\overline{RAS}$ precharge before the next access begins. If write completion time or $\overline{RAS}$ precharge time has not been satisfied, a subsequent data access can require up to eight cycles to complete. This is based on the worst-case data read immediately following a data-write operation.

The VDRAM random access read/write port is connected to the Am29000 data bus. The serial-access shifter port is connected to the Am29000 instruction bus.

INTERFACE LOGIC BLOCK DIAGRAM (Figure 7-1)

The Memory

The memories are 64K x 4 bit VDRAMs supplied by either Fujitsu (MB81461-12) or NEC (PD41264-12). These memories have common data in and out lines. Their access speed is 120 ns. Eight devices are required in each bank to form the 32-bit wide instruction word for the Am29000. These are shown as devices U15 through U30.

VDRAM is used in this design to illustrate the savings in complexity, component count, and cost that the VDRAM architecture can provide when used with the Am29000. Largely those savings come from the fact that the instruction and data words can reside in a common memory array that still allows concurrent dual port access. Using one memory array, instead of split instruction and data memories, eliminates one entire set of memory control logic and data buffers. Also, the number of remaining control-logic and data-buffer circuits is reduced, since external buffers are no longer needed to support both data and instruction ports into the instruction memory.

Further, the VDRAM structure allows the boundary between instruction and data space to be flexible and dynamic, thereby providing for more efficient use of memory than a system that splits memory. This, in turn, may lead to reduced memory requirements in general.

Data Bus Transceivers

The memory random access data I/O port is connected to the Am29000 data bus lines via high-speed Am29C863 transceivers, U31 through U38 in Figure 7-2. These provide sufficient drive current to handle any reasonable capacitive load on the data bus.

In a system known to have minimal capacitive load on the data bus, it is possible to eliminate these transceivers. Note: if this is done, the Row Address Strobe ($\overline{RAS}$), Transfer/Output Enable (TR/$\overline{OE}$) and Serial Output Enable ($\overline{SOE}$) signals of the VDRAM may need to be qualified by address line 2 ($\overline{AX2}$) during data accesses so only one memory bank can be output enabled for each access. A side benefit of doing this may be lower power consumption by the memory system.

Instruction Bus Buffers

The memory serial-data outputs are connected to the instruction bus lines via buffers. These buffers serve to isolate the data outputs of this memory block from those outputs of other memory blocks which may also drive the instruction bus. Also, the buffers serve to isolate the even and odd banks of this memory block from each other so that simultaneous data access can go on in each bank independently. These buffers are shown as devices U39 through U46 in Figure 7-2.

Address Multiplexers

The upper and lower eight bits of memory address must be multiplexed into the address inputs of the memories. Discrete multiplexers are used to perform this function. These devices are shown as U5 through U8.

Note that in this design, unlike all previous chapters, the address is taken directly from the bus and through the multiplexers to the memories. No latching or registering of the address is done. This approach was taken to reduce the component count and complexity of the design as part of the overall goal of illustrating a lower cost memory design. Doing this requires that the memory control logic force the Am29000 to hold the address stable on the bus until after the $\overline{RAS}$ and Column Address Strobes ($\overline{CAS}$) have gone active. This is done by delaying the assertion of $\overline{IBACK}$, or $\overline{PEN}$ during instruction or data accesses respectively.

This reduces system performance somewhat, at least as compared with a split instruction and data memory system, or, a system in which there are multiple blocks of VDRAM in which one block could be addressed for an instruction fetch while another block is addressed for a data access. This is because the processor must, at times, hold an address on the bus when it might otherwise have been able to begin another access on an alternate memory block, assuming a memory that latches the address.

But, in a system having a single block of VDRAM, there is no benefit to latching the address from the bus. This is because the memory can not be ready to begin another access until the access in progress is completed and the memory has completed the precharge cycles that must occur between all non-sequential accesses.

NOTE: A word of warning, don't use inverting buffers or multiplexers on VDRAM address lines. Inverted random access I/O (DQ) port addressing would conflict with the sequentually incremented addressing required by the design of the serial port.

Bank Selector

Since a VDRAM uses a shift mechanism to provide the serial output of instructions, there is no need for an address counter. The initial address for an instruction burst request determines the starting location in the memory row to be shifted out. All subsequent instruction words are read by providing a shift clock to the VDRAM. Also, because the VDRAM shifter row is 256 words, the Am29000 always provides a new address at the right time when a row boundary is crossed. In addition no address counter is required for data accesses since no burst data accesses are supported in this memory design.

This design does, however, use bank interleaving to overcome the access delay of the VDRAM serial shifter port, so there must be a way provided to keep track of which bank

should be output enabled on to the instruction bus during any given cycle. Also, a way is needed to control the shift clock to each bank so that the instruction accesses are overlapped properly.

This tracking function is provided by registering address line A02 at the beginning of an access and then toggling the registered bit for each completed instruction access. This registered output is called Q02E as in the past chapters.

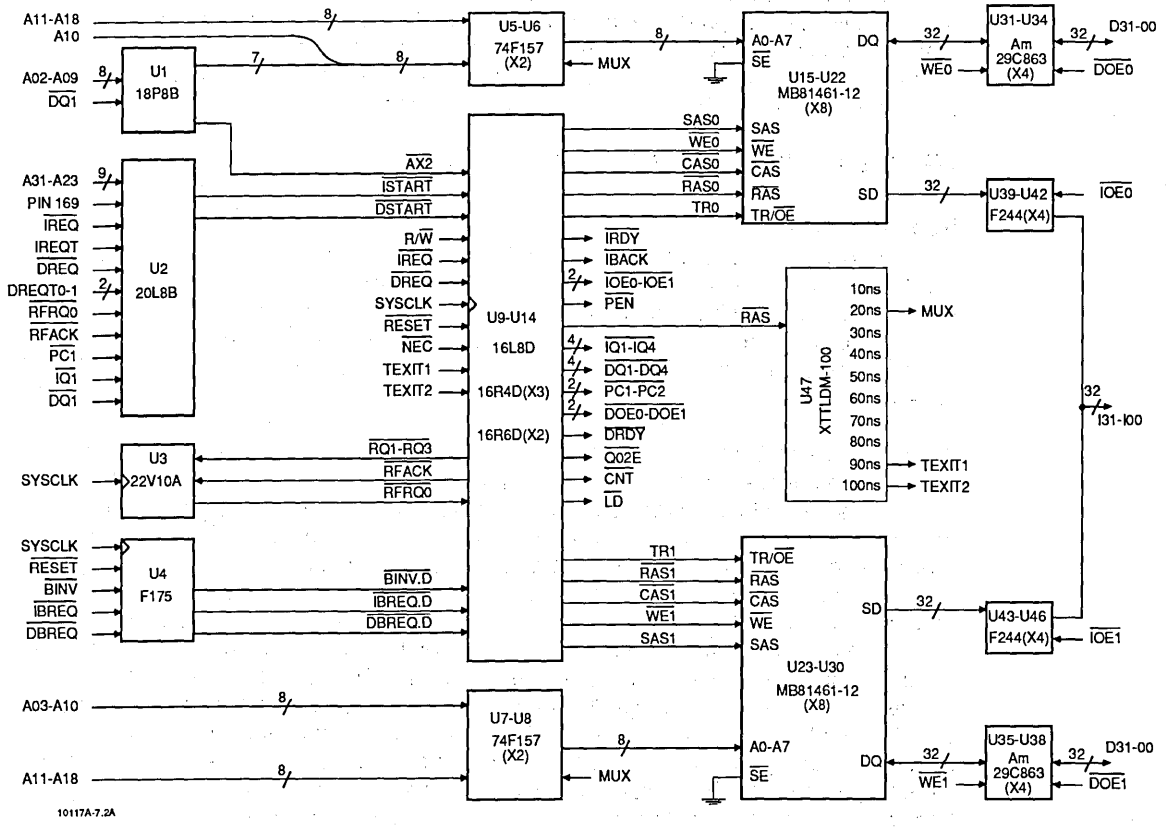
Registered Control Signals

As noted earlier, the timing of the $\overline{IBREQ}$, $\overline{DBERQ}$, and $\overline{BINV}$ control signals require that they be registered by a low-setup-time register; a F175 register, U4, shown in Figure 7-2 is used.

Interface Control Logic

This logic must generate the memory response signals, manage the loading of memory addresses, generate $\overline{RAS}$ and $\overline{CAS}$ signals, control the data buffer output enables, and perform memory refresh. The logic functions needed for this require 9 PALs: one

Figure 7-2



Interface Logic Block Diagram

AmPAL20L8B, three AmPAL16R4D, two AmPAL16R6D, one AmPAL16L8D, one AmPAL22V10A, and one AmPAL18P8B.

Referring to Figure 7-2, device U1, an AmPAL18P8B, serves to increment the memory address for the even bank when the initial address of an instruction access is odd. This causes the even bank to access the next even-bank word following the initial odd word.

Device U2, an AmPAL20L8B PAL, performs address decode for instruction and data accesses. Its outputs indicate when this memory block has been addressed and an access is to begin.

Device U3, an AmPAL22V10A, acts as a refresh-interval counter and refresh-request logic.

Devices U9 through U14, two AmPAL16R6D, three AmPAL16R4D PALs, and an AmPAL16L8D form a state machine that controls the $\overline{RAS}$, $\overline{CAS}$, shift clock, transfer cycle enable, bank selector, output buffer enables, write enables, and memory-response signals.

Response Signal Gating

As noted in the last chapter, the memory-response signals from all system bus devices are logically ORed together before being returned to the Am29000 processor. An example of this circuitry was shown in Figure 4-3. These gates are not included in the component count of this memory design since they are shared by all the bus devices in the system, and as such, are part of the overhead needed in any Am29000 system.

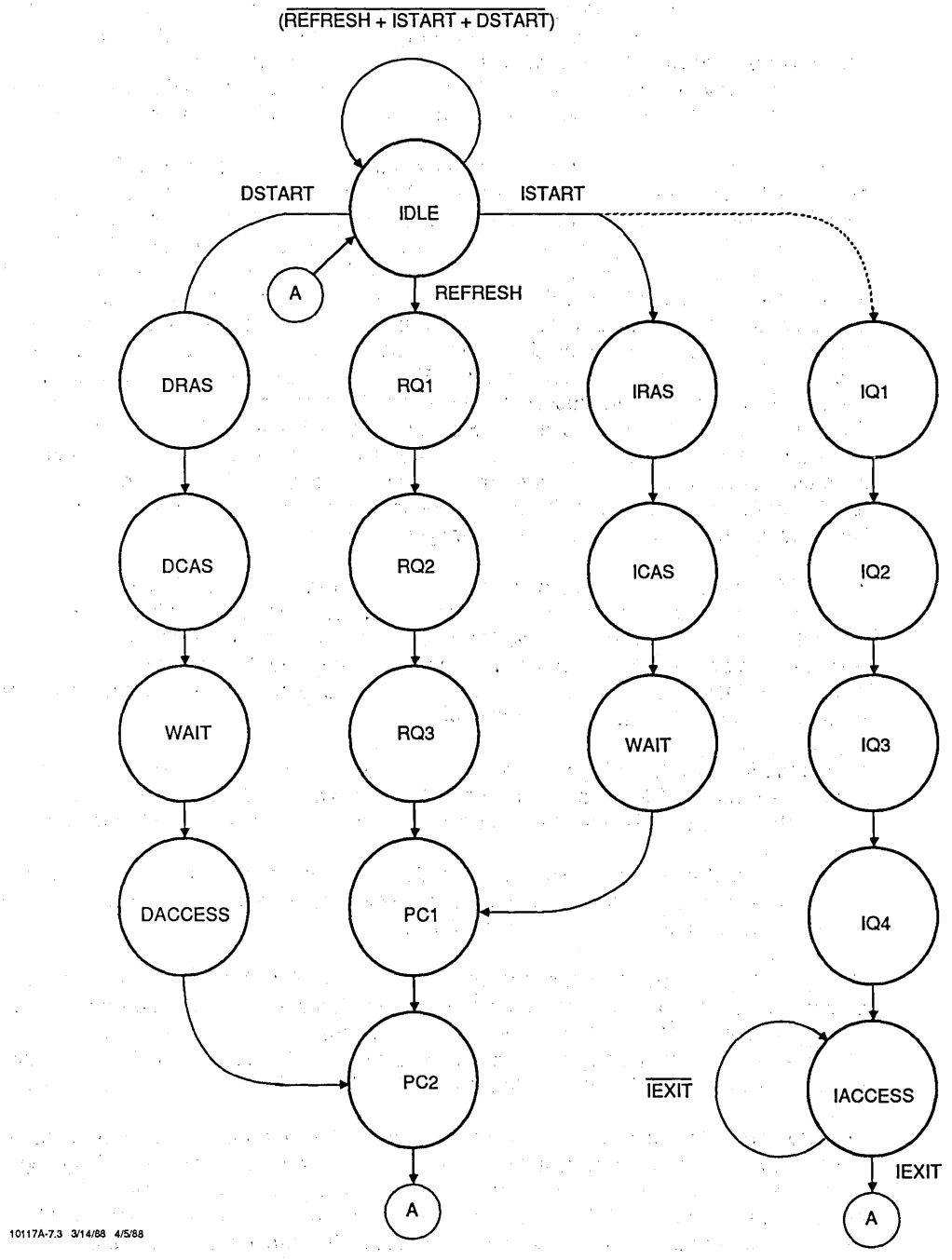
MEMORY INTERFACE LOGIC EQUATIONS

State Machine

The control logic for this memory can be thought of as a Mealy-type state machine in which the outputs are a function of the inputs and the present state of the machine. This structure is required since some of the output signals must be based on inputs which are not valid until the same cycle in which the outputs are required to effect control of the memory. As shown in Figure 7-3, this state machine can be described as having 18 states.

IDLE is the default state of the interface state machine. It is characterized by there being no instruction access in progress, or no data access in progress, and no refresh activity in progress. This state serves as a way of identifying when the memory is not being accessed and could be placed into a low power mode. This state also serves as a precharge cycle for the memory when a transition is made between instruction, data, and refresh sequences. A transition to either the IRAS or DRAS states occurs when an address selecting this memory block is placed on the address bus. A transition to the RQ1 state occurs when a refresh request is active. Refresh takes priority over any pending instruction or data-access request. There are five "Virtual States" shown in Figure 7-3; they are IQ1 through IQ4 and IACC. These states are needed due to the fact that the serial data (SD) port of the VDRAM operates independently of the random access I/O (DQ) port after a row transfer cycle is completed. The states help illustrate what might be called the "split personality" of the state machine. Once a transfer cycle begins, there are in effect two active states in this state machine. One state tracks the activity of the serial port control signals, and the other tracks the activity of signals associated with the random access I/O port.

Figure 7-3



Interface Logic State Diagram

The active states might be thought of as two tokens labeled SD and DQ being moved around a game board. The DQ token is never allowed to follow the dotted line to the virtual states. The SD token is always in one of the virtual states or the IDLE state, it never enters any of the other states. When the SD token enters the IDLE state, it cannot leave until the DQ token is also in IDLE and the ISTART condition is true.

When this situation occurs, the SD token moves to the IQ1 state and the DQ token moves to the IRAS state. This would represent the beginning of a row transfer to the serial-shift port. The DQ token then tracks the progress of $\overline{RAS}$, $\overline{CAS}$, and address signals applied to the VDRAM. When the transfer sequence is finished, the DQ token goes through the precharge states and returns to IDLE. The SD token proceeds through the IQ states counting off the delay needed until the first instruction is ready at the output of the SD port. In the IQ2 state, $\overline{IBACK}$ is made active to release the address bus. In IQ3 and IQ4, the shift clock and bank select signals begin operation, to effect the access of the first instruction word. In IACCESS, $\overline{IRDY}$ is allowed to go active. During subsequent cycles of an instruction burst access, the active state remains IACCESS. While the active state for instruction accessing is IACCESS, the DQ token is free to move through data-access states or refresh states completely independently of the instruction access in progress. When an instruction burst ends, the SD token returns to IDLE and must wait until the DQ token completes an access or refresh sequence followed by precharge before a new transfer cycle may begin.

The IRAS state occurs during the first cycle of a row transfer to the SD port following a new instruction address being presented on the address bus. During this state, the instruction output buffer enables and Ready response lines are held inactive and the $\overline{RAS}$ lines go active. $\overline{RAS}$ is used as the input to a delay line whose output will switch the address mux to the column address after the row address hold time is satisfied. The transition to the ICAS state is unconditional.

During the ICAS state CAS goes active to start the transfer cycle. Since the RAS minimum pulse width is 120 ns, and minimum CAS pulse width is 60 ns, a WAIT state follows the ICAS state before the unconditional transition to the first precharge state.

During the precharge states, $\overline{RAS}$ goes inactive. The precharge period for the memory used is 100 ns so a second and third precharge cycle is done during the PC2 and IDLE states, which unconditionally follow the PC1 cycle.

During a DQ port read sequence, the DRAS state generates $\overline{RAS}$ and the address-mux select signals. The DCAS state makes $\overline{CAS}$ active. Since the access time from $\overline{CAS}$ is 60 ns, the total of $\overline{CAS}$-clock-to-output delay, plus access time, plus data-buffer delays, plus processor set-up time is in excess of 95 ns, which will require a WAIT cycle, finally followed by the DACCESS cycle. During DACCESS, the DRDY signal is made active.

The DQ port write access is different only in that the DRDY signal may be made active during DCAS since the data from the bus is written into the memory by the falling edge of the $\overline{CAS}$ signal. Doing this allows the processor to begin a new address cycle on the address bus during the WAIT cycle. This may help improve system performance if the new address is directed at a different memory block that can immediately begin a new access. The WAIT cycle is used to fulfill the minimum $\overline{CAS}$ active time requirement. The

DACCESS simplifies the design by allowing the logic that controls the state transitions to be the same for both read and write operations.

Finally there is the refresh sequence. Once the IDLE state is reached and a refresh is pending, the refresh sequence starts as the highest priority task of the memory. In fact, during the IDLE cycle, $\overline{CAS}$ will go active to setup for a $\overline{CAS}$ -before-$\overline{RAS}$ refresh cycle. This type of refresh cycle makes use of the VDRAM internal refresh counters to supply the refresh address. During RQ1, $\overline{RAS}$ is made active as during IRAS and DRAS cycles. The RQ2 and RQ3 cycles are used to supply two additional wait states to make up the three cycles needed to satisfy the minimum $\overline{RAS}$ active time of 120 ns.

Logic Details—Signal By Signal

All signals are described in active high terms so that the design is a little easier to follow. The signals as implemented in the final PAL outputs are often active low as required by the actual circuit design. The actual PAL Definition files are included in Figures 7-4 through 7-12 at the end of this section.

NOTE: All PAL equations use the following convention:

- Where a PAL equation uses a colon followed by an equals sign (:=), the equation signals are REGISTERED PAL outputs.
- Where a PAL equation uses only an equals sign (=), the equation signals are COMBINATORIAL PAL outputs.

RFQ (Refresh Request)

Funny thing about dynamic memories, they're very forgetful. They need to be completely refreshed every 4 ms. Which translates into at least one row refreshed every 15.6 µs on average. To keep track of this time, a counter is used. Once a refresh interval has passed, a latch is used to remember that a refresh is requested while the counter continues to count the next interval. Once the refresh has been performed the latch is cleared.

The counter and refresh request latch is implemented in an AmPAL22V10A. Nine of the outputs form the counter which is incremented by the system clock at 25MHz. This gives up to 512 x 40 ns = 20.48 µs refresh periods. The synchronous preset term for all the registers is programmed to go active on a count value of 389 which will produce a refresh interval of 390 cycles x 40 ns = 15.6 µs. The one remaining output is used to implement the refresh request latch. That latch function (registered output) is also set by the synchronous preset term.

The equations for the counter are shown in Figure 7-4. Below are the preset and refresh latch equation:

$$\text{SYNCHRONOUS PRESET} = RFQ2 \cdot \overline{RFQ3} \cdot RFQ4 \cdot \overline{RFQ5} \cdot \overline{RFQ6} \cdot \overline{RFQ7} \cdot \overline{RFQ8} \cdot RFQ9 \cdot RFQ10$$

$$RFRQ0 := RFRQ0 \cdot \overline{(RFACK \cdot RQ1)}$$

Refresh Sequence Equations

A refresh of the memory requires multiple clocks so that the minimum RAS active time of 120 ns can be satisfied. To manage this, the following equations are used.

RFACK — The Refresh Acknowledge (RFACK) signal is used to begin a refresh sequence and to clear the pending refresh request. The RFACK signal goes active when the state machine (DQ token) re-enters the IDLE state as controlled by $\overline{IQ1}$ and $\overline{DQ1}$. RFACK is held active until the refresh request is cleared, indicated by $\overline{RFRQ0}$ • RQ3.

$$RFACK := \overline{DQ1} \cdot \overline{IQ1} \cdot \overline{RFRQ0} + RFACK \cdot (\overline{RFREQ0} \cdot RQ3)$$

RQ1, RQ2, RQ3 — The three cycles needed for a refresh are tracked by RQ1, RQ2, and RQ3. RQ1 will not go active until the cycle following the IDLE state. This is controlled by $\overline{RQ1} \cdot \overline{PC1}$ • RFACK which is only true during IDLE. RQ1 is held active for all three refresh cycles to provide a single signal to identify when a refresh is in progress. RQ2 and RQ3 simply follow RQ1 with RQ3 signaling the last cycle of the refresh sequence.

$$RQ1 := \overline{RQ1} \cdot \overline{PC1} \cdot RFACK + RQ1 \cdot \overline{RQ3}$$
$$RQ2 := RQ1 \cdot \overline{RQ3}$$
$$RQ3 := RQ2 \cdot \overline{RQ3}$$

IME

The use of the Instruction for ME (IME) signal is based on the assumption that other blocks of instruction or data memory may be added later and that there may be valid addresses in address spaces other than instruction/data space.

This means that this memory will only respond with IBACK or DRDY active when this block has been selected by valid addresses in the instruction/data space. This requires that at least some of the more significant address lines above the address range of this memory block be monitored to determine when this memory block is addressed. Also, it means the Instruction Request Type (IREQT), Data Request Type (DREQT 0, DREQT1), and Pin 169 lines must be monitored to determine that an address is valid and lies in the instruction/data space.

IME is the indication that the address of this memory block is present on the upper address lines, an instruction request is active, Pin 169 is inactive (test hardware has not taken control), and instruction/data address space is indicated. In other words this memory block is receiving a valid instruction access request. This example design will assume that the address of this memory block is equal to $\overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27} \cdot \overline{A26} \cdot \overline{A25} \cdot \overline{A24} \cdot \overline{A23}$. The equation for this signal is:

$$IME = IREQ \cdot \overline{IREQT} \cdot \overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27} \cdot \overline{A26} \cdot \overline{A25} \cdot \overline{A24} \cdot \overline{A23} \cdot \overline{Pin169}$$

Note that IME is not directly implemented as a PAL output in this design. The terms are used in the generation of the ISTART term.

DME

The Data for ME (DME) signal is the indication that the address of this memory block is present on the upper address lines, a data request is active, Pin 169 is inactive, and instruction/data address space is indicated. In other words this memory block is receiving a valid data access request. This example design will assume that the address of this memory block is equal to: $\overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27} \cdot \overline{A26} \cdot \overline{A25} \cdot \overline{A24} \cdot \overline{A23}$. Note that for this design both the instruction and data blocks reside in the same address space. This is possible because of the common memory array of the VDRAM that is accessible to either the instruction serial port or the data I/O port.

The equation for this signal is:

$$DME = DREQ \cdot \overline{DREQT0} \cdot \overline{DREQT1} \cdot \overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28} \cdot \overline{A27} \cdot \overline{A26} \cdot \overline{A25} \cdot \overline{A24} \cdot \overline{A23} \cdot \overline{Pin169}$$

As with IME, this term is not directly implemented.

ISTART

The Instruction Start (ISTART) signal causes the transition from IDLE to IRAS and IQ1 states. It is valid only in the IDLE state with no refresh sequence starting, identified by not being in any other state via $\overline{IQ1} \cdot \overline{DQ1} \cdot \overline{RFACK} \cdot \overline{PC1} \cdot \overline{PC2} \cdot \overline{RFRQ0}$. So when in the IDLE state and IME is active, ISTART is active.

$$ISTART = \overline{IQ1} \cdot \overline{DQ1} \cdot \overline{RFACK} \cdot \overline{PC1} \cdot \overline{PC2} \cdot \overline{RFRQ0} \cdot IME$$

DSTART

The Data Start (DSTART) signal is the same as ISTART except that DME is the qualifier.

$$DSTART = \overline{IQ1} \cdot \overline{DQ1} \cdot \overline{RFACK} \cdot \overline{PC1} \cdot \overline{PC2} \cdot \overline{RFRQ0} \cdot DME$$

IBACK

The Instruction Burst Acknowledge (IBACK) signal is applied to the Am29000 and is in effect the indication that the interface state machine is in an active or suspended instruction access. The equation is:

$$IBACK = IQ2 + \overline{IREQ} \cdot IBACK$$

The IBACK active state is entered during the IQ2 state. IBACK is delayed until IQ2 in order to hold the instruction address active on the bus until the CAS signal has gone active, thus eliminating the need for address latches or registers.

IBACK remains active until a new instruction access begins. The IBACK signal is combinatorial so that it will go inactive in the same cycle that IREQ goes active. This is required to hold the address on the bus until a new row transfer sequence can begin. The address must be held since there are no address latches or registers in this design to take the address from the bus. Address latches or registers would be required if IBACK were left active throughout the IREQ cycle.

This places a timing constraint on the IBACK response signal path that is different from all the earlier memory designs. IREQ is a signal that will not be stable until 14 ns into a cycle. The D-speed PAL logic that implements the IBACK logic has a propagation delay

of 10 ns. The Am29000 has a response signal setup time of 12 ns. These total 36 ns, which means that the logic OR gate used to combine all IBACK response signals in the system (Figure 4-3) must have a worst-case propagation delay of 4 ns. That is not easy to achieve when several IBACK response lines in the system must be logically ORed.

A solution to this is to move a copy of the VDRAM-block IBACK logic down into the PAL used to implement the IBACK response signal logical OR gate. That will eliminate one level of PAL delay. The equation for the response OR-gate function would then become:

$$\begin{aligned} IBACK = {} & IBACK0 \\ & + IBACK1 \\ & + IBACK2 \\ & + IBACK3 \\ & + IBACK4 \\ & + IBACK5 \\ & + IQ2 \\ & + \overline{IREQ} \cdot IBACK \end{aligned}$$

where the numbered IBACK inputs are the IBACK signals from other bus devices and the IQ2 + $\overline{IREQ}$ • IBACK inputs are from the VDRAM control logic.

The IBACK logic defined earlier remains to provide a version of IBACK local to the VDRAM control logic. That version of the IBACK is not as time critical since it will simply be registered. Only IBACK.D is needed by other parts of the VDRAM control logic.

IBACK.D

The IBACK Delayed (IBACK.D) signal is simply a 1-cycle delayed version of IBACK. The logic for IBACK is implemented directly in the IBACK.D equation.

$$\begin{aligned} IBACK.D \; := {} & IQ2 \\ & + \overline{IREQ} \cdot BACK \end{aligned}$$

It is used in the generation of IRDY, IOE0, IOE1, and CNT.

Instruction Initial Access States

Signals IQ1, IQ2, IQ3, and IQ4 are used to control the state transitions from IQ1 to IACCESS and IRAS through WAIT, during the first instruction access. The IQ1 signal goes active during the IQ1 and IRAS states and remains active for four additional cycles. IQ1 will go active only when there is a valid ISTART.

The IQ2, IQ3, and IQ4 signals are used to count the five cycles during which IQ1 is active. IQ3 is inactive during the fifth cycle after IQ1 goes active. This is used as a way of identifying the fifth cycle as the condition of $\overline{IQ3}$ • IQ4. This eliminates the need for an additional signal to directly indicate the fifth cycle.

$$\begin{aligned} IQ1 \; := {} & \overline{BINV} \cdot \overline{IQ1} \cdot ISTART \\ & + IQ1 \cdot \overline{(Q3 \cdot Q4)} \\ IQ2 \; := {} & IQ1 \cdot (\overline{IQ3} \cdot Q4) \\ IQ3 \; := {} & IQ2 \cdot \overline{IQ4} \\ IQ4 \; := {} & IQ3 \end{aligned}$$

Data Initial Access States

These equations are similar in function to the IQ1–IQ4 signals. They control state transitions during data accesses. DQ1 goes active during the DQ1 state as a result of a valid DSTART signal during the IDLE state. DQ2 through DQ4 simply count off the four DQ states.

$$DQ1 := \overline{BINV} \cdot \overline{DQ1} \cdot DSTART + DQ1 \cdot \overline{DQ4}$$

$$DQ2 := DQ1 \cdot \overline{DQ4}$$

$$DQ3 := DQ2 \cdot \overline{DQ4}$$

$$DQ4 := DQ3 \cdot \overline{DQ4}$$

Precharge States

At the end of any DQ port access, the RAS lines must be made inactive to precharge internal memory buses before another access with a different row address may begin. Three cycles are needed and are indicated by the signals PC1 and PC2. The PC1 signal is active during the PC1 state and the PC2 state. The PC2 signal is active during the PC2 state and the first IDLE state that follows the PC2 state. PC1 goes active following the third cycle of any instruction, data, or refresh sequence. In other words, once the minimum RAS pulse width requirement is satisfied, RAS is made inactive to begin precharging for the next access. In the case of a data read where the output data must be held valid after RAS goes inactive, the CAS signal is kept active to hold the data.

$$PC1 := \overline{PC1} \cdot IQ3 + \overline{PC1} \cdot DQ3 + \overline{PC1} \cdot RQ3 + PC1 \cdot \overline{PC2}$$

$$PC2 := PC1$$

LD

The Load (LD) signal enables address bit A02 to be loaded into the bank selection register (Q02E) on the next rising edge of SYSCLK. The equation is:

$$LD = IREQ \cdot \overline{IQ1}$$

In this design bank selection is only meaningful for an instruction access since no burst data accesses are supported. LD is thus active as a result of IREQ except during the access time of the first instruction word. This limitation in effect turns off LD after an instruction access begins so that LD will not interfere with the bank selection bit toggling activity that must go on during the initial access.

The LD signal is combinatorial so that it can be active during the first cycle of a new instruction request.

Bank Select Signal

The Q02E register bit is used to indicate which memory bank should provide valid instruction data to the instruction bus in any given cycle. Each time another instruction word is accessed this bit is toggled. The bit is originally loaded from the address-bus bit A02.

$$
\begin{aligned}
Q02E := {} & \overline{LD} \cdot AX2 \\
+ {} & \overline{LD} \cdot \overline{CNT} \cdot \overline{Q3} \cdot \overline{Q4} \cdot Q02E \\
+ {} & \overline{LD} \cdot IQ3 \cdot \overline{Q02E} \\
+ {} & \overline{LD} \cdot IQ4 \cdot \overline{Q02E} \\
+ {} & \overline{LD} \cdot CNT \cdot \overline{Q02E} \cdot \overline{BINV} \\
+ {} & \overline{LD} \cdot CNT \cdot Q02E \cdot BINV
\end{aligned}
$$

The use of BINV input will prevent Q02E from changing state during a cycle in which the bus is invalid. This prevents a state change in the memory resulting from bus control signals which may be invalid.

Q02E is used directly in the generation of the serial shift clock for the VDRAM. Before the first word in the serial shifter is available at the SD output of the VDRAM, one serial shift clock rising edge must occur. The IQ3 and IQ4 signals are used to force the first rising edges on the serial shift clock for each memory bank. After the IQ1 signal goes invalid any further toggling of the bank select signal and the serial port shift clock will come as a result of valid IBREQ cycles.

Even Bank Address Incrementer and LSB Latch

In this design, the lack of address counters requires a new way of satisfying the need to increment the even bank address before the first word access, when the initial address is odd. To deal with this need, an AmPAL18P8B is used to build a flow-through incrementer. The increment function is selective in that when address bit A02 is low, indicating an even word initial address, no increment is done and the address passes through unchanged. When A02 is high, the memory address is incremented. The A02 bit is used to select which bank is read or written during a data access. Thus, the A02 bit is required to be stable throughout the entire access. So that it may be held stable after the address bus is released, the A02 bit is latched within the incrementer by the DQ1 signal. The equations for the increment and latch functions are shown in Figure 7-12.

Count Signal

The Count (CNT) signal in this design is reduced to being an enable on the toggling action of the Q02E bit. Following the initial instruction word access, determined by $\overline{IQ1}$, the CNT signal is active for each valid instruction burst request, determined by IBREQ.D and IBACK.D.

$$CNT = \overline{IQ1} \cdot IBREQ.D \cdot IBACK.D$$

Transfer Cycle Enable and DQ Port Output Enable

On a VDRAM, there is a dual function signal, called Transfer (TR), which controls when a row transfer cycle is performed and also when the random I/O data port is output enabled. When TR is active during the active edge of RAS, a transfer cycle is performed.

The timing of TR is critical when performing this function. It must stay active for a minimum of 90 or 100 ns after RAS goes active when the Fujitsu VDRAM (MB81461-12) or NEC VDRAM (PD41264-12) respectively is used. The signal must also be inactive 25 ns or 10 ns respectively before the serial shift clock may go from low to high, to clock out the first instruction word.

To make the above timing constraint fit within the 6-cycle initial access time of this memory design, a delay line must be used to precisely set the duration of the TR signal. A separate RAS signal, which is not loaded by the capacitance of either memory bank, is the input to the delay line. The output for a 90 ns delay is TEXIT1 and for a 100 ns delay is TEXIT2. More details of this timing are provided in the intra-cycle timing section of this chapter.

TR goes active with IREQ, so that TR is set up before RAS goes active. TR latches itself active until the appropriate TEXIT signal goes active. The NEC input is strapped to low when the NEC memory is used, or to high when the Fujitsu VDRAM is used.

Finally, when DQ2 is active during a non-transfer cycle of a read operation, the active TR signal enables the DQ port output.

$$
\begin{aligned}
TR0 = {} & \overline{DQ1} \cdot IREQ \\
& + \overline{DQ1} \cdot TR0 \cdot \overline{NEC} \cdot \overline{TEXIT1} \\
& + \overline{DQ1} \cdot TR0 \cdot NEC \cdot \overline{TEXIT2} \\
& + DQ2 \cdot \overline{WE1}
\end{aligned}
$$

Shift Clock

The signal that clocks each new instruction out of the serial port is referred to as SAS. This signal must be low at the time TR goes inactive and it must remain low for the 25 ns or 10 ns period noted earlier. Once that timing constraint is satisfied, the next rising edge of SAS clocks the serial port output. SAS is held low while IQ1 is active and IQ4 is inactive. After that time, SAS is controlled by the Q02E bank selection signal so that a new instruction is clocked out every other system clock cycle when the CNT signal is active.

There is a special requirement on SAS immediately following system power-on time. The SAS signal must be cycled at least eight times before proper device operation is achieved following a power-on sequence. To ensure this is done, the system reset signal is used to connect the system clock to SAS. This ensures SAS is cycled during the system power-on reset time.

$$
\begin{aligned}
SAS0 = {} & RESET \cdot SYSCLK \\
& + \overline{RESET} \cdot IQ1 \cdot IQ4 \\
& + \overline{RESET} \cdot IQ4 \cdot Q02 \\
& + \overline{RESET} \cdot IQ1 \cdot Q02E
\end{aligned}
$$

$$
\begin{aligned}
SAS1 = {} & RESET \cdot SYSCLK \\
& + \overline{RESET} \cdot IQ1 \cdot IQ4 \\
& + \overline{RESET} \cdot IQ4 \cdot Q02E \\
& + \overline{RESET} \cdot IQ1 \cdot Q02E
\end{aligned}
$$

IRDY

The Instruction Ready (IRDY) signal indicates that there is valid read data on the instruction bus.

$$IRDY = \overline{Q3} \cdot IQ4 + \overline{BINV.D} \cdot \overline{IQ1} \cdot IBREQ.D \cdot IBACK.D$$

This memory design is always ready with data in the IACCESS state indicated by $\overline{IQ3} \cdot IQ4$.

The memory is also ready when IBREQ is active with IBACK in the previous cycle with no invalid bus condition, following the initial instruction word access.

The reason that IRDY must be a combinatorial signal is that IBREQ comes very late in the previous cycle and must be registered. There is no IBREQ qualifying time available in the previous cycle before SYSCLK rises. This means that the information that IBREQ was active in the last cycle is not available until the cycle in which IRDY should go active for a resumption of a suspended burst access.

IOE0 and IOE1

The Instruction Output Enable (IOE) signals control the even and odd memory banks are used to control which bank is allowed to drive the instruction bus during each cycle. The signals use essentially the same logic as IRDY except that each signal is further qualified by the bank select signal (Q02E). This bit keeps track of which memory bank is ready to provide data to the instruction bus. The even bank is enabled when IRDY is active and the Q02E bit is one. The odd bank is enabled when IRDY is active and Q02E is zero.

$$IOE0 = \overline{Q02E} \cdot \overline{IQ3} \cdot IQ4 + \overline{BINV.D} \cdot \overline{Q02E} \cdot \overline{Q1} \cdot IBREQ.D \cdot IBACK.D$$

$$IOE1 = Q02E \cdot \overline{Q3} \cdot IQ4 + \overline{BINV.D} \cdot Q02E \cdot \overline{IQ1} \cdot IBREQ.D \cdot IBACK.D$$

DRDY

The Data Ready (DRDY) signal is the equivalent of IRDY, but for data accesses. The difference is that since no burst accesses are supported, DRDY will go active only once in each simple access during the DACCESS state in a read, or during DCAS or WAIT in a write operation. Due to different data hold times for the Fujitsu and NEC VDRAM the DRDY must be held until the WAIT state when using the NEC VDRAM.

$$DRDY = \overline{WE0} \cdot DQ4 + WE0 \cdot DQ2 \cdot \overline{DQ3} \cdot \overline{NEC} + WE0 \cdot DQ3 \cdot \overline{DQ4} \cdot NEC$$

DOE0 and DOE1

The Data buffer Output Enable (DOE) signals serve the same function for DRDY as does the IOE0 & IOE1 signals do for IRDY. They are active only during read operations and the selected bank is determined by the latched version of address bit 2 (AX2).

$$DOE0 = \overline{WE0} \cdot AX2 \cdot DQ3$$

$$DOE1 = \overline{WE0} \cdot AX2 \cdot DQ3$$

Pipeline Enable

During a read operation the data address is no longer needed on the address bus following the DCAS state. So, to help improve system performance, the Pipeline ENable (PEN) signal response is made active during the DCAS state. This active PEN signal tells the processor that the address is no longer needed and it allows the processor to place a new address on the bus. In cases where the next address to be issued is for an instruction or data access from a different block of memory, the next access can begin while the current data access finishes.

$$PEN = DQ2 \cdot \overline{DQ3}$$

WE

Write Enable (WE) signal is not allowed to be active during the row transfer sequence that begins each non-sequential instruction access. This is because no write operations are supported for the serial port. During a data access, the read/write line is latched by the DQ2 signal at the end of the DCAS state.

Two WE signals are defined simply to reduce the capacitive load on the signals. There is one WE for each bank.

$$\begin{aligned} WE0 &= \overline{Q1} \cdot DQ1 \cdot \overline{DQ2} \cdot \overline{RW} \\ &+ \overline{IQ1} \cdot DQ1 \cdot DQ2 \cdot WE0 \\ WE1 &= \overline{IQ1} \cdot DQ1 \cdot \overline{DQ2} \cdot \overline{RW} \\ &+ \overline{IQ1} \cdot DQ1 \cdot DQ2 \cdot WE1 \end{aligned}$$

Row Address Strobes

There are three duplicate Row Address Strobe (RAS) lines. Two are used to drive the memories and one drives the delay line used to switch the address mux at the appropriate time and to control the duration of the transfer signal. Multiple lines are used to split the capacitive and inductive load of the memory array to improve signal speed.

RAS is made active by a valid ISTART, DSTART or refresh condition. RAS is held active for 3 cycles to satisfy the minimum pulse-width requirement on RAS.

$$\begin{aligned} RAS := &\ \overline{BINV} \cdot \overline{RAS} \cdot ISTART \\ &+ \overline{BINV} \cdot \overline{RAS} \cdot DSTART \\ &+ \overline{BINV} \cdot \overline{RAS} \cdot \overline{PC1} \cdot RFACK \\ &+ RAS \cdot IQ1 \cdot \overline{IQ3} \\ &+ RAS \cdot DQ1 \cdot \overline{DQ3} \\ &+ RAS \cdot RFACK \cdot \overline{RQ3} \end{aligned}$$

Column Address Strobes

As with the RAS lines, the CAS lines are duplicated to split the memory load. CAS goes active in the cycle after RAS during instruction or data accesses. During a data write access CAS is enabled only when the appropriate bank is written with data. This is controlled by the latched value of the address bit 2 (AX2). Only in the case of a refresh sequence will CAS be made active prior to RAS. This will initiate a CAS before RAS refresh cycle in the memories. In this case CAS is made active during the IDLE state.

$$\begin{aligned} CAS0 := &\ RAS \cdot IQ1 \\ &+ RAS \cdot DQ1 \cdot \overline{AX2} \\ &+ \overline{RAS} \cdot \overline{Q1} \cdot \overline{DQ1} \cdot RFRQ0 \\ CAS1 := &\ RAS \cdot IQ1 + RAS \cdot DQ1 \cdot AX2 \\ &+ \overline{RAS} \cdot \overline{Q1} \cdot \overline{DQ1} \cdot RFRQ0 \end{aligned}$$

PAL DEFINITION FILES

The PAL definition files are provided in Figures 7-4 through 7-12.

NOTE: All PAL equations in this handbook use the following convention:

- Where a PAL equation uses a colon followed by an equals sign (:=), the equation signals are REGISTERED PAL outputs.
- Where a PAL equation uses only an equals sign (=), the equation signals are COMBINATORIAL PAL outputs.
- The Device Pin list is shown near the top of each figure as two lines of signal names. The names occur in pin order, numbered from left to right 1 through 20. The polarity of each name indicates the actual input or output signal polarity. Signals within the equations are shown as active high, e.g., where signal names in the pin list are: $\overline{A}$ B $\overline{C}$; the equation is C = A • $\overline{B}$; the inputs are $\overline{A}$ = low, B = low; then the $\overline{C}$ output will be low.

Figure 7-4

AmPAL22V10A VRAM Refresh Counter/Request Generator
Device U3

CLK $\overline{RFACK}$ $\overline{RQ1}$ $\overline{RQ2}$ $\overline{RQ3}$ NC6 NC7 NC8 NC9 NC10 NC11 GND
NC13 $\overline{RFRQ0}$ $\overline{RFQ2}$ $\overline{RFQ3}$ $\overline{RFQ4}$ $\overline{RFQ5}$ $\overline{RFQ6}$ $\overline{RFQ7}$ $\overline{RFQ8}$ $\overline{RFQ10}$ $\overline{RFQ9}$ VCC

RFQ2 := $\overline{RFQ2}$

RFQ3 := RFQ2 • $\overline{RFQ3}$
+ $\overline{RFQ2}$ • RFQ3

RFQ4 := RFQ2 • RFQ3 • $\overline{RFQ4}$
+ $\overline{RFQ2}$ • RFQ4
+ RFQ3 • RFQ4

RFQ5 := RFQ2 • RFQ3 • RFQ4 • $\overline{RFQ5}$
+ $\overline{RFQ2}$ • RFQ5
+ $\overline{RFQ3}$ • RFQ5
+ $\overline{RFQ4}$ • RFQ5

RFQ6 := RFQ2 • RFQ3 • RFQ4 • RFQ5 • $\overline{RFQ6}$
+ $\overline{RFQ2}$ • RFQ6
+ $\overline{RFQ3}$ • RFQ6
+ $\overline{RFQ4}$ • RFQ6
+ $\overline{RFQ5}$ • RFQ6

RFQ7 := RFQ2 • RFQ3 • RFQ4 • RFQ5 • RFQ6 • $\overline{RFQ7}$
+ $\overline{RFQ2}$ • RFQ7
+ $\overline{RFQ3}$ • RFQ7
+ $\overline{RFQ4}$ • RFQ7
+ $\overline{RFQ5}$ • RFQ7
+ $\overline{RFQ6}$ • RFQ7

Figure 7-4 (Continued)

Device U3 (Continued)

RFQ8 := RFQ2 • RFQ3 • RFQ4 • RFQ5 • RFQ6 • RFQ7 • $\overline{RFQ8}$
+ $\overline{RFQ2}$ • RFQ8
+ $\overline{RFQ3}$ • RFQ8
+ $\overline{RFQ4}$ • RFQ8
+ $\overline{RFQ5}$ • RFQ8
+ $\overline{RFQ6}$ • RFQ8
+ $\overline{RFQ7}$ • RFQ8

RFQ9 := RFQ2 • RFQ3 • RFQ4 • RFQ5 • RFQ6 • RFQ7 • RFQ8 • $\overline{RFQ9}$
+ $\overline{RFQ2}$ • RFQ9
+ $\overline{RFQ3}$ • RFQ9
+ $\overline{RFQ4}$ • RFQ9
+ $\overline{RFQ5}$ • RFQ9
+ $\overline{RFQ6}$ • RFQ9
+ $\overline{RFQ7}$ • RFQ9
+ $\overline{RFQ8}$ • RFQ9

RFQ10 := RFQ2 • RFQ3 • RFQ4 • RFQ5 • RFQ6 • RFQ7 • RFQ8 • RFQ9 • $\overline{RFQ10}$
+ $\overline{RFQ2}$ • RFQ10
+ $\overline{RFQ3}$ • RFQ10
+ $\overline{RFQ4}$ • RFQ10
+ $\overline{RFQ5}$ • RFQ10
+ $\overline{RFQ6}$ • RFQ10
+ $\overline{RFQ7}$ • RFQ10
+ $\overline{RFQ8}$ • RFQ10
+ $\overline{RFQ9}$ • RFQ10

SYNCHRONOUS PRESET = RFQ2 • $\overline{RFQ3}$ • RFQ4 • $\overline{RFQ5}$ • $\overline{RFQ6}$ • $\overline{RFQ7}$ • $\overline{RFQ8}$ • RFQ9 • RFQ10

RFRQ0 := RFRQ0 • ($\overline{RFACK \cdot RQ1}$)

Figure 7-5

AmPAL20L8B VRAM State Decoder—Interleaved Device U2

$\overline{IREQ}$ DREQT0 IREQT A31 A30 A29 A28 A27 A26 A25 A24 GND
$\overline{DREQ}$ DREQT1 $\overline{ISTART}$ $\overline{RFRQ0}$ $\overline{RFACK}$ PIN169 $\overline{IQ1}$ $\overline{DQ1}$ $\overline{PC1}$ $\overline{DSTART}$ A23 VCC

ISTART = $\overline{IQ1}$ • $\overline{DQ1}$ • $\overline{RFACK}$ • $\overline{PC1}$ • $\overline{PC2}$ • $\overline{RFRQ0}$ • IME

DSTART = $\overline{IQ1}$ • $\overline{DQ1}$ • $\overline{RFACK}$ • $\overline{PC1}$ • $\overline{PC2}$ • $\overline{RFRQ0}$ • DME

NOTE: In the above equations, IME and DME are used only for clarity. The actual input terms should be substituted when compiling this device.

IME = IREQ • $\overline{IREQT}$ • $\overline{A31}$ • $\overline{A30}$ • $\overline{A29}$ • $\overline{A28}$ • $\overline{A27}$ • $\overline{A26}$ • $\overline{A25}$ • $\overline{A24}$ • $\overline{A23}$ • PIN169

DME = DREQ • DREQT0 • DREQT1 • A31 • $\overline{A30}$ • $\overline{A29}$ • $\overline{A28}$ • $\overline{A27}$ • $\overline{A26}$ • $\overline{A25}$ • $\overline{A24}$ • $\overline{A23}$ • PIN169

Figure 7-6

AmPAL16R4D VRAM Instruction State Generator—Interleaved Device U9

```
CLK /IREQ /ISTART NC4 NC5 /Q02E /IBREQ.D /BINV /BINV.D GND
/OE /IOE0 /IOE1 /IQ1 /IQ2 /IQ3 /IQ4 /IRDY /IBACK.D VCC

IQ1   := /BINV * /IQ1 * ISTART
       + IQ1 * (/IQ3 * IQ4)

IQ2   := IQ1 * /(/IQ3 * IQ4)

IQ3   := IQ2 * /IQ4

IQ4   := IQ3

IRDY  = /IQ3 * IQ4
       + /BINV.D * /IQ1 * IBREQ.D * IBACK.D

IOE0  = /Q02E * /IQ3 * IQ4
       + /BINV.D * /Q02E * /IQ1 * IBREQ.D * IBACK.D

IOE1  = Q02E * /IQ3 * IQ4
       + /BINV.D * Q02E * /IQ1 * IBREQ.D * IBACK.D
```

Figure 7-7

AmPAL16R4D VRAM Data State Generator—Interleaved Device U10

```
CLK /DSTART AX2 /WE0 /NEC NC6 NC7 /BINV NC9 GND
/OE /DOE0 /DOE1 /DQ1 /DQ2 /DQ3 /DQ4 /DRDY /PEN VCC

DQ1   := /BINV * /DQ1 * DSTART
       + DQ1 * /DQ4

DQ2   := DQ1 * /DQ4

DQ3   := DQ2 * /DQ4

DQ4   := DQ3 * /DQ4

DRDY  = /WE0 * DQ4
       + WE0 * DQ2 * /DQ3 * /NEC
       + WE0 * DQ3 * /DQ4 * NEC

DOE0  = /WE0 * /AX2 * DQ3

DOE1  = /WE0 * AX2 * DQ3

PEN   = DQ2 * /DQ3
```

Figure 7-8

AmPAL16L8D VRAM Transfer Generator—Interleaved
Device U14

```
Q02E TEXIT1 TEXIT2 DQ1 DQ2 IREQ WE1 NEC NC9 GND
SYSCLK SAS0 TR0 RESET IQ1 IQ4 NC17 TR1 SAS1 VCC

SAS0 = RESET • SYSCLK
     + RESET • IQ1 • IQ4
     + RESET • IQ4 • Q02E
     + RESET • IQ1 • Q02E

SAS1 = RESET • SYSCLK
     + RESET • IQ1 • IQ4
     + RESET • IQ4 • Q02E
     + RESET • IQ1 • Q02E

TR0  = DQ1 • IREQ
     + DQ1 • TR0 • NEC • TEXIT1
     + DQ1 • TR0 • NEC • TEXIT2
     + DQ2 • WE1

TR1  = DQ1 • IREQ
     + DQ1 • TR1 • NEC • TEXIT1
     + DQ1 • TR1 • NEC • TEXIT2
     + DQ2 • WE1
```

Figure 7-9

AmPAL16R6D VRAM RAS Generator—Interleaved
Device U12

```
CLK ISTART DSTART IQ1 DQ1 IQ3 DQ3 RQ3 BINV GND
OE RFACK RAS0 RAS1 RAS PC1 PC2 NC18 NC19 VCC

RAS0 := BINV • RAS0 • ISTART
      + BINV • RAS0 • DSTART
      + BINV • RAS0 • PC1 • RFACK
      + RAS0 • IQ1 • IQ3
      + RAS0 • DQ1 • DQ3
      + RAS0 • RFACK • RQ3

RAS1 := BINV • RAS1 • ISTART
      + BINV • RAS1 • DSTART
      + BINV • RAS1 • PC1 • RFACK
      + RAS1 • IQ1 • IQ3
      + RAS1 • DQ1 • DQ3
      + RAS1 • RFACK • RQ3

RAS  := BINV • RAS • ISTART
      + BINV • RAS • DSTART
      + BINV • RAS • PC1 • RFACK
      + RAS • IQ1 • IQ3
      + RAS • DQ1 • DQ3
      + RAS • RFACK • RQ3

PC1  := PC1 • IQ3
      + PC1 • DQ3
      + PC1 • RQ3
      + PC1 • PC2

PC2  := PC1
```

Figure 7-10

AmPAL16R6D VRAM CAS Generator—Interleaved Device U13

```
CLK /PC1 /IQ1 /DQ1 /DQ2 /RAS /RFRQ0 /RW AX2 GND
/OE /WE0 /CAS0 /CAS1 /RFACK /RQ1 /RQ2 /RQ3 /WE1 VCC

CAS0  := RAS • IQ1
       + RAS • DQ1 • /AX2
       + /RAS • /IQ1 • /DQ1 • RFRQ0

CAS1  := RAS • IQ1
       + RAS • DQ1 • AX2
       + /RAS • /IQ1 • /DQ1 • RFRQ0

WE0   = /IQ1 • DQ1 • /DQ2 • /RW
       + /IQ1 • DQ1 • DQ2 • WE0

WE1   = /IQ1 • DQ1 • /DQ2 • /RW
       + /Q1 • DQ1 • DQ2 • WE1

RFACK := /IQ1 • /IQ1 • /RFRQ0
       + RFACK • (/RFRQ0 • RQ3)

RQ1   := /RQ1 • /PC1 • RFACK
       + RQ1 • /RQ3

RQ2   := RQ1 • /RQ3

RQ3   := RQ2 • RQ3
```

Figure 7-11

AmPAL16R4D VRAM Counter Load—Interleaved Device U11

```
CLK NC2 /IBREQ.D /IREQ /IQ1 /IQ2 /IQ3 /IQ4 /BINV GND
/OE /CNT /IBACK /Q02E /IBACK.D NC16 NC17 /LD AX2 VCC

LD      = IREQ • /IQ1

CNT     = /IQ1 • IBREQ.D • IBACK.D

Q02E    := LD • AX2
         + /LD • /CNT • /IQ3 • /IQ4 • Q02E
         + /LD • IQ3 • /Q02E
         + /LD • IQ4 • /Q02E
         + /LD • CNT • /Q02E • /BINV
         + /LD • CNT • Q02E • BINV

IBACK.D := IQ2
         + /IREQ • IBACK
```

Figure 7-12

AmPAL18P8B VRAM Address Incrementer
Device U1

$\overline{DQ1}$ A02 A03 A04 A05 A06 A07 A08 A09 GND
NC11 AX9 AX8 AX7 AX6 AX5 AX4 AX3 AX2 VCC

$$\overline{AX2} = \overline{DQ1} \cdot \overline{A02} + DQ1 \cdot \overline{AX2}$$

$$\overline{AX3} = \overline{A02} \cdot \overline{A03} + A02 \cdot A03$$

$$\overline{AX4} = \overline{A02} \cdot \overline{A04} + A02 \cdot A03 \cdot A04 + A02 \cdot \overline{A03} \cdot \overline{A04}$$

$$\overline{AX5} = \overline{A02} \cdot \overline{A05} + A02 \cdot A03 \cdot A04 \cdot A05 + A02 \cdot \overline{A03} \cdot \overline{A05} + A02 \cdot \overline{A04} \cdot \overline{A05}$$

$$\overline{AX6} = \overline{A02} \cdot \overline{A06} + A02 \cdot A03 \cdot A04 \cdot A05 \cdot A06 + A02 \cdot \overline{A03} \cdot \overline{A06} + A02 \cdot \overline{A04} \cdot \overline{A06} + A02 \cdot \overline{A05} \cdot \overline{A06}$$

$$\overline{AX7} = \overline{A02} \cdot \overline{A07} + A02 \cdot A03 \cdot A04 \cdot A05 \cdot A06 \cdot A07 + A02 \cdot \overline{A03} \cdot \overline{A07} + A02 \cdot \overline{A04} \cdot \overline{A07} + A02 \cdot \overline{A05} \cdot \overline{A07} + A02 \cdot \overline{A06} \cdot \overline{A07}$$

$$\overline{AX8} = \overline{A02} \cdot \overline{A08} + A02 \cdot A03 \cdot A04 \cdot A05 \cdot A06 \cdot A07 \cdot A08 + A02 \cdot \overline{A03} \cdot \overline{A08} + A02 \cdot \overline{A04} \cdot \overline{A08} + A02 \cdot \overline{A05} \cdot \overline{A08} + A02 \cdot \overline{A06} \cdot \overline{A08} + A02 \cdot \overline{A07} \cdot \overline{A08}$$

$$\overline{AX9} = \overline{A02} \cdot \overline{A09} + A02 \cdot A03 \cdot A04 \cdot A05 \cdot A06 \cdot A07 \cdot A08 \cdot A09 + A02 \cdot \overline{A03} \cdot \overline{A09} + A02 \cdot \overline{A04} \cdot \overline{A09} + A02 \cdot \overline{A05} \cdot \overline{A09} + A02 \cdot \overline{A06} \cdot \overline{A09} + A02 \cdot \overline{A07} \cdot \overline{A09} + A02 \cdot \overline{A08} \cdot \overline{A09}$$

INTRA-CYCLE TIMING

This memory architecture has five timing sequences of interest. The first is a cycle used to decode the memory address and control signals from the processor. At the end of this decode cycle, the $\overline{RAS}$ registers are loaded to begin the initial access of memory, if the address selects the memory block.

Following the decode cycle, is the Row Address cycle, in which the row address strobe is made active at the beginning of the cycle, and in which the address multiplexer is later switched between the row address and the column address.

The third timing is a data access, where the $\overline{CAS}$ signal goes active to begin a read operation or perform a write operation.

The fourth is the critical timing sequence between $\overline{RAS}$ going active and the first shift clock (SAS) active edge which occurs in the row transfer of the initial access of an instruction burst.

The fifth timing is that of a burst access. This is the timing between SAS going high and a valid instruction being transferred to the processor. This time is designed to fit within two clock cycles.

The combination of a decode cycle followed by the row-address cycle and by a data-read access time defines a five cycle read of data. Subsequent data-read operations may be six cycles long if the next data address appears during the PC2 precharge state.

For a data write, the access time is made up of a decode cycle followed by a data write, in which DRDY is active in the second or third cycle after decode. The write operation thus takes three to four cycles. Subsequent data-write cycles may take up to six cycles to complete if the next address appears during the data WAIT state, i.e., during the memory-precharge time. A read following a write could take up to eight cycles to complete if it started during the precharge time of the previous access.

The initial access time of an instruction access is made up of a decode cycle, plus a row transfer sequence, plus the first burst access. This totals 6 cycles. Again this could be extended up to nine cycles if the instruction address were to appear during the precharge time following a data write operation or up to seven cycles if it followed a data read.

After the initial access, all burst instruction accesses use a 2-clock-cycle timing. Because two memory banks are interleaved, the apparent access time from the viewpoint of the system bus is only one cycle per burst access following the initial access.

Decode Timing

Within the decode cycle the address timing path is made up of:

- The Am29000 clock to address & control valid delay of 14 ns,
- Address decode logic PAL delay of 10 ns,
- And the set-up time of the RAS PAL, 10 ns.

Assuming D-speed PALs, those times total 34 ns as shown in Figure 7-13.

Also, within the decode cycle time is the control signal to response signal path. In fact this timing path is present in every cycle in the sense that the memory response signals must be valid in every clock cycle. This delay path is made up of:

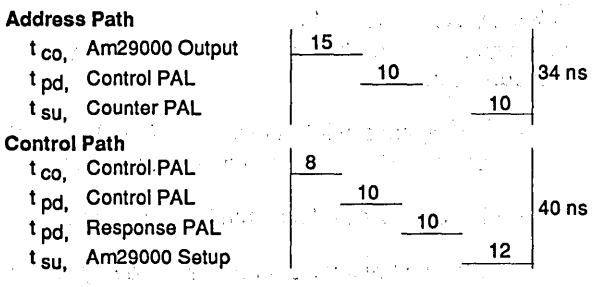
- Clock-to-output time of registers within the control logic state machine PAL, 8 ns;
- Propagation delay of the control logic PAL, 10 ns;
- Propagation delay of a logical OR gate on the response signals from each memory block, 10 ns;
- And control signal set-up time of the processor, 12 ns.

Again assuming D-speed PALs, these delay path times total 40 ns.

Row Address Timing

Referring to Figure 7-14, within the row-address cycle, the $\overline{RAS}$ line goes low which initiates a time delay signal which later causes the address multiplexer to change from the row to the column address.

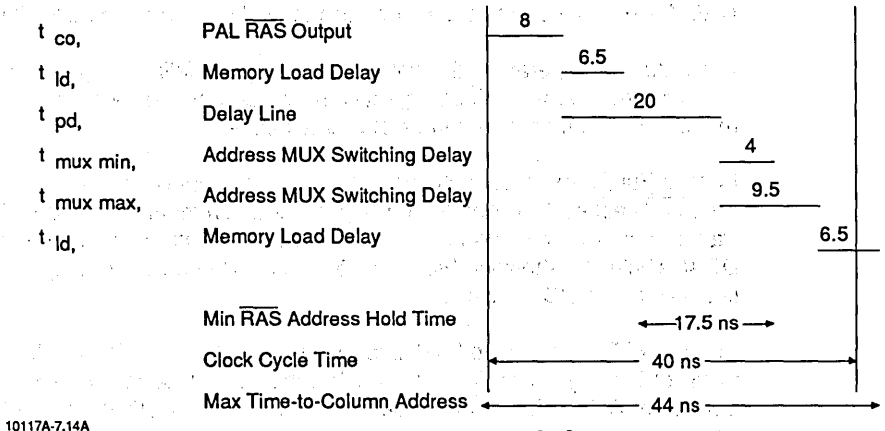
Figure 7-13



10117A-7.13A

VDRAM Interleaved Bank Memory Decode Cycle

Figure 7-14



10117A-7.14A

Row Address Timing

This delay path is made up of:

- Clock-to-output time of $\overline{RAS}$ signal registers within the control-logic state machine PAL (8 ns) plus an added delay due to capacitive and inductive loading by the memory array of the PAL outputs. Since this load is in excess of standard data sheet test loads, the equations in Appendix A are used to estimate the added delay. The estimated delay is 6.5 ns. This is added to the 8 ns (standard 50 pF load) delay of the $\overline{RAS}$ line for a total of 14.5 ns worst case.

- Mux switch control signal delay path, which runs in parallel with the memory $\overline{RAS}$ delay just described. This mux signal delay is made up of the clock-to-output delay of a lightly loaded $\overline{RAS}$ signal (8 ns) plus the delay line time (20 ns);

- Minimum and maximum switching time of the address multiplexer, 4 ns to 9.5 ns, plus added delay for heavy loading (same as calculated above), 6.5 ns.

Thus the memory $\overline{RAS}$ signals are stable no later than 14.5 ns into the cycle and the address mux output can change no sooner than 32 ns (assuming $\overline{RAS}$ outputs from the same PAL will always have similar delays). So the address hold time after $\overline{RAS}$ is 17.5 ns. This works out to satisfy the 15 ns of required hold time of address after $\overline{RAS}$ goes active. Also the column address is settled by no later than 44 ns in to the cycle. So, the column address will be set up prior the $\overline{CAS}$ going active in the next cycle.

$\overline{CAS}$-to-Data Ready

In a data read operation the Column Address Strobe ($\overline{CAS}$) signal-to-end of DRDY cycle is made up of:

- $\overline{CAS}$ signal clock-to-output time (8 ns) plus added delay for heavier-than-normal output loading, as determined above, (6.5 ns).

- Memory access delay from $\overline{CAS}$ (60 ns).

- Data bus transceiver propagation delay (10 ns).

- Processor set-up time (6 ns).

This totals 90.5 ns, which translates into just a little more than two cycles. Therefore DRDY is not made active until the second cycle following the DCAS state.

In a data-write operation, the data is written by the falling edge of $\overline{CAS}$. But the data hold time relative to $\overline{RAS}$ going active may also have to be satisfied before DRDY is made active to free the address and data buses.

For the Fujitsu memory, only the data hold time relative-to-$\overline{CAS}$ is required, this is 30 ns after $\overline{CAS}$ active. The Am29000 will provide a minimum of 4 ns data hold time. The data transceiver will provide an additional minimum of 4 ns hold time beyond the end of the DCAS cycle. As shown in Figure 7-15, these will ensure meeting the hold time if DRDY is active in the DCAS cycle.

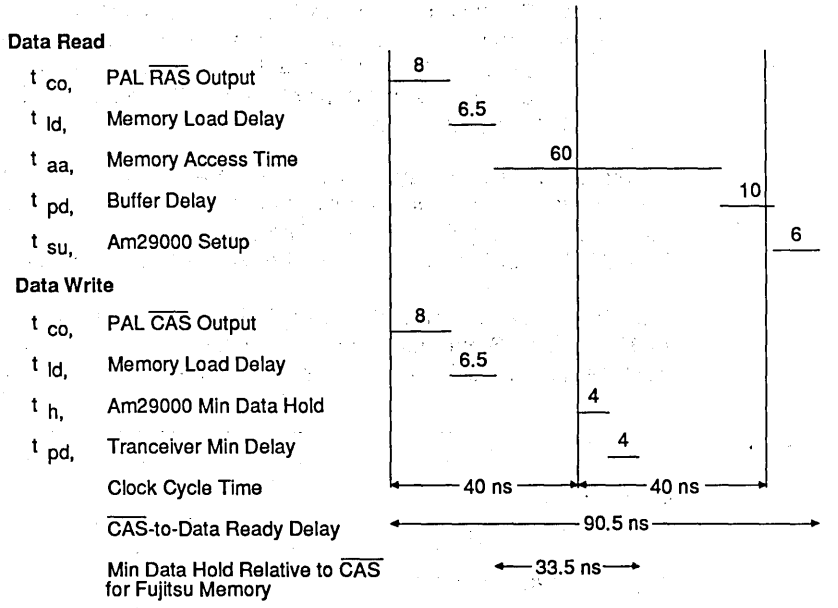
For the NEC memory the hold time relative to $\overline{RAS}$ is the longer delay path, this is 95 ns from $\overline{RAS}$ going active. This implies that the data must be held 29.5 ns into the WAIT state after DCAS. So, in this case DRDY must not go active until the WAIT state after DCAS as shown in Figure 7-16.

$\overline{\text{RAS}}$-to-Shift Clock Timing

Referring to Figure 7-16, in order to maintain a 6 cycle initial instruction access time only 3 cycles can be used for the timing of signals between RAS and SAS. In that time the TR signal must be active for 90 ns to 100 ns after RAS and it must be inactive 25 ns to 10 ns before SAS goes active, depending on the memory used. That is to say the least, a tight fit. The timing is as follows:

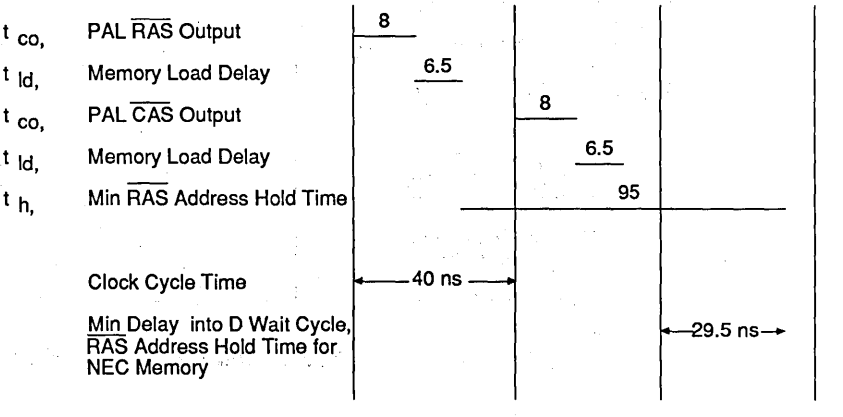
- Clock-to-memory $\overline{\text{RAS}}$ delay (8 ns) plus the added delay for heavy output loading of 6.5 ns for a total of 14.5 ns.

Figure 7-15



$\overline{\text{CAS}}$ to Data Ready Timing

Figure 7-16



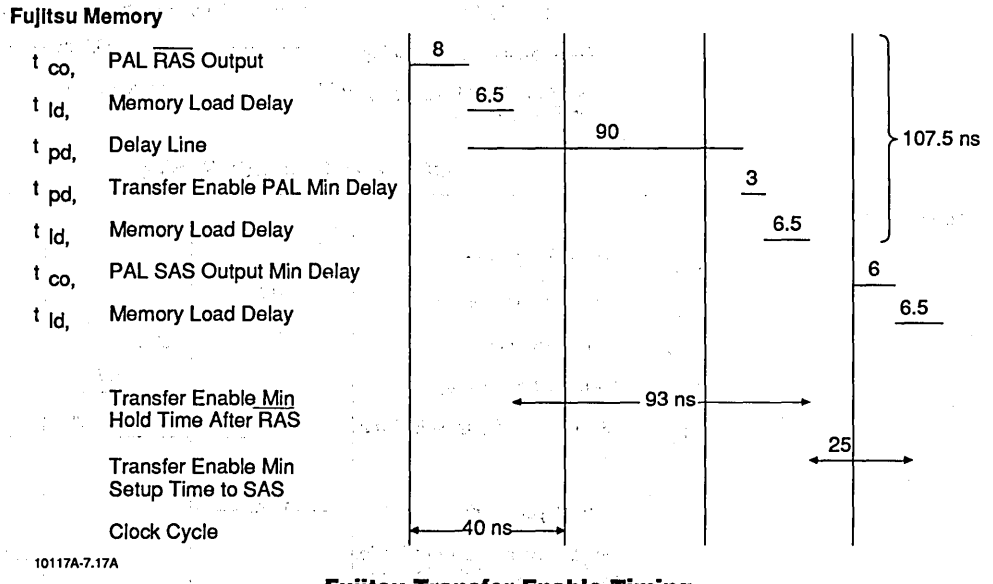
NEC Memory Write Data Hold Time

- In parallel with the memory $\overline{RAS}$, a separate copy of $\overline{RAS}$ which is not loaded by the memory array is used to drive the delay line which determines the end of the TR signal. Its clock-to-output delay time is 8 ns.
- Delay line time of 90 or 100 ns.
- Propagation delay of the PAL which generates TR from the output of the delay line is a minimum of 3 ns and a maximum of 10 ns plus an output loading delay of 6.5 ns.
- The SAS output is combinatorial and is dependent on input signals that are registered. So its minimum delay is the minimum clock-to-output delay plus the minimum propagation delay of a D-speed PAL plus the added delay for memory loading (3 ns + 3 ns + 6.5 ns = 12.5 ns). Its maximum delay consists of 8 ns of clock-to-output delay, 10 ns of propagation delay and a loading delay of 6.5 ns for a total delay of 24.5 ns.

Assuming minimum delays in the TR and SAS signals and maximum delays in the $\overline{RAS}$ signals, the hold time for TR will just be met for either the NEC or Fujitsu memories. For the Fujitsu memory the TR setup time before SAS will also just be met as shown in Figure 7-17; For the NEC memory there is 5 ns of margin as shown in Figure 7-18.

The above relies on the fact that all RAS outputs are implemented in the same PAL and that TR and SAS outputs reside in the same PAL. The PAL outputs for related signals will thus always track each other as to minimum or maximum delay times.

Figure 7-17



Fujitsu Transfer Enable Timing

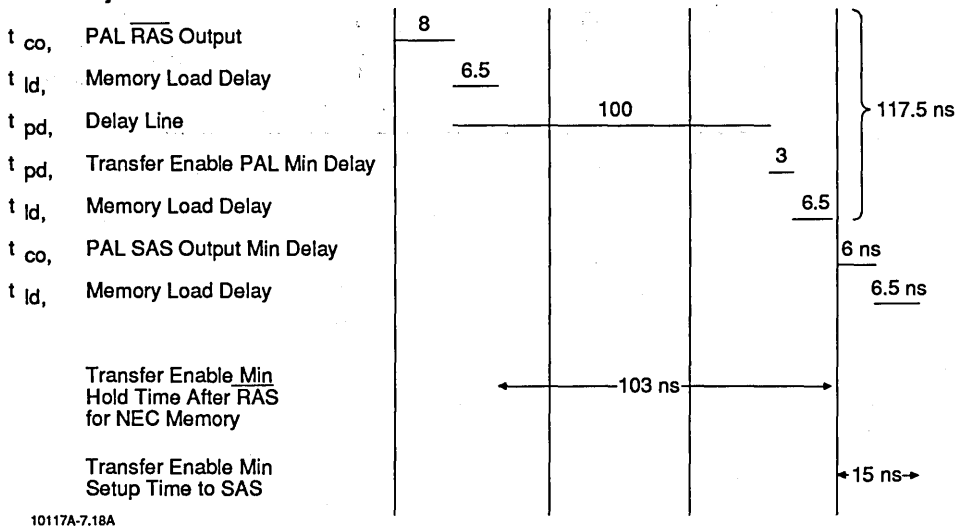
Burst Timing

Within the burst access cycle the address to data path timing is determined by:

- The clock-to-output time of Q02E (8 ns for a D-speed, PAL)
- Propagation delay of SAS PAL (10 ns) plus added delay for heavy capacitive and inductive load as was done for the RAS line. The same derating delay of 6.5 ns will apply.
- Memory access time for serial port, 40 ns,
- Data buffer delay (F244 = 6.2 ns),
- And the processor set-up time (6 ns).

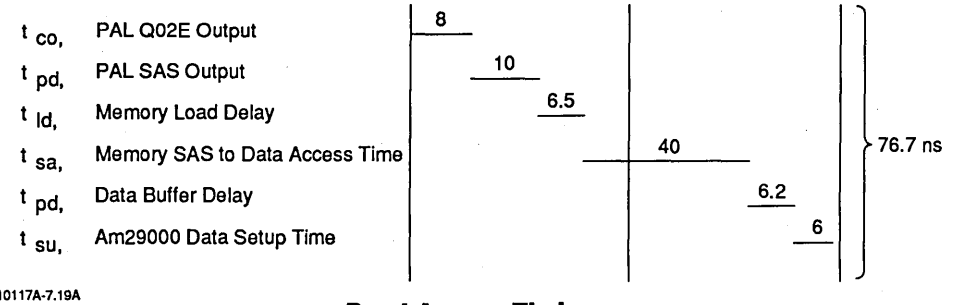
Those delays produce a worst-case total 76.7 ns as shown in Figure 7-19

Figure 7-18 NEC Memory



NEC Transfer Enable Timing

Figure 7-19



Burst Access Timing

INTER-CYCLE TIMING

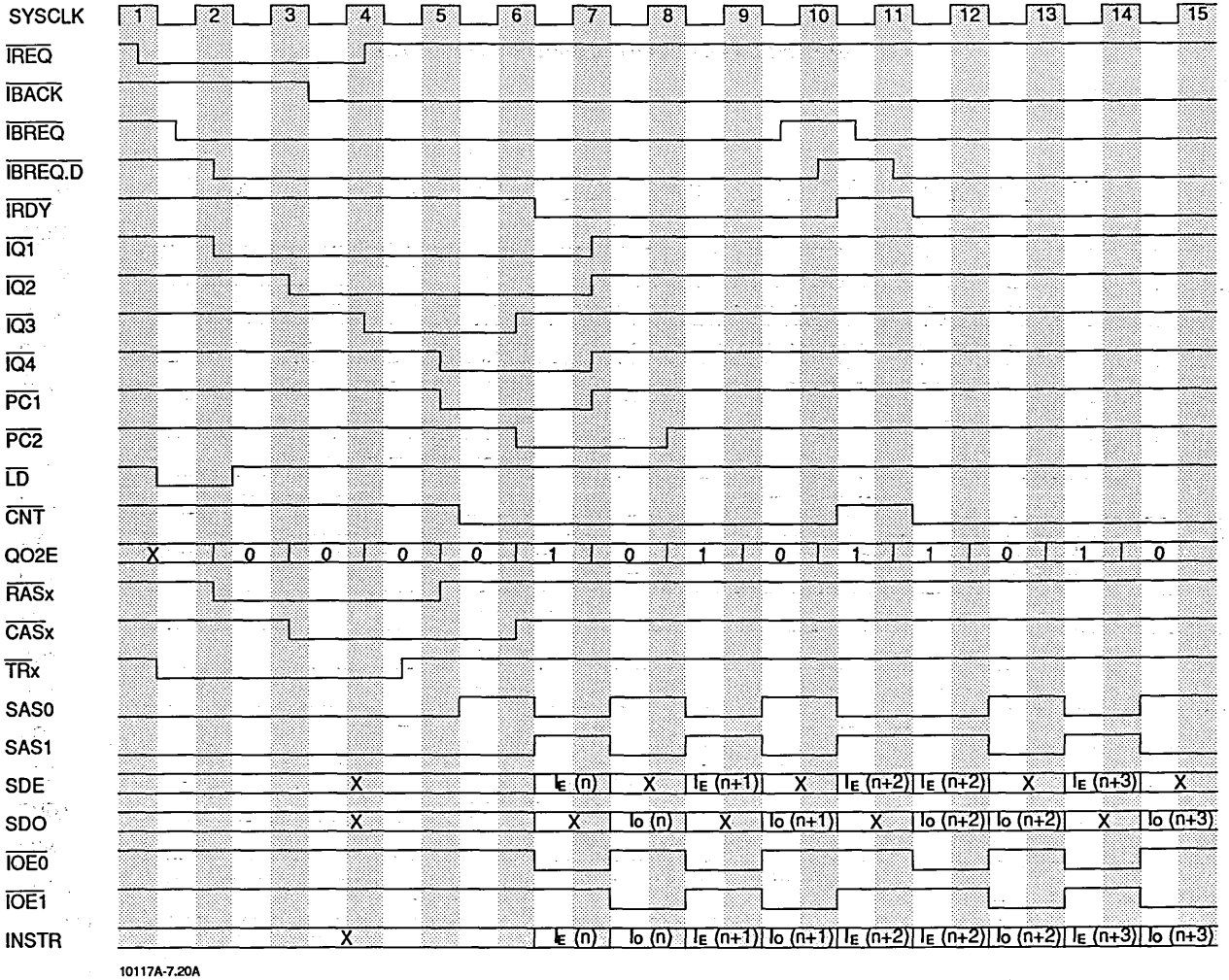
Inter-cycle timing for instruction, data read and data write cycles are provided in Figures 7-20 through 7-22.

PARTS LIST

The part list for the Am29000 Interleaved Video RAM Interface is provided in Table 7-1.

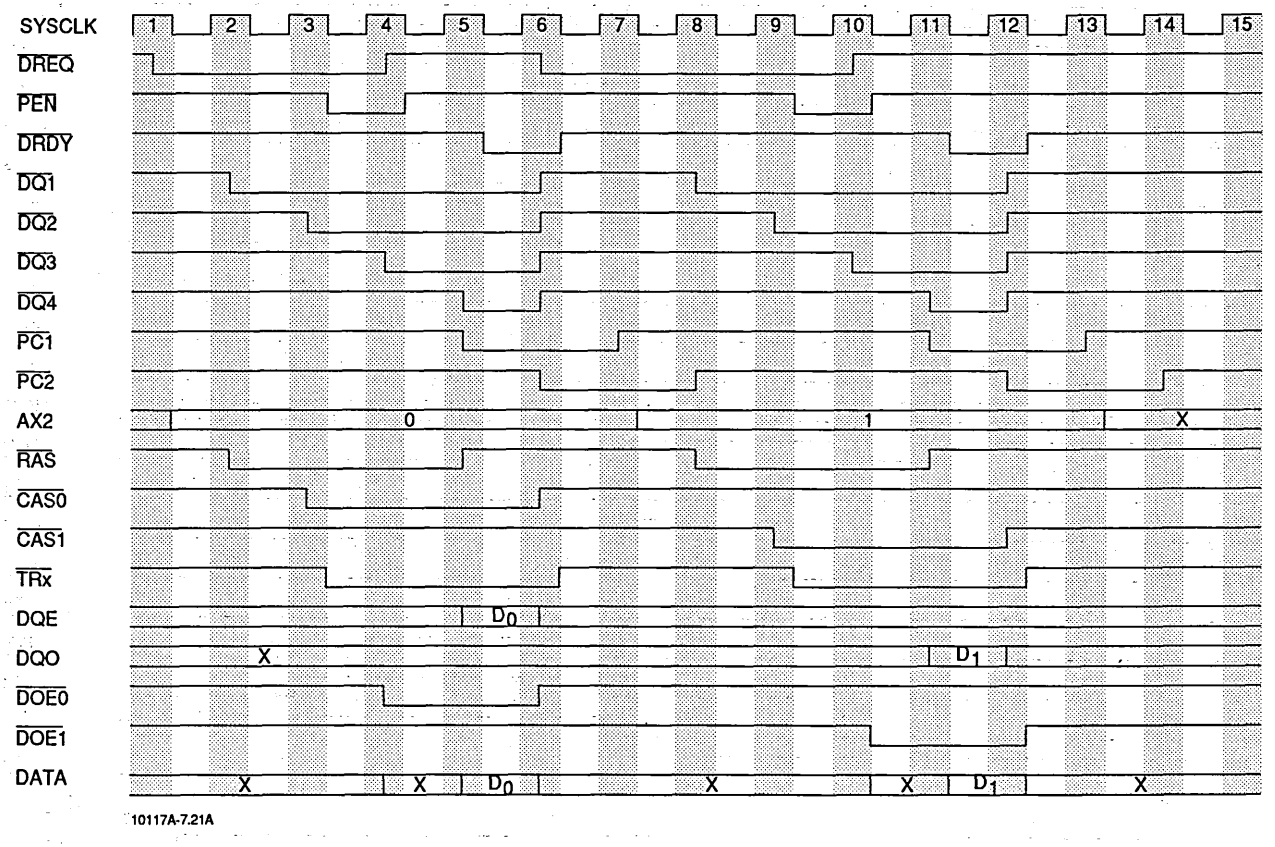
Table 7-1 **Am29000 Interleaved Video RAM Interface Parts List**

Item No.	Quantity	Device Description
U1	1	AmPAL18P8B
U2	1	AmPAL20L8B
U3	1	AmPAL22V10A
U4	1	74F175
U5-U8	4	74F157
U9-U11	3	AmPAL16R4D
U12,U13	2	AmPAL16R6D
U14	1	AmPAL16L8D
U15-U30	16	MB81461-12 or PD41264
U31-U38	8	Am29C863
U39-U46	8	74F244
U47	1	XTTLDM-100
	47 pkgs	



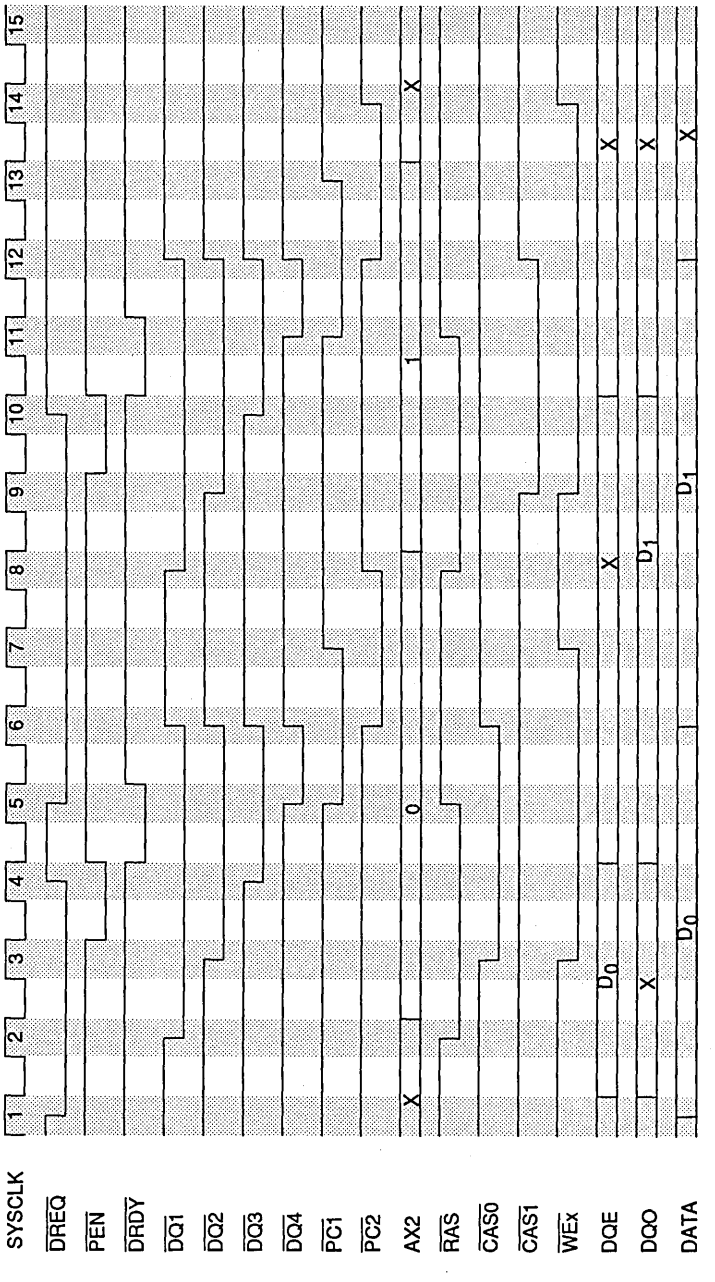
10117A-7.20A

VDRAM Data Write Timing (20 ns/Division)



VDRAM Data Read Timing (20 ns/Division)

Figure 7-22



VDRAM Instruction Timing (20 ns/Division)

CHAPTER 8
Memory Example Comparisons

Introduction 8-1
Memory Block Address Range 8-1
Memory Board Space Consumption 8-1
Memory Power Consumption 8-3
Memory Cost 8-4
Memory Access Speed 8-4
System Benchmark Performance 8-7
Summary 8-7

Chapter 8

MEMORY EXAMPLE COMPARISONS

This chapter compares each of the example designs presented in this handbook. The areas of comparison are given below.

- Memory block address range.
- Memory board space consumption.
- Memory power consumption.
- Memory cost.
- Memory access speed.
- System benchmark performance.

The ground rules for each comparison are discussed and the chapter summary provides a table that shows all the results. Consistent ground rules are used in the calculations. Different ground rules will give different results; however, the ratios will remain roughly the same.

MEMORY BLOCK ADDRESS RANGE

The non-interleaved SRAM example of Chapter 4 provides a single bank of 16K words for the instruction block and a similar bank for the data memory block. The bank interleaved SRAM example of Chapter 5 provides dual 64K-word banks in the instruction and data memory blocks. So, the instruction and data blocks each contain 128K words of memory.

The SCDRAM example of Chapter 6 provides dual 1M-word banks in each memory block. So, the instruction and data blocks each contain 2M words of memory.

The VDRAM example of Chapter 7 provides dual 64K-word banks for a common-instruction and data-block address space. So, the combined instruction and data block contains 128K words of memory.

MEMORY BOARD SPACE CONSUMPTION

The consumption of board space is estimated by the quick and crude method of dividing the total pin count by a pins-per-square inch density factor. Accuracy of this method is open to question but the intent is to provide a quick and consistent way of indicating the relative board space required by each design.

Tables 8-1 through 8-4 show the parts list and pin count for each design. Those tables are used as the basis of comparison. Each table lists only the parts needed to implement the instruction memory block (except the VDRAM design). For ease of calculation, the data-memory-block is assumed to be identical to the instruction block; therefore the total pin count is double that shown in each of the Tables 8-1 through 8-3. The value in Table 8-4 is not to be doubled since the VDRAM design supports both instruction and data memories in a single memory block.

The density factor is 40 pins per square inch. Thus, the total square inches estimated for each design is shown in Table 8-5.

Table 8-1

Am29000 High-speed Static RAM Interface Parts List

Qty	Device Description	Pins/ Device	Pins Total	Power/ Device mW	Power Total mW	Cost/ Device $	Cost Total $
1	AmPAL16R4D	20	20	945	945	5.00	5.00
1	74F175	16	16	187	187	.60	.60
1	AmPAL16L8D	20	20	945	945	5.00	5.00
3	AmPAL16R6D	20	60	945	2835	5.00	15.00
1	74F32	14	14	51	51	.50	.50
8	P4C1982-20	28	224	550	4400	15.00	120.00
4	IDT74FCT244	20	80	345	1380	1.50	6.00
8	IDT74FCT244A	20	160	345	5520	2.00	16.00
27			594		16263		168.10

Table 8-2

Am29000 Medium-speed Bank Interleaved Static RAM Interface Parts List

Qty	Device Description	Pins/ Device	Pins Total	Power/ Device mW	Power Total mW	Cost/ Device $	Cost Total $
2	AmPAL16L8D	20	40	945	1890	5.00	10.00
4	AmPAL16R4D	20	80	945	3780	5.00	20.00
1	74F175	16	16	187	187	.60	60
2	Am29823A	24	48	550	1100	2.00	4.00
2	AmPAL16R6D	20	40	945	1890	5.00	10.00
64	IDT7187S-55 or CY7C187-55	22	1408	660	42240	5.00	320.00
8	Am29825A	24	192	517	4136	2.00	16.00
16	74AS244	20	320	495	7920	1.50	24.00
99			2144		63143		404.60

Table 8-3

Am29000 Interleaved Dynamic RAM Interface Parts List

Qty	Device Description	Pins/ Device	Pins Total	Power/ Device mW	Power Total mW	Cost/ Device $	Cost Total $
1	AmPAL16L8B	20	20	945	945	5.00	5.00
1	AmPAL22V10A	24	24	990	990	6.00	6.00
2	AmPAL20L8B	20	40	945	1890	3.00	6.00
4	AmPAL16R4D	20	80	945	3780	5.00	20.00
6	AmPAL16R6D	20	160	945	5670	5.00	30.00
2	AmPAL16L8D	20	40	945	1890	5.00	10.00
64	TC511002-100	18	1152	330	21120	25.00	1600.00
1	74F175	16	16	187	187	.60	.60
6	74F158	16	120	83	498	.60	3.60
8	Am29C843	24	192	488	3904	2.00	16.00
16	IDT74FCT244A	20	80	345	1380	2.00	32.00
1	MTTLDL-8	16	16	330	330	5.00	5.00
112			1940		42584		1734.20

Table 8-4

Am29000 Interleaved Video RAM Interface Parts List

Qty	Device Description	Pins/ Device	Pins Total	Power/ Device mW	Power Total mW	Cost/ Device $	Cost Total $
1	AmPAL18P8B	20	20	945	945	3.00	3.00
1	AmPAL20L8B	20	20	945	945	3.00	3.00
1	AmPAL22V10A	24	24	990	990	6.00	6.00
1	74F175	16	16	187	187	.60	.60
4	74F158	16	64	83	332	.60	2.40
3	AmPAL16R4D	20	60	945	2835	5.00	15.00
2	AmPAL16R6D	20	40	945	1890	5.00	10.00
1	AmPAL16L8D	20	20	945	945	5.00	5.00
16	MB81461-12 or PD41264	24	384	523	8368	6.00	96.00
8	Am29C863	24	192	643	5144	2.00	16.00
8	74F244	20	160	495	3960	1.00	8.00
1	XTTLDM-100	16	16	550	550	5.00	5.00
47			1016		27091		170.00

MEMORY POWER CONSUMPTION

The power consumption for each design is estimated by totaling the worst-case power consumption (maximum supply current times maximum operating V_{cc} at 25 MHz signal toggle rate) for all devices.

These power-consumption parameters are not to be considered representative of the power consumption normally expected in these designs. They represent the absolute maximum possible consumption in the extremely unusual event that all devices simultaneously operated at maximum power consumption. These power estimates are used only as a means to consistently determine relative power consumption between the designs.

As was done before in the last section, the values from Tables 8-1 through 8-3 are doubled to estimate the power use for both instruction and data memory blocks. The value of Table 8-4 is not doubled since the VDRAM design supports both instruction and data memories in a single memory block. The estimated total power consumption results are shown in Table 8-5.

MEMORY COST

The cost of a memory system is difficult to estimate because the prices of individual devices change with the market place over time and prices can vary widely depending on the required volume of devices. The prices used for this comparison are rough "ballpark" numbers that were obtained in March 1988 for quantities of 1K per logic device and 10K per memory device.

Tables 8-1 through 8-4 show the estimated cost for each memory block. Table 8-5 summarizes the costs, again doubling the cost of the first three designs to account for both the instruction and data-memory blocks.

MEMORY ACCESS SPEED

The access speed of each design is summarized in Table 8-5.

Non-Interleaved SRAM

The high-speed non-interleaved SRAM design has an initial access time of two cycles (one wait state) and a single-cycle (zero wait state) burst access time for all subsequent sequential accesses. This performance is the same for either the instruction-memory or data memory block.

Bank-Interleaved SRAM

The medium-speed bank-interleaved SRAM design has an initial access time of three cycles (two wait states) and single cycle (zero wait state) burst access. Again this is true for both instruction and data-memory blocks.

SCDRAM

The SCDRAM design provides a basic initial access time of four cycles (three wait states) and single-cycle (zero wait state) burst access. However, with dynamic memories, the initial access time is not always consistent. Dynamic memories introduce some overhead cycles into the normal access sequence. This overhead comes in the form of refresh sequences and precharge time.

The SCRAM requires an average refresh sequence of 5 cycles every 15.6 μs. If a refresh sequence preempts a burst access, that access incurs additional overhead because it is forced to resend an address to re-establish the burst access. This will require a 4-cycle initial access time in addition to the 5-cycle refresh sequence. Depending on how often a burst access is in progress at the time a refresh is required; the refresh sequence could require up to nine cycles out of every 390 cycles (refresh

period in cycles = 15.6 μs/40 ns = 390). Thus, refresh can cost up to 2.3% of the overall memory performance when the memory is constantly being accessed. Refresh sequences that occur when the memory is otherwise not in use cost nothing, since the refresh does not contend with system use of the memory.

Precharge overhead is required each time a new memory request and address are presented to the memory. The new address is presumed to access any random location and thus requires a full row and column address sequence to initiate the new access. Whenever one row address is changed to a new row address there is a required 2-cycle precharge delay between the end of the first access and the beginning of the second.

In cases where a previous access has ended one or more cycles before a new access begins there is no precharge penalty since the precharge time between accesses has already been satisfied. If a previous access has not ended at the time a new address is presented, the new access must be delayed during the required precharge time. This situation is very common. From the view point of the memory, this is almost always the situation if burst accesses are assumed to be the normal mode of access. The Am29000 bus protocol provides notice of a burst-access cancellation (end) by the appearance of the next memory-request address. Until the new request appears, a memory system must assume that any burst access is either active or suspended (but not ended). Therefore, for the instruction-memory block, where burst accesses are almost always used by the Am29000, the memory control logic is designed to always assume burst accesses. Virtually every new memory request (initial access) incurs the 2-cycle precharge delay in addition to the normal initial access delay. Note that since the Idle state serves both as a precharge cycle and an address decode cycle, the precharge time is overlapped with the first cycle of the new initial access. The total initial access time is thus five cycles in the above case.

The only exceptions to this occur when a different instruction memory block is addressed and the instruction memory block of interest recognizes the address of a different block as the end of any suspended burst access. This recognition of the end of a burst allows the memory of interest to go through the precharge delay prior to the beginning of any subsequent access. Thus, any following access to the memory block of interest will only incur the basic 4-cycle initial access delay.

The data-memory block can take advantage of the fact that the Am29000 processor never converts a simple or pipelined access into a burst access. Any burst access is indicated from the very beginning of the memory request. Also, a data burst access is never suspended. Together these facts indicate that a data memory can always recognize the end of an access as signaled by Data Burst Request ($\overline{\text{DBREQ}}$) being inactive. This allows the data-memory logic to end an access and satisfy the precharge delay, in many cases, before a new access request appears. Therefore the data-memory block can most often incur only the normal 4-cycle initial access delay without any precharge overhead.

The bottom line of this whole dissertation is that the instruction-memory block almost always incurs precharge delay in addition to the initial access delay; therefore the typical access time is five cycles. The data-memory block can however avoid the precharge overhead in most cases the typical initial access time is four cycles. Finally, for either memory block, burst access cycles are always single cycle.

A valuable enhancement for the above design would be the addition of a row-address comparator and a modification to the control state machine to allow the memory interface logic to recognize when a new memory request address lies in the row currently being accessed. Remember that with SCDRAM, access to any random location within the currently addressed row requires only that the column address be changed. There is no precharge or row address transfer time required. When the memory interface logic compares the current row address with the new request address and determines a match, the control state machine can pass the new column address on to the memories and completely avoid any need to precharge or go through the normal initial access sequence. This means that for any access within the current row, the initial access time can be reduced to three cycles: One cycle to recognize the situation and two cycles to access the first word. Again all burst accesses would still be single cycle. The preemption for Refresh would guarantee that the maximum $\overline{RAS}$ pulse width would not be violated.

Although this design option was not implemented in the SCDRAM design shown in Chapter 6, the design changes required have been estimated as the addition of two 74AS866 comparators and one AmPAL16R4. The performance of a design with row comparators was simulated and is included in the final summary, Table 8-5.

VDRAM

The VDRAM design has a basic initial access time of six cycles (five wait states) for instructions and five cycles (four wait states) for data read. Data-write initial access time is three or four cycles depending on the particular memory used to implement the design. The burst access time for instructions is single cycle and no burst accesses are supported for data.

Like the SCDRAM described in the last section the VDRAM design requires similar overhead cycles for refresh and precharge functions. The overhead for refresh affects data accesses much more often than instruction accesses. This is because the shifter port used for instructions on a VDRAM operates independently of the data I/O port. Once an instruction access is initiated, subsequent burst accesses require no interaction with the data I/O port. This means that refresh sequences that involve the data I/O port can go on in parallel with instruction accesses. It is only when a new instruction request appears during a refresh sequence that the instruction request is delayed by the refresh activity. The refresh interval is 390 clock cycles for the VDRAM and a refresh sequence requires six cycles; so, the maximum percentage of cycles that may be lost to refresh overhead is 1.5%.

The VDRAM requires a precharge time of three cycles between the end of one access and the beginning of another. In cases where a previous access has ended two or more cycles before a new access begins, there is no precharge penalty since the precharge time between accesses has already been satisfied. As noted for the SCDRAM design, the Idle state serves both as a precharge cycle and an address decode cycle. The precharge time is overlapped with the first cycle of the new initial access; thus only a two cycle space between accesses is required.

In the situation that a previous access has not ended at the time a new address is presented, the new access must be delayed during the required precharge time.

There is one additional overhead delay in the event that a new memory request follows a data write operation before the write and precharge sequence is complete. When this happens the new access will be delayed by up to three cycles.

SYSTEM BENCHMARK PERFORMANCE

Advanced Micro Devices provides an architectural simulator program for evaluating the Am29000. The simulator executes compiled or assembled code and provides a detailed analysis of the Am29000 performance for that code. It provides the ability to define the access time expected from instruction memory, ROM, and data memory. This allows performance on standard benchmark programs to be evaluated across a wide range of performance variations in the Am29000-system memory. The simulator is limited with regard to DRAM or VDRAM memory designs , since it is unable to simulate refresh or precharge delays. Therefore, the actual performance of dynamic-memory-based systems will be slightly less than that indicated by the results of simulation. In the case of the SCDRAM example, some of this error in reported performance is compensated for by listing the initial access time as five cycles for instruction accesses. That access time includes the normal precharge delay that the SCDRAM memory experiences.

The benchmark chosen for comparison of the example memory designs is called Dhrystone version 1.1. This program is designed as a statistically correct mix of instructions that is representative of a wide range of frequently executed programs. This benchmark program has been executed on virtually all microprocessor systems sold, so comparison with competing microprocessor solutions should be relatively easy.

The Dhrystone 1.1 benchmark program was compiled with the High C* compiler for the Am29000. The results of benchmark execution are shown in Table 8-5.

SUMMARY

Table 8-5 brings together a summary of all the features and performance factors for each of the example designs. In addition to the four designs shown in Chapters 4 through 7, two other variations are estimated and shown.

As a comparison to the SCDRAM design, a column is added to show a SCDRAM design including row-address comparators.

For VDRAM, a column is added to show how newer 1M bit density VDRAMs would compare with the design based on the older technology 256K-bit VDRAMs. The 1M-bit VDRAMs are assumed to require an 18-pin package, have power consumption equal to the 256K-bit VDRAMs, and to cost $50 each (double the assumed price of SCDRAM).

* Trademark of Metaware Inc.

Table 8-5

Memory Design Example Feature and Performance Summary Showing System Totals for Instruction and Data Memory

Comparison Item	Design Example: High Speed SRAM	Medium Speed SRAM	SCDRAM	SCDRAM With Row Add Compare	VDRAM 256K Bit Type	VDRAM 1M Bit Type
Total Words of Memory	32K	256K	4M	4M	128K	512K
Board Space Consumption sq in.	29.7	107.2	97	100.8	25.4	26.2
Board Space Consumption words/sq in.	1103	2445	43240	41610	5160	20010
Power Consumption mW	32526	126286	85168	88753	27091	27091
Power Consumption words/mW	1.007	2.08	49.25	47.26	4.84	19.35
Cost $	336	809.2	3468.4	3488.4	170	874
Cost words/$	97.5	323.9	1209.3	1202.4	771	600
Access Speed in Cycles						
Instructions						
Initial	2	3	5	3 to 4	6	6
Burst	1	1	1	1	1	1
Data						
Initial	2	3	4	3 to 4	5	5
Burst	1	1	1	1	NA	NA
Benchmark Performance						
dhrystones/s	37203	32271	28108	31183	21946	21946
MIPS	19.4	16.87	14.71	16.31	11.53	11.53

As expected, the SRAM designs provide the best performance while consuming the most power and board space per word of memory.

SCDRAM provides the highest density, lowest power, and lowest cost-per-word memory system with only a 25% performance reduction as compared with the high speed SRAM design. When row-address comparators are included, the performance jumps to within 16% of the high-speed SRAM design and within 3.3% of the medium speed SRAM design.

The VDRAM design shows a lower density than SCDRAM even when comparing designs with equal bit-density memory devices. This is mainly due to the much higher ratio of control logic to memory devices involved in the specific VDRAM example design. Since VDRAMS have a "by 4" organization, far fewer memory devices are needed per bank of memory but the number of memory control devices remains nearly the same for one to several banks of memory devices. For a design of equal system-memory size (same number of memory devices), the control logic would become a much lower percentage of the overall device count in a VDRAM design. For equal-bit-density memory devices, i.e. 1M-bit SCDRAM vs 1M-bit VDRAM, and equal memory-system size, the board-space density of the SCDRAM and VDRAM designs should be more closely matched with VDRAM having an advantage due to simpler and smaller control logic.

The primary advantage of VDRAM is in the simpler control and interface logic vs any equivalent size SCDRAM design. This is especially true when the system performance requirements can be relaxed to slow the clock rate enough that the VDRAM shifter port can keep up with the Am29000 cycle rate without the use of dual bank memory-system design.

A further advantage is the ability to make more efficient use of a common instruction and data memory address space, thus, potentially reducing overall memory-system size requirements. At 11.5 MIPS and 21946 dhrystones the VDRAM still provides very respectable performance.

Bottom line: the Am29000 sustains the best performance in town with high-speed memories and maintains high performance when connected directly to low-cost, high-density, dynamic memories.

Expensive and complex cache memory support can be avoided entirely while sustaining performance well beyond that available from other microprocessor solutions.

That's the price/performance advantage unique to the Am29000.

CHAPTER 9

Am29000 Dhrystone 1.1 Memory Benchmarks 9-1

Am29000 DHRYSTONE 1.1 MEMORY BENCHMARKS

by Drew Dutton, Southwest Area Technical Manager

The Am29000 processor has been specifically designed to reduce the cost of memory necessary to sustain the bandwidth requirements of near single-cycle performance. Such techniques as pipelining accesses and banking or interleaving memory have been used throughout the years to improve system performance and both these techniques are available with the Am29000. This chapter is intended to demonstrate the wide range of memory speeds that still provide the necessary performance level for a system as well as pointing out the importance of memory issues other than access speed alone.

Table 9-1 contains the simulated performance of different memory speeds and interfaces using the Dhrystone 1.1 benchmark compiled on the High C* compiler for the Am29000. With 4-cycle first access memory, 33,471 Dhrystones and 17.49 MIPS performance can still be achieved. The range in performance runs from 41,920 Dhrystones and 21.82 MIPS to 10,550 Dhrystones and 5.56 MIPS. The lowest performance was not with the slowest memory but with only simple memory accesses allowed. In general, the most significant changes in performance were due to memory interface changes and not memory speed changes.

All of the benchmark information was gathered using the Advanced Micro Devices Am29000 Architectural Simulator Version 4 running on an IBM-PC/AT with 640K bytes of memory. This simulator models the complete behavior of the Am29000 processor and has been verified against actual hardware. Am29000 memory is mapped into the IBM-PC memory and its speed is modeled with user-specified parameters. The results in Table 9-1 reflect data gathered by changing these memory parameters and re-running the Dhrystone 1.1 benchmark for each unique memory configuration. Read and write timing were assumed to be the same. None of the memory models use a cache, but Static Column DRAM(SCDRAM) with address comparators is simulated. The simulator does *not* simulate any refresh or pre-charge of DRAMs. Therefore, the actual performance of a DRAM-based system would be slightly lower than that simulated.

To read the benchmark table, first note the number of Dhrystones per second. This is the measure of performance provided by a particular memory architecture. After listing the number of clock cycles necessary to execute 50 passes through the Dhrystone loop, the actual speed is given for the three different memories in the simulated system. These memories are Instruction memory, ROM memory and Data memory. Memory speed is listed in system clock cycles. The Dhrystone number assumes that each of these clock cycles is 40 ns and that the system clock is 25 MHz. Although faster versions of the Am29000 are now available, this was the basis for performance measurements.

For each memory, there are several parameters listed. First, are the number of clock cycles necessary to complete a simple, non-burst, non-pipeline access. For example, if the instruction memory was able to respond in 120 ns (after taking into consideration 29000 timing parameters) the memory would be listed as three cycles for a simple

* High C is a trademark of MetaWare Inc.

access (two wait states). If it were 180 ns, the memory would have been listed as four cycles for a simple access (three wait states). If the memory system can provide data in bursts, then the speed of burst access is listed by first stating the number of clocks necessary to initiate a burst and then the number of clocks for each 32-bit word during the burst. The time to do the first burst access is the same as the time necessary to do a simple access in all the examples shown and is thus listed in the same column as a simple access. Subsequent burst accesses are always one cycle for the examples shown.

If the memory is a SCDRAM, then it is possible to have faster access when within a column. Therefore, the speed of a first access within a static column and the size of the static column (in 32-bit words) are listed for this type of memory in the table.

The access speed of Instruction memory is listed first; ROM, which cannot burst in this version of the simulator, then Data memory follow. After the speed of the memories is listed, the number of system clock cycles, the number of Am29000 instructions executed and the resulting MIP rate are shown.

Notes And Conclusion

Although the highest performance was gained through the use of zero-wait-state memory designs, the huge cost differential between these designs and designs utilizing one wait state with pipelined and burst accesses makes it clear that a more optimal cost/performance trade-off exists using slower memory with a more sophisticated interface. For the Dhrystone 1.1 benchmark compiled on the pre-release version of the MetaWare High C compiler, perhaps the best cost/performance trade-off exists with a SCDRAM design. The three-wait-state DRAM design, using one-wait-state access when within a static column, provides 33,471 Dhrystones/second and 17.49 MIPS. This same DRAM design without pipelined access on the data bus provides 29,630 Dhrystones/second.

Support for single-cycle burst is important to sustain single-cycle execution whenever possible. Pipelining on the data bus is an important performance aid due to a high number of loads followed by branches produced by the compiler. A different benchmark or different compiler may not have such a strong need for data pipelining. It should also be noted that this benchmark does not use the Load-Multiple or Store-Multiple instructions and therefore never does a data burst.

The Am29000 sustains a very high MIP/Dhrystone rate when provided with single-cycle burst on the instruction bus and pipelined accesses on the data bus. Even with 6-cycle first-access memory, the Am29000 can provide over 30,000 Dhrystones and 15 MIPs!

Table 9.1 Statistics of Dhrystone 1.1 Simulation

Dhrystone Performance		Instruction Memory		ROM	Data Memory		Simulation Performance					
Dhrystones per Second	Time for 50 Passes	Simple/ 1st Burst Access	Access Mode	Simple Access	Simple/ 1st Burst Access	Access Mode	User Mode	Supervisor Mode	Total			
	Cycles	Cycles	see note	Cycles	Cycles	see note	Cycles	Cycles	Cycles	Seconds	MIPS	Cycles/Inst
41920	29818	1	burst	1	1	burst	30749	187	30936	0.00123744	21.82	1.15
39698	31487	1	simple	1	1	simple	32406	189	32595	0.00130380	20.71	1.21
37203	33599	2	burst	2	1	pipeline	34594	208	34802	0.00139208	19.4	1.29
35760	34955	2	burst	2	2	simple	35959	208	36167	0.00144668	18.67	1.34
33471	37345	4	burst SC	4	4	burst SC	38368	226	38594	0.00154376	17.49	1.43
32271	38734	3	burst	3	3	pipeline	39800	221	40021	0.00160084	16.87	1.48
31442	39755	5	burst SC	5	5	pipeline SC	40793	232	41025	0.00164100	16.46	1.52
30257	41312	3	burst	3	3	simple	42408	221	42629	0.00170516	15.84	1.58
30104	41522	6	burst SC	6	6	burst SC	42576	244	42820	0.001171280	15.77	1.59
29911	41790	4	burst	4	4	pipeline	42915	247	43162	0.00172648	15.64	1.6
29630	42186	4	burst SC	4	4	simple SC	43252	226	43478	0.00173912	15.53	1.61
26924	46426	5	burst	5	5	pipeline	47639	259	47898	0.00191592	14.10	1.77
26032	48017	4	burst	4	4	simple	49191	247	49438	0.00197752	13.66	1.83
25629	48771	2	pipeline	2	2	pipeline	49955	314	50269	0.00201076	13.43	1.86
23573	53026	2	pipeline	2	2	simple	54251	314	54565	0.00218260	12.37	2.02
22826	54760	5	burst	5	5	simple	56038	257	56295	0.00225180	11.99	2.08
20062	62305	6	burst	6	6	simple	63671	291	63962	0.00255848	10.56	2.37
19047	65626	3	pipeline	3	3	pipeline	67103	437	67540	0.00270160	10.00	2.50
17001	73522	3	pipeline	3	3	simple	75055	437	75492	0.00301968	8.94	2.80
14467	86398	4	pipeline	4	4	pipeline	88175	580	88755	0.00355020	7.61	3.29
13011	96068	4	pipeline	4	4	simple	97923	580	98503	0.00394012	6.85	3.65
11708	106764	5	pipeline	5	5	pipeline	108848	695	10943	0.00438172	6.16	4.06
10550	118476	5	pipeline	5	5	simple	120652	695	121347	0.00485388	5.56	4.49

Note: Access Mode Definitions –

Simple – Simple Accesses only, no Burst or Pipeline Access Support

Pipeline – Simple and Pipeline Accesses only; no Burst Access Support

Burst – Simple, Pipeline, and Burst Access Supported. Pipeline Enable or Burst Acknowledge Signals are active during the first Access cycle. All Burst Accesses beyond the first are completed in a single cycle.

Simple SC, Pipeline SC, Burst SC – Burst, Pipeline, or Simple Access with Static Column DRAM Address Comparators assumes one cycle to decode a hit within a previously accessed Static Column, plus one cycle for the first access. Subsequent burst accesses are single cycle. A Static Column size of 1024 words is assumed.

APPENDIX A
Memory Array Loading Delay Calculations

Overview A-1
Memory Array Models A-1
Determining Memory Load Factors A-2
Layout Effects A-5
Layout Models A-7
Transmission Lines or RLC Circuits? A-7
Transmission Line Model—The Basics A-7
    Memory Specific Example A-9
RLC Model A-15
    Memory Specific Example A-16
Design Example Delay Values A-17
    Non-Interleaved SRAM Example A-17
    Bank-Interleaved SRAM Example A-18
    SCDRAM Example A-18
    VDRAM Example A-18
    Damping Resistors A-19
    Program Listings A-20

Appendix A

MEMORY ARRAY LOADING DELAY CALCULATIONS

OVERVIEW

An array of memory devices may present an inductive and capacitive load much larger and more complex than normally anticipated by most signal driver specifications. Most devices are specified with propagation delays or clock-to-output delays that assume only a local capacitive and resistive load. As shown in Figure A-1, a typical test load circuit would be the driving device output connected to a voltage divider with integrating capacitor (R_1=200 Ω, R_2=390 Ω and C_L=50 pF).

A memory array can easily present a capacitive load of 180 pF to over 400 pF with inductive loading of greater than 170 nH /foot of printed circuit board trace. In addition, depending on the memory layout, the memory array may appear to the driving device like a lumped RLC circuit or like a transmission line.

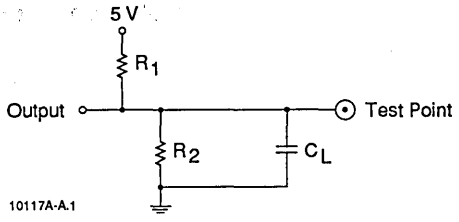
The heavy load presented by a memory array can significantly slow the apparent output-driver switching speeds and may also cause unwanted overshoot or undershoot of the affected signal. Therefore it is important to take into account how a memory array affects the output-delay specifications of any device driving memory-array signals.

MEMORY ARRAY MODELS

Depending on the physical layout of the memory array and on the switching speed of a memory signal driver, a memory array may be modeled by either a lumped RLC circuit or as a distributed RLC network (also called a transmission line) similar to the models shown in Figure A-2.

A transmission line model is appropriate when twice the propagation delay time, from the signal driver to the end of the memory signal trace, significantly exceeds the rise or fall time of the driving signal. In this situation, the distributed nature of the capacitive and inductive loads presented by memories and printed circuit board traces, in effect, prevents the driver from "seeing" the entire load during the signal switching rise or fall time. Changes in voltage and current levels must propagate to the end of the transmission line and any reflections returned back to the source before the driver "sees" the effect of the entire load. In this case the propagation delay of the transmission line

Figure A-1



Typical Signal Driver Test Load

determines the worst-case delay to be added to the propagation delay or clock-to-output delay specified for a memory signal driver.

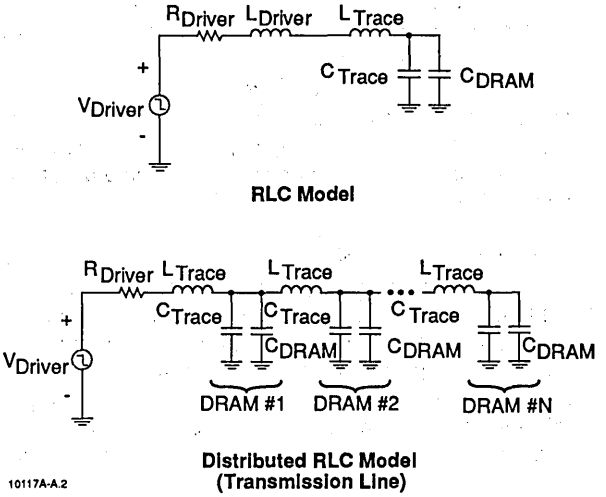
When twice the propagation delay to end of the memory signal trace is significantly shorter than the switching rise or fall time of the signal driver, the memory array is better modeled by a lumped RLC circuit (sometimes called a resonant or tank circuit). This is because the effect of the entire load is seen by the driver as the output is switched and the entire load determines the switching speed of the output.

DETERMINING MEMORY LOAD FACTORS

As shown in Figure A-3, the printed circuit board (pcb) trace capacitance, inductance and impedance is a function of the pcb material and trace dimensions. The primary characteristics are defined as:

Er = Relative dielectric constant of board material. Typical materials are G-10 (Er = 4.7 to 5) and FR-4 (Er = 4.5 to 5.2) with the Er values determined by exact details of board construction and specification of test condition when determining the value of Er. An average value of 5 is used as the value of Er in all calculations shown.

Figure A-2



RLC and Transmission Line Models

w = Width of the trace in inches. 0.01 inches is used as a typical value for memory trace width.

h = Height of the trace above a ground plane in inches. 0.03 inches is used as a typical value.

t = Thickness of trace in inches. 0.003 is used as a typical value for 2-ounce copper traces.

Calculations for trace loads shown in this appendix are for microstrip lines.

Strip line values are significantly different and the references listed at the end of the appendix should be consulted for appropriate calculations.

Characteristic Impedance

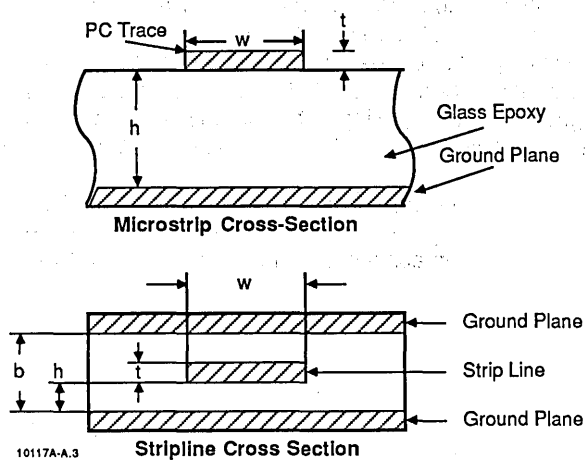
Trace impedance (Zo) is defined as:

$$Zo = \frac{87}{\sqrt{(Er + 1.41)}} \cdot \ln\left(\frac{5.98\,h}{0.8\,w + t}\right)$$

$$= \frac{87}{\sqrt{(5 + 1.41)}} \cdot \ln\left(\frac{5.98\,(0.03)}{0.8\,(0.01) + 0.003}\right)$$

$$= 95.93\ \Omega$$

Figure A-3



PCB Trace Dimensions

Characteristic Propagation Delay

The trace propagation (tpd) velocity is defined as:

$$tpd = 1.017\sqrt{(0.475Er + 0.67)}\ ns/ft.$$

$$= 1.774\ ns/ft$$

Capacitance

The capacitive load comes from the pcb trace capacitance and the input capacitance of each memory device. The input capacitance is typically specified in the memory datasheet. The appropriate value is simply multiplied by the number of memories attached to the signal trace in question. The printed circuit board trace capacitance is determined by the physical characteristics of the board and trace dimensions.

Large area capacitance is determined as:

$$C = \frac{0.224\ Er\ A}{h}$$

Where: C is in picofarads
Er is the board material dielectric constant.
A is the electrode surface area in square inches.
h is the height (separation) of the electrode above the ground plane.

But at the typical dimensions of traces used on a pcb, fringe capacitance becomes a very significant component of the trace capacitance. Calculating this directly is very complex. The trace capacitance (Co) is more easily determined as a function of the trace impedance and propagation delay:

$$Co = 1000(tpd/Zo)\ pF/ft$$

$$= 1000(1.774 / 95.93)$$

$$= 18.5\ pF/ft$$

For transmission line calculations the distributed capacitance (Cd) of the memories is the parameter of interest. This is a value for capacitance per distance along the trace. This is layout dependent and is defined by the spacing between memory packages. For a standard 0.3-inch-wide DIP, it is assumed that memories may be placed along a signal trace at a spacing of two per inch or 24 per foot of trace. Assuming an average input capacitance of 7 pF, the value of Cd is determined as:

$$Cd = \frac{\text{input capacitance pF/memory}}{\text{spacing in feet/memory}} = \frac{7\ pF}{0.0416\ ft} = 168\ pF/ft$$

Inductance

Trace inductance (Lo), like trace capacitance, is rather complex to determine directly. The value of Lo is easier to determine as a function of the trace impedance and capacitance:

$$Lo = (Zo)^2 \, Co \text{ pH/ft}$$

$$= 95.93^2 \, (18.5)$$

$$= 170.18 \text{ nH/ft}$$

Significant inductance is also found in the output and ground pins and bond wires of the signal driver package. These inductances total 15 nH to 25 nH. The driver inductance is worth noting because all the current flowing to-or-from the trace passes through the driver. The memory devices have similar inductance on their inputs but most memories have very low input current loads so that their input inductance will not have a significant effect on the driving signal.

Loaded Trace Impedance

When the capacitance of the memories is added to the characteristic capacitance of the signal trace, the characteristic line impedance (Zo') changes significantly. The new value of Zo is determined as:

$$Zo' = \frac{Zo}{\sqrt{(1 + Cd/Co)}}$$

$$= \frac{95.93}{\sqrt{(1 + 168/18.5)}}$$

$$= 30.21 \, \Omega$$

Loaded Propagation Delay

Similarly the propagation delay is affected when the capacitive load of the memories is taken into account. The new value of tpd is determined as:

$$tpd' = tpd \sqrt{(1 + Cd/Co)}$$

$$= 1.774 \sqrt{(1+ 168/18.5)}$$

$$= 5.633 \text{ ns/ft}$$

LAYOUT EFFECTS

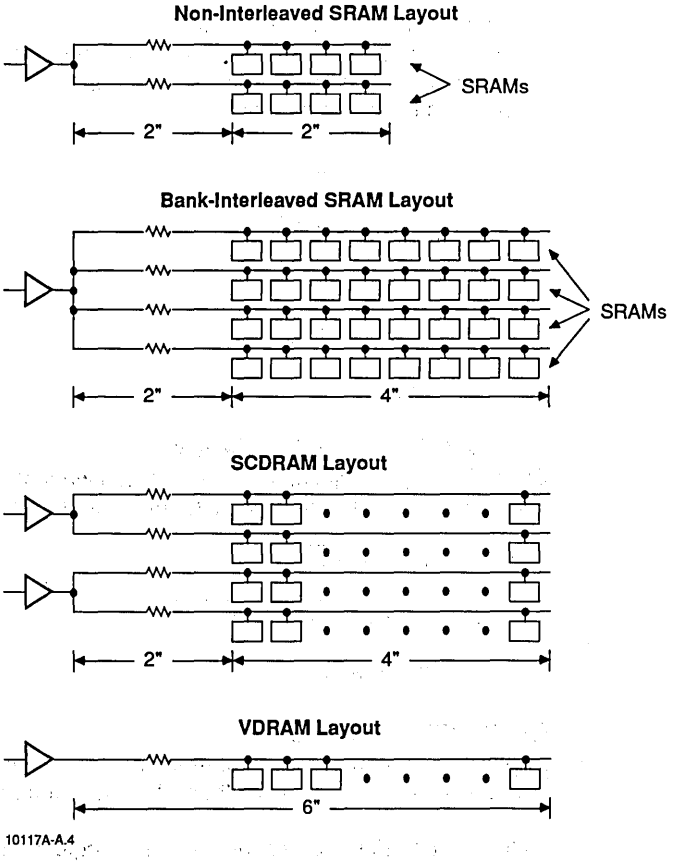
Depending on how the array of memory chips is laid out, it is possible to force the memory system to look like either a transmission line or a lumped RLC circuit.

If all the memories are attached along a single set of serially routed signal traces then each trace will act as a transmission line. Assuming a typical memory array of 32

devices the traces would need to be 1.33 feet long. Using the calculations shown in the last section, two times the line propagation delay would be 14.6 ns. This value surpasses the 2 to 5 ns rise or fall time of a typical high-speed buffer. So this layout should be treated as a transmission line.

If all the memories are very closely grouped to the driver by splitting the signal traces into a tree-like structure with very few memories on each branch. The root-to-branch-end length can be made very short. Assuming the same memory array of 32 devices split into 8 branches of 4 devices each, the branch length could be limited to about 4 inches. This assumes 2 inches of each branch contains memory devices and there is about 2 inches of routing required between the driver output and the first memory on any one branch. In this configuration the propagation delay to the end of a branch is 1.87 ns. Two times the the delay is 3.75 ns which is within the range of normal rise and fall times for a signal driver. This means that the memory array will behave more like a lumped RLC circuit than like a transmission line.

Figure A-4



Memory Layout Models

LAYOUT MODELS

Chapters 4 through 7 of this handbook show four different memory systems. The medium-speed bank interleaved SRAM design and the SCDRAM design each use 32 memory devices per bank of memory. The VDRAM design and high-speed non-interleaved SRAM design use only eight memories per bank. Memory layout models of the SRAM, SCDRAM and VDRAM designs are shown in Figure A-4.

The non-interleaved SRAM design uses as few memory devices as possible and places the memory devices as close to the processor as possible. The eight memories are placed into two rows of four devices each. This gives a two-branch tree structure to the pcb trace layout. Each branch is assumed to be 4 inches in length with memories placed two per inch along 2 inches of the trace and the remaining 2 inches of trace used for routing to the processor.

For the bank-interleaved-SRAM design, the layout places the 32 memories into 4 rows of 8 devices each. This creates a tree structure with each branch being 4 inches long, assuming that memories are placed two per inch along the trace. To allow for trace routing from the driver to each branch, 2 inches will be added to each branch. Therefore, the "driver to end of branch length" will be 6 inches.

The SCDRAM design is a subset of the above in that dual RAS and CAS drivers are provided so that the set of 32 memories may be broken into two separate tree structures, each with two branches. This maintains the driver-to-end-of-branch length at 6 inches; however, it lowers the total capacitive and inductive load on each driver.

The VDRAM model is a subset of the above. The eight memories will be placed on a single trace 6 inches long.

TRANSMISSION LINES OR RLC CIRCUITS?

From the discussion of memory loading factors, it can be seen, that a representative value of propagation delay for a memory trace is about 5 ns per foot. With trace lengths of 6 inches, the two propagation delays time of a trace will remain at 5 ns. That value very closely approximates the rise and fall times of common signal drivers, which for D-speed PALs can range from 2 to 5 ns.

So, opinion is divided on whether the RLC circuit or the transmission line model is more accurate in the above situation. Therefore the memory designs are analyzed with both models and the most conservative delay values that result are used in the design timing estimates.

TRANSMISSION LINE MODEL—THE BASICS

In the ideal transmission-line model, the line is infinitely long with a constant characteristic impedance. A signal sent down such a line, will travel along the line without distortion. The propagation rate is determined by the dielectric constant surrounding the signal line, and by the capacitive loading of the line. A less than infinitely long line can be made to appear so, if the end of the line is terminated by a resistance equal to the characteristic impedance.

When this ideal is not met, due to variations in impedance or a mismatch in the terminating (load) impedance of the line, there are resulting voltage and current reflections that travel back along the line. The magnitude of the reflection is directly related to the difference between the load impedance and characteristic line impedance. This relationship is given by:

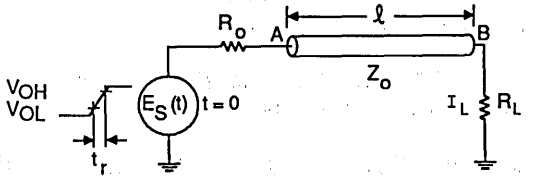
$$P_L = \frac{R_L - Z_o}{R_L + Z_o}$$

Similarly, when those reflections reach the source end of the line they will in turn be reflected back toward the load end of the line if the source impedance does not match the line impedance. The reflection coefficient at the source is:

$$P_s = \frac{R_s - Z_o}{R_s + Z_o}$$

To determine the voltage at a given point on the transmission line, at a given time, the model of Figure A-5 is used.

Figure A-5



$$V(X,t) = V_A(t)[U(t - t_{pd}X) + P_L U(t - t_{pd}(2\ell - X) + P_L P_S U(t - t_{pd}(2\ell + X)) + P_L^2 P_S U(t - t_{pd}(4\ell - X)) + P_L^2 P_S^2 (t - t_{pd}(4\ell + X)) + \ldots] + V_{dc}$$

Where: $V_A(t) = E_S(t)\left(\frac{Z_o}{Z_o + R_o}\right)$

V_A = voltage at point A,
X = the distance to an arbitrary point on the line
ℓ = total line length,
t_{pd} = propagation delay of the line in ns/unit distance,
$T_D = \ell t_{pd}$,
$U(t)$ = a unit step function occurring at t = 0, and
$E_S(t)$ = internal voltage swing in the circuit ($V_{OH} - V_{OL}$)

$$P_L = \frac{R_L - Z_o}{R_L + Z_o}$$

$$P_S = \frac{R_o - Z_o}{R_o + Z_o}$$

10117A-A.5

Transmission Line Models

Memory Specific Example

Determining Transmission Line Impedance and Propagation Delay.
In each of the layout models described for the memory system, the branch length remains nearly the same. There are small variations in capacitive loading depending on the specific memories used, but in general, each model looks very similar.

Each transmission line has a two-inch section with no capacitive memory load followed by 2 to 4 inches of trace with two memories per inch. This structure complicates the model a little since it looks like a 95 Ω transmission line connected to a 30 Ω impedance line. This results in different propagation times along the trace and signal reflections at the points of impedance change.

To simplify the model for the remaining discussion the memory capacitance is viewed as distributed across the entire length of the line, e.g., Cd = 24 devices/ft x 7 pF/device x (4 in. memory loaded length/6 in. total length) = 112 pF/ft. This more closely approximates the overall delay of the line and simplifies the analysis to deal only with reflections at the source and load ends of the transmission line.

So, for 7 pF per memory input loading, the transmission line impedance and propagation delay would be:

$$Zo' = \frac{Zo}{\sqrt{(1 + Cd/Co)}}$$

$$= \frac{95.93}{\sqrt{(1 + 112/18.5)}}$$

$$= 36.12\ \Omega$$

$$t_{pd}' = t_{pd}\sqrt{(1 + Cd/Co)}$$

$$= 1.774\sqrt{(1+ 112/18.5)}$$

$$= 4.71\ ns/ft$$

A table for various input capacitance levels is shown in Table A-1, that reflects the effect on respective impedances and delays using the calculations methods just outlined:

Table A-1

Input Capacitance Levels

pF/Input	Cd pF/ft	Zo' Ω	t_{pd}' ns/ft
5	80	41.5	4.09
6	96	38.5	4.41
7	112	36.12	4.71
8	128	34.08	4.99
9	144	32.36	5.25
10	160	30.88	5.51

Load Impedance

For this analysis the load impedance will be assumed to be infinite, resulting from no termination resistance being placed at the end of the line.

Source Impedance

The source impedance is that of a D-speed PAL output. The output impedance for this type of device (and for most TTL outputs) is different for the output-low condition verses the output-high condition.

For the output-low condition a worst-case impedance estimate can be made by dividing V_{OL} by I_{OL}. For a D-speed AmPAL16L8, that would be 0.5 V/.024 A = 20.8 Ω. This is truly the worst possible case with static output conditions. The output driver is able to hold that voltage level forever as long as the output current does not exceed the 24 mA limit. That, however, is not representative of the actual output impedance apparent during the few nanoseconds that it takes to switch the output from high to low. Based on the experience of PAL circuit designers, a more realistic estimate is about 5 Ω.

For the output-high condition, a worst-case impedance is more difficult to define. Its output impedance varies as the output voltage rises. When the driver begins to pull the output up, the output current provided by the driver is much more than is available when the output is held at V_{OH}. Determined empirically, the typical value for the high-level output impedance during low-to-high switching is about 25 Ω.

Source Voltage Swing

The data-sheet-guaranteed worst-case output high and low voltages for a TTL driver are: V_{OH} = 2.4 V and V_{OL} = 0.5 V. But, these are rarely seen in actual circuits. More realistic output levels typical of a D-speed PAL are: V_{OH} = 4 V and V_{OL} = 0.2 V. This gives a voltage swing of 3.8 V.

Output rise time is measured from V_{OL} = 0.2 V to the TTL standard V_{IL} = 2 V. The fall time is measured from the V_{OH} = 4 V to the TTL standard V_{IL} = 0.8 V.

High-to-Low Transition Analysis

In general the high-to-low transition of the signal driver is the more interesting event to analyze. This is because the undershoot that results from the unterminated transmission line is a critical parameter for many memories. *Too much undershoot and the memories can be damaged.*

Also, reflections (of the undershoot) at the source end of the line can result in positive transitions above V_{IL} (input-low voltage threshold). Any transitions above V_{IL} delay the settling time to a valid input-low level.

The analysis begins by filling in the variables of Figure A-5.

1. Es(t) is set equal to the voltage swing of the source, –3.8 V.
2. Zo is the load impedance of the line assuming 7 pF/input memories, 36.12 Ω.
3. Ro is the source impedance for the output-low condition, 5 Ω.
4. VA(t) is the voltage swing resulting at point A (source end) on the transmission line, calculated to be –3.338 V.

5. U(t) is the unit impulse function which is equal to zero for values of t less than zero, and equal to one for t greater than or equal to zero. This function is used because, according to theory, the rise or fall time of the driving voltage source is not affected by the capacitance of the transmission line. Therefore, the U(t) function serves to switch on VA(t) or the reflected values of VA(t) at the appropriate times.

6. P_L is the coefficient of reflection at the load and is calculated to be nearly equal to one.

7. P_S is the coefficient of reflection at the source and is calculated to be –0.7568.

8. ℓ is the total line length of 0.5 ft.

9. t_{pd} is the propagation delay of 4.71 ns.

10. T_D is the propagation delay time down one length of the line; tpd times ℓ.

11. The points of interest on the transmission line for this analysis will be at the source and load ends of the line at times that are integer multiples of t_{pd}. Therefore X will be equal to 0, t_{pd}, 2 t_{pd}, ... which would be (X times 4.71 ns times 0.5 ft) 0, 2.355 ns, 4.71 ns, 7.065 ns...etc.

12. Vdc is the steady state voltage of the transmission line before the signal voltage transition at t = 0, 4 V.

The values shown in Table A-2 were calculated using the equations of Figure A-5.

Table A-2

Values Calculated From Equations Provided in Figure A-5.

t T_D	VA Volts	VB Volts
0	0.662	4.0
1	0.662	–2.676
2	–0.150	–2.676
3	–0.150	2.376
4	0.465	2.376
5	0.465	–1.447
6	0.00	–1.447
7	0.00	1.447
8	0.352	1.447
9	0.352	–0.743
10	0.085	–0.743
11	0.085	0.914
12	0.287	0.914

Even after 12 transitions of the line (28 ns), the signal level has not settled to below the valid input-low level as a result of the reflections at the source and load impedance mismatches.

Note, a listing of the BASIC language program used to calculate the above table (sometimes referred to as a lattice diagram) is shown in Figure A-10.

Overshoot and Undershoot

Also, from the above table, it can be seen that undershoot in excess of –2.5 V is present on the line. That degree of undershoot can be damaging to DRAMs. Some SRAMs are designed to handle up to –3 V undershoot, but even if the memory can handle the voltage stress, the settling time delay to a valid low level is still excessive.

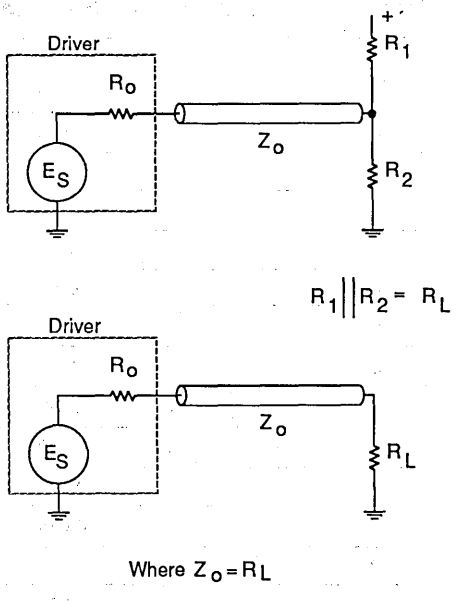
Overshoot values can also be calculated for the low-to-high transition situation. The overshoot will reach values near 4.7 V which is not a threat to any standard memory device.

Termination

From the above discussion, it is clear that something must be done to reduce the degree of reflections at load or source end of the transmission line. This can be done by adding a resistance load to either end of the line. The load can be a resistor-to-ground or a voltage divider between power and ground in which case the load value is the Thevenin equivalent. This method, shown in Figure A-6, is called parallel termination.

When done at the load end of the line, this is the best way to terminate the line in terms of signal settling time. Proper parallel termination gets rid of reflection entirely at the load end of the line. Therefore only one propagation delay time down the line is required before the entire line settles to the desired voltage level.

Figure A-6



Parallel Termination

But there is a problem with this method. Parallel termination to power or ground at the near 30 Ω characteristic impedance of the loaded transmission line would overwhelm the dc-drive capability of a D-speed PAL output used to drive the line. This is especially true when considering the dc load of parallel termination on multiple transmission lines tied to one driver.

So, unless a high-current driver is used with the memory array, parallel termination is not appropriate. If parallel termination is used, the added propagation time of the high-current driver must be traded off with the shorter settling time of the signal.

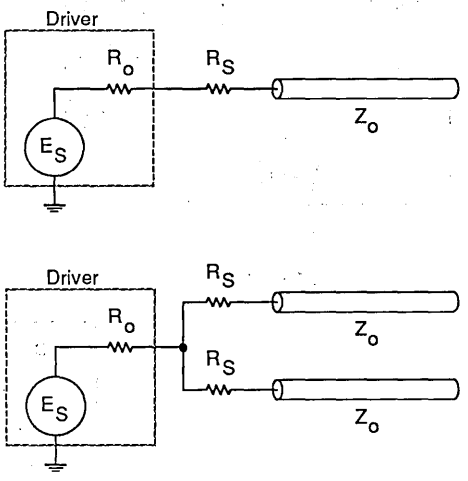
Another more common termination method is called serial damping. With this method, a resistor is placed in series with the driver and transmission line. The value of the resistance is chosen to be equal to the line impedance when added to the driver impedance. In this way, when looking at the source end of the transmission line, the combination of the driver impedance and series resistance matches the line impedance.

With a matched impedance at the source end of the line, there can only be reflections at the load end of the line. Thus, when reflections from the load end of the transmission line return to the source end of the line, the entire line will have settled to the desired voltage level.

So, with series damping the settling time is equal to two times the propagation delay of the line. Also, there is no dc load imposed by the termination resistance so a standard signal driver can be used.

As shown in Figure A-7, where multiple transmission lines are tied to a single driver, each transmission line should have its own serial-damping resistor to match the impedance to each line. Very often, memory system designers will use a single resistor

Figure A-7

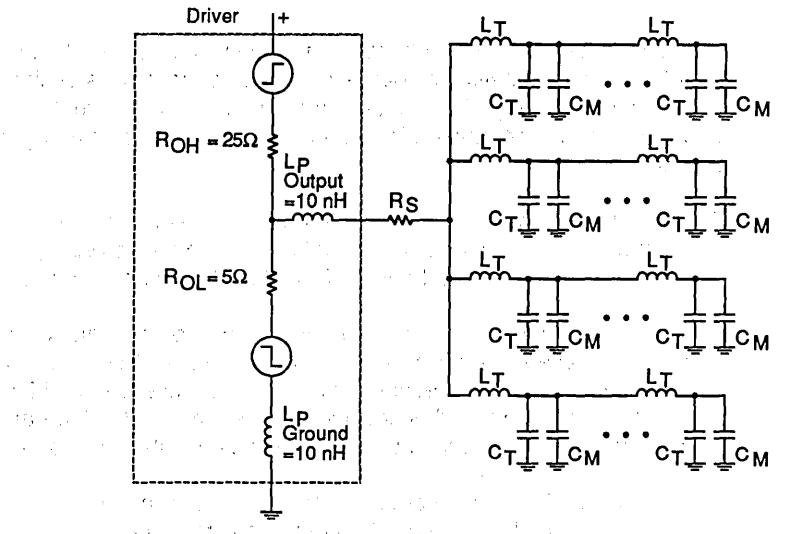


10117A-A.7

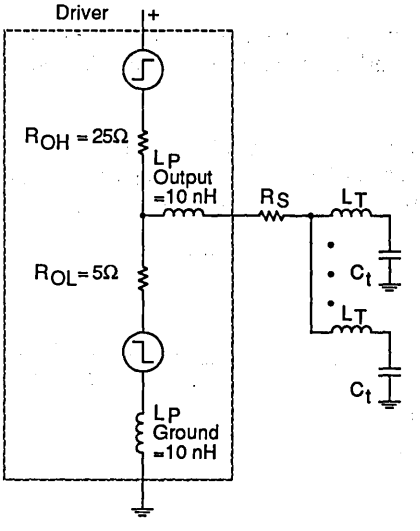
Where $R_o + R_S = Z_o$

Series Damping

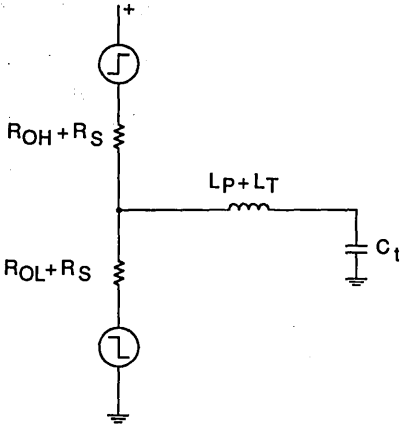
Figure A-8



Step a.



Step b.



Step c.

$C_t = C_{total}$
$C_T = C_{Trace}$
$C_M = C_{Memory}$
$L_T = L_{Trace}$
$L_P = L_{Package}$
$R_S = R_{Series}$

10117A-A.8

RLC Model Simplication Steps

between the driver and all the transmission lines as a compromise that reduces component count at the cost of a higher, but acceptable, degree of signal reflections.

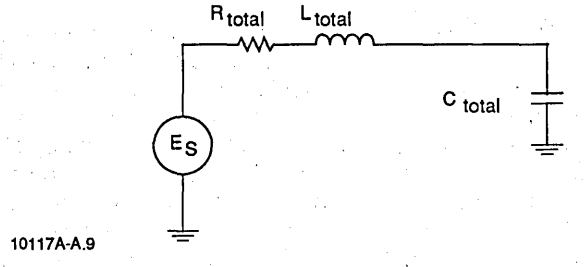
Therefore, in all of the memory designs presented in this handbook, serial damping-resistors are used in all memory address and control lines. The resistor value used is in the range of 20 to 30 Ω. The exact value should be determined empirically to minimize reflections.

RLC MODEL

The RLC model lumps all the capacitive and inductive loads into single elements arranged as shown in Figure A-8. The distributed capacitive loads of the memories on each branch of the memory layout can be totaled, then the capacitance on each branch is considered to be in parallel and is thus totaled into the value for a single equivalent component.

Similarly, the inductive loads in each branch are totaled since those elements lie in series. Then the inductance for each branch is viewed as being in parallel with the inductors of the other branches and thus their value is divided by the number of branches to determine the value for a single equivalent component. To that component is added the inductance of the driver package pins and internal bond wires. The output switching voltage generators, output impedances, and any damping resistance is then added. Since the output voltage swing is the same for either a high-to-low or a low-to-high transition, the model can be simplified one additional step to that shown in Figure A-9. In this model, the equations for either switching transition are the same; only the polarity of the voltage and the value of the output resistance is changed.

Figure A-9



Final RLC Model

This model is then analyzed with LaPlace transforms to yield an equation for current flow overtime:

$$I_t = \frac{A}{L\,B}\, e^{-at} \sin B\,t$$

Where:

$$B = \sqrt{\left(\frac{1}{LC} - \frac{R^2}{4\,L^2}\right)}$$

$$a = \frac{R}{2L}$$

A = voltage switching step function magnitude

$$\frac{1}{LC} > \frac{R^2}{4\,L^2}$$

The output voltage is then:

$$Vout = \frac{1}{C}\int_0^t I_t\, d_t$$

$$= A\left(1 - \left(e^{-at}\left(\frac{a}{B}\sin B\,t + \cos B\,t\right)\right)\right)$$

It should be noted that this model will predict overshoot, undershoot, and delay values somewhat in excess of that expected for a real implementation. This is mainly due to the use of a step function to model the initial voltage transition rather than the use of a ramp function which would better model the rise or fall time to be found in a real system example. This model also does not deal with the amount of delay related to a standard test load which is already accounted for by worst-case delay values of the driver as shown in its data sheet. To obtain a more accurate estimate of the RLC circuit's added delay, the difference between the driver's data sheet worst-case delay and the driver's intrinsic (no output load) delay should be subtracted from the RLC circuit delay estimate. The driver's intrinsic delay can be determined by experimentation or through consultation with the device manufacturer.

Memory Specific Example

Determining Element Values

The initial transition voltage is set by V_{OH}–V_{OL}, which as noted before is about 3.8 V for a D-speed PAL output. The voltage step is positive on low-to-high transitions and negative for high-to-low transitions. The source impedances are the same as used earlier. High-to-low transition is 5 Ω and low-to-high transition is 25 Ω. Damping will initially be set to zero to see what sort of overshoot and undershoot occurs in an undamped circuit.

Driver output inductance is assumed to be 20 nH. The trace inductance is derived from the pcb characteristics defined earlier. The value found was 170 nH per foot of trace length. Since each branch of the memory layout is 6 inches long, the value per branch is 85 nH. With four branches viewed in parallel, the effective inductance is 21.25 nH.

Assuming each memory input has 7 pF of capacitance, the 32 memories in the layout total 224 pF.

The trace capacitance is derived from the pcb characteristics defined earlier. The value found was 18.5 pF per foot of trace length. The total branch length in this design is two feet, therefore total trace capacitance is 37 pF.

The Results

A simple program written in the BASIC language was used to calculate the RLC model behavior based on the above equations and input parameters. A listing of this program is shown in table A-11 (located at the end of the chapter). The result was to predict that, with no damping resistance, the undershoot would reach a maximum of –2.1 V with a subsequent rebound to +1.5 V. In fact, a high-to-low transition would not settle below 0.8 V until after 22 ns. The low-to-high transition settled above 2.4 V within 6 ns.

This result obviously is unacceptable both in the level of undershoot, which could damage memories and in the excessive settling time. The circuit was modified to include a 5 Ω damping resistor. The high-to-low transition undershoot was then limited to –0.8 V and the settling time to a level below 0.8 V was reduced to 6 ns. The low-to-high transition time remained at nearly 6 ns.

DESIGN EXAMPLE DELAY VALUES

The memory system loading delay values used in each of the memory design example chapters are derived below.

Non-Interleaved SRAM Example

As noted in Chapter 4, the total of all the other delay elements in this SRAM design example already equal 38.3 ns , leaving little room for an overly conservative estimate of the added delay associated with driving the memory array. So, lets look at refining the above estimates.

The transmission line delay of 2.4 ns is essentially equal to the typical rise or fall time of a PAL output driver. Thus, the driver "sees" most of the load during the output transition time. That load of 52 pF and 48 nH (including driver package inductance) is nearly equal to the test load used to determine the worst case output delay time quoted for the driver. Therefore, a transmission-line model does not appear to be valid for this design situation.

The RLC model predicts the delay for driving the entire load and thus that delay should be added to the propagation delay measured for a driver with zero load (intrinsic driver propagation delay). But, the data-sheet values for driver delay only indicate the delay when driving a 50 pF capacitive load combined with driver package inductance and some small inductance from the test load circuit layout. This is essentially equal to the load presented by this SRAM design. Therefore, it is fairly reasonable to assume that the worst case delays quoted for the driver already include the time required to drive the

load presented by this memory design. But, for the sake of being a little conservative, the difference between D-speed PAL driver intrinsic delay and delay with test load was determined experimentally. The intrinsic delay is about 1.3 ns less than the delay with the test load. Adjusting the estimated RLC delay to account for delay already included in the quoted worst-case delay (2.8 ns–1.3 ns) leaves 1.5 ns of excess delay predicted by the RLC model. This value will be used as the estimated RLC delay.

The remaining designs, to be honest, allow more room to be conservative and thus will use the raw delay values from the transmission line and RLC models.

Bank-Interleaved SRAM Example

This memory design uses four branches, each 6 inches long. The SRAM memory device used has an input capacitance of 5 pF for all inputs.

The transmission-line model predicts a delay of 4 ns that must be added to the output delay of the memory driver. A 20 to 30 Ω damping resistor is used on each branch.

The RLC model predicts a delay of 5 ns. The undershoot in this case is –1.2 V which is allowable for the SRAM memories that are able to handle –3 V. The assumptions for this model are:

- an inductive load of 42 nH,
- a capacitive load of 200 pF,
- a 5 Ω damping resistor.

SCDRAM Example

The SCDRAM devices used have 5 pF capacitive input on address lines but 7 pF on each control line such as $\overline{RAS}$, $\overline{CAS}$, $\overline{WE}$. So address lines are modeled separately from the control lines.

The address lines are assumed to be laid out like the SRAM examples with four branches containing 32 memories. The transmission line model predicts the same 4 ns delay as seen in the SRAM example. However, the RLC model for the SCDRAM is different. In order to limit the undershoot to less than –1 V as required by the SCDRAM, the RLC model damping resistor value is set at 8 to 10 Ω. This produces an undershoot of –0.8 V and a delay of 6 ns.

For the control lines a different layout model is used. Two separate dual-branch traces are used to drive the memories so that only 16 devices will load each memory driver. This was done early in the design in the hopes that it would improve the signal speed with the very small cost of four additional PAL outputs being required. As it turns out, neither delay model predicts a very significant improvement. The transmission line model predicts a 4.7 ns delay. The RLC model predicts a 6.5 ns delay. Assuming an inductive load of 62 nH, a capacitive load of 150 pF, a 15 Ω damping resistor, and –0.8 V undershoot.

VDRAM Example

The VDRAM design needs only eight memory devices per bank since the memories are each four bits wide. These are placed on a single 6-inch trace. The input capacitance ranges from 5 pF to 10 pF depending on input and manufacturer. The worst case value of 10 pF is assumed. The transmission line model predicts 5.5 ns delay. The RLC model predicts 6.5 ns delay, assuming an inductive load of 105 nH, a capacitive load of 120 pF, a 22 Ω damping resistor, and –0.7 V undershoot.

Damping Resistors

Note that for each of the damping resistor values shown in the RLC models, the value of the common damping resistor is essentially the Thevenin equivalent of having one resistor for each branch between the driver and the branch, where the value the resistor is in the 20 to 30 Ω range. This fits nicely with the transmission line model that requires a serial damping resistor on each branch. So, for the sake of having a common layout plan, it assumed that all the memory designs implement the needed damping resistance by placing resistors on each signal branch.

Summary

Table A-3 summarizes the results of the delay model analysis on each design example. For the sake of being conservative, the longest delay value is used in each case. In each case this turns out to be the value predicted by the RLC model.

Table A-3

Summary of Delay Model Analysis Results

Example	Capacitance of Input pF	Transmission Line Delay ns	RLC Model Delay ns
Non Interleaved SRAM	5	N/A	1.5
Bank Interleaved SRAM	5	4	5
SCDRAM	5	4	6
	7	4.7	6.5
VDRAM	10	5.5	6.5

REFERENCES AND ACKNOWLEDGEMENTS

1. William R. Blood, Jr. "ECL Systems Design Handbook." Motorola Semiconductor Products Inc., Mesa, AZ, May 1983 (fourth edition); chapters 3 and 7.

2. Special thanks to David Stoenner, Advanced Micro Devices Field Applications Engineer, Newport Beach, CA., for his help in preparing this appendix.

Figure A-10

```
10 REM****************** Transmission Line Analyzer ******************
20 REM ***************************************************************
30 REM
40 REM ********************** input initial values ********************
50 VOH = 4
60 VOL = 0.2
70 RL = 5
80 RH = 25
90 RD = 22
100 ER = 5
120 CL = 7E-12
130 T = 0.003
140 H = 0.03
150 W = 0.01
160 SP = 0.75
170 RLOAD = 1E+09
180 L = 6
900 REM ****************** parameter display ************************
1000 CLS
1001 PRINT "Memory System Transmission Line Analyzer"
1010 PRINT
1020 PRINT "Type the number of the value you wish to change:"
1030 PRINT
1040 PRINT
1050 PRINT " 0) no changes"
1060 PRINT " 1)Voh",VOH;"V"
1070 PRINT " 2)Vol",VOL;"V"
1080 PRINT " 3)Rh",RH;"ohms",,"totem pole resistance to VCC"
1090 PRINT " 4)Rl",RL;"ohms",,"totem pole resistance to GND"
1100 PRINT " 5)Rd",RD;"ohms",,"series damping resistance"
1120 PRINT " 6)Er",ER,,"relative dielectric of pcb"
1130 PRINT " 7)w",W;"inches",,"width of pcb trace"
1140 PRINT " 8)h",H;"inches",,"height of pcb trace above ground"
1150 PRINT " 9)t",T;"inches",,"thickness of pcb trace"
1160 PRINT " 10)l",L;"inches",,"length of pcb trace"
1170 PRINT " 11)Cl",CL;"F",,"capacitance of memory input"
1175 PRINT " 12)Sp",SP;"inches",,"spacing between memories"
1176 PRINT " 13)Rl",RLOAD;"ohms",,"end of line load resistance"
1180 PRINT
1183 PRINT"change number ";
1185 INPUT VARIABLE
1190 IF VARIABLE >=0 AND VARIABLE <= 13 THEN GOTO 1220
1200 PRINT " invalid parameter number ... please reenter choice"
1210 GOTO 1000
1215 REM ****************** parameter modification*********************
1220 ON VARIABLE GOSUB
2100,2200,2300,2400,2500,2600,2700,2800,2900,3000,3100,3200,3300
1230 IF VARIABLE = 0 THEN GOSUB 10000
2000 GOTO 1000
2100 PRINT" Voh = (volts) ";
2110 INPUT VOH
2120 RETURN
2200 PRINT "Vol = (volts) ";
2210 INPUT VOL
2220 RETURN
```

Transmission Line Program Listing for MS-DOS

Figure A-10

```
2300 PRINT "Roh = (ohms) ";
2310 INPUT RH
2320 RETURN
2400 PRINT "Rol = (ohms) ";
2410 INPUT RL
2420 RETURN
2500 PRINT "Rd = (ohms) ";
2510 INPUT RD
2520 RETURN
2600 PRINT "Er = ";
2610 INPUT ER
2620 RETURN
2700 PRINT "w = (inches) ";
2710 INPUT W
2720 RETURN
2800 PRINT "h = (inches) ";
2810 INPUT H
2820 RETURN
2900 PRINT "t = (inches) ";
2910 INPUT T
2920 RETURN
3000 PRINT "l = (inches) ";
3010 INPUT L
3020 RETURN
3100 PRINT "Cl = (Farads) ";
3110 INPUT CL
3120 RETURN
3200 PRINT "Sp = (inches) ";
3210 INPUT SP
3220 RETURN
3300 PRINT "Rl = (ohms) ";
3310 INPUT RLOAD
3320 RETURN
10000 REM ********** calculate transmission line characteristics*******
10100 Z0 = (87/SQR(ER + 1.41))*LOG((5.98*H)/(.8*W+T))
10110 TPD0 = 1.017*SQR(.475*ER + 0.67)
10120 C0 = 1000*(TPD0/Z0)
10130 CD = (CL/SP)*12*1E+12
10140 Z1= Z0/SQR(1+(CD/C0))
10150 TPD1 = TPD0 * SQR(1+(CD/C0))
10160 ES = VOH-VOL
10190 PL = (RLOAD-Z1)/(RLOAD+Z1)
10200 RSOURCEHL = RD + RL
10210 PSHL = (RSOURCEHL - Z1)/(RSOURCEHL + Z1)
10220 RSOURCELH = RD + RH
10230 PSLH = (RSOURCELH - Z1)/(RSOURCELH + Z1)
10240 VDCHL = VOH
10250 VDCLH = VOL
10270 VAHL = -1*ES*(Z1/(Z1+RSOURCEHL))
10280 VALH = ES*(Z1/(Z1+RSOURCELH))
10290 REM********************* display line characteristics ************
10300 CLS
10310 PRINT "Transmission Line Analysis"
10320 PRINT
10330 PRINT
```

Transmission Line Program Listing for MS-DOS (Cont'd.)

Figure A-10

```
10340 PRINT"Driver voltage step =",,ES,"Volts"
10350 PRINT"Driver source impedance, high to low",RL,"ohms"
10360 PRINT"Driver source impedance, low to high",RH,"ohms"
10370 PRINT"Damping resistance",,RD,"ohms"
10380 PRINT"Line impedance",,Z1,"ohms"
10390 PRINT"Line capacitance",,CD*(L/12)+C0*(L/12),"picoFarads"
10400 PRINT"Line inductance",,(Z0^2*C0*.001)*(L/12),"nanoHenrys"
10410 PRINT"Line length",,,L,"inches"
10415 PRINT"Line propagation rate",,TPD1,"ns/ft"
10420 PRINT"Line propagation delay",,(L/12)*TPD1,"ns"
10430 PRINT"Load impedance",,RLOAD,"ohms"
10440 PRINT
10450 PRINT
10460 PRINT
10470 PRINT"hit return when ready to proceed...  ";
11000 REM ************* lattice diagram calculations ******************
11010 CLS
11020 PRINT "Lattice Diagrams for High to Low and Low to High
        Transitions"
11030 PRINT
11040 PRINT TAB(18);"High to Low:";TAB(45);"Low to High:"
11050 PRINT TAB(18);"------------";TAB(45);"------------"
11060 PRINT "TD";TAB(6);"Time";TAB(18);"Vs";TAB(30);"Vl";TAB(45);"Vs";
TAB(57);"Vl"
11070 PRINT
11072 F1$ ="####  ###.###    ###.###                ###.###"
11073 F2$ ="####  ###.###                ###.###                ###.###"
11074 UHL = 0
11075 ULH = 0
11076 I=0
11080 FOR TD = (0 + I) TO (15 + I)
11085 UHL = PL^(INT(TD/2 +.5)) * PSHL^(INT(TD/2)) + UHL
11087 ULH = PL^(INT(TD/2 +.5)) * PSLH^(INT(TD/2)) + ULH
11090 VTHL = (VAHL * UHL)+ VDCHL
11100 VTLH = (VALH * ULH)+ VDCLH
11110 IF ( (TD/2 - INT(TD/2)) > 0 ) THEN GOTO 11150
11120 PRINT USING F1$;TD;TD*TPD1*(L/12);VTHL;VTLH
11130 GOTO 11190
11150 ' else
11160 PRINT USING F2$;TD;TD*TPD1*(L/12);VTHL;VTLH
11190 NEXT TD
11195 I= I +16
11200 PRINT
11210 PRINT "more (y/n) ";
11220 INPUT YESNO$
11230 IF YESNO$ <> "n" THEN GOTO 11080
11240 PRINT "do you want to run the program again "
11250 INPUT YESNO$
11260 IF YESNO$ <> "n" THEN RETURN
12000 END
```

Transmission Line Program Listing for MS-DOS (Cont'd.)

Figure A-11

```
10 REM ****************** Over & Undershoot Analyzer *****************
20 REM ****************** for RLC networks *****************
30 REM ************************************************************
40 REM ******************** input initial values *********************
50 VOH = 4
60 VOL = 0.2
70 RL = 5
80 RH = 25
90 RD = 22
100 LP = 2E-08
110 LT = 1.08E-07
120 CL = 2.5E-10
130 CT = 2E-11
900 REM ********************** display parameters **********************
1000 CLS
1001 PRINT "Over & Undershoot Analyzer"
1010 PRINT
1020 PRINT "Type the number of the value you wish to change:"
1030 PRINT
1040 PRINT
1050 PRINT " 0) no changes"
1060 PRINT " 1)Voh",VOH;"V"
1070 PRINT " 2)Vol",VOL;"V"
1080 PRINT " 3)Rh",RH;"ohms",,"totem pole resistance to VCC"
1090 PRINT " 4)Rl",RL;"ohms",,"totem pole resistance to GND"
1100 PRINT " 5)Rd",RD;"ohms",,"series damping resistance"
1110 PRINT " 6)Lp",LP;"H",,"inductance of driver package"
1120 PRINT " 7)Lt",LT;"H",,"inductance of PC trace"
1130 PRINT " 8)Cl",CL;"F",,"capacitance of load"
1140 PRINT " 9)Ct",CT;"F",,"capacitance of PC trace"
1150 PRINT
1160 PRINT
1170 PRINT "change number ";
1180 INPUT VARIABLE
1190 IF VARIABLE >=0 AND VARIABLE <= 9 THEN GOTO 1220
1200 PRINT " invalid parameter number ...  please reenter choice"
1210 GOTO 1000
1215 REM ****************** parameter modification ******************
1220 ON VARIABLE GOSUB 2100,2200,2300,2400,2500,2600,2700,2800,2900
1230 IF VARIABLE = 0 THEN GOSUB 10000
2000 GOTO 1000
2100 PRINT" Voh = (volts) ";
2110 INPUT VOH
2120 RETURN
2200 PRINT "Vol = (volts) ";
2210 INPUT VOL
2220 RETURN
2300 PRINT "Rh = (ohms) ";
2310 INPUT RH
2320 RETURN
2400 PRINT "Rl = (ohms) ";
2410 INPUT RL
2420 RETURN
2500 PRINT "Rd = (ohms) ";
2510 INPUT RD
```

RLC Circuit Program Listing for MS-DOS

Figure A-11

```
2520 RETURN
2600 PRINT "Lp = (henrys) ";
2610 INPUT LP
2620 RETURN
2700 PRINT "Lt = (henrys) ";
2710 INPUT LT
2720 RETURN
2800 PRINT "Cl = (Farads) ";
2810 INPUT CL
2820 RETURN
2900 PRINT "Ct = (Farads) ";
2910 INPUT CT
2920 RETURN
9000 REM ****************** calculate RLC characteristics ************
10000 VHL = -(VOH-VOL)
10100 VLH = VOH-VOL
10200 RHL = RL + RD
10300 RLH = RH + RD
10400 L = LP + LT
10500 C = CL + CT
10600 LCINV = 1/(L*C)
10700 R24L2HL = (RHL^2)/(4*(L^2))
10710 R24L2LH = (RLH^2)/(4*(L^2))
10800 ALPHAHL = RHL/(2*L)
10810 ALPHALH = RLH/(2*L)
10900 BETAHL = SQR(ABS(LCINV - R24L2HL))
10910 BETALH = SQR(ABS(LCINV - R24L2LH))
11900 REM ****************** display RLC characteristics **************
12000 CLS
12100 PRINT "high to low transition";TAB(40);"low to high transition"
12200 PRINT
12300 PRINT "Vhl = ";VHL;TAB(40);"Vlh = ";VLH
12400 PRINT "Rhl = ";RHL;TAB(40);"Rlh = ";RLH
12500 PRINT "R^2/4L^2 = ";R24L2HL;TAB(40);"R^2/4L^2 = ";R24L2LH
12600 PRINT "R/2L = ";ALPHAHL;TAB(40);"R/2L = ";ALPHALH
12700 PRINT "Beta =";BETAHL;TAB(40);"Beta = ";BETALH
13000 PRINT
13100 PRINT
13105 IF LCINV > R24L2HL THEN GOTO 13119
13110 PRINT "Opps its hyperbolic"
13115 PRINT " R > ";SQR(LCINV * (4*(L^2)))
13118 PRINT "falling edge waveform is invalid"
13119 IF LCINV > R24L2LH THEN GOTO 13201
13120 PRINT TAB(40);"Opps its hyperbolic";
13125 PRINT TAB(40);" R > ";SQR(LCINV * (4*(L^2)))
13150 PRINT TAB(40);"rising edge waveform is invalid"
13201 PRINT "L = ";L
13202 PRINT "C = ";C
13203 PRINT "1/LC = ";LCINV
13300 PRINT
13600 PRINT "display the output waveform; rising/falling/none (r/f/n) ";
13610 INPUT RFN$
13620 IF RFN$ = "r" THEN GOSUB 30000
13630 IF RFN$ = "f" THEN GOSUB 20000
13640 IF RFN$ = "n" THEN GOTO 13800
```

RLC Circuit Program Listing for MS-DOS (Cont'd.)

Figure A-11

```
13650 GOTO 13600
13800 PRINT "do you want to run the program again (y/n) ";
13900 INPUT YESNO$
14000 IF YESNO$ <> "n" THEN RETURN
15000 END
20000 REM**************** high to low waveform **************
20010 I = 0
20100 CLS
20200 PRINT "Tns | ---- Vout (volts) ++++ "
20300 PRINT "     3....+....2....+....1....+....0....+....1....+....
      2....+....3....+....4"
20400 FOR T = (1+ I) TO (20 + I)
20500 VOUT = VHL*(1-(EXP(-(ALPHAHL*T*1E-09))*(((ALPHAHL/BETAHL)*
SIN(BETAHL*T*1E-09))+COS(BETAHL*T*1E-09))))+VOH
20600 VSCALE = INT((ABS(3+VOUT)*10)+.5)
20700 IF VSCALE > 70 THEN VSCALE = 70
20800 PRINT T;TAB(6);"|";TAB(VSCALE+7)"*"
20900 NEXT T
20905 I=I+20
20910 PRINT "more (y/n) ";
20920 INPUT YESNO$
20930 IF YESNO$ <> "n" THEN GOTO 20100
20990 RETURN
30000 REM**************** low to high waveform **************
30010 I = 0
30100 CLS
30200 PRINT "Tns | ---- Vout (volts) ++++ "
30300 PRINT "     3....+....2....+....1....+....0....+....1....+....
      2....+....3....+....4"
30400 FOR T = (1+ I) TO (20 + I)
30500 VOUT = VLH*(1-(EXP(-(ALPHALH*T*1E-09))*(((ALPHALH/BETALH)*
SIN(BETALH*T*1E-09))+COS(BETALH*T*1E-09))))+VOL
30600 VSCALE = INT((ABS(3+VOUT)*10)+.5)
30700 IF VSCALE > 70 THEN VSCALE = 70
30800 PRINT T;TAB(6);"|";TAB(VSCALE+7)"*"
30900 NEXT T
30905 I=I+20
30910 PRINT "more (y/n) ";
30920 INPUT YESNO$
30930 IF YESNO$ <> "n" THEN GOTO 30100
30990 RETURN
```

RLC Circuit Program Listing for MS-DOS (Cont'd.)

Figure A-12

```
REM This is a transcription of the Transmission Line Analyzer
REM from the 29K Memory Handbook
REM Copyright Advanced Micro Devices Inc 1988
REM Transcription by Tom Crawford Jun 88
REM Assign Initial Values
decf$="####.#^^^^"
dec3$="###.###"
voh=4
vol=.2
rl=5              'totem pole resistance to ground
rh=25             'totem pole resistance to vcc
rd=22             'series damping resister
er=5              'dielectric constant
cl=7              'memory input cap in pF
t=.003            'trace thickness in inches
h=.03             'height of trace above ground in inches
w=.01             'width of trace
sp=.75            'spacing between memory chips in inches
rld=1000000!      'end of line load resistance
l=6               'total length of trace
obscure=1         'we need to redraw windows one and two

CALL TEXTFONT(4)                    'computer looking output

REM open the windows
currentfield=1                'the field we moved out of
junk=DIALOG(0)                'take any left over dialog away

loop:
  IF obscure=1 THEN GOSUB openone          'make the normal windows
  d0=DIALOG(0)                'get any dialog
  IF d0=0 THEN GOTO loop      'wait for something to happen
  ON d0 GOSUB  butt,cfield,cwindow,goaway,refresh,retkey,tabkey
  GOTO loop

tabkey:
   currentwindow=WINDOW(0)          'save current output window
  WINDOW OUTPUT 2                   'choose utility window
  CLS
  PRINT "Tab Key in Active Window"
  WINDOW OUTPUT currentwindow
  RETURN

retkey:
  GOTO gotok
```

Transmission Line Program Listing for Macintosh

Figure A-12

```
refresh:
  RETURN

goaway:
STOP

cwindow:
    currentwindow=WINDOW(0)          'save current output window
  WINDOW OUTPUT 2                    'choose utility window
  CLS
  PRINT "User Clicked in inActive Window ";DIALOG(3)
  WINDOW OUTPUT currentwindow
  RETURN

cfield:
    currentwindow=WINDOW(0)          'save current output window
    editstring$=EDIT$(currentfield)  'see what he changed it to
  WINDOW OUTPUT 2                    'choose utility window
  CLS
  PRINT "Clicked out of field ";currentfield
  PRINT "The string is ";editstring$
  ON currentfield GOSUB vohx,volx,rlx,rhx,rdx,erx,clx,tx,hx,wx,spx,rldx,lx
  d2=DIALOG(2)                       'field we clicked into
  PRINT "Clicked into new field ";d2
  IF d2<> 0 THEN currentfield=d2
  WINDOW OUTPUT currentwindow
  RETURN

vohx:
  voh=VAL(editstring$): PRINT voh: RETURN
volx:
  vol=VAL(editstring$): PRINT vol: RETURN
rlx:
  rl=VAL(editstring$): PRINT rl: RETURN
rhx:
  rh=VAL(editstring$): PRINT rh: RETURN
rdx:
  rd=VAL(editstring$): PRINT rd: RETURN
erx:
  er=VAL(editstring$): PRINT er: RETURN
clx:
  cl=VAL(editstring$): PRINT cl: RETURN
tx:
  t=VAL(editstring$): PRINT t: RETURN
```

Transmission Line Program Listing for Macintosh (Cont'd.)

Figure A-12

```
hx:
   h=VAL(editstring$): PRINT h: RETURN
wx:
   w=VAL(editstring$): PRINT w: RETURN
spx:
   sp=VAL(editstring$): PRINT sp: RETURN
rldx:
   rld=VAL(editstring$): PRINT rld: RETURN
lx:
   l=VAL(editstring$): PRINT l: RETURN

butt:
    currentwindow=WINDOW(0)          'save current output window
  d1=DIALOG(1)
  IF d1=14 THEN GOTO gotok           'do this before swapping windows
  WINDOW OUTPUT 2                    'choose utility window
  CLS
  ON d1 GOSUB  vohh,volh,rlh,rhh,rdh,erh,clh,th,hh,wh,sph,rldh,lh
  WINDOW OUTPUT currentwindow
  RETURN
vohh:
   PRINT "vOH is the HIGH level output"
   PRINT "voltage.  For CMOS it is typically"
   PRINT "between Vcc and Vcc -1.0 Volts."
   PRINT "For TTL it is typically between"
   PRINT "2.5 and 3.5 Volts.  The units are"
   PRINT "volts.";
   RETURN
volh:
   PRINT "vOL is the LOW level output"
   PRINT "voltage.  For CMOS it is typically"
   PRINT "between 0.2V and ground.  For TTL"
   PRINT"it is typically between 0.4V and"
   PRINT "ground.  The units are volts."
   RETURN
rlh:
   PRINT "RL is the totem pole resistance"
   PRINT "to ground.  It is typically on the"
   PRINT "order of 5 - 10 ohms.  The units are"
   PRINT "ohms."
   RETURN
rhh:
   PRINT "RH is the totem pole resistance"
   PRINT "to VCC.  It is typically on the order"
   PRINT "of a few tens of ohms.  The units are"
```

Transmission Line Program Listing for Macintosh (Cont'd.)

Figure A-12

```
    PRINT "ohms."
    RETURN
rdh:
    PRINT "RD is the series output resistance."
    PRINT "It is typically on the order of a few"
    PRINT "tens of ohms.  The units are ohms."
    RETURN
erh:
    PRINT "ER is the dielectric constant of the"
    PRINT "printed circuit board.  Typical "
    PRINT"numbers are between 4.7 and 5.  "
    PRINT"This is a dimensionless number."
    RETURN
clh:
    PRINT"CL is the input capacitance of each "
    PRINT"memory device.  Typical numbers"
    PRINT"are 5-7 picoFarads.  The units are "
    PRINT"picoFarads."
    RETURN
th:
    PRINT"T is the thickness of the pcb "
    PRINT"trace.  1 oz copper is .0015 inch "
    PRINT"and 2 oz copper is .003 inch.  "
    PRINT"The units are inches."
    RETURN
hh:
    PRINT"H is the height of the pcb trace"
    PRINT"above the (AC) ground plane.  Four"
    PRINT"layer boards are typically .03 inch"
    PRINT" and six layer boards are typically"
    PRINT".02 inch.  The units are inches."
    RETURN
wh:
    PRINT"W is the width of the pcb trace.  "
    PRINT"The units are inches."
    RETURN

sph:
    PRINT "SP is the spacing between memory "
    PRINT"chips along the transmission line.  "
    PRINT "The units are inches."
    RETURN
rldh:
    PRINT"RLD is the termination resistor "
    PRINT"at the end of the transmission"
```

Transmission Line Program Listing for Macintosh (Cont'd.)

Figure A-12

```
    PRINT"line furthest from the driver.  "
    PRINT"The units are ohms."
    RETURN
lh:
    PRINT"L is the length of the transmission "
    PRINT"line.   The units are inches."
    RETURN


gotok:
    GOSUB cfield                'take care of last field we clicked out of
    REM ok now do the arithmetic
     zo=(87/SQR(er+1.41))*LOG((5.98*h)/(.8*w+t))
     tpdo=1.017*SQR(.475*er+.67)
      co=1000*(tpdo/zo)
     cd=(cl/sp)*12              'cl already in picofarads
     z1=zo/SQR(1+(cd/co))
     tpd1=tpdo*SQR(1+(cd/co))
     es=voh-vol
       pl=(rld-z1)/(rld+z1)
      rsourcehl=rd+rl
       pshl=(rsourcehl-z1)/(rsourcehl+z1)
      rsourcelh=rd+rh
       pslh=(rsourcelh-z1)/(rsourcelh+z1)
     vdchl=voh
     vdclh=vol
       vahl=-1*es*(z1/(z1+rsourcehl))
       valh=es*(z1/(z1+rsourcelh))
     currentwindow=WINDOW(0)        'save current window
    WINDOW OUTPUT 2                 'utility window
    CLS
    PRINT "Driver step (Volts):";TAB(20);:PRINT USING dec3$;es
    PRINT "Line impedance (ohms):";TAB(20);:PRINT USING dec3$;z1
    PRINT "Line capacitance (pF):";TAB(20);:PRINT USING dec3$;cd*(l/12)+co*(l/12)
    PRINT "Line inductance (nH):";TAB(20);:PRINT USING dec3$;(zo^2*co*.001)*(l/12)
    PRINT "Line prop rate(nS/ft):";TAB(20);:PRINT USING dec3$;tpd1
    PRINT "Line prop delay (nS):";TAB(20);:PRINT USING dec3$;(l/12)*tpd1
    PRINT "Click Mouse to continue..."
    WHILE MOUSE(0)=0 AND DIALOG(0)=0
      WEND
     REM now do a lattice diagram
    WINDOW 3,"Lattice Diagram",(1,16)-(500,320),1
     obscure=1
    WINDOW OUTPUT 3
    CLS
```

Transmission Line Program Listing for Macintosh (Cont'd.)

Figure A-12

```
   PRINT TAB(18);"High to Low:";TAB(45);"Low to High:"
   PRINT TAB(4);"TD";TAB(10);"time";TAB(18);"Vs";TAB(30);"Vl";TAB(45);"Vs";TAB(5
4);"Vl"
   f1$="####      ###.###   ###.###                                    ###.###"
   f2$="####      ###.###                    ###.###                              ###.###"
   uhl=0
   ulh=0
   FOR td=0 TO 13
    uhl=pl^(INT(td/2+.5))*pshl^(INT(td/2))+uhl
    ulh=pl^(INT(td/2+.5))*pslh^(INT(td/2))+ulh
     vthl=(vahl*uhl)+vdchl
     vtlh=(valh*ulh)+vdclh
   IF ((td/2 - INT (td/2))>0) THEN GOTO pf2
   PRINT USING  f1$;td;td*tpd1*(l/12);vthl;vtlh
   GOTO pf3
pf2:
   PRINT USING  f2$;td;td*tpd1*(l/12);vthl;vtlh
pf3:
   NEXT td
   PRINT "Click Mouse to continue..."
wait2:
  IF MOUSE(0)=0 THEN GOTO wait2
  RETURN

openone:
REM open and update window number 1
WINDOW  2, "Utility  Window",(251,40)-(500,180),1

REM now make them strings suitable for MacEditFields
voh$=LEFT$(STR$(voh),6)
vol$=LEFT$(STR$(vol),6)
rl$=LEFT$(STR$(rl),6)
rh$=LEFT$(STR$(rh),6)
rd$=LEFT$(STR$(rd),6)
er$=LEFT$(STR$(er),6)
cl$=LEFT$(STR$(cl),6)
t$=LEFT$(STR$(t),6)
h$=LEFT$(STR$(h),6)
w$=LEFT$(STR$(w),6)
sp$=LEFT$(STR$(sp),6)
IF rld<1000 THEN
   rld$=LEFT$(STR$(rld),6)              'ohms case
  ELSE
   rld$=LEFT$(STR$(rld/1000000!),6)              'megohms case
   rld$=rld$+"E6"                                'fake it for edit field
```

Transmission Line Program Listing for Macintosh (Cont'd.)

Figure A-12

```
    END IF
l$=LEFT$(STR$(l),6)
WINDOW 1,"Parameter Values",(1,40)-(250,180),1
fbx=60:fby=5                    'upper left corner of first edit field
fex=100:fey=18                  'lower right corner of first edit field
bbx=5:bby=5                     'upper left corner of first button
bex=60:bey=18                   'lower right corner of first button
incx=120:incy=19                'button and field spacing
BUTTON 1,1,"vOH",(bbx+0*incx,bby+0*incy)-(bex+0*incx,bey+0*incy),3
EDIT FIELD 1,voh$,(fbx+0*incx,fby+0*incy)-(fex+0*incx,fey+0*incy),1
BUTTON 2,1,"vOL",(bbx+0*incx,bby+1*incy)-(bex+0*incx,bey+1*incy),3
EDIT FIELD 2,vol$,(fbx+0*incx,fby+1*incy)-(fex+0*incx,fey+1*incy),1
BUTTON 3,1,"RL",(bbx+0*incx,bby+2*incy)-(bex+0*incx,bey+2*incy),3
EDIT FIELD 3,rl$,(fbx+0*incx,fby+2*incy)-(fex+0*incx,fey+2*incy),1
BUTTON 4,1,"RH",(bbx+0*incx,bby+3*incy)-(bex+0*incx,bey+3*incy),3
EDIT FIELD 4,rh$,(fbx+0*incx,fby+3*incy)-(fex+0*incx,fey+3*incy),1
BUTTON 5,1,"RD",(bbx+0*incx,bby+4*incy)-(bex+0*incx,bey+4*incy),3
EDIT FIELD 5,rd$,(fbx+0*incx,fby+4*incy)-(fex+0*incx,fey+4*incy),1
BUTTON 6,1,"er",(bbx+0*incx,bby+5*incy)-(bex+0*incx,bey+5*incy),3
EDIT FIELD 6,er$,(fbx+0*incx,fby+5*incy)-(fex+0*incx,fey+5*incy),1
BUTTON 7,1,"CL",(bbx+0*incx,bby+6*incy)-(bex+0*incx,bey+6*incy),3
EDIT FIELD 7,cl$,(fbx+0*incx,fby+6*incy)-(fex+0*incx,fey+6*incy),1
BUTTON 8,1,"T",(bbx+1*incx,bby+0*incy)-(bex+1*incx,bey+0*incy),3
EDIT FIELD 8,t$,(fbx+1*incx,fby+0*incy)-(fex+1*incx,fey+0*incy),1
BUTTON 9,1,"H",(bbx+1*incx,bby+1*incy)-(bex+1*incx,bey+1*incy),3
EDIT FIELD 9,h$,(fbx+1*incx,fby+1*incy)-(fex+1*incx,fey+1*incy),1
BUTTON 10,1,"W",(bbx+1*incx,bby+2*incy)-(bex+1*incx,bey+2*incy),3
EDIT FIELD 10,w$,(fbx+1*incx,fby+2*incy)-(fex+1*incx,fey+2*incy),1
BUTTON 11,1,"SP",(bbx+1*incx,bby+3*incy)-(bex+1*incx,bey+3*incy),3
EDIT FIELD 11,sp$,(fbx+1*incx,fby+3*incy)-(fex+1*incx,fey+3*incy),1
BUTTON 12,1,"RLD",(bbx+1*incx,bby+4*incy)-(bex+1*incx,bey+4*incy),3
EDIT FIELD 12,rld$,(fbx+1*incx,fby+4*incy)-(fex+1*incx,fey+4*incy),1
BUTTON 13,1,"L",(bbx+1*incx,bby+5*incy)-(bex+1*incx,bey+5*incy),3
EDIT FIELD 13,l$,(fbx+1*incx,fby+5*incy)-(fex+1*incx,fey+5*incy),1
BUTTON 14,1,"OK",(bbx+1*incx,bby+6*incy)-(bex+1*incx,bey+6*incy),1
obscure=0                       'we can now see window one
RETURN
```

Transmission Line Program Listing for Macintosh (Cont'd.)

Figure A-13

```
REM This is a transcription of the Over and Undershoot Analyzer
REM from the 29K Memory Handbook
REM Copyright Advanced Micro Devices Inc 1988
REM Transcription by Tom Crawford Jun 88
REM Assign Initial Values

decf$="####.#^^^^"
voh=4
vol=.2
rl=5            'totem pole resistance to ground
rh=25           'totem pole resistance to vcc
rd=22           'series damping resister
lp=20           'package inductance in nanohenries
lt=108          'trace inductance in nanohenries
cl=250          'load capacitance in picofarads
ct=20           'trace capacitance in picofarads

REM now make them strings suitable for MacEditFields
voh$=LEFT$(STR$(voh),6)
vol$=LEFT$(STR$(vol),6)
rl$=LEFT$(STR$(rl),6)
rh$=LEFT$(STR$(rh),6)
rd$=LEFT$(STR$(rd),6)
lp$=LEFT$(STR$(lp),6)
lt$=LEFT$(STR$(lt),6)
cl$=LEFT$(STR$(cl),6)
ct$=LEFT$(STR$(ct),6)
CALL TEXTFONT(4)                'computer looking output

REM open the three windows
WINDOW 3,"Waveforms",(1,160)-(500,350),1
WINDOW 2, "Utility Window",(251,40)-(500,140),1
WINDOW 1,"Parameter Values",(1,40)-(250,140),1
fbx=60:fby=5                    'upper left corner of first edit field
fex=100:fey=18                  'lower right corner of first edit field
bbx=5:bby=5                     'upper left corner of first button
bex=60:bey=18                   'lower right corner of first button
incx=120:incy=19                'button and field spacing
BUTTON 1,1,"vOH",(bbx+0*incx,bby+0*incy)-(bex+0*incx,bey+0*incy),3
EDIT FIELD 1,voh$,(fbx+0*incx,fby+0*incy)-(fex+0*incx,fey+0*incy),1
BUTTON 2,1,"vOL",(bbx+0*incx,bby+1*incy)-(bex+0*incx,bey+1*incy),3
EDIT FIELD 2,vol$,(fbx+0*incx,fby+1*incy)-(fex+0*incx,fey+1*incy),1
BUTTON 3,1,"RL",(bbx+0*incx,bby+2*incy)-(bex+0*incx,bey+2*incy),3
EDIT FIELD 3,rl$,(fbx+0*incx,fby+2*incy)-(fex+0*incx,fey+2*incy),1
BUTTON 4,1,"RH",(bbx+0*incx,bby+3*incy)-(bex+0*incx,bey+3*incy),3
```

Over and Undershoot Analyzer for Macintosh

Figure A-13

```
EDIT FIELD  4,rh$,(fbx+0*incx,fby+3*incy)-(fex+0*incx,fey+3*incy),1
BUTTON  5,1,"RD",(bbx+0*incx,bby+4*incy)-(bex+0*incx,bey+4*incy),3
EDIT FIELD  5,rd$,(fbx+0*incx,fby+4*incy)-(fex+0*incx,fey+4*incy),1
BUTTON  6,1,"LP",(bbx+1*incx,bby+0*incy)-(bex+1*incx,bey+0*incy),3
EDIT FIELD  6,lp$,(fbx+1*incx,fby+0*incy)-(fex+1*incx,fey+0*incy),1
BUTTON  7,1,"LT",(bbx+1*incx,bby+1*incy)-(bex+1*incx,bey+1*incy),3
EDIT FIELD  7,lt$,(fbx+1*incx,fby+1*incy)-(fex+1*incx,fey+1*incy),1
BUTTON  8,1,"CL",(bbx+1*incx,bby+2*incy)-(bex+1*incx,bey+2*incy),3
EDIT FIELD  8,cl$,(fbx+1*incx,fby+2*incy)-(fex+1*incx,fey+2*incy),1
BUTTON  9,1,"CT",(bbx+1*incx,bby+3*incy)-(bex+1*incx,bey+3*incy),3
EDIT FIELD  9,ct$,(fbx+1*incx,fby+3*incy)-(fex+1*incx,fey+3*incy),1
BUTTON  10,1,"OK",(bbx+1*incx,bby+4*incy)-(bex+1*incx,bey+4*incy),1
currentfield=9                  'the field we moved out of
junk=DIALOG(0)                  'take any left over dialog away

loop:
   d0=DIALOG(0)                 'get any dialog
   IF d0=0 THEN GOTO loop       'wait for something to happen
   ON d0 GOSUB  butt,cfield,cwindow,goaway,refresh,retkey,tabkey
   GOTO loop

tabkey:
    currentwindow=WINDOW(0)             'save current output window
   WINDOW OUTPUT 2                      'choose utility window
   CLS
   PRINT "Tab Key in Active Window"
   WINDOW OUTPUT  currentwindow
   RETURN

retkey:
   GOTO gotok

refresh:
   RETURN

goaway:
   STOP

cwindow:
    currentwindow=WINDOW(0)             'save current output window
   WINDOW OUTPUT 2                      'choose utility window
   CLS
   PRINT "User Clicked in inActive Window ";DIALOG(3)
   WINDOW OUTPUT  currentwindow
   RETURN
```

Over and Undershoot Analyzer for Macintosh (Cont'd.)

Figure A-13

```
cfield:
    currentwindow=WINDOW(0)          'save current output window
    editstring$=EDIT$(currentfield)  'see what he changed it to
  WINDOW OUTPUT 2                    'choose utility window
  CLS
  PRINT "Clicked out of field ";currentfield
  PRINT "The string is ";editstring$
  ON currentfield GOSUB  vohx,volx,rlx,rhx,rdx,lpx,ltx,clx,ctx
   d2=DIALOG(2)                      'field we clicked into
  PRINT "Clicked into new field ";d2
  IF d2<> 0 THEN  currentfield=d2
  WINDOW OUTPUT  currentwindow
  RETURN

vohx:
   voh=VAL(editstring$): PRINT voh: RETURN
volx:
   vol=VAL(editstring$): PRINT vol: RETURN
rlx:
   rl=VAL(editstring$): PRINT rl: RETURN
rhx:
   rh=VAL(editstring$): PRINT rh: RETURN
rdx:
   rd=VAL(editstring$): PRINT rd: RETURN
lpx:
   lp=VAL(editstring$): PRINT lp: RETURN
ltx:
  lt=VAL(editstring$): PRINT lt: RETURN
clx:
   cl=VAL(editstring$): PRINT cl: RETURN
ctx:
  ct=VAL(editstring$): PRINT ct: RETURN

butt:
    currentwindow=WINDOW(0)          'save current output window
  d1=DIALOG(1)
  IF d1=10 THEN GOTO gotok           'do this before swapping windows
  WINDOW OUTPUT 2                    'choose utility window
  CLS
  ON d1 GOSUB  vohh,volh,rlh,rhh,rdh,lph,lth,clh,cth
  WINDOW OUTPUT  currentwindow
  RETURN
vohh:
  PRINT "vOH is the HIGH level output"
```

Over and Undershoot Analyzer for Macintosh (Cont'd.)

Figure A-13

```
    PRINT "voltage.  For CMOS it is typically"
    PRINT "between Vcc and Vcc -1.0 Volts."
    PRINT "For TTL it is typically between"
    PRINT "2.5 and 3.5 Volts.  The units are"
    PRINT "volts.";
    RETURN

volh:
    PRINT "vOL is the LOW level output"
    PRINT "voltage.  For CMOS it is typically"
    PRINT "between 0.2V and ground.  For TTL"
    PRINT"it is typically between 0.4V and"
    PRINT "ground.  The units are volts."
    RETURN

rlh:
    PRINT "RL is the totem pole resistance"
    PRINT "to ground.  It is typically on the"
    PRINT "order of 5 - 10 ohms.  The units are"
    PRINT "ohms."
    RETURN

rhh:
    PRINT "RH is the totem pole resistance"
    PRINT "to VCC.  It is typically on the order"
    PRINT "of a few tens of ohms.  The units are"
    PRINT "ohms."
    RETURN
rdh:
    PRINT "RD is the series output resistance."
    PRINT "It is typically on the order of a few"
    PRINT "tens of ohms.  The units are ohms."
    RETURN

lph:
    PRINT "LP is the package inductance.  It is"
    PRINT "typically around 10-20 nanoHenries."
    PRINT "The units are nanoHenries."
    RETURN

lth:
    PRINT "LT is the total trace inductance."
    PRINT "The units are nanoHenries."
    RETURN
```

Over and Undershoot Analyzer for Macintosh (Cont'd.)

Figure A-13

```
clh:
  PRINT"CL is the total load capacitance.  It"
  PRINT"is typically 5-10 picoFarads per "
  PRINT"memory device.  The units are"
  PRINT"picoFarads."
  RETURN

cth:
  PRINT "CT is the total trace capacitance."
  PRINT "The unit are picoFarads."
  RETURN


gotok:
  GOSUB cfield                'take care of last field we clicked out of
   vhl=-(voh-vol)
   vlh=voh-vol
   rhl=rl+rd
   rlh=rh+rd
   l=1E-09*(lp+lt)            'make this into henries
   c=1E-12*(cl+ct)            'and this into farads
    lcinv=1/(l*c)
     r24l2hl=(rhl^2)/(4*(l^2))
     r24l2lh=(rlh^2)/(4*(l^2))
    alphahl=rhl/(2*l)
    alphalh=rlh/(2*l)
   betahl=SQR(ABS(lcinv-r24l2hl))
   betalh=SQR(ABS(lcinv-r24l2lh))
    currentwindow=WINDOW(0)          'choose the utility window
  WINDOW OUTPUT 2
  CLS
  PRINT TAB(8);"HILO";TAB(18);"LOHI"
  PRINT "Volts";TAB(8);vhl;TAB(18);vlh
  PRINT "Resis";TAB(8);rhl;TAB(18);rlh
  PRINT"R^2/4L^2";TAB(8);:PRINT USING decf$;r24l2hl;:PRINT TAB(18);
  PRINT USING decf$;r24l2lh
  PRINT "R/2L";TAB(8);:PRINT USING decf$;alphahl;:PRINT TAB(18);
  PRINT USING decf$;alphalh
  PRINT"Beta";TAB(8);:PRINT USING decf$;betahl;:PRINT TAB(18);
  PRINT USING decf$;betalh;

  REM now draw the scales on the plotter
  WINDOW OUTPUT 3          'choose the plotter window
  CLS
   vscale= -16             'pixels per volt vertically (plus is up on screen)
```

Over and Undershoot Analyzer for Macintosh (Cont'd.)

Figure A-13

```
 vzero=-7*vscale+20          '+7 volts to -3 volts
 hzero=20
 hscale=7                    'pixel per nsec
 htotal=60                   'we will always plot the same number of ns
LINE (hzero,vzero)-((hscale*htotal)+hzero,vzero),33          'zero volts
FOR nsec = 0 TO htotal
  LINE ((nsec*hscale)+hzero,vzero+2)-((nsec*hscale)+hzero,vzero-2)
  NEXT nsec

FOR nsec = 0 TO htotal STEP 5
  LINE ((nsec*hscale)+hzero,vzero+5)-((nsec*hscale)+hzero,vzero-5)
  NEXT nsec

FOR nsec = 0 TO htotal STEP 10
  LINE ((nsec*hscale)+hzero,vzero+10)-((nsec*hscale)+hzero,vzero-10)
  NEXT nsec

LINE (hzero,vzero-(vscale*3))-(hzero,vzero+(vscale*7))
FOR volts=-3 TO 7
  LINE (hzero-2,vzero+(vscale*volts))-(hzero+2,vzero+(vscale*volts))
  NEXT volts

  REM now plot the high to low transition
  FOR nsec=1 TO htotal
     t=nsec*1E-09                 'seconds units
     cospart=COS(betahl*t)
      sinpart=SIN(betahl*t)
       volts=vhl*(1-(EXP(-alphahl*t))*((alphahl/betahl)*sinpart+cospart))+voh
    CIRCLE(hzero+(nsec*hscale),vzero+(volts*vscale)),2
    NEXT nsec

  REM now plot the low to high transition
  FOR nsec=1 TO htotal
     t=nsec*1E-09                 'seconds units
     cospart=COS(betalh*t)
      sinpart=SIN(betalh*t)
       volts=vlh*(1-(EXP(-alphalh*t))*((alphalh/betalh)*sinpart+cospart))+vol
    CIRCLE(hzero+(nsec*hscale),vzero+(volts*vscale)),1
    NEXT nsec


WINDOW OUTPUT currentwindow
RETURN
```

Over and Undershoot Analyzer for Macintosh (Cont'd.)

APPENDIX B
Building a Single-Cycle Memory System

Overview B-1
Up Against the Wall B-1
The Side Effects, Nothing To Spare B-2
System Clock Provided by Processor B-2
System Clock Provided by External Oscillator B-4
Timing is Everything B-5

Appendix B

BUILDING A SINGLE-CYCLE MEMORY SYSTEM

OVERVIEW

The designers of the Am29000 spent a great deal of time and silicon to build a processor that can provide the best in state-of-the-art performance, without the requirement for single-cycle memory access speed.

The branch target cache is able to hide three cycles of access time, typically, in 60% of all branch instruction executions. The instruction prefetch buffer can in many cases hide additional instruction access time.

The large register file reduces the need to load or store data since the variables for multiple procedures may be held in the register file across procedure calls and returns. Overlapping of loads and stores with continued instruction execution further hides data memory access time. Therefore, in most cases, slower and less expensive memory systems can serve nearly as well as if single-cycle memory were used.

But even so, there will always be someone who wants to squeeze out every last ounce of performance regardless of the difficulty or cost. To that end, this Appendix describes the constraints imposed on a single-cycle memory system and Figure B-1 shows how to build one. The fundamental constraint on single-cycle memory is that its access time must be equal to, or better than the time leftover from one clock cycle after processor address and control delay and data and instruction setup time are subtracted.

UP AGAINST THE WALL

The processor address and control lines are not valid until 14 ns into a clock cycle. The processor-instruction and data-setup times are 6 ns. That leaves 20 ns from a 40 ns cycle. Even this available time must be reduced by buffer delays or capacitance-load delay where the memory load on the processor address lines exceeds the standard capacitance-load limit.

Finally, there is the problem presented by the need to control the $\overline{\text{Chip Enable}}$ ($\overline{\text{CE}}$) signal to the memory so that the memory will not contend for the bus during the early part of a write operation.

The problem is that until 14 ns into the cycle, the write control signal from the processor is not valid and may indicate a read or write operation incorrectly. If the memory were enabled throughout each cycle, it would be possible for the memory to present read data at the same time that write data from the processor begins to be driven for a write operation. This contention results from the memory seeing a read operation before the memory's $\overline{\text{Write Enable}}$ ($\overline{\text{WE}}$) line becomes active and valid. Bus contention can then continue until the $\overline{\text{WE}}$ line has time to disable the memory read-data output. In addition, there is no guarantee that the $\overline{\text{WE}}$ line will not have spurious noise-induced $\overline{\text{WE}}$ pulses before the processor's valid output delay time is satisfied.

It is therefore clear that a single-cycle access time memory should not be chip enabled prior to the end of the output valid delay for the processor's Read/$\overline{\text{Write}}$ (R/$\overline{\text{W}}$) line. System Clock (SYSCLK) is a very convenient signal to use as the $\overline{\text{CE}}$ control. It is high during the first half of the cycle and disables the memory; and it is low during the latter half of the clock cycle when the address and R/$\overline{\text{W}}$ lines are stable.

Using the SYSCLK as $\overline{\text{CE}}$ provides both a solution and a limitation. The limitation is that the system clock can go active no sooner than 19 ns and may be as late as 21 ns. This says that the limit on available access time for the memory is set by the time remaining after the SYSCLK delay and processor instruction or data setup time are subtracted from a 40 ns clock cycle. That is, 40 ns – 21 ns – 6 ns = 13 ns.

THE SIDE EFFECTS, NOTHING TO SPARE

With only 13 ns available for memory access time, there is simply no time available for dynamic address decoding or data-path buffering. Address lines may be buffered since there is 5 ns to 7 ns available between the time that address from the processor is valid and the time that the memory $\overline{\text{CE}}$ provided by SYSCLK is active. $\overline{\text{CE}}$ must be provided directly from SYSCLK, or from a signal with the same timing specification as SYSCLK, since $\overline{\text{CE}}$ is in the critical timing path.

Within these restrictions,there are at least two possible implementation approaches. The two approaches differ in the way that SYSCLK is delivered to the system. The first scheme is the simple direct use of SYSCLK as provided by the Am29000 processor. The second approach relies on clock generation and gating logic external to the the Am29000 processor.

SYSTEM CLOCK PROVIDED BY PROCESSOR

The single-cycle memory with processor provided SYSCLK signal is shown in Figure B-1.

Potential Clock Overload

The system clock, if derived from the processor, is very heavily loaded with capacitance because it must drive all the memories in the instruction and data blocks. The system clock may not be buffered, because to do so would add delay into the $\overline{\text{CE}}$-signal path of the memories. These added delays would reduce the available read access time.

Limited Memory Size

Unless the memory devices used have multiple $\overline{\text{CE}}$ inputs, there can only be a single block of memory in the instruction space and one block in the data space. Additional blocks require either address decoding to select the blocks or data path buffers that can isolate the blocks from the bus; neither of which is possible when the processor provides the clock.

Special Method-To-Access Instruction Memory Is Needed

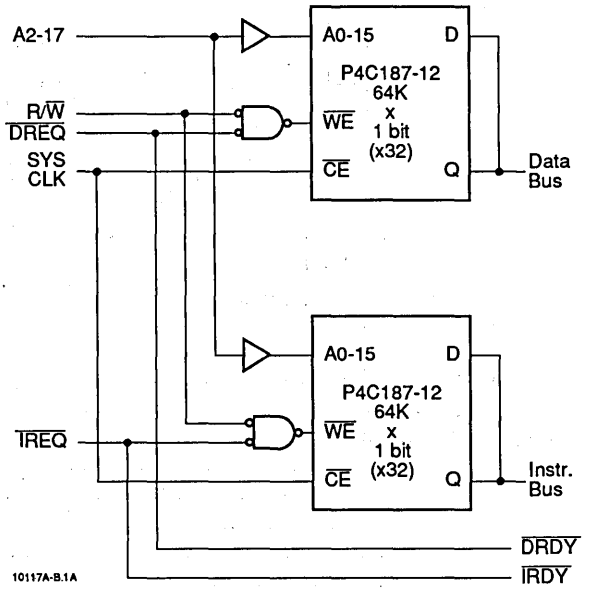
The data-and-instruction memory blocks are both being selected for read or write in the latter half of every cycle. It is therefore not possible to give the instruction memory access to the data bus so that the instruction memory can be loaded and read via the data bus. If this were attempted, the data memory would always contend with the instruction-memory-to-data-bus buffer.

Therefore to gain access to the instruction memory, it is necessary to provide a DMA device that can request the bus from the processor. This DMA device must have the buffers necessary to gain access to either the data or instruction bus. The DMA device is responsible for moving instructions into the instruction memory via the instruction bus. The instructions, most likely, come from a remote bus which the DMA device could access.

$\overline{WE}$

Again, because the data and instruction memory blocks are both being selected for read or write in the latter half of every cycle, it is necessary to qualify the $\overline{WE}$ line to the memories with the appropriate Memory Request signal. That way a data bus write affects only the data memory and an instruction bus write, via the DMA device, affects only the instruction memory.

Figure B-1



Single-Cycle Memory with Processor-Provided SYSCLK Signal

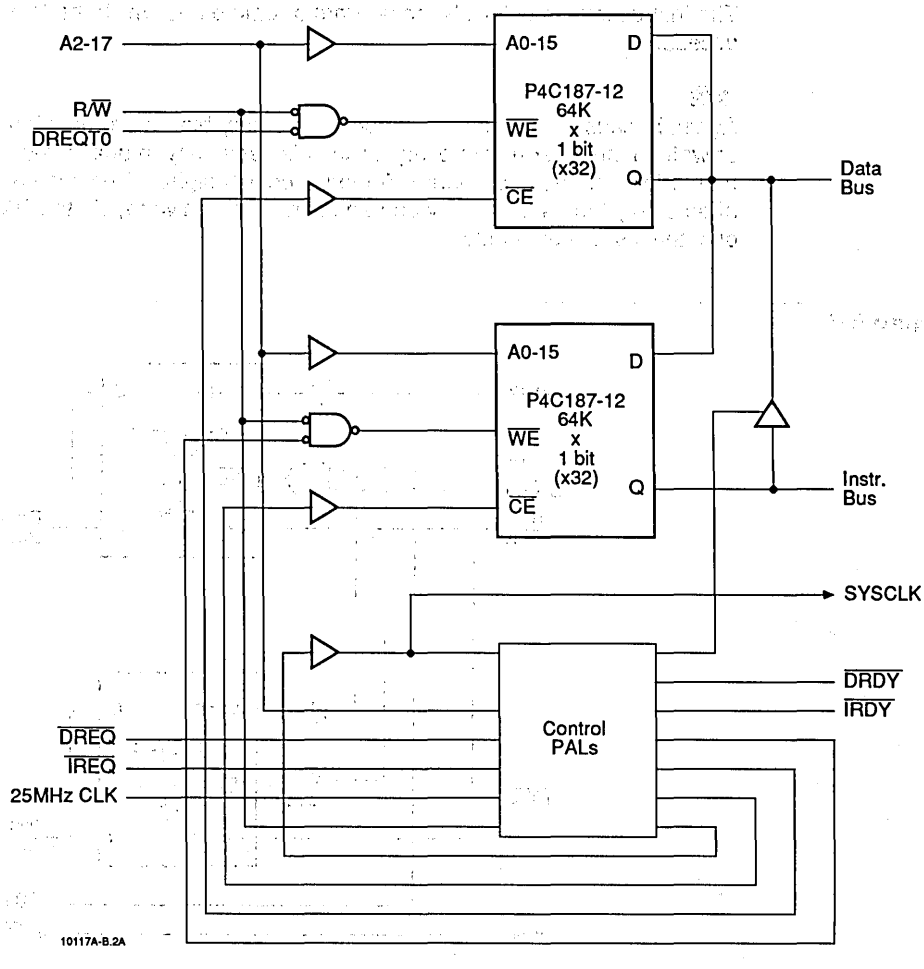
SYSTEM CLOCK PROVIDED BY EXTERNAL OSCILLATOR

The single-cycle memory with bank selection is shown in Figure B-2.

Lower Clock Loading Possible

If SYCLK is provided to both the processor and the memory blocks from an external oscillator, multiple clock buffers can be used to split the memory capacitance load. The delay of the memory clock buffers would be in parallel with the delay of the clock buffer driving the processor. This would maintain the timing relationship between the processor and memory without inducing additional delay.

Figure B-2



Single-Cycle Memory with Bank Selection

Address Decoding, Multiple Memory Banks, Now Possible

By splitting clock distribution, it is possible to selectively qualify each SYSCLK signal used as a memory $\overline{CE}$ signal. This is done by passing SYSCLK from the external oscillator through a PAL which selectively qualifies each output clock. The qualified clocks then go through buffers that drive the memory arrays. By passing all the clocks through the same gating and buffering levels the phase relationship of all the clocks can be maintained, i.e. minimize system clock skew. The ability to qualify the $\overline{CE}$ line now allows multiple memory banks within the instruction or data blocks to be addressed.

Due to the skew between the input oscillator signal and SYSCLK, the bank selection cannot be changed on a cycle-by-cycle basis. It is only possible to register a value that selects a given memory bank. The switching process from one bank to another takes at least one cycle. This switching of banks can be done by an explicit access to some specific address. The PAL control logic recognizes the address and loads the registers that gate the $\overline{CE}$. The next memory access is then directed to the newly selected memory bank.

Simpler Access to Instruction Memory

Since it is possible to deselect all data memory banks and enable a buffer to connect an instruction memory bank to the data bus, the processor can directly access one bank of instruction memory as data while executing code from another bank of instruction RAM or ROM. The added delay of the instruction bus-to-data bus buffer requires that these data bus accesses of instruction memory be slowed to two cycles per access via control over the $\overline{DRDY}$.

TIMING IS EVERYTHING

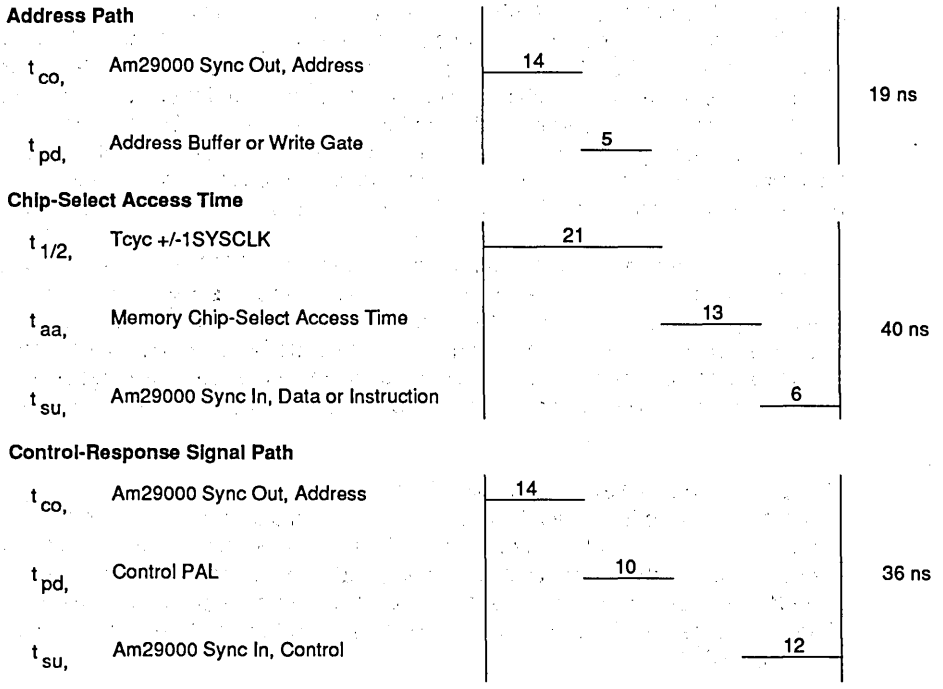
The timing for a single-cycle memory access is shown in Figure B-3.

As noted earlier, when SYSCLK is used as $\overline{CE}$, it becomes part of the critical path. This critical path, is made up of the worst-case system-clock output delay, plus memory access time, plus processor set-up time, it's total delay is 40 ns

The control-to-$\overline{CE}$ signal path is the next most critical. This critical path is the processor control output valid delay of 14 ns. Of the total 19 to 21 ns delay possible, this leaves 5 ns until the earliest point at which SYSCLK ($\overline{CE}$ signal) could go active. Since it is important that the $\overline{WE}$ line and addresses settle before the chip is enabled ($\overline{CE}$ goes active) , the maximum delay for the address buffers and control gates is 5 ns. To achieve this, it may be necessary to duplicate buffers and gates so as to split up the memory array into groups whose capacitive load does not exceed the load specifications of the signal drivers.

The processor control-to-response signal path is made up of the "processor control output valid delay" of 14 ns, the PALs used to control the memory delay of 10 ns, and the processor control signal setup time delay of 12 ns for a total of 36 ns.

Figure B-3



10117A-B.3A

Single-Cycle Memory Timing

Sales Offices

North American

ALABAMA	(205) 882-9122
ARIZONA	(602) 242-4400
CALIFORNIA,	
Culver City	(213) 645-1524
Newport Beach	(714) 752-6262
San Diego	(619) 560-7030
San Jose	(408) 452-0500
Woodland Hills	(818) 992-4155
CANADA, Ontario,	
Kanata	(613) 592-0060
Willowdale	(416) 224-5193
COLORADO	(303) 741-2900
CONNECTICUT	(203) 264-7800
FLORIDA,	
Clearwater	(813) 530-9971
Ft. Lauderdale	(305) 776-2001
Orlando	(407) 830-8100
GEORGIA	(404) 449-7920
ILLINOIS,	
Chicago	(312) 773-4422
Naperville	(312) 505-9517
INDIANA	(317) 244-7207
KANSAS	(913) 451-3115
MARYLAND	(301) 796-9310
MASSACHUSETTS	(617) 273-3970
MINNESOTA	(612) 938-0001
MISSOURI	(913) 451-3115
NEW JERSEY,	
Cherry Hill	(609) 662-2900
Parsippany	(201) 299-0002
NEW YORK,	
Liverpool	(315) 457-5400
Poughkeepsie	(914) 471-8180
Woodbury	(516) 364-8020
NORTH CAROLINA	(919) 878-8111
OHIO,	
Columbus	(614) 891-6455
Dayton	(513) 439-0470
OREGON	(503) 245-0080
PENNSYLVANIA	(215) 398-8006
SOUTH CAROLINA	(803) 772-6760
TEXAS,	
Austin	(512) 346-7830
Dallas	(214) 934-9099
Houston	(713) 785-9001
WASHINGTON	(206) 455-3600
WISCONSIN	(414) 792-0590

International

BELGIUM, Bruxelles	TEL	(02) 771-91-42
	FAX	(02) 762-37-12
	TLX	61028
FRANCE, Paris	TEL	(1) 49-75-10-10
	FAX	(1) 49-75-10-13
	TLX	263282
WEST GERMANY,		
Hannover area	TEL	(0511) 736085
	FAX	(0511) 721254
	TLX	922850
München	TEL	(089) 4114-0
	FAX	(089) 406490
	TLX	523883
Stuttgart	TEL	(0711) 62 33 77
	FAX	(0711) 625187
	TLX	721882
HONG KONG	TEL	852-5-8654525
	FAX	852-5-8654335
	TLX	67955AMDAPHX
ITALY, Milan	TEL	(02) 3390541
		(02) 3533241
	FAX	(02) 3498000
	TLX	315286
JAPAN,		
Kanagawa	TEL	462-47-2911
	FAX	462-47-1729

International (Continued)

Tokyo	TEL	(03) 345-8241
	FAX	(03) 342-5196
	TLX	J24064AMDTKOJ
Osaka	TEL	06-243-3250
	FAX	06-243-3253
KOREA, Seoul	TEL	82-2-784-7598
	FAX	82-2-784-8014
LATIN AMERICA,		
Ft. Lauderdale	TEL	(305) 484-8600
	FAX	(305) 485-9736
	TEL	5109554261 AMDFTL
NORWAY, Hovik	TEL	(02) 537810
	FAX	(02) 591959
	TLX	79079
SINGAPORE	TEL	65-2257544
	FAX	65-2246113
	TLX	RS55650 MMI RS
SWEDEN,		
Stockholm	TEL	(08) 733 03 50
	FAX	(08) 733 22 85
	TLX	11602
TAIWAN	TLX	886-2-7122066
	FAX	886-2-7122017
UNITED KINGDOM,		
Manchester area	TEL	(0925) 828008
	FAX	(0925) 827693
	TLX	628524
London area	TEL	(04862) 22121
	FAX	(0483) 756196
	TLX	859103

North American Representatives

CANADA	
Burnaby, B.C.	
DAVETEK MARKETING	(604) 430-3680
Calgary, Alberta	
VITEL ELECTRONICS	(403) 278-5833
Kanata, Ontario	
VITEL ELECTRONICS	(613) 592-0090
Mississauga, Ontario	
VITEL ELECTRONICS	(416) 676-9720
Quebec	
VITEL ELECTRONICS	(514) 636-5951
IDAHO	
INTERMOUNTAIN TECH MKTG	(208) 888-6071
ILLINOIS	
HEARTLAND TECHNICAL MARKETING	(312) 577-9222
INDIANA	
ELECTRONIC MARKETING CONSULTANTS, INC	(317) 921-3452
IOWA	
LORENZ SALES	(319) 377-4666
KANSAS	
Merriam – LORENZ SALES	(913) 384-6556
Wichita – LORENZ SALES	(316) 721-0500
KENTUCKY	
ELECTRONIC MARKETING CONSULTANTS, INC	(317) 921-3452
MICHIGAN	
Birmingham – MIKE RAICK ASSOCIATES	(313) 644-5040
Holland – COM-TEK SALES, INC	(616) 399-7273
Novi – COM-TEK SALES, INC	(313) 344-1409
MISSOURI	
LORENZ SALES	(314) 997-4558
NEBRASKA	
LORENZ SALES	(402) 475-4660
NEW MEXICO	
THORSON DESERT STATES	(505) 293-8555
NEW YORK	
NYCOM, INC	(315) 437-8343
OHIO	
Centerville – DOLFUSS ROOT & CO	(513) 433-6776
Columbus – DOLFUSS ROOT & CO	(614) 885-4844
Strongsville – DOLFUSS ROOT & CO	(216) 238-0300
PENNSYLVANIA	
DOLFUSS ROOT & CO	(412) 221-4420
UTAH	
R^2 MARKETING	(801) 595-0631
WISCONSIN	
HEARTLAND TECHNICAL MARKETING	(414) 796-1128

Advanced Micro Devices, Inc. 901 Thompson Place, P.O. Box 3453, Sunnyvale, CA 94088, USA
Tel: (408) 732-2400 • TWX: 910-339-9280 • TELEX: 34-6306 • TOLL FREE: (800) 538-8450
APPLICATIONS HOTLINE TOLL FREE: (800) 222-9323 • (408) 749-5703

12/7/88
WCP-12M-12/88-2 Printed in USA

ADVANCED MICRO DEVICES, INC.
901 Thompson Place
P.O. Box 3453
Sunnyvale,
California 94088-3453
(408) 732-2400
TWX: 910-339-9280
TELEX: 34-6306
TOLL-FREE
(800) 538-8450

APPLICATIONS HOTLINE
(800) 222-9323
(408) 749-5703

Printed in USA

10623A