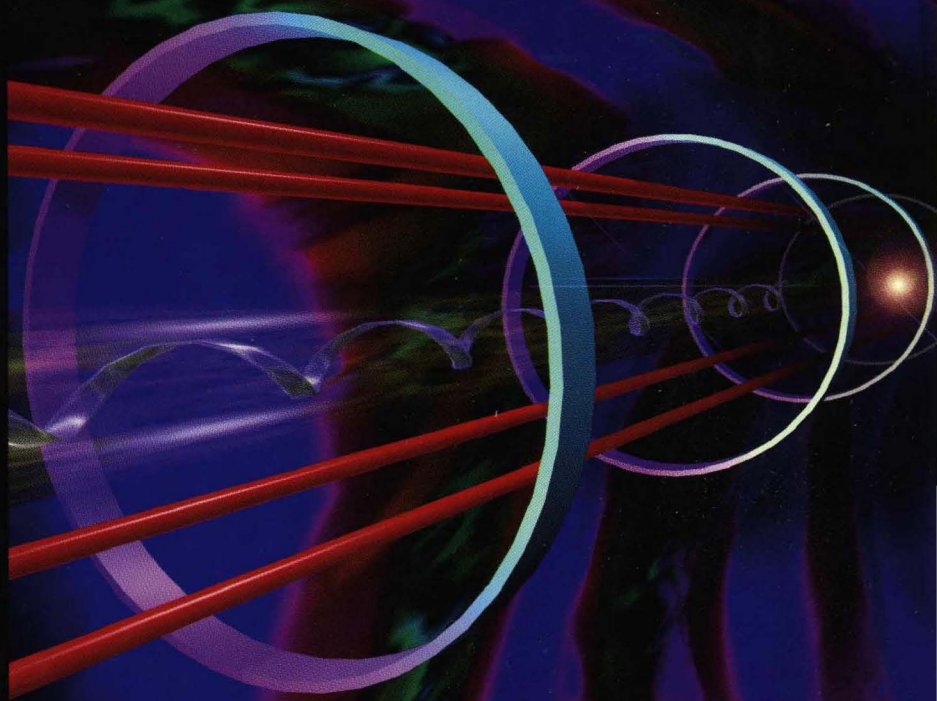
Flash Memory Databook Documentation Update

New Documents

Revised Documents

Revised Pages



intel®

intel®

1995 Flash Memory Databook Documentation Update

1995

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

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

or call 1-800-879-4683

PREFACE

Information in the 1995 Flash Memory Databook Documentation Update replaces information found in the 1995 Flash Memory Databook, Volumes I and II. In some chapters of the databook, individual pages have been revised. In some cases, entire documents have been replaced.

Please note that this update contains several additional documents not found in the 1995 Flash Memory Databook:

- AP-603 Symmetric Block Format Exchanging Data with FFS Systems
- AP-608 Implementing a Plug and Play BIOS Using Intel's Boot Block Flash Memory
- AP-609 Interfacing the Intel386™ EX Embedded Processor to Intel Flash
- AP-610 Flash Memory In-System Code and Data Update Techniques

Thank you for your interest in Intel's Flash memory products.

Contents

New Documents

intel®

AP-610

APPLICATION NOTE

Flash Memory In-System Code and Data Update Techniques

BRIAN DIPERT
SENIOR TECHNICAL
MARKETING ENGINEER

February 1995

Order Number: 292163-002

1.0 INTRODUCTION

The ability to update flash memory contents with the system operational distinguishes flash memory from other nonvolatile technologies such as ROM and EPROM. This capability is key for using flash memory in a wide range of applications:

- Code storage/execution (code DRAM and ROM replacement),
- Data storage (EEPROM, battery RAM emulation, etc.), and
- File storage (flash-based solid state drive)

System software implementations for in-system code and data update must comprehend algorithm execution during flash memory program/erase. Implementations also vary according to the level of system code/data access required during update.

This application note discusses these topics and gives general recommendations that can be tailored to specific system needs. It focuses on Intel's Boot Block, FlashFile™ and Embedded Flash RAM memories which have on-chip program/erase automation and block erasure. However, many of the concepts can be equally applied to Intel's bulk erase flash memories.

2.0 GENERAL INFORMATION

Definition of Terms

Design engineers can select from up to three unique approaches to update stored flash memory information. Before proceeding, let's define these terms to make it clear what we will and won't be discussing in this application note:

1. *In-System Write (ISW):* As first described earlier, during an in-system update the system is powered up and either partially or fully operational. The system CPU (see Figure 1) executes the flash memory program/erase algorithms and obtains new code/data from one of several sources (serial or parallel port, floppy or hard disk drive, modem, etc.).

2. *On-Board Programming (OBP):* In this approach, the flash memory is also installed on the system board. However, OBP does not use the system CPU and, in fact, commonly powers down the processor or holds it in a HALT mode. The flash memory connects to an off-board "computer" such as a board tester or prom programmer. This off-board intelligence provides the necessary commands and data to erase and reprogram the flash memory. This technique is covered in other documentation available from Intel Corporation. See the Additional Information section of this application note.

3. *PROM Programming:* This approach was first made popular back in the days of PROMs and EPROMs. The flash memory is initially programmed, and is reprogrammed, by removing it from the system board and socketing it in dedicated hardware called a PROM programmer. Intel works closely with a wide range of PROM programmer vendors to ensure support for all of its flash memory products. See the Additional Information section of this application note for details.

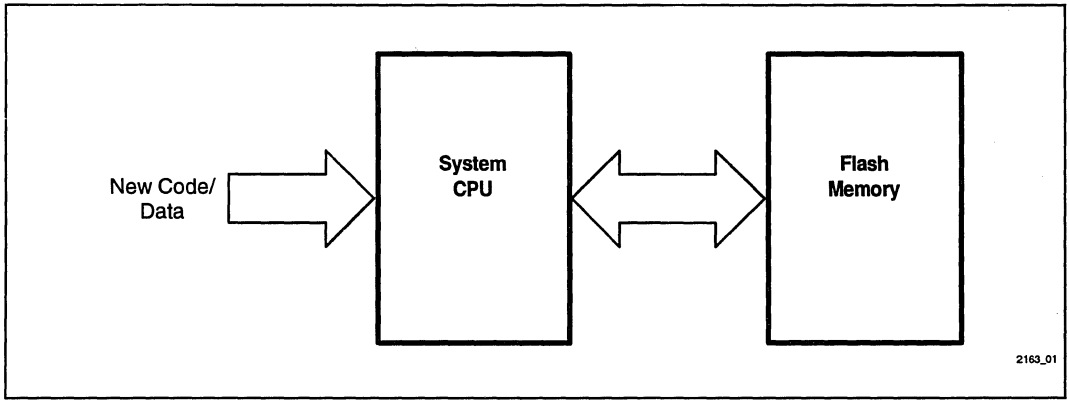


Figure 1. The System CPU Controls the In-System-Write Flash Memory Update

Read-While-Write

The fundamental concept to understand when considering in-system updates is that of read-while-write. Stated simply, it is currently NOT possible with any of today's flash memory technology alternatives to read from the flash memory array while simultaneously programming or erasing it. There are several basic reasons for this:

a. During program or erase, the flash memory row and column decode architecture results in high voltages present throughout the array. Isolating these voltages to a specific byte/word or block would have excessive die size and (therefore) silicon cost impacts given that inexpensive system implementation alternatives exist. Keep reading for details!

b. Intel's Boot Block, FlashFile and Embedded Flash RAM memories all have on-chip program/erase automation. After these flash memories receive program or erase command sequences, they automatically transition to a mode where they provide status register information (versus array or other data). This transition quickly provides the system with the information it needs to determine program/erase status, minimizing system software overhead and maximizing effective write/erase performance.

Flash memory array reads (to access code or data) CAN take place at any time that the flash memory automation is READY (either completed or suspended). Figure 2 gives a simple flash memory update algorithm example. It shows portions of the code that must be executed off-chip, and shaded areas show "windows" where the flash memory array can be accessed, if needed, by writing the Read Array command and then reading from desired locations. These "windows" will be described in detail in Section 4.0. Thanks to on-chip automation, the amount of code executed off-chip to actually program/erase the flash memory is very small. Overhead needed to obtain new code/data from the system varies with the method chosen.

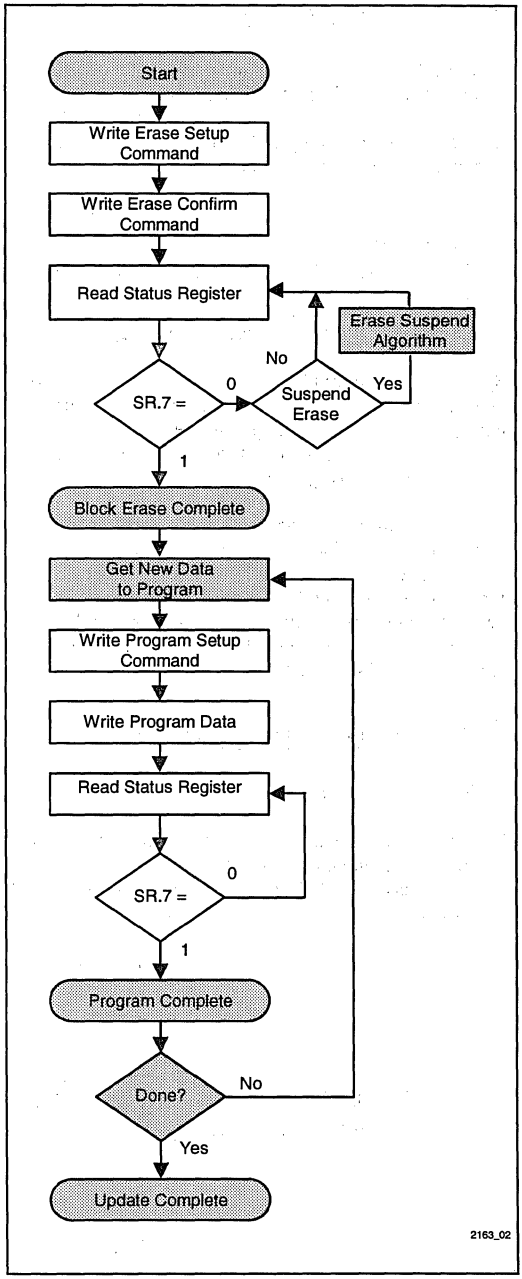


Figure 2. Simple Code/Data Block Update Algorithm Shows Shaded "Window" Opportunities for Array Reads

What Amount of System Functionality Is Needed During Update?

The answer to the above question is key to understanding the amount of software architecting needed to integrate flash memory into your design. Use the following question as a reference for where to continue reading:

Q. Can you dedicate the system exclusively to the flash memory update and ignore all other non-related interrupts? Said another way, can you take the system "off-line" during flash memory updates?

A_1. If your answer is "yes," the software implementation is very straightforward. See Sections 3.0 and 5.0.

A_2. If your answer is "no," the specific software implementation varies. One approach uses redundant system memory to separate the execution and storage/backup regions. Another technique eliminates this redundancy but depends on an understanding of interrupt latency, interrupt frequency and its variability with time. See Sections 4.0 and 5.0.

Dedicated Blocks for System Boot Code: Recovery from System Power Loss or Reset during Flash Memory Update

Several of the approaches described in Sections 3.0 and 4.0 that follow use system RAM to execute the flash memory update algorithms. This brings up a logical question; what happens if the system resets or loses power in the middle of a flash memory update? In this case, system RAM contents will be invalid, including the flash memory update code. The byte being programmed or the block being erased when system reset/power loss occurs will be left in an indeterminate state and will need to be reprogrammed/erased.

Flash memory's blocked architecture provides protection for system boot code and enables the system to recover fully from incomplete code updates. All boot block components as well as 16-/32-Mbit FlashFile and embedded flash RAM memories also allow hardware "lock" of boot code for additional protection. This boot code, after minimally initializing system hardware, should execute a checksum verify of the remainder of the flash memory. If this checksum "passes," system boot can continue. If a checksum "fail" is obtained, this reflects an incomplete program or erase, and the system should alert the user and execute a repeat update. Figure 3 flowcharts this algorithm.

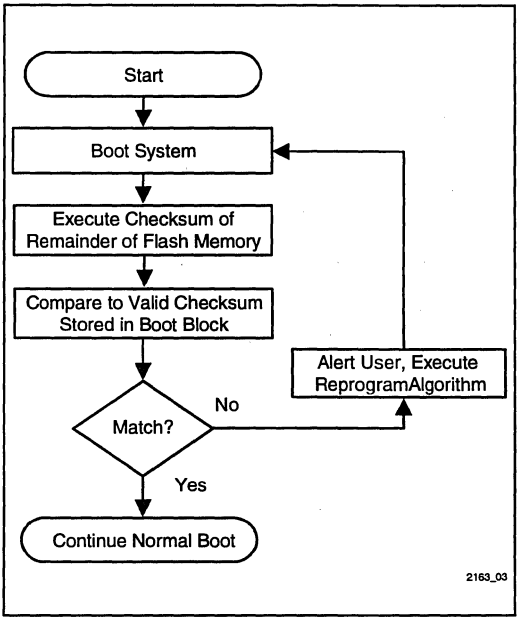


Figure 3. Checksum Validation Confirms Flash Memory Integrity

Intel's 16-/32-Mbit FlashFile and embedded flash RAM memories indicate via Status Register feedback whether an erase in progress has been aborted by power loss or hardware power-down. The 16-Mbit Flash Product Family User's Manual covers this topic in detail. See the Additional Information section of this application note.

3.0 "OFF-LINE" FLASH MEMORY UPDATES

Reviewing the Q-and-A discussion earlier, you should be reading this section if you can ensure that the system will receive no interrupts that will require flash memory array access during the update process. Examples of this scenario are numerous:

- Cellular phones that are placed in a special "maintenance" mode for updates.
- PC BIOS applications where the user runs a dedicated "update" routine to upgrade the resident flash memory code.
- Laser printers that can be taken "off-line" prior to the update process.
- Many other applications. . . .

Again referencing Figure 2, we see that the shaded areas of the algorithm can be ignored since flash memory array access is not needed until after the update is complete. The resultant algorithm, shown in Figure 4, is small in size and straightforward in implementation. It can be stored within one of the flash memory blocks if desired, and is copied to/executed from an external memory. Scenarios that follow show two of the many possible implementation options.

Technique 3.1: Algorithm Execution from RAM

The RAM in this technique can be located in several different places within the system, such as:

- In a discrete SRAM or DRAM chip
- Integrated within an embedded microprocessor or microcontroller
- Integrated within a system ASIC
- In a Page Buffer of a separate 16-Mbit FlashFile memory

An important requirement is that the system be able to execute code (not just read and write data) out of the RAM. Ideally, to minimize system overhead and maximize effective update throughput, the update algorithm should be present in RAM at all times during system operation. If this is not possible due to "RAM crunch," the up-front time required to upload the algorithm to RAM must be factored into system update performance calculations.

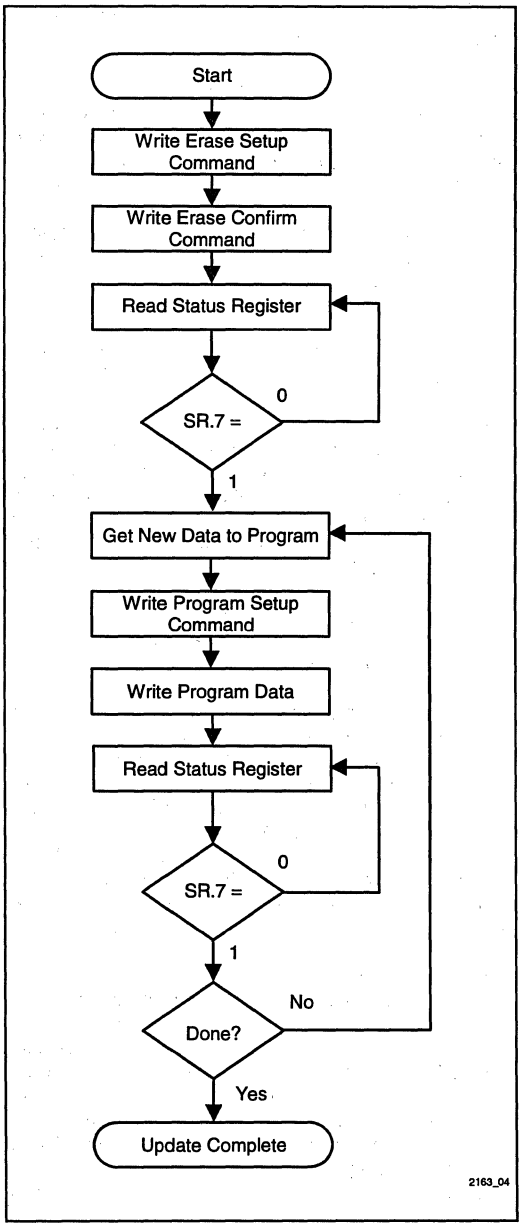


Figure 4. The Block Erase/Program Algorithm Is Simplified for "Offline" Updates

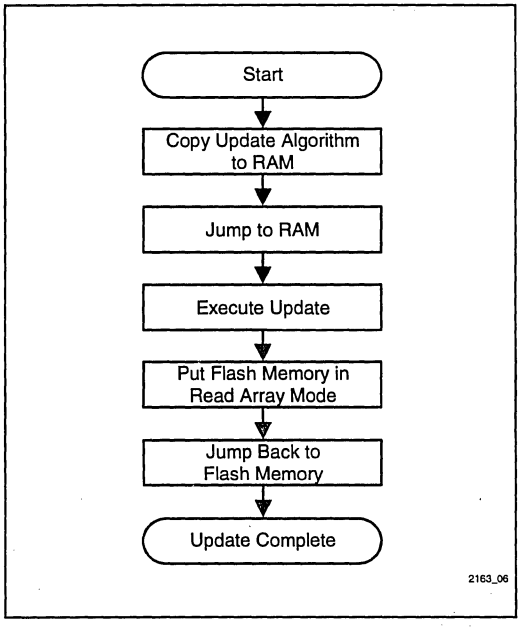


Figure 5. Executing the Update Algorithm Requires Minimal System RAM

Figure 5 shows the overall flowchart used when executing the update algorithm out of system RAM. As mentioned earlier, flash memory automation means that the amount of code executed off-chip to actually program/erase the flash memory is very small. Overhead needed to obtain new code/data from the system varies with the method chosen (diskette, modem, serial or parallel port, etc.).

Does your system include at least one 16-/32-Mbit FlashFile memory and other flash memories? If so, you can potentially use the 256 byte page buffer of the FlashFile memory as the execution RAM while updating the other flash memories! Note: it is NOT possible to completely execute an update algorithm from the page buffer of a flash memory while simultaneously updating that same memory.

Technique 3.2: Algorithm Execution from Nonvolatile Memory

If the system contains multiple flash memories, implementation is very straightforward. Store a duplicate copy of the update code in each flash memory, and execute from one device while updating the other(s). Figure 6 gives one example, using two Intel 28F001BX Boot Block flash memories.

This same technique can be applied to any other nonvolatile memory in the system. Examples include boot ROM, ROM locations within an ASIC or nonvolatile memory integrated within an embedded microprocessor or microcontroller.

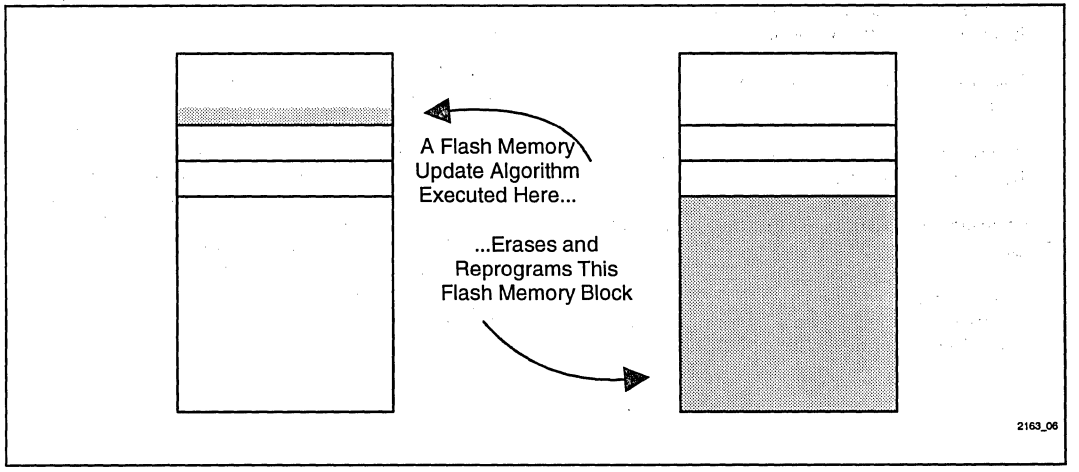


Figure 6. Executing the Flash Memory Update Algorithm from Another Nonvolatile Memory Requires No Dedicated RAM

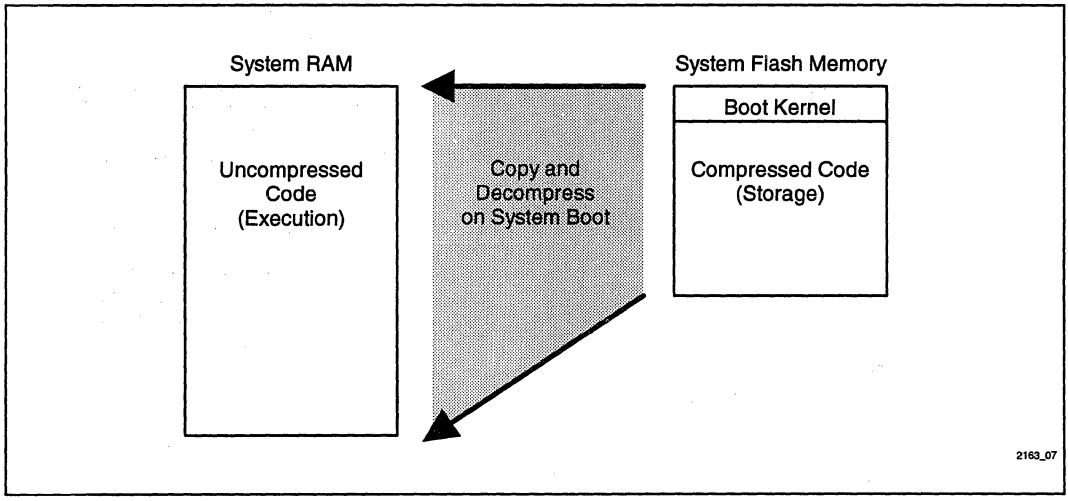


Figure 7. Redundant System RAM Enables Access to All Code during Flash Memory Update

4.0 "ON-LINE" FLASH MEMORY UPDATES

Reviewing the Q-and-A discussion earlier, you should be reading this section if the system must be partially or fully operational during the flash memory update process. Said another way, it must be able to detect and service some or all possible system interrupts. Examples of this scenario include:

- Cellular base stations that must be able to service incoming connection requests.
- Data Communications router and hub networks that cannot be taken off-line.
- Telecommunications PBX switch networks that must be always-operational.

Technique 4.1: Code Redundancy in System RAM

This system memory configuration, shown in Figure 7, is relatively common today in high-performance systems. The system boots from flash memory, copies code to code DRAM (sometimes decompressing in the process) and jumps to DRAM for execution. DRAM is used here primarily because of its high-performance reads.

In this case, the system has access to all interrupt service routines during the flash memory update process. After update is complete, a quick system "reset" will reboot the system and load DRAM with the new code. The amount of time that the system cannot service interrupts is the combination of system reboot and copy-to-DRAM delays.

Technique 4.2: Code Redundancy in System Flash Memory

Figure 8 gives an example of this system memory configuration. Two banks of flash memory components store "previous" and "latest" versions of system code. The system executes from one bank while updating the other bank. Once update successfully completes, an address or control signal "toggle" swaps the "previous" and "latest" banks and enables immediate execution of the latest software version.

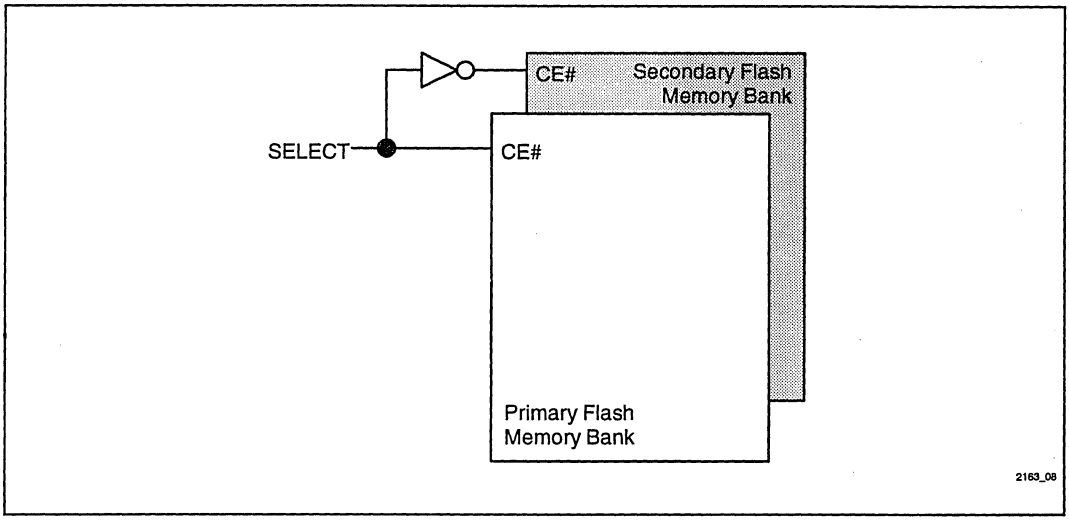


Figure 8. Dual Flash Memory Banks Eliminate RAM Reload Delays

The obvious advantage of this approach include constant access to all interrupt service routines and a non-existent reboot delay. Memory redundancy will incur additional system cost, which must be balanced against advantages and compared to total system cost (and price) to determine applicability of this approach.

Technique 4.3: Leveraging Flash Memory Automation: Programming Performance and Erase Suspend Latency

This approach eliminates both the redundancy of multiple memories and the reboot delay of the flash/DRAM solution in Technique 4.1. It is especially attractive for use with Intel's Embedded Flash RAM memories, whose read performance approaches or exceeds that of DRAM. In this case, the need for redundant code DRAM (for performance reasons) is eliminated.

Before continuing your reading of this section, please do the following research:

- Analyze the latencies of each of your system interrupt routines. Which routines take the longest to execute, and how long do they take?
- Analyze the profile of frequency of interrupts. How often do interrupts occur, and how does this frequency vary with time of day, week, month and year? Can updates be scheduled for times when the interrupt frequency is low (or ideally, zero)?

The flash memory automation approach "hides" byte/word programming operations within the time delay between interrupts. It also "hides" slow block erase by using erase suspend/resume to read from the flash memory when required. Referring back to Figure 2, we see that reads from the flash memory (to access interrupt service routines) can occur at the conclusion of programming, at the conclusion of erase and while erase is suspended. This approach exploits these access "windows."

As an example, we'll construct the following scenario (reference Figure 9).

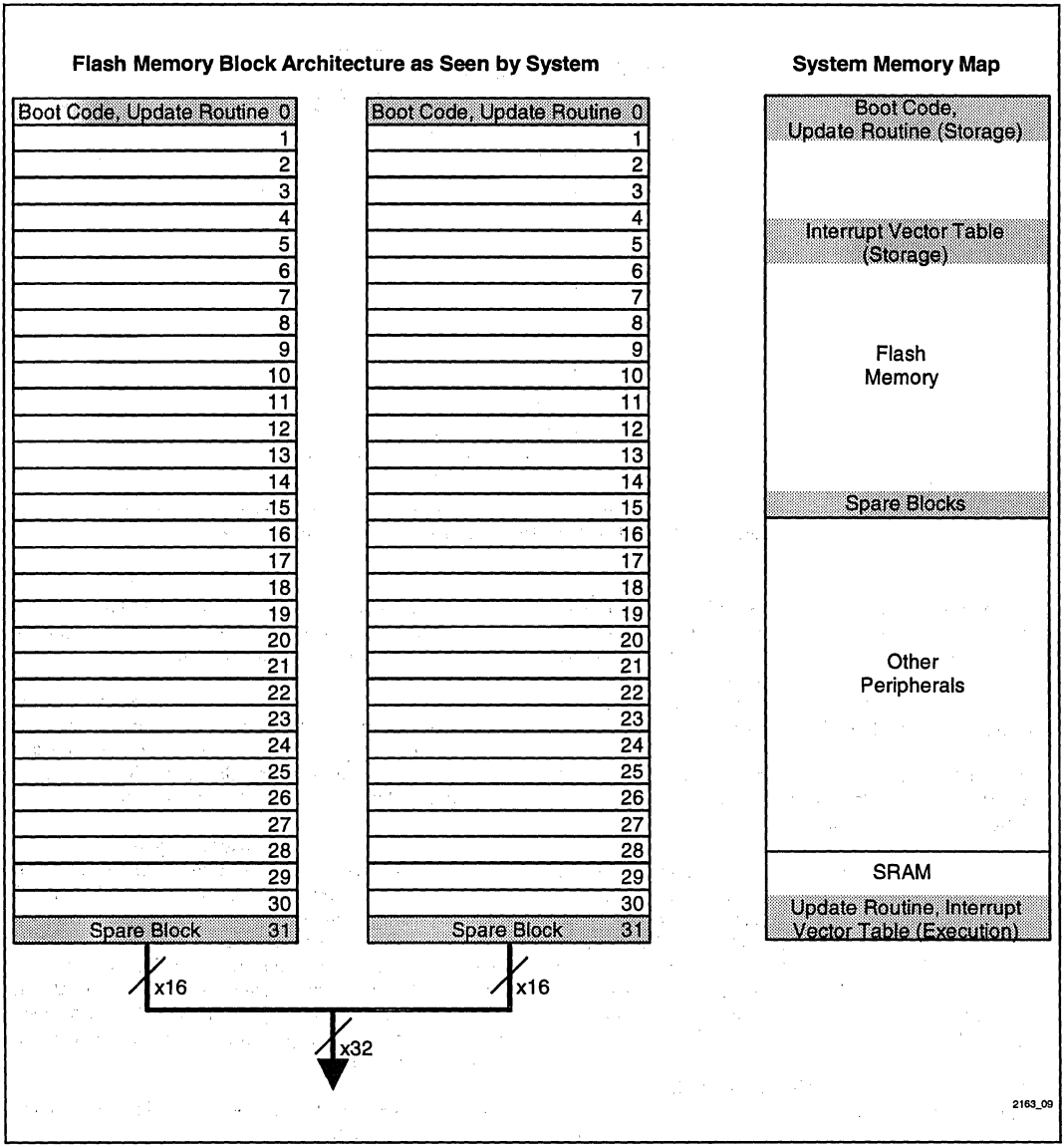


Figure 9. Leveraging Flash Memory Automation Eliminates System Memory Redundancy, Enables Full Interrupt Servicing throughout the Update Process

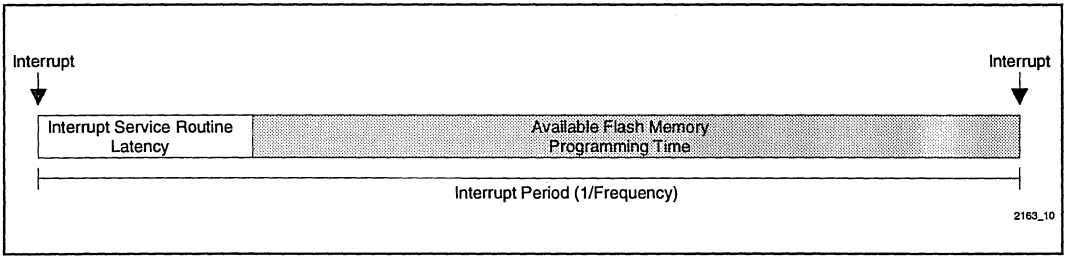


Figure 10. Available Time between System Interrupts Enable Flash Memory Programming

Components

Two 28F016SV flash memories (5V V_{CC}, 12V V_{PP}), each x16, interfacing to a x32 system bus

Small system SRAM

Timings

System interrupt frequency (period) = every 200 µs.

Longest interrupt service routine latency = 50 µs.

Flash memory per-location programming time = 6 µs (typical)

Flash memory erase suspend latency = 10 µs (typical)

Interrupts During Programming

Looking first at programming (Figure 10), we see that the goal is to execute at least one programming operation within the period between interrupts. In the scenario described above, subtracting interrupt service routine latency from interrupt period gives a 150 µs "window" in which programming can occur. At 6 µs per double-word, up to 25 locations can be programmed within each interrupt period.

200 µs (interrupt period) – 50 µs (ISR latency) = 150 µs (programming "window")

150 µs (window)/6 µs (programming time per location) = 25 locations

Intel's 16-/32-Mbit FlashFile memories contain on-chip page buffers, each 256 bytes in size, that dramatically increase effective per-byte programming performance. For example (averaged over a page), typical programming performance for the Intel 28F016SV is 2.1 µs/byte at 5V V_{CC} and 12V V_{PP}. Using these page buffers may, in some cases, allow the system to program even more bytes within each interrupt programming "window."

Interrupts During Erase

Now for erase. If an interrupt occurs during erase, the system must be able to suspend erase, read the flash memory array and service the interrupt, all before the next interrupt. Looking at Figure 11, adding erase suspend latency to interrupt service routine latency and subtracting from interrupt period shows that 140 µs of flash memory erase automation can execute between each interrupt. Obviously, block erase time will extend beyond that specified in the device datasheet since erase is being repeatedly suspended.

200 µs (interrupt period) – [10 µs (erase suspend latency) + 50 µs (ISR latency)] = 140 µs (erase "window")

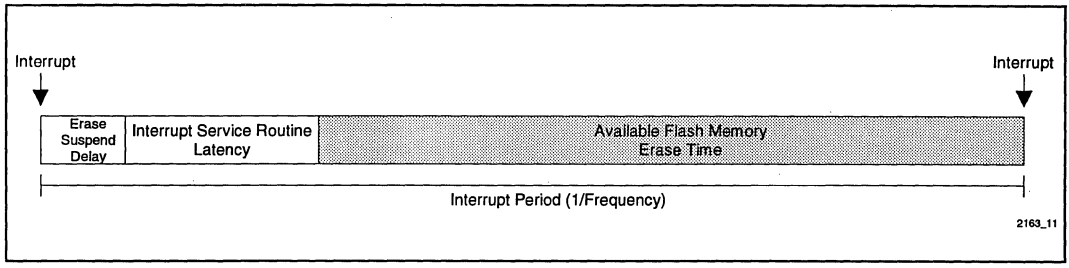


Figure 11. Available Time between System Interrupts Enables Flash Memory Erase, and Erase Suspend Allows Array Access for Interrupt Service Routines

Accessing the Existing Version of Code in a Block Being Updated

All well and good. We've shown how to access code in other flash memory blocks (for example blocks 2–30) while erasing or reprogramming another block (for example, block 1). But what happens if the code you need to read is the code in the process of being updated? Where do you put the previous version of this code?

One approach, shown in Figure 9, assumes that at least one spare block is available in each flash memory (for example, block 31). Before updating any block, copy that block's contents to the spare block and redirect appropriate interrupt vectors to point to that block. After update is complete, redirect interrupt vectors back to the original block, erase the spare block and move to the next block to be updated. This approach will obviously "cycle" block 31 more than any of the others, but this is often acceptable if the number of expected code updates through system lifetime is not excessive.

If spare blocks are not available or expected updates are numerous, copy block information to RAM before updating. This approach requires dedicated RAM for this function but needs much less RAM than a technique like Technique 4.1, where the entire flash memory array is shadowed.

Putting It All Together

Referring back to our example scenario in Figure 9, we conclude with the following summary.

Component block 0 is locked and stores system boot code and the flash memory update routine. The interrupt vector table, stored in an unlocked block to enable its revision, is copied from flash memory to RAM on system power-up. During flash memory update, interrupt vector table contents point to the flash memory update routine, also copied to RAM. When an interrupt occurs, this routine determines via a bit "flag" if block erase is in progress and if so, suspends erase before jumping to the necessary interrupt service code. After servicing the interrupt, the update routine resumes erase or executes location programming operations, depending on where in the update the interrupt occurred.

Ideally, to minimize system overhead and maximize effective update throughput, the update algorithm should be present in RAM at all times during system operation. If this is not possible due to "RAM crunch," the up-front time required to upload the algorithm to RAM must be factored into system update performance calculations.

Before erasing and reprogramming a flash memory block, system software copies block contents to the spare block and appropriately redirects the interrupt vector table. After block erase/reprogram completes, the update routine redirects interrupts back to the block, erases the spare block and moves to the next block to be updated.

Program/Erase Suspend Performance, Typical/Max vs. Cycling

Depending on how "tight" the timings are using the equations of Technique 4.3 with your specifications, and depending on the expected flash memory update frequency (cycling) through system lifetime, additional information may be needed to determine whether this technique is applicable to your design. In this case, please contact your local Intel or distribution sales office for additional information on typical/max program, erase and erase suspend specifications as a function of cycling for the Intel flash memory of interest.

What If Interrupt Period Is too Short or Interrupt Latency Is too Long?

Technique 4.3 assumes that system interrupt timings allowed sufficient time for erase suspend and byte/word programming. If at first inspection this does not seem to be the case for your design, answer the following

questions in the process of further analyzing your system interrupt profile:

- Do interrupts occur fairly regularly as a function of time, or in bursts of activity followed by periods of "quiet?" If the latter, your system software can hold off attempting location programming or resuming erase until it detects a specified time span of system "inactivity."

- Do one or several interrupt service routines have substantially longer latencies than others? If so, system software can hold off attempting programming or initiating/resuming erase when these specific interrupts occur.

In some cases, it may be difficult to hold off programming due to a fixed data write transfer rate to the flash memory subsystem. In these cases, a small RAM FIFO can potentially be integrated within the interface logic (ASIC, FPGA, etc.). This FIFO acts as a buffer between system and flash memory and accommodates programming delays due to interrupt bursts or long ISR latencies.

As an alternative, the approaches described in Techniques 4.1 and 4.2 can be reviewed to determine applicability with your system design criteria.

Programming (Writing) during Erase

Some system designs require both the ability to quickly read code from flash memory and to quickly write information to flash memory in response to an interrupt. Intel's 16-Mbit FlashFile memories offer enhanced on-chip automation that, among its features, automatically suspends block erase to service queued programming operations to other blocks.

5.0 CONCLUSION

Intel has developed a wide range of documentation and other collateral to assist you in developing system software solutions and profiling cycling through system lifetime. Please contact your local Intel or Distribution Sales Office for more information on Intel's flash memory products.

6.0 ADDITIONAL INFORMATION

Documentation

Device datasheets provide in-depth information on device operating modes and specifications.

The 16-Mbit Flash Product Family User's Manual (order #297372) gives detailed information on the enhanced automation of Intel's 16-/32-Mbit FlashFile and Embedded Flash RAM memories. Included flowcharts assist you in developing system software.

The following application notes deal specifically with software interfacing to Intel flash memories:

Order Number	Document
292046	AP-316 "Using Flash Memory for In-System Reprogrammable Nonvolatile Storage"
292059	AP-325 "Guide to First Generation Flash Memory Reprogramming"
292077	AP-341 "Designing an Updateable BIOS Using Flash Memory"
292095	AP-360 "28F008SA Software Drivers"
292099	AP-364 "28F008SA Automation and Algorithms"
292148	AP-604 "Using Intel's Boot Block Flash Memory Parameter Blocks to Replace EEPROM"
292126	AP-377 "16-Mbit Flash Product Family Software Drivers"

NOTES:

Please call the Intel Literature Center at 1-800-548-4725 to request Intel documentation. International customers should contact their local Intel or distribution sales office.

Additional information can be requested from Intel's automated FaxBACK* system at 1-800-628-2283 or 916-356-3105 (+44(0)793-496646 in Europe).

FLASHBuilder

This Windows-based utility is a hypertext aid to understanding the automation of Intel's 16-Mbit FlashFile and Embedded Flash RAM memories. FLASHBuilder automatically generates code segments in C or ASM-86 for flash memory program/erase that you can easily "paste" into your system software. It also includes a cycling utility and power/performance benchmark utilities for the 28F016XS and 28F016XD.

FLASHBuilder is available from the Intel Literature Center via order number #297508. It can also be downloaded from the Intel BBS at 916-356-3600 (+44(0)793-49-6340 in Europe).

VHDL and Verilog Models

VHDL functional simulation models for the 28F016SV, 28F016XD and 28F016XS are available now; please contact your local Intel or distribution sales office. Verilog models for these devices will be available in early 1995.

PROM Programming Support

Intel works closely with a large number of world-wide PROM programmer vendors to ensure timely support for its flash memory products. This programming support information, updated frequently, is available on FaxBACK.

On-Board Programming

An application note will be available in early 1995 that discusses hardware and software recommendations for OBP using either a board tester or PROM programmer. Contact your local Intel or distribution sales office for more details.

intel®

AP-609

APPLICATION NOTE

Interfacing the Intel386™ EX Embedded Processor to Intel Flash

TONY SHABERMAN
TECHNICAL MARKETING ENGINEER

DR. MAHESH RAO
APPLICATIONS ENGINEER

January 1995

Order Number: 292160-001

1.0 INTRODUCTION

The Intel386™ microprocessor family has gained a wide acceptance in the world of embedded applications. The Intel386 EX embedded processor is a very highly-integrated member of the Intel386 microprocessor family. There is a vast base of embedded applications developed for the 80C186 product family. When these applications require higher performance and address space, the Intel386 EX architecture provides a natural migration path to protect the code investment in the Intel architecture along with DOS compatibility. A DOS-based PC provides an easy, cost-effective means to develop, test, debug and port embedded application code.

As embedded system designers take advantage of DOS capability in the PC platform, a revolutionary system architecture is required to meet space and power requirements.

- An architecture that is not bound by what has been done before with existing memory architecture, but free to meet the demanding requirements of embedded end-users.
- An architecture free to adopt and accommodate new technological advances in software and hardware, while protecting end-users initial base hardware investment.

Implementing this new system architecture requires an alternative to the traditional PC storage media such as ROM, DRAM, floppy disk and hard disk. The solution is Intel's Boot Block and FlashFile™ memory (see architecture comparison in Figure 1).

APPLICATION	DATA MANIPULATION	CODE EXECUTION	FILE & CODE STORAGE
Desktops	DRAM	DRAM/ROM	FDD/HDD
Embedded	DRAM	FLASH	FLASH - Resident Disk - Flash Card - Flash Drive

Figure 1. Architecture Comparison

Intel Flash memory provides in-system write capabilities, along with selective block erase and program/erase automation which are gaining wide acceptance in the embedded market. These features help cost-effective field updates and provide quick time-to-market solutions in most applications.

By combining flash memory with this new system architecture, completely new types of computers are now possible that fit in the palm of your hand and replace or integrate many of the code or storage functions of other memory types. Flash memory can be used for storing eXecute-In-Place (XIP) code, such as ROMed DOS, in the system's memory map while additionally functioning like a disk for file and program storage. Since this type of design features flash memory resident on the embedded system's motherboard and is typically arranged in an array, it is described as a Resident Flash Array (or RFA). To further differentiate the two tasks of an RFA, the file store task is called a Resident Flash Disk (RFD), while the XIP task is called Resident Flash for XIP (or RFX) code storage.

1.1 Why a New Memory Architecture?

The ideal embedded memory system is:

- Power Conscious (prolongs battery life and reduces heat)
- Dense (stores lots of code and data in a small amount of space but weighs very little)
- Updateable (allows in-situ code enhancements)
- Fast (lets you read and write data quickly)
- Inexpensive (low cost-per-megabyte)
- Reliable (retains data when exposed to extreme temperature and mechanical shock)

While embedding the PC architecture, designers have grappled with how to construct memory systems that meet the above criteria. Embedded computing makes the system design even tougher with more stringent requirements for low power, low volume, less weight and harsh environments. The best combination available for embedded PC designs in their infancy was the same as used for the desktop; solid-state memory and magnetic storage, i.e., SRAMs, DRAMs plus magnetic hard disks. DRAMs are dense and inexpensive, yet slower than the processors they serve, and they are volatile. SRAMs, although fast enough to keep pace with processors, are relegated to caching schemes (compensating for DRAM's slowness) due to low density and high cost while also being volatile.

Magnetic hard disks are dense, inexpensive on a cost-per-megabyte basis and nonvolatile. However, they are also slow, power-hungry, susceptible to damage from physical shock, and take up a sizable amount of volume.

Embedded computing designs cannot depend on hard drives as do desktop or portable PCs, due to the size limitations. Furthermore, vitally important data such as credit card numbers or transactions, signatures, or patient monitoring information demands reliability of the highest order. The solution is Intel Flash memory.

1.2 The Flash Memory Alternative

High Density

Intel's ETOX™ IV flash memory cell is 30% smaller than equivalent DRAM cells; therefore, it will closely track DRAM density. Flash memory is more scaleable than DRAM because the flash storage cell is not sensitive to soft error failure; therefore, it can have a more simple cell structure. As density increases and process lithography continues to shrink, flash memory will pace, and ultimately overtake, the DRAM technology treadmill.

Updateable

ROMs and EPROMs may offer lower device costs, but if servicing the customer or end-user is important to an OEM, overall system cost must be factored in. Although ROMs and EPROMs are nonvolatile, changing the code within them is either very difficult (in the case of EPROMs), or entirely impossible (in the case of ROMs). Whole inventories of ROMs could be lost in the event of a catastrophic bug, while an innovative design with flash memory can be updated in the factory or by end-users via networks, OEM Bulletin Board Systems, or other memory cards. Updating systems could actually become a second source of income for OEMs and Independent Software Vendors (ISVs), enhancing the quality of the product while increasing end-user satisfaction.

Power Conscious

Intel's flash memory provides a deep power-down mode, reducing power consumption to typically less than 0.2 µA. Typical read current is only 20 mA, while typical standby current (flash memory not being accessed with CE# high) is only 30 µA. Additionally, flash devices operating at 3.3V are available for state-of-the-art low-power consumption designs.

Fast

Do not be misled by technology-to-technology speed comparisons. Architecting a system around flash memory bypasses the code/data bottleneck created by connecting slow mechanical serial memory (such as disks) to a high-performance, parallel bus system. For example, data seek time for a 1.8" magnetic hard disk is 20 ms, plus an 8 ms average rotational delay, while flash memory write time is less than 0.1 ms. At the chip level, read speeds for flash memory are about 70 ns. Therefore, either direct execution of code from flash memory or downloading to system RAM will dramatically enhance overall system performance.

Nonvolatile

Unlike DRAM or SRAM, flash memory requires no battery back-up. Further, Intel's flash devices retain data well beyond the useful lifetime of most applications.

Rugged and Reliable

On average, today's hard-disk drives can withstand up to 10 Gs of operating shock. Intel's flash memory can withstand as much as 1000 Gs. Flash components can operate beyond 70°C while magnetic drives are limited to 55°C. Intel's flash memory can be cycled 100,000 times per block or segment. By employing wear-leveling techniques, the cycling of a device can be minimized. For example, a 10-KB file written every 5 minutes, 24-hours a day to a 20-MB flash array takes 16 million hours, or 1826 years, before reaching the 100,000 cycle level.

1.3 Summary

Many applications benefit from ROMed or XIP versions of code, particularly hand-held personal computers, vertical application pen-based clipboards, and industrial control and data accumulation equipment. These applications pose system design constraints requiring small form factor, low-power consumption, and rugged construction due to active mobile users or harsh environments. Exposure to shock, vibration, or temperature extremes is common, precluding the use of rotating media. Flash memory provides an excellent code storage choice for such system designs featuring thin TSOP packaging, low (deep power-down mode) or zero (capability to shut off power without losing data) power consumption, 1000 G shock resistance and extended temperature products. Additionally, flash memory provides remote or end-user update capability

allowing OEMs to service their products more efficiently and add new software features and applications after the sale.

Compared to RAMs and ROMs, the timing requirements for flash are slightly different. This application note explores those differences and provides a detailed analysis of the interface between the 28F400BX and the Intel386 EX CPU. Along with the analysis, one possible solution is provided. To make all of the read and write calculations easier for the users, a spreadsheet-based timing analysis for the Intel386 EX CPU interface to flash is posted on Intel's application support BBS at 916-356-3600. The filename is EXFLASH.ZIP and it is located in the E3X\REF_DSGN section. A snapshot of this spreadsheet is provided in Appendix C.

2.0 FLASH TIMING PARAMETERS

The timing parameters provided in Tables 1 and 2 lists the most important parameters to pay attention to when interfacing to Intel's Boot Block and FlashFile memory families.

Table 1. Read Timing Parameters

Timing Description	28F001BX	28F200BV(2)	28F400BV(3)	28F008SA(6)	28F016SA	28F016SV
Address Valid to Data Valid, t_{ACC}/t_{AVQV} (max)	120 ns	60 ns	60 ns	85 ns	70 ns	65 ns
CE# Valid to Data Valid, t_{DF}/t_{ELQV} (max)	120 ns	60 ns	60 ns	85 ns	70 ns	65 ns
OE# Valid to Output Delay, t_{OE}/t_{GLQV} (max)	50 ns	30 ns	30 ns	40 ns	30 ns	30 ns
OE# High to Data Float, t_{DF}/t_{GHQZ} (max)	30 ns	20 ns	20 ns	30 ns	25 ns	25 ns

Table 2. Write Timing Parameters

Timing Description	28F001BX	28F200BV(2)	28F400BV(3)	28F008SA(6)	28F016SA	28F016SV
Address Valid to WE# High, t_{AS}/t_{AVWH} (min)	50 ns	50 ns	50 ns	40 ns	50 ns	40 ns
Data Valid to WE# High, t_{DS}/t_{DVWH} (min)	50 ns	50 ns	50 ns	40 ns	50 ns	40 ns
WE# Pulse Width, t_{WP}/t_{WLWH} (min)	50 ns	50 ns	50 ns	40 ns	50 ns	45 ns
Address Hold from WE# High, t_{AH}/t_{WHAX} (min)	10 ns	10 ns	10 ns	5 ns	10 ns	10 ns
WE# Pulse Width High, t_{WPH}/t_{WHWL} (min)	50 ns	10 ns	10 ns	30 ns	30 ns	15 ns

NOTES:

1. The read and write timings provided in Tables 1 and 2 were taken from the respective component's datasheet and assume a commercial temperature range, 30 pF test load, and V_{CC} 5V ± 5% for the fastest part available. The 28F001BX is the only exception with V_{CC} ± 10% and a 100 pF test load.
2. The timings listed in Tables 2 and 3 for the 28F200BV are the same for the 28F002BX, 28F200BX, and 28F002BV.
3. The timings listed in Tables 2 and 3 for the 28F400BV are the same for the 28F004BX, 28F400BX, and 28F004BV.
4. The write timing parameters assume WE#-controlled writes.
5. As can be seen from the preceding tables, the majority of the flash timing parameters are close enough to each other that the interface to the Intel386 EX processor will not change much.
6. In addition to the 28F008SA, Intel also makes a 28F008BV and 28F800BV. These parts are 8-Mbit members of Intel's Boot Block Flash memory family and incorporate SmartVoltage technology. However, at the time of publication for this document, A/C timing information was unavailable.

3.0 READ TIMING ANALYSIS

The remaining sections of this application note analyze the interface between a 25 MHz Intel386 EX CPU and a 60 ns 28F400BX boot block flash. The procedure used to analyze this interface should be used when interfacing the Intel386 EX CPU to any Intel flash device. When comparing the Intel386 EX CPU parameters to flash, this application note will refer to the flash signals names CE#, OE# and WE#. However, the Intel386 EX CPU respective signals are called CS#, RD# and WR#. Note that the Intel386 EX CPU also has a W/R# signal that can be used to distinguish between read and write cycles. CLK2 refers to the 50 MHz clock that drives the Intel386 EX CPU. This clock is internally divided by two to make a 25 MHz internal clock for the internal peripherals.

Table 3 provides the memory requirements for a zero wait-state read cycle. Unfortunately, the memory requirements for a zero wait-state design are too difficult for most memory devices to meet. By adding one wait-state a 60 ns flash device can successfully interface to the Intel386 EX CPU. Table 4 compares the 28F400BX parameters to the Intel386 EX CPU parameters in a one wait-state design.

The timings for the 28F400BX listed in Tables 4 and 6 are the same for the 28F400BV. This application note analyzes the interface to the 28F400BX in order to address issues with current designs. However, Intel recommends that all new designs use the 28F400BV. The 28F400BV has SmartVoltage technology enabling read capability at 3.3V or 5V, and program/erase capability at 5V or 12V.

Table 3. Read Parameter Time Comparison at Zero Wait-States

Timing Description	i386 EX CPU Paramete(4)	Memory Parameter
Address Valid to Data Valid	t_{47} = 4CLK2-36 = 44 ns max	Must Be Less Than 44 ns
CE# Valid to Data Valid	t_{47a} = 4CLK2-46 = 34 ns max	Must Be Less Than 34 ns
OE# Valid to Data Valid	t_{48} = 3CLK2-36 = 24 ns max	Must Be Less Than 24 ns
OE# High to Data Float	t_{50} = 10 ns max	Must Be Less Than 10 ns

Table 4. Read Parameter Time Comparison at One Wait-State

i386 EX CPU Parameter	28F400BX Parameter(5)	Violated Timing?
t_{47} = 44+2CLK2 = 84 max (2)	t_{ACC} = 60 ns Max (1)	84 > 60, No
t_{47a} = 34+2CLK2 = 74 max (2)	t_{CE} = 60 ns Max (1)	74 > 60, No
t_{48} = 24+2CLK2 = 64 max (2)	t_{OE} = 30 ns Max (1)	64 > 30, No
t_{50} = 10 ns max	t_{DF} = 20 ns Max	10 < 20, Yes(3)

NOTES:

1. These parameters assume the flash data lines are connected directly to the Intel386 EX CPU. They do not include any additional delays due to a buffer.
2. For every wait-state it is necessary to add 2CLK2 to each parameter dependent on CLK2. 2CLK2 is added to the first three parameters for a one wait-state system.
3. Adding additional wait-states will not fix the t_{DF} timing violated.
4. All Intel386 EX CPU timings were taken from the fourth revision datasheet (order #272420-004).
5. All 28F400BX timings were taken from the third revision datasheet (order #290451-003).

3.1 Read Timing Solution

3.1.1 OE# HIGH TO DATA FLOAT

Table 4 indicates that OE# high to data float is violated. After OE# goes inactive, the 28F400BX takes a maximum of 20 ns for its data lines to go to a high impedance state. This becomes a problem when the Intel386 EX CPU tries to do a write immediately after reading from the 28F400BX. This problem is solved by either using a buffer to control the bus contention, or by making sure your code has a NOP in between every read from flash followed by a write. In this example, a buffer will be used to address this issue. Section 4.1 details the specific implementation of the buffer. Figure 9 shows a complete timing diagram for a read followed by a write.

3.1.2 DATA READ SETUP TIME VERIFICATION

At 25 MHz, the Intel386 EX CPU requires a min of 7 ns of data setup time before the end of the read cycle. The end of a one wait-state cycle at 25 MHz is 120 ns. In order to meet the read setup time, data must be presented to the processor no later than 113 ns into the read cycle. The processor provides a valid chip select a max of 39 ns into the read cycle. The 28F400BX will respond with data 60 ns later. Since a buffer is used, 11 ns propagation delay (t_{PLH}, max) is added to the calculation. This means that data is presented to the processor a max of 110 ns (39+60+11) from the beginning of the read cycle (see Figure 2). Since this is less than 113 ns, the read setup time is met.

3.1.2.1 Capacitive Loading

An access time of 60 ns on the 28F400BX was specified with a 30 pF load (see high speed test configuration in datasheets). Derating curves for the 28F400BX show that at 50 pF the access time only increases by approximately 2 ns. Since our example meets the data read setup time by 3 ns, a one wait-state read cycle at 50 pF is still possible to achieve. If the systems capacitive loading is higher than 50 pF, a faster buffer will be needed in order to achieve a one wait-state system.

3.1.3 DATA READ HOLD TIME VERIFICATION

The Intel386 EX CPU requires a min of 5 ns of data hold time after the end of the read cycle. The 28F400BX will hold its data as long as OE#, CE# and the address are active. In this implementation, a PLD turns off the buffer's EN# line as early as 2 ns (t_{CO1}) after the end of the read cycle. As long as the buffer has a min t_{PLZ} and a min t_{PHZ} of 3 ns or more, the data read hold time will be met (see Figure 2). In general, as long as the PLD's t_{CO1} plus the buffers t_{PLZ} or t_{PHZ} are at least 5 ns, the data read hold time will be met.

3.1.4 READ SUMMARY

A one wait-state read cycle is possible with a 60 ns 28F400BX with loads up to 50 pF by using the given implementation. It is advised to use a buffer in order to prevent bus contention on read followed by write cycles This will fix the flash memory data float delay violation (t_{50}). Adding a buffer has other benefits as well. It makes the load to the flash data lines smaller for faster read performance, and it makes the load to the CPU smaller so it can drive more devices. In addition to flash memory, typical embedded systems will incorporate some SRAM or DRAM. Some RAM is necessary in order to do in-circuit programming of flash. The Intel386 EX CPU has CMOS inputs so the buffer helps the other memory devices drive to these levels. See Figure 7 for a complete read timing diagram, and Figure 9 for a read followed by write timing diagram.

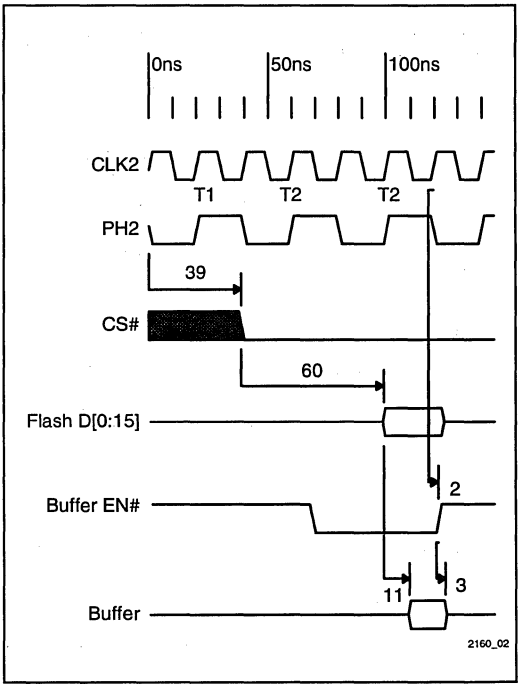
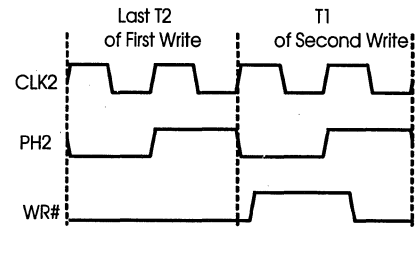


Figure 2. Read Setup and Hold Timing

4.0 WRITE TIMING ANALYSIS

Table 5 provides the memory requirements for a zero wait-state write cycle. Unfortunately, the memory requirements for a zero wait-state design are too difficult for most memory devices to meet. Table 6 compares the 28F400BX parameters to the Intel386 EX parameters in a one wait-state design. Section 4.1.2 provides a detailed explanation for why two wait-states are required for this interface.



NOTE:
WR# will go inactive in phase 1 of the first T-state after the last T2 of the write. WR# will go active in phase 2 of T1. For back-to-back writes WR# will go inactive for one CLK2.

Figure 3. WR# Active/Inactive Timings

Table 5. Write Parameter Time Comparison at Zero Wait-State

Timing Description	I386 EX CPU Parameter	Memory Parameter
Address Valid to WE# High	$t_{46}+t_{41}$ = 3CLK2-15 = 45 ns min	Must Be Less Than 45 ns
Data Valid to WE# High	t_{43} = 3CLK2-27 = 33 ns min	Must Be Less Than 33 ns
WE# Pulse Width	t_{46} = 3CLK2-15 = 45 ns min	Must Be Less Than 45 ns
Address Hold from WE# High	t_{42} = 5 ns min	Must Be Less Than 5 ns
WE# Pulse Width High	Implied 20 ns(1) min	Must Be Less Than 20 ns

NOTE:
1. The implied 20 ns comes from the fact that WR# on the Intel386 EX CPU goes inactive from the previous write in phase 1 of T1, and goes active for the current write in phase 2 of T1. At 25 MHz each phase is 20 ns, therefore the WR# pulse high time would be 20 ns (See Figure 3 above).

Table 6. Write Parameter Time Comparison at One Wait-State

I386 EX CPU Parameter	28F400BX Parameter	Violated Timing?
$t_{46}+t_{41}$ = 45+2CLK2 = 85 ns min (2)	t_{AS} = 50 ns min	85>50, No
t_{43} = 33+2CLK2 = 73 ns min (2)	t_{DS} = 50 ns min (1)	73>50, No
t_{46} = 45+2CLK2 = 85 ns min (2)	t_{WP} = 50 ns min	85>50, No
t_{42} = 5 ns min	t_{AH} = 10 ns min	5<10, Yes (3)
Implied 20 ns min	t_{WPH} = 10 ns min	20>10, No

NOTES:
1. These parameters assume the flash data lines are connected directly to the Intel386 EX CPU. They do not include any additional delays due to a buffer.
2. For every wait-state it is necessary to add 2CLK2 to each parameter dependent on CLK2. 2CLK2 is added to the first three parameters for a one wait-state system.
3. Adding additional wait-states will not fix the t_{AH} timing violated.

4.1 Write Timing Solution

4.1.1 ADDRESS HOLD FROM WE# HIGH

Table 6 indicates that address hold from WE# high is violated. The 28F400BX requires that the processor hold the address 10 ns after WE# goes inactive. The Intel386 EX CPU only guarantees that it will hold the address for 5 ns (t_{42}). This can be fixed by controlling WE# with a PLD. If the PLD pulls WE# high at least 5 ns earlier, then the problem is solved.

4.1.2 DATA VALID TO WE# HIGH AT ONE WAIT-STATE

The section above states that pulling WE# high early with a PLD solves the violated write timing. However, pulling WE# high early shortens t_{DS} (Data Valid to WE# high). If WE# is pulled high too early, t_{DS} will be violated. Figure 4 illustrates the timings for WE#. WE# is pulled high one CLK2 early by a PLD with a minimum propagation delay of 2 ns. With one wait-state, this occurs 102 ns into the write cycle. For the 28F400BX, t_{DS} is 50 ns, therefore data has to be valid no later than 52 ns into the write cycle (102–50). The Intel386 EX CPU places data on the bus as late as 51 ns from the beginning of the write cycle (31 ns from the middle of T1). However, in order to avoid bus contention on a read followed by write cycle, a buffer was added causing additional delays of up to 11 ns in worst case (see Section 3.1.1). These additional delays cause t_{DS} to be violated in a one wait-state write cycle. Since t_{DS} is violated, it becomes necessary to add a second wait-state to the write cycle.

4.1.3 TWO WAIT-STATE WRITE

With a second wait-state, the write cycle extends to 160 ns. WE# still needs to be pulled high one CLK2 early. With two waits states, this occurs 142 ns into the write cycle (see Figure 4). With a t_{DS} of 50 ns, data has to be valid no later than 92 ns into the write cycle (142–50). This leaves plenty of time to meet t_{DS}.

4.1.4 WRITE SUMMARY

By controlling WE# with a PLD as implemented above, a two wait-state write cycle is achieved. For the Intel386 EX CPU, these two wait-states could be easily programmed using the internal chip-select logic or implemented with external ready logic. Ready generation is explained in detail in Section 5.2. See Figure 8 for a complete write timing diagram.

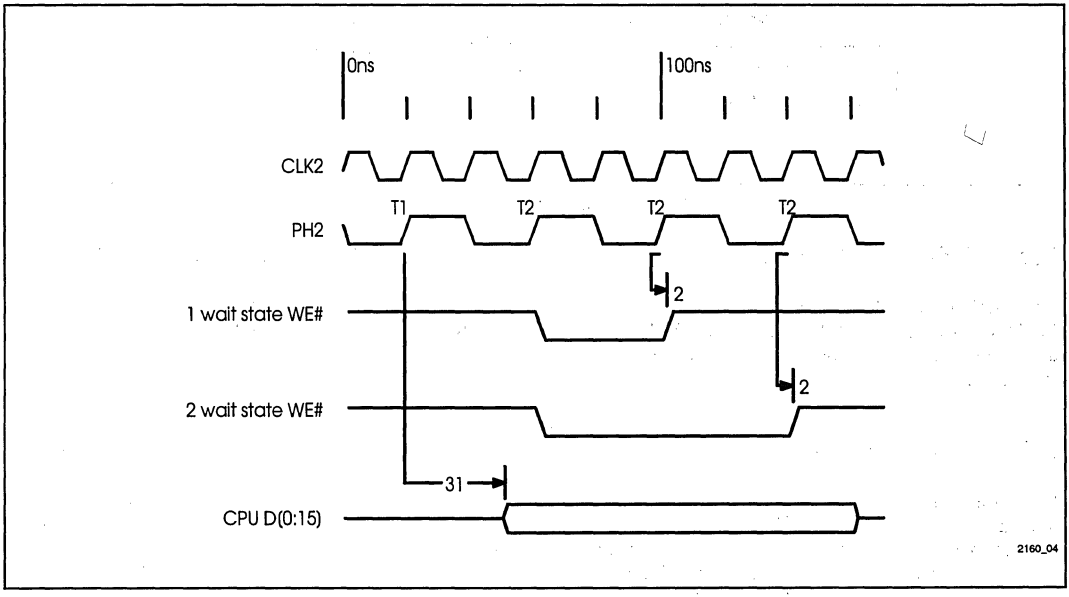


Figure 4. WE# Control Timing

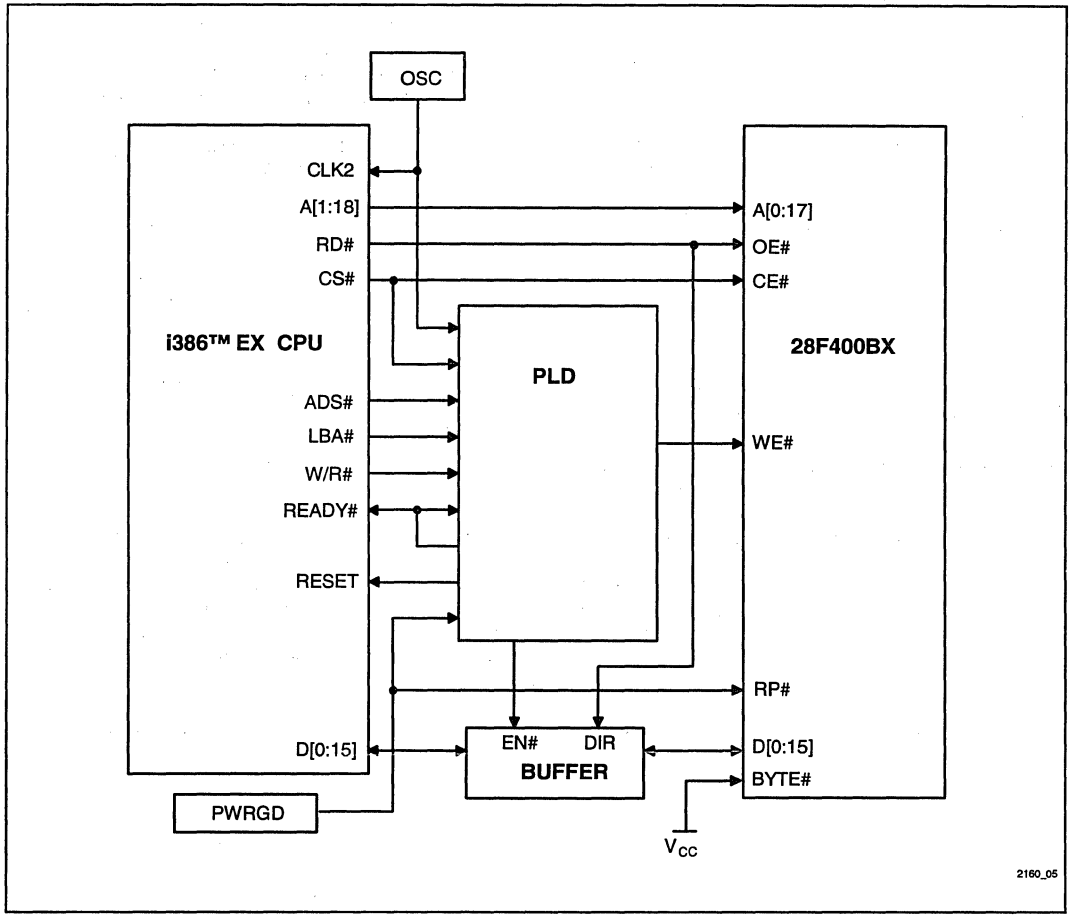


Figure 5. System Block Diagram

5.0 INTERFACE LOGIC

This design example will show how to interface the Intel386 EX CPU to the 28F400BX with a PLD (22V10) and a buffer. Figure 5 shows the basic block diagram of the interface.

5.1 WE# Control

From the Write Timing Analysis in Section 4.0, address hold from WE# high and CE# hold from WE# high are the parameters which are violated. These parameters can be met by controlling WE# with a PLD as indicated in Section 4.1 and Figure 4. WE# is enabled during a write cycle in the middle of the first T2 and is then disabled in the middle of the third T2. A complete timing diagram illustrating the write cycle is found in Figure 8. The PLD equation for WE# is provided in Appendix A.

5.2 Ready Generation

In this example, the read cycle requires one wait-state and the write cycle requires two. This can be achieved by setting the RDY bit in the UCSADL or CSADL register and generating READY# with the PLD. Another approach would be to use the ready logic built into the Intel386 EX CPU chip-select unit. For this approach the chip select unit would need to be programmed to

generate two wait-states for both read and write cycles. However, since the read cycle only requires one wait-state, this is not the optimum solution. For this example, a PLD generates one wait-state for a read cycle and two wait-states for a write cycle. READY# is also brought in as an input and is used to disable the buffer. Timing diagrams illustrating READY# generation for read and write cycles are provided in Figures 7 and 8 respectively. The PLD equations for READY# are provided in Appendix A.

5.3 Reset Conditions

Upon reset, the UCSADL register defaults to FF6FH. This means that the Intel386 EX CPU will insert 15 wait-states and drive READY#. If you are using a PLD to drive READY# (like in this example) it is necessary to tristate the PLD's READY# to avoid bus contention with the CPU's READY#. This can be accomplished by sampling LBA#. The Intel386 EX CPU will assert LBA# on accesses where it is driving READY#. When the PLD samples LBA# active it will tristate its READY#. The PLD equation for tristating READY# is provided in Appendix A.

For the interface implemented in Figure 5, PWRGD (power good) is the signal that indicates to the PLD to RESET the CPU. It is also important to connect PWRGD to RP# (reset/power-down) on the flash device for two reasons. First, it is important for the flash command user interface (CUI) to reset at the same time as the CPU, putting the CUI in read array mode. At power-up the CUI will default to read array mode. However, some designs allow the system to reset while the power remains on (reset button). At reset, the CUI may not be in read array mode for the CPU to boot, therefore it is necessary for the system to reset the CUI at the same time as the CPU. Triggering RP# with PWRGD ensures that this happens. The second reason is to guard against spurious writes. With bus signals in an indeterminate state at power-up, connecting PWRGD to RP# provides memory protection by masking unwanted bus conditions.

5.4 Buffer Control

The 28F400BX data float time is longer than the allowable limit specified in the Intel386 EX CPU datasheets. One solution to this problem is to add a buffer. In this implementation, a PLD controls the enabling of the buffer. For a write, the buffer is enabled in the middle of the first T2 and then is disabled at the end of the third T2 (see Figure 6 below). For a read, the buffer is enabled in the middle of the first T2 and is disabled at the end of the second T2. The direction of the buffer is controlled by the Intel386 EX CPU RD# signal (see Figure 5). A complete timing diagram illustrating the buffers enable and disable during a read followed by write cycle is found in Figure 9. The PLD equation for EN# is provided in Appendix A.

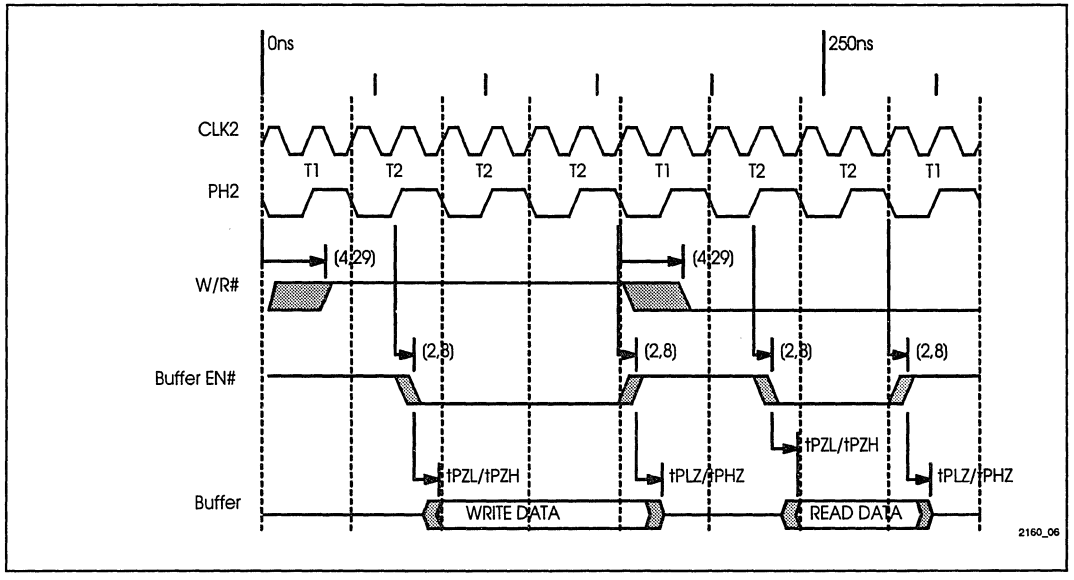
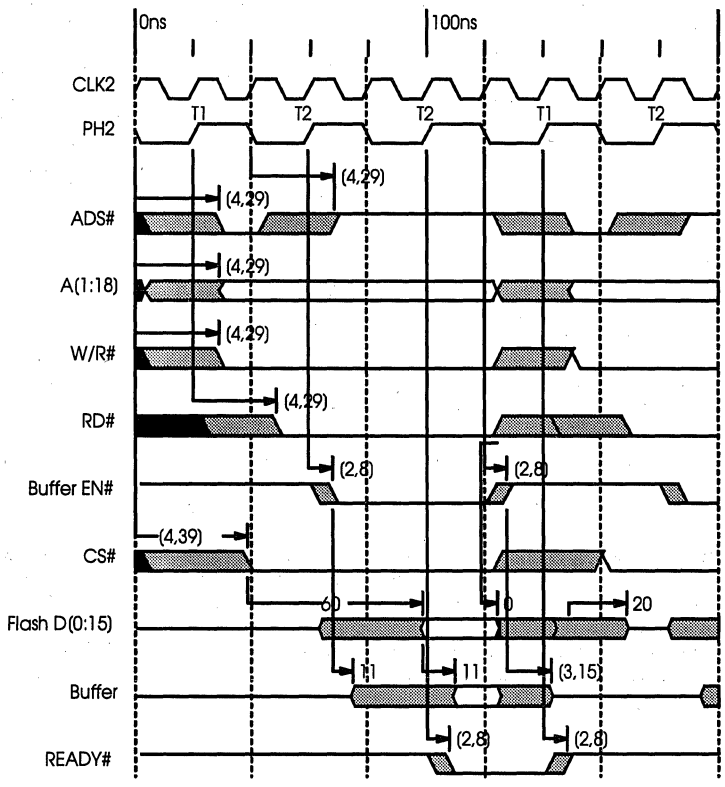


Figure 6. Buffer EN# Control

6.0 READ TIMING DIAGRAM

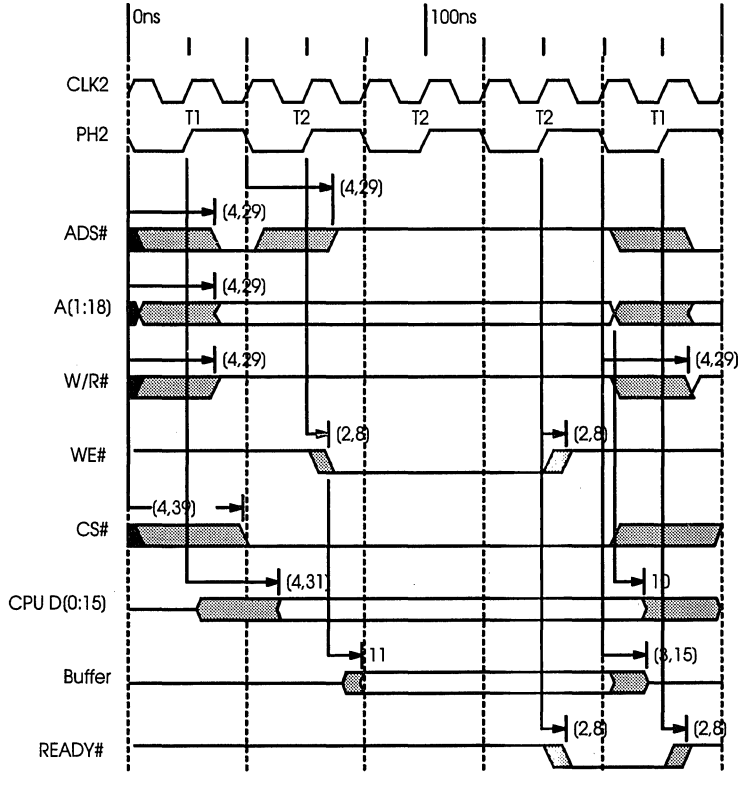


NOTE:

The processor puts the address and W/R# on the bus a max of 29 ns from the beginning of T1. CS# goes active a max of 39 ns from the beginning of T1. The PLD samples W/R# in the middle of the first T2 and enables the buffer. Data from the 28F400BX is guaranteed 60 ns after CS# is active. Since the buffer has a max delay of 11 ns, data will be on the data bus a max of 110 ns from the beginning of T1. This gives 10 ns of setup time before the data is latched. In this example, the PLD could disable the buffer as soon as 2 ns after the data is latched. With 3 ns of t_{PHZ} and t_{PLZ} for the buffer, data will hold for a minimum of 5 ns after it is latched.

Figure 7. Read Timings for the Interface

7.0 WRITE TIMING DIAGRAM

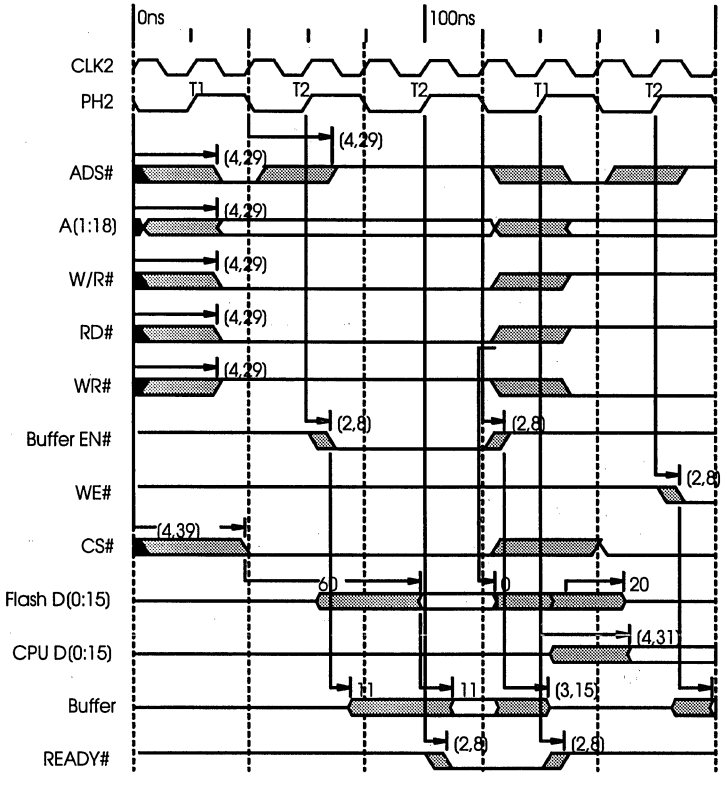


NOTE:

The processor puts the address and W/R# on the bus a max of 29 ns from the beginning of T1. CS# goes active a max of 39 ns from the beginning of T1. The PLD samples W/R# in the middle of the first T2 (60 ns into the write cycle) and enables WE# as well as the buffer. The PLD enables the buffer a max of 8 ns from the middle of the first T2. The buffer has a max t_{PHZ} and t_{PLZ} of 11 ns, therefore data will be presented to the 28F400BX a max of 79 ns into the write cycle (60+8+11). WE# is pulled high as early as 2 ns from the middle of the third T2 (142 ns into the write cycle). This yields a min of 64 ns (142–79) of data setup to WE# high.

Figure 8. Write Timings for the Interface

8.0 READ FOLLOWED BY WRITE TIMING DIAGRAM



NOTE:

At the end of the read cycle the 28F400BX requires a max of 20 ns from RD# high to go to a high impedance state. If the next bus cycle is a write, the processor will drive data on to the bus. This causes the potential for bus contention and can be seen in the figure by the overlap of Flash D[0:15] with CPU D[0:15]. A solution to this problem is to turn off the buffer during this period of overlap. This allows the 28F400BX to float its data lines, and allows the processor to drive the data bus without contention.

Figure 9. Read Followed by Write Timing Cycles

9.0 ADDITIONAL INFORMATION

9.1 References

Order Number	Title
272420	Intel386™ EX Embedded Microprocessor Datasheet
290406	28F001BX 1-Mbit Boot Block Flash Memory Datasheet
290429	28F008SA 8-Mbit FlashFile™ Memory Datasheet
290448	28F200BX, 28F002BX 2-Mbit Boot Block Flash Memory Family Datasheet
290451	28F400BX, 28F004BX 4-Mbit Boot Block Flash Memory Family Datasheet
290531	2-Mbit SmartVoltage Boot Block Flash Memory Family
290530	4-Mbit SmartVoltage Boot Block Flash Memory Family
290539	8-Mbit SmartVoltage Boot Block Flash Memory Family
290489	28F016SA 16-Mbit FlashFile™ Memory Datasheet
290528	28F016SV 16-Mbit FlashFile™ Memory Datasheet
272425	AP-499, "Introducing Intel's Family of Embedded Intel386™ Microprocessors"
292163	AP-610, "Flash Memory In-System Code and Data Update Techniques"

APPENDIX A
PLD EQUATIONS

*These PLD equation are posted on the BBS under EXFLASH.ZIP

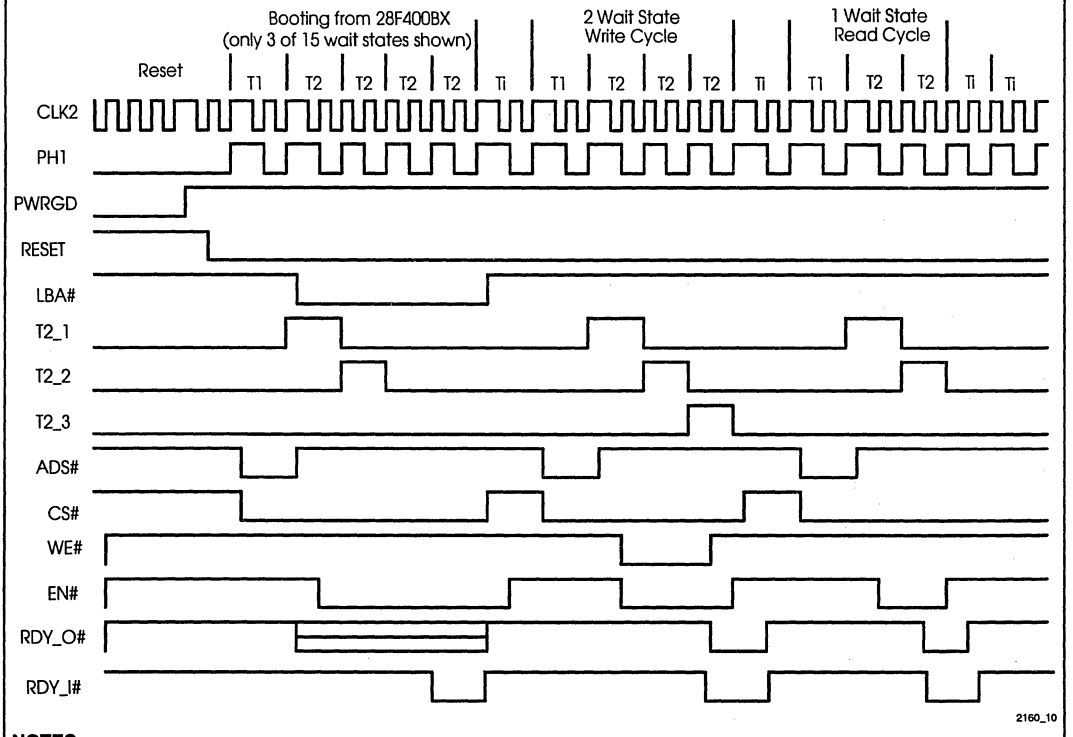
TITLE	MEMORY INTERFACE
PATTERN	PDS
REVISION	2.0
AUTHOR	TONY SHABERMAN
COMPANY	INTEL
DATE	1/20/95

; This design has not been verified, it is sample code only.
; Intel assumes no responsibility for any errors which may appear
; in this code. This PLD performs the functions necessary for interfacing
; the Intel386™ EX CPU to the 28F400BX. It controls WE# to the 28F400BX,
; EN# to the buffer, READY# and RESET to the processor.
;
; NOTES:
;
; T2_1 will go active for the first half of the first T2 on all cycles (1 clk2).
; T2_1 will stay active for the second half of T2 if CS# is active. It is
; necessary to place CS# at the second half of the equation because CS# has a
; 39 ns max valid delay (1 ns before the first half of the equation is sampled).
;
; T2_2 will go active for the duration of the second T2 under the condition
; that the second half of T2_1 was generated
;
; T2_3 will go active for the duration of the third T2 under the conditions
; that the second half of T2_2 was generated and it is a write cycle
;
; EN will go active in the middle of the first T2 if CS# is active. EN is
; disabled either by the CPU or the PLD driving RDY_I#. Since the CPU could
; drive RDY_I# early in the last T2, a third term (EN and PH1) is added to the
; equation to ensure that the buffer stays active during the second half of
; the last T2.
;
; READY# is represented by two pins, RDY_I# (ready in) and RDY_O# (ready out).
; When booting from flash, the CPU will drive RDY_I# so the PLD must tristate
; its RDY_O#. Some PLD devices support a pin feedback feature which would
; eliminate the need for using two pins.

```
CHIP   MEMORY_INTERFACE      iPLD22V10

; INPUTS
PIN 2  CLK2          ; 2X INPUT CLOCK (50 MHZ)
PIN    PWRGD         ; POWER GOOD USED TO GENERATE RESET TO CPU
PIN    W_R           ; FROM CPU W/R# SIGNAL
PIN    /ADS          ; FROM CPU ADS# SIGNAL
PIN    /CS           ; FROM CPU CS# SIGNAL
PIN    /LBA          ; FROM CPU LBA# SIGNAL
PIN    /RDY_I        ; FROM CPU READY# SIGNAL AND PLD RDY_O# SIGNAL

; OUTPUTS
PIN    /WE           ; TO FLASH WE# SIGNAL
PIN    /EN           ; TO BUFFEROUTPUT ENABLE SIGNAL
PIN    /RDY_O        ; TO CPU READY# SIGNAL AND PLD RDY_I SIGNAL
PIN    RESET         ; TO CPU RESET SIGNAL

; NODES
NODE   PH1           ; MATCHES CPU PH1, USED FOR TIMING
NODE   T2_1          ; ACTIVE DURING FIRST T2, USED FOR TIMING
NODE   T2_2          ; ACTIVE DURING SECOND T2, USED FOR TIMING
NODE   T2_3          ; ACTIVE DURING THIRD T2, USED FOR TIMING

EQUATIONS

RESET:=/PWRGD
PH1:=(/PH1*/RESET)                    ; PH1 SYNCS BY RESET
T2_1:=(/PH1*ADS)+(PH1*T2_1*CS)        ; T2_1 ACTIVE FOR 2 CLK2 CYCLES AFTER ADS#
T2_2:=(/PH1*T2_1)+(PH1*T2_2)          ; T2_2 ACTIVE FOR 2 CLK2 CYCLES AFTER T2_1
T2_3:=(/PH1*T2_2*W_R)+(PH1*T2_3)      ; T2_3 ACTIVE FOR 2 CLK2 CYCLES AFTER T2_2

WE:=W_R*(T2_1+T2_2)                   ; WE ACTIVE FROM MIDDLE OF T2_1 TIL MIDDLE OF T2_3
EN:= CS*(T2_1+(EN*/RDY_I)+(EN*PH1)); EN ACTIVE UNTIL READY
RDY_O:=(W_R*T2_3)                     ; 2 WAIT STATE READY GENERATION FOR WRITE
       +(/W_R*T2_2)                   ; 1 WAIT STATE READY GENERATION FOR READ
RDY_O.TRST=/LBA                       ; RDY_O TRISTATED WHEN LBA# IS ACTIVE (CPU
                                      ; GENERATING RESET)

SIMULATION

TRACE_ON CLK2 PH1 PWRGD RESET LBA T2_1 T2_2 T2_3 ADS CS WE EN RDY_O RDY_I
SETF CLK2 /PWRGD /ADS /CS /LBA /RDY_I
PRLDF /PH1 /T2_1 /T2_2 /T2_3
```

```
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
SETF PWRGD
CLOCKF CLK2
CLOCKF CLK2
SETF ADS /W_R CS
CLOCKF CLK2
CLOCKF CLK2
SETF /ADS LBA
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
SETF RDY_I
CLOCKF CLK2
CLOCKF CLK2
SETF /CS /LBA /RDY_I
CLOCKF CLK2
CLOCKF CLK2
SETF ADS W_R CS
CLOCKF CLK2
CLOCKF CLK2
SETF /ADS
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
SETF RDY_I
CLOCKF CLK2
SETF /CS
CLOCKF CLK2
SETF /RDY_I
CLOCKF CLK2
SETF  CS ADS /W_R
CLOCKF CLK2
CLOCKF CLK2
SETF /ADS
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
SETF RDY_I
CLOCKF CLK2
CLOCKF CLK2
SETF /RDY_I
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2

TRACE_OFF
```

APPENDIX B
PLDShell Plus* WAVEFORM



NOTES:

1. This waveform was exported from PLDShell PLUS*. The waveform files are included on the BBS under EXFLASH.ZIP. The T-state annotation was added and is not normally included with the PLDshell PLUS waveform.
2. PH1 is generated by the PLD and is used instead of PH2 for consistency with the EV386EX evalution board. PH1 is simply the inverse of the PH2 as seen in all the previous figures.
3. T2_1, T2_2, and T2_3 are used to provide timing for WE# and RDY_O# generation. When booting from the 28F400BX the Intel386 EX CPU drives READY# so T2_1 and T2_2 are not needed. However, they are still be generated and can be ignored.
4. For this simulation, CLK2, PWRGD, LBA#, ADS#, CS#, and RDY_I# are all forced inputs.

PLDShell PLUS* Waveform

APPENDIX C
SAMPLE MEMORY REQUIREMENT CALCULATIONS

This spreadsheet is posted on the Intel Application Support BBS @ 916-356-3600. The filename is EXFLASH.ZIP and it is located in the E3X\REF_DSGN section.

The flash memory timing specs must lie within the minimum and maximum specs given below for a desired wait-state performance. However, these calculations do not take into account the delays due to needed glue logic or buffers and assume a non-pipelined interface.

Memory Requirements for the Intel386 EX Microprocessor

Read Specs			16 MHz			20 MHz			25 MHz		
			0 WS	1 WS	2 WS	0 WS	1 WS	2 WS	0 WS	1 WS	2 WS
t_{AVQV} t_{ACC}	Address Valid to Data Valid	Max	74	137	199	53	103	153	44	84	124
t_{ELQV} t_{CE}	CE# Valid to Data Valid	Max	64	127	189	46	93	143	34	74	114
t_{GLQV} t_{OE}	OE# Valid to Data Valid	Min	43	105	168	30	70	80	24	64	104
t_{GHQZ} t_{DF}	OE# High to Output High Z	Max	18	18	18	15	15	15	10	10	10
Write Specs											
t_{AVWH} t_{AS}	Address Valid to WE# High	Min	79	141	204	60	110	160	45	85	125
t_{DVWH} t_{DS}	Data Valid to WE# High	Min	54	116	179	39	89	139	33	73	113
t_{WLWH} t_{WP}	WE# Pulse Width	Min	79	141	204	60	110	160	45	85	125
t_{WHAX} t_{AH}	Address Hold from WE# High	Min	5	5	5	5	5	5	5	5	5
t_{WHWL} t_{WPH}	WE# Pulse Width High	Min	31	31	31	25	25	25	20	20	20

intel®

AP-608

APPLICATION NOTE

Implementing a Plug and Play BIOS Using Intel's Boot Block Flash Memory

CHARLES A. ANYIMI
TECHNICAL MARKETING ENGINEER

February 1995

Order Number: 292161-001

1.0 INTRODUCTION

Today's PC can perform a myriad of functions, perhaps far more than the original designers of the PC ever envisioned. The number of software packages available for mass consumption is staggering, and more are being unveiled each year. Multimedia has enabled the PC to invade every nook and cranny of the home. The number of systems being purchased with CD-ROMS, sound cards, and graphics accelerators is growing at a phenomenal rate. The demand for additional hardware has led to an unprecedented craze for add-in cards and peripherals. While all this growth has been a tremendous boon, it has had its downside as well. The PC has become so sophisticated that new bus structures have been defined, new protocols have been introduced and new system configurations have emerged. All these changes have made an already complex tool more difficult to use by the PC user.

With the advent of Plug and Play (PnP), it seems a solution is in sight. Simply put, Plug and Play is a way of adding new features to a system without the usual headaches—like reconfiguring switches and jumpers, updating system configuration files or other frustrating things. PnP enables a system to automatically configure system components, peripherals and add-in devices just prior to boot time. The basic input/output system (BIOS) of PnP is responsible for the majority of the auto-configuration of the system.

With its previous history of changes and the obvious future changes that will take place as a result of PnP, it only makes sense to store the BIOS on a medium that allows the most flexible means of updating, while maintaining a high level of reliability. Of course, all this has to be accomplished without adding to system costs or making it more difficult for the PC user to get their work done. A PnP BIOS based on Intel's boot block flash meets all of these demands and more. The PnP Specification, for instance, requires nonvolatile storage for old bus standards; the parameter blocks of the boot block flash were designed for such a function. As far as recovery code is concerned, the hardware-lockable boot block area of the boot block flash memory provides unparalleled data protection and guarantees system recovery. The inherent updateability of the boot block flash memory, while in-system, reduces cost by supporting easy code changes and eliminating sockets.

This application note provides a methodology for implementing a flash memory PnP BIOS using Intel's boot block flash. Emphasis is placed on PnP requirements, hardware, and software considerations. Section 2 discusses the current BIOS paradigm and its future. Section 3 provides an understanding of the Plug and Play architecture. Section 4 gives a high-level overview of the boot block products. Section 5 discusses the implementation while Section 6 looks at alternative solutions.

2.0 BENEFITS OF A PNP BOOT BLOCK FLASH BIOS

Perhaps you are wondering why flash is the medium of choice advocated in this application note as the means for storing system BIOS, particularly for Plug and Play? From a PC user's perspective, a PnP flash BIOS enables users to install new hardware without having to call the support number. This translates to ease-of-use:

- Easily updatable code assuring optimal BIOS performance
- No need to edit the CONFIG.SYS file.
- No need to determine system type and match, somehow, to jumper settings.
- No need to investigate available system resources and muddle through reassigning them.
- No need to fiddle with system memory reallocation.
- No need to buy a new system every time something changes (increases system's useful lifespan).

Plug and Play can make the usual experience of adding new functionality to an existing system as easy as:

1. Turn the system off.
2. Insert the new device.
3. Turn the system on.

PnP flash BIOS can also extend the life of the PC. By enabling simple updates to the BIOS (simply insert the upgrade disk and the software programs the new BIOS), the user can get more out of their PC investment.

From an OEM or system manufacturer's standpoint, a PnP flash BIOS enables the following benefits:

- Eliminates the need for excessive EPROM inventories.
- Reduces the need for sockets since flash can be soldered onto the motherboard and updated in-system
- Minimizes system chip count and system cost.
- Reduces support costs.
- Improves end-user perception of product upgradeability and ease-of-use.

When new features are added to the BIOS, a simple disk sent to the user or a connection to an on-line network is all it takes for the user to achieve the upgrade. This has one very important benefit for the OEM/manufacturer—it strengthens user faith in the product (and its manufacturer) and enhances their perception of the product's ease-of-use—which can be a great differentiation. Additionally, since each user no longer needs to call the technical support line as often, this costly overhead can be reduced, saving even more money.

All these benefits are a result of the capabilities that a flash PnP BIOS enables. Looking back to the original PC, it was a fairly simple machine (by today's standards) with simple code designed to perform computational math functions, database creation and management, and word processing. Even though the BIOS code was elementary, few people understood it (or needed to). Since that time, the PC BIOS has (and will) continued to evolve in order to accommodate the growing needs of the PC user.

2.1 The Old BIOS Paradigm

In the original PC architecture, the BIOS code was fairly straightforward and required little memory space—about 32 KB total. BIOS code provided the lowest-level of interface between the operating system and the hardware. It was located at the top of memory for the 8088 system (the original PC had a memory limit of 1 MB). Even though only 32 KB were required, the PC designers knew the benefit of "breathing room" and, with great foresight, reserved a total of 128 KB (from hex address E0000h to FFFFFh) for BIOS code storage, as shown in Figure 1. When the AT was launched in 1984, its BIOS functions were extended by another 32 KB, making a total of 64 KB.

However, the introduction of BIOS created a crutch that has been integrated into the PC: every new system hardware component that was not already supported in the BIOS required a new BIOS to be generated, or at least an upgrade of the existing BIOS. This meant OEMs and system manufacturers needed to keep abundant quantities of ROMs and EPROMs handy to accommodate new BIOS versions. This was a cumbersome and expensive solution, but it was soon accepted as the norm. And, as long as the frequency of change to the BIOS remained minimal, this solution was adequate.

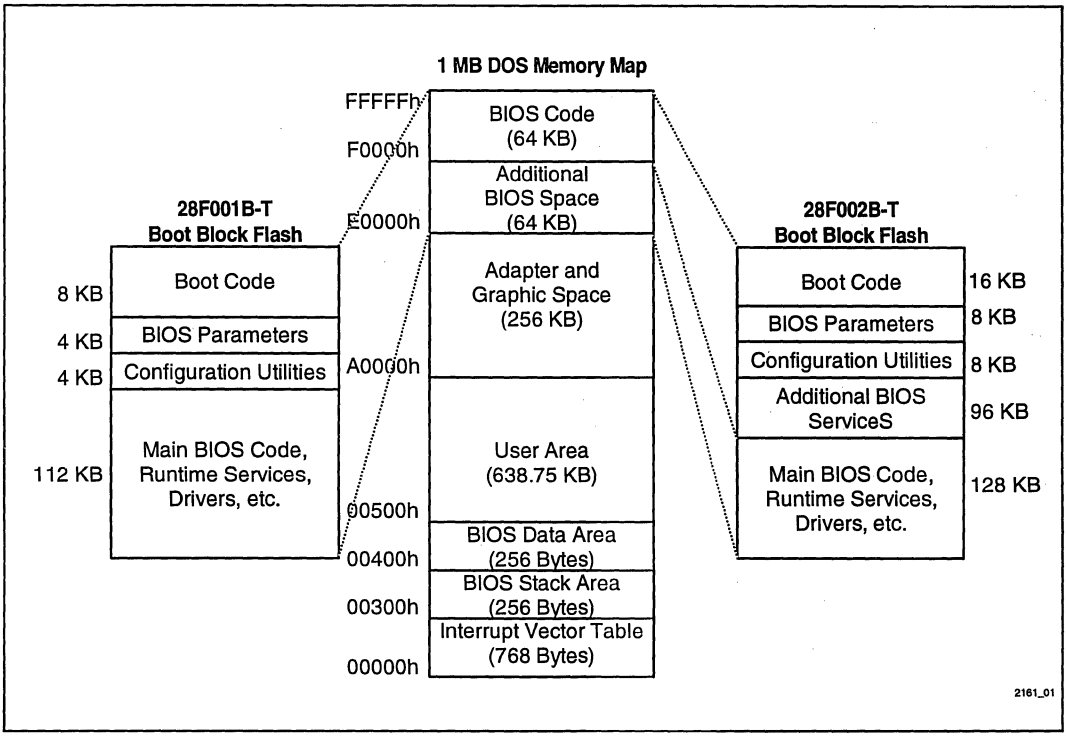


Figure 1. Possible BIOS Segmentations Using Flash Components (1-Mbit or 2-Mbit)

2.2 Beyond the 128 KB Limit

Since then, BIOS code has undergone continual development, including the addition of custom features to enhance system flexibility and more complex set-up routines. Today's BIOS includes support for previously "advanced" features such as system configuration analysis (diagnostics), power management, networking capability and I/O support for devices with extremely high transfer rates. This expansion continued until the AT BIOS eventually consumed the entire reserved BIOS space—all 128 KB!

So now what? With the definition of new bus specifications (EISA, VL, PCI, to name a few) and the further proliferation of features such as ROM versions of application software and enhanced video BIOS, the potential for future BIOS changes looms larger than ever, and the frequency of change seems no less daunting. A further look at the 1-MB memory map of the PC indicates that BIOS needs to prepare for more change and remain open-minded about the future (and it would not hurt any if the BIOS acquired more memory space either—but this would mean changing the architecture of the standard PC).

Today, PnP BIOS requirements may extend beyond 128 KB. A standard system BIOS consumes roughly 64 KB; PnP support is another 12 KB; automatic power management support adds 2 to 10 KB; PCI (Peripheral Components Interconnect) extensions hoard some 20 KB of the BIOS; and on-board VGA (Video Graphics Adapter) controllers utilize an extra 30 KB to 40 KB of space. The potential size of the BIOS for such a system is almost 150 KB. Since new developments for the PC show no signs of abatement, it seems clear that a means of updating as well as expanding BIOS quickly and easily has become a necessity. Plug and Play is only one of the many technologies that can be implemented better with flash. As users realize the benefits of PnP, the demand generated will drive the number of Plug and Play systems upward.

3.0 PLUG AND PLAY

The Plug and Play architecture is intended to alleviate configuration woes and provide the end-user with an easy means of expanding the capability of their PC. To support PnP, several new components need to be added to the host system and add-in cards. For instance, a new type of add-in card capable of auto-configuration is required. Additionally, system firmware and software is necessary to supervise the allocation of system resources and carry out the configuration of these new add-in cards. For widespread acceptance, PnP has to support all the major bus structures. PnP executes on the PCI bus by design. The PnP-ISA extensions enable Plug and Play functionality on the ISA bus while maintaining support for traditional (non-PnP) add-in cards—affectionately referred to as legacy devices. All systems that claim PnP support must recognize the existence of legacy devices and auto-configure new PnP devices around these static devices. On-going development will soon qualify Plug and Play on the VL, EISA and MCA bus structures as well as the PCMCIA interface.

3.1 PnP Components

For a system to be fully PnP-compliant, four basic elements are required:

1. System/PnP BIOS
2. PnP operating system
3. PnP hardware devices
4. PnP application software

While waiting for full-featured PnP operating systems like Windows* 95, Windows NT and PnP versions of OS/2*, solutions are available from individual vendors, including Phoenix Technologies*, SystemSoft* and Intel. In addition, the requirements for PnP implementation vary slightly depending on which bus architecture is being utilized. Even though Plug and Play is an extension of the emerging PCI bus definition, it will revolutionize standard buses like ISA and EISA by making them more user-friendly. The software architecture for supporting PnP consists of the following components (see Figure 2):

1. *Platform BIOS*: interfaces to the PnP BIOS Extensions block; provides error reporting, buffer allocation and platform-specific configuration; provides ESCD interface.
2. *PnP BIOS Extensions*: auto-configures PCI cards, add-in cards, and system board devices during power-up as part of the power-on self test (POST) procedure; provides run-time support to system.
3. *Extended System Configuration Data (ESCD)*: a data structure designed to store configuration information for all system devices, including the last working system configuration; the ESCD **must** be stored in a nonvolatile area.

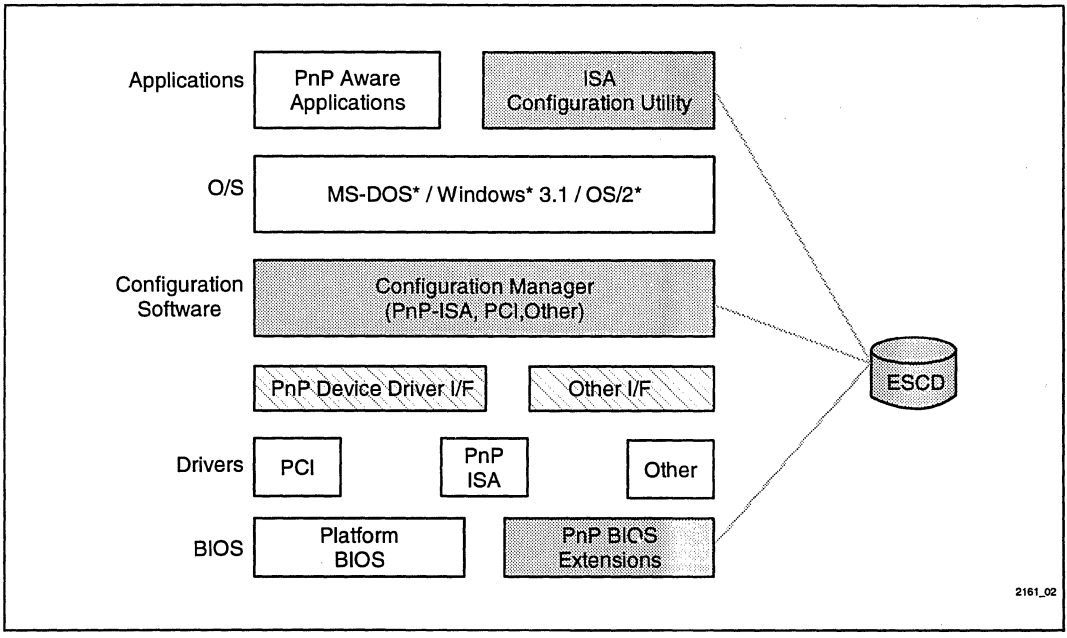


Figure 2. Plug and Play Software Architecture Components. Note the many layers that depend on the ESCD. The parameter blocks of the boot block flash memory enable this nonvolatile storage capability of Plug and Play.

The following components are operating system dependent:

4. *Configuration Manager (CM)*: a DOS/Win device driver that auto-configures any ISA add-in cards not configured by the PnP BIOS Extensions during POST; provides other device drivers and PnP-aware applications like the ICU access to configuration information for all system devices (two VxDs provide similar access privileges to PnP-aware drivers and applications running under windows); provides interfaces for PCMCIA software to get configuration data, but does not provide configuration services.
5. *ISA Configuration Utility (ICU)*[1]: a utility designed to assist users in selecting conflict-free configurations for legacy ISA add-in cards; advises end-users on resource settings and saves this new information into the extended system configuration data (ESCD) for future use by PnP BIOS Extensions or the CM; supports manual configuration of PnP devices. (Caution: it is best to know what you are doing before embarking down this road—advanced users haven.)

Of the components listed above, the platform BIOS, PnP BIOS extensions, and ESCD are the system critical portions without which PnP support would be incomplete. The configuration manager and the ISA configuration utility provide functionality that may be incorporated into the firmware of the operating system. The PnP BIOS and the ESCD are the components that initialize the system when power is applied.

3.2 PnP Functionality

As already mentioned, the PnP BIOS is an essential part of the PnP system. The traditional system BIOS does not address resource management. Its knowledge is limited to hard-wired devices, only. In a PnP environment, the

[1] The Intel PnP Kit R1.21 and R1.23 update include both DOS and Windows versions of the ICU (release R1.3 and R1.4 are Windows only at this time. Other configuration kits are also available: SystemSoft offers PnPView and Phoenix Technologies offers Phoenix System Essential.

system BIOS knows what resources are being used by system board devices and peripherals. During POST, the system BIOS communicates this information to the PnP BIOS, which then detects any PnP hardware devices and initializes them. PnP BIOS also adds the capability of runtime configuration; the system can now dynamically change the resources allocated to system board devices and add-in peripherals (if they have been so designed) after the operating system has been loaded. Working in tandem with the existing system BIOS, the PnP BIOS can detect newly installed devices during the POST and communicate this information to the system at runtime (see Figure 3).

Furthermore, the PnP BIOS is capable of event management. Through its event management interfaces, the PnP BIOS can alert the system about new devices, like a notebook docking station, added or removed during runtime.

The POST procedure of the PnP BIOS identifies, tests, and configures the system before passing control to the operating system. The POST process must maintain previous POST compatibility, configure all legacy devices known to the PnP BIOS, arbitrate resources, initialize the IPL (Initial Program Load—this is how the operating system is launched), and support both PnP and non-PnP operating systems. Upon completion of the POST process, the BIOS attempts to have all necessary system devices initialized and enabled before the operating system is loaded; the PnP BIOS aspires further to provide the operating system a conflict-free environment in which to boot.

The key to a conflict-free operating environment is effective management of system resources; unfortunately there is no definitive directive on how system resources should be allocated. The firmware of most devices does not contain information on how much memory the device will need. Even if this knowledge was available, there is no consistent way of extracting this information. The PnP BIOS Specification identifies three methods for attaining effective resource management: static resource allocation, dynamic resource allocation, and combined resource allocation. The choice of which is used depends on the devices that are being supported by the system.

- *Static Resource Allocation*: This method advocates the allocation of system resources based on the Last Working Configuration (LWC) and is best for systems that have many legacy or static devices that must be resourced. As its name implies, the resource allocation for all configurable devices is predetermined and fixed. This information on the specific resources assigned to all configurable devices must be stored in some nonvolatile location until needed. The ESCD is the structure specified for storing this information. The ESCD in this allocation scheme requires updateability (in case a new device is added and the resource allocations have to be adjusted) and nonvolatility (so the information is always available to the system BIOS during POST). As long as new devices are not added, or previous ones removed, conflict detection and resolution (CDR) is not necessary and the LWC is used at each boot. However, the capability to perform CDR must be available when it is needed.
- *Dynamic Resource Allocation*: This approach assumes that all PnP devices know exactly how much space they will require and what resources will be needed. This approach is best suited for systems with few static devices or a system whose configuration changes frequently. A complete knowledge of which legacy devices are being used and what resources they consume is necessary to insure true conflict-free operation. The legacy device information (and any locked PnP card configuration) should be stored in nonvolatile memory. The principle benefits of this approach are its minimal nonvolatile memory requirements and the flexibility of support for PnP devices. Each boot could potentially yield a different configuration and PnP devices could be added or removed without changing the legacy information in the ESCD. The CDR algorithm will run each time a new system configuration has to be determined.
- *Combined Static and Dynamic Resource Allocation*: This method bases resource allocation on the last working configuration (stored in the ESCD) and also probes other devices, whose information is not contained in the ESCD, for their resource needs. A balanced environment is the primary target for this type of allocation. The system dynamically caters to its new requirements, shuffling previous resource assignments as necessary to satisfy as many devices as possible. Once a new conflict-free environment is established, the system updates the ESCD with the new configuration and checks against this new information during the next boot or system reset. As with the dynamic scheme, every new system configuration that must be determined requires some kind of CDR algorithm in order to insure a conflict-free operating environment.

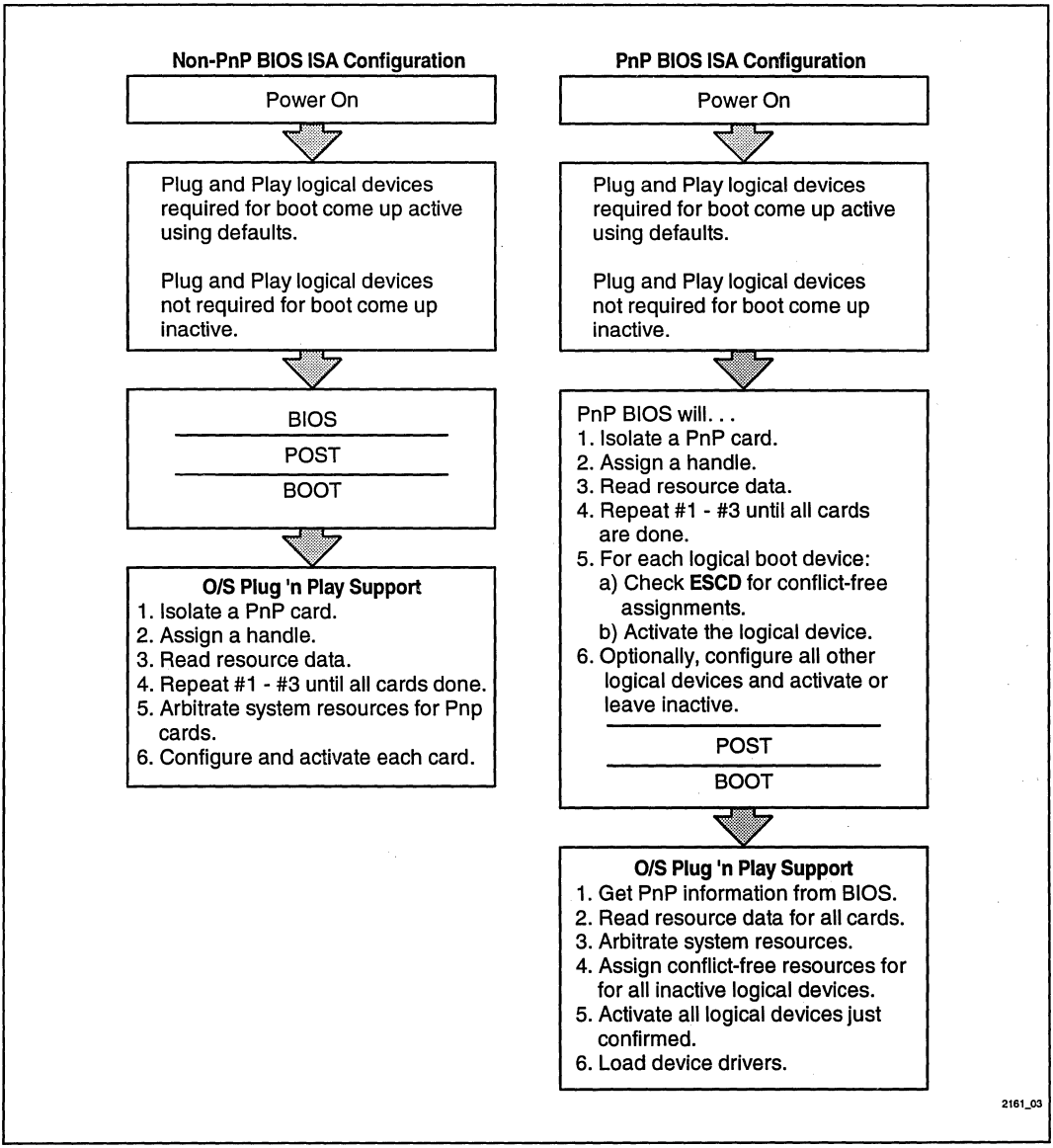


Figure 3. Possible PnP/Non-PnP BIOS ISA Add-In Card Configuration Flow. Note the importance of the ESCD in the Plug and Play Configuration Flow

Without an ESCD, the PnP BIOS must perform resource allocation as well as conflict detection and resolution each and every time the system is booted, and any locking of devices must be done by the supporting PnP operating system. Although the need for nonvolatile storage cannot be disputed, the amount of storage required depends on whether or not the PnP system needs real time updates. Ultimately, it will be decided by the methodology selected for resource management. A static or dynamic allocation scheme requires a fixed

amount of nonvolatile memory for the ESCD and other BIOS parameters. A combined scheme for allocation constantly adds or removes from the ESCD data structure. Other parameters may need to be updated as a result. Regardless of the storage method chosen, the BIOS must know how to resolve any untimely interruption of a crucial system or BIOS function. The underlying mechanism for appeasing all these requirements is flash, and Intel's boot block flash is the solution for today's demands and tomorrow's inclinations.

3.3 Locked Devices & Bitmap Storage

The ESCD has the further capability of locking particular resources allocated to a particular device. This allocation is maintained no matter how often the system configuration is changed. All future system configurations are performed around the resources assigned to this locked device. This capability is useful for performance reasons or advanced applications such as server or network.

A bitmap structure for storing configuration information is not capable of locking due to the limited information it contains. Compared to an ESCD storage structure, a bitmap contains bits which give only binary information, e.g., which I/O range is being used or which interrupts and DMAs are unavailable. Although the bitmap storage structure uses less nonvolatile memory, it does not contain the detailed information needed for full Plug and Play support. For instance, you could not utilize the robust allocation algorithms of the operating system with a bitmap structure. In addition to not supporting locking, the bitmap structure also does not allow device enabling and disabling, which is essential for disabling PnP devices in non-PnP environments.

4.0 INTEL'S BOOT BLOCK: THE PNP FLASH BIOS SOLUTION

With the background information on Plug and Play explained in the previous sections, a full analysis of the Intel boot block and how it meets the needs of Plug and Play follows. Included are a description of the boot block family of products, how they meet the PnP requirements, and a hardware design example.

The Intel boot block (BB) flash memory products are particularly well suited for BIOS applications. Boot block flash is segmented into a lockable boot block, two parameter blocks and one or more main blocks. All boot block devices are manufactured on Intel's ETOX™ flash memory process technology. An on-chip Write State Machine (WSM) provides automated program and erase algorithms with an SRAM-compatible write interface. The key feature of the boot block architecture that differentiates it from other flash memories is its hardware-lockable boot block, which allows system recovery from fatal crashes. Additional features of boot block flash include:

- Hardware write protection via pin
- Hardware locking of boot block
- Hardware pin for system reset during write
- High performance reads (for speedy access to data)
- Deep power-down mode (a key feature for "green" PCs)
- Extended cycling capability (100,000 block/erase cycles)

Armed with this impressive array of features, Intel's boot block flash memory tackles the PnP BIOS challenge and prevails with a winning solution. Boot block flash memory meets the needs of the Plug and Play BIOS, the PnP data storage area (ESCD), and the PnP BIOS boot code.

The main block(s) of the boot block flash can be used to house PnP BIOS code. This code will doubtless include the standard AT BIOS code (all 64 Kbytes) as well as the PnP specific additions to the standard BIOS, an extra 10 Kbytes–20 Kbytes. Any additional features that particular OEMs or vendors wish to implement (i.e., power management, virus protection, PCI, etc.) will exhaust more memory. The main block of the 1-Mb boot block is 112 KB; the 2-Mb has one 96-KB main block and one 128-KB block. The 4-Mb boot block is similar to the 2-Mb except it has three 128-KB main blocks. The choice of which boot block to use depends on BIOS specifications and other system requirements. For some systems, the 1-Mb boot block is sufficient. However, other systems have advanced features that require more memory, for example

- Improved help files
- System diagnostics code
- Foreign language support
- Integrated SCSI subsystems

For these applications, a 2-Mb boot block is the prudent choice. With twice the memory space as the standard PC memory map allots to BIOS, the 2-Mb boot block flash offers plenty of head room—just what a growing PC needs.

The previous section touched on some of the methods available for resource management and the amount of nonvolatile memory necessary for each approach. Boot block flash solves the issue of the ESCD for each of the possible allocation schemes.

The static allocation scheme stores the resource requirements for all system and add-in devices within the ESCD. These system configurations can be programmed into one of the parameter blocks of the boot block flash as the ESCD. Parameter blocks are either 4 KB (1-Mb boot block flash) or 8 KB (2-Mb and 4-Mb boot block flash) in size. The ESCD Specification calls for 4 KB of nonvolatile memory for the ESCD. Of course, an OEM may elect to define its own version of the ESCD, but that structure will still need to be stored in some nonvolatile location. If the OEM or system manufacturer decides to include additional features within the ESCD (or keep a second copy of the ESCD for recovery purposes), the other parameter block can certainly be used for this purpose.

With dynamic resource allocation, the boot block architecture's parameter block is an excellent choice for storing the resource information of legacy devices. When systems are being put together, all devices on that system can be pre-assigned specific resources and this information is saved into the ESCD. This implies that some conflict resolution protocol or intelligent allocation algorithm needs to be implemented to assign resources to any devices added to the system by the end-user. This protocol can be included in the BIOS or as part of the operating system.

For the combined approach, the dual parameter blocks of the boot block family again come into play. Using this feature of the boot block architecture, two versions of working system configurations can be saved. This means that the last two working configurations are always available to the system, further insuring recovery in case of irreconcilable conflicts. If this is not desired, then the ESCD can be written alternatively to each parameter block, thereby minimizing the number of writes to the same location and extending the life of the flash component.

Finally, the hardware-lockable boot block of the flash architecture is ideal for BIOS boot code. The boot code is the first piece of code executed each and every time the system is turned on or rebooted. Boot code consists of a jump vector, checksum routine and recovery code. The jump vector, which is 16 bytes long, is the beginning address of the main BIOS. This is the address jumped to after the checksum routine returns a valid checksum, indicating that the current BIOS is good. If the checksum routine does not validate the goodness of the available BIOS, the recovery code is then used to load a new BIOS.

In the AT system, the boot code was usually no more than 8 KB in size. However, the improvements made to start-up routines, checksum routines and recovery code to keep up with the constantly changing times have forced this boot code to grown outside of its intended limit. Further PC enhancements will advance rather than curb this growth. The boot block area of the 1-Mb flash, with its 8 KB size, is the consummate solution for storage of the standard boot code. The 16 KB size of the 2-Mb and 4-Mb boot block is the answer to the growing needs of today, and the unyielding promise of tomorrow, by enabling the boot code to expand beyond itself to better serve the user.

Since this boot code is so important to the operation of the system, it is easy to understand why it needs to be protected. Storing it within the hardware locked boot block provides maximum protection to the system. If a reset occurs during reprogramming of the flash, for instance (the dog has been known to run over the power cord at the most inopportune times), a hardware-locked boot code means that system recovery is not only possible, it is guaranteed! This capability is an asset to OEMs users alike. The user no longer needs to suffer long delays following a system crash; OEMs (and system manufacturers) no longer need to incur the cost of person-dependent recovery. The user does not feel helpless and out of control and the OEM saves money and gains a faithful customer in the process.

5.0 IMPLEMENTING A PNP FLASH BIOS USING A 2-Mb BOOT BLOCK

By now it is evident that Intel's boot block flash memory is an ideal solution for implementing PnP BIOS within a system. However, there are implementation criteria that need to be explained:

- How does this hardware-locking of the boot block work?
- How does one connect the flash memory to a system?
- What about address mapping?
- How does one utilize a 256-KB BIOS within an defined 128-KB BIOS space?

These are just some of the implementation concerns that need to be addressed. The following section will shed some light on these questions and provide an example implementation of a PnP flash BIOS.

5.1 Overview of the 2-Mb Boot Block

Figure 1 showed that either a 1-Mb or 2-Mb flash device can be used to implement BIOS. The choice of device depends on the contents of the BIOS and the level of sophistication desired in the design. The 1-Mb boot block flash memory is an established standard for implementing flash BIOS, not just for PnP. However, more BIOS space will be needed to support standardization of current features (like power management and virus aids) and impending future enhancements (like Windows 95 and Desktop Management Interfaces*, DMI). As consumers clamor for "more bang for the buck," more features and functions will be integrated into the standard PC. The migration beyond a 128-KB BIOS is inevitable. Already, some vendors have adopted code compression of BIOS in order to adhere to this 128-KB space limitation. This is a viable alternative that requires additional code decompression algorithms and consumes additional RAM space to store the full BIOS. Fortunately, Intel provides a secure means of code storage as well as a built-in growth path with its boot block architecture.

Within this implementation section, references to the flash BIOS device assume the 28F002B boot block flash. A similar approach can be followed to implement a similar solution using the 1-Mb boot block. However, some of the techniques included in this design example will not be applicable. The organization and addressing scheme of the 28F002B device, along with that of the 1-Mb boot block (as reference), is depicted is Figure 4. The pinout for the Thin Small Outline Package (TSOP) and Plastic Small Outline Package (PSOP) of the 28F002B are shown in the Appendix. A table listing the functions of each of the pins identified in the pin diagrams is also available in the Appendix. The five, independently erasable blocks consist of the hardware-lockable boot block (16 KB), the two parameter blocks (8 KB each), and two main blocks (a 96-KB block and a 128-KB block). The hardware-lockable boot block can be located at either the top (28F002B-T) or bottom (28F002B-B) of the 1-MB memory map, enabling easy interface to all Intel architecture microprocessors as well as embedded processor like the i960® processor (KA/SA) and non-Intel microprocessors that support location of the BIOS memory area at the low address.

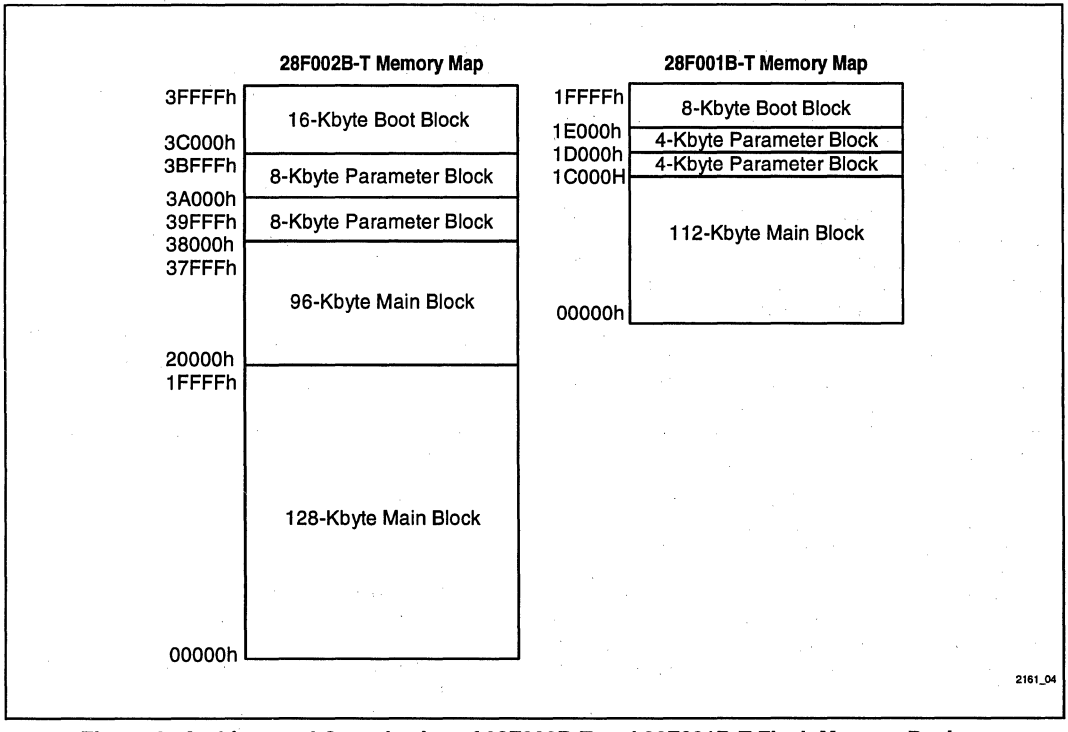


Figure 4. Architectural Organization of 28F002B-T and 28F001B-T Flash Memory Devices

The 16-KB boot block is intended for storage of the system critical BIOS boot code. This block is unlocked when the RP# pin is between the specified range for V_{PPH}; after unlocking, program and erase operations can be performed. Taking the RP# pin below the specified minimum value for V_{PPH} locks this block, disabling program and erase functions. The two parameter blocks can contain supplementary boot code or the system configuration information (ESCD). They are intended for storage of frequently updated system parameters or configurations. In addition to the ESCD, the nonvolatile parameter blocks can be used to retain a copy of the CMOS setup or to store/track add-in card addresses, DMA channels, or interrupt values/level. The main blocks may be used for the storage of the main PnP BIOS code and runtime services. The WE# input provides write protection for the entire flash memory device. The V_{PP} pin offers additional write protection since standard boot block flash requires V_{PPH} be between 11.4V and 12.6V before any Program or Erase command sequences are recognized.

Sometimes, however, design constraints make it difficult to provide 12V ± 5% to the flash memory. It was to accommodate situations such as these that Intel designed SmartVoltage flash memory. SmartVoltage (SVT) boot block products include V_{PP} sense circuitry which allows both 12V and 5V program and erase. In addition, read capability is available at both 5V and 3.3V. These features provide additional design benefits and implementation flexibility to serve multiple environments. Higher assembly line throughput, for example, can be achieved using 12V V_{PP} whereas some in-system programming layout and trace difficulties can be greatly eased with a 5V V_{PP}. When the V_{PP} input of a SmartVoltage device is connected to 5V, the range for V_{PPH} is 4.5V to 5.5V. The SVT devices even include a dedicated write protect pin for locking and/or unlocking the boot block with a logic signal. The combination of SmartVoltage and the boot block architecture provides a robust solution that eliminates design barriers and quickens system time-to-market.

A complete solution cannot be achieved without mention of available package options. The Boot block products are offered in PDIP, PLCC, PSOP, and TSOP form factors. Due to handling similarities in the manufacturing flow, more vendors are changing from the previous PLCC package standard to the smaller PSOP package.

5.2 PnP Boot Block Flash BIOS Implementation (Hardware)

In order to implement a PnP flash BIOS, certain hardware criteria must be met. For standard boot block flash and SmartVoltage boot block in 12V mode, there must be a means for raising V_{PP} to 12V and lowering it to normal levels after programming or erasure is complete. There must also exist a method for write-enabling the entire flash device. A way of gating the RP# input is necessary to insure the integrity of the program signal for the boot block (usually a POWERGOOD signal is appropriate). Bi-directional transceivers or data buffers may be needed for the DQ pins. Lastly, there must be a means for relocating the recovery code after the system boots when a 2-Mb density boot block flash is used.

5.2.1 BIOS BOOT CODE RELOCATION

Following a system reset or power-on, the typical system first goes to the high memory area of the 1-MB memory map, where the boot code is stored. The checksum routine (or whatever means of verification is employed) is run to verify the status of the BIOS currently available to the system. If the main BIOS is determined to be good, this code proceeds to initialize the system and its peripherals and passes control to the operating system; as mentioned earlier, this is done by jumping to the address pointed to by the jump vector. If, however, the main BIOS is found to be corrupted or unusable (i.e., the checksum value read does not match the expected value), the BIOS recovery code must reconstruct a new BIOS. The recovery code reconstructs the BIOS by reading the BIOS update file from a floppy or COM port (modem), erasing all the other blocks (except the boot block), reprogramming the flash with the new BIOS data, and initiating a RESET to reboot the system. As a result, there are two modes of operation in which the system can function: boot mode and runtime mode.

In boot mode, which occurs at power-up, the system expects the kernel code to be located at physical address FE000h–FFFFFh. The system then validates the status of main BIOS. If this check results in a good BIOS, the system is initialized and control is transferred to the operating system via the jump vector. At this point runtime mode is entered; the system now expects the main BIOS to be located at physical address E0000h–FFFFFh (see Figure 5). The 1-Mb boot block maps directly into this 128-KB space allocated for BIOS usage. The 2-Mb boot block, however, must be able to switch between its upper 128-KB block (which includes the boot and parameter blocks) and its lower 128-KB block (which is where the main BIOS should be stored).

The system initialization routine and the configuration utilities should be located in the upper 128-KB block of the 2-Mb boot block. The lower 128-KB block should be used for runtime BIOS services and other advanced features.

To achieve this swapping of boot BIOS and main BIOS, several methods can be applied. A small piece of logic can be added to the board to swap the address ranges; an easier approach is to simply invert the A_{17} input to the flash. For runtime mode operation, A_{17} is maintained at logic low, thus mapping the lower 128-KB block of the 2-Mb boot block to the 128-KB BIOS area in main memory. During recovery, the polarity of A_{17} can be flipped to shift the kernel code to the F segment, i.e., swap the lower 128-KB block for the upper 128-KB block. A keyboard sequence, motherboard switch, or jumper can be used to toggle A_{17}. The hardware example in Figure 6 illustrates the use of this tactic. Another approach would be to locate the flash BIOS at a high memory area (above 1 MB) and use BIOS extension calls to access the runtime BIOS services in the 128-KB block. This method, however, takes quite a bit of time because of the overhead associated with the software BIOS call (saving status, return location after call, etc.). Note that depending on the particular BIOS implementation (i.e., vendor, platform, etc.), the addresses indicated in this example may change slightly.

When the flash memory is being reprogrammed, it is necessary to relocate the BIOS recovery code to RAM before proceeding, as Figure 5 illustrates. Although the flash memory allows suspension of an erase cycle to permit reading of another block, attempting to read one of the flash blocks while a write to another block is in progress is not permitted. This is a characteristic of all flash products currently on the market today. As a result, the BIOS recovery code must be copied to RAM and executed out of RAM in order to reprogram the flash memory with the new BIOS code.

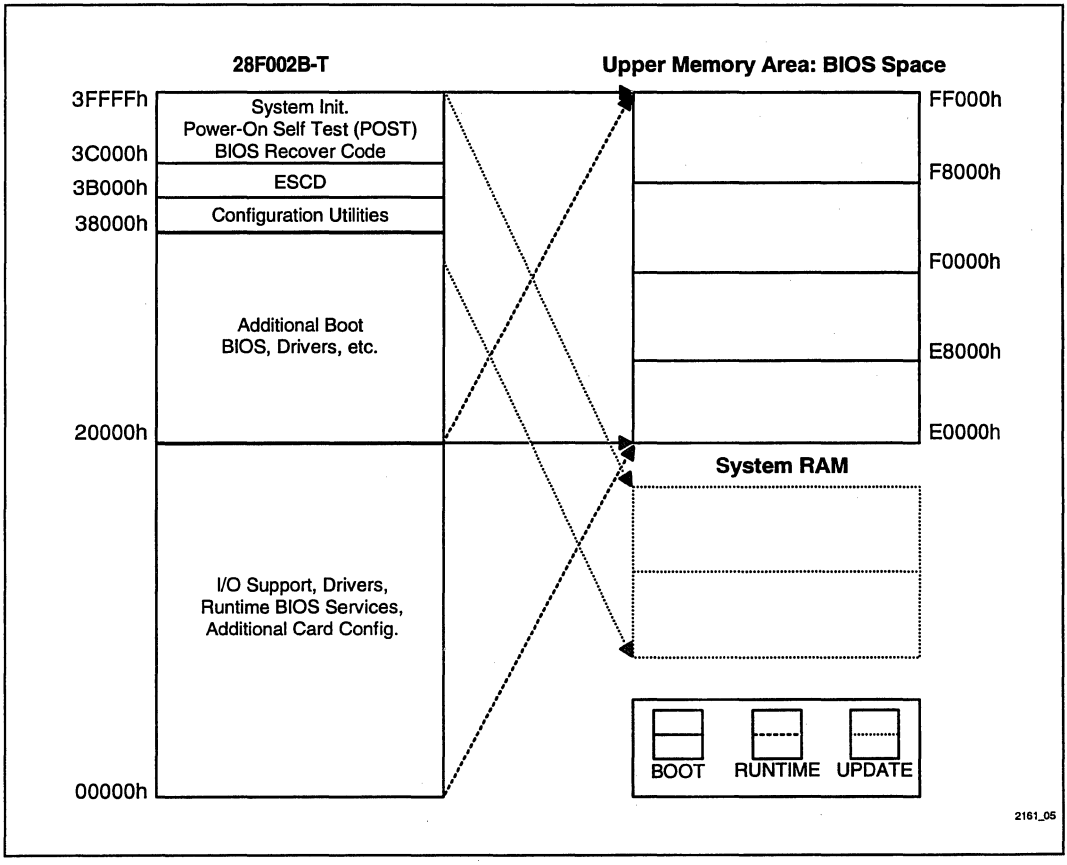


Figure 5. BIOS Relocation for 2-Mb Boot Block

5.2.2 V_{PP} GENERATION AND WRITE PROTECTION

The V_{PP} pin enables programming and erasure of the flash device. In addition to this, V_{PP} also provides write protection of the flash memory blocks. If V_{PP} is below its required level, no Program or Erase command sequence, whether valid or not, will have any affect on the flash. If code (or data) security is of paramount importance to a particular design then a flash memory device with a dedicated V_{PP} pin is the best choice.

As for PnP, the BIOS will be tweaked from time to time by vendors, new features will be added, and standard routines will be enhanced. One particular allocation scheme of PnP requires that the ESCD be re-written each time a new system device is added. Although flash memory enables all these benefits, none of them can be achieved without the generation of the programming voltage, V_{PP}. It is imperative that V_{PP} be generated cleanly to avoid incorrect programming or erase as well as spurious writes (which can lead to unwanted system crashes). Keep in mind that a clean V_{CC} is as important for fail-safe flash operation as a clean V_{PP}.

The IBM PC technical reference manual specifies a 12V supply with a tolerance of + 5% to – 4%. The V_{PP} specifications of the boot block flash memory align to this standard. If the power supply employed in the design meets the IBM specification and has CMOS logic, the 12V supply from the power supply can be tied directly to the 28F002B. This approach, however, is not recommended since it can degrade program/erase performance or unfavorably affect reliability. In most desktops, an unregulated 12V supply exists in addition to a 5V. It is recommended that 5V be used to obtain the 12V ± 5% rail. In addition to being more efficient and more economical than the unregulated method, this

approach does not require a minimum load to maintain the regulation, as is necessary when buffering an unregulated supply using modular solutions. Likewise, V_{PP} can be generated by regulating (or stepping down) from a higher voltage. Additional information on V_{PP} generation strategies can be found in Application Note 357: "Power Supply Solutions for Flash Memory," (order # 292092) and the technical paper entitled "Small and Low-Cost Power Supply Solutions for Intel's Flash Memory Products," (order # 297534).

Of course, if a 5V environment is necessary, SmartVoltage is the irrefutable choice. These voltage sensing devices allow manufacturers or OEMs to choose either a 12V or 5V V_{PP} level, depending on their specific design needs. In this way, the extra write protection provided by having a separate V_{PP} pin is retained and the appropriate hardware environment can be maintained. When V_{PP} falls below the specified value for V_{PPL} (V_{PPLK} for SmartVoltage devices), program and erase cycles to the flash device are prohibited (ignored), although the device can still be read normally. Additional software protection for V_{PP} can be added by requiring a password before enabling V_{PP} to proper program/erase levels. The RP# pin, gated by the POWERGOOD signal of the power supply and the system RESET# pin, provides further write protection for the information stored within the boot block.

5.2.3 HARDWARE DESIGN EXAMPLE

Figure 6 shows an example design for implementing a flash memory-based BIOS within a PC motherboard. Specific signal generation is discussed in the following sections. The V_{PP} generation circuit used can drive 200 mA of V_{PP} current with an efficiency rating of 88%. Even though a transceiver may not be necessary, it is specified in this example as reference. Standard PCs expect a ROM-based BIOS and do not enable the data bus to the BIOS ROM during write sequences (in fact, standard PCs do not generate the write enable signal when the address decoded references the BIOS area). The transceiver are used to allow reading and writing of the flash BIOS within specified timings.

5.2.3.1 V_{PP} Generation Circuit

The LT1301 used in Figure 6 is a micropower step-up DC converter available in an 8-pin surface mount package. The input of the LT1301 is capable of selecting between 5V or 12V outputs, ideal for use with SmartVoltage flash memory. Since series inductance in the filter capacitor and diode switching transients may cause high frequency noise spikes at the output, an optional filter design is included with the example.

The Max662A provides a regulated output voltage at 30 mA from a 5V ± 5% power supply. It uses internal charge pumps and external capacitors to generate 12V, eliminating inductors. Regulation is provided by a pulse skipping scheme that monitors the output voltage level and turns on the charge pumps when the output voltage starts to droop.

Both solutions offer a shutdown pin for protection against accidental erasure of the flash memory. Additionally, good PC-board layout practices can also eliminate this potential problem.

5.2.3.2 RP#

This section gives a sample implementation of the PnP flash BIOS. The block diagram that supports this implementation is shown in Figure 6. The RP# gating methodology described in the previous section is implemented in this sample design; the PWRGOOD signal (or V_{CC} input) and the hardware generated RESET# signal are monitored for appropriate voltage levels using a voltage monitor. This scheme masks invalid bus conditions from the flash device, thus providing additional buffering accidental erasure. As one might expect, the flash memory defaults to read array mode. It may also be desirable to gate RP# with a General Purpose Input/Output (GPIO) line to enable shutting off the BIOS after it has been shadowed. A jumper to 12V (with some kind of protection, like decoupling capacitors or buffer circuit) can be used to unlock the boot block. This control may also be accomplished via software interrupt, although this is a less secure means than the straight hardware method. The SmartVoltage boot block products support this type of boot block locking and unlocking; however, there is a separate WP# pin that permits locking and unlocking of the boot block with 5V if 12V is not being supplied to the flash memory.

5.2.3.3 WE#

The WE# signal generated by the processor usually cannot be used in this implementation because the processor does not expect this area to be writeable (i.e., it thinks the BIOS is stored in a ROM). Therefore, the WE# signal must be generated externally using the bus definition signals and some discrete components. A write condition to the BIOS is established when the M/IO# (memory or I/O) signal indicates memory and the MEMW/R# (memory write or read) signal indicates write. Figure 6 illustrates this scheme. Further write protection for the BIOS can be achieved by gating the WE# signal with a GPIO line, effectively disabling writes to the flash BIOS unless permitted by the BIOS update algorithm. I/O port bits can be ANDed with the actual write pin to control the generation of the WE# signal to the flash memory. The bits used should be both readable and writeable. Flash memory devices that do not have a WE# pin suffer from more frequent spurious writes. Such memories use the CE#, OE#, and V_{PP} pins to decode a write sequence. Because the CE# input is decoded from switching address pins, it is not unheard of for this input to incur glitches. With V_{PP} at specified tolerance levels, this glitching can initiate writes—hardly a favorable state when updating BIOS code.

5.2.3.4 CE#

The CE# input is defined by the address condition that enables the flash device. No access to the flash chip will be permitted if the CE# input is deasserted. This input can be generated multiple ways as well. Chipsets often take care of this type of decoding internally, returning the lone chip select signal on one of their output pins. If a chipset is not used, µPLD or decoder can be used to generate the CE# input based on the address inputs used to indicate the flash memory device. Boot block products allow both CE# controlled program/erase as well as WE# controlled program/erase. The logic is such that whichever is asserted first controls the current program/erase sequence and latches the valid address. The WSM begins operation when that signal is deasserted (see Command User Interface, Section 5.3.2.2, for more details).

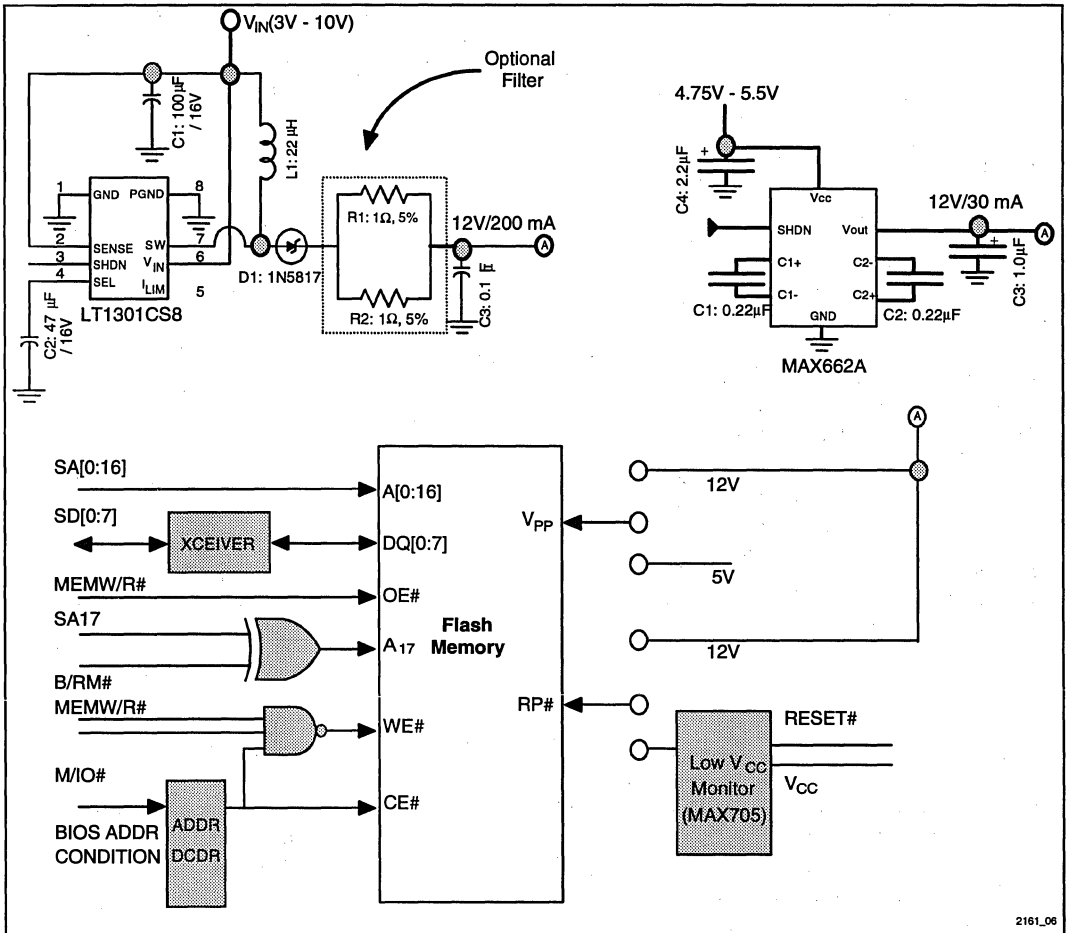
5.2.3.5 OE#

Whenever a read cycle is performed, the OE# input needs to be asserted. OE# is therefore gated by a memory read to the flash when it is enabled. This example uses the MEMW/R# signal to control the generation of OE#. It may be necessary to invert this signal in order to provide the correct signal polarity to this input. As with the WE# input, I/O bits as well as discrete components (that define, in this case, a read cycle) can be used.

5.2.3.6 Address Inversion for 2-Mb Boot Block

The address inversion scheme described earlier is used in this example. This input to the flash can also be gated by I/O port control bits if a software implementation is more appropriate. This example uses the Boot/Runtime Mode (B/RM#) selection pin (which may be software generated—controlled, say, by the checksum value) to control the inversion of A_{17} Alternatively, a programmable device (like a µPLD) can be used to actually decode the high order address bits and achieve the same result as the bit inversion.

Some of the features employed in this example may be incorporated into a chipset, and therefore, the external circuitry may be unnecessary. The CE# input, for example, is available on most chipsets as a ROMCS# output and can be hooked directly to the CE# input of the flash memory.



NOTES (LT1302CS8):
Input Range: 3.0V to 10V
Output Current: Up to 200 mA @ V_{IN} = 5V
Typical Efficiency: 88% @ I_{LOAD} = 200 mA, V_{IN} = 5V
Switching Frequency: 155 Khz
Operating Quiescent Current: 120 µA (typical)
Shutdown Feature
Shutdown Quiescent Current: 15 µA (max.)
Rise time from shutdown: 1.2 ms (typical)

NOTES (MAX662A):
Input Range: 4.75V - 5.5V
Output Current: Up to 30 mA @ V_{IN} = 5V
Typical Efficiency: 74% @ I_{LOAD} = 30 mA, V_{IN} = 5V
Switching Frequency: 500KHz
Operating Quiescent Current: ~320 µA
Shutdown Feature
Shutdown Quiescent Current: ~70 µA
Rise time from shutdown: ~500 µs (typical)

Figure 6. PnP Boot Block Flash BIOS Hardware Implementation Example

5.3 PnP Boot Block Flash BIOS Implementation (Software)

In order to update the PnP BIOS, some type of flash programming utility must be employed. This utility cannot program the boot block due to the hardware protection provided by the RP# pin (or the WP# pin on SmartVoltage flash memory), thereby preserving the boot or recovery code contained therein. In most cases, however, this programming utility will be unique for each system because it is dependent on the hardware used in the design. The method of raising and lowering V_{PP}, for instance, is dependent on hardware, as is the methodology for disabling shadow RAM or cache. Since reprogramming of the flash memory will result in a reboot, it is not necessary to keep track of system status information. (like shadow status, cache status, power management status, cursor position, etc.).

Boot block devices have on-board programming and erase algorithms with a built-in SRAM-compatible interface for simplified software creation and debugging. This is accomplished through the on-chip write state machine (WSM), status and command registers. These registers perform all of the necessary actions, from V_{PP} monitoring to erase suspend. The WSM even times the programming pulses, obviating the need for program timers and preconditions blocks as part of its erase process, eliminating the need to "0" program prior to erase.

5.3.1 PROGRAMMING CONSIDERATIONS

Due to the difficulty of discussing every possible PnP flash programming utility in this application note, a generic programming utility that can work on all platforms will be examined. Inherent to this approach is an interface between the programming utility and the PnP BIOS. This interface is accomplished by selecting a ROM BIOS interrupt number and assigning a function number through which all flash-specific functions can be accessed. Table 1 lists some possible functions that might be defined for the interface. Table 4, in the Appendix, provides a list of the ROM BIOS interrupts currently used or preassigned. Once an available (or unused) interrupt number and/or function has been determined, the flash programming functions can then be defined. For this example, assume interrupt 17, function 0Fh has been selected for implementing the flash programming subfunctions.

Table 1. Generic Subfunctions for Flash Programming Utility

Subfunction	Description
00h	Validate Checksum
01h	Raise Programming Voltage (V_{PP})
02h	Lower Programming Voltage (V_{PP})
03h	Flash Write Enable
04h	Flash Write Disable
05h - FEh	Reserved for Future Use
FFh	Generate System RESET

The generic specification is as follows:

Input: AH = 0Fh
AL = Subfunction

Output: If CARRY FLAG set = Error
If CARRY FLAG clear = Success
AL = 85h

(If 85h is defined as a subfunction in the future, a new return value must be specified.)

The carry flag was chosen because most instruction sets include specific instructions for setting and clearing this bit. The overflow or zero flag may be used in place of the carry flag. Likewise, a register value may be returned in case of an error (as is done with the success case in this example). The methodology may be changed but the function must be preserved, i.e., regardless of how it is done, there must be some way of informing the system of a successful or unsuccessful instruction execution.

A few caveats of which to be aware:

1. A PnP BIOS version subfunction may be defined for distinction or to enable/disable features not supported in every system

2. If both a flash chip and an EPROM exist on the system board (for example SCSI or keyboard BIOS), two additional subfunctions need to be defined to select the flash memory instead of the EPROM.

3. The Validate Checksum function can be defined many ways, but basically it needs to be able to compare the checksum of the current BIOS (this

may need to be calculated) with the saved checksum value. If they match, then boot can continue; otherwise, a new BIOS needs to be uploaded.

4. Keep in mind that all registers used by these subfunctions will be destroyed. If their value needs to be maintained, the register should be pushed onto the stack (or saved) prior to use and then restored afterwards.

Subfunction 00h: Validate Checksum—checks the checksum of the loaded BIOS against that stored in memory. If they match, it returns true, else it returns an error and a new BIOS should be loaded.

Input: AH = 0Fh
AL = 00h
Output: CF set = Error
CF clear = Success
AL = 85h

Subfunction 01h: Raise Programming Voltage (V_{PP})—raises V_{PP} to the required voltage level (in this case, greater than 11.4V) and waits until the voltage is steady.

Input: AH = 0Fh
AL = 01h
Output: CF set = Error
CF clear = Success
AL = 85h

If the boot block area of the flash memory is to be accessed and software control of the RP# input is desired, this function may be used to raise the voltage on the RP# signal. Alternatively, another subfunction may be defined to accomplish this purpose. Remember, however, that software control of the RP# input is not recommended as it eliminates the hardware protection feature of the boot block .

Subfunction 02h: Lower Programming Voltage (V_{CC})—lowers V_{PP} to its normal level (in this case, less than 6.5V) and waits until the voltage is steady.

Input: AH = 0Fh
AL = 02h
Output: CF set = Error
CF clear = Success
AL = 85h

If access to the boot block area of the flash memory is completed and software control of the RP# input is in effect, this function may be used to lower the voltage on the RP# signal. Alternatively, another subfunction may be defined to accomplish this purpose. Remember, however, that software control of the RP# input is not recommended as it eliminates the hardware protection feature of the boot block .

Subfunction 03h: Flash Memory Write Enable—enables erase/program commands to the flash chip and waits the required amount of time for stabilization (if necessary).

Input: AH = 0Fh
AL = 03h
Output: CF set = Error
CF clear = Success
AL = 85h

Subfunction 04h: Flash Memory Write Disable—disables Erase/Program commands to the flash chip and waits the required amount of time for stabilization (if necessary).

Input: AH = 0Fh
AL = 04h
Output: CF set = Error
CF clear = Success
AL = 85h

Subfunction FFh: Generate System RESET—issues the RESET command necessary to reboot the system after the flash memory has been altered.

Input: AH = 0Fh
AL = FFh
Output: None

5.3.2 REPROGRAMMING CONSIDERATIONS

One of the prime benefits of a flash-based PnP BIOS is the ability to do in-system updating. When a flash chip is soldered directly onto a system board, there are two methods available for reprogramming: in-system writing (ISW) and on-board programming (OBP). The major difference is in how V_{PP} is supplied and whether the programming process is controlled by the system or some external hardware. The V_{PP} voltage is supplied locally and the system is responsible for reprogramming with the ISW approach. With OBP, the external PROM programmer supplies the necessary V_{PP} for

programming, and controls the reprogramming process. Cost, ease-of-implementation, and reprogramming environment are some of the trade-offs that must be made when considering which methodology is best. There are advantages to both methods. This application note focuses on the ISW approach.

5.3.2.1 In-System Write Considerations

The following items are required to have an ISW capable system for updating the PnP BIOS:

- Microprocessor or controller (to control the reprogramming process)
- PnP BIOS boot code, communications software, and PnP BIOS update algorithm
- Data import capability (floppy disk, serial, network, etc.)
- V_{PP} generator or regulator (12V products only)

V_{PP} Generation

V_{PP} generation has already been discussed in previous sections, and since most ISW systems include voltage divider circuits that provide a path to ground, ESD protection is not needed for the V_{PP} pin. If, however, a system does not have this voltage divider circuitry (check the schematics) or the V_{PP} supply is switched directly, a resistor to ground should be added to prevent damage due to electrostatic discharge. The tolerance of the V_{PP} pin is also important to be wary of. Although 5% tolerance is tighter than 10%, it usually yields a higher programming time. Such a trade-off may be necessary to make for certain applications. When using the SmartVoltage devices in 12V mode, the same care must be taken in generating V_{PP} as with the standard boot block. In the 5V mode of the SVT devices, however, this extra protection is not necessary. Nonetheless, the V_{CC} signal should be as clean as the V_{PP} signal.

Data Import Capability

The flash memory does not care how the new PnP update code is fed to it—any convenient means of downloading the necessary information is acceptable. This means the flash memory will not be a barrier to completion if, for instance, design constraints call for a parallel link instead of a serial link. Even though most communication is serial, error free serial communication still needs some kind of buffering to allow for proper packet reconstruction after transmission. The download time is another factor in deciding on a data import methodology. Although a serial interface, like JTAG, is easier to implement, in practice it is actually slower than other methods. An assembly line will see noticeable differences in program time when using a JTAG interface versus a parallel interface, for instance.

ISW Boot Code

The PnP BIOS boot code stored in the boot block of the flash memory should be able to handle remote updates by the processor as well as basic communication and reprogramming capabilities. This insures that any interruption of the reprogramming process would be recovered by resetting the flash and checking BIOS status or some reprogramming flags.

Suppose the boot code begins execution after a system reset or power-on and determines that an invalid PnP BIOS is loaded in the system. This code should begin the reprogramming process by preparing the flash device for erasure and establishing a connection to the reprogramming protocol, perhaps through an interrupt, say R_INTR. Once this connection is established, the reprogramming can commence. Some kind of valid (or complete) signal needs to be provided to the boot code to let it know that reprogramming is complete, say an R_DONE interrupt from the update protocol. Should this reprogramming be interrupted, the boot code should be able to recover by recognizing that a valid BIOS still has not been loaded and re-initiating the reprogram algorithm.

Communications Software

Whatever means is used to download the information to be programmed should guarantee accurate data transmission. The protocol employed can be a simple read-back technique or a complex error-free communications protocol. The simple read-back methodology consists of the CPU indicating to the system that it wishes to update the BIOS by asserting the R_INTR interrupt. Once the PnP BIOS acknowledges this request, it prepares the flash device for updating and transfers control to the processor. The flash memory then waits for the R_DONE interrupt. Once the reprogramming is complete, the system should resend the update code to verify the programming sequence.

PnP BIOS Reprogramming Routine

In system reprogramming of the system BIOS is one of the many advantages that flash memory brings to BIOS world. The algorithm needed to accomplish this reprogramming will vary from vendor to vendor. Rather than advocate any one method of implementation, the

flowchart in Figure 7 is provided to serve as a guide. This enables flexibility of design while insuring that all necessary components are incorporated into the reprogramming code. Even though the flowchart may not indicate so, user consideration should be embedded in the update routine, i.e., status bars, confirmation prompts, etc.

5.3.2.2 Command User Interface

The built-in Command User Interface (CUI) of the boot block (and all Intel second-generation flash devices) provides a standard interface to the internal Write State Machine (WSM) of the flash memory. Table 2 lists the commands available through the CUI and the number of cycles each requires. The CUI simplifies processor interfacing by granting full read/write functionality to the CE#, WE#, and OE# inputs. Raising V_{PP} to V_{PPH} or lowering it to V_{PPL} toggles the flash memory between read/write mode and read-only mode. When in read-only mode, only the first three commands listed in Table 2 are accessible. In read/write mode, all commands are permitted.

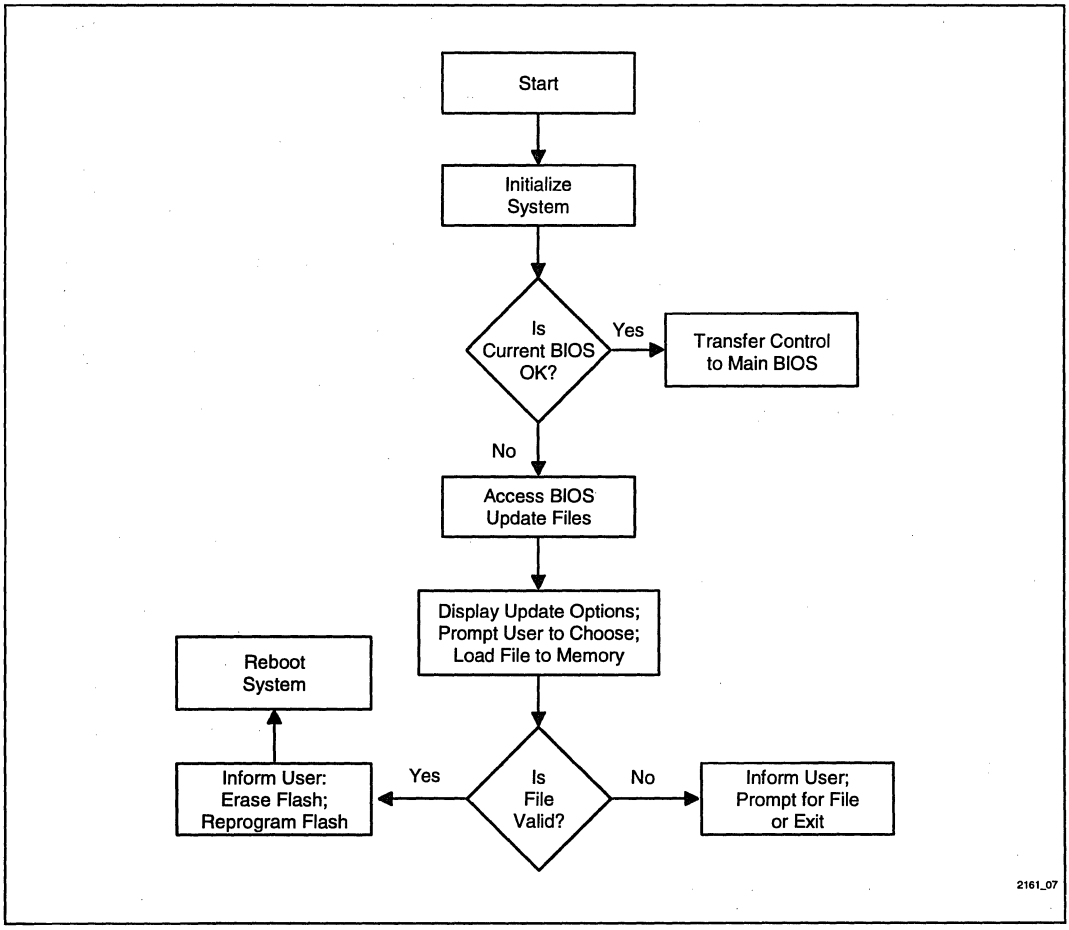


Figure 7. Flowchart for Update Algorithm. Note that although this is a fairly generic algorithm, similar flows have been implemented by BIOS vendors and OEMs since 1991.

Table 2. CUI Commands for the 28F200/002B Flash Memory

Command	# of Cycles	First Bus Cycle			Second Bus Cycle		
		Oper	Addr	Data	Oper	Addr	Data
Read Array/Reset	1	Write	X	FFh			
Intelligent Identifier	3	Write	X	90h	Read	IA	IID
Read Status Register	2	Write	X	70h	Read	X	SRD
Clear Status Register	1	Write	X	50h			
Erase Setup/Erase Confirm	2	Write	BA	20h	Write	BA	D0h
Word/Byte Write Setup/Write	2	Write	WA	40h	Write	WA	WD
Erase Suspend/Erase Resume	2	Write	X	B0h	Write	X	D0h
Alternate Word/Byte Write Setup/Write	2	Write	WA	10h	Write	WA	WD

NOTE:

To avoid excess current usage, the high order 8-bits of the data bus should be tied to V_{CC} or V_{SS} if a 16-bit wide data bus is being used (16-bit data bus only valid for the 28F200B devices)

BA = Block Address to be erased

WA = Address to be programmed

WD = Data to be programmed at address WA

IA = Identifier Address: 00h for manufacturer code; 01h for device code
(following this command, two read operations access the manufacturer and device codes)

IID = Intelligent Identifier Data

SRD = Status Register Data

Read Array/Reset (FFh): This single command points the read path at the memory array. If the processor performs a CE#/OE#-controlled read following a two-write sequence, the device will output the status register contents. If the read command is given following an Erase Setup command, the device is reset to read the array. Two sequential Read Array commands is required to place the device in read array mode after write setup. If the system leaves V_{PP} turned on during a system reset, incorporate a command register device reset into the hardware initialization routine. This is a safeguard in case the flash device is being programmed or erased when the system reset occurs.

Intelligent Identifier (90h): This commands points the output path to the Intelligent Identifier circuitry. Only values at address 0 and 1 can be read (only address A_0 is valid in this mode). All other inputs are ignored.

Read Status Register (70h): After this command, the subsequent read will output the contents of the status register, regardless of the value on the address pins. This is one of two commands that can be issued while the WSM is operating. The device automatically enters this mode following write (program) or erase completion.

Clear Status Register (50h): This command clears the program status and erase status bits of the status register. The WSM is only allowed to set these bits when it is performing one of these tasks; however, it cannot clear them. This is to allow synchronization with the processor.

Erase Setup (20h): This command prepares the flash memory for erasure and waits for the Erase Confirm command. If the next command is not the Erase Confirm command, then the program status and erase status bits of the status register are set. The device is placed in read status register mode and awaits the next command.

Erase Confirm (D0h): If the previous command is verified to be the Erase Setup command, the CUI enables the WSM, latches the address and data lines and responds only to the Read Status Register and Erase Suspend commands. While the WSM is operating, toggling the OE# input causes the device to output Status Register information.

Erase Suspend (B0h): This command is only valid when the WSM is executing an Erase command sequence. Once it has been acknowledged, the CUI instructs the WSM to suspend its current erase operation; the CUI then waits for the Read Status Register or Erase Resume commands, ignoring all other commands. When the WSM responds to the CUI that it has suspended erase operations (by setting the WSM status bit in the Status register), the Read Array command can also be recognized by the CUI. Even though the address and data latches are locked, the address lines can still drive the read path. The WSM will continue to run after the suspend.

Erase Resume (D0h): This command causes the CUI to clear the WSM status bit in the Status Register and instructs the WSM to resume the last suspended erase operation. This is only done if an Erase Suspend command was previously issued; otherwise, this command has no affect.

More information on the specific state of input pins and the actual bus definitions for these commands can be found in the datasheets.

6.0 DESIGNING FOR THE FUTURE

Due to the abundance of healthy competition in the flash market, vendors and OEMs always seek out alternative solutions for designs. Most of the discussions seem centered around three areas: programming voltage, write protection, and blocking architecture. Intel is committed to the boot block architecture and has invested considerable time and resources into proliferating the family to meet market demands.

6.1 The 5V-Only Question

Many system manufacturers are concerned with the cost, space and analog design necessary to accommodate the 12V requirement for program or erase of a flash memory. is is a justified concern, one that is answerable a number of ways. The question that must be answered, however, is not "What new changes are needed to support an on-board 12V supply?" but rather, "What is the best solution for the problem of reprogramming in-system?" Intel set out to answer the latter question—the result is the SmartVoltage (SVT) boot block products.

Simply put, SmartVoltage flash memory supports either the 12V or the 5V paradigm. Manufacturers and OEMs can now decide which method is best for their particular environment and proceed with their choice without having to purchase separate components. An OEM might have some platforms that need 12V to support highest performance write systems. Low-end systems, however, are typically more power and cost-sensitive. SmartVoltage supports both implementations, allowing the OEM to make the trade-offs necessary for the intended market. The desired program/erase speed is one of the considerations that will determine which choice is best for a particular application, since performing program/erase at 12V is faster than at 5V.

With the move from 12V / 5V to 5V / 3V on the horizon, it is easy to see how SmartVoltage technology will enable all types of system capabilities with its dual supply capability. The same SmartVoltage device can be programmed in the manufacturing flow with 12V for improved throuput. When the product is in the field and 12V is no longer available, the same SmartVoltage device adapts to the environment, enabling updates using a 5V V_{PP} supply and 3.3V or 5V V_{CC} supply. This is the type of flexibility and ease of design that SmartVoltage flash memory products will drive.

6.1.1 DESIGNING FOR SmartVoltage

SmartVoltage brings 3.3V and 5V technology to the boot block family. Although both devices are pin compatible, it is necessary to know that some previously unused pins are now being used. If you plan to migrate your designs to SmartVoltage, you need to be aware of the implications to your design.

The DU pin on the standard 2-Mb and 4-Mb boot block parts is the WP# pin on the SmartVoltage devices. Floating this pin is not good design practice if migrating to SVT is in the plan. The solution is to connect this pin to V_{CC}, GND, or a control pin as per the design. SmartVoltage devices use the WP# pin in conjunction with the RP# pin to control boot block locking and unlocking as well as array protection.

SVT Write Protection

V_{PP}	RP#	WP#	Write Protection
V_{IL}	X	X	All Blocks Locked
V_{PPLK}	V_{IL}	X	All Blocks Locked
V_{PPLK}	V_{HH}	X	All Blocks Unlocked
V_{PPLK}	V_{IH}	V_{IL}	Boot Block Locked
V_{PPLK}	V_{IH}	V_{IH}	All Blocks Unlocked

In addition to being backwards-compatible to the standard boot block products, SVT products include other features. In the event that a 12V trace is not supplied to the V_{PP} input, there is a 5V tolerant WP# pin that allows the boot block to be locked/unlocked without the need for high voltage. Only one of these locking schemes needs to be utilized; the internal circuitry is smart enough to figure out which is being used and adjusts accordingly, shifting V_{IL} and V_{IH} levels to match the supply source. Unlike other architectures, there is no need to apply 12V to some of the input pins to unlock blocks or access certain features. SmartVoltage offers uncompromised 5V-only technology. SmartVoltage is even capable of 3.3V read and will have 2.7V read capability in the future.

7.0 SUMMARY

PC users have always felt that the computer should be as simple to use as possible. To them, it was simply common sense that if they added something new to the system, it should simply work. In their opinion, if something changed in the system, the system should correct itself to adapt to this change, even if they (the user) caused this change. From their perspective, for all the money they spend on the computer, they shouldn't have to worry about how to fix it too. We have all shared some of these thoughts, but up until now, it has always seemed just beyond our reach.

Plug and Play promises to bring some of these long sought-after requests to fruition. The prospect of alleviating installation frustrations for end-users is very compelling, especially for the end-user. This concept even has appeal for manufacturers and designers alike, promising cost savings, consumer confidence in their products, and product differentiation. As it turns out, one of the ways of enabling this saving grace is using boot block flash to implement the system BIOS.

In this application note, the features of boot block flash, as it relates to the PnP BIOS, have been carefully laid out. First the needs of Plug and Play were outlined:

1. System BIOS storage
2. Nonvolatile area for system configuration database
3. Recovery code for updateability
4. Backwards compatibility to established standards

Then the pertinent issues for implementing this design were examined—from hardware lockability to generating programming voltages; from software requirements to BIOS recovery code; from implementation specific options to reprogramming algorithms. An example implementation was also provided, which included both hardware and software considerations. Even the benefits to the user as well as the manufacturer were explored. The solution to the BIOS challenges brought about by Plug and Play have been met. Boot block flash caters to all the requirements of a PnP BIOS without compromising design flexibility or creativity.

Plug and Play is more than just a term used to mean making the PC more like a Mac (what foolishness—you cannot even **add** hardware to a Mac). It is a real specification that will change the way PCs are used in the future. As the buzzword garners momentum, its implementation will become more widespread. This expansion will bring forth more people delving into the make-up of PnP and attempting to "tweak" it for differing purposes. BIOS must be able to support the current standards and conform to all the new techniques and implementations yet to come. Intel's boot block flash offers the best solution to this quiet revolution.

8.0 ADDITIONAL INFORMATION

8.1 References

For more information the concepts and ideas presented in this application note, the reader is directed to the following reference materials for further reading.

Order Number	Document
292077	AP-341: "Designing an Updateable BIOS Using Flash Memory"
292092	AP-357: "Power Supply Solutions for Flash Memory"
292098	AP-363: "Extended Flash BIOS Concepts for Portable PCs"
292148	AP-604: "Using Intel's Boot Block Flash Memory Parameter Blocks to Replace EEPROM"
290406	28F001BX-T/28F001BX-B 1M CMOS Flash Memory Datasheet
290448	28F200BX-T/B, 28F002BX-T/B 2-Mbit Boot Block Flash Memory Family Datasheet
290531	2-Mbit SmartVoltage Boot Block Flash Memory Familyt
Contact Intel/Distribution Sales Office	Plug and Play BIOS Specification v1.0A by Compaq, Phoenix, & Intel, May 1994
Contact Intel/Distribution Sales Office	Extended System Configuration Data Specification v1.02A by Compaq, Phoenix, & Intel, May 1994
Contact Intel/Distribution Sales Office	Plug and Play ISA Specification v1.0A by Microsoft and Intel, May 1994
Contact Intel/Distribution Sales Office	Plug and Play BIOS Extensions Design Guide v1.2 by Intel, May 1994

Designing with Flash Memory by Brian Dipert and Markus Levy, 1993 Annabooks Publishers

PC Interrupts by Ralf Brown and Jim Kyle, 1991 Addison-Wesley

"Transforming the PC: Plug and Play" by Tom Halfhill, September 1994 Byte Magazine

Plug and Play SCSI Specification by Adaptec, DEC, et. al., March 1994

APPENDIX A
PINOUTS, LEAD DESCRIPTIONS AND BIOS-SPECIFIC INTERRUPTS

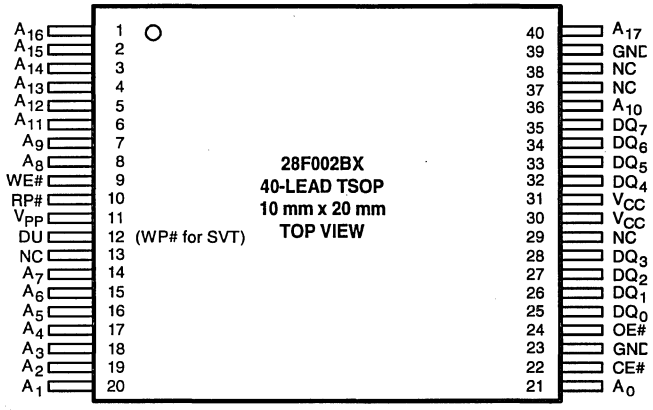


Figure 8. 40-Lead TSOP 28F002BX Flash Device Pinout

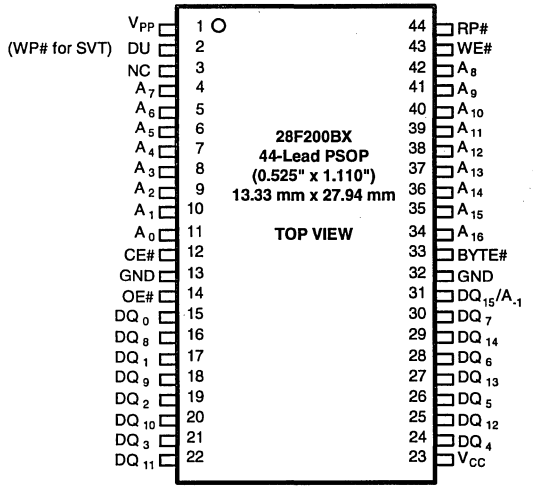


Figure 9. 44-Lead PSOP 28F002BX Flash Device Pinout

Table 3. Definition of 28F002B Pins

Symbol	Name and Function
A_0–A_{17}	**ADDRESS INPUT PINS:** Address inputs for memory addresses. Addresses are internally latched during a write cycle (on the rising edge of the WE# pulse).
A_9	**ADDRESS INPUT 9:** When A_9 is at 12V, the signature mode is accessed. In this mode, A_0 decodes between the manufacturer and device IDs.
DQ_0–DQ_7	**DATA INPUT/OUTPUT PINS:** Inputs array data on the second CE# and WE# cycle during a program command. Inputs commands to the Command User Interface when CE# and WE# are active. Data is internally latched during write and program cycles. Outputs array, intelligent identifier, and status register data. The data pins float to tri-state when the chip is deselected or outputs are disabled.
CE#	**CHIP ENABLE:** Activates the device's control logic, input buffers, decoders, and sense amplifiers. When this active low signal is at logic high, it disables the memory device and reduces power consumption to standby levels. When CE# is logic low, the memory device is enabled.
RP#	**RESET/DEEP POWER-DOWN:** When this signal is at logic high, V_{IH} (6.4V max.), it locks the boot block from program and erase. When RP# is 11.4V min., the boot block is unlocked and can be programmed or erased. When RP# is at logic low, V_{IL}, the boot block is locked, deep power-down mode is engaged and the WSM prevents all blocks from being programmed or erased. When RP# transitions from low to high, the device entered the read-array mode.
OE#	**OUTPUT ENABLE:** Gates the device's outputs through the data buffers during a read cycle. This signal is active low.
WE#	**WRITE ENABLE:** Controls writes to the Command Register and array blocks. This signal is active low. Address and data are latched on the rising edge of WE# pulse.
V_{PP}	**PROGRAM/ERASE POWER SUPPLY:** 12V ± 10%, 12V ± 5%
V_{CC}	**DEVICE POWER SUPPLY:** 5V ± 10%, 5V ± 5%
GND	**GROUND:** Ground for all internal circuitry.
NC	**NO CONNECT:** Pin may be driven or left floating.
DU	**DO NOT USE PIN:** This pin is replaced by the WP# pin on the SmartVoltage products. To insure upgrade to SVT, connect this pin to V_{CC}, GND, or a control pin as necessary.

Table 4. Full Listing of BIOS-Specific Interrupts

Interrupt Number	Function
05	Print Screen
10	Function 00h - 13h: Standard Video Functions Function 14h - 15h: LCD Functions Function 1Ah - 1Ch: VGA Functions Function 30h: 3270PC Function Function 40h - 4Fh: Hercules VGA Functions Function 6Ah - 70h: Various VGA Functions Function 71h - 73h: Tandy 2000 Functions Function 80h - 82h: DESQview v2.0x Functions Function BFh: Compaq Notebook Functions Function CCh - CDh: UltraVision BIOS Functions Function EFh: Extended Hercules Functions Function F0h - F7h: EGA RIL Functions Function FAh: EGA RIL Function Function FFh: DJ G032.EXE Extender Function
11	Get Equipment List
12	Get Memory Size
15	Function 00h - 03h: Cassette (PC & PCjr) Functions Function 04h - 05h: PS & PS2 System ABIOS Table Function 0Fh: PS/2 Format ESDI Drive Function 20h - 21h: O/S Functions Function 40h - 44h: System Functions Function 4Fh: PS/2 Keyboard Intercept Function 53h: AMIBIOS APM Functions Function 80h - 89h: O/S & System Functions Function 90h - 91h: O/S Functions Function C0h: Get system Configuration Function C1h - C2h: PS/2 BIOS Functions Function C3h - C5h: O/S & System Functions Function C6h - CFh: PS/2 Model 95 Functions Function D8h: AMIBIOS EISA Support

Table 4. Full Listing of BIOS-Specific Interrupts (Continued)

Interrupt Number	Function
16	Function 00h - 05h: Keyboard Functions Function 10h - 12h: Extended Keyboard Functions Function 12h: AT & PS/2 Extended keyboard Functions Function F0h - F4h: AMIBIOS CPU & Cache Controller Functions
17	Function 00h - 02h: Printer Functions
18	Start Cassette Basic (Genuine IBM Machines Only)
19	System Bootstrap Loader
1A	Function 00h - 0Bh: Real-Time Clock Functions Function 80h, 83h-90h: AMIBIOS Socket Functions Function 95h-A1h, AEh: AMIBIOS Socket Function Function B1h: AMIBIOS PCI Functions
1B	Control-Break Handler
1C	System Timer Tick

intel®

AP-603

APPLICATION NOTE

Symmetric Block Format Exchanging Data with FFS Systems

PETER J. TORELLI
MCD SOFTWARE MARKETING ENGINEER

January 1995

Order Number: 292155-001

1.0 INTRODUCTION

The flexibility of the Microsoft Flash File System (FFS) data structures makes it possible to arrange data in a symmetric layout on an Intel Flash PCMCIA card and still maintain FFS compatibility. Embedded applications that would like to have the exchangeability of FFS without the overhead of FFS can use this concept to their advantage. The FFS Symmetric Block Format (SBF) is a method of formatting a card to maintain FFS compatibility without using FFS in the embedded system.

Symmetric Block Formatting is not a Flash File System, but rather a method to store data on a flash PCMCIA card by placing it within the flash space an FFS would perceive as file space. This is accomplished by formatting a card with FFS data structures present for a predetermined number of files. By defining where FFS looks for the files, we can alter the data contained within that space without upsetting the FFS format. The format operates on one assumption: the size and number of data objects must be known before formatting. This is where a Symmetric Block Format differs from a complete FFS: after the card is formatted, there can be no deviation from the size and number of files that were created unless the card is erased and reformatted.

Symmetric Block Formatting fulfills the requirements of many embedded applications via a way to store data from an embedded system to a flash card, insert it into a PC running FFS, and be able to retrieve the data through FFS. The only variation, or implementation specific detail is the format, which is derived from the nature of the data.

Before attempting to explain how to create a custom format, the following sections will introduce a basic Symmetric Block Format and explain concepts behind creating a custom Symmetric Block Format.

2.0 THE SYMMETRIC FORMAT

Each time FFS writes a file to a flash PCMCIA card, it creates several data structures within the card's flash array that contain the file data. Where the structures end up in the card is anyone's guess. It varies on the previous usage of the card and between one FFS implementation to another, but it's usually scattered across the flash card in varying sized extents (see Figure 1). Only another FFS compatible driver can locate and reassemble the file correctly.

FFS data structures permit data to be placed anywhere on the card. That is the first concept the Symmetric Block Format uses to its advantage. The format places all of the file data extents in contiguous locations within the flash array, and all of the structure data into a fixed location in each block.

The first example of a format divides each block in the flash card into a fixed number of same sized pseudo-files; the total number is an even multiple of the block size. In the sample code at the end of this document, this is referred to as "Symmetric Formatting" (see Figure 2). For each block, the uppermost pseudo-file, known as a structure file, contains the structure data that enables FFS to read the card.

A card formatted to the above conditions is just a card with a number of files containing FFH. However, notice that the file data is located contiguously within each block. A simple embedded program could write data to a flash address within the card without any difficulty. It could keep writing, as long as it avoided every fourth file by skipping the number the address range containing the structure file data. In fact, it could keep writing until it ran out of space. The card would then contain the embedded system's data written to the space that FFS believes to be a file. If this card was removed from the embedded system and placed in a PC with Microsoft's FFS loaded, FFS would be able to manage the files. If a file was opened with a file editor, it would contain the embedded system's data. For systems that store x number of same-size records, this is all they would need to transfer data from an embedded system to a desktop.

As we will see, this concept can be expanded to place the file space anywhere on the card within any number of files.

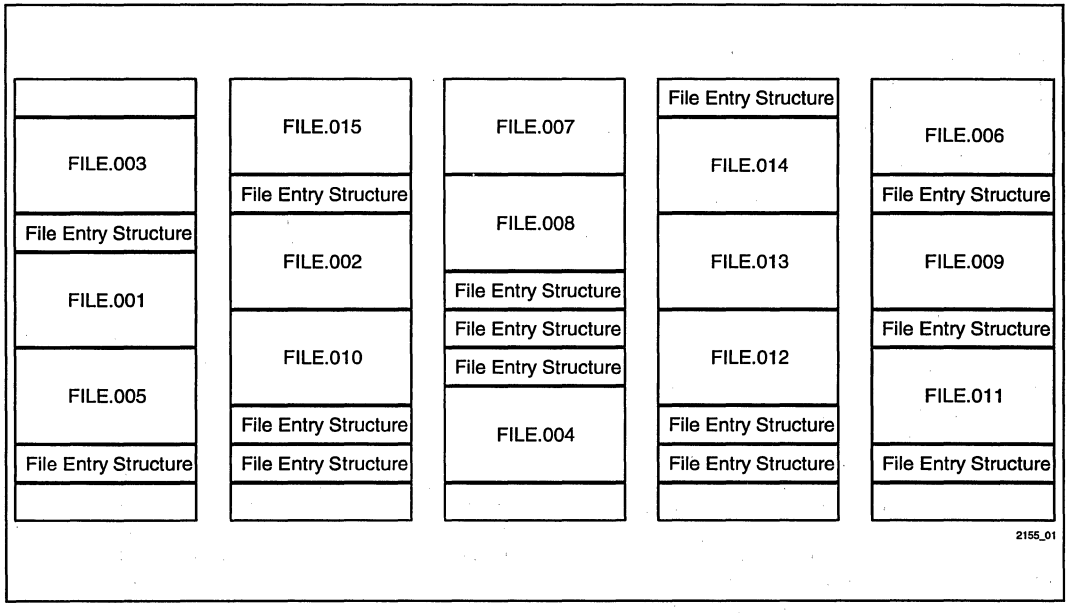


Figure 1. Normal Appearance of FFS Media

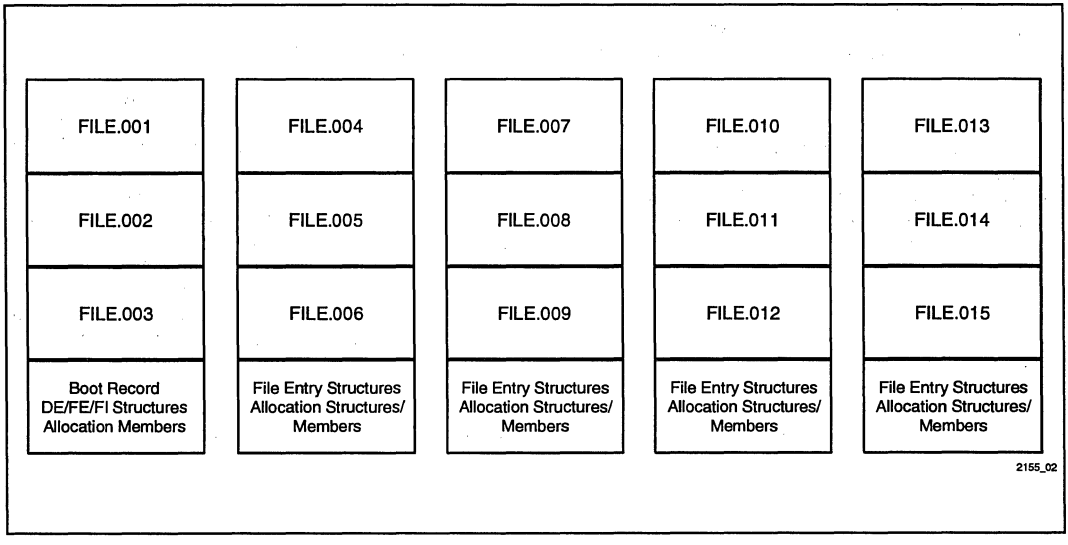


Figure 2. Symmetric FFS Format

3.0 FFS FUNDAMENTALS

FFS stores a file as a linked-list with each link a structure containing a pointer to a portion of that file's data, called an extent. By traversing the list of structures, FFS reassembles the file into the proper order. FFS stores file names and subdirectories the same way: File entries have a structure that points to the first extent of the file's data, and directory entries have a structure that points to the first file entry structure inside it. These linked-lists and data structures allow FFS to manage flash media effectively.

The Symmetric Block Format has no knowledge of what these structures mean, they merely exist on the card to manipulate FFS into treating specific regions of the card as files. The structures are explained to facilitate the creation of a custom format.

3.1 FFS Data Structures

Microsoft defines (in their FFS Media Control Structures Specification, available from Microsoft) four different data structures for storing and arranging data in the MS-FFS format: File Entry, Directory Entry, File Info, and Boot Record. The Boot Record structure contains data describing the media's geometry, as well as FFS version information. Since only one copy of the Boot Record exists, it will be excluded from all future references to the term "data structures" in this document for clarity.

The file-directory hierarchy exists in the FEDE list (File Entry Directory Entry). Typing "DIR" (or "ls" in UNIX) lists that directory's entries. In an FFS formatted flash card, the information displayed corresponds to the file and directory entries in that directory's FEDE chain (see Figure 4). All files or subdirectory entries at the same level are part of one FEDE chain, and are referred to as siblings. If a subdirectory exists in that FEDE chain, it points to another FEDE chain. The tree continues if more subdirectories exist in that FEDE chain, and so on.

Actual File Entry file data resembles the FEDE list, except each entry in the file's list is a File Info Structure. This list of File Info Structures sequentially points to the regions of the card that contain the file's data. FFS performs a read file request by locating the proper file entry, traversing its File Info chain, and returning the requested data.

The File Entry and Directory Entry structures contain three pointers: Primary, Sibling and Secondary (see Figure 3). The Sibling Pointer always points to the next entry in the same level as that structure. The Primary Pointer of a Directory Entry points to the first entry in that directory's FEDE chain. The Primary Pointer of a File Entry points to that file's File Info chain. The Secondary Pointers of both structures point to files or directories that supersede the existing structures.

```
struct FileOrDirectoryEntry {
        word        Status;
        dword       SiblingPtr;
        dword       PrimaryPtr;
        dword       SecondaryPtr;
        byte        Attributes;
        word        Time;
        word        Date;
        word        VarStructLen;
        byte        NameLen;
        byte        Name[8];
        byte        Ext[3];
};
```

```
struct FileInfoStructure {
        word        Status;
        dword       SiblingPtr;
        dword       ExtentPtr;
        dword       SecondaryPtr;
        byte        Attributes;
        word        Time;
        word        Date;
        word        VarStructLen;
        word        UncompressedExtentLen
        word        CompressedExtentLen

};
```

Figure 3. File Entry, Directory Entry and File Info Data Structures

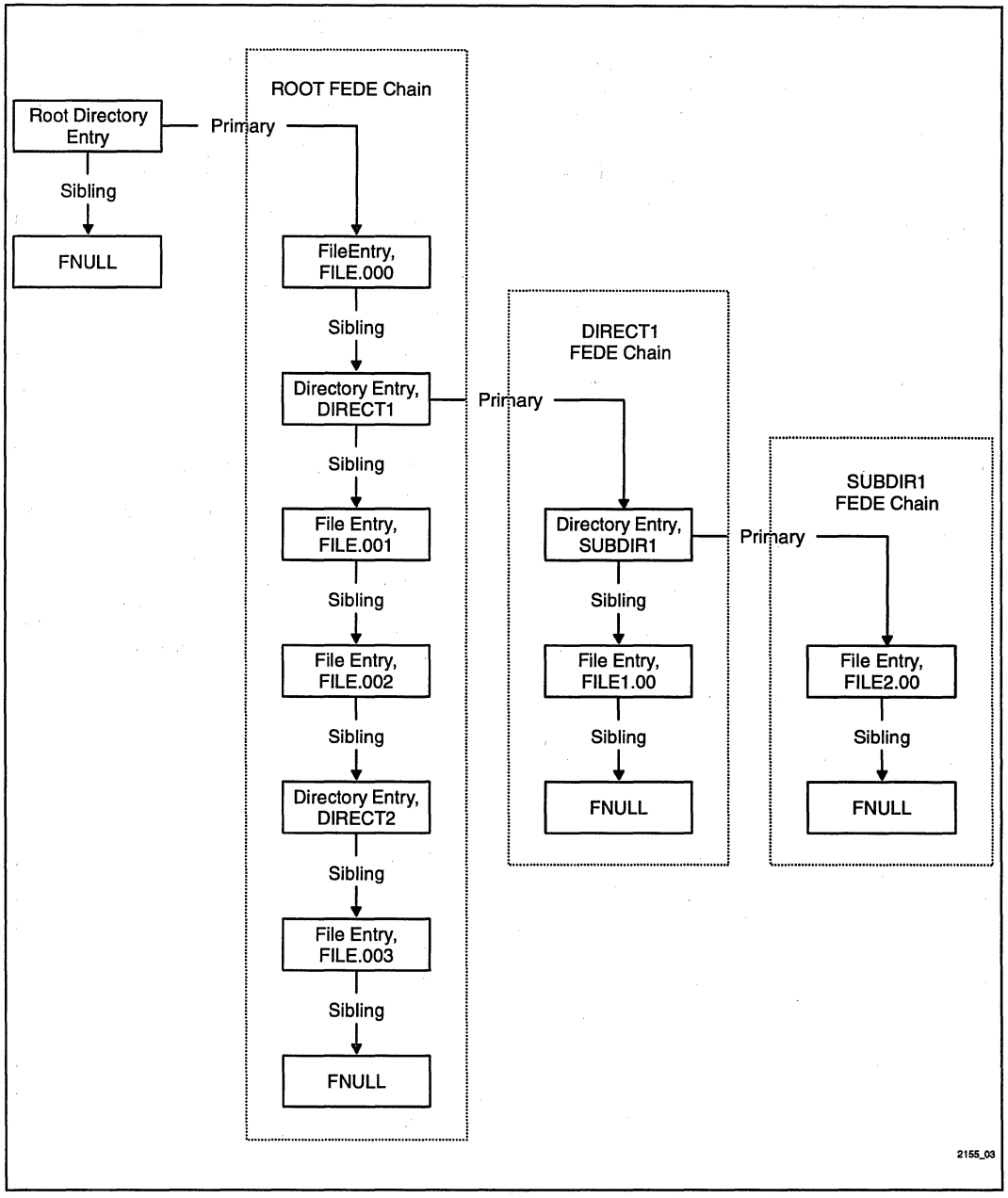


Figure 4. FEDE Chain Hierarchy

The chain of File Info Structures point to regions of the card that contain file data, called extents. The maximum size of an extent is 65,535 bytes. Like File Entries and Directory Entries, File Info Structures contain Sibling and Secondary Pointers, but the Primary Pointer has been replaced by an Extent Pointer. The Extent Pointer points to the first extent of that file's data. The Sibling Pointer addresses the next File Info Structure in the chain, and the Secondary Pointer indicates where to find updated or superseded extent data.

3.2 Allocation of Flash Media Space for Structures and File Data

The pointers in the previously explained structures don't explicitly reference the physical address of a structure or extent within a block; instead, they point to Block Allocation Members, which in turn point to the physical location of that structure. This brings us to the second detail of FFS: Allocation Members.

To maintain organization of the above data structures and file extents, FFS uses Block Allocation Members (BAM). Every File Entry, File Info Structure, Boot Record, Directory Entry or File Extent MUST have a BAM associated with it. These six-byte fields begin at the top of each erase block and grow downwards as more structures and extents are written to that block. They contain the length of the data region pointed to, the beginning offset of that data relative to address zero of that block, and a status field indicating whether or not the data being pointed to is valid or deleted. The status field allows FFS to determine which regions of the card are useful, and which can be discarded during a clean-up operation (called Reclaim).

The Block Allocation Structure (BAS) in Figure 5 exists at the topmost address of every erase block. It contains the block's logical number, whether or not it is a spare, the block's erase count, etc. The erase count field helps the FFS wear-leveling algorithm decide the priority of that blocks clean-up status. This feature benefits all flash media by insuring that the difference between the number of times adjoining blocks have been erased never exceeds a certain threshold, which has the potential to corrupt data in adjacent blocks.

```
struct BlockAllocMember {
       byte        Status;
       byte        Offset[3];
       word        Len;
};
```

```
struct BlockAllocStructure {
       dword       BootRecordPtr;
       dword       EraseCount;
       word        BlockSeq;
       word        BlockSeqChecksum;
       word        Status;
};
```

Figure 5. Block Allocation Structures

4.0 SYMMETRIC BLOCK FORMATTING EXPLAINED

All FFS formatted cards must contain the following information:

- One Boot Record + its BAM
- One Root Directory Structure + its BAM
- A BAS in every block with a unique logical block number

These three requirements are necessary for all formats.

When FFS writes a file to a flash card, it creates the following structures:

- File Entry (appended to FEDE Chain) + its BAM
- File Info + its BAM
- Extent + its BAM
- + Additional File Info/Extent Pairs, depending on the size of the File Entry

Modeling a custom format is just a matter of arranging the above structures to inhabit a reserved location in each block. Since 64 KB – 1 is the largest size of an FFS extent, files larger than that need multiple File Info structures.

To help in the visualization of what needs to be written to the card, observe Figure 6.

Physical Block One		All other Physical Blocks (e.g., Block #2)	
00000H to		20000H to	
1BFFFH	- Pseudo file data space	3BFFF	- Pseudo file data space
:	(extents) for first 7 16-Kbyte	:	(extents) for next 7 files.
:	files.	:	
:		:	
: ;Start Structure File space here...		: ;Start Structure File space here...	
1C000H	- Boot Record	3C000H	- File Entry (FILE.008)
1C01AH	- ROOT Directory Entry	3C021H	- File Info (FILE.008)
1C03BH	- Volume Label	3C03AH	- File Entry (FILE.009)
1C05CH	- File Entry (FILE.001)	:	
1C07DH	- File Info (FILE.001)	:	
1C096H	- File Entry (FILE.002)	:	
:		:	
1C1B8H	- File Entry (FILE.007)	3C15CH	- File Entry (FILE.014)
1C1D9H	- File Info (FILE.007)	3C17DH	- File Info (FILE.014)
:		:	
:		:	
1C1F2H to 1FF61H - Not Used		*3C196H to 3FF73H - Not Used*	
:		:	
:		:	
1FF68H	- BAM (Extent for FILE.007)	3FF7AH	- BAM (Extent for FILE.014)
:		:	
1FFCEH	- BAM (FE FILE.002)	:	
1FFD4H	- BAM (Extent for FILE.001)	:	
1FFDAH	- BAM (FI FILE.001)	:	
1FFE0H	- BAM (FE FILE.001)	3FFE0H	- BAM (FE FILE.009)
1FFE6H	- BAM (Volume Label)	3FFE6H	- BAM (Extent for FILE.008)
1FFECH	- BAM (ROOT Directory)	3FFECH	- BAM (FI FILE.008)
1FFF2H	- BAM (Boot Record)	3FFF2H	- BAM (FE FILE.008)
20000H	- BAS (Logical Block 0)	40000H	- BAS (Logical Block 1...)

NOTE:
This layout assumes a flash card with 128-Kbyte erases blocks. The format divides each block by eight, resulting in seven 16-Kbyte pseudo-files and a 16-Kbyte space for one data structure.

Figure 6. Symmetric Format Example

Thus, a 2-Mbyte card would appear to FFS as a 2-MB drive with 112 16-Kbyte files, and to the embedded system as a storage space capable of holding 128 files, but every eighth file should be skipped.

The code in Appendix A provides three procedures for formatting the card: InitialFormat, DoSymmetric and DoEntireCard. InitialFormat writes a Boot Record, ROOT Directory Entry and Volume Label to the card's block zero. DoSymmetric performs the symmetric format described earlier. DoEntireCard is another variation on the Symmetric Block Format. This procedure formats the entire card as one large file. The reference code is described in the last section of this application note.

5.0 EMBEDDED SYSTEM REQUIREMENTS

The difference between a Symmetric Block Format in an embedded system and a full FFS implementation lies in the amount of FFS capability built into the embedded system. Deciding how much functionality to build into the embedded system can be associated with a "Kbyte per level of functionality" cost. Adding more FFS-like features requires larger embedded code. The following sections describe how to implement four fundamental FFS concepts based on MS-FFS data structures.

5.1 Where to Put the Formatter

Regardless of the amount of functionality desired, the key to the SBF lies in the format of the card. Formatting requires writing all of the data and allocation structures to the card before it's used. A special formatter located either on a PC or on the embedded system itself writes the structure file data to the card.

Using the formatter outside the embedded system and on the PC shrinks the size of the embedded code, but requires the extra handling of an external program.

Designing the formatter into the embedded system increases the size of the embedded code, but insures that formatting and downloading can be done at the same location.

Regardless of where the formatter is located, it must perform the same function: writing the necessary structures to convince FFS that files exist on the card. If all of the format data is known, a simple hex dump to the card at the proper address in each block can be used. This way, the computer doesn't have to calculate where to place all of the structures, it merely copies the numbers from internal storage to the card. However, having the format data in hardware makes it non-updateable. The other way to format requires a utility that prompts the user for the specific number of files and their size. This utility then calculates how to pack the structures into a small enough space in the block and displays to the user where the forbidden structure file locations exist.

5.2 Writing to the Formatted Card

Since all of the pseudo-file space created by the formatter exists in a contiguous region of flash, the embedded system needs to know how to avoid this forbidden region. This can be done two ways, depending on the read/write mechanism of the embedded system.

The first method employs a protection algorithm that's inserted into the embedded system's normal write algorithm. Typically, a generic write algorithm would look something like Figure 7. When the embedded system decides to write a byte, it obtains the current address, calls the flash programming algorithm, and then increments the address. The protection algorithm, shown in Figure 8 avoids the region of space at the top of the block reserved for the structure file. Since the size of the structure file may vary, its location is determined by subtracting the size of the file from the block size. Reference code for these algorithms are located in Appendix A.

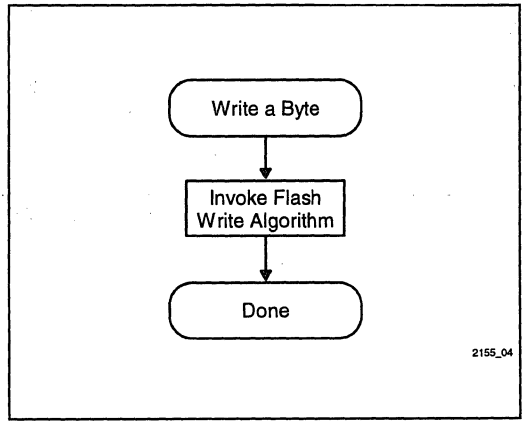


Figure 7. Conventional Byte-Write Process

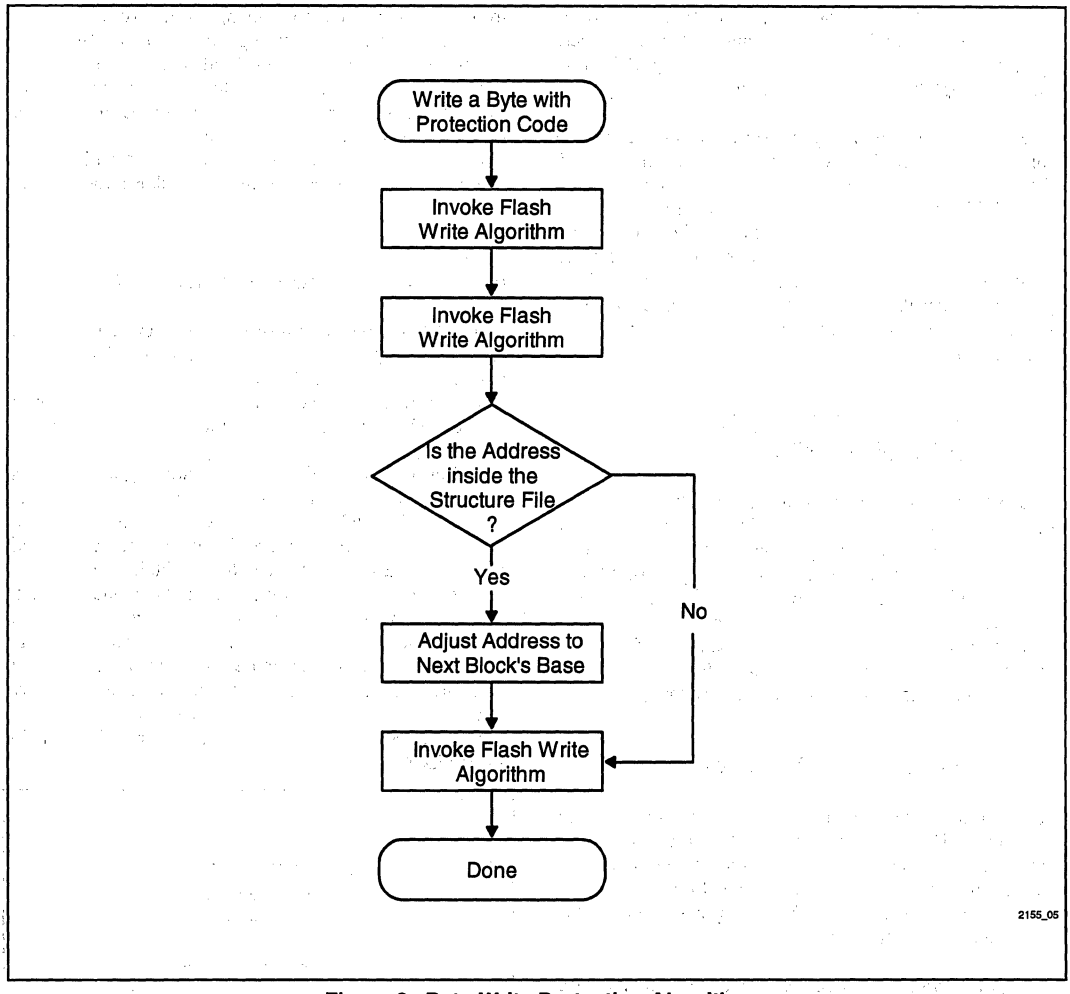


Figure 8. Byte-Write Protection Algorithm

```
    ; Values defined before compile time.
    define BLOCK_SIZE    = block size
    define BLOCK_BOUND   = 2's complement of block size
    define IN_BLOCK      = block size - 1
    define CHECK     = block size - structure file size

    Assuming ebx         = (32-bit) address in the card about to be written to

    :
    :
protection_check:

    mov eax, ebx     ; Grab the current address
    and eax, IN_BLOCK        ; Truncate the address to a single block
    sub eax, CHECK       ; Subtract the distance to the structure file

    js  write_flash      ; If accumulator goes negative, we're safe

    and ebx, BLOCK_BOUND; Otherwise, mask the address to a block boundary
    add ebx, BLOCK_SIZE ; and increment to the next block

write_flash:

    callwrite_flash_byte
    :

    :
```

Figure 9. Protection Code for Byte-by-Byte Writing

Another way to write to the formatted card involves using equally sized records. For example, a data logging device that needs to write a 4-KB record every so often can speed up the write process by eliminating the need to use protection code. Assuming the card has been formatted to 4-KB file sizes, the embedded system need only be aware of what file number is being written. A 128-Kbyte erase block partitioned into 4-KB pseudo-files would have 32 file spaces, with every 32nd space being a structure file. By avoiding the 32nd space and writing 4 KB at a time, the embedded system would successfully write to the card.

Here is the code for file-oriented data writing on a record-by-record basis. Assuming 4-Kbyte records:

```
    define FILES_PER_BLOCK              = Number of ( PseudoFiles + Structure Files ) /
                                          block
    define FILE_SIZE
    cx                                  = Next available filespace's number

    :
    :
find_flash_start_address:
    mov ax, FILE_SIZE                   ; Put the pseudo-file size in AX
    mul cx                              ; Multiply AX by the current filespace's number
    callwrite_4kbytes_to_card           ; Call the routine to write 4K to the found address

increment_to_next_file:
    inc cx                              ; Increment to next valid filespace
    mov ax, cx                          ; Move this number into AX
    mov bx, FILES_PER_BLOCK             ; Prepare to do a modulus operation
    div bx                              ; Divide number of files by the current file number
    or  dx, dx                          ; (Remainder in DX)
    jnz done                            ; Was the remainder Zero?
    inc cx                              ; Was the remainder Zero?
done:                                   ; Then skip this filespace
    :
    :
```

Figure 9. Record-by-Record File-Oriented Data Writing

In most instances, this is the maximum functionality the embedded system would need: insert a freshly formatted card into the embedded machine, let it dump its data, and now the card can be inserted and read by any PC running FFS.

6.0 READING FROM THE FORMATTED CARD

6.1 Embedded System to an FFS-Based PC

The entire process of writing data to the formatted card enables any FFS-based PC the ability to read the data off the card. This case has already been defined.

6.2 Embedded System to Same Type of Embedded System

Passing data between two of the same embedded systems is quite simple, assuming they both expect the same format. The only way an embedded system can read data from the card is if it knows where the data is. A file pointer that references the start of a pseudo file that the embedded system wishes to reference can be obtained from the format information that the system has already been programmed to understand. If the data is a binary image (i.e., a large graphic picture taking up the entire card), the same protection algorithm must be used for reading as well as writing in order to recreate the file.

6.3 FFS-Based PC to an Embedded System

Since the write algorithm operates on the principle of permanent structure file space, a full FFS cannot write to a card. If the card is inserted into a PC running FFS, and a file is written to it, the format will be lost. Therefore, there is no direct way to write data from a PC to the formatted card using FFS. If a designer wishes to place data into the Symmetric Block Format, it must be done with a separate PCMCIA hex dump/editor utility.

7.0 REFERENCE CODE FOR DOS-BASED PCs

When compiled, the reference code in this section will format the card to either of the following options: A Symmetric Format with a user-definable number of files, and an Entire Card Format which uses the entire space of the card as one large file. The formatter uses the DOS Generic IOCTL interface supplied by a PCMCIA Compliant Memory Card Device Driver. Intel's iCardrv1, iCard2.9 and iCardrv3 all provide the Generic IOCTL services used here, as well as Card Drivers by Phoenix, AMI, SystemSoft and Award. These drivers must be loaded in CONFIG.SYS for this program to work properly. If iCardrv1 or 2.9 is being used, this program will support Intel Series 1, 2 and 2+ Flash memory cards. Otherwise, the proper technology driver must be loaded.

Although the low-level portion of this program is based on a DOS interface, these routines may be removed and replaced with other platform-specific low-level functions. The formatting concepts remain the same for all FFS based systems.

The first thing the program does upon invocation is to obtain the card's geometry: size, number of blocks, size of block, etc. This information will be used later to calculate how many structures will be needed to format the card. Next, the program prompts the user for a format selection. At this point, the user may select either a symmetric format or an entire card format. Selecting a symmetric format further queries the user as to the number of files per block. The program determines this range dynamically by computing the maximum and minimum amount of files that may be written to a block. Note, that this calculation is made to keep the structure file data within one file space. Depending on the implementation, the structure file space can be any size: it's up to the implementation to tailor the embedded system to the format. The second option: Entire Card will format the entire card as one large file. Before the format is installed, the program will ask whether or not the card needs to be erased.

The three reference code procedures of interest are: InitialFormat(), DoSymmetric() and DoEntireCard(). InitialFormat() places on the card the basic elements provided by an FFS Format: a Boot Record, ROOT Directory entry and a Volume Label. In reality, the Volume Label is optional and may be excluded from the format, but it's included as an additional reference. Two important variables are initialized by this procedure: BottomPtr and BAMptr. BottomPtr references the base address of the structure file within that block. If the block size is 128 KB and the structure file is 8 KB, the BottomPtr would be 128 KB–8 KB or 1E000H. Each time a structure is added to the file, the BottomPtr increases by the size of the structure. A 25-byte File Info structure increases the BottomPtr by 25 bytes. Each time the formatter enters a new block, it resets the BottomPtr. The BAMptr always references the currently available BAM. Each time a new structure is written to the structure file, its BAM should be chosen using the BAMptr and then written with the function WriteBAM(). WriteBAM() calculates the correct address within the card to write the BAM. All the user needs to specify is the 32-bit BAM value and the structure containing the BAM's data. Each time a BAM is used, BAMptr is incremented. If the next structure needs to be placed in the next block, BAMptr is incremented by 10000H.

DoSymmetric() takes the BAMptr and BottomPtr as well as the user-specified PseudoFileCount values and formats each block with x number of files. DoSymmetric() iterates through each block in the card (except the spare) and writes a File Entry and File Info structure for each file. The result of this format is a FEDE chain of File Entries.

DoEntireCard() writes a single File Entry to block zero and multiple 32-KB file extents to each block, thus encapsulating the entire flash card as one file. The procedure does not use the PseudoFile values; instead, it uses the InfoCount value calculated by the program before calling this function. InfoCount indicates the number of File Info Structures required in each block based on the size of the block.

APPENDIX A
FFS FILE IMAGE FORMATTER

```
/*****************************************************************************
*
*   FFS File Image Formatter
*
*   Copyright (c) Intel Corporation, 1994
*
*   Peter J Torelli/James R. Massoni
*
*   Revision History:
*
*       1.00    First release: symmetric formatting only (PJT,6-17-94)
*       1.01    Added entire card capability.
*       1.02    Modified program structure (JRM, 8-26-94)
*
*   Future modifications:
*
*       1. Better error handling.
*       2. Check for Card Driver
*       3. Query IOCTLs first
*
*   Environment:
*
*       Borland C/C++ 3.1 IDE, Example: BCC -mh SBM.C
*       Huge Memory Model
*       Force pointers as "far" for DWORD compatibility
*
*****************************************************************************
    PAGE
*****************************************************************************/

#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "sbm.h"
#include "sbm_msg.h"

/*
** Global Variables
*/
```

```
DWORD  InfoCount=0UL;        /* Number of FileInfos for Entire Format */
DWORD  ResidualSpace=0UL;       /* Non 32K extent for Entire Card Format */
DWORD  BottomPtr=0UL;        /* Bottom of FFS Structure File */
DWORD  BAMptr=0UL;           /* Incremental BAM pointer */
DWORD  StructureFileSize=0UL;       /* Structure File Size */
DWORD  PseudoFileSize=0UL;      /* PseudoFile Size */
WORD   PseudoFileCnt=0;      /* Number of Pseudo Files */
BYTE   DriveNumber=0;        /* Current Drive number */
CardInfo  Card;           /* Current Card's Geometry */

union IOCTLDataPkt DataPkt;
union IOCTLDataPkt *pDataPkt=&DataPkt;

/*
**  Function Prototypes
*/

WORD GenericIOCTL ( BYTE, union IOCTLDataPkt far *, BYTE );
void CommonWrite ( DWORD, BYTE *, WORD );
void CommonRead ( DWORD, BYTE *, WORD );
WORD Log2Phy ( WORD lblock );
void WriteBAM ( DWORD, BlockAllocMember );
void WriteBAS ( WORD, WORD, WORD );
void DoSymmetric ( void );
void DoEntireCard ( void );
void InitialFormat ( void );


/*****************************************************************************
   PAGE
*****************************************************************************
*
*   Title:     Main
*
*   Description:  Entry point for Symmetric Block Manager.  This utility
*              is used to format a flash PCMCIA card symmetrically so
*              data may be written to the card at pre-known addresses.
*
*****************************************************************************/

void main ( int argc, char *argv[] )
{
    /* Calculation Values. */
    DWORD  CompareSize=0UL;
    WORD   FileOverhead=0;
    WORD   BaseOverhead=0;
    WORD   TotalOverhead=0;
    WORD   MinFiles=0;
    WORD   MaxFiles=0;

    /* User Input. */
    BYTE   format_type=0;
    BYTE   keypress=0;

    /* Initial Format fields. */
    WORD    flags=0;
```

```
WORD    block_ctr=0;

fprintf( stdout, LOGON_MSG, REVISION );

if ( argc < 2 )
{
    fprintf( stderr, SPECIFY_MSG );
    fprintf( stderr, USAGE_MSG );
    exit( 1 );
}
else if ( argc > 2 )
{
    fprintf( stderr, TOO_MANY_MSG );
    fprintf( stderr, USAGE_MSG );
    exit( 1 );
}
if ( ( ( DriveNumber=toupper(argv[1][0])-'A'+1 ) < 1 ) ||
     ( DriveNumber > 26 ) )
{
    fprintf( stderr, INVL_DRV_LET_MSG );
    fprintf( stderr, USAGE_MSG );
    exit( 1 );
}

/* Get Card Size Metrics. */
fprintf( stdout, OBTAINING_MSG );
pDataPkt->MediaInfo.Len=sizeof( pDataPkt->MediaInfo );
GenericIOCTL( DriveNumber, pDataPkt, MEDIA_INFO );

Card.BlockSize=pDataPkt->MediaInfo.BlockSize;
Card.Size=pDataPkt->MediaInfo.PartSize;
Card.NumBlocks=(DWORD) Card.Size / (DWORD) Card.BlockSize;

fprintf( stdout, CHOOSE_MSG );
format_type=toupper(getche());
fprintf( stdout, DBL_CRLF_MSG );

switch( format_type ) {
    case SYMMETRIC: {

        /* Calculate min/max number of symmetric files. */
        MinFiles = (DWORD) Card.BlockSize / 0x8000;
        MaxFiles = 0;

        /* Determine the base overhead. */
        BaseOverhead = sizeof( BlockAllocStruct ) +
               sizeof( BootRecord ) +
               sizeof( BlockAllocMember ) +
               sizeof( DirectoryEntry ) +
               sizeof( BlockAllocMember ) +
               sizeof( FileEntry ) +
               sizeof( BlockAllocMember );

        /* Determine the amount of overhead per file. */
        FileOverhead = sizeof( FileEntry ) +
               sizeof( BlockAllocMember ) +
```

```
            sizeof( FileInfo ) +
            sizeof( BlockAllocMember ) +
            sizeof( BlockAllocMember );

    /* Do a max/min calculation to determine the number
         of files per block based on block size. */
    do {
        MaxFiles++;
        TotalOverhead = ( MaxFiles * FileOverhead ) +
        BaseOverhead;
        CompareSize =
        DWORD)((DWORD)Card.BlockSize)/((DWORD)MaxFiles);
    } while( CompareSize > TotalOverhead );
    MaxFiles--;

    /* Offer the user a range to choose from. */
    fprintf( stdout, NUM_PFILES_MSG, MinFiles, MaxFiles );
    scanf( "%d", &PseudoFileCnt );
    fflush( stdin );
    fprintf( stdout, CRLF_MSG );

    /* Calculate the Pseudo File Size. */
    PseudoFileSize=(DWORD) Card.BlockSize / PseudoFileCnt;
    StructureFileSize=PseudoFileSize;
    BottomPtr=Card.BlockSize-StructureFileSize;
    break;
}

case ENTIRE: {

    /* Calculate the Base Overhead */
    BaseOverhead =sizeof( BlockAllocStruct ) +
            sizeof( BootRecord ) + sizeof( BlockAllocMember ) +
            sizeof( DirectoryEntry ) +
            sizeof( BlockAllocMember ) +
            sizeof( FileEntry ) + sizeof( BlockAllocMember ) +
            sizeof( FileEntry ) + sizeof( BlockAllocMember );

    /* Determine how many 32k extents fill a block.*/
    InfoCount = (DWORD) Card.BlockSize / 0x8000;

    /* Determine how many File Infos would be needed to store those
    **   extents.  Each FI has an FI struct, FI BAM and
    **   Extent BAM. */
    FileOverhead =InfoCount * ( sizeof( BlockAllocMember ) +
            sizeof( FileInfo ) +
            sizeof( BlockAllocMember ) );

    /* Combine the two overheads to determine the structure file
    **   size. */
    StructureFileSize=BaseOverhead + FileOverhead;

    /* The last extent of each block is some # less than 32k. */
    ResidualSpace=0x8000-StructureFileSize;

    /* Adjust the bottom pointer. */
```

```
        BottomPtr=Card.BlockSize-StructureFileSize;

        /* The file is the size of the card minus all structure
        **   files. */
        PseudoFileSize=BottomPtr * (Card.NumBlocks-1);
        PseudoFileCnt=1;
        break;
    }
    default: {
        fprintf( stdout, INVL_SELECT_MSG );
        exit(1);
    }
}

fprintf( stdout, ERASE_PROMPT_MSG );
keypress=toupper( getche() );
fprintf( stdout, CRLF_MSG );
if ( keypress == YES )
{
    fprintf( stdout, PLEASE_WAIT_MSG );
    GenericIOCTL( DriveNumber, pDataPkt, ERASE_DRIVE );
    fprintf( stdout, DONE_MSG );
}
fprintf( stdout, CRLF_MSG );

/* Write BASs. */
block_ctr=0;
fprintf( stdout, WRITE_BAS_MSG, block_ctr );
while( block_ctr < Card.NumBlocks ) {
    if( block_ctr == 0 ) {
        flags |= BOOT_RECORD;
    }
    if( block_ctr+1 == Card.NumBlocks ) {
        flags |= SPARE_BLOCK;
    }
    WriteBAS( block_ctr, block_ctr, flags );
    flags=0;
    block_ctr++;
    fprintf( stdout,"%c%c%c%c%03Xh", 8,8,8,8,block_ctr );
}
fprintf( stdout, DOT_DONE_MSG );

/* Lay down initial format. */
InitialFormat();

/* Do the specific format. */
switch(format_type) {
    case SYMMETRIC: {
        DoSymmetric();

        /* Reset the drive so our format info is recognized by DOS. */
        fprintf( stdout, RESET_MEDIA_MSG );
        GenericIOCTL( DriveNumber, pDataPkt, MEDIA_CHANGE );
        fprintf( stdout, DOT_DONE_MSG );

        /* Display the format information. */
```

```
            fprintf( stdout, SYMM_INFO_MSG,
            ((DWORD) PseudoFileCnt * (Card.NumBlocks-1))-
            (Card.NumBlocks-1 ),
                   Card.NumBlocks-1, ((DWORD) PseudoFileCnt * (Card.NumBlocks-1)),
            (DWORD) PseudoFileSize/1024, Card.BlockSize-StructureFileSize,
            Card.BlockSize-1 );
            break;
        }
        case ENTIRE: {
            DoEntireCard();

            /* Reset the drive so our format info is recognized by DOS. */
            fprintf( stdout, RESET_MEDIA_MSG );
            GenericIOCTL( DriveNumber, pDataPkt, MEDIA_CHANGE );
            fprintf( stdout, DOT_DONE_MSG );

            /* Display the format information. */
            fprintf( stdout, ENTIRE_INFO_MSG, ((DWORD) PseudoFileCnt),
            Card.NumBlocks-1, (DWORD) PseudoFileSize/1024,
            Card.BlockSize-StructureFileSize, Card.BlockSize-1 );
            break;
        }
        default: break;
    }
}

/*****************************************************************************
    PAGE
*****************************************************************************
*
*   Title:     GenericIOCTL
*
*   Description:  This procedure invokes a DOS generic IOCTL 440Dh.
*
*****************************************************************************/

WORD GenericIOCTL ( BYTE DriveNumber, union IOCTLDataPkt far *pPkt, BYTE Code )
{
    union   REGS    inregs, outregs;
    struct  SREGS   sregs;
    WORD    ReturnAX=0;

    inregs.x.ax=GENERIC_IOCTL;
    inregs.h.bl=DriveNumber;
    inregs.h.bh=0;
    inregs.h.ch=DISK_DRIVE;
    inregs.h.cl=Code;
    inregs.x.dx=FP_OFF(pPkt);
    sregs.ds=FP_SEG(pPkt);
    ReturnAX=intdosx(&inregs,&outregs,&sregs);
    if( outregs.x.cflag )
        fprintf( stderr, ERROR_MSG );
    return ( outregs.x.cflag ? ReturnAX : 0 );
}

/*****************************************************************************
```

```
    PAGE
*****************************************************************************
*
*   Title:      CommonWrite
*
*   Description:  Common memory write procedure.
*
*****************************************************************************/

void CommonWrite ( DWORD to, BYTE *data, WORD length )
{
    pDataPkt->CommonMemWrite.Len=length;
    pDataPkt->CommonMemWrite.Offset=to;
    pDataPkt->CommonMemWrite.PtrBuffer=data;
    GenericIOCTL( DriveNumber, pDataPkt, COMM_WRITE );
}

/*****************************************************************************
    PAGE
*****************************************************************************
*
*   Title:      CommonRead
*
*   Description:  Common memory read procedure.
*
*****************************************************************************/

void CommonRead ( DWORD to, BYTE *data, WORD length )
{
    pDataPkt->CommonMemRead.Len=length;
    pDataPkt->CommonMemRead.Offset=to;
    pDataPkt->CommonMemRead.PtrBuffer=data;
    GenericIOCTL( DriveNumber, pDataPkt, COMM_READ );
}

/*****************************************************************************
    PAGE
*****************************************************************************
*
*   Title:      Log2Phy
*
*   Description:  This procedure converts a logical block number to a
*         physical one by reading each BAS sequence number.
*
*****************************************************************************/

WORD Log2Phy ( WORD lblock )
{
    BlockAllocStruct CurBAS;
    DWORD address=0UL;
    WORD   block=1;

    /* Mustn't try to read a block that doesn't exist. */
    if(lblock>=Card.NumBlocks) return (ERROR);

    /* Read each BAS. */
```

```
    while( block <= Card.NumBlocks )
    {
        address=(DWORD) block * (DWORD) Card.BlockSize;
        address-=sizeof( BlockAllocStruct );
        CommonRead( address, pBYTE &CurBAS, sizeof( BlockAllocStruct ) );
        if( CurBAS.BlockSeq == lblock ) return block;
        block++;
    }
    return (ERROR);
}

/*****************************************************************************
    PAGE
*****************************************************************************
*
*   Title:     WriteBAM
*
*   Description:  This procedure writes the current BAM to the location
*         specified by the BAM pointer.  The actual physical
*         address of the BAM is derived from the BAM pointer.
*
*****************************************************************************/

void WriteBAM ( DWORD BAMptr, BlockAllocMember CurBAM )
{
    WORD     block;
    DWORD  address;

    block=Log2Phy( (DWORD)BAMptr >> 16 );
    address=block*Card.BlockSize;
    address-=sizeof( BlockAllocStruct );
    address-=( ( ( BAMptr & 0xFFFF ) + 1 ) * sizeof( BlockAllocMember ) );
    CommonWrite( address, pBYTE &CurBAM, sizeof( BlockAllocMember ) );
}

/*****************************************************************************
    PAGE
*****************************************************************************
*
*   Title:     WriteBAS
*
*   Description:  This procedure writes a BAS to a physical block.  The
*         lblock value assigns the physical block it's unique logical
*         number.  The flags field determines whether or not the
*         block is spare, and if the boot record is present.
*
*****************************************************************************/

void WriteBAS ( WORD LogBlock, WORD PhyBlock, WORD Flags )
{
    DWORD  CurAddress=0UL;
    BlockAllocStruct  CurBAS;

    CurBAS.BootRecordPtr=FNULL;
    CurBAS.EraseCount=69;
    CurBAS.BlockSeq=LogBlock;
```

```
    CurBAS.BlockSeqChecksum=CurBAS.BlockSeq^0xFFFF;
    CurBAS.Status=0xC3FF;

    /* Make Spare Block Modifications */
    if( Flags & SPARE_BLOCK )
    {
        CurBAS.BlockSeq=0xFFFF;
        CurBAS.BlockSeqChecksum=0xFFFF;
        CurBAS.Status=0xFFF3;
    }

    /* Make Boot Record Modifications. */
    if( Flags & BOOT_RECORD )
    {
        CurBAS.BootRecordPtr=((DWORD)LogBlock<<16)+0;
        CurBAS.Status=0xC3FE;
    }

    /* Calculate where to write the BAS. */
    CurAddress=((( PhyBlock+1 ) * Card.BlockSize )-sizeof( BlockAllocStruct ));
    CommonWrite( CurAddress, pBYTE &CurBAS, sizeof( BlockAllocStruct ) );
}

/*****************************************************************************
    PAGE
*****************************************************************************
*
*   Title:      DoSymmetric
*
*   Description:  Symmetric formatter procedure.  It writes x number of
*          pseudo files per block as specified by the user in main().
*
*****************************************************************************/

void DoSymmetric ( void )
{
    DWORD   pfile_num=0;          /* Current File Number. */
    WORD    block_ctr=0;             /* Current block. */
    WORD    file_ctr=0;           /* Current File Counter. */
    DWORD   CurrentBase=0UL;      /* Base address of current block. */
    BYTE    tempname[8];          /* Temporary extension field. */
    BlockAllocMember  BAM;
    FileEntry  FE;
    FileInfo   FI;

    fprintf( stdout, WRITE_PSEUDO_MSG, pfile_num );

    /* Begin writing the pseudo files for each block. */
    while( block_ctr < (Card.NumBlocks-1) )
    {
        file_ctr=0;

        /* Calculate the physical base of the block. */
        CurrentBase=(DWORD) Card.BlockSize * block_ctr;

        /* The BottomPtr of the structure file is initially set from
```

```
**    the InitialFormat procedure. */
if( block_ctr > 0 )
{
    /* If this isn't block zero, then recalculate the new
    BottomPtr. */
    BottomPtr=(DWORD) Card.BlockSize-PseudoFileSize;
    BAMptr=(DWORD) block_ctr << 16;
}

/* For each block, write x number of PseudoFiles. */
while( file_ctr < (PseudoFileCnt - 1) )
{
    /* Write FileEntry BAMs. */
    BAM.Status=0x3F;
    BAM.Offset[0]=BottomPtr&0xFF;
    BAM.Offset[1]=(BottomPtr>>8)&0xFF;
    BAM.Offset[2]=(BottomPtr>>16)&0xFF;
    BAM.Len=sizeof( FileEntry );
    WriteBAM( BAMptr, BAM );
    BAMptr++;

    /* Write File Entry. */
    FE.Status=0x00A7;

    /* Last PseudoFile? */
    if((file_ctr+1)==(PseudoFileCnt-1))
    {
        /* Last block? */
        if( block_ctr == ( Card.NumBlocks - 2 ) )
        {
            FE.SiblingPtr=FNULL;
            FE.Status=0x00E7;
        }
        else
        {
            FE.SiblingPtr=( (DWORD) ( block_ctr + 1 )
            << 16 );
        }
    }
    else
    {
        FE.SiblingPtr=BAMptr+2;
    }
    FE.PrimaryPtr=BAMptr;
    FE.SecondaryPtr=FNULL;
    FE.Attributes=0x00;
    FE.Time=0x8800;
    FE.Date=0x02F4;
    FE.VarStructLen=0;
    FE.NameLen=0x0B;
    sprintf( tempname, "%08ld", pfile_num );
    strncpy( FE.Name, tempname, 8 );
    strncpy( FE.Ext, "BIN", 3 );
    CommonWrite( BottomPtr+CurrentBase, pBYTE &FE,
    sizeof( FileEntry ) );
    BottomPtr+=sizeof( FileEntry );
```

```
            /* Write FileInfo BAMs. */
            BAM.Status=0x3F;
            BAM.Offset[0]=BottomPtr & 0xFF;
            BAM.Offset[1]=( BottomPtr >> 8 ) & 0xFF;
            BAM.Offset[2]=( BottomPtr >> 16 ) & 0xFF;
            BAM.Len=sizeof( FileInfo );
            WriteBAM( BAMptr, BAM );
            BAMptr++;

            /* Write File Info Structures. */
            FI.Status=0xFCBB;
            FI.ExtentPtr=BAMptr;
            FI.PrimaryPtr=FNULL;
            FI.SecondaryPtr=FNULL;
            FI.Attributes=0x00;
            FI.Time=0x8800;
            FI.Date=0x02F4;
            FI.VarStructLen=0;
            FI.UncompressedExtentLen=PseudoFileSize;
            FI.CompressedExtentLen=PseudoFileSize;
            CommonWrite( BottomPtr+CurrentBase, pBYTE &FI,
            sizeof( FileInfo ) );
            BottomPtr+=sizeof( FileInfo );

            /* Write Extent BAMs */
            /* Last BAM in block? */
            if ( (file_ctr+1)==(PseudoFileCnt-1) ) BAM.Status=0xBF;
            else BAM.Status=0x3F;
            BAM.Offset[0]=((DWORD)PseudoFileSize*file_ctr)&0xFF;
            BAM.Offset[1]=(((DWORD)PseudoFileSize*file_ctr)>>8)&0xFF;
            BAM.Offset[2]=(((DWORD)PseudoFileSize*file_ctr)>>16)&0xFF;
            BAM.Len=PseudoFileSize;
            WriteBAM( BAMptr, BAM );
            BAMptr++;

            /* Update the display. */
            pfile_num++;
            fprintf( stdout, "%c%c%c%c%c%5d", 8,8,8,8,8,pfile_num );
            file_ctr++;
        }
        block_ctr++;
    }
    fprintf( stdout, DOT_DONE_MSG );
}

/*****************************************************************************
    PAGE
*****************************************************************************
*
*   Title:     DoEntireCard
*
*   Description:  This procedure formats the entire card as one huge file.
*           It creates InfoCount number of File Info's and Extents in
*           each block's structure file space.
*
*****************************************************************************/
```

```
void DoEntireCard ( void )
{
    WORD   block_ctr=0;          /* Block counter. */
    WORD   info_ctr=0;           /* FI counter. */
    DWORD CurrentBase=0UL;          /* Current structure file address in
                        block.*/
    BlockAllocMember  BAM;
    FileEntry  FE;
    FileInfo   FI;

    fprintf( stdout, WRITING_FILE_MSG );

    /* We only need one file entry structure. */
    /* Write FileEntry BAM. */
    BAM.Status=0x3F;
    BAM.Offset[0]=BottomPtr&0xFF;
    BAM.Offset[1]=(BottomPtr>>8)&0xFF;
    BAM.Offset[2]=(BottomPtr>>16)&0xFF;
    BAM.Len=sizeof( FileEntry );
    WriteBAM( BAMptr, BAM );
    BAMptr++;

    /* Write File Entry. */
    FE.SiblingPtr=FNULL;
    FE.Status=0x00E7;
    FE.PrimaryPtr=BAMptr;
    FE.SecondaryPtr=FNULL;
    FE.Attributes=0x00;
    FE.Time=0x8800;
    FE.Date=0x02F4;
    FE.VarStructLen=0;
    FE.NameLen=DOS83;
    strncpy( FE.Name, "00000001", 8 );
    strncpy( FE.Ext, "BIN", 3 );
    CommonWrite( BottomPtr+CurrentBase, pBYTE &FE, sizeof( FileEntry ) );
    BottomPtr+=sizeof( FileEntry );

    /* Loop through the blocks. */
    while( block_ctr < ( Card.NumBlocks - 1 ) ) {
        info_ctr=0;

        /* Calculate the physical base of the block. */
        CurrentBase=(DWORD) Card.BlockSize * block_ctr;

        /* The BottomPtr of the structure file is initially set from
        **   the InitialFormat procedure. */
        if( block_ctr > 0 )
        {
            /* If this isn't block zero, then recalculate the new
            BottomPtr. */
            BottomPtr=(DWORD) Card.BlockSize-StructureFileSize;
            BAMptr=(DWORD) block_ctr << 16;
        }

        /* For each block, write x number of InfoCounts. */
        while( info_ctr < InfoCount ) {
```

```
/* Write FileInfo BAMs. */
BAM.Status=0x3F;
BAM.Offset[0]=BottomPtr & 0xFF;
BAM.Offset[1]=( BottomPtr >> 8 ) & 0xFF;
BAM.Offset[2]=( BottomPtr >> 16 ) & 0xFF;
BAM.Len=sizeof( FileInfo );
WriteBAM( BAMptr, BAM );
BAMptr++;

/* Write File Info Structures. */
FI.Status=0xFCAB;
FI.ExtentPtr=BAMptr;

/* Last file info? */
if( info_ctr == ( InfoCount - 1 ) )
{
    /* Last block? */
    if( block_ctr == ( Card.NumBlocks - 2 ) )
    {
        FI.PrimaryPtr=FNULL;
        FI.Status=0xFCBB;
    }
    else
    {
        FI.PrimaryPtr=( (DWORD) ( block_ctr + 1 )
        << 16 );
    }
}
else
{
    FI.PrimaryPtr=BAMptr+1;
}
FI.SecondaryPtr=FNULL;
FI.Attributes=0x00;
FI.Time=0x8800;
FI.Date=0x02F4;
FI.VarStructLen=0;

/* Last File Info in block ? */
if( info_ctr == ( InfoCount - 1 ) )
{
    FI.UncompressedExtentLen=ResidualSpace;
    FI.CompressedExtentLen=ResidualSpace;
}
else
{
    FI.UncompressedExtentLen=0x8000;
    FI.CompressedExtentLen=0x8000;
}
CommonWrite( BottomPtr+CurrentBase, pBYTE &FI, sizeof(
FileInfo ) );
BottomPtr+=sizeof( FileInfo );

/* Write Extent BAMs */
/* Last BAM in block? */
if( info_ctr == ( InfoCount - 1 ) )
```

```
            {
                BAM.Status=0xBF;
                BAM.Len=ResidualSpace;
            }
            else
            {
                BAM.Status=0x3F;
                BAM.Len=0x8000;
            }
            BAM.Offset[0]=((DWORD)0x8000*info_ctr)&0xFF;
            BAM.Offset[1]=((DWORD)0x8000*info_ctr>>8)&0xFF;
            BAM.Offset[2]=((DWORD)0x8000*info_ctr>>16)&0xFF;
            WriteBAM( BAMptr, BAM );
            BAMptr++;
            info_ctr++;
        }
        block_ctr++;
    }
    fprintf( stdout, DOT_DONE_MSG );
}

/*****************************************************************************
    PAGE
*****************************************************************************
*
*   Title:      InitialFormat
*
*   Description:  This procedure places a Boot Record, ROOT Directory and
*           Volume Label in block zero (at address BottomPtr) of the
*           card.
*
*****************************************************************************/

void InitialFormat ( void )
{
    /* Initial Format Structures. */
    BlockAllocMember  BAM;
    FileEntry     FE;
    DirectoryEntry    DE;
    BootRecord    BR;

    /* Write Boot Record BAM in Physical Block Zero. */
    BAM.Status=0x3F;
    BAM.Offset[0]=BottomPtr & 0xFF;
    BAM.Offset[1]=( BottomPtr >> 8 ) & 0xFF;
    BAM.Offset[2]=( BottomPtr >> 16 ) & 0xFF;
    BAM.Len=sizeof( BootRecord );
    fprintf( stdout, WRITE_BAM_MSG );
    WriteBAM( BAMptr, BAM );
    BAMptr++;
    fprintf( stdout, DONE_MSG );

    /* Write Boot Record in physical block zero. */
    BR.Signature=0xF1A5;
    BR.SerialNumber=0x66677788UL;
    BR.FFSWriteVersion=FFS_WRITE_VER;
```

```
BR.FFSReadVersion=FFS_READ_VER;
BR.TotalBlockCount=Card.NumBlocks;
BR.SpareBlockCount=1;
BR.BlockLen=Card.BlockSize;
BR.RootDirectoryPtr=BAMptr;
BR.Status=0xFFFF;
BR.BootCodeLen=0;
fprintf( stdout, WRITE_BOOT_MSG );
CommonWrite( BottomPtr, pBYTE &BR, sizeof( BootRecord ) );
BottomPtr+=sizeof( BootRecord );
fprintf( stdout, DONE_MSG );

/* Write ROOT Directory BAM in physical block zero. */
BAM.Status=0x3F;
BAM.Offset[0]=BottomPtr & 0xFF;
BAM.Offset[1]=( BottomPtr >> 8 ) & 0xFF;
BAM.Offset[2]=( BottomPtr >> 16 ) & 0xFF;
BAM.Len=sizeof( DirectoryEntry );
fprintf( stdout, WRITE_ROOT_BAM_MSG );
WriteBAM( BAMptr, BAM );
BAMptr++;
fprintf( stdout, DONE_MSG );

/* Write ROOT Directory Entry. */
DE.Status=0xFFE3;
DE.SiblingPtr=FNULL;
DE.PrimaryPtr=BAMptr;
DE.SecondaryPtr=FNULL;
DE.Attributes=0x10;
DE.Time=0x8800;
DE.Date=0x02F4;
DE.VarStructLen=0;
DE.NameLen=DOS83;
strncpy( DE.Name ,"ROOT    ", 8);
strncpy( DE.Ext ,"   ", 3);
fprintf( stdout, WRITE_ROOT_MSG );
CommonWrite( BottomPtr, pBYTE &DE, sizeof( DirectoryEntry ) );
BottomPtr+=sizeof( DirectoryEntry );
fprintf( stdout, DONE_MSG );

/* Write Volume Label BAM in physical block zero. */
BAM.Status=0x3F;
BAM.Offset[0]=BottomPtr & 0xFF;
BAM.Offset[1]=( BottomPtr >> 8 ) & 0xFF;
BAM.Offset[2]=( BottomPtr >> 16 ) & 0xFF;
BAM.Len=sizeof( FileEntry );
fprintf( stdout, WRITE_VOL_BAM_MSG );
WriteBAM( BAMptr, BAM );
BAMptr++;
fprintf( stdout, DONE_MSG );

/* Write Volume Label Entry. */
FE.Status=0x00B7;
FE.SiblingPtr=BAMptr;
FE.PrimaryPtr=FNULL;
FE.SecondaryPtr=FNULL;
```

```
    FE.Attributes=0x28;
    FE.Time=0x8800;
    FE.Date=0x02F4;
    FE.VarStructLen=0;
    FE.NameLen=DOS83;
    strncpy(FE.Name,"EMBEDDED",8);
    strncpy(FE.Ext,"FFS",3);
    fprintf( stdout, WRITE_VOLUME_MSG );
    CommonWrite( BottomPtr, pBYTE &FE, sizeof( FileEntry ) );
    BottomPtr+=sizeof( FileEntry );
    fprintf( stdout, DONE_MSG );
}

/*  End of File: SBM.C
**  COPYRIGHT (c) 1994 Intel Corporation, ALL RIGHTS RESERVED
*****************************************************************************/
```

```
/*****************************************************************************
*
*  FILE NAME:  SBM.H
*
*  PURPOSE:    Defines, Typedefs, and Structures for file Symmetric Block
*              Manager.
*
*  AUTHOR:     Peter J Torelli/James R. Massoni
*
*  COPYRIGHT (c) 1994 Intel Corporation, ALL RIGHTS RESERVED
*
*****************************************************************************/

#ifndef SBM.H            /* Has this file been included before? */
#define SBM.H               /* No, remember it has been now */

/*
**  Defines  ***************************************************************
*/

#define REVISION  1.02
#define BOOT_RECORD  0x01
#define COMM_READ 0x71
#define COMM_WRITE   0x51
#define DISK_DRIVE   0x08
#define DOS83     0x0B
#define ENTIRE    'E'
#define ERASE_DRIVE  0x54
#define ERROR     -1
#define FFS_READ_VER 0x200
#define FFS_WRITE_VER 0x200
#define FNULL     0xFFFFFFFFUL
#define GENERIC_IOCTL 0x440D
#define MEDIA_CHANGE 0x52
#define MEDIA_INFO   0x73
#define SPARE_BLOCK  0x02
#define SYMMETRIC 'S'
#define YES       'Y'
#define pBYTE     (BYTE *)

/*
**  Typedefs  **************************************************************
*/

typedef unsigned char BYTE;
typedef unsigned int  WORD;
typedef unsigned long DWORD;

typedef struct {
   DWORD  Size;
   DWORD  BlockSize;
   WORD   NumBlocks;
} CardInfo;

typedef struct {
   BYTE   Status;
```

```
    BYTE    Offset[3];
    WORD    Len;
} BlockAllocMember;

typedef struct {
    DWORD   BootRecordPtr;
    DWORD   EraseCount;
    WORD    BlockSeq;
    WORD    BlockSeqChecksum;
    WORD    Status;
} BlockAllocStruct;

typedef struct {
    WORD    Signature;
    DWORD   SerialNumber;
    WORD    FFSWriteVersion;
    WORD    FFSReadVersion;
    WORD    TotalBlockCount;
    WORD    SpareBlockCount;
    DWORD   BlockLen;
    DWORD   RootDirectoryPtr;
    WORD    Status;
    WORD    BootCodeLen;
/*  BYTE    BootCode[0];    Boot Code is ZERO. */
} BootRecord;

typedef struct {
    WORD    Status;
    DWORD   SiblingPtr;
    DWORD   PrimaryPtr;
    DWORD   SecondaryPtr;
    BYTE    Attributes;
    WORD    Time;
    WORD    Date;
    WORD    VarStructLen;
    BYTE    NameLen;
    BYTE    Name[8];
    BYTE    Ext[3];
} FileEntry, DirectoryEntry;

typedef struct {
    WORD    Status;
    DWORD   ExtentPtr;
    DWORD   PrimaryPtr;
    DWORD   SecondaryPtr;
    BYTE    Attributes;
    WORD    Time;
    WORD    Date;
    WORD    VarStructLen;
    WORD    UncompressedExtentLen;
    WORD    CompressedExtentLen;
} FileInfo;

/*
**  Unions and Structures  ************************************************
*/
```

```
union IOCTLDataPkt{

    struct EraseDrive {
        BYTE    Status;
    } ED;

    struct CommonMemReadWritePkt {
        BYTE    Status;
        WORD    Len;
        DWORD   Offset;
        BYTE far *PtrBuffer;
    } CommonMemRead, CommonMemWrite;

    struct MediaInfoPkt {
        BYTE    Status;
        BYTE    Len;
        BYTE    DevType;
        WORD    JedecID;
        BYTE    DriveSlot;
        BYTE    TotalSlots;
        BYTE    PartType;
        DWORD   PartBegin;
        DWORD   PartSize;
        DWORD   MediaSize;
        DWORD   BlockSize;
        BYTE    PartNum;
        BYTE    CurPartInSlot;
        BYTE    MaxPartInSlot;
        BYTE    TotPartInSlot;
        BYTE    DevInfoNum;
        BYTE    DevInfoSize;
        void far *DevInfoPtr;
        WORD    InitYear;
        BYTE    InitMonth;
        BYTE    InitDay;
        BYTE    InitHour;
        BYTE    InitSec;
        WORD    BatRepYear;
        BYTE    BatRepMonth;
        BYTE    BatRepDay;
        WORD    BatExpYear;
        BYTE    BatExpMonth;
        BYTE    BatExpDay;
        WORD    Flags;
        BYTE    VendorName[11];
        WORD    ChangeCount;
        DWORD   Reserved[5];
    } MediaInfo;
};

#endif

/*  End of File: SBM.H
**  COPYRIGHT (c) 1994 Intel Corporation, ALL RIGHTS RESERVED
*****************************************************************************/
```

```
/*****************************************************************************
*
*  FILE NAME:  SBM_MSG.H
*
*  PURPOSE:    Defines, Typedefs, and Structures for file Symmetric Block
*              Manager.
*
*  AUTHOR:     Peter J Torelli/James R. Massoni
*
*  COPYRIGHT (c) 1994 Intel Corporation, ALL RIGHTS RESERVED
*
*****************************************************************************/

#ifndef SBM_MSG.H        /* Has this file been included before? */
#define SBM_MSG.H        /* No, remember it has been now. */

/*
**  Defines  ***************************************************************
*/

#define CHOOSE_MSG          "Done\n\nChoose a format:\n\t(S)ymmetric\n\t(E)ntire\
Card as One File\n\nSelection (S/E): "

#define CRLF_MSG            "\n"
#define DBL_CRLF_MSG        "\n\n"
#define DONE_MSG            "Done.\n"
#define DOT_DONE_MSG        "...Done.\n"
#define ERASE_PROMPT_MSG    "Note: The media must be erased before\
formatting.\nDoes this media need to be erased? (y/n): "

#define OBTAINING_MSG       "Obtaining media geometry..."
#define LOGON_MSG           "LFM Linear File Formatter v%.2f (c) Intel\
Corporation\n\n"

#define NUM_PFILES_MSG      "Enter the number of PseudoFiles/Block [%d-%d]: "
#define PLEASE_WAIT_MSG     "\nPlease wait, media being erased..."
#define RESET_MEDIA_MSG     "Resetting media"
#define WRITE_BAM_MSG       "Writing Boot Record BAM in physical block zero..."
#define WRITE_BAS_MSG       "Writing BAS block %03Xh"
#define WRITE_BOOT_MSG      "Writing Boot Record in physical block zero..."
#define WRITE_PSEUDO_MSG    "Writing Pseudo File number %05d"
#define WRITE_ROOT_MSG      "Writing ROOT Directory Entry in physical block\
zero..."

#define WRITE_ROOT_BAM_MSG  "Writing ROOT BAM in physical block zero..."
#define WRITE_VOL_BAM_MSG   "Writing Volume Label BAM in physical block zero..."
#define WRITE_VOLUME_MSG    "Writing Volume Label in physical block zero..."
#define WRITING_FILE_MSG    "Writing File..."

/*  Define the format symmetric block screen */
#define SYMM_INFO_MSG       "\n\
Symmetric Preformat installed.\n\n\
  Number of Pseudo Files:             %6ld\n\
+ Number of Structure Files:          %6d\n\
  ___________________________________ ______\n\
= Total Number of Pseudo File Spaces: %6ld\n\n\
```

```
  Each Pseudo File Space is %ldk.\n\n\
  Avoid address ranges %lX-%lX in each block.\n"

/*  Define the format entire screen */
#define ENTIRE_INFO_MSG     "\n\
Entire-Card Preformat installed.\n\n\
  Number of Pseudo Files:             %6ld\n\
  Number of Structure Files:          %6d\n\n\
  The Pseudo File is %ldk.\n\n\
  Avoid address ranges %lX-%lX in each block.\n"

/*
**  Error Messages:
*/

#define ERROR_MSG           "\nError"
#define INVL_DRV_LET_MSG    "The drive letter entered is invalid.\n"
#define INVL_SELECT_MSG     "Error, Invalid selection.\n"
#define SPECIFY_MSG         "Please specify one drive letter.\n"
#define TOO_MANY_MSG        "Too many parameters, please specify only a drive\
letter.\n"

#define USAGE_MSG           "Correct usage example: C:>SBM E:\n"


#endif

/*  End of File: SBM_MSG.H
**  COPYRIGHT (c) 1994 Intel Corporation, ALL RIGHTS RESERVED
*****************************************************************************/
```

Revised Documents

28F016XS
16-MBIT (1 MBIT X 16, 2 MBIT X 8) SYNCHRONOUS FLASH MEMORY

- Effective Zero Wait-State Performance up to 33 Mhz
  - Synchronous Pipelined Reads
- SmartVoltage Technology
  - User-Selectable 3.3V or 5V V_{CC}
  - User-Selectable 5V or 12V V_{PP}
- 0.33 MB/sec Write Transfer Rate
- Configurable x8 or x16 Operation
- 56-Lead TSOP Type I Package
- Backwards-Compatible with 28F008SA Command-Set
- 2 µA Typical Deep Power-Down
- 1 mA Typical Active I_{CC} Current in Static Mode
- 16 Separately-Erasable/Lockable 128-Kbyte Blocks
- 1 Million Erase Cycles per Block
- State-of-the-Art 0.6 µm ETOX™ IV Flash Technology

Intel's 28F016XS 16-Mbit Flash memory is a revolutionary architecture which is the ideal choice for designing truly revolutionary high-performance products. Combining very high read performance with the intrinsic non-volatility of flash memory, the 28F016XS eliminates the traditional redundant memory paradigm of shadowing code from a slow nonvolatile storage source to a faster execution memory, such as DRAM, for improved system performance. The innovative capabilities of the 28F016XS enable the design of direct-execute code and mass storage data/file flash memory systems.

The 28F016XS is the highest performance high density nonvolatile read/write flash memory solution available today. Its synchronous pipelined read interface, flexible V_{CC} and V_{PP} voltages, extended cycling, fast write and read performance, symmetrically blocked architecture, and selective block locking provide a highly flexible memory component suitable for resident flash component arrays on the system board or SIMMs. The synchronous pipelined interface and x8/x16 architecture of the 28F016XS allow easy interface with minimal glue logic to a wide range of processors/buses, providing effective zero wait-state read performance up to 33 MHz. The 28F016XS's dual read voltage allows the same component to operate at either 3.3V or 5.0V V_{CC}. Programming voltage at 5V V_{PP} minimizes external circuitry in minimal-chip, space critical designs, while the 12V V_{PP} option maximizes write/erase performance. Its high read performance combined with flexible block locking enable both storage and execution of operating systems/application software and fast access to large data tables. The 28F016XS is manufactured on Intel's 0.6 µm ETOX™ IV process technology.

Order Number 290532-002

1.0 INTRODUCTION

The documentation of the Intel 28F016XS Flash memory device includes this datasheet, a detailed user's manual, a number of application notes and design tools, all of which are referenced at the end of this datasheet.

The datasheet is intended to give an overview of the chip feature-set and of the operating AC/DC specifications. The 16-Mbit Flash Product Family User's Manual provides complete descriptions of the user modes, system interface examples and detailed descriptions of all principles of operation. It also contains the full list of software algorithm flowcharts, and a brief section on compatibility with the Intel 28F008SA.

Significant 28F016XS feature revisions occurred between datasheet revisions 290532-001 and 290532-002. These revisions center around removal of the following features:

- All page buffer operations (read, write, programming, Upload Device Information)
- Command queuing
- Software Sleep and Abort
- Erase all Unlocked Blocks and Two-Byte Write
- RY/BY# reconfiguration as part of the Device Configuration command

Intel recommends that all customers obtain the latest revisions of 28F016XD documentation.

1.1 Product Overview

The 28F016XS is a high-performance, 16-Mbit (16,777,216-bit) block erasable nonvolatile random access memory organized as either 1 Mword x 16 or 2 Mbyte x 8, subdivided into even and odd banks. Address A_1 makes the bank selection. The 28F016XS includes sixteen 128-Kbyte (131,072 byte) blocks or sixteen 64-Kword (65,536 word) blocks. Chip memory maps for x8 and x16 modes are shown in Figures 3 and 4.

The implementation of a new architecture, with many enhanced features, will improve the device operating characteristics and result in greater product reliability and ease-of-use as compared to other flash memories. Significant features of the 28F016XS as compared to previous asynchronous flash memories include:

- Synchronous Pipelined Read Interface
- Significantly Improved Read and Write Performance
- SmartVoltage Technology
  - — Selectable 3.3V or 5.0 V_{CC}
  - — Selectable 5.0V or 12.0 V_{PP}
- Internal 3.3V/5.0V V_{CC} Detection Circuitry
- Block Write/Erase Protection

The 28F016XS's synchronous pipelined interface dramatically raises read performance far beyond previously attainable levels. Addresses are synchronously latched and data read from a 28F016XS bank every 30 ns (5V V_{CC}, SFI Configuration = 2). This capability translates to 0-wait-state reads at clock rates up to 33 MHz at 5V V_{CC}, after an initial address pipeline fill delay and assuming even and odd banks within the flash memory are alternately accessed. Data is latched and driven valid 20 ns (t_{CHQV}) after a rising CLK edge. The 28F016XS is capable of operating up to 66 MHz (5V V_{CC}); its programmable SFI Configuration enables system design flexibility, optimizing the 28F016XS to a specific system clock frequency. See Section 4.9, SFI Configuration Table, for specific SFI Configurations for given operating frequencies.

The SFI Configuration optimizes the 28F016XS for a wide range of system operating frequencies. The default SFI Configuration is 4, which allows system boot from the 28F016XS at any frequency up to 66 MHz at 5V V_{CC}. After initiating an access, data is latched and begins driving on the data outputs after a CLK count corresponding to the SFI Configuration has elapsed. The 28F016XS will hold data valid until CE_X# or OE# is deactivated or a CLK count corresponding to the SFI Configuration for a subsequent access has elapsed.

The CLK and ADV# inputs, new to the 28F016XS in comparison to previous flash memories, enable synchronous latching of input addresses for reads. The CLK input controls the device latencies, decrements the SFI Configuration counter and synchronizes data outputs. ADV# indicates the presence of a valid address on the 28F016XS address inputs. During read operations, addresses are latched and accesses are initiated on a rising CLK edge in conjunction with ADV# low. Both CLK and ADV# are ignored by the 28F016XS during command/data write sequences.

The 28F016XS incorporates SmartVoltage technology, providing V_{CC} operation at both 3.3V and 5.0V and program and erase capability at V_{PP} = 12.0V or 5.0V. Operating at V_{CC} = 3.3V, the 28F016XS consumes less than one half the power consumption at 5.0V V_{CC}, while 5.0V V_{CC} provides highest read performance capability. V_{PP} operation at 5.0V eliminates the need for a separate 12.0V converter, while the V_{PP} = 12.0V option maximizes write/erase performance. In addition to the flexible program and erase voltages, the dedicated V_{PP} gives complete code protection with $V_{PP} \leq V_{PPLK}$.

Internal 3.3V or 5.0V V_{CC} detection automatically configures the device for optimized 3.3V or 5.0V read/write operation. Hence, the 28F016SA's 3/5# pin is not required and is a no-connect (NC) on the 28F016XS, maintaining pin-out backwards-compatibility between components.

A Command User Interface (CUI) serves as the system interface between the microprocessor or microcontroller and the internal memory operation.

Internal Algorithm Automation allows byte/word writes and block erase operations to be executed using a Two-Write command sequence to the CUI in the same way as the 28F008SA 8-Mbit FlashFile™ memory.

Software locking of memory blocks is an added feature of the 28F016XS as compared to the 28F008SA. The 28F016XS provides selectable block locking to protect code or data such as direct-executable operating systems or application code. Each block has an associated nonvolatile lock-bit which determines the lock status of the block. In addition, the 28F016XS has a master Write Protect pin (WP#) which prevents any modifications to memory blocks whose lock-bits are set.

Writing of memory data is performed in either byte or word increments, typically within 6 µsec at 12.0V V_{PP}, which is a 33% improvement over the 28F008SA. A block erase operation erases one of the 16 blocks in typically 1.2 sec, independent of the other blocks.

Each block can be written and erased a minimum of 100,000 cycles. Systems can achieve one million Block Erase Cycles by providing wear-leveling algorithms and graceful block retirement. These techniques have already been employed in many flash file systems and hard disk drive designs.

All operations are started by a sequence of Write commands to the device. Three Status Registers (described in detail later in this datasheet) and a RY/BY# output pin provide information on the progress of the requested operation.

The following Status Registers are used to provide device and WSM operation information to the user:

- A Compatible Status Register (CSR) which is 100% compatible with the 28F008SA FlashFile memory Status Register. The CSR, when used alone, provides a straightforward upgrade capability to the 28F016XS from a 28F008SA-based design.
- A Global Status Register (GSR) which also informs the system of overall Write State Machine (WSM) status.
- 16 Block Status Registers (BSRs) which provide block-specific status information such as the block lock-bit status.

The GSR and BSR memory maps for Byte-Wide and Word-Wide modes are shown in Figures 5 and 6.

The 28F016XS incorporates an open drain RY/BY# output pin. This feature allows the user to OR-tie many RY/BY# pins together in a multiple memory configuration such as a Resident Flash Array.

The 28F016XS also incorporates a dual chip-enable function with two input pins, CE_0# and CE_1#. These pins have exactly the same functionality as the regular chip-enable pin, CE#, on the 28F008SA. For minimum chip designs, CE_1# may be tied to ground and system logic may use CE_0# as the chip enable input. The 28F016XS uses the logical combination of these two signals to enable or disable the entire chip. Both CE_0# and CE_1# must be active low to enable the device. If either one becomes inactive, the chip will be disabled. This feature, along with the open drain RY/BY# pin, allows the system designer to reduce the number of control pins used in a large array of 16-Mbit devices.

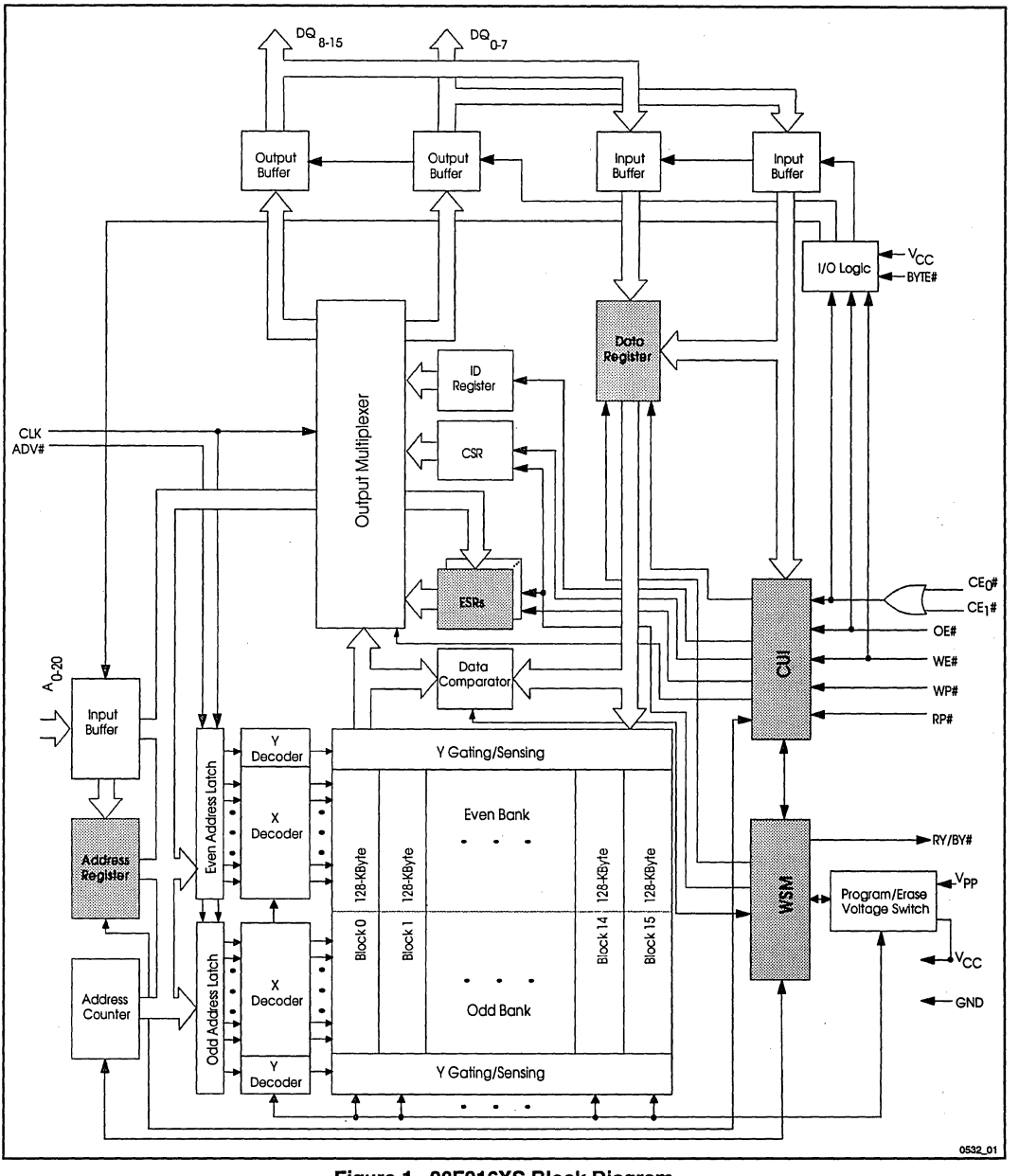


Figure 1. 28F016XS Block Diagram
Architectural Evolution Includes Synchronous Pipelined Read Interface, SmartVoltage Technology, and Extended Status Registers

The BYTE# pin allows either x8 or x16 read/writes to the 28F016XS. BYTE# at logic low selects 8-bit mode with address A_0 selecting between low byte and high byte. On the other hand, BYTE# at logic high enables 16-bit operation with address A_1 becoming the lowest order address and address A_0 is not used (don't care). A device block diagram is shown in Figure 1.

The 28F016XS incorporates an Automatic Power Saving (APS) feature, which substantially reduces the active current when the device is in static mode of operation (addresses not switching). In APS mode, the typical I_{CC} current is 1 mA at 5.0V (3 mA at 3.3V).

A deep power-down mode of operation is invoked when the RP# (called PWD# on the 28F008SA) pin transitions low. This mode brings the device power consumption to less than 2.0 µA, typically, and provides additional write protection by acting as a device reset pin during power transitions. A reset time of 300 ns (5V V_{CC}) is required from RP# switching high before latching an address into the 28F016XS. In the Deep Power-Down state, the WSM is reset (any current operation will abort) and the CSR, GSR and BSR registers are cleared.

A CMOS standby mode of operation is enabled when either CE_0# or CE_1# transitions high and RP# stays high with all input control pins at CMOS levels. In this mode, the device typically draws an I_{CC} standby current of 70 µA at 5V V_{CC}.

The 28F016XS is available in a 56-Lead, 1.2mm thick, 14mm x 20mm TSOP Type I package. The package's form factor and pinout allow for very high board layout densities.

2.0 DEVICE PINOUT

The 28F016XS is pinout compatible with the 28F016SA/SV 16-Mbit FlashFile memory components, providing a performance upgrade path to the 28F016XS. The 28F016XS 56-Lead TSOP pinout configuration is shown in Figure 2.

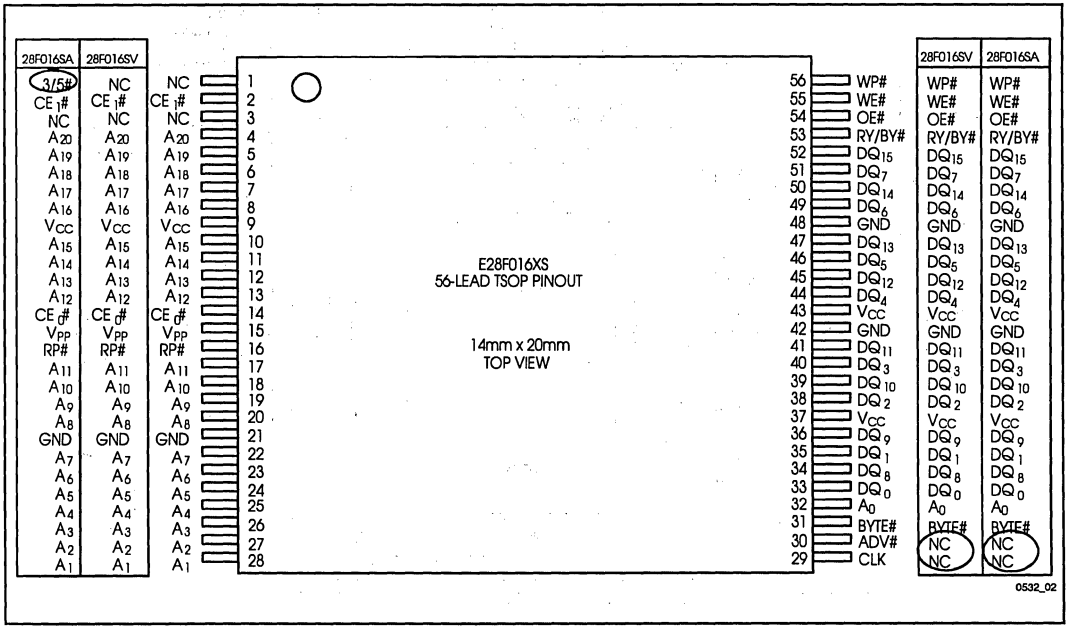


Figure 2. 28F016XS 56-Lead TSOP Pinout Configuration Shows Compatibility with the 28F016SA/SV, Allowing for Easy Performance Upgrades from Existing 16-Mbit Designs

2.1 Lead Descriptions

Symbol	Type	Name and Function
A_0	INPUT	**BYTE-SELECT ADDRESS:** Selects between high and low byte when device is in x8 mode. This address is latched in x8 data writes and ignored in x16 mode (i.e., the A_0 input buffer is turned off when BYTE# is high).
A_1	INPUT	**BANK-SELECT ADDRESS:** Selects an even or odd bank in a selected block. A 128-Kbyte block is subdivided into an even and odd bank. A_1 = 0 selects the even bank and A_1 = 1 selects the odd bank, in both byte-wide mode and word-wide mode device configurations.
A_2–A_{16}	INPUT	**WORD-SELECT ADDRESSES:** Select a word within one 128-Kbyte block. Address A_1 and A_{7-16} select 1 of 2048 rows, and A_{2-6} select 16 of 512 columns. These addresses are latched during both data reads and writes.
A_{17}–A_{20}	INPUT	**BLOCK-SELECT ADDRESSES:** Select 1 of 16 Erase blocks. These addresses are latched during data writes, erase and lock-block operations.
DQ_0–DQ_7	INPUT OUTPUT	**LOW-BYTE DATA BUS:** Inputs data and commands during CUI write cycles. Outputs array, identifier or status data in the appropriate read mode. Floated when the chip is de-selected or the outputs are disabled.
DQ_8–DQ_{15}	INPUT OUTPUT	**HIGH-BYTE DATA BUS:** Inputs data during x16 data-write operations. Outputs array or identifier data in the appropriate read mode; not used for Status Register reads. Outputs floated when the chip is de-selected, the outputs are disabled (OE# = V_{IH}) or BYTE# is driven active.
CE_0#, CE_1#	INPUT	**CHIP ENABLE INPUTS:** Activate the device's control logic, input buffers, decoders and sense amplifiers. With either CE_0# or CE_1# high, the device is de-selected and power consumption reduces to standby levels upon completion of any current data-write or erase operations. Both CE_0# and CE_1# must be low to select the device. All timing specifications are the same for both signals. Device Selection occurs with the latter falling edge of CE_0# or CE_1#. The first rising edge of CE_0# or CE_1# disables the device.
RP#	INPUT	**RESET/POWER-DOWN:** RP# low places the device in a Deep Power-Down state. All circuits that consume static power, even those circuits enabled in standby mode, are turned off. When returning from Deep Power-Down, a recovery time of t_{PHCH} is required to allow these circuits to power-up. When RP# goes low, the current WSM operation is terminated, and the device is reset. All Status Registers return to ready, clearing all status flags. Exit from Deep Power-Down places the device in read array mode.
OE#	INPUT	**OUTPUT ENABLE:** Drives device data through the output buffers when low. The outputs float to tri-state off when OE# is high. CE_X# overrides OE#, and OE# overrides WE#.
WE#	INPUT	**WRITE ENABLE:** Controls access to the CUI, Data Register and Address Latch. WE# is active low, and latches both address and data (command or array) on its rising edge.

2.1 Lead Descriptions (Continued)

Symbol	Type	Name and Function
CLK	INPUT	**CLOCK:** Provides the fundamental timing and internal operating frequency. CLK latches input addresses in conjunction with ADV#, times out the desired output SFI Configuration as a function of the CLK period, and synchronizes device outputs. CLK can be slowed or stopped with no loss of data or synchronization. CLK is ignored during write operations.
ADV#	INPUT	**ADDRESS VALID:** Indicates that a valid address is present on the address inputs. ADV# low at the rising edge of CLK latches the address on the address inputs into the flash memory and initiates a read access to the even or odd bank depending on the state of A_1. ADV# is ignored during write operations.
RY/BY#	OPEN DRAIN OUTPUT	**READY/BUSY:** Indicates status of the internal WSM. When low, it indicates that the WSM is busy performing an operation. RY/BY# high indicates that the WSM is ready for new operations, Erase is Suspended, or the device is in deep power-down mode. This output is always active (i.e., not floated to tri-state off when OE# or CE_0#, CE_1# are high).
WP#	INPUT	**WRITE PROTECT:** Erase blocks can be locked by writing a nonvolatile lock-bit for each block. When WP# is low, those locked blocks as reflected by the Block-Lock Status bits (BSR.6), are protected from inadvertent data writes or erases. When WP# is high, all blocks can be written or erased regardless of the state of the lock-bits. The WP# input buffer is disabled when RP# transitions low (deep power-down mode).
BYTE#	INPUT	**BYTE ENABLE:** BYTE# low places device in x8 mode. All data is then input or output on DQ_{0-7}, and DQ_{8-15} float. Address A_0 selects between the high and low byte. BYTE# high places the device in x16 mode, and turns off the A_0 input buffer. Address A_1 then becomes the lowest order address.
V_{PP}	SUPPLY	**WRITE/ERASE POWER SUPPLY (12.0V ± 0.6V, 5.0V ± 0.5V) :** For erasing memory array blocks or writing words/bytes into the flash array. V_{PP} = 5.0V ± 0.5V eliminates the need for a 12V converter, while the 12.0V ± 0.6V option maximizes Write/Erase Performance. Successful completion of write and erase attempts is inhibited with V_{PP} at or below 1.5V. Write and Erase attempts with V_{PP} between 1.5V and 4.5V, between 5.5V and 11.4V, and above 12.6V produce spurious results and should not be attempted.
V_{CC}	SUPPLY	**DEVICE POWER SUPPLY (3.3V ± 0.3V, 5.0V ± 0.5V):** Internal detection configures the device for 3.3V or 5.0V operation. To switch 3.3V to 5.0V (or vice versa), first ramp V_{CC} down to GND, and then power to the new V_{CC} voltage. Do not leave any power pins floating.
GND	SUPPLY	**GROUND FOR ALL INTERNAL CIRCUITRY:** Do not leave any ground pins floating.
NC		**NO CONNECT:** Lead may be driven or left floating.

3.0 MEMORY MAPS

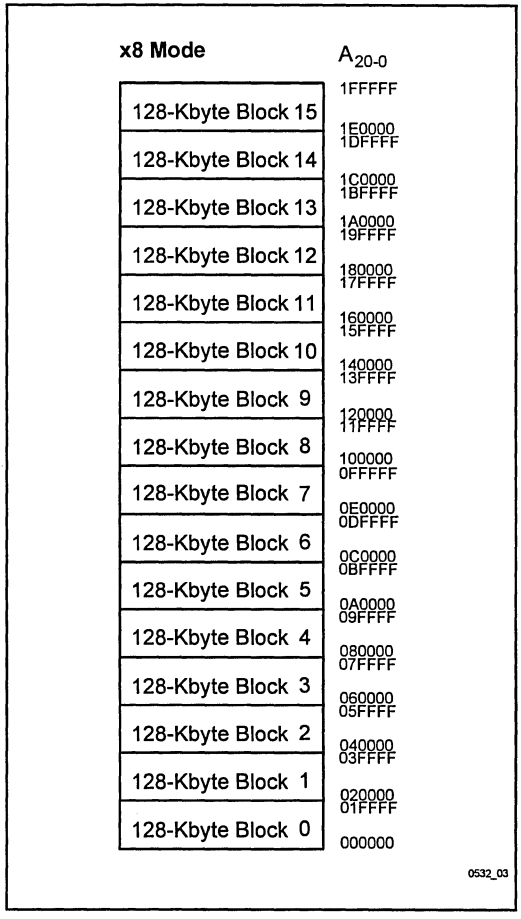


Figure 3. 28F016XS Memory Map (Byte-Wide Mode)

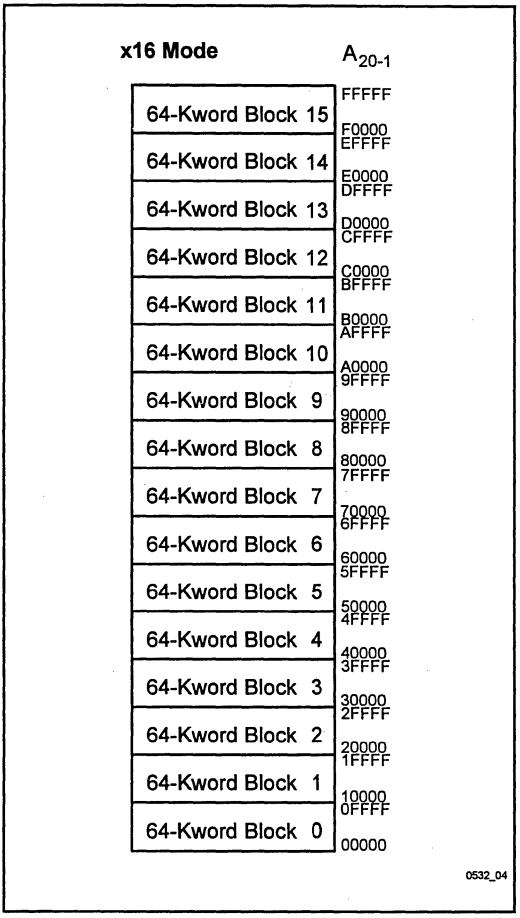


Figure 4. 28F016XS Memory Map (Word-Wide Mode)

3.1 Extended Status Register Memory Map

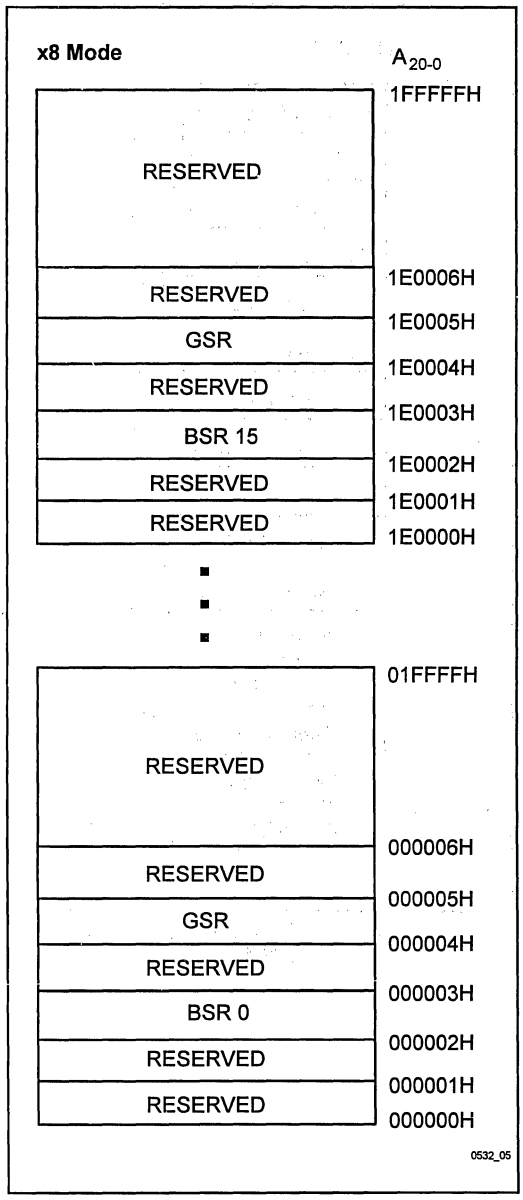


Figure 5. Extended Status Register Memory Map (Byte-Wide Mode)

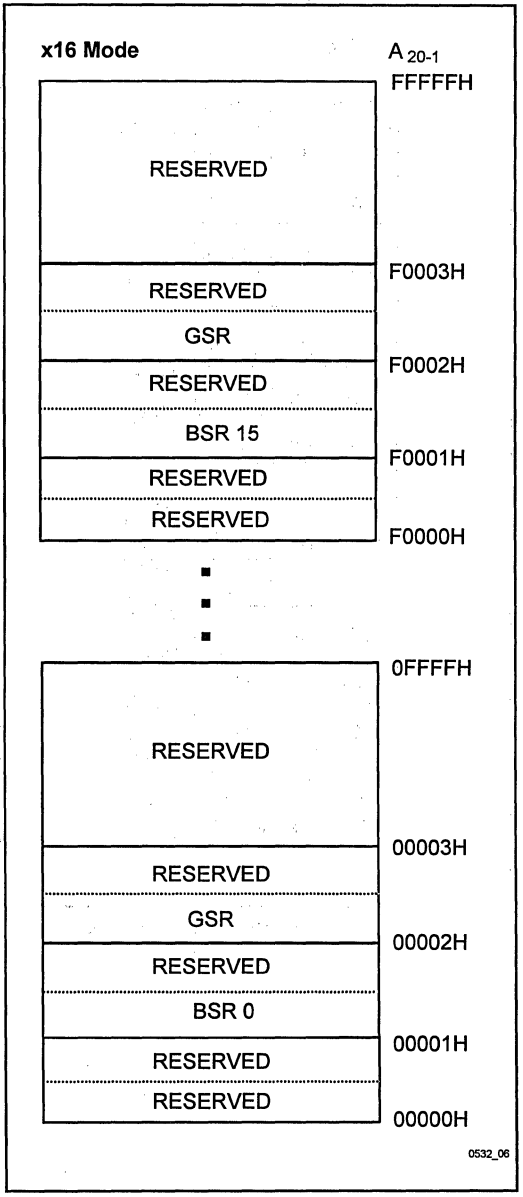


Figure 6. Extended Status Register Memory Map (Word-Wide Mode)

4.0 BUS OPERATIONS, COMMANDS AND STATUS REGISTER DEFINITIONS

4.1 Bus Operations for Word-Wide Mode (BYTE# = V_{IH})

Mode	Notes	RP#	CE_{0-1}#	OE#	WE#	ADV#	CLK	A_1	DQ_{0-15}	RY/BY#
Latch Read Address	1,9,10	V_{IH}	V_{IL}	X	V_{IH}	V_{IL}	↑	X	X	X
Inhibit Latching Read Address	1,9	V_{IH}	V_{IL}	X	V_{IH}	V_{IH}	↑	X	X	X
Read	1,2,7,9	V_{IH}	V_{IL}	V_{IL}	V_{IH}	X	↑	X	D_{OUT}	X
Output Disable	1,6,7,9	V_{IH}	V_{IL}	V_{IH}	V_{IH}	X	X	X	High Z	X
Standby	1,6,7,9	V_{IH}	V_{IL}	X	X	X	X	X	High Z	X
Deep Power-Down	1,3	V_{IL}	X	X	X	X	X	X	High Z	V_{OH}
Manufacturer ID	1,4,9	V_{IH}	V_{IL}	V_{IL}	V_{IH}	X	↑	V_{IL}	0089H	V_{OH}
Device ID	1,4,8,9	V_{IH}	V_{IL}	V_{IL}	V_{IH}	X	↑	V_{IH}	66A8H	V_{OH}
Write	1,5,6,9	V_{IH}	V_{IL}	V_{IH}	V_{IL}	X	X	X	D_{IN}	X

NOTES:

1. X can be V_{IH} or V_{IL} for address or control pins except for RY/BY#, which is either V_{OL} or V_{OH}, or High Z or D_{OUT} for data pins depending on whether or not OE# is active.
2. RY/BY# output is open drain. When the WSM is ready, Erase is suspended, or the device is in deep power-down mode, RY/BY# will be at V_{OH} if it is tied to V_{CC} through a resistor. RY/BY# at V_{OH} is independent of OE# while a WSM operation is in progress.
3. RP# at GND ± 0.2V ensures the lowest deep power-down current.
4. A_0 and A_1 at V_{IL} provide device manufacturer codes in x8 and x16 modes respectively. A_0 and A_1 at V_{IH} provide device ID codes in x8 and x16 modes respectively. All other addresses are set to zero.
5. Commands for erase, data write, or lock-block operations can only be completed successfully when $V_{PP} = V_{PPH1}$ or $V_{PP} = V_{PPH2}$.
6. While the WSM is running, RY/BY# stays at V_{OL} until all operations are complete. RY/BY# goes to V_{OH} when the WSM is not busy or in erase suspend mode.
7. RY/BY# may be at V_{OL} while the WSM is busy performing various operations (for example, a Status Register read during a write operation).
8. The 28F016XS shares an identical device identifier with the 28F016XD.
9. CE_{0-1}# at V_{IL} is defined as both CE_0# and CE_1# low, and CE_{0-1}# at V_{IH} is defined as either CE_0# or CE_1# high.
10. Addresses are latched on the rising edge of CLK in conjunction with ADV# low. Address $A_1 = 0$ selects the even bank and $A_1 = 1$ selects the odd bank, in both byte-wide mode and word-wide mode device configurations.

4.2 Bus Operations for Byte-Wide Mode (BYTE# = V_{IL})

Mode	Notes	RP#	CE_{0-1}#	OE#	WE#	ADV#	CLK	A_0	DQ_{0-7}	RY/BY#
Latch Read Address	1,9,10	V_{IH}	V_{IL}	X	V_{IH}	V_{IL}	↑	X	X	X
Inhibit Latching Read Address	1,9	V_{IH}	V_{IL}	X	V_{IH}	V_{IH}	↑	X	X	X
Read	1,2,7,9	V_{IH}	V_{IL}	V_{IL}	V_{IH}	X	↑	X	D_{OUT}	X
Output Disable	1,6,7,9	V_{IH}	V_{IL}	V_{IH}	V_{IH}	X	X	X	High Z	X
Standby	1,6,7,9	V_{IH}	V_{IH}	X	X	X	X	X	High Z	X
Deep Power-Down	1,3	V_{IL}	X	X	X	X	X	X	High Z	V_{OH}
Manufacturer ID	1,4,9	V_{IH}	V_{IL}	V_{IL}	V_{IH}	X	↑	V_{IL}	89H	V_{OH}
Device ID	1,4,8,9	V_{IH}	V_{IL}	V_{IL}	V_{IH}	X	↑	V_{IH}	A8H	V_{OH}
Write	1,5,6,9	V_{IH}	V_{IL}	V_{IH}	V_{IL}	X	X	X	D_{IN}	X

NOTES:

1. X can be V_{IH} or V_{IL} for address or control pins except for RY/BY#, which is either V_{OL} or V_{OH}, or High Z or D_{OUT} for data pins depending on whether or not OE# is active.
2. RY/BY# output is open drain. When the WSM is ready, Erase is suspended, or the device is in deep power-down mode, RY/BY# will be at V_{OH} if it is tied to V_{CC} through a resistor. RY/BY# at V_{OH} is independent of OE# while a WSM operation is in progress.
3. RP# at GND ± 0.2V ensures the lowest deep power-down current.
4. A_0 and A_1 at V_{IL} provide device manufacturer codes in x8 and x16 modes respectively. A_0 and A_1 at V_{IH} provide device ID codes in x8 and x16 modes respectively. All other addresses are set to zero.
5. Commands for erase, data write, or lock-block operations can only be completed successfully when $V_{PP} = V_{PPH1}$ or $V_{PP} = V_{PPH2}$.
6. While the WSM is running, RY/BY# stays at V_{OL} until all operations are complete. RY/BY# goes to V_{OH} when the WSM is not busy or in erase suspend mode.
7. RY/BY# may be at V_{OL} while the WSM is busy performing various operations (for example, a Status Register read during a write operation).
8. The 28F016XS shares an identical device identifier with the 28F016XD.
9. CE_{0-1}# at V_{IL} is defined as both CE_0# and CE_1# low, and CE_{0-1}# at V_{IH} is defined as either CE_0# or CE_1# high.
10. Addresses are latched on the rising edge of CLK in conjunction with ADV# low. Address $A_1 = 0$ selects the even bank and $A_1 = 1$ selects the odd bank, in both byte-wide mode and word-wide mode device configurations.

4.3 28F008SA—Compatible Mode Command Bus Definitions

Command	Notes	First Bus Cycle			Second Bus Cycle		
		Oper	Addr	Data[4]	Oper	Addr	Data[4]
Read Array		Write	X	xxFFH	Read	AA	AD
Intelligent Identifier	1	Write	X	xx90H	Read	IA	ID
Read Compatible Status Register	2	Write	X	xx70H	Read	X	CSRD
Clear Status Register	3	Write	X	xx50H			
Word/Byte Write		Write	X	xx40H	Write	WA	WD
Alternate Word/Byte Write		Write	X	xx10H	Write	WA	WD
Block Erase/Confirm		Write	X	xx20H	Write	BA	xxD0H
Erase Suspend/Resume		Write	X	xxB0H	Write	X	xxD0H

ADDRESS

AA = Array Address

BA = Block Address

IA = Identifier Address

WA = Write Address

X = Don't Care

DATA

AD = Array Data

CSRD = CSR Data

ID = Identifier Data

WD = Write Data

NOTES:

1. Following the Intelligent Identifier command, two read operations access the manufacturer and device signature codes.
2. The CSR is automatically available after device enters data write, erase, or suspend operations.
3. Clears CSR.3, CSR.4 and CSR.5. Also clears GSR.5 and all BSR.5, BSR.4 and BSR.2 bits. See Status Register definitions.
4. The upper byte of the data bus (D_{8-15}) during command writes is a "Don't Care" in x16 operation of the device.

4.4 28F016XS—Enhanced Command Bus Definitions

Command	Notes	First Bus Cycle			Second Bus Cycle		
		Oper	Addr	Data[4]	Oper	Addr	Data[4]
Read Extended Status Register	1	Write	X	xx71H	Read	RA	GSRD BSRD
Lock Block/Confirm		Write	X	xx77H	Write	BA	xxD0H
Upload Status Bits/Confirm	2	Write	X	xx97H	Write	X	xxD0H
Device Configuration	3	Write	X	xx96H	Write	X	DCCD

ADDRESS

BA = Block Address

RA = Extended Register Address

WA = Write Address

X = Don't Care

DATA

AD = Array Data

BSRD = BSR Data

GSRD = GSR Data

DCCD = Device Configuration Code Data

NOTES:

1. RA can be the GSR address or any BSR address. See Figures 4 and 5 for Extended Status Register memory maps.
2. Upon device power-up, all BSR lock-bits come up locked. The Upload Status Bits command must be written to reflect the actual lock-bit status.
3. This command sets the SFI Configuration allowing the device to be optimized for the specific sytem operating frequency.
4. The upper byte of the Data bus (D_{8-15}) during command writes is a "Don't Care" in x16 operation of the device.

4.5 Compatible Status Register

WSMS	ESS	ES	DWS	VPPS	R	R	R
7	6	5	4	3	2	1	0

CSR.7 = WRITE STATE MACHINE STATUS
1 = Ready
0 = Busy

CSR.6 = ERASE-SUSPEND STATUS
1 = Erase Suspended
0 = Erase In Progress/Completed

CSR.5 = ERASE STATUS
1 = Error In Block Erasure
0 = Successful Block Erase

CSR.4 = DATA-WRITE STATUS
1 = Error in Data Write
0 = Data Write Successful

CSR.3 = V_{PP} STATUS
1 = V_{PP} Error Detect, Operation Abort
0 = V_{PP} OK

CSR.2-0 = RESERVED FOR FUTURE ENHANCEMENTS
These bits are reserved for future use; mask them out when polling the CSR.

NOTES:

RY/BY# output or WSMS bit must be checked to determine completion of an operation (Erase, Erase Suspend, or Data Write) before the appropriate Status bit (ESS, ES or DWS) is checked for success.

If DWS and ES are set to "1" during an erase attempt, an improper command sequence was entered. Clear the CSR and attempt the operation again.

The VPPS bit, unlike an A/D converter, does not provide continuous indication of V_{PP} level. The WSM interrogates V_{PP}'s level only after the Data Write or Erase command sequences have been entered, and informs the system if V_{PP} has not been switched on. VPPS is not guaranteed to report accurate feedback between V_{PPLK}(max) and V_{PPH1}(min), between V_{PPH1}(max) and V_{PPH2}(min), and above V_{PPH2}(max).

4.6 Global Status Register

WSMS	OSS	DOS	R	R	R	R	R
7	6	5	4	3	2	1	0

GSR.7 = WRITE STATE MACHINE STATUS
- 1 = Ready
- 0 = Busy

GSR.6 = OPERATION SUSPEND STATUS
- 1 = Operation Suspended
- 0 = Operation in Progress/Completed

GSR.5 = DEVICE OPERATION STATUS
- 1 = Operation Unsuccessful
- 0 = Operation Successful or Currently Running

GSR.4-0 = RESERVED FOR FUTURE ENHANCEMENTS
These bits are reserved for future use; mask them out when polling the GSR.

NOTES:

RY/BY# output or WSMS bit must be checked to determine completion of an operation (Block Lock, Suspend, Upload Status Bits, Erase or Data Write) before the appropriate Status bit (OSS or DOS) is checked for success.

4.7 Block Status Register

BS	BLS	BOS	R	R	VPPS	VPPL	R
7	6	5	4	3	2	1	0

BSR.7 = BLOCK STATUS
1 = Ready
0 = Busy

BSR.6 = BLOCK LOCK STATUS
1 = Block Unlocked for Write/Erase
0 = Block Locked for Write/Erase

BSR.5 = BLOCK OPERATION STATUS
1 = Operation Unsuccessful
0 = Operation Successful or Currently Running

BSR.2 = V_{PP} STATUS
1 = V_{PP} Error Detect, Operation Abort
0 = V_{PP} OK

BSR.1 = V_{PP} LEVEL
1 = V_{PP} Detected at 5.0V ± 10%
0 = V_{PP} Detected at 12.0V ± 5%

NOTES:

RY/BY# output or BS bit must be checked to determine completion of an operation (Block Lock, Suspend, Erase or Data Write) before the appropriate Status bits (BOS, BLS) is checked for success.

BSR.1 is not guaranteed to report accurate feedback between the V_{PPH1} and V_{PPH2} voltage ranges. Writes and erases with V_{PP} between V_{PPLK}(max) and V_{PPH1} (min), between V_{PPH1}(max) and V_{PPH2}(min), and above V_{PPH2}(max) produce spurious results and should not be attempted.

BSR.4,3,0 = RESERVED FOR FUTURE ENHANCEMENTS
These bits are reserved for future use; mask them out when polling the BSRs.

4.8 Device Configuration Code

R	R	SFI2	SFI1	SFI0	R	R	R
7	6	5	4	3	2	1	0

DCC.5-DCC.3 = SFI CONFIGURATION (SFI2-SFI0)
001 = SFI Configuration 1
010 = SFI Configuration 2
011 = SFI Configuration 3
100 = SFI Configuration 4 (Default)

NOTES:
Default SFI Configuration on power-up or return from deep power-down mode is 4, allowing system boot from the 28F016XS at any frequency up to the device's maximum frequency. Undocumented combinations of SFI2-SFI0 are reserved by Intel Corporation for future implementations and should not be used.

DCC.7-6,2-0 = RESERVED FOR FUTURE ENHANCEMENTS
These bits are reserved for future use; mask them out when reading the Device Configuration Code. Set these bits to "0" when writing the desired SFI Configuration to the device.

4.9 SFI Configuration Table

SFI Configuration	Notes	28F016XS-15 Frequency (MHz)	28F016XS-20 Frequency (MHz)	28F016XS-25 Frequency (MHz)
4	1	66 (and below)	50 (and below)	40 (and below)
3		50 (and below)	37.5 (and below)	30 (and below)
2		33 (and below)	25 (and below)	20 (and below)
1		16.7 (and below)	12.5 (and below)	10 (and below)

NOTE:
1. Default SFI Configuration after power-up or return from deep power-down mode via RP# low.

5.0 ELECTRICAL SPECIFICATIONS

NOTICE: This datasheet contains information on products in the sampling and initial production phases of development. The specifications are subject to change without notice. Verify with your local Intel Sales office that you have the latest datasheet before finalizing a design.

5.1 Absolute Maximum Ratings*

Temperature Under Bias .....0°C to +80°C

Storage Temperature .....–65°C to +125°C

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

V_{CC} = 3.3V ± 0.3V Systems

Symbol	Parameter	Notes	Min	Max	Units	Test Conditions
T_A	Operating Temperature, Commercial	1	0	70	°C	Ambient Temperature
V_{CC}	V_{CC} with Respect to GND	2	–0.2	7.0	V	
V_{PP}	V_{PP} Supply Voltage with Respect to GND	2,3	–0.2	14.0	V	
V	Voltage on any Pin (except V_{CC},V_{PP}) with Respect to GND	2,5	–0.5	V_{CC} + 0.5	V	
I	Current into any Non-Supply Pin	5		± 30	mA	
I_{OUT}	Output Short Circuit Current	4		100	mA	

V_{CC} = 5.0V ± 0.5V Systems

Symbol	Parameter	Notes	Min	Max	Units	Test Conditions
T_A	Operating Temperature, Commercial	1	0	70	°C	Ambient Temperature
V_{CC}	V_{CC} with Respect to GND	2	–0.2	7.0	V	
V_{PP}	V_{PP} Supply Voltage with Respect to GND	2,3	–0.2	14.0	V	
V	Voltage on any Pin (except V_{CC},V_{PP}) with Respect to GND	2,5	–2.0	7.0	V	
I	Current into any Non-Supply Pin	5		± 30	mA	
I_{OUT}	Output Short Circuit Current	4		100	mA	

NOTES:

1. Operating temperature is for commercial product defined by this specification.
2. Minimum DC voltage is –0.5V on input/output pins. During transitions, this level may undershoot to –2.0V for periods <20 ns. Maximum DC voltage on input/output pins is V_{CC} +0.5V which may overshoot to V_{CC} +2.0V for periods <20 ns.
3. Maximum DC voltage on V_{PP} may overshoot to +14.0V for periods <20 ns.
4. Output shorted for no more than one second. No more than one output shorted at a time.
5. This specification also applies to pins marked "NC."

5.2 Capacitance

For a 3.3V ± 0.3V System:

Symbol	Parameter	Notes	Typ	Max	Units	Test Conditions
C_{IN}	Capacitance Looking into an Address/Control Pin	1	6	8	pF	T_A = 25°C, f = 1.0 MHz
C_{OUT}	Capacitance Looking into an Output Pin	1	8	12	pF	T_A = 25°C, f = 1.0 MHz
C_{LOAD}	Load Capacitance Driven by Outputs for Timing Specifications	1		50	pF	For the 28F016XS-20 and 28F016XS-25

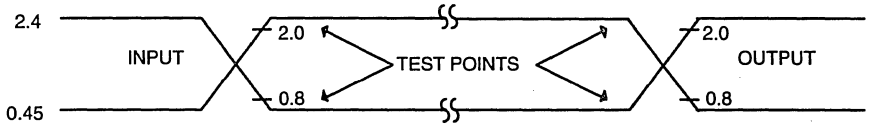
For 5.0V ± 0.5V System:

Symbol	Parameter	Notes	Typ	Max	Units	Test Conditions
C_{IN}	Capacitance Looking into an Address/Control Pin	1	6	8	pF	T_A = 25°C, f = 1.0 MHz
C_{OUT}	Capacitance Looking into an Output Pin	1	8	12	pF	T_A = 25°C, f = 1.0 MHz
C_{LOAD}	Load Capacitance Driven by Outputs for Timing Specifications	1		100	pF	For the 28F016XS-20
				30	pF	For the 28F016XS-15

NOTE:

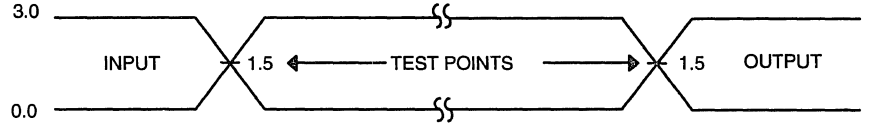
1. Sampled, not 100% tested. Guaranteed by design.
2. To obtain iBIS models for the 28F016XS, please contact your local Intel/Distribution Sales Office.

5.3 Transient Input/Output Reference Waveforms



AC test inputs are driven at V_{OH} (2.4 VTTL) for a Logic "1" and V_{OL} (0.45 VTTL) for a Logic "0." Input timing begins at V_{IH} (2.0 VTTL) and V_{IL} (0.8 VTTL). Output timing ends at V_{IH} and V_{IL}. Input rise and fall times (10% to 90%) <10 ns.

Figure 7. Transient Input/Output Reference Waveform (V_{CC} = 5.0V ± 0.5V) for Standard Testing Configuration(1)



AC test inputs are driven at 3.0V for a Logic "1" and 0.0V for a Logic "0." Input timing begins, and output timing ends, at 1.5V. Input rise and fall times (10% to 90%) <10 ns.

Figure 8. Transient Input/Output Reference Waveform (V_{CC} = 3.3V ± 0.3V) High Speed Reference Waveform(2) (V_{CC} = 5.0V ± 0.5V)

NOTES:

1. Testing characteristics for 28F016XS-20 at 5V V_{CC}.
2. Testing characteristics for 28F016XS-15 at 5V V_{CC} and 28F016XS-20/28F016XS-25 at 3.3V V_{CC}.

5.4 DC Characteristics

V_{CC} = 3.3V ± 0.3V, T_A = 0°C to +70°C

Symbol	Parameter	Notes	Min	Typ	Max	Units	Test Conditions
I_{LI}	Input Load Current	1			± 1	µA	$V_{CC} = V_{CC}$ Max, $V_{IN} = V_{CC}$ or GND
I_{LO}	Output Leakage Current	1			± 10	µA	$V_{CC} = V_{CC}$ Max, $V_{OUT} = V_{CC}$ or GND
I_{CCS}	V_{CC} Standby Current	1,5		70	130	µA	$V_{CC} = V_{CC}$ Max, CE_0#, CE_1#, RP# = V_{CC} ± 0.2V BYTE#, WP# = V_{CC} ± 0.2V or GND ± 0.2V
				1	4	mA	$V_{CC} = V_{CC}$ Max, CE_0#, CE_1#, RP# = V_{IH} BYTE#, WP# = V_{IH} or V_{IL}
I_{CCD}	V_{CC} Deep Power-Down Current	1		2	5	µA	RP# = GND ± 0.2V BYTE# = V_{CC} ± 0.2V or GND ± 0.2V
I_{CCR}[1]	V_{CC} Word/Byte Read Current	1,4,5		65	85	mA	$V_{CC} = V_{CC}$ Max CMOS: CE_0# ,CE_1# = GND ± 0.2V BYTE# = GND ± 0.2V or V_{CC} ± 0.2V Inputs = GND ± 0.2V or V_{CC} ± 0.2V 4-Location Access Sequence: 3-1-1-1 (clocks) f = 25 MHz, I_{OUT} = 0 mA
I_{CCR}[2]	V_{CC} Word/Byte Read Current	1,4, 5,6		60	75	mA	$V_{CC} = V_{CC}$ Max CMOS: CE_0#, CE_1# = GND ± 0.2V BYTE# = GND ± 0.2V or V_{CC} ± 0.2V Inputs = GND ± 0.2V or V_{CC} ± 0.2V 4-Location Access Sequence: 3-1-1-1 (clocks) f = 16 MHz, I_{OUT} = 0 mA

5.4 DC Characteristics (Continued)

V_{CC} = 3.3V ± 0.3V, T_A = 0°C to +70°C

Symbol	Parameter	Notes	Min	Typ	Max	Units	Test Conditions
I_{CCW}	V_{CC} Write Current	1,6		8	12	mA	Word/Byte Write in Progress V_{PP} = 12.0V ± 5%
				8	17	mA	Word/Byte Write in Progress V_{PP} = 5.0V ± 10%
I_{CCE}	V_{CC} Block Erase Current	1,6		6	12	mA	Block Erase in Progress V_{PP} = 12.0V ± 5%
				9	17	mA	Block Erase in Progress V_{PP} = 5.0V ± 10%
I_{CCES}	V_{CC} Erase Suspend Current	1,2		3	6	mA	CE_0#, CE_1# = V_{IH} Block Erase Suspended
I_{PPS}	V_{PP} Standby/Read Current	1		±1	±10	µA	$V_{PP} \leq V_{CC}$
I_{PPR}				30	200	µA	$V_{PP} > V_{CC}$
I_{PPD}	V_{PP} Deep Power-Down Current	1		0.2	5	µA	RP# = GND ± 0.2V
I_{PPW}	V_{PP} Write Current	1,6		10	15	mA	V_{PP} = 12.0V ± 5% Word/Byte Write in Progress
				15	25	mA	V_{PP} = 5.0V ± 10% Word/Byte Write in Progress
I_{PPE}	V_{PP} Erase Current	1,6		4	10	mA	V_{PP} = 12.0V ± 5% Block Erase in Progress
				14	20	mA	V_{PP} = 5.0V ± 10% Block Erase in Progress
I_{PPES}	V_{PP} Erase Suspend Current	1		30	50	µA	V_{PP} = V_{PPH1} or V_{PPH2}, Block Erase Suspended
V_{IL}	Input Low Voltage	6	–0.3		0.8	V	
V_{IH}	Input High Voltage	6	2.0		V_{CC} +0.3	V	
V_{OL}	Output Low Voltage	6			0.4	V	V_{CC} = V_{CC} Min and I_{OL} = 4 mA
$V_{OH}1$	Output High Voltage	6	2.4			V	I_{OH} = –2.0 mA V_{CC} = V_{CC} Min
$V_{OH}2$			V_{CC} –0.2			V	I_{OH} = –100 µA V_{CC} = V_{CC} Min

5.4 DC Characteristics (Continued)

V_{CC} = 3.3V ± 0.3V, T_A = 0°C to +70°C

Symbol	Parameter	Notes	Min	Typ	Max	Units	Test Conditions
V_{PPLK}	V_{PP} Erase/Write Lock Voltage	3,6	0.0		1.5	V	
V_{PPH1}	V_{PP} during Write/Erase Operations	3	4.5	5.0	5.5	V	
V_{PPH2}	V_{PP} during Write/Erase Operations	3	11.4	12.0	12.6	V	
V_{LKO}	V_{CC} Erase/Write Lock Voltage		2.0			V	

NOTES:

1. All currents are in RMS unless otherwise noted. Typical values at V_{CC} = 3.3V, V_{PP} = 12.0V or 5.0V, T = 25°C. These currents are valid for all product versions (package and speeds).
2. I_{CCES} is specified with the device de-selected. If the device is read while in erase suspend mode, current draw is the sum of I_{CCES} and I_{CCR}.
3. Block erases, word/byte writes and lock block operations are inhibited when $V_{PP} \leq V_{PPLK}$ and not guaranteed in the ranges between V_{PPLK}(max) and V_{PPH1}(min), between V_{PPH1} (max) and V_{PPH2}(min) and above V_{PPH2}(max).
4. Automatic Power Savings (APS) reduces I_{CCR} to 3 mA typical in static operation.
5. CMOS Inputs are either V_{CC} ± 0.2V or GND ± 0.2V. TTL Inputs are either V_{IL} or V_{IH}.
6. Sampled, but not 100% tested. Guaranteed by design.

5.5 DC Characteristics

V_{CC} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Symbol	Parameter	Notes	Min	Typ	Max	Units	Test Conditions
I_{LI}	Input Load Current	1			± 1	µA	$V_{CC} = V_{CC}$ Max $V_{IN} = V_{CC}$ or GND
I_{LO}	Output Leakage Current	1			± 10	µA	$V_{CC} = V_{CC}$ Max $V_{OUT} = V_{CC}$ or GND
I_{CCS}	V_{CC} Standby Current	1,5		70	130	µA	$V_{CC} = V_{CC}$ Max CE_0#, CE_1#, RP# = V_{CC} ± 0.2V BYTE#, WP# = V_{CC} ± 0.2V or GND ± 0.2V
				2	4	mA	$V_{CC} = V_{CC}$ Max CE_0#, CE_1#, RP# = V_{IH} BYTE#, WP# = V_{IH} or V_{IL}
I_{CCD}	V_{CC} Deep Power-Down Current	1		2	5	µA	RP# = GND ± 0.2V BYTE# = V_{CC} ± 0.2V or GND ± 0.2V
I_{CCR}[1]	V_{CC} Read Current	1,4,5		120	175	mA	$V_{CC} = V_{CC}$ Max, CMOS: CE_0# ,CE_1# = GND ± 0.2V BYTE# = GND ± 0.2V or V_{CC} ± 0.2V Inputs = GND ± 0.2V or V_{CC} ± 0.2V 4-Location Access Sequence: 3-1-1-1 (clocks) f = 33 MHz, I_{OUT} = 0 mA
I_{CCR}[2]	V_{CC} Read Current	1,4, 5,6		105	150	mA	$V_{CC} = V_{CC}$ Max, CMOS: CE_0#, CE_1# = GND ± 0.2V BYTE# = GND ± 0.2V or V_{CC} ± 0.2V Inputs = GND ± 0.2V or V_{CC} ± 0.2V 4-Location Access Sequence: 3-1-1-1 (clocks) f = 20 MHz, I_{OUT} = 0 mA

5.5 DC Characteristics (Continued)

V_{CC} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Symbol	Parameter	Notes	Min	Typ	Max	Units	Test Conditions
I_{CCW}	V_{CC} Write Current	1,6		25	35	mA	Word/Byte in Progress V_{PP} = 12.0V ± 5%
				25	40	mA	Word/Byte in Progress V_{PP} = 5.0V ± 10%
I_{CCE}	V_{CC} Erase Suspend Current	1,6		18	25	mA	Block Erase in Progress V_{PP} = 12.0V ± 5%
				20	30	mA	Block Erase in Progress V_{PP} = 5.0V ± 10%
I_{CCES}	V_{CC} Block Erase Current	1,2		5	10	mA	CE_0#, CE_1# = V_{IH} Block Erase Suspended
I_{PPS}	V_{PP} Standby/Read Current	1		± 1	± 10	µA	$V_{PP} \leq V_{CC}$
I_{PPR}				30	200	µA	$V_{PP} > V_{CC}$
I_{PPD}	V_{PP} Deep Power-Down Current	1		0.2	5	µA	RP# = GND ± 0.2V
I_{PPW}	V_{PP} Write Current	1,6		7	12	mA	V_{PP} = 12.0V ± 5% Word/Byte Write in Progress
				17	22	mA	V_{PP} = 5.0V ± 10% Word/Byte Write in Progress
I_{PPE}	V_{PP} Block Erase Current	1,6		5	10	mA	V_{PP} = 12.0V ± 5% Block Erase in Progress
				16	20	mA	V_{PP} = 5.0V ± 10% Block Erase in Progress
I_{PPES}	V_{PP} Erase Suspend Current	1		30	50	µA	V_{PP} = V_{PPH1} or V_{PPH2}, Block Erase Suspended
V_{IL}	Input Low Voltage	6	–0.5		0.8	V	
V_{IH}	Input High Voltage	6	2.0		V_{CC} +0.5	V	
V_{OL}	Output Low Voltage	6			0.45	V	V_{CC} = V_{CC} Min I_{OL} = 5.8 mA
$V_{OH}1$	Output High Voltage	6	0.85 V_{CC}			V	I_{OH} = –2.5 mA V_{CC} = V_{CC} Min
$V_{OH}2$			V_{CC} –0.4				I_{OH} = –100 µA V_{CC} = V_{CC} Min

5.5 DC Characteristics (Continued)

V_{CC} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Symbol	Parameter	Notes	Min	Typ	Max	Units	Test Conditions
V_{PPLK}	V_{PP} Write/Erase Lock Voltage	3,6	0.0		1.5	V	
V_{PPH1}	V_{PP} during Write/Erase Operations		4.5	5.0	5.5	V	
V_{PPH2}	V_{PP} during Write/Erase Operations		11.4	12.0	12.6	V	
V_{LKO}	V_{CC} Write/Erase Lock Voltage		2.0			V	

NOTES:

1. All currents are in RMS unless otherwise noted. Typical values at V_{CC} = 5.0V, V_{PP} = 12.0V or 5.0V, T = 25°C. These currents are valid for all product versions (package and speeds) and are specified for a CMOS rise/fall time (10% to 90%) of <5 ns and a TTL rise/fall time of <10 ns.
2. I_{CCES} is specified with the device de-selected. If the device is read while in erase suspend mode, current draw is the sum of I_{CCES} and I_{CCR}.
3. Block erases, word/byte writes and lock block operations are inhibited when $V_{PP} \leq V_{PPLK}$ and not guaranteed in the ranges between V_{PPLK}(max) and V_{PPH1}(min), between V_{PPH1} (max) and V_{PPH2}(min) and above V_{PPH2}(max).
4. Automatic Power Saving (APS) reduces I_{CCR} to 1 mA typical in static operation.
5. CMOS Inputs are either V_{CC} ± 0.2V or GND ± 0.2V. TTL Inputs are either V_{IL} or V_{IH}.
6. Sampled, but not 100% tested. Guaranteed by design.

5.6 Timing Nomenclature

All 3.3V system timings are measured from where signals cross 1.5V.

For 5.0V systems, use the standard JEDEC cross point definitions (standard testing) or from where signals cross 1.5V (high speed testing).

Each timing parameter consists of 5 characters. Some common examples are defined below:

t_{ELCH} time(t) from CE# (E) going low (L) to CLK (C) going high (H)

t_{AVCH} time(t) from address (A) valid (V) to CLK (C) going high (H)

t_{WHDX} time(t) from WE# (W) going high (H) to when the data (D) can become undefined (X)

	Pin Characters		Pin States
A	Address Inputs	H	High
C	CLK (Clock)	L	Low
D	Data Inputs	V	Valid
Q	Data Outputs	X	Driven, but Not Necessarily Valid
E	CE# (Chip Enable)	Z	High Impedance
F	BYTE# (Byte Enable)	L	Latched
G	OE# (Output Enable)		
W	WE# (Write Enable)		
P	RP# (Deep Power-Down Pin)		
R	RY/BY# (Ready Busy)		
V	ADV# (Address Valid)		
5V	V_{CC} at 4.5V Minimum		
3V	V_{CC} at 3.0V Minimum		

5.7 AC Characteristics—Read Only Operations[1]

V_{CC} = 3.3V ± 0.3V, T_A = 0°C to +70°C

Versions[3]			28F016XS-20		28F016XS-25		
Symbol	**Parameter**	**Notes**	**Min**	**Max**	**Min**	**Max**	**Units**
f_{CLK}	CLK Frequency	7		50		40	MHz
t_{CLK}	CLK Period		20		25		ns
t_{CH}	CLK High Time		6		8.5		ns
t_{CL}	CLK Low Time		6		8.5		ns
t_{CLCH}	CLK Rise Time			4		4	ns
t_{CHCL}	CLK Fall Time			4		4	ns
t_{ELCH}	CE_X# Setup to CLK	6	25		35		ns
t_{VLCH}	ADV# Setup to CLK		20		25		ns
t_{AVCH}	Address Valid to CLK		20		25		ns
t_{CHAX}	Address Hold from CLK		0		0		ns
t_{CHVH}	ADV# Hold from CLK		0		0		ns
t_{GLCH}	OE# Setup to CLK		20		25		ns
t_{CHQV}	CLK to Data Delay			30		35	ns
t_{PHCH}	RP# High to CLK		480		480		ns
t_{CHQX}	Output Hold from CLK	2	6		6		ns
t_{ELQX}	CE_X# to Output Low Z	2,6	0		0		ns
t_{EHQZ}	CE_X# High to Output High Z	2,6		30		30	ns
t_{GLQX}	OE# to Output Low Z	2	0		0		ns
t_{GHQZ}	OE# High to Output High Z	2		30		30	ns
t_{OH}	Output Hold from CE_X# or OE# Change, Whichever Occurs First	6	0		0		ns

5.7 AC Characteristics—Read Only Operations[1] (Continued)

V_{CC} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Versions[3]			28F016XS-15[4]		28F016XS-20[5]		
Symbol	**Parameter**	**Notes**	**Min**	**Max**	**Min**	**Max**	**Units**
f_{CLK}	CLK Frequency	7		66		50	MHz
t_{CLK}	CLK Period		15		20		ns
t_{CH}	CLK High Time		3.5		6		ns
t_{CL}	CLK Low Time		3.5		6		ns
t_{CLCH}	CLK Rise Time			4		4	ns
t_{CHCL}	CLK Fall Time			4		4	ns
t_{ELCH}	CE_X# Setup to CLK	6	25		30		ns
t_{VLCH}	ADV# Setup to CLK		15		20		ns
t_{AVCH}	Address Valid to CLK		15		20		ns
t_{CHAX}	Address Hold from CLK		0		0		ns
t_{CHVH}	ADV# Hold from CLK		0		0		ns
t_{GLCH}	OE# Setup to CLK		15		20		ns
t_{CHQV}	CLK to Data Delay			20		30	ns
t_{PHCH}	RP# High to CLK		300		300		ns
t_{CHQX}	Output Hold from CLK	2	5		5		ns
t_{ELQX}	CE_X# to Output Low Z	2,6	0		0		ns
t_{EHQZ}	CE_X# High to Output High Z	2,6		30		30	ns
t_{GLQX}	OE# to Output Low Z	2	0		0		ns
t_{GHQZ}	OE# High to Output High Z	2		30		30	ns
t_{OH}	Output Hold from CE_X# or OE# Change, Whichever Occurs First	6	0		0		ns

NOTES:

1. See AC Input/Output Reference Waveforms for timing measurements.
2. Sampled, not 100% tested. Guaranteed by design.
3. Device speeds are defined as:
   15 ns at V_{CC} = 5.0V equivalent to 20 ns at V_{CC} = 3.3V
   20 ns at V_{CC} = 5.0V equivalent to 25 ns at V_{CC} = 3.3V
4. See the high speed AC Input/Output Reference Waveforms.
5. See the standard AC Input/Output Reference Waveforms.
6. CE_X# is defined as the latter of CE_0# or CE_1# going low, or the first of CE_0# or CE_1# going high.
7. Page buffer reads are valid at any frequency up to the corresponding SFI Configuration setting of 2. Page buffer reads above this frequency may produce invalid results and should not be attempted. See Section 4.9 for SFI Configuration frequency settings.

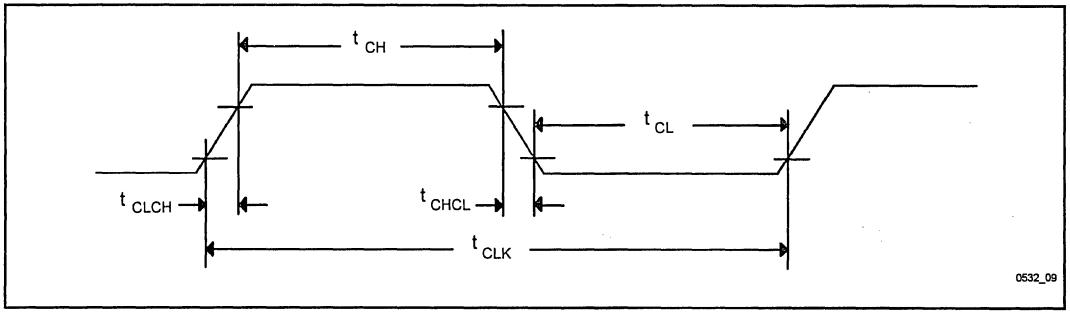
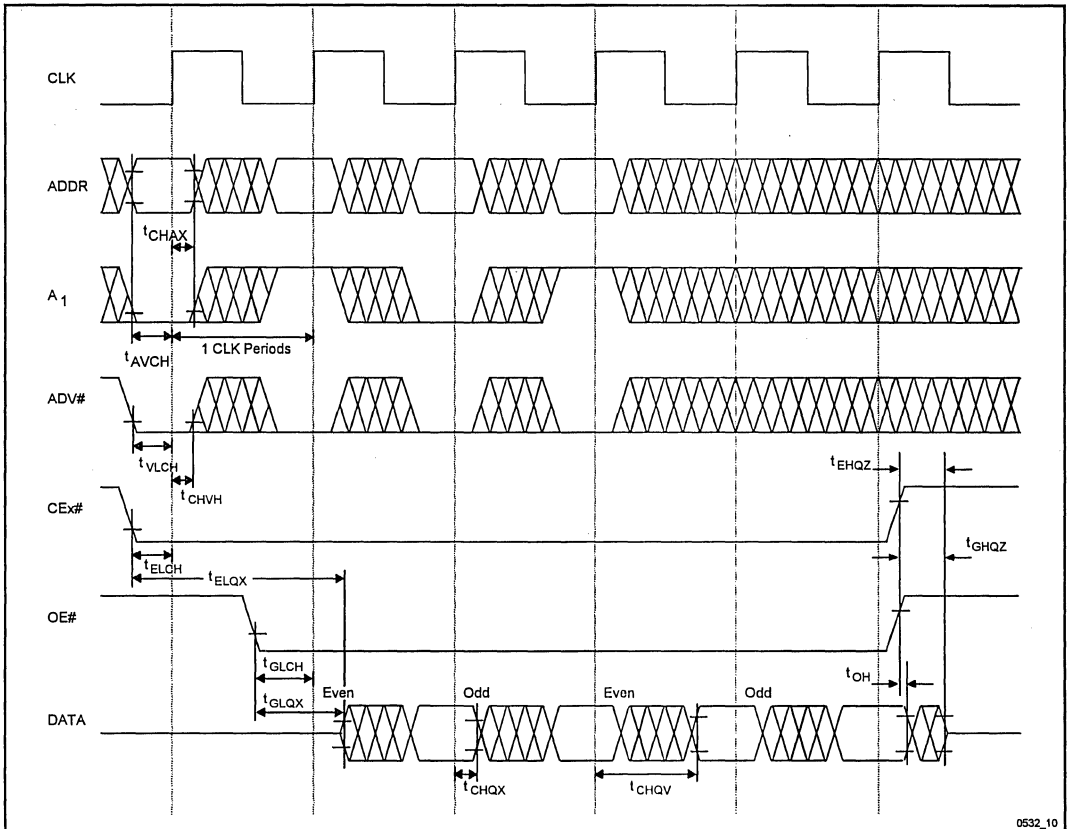


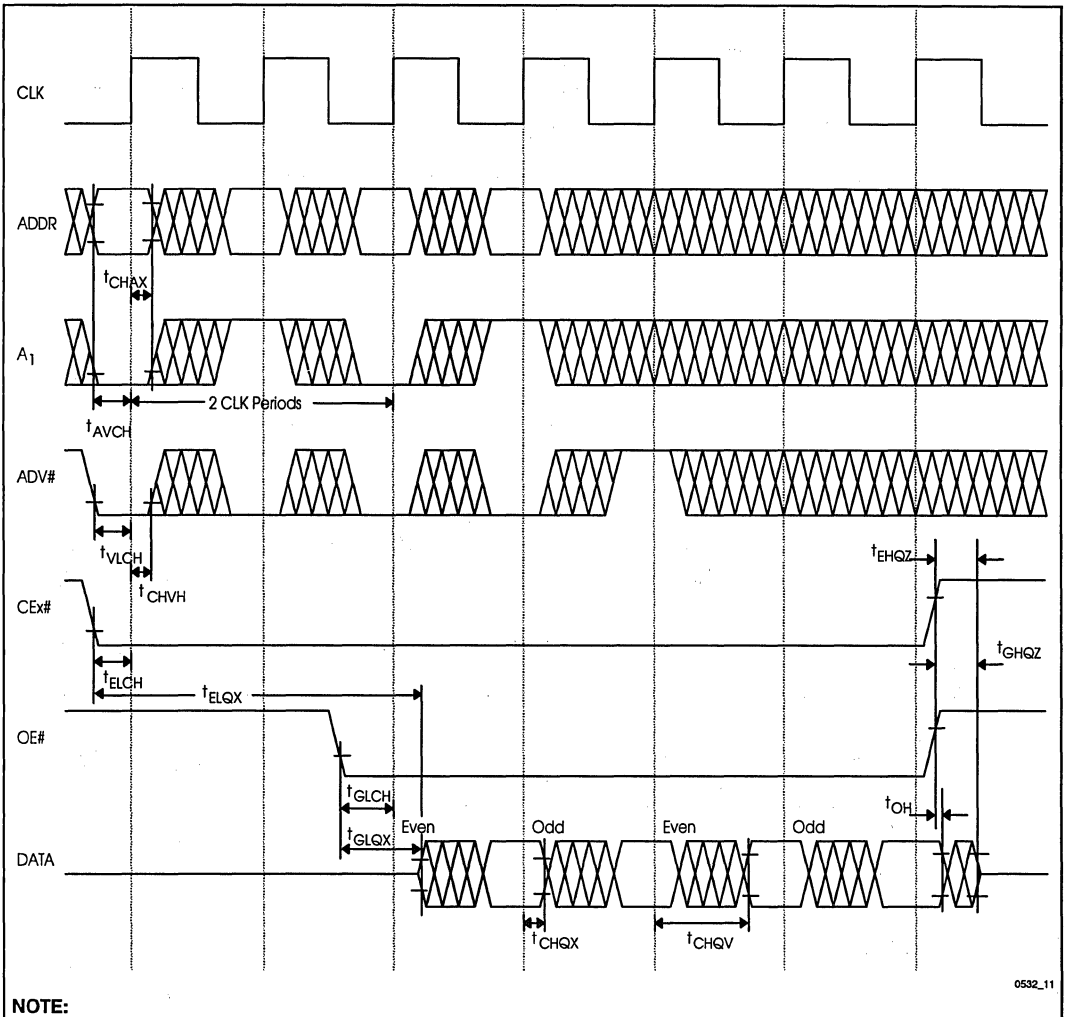
Figure 9. CLK Waveform



NOTE:

1. The 28F016XS can sustain an optimized burst access throughout the 28F016XS array assuming alternating bank accesses; the length of the burst access is dictated by the control CPU or bus architecture.

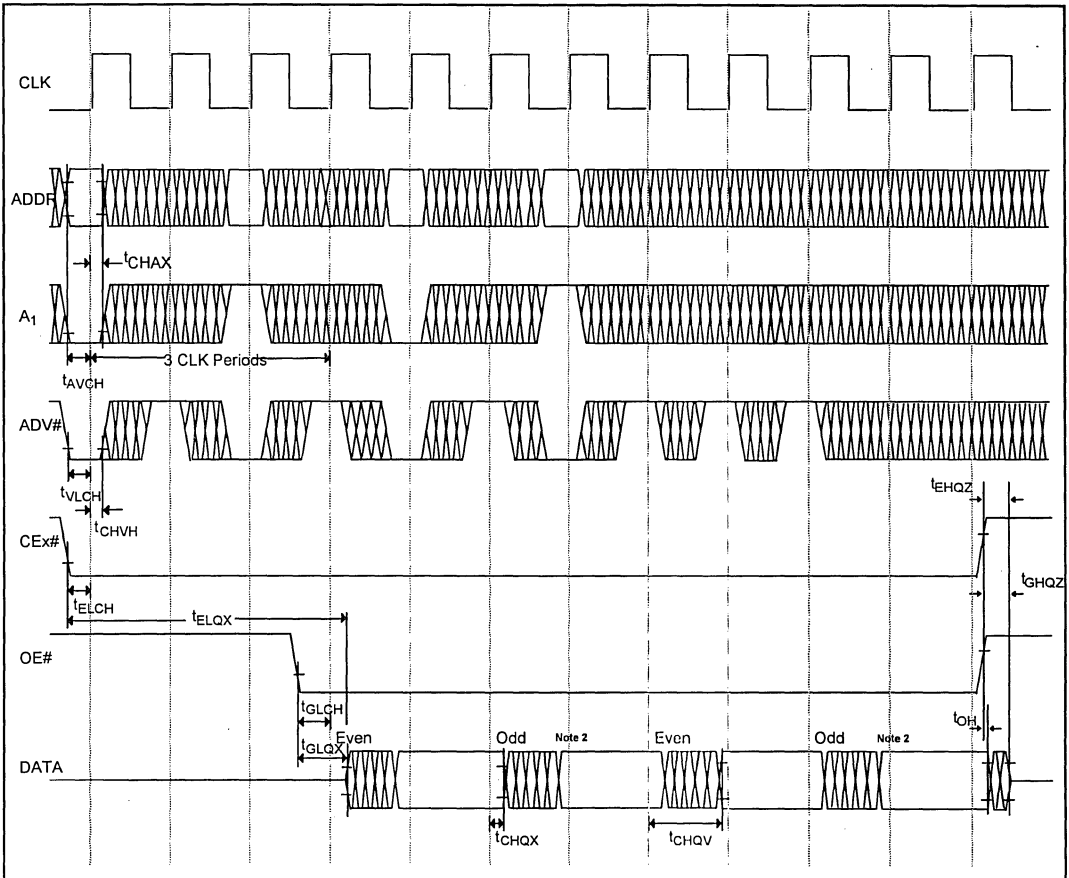
Figure 10. Read Timing Waveform(1)
(SFI Configuration = 1, Alternate-Bank Accesses)



NOTE:

1. The 28F016XS can sustain an optimized burst access throughout the 28F016XS array assuming alternating bank accesses; the length of the burst access is dictated by the control CPU or bus architecture.

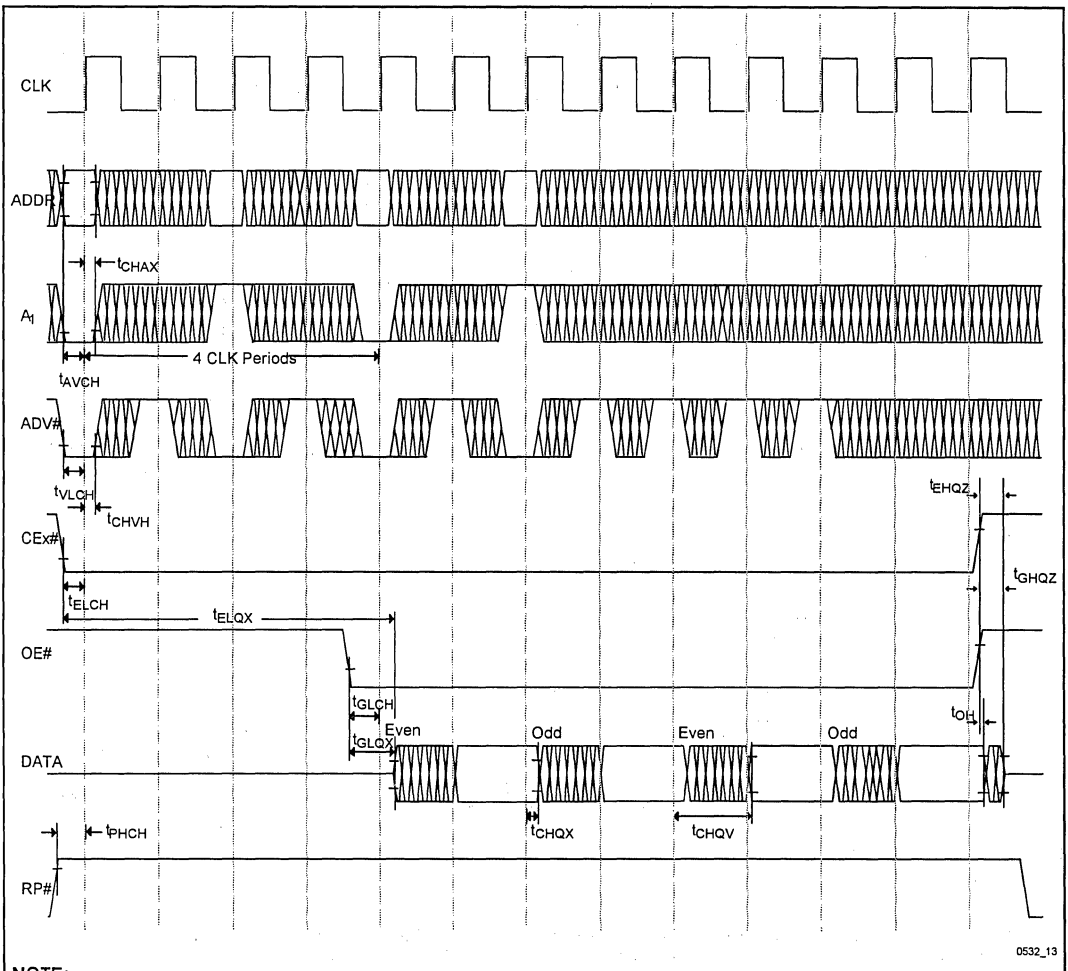
Figure 11. Read Timing Waveform(1)
(SFI Configuration = 2, Alternate-Bank Accesses)



NOTES:

1. The 28F016XS can sustain an optimized burst access throughout the 28F016XS array assuming alternating bank accesses; the length of the burst access is dictated by the control CPU or bus architecture.
2. Depending on the actual operation frequency, a consecutive alternating bank access can be initiated one clock period earlier. See AP-398 for further information.

Figure 12. Read Timing Waveform(1)
(SFI Configuration = 3, Alternate-Bank Accesses)



NOTE:

1. The 28F016XS can sustain an optimized burst access throughout the 28F016XS array assuming alternating bank accesses; the length of the burst access is dictated by the control CPU or bus architecture.

Figure 13. Read Timing Waveform(1)
(SFI Configuration = 4, Alternating Bank Accesses)

5.8 AC Characteristics for WE#—Controlled Write Operations[1]

V_{CC} = 3.3V ± 0.3V, T_A = 0°C to +70°C

Versions			28F016XS-20			28F016XS-25			
Symbol	Parameter	Notes	Min	Typ	Max	Min	Typ	Max	Unit
t_{AVAV}	Write Cycle Time		75			75			ns
t_{VPWH}	V_{PP} Setup to WE# Going High	1,2,3	100			100			ns
t_{PHEL}	RP# Setup to CE_X# Going Low	3,7	480			480			ns
t_{ELWL}	CE_X# Setup to WE# Going Low	3,7	0			0			ns
t_{AVWH}	Address Setup to WE# Going High	2,6	60			60			ns
t_{DVWH}	Data Setup to WE# Going High	2,6	60			60			ns
t_{WLWH}	WE# Pulse Width		60			60			ns
t_{WHDX}	Data Hold from WE# High	2	5			5			ns
t_{WHAX}	Address Hold from WE# High	2	5			5			ns
t_{WHEH}	CE_X# hold from WE# High	3,7	5			5			ns
t_{WHWL}	WE# Pulse Width High		15			15			ns
t_{GHWL}	Read Recovery before Write	3	0			0			ns
t_{WHRL}	WE# High to RY/BY# Going Low	3			100			100	ns
t_{RHPL}	RP# Hold from Valid Status Register (CSR, GSR, BSR) data and RY/BY# High	3	0			0			ns
t_{PHWL}	RP# High Recovery to WE# Going Low	3	480			480			ns
t_{WHCH}	Write Recovery before Read		20			20			ns
t_{QVVL}	V_{PP} Hold from Valid Status Register (CSR, GSR, BSR) Data and RY/BY# High	1,2,3	0			0			µs
t_{WHQV}	Duration of Word/Byte Write Operation	1	5	9	TBD	5	9	TBD	µs
t_{WHQV}	Duration of Block Erase Operation	2,3,4	0.6	1.6	20	0.6	1.6	20	sec

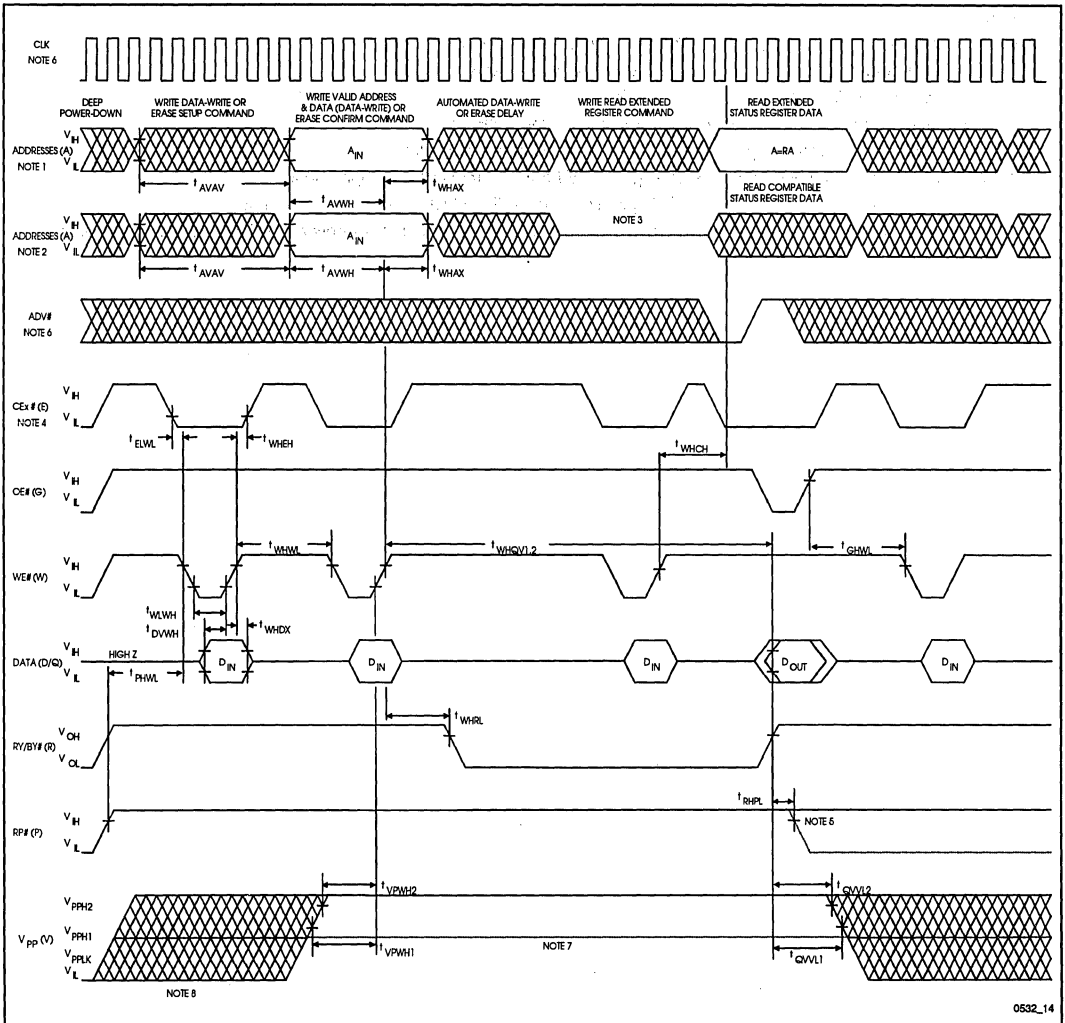
5.8 AC Characteristics for WE#—Controlled Write Operations[1] (Continued)

V_{CC} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Versions			28F016XS-15			28F016XS-20			
Symbol	Parameter	Notes	Min	Typ	Max	Min	Typ	Max	Unit
t_{AVAV}	Write Cycle Time		65			65			ns
t_{VPWH}	V_{PP} Setup to WE# Going High	1,2,3	100			100			ns
t_{PHEL}	RP# Setup to CE_X# Going Low	3,7	300			300			ns
t_{ELWL}	CE_X# Setup to WE# Going Low	3,7	0			0			ns
t_{AVWH}	Address Setup to WE# Going High	2,6	50			50			ns
t_{DVWH}	Data Setup to WE# Going High	2,6	50			50			ns
t_{WLWH}	WE# Pulse Width		50			50			ns
t_{WHDX}	Data Hold from WE# High	2	0			0			ns
t_{WHAX}	Address Hold from WE# High	2	5			5			ns
t_{WHEH}	CE_X# hold from WE# High	3,7	5			5			ns
t_{WHWL}	WE# Pulse Width High		15			15			ns
t_{GHWL}	Read Recovery before Write	3	0			0			ns
t_{WHRL}	WE# High to RY/BY# Going Low	3			100			100	ns
t_{RHPL}	RP# Hold from Valid Status Register (CSR, GSR, BSR) data and RY/BY# High	3	0			0			ns
t_{PHWL}	RP# High Recovery to WE# Going Low	3	300			300			ns
t_{WHCH}	Write Recovery before Read		20			20			ns
t_{QVVL}	V_{PP} Hold from Valid Status Register (CSR, GSR, BSR) Data and RY/BY# High	1,2,3	0			0			µs
t_{WHQV1}	Duration of Word/Byte Write Operation	1,3,4, 5,8	4.5	6	TBD	4.5	6	TBD	µs
t_{WHQV2}	Duration of Block Erase Operation	3,4	0.6	1.2	20	0.6	1.2	20	sec

NOTES:

1. Read timings during write and erase are the same as for normal read.
2. Refer to command definition tables for valid address and data values.
3. Sampled, but not 100% tested. Guaranteed by design.
4. Write/Erase durations are measured to valid Status Register (CSR) Data.
5. Word/byte write operations are typically performed with 1 Programming Pulse.
6. Address and Data are latched on the rising edge of WE# for all command write operations.
7. CE_X# is defined as the latter of CE_0# or CE_1# going low, or the first of CE_0# or CE_1# going high.
8. The TBD information will be available in a technical paper. Please contact Intel's Application Hotline or your local sales office for more information.



NOTES:

1. This address string depicts Data Write/Erase cycles with corresponding verification via ESRD.
2. This address string depicts Data Write/Erase cycles with corresponding verification via CSRD.
3. This cycle is invalid when using CSRD for verification during data write/erase operations.
4. CE_X# is defined as the latter of CE_0# or CE_1# going low or the first of CE_0# or CE_1# going high.
5. RP# low transition is only to show t_{RHPL}; not valid for above Read and Write cycles.
6. Data Write/Erase cycles are asynchronous; CLK and ADV# are ignored.
7. V_{PP} voltage during data write/erase operations valid at both 12.0V and 5.0V.
8. V_{PP} voltage equal to or below V_{PPLK} provides complete flash memory array protection.

Figure 14. AC Waveforms for WE#—Command Write Operations, Illustrating a Two Command Write Sequence Followed by an Extended Status Register Read

5.9 AC Characteristics for CE_X#—Controlled Write Operations(1)

V_{CC} = 3.3V ± 0.3V, T_A = 0°C to +70°C

Versions			28F016XS-20			28F016XS-25			
Symbol	Parameter	Notes	Min	Typ	Max	Min	Typ	Max	Unit
t_{AVAV}	Write Cycle Time		75			75			ns
t_{VPEH}1,2	V_{PP} Setup to CE_X# Going High	3,7	100			100			ns
t_{PHWL}	RP# Setup to WE# Going Low	3	480			480			ns
t_{WLEL}	WE# Setup to CE_X# Going Low	3,7	0			0			ns
t_{AVEH}	Address Setup to CE_X# Going High	2,6,7	60			60			ns
t_{DVEH}	Data Setup to CE_X# Going High	2,6,7	60			60			ns
t_{ELEH}	CE_X# Pulse Width	7	60			60			ns
t_{EHDX}	Data Hold from CE_X# High	2,7	10			10			ns
t_{EHAX}	Address Hold from CE_X# High	2,7	10			10			ns
t_{EHWH}	WE hold from CE_X# High	3,7	5			5			ns
t_{EHEL}	CE_X# Pulse Width High	7	15			15			ns
t_{GHEL}	Read Recovery before Write	3	0			0			ns
t_{EHRL}	CE_X# High to RY/BY# Going Low	3,7			100			100	ns
t_{RHPL}	RP# Hold from Valid Status Register (CSR, GSR, BSR) Data and RY/BY# High	3	0			0			ns
t_{PHEL}	RP# High Recovery to CE_X# Going Low	3,7	480			480			ns
t_{EHCH}	Write Recovery before Read		20			20			ns
t_{QVVL}1,2	V_{PP} Hold from Valid Status Register (CSR, GSR, BSR) Data and RY/BY# High	3	0			0			µs
t_{EHQV}1	Duration of Word/Byte Write Operation	3,4,5,8	5	9	TBD	5	9	TBD	µs
t_{EHQV}2	Duration of Block Erase Operation	3,4	0.6	1.6	20	0.6	1.6	20	sec

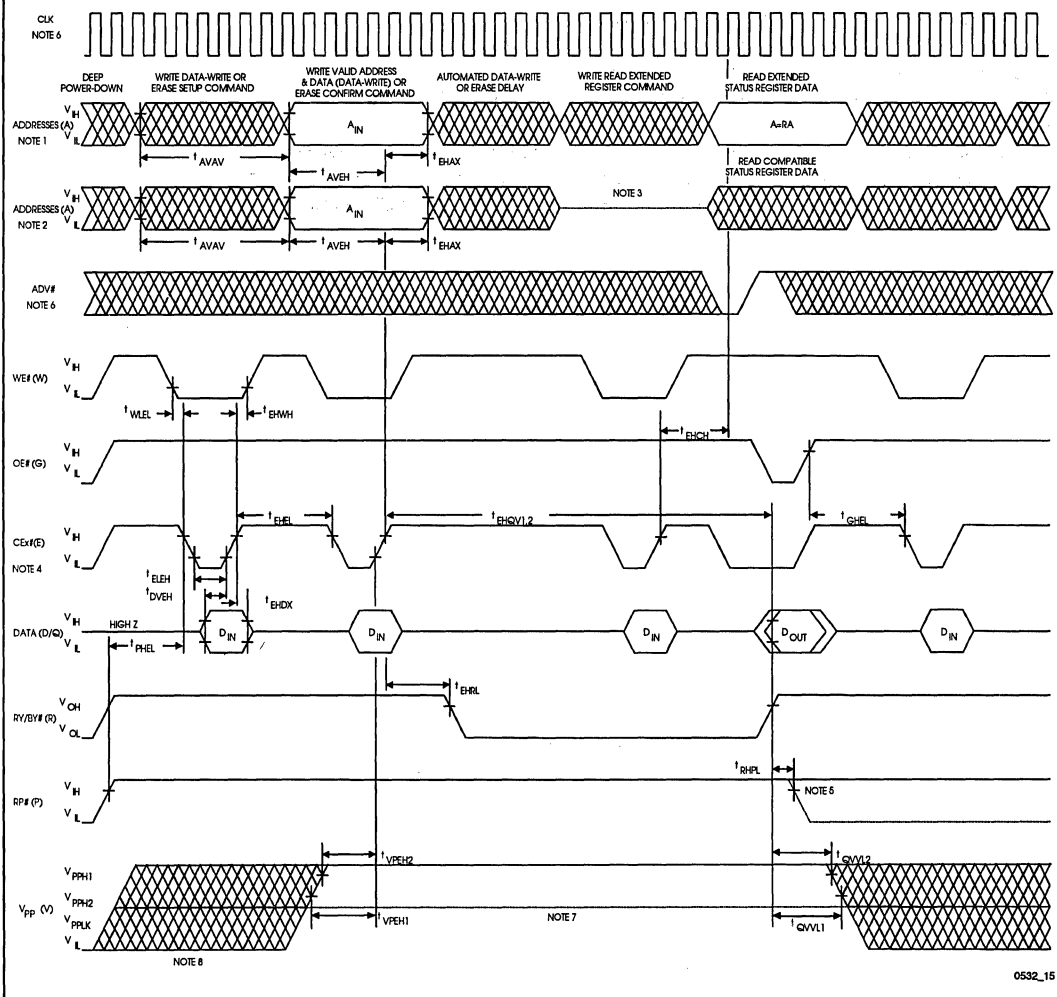
5.9 AC Characteristics for CE_X#—Controlled Write Operations[1] (Continued)

V_{CC} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Versions			28F016XS-15			28F016XS-20			
Symbol	Parameter	Notes	Min	Typ	Max	Min	Typ	Max	Unit
t_{AVAV}	Write Cycle Time		60			60			ns
t_{VPEH}1,2	V_{PP} Setup to CE_X# Going High	3,7	100			100			ns
t_{PHWL}	RP# Setup to WE# Going Low	3	300			300			ns
t_{WLEL}	WE# Setup to CE_X# Going Low	3,7	0			0			ns
t_{AVEH}	Address Setup to CE_X# Going High	2,6,7	45			45			ns
t_{DVEH}	Data Setup to CE_X# Going High	2,6,7	45			45			ns
t_{ELEH}	CE_X# Pulse Width	7	45			45			ns
t_{EHDX}	Data Hold from CE_X# High	2,7	0			0			ns
t_{EHAX}	Address Hold from CE_X# High	2,7	5			5			ns
t_{EHWH}	WE hold from CE_X# High	3,7	5			5			ns
t_{EHEL}	CE_X# Pulse Width High	7	15			15			ns
t_{GHEL}	Read Recovery before Write	3	0			0			ns
t_{EHRL}	CE_X# High to RY/BY# Going Low	3,7			100			100	ns
t_{RHPL}	RP# Hold from Valid Status Register (CSR, GSR, BSR) Data and RY/BY# High	3	0			0			ns
t_{PHEL}	RP# High Recovery to CE_X# Going Low	3,7	300			300			ns
t_{EHCH}	Write Recovery before Read		20			20			ns
t_{QVVL}1,2	V_{PP} Hold from Valid Status Register (CSR, GSR, BSR) Data and RY/BY# High	3	0			0			µs
t_{EHQV}1	Duration of Word/Byte Write Operation	3,4,5,8	4.5	6	TBD	4.5	6	TBD	µs
t_{EHQV}2	Duration of Block Erase Operation	3,4	0.6	1.2	20	0.6	1.2	20	sec

NOTES:

1. Read timings during write and erase are the same as for normal read.
2. Refer to command definition tables for valid address and data values.
3. Sampled, but not 100% tested. Guaranteed by design.
4. Write/Erase durations are measured to valid Status Register (CSR) Data.
5. Word/byte write operations are typically performed with 1 Programming Pulse.
6. Address and Data are latched on the rising edge of WE# for all command write operations.
7. CE_X# is defined as the latter of CE_0# or CE_1# going low, or the first of CE_0# or CE_1# going high.
8. The TBD information will be available in a technical paper. Please contact Intel's Application Hotline or your local sales office for more information.

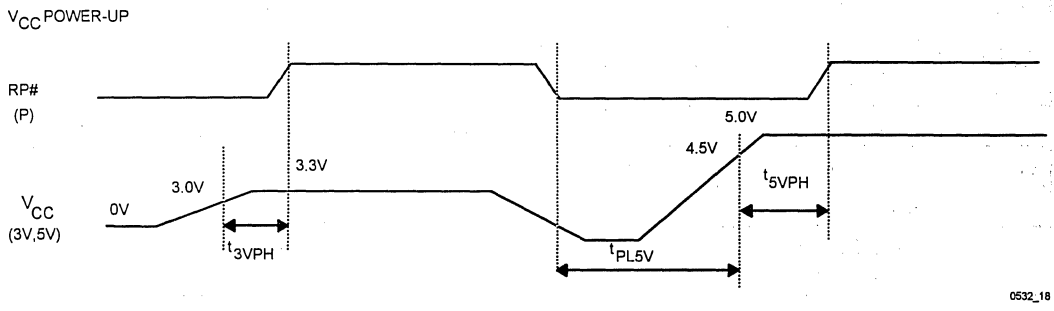


NOTES:

1. This address string depicts Data Write/Erase cycles with corresponding verification via ESRD.
2. This address string depicts Data Write/Erase cycles with corresponding verification via CSRD.
3. This cycle is invalid when using CSRD for verification during data write/erase operations.
4. CE_X# is defined as the latter of CE_0# or CE_1# going low or the first of CE_0# or CE_1# going high.
5. RP# low transition is only to show t_{RHPL}; not valid for above Read and Write cycles.
6. Data Write/Erase cycles are asynchronous; CLK and ADV# are ignored.
7. V_{PP} voltage during data write/erase operations valid at both 12.0V and 5.0V.
8. V_{PP} voltage equal to or below V_{PPLK} provides complete flash memory array protection.

Figure 15. AC Waveforms for CE_X#—Controlled Write Operations, Illustrating a Two Command Write Sequence Followed by an Extended Status Register Read

5.10 Power-Up and Reset Timings



NOTE:
For read timings following reset see Section 5.7.

Figure 16. V_{CC} Power-Up and RP# Reset Waveforms

Symbol	Parameter	Notes	Min	Max	Unit
t_{PL5V}	RP# Low to V_{CC} at 4.5V (Minimum)	2	0		µs
t_{PL3V}	RP# Low to V_{CC} at 3.0V (Minimum)	2	0		µs
t_{5VPH}	V_{CC} at 4.5V Minimum) to RP# High	1	2		µs
t_{3VPH}	V_{CC} at 3.0V (Minimum) to RP# High	1	2		µs

NOTES:
1. The t_{5VPH} and/or t_{3VPH} times must be strictly followed to guarantee all other read and write specifications for the 28F016XS.
2. The power supply may start to switch concurrently with RP# going low.

5.11 Erase and Word/Byte Write Performance[3,4]

V_{CC} = 3.3V ± 0.3V, V_{PP} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Symbol	Parameter	Notes	Min	Typ[1]	Max	Units	Test Conditions
$t_{WHRH}1A$	Byte Write Time	2,5	TBD	29	TBD	µs	
$t_{WHRH}1B$	Word Write Time	2,5	TBD	35	TBD	µs	
$t_{WHRH}2$	Block Write Time	2,5	TBD	3.8	TBD	sec	Byte Write Mode
$t_{WHRH}3$	Block Write Time	2,5	TBD	2.4	TBD	sec	Word Write Mode
	Block Erase Time	2,5	TBD	2.8	TBD	sec	
	Erase Suspend Latency Time to Read		1.0	12	75	µs	

V_{CC} = 3.3V ± 0.3V, V_{PP} = 12.0V ± 0.6V, T_A = 0°C to +70°C

Symbol	Parameter	Notes	Min	Typ[1]	Max	Units	Test Conditions
$t_{WHRH}1$	Word/Byte Write Time	2,5	5	9	TBD	µs	
$t_{WHRH}2$	Block Write Time	2,5	TBD	1.2	4.2	sec	Byte Write Mode
$t_{WHRH}3$	Block Write Time	2,5	TBD	0.6	2.0	sec	Word Write Mode
	Block Erase Time	2	0.6	1.6	20	sec	
	Erase Suspend Latency Time to Read		1.0	9	55	µs	

V_{CC} = 5.0V ± 0.5V, V_{PP} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Symbol	Parameter	Notes	Min	Typ(1)	Max	Units	Test Conditions
$t_{WHRH}1A$	Byte Write Time	2,5	TBD	20	TBD	µs	
$t_{WHRH}1B$	Word Write Time	2,5	TBD	25	TBD	µs	
$t_{WHRH}2$	Block Write Time	2,5	TBD	2.8	TBD	sec	Byte Write Mode
$t_{WHRH}3$	Block Write Time	2,5	TBD	1.7	TBD	sec	Word Write Mode
	Block Erase Time	2,5	TBD	2.0	TBD	sec	
	Erase Suspend Latency Time to Read		1.0	9	55	µs	

V_{CC} = 5.0V ± 0.5V, V_{PP} = 12.0V ± 0.6V, T_A = 0°C to +70°C

Symbol	Parameter	Notes	Min	Typ(1)	Max	Units	Test Conditions
$t_{WHRH}1$	Word/Byte Write Time	2,5	4.5	6	TBD	µs	
$t_{WHRH}2$	Block Write Time	2,5	TBD	0.8	4.2	sec	Byte Write Mode
$t_{WHRH}3$	Block Write Time	2,5	TBD	0.4	2.0	sec	Word Write Mode
	Block Erase Time	2	0.6	1.2	20	sec	
	Erase Suspend Latency Time to Read		1.0	7	40	µs	

NOTES:

1. 25°C, and nominal voltages.
2. Excludes system-level overhead.
3. These performance numbers are valid for all speed versions.
4. Sampled, but not 100% tested. Guaranteed by design.
5. The TBD information will be available in a technical paper. Please contact Intel's Application Hotline or your local sales office for more information.

6.0 MECHANICAL SPECIFICATIONS

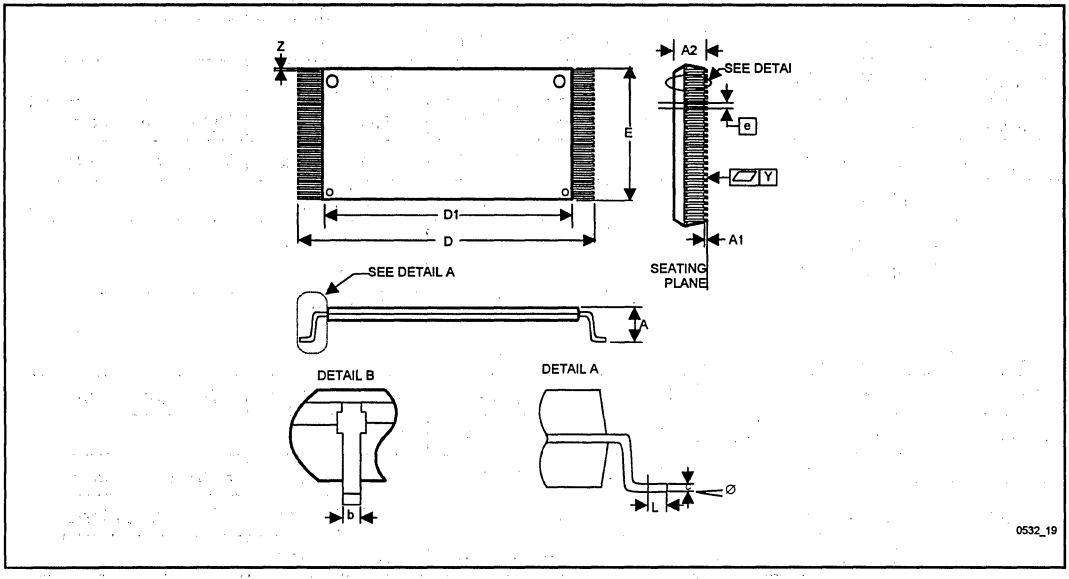
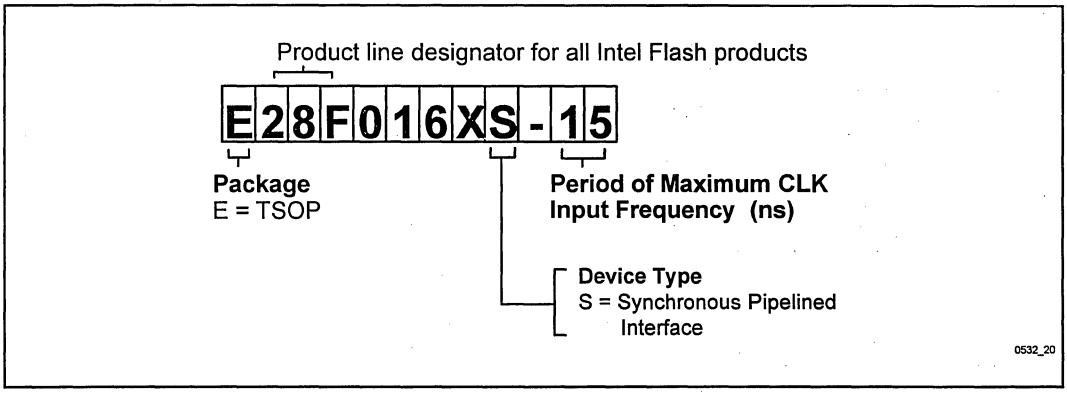


Figure 17. Mechanical Specifications of the 28F016XS 56-Lead TSOP Type I Package

Family: Thin Small Out-Line Package				
Symbol	Millimeters			Notes
	Minimum	Nominal	Maximum	
A			1.20	
A_1	0.50			
A_2	0.965	0.995	1.025	
b	0.100	0.150	0.200	
c	0.115	0.125	0.135	
D_1	18.20	18.40	18.60	
E	13.80	14.00	14.20	
e		0.50		
D	19.80	20.00	20.20	
L	0.500	0.600	0.700	
N		56		
Ø	0°	3°	5°	
Y			0.100	
Z	0.150	0.250	0.350	

DEVICE NOMENCLATURE AND ORDERING INFORMATION



Option	Order Code	Valid Combinations		
		V_{CC} = 3.3V ± 0.3V, 50 pF load, 1.5V I/O Levels(1)	V_{CC} = 5.0V ± 10%, 100 pF load TTL I/O Levels(1)	V_{CC} = 5.0V ± 10%, 30 pF load 1.5V I/O Levels(1)
1	E28F016XS15	28F016XS-20		28F016XS-15
2	E28F016XS20	28F016XS-25	28F016XS-20	

ADDITIONAL INFORMATION

Order Number	Document/Tool
297372	16-Mbit Flash Product Family User's Manual
292165	AB-62, "Compiling Optimized Code for Embedded Flash RAM Memories"
292126	AP-360, "16-Mbit Flash Product Family Software Drivers, 28F016SA/SV/XD/XS"
292147	AP-398, "Designing with the 28F016XS"
292146	AP-600, "Performance Benefits and Power/Energy Savings of 28F016XS Based System Designs"
292163	AP-610, "Flash Memory In-System Code and Data Update Techniques"
297500	"Interfacing the 28F016XS to the i960® Microprocessor Family"
297504	"Interfacing the 28F016XS to the Intel486™ Microprocessor Family"
294016	ER-33, "ETOX™ Flash Memory Technology—Insight to Intel's Fourth Generation Process Innovation"
297508	FLASHBuilder Utility
Contact Intel/Distribution Sales Office	28F016XS Benchmark Utility
Contact Intel/Distribution Sales Office	Flash Cycling Utility
Contact Intel/Distribution Sales Office	28F016XS iBIS Models
Contact Intel/Distribution Sales Office	28F016XS VHDL/Verilog Models
Contact Intel/Distribution Sales Office	28F016XS Timing Designer Library Files
Contact Intel/Distribution Sales Office	28F016XS Orcad/Viewlogic Schematic Symbols

DATASHEET REVISION HISTORY

Number	Description
001	Original Version
002	Removed support of the following features: • All page buffer operations (read, write, programming, Upload Device Information) • Command queuing • Software Sleep and Abort • Erase all Unlocked Blocks and Two-Byte Write • RY/BY# Configuration as part of the Device Configuration command Changed definition of "NC." Removed "No internal connection to die" from description. Added "xx" to Upper Byte of Command (Data) Definition in Sections 4.3 and 4.4. Modified parameters "V" and "I" of Section 5.1 to apply to "NC" pins. Increased I_{PPR} (V_{PP} Read Current) for $V_{PP} > V_{CC}$ to 200 µA at V_{CC} = 3.3V/5.0V. Changed V_{CC} = 5.0V DC Characteristics (Section 5.5) marked with Note 1 to indicate that these currents are specified for a CMOS rise/fall time (10% to 90%) of <5 ns and a TTL rise/fall time of <10 ns. Corrected t_{PHCH} (RP# High to CLK) to be a "Min" specification at V_{CC} = 3.3V/5.0V. Corrected the graphical representation of t_{WHCH} and t_{EHCH} in Figures 14 and 15. Increased Typical "Byte/Word Write Times" (t_{WHRH1A}/t_{WHRH1B}) for V_{PP} = 5.0V (Sec. 5.13): t_{WHRH1A} from 16.5 µs to 29.0 µs and t_{WHRH1B} from 24.0 µs to 35.0 µs at V_{CC} = 3.3V t_{WHRH1A} from 11.0 µs to 20.0 µs and t_{WHRH1B} from 16.0 µs to 25.0 µs at V_{CC} = 5.0V. Increased Typical "Block Write Times" (t_{WHRH2}/ t_{WHRH3}) for V_{PP} = 5.0V (Section 5.13): t_{WHRH2} from 2.2 sec to 3.8 sec and t_{WHRH3} from 1.6 sec to 2.4 sec at V_{CC} = 3.3V t_{WHRH2} from 1.6 sec to 2.8 sec and t_{WHRH3} from 1.2 sec to 1.7 sec at V_{CC} = 5.0V. Changed "Time from Erase Suspend Command to WSM Ready" spec name to "Erase Suspend Latency Time to Read"; Modified typical values and Added Min/Max values at V_{CC} =3.3/5.0V and V_{PP} =5.0/12.0V (Section 5.13). Minor cosmetic changes throughout document.

ADVANCE INFORMATION

28F016XD
16-MBIT (1 MBIT x 16)
DRAM-INTERFACE FLASH MEMORY

- 85 ns Access Time (t_{RAC})
  - — Supports both Standard and Fast-Page-Mode Accesses
- Multiplexed Address Bus
  - — RAS# and CAS# Control Inputs
- No-Glue Interface to Many Memory Controllers
- SmartVoltage Technology
  - — User-Selectable 3.3V or 5V V_{CC}
  - — User-Selectable 5V or 12V V_{PP}
- 0.33 MB/sec Write Transfer Rate
- x16 Architecture
- 56-Lead TSOP Type I Package
- Backwards-Compatible with 28F008SA Command Set
- 2 µA Typical Deep Power-Down Current
- 1 mA Typical I_{CC} Active Current in Static Mode
- 32 Separately-Erasable/Lockable 64-Kbyte Blocks
- 1 Million Erase Cycles per Block
- State-of-the-Art 0.6 µm ETOX™ IV Flash Technology

Intel's 28F016XD 16-Mbit Flash memory is a revolutionary architecture which is the ideal choice for designing truly revolutionary high-performance products. Combining its DRAM-like read performance and interface with the intrinsic nonvolatility of flash memory, the 28F016XD eliminates the traditional redundant memory paradigm of shadowing code from a slow nonvolatile storage source to a faster execution memory, such as DRAM, for improved system performance. The innovative capabilities of the 28F016XD enable the design of direct-execute code and mass storage data/file flash memory systems.

The 28F016XD's DRAM-like interface with a multiplexed address bus, flexible V_{CC} and V_{PP} voltages, power saving features, extended cycling, fast write and read performance, symmetrically blocked architecture, and selective block locking provide a highly flexible memory component suitable for resident flash component arrays on the system board or SIMMs. The DRAM-like interface with RAS# and CAS# control inputs allows for easy migration to flash memory in existing DRAM-based systems. The 28F016XD's dual read voltage allows the same component to operate at either 3.3V or 5.0V V_{CC}. Programming voltage at 5V V_{PP} minimizes external circuitry in minimal-chip, space critical designs, while the 12V V_{PP} option maximizes write/erase performance. The x16 architecture allows optimization of the memory-to-processor interface. Its high read performance combined with flexible block locking enable both storage and execution of operating systems/application software and fast access to large data tables. The 28F016XD is manufactured on Intel's 0.6 µm ETOX™ IV process technology.

Order Number 290533-002

1.0 INTRODUCTION

The documentation of the Intel 28F016XD flash memory device includes this datasheet, a detailed user's manual, and a number of application notes and design tools, all of which are referenced at the end of this datasheet.

The datasheet is intended to give an overview of the chip feature-set and of the operating AC/DC specifications. The 16-Mbit Flash Product Family User's Manual provides complete descriptions of the user modes, system interface examples and detailed descriptions of all principles of operation. It also contains the full list of software algorithm flowcharts, and a brief section on compatibility with the Intel 28F008SA.

Significant 28F016XD feature revisions occurred between datasheet revisions 290533-001 and 290533-002. These revisions center around removal of the following features:

- All page buffer operations (read, write, programming, Upload Device Information)
- Command queuing
- Software Sleep and Abort
- Erase all Unlocked Blocks
- Device Configuration command

Intel recommends that all customers obtain the latest revisions of 28F016XD documentation.

1.1 Product Overview

The 28F016XD is a high-performance, 16-Mbit (16,777,216-bit) block erasable, nonvolatile random access memory, organized as 1 Mword x 16. The 28F016XD includes thirty-two 32-KW (32,768 word) blocks. A chip memory map is shown in Figure 3.

The implementation of a new architecture, with many enhanced features, will improve the device operating characteristics and result in greater product reliability and ease-of-use as compared to other flash memories. Significant features of the 28F016XD include:

- No-Glue Interface to Memory Controllers
- Improved Word Write Performance
- SmartVoltage Technology
  - — Selectable 3.3V or 5.0V V_{CC}
  - — Selectable 5.0V or 12.0V V_{PP}
- Internal 3.0V/5.0V V_{CC} Detection Circuitry
- Block Write/Erase Protection

The 28F016XD's multiplexed address bus with RAS# and CAS# inputs allows for a "No Glue" interface to many existing in-system memory controllers. As such, 28F016XD-based SIMMs (72-pin JEDEC Standard) offer attractive advantages over their DRAM counterparts in many applications. For more information on 28F016XD-based SIMM designs, see the application note referenced at the end of this datasheet.

The 28F016XD incorporates SmartVoltage technology, providing V_{CC} operation at both 3.3V and 5.0V and program and erase capability at V_{PP} = 12.0V or 5.0V. Operating at V_{CC} = 3.3V, the 28F016XD consumes less than 60% of the power consumption at 5.0V V_{CC}, while 5.0V V_{CC} provides the highest read performance capability. V_{PP} = 5.0V operation eliminates the need for a separate 12.0V converter, while V_{PP} = 12.0V maximizes write/erase performance. In addition to the flexible program and erase voltages, the dedicated V_{PP} gives complete code protection with $V_{PP} \leq V_{PPLK}$.

Internal 3.3V or 5.0V V_{CC} detection automatically configures the device for optimized 3.3V or 5.0V read/write operation.

A Command User Interface (CUI) serves as the system interface between the microprocessor or microcontroller and the internal memory operation.

Internal Algorithm Automation allows word writes and block erase operations to be executed using a Two-Write command sequence to the CUI in the same way as the 28F008SA 8-Mbit FlashFile™ memory.

Software Locking of Memory Blocks is an added feature of the 28F016XD as compared to the 28F008SA. The 28F016XD provides selectable block locking to protect code or data such as direct-executable operating systems or application code. Each block has an associated nonvolatile lock-bit which determines the lock status of the block. In addition, the 28F016XD has a master Write Protect pin (WP#) which prevents any modifications to memory blocks whose lock-bits are set.

Writing of memory data is performed in word increments typically within 6 µsec (12.0V V_{PP})–a 33% improvement over the 28F008SA. A block erase operation erases one of the 32 blocks in typically 0.6 sec (12.0V V_{PP}), independent of the other blocks, which is about a 65% improvement over the 28F008SA.

Each block can be written and erased a minimum of 100,000 cycles. Systems can achieve one million Block Erase Cycles by providing wear-leveling algorithms and graceful block retirement. These techniques have already been employed in many flash file systems and hard disk drive designs.

All operations are started by a sequence of Write commands to the device. Three types of Status Registers (described in detail later in this datasheet) and a RY/BY# output pin provide information on the progress of the requested operation.

The following Status Registers are used to provide device and WSM information to the user :

- A Compatible Status Register (CSR) which is 100% compatible with the 28F008SA FlashFile memory Status Register. The CSR, when used alone, provides a straightforward upgrade capability to the 28F016XD from a 28F008SA-based design.
- A Global Status Register (GSR) which also informs the system of overall Write State Machine (WSM) status.
- 32 Block Status Registers (BSRs) which provide block-specific status information such as the block lock-bit status.

The GSR and BSR memory maps are shown in Figure 4.

The 28F016XD incorporates an open drain RY/BY# output pin. This feature allows the user to OR-tie many RY/BY# pins together in a multiple memory configuration such as a Resident Flash Array.

The 28F016XD is specified for a maximum fast page mode cycle time of 65 ns ($t_{PC,R}$) at 5.0V operation (4.75V to 5.25V) over the commercial temperature range (0°C to +70°C). A corresponding maximum fast page mode cycle time of 75 ns at 3.3V (3.0V to 3.6V and 0°C to +70°C) is achieved for reduced power consumption applications.

The 28F016XD incorporates an Automatic Power Saving (APS) feature, which substantially reduces the active current when the device is in static mode of operation (addresses not switching). In APS mode, the typical I_{CC} current is 1 mA at 5.0V (3.0 mA at 3.3V).

A deep power-down mode of operation is invoked when the RP# (called PWD# on the 28F008SA) pin transitions low. This mode brings the device power consumption to less than 2.0 µA, typically, and provides additional write protection by acting as a device reset pin during power transitions. A reset time of 300 ns (5.0V V_{CC} operation) is required from RP# switching high until dropping RAS#. In the Deep Power-Down state, the WSM is reset (any current operation will abort) and the CSR, GSR and BSR registers are cleared.

A CMOS standby mode of operation is enabled when RAS# and CAS# transition high and RP# stays high with all input control pins at CMOS levels. In this mode, the device typically draws an I_{CC} standby current of 70 µA at 5V V_{CC}.

The 28F016XD is available in a 56-Lead, 1.2mm thick, 14mm x 20mm TSOP Type I package. This form factor and pinout allow for very high board layout densities.

2.0 DEVICE PINOUT

The 28F016XD 56-Lead TSOP Type I pinout configuration is shown in Figure 2.

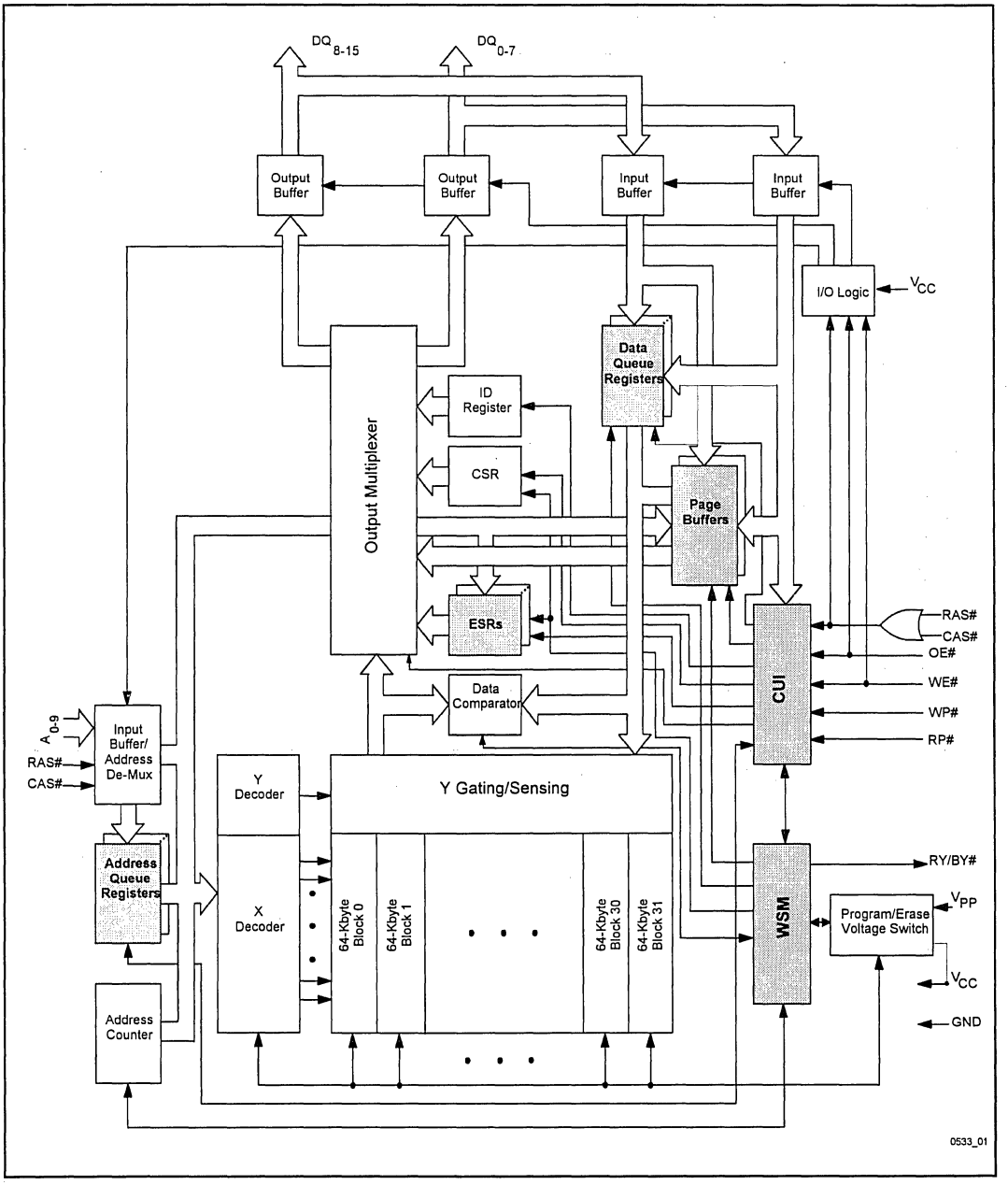


Figure 1. 28F016XD Block Diagram
Architectural Evolution Includes Multiplexed Address Bus, SmartVoltage Technology, and Extended Registers

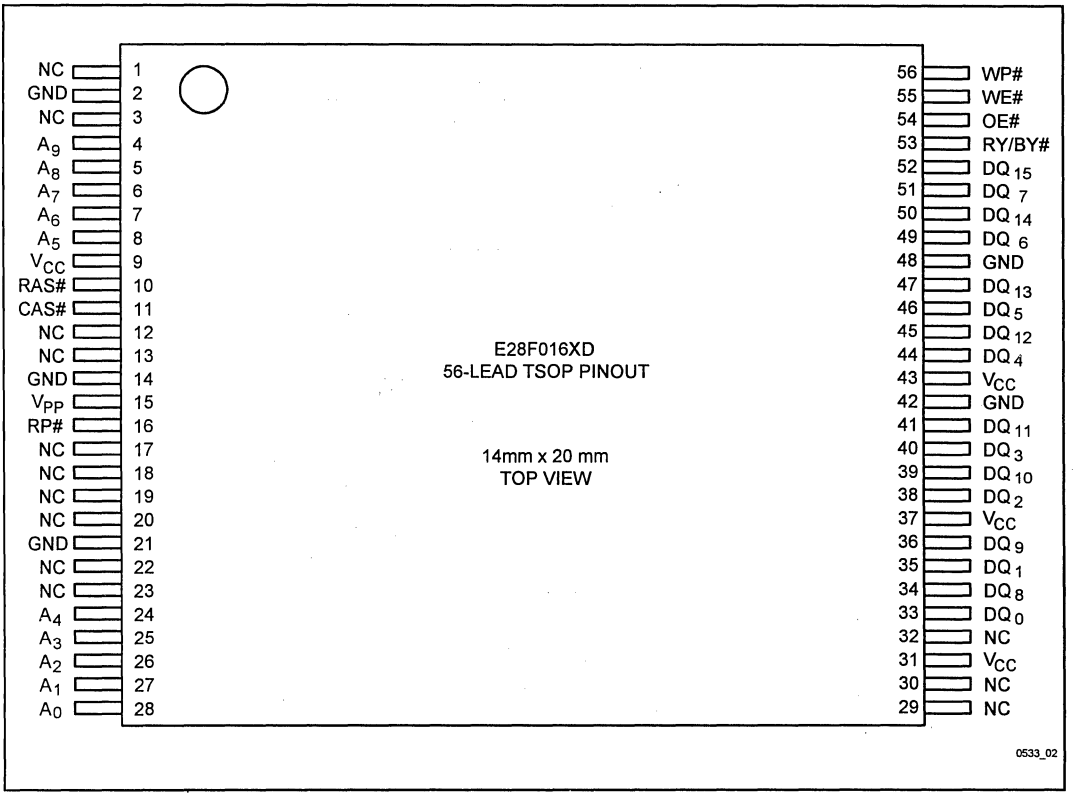


Figure 2. 28F016XD 56-Lead TSOP Type I Pinout Configuration

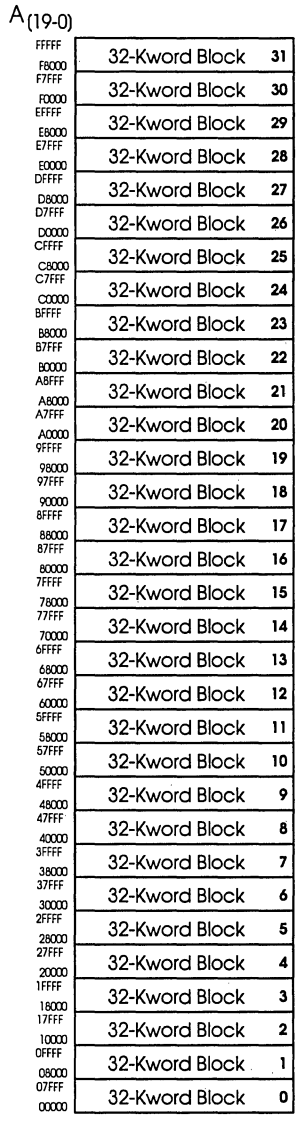
2.1 Lead Descriptions

Symbol	Type	Name and Function
A_0–A_9	INPUT	**MULTIPLEXED ROW/COLUMN ADDRESSES:** Selects a word within one of thirty-two 32-Kword blocks. Row (upper) addresses are latched on the falling edge of RAS#, while column (lower) addresses are latched on the falling edge of CAS#.
DQ_0–DQ_{15}	INPUT/OUTPUT	**DATA BUS:** Inputs data and commands during CUI write cycles. Outputs array, identifier or status data (DQ_{0-7}) in the appropriate read mode. Floated when the chip is de-selected or the outputs are disabled.
RAS#	INPUT	**ROW ADDRESS STROBE:** Latches row address information on inputs A_{9-0} when RAS# transitions low. A subsequent CAS# low transition initiates 28F016XD read or write operations.
CAS#	INPUT	**COLUMN ADDRESS STROBE:** Latches column address information on inputs A_{9-0} when CAS# transitions low. When preceded by a RAS# low transition, CAS# low initiates 28F016XD read or write operations, along with OE# and WE#. Subsequent CAS# low transitions, with RAS# held low, enable fast page mode reads/writes
RP#	INPUT	**RESET/POWER-DOWN:** RP# low places the device in a Deep Power-Down state. All circuits that consume static power, even those circuits enabled in standby mode, are turned off. When returning from Deep Power-Down, a recovery time of 300 ns at 5.0V V_{CC} is required to allow these circuits to power-up. When RP# goes low, the current WSM operation is terminated, and the device is reset. All Status Registers return to ready (with all status flags cleared). Exit from Deep Power-Down places the device in read array mode.
OE#	INPUT	**OUTPUT ENABLE:** Gates device data through the output buffers when low in combination with RAS# and CAS# low. The outputs float to tri-state off when OE# is high. OE# can be tied to GND if not controlled by the system memory controller. RAS# and CAS# high override OE# low. WE# low also overrides OE# low.
WE#	INPUT	**WRITE ENABLE:** Controls access to the CUI, Data Register and Address Latch. WE# is active low and initiates writes in combination with RAS# and CAS# low. WE# low overrides OE# low. RAS# and CAS# high override WE# low.
RY/BY#	OPEN DRAIN OUTPUT	**READY/BUSY:** Indicates status of the internal WSM. When low, it indicates that the WSM is busy performing an operation. RY/BY# floating indicates that the WSM is ready for new operations, Erase is Suspended, or the device is in deep power-down mode. This output is always active (i.e., not floated to tri-state off when OE#, RAS# or CAS# are high).
WP#	INPUT	**WRITE PROTECT:** Erase blocks can be locked by writing a nonvolatile lock-bit for each block. When WP# is low, those locked blocks as reflected by the Block-Lock Status bits (BSR.6), are protected from inadvertent Data Writes or Erases. When WP# is high, all blocks can be written or erased regardless of the state of the lock-bits. The WP# input buffer is disabled when RP# transitions low (deep power-down mode).

2.1 Lead Descriptions (Continued)

Symbol	Type	Name and Function
V_{PP}	SUPPLY	**WRITE/ERASE POWER SUPPLY (12.0V ± 0.6V, 5.0V ± 0.5V):** For erasing memory array blocks or writing words into the flash array. V_{PP} = 5.0V ± 0.5V eliminates the need for a 12V converter, while connection to 12.0V ± 0.6V maximizes Write/Erase Performance. **NOTE:** Successful completion of write and erase attempts is inhibited with V_{PP} at or below 1.5V. Write and Erase attempts with V_{PP} between 1.5V and 4.5V, between 5.5V and 11.4V, and above 12.6V produce spurious results and should not be attempted.
V_{CC}	SUPPLY	**DEVICE POWER SUPPLY (3.3V ± 0.3V, 5.0V ± 0.5V):** Internal detection configures the device for 3.3V or 5.0V operation. To switch 3.3V to 5.0V (or vice versa), first ramp V_{CC} down to GND, and then power to the new V_{CC} voltage. Do not leave any power pins floating.
GND	SUPPLY	**GROUND FOR ALL INTERNAL CIRCUITRY:** Do not leave any ground pins floating.
NC		**NO CONNECT:** Lead may be driven or left floating.

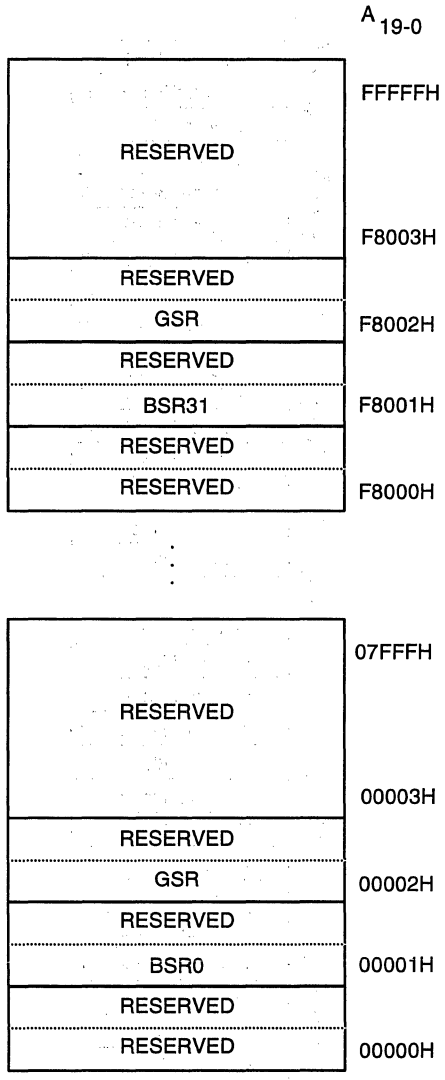
3.0 MEMORY MAPS



NOTE:
The upper 10 bits (A_{19-10}) reflect 28F016XD addresses A_{9-0}, latched by RAS#.
The lower 10 bits (A_{9-0}) reflect 28F016XD addresses A_{9-0}, latched by CAS#.

Figure 3. 28F016XD Memory Map

3.1 Extended Status Registers Memory Map



NOTE:
The upper 10 bits (A_{19-10}) reflect 28F016XD addresses A_{9-0}, latched by RAS#.
The lower 10 bits (A_{9-0}) reflect 28F016XD addresses A_{9-0}, latched by CAS#.

Figure 4. Extended Status Registers Memory Map

4.0 BUS OPERATIONS, COMMANDS AND STATUS REGISTER DEFINITIONS

4.1 Bus Operations

Mode	Notes	RP#	RAS#	CAS#	OE#	WE#	DQ_{0-15}	RY/BY#
Row Address Latch	1,2,9	V_{IH}	↓	V_{IH}	X	X	X	X
Column Address Latch	1,2,9	V_{IH}	V_{IL}	↓	X	X	X	X
Read	1,2,7	V_{IH}	V_{IL}	V_{IL}	V_{IL}	V_{IH}	D_{OUT}	X
Output Disable	1,6,7	V_{IH}	V_{IL}	V_{IL}	V_{IH}	V_{IH}	High Z	X
Standby	1,6,7	V_{IH}	V_{IH}	V_{IH}	X	X	High Z	X
Deep Power-Down	1,3	V_{IL}	X	X	X	X	High Z	V_{OH}
Manufacturer ID	4,8	V_{IH}	V_{IL}	V_{IL}	V_{IL}	V_{IH}	0089H	V_{OH}
Device ID	4,8	V_{IH}	V_{IL}	V_{IL}	V_{IL}	V_{IH}	66A8H	V_{OH}
Write	1,5,6	V_{IH}	V_{IL}	V_{IL}	X	V_{IL}	D_{IN}	X

NOTES:

1. X can be V_{IH} or V_{IL} for address or control pins except for RY/BY#, which is either V_{OL} or V_{OH}, or High Z or D_{OUT} for data pins depending on whether or not OE# is active.
2. RY/BY# output is open drain. When the WSM is ready, Erase is suspended or the device is in deep power-down mode, RY/BY# will be at V_{OH} if it is tied to V_{CC} through a resistor. RY/BY# at V_{OH} is independent of OE# while a WSM operation is in progress.
3. RP# at GND ± 0.2V ensures the lowest deep power-down current.
4. A_0 (latched by CAS#) at V_{IL} provides the Manufacturer ID code. A_0 (latched by CAS#) at V_{IH} provides the Device ID code. All other addresses (row and column) should be set to zero.
5. Commands for erase, data write, or lock-block operations can only be completed successfully when $V_{PP} = V_{PPH1}$ or $V_{PP} = V_{PPH2}$.
6. While the WSM is running, RY/BY# stays at V_{OL} until all operations are complete. RY/BY# goes to V_{OH} when the WSM is not busy or in erase suspend mode.
7. RY/BY# may be at V_{OL} while the WSM is busy performing various operations (for example, a Status Register read during a write operation).
8. The 28F016XD shares an identical device identifier with the 28F016XS.
9. Row (upper) addresses are latched via inputs A_{0-9} on the falling edge of RAS#. Column (lower) addresses are latched via inputs A_{0-9} on the falling edge of CAS#. Row addresses must be latched before column addresses are latched.

4.2 28F008SA—Compatible Mode Command Bus Definitions

Command	Notes	First Bus Cycle			Second Bus Cycle		
		Oper	Addr	Data[4]	Oper	Addr	Data[4]
Read Array		Write	X	xxFFH	Read	AA	AD
Intelligent Identifier	1	Write	X	xx90H	Read	IA	ID
Read Compatible Status Register	2	Write	X	xx70H	Read	X	CSRD
Clear Status Register	3	Write	X	xx50H			
Word Write		Write	X	xx40H	Write	WA	WD
Alternate Word Write		Write	X	xx10H	Write	WA	WD
Block Erase/Confirm		Write	X	xx20H	Write	BA	xxD0H
Erase Suspend/Resume		Write	X	xxB0H	Write	X	xxD0H

ADDRESS
AA = Array Address
BA = Block Address
IA = Identifier Address
WA = Write Address
X = Don't Care

DATA
AD = Array Data
CSRD = CSR Data
ID = Identifier Data
WD = Write Data

NOTES:
1. Following the Intelligent Identifier command, two read operations access the manufacturer and device signature codes.
2. The CSR is automatically available after device enters data write, erase, or suspend operations.
3. Clears CSR.3, CSR.4 and CSR.5. Also clears GSR.5 and all BSR.5, BSR.4 and BSR.2 bits. See Status Register definitions.
4. The upper byte of the data bus (D_{8-15}) during command writes is a "Don't Care."

4.3 28F016XD—Enhanced Command Bus Definitions

Command	Notes	First Bus Cycle			Second Bus Cycle		
		Oper	Addr	Data[3]	Oper	Addr	Data[3]
Read Extended Status Register	1	Write	X	xx71H	Read	RA	GSRD BSRD
Lock Block/Confirm		Write	X	xx77H	Write	BA	xxD0H
Upload Status Bits/Confirm	2	Write	X	xx97H	Write	X	xxD0H

ADDRESS
BA = Block Address
RA = Extended Register Address
WA = Write Address
X = Don't Care

DATA
AD = Array Data
BSRD = BSR Data
GSRD = GSR Data

NOTES:
1. RA can be the GSR address or any BSR address. See Figure 4 for the Extended Status Register memory map.
2. Upon device power-up, all BSR lock-bits come up locked. The Upload Status Bits command must be written to reflect the actual lock-bit status.
3. The upper byte of the data bus (D_{8-15}) during command writes is a "Don't Care."

4.4 Compatible Status Register

WSMS	ESS	ES	DWS	VPPS	R	R	R
7	6	5	4	3	2	1	0

CSR.7 = WRITE STATE MACHINE STATUS
- 1 = Ready
- 0 = Busy

CSR.6 = ERASE-SUSPEND STATUS
- 1 = Erase Suspended
- 0 = Erase In Progress/Completed

CSR.5 = ERASE STATUS
- 1 = Error In Block Erasure
- 0 = Successful Block Erase

CSR.4 = DATA-WRITE STATUS
- 1 = Error in Data Write
- 0 = Data Write Successful

CSR.3 = V_{PP} STATUS
- 1 = V_{PP} Error Detect, Operation Abort
- 0 = V_{PP} OK

CSR.2-0 = RESERVED FOR FUTURE ENHANCEMENTS
These bits are reserved for future use; mask them out when polling the CSR.

NOTES:

RY/BY# output or WSMS bit must be checked to determine completion of an operation (Erase, Erase Suspend, or Data Write) before the appropriate Status bit (ESS, ES or DWS) is checked for success.

If DWS and ES are set to "1" during an erase attempt, an improper command sequence was entered. Clear the CSR and attempt the operation again.

The VPPS bit, unlike an A/D converter, does not provide continuous indication of V_{PP} level. The WSM interrogates V_{PP}'s level only after the Data Write or Erase command sequences have been entered, and informs the system if V_{PP} has not been switched on. VPPS is not guaranteed to report accurate feedback between V_{PPLK}(max) and V_{PPH1}(min), between V_{PPH1}(max) and V_{PPH2}(min) and above V_{PPH2}(max).

4.5 Global Status Register

WSMS	OSS	DOS	R	R	R	R	R
7	6	5	4	3	2	1	0

GSR.7 = WRITE STATE MACHINE STATUS
1 = Ready
0 = Busy

GSR.6 = OPERATION SUSPEND STATUS
1 = Operation Suspended
0 = Operation in Progress/Completed

GSR.5 = DEVICE OPERATION STATUS
1 = Operation Unsuccessful
0 = Operation Successful or Currently Running

GSR.4-0 = RESERVED FOR FUTURE ENHANCEMENTS
These bits are reserved for future use; mask them out when polling the GSR.

NOTES:
RY/BY# output or WSMS bit must be checked to determine completion of an operation (Block Lock, Suspend, Upload Status Bits, Erase or Data Write) before the appropriate Status bit (OSS or DOS) is checked for success.

4.6 Block Status Register

BS	BLS	BOS	R	R	VPPS	VPPL	R
7	6	5	4	3	2	1	0

BSR.7 = BLOCK STATUS
1 = Ready
0 = Busy

BSR.6 = BLOCK LOCK STATUS
1 = Block Unlocked for Write/Erase
0 = Block Locked for Write/Erase

BSR.5 = BLOCK OPERATION STATUS
1 = Operation Unsuccessful
0 = Operation Successful or Currently Running

BSR.2 = V_{PP} STATUS
1 = V_{PP} Error Detect, Operation Abort
0 = V_{PP} OK

BSR.1 = V_{PP} LEVEL
1 = V_{PP} Detected at 5.0V ± 10%
0 = V_{PP} Detected at 12.0V ± 5%

NOTES:
RY/BY# output or BS bit must be checked to determine completion of an operation (Block Lock, Suspend, Erase or Data Write) before the appropriate Status bits (BOS, BLS) is checked for success.

BSR.1 is not guaranteed to report accurate feedback between the V_{PPH1} and V_{PPH2} voltage ranges. Writes and erases with V_{PP} between V_{PPLK}(max) and V_{PPH1} (min), between V_{PPH1}(max) and V_{PPH2}(min), and above V_{PPH2}(max) produce spurious results and should not be attempted.
BSR.1 was a RESERVED bit on the 28F016SA.

BSR.4,3,0 = RESERVED FOR FUTURE ENHANCEMENTS
These bits are reserved for future use; mask them out when polling the BSRs.

5.0 ELECTRICAL SPECIFICATIONS

NOTICE: This datasheet contains information on products in the sampling and initial production phases of development. The specifications are subject to change without notice. Verify with your local Intel Sales office that you have the latest datasheet before finalizing a design.

5.1 Absolute Maximum Ratings*

Temperature Under Bias..... 0°C to +80°C

Storage Temperature..... –65°C to +125°C

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

V_{CC} = 3.3V ± 0.3V Systems

Sym	Parameter	Notes	Min	Max	Units	Test Conditions
T_A	Operating Temperature, Commercial	1	0	70	°C	Ambient Temperature
V_{CC}	V_{CC} with Respect to GND	2	–0.2	7.0	V	
V_{PP}	V_{PP} Supply Voltage with Respect to GND	2,3	–0.2	14.0	V	
V	Voltage on any Pin (except V_{CC},V_{PP}) with Respect to GND	2,5	–0.5	V_{CC} + 0.5	V	
I	Current into any Non-Supply Pin	5		± 30	mA	
I_{OUT}	Output Short Circuit Current	4		100	mA	

V_{CC} = 5.0V ± 0.5V Systems

Sym	Parameter	Notes	Min	Max	Units	Test Conditions
T_A	Operating Temperature, Commercial	1	0	70	°C	Ambient Temperature
V_{CC}	V_{CC} with Respect to GND	2	–0.2	7.0	V	
V_{PP}	V_{PP} Supply Voltage with Respect to GND	2,3	–0.2	14.0	V	
V	Voltage on any Pin (except V_{CC},V_{PP}) with Respect to GND	2,5	–2.0	7.0	V	
I	Current into any Non-Supply Pin	5		± 30	mA	
I_{OUT}	Output Short Circuit Current	4		100	mA	

NOTES:

1. Operating temperature is for commercial product defined by this specification.
2. Minimum DC voltage is -0.5V on input/output pins. During transitions, this level may undershoot to -2.0V for periods <20 ns. Maximum DC voltage on input/output pins is V_{CC} + 0.5V which, during transitions, may overshoot to V_{CC} + 2.0V for periods <20 ns.
3. Maximum DC voltage on V_{PP} may overshoot to +14.0V for periods <20 ns.
4. Output shorted for no more than one second. No more than one output shorted at a time.
5. This specification also applies to pins marked "NC."

5.2 Capacitance

For a 3.3V ± 0.3V System:

Sym	Parameter	Notes	Typ	Max	Units	Test Conditions
C_{IN}	Capacitance Looking into an Address/Control Pin	1	6	8	pF	T_A = 25°C, f = 1.0 MHz
C_{OUT}	Capacitance Looking into an Output Pin	1	8	12	pF	T_A = 25°C, f = 1.0 MHz
C_{LOAD}	Load Capacitance Driven by Outputs for Timing Specifications	1,2		50	pF	

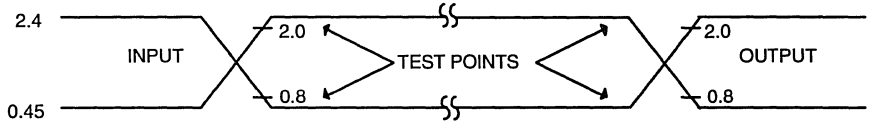
For 5.0V ± 0.5V System:

Sym	Parameter	Notes	Typ	Max	Units	Test Conditions
C_{IN}	Capacitance Looking into an Address/Control Pin	1	6	8	pF	T_A = 25°C, f = 1.0 MHz
C_{OUT}	Capacitance Looking into an Output Pin	1	8	12	pF	T_A = 25°C, f = 1.0 MHz
C_{LOAD}	Load Capacitance Driven by Outputs for Timing Specifications	1,2		100	pF	

NOTE:

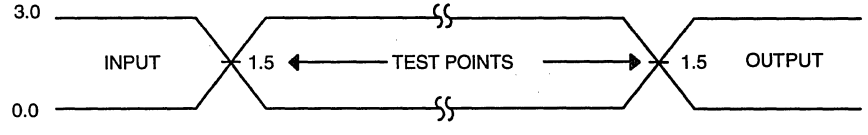
1. Sampled, not 100% tested.
2. To obtain iBIS models for the 28F016XD, please contact your local Intel/Distribution Sales Office.

5.3 Transient Input/Output Reference Waveforms



AC test inputs are driven at V_{OH} (2.4 VTTL) for a Logic "1" and V_{OL} (0.45 VTTL) for a Logic "0." Input timing begins at V_{IH} (2.0 VTTL) and V_{IL} (0.8 VTTL). Output timing ends at V_{IH} and V_{IL}. Input rise and fall times (10% to 90%) <10 ns.

Figure 5. Transient Input/Output Reference Waveform for V_{CC} = 5.0V ± 0.5V(1)



AC test inputs are driven at 3.0V for a Logic "1" and 0.0V for a Logic "0." Input timing begins, and output timing ends, at 1.5V. Input rise and fall times (10% to 90%) <10 ns.

Figure 6. Transient Input/Output Reference Waveform for V_{CC} = 3.3V ± 0.3V(2)

NOTES:

1. Testing characteristics for 28F016XD-85.
2. Testing characteristics for 28F016XD-95.

5.4 DC Characteristics

V_{CC} = 3.3V ± 0.3V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ	Max	Unit	Test Condition
$I_{CC}1$	V_{CC} Word Read Current	1,4,5		50	70	mA	$V_{CC} = V_{CC}$ Max RAS#, CAS# = V_{IL} RAS#, CAS#, Addr. Cycling @ t_{RC} = min I_{OUT} = 0 mA Inputs = TTL or CMOS
$I_{CC}2$	V_{CC} Standby Current	1,5		1	4	mA	$V_{CC} = V_{CC}$ Max RAS#, CAS#, RP# = V_{IH} WP# = V_{IL} or V_{IH}
$I_{CC}3$	V_{CC} RAS#-Only Refresh Current	1,5		50	70	mA	$V_{CC} = V_{CC}$ Max CAS# = V_{IH} RAS# = V_{IL} RAS#, Addr. Cycling @t_{RC} = min Inputs = TTL or CMOS
$I_{CC}4$	V_{CC} Fast Page Mode Word Read Current	1,4,5		40	60	mA	$V_{CC} = V_{CC}$ Max RAS#, CAS# = V_{IL} CAS#, Addr. Cycling @t_{PC} = min I_{OUT} = 0 mA Inputs = V_{IL} or V_{IH}
$I_{CC}5$	V_{CC} Standby Current	1,5		70	130	µA	$V_{CC} = V_{CC}$ Max RAS# CAS# RP# = V_{CC} ± 0.2V WP# = V_{CC} ± 0.2V or GND ± 0.2V
$I_{CC}6$	V_{CC} CAS#-before-RAS# Refresh Current	1,5		40	55	mA	$V_{CC} = V_{CC}$ Max CAS#, RAS# = V_{IL} CAS#, RAS#, Addr. Cycling @t_{RC} = min Inputs = TTL or CMOS
$I_{CC}7$	V_{CC} Standby Current (Self Refresh Mode)	1,5		40	55	mA	$V_{CC} = V_{CC}$ Max RAS#, CAS# = V_{IL} I_{OUT} = 0 mA Inputs = V_{IL} or V_{IH}
I_{LI}	Input Load Current	1			± 1	µA	$V_{CC} = V_{CC}$ Max $V_{IN} = V_{CC}$ or GND
I_{LO}	Output Leakage Current	1			± 10	µA	$V_{CC} = V_{CC}$ Max $V_{OUT} = V_{CC}$ or GND
I_{CCD}	V_{CC} Deep Power-Down Current	1		2	5	µA	RP# = GND ± 0.2V

5.4 DC Characteristics (Continued)

V_{CC} = 3.3V ± 0.3V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ	Max	Unit	Test Condition
I_{CCW}	V_{CC} Word Write Current	1,6		8	12	mA	Word Write in Progress V_{PP} = 12.0V ± 5%
				8	17	mA	Word Write in Progress V_{PP} = 5.0V ± 10%
I_{CCE}	V_{CC} Block Erase Current	1,6		6	12	mA	Block Erase in Progress V_{PP} = 12.0V ± 5%
				9	17	mA	Block Erase in Progress V_{PP} = 5.0V ± 10%
I_{CCES}	V_{CC} Erase Suspend Current	1,2		1	4	mA	RAS#, CAS# = V_{IH} Block Erase Suspended
I_{PPS}	V_{PP} Standby/Read Current	1		± 1	± 10	µA	$V_{PP} \leq V_{CC}$
				30	200	µA	$V_{PP} > V_{CC}$
I_{PPD}	V_{PP} Deep Power-Down Current	1		0.2	5	µA	RP# = GND ± 0.2V
I_{PPW}	V_{PP} Word Write Current	1,6		10	15	mA	V_{PP} = 12.0V ± 5% Word Write in Progress
				15	25	mA	V_{PP} = 5.0V ± 10% Word Write in Progress
I_{PPE}	V_{PP} Block Erase Current	1,6		4	10	mA	V_{PP} = 12.0V ± 5% Block Erase in Progress
				14	20	mA	V_{PP} = 5.0V ± 10% Block Erase in Progress
I_{PPES}	V_{PP} Erase Suspend Current	1		30	50	µA	Block Erase Suspended
V_{IL}	Input Low Voltage	6	-0.3		0.8	V	
V_{IH}	Input High Voltage	6	2.0		V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage	6			0.4	V	V_{CC} = V_{CC} Min I_{OL} = 4.0 mA
V_{OH1}	Output High Voltage	6	2.4			V	I_{OH} = –2.0 mA V_{CC} = V_{CC} Min
V_{OH2}		6	V_{CC}- 0.2			V	I_{OH} = –100 µA V_{CC} = V_{CC} Min
V_{PPLK}	V_{PP} Erase/Write Lock Voltage	3,6	0.0		1.5	V	
$V_{PPH}1$	V_{PP} during Write/Erase Operations	3	4.5	5.0	5.5	V	
$V_{PPH}2$	V_{PP} during Write/Erase Operations	3	11.4	12.0	12.6	V	
V_{LKO}	V_{CC} Erase/Write Lock Voltage		2.0			V	

NOTES:

1. All currents are in RMS unless otherwise noted. Typical values at V_{CC} = 3.3V, V_{PP} = 12.0V or 5.0V, T = 25°C.
2. I_{CCES} is specified with the device de-selected. If the device is read while in Erase Suspend mode, current draw is the sum of I_{CCES} and I_{CC1}/I_{CC4}.
3. Block erases, word writes and lock block operations are inhibited when $V_{PP} = V_{PPLK}$ and not guaranteed in the ranges between V_{PPLK}(max) and V_{PPH1}(min), between $V_{PPH1(max)}$ and $V_{PPH2(min)}$, and above $V_{PPH2(max)}$.
4. Automatic Power Saving (APS) reduces I_{CC1} and I_{CC4} to 3.0 mA typical in static operation.
5. CMOS inputs are either $V_{CC} \pm 0.2V$ or GND ± 0.2V. TTL inputs are either V_{IL} or V_{IH}.
6. Sampled, but not 100% tested. Guaranteed by design.

5.5 DC Characteristics

V_{CC} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ	Max	Unit	Test Condition
$I_{CC}1$	V_{CC} Word Read Current	1,4,5		90	120	mA	$V_{CC} = V_{CC}$ Max RAS#, CAS# = V_{IL} RAS#, CAS#, Addr. Cycling @ t_{RC} = min I_{OUT} = 0 mA Inputs = TTL or CMOS
$I_{CC}2$	V_{CC} Standby Current	1,5		2	4	mA	$V_{CC} = V_{CC}$ Max RAS#, CAS#, RP# = V_{IH} WP# = V_{IL} or V_{IH}
$I_{CC}3$	V_{CC} RAS#-Only Refresh Current	1,5		90	120	mA	$V_{CC} = V_{CC}$ Max CAS# = V_{IH} RAS# = V_{IL} RAS#, Addr. Cycling @t_{RC} = min Inputs = TTL or CMOS
$I_{CC}4$	V_{CC} Fast Page Mode Word Read Current	1,4,5		80	110	mA	$V_{CC} = V_{CC}$ Max RAS#, CAS# = V_{IL} CAS#, Addr. Cycling @t_{PC} = min I_{OUT} = 0 mA Inputs = V_{IL} or V_{IH}
$I_{CC}5$	V_{CC} Standby Current	1,5		70	130	µA	$V_{CC} = V_{CC}$ Max RAS#,CAS#,RP# = V_{CC} ± 0.2V WP# = V_{CC} ± 0.2V or GND ± 0.2V
$I_{CC}6$	V_{CC} CAS#-before-RAS# Refresh Current	1,5		50	65	mA	$V_{CC} = V_{CC}$ Max CAS#, RAS# = V_{IL} CAS#, RAS#, Addr. Cycling @t_{RC} = min Inputs = TTL or CMOS
$I_{CC}7$	V_{CC} Standby Current (Self Refresh Mode)	1,5		50	65	mA	$V_{CC} = V_{CC}$ Max RAS#, CAS# = V_{IL} I_{OUT} = 0 mA Inputs = V_{IL} or V_{IH}
I_{LI}	Input Load Current	1			± 1	µA	$V_{CC} = V_{CC}$ Max $V_{IN} = V_{CC}$ or GND
I_{LO}	Output Leakage Current	1			± 10	µA	$V_{CC} = V_{CC}$ Max $V_{OUT} = V_{CC}$ or GND
I_{CCD}	V_{CC} Deep Power-Down Current	1		2	5	µA	RP# = GND ± 0.2V
I_{CCW}	V_{CC} Word Write Current	1,6		25	35	mA	Word Write in Progress V_{PP} = 12.0V ± 5%
				25	40	mA	Word Write in Progress V_{PP} = 5.0V ± 10%

5.5 DC Characteristics (Continued)

V_{CC} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ	Max	Unit	Test Condition
I_{CCE}	V_{CC} Block Erase Current	1,6		18	25	mA	Block Erase in Progress V_{PP} = 12.0V ± 5%
				20	30	mA	Block Erase in Progress V_{PP} = 5.0V ± 10%
I_{CCES}	V_{CC} Erase Suspend Current	1,2		2	4	mA	RAS#, CAS# = V_{IH} Block Erase Suspended
I_{PPS}	V_{PP} Standby/Read Current	1		± 1	± 10	µA	$V_{PP} \leq V_{CC}$
				30	200	µA	$V_{PP} > V_{CC}$
I_{PPD}	V_{PP} Deep Power-Down Current	1		0.2	5	µA	RP# = GND ± 0.2V
I_{PPW}	V_{PP} Word Write Current	1,6		7	12	mA	V_{PP} = 12.0V ± 5% Word Write in Progress
				17	22	mA	V_{PP} = 5.0V ± 10% Word Write in Progress
I_{PPE}	V_{PP} Block Erase Current	1,6		5	10	mA	V_{PP} = 12.0V ± 5% Block Erase in Progress
				16	20	mA	V_{PP} = 5.0V ± 10% Block Erase in Progress
I_{PPES}	V_{PP} Erase Suspend Current	1		30	50	µA	Block Erase Suspended
V_{IL}	Input Low Voltage	6	–0.5		0.8	V	
V_{IH}	Input High Voltage	6	2.0		V_{CC} + 0.5	V	
V_{OL}	Output Low Voltage	6			0.45	V	V_{CC} = V_{CC} Min I_{OL} = 5.8 mA
V_{OH1}	Output High Voltage	6	0.85 V_{CC}			V	I_{OH} = –2.5 mA V_{CC} = V_{CC} Min
V_{OH2}		6	V_{CC}– 0.4			V	I_{OH} = –100 µA V_{CC} = V_{CC} Min
V_{PPLK}	V_{PP} Erase/Write Lock Voltage	3,6	0.0		1.5	V	
$V_{PPH}1$	V_{PP} during Write/Erase Operations	3	4.5	5.0	5.5	V	
$V_{PPH}2$	V_{PP} during Write/Erase Operations	3	11.4	12.0	12.6	V	
V_{LKO}	V_{CC} Erase/Write Lock Voltage		2.0			V	

NOTES:

1. All currents are in RMS unless otherwise noted. Typical values at V_{CC} = 5.0V, V_{PP} = 12.0V or 5.0V, T = 25°C. These currents are specified for a CMOS rise/fall time (10% to 90%) of <5 ns and a TTL rise/fall time of <10 ns.
2. I_{CCES} is specified with the device de-selected. If the device is read while in Erase Suspend mode, current draw is the sum of I_{CCES} and I_{CC1}/I_{CC4}.
3. Block erases, word writes and lock block operations are inhibited when V_{PP} = V_{PPLK} and not guaranteed in the ranges between V_{PPLK}(max) and V_{PPH1}(min), between V_{PPH1}(max) and V_{PPH2}(min), and above V_{PPH2}(max).
4. Automatic Power Saving (APS) reduces I_{CC1} and I_{CC4} to 1 mA typical in static operation.
5. CMOS inputs are either V_{CC} ± 0.2V or GND ± 0.2V. TTL inputs are either V_{IL} or V_{IH}.
6. Sampled, not 100% tested. Guaranteed by design.

5.6 AC Characteristics[11]

V_{CC} = 3.3V ± 0.3V, T_A = 0°C to +70°C

Read, Write, Read-Modify-Write and Refresh Cycles (Common Parameters)

Versions			28F016XD-95		Units
Sym	Parameter	Notes	Min	Max	
t_{RP}	RAS# precharge time		10		ns
t_{CP}	CAS# precharge time		15		ns
t_{ASR}	Row address set-up time	9	0		ns
t_{RAH}	Row address hold time	9	15		ns
t_{ASC}	Column address set-up time	9	0		ns
t_{CAH}	Column address hold time	9	20		ns
t_{AR}	Column address hold time referenced to RAS#	3,9	35		ns
t_{RAD}	RAS# to column address delay time	8,9	15	15	ns
t_{CRP}	CAS# to RAS# precharge time		10		ns
t_{OED}	OE# to data delay	10	30		ns
t_{DZO}	OE# delay time from data-in	10	0		ns
t_{DZC}	CAS# delay time from data-in	10	0		ns
t_T	Transition time (rise and fall)	10	2	4	ns

Read Cycle

Versions			28F016XD-95		Units
Sym	Parameter	Notes	Min	Max	
$t_{RC(R)}$	Random read cycle time		105		ns
$t_{RAS(R)}$	RAS# pulse width (reads)		95	∞	ns
$t_{CAS(R)}$	CAS# pulse width (reads)		40	∞	ns
$t_{RCD(R)}$	RAS# to CAS# delay time (reads)	1	15	55	ns
$t_{RSH(R)}$	RAS# hold time (reads)		30		ns
$t_{CSH(R)}$	CAS# hold time (reads)		95		ns
t_{RAC}	Access time from RAS#	1,8		95	ns
t_{CAC}	Access time from CAS#	1,2		40	ns
t_{AA}	Access time from column address	8		75	ns
t_{OEA}	OE# access time			40	ns

Read Cycle (Continued)

Versions			28F016XD-95		Units
Sym	Parameter	Notes	Min	Max	
t_{ROH}	RAS# hold time referenced to OE#		40		ns
t_{RCS}	Read command setup time		5		ns
t_{RCH}	Read command hold time referenced to CAS#	6,10	0		ns
t_{RRH}	Read command hold time referenced to RAS#	6,10	0		ns
t_{RAL}	Column address to RAS# lead time	9	15		ns
t_{CAL}	Column address to CAS# lead time	9	75		ns
t_{CLZ}	CAS# to output in Low-Z		0		ns
t_{OH}	Output data hold time		0		ns
t_{OHO}	Output data hold time from OE#		0		ns
t_{OFF}	Output buffer turn-off delay	4		30	ns
t_{OEZ}	Output buffer turn off delay time from OE#			30	ns
t_{CDD}	CAS# to data-in delay time		30		ns

Write Cycle

Versions			28F016XD-95		Units
Sym	Parameter	Notes	Min	Max	
$t_{RC(W)}$	Random write cycle time		90		ns
$t_{RAS(W)}$	RAS# pulse width (writes)		80	∞	ns
$t_{CAS(W)}$	CAS# pulse width (writes)		65	∞	ns
$t_{RCD(W)}$	RAS# to CAS# delay time (writes)	1	15	15	ns
$t_{RSH(W)}$	RAS# hold time (writes)		65		ns
$t_{CSH(W)}$	CAS# hold time (writes)		80		ns
t_{WCS}	Write command set-up time	5	0		ns
t_{WCH}	Write command hold time		15		ns
t_{WCR}	Write command hold time referenced to RAS#	3	30		ns
t_{WP}	Write command pulse width		15		ns
t_{RWL}	Write command to RAS# lead time		65		ns
t_{CWL}	Write command to CAS# lead time		65		ns
t_{DS}	Data-in set-up time	7,9	0		ns
t_{DH}	Data-in hold time	7,9	15		ns
t_{DHR}	Data-in hold time referenced to RAS#	3,9	30		ns

Read-Modify-Write Cycle

Versions			28F016XD-95		Units
Sym	Parameter	Notes	Min	Max	
t_{RWC}	Read-modify-write cycle time	10	200		ns
t_{RWD}	RAS# to WE# delay time	5,10	125		ns
t_{CWD}	CAS# to WE# delay time	5,10	70		ns
t_{AWD}	Column address to WE# delay time	5,9,10	105		ns
t_{OEH}	OE# command hold time	10	15		ns

Fast Page Mode Cycle

Versions			28F016XD-95		Units
Sym	Parameter	Notes	Min	Max	
$t_{PC(R)}$	Fast page mode cycle time (reads)		75		ns
$t_{PC(W)}$	Fast page mode cycle time (writes)		80		ns
$t_{RASP(R)}$	RAS# pulse width (reads)		95	∞	ns
$t_{RASP(W)}$	RAS# pulse width (writes)		80	∞	ns
t_{CPA}	Access time from CAS# precharge			85	ns
t_{CPW}	WE# delay time from CAS# precharge	10	0		ns
$t_{CPRH(R)}$	RAS# hold time from CAS# precharge (reads)		75		ns
$t_{CPRH(W)}$	RAS# hold time from CAS# precharge (writes)		80		ns

Fast Page Mode Read-Modify-Write Cycle

Versions			28F016XD-95		Units
Sym	Parameter	Notes	Min	Max	
t_{PRWC}	Fast page mode read-modify-write cycle time	10	170		ns

Refresh Cycle

Versions			28F016XD-95		Units
Sym	Parameter	Notes	Min	Max	
t_{CSR}	CAS# set-up time (CAS#-before-RAS# refresh)	10	10		ns
t_{CHR}	CAS# hold time (CAS#-before-RAS# refresh)	10	10		ns
t_{WRP}	WE# setup time (CAS#-before-RAS# refresh)	10	10		ns
t_{WRH}	WE# hold time (CAS#-before-RAS# refresh)	10	10		ns
t_{RPC}	RAS# precharge to CAS# hold time	10	10		ns
t_{RASS}	RAS# pulse width (self-refresh mode)	10	0		ns
t_{RPS}	RAS# precharge time (self-refresh mode)	10	10		ns
t_{CPN}	CAS# precharge time (self-refresh mode)	10	10		ns
t_{CHS}	CAS# hold time (self-refresh mode)	10	0		ns

Refresh

Versions			28F016XD-95		Units
Sym	Parameter	Notes	Min	Max	
t_{REF}	Refresh period	10		∞	ms

Misc. Specifications

Versions		28F016XD-95		Units
Parameter	Notes	Min	Max	
RP# high to RAS# going low	10	480		ns
RP# set-up to WE# going low	10	480		ns
V_{PP} set-up to CAS# high at end of write cycle	10	100		ns
WE# high to RY/BY# going low	10		100	ns
RP# hold from valid status register data and RY/BY# high	10	0		ns
V_{PP} hold from valid status register data and RY/BY# high	10	0		ns

NOTES:

1. Operation within the $t_{RCD(max)}$ limit insures that $t_{RAC(max)}$ can be met. $t_{RCD(max)}$ is specified as a reference point.
2. Assumes that $t_{RCD} \geq t_{RCD(max)}$.
3. t_{AR}, t_{WCR}, t_{DHR} are referenced to $t_{RAD(max)}$.
4. $t_{OFF(max)}$ defines the time at which the output achieves the open circuit condition and is not referenced to V_{OH} or V_{OL}.
5. t_{WCS}, t_{RWD}, t_{CWD} and t_{AWD} are non restrictive operating parameters. They are included in the datasheet as electrical characteristics only. If $t_{WCS} \geq t_{WCS(min)}$ the cycle is an early write cycle and the data output will remain high impedance for the duration of the cycle. If $t_{CWD} \geq t_{CWD(min)}$, $t_{RWD} \geq t_{RWD(min)}$, $t_{AWD} \geq t_{AWD(min)}$, then the cycle is a read-write cycle and the data output will contain the data read from the selected address. If neither of the above conditions are satisfied, the condition of the data out is indeterminate.
6. Either t_{RCH} or t_{RRH} must be satisfied for a read cycle.
7. These parameters are referenced to the CAS# leading edge in early write cycles and to the WE# leading edge in read-write cycles.
8. Operation within the $t_{RAD(max)}$ limit ensures that $t_{RAC(max)}$ can be met, $t_{RAD(max)}$ is specified as a reference point only. If t_{RAD} is greater than the specified $t_{RAD(max)}$ limit, then the access time is controlled by t_{AA}.
9. Refer to command definition tables for valid address and data values.
10. Sampled, but not 100% tested. Guaranteed by design.
11. See AC Input/Output Reference Waveforms for timing measurements.

5.7 AC Characteristics(11)

V_{CC} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Read, Write, Read-Modify-Write and Refresh Cycles (Common Parameters)

Versions			28F016XD-85		Units
Sym	Parameter	Notes	Min	Max	
t_{RP}	RAS# precharge time		10		ns
t_{CP}	CAS# precharge time		15		ns
t_{ASR}	Row address set-up time	9	0		ns
t_{RAH}	Row address hold time	9	15		ns
t_{ASC}	Column address set-up time	9	0		ns
t_{CAH}	Column address hold time	9	20		ns
t_{AR}	Column address hold time referenced to RAS#	3,9	35		ns
t_{RAD}	RAS# to column address delay time	8,9	15	15	ns
t_{CRP}	CAS# to RAS# precharge time		10		ns
t_{OED}	OE# to data delay	10	30		ns
t_{DZO}	OE# delay time from data-in	10	0		ns
t_{DZC}	CAS# delay time from data-in	10	0		ns
t_T	Transition time (rise and fall)	10	2	4	ns

Read Cycle

Versions			28F016XD-85		Units
Sym	Parameter	Notes	Min	Max	
$t_{RC(R)}$	Random read cycle time		95		ns
$t_{RAS(R)}$	RAS# pulse width (reads)		85	∞	ns
$t_{CAS(R)}$	CAS# pulse width (reads)		35	∞	ns
$t_{RCD(R)}$	RAS# to CAS# delay time (reads)	1	15	50	ns
$t_{RSH(R)}$	RAS# hold time (reads)		30		ns
$t_{CSH(R)}$	CAS# hold time (reads)		85		ns
t_{RAC}	Access time from RAS#	1,8		85	ns
t_{CAC}	Access time from CAS#	1,2		35	ns
t_{AA}	Access time from column address	8		65	ns
t_{OEA}	OE# access time			35	ns
t_{ROH}	RAS# hold time referenced to OE#		35		ns
t_{RCS}	Read command setup time		5		ns
t_{RCH}	Read command hold time referenced to CAS#	6,10	0		ns
t_{RRH}	Read command hold time referenced to RAS#	6,10	0		ns
t_{RAL}	Column address to RAS# lead time	9	15		ns
t_{CAL}	Column address to CAS# lead time	9	65		ns
t_{CLZ}	CAS# to output in Low-Z	10	0		ns
t_{OH}	Output data hold time	10	0		ns
t_{OHO}	Output data hold time from OE#	10	0		ns
t_{OFF}	Output buffer turn-off delay	4,10		30	ns
t_{OEZ}	Output buffer turn off delay time from OE#	10		30	ns
t_{CDD}	CAS# to data-in delay time	10	30		ns

Write Cycle

Versions			28F016XD-85		Units
Sym	Parameter	Notes	Min	Max	
$t_{RC(W)}$	Random write cycle time		75		ns
$t_{RAS(W)}$	RAS# pulse width (writes)		65	∞	ns
$t_{CAS(W)}$	CAS# pulse width (writes)		50	∞	ns
$t_{RCD(W)}$	RAS# to CAS# delay time (writes)	1	15	15	ns
$t_{RSH(W)}$	RAS# hold time (writes)		50		ns
$t_{CSH(W)}$	CAS# hold time (writes)		65		ns
t_{WCS}	Write command set-up time	5	0		ns
t_{WCH}	Write command hold time		15		ns
t_{WCR}	Write command hold time referenced to RAS#	3	30		ns
t_{WP}	Write command pulse width		15		ns
t_{RWL}	Write command to RAS# lead time		50		ns
t_{CWL}	Write command to CAS# lead time		50		ns
t_{DS}	Data-in set-up time	7,9	0		ns
t_{DH}	Data-in hold time	7,9	15		ns
t_{DHR}	Data-in hold time referenced to RAS#	3,9	30		ns

Read-Modify-Write Cycle

Versions			28F016XD-85		Units
Sym	Parameter	Notes	Min	Max	
t_{RWC}	Read-modify-write cycle time	10	175		ns
t_{RWD}	RAS# to WE# delay time	5,10	115		ns
t_{CWD}	CAS# to WE# delay time	5,10	65		ns
t_{AWD}	Column address to WE# delay time	5,9,10	100		ns
t_{OEH}	OE# command hold time	10	15		ns

Fast Page Mode Cycle

Versions			28F016XD-85		Units
Sym	Parameter	Notes	Min	Max	
$t_{PC(R)}$	Fast page mode cycle time (reads)		65		ns
$t_{PC(W)}$	Fast page mode cycle time (writes)		65		ns

Fast Page Mode Cycle Continued

Versions			28F016XD-85		Units
Sym	**Parameter**	**Notes**	**Min**	**Max**	
$t_{RASP(R)}$	RAS# pulse width (reads)		85	∞	ns
$t_{RASP(W)}$	RAS# pulse width (writes)		65	∞	ns
t_{CPA}	Access time from CAS# precharge			70	ns
t_{CPW}	WE# delay time from CAS# precharge	10	0		ns
$t_{CPRH(R)}$	RAS# hold time from CAS# precharge (reads)		65		ns
$t_{CPRH(W)}$	RAS# hold time from CAS# precharge (writes)		65		ns

Fast Page Mode Read-Modify-Write Cycle

Versions			28F016XD-85		Units
Sym	**Parameter**	**Notes**	**Min**	**Max**	
t_{PRWC}	Fast page mode read-modify-write cycle time	10	145		ns

Refresh Cycle

Versions			28F016XD-85		Units
Sym	**Parameter**	**Notes**	**Min**	**Max**	
t_{CSR}	CAS# set-up time (CAS#-before-RAS# refresh)	10	10		ns
t_{CHR}	CAS# hold time (CAS#-before-RAS# refresh)	10	10		ns
t_{WRP}	WE# setup time (CAS#-before-RAS# refresh)	10	10		ns
t_{WRH}	WE# hold time (CAS#-before-RAS# refresh)	10	10		ns
t_{RPC}	RAS# precharge to CAS# hold time	10	10		ns
t_{RASS}	RAS# pulse width (self-refresh mode)	10	0		ns
t_{RPS}	RAS# precharge time (self-refresh mode)	10	10		ns
t_{CPN}	CAS# precharge time (self-refresh mode)	10	10		ns
t_{CHS}	CAS# hold time (self-refresh mode)	10	0		ns

Refresh

Versions			28F016XD-85		Units
Sym	**Parameter**	**Notes**	**Min**	**Max**	
t_{REF}	Refresh period	10		∞	ms

Misc. Specifications

Versions		28F016XD-85		Units
Parameter	Notes	Min	Max	
RP# high to RAS# going low	10	300		ns
RP# set-up to WE# going low	10	300		ns
V_{PP} set-up to CAS# high at end of write cycle	10	100		ns
WE# high to RY/BY# going low	10		100	ns
RP# hold from valid status register data and RY/BY# high	10	0		ns
V_{PP} hold from valid status register data and RY/BY# high	10	0		ns

NOTES:

1. Operation within the $t_{RCD(max)}$ limit insures that $t_{RAC(max)}$ can be met. $t_{RCD(max)}$ is specified as a reference point.
2. Assumes that $t_{RCD} \geq t_{RCD(max)}$.
3. t_{AR}, t_{WCR}, t_{DHR} are referenced to $t_{RAD(max)}$.
4. $t_{OFF(max)}$ defines the time at which the output achieves the open circuit condition and is not referenced to V_{OH} or V_{OL}.
5. t_{WCS}, t_{RWD}, t_{CWD} and t_{AWD} are non restrictive operating parameters. They are included in the datasheet as electrical characteristics only. If $t_{WCS} \geq t_{WCS(min)}$ the cycle is an early write cycle and the data output will remain high impedance for the duration of the cycle. If $t_{CWD} \geq t_{CWD(min)}$, $t_{RWD} \geq t_{RWD(min)}$, $t_{AWD} \geq t_{AWD(min)}$, then the cycle is a read-write cycle and the data output will contain the data read from the selected address. If neither of the above conditions are satisfied, the condition of the data out is indeterminate.
6. Either t_{RCH} or t_{RRH} must be satisfied for a read cycle.
7. These parameters are referenced to the CAS# leading edge in early write cycles and to the WE# leading edge in read-write cycles.
8. Operation within the t_{RAD}(max) limit ensures that $t_{RAC(max)}$ can be met, $t_{RAD(max)}$ is specified as a reference point only. If t_{RAD} is greater than the specified $t_{RAD(max)}$ limit, then the access time is controlled by t_{AA}.
9. Refer to command definition tables for valid address and data values.
10. Sampled, but not 100% tested. Guaranteed by design.
11. See AC Input/Output Reference Waveforms for timing measurements.

5.8 AC Waveforms

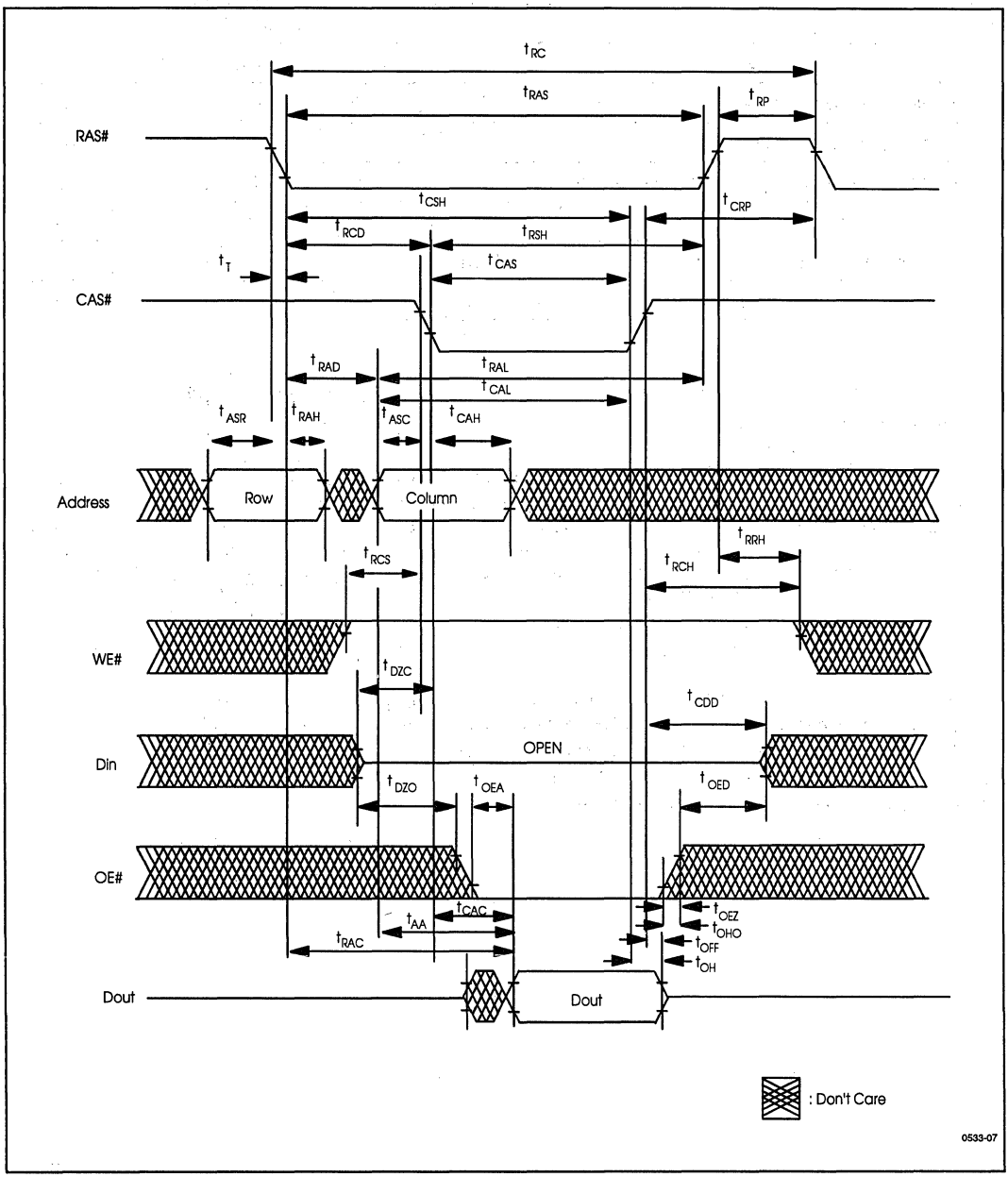


Figure 7. AC Waveforms for Read Operations

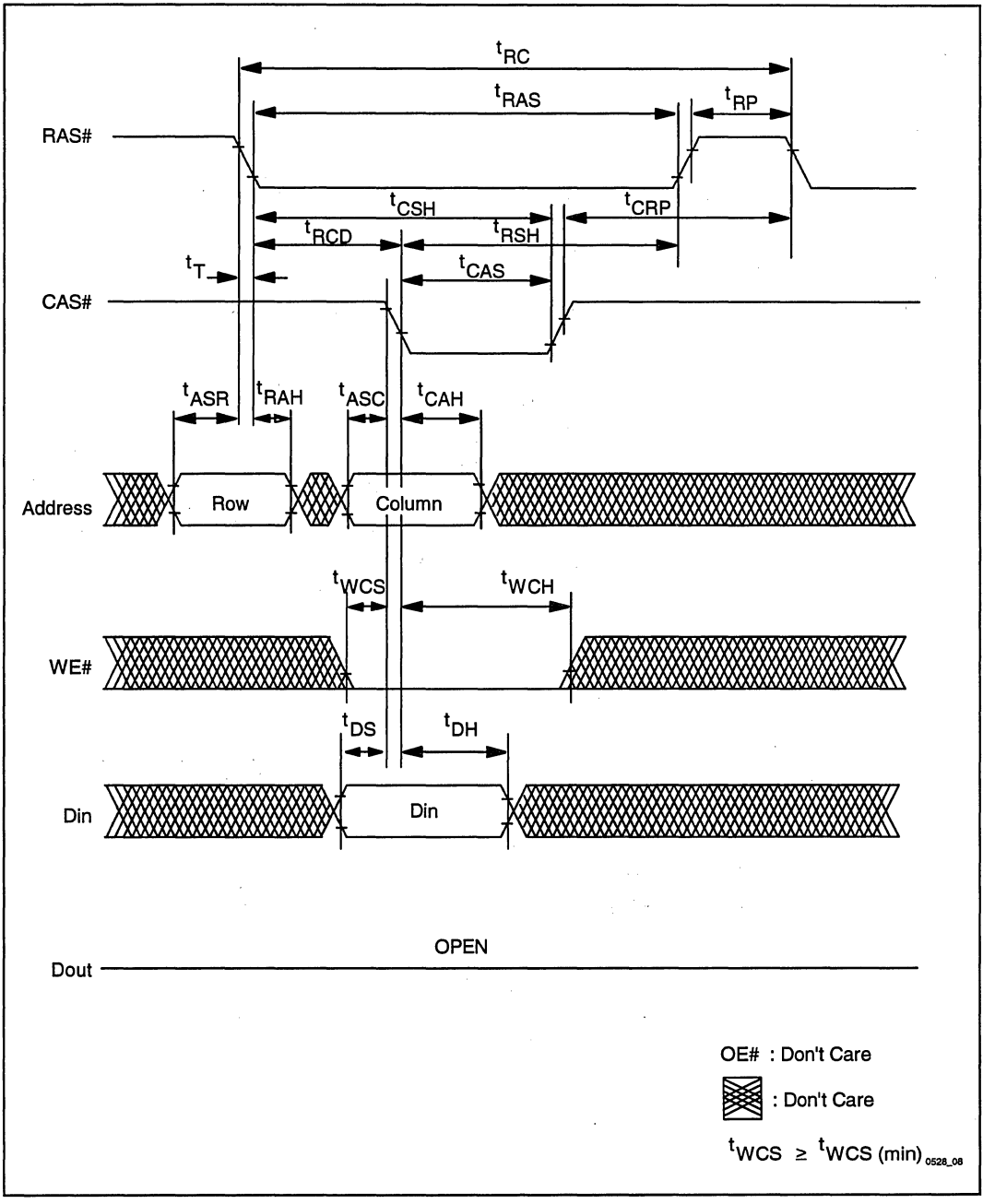


Figure 8. AC Waveforms for Early Write Operations

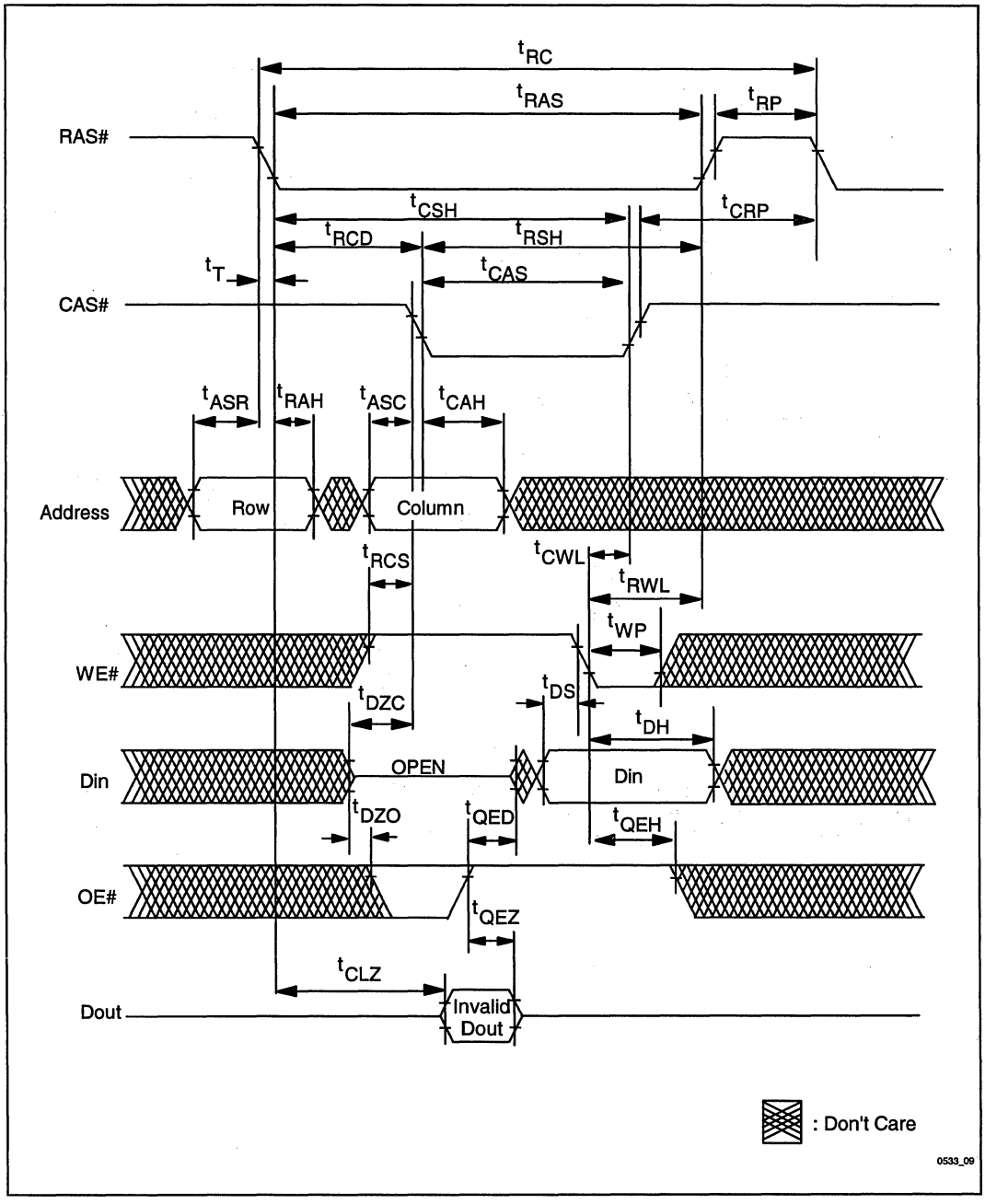


Figure 9. AC Waveforms for Delayed Write Operations

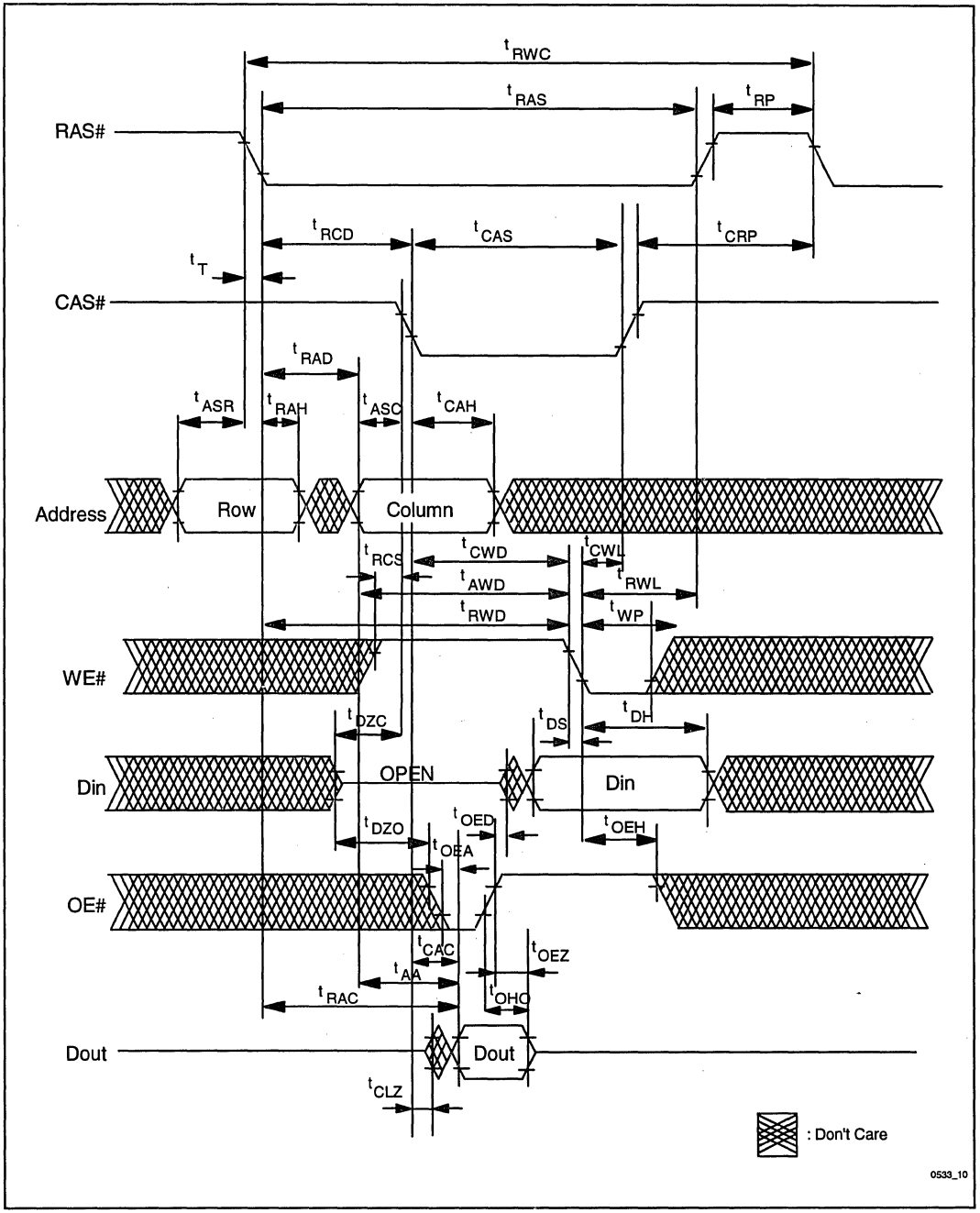


Figure 10. AC Waveforms for Read-Modify-Write Operations

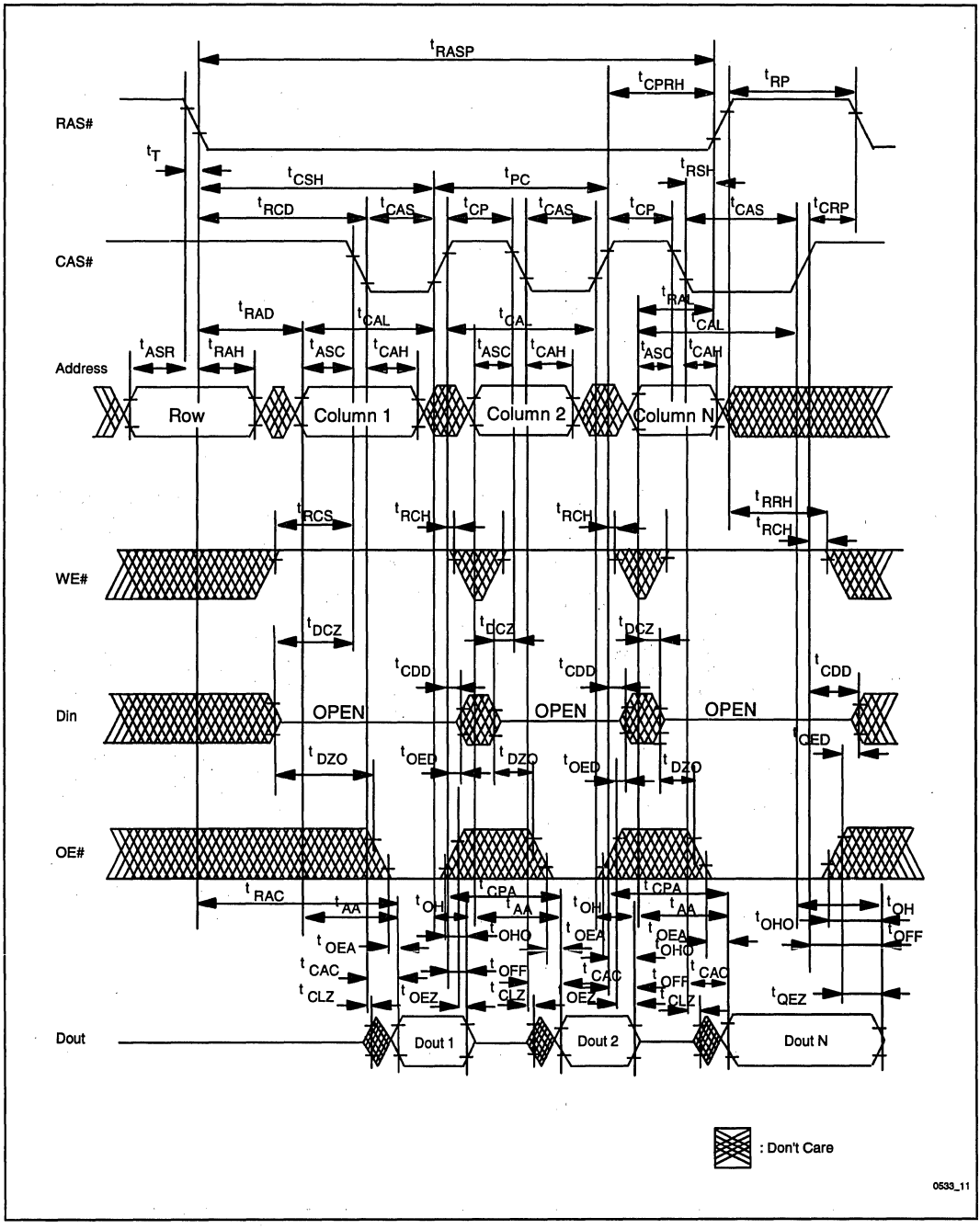


Figure 11. AC Waveforms for Fast Page Mode Read Operations

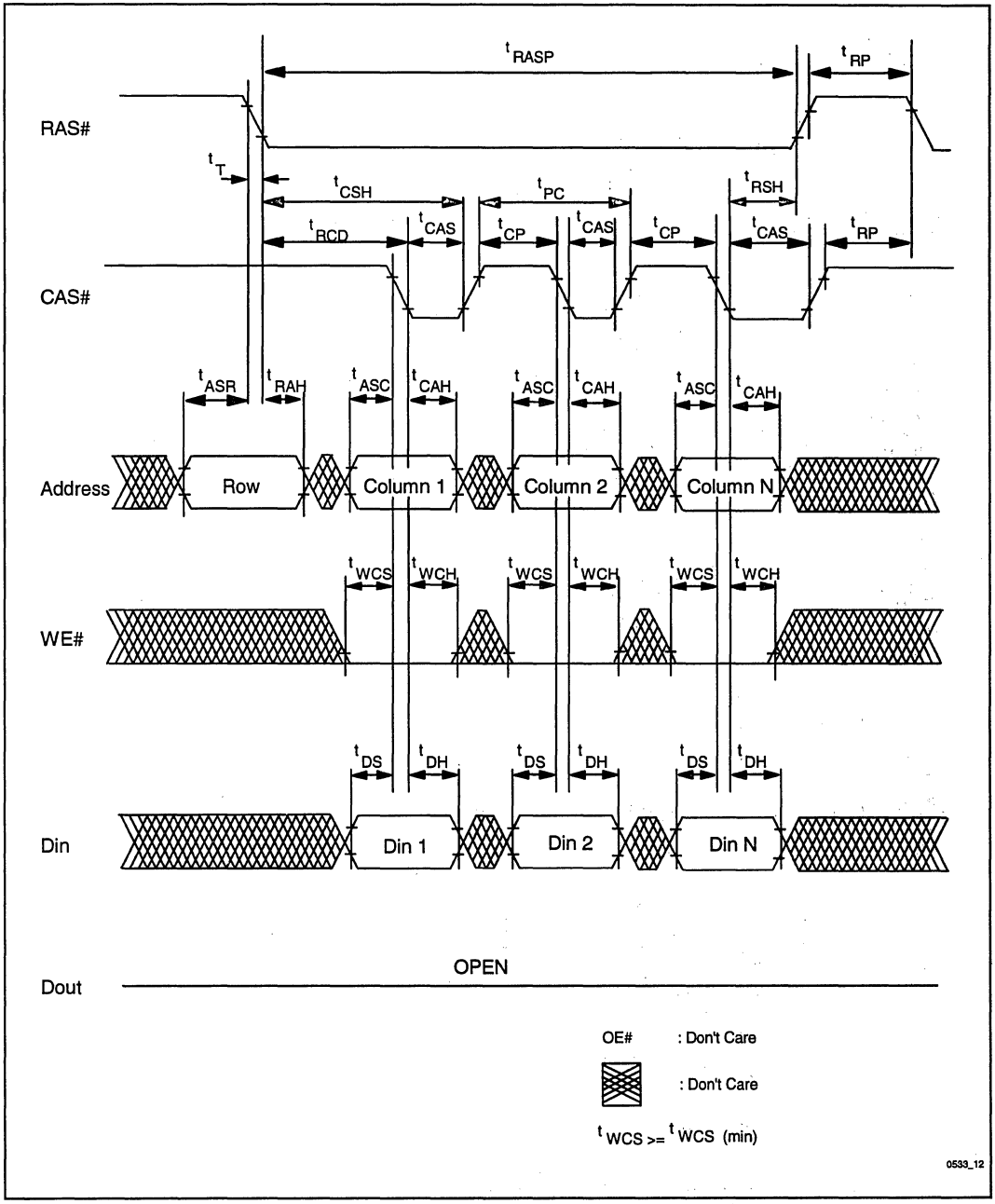


Figure 12. AC Waveforms for Fast Page Mode Early Write Operations

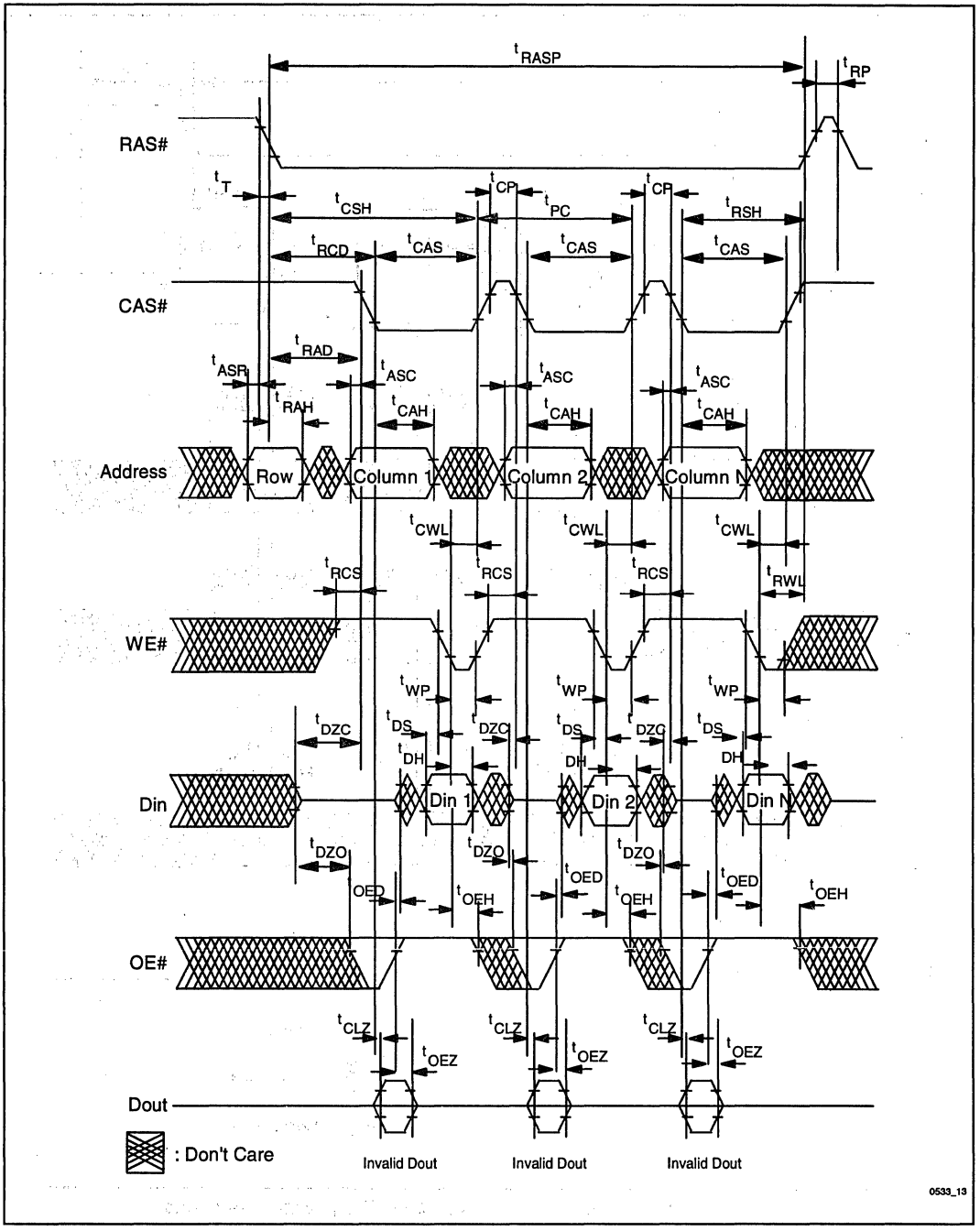


Figure 13. AC Waveforms for Fast Page Mode Delayed Write Operations

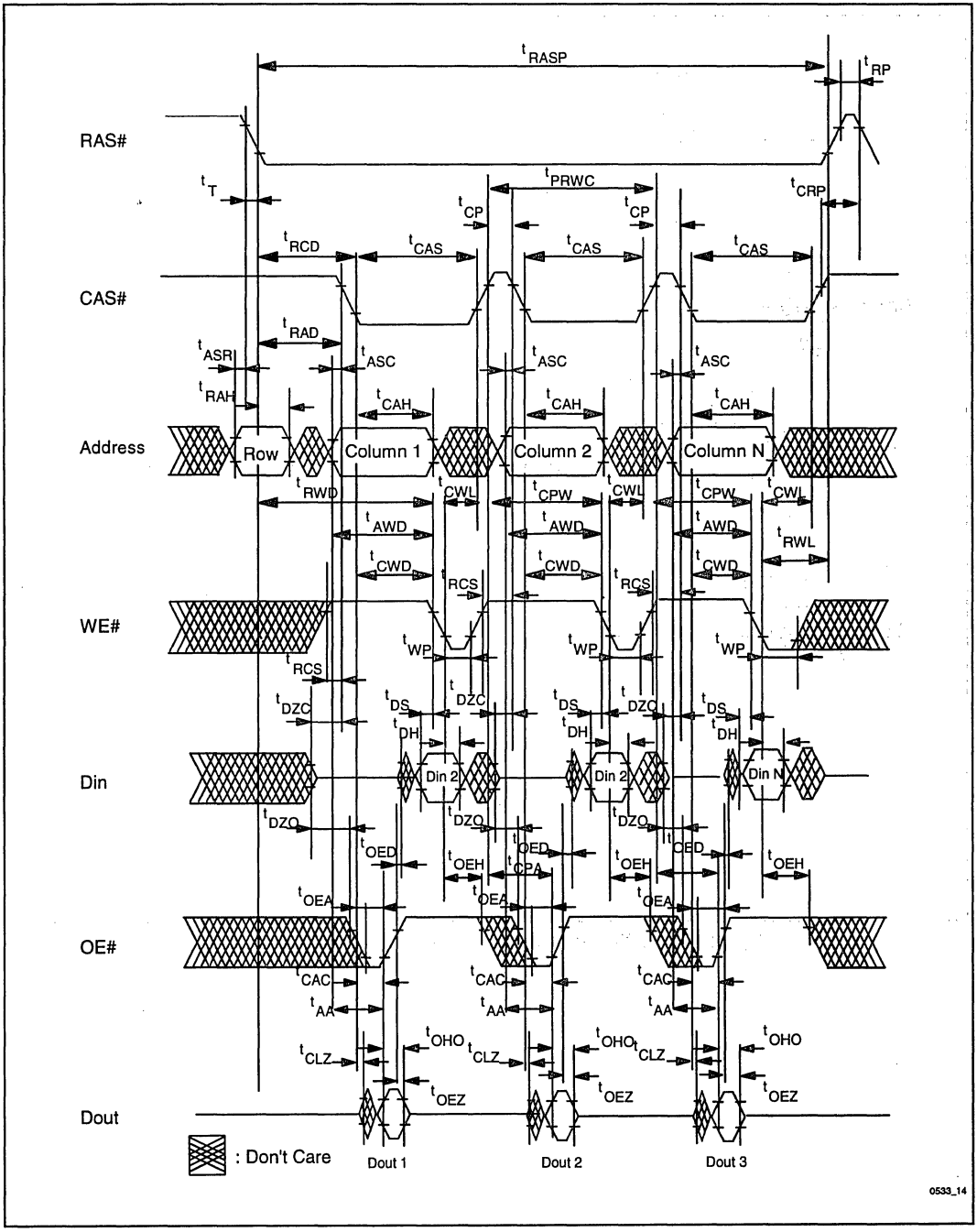


Figure 14. AC Waveforms for Fast Page Mode Read-Modify-Write Operations

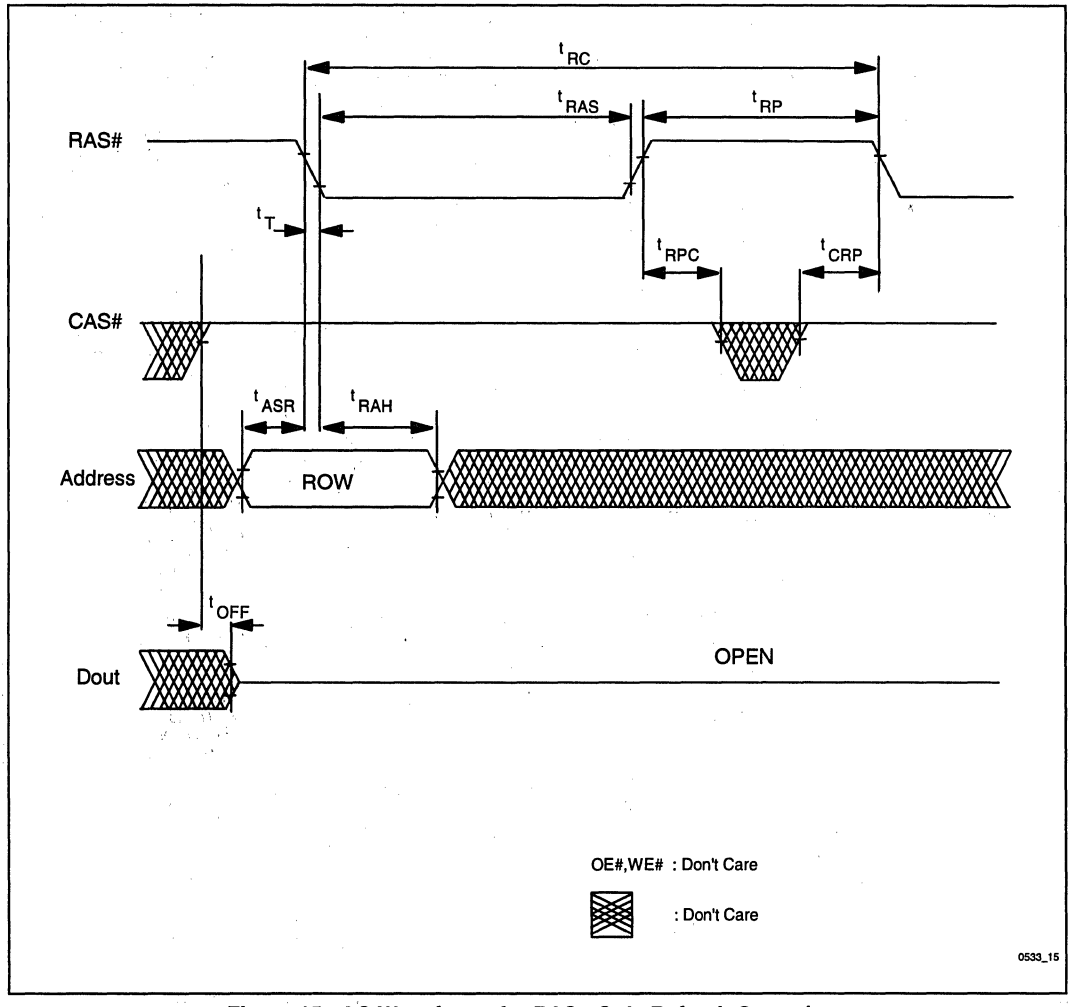


Figure 15. AC Waveforms for RAS#-Only Refresh Operations

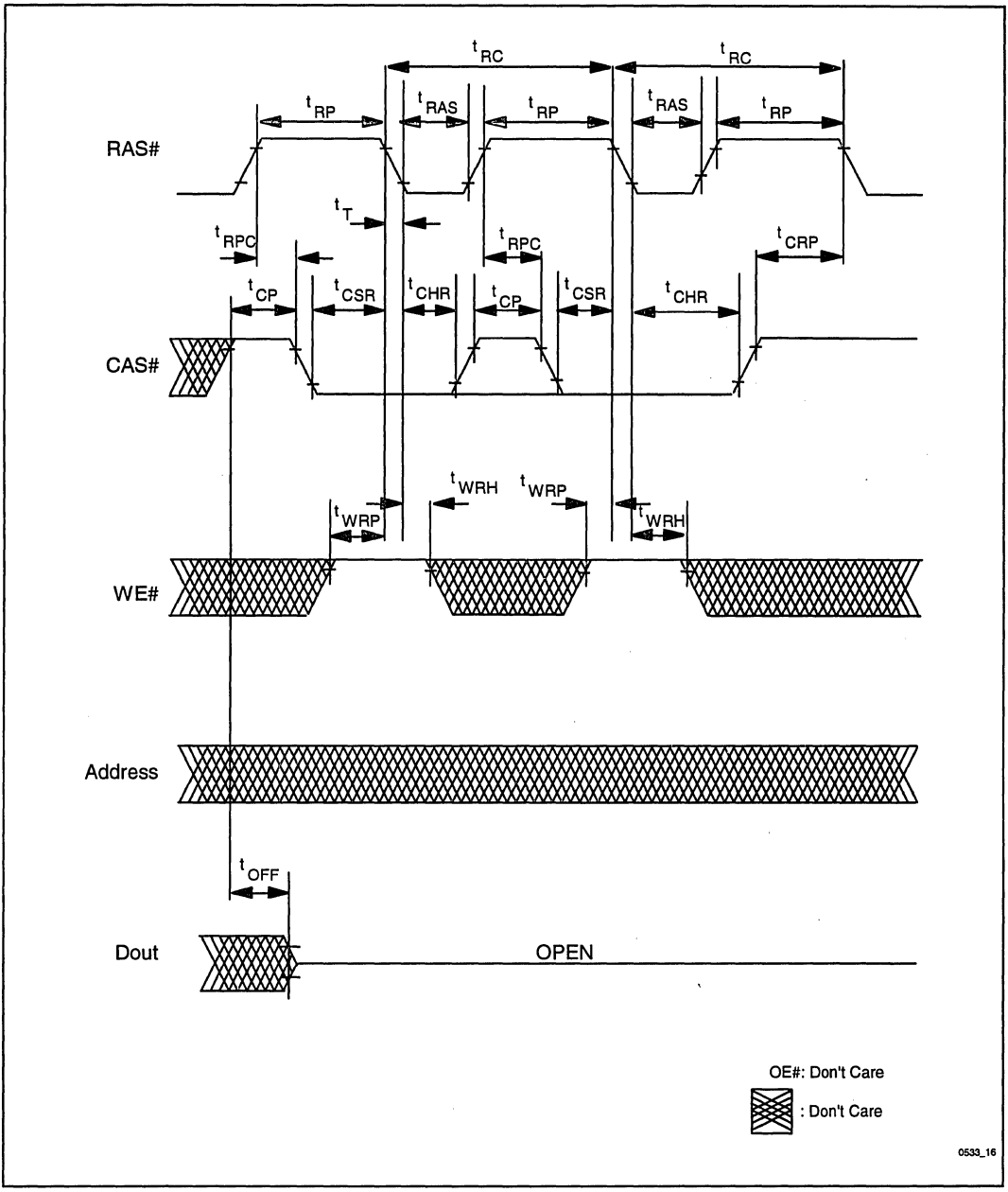


Figure 16. AC Waveforms for CAS#-before-RAS# Refresh Operations

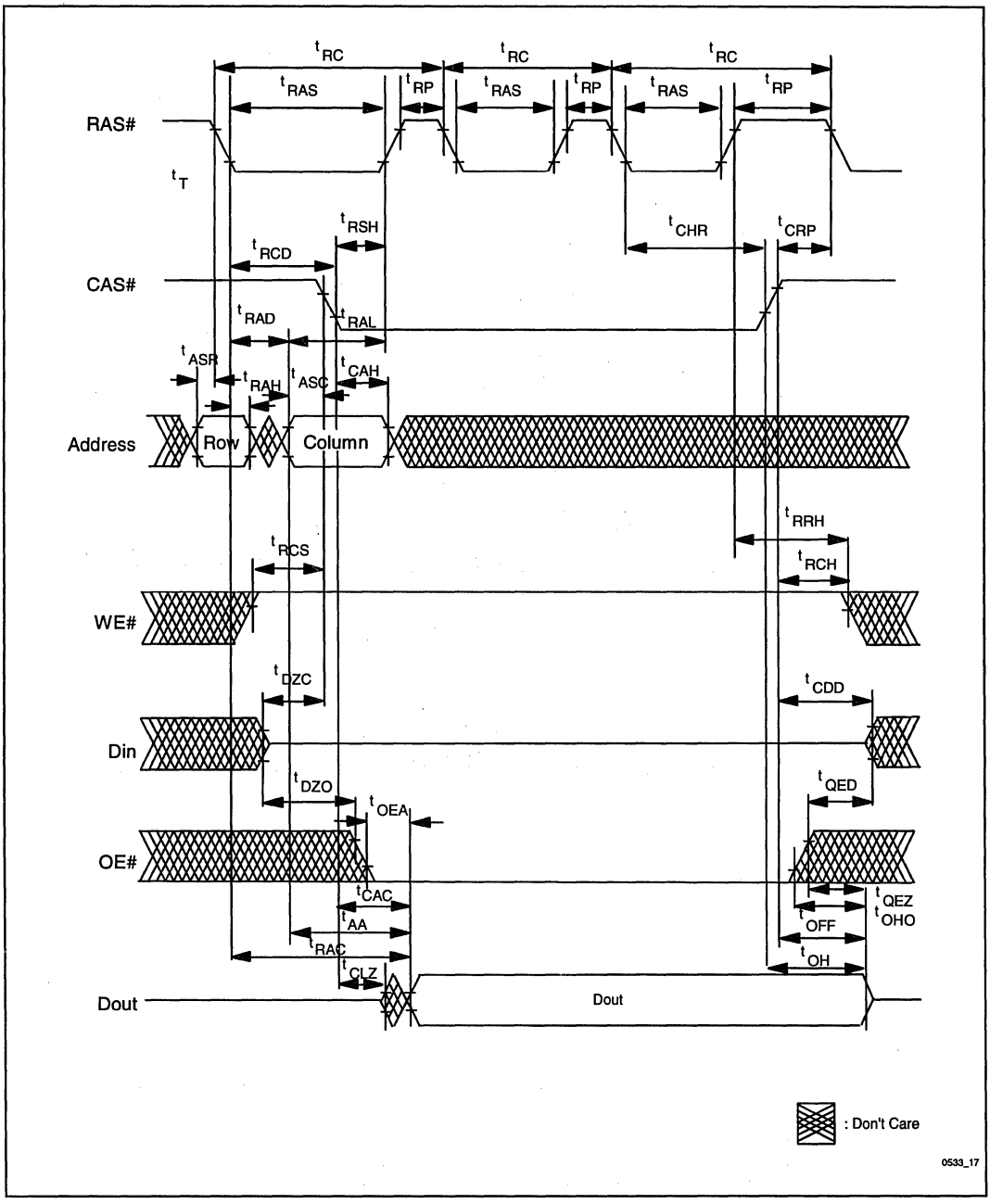


Figure 17. AC Waveforms for Hidden Refresh Operations

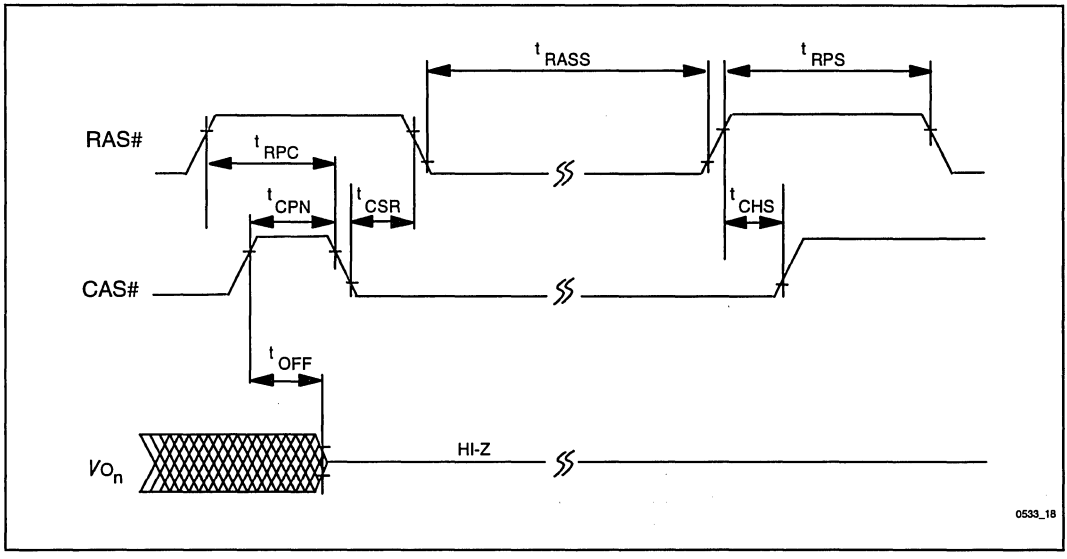


Figure 18. AC Waveforms for Self-Refresh Operations

5.9 Power-Up and Reset Timings

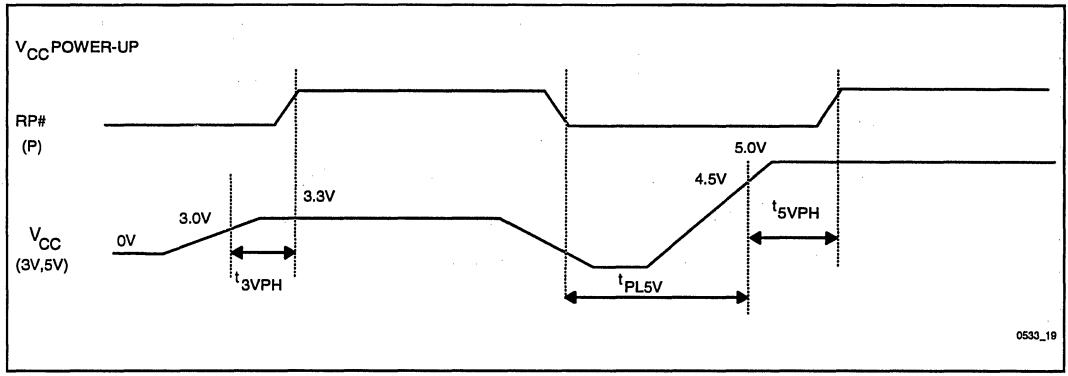


Figure 19. V_{CC} Power-Up and RP# Reset Waveforms

Sym	Parameter	Notes	Min	Max	Unit
t_{PL5V}	RP# Low to V_{CC} at 4.5V (Minimum)	2	0		µs
t_{PL3V}	RP# Low to V_{CC} at 3.0V (Minimum)	2	0		µs
t_{5VPH}	V_{CC} at 4.5V Minimum) to RP# High	1	2		µs
t_{3VPH}	V_{CC} at 3.0V (Minimum) to RP# High	1	2		µs

NOTES:
For Read Timings following Reset, see sections 5.6 and 5.7.

1. The t_{5VPH} and/or t_{3VPH} times must be strictly followed to guarantee all other read and write specifications for the 28F016XD
2. The power supply may start to switch concurrently with RP# going low.

5.10 Erase and Word Write Performance(3,4)

V_{CC} = 3.3V ± 0.3V, V_{PP} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Symbol	Parameter	Notes	Min	Typ(1)	Max	Units
$t_{WHRH}1$	Word Write Time	2,5	TBD	35.0	TBD	µs
$t_{WHRH}3$	Block Write Time	2,5	TBD	1.2	TBD	sec
	Block Erase Time	2,5	TBD	1.4	TBD	sec
	Erase Suspend Latency Time to Read		1.0	12.0	75.0	µs

V_{CC} = 3.3V ± 0.3V, V_{PP} = 12.0V ± 0.6V, T_A = 0°C to +70°C

Symbol	Parameter	Notes	Min	Typ(1)	Max	Units
$t_{WHRH}1$	Word Write Time	2,5	5	9	TBD	µs
$t_{WHRH}3$	Block Write Time	2,5	TBD	0.3	1.0	sec
	Block Erase Time	2	0.3	0.8	10	sec
	Erase Suspend Latency Time to Read		1.0	9.0	55.0	µs

V_{CC} = 5.0V ± 0.5V, V_{PP} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Symbol	Parameter	Notes	Min	Typ(1)	Max	Units
$t_{WHRH}1$	Word Write Time	2,5	TBD	25.0	TBD	µs
$t_{WHRH}3$	Block Write Time	2,5	TBD	0.85	TBD	sec
	Block Erase Time	2,5	TBD	1.0	TBD	sec
	Erase Suspend Latency Time to Read		1.0	9.0	55.0	µs

V_{CC} = 5.0V ± 0.5V, V_{PP} = 12.0V ± 0.6V, T_A = 0°C to +70°C

Symbol	Parameter	Notes	Min	Typ(1)	Max	Units
$t_{WHRH}1$	Word Write Time	2,5	4.5	6	TBD	µs
$t_{WHRH}3$	Block Write Time	2,5	TBD	0.2	1.0	sec
	Block Erase Time	2	0.3	0.6	10	sec
	Erase Suspend Latency Time to Read		1.0	7.0	40.0	µs

NOTES:

1. 25°C, and nominal voltages.
2. Excludes system-level overhead.
3. These performance numbers are valid for all speed versions.
4. Sampled, but not 100% tested. Guaranteed by design.
5. The TBD information will be available in a technical paper. Please contact Intel's Application Hotline or your local sales office for more information.

6.0 MECHANICAL SPECIFICATIONS

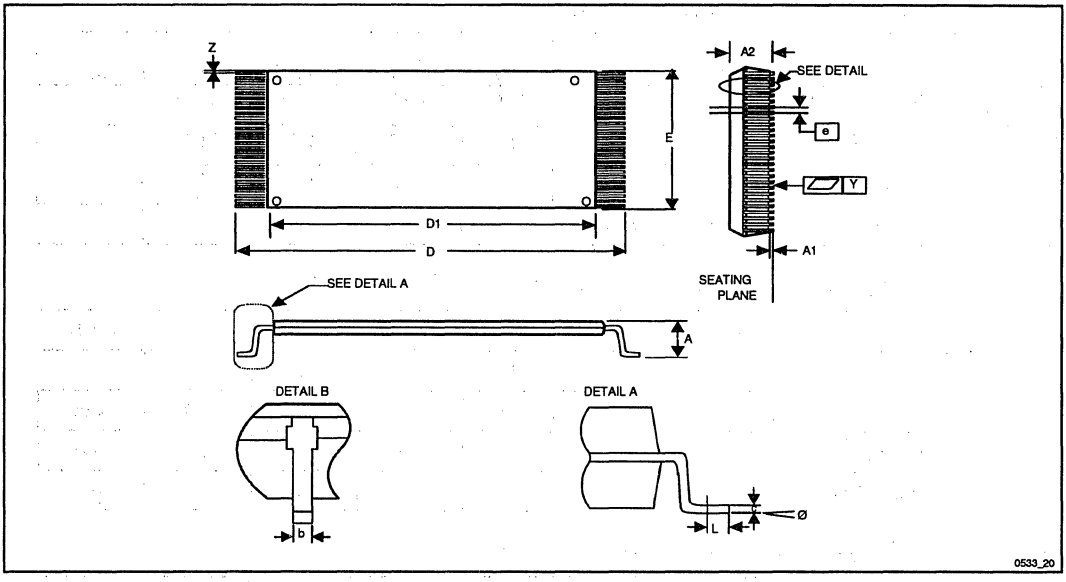
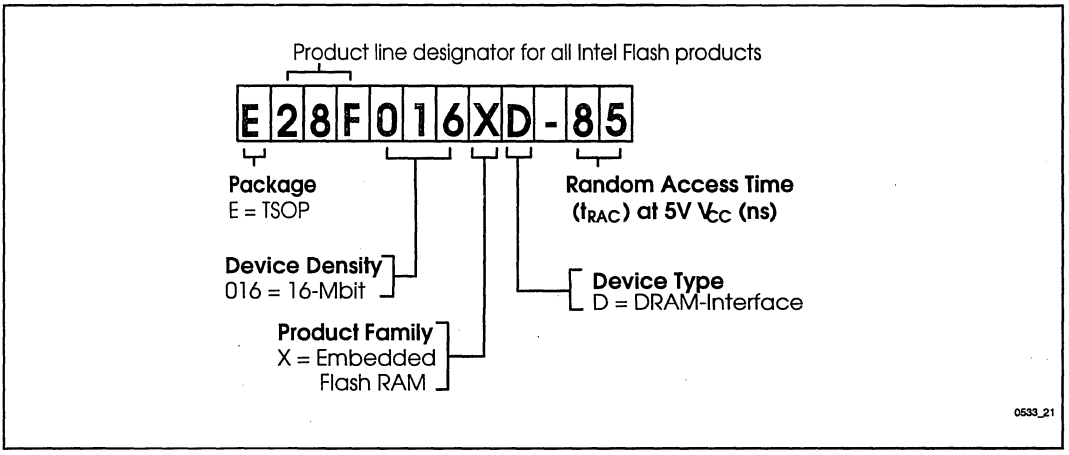


Figure 20. Mechanical Specifications of the 28F016XD 56-Lead TSOP Type I Package

Family: Thin Small Out-Line Package				
Symbol	**Millimeters**			**Notes**
	Minimum	**Nominal**	**Maximum**	
A			1.20	
A_1	0.50			
A_2	0.965	0.995	1.025	
b	0.100	0.150	0.200	
c	0.115	0.125	0.135	
D_1	18.20	18.40	18.60	
E	13.80	14.00	14.20	
e		0.50		
D	19.80	20.00	20.20	
L	0.500	0.600	0.700	
N		56		
Ø	0°	3°	5°	
Y			0.100	
Z	0.150	0.250	0.350	

DEVICE NOMENCLATURE AND ORDERING INFORMATION



Order Code	Valid Combinations	
	V_{CC} = 3.3V ± 0.3V, 50 pF load, 1.5V I/O Levels(1)	V_{CC} = 5.0V ± 10%, 100 pF load, TTL I/O Levels(1)
E28F016XD 85	E28F016XD-95	E28F016XD-85

NOTE:
1. See Section 5.2 for Transient Input/Output Reference Waveforms.

ADDITIONAL INFORMATION

Order Number	Document/Tool
297372	16-Mbit Flash Product Family User's Manual
292152	AB-58, "28F016XD-Based SIMM Designs"
292165	AB-62, "Compiling Optimized Code for Embedded Flash RAM Memories"
292092	AP-357, "Power Supply Solutions for Flash Memory"
292123	AP-374, "Flash Memory Write Protection Techniques"
292126	AP-377, "16-Mbit Flash Product Family Software Drivers, 28F016SA/SV/XD/XS"
292131	AP-384, "Designing with the 28F016XD"
292163	AP-610, "Flash Memory In-System Code and Data Update Techniques"
292168	AP-614, "Using the 28F016XD in Embedded PC Designs"
294016	ER-33, "ETOX™ Flash Memory Technology—Insight to Intel's Fourth Generation Process Innovation"
297508	FlashBuilder Utility
Contact Intel/Distribution Sales Office	28F016XD Benchmark Utility
Contact Intel/Distribution Sales Office	Flash Cycling Utility
Contact Intel/Distribution Sales Office	28F016XD iBIS Models
Contact Intel/Distribution Sales Office	28F016XD VHDL/Verilog Models
Contact Intel/Distribution Sales Office	28F016XD Timing Designer Library Files
Contact Intel/Distribution Sales Office	28F016XD Orcad and ViewLogic Schematic Symbols

DATASHEET REVISION HISTORY

Number	Description
001	Original Version
002	Removed support of the following features: • All page buffer operations (read, write, programming, Upload Device Information) • Command queuing • Software Sleep and Abort • Erase All Unlocked Blocks • Device Configuration command Changed definition of "NC." Removed "No internal connection to die" from description. Added "xx" to Upper Byte of Command (Data) Definition in Sections 4.2 and 4.3. Modified parameters "V" and "I" of Section 5.1 to apply to "NC" pins. Increased I_{PPS} (V_{PP} Read Current) for $V_{PP} > V_{CC}$ to 200 µA at V_{CC} = 3.3V/5.0V. Changed V_{CC} = 5.0V DC Characteristics (Section 5.5) marked with Note 1 to indicate that these currents are specified for a CMOS rise/fall time (10% to 90%) of <5 ns and a TTL rise/fall time of <10 ns. Corrected "RP# high to RAS# going low" to be a "Min" specification at V_{CC} = 3.3V/5.0V. Increased Typical "Word/Block Write Times" (t_{WHRH1}/t_{WHRH3}) for V_{PP} = 5.0V: t_{WHRH1} from 24.0 µs to 35.0 µs and t_{WHRH3} from 0.8 sec to 1.2 sec at V_{CC} = 3.3V t_{WHRH1} from 16.0 µs to 25.0 µs and t_{WHRH3} from 0.6 sec to 0.85 sec at V_{CC} = 5.0V Changed "Time from Erase Suspend Command to WSM Ready" spec name to "Erase Suspend Latency Time to Read"; modified typical values and added Min/Max values at V_{CC} =3.3/5.0V and V_{PP} =5.0/12.0V (Section 5.10). Minor cosmetic changes throughout document.

intel®

TECHNICAL PAPER

Interfacing the 28F016XS to the Intel486™ Microprocessor Family

KEN MCKEE
TECHNICAL MARKETING ENGINEER

PHILIP BRACE
APPLICATIONS ENGINEER

February 1995

Order Number: 297504-002

1.0 INTRODUCTION

This technical paper describes designs interfacing the high performance 28F016XS flash memory to the Intel486™ microprocessor. These designs are based on preliminary 28F016XS specifications. Please contact your Intel or distribution sales office for up-to-date information.

The 28F016XS is a 16-Mbit flash memory with a synchronous pipelined read interface. This optimized flash memory interface delivers equivalent or better read performance compared to DRAM. The 28F016XS combines ROM-like non-volatility, DRAM-like read performance and in-system updateability into one memory technology. These inherent capabilities will improve performance and lower the over-all system cost. The 28F016XS delivers optimal performance when interfacing to a burst processor, such as the Intel486 microprocessor. The Intel486 microprocessor sees widespread use in a variety of applications ranging from the PC to numerous embedded products, while providing code compatibility with thousands of commercially available software packages and the performance necessary for today's leading-edge systems. The Intel486 microprocessor's bus interface provides a burst transfer mechanism whereby four consecutive data items are fetched in one access sequence. The 28F016XS's synchronous pipelined read interface makes special use of the burst transfer mechanism to achieve extremely high read performance.

When interfacing the 28F016XS to a processor that executes an Intel or linear burst cycle, up to three simultaneous read accesses can be pipelined into the 28F016XS, sustaining a high read transfer rate. At 33 MHz, the 28F016XS-15 delivers zero wait-state performance after the initial pipeline fill. This enhanced read performance eliminates the costly expense of shadowing code from slow non-volatile memory (ROM, hard disk drive, etc.) to fast DRAM for increased system performance. The 28F016XS enables direct code execution out of the flash memory array, eliminating unnecessary software and hardware overhead involved in shadowing code.

In an Intel486 microprocessor-based environment, BAPCo benchmarking analysis revealed a *13%* system performance improvement using the 28F016XS-15 over 70 ns DRAM.

In addition to the increased read performance, the 28F016XS offers an Intel486 microprocessor-based system a low power, non-volatile memory that is electrically updateable via local processor control. The 28F016XS's low power consumption reduces system power dissipation and heat emission, and its updateability increases code flexibility and system reliability. Combined, the 28F016XS and the Intel486 microprocessor deliver a high performance, low power and cost-effective system solution.

The Intel486 microprocessor interface to the 28F016XS requires minimal logic while offering significant system enhancements. One programmable logic device (PLD), a 22V10-15, generates and monitors all 28F016XS and Intel486 microprocessor control signals. This technical paper explores the interface between the 28F016XS-15 and the Intel486™ SX-33 microprocessor, describing the interface circuitry, explaining the read and write cycles and providing the interfacing PLD equations. It also provides detailed design suggestions for interfacing the 28F016XS to other Intel486 microprocessors.

2.0 OPTIMIZED 28F016XS / INTEL486 SX MICROPROCESSOR INTERFACE

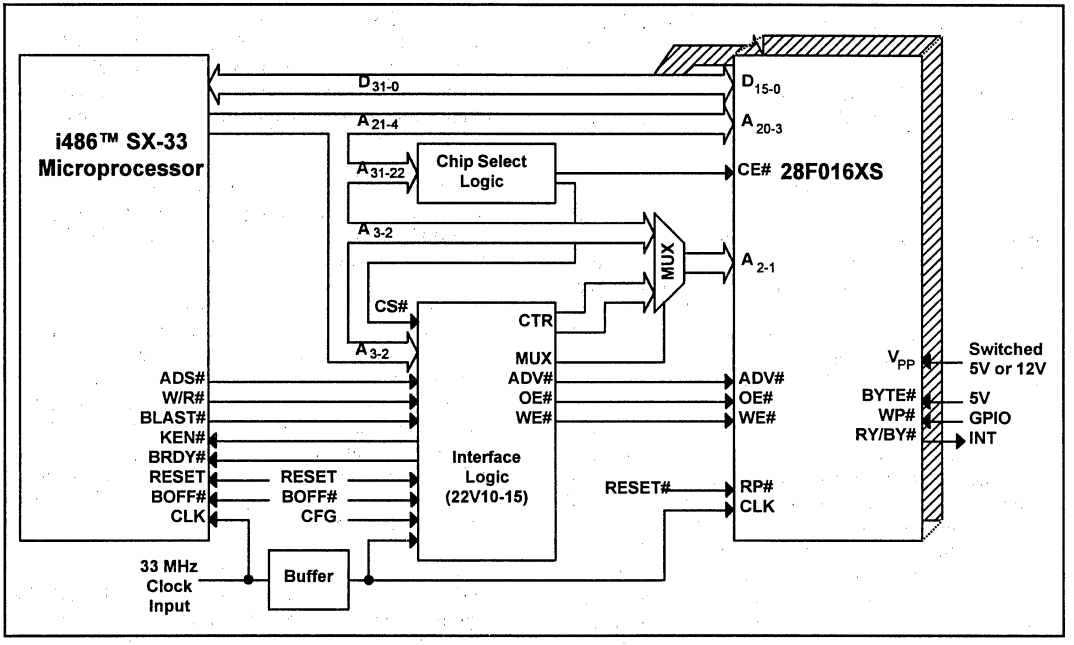


Figure 1. Optimized 28F016XS Interface to the Intel486 Microprocessor with Wait-State Profile of 2-0-0-0 Up to 33 MHz

The 28F016XS-15 interface to the Intel486 SX-33 microprocessor, illustrated in Figure 1, delivers 2-0-0-0 wait-state read performance. Consult your Intel or distribution sales office for schematic and PLD files for this design.

See Section 3.0 for an alternative design.

2.1 Circuitry Description

This section will describe the circuitry involved in this design.

Memory Configuration

This design uses two 28F016XS-15s, each configured in x16 mode and arranged in parallel to match the Intel486 SX microprocessor's 32-bit data bus. This memory configuration provides 4 Mbytes of flash memory for system usage. Signals A_{21-4} from the Intel486 SX microprocessor and CTR_{1-0} from the PLD select locations within the 28F016XS memory space, arranged as 1 Meg double words. The two-bit counter implemented in the PLD supplies consecutive burst addresses to the 28F016XSs.

Reset

The Intel486 SX microprocessor requires an active high reset signal, while the 28F016XSs use an active low RESET#. Figure 2 illustrates a suggested logic configuration for generating both an active high and low reset signal. The active high RESET controls the Intel486 SX microprocessor and PLD RESET inputs, while the active low RESET# drives the 28F016XS RP# input.

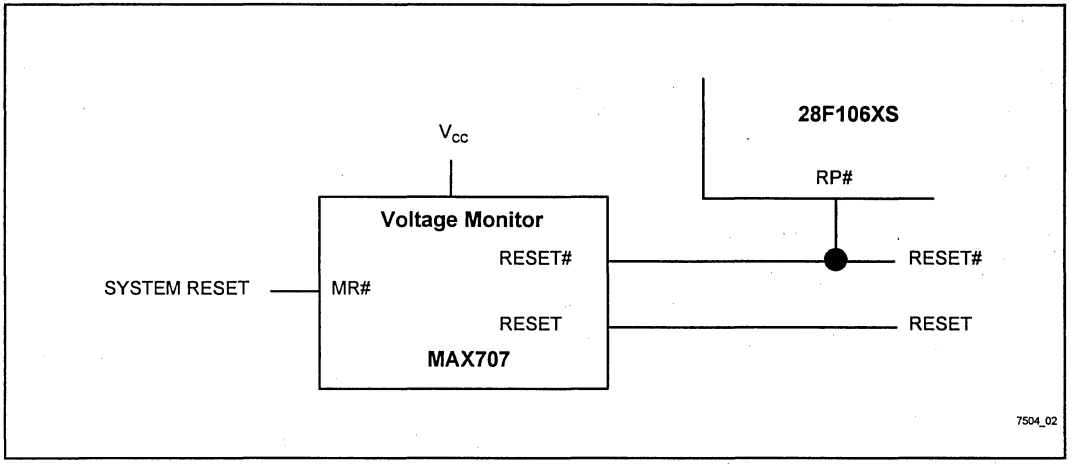


Figure 2. RESET Generation Method

Chip Select Logic

Chip select decode logic may use A_{31-22} to generate active low chip select signals, CE_X#, for the 28F016XS memory space and other system peripherals. The chip select addressing the 28F016XS memory space drives CE_0# on each 28F016XS-15 and a control input to the PLD. The 28F016XS-15s' CE_1# inputs are grounded. For many systems, using the upper address bits in a linear selection scheme may provide a sufficient number of chip select signals, thus eliminating chip select decode logic. (See Figure 3 for an example of using linear selection for chip selects.) When using a linear chip select scheme however, software must avoid using addresses that may select more than one device, which could result in bus contention. For example, addresses 01000000H through 010FFFFFH drive both A_{22} and A_{23} to a logic "0," which inadvertently selects two peripheral devices.

CLK Option

A 33 MHz CLK drives the Intel486 SX microprocessor. The buffer in Figure 1 delays this processor CLK input and drives the PLD and the 28F016XS-15s. The buffer introduces an intentional system clock skew. This skew provides additional time for the processor to meet the 28F016XSs' address setup time.

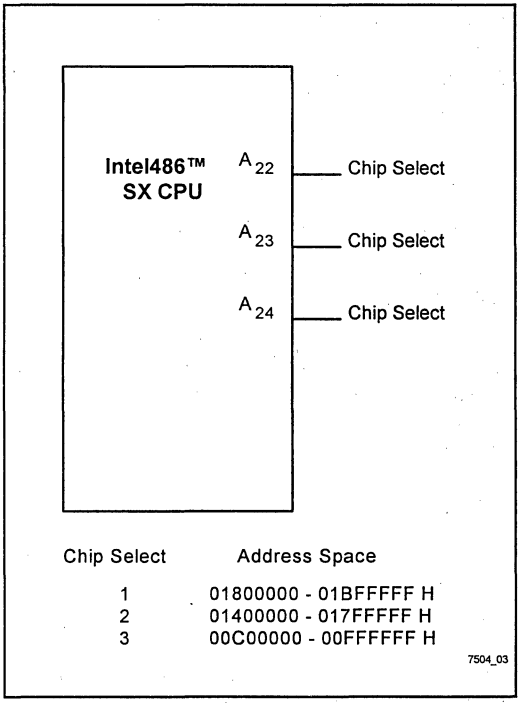


Figure 3. Example of Using Linear Chip Selection with Active Low Chip Select Signals

Multiplexer (MUX)

To achieve this type of wait-state profile, the Intel486 SX microprocessor directly loads the 28F016XS with the address of the first read access. The interface logic enables the MUX to permit the processor's lower address lines A_{3-2} access to the lower flash memory address lines during the initial access of a burst or single read transaction. Next, the interface logic switches the data flow path through the MUX in anticipation of a burst transaction. The two-bit counter integrated into the control logic then takes over driving the 28F016XSs' A_{2-1}. The counter supplies the flash memory with consecutive burst addresses for the remaining duration of the transaction.

Interface Control Signals

The interfacing state machine monitors the Intel486 SX microprocessor's external bus signals to control the two-bit counter and generate OE#, WE# and ADV# signals to the 28F016XS-15s. At the beginning of the burst cycle, the interface logic loads the two-bit counter. The state machine also generates KEN# and BRDY# signals, informing the Intel486 SX microprocessor of the nature of the bus cycle.

Configuration Signal

A general purpose input/output (GPIO) generates the configuration signal input to the state machine. The configuration signal must reset to logic "0" on power-up and system reset to ensure that the operation of the state machine matches the initial SFI Configuration of the 28F016XS-15s. After optimizing the SFI Configuration, the GPIO must switch to logic "1" in order to take advantage of the optimized flash memory state. See Section 2.3 for more information regarding the configuration signal.

Additional 28F016XS Control Signals

The BYTE# input to the 28F016XS-15s is tied to 5.0V to configure the 28F016XS-15s for x16 mode, and A_0 is tied to GND (A_0 is only used for byte addressing). A GPIO controls the write protect input, WP#, to the 28F016XS-15s. The 28F016XS is compatible with either a 5.0V or a 12.0V V_{PP} voltage and is completely protected from data alteration when V_{PP} is switched to GND. With $V_{PP} \leq V_{PPLK}$, the 28F016XS will not successfully complete Data Write and Erase operations, resulting in absolute flash memory data protection. Figure 1 also illustrates the 28F016XS-15's RY/BY# output connecting directly to a system interrupt, which enables background Write/Erase operations. RY/BY#, WP#, and V_{PP} implementations are application dependent. Consult the Additional Information section of this technical paper for documentation covering these topics in more detail.

2.2 Interfacing Signal Definitions

The interface logic that controls the 28F016XS-15 interface to the Intel486™ SX-33 microprocessor monitors and regulates specific system signals. The next two sections describe these signals in detail.

2.2.1 28F016XS Signal Descriptions

This section describes the 28F016XS signals that are pertinent to this design.

ADV# - Address Valid (Input)

This active low signal informs the 28F016XS that a valid address is present on its address pins. ADV#, in conjunction with a rising CLK edge, initiates a read access to the 28F016XS. This signal is ignored during write operations.

CLK - Clock (Input)

CLK provides the fundamental timing and internal operating read frequency for the 28F016XS. CLK initiates read accesses (in conjunction with ADV#), times out the SFI Configuration, and synchronizes device outputs. CLK can be slowed or stopped with no loss of data synchronization. This signal, like ADV#, is ignored during write operations.

OE# - Output Enable (Input)

This active low signal activates the 28F016XS's output buffers when OE# equals "0". The outputs tri-state when OE# is driven to "1".

WE# - Write Enable (Input)

This active low signal controls access to the Control User Interface (CUI). Addresses (command or array) and data are latched on the rising edge of WE# during write cycles.

2.2.2 INTEL486 SX Microprocessor Signal Descriptions

This section describes the Intel486 SX microprocessor signals that are relevant to this interface. This interface assumes processor inputs are driven by only one controlling device (the PLD). If more than one device drives a processor input, the PLD output should be configured as open drain to avoid signal contention. Many PLDs, FPGAs and ASICs provide output configuration capability.

ADS# - Address Status (Output)

This active low output signal from the Intel486 SX microprocessor indicates the presence of valid bus cycle and address signals on the bus. ADS# is driven in the same clock as the address signals. Typically, external circuitry uses ADS# to indicate the beginning of a bus cycle.

KEN# - Cache Enable (Input)

This active low input to the Intel486 SX microprocessor determines whether data being returned in the current bus cycle will be cached. In order for the current data to be cached, KEN# must be returned active in the clock prior to the first RDY# or BRDY# of the cycle and must also be returned active in the last clock of the data transfer.

BRDY# - Burst Ready (Input)

This active low input to the microprocessor performs the same function during a burst cycle as RDY# performs during a non-burst cycle. During a burst cycle, BRDY# is sampled on the rising edge of every clock. Upon sampling BRDY# active, the data on the data bus will be latched into the microprocessor (for burst reads). ADS# will be negated during the second transfer of the burst cycle; however, the lower address lines and byte enables may change to indicate the next data item requested by the processor.

BLAST# - Burst Last (Output)

This active low output from the microprocessor signals the final transfer in a burst cycle. The next time BRDY# is returned, it will be treated the same as RDY# and thus terminate any multiple cycle transfers.

2.3 System Interface Requirement

The system logic controlling the 28F016XS-15 interface to the Intel486 SX microprocessor incorporates an initial and an optimized read configuration, which correlates to specific SFI Configuration values. The interface read configuration is dependent upon the value of CFG (PLD input). CFG informs the interface of the SFI Configuration status. Note, the SFI Configuration status does not affect Write operations.

Initial Read Configuration

Upon power-up/reset, the 28F016XS-15 defaults to a SFI Configuration value of 4, and the interface logic supports burst read accesses to the flash memory space. The interface returns BRDY# to inform the processor that the interface supports burst read transaction. A general purpose input/output (GPIO) informs the system interface of the status of the SFI Configuration.

The GPIO entitled CFG is set to logic "0" on power-up/reset. With CFG driven low, the state machine correctly matches the 28F016XS-15s' default configuration.

Optimized Read Configuration

At 33 MHz, the 28F016XS-15 operates at highest performance with a SFI Configuration value set to 2. To reconfigure the 28F016XS-15, program control should jump to an area of RAM to execute the configuration sequence. After reconfiguring the 28F016XS-15, the GPIO value must change to logic "1," in order to take advantage of the 28F016XS-15's optimized configuration. A pseudocode flow for this configuration sequence is shown below.

Execute Device Configuration command sequence
Activate CFG signal
End

In the optimized read configuration, the system logic supports burst cycles by generating BRDY#, which informs the microprocessor that the memory subsystem is capable of handling a burst transfer. The 28F016XS-15 memory array, after the initial pipeline fill delay from the first access, transfers data to the processor at a rate of 133 Mbytes/sec.

2.4 Read Control for Burst Transactions

The interface logic controlling the handshaking between the 28F016XS and processor performs one of two different read cycles, depending upon the CFG input signal.

Read Abort Condition

A read cycle will abort only when an external system bus master asserts BOFF#, which forces the processor to give immediate bus control to the requester. When this situation occurs, the Intel486 SX microprocessor floats the address bus, which will cause the address decode logic to de-select the 28F016XS memory space. Monitoring BOFF#, the interfacing logic will transition to an idle state and wait for the processor to re-initiate the interrupted bus cycle after the bus master has relinquished the bus to the processor. OE# is immediately driven high, deactivating the 28F016XS-15's output buffers, upon detecting BOFF# driven active. This BOFF# condition can occur in both the initial and optimized configurations described in the paragraphs that follow.

Initial Configuration

Refer to Figures 4 and 5 for the following read cycle discussion.

With CFG set to logic "0," the interfacing read state machine executes cacheable burst read cycles. This configuration occurs upon power-up and reset.

Initially, the interface logic drives ADV# and MUX active while waiting for the Intel486 SX processor to initiate a bus cycle targeting the 28F016XS. With the MUX active, the processor's A_{3-2} drive the lower flash memory address lines in anticipation of a flash memory access. During this anticipation state, CE# is active to prevent a t_{ELCH} violation on the first access initiated by the processor. The delayed CLK also prevents a possible timing violation, providing the processor with sufficient time to meet the 28F016XSs' t_{AVCH} specification when initiating the first access.

When the microprocessor does initiate a read access to the 28F016XS memory space, it will provide an address, drive W/R# low and activate ADS#. Monitoring these signals, the state machine transitions into read control.

If ADS# = 0 and W/R# = 0 then READ CONTROL

At this point, CFG and CS# are examined to determine the configuration status of the 28F016XS-15s, and whether or not the current address targets the 28F016XS memory space. If CS# = "1," the state machine returns to an idle state waiting for a new access. Otherwise, the state machine will continue the read access, regulating ADV#, BRDY# and OE#.

At N = 1 (Figure 5), the interfacing read state machine loads and increments the two-bit counter. The counter is incremented because the processor supplies the flash memory with the initial address. The counter then provides the flash memory with the subsequent burst addresses throughout the remaining duration of the bus transaction.

With ADV# at logic "0," the interface initiates a read access to the 28F016XS-15s at N = 1. Next, ADV# immediately switches to a logic "1" at N = 1 and then toggles active on every other clock edge until N = 8. After this time, ADV# will remain inactive.

In the default SFI Configuration (SFI Configuration = 4), the first data will be accessible to the processor at N = 6. The rest remaining data will be available for the processor to retrieve at N = 8, 10 and 12. The 28F016XS-15's output buffers are enabled at N = 3 and BRDY# is driven low at N = 5. The processor will sample BRDY# active and latch the information residing on the data bus at N = 6. If the processor drives BLAST# inactive, indicating a burst transaction is in process, the interface logic will drive BRDY# active on every other clock edge until BLAST# is sampled active by the interface logic. Then the state machine will transition to an idle state where it deactivates OE# and waits for the processor to initiate a new bus cycle.

Initial Configuration Timing Consideration

In this initial read configuration design, there are important timing considerations that need to be taken into consideration.

First, the buffer delay can cause possible timing violations if not chosen correctly. The purpose of the buffer is to provide time for the processor to load the 28F016XSs with the initial address during read transactions. Therefore, the buffer must have a minimum delay which satisfies the flash memory's' t_{AVCH}.

$$t_{AVCH} + t_6 - 1/33\ MHz = 1\ ns$$

The buffer can also affect the processor's data setup time. Hence, the buffer must have a maximum delay of no greater than:

$$1/33\ MHz - t_{CHQV} - t_{22} = 5\ ns$$

Another important timing parameter is the Intel486 SX microprocessor's data hold time. Since the 28F016XS specifies a 0 ns guaranteed data hold time from CE# or OE# high, these two signals must be driven active until the processor's hold time is satisfied. CE# hold delay will not be concern because CE# is held active during the state machine's idle state. OE# has only .5 ns of margin to the processor's specification for the buffer used in this design. OE# hold time equals:

$$t_{PZX(min)} + t_{PHL(min)} = 3.5\ ns$$

Consult the appropriate datasheets for full timing information.

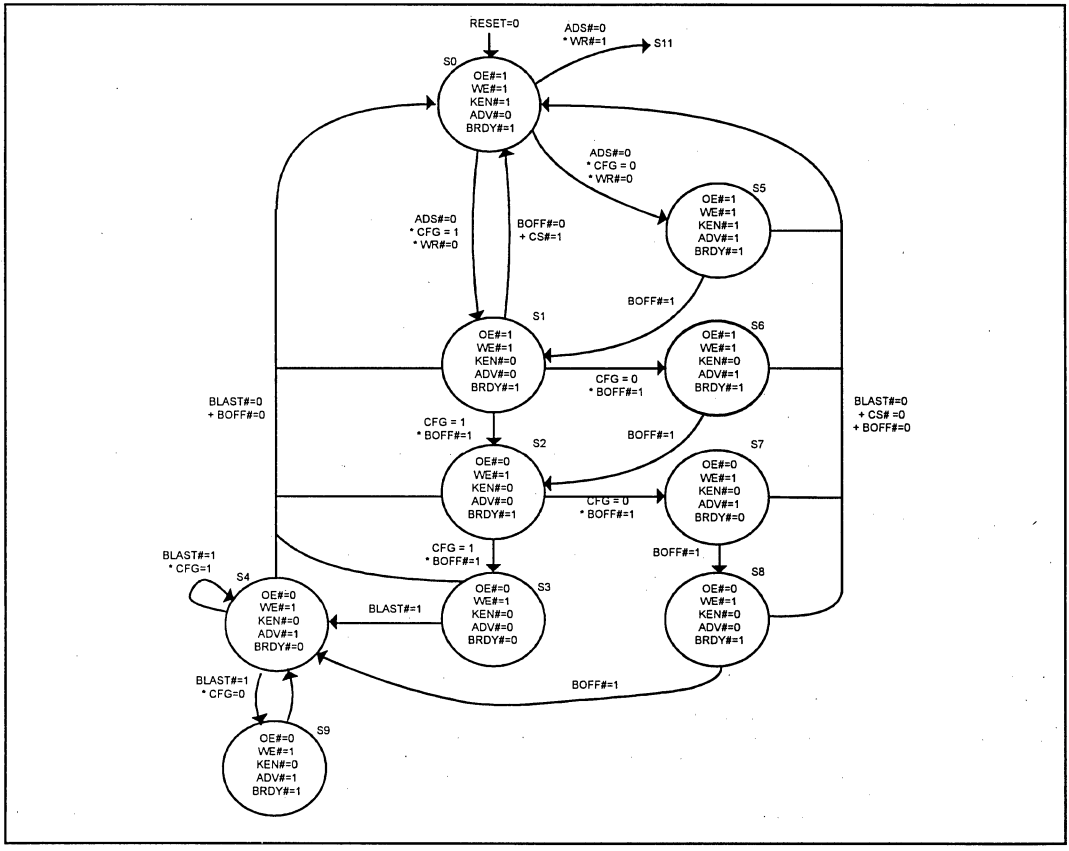


Figure 4. Optimized Read State Diagram for Burst Read Control (Interface Shown in Figure 1)

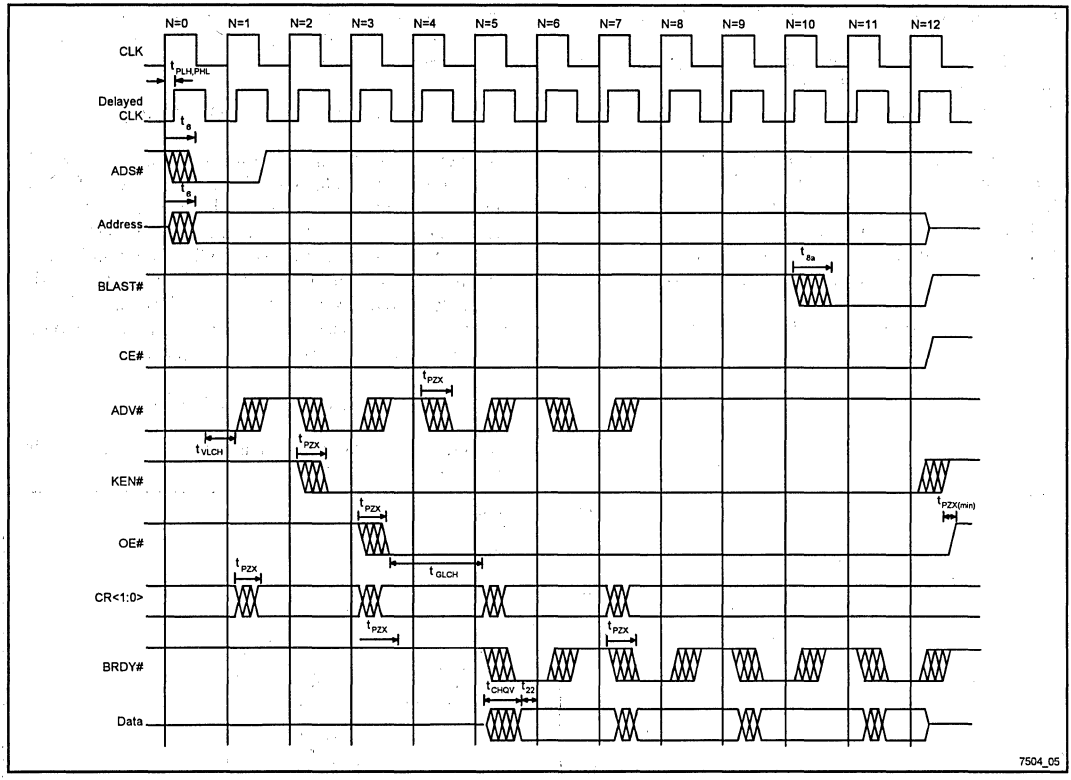


Figure 5. Example Initial Burst Read Cycle Showing Key Timing Specifications Requiring Consideration

Table 1. Example Optimized Read Cycle Specifications at 5V V_{CC}

Symbol	Description	Min	Max	Unit
$t_{PHL,PLH}$	Buffer Delay	1.5	5	ns
t_6	ADS# Delay (Intel486 SX-33 microprocessor)	3	16	ns
t_{8a}	BLAST# Delay (Intel486 SX-33 microprocessor)	3	20	ns
t_{22}	D_{31-0} Setup Time (Intel486 SX-33 microprocessor)	5		ns
t_{ELCH}	CE_X# Setup Time to CLK (28F016XS-15)	25		ns
t_{VLCH}	ADV# Setup Time to CLK (28F016XS-15)	15		ns
t_{GLCH}	OE# Setup Time to CLK (28F016XS-15)	15		ns
t_{CHQV}	CLK to Data Delay (28F016XS-15)		20	ns
t_{PZX}	CLK Output Delay (22V10-15)	2	8	ns

NOTE:
Consult appropriate datasheets for up-to-date specifications.

Optimized Configuration

Refer to Figures 4 and 6 for the following discussion.

With the 28F016XS-15s in the optimized configuration (SFI Configuration = 2 at 33 MHz), and CFG set to a logic "1" value, the system interface executes cacheable burst cycles.

Like the initial configuration, the connecting logic initially drives ADV# and MUX active while waiting for the Intel486 SX processor to initiate a bus cycle targeting the 28F016XS. With the MUX active, the processor's A_{3-2} drive the lower flash memory address lines in anticipation of a flash memory access. During this anticipation state, CE# is active to prevent a t_{GLCH} violation on the first access initiated by the processor. The delayed CLK also inhibits also possible timing violations from occurring. It provides the processor with sufficient time to meet the 28F016XSs' t_{AVCH} specification when initiating the first access.

When the processor drives ADS# low, it notifies the interface logic that a valid address is on the address bus. Monitoring the external bus of the Intel486 SX microprocessor, the state machine then transitions into read control.

If ADS# = 0 and W/R# = 0 then READ CONTROL

In optimized read control, the state machine controls OE#, KEN# and BRDY#. If CS# = "0," the state machine at N = 1 loads and increments the two-bit counter, switches the data flow path through the MUX and holds ADV# active for the next three consecutive clock periods. While ADV# is driven low, the counter increments through the Intel burst order (Table 2), supplying the 28F016XS-15s with a new address at N = 2, 3 and 4. If CS# ="1," the state machine returns to an idle state waiting for a new memory access.

Table 2. Intel Burst Order (A_{3-2})

First Address	Second Address	Third Address	Fourth Address
0	4	8	C
4	0	C	8
8	C	0	4
C	8	4	0

At N = 2, the state machine drives KEN# active and holds it active until the end of the burst cycle, thereby executing a cache line fill.

The state machine activates the 28F016XS-15 output buffers (OE# driven to a logic "0" value) at N = 2 and holds them active throughout the burst read cycle.

With the SFI Configuration value set to 2, data will be available at N = 4, 5, 6 and 7. Driving BRDY# low at N = 3, the Intel486 SX microprocessor will sample BRDY# active at N = 4, which informs the processor of valid information on data pins D_{31-0} and that the 28F016XS memory space supports a burst read transfer. BRDY# is held low until the end of the burst cycle while the processor retrieves data on every rising clock edge. BRDY# is driven high upon sampling BLAST# low, marking the end of the burst cycle. Then the state machine will transition to and idle state where it deactivates OE# and waits for the processor to initiate a new bus cycle.

Optimized Configuration Timing Considerations

In the optimized configuration, the same timing consideration regarding the buffer propagation delay and OE# hold time require attention. For information regarding these concerns, see Section 2.4 Initial Configuration Timing Considerations.

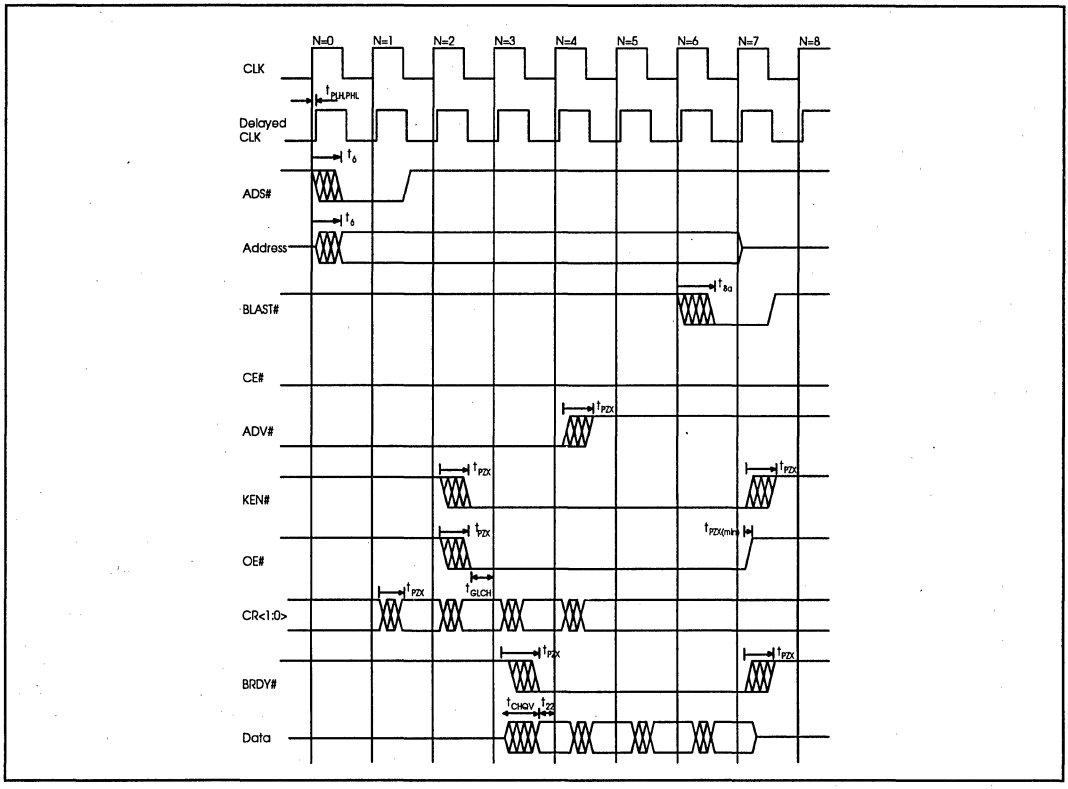


Figure 6. Example Burst Read Timing Waveform Illustrating Key Timing Specifications Requiring Consideration

Table 3. Example Optimized Read Cycle Specifications at 5V V_{CC}

Symbol	Description	Min	Max	Unit
$t_{PHL,PLH}$	Buffer Delay	1.5	5	ns
t_6	ADS# Delay (Intel486 SX-33 microprocessor)	3	16	ns
t_{8a}	BLAST# Delay (Intel486 SX-33 microprocessor)	3	20	ns
t_{22}	D_{31-0} Setup Time (Intel486 SX-33 microprocessor)	5		ns
t_{ELCH}	CE_X# Setup Time to CLK (28F016XS-15)	25		ns
t_{VLCH}	ADV# Setup Time to CLK (28F016XS-15)	15		ns
t_{GLCH}	OE# Setup Time to CLK (28F016XS-15)	15		ns
t_{CHQV}	CLK to Data Delay (28F016XS-15)		20	ns
t_{PZX}	CLK Output Delay (22V10-15)	2	8	ns

NOTE:
Consult appropriate datasheets for up-to-date specifications.

2.5 Write Cycle Control

Refer to Figures 7 and 8 for the following write cycle discussion.

Write Abort Condition

A write cycle will abort only when an external system bus master asserts BOFF#, which forces the processor to give immediate bus control to the requester. When this situation occurs, the Intel486 SX microprocessor will float the address bus, causing the address decode logic to de-select the 28F016XS memory space. The interfacing logic, monitoring BOFF#, will transition to an idle state where it will wait for the processor to re-initiate the interrupted bus cycle after the bus master has relinquished the bus to the processor. WE# is immediately deactivated upon sensing BOFF# low.

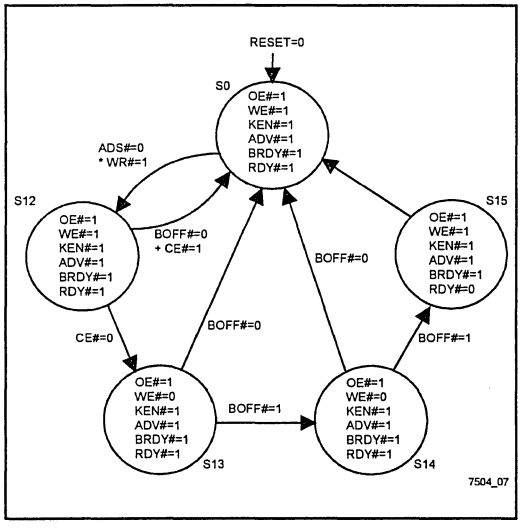


Figure 7. Non-Burst Write State Diagram Controlling the Interface Shown in Figure 1

Write Cycle Description

The 28F016XS-15 executes asynchronous write cycles like traditional flash memory components such as the 28F016SA/SV. The SFI Configuration does not influence write operations; therefore, the interfacing state machine does not examine CFG once detecting a write cycle.

During the first clock period, the Intel486 SX microprocessor drives ADS# low and W/R# high. The state machine, upon detecting a write cycle, immediately switches the data flow path through the MUX. The processor does not drive the flash memory's lower address lines during write cycles. The counter loads and supplies the address to the 28F016XSs' lower two address lines, A_{2-1}. The state machine then transitions to write control at N =1. The counter only supplies the 28F016XS-15 with one address throughout the entire write operation. A write transaction must complete fully before issuing a second write operation.

If ADS# = 0 and W/R# = 1 then WRITE CONTROL

In write control, the state machine performs WE#-controlled command write operations to the 28F016XS-15s. Data is written to the 28F016XS memory space via processor control. The interface only supports double word writes.

For the next two clock periods the state machine holds WE# low to satisfy the 28F016XS-15s' WE# active requirement. At N = 4, WE# transitions to a logic "1," which latches the address and data into the 28F016XS-15s.

BRDY# is not returned to the processor at N = 4 because the Intel486 SX microprocessor will only hold an address 3 ns after sampling BRDY# low. Instead, the interface activates BRDY# after N = 4, causing the processor to hold the address valid for an additional clock cycle, which satisfies the 28F016XS-15's address hold specification (t_{WHAX}). The state machine then returns to an inactive state at N = 5, waiting for a new memory access.

Write Timing Consideration

When performing a write operation, CS# is a critical system timing parameter, which must satisfy the interface logic's required setup time. The 22V10-15 requires a 9 ns setup time to CLK. Therefore, the system decode logic must generate a valid CS# to the interface within:

$$2 \times 1/33 \text{ MHz} - t_6 - t_{SU} = 35 \text{ ns}$$

Consult the appropriate datasheets for full timing information.

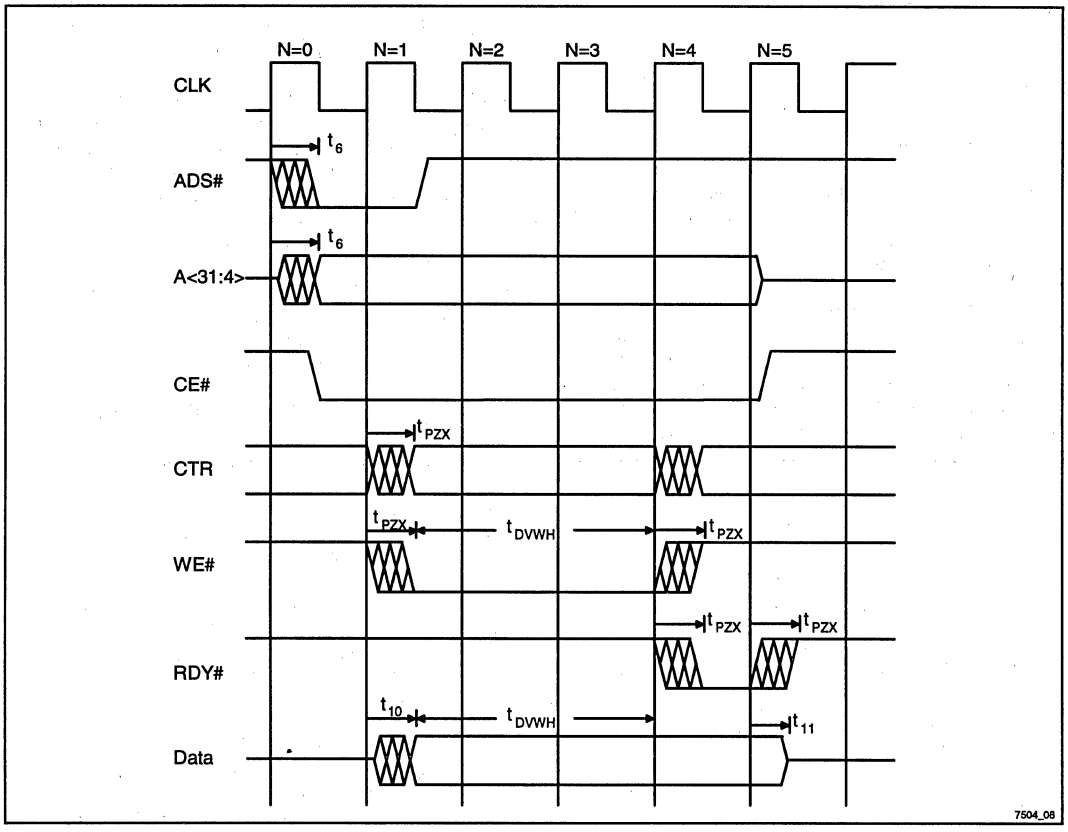


Figure 8. Example Write Cycle Showing Key Timing Specifications Requiring Consideration

Table 4. Example Write Cycle Timing Specifications at 5V V_{CC}

Symbol	Description	Min	Max	Unit
t_6	ADS# Delay(Intel486 SX-33 microprocessor)	3	16	ns
t_{10}	Data Write Valid Delay (Intel486 SX-33 microprocessor)	3	18	ns
t_{11}	Data Write Float Delay (Intel486 SX-33 microprocessor)		20	ns
t_{WLWH}	WE# Pulse Width (28F016XS-15)		50	ns
t_{DVWH}	Data Setup to WE# Going High (28F016XS-15)		50	ns
t_{PZX}	CLK Output Delay (22V10-15)	2	8	ns

NOTE:
Consult appropriate datasheets for up-to-date specifications.

3.0 STANDARD 28F016XS / INTEL486 SX MICROPROCESSOR INTERFACE

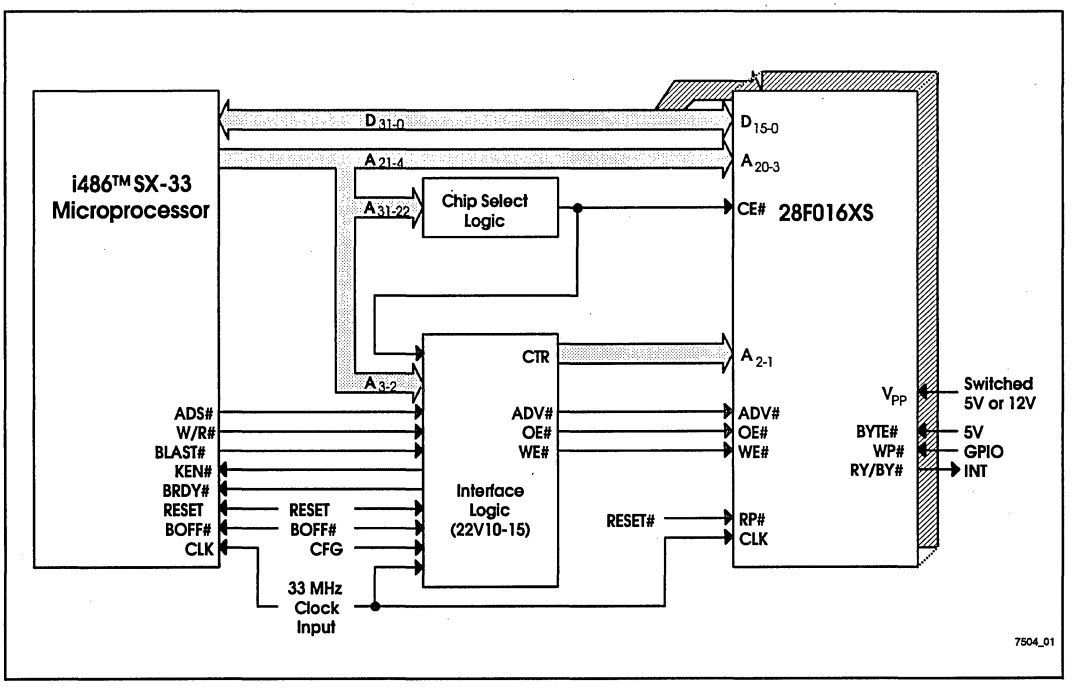


Figure 9. Minimal Glue Logic in Interfacing the 28F016XS-15 to the Intel486 SX-33 Microprocessor with a Wait-State Profile of 3-0-0-0 Up to 33 MHz

The 28F016XS-15 interface to the Intel486 SX-33 microprocessor illustrated in Figures 1 delivers 3-0-0-0 wait-state read performance. The design requires only one 22V10 to handle all interfacing requirements. Consult your Intel or distribution sales office for schematic and PLD equations for the interface documented in this section.

See Section 2.0 for an alternative design.

3.1 Interface Circuitry Description

This interface is extremely similar to the optimized design described in Section 2. The circuitry elements involved in the design are exactly the same except to for the elimination of the buffer and multiplexer in this interface. For specific circuitry information about the individual aspects of this design, refer to Section 2.

CLK Option

Unlike the optimized 28F016XS/Intel486 SX microprocessor interface, a buffer is not implemented in this design. The processor's 33 MHz CLK input drives both the PLD and flash memory. To reduce system clock skew, position the PLD and 28F016XSs within close proximity to the microprocessor.

3.2 Read Control For Burst Transactions

Similar to the optimized design described in Section 2, the read state machine will perform one of two different read cycles, depending upon the CFG input value. This section will concentrate on the differences between the read operations between the optimized and standard interface.

Initial Configuration

Refer to Figures 10 and 11 for the following read cycle discussion.

With CFG set to logic "0," the interfacing read state machine executes cacheable burst read cycles. This configuration will occur upon power-up and reset.

The microprocessor initiates a read access to the 28F016XS memory space by providing an address, driving W/R# low and activating ADS#. Monitoring the external bus of the Intel486 SX microprocessor, the state machine transitions into read control.

If ADS# = 0 and W/R# = 0 then READ CONTROL

At this point, CFG and CE# are examined to determine the configuration status of the 28F016XS-15s, and whether or not the current address targets the 28F016XS memory space. If CE# = "1," the state machine returns to an idle state waiting for a new access. Otherwise, the state machine will continue read access. In the initial configuration (CFG = "0"), read control will regulate ADV#, BRDY# and OE#.

At N = 1 (Figure 5), the counter loads address bits A_{3-2} and transitions ADV# low. With ADV# at logic "0," the interface initiates a read access to the 28F016XS-15s at N = 2. After initiating the read access, ADV# immediately switches to a logic "1" at N = 2 and then toggles active on every other clock edge until N = 8. After which, ADV# will remain inactive.

In the default SFI Configuration (SFI Configuration = 4), the first data will be accessible to the processor at N = 7. The rest of the data will be available for the processor to retrieve at N = 9, 11 and 13. The 28F016XS-15's output buffers are enabled at N = 4 and BRDY# is driven low at N = 6. The processor will sample BRDY# active and latch the information residing on the data bus at N = 7. If the processor drives BLAST# inactive indicating a burst transaction is in process, the interface logic will drive BRDY# active on every other clock edge until BLAST# is sampled active by the interface logic.

The Intel486 SX microprocessor requires a 3 ns hold time after sampling BRDY# active, therefore, the state machine will hold OE# active for 15 ns after the processor reads the last double-word. Then, the state machine will transition to an idle state where it deactivates OE# and waits for the Intel486 SX microprocessor to initiate a new bus cycle targeting the 28F016XS memory space.

Initial Configuration Timing Consideration

In the initial read configuration, CE# setup is a key system timing parameter.

To satisfy the 28F016XS-15 setup requirement, CE# must be valid 25 ns prior to the first rising CLK edge with ADV# = "0." Therefore, the maximum time allotted for the address decoding logic to generate CE# equals:

$$2*1/33\ \text{MHz} - t_6 - t_{ELCH} = 19\ \text{ns}$$

Consult the appropriate datasheets for full timing information.

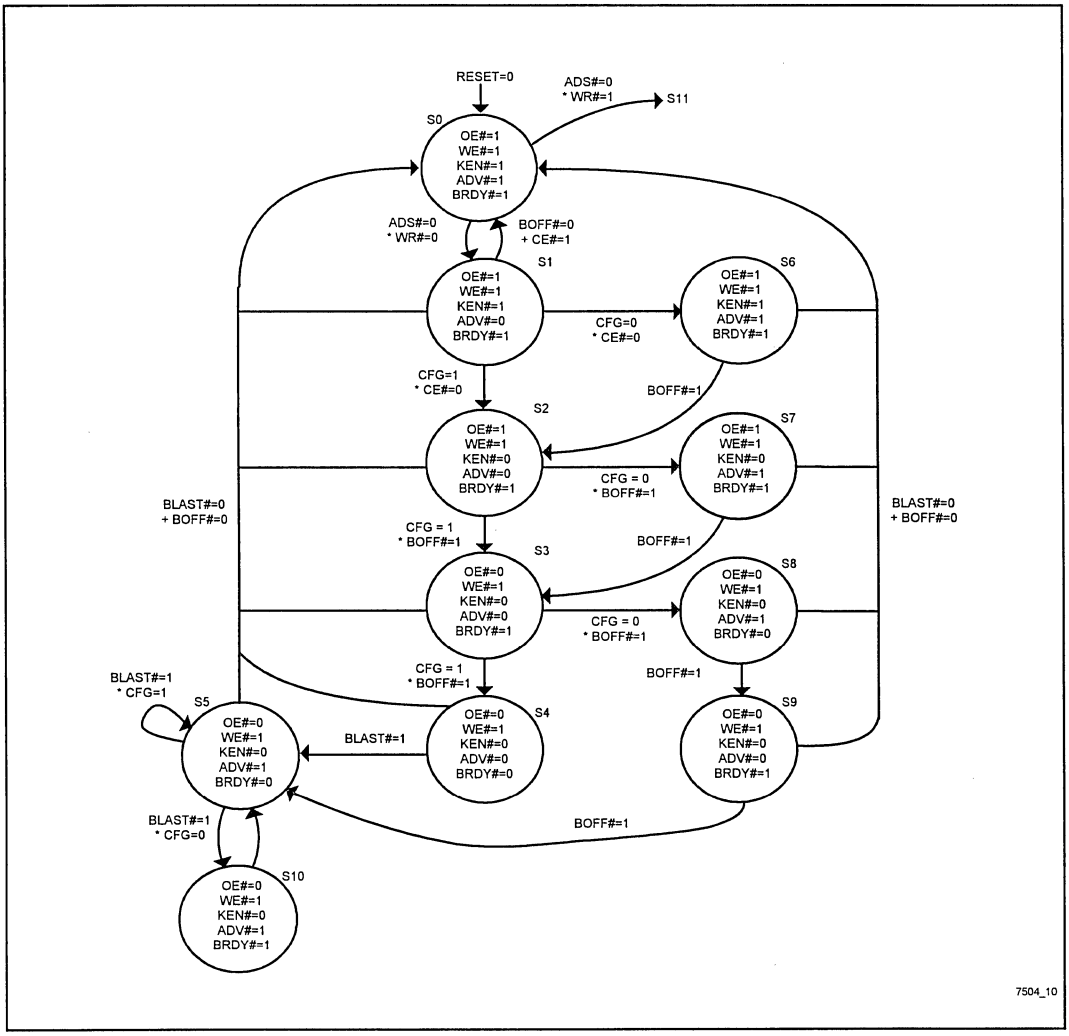


Figure 10. State Diagram of Single and Burst Read Control (Interface Shown in Figure 1)

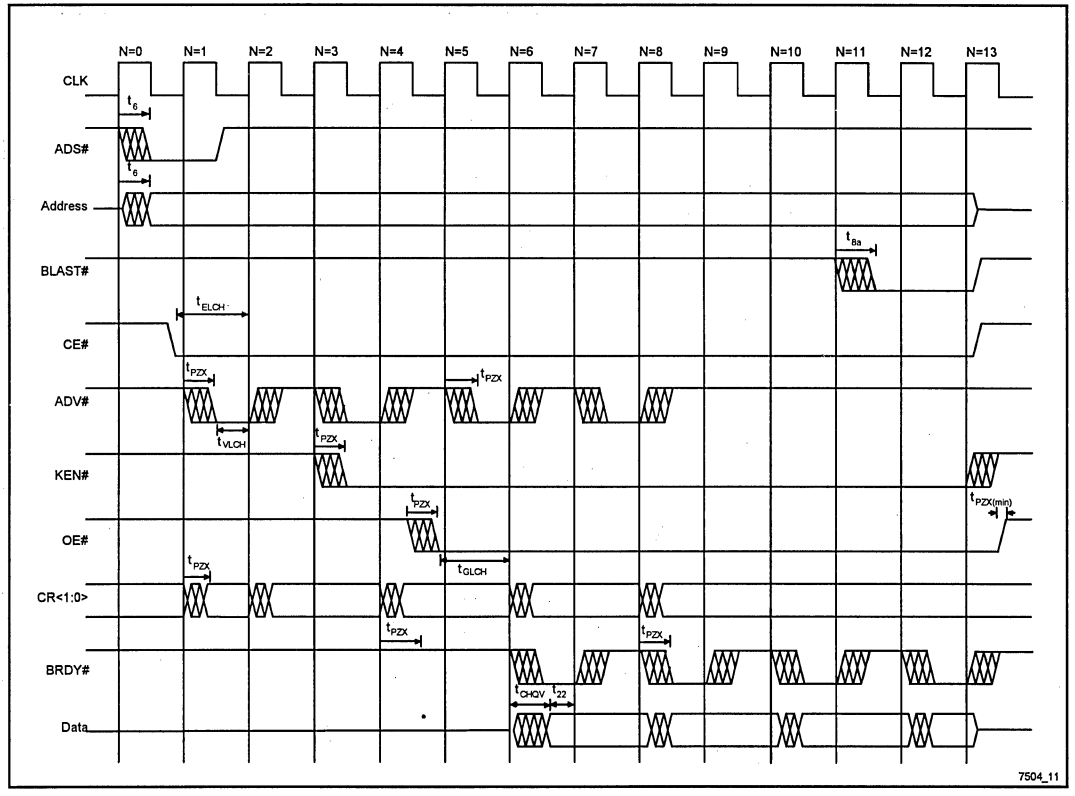


Figure 11. Example Initial Burst Read Cycle Showing Key Timing Specifications Requiring Consideration

Table 5. Example Initial Read Cycle Timing Specifications at 5V V_{CC}

Symbol	Description	Min	Max	Unit
t_6	ADS# Delay (Intel486 SX-33 microprocessor)	3	16	ns
t_{16}	RDY# Setup Time (Intel486 SX-33 microprocessor)	5		ns
t_{22}	D_{31-0} Setup Time (Intel486 SX-33 microprocessor)	5		ns
t_{ELCH}	CE_X# Setup Time to CLK (28F016XS-15)	25		ns
t_{VLCH}	ADV# Setup Time to CLK (28F016XS-15)	15		ns
t_{GLCH}	OE# Setup Time to CLK (28F016XS-15)	15		ns
t_{CHQV}	CLK to Data Delay (28F016XS-15)		20	ns
t_{PZX}	CLK Output Delay (22V10-15)	2	8	ns

NOTE:
Consult appropriate datasheets for up-to-date specifications.

Optimized Configuration

Refer to Figures 10 and 12 for the following discussion.

With the 28F016XS-15s in the optimized configuration (SFI Configuration = 2 at 33 MHz), and CFG set to a logic "1" value, the system interface executes cacheable burst cycles.

The Intel486 SX processor drives ADS# low, notifying the interface logic that a valid address is on the address bus. Monitoring the external bus of the Intel486 SX microprocessor, the state machine then transitions into read control.

If ADS# = 0 and W/R# = 0 then READ CONTROL

In optimized read control, the state machine controls OE#, KEN# and BRDY#. If CE# = "0," the state machine at N = 1 loads the two-bit counter (A_{3-2}) and activates ADV# (ADV# = "0") for the next four consecutive clock periods (N = 1 through 5). While ADV# is driven low, the counter increments through the Intel burst order (Table 2), supplying the 28F016XS-15s with a new address at N = 2, 3, 4 and 5. If CE# ="1," the state machine returns to an idle state waiting for a new memory access.

With the SFI Configuration value set to 2, new data will be available at N = 5, 6, 7 and 8. Driving BRDY# low at N = 4, the Intel486 SX microprocessor will sample BRDY# active at N = 5, which informs the processor of valid information on data pins D_{31-0} and that the 28F016XS memory space supports a burst read transfer. BRDY# is held low until the end of the burst cycle while the processor retrieves data on every rising clock edge. BRDY# is driven high upon sampling BLAST# low, marking the end of the burst cycle.

The Intel486 SX microprocessor requires a 3 ns hold time after sampling the last BRDY# active in a burst cycle. Therefore, the state machine will hold OE# active for 15 ns after the microprocessor samples the last double-word. At N = 9, the state machine transitions to an idle state, where it deactivates OE# and waits for the Intel486 SX microprocessor to initiate a new bus cycle targeting the 28F016XS memory space.

Optimized Configuration Timing Considerations

In the optimized configuration, CE# setup time is again a key system timing parameter. For information regarding the CE# setup time requirement, see the Initial Configuration Timing Considerations Timing section.

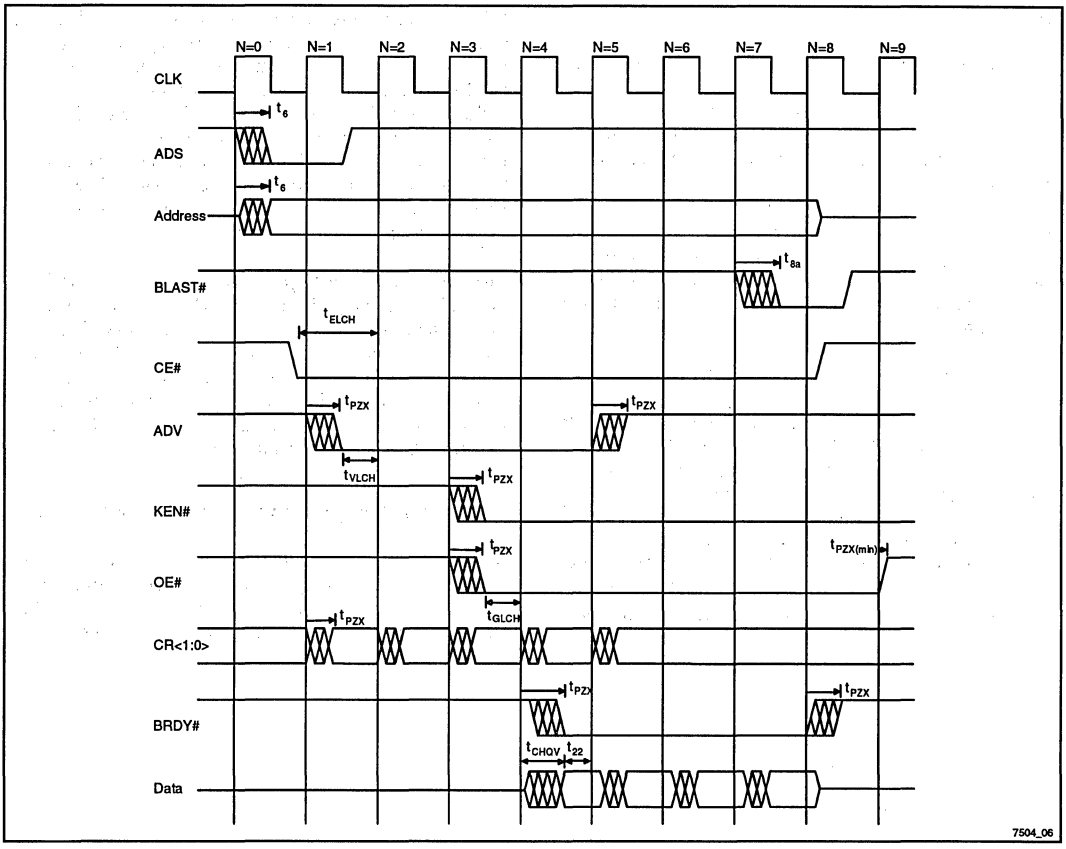


Figure 12. Example Burst Read Cycle Showing Key Timing Specifications Requiring Consideration

Table 6. Example Optimized Read Cycle Specifications at 5V V_{CC}

Symbol	Description	Min	Max	Unit
t_6	ADS# Delay (Intel486 SX-33 microprocessor)	3	16	ns
t_{8a}	BLAST# Delay (Intel486 SX-33 microprocessor)	3	20	ns
t_{22}	D_{31-0} Setup Time (Intel486 SX-33 microprocessor)	5		ns
t_{ELCH}	CE_X# Setup Time to CLK (28F016XS-15)	25		ns
t_{VLCH}	ADV# Setup Time to CLK (28F016XS-15)	15		ns
t_{GLCH}	OE# Setup Time to CLK (28F016XS-15)	15		ns
t_{CHQV}	CLK to Data Delay (28F016XS-15)		20	ns
t_{PZX}	CLK Output Delay (22V10-15)	2	8	ns

NOTE:
Consult appropriate datasheets for up-to-date specifications.

3.3 Write Cycle Control

The write interface in for this design functionally behaves like the optimized design's write interface. The only difference between the two designs is the absence of the MUX in the standard interface. Therefore, the interface logic for this design does not have to concern itself with changing the data flow through the MUX. Instead, the interface simply loads and holds the address for the duration of the write operation.

For further detailed information about this cycle and write timing waveform, refer to Section 2.4.

4.0 INTERFACING TO OTHER INTEL486 MICROPROCESSORS

The Intel486 microprocessor family provides designers a large and diverse selection of CPUs, which offers designers different performance points to meet different market segment needs. Throughout the product family, the external bus architecture has remained consistent, which makes the 28F016XS interface to the entire Intel486 microprocessor family similar, if not identical, to the Intel486 SX microprocessor interface described in Sections 2 and 3. The 28F016XS-15 interface to the Intel486 SX-33 microprocessor works equally well for the following microprocessors at 5.0V V_{CC}.

- Intel486™ SX-20, 25 processors
- Intel486™ SX2-50 processor
- Intel486™ DX-25, 33 processors
- Intel486™ DX2-40, 50, 66 processors
- IntelDX4™-75, 100 processors(I/O buffers configured for 5.0V, V_{CC5} = 5.0V)

The 28F016XS-15 interface to the Intel486 SX-33 microprocessor also works well for the following Intel486 microprocessors at 3.3V V_{CC}. The 3.3V V_{CC} design utilizes a 22V10-15 low voltage PLD to control the interface between the 28F016XS-15 (operating at 3.3V V_{CC}) and the processor.

- Intel486 SX-20, 25 processors
- Intel486 DX-25 processor
- Intel486 DX2-40, 50 processors
- IntelDX4-75 processors (I/O buffers configured for 3.3V, V_{CC5} = 3.3V)

When the external bus frequency falls outside the 16.7 MHz through 33 MHz frequency range at 5.0V V_{CC} (12.5 MHz through 25 MHz at 3.3V V_{CC}), the optimized SFI Configuration value for the 28F016XS-15 differs in respect to the Intel486 SX-33 microprocessor design documented earlier. The state machine, therefore, requires slight modifications to accommodate the different SFI Configuration. Note, the initial read configuration and write control state machine remains consistent throughout all designs because they are not affected by the optimized SFI Configuration.

Intel486 SX-16 Microprocessor Interface at 5.0V V_{CC}

The 28F016XS-15's optimized SFI Configuration at 16 MHz equals 1. Therefore, 28F016XS-15 will begin driving data one CLK period after initiating the first read access. The optimized state machine must drive BRDY# and OE# active upon initiating the first access to the 28F016XS-15. BRDY# and OE# remain low throughout the burst cycle. The optimized and standard 28F016XS-15 interface to the Intel486 SX-16 microprocessor at this specific frequency deliver 1-0-0-0 and 2-0-0-0 wait-state read performance respectively.

Intel486 DX-50 Microprocessor Interface at 5V V_{CC}

Operating at 50 MHz, the 28F016XS-15's optimized SFI Configuration equals 3. The interface loads the two-bit counter and drives ADV# active at the first rising CLK edge after the processor initiates the read access. The optimized read state machine increments the two-bit counter and drives ADV# low every other CLK, thereby adhering to the *Alternating-A_1* and *Same-A_1* access rules (see Additional Information). The optimized and standard 28F016XS-15 interface to the Intel486 DX-50 microprocessor at this given frequency deliver 4-1-1-1 and 5-1-1-1 wait-state read performance respectively.

Intel486 DX-33 and IntelDX4-100 Microprocessor Interface at 3.3V V_{CC}

Operating at 33 MHz with 3.3V V_{CC}, the 28F016XS-15's optimized SFI Configuration equals 3. The interface loads the two-bit counter and drives ADV# active at the first rising CLK edge after the processor initiates the read access. The optimized read state machine increments the two-bit counter and drives ADV# low for two CLK periods and then strobes ADV# high for one CLK period. ADV# is again driven low for two CLK periods finishing the burst cycle. Refer to the *Alternating-A_1* and *Same-A_1* access rules (see Additional Information) for further information on consecutive accesses. The optimized and standard 28F016XS-15 interface to the Intel486 microprocessor

at this frequency deliver 3-0-1-0 and 4-0-1-0 wait-state read performance respectively.

5.0 CONCLUSION

This technical paper has described the interface between the 28F016XS 16-Mbit flash memory component and the Intel486 microprocessor. This simple design has been implemented with a minimal number of components and achieves exceptional read performance. The 28F016XS provides the microprocessor with the non-volatility and updateability of flash memory and the performance of DRAM. For further information about 28F016XS-15, consult reference documentation for a more comprehensive understanding of device capabilities and design techniques. Please contact your local Intel or distribution sales office for more information on Intel's flash memory products.

ADDITIONAL INFORMATION

Order Number	Document/Tools
290532	28F016XS Datasheet
297500	"Interfacing the 28F016XS to the i960® Microprocessor Family"
292147	AP-398, "Designing with the 28F016XS"
292146	AP-600, "Performance Benefits and Power/Energy Savings of 28F016XS Based System Designs"
292163	AP-610, "Flash Memory In-System Code and Data Update Techniques"
292165	AB-62, "Compiled Code Optimizations For Embedded Flash RAM Memories"
297372	16-Mbit Flash Product User's Manual,
297508	FLASHBuilder Utility
Contact Intel/Distribution Sales Office	28F016XS Benchmark Utility
Contact Intel/Distribution Sales Office	28F016XS iBIS Models
Contact Intel/Distribution Sales Office	28F016XS VHDL/Verilog Models
Contact Intel/Distribution Sales Office	28F016XS Timing Designer Library Files
Contact Intel/Distribution Sales Office	28F016XS Orcad and ViewLogic Schematic Symbols

REVISION HISTORY

Number	Description
001	Original Version
002	Incorporated initial read burst configuration, replacing the single read cycle Added optimized design, improving system read performance

APPENDIX A
PLD FILE FOR THE 28F016XS INTEL486 MICROPROCESSOR INTERFACE

PLD file for the optimized interface described in Section 2.

```
TITLE                   Optimized 28F016XS / 486 Interface
PATTERN                 PDS
REVISION                1
AUTHOR                  Example
COMPANY NAME            Intel
DATE                    2/6/95

CHIP  OPTIMIZED_28F016XS_486_INTERFACE 85C22V10

; input pins
PIN   1    CLK          ; clk frequency 33mhz
PIN        ADS          ; address strobe from 486
PIN        WR           ; multiplexed read/write strobe
PIN        BLAST        ; BLAST from the 486
PIN        CE           ; CE from the address decoding logic
PIN        CFG          ; informs interface to changes to the SFI Configuration
PIN        A2           ; lower address lines from the 486 used
PIN        A3           ; in loading the counter
PIN        RESET        ; system reset
PIN   25   GLOBAL
; output pins
PIN        ADV          ; address valid input
PIN        /KEN         ; cache control
PIN        /OE          ; output enable input
PIN        /WE          ; write enable input
PIN        /BRDY        ; initiating a burst cycle, 486 input
PIN        SWITCH       ; control MUX switching
PIN        Q0           ; state variable
PIN        CTR0         ; lower bit of the 2bit counter
PIN        CTR1         ; higher bit of the 2bit counter

STRING  LD '(/ADS)'     ; load
STRING  INC '(/ADV)'    ; increment

STATE MOORE_MACHINE
DEFAULT_BRANCH S0

; state assignments
S0 = /ADV * /BRDY * /OE * /WE * /KEN * /SWITCH * /Q0
S1 = /ADV * /BRDY * /OE * /WE * /KEN *  SWITCH * /Q0
S2 = /ADV * /BRDY *  OE * /WE *  KEN *  SWITCH * /Q0
S3 = /ADV *  BRDY *  OE * /WE *  KEN *  SWITCH * /Q0
S4 =  ADV *  BRDY *  OE * /WE *  KEN *  SWITCH * /Q0
S5 =  ADV * /BRDY *  OE * /WE *  KEN *  SWITCH * /Q0
```

```
S6 = ADV * /BRDY * /OE * /WE * /KEN *  SWITCH *  Q0
S7 = ADV * /BRDY * /OE * /WE *  KEN *  SWITCH *  Q0
S8 = ADV *  BRDY *  OE * /WE *  KEN *  SWITCH *  Q0
S9 =/ADV * /BRDY *  OE * /WE *  KEN *  SWITCH *  Q0
S10 = ADV * /BRDY * /OE * /WE * /KEN *  SWITCH * /Q0
S11 = ADV * /BRDY * /OE *  WE * /KEN *  SWITCH * /Q0
S12 = ADV * /BRDY * /OE *  WE * /KEN *  SWITCH *  Q0
S13 = ADV *  BRDY * /OE * /WE * /KEN *  SWITCH * /Q0

; state transitions
S0 :=  (/ADS * /WR * /CFG)          -> S6  ; initial config READ cycle
  +  (/ADS * /WR *  CFG)            -> S1  ; optimized config READ cycle
  +  (/ADS *  WR)                   -> S10
                                   +-> S0
S1 :=  (/CFG * /CE *  BOFF)         -> S7
  +  ( CFG * /CE *  BOFF)           -> S2
  +   CE * /BOFF                    -> S0
S2 :=  /CFG * BOFF                  -> S8
  +   CFG * BOFF                    -> S3
                                   +-> S0
S3 :=  /BLAST + /BOFF               -> S0
  +   BLAST                         -> S4
S4 :=  /BLAST + /BOFF               -> S0
  +  ( BLAST * /CFG)                -> S5
  +  ( BLAST *  CFG)                -> S4
S5 :=   BOFF                        -> S4
  +  /BOFF                          -> S0
S6 :=   BOFF                        -> S1
  +  /BOFF                          -> S0
S7 :=   BOFF                        -> S2
  +  /BOFF                          -> S0
S8 :=  /BLAST + /BOFF               -> S0
  +   BLAST                         -> S9
S9 :=   BOFF                        -> S4
  +  /BOFF                          -> S0
S10:=  /CE                          -> S11  ; write cycle
  +   CE + /BOFF                    -> S0
S11:=   BOFF                        -> S12  ; WE low for two clocks
  +  /BOFF                          -> S0
S12:=   BOFF                        -> S13
  +  /BOFF                          -> S0
S13:=   VCC                         -> S0  ; ready

EQUATIONS
 ; implement RESET
  GLOBAL.RSTF = /RESET
 ; implement 2-bit burst counter
  CTR1 := (LD * A3) + (/LD * INC * /CTR1 * S1)
      + (/LD * INC * CTR1 * /S1) + (/LD * /INC * CTR1 * /S1)
  CTR0 := (/WR * LD * /A2) + (WR * LD * A2)
      + (/LD * INC * /CTR0) + (/LD * /INC * CTR0)
```

PLD file for the standard interface described in Section 3.

```
Title           Standard 28F016XS / Intel486™ Interface
Pattern         PDS
Revision        1
Author          Example
Company Name    Intel
Date            2/14/94

CHIP   28F016XS_486_Interface   85C22V10   ; 85C22V10-15

; input pins
PIN    1     CLK          ; clk frequency 33mhz
PIN          ADS          ; address strobe from 486
PIN          WR           ; multiplexed read/write strobe
PIN          BLAST        ; BLAST from the 486
PIN          CE           ; CE from the address decoding logic
PIN          CFG          ; informs interface to changes to the SFI Configuration
PIN          BOFF         ; BOFF input to processor
PIN          A2           ; lower address lines from the 486 used
PIN          A3           ; in loading the counter
PIN          RESET        ; system reset
PIN    25    GLOBAL
;output pins
PIN          /ADV         ; address valid input
PIN          /KEN         ; cache control
PIN          /OE          ; output enable input
PIN          /WE          ; write enable input
PIN          /BRDY        ; initiating a burst cycle, 486 input
PIN          Q0           ; state variable
PIN          Q1           ; state variable
PIN          CONT0        ; lower bit of the 2bit counter
PIN          CONT1        ; higher bit of the 2bit counter

STRING  LD '(/ADS)'       ; load
STRING  INC '(/ADV)'

STATE MOORE_MACHINE
DEFAULT_BRANCH  S0
; state assignments
S0  = /ADV * /KEN * /WE * /BRDY * /Q0 * /Q1
S1  =  ADV * /KEN * /WE * /BRDY * /Q0 * /Q1
S2  =  ADV *  KEN * /WE * /BRDY * /Q0 * /Q1
S3  =  ADV *  KEN * /WE * /BRDY * /Q0 *  Q1
S4  =  ADV *  KEN * /WE *  BRDY * /Q0 * /Q1
S5  = /ADV *  KEN * /WE *  BRDY * /Q0 * /Q1
S6  = /ADV * /KEN * /WE * /BRDY * /Q0 *  Q1
S7  = /ADV *  KEN * /WE * /BRDY * /Q0 * /Q1
S8  = /ADV *  KEN * /WE * /BRDY * /Q0 *  Q1
S9  =  ADV *  KEN * /WE *  BRDY * /Q0 *  Q1
S10 = /ADV *  KEN * /WE * /BRDY *  Q0 *  Q1
S11 = /ADV * /KEN * /WE * /BRDY *  Q0 * /Q1
```

```
S12 = /ADV * /KEN *  WE * /BRDY * /Q0 * /Q1
S13 = /ADV * /KEN *  WE * /BRDY * /Q0 *  Q1
S14 = /ADV * /KEN * /WE *  BRDY *  Q0 * /Q1

; state transitions
S0  :=  ADS * /BOFF          -> S0      ; start of an access
    + /ADS * /WR * /BOFF     -> S1
    + /ADS *  WR * /BOFF     -> S11
S1  := /CFG * /CE            -> S6      ; not reconfigured
    +  CFG * /CE             -> S2      ; reconfig active low
    +  CE                    -> S0
S2  := /CFG *  BOFF          -> S7      ; if chip enable is de-asserted,
    +  CFG *  BOFF           -> S3      ; quit the access and return
    + /BOFF                  -> S0
S3  := /CFG *  BOFF          -> S8
    +  CFG *  BOFF           -> S4
    + /BOFF                  -> S0
S4  := /BLAST + /BOFF        -> S0      ; situations will only ocur during
    +  BLAST                 -> S5      ; a BOFF.
S5  := /BLAST + /BOFF        -> S0      ; continous cycling until BLAST is
    + /CFG * BLAST           -> S10     ; presented - end the burst cycle
    +  CFG * BLAST           -> S5
S6  := /BOFF                 -> S0
    +  BOFF                  -> S2
S7  := /BOFF                 -> S0
    +  BOFF                  -> S3
S8  := /BOFF + /BLAST        -> S0
    +  BOFF                  -> S9
S9  := /BOFF                 -> S0
    +  BOFF                  -> S5
S10 := /BOFF                 -> S0
    +  BOFF                  -> S5
S11 := /CE                   -> S12     ; situation will only occur during
    +  CE + /BOFF            -> S0      ; during a BOFF.
S12 := /BOFF                 -> S0
    +  BOFF                  -> S13
S13 := /BOFF                 -> S0
    +  BOFF                  -> S14
S14 :=  VCC                  -> S0

EQUATIONS
 ; implement RESET
   GLOBAL.RSTF = /RESET
 ; implement 2-bit burst counter
   CTR1 := (LD * A3) + (/LD * INC * /CTR1 * S1)
       + (/LD * INC * CTR1 * /S2) + (/LD * /INC * CTR1 * /S2)
    CTR0 := (/WR * LD * A2) + (/LD * INC * /CTR0) + (/LD * /INC * CTR0)
 ; output enable control, triggered on falling clock edge
   OE := S2 + S3 + S4 + S5 + S7 + S8 + S9 + S10
   OE.CLKF = /CLK
```

intel®

TECHNICAL PAPER

Interfacing the 28F016XS to the i960® Microprocessor Family

KEN MC KEE
TECHNICAL MARKETING ENGINEER

TIM KELLY
ENGINEER

RANNA PRAJAPATI
ENGINEER

February 1995

Order Number: 297500-002

1.0 INTRODUCTION

This technical paper describes several designs interfacing the high-performance 28F016XS Flash memory to the i960® microprocessor family. All designs are based on preliminary 28F016XS specifications. Please contact your Intel or distribution sales office for up-to-date specifications before finalizing any design.

The 28F016XS is a 16-Mbit block erasable flash memory with a high-performance synchronous pipelined read interface. This optimized interface can sustain a high read transfer rate and makes the 28F016XS the ideal flash memory component when interfacing to a burst processor, such as the i960 microprocessor. The 28F016XS combines ROM-like non-volatility, DRAM-like read performance and in-system update ability in one memory technology. These characteristics enable code execution directly from the 28F016XS memory space, replacing the costly practice of shadowing code from HDD or ROM to DRAM. The i960 microprocessor family sees widespread use in various applications, including imaging and data communications. Combined, the 28F016XS and the i960 processor constitute a rugged, high performance and cost-effective solution.

The 28F016XS performs synchronous pipelined reads. Up to three accesses can be initiated before reading data output from the initial cycle. This pipelined structure is ideal for use with the i960 microprocessor's burst transfer mechanism. The 28F016XS brings significant system performance enhancements to an i960 microprocessor-based environment. This technical paper describes processor-to-memory interfaces that exploit these capabilities to achieve maximum system performance. Figures 1 and 2 illustrate relative system performance enhancements that the 28F016XS brings to an i960 microprocessor-based environment, compared to other technologies. The benchmark parameters are documented in Appendix B.

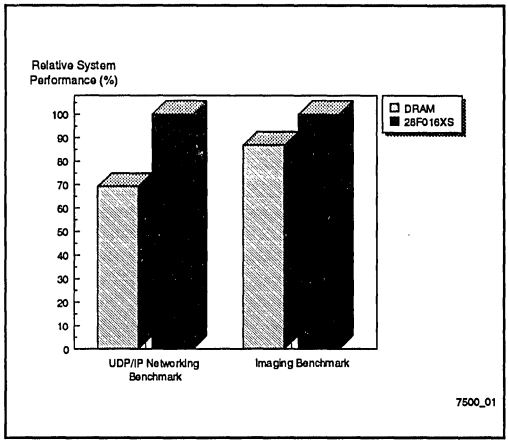


Figure 1. Relative System Performance Enhancement of the 28F016XS Compared to Other Memory Technologies In an i960 KB-25 Microprocessor-Based Design

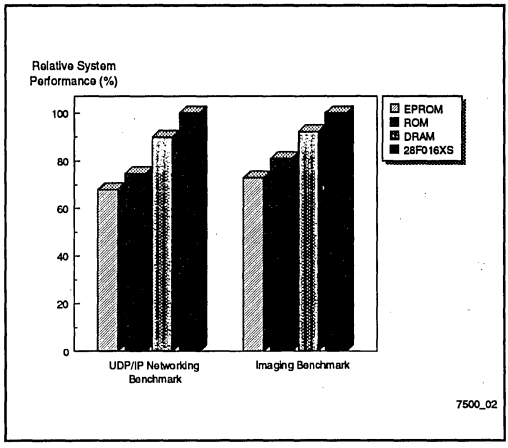


Figure 2. Relative System Performance Enhancement of the 28F016XS Compared to Other Memory Technologies in an i960 CA-33 Processor-Based Design

2.0 i960 CA-33 MICROPROCESSOR INTERFACE

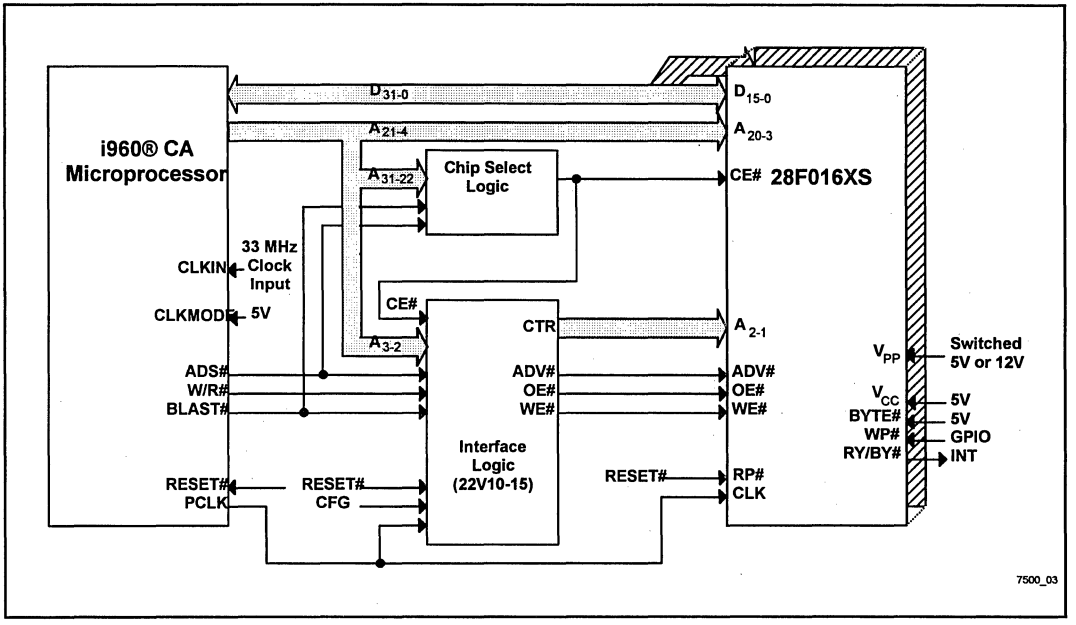


Figure 3. Minimal Logic Required in Interfacing the 28F016XS-15 to the i960 CA-33 Microprocessor to Sustain 3-0-0-0-2-0-0-0.. Burst Pipelined Read Performance Up to 33 MHz

Using this interface, an i960 CA-33 microprocessor based system executing code directly out of the 28F016XS can achieve 3-0-0-0-2-0-0-0. . .wait-state read performance. This interface supports both burst transfers and address pipelining. For schematic and PLD files contact your Intel or distrution sales office.

2.1 Circuit Description

This section describes the 28F016XS-15 interface to the i960 CA-33 microprocessor interface block diagram in Figure 3.

Memory Configuration

This design uses two 28F016XS-15s, in x16 mode, to match the i960 CA microprocessor's 32-bit data bus, providing 4 Mbytes of flash memory. Signals A_{21-4} from the i960 CA microprocessor and CTR_{1-0} from the PLD select locations within the 28F016XS memory space, arranged as 1-Meg double words. The two-bit counter implemented in the PLD loads from the processor's lower addresses at the beginning of each memory cycle and generates the lower two bits of the burst addresses on its outputs CTR_{1-0}. The counter feeds burst addresses so that the 28F016XS-15s do not stall waiting for the processor to supply the next address.

Chip Select Logic

Chip select decode logic may use A_{31-22} to generate an active low chip select signal, CE#, for the 28F016XS-15 memory space and other system peripherals. The chip select drives CE_0# on each 28F016XS-15 and a control input to the PLD. The 28F016XS-15's CE_1# input is grounded.

In support of address pipelining, the chip select logic latches CE#, holding it active throughout the duration of the memory access. This will prevent potential CE# problems caused by using combinatorial logic when utilizing the address pipelining capability of i960 CA microprocessor.

If address pipelining functionality is not implemented, simple combinatorial logic can be utilized in generating the system CE# for the 28F016XS memory space, and

the chip select logic shown in Figure 3 does not examine BLAST# and ADS#. For many systems using the upper address bits in a linear selection scheme may provide a sufficient number of chip select signals, thus eliminating system chip select decode logic. (See Figure 4 for an example of using linear selection for chip selects.) When using a linear chip select scheme however, the software must avoid using addresses that may select more than one device, which could result in bus contention.

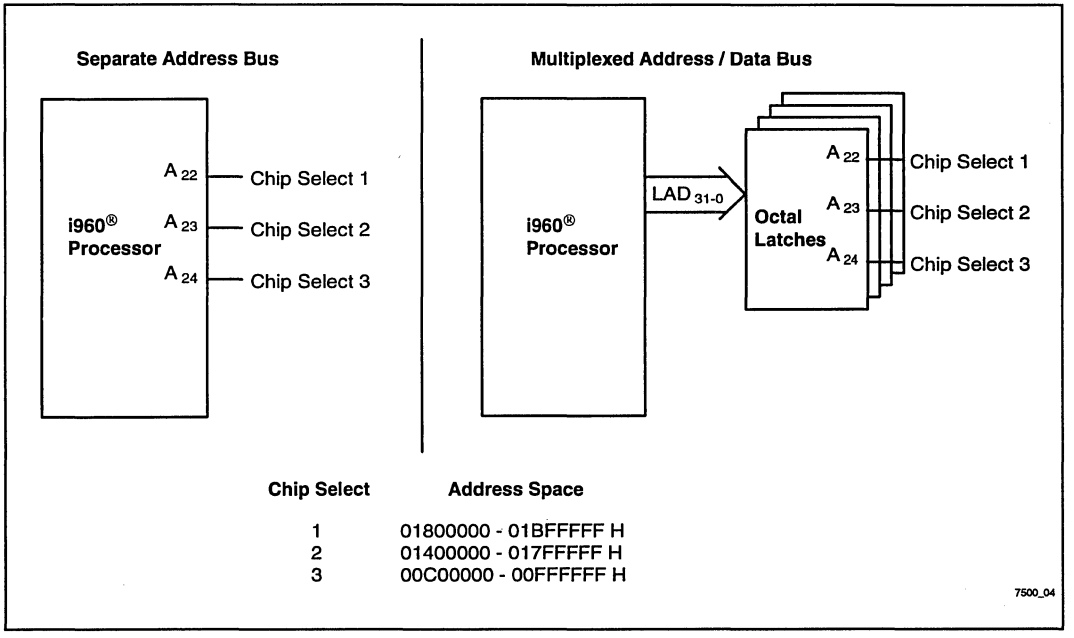


Figure 4. Example of Using Linear Chip Selection

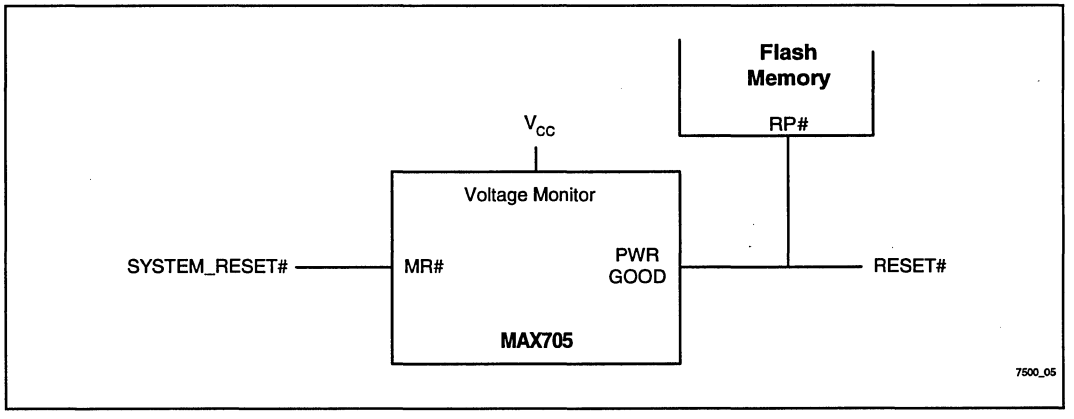


Figure 5. Example RESET Generation Circuitry

CLK Option

A 33 MHz clock signal drives the i960 CA microprocessor CLKIN input. Driving CLKMODE to a logic "1" configures the i960 CA microprocessor for a 1x CLK input. The i960 CA microprocessor outputs an internally-referenced 33 MHz clock on its PCLK1 and PCLK2 pins (the signals on PCLK1 and PCLK2 are identical), which drives the CLK inputs of the PLD and the 28F016XS-15s.

Reset

An active-low reset signal, RESET#, connects to the RESET# inputs of the i960 CA microprocessor, and the PLD, and to the RP# input of the 28F016XS-15s. Figure 5 illustrates a suggested logic configuration for generating RESET#.

Interface Control Signals

The i960 CA-33 microprocessor external bus signals, BLAST#, ADS# and W/R#, serve as inputs to the state machine, which controls the two-bit counter and generates OE#, WE# and ADV#. The counter is loaded at the beginning of the memory access, generating the burst addresses to the 28F016XS-15s. ADV# indicates that a valid address is available to the 28F016XS-15. Addresses are latched and a read cycle is initiated on a rising CLK edge. WE# controls writes to the 28F016XS-15, latching data into the 28F016XS-15 on its rising edge if the applicable timing requirements are satisfied.

Configuration Signal

A general purpose input/output (GPIO) generates the configuration signal input to the state machine. The configuration signal must reset to logic "0" on power-up and system reset to ensure that the operation of the state machine matches the initial configurations of the 28F016XS-15s and the i960 CA microprocessor. After optimizing the 28F016XS-15s and i960 CA microprocessor, the configuration signal must switch to logic "1."

Additional 28F016XS Control Signals

The BYTE# input to the 28F016XS-15s is tied to 5.0V to configure the 28F016XS-15s for x16 mode, and A_0 is tied to GND (A_0 is only used for byte addressing). A GPIO controls the write protect input, WP#, to the 28F016XS-15s. As shown in Figure 3, the 28F016XS-15 is compatible with either a 5.0V or a 12.0V V_{PP} voltage and is completely write protected by switching V_{PP} to GND. When V_{PP} drops below V_{PPLK}, the 28F016XS-15 will not successfully complete Program and Erase operations. Figure 3 also illustrates the 28F016XS-15 RY/BY# output connected to a system interrupt for background erase operation. RY/BY#, WP#, and V_{PP} implementation are application dependent. See the Additional Information section of this technical paper for documentation that cover these topics in more detail.

2.2 Software Interface Considerations

Boot-up Capability / Configuration

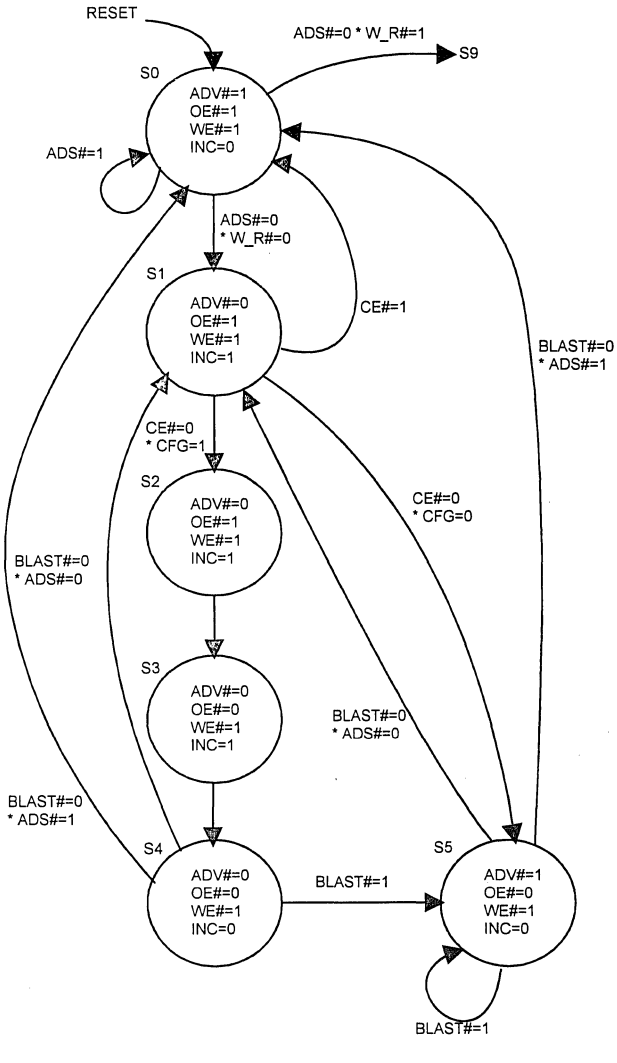
This interface supports processor boot-up from the 28F016XS memory space upon power-up or system reset. However, the boot code must follow some restrictions until it has properly configured the 28F016XS-15s, i960 CA microprocessor and CFG state machine input valve. In the default configuration state, the i960 CA microprocessor supports only non-burst reads and writes. Program control should jump to an area of RAM to execute the configuration sequence. The code will configure the 28F016XS-15s and all necessary i960 CA microprocessor programmable attributes before setting the CFG input to logic "1." Table 1 illustrates the required configuration settings for both the 28F016XS-15s and the i960 CA-33 microprocessor.

Table 1. Configuration Settings for the 28F016XS-15 and i960 CA-33 Microprocessor Employing Address Pipelining at 33 MHz

Part	Parameter	Setting
28F016XS-15 (5V V_{CC})	SFI Configuration	2
i960 CA-33 Microprocessor	Ready Inputs	OFF
	Byte Ordering	LITTLE ENDIAN
	Bus Width	32-BIT
	Wait-States: Nrad	3
	Nrdd	0
	Nwad	2
	Nwdd	2
	Nxda	0
	Address Pipelining	ON
	Burst mode	ON

2.3 Single and Burst Read Cycle Description

Refer to the read cycle timing diagrams (Figures 7 and 8) and the state diagram (Figure 6) for the following read cycle discussion.



NOTE:
OE# and WE# are clocked on the inverted CLK edge

Figure 6. Read State Diagram of Single and Address Pipelined Burst Control Interface Shown in Figure 3

Initial Configuration

Figure 7 illustrates a read cycle with the 28F016XS-15s and i960 CA microprocessor in a power-up/reset configuration state. The initial configuration permits only non-burst transfers. The i960 CA microprocessor initiates a read cycle by asserting ADS# with W/R# = "0," presenting the valid address and control signals. At N = 1, the two-bit counter loads the values on the processor's lower address lines, A_{3-2}. The state machine asserts ADV# for the next clock (N = 2), where the 28F016XS-15 will latch in the address if CE# is asserted. If CE# is not asserted, the state machine returns to inactive state at N = 2.

If the flash memory is selected, the state machine will assert ADV# for only one clock before entering a hold state to await the assertion of BLAST# by the i960 CA microprocessor. The state machine asserts OE# (to meet timing requirements OE# is falling-edge triggered) on the falling edge between N = 2 and N = 3 to enable the 28F016XS-15 data output buffers. With SFI Configuration = 4, the data will be valid at the N = 7. The 28F016XS-15s will hold data on the bus until the i960 CA microprocessor asserts BLAST#. During the clock period following N = y, the state machine returns to its inactive state, de-asserting OE# to tri-state the 28F016XS-15 data outputs.

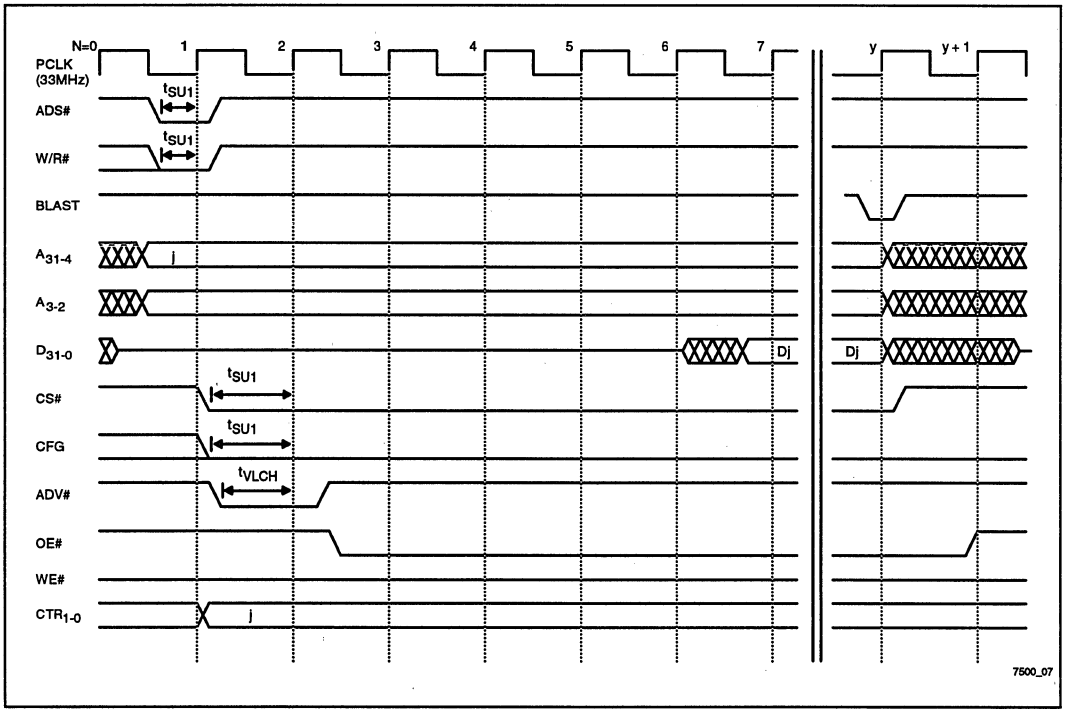


Figure 7. Example Read Cycle, Initial Configuration, Showing Key Specifications Requiring Consideration

Optimized Configuration

Figure 8 illustrates a two double-word burst read followed by a four double-word burst read with the 28F016XS-15s, i960 CA microprocessor and state machine configured for optimum read performance. With CFG = 1, the counter increments the two lower bits of the address at N = 2, N = 3 and N = 4, and ADV# remains asserted so that the 28F016XS-15 latches in four successive addresses at N = 2–5. With SFI Configuration = 2, the first data will be available at N = 5 for the processor to read. If a second read cycle follows the current read cycle, the i960 CA microprocessor will assert ADS# one clock after asserting BLAST#. The state machine will respond by immediately re-entering the read cycle. Otherwise, the state machine will return to its inactive state waiting for a new access targeting the 28F016XS memory space.

When implementing the i960 CA microprocessor address pipelining capability, the state machine controlling CE# monitors the upper address lines, ADS# and BLAST#. CE# is held active upon detecting an access targeting the flash memory space until BLAST# is asserted with ADS# de-asserted. When BLAST# and ADS# are active at the same time, a pipelined read access is in progress. If the current pipelined access is aimed at the 28F016XS memory space, the CE# state machine will hold CE# active until BLAST# is sampled active again.

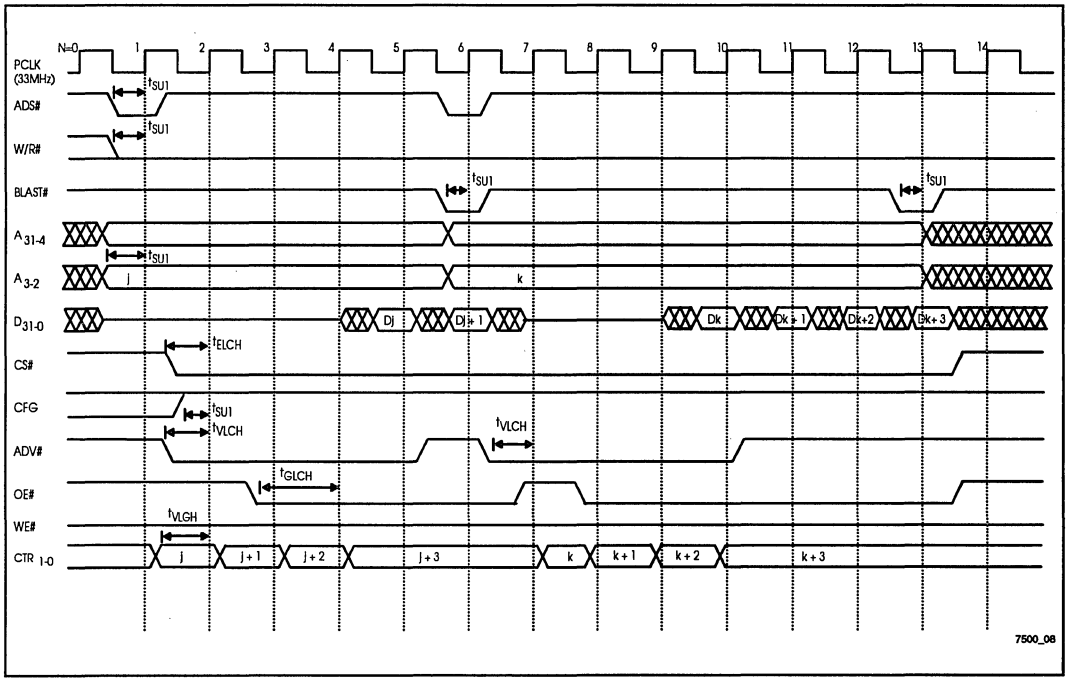


Figure 8. Example Two Double-Word Burst Read Followed by Pipelined Four Double Word Burst Read Showing Key Specifications Requiring Consideration

Critical Timings

Table 2 describes the critical timings illustrated in Figures 7 and 8. One particularly critical timing in this designs, is the data hold time. The i960 CA-33 microprocessor requires a 5 ns hold time after the clock edge. The 28F016XS-15 guarantees a 5 ns data hold after clock, meeting the processor's hold requirement with 0 ns of margin.

This design provides 7 ns of margin to meeting the 3 ns setup time of the i960 CA-33 microprocessor data inputs, outputting data t_{CHQV} after a rising CLK edge.

Another critical area concerns CE# during pipelined read accesses. Since the 28F016XS-15 specifies zero data hold from CE# going high, the chip select state machine must hold CE# active for 5 ns to satisfy the i960 CA-33 microprocessor data input hold specification of 5 ns. Hence, the chip select state machine holds CE# active for an additional clock period after detecting BLAST# active.

The i960 CA-33 microprocessor control outputs ADS# and W/R# have 3 ns of margin and BLAST# has 5 ns of margin to meeting the 22V10-15 input setup requirement.

Consult the appropriate datasheets for full timing information.

Table 2. Example Read Cycle Timing Specifications at 5V V_{CC}

Part	Symbol	Parameter	Minimum Specified Value (ns)
22V10-15	t_{SU1}	Input Setup Time to CLK	9
i960 CA-33 Microprocessor	T_{IS1}	Input Setup D_{31-0}	3
	T_{IH1}	Input Hold D_{31-0}	5
28F016XS-15	t_{ELCH}	CE# Setup to CLK	25
	t_{VLCH}	ADV# Setup to CLK	15
	t_{AVCH}	Address Setup to CLK	15
	t_{GLCH}	OE# Setup to CLK	15

NOTE:
Consult appropriate datasheets for up-to-date specifications.

2.4 Single Burst Write Cycle Description

Refer to the write cycle timing diagrams and the state diagram (Figure 9) for the following write cycle discussion.

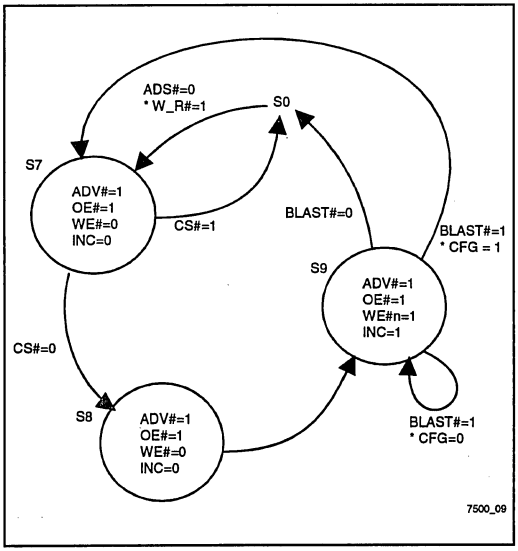


Figure 9. Write State Diagram of Control Interface Shown in Figure 3

Initial Configuration

Figure 10 illustrates a write cycle. In the reset/power-up configuration state, the interface supports only non-burst writes. The i960 CA microprocessor initiates a write cycle by asserting ADS# with W/R# = "1," presenting a valid address and control signals. At N = 1, the two-bit counter loads the values on address bits A_{3-2}. The state machine asserts WE# (to meet timing requirements, WE# is falling-edge triggered) on the falling edge between N = 1 and N = 2. WE# remains asserted for two clock periods, in order to meet the 28F016XS-15 timing requirements. The state machine then enters a holding state until the processor asserts BLAST#, after which time the interface state machine will return to S0.

Optimized Configuration

Figure 11 illustrates a two double-word burst write with CFG = "1." When the first data write is complete at N = 4, the counter increments the two lower address bits, and the state machine asserts WE# on the next falling clock edge to begin the next the 28F016XS-15 data write. The i960 CA microprocessor must provide the next data during the clock period following N = 4. The data writes continue to the next consecutive addresses until the i960 CA microprocessor asserts BLAST#, indicating the end of the burst write cycle.

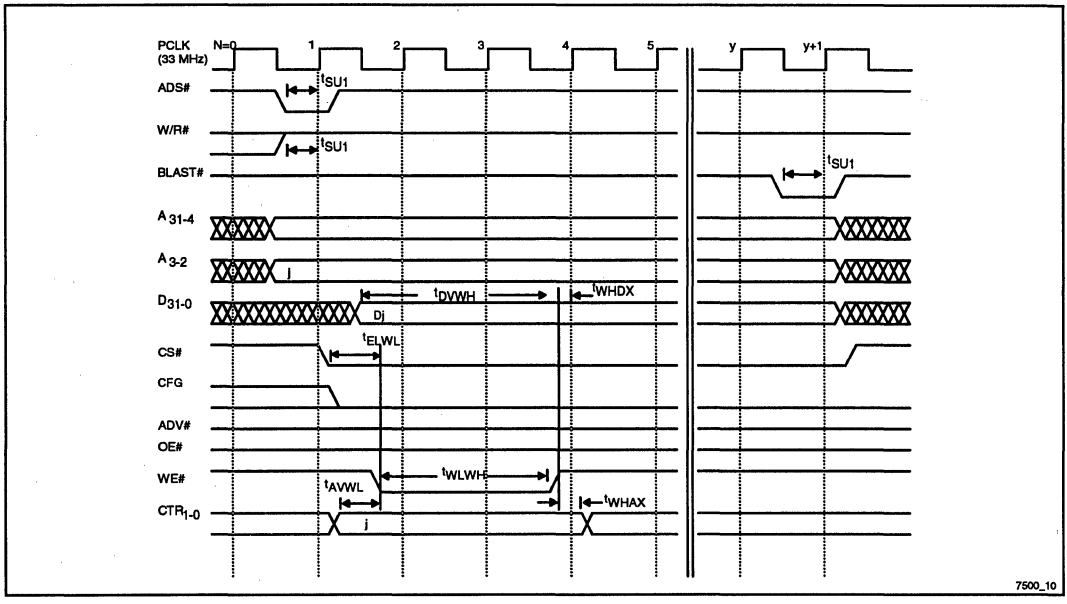


Figure 10. Example Write Cycle, Initial Configuration, Showing Key Specifications Requiring Consideration

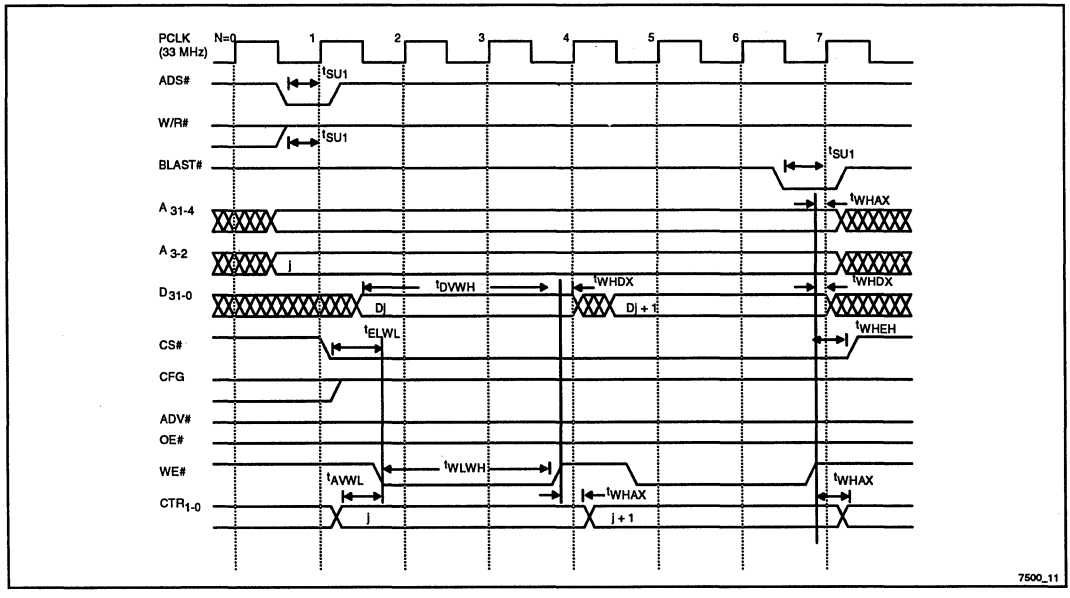


Figure 11. Example Two Double-Word Burst Write Illustrating Key Specifications Requiring Considerations

Critical Timings

Table 3 describes the critical timings illustrated in Figures 10 and 11.

One critical hold time to notice is t_{WHAX}. WE# is guaranteed to transition within 8 ns from the falling clock edge. Therefore, the t_{WHAX} requirement has 2 ns of margin on CTR_{1-0}, and 5 ns of margin on A_{31-4}.

Also notice that CTR_{1-0} must be valid before WE# is asserted. CTR_{1-0} are guaranteed valid 8 ns after the rising clock edge, providing 9 ns of margin.

Consult the appropriate datasheets for full timing information.

Table 3. Example Write Cycle Timing Parameters at 5V V_{CC}

Part	Symbol	Parameter	Minimum Specified Value (ns)
22V10-15	t_{SU1}	Input Setup Time to CLK	9
28F016XS-15	t_{ELWL}	CE# Setup to WE# Going Low	0
	t_{AVWL}	Address Setup to WE# Going Low	0
	t_{WLWH}	WE# Pulse Width	50
	t_{DVWH}	Data Setup to WE# Going High	50
	t_{WHDX}	Data Hold from WE# High	0
	t_{WHAX}	Address Hold from WE# High	5
	t_{WHEH}	CE# Hold from WE# High	5

NOTE:
Consult appropriate datasheets for up-to-date specifications.

3.0 Optimized I960 JF MICROPROCESSOR INTERFACE

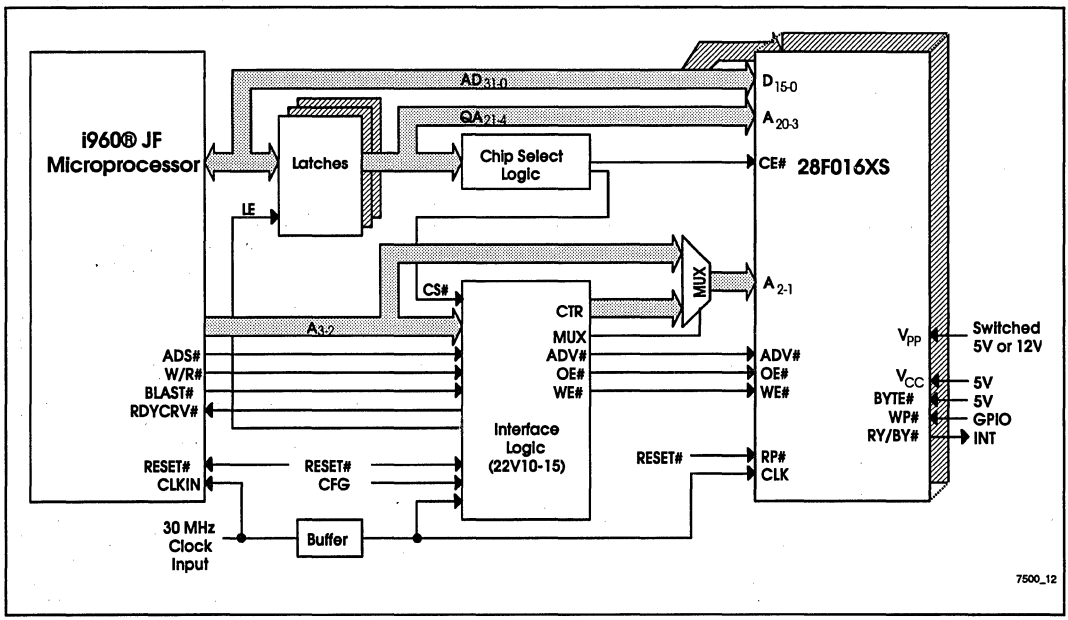


Figure 12. Minimal Interface Logic Required in Interfacing the 28F016XS-15 to the i960 JF-33 Microprocessor to Sustain 2-0-0-0 Burst Read Performance Up to 30 MHz

Using this interface, the 28F016XS interface to the i960 JF-33 microprocessor can achieve 2-0-0-0 wait-state read performance up to 30 MHz, supporting burst transfers. Contact your Intel or distribution sales office for schematic and PLD files for the design documented in the section.

See Section 4.0 for an alternative i960 JF design.

3.1 Circuit Description

This design, illustrated in Figure 12, uses two 28F016XS-15s to match the 32-bit data bus of the i960 JF-33 microprocessor. This configuration provides the system with 4 Mbytes of flash memory. Four octal latches, enabled by the LE signal, de-multiplex the 32-bit address from the AD bus. The latched address bits QA_{21-4} and the counter outputs CTR_{1-0} from the PLD select locations within the 28F016XS memory space. The two-bit counter implemented in the PLD loads and increments the processor's lower address lines at the beginning of each memory cycle. The processor supplies the 28F016XS-15s with the initial address during a read transaction, and the counter provides the subsequent burst addresses for the remaining duration of the read transaction.

CLK Option

A 33 MHz clock signal drives the i960 CA microprocessor CLKIN input. The buffer then delays the system CLK and drives the PLD and 28F016XS-15s. The buffer introduces an intentional system clock skew, providing additional time for the processor to meet the 28F016XSs' address setup time. At a slower operating frequency, this intentional delay may not be required to satisfy the 28F016XSs' address setup specification.

Multiplexer (MUX)

To achieve this type of wait-state profile, the processor directly loads the 28F016XSs with the address of the first access. The interface logic enables the MUX to permit the processor's lower address lines, A_{3-2}, access to the lower flash memory address lines during the initial access of a burst or single read transaction. Next, the interface logic switches the data flow path through the MUX. The two-bit counter integrated into the control logic then takes over, driving the 28F016XSs'

A_{2-1} address inputs. The counter supplies the flash memory with consecutive burst addresses for the remaining duration of the read transaction.

Reset

An active-low reset signal, RESET#, connects to the RESET# inputs of the i960 JF microprocessor and PLD and to the RP# input of the 28F016XS-15. Figure 5 illustrates a suggested logic configuration for generating RESET#.

Interface Control Signals

ADS# and W/R# i960 JF microprocessor signals, just as in the i960 CA microprocessor design, serve as inputs to the state machine, which controls the two-bit counter and generates the OE#, WE# and ADV# signals for the 28F016XS-15s. The state machine also generates the RDYRCV# signal for the i960 JF microprocessor to control the insertion of wait-states during data transfers.

Configuration Signal

A general purpose input/output (GPIO) generates the configuration signal for input to the state machine. The configuration signal must be reset to logic "0" on power-up and system reset to ensure that the operation of the state machine matches the 28F016XS-15s. After optimizing the 28F016XS-15s, the reconfiguration signal must switch to a logic "1" to take advantage of the new configuration.

Additional Control Signals

For information regarding BYTE#, WP#, RY/BY# and Vpp, see Section 2.1.

3.2 Software Interface Considerations

Boot-up Capability

This interface supports processor boot-up from the 28F016XS-15 memory space after power-up or system reset. Burst reads and writes may commence with no configuration. However, read wait-state performance will be 4-1-1-1 until the SFI Configuration is set to 2 and the CFG input is set to logic "1." Program control should jump to an area of RAM to execute the configuration sequence. A pseudocode flow for this configuration sequence is shown below.

Execute Device Configuration command sequence
Activate CFG signal
End

The SFI Configuration must be set to 2 before the CFG input is set to logic "1." Thereafter, burst read wait-state performance will improve to 2-0-0-0.

3.3 Burst Read Cycle Description

Refer to the read cycle timing diagrams and state diagram (Figure 13) for the following discussions of the read cycle.

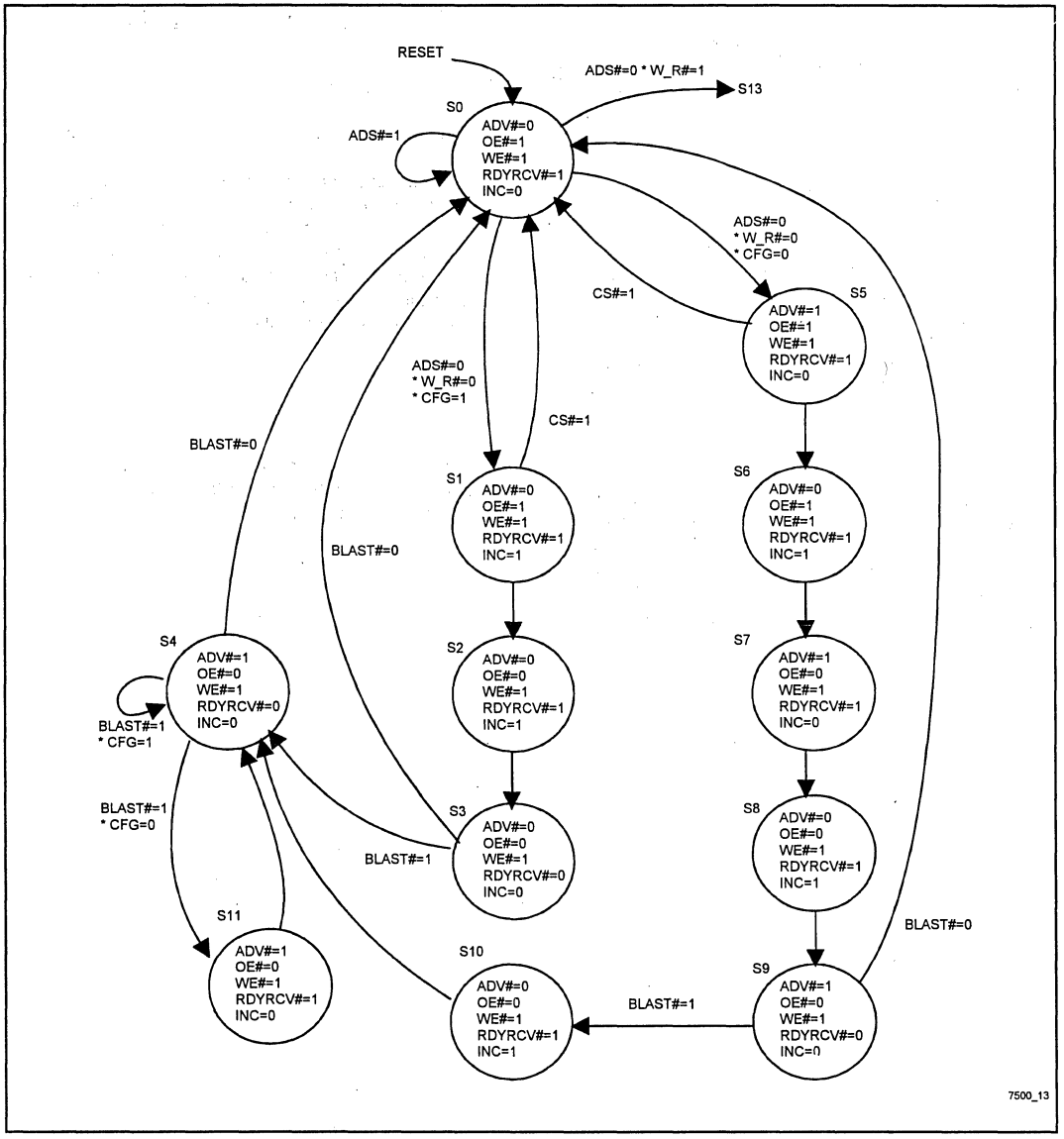


Figure 13. Read State Diagram of Burst Control Interface Shown in Figure 12

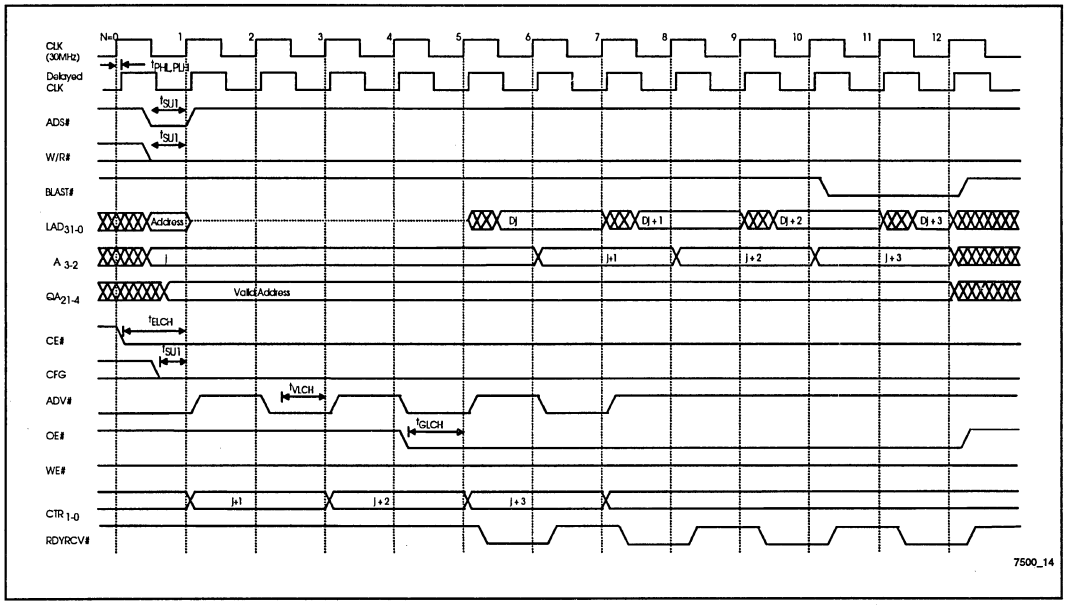


Figure 14. Example Four Double-Word Burst Read in Initial Configuration Showing Key Specifications Requiring Consideration

Initial Configuration

Figure 14 illustrates a four double-word burst read cycle with the 28F016XS-15s and state machine in a reset/power-up configuration state.

Initially, the interface logic drives LE, ADV# and MUX active while waiting for the processor to initiate a read cycle targeting the 28F016XSs. With the MUX active, the processor drives the lower flash memory address lines in anticipation of a flash memory access. During this anticipation state, CE# is active to prevent a t_{ELCH} violation on the first access initiated by the processor. The delayed CLK also prevents a possible timing violation from occurring by providing the processor with sufficient time to meet the 28F016XSs' t_{AVCH} specification when initiating the first access.

When the i960 JF microprocessor initiates a read cycle by asserting ADS# with W/R# = "0," presenting a valid address and control signals, the state machine loads and increments the two-bit counter. The counter is incremented upon the initial load sequence because the processor supplies the flash memory with the initial address.

The state machine will then de-assert ADV# at N = 1 and switches the data flow path through the MUX. The two-bit count then drives the flash memory's lower address lines for the remaining duration of the read transaction. The state machine then asserts ADV# at N = 3 to load the next read address into the 28F016XS-15s. De-asserting ADV# for one clock cycle (at N = 2, 4, 6 and 8) between accesses forces the 28F016XS-15s to hold data output for two clock cycles (access stretching), which allows time for the data to stabilize and meet the timing requirements of the i960 JF microprocessor bus.

While ADV# is asserted, the counter increments the two lower bits of the address, providing successive burst addresses. The state machine asserts OE# at N = 4 to enable the 28F016XS-15 data output buffers. With the SFI Configuration = 4, the data will be valid at the i960 JF microprocessor data inputs at N = 6, 8, 10 and 12.

The state machine asserts RDYRCV# to inform the i960 JF microprocessor that the data is valid. RDYRCV# is returned active at N = 6, 8, 10 and 12. The interface will follow this methodology until the processor asserts BLAST#, which identifies the end of the burst transaction. Upon detecting BLAST# active, the interface will transition to its idle state, waiting for another bus transaction to take place.

Optimized Configuration

Figure 15 illustrates a four double-word burst read with the 28F016XS-15s and state machine configured for optimum read performance. While the state machine waits for the i960 JF microprocessor to initiate a read cycle, ADV#, LE and MUX are held active enabling the processor with the capability of providing the flash memory with the initial address.

After the first access is initiated at N = 1, ADV# is held active and the counter increments through the remaining burst sequence. Data from the initial access will be available at N = 4. Subsequent data will be valid at N = 5, 6 and 7. All other signal monitoring and generation is identical to the reset/power-up configuration read cycle documented in the preceding section.

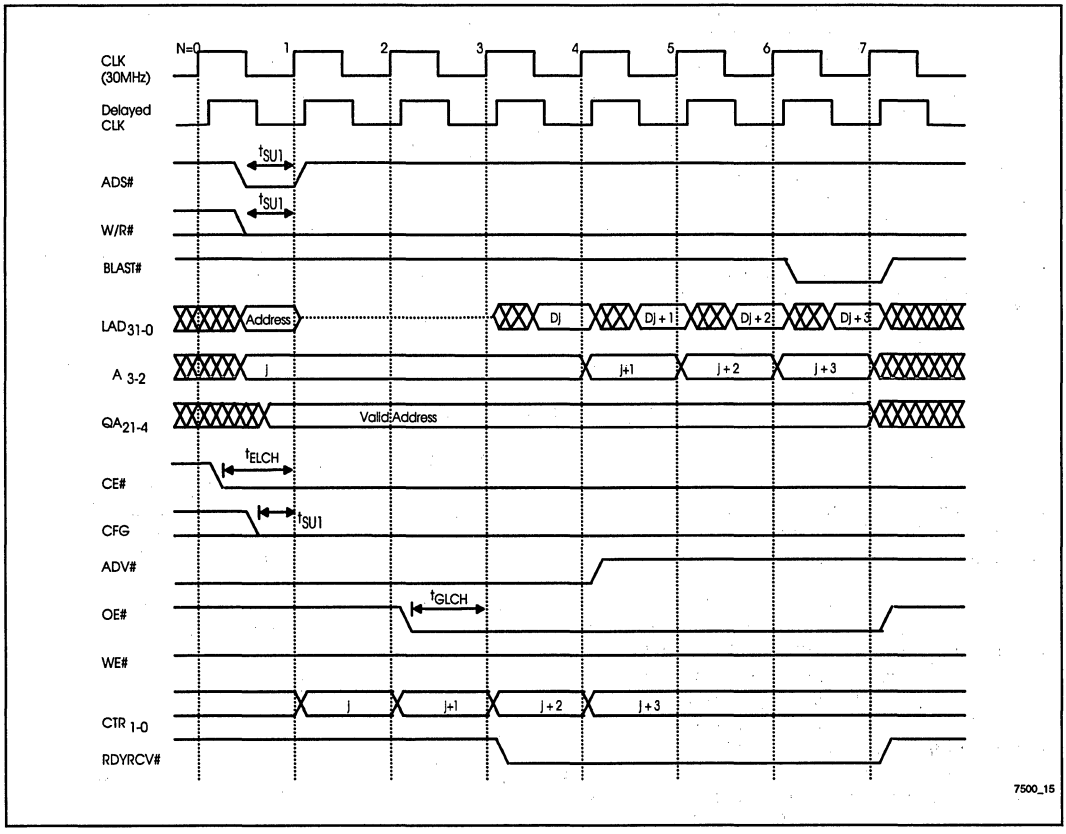


Figure 15. Example Four Double-Word Burst Read Illustrating Important Timing Parameters Requiring Consideration

Critical Timings

In these two read configurations, there are some important timing considerations that need to be taken into consideration. Table 4 depicts critical timings that are illustrated in Figures 14 and 15.

First, the buffer delay can cause possible timing violations if not chosen correctly for a given system operating frequency. The purpose of the buffer is to provide sufficient time for the processor to load the 28F016XSs with the initial address during a read transaction. Therefore, the buffer must have a minimum delay that satisfies the flash memory's t_{AVCH}.

Latch Delay - t_{OV1} - t_{AVCH} -1/CLKIN = Buffer Delay

As the previous equation illustrates, the minimum buffer delay is dependent upon several different variables: CLKIN frequency and latch delay. At a slow operation frequency, the interface does not require a buffer delay.

The buffer can also affect the processor's data setup time. Hence, the buffer must have a maximum delay of no greater than:

1/CLKIN - t_{CHQV} - t_{IS1} = Buffer Delay

Consult the appropriate datasheets for full timing information.

Table 4. Example Read Cycle Timing Parameters at 5V V_{CC}

Part	Symbol	Parameter	Minimum Specified Value (ns)
Buffer	$t_{PHL,PLH}$	Buffer Delay	1.5
22V10-15	t_{SU1}	Input Setup Time to CLK	9
28F016XS-15	t_{ELCH}	CE# Setup to CLK	25
	t_{VLCH}	ADV# Setup to CLK	15
	t_{AVCH}	Address Setup to CLK	15
	t_{GLCH}	OE# Setup to CLK	15

NOTE:
Consult appropriate datasheets for up-to-date specifications.

3.4 Burst Write Cycle Description at 33 MHz

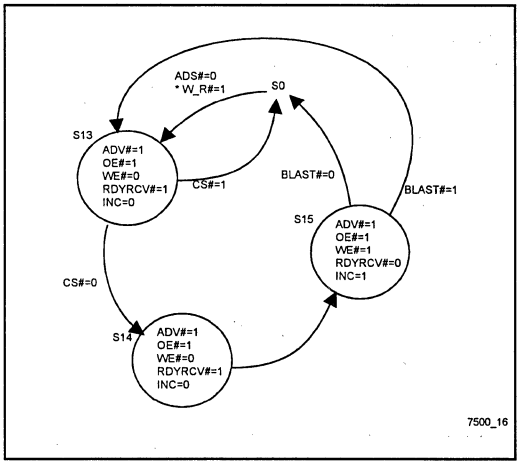


Figure 16. Write State Diagram of Burst Control Interface Shown in Figure 12

Write Configuration

Figure 17 illustrates a two double-word burst write cycle. The i960 JF microprocessor initiates a write cycle by asserting ADS# with W/R# = 1 and presenting a valid address and control signals. When detecting a write operation, the state machine immediately switches the data flow path through the MUX. The counter is given sole control to drive the 28F016XSs' lower address lines during write operations. At N = 1 with ADS# = 0, the two-bit counter loads the values on the processor's lower address lines. The state machine asserts WE# (to meet timing requirements, WE# is falling-edge triggered) on the falling edge between N = 1 and N = 2. WE# remains asserted for four clock periods, in order to meet 28F016XS-15 timing requirements. The state machine asserts RDYRCV# for N = 4 to inform the i960 KB microprocessor to supply the next data. At N = 4, the counter increments the two lower address bits, and the state machine asserts WE# on the next falling clock edge to begin the next data write to the 28F016XS-15s. The data writes continue until the processor asserts BLAST#, noting the end of the current write transaction. The SFI Configuration has no effect on the write cycle.

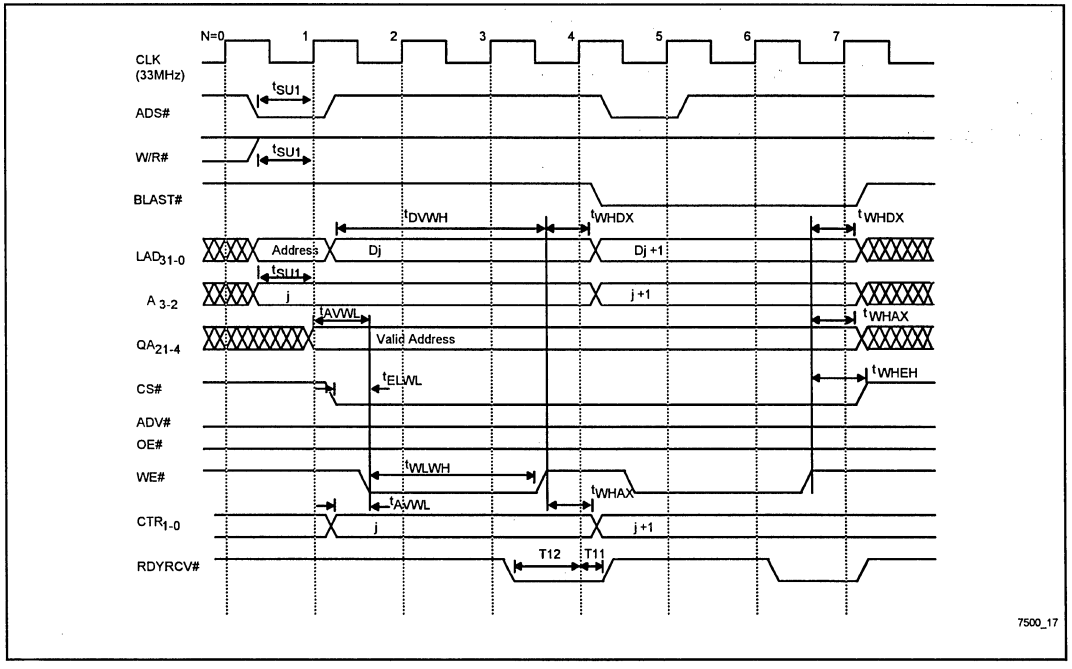


Figure 17. Two Double-Word Burst Write

Critical Timings

Table 5 describes the critical timings illustrated in Figure 17.

Notice that CTR_{1-0}, and CS# must be valid before WE# is asserted. CTR_{1-0} are guaranteed valid 8 ns after the rising clock edge, providing 12 ns of margin.

Consult the appropriate datasheets for full timing information.

Table 5. Example Write Cycle Timing Parameters at 5V V_{CC}

Part	Symbol	Parameter	Minimum Specified Value (ns)
22V10-15	t_{SU1}	Input Setup Time to CLK	9
28F016XS-15	t_{ELWL}	CE# Setup to WE# Going Low	0
	t_{AVWL}	Address Setup to WE# Going Low	0
	t_{WLWH}	WE# Pulse Width	50
	t_{DVWH}	Data Setup to WE# Going High	50
	t_{WHDX}	Data Hold from WE# High	0
	t_{WHAX}	Address Hold from WE# High	5
	t_{WHEH}	CE# Hold from WE# High	5

NOTES:
Consult appropriate datasheets for up-to-date specifications.

4.0 Standard i960 JF MICROPROCESSOR INTERFACE

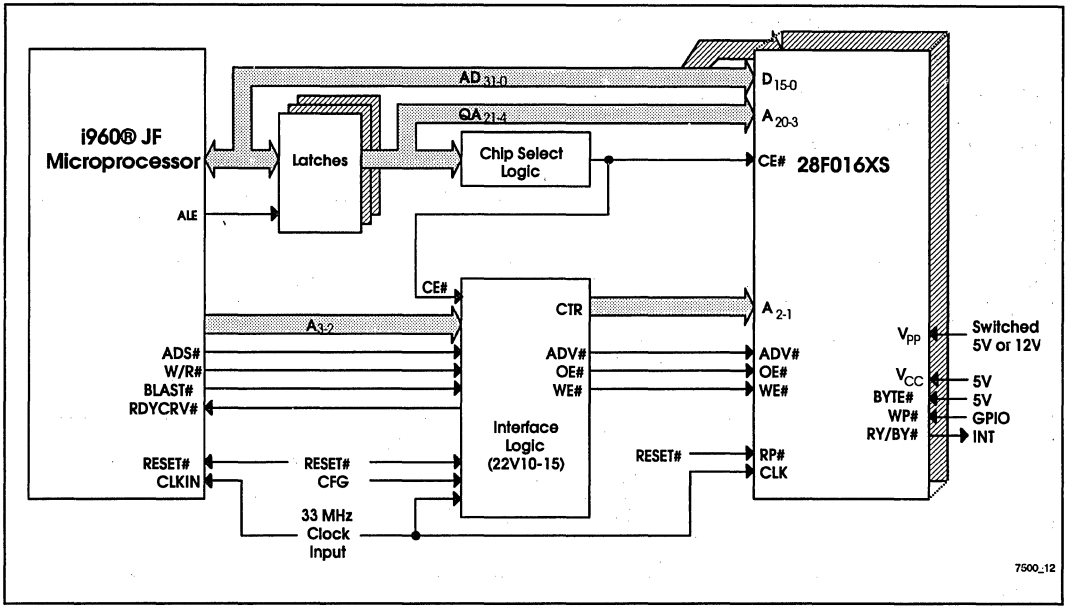


Figure 18. Minimal Interface Logic Required in Interfacing the 28F016XS-15 to the i960 JF-33 Microprocessor to Sustain 3-0-0-0 Burst Read Performance Up to 33 MHz

The 28F016XS-15 interface to the i960 JF-33 microprocessor, illustrate in Figure 12, delivers 3-0-0-0 wait-state read performance up to 33 MHz. The design requires only one 22V10 to handle all interfacing requirements. Contact your Intel or distribution sales office for schematic and PLD files for the interface documented in this section.

See Section 3.0 for an alternative design.

4.1. Circuit Description

This interface is very similar to the optimized i960 JF design described in Section 3.0. This design however, eliminates the buffer and multiplexer requirement. For specific circuitry information involved in this standard interface, reference Section 3.0.

CLK Option

Unlike the optimized design in Section 3.0, a buffer is not implemented in this interface. The 33 MHz system clock drives the processor, PLD and flash memory. To reduce system clock skew, position the PLD and 28F016XSs within close proximity to the microprocessor.

4.2. Software Interface Considerations

This interface supports processor boot-up from the flash memory space after power-up or system reset. Initially, the read wait-state performance will be 5-1-1-1 until the SFI Configuration value is modified. The 28F016XS operates at optimal performance with a SFI Configuration value of 2 at 33 MHz. Program control should jump to an area of RAM to execute a configuration sequence which optimizes the flash memory and interface state machine for the given operating frequency.

4.3 Burst Read Cycle Description at 33 MHz

Refer to the read cycle timing diagrams and state diagram (Figure 13) for the following discussions of the read cycle.

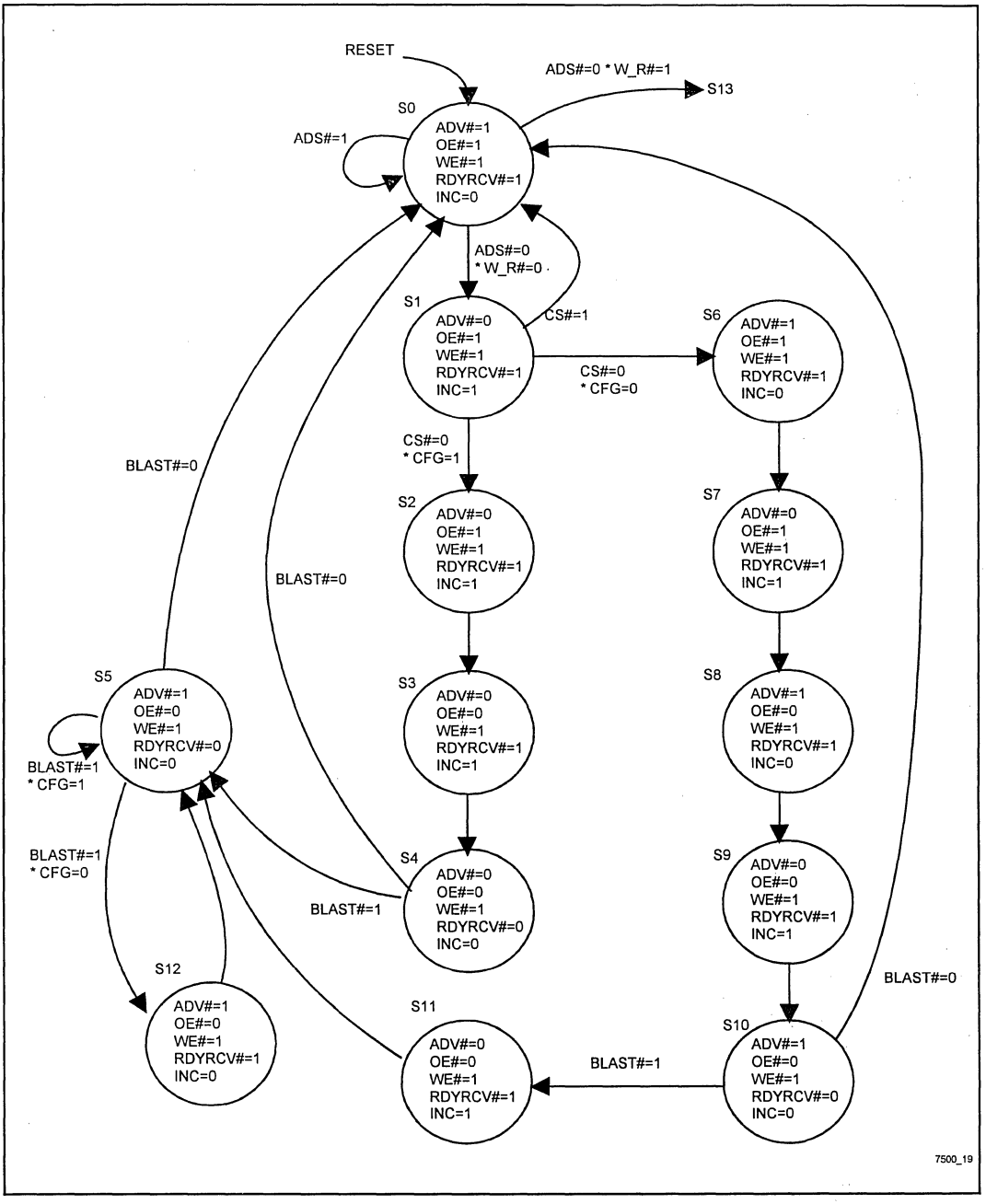


Figure 19. Read State Diagram of Burst Control Interface Shown in Figure 18

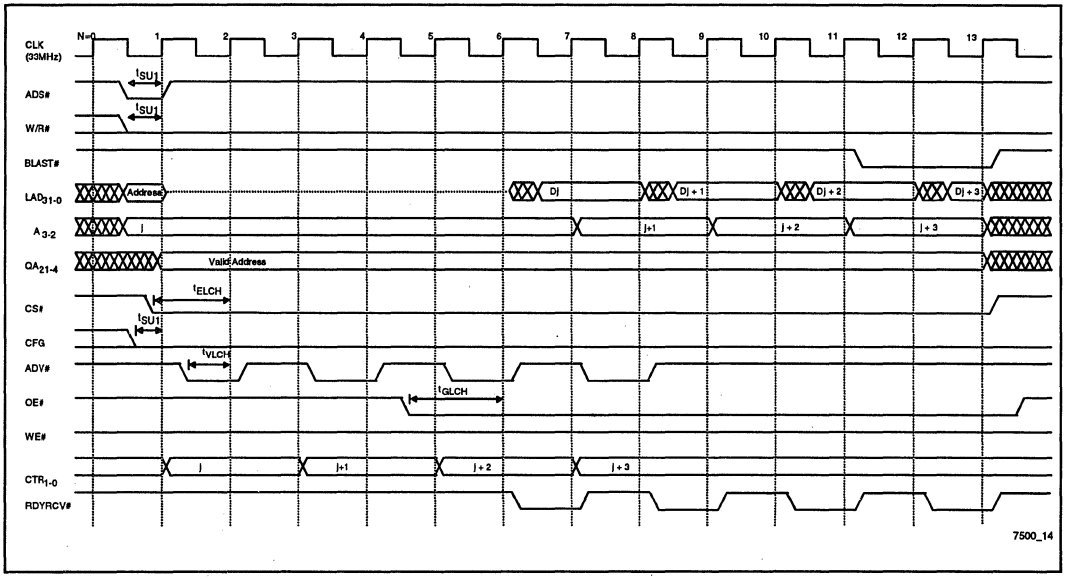


Figure 20. Example Four Double-Word Burst Read in Initial Configuration Showing Key Specifications Requiring Consideration

Initial Configuration

Figure 20 illustrates a four double-word burst read cycle with the 28F016XS-15s and the state machine in the reset/power-up configuration state. The i960 JF microprocessor initiates a read cycle by asserting ADS# with W/R# = "0," presenting a valid address and control signals. At N = 2 with ADS# = "0" and CLK = "0," the two-bit counter loads the values on the address bits A_{3-2}.

The state machine asserts ADV# after clock edge N = 1 where the 28F016XS-15s will clock in the first address at the next rising clock edge (N=2), if CS# is asserted. If CS# is not asserted, the state machine will return to its inactive state at N = 2.

The state machine de-asserts ADV# at N = 2. The state machine then asserts ADV# at N = 3 to load the next read address into the 28F016XS-15s. De-asserting ADV# for one clock cycle (at N = 2, 4, 6 and 8) between accesses forces the 28F016XS-15s to hold data output for two clock cycles (access stretching), which allows time for the data to stabilize and meet the timing requirements of the i960 JF microprocessor bus.

The counter increments the two lower bits of the address at N = 3, 5 and 7 to provide the four successive burst addresses. The state machine asserts OE# at N = 4 to enable the 28F016XS-15 data output buffers. With the SFI Configuration = 4, the data will be valid at the i960 JF microprocessor data inputs at N = 7.

The state machine asserts RDYRCV# to inform the i960 JF microprocessor that the data is valid. RDYRCV# is returned active at N = 7, 9, 11 and 12. When sampling BLAST# active, the state machine ends the read transfer and returns to an idle state. In the idle state, the state machine simply waits for the processor to initiate another bus transaction.

Optimized Configuration

Figure 15 illustrates a four double-word burst read with the 28F016XS-15s and state machine configured for optimum read performance. With the SFI Configuration = 2, ADV# is held active and the counter increments at N = 2, 3 and 4, supplying the 28F016XS-15s with four consecutive accesses. Data from the initial access will be valid for transfer at N = 5. Subsequent data will be valid at N = 6, 7 and 8. This interface and 28F016XS-15 configuration improve read wait-state performance to 3-0-0-0. All other signal monitoring and generation are identical to the reset/power-up configuration read cycle documented in the preceding section.

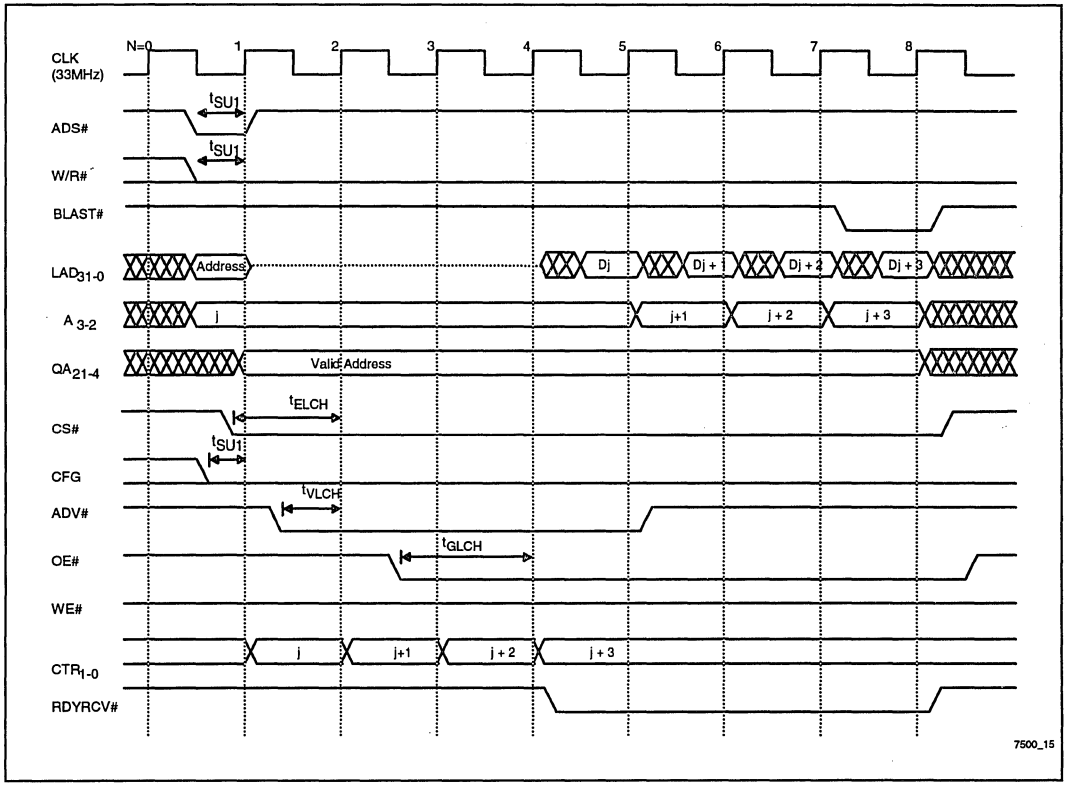


Figure 21. Example Four Double-Word Burst Read Illustrating Important Timing Parameters Requiring Consideration

Critical Timings

Table 4 describes the critical timings illustrated in Figures 14 and 15. One particularly critical timing in this design is the data hold time, which the 28F016XS-15 meets with 0 ns margin. The 28F016XS holds data for 5 ns after a rising clock.

The 28F016XS-15 will provide data 10 ns before the rising edge of the system clock, which satisfies the i960 JF-33 microprocessor's data input requirement.

ADV# and CTR_{1-0} are guaranteed valid 8 ns after the rising clock edge. Setup times for these inputs to 28F016XS-15 are each 15 ns. Since the clock period is 30 ns, this allows 7 ns margin for these timings.

RDYRCV# is guaranteed valid 8 ns after the rising clock edge to met the microprocessor's setup time to rising clock edge.

Consult the appropriate datasheets for full timing information.

Table 6. Example Write Cycle Timing Parameters at 5V V_{CC}

Part	Symbol	Parameter	Minimum Specified Value (ns)
22V10-15	t_{SU1}	Input Setup Time to CLK	9
28F016XS-15	t_{ELCH}	CE# Setup to CLK	25
	t_{VLCH}	ADV# Setup to CLK	15
	t_{AVCH}	Address Setup to CLK	15
	t_{GLCH}	OE# Setup to CLK	15

NOTE:
Consult appropriate datasheets for up-to-date specifications.

4.4 Burst Write Cycle Description at 33 MHz

The write interface in for this design behaves similar to the optimized design's write interface described in Section 3.4. The only difference between the two designs is the absence of the MUX. Therefore, this design's write state machine does not have to concern itself with changing the data flow through the MUX. Instead, the interface simply loads and holds the address presented by the processor for the duration of the write operation.

For further details information about this cycle and write timing waveform, refer to Section 3.4.

5.0 INTERFACING TO OTHER i960 MICROPROCESSORS

i960 CF-16, i960 CF-25 and I960 CF-33 Microprocessors

The i960 CF microprocessor bus interface is completely compatible with the i960 CA microprocessor bus interface. Therefore, the 28F016XS-15 interfaces described above for the i960 CA-33 microprocessor work equally well with the i960 CF-25 and 33 MHz microprocessors.

At 16 MHz, the interface requires a slight modification because the SFI Configuration value at 16 MHz equals 1. The 28F016XS-15 will begin driving the data pin 1 CLK period after initiating a read access. The interface returns READY# to the i960 CF-16 microprocessor, 1 CLK cycle earlier. Therefore, the 28F016XS-15 interface to the i960 CF-16 microprocessor will deliver 3-0-0-0 wait-state read performance.

i960 KA and i960 KB Microprocessors

The 28F016XS interface to the i960 Kx microprocessor series will be very similar to the i960 JF interface described in Sections 3.0 and 4.0. The only difference between the two designs is the i960 Kx's lack of a BLAST# output signal. The i960 JF interfaces use this signal to determine the end to the burst transaction. Since the i960 Kx microprocessor does not have a BLAST# signal, the interface logic must examine the processor's lower address lines to determine the length of the read transaction.

For further detailed information about this interface, please reference the i960 JF interfaces. Contact your Intel or distribution sales office for schematic and PLD files for the 28F016XS interface to the i960 Kx microprocessor.

i960 SA Microprocessor, i960 SB Microprocessor

The 28F016XS's interface to the i960 Sx microprocessor series will be similar to the i960 JF microprocessor interfaces, with the following differences:

- The i960 Sx microprocessor series has a 16-bit data bus multiplexed with the lower 16 of 32 address bits. Therefore, a single 28F016XS will match the width of the data bus.
- The i960 Sx microprocessor series supports eight double-word burst transfers. Therefore, the 28F016XS interface will require a three-bit counter to generate the lower three bits of the burst addresses.

6.0 CONCLUSION

This technical paper has described the interface between the 28F016XS 16-Mbit Flash memory component and the i960 microprocessor family. This simple design has been implemented with a minimal number of components and achieves exceptional read performance. The 28F016XS provides the microprocessor with the non-volatility and updateability of flash memory and the performance of DRAM. For further information about the 28F016XS, reference the Additional Information section of this technical paper. Please contact your local Intel or distribution sales office for more information on Intel's flash memory products.

ADDITIONAL INFORMATION

Order Number	Document/Tools
290532	28F016XS Datasheet
297500	"Interfacing 28F016XS to the Intel486™ Microprocessor Family"
292147	AP-398, "Designing with the 28F016XS"
292146	AP-600, "Performance Benefits and Power/Energy Savings of 28F016XS Based System Designs"
292163	AP-610, "Flash Memory In-System Code and Data Update Techniques"
292165	AB-62, "Compiled Code Optimizations For Embedded Flash RAM Memories"
297372	16-Mbit Flash Product Family User's Manual
297508	FLASHBuilder Utility
Contact Intel/Distribution Sales Office	28F016XS Benchmark Utility
Contact Intel/Distribution Sales Office	28F016XS iBIS Models
Contact Intel/Distribution Sales Office	28F016XS VHDL/Verilog Models
Contact Intel/Distribution Sales Office	28F016XS Timing Designer Library Files
Contact Intel/Distribution Sales Office	28F016XS Orcad and ViewLogic Schematic Symbols

REVISION HISTORY

Number	Description
001	Original Version
002	Added Optimized i960 JF Microprocessor Interface that Delivers 2-0-0-0 Wait-State Performance Up to 30 MHz. Removed Detailed i960 KB Microprocessor Interface Description Added i960 KB Microprocessor Interfacing Guidelines to Section 5.0, "Interfacing to Other i960 Microprocessors"

APPENDIX A
PLD FILES

PLD file for the standard 28F016XS interface to the i960 CA Microprocessor described in Section 2.0.

```
Title           28F016XS / i960® CA Microprocessor Interface State Machine
Pattern         PDS
Revision        1
Authors         Example
Company         Intel Corporation - Folsom, California
Date            1-25-94

CHIP      STATEMACHINE      85C22V10

; inputs
PIN   1         CLK
PIN             ADS_n           ; address status - i960 CA microprocessor
PIN             W_R_n           ; W/R# - i960 CA microprocessor
PIN             BLAST_n         ; burst last - i960 CA microprocessor
PIN             CS_n            ; chip select - 28F016XS
PIN             CFG             ; 28F016XS/i960 CA microprocessor config status set input
PIN             A2              ; LAD bit 2
PIN             A3              ; LAD bit 3
PIN             RESET           ; resets all FFs in device
PIN   25        GLOBAL          ; virtual pin to implement reset

; outputs
PIN             CTR0            ; burst counter out - 28F016XS-A1
PIN             CTR1            ; burst counter out - 28F016XS-A2
PIN             /WE             ; write enable - 28F016XS
PIN             /OE             ; output enable - 28F016XS
PIN             Q0              ; state variables
PIN             Q1
PIN             /ADV            ; state variable and address valid - 28F016XS
PIN             Q3

 ; burst counter control signals
STRING  LD '(/ADS_n)'                   ; load
STRING  INC '(/ADV + ADV * Q3 * Q1 * Q0)'       ; increment

STATE  MOORE_MACHINE
DEFAULT_BRANCH S0

; state assignments
        S0 =    /Q3 * /ADV * /Q1 * /Q0
        S1 =    /Q3 *  ADV * /Q1 * /Q0
        S2 =     Q3 *  ADV * /Q1 * /Q0
        S3 =     Q3 *  ADV * /Q1 *  Q0
        S4 =    /Q3 *  ADV * /Q1 *  Q0
```

```
            S5 =  /Q3 * /ADV * /Q1 *  Q0
            S6 =  /Q3 * /ADV *  Q1 * /Q0
            S7 =   Q3 * /ADV *  Q1 * /Q0
            S8 =   Q3 * /ADV * /Q1 * /Q0

; state transitions
S0 :=   (/ADS_n * /W_R_n)   -> S1     ; READ cycle
   +    (/ADS_n * W_R_n)    -> S6     ; WRITE cycle
                           +-> S0     ; else, stay
S1 :=   (/CS_n * /CFG)       -> S5     ; 28F016XS selected, initial configurations
   +    (/CS_n * CFG)        -> S2     ; 28F016XS selected, 28F016XS and i960 CA microprocessor configured
                           +-> S0     ; else, return to idle state
S2 :=   VCC                  -> S3
S3 :=   VCC                  -> S4     ; 28F016XS is configured to wait 4 clocks
S4 :=   (/BLAST_n * ADS_n)  -> S0     ; 1 double word  read
   +    (/BLAST_n * /ADS_n) -> S1     ; pipelined read
                           +-> S5     ; else, continue
S5 :=   (/BLAST_n * ADS_n)  -> S0     ; burst read finished
  +     (/BLAST_n * /ADS_n) -> S1     ; pipelined read
                           +-> S5     ; else, continue
S6 :=   /CS_n                -> S7     ; 28F016XS selected, continue
                           +-> S0     ; else, return to idle state
S7 :=   VCC                  -> S8
S8 :=   (BLAST_n * CFG)      -> S6     ; continue burst
   +    (BLAST_n * /CFG)     -> S8     ; pre-config write
                           +-> S0     ; write is finished

; transition outputs
S0.OUTF :=     /OE * /WE
S1.OUTF :=     /OE * /WE
S2.OUTF :=      OE * /WE
S3.OUTF :=      OE * /WE
S4.OUTF :=      OE * /WE
S5.OUTF :=      OE * /WE
S6.OUTF :=      OE * /WE
S7.OUTF :=     /OE *  WE
S8.OUTF :=     /OE *  WE
S9.OUTF :=     /OE * /WE

EQUATIONS
  ; implement RESET
      GLOBAL.RSTF = /RESET
  ; implement 2-bit burst counter -  registered counter equations
      CTR1 := (LD * A3) + (/LD * INC * CTR0 * /CTR1)
            + (/LD * INC * /CTR0 * CTR1) + (/LD * /INC * CTR1)
      CTR0 := (LD * A2) + (/LD * INC * /CTR0) + (/LD * /INC * CTR0)
  ; flop OE and WE on falling edge
      OE.CLKF = /CLK
      WE.CLKF = /CLK
```

PLD file for the optimized 28F016XS interface to the i960 JF Microprocessor described in Section 3.0.

```
Title           Optimized 28F016XS/i960® JF Microprocessor Interface State Machine
Pattern         PDS
Revision        1
Authors         Example
Company         Intel Corporation - Folsom, California
Date            2-16-95

CHIP      STATEMACHINE      85C22V10

; inputs
PIN   1         CLK
PIN             ADS             ; address status - i960 JF microprocessor
PIN             W_R             ; W/R# - i960 JF microprocessor
PIN             BLAST           ; burst last - i960 JF microprocessor
PIN             CS              ; chip setect
PIN             CFG             ; 28F016XS/i960 JF microprocessor config status set input
PIN             A2              ; A bit 2
PIN             A3              ; A bit 3
PIN             RESET           ; resets all FFs in device
PIN   25        GLOBAL          ; virtual pin to implement reset

; outputs
PIN             CTR0            ; burst counter out - 28F016XS-A1
PIN             CTR1            ; burst counter out - 28F016XS-A2
PIN             /WE             ; write enable - 28F016XS
PIN             /OE             ; output enable - 28F016XS
PIN             /RDYRCV         ; wait-state control
PIN             MUX             ; control data flow through the multiplexer
PIN             Q0              ; state variables
PIN             Q1
PIN             /ADV            ; state variable and address valid - 28F016XS
PIN             Q3

 ; burst counter control signals
STRING  LD '(/ADS)'                     ; load
STRING  INC '(/ADV + /RDYRCV * Q3 * Q1 * Q0)'    ; increment

STATE MOORE_MACHINE
DEFAULT_BRANCH S0

; state assignments
        S0 =  /RDYRCV * /Q3 *  ADV * /Q1 * /Q0 * /OE * /WE * /MUX
        S1 =  /RDYRCV * /Q3 *  ADV * /Q1 *  Q0 *  OE * /WE* MUX
        S2 =  /RDYRCV * /Q3 *  ADV *  Q1 * /Q0 *  OE * /WE * MUX
        S3 =   RDYRCV * /Q3 *  ADV *  Q1 *  Q0 *  OE * /WE* MUX
        S4 =   RDYRCV * /Q3 * /ADV * /Q1 *  Q0 *  OE * /WE* MUX
        S5 =  /RDYRCV * /Q3 * /ADV *  Q1 * /Q0 * /OE * /WE* MUX
```

```
        S6 =  /RDYRCV *  Q3 *  ADV * /Q1 * /Q0 *  OE * /WE * MUX
        S7 =  /RDYRCV * /Q3 * /ADV *  Q1 *  Q0 *  OE * /WE* MUX
        S8 =  /RDYRCV *  Q3 *  ADV * /Q1 *  Q0 *  OE * /WE* MUX
        S9 =  RDYRCV *  Q3 * /ADV * /Q1 * /Q0 *  OE * /WE* MUX
        S10 = /RDYRCV *  Q3 *  ADV *  Q1 * /Q0 *  OE * /WE* MUX
        S11 = /RDYRCV *  Q3 * /ADV * /Q1 *  Q0 *  OE * /WE* MUX
        S13 = /RDYRCV *  Q3 * /ADV *  Q1 * /Q0 * /OE *  WE* MUX
        S14 = /RDYRCV *  Q3 * /ADV *  Q1 *  Q0 * /OE *  WE* MUX
        S15 =  RDYRCV *  Q3 * /ADV *  Q1 *  Q0 * /OE * /WE* MUX

; state transitions
S0 :=   (/ADS * /W_R *  CFG) -> S1      ; READ cycle
  +     (/ADS * /W_R * /CFG) ->S5
  +     (/ADS *  W_R)          -> S13   ; WRITE cycle
                              +-> S0    ; else, stay
S1 :=    /CS                   -> S2    ; 28F016XS selected, init configurations
  +       CS                   -> S0    ; 28F016XS selected, optimized configured
S2 :=   VCC                    -> S3
S3 :=   (/BLAST)               -> S0    ; 1 double word read
                              +-> S4    ; else, continue
S4 :=   /BLAST                 -> S0    ; burst read finished
  +     (BLAST *  CFG)         -> S4    ; continue, optimized configuration
  +     (BLAST * /CFG)         -> S11   ; continue, initial configuration
S5 :=   VCC                    -> S6
S6 :=   VCC                    -> S7
S7 :=   VCC                    -> S8
S8 :=   VCC                    -> S9
S9 :=   /BLAST                 -> S0    ; BLAST -  end of the burst read transaction
  +     BLAST                  -> S10
S10:=   VCC                    -> S4
S11:=   VCC                    -> S4
S13:=   CS                     -> S0    ; write cycle control
  +     /CS                    -> S14
S14:=   VCC                    -> S15
S15:=   BLAST                  -> S13   ; BLAST - end of burst write transaction
  +     /BLAST                 -> S0

EQUATIONS
  ; implement RESET
      GLOBAL.RSTF = /RESET
  ; implement 2-bit burst counter -  registered counter equations
CTR1 := (/WR * LD * A3 * /A2) + (WR * LD * A3) + (/LD * INC * CTR0 * /CTR1)
       + (/LD * INC * /CTR0 * CTR1) + (/LD * /INC * CTR1)
CTR0 := (/WR * LD * /A2) + (WR * LD * A2) + (/LD * INC * /CTR0) + (/LD * /INC * CTR0)
```

PLD file for the standard 28F016XS interface to the i960 JF Microprocessor described in Section 4.0.

```
Title           28F016XS/i960® JF Microprocessor Interface State Machine
Pattern         PDS
Revision        1
Authors         Example
Company         Intel Corporation - Folsom, California
Date            8-16-94

CHIP      STATEMACHINE      85C22V10

; inputs
PIN   1         CLK
PIN             ADS             ; address status - i960 FJ microprocessor
PIN             W_R             ; W/R# - i960 FJ microprocessor
PIN             BLAST           ; burst last - i960 JF microprocessor
PIN             CS              ; chip setect
PIN             CFG             ; 28F016XS/i960 JF microprocessor config status set input
PIN             A2              ; A bit 2
PIN             A3              ; A bit 3
PIN             RESET           ; resets all FFs in device
PIN   25        GLOBAL          ; virtual pin to implement reset

; outputs
PIN             CTR0            ; burst counter out - 28F016XS-A1
PIN             CTR1            ; burst counter out - 28F016XS-A2
PIN             /WE             ; write enable - 28F016XS
PIN             /OE             ; output enable - 28F016XS
PIN             /RDYRCV         ; wait-state control
PIN             Q0              ; state variables
PIN             Q1
PIN             /ADV            ; state variable and address valid - 28F016XS
PIN             Q3

 ; burst counter control signals
STRING  LD '(/ADS)'             ; load
STRING  INC '(/ADV + /RDYRCV * Q3 * Q1 * Q0)'   ; increment

STATE MOORE_MACHINE
DEFAULT_BRANCH S0

; state assignments
        S0 =    /RDYRCV * /Q3 * /ADV * /Q1 * /Q0 * /OE * /WE
        S1 =    /RDYRCV * /Q3 *  ADV * /Q1 * /Q0 * /OE * /WE
        S2 =    /RDYRCV * /Q3 *  ADV * /Q1 *  Q0 *  OE * /WE
        S3 =    /RDYRCV * /Q3 *  ADV *  Q1 * /Q0 *  OE * /WE
        S4 =     RDYRCV * /Q3 *  ADV *  Q1 *  Q0 *  OE * /WE
        S5 =     RDYRCV * /Q3 * /ADV * /Q1 *  Q0 *  OE * /WE
        S6 =    /RDYRCV * /Q3 * /ADV *  Q1 * /Q0 * /OE * /WE
```

```
        S7  =  /RDYRCV *  Q3 *  ADV * /Q1 * /Q0 *  OE * /WE
        S8  =  /RDYRCV * /Q3 * /ADV *  Q1 *  Q0 *  OE * /WE
        S9  =  /RDYRCV *  Q3 *  ADV * /Q1 *  Q0 *  OE * /WE
        S10 =   RDYRCV *  Q3 * /ADV * /Q1 * /Q0 *  OE * /WE
        S11 =  /RDYRCV *  Q3 *  ADV *  Q1 * /Q0 *  OE * /WE
        S12 =  /RDYRCV *  Q3 * /ADV * /Q1 *  Q0 *  OE * /WE
        S13 =  /RDYRCV *  Q3 * /ADV *  Q1 * /Q0 * /OE *  WE
        S14 =  /RDYRCV *  Q3 * /ADV *  Q1 *  Q0 * /OE *  WE
        S15 =   RDYRCV *  Q3 * /ADV *  Q1 *  Q0 * /OE * /WE

; state transitions
S0 :=   (/ADS * /W_R)          -> S1     ; READ cycle
  +     (/ADS *  W_R)          -> S13    ; WRITE cycle
                              +-> S0     ; else, stay
S1 :=   (/CS * /CFG)           -> S6     ; 28F016XS selected, init configurations
  +     (/CS * CFG)            -> S2     ; 28F016XS selected, optimized configured
                              +-> S0     ; else, return to idle state
S2 :=   VCC                    -> S3
S3 :=   VCC                    -> S4     ; 28F016XS is configured to wait 4 clocks
S4 :=   (/BLAST * ADS)         -> S0     ; 1 double word read
                              +-> S5     ; else, continue
S5 :=   /BLAST                 -> S0     ; burst read finished
  +     (BLAST *  CFG)         -> S5     ; continue, optimized configuration
  +     (BLAST * /CFG)         -> S12    ; continue, initial configuration
S6 :=   VCC                    -> S7
S7 :=   VCC                    -> S8
S8 :=   VCC                    -> S9
S9 :=   VCC                    -> S10
S10:=   /BLAST                 -> S0     ; BLAST -  end of the burst read transaction
  +     BLAST                  -> S11
S11:=   VCC                    -> S5
S12:=   VCC                    -> S5
S13:=   CS                     -> S0     ; write cycle control
  +     /CS                    -> S14
S14:=   VCC                    -> S15
S15:=   BLAST                  -> S13    ; BLAST - end of burst write transaction
  +     /BLAST                 -> S0

EQUATIONS
  ; implement RESET
      GLOBAL.RSTF = /RESET
  ; implement 2-bit burst counter -  registered counter equations
      CTR1 := (LD * A3) + (/LD * INC * CTR0 * /CTR1)
              + (/LD * INC * /CTR0 * CTR1) + (/LD * /INC * CTR1)
      CTR0 := (LD * A2) + (/LD * INC * /CTR0) + (/LD * /INC * CTR0)
```

APPENDIX B
BENCHMARK PERFORMANCE ANALYSIS

The following section provides detailed memory technology information used in the performance analysis (UDP/IP Networking and Imaging Benchmarks) contained in the introduction. The performance analysis was based on actual memory component performance in an i960 processor-based environment. System interface delay between microprocessor and memory was not included in the analysis. The two benchmarks illustrate relative system memory performance.

A. 28F016XS Flash Memory

28F016XS is capable of 3-1-1-1-1-1-1-1. . .read performance at 5.0V V_{CC} and 33 MHz or 25 MHz (2-0-0-0-0-0-0-0. . .in terms of wait-states). The benchmarking analysis is shown below:

	UDP/IP Networking Benchmark	Imaging Benchmark
	Time (sec)	*Time (sec)*
i960 KB-25 Microprocessor	1.30	1.64
i960 CA-33 Microprocessor	.89	.89
i960 CF-33 Microprocessor	.53	.59

B. 16-Mbit DRAM

16-Mbit DRAMs were, at the time this technical paper was published, only beginning to ramp into production. Only advance information for the wider x16, 16-Mbit DRAMs was available for use in the calculations that follow.

Sequential reads allow use of the DRAM fast page mode. Assumed DRAM specifications are shown below:

- 80 ns t_{RAC}, 40 ns t_{AA} (5.0V V_{CC})
- 256 word (512 byte) page buffer

Therefore, 16-Mbit DRAMs are capable of 3-2-2-2-2-2-2-2. . .read performance at 33 MHz and 25 MHz (2-1-1-1-1-1-1-1 in terms of wait-states. . .). The benchmarking analysis is shown below:

	UDP/IP Networking Benchmark	Imaging Benchmark
	Time (sec)	*Time (sec)*
i960 KB-25 Microprocessor	1.88	1.89
i960 CA-33 Microprocessor	1.06	1.03
i960 CF-33 Microprocessor	.59	.64

C. 4-Mbit EPROM

Calculations that follow used the x16 version of the 4-Mbit EPROM (Intel 27C400 or equivalent).

The assumed 5.0V V_{CC} 4-Mbit EPROM random access time is 150 ns. Therefore, 4-Mbit EPROMs are capable of 5-5-5-5-5-5-5-5. . .read performance at 5.0V V_{CC} and 33 MHz (4-4-4-4-4-4-4-4. . .in terms of wait-states). The benchmarking analysis is shown below:

	UDP/IP Networking Benchmark	Imaging Benchmark
	Time (sec)	*Time (sec)*
i960 KB-25 Microprocessor	NA	NA
i960 CA-33 Microprocessor	1.56	1.49
i960 CF-33 Microprocessor	.78	.81

D. 16-Mbit PAGED MASK ROM

Calculations that follow used the x16 version of the 16-Mbit paged mask ROM, which is not yet widely available from multiple vendors. The x8, 16-Mbit paged mask ROM is the more common version today.

Sequential reads allow use of the mask ROM page mode. The assumed 5.0V V_{CC} 16-Mbit mask ROM random access time is 150 ns, with 75 ns accesses in page mode (4-word page). Therefore, 16-Mbit mask ROMs are capable of 5-3-3-3-5-3-3-3. . .read performance at 5.0V V_{CC} and 33 MHz (4-2-2-2-4-2-2-2. . .in terms of wait-states). The benchmarking analysis is shown below:

	UDP/IP Networking Benchmark	Imaging Benchmark (sec)
	Time (sec)	*Time (sec)*
i960 KB-25 Microprocessor	NA	NA
i960 CA-33 Microprocessor	1.35	1.30
i960 CF-33 Microprocessor	.71	.73

intel®

Revised Pages

intel®

Flash Memory Overview

In contrast, Intel flash memory is inherently nonvolatile, and the single transistor cell design of Intel's ETOX manufacturing process is extremely scaleable, allowing the development of continuously higher densities and steady cost improvement over SRAM (Figure 2).

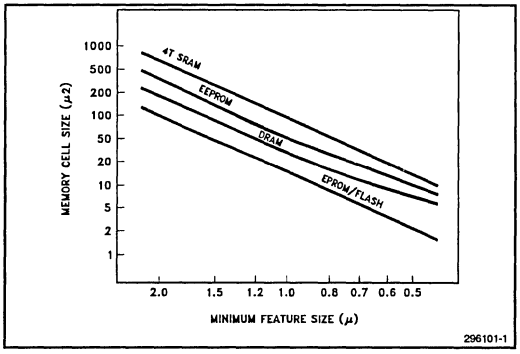


Figure 2

— EPROM (electrically programmable read-only memory) is a mature, high-density, nonvolatile technology which provides a degree of updatability not found in ROM. An OEM may program EPROM as needed to accommodate code changes or varying manufacturing unit quantities. Once programmed, however, the EPROM may only be erased by removing it from the system and then exposing the memory component to ultraviolet light—an impractical and time-consuming procedure form many OEMs and a virtually impossible task for end-users.

Unlike EPROM, flash memory is electrically re-writable within the host system, making it a much more flexible and easier to use alternative. Flash memory offers OEMs not only high density and nonvolatility, but higher functionality and the ability to differentiate their systems.

	Intel ETOX Flash	EEPROM
Transistors	1	2
Cell Size (1-Micro Lithography)	15 μ	38 μ
Cycling Features	0.1%	5%

Figure 3

— EEPROM (electrically erasable programmable read-only memory) is nonvolatile and electrically byte-erasable. Such byte-alterability is needed in certain applications but involves a more complex cell structure, and significant trade-offs in terms of limited density, lower reliability and higher cost, making it unsuitable as a mainstream memory.

Unlike EEPROM, Intel flash memory technology utilizes a one-transistor cell, allowing higher densities, scalability, lower cost, and higher reliability, while taking advantage of in-system, electrical erasability (Figure 3).

— DRAM (dynamic random access memory) is a volatile memory known for its density and low cost. Because of its volatility, however it requires not only a constant power supply to retain data, but also an archival storage technology, such as disk, to back it up.

Partnered with hard disks for permanent mass storage, DRAM technology has provided a low-cost, yet space and power-hungry solution for today's PCs.

With ETOX process technology, Intel manufactures a flash memory cell that is 30% smaller than equivalent DRAM cells. Flash memory's scalability offers a price advantage as well, keeping price parity with DRAM, and also becoming more attractive as a hard disk replacement in portable systems as densities grow and costs decline.

Intel flash memory combines advantages from each of these memory technologies. In embedded memory applications, flash memory provides higher-performance and more flexibility than ROM and EPROM, while providing higher density and better cost effectiveness than battery-backed SRAM and EEPROM. Moreover, the true nonvolatility and low power consumption characteristics of flash memory make it a compelling alternative to DRAM in many applications.

ETOX III TECHNOLOGY

Unlike other approaches to flash memory, Intel ETOX is a proven technology. As its name suggest, ETOX (or "EPROM tunnel oxide") technology evolved from EPROM. With 95% process compatibility with EPROM, ETOX taps experience gained from a mature high-volume manufacturing base pioneered by Intel in the 1970s.

Representing the third generation of Intel flash memory technology, the ETOX III 0.8 μ process delivers 100,000 write cycles per block. This capability significantly exceeds the cycling requirements of even the most demanding applications.

intel®

DD28F032SA, 32-Mbit FlashFile™ Memory Datasheet

The DD28F032SA contains three types of Status Registers to accomplish various functions:

- A Compatible Status Register (CSR) which is 100% compatible with the 28F008SA FlashFile memory's Status Register. This register, when used alone, provides a straightforward upgrade capability to the DD28F032SA from a 28F008SA-based design.
- A Global Status Register (GSR) which informs the system of command Queue status, Page Buffer status, and overall Write State Machine (WSM) status.
- 64 Block Status Registers (BSRs) which provide block-specific status information such as the block lock-bit status.

The GSR and BSR memory maps for Byte-Wide and Word-Wide modes are shown in Figures 4 and 5.

The DD28F032SA incorporates an open drain RY/BY# output pin. This feature allows the user to OR-tie many RY/BY# pins together in a multiple memory configuration such as a Resident Flash Array. Other configurations of the RY/BY# pin are enabled via special CUI commands and are described in detail in the 16-Mbit Flash Product Family User's Manual.

The DD28F032SA also incorporates three chip-enable input pins, CE_0#, CE_1# and CE_2#. The active low combination of CE_0# and CE_1# controls the first 28F016SA. The active low combination of CE_0# and CE_2# controls the second 28F016SA.

The BYTE# pin allows either x8 or x16 read/writes to the DD28F032SA. BYTE# at logic low selects 8-bit mode with address A_0 selecting between low byte and high byte. On the other hand, BYTE# at logic high enables 16-bit operation with address A_1 becoming the lowest order address and address A_0 is not used (don't care). A device block diagram is shown in Figure 1.

The DD28F032SA incorporates an Automatic Power Saving (APS) feature which substantially reduces the active current when the device is in static mode of operation (addresses not switching).

A deep power-down mode of operation is invoked when the RP# (called PWD# on the 28F008SA) pin is driven low. This mode provides additional write protection by acting as a device reset pin during power transitions. In the Deep Power-Down state, the WSM is reset (any current operation will abort) and the CSR, GSR and BSR registers are cleared.

A CMOS standby mode of operation is enabled when either CE_0#, or both CE_1# and CE_2#, transition high and RP# stays high with all input control pins at CMOS levels.

2.0 DEVICE PINOUT

The DD28F032SA Standard 56L-Dual Die TSOP Type I pinout configuration is shown in Figure 2.

2.1 Lead Descriptions

Symbol	Type	Name and Function
A_0	INPUT	**BYTE-SELECT ADDRESS:** Selects between high and low byte when device is in x8 mode. This address is latched in x8 Data Writes. Not used in x16 mode (i.e., the A_0 input buffer is turned off when BYTE# is high).
A_1 - A_{15}	INPUT	**WORD-SELECT ADDRESSES:** Select a word within one 64-Kbyte block. A_{6-15} selects 1 of 1024 rows, and A_{1-5} selects 16 of 512 columns. These addresses are latched during Data Writes.
A_{16} - A_{20}	INPUT	**BLOCK-SELECT ADDRESSES:** Select 1 of 64 Erase blocks. These addresses are latched during Data Writes, Erase and Lock-Block operations.
DQ_0 - DQ_7	INPUT/OUTPUT	**LOW-BYTE DATA BUS:** Inputs data and commands during CUI write cycles. Outputs array, buffer, identifier or status data in the appropriate read mode. Floated when the chip is de-selected or the outputs are disabled.
DQ_8 - DQ_{15}	INPUT/OUTPUT	**HIGH-BYTE DATA BUS:** Inputs data during x16 Data-Write operations. Outputs array, buffer or identifier data in the appropriate read mode; not used for Status register reads. Floated when the chip is de-selected or the outputs are disabled.
CE_0# CE_X# = CE_1# or CE_2#	INPUT	**CHIP ENABLE INPUTS:** Activate the device's control logic, input buffers, decoders and sense amplifiers. CE_0# or CE_1# enable/disable the first 28F016SA (16 Mbit No. 1) while CE_0#, CE_2# enable/disable the second 28F016SA (16 Mbit No. 2). CE_0# active low enables chip operation while CE_1# or CE_2# select between the first and second device, respectively. CE_1# or CE_2# must not be active low simultaneously. Reference Table 3.0.
RP#	INPUT	**RESET/POWER-DOWN:** RP# low places the device in a Deep Power-Down state. All circuits that burn static power, even those circuits enabled in standby mode, are turned off. When returning from Deep Power-Down, a recovery time of 400 ns is required to allow these circuits to power-up. When RP# goes low, any current or pending WSM operation(s) are terminated, and the device is reset. All Status registers return to ready (with all status flags cleared).
OE#	INPUT	**OUTPUT ENABLE:** Gates device data through the output buffers when low. The outputs float to tri-state off when OE# is high. **NOTE:** CE_X# overrides OE#, and OE# overrides WE#.
WE#	INPUT	**WRITE ENABLE:** Controls access to the CUI, Page Buffers, Data Queue Registers and Address Queue Latches. WE# is active low, and latches both address and data (command or array) on its rising edge.

6.4 DC Characteristics

V_{CC} = 3.3V ± 0.3V, T_A = 0°C to +70°C
3/5# = Pin Set High for 3.3V Operations

Symbol	Parameter	Notes	Min	Typ	Max	Units	Test Conditions
I_{IL}	Input Leakage Current	1			± 2	µA	$V_{CC} = V_{CC}$ Max, $V_{IN} = V_{CC}$ or GND
I_{LO}	Output Leakage Current	1			± 20	µA	$V_{CC} = V_{CC}$ Max, $V_{IN} = V_{CC}$ or GND
I_{CCS}	V_{CC} Standby Current	1,5,6		100	200	µA	$V_{CC} = V_{CC}$ Max, CE_0#, CE_X#, RP#, = V_{CC} ± 0.2V BYTE#, WP#, 3/5# = V_{CC} ± 0.2V or GND ± 0.2V
				2	8	mA	$V_{CC} = V_{CC}$ Max, CE_0#, CE_X#, RP# = V_{IH} BYTE#, WP#, 3/5# = V_{IH} or V_{IL}
I_{CCD}	V_{CC} Deep Power-Down Current	1		2	10	µA	RP# = GND ± 0.2V BYTE# = V_{CC} ± 0.2V or GND ± 0.2V
I_{CCR}1	V_{CC} Read Current	1,4,5, 6		25	30	mA	$V_{CC} = V_{CC}$ Max CMOS: CE_0#, CE_X# = GND ± 0.2V BYTE# = GND ± 0.2V or V_{CC} ± 0.2V Inputs = GND ± 0.2V or V_{CC} ± 0.2V f = 6.67 MHz, I_{OUT} = 0 mA
				26	34	mA	TTL: CE_0#, CE_X# = V_{IL}, BYTE# = V_{IL} or V_{IH} INPUTS = V_{IL} or V_{IH}, f = 6.67 MHz, I_{OUT} = 0 mA
I_{CCW}	V_{CC} Write Current	1,7		8	12	mA	Word/Byte Write in Progress

6.11 Erase and Word/Byte Write Performance, Cycling Performance and Suspend Latency(1,3)

V_{CC} = 3.3V ± 0.3V, V_{PP} = 12.0V ± 0.6V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ(1)	Max	Units	Test Conditions
	Page Buffer Byte Write Time	2,4		3,26	Note 6	µs	
	Page Buffer Word Write Time	2,4		6.53	Note 6	µs	
$t_{WHRH}1$	Word/Byte Write Time	2		9	Note 6	µs	
$t_{WHRH}2$	Block Write Time	2		0.6	2.1	Sec	Byte Write Mode
$t_{WHRH}3$	Block Write Time	2		0.3	1.0	Sec	Word Write Mode
	Block Erase Time	2		0.8	10	Sec	
	Full Chip Erase Time	2		51.2		Sec	
	Erase Suspend Latency Time to Read			7.0		µs	
	Auto Erase Suspend Latency Time to Write			10.0		µs	
	Erase Cycles	5	100,000	1,000,000		Cycles	

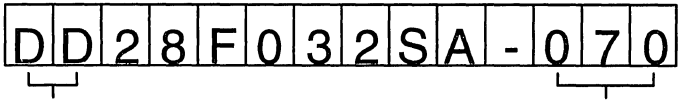
V_{CC} = 5.0V ± 0.5V, V_{PP} = 12.0V ± 0.6V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ(1)	Max	Units	Test Conditions
	Page Buffer Byte Write Time	2,4		2.76	Note 6	µs	
	Page Buffer Word Write Time	2,4		5.51	Note 6	µs	
$t_{WHRH}1$	Word Byte/Write Time	2		6	Note 6	µs	
$t_{WHRH}2$	Block Write Time	2		0.4	2.1	Sec	Byte Write Mode
$t_{WHRH}3$	Block Write Time	2		0.2	1.0	Sec	Word Write Mode
	Block Erase Time	2		0.6	10	Sec	
	Full Chip Erase Time	2		38.4		Sec	
	Erase Suspend Latency Time to Read			5.0		µs	
	Auto Erase Suspend Latency Time to Write			8.0		µs	
	Erase Cycles	5	100,000	1,000,000		Cycles	

NOTES:

1. 25°C, V_{CC} = 3.3V or 5.0V nominal, 10K cycles.
2. Excludes system-level overhead.
3. These performance numbers are valid for all speed versions.
4. This assumes using the full Page Buffer to Write to Flash (256 bytes or 128 words).
5. 1,000,000 cycle performance assumes the application uses block retirement techniques.
6. This information will be available in a technical paper. Please call Intel's Application hotline or your local sales office for more information.

DEVICE NOMENCLATURE/ORDERING INFORMATION



NOTES:
Two valid combinations of speeds exist:
DD28F032SA-070, DD28F032SA-080, DD28F032SA-120
or
DD28F032SA-100, DD28F032SA-150

Option	Order Code	Valid Combinations		
		V_{CC} = 3.3V ± 0.3V, 50 pF	V_{CC} = 5.0V ± 5%, 30 pF	V_{CC} = 5.0V ± 10%, 100 pF
1	DD28F032SA 070	DD28F032SA-150	DD28F032SA-070	DD28F032SA-080
2	DD28F032SA 100	DD28F032SA-150		DD28F032SA-100

ADDITIONAL INFORMATION

Order Number	Item
297372	16-Mbit Flash Product Family User's Manual
290489	28F016SA 16-Mbit FlashFile™ Memory Data Sheet
290528	28F016SV FlashFile™ Memory Data Sheet
290429	28F008SA Data Sheet
292159	AP-607 "Multi-Site Layout Planning w/Intel's FlashFile™ Components Including ROM Compatibility"
292144	AP 393 "28F016SV Compatibility with 28F016SA"
292127	AP 378 "System Optimization Using the Enhanced Features of the 28F016SA"
292126	AP 377 "28F016SA Software Drivers"
292124	AP 375 "Upgrade Considerations from the 28F008SA to the 28F016SA"
292123	AP 374 "Flash Memory Write Protection Techniques"
292092	AP 357 "Power Supply Solutions For Flash Memory"
294016	ER 33 "ETOX™ Flash Memory Technology—Insight to Intel's Fourth Generation Process Innovation"
297508	FLASHBuilder Design Resource Tool
297534	Small and Low-Cost Power Supply Solution for Intel's Flash Memory Products (Technical Paper)

Please check with Intel Literature for availability.

DATASHEET REVISION HISTORY

Number	Description
001	Original Version
002	Never Published
003	Full Datasheet with Specifications CE_0#, CE_1# control 28F016SA No. 1 CE_0#, CE_2# control 28F016SA No. 2
004	Lead Descriptions: Block Select Addresses: Select 1 of 64 Erase Blocks DC Characteristics (3.3V V_{CC}): I_{CCR}[1] (TTL): BYTE# = V_{IL} or V_{IH} Full Chip Erase Time (3.3V V_{CC}) = 51.2 sec typ Full Chip Erase Time (5.0V V_{CC}) = 38.4 sec typ Minor cosmetic changes

intel®

28F016SV, 16-Mbit FlashFile™ Memory Datasheet

2.1 Lead Descriptions (Continued)

Symbol	Type	Name and Function
RY/BY#	OPEN DRAIN OUTPUT	**READY/BUSY:** Indicates status of the internal WSM. When low, it indicates that the WSM is busy performing an operation. RY/BY# floating indicates that the WSM is ready for new operations (or WSM has completed all pending operations), or Erase is Suspended, or the device is in deep power-down mode. This output is always active (i.e., not floated to tri-state off when OE# or CE_0#, CE_1# are high), except if a RY/BY# Pin Disable command is issued.
WP#	INPUT	**WRITE PROTECT:** Erase blocks can be locked by writing a nonvolatile lock-bit for each block. When WP# is low, those locked blocks as reflected by the Block-Lock Status bits (BSR.6), are protected from inadvertent Data Writes or Erases. When WP# is high, all blocks can be written or erased regardless of the state of the lock-bits. The WP# input buffer is disabled when RP# transitions low (deep power-down mode).
BYTE#	INPUT	**BYTE ENABLE:** BYTE# low places device in x8 mode. All data is then input or output on DQ_{0-7}, and DQ_{8-15} float. Address A_0 selects between the high and low byte. BYTE# high places the device in x16 mode, and turns off the A_0 input buffer. Address A_1, then becomes the lowest order address.
V_{PP}	SUPPLY	**WRITE/ERASE POWER SUPPLY (12.0V ± 0.6V, 5.0V ± 0.5V) :** For erasing memory array blocks or writing words/bytes/pages into the flash array. V_{PP} = 5.0V ± 0.5V eliminates the need for a 12V converter, while connection to 12.0V ± 0.6V maximizes Write/Erase Performance. **NOTE:** Successful completion of write and erase attempts is inhibited with V_{PP} at or below 1.5V. Write and Erase attempts with V_{PP} between 1.5V and 4.5V, between 5.5V and 11.4V, and above 12.6V produce spurious results and should not be attempted.
V_{CC}	SUPPLY	**DEVICE POWER SUPPLY (3.3V ± 0.3V, 5.0V ± 0.5V, 5.0 ± 0.25V):** Internal detection configures the device for 3.3V or 5.0V operation. To switch 3.3V to 5.0V (or vice versa), first ramp V_{CC} down to GND, and then power to the new V_{CC} voltage. Do not leave any power pins floating.
GND	SUPPLY	**GROUND FOR ALL INTERNAL CIRCUITRY:** Do not leave any ground pins floating.
NC		**NO CONNECT:** Lead may be driven or left floating.

4.4 28F016SV—Performance Enhancement Command Bus Definitions

Command	Mode	Notes	First Bus Cycle			Second Bus Cycle			Third Bus Cycle		
			Oper	Addr	Data	Oper	Addr	Data	Oper	Addr	Data
Read Extended Status Register		1	Write	X	xx71H	Read	RA	GSRD BSRD			
Page Buffer Swap		7	Write	X	xx72H						
Read Page Buffer			Write	X	xx75H	Read	PA	PD			
Single Load to Page Buffer			Write	X	xx74H	Write	PA	PD			
Sequential Load to Page Buffer	x8	4,6,10	Write	X	xxE0H	Write	X	BCL	Write	X	BCH
	x16	4,5,6,10	Write	X	xxE0H	Write	X	WCL	Write	X	WCH
Page Buffer Write to Flash	x8	3,4,9,10	Write	X	xx0CH	Write	A_0	BC(L,H)	Write	WA	BC(H,L)
	x16	4,5,10	Write	X	xx0CH	Write	X	WCL	Write	WA	WCH
Two-Byte Write	x8	3	Write	X	xxFBH	Write	A_0	WD(L,H)	Write	WA	WD(H,L)
Lock Block/Confirm			Write	X	xx77H	Write	BA	xxD0H			
Upload Status Bits/Confirm		2	Write	X	xx97H	Write	X	xxD0H			
Upload Device Information/Confirm		11	Write	X	xx99H	Write	X	xxD0H			
Erase All Unlocked Blocks/Confirm			Write	X	xxA7H	Write	X	xxD0H			
RY/BY# Enable to Level-Mode		8	Write	X	xx96H	Write	X	xx01H			
RY/BY# Pulse-On-Write		8	Write	X	xx96H	Write	X	xx02H			
RY/BY# Pulse-On-Erase		8	Write	X	xx96H	Write	X	xx03H			
RY/BY# Disable		8	Write	X	xx96H	Write	X	xx04H			
RY/BY# Pulse-On-Write/Erase		8	Write	X	xx96H	Write	X	xx05H			
Sleep	12		Write	X	xxF0H						
Abort			Write	X	xx80H						

ADDRESS
BA = Block Address
PA = Page Buffer Address
RA = Extended Register Address
WA = Write Address
X = Don't Care

DATA
AD = Array Data
PD = Page Buffer Data
BSRD = BSR Data
GSRD = GSR Data

WC (L,H) = Word Count (Low, High)
BC (L,H) = Byte Count (Low, High)
WD (L,H) = Write Data (Low, High)

NOTES:

1. RA can be the GSR address or any BSR address. See Figures 4 and 5 for Extended Status Register memory maps.
2. Upon device power-up, all BSR lock-bits come up locked. The Upload Status Bits command must be written to reflect the actual lock-bit status.
3. A_0 is automatically complemented to load second byte of data. BYTE# must be at V_{IL}.
   A_0 value determines which WD/BC is supplied first: A_0 = 0 looks at the WDL/BCL, A_0 = 1 looks at the WDH/BCH.
4. BCH/WCH must be at 00H for this product because of the 256-byte (128-word) Page Buffer size, and to avoid writing the Page Buffer contents to more than one 256-byte segment within an array block. They are simply shown for future Page Buffer expandability.
5. In x16 mode, only the lower byte DQ_{0-7} is used for WCL and WCH. The upper byte DQ_{8-15} is a don't care.
6. PA and PD (whose count is given in cycles 2 and 3) are supplied starting in the fourth cycle, which is not shown.
7. This command allows the user to swap between available Page Buffers (0 or 1).
8. These commands reconfigure RY/BY# output to one of three pulse-modes or enable and disable the RY/BY# function.
9. Write address, WA, is the Destination address in the flash array which must match the Source address in the Page Buffer. Refer to the 16-Mbit Flash Product Family User's Manual.
10. BCL = 00H corresponds to a byte count of 1. Similarly, WCL = 00H corresponds to a word count of 1.
11. After writing the Upload Device Information command and the Confirm command, the following information is output at Page Buffer addresses specified below:

Address	Information
06H, 07H (Byte Mode)	Device Revision Number
03H (Word Mode)	Device Revision Number
1EH (Byte Mode)	Device Configuration Code
0FH (DQ_{0-7})(Word Mode)	Device Configuration Code
1FH (Byte Mode)	Device Proliferation Code (01H)
0FH (DQ_{8-15})(Word Mode)	Device Proliferation Code (01H)

    A page buffer swap followed by a page buffer read sequence is necessary to access this information. The contents of all other Page Buffer locations, after the Upload Device Information command is written, are reserved for future implementation by Intel Corporation. See Section 4.8 for a description of the Device Configuration Code. This code also corresponds to data written to the 28F016SV after writing the RY/BY# Reconfiguration command.
12. To ensure that the 28F016SV's power consuption during Sleep Mode reaches the Deep Power-Down current level, the system also needs to de-select the chip by taking either or both CE_0# or CE_1# high.
13. The upper byte of the data bus (D_{8-15}) during command writes is a "Don't Care" in x16 operation of the device.

5.0 ELECTRICAL SPECIFICATIONS

5.1 Absolute Maximum Ratings*

Temperature Under Bias..... 0°C to +80°C

Storage Temperature.....–65°C to +125°C

NOTICE: This datasheet contains information on products in the sampling and initial production phases of development. The specifications are subject to change without notice. Verify with your local Intel Sales office that you have the latest datasheet before finalizing a design.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

V_{CC} = 3.3V ± 0.3V Systems

Sym	Parameter	Notes	Min	Max	Units	Test Conditions
T_A	Operating Temperature, Commercial	1	0	70	°C	Ambient Temperature
V_{CC}	V_{CC} with Respect to GND	2	-0.2	7.0	V	
V_{PP}	V_{PP} Supply Voltage with Respect to GND	2,3	-0.2	14.0	V	
V	Voltage on any Pin (except V_{CC},V_{PP}) with Respect to GND	2,5	-0.5	V_{CC} + 0.5	V	
I	Current into any Non-Supply Pin	5		± 30	mA	
I_{OUT}	Output Short Circuit Current	4		100	mA	

V_{CC} = 5.0V ± 0.5V, 5.0V ± 0.25V Systems(6)

Sym	Parameter	Notes	Min	Max	Units	Test Conditions
T_A	Operating Temperature, Commercial	1	0	70	°C	Ambient Temperature
V_{CC}	V_{CC} with Respect to GND	2	-0.2	7.0	V	
V_{PP}	V_{PP} Supply Voltage with Respect to GND	2,3	-0.2	14.0	V	
V	Voltage on any Pin (except V_{CC},V_{PP}) with Respect to GND	2,5	-2.0	7.0	V	
I	Current into any Non-Supply Pin	5		± 30	mA	
I_{OUT}	Output Short Circuit Current	4		100	mA	

NOTES:

1. Operating temperature is for commercial product defined by this specification.
2. Minimum DC voltage is -0.5V on input/output pins. During transitions, this level may undershoot to -2.0V for periods <20 ns. Maximum DC voltage on input/output pins is V_{CC} + 0.5V which, during transitions, may overshoot to V_{CC} + 2.0V for periods <20 ns.
3. Maximum DC voltage on V_{PP} may overshoot to +14.0V for periods <20 ns.
4. Output shorted for no more than one second. No more than one output shorted at a time.
5. This specification also applies to pins marked "NC."
6. 5% V_{CC} specifications refer to the 28F016SV-065 and 28F016SV-070 in its high speed test configuration.

5.3 DC Characteristics (Continued)

V_{CC} = 3.3V ± 0.3V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ	Max	Units	Test Conditions
I_{CCW}	V_{CC} Write Current	1,6		8	12	mA	Word/Byte Write in Progress V_{PP} = 12.0V ± 5%
				8	17	mA	Word/Byte Write in Progress V_{PP} = 5.0V ± 10%
I_{CCE}	V_{CC} Block Erase Current	1,6		6	12	mA	Block Erase in Progress V_{PP} = 12.0V ± 5%
				9	17	mA	Block Erase in Progress V_{PP} = 5.0V ± 10%
I_{CCES}	V_{CC} Erase Suspend Current	1,2		1	4	mA	CE_0#, CE_1# = V_{IH} Block Erase Suspended
I_{PPS}	V_{PP} Standby/Read	1		± 1	± 10	µA	$V_{PP} \leq V_{CC}$
I_{PPR}	Current			30	200	µA	$V_{PP} > V_{CC}$
I_{PPD}	V_{PP} Deep Power-Down Current	1		0.2	5	µA	RP# = GND ± 0.2V
I_{PPW}	V_{PP} Write Current	1,6		10	15	mA	V_{PP} = 12.0V ± 5% Word/Byte Write in Progress
				15	25	mA	V_{PP} = 5.0V ± 10% Word/Byte Write in Progress
I_{PPE}	V_{PP} Erase Current	1,6		4	10	mA	V_{PP} = 12.0V ± 5% Block Erase in Progress
				14	20	mA	V_{PP} = 5.0V ± 10% Block Erase in Progress
I_{PPES}	V_{PP} Erase Suspend Current	1		30	50	µA	V_{PP} = V_{PPH1} or V_{PPH2}, Block Erase Suspended
V_{IL}	Input Low Voltage	6	-0.3		0.8	V	
V_{IH}	Input High Voltage	6	2.0		V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage	6			0.4	V	V_{CC} = V_{CC} Min and I_{OL} = 4 mA

5.4 DC Characteristics (Continued)

V_{CC} = 5.0V ± 0.5V, 5.0V ± 0.25V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ	Max	Units	Test Conditions
I_{CCW}	V_{CC} Write Current	1,6		25	35	mA	Word/Byte in Progress V_{PP} = 12.0V ± 5%
				25	40	mA	Word/Byte in Progress V_{PP} = 5.0V ± 10%
I_{CCE}	V_{CC} Block Erase Current	1,6		18	25	mA	Block Erase in Progress V_{PP} = 12.0V ± 5%
				20	30	mA	Block Erase in Progress V_{PP} = 5.0V ± 10%
I_{CCES}	V_{CC} Erase Suspend Current	1,2		2	4	mA	CE_0#, CE_1# = V_{IH} Block Erase Suspended
I_{PPS}	V_{PP} Standby/Read	1		± 1	± 10	µA	$V_{PP} \leq V_{CC}$
I_{PPR}	Current			30	200	µA	$V_{PP} > V_{CC}$
I_{PPD}	V_{PP} Deep Power-Down Current	1		0.2	5	µA	RP# = GND ± 0.2V
I_{PPW}	V_{PP} Write Current	1,6		7	12	mA	V_{PP} = 12.0V ± 5% Word/Byte Write in Progress
				17	22	mA	V_{PP} = 5.0V ± 10% Word/Byte Write in Progress
I_{PPE}	V_{PP} Block Erase Current	1,6		5	10	mA	V_{PP} = 12.0V ± 5% Block Erase in Progress
				16	20	mA	V_{PP} = 5.0V ± 10% Block Erase in Progress
I_{PPES}	V_{PP} Erase Suspend Current	1		30	50	µA	V_{PP} = V_{PPH1} or V_{PPH2}, Block Erase Suspended
V_{IL}	Input Low Voltage	6	-0.5		0.8	V	
V_{IH}	Input High Voltage	6	2.0		V_{CC} +0.5	V	

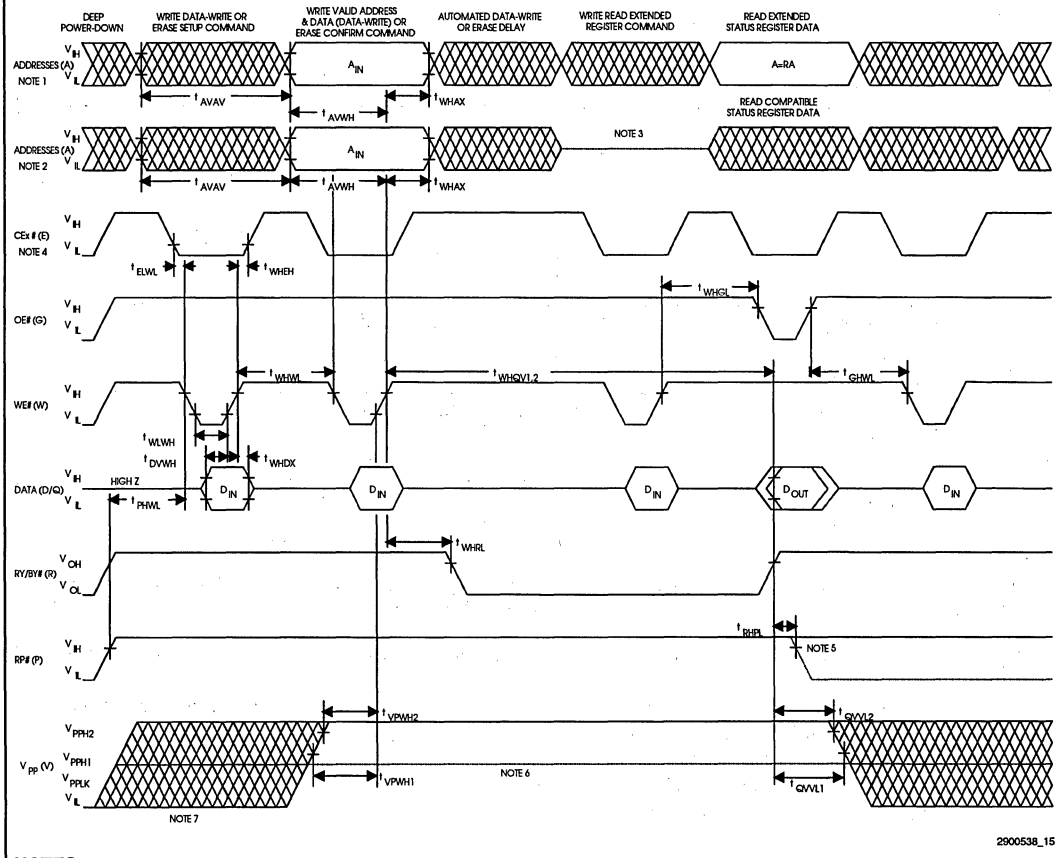
5.4 DC Characteristics (Continued)

V_{CC} = 5.0V ± 0.5V, 5.0V ± 0.25V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ	Max	Units	Test Conditions
V_{OL}	Output Low Voltage	6			0.45	V	$V_{CC} = V_{CC}$ Min I_{OL} = 5.8 mA
$V_{OH}1$	Output High Voltage	6	0.85 V_{CC}			V	I_{OH} = -2.5 mA $V_{CC} = V_{CC}$ Min
$V_{OH}2$		6	V_{CC} -0.4				I_{OH} = -100 µA $V_{CC} = V_{CC}$ Min
V_{PPLK}	V_{PP} Write/Erase Lock Voltage	3,6	0.0		1.5	V	
V_{PPH1}	V_{PP} during Write/Erase Operations		4.5	5.0	5.5	V	
V_{PPH2}	V_{PP} during Write/Erase Operations		11.4	12.0	12.6	V	
V_{LKO}	V_{CC} Write/Erase Lock Voltage		2.0			V	

NOTES:

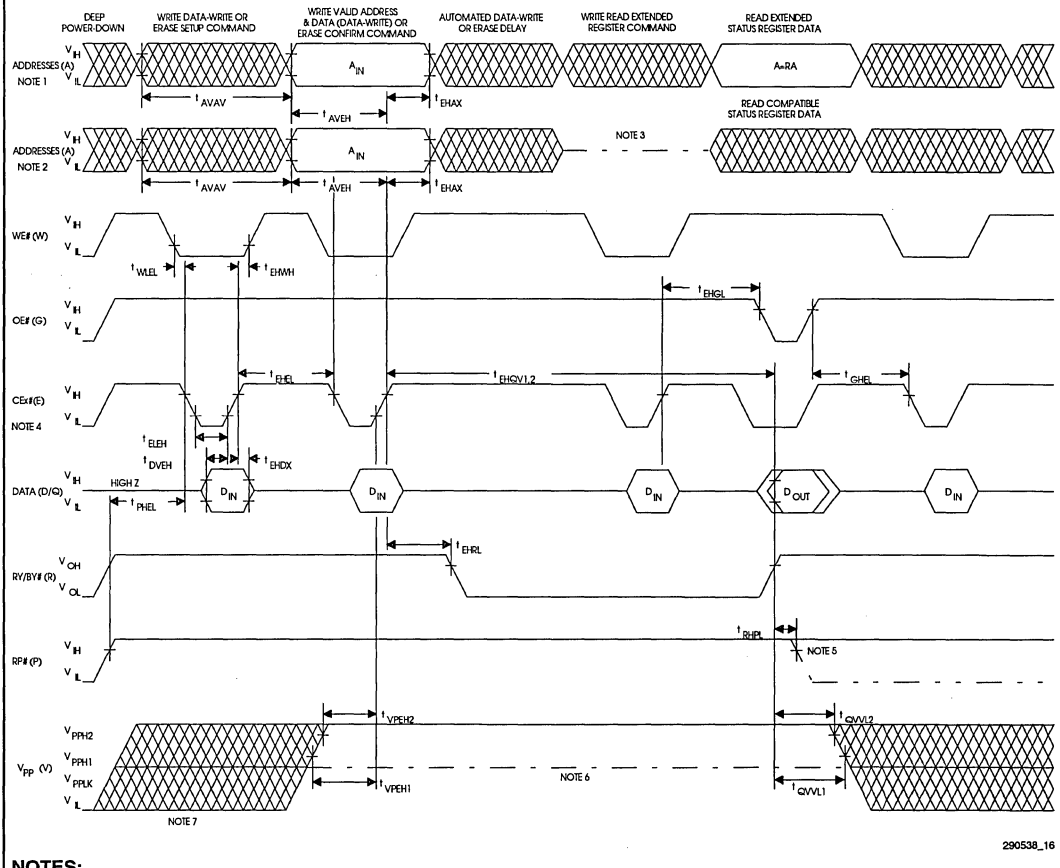
1. All currents are in RMS unless otherwise noted. Typical values at V_{CC} = 5.0V, V_{PP} = 12.0V or 5.0V, T = 25°C. These currents are valid for all product versions (package and speeds) and are specified for a CMOS rise/fall time (10% to 90%) of <5 ns and a TTL rise/fall time of <10 ns.
2. I_{CCES} is specified with the device de-selected. If the device is read while in erase suspend mode, current draw is the sum of I_{CCES} and I_{CCR}.
3. Block Erases, Word/Byte Writes and Lock Block operations are inhibited when $V_{PP} \leq V_{PPLK}$ and not guaranteed in the ranges between V_{PPLK}(max) and V_{PPH1}(min), between V_{PPH1} (max) and V_{PPH2}(min) and above V_{PPH2}(max).
4. Automatic Power Saving (APS) reduces I_{CCR} to 1 mA typical in Static operation.
5. CMOS Inputs are either V_{CC} ± 0.2V or GND ± 0.2V. TTL Inputs are either V_{IL} or V_{IH}.
6. Sampled, not 100% tested. Guaranteed by design.



NOTES:

1. This address string depicts Data Write/Erase cycles with corresponding verification via ESRD.
2. This address string depicts Data Write/Erase cycles with corresponding verification via CSRD.
3. This cycle is invalid when using CSRD for verification during Data Write/Erase operations.
4. CEx# is defined as the latter of CE_0# or CE_1# going low or the first of CE_0# or CE_1# going high.
5. RP# low transition is only to show t_{RHPL}; not valid for above Read and Write cycles.
6. V_{PP} voltage during Write/Erase operations valid at both 12.0V and 5.0V.
7. V_{PP} voltage equal to or below V_{PPLK} provides complete flash memory array protection.

Figure 15. AC Waveforms for Command Write Operations



NOTES:

1. This address string depicts Data Write/Erase cycles with corresponding verification via ESRD.
2. This address string depicts Data Write/Erase cycles with corresponding verification via CSRD.
3. This cycle is invalid when using CSRD for verification during Data Write/Erase operations.
4. CEx# is defined as the latter of CE_0# or CE_1# going low or the first of CE_0# or CE_1# going high.
5. RP# low transition is only to show t_{RHPL}; not valid for above Read and Write cycles.
6. V_{PP} voltage during Write/Erase operations valid at both 12.0V and 5.0V.
7. V_{PP} voltage equal to or below V_{PPLK} provides complete flash memory array protection.

Figure 16. Alternate AC Waveforms for Command Write Operations

5.12 Erase and Word/Byte Write Performance[3,5]

V_{CC} = 3.3V ± 0.3V, V_{PP} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ[1]	Max	Units	Test Conditions
	Page Buffer Byte Write Time	2,6	TBD	8.0	TBD	µs	
	Page Buffer Word Write Time	2,6	TBD	16.0	TBD	µs	
$t_{WHRH}1A$	Byte Write Time	2	TBD	29.0	TBD	µs	
$t_{WHRH}1B$	Word Write Time	2	TBD	35.0	TBD	µs	
$t_{WHRH}2$	Block Write Time	2	TBD	1.9	TBD	sec	Byte Write Mode
$t_{WHRH}3$	Block Write Time	2	TBD	1.2	TBD	sec	Word Write Mode
	Block Erase Time	2	TBD	1.4	TBD	sec	
	Full Chip Erase Time	2	TBD	44.8	TBD	sec	
	Erase Suspend Latency Time to Read	4	1.0	12	75	µs	
	Auto Erase Suspend Latency Time to Write		4.0	15	80	µs	

V_{CC} = 3.3V ± 0.3V, V_{PP} = 12.0V ± 0.6V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ[1]	Max	Units	Test Conditions
	Page Buffer Byte Write Time	2,6	TBD	2.2	TBD	µs	
	Page Buffer Word Write Time	2,6	TBD	4.4	TBD	µs	
$t_{WHRH}1$	Word/Byte Write Time	2	5	9	TBD	µs	
$t_{WHRH}2$	Block Write Time	2	TBD	0.6	2.1	sec	Byte Write Mode
$t_{WHRH}3$	Block Write Time	2	TBD	0.3	1.0	sec	Word Write Mode
	Block Erase Time	2	0.3	0.8	10	sec	
	Full Chip Erase Time	2	TBD	25.6	TBD	sec	
	Erase Suspend Latency Time to Read	4	1.0	9	55	µs	
	Auto Erase Suspend Latency Time to Write		4.0	12	60	µs	

5.12 Erase and Word/Byte Write Performance[3,5] (Continued)

V_{CC} = 5.0V ± 0.5V, 5.0V ± 0.25V, V_{PP} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ[1]	Max	Units	Test Conditions
	Page Buffer Byte Write Time	2,6	TBD	8.0	TBD	µs	
	Page Buffer Word Write Time	2,6	TBD	16.0	TBD	µs	
$t_{WHRH}1A$	Byte Write Time	2	TBD	20	TBD	µs	
$t_{WHRH}1B$	Word Write Time	2	TBD	25	TBD	µs	
$t_{WHRH}2$	Block Write Time	2	TBD	1.4	TBD	sec	Byte Write Mode
$t_{WHRH}3$	Block Write Time	2	TBD	0.85	TBD	sec	Word Write Mode
	Block Erase Time	2	TBD	1.0	TBD	sec	
	Full Chip Erase Time	2	TBD	32.0	TBD	sec	
	Erase Suspend Latency Time to Read	4	1.0	9	55	µs	
	Auto Erase Suspend Latency Time to Write		3.0	12	60	µs	

V_{CC} = 5.0V ± 0.5V, 5.0V ± 0.25V, V_{PP} = 12.0V ± 0.6V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ[1]	Max	Units	Test Conditions
	Page Buffer Byte Write Time	2,6	TBD	2.1	TBD	µs	
	Page Buffer Word Write Time	2,6	TBD	4.1	TBD	µs	
$t_{WHRH}1$	Word/Byte Write Time	2	4.5	6	TBD	µs	
$t_{WHRH}2$	Block Write Time	2	TBD	0.4	2.1	sec	Byte Write Mode
$t_{WHRH}3$	Block Write Time	2	TBD	0.2	1.0	sec	Word Write Mode
	Block Erase Time	2	0.3	0.6	10	sec	
	Full Chip Erase Time	2	TBD	19.2	TBD	sec	
	Erase Suspend Latency Time to Read	4	1.0	7	40	µs	
	Auto Erase Suspend Latency Time to Write		3.0	10	45	µs	

NOTES:

1. 25°C, and nominal voltages.
2. Excludes system-level overhead.
3. These performance numbers are valid for all speed versions.
4. Specification applies to interrupt latency for Single Block Erase. Suspend latency for Erase All Unlocked Block operation extends the maximum latency time to 270 µs.
5. Sampled, but not 100% tested. Guaranteed by design.
6. Assumes using the full Page Buffer to write to flash (256 Bytes or 128 Words).

intel®

28F016SA, 16-Mbit FlashFile™ Memory Datasheet

The 28F016SA contains three types of Status Registers to accomplish various functions:

- A Compatible Status Register (CSR) which is 100% compatible with the 28F008SA FlashFile memory's Status Register. This register, when used alone, provides a straightforward upgrade capability to the 28F016SA from a 28F008SA-based design.
- A Global Status Register (GSR) which informs the system of command Queue status, Page Buffer status, and overall Write State Machine (WSM) status.
- 32 Block Status Registers (BSRs) which provide block-specific status information such as the block lock-bit status.

The GSR and BSR memory maps for Byte-Wide and Word-Wide modes are shown in Figures 5 and 6.

The 28F016SA incorporates an open drain RY/BY# output pin. This feature allows the user to OR-tie many RY/BY# pins together in a multiple memory configuration such as a Resident Flash Array.

Other configurations of the RY/BY# pin are enabled via special CUI commands and are described in detail in the 16-Mbit Flash Product Family User's Manual.

The 28F016SA also incorporates a dual chip-enable function with two input pins, CE_0# and CE_1#. These pins have exactly the same functionality as the regular chip-enable pin CE# on the 28F008SA. For minimum chip designs, CE_1# may be tied to ground and use CE_0# as the chip enable input. The 28F016SA uses the logical combination of these two signals to enable or disable the entire chip. Both CE_0# and CE_1# must be active low to enable the device and, if either one becomes inactive, the chip will be disabled. This feature, along with the open drain RY/BY# pin, allows the system designer to reduce the number of control pins used in a large array of 16-Mbit devices.

The BYTE# pin allows either x8 or x16 read/writes to the 28F016SA. BYTE# at logic low selects 8-bit mode with address A_0 selecting between low byte and high byte. On the other hand, BYTE# at logic high enables 16-bit operation with address A_1 becoming the lowest order address and address A_0 is not used (don't care). A device block diagram is shown in Figure 1.

The 28F016SA is specified for a maximum access time of 70 ns (t_{ACC}) at 5.0V operation (4.75V to 5.25V) over the commercial temperature range (0° C to +70°C). A corresponding maximum access time of 120 ns at 3.3V (3.0V to 3.6V and 0°C to +70°C) is achieved for reduced power consumption applications.

The 28F016SA incorporates an Automatic Power Saving (APS) feature which substantially reduces the active current when the device is in static mode of operation (addresses not switching).

In APS mode, the typical I_{CC} current is 1 mA at 5.0V (0.8 mA at 3.3V).

A deep power-down mode of operation is invoked when the RP# (called $\overline{PWD}$ on the 28F008SA) pin transitions low. This mode brings the device power consumption to less than 1.0 µA, typically, and provides additional write protection by acting as a device reset pin during power transitions. A reset time of 400 ns is required from RP# switching high until outputs are again valid. In the Deep Power-Down state, the WSM is reset (any current operation will abort) and the CSR, GSR and BSR registers are cleared.

A CMOS standby mode of operation is enabled when either CE_0# or CE_1# transitions high and RP# stays high with all input control pins at CMOS levels. In this mode, the device typically draws an I_{CC} standby current of 50 µA.

2.0 DEVICE PINOUT

The 28F016SA 56 lead TSOP Type I pinout configuration is shown in Figure 2. The 56 lead SSOP pinout configuration is shown in Figure 3.

5.11 Erase and Word/Byte Write Performance, Cycling Performance and Suspend Latency[1,3]

V_{CC} = 3.3V ± 0.3V, V_{PP} = 12.0V ± 0.6V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ	Max	Units	Test Conditions
	Page Buffer Byte Write Time	1,2,4		3.26	Note 6	µs	
	Page Buffer Word Write Time	1,2,4		6.53	Note 6	µs	
$t_{WHRH}1$	Word/Byte Write Time	1,2		9	Note 6	µs	
$t_{WHRH}2$	Block Write Time	1,2		0.6	2.1	Sec	Byte Write Mode
$t_{WHRH}3$	Block Write Time	1,2		0.3	1.0	Sec	Word Write Mode
	Block Erase Time	1,2		0.8	10	Sec	
	Full Chip Erase Time	1,2		25.6		Sec	
	Erase Suspend Latency Time to Read			7.0		µs	
	Auto Erase Suspend Latency Time to Write			10.0		µs	
	Erase Cycles	5	100,000	1,000,000		Cycles	

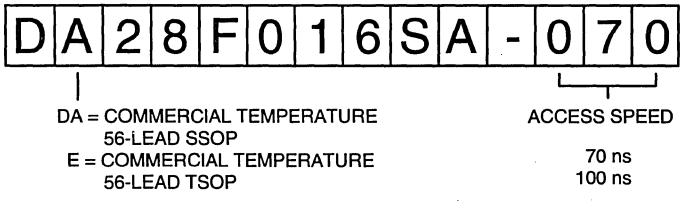
V_{CC} = 5.0V ± 0.5V, V_{PP} = 12.0V ± 0.6V, T_A = 0°C to +70°C

Sym	Parameter	Notes	Min	Typ	Max	Units	Test Conditions
	Page Buffer Byte Write Time	1,2,4		2.76	Note 6	µs	
	Page Buffer Word Write Time	1,2,4		5.51	Note 6	µs	
$t_{WHRH}1$	Word Byte/Write Time	1,2		6	Note 6	µs	
$t_{WHRH}2$	Block Write Time	1,2		0.4	2.1	Sec	Byte Write Mode
$t_{WHRH}3$	Block Write Time	1,2		0.2	1.0	Sec	Word Write Mode
	Block Erase Time	1,2		0.6	10	Sec	
	Full Chip Erase Time	1,2		19.2		Sec	
	Erase Suspend Latency Time to Read			5.0		µs	
	Auto Erase Suspend Latency Time to Write			8.0		µs	
	Erase Cycles	5	100,000	1,000,000		Cycles	

NOTES:

1. 25°C, V_{PP} = 12.0V nominal, 10K cycles.
2. Excludes system-level overhead.
3. These performance numbers are valid for all speed versions.
4. This assumes using the full Page Buffer to Write Flash (256 bytes or 128 words).
5. Typical 1,000,000 cycle performance assumes the application uses block retirement techniques.
6. This information will be available in a technical paper. Please call Intel's Application Hotline or your local Intel Sales office for more information.

9.0 DEVICE NOMENCLATURE AND ORDERING INFORMATION



Option	Order Code	Valid Combinations		
		V_{CC} = 3.3V ± 0.3V, 50 pF Load	V_{CC} = 5.0V ± 10%, 100 pF Load	V_{CC} = 5.0V ± 5%, 30 pF Load
1	E28F016SA-070	E28F016SA-120	E28F016SA-080	E28F016SA-070
2	E28F016SA-100	E28F016SA-150	E28F016SA-100	
3	DA28F016SA-070	DA28F016SA-120	DA28F016SA-080	DA28F016SA-070
4	DA28F016SA-100	DA28F016SA-150	DA28F016SA-100	

9.1 References

Order Number	Document/Tool
297372	16-Mbit Flash Product Family User's Manuall
290490	DD28F032SA 32-Mbit FlashFile™ Memory Datasheet
290528	28F016SV FlashFile™ Memory Datasheet
290435	28F008SA 8-Mbit FlashFile™ Memory Datasheet
292159	AP-607 Multi-Site Layout Planning with Intel's Flash File™ Components
292144	AP-393 28F016SV Compatibility with 28F016SA
292127	AP-378 System Optimization Using the Enhanced Features of the 28F016SA
292126	AP-377 28F016SA Software Drivers
292124	AP-375 Upgrade Considerations from the 28F008SA to the 28F016SA
292123	AP-374 Flash Memory Write Protection Techniques
292092	AP-357 Power Supply Solutions for Flash Memory

Order Number	Document/Tool
294016	ER-33 ETOX™ Flash Memory Technology - Insight to Intel's Fourth Generation Process Innovation
297534	Small and Low-Cost Power Supply solution for Intel's Flash Memory Products (Technical Paper)
297508	FLASHBuilder Design Resource Tool

9.2 Revision History

Number	Description
001	Original Version
002	— Added 56 Lead SSOP Package —Separated AC Reading Timing Specs t_{AVEL}, t_{AVGL} for Extended Status Register Reads — Modified DEVICE NOMENCLATURE — Added ORDERING INFORMATION — Added Page Buffer Typical Write Performance numbers — Added Typical Erase Suspend Latencies — For I_{CCD} (Deep Power-Down current, BYTE# must be at CMOS levels) — Added SSOP package mechanical specifications
003	— Renamed referenced "28F016SA User's Manual" as "16-Mbit Flash Product Family User's Manual —Section 5.11: Renamed specification "Erase Suspend Latency Time to Write" as "Auto Erase Suspend Latency Time to Write" — Minor cosmetic changes

Extended Temperature 28F016SA, 16-Mbit FlashFile™ Memory Datasheet

2.1 Lead Descriptions

Lead Descriptions

Symbol	Type	Name and Function
A_0	INPUT	**BYTE-SELECT ADDRESS:** Selects between high and low byte when device is in x8 mode. This address is latched in x8 Data Writes. Not used in x16 mode (i.e., the A_0 input buffer is turned off when BYTE# is high).
A_1–A_{15}	INPUT	**WORD-SELECT ADDRESSES:** Select a word within one 64-Kbyte block. A_{6-15} selects 1 of 1024 rows, and A_{1-5} selects 16 of 512 columns. These addresses are latched during Data Writes.
A_{16}–A_{20}	INPUT	**BLOCK-SELECT ADDRESSES:** Select 1 of 32 Erase blocks. These addresses are latched during Data Writes, Erase and Lock-Block operations.
DQ_0–DQ_7	INPUT/ OUTPUT	**LOW-BYTE DATA BUS:** Inputs data and commands during CUI write cycles. Outputs array, buffer, identifier or status data in the appropriate read mode. Floated when the chip is de-selected or the outputs are disabled.
DQ_8–DQ_{15}	INPUT/ OUTPUT	**HIGH-BYTE DATA BUS:** Inputs data during x16 Data-Write operations. Outputs array, buffer or identifier data in the appropriate read mode; not used for Status Register reads. Floated when the chip is de-selected or the outputs are disabled.
CE_0#, CE_1#	INPUT	**CHIP ENABLE INPUTS**: Activate the device's control logic, input buffers, decoders and sense amplifiers. With either CE_0# or CE_1# high, the device is de-selected and power consumption reduces to standby levels upon completion of any current Data-Write or Erase operations. Both CE_0#, CE_1# must be low to select the device. All timing specifications are the same for both signals. Device Selection occurs with the latter falling edge of CE_0# or CE_1#. The first rising edge of CE_0# or CE_1# disables the device.
RP#	INPUT	**RESET/POWER-DOWN:** RP# low places the device in a Deep Power-Down state. All circuits that burn static power, even those circuits enabled in standby mode, are turned off. When returning from Deep Power-Down, a recovery time of 550 ns at 5.0V V_{CC} is required to allow these circuits to power-up. When RP# goes low, any current or pending WSM operation(s) are terminated, and the device is reset. All Status Registers return to ready (with all status flags cleared). Exit from Deep Power-Down places the device in read array mode.
OE#	INPUT	**OUTPUT ENABLE:** Gates device data through the output buffers when low. The outputs float to tri-state off when OE# is high. **NOTE:** CE_X# overrides OE#, and OE# overrides WE#.
WE#	INPUT	**WRITE ENABLE:** Controls access to the CUI, Page Buffers, Data Queue Registers and Address Queue Latches. WE# is active low, and latches both address and data (command or array) on its rising edge. Page Buffer addresses are latched on the falling edge of WE#.
RY/BY#	OPEN DRAIN OUTPUT	**READY/BUSY:** Indicates status of the internal WSM. When low, it indicates that the WSM is busy performing an operation. RY/BY# high indicates that the WSM is ready for new operations (or WSM has completed all pending operations), or Erase is Suspended, or the device is in deep power-down mode. This output is always active (i.e., not floated to tri-state off when OE# or CE_0#, CE_1# are high), except if a RY/BY# Pin Disable command is issued.

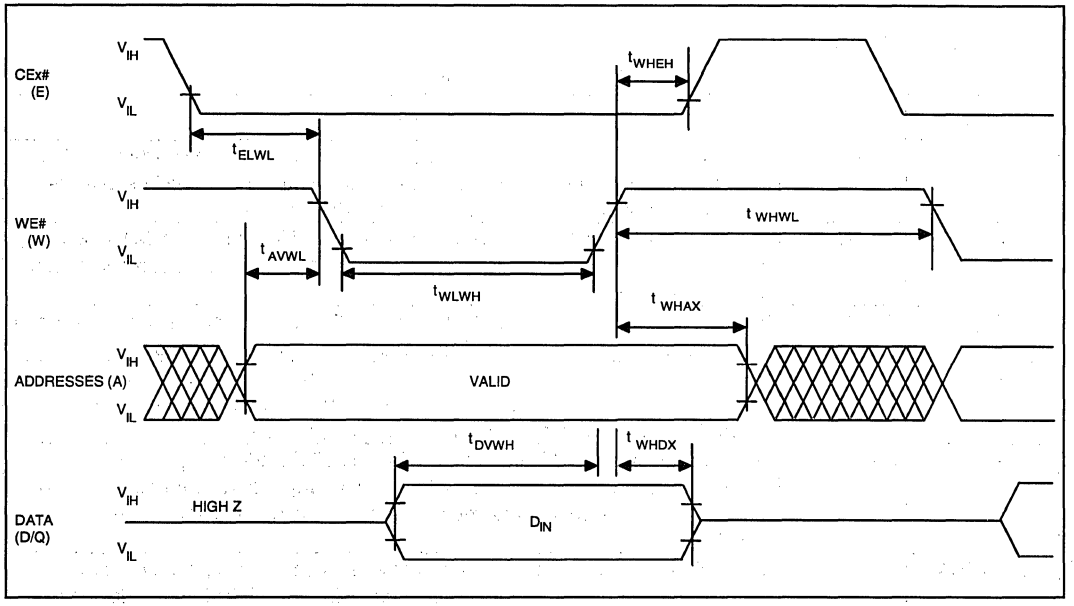


Figure 14. Page Buffer Write Timing Waveforms (Loading Data to the Page Buffer)

5.11 Erase and Word/Byte Write Performance, Cycling Performance and Suspend Latency[1, 3] : EXTENDED TEMPERATURE OPERATION

V_{CC} = 3.3V ± 0.3V, V_{PP} = 12.0V ± 0.6V, T_A = –40°C to +85°C

Symbol	Parameter	Notes	Min	Typ[1]	Max	Units	Test Conditions
	Page Buffer Byte Write Time	1,2,4		3.26		µs	
	Page Buffer Word Write Time	1,2,4		6.53		µs	
$t_{WHRH}1$	Word/Byte Write Time	1,2		9		µs	
$t_{WHRH}2$	Block Write Time	1,2		0.6		sec	Byte Write Mode
$t_{WHRH}3$	Block Write Time	1,2		0.3		sec	Word Write Mode
	Block Erase Time	1,2		1.5		sec	
	Full Chip Erase Time	1,2		48		sec	
	Erase Suspend Latency Time to Read			7.0		µs	
	Auto Erase Suspend Latency Time to Read			10.0		µs	
	Erase Cycles	5	100,000	1,000,000		Cycles	

5.11 Erase and Word/Byte Write Performance (1,3) (Continued): EXTENDED TEMPERATURE OPERATION

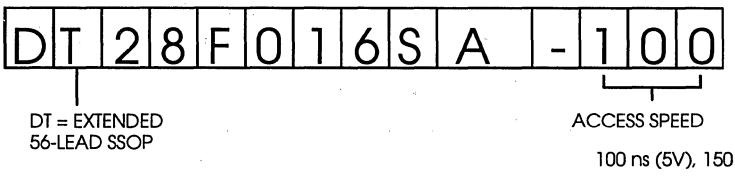
V_{CC} = 5.0V ± 0.5V, V_{PP} = 12.0V ± 0.6V, T_A = -40°C to +85°C

Sym	Parameter	Notes	Min	Typ(1)	Max	Units	Test Conditions
	Page Buffer Byte Write Time	1,2,4		2.76	Note 6	µs	
	Page Buffer Word Write Time	1,2,4		5.51	Note 6	µs	
t_{WHRH}	Word Byte/Write Time	1,2		6	Note	µs	
t_{WHRH}	Block Write Time	1,2		0.4	2.1	sec	Byte Write Mode
t_{WHRH}	Block Write Time	1,2		0.2	1.0	sec	Word Write Mode
	Block Erase Time	1,2		1.2	10	sec	
	Full Chip Erase Time	1,2		41.6		sec	
	Erase Suspend Latency Time to Read			5.0		µs	
	Erase Suspend Latency Time to Write			8.0		µs	
	Erase Cycles	5	100,000	1,000,000		Cycles	

NOTES:

1. 25°C, and normal voltages.
2. Excludes system-level overhead.
3. These performance numbers are valid for all speed versions.
4. This assumes using the full Page Buffer to Write to Flash (256 bytes or 128 words).
5. Typical 1,000,000 cycles performance assumes the application uses block retirement techniques.
6. This information will be available in a technical paper. Pleas call Intel's application Hotline or your local sales office for more information.

DEVICE NOMENCLATURE



Valid Combinations

Order Code	V_{CC} = 3.3V ± 0.3V	V_{CC} = 5.0V ± 10%
DT28F016SA-100	DT28F016SA-150	DT28F016SA-100

ADDITIONAL INFORMATION

	Item	Order Number
	16-Mbit Flash Product Family User's Manual	297372
	Commercial Temperature 28F016SA 16-Mbit FlashFile™ Memory Datasheet	290489
	Commercial Temperature 28F016SV SmartVoltage 16-Mbit FlashFile™ Memory Datasheet	290528
	28F008SA 8 Mbit FlashFile™ Memory Datasheet	290429
AP 393	28F016SV Compatibility with 28F016SA	292144
AP 378	System Optimization Using the Enhanced Features of the 28F016SA	292127
AP 377	28F016SA Software Drivers	292126
AP 375	Upgrade Considerations from the 28F008SA to the 28F016SA	292124
AP-357	Power Supply Solutions for Flash Memory	292092
ER 33	ETOX™ Flash Memory Technology—Insight to Intel's Fourth Generation Process Innovation	294016

Please check with Intel Literature for availability.

DATA SHEET REVISION HISTORY

Number	Description
001	Original Version
002	Renamed referenced "28F016SA User's Manual" as 16-Mbit Product Family User's Manual" Section 5.11: Renamed specification "Erase Suspend Latency Time to Write" as "Auto Erase Suspend Latency Time to Write" Section 2.1 RP# recovery time is 550 ns Minor cosmetic changes

intel®

4-Mbit SmartVoltage Boot Block Flash Memory Family Datasheet

Refer to the DC Characteristics Table, Section 4.2 (commercial temperature) and Section 5.2 (extended temperature), for complete current and voltage specifications. Refer to the AC Characteristics Table, Section 4.3 (commercial temperature) and Section 5.3 (extended temperature), for read, write and erase performance specifications.

1.3 Applications

The 4-Mbit boot block flash memory family combines high-density, low-power, high-performance, cost-effective flash memories with blocking and hardware protection capabilities. Their flexibility and versatility reduce costs throughout the product life cycle. Flash memory is ideal for Just-In-Time production flow, reducing system inventory and costs, and eliminating component handling during the production phase.

When your product is in the end-user's hands, and updates or feature enhancements become necessary, flash memory reduces the update costs by allowing user-performed code changes instead of costly product returns or technician calls.

The 4-Mbit boot block flash memory family provides full-function, blocked flash memories suitable for a wide range of applications. These applications include extended PC BIOS and ROM-able applications storage, digital cellular phone program and data storage, telecommunication boot/firmware, printer firmware/font storage and various other embedded applications where program and data storage are required.

Reprogrammable systems such as personal computers, are ideal applications for the 4-Mbit flash memory products. Increasing software sophistication greatens the probability that a code update will be required after the PC is shipped. For example, the emerging of "plug and play" standard in desktop and portable PCs enables auto-configuration of ISA and PCI add-in cards. However, since the "plug and play" specification continues to evolve, a flash BIOS provides a cost-effective capability to update existing PCs. In addition, the parameter blocks are ideal for storing the required auto-configuration parameters, allowing you to integrate the BIOS PROM and parameter storage EEPROM into a single component, reducing parts costs while increasing functionality.

The 4-Mbit flash memory products are also excellent design solutions for digital cellular phone and telecommunication switching applications requiring very low power consumption, high-performance, high-density storage capability, modular software designs, and a small form factor package. The 4-Mbit's blocking scheme allows for easy segmentation of embedded code with 16 Kbytes of hardware-protected boot code, four main blocks for program code, and two 8-Kbyte parameter blocks for frequently updated data storage and diagnostic messages (e.g., phone numbers, authorization codes).

Intel's high integration flash architecture provides a flexible voltage solution for the different design needs of various applications. The asymmetrically blocked memory map allows the integration of several memory components into a single flash device. The boot block provides a secure boot PROM; the parameter blocks can emulate EEPROM functionality for parameter store with proper software techniques; and the main blocks provide code and data storage with access times fast enough to execute code in place, decreasing RAM requirements.

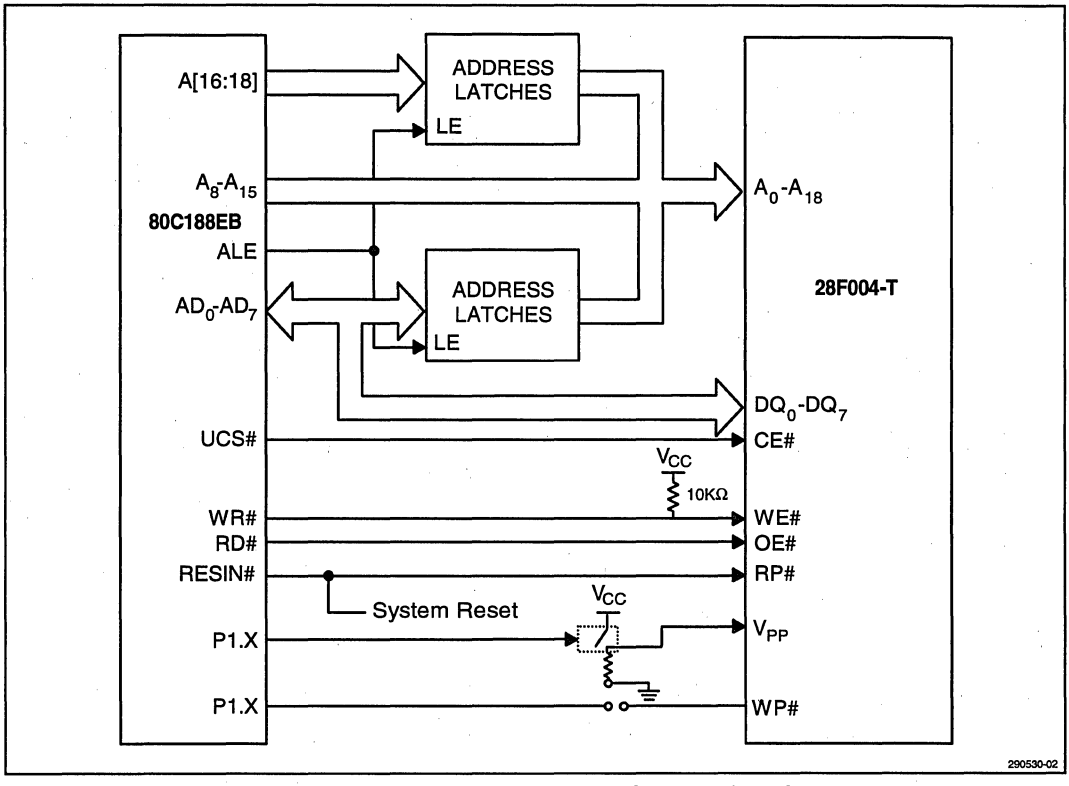
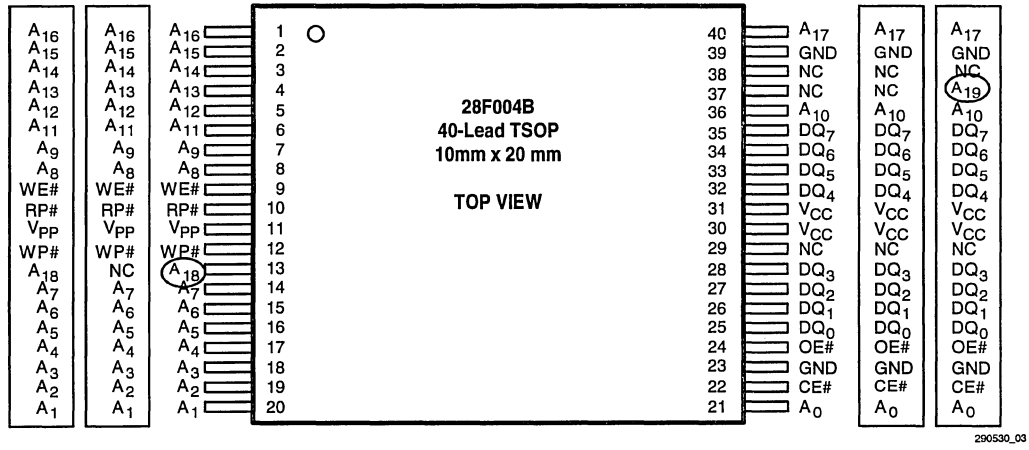


Figure 2. 28F004 Interface to Intel80C188EB 8-Bit Embedded Microprocessor

1.4 Pinouts

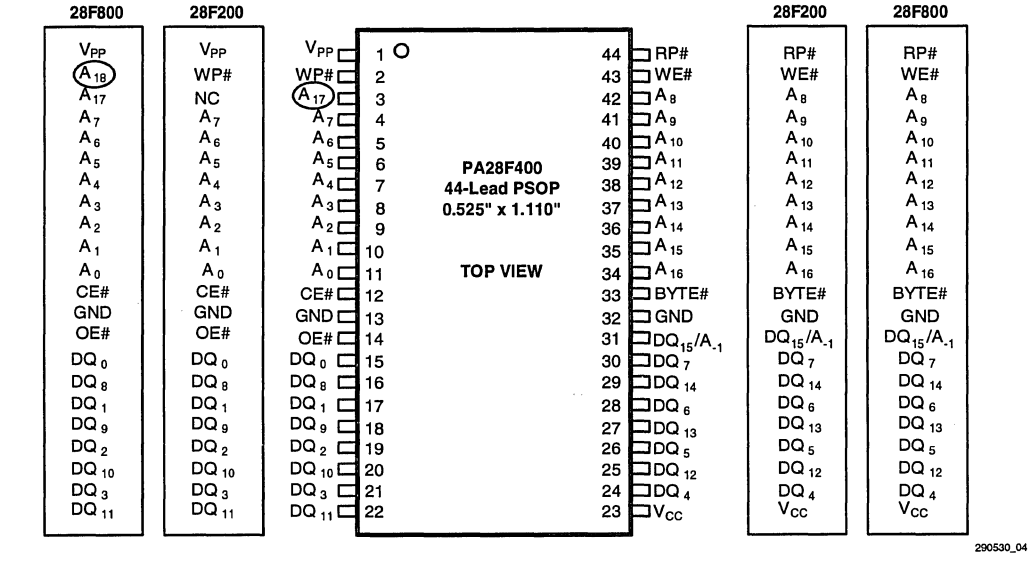
Intel's SmartVoltage Boot Block architecture provides upgrade paths in most package pinout to the 8-Mbit density. The 28F004 40-lead TSOP pinout for space-constrained designs is shown in Figure 3. The 28F400 44-lead PSOP pinout follows the industry-standard ROM/EPROM pinout, as shown in Figure 4. For designs that require x16 operation but have space concerns, refer to the 48-lead pinout in Figure 5. Furthermore, the 28F400 56-lead TSOP pinout shown in Figure 6 provides backwards compatibility with exisiting 28F400BX designs.

Pinouts for the corresponding 2-Mbit and 8-Mbit components are also provided for convenient reference. 4-Mbit pinouts are given on the chip illustration in the center, with 2-Mbit and 8-Mbit pinouts going outward from the center.



NOTE:
1. Pin 12 is DU for BX/BL 12V V_{PP} Versions.
2. The 28F008B pinout shown is for the 8-Mbit boot block and not the 28F008SA FlashFile™ memory.

Figure 3. The 40-Lead TSOP Offers the Smallest Form Factor for Space-Constrained Applications



NOTE: Pin 2 is DU for BX/BL 12V V_{PP} Versions, but for the 8-Mbit device, pin 2 has been changed to A_{18} (WP# on 2/4 Mbit). Designs planning on upgrading to the 8-Mbit density from the 2/4-Mbit density in this package should design pin 2 to control WP# functionality at the 2/4-Mbit level and allow for pin 2 to control A_{18} after upgrading to the 8-Mbit density.

Figure 4. The 44-Lead PSOP Offers a Convenient Upgrade from JEDEC ROM Standards

Register is set to a "1" to indicate a Program Failure. If bit 3 is set to a "1," then V_{PP} was not within acceptable limits, and the WSM did not execute the programming sequence. If the program operation fails, bit 4 of the Status Register will be set within 1.5 ms as determined by the timeout of the WSM.

The Status Register should be cleared before attempting the next operation. Any CUI instruction can follow after programming is completed; however, reads from the Memory Array, Status Register, or Intelligent Identifier cannot be accomplished until the CUI is given the Read Array command.

3.3.4 ERASE MODE

Erasure of a single block is initiated by writing the Erase Setup and Erase Confirm commands to the CUI, along with the addresses identifying the block to be erased. These addresses are latched internally when the Erase Confirm command is issued. Block erasure results in all bits within the block being set to "1."

The WSM will execute a sequence of internally timed events to:

1. Program all bits within the block to "0."
2. Verify that all bits within the block are sufficiently programmed to "0."
3. Erase all bits within the block.
4. Verify that all bits within the block are sufficiently erased.

While the erase sequence is executing, bit 7 of the Status Register is a "0."

When the Status Register indicates that erasure is complete, the status bits, which indicate whether the Erase operation was successful, should be checked. If the Erase operation was unsuccessful, bit 5 of the Status Register will be set to a "1," indicating an Erase Failure. If V_{PP} was not within acceptable limits after the Erase Confirm command is issued, the WSM will not execute an erase sequence; instead, bit 5 of the Status Register is set to a "1" to indicate an Erase Failure, and bit 3 is set to a "1" to identify that V_{PP} supply voltage was not within acceptable limits.

The Status Register should be cleared before attempting the next operation. Any CUI instruction can follow after erasure is completed; however, reads from the Memory Array, Status Register, or Intelligent Identifier cannot be accomplished until the CUI is given the Read Array command.

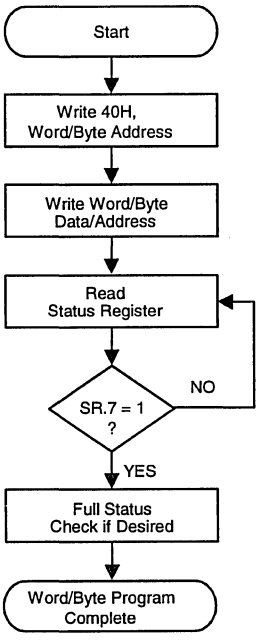
3.3.4.1 Suspending and Resuming Erase

Since an erase operation requires on the order of seconds to complete, an Erase Suspend command is provided to allow erase-sequence interruption in order to read data from another block of the memory. Once the erase sequence is started, writing the Erase Suspend command to the CUI requests that the WSM pause the erase sequence at a predetermined point in the erase algorithm. The Status Register must then be read to determine if the erase operation has been suspended.

At this point, a Read Array command can be written to the CUI in order to read data from blocks other than that which is being suspended. The only other valid command at this time is the Erase Resume command or Read Status Register command.

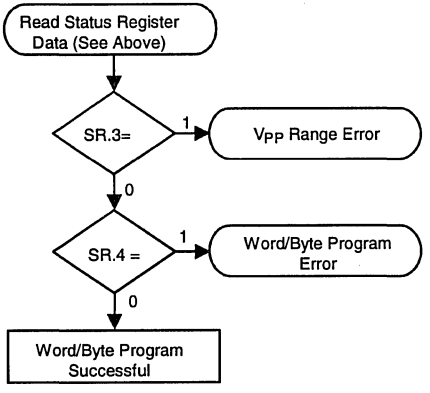
During erase suspend mode, the chip can go into a pseudo-standby mode by taking CE# to V_{IH}, which reduces active current draw.

To resume the erase operation, the chip must be enabled by taking CE# to V_{IL}, then issuing the Erase Resume command. When the Erase Resume command is given, the WSM will continue with the erase sequence and complete erasing the block. As with the end of a standard erase operation, the Status Register must be read, cleared, and the next instruction issued in order to continue.



Bus Operation	Command	Comments
Write	Setup Program	Data = 40H Addr = Word/Byte to Program
Write	Program	Data = Data to Program Addr = Location to Program
Read		Status Register Data Toggle CE# or OE# to Update SRD.
Standby		Check SR.7 1 = WSM Ready 0 = WSM Busy
Repeat for subsequent Word/Byte Writes. SR Full Status Check can be done after each Word/Byte Write, or after a sequence of Word/Byte Writes. Write FFH after the last write operation to reset device to read array mode.		

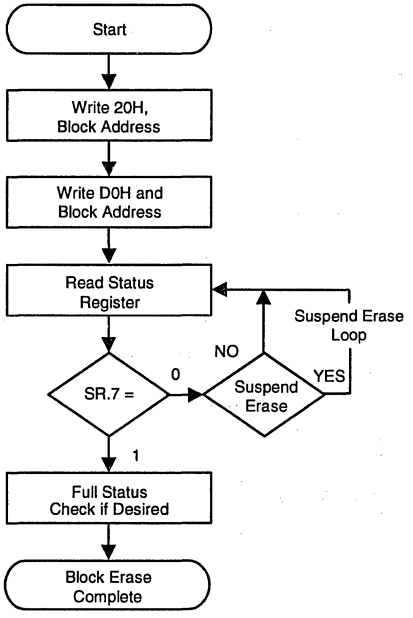
FULL STATUS CHECK PROCEDURE



Bus Operation	Command	Comments
Standby		Check SR.3 1 = V_{PP} Low Detect
Standby		Check SR.4 1 = Word/Byte Program Error
SR.3 MUST be cleared, if set during a program attempt, before further attempts are allowed by the Write State Machine. SR.4 is only clear by the Clear Status Register Command, in cases where multiple bytes are programmed before full status is checked. If error is detected, clear the Status Register before attempting retry or other error recovery.		

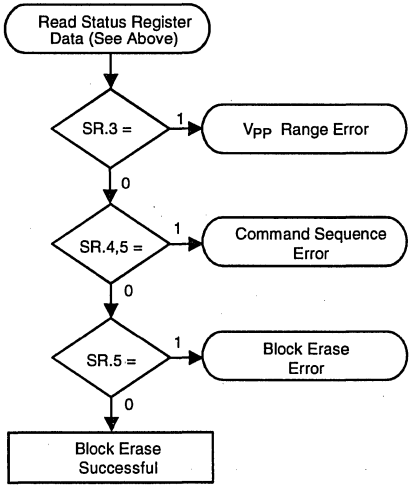
290530_09

Figure 9. Automated Word/Byte Programming Flowchart



Bus Operation	Command	Comments
Write	Erase Setup	Data = 20H Addr = Within Block to be Erased
Write	Erase Confirm	Data = D0H Addr = Within Block to be Erased
Read		Status Register Data Toggle CE# or OE# to Update Status Register
Standby		Check SR.7 1 = WSM Ready 0 = WSM Busy
Repeat for subsequent block erasures. Full Status Check can be done after each block erase, or after a sequence of block erasures. Write FFH after the last operation to reset device to read array mode.		

FULL STATUS CHECK PROCEDURE



Bus Operation	Command	Comments
Standby		Check SR.3 1 = V_{PP} Low Detect
Standby		Check SR.4,5 Both 1 = Command Sequence Error
Standby		Check SR.5 1 = Block Erase Error
SR.3 MUST be cleared, if set during an erase attempt, before further attempts are allowed by the Write State Machine. SR.5 is only clear by the Clear Status Register Command, in cases where multiple blocks are erase before full status is checked. If error is detected, clear the Status Register before attempting retry or other error recovery.		

290530_10

Figure 10. Automated Block Erase Flowchart

2-Mbit SmartVoltage Boot Block Flash Memory Family Datasheet

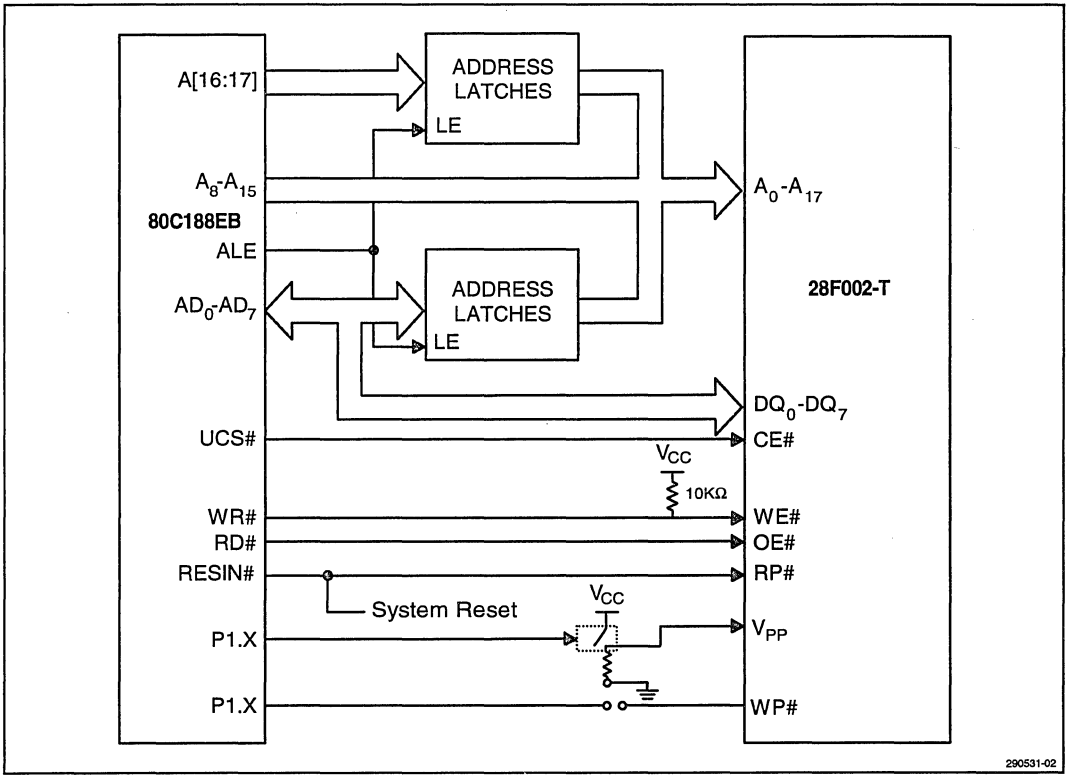
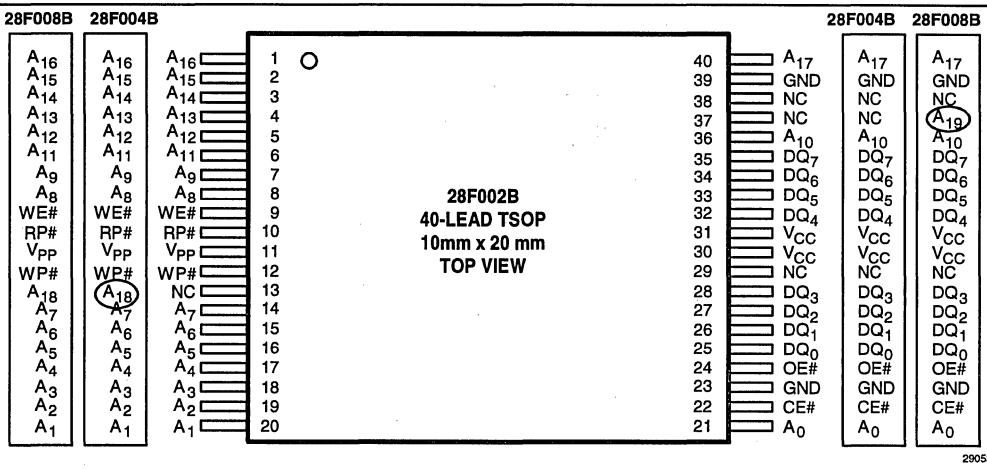


Figure 1. 28F002 Interface to Intel80C188EB 8-Bit Embedded Microprocessor

1.4 Pinouts

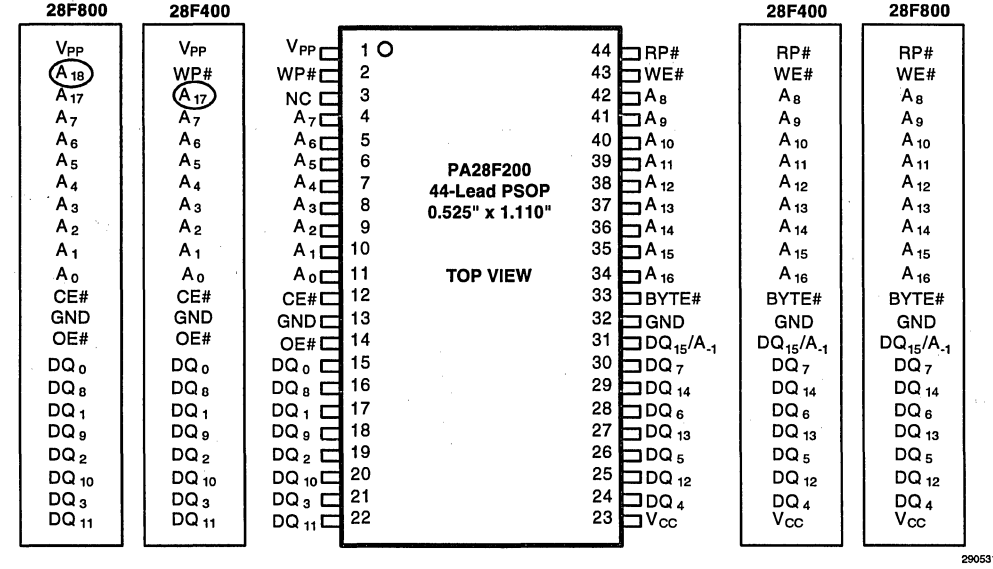
Intel's SmartVoltage Boot Block architecture provides upgrade paths in every package pinout to the 8-Mbit density. The 28F002 40-lead TSOP pinout for space-constrained designs is shown in Figure 3. The 28F200 44-lead PSOP pinout follows the industry-standard ROM/EPROM pinout, as shown in Figure 4. For designs that require x16 operation but have space concerns, refer to the 48-lead pinout in Figure 5. Furthermore, the 28F200 56-lead TSOP pinout shown in Figure 6 provides density upgrades to the 28F400BV as well as backwards compatibility with existing 28F200BX/BL designs.

Pinouts for the corresponding 4-Mbit and 8-Mbit components are also provided for convenient reference. 2-Mbit pinouts are given on the chip illustration in the center, with 4-Mbit and 8-Mbit pinouts going outward from the center.



NOTES:
1. Pin 12 is DU for BX/BL 12V V_{PP} Versions.
2. The 28F008B pinout shown is for the 8-Mbit boot block and not the 28F008SA FlashFile™ memory.

Figure 3. The 40-Lead TSOP Offers the Smallest Form Factor for Space-Constrained Applications



NOTE: Pin 2 is DU for BX/BL 12V V_{PP} Versions, but for the 8-Mbit device, pin 2 has been changed to A_{18} (WP# on 2/4 Mbit). Designs planning on upgrading to the 8-Mbit density from the 2/4-Mbit density in this package should design pin 2 to control WP# functionality at the 2/4-Mbit level and allow for pin 2 to control A_{18} after upgrading to the 8-Mbit density.

Figure 4. The 44-Lead PSOP Offers a Convenient Upgrade from JEDEC ROM Standards

Register is set to a "1" to indicate a Program Failure. If bit 3 is set to a "1," then V_{PP} was not within acceptable limits, and the WSM did not execute the programming sequence. If the program operation fails, bit 4 of the Status Register will be set within 1.5 ms as determined by the timeout of the WSM.

The Status Register should be cleared before attempting the next operation. Any CUI instruction can follow after programming is completed; however, reads from the Memory Array, Status Register, or Intelligent Identifier cannot be accomplished until the CUI is given the Read Array command.

3.3.4 ERASE MODE

Erasure of a single block is initiated by writing the Erase Setup and Erase Confirm commands to the CUI, along with the addresses identifying the block to be erased. These addresses are latched internally when the Erase Confirm command is issued. Block erasure results in all bits within the block being set to "1."

The WSM will execute a sequence of internally timed events to:

1. Program all bits within the block to "0."
2. Verify that all bits within the block are sufficiently programmed to "0."
3. Erase all bits within the block.
4. Verify that all bits within the block are sufficiently erased.

While the erase sequence is executing, bit 7 of the Status Register is a "0."

When the Status Register indicates that erasure is complete, the status bits, which indicate whether the Erase operation was successful, should be checked. If the Erase operation was unsuccessful, bit 5 of the Status Register will be set to a "1," indicating an Erase Failure. If V_{PP} was not within acceptable limits after the Erase Confirm command is issued, the WSM will not execute an erase sequence; instead, bit 5 of the Status Register is set to a "1" to indicate an Erase Failure, and bit 3 is set to a "1" to identify that V_{PP} supply voltage was not within acceptable limits.

The Status Register should be cleared before attempting the next operation. Any CUI instruction can follow after erasure is completed; however, reads from the Memory Array, Status Register, or Intelligent Identifier cannot be accomplished until the CUI is given the Read Array command.

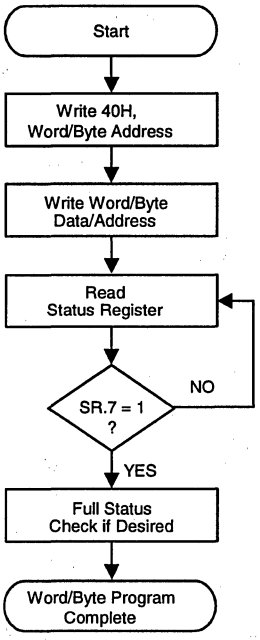
3.3.4.1 Suspending and Resuming Erase

Since an erase operation requires on the order of seconds to complete, an Erase Suspend command is provided to allow erase-sequence interruption in order to read data from another block of the memory. Once the erase sequence is started, writing the Erase Suspend command to the CUI requests that the WSM pause the erase sequence at a predetermined point in the erase algorithm. The Status Register must then be read to determine if the erase operation has been suspended.

At this point, a Read Array command can be written to the CUI in order to read data from blocks other than that which is being suspended. The only other valid command at this time is the Erase Resume command or Read Status Register command.

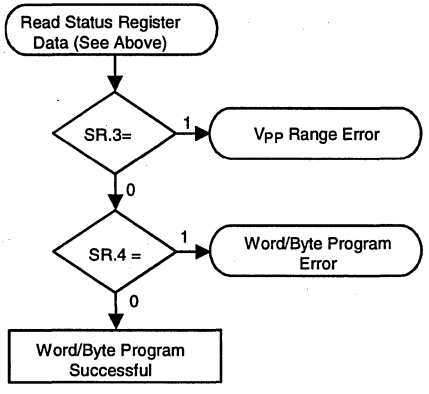
During erase suspend mode, the chip can go into a pseudo-standby mode by taking CE# to V_{IH}, which reduces active current draw.

To resume the erase operation, the chip must be enabled by taking CE# to V_{IL}, then issuing the Erase Resume command. When the Erase Resume command is given, the WSM will continue with the erase sequence and complete erasing the block. As with the end of a standard erase operation, the Status Register must be read, cleared, and the next instruction issued in order to continue.



Bus Operation	Command	Comments
Write	Setup Program	Data = 40H Addr = Word/Byte to Program
Write	Program	Data = Data to Program Addr = Location to Program
Read		Status Register Data Toggle CE# or OE# to Update SRD.
Standby		Check SR.7 1 = WSM Ready 0 = WSM Busy
Repeat for subsequent Word/Byte Writes. SR Full Status Check can be done after each Word/Byte Write, or after a sequence of Word/Byte Writes. Write FFH after the last write operation to reset device to read array mode.		

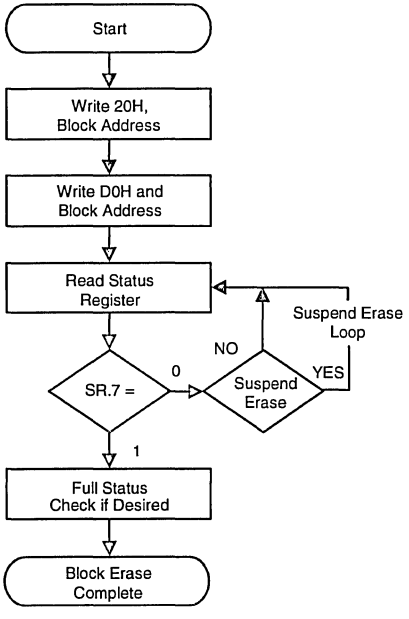
FULL STATUS CHECK PROCEDURE



Bus Operation	Command	Comments
Standby		Check SR.3 1 = V_{PP} Low Detect
Standby		Check SR.4 1 = Word/Byte Program Error
SR.3 MUST be cleared, if set during a program attempt, before further attempts are allowed by the Write State Machine. SR.4 is only clear by the Clear Status Register Command, in cases where multiple bytes are programmed before full status is checked. If error is detected, clear the Status Register before attempting retry or other error recovery.		

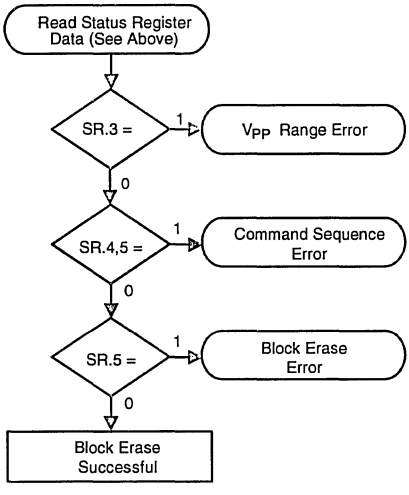
290531_09

Figure 9. Automated Word/Byte Programming Flowchart



Bus Operation	Command	Comments
Write	Erase Setup	Data = 20H Addr = Within Block to be Erased
Write	Erase Confirm	Data = D0H Addr = Within Block to be Erased
Read		Status Register Data Toggle CE# or OE# to Update Status Register
Standby		Check SR.7 1 = WSM Ready 0 = WSM Busy
Repeat for subsequent block erasures. Full Status Check can be done after each block erase, or after a sequence of block erasures. Write FFH after the last operation to reset device to read array mode.		

FULL STATUS CHECK PROCEDURE



Bus Operation	Command	Comments
Standby		Check SR.3 1 = V_{PP} Low Detect
Standby		Check SR.4,5 Both 1 = Command Sequence Error
Standby		Check SR.5 1 = Block Erase Error
SR.3 MUST be cleared, if set during an erase attempt, before further attempts are allowed by the Write State Machine. SR.5 is only clear by the Clear Status Register Command, in cases where multiple blocks are erase before full status is checked. If error is detected, clear the Status Register before attempting retry or other error recovery.		

290531_10

Figure 10. Automated Block Erase Flowchart

intel®

28F001BX-T/28F001BX-B
1M CMOS Flash Memory

Program Setup/Program Commands

Programming is executed by a two-write sequence. The Program Setup command (40H) is written to the Command Register, followed by a second write specifying the address and data (latched on the rising edge of WE#) to be programmed. The WSM then takes over, controlling the program and verify algorithms internally. After the two-command program sequence is written to it, the 28F001BX automatically outputs Status Register data when read (see Figure 9: Byte Program Flowchart). The CPU can detect the completion of the program event by analyzing the WSM Status bit of the Status Register. Only the Read Status Register command is valid while programming is active.

When the Status Register indicates that programming is complete, the Program Status bit should be checked. If program error is detected, the Status Register should be cleared. The internal WSM only detects errors for "1s" that do not successfully program to "0s". The Command Register remains in Read Status Register mode until further commands are issued to it. If byte programming is attempted while $V_{PP} = V_{PPL}$, the V_{PP} Status bit will be set to "1". Program attempts while $V_{PPL} < V_{PP} < V_{PPH}$ produce spurious results and should not be attempted.

EXTENDED ERASE/PROGRAM CYCLING

EEPROM cycling failures have always concerned users. The high electric fields required by thin oxide EEPROMs for tunneling can literally tear apart the oxide at defect regions. To combat this, some suppliers have implemented redundancy schemes, reducing cycling failures to insignificant levels. However, redundancy requires that cell size be doubled—an expensive solution.

Intel has designed extensive cycling capability into its ETOX flash memory technology. Resulting improvements in cycling reliability come without increasing memory cell size or complexity. First, an advanced tunnel oxide increases charge carrying ability ten-fold. Second, the oxide area per cell subjected to the tunneling electric field is one-tenth that of common EEPROMs, minimizing the probability of oxide defects in the region. Finally, the peak electric field during erasure is approximately 2 MV/cm lower than EEPROM. The lower electric field greatly reduces the oxide stress and the probability of failure.

The 28F001BX-B and the 28F001BX-T are capable of 100,000 program/erase cycles on each parameter block, main block, and boot block.

ON-CHIP PROGRAMMING ALGORITHM

The 28F001BX integrates the Quick-Pulse programming algorithm of prior Intel flash memory devices on-chip, using the Command Register, Status Register, and Write State Machine (WSM). On-chip integration dramatically simplifies system software and provides processor-like interface timings to the Command and Status Registers. WSM operation, internal program verify, and V_{PP} high voltage presence are monitored and reported via appropriate Status Register bits. Figure 9 shows a system software flowchart for device programming. The entire sequence is performed with V_{PP} at V_{PPH}. Program abort occurs when RP# transitions to V_{IL} or V_{PP} drops to V_{PPL}. Although the WSM is halted, byte data is partially programmed at the location where programming is aborted. Block erasure or a repeat of byte programming will initialize this data to a known value.

ON-CHIP ERASE ALGORITHM

As above, the Quick-Erase algorithm of prior Intel flash memory devices is implemented internally, including all preconditioning of block data. WSM operation, erase success, and V_{PP} high voltage presence are monitored and reported through the Status Register. Additionally, if a command other than Erase Confirm is written to the device after Erase Setup has been written, both the Erase Status and Program Status bits will be set to "1". When issuing the Erase Setup and Erase Confirm commands, they should be written to an address within the address range of the block to be erased. Figure 10 shows a system software flowchart for block erase.

Erase typically takes 1-4 seconds per block. The Erase Suspend/Erase Resume command sequence allows interruption of this erase operation to read data from a block other than the block in which erase is being performed. A system software flowchart is shown in Figure 11.

The entire sequence if performed with V_{PP} at V_{PPH}. Abort occurs when RP# transitions to V_{IL} or V_{PP} falls to V_{PPL}, while erase is in progress. Block data is partially erased by this operation, and a repeat if the erase is required to obtain a fully erased block.

intel®

28F020
2048K CMOS Flash Memory

Program-Verify Command

The 28F020 is programmed on a byte-by-byte basis. Byte programming may occur sequentially or at random. Following each programming operation, the byte just programmed must be verified.

The program-verify operation is initiated by writing C0H into the command register. The register write terminates the programming operation with the riding edge of its WE# pulse. The program-verify operation stages the device for verification of the byte last programmed. No new address information is latched.

The 28F020 applies its internally-generated margin voltage to the byte. A microprocessor read cycle outputs the data. A successful comparison between the programmed byte and the true data means the byte was successfully programmed. Programming then proceeds to the next desired byte location. Figure 5, the 28F020 Quick-Pulse algorithm, illustrates how commands are combined with bus operations to perform byte programming. Refer to AC Programming Characteristics and Waveforms for specific timing parameters.

Reset Command

A reset command is provided as a means to safely abort the erase- or program-command sequences. Following either setup command (erase or program) with two consecutive write of FFH will safely abort the operation. Memory contents will not be altered. A valid command must then be written to place the device in the desired state.

EXTENDED ERASE/PROGRAM CYCLING

EEPROM cycling failures have always concerned users. The high electric fields required by thin oxide EEPROMs for tunneling can literally tear apart the oxide at defect regions. To combat this, some suppliers have implemented redundancy schemes, reducing cycling failures to insignificant levels. However, redundancy requires that cell size be doubled—an expensive solution.

Intel has designed extensive cycling capability into its ETOX flash memory technology. Resulting improvements in cycling reliability come without increasing memory cell size or complexity. First, an advanced tunnel oxide increases charge carrying ability ten-fold. Second, the oxide area per cell subjected to the tunneling electric field is one-tenth that of common EEPROMs, minimizing the probability of oxide defects in the region. Finally, the peak electric field during erasure is approximately 2 MV/cm lower than EEPROM. The lower electric field greatly reduces the oxide stress and the probability of failure.

The 28F020 is capable of 100,000 program/erase cycles. The device is programmed and erased using Intel's Quick-Pulse Programming and Quick-Erase algorithms. Intel's algorithmic approach uses a series of operations (pulses), along with byte verification, to completely and reliably erase and program the device.

For further information, see Reliability Report RR-60.

QUICK-PULSE PROGRAMMING ALGORITHM

The Quick-Pulse Programming algorithm uses programming operations of 10 ms duration. Each operation is followed by a byte verification to determine when the addressed byte has been successfully programmed. The algorithm allows for up to 25 programming operations per byte, although most bytes verify on the first or second operation. The entire sequence of programming and byte verification is performed with V_{PP} at high voltage. Figure 5 illustrates the Quick-Pulse Programming algorithm.

QUICK-ERASE ALGORITHM

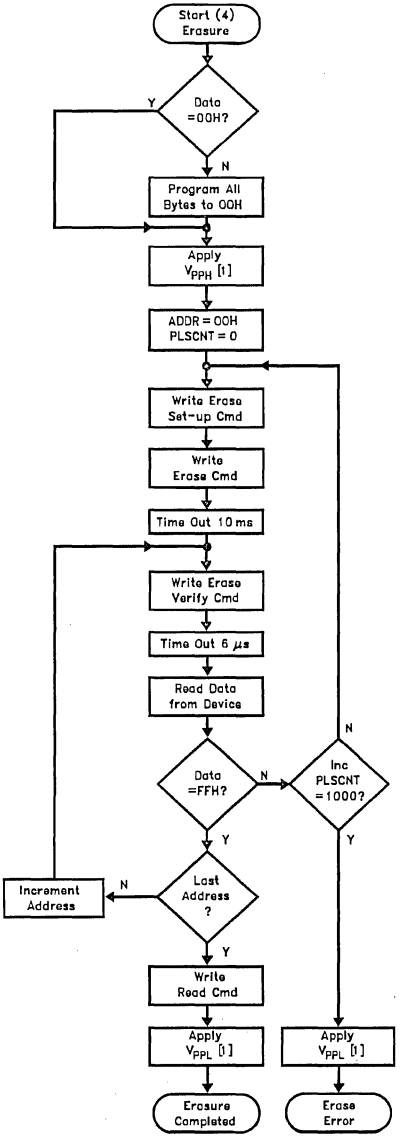
Intel's Quick-Erase algorithm yields fast and reliable electrical erasure of memory contents. The algorithm employs a closed-loop flow, similar to the Quick-Pulse Programming algorithm, to simultaneously remove charge from all bits in the array.

Erasure begins with a read of memory contents. The 28F020 is erased when shipped from the factory. Reading FFH from the device would immediately be followed by device programming.

For devices being erased and programmed, uniform and reliable erasure is ensured by first programming all bits in the device to their charged state (Data = 00H). This is accomplished, using the Quick-Pulse Programming algorithm, in approximately four seconds.

Erase execution then continues with an initial erase operation. Erase verification (Data = FFH) begins at address 0000H and continues through the array to the last address, or until data other than FFH is encountered. With each erase operation, an increased number of bytes verify to the erased state. Erase efficiency may be improved by storing the address of the last byte verified in a register. Following the next erase operation, verification starts at that stored address location. Erasure typically occurs in two seconds. Figure 6 illustrates the Quick-Erase algorithm.

5-10 (Continued)



Bus Operation	Command	Comments
		Entire Memory Must = 00H before Erasure
		Use Quick-Pulse Programming Algorithm (Figure 5)
Standby		Wait for V_{pp} Ramp to V_{PPH} (1)
		Initialize Addresses and Pulse-Count
Write	Set-Up Erase	Data = 20H
Write	Erase	Data = 20H
Standby		Duration of Erase Operation (t_{WHWH2})
Write	Erase (2) Verify	Addr = Byte to Verify; Data = A0H; Stops Erase Operation (3)
Standby		t_{WHGL}
Read		Ready Byte to Verify Erasure
Standby		Compare Output to FFH Increment Pulse-Count
Write	Read	Data = 00H, Resets the Register for Read Operations
Standby		Wait for V_{PP} Ramp to V_{PPH} (1)

290245-8

1. See DC Characteristics for the value of V_{PPH} and V_{PPL}
2. Erase Verify is performed only after chip-erasure. A final read/compare may be performed (optional) after the register is written with the Read command.
3. Refer to principles of operation.
4. **CAUTION: The algorithm MUST BE FOLLOWED to ensure proper and reliable operation of the device.**

Figure 6. 20F020 Quick-Erase Algorithm

AC CHARACTERISTICS—Read Only Operations—Commercial and Extended Temperature Products

Versions		V_{CC} + 5%	28F020-70(4)								Unit
		V_{CC} + 10%			28F020-70(5)		28F020-90(5)		28F020-150(5)		
Symbol	Characterisitcs	Notes	Min	Max	Min	Max	Min	Max	Min	Max	
t_{AVAV}/t_{RC}	Read Cycle Time		70		80		90		150		ns
t_{ELQV}/t_{CE}	Chip Enable Access Time			70		80		90		120	ns
t_{AVQV}/t_{ACC}	Address Access Time			70		80		90		120	ns
t_{GLQV}/t_{OE}	Output Enable Access Time			28		30		35		50	ns
t_{ELQX}/t_{LZ}	Chip Enable to Output in Low Z	2,3	0		0		0		0		ns
t_{EHQZ}	Chip Disable to Output in High Z	2		35		40		45		55	ns
t_{GLQX}/t_{OLZ}	Output Enable to Output in Low Z	2,3	0		0		0		0		ns
t_{GHQZ}/t_{DF}	Output Disable to Ouput in High Z	2		30		30		30		30	ns
t_{OH}	Output Hold from Address, CE#, or OE# Change	1,2	0		0		0		0		ns
t_{WHGL}	Write Recovery Time before Read		6		6		6		6		µs

NOTES:

1. Whichever occurs first.
2. Sampled, not 100% tested.
3. Guaranteed by design.
4. See High Speed AC Input/Output reference Waveforms and High Speed AC Testing Load Circuits for testing characteristics.
5. See AC Input/Output reference Waveforms and AC Testing Load Circuits for testing characteristics.

intel®

28F010
1024K CMOS Flash Memory

Program-Verify Command

The 28F010 is programmed on a byte-by-byte basis. Byte programming may occur sequentially or at random. Following each programming operation, the byte just programmed must be verified.

The program-verify operation is initiated by writing C0H into the command register. The register write terminates the programming operation with the riding edge of its WE# pulse. The program-verify operation stages the device for verification of the byte last programmed. No new address information is latched.

The 28F010 applies its internally-generated margin voltage to the byte. A microprocessor read cycle outputs the data. A successful comparison between the programmed byte and the true data means the byte was successfully programmed. Programming then proceeds to the next desired byte location. Figure 5, the 28F010 Quick-Pulse algorithm, illustrates how commands are combined with bus operations to perform byte programming. Refer to AC Programming Characteristics and Waveforms for specific timing parameters.

Reset Command

A reset command is provided as a means to safely abort the erase- or program-command sequences. Following either setup command (erase or program) with two consecutive writes of FFH will safely abort the operation. Memory contents will not be altered. A valid command must then be written to place the device in the desired state.

EXTENDED ERASE/PROGRAM CYCLING

EEPROM cycling failures have always concerned users. The high electric fields required by thin oxide EEPROMs for tunneling can literally tear apart the oxide at defect regions. To combat this, some suppliers have implemented redundancy schemes, reducing cycling failures to insignificant levels. However, redundancy requires that cell size be doubled—an expensive solution.

Intel has designed extensive cycling capability into its ETOX flash memory technology. Resulting improvements in cycling reliability come without increasing memory cell size or complexity. First, an advanced tunnel oxide increases charge carrying ability ten-fold. Second, the oxide area per cell subjected to the tunneling electric field is one-tenth that of common EEPROMs, minimizing the probability of oxide defects in the region. Finally, the peak electric field during erasure is approximately 2 MV/cm lower than EEPROM. The lower electric field greatly reduces the oxide stress and the probability of failure.

The 28F010 is capable of 100,000 program/erase cycles. The device is programmed and erased using Intel's Quick-Pulse Programming and Quick-Erase algorithms. Intel's algorithmic approach uses a series of operations (pulses), along with byte verification, to completely and reliably erase and program the device.

For further information, see Reliability Report RR-60.

QUICK-PULSE PROGRAMMING ALGORITHM

The Quick-Pulse Programming algorithm uses programming operations of 10 µs duration. Each operation is followed by a byte verification to determine when the addressed byte has been successfully programmed. The algorithm allows for up to 25 programming operations per byte, although most bytes verify on the first or second operation. The entire sequence of programming and byte verification is performed with V_{PP} at high voltage. Figure 5 illustrates the Quick-Pulse Programming algorithm.

QUICK-ERASE ALGORITHM

Intel's Quick-Erase algorithm yields fast and reliable electrical erasure of memory contents. The algorithm employs a closed-loop flow, similar to the Quick-Pulse Programming algorithm, to simultaneously remove charge from all bits in the array.

Erasure begins with a read of memory contents. The 28F010 is erased when shipped from the factory. Reading FFH from the device would immediately be followed by device programming.

For devices being erased and programmed, uniform and reliable erasure is ensured by first programming all bits in the device to their charged state (Data = 00H). This is accomplished, using the Quick-Pulse Programming algorithm, in approximately four seconds.

Erase execution then continues with an initial erase operation. Erase verification (Data = FFH) begins at address 0000H and continues through the array to the last address, or until data other than FFH is encountered. With each erase operation, an increased number of bytes verify to the erased state. Erase efficiency may be improved by storing the address of the last byte verified in a register. Following the next erase operation, verification starts at that stored address location. Erasure typically occurs in two seconds. Figure 6 illustrates the Quick-Erase algorithm.

5-33 (Continued)

AC CHARACTERISTICS—Read Only Operations—Commercial and Extended Temperature Products

Versions		V_{CC} + 5%	28F010-65(4)									
		V_{CC} + 10%			28F010-65(5)		28F010-90(5)		28F010-120(5)		28F010-150(5)		Unit
Symbol	Characterisitcs	Notes	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	
t_{AVAV}/t_{RC}	Read Cycle Time		65		70		90		120		150		ns
t_{ELQV}/t_{CE}	Chip Enable Access Time			65		70		90		120		150	ns
t_{AVQV}/t_{ACC}	Address Access Time			65		70		90		120		150	ns
t_{GLQV}/t_{OE}	Output Enable Access Time			25		28		35		50		55	ns
t_{ELQX}/t_{LZ}	Chip Enable to Output in Low Z	2,3	0		0		0		0		0		ns
t_{EHQZ}	Chip Disable to Output in High Z	2		35		40		45		55		55	ns
t_{GLQX}/t_{OLZ}	Output Enable to Output in Low Z	2,3	0		0		0		0		0		ns
t_{GHQZ}/t_{DF}	Output Disable to Ouput in High Z	2		30		30		30		30		35	ns
t_{OH}	Output Hold from Address, CE#, or OE# Change	1,2	0		0		0		0		0		ns
t_{WHGL}	Write Recovery Time before Read		6		6		6		6		6		µs

NOTES:
1. Whichever occurs first.
2. Sampled, not 100% tested.
3. Guaranteed by design.
4. See High Speed AC Input/Output reference Waveforms and High Speed AC Testing Load Circuits for testing characteristics.
5. See AC Input/Output reference Waveforms and AC Testing Load Circuits for testing characteristics.

ERASE AND PROGRAMMING PERFORMANCE

Parameter	Notes	Min	Typical	Max	Unit
Chip Erase Time	1,3,4		2	30	Sec
Chip Program Time	1,2,4		4	25	Sec

NOTES:

1. "Typicals" are not guaranteed, but based on samples from production lots. Data taken at 25°C, 12.0V V_{PP}.
2. Minimum byte programming time excluding system overhead is 16 µsec (10 µsec program + 6 µsec write recovery), while maximum is 400 µsec/byte (16 µsec x 25 loops allowed by algorithm). Max chip programming time is specified lower than the worst case allowed by the programming algorithm since most bytes program significantly faster than the worst case byte.
3. Excludes 00H programming prior to erasure.
4. Excludes system level overhead.

intel®

Application Note-600

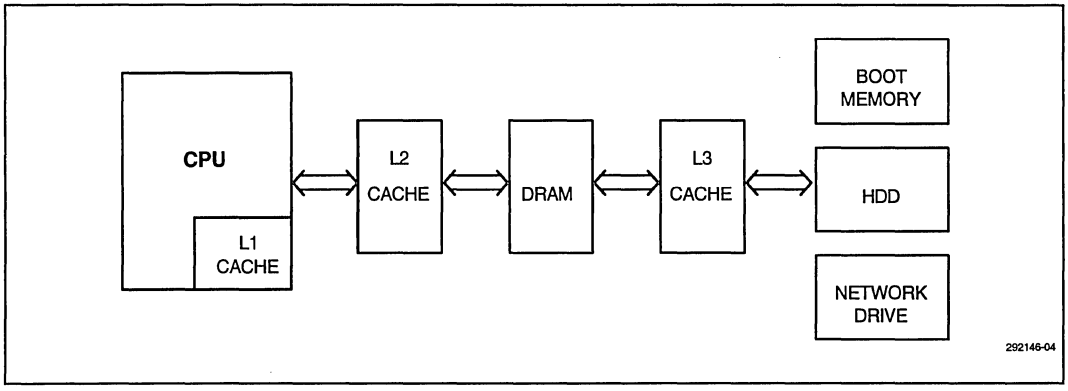


Figure 4. The Traditional System Memory Architecture Adds Complexity in Order to Optimize Performance

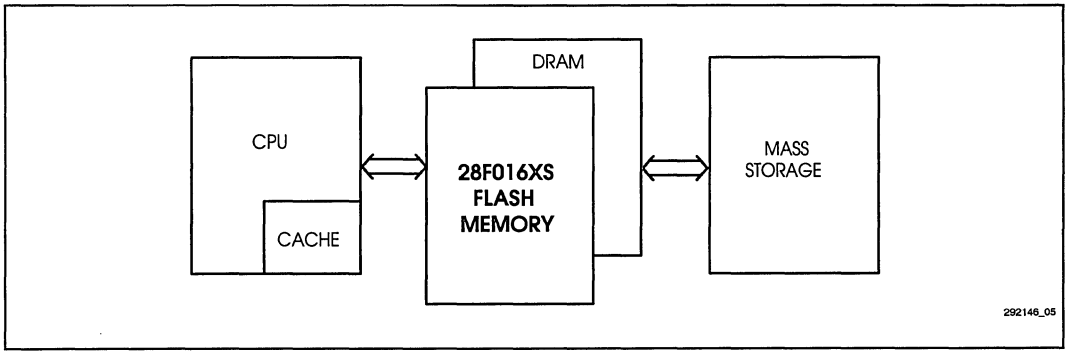


Figure 5. A Flash Memory-Based System Memory Architecture Achieves Performance without Tradeoffs

This application note compares the read performance and power consumption of Intel's 28F016XS Flash memory to that of more traditional memory alternatives, based on specifications available at the time this document was published. The 28F016XS Flash memory is a new member of the Intel 16-Mbit flash memory product family. Significant 28F016XS enhancements compared to previous flash memories include:

- A synchronous pipelined read interface that optimizes the performance of today's leading-edge microprocessors and buses, and
- SmartVoltage technology

This analysis focuses on the highest read performance versions of the highest-density products for each memory technology. Also discussed, are 28F016XS-based system memory architecture advantages over traditional alternatives in terms of performance, complexity, cost, power consumption and reliability. For complete information on Intel's 28F016XS flash memory, consult documentation listed in the Additional Information section.

intel®

Application Note-399

getting the system's BIOS code to copy the file before or during POST.

Either option is possible. The choice is dependent on determining which is easier, modifying hardware or modifying a BIOS boot-up process.

3.4 RFA Control Logic Overview

This section describes the logic design for the RFA controller. Timing analysis is also provided to draw special attention to some of the difficulties and their resolutions. Lastly, a discussion of possible future enhancements is presented.

As shown in the block diagram, Figure 1, the EPX780 provides the interface between the CPU and the flash memory. The following components are utilized in this sub-system design:

- 25 MHz 3.3V 486 SX CPU
- Altera's EPX780-132 PLD
- 4 28F016SV-75 1M x 16-bit flash memory devices
- 32-bit Transceiver
  (4 x 8-bit or 2 x 16-bit, t_{PD} <15 ns)

The signal names used in the diagrams are described in Table 1

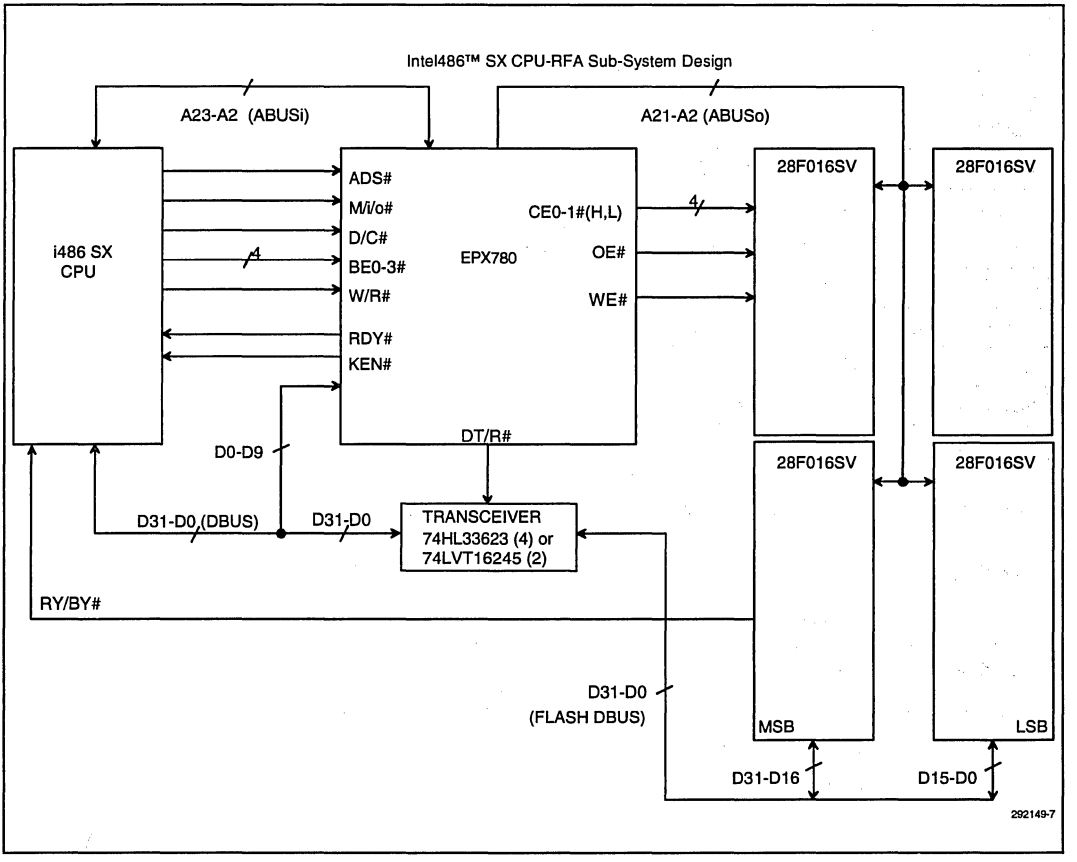


Figure 1. RFA Block Diagram

Table 1. Signal Name Descriptions

Signal name	Signal Description
CLK	Clock input to the subsystem, assumed 25 MHz
ADS#	Address Strobe from the CPU
W/R#	Write/Read# signal from CPU
M/i/o#	Memory/i/o# signal from CPU
D/C#	Data/Code# signal from CPU
WE#	Write Enable# signal from EPX780 to flash memory
OE#	Output Enable# signal from EPX780 to flash memory
RDY#	Ready# signal from EPX780 to CPU
KEN#	Cache Enable# signal from EPX780 to CPU
DT/R#	Data Transmit/Receive# signal from EPX780 to transceiver
ABUSi	Address Bus, A23–A2 from CPU to EPX780
ABUSo	Address Bus, A21–A2 from EPX780 to flash memory
BSEL#	Byte Enable# signals, BE0-3# from CPU to EPX780
CE#	Chip Enable# signals, CEH0-1# & CEL0-1 from EPX780 to flash memory
DBUS	Data Bus, D31–D0 from CPU to EPX780 and transceiver
FLASH DBUS	Data Bus, D31–D0 from transceiver to flash memory
ADSWREG	Sliding Window Register select
ADS1..4	Address Strobe delay flip-flop chain
SWREG	Sliding Window Register

NOTE:
A "#" after the name means inverted or active low.

Logic Design

The logic design (see Figure 8) for the EPX780 RFA controller was split into the following three subdivisions from a design standpoint:

- Sliding Window Address Register interface
- ROM Stub Window mapping function
- Chip enable logic for memory devices

Sliding Window Address Register Interface

The necessary address decoding for the actual loading of the Sliding Window Address Register with the base address of the window on the data bus is done using a simple AND function. Similarly, allowing the address bits to propagate out of the register to the address bus upon access to the window is done using a 2x1 mux for each bit with the address decode as the select.

APPENDIX D
PLD EQUATIONS

Included below are the equations for the EPX780

EPX780-10 PLD

```
; Functional model for Simulation of Resident Flash Array design
Title     486SX RFA design
Pattern   1
Revision  4.0
Author    Rajiv Parikh
Company   Intel Corporation
Date      9/20/94

; This design has not been verified, it is sample code only.
; Intel assumes no responsibility for any errors which may appear
; in this code.

OPTIONS DRIVE_LEVEL = 3VOLT  ; Default voltage is 3 Volts
CHIP   U1      NFX780_132


; Pinlist
; inputs
PIN          CLK              ; main clock (used by register)
; Control signals from CPU
PIN          W_R              ; Write/Read# from CPU
PIN          M_IO             ; M/i/o# from CPU
PIN          /ADS             ; ADS# from CPU
PIN          D_C              ; D/C# from CPU


PIN          A[23:2]          ; Address lines from CPU, 0,1,24-31 not used
PIN          /BE[3:0]         ; Byte Enable lines from CPU
; Control signals from PLD to memory
; outputs
PIN          B[23:2]          ; Address lines out of PLD to memory (23,22 NC)
PIN          /CEH[1:0]        ; Chip enable out of PLD to memory high word
PIN          /CEL[1:0]        ; Chip enable out of PLD to memory low word


PIN          /WE              ; Write Enable# from PLD to memory
PIN          /OE              ; Output Enable# from PLD to memory


; Other signals
; input
PIN          RY_BY            ; Ready/Busy# from memory to PLD/486
; outputs
PIN          /RDY             ; Ready# from PLD to CPU
PIN          /KEN             ; Cache enable# from PLD to CPU
```

intel®

Application Note-398

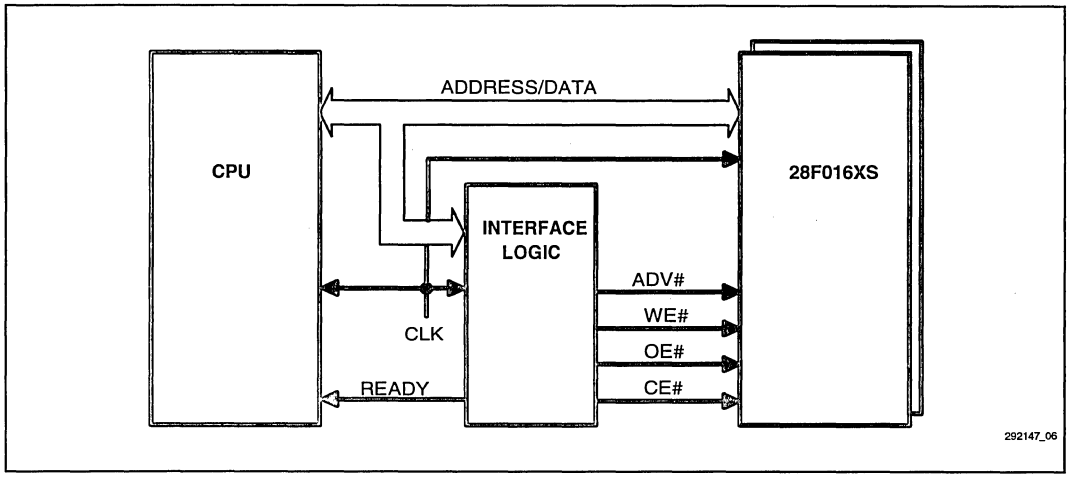


Figure 6. Comparing Past Address with Current Address to Determine Whether an Alternating-A_1 or Same-A_1 Access Occurs When Interfacing the 28F016XS to a Pipelined Bus Processor

Pipelined Bus

A pipelined bus activates the address and control signals for the next cycle before completing the current cycle. Pipelined buses have no defined access order; therefore, *Alternating-A_1* accesses are not guaranteed. The interface must guard against a possible mixture of consecutive *Alternating-A_1* and *Same-A_1* accesses. Figure 6 illustrates a pipelined bus interface to the 28F016XS.

In a pipelined interface, the system logic does not increment the 28F016XS's lower addresses. The system logic instead latches address A_1 and compares it to A_1 of the following cycle. Comparing A_1, the interface logic can identify *Alternating-A_1* and *Same-A_1* accesses, which directly informs the interface logic when it can initiate a read access to the 28F016XS. The *Alternating-A_1* and *Same-A_1* access rules define the minimum delay between consecutive accesses (see Section 4.2 and 4.3).

In the past, external latches were required to latch the next address and control signals. The 28F016XS eliminates this extra system overhead, latching the next address internally and initiating the next cycle prior to completing the current cycle. The 28F016XS's synchronous pipelined interface takes full advantage of a pipelined bus.

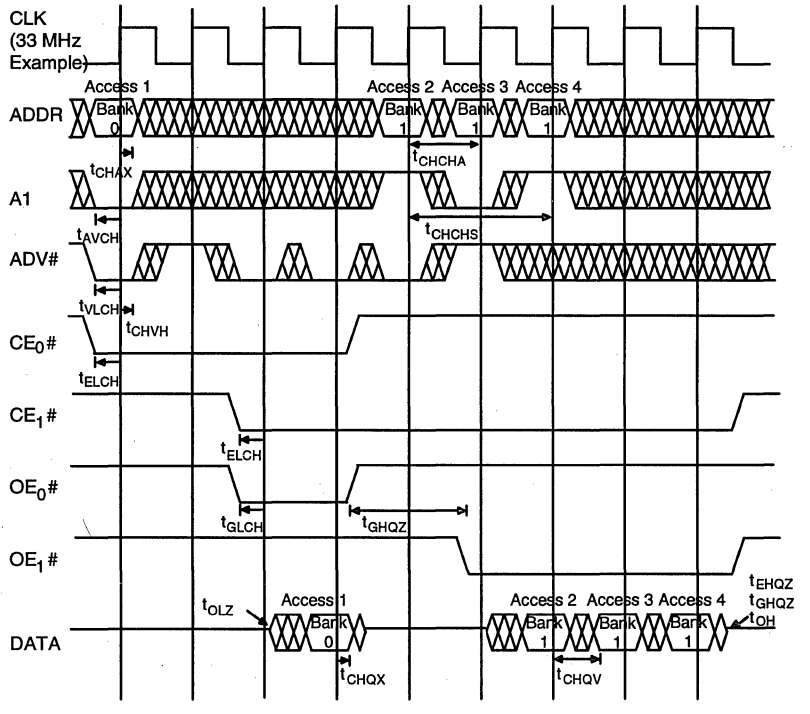
4.0 INTERFACING TO THE 28F016XS

The 28F016XS can interface to a wide range of CPUs and bus architectures. Glue logic is minimal and the performance enhancements are significant. Below are key considerations to keep in mind when interfacing to the 28F016XS:

- Clocking Options
- *Alternating-A_1* Access Rule
- *Same-A_1* Access Rule
- Optimizing Read Performance in x8 Mode
- Consecutive Accesses across Bank Boundaries
- Handling Asynchronous Write Cycles
- System Boot-Up out of the 28F016XS

4.1 Clocking Options

In choosing a CLK option, keep in mind that the 28F016XS operates at optimum performance with a CLK frequency at an upper SFI Configuration boundary (50 MHz and 33 MHz are two of four SFI Configuration upper boundaries for the 28F016XS-15 at 5.0V V_{CC}). See Section 2.3 for information about the SFI Configuration.



NOTE:
Refer to the 28F016XS datasheet for timing specifications.

Figure 23. Consecutive *Alternating-A$_1$* Accesses Crossing Bank Boundaries, One OE# per Bank (28F016XS-15, SFI Configuration = 2)

4.7 Handling Asynchronous Writes

The 28F016XS write interface is asynchronous, similar to other Intel flash memories. The 28F016XS's write interface is not pipelined, therefore a write cycle must complete before another begins.

When the interfacing CPU can execute a burst write cycle, the system logic needs to manage the write cycle, allowing only one write cycle to the 28F016XS at a time. Write-back caches, for example, support burst write cycles.

When the interfacing CPU does not support burst writes, the 28F016XS's asynchronous write interface is not an issue. The processor will only execute one write cycle at a time.

When executing a write cycle, the interfacing processor or bus will drive a write signal informing the system of the desired operation. Monitoring this signal, the interfacing logic can correctly transition into an appropriate state machine sequence. In this situation, the interfacing logic directly controls the 28F016XS's write control signals (Figure 24), just as with asynchronous Intel flash memories.

Application Note-384

1.0 INTRODUCTION

This application note discusses comparisons between the 28F016XD and DRAM memories. It also offers recommendations for determining compatibility between the 28F016XD and DRAM controllers, and provides suggestions for designing DRAM controllers with the 28F016XD in mind. The 28F016XD, an Intel 16-Mbit Flash memory component, retains full software backwards-compatibility with the 28F008SA and adds the following features:

- Multiplexed address/address interface with RAS# and CAS# control inputs
- SmartVoltage technology
- Internal 3.3V/5.0V

The 28F016XD leverages the existing DRAM controller in system designs and thereby minimizes the glue logic required to interface to flash memory. It is a 16-Mbit device, organized as 1 Mbyte x 16. The 28F016XD has ten row addresses and ten column addresses, multiplexed on inputs A_0-A_9.

The 28F016XD is fully non-volatile, giving it significant power and performance advantages over the traditional disk-plus-DRAM alternative. The 28F016XD does not lose data when power is removed from the device. By permanently storing and executing programs from the 28F016XD, the inherently slow disk drive-to-DRAM load delay is eliminated. The 28F016XD also does not require refresh cycles (although the 28F016XD will properly ignore any refresh cycles that are issued to it).

2.0 28F016XD COMPARISONS TO DRAM

The following sections discuss specific areas of comparison between the 28F016XD and 16-Mbit (1M x 16) DRAMs in 60 ns and 70 ns speed bins. Please reference the 28F016XD datasheet for a full description of the 28F016XD.

2.1 Voltage and Current Specifications

One obvious difference between the 28F016XD and DRAMs is that flash memory specifications reference a V_{PP} voltage, used with Data Write and Erase operations. All V_{PP}-related voltage and current specifications are unique to the 28F016XD. Note that the 28F016XD offers the option to connect V_{PP} either to 12.0V ± 5% or to 5.0V ± 10% (which may also be the V_{CC} operating voltage). The 28F016XD includes V_{CC} current specifications during Data Write, Erase and Erase Suspend operations. These operations are unique to flash memory; therefore, these specifications are not found in DRAM datasheets.

The 28F016XD specifies a V_{LKO} (lockout) voltage. This specification relates to circuitry within the flash memory that protects it from unwanted data alteration. The V_{LKO} specification is not found in DRAM datasheets. The 28F016XD also provides the deep power-down mode, not available on DRAMs. 28F016XD read, standby (CMOS), RAS#-only refresh and CAS#-before-RAS# refresh currents are lower than those seen with DRAMs.

A comparison between the 28F016XD and representative 16-Mbit DRAMs (valid at the time this application note was written) is shown in Table 1.

Table 2. 28F016XD Added/Revised DC Characteristics (Continued)

V_{CC} = 3.3V ± 0.3V, T_A = 0°C to + 70°C

Sym	Parameter	Min	Typ	Max	Unit	Test Condition
$I_{CC}7$	V_{CC} Standby Current (Self Refresh Mode)		40	55	mA	V_{CC} = V_{CC} Max RAS#, CAS# = V_{IL} I_{OUT} = 0 mA Inputs = V_{IL} or V_{IH}
I_{CCD}	V_{CC} Deep Power-Down Current		2	5	µA	RP# = GND ± 0.2V
I_{CCW}	V_{CC} Word Write Current		8	12	mA	Word Write in Progress V_{PP} = 12.0V ± 5%
			8	17	mA	Word Write in Progress V_{PP} = 5.0V ± 10%
I_{CCE}	V_{CC} Block Erase Current		6	12	mA	Block Erase in Progress V_{PP} = 12.0V ± 5%
			9	17	mA	Block Erase in Progress V_{PP} = 5.0V ± 10%
I_{CCES}	V_{CC} Erase Suspend Current		1	4	mA	RAS#, CAS# = V_{IH} Block Erase Suspended
I_{PPS}	V_{PP} Standby/Read Current		± 1	± 10	µA	$V_{PP} \leq V_{CC}$
			30	200	µA	$V_{PP} > V_{CC}$
I_{PPD}	V_{PP} Deep Power-Down Current		0.2	5	µA	RP# = GND ± 0.2V
I_{PPW}	V_{PP} Word Write Current		10	15	mA	V_{PP} = 12.0V ± 5% Word Write in Progress
			15	25	mA	V_{PP} = 5.0V ± 10% Word Write in Progress
I_{PPE}	V_{PP} Block Erase Current		4	10	mA	V_{PP} = 12.0V ± 5% Block Erase in Progress
			14	20	mA	V_{PP} = 5.0V ± 10% Block Erase in Progress
I_{PPES}	V_{PP} Erase Suspend Current		30	50	µA	Block Erase Suspended
V_{PPLK}	V_{PP} Erase/Write Lock Voltage	0.0		1.5	V	
$V_{PPH}1$	V_{PP} during Write/Erase Operations	4.5	5.0	5.5	V	
$V_{PPH}2$	V_{PP} during Write/Erase Operations	11.4	12.0	12.6	V	
V_{LKO}	V_{CC} Erase/Write Lock Voltage	2.0			V	

Table 3. 28F016XD Added/Revised DC Characteristics (Continued)

V_{CC} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Sym	Parameter	Min	Typ	Max	Unit	Test Condition
I_{PPS}	V_{PP} Standby/Read Current		± 1	± 10	µA	$V_{PP} \leq V_{CC}$
			30	200	µA	$V_{PP} > V_{CC}$
I_{PPD}	V_{PP} Deep Power-Down Current		0.2	5	µA	RP# = GND ± 0.2V
I_{PPW}	V_{PP} Word Write Current		7	12	mA	V_{PP} = 12.0V ± 5% Word Write in Progress
			17	22	mA	V_{PP} = 5.0V ± 10% Word Write in Progress
I_{PPE}	V_{PP} Block Erase Current		5	10	mA	V_{PP} = 12.0V ± 5% Block Erase in Progress
			16	20	mA	V_{PP} = 5.0V ± 10% Block Erase in Progress
I_{PPES}	V_{PP} Erase Suspend Current		30	50	µA	Block Erase Suspended
V_{PPLK}	V_{PP} Erase/Write Lock Voltage	0.0		1.5	V	
V_{PPH1}	V_{PP} during Write/Erase Operations	4.5	5.0	5.5	V	
V_{PPH2}	V_{PP} during Write/Erase Operations	11.4	12.0	12.6	V	
V_{LKO}	V_{CC} Erase/Write Lock Voltage	2.0			V	

2.2 Timing Specifications

28F016XD timing specifications are divided into the following categories in the datasheet:

- Common Parameters
- Read Cycle
- Write Cycle
- Read-Modify-Write Cycle
- Fast Page Mode Cycle (including fast page mode read-modify-write)
- Refresh Cycle (including refresh period)
- Miscellaneous

Many 28F016XD specifications match or improve on those of 60 ns and 70 ns DRAMs. Programming additional DRAM controller wait states will accommodate most slower 28F016XD specs.

In some cases, specifications that have identical values for both reads and writes to DRAM (such as RAS# and CAS# pulse widths and hold times), have been differentiated (separate specs for read and write) on the 28F016XD. This differentiation both accurately reflects 28F016XD functionality and improves the DRAM controller interface to 28F016XD, in some cases.

Common Parameters

Table 4 compares 28F016XD common parameters to DRAM, with incompatible specifications shaded for emphasis. Areas where the 28F016XD improves upon DRAM specifications are outlined in bold. Notice that the 28F016XD's RAS# precharge time specification is much shorter than that for DRAM, while the 28F016XD's CAS# precharge time specification is slightly longer. Also, the 28F016XD's row address hold time after RAS#, column address hold time after CAS# and CAS#-to-RAS# precharge time are slightly longer than those for DRAM.

Fast Page Mode Cycle Specifications

28F016XD fast page mode cycle specification incompatibilities compared to DRAM have the same root causes as the read and write cycle incompatibilities described earlier. Fast page mode read-modify-write cycles to the 28F016XD will not occur in the majority of applications.

Table 8 compares 28F016XD fast page mode cycle specifications to DRAM, with incompatible specifications shaded for emphasis. Areas where the 28F016XD improves upon DRAM specifications are outlined in bold.

Refresh Cycle Specifications

Flash memory does not require refresh to retain stored data contents. However, by interfacing to a DRAM controller, it will automatically receive the same refresh cycles that DRAM receives. The 28F016XD supports all common refresh cycles; CAS#-before-RAS#, RAS#-only, hidden and self-refresh. In these modes, it will either drive or float the data bus just as a DRAM would. Refresh cycles have no other effect on 28F016XD stored data.

Table 9 compares 28F016XD refresh cycle specifications to DRAM, with incompatible specifications shaded for emphasis. Areas where the 28F016XD improves upon DRAM specifications are outlined in bold.

Table 8. 28F016XD Fast Page Mode Cycle Specifications Compared to 60–70 ns 16-Mbit DRAM

Sym	Description	DRAM (3.3V)	28F016XD (3.3V)	DRAM (5.0V)	28F016XD (5.0V)
$t_{PC(R)}$	Fast Page Mode Cycle Time (Reads) (min)	35-45 ns	75 ns	35-45 ns	65 ns
$t_{PC(W)}$	Fast Page Mode Cycle Time (Writes) (min)	35-45 ns	80 ns	35-45 ns	65 ns
$t_{RASP(R)}$	RAS# Pulse Width (Reads) (min)	60-70 ns	95 ns	60 ns	85 ns
$t_{RASP(W)}$	RAS# Pulse Width (Writes) (min)	60-70 ns	80 ns	60 ns	65 ns
t_{CPA}	Access Time from CAS# Precharge (max)	35-40 ns	85 ns	35-40 ns	70 ns
t_{CPW}	WE# Delay Time from CAS# Precharge (min)	10 ns	0 ns	10 ns	0 ns
$t_{CPRH(R)}$	RAS# Hold Time from CAS# Precharge (reads) (min)	35 ns	75 ns	35-45 ns	65 ns
$t_{CPRH(W)}$	RAS# Hold Time from CAS# Precharge (writes) (min)	35 ns	80 ns	35-45 ns	65 ns
t_{PRWC}	Fast Page Mode Read-Modify-Write Cycle Time (min)	85 ns	170 ns	85-100 ns	145 ns

Table 10 28F016XD Added/Revised AC Timings (Continued)

V_{CC} = 3.3V ± 0.3V, V_{PP} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Sym	Parameter	Min	Typ	Max	Units
$t_{WHRH}1$	Word Write Time	TBD	35	TBD	µs
$t_{WHRH}3$	Block Write Time	TBD	1.2	TBD	sec
	Block Erase Time	TBD	1.4	TBD	sec
	Erase Suspend Latency Time to Read	1.0	12	75	µs

V_{CC} = 3.3V ± 0.3V, V_{PP} = 12.0V ± 0.6V, T_A = 0°C to +70°C

Sym	Parameter	Min	Typ	Max	Units
$t_{WHRH}1$	Word Write Time	5	9	TBD	µs
$t_{WHRH}3$	Block Write Time	TBD	0.3	1.0	sec
	Block Erase Time	0.3	0.8	10	sec
	Erase Suspend Latency Time to Read	1.0	9	55	µs

Table 11. 28F016XD Added/Revised AC Timings

V_{CC} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Versions	28F016XD - 85		
Parameter	**Min**	**Max**	**Unit**
RP# High to RAS# going low	300		ns
RP# Set-Up to WE# going low	300		ns
V_{PP} Set-Up to CAS# high at end of write cycle	100		ns
WE# High to RY/BY# going low		100	ns
RP# Hold from Valid Status Register Data and RY/BY# High	0		ns
V_{PP} Hold from Valid Status Register Data and RY/BY# High	0		ns
V_{CC} at 4.5V (minimum) to RP# High	2		µs

Table 11 28F016XD Added/Revised AC Timings (Continued)

V_{CC} = 5.0V ± 0.5V, V_{PP} = 5.0V ± 0.5V, T_A = 0°C to +70°C

Sym	Parameter	Min	Typ	Max	Units
$t_{WHRH}1$	Word Write Time	TBD	25	TBD	µs
$t_{WHRH}3$	Block Write Time	TBD	0.85	TBD	sec
	Block Erase Time	TBD	1.0	TBD	sec
	Erase Suspend Latency Time to Read	1.0	9	55	µs

V_{CC} = 5.0V ± 0.5V, V_{PP} = 12.0V ± 0.6V, T_A = 0°C to +70°C

Sym	Parameter	Min	Typ	Max	Units
$t_{WHRH}1$	Word Write Time	4.5	6	TBD	µs
$t_{WHRH}3$	Block Write Time	TBD	0.2	1.0	sec
	Block Erase Time	0.3	0.6	10	sec
	Erase Suspend Latency Time to Read	1.0	7	40	µs

2.3 Package and Pinout

Although the 28F016XD includes all necessary inputs and outputs for interfacing to DRAM controllers, its pinout and package do not match those of DRAMs but instead evolve from other 16-Mbit Intel flash memories. The 28F016XD uses a 56-lead TSOP package, with pinout shown in Figure 1 and package dimensions shown in Figure 2.

Comparable 1M x 16 (16-Mbit) DRAMs use two packages, a 42-lead SOJ and 44-lead TSOP. Examples of these DRAM pinouts are shown in Figures 3 and 4.

Table 12 summarizes pinout comparisons between the 28F016XD in 56-lead TSOP and various DRAM package options.

If compatibility between the 28F016XD and DRAM "footprints" is desired, 28F016XD flash memories can be placed on DRAM-compatible SIMMs. Please see the Additional Information section of this application note for documentation that covers this topic in more detail.

intel®

Application Note-377

28F016SA Commands

The 28F016SA command set is a superset of the 28F008SA command set, giving existing 28F008SA code the ability to run on the 28F016SA with minimal modifications.

28F008SA-Compatible Commands

00	invalid/reserved
20	single block erase
40	word/byte write
50	clear status registers
70	read CSR
90	read ID codes
B0	erase suspend
D0	confirm/resume
FF	read flash array

28F016SA Performance-Enhancement Commands

0C	page buffer write to flash
71	read GSR and BSRs (i.e. the ESR)
72	page buffer swap
74	single load to page buffer
75	read page buffer
77	lock block
80	abort
96,xx	RY/BY# reconfiguration and SFI configuration (28F016XS)
97	upload BSRs with lock bit
99	upload device information
A7	erase all unlocked blocks
E0	sequential load to page buffer
F0	sleep
FB	two-byte write

28F016XD and 28F016XS Feature Sets

The following features are not supported on the 28F016XD and 28F016XS Embedded Flash RAM memories (as compared to the 28F016SA/SV/32SA FlashFile™ memories):

- All page buffer operations (read, load, program, Upload Device Information)
- Command queuing
- Erase All Unlocked Blocks and Two-Byte Write
- Software Sleep and Abort
- RY/BY# reconfiguration via the Device Configuration command

intel®

Application Note-343

Memory Cards

Many computer manufacturers are pursuing the IC memory card to incorporate a removable mass storage medium. This is an ideal application for the Intel Flash Memory TSOP, due to the package's minimal height.

Solid-State Memory Alternatives

ROM and SRAM are currently the dominant IC card memory technologies. ROM has the advantage of being inexpensive, but is not changeable. When newer software revisions (e.g., Lotus* 123, Windows, etc.) are available, the user must buy a new ROM card for each upgrade. Intel Flash Memory's reprogrammability minimizes the user's expense and the OEM's inventory risk.

SRAM is reprogrammable but requires batteries to maintain data, risking data loss. Like magnetic disks, flash memory is truly nonvolatile and thus has very long storage time with power off. Additionally, SRAM is expensive and not a high density solution. Intel Flash Memory provides a denser, more cost effective and reliable solution.

System level cost is about the same for Intel Flash Memory and SRAM + battery—

Flash memory requires 12V for programming and erasing. If a 12V supply is not available, 5V can easily be boosted. (See Application Note AP-316.) SRAM + battery requires battery state detect circuitry.

Card level cost differences are substantial (Figure 3)—

SRAM must have a battery to retain data. It also requires a V_{CC} monitor and Write Lockout circuitry. Intel's Flash Memory only requires Write Lockout circuitry (switching V_{PP} to 0V is an alternative write protect). This leads to increased area for memory components. More importantly, Intel's Flash Memory density is 4 times that of static RAM, yielding lower cost per bit.

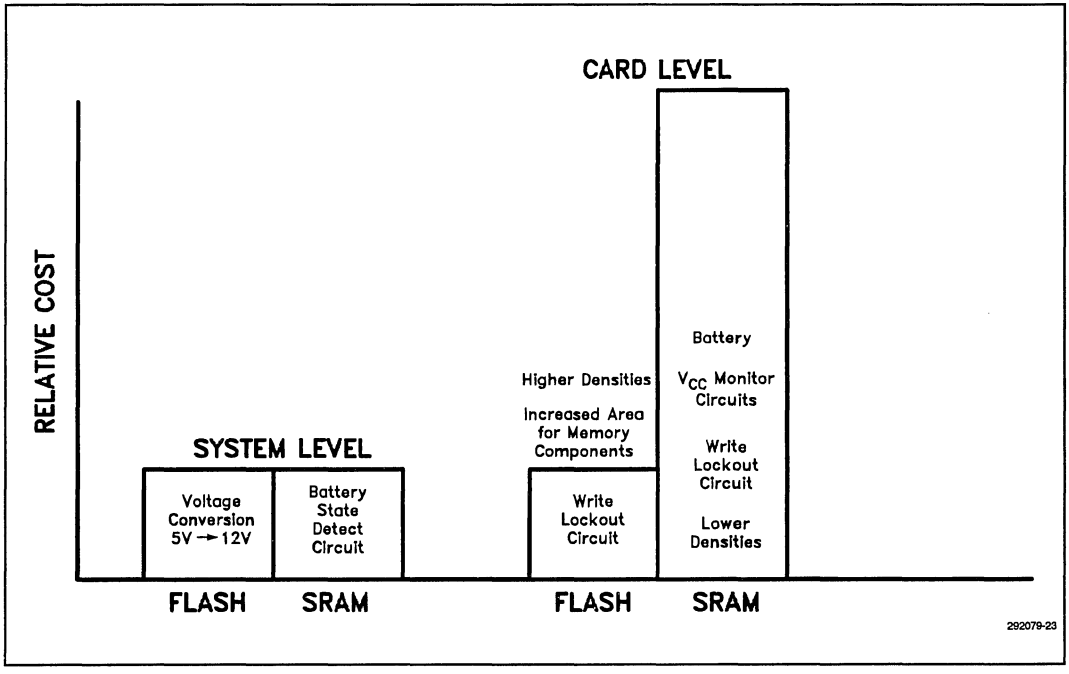


Figure 3. Support Circuitry Cost Comparison

intel®

Application Note-325

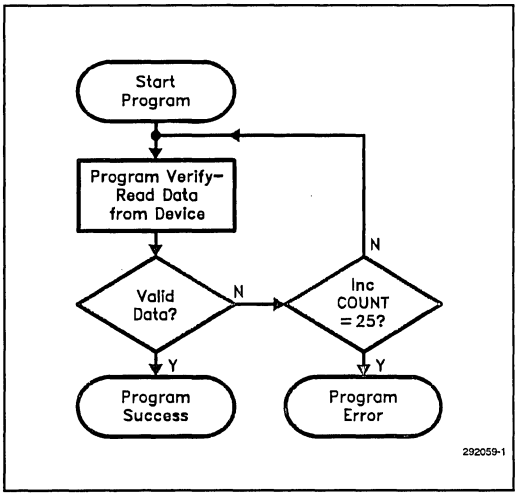


Figure 1. The flow chart shows the fundamental nature of an adaptive algorithm. Based on the outcome of program verification, the flow may loop back for another program operation.

Moving Charge and Other Factors You Should Know

This section discusses the mechanics of flash memory programming. For most system designers, transistor-level discussions were last heard in college. We may recall that DRAM consists of a storage capacitor and a transistor. We remember this clearly because failure to refresh that capacitor causes systems to malfunction. **In like fashion, one should understand the fundamentals of flash memory reprogramming. The understanding will enable error-free memory operation and reliable system performance.**

In simplest terms, each data bit equates to a memory cell. Intel's flash memory uses one transistor per cell with the smallest possible architecture. This delivers the lowest cost per bit and highest capacity, levering system software (rather than bulky, complex cells) for reprogramming control.

Figure 2 shows a simplified cross section of Intel's flash memory transistor. Note the structure; the cell is a stacked gate MOS transistor. An isolated floating gate stores the memory charge. The floating gate consists of a layer of (conductive) polysilicon surrounded by (non-conductive) oxide layers.

On a DRAM cell, each transistor connects to a capacitor which stores the memory charge. The major difference between flash memory and DRAM derives from their cell structure. The DRAM cell loses its charge if not refreshed within a few milliseconds. On the other hand, the flash memory floating gate maintains its charge as a fully nonvolatile memory, similar to ROM or EPROM. The structure is isolated and insulated by the field and gate oxides-hence the name "floating" gate.

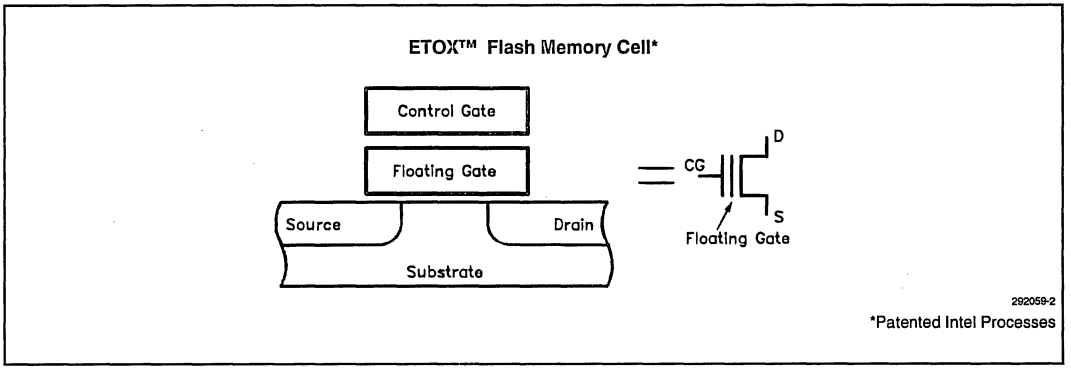


Figure 2. Simplicity of design assures increasing densities, manufacturability and reliability. These are the attributes that drive mainstream memories.

intel®

Engineering Report-33

Cycling Performance

Cycling durability has made tremendous gains from early first-generation devices, such as the 28F256-170P1C2 with a 100-cycle specification. Current products, such as Intel's ETOX III 28F008SA, are registering above 1,000,000 cycles. Experiments with these units have been conducted to prove their reliability. These parts typically do not produce hard failures, instead they see program and erase timings push out.

ETOX cycling longevity has progressed due to continued improvements in tunnel-oxide quality. As oxide layers become cleaner and more consistent, cycling-induced electron trapping minimizes. Reduced electron trapping helps keep V_t distributions tight. Additionally, reduced electron trapping improves program and erase pulsing efficiencies. This improved efficiency lessens the number of pulses required for a given operation to succeed, which in turn also reduces electron trapping. Since trapping is the primary source of program/erase time pushout (i.e., more pulses required), higher quality oxides reduce this degradation and extend cycling performance.

Improved source-to-floating gate coupling consistency via enhanced cell/array uniformity also extends cycling longevity by lowering the number of erase pulses required. Additionally, the Intel ETOX cell, in combination with optimal WSM control, typically requires a very low number of pulse repetitions to achieve a program or erase state (typically one pulse for programming). This also lessens the pulse count, extending cycling livelihood.

Improvements in isolation, both from cell to cell and around the floating gate, have mitigated charge retention and data disturb concerns.

Read Access

Intel Flash Memory architecture has evolved by leaps and bounds since its inception. Read access time (t_{ACC}) has made significant gains, while maintaining very high levels of noise immunity. The 27F64's fastest speed bin was 150 ns, while the 28F016SA will offer a 5.0V 70-ns/80-ns version with ±5%/10% V_{CC} tolerance and a 3.3V ±0.3V 120-ns version.

Double metal, introduced on ETOX III, has been the biggest contributing factor toward improving read access times. This enhancement reduces wordline and bitline resistance, thereby enabling faster cell turn-on and sensing paths during read operations. Continually-improved circuit designs also contribute to shorter read timings. Again, cell and array compaction enables inclusion of new/additional circuitry that enhance read performance, and will also allow future 3.3V parts with access times much faster than their 5.0V predecessors.

FLASH vs. OTHER SEMICONDUCTOR MEMORY TECHNOLOGIES

Intel's scaling advances in flash memory manufacturing and design provide optimal cell/array compaction. In roughly twenty-two years, Intel non-volatile memory density has grown from 2,048 bits to 16,777,216 bits, a factor of 8192x. Figure 15 compares other memory types to show relative density progression. The fast ramp in ETOX flash memory density results from its similarity to EPROM.

Figure 16 illustrates the relationship between cell sizes of different memory types and minimum geometries. As dimensions scale, certain memory types become cell-size limited (i.e. some components cannot shrink proportionally). The memory cost-per-bit learning curve shows flash in a strong position. This curve, shown in Figure 17, reflects how Intel's experience reduces cost for increased memory density.

Since the late 1980's, a new memory sub-system has arrived on the market offering an alternative to high-density file system media. Intel's Series 2+ Flash Memory Cards take advantage of the 28F016SA and its third-generation architecture to provide card densities of up to 40 Mbytes and new functionality. This relatively new technology offers a solid-state file system (Figure 18) that will double in density with new ETOX generations.

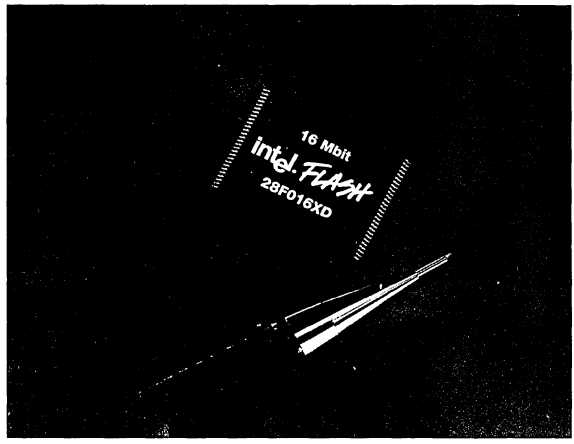
intel®

28F016XD
Embedded Flash RAM
Product Brief

Intel 28F016XD Embedded Flash RAM Product Brief

Product Highlights

- Innovative embedded flash RAM device:
  - DRAM-like system interface
  - Nonvolatile, updatable and low power
- Better price and performance than DRAM + ROM and DRAM + HDD:
  - Offers higher integration and 3.3V capability
  - Requires lower power and less board space
- Ideal for embedded applications:
  - Datacom
  - Office automation
  - Telecom
  - Computing
  - Games
- Simple DRAM-like interface:
  - Allows use of standard DRAM controllers
  - Optimizes time to market
  - Sits directly on a cachable bus
- Ideal for portable applications
- Easy interface to systems using embedded processors:
  - For i960®KX, i960CS, i960JX and Intel386™EX CPU-based systems
- Internally partitioned into 32 software-lockable, 64-KB blocks
- SmartVoltage feature:
  - Supports both 5V and 12V device writes and 5V or 3.3V device reads
- First flash memory device with DRAM-like interface



Product Description

The Intel 28F016XD embedded flash RAM is an innovative 16Mb memory component that combines a DRAM-like system interface with the nonvolatility, updatability and lower power of flash memory. The 28F016XD embedded flash RAM is an ideal solution for embedded applications where redundant code DRAM + ROM or code DRAM + HDD were used for code execution/storage memory.

Its simple DRAM-like interface allows use of standard DRAM controllers, optimizing time to market. In addition, the Intel 28F016XD embedded flash RAM is in-system updatable, reducing the risk of early manufacturing as compared to DRAM + ROM and DRAM + OTP EPROM options. The Intel 28F016XD embedded flash RAM can easily interface to systems using such embedded processors as the i960®KX, i960X, i960JX and the Intel386™EX CPU-based systems. Several vendors offer 28F016XD embedded flash RAM SIMM modules, making it easier to upgrade code DRAM designs to embedded flash RAM.

The Intel 28F016XD embedded flash RAM is internally partitioned into 32 software-lockable, 64-KB blocks. Like Intel's 28F016SV SmartVoltage device, the 28F016XD embedded flash RAM supports both 5V and 12V device writes as well as 5V or 3.3V device reads. Because the Intel 28F016XS embedded flash RAM is a nonvolatile, code storage and execution solution, there is no need for refresh, redundant memory or HDD "spin-up" latency when returning from deep power-down mode. It also enables instant-on system design.

The Intel 28F016XD embedded flash RAM, which is the first flash memory device with a DRAM-like interface, is particularly well suited for data communications, office automation, telecommunications, computing and games.

Printed in USA/0395/7.5K/ASI/LK

Order Number: 297547-002

Order Number: 210830-014
Printed in USA/0195/21K/RRD/TS
Memory Products