



AP-605

**APPLICATION
NOTE**

**Implementing a
Resident Flash Disk
with an Intel386™ EX
Embedded Processor**

**TONY SHABERMAN
TECHNICAL MARKETING
ENGINEER**

September 1995

Order Number: 292157-001



Information in this document is provided solely to enable use of Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

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

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

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

MDS is an ordering code only and is not used as a product name or trademark of Intel Corporation.

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

*Other brands and names are the property of their respective owners.

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

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641
or call 1-800-879-4683



1.0 INTRODUCTION

The Intel386™ microprocessor family has gained a wide acceptance in the world of embedded applications. One reason for this acceptance is DOS-compatibility. A DOS-based design provides an easy, cost-effective means to develop, test, debug and port embedded application code.

As embedded system designers take advantage of DOS capability, a revolutionary system architecture is required to meet space and power requirements. The requirements of this revolutionary system are:

- An architecture that is not bounded by what has been done before with existing memory architectures, but free to meet the demanding requirements of embedded end-users.
- An architecture which is 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 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

2157_01

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 allow cost-effective field updates and provide quick time-to-market solutions in most applications.

Flash memory enables the design of completely new types of computers that fit in the palm of your hand by replacing many of the code and storage functions of other memory types. These hand-held designs feature flash memory resident on the embedded system's motherboard. In these designs flash memory is typically arranged in an array, called a Resident Flash Array (RFA). There are two types of RFAs (see Figure 2). The first type of RFA is for storing eXecute-In-Place (XIP) code, such as ROMed DOS, in the system's memory map. Used in this manner, the RFA is called a Resident Flash for XIP (RFX). The second type of RFA performs file and program storage functions traditionally provided by a hard disk. Used in this manner the RFA is called a Resident Flash Disk (RFD). This application note details the hardware and software components necessary to implement a resident flash disk. A hardware design example for the Intel386 EX is also provided.

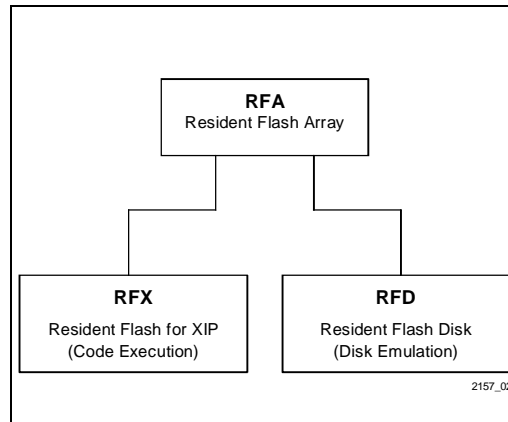


Figure 2. Types of Resident Flash Arrays

2.0 RESIDENT FLASH DISK

Solid-state disks are becoming a reality with the ever-decreasing costs of flash memory components. These devices provide erase blocking, automated write and erase algorithms and a straightforward interface to the CPU system bus. Many systems will choose the versatility and reliability of solid-state media exclusively, or in conjunction with traditional magnetic media.

A Resident Flash Disk based on Intel's 16-Mbit FlashFile memories provides distinct advantages over traditional Hard Disk Drives (HDD). For example:

- Lower Power Requirements
 - 3.3V Operation
 - 1 μ A per Chip in Deep Power-Down Mode
 - \ll 1 mA of Static Current
- Higher Performance
- Lower Weight
- Superior Reliability
- Superior Ruggedness
- Higher Integration and a More Compact Footprint

Many embedded designs do not require the high densities offered by today's typical HDDs. Flash memory arrays provide a more cost-effective mass storage solution when compared against the "floor cost" of a HDD. As PC architecture-based systems become specialized in their functions and convert to direct-execute software solutions, their mass storage requirements will similarly decrease. The conversion to an all solid-state flash memory storage solution becomes an attractive (and in some cases necessary) alternative.

3.0 RFD SOFTWARE REQUIREMENTS

Software plays a key role when implementing a resident flash disk. The following sections briefly describe the differences between flash memory and traditional magnetic media, and the software used to easily manage these differences. For more detailed information on RFD software issues, refer to AP-618, "Software Concerns of Implementing a Resident Flash Disk."

3.1 Magnetic Media

DOS-based systems have random write and erase capabilities when accessing magnetic media. This means a sector of the magnetic media can be programmed,

reprogrammed, and erased at any time. A File Allocation Table (FAT) is used to manage files. This table is updated anytime a file is created, updated or erased.

3.2 Flash Media

Flash memory architecture is very different from magnetic media. Flash memory is subdivided into separate erase blocks. When flash memory is completely erased, every bit is set to a "1." A completely erased device will allow writes to any location. However, once bits within a block have been written to zero, the whole block must be erased before those same bits can be written back to a "1." This characteristic of flash memory prevents the direct use of a traditional FAT and requires an alternative flash media manger to manage the files in a RFD.

3.3 Flash Media Managers

There are many different flash media managers available today. Each flash media manager has different features tailored toward specific applications. For disk drive replacement in a DOS environment, the Flash Translation Layer (FTL) is the recommended selection. For more information on alternative flash media managers call Intel's Application Support FaxBack* service at 800-628-2283 and request document #2258. A list of current FTL developers is provided in Appendix D.

3.4 Flash Translation Layer

The Flash Translation Layer is a sector-based media manger that uses an existing sector-based file system, such as DOS FAT, to provide the upper level file handling capabilities. By translating received requests from DOS, a FTL driver appears as a normal sector based drive to the upper layer software. Upper layer software expects to be able to modify these sectors at any time. As indicated in Section 3.2, flash requires that a block be erased before files within that block are modified. When the upper layer software tries to re-write a sector, FTL remaps the request to a free area of flash. These remapped sectors are treated as logical read/write blocks rather than physical sectors. FTL subdivides each flash block into smaller read/write blocks. Each read/write block is the same size as a sector (typically 512 bytes). Within a 28F008SA flash device there are 16 64-Kbyte blocks. When using a 512-byte sector size, each 64-Kbyte block divides into 128 read/write blocks (see Figure 3).



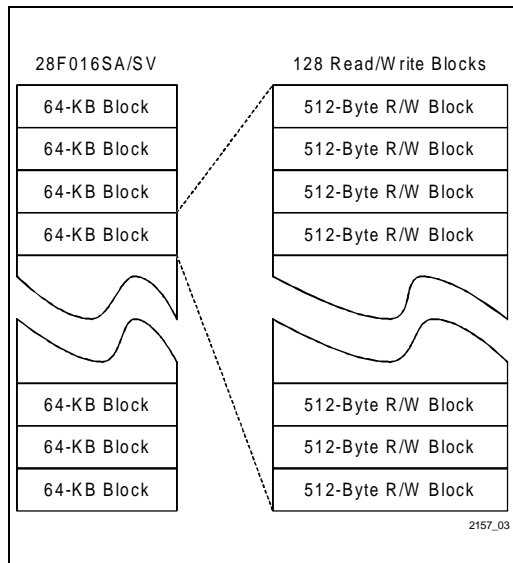


Figure 3. Read/Write Block Map

4.0 FTL IMPLEMENTATIONS

There are three primary implementations being provided by FTL developers today (see Figure 4). These implementations are divided into separate layers and have differences that are described in the following sections.

4.1 FTL for Card Services

Card Services is the layer responsible for allocating system resources for a PCMCIA card. With the appropriate client driver, card services manages the support for any memory or I/O card. This software layer would be used in a system that requires support for I/O cards.

Socket Services is the driver that configures the window that accesses the RFD. Socket services is a PCMCIA-defined standard that was originally developed to configure PCMCIA sockets. Even though there isn't a socket for a RFD, many FTL developers still refer to this driver as socket services because it follows the PCMCIA

standard. The PCMCIA socket services queries the inserted card for attribute information. Since the RFD is nonremovable, all the attribute information about the RFD is fixed. The RFD socket services would simply include all of these fixed values. Most FTL developers will provide sample source code for socket services which can be customized for a particular design.

Sliding Window Logic is the hardware that controls the window for accessing flash. Systems that need to support PCMCIA memory cards require a PCMCIA socket controller such as Intel's 82365SL PCIC. This controller manages access to flash cards through a sliding window. A RFD can also be accessed through a sliding window. Systems that already implement a PCIC for memory card support can use that device to access the RFD. If a PCIC is not used, the sliding window can be implemented with a page register. For more information regarding the sliding window hardware refer to Section 5.4.

Resident Flash refers to the physical flash components. Intel's FlashFile memory family is the best fit for implementing resident flash disks.

4.2 FTL for Socket Services

The only difference between FTL for socket services and FTL for card services is the elimination of the card services layer. Removing the card services layer sacrifices support for I/O cards. If the system only uses memory cards or a RFD, card services is not needed and precious memory space can be saved by not installing the card service software.

4.3 FTL for Resident Flash

It is possible to access the resident flash without using a sliding window. This is done by mapping the resident flash into extended memory. The memory is accessed linearly and requires putting the Intel386 EX CPU into protected mode. This is the least expensive solution eliminating the need for extra sliding window logic. This solution requires a FTL designed specifically to access flash memory that is linearly mapped into extended memory. For more information regarding this "protected mode FTL," contact the FTL developer. A list of FTL developers is provided in Appendix D.



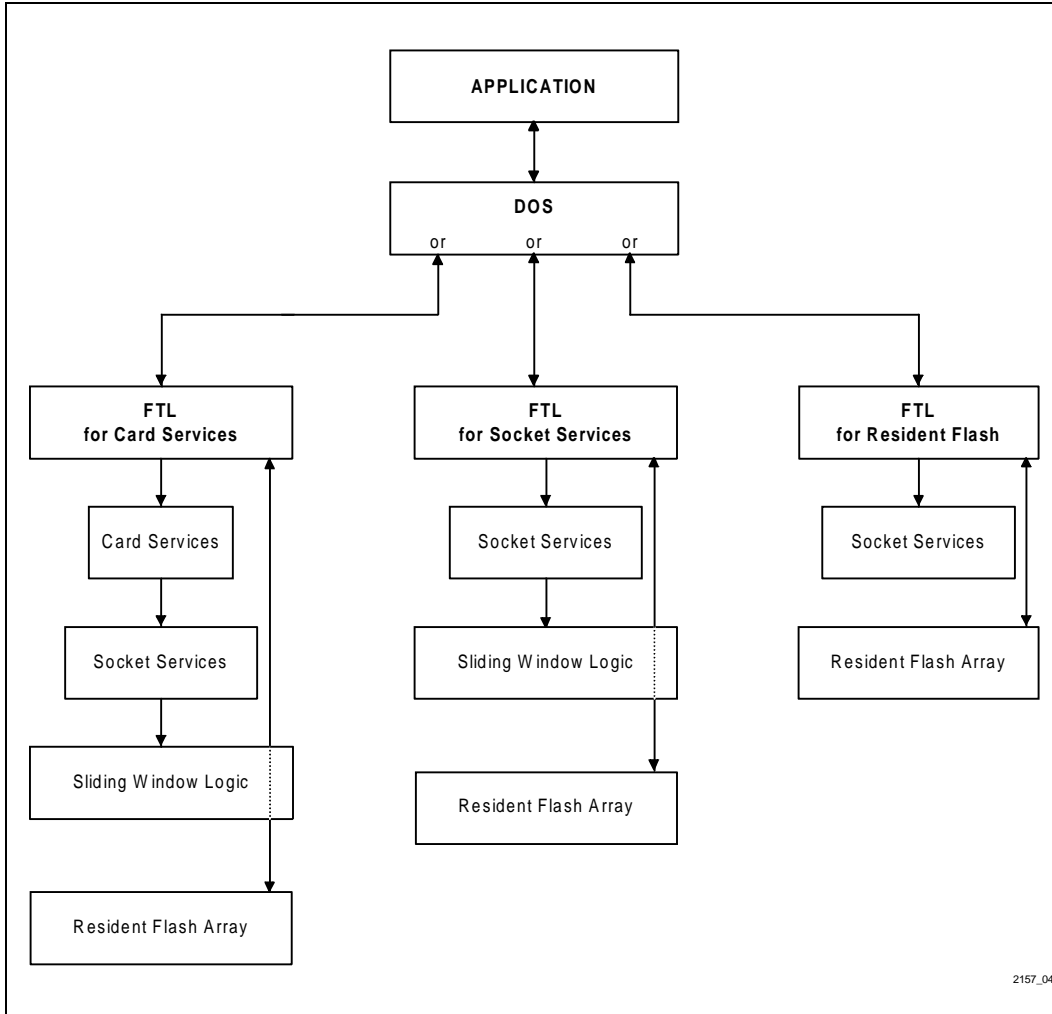


Figure 4. FTL Implementations

4.4 Wear Leveling

Wear leveling is a technique used by FTL developers to extend the life of the flash device. Wear leveling evenly distributes erase operations to all blocks of the device. For example, without wear leveling, a 64-Kbyte file written to the same flash block in a 28F016SV every hour has a block cycling rate of one cycle per hour. This is because that block must be erased each time the file is

written. If the file system implements wear leveling over all 32 flash blocks, the file might be written to a different block every hour until all 32 blocks have been written. At this point each block now has a cycling rate of 1 cycle per 32 hours. Most flash devices support a minimum of 100,000 erase cycles per block and most FTL file systems implement wear leveling. Intel's 16-Mbit FlashFile components are rated for 1,000,000 cycles.



4.5 Flash Clean-Up

Once a flash bit is written to a 0, it cannot be written back to a 1 without erasing the entire flash erase block. Therefore, whenever a file gets updated the flash media manager cannot rewrite that file to the same location as the original. Instead the file gets written to a new location, and the old file is marked “dirty.” As the flash media manager continues to update files, more areas of flash become dirty. At some point, the flash media manager will decide to convert the dirty read/write blocks into useable free space. This procedure is called clean-up. The file manager copies the remaining clean read/write blocks in a dirty flash erase block to another free flash erase block. Then the dirty erase block is erased. All FTL developers must implement some form of clean-up.

4.6 FTL Installation

Most FTL developers allow the flexibility to load FTL several different ways. It can be loaded as a device driver, TSR, or ROM BIOS extension. For more information on the installation of FTL, contact the FTL developer. A list of FTL developers is provided in Appendix D.

5.0 RFD HARDWARE REQUIREMENTS

This section describes the general hardware requirements for implementing a RFD. These concepts can be easily incorporated into new chip-set or ASIC designs to take advantage of the benefits that flash memory brings to embedded designs.

5.1 FlashFile Memory Overview

Intel’s FlashFile memories are revolutionary architectures which are the ideal choice for designing embedded mass storage data/file flash memory systems. With innovative capabilities, low-power operation and high read/write performance, FlashFile memory enables the design of truly mobile, high-performance personal computing and communications products.

The 28F008SA, 28F016SA, 28F016SV, and DD28F032SA are the four members of Intel’s FlashFile memory family. Table 1 below lists the features of each of the FlashFile devices. The 28F008SA is the smallest density and smallest feature set member of the FlashFile memory family. The 28F016SA, 28F016SV, and

DD28F032SA are very high-density, high-performance nonvolatile read/write solutions ideal for solid-state storage applications. They have symmetrically-blocked architectures (100% compatible with the 28F008SA 8-Mbit FlashFile memory), very high cycling ability, low power 3.3V operation, very fast write and read performance, and selective block locking. These features provide a highly flexible memory component suitable for high-density Resident Flash Arrays. The dual read voltage enables the design of memory cards which can be interchangeably read/written in 3.3V and 5.0V systems. Their x8/x16 architecture allows the optimization of memory to processor interface. The flexible block locking option enables bundling of executable application software within the RFA. The 28F016SA, 28F016SV, and DD28F032SA are manufactured on Intel’s 0.6 μm ETOX™ IV process technology.

5.2 SmartVoltage Technology

The difference between the 28F016SA and the 28F016SV is SmartVoltage technology. The 28F016SV incorporates SmartVoltage technology, providing read operation at both $V_{CC} = 3.3\text{V}$ and $V_{CC} = 5.0\text{V}$ and program/erase capability at $V_{PP} = 12.0\text{V}$ or $V_{PP} = 5.0\text{V}$. Reading at $V_{CC} = 3.3\text{V}$, the 28F016SV consumes approximately one-third the power consumption that it would at 5.0V V_{CC} , while 5.0V V_{CC} provides the highest read performance capability. $V_{PP} = 5.0\text{V}$ operation eliminates the need for a separate 12.0V converter, while $V_{PP} = 12.0\text{V}$ 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.

5.3 28F016XD

A RFD may also be implemented with a 28F016XD. The 28F016XD is a member of the Embedded Flash RAM Family. It has a DRAM interface ideal for interfacing to a system with an integrated DRAM controller.

5.4 Design Example

In order to illustrate the required hardware components of a RFD, a design example is provided. There are 4 main components used in this design. An Intel386 EX Embedded Processor, 28F016SA FlashFile memory, a

generic 22V10 PLD, and a generic '374 latch. A block diagram of this example is provided in Figure 5. In this design example the RFD is running off of the Intel386 EX local bus. The same concepts can also be applied to implement a RFD for an expansion bus like ISA or PC/104.

5.5 Flash Control Logic

The logic required to interface the Intel386 EX embedded processor to Intel flash devices is very simple. For a complete analysis of the interface, refer to AP-609 "Interfacing the Intel386 EX Embedded Processor to Intel Flash." Some logic is required because of some mismatched timings between the Intel386 EX and Intel Flash. Since most design optimizations reduce glue logic, future interface designs may not require the interface described. As shown in Figure 5, the PLD controls WE# to flash, and EN# to a buffer.

5.5.1 BUFFER

A buffer prevents bus contention on read followed by write cycles. After a read from flash, it takes the 28F016SA 30 ns to float its data lines. This becomes an issue on a read followed by a write cycle because the Intel386 EX CPU drives data in the middle of the first T-state. Adding a buffer and disabling it during this

period of potential bus contention solves this issue. Enable logic for the buffer is also required.

5.5.2 WE# CONTROL

In addition to controlling the buffer, the PLD also needs to pull WE# high early. The 28F016SA requires 10 ns of address hold from WE# high. The Intel386 EX CPU will not provide this so WE# must be controlled by a PLD. Pulling WE# high early meets address hold time.

5.5.3 BYTE WRITES

Most FTL solutions require the ability to do byte writes. To provide this capability, in this design example the x16 data bus is created by placing two flash devices side by side in a x8 configuration. BHE# and BLE# are generated by the Intel386 EX CPU. The PLD decodes these signals and generates a CE# for the appropriate flash device.

In this application example, the byte lanes of the x16 bus are controlled by BHE# and BLE#. However, the RFD could be implemented with a x8 data bus. The performance of a x8 solution is obviously less than a x16 solution, however controlling byte lanes is no longer required.

Table 1. Component Feature Overview

	28F008SA	28F016SA/DD28F032SA	28F016SV	28F016XD
Symmetrical Blocking	X	X	X	X
On-Chip Automation	X	X	X	X
Fast 3.3V Read		X	X	X
Page Buffers		X	X	
x16 Interface		X	X	X
Command Queuing		X	X	
Block Locking		X	X	X
SmartVoltage			X	X
DRAM Interface				X

NOTE:

For more information about the different features, refer to the component datasheets or the 16-Mbit Flash Product Family User's Manual, or FLASHBuilder. FLASHBuilder is a software application which provides info on all 16-Mbit components. Refer to Section 6.1 for all order numbers.



Table 2. Signal Description

Name	Description	Generated By	Connected To
A[13:1]	Lower Address	CPU	Flash A[12:0]
CLK2	50 MHz Clock	Oscillator	PLD, CPU CLK2
M/IO#	Memory/IO	CPU	PLD
ADS#	Address Status	CPU	PLD
W/R#	Write/Read	CPU	PLD
READY#	Ready	CPU	PLD
CS0#	Chip Select 0	CPU (page register)	PLD
CS1#	Chip Select 1	CPU (sliding window select)	PLD, Latch OC#
RD#	Read Enable	CPU	Flash OE#, Buffer DIR
BHE#	Byte High Enable	CPU	PLD
BLE#	Byte Low Enable	CPU	PLD
CEL#	Chip Enable Low Byte	PLD	Low Byte Flash CE#
CEH#	Chip Enable High Byte	PLD	High Byte Flash CE#
D[15:0]	Data Bus	CPU, Buffer D[15:0]	Buffer D[15:0], Latch D[7:0]
DIR	Buffer Direction	CPU RD#	Buffer DIR
EN#	Buffer Enable	PLD	Buffer EN#
RP#	Reset Power-Down	System PWRGD	Flash RP#
PWRGD	System Power Good	System PWRGD	Flash RP#, PLD
RESET	CPU Reset	PLD	CPU Reset
WE#	Write Enable	PLD	Flash WE#
A[20:13]	Upper Address	Latch	Flash A[20:13]
OC#	Output Control	CS1#	Latch OC#
CLK	Latch Clock	PLD	Latch CLK

5.6 Sliding Window Logic

This section describes the RFD implementation using sliding window logic. Figure 6 shows a typical memory map of a system including the RFD memory. In real mode DOS, all memory must be accessed under the 1-Mbyte boundary. In order to access the RFD memory array, it is necessary to map that memory into pages and read those pages through a RFD sliding window placed under the 1-Mbyte boundary. This window is called a sliding window since it slides over a large memory space. In hardware, all that is required to implement this sliding window is a latch which is used as a register

(see Figure 5). The register holds the upper address lines connected to the flash device. These upper address lines determine the page in the array. This register is referred to as the page register.

5.6.1 PAGE REGISTER

As indicated in the previous section, the page register is implemented with a latch. The system must have the ability to write the desired page to the page register, and provide the page to the flash array when the sliding window is accessed.

5.6.1.1 Page Register Write

A page register write is triggered by a write to the page register I/O port. The Intel386 EX CPU chip select unit decodes the I/O port address and enables CS0# which is sampled by the PLD. After sampling CS0# active with W/R# high, the PLD generates a CLK pulse to the latch. On the rising edge of CLK, the data is latched. For this example, 3E0h was picked for the page register port address. However, any available port address is acceptable. For more information on programming the Intel386 EX CPU chip select unit, refer to Section 5.5. The PLD equations for implementing a page register write is provided in Appendix B.

5.6.1.2 Flash Array Access

The flash array is accessed through the implemented sliding window. The Intel386 EX CPU decodes the memory address and enables CS1# when the sliding window is accessed. CS1# is connected directly to the

PLD and the latch's OC# signal. When CS1# drives OC#, the latch drives the upper address lines to the flash device. The Intel386 EX CPU drives the lower address lines, and the PLD drives CE#.

5.6.2 PAGE SIZES

The size of each page is determined by the size of the array and the number of bits used for the page register. If all 8-bits are used in the page register, there will be 256 unique pages as illustrated in Figure 6. A 4-Mbyte array divided into 256 pages yields 16 Kbytes per page. See Table 3 for a listing of several page combinations. Most FTL solutions allow page sizes of 4 Kbytes, 8 Kbytes, 16 Kbytes, 32 Kbytes, and 64 Kbytes. For this design example a 16-Kbyte window from C8000h to CC000h was picked for the sliding window. However, any window residing in adapter space (C0000h to F0000h) is acceptable.

Table 3. Page Data Matrix

	1-MB Array	2-MB Array	4-MB Array
5-Bit Page Register	32 32-Kbyte Pages	32 64-Kbyte Pages	32 128-Kbyte Pages
6-Bit Page Register	64 16-Kbyte Pages	64 32-Kbyte Pages	64 64-Kbyte Pages
7-Bit Page Register	128 8-Kbyte Pages	128 16-Kbyte Pages	128 32-Kbyte Pages
8-Bit Page Register	256 4-Kbyte Pages	256 8-Kbyte Pages	256 16-Kbyte Pages ⁽¹⁾

NOTE:

This is the combination implemented in Figure 5.



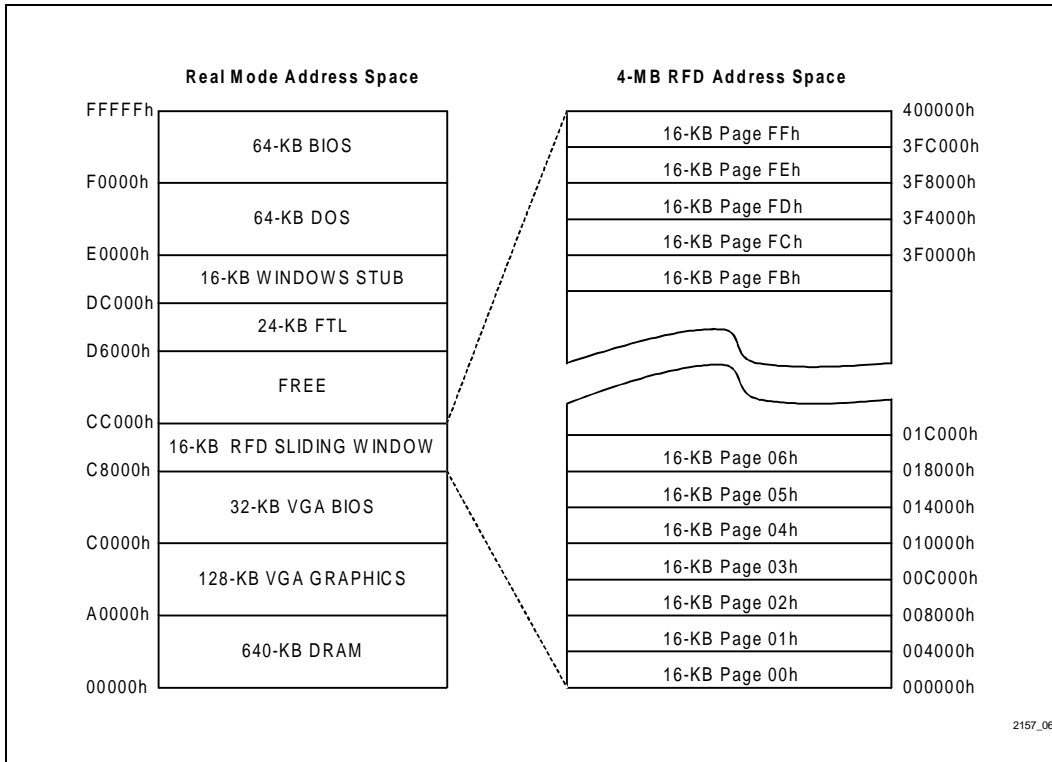


Figure 6. System Memory Map



5.7 Programming the Intel386 EX CPU Chip-Select Unit

This design example uses two of the Intel386 EX CPU chip-selects, CS0# and CS1#. However, any two available chip-selects can be used. If two chip selects are not available, external logic can be used for address decode and READY# generation.

ApBUILDER is a utility that can be used to generate the code to program the Chip-Select Unit. From the ApBUILDER main menu, select “CSU” and “Peripheral.” Fill in the ApBUILDER table with the information given in Table 4 below. After the table is filled in with the correct information, click on

“Showcode” to view the code. Sample code for programming the Intel386 EX CPU Chip-Select Unit is provided in Appendix E. The order number for ApBUILDER is provided in Section 6.1.

5.8 Summary

A Resident Flash Disk is the alternative to traditional magnetic media in embedded applications. It is faster, more rugged, more reliable, smaller, and uses less power than hard disk drives. This complete hardware and software solution is implemented easily and available today.

Table 4. ApBUILDER Chip-Select Unit Data

Chip Select	Chip Select Enable	Bus Ready Enable	16-Bit Width Select	Memory Space Select	SMM (2) Access During	Boundary Address	Region Size	Wait-States
CS0	X	O	X	O	Both	Begin 3E0h	2 Byte	1
CS1	X	O	X	X	Both	Begin C8000h	16 Kbyte	2

NOTES:

1. “X” indicates that this field is selected. “O” indicates that this field is not selected.
2. Selecting “Both” in this field will generate the chip-select regardless of whether or not the CPU is in SMM. Selecting “Memory” in this field is also valid and will generate the chip-select only when the processor is not in SMM.

6.0 ADDITIONAL INFORMATION

6.1 References

Order Number	Title
272420	Intel386™ EX Embedded Microprocessor Datasheet
290429	28F008SA 8-Mbit FlashFile™ Memory Datasheet
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”
292160	AP-609, “Interfacing the Intel386™ EX Embedded Processor to Intel Flash”
292173	AP-618, “Software Concerns of Implementing a Resident Flash Disk”
272485	Intel386™ EX Embedded Microprocessor Hardware Reference
297372	16-Mbit Flash Product Family User’s Manual
297508	FLASHBuilder Utility for 28F016SA, 28F016SV, 28F016XS, 28F016XD
272216	ApBUILDER Interactive Programming Package

6.1 Revision History

Number	Description
-001	Original Version

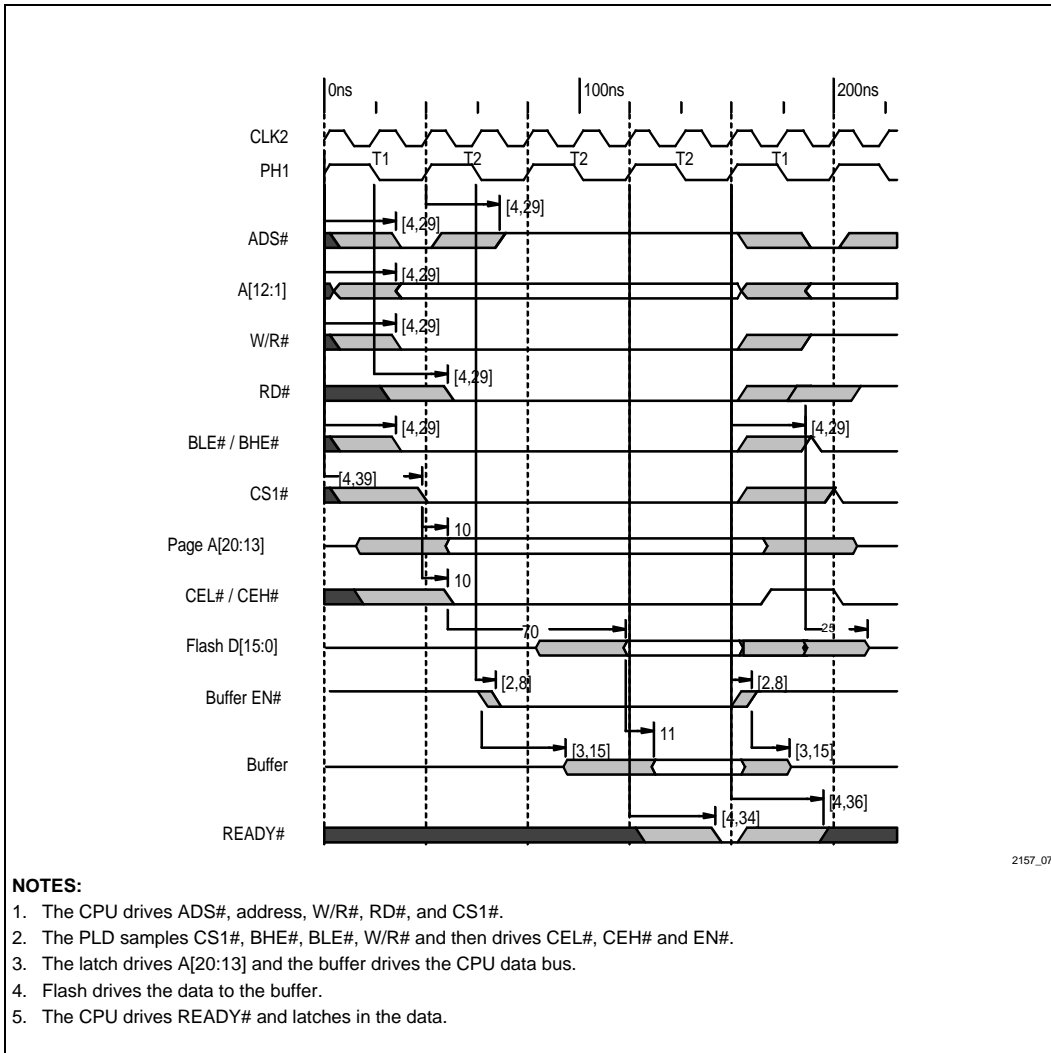


APPENDIX A TIMING DIAGRAMS

For all the following timing diagrams:

- The timing annotation is [minimum delay, maximum delay] or just maximum delay. This datasheet is valid for devices marked with a “B” at the end of the top side tracking number.
- Intel386 EX CPU timing data was taken from the fourth revision datasheet (order# 272240-004).
- 28F016SV timing data was taken from the third revision datasheet (order# 290528-003).
- Buffer timing data was taken from a Texas Instruments 74ACT16245 datasheet (revised April 1993)
- Latch propagation delay data assumed a generic 10 ns t_{PD} .
- PLD data assumed a generic 10 ns t_{PD} and 2 ns to 8 ns t_{CO} .

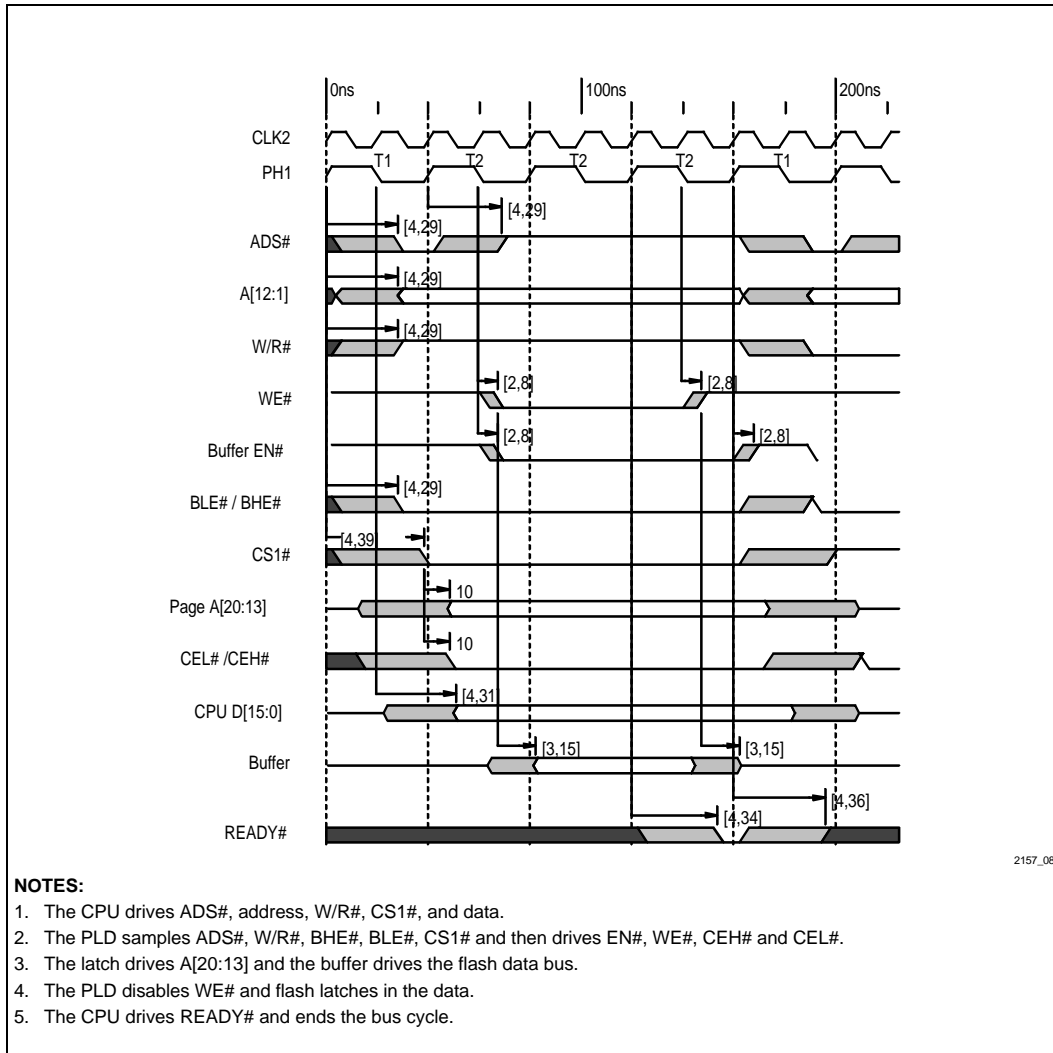




2157_07

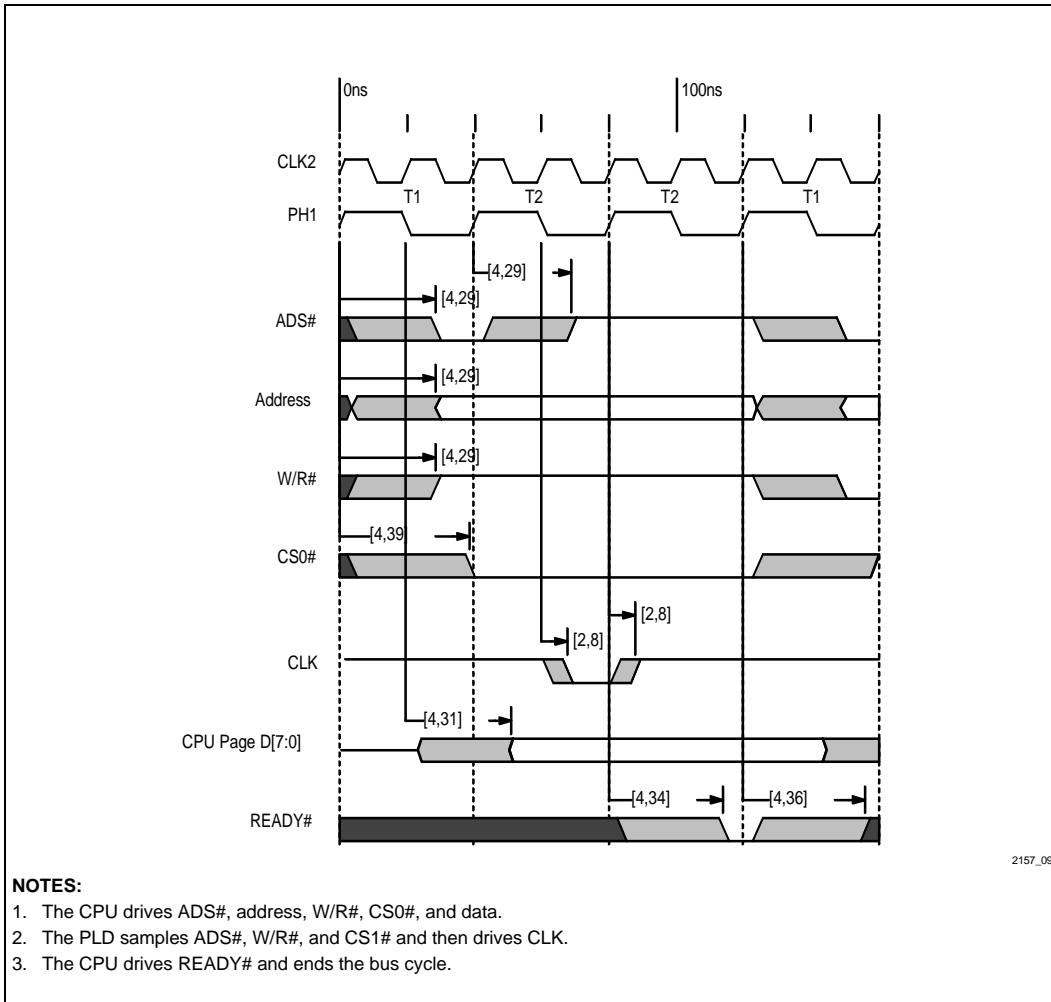
RFD Read Cycle Timing Diagram





2157_08

RFD Write Cycle Timing Diagram



2157_09

Page Register Write Cycle Timing Diagram



APPENDIX B PLD EQUATIONS

TITLE	Intel386(TM) EX CPU RFD INTERFACE
PATTERN	PDS
REVISION	2.0
AUTHOR	TONY SHABERMAN
COMPANY	INTEL
DATE	4/25/95

; 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 implementing a RFD with
; the Intel386 (TM) EX CPU and the 28F016SA. It controls WE# to the 28016SA,
; EN# to the buffer, and CLK to the latch.

; NOTES

; T2_1 will go active for the duration of the first T2 on all cycles.
;
; T2_2 will go active for the duration of the second T2 on all cycles.
;
; Chip-selects have a 39 ns valid delay. This does not allow enough setup time to sample
; CS0# and CS1# in T1. Because of this, CS0# and CS1# are sampled in the middle of the
; first T2. Since EN is dependent on CS1# and CLK is dependent on a CS0#, these signals
; will not be generated until the middle of the first T2.
;
; EN will go active in the middle of the first T2 if CS1# is active. EN is
; disabled by the CPU driving READY#. Since the CPU could drive READY# early in
; the last T2, (+ PH1) is added to the equation to ensure that the buffer stays active during
; the second half of the last T2.
;
; CLK will go active in the middle of the first T2 if CS0# is active. CLK stays active for
; only 1 CLK2 allowing plenty of data setup time before the rising edge.
;
; RESET is used to synchronize the PLD's PH1 to the processor's PH1. In this design example
; it is assumed that another device is generating RESET to the CPU. However, the RESET in this
; PLD can be used to RESET the CPU if desired.

```

CHIP MEMORY_INTERFACE    iPLD22V10N

; INPUTS
PIN 2 CLK2                ; 2X INPUT CLOCK (50MHZ)
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 /CS0                  ; FROM CPU CS0# SIGNAL
PIN /CS1                  ; FROM CPU CS1# SIGNAL
PIN /READY                ; FROM CPU READY# SIGNAL
PIN /BHE                  ; FROM CPU BHE# SIGNAL
PIN /BLE                  ; FROM CPU BLE# SIGNAL

; OUTPUTS
PIN /WE                   ; TO FLASH WE# SIGNAL
PIN /EN                   ; TO BUFFER OUTPUT ENABLE SIGNAL
PIN /CLK                  ; TO LATCH CLK SIGNAL
PIN /CEH                  ; TO FLASH CE# HIGH BYTE
PIN /CEL                  ; TO FLASH CE# LOW BYTE

; 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
PIN RESET                 ; TO CPU RESET

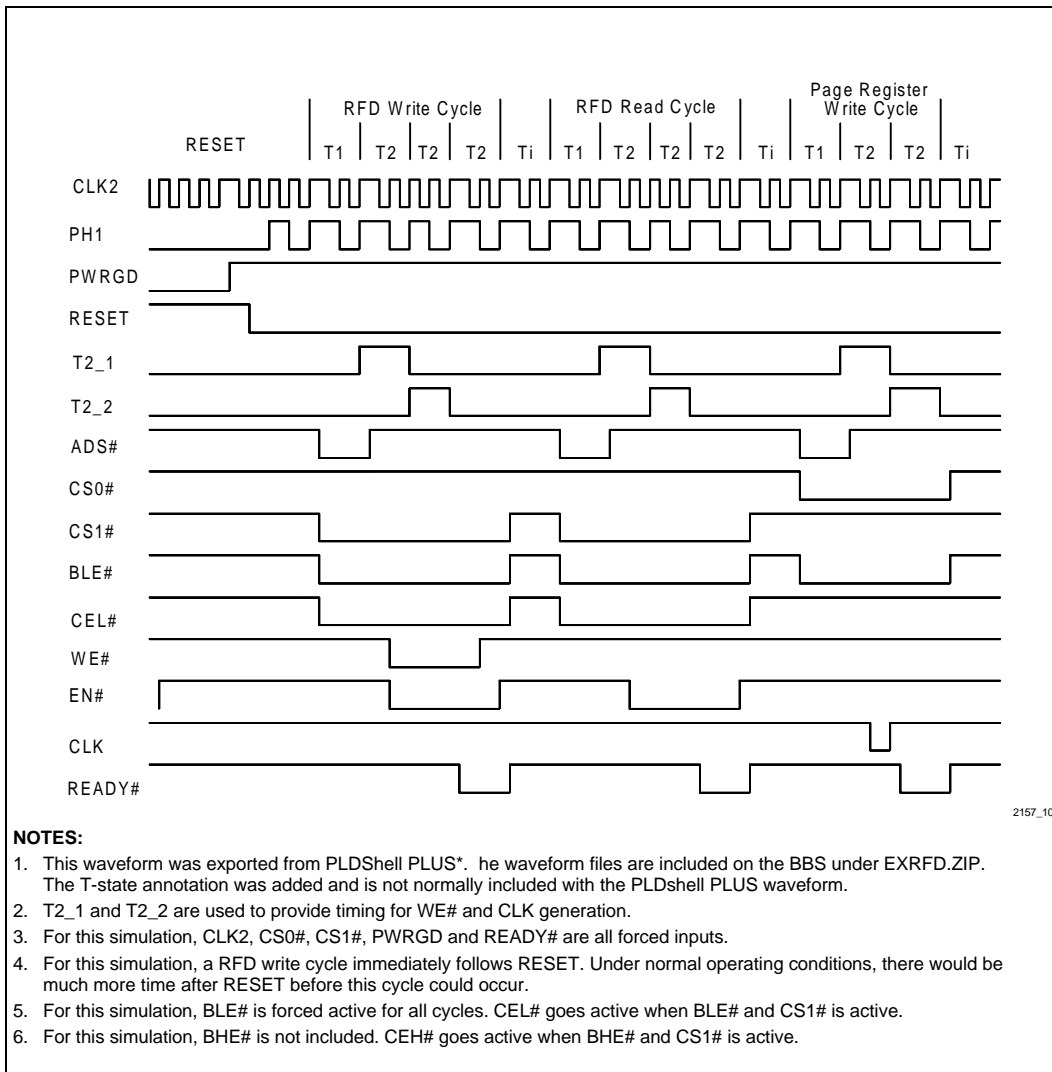
;EQUATIONS
RESET:=/PWRGD            ; RESET TRIGGERED BY PWRGD
PH1:=(/PH1*/RESET)      ; PH1 SYNCs BY RESET
T2_1:=(/PH1*ADS)+(PH1*T2_1) ; T2_1 ACTIVE FOR 2 CLK2 CYCLES AFTER ADS#
T2_2:=(/PH1*T2_1)+(PH1*T2_2) ; T2_2 ACTIVE FOR 2 CLK2 CYCLES AFTER T2_1
WE:=CS1*W_R*(T2_1+T2_2)   ; WE ACTIVE FROM MIDDLE OF T2_1 TIL MIDDLE OF T2_3
EN:=CS1*(T2_1+(EN*/READY+PH1)) ; EN ACTIVE FROM MIDDLE OF T2_1 TIL READY#
CLK:=CS0*W_R*T2_1*PH1    ; CLK ACTIVE DURING THE SECOND HALF OF T2_1
CEH=CS1*BHE
CEL=CS1*BLE

```

```
SIMULATION
TRACE_ON CLK2 PH1 PWRGD RESET T2_1 T2_2 ADS CS0 CS1 BLE CEL WE EN CLK READY
SETF CLK2 /PWRGD /ADS /CS0 /CS1 /READY /BLE
PRLDF /PH1 /T2_1 /T2_2CLOCKF CLK2

CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
SETF PWRGD
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
SETF ADS W_R CS1 BLE
CLOCKF CLK2
CLOCKF CLK2
SETF /ADS
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
SETF READY
CLOCKF CLK2
CLOCKF CLK2
SETF /CS1 /READY /BLE
CLOCKF CLK2
CLOCKF CLK2
SETF CS1 ADS /W_R BLE
CLOCKF CLK2
CLOCKF CLK2
SETF /ADS
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
CLOCKF CLK2
SETF READY
CLOCKF CLK2
CLOCKF CLK2
SETF /READY /CS1 /BLE
CLOCKF CLK2
CLOCKF CLK2
SETF CS0 W_R ADS BLE
CLOCKF CLK2
CLOCKF CLK2
SETF /ADS
CLOCKF CLK2
CLOCKF CLK2
SETF READY
CLOCKF CLK2
CLOCKF CLK2
SETF /READY /CS0 /BLE
CLOCKF CLK2
CLOCKF CLK2
TRACE_OFF
```

APPENDIX C PLDShell WAVEFORM



PLDShell PLUS+ Waveform



APPENDIX D FTL DEVELOPERS

Datalight, Inc.: CardTrick*
307 N. Olympic Ave.
Suite #200
Arlington, WA 98223
(360) 435-8086

M-SYSTEMS: TrueFFS*
4655 Old Ironsides Dr.
Suite #200
Santa Clara, CA 95054
(408) 654-5820

SCM Microsystem, Inc.: S-FTL*
131 Albright Way
Los Gatos, CA 95030
(408) 370-4888

System Soft: SS-FTL
313 Speen St.
Natick, MA 01760
(508) 651-0088

For an updated list of FTL developers call the Intel FaxBack* system and request document #2255.



APPENDIX E CHIP-SELECT UNIT CODE

```

;Initialize Chip Select Unit for:

; CS0: Start address is 03E0H.
;   Region size is 2 bytes.
;   1 wait states.
;   Chip select 0 is Enabled.
;   16 bit data bus size in I/O space.
;   External bus ready is Disabled.
;   SMM region is accessible during SMI access and memory access.

; CS1: Start address is 0C8000H.
;   Region size is 16 Kbytes.
;   2 wait states.
;   Chip select 1 is Enabled.
;   16 bit data bus size in memory space.
;   External bus ready is Disabled.
;   SMM region is accessible during SMI access and memory access.

INCLUDE 80386EX.INC
_TEXT SEGMENT PUBLIC 'CODE'
ASSUME CS:_TEXT

Init_CSU Proc Far

; Enable expanded I/O space for peripheral initialization.
MOV  AX, 08000H    ;Enable expanded I/O space
OUT  REMAPCFGH, AL ; and unlock the re-map bits
XCHG AL, AH
OUT  REMAPCFGL, AL
OUT  REMAPCFG, AX

_SetEXRegWord CS0ADL, 08601H    ;Configure chip select 0
_SetEXRegWord CS0ADH, 0FH
_SetEXRegWord CS0MSKL, 0401H
_SetEXRegWord CS0MSKH, 00H

_SetEXRegWord CS1ADL, 08702H    ;Configure chip select 1
_SetEXRegWord CS1ADH, 0CH
_SetEXRegWord CS1MSKL, 03C01H
_SetEXRegWord CS1MSKH, 00H

; Restore I/O space to original condition.
_SetEXRegByte REMAPCFGH, 00H ;Disables expanded I/O space
RET
Init_CSU ENDP
_TEXT ENDS
END
}

```


Filename: 292157_1.DOC
Directory: C:\TESTDOCS\DOCS
Template: C:\WINDOWS\WINWORD6\TEMPLATE\ZAN____1.DOT
Title: E
Subject:
Author: Kevin Culcasi
Keywords:
Comments:
Creation Date: 08/14/95 11:52 PM
Revision Number: 44
Last Saved On: 11/28/95 9:31 AM
Last Saved By: Ward McQueen
Total Editing Time: 382 Minutes
Last Printed On: 11/28/95 9:31 AM
As of Last Complete Printing
Number of Pages: 24
Number of Words: 5,157 (approx.)
Number of Characters: 29,399 (approx.)