AP-614

APPLICATION NOTE

# Adapting DRAM-Based Designs for the 28F016XD

**SUJAN KAMRAN**
TECHNICAL MARKETING
ENGINEER

November 1995

Order Number: 292168-001

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



CG-041493

## 1.0 INTRODUCTION

The Intel 28F016XD DRAM-interface flash memory obsoletes the redundant two-memory paradigm of nonvolatile memory (NVM) shadowed to DRAM in many embedded designs. Traditionally, system architectures stored code in nonvolatile and relatively low performance memory sources (HDD, ROM, Bulk-Flash) and then downloaded it to a faster volatile source for execution. Using the 28F016XD for both the nonvolatile code storage memory AND the code execution memory eliminates this need for redundancy as shown in Figure 1.

Advantages derived from eliminating the traditional dual-memory scenario include:

- System Cost Savings
- Equivalent or Higher Performance
- Faster System Boot
- Reduced Board Space
- Lower Overall Power Requirements
- Simplified Design
- Increased Reliability

The 28F016XD device is priced competitively with respect to the combination of 16-Mbit x16 DRAMs and ROM/HDD, while offering comparable performance to DRAM. Systems in which the HDD can be replaced benefit further with the removal of the Drive Controller. The boot ROM/Flash code can also be incorporated into the 28F016XD. These are all examples of how the 28F016XD makes it possible to integrate many memories into one, reducing cost and board space while simplifying the overall design.

The 28F016XD leverages the existing DRAM controller in system designs, thereby minimizing the glue logic required to interface to flash memory. It is a 16-Mbit device, organized as 1-Mbyte x 16, with ten row and ten column addresses multiplexed on $A_0$–$A_9$. The 28F016XD's presence on the main-memory bus assures cacheability, maximizing the effective system read performance. Flash memory also presents significant power/energy and reliability advantages compared to DRAM, since it does not require refresh and is not susceptible to alpha-particle soft errors.

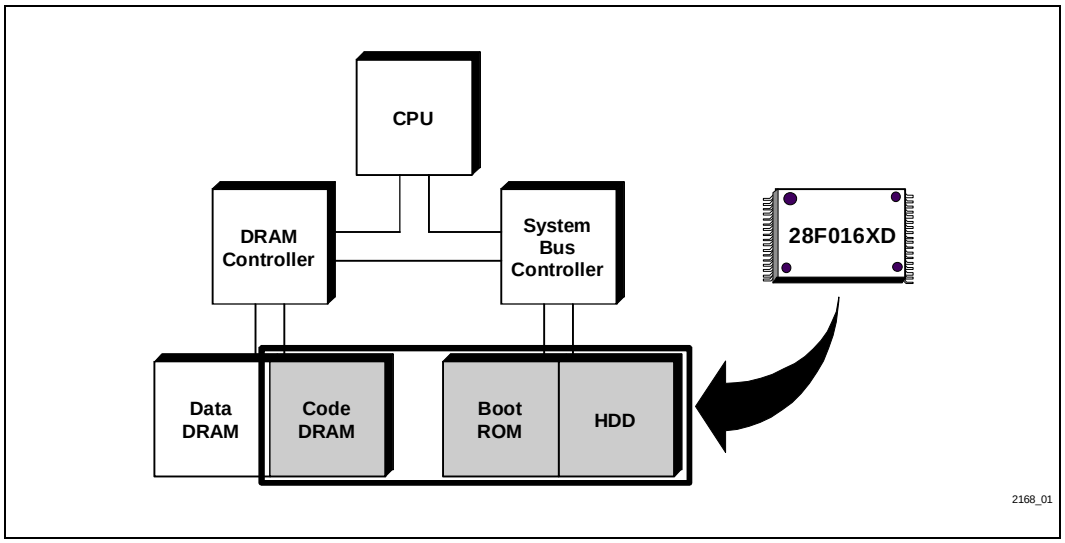



**Figure 1. The 28F016XD Eliminates the Need to Shadow Code from a Slow, Nonvolatile Memory Source to a Faster, Volatile Execution Memory**

Modifications to DRAM-based systems to become compatible with Intel's 28F016XD 16-Mbit DRAM-interface flash memory are straightforward. This application note discusses the design considerations that one must consider when converting a traditional code-DRAM-based design to its 28F016XD-based alternative including:

- Hardware
  - — DRAM Controller Timing Compatibility
  - — Memory Address Multiplexing
  - — Parity
- Software
  - — Direct Execute O/S and Applications
  - — BIOS Memory Scan

This document highlights two examples of reference designs that Intel has completed with the 28F016XD. The first details the simple Intel 386™ EX embedded microprocessor evaluation board (MB1) modifications to make the system 28F016XD-compatible. The second example is an Intel486™ CPU-based motherboard (with the Intel 82420ZX PCIset) which was adapted to utilize the 28F016XD. The focus remains on the process and analysis, using these reference designs as examples, allowing this document to serve as a guide for determining 28F016XD compatibility with many systems.

## 2.0 HARDWARE CONSIDERATIONS/CHANGES

### 2.1 Wait-State Profile/Memory Controller Timings

The key to determining the compatibility of a system with the 28F016XD flash memory is the flexibility of its DRAM controller. The system memory controller, whether it be integrated in an embedded micro-controller/chipset or implemented in a custom ASIC/FPGA, bridges the processor and main memory sub-system. This logic generates the necessary control signals (RAS# and CAS#) for DRAM read, write, and refresh cycles.

System compatibility with the 28F016XD hinges on the ability of the memory controller to compensate for the timing differences between "standard" DRAMs and the 28F016XD. Depending on the system configuration, one or a combination of the following may need to be done to the controller timings in order for the system to become 28F016XD-compatible:

- nothing
- reduce overall system operating frequency
- increase CAS# access time (wait-states) for 28F016XD-based memory banks

The most significant timing differences between the 28F016XD and DRAMs (which can be compensated for with additional wait-states) are shown in Table 1. See the 28F016XD datasheet referenced in the Additional Information section for complete timing information. Generally, memory controllers which allow for the CAS# pulse-width ($t_{CAS}$) to be programmed will accommodate the other access time-related inconsistencies. Memory controllers which support per-bank wait-state control enable maximum system performance by optimizing access times to both DRAM and 28F016XD memory banks.

Unlike the specifications shown in Table 1, some timing incompatibilities (with respect to DRAM) cannot be resolved by simply configuring DRAM controller wait-states. Determining DRAM-controller/system compatibility with the specifications shown in Table 2 is crucial for ensuring the ability to interface to the 28F016XD.

Most DRAM controller designs up to 33 MHz will be compatible with the 28F016XD's hold and precharge times. However, one should examine carefully the impact of the "Output Buffer Turn-Off Delay" ($t_{OFF}$) timing difference with respect to each specific design.

The DRAM controller, processor or other system memory (e.g., DRAM) must not drive the data bus (after reading from the 28F016XD) until $t_{OFF}$ has elapsed.

High-speed transceivers placed in the 28F016XD-data path can eliminate such contention issues, as illustrated in Figure 2. Note that some systems which incorporate buffers on the data path to minimize CPU local bus loading may not require additional transceivers to avoid bus contention.

See AP-384, "Designing with the 28F016XD" for complete details in understanding the timing differences between the 28F016XD and DRAMs, and to determine memory controller compatibility.

**Table 1. Key 28F016XD Timing Parameters Compared to 60ns–70 ns 16-Mbit DRAMs for Determining Memory Controller Compatibility ($V_{CC}$ = 5V)**

| Symbol | Description | 28F016XD | DRAM |
|---|---|---|---|
| $t_{AA}$ | Access Time from Column Address (max) | 65 ns | 30–35 ns |
| $t_{CAS}$ | CAS# Pulse Width (Reads/Writes) (max) | 35/50 ns | 15–20 ns |
| $t_{CAC}$ | Access Time from CAS# (max) | 35 ns | 15–20 ns |

**Table 2. Additional 28F016XD Timing Parameters Compared to 60–70 ns 16-Mbit DRAMs for Determining Memory Controller Compatibility ($V_{CC}$ = 5V)**

| Symbol | Description | 28F016XD | DRAM |
|---|---|---|---|
| $t_{CAH}$ | Column Address Hold Time (min) | 20 ns | 10–15 ns |
| $t_{RAH}$ | Row Address Hold Time (min) | 15 ns | 10 ns |
| $t_{CP}$ | CAS# Precharge Time (min) | 15 ns | 10 ns |
| $t_{CRP}$ | CAS# to RAS# Precharge Time (min) | 10 ns | 5 ns |
| $t_{OFF}$ | Output Buffer Turn-Off Delay (max) | 30 ns | 15 ns |

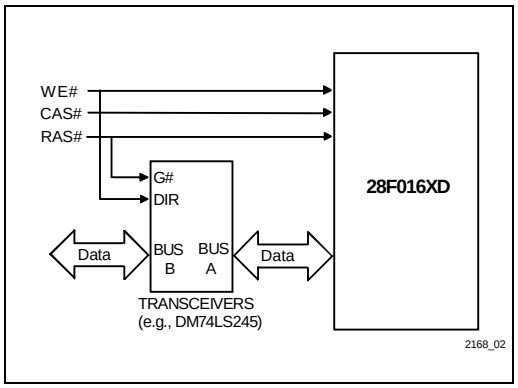



**Figure 2. 28F016XD-Data Path Transceivers Eliminate Potential Bus Contention Issues**

## 2.2 Memory Address Translation

Understanding the memory translation of the system memory controller is another vital element in converting a design to the 28F016XD. Most DRAM controllers do not obey a 100% sequential ordering of memory addressing between the CPU and memory components, as shown in Figure 3. In DRAM-based systems, the order of the individual bits doesn't matter as long as the address is unique. This flexibility only holds true because the memory is written to and read from in the same system; the DRAM of any design satisfies this requirement. The 28F016XD memory, on the other hand, may be mass-programmed by a system with a different address translation scheme than that of the actual platform in which it will eventually reside.

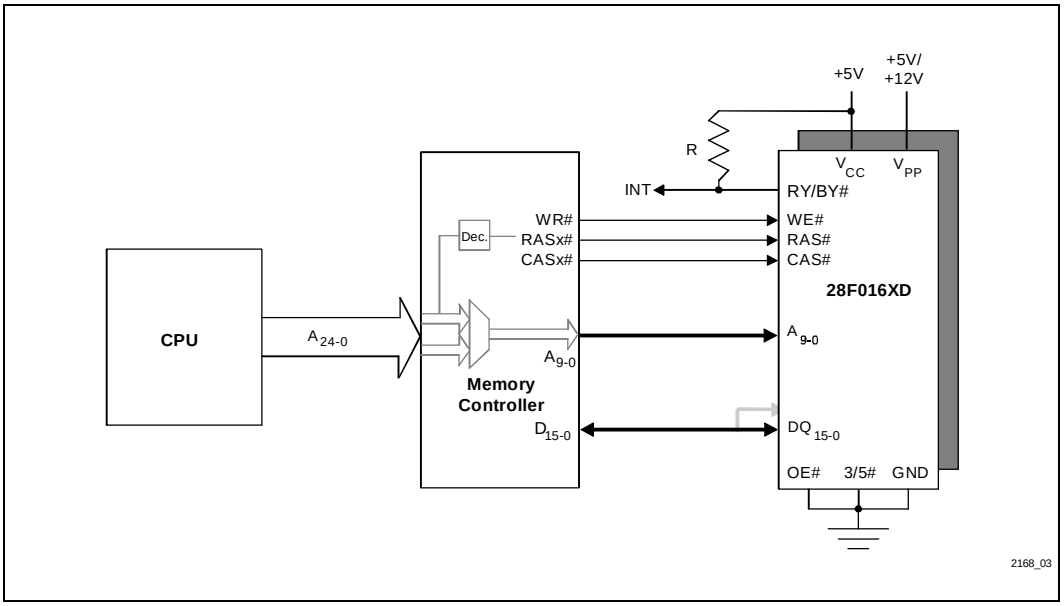



**Figure 3. The DRAM Controller/Chipset Multiplexes and Perhaps Scrambles the Host Address Bus between the CPU and the Actual Memory Components**

The chipset/memory controller determines the address bus multiplexing between the host CPU and main memory. For example, Table 3 shows the memory address translation of the Intel 82420ZX PCIset. Table 4 shows the memory address translation of the Needham's EMP-20 programmer.

Cases where the programming (writing) system is different from the reading (host) system must synchronize the memory address translations to match one another. A possible solution to synchronize the memory address translations would be to modify the host/programmer's translation scheme to match the others. This may be as simple as changing PLD equations. More than likely, however, this would require significant hardware changes to the board or the re-design of the memory controller/ASIC—an extremely expensive proposition, both in terms of dollars and time.

**Table 3. Host Address Translation of the Intel 82420ZX PCIset**

| MA[10:0][1] | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Row Address | $A_{24}$ | $A_{22}$ | $A_{20}$ | $A_{19}$ | $A_{18}$ | $A_{17}$ | $A_{16}$ | $A_{15}$ | $A_{14}$ | $A_{13}$ | $A_{12}$ |
| Column Address | $A_{23}$ | $A_{21}$ | $A_{11}$ | $A_{10}$ | $A_{09}$ | $A_{08}$ | $A_{07}$ | $A_{06}$ | $A_{05}$ | $A_{04}$ | $A_{03}$ |

**NOTE:**

1. MA[10:0] are the eleven main-memory address lines. These correspond to (28F016XD A[9:0], EMA0/OMA0).

**Table 4. Host Address Translation of the Needham's EMP-20 Programmer**

| MA[9:0] | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| Row Address | $A_{21}$ | $A_{20}$ | $A_{19}$ | $A_{18}$ | $A_{17}$ | $A_{16}$ | $A_{15}$ | $A_{14}$ | $A_{13}$ | $A_{12}$ |
| Column Address | $A_{11}$ | $A_{10}$ | $A_{09}$ | $A_{08}$ | $A_{07}$ | $A_{06}$ | $A_{05}$ | $A_{04}$ | $A_{03}$ | $A_{02}$ |

An alternative to making the actual translations match in hardware would be to make the translations match by software—"scrambling" the image file read into the programmer to match the host system's translation scheme. Appendix A lists a "C" program that shuffles an image file (Intel Hex format) to match the Needham's EMP-20 and Intel386 EX embedded microprocessor evaluation board memory address translations. This example code is also available on the Intel Applications Support BBS and can easily be modified to account for the memory address translations and scrambling of many systems.

Programming and reading the 28F016XD in the same system avoids the need to compensate for differing memory address translations between a SIMM programmer and the final system. Using the host CPU and system to program the flash SIMM eliminates the need for a software "reshuffling" routine and provides a significantly more elegant solution. Refer to AP-610, "In-System Flash Code and Data Update Techniques" for information on writing to flash memory in the host system.

## 2.3 Parity

The 28F016XD flash memory component is intrinsically nonvolatile and therefore not susceptible to alpha-particle soft errors, as are DRAMs. Therefore, in general, 28F016XD-based memory banks do not require parity. Most DRAM controllers today allow for enabling/disabling parity support via a configuration register. The use of parity is continually decreasing as DRAM manufacturing processes mature, thereby drastically reducing the probability of soft errors.

In systems requiring parity for DRAM banks, simple logic can generate parity bits if the memory controller does not support per-bank parity control. Figure 4 illustrates this parity-generation logic, which can be incorporated directly on the SIMM or system board. For a list of SIMM vendors that provide 28F016XD-based SIMMs, see Appendix B or consult Intel's FaxBack* system for SIMM product briefs.

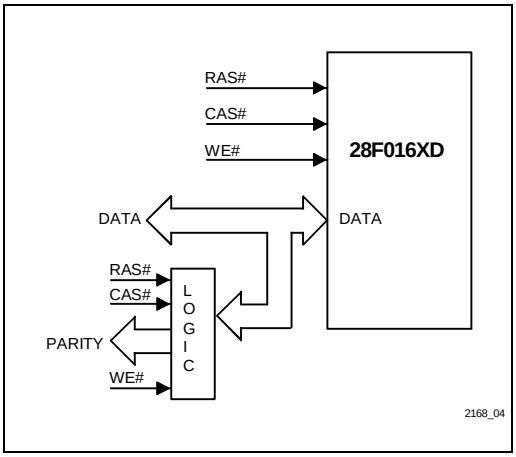



**Figure 4. Simple Logic Can Implement Parity Generation for 28F016XD-Based Memory Banks**

## 2.4 Intel386 EX Embedded Microprocessor Evaluation Board Hardware Considerations

### Timing Issues

Timing/wait-state profile changes were not required to make the Intel386 EX embedded microprocessor evaluation board 28F016XD-compatible. The system operating frequency of 25 MHz, combined with the simple DRAM control implemented in a 22V10 PLD, was already compatible with the 28F016XD timings. The DRAM control for this platform was designed to support two wait-state, non-page-mode accesses; these timings match those of the 28F016XD.

### Other MB1 Hardware Changes

The following list describes the minor hardware changes to the standard Intel386 EX embedded microprocessor evaluation board that were required in order to replace the DRAM SIMM with a 28F016XD SIMM:

- Utilized (optional) 32-K SRAM as a source of "scratch-pad" memory for program variables
- Modified jumper settings to account for 4-MB 28F016XD-based SIMM

- Reconfigured chip-selects to support the 4-MB 28F016XD-based SIMM and re-mapping of the optional SMM SRAM

Complete implementation details of modifying the Intel386 EX embedded microprocessor evaluation board to operate with a 28F016XD-based SIMM are available as a reference design on the Intel Applications Support BBS (file: E3X_XDFL.EXE).

## 2.5 Intel486 Embedded Microprocessor Hardware Considerations

### Timing Issues

The primary hardware concern in adapting the 82420ZX PCIset-based Intel486 embedded microprocessor system for 28F016XD compatibility centered around configuring the wait-state profile of the DRAM control logic. Accommodating the 28F016XD timings required some modification of the DRAM-control default settings. An ISA stub at address 0C8000H contained the code to modify the appropriate configuration registers of the chipset.

Appendix C lists the code used to write to the 82420ZX PCIset's configuration registers. This example code can be adapted for a wide array of systems/controllers. In short, this software routine writes certain values to different I/O addresses (control registers of the chipset) to configure the DRAM timing profiles and memory map of the system. Section 3.3 discusses incorporating this reconfiguration code into the system BIOS.

The 82420ZX PCIset supports two-way interleaved DRAM memory banks for maximum memory sub-system performance. In general, interleaving is a configurable option to boost system performance. The 82420ZX PCIset, however, does not allow for non-interleaved main memory banks. This interleaving resulted in a timing incompatibility between this Intel486 embedded microprocessor system and the 28F016XD at 33 MHz. Specifically, the $t_{OFF}$ specification for the 28F016XD of 30 ns does not allow sufficient time for one-bank of flash to tri-state before an access to the other bank (in the same interleaved pair) would begin to drive the data bus. This contention issue was resolved by slowing the overall system operating frequency to 25 MHz, thereby allowing an additional 10 ns to compensate for the extended $t_{OFF}$ of the 28F016XD. Disabling interleaving, although not an option with this chipset, may have allowed 33 MHz performance (perhaps even without the addition of wait-states).

### Memory Address Translation

As discussed in Section 2.2, the memory address translation differences between the SIMM programmer and the host system needed to be well understood. Instead of developing a software utility to "shuffle" the binary code file to compensate for the translation difference between the host system and programmer, our software team developed routines that enable programming the 28F016XD SIMMs directly in the Intel486 embedded microprocessor system. Using the host CPU and system to program the flash SIMMs eliminated the need for a software "reshuffling" routine and provided a significantly more elegant solution. Example code for writing to the flash SIMMs is available on the Intel Applications Support BBS (filename: 614_CODE.EXE).

# 3.0 SOFTWARE CONSIDERATIONS

One important aspect to keep in mind when developing code for direct execution out of a nonvolatile memory concerns code and data segments—they must be separated! This separation prevents program data from accidentally being written to nonvolatile memory. Flash memory is in-system updateable, but not fully bit alterable. Therefore, specific address assignments are necessary to fit code and data into a given target system's memory map.

## 3.1 Compiled Code

The compiler is an important tool in developing optimized code for flash memory. It's the role of the compiler to convert source code into the actual machine language for the target microprocessor. This tool, however, only generates machine code. It does not resolve memory allocation addresses for code and data segments. Memory allocation is left up to the linker to complete.

Fortunately, many linkers provide a mechanism to explicitly place code and data in a specific location for embedded applications. Therefore, the source code structure does not have to define specific locations for data structures when generating code for direct execution out of a nonvolatile memory. This linker mechanism furnishes a simple process for porting code from one system to another without requiring any code modifications.

Numerous compilers/linkers available in the marketplace today specify code and data segments locations via a command line switch. Through the command line, the linker receives specific segment addresses. The linker then uses this information to place the code and data segments into the target system. AB-62, "Compiled Code Optimizations for Flash Memories," describes this process in detail for developing new applications as well as recompiling legacy code for direct execution out of nonvolatile memory.

## 3.2 Direct Execute Operating Systems

Many operating systems available today for the embedded market are so-called "ROM-able. For example, the Intel386 EX embedded microprocessor evaluation board showcased Datalight's ROM-DOS*, while the Intel486 embedded microprocessor system operated under the Microsoft MS-DOS ROM-Executable* environment. In both cases, the operating system was executed directly from the 28F016XD-based SIMM. Reference Appendix D for a listing of embedded operating systems providers.

Creating the direct-execute image files of Microsoft MS-DOS ROM-Executable generates two files. The first is the "low" file which needs to be placed at any 16-Kbyte address boundary between 0C0000H–0DFFFFH. This address region is scanned at boot-up for "stub" files (BIOS extensions) indicating the presence of the O/S. This "low" file also contains a user-specified pointer to the area in "main-memory" where the remaining code exists. The second file created (the "high" file) is this remaining code. For further information about implementing a direct-execute DOS-based system, refer to AP-362, "Implementing Mobile PC Designs Using High-Density FlashFile™ Components."

Figures 5 and 6 show the memory map of Intel386 EX embedded microprocessor evaluation board and Intel486 microprocessor system, respectively.

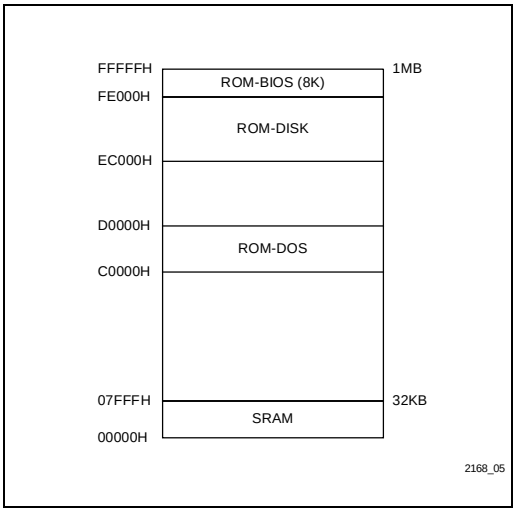



**Figure 5. Intel386 EX Embedded Microprocessor Evaluation Board System Memory Map with 28F016XD SIMM**

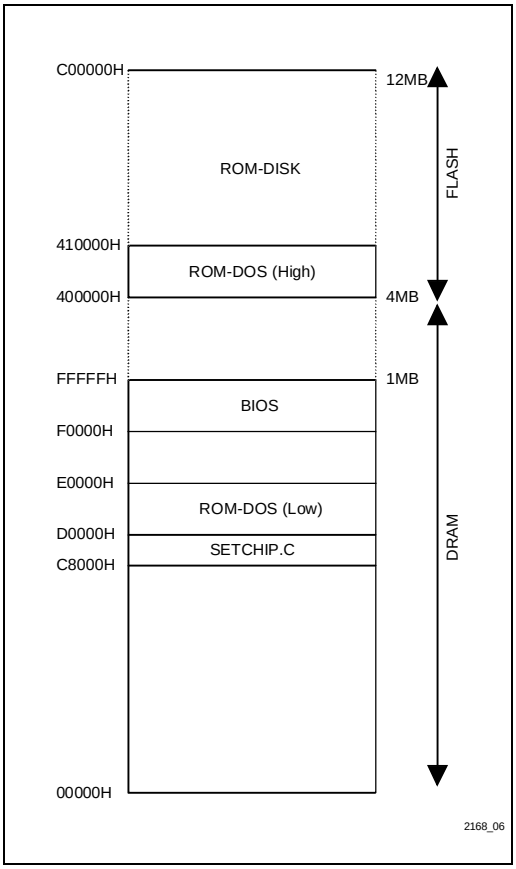



**Figure 6. Intel486 Microprocessor System Memory Map with 28F016XD SIMM**

## 3.3 Memory Self-Test/BIOS Memory Scan

Systems which use the 28F016XD must consider the BIOS memory scan of the system upon boot-up. Generally, the BIOS will write and read back a data pattern such as 55AAH to memory locations to determine the amount of DRAM in the system. This method will not be able to detect the amount of flash memory in the system, since writing to flash involves a two-cycle command sequence.

Reserving a region of system memory for the 28F016XD and restricting the memory scan from this area will avoid any possible corruption of the flash memory. The amount of DRAM-interface flash memory can be user-definable (in the BIOS set-up utility) or jumper selectable for appropriate configuration of the chipset's memory map registers. The existing BIOS can incorporate this task of reconfiguring the chipset easily.

## 3.4 Intel386 EX Embedded Microprocessor Software

Loading the Datalight ROM-DOS files into the 28F016XD-based SIMMs presented the greatest software challenge for the Intel386 EX embedded microprocessor evaluation board project. As discussed in Section 2.2, the memory address translation of the EMP-20 programmer and the Intel386 EX embedded microprocessor evaluation board do not match. Our software team developed a routine to "scramble" the image file read into the programmer to match the host system's translation scheme. Appendix A lists this "C" program; it is also available on the Intel Applications Support BBS. One can easily modify this example code to account for the memory address translations and scrambling of many systems.

## 3.5 Intel486 Embedded Microprocessor Software

The software issues relating to the Intel486 embedded microprocessor system centered around the development of the software routine to reconfigure the 82420ZX PCIset. Initially this code was implemented in a ISA stub because we did not have control of the BIOS. This reconfiguration code, shown as an example in Appendix C, accomplished three specific tasks:

- Modified DRAM Control Timing Registers to accommodate 28F016XD specifications
- Modified Main Memory Map (DRAM) to include 28F016XD SIMMs
- Reset flash components to read array mode before returning control of system to processor

Section 2.5 discussed the modifications of the timing registers. Writing 00H to register 55H of the 82420ZX PCIset configured the chipset to support a 4-2-2-2 (clocks) read cycle profile, matching the timing requirements of the 28F016XD.

This extended BIOS code also reconfigured the chipset's memory map registers to account for the flash memory in the system. As discussed earlier, the traditional AA55H memory-scan will not detect 28F016XD-based SIMMs. Therefore, the chipset was configured specifically to recognize the 28F016XD flash memory in the main memory region.

A sequence of two FFH write cycles was issued to each of the flash components, "resetting" them to read array mode. Since the system is not aware of flash in the main memory regions, many different writes may have been issued to the flash memory during power-up. These could be interpreted by the Command User Interface of the flash components and thus change their state. Writing two "Read Array" commands in sequence will clear any previously issued commands and return the components to a known state, e.g., read mode. Ensuring that the BIOS does not unnecessarily write to the 28F016XD components during power-up avoids this issue all together.

## 4.0 CONCLUSION

The 28F016XD flash memory component is an ideal memory solution for numerous reasons. The embedded environment lends itself to code and data segmentation with the wide-spread availability of direct execute operating systems and applications. Adapting existing systems as well as designing new platforms to accommodate the 28F016XD is a simple and straightforward task that will reap cost, power and space savings while improving overall system read performance and reliability.

## 5.0 ADDITIONAL INFORMATION

### 5.1 References

| Order Number | Document/Tool |
|---|---|
| 297372 | "16-Mbit Flash Product Family User's Manual" |
| 272525 | "Intel 386™ EX Embedded Microprocessor Evaluation Board Manual" |
| 290533 | 28F016XD DRAM-Interface Flash Memory Datasheet |
| 290467 | 82420ZX PCIset Datasheet |
| 292092 | AP-357 "Power Supply Solutions for Flash Memory" |
| 292097 | AP-362 "Implementing Mobile PC Designs Using High-Density FlashFile™ Components" |
| 292123 | AP-374 "Flash Memory Write Protection Techniques" |
| 292131 | AP-384 "Designing with the 28F016XD" |
| 292163 | AP-610 "Flash Memory In-System Code and Data Update Techniques" |
| 292152 | AB-58 "28F016XD-Based SIMM Designs" |
| 292165 | AB-62 "Compiled Code Optimizations for Flash Memories" |
| 297508 | FLASHBuilder Utility |
| Contact Intel/Distribution Sales Office | 28F016XD Benchmark Utility |
| Contact Intel/Distribution Sales Office | 28F016XD iBIS Model |
| Contact Intel/Distribution Sales Office | 28F016XD VHDL Model |
| Contact Intel/Distribution Sales Office | 28F016XD Timing Designer Library Files |
| Contact Intel/Distribution Sales Office | 28F016XD Orcad and ViewLogic Schematic Symbols |

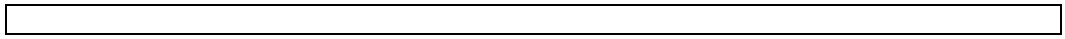
### 5.2 Revision History

| Number | Description |
|---|---|
| -001 | Original Version |

# APPENDIX A

# EXAMPLE CODE FOR MEMORY ADDRESS TRANSLATION MATCHING OF INTEL386 EX EVALUATION BOARD AND NEEDHAM'S EMP-20 PROGRAMMER

```
/*****************************************************************
 *                                  Copyright Intel Corporation 1994
 *    AUTHOR:
 *       Steve Gorman / Garry Mion
 *
 *    FILE: mergeit.c
 *
 *****************************************************************/
#include <stdio.h>
#include <stdlib.h>

#define FILL_LOCHAR 0x31
#define FILL_HICHAR 0x32
unsigned char buffer[0x8000];

void main(int argc, char *argv[])
{
       FILE *InFile, *OutFile;
       FILE *LoFile, *HiFile;
       FILE *TmpFileA, *TmpFileB,*TmpFileC, *TmpFileD, *TmpFile;
       int i, BytesRead;
       long  TBytesRead;

       if(argc < 3)
       {
              printf("Parmeters required --> %s <Input File> <Low Output File>
<Hi Output File>\n",argv[0]);
              exit(0);
       }

  if( (InFile = fopen(argv[1],"rb")) == NULL)
  {
    printf("Unable to open Input file : %s\n", argv[1]);
    exit(3);
  }

  if( (TmpFileA = fopen("tmpa.wrk","w+b")) == NULL)
  {
    printf("Unable to open Temporary output file mergeit.wrk\n");
    fclose(InFile);
    exit(3);
  }
  if( (TmpFileB = fopen("tmpb.wrk","w+b")) == NULL)
  {
    printf("Unable to open Temporary output file mergeit.wrk\n");
    fclose(InFile);
    exit(3);
  }
```



```
  while(!feof(InFile))
  {
        BytesRead = fread(buffer,1,0x800,InFile);
        if(BytesRead < 0x800)
                break;
        fwrite(buffer,1,BytesRead, TmpFileA);
        BytesRead = fread(buffer,1,0x800,InFile);
        if(BytesRead < 0x800)
                break;
        fwrite(buffer,1,BytesRead, TmpFileB);
  }
  fclose(InFile);

  fseek(TmpFileA,0L,SEEK_SET);        // Start back at the beginning of merged
  fseek(TmpFileB,0L,SEEK_SET);        // Start back at the beginning of merged

  if( (LoFile = fopen(argv[2],"wb")) == NULL)
  {
    printf("Unable to open low output file %s\n",argv[2]);
    fclose(TmpFile);
    exit(3);
  }

  for(i=0; i < 8; i+=2)
  {
         buffer[i] = FILL_LOCHAR;
         buffer[i+1] = FILL_HICHAR;
  }
  while(!feof(TmpFileA))
  {
        BytesRead = fread(&buffer[2],1,2,TmpFileA);
        if(BytesRead < 2)
          continue;
        fwrite(buffer,1,4,LoFile);
  }
  fclose(LoFile);
  if( (HiFile = fopen(argv[3],"wb")) == NULL)
  {
    printf("Unable to open hi output file %s\n",argv[3]);
    exit(3);
  }

  while(!feof(TmpFileB))
  {
        BytesRead = fread(&buffer[2],1,2,TmpFileB);
        if(BytesRead < 2)
          continue;
        fwrite(buffer,1,4,HiFile);
  }

}
```

```
# Microsoft Visual C++ generated build script - Do not modify

PROJ = MERGEIT
DEBUG = 1
PROGTYPE = 6
CALLER =
ARGS = flshdemo.bin lo.bin hi.bin
DLLS =
D_RCDEFINES = -d_DEBUG
R_RCDEFINES = -dNDEBUG
ORIGIN = MSVC
ORIGIN_VER = 1.00
PROJPATH = D:\SOURCES\DATA\
USEMFC = 0
CC = cl
CPP = cl
CXX = cl
CCREATEPCHFLAG =
CPPCREATEPCHFLAG =
CUSEPCHFLAG =
CPPUSEPCHFLAG =
FIRSTC = MERGEIT.C
FIRSTCPP =
RC = rc
CFLAGS_D_DEXE = /nologo /W3 /FR /G2 /Zi /D_DEBUG /Od /AM /D_DOS
CFLAGS_R_DEXE = /nologo /W3 /FR /G2 /DNDEBUG /Gs /Ox /AM /D_DOS
LFLAGS_D_DEXE = /NOLOGO /ONERROR:NOEXE /NOI /CO /STACK:5120
LFLAGS_R_DEXE = /NOLOGO /ONERROR:NOEXE /NOI /STACK:5120
LIBS_D_DEXE = oldnames mlibce
LIBS_R_DEXE = oldnames mlibce
RCFLAGS = /nologo
RESFLAGS = /nologo
RUNFLAGS =
OBJS_EXT =
LIBS_EXT =
!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS_D_DEXE)
LFLAGS = $(LFLAGS_D_DEXE)
LIBS = $(LIBS_D_DEXE)
MAPFILE = nul
RCDEFINES = $(D_RCDEFINES)
!else
CFLAGS = $(CFLAGS_R_DEXE)
LFLAGS = $(LFLAGS_R_DEXE)
LIBS = $(LIBS_R_DEXE)
MAPFILE = nul
RCDEFINES = $(R_RCDEFINES)
!endif
!if [if exist MSVC.BND del MSVC.BND]
!endif
SBRS = MERGEIT.SBR


MERGEIT_DEP =

all:    $(PROJ).EXE $(PROJ).BSC

MERGEIT.OBJ:   MERGEIT.C $(MERGEIT_DEP)
       $(CC) $(CFLAGS) $(CCREATEPCHFLAG) /c MERGEIT.C
```

```
$(PROJ).EXE::  MERGEIT.OBJ $(OBJS_EXT) $(DEFFILE)
        echo >NUL @<<$(PROJ).CRF
MERGEIT.OBJ +
$(OBJS_EXT)
$(PROJ).EXE
$(MAPFILE)
c:\msvc\lib\+
c:\msvc\mfc\lib\+
$(LIBS)
$(DEFFILE);
<<
        link $(LFLAGS) @$(PROJ).CRF

run: $(PROJ).EXE
        $(PROJ) $(RUNFLAGS)


$(PROJ).BSC: $(SBRS)
        bscmake @<<
/o$@ $(SBRS)
<<
```

# APPENDIX B

# 28F016XD SIMM VENDOR INFORMATION

| Company | Address | Telephone/Fax | Contact Person |
|---|---|---|---|
| First Tech Corporation | 12201 Technology Blvd.,<br>Suite 130<br>Austin, TX 78727 | Tel: (512) 258-3570<br>Fax: (512) 258-3689 | Karl Baumbach |
| Lifetime Memory Products Inc. | 305 17th Street<br>Huntington Beach, CA 92648 | Tel: (714) 969-2421<br>Fax: (714) 960-0638 | Paul Columbus |
| Smart Modular Technologies | 45531 Northport Loop West<br>Fremont, CA 94538 | Tel: (510) 623-1231<br>Fax: (510) 623-1434 | Bill Johnston |
| Syntaq Limited | 16-17 Enterprise Court<br>Cramlington, Northumberland<br>NE23 9LZ England | Tel: 44-670-731-866<br>Fax: 44-670-731-741 | |

**NOTE:**

Product Briefs of these SIMM vendors are also available on the Intel FaxBack Service.

# APPENDIX C
# EXAMPLE CODE FOR 82420ZX PCIset RECONFIGURATION
## (TO ACCOMMODATE 28F016XD TIMINGS)

```
/*********************************************************************
 *                                  Copyright Intel Corporation 1995
 *
 *    DESCRIPTION:
 *       82420ZX PCIset Configuration for Flash SIMM Devices
 *
 *    AUTHORS:
 *       Andrew Gafken and John Pierron
 *
 *    FILE: setchip.c
 *
 *********************************************************************/

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>


void main()
{
printf("\n82420ZX PCISet Configuration\n");

  asm {
               mov     dx, 0cf8h
               mov     al, 0f0h
               out     dx, al

               mov     dx, 0c055h
               mov     al, 0h
               out     dx, al

               mov     dx, 0c060h
               mov     al, 02h
               out     dx, al

               inc     dx
               mov     al, 04h
               out     dx, al

               inc     dx
               mov     al, 0ch
               out     dx, al

               inc     dx
               mov     al, 0ch
               out     dx, al

               mov     dx, 0cf8h
               mov     al, 00h
               out     dx, al
         }
```

```
}
```

# APPENDIX D

# EMBEDDED DIRECT-EXECUTE OPERATING SYSTEM PROVIDERS CONTACT INFORMATION

| Application | Product | Company | Contact Telephone |
|---|---|---|---|
| Direct-Execute DOS and DOS-Based O/S | Microsoft MS-DOS ROM-Executable, MS-Windows ROM Executable | Annabooks Corporation | 800-462-1042 (619-673-0870) |
| | ROM-DOS | Datalight | 800-221-6630 (360-435-8086) |
| Real-Time Embedded O/S | Chorus Nucleus | Chorus Systems Inc. | 503-690-2300 |
| | LynxOS | Lynx Real-Time Systems, Inc. | 408-354-7770 |
| | OS/9 | Microware Systems Corp. | 800-475-9000 (515-224-1929) |
| | pSOSystem | Integrated Systems, Inc. | 408-980-1500 |
| | PSX | JMI Software Systems, Inc. | 215-628-0840 |
| | QNX | QNX Software Systems Ltd. | 613-591-0931 |
| | Venix | VentureCom Inc. | 617-661-1230 |
| | VRTX | Microtec Research | 800-950-5554 |
| | VxWorks | WindRiver Systems | 800-545-WIND (510-748-4100) |
| Handheld Computing O/S | Geos | GeoWorks | 510-814-1660 |
| | MagicCap | General Magic | 408-774-4000 |
| | Newton O/S | Apple Computer Corp. | 408-996-1010 |

| | |
|---|---|
| Filename: | 292168_1.DOC |
| Directory: | C:\TESTDOCS\DOCS |
| Template: | C:\WINDOWS\WINWORD6\TEMPLATE\ZAN____1.DOT |
| Title: | E |
| Subject: | |
| Author: | s |
| Keywords: | |
| Comments: | |
| Creation Date: | 09/17/95 9:53 AM |
| Revision Number: | 31 |
| Last Saved On: | 11/28/95 10:16 AM |
| Last Saved By: | Ward McQueen |
| Total Editing Time: | 227 Minutes |
| Last Printed On: | 11/28/95 10:17 AM |
| As of Last Complete Printing | |
| Number of Pages: | 21 |
| Number of Words: | 4,792 (approx.) |
| Number of Characters: | 27,317 (approx.) |