AP-618

APPLICATION NOTE

Software Concerns of Implementing a Resident Flash Disk

KIRK BLUM
TECHNICAL MARKETING ENGINEER

December 1995

Order Number: 292173-001

Information in this document is provided in connection with 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 retains the right to make changes to these specifications at any time, without notice. Microcomputer Products may have minor variations to this specification known as errata.

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

†Since publication of documents referenced in this document, registration of the Pentium, OverDrive and iCOMP trademarks has been issued to Intel Corporation.

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

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

SOFTWARE CONCERNS OF IMPLEMENTING A RESIDENT FLASH DISK

CONTENTS

CONTENTS

PURPOSE

The purpose of this Ap-Note is to detail the software aspects of implementing an Intel Resident Flash Array (RFA) as a Resident Flash Disk (RFD) in an embedded system with commercially available Flash Translation Layer (FTL) software.

An RFD can help mitigate many issues that constrain the storage subsystem for embedded systems. Those issues include:

- Embedded systems must operate in harsh environments; they are dropped, banged, vibrated, overheated, etc.:
  - — A Flash RFD is rugged and non-volatile because it is solid-state and has no moving parts.
- Embedded systems must have high performance:
  - — A Flash RFD, because it is basically memory, not rotating magnetic media, has extremely fast access capability.
- Embedded systems can be battery operated and must be low power:
  - — A Flash RFD is very low power as it is silicon and has no motors to spin, or servos to move. The standby/deep dower-down mode current draw of a flash part is typically measured in micro-amps (μA).
- Embedded systems may be limited in size:
  - — A Flash RFD requires very little board space and is tiny compared to even the smallest hard drive.
- Embedded systems tend to be price sensitive:
  - — A Flash RFD is relatively inexpensive.
- Embedded systems have to be very flexible:
  - — A Flash RFD is adaptable to just about ANY situation.

For the purposes of this document, we will use a "real-life" example based on the Intel "Ultra Low Power" (ULP) Intel486™ SX Processor Evaluation Board and M-Systems TrueFFS* FTL flash software for DOS.

Familarity with the PCMCIA PC Card Standard is helpful for understanding the software pieces of our example. Please see the Reference section for information on contacting PCMCIA in order to obtain a copy of "The Standard."

INTRODUCTION

The exacting requirements for today's embedded processor platform market makes extreme demands upon all of the associated embedded sub-systems. Among those are the data and code storage functions. Intel's Flash components are well suited to implement a ruggidized, low-power, low-cost storage system, in the form of a Resident Flash Disk (RFD).

A RFD is basically a Resident Flash Array (RFA), which is one or more flash components typically designed onto the base system, used with software to make it operate as a drive.

The flash that comprises the RFD has the unique capability of non-volatility (like ROM) combined with in-place erase and write (similar to RAM). With the proper software, Intel Flash can function as a solid-state disk drive, without exhibiting the disadvantages inherent in a rotating/magnetic-media drive. In general, as well as for the purposes of this Ap-Note, the proper software is FTL. Although there are quite a few sources for FTL, we base our example on M-System's[1] TrueFFS* FTL[2]. Please keep in mind that though the details of the implementation will be specific to TrueFFS, the general concepts and functions will apply to virtually any of the available FTLs or other flash managers and/or flash filing systems. This also applies for the example target platform. For our example, the specific target platform is the Intel Ultra Low Power (ULP) 486 SX Evaluation Board. However, the general concepts presented here will apply to virtually any embedded system.

[1]M-Systems provides a full range of Turn-key FlashDisk solutions that uses TrueFFS, are plug and play and uses Intel advanced Flash components. These solutions include ISA Bus and PC104 FlashDisks,and the DiskOnChip* which is a Plug & Play FlashDisk module in a standard 28/32 pin DIP JEDEC compatible package.

[2]Some of the information in this document is taken from M-System's TrueFFS documentation.

SYSTEM REQUIREMENTS

We are using the ULP 486 SX for our example because it is a high performance demo platform which exhibits characteristics that are pertinent to a wide range of embedded applications. Its compact size, high-degree of functionality, high performance, extreme flexibility, availability of many development environments and operating systems, and PC desktop/notebook compatibility make it easily adapted to the many and varied requirements placed on embedded systems. It also contains a considerable quantity and variety of flash, including a simple RFD which is well suited for use with M-Systems TrueFFS FTL. Figure 1 shows the general layout of the ULP 486 SX

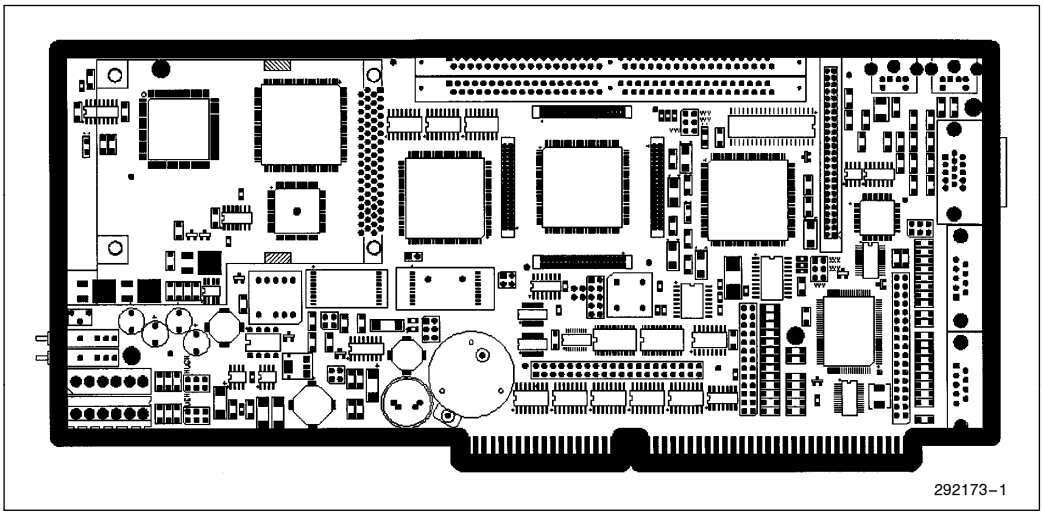

Figure 1. Intel486™ SX ULP Evaluation Board

The following chart gives an operational overview of our target embedded platform. Included is an assessment of how our specific features might apply to a different implementation.

Item	This Specific Example	Your Application
Processor:	Intel486™ SX ULP (80x86)	Any processor can be used, however a version of FTL for your particular processor will be required.
Flash:	RFD- 1 x Intel 28F008	Varies. Could be an RFD based on any variety or quantity of Intel Flash components, or it could be a Flash Card plugged into a PC Card socket.
Flash Media Manager:	FTL (M-System's TrueFFS*)	Any FTL. The low level routines of the FTL you select will probably be different but will have similar requirements and functions.
Operating system:	DOS-FAT	A ''sector'' based filing system of some sort is required for most FTLs. The FTL typically needs to be ''ported'' to your specific operating system (if other than DOS).
Other Flash:	PC Card Socket (Cirrus 6710)	Varies. A typical system will usually only implement a single ''variety'' of flash. A PC Card socket accepts flash in a Flash Card.
	XD Simm (not covered)	Varies. If your application would benefit from execute in place (XIP) on the memory bus, then you might consider using this style of flash. Please contact your Intel representative for more information on XIP and Intel XD Flash.
Low Level FTL Interface:	Socket Services Like Interface	Varies depending on the particular FTL you decide to implement in your design. However, most of the elements of, and all of the functionality described here in the Socket Services like routines, are typically required in the low level routines of any flash undertaking.

WHY DO I NEED SOFTWARE FOR FLASH?

One of the unique characteristics of flash technologies is the typical requirement that a flash cell be erased before it can be written.[3] For Intel Flash, generally,[4] a flash cell starts as erased to a 1 (one), and can be written/programmed to become a 0 (zero). Early flash chips were organized such that the entire chip was erased at one time (called "bulk erase"). Newer flash chips are organized into erase blocks—typically 64 KBytes[5] in size. Arranging the flash chip into erase blocks provides a mechanism for software to manage the data stored on the chip. Keep this in mind, as there will be a quiz later.

Traditional data storage programs, such as DOS and other file systems, are designed to update or rewrite data in place. Here is a specific example: think of how DOS deals with the File Allocation Table (FAT) when you write a file. It takes a specific sector on the media and rewrites it with the new or changed data concerning the file. Add or change a file and the sector gets changed. The same sector or sectors are constantly rewritten with new or different FAT data. This is a reasonable mode of operation for rotating magnetic media but at odds with the basic operation of flash. Another difference between drives and flash is the traditional drive sector (512 bytes is typical) tends to be quite a number of bits smaller than the typical flash block.

Flash management software can solve these issues in a number of ways, two of which follow:

1. When rewriting any sector (512 bytes of data typical), the flash manager can perform the following steps:
   a. Save off the rest of the "good data" stored in the larger (64 KB typical) flash erase unit. This will be comprised of the other valid "sectors" in the flash block. It will have to put the good data "somewhere" else such as a spare flash block.
   b. Next, erase the current block where the data wants to be rewritten.
   c. Write in the new data.
   d. Restore the rest of the good data from the temporary storage.
   e. Finally clear out the temporary storage location (i.e. erase the spare flash block).

Whew, this sure looks and sounds tedious and highly inefficient! No matter what the style of flash, a purely erase before write type flash management scheme is definitely NOT the best overall performance solution.

2. When rewriting a sector it can:
   a. Write the updated data to another free/erased portion of the flash.
   b. Then point a lookup entry, or translation table to the other location in flash
   c. Finally, the old portion of flash is marked as "dirty" or deleted for later clean-up.

The second method is what, in general, the FTL software does for flash. This is the most write efficient, highest performance and most highly recommended method for handling flash.

FTL IS IT!

Flash Translation Layer is a robust, widely accepted, industry standard flash manager. It is widely available from a multitude of sources. Its function is to take disk drive specific software requests and convert/translate them to flash media accesses. It handles flash blocks by creating small, virtual, sector-sized (usually 512 byte) blocks out of the larger flash blocks. Additionally, FTL handles the "special needs" of flash such as handling the special read, write, and erase requirements that flash technology exhibits. Also, many, if not all, of the varieties of FTL offer a boot solution. This makes the operating system loadable from flash as if it were

[3]See the section on Bit Twiddling for the exception to this "rule".

[4]Current flash technology (for example i28F008 and i28F016) uses 1 flash cell per bit. Intel has announced a technology that in the near future will be able to represent more bits per flash cell using a technology known as Multi-Level Cell or MLC. The concepts presented here may not all apply to MLC technology.

[5]Flash block size and organization may vary from component to component- please refer to the data-sheet for the particular flash component you are using. In general, RFA/RFD implementations use Intel FlashFile™ components (i28F008, i28F016, etc.) which are organized in symmetrical 64KB and 128KB erase blocks. Boot Block components can be used but pose extra burdens on the erase and write routines in order to adapt to the non-symmetrical erase units present in the components.

loading from a more common bootable device such as a floppy or hard disk (more on this in a later section).

In the next section, we will describe generally how FTL works.

FTL FUNCTIONAL OVERVIEW: WHAT IS A FLASH TRANSLATION LAYER?

As we have said, FTL is a sector based flash manager that provides logical to physical sector translation. Thus the name Flash Translation Layer. FTL performs sector mapping to allow Flash to appear as a drive-like, sectored, rewrite "in place" type storage media. While the host file system sees the Flash card or resident Flash array (RFA) as a continuous sectored medium, FTL relocates these sectors transparently to the operating system, and tracks the logical-to-physical relationship. Figure 2 provides a simplified graphical representation of the sector translation and relocation that occurs with FTL. The MAP reference in the diagram is explained in greater details in the "FTL Functional Details" section of this document.

This logical-to-physical mapping allows the Operating System to concern itself with only file operations. Because the O/S already oversees these file operations, the FTL solution can provide compatibility with existing applications and media utilities while presenting a small code footprint. Of important note here is the fact that FTL is able to relocate sectors to any position in the Flash media, making large Flash blocks appear as smaller erasable sectors. This is basically what FTL does to earn its keep.

WHERE DOES FTL FIT IN THE SOFTWARE SCHEME?

FTL, for the most part, needs to intercept the data store/retrieve requests somewhere between the software wanting to talk to the drive and the low level routines that specifically read or write data to the drives. In a x86 PC like system, and on other types of platforms, there are typically two logical places to grab the drive access routines in order to handle the drive requests actually being serviced by flash. Each has its advantages and disadvantages which we will discuss.

Block Device Driver

One of the most common ways to intercept the normal drive handlers is to have the operating system load a Block Device Driver which inserts itself in the lower levels of the operating system. M-System's FTL TFFS.COM and TFFSCS.COM are two examples of a

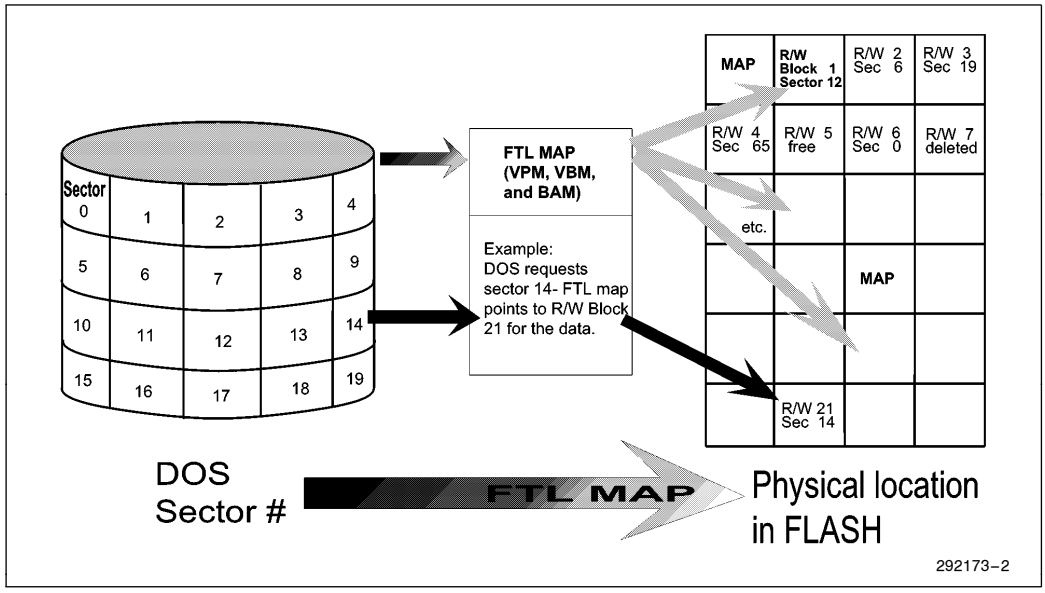

Figure 2. FTL Sector Relocation

FTL Block Device Driver. The major advantage to using the Block Device Driver approach is that it is specific to the operating system and therefore can have better knowledge about how the filing system works. Because it knows how the operating system works, it can use performance enhancement tricks specific to that operating system. One such trick used by M-System's TrueFFS DOS FTL driver is a technique called "FAT Snooping." This allows the FTL to monitor writes to the File Allocation Table to know when DOS has deleted a sector thus allowing the FTL to perform clean-up more efficiently.

The major disadvantages to grabbing the drive control at this level are:

— The driver must be operating system specific. This requires that a new driver be developed for each operating system. (Hmmm, this is also an advantage- let's not go into an explanation on the inherent duality of nature!)

— You cannot boot from the flash as, typically, the device drivers, including the FTL block device drivers, are loaded AFTER the operating system starts. You can't read from the flash to boot as the driver has to be loaded by the operating system you are trying to load. We will go into more boot issues in the next section, as well as later in this paper.

— Software that bypasses the operating system to talk to the low level Int13h BIOS routines OR talks directly to the drive controller hardware will not work with a flash/FTL type drive.

Figure 3 gives a general layout of this layered approach, where FTL works with the existing File System to control the Flash media.

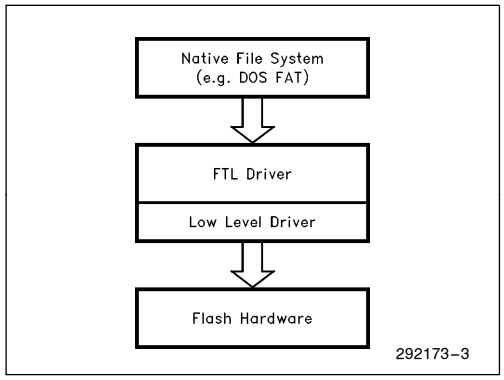

Figure 3. FTL Block Device Driver Software Layers

BIOS Interception Driver

The other location to intercept drive requests in order to have flash handle the data is in the lowest levels of the system itself. In the vast majority of systems, that lowest level software is something known as BIOS. BIOS stands for Basic Input/Output System. It provides the services to handle the lowest level interface tasks to the various standard I/O systems like drives, serial ports, parallel ports, keyboard, etc. In x86 PC-like systems, the INT13h handler is the one responsible for servicing the system drives (HDD and FD) and it is this software vector that needs to be intercepted by an FTL. M-Systems provides a driver called TFFSBIOS.EXB which is loaded as an Expansion ROM and intercepts INT13h.

The advantages to this approach are:

— Boot capable—the BIOS uses INT13h BIOS calls to boot so if the FTL/flash device is set up as one of the standard boot drives then the OS will look to the FTL partition on the flash to try and boot.

— Software that bypasses the operating system and interfaces directly to the low level BIOS INT13h routines will still work on flash.

Disadvantages:

— Flash Software must be present in non-volatile memory such as system BIOS flash. This is more complicated to deal with than a driver stored on a disk drive and loaded to memory.

— No performance tricks such as snooping can be performed as we do not know what will be accessing us.

— The software that accesses the drive must be "well behaved" in order for the interception to work. Software that tries to talk directly to a HDD controller will NOT work even with a BIOS Intercept driver.

Now that we have seen the Why and Where FTL works, let's take a look at the specific details of HOW it performs its magic.

FTL FUNCTIONAL DETAILS—THE GUTS OF THE BEAST

In these sections, in reasonable technical depth, we will describe the details of the FTL structures as defined by the industry standard FTL specification (available from PCMCIA).

FTL Format Overview

When a FTL format occurs, all of the FTL overhead, such as the Virtual Block Maps (VBMs), FAT table, Root Dir Entries, and any other structures, is put in the first two logical blocks. Figure 4 illustrates how multiple flash chips in a flash array might be organized with the FTL structures after a format.

When writing a new sector or changing an existing one, FTL looks for available (erased) flash in the form of an unallocated Read/Write Block, writes the data, and sets up the pointers and maps accordingly. Sectors containing valid data for a specific file can be scattered anywhere on the flash memory in this fashion. You never know exactly where the data will go physically in the flash, but rest assured that you will definitely get it back when you ask for it.

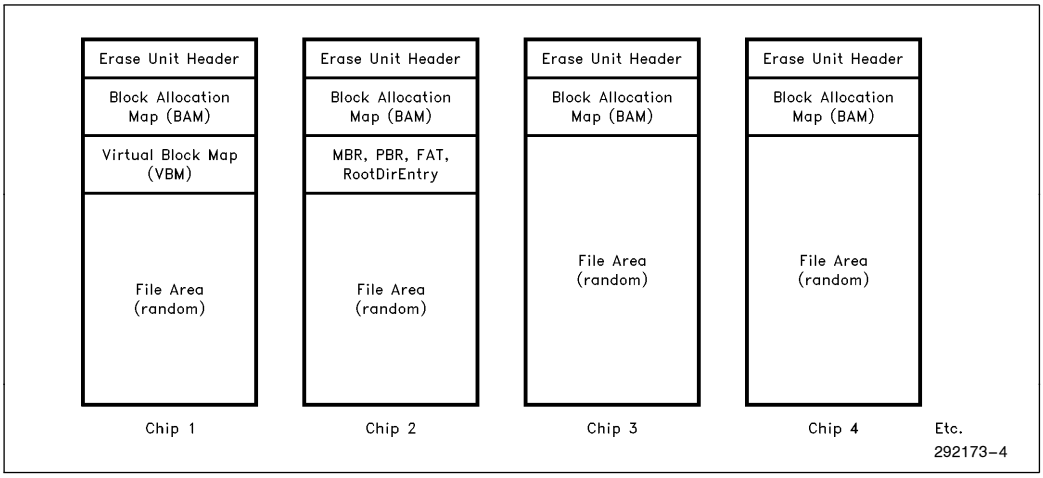

Figure 4. Typical FTL Overhead Organization

Erase Unit Header and Block Allocation Information

For allocation purposes, each Erase Unit is evenly divided into arrays of Read/Write Blocks of equal size. For the purposes of DOS, and also to appear more "drive-like", the Read/Write blocks tend to be 512 bytes in length. You can also think of them as sector sized.

Figure 5 illustrates the multiple Read/Write Block per Erase Unit concept.

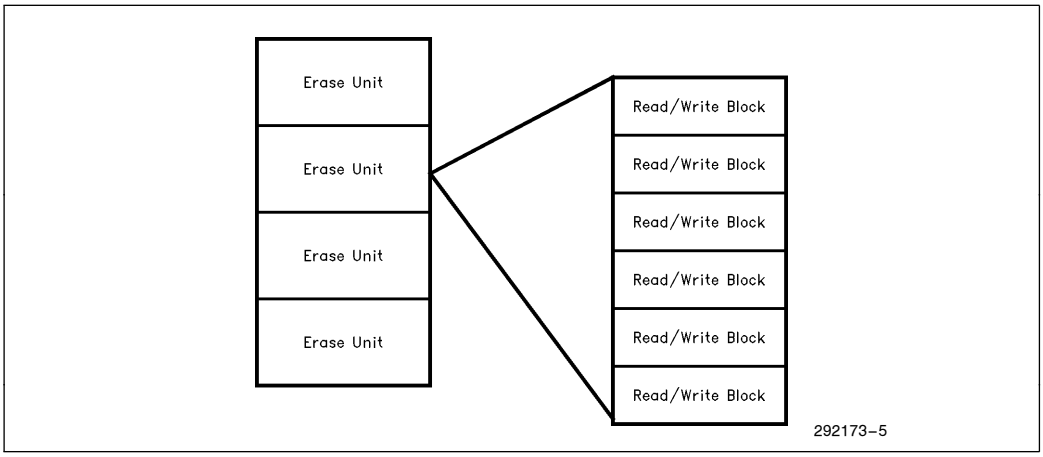

Figure 5. Erase Unit Organization

The size of a Read/Write block is the same as a virtual block as viewed by FAT. A Read/Write block used to store a FTL overhead structure is called a Control Unit. The BAM, VBM, EUH, etc. all reside in various Control Units. The first Read/Write block of each Erase Unit is a Control Unit which contains the Erase Unit Header (EUH) for that Erase Unit. It includes specific information about the Erase Unit and global information about the format of the FTL partition. Table 1 details the contents of a typical EUH.

Table 1. Erase Unit Header Fields

Offset	Field
0	LinkTarget Tuple
5	DataOrganization Tuple
15	Number of Transfer Units
16	Erase Count
20	Logical Erase Unit (LUN) Number
22	Read/Write (sector) Size
23	Erase Unit Size(in log2 form)
24	First Physical Erase Unit of Start of Partition
26	Number of Erase Units
28	Formatted Size
32	First Virtual Map Address on the Media
36	Number of Virtual Map Pages
38	Flags
39	Code
40	Serial Number
44	Alternate Erase Unit Header Offset
48	BAM Offset
52	Reserved

Each Erase Unit also contains allocation information for all of the Read/Write Blocks within the unit.

For each Read/Write Block, a 4-byte value tracks the block's current state. This is the case for all Read/Write blocks in that Erase Unit. This section, comprised of one or more Control Units, is located right after the EUH Control unit and is called the Block Allocation Map or BAM. At any point in time, a Read/Write Block in an Erase Unit may be free, deleted, bad or allocated. Table 2 details the exact status values used to indicate those states.

Table 2. BAM Status Values

Value	Meaning
FFFFFFFF	Free
00000000	Deleted
00000070	Bad
00000030	Control
xxxxxx40	Data of Map Page
xxxxxx60	Replacement Page

In Figure 6, we show an example of what a BAM might look like after the FTL partition has been used.

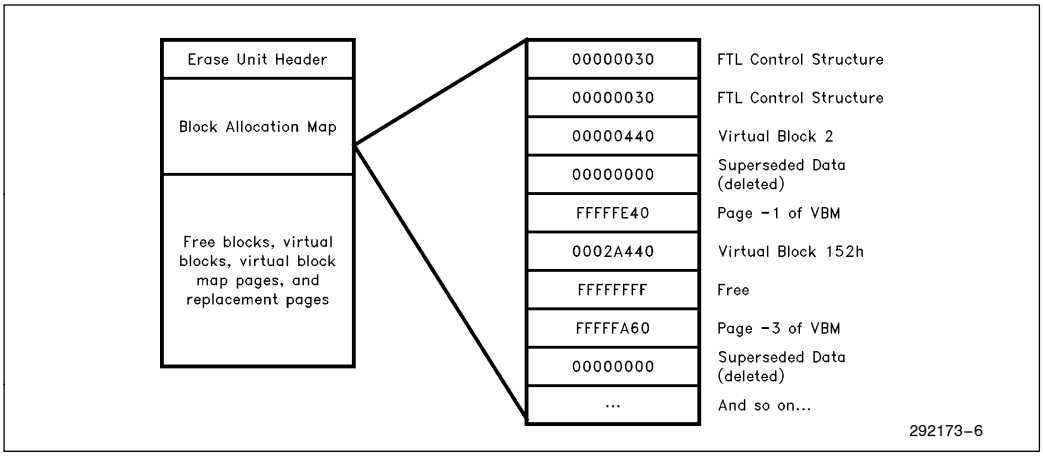

Figure 6. BAM Example

The BAM entries for Virtual Block Maps are negative numbered in contrast to the BAMs for Virtual Block data which are positive numbered. This is the only way to distinguish between the two. In the example in Figure 6, the 00000440 in the BAM is the virtual block data number 2 (each block is 200 hex bytes) while the FFFFFE40 BAM entry is the last page of the VBM.

The number of Control Units used depends on the size of the BAM which depends on the ratio of Erase Unit size to the Read/Write block size

Virtual Block Map

The FTL uses a data structure known as the Virtual Block Map (VBM) to map requests for sectors from the higher level software layers to the flash. The file system *thinks* it is requesting physical sectors but through FTL, they are actually virtual sectors, also known as Virtual Blocks. FTL maps the Virtual Block requests through the VBM to obtain a logical address. The physical address on the flash is then determined from the logical address and the data is returned to the filing system.

The VBM is comprised of an array of 32-bit entries, each of which represents a logical address on the media where a Virtual Block's data is stored. The Virtual Block number requested by the higher level software layer is used as an index into this array. Please see Figure 7 for an example of a VBM.

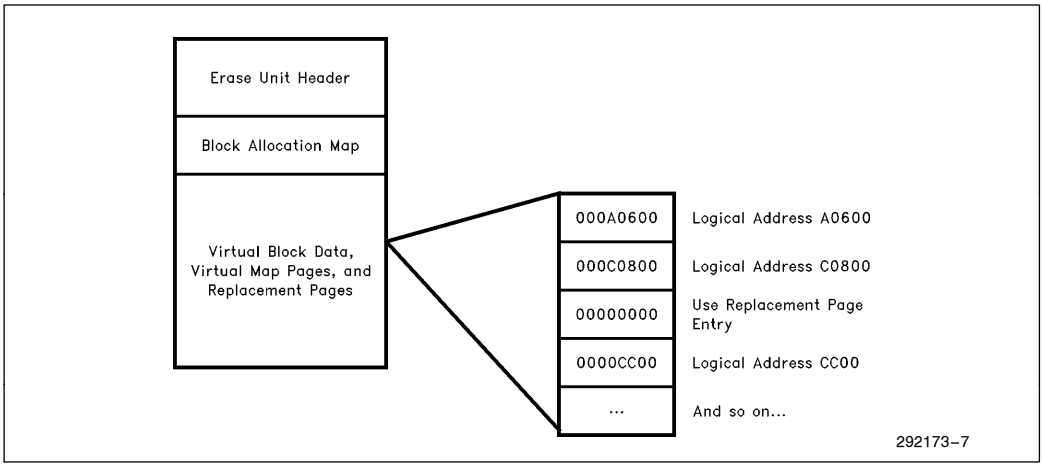

Figure 7. Virtual Block Map Example

The VBM is subdivided into pages. Each page of the VBM is the same size as a Virtual sector of the FAT file system. Since each entry in the VBM is 4 bytes, each page holds (Virtual Blocks/4) number of entries, and from that we can figure out how much virtual space each map represents and how many pages we need to map the whole virtual space.

Space is always reserved on the media to store a VBM large enough to track the allocation of all the Virtual Blocks on the flash. However, when the flash is formatted, the FTL may choose to keep only a portion of the VBM on the media, and store the rest of it in RAM. The amount of VBM stored on the media is indicated by the FirstVMAddress field of the Erase Unit Header (see Table 1). If the first VMAddress is set to 0, the FTL maintains all of the VBM entries on the media. If the FirstVMAddress exceeds the FormattedSize, none of the VBM entries are maintained on the media by the FTL.

When all or a portion of the VBM is not maintained on the media, it has to be reconstructed in RAM every time the system is booted up running FTL, or, as in the case of a PC Card slot with flash cards, every time a card is re-inserted. FTL uses the BAM information to fill out the entries of the VBM in RAM.

If a VBM entry is all ones, the Virtual Block does not exist on the media. If it is all zeroes, the logical address of the Virtual Block is described on a Replacement Page.

Replacement Pages

Each page of the Virtual Block Map may have a Replacement Page. Values in a Replacement Page override entries in the original VBM pages. Replacement pages are allocated from free Read/Write Blocks in any Erase Unit. The FTL locates allocated Replacement Pages by scanning the block allocation information on the media. This scan may be performed when the media is inserted in the host system or when a VBM entry of zero is encountered. Replacement Pages cannot themselves be replaced. The block allocation information entry for a Replacement page uses the same virtual address as the original VBM page. FTL distinguishes between the two by looking at the last byte of the entry: 40H for VBM pages and 60H for Replacement Pages.

SOCKET SERVICES—WHAT IS IT AND WHY DO I NEED TO USE IT?

The lowest level of our example RFD software solution is a PC Card-like Socket Services (SS) subset. The illustration in Figure 8 gives a general overview of the details of the full PC Card standard software layers.

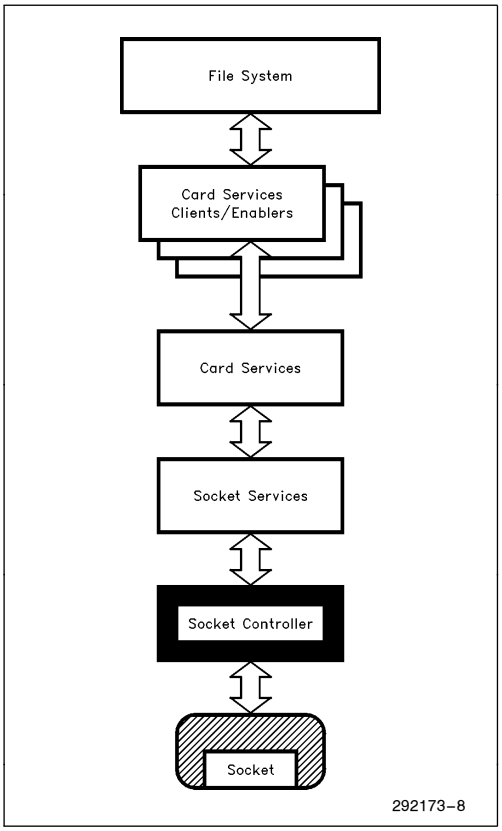

Figure 8. PC Card Software Suite General Layout

The Socket Services (SS) standard gives an industry recognized, standardized, interface method for other software (i.e. the FTL core) to interact with the low level routines. This separation of the low level routines from the other PC Card software allows the often different and hardware specific portion of the code to be cleanly separated from the (mostly) unchanging higher level functions. Please refer to Appendix B for a listing of the full PC Card Socket Services functions for reference.

Other FTL implementations may organize the low level interface differently than the SS interface we describe here but will typically have similar requirements, and utilize similar sorts of functions. The following figure 9 illustrates the software layers for our specific implementation.

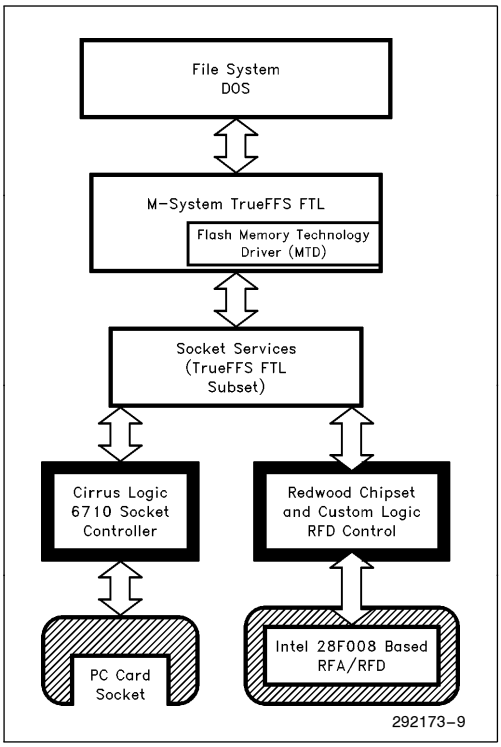

Figure 9. M-System's FTL Software Layers

Implementation Issue: Real-Mode Flash Memory Sliding Window

There is a technical requirement of the example FTL that is difficult to describe but relatively easy to implement. It is something called a real-mode sliding window to the RFD. "A real-mode sliding what?" you might be asking yourself right now. Basically, this a method for accessing the flash memory of the RFD that is mapped "somehow" into the lower one megabyte region of memory. That region is known as the real-mode, or DOS memory region. Typically, some form of hardware such as the main system chipset, special logic (like a PLD), or a combination of the two, will cause the RFD to be accessible in a real-mode part

of memory. That part of memory is usually in the upper memory block (UMB) region (A000:0 FFFF:F) starting in the D000 page. For the ULP platform, a special capability of the Cirrus Logic PicoPower Redwood chipset is used along with logic in a Xilinx PLD to do just that. "But wait a minute, what about the sliding part??" Ahhh, here is the tricky part. The flash memory size of the RFD is usually quite large in order to implement a reasonable size flash drive. The range of RFD sizes will normally start at a minimum one megabyte, single 28F008 8 megabit FlashFile component. The ULP RFD is this exactly. If we were to map the entire RFD into real-mode memory, the one megabyte of flash memory would take up the ENTIRE real-mode range leaving no room for DRAM, operating system (DOS), video RAM, system BIOS, other BIOS, etc! The trick is to only map into real-mode a small portion of the flash, or window, and then move the window around to be able to access the entire address range of the flash RFD. This concept is similar to the LIM/EMM memory scheme. Our example FTL is quite happy with a minimum 4 kilobyte window into the flash and will work well with that size or whatever the hardware is capable of supplying. The ULP logic controlled window can be programmed for 16K, 32K, or 64K windows but for the ULP implementation, we have settled on using 16K windows in order to conserve the UMB area. A window control register is used to pick where in the flash we are looking. The exact operation of the ULP registers will be discussed in detail in a following section.

IMPORTANT:

It is important to remember to exclude the real-mode UMB flash window from use by a memory manager. The window will not work if a memory manager, such as EMM386, has paged DRAM into the memory space needed by the sliding window!

The following Figure 10 attempts to illustrate the concept of the real-mode sliding window.

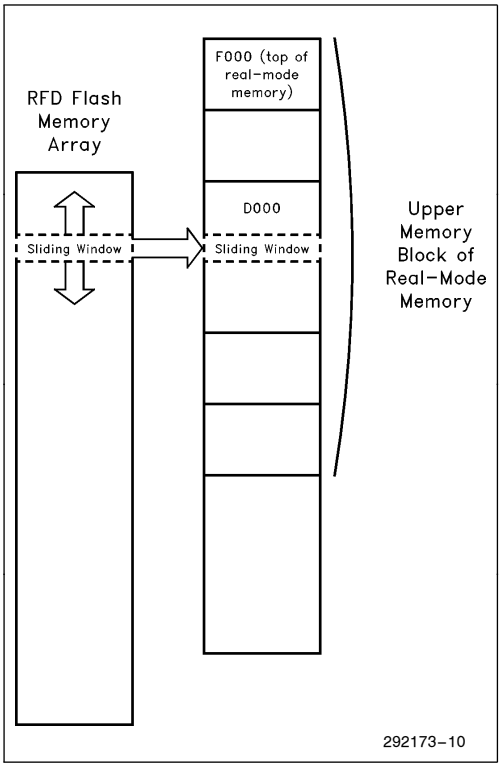

Figure 10. Real-Mode Sliding Window

The real-mode sliding window is the most common flash mapping technique as it eliminates the difficulties of accessing protected mode addresses from a DOS driver. In theory, accessing a RFD that is mapped in protected mode is possible, however, the majority of FTL solutions including the M-Systems FTL are limited to operating strictly from real-mode. If your microcontroller, embedded system and/or operating system enjoys the ability to address flash memory in a directly mapped, linear fashion, then the examples of windowing in the code listings that follow do not apply to you.

Now let's take a look at the requirements for the FTL socket services.

Low Level: FTL Socket Services Subset

The Socket Services required for our example is a simple subset of the PC Card standard socket services. As you have seen before, Figure 3 shows the typical software layers of FTL and Figure 9 shows our reference software's specific layers. For the socket services/low level driver layer, there are only 11 functions out of the full PC Card standard 30 valid functions (refer to appendix B) that you need to support for the M-System's FTL. The required functions are as follows:

Name	Number	Description
GetAdapterCount	*(80h)*	Returns the number of adapters supported by all Socket Services handlers in the host system. Also used to determine if one or more Socket Services handlers are installed.
GetSocket	*(8Dh)*	Returns the current configuration of the socket identified by the input parameters.
GetSSInfo	*(83h)*	Returns the version/ compliance level of the Socket Services interface supporting the adapter as specified by the input parameters. It also identifies the adapters serviced by the handler.
GetStatus	*(8Fh)*	Returns the current status of the card, socket, controls and indicators for the socket identified by the input parameters.
GetWindow	*(88h)*	Returns the current configuration of the window specified by the input parameters.
InquireAdapter	*(84h)*	Returns information about the capabilities of the adapter specified by the input parameters.
InquireSocket	*(8Ch)*	Returns information about the capabilities of the socket specified by the input parameters.
InquireWindow	*(87h)*	Returns information about the capabilities of the window specified by the input parameters.
SetPage	*(8Bh)*	Configures the page specified by the input parameters. Only valid for memory windows. This service is unsupported by PC Card-32.
SetSocket	*(8Eh)*	Sets the current configuration of the socket identified by the input parameters.
SetWindow	*(89h)*	Sets the configuration of the window specified by the input parameters.

FTL Socket Services: The CODE

In this section, we will give you the details of, and where to look for listings of the important parts of the actual code. This code, or a reasonable facsimile thereof, was used to implement the socket services for M-Systems FTL on the ULP Evaluation board. Please be aware that because the board also contains a Cirrus Logic 6710 Socket Controller, some of the code may be specific to it. The PC Card implementation of the ULP board does not fall into the scope of this Ap-Note, so we will be conveniently ignoring any such code here.

On the ULP board, the page register for sliding window control is located at I/O address 1096. The Window size and enable register is located at I/O address 2096. Watch for references to these ports as they will be important! Your design will probably locate these control ports in a differnt I/O location with different meanings for the bits. There is additional "flash" control logic handled by the PicoPower Redwood chipset. Within the chipset functions, there is advanced memory handling and address range decoder functions that are used to cause the RFA flash memory window to appear in the upper memory block (high memory range) of the system memory. The custom logic mentioned before handles sliding the window to the flash.

The code listings can be found in Appendix E. Due to the volatile nature of software, most likely the listings printed in this Ap-Note will NOT be the most current revision. The full, and current version of the code for this socket services portion of the flash software is available with the ULP Evaluation Kit.

BOOT ISSUES

As we promised earlier in this document, we will now face the issues involved with booting and how they relate to flash. The ULP Evaluation Board has the ability to boot from floppy disk, hard disk drive, flash RFD, or a PC Card. This necessarily places burdens on both the system as well as the software supporting the flash capabilities of the board (i.e., FTL/SS). The system BIOS, through the use of the BIOS Setup can control what device will be the primary boot device. When flash is the boot device, the following issues need to be handled.

The FTL software must load before boot and typically using the previously discussed INT13h interception method. In order to do this, both the FTL socket services and the FTL itself must load BEFORE the boot is attempted. This is accomplished by storing the software on the system as an option/expansion BIOS. When an x86 PC-like system is starting up, the system BIOS will go out into the UMB and search for code that has the signature (AA55) and proper checksum (zero-sum) of an option/expansion BIOS. The M-System TFFSBIOS code includes utilities to do this in their Integrators TrueFFS FTL package (see the TFFSBIOS Integrator's Guide, TFFSBIOS.DOC, included on the M-Systems disk). Because the ULP uses an Intel Boot Block for the system BIOS, it is a simple matter to store the FTL "Expansion BIOS" or "Option" code in an appropriate spot in memory. For the ULP board, the FTL Socket Services can be found at C800–C8FF. The SS code is approximately 4 KB and is designed to execute in place and as such only requires a small amount of RAM (1K) for its data segment. The FTL itself, in the form of M-System's TFFSBIOS, is located at C900–CFFF. Version 3.2.10 and lower are not specifically designed to execute in place and as such must be copied to DRAM for execution. During the TFFSBIOS code initialization, the code image and data segment are copied to the top of real mode memory (640K memory region) along with the data segment for the socket services. The system's available memory size indicator is appropriately decreased. The combined TFFSBIOS image, buffers, and data segments take a total of 36 KB–40 KB off of the top of memory. For version 3.2.20 and higher of TFFSBIOS, the code has been designed to operate in place (or execute-in-place) from the flash itself (or a "shadowed" version in Shadow Ram). In this case, only the data segments (including buffers) for TFFSBIOS and the socket services need occupy system ram. The combined requirements total up to approximately 12K taken off the top of memory.

Once the FTL code is recognized and loaded by the BIOS, the flash drive still typically needs to operate as a drive letter that will normally be interrogated by the system BIOS for a boot image. If there are no other "hard drives" installed in the system, this is not a problem as TFFSBIOS automatically grabs the first available "hard drive" spot which just happens to be the usual boot "hard drive." If another hard drive exists is the system, and you want to boot from the FTL flash drive, then special intervention is required. Fortunately, the M-Systems utilities give you an option to force the FTL drive to be drive C: (moving the other hard drives up a spot) and therefore making the FTL flash drive function normally in the boot sequence.

The last major boot issue, (and fairly obvious once you think about it), is the need for a bootable format to exist on the flash from which you are going to boot. The actual requirement is for a master boot record with valid boot information to exist in logical sector 0 of the boot device. This is accomplished in a similar fashion to formatting a hard drive or floppy disk to be bootable. Let's assume the flash in the RFD has never been formatted or has been totally or partially erased. All that is required is to boot up the ULP board normally from another boot device (i.e. a bootable floppy) then invoke the M-System's format utility **TFORMAT /1** (/1 assumes the RFD has been selected as the primary "boot" flash device and is therefore acting as socket 1 out of 2. Use /2 if the RFD is operating secondary to flash in the PC Card socket.) After the FTL format has been laid down on the flash by TFORMAT, it is advisable to reboot the system from the floppy in order for the system to recognize the "new" valid FTL drive. Next, a bootable master boot record has be created on the flash. This is accomplished with the operating system disk FORMAT utility, making sure to specify that you wish to transfer the system to the "disk". For DOS, use: **FORMAT /S** *d:* where *d:* is the drive letter of the FTL flash drive. With a properly formatted and boot image enabled FTL drive as the primary drive, a normal boot can then occur from the flash RFD or Flash Card.

FTL AVAILABILITY—I WANT IT, WHERE CAN I GET IT?

"Man, this FTL software sounds GREAT! It will do exactly what I want to do on my system! Where can I get it?" This is a very astute question. There are quite a number of sources for commercially supported FTL. The FTLs from each of the vendors has unique characteristics. Pricing, performance, adaptability, etc. vary from version to version. Please see Appendix A for a listing of the most popular FTL vendors. As stated before, M-Systems is the creator of the specific FTL upon which this document is based.

CONCLUSION

The general benefits of flash in an embedded system, and especially a resident flash array operating as a drive with FTL software, are quite obvious. The wide availability and market acceptance of the FTL standard (FTL Is It!) makes the implementation and use of flash in this way a simple matter. And although concepts presented here were presented on a specific platform with a specific set of features, the concepts do apply to virtually any system that needs this kind of capability. With some thought, and a little development work, you too can soon be enjoying the advantages of flash and FTL.

ADDITIONAL INFORMATION

References

Order Number	Document
290429	28F008SA 8-Mbit (1-Mbit x 8) FlashFile™ Memory Datasheet
272731	Embedded Ultra-Low Power Intel486™ SX Processor Datasheet
272755	Embedded Ultra-Low Power Intel486™ GX Processor Datasheet
292157	AP-605, Implementing a Resident Flash Disk with an Intel386™ EX Embedded Processor
272324	AP-477, Low Voltage Embedded Design
Contact Intel/ Distribution Sales Office	Ultra Low Power Intel486™ SX Evaluation Board Reference Guide/User's Manual

General Information Hotline

US/Canada:	*1 (800) 628-8686 or (916) 356-7599*
Japan/APAC:	*(916) 356-7599*
Europe:	*+440 793-69-6776*

Literature Orders:

US/Canada:	*1 (800) 548-4725*
International:	*Please contact your local Intel office*
BBS:	*(916) 356-3600 or +4401793-496340*
FaxBack:*	*1(800) 628-2283 or (916) 356-3105*

Revision History

Number	Description
-001	Original Version

APPENDIX A
FTL AVAILABILITY

M-SYSTEMS

TrueFFS (v3.2 and up are FTL)*
4655 Old Ironsides Dr. Suite #200
Santa Clara, CA 95054
(408) 654-5820
FAX: (408) 654-9107

System Soft Corporation

SS FTL
313 Speen St.
Natick, MA 01760
(508) 651-0088
FAX: (508) 651-8188

SCM Microsystem, Inc.

S-FTL (v3.0 and up are FTL)
131 Albright Way.
Los Gatos, CA 95030
(408) 370-4888
FAX: (408) 370-4880

Datalight, Inc.

CardTrick FTL
307 N. Olympic Ave. Suite 200
Arlington, WA 98223
(206) 435-8086
FAX: (206) 435-0253

APPENDIX B
PC CARD SOCKET SERVICES DESCRIPTIONS AND FUNCTION NUMBERS

AccessConfigurationSpace *(A2h)*
This function is for CardBus. It provides an interface for Card Services to read and write values in the CardBus configuration space.

AcknowledgeInterrupt *(9Eh)*
Returns information about which socket(s) on the adapter specified by the input parameters has had a change in status.

GetAccessOffsets *(A1h)*
Fills the indicated buffer with an array of offsets for adapters using register-based, I/O port access to PC Card memory address space. Used for adapter-specific, low-level, optimized PC Card access routines.
Cards that use memory windows, directly mapped into the system memory space do not support this function.

GetAdapter *(85h)*
Returns the current configuration of the specified adapter.

GetAdapterCount *(80h)*
Returns the number of adapters supported by all Socket Services handlers in the host system. Also used to determine if one or more Socket Services handlers are installed.

GetEDC *(96h)*
Returns the current configuration of the EDC (Error Detect and Correct) generator specified by the input parameters.

GetPage *(8Ah)*
Returns the current configuration of the page specified by the input parameters. Only valid for memory windows (WS__IO is reset for the window).

GetSetPriorHandler *(9Fh)*
Gets, or replaces the entry point of a prior handler for the Adapter specified by the input parameters.

GetSetSSAddr *(A0h)*
Returns code and data area descriptions and provides a way to pass mode-specific data area descriptors to a Socket Services handler.

GetSocket *(8Dh)*
Returns the current configuration of the socket identified by the input parameters.

GetSSInfo *(83h)*
Returns the version / compliance level of the Socket Services interface supporting the adapter specified by the input parameters. It also identifies the adapters serviced by the handler.

GetStatus *(8Fh)*
Returns the current status of the card, socket, controls and indicators for the socket identified by the input parameters.

GetVendorInfo *(9Dh)*
Returns information about the vendor implementing Socket Services for the adapter specified in the input parameters.

GetWindow *(88h)*
Returns the current configuration of the window specified by the input parameters.

InquireAdapter *(84h)*
Returns information about the capabilities of the adapter specified by the input parameters.

InquireEDC *(95h)*
Returns the capabilities of the EDC generator specified by the input parameters.

InquireSocket *(8Ch)*
Returns information about the capabilities of the socket specified by the input parameters.

InquireWindow *(87h)*
Returns information about the capabilities of the window specified by the input parameters.

PauseEDC *(99h)*
Pauses EDC generation on a configured and in-use EDC generator specified by the input parameters.

ReadEDC *(9ch)*
Reads the EDC value computed by the EDC generator specified in the input parameters.

ResetSocket *(90h)*
Resets the PC Card in the socket and returns socket hardware to its power-on default state.

ResumeEDC *(9Ah)*
Resumes EDC generation on a configured and paused EDC generator specified by the input parameters.

SetAdapter *(86h)*
Sets the configuration of the specified adapter.

SetEDC *(97h)*
Sets the configuration of the EDC generator specified by the input parameters.

SetPage *(8Bh)*
Configures the page specified by the input parameters. Only valid for memory windows. This service is unsupported by PC Card-32.

SetSocket *(8Eh)*
Sets the current configuration of the socket identified by the input parameters.

SetWindow *(89h)*
Sets the configuration of the window specified by the input parameters.

StartEDC *(98h)*
Starts a previously configured EDC generator specified by the input parameters.

StopEDC *(9Bh)*
Stops EDC generation on a configured and functioning EDC generator specified by the input parameters.

VendorSpecific *(AEh)*
As the name implies, this service is vendor specific. It is reserved for vendors to add proprietary extensions to the Socket Services interface.

APPENDIX C
BIT TWIDDLING

An interesting capability of Intel Flash, in general, is that a flash byte or word can be "written" as many times as you wish *without* having to first erase it and without affecting the component's lifetime. You might be thinking "Now wait just a minute here! You just said I have erase flash before I write." That is only necessary only for a certain circumstance: when you need any bits to change from a zero to a one. However, you can *rewrite* flash as long as bits are being changed from 1 to zero, or are not changing at all. The following example illustrates the point:

Flash Byte What	Flash Byte (hex)	(Binary)	OK or Fail
Flash byte Erase:	FF	1 1 1 1 1 1 1 1 ↓ ↓ ↓ ↓	
Flash byte write to AAh:	AA	1 0 1 0 1 0 1 0 ↓	OK!
Flash byte write to A8h	A8	1 0 1 0 1 0 0 0 ↓	OK!
Flash byte write to 28h	28	0 0 1 0 1 0 0 0 ↓ ↓	OK!
Flash byte write to 00h	00	0 0 0 0 0 0 0 0	OK!
Flash byte write to 01h	0	0 0 0 0 0 0 0 0	FAIL

Erase is required to rewrite this flash byte to anything other than 00h.

APPENDIX D
RELATED THIRD PARTY DOCUMENTATION

M-Systems TrueFFS User's Manual

M-Systems
4655 Old Ironsides Dr. Suite #200
Santa Clara, CA 95054
(408) 654-5820
FAX: (408) 654-9107

PT86C768 & PT86C718 "Redwood" Chipset Manual

Cirrus Logic / PicoPower
3100 West Warren Ave.
Fremont, CA 94538
(510) 623-8300
FAX: (510) 252-6020

PC Card Standard

PCMCIA
2635 North 1st Street
San Jose, California 95134
(408) 433-2273
FAX: (408) 433-9558

APPENDIX E
CODE LISTINGS

```
; $Log: SS.ASM $
; Revision 1.4  1995/09/08 12:55:56  mgianopulos
; InitSocketServices() - Bug fix when booting from PCIC slot.
; Added option to boot from RFA only:InitSocketServices()
;  &    GetBootDevice().
; Revision 1.1  1995/08/30 09:25:57
; Initial revision
;
;  ------------------------------------------------------
;  July/Aug. 1995          Mark Gianopulos - Intel Corp.
;  Modified for 486sx(TM) ULP  RFA & PCMCIA Socket
;  ------------------------------------------------------
;
;     Rev 3.3   28 Feb 1994 10:41:26
;  Fixed ResetSocket bug
;
;     Rev 3.2   08 Sep 1993 21:18:14
;  Version 2.01
;
;     Rev 3.1   01 Sep 1993 11:31:06
;
;     Rev 3.0   21 Jul 1993 11:38:34
;  Speed & IRQ additions
;
;     Rev 1.4   25 Jun 1993 12:47:34
;  No window, Step B, and other changes
;
;     Rev 1.3   30 Mar 1993 10:42:38
;  GetVersion change
;
;     Rev 1.2   07 Mar 1993 16:58:14
;  Power table correction
;
;     Rev 1.1   18 Jan 1993 11:44:04
;  SS Ver 2.0 final draft
;
;     Rev 1.0   12 Jan 1993 16:08:48
;  Initial revision.
;

        PAGE   78, 132
        TITLE  SS.ASM
```

292173-11

```
COMMENT
/*************************************************************************

       Title:           Intel ULP 486sx RFA Socket Services

                        Copyright (c) 1995 Intel Corp.

       Author:          Amir Ban
                        M-Systems, Ltd.
       co-author:       Mark Gianopulos
                        Intel Corp.

       Copyright (C) by M-Systems Ltd. 1992.  All rights reserved.

*************************************************************************/

.486

INCLUDE      SSDEFS.INC         ; global definitions
INCLUDE         RFA.INC
INCLUDE        PCIC.INC
INCLUDE         VER.INC                 ; version number

             ASSUME  CS:NOTHING, DS:NOTHING, ES:NOTHING

ABSO          SEGMENT AT 00H PUBLIC USE16     ; Zero page definition for
vectors
ABSO        ENDS

DGROUP       GROUP  _TEXT, _DATA, _INIT

; ---       Subroutine definitions

_TEXT        SEGMENT BYTE PUBLIC 'CODE' USE16

        EXTRN   pGetWindow:NEAR
        EXTRN   pSetWindow:NEAR
        EXTRN   pGetPage:NEAR
        EXTRN   pSetPage:NEAR
        EXTRN   pGetSocket:NEAR
        EXTRN   pSetSocket:NEAR
        EXTRN   pGetStatus:NEAR
        EXTRN   pResetSocket:NEAR

        EXTRN   rGetWindow:NEAR
        EXTRN   rSetWindow:NEAR
        EXTRN   rGetPage:NEAR
        EXTRN   rSetPage:NEAR
        EXTRN   rGetSocket:NEAR
        EXTRN   rSetSocket:NEAR
        EXTRN   rGetStatus:NEAR

_TEXT        ENDS
```

292173-12

```
COMMENT
/*************************************************************************
       Customization Definitions
*************************************************************************/

DISPATCH_ENTRY      STRUC               ; Flash Entry dispatch table
  pFunction    DW     ?                  ; Pointer to function
DISPATCH_ENTRY      ENDS

; ----- Equates
NO_OF_PAGES             EQU     1
NO_OF_EDCS              EQU     0       ; no support for EDCs

BOOT_PCIC               EQU     0
BOOT_RFA                EQU     1
BOOT_RFA_ONLY           EQU     2
BOOT_OTHER              EQU     3

_DATA        SEGMENT PARA PUBLIC 'DATA' USE16
; -------------
          PUBLIC NoOfAdapters
          PUBLIC NoOfWindows
NoOfAdapters    DB  ?                   ; No. of installed adapters
NoOfWindows     DB  ?                   ; No. of installed windows
NoOfSockets     DB  ?                   ; No. of installed sockets

          PUBLIC RfaSocketNo
          PUBLIC PcicSocketNo
RfaSocketNo     DB  0                   ; Socket number assigned to RFA
PcicSocketNo    DB  0                   ; Socket number assigned to PCIC

BootDevice      DB  ?                   ; System startup boot device
                                        ; Use BOOT_XXX equates only
          PUBLIC MemWndCaps
          PUBLIC Fastest
WindowCharsTable LABEL NEAR
MemWndCaps    DW      0000000011001111b   ; Capabilities (see SS document-
InquireWindow)
MinWinBase    DW      ?                 ; Minimum base address (4K blocks)
MaxWinBase    DW      ?                 ; Maximum base address (4K blocks)
MinWinSize    DW      4                   ; 16K, Minimum window size (4K
blocks)
MaxWinSize    DW      4                   ; 16K, Maximum window size (4K
blocks)
ReqGran       DW      1                 ; 4K, Window size granularity
ReqBase       DW      4                 ; 16K, Base address alignment
CardOffAlign  DW      1                 ; Card offset alignment
Slowest       DB      4                 ; 100ns, Slowest access speed
Fastest       DB      4                 ; 100ns, Fastest access speed
WindowTableLength = $ - WindowCharsTable

RoutineAddress  DW  ?

_DATA        ENDS
```

292173-13

```
_TEXT        SEGMENT BYTE PUBLIC 'CODE' USE16

;  -----          Flash  Entry  point  dispatch  table,  contains  pointer  to
function
;              and whether media check needs to be performed.

SSDispatch   DISPATCH_ENTRY      <GetAdapterCount>      ; 80
             DISPATCH_ENTRY      <NotImplemented>       ; 81
             DISPATCH_ENTRY      <NotImplemented>       ; 82
             DISPATCH_ENTRY      <GetSSInfo>            ; 83
             DISPATCH_ENTRY      <InquireAdapter>       ; 84
             DISPATCH_ENTRY      <GetAdapter>           ; 85
             DISPATCH_ENTRY      <SetAdapter>           ; 86
             DISPATCH_ENTRY      <InquireWindow>        ; 87
             DISPATCH_ENTRY      <GetWindow>            ; 88
             DISPATCH_ENTRY      <SetWindow>            ; 89
             DISPATCH_ENTRY      <GetPage>              ; 8a
             DISPATCH_ENTRY      <SetPage>              ; 8b
             DISPATCH_ENTRY      <InquireSocket>        ; 8c
             DISPATCH_ENTRY      <GetSocket>            ; 8d
             DISPATCH_ENTRY      <SetSocket>            ; 8e
             DISPATCH_ENTRY      <GetStatus>            ; 8f
             DISPATCH_ENTRY      <ResetSocket>          ; 90
             DISPATCH_ENTRY      <NotImplemented>       ; 91
             DISPATCH_ENTRY      <NotImplemented>       ; 92
             DISPATCH_ENTRY      <NotImplemented>       ; 93
             DISPATCH_ENTRY      <NotImplemented>       ; 94
             DISPATCH_ENTRY      <NotImplemented>       ; 95
             DISPATCH_ENTRY      <NotImplemented>       ; 96
             DISPATCH_ENTRY      <NotImplemented>       ; 97
             DISPATCH_ENTRY      <NotImplemented>       ; 98
             DISPATCH_ENTRY      <NotImplemented>       ; 99
             DISPATCH_ENTRY      <NotImplemented>       ; 9a
             DISPATCH_ENTRY      <NotImplemented>       ; 9b
             DISPATCH_ENTRY      <NotImplemented>       ; 9c
             DISPATCH_ENTRY      <GetVendorInfo>        ; 9d
             DISPATCH_ENTRY      <NotImplemented>       ; 9e
             DISPATCH_ENTRY      <GetSetPriorHandler>   ; 9f
             DISPATCH_ENTRY      <GetSetSSAddr>         ; a0

NUM_Functions   =      ($ - OFFSET SSDispatch) / TYPE DISPATCH_ENTRY

Implementor  DB     'HEB_SS  '

SocketCharsTable  LABEL  NEAR
             DW     1                   ; Interface-type support (memory)
             DD     0                   ; IRQ levels inverting IREQ line
              DD      0                     ; IRQ levels not inverting IREQ
line

SocketTableLength = $ - SocketCharsTable
```

292173-14

```
AdapterCharsTable LABEL NEAR
            DW      0                   ; Capabilities (individual)
            DW      0                   ; Active high IRQ levels
             DW       08h                     ; - required by SysSoft Card
Services
            DD      0                   ; Active low IRQ levels

PowerManagementTable LABEL NEAR
            DW      0                    ; No entries - assume required pwr
exists

AdapterTableLength      = $ - AdapterCharsTable


VendorInfo      LABEL NEAR
            DB      'Intel ULP Eval Board SS', 0

VendorInfoLength        = $ - VendorInfo

            PAGE
COMMENT
/*********************************************************************

       Procedure:   GetAdapterCount

       Entry:       No significant registers

       Exit:        [AL] = Number of adapters supported
                    [CX] = 'SS'  (Socket Services id)

*********************************************************************/

GetAdapterCount PROC   NEAR
            ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

            MOV     AL, NoOfAdapters    ; One adapter
            MOV     CX, 'SS'

            CLC
            RET

GetAdapterCount ENDP
```

292173-15

```
            PAGE
COMMENT
/*************************************************************************

        Procedure:   GetSSInfo

        Entry:       [AL] = Adapter

        Exit:        [AX] = 0 (Version 1.0 compatibility)
                     [BX] = Socket Services version number
                     [CH] = Number of adapters supported by handler
                     [CL] = First adapter supported by handler

*************************************************************************/

GetSSInfo    PROC NEAR
             ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

             MOV    BX, PCMCIA_COMPLIANCE    ; PCMCIA SS Release
                MOV     CH, 1                  ; This SS supports only One
adapter
             MOV    CL, NoOfAdapters    ; first adapter number
             DEC    CL                  ; = total system adapters - 1

             XOR    AX, AX              ; Return success (resets [CF])
             RET

GetSSInfo    ENDP
```

292173-16

```
            PAGE
COMMENT
/*********************************************************************

       Procedure:   InquireAdapter

       Entry:       [AL] = Adapter
                         [ES]:[DI]  =  Pointer  to  buffer  for  adapter
characteristics
                           and power management tables

       Exit:        [BH] = Number of windows
                    [BL] = Number of sockets
                    [CX] = Number of EDCs
                    [ES]:[DI] = Unchanged

*********************************************************************/

InquireAdapter PROC   NEAR
             ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

             MOV    CX, AdapterTableLength
             MOV    ES:[DI + 2], CX             ; return wDataLength
             CMP    CX, ES:[DI]                 ; do we need to truncate?
             JB     IA2
              MOV    CX, ES:[DI]                   ; yes, truncate returned
data
IA2:
             PUSH   SI
             PUSH   DI
             MOV    SI, OFFSET DGROUP:AdapterCharsTable
             ADD    DI, 4
             REP MOVSB                  ; copy AdapterCharsTable to ES:DI
             POP    DI
             POP    SI

             MOV    BH, NoOfWindows
             MOV    BL, NoOfSockets
             MOV    CX, NO_OF_EDCS

             XOR    AH, AH              ; Return success (resets [CF])
             RET

InquireAdapter ENDP
```

292173-17

```
            PAGE
COMMENT
/************************************************************************

       Procedure:   GetAdapter

       Entry:       [AL] = Adapter

       Exit:        [DH] = Adapter State:
                        Bit 0   Reduced power consumption
                        Bit 1   State information preserved
                    [DI] = IRQ level used for status change interrupts

************************************************************************/

GetAdapter   PROC   NEAR
             ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

             XOR    DH, DH
             XOR    DI, DI

             XOR    AH, AH              ; Return success (resets [CF])
             RET

GetAdapter   ENDP


             PAGE
COMMENT
/************************************************************************

       Procedure:   SetAdapter

       Entry:       [AL] = Adapter
                    [DH] = Adapter State:
                        Bit 0   Reduced power consumption
                        Bit 1   State information preserved
                    [DI] = IRQ level used for status change interrupts

       Exit:        All registers preserved

************************************************************************/

SetAdapter   PROC   NEAR
             ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

             XOR    AH, AH              ; Return success (resets [CF])
             RET

SetAdapter   ENDP
```

292173-18

```
            PAGE
COMMENT
~************************************************************************

       Procedure:   InquireWindow

       Entry:      [AL] = Adapter
                   [BH] = Window
                   [ES]:[DI] = Pointer to buffer for window characteristics

       Exit:         [BL] = Capabilities:
                         Bit 0   Common memory
                         Bit 1   Attribute memory
                         Bit 2   I/O space
                         Bit 7   Wait supported
                       [CX] = Bit-map of assignable sockets
                       [ES]:[DI] = buffer filled with window characteristics

************************************************************************~

InquireWindow PROC   NEAR
              ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

              CMP     BH, NoOfWindows
              JB      IW2
              MOV     AH, BadWindow
              STC
              RET
IW2:
              CMP     BH, PCIC_WINDOW
              JNE     IW3

              ; PCIC Window Characteristics (Window 1, Socket ?)
              MOV     MinWinBase, pMinWinBase
              MOV     MaxWinBase, pMaxWinBase

              MOV     BL, 3               ; Common & Attribute memory
              XOR     CX, CX
              MOV     CL, PcicSocketNo    ; window available for this socket
              INC     CX                  ; CX = PcicSocketNo + 1
              JMP     IW4
IW3:
              ; RFA Window Characteristics (Window 0, Socket ?)
              MOV     MinWinBase, rMinWinBase
              MOV     MaxWinBase, rMaxWinBase

              MOV     BL, 1               ; Common memory only
              XOR     CX, CX
              MOV     CL, RfaSocketNo     ; window available for this socket
              INC     CX                  ; CX = RfaSocketNo + 1
IW4:
              PUSH    CX
              MOV     CX, WindowTableLength
              MOV     ES:[DI + 2], CX
              CMP     CX, ES:[DI]
              JB      IW5
              MOV     CX, ES:[DI]
```

292173-19

```
IW5:
                PUSH    SI
                PUSH    DI
                MOV     SI, OFFSET DGROUP:WindowCharsTable
                ADD     DI, 4
                REP MOVSB
                POP     DI
                POP     SI
                POP     CX

                XOR     AH, AH              ; Return success (resets [CF])
                RET

InquireWindow ENDP


                PAGE
COMMENT
~*********************************************************************

        Procedure:    GetWindow

        Entry:        [AL] = Adapter
                      [BH] = Window

        Exit:         [BL] = Socket
                      [CX] = Window size
                      [DH] = Window state:
                          Bit 0   Memory or I/O
                          Bit 1   Disabled or enabled
                          Bit 2   8-bit path or 16-bit path
                          Bit 3   Subdivided into pages
                      [DL] = Access speed
                      [DI] = Window base address

*********************************************************************~

GetWindow     PROC    NEAR
              ASSUME          CS:_TEXT, DS:DGROUP, ES:NOTHING

              CMP     BH, NoOfWindows
              JB      GW2
              MOV     AH, BadWindow
              STC
              RET
GW2:
              CMP     BH, PCIC_WINDOW
              JNE     GW3

              CALL    pGetWindow
              JMP     GW4
GW3:
              CALL    rGetWindow
GW4:
              RET

GetWindow     ENDP
```

292173-20

```
            PAGE
COMMENT
~*************************************************************************

       Procedure:   SetWindow

       Entry:       [AL] = Adapter
                    [BH] = Window
                    [BL] = Socket
                    [CX] = Window size
                    [DH] = Window state:
                        Bit 0   Memory or I/O
                        Bit 1   Disabled or enabled
                        Bit 2   8-bit path or 16-bit path
                        Bit 3   Subdivided into pages/EISA I/O mapping
                        Bit 4   EISA common I/O accesses
                    [DL] = Access speed
                    [DI] = Window base address

       Exit:        All registers preserved

*************************************************************************~

SetWindow    PROC   NEAR
             ASSUME       CS:_TEXT, DS:DGROUP, ES:NOTHING

             CMP    BH, NoOfWindows
             JB     SW2
             MOV    AH, BadWindow
             STC
             RET
SW2:
             ; Check for valid socket
             CMP     BL, NoOfSockets
             JB      SW3
             MOV     AH, BadSocket
             STC
             RET
SW3:
             CMP    BH, PCIC_WINDOW
             JNE    SW4

             CALL   pSetWindow
             JMP    SW5
SW4:
             CALL   rSetWindow
SW5:
             RET

SetWindow    ENDP
```

292173-21

```
                PAGE
COMMENT
~*************************************************************************

        Procedure:    GetPage

        Entry:        [AL] = Adapter
                      [BH] = Window
                      [BL] = Page

        Exit:         [DL] = Page state
                          Bit 0  Common space or attribute space
                          Bit 1  Disabled or enabled
                      [DI] = Memory card offset (4 KByte units)

*************************************************************************~

GetPage         PROC    NEAR
                ASSUME          CS:_TEXT, DS:DGROUP, ES:NOTHING

                CMP     BH, NoOfWindows
                JB      GP2
                MOV     AH, BadWindow
                STC
                RET
GP2:
                CMP     BL, NO_OF_PAGES
                JB      GP3
                MOV     AH, BadPage
                STC
                RET
GP3:
                CMP     BH, PCIC_WINDOW
                JNE     GP4

                CALL    pGetPage
                JMP     GP5
GP4:
                CALL    rGetPage
GP5:
                RET

GetPage         ENDP
```

292173-22

```
                PAGE
COMMENT
~*************************************************************************

        Procedure:    SetPage

        Entry:        [AL] = Adapter
                      [BH] = Window
                      [BL] = Page
                      [DL] = Page state
                          Bit 0  Common space or attribute space
                          Bit 1  Disabled or enabled
                          Bit 2  WP - Write protection
                      [DI] = Memory card offset (4 KByte units)

        Exit:         All registers preserved

*************************************************************************~

SetPage         PROC    NEAR
                ASSUME          CS:_TEXT, DS:DGROUP, ES:NOTHING

                CMP     BH, NoOfWindows
                JB      SP2
                MOV     AH, BadWindow
                STC
                RET
SP2:
                CMP     BL, NO_OF_PAGES
                JB      SP3
                MOV     AH, BadPage
                STC
                RET
SP3:
                CMP     BH, PCIC_WINDOW
                JNE     SP4

                CALL    pSetPage
                JMP     SP5
SP4:
                CALL    rSetPage
SP5:
                RET

SetPage         ENDP
```

292173-23

```
COMMENT
~************************************************************************

        Procedure:    InquireSocket

        Entry:        [AL] = Adapter
                      [BL] = Socket
                      [ES]:[DI] = Pointer to Socket Characteristics

        Exit:         [BH] = Status Change Interrupt Capabilities
                          Bit 0   Write Protect Change
                          Bit 1   Card Lock Change
                          Bit 2   Ejection Request
                          Bit 3   Insertion request
                          Bit 4   Battery dead change
                          Bit 5   Battery warning change
                          Bit 6   Ready change
                          Bit 7   Card detect change
                      [DH] = Status Change reporting capabilities:
                          Bit 0   Write Protect Change
                          Bit 1   Card Lock Change
                          Bit 2   Ejection Request
                          Bit 3   Insertion request
                          Bit 4   Battery dead change
                          Bit 5   Battery warning change
                          Bit 6   Ready change
                          Bit 7   Card detect change
                      [DL] = Control and indicators capabilities
                          Bit 0   Write protect status
                          Bit 1   Card lock status
                          Bit 2   Motorized card ejection
                          Bit 3   Motorized card ejection
                          Bit 4   Card lock
                          Bit 5   Battery status
                          Bit 6   Busy status
                          Bit 7   XIP status
                      [ES]:[DI] = Unchanged

************************************************************************~

InquireSocket PROC   NEAR
              ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

              CMP     BL, NoOfSockets
              JB      IS2
              MOV     AH, BadSocket
              STC
              RET
IS2:
              PUSH    CX
              MOV     CX, SocketTableLength
              MOV     ES:[DI + 2], CX
              CMP     CX, ES:[DI]
              JB      IS3
              MOV     CX, ES:[DI]
IS3:
              PUSH    SI
```

292173-24

```
                PUSH    DI
                MOV     SI, OFFSET DGROUP:SocketCharsTable
                ADD     DI, 4
                REP MOVSB
                POP     DI
                POP     SI
                POP     CX

                MOV     BH, 0
                MOV     DH, 80h     ; Report card detect (always inserted)
                MOV     DL, 0

                XOR     AH, AH              ; Return success (resets [CF])
                RET

InquireSocket ENDP
```

292173-25

```
~*********************************************************************
      Procedure:   GetSocket

      Entry:       [AL] = Adapter
                   [BL] = Socket

      Exit:        [BH] = Status change interrupt mask
                       Bit 0   Write Protect Change
                       Bit 1   Card Lock Change
                       Bit 2   Ejection Request
                       Bit 3   Insertion request
                       Bit 4   Battery dead change
                       Bit 5   Battery warning change
                       Bit 6   Ready change
                       Bit 7   Card detect change
                   [CH] = VCC level
                   [CL] = VPP1 & VPP2 levels
                   [DH] = Socket state
                       Bit 0   Write Protect Change
                       Bit 1   Card Lock Change
                       Bit 2   Ejection Request
                       Bit 3   Insertion request
                       Bit 4   Battery dead change
                       Bit 5   Battery warning change
                       Bit 6   Ready change
                       Bit 7   Card detect change
                   [DL] = Control and indicators state
                       Bit 0   Write protect status
                       Bit 1   Card lock status
                       Bit 2   Motorized card ejection
                       Bit 3   Motorized card ejection
                       Bit 4   Card lock
                       Bit 5   Battery status
                       Bit 6   Busy status
                       Bit 7   XIP status
                   [DI] = IRQ level steering/Interface type

*********************************************************************~
GetSocket    PROC   NEAR
             ASSUME       CS:_TEXT, DS:DGROUP, ES:NOTHING

             CMP    BL, NoOfSockets
             JB     GS2
             MOV    AH, BadSocket
             STC
             RET
GS2:
             CMP    BL, PcicSocketNo
             JNE    GS3

             CALL   pGetSocket
             JMP    GS4
GS3:
             CALL   rGetSocket
GS4:
             RET
GetSocket    ENDP
```

292173-26

```
~*********************************************************************
   Procedure:   SetSocket

        Entry:       [AL] = Adapter
                     [BL] = Socket
        Exit:        [BH] = Status change interrupt mask
                         Bit 0   Write Protect Change
                         Bit 1   Card Lock Change
                         Bit 2   Ejection Request
                         Bit 3   Insertion request
                         Bit 4   Battery dead change
                         Bit 5   Battery warning change
                         Bit 6   Ready change
                         Bit 7   Card detect change
                     [CH] = VCC level
                     [CL] = VPP1 & VPP2 levels
                     [DH] = Socket state
                         Bit 0   Write Protect Change
                         Bit 1   Card Lock Change
                         Bit 2   Ejection Request
                         Bit 3   Insertion request
                         Bit 4   Battery dead change
                         Bit 5   Battery warning change
                         Bit 6   Ready change
                         Bit 7   Card detect change
                     [DL] = Control and indicators state
                         Bit 0   Write protect status
                         Bit 1   Card lock status
                         Bit 2   Motorized card ejection
                         Bit 3   Motorized card ejection
                         Bit 4   Card lock
                         Bit 5   Battery status
                         Bit 6   Busy status
                         Bit 7   XIP status
                     [DI] = IRQ level steering/Interface type

         Exit:       All register preserved
*********************************************************************~
SetSocket    PROC   NEAR
             ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

             CMP    BL, NoOfSockets
             JB     SS2
             MOV    AH, BadSocket
             STC
             RET
SS2:
             CMP    BL, PcicSocketNo
             JNE    SS3

             CALL   pSetSocket
             JMP    SS4
SS3:
             CALL   rSetSocket
SS4:
             RET
SetSocket    ENDP
```

292173-27

```
~*********************************************************************

        Procedure:    GetStatus

        Entry:        [AL] = Adapter
                      [BL] = Socket

        Exit:         [BH] = Current card state:
                          Bit 0   Write protect
                          Bit 1   Card Locked
                          Bit 2   Ejection Request
                          Bit 3   Insertion Request
                          Bit 4   Battery dead
                          Bit 5   Battery low
                          Bit 6   Card ready
                          Bit 7   Card detected
                      [DH] = Socket state
                      [DL] = Control and indicator state
                      [DI] = IRQ level steering/Interface type

*********************************************************************~

GetStatus     PROC    NEAR
              ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

              CMP     BL, NoOfSockets
              JB      GC2
              MOV     AH, BadSocket
              STC
              RET
GC2:
              CMP     BL, PcicSocketNo
              JNE     GC3

              CALL    pGetStatus
              JMP     GC4
GC3:
              CALL    rGetStatus
GC4:
              RET

GetStatus     ENDP
```

292173-28

```
~*********************************************************************
   Procedure:   ResetSocket
       Entry:       [AL] = Adapter
                    [BL] = Socket

       Exit:        All registers preserved
*********************************************************************~
ResetSocket  PROC   NEAR
             ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING
             CMP    BL, NoOfSockets
             JB     RS2
             MOV    AH, BadSocket
             STC
             RET
RS2:
             CMP    BL, PcicSocketNo
             JNE    RS3
             CALL   pResetSocket
             JMP    RS4
RS3:
             XOR    AH, AH              ; Return success (resets [CF])
RS4:
             RET
ResetSocket  ENDP

~*********************************************************************
   Procedure:   GetVendorInfo
       Entry:       [AL] = Adapter
                    [BL] = Type
                    [ES]:[DI] = Vendor info buffer

       Exit:        [DX] = Vendor Release
*********************************************************************~
GetVendorInfo PROC   NEAR
             ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING
             CMP    BL, 0
             JNE    GV1
             PUSH   CX
             MOV    CX, VendorInfoLength
             MOV    ES:[DI + 2], CX
             CMP    CX, ES:[DI]
             JB     GV3
             MOV    CX, ES:[DI]
GV3:
             PUSH   SI
             PUSH   DI
             MOV    SI, OFFSET DGROUP:VendorInfo
             ADD    DI, 4
             REP MOVSB
             POP    DI
             POP    SI
             POP    CX
GV1:
             MOV    DX, VENDOR_VERSION  ; Our version no.
             XOR    AH, AH              ; Return success (resets [CF])
             RET
GetVendorInfo ENDP
```

292173-29

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

                    Unimplemented Socket Services

************************************************************************/

NotImplemented  PROC NEAR
            ASSUME        CS:_TEXT, DS:NOTHING, ES:NOTHING

            MOV     AX, UnsupportedFunction
            STC

            RET
NotImplemented  ENDP


/************************************************************************
Procedure:   _SocketServices

       Purpose:     Main entry point of Socket Services

       Entry:       [AH] = Function, starting at 80h
                    Others vary by function

       Exit:        [CF] = Status
                    If [CF] set, [AX] = error
                    Others vary by function
************************************************************************/
            PUBLIC _SocketServices

_SocketServices PROC   FAR
            ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

            AND     AH, 7Fh
            CMP     AH, NUM_Functions   ; Valid request ?
            JB      SoS2                ; Yes, continue

            MOV     AX, UnsupportedFunction ; No, invalid command
            STC                         ; Set carry bit to indicate error
            RET
SoS2:
            PUSH    AX
            PUSH    BX
            XOR     AL, AL
            XCHG    AH, AL
            ADD     AX, AX
            MOV     BX, AX
            ADD     BX, OFFSET DGROUP:SSDispatch ; Add base address of tbl
            MOV     AX, CS:[BX.pFunction]
            MOV     RoutineAddress, AX
            POP     BX
            POP     AX
            CALL    [RoutineAddress]    ; Perform requested function
                                ; AX contains error code from function
            RET
_SocketServices ENDP
_TEXT       ENDS
```

292173-30

```
_INIT        SEGMENT BYTE PUBLIC 'INIT' USE16

             ASSUME        CS:DGROUP, DS:DGROUP, ES:NOTHING

        EXTRN   PcicInit:NEAR
        EXTRN   RfaInit:NEAR

szSignonMsg  DB     CR, LF
             DB     'Intel ULP 486sx (TM) Evaluation Board PCMCIA '
              DB     sPCMCIA_COMPLIANCE, ' Socket Services Interface.', CR,
LF
             DB     '(C) Copyright 1992, M-Systems Ltd.', CR, LF
             DB     '(C) Copyright 1995, Intel Corporation, Version '
             DB     sVENDOR_VERSION, CR, LF
             DB     LF
             DB     EOS

szAbortMsg   DB     CR, LF
             DB     'Initialization aborted.', CR, LF
             DB     EOS

szPressKeyMsg DB     CR, LF
             DB     'Press any key to continue ... ', CR, LF
             DB     EOS

szLongDelay  DB     'Hard Disk Drive detected!', CR, LF
             DB     'This will cause a 45 second delay.  Please wait...'
             DB     CR, LF
             DB     CR, LF
             DB     EOS

szSelectBootDevice DB     ' Select Boot Device', CR, LF, LF
             DB     '    1) PCMCIA Slot + RFA', CR, LF
             DB     '    2) RFA + PCMCIA Slot', CR, LF
             DB     '    3) RFA only', CR, LF
             DB     CR, LF
             DB     ' Enter a number: '
             DB     EOS

szResponse1  DB     '1'
             DB     CR, LF
             DB     EOS

szResponse2  DB     '2'
             DB     CR, LF
             DB     EOS

szResponse3  DB     '3'
             DB     CR, LF
             DB     EOS

szCrLf       DB     CR, LF
             DB     EOS
```

292173-31

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

       Procedure:   _InitSocketServices

       Purpose:     Do first-time initialization

       Entry:       [AL] = Adapter no.
                    [DX] = BIOS Expansion segment (0 = not applicable)
                    [ES]:[DI] = 8-Character implementation name buffer

       Exit:        [AX] = status

       Locate Adapter and establish Adapter Data
       Install Socket Services

*************************************************************************/

             PUBLIC _InitSocketServices
_InitSocketServices PROC   NEAR
             ASSUME        CS:DGROUP, DS:DGROUP, ES:NOTHING

             PUSH   DX

             INC    AL
             MOV    NoOfAdapters, AL

             MOV    DX, OFFSET DGROUP:szSignonMsg
             CALL   PrintMsg

             PROTO1_BUGFIX

             CALL   _GetBootDevice

             CMP    AL, BOOT_PCIC
             JNE    Init1

             MOV    PcicSocketNo, 0
             MOV    RfaSocketNo, 1
             MOV    NoOfSockets, 2
             MOV    NoOfWindows, 2
             JMP    Init2
Init1:
             MOV    RfaSocketNo, 0
             MOV    PcicSocketNo, 1
             MOV    NoOfSockets, 2
             MOV    NoOfWindows, 2

             CMP    AL, BOOT_RFA_ONLY
             JNE    Init2
             MOV    PcicSocketNo, -1
             MOV    NoOfSockets, 1
             MOV    NoOfWindows, 1
             JMP    Init3
Init2:
             ; (3) Initialize Socket Services
             CALL   PcicInit
             JNC    Init3
```

292173-32

```
                MOV     DX, OFFSET DGROUP:szAbortMsg
                CALL    PrintMsg

                MOV     DX, OFFSET DGROUP:szPressKeyMsg
                CALL    PressKey            ; Wait for acknowledgement

                POP     DX
                RET
Init3:
                CALL    RfaInit
                POP     DX

                ; Copy Implementor string into ES:DI buffer
                PUSH    SI
                PUSH    CX

                MOV     CX, 8
                MOV     SI, OFFSET DGROUP:Implementor
                REP MOVSB

                POP     CX
                POP     SI

                XOR     AX, AX              ; Return success (resets [CF])
                RET

_InitSocketServices ENDP

COMMENT
/************************************************************************

        Procedure:    _GetBootDevice

        Purpose:      Determine which device is primary boot device.
                      (i.e. IDE hard drive, RFA, or PCMCIA slot

        Exit:         [AL] = BOOT_xxx (boot device)

************************************************************************/

                PUBLIC _GetBootDevice
_GetBootDevice PROC   NEAR
                ASSUME        CS:DGROUP, DS:DGROUP, ES:NOTHING

                ; Does IDE Hard drive exist?
                PUSH    ES

                MOV     AX, ABSO            ; Point to ABSO segment
                MOV     ES, AX
                MOV     AL, ES:HardDrive

                POP     ES

                CMP     AL, 0
                JE      GBD2
                ; Yes!
```

292173-33

```
                MOV     DX, OFFSET DGROUP:szLongDelay
                CALL    PrintMsg
                MOV     AL, BOOT_OTHER
                JMP     GBD3                                    ; leave
GBD2:
                ; No! Hard drive does not exist, therefore
                ; prompt for boot device (RFA or PCMCIA Slot)
                MOV     DX, OFFSET DGROUP:szSelectBootDevice
                CALL    PrintMsg
GBD6:
                XOR     AH, AH
                INT     16h             ; Read character from keyboard buffer

                OR      AL, AL
                JZ      GBD5
                MOV     DX, OFFSET DGROUP:szResponse1       ; Echo character
                CMP     AL, 31h
                JE      GBD7
                MOV     DX, OFFSET DGROUP:szResponse2
                CMP     AL, 32h
                JE      GBD7
                MOV     DX, OFFSET DGROUP:szResponse3
GBD7:
                CALL    PrintMsg
                MOV     DX, OFFSET DGROUP:szCrLf
                CALL    PrintMsg

                AND     AX, 000Fh
                DEC     AL
                CMP     AL, BOOT_OTHER
                JAE     GBD6
                JMP     GBD3
GBD5:
          ; Send Function key back to keyboard buffer for BIOS to process
                CMP     AH, 3Ch             ; F2 key?
                JNE     GBD6
                MOV     AH, 05
                XOR     CL, CL
                MOV     CH, 3Ch
                INT     16h
GBD3:
                RET

_GetBootDevice ENDP

_INIT           ENDS

                END
```

292173-34

```
; $Log: RFA.ASM $
; Revision 1.1  1995/08/30 09:25:06
; Initial revision
;
;  ------------------------------------------------------
;  July/Aug. 1995          Mark Gianopulos - Intel Corp.
;  Created for 486sx(TM) ULP RFA & PCMCIA Socket
;  ------------------------------------------------------
;

        PAGE   78, 132
        TITLE  RFA.ASM
COMMENT ~*****************************************************************************

       Title:       RFA.ASM

       Purpose:     RFA Specific functions for ULP Socket Services

       Author:      Mark Gianopulos
                    Intel Corporation

       Copyright (C) by Intel 1995.  All rights reserved.

*****************************************************************************~

.486

INCLUDE      SSDEFS.INC         ; global definitions
INCLUDE      RFA.INC

             ASSUME  CS:NOTHING, DS:NOTHING, ES:NOTHING

DGROUP       GROUP  _TEXT, _DATA, _INIT

_DATA        SEGMENT PARA PUBLIC 'DATA' USE16

EXTRN        RfaSocketNo:BYTE           ; ss.asm
EXTRN        Fastest:BYTE               ; ss.asm

MemCardOffset   DW  ?                   ; Current card offset (can't read
                                        ; page register)
 DATA        ENDS
```

292173-35

```
_TEXT        SEGMENT BYTE PUBLIC 'CODE' USE16

; ------------------------------------
; Proc:  SetWindowBaseAddr
; Desc:  Sets the RFA system window
;        to the value specified in BX
; I: BX - contains C8, D0, D4, D8, or DC
;
; Trashes: AX, BX, DX
; ------------------------------------
SetWindowBaseAddr    PROC   NEAR

             SHR    BX, 2               ; D0 ==> 34 (bits14-bits21)
             MOV    BH, 80h             ; enable GPCS2#, (bits22-bits23)

             CLI                        ; Must not be interrupted
             IODELAY
             MOV    DX, REDWOOD_INDEX   ; Set base address
             MOV    AX, WINDOW_BASE_REG
             OUT    DX, AX
             IODELAY
             MOV    DX, REDWOOD_DATA
             MOV    AX, BX
             OUT    DX, AX

             IODELAY
             MOV    DX, REDWOOD_INDEX   ; Enable base address
             MOV    AX, WINDOW_BASE_EN_REG
             OUT    DX, AX
             IODELAY
             MOV    DX, REDWOOD_DATA
;             MOV    AX, 3FFCh           ; assumes 64K windows only!
;             MOV    AX, 3FFEh           ; assumes 32K windows only!
             MOV    AX, 3FFFh           ; assumes 16K windows only!
             OUT    DX, AX
             STI

             RET

SetWindowBaseAddr   ENDP
```

292173-36

```
; ------------------------------------
; Proc:  GetWindowBaseAddr
; Desc:  Returns the RFA system window
;        address in 4K units. (ie: C8, D0, D4, D8, or DC)
; O: AX - returns address in AX register
;
; Trashes: AX, DX
; ------------------------------------
GetWindowBaseAddr    PROC   NEAR
             ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

             CLI                        ; Must not be interrupted
             IODELAY
             MOV    DX, REDWOOD_INDEX   ; Get base address
             MOV    AX, WINDOW_BASE_REG
             OUT    DX, AX
             IODELAY
             IN     AX, REDWOOD_DATA
             STI

             AND    AX, 03FFh           ; mask out top 6 bits
             SHL    AX, 2               ; 34 ==> D0

             RET
GetWindowBaseAddr   ENDP

COMMENT ~*************************************************************************
       Procedure:   rGetWindow

       Entry:       [AL] = Adapter
                    [BH] = Window

       Exit:        [BL] = Socket
                    [CX] = Window size
                    [DH] = Window state:
                        Bit 0   Memory or I/O
                        Bit 1   Disabled or enabled
                        Bit 2   8-bit path or 16-bit path
                        Bit 3   Subdivided into pages
                    [DL] = Access speed
                    [DI] = Window base address
*************************************************************************~
             PUBLIC rGetWindow
rGetWindow   PROC   NEAR
             ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

             ;[BL] = Socket
             MOV    BL, RfaSocketNo

             ;[DI] = Window base address
             call   GetWindowBaseAddr   ; returned in AX as 4K unit
             MOV    DI, AX

             ;[CX] = Window size
             MOV    DX, WINDOW_SIZE_REG
             IN     AL, DX
             PUSH   AX                          ; save register contents
```

292173-37

```
                NOT     AL                      ; complement
                AND     AL, 03h                 ; mask off all but bits 0 and 1
                INC     AL                      ; add 1
                MOV     CL, AL                  ; save in cl (shift factor)
                MOV     AX, 01                  ; bit to be shifted (2,4,8,16)
                SHL     AL, CL
                MOV     CX, AX

                ;[DH] = Window state
                POP     AX                              ; restore register
                MOV     DH, 0                           ; 8bit + disabled + mem
                TEST    AL, 04h
                JZ      rGW6
                OR      DH, 10b                         ; 8bit + enabled + mem
rGW6:
                ;[DL] = Access speed
                MOV     DL, Fastest

                XOR     AH, AH                  ; Return success (resets [CF])
                RET
rGetWindow      ENDP


COMMENT ~************************************************************************
   Procedure:   rSetWindow

        Entry:          [AL] = Adapter
                        [BH] = Window
                        [BL] = Socket
                        [CX] = Window size
                        [DH] = Window state:
                            Bit 0   Memory or I/O
                            Bit 1   Disabled or enabled
                            Bit 2   8-bit path or 16-bit path
                            Bit 3   Subdivided into pages/EISA I/O mapping
                            Bit 4   EISA common I/O accesses
                        [DL] = Access speed
                        [DI] = Window base address

        Exit:           All registers preserved
************************************************************************~
                PUBLIC rSetWindow
rSetWindow      PROC    NEAR
                ASSUME          CS:_TEXT, DS:DGROUP, ES:NOTHING

                CMP     BL, RfaSocketNo
                JE      rSW1
                MOV     AH, BadSocket
                STC
                RET
rSW1:
                ; Check for valid window size
                MOV     AH, WINDOW_SIZE_08K
                CMP     CX, 2                           ;   8K
                JE      rSW4
                MOV     AH, WINDOW_SIZE_16K
```

292173-38

```
                CMP     CX, 4                       ;  16K
                JE      rSW4
                MOV     AH, WINDOW_SIZE_32K
                CMP     CX, 8                       ;  32k
                JE      rSW4
                MOV     AH, WINDOW_SIZE_64K
                CMP     CX, 10h                     ;  64K
                JE      rSW4
                CMP     CX, 0                       ;  64K - MAX(WORD)
                JE      rSW4
                MOV     AH, BadSize
                STC
                RET
rSW4:
                ; Set new window size
                PUSH    DX
                MOV     DX, WINDOW_SIZE_REG
                IN      AL, DX
                AND     AL, 0FCh            ; Clear bits 0,1
                OR      AL, AH              ; assign bits 0,1
                OUT     DX, AL
                POP     DX

                ; Check for valid Window Base address
                CMP     DI, rMinWinBase
                JB      rSW5
                CMP     DI, rMaxWinBase
                JBE     rSW6
rSW5:
                MOV     AH, BadOffset
                STC
                RET
rSW6:
                ; Set new window base address
                PUSH    DX
                MOV     BX, DI
                call    SetWindowBaseAddr
                POP     DX

                ; Check for valid window state
                CMP     DH, 0               ; Memory, disabled, 8bit, 1page
                JE      rSW7
                CMP     DH, 2               ; Memory, enabled,  8bit, 1page
                JE      rSW8
                MOV     AH, BadAttribute
                STC
                RET

rSW7:            ; disable window
                PUSH    DX
                MOV     DX, WINDOW_ENABLE_REG
                IN      AL, DX
                AND     AL, 0FBh            ; clear bit 3
                OUT     DX, AL
                POP     DX
                JMP     rSW9
```

292173-39

```
rSW8:          ; enable window
               PUSH    DX
               MOV     DX, WINDOW_ENABLE_REG
               IN      AL, DX
               OR      AL, 04h              ; set bit 3
               OUT     DX, AL
               POP     DX
rSW9:
               ; Check for valid access speed
               CMP     DL, 04               ; must be 100ns speed
               JE      rSW10
               MOV     AH, BadSpeed
               STC
               RET
rSW10:
               XOR     AH, AH               ; Return success (resets [CF])
               RET

rSetWindow     ENDP
```

292173-40

```
COMMENT ~************************************************************************
Procedure:   rGetPage

       Entry:       [AL] = Adapter
                    [BH] = Window
                    [BL] = Page

       Exit:        [DL] = Page state
                        Bit 0  Common space or attribute space
                        Bit 1  Disabled or enabled
                    [DI] = Memory card offset (4 KByte units)
************************************************************************************~
             PUBLIC rGetPage
rGetPage     PROC   NEAR
             ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

             ;[DL] = Page state
             MOV    DX, WINDOW_ENABLE_REG
             IN     AL, DX
             MOV    DL, 00h             ; !wp + disabled + common
             TEST   AL, 04h
             JZ     rGP4
             MOV    DL, 02h             ; set enabled bit
rGP4:
             ;[DI] = Memory card offset (4 KByte units)
             MOV    DI, MemCardOffset

             XOR    AH, AH              ; Return success (resets [CF])
             RET
rGetPage     ENDP

COMMENT ~************************************************************************
   Procedure:   rSetPage

       Entry:       [AL] = Adapter
                    [BH] = Window
                    [BL] = Page
                    [DL] = Page state
                        Bit 0  Common space or attribute space
                        Bit 1  Disabled or enabled
                        Bit 2  WP - Write protection
                    [DI] = Memory card offset (4 KByte units)

       Exit:        All registers preserved
************************************************************************************~
             PUBLIC rSetPage
rSetPage     PROC   NEAR
             ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

             ;[DL] = Page state
             ; Check for valid page state
             CMP    DL, 00              ; common, disabled, !WP
             JE     rSP4
             CMP    DL, 02              ; common, enabled, !WP
             JE     rSP5
             MOV    AH, BadAttribute
             STC
```

292173-41

```
                RET

rSP4:           ; disable window
                MOV     DX, WINDOW_ENABLE_REG
                IN      AL, DX
                AND     AL, 0FBh            ; clear bit 3
                OUT     DX, AL
                JMP     rSP6
rSP5:
                ; enable window
                MOV     DX, WINDOW_ENABLE_REG
                IN      AL, DX
                OR      AL, 04h             ; set bit 3
                OUT     DX, AL
rSP6:
                ;[DI] = Memory card offset (4 KByte units)
                ; 0) Save new offset in global variable
                MOV     MemCardOffset, DI

                ; 1) Determine window size and save multiplier in cl
                MOV     DX, WINDOW_SIZE_REG
                IN      AL, DX              ; a) read window size
                NOT     AL                  ; b) complement
                AND     AL, 03h             ; c) mask off all but bits 0 and 1
                INC     AL                  ; d) add 1
                MOV     CL, AL              ; e) save in cl (shift factor)
                MOV     CH, AL              ; f) save in ch (shift factor)

                ; 2) Calculate Page value, store in AL
                MOV     AX, DI
                XOR     AH, AH              ; highest value is 00FE
                SHR     AL, CL              ; c) Divide offset by WinSize/4K

                ; 3) Swap the page (AL) bits from low to high
                ;    i.e. 0000 0001 ==> 1000 0000
                ;    BL == i, AL == page, AH == fpage
                XOR     BL, BL              ; for loop counter i=0, for (i=0;;)
rSP7:
                MOV     CL, BL
                MOV     DL, 1
                SHL     DL, CL              ; (1 << i)
                TEST    AL, DL              ; if ((page & (1 << i)) != 0)
                JZ      rSP8
                MOV     DL, 7
                SUB     DL, CL              ; (7 - i)
                MOV     CL, DL
                MOV     DL, 1
                SHL     DL, CL              ; 1 << (7 - i)
                OR      AH, DL              ; fpage |= 1 << (7 - i);
rSP8:
                INC     BL                  ; for (;;i++)
                CMP     BL, 8               ; for (;i<8;)
                JB      rSP7

                ; 4) Output the right page number
                MOV     CL, CH              ; restore shift factor from 1.f
                SHR     AH, CL              ; fpage >> i
```

292173-42

```
                MOV     AL, AH

                MOV     DX, WINDOW_PAGE_REG
                OUT     DX, AL

                XOR     AH, AH              ; Return success (resets [CF])
                RET
rSetPage        ENDP

COMMENT ~*************************************************************************
        Procedure:    rGetSocket

        Entry:        [AL] = Adapter
                      [BL] = Socket

        Exit:         [BH] = Status change interrupt mask
                          Bit 0   Write Protect Change
                          Bit 1   Card Lock Change
                          Bit 2   Ejection Request
                          Bit 3   Insertion request
                          Bit 4   Battery dead change
                          Bit 5   Battery warning change
                          Bit 6   Ready change
                          Bit 7   Card detect change
                      [CH] = VCC level
                      [CL] = VPP1 & VPP2 levels
                      [DH] = Socket state
                          Bit 0   Write Protect Change
                          Bit 1   Card Lock Change
                          Bit 2   Ejection Request
                          Bit 3   Insertion request
                          Bit 4   Battery dead change
                          Bit 5   Battery warning change
                          Bit 6   Ready change
                          Bit 7   Card detect change
                      [DL] = Control and indicators state
                          Bit 0   Write protect status
                          Bit 1   Card lock status
                          Bit 2   Motorized card ejection
                          Bit 3   Motorized card ejection
                          Bit 4   Card lock
                          Bit 5   Battery status
                          Bit 6   Busy status
                          Bit 7   XIP status
                      [DI] = IRQ level steering/Interface type

*************************************************************************~
                PUBLIC rGetSocket
rGetSocket      PROC    NEAR
                ASSUME          CS:_TEXT, DS:DGROUP, ES:NOTHING
              ; [BH] = Status change interrupt mask
                MOV     BH, 0               ; none
              ; [CH] = VCC level
                MOV     CH, 0               ; no pwrentry
              ; [CL] = VPP1 & VPP2 levels
                MOV     CL, 0               ; no pwrentry
              ; [DH] = Socket state
```

292173-43

```
            MOV    DH, 0              ; no states to report
         ; [DL] = Control and indicators state
            MOV    DL, 0              ; none
         ; [DI] = LowByte=IRQ level steering/HiByte=Interface type
            MOV    DI, 0100h          ; No IRQ's, memory only
            XOR    AH, AH             ; Return success (resets [CF])
            RET
rGetSocket  ENDP

COMMENT ~************************************************************************
   Procedure:   rSetSocket

       Entry:       [AL] = Adapter
                    [BL] = Socket
       Exit:        [BH] = Status change interrupt mask
                        Bit 0   Write Protect Change
                        Bit 1   Card Lock Change
                        Bit 2   Ejection Request
                        Bit 3   Insertion request
                        Bit 4   Battery dead change
                        Bit 5   Battery warning change
                        Bit 6   Ready change
                        Bit 7   Card detect change
                    [CH] = VCC level
                    [CL] = VPP1 & VPP2 levels
                    [DH] = Socket state
                        Bit 0   Write Protect Change
                        Bit 1   Card Lock Change
                        Bit 2   Ejection Request
                        Bit 3   Insertion request
                        Bit 4   Battery dead change
                        Bit 5   Battery warning change
                        Bit 6   Ready change
                        Bit 7   Card detect change
                    [DL] = Control and indicators state
                        Bit 0   Write protect status
                        Bit 1   Card lock status
                        Bit 2   Motorized card ejection
                        Bit 3   Motorized card ejection
                        Bit 4   Card lock
                        Bit 5   Battery status
                        Bit 6   Busy status
                        Bit 7   XIP status
                    [DI] = IRQ level steering/Interface type

        Exit:       All register preserved
************************************************************************~

            PUBLIC rSetSocket
rSetSocket  PROC   NEAR
            ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

            XOR    AH, AH             ; Return success (resets [CF])
            RET

rSetSocket  ENDP
```

292173-44

```
COMMENT ~************************************************************************

        Procedure:    rGetStatus

        Entry:        [AL] = Adapter
                      [BL] = Socket

        Exit:         [BH] = Current card state:
                          Bit 0   Write protect
                          Bit 1   Card Locked
                          Bit 2   Ejection Request
                          Bit 3   Insertion Request
                          Bit 4   Battery dead
                          Bit 5   Battery low
                          Bit 6   Card ready
                          Bit 7   Card detected
                      [DH] = Socket state
                      [DL] = Control and indicator state
                      [DI] = IRQ level steering/Interface type

************************************************************************~

              PUBLIC rGetStatus
rGetStatus     PROC   NEAR
               ASSUME        CS:_TEXT, DS:DGROUP, ES:NOTHING

               ; [BH] = Status change interrupt mask
               MOV    BH, 80h             ; card inserted

               ; [DH] = Socket state
               MOV    DH, 0               ; no states to report

               ; [DL] = Control and indicators state
               MOV    DL, 0               ; none

               ; [DI] = LowByte=IRQ level steering/HiByte=Interface type
               MOV    DI, 0100h           ; No IRQ's, memory only

               XOR    AH, AH              ; Return success (resets [CF])
               RET

rGetStatus     ENDP

_TEXT          ENDS

_INIT          SEGMENT BYTE PUBLIC 'INIT' USE16

               ASSUME        CS:DGROUP, DS:DGROUP, ES:NOTHING
```

292173-45

```
COMMENT /************************************************************************
        Procedure:   RfaInit

        Purpose:     Do first-time initialization for RFA

        Entry:       nothing

        Exit:        [AX] = status

        Initialize chip set and configure window for RFA
************************************************************************/
              PUBLIC RfaInit
RfaInit PROC   NEAR
              ASSUME        CS:DGROUP, DS:DGROUP, ES:NOTHING

              PUSH   BX
              PUSH   DX

;             0) Setup REDWOOD Registers

              CLI                          ; Must not be interrupted
              IODELAY
              MOV    DX, REDWOOD_INDEX
              MOV    AX, REDWOOD_PS_REG3 ; 112h
              OUT    DX, AX
              IODELAY
              MOV    DX, REDWOOD_DATA
              IN     AX, DX
              OR     AX, 4
              OUT    DX, AX

              IODELAY
              MOV    DX, REDWOOD_INDEX
              MOV    AX, REDWOOD_PS_REG1 ; 110h
              OUT    DX, AX
              IODELAY
              MOV    DX, REDWOOD_DATA
              IN     AX, DX
              OR     AX, 0100h
              OUT    DX, AX
              STI

;             1) Disable FlashROMdisk (RFA)
              MOV    DX, WINDOW_ENABLE_REG
              IN     AL, DX
              AND    AL, 0FBh             ; clear bit 3
              OUT    DX, AL

;             2) Configure system window base address
              MOV    BX, rMinWinBase
              call   SetWindowBaseAddr

;             3) Configure system window size
              MOV    DX, WINDOW_SIZE_REG
              IN     AL, DX
              AND    AL, 0FCh             ; Clear bits 0,1
              OR     AL, WINDOW_SIZE_16K ; assign bits 0,1
```

292173-46

```
                OUT     DX, AL

;               4) Configure page address (card address 0)
                MOV     MemCardOffset, 0
                MOV     AL, 0
                MOV     DX, WINDOW_PAGE_REG
                OUT     DX, AL

;               5) Enable FlashROMdisk (RFA)
                MOV     DX, WINDOW_ENABLE_REG
                IN      AL, DX
                OR      AL, 04h             ; set bit 3
                OUT     DX, AL

                POP     DX
                POP     BX

                XOR     AX, AX              ; Return success (resets [CF])
                RET

RfaInit ENDP

_INIT           ENDS

                END
```

292173-47