intel®

AP-608

APPLICATION NOTE

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

CHARLES A. ANYIMI
TECHNICAL MARKETING ENGINEER

February 1995

Order Number: 292161-001

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

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

or call 1-800-879-4683

CG-041493

1.0 INTRODUCTION

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

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

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

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

2.0 BENEFITS OF A PnP BOOT BLOCK FLASH BIOS

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

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

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

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

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

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

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

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

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

2.1 The Old BIOS Paradigm

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

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

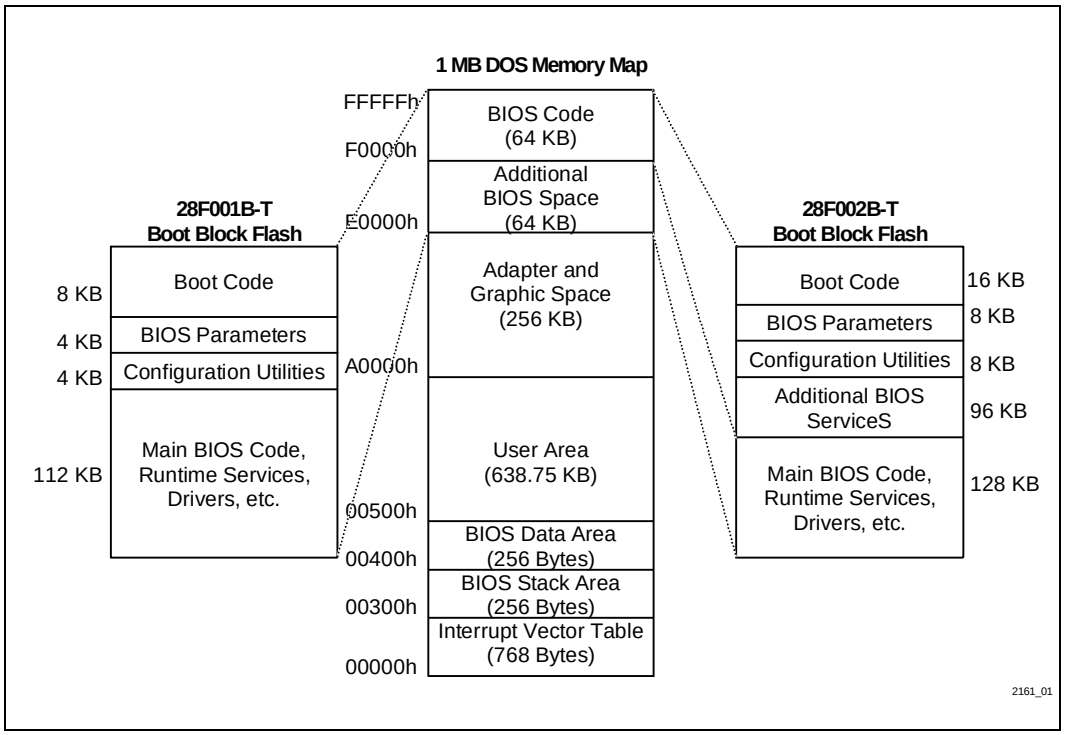


Figure 1. Possible BIOS Segmentations Using Flash Components (1-Mbit or 2-Mbit)

2.2 Beyond the 128 KB Limit

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

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

Today, PnP BIOS requirements may extend beyond 128 KB. A standard system BIOS consumes roughly 64 KB; PnP support is another 12 KB; automatic power management support adds 2 to 10 KB; PCI (Peripheral Components Interconnect) extensions hoard some 20 KB of the BIOS; and on-board VGA (Video Graphics Adapter) controllers utilize an extra 30 KB to 40 KB of space. The potential size of the BIOS for such a system is almost 150 KB. Although most of this code may be compressed, certain code segments cannot; besides even compression has its limits. Since new developments for the PC show no signs of abatement, it seems clear that a means of updating as well as expanding BIOS quickly and easily has become a necessity. Plug and Play is only one of the many technologies that can be implemented better with flash. As users realize the benefits of PnP, the demand generated will drive the number of Plug and Play systems upward.

3.0 PLUG AND PLAY

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

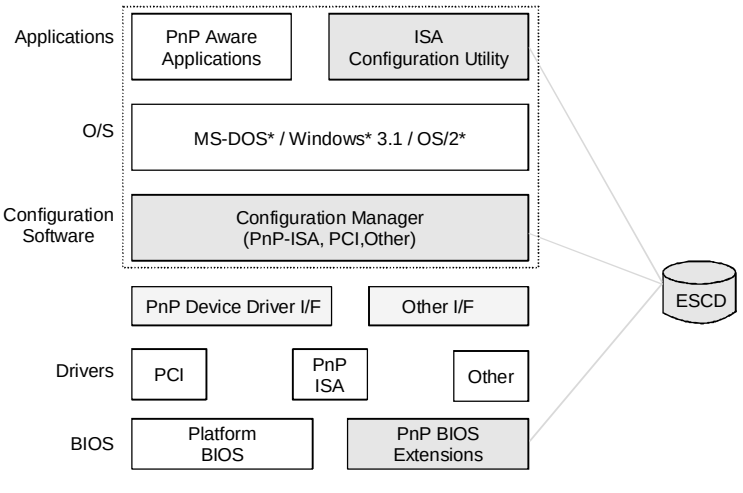
3.1 PnP Components

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

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

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

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



NOTE:
The area enclosed in the dotted-line box references modules that can be incorporated into the PnP operating system, e.g., Windows 95, OS/2.

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

The following components are operating system dependent:

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

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

3.2 PnP Functionality

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

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

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

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

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

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

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

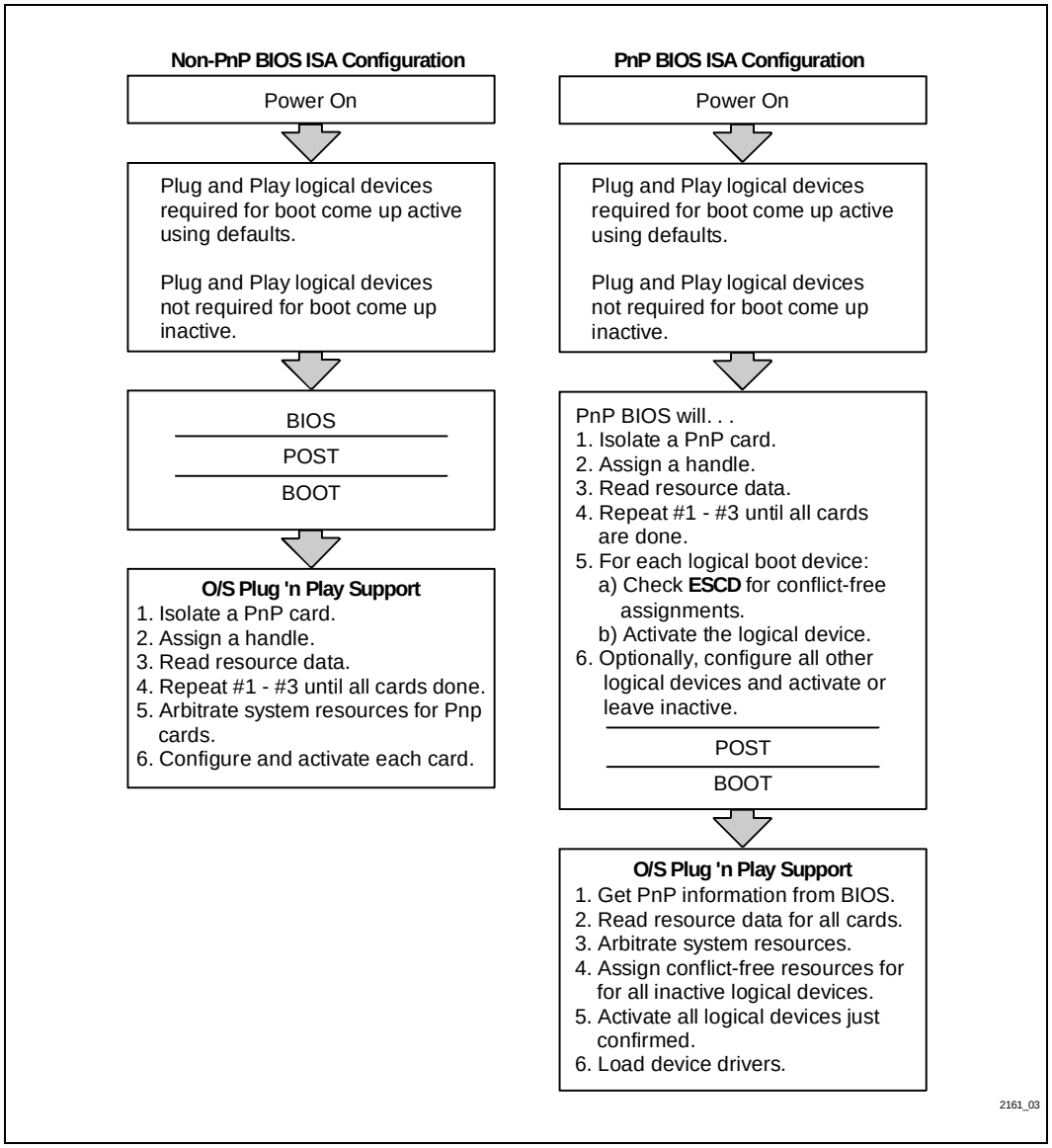


Figure 3. Possible PnP/Non-PnP BIOS ISA Add-In Card Configuration Flow. Note the importance of the ESCD in the Plug and Play Configuration Flow

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

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

3.3 Locked Devices & Bitmap Storage

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

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

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

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

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

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

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

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

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

For these applications, a 2-Mb boot block is the prudent choice. With twice the memory space as the standard PC memory map allots to BIOS, the 2-Mb boot block flash

offers plenty of head room—just what a growing PC needs.

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

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

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

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

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

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

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

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

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

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

These are just some of the implementation concerns that need to be addressed. The following section will shed

some light on these questions and provide an example implementation of a PnP flash BIOS.

5.1 Overview of the 2-Mb Boot Block

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

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

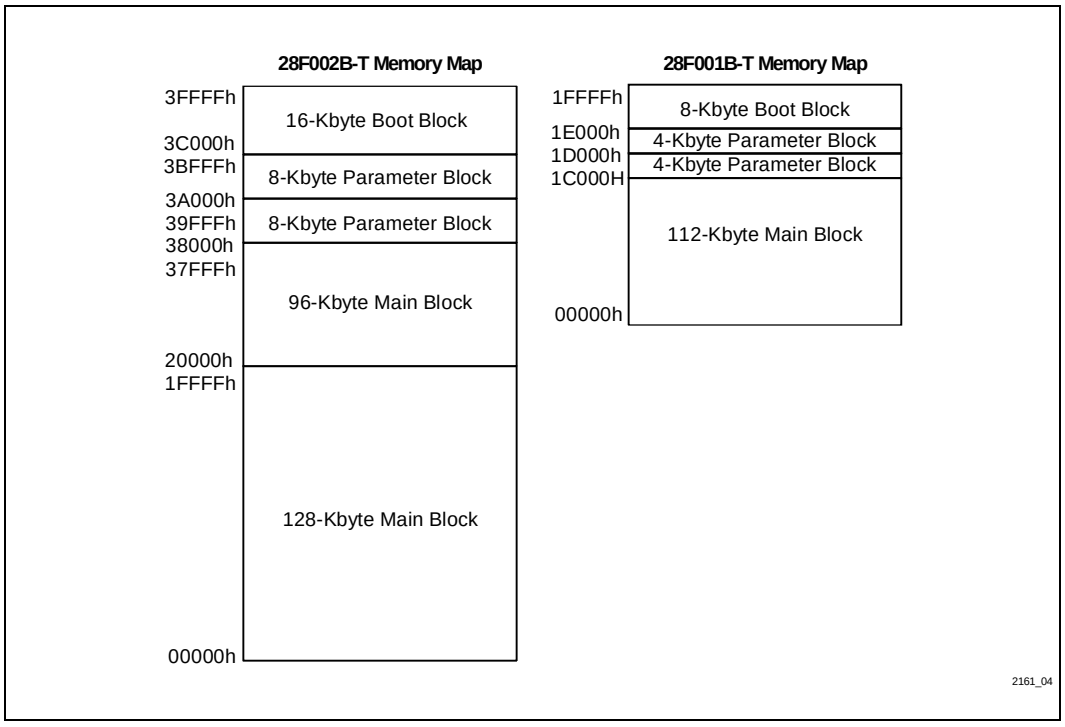


Figure 4. Architectural Organization of 28F002B-T and 28F001B-T Flash Memory Devices

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

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

A complete solution cannot be achieved without mention of available package options. The boot block products are offered in PDIP, PLCC, PSOP, and TSOP form factors. Due to handling similarities in the manufacturing

flow, more vendors are changing from the previous PLCC package standard to the smaller PSOP package.

5.2 PnP Boot Block Flash BIOS Implementation (Hardware)

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

5.2.1 BIOS BOOT CODE RELOCATION

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

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

switch between its upper 128-KB block (which includes the boot and parameter blocks) and its lower 128-KB block (which is where the main BIOS could be stored).

The system initialization routine and the configuration utilities should be located in the upper 128-KB block of the 2-Mb boot block. The lower 128-KB block should be used for runtime BIOS services and other advanced features. Alternatively, the upper 128-KB main block may store the initialization routine as well as the bulk of the runtime services or functions. The lower 128-KB block can then be used for additional advanced functions or esoteric service routines that are used infrequently.

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

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

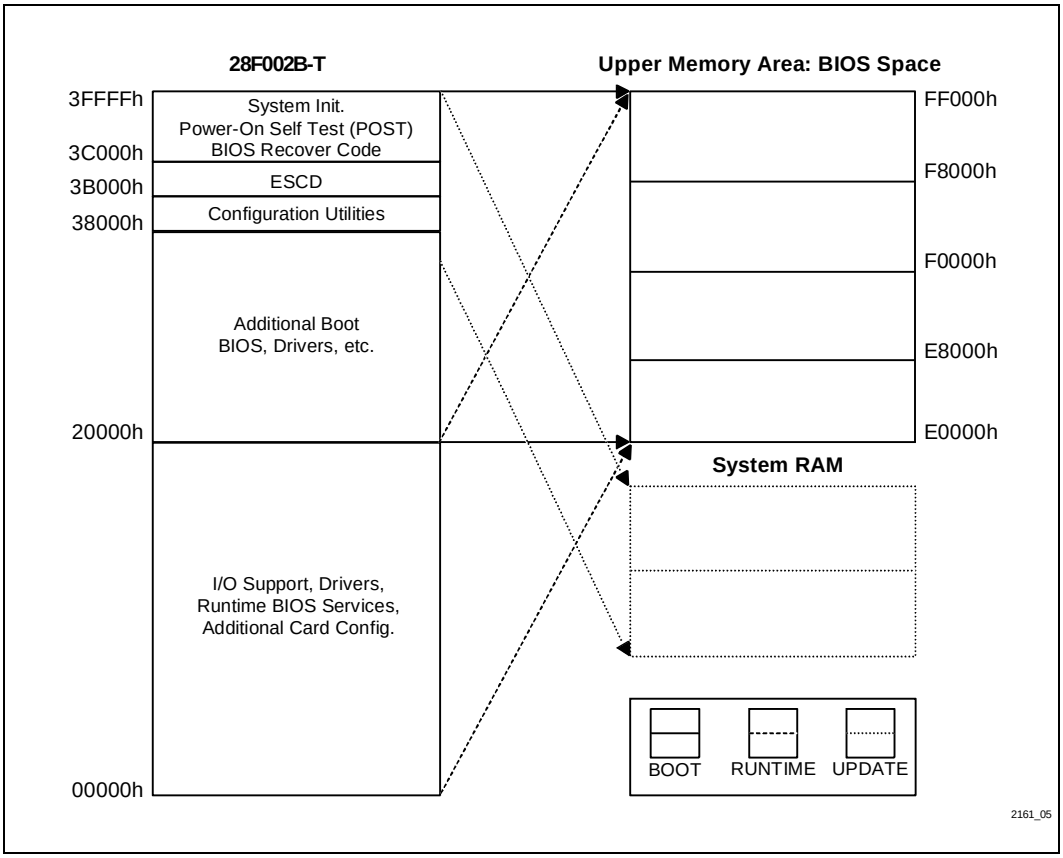


Figure 5. BIOS Relocation for 2-Mb Boot Block

5.2.2 V_{PP} GENERATION AND WRITE PROTECTION

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

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

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

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

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

5.2.3 HARDWARE DESIGN EXAMPLE

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

5.2.3.1 V_{PP} Generation Circuit

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

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

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

5.2.3.2 RP#

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

5.2.3.3 WE#

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

5.2.3.4 CE#

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

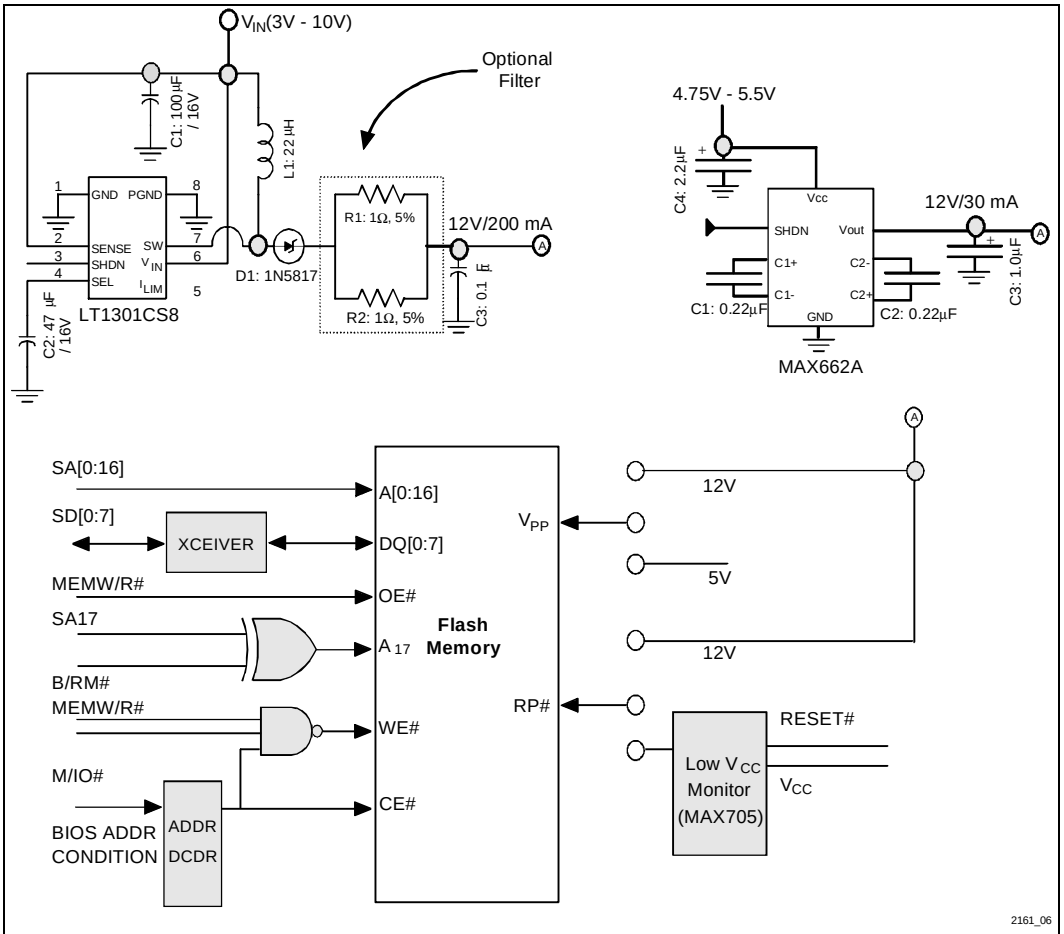
5.2.3.5 OE#

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

5.2.3.6 Address Inversion for 2-Mb Boot Block

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

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



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

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

Figure 6. PnP Boot Block Flash BIOS Hardware Implementation Example

5.3 PnP Boot Block Flash BIOS Implementation (Software)

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

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

5.3.1 PROGRAMMING CONSIDERATIONS

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

Table 1. Generic Subfunctions for Flash Programming Utility

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

The generic specification is as follows:

Input: AH = 0Fh
AL = Subfunction

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

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

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

A few caveats of which to be aware:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.3.2 REPROGRAMMING CONSIDERATIONS

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

environment are some of the trade-offs that must be made when considering which methodology is best. There are advantages to both methods. This application note focuses on the ISW approach.

5.3.2.1 In-System Write Considerations

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

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

V_{PP} Generation

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

Data Import Capability

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

Reprogramming Routine

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

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

In system reprogramming of the system BIOS is one of the many advantages that flash memory brings to BIOS world. The algorithm needed to accomplish this reprogramming will vary from vendor to vendor. Rather than advocate any one method of implementation, the flowchart in Figure 7 is provided to serve as a guide. This enables flexibility of design while insuring that all necessary components are incorporated into the reprogramming code. Even though the flowchart may not indicate so, user consideration should be embedded in the update routine, i.e., status bars, confirmation prompts, etc.

Communications Software

Whatever means is used to download the information to be programmed should guarantee accurate data transmission. The protocol employed can be a simple read-back technique or a complex error-free communications protocol. The simple read-back methodology consists of the CPU indicating to the

system that it wishes to update the BIOS by asserting the R_INTR interrupt. Once the PnP BIOS acknowledges this request, it prepares the flash device for updating and transfers control to the processor. The flash memory then waits for the R_DONE interrupt. Once the reprogramming is complete, the system should resend the update code to verify the programming sequence.

5.3.2.2 Command User Interface

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

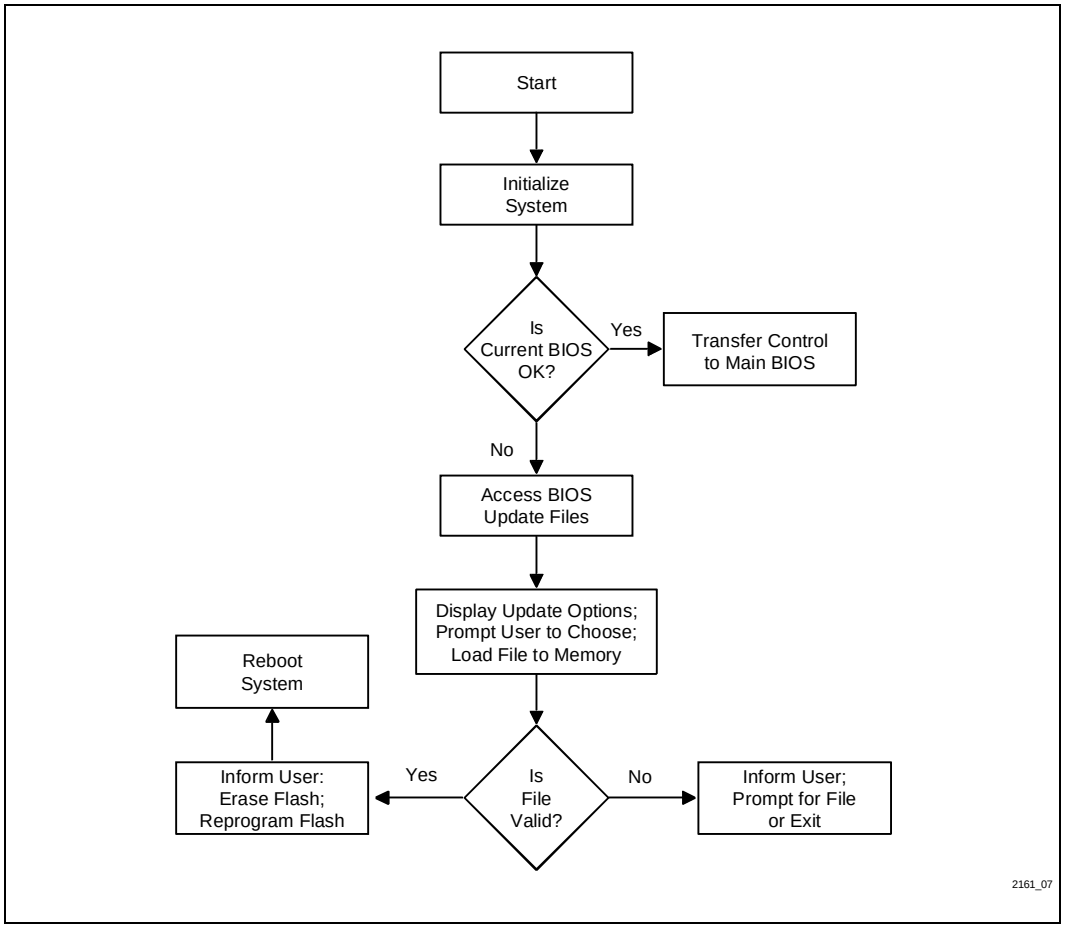


Figure 7. Flowchart for Update Algorithm.
Note that although this is a fairly generic algorithm, similar flows have been implemented by BIOS vendors and OEMs since 1991.

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

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

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

BA = Block Address to be erased

WA = Address to be programmed

WD = Data to be programmed at address WA

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

IID = Intelligent Identifier Data

SRD = Status Register Data

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

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

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

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

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

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

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

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

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

6.0 DESIGNING FOR THE FUTURE

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

6.1 The 5V-Only Question

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

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

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

6.1.1 DESIGNING FOR SmartVoltage

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

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

SVT Write Protection

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

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

7.0 SUMMARY

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

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

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

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

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

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

8.0 ADDITIONAL INFORMATION

8.1 References

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

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

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

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

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

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

APPENDIX A
PINOUTS, LEAD DESCRIPTIONS AND BIOS-SPECIFIC INTERRUPTS

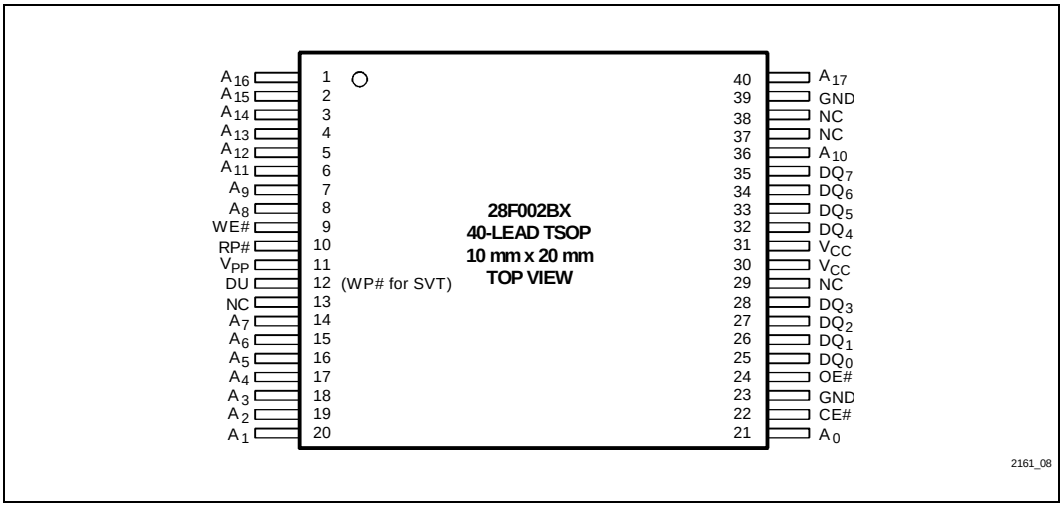


Figure 8. 40-Lead TSOP 28F002BX Flash Device Pinout

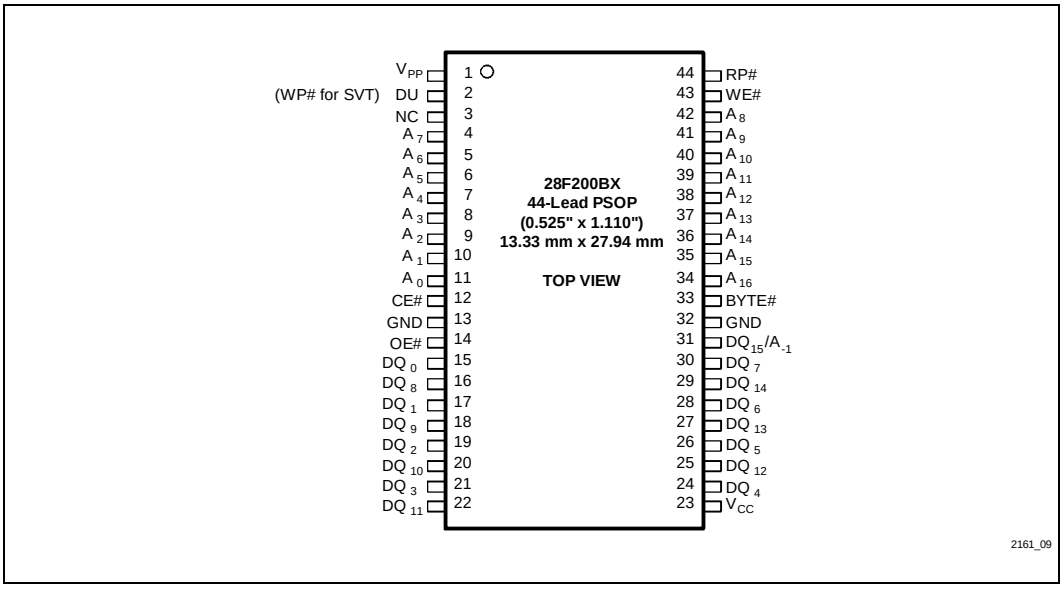


Figure 9. 44-Lead PSOP 28F002BX Flash Device Pinout

Table 3. Definition of 28F002B Pins

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

Table 4. Full Listing of BIOS-Specific Interrupts

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

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

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