intel®

AP-604

APPLICATION NOTE

Using Intel's Boot Block Flash Memory Parameter Blocks To Replace Eeprom

PETER HAZEN
SENIOR TECHNICAL
MARKETING ENGINEER

November 1995

Order Number: 292148-002

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

Intel's boot block flash memories provide updatable code and data storage for a wide range of applications including cellular phones, modems, PC BIOS, automobile engine control and many others. System designers reduce system cost and improve reliability by using Intel's flash memory parameter blocks to replace EEPROM for parameter data storage.

Using software techniques described in this paper, designers can replace EEPROM with Intel's flash memory parameter blocks in many applications that previously used EEPROM for parameter data storage. For example, cellular phone designs use Intel's flash memory parameter blocks to store data such as telephone numbers, time of use and user identification information. Automobile manufacturers use Intel's parameter blocks in engine control applications to store fault codes and engine optimization parameters. In each of these cases, manufacturers save both EEPROM component and inventory costs by using Intel's boot block flash memory for parameter storage in addition to storing application code. Additionally, improved reliability is achieved with lower system device and pin counts. Finally, the amount and frequency of parameter storage is improved.

This paper describes a linked-list data structure for storing parameters in Intel's flash memory parameter blocks using a scheme that emulates byte alterablility. A review of flash memory fundamentals shows how flash is used in a system and defines the constraints for implementing the software scheme. Reference source code is available from Intel.

2.0 REVIEW OF FLASH MEMORY FUNDAMENTALS

Flash technology brings unique attributes to system memory. Like RAM, flash memory is electrically modified in-system. Like ROM, flash is nonvolatile, retaining data after power is removed. However, unlike RAM, flash cannot be rewritten on a byte basis. Flash memory reads and writes on a byte-by-byte basis, and adds a new requirement: it must be erased before it can be rewritten. Table 1 shows each flash memory operation, the size of data, and the time it takes for each operation.

Writing (or programming) flash is the process of changing "1"s to "0"s. Erasing flash is the process of changing "0"s to "1"s flash memory is erased on a block-by-block basis. Blocks are defined by a fixed address range as shown in Intel's 4-M Boot Block Memory Map in Figure 1. When a block is erased, all address locations within a block are erased in parallel, independent of other blocks in the flash memory device.

Intel's flash memory boot block products are specified to work over 100,000 cycles when operating at 5V V_{CC}. A cycle is defined as an erase operation. For example, if each of the 8 Kbytes in one of the parameter blocks are successively programmed and then the block erased, 1 cycle has completed. This specification is important in determining how many parameters can be stored and how many times those parameters can be updated.

Since flash memory cannot be re-written to the same address location without first erasing an entire block of memory, software techniques are used to emulate byte alterability using the two 8-KB parameter blocks shown in Figure 1.

Table 1. Flash Memory Read, Write and Erase Operations*

Operation	Min Segment	Typical Time	Max Time
Read	Byte	60 ns	60 ns
Write	Byte	10 µs	160 µs
Erase	Block (8 KB)	0.8 sec	4.4 sec

*Specifications for Intel's SmartVoltage 4-M boot block product operating in x8 mode at 5.0V V_{CC} and 5.0V V_{PP}. Refer to the SmartVoltage 2/4-M datasheets.

3.0 SYSTEM OPERATION

In addition to storing parametric data, Intel's boot block flash memory is often used to store updatable application code. In many systems, the hardware- lockable boot block stores the kernel code necessary to initialize the system and invoke a recovery routine if code is lost. The boot block also typically stores the code necessary to program and erase the flash memory.[1] Today, flash memory products do not provide the capability to read from one address location in the flash device while writing to another address in the same device. This means any code that writes to flash must be downloaded to RAM.

[1]Program and Erase code for Intel's boot block flash memory products is available on the Intel BBS.

PRELIMINARY

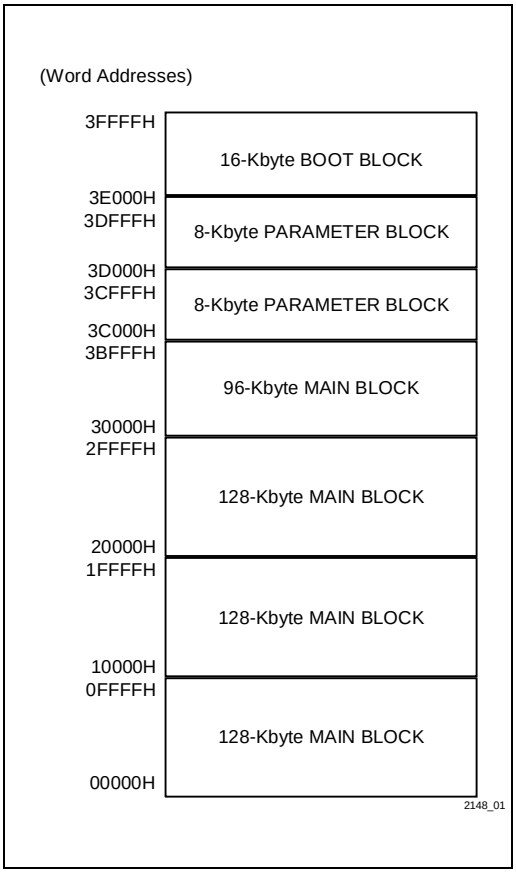

Figure 1. Intel's Boot Block Flash Memory Map

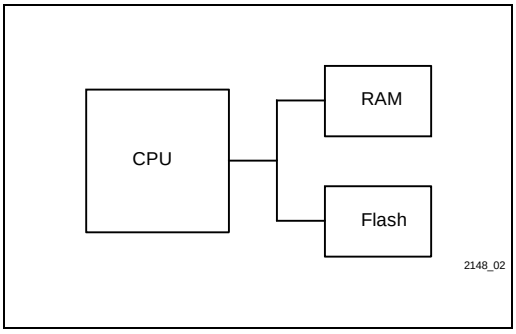

Figure 2. For In-System Write to Flash, Code is Executed out of RAM

4.0 SOFTWARE SCHEME FOR EMULATING BYTE ALTERABILITY

By using the two parameter blocks in Intel's boot block components along with software techniques, data can be stored a byte at a time and the erase operation of flash can be completed as a background task, to emulate byte alterability.

5.0 PARAMETER DATA STRUCTURE

A linked-list data structure provides an arrangement of data that is well-suited to the needs of flash memory. For example, suppose we want to store three parameters that will be updated. We store each parameter in a record. Each record has two fields: parameter_value and next_record. The first field contains the value of the parameter. The second field is a pointer containing the address of the next record for that parameter. ParameterX is a pointer variable that contains the address of the first record for that parameter. So Parameter1 represents an address. Stored at that address is the address of the first record of Parameter1. The first record contains the first Parameter1 value and the address of the second Parameter1 record. The second record contains the latest value for that parameter and the address of the third record, and so on. In the last record, the next_record field contains FFH to indicate there are no more records. FFH is used here to indicate no more records, since this is the default state of an erased byte in flash memory. Each time a parameter is updated, the code searches for the first available address location in the parameter block, writes the new value in the value field of the new record, and then updates the next_record field of the previous record. So, each record includes a value and a pointer, or link to the next record. Such a data structure is called a linked list. By using a linked list the system can quickly find the latest value for a given parameter.

Figure 3 shows an example of the linked-list structure. To simplify the example, a one-byte field is used for both parameter_value and next_record. In an actual implementation, at least two bytes would be needed for next_record, which points to another location in the parameter block. The number of bytes required for parameter_value will depend on the specific parameter information being stored.

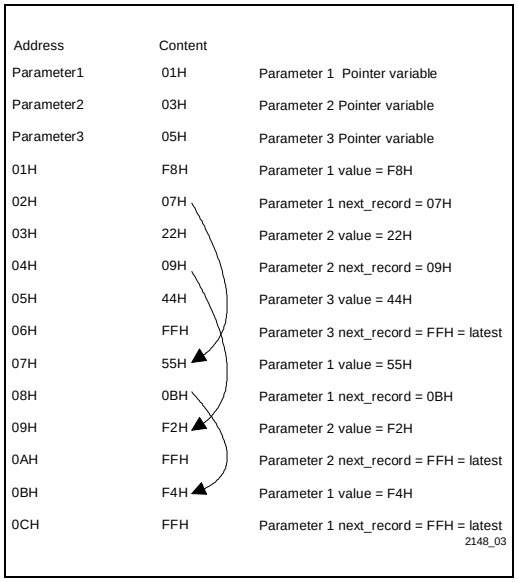

Figure 3. A Linked List Parameter Record Includes Two Fields: Parameter_Value and Next_ Record

An alternative approach to using a linked-list structure, is to define a parameter ID field and a parameter status field that indicates whether the current parameter record is the latest. In this alternate scheme, to retrieve the latest parameter, the system reads through every parameter instance until it finds the latest value for a given parameter.

6.0. BLOCK TRANSFERS

Parameters are stored until the parameter block is filled or until there is not enough memory in the parameter block to store another complete record. When this point is reached, the latest value for each parameter is transferred to the second parameter block, and the linked-list data structure continues in the second parameter block.

A Block_Header record at the beginning of each parameter block indicates the status of the block. That is, information such as which parameter block is the active block, if the block is transferring data, and if the block is erased.

7.0 ERASING THE PARAMETER BLOCK

After the valid parameters are transferred from the first parameter block to the second, the first parameter block is erased. Recall that flash memory requires approximately half a second to erase each parameter block. Since this much time may not be available during system operation, Intel's flash memory products feature an Erase Suspend command. With this command, the erase operation can be suspended to allow the system to read from another block in the device. When an Erase Suspend command is given to the flash memory, the erase operation is suspended and the memory enters an erase suspend state where the system can read from another block in the flash memory. When time is available again for erasure, the Erase Resume command instructs the flash memory to continue erase from the point where it previously suspended erase. This allows the erase operation to be implemented within a finite software loop time, by using multiple calls. Once the first parameter block is completely erased, it is ready to store parameter records after the second parameter block is filled. It is important to note that no new parameters may be written until the block erase operation completes. Current boot block flash memories do not allow writes while erase is suspended. See Figure 4 for a review of the parameter storage scheme.

8.0 SYSTEM REQUIREMENTS

As described earlier, RAM is required to execute code during program and erase operations. The amount of RAM required depends on the complexity of the parameter storage data base. The code that must be downloaded to RAM includes the flash memory read, write and erase routines. The size of this code is in the range of 512 bytes to 1 Kbyte. In addition, flash memory space will be necessary to store the program code. Reference code that will be available from Intel, is approximately 15 KB in size. Only a small subset if this, approximately 1 KB, is downloaded to RAM.

Another system requirement is an adequate V_{PP} voltage for write and erase. Many of today's flash memory products require 12V to write and erase in-system. Intel's new SmartVoltage products feature 5V write and erase operation when 12V is not available.

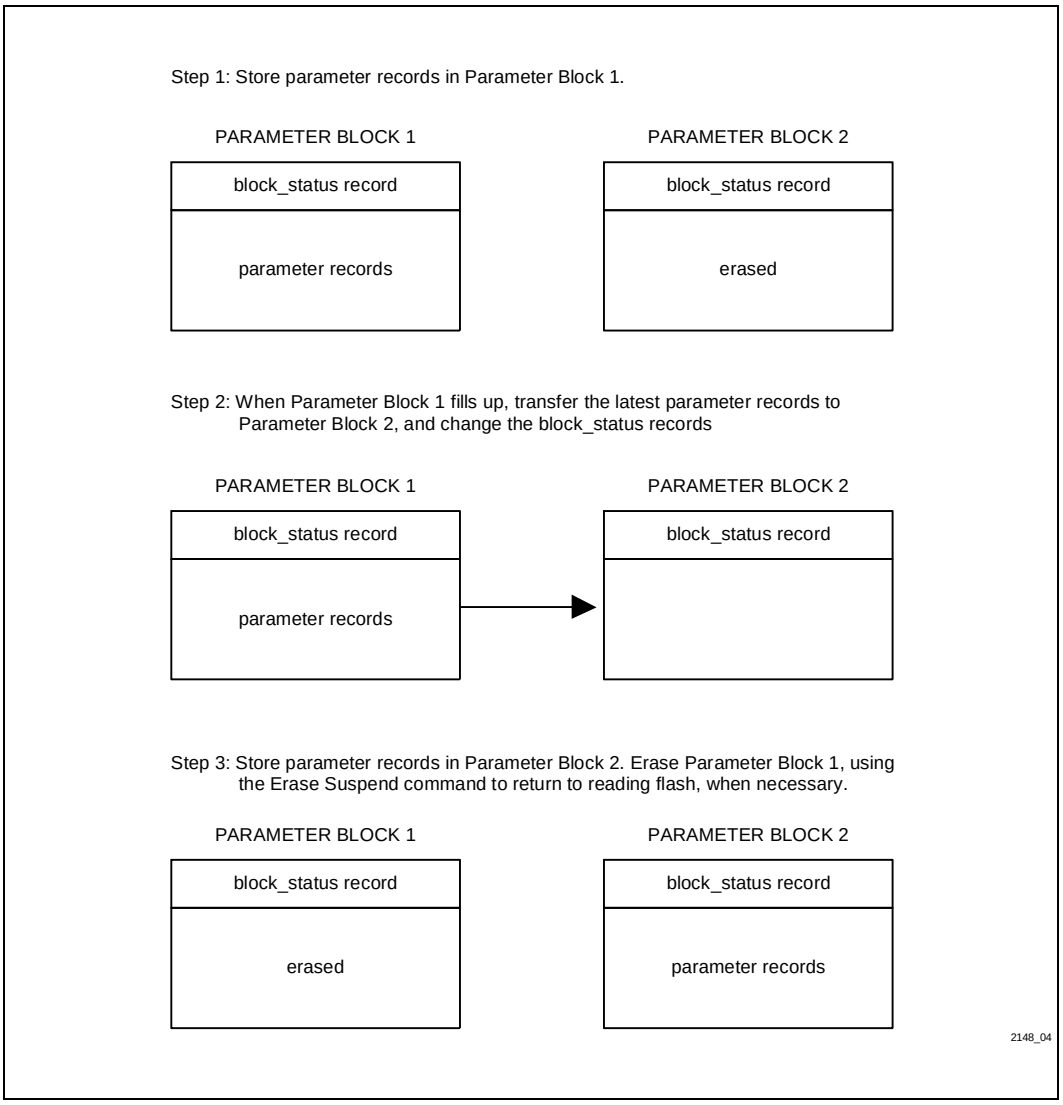

Figure 4. Using Two Parameter Blocks to Emulate Byte Alterability

9.0 POWER LOSS

What if power is lost in the middle of an erase or in the process of updating parameter values? Power loss can be reliably handled by defining additional fields in both the parameter and block records. For example, in addition to the parameter_value and next_record fields that we defined for our parameter record, we can define a parameter_status field. One status field bit indicates that a parameter update is beginning and another that the parameter update has completed. So, if we lose power in the process of updating a parameter, we know the status of each parameter entry when power is restored. For example, when power is restored, if we find that the status bit reflects that a parameter update began but did not finish, we know that record is invalid and should be

discarded. The same concept can be applied to the block_status record to handle erase operations that may be interrupted by power failure as well as parameter transfers between blocks.

10.0 INITIALIZATION

An initialization process determines the state of the parameter blocks. By reading the block_status record you can determine which block is the active block and whether the other block must be erased. Upon the first initialization, the parameter blocks can be erased and the block_status records created.

11.0 BIT MANIPULATION

Earlier, we reviewed how flash memory is read and programmed on a byte-by-byte basis, and erased on a block basis. Intel's flash memory actually has the capability to be programmed one bit (or multiple bits) at a time. Recall flash memory programming is the process of changing "1"s to "0"s. Single bits can be programmed by masking the other bits in a byte or word with "1"s as shown in Figure 5. By taking advantage of this feature, you can minimize the memory space necessary for the various status fields.

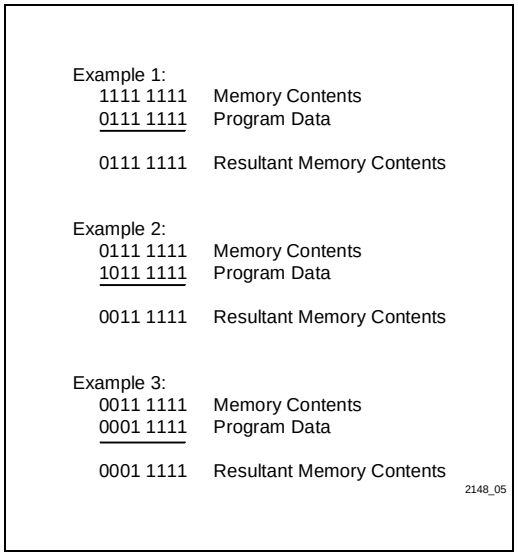

Figure 5. Flash Memory can be Programmed a Bit at a Time by Masking all Other Bits in a Byte with "1"s

12.0 TIMING

System timing analysis is required to determine the amount of time available for:

- Reading parameters
- Downloading write/erase code to RAM
- Writing parameters
- Transferring parameters to a new block
- Erasing a parameter block

Precise timing will depend on the specific system implementation. In addition to the device timing, software overhead timing should be considered.

The time required for reading parameters will depend on the size of each parameter record and the number of parameter record instances that must be read before reaching the valid parameter record. Multiply the number of byte or word reads by the system read cycle time to determine the total time for reading a valid parameter.

Each time a write or erase operation is executed on the flash memory, a sub-routine containing the program and erase drivers must first be downloaded to RAM. The time required for downloading this code to RAM depends on the size of the code, which is likely to be 1 KB or less. Multiplying the code size by the write cycle time determines the time for downloading the code to RAM.

To determine the maximum time required for writing a parameter, the worst case word or byte write time is needed for the flash memory component. By multiplying the worst-case word write time by the number of words per parameter record, the worst-case parameter write time can be determined.

The time for transferring valid parameters from one parameter block to the other will, of course, depend on the number of parameters stored. If this operation is completed as a foreground task, a block of time will be required that includes the time to read the valid parameters and write these parameters to the new parameter block. This operation can also be treated as a background task which is often necessary. For applications with a defined main software loop time, the transfer operation can be executed by determining the available time in the main loop, beginning the transfer of parameters and then suspending this task before the main loop time expires. Multiple calls of the main loop are required to completely transfer all parameters to the new block. Total time to complete the task will depend on the

amount of time available in each loop and the number of calls necessary to complete the operation.

Like parameter transfers, block erase can be treated as a foreground or background operation. When treating erase as a background operation, total erase time will depend on the amount of time available in the software loop. Determine the number of calls required by dividing the total erase time by the amount of time available in each call. Multiply the number of calls by the total time per loop to determine the total time for erasing a parameter block.

13.0 CYCLING

Intel's flash memory boot block products are specified at 100,000 erase cycles. To determine how this affects your parameter storage, use the equation shown in Figure 6. The equation can be solved for either the number of parameter_record instances or the parameter_record size, depending on which variables are known. This implementation provides improved cycling endurance, as compared to EEPROM.

$$\text{number of parameter\_record instances} = \frac{\text{8 KB - (block\_record size)}}{\text{parameter\_record size}} \times 100{,}000 \text{ Cycles}$$

Figure 6. Parameter Storage Equation

14.0 CONCLUSION

This paper describes software techniques for emulating byte alterability using two flash memory parameter blocks. System designers reduce system cost and improve reliability by using Intel's boot block parameter blocks for parameter data storage, replacing EEPROM.

15.0 ADDITIONAL INFORMATION

15.1 References

Order Number	Document
290530	2-Mbit SmartVoltage Boot Block Flash Memory Family
290531	4-Mbit SmartVoltage Boot Block FlashMemory Family
292130	AB-57 "Intel's Boot Block Architecture for Safe Firmware Updates"

15.2 Revision History

Item	Description
-001	Original Version
-002	Parameter Storage Equation changed

Filename:	292148_2.DOC
Directory:	C:\TESTDOCS
Template:	C:\WORD\ZAN____1.DOT
Title:	Using Intel's BB Flash Memory Parameter Blocks to Replace EEPRO<
Subject:	
Author:	Mary Ann Hooker
Keywords:	
Comments:	
Creation Date:	10/19/95 9:42 AM
Revision Number:	5
Last Saved On:	01/19/96 9:04 AM
Last Saved By:	Ward McQueen
Total Editing Time:	2 Minutes
Last Printed On:	01/19/96 9:04 AM
As of Last Complete Printing	
Number of Pages:	8
Number of Words:	2,650 (approx.)
Number of Characters:	15,107 (approx.)