TECHNICAL
PAPER

Software Download Utility for Intel Boot Block Components

SAMUEL C. DUFOUR
MEMORY COMPONENTS
DIVISION

July 1996

Order Number: 297707-001

CONTENTS

Revision History

Number	Description
-001	Original Version

1.0 INTRODUCTION

Intel's boot block flash memories provide updateable 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, improve reliability and allow for easy upgradeability by integrating Intel's boot block flash memory into their products for code storage.

Using software techniques described in this paper, designers can easily update portions of blocks while retaining valid portions of blocks. This essentially means that portions of code in blocks can be updated thereby reducing the external memory (RAM or flash) overhead needed in an environment utilizing flash for code storage.

This paper describes flash reference software written in C for updating code stored in flash (chunks of code within any given block that are less than or equal to the size of the block of which they are contained) without disturbing surrounding code.

NOTE:

This utility has only been used in a simulated environment and should only be considered as reference in creating a product.

2.0 REVIEW OF FLASH MEMORY FUNDAMENTALS

Flash technology brings unique attributes to system memory. Like RAM, flash memory is electronically modified in-system. Like ROM, flash is nonvolatile, retaining data after power is removed. However, unlike RAM, flash cannot be rewritten on a byte or module basis. Flash memory reads and writes on a bit-by-bit basis, and adds a new requirement: it must be erased before a bit can be reprogrammed. An erase operation must occur at the block level. The overall effect is that individual code objects within flash blocks cannot be altered in-place like that of mechanical drives or RAM.

Flash is preferred, in most instances, over mechanical drives due to incredibly rapid storage and retrieval of code. To give an idea of how rapidly flash works, Table 1 depicts each flash memory operation, the size of data, and the time needed to execute each operation.

Table 1. Example Flash Memory Read, Write and Erase Operations(1)

Operation	Min Segment Size	Typical Time	Max Time
Read	Byte	60 ns	60 ns
Write	Byte	10 µs	160 µs
Erase	Block (8-KB Parameter Block)	0.8 sec	7 sec

NOTE:

1. Times for Intel's SmartVoltage 4-Mb boot block product operation in x8 mode at 5.0V V_{CC} and 5.0V V_{PP}. Refer to the SmartVoltage 2-/4-Mb datasheets.

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-Mb Boot Block Memory Map, 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 boot block flash memory products are capable of being cycled over 100,000 times when operating at 5V V_{CC}. In short, the definition of a block "cycle" is the act of both programming and erasing one block a single time. For example, if all of a 128-KB main block is successively programmed and then the block is erased, one cycle has completed. This specification is important in determining how many times code can be stored and how many times this code can be updated.

Since flash memory cannot be re-written to the same address location without first erasing an entire block of memory, this reference software can be used to emulate code alterability using the upload utility (see Figure 1).

NOTE:

For current Intel NOR flash technology, a flash byte or word actually can be rewritten as many times as you wish, as long as you never need to change a bit from a "0" to a "1."

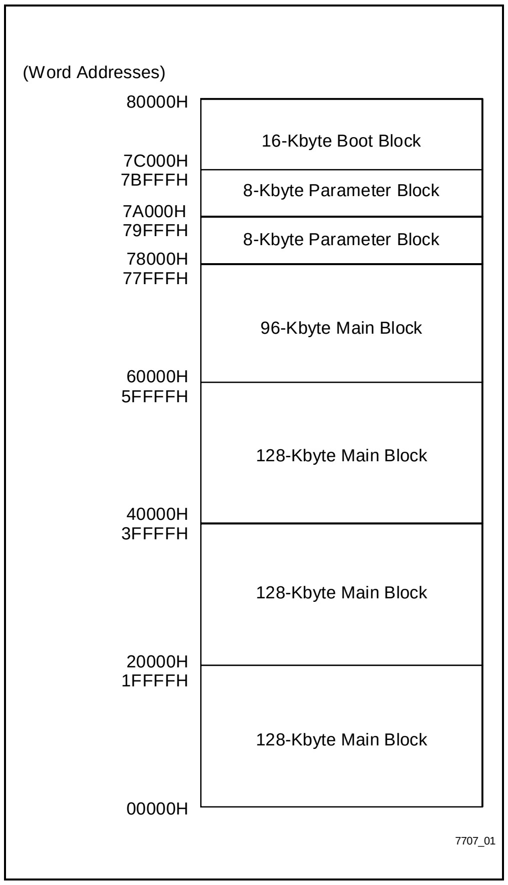

Figure 1. Intel's 4-Mbit Boot Block Flash Memory Map

3.0 BLOCK DOWNLOAD

By using adequate RAM or flash along with flash boot block components and the download utility, code can be seamlessly rewritten in smaller subsets than the full block requirement imposed by flash erase operation. Adequate RAM or flash is defined to be: erase block size *minus* largest amount of code required to be copied.

When a request is made to alter code within a main block of a flash component, the download utility copies the code which will not be changed to spare memory. RAM or flash erases the flash block, copies the code from spare memory back to the original block and downloads new code to the space that has been cleared for it. The download process also tracks status of each step in the process to allow power-off recovery.

4.0 DOWNLOAD STATUS STRUCTURE

Prior to, and during, the altering of code in the designated block, the download status structure table (located in one of the 8-KB parameter blocks) is updated to reflect the state of the code being manipulated; each step of the download process will be reflected in the status. The download status structure must be maintained to assist the recovery procedures in determining where the download left off in case of an accidental power-off. If a download was in progress and the spare memory being used for download was flash memory, the initialization can continue the process where it left off. If the spare memory being used was RAM, the valid code which was lost can be flagged to the caller, allowing the user to know which code needs to be downloaded. This status information is maintained in the parameter blocks

For greater detail regarding the makeup of download status structure see Appendix A.

5.0 SYSTEM REQUIREMENTS

- The system must have around 1 KB of RAM available for download of flash programming algorithms if executing out of the boot block component.
- The system must have at least 6 KB of flash memory available for this utility software.
- The system must contain enough memory in either RAM or flash to temporarily store the maximum amount of valid code to be retained (e.g., to download 48 KB to a 128 KB block: 128 KB – 48 KB = 80 KB of external RAM/flash is required—see Figures 2 and 3).
- This implementation assumes that one of the 8-KB parameter blocks is available for use in storing the 105-byte download status table.

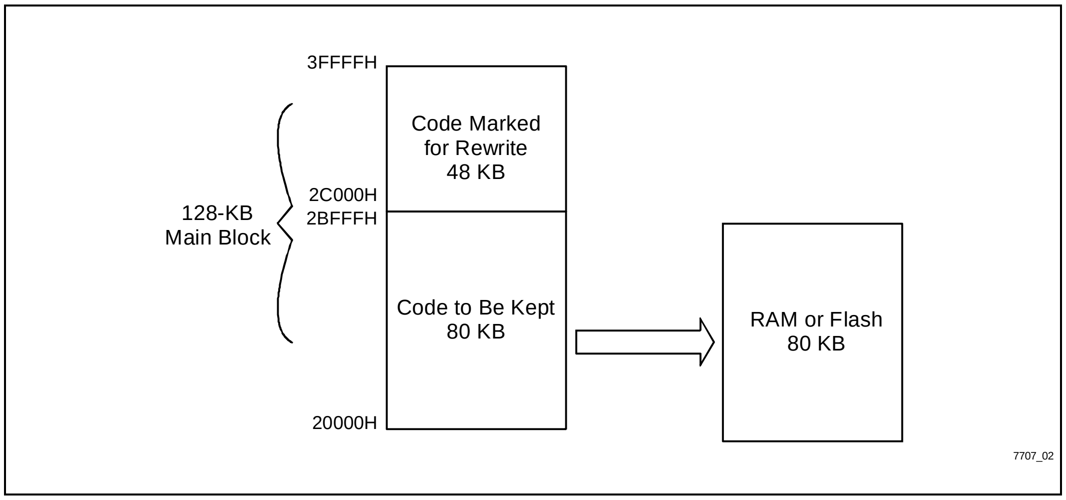

Figure 2. Copying Good Code to Boot Block for Block Erase

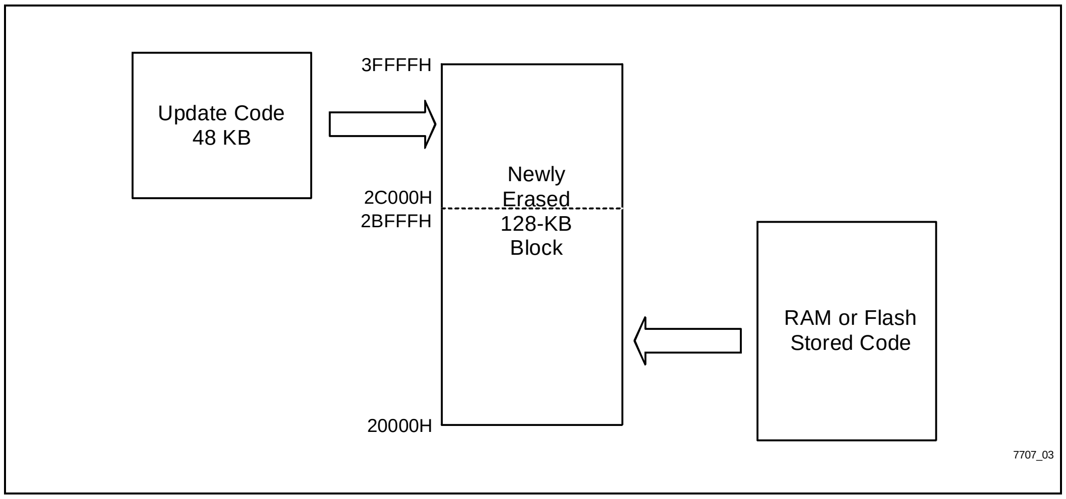

Figure 3. Download of Update Code and Code Held in RAM/Flash to Boot Block

6.0 IMPLEMENTATION CONSTRAINTS

The following information pertains to the restrictions that must be observed while implementing this utility.

- The boot block components data I/O lines do not support a read from memory while writing to the same memory. This forces any code executing from flash which will need to write to the same device to be downloaded to RAM before execution.
- Interrupts will be disabled during erases and during word/byte writes.
- A status table will be kept in the flash parameter block which will determine current status of download. This will allow the system to determine what steps it should take to complete the process upon initialization.
- Software will have an option to compile to execute from either RAM or flash. This will determine if the flash programming algorithms are downloaded to RAM or not.

7.0 RECOVERING FROM POWER LOSS

Recovery from power loss is one of the greatest strengths of this download utility. As mentioned in Section 4, by using a status control table the download operation can be reliably tracked from beginning to end. Should a power-down occur, operation will continue from where it left off, if possible. Otherwise, notification will be delivered to help correct inconsistencies in data integrity.

8.0 CONCLUSION

This paper described software techniques for emulating code alterability less than or equal to a single block using the software download utility to control flash media structures. System developers reduce system overhead and improve reliability by using Intel’s boot block flash. The download utility can further increase the value of boot block flash by offering bit-wise alterability and fault tolerant power-down recovery.

APPENDIX A
DOWNLOAD STATUS STRUCTURE

The structure below contains the status information pertaining to the state in which the download is in throughout the entire download event.

```
typedef struct  download_status      {
    byte        DownloadStatus;      /* Indicates current status of the download
    dword       pCode;               /* Pointer to Valid Code to be retained or Address being downloaded. */
    dword       pSpareMemory;        /* Pointer to spare area where valid code may be copied */
    dword       Size;                /* Size of code to be retained or code being downloaded */
    byte        SpareType;           /* RAM or flash for aid in power-off recovery */
    byte        DataTransferStatus;  /* Ensures validity of this structure */
} DOWNLOAD_STATUS;
```

Below are the descriptions of the variables contained within the Download_Status structure along with what the DownloadStatus's value indicates:

Field	Description
DownloadStatus	A bit mapped status field which indicates the current status of the download. Table I depicts the definitions associated with each valid condition status value.
pCode	This pointer will point to the valid code to be retained if in the download preparation phase or if download is in progress, will point to the address being downloaded to.
pSpareMemory	This pointer will point to the spare memory (RAM, Boot Block, Main Block or Parameter Block) if the status indicates download preparation phase, or will indicate location being downloaded from if during download. Location being downloaded from is useful for debugging only, not for recovery.
Size	This field indicates the size of the code being retained if in the download preparation phase, or will indicate amount of code being downloaded if in the download phase.
SpareType	This field indicates if the valid code for download preparation is being downloaded to RAM or to flash. This assists the initialization procedure in determining if the download can be continued or if an error code should be flagged to the user.
DataTransferStatus	This field indicates to the initialization procedure if this structure was completely finished being written when a power off occurred. If this field is FF, this structure contains unreliable information. If this field is 00, this structure is valid.

Table I. DownloadStatus Values with Associated Definitions

Condition Status Value	Definition
11111111b	Erased status. Indicates end of status table.
01111111b	Copy Valid Code to Spare.
00111111b	Erase Original Block.
00011111b	Copy from Spare to Original Block.
00001111b	Erase Spare Block (only entered if SpareType = Flash).
00000111b	Download in progress.
00000011b	Download complete.
00000001b through 00000000b	Reserved for future use.

APPENDIX B
DOWNLOAD INITIALIZATION

The Download Initialization routine should be called upon the system's initialization. This routine will evaluate the download status table to determine if a download was interrupted by a power-off. If the SpareType is RAM, the entire status will be returned to indicate the failure. This will allow the system to reload the entire block if necessary.

If the SpareType is flash, the download preparation will be continued.

Status	Action
Copy to Spare	Will use pointers in table to re-copy the valid code to the spare area. Will then progress to the next step.
Erase Original	If SpareType is flash, will verify block is erased. If not erased, will perform the erase and progress to the next step. If the SpareType is RAM, will indicate to the user that all software in the block may have been lost.
Copy Spare Back to Original	If SpareType is flash, will copy from spare back to the original block and will return indication that download preparation is complete. If SpareType is RAM, will indicate to the user that all software in the block may have been lost.
Erase Spare	If SpareType is flash, will erase the spare block. SpareType of RAM should never enter this state.
Download in Progress	Will indicate error to the user. This indicates that any download preparation was completed and that the new code must still be downloaded.
Download Complete	No action required.

Download Initialization Call Format

The 'C' call to the initialization procedure will be as follows:

ERR_STATUS * DownloadInitialization( );

Below is the err_status structure which contains an initialization status followed by the most recent DOWNLOAD_STATUS entry:

```
typedef         struct      err_status  {
     byte                          InitializationStatus;
     Download_Status               DownloadInformation;
} ERR_STATUS;
```

The descriptions of the values contained within the ERR_STATUS structure are described below:

Field	Description
InitializationStatus - (int)	Contains the status of the call. Table II depicts the definitions associated with each valid condition status value.
Download_Status	

Table II. InitializationStatus Values with Associated Definitions

Condition Status Value	Definition
00000000b	OK. No downloads were interrupted.
00000001b	Download preparation ERROR. All code in the block to be downloaded was lost.
00000010b	Download ERROR. Download preparation completed, but new code must be downloaded again.

APPENDIX C
DOWNLOAD PREPARATION

The Download Preparation function will allow the user to save a portion of a block into RAM or flash while the block is being erased. It will then copy the information back to the block. Once all of this has completed, the block is ready to have new code downloaded to the portion that was not saved. This program assumes all code to be saved is contiguous (Note: If code is not contiguous, this interface could be changed to accept an array. This would require slight modifications to the interface). Enough memory (RAM or flash) must exist to save the code which should be retained.

If software is executing from boot block components, programming or erasing flash will require a routine to be downloaded into RAM to interface with the command user interface of the flash part and to handle all processes while the part is read status mode.

Download Preparation Call Format

The 'C' call to this procedure will be as follows:

int DownloadPreparation (byte Command, dword pSpare, byte SpareType, dword pValid, dword Length);

Below are the descriptions of the variables passed to the DownloadPreparation function along with a description of the value which is returned from this function:

Field	Input/Output	Description
status (int returned)	Output	Contains the status of the call. Table III depicts the definitions associated with each valid condition status value.
Command	Input	Indicates if this is a new download preparation or one to be continued. Users should only use the new download option. Initialization will use the continue download option.
pSpare	Input	Pointer to where the spare are is which will temporarily contain the code to be retained.
SpareType	Input	Indicates if the memory which is used temporarily is RAM or flash.
pValid	Input	Pointer to the valid code which should be retained.
Length	Input	Length of valid code to be retained.

Table III. Download Preparation Status Values with Associated Definitions

Condition Status Value	Definition
00000000b	OK. All requested actions were successfully performed.
00000001b	Error. Copy to Spare failed.
00000010b	Error. Erase of Original Block failed.
00000100b	Error. Copy from Spare back to Original failed.

APPENDIX D
DOWNLOAD CODE

The Download Code function will assist in downloading software from a location in memory (RAM) to a location in the boot block component. The area in the boot block component should already be erased.

Download Code Call Format

The 'C' call to the download procedure will be as follows:

int DownloadCode(dword pCodeAddr, dword pDownloadAddr, dword Length);

Below are the descriptions of the variables passed to the DownloadCode function along with a description of the value which is returned from this function:

Field	Input/Output	Description
status (int returned)	Output	Contains the status of the call. Table IV depicts the definitions associated with each valid condition status value.
pCodeAddr	Input	Pointer to memory which contains code to be programmed into the flash device.
pDownloadAddr	Input	Pointer to flash memory which the code will be programmed into.
Length	Input	Length of code to be downloaded.

Table IV. DownloadCode Status Values with Associated Definitions

Condition Status Value	Definition
00000000b	OK. All requested actions were successfully performed.
00000001b	Error in download.

APPENDIX E DOWNLOAD STATUS CLEANUP

The Download Status Cleanup function will evaluate the amount of space left in the parameter block which contains the DOWNLOAD_STATUS table. If the table approaches the end of the block, the block will be erased. Otherwise, the function will return. This function will be called from the DownloadCode function after each download complete.

Download Status Cleanup Call Format

The 'C' call to this procedure will be as follows:

int DownloadStatusCleanup ();

Below is the description of the value which is returned from the DownloadStatusCleanup function:

Field	Input/Output	Description
status (int returned)	Output	Contains the status of the call. Table V depicts the definitions associated with each valid condition status value.

Table V. DownloadCode Status Values with Associated Definitions

Condition Status Value	Definition
00000000b	OK. All requested actions were successfully performed.
00000001b	Error.

APPENDIX F
ADDITIONAL INFORMATION

References

Order Number	Document
290530	*2-Mbit SmartVoltage Boot Block Flash Memory Family Datasheet*
290531	*4-Mbit SmartVoltage Boot Block Flash Memory Family Datasheet*
290539	*8-Mbit SmartVoltage Boot Block Flash Memory Family Datasheet*