TECHNICAL PAPER

EEPROM Replacement with Flash Memory

March 1997

Order Number: 297828-001

CONTENTS

REVISION HISTORY

Number	Description
-001	Original version

1.0 INTRODUCTION

Flash memory is used in a wide range of applications for embedded control code storage· Many of these applications including cellular phones, modems, automobile engine control and others also use a separate EEPROM to store factory, system, and/or user data. With ever-increasing pressure to eliminate components and reduce system cost, designers are looking to use flash memory to emulate EEPROM for simultaneous code and data storage.

Intel introduced an EEPROM emulation methodology based on linked data list structures that was successful in applications such as automobile engine control. However, in time-synchronized applications like the cellular phone, the inability of flash memory to write during an erase suspend operation and the undeterministic maximum write and erase flash timing may have prevented EEPROM emulation using standard flash components in certain market segments. Time-critical applications, such as cellular telephones, must service system interrupts by providing access to processor code stored in flash while simultaneously supporting data writes to flash. For this reason, research has been undertaken by Intel and others to develop a simultaneous read operation while writing or erasing another flash memory partition (block). Specialized components have been proposed to support simultaneous read and write operations, but they incur from 10- to 20-percent increase in silicon die area due to redundant circuitry, and have not been manufactured in volume production. Although specialized circuits enable simultaneous read-while-write (RWW) operation, the added cost is less attractive in cost-critical, high-volume manufactured applications. New hardware-assisted suspend/resume circuitry with fast latency offer a technically-feasible approach to emulate simultaneous RWW operation without the cost impact of specialized circuits.

Regardless of specialized flash circuits, flash media management software is required to manage the larger (8 or 64 Kbytes) flash memory block partitions. This is true, since flash memory cannot be erased on the byte level common to memory such as EEPROM, but must be erased on a block granularity. The development of a flash memory manager requires a keen understanding of flash technology and data management methods. Fortunately, Intel has designed the necessary flash media manager, known as Flash Data Integrator (FDI), which handles variable length parameter storage, while utilizing hardware-assisted circuitry to emulate simultaneous code execution. This new method handles power loss recovery in the case of battery removal during data storage, providing a reliable EEPROM replacement.

This paper describes the hardware and software architecture necessary to emulate EEPROM memory in flash. Section 2.0 reviews the fundamentals of flash and EEPROM technology. Critical new timing parameters and hardware limitations are examined, along with a description of new hardware suspend to read/write capabilities common to many standard flash architectures. The software architecture for EEPROM emulation in flash memory is reviewed in Section 3.0. The software modules and features are also discussed in Section 3.0. Resource and system requirements are presented in Sections 4.0 and 5.0. Parameter cycling is characterized in Section 6.0, and power loss recovery techniques are described in Section 7.0. Section 8.0 reviews the flexibility of extending FDI to support enriched data storage and remote code updates.

2.0 MEMORY FUNDAMENTALS

2.1 Memory Architecture

Flash memory technology offers the electrical erasability of random access memory (RAM), and nonvolatility of read only memory (ROM) to retain information after power is removed. Unlike RAM, flash cannot be erased on a byte basis. Flash memory supports writing (programming), the processes of changing a logic “1” to a “0,” on a byte or word [double byte] basis. Certain flash memory components, including those from Intel, have the added capability to be programmed one bit (or multiple bits) at a time.

Erasing flash is the process of changing a logical “0” to a “1” on a block-by-block basis. Physical block partitioning is set by a fixed address range of the component (see Figure 1). Typical block sizes range from 8 Kbytes to 64 Kbytes for parameter and main blocks, respectively.

Flash memory stores data as charge on the floating gate of a single transistor as compared to other memory types that require additional components to hold a charge or the state of a latch circuit. As a result, flash has significant silicon area and cost advantages when compared with other memory types (see Table 1), offering a cost-effective means of storing data.

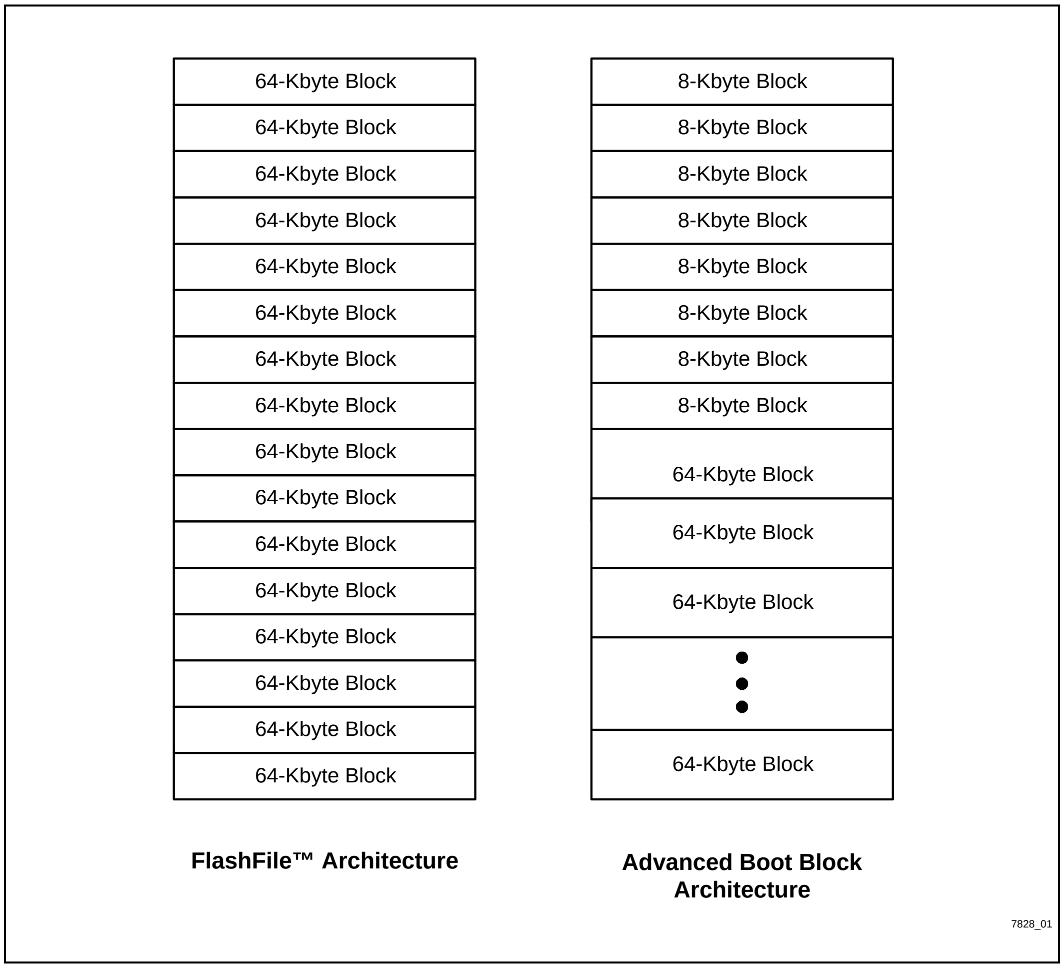


Figure 1. Intel's Standard Flash Memory Architectures

2.2 Program/Erase Timing

EEPROM supports byte alterability by rewriting a page, typically 16, 32 or 64 bytes. The system must wait 10 ms to allow time for the data to be written to the EEPROM cell in the background. This limits EEPROM write times from 157 µs to 625 µs/byte or 12.5 Kb/s to 49.7 Kb/s. Flash memory, on the other hand, supports data writes at a continuous 17 µs/byte (22 µs/word) typical 2.7 volt program time in the foreground, thereby supporting data write rates up to 710 Kb/s and reducing the amount of time a system spends writing data from 93% to 98%. Continuous data programming may be essential for streaming data packets such as short message service (SMS), fax, or digitized voice recording. This also doesn't account for any overhead time lost rewriting an entire page in EEPROM when only a single byte update is required—providing even a further reduction in data write time overhead.

Unlike flash, EEPROM does not require a block erase operation to free up space before data can be rewritten. This means that some form of software management is required to store data in flash. However, EEPROM technology is also limited to a maximum number of data writes [cycles] between 10,000 and 100,000. Flash memory, on the other hand, does not experience a device cycle until the block is erased. This means flash improves cycling reliability on the order of hundreds of times better than EEPROM technology. The details of parameter cycling is discussed in Section 5.0.

Table 2 compares the timing specifications of flash and EEPROM memory. Although the write performance of flash technology is fast compared to EEPROM, it is important to consider maximum program time. The time it takes to reliably store charge on the floating gate in flash memory is a function of process variation, temperature, voltage, and electron storage susceptibility. Under worst case conditions it may take as long as 170 µs to store a byte (200 µs for a word), as given by the specification t_{WHQV1} and t_{WHQV2}, respectively (see Table 3). The maximum time, however, does not occur across each of the flash cells, and is only realized in a single or few cells within a given address range. When writing a single byte or word, one should account for this maximum time t_{WHQV1} and t_{WHQV2}, respectively. However, when writing a page of data to flash memory the maximum write time is dependent on the page size and is given by the graph in Figure 2.

The erase time of the flash parameter and main blocks are given by the specification t_{WHQV3} and t_{WHQV4}, respectively (see Table 3). The distribution of erase time is similar to that of write times and are dependent on operating conditions and cycling. Erase times remain semi-constant for erase cycles less than 10,000. Above 10,000 cycles the erase time increases as illustrated in Figure 3. The manufacturer's specified cycling parameter is based on a given erase and program time. Flash can be reliably cycled beyond the specified value provided the design accommodates an increase in the erase and program time. For example, a flash device with specified 10K erase cycles can operate with 100K cycles with a typical erase time of 1.5 sec.

Fortunately, engineers need not design systems to wait the maximum specified values. Instead flash components commonly contain internal Status Registers that indicate when a program or erase operation is complete. By polling the internal register, the designer can determine when an operation is completed and the memory is available for another operation such as read.

Table 1. Die Area Comparison of Memory Technology

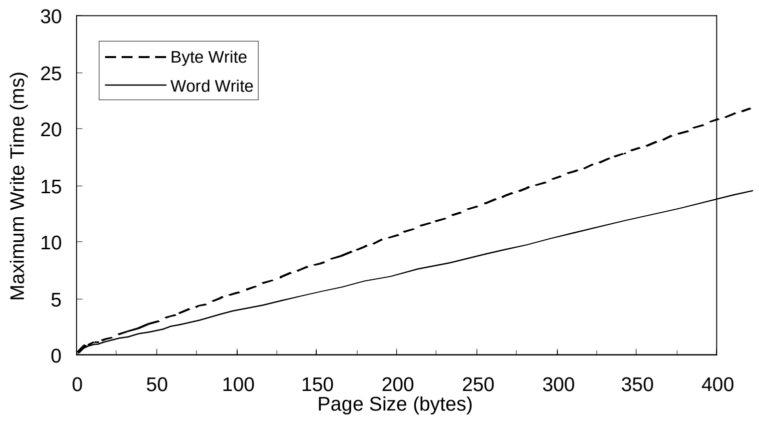
Features	Flash	DRAM	EEPROM	SRAM
Cell Components	1 Transistor	1 Transistor + 1 Capacitor	2 Transistor	4 Transistor + 2 Resistor
Cell Area ($\mu m^{2)}$ [0.4µ lithography]	2.0	3.2	4.2	22
Chip Area (mm^2) (16-Mbit density)	61	98	107	59 (1-Mbit Density)
Read Speed (ns)	80 (5V) 120 (3V)	60	150	<60

Table 2. Comparison of Flash and EEPROM

Features	Flash	EEPROM
Write Time (Typical)	10 µs / Byte (5V) 17 µs / Byte (3V)	3-5 ms / 32-Byte Page [47-157 µs/Byte]
Erase Time (Typical)	800 ms/8-KB Block (5V) 1000 ms/8-KB Block (3V)	NA
Internal Program/Erase Voltage	5V/12V (PSE) 5V/–10V (NGE)	5V/21V
Cycling	10–100K **Erase** Cycle/Block 10-300M Write Cycle/Byte[1]	10–100K **Write** Cycle/Byte

NOTES:

1. See Section 6.0.

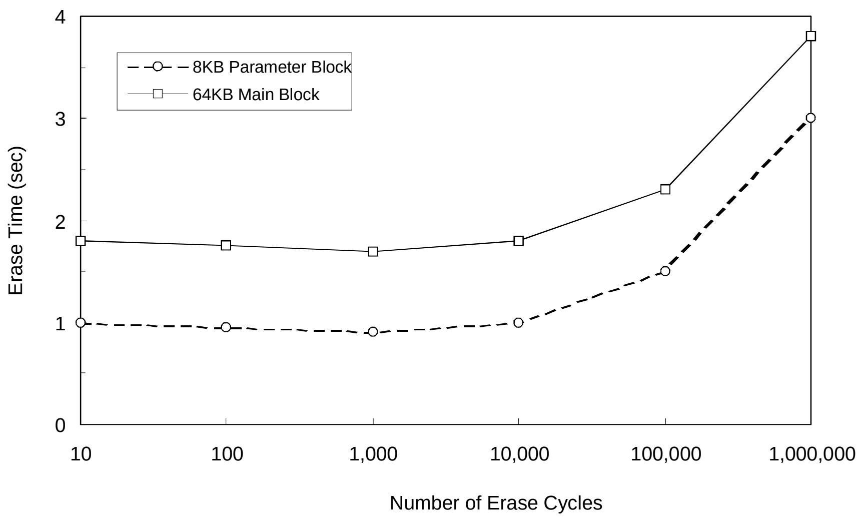


7828_02

NOTES:

Initial characterization; subject to change based on device validation.

Figure 2. Maximum Write Timing(1)
(V_{CC} = 2.7V–3.6V, T_A = –40°C to +85° C, 10K Cycles)



7828_03

Figure 3. Cycling Effects on Erase Time
(V_{CC} = 2.7V–3.6V, T_A = –40°C to +85°C)

Table 3. Flash Memory Erase and Program Timings(3,4)

Sym	Parameter	Notes	V_{PP} = 2.7V		V_{PP} = 12V		Unit
			Typ[1]	Max[5]	Typ[1]	Max[5]	
t_{WHQV1} t_{EHQV1}	Word Program Time	2	22	200	8	185	µs
t_{WHQV2} t_{EHQV2}	Byte Program Time	2	17	170	8	155	µs
t_{BWPB1}	Block Program Time (Word) (Parameter)	2	0.10	0.30	0.03	0.10	sec
t_{BWPB1}	Block Program Time (Byte) (Parameter)	2	0.16	0.48	0.07	0.21	sec
t_{BWMB2}	Block Program Time (Word) (Main)	2	0.80	2.40	0.24	0.80	sec
t_{BWMB2}	Block Program Time (Byte) (Main)	2	1.28	3.84	0.56	1.7	sec
t_{WHQV3} t_{EHQV3}	Block Erase Time (Parameter)	2	1	5.0	0.8	4.8	sec
t_{WHQV4} t_{EHQV4}	Block Erase Time (Main)	2	1.8	8.0	1.1	7.0	sec
t_{WHRH1} t_{EHRH1}	Word/Byte Program Suspend Latency Time to Read		6	10	5	6	µs
t_{WHRH2} t_{EHRH2}	Erase Suspend Latency Time to Read		13	20	10	12	µs

NOTES:

1. Typical values measured at T_A = +25°C and nominal voltages. Subject to change based on device characterization.
2. Excludes external system-level overhead.
3. These performance numbers are valid for all speed versions.
4. Characterized but not 100% tested.
5. Maximum values are based on typical process skews. Subject to change based on device characterization.

2.3 Specialized Flash RWW Circuits

Most currently available flash technology must complete a program or erase operation before code can be read from another memory block. Based on the maximum program/erase timing specifications of flash, there is a common misconception that EEPROM emulation can be supported only when the application can mask interrupts and allow a write or erase operation to complete. In time-synchronized applications with maximum latencies in the range of microseconds, such as a cellular phone, simultaneous operation may be difficult without some form of hardware assistance.

Many approaches to hardware assisted read-while-write (RWW) operation have been proposed for flash architecture (see Figure 4). One approach is to segment the standard memory array into separate physical partitions by duplicating the row and column (x/y) decoders, sense amplifiers and charge pump circuits—adding 12% to 17% to the silicon die area, and component cost. This form of hardware-assisted flash memory allows code to be read from one memory block, while a program or erase operation executes simultaneously in another block in the opposite physical partition.

Simultaneous read with background program/erase has higher peak power dissipation, but the total energy may be the same as the standard architecture. Segmented flash partitions further require that the data and code fit completely within the fixed partitions—making the selection of the partition size critical. If either code or data requirements exceed the maximum partition limit then simultaneous operation is no longer possible when data and code reside in the same partition, and the maximum suspend latency timing of the component must be considered. Although this method is attractive for EEPROM emulation, it does not offer the flexibility to support growing data needs.

On the other hand, a segmented architecture does minimize program/erase to read latency (see Table 5). This reduces the effect on system timing and may reduce testing if the design is time-critical.

An alternative approach is based on packaging two standard flash die into a single "dual-die" package. The "dual-die" approach supports simultaneous operation between the two die with the added flexibility that the size of the data or code partition can be changed to meet the needs of the application. Unfortunately, total peek memory system power is twice the power of a single standard flash component. Dual-die packaging, or two separate flash components, is attractive when the data requirements exceed 2 Mbits to 4 Mbits (256 KB to 512 KB). EEPROM emulation alone requires far less parameter storage needs, 8 Kbytes to 32 Kbytes, making a dual-die solution less cost-attractive.

A third approach to RWW is to combine EEPROM technology onto a two-transistor (2T) flash memory process. This approach eliminates the need for media management software, but has the disadvantage of increased memory system power and cost. Memory power dissipation can be as high as 200 mW compared to standard flash memory at 60 mW. The increase in die area necessary to support both memory technologies has an adverse impact on die yield and in turn product cost.

Yet another approach to hardware assistance is enhanced suspend circuits that allow program/erase operations to be suspended temporarily to read code from another partition. Suspend circuits allow time critical operations to be serviced without stalling the microprocessor (CPU). Unlike the other specialized RWW approaches, suspend circuits do not place limits on the code/data partition size, thereby increasing the flexibility and offering support as data storage needs grow. Suspend circuits do not increase the flash die size (cost), nor do they increase memory system power. The following section describes the suspend-resume operation in more detail.

Table 4 summarizes the comparison of the various RWW memory architectures.

2.4 New Hardware Assisted Suspend to Read/Write

Intel's 0.4µ ETOX™ V flash process technology components include two suspend commands; **Program and Erase Suspend**. Program and erase suspend mode allows system software to suspend both the word/byte program or block erase command in order to read from or write data to another block. Commands are written to the Command User Interface (CUI), connecting the microprocessor and the internal chip controller, using standard microprocessor write timings. Issuing a program or erase suspend command will begin to suspend a program/erase operation. The flash internal Status Register will indicate when the device reaches program/erase suspend mode. In this mode, the CUI will respond only to the Read Array, Read Status Register, Program, Program Resume, and Erase Resume commands. Flash specification t_{WHRH1} and t_{WHRH2} define the program and erase suspend latency, respectively (Table 3).

After a Program or Erase Resume is written to the flash memory, the flash device will continue with the program or erase process, respectively, (see Figures 5 and 6). The flash continues from the point where the suspend command was issued, eliminating the need to repeat the program or erase operation. The suspend to read/write operation provides a maximum latency of 10/20 µs, respectively, and allows system designers to emulate simultaneous RWW operation within the time constraints of the systems.

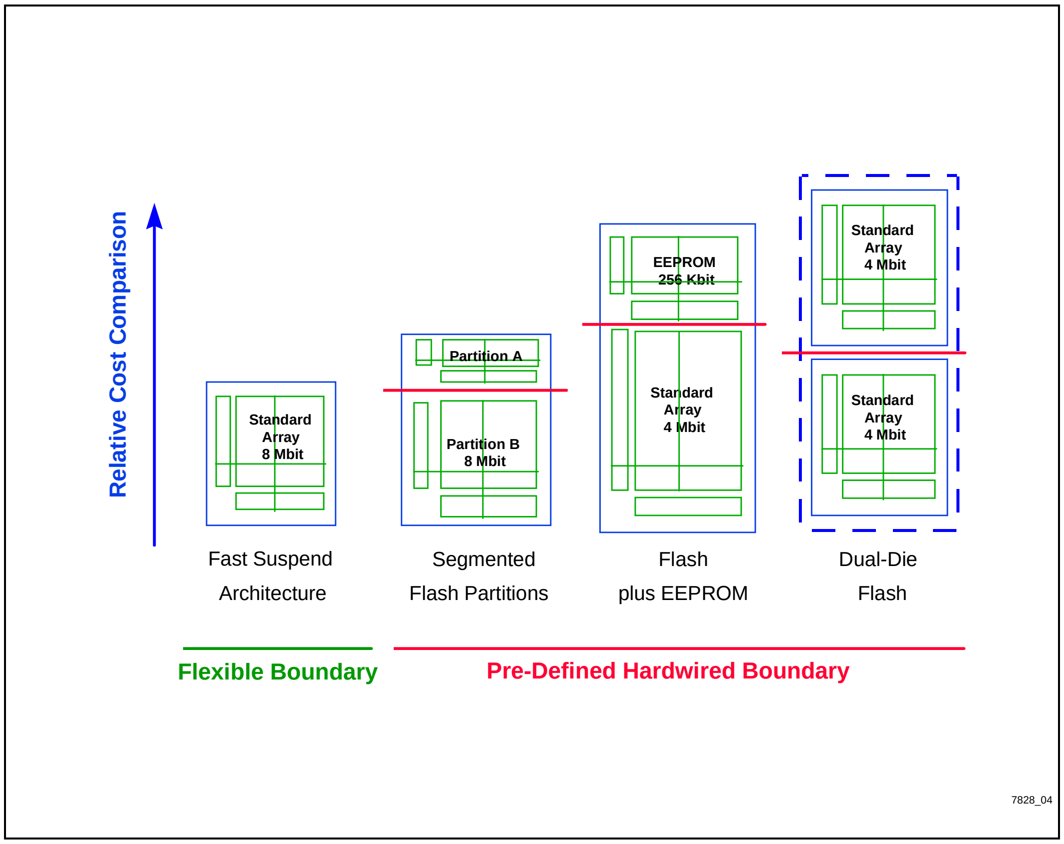


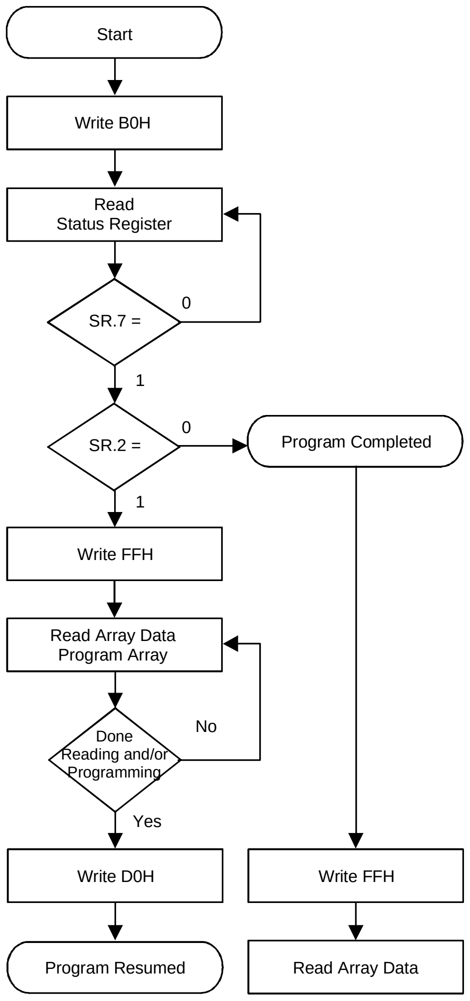
Figure 4. Comparison of Standard and Specialized Flash Memory Architectures for RWW

Table 4. Comparison of Standard and Specialized Flash Memory Architectures

Attribute	Advanced Boot Block Flash	Segmented RWW Flash	Dual-Die Flash	2T Flash Plus EEPROM
Die Size (mil/side) [8 Mbit, 0.4µ Lithography]	250	265-270	440 (2-Mbit data)	330 (256-Kbit EEPROM)
Min. Operating Voltage (Read/Write)	2.7V / 2.7V	2.7V / 2.7V	2.7V / 2.7V	4.5V / 4.5V
Max. Read Pwr.[1]	60 mW	60 mW	120 mW	200 mW
Max. Write/Erase Pwr.[1]	132/82.5 mW	132/132 mW	264/165 mW	200 mW
Max. Standby Pwr.[1]	165 µW	165 µW	330 µW	1,500 µW
Max. Latency to Read	10 to 20 µs	1 µs	120 ns	300 ns

NOTES:

1. Assumes maximum Vcc = 3.3V, and 5.0V for the 2T Flash plus EEPROM



Bus Operation	Command	Comments

Write	Program Suspend	Data = B0H
		Addr = X

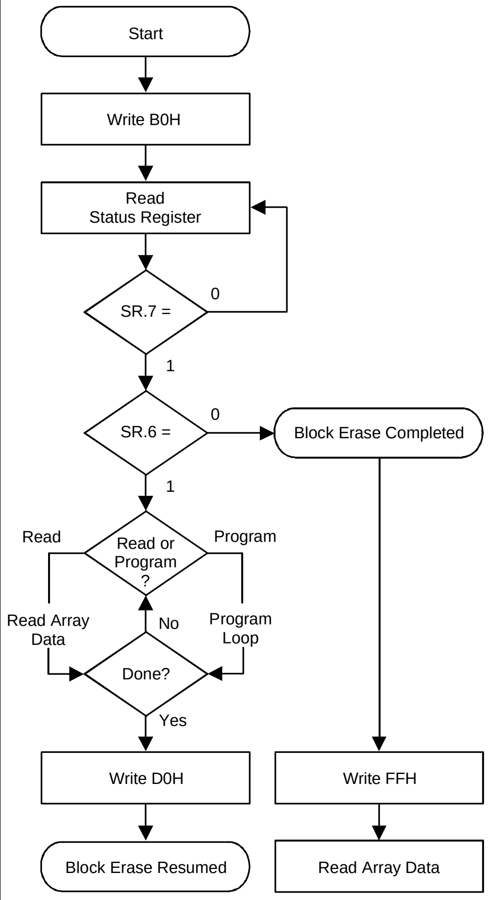
Read		Status Register Data Toggle CE# or OE# to Update SRD. Addr = X
Standby		Check SR.7 1 = WSM Ready 0 = WSM Busy
Standby		Check SR.2 1 = Program Suspended 0 = Program Completed

Write	Read Array	Data = FFH
		Addr = X

Read		Read array from block other than the one being erased.

Write	Program Resume	Data = D0H
		Addr = X

Figure 5. Program Suspend/Resume Flowchart



Bus Operation	Command	Comments

Write	Erase Suspend	Data = B0H
		Addr = X

Read		Status Register Data Addr = X
Standby		Check SR.7 1 = WSM Ready 0 = WSM Busy
Standby		Check SR.6 1 = Block Erase Suspended 0 = Block Erase Completed

Write	Erase Resume	Data = D0H
		Addr = X

7828_06

Figure 6. Block Erase Suspend/Resume Flowchart

3.0 FLASH DATA INTEGRATOR SOFTWARE STRUCTURE

Software is required to make all flash memory components, regardless of RWW circuitry, emulate an EEPROM. The approach of storing data on a nonvolatile media is well understood, but intricate due to the need to overcome the conditions described previously.

Intel has developed an open software architecture, known as the Flash Data Integrator (FDI), that enhances the flash technology. FDI allows the system designer to use a single low-cost flash memory component as a storage medium for both system code and data in real-time systems. This section describes the FDI flash media management software and reviews basic flash data management techniques.

3.1 Flash Data Integrator Functional Overview

The FDI architecture consists of three major subsystems; the Foreground Application Programming Interface (API), Background Manager, and Boot Code Manager.

System tasks and interrupts that need to store data, interface to the Foreground API functions. Through the API interface, commands that modify flash, and their corresponding data are queued by the system. The API commands such as open, close, read, write, and query are the interface between FDI and the system. The Background Manager reads commands/parameters from the queue, determines where the information should be stored, and performs any parameter storage or clean-up necessary while monitoring for interrupts. During initial system power-on the Boot Code Manager initializes the FDI control structures and performs any necessary power loss recovery.

Figure 7 illustrates the information flow between the system flash memory and SRAM. The system calls the foreground API function (0), with a command and data. The API function either, queues the command and data into SRAM for operations which modify flash (1W), or executes the command directly for commands which do not modify flash (1R). The FDI Background Manager (2W), executing out of flash memory, manages the queued tasks during available processor time. During a flash program or erase operation (4W), interrupts with vectors in flash are disabled and control is turned over to a small routine (less than 1 Kbyte) in SRAM (3W). This routine polls interrupts while monitoring progress of the program or erase operation. If a higher priority interrupt occurs, the polling routine suspends the flash memory program or erase operation, and allows the interrupt handler to then execute directly from flash memory. Upon completion of the interrupt routine, the flash program or erase operation is resumed by the SRAM polling routine. The Background Manager continues in this fashion until all events are handled and all necessary cleanup is complete.

A complete description of the FDI subsystems is provided in Chapter 3 of the *Flash Data Integrator (FDI) User's Manual*, order 297833.

3.2 EEPROM Parameter Types

Parameters stored in the EEPROM in a cellular phone can be characterized as either **factory, network or end-user data**. These parameter records vary in size and frequency of updates. For example, factory tuning data may be a long record (few hundred bytes) that is written to the EEPROM during the manufacturing process and may only be updated on an infrequent basis when the user brings the phone into a service center. On the other hand, the call timer parameter keeps track of the duration of a call and may be updated as often as every couple seconds during the process of the call. Table 5 lists the data parameter types commonly stored in the EEPROM of a cellular phone. The details of the parameter data structure are beyond the scope of this paper.

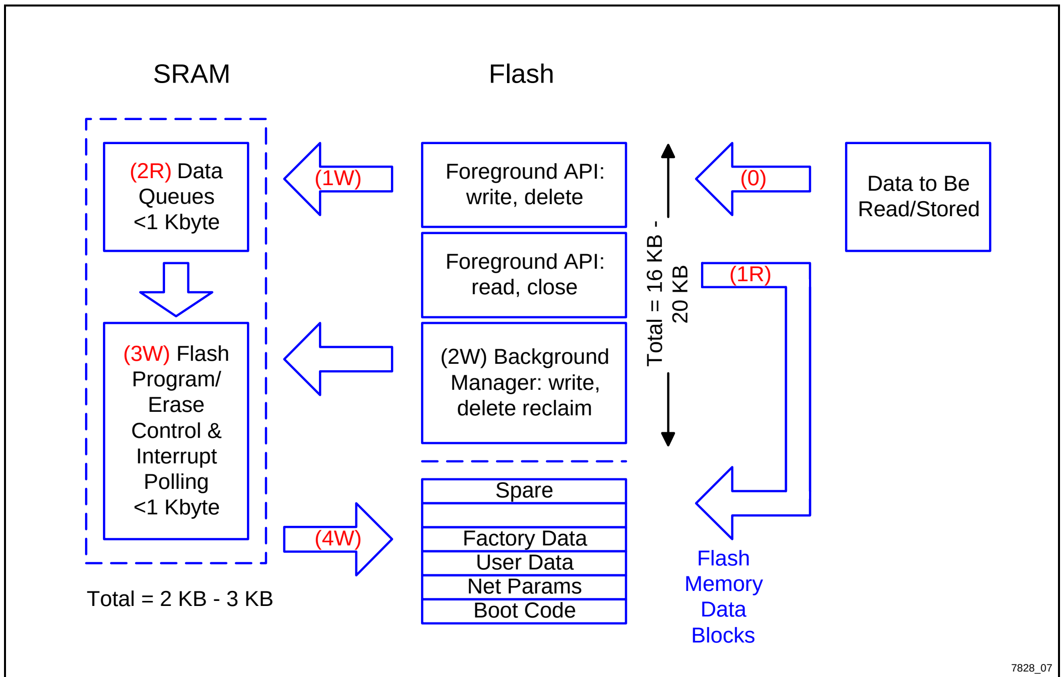
The frequency of parameter updates determines how often parameter blocks must be cleaned up [erased], to ensure free space is always available in flash for data writes. The write occurrence combined with the system time allocated for flash management and the timing parameters of the flash memory should be evaluated. Based on the low data write rate of cellular phone EEPROM data, flash memory with hardware assisted suspend/resume circuitry provides adequate timing to emulate an EEPROM and respond to system interrupts. It should be feasible to manage all flash memory program and erase operations during normal phone operation (e.g., during "dead" time while on a control channel or during a phone call). This maximizes the time the CPU remains in sleep mode.

3.3 Parameter Storage and Management

Unlike EEPROM, flash memory cannot be erased on a byte basis. By using software management techniques, data can be stored on a byte or variable length basis and flash erase operations can be completed using a suspend command to emulate byte alterability.

Data parameters are stored and tracked by software as virtual units within the physical boundaries of the flash block (see Figure 8). This is required whether specialized RWW circuits are available or not. Since a byte in flash may not be overwritten, an old occurrence of a parameter is marked “dirty” when the parameter is updated. The valid parameter is written to the next available memory location. The software media manager tracks the valid occurrences and controls access when requested by the system.

Parameters are stored until there is not enough “clean” space available in the block to insure new records can be written without over flowing the block. When this point is reached, the latest occurrence of each parameter is transferred to a clean [erased] block. Block header records associated with each parameter block indicates the status of the block. That is, information such as if the block is active [containing valid data], if the block is transferring data, or if the block is erased. After the valid parameters are transferred, the original block is marked for clean-up [erasing]. The parameter storage and management process if handled fully by FDI, and may be suspended by the system to write data provided free space is available.


SRAM
Flash
(2R) Data Queues <1 Kbyte
(3W) Flash Program/ Erase Control & Interrupt Polling <1 Kbyte
Total = 2 KB - 3 KB
(1W)
(4W)
Foreground API: write, delete
Foreground API: read, close
(2W) Background Manager: write, delete reclaim
Spare
Factory Data
User Data
Net Params
Boot Code
Total = 16 KB - 20 KB
(0)
(1R)
Data to Be Read/Stored
Flash Memory Data Blocks
7828_07

NOTES:
R = Read
W = Write

Figure 7. Flash Media Manager Software Data Flow

Table 5. EEPROM Data Parameters

Parameter Type	Size (Bytes)	Number	Amount (Bytes)	Occurrence
Factory	1–300	<10	~1,024	1–2 times/year
Network	5–20	25–50	~1,024	< few times/day
End-User	20–250	30–250	~6,144	Every few seconds during call

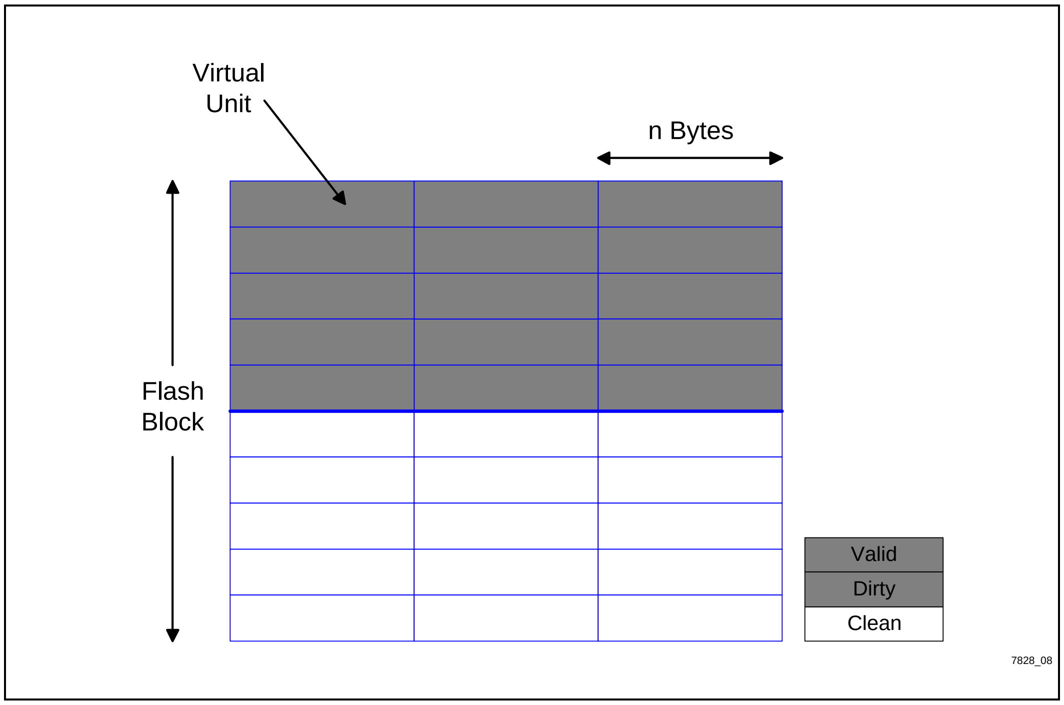


Figure 8. Flash Media Management

3.4 Read Latency

Unlike previous EEPROM emulation techniques that were based on a linked list approach, FDI uses a look up pointer to the parameter header to access the data. This provides uniform latency and simplifies system timing issues. (See *AP-604 Using Intel's Boot Block Flash Memory Parameter Blocks to Replace EEPROM*.)

3.5 Real Time Interrupt Support

To support real time interrupts, the flash management operations are suspended before servicing an interrupt. Upon completion of the interrupt routine, the flash operation is resumed until complete.

During a flash program or erase operation, a system interrupt hardware register is polled while waiting for the flash command to complete. If an interrupt occurs, the program or erase command is suspended and the interrupt handler is executed directly from flash memory after the maximum latency of the flash component (20 µs in the case of Intel’s Advanced Boot Block Flash). This eliminates the need to store interrupt handlers in SRAM. Upon completion of the interrupt, control is returned to flash program/erase routine to resume the operation.

Since non-specialized RWW flash components cannot be read from during a program or erase operation, a small (less than 1 Kbyte) software handler in the system’s static RAM (SRAM) is required. The SRAM and other system requirements are described in Section 5.0.

3.6 FDI Features

- Ability to easily integrate data and code into a variety of digital cellular environments. All areas of the code that require porting should be very limited and the code does not depend on the existence of non-ANSI ‘C’ libraries.
- Ability to suspend all data management activity when requested to execute code.
- Ability to resume data management activity following code execution.
- Ability to migrate the SW developed for standard flash architecture with hardware suspend/resume capability and flash components that support specialized RWW circuitry.
- Ability to support all EEPROM data storage requirements in the initial release.
- Ability to support enriched (larger) data types in future releases. These could include phone directories, audio recordings, or code updates. The initial release will not support these data types, however, the architecture is planned with this in mind.
- Ability to trade-off the features with the flash/RAM requirement.
- Low latency parameter/file read access.
- Support for variable parameter sizes without large overhead in media.
- Power-off recovery capabilities. An unexpected power loss should never corrupt or lose data. Replacement of old data with new should always provide the old data as a back-up until the new data can be guaranteed.
- Supports symmetrical block sizes through a portion of the flash component, but allows the block size to be definable at compile time.
- Flexible through use of defines, compile time options, or parameter options.

4.0 DEVELOPMENT RESOURCES

Although basic flash data management techniques may be well understood by the system software engineer, the work necessary to develop a reliable system takes significant time and resources. This effort has delayed many OEMs from fully emulating the EEPROM in flash. Fortunately, Intel’s FDI solution greatly reduces the OEM’s development effort.

Table 6 provides an estimate of the development resources required to integrate Intel’s FDI into an existing system compared to developing an internal media manager for a flash memory component using a specialized RWW flash component. Intel’s FDI may reduce the development time by 82%, allowing the OEM to bring the product to market faster.

Table 6. Projected EEPROM Emulation Development Time

Feature	Intel FDI	Internal Media Manager
FDI Definition	included	500 devl. hrs.
Flash Parameter Storage Management	included	600 devl.-hrs
Flash Storage Management Reclaim	included	600 devl.-hrs
EEPROM Interface	included	80 devl.-hrs
Power Loss Recovery	included	480 devl.-hrs
Flash Suspend/Resume Interface and Testing	included	N/A
API Integration	400 devl.-hrs	N/A
Total	400 devl.-hrs	2,260 devl.-hrs

Although additional time may be necessary to test future software changes that effect system timing. This may or may not be significant depending on the latency requirements of the system.

5.0 SYSTEM REQUIREMENTS

5.1 Random Access Memory Requirements

Some amount of system RAM (SRAM) is required to provide instructions during flash program and erase operation. The amount of RAM usage is dependent on the specific features needed. The size of this code is expected to be 2 Kbyte, 1 Kbyte for queue storage and less than 1 Kbyte of code. RAM should be used for queuing of events/data. The goal is to create a modular set of reference code, where the cellular phone OEM can pick and choose the various features needed in their product, and thus tailor the software to their specific product needs. Flash memory with specialized RWW circuitry does not have the requirement for available RAM as the component can provide instructions to control operation within the separate partition.

5.2 Flash Memory Requirements

Flash memory space will be necessary to store the Foreground and Background media manager program code. This is required for all flash memory types, standard and specialized. Intel's FDI media manager should require 16 Kbytes to 20 Kbytes of flash memory.

In addition, the flash memory component must include program and erase suspend commands (such as those in Intel's Advanced Boot Block components) or include specialized RWW circuitry as described in Section 2.3.

6.0 PARAMETER CYCLING

Intel's flash memory is specified to work over 100,000 erase cycles when operating over 0^{o}C to +70^{o}C, between 20,000 and 30,000 over the range of –25°C to +85°C, and 10,000 over the extended temperature range of –40^{o}C to +85^{o}C.

A cycle is defined as an erase operation, and not the number of data writes the device can support. For example, an 8-Kbyte block supports 8,192 byte writes before a single erase operation, one cycle, has completed. Therefore, parameter cycling is a function of the parameter size. This is important when determining how many parameter updates can be supported. Today, many OEMs limit writes to EEPROM due to 100K write cycling limit of EEPROM technology. Parameter updates in EEPROM over write the previous instance. The maximum life of the EEPROM is thereby limited to the update rate of the most frequently written parameter (e.g., call timer in the case of a digital cellular phone. Using flash for EEPROM emulation extends the effective number of data cycles.

The effective number of write cycles is dependent on the number of available parameter blocks and size of the parameter record and is given by:

$$Eff.\ Write\ \ Cycles = \frac{available\ \ bytes/block\ \ x\ \ no.\ \ blocks}{parameter\ \ record\ \ size} xMax.\ \ erase\ \ cycles/block$$

Assuming two 8-KB flash blocks are used to store a 5-byte record over an extended temperature range, and further assuming 5 Bytes/block status and 512 Bytes/block overhead results in:

$$Eff.\ \ Write\ \ Cycles = \frac{(8{,}192 - 5 - 512 Bytes/blk) x 2 Blks}{5 Bytes/\ parameter} x 10{,}000 cycles/blk\ .$$
$$= 30{,}700{,}000$$

This is a 300 times improvement over EEPROM memory that is limited to 100,000 write cycles. The same approach works for variable size records, where the effective write cycles are determined by the summation of the occurrences of the various records.

7.0 POWER LOSS RECOVERY

Power loss is handled in a reliable manner by adding a status field to the header of each data parameter block, as well as each parameter. The status field indicates that a parameter update has been initiated or the write was complete. If power is lost during a parameter update, the status is known when power is restored. Upon power recovery, the initiation process should check the status of each parameter. If the status indicates that a parameter update began but did not complete successfully, then the record can be marked invalid. The same process is used during clean-up operations when valid data is moved to a clean block. Because of the fast write capability of flash, critical parameters can be stored to flash sooner than an EEPROM component, thereby improving the robustness of the system.

To improve system power-on performance, the initiation process may be suspended, provided free space in flash is maintained.

8.0 ENRICHED DATA STORAGE AND REMOTE CODE UPDATES

Specialized RWW flash architectures with fixed size data partitions are limited in their ability to manage data that exceeds the partition size. Intel's FDI software is designed to manage a multiple number of memory blocks, offering a more flexible solution when combined with a component that is not limited by a physical partition, such as the Intel Advanced Boot Block flash memory.

FDI architecture supports extensions to manage enriched, streaming data types such as digitized voice, fax, company phone directories, and more. The architecture also enables the ability to remotely manage code modules stored in the main memory blocks.

9.0 CONCLUSION

A low-cost, flexible and reliable approach to EEPROM emulation in flash memory was presented for real time applications such as cellular phones. The approach is based on flash memory management software, known as the Flash Data Integrator (FDI), that emulates EEPROM functionality while enabling the flexibility for future data sotrage needs. This method reduces system cost, improves system write times by as much as 98%, supports data write rates up to 710 Kb/s, reduces memory system power by as much as 140 mW (compared with specialized RWW components), reduces development time by as much as 82%, and can increase parameter cycling 300 times over EEPROM memory. Power loss recovery techniques ensure data is not lost or corrupted in the event of power loss, eliminating the need for battery backed SRAM. EEPROM emulation in flash requires limited system resources depending on the needs and selected flash technology. Intel's FDI flash media management software and Advanced Boot Block flash memory offer a cost-effective, robust, reliable, and flexible solution to EEPROM replacement.

APPENDIX A
ADDITIONAL INFORMATION

Intel-Related Documents(1,2)

Order Number	Document/Tool
210830	*1997 Flash Memory Databook*
290580	*Smart 3 Advanced Boot Block 4-Mbit, 8-Mbit, 16-Mbit Flash Memory Family* Datasheet
292148	*AP-604 Using Intel's Boot Block Flash Memory Parameter Blocks to Replace EEPROM*
Contact Intel/Distribution Sales Office	*Intel Flash Data Integrator Functional Specification*(3)
Contact Intel/Distribution Sales Office	*Intel Flash Data Integrator System Environment Emulation Software Functional Specification*(3)

NOTE:

1. Please call the Intel Literature Center at (800) 548-4725 to request Intel documentation. International customers should contact their local Intel or distribution sales office.
2. Visit Intel's World Wide Web home page at http://www.Intel.com for technical documentation and tools.
3. This document is contained within the FDI Developer's Kit. Contact your local Intel or distribution sales office to obtain a copy.

Other Related Documents

Brian Dipert and Marcus Levy, *Designing with Flash Memory*, Annabooks, San Diego, CA, 1993.