FLASH MEMORY IN EMBEDDED SYSTEMS

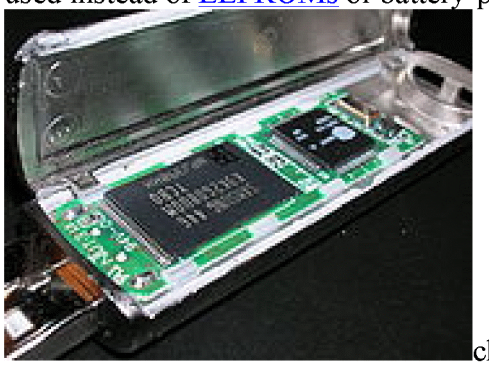
SUBMITTED BY:

POOJA
B.E. 2nd year(COE)
ROLL NO.100803069
Ph.no.9501216646

INDEX

INTRODUCTION

Flash memory is a non-volatile computer memory that can be electrically erased and reprogrammed. It is a technology that is primarily used in memory cards and USB flash drives for general storage and transfer of data between computers and other digital products. It is a specific type of EEPROM (Electrically Erasable Programmable Read-Only Memory) that is erased and programmed in large blocks; in early flash the entire chip had to be erased at once. Flash memory costs far less than byte-programmable EEPROM and therefore has become the dominant technology wherever a significant amount of non-volatile, solid state storage is needed. Example applications include PDAs (personal digital assistants), laptop computers, digital audio players, digital cameras and mobile phones. It has also gained popularity in the game console market, where it is often used instead of EEPROMs or battery-powered SRAM for game save data.



chip on left is flash memory

Reference: http://en.wikipedia.org/wiki/Flash_memory

ADVANTAGES

Since flash memory is non-volatile

- no power is needed to maintain the information stored in the chip.
- In addition, flash memory offers fast read access times (although not as fast as volatile DRAM memory used for main memory in PCs)
- better kinetic shock resistance than hard disks. These characteristics explain the popularity of flash memory in portable devices.
- Another feature of flash memory is that when packaged in a "memory card," it is enormously durable, being able to withstand intense pressure, extremes of temperature, and even immersion in water.[citation needed]

- Because erase cycles are slow, the large block sizes used in flash memory erasing give it a significant speed advantage over old-style EEPROM when writing large amounts of data.

Reference: http://en.wikipedia.org/wiki/Flash_memory

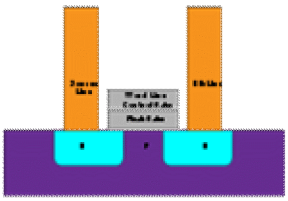
EMERGENCE OF FLASH

Intel saw the massive potential of the invention and introduced the first commercial NOR type flash chip in 1988.[3] **NOR-based flash** has long erase and write times, but ***provides full address and data buses, allowing random access to any memory location***. This makes it a suitable replacement for older ROM chips, which are used to store program code that rarely needs to be updated, such as a computer's BIOS or the firmware of set-top boxes. Its endurance is 10,000 to 1,000,000 erase cycles.[4] NOR-based flash was the basis of early flash-based removable media; CompactFlash was originally based on it, though later cards moved to less expensive NAND flash.

Toshiba announced **NAND flash** at the 1987 International Electron Devices Meeting. ***It has faster erase and write times, and requires a smaller chip area per cell, thus allowing greater storage densities and lower costs per bit than NOR flash;*** it also has up to ten times the endurance of NOR flash. However, the I/O interface of NAND flash does not provide a random-access external address bus. Rather, data must be read on a block-wise basis, with typical block sizes of hundreds to thousands of bits. This made NAND flash unsuitable as a drop-in replacement for program ROM since most microprocessors and microcontrollers required byte-level random access. In this regard NAND flash is similar to other secondary storage devices such as hard disks and optical media, and is thus very suitable for use in mass-storage devices such as memory cards. The first NAND-based removable media format was SmartMedia, and many others have followed, including MultiMediaCard, Secure Digital, Memory Stick and xD-Picture Card. A new generation of memory card formats, including RS-MMC, miniSD and microSD, and Intelligent Stick, feature extremely small form factors. ***For example, the microSD card has an area of just over 1.5 cm², with a thickness of less than 1 mm; microSD capacities range from 64 MB to 16 GB, as of August 2009.***[5] Reference: http://en.wikipedia.org/wiki/Flash_memory

Principles of operation

A flash memory cell.

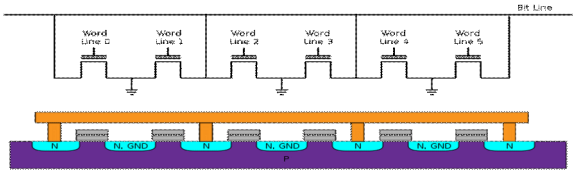


Flash memory stores information in an array of memory cells made from floating-gate transistors. In traditional single-level cell (SLC) devices, each cell stores only one bit of information. Some newer flash memory, known as multi-level cell (MLC) devices, can store more than one bit per cell by choosing between multiple levels of electrical charge to apply to the floating gates of its cells.

The floating gate may be conductive (typically polysilicon in most kinds of Flash memory) or non-conductive (as in SONOS Flash memory).[6]

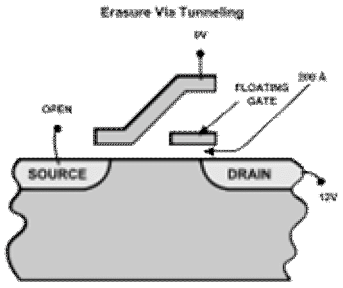
Reference: http://en.wikipedia.org/wiki/Flash_memory

NOR FLASH



NOR flash memory wiring and structure on silicon
In NOR gate flash, each cell has one end connected directly to ground, and the other end connected directly to a bit line.
This arrangement is called "NOR flash" because it acts like a NOR gate: when one of the word lines is brought high, the corresponding storage transistor acts to pull the output bit line low.
Reference: http://en.wikipedia.org/wiki/Flash_memory

Programming



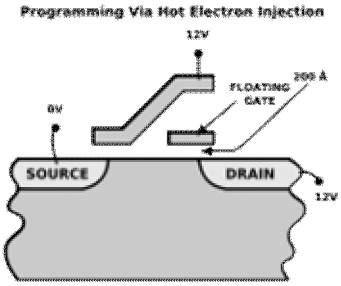
Programming a NOR memory cell (setting it to logical 0), via hot-electron injection

A single-level NOR flash cell in its default state is logically equivalent to a binary "1" value, because current will flow through the channel under application of an appropriate voltage to the control gate. A NOR flash cell can be programmed, or set to a binary "0" value, by the following procedure:

- an elevated on-voltage (typically >5 V) is applied to the CG
- the channel is now turned on, so electrons can flow from the source to the drain (assuming an NMOS transistor)
- the source-drain current is sufficiently high to cause some high energy electrons to jump through the insulating layer onto the FG, via a process called hot-electron injection

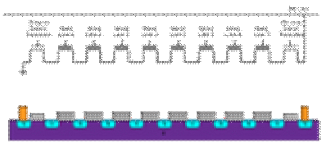
  Reference: http://en.wikipedia.org/wiki/Flash_memory

Erasing



Erasing a NOR memory cell (setting it to logical 1), via quantum tunneling.
To erase a NOR flash cell (resetting it to the "1" state), a large voltage *of the opposite polarity* is applied between the CG and source, pulling the electrons off the FG through quantum tunneling. Modern NOR flash memory chips are divided into erase segments (often called blocks or sectors). The erase operation can only be performed on a block-wise basis; all the cells in an erase segment must be erased together. Programming of NOR cells, however, can generally be performed one byte or word at a time.
Reference: http://en.wikipedia.org/wiki/Flash_memory

NAND flash



NAND flash memory wiring and structure on silicon

NAND flash also uses floating-gate transistors, but they are connected in a way that resembles a NAND gate: several transistors are connected in series, and only if all word

lines are pulled high (above the transistors' V_T) is the bit line pulled low. These groups are then connected via some additional transistors to a NOR-style bit line array.

To read, most of the word lines are pulled up above the V_T of a programmed bit, while one of them is pulled up to just over the V_T of an erased bit. The series group will conduct (and pull the bit line low) if the selected bit has not been programmed.

NAND flash uses tunnel injection for writing and tunnel release for erasing. NAND flash memory forms the core of the removable USB storage devices known as USB flash drives and most memory card formats available today.
Reference: http://en.wikipedia.org/wiki/Flash_memory

Limitations

Block erasure

One limitation of flash memory is that although it can be read or programmed a byte or a word at a time in a random access fashion, it must be erased a "block" at a time. This generally sets all bits in the block to 1. Starting with a freshly erased block, any location within that block can be programmed. However, once a bit has been set to 0, only by erasing the entire block can it be changed back to 1. In other words, flash memory (specifically NOR flash) offers random-access read and programming operations, but cannot offer arbitrary random-access rewrite or erase operations. A location can, however, be rewritten as long as the new value's 0 bits are a superset of the over-written value's. For example, a nibble value may be erased to 1111, then written as 1110. Successive writes to that nibble can change it to 1010, then 0010, and finally 0000. Filesystems built on NOR flash make use of this capability to represent sector metadata.[citation needed]

Although data structures in flash memory cannot be updated in completely general ways, this allows members to be "removed" by marking them as invalid. This technique may need to be modified for multi-level devices, where one memory cell holds more than one bit.
Reference: http://en.wikipedia.org/wiki/Flash_memory

Memory wear

Another limitation is that flash memory has a finite number of erase-write cycles. Most commercially available flash products are guaranteed to withstand around 100,000 write-erase-cycles, before the wear begins to deteriorate the integrity of the storage.[citation needed] The guaranteed cycle count may apply only to block zero (as is the case with TSOP NAND parts), or to all blocks (as in NOR). This effect is partially offset in some chip firmware or file system drivers by

- counting the writes and dynamically remapping blocks in order to spread write operations between sectors; this technique is called wear levelling.
- Another approach is to perform write verification and remapping to spare sectors in case of write failure, a technique called Bad Block Management (BBM).
- For portable consumer devices, these wearout management techniques typically extend the life of the flash memory beyond the life of the device itself, and some data loss may be acceptable in these applications.

For high reliability data storage, however, it is not advisable to use flash memory that would have to go through a large number of programming cycles. This limitation is meaningless for 'read-only' applications such as thin clients and routers, which are only programmed once or at most a few times during their lifetime.
Reference: http://en.wikipedia.org/wiki/Flash_memory

Low-level access

The low-level interface to flash memory chips differs from those of other memory types such as DRAM, ROM, and EEPROM, which support bit-alterability (both zero to one and one to zero) and random-access via externally accessible address buses.

While NOR memory provides an external address bus for read and program operations (and thus supports random-access); unlocking and erasing NOR memory must proceed on a block-by-block basis. With NAND flash memory, read and programming operations must be performed page-at-a-time while unlocking and erasing must happen in block-wise fashion.
Reference: http://en.wikipedia.org/wiki/Flash_memory

NOR memories

Reading from NOR flash is similar to reading from random-access memory, provided the address and data bus are mapped correctly. Because of this, most microprocessors can use NOR flash memory as execute in place (XIP) memory, meaning that programs stored in NOR flash can be executed directly without the need to first copy the program into RAM. NOR flash may be programmed in a random-access manner similar to reading. Programming changes bits from a logical one to a zero. Bits that are already zero are left unchanged. Erasure must happen a block at a time, and resets all the bits in the erased block back to one. Typical block sizes are 64, 128, or 256 KB.

Bad block management is a relatively new feature in NOR chips. In older NOR devices not supporting bad block management, the software or device driver controlling the memory chip must correct for blocks that wear out, or the device will cease to work reliably.

Apart from being used as random-access ROM, NOR memories can also be used as storage devices by taking advantage of random-access programming. Some devices offer read-while-write functionality so that code continues to execute even while a program or

erase operation is occurring in the background. For sequential data writes, NOR flash chips typically have slow write speeds compared with NAND flash.
Reference: http://en.wikipedia.org/wiki/Flash_memory

NAND memories

NAND flash architecture was introduced by Toshiba in 1989. These memories are accessed much like block devices such as hard disks or memory cards. Each block consists of a number of pages. The pages are typically 512[7] or 2,048 or 4,096 bytes in size. Associated with each page are a few bytes (typically 12–16 bytes) that should be used for storage of an error detection and correction checksum.

Typical block sizes include:

- 32 pages of 512 bytes each for a block size of 16 KB
- 64 pages of 2,048 bytes each for a block size of 128 KB
- 64 pages of 4,096 bytes each for a block size of 256 KB
- 128 pages of 4,096 bytes each for a block size of 512 KB

While reading and programming is performed on a page basis, erasure can only be performed on a block basis. Another limitation of NAND flash is data in a block can only be written sequentially. Number of Operations (NOPs) is the number of times the sectors can be programmed. So far this number for MLC flash is always one whereas for SLC flash it is four.[*citation needed*]

NAND devices also require bad block management by the device driver software, or by a separate controller chip. SD cards, for example, include controller circuitry to perform bad block management and wear leveling. When a logical block is accessed by high-level software, it is mapped to a physical block by the device driver or controller. A number of blocks on the flash chip may be set aside for storing mapping tables to deal with bad blocks, or the system may simply check each block at power-up to create a bad block map in RAM. The overall memory capacity gradually shrinks as more blocks are marked as bad.

When executing software from NAND memories, virtual memory strategies are often used: memory contents must first be paged or copied into memory-mapped RAM and executed there (leading to the common combination of NAND + RAM). A memory management unit (MMU) in the system is helpful, but this can also be accomplished with overlays. For this reason, some systems will use a combination of NOR and NAND memories, where a smaller NOR memory is used as software ROM and a larger NAND memory is partitioned with a file system for use as a nonvolatile data storage area.

NAND is best suited to systems requiring high capacity data storage. This type of flash architecture offers higher densities and larger capacities at lower cost with faster erase, sequential write, and sequential read speeds, sacrificing the random-access and execute in place advantage of the NOR architecture.

Reference: http://en.wikipedia.org/wiki/Flash_memory

Distinction between NOR and NAND flash

NOR and NAND flash differ in two important ways:

- the connections of the individual memory cells are different
- the interface provided for reading and writing the memory is different (NOR allows random-access for reading, NAND allows only page access)

It is important to understand that these two are linked by the design choices made in the development of NAND flash. An important goal of NAND flash development was to reduce the chip area required to implement a given capacity of flash memory, and thereby to reduce cost per bit and increase maximum chip capacity so that flash memory could compete with magnetic storage devices like hard disks

When NOR flash was developed, it was envisioned as a more economical and conveniently rewritable ROM than contemporary EPROM, EAROM, and EEPROM memories. Thus random-access reading circuitry was necessary. However, it was expected that NOR flash ROM would be read much more often than written, so the write circuitry included was fairly slow and could only erase in a block-wise fashion; random-access write circuitry would add to the complexity and cost unnecessarily.

Because of the series connection and removal of wordline contacts, a large grid of NAND flash memory cells will occupy perhaps only 60% of the area of equivalent NOR cells[13] (assuming the same CMOS process resolution, e.g. 130 nm, 90 nm, 65 nm). NAND flash's designers realized that the area of a NAND chip, and thus the cost, could be further reduced by removing the external address and data bus circuitry. Instead, external devices could communicate with NAND flash via sequential-accessed command and data registers, which would internally retrieve and output the necessary data. This design

choice made random-access of NAND flash memory impossible, but the goal of NAND flash was to replace hard disks, not to replace ROMs. Reference: http://en.wikipedia.org/wiki/Flash_memory

Flash file systems

Main article: Flash file system

Because of the particular characteristics of flash memory, it is best used with either a controller to perform wear-levelling and error correction or specifically designed flash file systems, which spread writes over the media and deal with the long erase times of NOR flash blocks. The basic concept behind flash file systems is: When the flash store is to be updated, the file system will write a new copy of the changed data over to a fresh block, remap the file pointers, then erase the old block later when it has time.

In practice, flash file systems are only used for "Memory Technology Devices" ("MTD"), which are embedded flash memories that do not have a controller. Removable flash memory cards and USB flash drives have built-in controllers to perform wear-levelling and error correction so use of a specific flash file system does not add any benefit. These removable flash memory devices use the FAT file system to allow universal compatibility with computers, cameras, PDAs and other portable devices with memory card slots or ports.
Reference: http://en.wikipedia.org/wiki/Flash_memory

Capacity

Multiple chips are often arrayed to achieve higher capacities for use in consumer electronic devices such as multimedia players or GPS. The capacity of flash chips generally increases exponentially because they are manufactured with many of the same integrated circuits techniques and equipment.

Consumer flash drives typically have sizes measured in powers of two (e.g. 512 MB, 8 GB).[*citation needed*] This includes SSDs as hard drive replacements, even though traditional hard drives tend to use decimal units. Thus, a 64 GB SSD is actually 64×1024^3 bytes. In reality, most users will have slightly less capacity than this available, due to the space taken by filesystem metadata.

In 2005, Toshiba and SanDisk developed a NAND flash chip capable of storing 1 GB of data using Multi-level Cell (MLC) technology, capable of storing 2 bits of data per cell. In September 2005, Samsung Electronics announced that it had developed the world's first 2 GB chip.[14]

In March 2006, Samsung announced flash hard drives with a capacity of 4 GB, essentially the same order of magnitude as smaller laptop hard drives, and in September 2006, Samsung announced an 8 GB chip produced using a 40 nanometer manufacturing process.[15]
Reference: http://en.wikipedia.org/wiki/Flash_memory

Transfer rates

Commonly advertised is the maximum read speed, NAND flash memory cards are much faster at reading than writing. As a chip gets worn out, its erase/program operations slow down considerably, requiring more retries and bad block remapping. Transferring multiple small files, smaller than the chip specific block size, could lead to much lower rate. Access latency has an influence on performance but is less of an issue than with their hard drive counterpart.

The speed is sometimes quoted in MB/s (megabytes per second), or as a multiple of that of a legacy single speed CD-ROM, such as 60x, 100x or 150x. Here 1x is equivalent to 150 kilobytes per second. For example, a 100x memory card gives 150 KB/s × 100 = 15,000 KB/s = 14.65 MB/s.

Applications in embedded systems

Serial flash

Serial flash is a small, low-power flash memory that uses a serial interface, typically SPI, for sequential data access. When incorporated into an embedded system, serial flash requires fewer wires on the PCB than parallel flash memories, since it transmits and receives data one bit at a time. This may permit a reduction in board space, power consumption, and total system cost.

There are several reasons why a serial device, with fewer external pins than a parallel device, can significantly reduce overall cost: Reference: http://en.wikipedia.org/wiki/Flash_memory

Firmware storage

With the increasing speed of modern CPUs, parallel flash devices are often much slower than the memory bus of the computer they are connected to. Because of this, it is often desirable to shadow code stored in flash into RAM; that is, the code is copied from flash into RAM before execution, so that the CPU may access it at full speed. Device firmware may be stored in a serial flash device, and then copied into SDRAM or SRAM when the device is powered-up.[18] Using an external serial flash device rather than on-chip flash removes the need for significant process compromise (a process that is good for high speed logic is generally not good for flash and vice-versa). Once it is decided to read the firmware in as one big block it is common to add compression to allow a smaller flash chip to be used. Typical applications for serial flash include storing firmware for hard drives, Ethernet controllers, DSL modems, wireless network devices, etc. Reference: http://en.wikipedia.org/wiki/Flash_memory

Flash memory as a replacement for hard drives

Main article: Solid-state drive

An obvious extension of flash memory would be as a replacement for hard disks. Flash memory does not have the mechanical limitations and latencies of hard drives, so the idea of a solid-state drive, or SSD, is attractive when considering speed, noise, power consumption, and reliability.

There remain some aspects of flash-based SSDs that make the idea unattractive. Most important, the cost per gigabyte of flash memory remains significantly higher than that of platter-based hard drives. Although this ratio is decreasing rapidly for flash memory, it is not yet clear that flash memory will catch up to the capacities and affordability offered by platter-based storage. Still, research and development is sufficiently vigorous that it is not clear that it will not happen, either.

There is also some concern that the finite number of erase/write cycles of flash memory would render flash memory unable to support an operating system. This seems to be a

decreasing issue as warranties on flash-based SSDs are approaching those of current hard drives.[19][20]

As of May 24, 2006, South Korean consumer-electronics manufacturer Samsung Electronics had released the first flash-memory based PCs, the Q1-SSD and Q30-SSD, both of which have 32 GB SSDs.[21] Dell Computer introduced the Latitude D430 laptop with 32 GB flash-memory storage in July 2007—at a price significantly above a hard-drive equipped version.[*citation needed*]

At the Las Vegas CES 2007 Summit Taiwanese memory company A-DATA showcased SSD hard disk drives based on Flash technology in capacities of 32 GB, 64 GB and 128 GB.[22] Sandisk announced an OEM 32 GB 1.8" SSD drive at CES 2007.[23] The XO-1, developed by the One Laptop Per Child (OLPC) association, uses flash memory rather than a hard drive. As of June 2007, a South Korean company called Mtron claims the fastest SSD with sequential read/write speeds of 100 MB/80 MB per second.[24]

Rather than entirely replacing the hard drive, hybrid techniques such as hybrid drive and ReadyBoost attempt to combine the advantages of both technologies, using flash as a high-speed cache for files on the disk that are often referenced, but rarely modified, such as application and operating system executable files. Also, Addonics has a PCI adapter for 4 CF cards,[25] creating a RAID-able array of solid-state storage that is much cheaper than the hardwired-chips PCI card kind.

The ASUS Eee PC uses a flash-based SSD of 2 GB to 20 GB, depending on model. The Apple Inc. Macbook Air has the option to upgrade the standard hard drive to a 128 GB Solid State hard drive. The Lenovo ThinkPad X300 also features a built-in 64 GB Solid State drive. **Reference: http://en.wikipedia.org/wiki/Flash_memory**

ADVANCEMENTS IN FLASH TECHNOLOGY

Compared to the earlier implementations, today's Flash memories usually require less complex programming algorithms and they are now divided into several sectors. The benefit of having sectors is that the Flash memory is sector-erasable, meaning you can erase one sector at a time. In the past, erase commands erased the entire memory chip - therefore to keep a working copy of that data during run-time, an application required additional memory.

Since Flash memory is integrated on-chip with microcontrollers, its usage became even easier. Having Flash memory and a microcontroller on the same chip opened up the opportunity to take advantage of the "additional intelligence".With the 89C51Rx2 microcontrollers, Philips Semiconductors created a variety of 8051 derivatives with on-chip Flash memory that take best advantage of having a combination of microcontroller and Flash memory on one chip. Philips did this by providing an additional ROM area containing code for handling the Flash programming. The code does not only provide functions to erase or program the Flash memory, it also provides boot code - even with a

completely erased Flash, the chip can still execute this boot code and accept inputs via the serial port.

Being ROM, this code area is not erasable; applications can rely on it as always being there. It can be used for recovery of a system, by downloading new code into the Flash memory via the serial port. Because of this feature, this code is also referred to as "boot loader". Reference:
http://www.esacademy.com/faq/docs/flash/

ISP vs. IAP

When it comes to re-programming Flash memory that is soldered down to a PCB (either integrated into the microcontroller or external), there are two programming methods: ISP and IAP.

ISP: In-System Programming

ISP allows for re-programming of a Flash memory device while it is soldered into the target hardware. However, the application needs to be stopped during the re-programming process. Usually, ISP requires that a service technician manually starts the re-programming procedure by halting the application and setting it into a special boot and/or programming mode. Only after programming is completed, the application can be restarted.

In the Philips 89C51Rx2 series, ISP is implemented with the boot loader. The chip is set to ISP mode either by driving pin PSEN high externally right after a hardware reset or by software. When in ISP mode, the 89C51Rx2 accepts Flash-programming commands via the serial interface.
Reference: http://www.esacademy.com/faq/docs/flash/

IAP: In-Application Programming

IAP allows for re-programming of a Flash memory device while it is soldered into the target hardware and while the application code is running. With IAP it is possible to implement applications that can be re-programmed remotely without the need of a service technician to actually be present.

In general, IAP can always be realized with external Flash memory, where microcontroller and memory are separated components. This is true as long as there is some additional code memory available ou[illegible] which the microcontroller can execute code, while the Flash memory is re-programmed.

With on-chip Flash, IAP is only possible if supported by the microcontroller. The Philips 89C51Rx2 parts support IAP also via the boot loader. The application code can call functions in the boot loader area by loading parameters into the

registers R0, R1 and DPTR and then calling a specific address in the boot loader. To make these functions easier to use, the ***Embedded Systems Academy*** provides a C library supporting all boot loader functions. This allows erasing and programming directly from the C level,simply by calling C functions.
Reference: http://www.esacademy.com/faq/docs/flash/

Self recovery

For any system using Flash memory, the worst-case scenario includes a system failure, crash or power outage, while the Flash is being erased or re-programmed.

For some applications, a recovery via ISP might be acceptable. A service technician capable of doing the recovery manually might be at hand. However, it's easy to imagine that in truly remote systems (for example marine buoys), a more sophisticated self-recovery mechanism, not involving any person, is more desirable.

As an example, let's assume we have a network of buoys covering an ocean and collecting data. Each buoy can send and receive data via radio. After a year, the project team working on the data realizes that they can optimize their research with a few modifications to the code executed in the controllers in the buoys.

When uploading new code, the system should be stable enough to recover by itself, even if the system has a reset (or power failure) just right after erasing the Flash memory. Sending a maintenance crew to the buoy might take weeks and cost a significant amount of money.

To ensure the recovery, it's vital to any application that the code starting at the reset vector (starting at location 0) never gets erased or re-programmed. Furthermore, all the code essential to the application must also be in code areas that never get erased. As a minimum, this must include all the code handling the IAP part of the program. In our case, that also includes all the radio communication routines.

In addition, routines are needed that can detect and decide if t[illegible] the re-programmable part of the application actually conta[illegible]d code. A jump or call to these program areas should surely only be exec[illegible]ed after ensuring that real code is at the expected location.

One way to implement this is using checksums. If ea[illegible] code segment in an area that can be modified / re-programmed by the appli[illegible]ion has a predefined checksum (which can be achieved by adding fill-by[illegible]s that ensure a certain checksum), then the code testing routine only nee[illegible] to calculate and match the checksum to decide if a piece of code is valid or no[illegible] Depending on the level of

security needed, several checksums can be implemented.

For our example with the buoys, the entire re-programming procedure could work as follows: The controller receives the radio command to erase the Flash memory with the data collection routines. It does execute the command, but never touches the reset sector. It now receives the new code via radio and programs it into the Flash memory.

In the worst case, the system has a malfunction right now, before the programming is complete. Let's assume the system restarts after a while: the code in the reset sector is still present and executes. It recognizes a checksum failure in the code sector handling the data collection. Using the radio link it transmits this status and waits to receive new re-programming commands.

Reference: http://www.esacademy.com/faq/docs/flash/

USES IN SATELLITES

- ***Being reprogrammable and having a large no. of write cycles,flash memory can be used in satellites for data transmission easily and at a rapid speed.***
- ***A combination of NOR and NAND flash memory can be used as NOR allows easy access to data stored and NAND provides large amount of space for data storage.***
- ***Self recovery is another important feature that makes flash of great use in satellite systems as if a satellite gets out of reach or loses contact with the ground station ,the data can be recovered without any loss.Same can be done if a satellite crashes during launch.***
- ***ON-CHIP FLASH with microcontrollers provides multiple functions like storing configuration data after a power-up or power down cycle***

and thus decreases parts count of embedded applications in satellites.

- ***Flash memory requires less power and is less costly as compared to EEPROM,hence it is a better choice for data storage and transmission in satellites.***

Summary

Flash technology is constantly changing, providing faster program and erase cycles, a bigger number of guaranteed erase and re-program cycles and longer data retention. Flash technology put on-chip with microcontrollers has now reached the point, where the in-application usability greatly improved.

In the past, storing of configuration data that stays available after a power-down and power-up cycle required an additional EEPROM or other storage device. Today, this functionality can be provided by on-chip Flash, further decreasing the parts count of embedded applications: an additional external EEPROM or SRAM might not be required anymore.