

NAND Flash Memory and Code Storage

Historically, NOR flash memory has been the storage medium of choice for housing executable code, such as power-on boot loaders and BIOS software. The relatively speedy access times of NOR and the ability to linearly address the flash within a platform's memory space, encourage its use for code storage applications.

NAND flash, which has somewhat slower access times, and a non-linear interface, has typically not been used for storing code – it has instead thrived as a disk-like repository for application data. Its higher density and superior write performance make it well suited for platforms that require large amounts of data storage.

Today, however, the traditional uses of these flash technologies is being revisited, as computer manufacturers scramble to reduce costs. With NAND priced at less than one half the cost of NOR, system designers are pioneering new methods of using NAND for code as well as data storage.

This article reviews the basic differences between NOR and NAND flash memory technologies and explains how these have commonly been used in system design. It also highlights some of the techniques that can be used to execute – and even boot – code that is stored in NAND flash.

NOR Flash

Housing system software such as motherboard boot code or a video adapter's BIOS, has typically been the domain of NOR flash chips. Since its inception in the mid eighties, NOR flash has been a welcome change from EPROMs – the ability to dynamically update code has been a godsend for the deploying of bug fixes and firmware upgrades.

To facilitate its use in these applications, NOR chips were originally packaged as pin-for-pin compatible EPROM replacements. This was made possible because NOR uses the same address lines and control signals within its memory-like interface. This addressing allows the system to execute code directly from NOR flash, just as it can from EPROM or SDRAM (accesses to NOR can be up to ten times slower than SDRAM however).

Depending on the application, the performance of this direct code execution – also known as execute-in-place or XIP – may be sufficient particularly if only small amounts of code are involved. Often however, SDRAM shadowing is used: code contained in NOR flash is copied into higher speed SDRAM where it is then executed. (This technique has been supported for many years by PC chipsets to enhance system performance.) Sometimes, as part this scheme, code is stored in a compressed format, and uncompressed into SDRAM prior to execution.

Even today, with NOR flash chips as large as 16MB being used to store entire operating system images such as Windows CE, SDRAM shadowing is still employed on platforms where performance is an important design consideration.

In devices such as basic cellular phones where code size is relatively small or where there's no performance requirement beyond what can be achieved with direct code execution, NOR flash chips have been the undisputed design choice. Further contributing to the success of NOR in these environments is the fact that there's typically no requirement for storing large amounts of data (although this requirement is changing for the emerging 2.5G and 3G cell technologies).

As a platform's need for data storage increases, the desirability of NOR flash diminishes. Its original design which favors relatively fast random access to the flash for the purpose of executing code, doesn't result in an internal management of flash that lends itself to swift write performance. The large erase blocks typically associated with NOR, the overhead that their erasure requires, and lengthy programming times, limit its use for significant data storage applications.

NAND Flash

NAND flash was developed a few years after the introduction of NOR. Its design emphasizes increased write performance, higher density, and lower cost. Write performance improvements are achieved through a sector-oriented management of the flash and smaller sized erase blocks. Higher densities are attained by using a smaller flash cell size. The scaled back cell size, and the reduced pin count of NAND's I/O interface, contribute to a lower manufacturing cost.

On host systems, NAND is readily connected via the same simple interface commonly found in many computer peripherals. Instead of requiring full data and address bus lines, data and commands are multiplexed onto eight I/O lines. This consumes only three addresses in the host's I/O space. Using this conduit into the flash, a register set can be read and written to dispatch commands and access data.

Each time a read or write command is issued, the amount of data transferred is always one page consisting of 528 bytes (512 bytes of data and 16 bytes typically containing ECC). This is the marked distinction between the NAND and NOR interfaces: data transfers with NAND flash always involve the issuing of a command and the accessing of a page's worth of data. There is no ability to randomly access a few selected bytes, and it's not possible to linearly address the flash, thus directly executing code from NAND is out of the question: SDRAM shadowing must always be employed for code storage applications.

With NAND, data from each page is always transferred one byte at a time. In newer generations of NOR flash, full word or double word transfers can be used as well to increase throughput. This contributes to NAND's somewhat slower data retrieval times. Another factor is the startup overhead incurred when each command is issued (each byte

of data must be internally transferred between the memory cell array and the register interface). On a 512 Mbit flash chip, this consumes about 25 microseconds – although this amount, when amortized across an entire page, becomes negligible, and raw read performance benchmark numbers become more in line with those of NOR.

NAND's write performance however, drastically distances it from NOR. When transferring large amounts of data – particularly data that crosses over erase block boundaries – the disparity tends to be dramatic. This is one of the reasons that NAND has been so successful in digital cameras and other platforms that require disk-like storage.

NAND's popularity also stems from its higher storage capacities and significantly lower costs.

Capacities available with NAND can reach 64MB with 512 Mbit binary (single-level cell) chips and doubled to 128MB with MLC (multi-level cell) 1Gbit versions of the flash. (Two 1Gbit dies can also be stacked in a single TSOP to achieve 256MB.) Respective NOR capacity maximums are 16MB based on 128Mbit binary chips and 32MB with 256 Mbit density MLC chips.

Price per megabyte tends to widely separate the two technologies as well. At the time of this writing, the device normalized unit cost per megabyte for NAND was less than one half that of NOR.

Data and Code Storage

This pricing differential is prominently showing up on the radar screens of system designers. In today's world of ultra-low priced portable electronics devices, where shaving every last cent off a product's cost has become the mantra, NAND is increasingly being used for purposes that transcend its traditional role as a data-only storage medium. Increased code and data storage requirements are also fueling this shift.

Examples of this are platforms such as high end set top-boxes and next generation cellular phones. Their sophistication brings with it increased requirements for disk-like data storage and the need for larger code space. A single high density NAND part furnishes more than adequate capacity to accommodate both of these functions. Using this configuration – even after adding additional SDRAM with which to shadow the code – results in reduced system costs.

This sounds great, but it does beg a chicken-and-egg-like question: How does the code get initially moved from NAND flash into the SDRAM? After all, at power on, system software needs to be directly executed to initialize the platform.

If the platform's processor provides support for embedded boot code, transferring the code from NAND into SDRAM is a trivial matter. Processors which can readily accomplish this include Motorola's MC9328MX1 and Neo Magic's (formerly Linkup

Systems) NMS7210A. The NMS7210A has a special provision for booting from an inexpensive serial PROM which can house the code that performs the transfer.

In environments where boot options aren't as flexible, a few hundred gates of programmable logic can be cleverly employed to do the job. The exact approach here will vary depending on the CPU, but the strategy is basically the same: gain control by decoding the power on execution address; issue a command to the NAND flash to retrieve the boot code; load it into SDRAM and transfer control to the code, so that it starts executing. With some processors such as the PowerPC, this is particularly easy, as addresses and signals can be manipulated such that code fetches at boot addresses actually pull data from the flash.

Conclusion

As we've seen, NAND flash is playing a broader role in today's platforms. While data storage requirements continue to call for NAND-based designs, NAND's use for code storage – even when booting is involved – is becoming increasingly popular as well. NAND flash accommodates both of these functions, while significantly reducing system costs.