25 juillet 2007

Flash memory 101: An Introduction to NAND flash

As the price of NAND flash tumbles, it become a viable replacement for a hard drive.

By Jim Cooke, Micron Technology Inc.

CommsDesign
mars 20, 2006

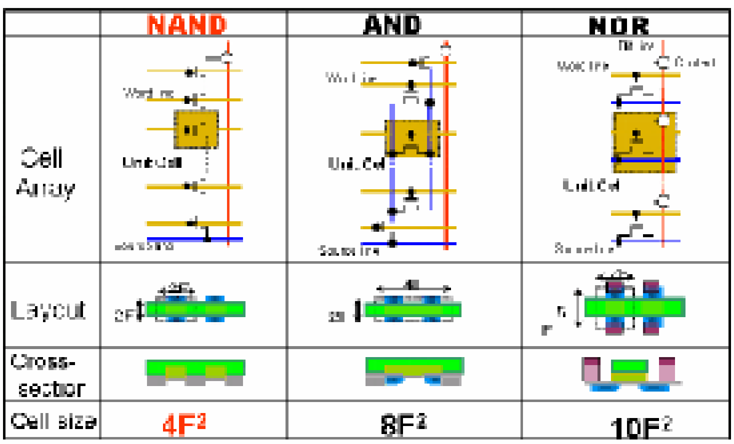
For many consumer audio and video products, NAND flash memory is a better storage choice than a hard drive, especially in low-capacity applications (4 Gbytes or less). As the quest continues for lower power, lighter, more robust products, NAND is proving to be very attractive.

The NAND flash array is grouped into a series of 128-kbyte blocks, which are the smallest erasable entity in a NAND device. Erasing a block sets all bits to "1" (all bytes to FFh). Programming is necessary to change erased bits from a 1 to a 0. The smallest entity that can be programmed is a byte. Some NOR Flash memory can perform read-while-write operations *(Fig. 1)*. Although NAND can't perform read and write simultaneously, it can accomplish this at a system level using a method called shadowing, which has been used on PCs for years by loading the BIOS from the slower ROM into the higher-speed RAM.

NAND efficiencies are due to the lack of metal contacts in the NAND string. NAND flash cell size is smaller than NOR, $4F^2$ verses $10F^2$, due to the fact that NOR cells require a separate metal contact for each cell.

NAND is similar to a hard-disk drive. It's sector-based (page-based) and suited for storing sequential data such as pictures, audio, or PC data. Although random access can be accomplished at the system level by shadowing the data to RAM, doing so requires additional RAM storage. Also, like a hard disk, NAND devices have bad blocks, and require error-correcting code (ECC) to maintain data integrity.

Due to the decrease in die area resulting from the small cell size, NAND provides the larger capacities required for today's low-cost consumer market. NAND flash is used in almost all removable memory cards. NAND's multiplexed interface provides a similar pin-out for all recent devices and densities. This pin-out lets designers use small densities and migrate to larger densities without any hardware changes to the pcb.



1. The figure shows a comparison of various flash cells.

NAND vs. NOR

NAND's advantages are fast write (program) and erase operations, while NOR's advantages are random access and byte write capability *(Fig. 2)*. NOR's random access ability allows for execute in place (XIP) capability, which is often a requirement in embedded applications. The disadvantages for NAND are slow random access, while NOR is hampered by are slow write and erase performance. NAND is better suited for file applications. However, more processors include a direct NAND interface and can boot directly from NAND (without NOR).

Characteristic	NAND Flash MT29F2G08A	NOR (Q-Flash) MT28F128J3
Random access read	25µs (first byte) .03µs each for remaining 2111 bytes	.12µs
Sustained read speed (sector basis)	23 MB/s (x8) or 37 MB/s (x16)	20.5 MB/s (x8) or 41 MB/s (x16)
Random write speed	~300µs/2112 bytes	180µs/32 bytes
Sustained write speed (sector basis)	5 MB/s	.178 MB/s
Erase block size	128KB	128KB
Erase time per block (typ)	2ms	750ms

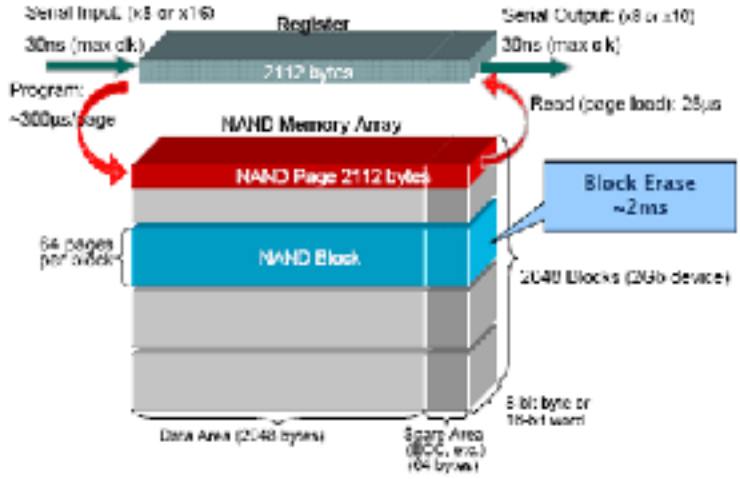
2. The random access time for NOR flash is specified at 0.12 Ms, whereas NAND random access is significantly slower for the first byte. Once the initial access is made, however, the remaining 2111 bytes are shifted out of NAND at only 0.03 Ms/byte.

The real benefits for NAND are faster program and erase times, as NAND provides over 5 Mbytes/s of sustained write performance. The block erase times are an impressive 2 ms for NAND versus 750 ms for NOR. Clearly, NAND has several significant positive attributes. However, it's not well-suited for direct random access.

NOR flash requires around 41 I/O pins for a 16-bit device, while NAND devices requires only 24 pins for a comparable interface. This pin savings comes from NAND's multiplexed command, address, and data bus. A benefit of the multiplexed interface is that larger NAND devices can be supported using the same hardware design and pcb. Because the common TSOP-1 package has been used for many years, this feature lets customers migrate to higher-density NAND devices on the same pcb. Another advantage of NAND is evident in it's packaging options: NAND offers a monolithic 2-Gbit die or can support up to four stacked die, allowing an 8-Gbit device in the same TSOP-1 package. This enables one package and interface to support higher densities in the future.

NAND basic operation

The 2-Gbit NAND device is organized as 2048 blocks, with 64 pages per block *(Fig. 3)*. Each page has 2112 bytes total, comprised of a 2048-byte data area and a 64-byte spare area. The spare area is typically used for ECC, wear-leveling information, and other software overhead functions, although it's physically no different from the rest of the page. NAND devices are offered with either an 8- or 16-bit interface. Host data is connected to the NAND memory through a bidirectional data bus, 8 or 16 bits wide. In 16-bit mode, commands and addresses use only the lower 8 bits. The upper 8 bits are only used during data-transfer cycles.



3. The 2-Gbyte NAND device is organized as 2048 blocks.

Erasing a block requires about 2 ms. Once the data is loaded in the register, programming a page requires about 300 —s. A page read requires approximately 25 —s, in which the page is accessed from the array and loaded into the 16,896-bit register. The register is then available for the user to clock out the data.

In addition to the I/O bus, the NAND interface is comprised of six major control signals:

- Chip enable (CE#): If CE is not asserted, the NAND device will remain in standby mode and not respond to any control signals.
- Write enable (WE#): WE# is responsible for clocking data, address, or commands into the NAND.
- Read enable (RE#): RE# will enable the output data buffers.
- Command latch enable (CLE): When CLE is high, commands are latched into the NAND command register on the rising edge of the WE# signal.
- Address latch enable (ALE): When ALE is high, addresses are latched into the NAND address register on the rising edge of the WE# signal.
- Ready/busy (R/B#): If the NAND device is busy, the R/B# signal will be asserted low. This signal is open drain and needs a pull-up resistor.

Data is shifted into or out of the NAND register 16 or 8 bits at a time. When doing a Program operation, the data to be programmed is clocked into the data register on the rising edge of the WE#. Special commands are used to randomly access or move data around within the register to make random access easier.

Data is output from the data register in a similar fashion using the RE# signal, which is responsible for outputting the current data and incrementing to the next location. The WE# and RE# clocks can run as fast as 30 ns. When RE# or CE# aren't asserted low, the output buffers will be tri-stated. This combination of CE# and RE# enables the output buffers, allowing NAND flash to share the data bus with other types of memory like NOR, SRAM, or DRAM. This feature is sometimes called "chip enable don't care." The primary purpose of this reference is to accommodate older NAND devices, which require CE# to be asserted for the entire cycle.

All NAND operations start supplying a command cycle *(Table 1)*. This is accomplished by placing the command on I/O bits 7:0, driving CE# low and CLE high while issuing a WE# clock. Note that commands, address, or data are clocked into the NAND device on the WE# signal's rising edge. Table 1 also shows that Most commands require a number of address cycles followed by a second command cycle. Note that with the exception of the Reset or Read Status commands, new commands shouldn't be issued if the device is busy.

Command	Command Cycle 1	Number of Address Cycles	Data Cycles Required	Command Cycle 2	Valid During Busy
PAGE READ	00h	5	No	30h	No
PAGE READ CACHE MODE START	31h	—	No	—	No
PAGE READ CACHE MODE START LAST	3Fh	—	No	—	No
READ for INTERNAL DATA MOVE	00h	5	No	35h	No
RANDOM DATA READ	05h	2	No	E0h	No
READ ID	90h	1	No	—	No
READ STATUS	70h	—	No	—	Yes
PROGRAM PAGE	80h	5	Yes	10h	No
PROGRAM PAGE CACHE MODE	80h	5	Yes	15h	No
PROGRAM for INTERNAL DATA MOVE	85h	5	Optional	10h	No
RANDOM DATA INPUT	85h	2	Yes	—	No
BLOCK ERASE	60h	3	No	D0h	No
RESET	FFh	—	No	—	Yes

Looking at the addressing scheme for 2Gb NAND devices, the first and second address cycles specify the column address, which specifies the starting byte within the page *(Table 2)*. Note that because the last column location is 2112, the address of this last location would be 08h (in the second byte) and 3Fh (in the first byte). PA5:0 specify the page address within the block and BA16:6 specify the block address. While the full 5-byte address is required for most Program and Read operations, only the first and second bytes are needed for operations that randomly access data within the page. The Block Erase operation only requires the three most significant bytes (third, fourth, and fifth) to select the block.

Cycle	I/O7	I/O6	I/O5	I/O4	I/O3	I/O2	I/O1	I/O0
First	CA7	CA6	CA5	CA4	CA3	CA2	CA1	CA0
Second	LOW	LOW	LOW	LOW	CA11	CA10	CA9	CA8
Third	BA7	BA6	PA5	PA4	PA3	PA2	PA1	PA0
Fourth	BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8
Fifth	LOW	LOW	LOW	LOW	LOW	LOW	LOW	BA16

Notes: 1. Block address concatenated with page address = actual page address. CA = column address; PA = page address, BA = block address.
2. If CA11 = "1" then CA[10:0] must be "0."

Reset operation
As discussed previously, CE# must be low for all NAND activities. The simplest NAND command is the Reset (FFh), which doesn't require any address or second cycle. Simply assert CLE and issue a write pulse with FFh on the data bus, and a Reset operation is performed. Reset is one of two commands that can be issued while the NAND device is busy. If it's busy processing a previous command, issuing a Reset aborts the previous operation. Note that if the previous operation was an Erase or Program command, issuing a Reset aborts the command prematurely, and the desired operation doesn't complete. Because Erase and Program can be time-consuming operations, they can be aborted with a Reset and re-issued later.

Read ID operation
The Read ID (90h) command requires one dummy address cycle (00h), but doesn't need a second command cycle *(Table 1, again)*. After issuing the command and dummy address, the ID data can be read out by keeping CLE and ALE low and toggling the RE# signal for each byte of ID.

Read status operation
Read Status (70h) is the second command that can be issued while the NAND device is busy. This command doesn't require an address or second command cycle. The NAND device's status can be interrogated by issuing the RE# clock signal. If the Read Status command is used to monitor the device's ready state, the command should only be issued one time and the status re-read by re-issuing the RE# clock. Alternatively, the RE# signal can be kept low waiting for the appropriate status bit. Read Status also includes the status of the write protect pin as well as the pass/fail status of the previous Program or Erase operations. It's mandatory that successful status be attained on Program or Erase operations to ensure proper data integrity.

The Block Erase (60h) operation erases an entire block of 64 pages or 128 bytes total. To issue this operation, use the WE# signal to clock in the Erase command (60h) with CLE asserted. Next, clock in three address cycles, keeping ALE asserted for each address byte. The three address cycles are the most significant address cycles of *Table 2*, which

includes the block and page addresses. The page address portion (the low order 6 bits of the third address cycle) is ignored, and only the block address portion of the three most significant bytes is used. Once the address is input completely, issue the second command (command cycle 2) of D0h, which also gets clocked in with WE# while CLE is being asserted. This confirms the erase operation, and the device goes busy for roughly 2 ms. When this operation completes, the device is ready for another command. Notice the Read Status command can be issued at any time, even when the device is busy during the erase. The processor or controller could interrogate the device using the read status command.

Program operations

Program operations can only program bits to 0, and assume that you started with a previously-erased block. If you don't want to program a bit, keep it in its erased state by setting that particular bit (or group of bits) to 1. When the Program Page (80h) command is received, the input register is reset to all 1s (internally). This lets the user input only data bytes that he wants to program with 0 bits. The Program operation starts with the 80h command, with CLE asserted *(Fig. 4)*. Next, drop CLE and assert ALE to input the full five address cycles.

Click here for Fig. 4

4. When the Program Page (80h) command is received, the input register is internally reset to all 1s, letting the user input only data bytes that he wants to program with 0 bits.

After the command and address are input, data is input to the register. Once all the data has been input, we issue the confirm command (10h) and start the programming operation. A Program operation typically requires 300 µs, although it may require up to 700 µs. It's mandatory that you read the status and check for successful operation. If the operation isn't successful, log the block and don't use it in the future. Move all data from the block to a good block.

Random data input operation

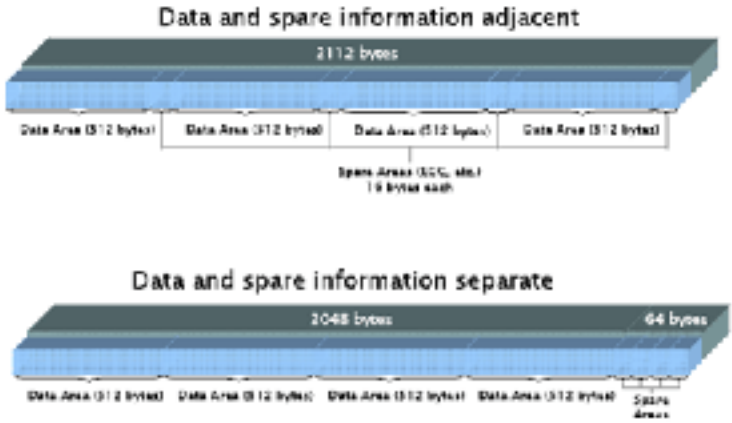
The next command is the random data input command (85h). This command only requires two bytes of address followed by the data (Fig. 5). It's useful when you want to jump around within a page, to access ECC data, for example. Random data input can be used to jump to the end of the page and write the ECC data. You can input as many address and data combinations as needed. It's only when the program confirm command (10h) is issued that the data is actually programmed to the selected page.

Click here for Fig. 5

5. Show is the Program command with random data input. The highlighted section shows that this command only requires two bytes of address followed by the data.

Due to the NAND page's large size, partial page programming is needed to store smaller-sized data. Each NAND page could accommodate four PC-sized 512-byte sectors. The spare area of each page provides additional storage for ECC and other information. While it's advantageous to write all four sectors at once, it's not always possible. An example of this is when you are appending a file. The file might start out as 512 bytes, with additional data coming later, making it 1024 bytes. In this case, writing the second 512 bytes would require a second page program operation to the NAND device. The maximum specification for partial page programming is eight; this would accommodate four data sectors and ECC, each programmed separately.

There are two common methods for storing data and spare information in the same page (Fig. 6). The first contains a data area of 512 bytes with the spare area directly adjacent to it. The spare area in this case is 16 bytes. Therefore, a 2112-byte page could contain four 528-byte elements. The second implementation involves storing the data and spare information separately. This means the 512-byte data is stored first, and the respective spare information for that data is stored at the end of the array starting at the beginning of the spare area. The second 512 bytes is stored next to the first 512, and the associated spare area for the second 512 is stored in the next 16-byte spare area. This continues until all four 512-byte sectors are stored with their respective 16-byte spare areas.

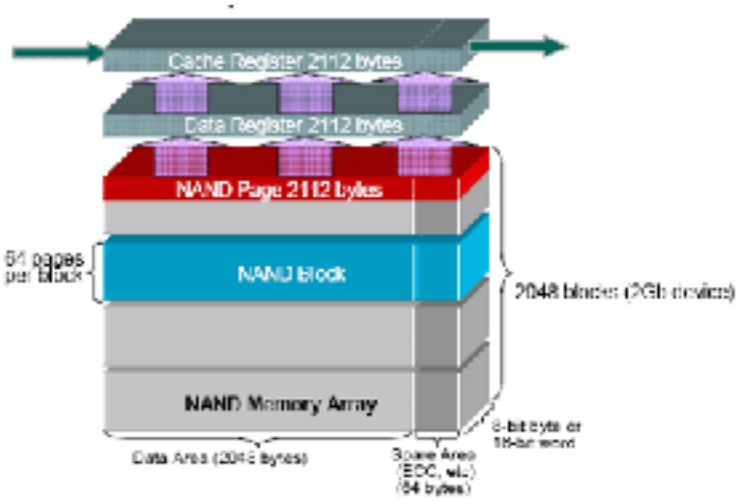


6. Typical storage methods are shown.

Read operation

A read operation starts with a command (00h), followed by five address cycles, followed by the read confirm (30h). After the read transfer time (tR) of approximately 25 µs, the data is loaded into the register and ready for output. Issuing the read enable (RE#) clock lets the NAND output the first byte corresponding to the column address specified in the address. Subsequent RE# transitions output successive locations. When RE# is high (not asserted), the I/O lines are tri-stated. Also, reading past the end of the device (byte 2112 or word 1056) results in reading invalid data. Random data can be directly accessed by issuing the (05h) command, two address cycles, and a (E0h) confirmation cycle. Once the page has been read from the array, this command provides rapid access to the data.

The NAND device actually has two registers: a data register and a cache register (Fig. 7). The Page read cache mode command lets you pipeline the next sequential access from the array while outputting the previously-accessed data. This double-buffered technique allows you to hide the read access time (tR). Data is first transferred from the NAND array to the data register. If the cache register is available (not busy), the data is quickly moved from the data register to the cache register. Once it's been transferred to the cache register, the data register is available and can start to load the next sequential page from the NAND array.



7. Shown is the page read cache mode.

A 33% performance improvement can be achieved on an 8-bit I/O device, resulting in up to 31 Mbytes/s of throughput. With the16-bit I/O device, throughput can be increased to 37 Mbytes/s, a 40% improvement over the normal Page read operation. Read cache can be especially useful during system boot-up, when large amounts of data are typically read from NAND and start-up time is critical.

Program page cache mode command

Program page cache mode provides performance improvement over normal Program page operations. This double-buffered technique lets the controller input data directly to the cache register and uses the data register as a holding register to supply the programming of the array.

This frees the cache register so that the next sequential page operation can be loaded in parallel. In many applications, the programming time (tPROG) can be completely hidden. Like the page read cache mode command, the data register maintains the data through the entire programming cycle. This frees up the cache register so that it can start receiving the next page of data from the controller.

Read for internal data move (00h, 35h), or copy back, is another useful system-level command. It provides the ability to move data from one page to another internally without leaving the NAND device. The Read for internal data move operation transfers the data read from the NAND array to the cache register. It can then be programmed into another page of the NAND device. This is beneficial in cases where the controller needs to move data out of a block before erasing that block. It's also possible to modify the data read before the program operation starts. This could be useful if you wanted to change the data before programming. This feature allows data to be moved around within the NAND device without tying up the processor or the I/O bus.

Connecting NAND to a processor

There are significant advantages to selecting a processor or a controller with a built-in NAND interface. If this option isn't available, it's possible to design a glueless interface between the NAND and almost any processor. The main difference between NAND and NOR flash is the multiplexed address and data bus. This bus is used to specify commands, address, or data. The CLE signal specifies command cycles, while the ALE signal specifies address cycles. Using these two control signals, it's possible to select a command, address, or data cycle. Connecting ALE to the processor's address bit five and CLE to the processor's address bit four enables the selection of either command, address, or data simply by changing the address that the processor outputs. This allows CLE and ALE to be asserted automatically at the appropriate time.

To supply a command, the processor outputs the intended command on the data bus and output address 0010h. To supply any number of address cycles, the processor simply needs to output the intended NAND address sequence to processor address 0020h. Note that many processors can specify several timing parameters around the processor's write signal, which is critical for proper timing. Using this technique, you can access commands, address, and data directly from the processor without any glue logic. In this case, ECC would have to be handled in the software.

Multi-level cell

A Multi-level cell (MLC) stores two bits per cell, versus traditional SLCs that can only store one bit. There are obvious density advantages for MLC technology. However, it doesn't offer the speed or reliability of its SLC counterpart (Table 3). Because of this, SLC is used in most media cards and wireless applications, while MLC devices are typically found in consumer and other low-cost products.

Click here for Table 3

As mentioned, NAND requires ECC to ensure data integrity. NAND flash includes extra storage on each page. The extra storage is the spare area of 64 bytes (16 bytes per 512-byte sector). This area can store the ECC code as well as other information like wear-leveling or logical-to-physical block-mapping. ECC can be performed in hardware or software, but hardware implementation provides an obvious performance advantage. During a programming operation, the ECC unit calculates the error-correcting code based on the data stored in the sector. The ECC code for the respective data area is then written to the respective spare area. When the data is read out, the ECC code is also read, and the reverse operation is applied to check that the data is correct.

It's possible for the ECC algorithm to correct data errors. The number of errors that can be corrected depends on the correction strength of the algorithm used. Including ECC in hardware or software provides a robust system-level solution. Simple Hamming codes provide the easiest hardware implementation, but can only correct single-bit errors. Reed-Solomon codes can provide a more robust error correction and are used on many of today's controllers. Also, BCH codes are becoming popular due to their improved efficiency over Reed-Solomon.

Software is needed to perform the NAND flash's block management. This software is responsible for wear-leveling or logical-to-physical mapping. The software may also provide the ECC code if the processor does not include ECC hardware.

It's important to read the status register after a program or erase operation, as it confirms successful completion of the operation. If the operation wasn't successful, the block should be marked bad and no longer used. Previously programmed data should be moved out of the bad block into a new (good) block. The spec for a 2-Gbyte NAND device states that it could have up to 40 bad blocks, a number that applies throughout the device's life (nominally 100,000 program/erase cycles). Due mostly to their large die size, NAND devices can ship from the factory with some bad blocks. The software managing the device is responsible for mapping the bad blocks and replacing them with good blocks.

The factory marks these blocks in such a way that the software can scan all the blocks to determine which are good and which aren't. The bad-block mark is fixed at the first location in the spare area (column location 2048). If location 2048 in either page 0 or 1 is 'non-FF,' then the block should be considered bad and mapped out of the system. The initialization software simply needs to scan through all blocks to determine which are bad and then build a table of these bad blocks for future reference.

It's important to take special care not to erase the bad-block marks. The factory tests NAND over a wide range of temperatures and voltages. Some blocks that are marked bad by the factory may be functional at certain temperatures or voltages but could fail in the future. If the bad-block information is erased, it can't be recovered.

There are several third-party NAND market today. Many of these companies provide bad-block management, directory support, and wear-leveling. Some third-party NAND software vendors include Datalight (www.datalight.com), CMX (www.cmx.com), HCC (www.hcc-embedded.com), and Blunk Microsystems (www.blunkmicro.com).

About the author

Jim Cooke *is a principal applications engineer for Micron's memory products group. He has over 20 years of hands-on systems-level design experience in embedded applications and digital consumer markets, and holds a BSEE from the University of Massachusetts. Cooke can be reached at jcooke@micron.com.*

This article is excerpted from a paper of the same name presented at the Embedded Systems Conference Silicon Valley 2006. Used with permission of the Embedded Systems Conference.