**How Nand flash works**

**Basics**

The basic interface is fairly simple. When asserted low, the chip enable (CE#) pin enables the NAND flash to accept bytes written to the chip when write enable (WE#) is asserted low or enable the output of a data byte when read enable (RE#) is asserted low. When CE# is high, the chip ignores RE# and WE# and the I/O is tri-stated. The Command Latch Enable (CLE) pin and the Address Latch Enable (ALE) pin act as multiplexer select pins by selecting which internal register is connected to the external I/O pins. There are only three valid states as shown in the table below:

**ALE CLE Register Selected**

0 0 Data register

0 1 Command register

1 0 Address register

1 1 Not defined

The key to understanding how the NAND flash operates is the realization that in the NAND flash, the read and program operation takes place on a page basis (i.e. 528 bytes at a time for most NAND devices) rather than on a byte or word basis like NOR flash. A page is the size of the data register. The erase operation takes place on a block basis (for most NAND devices, the block size is 32 pages). There are only three basic operations in a NAND flash: read a page, program a page, and erase a block.

**Operations**

**Page Read**

In a page read operation, a page of 528 bytes is transferred from memory into the data register

for output. The sequence is as follows:

***Command phase***

With CLE=1, ALE=0, the command byte 00h is placed on the I/O pins and WE# is brought low, then high. This stores the “read mode 1” command into the command register.

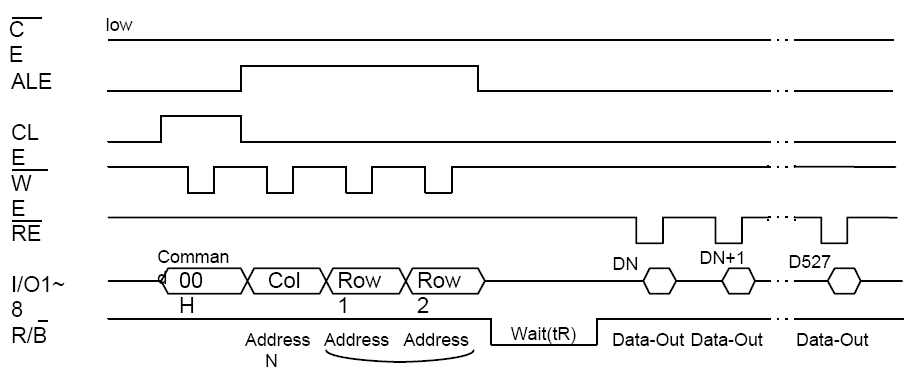
***Address phase***

With CLE=0, ALE=1, the first address byte is placed on the I/O pins and WE# is toggled. This first address byte “N” (called the column byte in the figure below) is usually set to 0 in order to start reading from the beginning of the page. It is possible to set N to any value between 0 and 255. Because the page is actually 528 bytes long, a different read command is used if you want output data to start from byte 256-511 (read mode 2 – command byte 01h is used instead of 00h). A third read command is used if you want output data to come from bytes 512-527 (read mode 3 – command byte 50h is used instead of 00h). It should be noted that the full page is read from memory into the register. The value N, in conjunction with the read command used, simply sets the output data pointer

within the register. The address bytes which follow after column byte N, indicated by Row1 and Row2 in the figure, are used to set the page within a block (lowest 5 bits in byte Row1), and the block within the device. In the higher density NAND devices, the address phase is 4 bytes long rather than 3.

***Data Transfer phase***

CLE and ALE are set to zero while the chip goes busy in preparation for data readout. During the busy period, the ready/busy pin (R/B) goes low for up to 25 microseconds while data is being read from the memory array and transferred into the data register. During this period, it is important that chip enable is held low to keep the read operation from being stopped mid-cycle (note: this restriction is removed in a new family of NAND flash devices known as CE don’t care).

******

***Read Out phase***

Once R/B returns high, data is available in the data register for read out. The first data byte output is byte N. Each RE# pulse reads out the next byte in the register. Once the last byte (D527) is read out, standard NAND flash will automatically go busy (another data transfer phase) in preparation for reading out the next page (with no additional command or address input). In the datasheet, this is called sequential read. If this is not desired, chip enable must be brought high (note: for the CE don’t care family of NAND flash, the automatic sequential read function does not exist).

**Page write**

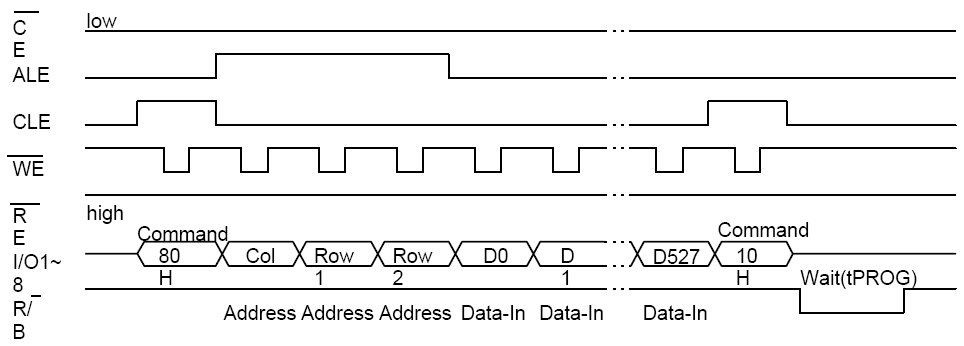
In a page program operation, a page of 528 bytes is written into the data register and then programmed into the memory array.

***Command phase***

With CLE=1, ALE=0, the command byte 80h is placed on the I/O pins and WE# is brought low, then high. This stores the “serial data input” command into the command register. This command also resets the register to all “1”s (all FFh).

***Address phase***

With CLE=0, ALE=1, the first address byte is placed on the I/O pins and WE# is toggled. This first address byte “N” (called the column byte in the figure below) is usually set to 0 in order to start writing from the beginning of the page. However, like the read command, it is also possible to set N to any value between 0 and 255. The first byte that is written in the data phase will then overwrite the FFh at location N in the register. If you desire to overwrite the register values starting at byte N (N=256-527), you need to precede the 80h command with either 01h or 50h (the read mode 2 and read mode 3 commands). It should be noted that the full page is programmed from the register into the memory each time the program command (10h) is received. However, since the serial data input command (80h) resets the register to all “1”s, bytes in the register that are not overwritten with data will remain “1” and should not will not affect the memory. Like the read mode, the address bytes which follow after column byte N, indicated by Row1 and Row2 in the figure, are used to set the page within a block (lowest 5 bits in byte Row1), and the block within the device. In the higher density NAND devices, the address phase is 4 bytes long rather than 3.



***Data Input phase***

CLE and ALE are set to zero, and data bytes are written into the data register. If you try to write more bytes than the page size, the last byte in the register will contain the last byte written.

***Program phase***

With CLE=1, ALE=0, the auto program command (10h) is written to the command register. The device then goes busy for tPROG (typically 250us). During this busy period, even if chip enable goes high, the device will finish programming.

***Timeout Check phase***

Although not shown on the diagram, it is typical to check the status after programming. If the device was unable to program a bit from 1 to 0 within the time allowed, the pass/fail bit returned by the status read command will indicate a failure. If this happens, the block should be considered bad because the device has already attempted to program the bit multiple times before the internal timeout occurred.

**Block Erase**

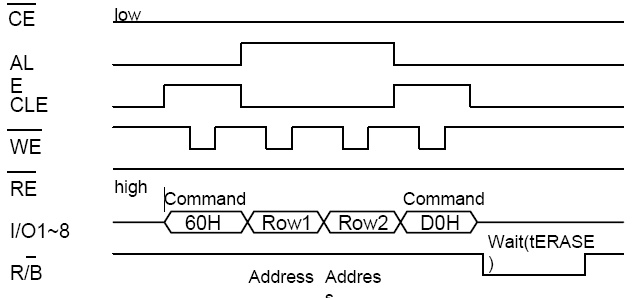
In a block erase operation, a group of consecutive pages (typically 32) is erased in a single operation. While programming turns bits from “1” to “0”, block erasure is necessary to turn bits from “0” back to “1”. In a brand new device, all usable (good) blocks are in the erased state.

***Command phase***

With CLE=1, ALE=0, the command byte 60h is placed on the I/O pins and WE# is brought low, then high. This stores the “auto block erase” command into the command register.

***Address phase***

With CLE=0, ALE=1, two address bytes are written into the address register. Notice that only two address bytes are required. There is no “column” byte as in the read and program operations. In the first address byte (Row1), only the upper 3 bits are used. The lower 5 bits of Row1are reserved for the page within the block (for device with 32 pages per block) and during a block erase operation, all pages within the block will be erased; therefore, the value of the least significant 5 bits is actually don’t care. The upper 3 bits of Row1 and the 8 bits of Row2 determine the block that will be erased. Because this is only 11 bits (2048 blocks max.), higher density NAND devices require 3 address bytes.



***Erase phase***

With CLE=1, ALE=0, the auto block erase confirm command (D0h) is written to the command register. The device then goes busy for tERASE (typically 2ms). During this busy period, even if chip enable goes high, the device will finish erasing the block.

***Timeout Check phase***

Although not shown on the diagram, it is typical to check the status after erasing to make sure a timeout (erase failure) did not occur. If the device was unable to erase the block successfully within the time allowed, the pass/fail bit returned by the status read command will indicate a failure. If this happens, the block should be considered bad because the device has already attempted to erase the block multiple times before the internal timeout occurred.

**Bad Blocks**

There is a finite number of write and erase cycles that all types of flash memory can achieve. The existence of bad blocks does not affect the good blocks because each block is independent and individually isolated from the bit lines by block select transistors. A bad block list (or bad block table) that can be updated needs to be maintained in the system. The cause of bad blocks could be a number of reasons (decoder failure, word line failure, memory cell failure) so once the bad blocks have been located, Toshiba recommends that the bad blocks no longer be accessed.

**Difference with NOR Flash**

* In the internal circuit configuration of **NOR Flash**, the individual memory cells are connected in parallel, which enables the device to achieve random access. NOR Flash is ideal for lower-density, high-speed read applications, which are mostly read only, often referred to as code-storage applications. **NAND Flash** was developed as an alternative optimized for high-density data storage, giving up random access capability in a tradeoff to achieve a smaller cell size, which translates to a smaller chip size and lower cost-per-bit. This was achieved by creating an array of eight memory transistors connected in a series.
* **NOR flash** has an SRAM interface. It has enough address pins to map its entire media, allowing for easy access to every byte contained in it. **NAND devices** are accessed serially via a complicated I/O interface, which may vary from one device or vendor to another. The same eight pins are used to convey control, address, and data information.
* In **NOR Flash**,the data addressing is done by byte addressing within the device(memory-mapped address).

But in case of **NAND** ,it is being representated as a page(row number). Pages are a power of two; commonly 512 or 2048 bytes. Optimized for reading and writing a page at a time. Sometimes supports column (byte) addressing, but this library does not expose such functionality.

* In Performance, see the below table

NOR Nand

Erase Very slow(5s) Fast(3ms)

Write slow Fast

Read Fast Slow

* **NOR flash** can execute in place but in case of **NAND flash**, its contents must be copied to RAM before they can be executed.
* There is factory default bad blocks in **NOR Flash** whereas in **NAND Flash** Typically up to 20 eraseblocks are marked as factory-bad in their OOB area. The OS is expected to scan these to create a Bad Block Table.
* There is no Error correction and detection present on NOR Flash but is available in NAND Flash.

**ATMEL NAND (MT29F2G08AABWP)**

1. Page size: 2,112 bytes (2,048 + 64 bytes)
2. Block size: 64 pages (128K + 4K bytes)
3. Device size: 256mB: 2,048 blocks;
4. Read performance:
   * Random read: 25μs
   * Sequential read: 30ns (3V x8 only)
5. Write performance:
   * Page program: 300μs (TYP)
   * Block erase: 2ms (TYP)
6. First block (block address 00h) guaranteed to be valid without ECC (up to 1,000 PROGRAM/ERASE cycles)
7. Data retention: 10 years

**Samsung Flash(K9 series)**

1. Page size: 2,112 bytes (2,048 + 64 bytes)
2. Block size: 128 pages (128K + 4K bytes)
3. Device size: 64mB: 256 blocks;
4. Automatic Program and Erase(Typical)
   * Page Program : (256 + 8)Byte in 250ms
   * Block Erase : (4K + 128)Byte in 2ms
   * Status Register
5. 264-Byte Page Read Operation
   * Random Access : 10ms(Max.)
   * Serial Page Access : 80ns(Min.)

**How to ignore bad blocks while reading/writing in u-boot**

***Implemented in Samsung board***

(refer in u-boot common/cmd\_nand.c)

Since in nand flash some of the blocks are bad , So while writing kernel and filesystem from u-boot(bootloader),we have to ignore some bad blocks in flash memory.It was not implemented in our bootloader (uboot).So when writing to flash using “nand write” command, one pointer is available which is incrementing the address location where we are going to write, there we are checking each location whether the location we are going to write is bad or not.

How: One function named “is\_nand\_bad()”which is already available in u-boot, using that function we will know whether that location is bad or not ,if bad it will return one and instead of writing to that location we will skip that location and will write to next location, therefore we ignored the bad blocks in nand.

**Nand flash test in u-boot**

To test nand flash in u-boot implement as a command in common/cmd\_test.c i.e

U\_BOOT\_CMD( cosmostest, 2, 1, do\_cosmostest,

"cosmostest - Test SDRAM & NAND in bootloader level.\n",

"Type cosmostest and start testing the Peripherals.\n"

);

Nand has been tested by comparing some bit pattern but reading and writing from SDRAM.Since the command in u-boot supports writing/reading through SDRAM(except erase) i.e

Nand write “SDRAM address” “nand offset” “nand size”

Nand read “SDRAM address” “nand offset” “nand size”

In the code…

To erase

data[0] = "nand";

data[1] = "erase";

data[2] = start\_addr;

data[3] = size;

ret = do\_nand(0, 1, 4, &data); //built in function in uboot for nand operations

To write

data[0] = "nand";

data[1] = "write";

data[2] = "20000000"; //SDRAM address

data[3] = start\_addr;

data[4] = size;

do\_nand(0,1,5,&data);

for(block=0; block<80; block++) {

for(page=0; page<64; page++) { //page is 64 since checking in SDRAM

offset=(block\*0x20000)+(page\*0x800); //1 Block = 128K,1 Page = 2048 bytes

baseAddress= CFG\_SDRAM\_START + offset;

for(count =0;count <128;count++)

{

\*((unsigned long\*)baseAddress) = bitpattern;

baseAddress+=4;

\*((unsigned long\*)baseAddress) = bitpattern;

baseAddress+=4;

\*((unsigned long\*)baseAddress) = bitpattern;

baseAddress+=4;

\*((unsigned long\*)baseAddress) = bitpattern;

baseAddress+=4;

}}}

For reading also same condition but the data[1] = “read” instead of write and since we are comparing in SDRAM and SDRAM is 32 bit(4 bytes) nand is 8 bit(1 byte)…

if(\*((unsigned long\*)baseAddress) != bitpattern) {

printf(“\nRead fail with a counter increment which tells how blocks gone bad\n”);

}

baseAddress+=4;