TOSHIBA

NAND Flash Applications Design Guide

System Solutions from

Toshiba America Electronic Components, Inc.

Atsushi Inoue, Staff MTS, Memory Business Unit
Doug Wong, Staff MTS, Memory Business Unit

Revision 1.0
April 2003

TOSHIBA

TOSHIBA AMERICA ELECTRONIC COMPONENTS, INC.

This "NAND Flash Applications Design Guide" and the information and know-how it contains constitute the exclusive property and trade secrets of Toshiba America Electronic Components, Inc. ("TAEC"), and may not be reproduced or disclosed to others without the express prior written permission of TAEC. Any permitted reproductions, in whole or in part, shall bear this notice.

The information in this "NAND Flash Applications Design Guide" has been checked, and is believed to be reliable; however, the reader understands and agrees that TAEC MAKES NO WARRANTY WITH RESPECT TO THIS DESIGN GUIDE, ITS CONTENTS OR THEIR ACCURACY, AND EXCLUDES ALL EXPRESS AND IMPLIED WARRANTIES, INCLUDING WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR NON-INFRINGEMENT. The reader further understands that he or she is solely responsible for all use of the information contained within, including, but not limited to, securing any necessary intellectual property rights, however denominated.

All information in this "NAND Flash Applications Design Guide" is subject to change without prior notice, at TAEC's sole discretion.

All trademarks, trade names, product, and/or brand names are the property of their respective holders.

TOSHIBA

Table of Contents

TOSHIBA

1. History of Flash at Toshiba

When Dr. Masuoka, who is now a professor at Tohoku University, joined Toshiba in 1971, he had been thinking that the substitution of magnetic memory would be indispensable for the development of semiconductor memory.

He also understood that the market size for memory is more dependent on its bit cost than its user-friendliness. For instance, let's compare the market size between DRAMs and SRAMs. SRAM is faster, requires no refresh, and is very user-friendly, but the market size for DRAMs is much larger than that of SRAMs. The only reason why the DRAM market is larger is because the cost of DRAMs is much lower than that of SRAMs. This is the RAM story, but the ROM story is similar. Like SRAM, the byte-EEPROM is very user-friendly because it can erase and program a single byte. But its cost is so high that it cannot be widely adopted. A hard disk, which can also be considered to be a non-volatile memory, does not offer byte reprogramming, but does offer sector reprogramming (block reprogramming) and is widely used because of low cost. So, what is required is not the flexibility of byte reprogramming, but a low cost per bit.

Based on the concepts above, Dr. Masuoka applied for a patent on simultaneously erasable EEPROMs in 1980. Although a conventional byte-EEPROM has two transistors per cell, a new memory cell which consists of only one transistor, was proposed to reduce cost. To realize a one-transistor cell, the bit erase scheme was dismissed and a simultaneous erase scheme was adopted. The development of a real test device was started in 1983 with Dr. Masuoka's colleagues: Mr. Asano and Mr. Iwahashi for the design, Mr. Tozawa, Mr. Komuro, Mr. Tanaka for the device technology, and supported by Mr. Suzuki, the memory senior manager. Fortunately, the device was verified be functional. In June of 1984, the first paper was submitted to IEDM. At that time, Dr. Masuoka recognized that it must be the first simultaneously erasable EEPROM in the world and thought about naming it with his colleagues. Mr. Ariizumi, one of his colleagues, proposed naming it "Flash" sometime in June of 1984, before submitting the IEDM paper. Why Mr. Ariizumi suggested the term "Flash" was because the device could erase a block simultaneously, which made him imagine the Flash of a camera. But no one, at the moment, could have dreamed that Flash memory would be used in digital cameras today. So what was first called simultaneously erasable EEPROM, was named "Flash" from 1984. The memory cell area for the first proposed Flash EEPROM was 64 sq. microns while a conventional byte-erasable EEPROM cell at that time occupied 272 sq. microns in the same lithography design rule of 2 microns.

In December of 1984, the first paper for the Flash EEPROM was presented at IEDM in San Francisco. A subsequent the paper on a 256k bit Flash EEPROM was presented at ISSCC in San Francisco in February of 1985. After that, Dr. Masuoka was interviewed by "Business Week" and the Flash EEPROM was reported in Business Week on Mar.25, 1985. On the news, Dr. R.D. Pashley of Intel was interviewed to provide counterpoints. But afterwards, Intel stopped developing the UV-EPROM (ultraviolet erasable) and focused on Flash memory development. Dr. Pashley would become the General Manager of the Flash memory division.

After Toshiba presented the 256k bit Flash EEPROM at the '85 ISSCC, Seeq developed a 128k bit Flash EEPROM and announced it at the '87 ISSCC. Seeq's memory cell was programmed by hot electron injection and erased by field emission from the floating gate to the drain. Therefore,

Seeq's cell could be realized by a dual polysilicon structure while Toshiba's Flash EEPROM cell used a triple polysilicon structure due to the formation of the erase gate. Intel presented a 256k bit Flash EEPROM at the '88 ISSCC. Intel adopted the same cell structure as that of the UV-EPROM. It is programmed by the hot electron injection like a UV-EPROM and erased by the field emission from the floating gate to the source. In principal, this concept is quite similar to that of the first proposed Flash EEPROM by Toshiba.

2. How Flash Works

Like a UV-EPROM (ultraviolet erasable programmable read only memory) cell, a Flash EEPROM (electrically erasable programmable read only memory) cell has a dual gate structure in which a floating gate exists between a control gate and a silicon substrate of a MOSFET. A floating gate is perfectly isolated by the insulator, i.e. silicon dioxide, so that the injected electrons cannot leak out of the floating gate after power is removed. This is the basic storage mechanism for the Flash EEPROM non-volatile memory. The charge retention mechanism for Flash EEPROM is the same as conventional UV-EPROM and byte-erasable EEPROM. Like a UV-EPROM, a Flash EEPROM is programmed by a hot electron injection mechanism, and erased by field emission from a floating gate, like a byte-erasable EEPROM. The erase mechanism for a Flash EEPROM cell is the same as that for a byte-erasable EEPROM cell; however, their basic use as LSI memories are typically different. In a Flash EEPROM, the whole chip can be erased simultaneously, while a byte-erasable EEPROM is erased only one byte at a time. When the byte erase function is eliminated, an electrically re-programmable non-volatile memory can be realized by utilizing only one transistor per cell. A UV-EPROM also simultaneously erases all its bits by exposure to ultraviolet light, and is programmed by hot electron injection mechanism. In this sense, a UV-EPROM is similar to a Flash EEPROM in functionality except that the erase operation is carried out by UV irradiation.

2.1 NAND vs NOR Flash

Current semiconductor memories achieve random access by connecting the memory cells to the bit lines in parallel, as in NOR-type flash. In NOR-type flash, if any memory cell is turned on by the corresponding word line, the bit line goes low (see figure 1). Since the logic function is similar to a NOR gate, this cell arrangement results in NOR flash.

However, speedy access is not always required in order to replace magnetic memory. The NAND Flash is a new flash configuration that reduces memory cell area so that a lower bit cost can be achieved. In 1987, Toshiba proposed the NAND Flash, and its NAND structured cell arranged as eight memory transistors in series. The NAND flash cell array, fabricated by using conventional self-aligned dual polysilicon gate technology, had only one memory transistor, one forth of a select transistor and one sixteenth of the contact hole area per bit. This technology realizes a small cell area without scaling down the device dimensions. The cell area per bit was half that of a DRAM using the same design rule of 1um (which was used for the 1M bit DRAM). As a result, Toshiba realized that it was possible for NAND Flash to be developed earlier than DRAM (for the same density) by one process generation. In comparison, conventional EEPROM was behind DRAM by one process generation at that time.

The most important item regarding memories is the bit cost. In the case of a semiconductor memory, the bit cost is dependent on the memory cell area per bit. Since the cell area of NAND Flash is smaller than that of NOR Flash, NAND Flash always had the potential from the start to be less expensive than NOR Flash. However, it takes a rather long time for a NAND Flash to read out the first data byte compared to NOR Flash because of the resistance of the NAND cell array, although it is much faster than the seek time for a hard disc by several orders of magnitude. Therefore, the aim of NAND Flash is to replace hard disks.

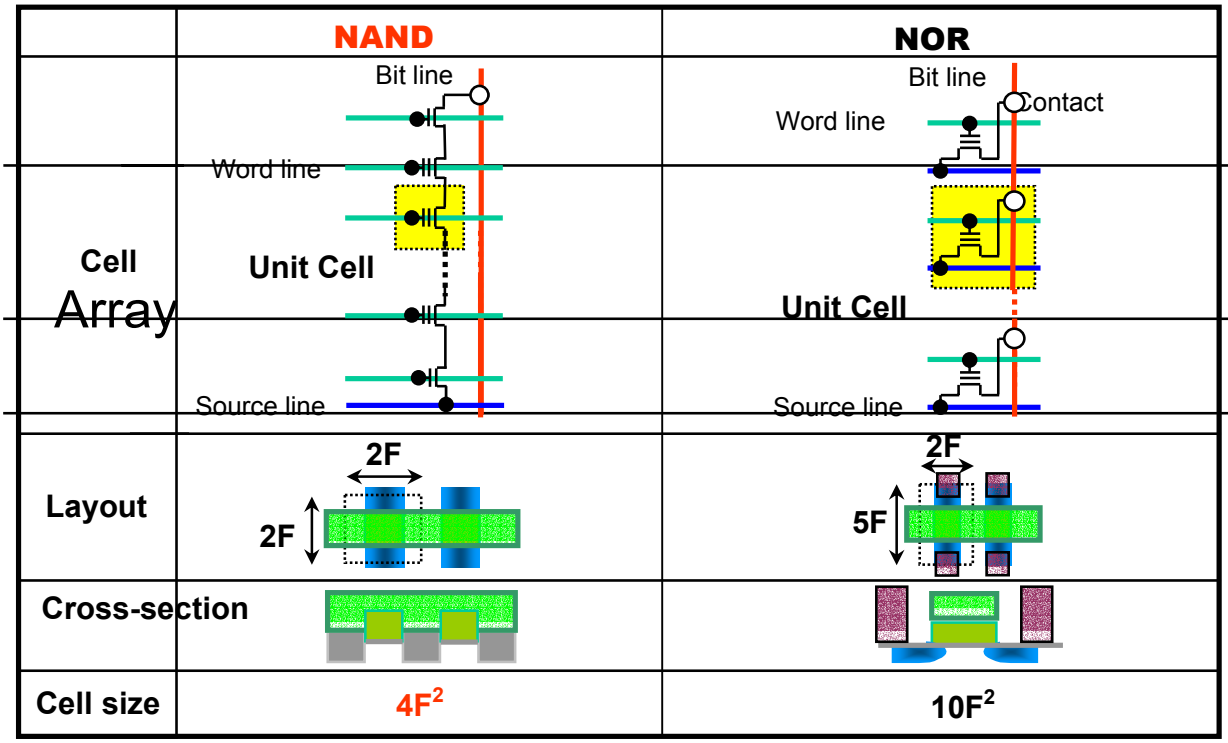


Figure 1. NAND Flash vs. NOR Flash

The advantages of NAND Flash are that the erasing and programming times are short. The programming current is very small into the floating gate because NAND Flash uses Fowler-Nordheim tunneling for both erasing and programming. Therefore, the power consumption for programming does not significantly increase even as the number of memory cells being programmed is increased. As a result, many NAND Flash memory cells can be programmed simultaneously so that the programming time per byte becomes very short. Conversely, the NOR Flash can be programmed only by byte or word, and since it uses the hot electron injection mechanism for programming, it also consumes more power and the programming time per byte is longer. The programming time for NOR Flash is typically more than a order of magnitude greater than that of NAND Flash.

The power consumption of NAND Flash or NOR Flash is about one tenth that of a hard disk drive. Also, the seek time for semiconductor memories is much faster than that of a hard disk. However, NAND Flash or NOR Flash must be erased before reprogramming while a hard disk

requires no erasure. Therefore, in the case of continuous programming where the seek time is negligibly small, a hard disk drive can be programmed more quickly.

For both for NOR Flash and NAND Flash, the endurance (which means the number of cycles a block or chip can be reprogrammed) is limited. In order to replace the UV-EPROM with Flash, and endurance of 1000 cycles was sufficient. It is estimated that at least 1,000,000 cycles are required to replace a hard disk drive. NOR Flash is typically limited to around 100,000 cycles. Since the electron flow-path due to the hot electron injection for programming is different from the one due to tunneling from the floating gate to the source for erasing, degradation is enhanced. However, in NAND Flash, both the programming and erasing is achieved by uniform Fowler-Nordheim tunneling between the floating gate and the substrate. This uniform programming and uniform erasing technology guarantees a wide cell threshold window even after 1,000,000 cycles. Therefore, NAND Flash has better characteristics with respect to program/erase endurance. In some recent scaled NOR Flash memories, their erasing scheme has been changed from source side erasing to uniform channel erasing, which is the same as the NAND Flash.

From a practical standpoint, the biggest difference a designer will notice when comparing NAND Flash and NOR Flash is the interface. NOR Flash has a fully memory-mapped random access interface like an EPROM, with dedicated address lines and data lines. Because of this, it is easy to "boot" a system using NOR Flash. On the other hand, NAND Flash has no dedicated address lines. It is controlled using an indirect I/O-like interface and is controlled by sending commands and addresses through a 8 bit bus to an internal command and address register. For example, a typical read sequence consists of the following: writing to the command register the "read" command, writing to the address register 4 byte of address, waiting for the device to put the requested data in the output data register, and reading a page of data (typically 528 bytes) from the data register. The NAND Flash's operation is similar to other I/O devices like the disk drive it was originally intended to replace. But because of its indirect interface, it is generally not possible to "boot" from NAND without using a dedicated state machine or controller. However, the indirect interfaces advantage is that the pinout does not change with different device densities since the address register is internal. Because NAND Flash is optimized for solid-state mass storage (low cost, high write speed, high erase speed, high endurance), it is the memory of choice for memory cards such as the SmartMedia™, SD™ card, CompactFlash™, and MemoryStick™.

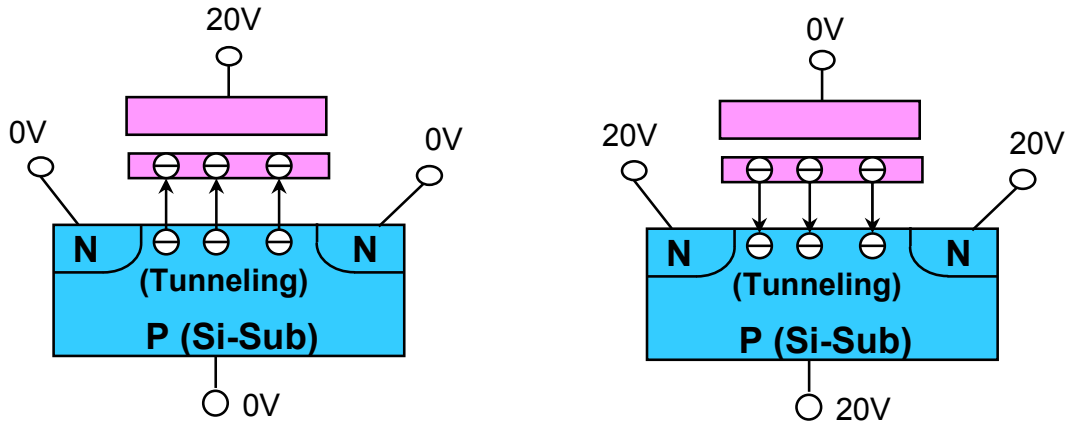


Figure 2. NAND Flash Cell Biasing

3. The NAND Flash Interface

The pinout of the standard NAND Flash in the TSOP I package is shown in figure 2 below.

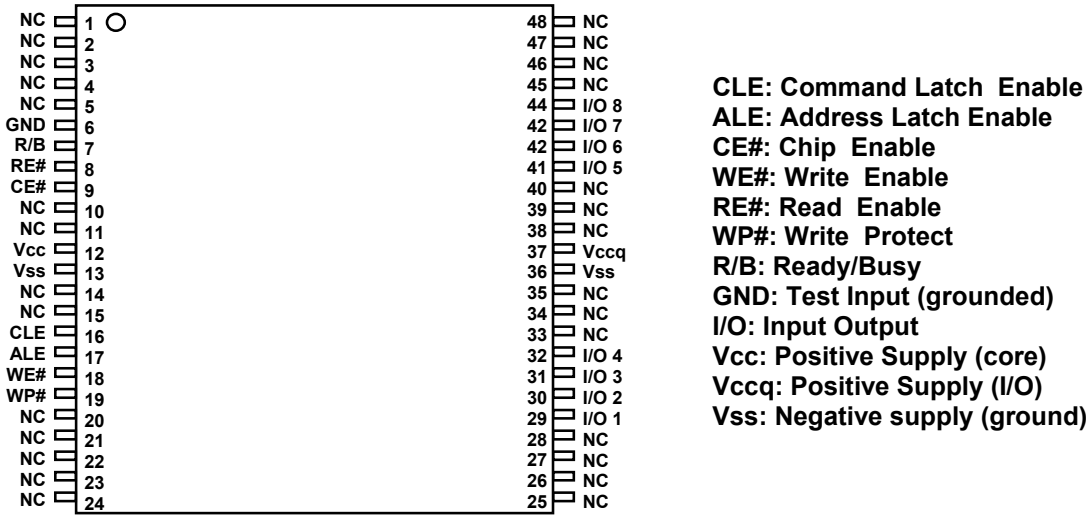


Figure 3. NAND Flash Pinout.

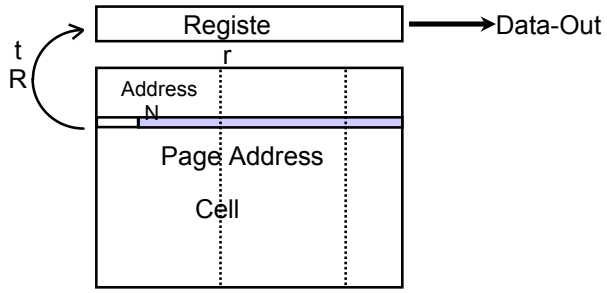
The basic interface is fairly simple. When asserted low, the chip enable (CE#) pin enables the NAND flash to accept bytes written to the chip when write enable (WE#) is asserted low or enable the output of a data byte when read enable (RE#) is asserted low. When CE# is high, the chip ignores RE# and 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

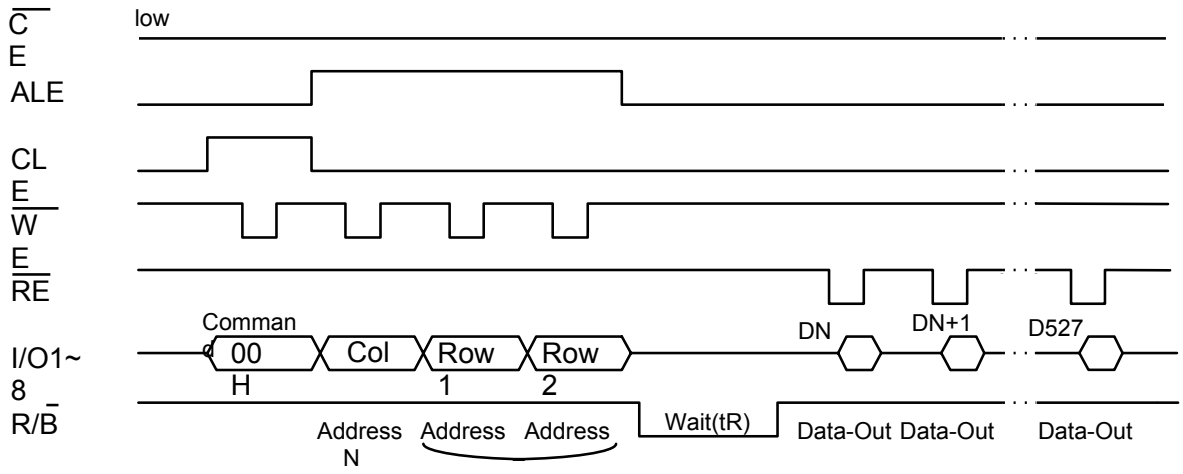
The key to understanding how the NAND flash operates is the realization that in the NAND flash, the read and program operation takes place on a page basis (i.e. 528 bytes at a time for most NAND devices) rather than on a byte or word basis like NOR flash. A page is the size of the data register. The 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. Let's examine each of these operations in more detail.

3.1 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.

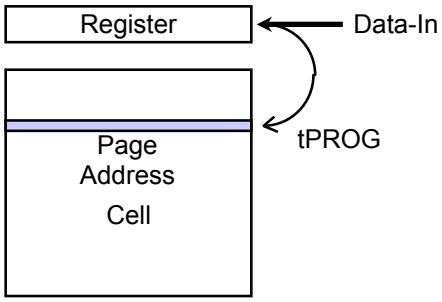


- Data Transfer phase: CLE and ALE are set to zero while the chip goes busy in preparation for data readout. During the busy period, the ready/busy pin (R/B) goes low for up to 25 microseconds while data is being read from the memory array and transferred into the data register. During this period, it 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).

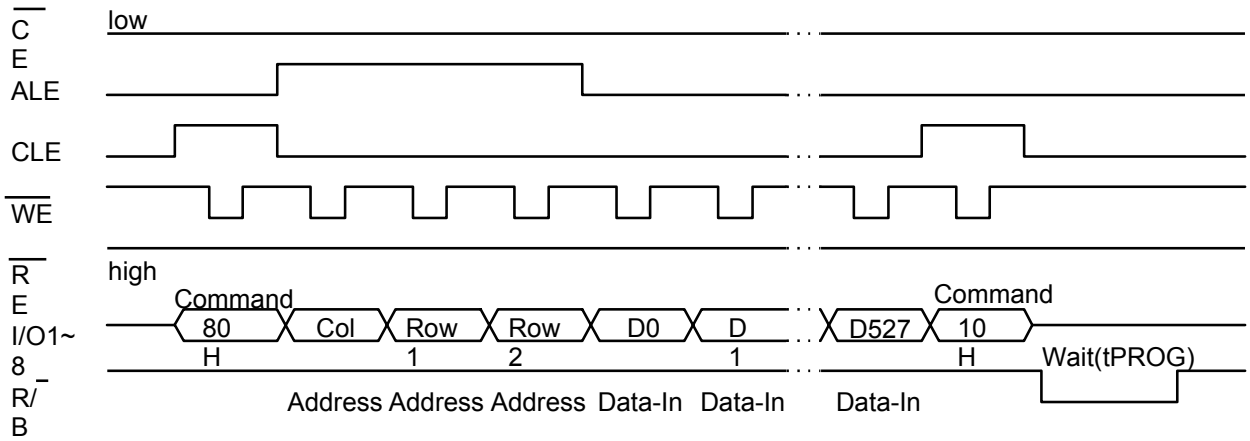
Why is a page 528 bytes long? Since the original intent of the NAND flash was to replace magnetic hard disk drives, the intention was for the page to be big enough to store 1 sector (512 bytes) worth of data with 16 bytes extra for overhead such as error correcting code. Because the use of ECC is common with NAND flash (sample code is in the appendix), read mode 1 is the most often used read command because it enables one to read the entire 528 byte page.

3.2 Page Program

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



- 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



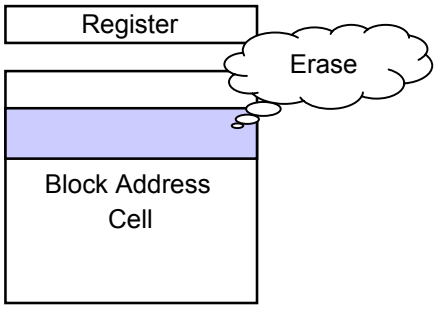
precede the 80h command with either 01h or 50h (the read mode 2 and read mode 3 commands). It should be noted that the full page is programmed from the register into the memory each time the program command (10h) is received. However, since the serial data input command (80h) resets the register to all "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.

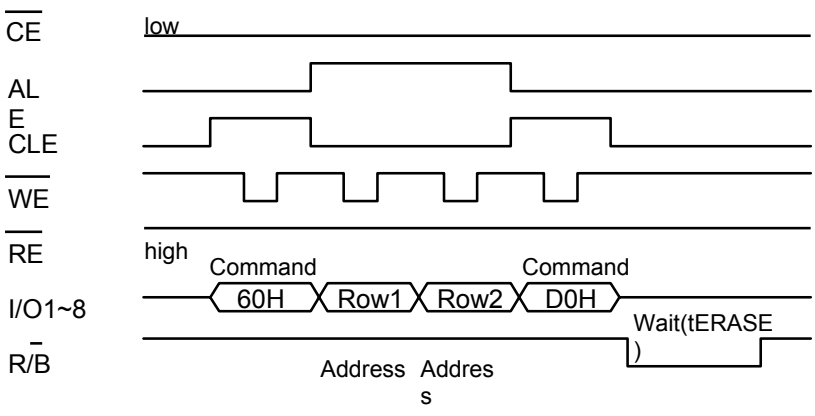
- Program phase: With CLE=1, ALE=0, the auto program command (10h) is written to the command register. The device then goes busy for tPROG (typically 250us). During this busy period, even if chip enable goes high, the device will finish programming.
- Timeout Check phase: Although not shown on the diagram, 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.

3.3 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 are 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 (and verify it is erased) multiple times before the internal timeout occurred.

4. Hardware Interfacing

When you examine the timing diagrams in the datasheets for standard NAND flash devices, you will notice that there was the expectation that NAND flash would be connected to a controller chip or specialized interface state machine because of two characteristics:

- The chip enable is shown asserted low continuously during the period of the operation. Actually, chip enable can be deasserted in between individual write cycles and read cycles; however, it must remain continuously asserted low during the read cycle busy period. For chip enable don't care NAND, this restriction is removed.
- Signal ALE is shown to be high continuously between individual write cycles. Actually, in between write cycles, ALE can go low as long as the setup and hold times are met.

These timing diagrams are relatively easy to achieve if you connect the NAND flash to a state machine. However, if you intend to connect the NAND to a microprocessor bus directly, some glue logic will be necessary. There are several ways to connect the NAND flash to the host:

1) Using general purpose input/output (GPIO) pins
2) Using a memory-mapped interface with glue logic
3) Using a chip-enable don't care NAND

The key requirement in all cases will be to meet the timing diagram restrictions. For example, the setup and hold times for CLE, ALE, CE#, and data input with respect to WE# are shown below. Note that CLE, ALE, and CE# are not required to be held in a particular state outside the interval. The practical implication is that CLE and ALE can be connected to the host address lines in order to select the internal register connected: data register, command register, or address register.

The data read cycle is shown below. Not shown on the diagram are CLE and ALE, which are both assumed to be low. This diagram is for the chip enable don't care NAND; notice that the chip enable state is don't care during the busy period preceding the data read cycle. For standard NAND, chip enable must be held low during the busy period preceding data read out.

Command In / Address In

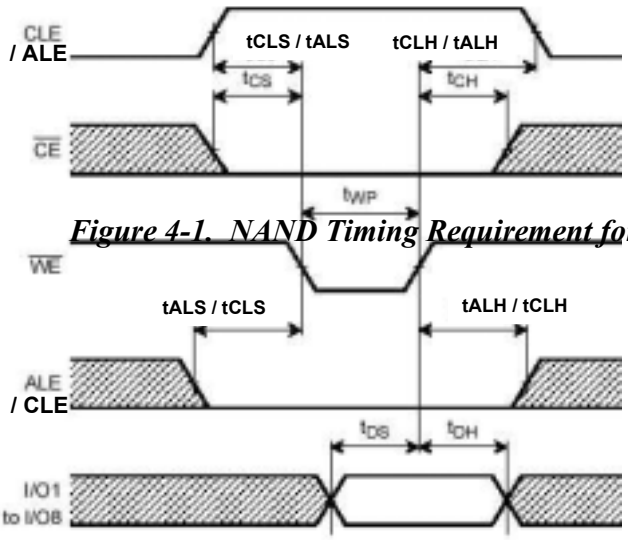


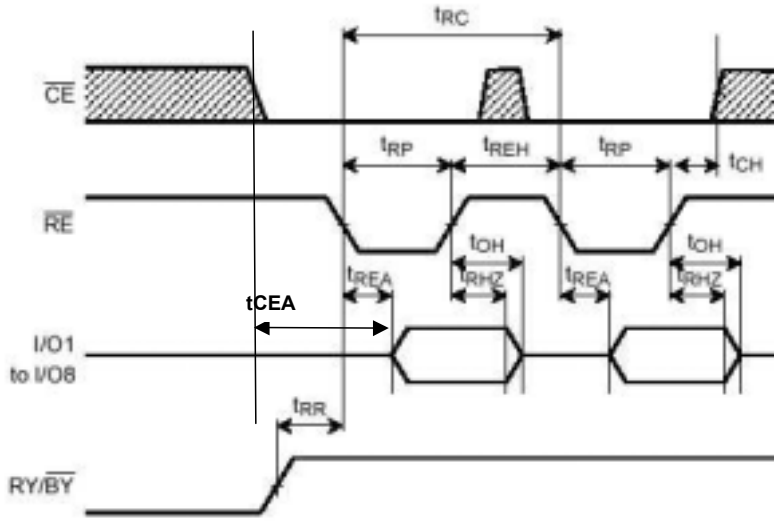
Figure 4-1. NAND Timing Requirement for Address/Command Inputs.

Set up time for ALE, CLE, -CE is based on the falling edge of -WE, hold time based on the rising edge of -WE.

Set up time for I/O is based only on the rising edge of -WE.

	Symbol	Spec
Setup	tCLS	0ns
	tALS	0ns
	tCS	0ns
	tDS	20ns
Hold	tCLH	10ns
	tALH	10ns
	tCH	10ns
	tDH	10ns
Other	tWP	25ns

Data Read



Symbol	Spec
tRC	50ns
tRP	35ns
tREH	15ns
tREA	35ns
tCEA	45ns
tOH	10ns
tRR	20ns

Figure 4-2. NAND Timing Requirement for Data Reads.

Using GPIO Pins

Using GPIO pins to control the NAND signals (such as ALE, CLE, /CE, /WE, and /RE) offers great flexibility in meeting the NAND timing requirements. However, unless the speed requirements are relatively low, the performance is likely to be a fraction of the NAND's potential performance. Also, GPIO pins are often scarce in a system, so this may not be an acceptable use of a scarce resource. However, although adding GPIO pins to the interface may involve additional cost, it may be easier to control the NAND for some platforms.

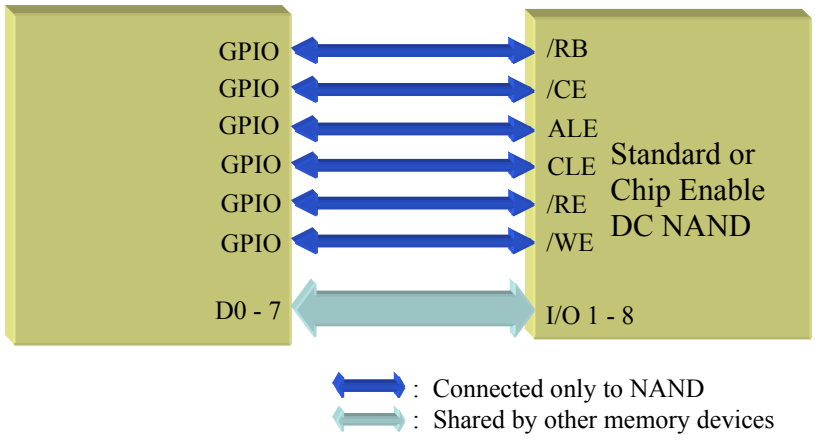


Figure 4-3. Physical Connection when GPIOs are used.

Memory-Mapped Interfacing using Glue Logic

In order to interface to standard NAND devices, it is necessary to use a latched signal to drive the NAND's chip enable. The simplest approach is to use a latched GPIO pin.

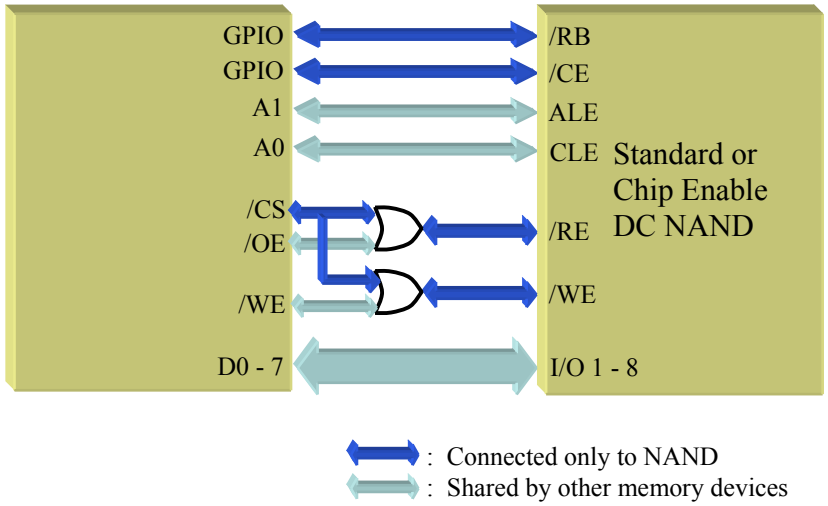


Figure 4-4. Physical Connection when Memory-Mapped.

The GPIO pin controlling the chip enable is asserted low at the beginning of the NAND read, program, or erase cycle and is not deasserted until the end of the entire cycle. Note that the read enable and write enable to the NAND is qualified by an address decoded chip select. In this way, only read or writes intended for the NAND actually toggle the NAND's read enable or write enable pins. When /CS is deasserted, the glue logic deasserts /RE and /WE, which tri-state the NAND's outputs.

Using a Chip Enable Don't Care NAND

Perhaps the simplest method to connect NAND to a microprocessor bus is the use of a chip enable don't care CEDC) NAND instead of standard NAND. The main difference between standard NAND and chip enable don't care NAND is that chip enable does not need to be continuously asserted low during the read busy period. The removal of this restriction allows chip enable to be deasserted between individual read or write cycle and enables the direct connection of the NAND to a microprocessor with no glue logic. The NAND chip enable will work as expected and qualify the read enable and write enable signals. The only function that was removed from standard NAND to make this possible was the elimination of the automatic sequential read function, which was rarely used anyway.

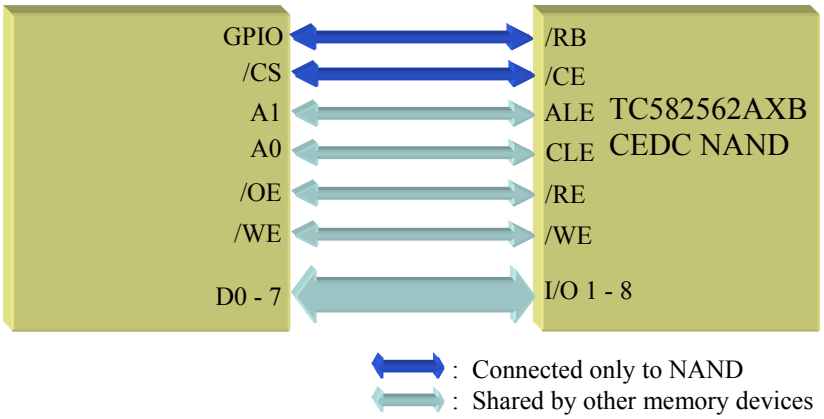


Figure 4-5. Connection using Chip Enable Don't Care NAND.

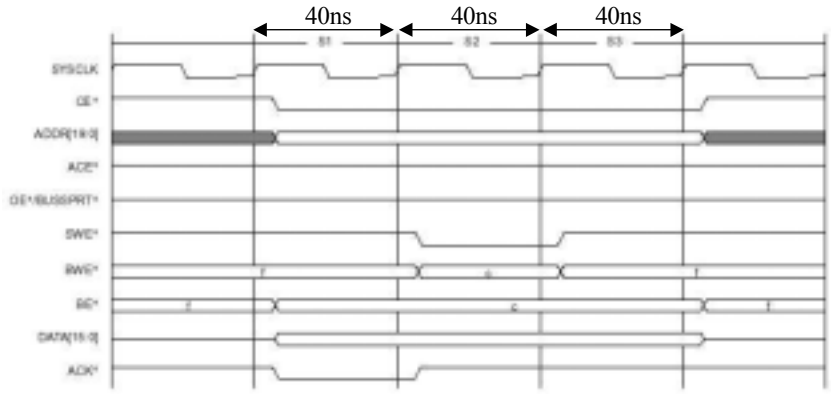
Unlike NOR flash, the NAND flash does not have any dedicated address pins to be connected to the microprocessor address pins. Therefore, most people think that a direct interface between NAND and a microprocessor is difficult. However, as shown in figure 4.5, the interface does not require any glue logic. Toshiba has demonstrated this glueless NAND connection between the Toshiba TX4927 MIPS processor and the Toshiba TC582562AXB NAND flash.

On the TX4927 demonstration board, the timing for the chip select (/CS) of the TX4927 was modified as described in figure 4-6 and 4-7 below. This was easily be done by changing the register values which controlled the timing for /CS. Most high end processors with integrated chip select circuitry have programmable timing. With CLE connected to A0 and ALE connected to A1, the software driver for the NAND need only access 3 address locations. Access to the base address for /CS accesses the NAND data register by setting CLE=0 (A0=0) and ALE=0 (A1=0). Writes to base address+1 writes the NAND command register by setting CLE=1 (A0=1) and ALE=0 (A1=0). Writes to base address+2 writes the NAND address register by setting CLE=0 (A0=0) and ALE=1 (A1=1).

With the introduction of the chip enable don't care NAND, interfacing to NAND flash has never been easier.

GBUSCLK = 100MHz = 10ns
The bus speed is set to 1/4 of the GBUSCLK → SYSCLK = 40ns.

Command In / Address In (120ns / cycle)

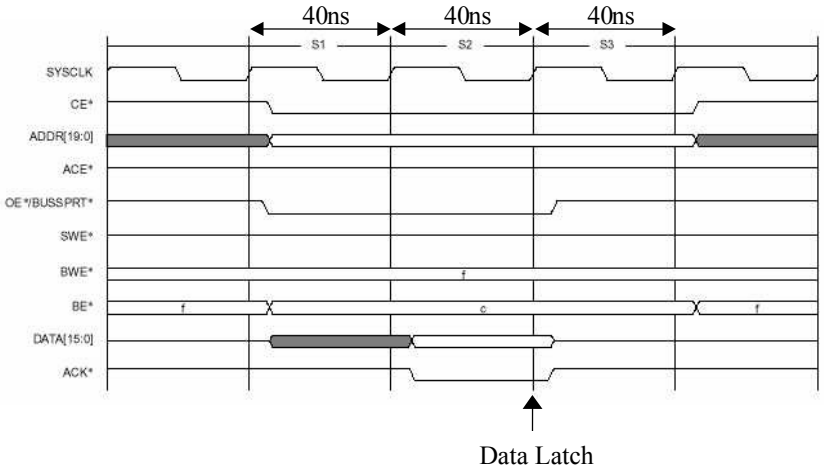


	Symbol	Spec	4927
Setup	tCLS	0ns	40ns
	tALS	0ns	40ns
	tCS	0ns	40ns
	tDS	20ns	80ns
Hold	tCLH	10ns	40ns
	tALH	10ns	40ns
	tCH	10ns	40ns
	tDH	10ns	40ns
Other	tWP	25ns	40ns

*) BWE, BE, ACK is not used.

Figure 4-6. Command In, Address Timing Generated by TX4927.

Data Read (120ns / cycle)



Symbol	Spec	4927
tRC	50ns	120ns
tRP	35ns	40ns
tREH	15ns	80ns
tREA	35ns	approx 80ns
tCEA	45ns	approx 80ns
tOH	10ns	- (*1)

*) BWE, BE, ACK is not used.
(*1) : Data latch point is within OE low, so not an issue.

Figure 4-7. Data Read Timing Generated by TX4927.

5. Large block vs. Small Block NAND

In the current NAND architecture, each page consists of 528 bytes, and each block consists of 32 pages. Future NAND devices will use the large page/large block structure in which a page will be 2112 bytes (4 times larger) and a block will consist of 64 pages (2 times larger) resulting in a block size that is 8 times larger. The first of these new large block NAND flash devices is the 1 Gbit TC58NVG0S3AFT00. Note that all large block devices will also have the chip enable don't care feature. The increased page and block size will enable faster program and erase speeds in future high density NAND flash.

Density	0.16 micron Small Page (528 B) Small Block (16kB)	0.13 micron Small Page (528 B) Small Block (16kB)	0.13 micron Large Page (2112 B) Large Block (128kB)
64 Mb	TC58V64BFT (standard)	N/A	N/A
128 Mb	TC58128AFT (standard) TC581282AXB (CEDC)	TC58DVM72A1FT00 (standard) TC58DVM72A1XB11 (CEDC)	N/A N/A
256 Mb	TC58256AFT (standard) TC58256AXB (CEDC)	TC58DVM82A1FT00 (standard) TC58DVM82A1XB11 (CEDC)	N/A N/A
512 Mb	TC58512FT (standard)	TC58DVM92A1FT00 (standard) TH58DVM92A1XB11 (CEDC)	N/A N/A
1 Gb	TH58100FT (standard)	TC58DVG02A1FT00 (standard)	TC58NVG0S3AFT00 (CEDC)
2 Gb	N/A	N/A	TH58NVG1S3AFT00 (CEDC)

Note: CEDC = Chip Enable Don't Care

Although the internal architecture will be different, the external physical interface will be the same. Therefore, in most cases, only the flash software needs to be updated in order to use these new devices.

The effective read speed of the large block NAND devices is similar to the small block devices:

Read Time = 6 cycles x 50ns + 25 μs + 2112 cycles x 50ns = 131 μs
Read Speed = 2112 bytes / 131 μs = 16.1 Mbytes /sec

The effective write speed of the large block NAND devices is more than 3 times faster.

Write Time = 5 cycles x 50ns + 2112 cycles x 50ns + 1 cycle x 50ns + 200μs = 306 μs
Write Speed = 2112 bytes / 306μs = 6.9 Mbytes / sec

The effective erase speed is nearly 8 times faster.

Erase Time = 4 cycles x 50ns + 2ms = 2ms
Erase Speed = 128kB / 2ms = 64 Mbytes / sec

6. Failure Mode Overview

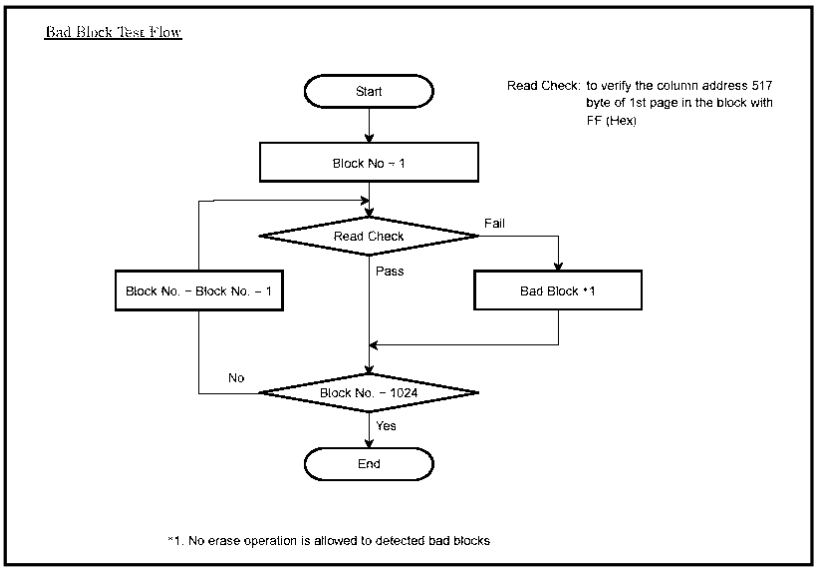
6.1 Bad Block Identification (Initial Bad Blocks)

The NAND flash was designed to serve as a low cost solid state mass storage medium. In order to achieve this goal, the standard specification for the NAND allows for the existence of bad blocks in a certain percentage. A bad block list (or bad block table) that can be updated needs to be maintained in the system. The bad block table can either be stored in one of the good blocks on the chip, or on another chip in the system such as RAM. A bad block table is also required because unlike magnetic media, flash memory does not possess infinite write/erase capability; there is a finite number of write and erase cycles that all types of flash memory can achieve. Because all flash memory will eventually wear-out and no longer be useable, a bad block table needs to be maintained to track blocks that fail during use.

Allowing for the existence of bad blocks increases the effective chip yield and enables a lower cost. 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.

During outgoing testing and burn-in testing, blocks that are considered bad by Toshiba are marked with a 00h in byte 0x205 (byte 517) in each page of a bad block (this is the same as the SmartMedia format for marking bad blocks). Toshiba determines that blocks are bad by performing extensive pattern testing over both temperature and voltage extremes.

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. To locate the bad blocks on a brand new device, read out each block. Any block that is not all FFh (all 1s) in byte 517 (starting from byte 0) of the 1st page of a block is a bad block. The figure below is a flowchart that shows how bad blocks can be detected by doing a read check on each block.



Once you erase a block, the non-FF bytes will be also be erased. If this occurs, re-identifying the bad blocks will be difficult without testing at different temperatures and voltages and running multiple test patterns, so if the list of bad blocks is lost, recovering bad block locations are extremely difficult.

6.2 Blocks that Fail During Use

As mentioned in the previous section, all flash memory has a finite lifetime and will eventually wear out. Since each block is an independent unit, each block can be erased and reprogrammed without affecting the lifetime of the other blocks. For NAND memory, each good block can be erased and reprogrammed more than 100,000 to 1,000,000 times typically before the end of life. This is described in the datasheet.

The primary wear out mechanism is believed to be excess charge trapped in the oxide of a memory cell and the net effect is that erase times increase until an internal timer times out (Narrowing Effect). The programming time seen by the user actually decreases slightly with increasing number of total write/erase cycles, so the device's end of life is not characterized by program failures. Generally, only a severe device failure could cause a page program failure.

Therefore, blocks should be marked as bad and no longer accessed if there is either a block erase failure or a page program failure. This can be determined by doing a status read after either operation. The status read command is used to determine the outcome of the previous erase or program operation. Block erase operations are automatically verified, so the entire block is FFh if the status bit indicates the erase operation passed. For programming, the status bit indicates the program operation passed if all zeros ("0") in the data register are correctly programmed into memory. One ("1") bits in the data register are not verified and are ignored. Therefore, if "0s" are already programmed into a page in memory, all program operations to that page, regardless of the data in the data register, would pass. By not verifying 1s, partial page programming is possible.

6.3 Failure Modes Mechanism and Symptoms

Although random bit errors may occur during use, this does not necessarily mean that a block is bad. Generally, a block should be marked as bad only if there is a program or erase failure. The four main failure modes that can be distinguished as "permanent failures" or "soft errors" are described below.

Permanent Failure

- Write/Erase Cycle Endurance - This error may be manifested as a cell, page, or block failure which can be detected by status read after either auto program or auto block erase (Figure 6.3.1).

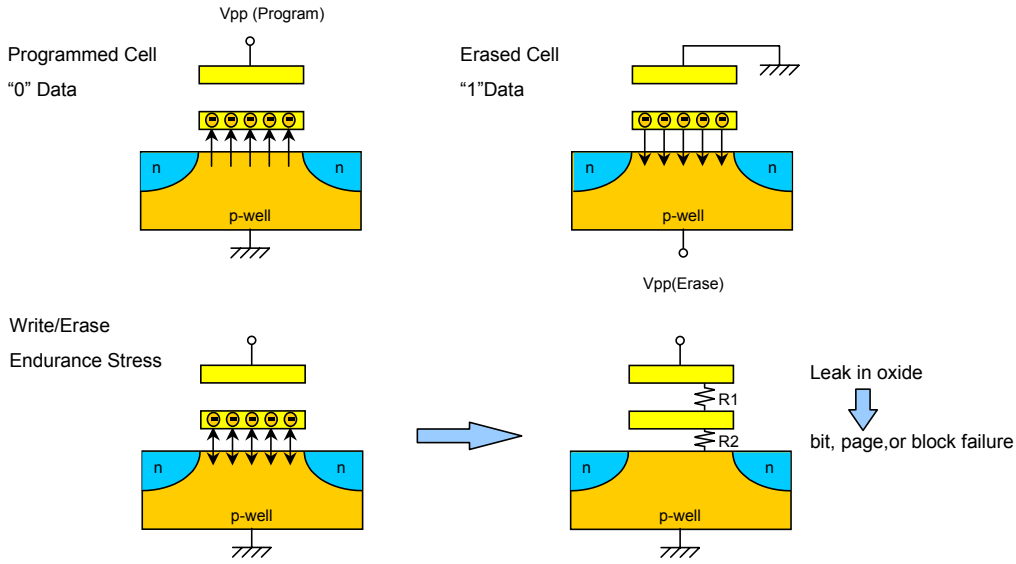


Figure 6.3.1. Write/Erase Endurance

Soft Errors

Over Programming - This is caused when the threshold voltage of a "0" data cell becomes too high as a result of excess programming current. Normally, all threshold voltages are below a bias voltage (Vbias) so that the application of Vbias to unselected pages will enable them to turn on (figure 6.3.2). If the threshold voltage of a cell is too high (Figure 6.3.3), the bias voltage that is supposed to be high enough to turn on any cell during the read cycle is insufficient, so the cell never turns on (figure 6.3.4).

Therefore the error occurs during a program, but can only be detected by reads. The resultant error symptom is that all cells on that bit line in the block read out as 0, so in the worse case scenario, if this bit is supposed to be "1" for all other pages in the block, there will be a one bit failure for each page in the block. This condition is cleared by a block erase.

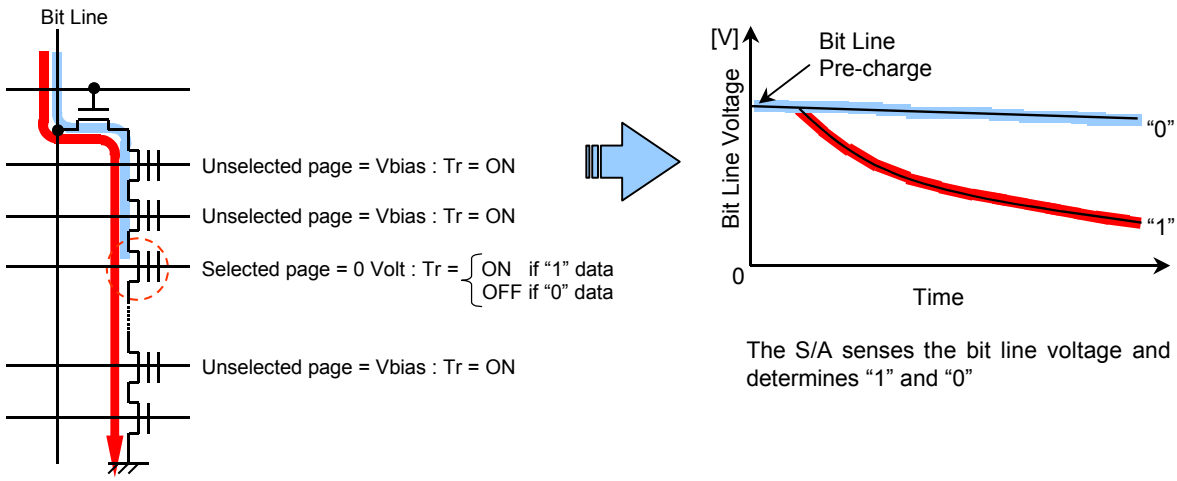


Figure 6.3.2. Normal Read

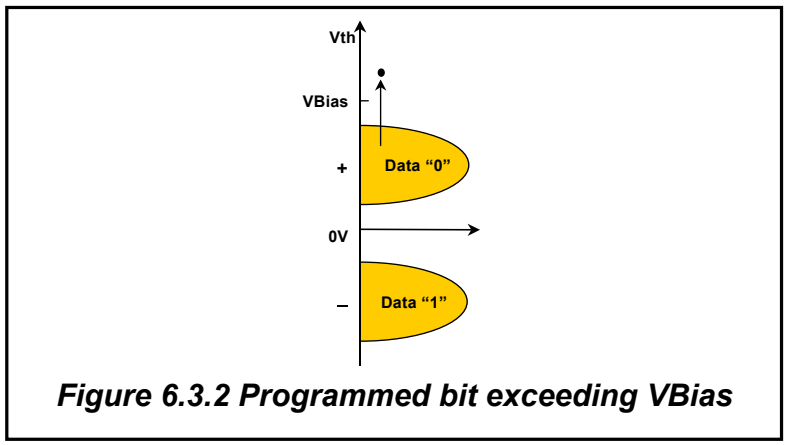


Figure 6.3.2 Programmed bit exceeding VBias

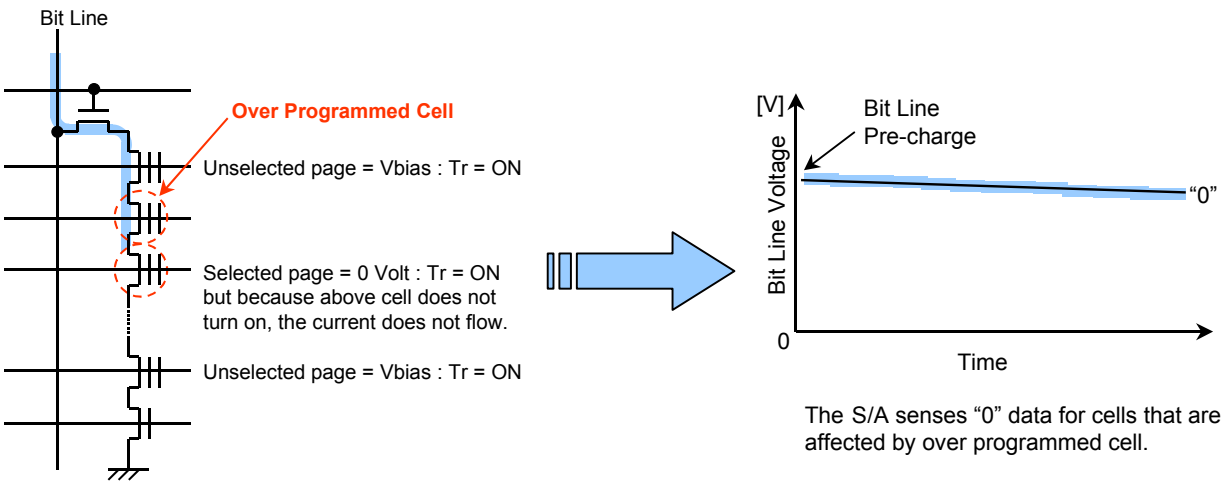


Figure 6.3.3. Read Operation with over programmed cell

Program Disturb - In this failure mode, a bit is unintentionally programmed from "1" to "0" during the programming of a page. The bit error may occur either on the page being programmed or on another page in the block. Bias voltage conditions in the block during page programming can cause a small amount of current to tunnel into memory cells. Multiple partial page programming attempts in a block can aggravate this error symptom. Since this error is caused by soft programming of memory cells, the condition is removed by block erasure. Program disturb effects are also worsened by randomly programming pages in a block. Therefore, the datasheets for NAND flash now require programming pages in sequential order only (from lowest page address to highest page address).

Understanding these error mechanisms is useful in order to understand how to interpret the NAND flash reliability report. Interpreting the reliability report is the subject of appendix A.

7. Managing NAND Flash

In order to use NAND flash effectively, the NAND flash must be managed by some kind of a controller (either software or hardware). This is necessary in order to make the NAND flash appear to the system as ideal block device.

7.1 Bad Block Management

In a brand new device, the standard NAND flash specification allows for the existence of initial bad blocks. Standard NOR flash devices have extra spare memory blocks that are used to replace bad blocks, but NAND flash devices have a minimal amount of redundant memory blocks because it was always expected that an intelligent controller would ignore the bad blocks. Since it was expected that NAND flash would be used for solid state mass storage, blocks would eventually wear out; therefore, it was expected that the system be able to handle bad blocks that would form during use.

The standard factory location for the bad block byte is byte 517 (the 518^{th} byte) of a NAND page. If this byte is FFh, the block is good, otherwise, the block is bad (typically indicated by 00h). This format for marking bad blocks is from SmartMedia card (NAND flash in a removable card package) and was standardized by the SSFDC Forum (Solid State Floppy Disk Card – the former name of SmartMedia). If additional bad blocks form during use, the block is marked bad. Generally, this is possible even if the block that you are marking is considered bad. To distinguish between factory marked bad blocks and blocks that go bad during use, two flag values are defined in the SmartMedia format: 00h (for initial factory marked bad blocks) and F0h (for blocks that go bad during system use).

An alternative approach to the "in block" method of keeping track of bad blocks is to maintain a bad block table. However, where to you store a bad block table since blocks could be bad? For NAND TSOP devices only, the first block of the NAND flash (block 0) is guaranteed to be good. Thus, block 0 could be used to hold a bad block table if desired. However, at power up, many systems simply scan the first page of each block to determine whether they are good or bad and build a bad block table in RAM.

7.2 Error Correcting Code

The use of an error correcting code is essential in order to maintain the integrity of stored code. Soft errors (especially during programming) occur at a rate of approximately 10^{-10} or about 1 bit per 10 billion bits programmed. Single bit correcting (two bit error detecting) Hamming code is sufficient for NAND flash. Toshiba has developed C sample code for implementing Hamming code. It is available in a separate document entitled the SmartMedia™ ECC Reference Manual.

7.3 Wear Leveling

If flash memory had infinite write/erase endurance, wear leveling would not be necessary. However, unlike magnetic media, flash memory eventually wears out and no longer programs or

erases in the allotted amount of time. Because the design of typical file systems assumed the characteristics of magnetic media, certain physical locations may be repeatedly rewritten. For example, in the DOS FAT file system, the FAT and directory areas must be modified multiple times each time a file is written or appended. When multiplied by the thousands of files in a typical file system, the FAT and directory areas of the disk will experience vastly more writes than any other area of the disk.

When flash memory is used to emulate a disk drive, the physical areas of the flash that contain the FAT and directory would be worn out first, leading to early failure of the file system stored on the flash. In order to spread out the writes across as much of the flash as possible, a wear leveling algorithm is implemented by the controller (software or firmware in a hardware controller) which translates a logical address to different physical addresses for each write. Generally, this logical to physical lookup table is implemented in RAM and is initialized at power up by reading each physical block in the NAND flash to determine its logical block value.

Ideally, wear leveling is intrinsic to the file system itself. Several new file system exist which write new data sequentially rather than overwriting a fixed location. These file systems use a technique known as journaling. For flash memory, JFFS2 (Journaling Flash File System 2) and YAFFS (Yet Another Flash File System) exist which automatically spread out wear by writing sequentially to free flash space.

7.4 Software Drivers

Software drivers for managing NAND flash are becoming available from a variety of sources. There are open source developments such as JFFS2 and YAFFS, as well as a number of drivers available from third parties. The table below lists the sources of NAND flash driver software we are currently aware of or have discovered on the web.

Product Name	**Company/Sponsor**	**Website**
F1Pack Angel & Jet	Tokyo Electron	http://tmg-eng.teldevice.co.jp/f1pack.html
FlashFX	Datalight	http://www.datalight.com
JFFS2	Red Hat	http://sources.redhat.com/jffs2/
NAND File system	Kyoto Software Research	Contact Toshiba http://www.toshiba.com/taec/
smxFFS	Micro Digital	http://www.smxinfo.com
TargetFFS-NAND	Blunk Microsystems	http://www.blunkmicro.com/ffs
TrueFFS	Wind River Systems	http://www.windriver.com/products/true_ffs/
YAFFS	Toby Churchill	http://www.aleph1.co.uk/armlinux/projects/yaffs/

7.5 Hardware Controllers

There are a number of sources for hardware controllers for NAND flash. To date, the main application for these controllers have been for use inside flash memory cards such as CompactFlash, USB drives, or flash memory card reader/writers. Manufacturers include SST, Cypress, Standard Microsystems Corp., and many others.

8. Tips for Using NAND Flash

8.1 MROM / NOR Replacement

In many cases, the intended use of the NAND flash is to act as a large read-only memory. There are two problems to consider. First, some type of bootstrap ROM is necessary (unless the processor has a built-in NAND controller state machine) since NAND flash is not a random access device. The bootstrap ROM will typically be MROM or NOR flash, although some processors have the ability to boot from a serial EEPROM. The bootstrap ROM code's job is to copy code from the NAND flash into system RAM. The second problem, the existence of initial bad blocks that must be skipped over, is handled by the bootstrap ROM code. Of course for systems without a significant amount of RAM space, shadowing code from the NAND into RAM is not a viable option. However, for most systems running on a 32 bit microprocessor and running an industrial strength real-time OS, significant amounts of RAM (SDRAM) are likely to be available and shadowing from NAND flash would be a very cost effective solution.

Typically, the bootstrap ROM code would be written in assembly language and should do minimal system initialization like setting up chip selects and initializing the DRAM controller. Then the bootstrap ROM code would:

1. Read the first page of a NAND block and examine the bad block mark location
2. Determine whether the block is good or not
3. If good, copy the data from the NAND flash into system DRAM and correct the data if necessary
4. If bad, skip over block
5. If additional blocks need to be transferred, repeat process

There is one question that often comes up "Is ECC really necessary?" After all, the likeliest cause of a bit error is during the programming process. For example, if you program a block, then verify it has no errors, how reliable is the data? In these ROM-like applications where the write/erase cycles is very low, the actual failure rate for a block is about 3 ppm after 10 years (i.e. 3 blocks out of every million blocks will have a bit error after 10 years) in which a block failure is defined as a single bit error. This result was derived from testing 29708 pieces of 512Mb NAND (0.16um) by writing a checkerboard pattern into blocks and storing at 125C. Since there will be a non-zero data retention failure rate, you should limit the amount of code to 1 block to achieve a low ppm probability of failure.

It is taken for granted that NAND flash is not bootable (at least for the moment) because of the lack of separate address and data lines, but there actually is a variant of NAND flash that is! Co-developed by Toshiba and M-Systems, the monolithic DiskOnChip™ has a true random access type of interface (13 address lines, 16 data lines, chip enable, write enable, output enable, etc) in a TSOP or BGA package. A small bootstrap loader program (1kB or 2kB) can be executed directly from the DiskOnChip without shadowing. TrueFFS™ software drivers have been written by M-Systems for the following operating systems: Windows CE, Linux, VxWorks, Symbian, Windows NT, PSOS, QNX, Nucleus, and DOS.

8.2 To Partition or Not to Partition

In the previous section, the NAND flash is used exclusively as a ROM in which a file system is unnecessary. However, many applications may wish to use part of the NAND flash as a ROM, and part as a file system. In this case, there are basically two approaches. In the first case, we can partition the NAND flash into two separate distinct regions in which code is stored in one partition and the file system is stored in the other. In the second case, we could use the entire NAND flash as a file system and store the code as a special file within it. The first case will be simpler to implement because the bootstrap loader program will not have to understand the file system in order to retrieve code from the NAND flash. However, the second case is more versatile. If code should grow in the future, there is no need to repartition the NAND flash. Development is easy because one can simply reload a new ROM image as a file. However, a more sophisticated bootstrap loader program requiring more space will be necessary.

8.3 Considerations for Preprogramming NAND

Preprogramming NAND flash is different than programming NOR flash primarily because the existence of bad blocks prevents the use of fixed physical addressing. Programmers that support NAND flash program complete blocks and skip over bad blocks. All overhead bytes (including ECC bytes) must be included in the data file itself. In the data file, every 518th byte (byte 517) out of every 528 bytes should be left as 0xFFh. As discussed in section 6.1, this byte is reserved as the bad block flag byte. A separate white paper describing the issues in preprogramming NAND flash is available from Toshiba America.

If the NAND flash is divided into 2 partitions as described in section 8.2, it will be necessary to program the NAND flash in two operations. In the first operation, the code portion is programmed. Since the bad block distribution will vary from chip to chip, the last physical block programmed will differ. If the second partition (i.e. file system partition) is to be written starting at the same physical address in every chip during the second program operation, several spare blocks (1-2%) typically needs to be added to the code partition to allow for bad blocks to enable the second partition to start at a fixed block location. Of course, there is still a possibility that in a particular chip, the bad blocks are concentrated in the code partition section. If this happens, there would be an insufficient number of good blocks in the physical block range allocated for code storage to actually store the code. Also, the hassle of dealing with two separate files (code and file system) to be programmed can lead to errors. Therefore, it will be more convenient to avoid partitioning the flash and implement case 2 in section 8.2 by storing the code as a special file in the file system and program a single file into the flash.

8.4 Considering Memory Cards

If portable storage is necessary, the easiest solution is to use one of the removable memory cards available. The advantage of using a memory card is that all memory cards (except SmartMedia) have a built-in memory controller chip. Toshiba, as the inventor of SmartMedia, co-inventor of the SD card, and a major manufacturer of CompactFlash cards, offers a variety of possible solutions. For further information on these cards, see:

- SmartMedia – http://www.ssfdc.or.jp
- SD Card – http://www.sdcard.org

- CompactFlash – http://www.compactflash.org

9. Introduction to CompactFlash

The CompactFlash™ card is a small, removable, storage and I/O card. Invented by Sandisk, the specifications are now determined by the CompactFlash Association (CFA) (http://www.compactflash.org), an organization that promotes the adoption of CompactFlash. The CompactFlash can be used in such applications as portable and desktop computers, digital cameras, handheld data collection scanners, PDAs, Pocket PCs, handy terminals, personal communicators, advanced two-way pagers, audio recorders, monitoring devices, set-top boxes, and networking equipment.

A CompactFlash card is essentially a small form factor card version of an ATA PC Card (AT Attachment) specification and includes a True IDE (Integrated Drive Electronics) mode which is compatible with the ATA/ATAPI-4 specification. As such, there are 3 distinct interface modes that a CompactFlash card can use:

- PC Card Memory Mode (uses WE#, OE# to access memory locations)
- PC Card I/O Mode (uses IOWR#, IORD# to access I/O locations)
- True IDE Mode (uses IOWR#, IORD# to access I/O locations)

The CompactFlash card is essentially a solid state ATA disk drive. To control an ATA disk drive, one writes to the task file registers. The values put into these task file registers control the drive (the ANSI T13 committee defines these registers and the commands used to control all ATA/IDE drives – see http://www.t13.org). These task file registers can be mapped into either memory or I/O address space.

A typical CompactFlash card consists of a controller and several NAND flash memory chips. The convenient aspect of using them is that the controller typically implements ECC in hardware and the NAND flash management in firmware offering both high reliability and high performance. Two application notes describing a reference interface between the CompactFlash card and either the MPC8260 or PPC405 are available from Toshiba.