©2010 International Journal of Computer Applications (0975 – 8887)
Volume 1 – No. 9

Performance Enhancement by the Design of Flash Controller for x8 NAND Flash Memory Devices

Pramod T. Talole
M.E. (Digital Electronics)
S.S.G.M.C.E., Shegaon, Dist: Buldana [MS]

Sarita T. Sawale
M.E. (CSE)
Govt. College of Engg. Aurangabad [MS]

ABSTRACT

The available flash devices are NAND flash and NOR flash memory devices. Each device is also divided with data width into x8 / x16 NAND flash memory and x8 / x16 NOR flash memory. We are going to implement Flash Controller which is the interface between x8 NAND flash device and the CPU/processor. The same concept will be used for other devices like x8 NOR also. Normal interface is between one x8 NAND flash device and CPU or Processor. This paper proposed a design to interface eight x8 NAND flash devices with CPU/processor. The main feature of the implementation is parallel write operation of eight x8 NAND flash devices. As the numbers of flash devices are 8, it requires dedicated DMA buffers for each x8 NAND flash device. Also to support parallel writes, it is required to read the data of a next DMA write from DRAM while current NAND flash write is in progress. Hence DMA engine is divided into two parts – Flash Side DMA Engine (DMA) and Host Side DMA Engine (HDE). These DMA engines are used to control the above operation.

Keywords

Design

1. INTRODUCTION

The flash interface unit is designed for system on chip (SoC) applications where flash memory acts as the main non volatile storage device. The need for non-volatile flash memory is increasing as many new applications require a system with no hard disks. Flash memory provides storage which is rugged, requires low power, and comes in a small and light form factor. The flash interface unit provides basic flash commands which can be used by the main CPU to access data in the flash memory. Since the flash memory contains information to initiate the boot process, it is assumed that the flash memory is non-removable. Since the flash memory acts as the main storage device, the main CPU should be able to read and write files like any generic file system. However, flash memory does not follow the standard read/write interface. Write access to flash memory is only through programs which change bits in memory from one to zero. In order to change bits in memory from zero to one, flash memory supports erases which sets blocks of memory to all ones.

2. MOTIVATION

The flash interface unit has been designed primarily to support DMP10, DMP20, and SC25. DMP10 and DMP20 are high end 2D/3D graphics and multimedia processors designed primarily for digital television, car navigation and pachinko applications. SC25 is an ultra low power 2D/3D graphics and multimedia processor designed primarily for mobile applications.

In order to support multimedia applications, the flash interface unit has been optimized for large block reads and writes. This interface must satisfy the bandwidth requirements required to transfer video and audio streams from flash memory, with special attention to *write bandwidth* to minimize downloading time. To minimize main CPU interaction, the flash interface unit supports DMA transfers from flash memory to system DRAM memory.

In SoC applications, the amount of on-chip ROM memory dedicated to booting is very limited. As a result, most or all of the boot information is typically stored in flash memory. A number of partitions for the boot loader code, the compressed root application or the flash file system are created in the flash memory.

Table 1. Properties

Devices	NOR	Small NAND	Large NAND
Approximate cost	$0.30/MB	$0.08/MB	$0.06/MB
Density/device	16Mb to 128Mb	64Mb to 2Gb	1Gb to 8Gb
Memory Access	Random Access XIP Support	Page Access No XIP Support	Page Access No XIP Support
Page Size	N/A	512 bytes/16 byte spare	2048 bytes/64 byte spare
Block Size	8KB/64KB	16KB (32 pages)	128KB (64 pages)
Read Access Time	65ns (first byte) 25ns (next 7 bytes)	50ns (cycle time) 10us (page time)	30ns (cycle time) 25us (page time)
Program Time	4us per byte	200us per page	200us per page
Block Erase Time	0.7 to 1.6 sec	2ms	2ms
Shipped with Bad Blocks	No	Yes	Yes
ECC Recommended	No	Yes	Yes

3. PERFORMANCE REQUIREMENT

Bandwidth for NAND, the peak read bit rate is

$$= \frac{n_p w}{t_{wc}\left(n_c + n_a\right) + t_r + t_{rc} n_p}$$

With

n_p the number of data accesses in a page

w the width of the data bus

n_c the number of commands to initiate a full page read, typically one.

n_a the number of address cycles to initiate a full page read

t_{wc} the write cycle time, used to send command and address to the flash

t_r the retrieval time to transfer the data into the page buffer (also called register for NAND)

t_{rc} the read cycle time per data access

3.1 Small Flash Bandwidth (x8 NAND)

DMP10 can be supported by a small NAND flash x8/x16. 6 Mbps write and 10 Mbps read. NAND Flash takes 10 us to access a page, and 50 ns to read each byte/word. Max x8 device FlashReadBW = 512 bytes/ (10 us + 512*50ns) = 14.4 MB/sec. Max x16 device FlashReadBW = 256 words/ (10us + 256*50ns) = 22.5 MB/sec. NAND Flash takes 200 us to program a page, and 50 ns to write each byte/word. Max x8 device FlashWriteBW = 512 bytes/ (200us + 512*50ns) = 2.27 MB/sec. Max x16 device FlashWriteBW = 256 words/ (200us + 256*50ns) = 2.41 MB/sec.

3.2 Large NAND Flash Bandwidth

DMP20 can be supported by a large NAND flash x8/x16. 19 Mbps write and 38 Mbps read. NAND Flash takes 25 us to access a page, and 30 ns to read each byte/word. Max x8 device FlashReadBW = 2048 bytes/ (25 us + 2048*30ns) = 23.7 MB/sec. Max x16 device FlashReadBW = 1024 words/ (25us + 1024*30ns) = 36.8 MB/sec. NAND Flash takes 200 us to program a page, and 30 ns to write each byte/word. Max x8 device FlashWriteBW = 2048 bytes/ (200us + 2048*30ns) = 7.83 MB/sec. Max x16 device FlashWriteBW = 1024 words/ (200us + 1024*30ns) = 8.88 MB/sec.

4. PARALLEL WRITE OF MULTIPLE NAND FLASH DEVICES.

The previous rate requirements were for one device. When multiple external devices are present, the aggregate write performance can be increased by concurrent use of multiple external devices. This can be done efficiently without resorting to full parallelization as explained below.

The following tables show numerical values for the BW of different operations. The table also shows a duty cycle, defined as the ratio between the amount of cycles where chip select is asserted to the total operation time and how many devices can operate on the bus without the bus becoming the bottleneck.

Table 2. Samsung K9xxG08UxM: 8 bits data bus, Np = 2k + 64, worst case T_{PROG} and T_{BERS}

Operation	T bus (ns)	T operation (ns)	BW (Mb/s)	Duty Cycle	Max Devices
Erase	225	40,000,225	27.0	0.00%	177,778
Write	53,050	753,050	22.4	7.04%	14
Read	52,975	72,825	232.0	72.74 %	1

Table 3. Same device, Samsung K9xxG08UxM, but typical case T_{PROG} and T_{BERS}

Operation	T bus (ns)	T operation (ns)	BW (Mb/s)	Duty Cycle	Max Devices
Erase	225	30,000,225	36.0	0.00%	133,334
Write	53,050	253,050	66.8	20.96 %	4 (almost 5)
Read	52,975	72,825	232.0	72.74 %	1

Table 4. Toshiba TC58NVG0S3AFT05: 8 bits data bus, Np = 2k + 64, worst case T_{PROG} and T_{BERS}

Operation	T bus (ns)	T operation (ns)	BW (Mb/s)	Duty Cycle	Max Devices
Erase	450	80,000,450	13.5	0.00%	177,778
Write	106,100	806,100	21.0	13.16 %	7
Read	105,950	130,650	129.3	81.09 %	1

Table 5. Same device, Toshiba TC58NVG0S3AFT05, but typical case T_{PROG} and T_{BERS}

Operation	T bus (ns)	T operation (ns)	BW (Mb/s)	Duty Cycle	Max Devices
Erase	450	40,000,450	27.0	0.00%	88,889
Write	106,100	306,100	55.2	34.66 %	2 (almost 3)
Read	105,950	130,650	129.3	81.09 %	1

When N devices are present, the peak rate can trivially be N times the above rate by providing independent channels to each device. However, if maximizing the write bandwidth is the goal, it is possible to increase the write bandwidth by a factor M using one bus and M chip select signals. This principle is shown in the figure 1.

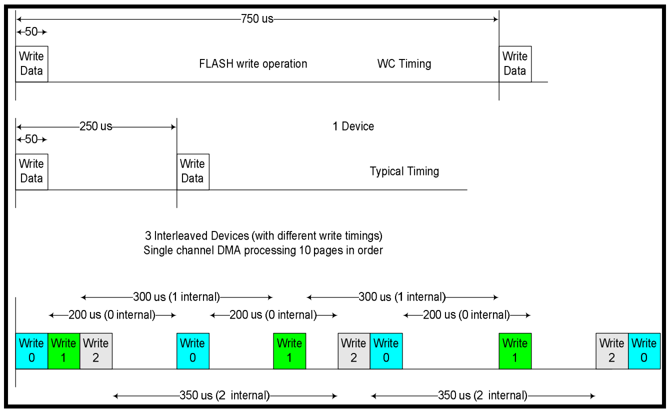


Figure 1. Flash writes operation.

The next table shows the peak write bandwidth in function of the number of devices on one data bus, bold value identify configurations where the bus is the bottleneck, not the devices. These values ignore the effects of collisions.

Table 6. Peak Writes Bandwidth

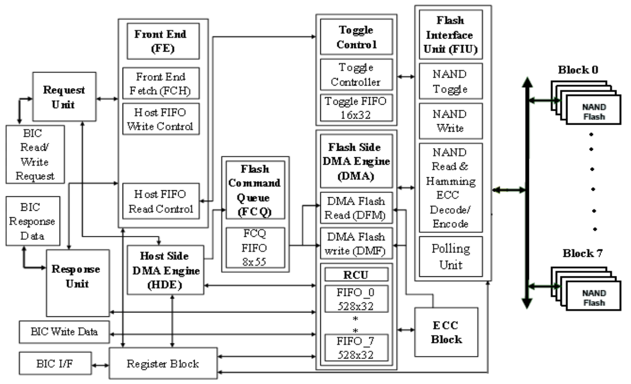
Device type	Samsung K9xxG08UxM		Toshiba TC58NVG0S3AFT05	
Device(s)	Worst case	Typical case	Worst case	Typical case
1	22.4	66.8	21.0	55.2
2	44.8	133.6	42.0	110.4
3	67.2	200.4	63.0	**160.0**
4	89.6	267.2	84.0	**160.0**
5	112.0	**320.0**	105.0	**160.0**
6	134.4	**320.0**	126.0	**160.0**
7	156.8	**320.0**	147.0	**160.0**
8	179.2	**320.0**	160.0	**160.0**

Based on these values and the requirements, the flash interface unit will operate with one NAND flash interface, supporting 8 chip select signals. To limit the number of signals, only one ready signal is used per interface and the status byte is polled to detect when a write operation is finished. The polling operation has low priority; especially polling doesn't take place if the DMA is transferring data to another device. Only DMA program operations are operated in this fashion.

5. FLASH CONTROLLER

The Flash Controller mainly comprises of toggle, DMA Engine and Flash Interface block.

Figure 2. NAND Flash Controller



5.1 Front-End Block

The Flash interface unit implements a small number of methods. Hence fetch, decode, validate, notify, package generation and emitter functionality is implemented in this block. Methods processed by DMA engine and toggle control are specified in class file. A Perl script will be written to generate verilog and F-model files. Front-end emits a package (DMA operation) on one bus to the DMA engine and bundles (toggle commands) on another bus to the toggle control block. Front-end waits for acknowledge from the engines that consume the packages before sending a new package or bundle.

5.1.1 Host

This block implements a 64-entry FIFO (64x64) to store the commands that are fetched from memory. Commands are fetched by performing DMA operation (getPtr/putPtr). The flushing semantics impacts the combination of FIFO in the host and the rest of the command buffer in memory.

The host fetches commands from memory when getPtr and putPtr are different and fetch is enabled. In particular, S/W is allowed to modify the putPtr at any time.

The host will forward commands to the execution engine in the following way. When the FIFO is not empty, the host checks the command at the head of FIFO and operates. If a block is currently active and the next command is not for the same block, wait the block becomes inactive before attempting to issue the next command. This insures that toggle and DMA are never active simultaneously. It also insures that an interrupt is only generated after a DMA transfer is fully completed.

An inactive block is assumed to be able to accept any command. An active block indicates provides an indication to the host about which commands it is able to accept (flow control). This is especially needed to insure parallel operation of the DMA write; the DMA provides at least 8 flags to the host, each flag indicating that the channel is able to accept a new command.

5.1.2 Notify

Notification is supported via flags in H/W registers and by interrupts. In the list of events previously notified, the DMP10 alternate support is shown in *italics*.

A 32-bit free running counter (wraps at ~43secs at 100MHz) is maintained to timestamp the notification messages. Notification is sent to software on the occurrence of the following events:

Upon reception of a notify command (P1). *Only an interrupt command as P0.*

Upon detection of a correctable error when connected to a NAND flash (P1). *Error captured in error log register as P0.*

Upon detection of an uncorrectable error when connected to a NAND flash (P1). *Error captured in error log register as P0.*

Upon detection of programming errors or read errors by the flash device (P1). *Only timeout interrupt as P0.*

Upon reception of a DMA command that overlaps multiple pages in a NAND flash device. *Only interrupt during execution of DMA command as P0.*

Upon reception of a DMA command that exceeds the supported size of the flash device. *Only interrupt during execution of DMA command as P0.*

5.1.3 Request Arbiter

Arbitrates between the three memories requesters: Front-end fetch, DMA engine and Notify block (P1). The order of priority in case of multiple requests arriving in the same clock is: Notify followed by Front-end fetch followed by DMA.

5.2 DMA Engine

DMA engine supports the following features:

Transfer size limited to 2KB maximum for large page NAND, and 512B for small page NAND flash devices.

Breaks down DMA transfers into smaller transfers depending on the size of bursts supported by the interface. The bursts should be as long as possible (64 bytes).

Support multiple outstanding read and write messages towards the memory controller so that memory controller can utilize memory bandwidth efficiently.

All data transfers are multiple of 32 bits and naturally aligned for ease of implementation.

Supports multiple write and erase program in parallel.

The DMA starts transferring data upon reception of the DMA packet (consisting of command type, source address, destination address and size) from the front-end.

DMA inside a NAND has some specific characteristics: The flash address is used to calculate a start page and the target device, the address passed to the NAND is a combination of page address and offset in the page. When needed the DMA interact with the ECC block to either generate the ECC bytes to program in the spare area or to correct data present in the data buffer. ECC errors are logged. For correctable errors, the correct amount of data is transferred into memory.

The DMA processes commands sequentially as fast as possible and overlaps DMA write and erase commands for best performance.

5.3 DMA Buffer

The data buffer is implemented as 8 blocks, each of (512 + 16) x 32 SRAM, managing the flow of information between DMA, enhanced ECC block and the external flash devices. The data buffer is statically provisioned with 2K + 64 bytes (one large page including spare area) per external flash device. The ECC block needs access to the DMA buffer, but only one of DMA or ECC is allowed access to one of the page buffer at any one time.

The normal sequence of operation is begin from DMA command results in data transfer into one of the DMA buffer, either from external DRAM or from flash. DMA pass the DMA buffer to ECC if needed, ECC either encodes or decodes the codeword. ECC signals to DMA that processing is done; control of DMA buffer is given back to DMA. DMA finishes the DMA command, transferring DMA buffer to either flash or external DRAM RAM Parameters like single cycle read and write access, byte enables for write and shared read/write port.

6. NAND INTERFACE WRITE BLOCK

The NAND interface write block is a part of DMA page program operation. The complete DMA page program operation is done through the NAND interface write block and NAND interface polling block. The write FSM is active up to the page program confirm command (10h) which initiates the programming process/cycle. The NAND interface write block contains two write state machines. Main write state machine is used to controls the main field area and spare field area. NAND write sub-state machine is used to control the exact write process both for main field area as well as spare field area and is dependent on write state machine [3, 4].

7. EXPERIMENTAL SETUP AND RESULTS

In this design, we have studied the logic implementation ideas used in the NAND - Flash 1.2-Flyer document from Arasan. In which the NAND flash interface provides an 8-bit or 16-bit interface to the flash memories. A total of 4 memory banks and 8 flash memories per bank are supported. Each memory bank provides up to 8 chip selects signals. The interface supports a maximum of 64 Gbytes of NAND flash memory [12].

From this concept the controller designs for the 8-bit interface to the NAND Flash memory to support parallel write operation. The NAND flash Verilog model-MT29F (Micron Technology, Inc.) is used for verification and for simulation- Modelsim SE 5.8 is used with 2.8GHz Pentium IV processor, 512 MB RAM running Windows XP system [10,11].

7.1 PAGE PROGRAM

The device is programmed basically on a page basis, but it does allow multiple partial page programming of a byte/word or consecutive bytes/words up to 528(X8 device) or 264(X16 device), in a single page program cycle. The number of consecutive partial page programming operation within the same page without an intervening erase operation should not exceed 2 for main array and 3 for spare array. The addressing may be done in any random order in a block. A page program cycle consists of a serial data loading period in which up to 528 bytes(X8 device) or 264 words(X16 device) of data may be loaded into the page register, followed by a non-volatile programming period where the loaded data is programmed into the appropriate cell. The serial data loading period begins by inputting the Serial Data Input command (80h), followed by the three cycle address input and then serial data loading. The words other than those to be programmed do not need to be loaded. The Page Program confirms command (10h) initiates the programming process. Writing 10h alone without previously entering the serial data will not initiate the programming process. The internal write controller automatically executes the algorithms and timings necessary for program and verify, thereby freeing the system controller for other tasks. Once the program process starts, the Read Status Register command may be entered, with RE and CE low, to read the status register. The system controller can detect the completion of a program cycle by monitoring the R/B output, or the Status bit (I/O 6) of the Status Register. Only the Read Status command and Reset command are valid while programming is in progress. When the Page Program is complete, the Write Status Bit (I/O 0) may be checked (Figure 5). The internal write verify detects only errors for "1"s that are not successfully programmed to "0"s. The command register remains in Read Status command mode until another valid command is written to the command register [3, 4].

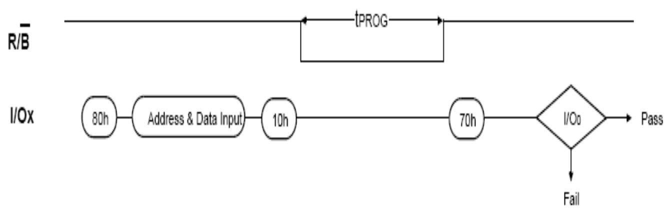


Fig 4: Program Operation

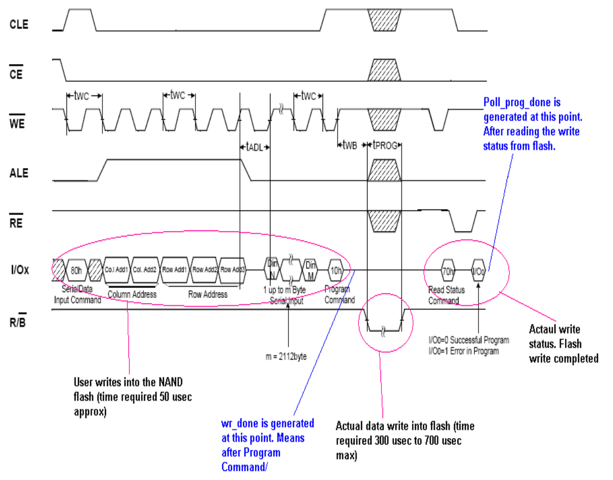

CLE
CE
WE
ALE
RE
I/Ox
R/B
tWC
tADL
tWB
tPROG
80h
Col Add1
Col Add2
Row Add1
Row Add2
Row Add3
SerialData Input Command
Column Address
Row Address
1 up to m Byte Serial Input
Program Command
70h
I/O0
Read Status Command
m = 2112byte
I/O0=0 Successful Program
I/O0=1 Error in Program
Poll_prog_done is generated at this point. After reading the write status from flash.
Actaul write status. Flash write completed
User writes into the NAND flash (time required 50 usec approx)
wr_done is generated at this point. Means after Program Command/
Actual data write into flash (time required 300 usec to 700 usec max)

Fig.5: Page Write Cycle

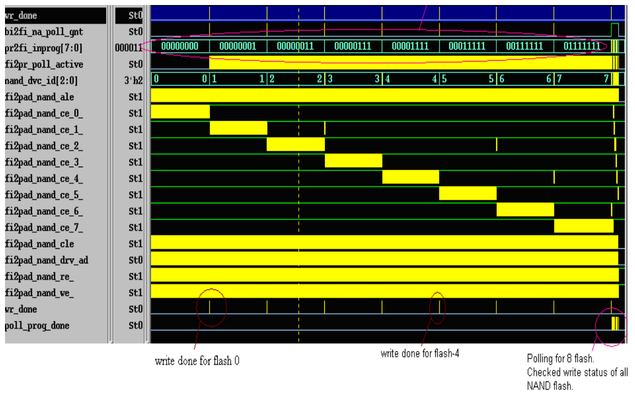


Fig.6: 8-Flash Parallel Programmed

8 NAND flash are programmed by Parallel Write method. The actual flash data write time is more, so the same time is utilized to program the next flash. First priority is given to the user to write the flash and second priority is given for polling to check the write status of each flash.

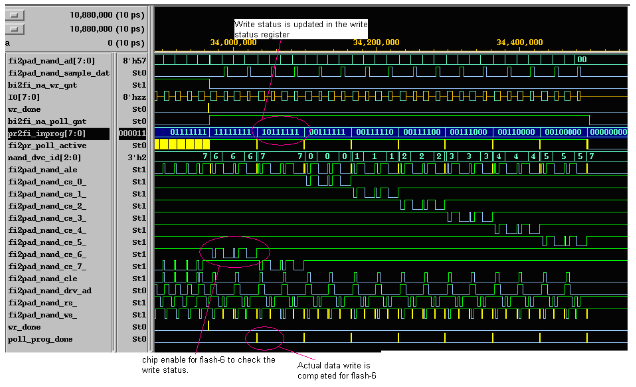

Write status is updated in the write status register
34,000,000
34,200,000
34,400,000
chip enable for flash-6 to check the write status.
Actual data write is competed for flash-6

Fig. 7: 8-Flash Parallel Programmed Polling Cycles

The actual data time into flash is 300 microseconds to 700 microseconds. In this case, we need less than 50usec to check the actually write completion status of each flash.

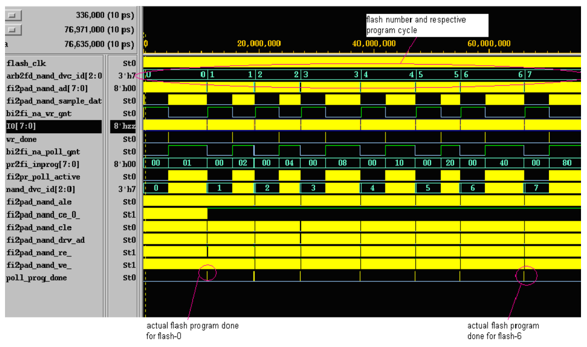

flash number and respective program cycle
20,000,000
40,000,000
60,000,000
actual flash program done for flash-0
actual flash program done for flash-6

Fig. 8: 8- Flash Programmed

8 NAND flash are programmed serially, one bye one as shown above. Consider typical actual flash program time of 300 usec.

Time required to program one flash =300+50=350 use. Total time required to program 8 NAND Flash=350*8=2800 usec (approx). Conclusion- More time for actual data write into flash. We can utilize this time to program another flash.

8. CONCLUSION

The NAND Flash Controller has only one I/O port and command/address/data multiplexed I/O port in addition to the simple command sets which are provided for the different operations. System uses huge amount of program time for another NAND flash program, so the total program time required for all eight NAND flash is very less as compared to eight individual NAND flash programming.

9. REFERENCES

[1] Yet Another Flash File System (YAFFS)
http://www.aleph1.co.uk/yaffs/

[2] Journaling Flash File System 2 (JFFS2)
http://sourceware.org/jffs2/
http://www.linux-mtd.infradead.org/

[3] Samsung K9K1208U0C 64Mx8/32Mx16 NAND Flash Memory 528 Byte/Page Data Sheet.
http://www.samsung.com/Products/Semiconductor/Flash/NAND/512Mbit/K9K1208U0C/ds_k9k12xxx0c_rev31.pdf

[4] Samsung K9K2G08U0A 256Mx8/128Mx16/512Mx8 NAND Flash Memory 2112 Byte/Page Data Sheet.
http://www.samsung.com/Products/Semiconductor/Flash/NAND/2Gbit/K9K2G08U0A/ds_k9k2g08ua_rev02.pdf

[5] Spansion S29GLxxxM MirrorBit Flash Family Data Sheet
http://www.spansion.com/datasheets/s29glxxxm_00_b4_e.pdf

[6] Toshiba TC58FVM7T2A 16Mx8 NOR Flash Memory Data Sheet.
http://www.toshiba.com/taec/components/Datasheet/58fvm7x2a.pdf

[7] Intel StrataFlash® Embedded Memory (P30) 64Mb – 1Gb Datasheet
http://download.intel.com/design/flcomp/datashts/30666601.pdf

[8] AN1823 APPLICATION NOTE Error Correction Code in Single Level Cell NAND Flash Memories
http://www.st.com/stonline/books/pdf/docs/10123.pdf

[9] I^{2}C Bus Specification Version 2.1
http://www.semiconductors.philips.com/acrobat_download/literature/9398/39340011.pdf

[10] Micron technical note 2906: ECC Module for Xilinx Spartan-3
http://download.micron.com/pdf/technotes/nand/tn2905.pdf

[11] Micron technical note 2908: Hamming Codes for NAND Flash Memories
http://download.micron.com/pdf/technotes/nand/tn2908.pdf

[12] NAND Flash Controller IP Core Data Sheet
http://www.arasan.com/datasheets/login.php