Proposal for NAND Flash Failure Analysis Tool

Sze-Hsiang Harper · sze.harper@utah.edu

Jacob Johns · JacobJohns@comcast.net

Hartman Rector · hartmandrector@hotmail.com

Cory Shirts · c.shirts@gmail.com

Michael Tomer · Jester3141@gmail.com

Table of Contents

Introduction

The demand for NAND flash memory has increased worldwide, made possible by decreased manufacturing costs and greater memory density in devices. Flash memory is also an attractive option because it requires a small amount of power. This technology is beneficial for devices such as digital photography, USB drives, cell phones, and embedded devices. However, current technology allows guaranteed operation until 100,000 program-and-erase cycles. Little data is available beyond this mark to identify failure rates after this specified limit.

The use of flash technology can introduce permanent as well as temporary memory failures, which increase over the lifespan of a device and will eventually lead to a total memory failure. Failure occurs in anywhere from a single bit to a complete memory block. These failures are intensified by specific use patterns, such as partial page programming, high block read cycle count, and high block erase/program count. The 100,000 program-and-erase cycle guarantee mentioned above uses basic Error Code Correction (ECC) algorithms that attempt to minimize the effect failures have on normal operation [1]. The aim of this project is to produce an open source tool for analyzing NAND Flash failure rates in order to provide a means to analyze current technology. The analysis that our tool enables will conceivably provide data that may assist in the development of more advanced ECC algorithms and improved manufacturing processes.

Project Tasks

We have divided the project tasks according to the various components and assigned team members to each task:

- Cory – Documentation
- Michael – GUI Design
- Sze-Hsiang – USB Controller, Interface
- Jacob – Daughter Board/Card and Field Programmable Gate array (FPGA)
- Hartman -- Database Interfaces

Each team member will be responsible for developing and documenting the interfaces used by the components they are assigned to. The system architecture involves the overall view of how the various Verilog components connect and interface to each other on the FPGA, daughter card and to the flash device itself.

Specific Task Interfaces

NAND Flash Module and Controller

Micron will provide a daughter card that holds the Flash device under test. It connects to an Altera DE2 FPGA board. The controller for the daughter card on the FPGA was written by last year's team and was developed in Verilog. The module will receive commands from the FPGA, which are described in the next section.

Field Programmable Gate Array (FPGA)

We will use an Altera DE2 FPGA development board to interface between the GUI and the daughter card holding the memory chip. To interface with the GUI will use the Phillips ISP1362 USB controller chip built on the DE2 board. We are planning to use the NIOS 2 soft processor on the FPGA so we can take advantage of pre-existing device drivers to make the interfacing with the USB more manageable. The FPGA will also implement a flash controller written in Verilog that will communicate directly with the daughter card. We will use a shared memory model that will allow the flash controller to read and write in memory that is directly accessible by the NIOS 2 soft processor. The FPGA will have to accept seven commands from the GUI. These commands will be:

1. Erase.
2. Check for erase failure flag (via Read Status Command).
3. Verify an erase status failure by reading the failing block and checking for all 0's.
4. Program page.
5. Verify programming was successful (via Read Status Command).
6. Read Page (discard what is read- for read disturb cycling).
7. Read Verify - compare/verify a block against originally programmed pattern.

The processor will receive an interrupt from the USB controller indicating the command has been sent to it on the USB. It will then parse the commands into a sequence of instructions to be sent to the flash controller. The instructions will be held in a FIFO queue to be accessed by the flash controller. The FPGA will then allocate memory for the flash controller to store its results, and then wait for an interrupt from the flash controller indicating that the commands have been executed. The processor can then gather any results that are needed and send this information back to the host pc over the USB. The flash controller will be designed in Verilog and will access commands from the FIFO queue and send the command to the daughter card using the interface specified on the micron data sheets. It will then receive the results of the command and write them back to the shared memory that has been allocated by the processor. If we have extra time other flash controllers could be written that would perform identical operations for separate devices allowing them to be tested and compared simultaneously.

Each component of the FPGA design can be tested individually before being integrated. The communication with the FPGA and GUI can be tested by sending a command from the GUI to the FPGA. A simple echo program can be written on the FPGA that will send the command back to the GUI where it can be verified. The FPGA could also be programmed to send a page of results across the USB to the GUI. We can assume that the NIOS 2 soft processor works, and the C code used to program it can be tested on a PC. The communication between the processor and flash controller can be tested using simulation tools. The flash controller design can also be tested using simulation tools. A sample simulation from the 2007-2008 clinic team is shown below.

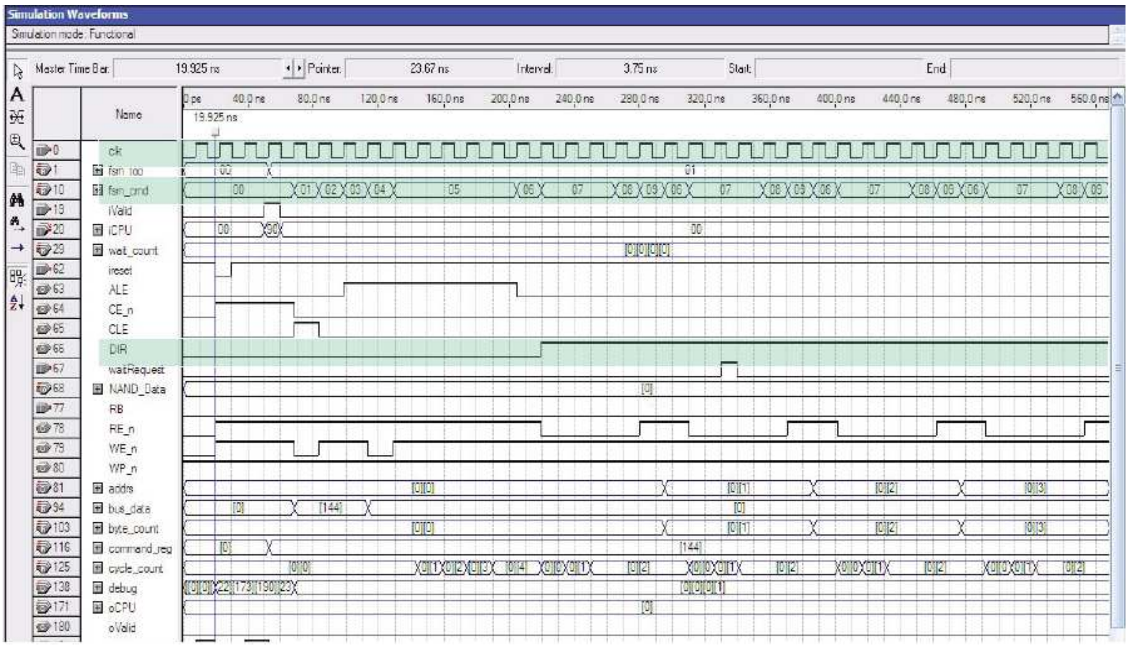


Figure 1 – Sample waveform simulation

Using simulation tools each command can be simulated and the output signals can be verified. When the FPGA design is complete, a command can be sent from the GUI and the results of the command can be observed at the I/O pins using a scope to insure that the command is being executed. An example of this from the 2007-2008 clinic team's website (http://code.google.com/p/2007-uofu-micron-clinic/) is shown below.

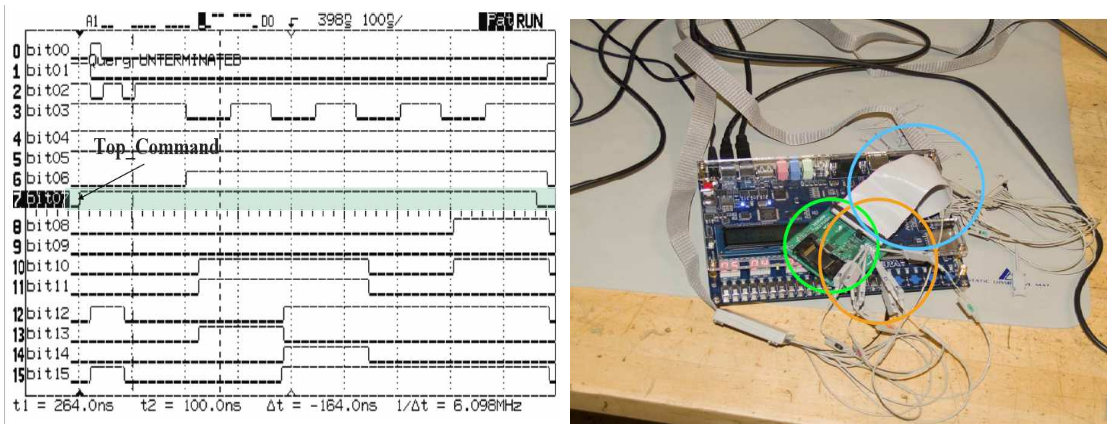


Figure 2 – Waveform and FPGA Board

Universal Serial Bus (USB)

Currently USB types generally support three bus speeds: high speed (480 Megabits/sec), full speed (12 Megabits/sec), and low speed (1.5 Megabits/sec). The USB host drivers in all recent PCs support all three speeds. In our project there are two locations that the USB will be used: USB 2.0 driver on PC and on the FPGA. This project utilized a design for the USB with the Host PC communicating with the firmware through the USB ports on the PC and DE2 development Board. Data and commands are transmitted to and from the firmware using the USB 2.0 Full speed protocol with 16 byte bursts of data.

When a command is received over the USB, an IRQ is generated, and then the processor jumps to the USB Interrupt Service Routine (ISR). In the ISR, data sent from the Host PC is received and parsed into the Op-Code, Bit Pattern Generation Algorithm, Start Address, End Address and Number of Cycles. The USB firmware receives a 32 byte command from the GUI and sends the results back to the Host PC up to 2112 bytes for read page by using as much as USB full-speed bandwidth.

Commands are executed in the order that they are received. The USB firmware issues the commands to the NAND Flash Controller. The results of the commands are then sent back to the host PC. The USB firmware repeats this until all commands have been executed.

The communication between the host PC and the FPGA is achieved through the USB interface. We will divide these interfaces as follows:

Front-end Graphical User Interface (GUI) on PC⇔ USB2.0 driver on PC

On the PC, we will be using the libusb-win32 library version 0.1.12.1 to establish bi-directional communication with the FPGA. The libusb-win32 library is a Windows port of the libusb generic driver for Linux/Unix systems. The library serves as a wrapper to a generic USB driver for Windows XP. There is an actively developed C language wrapper for the libusb-wind32 library called LibUsbDotNet [8]. This wrapper allows us to develop GUI code on the C#.NET platform and interact with the device through USB without using native device drivers. Using this library, we will be able to talk to the bus using code developed and executed in the user space and avoid the potential hazards of developing a custom or kernel space driver.

C Firmware on the FPGA ⇔ USB 2.0 driver on the FPGA

On the FPGA, the DE2 FPGA development board [2] has the Nios II soft processor available and compatible with the USB 2.0 specification. The USB on the FPGA supports full speed communication in12Mbits/sec. The bus must carry status, control, and error-checking signals. The data-transfer rates attainable at full speed will provide sufficient bandwidth for communication of commands and results. The FPGA will be able to process packets according to the USB 2.0 specification. It will interface with the libusb-win32 and C# code written for the GUI.

We are concerned there might be some restrictions during the process to allow the USB download to speed up. The USB controller can only support the High-Speed USB protocol, a maximum of 12Mbits/sec transfer. This will restrict the NAND Flash read transfer rates. For example, a 4GB NAND Flash device will require at least six minutes to transfer a full image. If block or page size transfers the majority of data composed in these transactions, this may not be an issue. Also, repetitive erase or write operations can be performed by the embedded microprocessor on the DE2 at device speeds rather than downloading images or patterns repeatedly over the USB interface. Other techniques, such as using an on-board DE2 SDRAM to store patterns may enhance cyclic write operations.

SQL Database

The results of the tests will be transmitted to the computer through the USB interface in XML format. The use of the XML format enables greater flexibility. An example of a result is shown in figure 3.

These results will be parsed and then stored in a Transactional SQL database. Transactional SQL database is an additional programming layer built onto the standard SQL command structure and has several improvements to a standard SQL database. T-SQL adds support for flow control and local variables. Flow control gives the ability to use commands such as IF, ELSE, and WHILE to give conditional results. Local variables can be used as counter variables in combination with flow control. T-SQL also has several improvements to existing SQL commands. In addition, the T-SQL DELETE and UPDATE statements allow joins from multiple tables.

The SQL database does not need to be installed on to the host PC. Instead, it can be installed on a server. This will allow a great degree of flexibility because multiple computers will be able to be testing flash memory simultaneously. This will ensure a larger sample size when analyzing failure patterns.

It is important that structure of the data being stored in this database will be normalized. Normalization of a database structure makes sure that the database is suitable for general purpose use and ensures that it is free of several flaws in non-normalized databases that could lead to loss of data integrity.

Initially this database will only store a few basic pieces of information such as the block, command, result, etc. The database may be expanded to store data for additional variables such as temperature and operating voltage. Any other custom reports or graphs that are not included in our GUI will can be created through the use of Microsoft Access. Access can interface with the SQL database and will allow for a more detailed analysis of the data.

```
        <job id=”4” opcode=”128” seed=”0” algorithm=”0” cycles=”10000” startAddress=”0x00000000”
endAddress=”0x000FC000” debug=”0”>

                <data>U3VjayBteSBiYWxscyE=</data>

                <error count=”1” address=”0x0x000FC000”>

                        <byte index=”23” received=”35”/>

                        <byte index=”444” received=”255”/>

                </error>
```

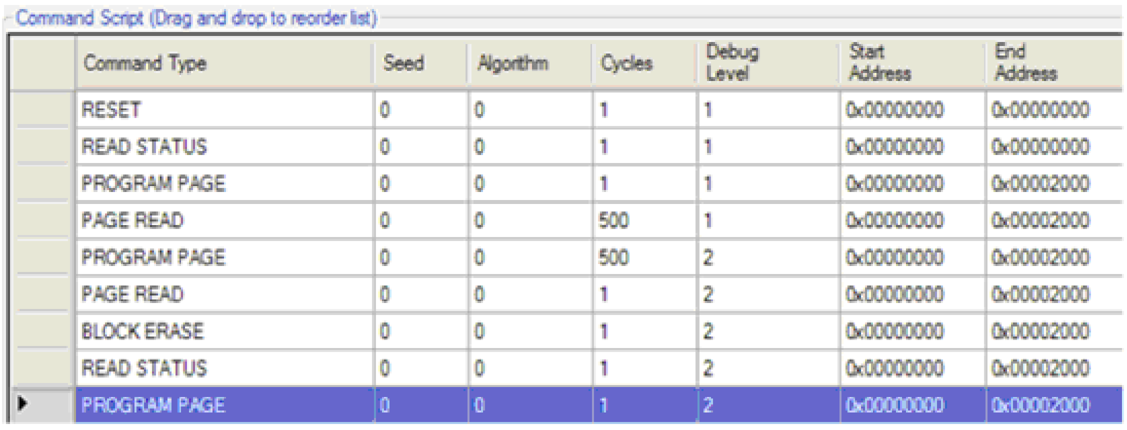
Figure 3

Graphical User Interface (GUI)

The GUI on the host PC will allow the user to run tests on the flash memory. The GUI will also be responsible for programming the FPGA and loading the firmware into memory. The GUI will need to provide an interface for connecting to and testing communications the SQL database. The GUI will use the ActiveX Data Objects Classes of the .NET framework to communicate with the database.

The user will be able to program series of commands to execute on the flash memory. Reading, writing, erasing, and getting status are the basic commands. The interface for this will need to be sufficiently broad to set up automated testing patterns that could run for a substantial amount of time without user interaction. **Figure 4** shows and example of an automated testing script.

The GUI will allow the user to save and load these script files. The script files are saved in the XML format. These scripts will also allow the user to set up complex test patterns in which they can specify: ranges of blocks to test, duration of test, specific memory patterns, random memory patterns, and debug levels. The debug level will allow the user to dump additional information for specified commands to a log file. This can provide extra information about certain commands that won't be stored in the SQL database.



Command Script (Drag and drop to reorder list)

Command Type	Seed	Algorithm	Cycles	Debug Level	Start Address	End Address
RESET	0	0	1	1	0x00000000	0x00000000
READ STATUS	0	0	1	1	0x00000000	0x00000000
PROGRAM PAGE	0	0	1	1	0x00000000	0x00002000
PAGE READ	0	0	500	1	0x00000000	0x00002000
PROGRAM PAGE	0	0	500	2	0x00000000	0x00002000
PAGE READ	0	0	1	2	0x00000000	0x00002000
BLOCK ERASE	0	0	1	2	0x00000000	0x00002000
READ STATUS	0	0	1	2	0x00000000	0x00002000
PROGRAM PAGE	0	0	1	2	0x00000000	0x00002000

Figure 4 – Sample Test Script

The GUI will also provide an interface for viewing the results. Several different visualization methods will be integrated into the interface. One of the visualizations we are planning on implementing will be a bitmap representation of the flash chip, with each pixel representing a bit of memory. Different colors of pixels will be used to denote a binary 1, 0, or an error as shown in F**igure 5**. This will quickly allow the user to see if certain blocks are more prone to failure than others. We are also planning on implementing several graphs and other hierarchal representations of the results. Engineers will analyze these pictures and graphs from various chips and look for trends in the data. The GUI will also allow the user to output the raw data of the results to file.

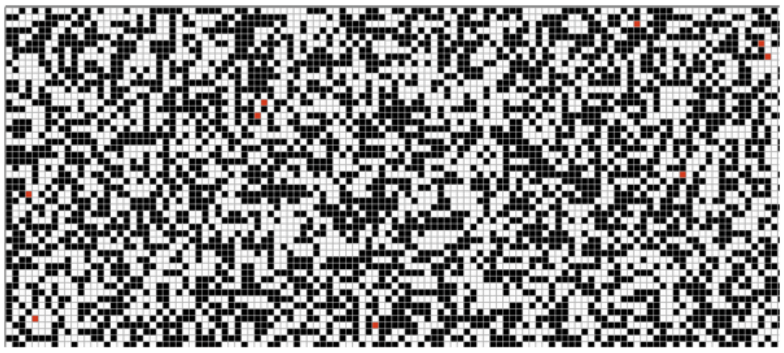


Figure 5 – Sample Block Tests Output

Schedule Flow

Preliminary estimates are shown in the table below in how we will complete this project. The Component Design Phase will consist of the team getting familiar with existing code and libraries as well as planning each component's implementation. After each component has been completed, integration and testing will begin, which will continue until a working product needs to be shown to Micron. Then beginning in the New Year, data will be collected from experiments performed. This will ensure that the system is robust and will also allow us to add any new features that the system might need or any features that were requested by Micron. After two months, we will present this project and initial test results at a technical open house.
The Implementation Phase involves the development of the GUI and the FPGA. These can be done concurrently, but we will focus working on things from the bottom up. For example, the FPGA development will begin with the USB interface and pin outs to the NAND device. After these have been tested and are working the controller can be developed to connect them, the entire FPGA section can then be verified and tested with the GUI. This schedule is illustrated in **Figure 6.**

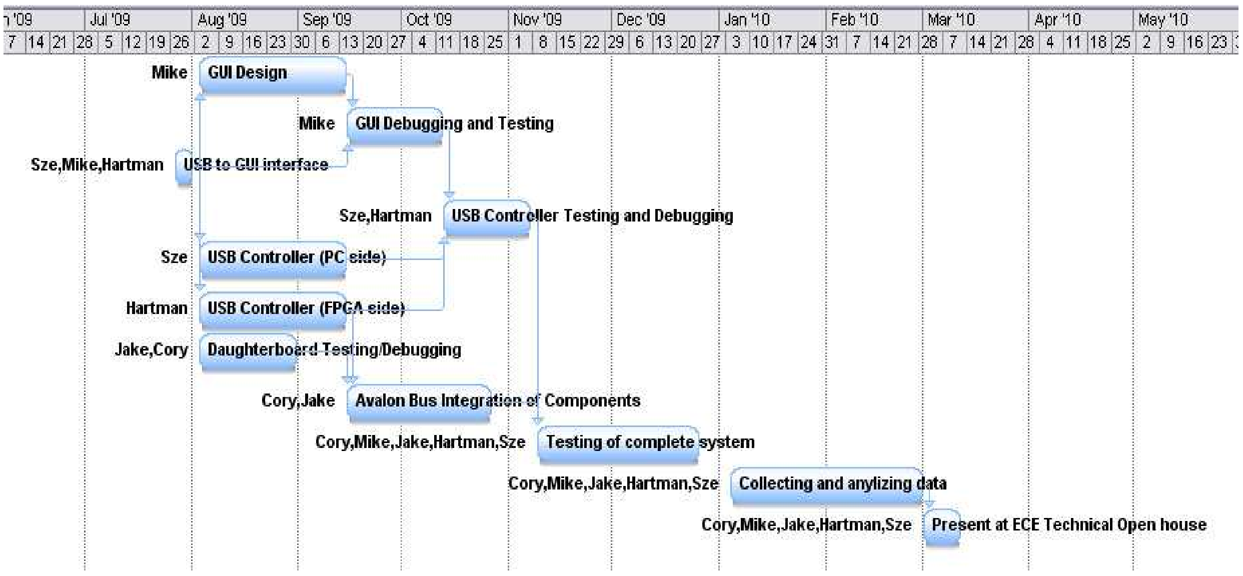


Figure 6 – Implementation Schedule

Testing and Integration Strategy

We will test each individual component of the system as we go. We will be sure to have at least two people working on one part of the project at anytime, so that when we integrate later there will be more than one that has knowledge of the different sections. This should allow for better and faster integration.

The tentative schedule we have planned so far is that we will first start on the USB interfaces between the different devices that we are going to need to communicate with, testing the initial daughterboard that was created by last year's team to make sure it works and that we can understand how it works a little better to better be able to implement the remaining parts of the project.

We will start early on the biggest part of our project, which is the GUI, and getting it so it will do all the different tests and print them in a format that can be easily read by those that would like to see the results. We will then come together to integrate the whole project together and begin extensive testing on the final project. Once we are convinced that the project works the way we believe it should we will begin to test some NAND devices, so that we can have some results by the time we are completely done with the project.

Group Management and Communication Plan

We will set up a website to deposit all code and to keep the developers as well as anyone that needs to know up to date with our progress. We will have weekly meetings where each team member will report on their progress, and then the notes from the meeting will be posted on the web page.

We will also post our progress as we go and any decisions that we make along the way. Part of posting our progress along the way will be a daily e-mail giving a very short description of what you, or if you worked in partners, what you and your partner did that day and what you plan on doing tomorrow. This will also be an excellent time to bring up any obstacles any team member might be having so that the rest of the group is aware of it and so we can divide up the resources accordingly, in order to finish each part of the project by the scheduled time.

Risks

One big challenge for this project is taking time to understand the existing code and hardware; we don't know how complex the process will be of comprehending it. It will also take time to understand and debug it.

Some of the main hardware risks will have to do with the availability of existing designs for the soft processor and the device drivers, because we need to design these by hand, which will require a lot of time and effort. There is also the risk with the existing designs not being fast enough to meet the timing constraints of the NAND device. There is also the issue of the speed of the entire system, if it is too slow then it might be impractical to test the NAND device to obtain failure data. Problems could also be encountered during testing, which include probe capture time limitations, and the large amount of data that needs to be watched.

Vendor List

The following hardware and software components will be used to complete this project:

NAND Flash Daughter board – Provided by Micron
NAND Flash SLC chips – Provided by Micron [1]
Altera DE2 FPGA development board – Provided through the University of Utah [2]
TortoiseSVN 1.4.5 subversion client – Free Download [3]
Subclipse plugin for Eclipse/NIOS IDE – Free Download [4]
Altera Quartus II Web Edition Verilog development environment – Free Download [5]
Altera Nios II Embedded Design Suite – Free Download [5]

LibUSBDotNet – C# LibUSB-Win32 wrapper for generic USB drivers – Free Download [8]
Visual Studio 2005 - IDE for C# GUI development – VS 2005 Express download [6]
Microsoft SQL 2005 Server - SQL database engine to store results -- MS SQL Express [7]

Bill of Materials

NAND FLASH VENDOR INFORMATION			
I	**Micron Technology**	contact:	Dennis Z.
8000 S Federal Way, Boise, ID 83706		F: 651-994-8186	P: 208-994-8200
Part # :	29F2G08AAC	2 Gbit w/ 8 bit I/O	
Time:	3-5 Days	Price/Unit Cost:	$0.00
Quantity:		Total Cost:	$0.00
II	**Luscombe Engineering Company Inc.**		
6465 S 3000 E Suite 101, SLC, UT 84121			P: 801-944-1280
Part # :	JS29FO4G08AANB1	2 Gbit w/ 8 bit I/O	
Time:	3-5 Days	Price/Unit Cost:	$ 0.00
Quantity:	1	Total Cost:	$ 0.00
FPGA BOARD VENDOR INFORMATION			
I	**Altera**	contact:	university@altera.com
357 S McCaslin Blvd, Louisville, CO 80027		F: 303-926-4945	P: 303-926-4955
Part # :	DE2	DE2 Development & Education Board	
Time:	3-5 Days	Price/Unit Cost:	$269.00
Quantity:	1	Total Cost:	$269.00

References

[1]**Micron NAND Flash 101 Technical Notes -**

http://download.micron.com/pdf/technotes/nand/tn2919.pdf2

[2]**Altera DE2 Board, http://www.altera.com/education/univ/materials/boards/unv-de2-board.html**

[3] **TortoiseSVN Client, http://tortoisesvn.net/downloads**

[4] **Eclipse Subversion plugin, http://www-128.ibm.com/developerworks/opensource/library/os-ecl-subversion/**

[5] Altera Software https://www.altera.com/support/software/download/altera_design/quartus_we/dnl-quartus_we.jsp

[6] Visual Studio Express, http://msdn.microsoft.com/express/

[7] MS SQL Express, http://www.microsoft.com/express/sql/download/

[8] LibUsbDotNet, http://sourceforge.net/projects/libusbdotnet/