FPGA Flash Memory High Speed Data Acquisition

April Gonzalez[1]
Undergraduate Student Research Program Intern, Houston, Texas, 77058

Nomenclature

ADC	=	analog to digital converter
DAC	=	digital to analog converter
FPGA	=	field programmable gate array
Gb	=	gigabyte
ONFI	=	Open NAND Flash Interface
MSB	=	most significant bit
MSPS	=	mega samples per second
Vdd	=	Supply voltage

I. Abstract

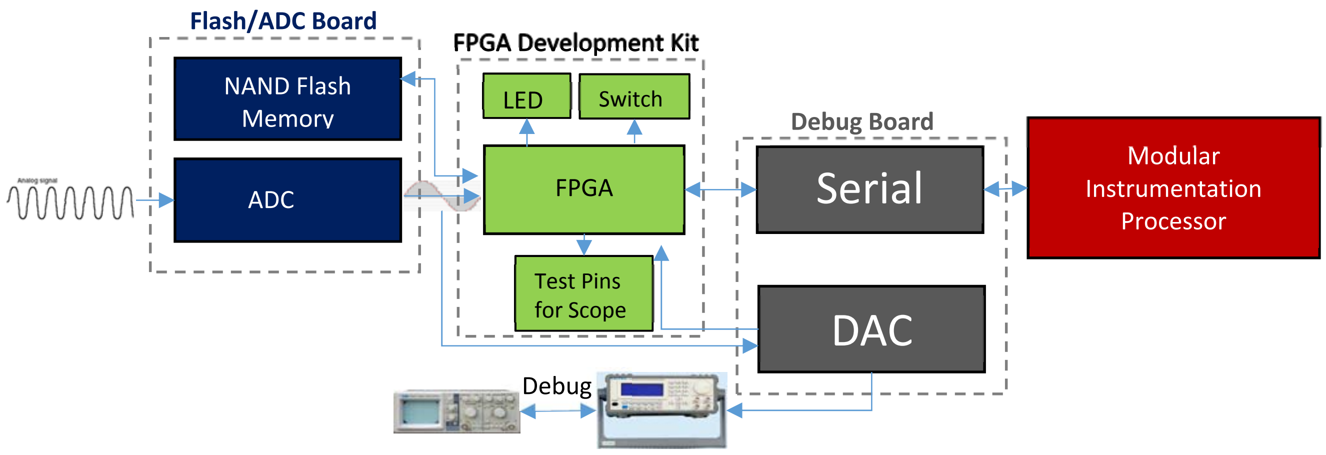
The purpose of this research is to design and implement a VHDL ONFI Controller module for a Modular Instrumentation System. The goal of the Modular Instrumentation System will be to have a low power device that will store data and send the data at a low speed to a processor. The benefit of such a system will give an advantage over other purchased binary IP due to the capability of allowing NASA to re-use and modify the memory controller module. To accomplish the performance criteria of a low power system, an in house auxiliary board (Flash/ADC board), FPGA development kit, debug board, and modular instrumentation board will be jointly used for the data acquisition. The Flash/ADC board contains four, 1 MSPS, input channel signals and an Open NAND Flash memory module with an analog to digital converter. The ADC, data bits, and control line signals from the board are sent to an Microsemi/Actel FPGA development kit for VHDL programming of the flash memory WRITE, READ, READ STATUS, ERASE, and RESET operation waveforms using Libero software. The debug board will be used for verification of the analog input signal and be able to communicate via serial interface with the module instrumentation. The scope of the new controller module was to find and develop an ONFI controller with the debug board layout designed and completed for manufacture. Successful flash memory operation waveform test routines were completed, simulated, and tested to work on the FPGA board. Through connection of the Flash/ADC board with the FPGA, it was found that the device specifications were not being meet with Vdd reaching half of its voltage. Further testing showed that it was the manufactured Flash/ADC board that contained a misalignment with the ONFI memory module traces. The errors proved to be too great to fix in the time limit set for the project.

II. Introduction

The purpose of this research is to develop a debug board and memory controller for a Modular Instrumentation System, Fig. 1. The system will be low power, capable of storing data, and able to send the data at a low speed to a processor. The four channel input analog signals have a throughput of 1 MSPS for 10 seconds which accounts for roughly 1.2GB of data. The Flash/ADC board contains a 16-bit analog to digital converter, AD7980, with the Open NAND Flash Interface (ONFI) memory module. Initially the project specification called for a 1 MB SRAM memory, however though it is noticeably easier than the more complex ONFI flash, the SRAM fell 3 orders of magnitude short of the required system capacity. To verify understanding of the ONFI, test routines based on the memory module are created using compatible Actel software Libero to program the FPGA Development Kit in VHDL. The Microsemi/Actel IGLOO series FPGA is used over low power processors due to the processors not

[1] USRP Intern, EV5, NASA Johnson Center, and University of Texas at Pan American.

being able to handle the data rate speed required for the system. The READ, WRITE, READ STATUS, ERASE, and RESET test routines generated for the FPGA to control the ONFI module can be externally tested using switches, LEDs, and I/O connectors located on the development kit. Successfully testing the Flash/ADC board and FPGA could not be accomplished due to a misalignment error in the placement of the ONFI module during layout creation. The error could not be fixed in the time constraints placed on the project. The debug board is the only board that is not manufactured, and contains a 16-bit digital to analog converter (DAC) along with a serial port. The serial interface could have been purchased, however would be more expensive than adding it to the development of the debug board. The layout for the debug board was constructed using an easy to use design software called McCAD. The board has been manufactured with parts order and ready for part placement.



III. Project Boards

A. Flash/ADC Board

The assembled project board was designed by NASA Engineering Technologist, John Dusl, who constructed a board that consists of an AD7980 analog to digital converter (ADC) and MT29F2G16ABBEAHC ONFI NAND Flash Memory module. The Flash/ADC board operates on a single power supply of 3V and contains four input channels that each acquire data acquisition of up to 1MSPS which about 1.2GB of data . The 16-bit AD7980 datasheet recommends a parallel interface when the device is connected to an SPI-compatible digital host (Ref. 1). This requires the bussing of pins for SCLK, SDO, and CNVRT as well as having a voltage reference before the analog signal enters the AD7980 device. The outputs from the AD7980 are bussed into four separate outputs that connect to the FPGA through the P5 connector. Additional signals, clock (CLK) and convert (CNVRT) are added to the busses.

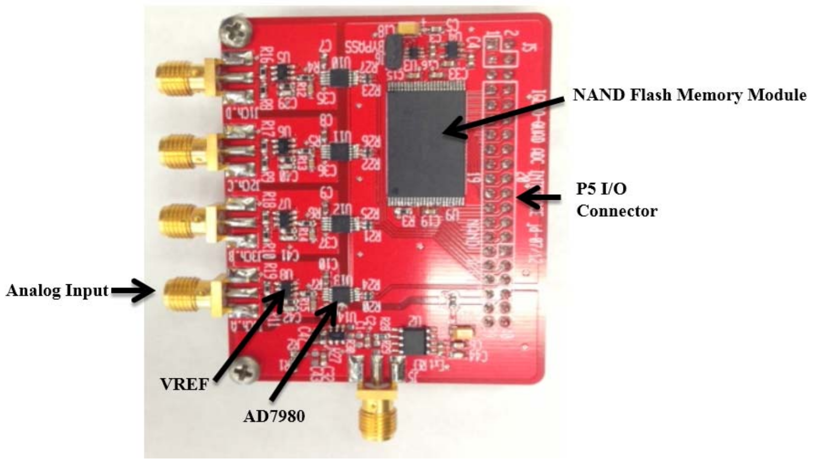


Figure 2. Flash/ADC Board

A 16-bit ONFI NAND Flash Memory module is located on the board alongside the ADC with the control signals and data bits bussed to the P5 connector. The FPGA development kit will use these signals to program the operation waveforms. The NAND flash memory is internally composed of an internal array architecture, Fig. 3, which consists of 2 GB (2048 blocks), 2 planes (1024 blocks per plane), 64 pages per block, and 1056 words per page.

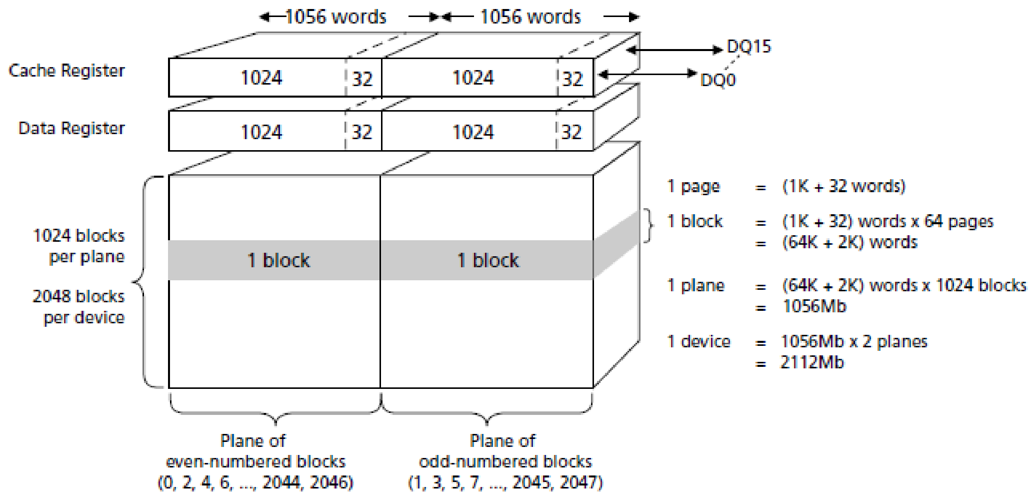
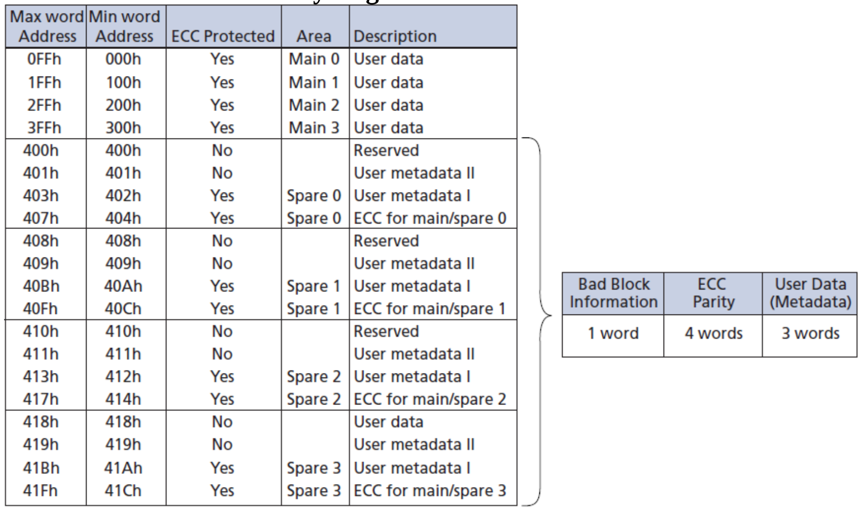

1056 words
1056 words
Cache Register
1024
32
1024
32
DQ15
DQ0
Data Register
1024
32
1024
32
1024 blocks per plane
2048 blocks per device
1 block
1 block
1 page = (1K + 32 words)
1 block = (1K + 32) words x 64 pages = (64K + 2K) words
1 plane = (64K + 2K) words x 1024 blocks = 1056Mb
1 device = 1056Mb x 2 planes = 2112Mb
Plane of even-numbered blocks (0, 2, 4, 6, ..., 2044, 2046)
Plane of odd-numbered blocks (1, 3, 5, 7, ..., 2045, 2047)

Figure 3. Array Organization of NAND Flash

The VHDL ONFI Controller will require that the flash memory successfully WRITE, READ, READ STATUS, ERASE, and RESET the data. Internally the memory module works by clocking in the required data 16-bits at a time, where it is loaded into a 1024 byte registers, and programmed into the NAND Flash memory array. The array addressing is formatted and sent back to the register when a READ operation is initiated. The data at the registers will be able to be clocked out 16-bits at a time by the user. The flash memories are known to potentially have "bad blocks" within the system that can affect the operations from storing correctly. The ONFI module purchased has an internal ECC for spare area mapping of a bad block. "The ECC (error-correcting code) is used to correct bit errors that can occur during normal operation as well as bit errors that occur due to charge loss/gain that develop over time."(Ref. 6) By implementing this error correction method the bad blocks can reliably be separated from the rest of the blocks and stored in the spare area of the module, Table 1. A technique called partial page programming is used to format the ECC for the spare area location. Typically the full page programming method is used for the writing of the entire devices blocks, planes, and pages but for the spare area mapping of a bad block the use of the entire device is unnecessary. The partial page allows for the programming of "smaller portions in a page" (Ref. 7) to implement the ECC.

Table 1. Array Organization of NAND Flash



Max word Address	Min word Address	ECC Protected	Area	Description
0FFh	000h	Yes	Main 0	User data
1FFh	100h	Yes	Main 1	User data
2FFh	200h	Yes	Main 2	User data
3FFh	300h	Yes	Main 3	User data
400h	400h	No		Reserved
401h	401h	No		User metadata II
403h	402h	Yes	Spare 0	User metadata I
407h	404h	Yes	Spare 0	ECC for main/spare 0
408h	408h	No		Reserved
409h	409h	No		User metadata II
40Bh	40Ah	Yes	Spare 1	User metadata I
40Fh	40Ch	Yes	Spare 1	ECC for main/spare 1
410h	410h	No		Reserved
411h	411h	No		User metadata II
413h	412h	Yes	Spare 2	User metadata I
417h	414h	Yes	Spare 2	ECC for main/spare 2
418h	418h	No		User data
419h	419h	No		User metadata II
41Bh	41Ah	Yes	Spare 3	User metadata I
41Fh	41Ch	Yes	Spare 3	ECC for main/spare 3

Bad Block Information	ECC Parity	User Data (Metadata)
1 word	4 words	3 words

Each mode of operation is dependent on seven control lines that control command bytes, address bytes, latching of data, resetting of device, write protect, and chip enable.

Table 2. Control Signals

Signal	Type	Description
ALE	Input	Address latch enable
CE#	Input	Chip enable
CLE	Input	Command latch enable
RE#	Input	Read enable
WE#	Input	Write enable
WP#	Input	Write protect
R/B#	Output	Ready/busy

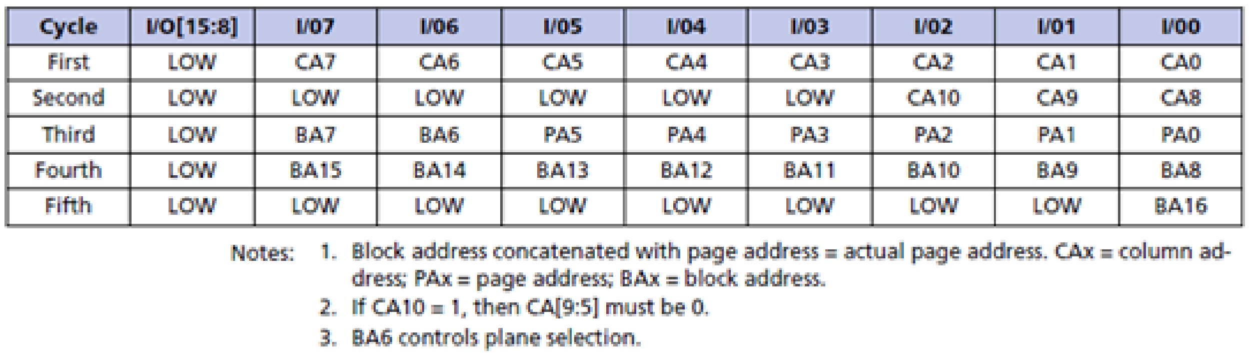
Depending on the control signal that is HIGH, will control the NAND flash memory module during the operation modes. The module is able to identify the operation being called by issuing out a unique command that will begin the operation cycles dependent on that command.

Table 3. Commands and Address Cycles

Command	Command Cycle 1	Number of Address Cycles	Command Cycle 2	Valid During Busy
RESET	FFh	-------	-------	Yes
ERASE	60h	3	D0h	No
WRITE	80h	5	10h	No
READ	00h	5	30h	No
READ STATUS	70h	-------	-------	No

The address cycles that are output after a command is issued contain column and row addresses that need to be set after each operation command is issued. The address bytes contain information about the memory module array such as the starting byte within a page, page address within the block, bad block, and the block address. By controlling these address bytes, the location of where the data will be written can be controlled. Table 4 shows the five cycles within the 16-bit NAND flash memory module.

Table 4: Array Addressing



Cycle	I/O[15:8]	I/07	I/06	I/05	I/04	I/03	I/02	I/01	I/00
First	LOW	CA7	CA6	CA5	CA4	CA3	CA2	CA1	CA0
Second	LOW	LOW	LOW	LOW	LOW	LOW	CA10	CA9	CA8
Third	LOW	BA7	BA6	PA5	PA4	PA3	PA2	PA1	PA0
Fourth	LOW	BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8
Fifth	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW	BA16

Notes:
1. Block address concatenated with page address = actual page address. CAx = column address; PAx = page address; BAx = block address.
2. If CA10 = 1, then CA[9:5] must be 0.
3. BA6 controls plane selection.

During a WRITE operation, data is clocked into the NAND Flash memory from the data registers on the rising edge of the write enable (WE#) control line. In order for the NAND Flash to understand the operation that is taking place, it is necessary to issue the WRITE command, 80h, when the command latch enable (CLE) control line is high. The bus latches the command and issues out five data address bytes when ALE is high followed by tADL delay of 70 ns. The write enable line then begins to toggle to latch the data onto the bus and ends when command 85h is issued.

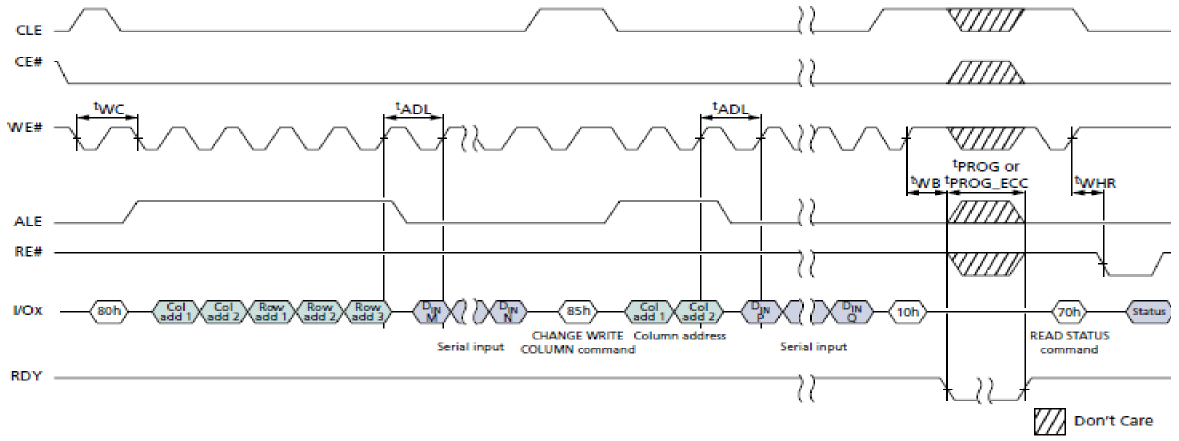


Figure 4. WRITE Operation

During a READ operation, the programmed data is taken from the NAND Flash memory and put back into the registers to allow the user to clock out the desired data. The read command, 70h, is issued to begin the READ operation which is latched on the bus at the raising edge of write enable. Once the command is clocked in five address bytes are output followed by the second READ command, 30h. This will begin the read latch after the tWB delay of 100 ns ends and RE is asserted high.

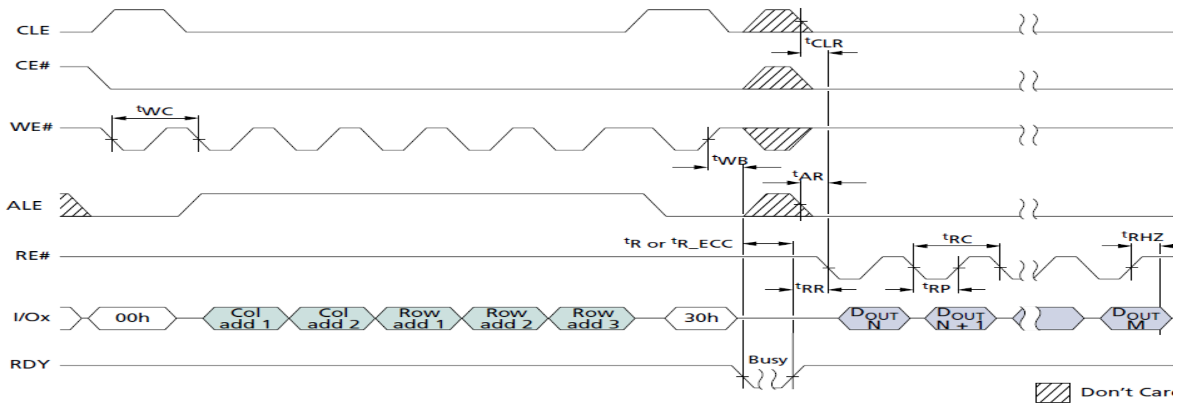


Figure 5. READ Operation

During an ERASE operation initiates when the erase command, 60h, is issued and latched when CLE is high. Once the command is latched the erase command will continue until tWC delay of 25 ns is met and then three address bytes will be latched when ALE is HIGH. The second command, D0h, will end the erase operation.

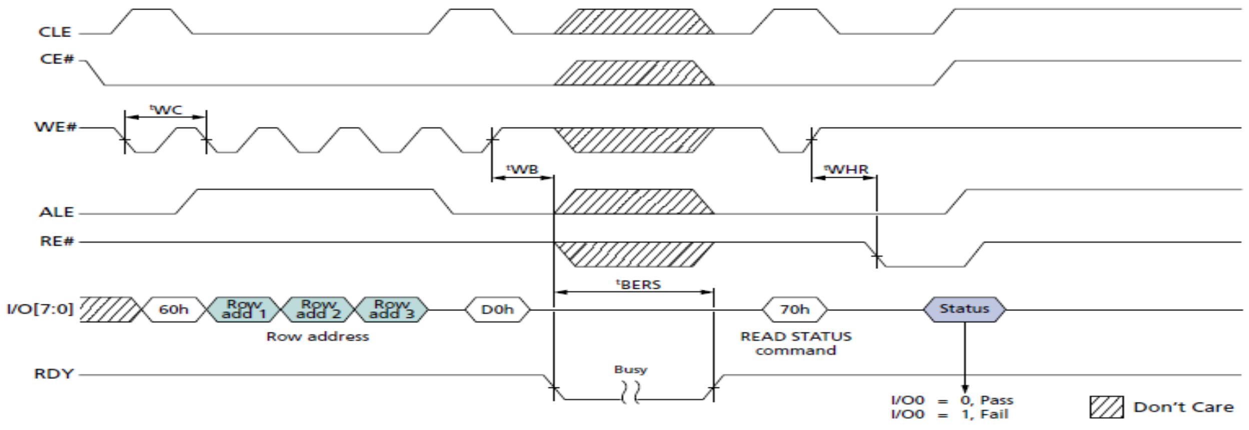


Figure 6. ERASE Operation

The READ STATUS is need to ensure that the data is being stored correctly by issuing a pass or fail status. The operation runs after an ERASE or WRITE operation executes by issuing out command 70h. The READ STATUS, unlike the previous operations, does not issue address bytes but instead will "return the status of the last-selected die on a target" (Ref. 1). The status will be returned once the tWHR delay of 80 ns is met.

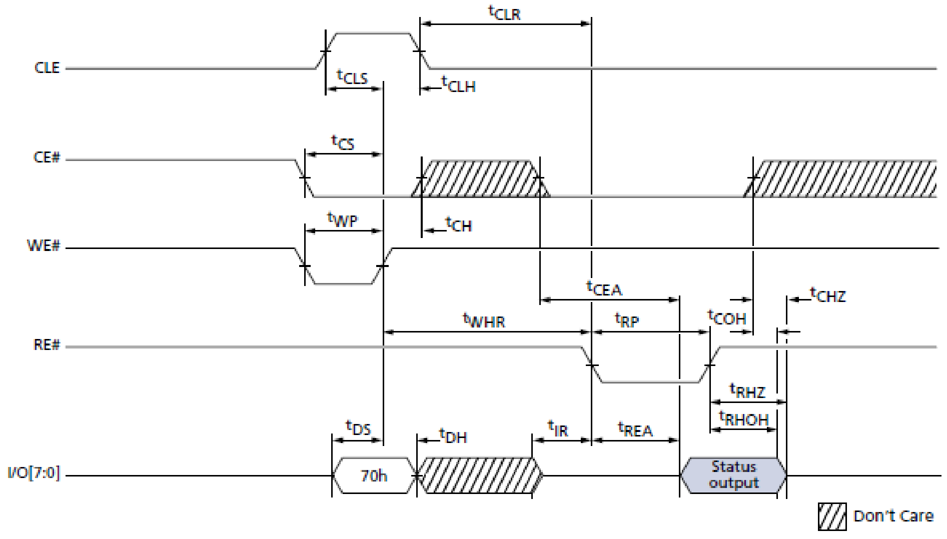


Figure 7. READ STATUS

The reset operation was included to allow a reset option during or after a READ, READ STATUS, WRITE, or ERASE. It is implemented when command, FFh is issued and CLE is HIGH. When a RESET is called, required wait times of tWB and tRST (5/10/50µs) has to be met before the next operation can begin.

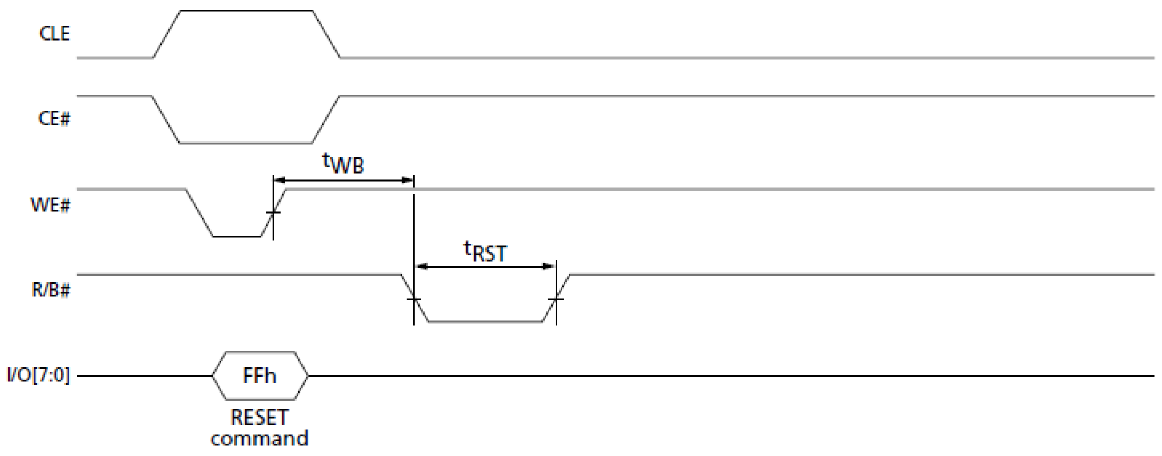


Figure 8. RESET Operation

B. FPGA Development Kit

The Flash/ADC board is used to convert analog signals into digital to allow the user to manipulate the signal data using the FPGA Development Kit. The IGLOO series FPGA Development Kit is an "ultra-low power programmable" advanced microprocessor-based FPGA that is radiation tolerant. (Ref. 2) The Actel Microsemi board requires the use of Libero IDE software, Fig. 10, for the in system programming of the FPGA. VHSIC hardware description language (VHDL) is used to program the development kit due to its ability to be "described (modeled) and verified (simulated)" (Ref. 10) before it is implemented into the design hardware. The development kit, Fig. 9, is composed of several components but the primary parts used for the project are the reprogrammable flash technology and advanced I/O connectors. The development kit has ten switches and LEDs that are used to give the user an external testing mode for the code. Switches are assigned to initiate the operations and accessing of the internal memory, with the LEDs set to output the lower byte read back information. The 9th LED is set to output a warning sign in the event that the board is in busy mode.

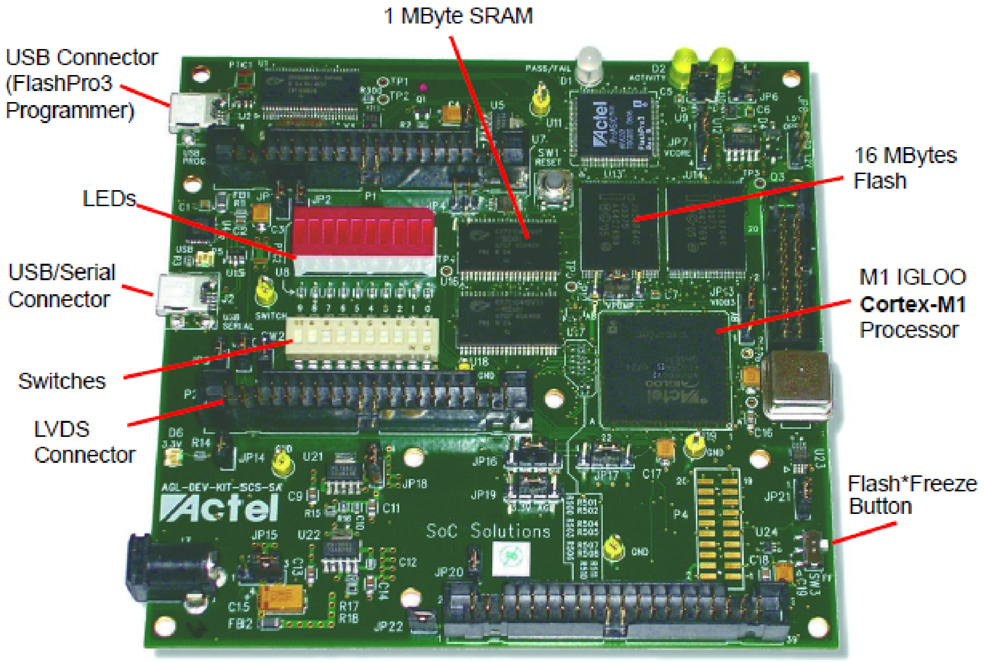
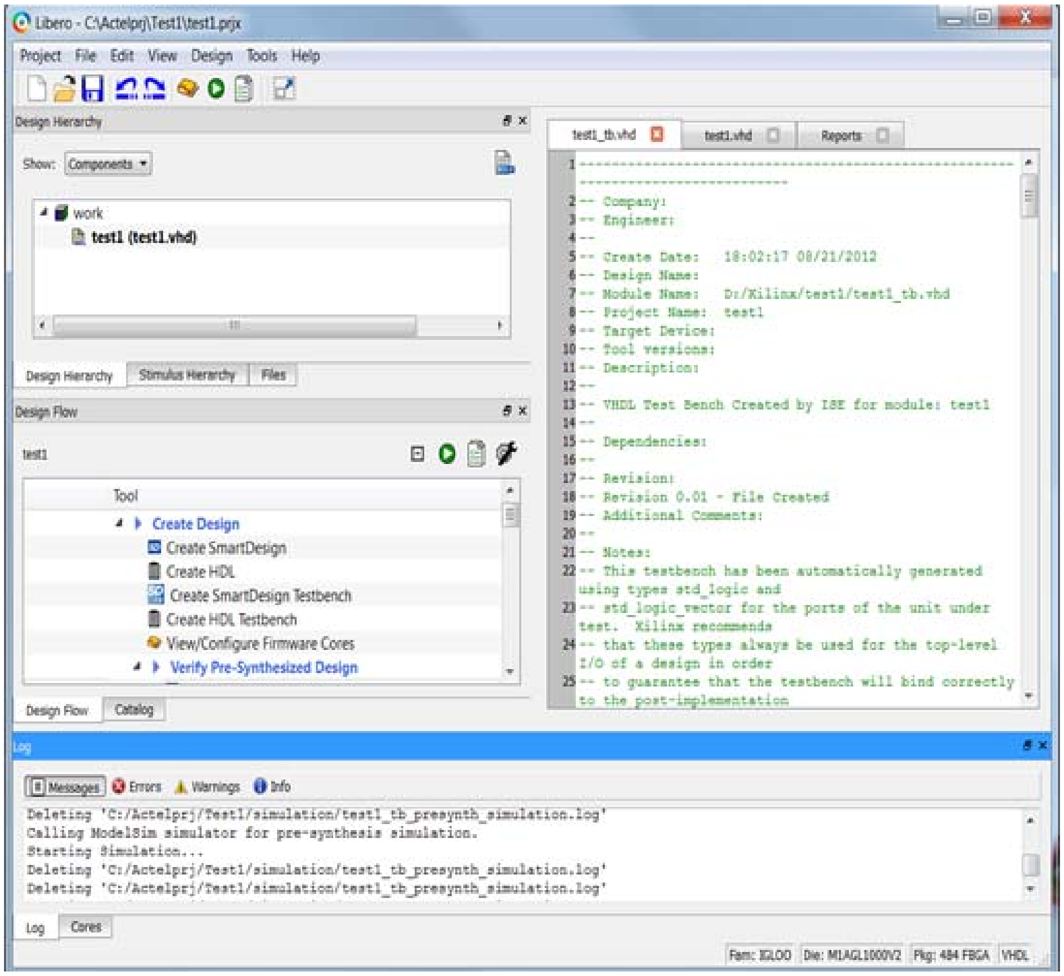


Figure 9. FPGA M1AGL Development Kit Board

The development kit is powered by the J3 connector and has an extension cord allowing it to be connected to a power outlet. A yellow LED light will turn ON indicating that the device is powered and ready to use. The USB PROG connector is then connected to the computer desktop, the PC will detect the new hardware and prompt installation of the driver for use of the FlashPro3 Programmer. Once Libero is synced with the FPGA, the device proprieties need to be set in order for Libero to know the type of Actel board being used. Libero will allow for the VHDL test bench and module to be synthesized, compiled, place-and-routed, and programmed for use by the FPGA. The test bench in Libero generates stimulus files for simulation which allows for the testing of the hardware module prior to the implementation on the FPGA Development Kit. The output responses will be compared with the expected values of the ONFI waveforms. The module holds the generated waveforms and model timing for the program, and when implemented into the hardware will be what controls operations and accessing of memory. Before a successful test on the FPGA hardware can be accomplished, the ports have to be assigned to the proper FG848 package. This is done by compiling the VHDL program and opening the I/O constraints section in Libero to assign the ports. Multiple modules and test benches can be placed in a single project for simulation and testing. In order for Libero to recognize the combination of modules and test benches that are needed for execution, it is necessary to change the “set root” in the design hierarchy and “set simulation” in the stimulus hierarchy.



```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   18:02:17 08/21/2012
-- Design Name:
-- Module Name:   D:/Xilinx/test1/test1_tb.vhd
-- Project Name:  test1
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: test1
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
```

```
Deleting 'C:/Actelprj/Test1/simulation/test1_tb_presynth_simulation.log'
Calling ModelSim simulator for pre-synthesis simulation.
Starting Simulation...
Deleting 'C:/Actelprj/Test1/simulation/test1_tb_presynth_simulation.log'
Deleting 'C:/Actelprj/Test1/simulation/test1_tb_presynth_simulation.log'
```

Libero - C:\Actelprj\Test1\test1.prjx
Project File Edit View Design Tools Help
Design Hierarchy
Show: Components
work
test1 (test1.vhd)
Design Hierarchy Stimulus Hierarchy Files
Design Flow
test1
Tool
Create Design
Create SmartDesign
Create HDL
Create SmartDesign Testbench
Create HDL Testbench
View/Configure Firmware Cores
Verify Pre-Synthesized Design
Design Flow Catalog
test1_tb.vhd test1.vhd Reports
Log
Messages Errors Warnings Info
Log Cores
Fam: IGLOO Die: M1AGL1000V2 Pkg: 484 FBGA VHDL

Figure 10. Libero

C. Debug Board

The debug board is used for verification of the analog to digital signal that is being generated on the FLASH/ADC board and give a serial interface to the Modular Instrumentation System. The board was constructed with the digital to analog converter (DAC) and serial port being the main components of the board. The chosen DAC used for the board is a TLV5619 which is a 2.7V to 5.5V, 12-bit parallel converter with optional power down mode that allows for the device to be low power. The output voltage of the DAC are “buffered by an x2 gain rail to rail amplifier to enable the output to update asynchronously using the $\overline{LDAC}$ pin.” (Ref. 4) The datasheet for the converter recommends that the pins $\overline{LDAC}$ and $\overline{CS}$ are grounded when a parallel interface is used for the system. Series termination resistors are added to the output of the data bits to reduce unwanted EMI and are directly traced to the P1 connector that will attach to the FPGA board. A DB-9 serial port is added to the debug board for future incorporation of a serial interface and is connected to a TRS3232 voltage converter through the receive and transmit/drive data ports. The output for the ports are connected to P1 connector in order to allow programming with the FPGA of these signals.

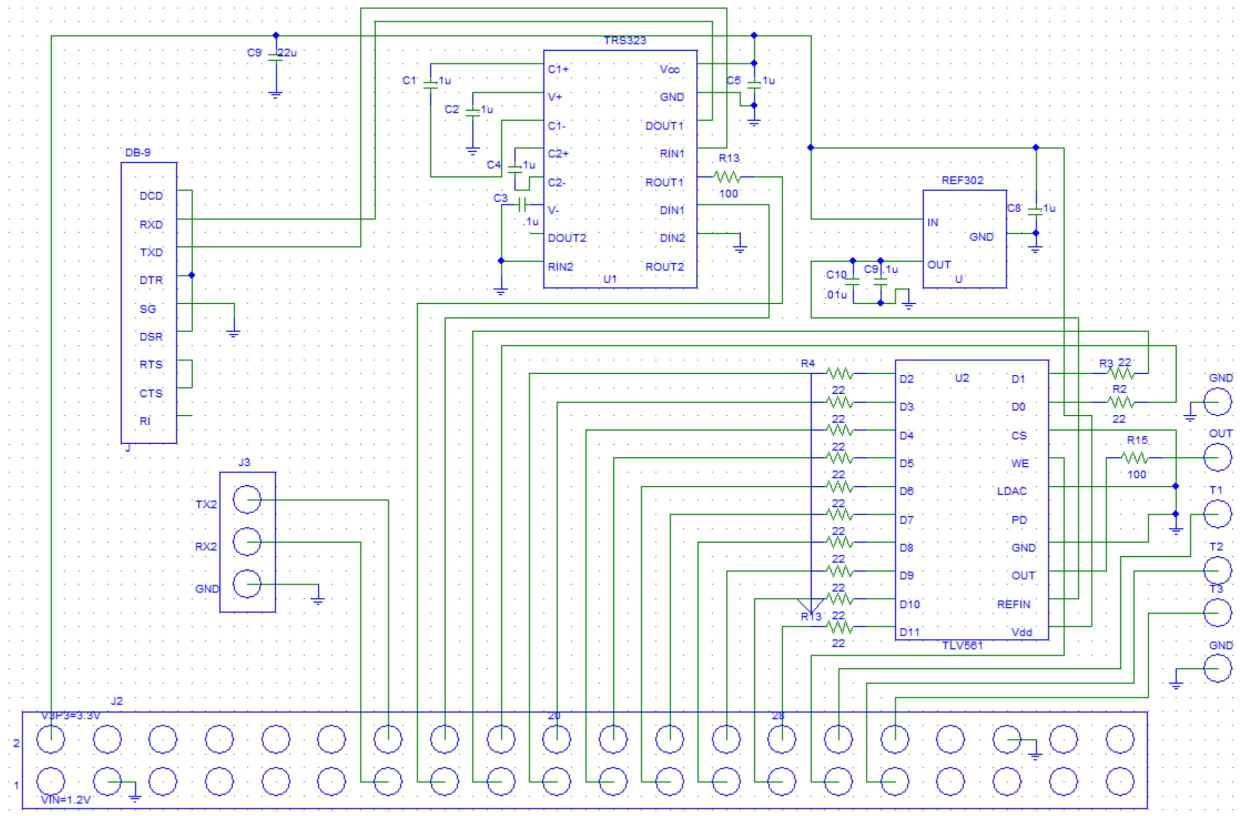


Figure 11. Debug Board Schematic

The debug board layout was designed using an electronic design system called McCAD. Referencing the schematic for the board, Fig. 11, a layout was created taking careful consideration of part placement to reduce the board size, potential noise, and trace complexity.

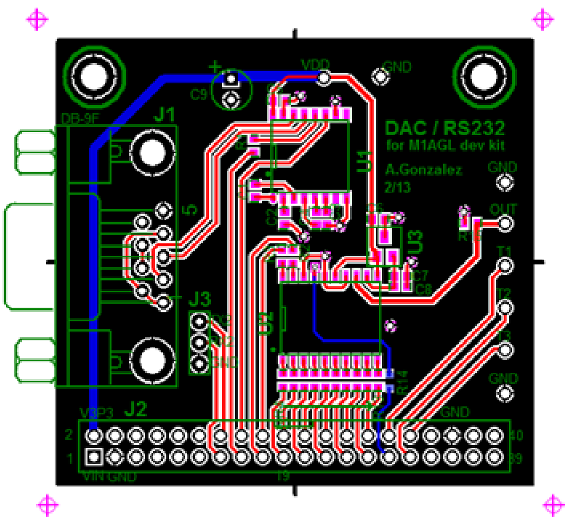


Figure 12. Debug Board Layout

Footprints for parts were obtained by using premade library layouts and by manually designing components using datasheets for dimension specifications. The debug board is two layered with the top layer acting as the power plane and the bottom layer as the ground plane. Trace sizes varied depending on the amount of current supplied through the trace to the component. For example, power traces were made to be two times larger than the normal traces due to the amount of voltage, 3.3V, at the power supply. Vias are placed on the common ground and $\overline{WE}$ pin to allow connections from one plane to another. Mounting holes at the top left and right corners of the board are added to give the board stability when being connected to another board. Drill sizes for holes, Table 5, were chosen by replicating a premade project.

Table 5. Top and Bottom Plane Colors

	Top Plane	**Bottom Plane**
Pad	Pink	Pad Light Blue
Traces	Red	Blue
Silk	Green	
Through Holes	White	White
Common Pads	Black	Black

Table 6: Drill Types and Sizes

Drill Type	**Drill Sizes**
x	.040’’
Hexagon	.035’’
Square	.020’’
Cross	.015’’
Diamond	.118’’
Triangle	.028’’
Circle	.125’’

Once the layout for the debug board was completed, the Gerber files were extracted from McCAD and used in the GerbTool software. The GerbTool software is a “PCB CAM tooling and analysis software” (Ref. 5) for Gerber verification, design verification, and manufacturing optimization. The layers of the board are viewed and verified for manufacture.

IV. Evaluated Codes

The initial steps of attempting generate the ONFI waveforms was to view projects that are Flash based and tailor those codes to the scope of the VHDL ONFI Controller project. The projects looked at are a Free Model Foundry code obtained online (Ref. 8) and a JSC flash radiation test project. The JSC flash memory project was considered for its successful completion of a NAND Flash memory interface of an 8-bit memory module. There are a few differences in the desired operations of the JSC project and the NAND Flash Memory Interface which are noted in the table below:

Table 7. Differences of Project and other JSC Project

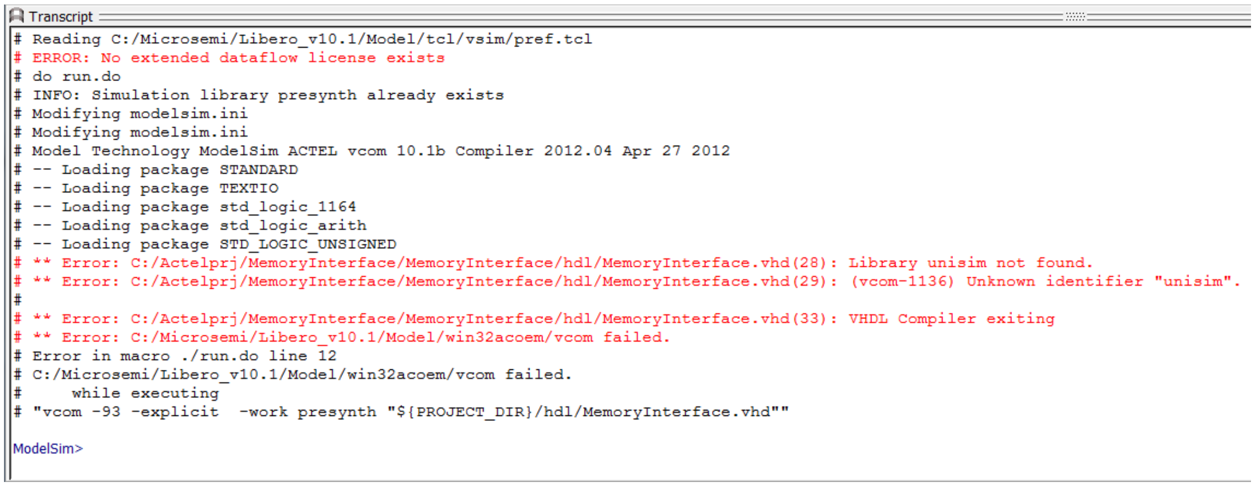
Free Model Foundry (FMF)	**JSC Flash Radiation Test Project**
Only VHDL	Mixed VHDL and Verilog
16-bit device	8-bit device
“Handshake” between user and module necessary	No user protocol

Simulation of the JSC Project were attempted, however, the noted differences proved to be detrimental to the successful simulation of the project.

A. JSC Flash Radiation Test Project

The project code for the Flash Radiation Test Project was supplied by the NASA engineer and was originally run on a different programming software called Xilinx. The code was created for an 8-bit flash memory with no user protocol which would have to be altered to the scope of the 16-bit flash module used in the project. The test bench of the radiation project is written in Verilog and the hardware modules (hdl) are coded in VHDL which is allowed with

Xilinx, however, posed an issue while simulating using Libero. When attempting to simulate the project, unisim library errors appeared due to the compatibility issues between Xilinx and Libero.



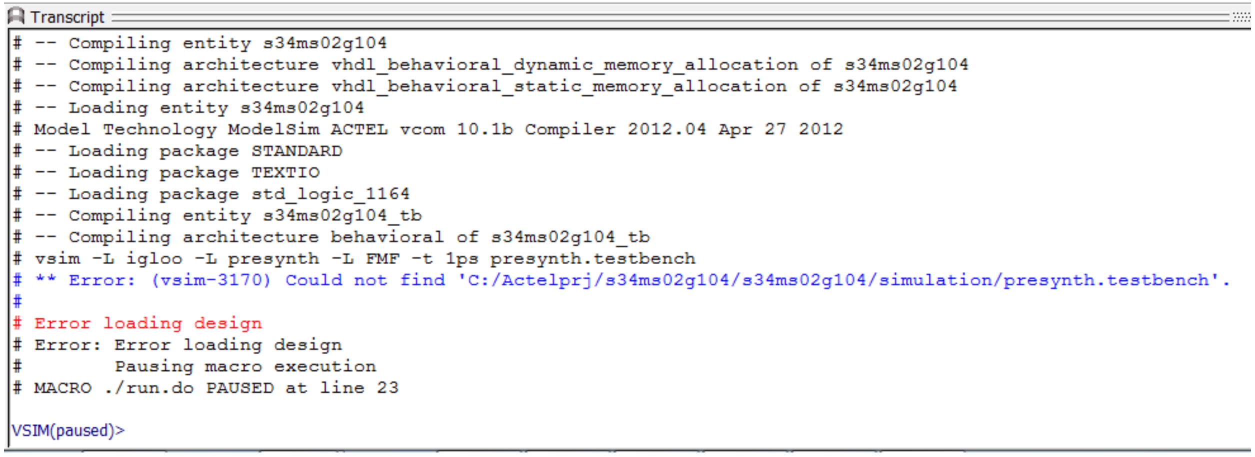
```
Transcript
# Reading C:/Microsemi/Libero_v10.1/Model/tcl/vsim/pref.tcl
# ERROR: No extended dataflow license exists
# do run.do
# INFO: Simulation library presynth already exists
# Modifying modelsim.ini
# Modifying modelsim.ini
# Model Technology ModelSim ACTEL vcom 10.1b Compiler 2012.04 Apr 27 2012
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Loading package std_logic_arith
# -- Loading package STD_LOGIC_UNSIGNED
# ** Error: C:/Actelprj/MemoryInterface/MemoryInterface/hdl/MemoryInterface.vhd(28): Library unisim not found.
# ** Error: C:/Actelprj/MemoryInterface/MemoryInterface/hdl/MemoryInterface.vhd(29): (vcom-1136) Unknown identifier "unisim".
#
# ** Error: C:/Actelprj/MemoryInterface/MemoryInterface/hdl/MemoryInterface.vhd(33): VHDL Compiler exiting
# ** Error: C:/Microsemi/Libero_v10.1/Model/win32acoem/vcom failed.
# Error in macro ./run.do line 12
# C:/Microsemi/Libero_v10.1/Model/win32acoem/vcom failed.
#     while executing
# "vcom -93 -explicit  -work presynth "${PROJECT_DIR}/hdl/MemoryInterface.vhd""

ModelSim>
```

Figure 13. Unisim Library Error

To remedy the error, the unisim library was commented out and the simulation was attempted again but errors were still occurring that did not give any direction in the location of the issue. At this point, it became clear that the attempt to get the code running would be very time consuming to correct and generating a new test bench in VHDL, instead of Verilog, would be a temporary fix. The uncertainty of the code running correctly with the entire code of the project is unknown and could potentially result in no progress towards the ONFI memory controller.

B. Free Model Foundry (FMF)

The next flash memory attempted is a Free Model Foundry (FMF) code that was found online. The open source VHDL model is free software that is meant for redistribution and modification to fit the needs of a project. The FMF code was tailored for a 2gb, 16-bit NAND interface which, if simulated, could potentially be useful to the project. The most notable issue that was seen prior to simulation is the lack of a test bench. Libero offers the option to automatically generate a test bench based on the hdl module that is in the project. After successful creation of a test bench, the project was run with the following errors generated:



```
Transcript
# -- Compiling entity s34ms02g104
# -- Compiling architecture vhdl_behavioral_dynamic_memory_allocation of s34ms02g104
# -- Compiling architecture vhdl_behavioral_static_memory_allocation of s34ms02g104
# -- Loading entity s34ms02g104
# Model Technology ModelSim ACTEL vcom 10.1b Compiler 2012.04 Apr 27 2012
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Compiling entity s34ms02g104_tb
# -- Compiling architecture behavioral of s34ms02g104_tb
# vsim -L igloo -L presynth -L FMF -t 1ps presynth.testbench
# ** Error: (vsim-3170) Could not find 'C:/Actelprj/s34ms02g104/s34ms02g104/simulation/presynth.testbench'.
#
# Error loading design
# Error: Error loading design
#        Pausing macro execution
# MACRO ./run.do PAUSED at line 23

VSIM(paused)>
```

Figure 14. FMF Errors

The error could have been generated due to the in proper linking of the FMF libraries into Libero but the error presented held no useful instruction in what caused it or how it could be corrected. A case was created in the Microsemi Actel website for professional guidance in fixing the simulation error. The error messages along with the project files were attached to the report where Actel assigned a technologist to look into the case. The technologist gave general answers into fixing the problem stating that "VHDL source code generated from FMF is not synthesizable. That can be used only for simulation." (Ref. 11) This input proved to not be useful due to the fact that simulation is the only action taken at getting the code working. Ultimately, trying to simulate and get this project running was becoming time consuming and resulting in little progress towards the NAND flash memory interface.

Due to the amount of time that would be needed in attempting to get the codes simulated and functioning according to the project requirements, proved to be too long in comparison to the time scope given for this project. The conclusion reached, was to generate test routines, along with my mentor Robert Shuler, that are based on the ONFI memory module. Following this method will give an advantage over using the other codes because a conceptual visual of the foundation will be known, allowing for easier debugging of the system.

V. Development Process

The development process for the construction of the VHDL ONFI Controller consisted of first looking at the internal packaging of the flash module in order to properly link the module with the FPGA I/O P5 connector that is used to program the signals. The clock cycles for the module were set to 25 ns in order to meet the module I/O performance speed and half speed clocks, Fig. 15, were created to compensate for a trailing glitch occurring with the WE latching of the command.

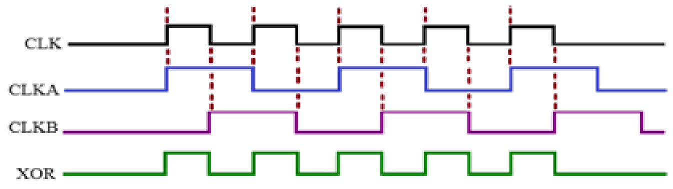
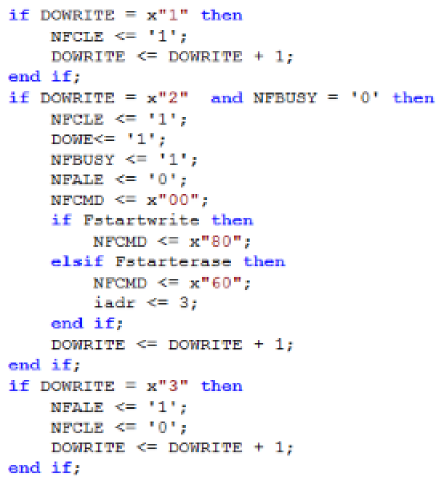


Figure 15: Half Speed Clocks

A sequential style structure was created within the code for each operation to be able to set the control lines within each the incrementing variable that represent clock cycles. The state variables were given names DOWRITE, DOREAD, DORESET, DOREADSTATUS, and DOERASE.



```
if DOWRITE = x"1" then
    NFCLE <= '1';
    DOWRITE <= DOWRITE + 1;
end if;
if DOWRITE = x"2"  and NFBUSY = '0' then
    NFCLE <= '1';
    DOWE<= '1';
    NFBUSY <= '1';
    NFALE <= '0';
    NFCMD <= x"00";
    if Fstartwrite then
        NFCMD <= x"80";
    elsif Fstarterase then
        NFCMD <= x"60";
        iadr <= 3;
    end if;
    DOWRITE <= DOWRITE + 1;
end if;
if DOWRITE = x"3" then
    NFALE <= '1';
    NFCLE <= '0';
    DOWRITE <= DOWRITE + 1;
end if;
```

Figure 16: Write Operation Sequential Code

A debounce for the switches being used for operations is added to clean up the signal output which is unstable at the initiation of the given operation. To account for delays in the ONFI waveforms, an integer internal signal was created allowing the signal to be set within a state variable with the size of the delay dependent on the integer value assigned.

The NAND flash interface design architecture is intended to control the ADC flash module by issuing operations for WRITE, READ, READ STATUS, ERASE, and RESET. The hardware module (refer to appendix for code) is constructed to have an external manual set feature by using the FPGA switches to control the memory and operations for the ONFI flash module. The setting of these external functions was done to properly diagnose the memory and operations are working as expected.

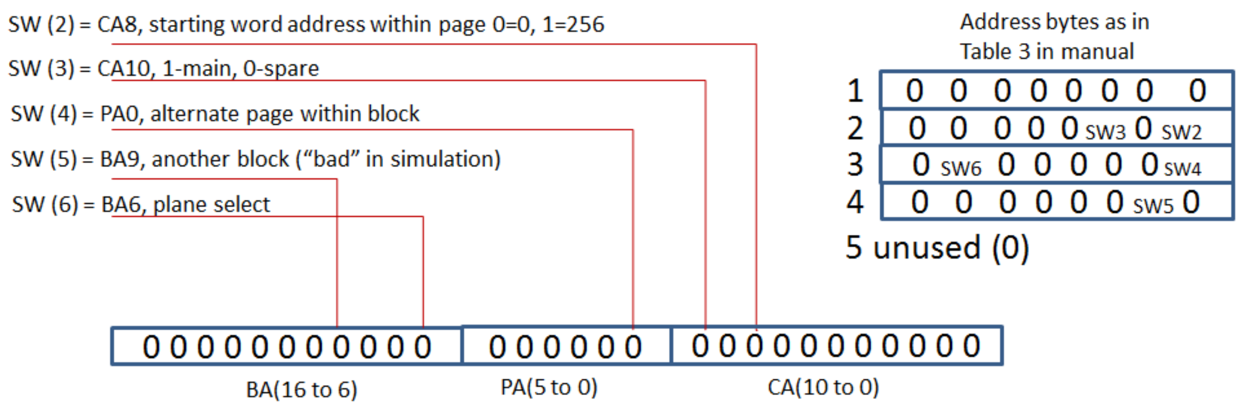

SW (2) = CA8, starting word address within page 0=0, 1=256
SW (3) = CA10, 1-main, 0-spare
SW (4) = PA0, alternate page within block
SW (5) = BA9, another block ("bad" in simulation)
SW (6) = BA6, plane select
Address bytes as in Table 3 in manual
1 0 0 0 0 0 0 0 0
2 0 0 0 0 0 SW3 0 SW2
3 0 SW6 0 0 0 0 0 SW4
4 0 0 0 0 0 0 SW5 0
5 unused (0)
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
BA(16 to 6)
PA(5 to 0)
CA(10 to 0)

Figure 17. Address Functions

The operation that is executed depends on the control functions of switches 7 to 8, where switch 9 needs to be toggled in order to initiate the set command and switch 0 to reset the busy signal. The listed combinations for switches 7 and 8 are used to initiate the operations: "00" = READ, "10" = WRITE, "01" = ERASE, and "11" = READ STATUS.

When WRITE data occurs, a word of data will be written, Fig. 18, with the upper byte set to 0's and the lower byte controlled by switches 0 to 1 and a 6-bit counter. Switches 0 to 1 are used to demonstrate an arbitrary data tag for read back verification with the counter used to verify which word is being read within a page.

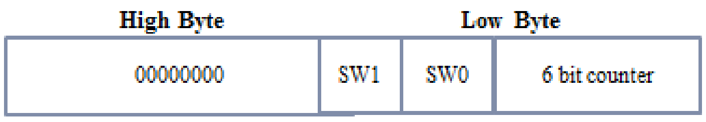


Figure 18. Word Data

LEDs 0 to 7 are set to visually show that the read back verification is correct with LED 9 displaying a BUSY warning. In order for any operation to be executed, the BUSY LED must be reset to allow it to be in the ready state.

The test bench (refer to appendix for code) was created to visually see the ONFI operations being simulated prior to programming of the FPGA Development Kit. The test bench allowed for the operation switches to be set, the number of address bytes to be shown, and the delay of the busy control signal to be set. To give a visual of which operations is being executed, FUNC was added internally to display the read, read status, erase, and write.

After successful completion of the operation codes, the simulation for each waveform operation, refer to section 2A, was obtained with the following results:

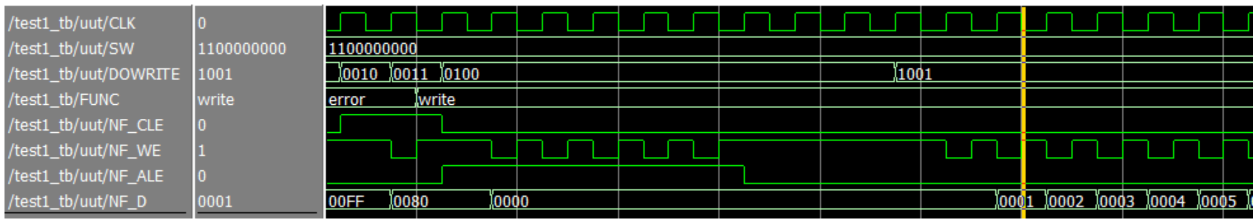


Figure 19. Write Operation Latching of Data

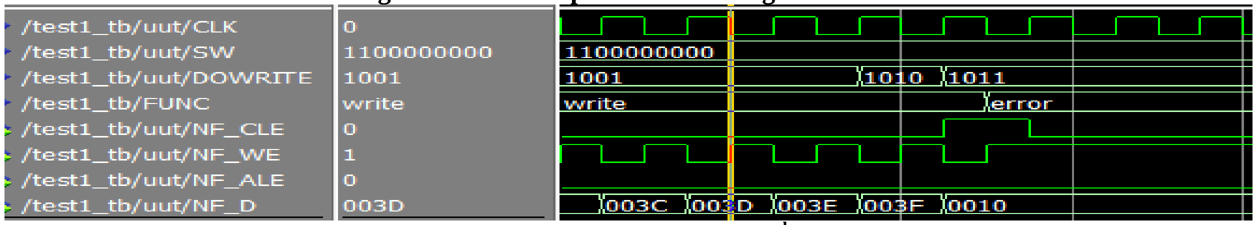


Figure 20. Write operation Latching of 2nd Command

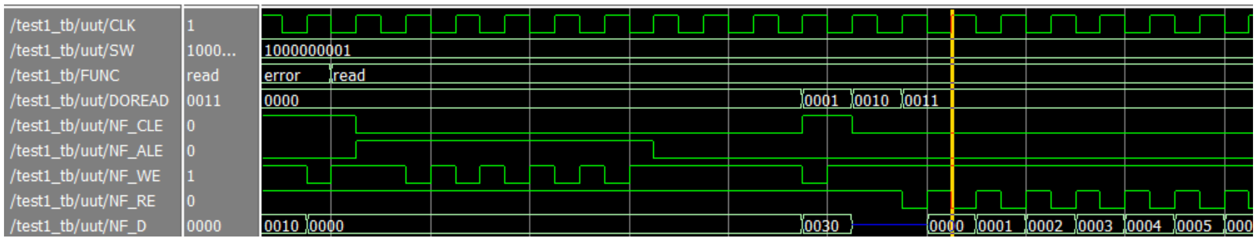


Figure 21. Read Operation

The read operation, Fig. 21, shows the command latching of command 00h, followed by the 5 address bytes being written onto the bus, a second command 30h ends the command sequence, and the latching of data occurs on the with the RE control signal toggling.

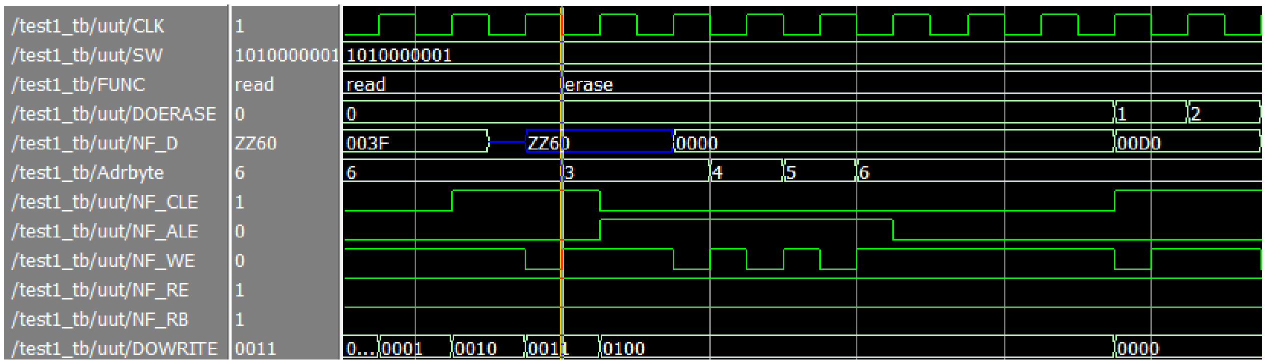


Figure 22. Erase operation

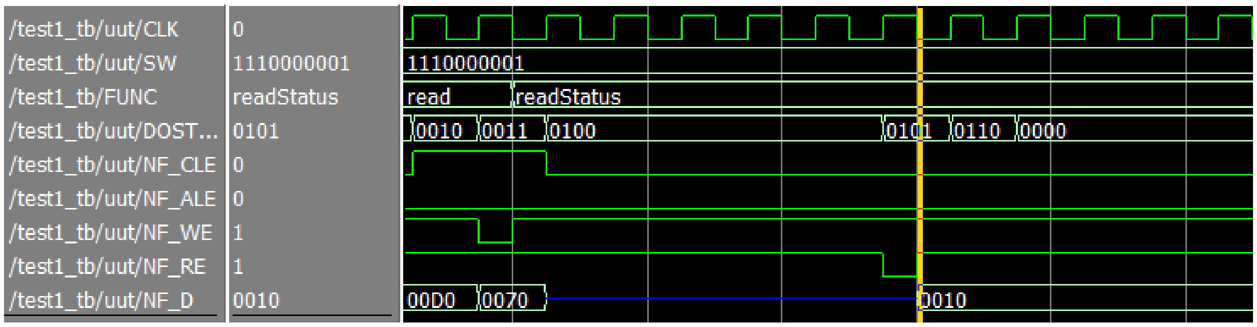


Figure 23.Read Status

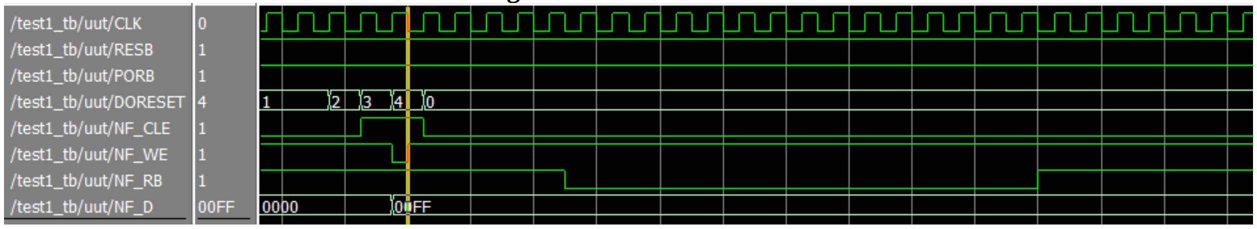


Figure 24. Reset operation

The simulations show the latching of the necessary commands at the rising edge of WE, followed by the address bytes written onto the bus (if applicable), the switches control the start of the operation, the start variable increments for the setting of the control signals within a given clock cycle, and the latching of data on the bus with RE or WE set to control the access of the data on the bus. Second commands for the read, write, and erase end the accessing of the data.

VI. Results

A. FPGA Waveform Operation Testing

After confirming that the each operation is working as expected through the simulation, the code was then programmed into the FPGA board to test on the oscilloscope to verify that the hardware is operating correctly. The following images are showing the operations on an oscilloscope with channels 1 to 4 connected to WE, CLE, ALE, and the state variable (dependent on the given operation, ex: DORESET is used for the RESET operation).

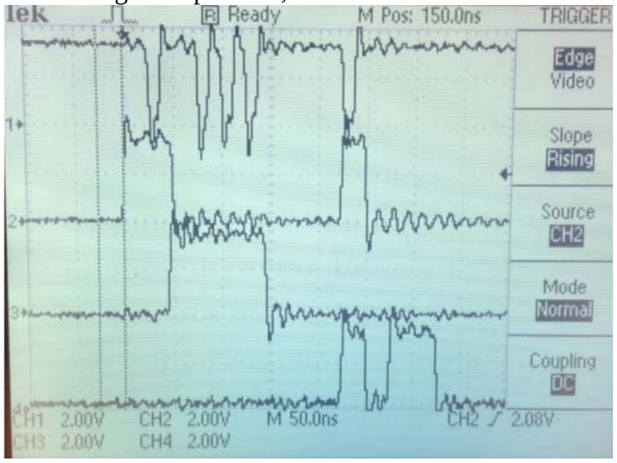


Figure 25. Erase

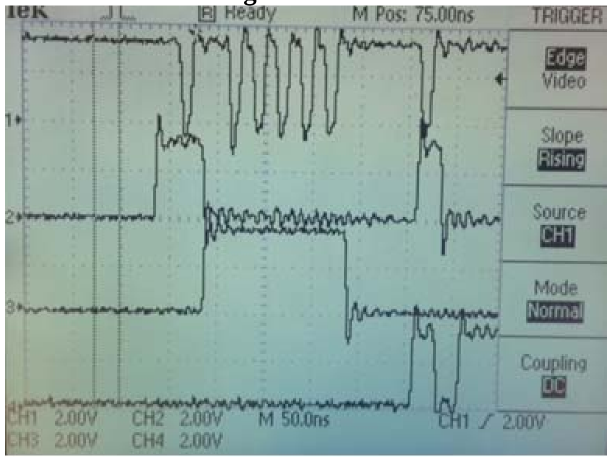


Figure 26. Read

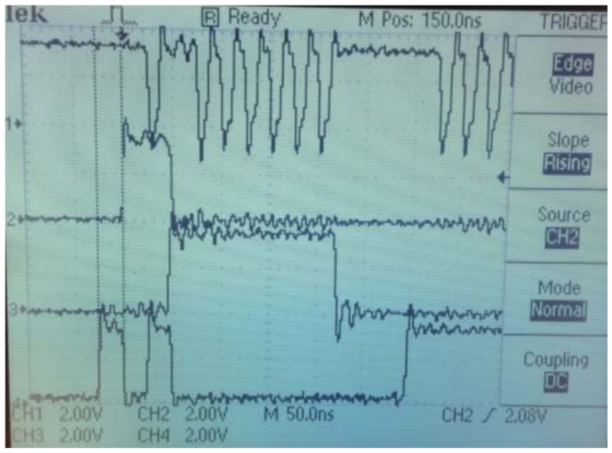


Figure 27. Write

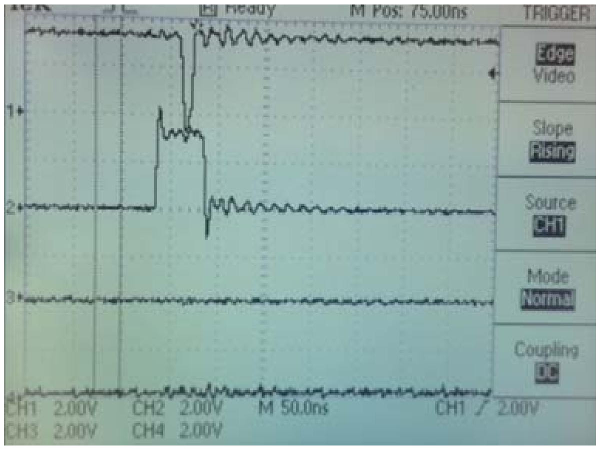


Figure 28. Read Status

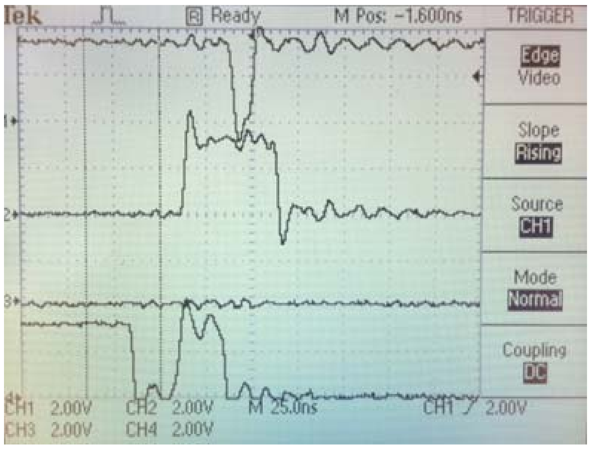


Figure 29. Reset

The READ, REAT STATUS, ERASE, and WRITE operations proved to be function correctly on the FPGA board by matching the simulation and datasheet waveforms.

B. FPGA and Flash/ADC Board Testing

After successfully testing the operations on the FPGA Development Kit, the board was then connected with the Flash/ADC board for further testing with the actual ONFI memory module. Once the P5 I/O connectors were linked together, the WRITE operation was then initiated with the following oscilloscope image obtained:

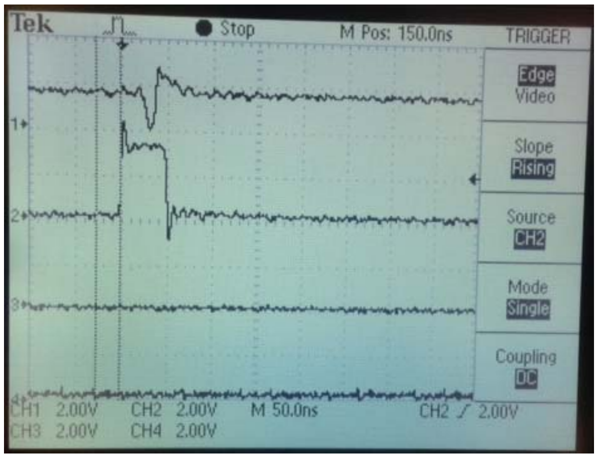


Figure 30. Testing of WE and CLE

The connection of the two boards had no output on the oscilloscope when the operation switches were externally set to activate. Through further testing it was found that the WE control line was reaching half of the Vdd which is not meeting device specifications. Further inspection into the issue lead to looking at the connection of the ONFI memory module on FLASH/ADC board. The problem was found to be that the module was incorrectly design by having the internal pins offset by one, Fig. 31.

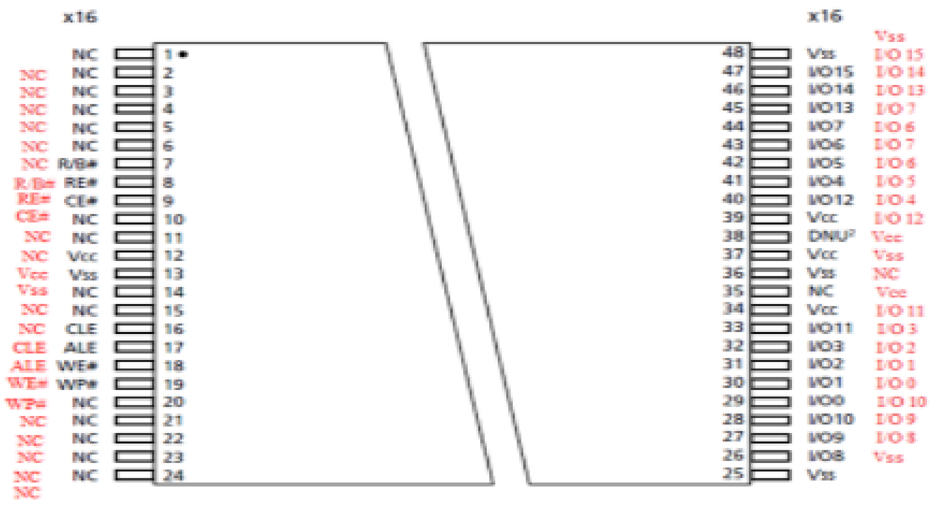


Figure 31. Internal Signals for ONFI Memory Module

This posed to be a detrimental issue to the continued testing of the ONFI test routines due to the board needing to be remanufactured.

VII. Future Work

The future work of the development of the Module Instrumentation System will be to fix the Flash/ADC board misalignment issues that occurred prior to manufacture, place the components on the debug board, create routines for the ADC and serial interface, connect the modular instrumentation processor, and create package test code for application control.

VIII. Conclusion

The development process for the VHDL ONFI Controller consisted of many components, from the creation of the test routines to the construction of the debug board. The most challenging aspects of the project were learning the complex ONFI internal architecture and the Actel compatible programming software that is specific to the FPGA chosen. The failure of the Flash/ADC board prevented the results of the FPGA test routines to be known, however once the board is completed the test routines are expected to function according the standards of the ONFI module.

Appendix

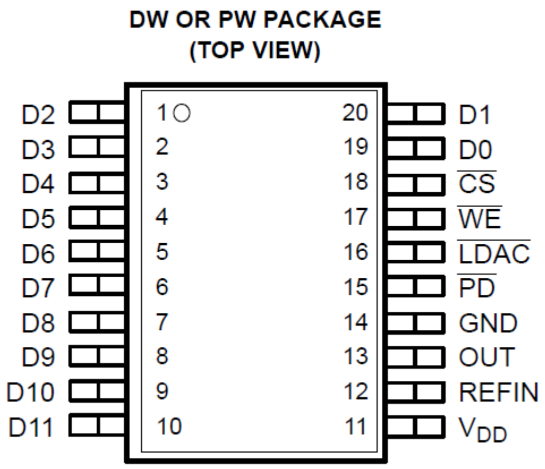


Figure 32. TLV5619 Pin Diagram

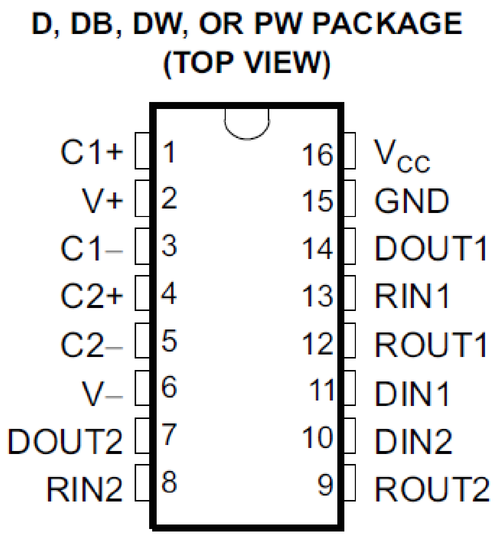


Figure 33. TRS3232 Pin Diagram

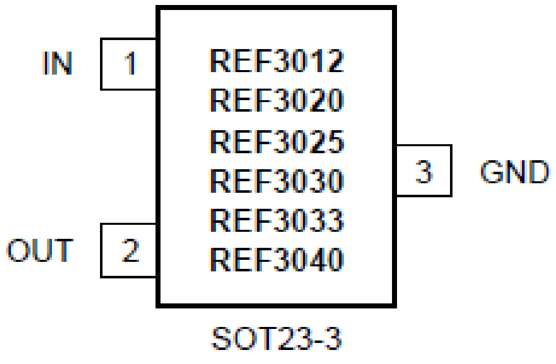


Figure 34. Ref3020 Pin Diagram

Table 8. AD7980 Pin Description

Mnemonic	Description
REF	Reference Input Voltage
VDD	Power Supply
IN+	Analog Input
IN-	Analog Input Ground Sense
GND	Power Supply Ground
CNV	Convert Input
SDO	Serial Data Output
SCK	Serial Data Clock Input
SDI	Serial Data Input
VIO	Input/Output Interface Digital Power

Table 9. Debug Board Component Values

Item #	Reference Designator	Qty.	Value
1	C1 C2 C3 C4 C5 C6 C8	7	.1µF
2	C7	1	0.01µF
3	C9	1	22µF
4	R1 R15	1	100Ω
5	R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14	13	22Ω
6	U1	1	TRS3232
7	U2	1	TLV5619
8	U3	1	REF3012
9	J1	1	DB-9F
10	J2	1	40-PIN CONN
11	J3	1	3-PIN CONN

VHDL Hardware Model for ONFI Controller

```
----------------------------------------------------------------------
----------
-- Company: NASA Johnson Space Center / Avionic Systems Division
-- File: test1.vhd
-- Description: Main test module (top level) for FPGA/DAC/FLASH
project
-- Targeted device: <Family::IGLOO> <Die::M1AGL1000V2> <Package::484
FBGA>
-- Author: Robert Shuler / EV5 / x35258 / robert.l.shuler@nasa.gov
--         and April Gonzalez / intern
----------------------------------------------------------------------
----------

--
**********************************************************************
******************************
```

```
-- ********** NOTE: BE SURE TO CHANGE DBCTR COMPARE BETWEEN BOARD AND
SIMULATION VERSIONS *************
--
**********************************************************************
******************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity test1 is port (
  -- dev kit signals
    CLK   : IN   std_logic;            -- E4
    RESB  : IN   std_logic;            -- T9
  PORB  : IN std_logic;            -- V7
  SW    : IN   std_logic_vector(9 downto 0);  --
D18,D17,E17,F16,D15,G14,E14,F14,G13,D14 (0 .. 9 order)
    LED   : OUT  std_logic_vector(9 downto 0);  --
E10,F10,U8,W5,U7,V6,U5,U4,V4,V5 (0 .. 9 order)
  -- ADC/NAND FLASH board
  SDO   : IN std_logic_vector(4 downto 1);  -- R16,T16,T1,U1 (1 .. 4
order)
  CNVRT : OUT  std_logic;              -- M3
  SCLK  : OUT  std_logic;            -- W2
  NF_WP : OUT  std_logic;            -- AB13  write protect active low
(WPB)
  NF_WE : OUT  std_logic;            -- AB6 write enable (WEB) host ->
flash when 0
  NF_ALE: OUT  std_logic;            -- AB12  address latch enable
  NF_CLE: OUT  std_logic;            -- AA7 command latch enable
  NF_CE : OUT  std_logic;            -- AA16 chip enable active low
(CEB)
  NF_RE : OUT  std_logic;            -- AA6 read enable active low
(REB) - flash -> host when 0 (opposite of WE)
  NF_RB : IN  std_logic;             -- Y2  ready/busy active low 0 =>
ready
  NF_D  : INOUT std_logic_vector(15 downto 0);   --
A17,A11,B13,B6,B17,B10,AB9,AB17,A7,A16,A10,B16,B7,AB8,AB16,AB7 (0 ..
15 order)
  --Connector P1(could not use connector P2 due to it being lvds diff
pair)
  OUTRESET : OUT std_logic;          -- G11 pin 8
  OUTWRITE : OUT std_logic;          -- H11 pin 12
  OUTREAD : OUT std_logic;           -- G12 pin 16
  OUTERASE : OUT std_logic;          -- D16 pin 22
  OUTSTATE : OUT std_logic_vector(3 downto 0) -- E15 pin 24, N5 pin
26, K4 pin 28, K6 pin 30
); end test1;

architecture behavior of test1 is
```

```
**********************************************************************
******************************************
  signal SIMULATION : boolean := TRUE;    -- set to TRUE only if
running Modelsim, else FALSE (controls debounce)
  constant RWAIT : integer := 5;     -- use 5 for sim, 500 for
synthesis
  --
**********************************************************************
******************************************

  component CLKINT port(
        A : in  std_logic;
        Y : out std_logic
    ); end component;

  -- internal signals for top module
  signal RESET : std_logic;          -- detect reset condition from
external lines
  signal DORESET : std_logic_vector(3 downto 0) := x"0"; -- FLASH
memory reset sequence state variable
  signal DOSTATUS : std_logic_vector(3 downto 0) :=x"0"; -- Read
status
  signal NFSTATUS : std_logic_vector(7 downto 0) := x"00"; -- status
byte
  signal WAITRESET : integer range 0 to 1023 := 0; -- to create delay
between board and chip reset functions, see below more info
  signal DOWE  : std_logic := '0';       -- send NF_WE(b) negative
pulse in 1st half of clock cycle
  signal DORE : std_logic := '0';
  signal NFBUSY : std_logic := '0';      -- set to 1 when an operation
in progress on nand flash, or waiting NF_RB
  signal NF_RB_LAST : std_logic := '1';  -- NF_RB from last clock
cycle (used to find when goes high after going low)
  signal SW_DB : std_logic_vector(9 downto 0) :=(others => '0');   --
Debounce switches
  signal DBCTR: std_logic_vector(19 downto 0):=(others => '0'); --
Counter for debounce
  signal SB9LAST : std_logic := '0';     -- transition 0->1 on SW9
initiates WRITE operation
  signal SB0LAST : std_logic := '0';     -- any transition reset
nfbusy
  signal DOWRITE : std_logic_vector(3 downto 0) := x"0"; -- state
variable for PROGRAM PAGE
  signal DOERASE : std_logic_vector(3 downto 0) := x"0"; -- state
variable for ERASE BLOCK
  signal DONEALE : boolean := false;     -- termination condition for
sending address bytes subroutine
  signal DOREAD : std_logic_vector(3 downto 0) := x"0"; -- state
variable for READ operation
```

```
  signal NFALE : std_logic := '0';     -- internal version of NF_ALE
so we can read it
  signal NFCLE : std_logic := '0';     -- internal version of NF_CLE
so we can read it
  signal tADL : integer range 0 to 3 := 0; -- counter for 3 clocks of
post address delay
  signal tWHR : integer range 0 to 4 := 0; -- counter for 4 clocks of
post read status command
  type BYTEARRAY5 is array (5 downto 1) of std_logic_vector(7 downto
0);
  signal NFADR : BYTEARRAY5 := (x"00",x"00",x"00",x"00",x"00"); --
Flash address
  alias NFCMD : std_logic_vector(7 downto 0) is NF_D(7 downto 0); --
command byte
  signal iadr : integer range 1 to 6 := 1; -- index for 5 bytes of
address info (has to be integer to be an array index)
  signal FDATA : std_logic_vector(15 downto 0):=(others => '0'); --
Flash data
  type WORDARRAY255 is array (255 downto 0) of std_logic_vector(15
downto 0);--*****************
  signal FDATAIN : WORDARRAY255;         -- buffer for read
data**************
  signal Fstartwrite : BOOLEAN := false;     -- flag this as a write
cycle (address states shared with read/write/erase cycle)
  signal FI : integer range 0 to 255 := 0; -- index to FDATAIN
  signal FI_MAX : integer range 0 to 255 := 255; -- max requested
input words
  signal Fstarterase : boolean := false;   -- flag this as a erase
cycle
  signal CLK2A : std_logic := '0';         -- half speed clock toggles
on leading edge of CLK (for delay compensation of WE)
  signal CLK2B : std_logic := '0';       -- half speed clock toggles
on falling edge of CLK
  signal CLKINV: std_logic := '0';         -- invert clk

begin
  -- COMPONENT INSTANCES
  CLKINT_0: CLKINT port map (A=>CLK, Y=>SCLK); -- (for ACTEL)
  --SCLK <= CLK; -- (for XILINX)

  -- ASYNCHRONOUS LOGIC
  RESET <= '1' when PORB = '0' or RESB = '0' else '0';
  NF_CE <= '0';                -- permanently select our one flash
chip
  NF_WP <= '1';               -- never write protected
  NF_ALE <= NFALE;               -- copy internal version to port
  NF_CLE <= NFCLE;               -- copy internal version to port
  DONEALE <= NFALE = '0' and tADL = 0;   -- termination (return) from
address bytes subroutine
  OUTRESET <= '0' when DORESET = x"0" else '1';-- debug flag set if
RESET states active - pin 8
```

```
  OUTWRITE <= DOWRITE(0);          -- copy internal version to port -
pin 12
  OUTREAD <= DOREAD(0);            -- copy internal version to port -
pin 16
  OUTERASE <= DOERASE(0);          -- copy internal version to port -
pin 22
  OUTSTATE <= DOWRITE when DOWRITE /= x"0"
    else DOREAD when DOREAD /= x"0"
    else DOERASE when DOERASE /= x"0";

  -- the trouble with these is a trailing glitch since the DO signals
are later than the CLOCK
  --NF_WE <= CLK nand DOWE;            -- generate write to flash
clock during 1st half of clock cycle
  --NF_RE <= CLK nand DORE;            -- generate read from flash
clock the same way
  -- try to fix by creating an approximately equally delayed clock as
the xor of 2 half speed clocks
  NF_WE <= (CLK2A xor CLK2B) nand DOWE;  -- generate write to flash
clock during 1st half of clock cycle
  NF_RE <= (CLK2A xor CLK2B) nand DORE;  -- generate read from flash
clock the same way
  CLKINV <= not CLK;               -- invert clk

  -- CLOCKED LOGIC (SYNCHRONOUS) (sometimes called "sequential" which
is misleading)

  process (CLKINV) begin if rising_edge(CLKINV) then
    if RESET = '1' then
      CLK2B <= '0';
    else
      if CLK2A = '1' then          -- half speed clock on falling edge
for WE delay compensation
        CLK2B <= '1';
      else
        CLK2B <= '0';
      end if;
    end if;
  end if; end process;


  process (CLK) begin if rising_edge(CLK) then    -- INTERNAL CLK
CLOCK DOMAIN
    if RESET = '1' then
      LED <= SW;
      NF_D <= x"0000";
      NFBUSY <= '0';
      NFCLE <= '0';
      NFALE <= '0';
      CLK2A <= '0';
      DOERASE <= x"0";
      DOSTATUS <= x"0";
      DOWRITE <= x"0";
      DOREAD <= x"0";
      DORESET <= x"1";
      DOWE <= '0';
      DORE <= '0';
    else
      CLK2A <= not CLK2A;         -- half speed clock for WE delay
compensation
      NF_RB_LAST <= NF_RB;        -- track NF_RB last value for
detection of end of a low pulse
      DOWE <= '0';              -- write pulse is set to NONE unless
otherwise determined
      DORE <= '0';
      LED <= NFBUSY & '0' & FDATAIN(0)(7 downto 0); -- display
internal state var and data read on LED's
      if SW(0) = '1' then LED(7 downto 0) <= FDATAIN(0)(15 downto 8);
end if; -- use SW0 to display upper byte of data read
      if SW(1) = '1' then LED(7 downto 0) <= FDATAIN(1)(7 downto 0);
end if; -- use SW1 to display lower byte of 2nd data word read
      if SW(1 downto 0) = "11" then LED(7 downto 0) <= NFSTATUS; end
if; -- SW0 & 1 set to display STATUS

      if (SDO = "0000") then
        CNVRT <= '0';
      else
        CNVRT <= '1';
      end if;

      -- FLASH MEMORY TEST ROUTINES
      -- RESET:
      if DORESET = x"1" then
        WAITRESET <= WAITRESET + 1;    -- wait 1000 clocks after board
reset before resetting chip
        if WAITRESET >= RWAIT then   -- (reason is, flash board is
pulling WE down to about 1/2 Vdd, wait for recovery)
          DORESET <= DORESET + 1;
          WAITRESET <= 0;
        end if;
      elsif DORESET = x"2" then
        NFCLE <= '1';         -- put FLASH in COMMAND mode (one cycle
early just for insurance)
        DORESET <= DORESET + 1;
      elsif DORESET = x"3" then
        NFBUSY <= '1';          -- declare NF is busy
        NFCMD <= x"FF";         -- set command to be sent
        DOWE <= '1';          -- enable write pulse for command
        DORESET <= DORESET + 1;
                          --   before resetting NFBUSY
      elsif DORESET = x"4" then
        NFCLE <= '0';
        DORESET <= x"0";        -- reset is done, BUT we are still
waiting on NF_RB
```

```
      end if;

      -- Wait for NF_RB for all Flash operations
      if NF_RB_LAST = '0' and NF_RB = '1' then -- wait for NF_RB to go
low and then high again
        NFBUSY <= '0';      --
      end if;

      -- Switch Debounce logic
      DBCTR <= DBCTR + 1;
      if SIMULATION then
        if DBCTR(3 downto 0) = "0000" then SW_DB <=  SW; end if; --
simulation version
      else
        if DBCTR(19 downto 0) = x"00000" then SW_DB <=  SW; end if; --
real board version
      end if;

      -- MANUALLY ALLOW FORCE NFBUSY TO ZERO (since NF_RB not working
on flash board)
      SB0LAST <= SW_DB(0);
      if SB0LAST /= SW_DB(0) then
        NFBUSY <= '0';
      end if;

      -- WRITE / READ PAGE INITIATION: (should be a whole block, but
whatever for now ... TEST PURPOSES)
      SB9LAST <= SW_DB(9);        -- look for state change on switch
9 to initiate WRITE or READ
      if (SB9LAST = '0' and SW_DB(9) = '1') then
        -- DO A WRITE (sw8=1) or READ (sw8=0) or ERASE sw7=1 operation
or READ_STATUS sw78=11****
        DOWRITE <= x"1";                -- (address logic is same for
either)
        NFADR(1) <= x"00";                 -- Words 0 to 255
        NFADR(2) <= "00000" & SW(3) & '0' & SW(2);    -- subpage
within page , SW3=CA10=main/spare, SW7=CA8 = 0 or 256
        NFADR(3) <= '0' & SW(6) & "00000" & SW(4);    -- plane select
BA6, alternate page within block PA0
        NFADR(4) <= "000000" & SW(5) & '0';      -- ignore(do
nothing), ignored blocks are bad, BA9
        NFADR(5) <= "00000000";
                                   -- SWITCH DEBUG COMMAND DECODE
        Fstartwrite <= false;              -- default SW_DB(8 downto
7)= "00" is READ
        Fstarterase <= false;
        if SW_DB(8 downto 7) = "10" then
          Fstartwrite <= true;
        elsif SW_DB(8 downto 7) = "01" then
          Fstarterase <= true;
        elsif SW_DB(8 downto 7) = "11" then
          DOWRITE <= x"0";
          DOSTATUS <= x"1";
        end if;
      end if;

      -- COMBINED START CODE FOR WRITE / READ PAGE
      if DOWRITE = x"1" then
        NFCLE <= '1';                   -- set CLE high one cycle
early so test bench can set bus to z's
        DOWRITE <= DOWRITE + 1;
      end if;
      if DOWRITE = x"2"  and NFBUSY = '0' then
        NFCLE <= '1';
        DOWE<= '1';
        NFBUSY <= '1';
        NFALE <= '0';
        NFCMD <= x"00";                 -- load the READ command in
data register
        if Fstartwrite then               -- CHECK FOR WRITE PAGE
COMMAND NEEDED
          NFCMD <= x"80";
        elsif Fstarterase then             -- CHECK FOR ERASE PAGE
COMMAND NEEDED*****
          --  (set proper command and iadr<=3)
          NFCMD <= x"60";
          iadr <= 3;
        end if;
        DOWRITE <= DOWRITE + 1;
      end if;
      if DOWRITE = x"3" then
        NFALE <= '1';                  -- NFALE=1 goes to address
subroutine (see below)
        NFCLE <= '0';
        DOWRITE <= DOWRITE + 1;
      end if;
      if DOWRITE = x"4" then                -- delay tWC for memory
to recognize ALE
        if DONEALE then                 -- if done sending address
bytes . . .
          if Fstartwrite then              -- if WRITE go to put data
words routine
            DOWRITE <= x"9";
          elsif Fstarterase then           -- if ERASE, send D0 cmd,
etc. *************
            -- (send D0 command [fix test bench to set tWC], set
DOERASE ... then at DOERASE reset DOWE and quit)
            NFCMD <= x"D0";
            NFALE <= '0';
            NFCLE <= '1';
            DOWE <= '1';
            DORE <= '0';
            FI<= 0;
            DOWRITE <= x"0";
            DOERASE <= x"1";            -- switch to erase mode
            Fstarterase <= false;
          else                    -- default is read
            NFCMD <= x"30";
            NFALE <= '0';
            NFCLE <= '1';
            DOWE <= '1';
            DOWRITE <= x"0";
            DOREAD <= x"1";             -- switch to read mode
            FI <= 0;                -- initialize buffer index
          end if;
        end if;
      end if;
      if DOWRITE = x"9" then          -- put data words on bus
        DOWE <= '1';          -- send the latch data command
        NFCMD <= SW(1) & SW(0) & FDATA(5 downto 0); -- send a data
byte
        FDATA <= FDATA + 1;       -- make some test data!
        if FDATA >= 255 then DOWRITE <= DOWRITE + 1; end if;
        NFCLE <= '0';
      end if;
      -- need to send 85h command, adjust the colum address and send 2
bytes, then send 3 words of data (zeroes?) ****
      if DOWRITE = x"A" then
        NFCLE <= '1';           -- send command to FLASH
        NFCMD <= x"10";         -- set command to be sent
        DOWE <= '1';          -- enable write pulse for command
        DOWRITE <= DOWRITE + 1;
      elsif DOWRITE = x"B" then
        NFCLE <= '0';
        if NFBUSY = '0' then -- wait for RB to go low then high again
          DOWRITE <= x"0";
          -- send 70h to read status, wait tWHR, capture status word
on BUS ****
        end if;
      end if;

      -- READ PAGE COMMAND (use a general read random command)
      if DOREAD = x"1" then
        NFCLE <= '0';
        NF_D <= "ZZZZZZZZZZZZZZZZ";
        DOWE <= '0';
        if NFBUSY = '0' then
--        NFCLE <= '1';         -- send command to FLASH
--        NFCMD <= x"05";       -- set command to READ RANDOM
--        DOWE <= '1';        -- enable write pulse for command
          DOREAD <= DOREAD + 1;
        end if;
        DOREAD <= DOREAD + 1;
      end if;
      if DOREAD = x"2" then
        NFBUSY <= '1';
```

```
          DORE <= '1';          -- triggers NFRE pulse to read data on 
next clock cycle
          DOREAD <= DOREAD + 1;
        end if;
        if DOREAD = x"3" then
          DORE <= '1';
          FDATAIN(FI) <= NF_D;      -- capture data word
          if FI >= FI_MAX then
            DORE <= '0';
            FI <= 0;
            DOREAD <= x"0";       -- terminate read operation
          else
            FI <= FI + 1;
          end if;
        end if;


        -- ERASE Block COMMAND
        if DOERASE = x"1" then
          NFBUSY <= '1';
          NFCLE <= '1';
          DOERASE <= DOERASE + 1;
        end if;
        if DOERASE = x"2" then
          NFCMD <= x"60";
          DOWE <= '1';
          DOERASE <= DOERASE + 1;
        end if;
        if DOERASE = x"3" then
          NFCLE <= '0';
          NFALE <= '1';
          DOERASE <= DOERASE + 1;
        end if;
        if DOERASE = x"4" then
          NFALE <= '1';
          NFCMD <= NFADR(2);          -- send row address 1
          DOERASE <= DOERASE + 1;
        end if;
        if DOERASE = x"5" then
          NFALE <= '1';
          NFCMD <= NFADR(3);          -- send row address 2
          DOERASE <= DOERASE + 1;
        end if;
        if DOERASE = x"6" then
          NFALE <= '1';
          NFCMD <= NFADR(4);          -- send row address 3
          DOERASE <= DOERASE + 1;
        end if;
        if DOERASE = x"7" then
          NFALE <= '0';
          NFCLE <= '1';
          DOERASE <= DOERASE + 1;
        end if;
        if DOERASE = x"8" then
          NFCMD <= x"D0";
          DOERASE <= DOERASE + 1;
        end if;
        if DOERASE = x"9" then
          NFCLE <= '0';
          if NFBUSY = '0' then
            DOERASE <= x"0";
          end if;
        end if;

        -- READ STATUS
        if DOSTATUS = x"1" then
          NFBUSY <= '1';
          NFCLE <= '1';
          DOSTATUS <= DOSTATUS + 1;
        end if;
        if DOSTATUS = x"2" then
          NFCMD <= x"70";
          DOWE <= '1';
          DOSTATUS <= DOSTATUS + 1;
        end if;
        if DOSTATUS = x"3" then
          NFCLE <= '0';
          DOWE <= '0';
          tWHR <= 0;
          NF_D <= "ZZZZZZZZZZZZZZZZ";
          DOSTATUS <= DOSTATUS + 1;
        end if;
        if DOSTATUS = x"4" then
          if tWHR < 4 then
            tWHR <= tWHR + 1;           -- wait for at least 80 ns
          else
            DORE <= '1';
            DOSTATUS <= DOSTATUS + 1;
          end if;
        end if;
        if DOSTATUS = x"5" then
          NFSTATUS <= NF_D(7 downto 0);      -- capture status byte
          DORE <= '0';
          DOSTATUS <= DOSTATUS + 1;
        end if;
        if DOSTATUS = x"6" then
          DOSTATUS <= x"0";
        end if;
        -- READ BAD BLOCK INFO (use a general read random command)


        -- SUBROUTINES
        -- SEND 5 ADDRESS BYTES (triggered by setting NFALE <= '1' ...
MUST be used)
```

```
      if NFALE = '1' then          -- caller must send NF_ALE and NOT
any data or command to implement tWC delay
        NF_D(15 downto 8) <= x"00";    -- LOW on upper bus bits
        NF_D(7 downto 0) <= NFADR(iadr);-- put next address byte on
bus
        -- (SHOULD STAY 1 UNTIL WE CHANGE IT) NF_ALE <= '1';
  -- send ALE
        DOWE <= '1';          -- send WE
        if (DOWE = '1') then
          iadr <= iadr + 1;     -- increment address byte index (after
iadr=0 has been sent)
        end if;
        if iadr >= 5 then       -- if have sent all address bytes ...
          --(WE DON'T KNOW CALLER) DOWRITE <= DOWRITE + 1; -- old
termination condition for inline version
          DOWE <= '0';        -- on NEXT cycle, we will not be sending
anything
          NFALE <= '0';       -- finished with address bytes
(termination condition for subroutine is caller must check NF_ALE='0'
and tADL=0)
          iadr <= 1 ;         -- reset iadr for next time
          tADL <= tADL + 1;       -- increment counter to start tADL
delay
        end if;
      end if; -- NFALE = '1'
      if tADL > 0 then          -- implement 3 cycle tADL delay
following address bytes
        tADL <= tADL + 1;
        if tADL >= 2 then
          tADL <= 0;            -- finished with tADL (termination
condition for subroutine is caller must check NF_ALE='0' and tADL=0)
        end if;
      end if; -- tADL > 0
      -- end of SEND ADDRESS BYTES (including tADL delay)

    end if;
  end if; end process;
end behavior;
```

Test Bench for VHDL ONFI Controller

```
----------------------------------------------------------------------
----------
-- Company:
-- Engineer:
--
-- Create Date:   18:02:17 08/21/2012
-- Design Name:
-- Module Name:   D:/Xilinx/test1/test1_tb.vhd
-- Project Name:  test1
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: test1
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types
std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx
recommends
-- that these types always be used for the top-level I/O of a design
in order
-- to guarantee that the testbench will bind correctly to the post-
implementation
-- simulation model.
----------------------------------------------------------------------
----------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY test1_tb IS
END test1_tb;

ARCHITECTURE behavior OF test1_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT test1
    PORT(
  -- dev kit signals
    CLK   : IN   std_logic;            -- E4
    RESB  : IN   std_logic;            -- T9
  PORB  : IN std_logic;            -- V7
  SW    : IN   std_logic_vector(9 downto 0);  --
D18,D17,E17,F16,D15,G14,E14,F14,G13,D14 (0 .. 9 order)
    LED   : OUT  std_logic_vector(9 downto 0);  --
E10,F10,U8,W5,U7,V6,U5,U4,V4,V5 (0 .. 9 order)
  -- ADC/NAND FLASH board
  SDO   : IN std_logic_vector(4 downto 1);  -- R16,T16,T1,U1 (1 .. 4
order)
  CNVRT : OUT  std_logic;              -- M3
  SCLK  : OUT  std_logic;            -- W2
  NF_WP : OUT  std_logic;            -- AB13  write protect active low
(WPB)
  NF_WE : OUT  std_logic;            -- AB6 write enable (WEB) host ->
flash when 0
  NF_ALE: OUT  std_logic;            -- AB12  address latch enable
  NF_CLE: OUT  std_logic;            -- AA7 command latch enable
  NF_CE : OUT  std_logic;            -- AA16 chip enable active low
(CEB)
  NF_RE : OUT  std_logic;            -- AA6 read enable active low
(REB) - flash -> host when 0 (opposite of WE)
  NF_RB : IN  std_logic;             -- Y2  ready/busy active low 0 =>
ready
  NF_D  : INOUT std_logic_vector(15 downto 0);   --
A17,A11,B13,B6,B17,B10,AB9,AB17,A7,A16,A10,B16,B7,AB8,AB16,AB7 (0 ..
15 order)
  -- Connect P1(could not use connector P2 due to it being lvds diff
pair)
  OUTRESET : OUT std_logic;          -- G11 pin 8
  OUTWRITE : OUT std_logic;          -- H11 pin 12
  OUTREAD : OUT std_logic;           -- G12 pin 16
  OUTERASE : OUT std_logic;          -- D16 pin 22
  OUTSTATE : OUT std_logic_vector(3 downto 0) -- E15 pin 24, N5 pin
26, K4 pin 28, K6 pin 30
    );
    END COMPONENT;


   --Inputs
   signal CLK : std_logic := '0';
   signal SW  : std_logic_vector(9 downto 0) := (others => '0');
   signal RESB: std_logic := '1';
   signal PORB: std_logic := '1';
   signal SDO : std_logic_vector(4 downto 1) := (others => '0');
   signal NF_RB : std_logic := '0';

  --Outputs
   signal LED   : std_logic_vector(9 downto 0);
   signal SCLK  : std_logic;
   signal CNVRT : std_logic;
   signal NF_WP : std_logic;
   signal NF_WE : std_logic;
   signal NF_ALE: std_logic;
   signal NF_CLE: std_logic;
   signal NF_CE : std_logic;
   signal NF_RE : std_logic;
   signal OUTRESET : std_logic;
   signal OUTWRITE : std_logic;
   signal OUTREAD : std_logic;
   signal OUTERASE : std_logic;
   signal OUTSTATE : std_logic_vector(3 downto 0);


   --In/Out
   signal NF_D  : std_logic_vector(15 downto 0) :=(others => 'Z');

   -- internal signals and constants
   constant CLK_period : time := 25 ns;         -- clock period
definition 40 MHz

  -- flash memory model signals
  signal tWB: std_logic_vector(11 downto 0) := (others => '0');  --
reset delay signal
  signal tRST: std_logic_vector(11 downto 0) := (others => '0'); --
reset delay signal
  signal tADL: std_logic_vector(11 downto 0) := (others => '0'); --
write delay signal
  signal tWC: std_logic_vector(11 downto 0) := (others => '0'); --
write delay signal
  signal tBERS: std_logic_vector(11 downto 0) := (others => '0'); --
erase delay signal ********
  alias BUS8: std_logic_vector(7 downto 0) is NF_D(7 downto 0);
  signal WElast: std_logic := '0';
  signal RElast: std_logic := '0';                -- track rising edge
of WE
  type Flashpage is array (1055 downto 0) of std_logic_vector(15
downto 0);
  type Flashmem8 is array (3 downto 0) of Flashpage; -- matches
Pageadr
  signal FMEM : Flashmem8 := (others =>(others => x"00FF")); -- Flash
address
  signal Wordadr : integer range 0 to 1055 := 0;
  signal Pageadr : integer range 0 to 3 := 0;
  signal Adrbyte : integer range 1 to 6 := 1;
  signal Badpage : boolean := false;   -- page address out of range,
will return bad block
  type Functiontyp is (write, read, readID, readStatus, erase, error);
  signal FUNC : Functiontyp := error;


BEGIN
```

```
-- Instantiate the Unit Under Test (UUT)
 uut: Test1 PORT MAP (
        CLK   => CLK,
    RESB  => RESB,
    PORB  => PORB,
        SW    => SW,
        LED   => LED,
    CNVRT => CNVRT,
    SDO   => SDO,
    SCLK  => SCLK,
    NF_D  => NF_D,
    NF_RB => NF_RB,
    NF_WP => NF_WP,
    NF_WE => NF_WE,
    NF_ALE=> NF_ALE,
    NF_CLE=> NF_CLE,
    NF_CE => NF_CE,
    NF_RE => NF_RE,
    OUTRESET => OUTRESET,
        OUTWRITE => OUTWRITE,
        OUTREAD => OUTREAD,
    OUTERASE => OUTERASE,
    OUTSTATE => OUTSTATE
      );

 -- Clock process definitions
 CLK_process :process
 begin
  CLK <= '0';
  wait for CLK_period/2;
  CLK <= '1';
  wait for CLK_period/2;
 end process;

 --Memory process definitions
 rst :process begin
 --set only in signals
  RESB <= '0';
  PORB <= '0';
  SDO <= "1000";
  SW <= "0000000000";
  wait for CLK_period*10;
  RESB <= '1';
  PORB <= '1';
  wait for 1000 ns;
  -- simulate noise switch to trigger write block
  SW(9) <= '1';
  wait for 55 ns;
  SW(9) <= '0';
  wait for 55 ns;
  SW(9) <= '1'; -- some operation
```

```
    SW(8) <= '1'; -- write page
    wait for 8500 ns;
    SW(8) <= '0';
    SW(9) <= '0';
    wait for 500 ns; -- wait for debounce
    SW(9) <= '1'; -- some operation (read)
    SW(0) <= '1';
    SW(1) <= '0';
    wait for 8500 ns;
    SW(9) <= '0';
    wait for 500 ns;
    SW(9) <= '1';
    SW(7) <= '1';  -- erase
    wait for 1000 ns;
    SW(9) <= '0';
    SW(7) <= '0';
    wait for 1000 ns;
    SW(9) <= '1';
    SW(8) <= '1';
    SW(7) <= '1';
    wait for 500 ns;
    SW(9) <= '0';
    wait;
   end process rst;

  fmemp: process(CLK, NF_WE) begin
    if falling_edge(CLK) then
      if RESB = '0' or PORB = '0' then NF_RB <= '1'; end if; -- reset
NF_RB
      WElast <= NF_WE;                    -- look for rising edge of
WE
      RElast <= NF_RE;                    -- look for rising edge of
RE
      if NF_CLE = '1' then                   -- set's bus to z's when
command is set
        NF_D <= "ZZZZZZZZZZZZZZZZ";
      end if;

      -- PROCESS INFORMATION ON BUS FROM FPGA
      -- WE rising edge strobes all data from FPGA to memory
      if NF_WE = '0' and WElast = '1' then          -- FPGA is
presenting data on bus, we should latch something
                                     -- (note: we assume WE will come
up immediately, so we do not have to copy everything)
                                     -- (if we WAIT for WE to come up,
we'd have to have copied all signals on bus)
        WElast <= '1';                    -- if we had a rising edge
of WE, then use first half cycle WE=1 state, not end cycle state
        if NF_CLE = '1' then                 --   IT IS A COMMAND ----
----------------------------------------
          if BUS8 = x"FF" then               --     RESET
```

```
             tWB <= x"001";                --       start tWB counter
to time RB response
           elsif BUS8 = x"10" then              --   END OF DATA
             tWB <= x"001";                --      use same mechanism
to do RB response
             FUNC <= error;                --      finished with write
             Adrbyte <= 1;--****
             Wordadr <= 0; --*****
           elsif BUS8 = x"80" then              --   PROGRAM PAGE
             Adrbyte <= 1;
             Pageadr <= 0;
             Wordadr <= 0;
             Badpage <= false;
             FUNC <= write;
           elsif BUS8 = x"85" or BUS8 = x"05" then     --   CHANGE
WRITE or READ COLUMN
             Wordadr <= 0;
             Adrbyte <= 1;
           elsif BUS8 = x"00" then              --   READ OPERATION
             Adrbyte <= 1;
             Pageadr <= 0;
             Wordadr <= 0;
             Badpage <= false;
             FUNC <= read;
           elsif BUS8 = x"30" then
             tWB <= x"001";                --      use same mechanism
to do RB response
             Badpage <= false;
             FUNC <= read;                 --      finished with write
           elsif BUS8 = x"60" then              --  ERASE (not erasing
anything)
             Adrbyte <= 3;
             Pageadr <= 0;
             Wordadr <= 0;
             Badpage <= false;
             FUNC <= erase;
           elsif BUS8 = x"D0" then
             tWB <= x"001";
           elsif BUS8 = x"70" then
             tWB <= x"001";
             Badpage <= false;
             FUNC <= readStatus;
           end if;
         elsif NF_ALE = '1' then                --   IT IS AN ADDRESS
BYTE ------------------------------
           if Adrbyte = 2 and BUS8(0)= '1' then
             Wordadr <= 512;
           elsif Adrbyte = 3 then
             if BUS8(6) = '1' then
               Pageadr <= 1;
             end if;
             if (BUS8 and "10111111") /= "00000000" then
```

```
              Badpage <= true;
            end if;
          end if;
          if Adrbyte > 3 then
            if BUS8 /= "00000000" then
              Badpage <= true;
            end if;
          end if;
          Adrbyte <= Adrbyte + 1;
        else                        --   IT IS A DATA WORD (16 bits)
---------------------------
          case FUNC is
            when write =>
              if Badpage = false then
                FMEM(Pageadr)(Wordadr) <= NF_D;
                if Wordadr < 1055 then Wordadr <= Wordadr + 1; end
if;
              end if;
            when others =>
              Assert TRUE Report "invalid function for data
operation" Severity Error;
          end case;
        end if;
      end if; -- WE RISING EDGE

      -- PUT READ DATA ON BUS TO FPGA (in response to rising edge of
RE which is set by FPGA)
      if NF_RE = '0' and RElast = '1' then          -- FPGA is
requestion data, we should put something on bus
        RElast <= '1';                    -- if we had a rising edge
of RE, then use first half cycle RE=1 state, not end cycle state
        if FUNC = readStatus then
          NF_D <= x"0010";                -- dummy status
        elsif NF_ALE = '0' and NF_CLE = '0' then
          case FUNC is
            when read =>
              --Assert Badpage Report "attempt to read from invalid
page" Severity Error;
              if Badpage = false then
                NF_D <= FMEM(Pageadr)(Wordadr);
                if Wordadr < 1055 then Wordadr <= Wordadr + 1; end
if;
              end if;
            when others =>
              Assert TRUE Report "invalid function for data
operation" Severity Error;
          end case;

        end if;
      end if; -- RE RISING EDGE

      -- PROCESS MY OWN STATE TRANSITIONS AND RESPONSES
      if tWB > 0 then                      -- IF DOING NF_RB delay
after WE for RESET command:
        if tWB >= 5 then                   --   wait 5 clocks (approx
100 ns)
          tWB <= x"000";                   --   terminate the after-WE
delay
          NF_RB <= '0';                  --   drop RB to match timing
diagram
          tRST <= x"001";                  --   start a new ctr to
time RB low duration
        else
          tWB <= tWB + 1;                  --   perform RB delay
        end if;
      end if;
      if tRST > 0 then                     -- IF DOING NF_RB DURATION
delay after RESET command
        if tRST >= 15 then                   --   wait 15 clocks (real
delay would be much larger)
          tRST <= x"000";                  --   terminate counter
          NF_RB <= '1';                  --   terminate RB
        else
          tRST <= tRST + 1;                --   perform RB duration
delay
        end if;
      end if;
    end if;
  end process fmemp;

END;
```

References

Datasheets

[1]"AD7980 Datasheet." Analog Devices, 2007. Web. 2009. <www.analog.com>.
[2]"ARM Cortex-M1-Enabled IGLOO." Microsemi, 2012. Web. 2013. <Microsemi.com>.
[3]"NAND Flash Memory." Micron, 2009. Web. 2013. <www.micron.com>.
[4]"Parallel Digital-To-Analog Converter with Power Down." Texas Instruments, Feb. 2004. Web. 2013. <www.ti.com>.

Web

[5]*GerberTool*.WiseSoftware,2012.Web.2013.<http://www.wssi.com/index.php?option=com_content&task=section&id=16&Itemid=167>.
[6]Harris, Daniel. "NAND Flash Memory." *Electronic Design Home Page*. Electronic Design, 13 Sept. 2007. Web. Apr. 2013. <http://electronicdesign.com/digital-ics/nand-flash-memory>.
[7]*Partial Page Programming*. Spansion, 14 Nov. 2005. Web. Apr. 2013. <http://www.eettaiwan.com/STATIC/PDF/200809/EETOL_2008IIC_Spansion_AN_70.pdf?%20SOURCES=DOWNLOAD>.
[8]"Open Source Simulation Models for System Level Verification." *Free Model Foundry*. N.p., 18 May 2010. Web. 25 Apr. 2013. <http://www.freemodelfoundry.com/>.
[9]" The Ultra-low-power Programmable Solution." *IGLOO FPGAs: IGLOO FPGA*. Microsemi, 2013. Web. 25 Apr. 2013. <http://www.actel.com/products/igloo/>.
[10]"VHDL." *Wikipedia*. Wikimedia Foundation, 23 Apr. 2013. Web. 25 Apr. 2013. <http://en.wikipedia.org/wiki/VHDL>.

Actel Tech

[11]Pingili, Rohini. Email Response. March 2013