Solid State Drive: Software and Hardware Design

18-545: Digital Design Project

Final Report

Kun Qian (kqian@andrew.cmu.edu)
Jihoon Kim (jihoonk@andrew.cmu.edu)

December 9, 2010

Contents

List of Figures

Abstract

Significant progress, in both software and hardware, has been made in previous iterations of this project in attempt to build a fully functional Solid State Drive. However, when our team attempted to pick up where the previous teams have left off, we found it very hard to fully utilize the progresses made by previous teams either due to the lack of clear documentation, or simply due to miscommunication during the project hand off. To remedy such effects for future iterations of this project, we will try to provide a clear and concise documentation on the qualifications we feel are needed to take on this project, as well as the process of bringing a new system up to speed with our current progress. We will also focus on describing the PCI-E interface as we feel it is the component that was missing from all previous documentations.

1 Introduction

What constitutes a fully functional Solid State Drive? Invariants to any SSD implementation includes a memory storage device, a memory controller for the storage device, firmware for the memory controller, a device driver for the host machine. Recognizing the complexity and significance of each component early on in the design process is very important, since the system cannot function with any component misbehaving, and each component will likely need to be developed separately but they must tested collectively throughout the development process. A brief overview of each component is provided below, note that host device refers to the device which comprises of all components that makes up an SSD, host machine refers to the actual machine that the SSD is plugged into:

1.1 Memory Storage Device

Primary storage unit, most commonly use NAND based flash chips. In addition to the storage units, the device must also provide signal translation units if there exists voltage differences between the chip signals and the host device bus. The entire device is then connected to the IO bus on the host device. Note that flash memory have different organization than traditional memory storage devices, background knowledge of flash memory is definitely required in understanding how to operate the memory storage device.

1.2 Memory Controller

Resides as reprogrammable hardware modules in the host device, acts as both administrator and caretaker of the memory storage device. Handles the transmission of request signals to the memory chips, as well as response signals from the memory chips. Since NAND flash cells can endure limited amount of erase cycles before becoming unreliable, it is crucial for the memory controller to provide wear leveling, an algorithm to monitor the health of the individual flash blocks, and avoid wearing out any individual block prematurely. Memory controller needs to be finished early, since it requires long duration synthesis, the idea is to have the memory controller emit all signals supported by the memory device, and handle the forwarding of all signals from the memory device onto the IO bus. Requires background in flash memory, Verilog, VHDL and memory scheduling algorithms.

1.3 Firmware for Memory Controller

Although the memory controllers are usually reprogrammable in the design project, they should not be in a real implementation of SSD, mainly due to the fact that reprogrammable hardware modules usually require power to maintain (e.g. FPGA implementations), and to improve performance they are usually fabricated. Since the memory controller are invariant, there would need to be something that can adapt to different host machine environments, acting as the translator of requests from the host machine to the SSD.

Firmware for the memory controller are C code run by an onboard core in the host device. It acts as the translator between the host machine's requests, and the memory controller's responses. Background knowledge required are embedded software development and basic knowledge of operating systems.

1.4 (Block) Device Driver for Host Machine

When the SSD is plugged into the PCI-E slots of the host machine, it is the device driver's job to recognize the SSD and handle application generated requests to the SSD. A device driver is different from a block device driver in that the sole purpose of the device driver is to recognize the SSD and only enabling a specific set of application to issue requests to the SSD, whereas block device driver will setup translation of traditional memory request signals to the ones specified by the memory controller firmware, and mount the SSD as a block memory storage device, thus enabling all applications to issue requests to the SSD through the OS.

1.5 Contributions

This paper makes the following main contributions:

1. Clean concise documentation to the process of establishing a benchmarked debuggable platform across the PCI-E
2. Clear definition of what needs to be done for the Memory Controller
3. Source code that is easy to read and understand
4. Consolidation of all previous efforts into one project report

By now you should have a good idea of what to expect if you choose to take this project. If you have what it takes, read on.

2 Initial Setup

This section will describe the steps to take in order to obtain the source code from our repositories and setup a testing framework within an hour.

2.1 How to get the latest repo

All project files are hosted in a svn repo at

```
/afs/ece/project/km_group/svn/src/ssd_repo
```

In order to gain access to this repo you must first contact a repo admin (e.g.) Yu Cai <yucai@andrew.cmu.edu>, the admin will need to give you access to the repo by running the following shell script:

```
#####
#!/bin/sh
fs sa /afs/ece/project/km_group [your_andrewid] rl
fs sa /afs/ece/project/km_group/svn [your_andrewid] rl
fs sa /afs/ece/project/km_group/svn/src [your_andrewid] rl
cd /afs/ece/project/km_group/svn/src/ssd_repo
find -type d -exec fs sa {} [your_andrewid] rl ';'
#####
```

Once the admin has run the shell script giving you the correct privileges, you may execute the following commands on afs.

```
mkdir ssd
cd ssd; svn checkout file:///afs/ece/project/km_group/svn/src/ssd_repo
```

You now have access to all the latest files in this project.

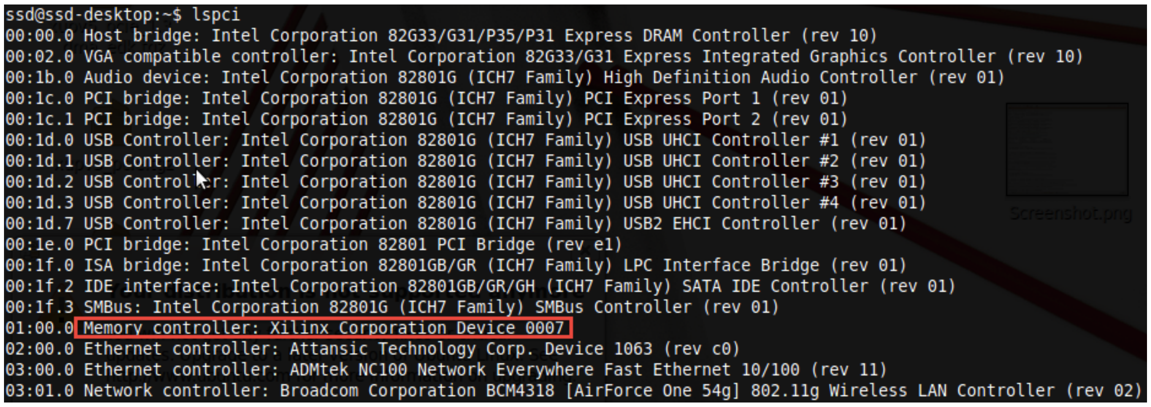
2.2 How to bring up the PCI-E across Virtex 5

In this project we focused on bringing up the PCI-E of the Virtex 5 board, for similar directions on how to do this for Virtex 6 see Appendix B. Bringing up PCI-E can be a pain initially if you do not know exactly what is happening in the background. Heres a brief overview:

Prior to attempting to debug the PCI-E interface the FPGA must be plugged into the PCI-Ex8 slot on your motherboard, and completely programmed with the memory controller and firmware running. Follow the memory controller section, Section 3.1 for more details. The host machine needs to be rebooted every time changes are made onboard the FPGA, simply because whenever bitstreams change on the FPGA, the host machine needs to run POST to detect hardware changes. This is only done at boot time.

By "bringing up PCI-E" we mean the process of getting the host machine to recognize the XUPv5 board plugged into one of its PCI-E slots, and enabling the PCI-E bus to be initialized to test for sending and receiving data through PCI-E. However, this is just a higher level description of a much more complex process. The entire process can be split into three stages:

1. Loading the memory controller into XUPv5
   This stage involves synthesizing the memory controller modules, and programming them into the FPGA. (For a complete step by step guide on how to synthesize the



```
ssd@ssd-desktop:~$ lspci
00:00.0 Host bridge: Intel Corporation 82G33/G31/P35/P31 Express DRAM Controller (rev 10)
00:02.0 VGA compatible controller: Intel Corporation 82G33/G31 Express Integrated Graphics Controller (rev 10)
00:1b.0 Audio device: Intel Corporation 82801G (ICH7 Family) High Definition Audio Controller (rev 01)
00:1c.0 PCI bridge: Intel Corporation 82801G (ICH7 Family) PCI Express Port 1 (rev 01)
00:1c.1 PCI bridge: Intel Corporation 82801G (ICH7 Family) PCI Express Port 2 (rev 01)
00:1d.0 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #1 (rev 01)
00:1d.1 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #2 (rev 01)
00:1d.2 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #3 (rev 01)
00:1d.3 USB Controller: Intel Corporation 82801G (ICH7 Family) USB UHCI Controller #4 (rev 01)
00:1d.7 USB Controller: Intel Corporation 82801G (ICH7 Family) USB2 EHCI Controller (rev 01)
00:1e.0 PCI bridge: Intel Corporation 82801 PCI Bridge (rev e1)
00:1f.0 ISA bridge: Intel Corporation 82801GB/GR (ICH7 Family) LPC Interface Bridge (rev 01)
00:1f.2 IDE interface: Intel Corporation 82801GB/GR/GH (ICH7 Family) SATA IDE Controller (rev 01)
00:1f.3 SMBus: Intel Corporation 82801G (ICH7 Family) SMBus Controller (rev 01)
01:00.0 Memory controller: Xilinx Corporation Device 0007
02:00.0 Ethernet controller: Attansic Technology Corp. Device 1063 (rev c0)
03:00.0 Ethernet controller: ADMtek NC100 Network Everywhere Fast Ethernet 10/100 (rev 11)
03:01.0 Network controller: Broadcom Corporation BCM4318 [AirForce One 54g] 802.11g Wireless LAN Controller (rev 02)
```

Figure 1: Expected output of lspci.

current project, follow Section 3.1) Doing so will enable the XUPv5 board emit the following lines to the BIOS upon restart of the host machine:

```
Memory controller: Xilinx Corporation Device
```

Caution: If the BIOS does not recognize the device plugged into the host machine's PCI-E slot, turn the host machine off completely, and turn off the XUPv5 board. And turn on the XUPv5 board first and program it, then turn on the host machine after the program is completed. Order is of the utter most importance here.

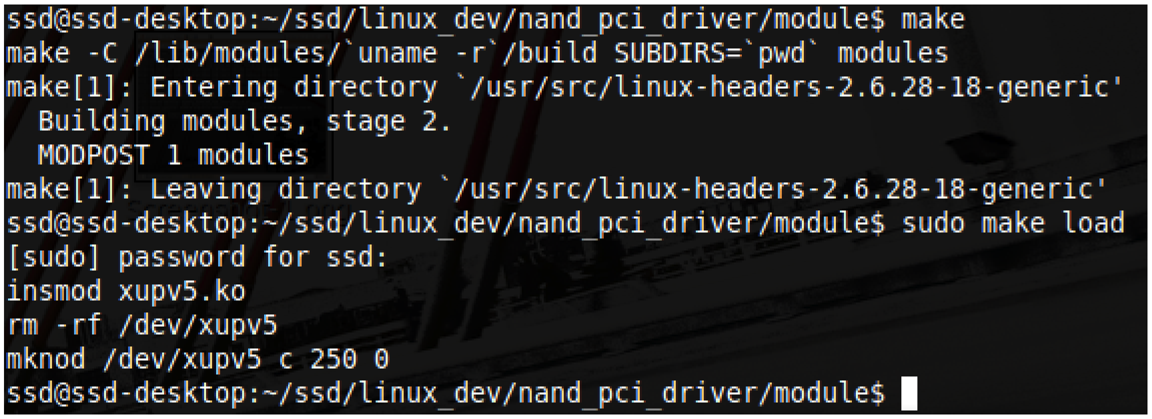
2. Loading the XUPv5 drivers into Linux Kernel
   After our BIOS correctly recognizes the XUPv5 board, boot up the host machine and double check with the following command:

   ```
   lspci | grep Xilinx
   ```

   You should see something similar to this line, the expected output is also shown in Figure 1

   ```
   01:00.0 Memory controller: Xilinx Corporation Device 0007
   ```

   Upon verification we need to allocate memory addresses to map to the PCI-E registers corresponding to the Xilinx device. This is done by a module named xupv5 made by Eric Cheung. We call this module the linux driver as per naming convention in previous reports. To load the linux driver run the following commands in the host machine, the expected output is shown in Figure 2:



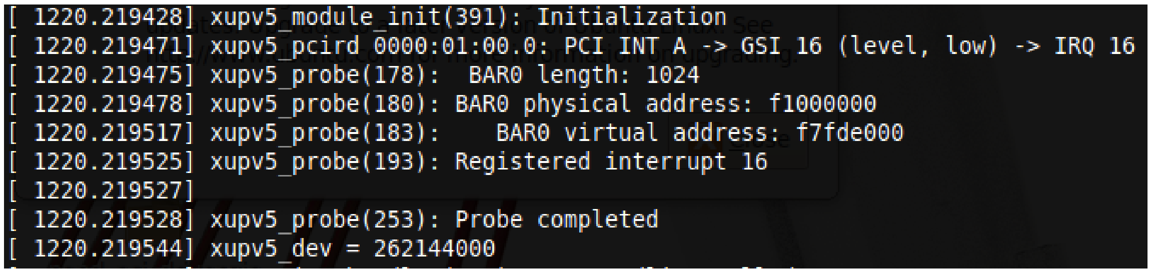
```
ssd@ssd-desktop:~/ssd/linux_dev/nand_pci_driver/module$ make
make -C /lib/modules/`uname -r`/build SUBDIRS=`pwd` modules
make[1]: Entering directory `/usr/src/linux-headers-2.6.28-18-generic'
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: Leaving directory `/usr/src/linux-headers-2.6.28-18-generic'
ssd@ssd-desktop:~/ssd/linux_dev/nand_pci_driver/module$ sudo make load
[sudo] password for ssd:
insmod xupv5.ko
rm -rf /dev/xupv5
mknod /dev/xupv5 c 250 0
ssd@ssd-desktop:~/ssd/linux_dev/nand_pci_driver/module$
```

Figure 2: Expected output of make;sudo make load.

```
cd <tree>/linux_dev/nand_pci_driver/module
make; sudo make load
```

If you see errors at this stage, do not proceed further, try restarting the host machine or reprogramming the FPGA. The first command gets into the module directory in the ssd repo, while the second command makes the module files and loads the module into the running linux kernel via insmod. A screen shot of the effect of the above command is shown below. If there were no errors printed on the screen after the execution, we can test if the module was probed correctly by running the following command, expected output are in Figure 3:

```
dmesg | grep xupv5
```



```
[ 1220.219428] xupv5_module_init(391): Initialization
[ 1220.219471] xupv5_pcird 0000:01:00.0: PCI INT A -> GSI 16 (level, low) -> IRQ 16
[ 1220.219475] xupv5_probe(178):  BAR0 length: 1024
[ 1220.219478] xupv5_probe(180): BAR0 physical address: f1000000
[ 1220.219517] xupv5_probe(183):    BAR0 virtual address: f7fde000
[ 1220.219525] xupv5_probe(193): Registered interrupt 16
[ 1220.219527]
[ 1220.219528] xupv5_probe(253): Probe completed
[ 1220.219544] xupv5_dev = 262144000
```

Figure 3: Expected output of dmesg.

Make sure you do not see errors in this screen. If errors exist, the memory controller was not detected correctly, and you need to restart the host machine, and reprogram the FPGA.

3. Run applications and verify requests to and response from PCI-E
   There are several benchmark programs that exists, the most commonly used, and the fastest program is the `nand_pci_driver`. It is a test for all the currently supported nand commands. For a full list of benchmark programs see Appendix A

You now have a working PCI-E interface.

2.3 Setting up debugging environment

Being able to debug is perhaps the simplest yet most important thing in any developing environment. Referring back to our system design, the only things visible to the programmer, if you followed our instructions so far, are the outputs from benchmark programs, which only has access to responses from the PCI-E bus. In order to see debug messages at the lower level (i.e. the firmware debug messages, and signal assertion in the memory controller) we must involve additional tools.

2.3.1 Serial port connection via HyperTerminal

The documentation from the Xilinx getting started guide is very good, but there are a few shortcomings, for example, they do not document the errors that may occur. Refer to the following for setting up serial port connection for both Virtex 5 and Virtex 6.

1. On the Virtex 5 board, set SW3 to 00010101 see Figure 4, ignore this step for Virtex 6

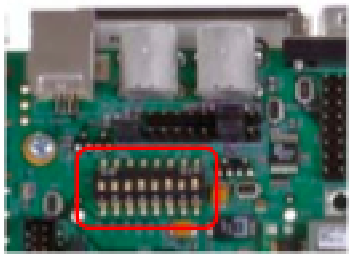


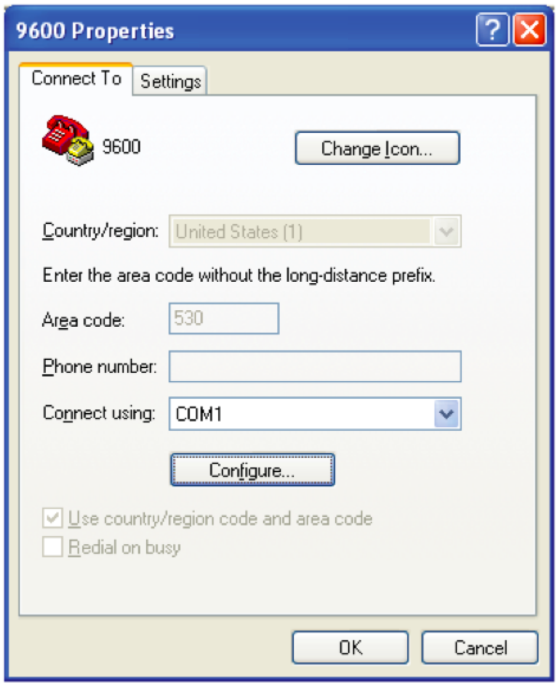
Figure 4: Pin configuration for SW3

2. Connect a modem serial cable between your Windows machine and the FPGA Board serial out.

   (a) On the windows machine, click Start → Program → Accessories → Communications → HyperTerminal

(b) Give any name to the Connection Description window, preferably something easy to remember, because you will need this over and over again.

(c) In the Connect To window click Cancel then select File → Properties. Be sure to select Connect using COM1. See Figure 5


9600 Properties

Connect To | Settings

9600

Change Icon...

Country/region: United States (1)

Enter the area code without the long-distance prefix.

Area code: 530

Phone number:

Connect using: COM1

Configure...

[✓] Use country/region code and area code

[ ] Redial on busy

OK | Cancel

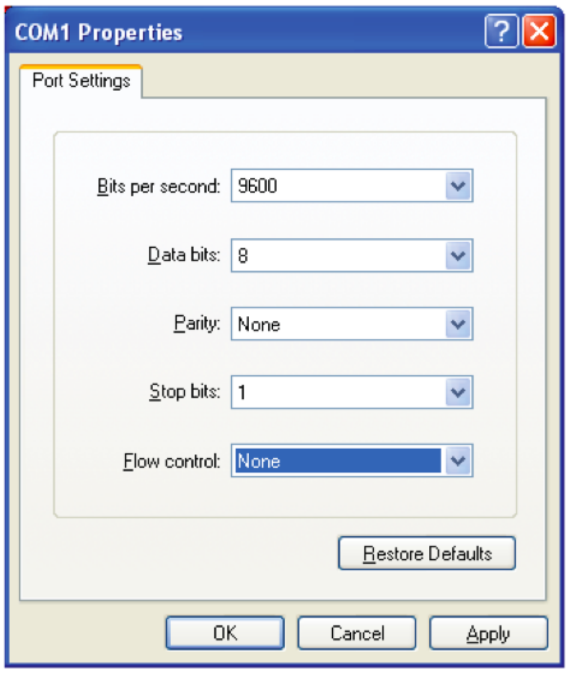
Figure 5: HyperTerminal Setup and Properties

(d) Click Configure and input the following settings. See Figure 6

i. Bits per second = 9600
ii. Data bits = 8
iii. Parity = None
iv. Stop bits = 1
v. Flow control = None
vi. Click OK to accept settings

(e) Select File → Properties.

i. Select the Settings tab and click on ASCII Setup. See Figure 7
ii. Character delay: 20 milliseconds
iii. Click OK to accept settings


COM1 Properties

Port Settings

Bits per second: 9600

Data bits: 8

Parity: None

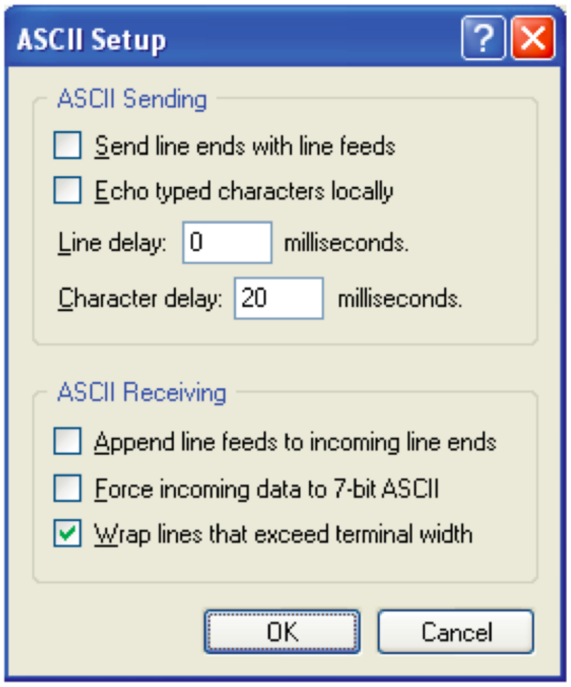
Stop bits: 1

Flow control: None

Figure 6: COM1 Properties

Your HyperTerminal should now display outputs from the firmware code. Sample firmware outputs from `nand_pci_driver` benchmark program are provided below:

```
TestApp PCIe-NAND -- Entering main() --
Got 1 Byte DMA Sync: 0
-----Starting HW Reset------
-----Finished HW Reset------

Waiting...commandloop: got garbage command
Waiting...command is TEST_READ --
----sent out data (8) bytes 0
Waiting...command is TEST_WRITE --
----got write data (8)bytes  70717075
Waiting...command is TEST_READ --
----sent out data (8) bytes 70717075
Waiting...command is NAND_RESET: Chip 1 --
-----Sending NAND RESET   cmd_reg = 8, status_reg = 4-----
     NAND RESET status = NAND_BUSY (2)
-----Finished NAND RESET: status = NAND_CMD_DONE (1)-----
```



ASCII Setup

ASCII Sending

- [ ] Send line ends with line feeds
- [ ] Echo typed characters locally

Line delay: 0 milliseconds.

Character delay: 20 milliseconds.

ASCII Receiving

- [ ] Append line feeds to incoming line ends
- [ ] Force incoming data to 7-bit ASCII
- [x] Wrap lines that exceed terminal width

Figure 7: ASCII Setup

```
Waiting...command is NAND_RESET: Chip 2 --
-----Sending NAND RESET   cmd_reg = 56, status_reg = 52-----
     NAND RESET status = NAND_BUSY (2)
-----Finished NAND RESET: status = NAND_CMD_DONE (1)-----

Waiting...command is NAND_RESET: Chip 3 --
-----Sending NAND RESET   cmd_reg = 104, status_reg = 100-----
     NAND RESET status = NAND_BUSY (2)
-----Finished NAND RESET: status = NAND_CMD_DONE (1)-----

Waiting...command is NAND_RESET: Chip 4 --
-----Sending NAND RESET   cmd_reg = 152, status_reg = 148-----
     NAND RESET status = NAND_BUSY (2)
-----Finished NAND RESET: status = NAND_CMD_DONE (1)-----

Waiting...command is NAND_READ_ID: Chip 1, Addr 0--
-----Sending NAND READ_ID: CMD_REG = 8, STATUS_REG = 4, DATA_BUF = 1C------
```

```
     NAND READ_ID: status = NAND_BUSY (2)
----- Finished READ_ID; status = NAND_CMD_DONE (1)
     Read_ID read val: D3902E64

----sent out data (8) bytes D3902E64
Waiting...command is NAND_READ_ID: Chip 1, Addr 20--
-----Sending NAND READ_ID: CMD_REG = 8, STATUS_REG = 4, DATA_BUF = 1C------
     NAND READ_ID: status = NAND_BUSY (2)
----- Finished READ_ID; status = NAND_CMD_DONE (1)
     Read_ID read val: 4E46494F

----sent out data (8) bytes 4E46494F
Waiting...command is NAND_READ_ID: Chip 2, Addr 0--
-----Sending NAND READ_ID: CMD_REG = 38, STATUS_REG = 34, DATA_BUF = 4C------
     NAND READ_ID: status = NAND_BUSY (2)
----- Finished READ_ID; status = NAND_CMD_DONE (1)
     Read_ID read val: D1902E60

----sent out data (8) bytes D1902E60
Waiting...
```

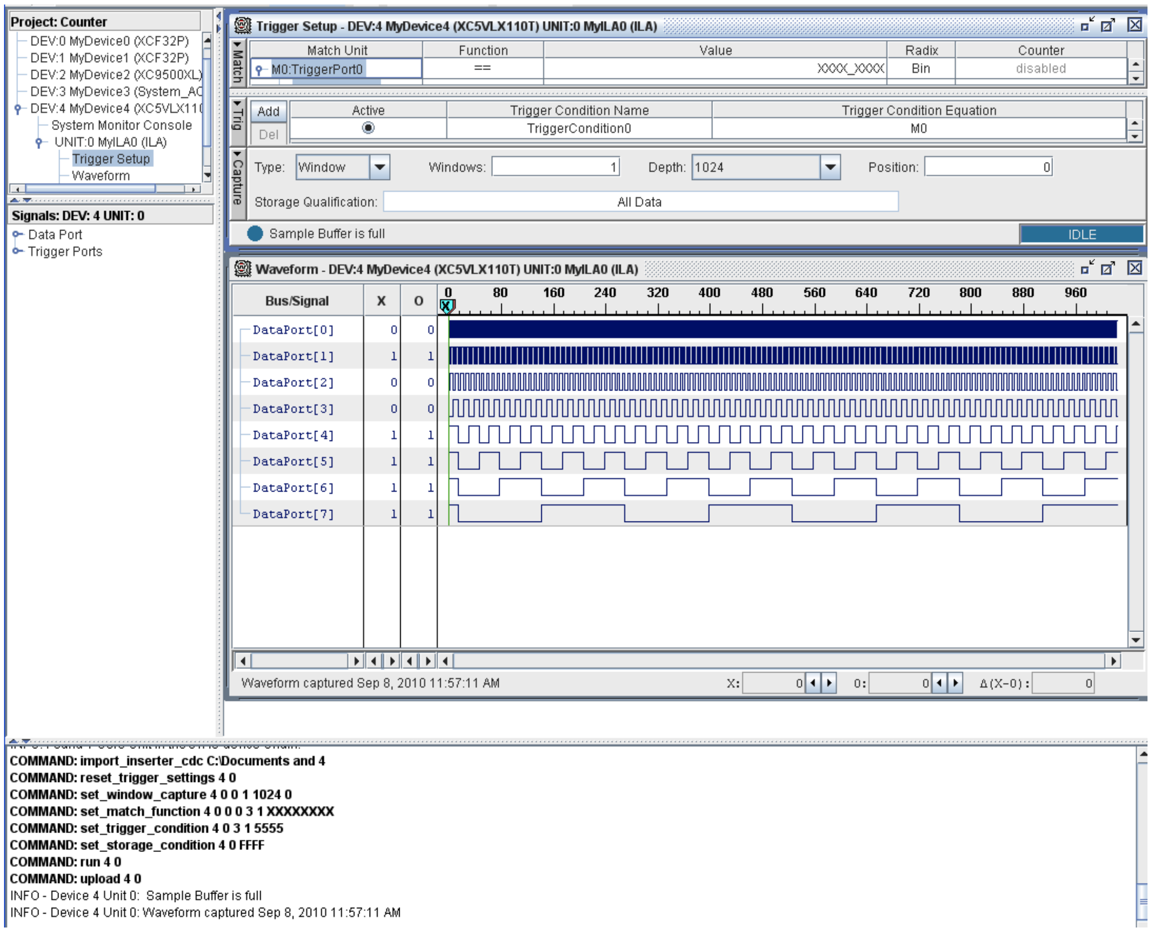
The above outputs are generally pretty self explanatory. A few note worthy register and signal names are clarified below:

NAND_RESET	Signal sent to nand chip to reset all signals, this is not the block reset signal
TEST_READ/WRITE	Test signal sent to nand chip to trigger sendback this is not the page read signal
CMD_REG	Command register on FPGA used to store incoming signals

2.3.2 Setting up Chipscope

Chipscope is one of the few Xilinx tools that prove to be worthy of the time spent to try to figure out how to make it work, and thus will not be discussed in detail here. We have included a picture of what the final result should look like see Figure 8

Note that Chipscope is called Chipscope Pro Analyzer under the Xilinx Tools. And the process involves creating an ILA Core and an ICON Core and inserting them into your design so that signals in your design could be monitored in real time based on trigger settings you specify.



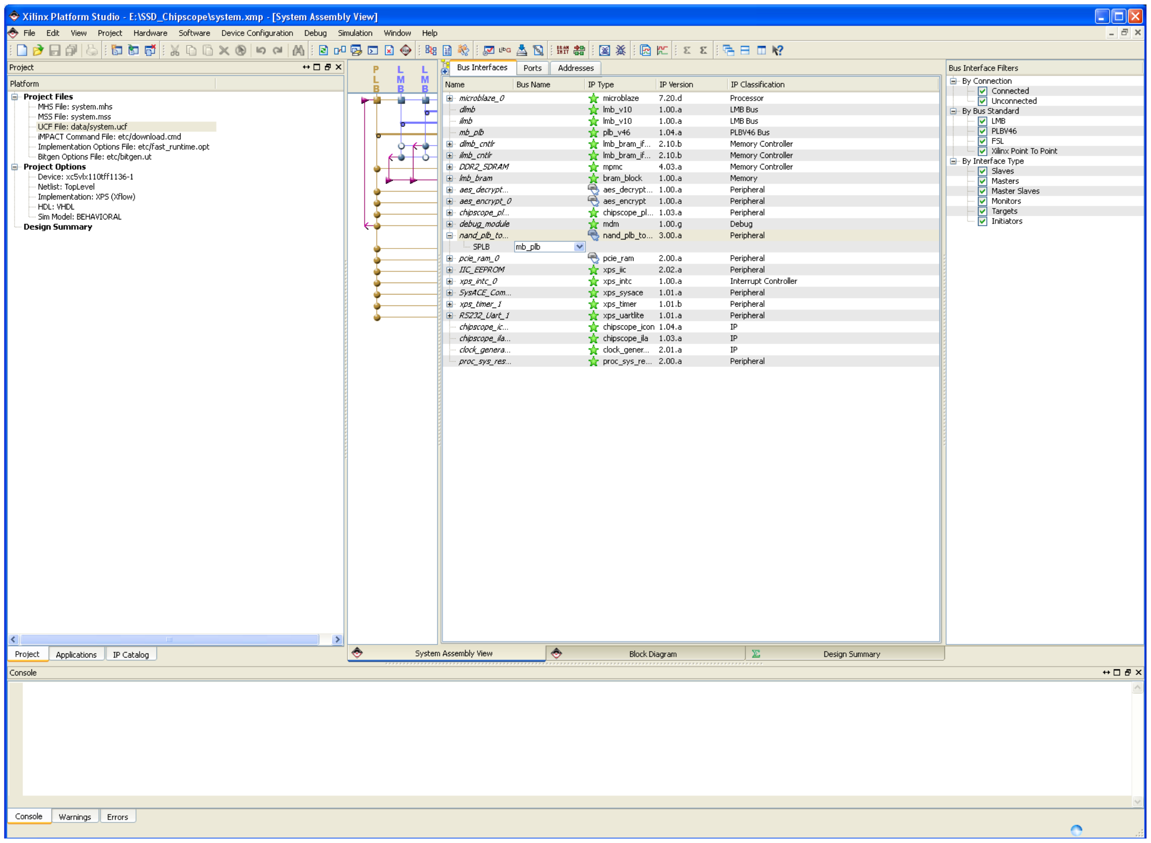
```
COMMAND: import_inserter_cdc C:\Documents and 4
COMMAND: reset_trigger_settings 4 0
COMMAND: set_window_capture 4 0 0 1 1024 0
COMMAND: set_match_function 4 0 0 0 3 1 XXXXXXXX
COMMAND: set_trigger_condition 4 0 3 1 5555
COMMAND: set_storage_condition 4 0 FFFF
COMMAND: run 4 0
COMMAND: upload 4 0
INFO - Device 4 Unit 0:  Sample Buffer is full
INFO - Device 4 Unit 0: Waveform captured Sep 8, 2010 11:57:11 AM
```

Figure 8: Chipscope for the Memory Controller

2.3.3 Setting up XPS project or Importing into an Existing XPS Project

We recommend new users of our project to use our original sources to avoid unnecessary hassle with Xilinx Tools. Upon receiving the latest repo, one should proceed to `<tree>/ssd_repo/ssd_Chipscope` and copy the directory over to a location on the local disk where the file path does not have space characters. This is very important, as Xilinx will refuse to open the xmp file if the project directory is on a path with spaces in the path name. Once we have the directory in a valid path, open system.xmp which should bring us to Figure 9, the home page of XPS.

While other functionalities of XPS exist, we primarily use XPS for two purposes, compiling the firmware code and synthesis and downloading of bitstream onto the FPGA. To see the firmware loaded in the current project click the Application tab at the lower left



Xilinx Platform Studio - E:\SSD_Chipscope\system.xmp - [System Assembly View]

Project

Platform

- Project Files
  - MHS File: system.mhs
  - MSS File: system.mss
  - UCF File: data/system.ucf
  - iMPACT Command File: etc/download.cmd
  - Implementation Options File: etc/fast_runtime.opt
  - Bitgen Options File: etc/bitgen.ut
- Project Options
  - Device: xc5vlx110tff1136-1
  - Netlist: TopLevel
  - Implementation: XPS (Xflow)
  - HDL: VHDL
  - Sim Model: BEHAVIORAL
- Design Summary

Name	Bus Name	IP Type	IP Version	IP Classification
microblaze_0		microblaze	7.20.d	Processor
dlmb		lmb_v10	1.00.a	LMB Bus
ilmb		lmb_v10	1.00.a	LMB Bus
mb_plb		plb_v46	1.04.a	PLBV46 Bus
dlmb_cntlr		lmb_bram_if...	2.10.b	Memory Controller
ilmb_cntlr		lmb_bram_if...	2.10.b	Memory Controller
DDR2_SDRAM		mpmc	4.03.a	Memory Controller
lmb_bram		bram_block	1.00.a	Memory
aes_decrypt...		aes_decrypt...	1.00.a	Peripheral
aes_encrypt_0		aes_encrypt	1.00.a	Peripheral
chipscope_pl...		chipscope_pl...	1.03.a	Peripheral
debug_module		mdm	1.00.g	Debug
nand_plb_to...		nand_plb_to...	3.00.a	Peripheral
SPLB	mb_plb			
pcie_ram_0		pcie_ram	2.00.a	Peripheral
IIC_EEPROM		xps_iic	2.02.a	Peripheral
xps_intc_0		xps_intc	1.00.a	Interrupt Controller
SysACE_Com...		xps_sysace	1.01.a	Peripheral
xps_timer_1		xps_timer	1.01.b	Peripheral
RS232_Uart_1		xps_uartlite	1.01.a	Peripheral
chipscope_ic...		chipscope_icon	1.04.a	IP
chipscope_ila...		chipscope_ila	1.03.a	IP
clock_genera...		clock_gener...	2.01.a	IP
proc_sys_res...		proc_sys_re...	2.00.a	Peripheral

Bus Interface Filters

- By Connection
  - ✓ Connected
  - ✓ Unconnected
- By Bus Standard
  - ✓ LMB
  - ✓ PLBV46
  - ✓ FSL
  - ✓ Xilinx Point To Point
- By Interface Type
  - ✓ Slaves
  - ✓ Masters
  - ✓ Master Slaves
  - ✓ Monitors
  - ✓ Targets
  - ✓ Initiators

Figure 9: Home Page of XPS upon opening the system.xmp file

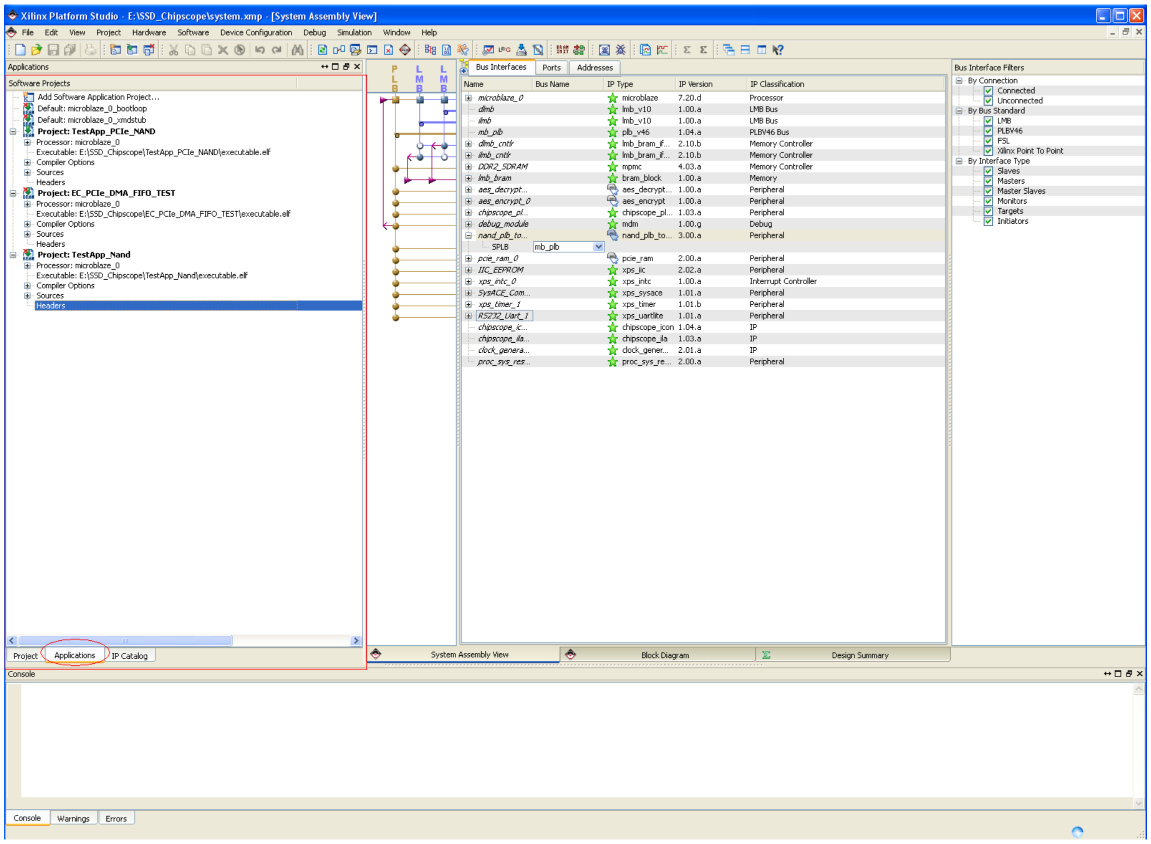
corner. As shown in Figure 10.

In this view, we can see the list of applications Figure 11 shows the expanded view of the Firmware applications. We can see the path to the source code for each of the firmware here. (Also very important, because Xilinx editor tends to play tricks on files and make them full of compile errors).

At this point, the host machine should be off, the FPGA should be on, and the Serial port monitoring the COM1 data port should be on. Find the update bitstream button and click it. See Figure 12

Note, this step is known more commonly as synthesis. Changes made to the memory controller will usually take about 40 minutes to an hour to finish updating the bitstream. When the bitstream finishes updating the console should display text as shown in Figure 13, at which point we can click the Download bitstream button to initiate the XPS tool to program our board, the button is shown in the same figure.

When the bitstream is successfully downloaded, we can proceed to power on the host


Xilinx Platform Studio - E:\SSD_Chipscope\system.xmp - [System Assembly View]
File Edit View Project Hardware Software Device Configuration Debug Simulation Window Help
Applications
Bus Interfaces Ports Addresses
Project Applications IP Catalog
System Assembly View Block Diagram Design Summary
Console
Console Warnings Errors

Software Projects

```
Add Software Application Project...
Default: microblaze_0_bootloop
Default: microblaze_0_xmdstub
Project: TestApp_PCIe_NAND
    Processor: microblaze_0
    Executable: E:\SSD_Chipscope\TestApp_PCIe_NAND\executable.elf
    Compiler Options
    Sources
    Headers
Project: EC_PCIe_DMA_FIFO_TEST
    Processor: microblaze_0
    Executable: E:\SSD_Chipscope\EC_PCIe_DMA_FIFO_TEST\executable.elf
    Compiler Options
    Sources
    Headers
Project: TestApp_Nand
    Processor: microblaze_0
    Executable: E:\SSD_Chipscope\TestApp_Nand\executable.elf
    Compiler Options
    Sources
    Headers
```

Name	Bus Name	IP Type	IP Version	IP Classification
microblaze_0		microblaze	7.20.d	Processor
dlmb		lmb_v10	1.00.a	LMB Bus
ilmb		lmb_v10	1.00.a	LMB Bus
mb_plb		plb_v46	1.04.a	PLBV46 Bus
dlmb_cntlr		lmb_bram_if...	2.10.b	Memory Controller
ilmb_cntlr		lmb_bram_if...	2.10.b	Memory Controller
DDR2_SDRAM		mpmc	4.03.a	Memory Controller
lmb_bram		bram_block	1.00.a	Memory
aes_decrypt...		aes_decrypt...	1.00.a	Peripheral
aes_encrypt_0		aes_encrypt	1.00.a	Peripheral
chipscope_pl...		chipscope_pl...	1.03.a	Peripheral
debug_module		mdm	1.00.g	Debug
nand_plb_to...		nand_plb_to...	3.00.a	Peripheral
SPLB	mb_plb			
pcie_ram_0		pcie_ram	2.00.a	Peripheral
IIC_EEPROM		xps_iic	2.02.a	Peripheral
xps_intc_0		xps_intc	1.00.a	Interrupt Controller
SysACE_Com...		xps_sysace	1.01.a	Peripheral
xps_timer_1		xps_timer	1.01.b	Peripheral
RS232_Uart_1		xps_uartlite	1.01.a	Peripheral
chipscope_ic...		chipscope_icon	1.04.a	IP
chipscope_ila...		chipscope_ila	1.03.a	IP
clock_genera...		clock_gener...	2.01.a	IP
proc_sys_res...		proc_sys_re...	2.00.a	Peripheral

Bus Interface Filters

```
By Connection
    [✓] Connected
    [✓] Unconnected
By Bus Standard
    [✓] LMB
    [✓] PLBV46
    [✓] FSL
    [✓] Xilinx Point To Point
By Interface Type
    [✓] Slaves
    [✓] Masters
    [✓] Master Slaves
    [✓] Monitors
    [✓] Targets
    [✓] Initiators
```

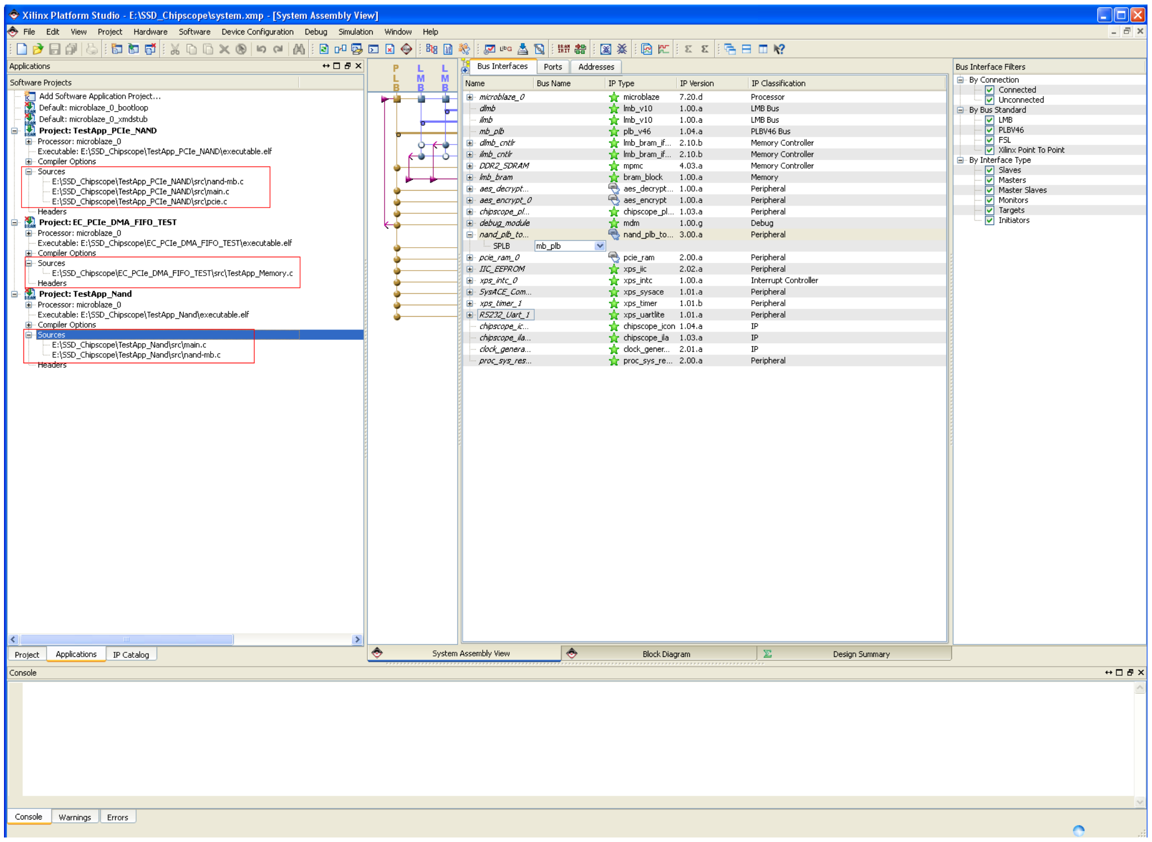
Figure 10: Highlights the Firmware Section in XPS View

machine and perform tests for PCI-E recognition and so on. You should see something similar to the text highlighted in Figure 14.

In order to import the project into an existing project. Given the XPS tool is 11.1, one must first ensure that all the libraries listed on the Figure 9 are present in their distribution of XPS, then check the version number.

Our team attempted to migrate the XPS 11.1 project into 12.1, and the results were not promising due to a few reasons. Certain packages such as MPMC does not remain compatible across different versions of XPS. If we modify their version numbers manually, then they will fail during synthesis. We must import newer versions of each incompatible package to make the transition between XPS tool versions successful.

Another obstable for inter-version importing/exporting of existing XPS projects is the UCF file. Generally the UCF file stays the same for all Virtex 5 boards, however, when a newer tool version deprecates a package version, it in turn also deprecates the connection assigned to that package in the UCF file. When such situations occure, one should


Xilinx Platform Studio - E:\SSD_Chipscope\system.xmp - [System Assembly View]
File Edit View Project Hardware Software Device Configuration Debug Simulation Window Help
Applications
Project Applications IP Catalog
System Assembly View Block Diagram Design Summary
Console
Console Warnings Errors

```
Software Projects
  Add Software Application Project...
  Default: microblaze_0_bootloop
  Default: microblaze_0_xmdstub
⊟ Project: TestApp_PCIe_NAND
    ⊞ Processor: microblaze_0
      Executable: E:\SSD_Chipscope\TestApp_PCIe_NAND\executable.elf
    ⊞ Compiler Options
    ⊟ Sources
        E:\SSD_Chipscope\TestApp_PCIe_NAND\src\nand-mb.c
        E:\SSD_Chipscope\TestApp_PCIe_NAND\src\main.c
        E:\SSD_Chipscope\TestApp_PCIe_NAND\src\pcie.c
      Headers
⊟ Project: EC_PCIe_DMA_FIFO_TEST
    ⊞ Processor: microblaze_0
      Executable: E:\SSD_Chipscope\EC_PCIe_DMA_FIFO_TEST\executable.elf
    ⊞ Compiler Options
    ⊟ Sources
        E:\SSD_Chipscope\EC_PCIe_DMA_FIFO_TEST\src\TestApp_Memory.c
      Headers
⊟ Project: TestApp_Nand
    ⊞ Processor: microblaze_0
      Executable: E:\SSD_Chipscope\TestApp_Nand\executable.elf
    ⊞ Compiler Options
    ⊟ Sources
        E:\SSD_Chipscope\TestApp_Nand\src\main.c
        E:\SSD_Chipscope\TestApp_Nand\src\nand-mb.c
      Headers
```

Bus Interfaces | Ports | Addresses

Name	Bus Name	IP Type	IP Version	IP Classification
microblaze_0		microblaze	7.20.d	Processor
dlmb		lmb_v10	1.00.a	LMB Bus
ilmb		lmb_v10	1.00.a	LMB Bus
mb_plb		plb_v46	1.04.a	PLBV46 Bus
dlmb_cntlr		lmb_bram_if...	2.10.b	Memory Controller
ilmb_cntlr		lmb_bram_if...	2.10.b	Memory Controller
DDR2_SDRAM		mpmc	4.03.a	Memory Controller
lmb_bram		bram_block	1.00.a	Memory
aes_decrypt...		aes_decrypt...	1.00.a	Peripheral
aes_encrypt_0		aes_encrypt	1.00.a	Peripheral
chipscope_pl...		chipscope_pl...	1.03.a	Peripheral
debug_module		mdm	1.00.g	Debug
nand_plb_to...		nand_plb_to...	3.00.a	Peripheral
SPLB	mb_plb			
pcie_ram_0		pcie_ram	2.00.a	Peripheral
IIC_EEPROM		xps_iic	2.02.a	Peripheral
xps_intc_0		xps_intc	1.00.a	Interrupt Controller
SysACE_Com...		xps_sysace	1.01.a	Peripheral
xps_timer_1		xps_timer	1.01.b	Peripheral
RS232_Uart_1		xps_uartlite	1.01.a	Peripheral
chipscope_ic...		chipscope_icon	1.04.a	IP
chipscope_ila...		chipscope_ila	1.03.a	IP
clock_genera...		clock_gener...	2.01.a	IP
proc_sys_res...		proc_sys_re...	2.00.a	Peripheral

```
Bus Interface Filters
⊟ By Connection
    ✓ Connected
    ✓ Unconnected
⊟ By Bus Standard
    ✓ LMB
    ✓ PLBV46
    ✓ FSL
    ✓ Xilinx Point To Point
⊟ By Interface Type
    ✓ Slaves
    ✓ Masters
    ✓ Master Slaves
    ✓ Monitors
    ✓ Targets
    ✓ Initiators
```

Figure 11: Highlights the Expanded Firmware Section in XPS View

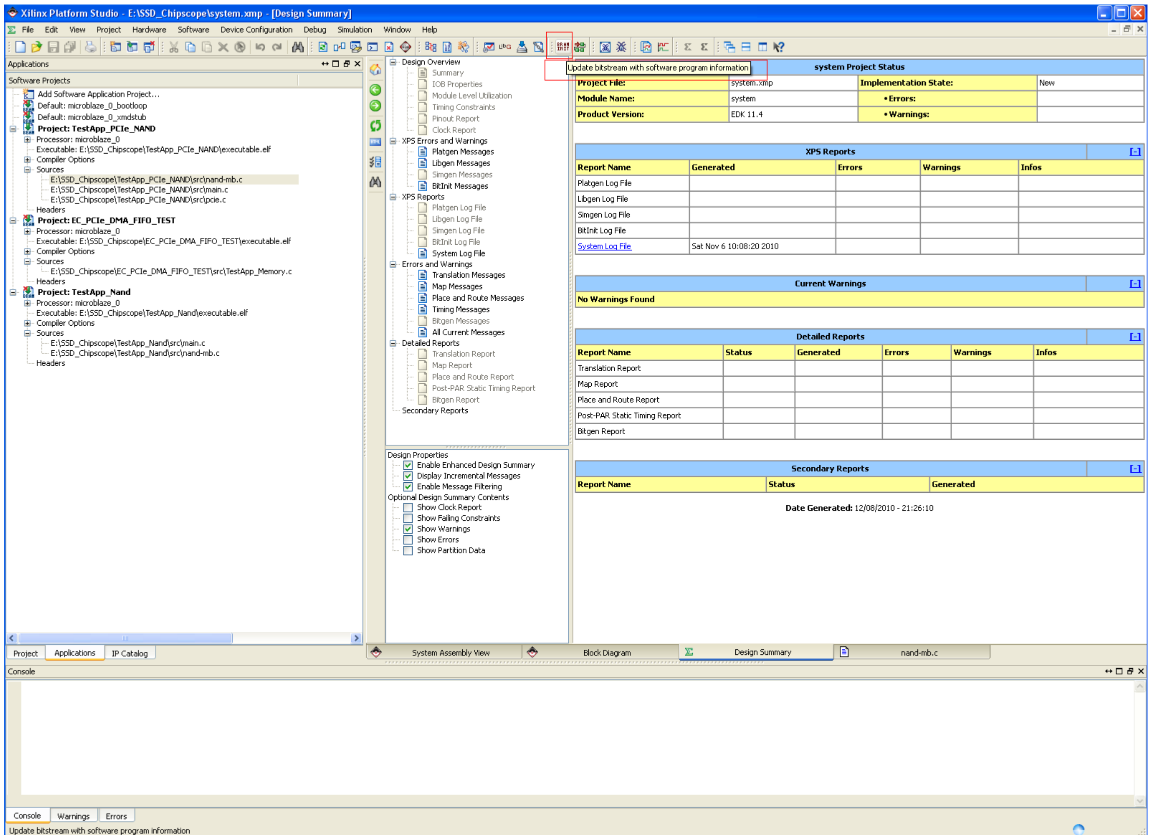
consolidate the Xilinx Forum for solutions. Many hacks exist on Xilinx forms dealing with exactly such situations.

3 System Design

3.1 Memory Controller

The memory controller has gone through multiple different revisions, after consolidating changes from previous project groups, we settled on the following design, see Figure 15: Below is a description of each of the FSM stages:

1. stage 1: WAIT_FOR_CMD This stage just waits for command to be pushed in. WAIT_FOR_CMD stage detected which command was pushed in and set several values for next stages in fsm. For example, it runs case tests of 8 bits in command register(nand_cmd_reg) to see which command is pushed in. Next step is setting



Update bitstream with software program information

system Project Status			
Project File:	system.xmp	**Implementation State:**	New
Module Name:	system	**• Errors:**	
Product Version:	EDK 11.4	**• Warnings:**	

XPS Reports				
Report Name	**Generated**	**Errors**	**Warnings**	**Infos**
Platgen Log File				
Libgen Log File				
Simgen Log File				
BitInit Log File				
System Log File	Sat Nov 6 10:08:20 2010			

Current Warnings
No Warnings Found

Detailed Reports					
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Translation Report					
Map Report					
Place and Route Report					
Post-PAR Static Timing Report					
Bitgen Report					

Secondary Reports		
Report Name	**Status**	**Generated**

Date Generated: 12/08/2010 - 21:26:10

Update bitstream with software program information

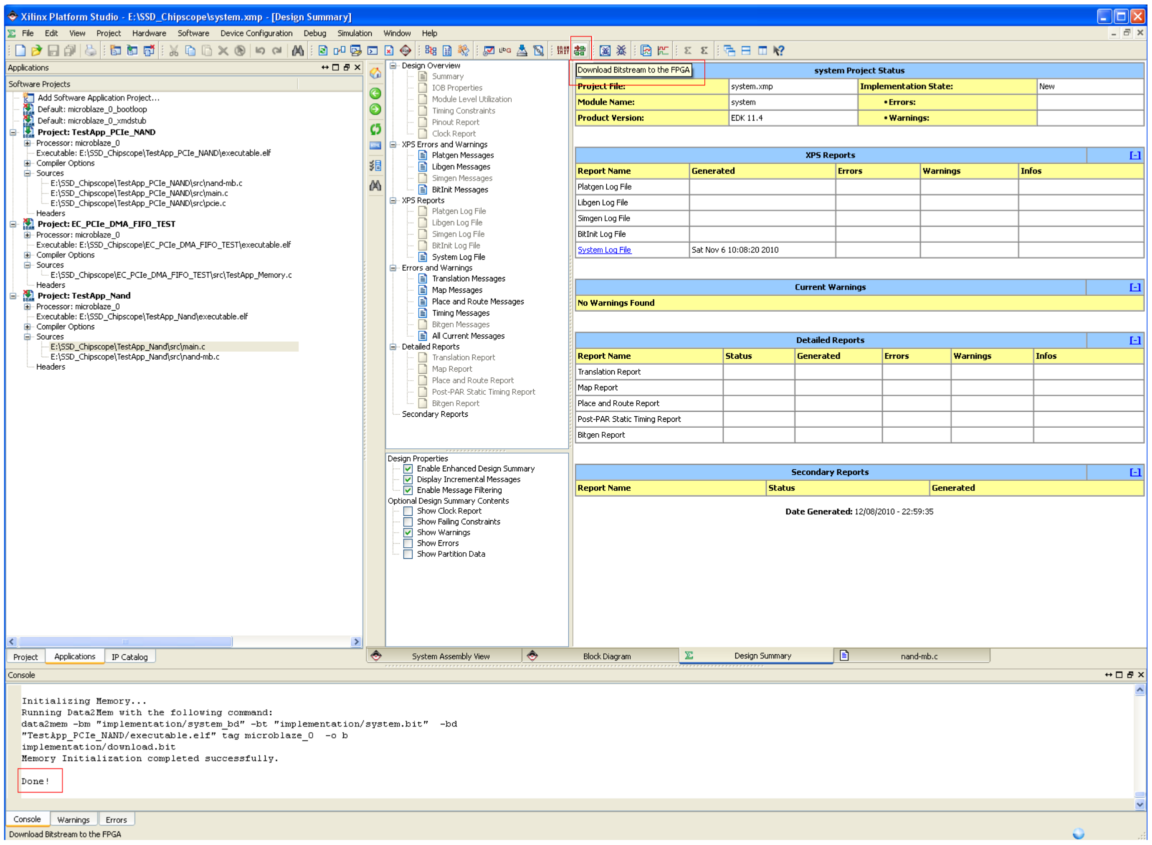
Figure 12: Highlights the Update Bitstream Button in XPS

correct value for flags. In our current fsm design, we have 5 flags to set up in WAIT_FOR_CMD command; num_addr_cycles, num_read_cycles, fsm_timer_nand_addr_delay, cmd_reads_data, and needs_second_cmd.

---

num_addr_cycles - number of cycles needed to latch address for further operation.
num_read_cycles - number of cycles needed to be run when reading from flash memory.
fsm_timer_nand_addr_delay - to synchronize with clock in hardware.
cmd_reads_data - 1 if current command need to read from flash memory, else 0.
needs_second_cmd - 1 if current command demands second command after addresses are latched, else 0.

After setting up right values for flag. from WAIT_FOR_CMD stage, we designed it to



Xilinx Platform Studio - E:\SSD_Chipscope\system.xmp - [Design Summary]

Download Bitstream to the FPGA

system Project Status			
Project File:	system.xmp	**Implementation State:**	New
Module Name:	system	**• Errors:**	
Product Version:	EDK 11.4	**• Warnings:**	

XPS Reports				
Report Name	**Generated**	**Errors**	**Warnings**	**Infos**
Platgen Log File				
Libgen Log File				
Simgen Log File				
BitInit Log File				
System Log File	Sat Nov 6 10:08:20 2010			

Current Warnings
No Warnings Found

Detailed Reports					
Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Translation Report					
Map Report					
Place and Route Report					
Post-PAR Static Timing Report					
Bitgen Report					

Secondary Reports		
Report Name	**Status**	**Generated**

Date Generated: 12/08/2010 - 22:59:35

Console

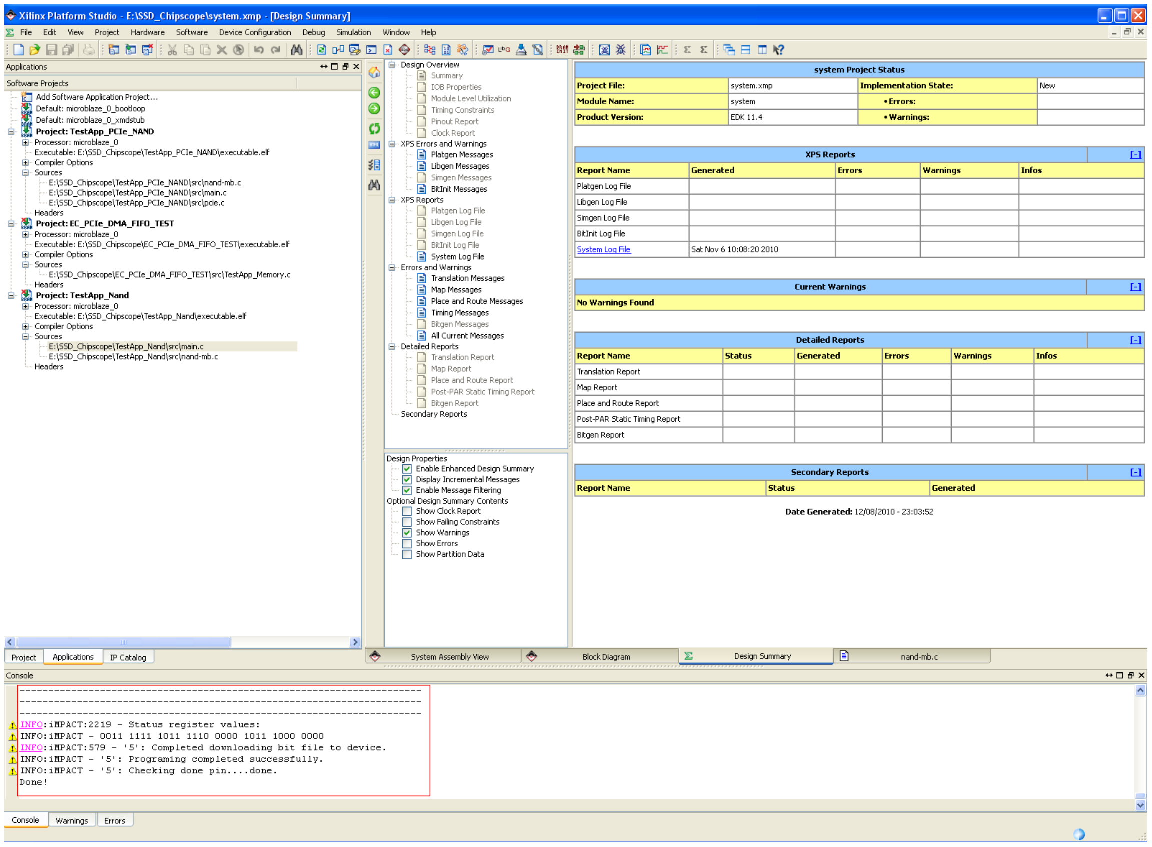
```
Initializing Memory...
Running Data2Mem with the following command:
data2mem -bm "implementation/system_bd" -bt "implementation/system.bit"  -bd
"TestApp_PCIe_NAND/executable.elf" tag microblaze_0  -o b
implementation/download.bit
Memory Initialization completed successfully.

Done!
```

Download Bitstream to the FPGA

Figure 13: Highlights the Console indicating bitstream updated successfully

go to CMD_LATCH stage automatically because first step for every single commands is latching command. However, there is a special case that does not need address latch cycle, reset. Reset command does not need an address latching, fsm goes straight to STATE_REST stage.

2. stage 2: CMD_LATCH For six command that we were supposed to implement, they share same wave from for latching command in. Thus, we decided to pull command latching step out of every command and make separate stage just for it. CMD_LATCH stage is consist of 2 stages. First stage is to open signal for command to be latched in and processed. After System finished CMD_LATCH_1, then it automatically goes to CMD_LATCH_2. There are only one difference between CMD_LATCH_1 and CMD_LATCH_2, state output. Since in first stage of command latch, command was latched in and it is time for step 2, closing gate for command. From second step of command latching, options for next state is split into 2 ways, READ_DATA and ADDR_LATCH. If current command is read status(in our design it


Xilinx Platform Studio - E:\SSD_Chipscope\system.xmp - [Design Summary]
File Edit View Project Hardware Software Device Configuration Debug Simulation Window Help
Applications
Project Applications IP Catalog
System Assembly View Block Diagram Design Summary nand-mb.c
Console Warnings Errors

```
Software Projects
  Add Software Application Project...
  Default: microblaze_0_bootloop
  Default: microblaze_0_xmdstub
  Project: TestApp_PCIe_NAND
    Processor: microblaze_0
    Executable: E:\SSD_Chipscope\TestApp_PCIe_NAND\executable.elf
    Compiler Options
    Sources
      E:\SSD_Chipscope\TestApp_PCIe_NAND\src\nand-mb.c
      E:\SSD_Chipscope\TestApp_PCIe_NAND\src\main.c
      E:\SSD_Chipscope\TestApp_PCIe_NAND\src\pcie.c
    Headers
  Project: EC_PCIe_DMA_FIFO_TEST
    Processor: microblaze_0
    Executable: E:\SSD_Chipscope\EC_PCIe_DMA_FIFO_TEST\executable.elf
    Compiler Options
    Sources
      E:\SSD_Chipscope\EC_PCIe_DMA_FIFO_TEST\src\TestApp_Memory.c
    Headers
  Project: TestApp_Nand
    Processor: microblaze_0
    Executable: E:\SSD_Chipscope\TestApp_Nand\executable.elf
    Compiler Options
    Sources
      E:\SSD_Chipscope\TestApp_Nand\src\main.c
      E:\SSD_Chipscope\TestApp_Nand\src\nand-mb.c
    Headers
```

```
Design Overview
  Summary
  IOB Properties
  Module Level Utilization
  Timing Constraints
  Pinout Report
  Clock Report
XPS Errors and Warnings
  Platgen Messages
  Libgen Messages
  Simgen Messages
  BitInit Messages
XPS Reports
  Platgen Log File
  Libgen Log File
  Simgen Log File
  BitInit Log File
  System Log File
Errors and Warnings
  Translation Messages
  Map Messages
  Place and Route Messages
  Timing Messages
  Bitgen Messages
  All Current Messages
Detailed Reports
  Translation Report
  Map Report
  Place and Route Report
  Post-PAR Static Timing Report
  Bitgen Report
Secondary Reports
```

Design Properties
- [✓] Enable Enhanced Design Summary
- [✓] Display Incremental Messages
- [✓] Enable Message Filtering

Optional Design Summary Contents
- [ ] Show Clock Report
- [ ] Show Failing Constraints
- [✓] Show Warnings
- [ ] Show Errors
- [ ] Show Partition Data

system Project Status			
Project File:	system.xmp	**Implementation State:**	New
Module Name:	system	**• Errors:**	
Product Version:	EDK 11.4	**• Warnings:**	

XPS Reports [-]

Report Name	Generated	Errors	Warnings	Infos
Platgen Log File				
Libgen Log File				
Simgen Log File				
BitInit Log File				
System Log File	Sat Nov 6 10:08:20 2010			

Current Warnings [-]

No Warnings Found

Detailed Reports [-]

Report Name	Status	Generated	Errors	Warnings	Infos
Translation Report					
Map Report					
Place and Route Report					
Post-PAR Static Timing Report					
Bitgen Report					

Secondary Reports [-]

Report Name	Status	Generated

Date Generated: 12/08/2010 - 23:03:52

Console

```
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
INFO:iMPACT:2219 - Status register values:
INFO:iMPACT - 0011 1111 1011 1110 0000 1011 1000 0000
INFO:iMPACT:579 - '5': Completed downloading bit file to device.
INFO:iMPACT - '5': Programing completed successfully.
INFO:iMPACT - '5': Checking done pin....done.
Done!
```

Figure 14: Highlights the Console indicating bitstream downloaded to board

is defined as CMD_READ_STATUS), then next state of fsm would be READ_DATA because it does not need an address to operate. Read status already knows where to read from. In any other cases, next state is ADDR_LATCH because other than read status operation, command needs an address to be executed. While system is in this stage, nand status register(nand_status_reg) indicates that nand chip is busy.

3. stage 3: ADDR_LATCH Just like we pull out command latching steps out from every command waveforms, we decided to do same for address latching steps for same reason. ADDR_LATCH is consist of two steps. First step is to set up right values to open gate for address to latched in. However, we are erasing command register unless current command demands second command later. To have more than one address to be latched in, we decided to loop this whole latching address stages as many times as we need according to num_addr_cycles register. num_addr_cycles register was assigned to certain value in WAIT_FOR_CMD stage. 1 complete loop from ADDR__LATCH_1 to ADDR_LATCH_2 would be one cycle for address latching. After completing enough

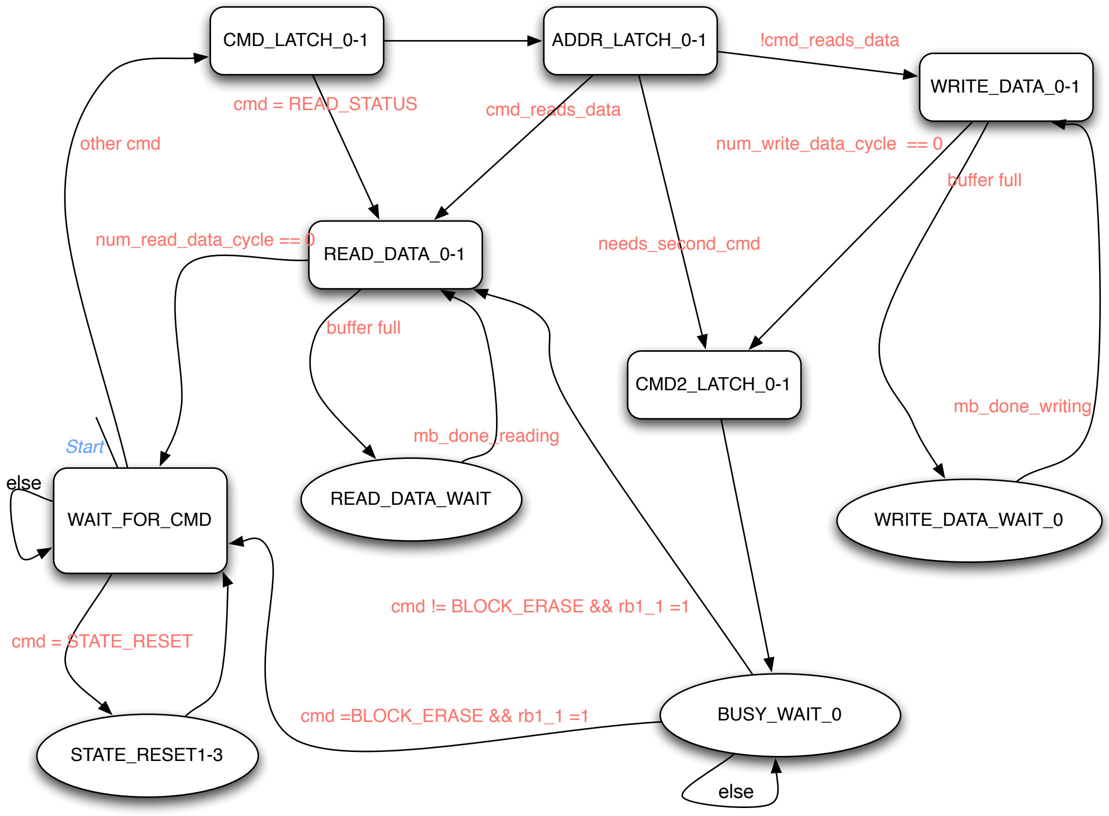


Figure 15: Memory Controller FSM

number of cycles for address, from second steps of latching address, system goes into 3 different stages under certain conditions. If current command needs second command to be completed, then system goes into CMD2_LATCH stages or it goes to READ_DATA if command reads data from nand chip or WRITE_DATA if it needs to write. While system is in this stage, nand status register is showing NAND_BUSY.

4. stage 4: CMD2_LATCH There are few operations that need second command after latching address in to proceed further. However, not every commands need second command, we decided to pull this stage out so that only those that need second command would go through this stage. This stages outputs certain value for second command on io_O wire so that it can be transferred. After putting second command in, we need wait for certain amount of time for requested operation to be finished. While processing nand flash memory chip, system waits for rb_1 signal to be set in BUSY_WAIT stage.

5. stage 5: BUSY_WAIT We have to guarantee enough time for process to read or write on flash chip. To guarantee such time, system is sitting in BUSY_WAIT stage doing nothing. We clear nand command register in here, too. When reading or writing is finished, hardware automatically sets rb1_1 signal as 1 indicating it is safe to proceed to next stage in sfm. When rb1_1 is set, then next stage is READ_DATA unless current command was block erase operation. When it was block erase, then block erase operation is completed done, going back to WAIT_FOR_CMD stage.

6. stage 6: READ_DATA This stage transfers data from flash chip to us. READ_DATA stage reads data from nand flash chip memory and put in buffer. Fsm loops through both step 1 and 2 of READ_DATA until number of read cycle is reached. We set correct value for num_read_data_cycles in WAIT_FOR_CMD stage in the beginning of the fsm. Value of num_read_data_cycles register is decremented every time when buffer is full. data_buffer_counter register is for system to know that how many bytes it has been reading from nand chip. We see buffer is full by multiplying 8 to data_buffer_counter and see if it is same as DATA_BUFFER_SIZE (128bytes for our specification) -1. When buffer is full, no more data can be read from flash chip. Data on buffer has to go somewhere, probably back to host PC in this case. To free the buffer, fsm goes into READ_DATA_WAIT stage. num_read_data_cycle register is decremented after each reading cycle and data_buffer_counter register is incremented at the same time where no certain condition rose. when num_read_data_cycles reaches zero, fsm goes to WAIT_FOR_CMD since all the operations has been completed for read command.

7. stage 7:READ_DATA_WAIT Fsm gets this stage when buffer is full during reading data from chip. Thus, this stage wait till motherboard reads all the data on the buffer and free them. When motherboard finished reading data, it signals by setting mb_done_reading signal. As a result, in this stage, system is looking for mb_done_reading signal to be 1. When it sees mb_done_reading signal, fms goes back to READ_DATA stage.

8. stage 8:WRITE_DATA WRITE_DATA is almost exact opposite of READ_DATA. It writes onto flash chip with buffer. WRITE_DATA stage keeps looping until either buffer is full or num_write_data_cycle reaches zero. Similarly to READ_DATA, num_write_data_cycles was assigned to correct value for writing cycle. When num_write_data_cycle reaches zero, fsm enters CMD2_LATCH because program page command needs a second command to be completed. Checking method to see buffer is full is same as used in READ_DATA. When buffer is full, fsm waits till motherboard finishes writing. When motherboard finishes writing, fsm goes back to WRITE_DATA stage and complete the cycle. Since program page needs a second command as well, after finishing writing cycles, fsm enters CMD2_LATCH stage.

9. stage 9: STATE_RESET Reset stage consists of 3 steps. After finishing all three steps, fsm goes back to WAIT_FOR_CMD stage because there is nothing more to be done for reset operation.

Through these listed stages we are essentially just emulating the following waveforms:

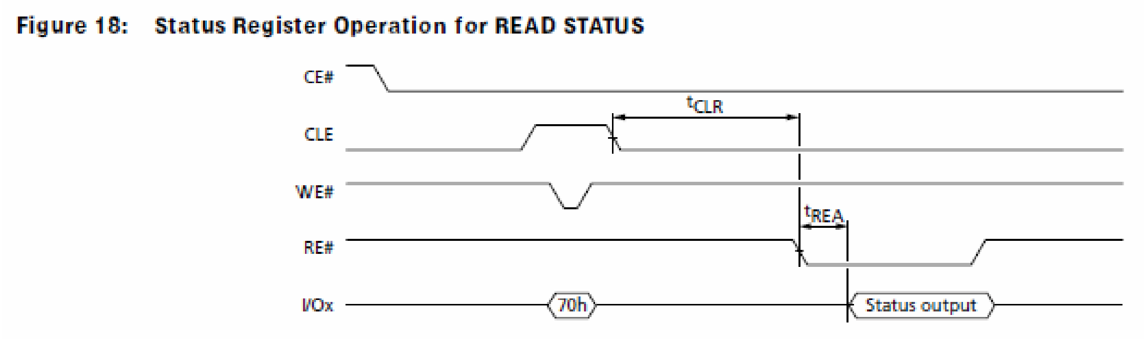


Figure 16: READ STATUS operation

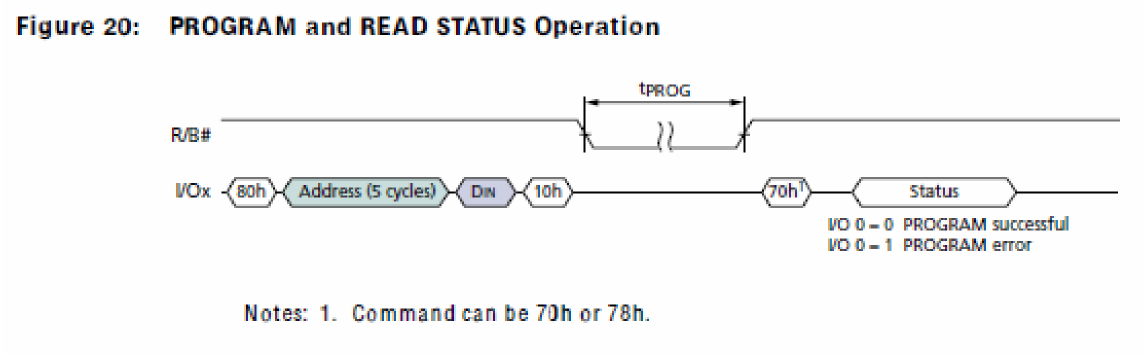


Figure 20: PROGRAM and READ STATUS Operation
tPROG
R/B#
I/Ox
80h
Address (5 cycles)
DIN
10h
70h1
Status
I/O 0 = 0 PROGRAM successful
I/O 0 = 1 PROGRAM error
Notes: 1. Command can be 70h or 78h.

Figure 17: PAGE WRITE operation

Note that the waveforms specified here must be followed exactly in terms of timing, otherwise their behavior is undefined, and almost always wrong. The timing is constrained by using 33 Mhz clock frequency which fits the Micron specifications, and thus the cycles of the FSM can fit the break points in the waveform nicely.

4 Project Environments and Possible Variants

For this project, we used a host machine with Linux Kernel 2.6.28.1, other kernels may not be used at this point, because newer kernels have a new way of handling the module

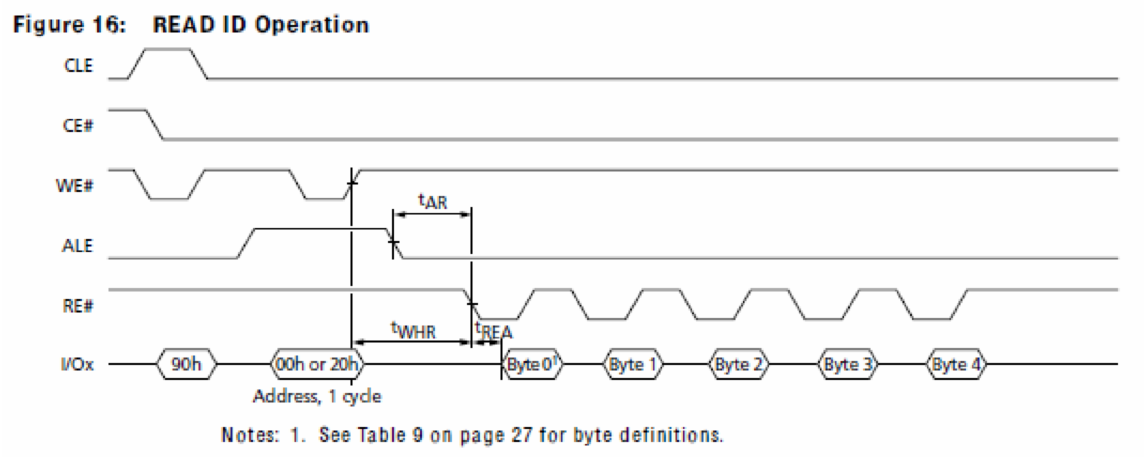


Figure 18: READ ID operation

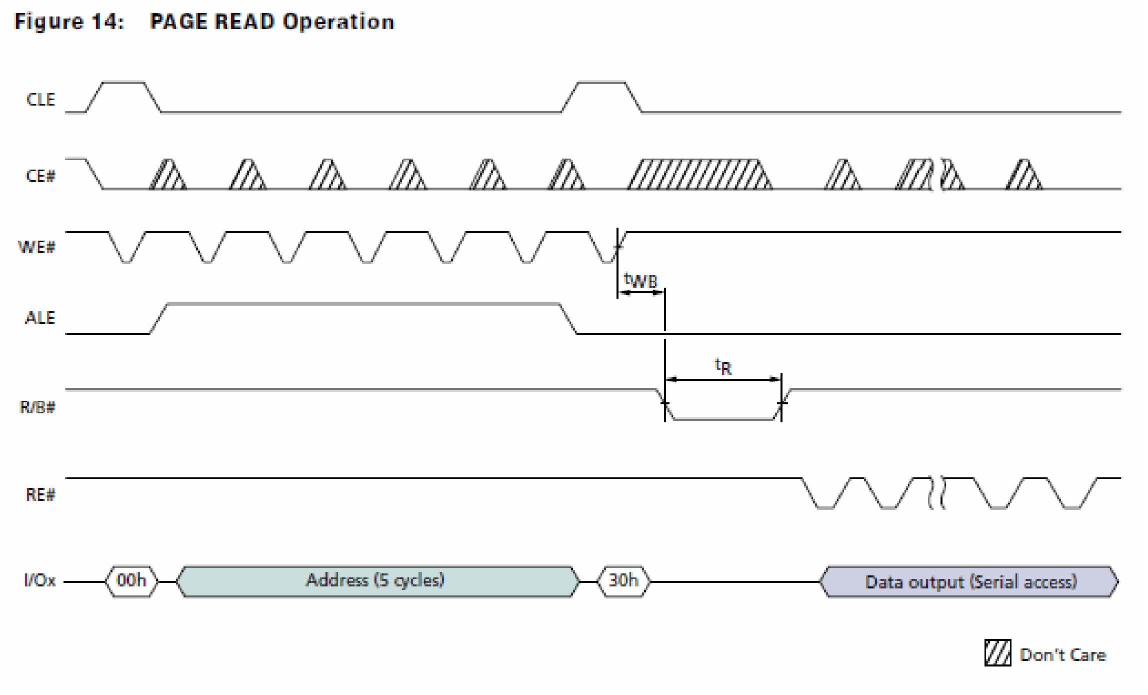


Figure 19: PAGE READ operation

insertion, which does not seem to be compatible with our current setup. We tested and verified this restriction on 2.6.34.7 and 2.6.32.26.

In this project we used a 4 lane PCI-E slot to host our XUPv5 board. During the

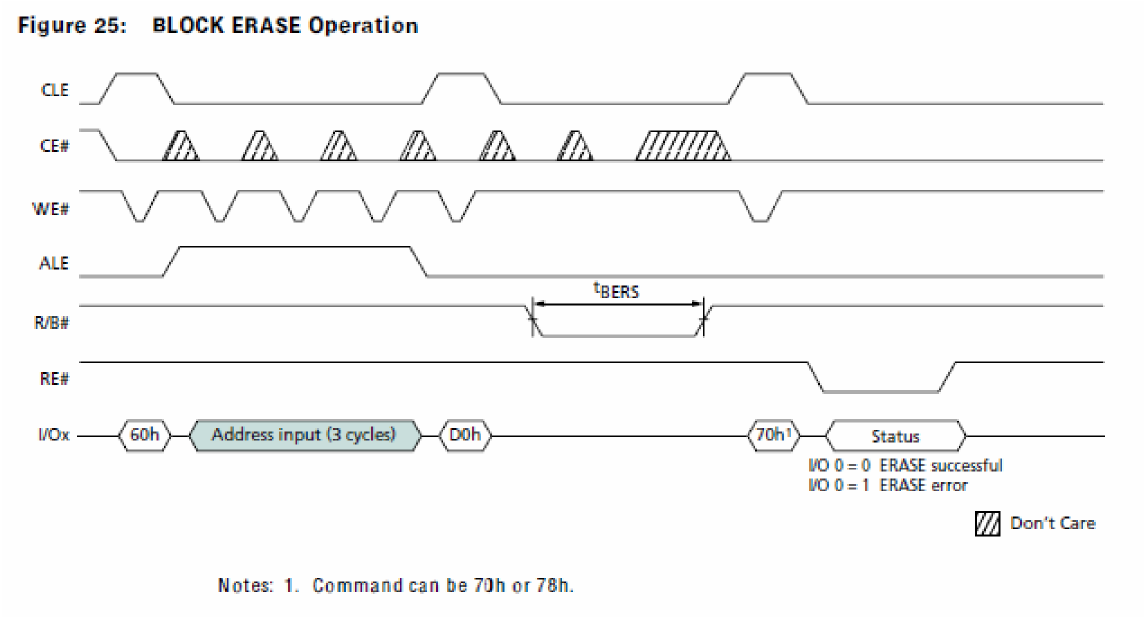


Figure 20: BLOCK ERASE operation

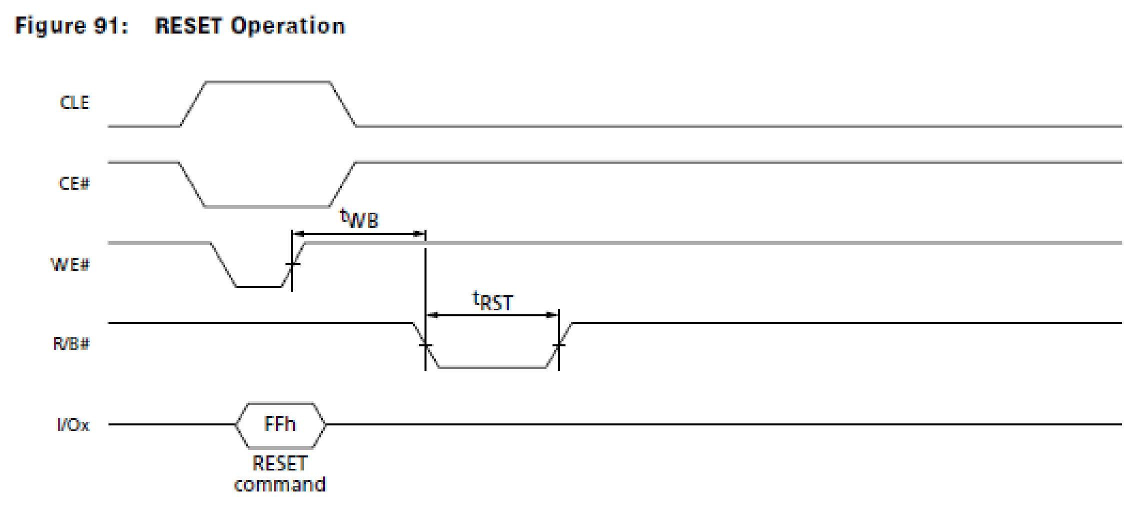


Figure 21: RESET operation

course of this project we discovered that other lane sizes may also be used. Our setup was tested separately on a 8 lane and a 16 lane sized PCI-E slot, and were able to send and receive to and from PCI-E bus with no errors.

5 Shortcomings and Future Work

Due to one of our teammates dropping the course, our team steered toward coming up with good documentation for the overall project instead of focusing on finishing the SSD implementation. As a result of this change, we were able to throughly document the PCI-E interface portions of our project in our source code. We placed a lot of focus on helping future teams who decide to take on this project. We hope this report can serve as a useful frame of reference in future development of this project.

The current SSD implementation lacks the complete set of six commands to make up a fully functional SSD. So far read_id, reset, block_erase, fetch_error appear to be working through our Chipscope analysis. The missing commands are page_read and page_write, both of which are multicycle commands that require synchronization between the firmware and the memory controller. The anticipated clock cycle synchronization appears to have bugs, and NAND chip 2 proves to be simply unusable.

Once the six fundamental commands are working and passes the listed benchmarks in Appendix A, a block device driver should be written to facilitate the abstraction between the OS and application layer, so that the SSD could be treated as a normal disk drive.

6 Individual Efforts

This section lists individual contribution to the project.

6.1 Kun Qian

1. Worked on researching and identifying the progress of previous groups, a seemingly simple task, but in reality took 3 weeks amidst confusion of repositories.
2. Maintainer of the newest repository.
3. Creator and maintainer of the team gantt chart.
4. Setup the initial SysACE demo application for presentation through VGA (about 1 hour).
5. Worked on the chipscope anaylsis tool for the memory controller. (about 2 hours)
6. Setup the Serial port connection to debug Mem controller firmware. (about 2 hours)
7. Worked on the Memory Controller FSM debugging. (3 weeks)
8. Worked on the Linux driver to add enhancements for block erase, page read command and page write command. (about 40 hours in 2 weeks)

9. Worked on the firmware application TestApp_PCIe_NAND, providing enhancements for block erase, page read and page write command. (about 20 hours in 2 weeks)
10. Developed the Design Review Demo application, which was a benchmark application showing activity across the PCI-E bus. (1 hour)
11. Developed the Status Demos, which demonstrated the functionality of the PCI-E driver. (about 2 hours)
12. Created presentation slides for all presentations (around 6 hours total in semester)
13. Documented developement of PCI-E (6 hours in 2 days)
14. Meetings with Yu Cai for help on Project. (6 hours in semester, avg 1 hour per meeting)
15. Regular meeting with Professor Kenneth Mai (6 hours in semester, avg 30 mins per meeting)
16. Drafted and formatted this Final Report (32 hours in 4 days)

6.2 Jihoon Kim

I, JiHoon Kim, worked on fsm and firmware mostly. From the beginning of the project, Matt Cheong who dropped in middle of semester, and I decided to work on six commands first. I finished fsm for block erase, reset and page read. To latch multiple addresses sequentially, we had to change basically entire fsm from last year. Last group did not have to worry about sequential address latching because they did not have command that required multiple addresses. However, most of our 6 commands require multiple addresses. After building fsm for commands, I worked on synchronizing with firmware. However, firmware took much more time than I expected, while working on firmware, one of our teammates dropped the course. After talking to professor Nace and Professor ken Mai, direction of our project was changed. It was to have same project on different board, virtex-6. After trying to compile everything with virtex-6, it didnt even compile. I had to change some stuff in system.mhs and system.mss. Virtex-6 and virtex-5 have different hardwares, such as memory microblaze version. We had to look up specification for virtex-6 and changed from .mms and .mhs files. Also, I thought I could go through all the pin assignments in .ucf file for virtex-5 and change assignments line by line. There are pin specifications for both virtex-5 and virtex-6. Thus, it was easy to find pin description for each board. However, I could not find any relationships between any pins in two different boards. If one pin is used in virtex-5 by our program, then corresponding pin in virtex-6 should be used too. However, I could not find corresponding pin for virtex-6 even with pin description. This project was not successful project at all. There are many obstacles that our team had to go through. We did not have enough communication among team members nor with professors. Our project repository changed many times too. Also, taking over project that some one worked on last semester is not an easy job. There must be better documentation for next team to look at and to understand project much more quickly. However, Kun and I decided to stay in class and learned a lot. First, when working in team, communication is much more important than I thought. Our team did not have any problems with each other. We just didnt meet enough times. If we meet more often and talk more, then project would have been in much better shape. Communication among team members is important, but also communicate with others who worked or is currently working on same project is important too. When having hard time understanding someone elses work, it could be much easier if one asks. I did learn many things technically, such as dealing with fpga board. Furthermore, I learned more valuable things, from choosing, scheduling the project, working in team for semester long, and going through problems with team.

7 Acknowledgments

We here by acknowledge the kind help of Yu Cai, Eric Chung, Will Constable, Tao Yang, Matthew Cheung in the development of this project. We thank Professor William Nace and Professor Kenneth Mai for their words of wisdom throughout the course of this project.

References

[1] E. Chung. http://www.ece.cmu.edu/protoflex/doku.php?id=internal:pci_express:pci_express_notes, 2009.

[2] Xilinx. http://china.xilinx.com/support/documentation/ip_documentation/mpmc.pdf, 2009.

[3] Xilinx. http://www.xilinx.com/itp/xilinx9/books/docs/xst/xst.pdf, 2009.

A List of All Benchmarks

The following benchmarks can be used to test the functionality of the PCI-E interface.

1. TestApp_Nand
   Location: `<tree>/ssd_repo/PCIe-MB-NAND-DIMM/TestAPP_Nand`
   There is a software project taken from the rev2 Nand Controller project, called TestApp_Nand. This test is a good place to start to make sure that the hardware is hooked up properly and you get the serial prints. If this test runs, you can see the readID values from each nand chip and compare them to the expected values.

2. EC_PCIe_DMA_FIFO_TEST
   Location: `<tree>/ssd_repo/PCIe-MB-NAND-DIMM/EC_PCIe_DMA_FIFO_TEST`
   it is a good idea to run this test again in the context of this XPS project before proceeding to the more complicated tests, just to verify the hardware is hooked up right.

   One difference is that our custom DIMM is used this time, not the standard 256MB DDR DIMM provided by Xilinx.

   There is an issue with our DDR controller which I have noted but not looked into. Currently, the XPS project is configured to use BRAM instead of DDR for all of the microblaze address space, since using DRAM caused an unexpected amount of slowdown (1 minute per DMA transfer instead of 300uS). There is probably a quick fix in the DRAM clocks or configuration of the DDR controller, but its easier to just disable the system at this point.

3. TestApp_PCIe_NAND
   Location: `<tree>/ssd_repo/PCIe-MB-NAND-DIMM/TestApp_PCIe_NAND`
   This project utilizes the PCIe channel to send specific messages from the host PC to the MB, issuing commands and transferring the associated data. The rev2 nand controller is used and the commands it supports at this point are made accessible to the host PC running the 'nand_pci_driver' project.

   On the XPS side, make sure to mark the 'TestApp_PCIe_NAND' project for BRAM initialization. The NAND+DDR dimm needs to be used, and apart from that the hardware setup for the PCIe project needs to be followed.

   On the host PC, the 'nand_pci_driver' project needs to be built and run. Enter the root directory of nand_pci_driver and run the command 'make'. This will build all the necessary files and then you can run ./nand_pci_driver. This assumes you have already loaded the kernel module (in the 'module' directory). You can modify the main.c file in 'nand_pci_driver' to change what commands are being issued, or change nand.c and nand.h to add more commands.

B Porting Virtex 5 Project to Virtex 6

Importin virtex-5 project onto virtex-6 board is not simple job. Basically, one would have to create a new project for virtext-6. Even with all the source code from virtex-5 proejct, one still has to create a new project for virtext-6. However, it could be much simpler than creating new virtex-6 project when one has access to virtex-5 project because logic and block diagrams would be similar between virtex-5 and virtex-6. Even though, pin assignment in virtex-5 board and virtex-6 board are obviously different. Thus, it is crucial to modify user constraint file(.ucf) to have project working properly on virtex-6 board. It is going to take a lot of time and effort to change every variable or ports from virtex-5 to virtex-6. Even, there is datasheet and pinout spreadsheet for both virtex-5 and virtex-6, it does not explicitly say that which pin virtex-5 is corresponding to which pin in virtex-6. With all pin assignments completed for virtex-5 board, one still have to go through pin assigning process again with virtex-6 board. There is software development tool called PlanAhead from Xilinx that helps you with pin assigning.

C Helpful Xilinx Tools

Project navigator and PlanAhead are two independent software each running under separate system process. There will be no synchronization for data between two separate software. Changing design data from one tool is not automatically recognized by other in real time. One should not attempt to modify constraint simultaneously in both tools. Just like data synchronization in multi threading programming, one should save changed data from one tool to update or see changes on the other software. PlanAhead in ISE Integration mode only enables physical constraint modification for I/O pins, logic LOC and `AREA_GROUP` constraints. Other PlanAhead features such that enabling logic or timing constraint modification are not available in ISE Integration mode, one should use PlanAhead by itself to do so. PlanAhead will try to maintain the origianl content and format of UCF files such as comments and incomplete constraints.