NESBoy

A Handheld, Portable NES

Nick Moore

Joshua Graham

Robert Gazdziak

Fall 2015

ECE 395: Advanced Digital Projects Lab

University of Illinois at Urbana-Champaign

**Terms**

NES- The Nintendo Entertainment System, a video game console released by Nintendo to Japan in 1983 as the Famicom, and the US in 1985 as the NES. There are subtle differences between the Famicom and the NES, including clocking frequencies, pinouts for the cartridges, and security.

ROM- Read-Only Memory file that is a copy of NES game cartridges. ROMS are generally used in software emulators for playing these games but claim that the ROMS are for legal backup purposes, though copying ROMS from semiconductors is explicitly illegal in either case.

PPU- Picture Processing Unit. How the NES wrote to the screen. The CPU would use control registers and direct memory transfers to communicate which information should be displayed. The PPU handled transferring this information to the video memory.

MMC- Memory Management Controller. Each cartridge with a larger address space than the 6502 supports came with its own MMC to manage memory accesses from the CPU to cartridge memory by switching memory banks, providing a primitive “virtual” memory. An effective use of these “mappers” in conjunction with the PPU also improved graphics performance of the NES.

2A03- Short for RP2A03, which is the CPU for the NES, and was manufactured by Ricoh. Its ISA was implemented with a 6502 core that lacked binary coded decimal instructions. It had an Audio Processing Unit, and a direct memory transfer function to reduce latency in copying video data.

2C02- Short for RP2C02, which is the PPU for the NES, and was manufactured by Ricoh.

6502- 8-bit Microprocessor created in 1975 that was much cheaper than its competition, while still having full features. It was based on an accumulator ISA and was used in many famous computer systems, including the Apple II and the Atari 5200.

**Abstract**

Our plan was to emulate an NES using FPGA chips and create a board that could play NES games. We intended to find some code online that people already emulated parts of the NES and edit them for our own usage. Unfortunately, most of the code was poorly written, and used illegal ways of emulating an NES (using a ROM instead of a cartridge), so our project just kept getting larger and larger (as we wanted to use actual cartridges). We ended up being able to split the CPU and PPU up into two different projects so we could have two FPGAs just like the original NES had these two chips separate, but we were only able to start testing on the CPU.

**Table of Contents**

**Overview ………………………………………………………………………………………. 5**

**Code ………………………………………………………………………………………….. 10**

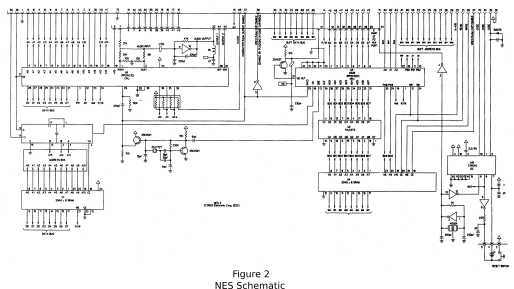
**Splitting up the Project ………………………………………………………………….... 12**

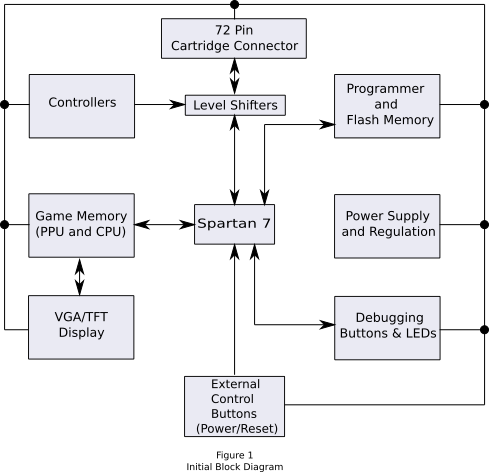
**Issues ….…………………………………………………………………………………….. 14**

**What’s Next …………………………………………………………………………………. 16**

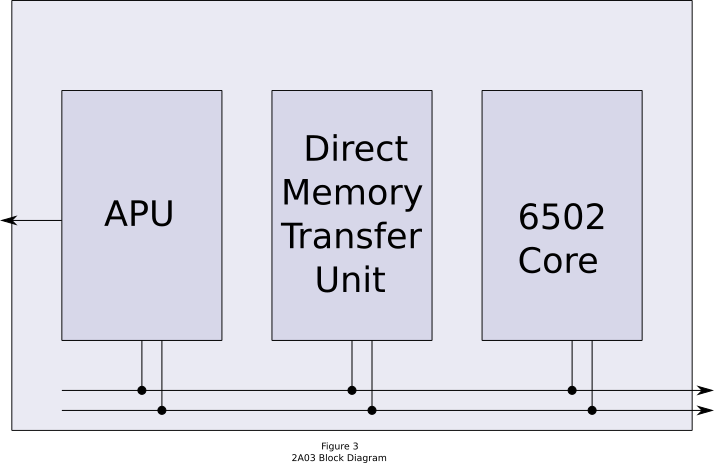
**Works Cited …………………………………………………………………………………. 18**

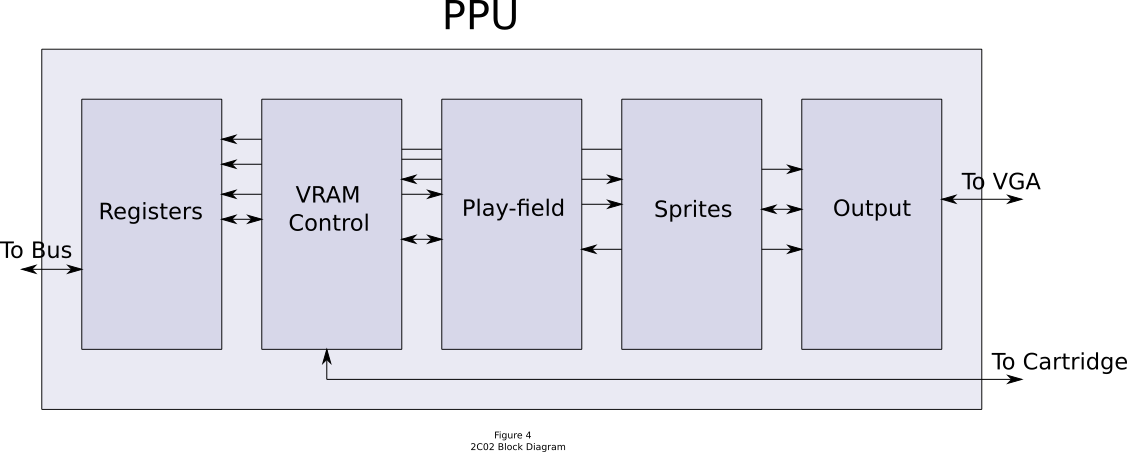
**Overview**

Our project had a fairly straightforward intent. Our original idea to make a portable NES system was intended to focus mainly on designing a board and focusing the bulk of the project on designing the project in EAGLE, quality design decisions, and building experience actually dealing with physical hardware, which meant we were to find an already existing fpga implementation of the NES and modify it. We also wanted a project we could recreate at home and bring a working device to show prospective employers at career fairs. For this outcome, we knew we would have to be able to build a board with modules for power supply and regulation, cartridge communication, controller ports, VGA output, memory for the CPU and PPU, miscellaneous buttons such as power and reset, programming the fpga, any debugging tools we might need, and of course the fpga. Also, because of the 5V standard used for the NES, we knew we would need to level shift the signals between the cartridge and the fpga, and also between controllers and the fpga. An initial block diagram is included below. There were some considerations in choosing the parts for our design. The original NES was built to try to deter people from being able to clone it, so it had several non-standard parts. 

The CPU was the 2A03, only used for the japanese and US versions of the NES. The cartridge reader was also non-standard, having 2.5 mm in each gap instead of 2.54 

mm. Also, each cartridge came with it’s own MMC, not all of which have been reverse-engineered, but if we used the actual cartridges we would not have to worry about having all the MMC’s in our code, even without legal considerations. The size of the fpga takes some serious considerations. We assumed the project is easiest to design only using one fpga, but in order to use cartridges we need at least 60 IO pins to interface with the cartridge, and still more pins for debugging purposes, graphical output, controller interfacing, and control buttons. When using ROMS, there is at least one source1 that claims the project can be completed successfully with a chip as small as a Xilinx Spartan 2, but this source used a development board with aftermarket external modules designed to increase the functionality of the board. We were unable to obtain his files to confirm that his approach would with our restrictions. There are several fpgas available with a suitable number of IO pins, but packaging concerns are also an issue. We were not able to find a single fpga with enough pins that was not in a BGA package, so a design where the CPU and PPU reside in separate chips was eventually agreed upon. The available schematics for the NES can be a good example to learn from in designing how the two chips would interact with the various modules in the project. Note that our original block diagram excludes any modules for sound. We felt that the sound was not necessary for actually playing and developing our project until the project was otherwise functional, so we chose to postpone inclusion of any sound modules until we were sufficiently satisfied with the functionality of the game processing, controller, and video aspects of the project.

The 2A03 has 3 main modules within it. The 6502 core implements the ISA, the APU generates the sound signals which are output to a DAC, and the Direct Memory Access controller is used to accelerate data transfer in certain situations. Because the 6502 is an accumulator architecture, transferring sprites from the cartridge to the PPU would require a large amount of load/store instruction pairs, along with potentially writing address changes to the PPU. The DMA is used to take over this type of data transfer from the cartridge to the PPU by sequentially transferring the data across the bus to the PPU automatically. This requires the 6502 to pause execution during the transfer due to the shared data bus, but significantly reduces the time of the transfer process. The 6502 is an 8-bit ISA, with 16-bit byte-addressable address space2. 

The 2C02 has less helpful documentation available, due to the fact that the 6502 is so well documented, as opposed to the documentation on the 2A03. Communication between the 2A03 and the 2C02 occurs through memory mapped registers. The 2C02 has its own address space, with 16KB of RAM, and another 256 bytes of Sprite-RAM to store sprite attributes. The 2C02 handled all of the video memory and sprite processing. The original NES used an RF adapter when transferring the video televisions, which is too outdated for this project, as it would restrict our video options too severely. The projects we chose already built VGA output drivers for handling the video display. Figure 4 is a block diagram of one implementation of the 2C022. 

For development of our code, we used two different development boards: the Spartan 3 Starter Board which uses the Xilinx Spartan 3 fpga, and the Opal Kelly XEM3001, which uses a larger Spartan 3. The Starter Board is easily programmed from ISE, the Xilinx development software, but the XEM3001 must be programmed separately with Opal Kelly software. The Opal Kelly software is on the computer at the Nesboy bench, which is the corner bench in the northeastern corner of the room, by the TA bench. If you need information about using the XEM3001, there are example projects on the computer, and Casey Smith is very knowledgable about the product, as he used to develop for Opal Kelly.

**Code**

The success of this project depends on using already complete and functional HDL code for at least the 2A03 and the 2C02. There are several projects on [www.github.com](http://www.github.com) which host code for this purpose. Most of these repositories lack any serious documentation for the project, and some of them rely on using proprietary IPs in their design like the Altera NIOS II. Most of the projects did not provide any evidence that their project actually worked, though a few claimed so. We found two projects3,4 with photographic evidence of functionality. There was a third project5 which looked promising, but which explicitly stated that it was still in development, and we found no visual evidence of functionality. Of the two repositories which we felt were promising, one of them was implemented in VHDL4, which we have no experience with, and in this project the 2A03 was implemented inside the 2C02, which meant we would need to possibly make some severe changes to the project with a language we have no experience in. We all agreed this was a recipe for failure. The other project was written in verilog3, which is an awkward language for us in that we were only practically familiar with the most recent standard of SystemVerilog, which offers us a higher abstraction level and simpler type declaration rules for newer data types than verilog offers. We felt that the verilog based project would be an easier project to understand the necessary changes to the project. The verilog code was forked from a Github repository so we could make any changes we felt were necessary. Among the code was a VisualStudio project to provide an interface to use a USB controller connected through a computer. This code also handled transferring ROM files to the fpga over a UART module the original contributor built. The entire project was developed on a development board called the Nexys 3, which was designed and sold by Digilent, and uses a Xilinx Spartan 6 fpga. The development board did not have its own clock or crystal, so the project used the fpga’s 100MHz clock, with a clock-generation wizard in the ISE development environment. The original contributor included what we believe to be all of the necessary files for making the project work on the development board he used, with a significant exception. The Digilent Adept 2 sdk with its included libraries must be downloaded from Digilent and included in the Visual Studio solution files in order to transfer ROM files and controller status words through the computer to the fpga6.

Github is a website which offers free hosted repositories for version management in developing projects. The version management tool used with Github is called Git, and is offered free for Windows, Linux, and Apple systems. We were able to download Github on our bench computer in order to manage our repository without administration privileges, otherwise we would have had to download the code in a zip file and keep all changes on the bench computer. We created our own repository, which is hosted at <http://www.github.com/NESBOY395/fpganes>, and we plan on continuing to contribute to the project after the semester ends.

**Splitting up the Project**

Because we were plannning on using cartridges instead of ROMS, we needed to modify the verilog code to extract a single module that implements the 2A03 completely and a single module that completely implements the 2C02. We were able to separate the 6502 module from the rest of the project, but we discovered that the 2A03 was not implemented as a single module in the project. The DMA controller was implemented separately, as well as the APU. Another surprise in the code is that the 6502 was implemented with separate output and input data signals, instead of a bidirectional data bus. This complicates any attempt at enclosing the necessary components into a 2A03 module, as we would need to multiplex the signals for when we have reads and writes. Any changes we would want to make to build the 2A03 module would be worthless if we are unable to confirm the 6502 module is functional, and we could not confirm the 2C02 is functional without a 2A03, so we attempted to verify the 6502’s functionality. We located a C compiler for the 6502 and built a small counter program which was to modify a specified address that we could output to some LEDs in the Spartan 3 Starter Board. Then we wrote a small translater to convert the bytes in the object file into hex values in ASCII text, and loaded that file into a memory module we modified from ECE411. We were unable to successfully compile the 6502 with this initialized memory, as the compilation stalled during verification of the synthesis. We believe the stall was potentially because of the limited space on the fpga, as compared to how much space the RAM module was using, but we have not been able to confirm our suspicions. This is the current state of the project.

**Issues**

There were several issues which contributed to the failure of our project. First, we made some very incorrect assumptions at the beginning of the semester. These assumptions were that when we found a working project, that project would be usable. This assumption was based on deeper assumptions that people who could make a project like this had the same understanding of proper development practices as we did. In reality, the projects we found demonstrated significant weaknesses in development, ranging from process weaknesses like poor commenting and poor code organization, to conceptual weaknesses like a misunderstanding of different abstraction levels. We feel the project we chose suffered severely from laziness in design, coding, and documentation, which we could only determine after analyzing the source line by line. One of the great benefits we learned from this project is the damage that poorly documented code can do to a project. While we all had learned that proper commenting and self documenting code is a good practice, we did not have a practical example to actually qualify the lesson for us. This project provided that example.

A lack of knowledge on hardware also contributed to our failure. We were not aware of the differences in chip packaging, and how that would affect our project. BGA packaging is the usual package for chips which have sufficient IO pins to implement this project on a single chip. BGA stands for Ball Grid Array, where all of the pins are on the underside of the packaging. This kind of chip needs to be baked on the board. While there are projects on the internet to build ovens to bake these chips, you cannot verify if the pins are soldered correctly without an X-ray machine. Unfortunately, the lab doesn’t have the capabilities to solder and verify BGA packaged chips, so we had to choose qfp packaged chips, in which the pins are on the sides of the chip, where we could access them with a soldering iron. The amount of time it took for us to realize this prevented us from developing an EAGLE project.

Programming the fpga is something that needs to be understood immediately before any work can be done. Most fpgas can be programmed through a JTAG interface, and some of the newer ones can be programmed through UART. Development boards often come with their own programming interface to simplify the process. The Spartan 3 Starter Board that we used is programmed via JTAG, but there are not any USB to JTAG cables. The Spartan 3 Starter Board comes with a parallel cable, so whichever computer being used must have a parallel out. The ECE 395 bench computers do have parallel ports, so you can use them to develop the 2A03 and the 2C02, but for programming projects at home a different solution must be built. The data has to be converted from USB to JTAG by a separate module, which is not usually sold for general applications. Digilent has a cable to do this, which costs $60 USD, which is unreasonable for our application.

**What’s Next?**

We plan on continuing the project to completion. One possible step would be to look at the code from the repository on Github which explicitly stated it is in development. With a second look, it appears that the incomplete portion of the project is mostly the MMCs needed for running the project off of ROMS, and that the 2A03 and the 2C02 are implemented as separate modules. The project’s README file states that the project includes a computer interface to run unit tests on completed modules. Another option is to build our own 2A03. We have 6502 documentation to build a multicycle processor with correct cycle timing specifications, and we have experience building the multicycle LC3b in ECE 411. After we have a functional NES, the next step would be to include a screen and a battery module to provide portability.

For any team interested in pursuing this project, there are some steps we would recommend as a prerequisite. There is a USB port JTAG programmer project7, which interfaces with OpenOCD, for building a programming module with the FTDI FT2232 chip. This would be a very good introduction to EAGLE, while building a tool which enables you to develop your own projects without relying on expensive proprietary tools. After this step, you should then attempt to create your own fpga development board. This project would require you to design circuits to power, program, and interface your board with modules to make the board useful. our proposed project doesn’t seem exciting, but it provides necessary experience with fewer surprises, and the ability to adapt your projects easier.

References

1. Strother, D. (2010, September 27). FPGA NES. Retrieved September 1, 2015, from http://danstrother.com/fpga-nes/
2. Diskin, P. (2004, August 4). NES Documentation. Retrieved August 29, 2015, from http://nesdev.com/NESDoc.pdf
3. fpganes. (2014, January 3). Retrieved September 5, 2015, from <https://github.com/strigeus/fpganes>
4. vhdl-nes. (2012, February 1). Retrieved September 5, 2015, from [https://github.com/chenxiao07/vhdl-nes/](https://github.com/chenxiao07/vhdl-nes/blob/master/READEME)
5. fpga\_nes. (2014, July 14). Retrieved September 5, 2015, from <https://github.com/strigeus/fpganes>
6. Digilent Adept. (2015). Retrieved December 1, 2015, from http://www.digilentinc.com/Products/Detail.cfm?Prod=ADEPT2
7. Tantos, A. (2010, October 1). USB port JTAG programmer. Retrieved October 31, 2015, from http://www.modularcircuits.com/blog/projects/development-tools/usb-port-jtag-programmer/