TEAM DEFENDER

Andrew Brock | Gabriel Garcia | Russell Nelson

# An Apple ][e on a Virtex-5

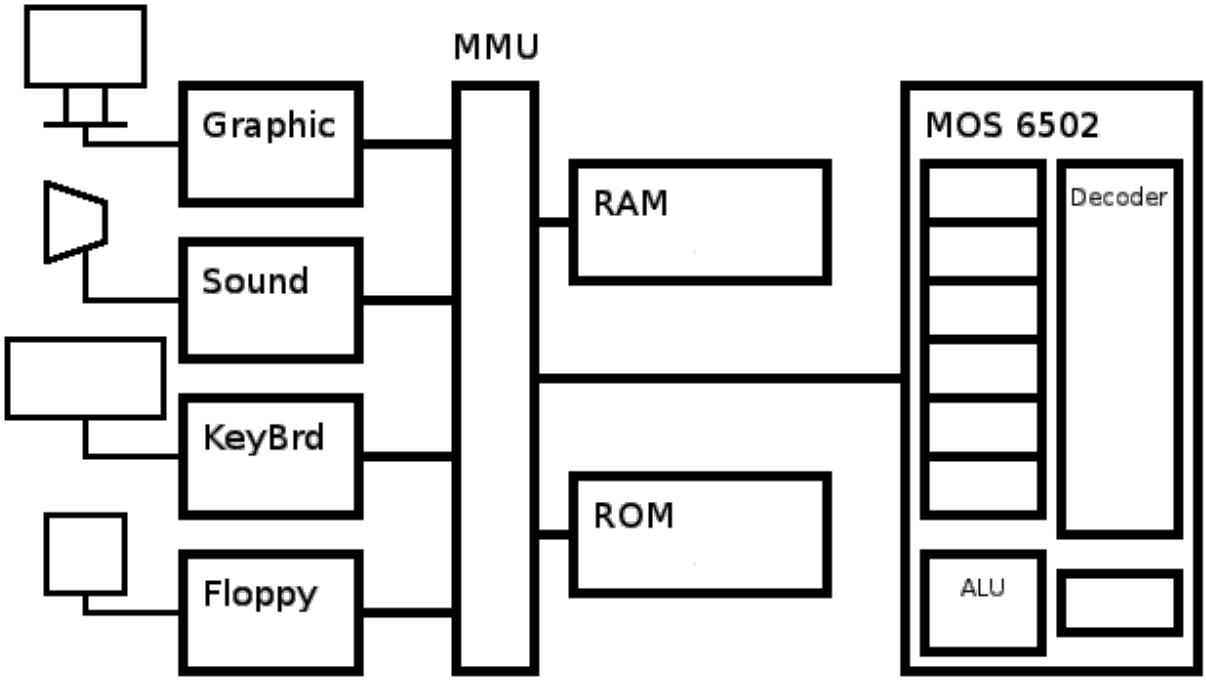
December 13, 2012

# Contents

# 1 Platform

We are using Xilinx's 64-bit ISE and the ML505 evaluation board with a xc5vlx110t Virtex-5 FPGA. This board was chosen because it had been used with great success by many past projects in this course, despite the software stack's lack of SystemVerilog support.

We used a Git repository hosted in AFS for version control. This allowed each of us to have our own versioning system in which to test experimental changes, while still allowing code to be shared without fear of merge conflicts with a single command.



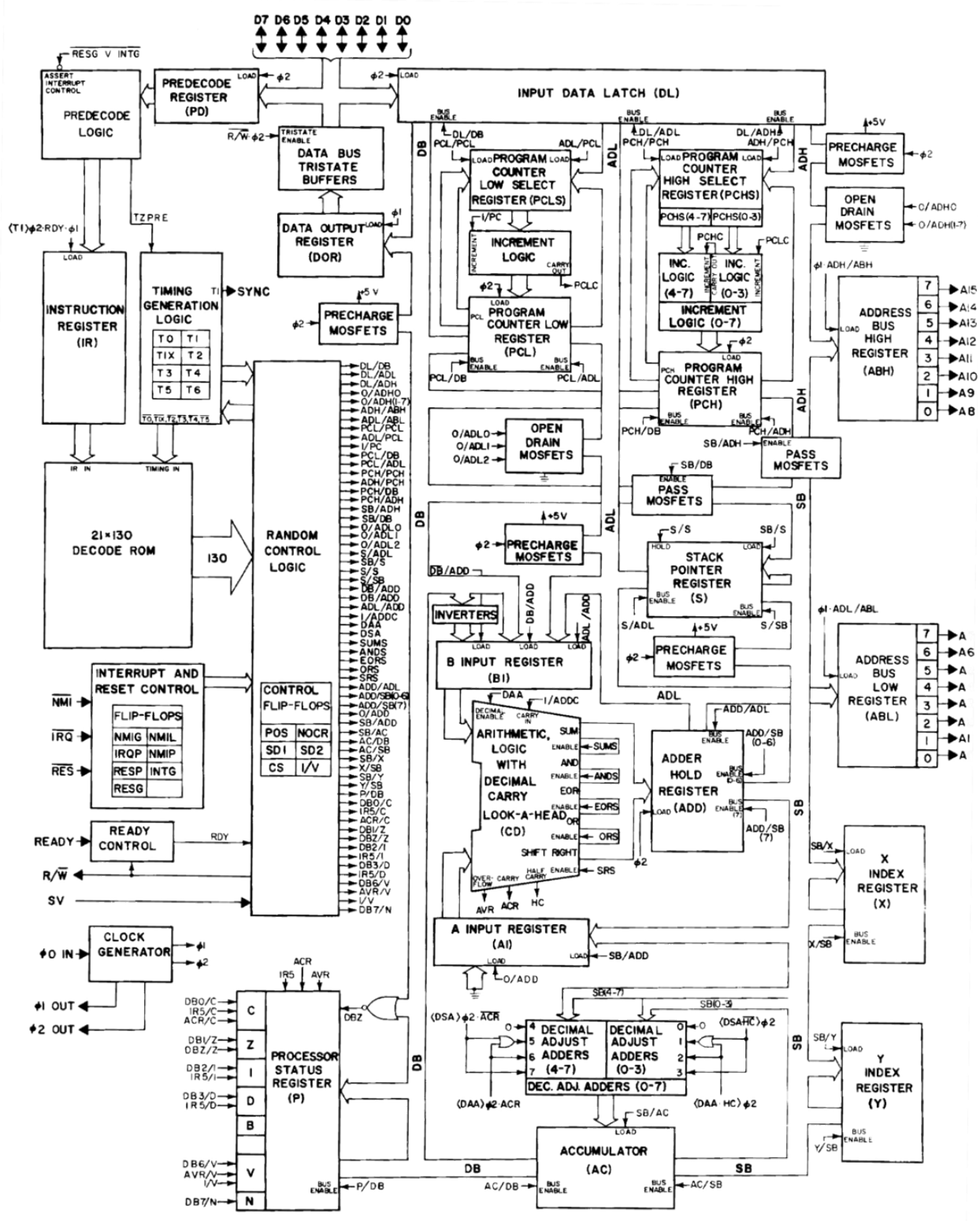

The Apple ][ is centered on its memory bus. We left this basic architecture unchanged, as it is very modular. Having all I/O be memory-mapped allowed us to completely ignore several subsystems: the boot code simply believes that they were not installed.

# 2 Functional Systems

## 2.1 Core

The Apple ][e is based on the 6502 microprocessor. Early versions used the NMOS version, while later versions converted to CMOS. It was an important factor in our design decision to implement the 6502 ourselves that the CMOS variant works: the 65C02 implements an instruction trap, whereas the 65N02 simply executes undefined instructions. This allowed us to avoid implementing the undefined instructions.

The 6502 has 11 addressing modes and 143 instructions. It has two index registers and an accumulator: not all operations can be performed on all combinations of registers.




We attempted to implement a 6502 ourselves, but our implementation schedule was overly optimistic. In late November, we substituted Arlet Ottens' excellent free design. There were no bugs found in the most recent version, and integration took about 3 hours.

In the attempt to implement our own 6502, VCS was very useful for simulation: its

interface is in general much better than ModelSim. We also attempted to use Verilator, which gives very good compile-time error messages, but its inability to handle tri-state busses rendered it completely useless for our system, which is based on a tri-state data bus. The processor would not execute under Verilator.

## 2.2 Memory

Most of our memory is based on block RAM and ROM IP cores. We have custom Perl scripts to convert binary image files that are freely available online to *.hex* and *.coe* files, which we then load into the core generator.

### 2.2.1 Soft Switches

Much of the Apple ][e's memory is bank-switched, including the main ROMs. Writes to certain memory locations will toggle which bank is selected. Others access built-in periphrials. Extracting these from the documentation was difficult, so we provide them here:

| Switch | Off Address | On Address | Read Address |
|---|---|---|---|
| 80STORE | W 0xC000 | W 0xC001 | R 0xC018 |
| RAMRD | W 0xC002 | W 0xC003 | R 0xC013 |
| RAMWRT | W 0xC004 | W 0xC005 | R 0xC014 |
| INTCXROM | W 0xC006 | W 0xC007 | R 0xC015 |
| ALTZP | W 0xC008 | W 0xC009 | R 0xC016 |
| SLOTC3ROM | W 0xC00A | W 0xC00B | R 0xC017 |
| 80COL | W 0xC00C | W 0xC00D | R 0xC01F |
| ALTCHARSET | W 0xC00E | W 0xC00F | R 0xC01E |
| TEXT | W 0xC050 | W 0xC051 | R 0xC01A |
| MIXED | W 0xC052 | W 0xC053 | R 0xC01B |
| PAGE2 | W 0xC054 | W 0xC055 | R 0xC01C |
| HIRES | W 0xC056 | W 0xC057 | R 0xC01D |
| BANK1* | A3' | A3 | R 0xC011 |
| HRAMRD* | A1 xor A2 | A1 xnor A2 | R 0xC012 |
| PRE-WRITE* | A0' + W | A0 · R | None |
| HRAMWRT'* | PRE WRITE · A0 · R | A0' | None |

* High Ram Control Addresses are in the 0xC08X range

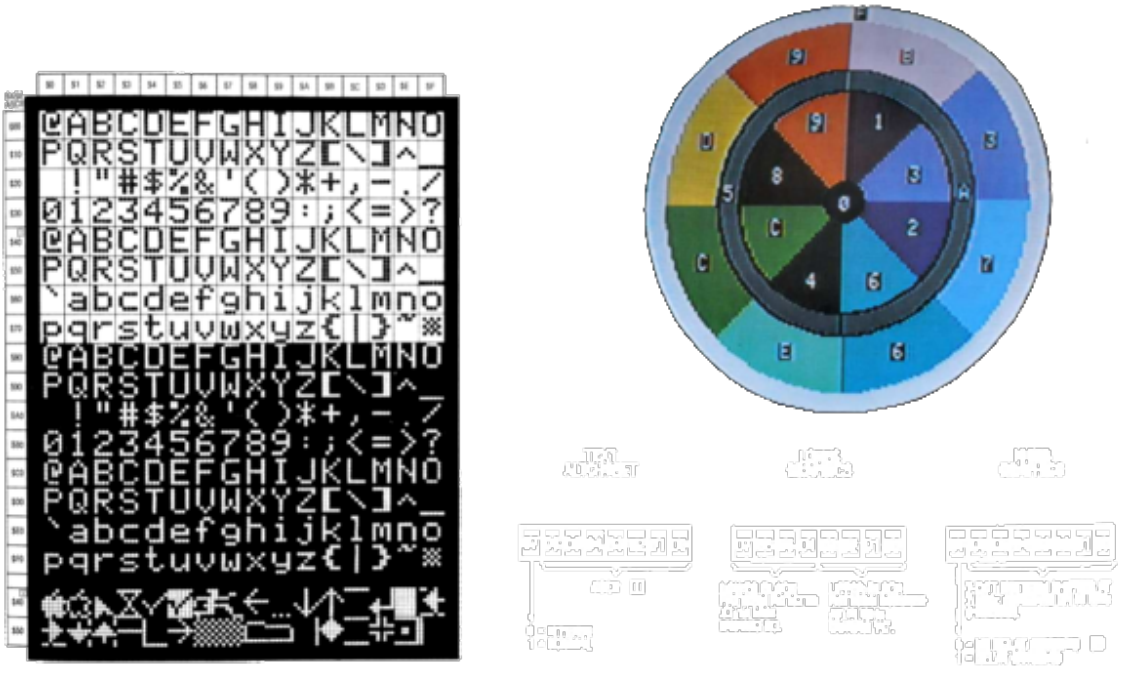
## 2.3 Video

The Apple 2e implemented a few very complex and different graphics modes ranging from high resolution to a text only display. To simplify things we only implemented low resolution graphics modes which included three of the four out of the box graphics modes. These modes were Low Resolution Graphics, 40 Line Text and Mixed Low Resolution Graphics. This mode could be switched at any time by changing the GRAPHICS, HIRES and MIXED soft switches.

Low Resolution Graphics modes has two pages acting as frame buffers in memory. These pages cover the memory ranges from 0x0400 to 0x07FF for page 1 and 0x0800 to 0x0BFF

for page 2. The page being interpreted and displayed can be switched by toggling the PAGE soft switch. In each page the memory was broken up into 128 byte ranges. Each 128 byte range would display 3 rows of text or graphics on the screen depending on the mode set. Because the screen is 40 columns wide and each column was filled by a byte in memory in this left 8 bytes that are unused per every three lines. An added complexity was that each set of 128 bytes' three lines were spaced out all over the screen. For example the first 128 bytes defined the first line, the ninth line and seventeenth line, the second 128 bytes defined the second line, the tenth line and the eighteenth line. This pattern added some complexity to our implementation when attempting to max pixel locations to memory addresses.

Each line on the Apple2e was 8 pixels high while the columns were 7 pixels wide. To make things easier on our implementation we made the columns 8 pixels wide: this allows division to be done with a bit shift. This resulted in each byte mapping to a 8 pixel by 8 pixel block on screen. In Low Resolution Graphics mode a byte was broken into two parts the first 4 bits would set the color for the top half of the block and the second 4 bit would set the color for the bottom half of the block. Because color is 4 bits there were only 16 possible colors. In Text mode a byte would contain a text sprite code that we used to retrieve a sprite for and draw each pixel of this 8 by 8 sprite into the block. Although text sprites in the Apple2e were only 7 pixels wide they were stored 8 pixels wide and thus we did not have to alter the sprite ROM for our implementation. In Mixed mode the top 20 lines of the screen were interpreted as Low Resolution Graphics while the bottom 4 lines were interpreted as Text combining the two modes in a single display.



In our implementation because we did not have a screen that would directly make our desired resolution we build a DVI interface that output a 640 by 480 pixel image with out 320 by 192 pixel screen centered on it. We designed a multi-stage pipeline that would draw

the pixels in order. This pipeline was clocked far faster then the rest of the system to keep the display up to date. To separate this from the main system the memory accessed done by this pipeline hit dual port RAM with separate clocks for each side.

The first step in the pipeline was to determine which address in memory and block the pixel we wanted to draw into was in. If we were outside the screen we set an out of bounds flag and set the address to zero to be dealt with in the next stage. We also calculated a pixel offset to determine which pixel in the block we wanted to draw. The second stage got the byte at the address we calculated or in case of out of bounds chose to pass on all zeros to the next stage. In the third stage on the pipeline we used the mode we were in to determine the pixel color we wanted to draw. If we were in Graphics mode or the graphics section of the Mixed mode we took the byte we were given from memory and used to offset calculated to determine if we wanted the high 4 bits as color or the low 4 bits as color. If we were in Text mode or the text section of the Mixed mode we passed the byte we got from memory concatenated with the offset into a sprite ROM to get the desired pixel color. If we were out of bounds we interpreted our zeros as graphics which results in a black border. In the next clock cycle we output the DVI color as 24 bit color selected with a case statement from our 4 bit colors. The DVI output used a clock at twice the speed of the pipeline because we output the 24 bit color in two 12 bit chunks and thus there were two ticks of this clock for every tick in the pipeline.

The original video driver for the Apple IIe used a DMA scheme. Because memories of the time were much faster than contemporary processors, their video controller was able to read a pixel in between processor ticks. To avoid the complexities of DMA, our design uses a dual-port RAM. This has the side effect that the 100MHz DVI driver code is isolated from the 3.125MHz main bus by the RAM IP core, so we do not have to write any synchronization logic.

#### 2.3.1 Notes

Even after reverse-engineering Team Dragonforce's DVI driver code, there were several tricky issues:

1. *DVI_DE* needs to go low after every h-line, i.e. it is only high when we are transmitting a pixel that is in the visible window

2. *DVI_RST_B* can just be held high: I2C will get it into a known state

# 3 Non-Functional Systems

## 3.1 Sound

The Apple ][e uses a 1-bit speaker driver, which we are emulating with the 1-bit piezo driver on the board. This works in a simple square-wave demo, but unfortunately the switch responsible refuses to latch when attached to the address bus. On an oscilloscope, the driver appears to enter a metastable state and then decay to 0 within a hundredth of a clock cycle instead of toggling. We suspect a subtle timing issue, but there are no warnings. Since the

demo software does not use sound anyway, fixing it was not sufficiently high priority to get it completed before demo.

### 3.2 Disk

The original intention was to implement the floppy disk system by using the CompactFlash slot as a backend. Already behind schedule due to the 6502 core, we started this section behind schedule only to learn that our original research had deceived us: the Apple ][e's floppy disk was accessed as a stream, not a block. Further, control was not via registers, but by direct control of the stepper motors. The SystemACE interface was also excessively complex. With less than a week left before demos, we decided to use a Block ROM as a backing store, but we were still unable to fully debug the state machine for track selection control, so floppy disks remain unimplemented.

This has a side effect that the fastest way to get a program into our computer is to type it into the BASIC interpreter. Entering binary programs is even more difficult.

## 4 Persistent Programming

### 4.1 ACE

SystemACE can, in addition to providing block-based CompactFlash access to a running FPGA, configure the FPGA from a FAT filesystem on CompactFlash. The CF card is first formatted as a FAT12 or FAT16 filesystem, with special parameters.

```
sudo /sbin/mkfs.msdos -F 16 -R 1 -s 8 /dev/sdb1}
```

A special directory structure, a *xilinx.sys* file, and a *.ace* file generated from the *.bit* file are then loaded onto the card. These files can be generated via IMPACT GUI or via the *bit2svf.scr* and *svf2ace.scr* files the course staff provided.

Unfortunately, this method did not work: we suspect the filesystem was formatted improperly, as the SystemACE controller is very picky about the exact parameters of its filesystem. Because the errors given by SystemACE are very opaque (the "ERR" LED lights up), and because PROM programming was working, this was not completed.

### 4.2 PROM

We were able to generate a *.mcs* PROM image for the Xilinx XCF32P flash PROM on the board and program it over JTAG using IMPACT. With bit 6 of the programming configuration switches set high, this causes our code to load at startup.

## 5 Software

At boot, our system runs the BASIC interpreter that was on the stock Apple ROM. Our public demo used a modified version of a BASIC Pong from Vectronics Apple World. It

uses low-resolution graphics to control the keyboard.The delay loop at 130 was inserted to compensate for our system running at 3 times the stock clock speed.

```
10 REM APPLESOFT PONG BY VECTRONIC
20 HOME
100 GOSUB 3000
110 X = 11: Y = 12: Z = 13: XB = 31: YB = 12: S = 0
120 COLOR= 9: PLOT X,Y: PLOT X, Z: COLOR= 13: PLOT XB,YB
130 L=0
131 V=2000
132 IF L=V THEN 135
133 L=L+1
134 GOTO 132
135 T= PEEK (-16384)
140 IF T = 139 AND Y &lt; &gt; 1 THEN 300
150 IF T = 138 AND Z &lt; &gt; 29 THEN 400
160 IF X + 1 = XB AND Y = YB THEN 4000
170 IF X + 1 = XB AND Z = YB THEN 4000
180 IF X = XB AND Y + 1 = YB THEN 4000
190 IF X = XB AND Z + 1 = YB THEN 4000
200 IF XB = 31 THEN 2500
210 IF XB = 8 THEN 2600
220 IF YB = 29 THEN 2700
230 IF YB = 1 THEN 2800
240 IF S = 0 THEN 2900
250 IF S = 2500 THEN 2515
260 IF S = 2600 THEN 2615
270 IF S = 2700 THEN 2715
280 IF S = 2800 THEN 2815
290 IF S = 4000 THEN 4015
299 GOTO 130

300 COLOR= 0: PLOT X,Y: PLOT X,Z
310 Y = Y - 1: Z = Z -1
320 COLOR= 9: PLOT X,Y: PLOT X,Z
330 T = PEEK(-16368)
340 GOTO 130

400 COLOR= 0: PLOT X,Y: PLOT X,Z
410 Y = Y + 1: Z = Z + 1
420 IF Z = 35 THEN 130
430 COLOR= 9: PLOT X,Y: PLOT X,Z
440 T = PEEK(-16368)
450 GOTO 130
```

```
2000 REM RANDOM NUMBER GENERATOR
2100 K = INT (10*RND(1))
2200 RETURN

2300 REM RANDOM SELECTOR
2320 IF K &lt; = 3 THEN 2370
2350 IF K &gt; 3 AND K &lt; 6 THEN 2380
2360 IF K &gt; = 6 THEN 2390
2370 K = 1: GOTO 2400
2380 K = 2: GOTO 2400
2390 K = 3
2400 RETURN

2500 GOSUB 2000
2510 GOSUB 2300
2515 S = 2500
2516 DR = 1
2520 IF K = 1 THEN 2550
2530 IF K = 2 THEN 2560
2540 IF K = 3 THEN 2570
2550 COLOR= 0: PLOT XB, YB: XB = XB - 1: YB = YB - 1: GOTO 2580
2560 COLOR= 0: PLOT XB, YB: XB =  XB - 1: YB = YB + 1: GOTO 2580
2570 COLOR= 0: PLOT XB, YB: XB  = XB - 1
2580 COLOR= 13: PLOT XB, YB: GOTO 130

2600 REM LOSE GAME
2605 Z = 0
2610 W = PEEK(-16336)
2620 IF Z = 20 THEN 2650
2630 Z = Z + 1
2640 GOTO 2610
2650 PRINT "YOU LOSE"
2660 PRINT "GO AGAIN? (Y OR N)"
2665 INPUT N$
2670 IF N$ = "Y" THEN 2691
2680 IF N$ = "N" THEN 2695
2690 IF N$ &lt; &gt; "Y" AND N$ &lt; &gt; "N" THEN 2660
2691 HOME: COLOR= 0: PLOT XB,YB
2692 GOTO 100
2695 TEXT: HOME
2699 END

2700 REM UPSWING
2715 S = 2700
2720 IF DR = 1 THEN 2750
```

```
2730 IF DR = 0 THEN 2760
2750 COLOR= 0: PLOT XB, YB: XB = XB - 1: YB = YB -1: GOTO 2780
2760 COLOR= 0: PLOT XB, YB: XB = XB + 1: YB = YB -1: GOTO 2780
2780 COLOR= 13: PLOT XB, YB: GOTO 130

2800 REM DOWNSWING
2815 S = 2800
2820 IF DR = 1 THEN 2850
2830 IF DR = 0 THEN 2860
2850 COLOR= 0: PLOT XB, YB: XB = XB - 1: YB = YB +1: GOTO 2880
2860 COLOR= 0: PLOT XB, YB: XB = XB + 1: YB = YB +1: GOTO 2880
2880 COLOR= 13: PLOT XB, YB: GOTO 130

2900 REM START THE BALL
2910 DR = 1
2920 GOTO 2500

3000 REM PLAY AREA
3100 GR: COLOR= 3: VLIN 0,30 AT 7
3200 COLOR= 12: VLIN 0,30 AT 32
3300 COLOR= 12: HLIN 7, 32 AT 0
3400 COLOR= 12: HLIN 7,32 AT 30
3999 RETURN

4000 GOSUB 2000
4010 GOSUB 2300
4015 S = 4000
4016 DR = 0
4020 IF K = 1 THEN 4050
4030 IF K = 2 THEN 4060
4040 IF K = 3 THEN 4070
4050 COLOR= 0: PLOT XB, YB: XB = XB + 1: YB = YB + 1: GOTO 4080
4060 COLOR= 0: PLOT XB, YB: XB = XB + 1: YB = YB - 1: GOTO 4080
4070 COLOR= 0: PLOT XB, YB: XB = XB + 1
4080 COLOR= 13: PLOT XB,YB: GOTO 130
```

# 6 Personal Statements

## 6.1 Russell Nelson

I am proud of what we accomplished this semester. We actually had the system working and running the basic interpreter. As with every project if we had planned a bit better and worked a bit harder we could have had so much more. But in our failure and success we learned the importance of strict research and planning. In my opinion it was not the

scheduling that was the problem but the research that went into determining our schedule. If we had done more research at the beginning of the project we would have better accessed what we could accomplish and scale our work to focus on those sections. This would have lead us to not try and build a 6502 from scratch, buy get a working floppy interface and focus on interfacing peripherals and different modes.

Another thing that we failed to due was make people responsible for deadlines in our schedule. Things dragged on after they should have because we did not enforce strict deadlines within our group. This also meant that when one aspect fell behind it fell very far behind.

I had fun succeeding and failing and our system works similar to how the original out of the box system works so I am very happy with that. I do wish that it did some cooler things like playing ROMS but we needed another week to get that working.

If I could do it all again I would pick the same project have every single detail mapped out by the proposal presentation. If we had this done it would have been smooth sailing but alas I don't have a time machine. I will be able to say I built an Apple 2e from scratch with pride when someone asks what I did my senior year of college.

## 6.2 Andrew Brock

This project taught me a lot about my limits. I was the primary person tasked with constructing our own 6502 core, mostly because I "thought it would be a cool thing to have done." I came very close to succeeding, but it tied up nearly two months of my time, all of which could have been better spent on other systems, like sound or the floppy emulator. In hindsight, even if the decision had been made to attempt to write our own 6502 again, I would have set a far earlier deadline for it and switched to acquired code more quickly.

We also should have enforced better use of version control and commit messages. The messages we have are incredibly terse, and while they give some idea what was done, they do not give a very good idea, nor necessarily tell who did it. This makes it extremely difficult to tell who did what work, or even if some group members did any work at all.

I am proud of what we accomplished, but I feel that if we had managed our time and our selves better, we could have done so much more. Both the sound and floppy subsystems were attempted at the very last minute, in large part because I tied myself up with the 6502 debacle for a month and a half.

## 6.3 Gabriel Garcia

I would like to start by saying I am incredibly pleased with our end project. Like everyone else, I wish we had more time to work on the project and feel like we could have achieved more if we planned better. Looking back on the project, I feel there are several key things that I wish I could have done better. First and most importantly with all group activities I feel we needed better communication. There were several occasions where I would misunderstand a request, spend a while working on them, only to realize that I was working on the wrong part; this lead to wasted time that could have been better used. On the other hand, there were also times when I failed to communicate the exact nature of a problem which lead my teammates to work on bad information.

Another area that we need to improve on was planning, its a common problem that could always be done better. There were times when we missed our deadlines, and instead of looking at quicker alternatives we kept extending our deadlines, and before we knew it we spent far too much time on a problem that could have been solved quickly if we just took a step back.

Finally, the area that I realized I needed to work on most was basic organization and documentation. I spent most nights doing research on various systems, making mental notes of them, only to wake up the next morning and forgetting which part of the document contained the relevant information.

Like I said before while we did not achieve everything we wished, I am still amazed with how well everything worked together.

# 7 References


`http://www.everything2.com/title/6502+addressing+modes` A very nice concise reference to 6502 addressing modes

`http://6502.org/tutorials/6502opcodes.html` Documentation of all documented NMOS 6502 instructions, good for processor implementation and disassembly of a trace of the data bus

`http://www.vectronicsappleworld.com/appleii/pong.html` Our demo software

`https://github.com/Arlet/verilog-6502` Arlet Ottens' free 6502 implementation, which saved us from biting off more than we could chew

**Sather, Jim. *Understanding the Apple ][e.* Quality Software: Chatsworth, CA, 1985.** A comprehensive reverse engineering of the Apple ][e, including insight into the design process from Steve Wozniak. This is the only source that understands all the soft switches and their purposes.