ECE 385

Final Project Report

Adam Urish and Jason Zou

(adamwu2, jasonz3)

5/12/21

Introduction

For our final project, we attempted to implement apple IIe functionality with our FPPGA. The Apple IIe was a personal computer released by apple in 1983 and was an enhanced version of earlier Apple II computers. For our project we were able to get the System Monitor, graphical output, and most of the low-level implementation of the Apple IIe working.

Written Description

In our design, we tried to match the design of the original Apple IIe as much as possible. We removed some features and redesigned some chips to optimize the computer for the FPGA, but we tried to stay in the "spirit" of the Apple IIe's design. In this section, we will describe the components piece by piece, and explain how they connect together.

The 6502 CPU

We obtained the Verilog files for the 6502 online. They were generated from scans of an original 6502's integrated circuit, so the behavior is very accurate. We used the 6502 as somewhat of a black box in our design, as did the original IIe. Important for us to consider were the address output pins, the data in and out pins, the read-write signal pin, and of course the clock pin. The IIe used a 1.022727 MHz clock for the processor, and we matched this as closely as possible in our design. We generate the clock signals used throughout with a PLL in our Platform Designer setup, we output 14.31818 MHz and 2.04545 MHz master signals which are divided as needed. The master signals are fed into the PAL, a clock controller which divides the masters and outputs the CPU clock, the video clock, and memory access clock. The address pins output the 16 bit memory address that the processor would like to access, and the data in and data out pins supply the data read from that address or to be written to that address respectively. The read-write pin dictates this behavior, if it is high data is being read, and if it is low data is being written.

The Memory

The IIe had 64 KB of RAM and 16 KB of ROM. If you are doing the math in your head, you may be wondering how this was accomplished with an only a 16 bit address space, which can only directly address 64K locations. The IIe, as well as our design, accomplishes this feat with bank switches memory. Before accessing ROM or RAM, the program must access a specific memory address in the 0xC08X range which will setup the specified bank of memory for reading or writing. This routing is controlled by the Memory Management Unit. On the IIe this was a custom IC that was physically separate from the memory, but in our design we grouped them into one module to keep track of the control signals. We also added direct video address and video data pins to our MMU, to accommodate the faster video data access required by the modern VGA standard we are using. We are accessing the video data at ~25 MHz, the standard 640x480 pixel timing, which is out of phase with the rest of the CPU timing. Aside from the actual memory data, the MMU also outputs some control signals for the I/O. If the CPU accesses the memory mapped I/O block, the MMU outputs a signal to switch the CPU's data input from the MMU to the IOU, which will be described later.

The I/O

In the original IIe, the I/O was managed by the Input Output Unit, a custom IC. We have a similar setup in our design, however the I/O is more complicated. The keyboard is the main method of input. It is first handled by a NIOS-II instance which communicates with the USB chip and exports the keycode on a wire. This keycode is then decoded by our AY3600 Pro module. This was an actual chip in the IIe as well, except instead of decoding keycodes it decoded electrical signals from the Apple Keyboard. The AY3600 combinationally converts whatever keycode is exported to its ASCII representation, which is what the IIe uses in programming and in graphics. Programs can get the currently pressed ASCII character by accessing a specific address in the memory mapped I/O block, and can access another address to check if any key is down. The main output device in our implementation is a VGA monitor. It required some tweaking to work with the IIe's display output, as the 25MHz pixel clock we use is much faster than the 7MHz clock used on the IIe. We accomplish the output by using the VGA controller from Lab 6, and scanning over all the pixels as usual. We then only access the display block of the IIe memory in a floating box of 240x192 pixels, a similar resolution to the original Apple display. This ensures that we do not have too many memory accesses that might mess up the CPU's functioning.

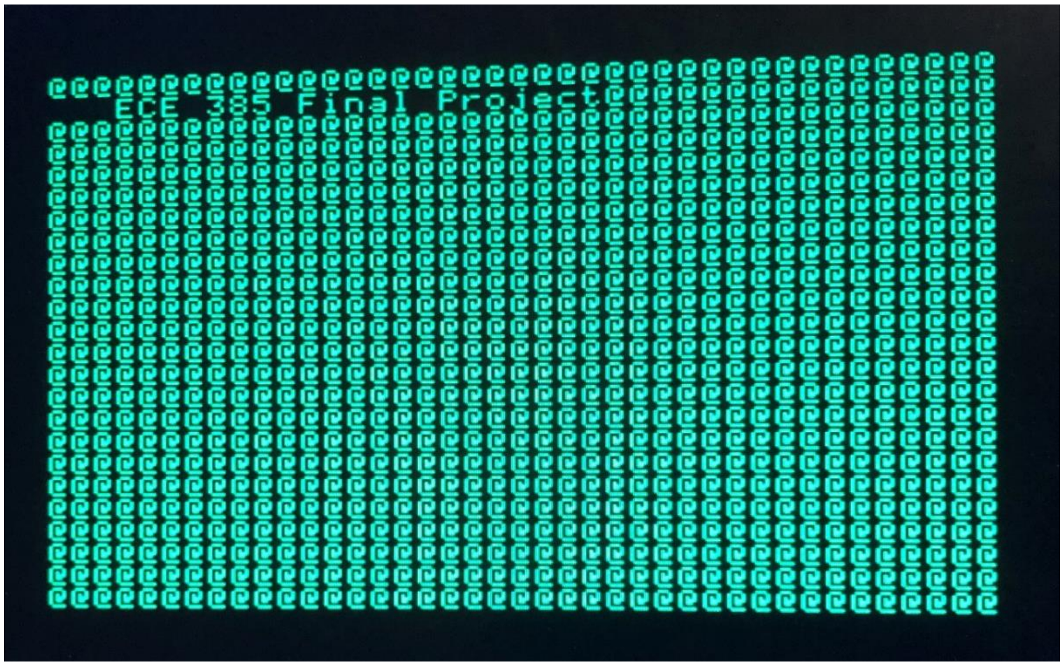
Software

The software that we are running on our implementation is a ROM dump from the original IIe. It contains the system monitor, a low level program that can explore memory, perform basic operations, and jump to other programs. This program is automatically started when the IIe is reset, which on our implementation is done by pressing the KEY1 switch. The monitor is entirely contained inside the 16 KB of ROM, however it has free access to the rest of the computer thru memory examination as well as control of the CPU. Outside of this, we are also running software on the NIOS-II, the same code from Lab 6. It handles the USB chip communication, and outputs the keycodes to the FPGA. Below we have included some sampled outputs generated by the IIe, showing graphical output and the monitor's functioning.

Graphical Output

Startup Splash Screen

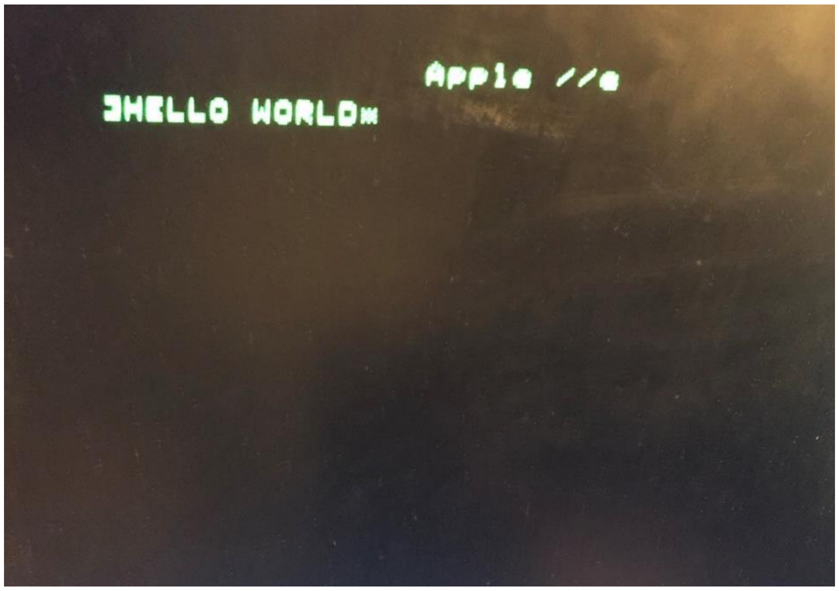
We manually insert the ASCII values for this splash message during programming



```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
   ECE 385 Final Project@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

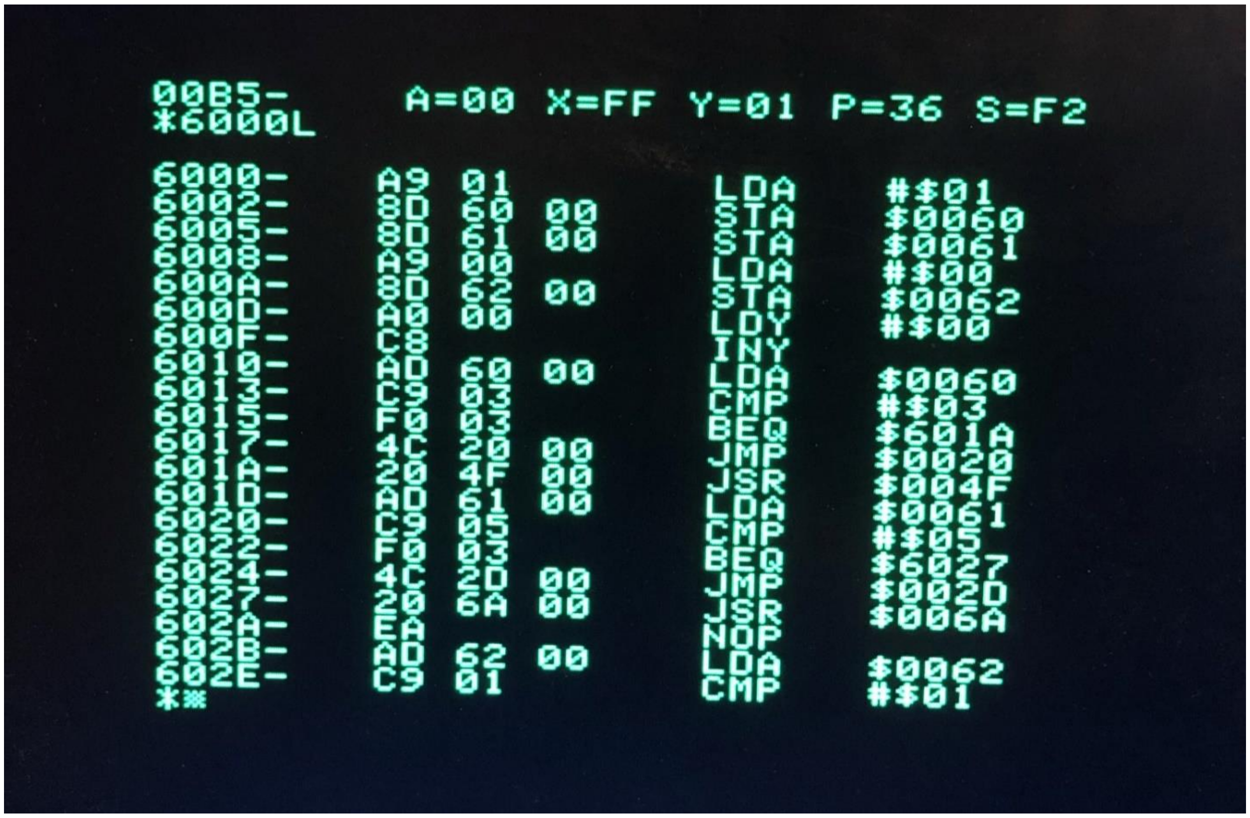
Apple IIe Prompt

“Hello World” was typed to showcase keyboard input



```
                  Apple //e
]HELLO WORLD
```

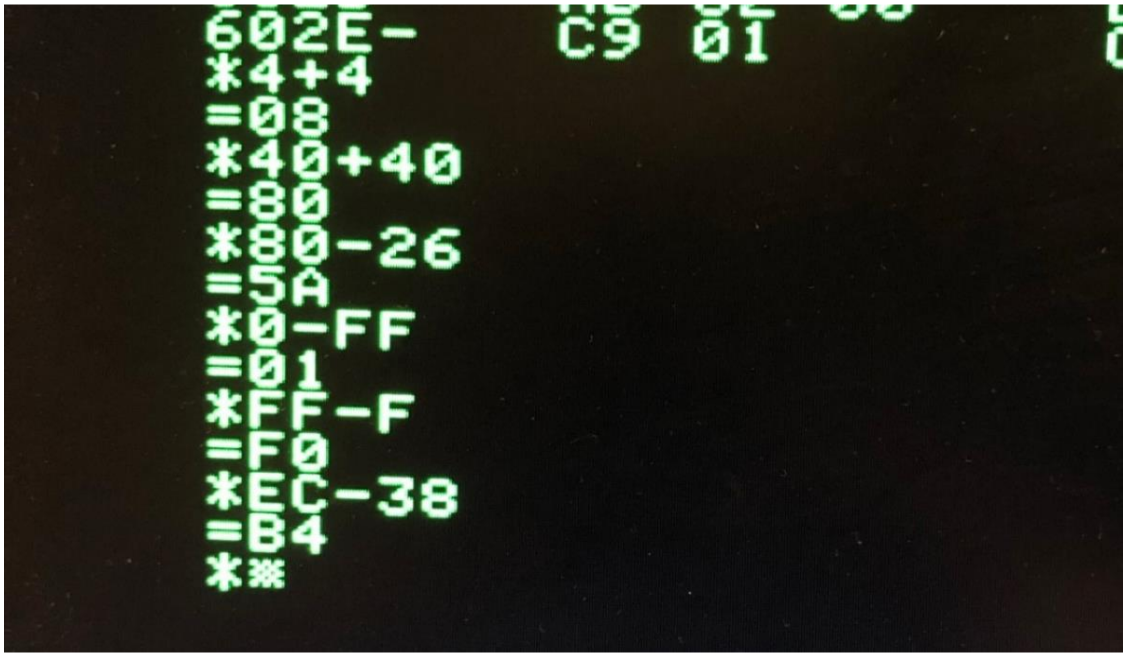
System Monitor Scanning Memory at 0x6000



```
00B5-    A=00 X=FF Y=01 P=36 S=F2
*6000L

6000-   A9 01       LDA   #$01
6002-   8D 60 00    STA   $0060
6005-   8D 61 00    STA   $0061
6008-   A9 00       LDA   #$00
600A-   8D 62 00    STA   $0062
600D-   A0 00       LDY   #$00
600F-   C8          INY
6010-   AD 60 00    LDA   $0060
6013-   C9 03       CMP   #$03
6015-   F0 03       BEQ   $601A
6017-   4C 20 00    JMP   $0020
601A-   20 4F 00    JSR   $004F
601D-   AD 61 00    LDA   $0061
6020-   C9 05       CMP   #$05
6022-   F0 03       BEQ   $6027
6024-   4C 2D 00    JMP   $002D
6027-   20 6A 00    JSR   $006A
602A-   EA          NOP
602B-   AD 62 00    LDA   $0062
602E-   C9 01       CMP   #$01
*
```

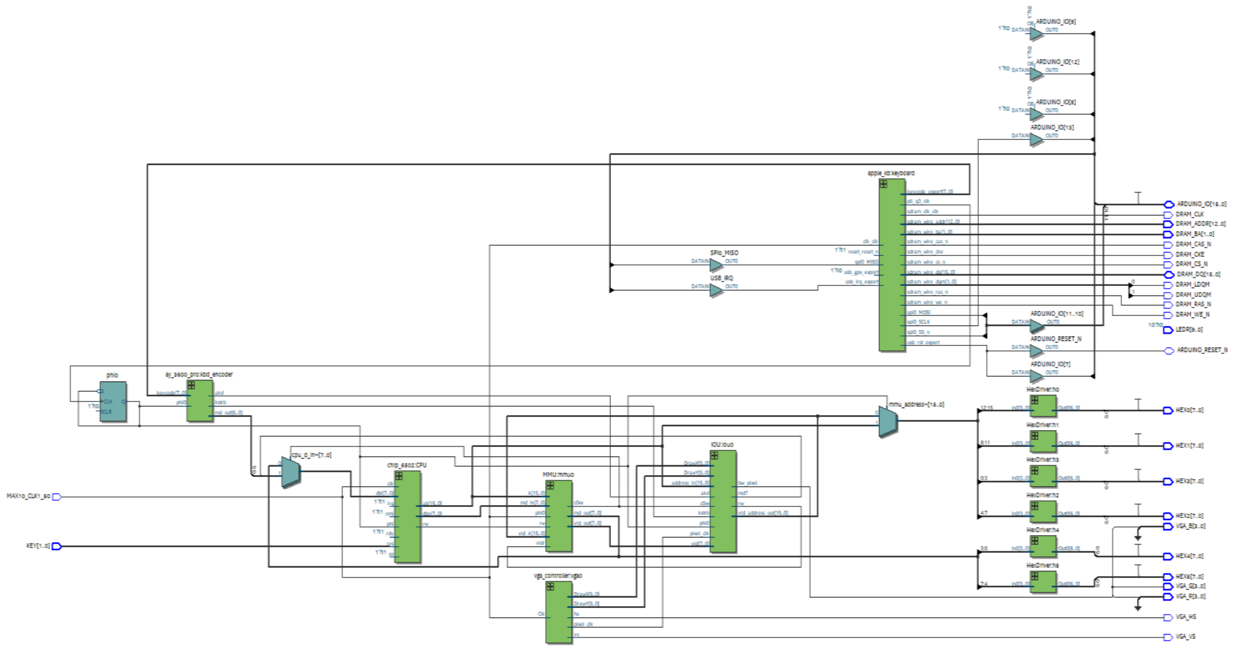
System Monitor Performing Basic Signed Calculations



```
602E-      C9  01
*4+4
=08
*40+40
=80
*80-26
=5A
*0-FF
=01
*FF-F
=F0
*EC-38
=B4
*
```

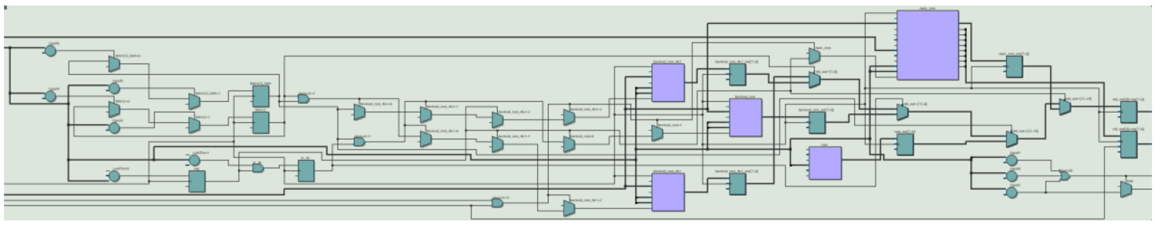
Block Diagrams

Top Level Block Diagram

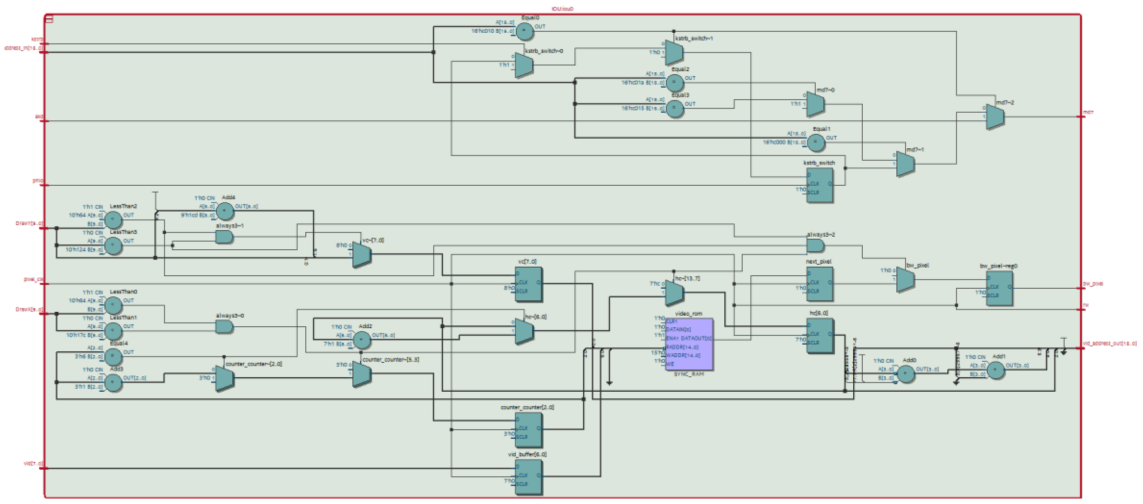


MMU Block Diagram

(Note the 5 blue boxes: these are the various RAMs and ROMS)



IOU Block Diagram



Design Resources and Statistics

LUT	**4758**
DSP	**0**
Memory (BRAM)	**732296 bits**
Flip-Flop	**3115**
Frequency	**98.29 MHz**
Static Power	**102.42 mW**
Dynamic Power	**0.94 mW**
Total Power	**112.66 mW**

Module Descriptions

Module: AppleIIe.sv

Inputs: MAX10_CLK1_50, [1:0] key

Outputs: [7:0] HEX0, [7:0] HEX1, [7:0] HEX2, [7:0] HEX3, [7:0] HEX4, [7:0] HEX5, DRAM_CLK, DRAM_CKE, [12:0] DRAM_ADDR, [1:0] DRAM_BA, DRAM_LDQM, DRAM_UDQM, DRAM_CS_N, DRAM_WE_N, DRAM_CAS_N, DRAM_RAS_N, VGA_HS, VGA_VS, [3:0] VGA_R, [3:0] VGA_G, [3:0] VGA_B

Inouts: [15:0] DRAM_DQ, [15:0] ARDUINO_IO, ARDUINO_RESET_N

Description: Houses the main inputs and outputs for the system. This module initializes various I/O components, such as the keyboard, and other contents that the system keeps track of.

Purpose: This module is the top-level module of the project. It establishes connections with and initializes the other modules.

Module: MMU.sv

Inputs: [15:0] A, [15:0] vid_A, [7:0] md_in, phi-, rw, vidr

Outputs: kbd, c0xx, [7:0] md_out, [7:0] vid_out

Description: The MMU takes in address and video address signals, the phi0 clock signal, and read write control rw and vidr signals and outputs kbd, c0xx, md_out, and vid_out signals.

Purpose: The memory management unit is responsible for managing the RAM of the Apple IIe.

Module: PAL.sv

Inputs: pll_14m, pll_q3

Outputs: clk_M7, clk_M3_5, phase1_clk, phase2_clk

Description: Takes in clock signals pll_14m and pll_q3 and divides them into signals for clk_M7, clk_M3_5, phase1_clk, phase2_clk.

Purpose: The Pal takes ~14 Mhz and ~2 Mhz input clocks and divides them down to ~7, ~3.5, and ~1 Mhz clocks for other components.

Module: IOU.sv

Inputs: phi0, c0xx, akd, pixel_clk, kstrb, [7:0] vid, [15:0] address_in, [9:0] DrawX, [9:0] DrawY

Outputs: [15:0] vid_address_out, md7, bw_pixel, rw

Description: The IOU takes in various clock signals, a video signal, address_in, as well as DrawX and DrawY parameters and outputs a video address, md7, bw_pixel, and rw signals. It also reads the video ROM.

Purpose: The IOU is responsible for managing the video output of the Apple IIe, such as what part of the screen should be drawn.

Module: dram_bundle.sv

Inputs: rw, clk, [15:0] A, [7:0] md_in

Outputs: [7:0] md_out

Description: Takes in rw, clk, md_in, and address signals and outputs md_out. At every positive clock edge, if the rw signal is low, md_in is written to memory at the address inputted. If the rw signal is high, the memory at the inputted address is read and passed to md_out.

Purpose: Functions as the dram for the system. The module also loads the ROM hex file into memory.

Module: DAC.v

Inputs: clk, [11:0] countA, [11:0] countB

Outputs: rstn, cs_n, sclk, mosi

Description: Provided in external 6502 code, used for controlling input and output.

Purpose: Used in conjunction with the 6502. Steps through various states.

Module: chip_6502.v

Inputs: clk, phi, res, so, rdy, nmi, irq, [7:0] dbi

Outputs: [7:0] dbo, rw, sync, [15:0] ab

Description: A 6502 CPU that was taken from an external source.

Purpose: Used as the main CPU of the Apple IIe.

Module: ay_3600_pro.sv

Inputs: phi0, [7:0] keycode

Outputs: [6:0] md_out, kstrb, akd

Description: Takes in a phi0 clock signal and keycode from the nios2 and outputs an ASCII keycode to md_out as well as a key strobe (kstrb) and any key down signal (akd).

Purpose: Maps the inputs from the keyboard to various ASCII keycodes.

Module: HexDriver.sv

Inputs: [3:0] In0

Outputs: [6:0] Out0

Description: Maps Out0 to hardcoded values based on In0. The hardcoded values represent the encoding for In0 on the hex display.

Purpose: To display values on the hex display. They need to be mapped to a specific constant to control which bars are illuminated.

Module: VGA_controller.sv

Inputs: clk, Reset

Outputs: hs, vs, pixel_clk, blank, sync, [9:0] DrawX, [9:0] DrawY

Description: Takes clock and reset as input signals and outputs horizontal sync pulse, vertical sync pulse, pixel clock, blanking intervals, composite sync signals, and the horizontal and vertical coordinates being drawn.

Purpose: VGA_controller produces the horizontal and vertical sync timing signals required by the VGA monitor.

Conclusion

The process of creating the Apple IIe was difficult and not everything was functional in the end. We had trouble with getting assembly code to run on the Apple IIe and had some trouble with getting the terminal to display properly; however, we were able to get the system monitor, graphical output, and general low-level implementation of the Apple IIe working. We would like to have implemented some sort of functionality to read external data to run games like Oregon Trail, however we ran out of time to set this up. We believe that with the base that we have built, it would be possible to implement the color high resolution graphics and output it to the VGA monitor.