CECS 301

Lab #5: Digital Clock Implementation on FPGA

Jonathan Fuentes

Dan Cregg

Due: April 27th, 2025

Table of Contents

Introduction

The purpose of this project was to finally implement the 24/12 hour clock that we have been working on for the last 5 labs. This is the closure to the long awaited lab implementation of a 24/12 hour AM PM clock. This lab lets us get more comfortable debugging in Verilog and gaining a greater understanding of combinational logic in Verilog. Being able to figure out what are our issues with our code can help give us a better understanding of the concept behind it.

Project Description

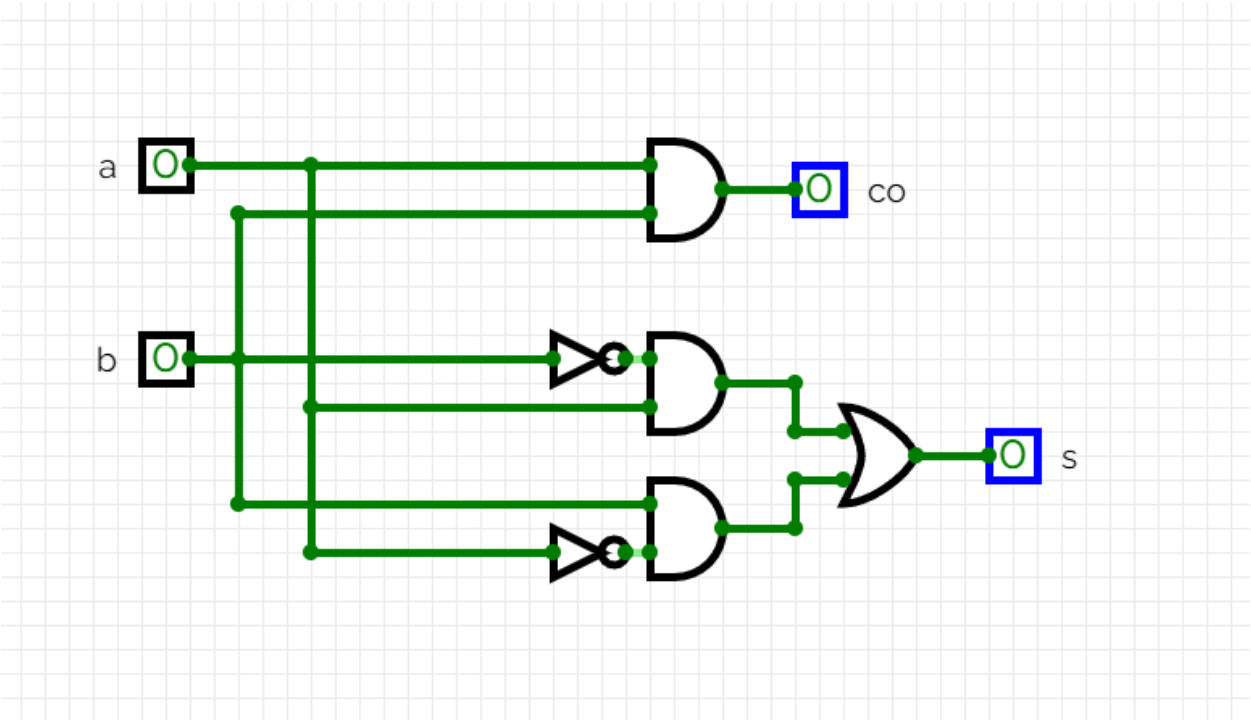
To do this lab we were first given the option to complete the 12 hour clock module, either in CircuitVerse or direct Verilog implementation on the Nexys A7 100T. Both options would eventually have to be implemented on the Nexys A7 100T board anyway. I chose the first option which was to do it in CircuitVerse since it seemed easier for me to understand the logic behind the 12 hour clock first and then to change it when implementing it in Verilog. Essentially making it easier for me to debug. The implementation would be the same functionality that we had in our previous lab, however, this time we would have to be able to switch between 12 and 24 hour time modules.

Details

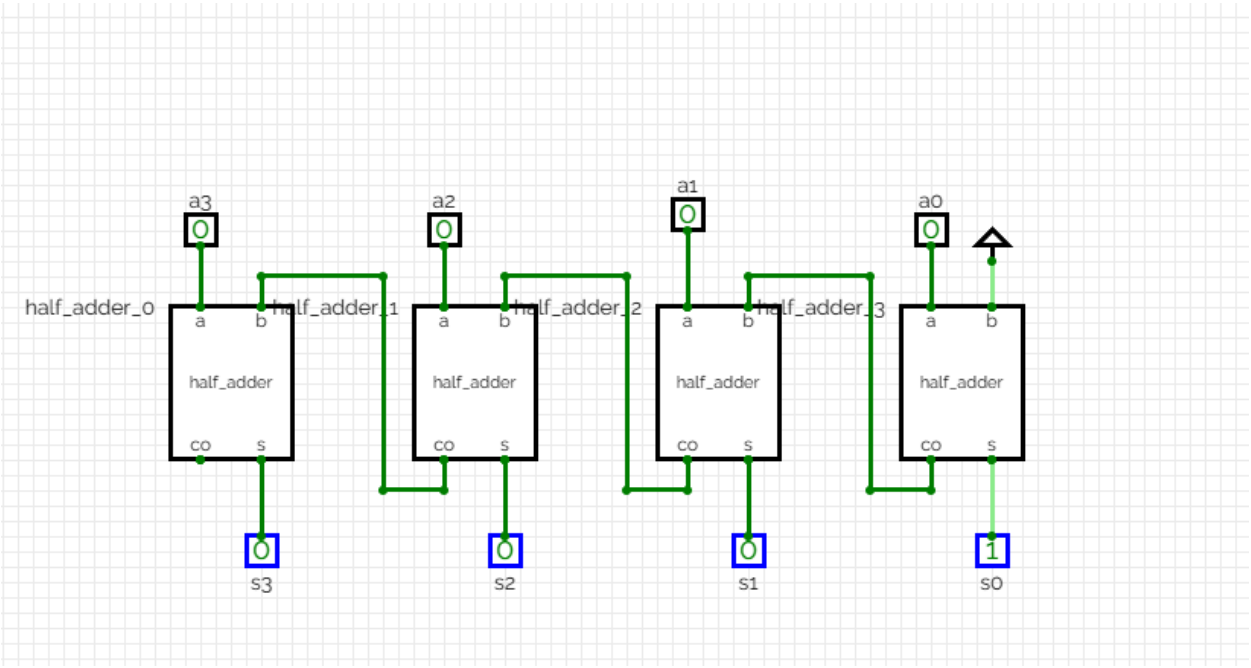
I first began by making the CircuitVerse implementation of my clock. This included making the logic behind the 12 hour time module, which is to reset after 9 to 0, but reset to 1 when it's 12. And then you also had to implement the switching between the 24/12 hour modules.

To implement the logic behind the 12 hour time module I had to create a new counter that could implement the logic necessary for 12 hour time. This includes resetting after 9 to 0, for the hours from 1-9 and then also resetting on when the clock is 12 to 1. To make my counter I included an incrementer that would increment initially after reset before the clock edge. I also added functionality that would allow me to choose the output between this incrementer and without the incrementer (starting at 1 vs. starting at 0) for both the decade and MOD6 counter. Another idea I had for this implementation was using a comparator to compare the different bits and feed it through a MUX but I decided that it was too difficult for me.

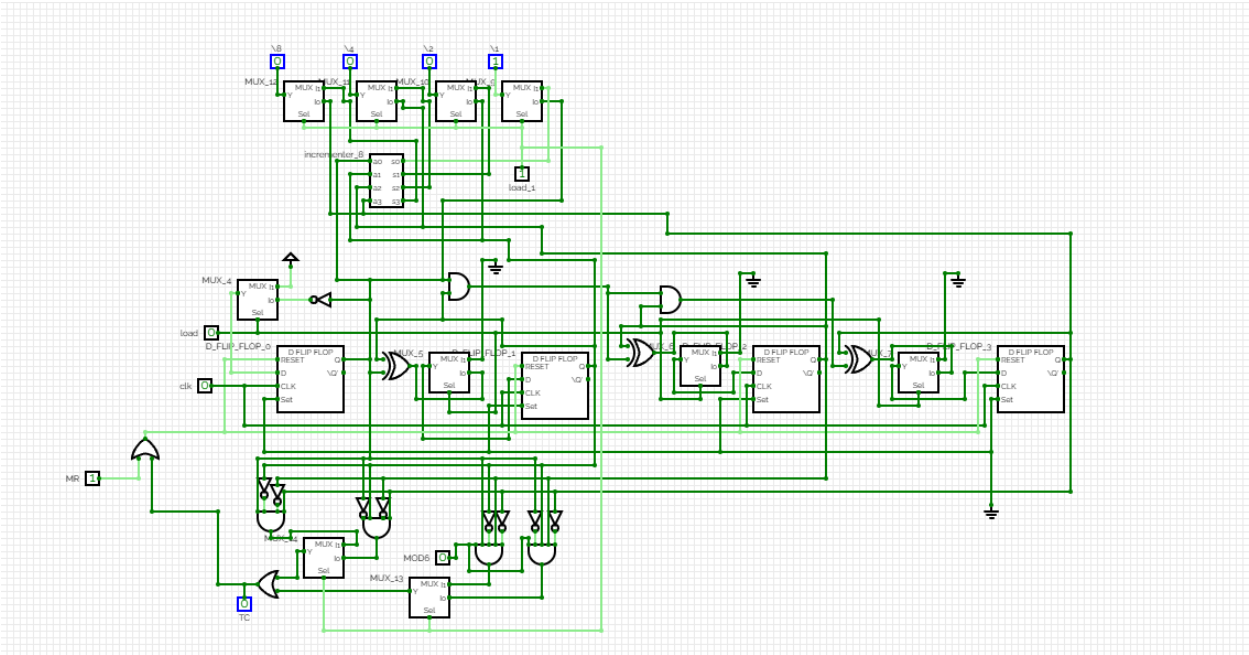
Simple Half Adder Module:



4-bit Incrementer:

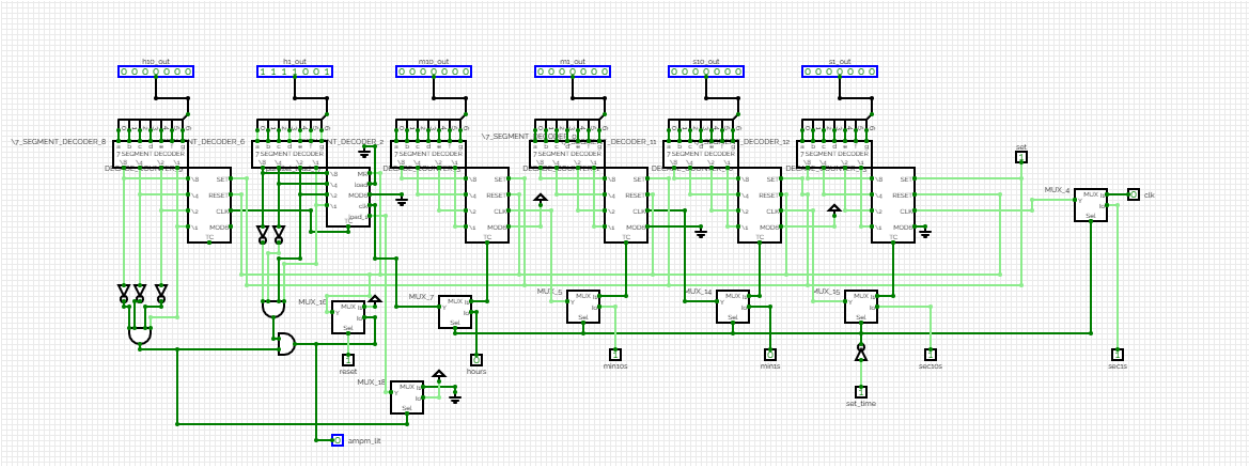


Parallel Incremental Counter Circuit:



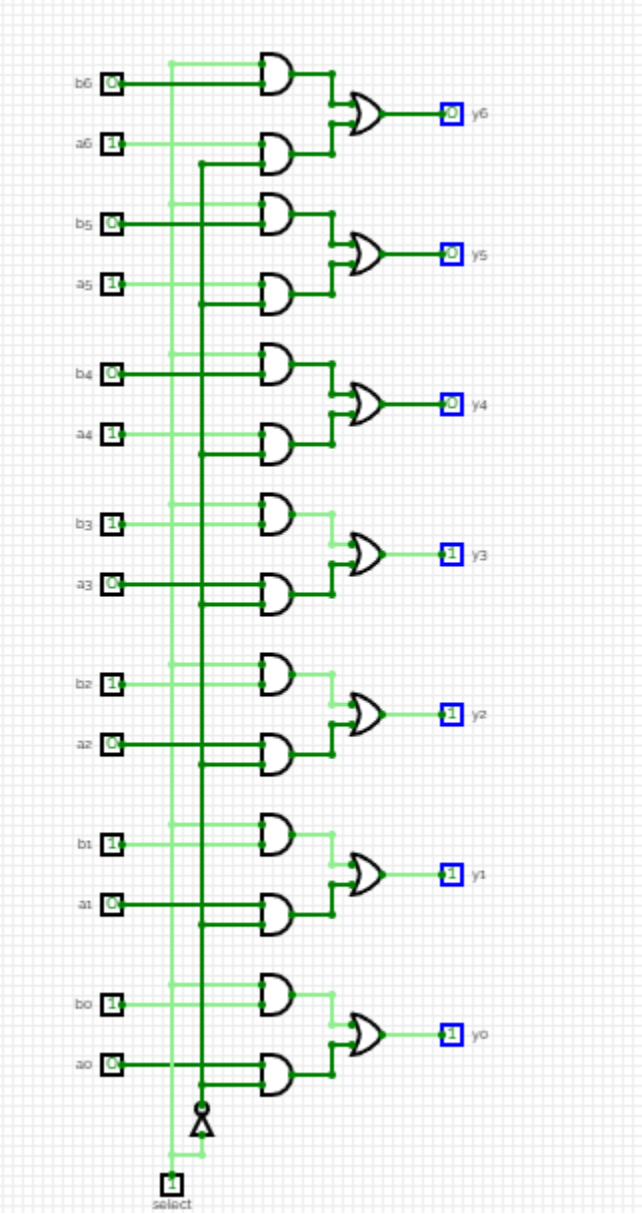
After I finished making the counter circuit I was then able to implement my 12 hour clock module circuit. I included the necessary logic to change from AM to PM when the hours reach 12 in the form of a pulse which would be sent to a D Flip Flop in a later module to toggle state on the clock using an LED.

CircuitVerse 12 Hour Time Module:



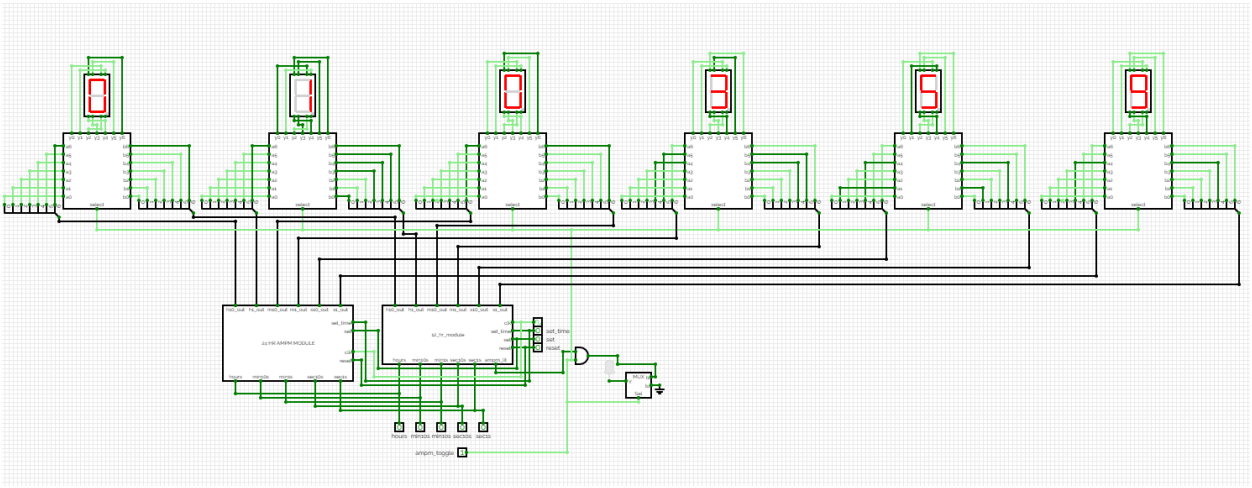
After finishing both the 24 hour and 12 hour time modules I had to find a way to be able to switch between them. This would be done by connecting the same inputs on both of them and

then multiplexing the outputs of the segments to be able to switch between them. To do this I used a 7-bit 2 to 1 multiplexor which would then be connected to each segment from both modules and would be toggled with an AMPM switch.



7-bit 2 to 1 MUX CircuitVerse Implementation:

Finished CircuitVerse Implementation of 24/12 hr Clock:



The next step for me after finishing the module in CircuitVerse was to implement it in Verilog using the export tool that CircuitVerse gives us in addition to previous Verilog code that I had for the 24 hour module.

Most of the logic behind this comes from the previous implementation of the 24 hour module, where essentially I multiplexed the segment outputs between for the seven segment display to display hours, minutes, and seconds. Since I essentially did the same thing as the original implementation, as such, most of the code looks the same. I still decided to code in my own D Flip Flop and to create a clock divider module. You also had to set the refresh rates for the seven segment displays so I handled that.

D Flip Flop Module Verilog Code:

```
module D_FLIP_FLOP(Q, notQ , Set, RESET, D, CLK);
  input D;
  input CLK;
  input RESET;
  input Set;
  output reg Q;
  output notQ;
  assign notQ = ~Q;
  always@(posedge CLK or posedge RESET or posedge Set)
  begin
    if(RESET == 1'b1)
        Q <= 1'b0;
    else if(Set == 1'b1)
        Q <= 1'b1;
    else
        Q <= D;
  end
endmodule
```

Clock Divider Verilog Code:

```
module clk_divider(clk, clkout);
input clk;
reg[26:0] second_counter;
output reg clkout;

always@(posedge clk)
  begin
    if(second_counter == 49_999_999)
    begin
        second_counter <= 0;
        clkout <= ~clkout;
    end
    else begin
        second_counter <= second_counter + 1;
    end
end
endmodule
```

Refresh Rate Seven Segment Display Verilog Code:

```
 always@(posedge clk)
  begin
    refresh_rate <= refresh_rate + 1;
  end
assign anode_counter = refresh_rate[19:17];
```

The errors that I had to handle were the same for the last lab which include the following:

```
## Note: As the Nexys 4 DDR was rebranded to the Nexys A7 with no substa

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets set_IBUF];
set_property SEVERITY {Warning} [get_drc_checks NSTD-1];
set_property SEVERITY {Warning} [get_drc_checks UCIO-1];
```

The handling of the seven segment displays was very similar to the last lab as well.

Seven Segment Display Multiplexing Verilog Code:

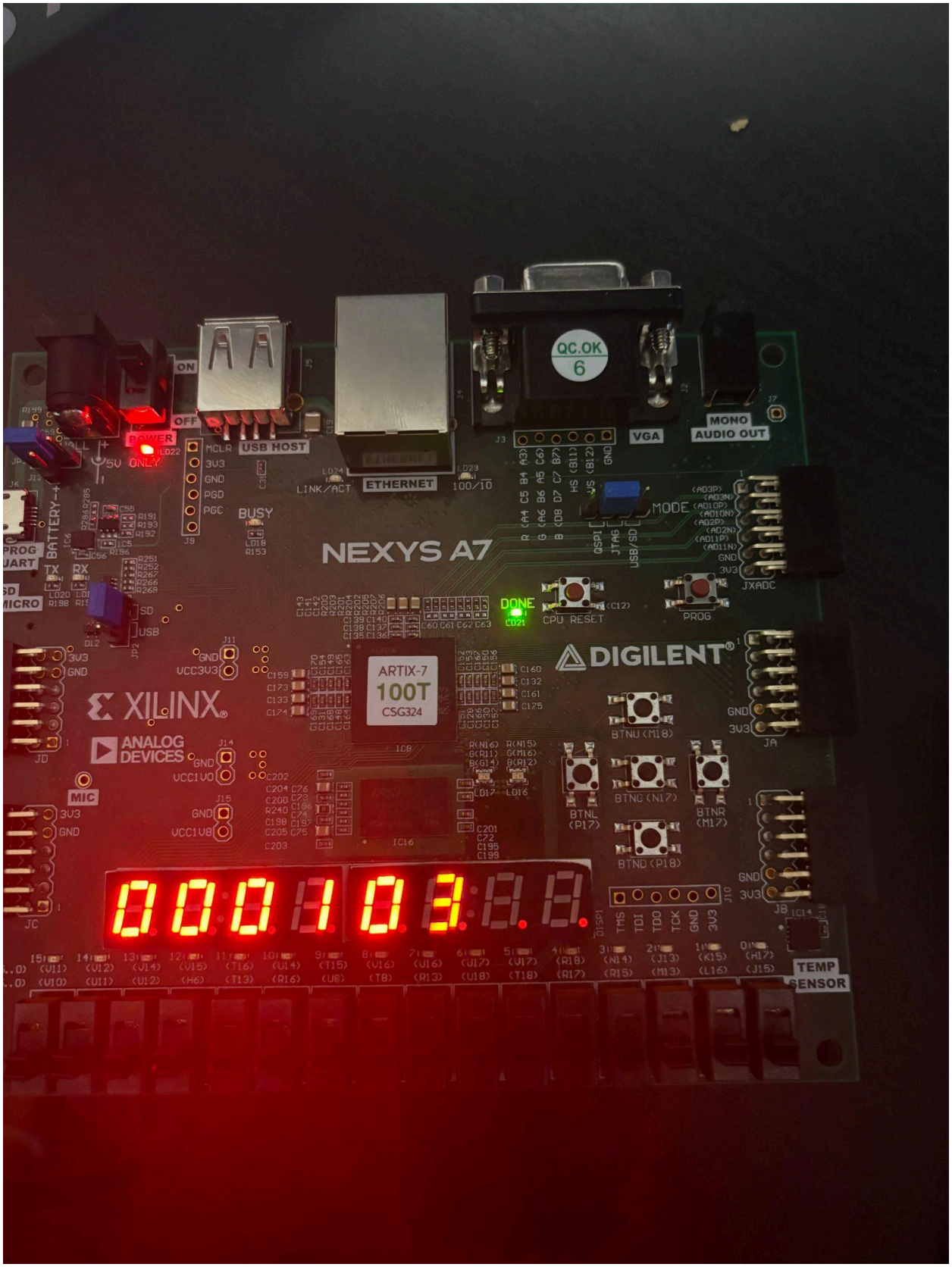
```
always@(posedge clk) begin
  case(anode_counter)
    3'b000: begin
        SA <= h1_out[0];
        SB <= h1_out[1];
        SC <= h1_out[2];
        SD <= h1_out[3];
        SE <= h1_out[4];
        SF <= h1_out[5];
        SG <= h1_out[6];

        anode_result <= 8'b1111_1110;
    end
    3'b001: begin
        SA <= h10_out[0];
        SB <= h10_out[1];
        SC <= h10_out[2];
        SD <= h10_out[3];
        SE <= h10_out[4];
        SF <= h10_out[5];
        SG <= h10_out[6];

        anode_result <= 8'b1111_1101;
    end
```

```
3'b010: begin
    SA <= m10_out[0];
    SB <= m10_out[1];
    SC <= m10_out[2];
    SD <= m10_out[3];
    SE <= m10_out[4];
    SF <= m10_out[5];
    SG <= m10_out[6];

    anode_result <= 8'b1111_1011;
end
3'b011: begin
    SA <= m1_out[0];
    SB <= m1_out[1];
    SC <= m1_out[2];
    SD <= m1_out[3];
    SE <= m1_out[4];
    SF <= m1_out[5];
    SG <= m1_out[6];

    anode_result <= 8'b1111_0111;
end
3'b100: begin
    SA <= s10_out[0];
    SB <= s10_out[1];
    SC <= s10_out[2];
    SD <= s10_out[3];
    SE <= s10_out[4];
    SF <= s10_out[5];
    SG <= s10_out[6];

    anode_result <= 8'b1110_1111;
end
```

```
        SA <= 1'b1;
        SB <= 1'b1;
        SC <= 1'b1;
        SD <= 1'b1;
        SE <= 1'b1;
        SF <= 1'b1;
        SG <= 1'b1;

        anode_result <= 8'b1011_1111;
    end
    3'b111: begin
        SA <= 1'b1;
        SB <= 1'b1;
        SC <= 1'b1;
        SD <= 1'b1;
        SE <= 1'b1;
        SF <= 1'b1;
        SG <= 1'b1;

        anode_result <= 8'b0111_1111;
    end
    default: begin
        SA <= 1'b1;
        SB <= 1'b1;
        SC <= 1'b1;
        SD <= 1'b1;
        SE <= 1'b1;
        SF <= 1'b1;
        SG <= 1'b1;

        anode_result <= 8'b1111_1111;
    end
    endcase
```

After this I then the bitstream for the code was generated and implemented on the FPGA board.

Picture of the Clock Running (default state (24hr), no set time):


NEXYS A7
DIGILENT
XILINX
ANALOG DEVICES
ARTIX-7
100T
CSG324
ETHERNET
USB HOST
VGA
MONO AUDIO OUT
JXADC
CPU RESET
PROG
DONE
BTNU (M18)
BTNL (P17)
BTNC (N17)
BTNR (M17)
BTND (P18)
TEMP SENSOR
MIC
BUSY
LINK/ACT
100/10
MODE
POWER
5V ONLY
QC.OK
6

Conclusion

This lab serves as a bittersweet ending to the AMPM clock lab series. This lab was challenging to me due to the trouble of understanding the logic behind a 12 hour clock since there are many ways that you are able to approach it, but once I understood I was able to implement it in CircuitVerse. The Verilog portion of the lab was a lot easier since we implemented the 24 hour lab for the last lab and now I can apply the same concepts to this lab since it's very similar. Overall this lab series taught me more about Verilog and a lot about how FPGA boards function.

References

Nexys A7 100T Master Board Files:

https://github.com/Digilent/digilent-xdc/blob/master/Nexys-A7-100T-Master.xdc