Lab 9 – Datapath and Control for the 4-digit, 7-segment Displays

Name: Robert Bara

Section #: 003

Date: 3/26/2020

Summary/Abstract (10 points)

Lab 9 introduces the concept of Datapath's and Controllers by creating a basic Datapath that will appear to light up all four anodes at once, displaying hexadecimal values by using the svn_seg_decoder module built in Lab3.

Introduction (10 points)

The Datapath can be built by using a case statement, following the diagram found in the lab manual, which explains that after instantiating the divider module from Lab 7, it will create a 1kHz cycle for m_sec. When m_sec=0, state will hold the position of the current state, however when m_sec=1, state will go to the next state. Based on the controller, the reset will initialize the anode_sel and digit_sec to be 0 again, while looping state to go back to D0 and start the process over. By generating a 1kHz cycle which is equal to 1ms, the code will deliver the input from the switches to the anodes and cathodes through the svn_seg_decoder module at such a high speed, that it will appear as if all 4 anodes are on displaying something, but actually it is just 1 on at a time moving at such a high speed that it tricks our eyes. This entire lab in particular is extremely useful to most household objects, or really any logic that needs to display a digital clock or timer, because a similar datapath could be implemented with a counter to either count time up or down and reset when needed.

Procedure (15 points)

To begin the design, the diagram from the Basys3 manual explains how the frequency can be generated with m_sec:

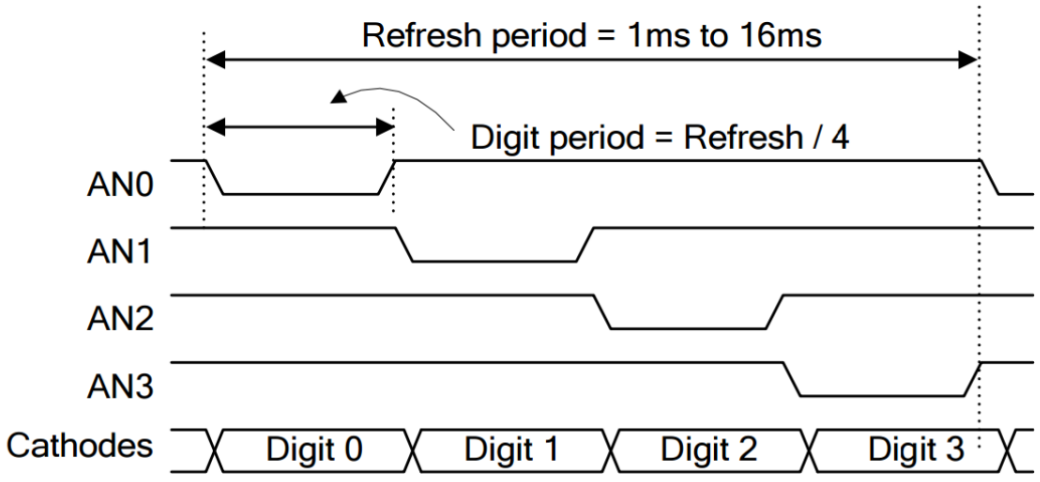
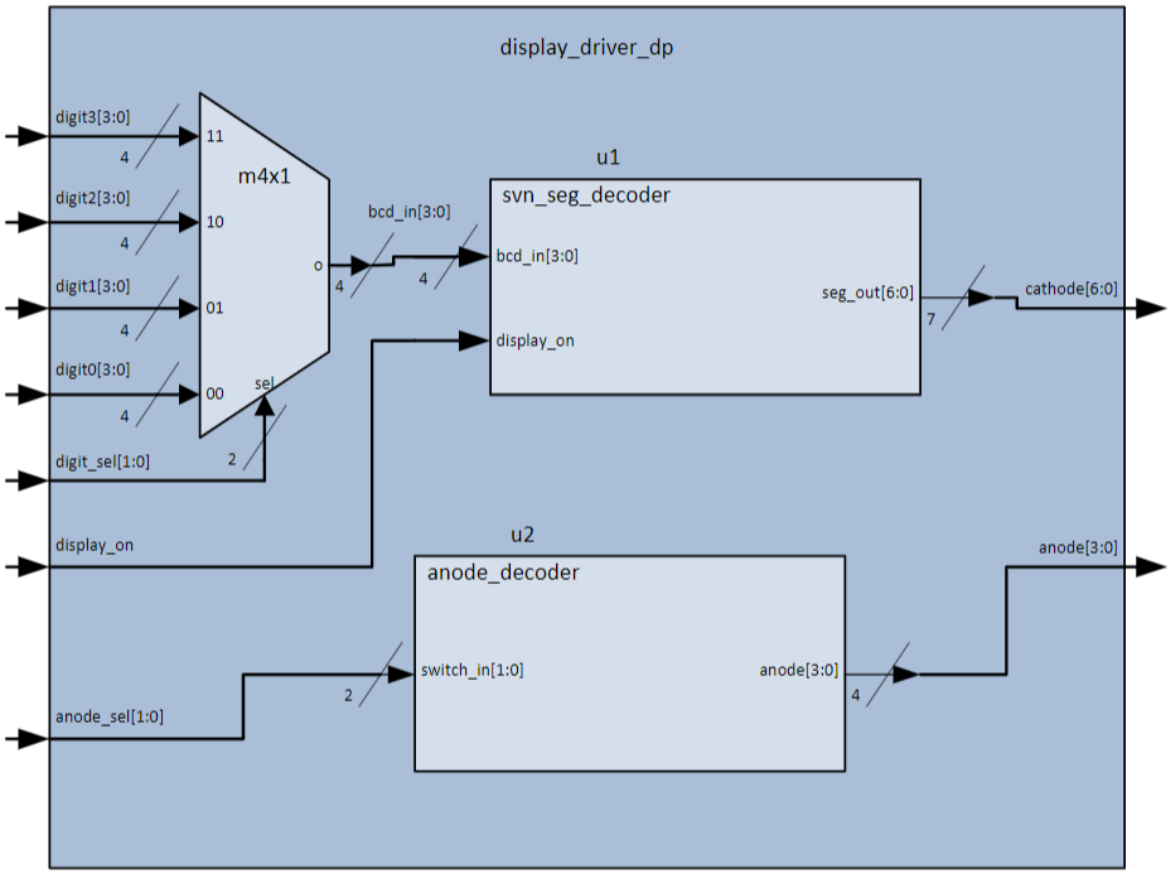


Figure 19. Four digit scanning display controller timing diagram.

The Lab 9 design will have a switching period of 1ms since $T = \frac{1}{F} = \frac{1}{1000Hz} = .001s = 1ms$ then there will be a refresh period of 4ms. This can be done in Verilog by instantiating the divider module from Lab

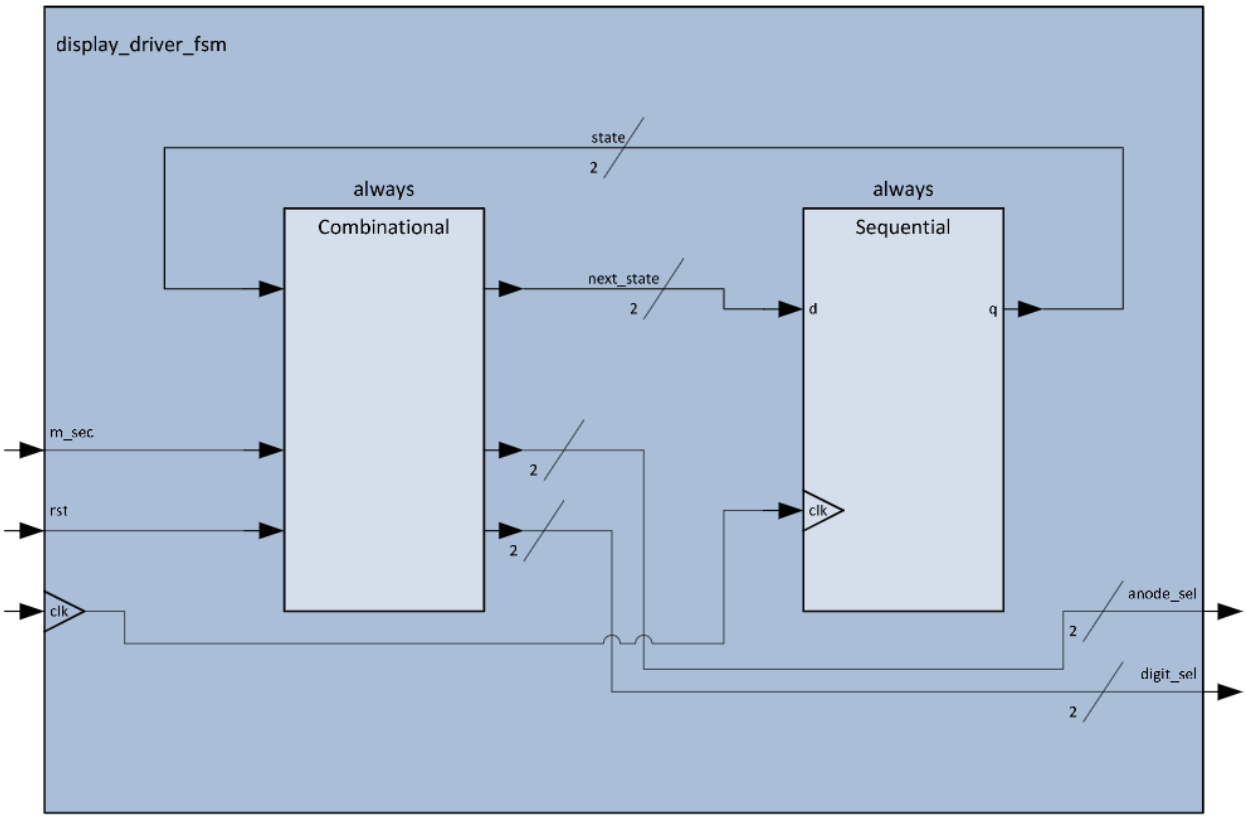
7 to generate this in the display_driver.sv module. To begin the Datapath and Control blocks, I instantiated the svn_seg_decoder and anode_decoder into the Datapath module, display_driver_dp.sv, the instantiation can be called from the following block diagram:



[Figure 1a: Display Driver Datapath Block Diagram]

To complete this module, a 4x1 multiplexer needs to built, and an external wire for bcd_in needs to be defined. Using combinational logic, I created a case statement for the multiplexer since I believed this would be the simplest way to do this.

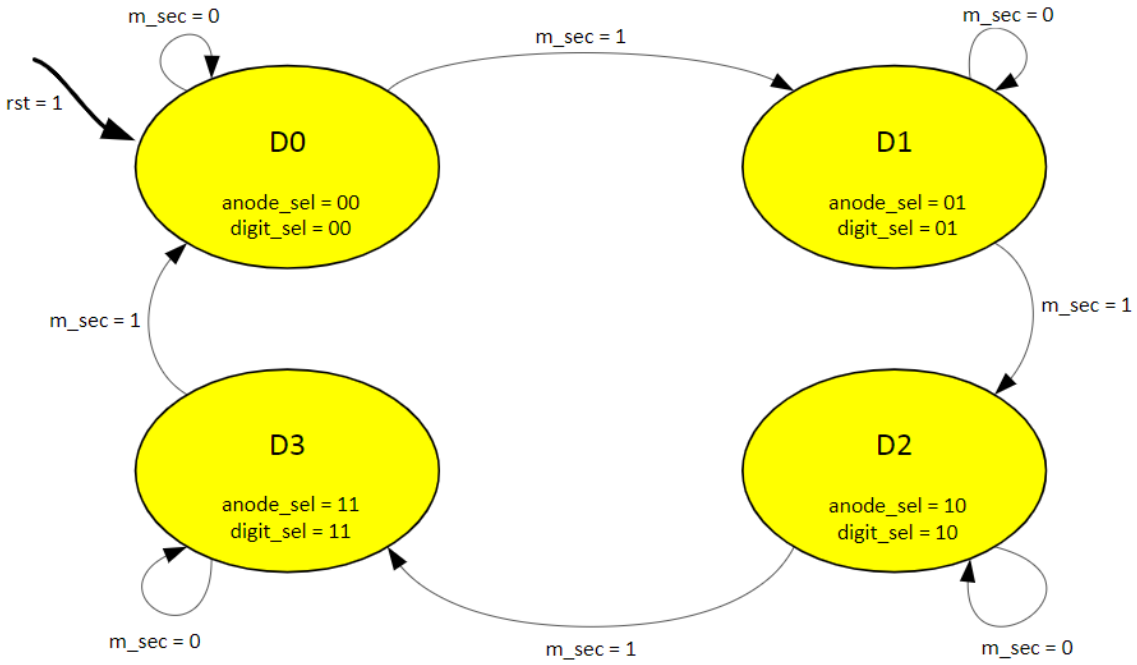
Moving on the Controller (Finite State Machine), the inputs and outputs are clearly defined in the Block Diagram:



[Figure 1b: Finite State Machine/Controller Block Diagram]

The sequential logic for this is simply an always block setting state to change upon the posedge clock signal. For the combinational logic, this state diagram can be implemented into a case statement, with the priority logic as an if statement (the reset which initializes everything and sets the anode to hold

D0's position and start over):



[Figure 1c: State Diagram for FSM]

Results (45 points – see sub-sections)

The simulation for this lab was fairly simple, after getting no mismatches, the .lxt2 file was put into the timing diagram (shown under simulation results), and clearly shows the corresponding anode and cathode signals matching the truth table from lab 3 that shows hexadecimal values by decoding a 4'b signal into a 7'b signal among the cathodes. Additionally, the m_sec correctly shows 1ms which is equal to 1kHz as described earlier. Therefore, the simulation was correct, and the hardware implementation proved to be a success.

Design Code (15 points)

display_driver_d × display_driver_fs × display_driver.sv × tb_display_driver ×

```
1  //
2  // lab9 : version 03/24/2020
3  // Robert Bara
4  `timescale 1ns / 1ps
5  //////////////////////////////////////////////////////////////////////////////////
6  //////////////////////////////////////////////////////////////////////////////////
7
8  module display_driver_dp(output logic [6:0] cathode, output logic [3:0] anode, input logic [3:0] digit3,
9      input logic [3:0] digit2, input logic [3:0] digit1, input logic [3:0] digit0,
10     input logic [1:0] digit_sel, input logic display_on, input logic [1:0] anode_sel);
11
12     logic [3:0] bcd_in;
13
14     svn_seg_decoder u1(.seg_out(cathode),.bcd_in(bcd_in),.display_on(display_on));
15     anode_decoder u2(.anode(anode),.switch_in(anode_sel));
16
17     always_comb begin
18         case(digit_sel)
19             2'b00: bcd_in=digit0;
20             2'b01: bcd_in=digit1;
21             2'b10: bcd_in=digit2;
22             default: bcd_in=digit3;
23         endcase
24     end
25 endmodule
26
```

[Figure 2a: display_driver_dp.sv Design Code]

display_driver_d | **display_driver_fs** | display_driver.sv | tb_display_driver

```
//
// lab9 : version 03/24/2020
// Robert Bara
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////

module display_driver_fsm(output logic [1:0] anode_sel, output logic [1:0] digit_sel,
    input logic clk, input logic rst, input logic m_sec);

    // define and enumerate state variables
    enum logic [1:0] {D0, D1, D2, D3} state, next_state;

    // sequential logic
    always_ff @(posedge clk) begin
        state <= next_state;
    end

    // combinational for state machine
    always_comb begin
// complete the combinational logic for the state machine
        // defaults

        // main logic
        case(state)
            D0: begin
                anode_sel=00;
                digit_sel=00;
                next_state=D1;
            end
            D1: begin
                anode_sel=01;
                digit_sel=01;
                next_state=D2;
            end
            D2: begin
                anode_sel=10;
                digit_sel=10;
                next_state=D3;
            end
            D3: begin
                anode_sel=11;
                digit_sel=11;
                next_state=D0;
            end
        endcase
        if(m_sec==1'b0)begin
            next_state=state;
        end
        // priority logic
        if (rst==1'b1)begin
            anode_sel=00;
            digit_sel=00;
            next_state=D0;
        end
    end // combinational state machine always

endmodule
```

[Figure 2b & 2c: display_driver_fsm.sv Design Code Part]

display_driver_dp × display_driver_fs × **display_driver.sv** × tb_display_driver ×

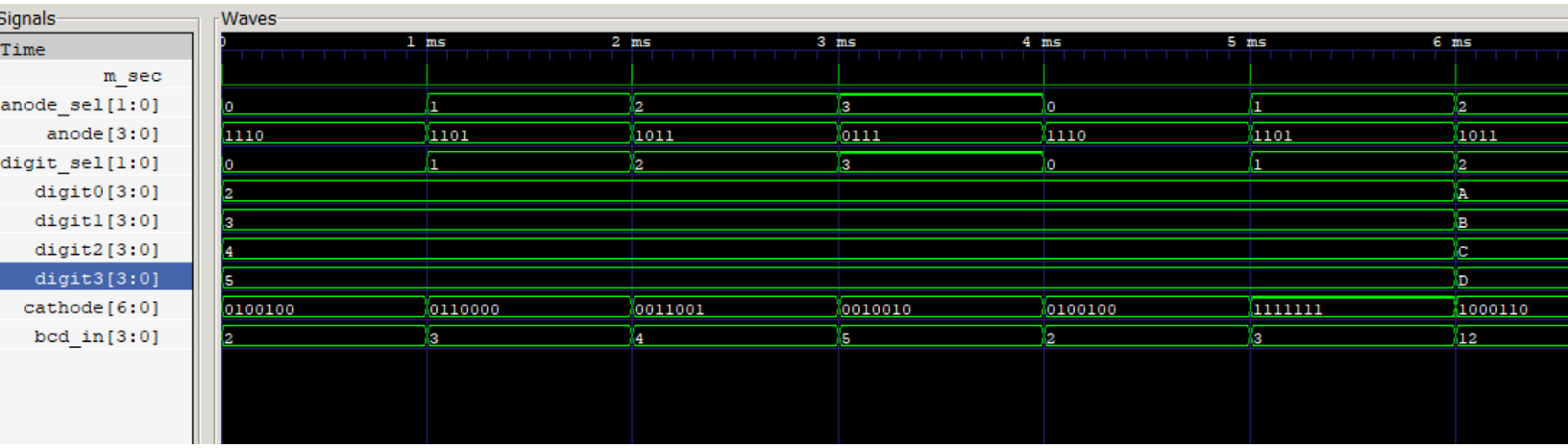
```
1  //
2  // lab9 : version 03/24/2020
3  //  Robert Bara
4  `timescale 1ns / 1ps
5  //////////////////////////////////////////////////////////////////////////////////
6  //////////////////////////////////////////////////////////////////////////////////
7
8  module display_driver( output logic [6:0] cathode, output logic [3:0] anode,
9      output logic [1:0] anode_sel, output logic m_sec,
10     input logic [3:0] digit3, input logic [3:0] digit2, input logic [3:0] digit1,
11     input logic [3:0] digit0, input logic display_on, input logic rst, input logic clk);
12
13     // insert your code here
14     logic [1:0] digit_sel;
15
16     display_driver_dp u1(.cathode(cathode), .anode(anode), .anode_sel(anode_sel), .digit_sel(digit_sel), .digit0(digit0), .digit1(digit1), .digit2(digit2), .digit3(digit3), .display_on);
17     display_driver_fsm u2(.anode_sel(anode_sel), .digit_sel(digit_sel), .m_sec(m_sec), .rst, .clk);
18     divider #(.BIT_SIZE(17)) u3(.tc(m_sec), .rst, .clk, .ena(1'b1), .init_count(17'd99999));
19
20 endmodule
21
```

[Figure 2d: display_driver.sv Design Code]

Simulation Results (15 points)

```
Simulation complete - no mismatches!!!
$finish called at time : 10 ms : File "/home/tuj22026/2613_2020s/lab9/tb_display_driver.sv" Line 105
run: Time (s): cpu = 00:00:00.02 ; elapsed = 00:00:09 . Memory (MB): peak = 1359.289 ; gain = 0.000 ; free physical = 4619 ;
## exit
```

[Figure 3a: Simulation Results]



[Figure 3b: Timing Diagarm]

Hardware Implementation (10 points)

My design was checked off by Yamin during the lab period of Thursday 3/26/20 at 1:32pm.

Conclusion (10 points)

This lab introduced the idea of Datapaths and Finite State Machines in a way that can we created modules that could be reused for real world applications such as creating a clock or timer. The Datapath was completed by instantiating previous lab modules to build off of each other, as well as use some

combinational logic to create a multiplexer. The Finite State Machine was interpreted from the block diagram and state diagram and using sequential and combinational logic I was able to create and instantiate the Datapath and Controller. My simulation was a success after examining the truth tables and frequency, and my hardware implementation proved to be a success.