Lab 6 Report

Name: Christian Han
UT EID: cjh3752
Section: 15300

Design Process

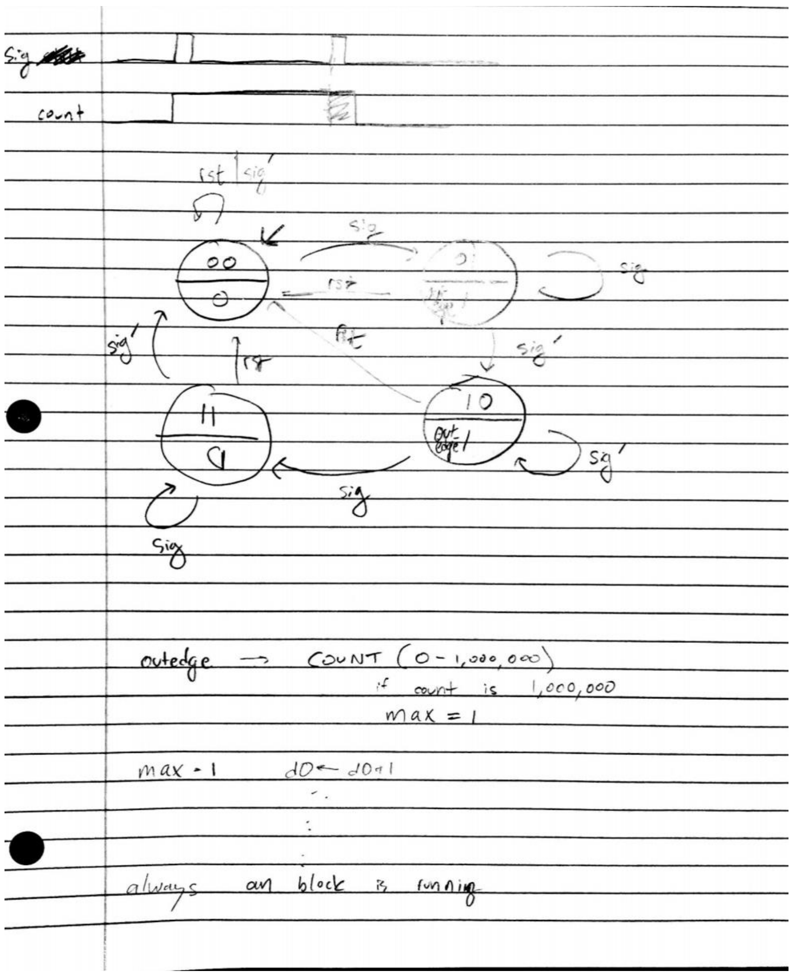
In Lab 6, we were instructed to program a stopwatch/timer design that would be implemented using RTL-design. The design had two main modules: a stopwatch (to count up from 00.00s) and a timer (to count down from 99.99s). I approached the design by using a divide and conquer method. I realized that I had to first create a stopwatch before starting on the timer.

First, I had to figure out a way to accurately get the two LSB digits to display time with a resolution of 10ms. I remember in lab 4 we used a clock divider to accurately get a resolution of 1s. I used the same principle to create a clock divider with a count register (20 bits wide) to count up at every positive edge of the clock of the board running at 100Mhz; creating a signal to refresh the display every 10ms.

I used this signal to drive a function that would start the counting up the 4bit wide registers each (pre loaded with 0) from 0-9. This signal was also used to create the 7-segment multiplexer to enable the [3:0] an signals at a resolution of 10ms (tricking the eye into thinking all the signals are on simultaneously). I reused the hexto7segment module from lab 4 to allow the bits to drive the 7-segment display.

I ended up getting the stopwatch to work but ran into a problem with debouncing the button. The stopwatch would only count up when I held down the start button and stop when I released it. I came up with a FSM that would debounce the button. This part was one of the more challenging parts of the lab. I had to incorporate an outedge signal from the debouncing button FSM that would start the count up from 0-1,000,000 in the 20bit count register.

I reused a lot of the code from lab 4 and the stopwatch to create the other 3 modules easily (only changing some of the logic). The major problem in the lab was creating a topmodule to incorporate all the modules using switches to select the different modes (modules). I realized the board can only accept one output of (an, dp, sseg) at a time and the 4 modules were all outputting values at the same time. I created a mux for each an, dp, sseg signal and had an always block of code using the switches as the select line to output one set of (an,dp,sseg) signals to output to the board. I learned a lot during this lab and felt like all the previous labs helped build up to lab 6.



outedge → COUNT (0-1,000,000)
if count is 1,000,000
max = 1

max = 1 d0 ← d0+1
:
:
:

always an block is running

Verilog Code

```
`timescale 1ns / 1ps


module topmodule(
input clk,
input reset,
input start,
input [15:0] sw,

output [3:0] an,
output dp,
output reg [6:0] sseg
);

wire [6:0] sseg1, sseg2, sseg3, sseg4; //wires to temporarily store each sseg value from
each module
wire [3:0] an1, an2, an3, an4;//wires "                        " an value from each module
wire dp1, dp2, dp3, dp4;//      wires "                       " dp value from each module
reg [19:0] COUNT;//to hold 1,000,000
wire max;// flag used to measure 10ms
wire [6:0] in0, in1, in2, in3;//wire used to store encoded inputs from hexto7seg
reg outedge;// flag for button

reg [1:0] state;
reg [1:0] next_state;

always @ (*) begin// this always block is a mux and its control signal is the switches to
select between the modes
            //  with each input being the sseg output from each module
  case (sw[15:14]) //Decoder
    2'b00: sseg = sseg1;
    2'b01: sseg = sseg2;
    2'b10: sseg = sseg3;
    2'b11: sseg = sseg4;
  endcase
end

always @ (*) begin// this always block is used to debounce the start button. it creates an
outedge signal when the button is pressed
            // the outedge is used as a flag to start the count register in the clock divider

case (state)

  2'b00: begin
    outedge = 1'b0;
    if (~start)
      next_state = 2'b00;
    else if (start)
      next_state = 2'b01;
    else if (reset)
      next_state = 2'b00;
    end

  2'b01: begin
    outedge = 1'b1;
    if (start)
      next_state = 2'b01;
    else if (~start)
      next_state = 2'b10;
    else if (reset)
      next_state = 2'b00;
    end

  2'b10: begin //insert code
    outedge = 1'b1;
    if (~start)
      next_state = 2'b10;
```

```
        else if (start)
           next_state = 2'b11;
        else if (reset)
              next_state = 2'b00;
        end

    2'b11: begin
        outedge = 1'b0;
        if (~start)
           next_state = 2'b00;
        else if (start)
           next_state = 2'b11;
        else if (reset)
              next_state = 2'b00;
        end

   default: begin
           next_state = 2'b00;
           outedge = 1'b0;
          end

endcase
end

always @ (posedge clk or posedge reset) begin // this always block drives the button
debouncing FSM continuously at every posedge clk
   if (reset)
      state <= 2'b00;
   else
      state <= next_state;
end

always @ (posedge clk or posedge reset) // this always block is running to start the
COUNT up whenever start is pressed
   begin
      if (reset)// resets the counter to 0
         COUNT <= 0;
      else if (COUNT == 1000000)//10 milisecond counter
         COUNT <= 0;
      else if (outedge)// outedge == 1 means button is pressed
         COUNT <= COUNT + 1;
   end

assign max = (COUNT == 1000000)? 1'b1 : 1'b0;//sets max to 1 every 10 milisecond

   reg [3:0] d0, d1, d2, d3;//data registers used for each digit in stop watch
   reg flag;// flag used to stop the stopwatch/timer
```

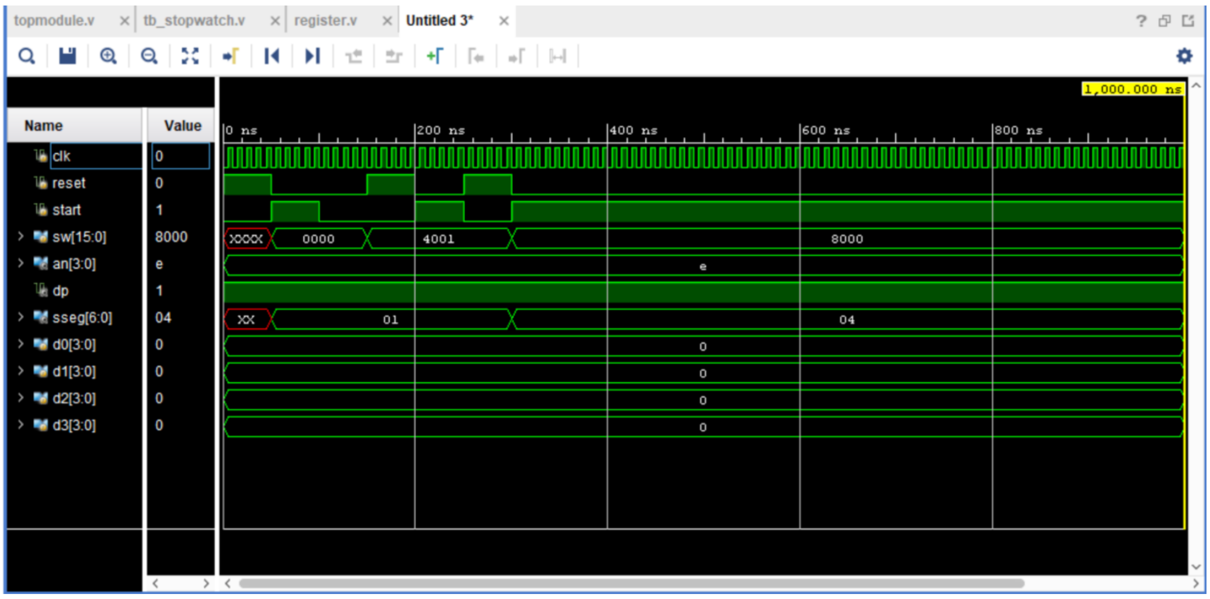
```
    always @ (posedge clk or posedge reset) // this always block triggers whenever start is
pressed.
                                            //It updates each digit register whenever the max flag is set
whenever COUNT reaches 1,000,000 (signaling 10ms has passed)
        begin
        if (reset)
            begin
                flag <= 0;
                d0 <= 0;
                d1 <= 0;
                d2 <= 0;
                d3 <= 0;
            end
        else if (flag)
            begin
                d0 <= 9;
                d1 <= 9;
                d2 <= 9;
                d3 <= 9;
            end
        else if (max)//increment every 10 milisecond
            if (~flag)

            begin
              if (d0 == 9)
              begin
                d0 <= 0;
                if (d1 == 9)
                begin
                  d1 <= 0;
                  if(d2 == 9)
                  begin
                    d2 <= 0;
                    if(d3 == 9)
                     begin
                      flag <= 1;
                     end
                    else
                      d3 <= d3 + 1;
                  end
                  else
                    d2 <= d2 + 1;
                end
                else
                  d1 <= d1 + 1;
              end
              else
            d0 <= d0 + 1;
          end
        end
    hexto7seg c0 (.x(d0), .r(in0));//instantiate each of the digit registers to send as an input
to the 7-segment display
    hexto7seg c1 (.x(d1), .r(in1));
    hexto7seg c2 (.x(d2), .r(in2));
    hexto7seg c3 (.x(d3), .r(in3));

    reg [19:0] COUNT2;
    reg [3:0] an_t;
    reg dp_t;

    always @ (posedge clk or posedge reset) begin
        if (reset)
        COUNT2 <= 0;
        else
        COUNT2 <= COUNT2 + 1;
        end

    always @ (*) begin
        case (COUNT2[19:18]) //Multiplexer for the an and dp signals to update every 10ms
to think all the lights are on simultaneously.
            2'b00: begin
                //sseg = in0;
                an_t = 4'b1110;
                dp_t = 1'b1;
               end
            2'b01: begin
                //sseg = in1;
                an_t = 4'b1101;
                dp_t = 1'b1;
               end
            2'b10: begin
                //sseg = in2;
                an_t = 4'b1011;
                dp_t = 1'b0;
               end
            2'b11: begin
                //sseg = in3;
                an_t = 4'b0111;
                dp_t = 1'b1;
               end
    endcase

    end
    assign an = an_t;
```

```
   assign dp = dp_t;


stopwatch a1 ( // instantiate each module
.clk(clk),
.reset(reset),
.start(start),
.sw(sw),
.an(an1),
.dp(dp1),
.sseg(sseg1)
);

my_stopwatch a2 (
.clk(clk),
.reset(reset),
.start(start),
.sw(sw),
.an(an2),
.dp(dp2),
.sseg(sseg2)
);

timer a3 (
.clk(clk),
.reset(reset),
.start(start),
.sw(sw),
.an(an3),
.dp(dp3),
.sseg(sseg3)
);

my_timer a4 (
.clk(clk),
.reset(reset),
.start(start),
.sw(sw),
.an(an4),
.dp(dp4),
.sseg(sseg4)
);


Endmodule

`timescale 1ns / 1ps
```

```
module hexto7seg(
input [3:0] x, //we need to make x = each data register
output reg [6:0] r
   );

always @ (*)
      case(x)
      4'b0000 : r = 7'b0000001;
      4'b0001 : r = 7'b1001111;
      4'b0010 : r = 7'b0010010;
      4'b0011 : r = 7'b0000110;
      4'b0100 : r = 7'b1001100;
      4'b0101 : r = 7'b0100100;
      4'b0110 : r = 7'b0100000;
      4'b0111 : r = 7'b0001111;
      4'b1000 : r = 7'b0000000;
      4'b1001 : r = 7'b0000100;
//      4'b1010 : r = 7'b0000100;
//      4'b1011 : r = 7'b0000100;
//      4'b1100 : r = 7'b0000100;
//      4'b1101 : r = 7'b0000100;
//      4'b1110 : r = 7'b0000100;
//      4'b1111 : r = 7'b0000100;
      endcase
endmodule
```

```
# Clock signal
set_property PACKAGE_PIN W5 [get_ports {clk}]

          set_property IOSTANDARD LVCMOS33 [get_ports {clk}]
          create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports
{clk}]

## Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
          set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
```

```
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
set_property PACKAGE_PIN T1 [get_ports {sw[14]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[14]}]
set_property PACKAGE_PIN R2 [get_ports {sw[15]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[15]}]

set_property PACKAGE_PIN W7 [get_ports {sseg[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sseg[6]}]
set_property PACKAGE_PIN W6 [get_ports {sseg[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sseg[5]}]
set_property PACKAGE_PIN U8 [get_ports {sseg[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sseg[4]}]
set_property PACKAGE_PIN V8 [get_ports {sseg[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sseg[3]}]
set_property PACKAGE_PIN U5 [get_ports {sseg[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sseg[2]}]
set_property PACKAGE_PIN V5 [get_ports {sseg[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sseg[1]}]
set_property PACKAGE_PIN U7 [get_ports {sseg[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sseg[0]}]

set_property PACKAGE_PIN V7 [get_ports {dp}]

        set_property IOSTANDARD LVCMOS33 [get_ports {dp}]

set_property PACKAGE_PIN U2 [get_ports {an[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]


##Buttons
set_property PACKAGE_PIN U18 [get_ports {reset}]

        set_property IOSTANDARD LVCMOS33 [get_ports {reset}]
set_property PACKAGE_PIN T17 [get_ports {start}]
            set_property IOSTANDARD LVCMOS33 [get_ports {start}]
```



Name	Value
clk	0
reset	0
start	1
sw[15:0]	8000
an[3:0]	e
dp	1
sseg[6:0]	04
d0[3:0]	0
d1[3:0]	0
d2[3:0]	0
d3[3:0]	0