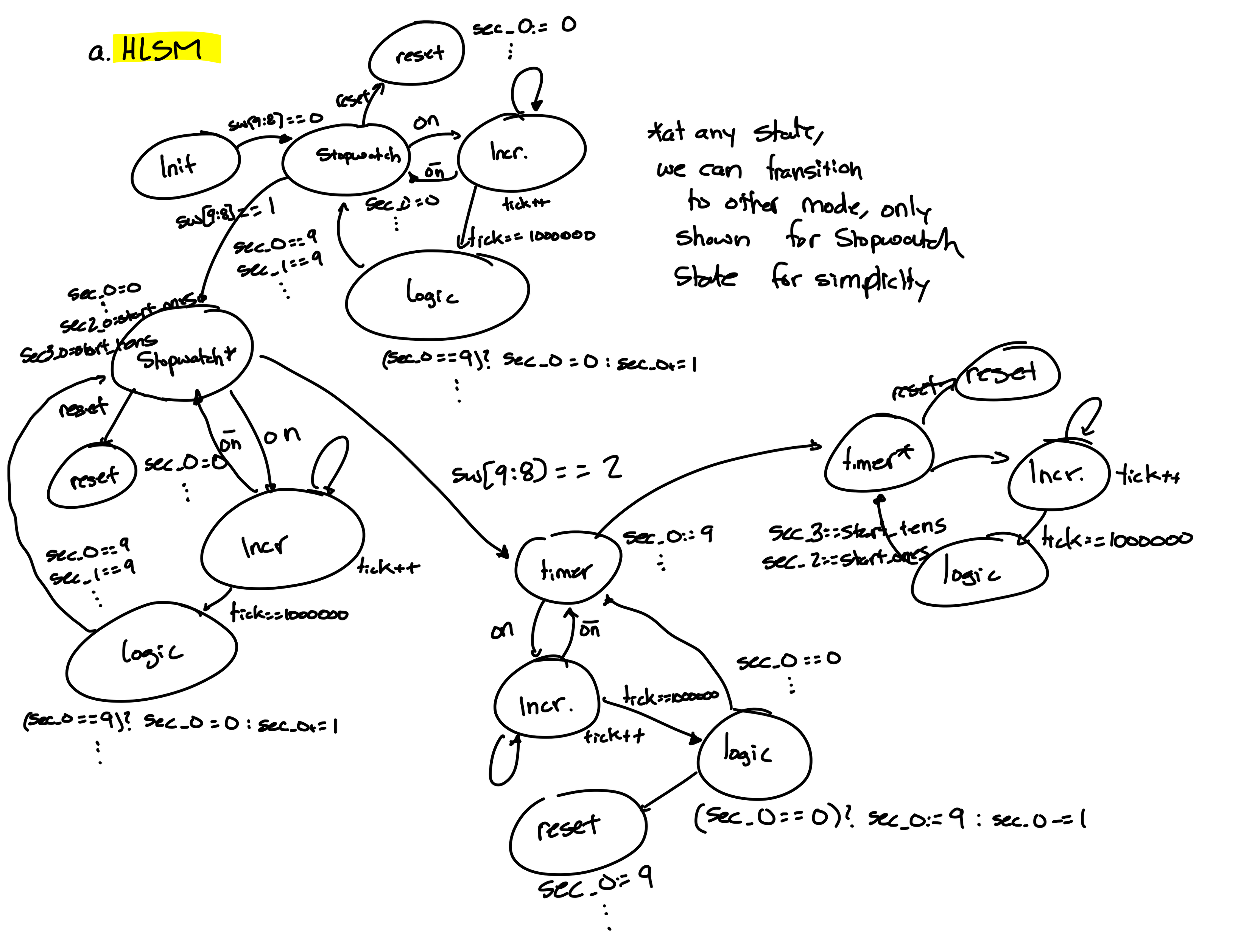
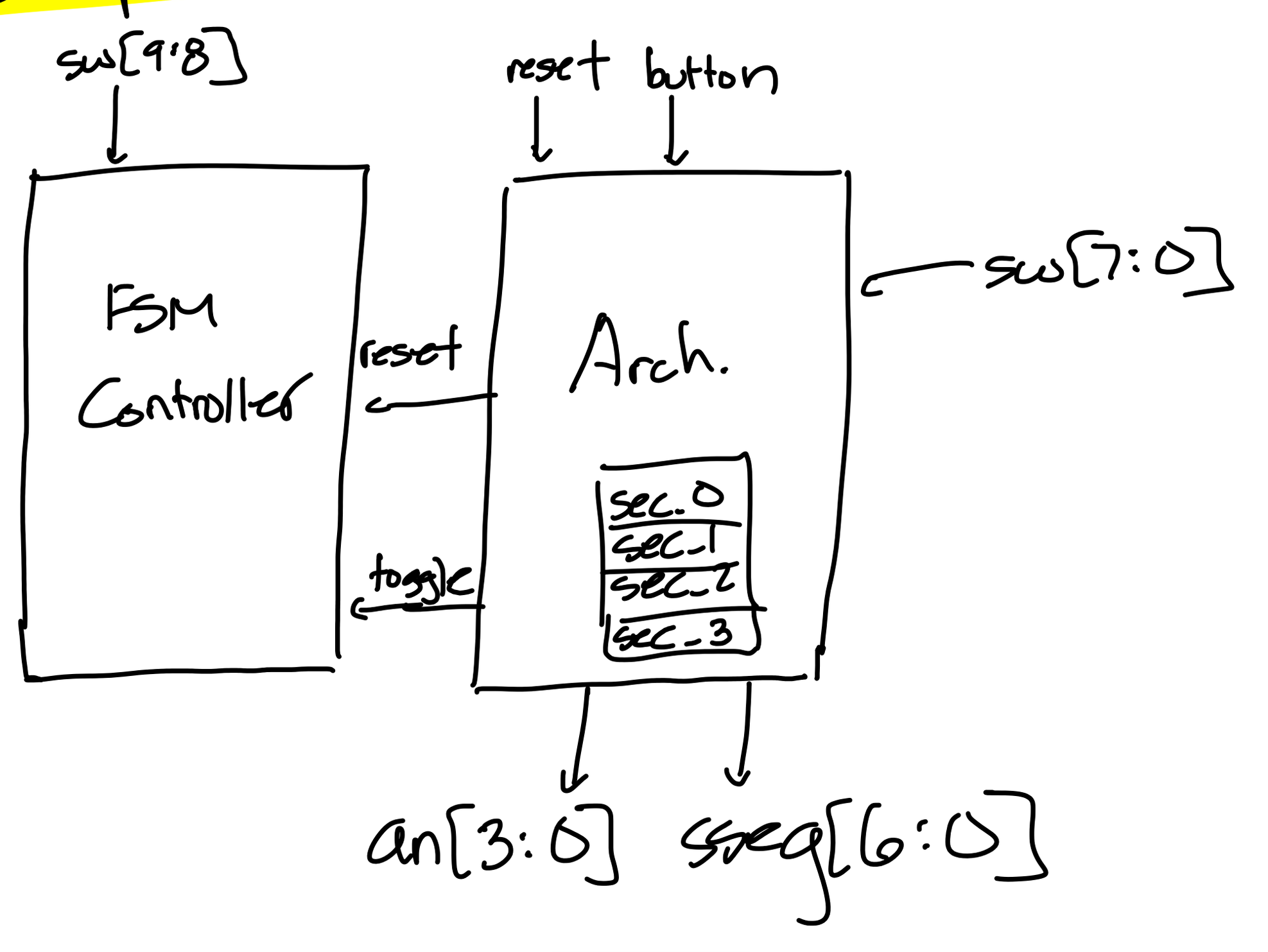
Lab 6

Tuesday, November 29, 2022 5:11 PM

a. HLSM



b. Datapath & Controller



c. Code

MAIN

```
`timescale 1ns / 1ps

module main(
   input clk,
   input reset,
   input toggle,
   input [9:0] sw, //[9:8] which mode, [7:4] tens digit, [3:0] ones digit
   output [6:0] sseg,
   output [3:0] an
   );

   reg [3:0] in0, in1, in2, in3;
   wire [3:0] s_in0, s_in1, s_in2, s_in3;
   wire [3:0] t_in0, t_in1, t_in2, t_in3;
   wire [6:0] out0, out1, out2, out3;
   wire slow_clk;
   wire button;

   initial in0 = 0;
   initial in0 = 1;
   initial in0 = 2;
   initial in0 = 3;

   //Module instantiation of clock divider
   clkdiv c5 (.clk(clk), .reset(rest), .clk_out(slow_clk));

   //Rising edge for button
   //rise_edge r0 (.clk(slow_clk), .signal(toggle), .reset(reset), .outedge(button));

   stopwatch s0 (.clk(clk), .reset(reset), .toggle(toggle), .mode_select(sw[9:8]), .start_tens(sw[7:4]), .start_ones(sw[3:0]),
              .sec_0(s_in0), .sec_1(s_in1), .sec_2(s_in2), .sec_3(s_in3));

   timer t0 (.clk(clk), .reset(reset), .toggle(toggle), .mode_select(sw[9:8]), .start_tens(sw[7:4]), .start_ones(sw[3:0]),
              .sec_0(t_in0), .sec_1(t_in1), .sec_2(t_in2), .sec_3(t_in3));

   always @(*) begin
   case(sw[9:8])
     2'b00: begin
         in0 = s_in0;
         in1 = s_in1;
         in2 = s_in2;
                 in3 = s_in3;
         end
     2'b01: begin
         in0 = s_in0;
         in1 = s_in1;
         in2 = s_in2;
         in3 = s_in3;
         end
     2'b10: begin
         in0 = t_in0;
         in1 = t_in1;
         in2 = t_in2;
         in3 = t_in3;
         end
     2'b11: begin
         in0 = t_in0;
         in1 = t_in1;
         in2 = t_in2;
         in3 = t_in3;
         end
     endcase

   end


   //Converts numbers calculated in timer or stopwatch to 7seg output
   hexto7segment c0 (.x(in0), .r(out0));
   hexto7segment c1 (.x(in1), .r(out1));
   hexto7segment c2 (.x(in2), .r(out2));
   hexto7segment c3 (.x(in3), .r(out3));



   time_mux_state_machine c6(
     .clk(slow_clk),
     .reset(reset),
     .out0(out0),
     .out1(out1),
     .out2(out2),
     .out3(out3),
     .an(an),
     .sseg(sseg));

endmodule
```

CLK_DIV

```
`timescale 1ns / 1ps

module clkdiv(
   input clk,
   input reset,
   output clk_out
);

   reg[15:0] COUNT;

   initial COUNT = 0;

   assign clk_out = COUNT[15];

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

endmodule
```

HEXTO7SEG

```
`timescale 1ns / 1ps

module hexto7segment(
   input [3:0] x,
   output reg [6:0] r
   );
   always @(*)
      case (x)
         4'b0000 : r = 7'b1000000;
         4'b0001 : r = 7'b1111001;
         4'b0010 : r = 7'b0100100;
         4'b0011 : r = 7'b0110000;
         4'b0100 : r = 7'b0011001;
         4'b0101 : r = 7'b0010010;
         4'b0110 : r = 7'b0000010;
         4'b0111 : r = 7'b1111000;
         4'b1000 : r = 7'b0000000;
         4'b1001 : r = 7'b0010000;
         default:  r = 7'b0010000;
      endcase
endmodule
```

TIME_MUX_STATE_MACHINE

```
`timescale 1ns / 1ps

module time_mux_state_machine(
   input clk,
   input reset,
   input [6:0] out0,
   input [6:0] out1,
   input [6:0] out2,
   input [6:0] out3,
   output reg [3:0] an,
   output reg [6:0] sseg
   );

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

   always @(*) begin
      case(state)        //State transition
         2'b00: next_state = 2'b01;
         2'b01: next_state = 2'b10;
         2'b10: next_state = 2'b11;
         2'b11: next_state = 2'b00;
      endcase
   end

   always @(*) begin
      case (state)       //Multiplexer
         2'b00: sseg = out0;
         2'b01: sseg = out1;
         2'b10: sseg = out2;
         2'b11: sseg = out3;
      endcase
   end

   always @(*) begin
      case (state)       //Decoder
         2'b00: an = 4'b1110;
         2'b01: an = 4'b1101;
         2'b10: an = 4'b1011;
         2'b11: an = 4'b0111;
      endcase
   end

   always @(posedge clk or posedge reset) begin
      if (reset)
         state <= 2'b00;
      else
         state <= next_state;
   end


endmodule
```

STOPWATCH

```
`timescale 1ns / 1ps

module stopwatch(
   input clk,
   input reset,
   input toggle,
   input [1:0] mode_select,
   input [3:0] start_tens, start_ones,
   output reg [3:0] sec_0, sec_1, sec_2, sec_3
   );

   reg [3:0] L_sec_0, L_sec_1, L_sec_2, L_sec_3;
   reg [23:0] tick;
   wire hs;
   reg on;

   //Initializing values
   initial on = 0;
   initial tick = 0;
   initial L_sec_0 = 4'b0000;
   initial L_sec_1 = 4'b0000;
   initial L_sec_2 = 4'b0000;
   initial L_sec_3 = 4'b0000;

  reg toggle_follow;

   //Handles counting of time
   always @ (posedge clk) begin
      toggle_follow <= toggle;
      if(mode_select == 0 || mode_select == 1) begin   //If user selects stopwatch (either from 0 or
loaded value)

         if(toggle && !toggle_follow)  //Handles the toggling of button
            on <= ~on;
         //Turn off counting and reset tick value
         if(reset) begin
            on <= 0;
            tick <= 0;
         end
         //If tick reaches value, 1 hundredth of a second has passed
         else if(tick == 1000000)
            tick <= 0;
         //If on, we should count
         else if(on)
            tick <= tick + 1;
      end
   end

   //1 hundredth of a second has passed
   assign hs = (tick == 1000000) ? 1'b1 : 1'b0;

   always @ (posedge clk) begin

      if(mode_select == 0 || mode_select == 1) begin
         if(reset) begin   //Sets all digits back to start value

            if(mode_select == 0) begin
               L_sec_0 <= 4'b0000;
               L_sec_1 <= 4'b0000;
               L_sec_2 <= 4'b0000;
               L_sec_3 <= 4'b0000;
            end
            else begin
               L_sec_0 <= 4'b0000;
               L_sec_1 <= 4'b0000;
               L_sec_2 <= start_ones;
               L_sec_3 <= start_tens;
            end
         end

//          else if(~on) begin    //While stopwatch not running, update values to reflect what user is
choosing
//             if(mode_select == 1) begin   //Mode 0 so starts from 0
//                L_sec_0 <= 4'b0000;
//                L_sec_1 <= 4'b0000;
//                L_sec_2 <= start_ones;
//                L_sec_3 <= start_tens;
//             end
//          end

         else if(hs) begin //increment timer
            if(L_sec_0 == 9) begin
               L_sec_0 <= 4'b0000; //xx.x9  overflows
               if(L_sec_1 == 9) begin
                  L_sec_1 <= 4'b0000; //xx.99 overflows
                  if(L_sec_2 == 9) begin
                     L_sec_2 <= 4'b0000; //x9.99 overflows
                     if(L_sec_3 == 9) begin
                        L_sec_3 <= 9; //stay at 99.99
                        L_sec_2 <= 9;
                        L_sec_1 <= 9;
                        L_sec_0 <= 9;
                     end
                     else
                        L_sec_3 <= L_sec_3 + 1;  //Not an overflow, add one as normal
                  end
                  else
                     L_sec_2 <= L_sec_2 + 1;  //Not an overflow, add one as normal
               end
               else
                  L_sec_1 <= L_sec_1 + 1;   //Not an overflow, add one as normal
            end
            else
               L_sec_0 <= L_sec_0 + 1;  //Not an overflow, add one as normal
         end

        sec_0 <= L_sec_0;
        sec_1 <= L_sec_1;
        sec_2 <= L_sec_2;
        sec_3 <= L_sec_3; //Write values to output
      end
   end


endmodule
```

TIMER

```
`timescale 1ns / 1ps

module timer(
   input clk,
   input reset,
   input toggle,
   input [1:0] mode_select,
   input [3:0] start_tens, start_ones,
   output reg [3:0] sec_0, sec_1, sec_2, sec_3
   );

   reg [3:0] L_sec_0, L_sec_1, L_sec_2, L_sec_3;
   reg [23:0] tick;
   wire hs;
   reg on;

   //Initializing values
   initial on = 0;
   initial tick = 0;
   initial L_sec_0 = 0;
   initial L_sec_1 = 0;
   initial L_sec_2 = 0;
   initial L_sec_3 = 0;

   reg toggle_follow;

   //Handles counting of time
   always @ (posedge clk)
   begin
      toggle_follow <= toggle;
      if(mode_select == 2 || mode_select == 3)    //If user selects stopwatch (either from 0 or loaded
value)
      begin
         if(toggle && !toggle_follow)  //Handles the toggling of button
            on <= ~on;

         if(reset) begin
            tick <= 0;
            on <= 0;
         end
         else if(tick == 1000000)
            tick <= 0;
         else if(on)
            tick <= tick + 1;

      end
   end

   //1 hundredth of a second has passed
   assign hs = (tick == 1000000) ? 1'b1 : 1'b0;

   always @ (posedge clk)
   begin
      if(mode_select == 2 || mode_select == 3) begin
         if(reset) begin  //Sets all digits back to start value

            if(mode_select == 2) begin
               L_sec_0 <= 4'd9;
               L_sec_1 <= 4'd9;
               L_sec_2 <= 4'd9;
               L_sec_3 <= 4'd9;
            end
            else begin
               L_sec_0 <= 4'd0;
               L_sec_1 <= 4'd0;
               L_sec_2 <= start_ones;
               L_sec_3 <= start_tens;
            end
         end

         else if(hs) begin //increment timer

            if(L_sec_0 == 0) begin
               //L_sec_0 <= 0; //xx.x0 -> xx.x9
               if(L_sec_1 == 0) begin
                  //L_sec_1 <= 0; //xx.99 overflows
                  if(L_sec_2 == 0) begin
                     //L_sec_2 <= 0; //x9.99 overflows
                     if(L_sec_3 == 0)
                        L_sec_3 <= 0; //stay at 00.00
                     else begin
                           L_sec_3 = L_sec_3 - 1;
                           L_sec_2 = 4'd9;
                           L_sec_1 = 4'd9;
                           L_sec_0 = 4'd9;
                     end
                  end
                  else begin
                     L_sec_2 = L_sec_2 - 1;
                     L_sec_1 = 4'd9;
                     L_sec_0 = 4'd9;
                   end
               end

               else begin
                  L_sec_1 = L_sec_1 - 1;
                  L_sec_0 = 4'd9;
               end
            end
            else
               L_sec_0 = L_sec_0 - 1;
         end
      sec_0 <= L_sec_0;
      sec_1 <= L_sec_1;
      sec_2 <= L_sec_2;
      sec_3 <= L_sec_3;

      end

   end

endmodule
```

CONSTRAINTS

```
## Clock signal - Uncomment if needed (will be used in future labs)
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 V2 [get_ports {sw[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]

##7 segment display
set_property PACKAGE_PIN W7 [get_ports {sseg[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sseg[0]}]
set_property PACKAGE_PIN W6 [get_ports {sseg[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sseg[1]}]
set_property PACKAGE_PIN U8 [get_ports {sseg[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sseg[2]}]
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[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sseg[4]}]
set_property PACKAGE_PIN V5 [get_ports {sseg[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sseg[5]}]
set_property PACKAGE_PIN U7 [get_ports {sseg[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sseg[6]}]

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 toggle]
set_property IOSTANDARD LVCMOS33 [get_ports toggle]
set_property PACKAGE_PIN T18 [get_ports reset]
set_property IOSTANDARD LVCMOS33 [get_ports reset]
```

Video Link

https://drive.google.com/file/d/1wJkSH2m-VbW19YC9bKBEIApKndfnThSe/view?usp=share_link