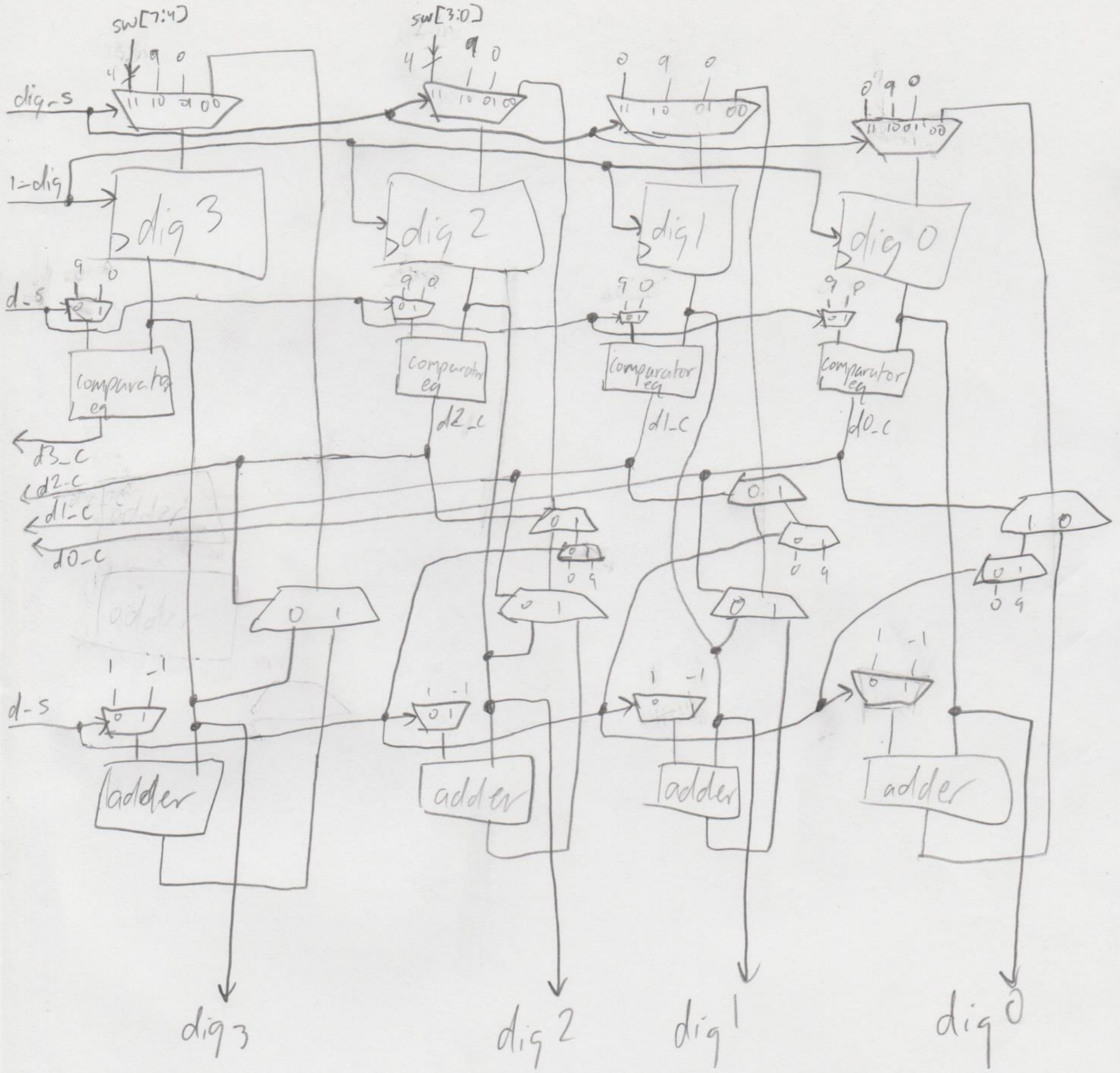
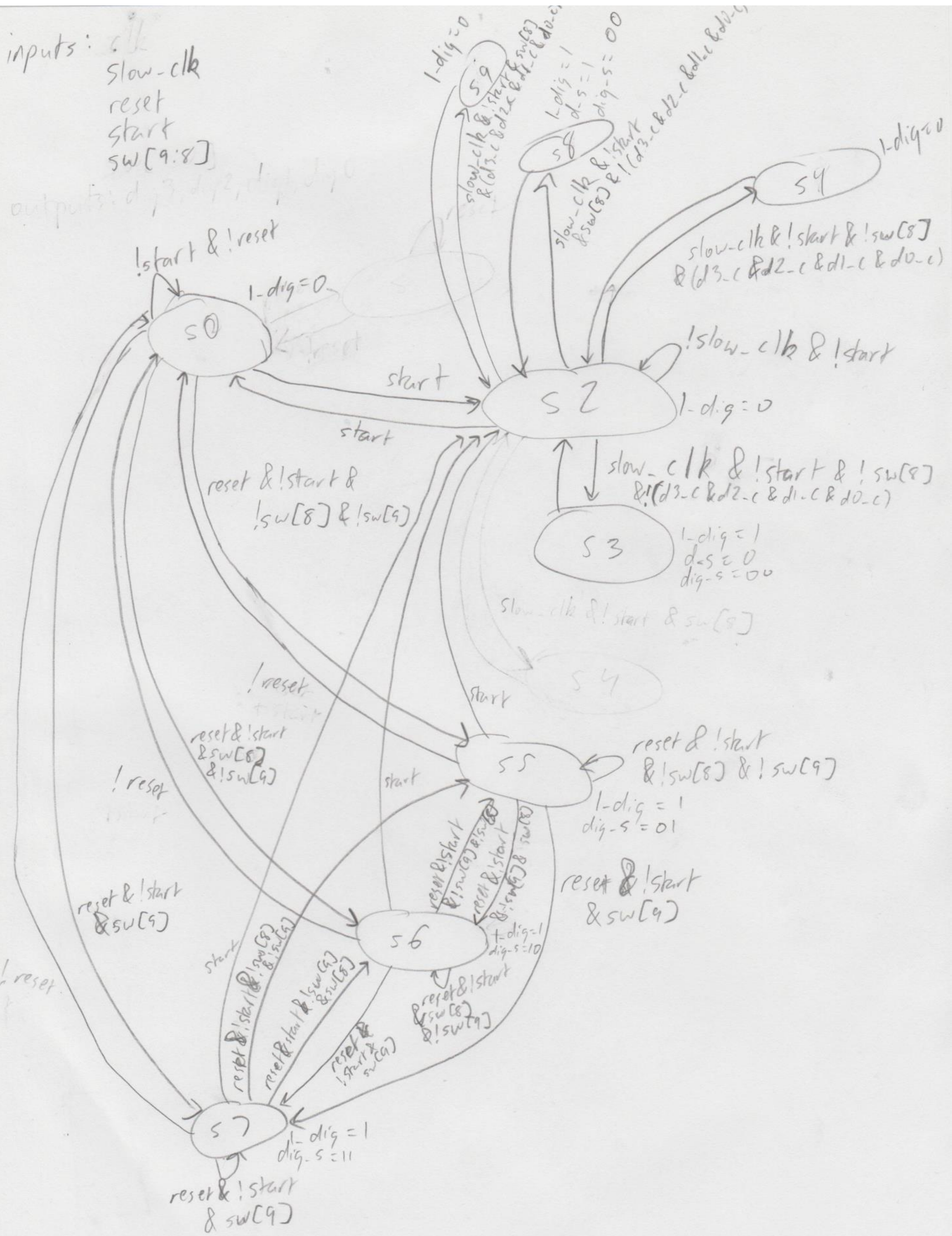
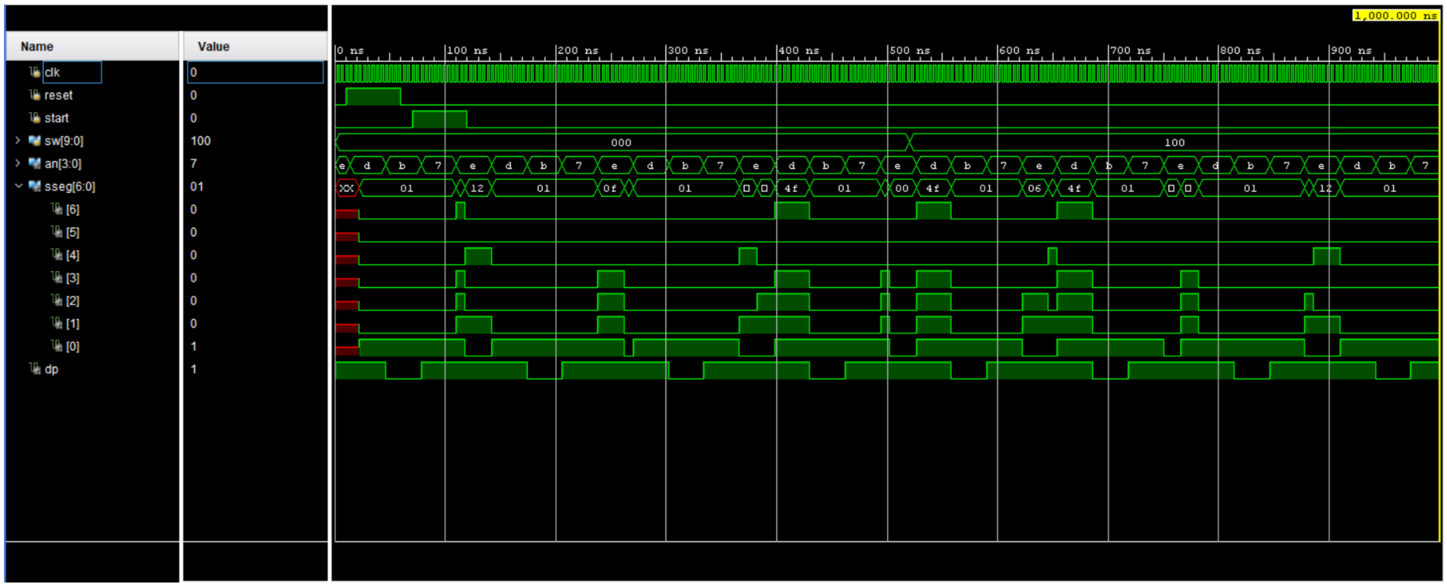
Noah Kessler

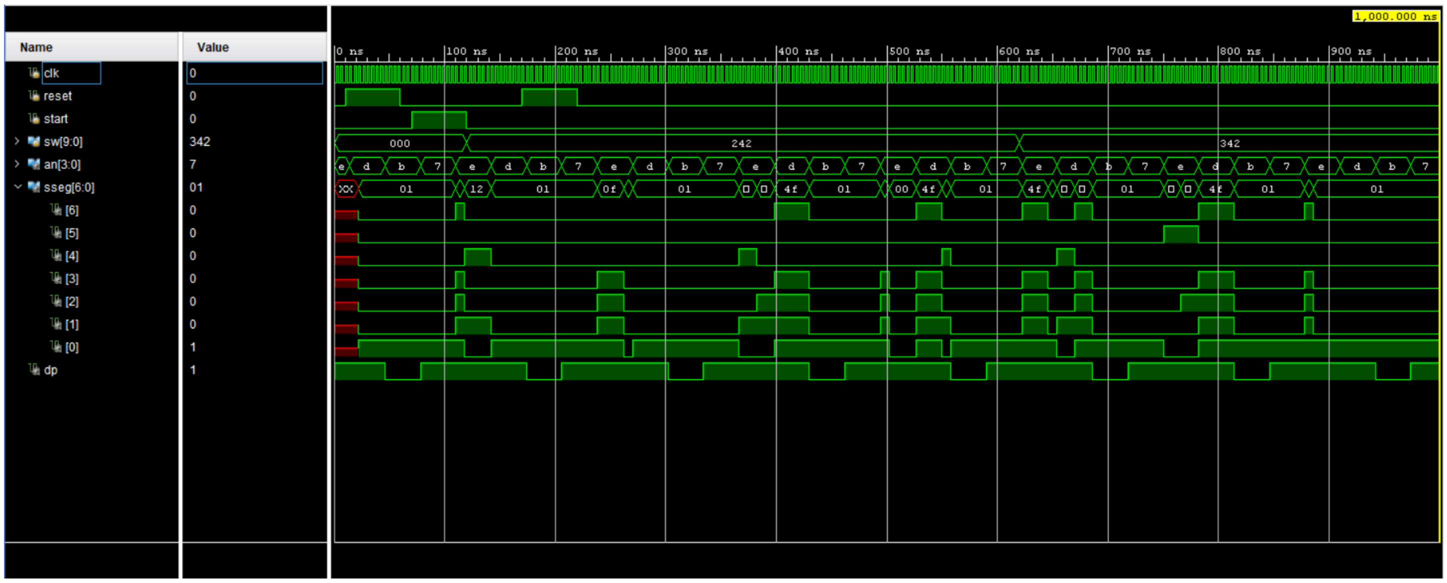
The goal of the design process was to make all the code simple and to use as much of the code from previous labs as possible. I wanted to have one module that would take the switch inputs and do all the reset logic and the counting both up and down. That way the only totally new code would be this module and everything else could be taken from old labs and modified to simplify the design process. I made this module work by splitting up the digits on the seven segments into their own registers before going through the counting logic. The counting works by first checking if the lowest digit is equal to nine. If not then it adds one, if it is then it sets it to zero and adds one to the next digit. If that is also nine, it sets that to zero and increments the next. If they are all equal to nine, then it leaves them all as nine. Counting down works the same except it checks if it is zero instead of nine. One of the main problems I had was getting the start button to work correctly. I initially just checked every cycle to see if the start button was pressed but that didn't work because it would change states every cycle as long as the button was held down. I resolved this issue by changing the button logic to a simple state machine with four states, two with the output high and two with the output low. If you push the button, the output is high, and it waits until the button is released before changing states. Then the output is still high until the button is pressed again, when it turns low. It then waits in the same state until the button is released again when it changes to the second state where the output is low and waits there until the button is pressed, when it starts the cycle over again. This way once the button is pressed the output remains the same until it is released and pressed again.


sw[7:4]
sw[3:0]
4
9
0
11 10 01 00
dig_s
1-dig
dig 3
dig 2
dig1
dig 0
d_s
0 1
comparator eq
d3_c
d2_c
d1_c
d0_c
1 -1
adder
dig 3
dig 2
dig 1
dig 0

inputs: clk
slow-clk
reset
start
sw[9:8]


s0
!start & !reset
l-dig=0
start
start
s2
!slow-clk & !start
l-dig=0
slow-clk & !start & !sw[8] & (d3-c & d2-c & d1-c & d0-c)
s4
l-dig=0
slow-clk & !start & sw[8] & !(d3-c & d2-c & d1-c & d0-c)
s8
l-dig=1
d-s=1
dig-s=00
slow-clk & !start & sw[8] & (d3-c & d2-c & d1-c & d0-c)
s9
l-dig=0
slow-clk & !start & !sw[8] & !(d3-c & d2-c & d1-c & d0-c)
s3
l-dig=1
d-s=0
dig-s=00
reset & !start & !sw[8] & !sw[9]
!reset
start
s5
reset & !start & !sw[8] & !sw[9]
l-dig=1
dig-s=01
reset & !start & sw[8] & !sw[9]
!reset
start
reset & !start & !sw[9] & !sw[8]
reset & !start & !sw[9] & sw[8]
reset & !start & sw[9]
s6
l-dig=1
dig-s=10
reset & !start & sw[8] & !sw[9]
reset & !start & sw[9]
start
reset & !start & !sw[8] & !sw[9]
reset & !start & !sw[9] & sw[8]
reset & !start & sw[9]
!reset
s7
l-dig=1
dig-s=11
reset & !start & sw[9]





```
module stopwatch_main(
   input clk,
   input reset,
   input start,
   input[9:0] sw,
   output[3:0] an,
   output[6:0] sseg,
   output dp
   );
```

```
wire[3:0] dig0, dig1, dig2, dig3;

wire time_clk;

wire display_clk;

wire[6:0] in0, in1, in2, in3;

wire go;


rising_edge_detector c1 (.clk(clk), .signal(start), .out(go));

clk_div_time c2 (.clk(clk), .clk_out(time_clk));

counter c3 (.clk(time_clk), .reset(reset), .go(go), .sw(sw), .in0(dig0), .in1(dig1), .in2(dig2), .in3(dig3), .out0(dig0),
.out1(dig1), .out2(dig2), .out3(dig3));


hex_to_7seg c4 (.x(dig0), .r(in0));

hex_to_7seg c5 (.x(dig1), .r(in1));

hex_to_7seg c6 (.x(dig2), .r(in2));

hex_to_7seg c7 (.x(dig3), .r(in3));


clk_div_disp c8 (.clk(clk), .clk_out(display_clk));


time_mux_state_machine c9 (.clk(display_clk), .in0(in0), .in1(in1), .in2(in2), .in3(in3), .dp(dp), .an(an), .sseg(sseg));


endmodule



module rising_edge_detector(

   input clk,

   input signal,

   output reg out

   );


reg[1:0] state;

reg[1:0] next_state;
```

```
always @(*) begin
    case(state)
        2'b00: begin
            out = 1'b0;
            if(~signal) next_state = 2'b00;
            else next_state = 2'b01;
            end
        2'b01: begin
            out = 1'b1;
            if(signal) next_state = 2'b01;
            else next_state = 2'b10;
            end
        2'b10: begin
            out = 1'b1;
            if(~signal) next_state = 2'b10;
            else next_state = 2'b11;
            end
        2'b11: begin
            out = 1'b0;
            if(signal) next_state = 2'b11;
            else next_state = 2'b00;
            end
        default: begin
            next_state = 2'b00;
            out = 1'b0;
            end
    endcase
end

always @(posedge clk) begin
    state <= next_state;
end
```

```
module clk_div_time(
   input clk,
   output reg clk_out
   );

reg[19:0] COUNT = 0;

always @(posedge clk) begin
   if(COUNT >= 1000000) begin
      clk_out = 1;
      COUNT = 0;
   end
   else begin
      COUNT = COUNT + 1;
      clk_out = 0;
   end
end

endmodule



module counter(
   input clk,
   input reset,
   input go,
   input[9:0] sw,
   input[3:0] in0,
   input[3:0] in1,
   input[3:0] in2,
   input[3:0] in3,
   output reg[3:0] out0,
   output reg[3:0] out1,
   output reg[3:0] out2,
   output reg[3:0] out3
   );

always @ (posedge clk) begin
   if(!go) begin
      if(reset) begin
          if(sw[9]) begin
              out0 = 0; out1 = 0; out2 = sw[3:0]; out3 = sw[7:4];
              if(out2 > 9) out2 = 0;
              if(out3 > 9) out3 = 0;
          end
          else if(sw[8]) begin
              out0 = 9; out1 = 9; out2 = 9; out3 = 9;
          end
          else begin
              out0 = 0; out1 = 0; out2 = 0; out3 = 0;
          end
      end
      else begin
          out0 = in0; out1 = in1; out2 = in2; out3 = in3;
      end
   end
   else begin
      if(!sw[8]) begin
          if((out3 == 4'd9) && (out2 == 4'd9) && (out1 == 4'd9) && (out0 == 4'd9)) begin
              out0 = in0; out1 = in1; out2 = in2; out3 = in3;
          end
          else begin
              if(out0 == 4'd9) begin
            out0 = 0;
            if(out1 == 4'd9) begin
               out1 = 0;
               if(out2 == 4'd9) begin
                  out2 = 0;
                  out3 = out3 + 1;
               end
               else out2 = out2 + 1;
            end
            else out1 = out1 + 1;
         end
         else out0 = out0 + 1;
      end
   end
   else begin
      if((out3 == 4'd0) && (out2 == 4'd0) && (out1 == 4'd0) && (out0 == 4'd0)) begin
         out0 = in0; out1 = in1; out2 = in2; out3 = in3;
      end
      else begin
         if(out0 == 4'd0) begin
            out0 = 9;
            if(out1 == 4'd0) begin
               out1 = 9;
               if(out2 == 4'd0) begin
                  out2 = 9;
                  out3 = out3 - 1;
               end
               else out2 = out2 - 1;
            end
            else out1 = out1 - 1;
         end
         else out0 = out0 - 1;
```

```
            end
        end
    end
end

endmodule



module hex_to_7seg(
    input[3:0] x,
    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'b0001100;
    endcase

endmodule



module clk_div_disp(
    input clk,
    output clk_out
    );

reg[15:0] COUNT = 0;

assign clk_out = COUNT[15];

always @(posedge clk) begin
    COUNT = COUNT + 1;
end

endmodule


module time_mux_state_machine(
    input clk,
    input [6:0] in0,
    input [6:0] in1,
    input [6:0] in2,
    input [6:0] in3,
    output reg dp,
    output reg[3:0] an,
    output reg[6:0] sseg
    );

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

always @(*) begin
    case(state)
        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)

        2'b00: begin sseg = in0; dp = 1; end

        2'b01: begin sseg = in1; dp = 1; end

        2'b10: begin sseg = in2; dp = 0; end

        2'b11: begin sseg = in3; dp = 1; end

    endcase

end


always @(*) begin

    case(state)

        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) begin

    state <= next_state;

end


endmodule



module tb_stopwatch_main;

reg clk;
reg reset;

reg start;

reg[9:0] sw;

wire[3:0] an;

wire[6:0] sseg;

wire dp;


stopwatch_main uut(

    .clk(clk),

    .reset(reset),

    .start(start),

    .sw(sw),

    .an(an),

    .sseg(sseg),

    .dp(dp)

);


initial begin


clk = 0;

sw[9:0] = 0;

reset = 0;

start = 0;


#10;


reset = 1;


#50;


reset = 0;
```

```
#10;

start = 1;

#50;

start = 0;

#400;

sw[8] = 1;

#400;

start = 1;

#50;

sw[9] = 1;
sw[8] = 0;
sw[7:4] = 4;
sw[3:0] = 2;
start = 0;

#50;

reset = 1;

#50;

reset = 0;
```

```
#400;

sw[8] = 1;

#400;

end



## 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 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[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 start]
            set_property IOSTANDARD LVCMOS33 [get_ports start]
set_property PACKAGE_PIN T18 [get_ports reset]
            set_property IOSTANDARD LVCMOS33 [get_ports reset]
```