Lab 6

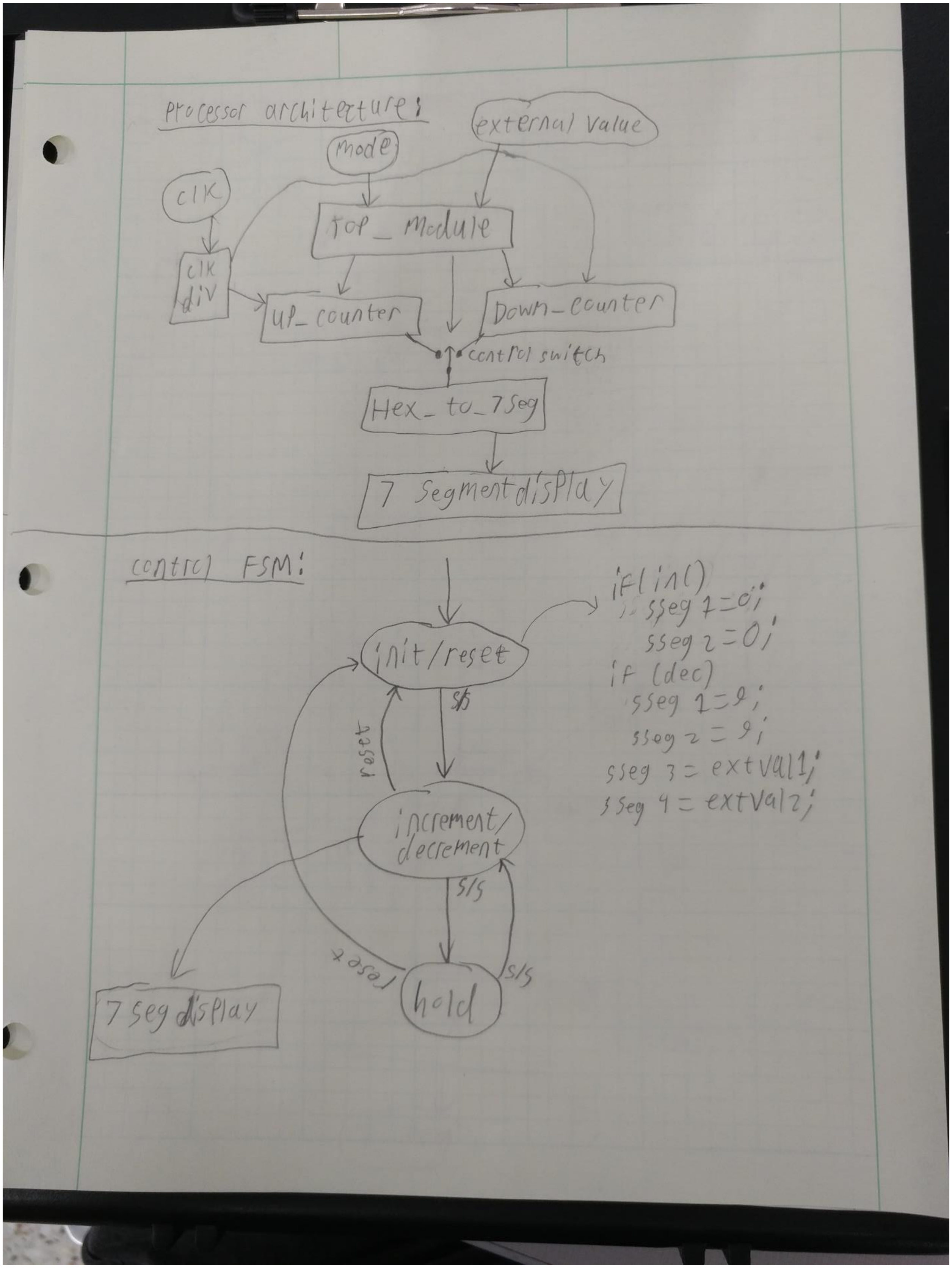
Donald Maze-England

dsm2588

Design Process:

When I first started working on lab 6 I had no idea where to start. We had not been given a lab like this before. Previously we were given most of the code we needed for our Verilog assignments, but this time we had to write all the code ourselves. Eventually I decided to try and design an FSM that would control the counting of the stop watch. This FSM would have 4 registers to keep track of each of the 4 digits on the seven segment display. The least significant bit register would be incremented by a clock at 100hz, and would overflow into the other 3 bits. The incrementing could be changed to decrementing when the mode switches were flipped on the board. This design had several issues. Mainly it was overly complex and very difficult to debug. Also, I ran into timing issues with this design. Eventually I scrapped this design and went for a cascading single digit counter module. When one would overflow it would cause the next higher bit to increment. This worked well and was fairly simple, however I could not find a good way to prevent it from rolling over back to zero once it counted up to 9999. I scrapped the single digit counter design in favor of a counter that would include all 4 digits in one module. This design used cascading if statements to control the incrementing and decrementing of the digits. Having all 4 digits in one module allowed me to check when all digits were at 9 and prevent the clock from rolling over back to 0. The problem now was that I also needed a decrementing counter. To fix this I made a separate decrementing counter that was almost identical to the incrementing counter but it decremented instead of incremented. Now I had two counters and one display, so I had to figure out a way to switch what the display would show depending on the mode. To do this I made the mode switches control which counter output (increment or decrement) was stored in several registers. Then I used these registers as the display vales. One of the major problems I faced in the lab was overcoming my urge to use Verilog like a normal coding language. When using behavioral code it is easy to forget that Verilog is a hardware specification language and has limitations beyond just syntax errors. Dealing with all the errors this caused took the majority of my time completing this lab.

Processor Architecture and Controller FSM:



```
if(inc)
  sseg 1=0;
  sseg 2=0;
if (dec)
  sseg 1=9;
  sseg 2=9;
sseg 3=extval1;
sseg 4=extval2;
```

Verilog Code:

Top module:

```
`timescale 1ns / 1ps


module top_module(
    input reset,
    input ss,
    input clk,
    input [1:0] mode,
    input [7:0] extVal,
    output wire [6:0] sseg,
    output wire [3:0] an,
    output wire dp
    );



    reg [3:0] ssegIntExt1 = 0;
    reg [3:0] ssegIntExt2 = 0;
    reg [3:0] extValReg1 = 0;
    reg [3:0] extValReg2 = 0;
    reg inc_dec;


    reg [3:0] cntToDisp1 = 0;
    reg [3:0] cntToDisp2 = 0;
    reg [3:0] cntToDisp3 = 0;
    reg [3:0] cntToDisp4 = 0;


    wire slow_clk;
    wire [3:0] cntTo7SegUp1;
    wire [3:0] cntTo7SegUp2;
    wire [3:0] cntTo7SegUp3;

    wire [3:0] cntTo7SegUp4;

    wire [3:0] cntTo7SegDn1;

    wire [3:0] cntTo7SegDn2;

    wire [3:0] cntTo7SegDn3;

    wire [3:0] cntTo7SegDn4;

    wire [6:0] ssegToDisp1;

    wire [6:0] ssegToDisp2;

    wire [6:0] ssegToDisp3;

    wire [6:0] ssegToDisp4;

    wire ss_toggle;

    wire clk_refresh;



    always @(*) begin

       extValReg1 = extVal [3:0];

       extValReg2 = extVal [7:4];

    end


    always @(*) begin

       case (mode)

          2'b00: begin

             inc_dec = 1;

             ssegIntExt1 = 0;

             ssegIntExt2 = 0;

          end


          2'b01: begin

             inc_dec = 1;

             if (extValReg1 < 10 && extValReg2 < 10) begin   //if the external values can be displayed with 1 digit

                ssegIntExt1 = extValReg1;
```

```
        ssegIntExt2 = extValReg2;
      end
      else begin      //if they can't be displayed wit one digit show 0
        ssegIntExt1 = 0;
        ssegIntExt2 = 0;
      end
    end

    2'b10: begin
      inc_dec = 0;
      ssegIntExt1 = 9;
      ssegIntExt2 = 9;
    end

    2'b11: begin
      inc_dec = 0;
      if (extValReg1 < 10 && extValReg2 < 10) begin   //if the external values can be displayed with 1 digit
        ssegIntExt1 = extValReg1;
        ssegIntExt2 = extValReg2;
      end
      else begin  //if they can't be displayed wit one digit show 0
        ssegIntExt1 = 9;
        ssegIntExt2 = 9;
      end
    end
  endcase
end

ss_toggle c8(ss, reset, ss_toggle);

clkdiv_refresh c9(clk, clk_refresh);
```

```
  clk_div_count c1(clk, slow_clk);


  up_counter c2 (slow_clk, reset, ss_toggle, ssegIntExt1, ssegIntExt2, cntTo7SegUp1, cntTo7SegUp2,
cntTo7SegUp3, cntTo7SegUp4);


  down_counter c10 (slow_clk, reset, ss_toggle, ssegIntExt1, ssegIntExt2, cntTo7SegDn1, cntTo7SegDn2,
cntTo7SegDn3, cntTo7SegDn4);


  //time_mux_state_machine c2(slow_clk, reset, ss_toggle, ssegIntExt1, ssegIntExt2, inc_dec, ssegInt1, ssegInt2,
ssegInt3, ssegInt4)



  always @(*) begin
    if (inc_dec) begin
      cntToDisp1 = cntTo7SegUp1;
      cntToDisp2 = cntTo7SegUp2;
      cntToDisp3 = cntTo7SegUp3;
      cntToDisp4 = cntTo7SegUp4;
    end

    else begin
      cntToDisp1 = cntTo7SegDn1;
      cntToDisp2 = cntTo7SegDn2;
      cntToDisp3 = cntTo7SegDn3;
      cntToDisp4 = cntTo7SegDn4;
    end
  end



  hex_to_7seg c3(cntToDisp1, ssegToDisp1);
  hex_to_7seg c4(cntToDisp2, ssegToDisp2);
```

```
    hex_to_7seg c5(cntToDisp3, ssegToDisp3);
    hex_to_7seg c6(cntToDisp4, ssegToDisp4);


    disp_fsm c7(clk_refresh, ssegToDisp1, ssegToDisp2, ssegToDisp3, ssegToDisp4, an, sseg, dp);

endmodule
```

hex_to_7seg:

```
`timescale 1ns / 1ps


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;
            4'b1010 : r = 7'b0001000;
            4'b1011 : r = 7'b1100000;
            4'b1100 : r = 7'b0110001;
            4'b1101 : r = 7'b1000010;
            4'b1110 : r = 7'b0110000;
            4'b1111 : r = 7'b0111000;
        endcase

endmodule
```

up_counter:

```
`timescale 1ns / 1ps


module up_counter(
    input clk,
    input reset,
    input ss,
    input[3:0] ext_ld1,
    input[3:0] ext_ld2,
    output reg [3:0] reg_d1,
    output reg [3:0] reg_d2,
    output reg [3:0] reg_d3,
    output reg [3:0] reg_d4
    );

    initial begin
        reg_d1 = 0;
        reg_d2 = 0;
        reg_d3 = 0;
        reg_d4 = 0;
    end

    always @(posedge clk) begin

        if (reset) begin
            reg_d1 <= 0;
            reg_d2 <= 0;
            reg_d3 <= ext_ld1;
            reg_d4 <= ext_ld2;
        end

            if(!(reg_d1==9 && reg_d2==9 && reg_d3==9 && reg_d4==9)) begin

                if(reg_d1 == 9) begin
                    reg_d1 <= 0;
                    if(reg_d2 == 9) begin
                        reg_d2 <= 0;
                        if (reg_d3 == 9) begin
                            reg_d3 <= 0;
                            if (reg_d4 == 9) begin
                            end
                            else
                                reg_d4 <= reg_d4+1;
                        end
                        else
                            reg_d3 <= reg_d3+1;
                    end
                    else
```

```
                        reg_d2 <= reg_d2+1;
                end
                else begin
                    if (ss)
                        reg_d1 <= reg_d1+1;
                end

            end
    end

endmodule
```

down_counter:

```
`timescale 1ns / 1ps

module down_counter(
    input clk,
    input reset,
    input ss,
    input[3:0] ext_ld1,
    input[3:0] ext_ld2,
    output reg [3:0] reg_d1,
    output reg [3:0] reg_d2,
    output reg [3:0] reg_d3,
    output reg [3:0] reg_d4
);

initial begin
```

```
    reg_d1 = 0;
    reg_d2 = 0;
    reg_d3 = 0;
    reg_d4 = 0;
end

always @(posedge clk) begin

    if (reset) begin
        reg_d1 = 9;
        reg_d2 = 9;
        reg_d3 = ext_ld1;
        reg_d4 = ext_ld2;
    end

        if (!(reg_d1==0 && reg_d2==0 && reg_d3==0 && reg_d4==0)) begin

            if(reg_d1 == 0) begin
                reg_d1 <= 9;
                if(reg_d2 == 0) begin
                    reg_d2 <= 9;
                    if (reg_d3 == 0) begin
                        reg_d3 <= 9;
                        if (reg_d4 == 0) begin
                        end
                        else
                            reg_d4 <= reg_d4-1;
                    end
                    else
                        reg_d3 <= reg_d3-1;
                end
                else
                    reg_d2 <= reg_d2-1;
            end
            else begin
                if(ss)
                    reg_d1 <= reg_d1-1;
            end

        end
end

endmodule
```

ss_toggle:

```
`timescale 1ns / 1ps


module ss_toggle(
    //input clk,
    input signal,
    input reset,
    output reg toggle
    );

    //reg [1:0] state = 2'b00;
    //reg [1:0] next_state;
    reg buffer;
```

```
initial begin
    toggle = 0;
    buffer = 1;
end

always @(posedge signal or posedge reset) begin
    if (reset)
        toggle <= 0;
    else begin
        toggle <= buffer;
        buffer <= ~buffer;
    end
end



//combinational logic
/*
always@(*) begin
    case (state)

        2'b00 : begin
            if(~signal)
                next_state = 2'b00;
            else
                next_state = 2'b01;
        end

        2'b01 : begin
            toggle = ~toggle;
            next_state = 2'b10;
        end
        2'b10: begin
           if(~signal)
              next_state = 2'b00;
           else
              next_state = 2'b10;
        end

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

     endcase
  end



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



Endmodule
```

Clkdiv_refresh:

```
`timescale 1ns / 1ps


module clkdiv_refresh(
    input clk,
    output clk_out
    );

    reg [8:0] COUNT = 0;

    assign clk_out = COUNT[8];

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

endmodule
```

clk_div_count:

```
`timescale 1ns / 1ps

module clk_div_count(
    input clk,
    output reg clk_out
    );

    reg [18:0] COUNT = 0;
    //reg [1:0] COUNT = 0;

    initial begin
        clk_out = 0;
    end

    always @(posedge clk) begin

        COUNT = COUNT+1;

        if (COUNT >= 500000) begin
            clk_out <= !clk_out;
            COUNT <= 0;
        end
    end

endmodule
```

disp_fsm:

```
`timescale 1ns / 1ps



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

    reg [1:0] state = 2'b00;
    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: sseg = in0;
            2'b01: sseg = in1;
            2'b10: sseg = in2;
            2'b11: sseg = in3;
        endcase

        case(state)
            2'b00: an = 4'b1110;
            2'b01: an = 4'b1101;
            2'b10: an = 4'b1011;
```

```
            2'b11: an = 4'b0111;
        endcase

        case (state)
            2'b00: dp = 1;
            2'b01: dp = 1;
            2'b10: dp = 0;
            2'b11: dp = 1;
        endcase
    end

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

endmodule
```

Constraints:

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project

## 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]
        set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets ss_IBUF]
```

```
## Switches
set_property PACKAGE_PIN V17 [get_ports {mode[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {mode[0]}]
set_property PACKAGE_PIN V16 [get_ports {mode[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {mode[1]}]
set_property PACKAGE_PIN W16 [get_ports {extVal[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {extVal[0]}]
set_property PACKAGE_PIN W17 [get_ports {extVal[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {extVal[1]}]
set_property PACKAGE_PIN W15 [get_ports {extVal[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {extVal[2]}]
set_property PACKAGE_PIN V15 [get_ports {extVal[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {extVal[3]}]
set_property PACKAGE_PIN W14 [get_ports {extVal[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {extVal[4]}]
set_property PACKAGE_PIN W13 [get_ports {extVal[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {extVal[5]}]
set_property PACKAGE_PIN V2 [get_ports {extVal[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {extVal[6]}]
set_property PACKAGE_PIN T3 [get_ports {extVal[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {extVal[7]}]


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

Test Bench Code:

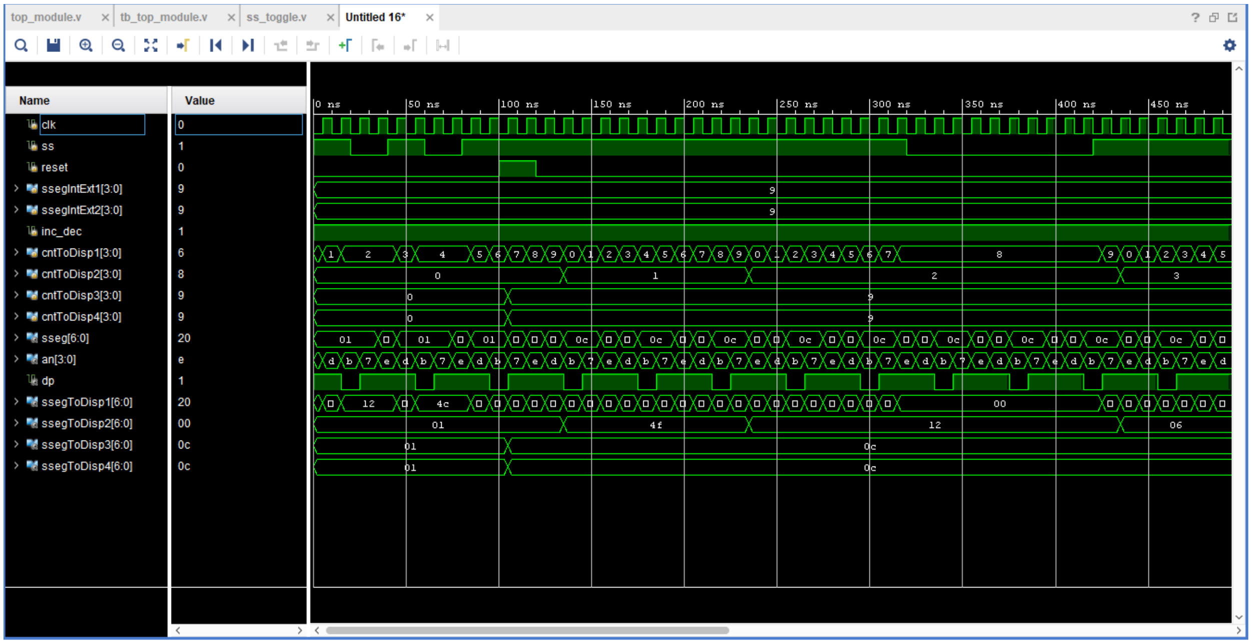
```
`timescale 1ns / 1ps


module tb_top_module;


    reg clk = 0;

    reg ss = 0;

    reg reset = 0;

    reg [3:0] ssegIntExt1 = 9;

    reg [3:0] ssegIntExt2 = 9;

    reg inc_dec = 0;

    reg [3:0] cntToDisp1;

    reg [3:0] cntToDisp2;

    reg [3:0] cntToDisp3;

    reg [3:0] cntToDisp4;





    wire [6:0] sseg;

    wire [3:0] an;

    wire dp;

    wire [3:0] cntTo7SegUp1;

    wire [3:0] cntTo7SegUp2;

    wire [3:0] cntTo7SegUp3;

    wire [3:0] cntTo7SegUp4;


    wire [3:0] cntTo7SegDn1;

    wire [3:0] cntTo7SegDn2;

    wire [3:0] cntTo7SegDn3;

    wire [3:0] cntTo7SegDn4;
    wire [6:0] ssegToDisp1;
    wire [6:0] ssegToDisp2;
    wire [6:0] ssegToDisp3;
    wire [6:0] ssegToDisp4;

    //top_module ut (clk, ss, reset, mode, in, sseg, an, dp);

    up_counter c2 (clk, reset, ss, ssegIntExt1, ssegIntExt2, cntTo7SegUp1, cntTo7SegUp2, cntTo7SegUp3, cntTo7SegUp4);

    down_counter c10 (clk, reset, ss, ssegIntExt1, ssegIntExt2, cntTo7SegDn1, cntTo7SegDn2, cntTo7SegDn3, cntTo7SegDn4);

    always @(*) begin
      if (inc_dec) begin
          cntToDisp1 = cntTo7SegUp1;
          cntToDisp2 = cntTo7SegUp2;
          cntToDisp3 = cntTo7SegUp3;
          cntToDisp4 = cntTo7SegUp4;
      end

      else begin
          cntToDisp1 = cntTo7SegDn1;
          cntToDisp2 = cntTo7SegDn2;
          cntToDisp3 = cntTo7SegDn3;
          cntToDisp4 = cntTo7SegDn4;
      end
    end

    hex_to_7seg c3(cntToDisp1, ssegToDisp1);
```

```
hex_to_7seg c4(cntToDisp2, ssegToDisp2);
hex_to_7seg c5(cntToDisp3, ssegToDisp3);
hex_to_7seg c6(cntToDisp4, ssegToDisp4);


disp_fsm c7(clk, ssegToDisp1, ssegToDisp2, ssegToDisp3, ssegToDisp4, an, sseg, dp);

initial begin

reset = 1;

#20;

reset = 0;
ss = 1;

#20;

ss = 0;

#20;

ss = 1;

#20;

ss = 0;

#20;

ss = 1;
```

```
   #20;

   reset = 1;

   #20;

   reset = 0;
   ss = 1;

   #200;

   ss = 0;

   #100;

   ss = 1;

   #300;

   end

   always
   #5 clk = ~clk;


Endmodule
```

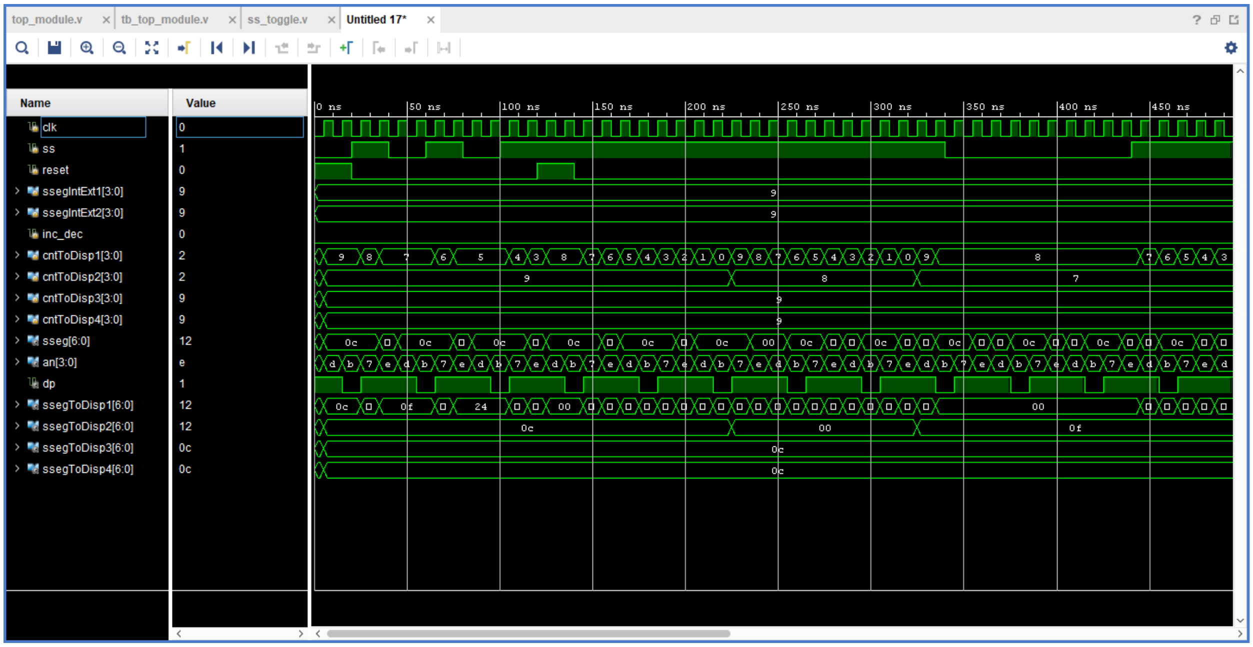
Test Bench Waveforms:

Incrementing:



Name	Value
clk	0
ss	1
reset	0
ssegIntExt1[3:0]	9
ssegIntExt2[3:0]	9
inc_dec	1
cntToDisp1[3:0]	6
cntToDisp2[3:0]	8
cntToDisp3[3:0]	9
cntToDisp4[3:0]	9
sseg[6:0]	20
an[3:0]	e
dp	1
ssegToDisp1[6:0]	20
ssegToDisp2[6:0]	00
ssegToDisp3[6:0]	0c
ssegToDisp4[6:0]	0c

Decrementing:



Name	Value
clk	0
ss	1
reset	0
ssegIntExt1[3:0]	9
ssegIntExt2[3:0]	9
inc_dec	0
cntToDisp1[3:0]	2
cntToDisp2[3:0]	2
cntToDisp3[3:0]	9
cntToDisp4[3:0]	9
sseg[6:0]	12
an[3:0]	e
dp	1
ssegToDisp1[6:0]	12
ssegToDisp2[6:0]	12
ssegToDisp3[6:0]	0c
ssegToDisp4[6:0]	0c