Name: _____Servando_Olvera________ ID# ________1001909287_______

Date Submitted: _____02-06-2024____ Time Submitted _____9:00_pm______

CSE 3341 Digital Logic Design II

CSE 5357 Advanced Digital Logic Design

Spring Semester 2024

Lab 2 –Multiplexed Seven-Segment Displays

Due Date – February 6, 2024, 11:59 PM

Submit on Canvas Assignments

DESIGN REQUIREMENTS

Design a decoder/controller for the *HexBoard* four-digit multiplexed seven segment display. Capture your design using SystemVerilog. Realize your design on the DE10-Lite + HexBoard devices found in your ADL Lab Kit. In Part A, you will test your decoder/controller by displaying, in hexadecimal, the output of a 16-bit binary counter as shown in Figure 1. In Part B, you will use the *HexBoard* to display a crawling message. This can be realized by replacing the 16-bit counter with a circular buffer as shown in Figure 2.

- SystemVerilog must be used for coding the solutions.
- Structured design must be used and documented by a hierarchy diagram.
- Code must be commented.

Part A – Multiplexed seven-segment decoder/controller

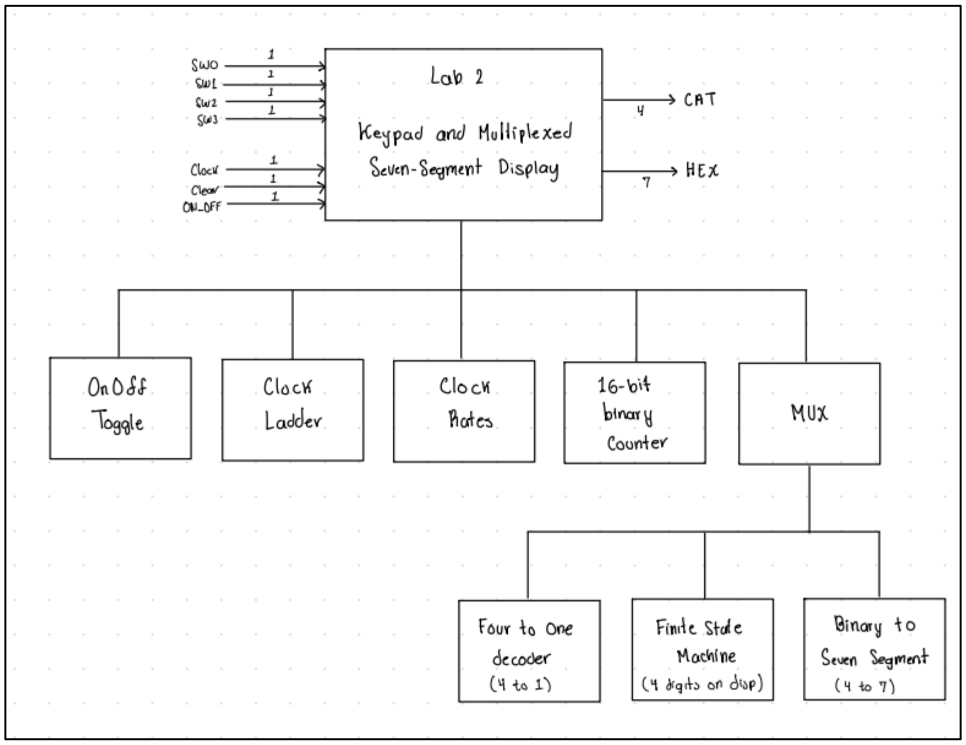
1. Design and implement the circuit in Figure 1.
2. Demonstrate that the counter counts and displays properly from 0000 to FFFF and repeats.
3. Demonstrate that the On/Off and Reset features work.
4. Mathematically derive the count rate and mux rate for the clock ladder settings in Figure 1.
5. Add a feature that allows the mux rate to be controlled from the DE10-Lite+HexBoard switches and pushbuttons.
6. Experimenally determine the slowest mux rate that does not produce flicker on the display.
7. Does speeding up the mux rate from this minimum improve the display?

Part B – Crawling message display

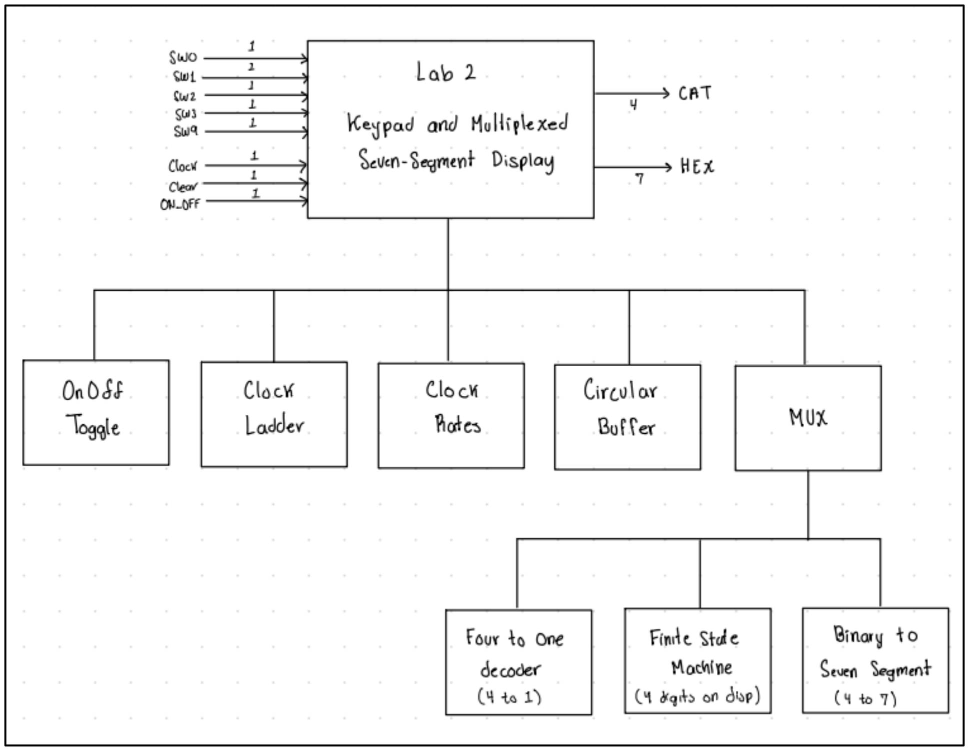
1. Design and implement the circuit in Figure 2.
2. Demonstrate that it can continuously display the message *0123456789AbCdEF* in hex.
3. Increase the crawl rate by a factor of 4 and observe the effect on the display.
4. Reprogram your circular buffer to display the message HELLO 2 YOU.

HIRERARCHY DIAGRAMS

Part A

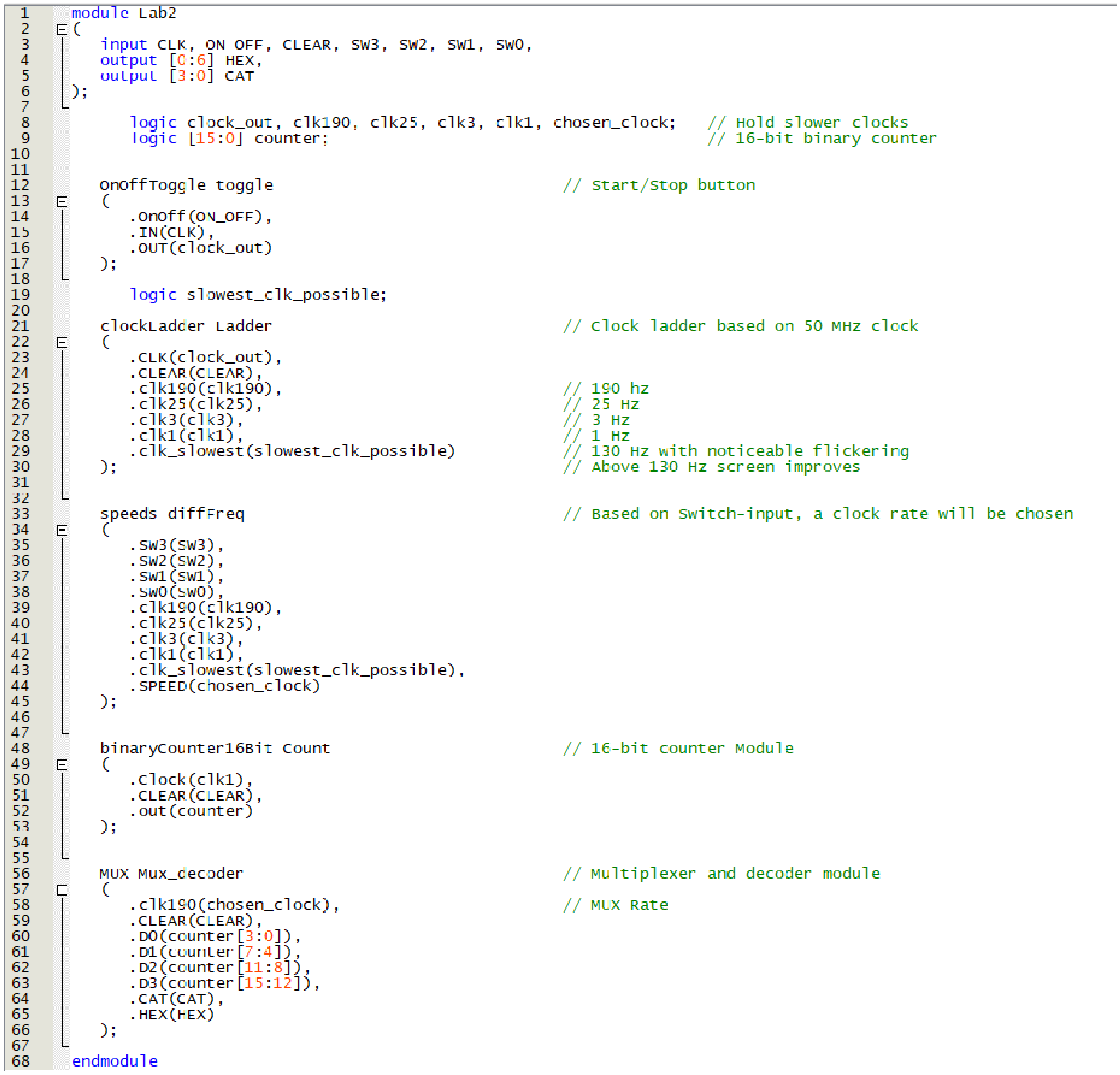

SW0
SW1
SW2
SW3
Clock
Clear
ON_OFF
1
Lab 2
Keypad and Multiplexed Seven-Segment Display
4
CAT
7
HEX
On Off Toggle
Clock Ladder
Clock Rates
16-bit binary Counter
MUX
Four to One decoder (4 to 1)
Finite State Machine (4 digits on disp)
Binary to Seven Segment (4 to 7)

Part B


SW0
SW1
SW2
SW3
SW9
Clock
Clear
ON_OFF
1
Lab 2
Keypad and Multiplexed Seven-Segment Display
4
CAT
7
HEX
On Off Toggle
Clock Ladder
Clock Rates
Circular Buffer
MUX
Four to One decoder (4 to 1)
Finite State Machine (4 digits on disp)
Binary to Seven Segment (4 to 7)

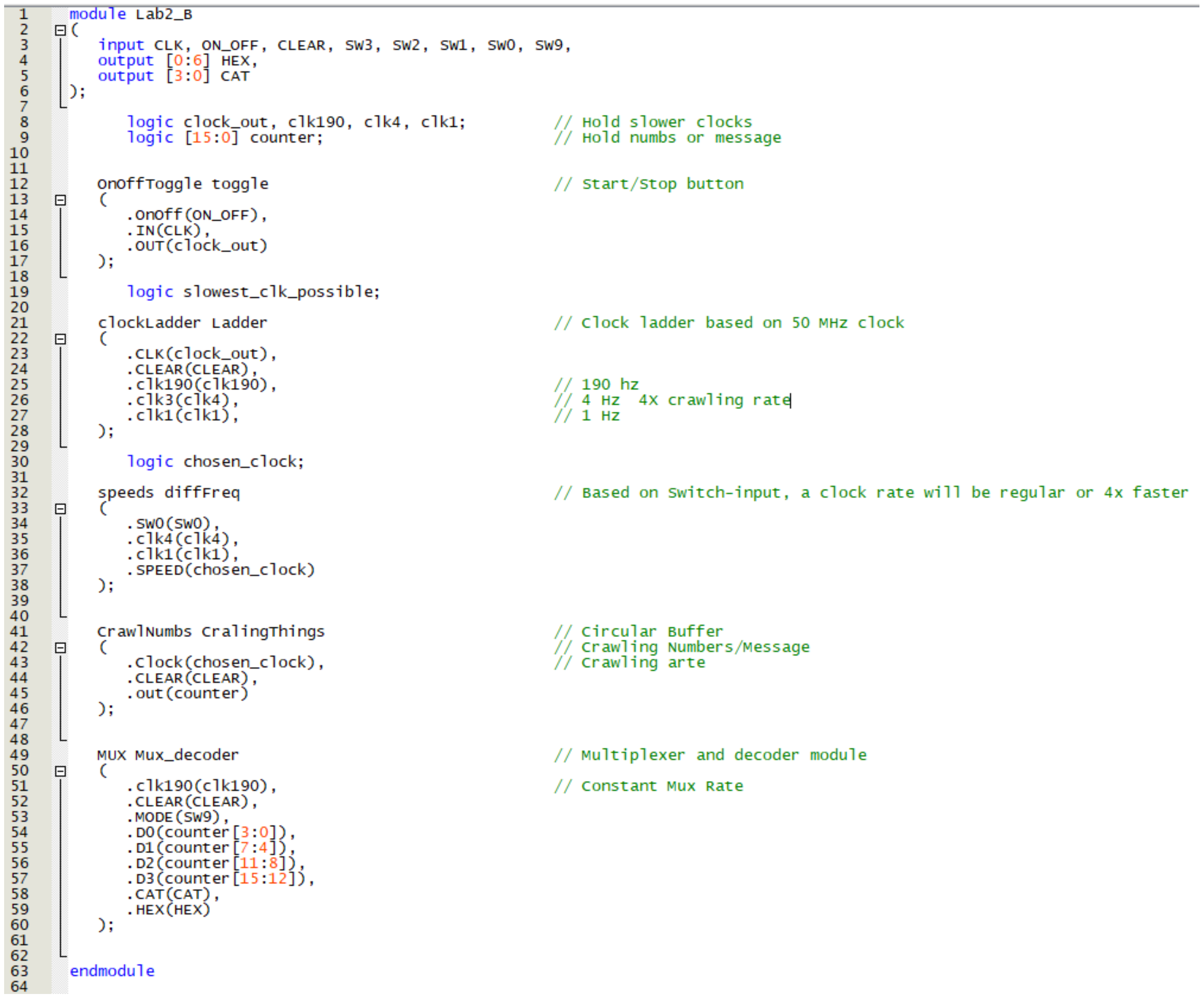
SYSTEM-VERILOG CODE

Part A Top Module



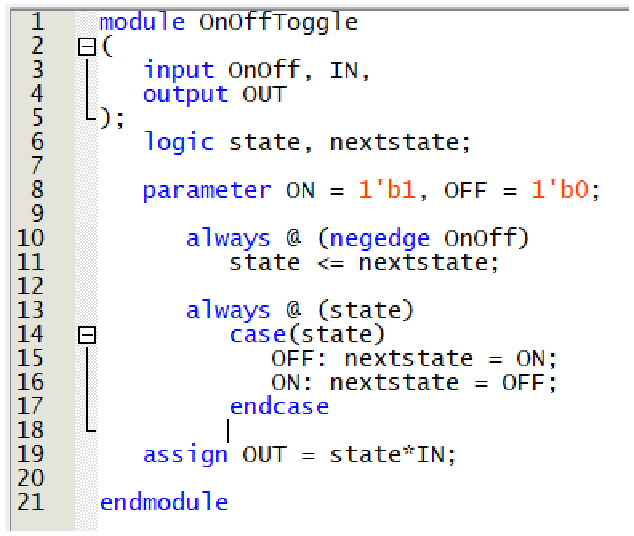
```
module Lab2
(
   input CLK, ON_OFF, CLEAR, SW3, SW2, SW1, SW0,
   output [0:6] HEX,
   output [3:0] CAT
);

      logic clock_out, clk190, clk25, clk3, clk1, chosen_clock;    // Hold slower clocks
      logic [15:0] counter;                                        // 16-bit binary counter


   onOffToggle toggle                               // Start/Stop button
   (
      .onOff(ON_OFF),
      .IN(CLK),
      .OUT(clock_out)
   );

      logic slowest_clk_possible;

   clockLadder Ladder                               // Clock ladder based on 50 MHz clock
   (
      .CLK(clock_out),
      .CLEAR(CLEAR),
      .clk190(clk190),                              // 190 hz
      .clk25(clk25),                                // 25 Hz
      .clk3(clk3),                                  // 3 Hz
      .clk1(clk1),                                  // 1 Hz
      .clk_slowest(slowest_clk_possible)            // 130 Hz with noticeable flickering
   );                                               // Above 130 Hz screen improves


   speeds diffFreq                                  // Based on Switch-input, a clock rate will be chosen
   (
      .SW3(SW3),
      .SW2(SW2),
      .SW1(SW1),
      .SW0(SW0),
      .clk190(clk190),
      .clk25(clk25),
      .clk3(clk3),
      .clk1(clk1),
      .clk_slowest(slowest_clk_possible),
      .SPEED(chosen_clock)
   );


   binaryCounter16Bit Count                         // 16-bit counter Module
   (
      .Clock(clk1),
      .CLEAR(CLEAR),
      .out(counter)
   );


   MUX Mux_decoder                                  // Multiplexer and decoder module
   (
      .clk190(chosen_clock),                        // MUX Rate
      .CLEAR(CLEAR),
      .D0(counter[3:0]),
      .D1(counter[7:4]),
      .D2(counter[11:8]),
      .D3(counter[15:12]),
      .CAT(CAT),
      .HEX(HEX)
   );

endmodule
```

Part B Top Module



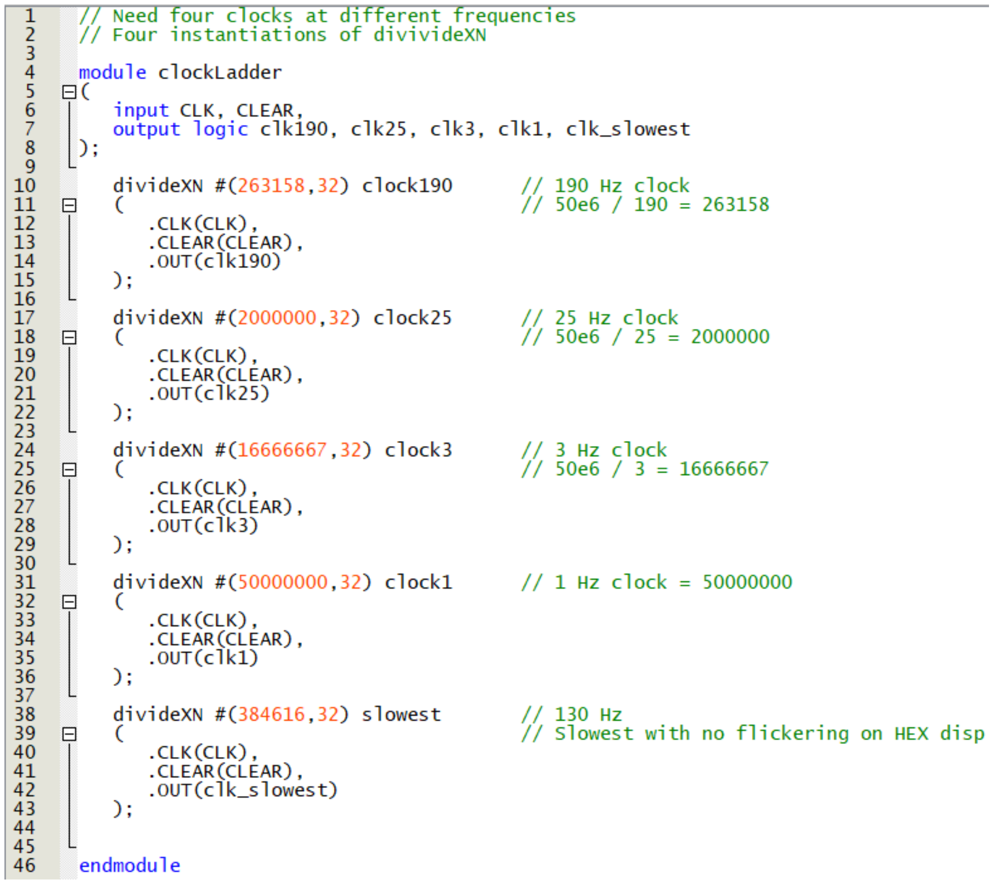
```
module Lab2_B
(
    input CLK, ON_OFF, CLEAR, SW3, SW2, SW1, SW0, SW9,
    output [0:6] HEX,
    output [3:0] CAT
);

        logic clock_out, clk190, clk4, clk1;           // Hold slower clocks
        logic [15:0] counter;                           // Hold numbs or message


    OnOffToggle toggle                                  // Start/Stop button
    (
        .OnOff(ON_OFF),
        .IN(CLK),
        .OUT(clock_out)
    );

        logic slowest_clk_possible;

    clockLadder Ladder                                  // Clock ladder based on 50 MHz clock
    (
        .CLK(clock_out),
        .CLEAR(CLEAR),
        .clk190(clk190),                                // 190 hz
        .clk3(clk4),                                    // 4 Hz  4X crawling rate
        .clk1(clk1),                                    // 1 Hz
    );

        logic chosen_clock;

    speeds diffFreq                                     // Based on Switch-input, a clock rate will be regular or 4x faster
    (
        .SW0(SW0),
        .clk4(clk4),
        .clk1(clk1),
        .SPEED(chosen_clock)
    );


    CrawlNumbs CralingThings                            // Circular Buffer
    (                                                   // Crawling Numbers/Message
        .clock(chosen_clock),                           // Crawling arte
        .CLEAR(CLEAR),
        .out(counter)
    );


    MUX Mux_decoder                                     // Multiplexer and decoder module
    (
        .clk190(clk190),                                // Constant Mux Rate
        .CLEAR(CLEAR),
        .MODE(SW9),
        .D0(counter[3:0]),
        .D1(counter[7:4]),
        .D2(counter[11:8]),
        .D3(counter[15:12]),
        .CAT(CAT),
        .HEX(HEX)
    );


endmodule
```

ON & OFF Toggle Module



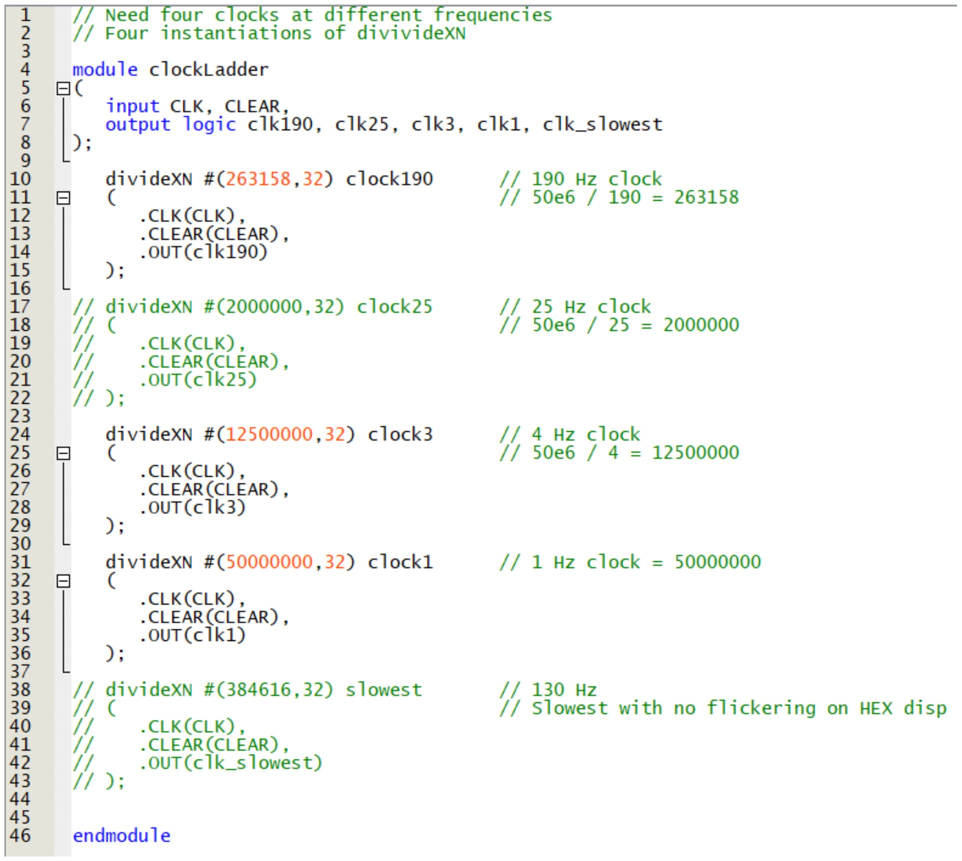
```
module OnOffToggle
(
    input OnOff, IN,
    output OUT
);
    logic state, nextstate;

    parameter ON = 1'b1, OFF = 1'b0;

        always @ (negedge OnOff)
            state <= nextstate;

        always @ (state)
            case(state)
                OFF: nextstate = ON;
                ON: nextstate = OFF;
            endcase

    assign OUT = state*IN;

endmodule
```

Clock Ladder Module [Part A]



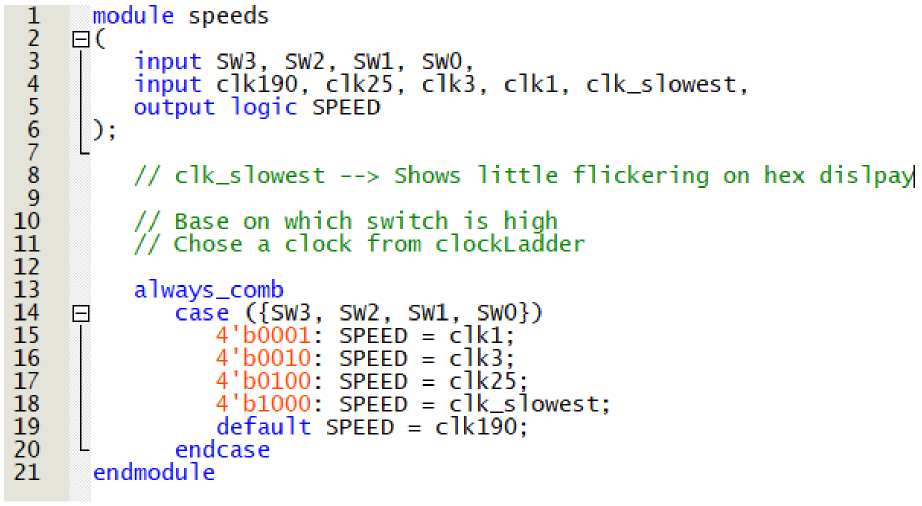
```
// Need four clocks at different frequencies
// Four instantiations of divivideXN

module clockLadder
(
    input CLK, CLEAR,
    output logic clk190, clk25, clk3, clk1, clk_slowest
);

    divideXN #(263158,32) clock190      // 190 Hz clock
    (                                   // 50e6 / 190 = 263158
        .CLK(CLK),
        .CLEAR(CLEAR),
        .OUT(clk190)
    );

    divideXN #(2000000,32) clock25      // 25 Hz clock
    (                                   // 50e6 / 25 = 2000000
        .CLK(CLK),
        .CLEAR(CLEAR),
        .OUT(clk25)
    );

    divideXN #(16666667,32) clock3      // 3 Hz clock
    (                                   // 50e6 / 3 = 16666667
        .CLK(CLK),
        .CLEAR(CLEAR),
        .OUT(clk3)
    );

    divideXN #(50000000,32) clock1      // 1 Hz clock = 50000000
    (
        .CLK(CLK),
        .CLEAR(CLEAR),
        .OUT(clk1)
    );

    divideXN #(384616,32) slowest       // 130 Hz
    (                                   // Slowest with no flickering on HEX disp
        .CLK(CLK),
        .CLEAR(CLEAR),
        .OUT(clk_slowest)
    );


endmodule
```

Clock Ladder Module [Part B]



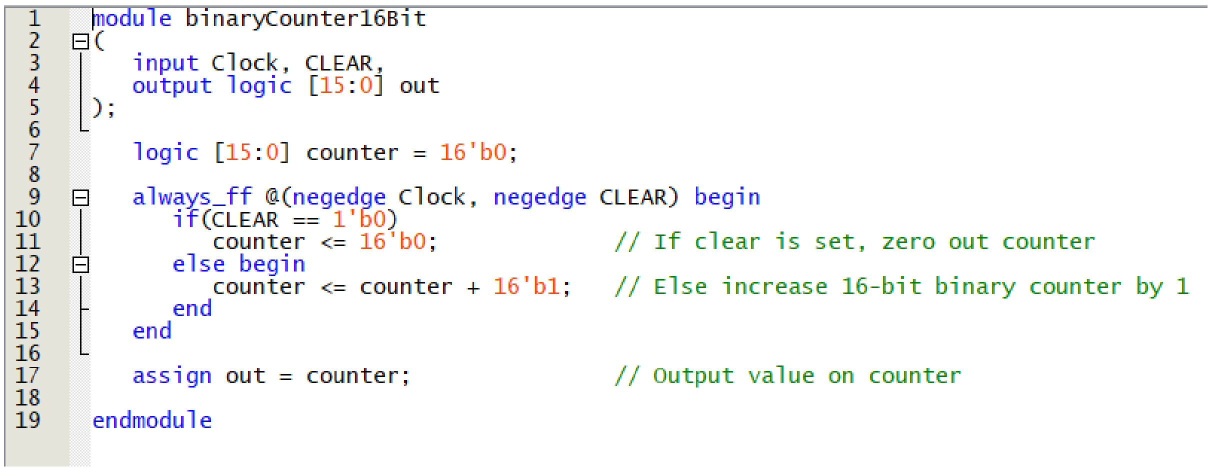
```
// Need four clocks at different frequencies
// Four instantiations of divivideXN

module clockLadder
(
    input CLK, CLEAR,
    output logic clk190, clk25, clk3, clk1, clk_slowest
);

    divideXN #(263158,32) clock190      // 190 Hz clock
    (                                   // 50e6 / 190 = 263158
        .CLK(CLK),
        .CLEAR(CLEAR),
        .OUT(clk190)
    );

//  divideXN #(2000000,32) clock25      // 25 Hz clock
//  (                                   // 50e6 / 25 = 2000000
//      .CLK(CLK),
//      .CLEAR(CLEAR),
//      .OUT(clk25)
//  );

    divideXN #(12500000,32) clock3      // 4 Hz clock
    (                                   // 50e6 / 4 = 12500000
        .CLK(CLK),
        .CLEAR(CLEAR),
        .OUT(clk3)
    );

    divideXN #(50000000,32) clock1      // 1 Hz clock = 50000000
    (
        .CLK(CLK),
        .CLEAR(CLEAR),
        .OUT(clk1)
    );

//  divideXN #(384616,32) slowest       // 130 Hz
//  (                                   // Slowest with no flickering on HEX disp
//      .CLK(CLK),
//      .CLEAR(CLEAR),
//      .OUT(clk_slowest)
//  );


endmodule
```

Clock Rates Module



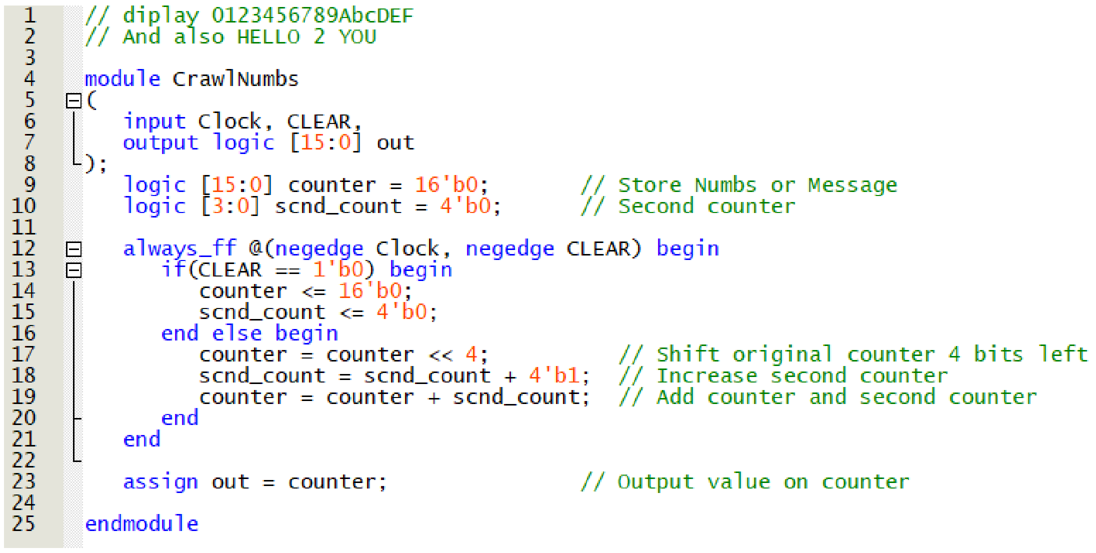
```
module speeds
(
    input SW3, SW2, SW1, SW0,
    input clk190, clk25, clk3, clk1, clk_slowest,
    output logic SPEED
);

    // clk_slowest --> Shows little flickering on hex dislpay

    // Base on which switch is high
    // Chose a clock from clockLadder

    always_comb
        case ({SW3, SW2, SW1, SW0})
            4'b0001: SPEED = clk1;
            4'b0010: SPEED = clk3;
            4'b0100: SPEED = clk25;
            4'b1000: SPEED = clk_slowest;
            default SPEED = clk190;
        endcase
endmodule
```

16-bit Binary Counter Module [Part A]



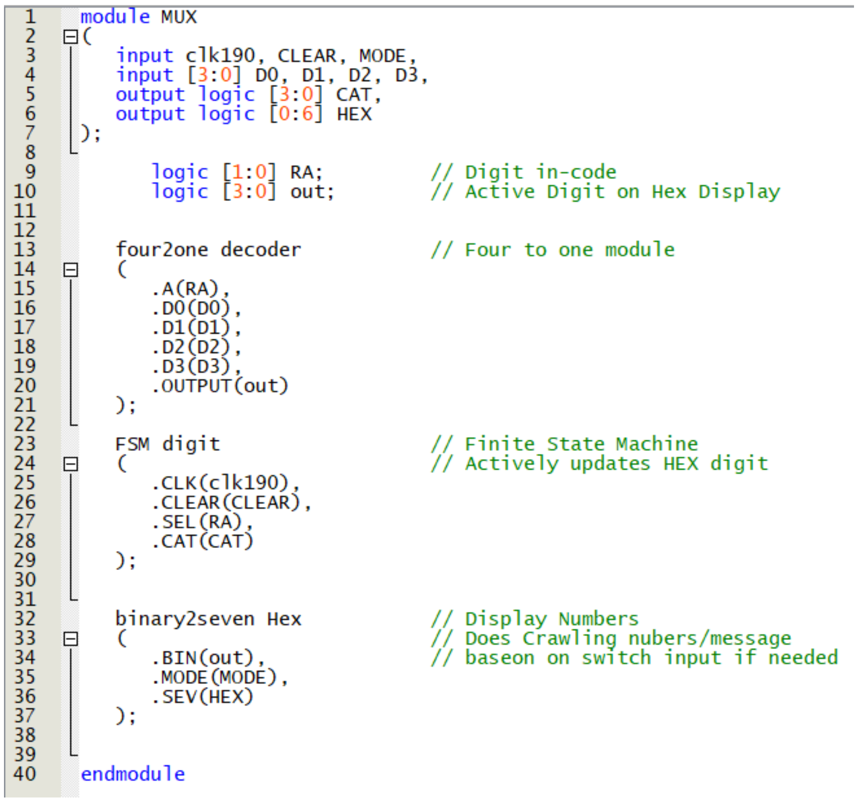
```
module binaryCounter16Bit
(
    input Clock, CLEAR,
    output logic [15:0] out
);

    logic [15:0] counter = 16'b0;

    always_ff @(negedge Clock, negedge CLEAR) begin
        if(CLEAR == 1'b0)
            counter <= 16'b0;               // If clear is set, zero out counter
        else begin
            counter <= counter + 16'b1;     // Else increase 16-bit binary counter by 1
        end
    end

    assign out = counter;                   // Output value on counter

endmodule
```

Circular Buffer Module [Part B]



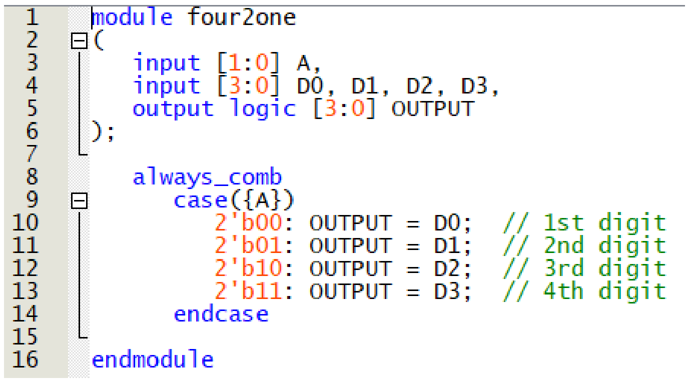
```
// diplay 0123456789AbcDEF
// And also HELLO 2 YOU

module CrawlNumbs
(
    input Clock, CLEAR,
    output logic [15:0] out
);
    logic [15:0] counter = 16'b0;       // Store Numbs or Message
    logic [3:0] scnd_count = 4'b0;      // Second counter

    always_ff @(negedge Clock, negedge CLEAR) begin
        if(CLEAR == 1'b0) begin
            counter <= 16'b0;
            scnd_count <= 4'b0;
        end else begin
            counter = counter << 4;             // Shift original counter 4 bits left
            scnd_count = scnd_count + 4'b1;     // Increase second counter
            counter = counter + scnd_count;     // Add counter and second counter
        end
    end

    assign out = counter;               // Output value on counter

endmodule
```

MUX Module



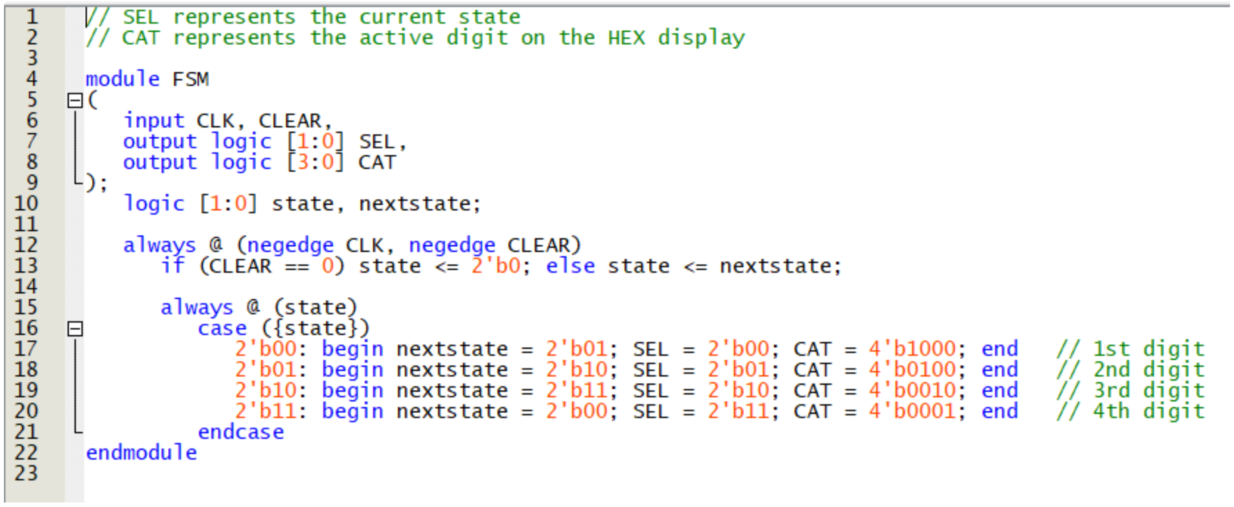
```
module MUX
(
   input clk190, CLEAR, MODE,
   input [3:0] D0, D1, D2, D3,
   output logic [3:0] CAT,
   output logic [0:6] HEX
);

      logic [1:0] RA;         // Digit in-code
      logic [3:0] out;        // Active Digit on Hex Display


   four2one decoder           // Four to one module
   (
      .A(RA),
      .D0(D0),
      .D1(D1),
      .D2(D2),
      .D3(D3),
      .OUTPUT(out)
   );

   FSM digit                  // Finite State Machine
   (                          // Actively updates HEX digit
      .CLK(clk190),
      .CLEAR(CLEAR),
      .SEL(RA),
      .CAT(CAT)
   );


   binary2seven Hex           // Display Numbers
   (                          // Does Crawling nubers/message
      .BIN(out),              // baseon on switch input if needed
      .MODE(MODE),
      .SEV(HEX)
   );


endmodule
```

Four To One Module



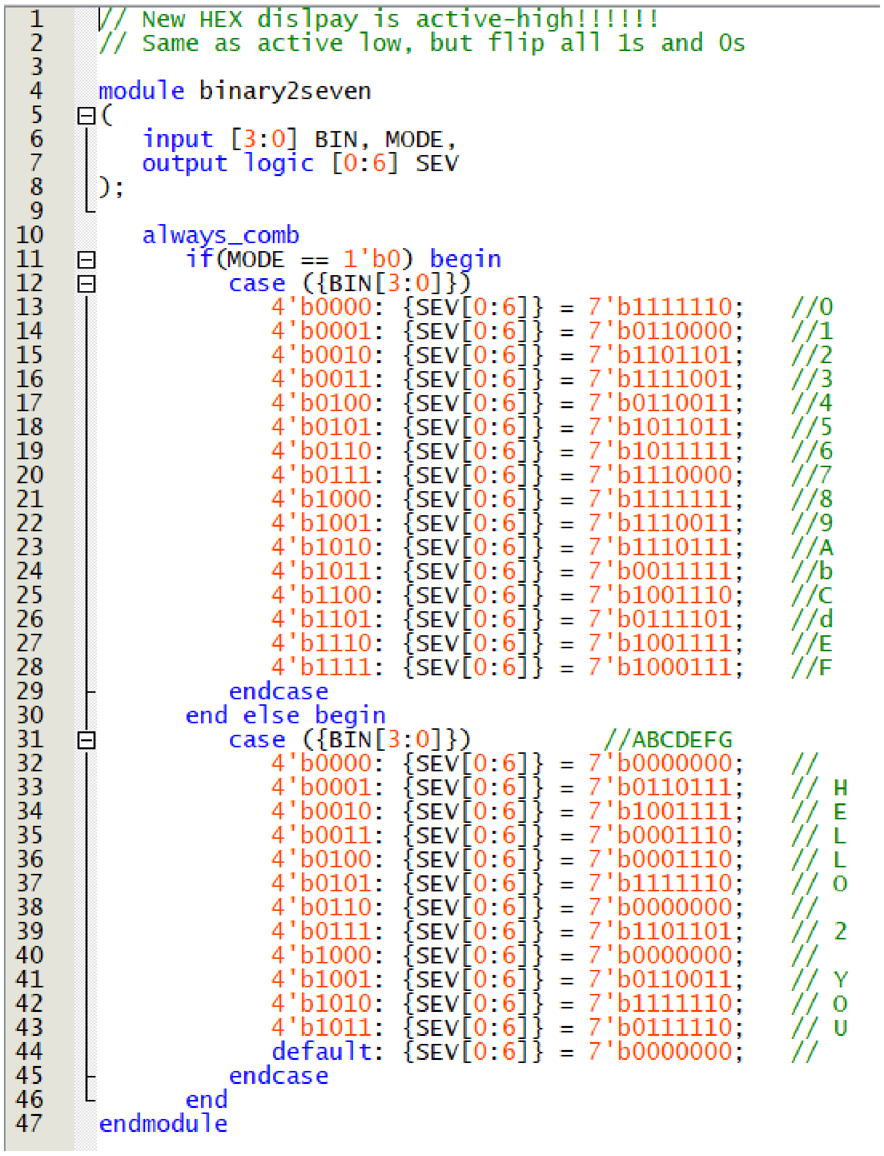
```
module four2one
(
   input [1:0] A,
   input [3:0] D0, D1, D2, D3,
   output logic [3:0] OUTPUT
);

   always_comb
      case({A})
         2'b00: OUTPUT = D0;  // 1st digit
         2'b01: OUTPUT = D1;  // 2nd digit
         2'b10: OUTPUT = D2;  // 3rd digit
         2'b11: OUTPUT = D3;  // 4th digit
      endcase

endmodule
```

Finite State Machine Module



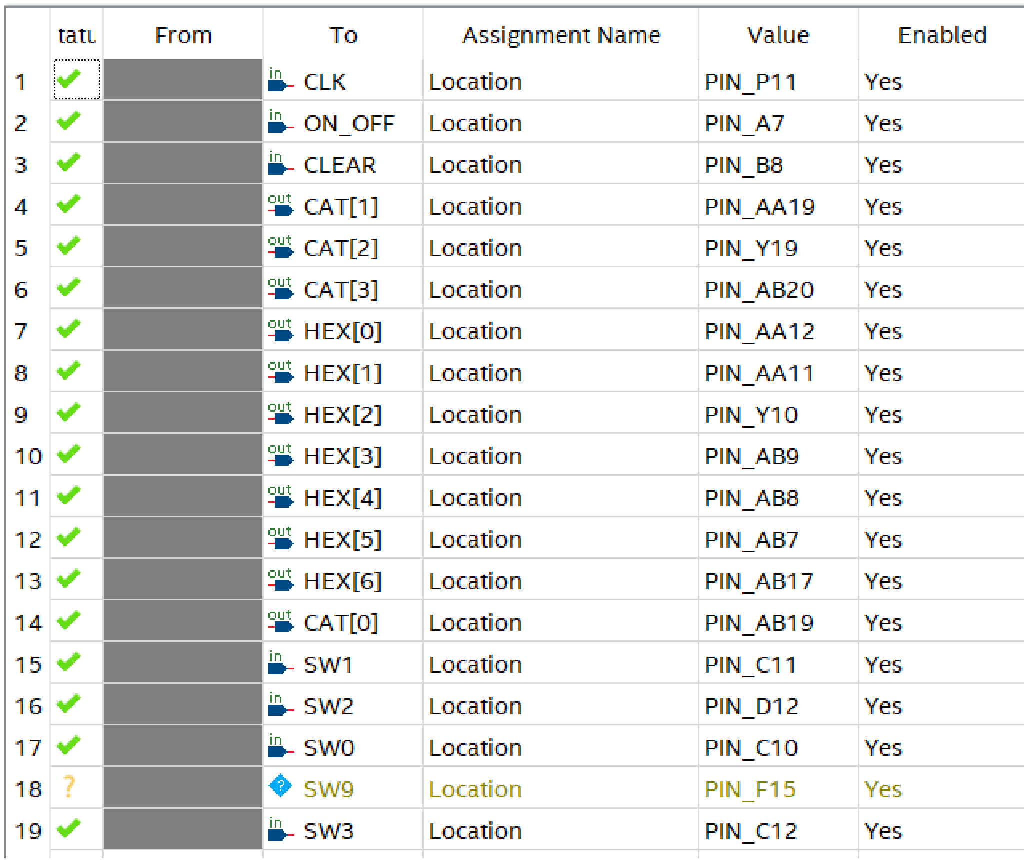
```
// SEL represents the current state
// CAT represents the active digit on the HEX display

module FSM
(
   input CLK, CLEAR,
   output logic [1:0] SEL,
   output logic [3:0] CAT
);
   logic [1:0] state, nextstate;

   always @ (negedge CLK, negedge CLEAR)
      if (CLEAR == 0) state <= 2'b0; else state <= nextstate;

      always @ (state)
         case ({state})
            2'b00: begin nextstate = 2'b01; SEL = 2'b00; CAT = 4'b1000; end   // 1st digit
            2'b01: begin nextstate = 2'b10; SEL = 2'b01; CAT = 4'b0100; end   // 2nd digit
            2'b10: begin nextstate = 2'b11; SEL = 2'b10; CAT = 4'b0010; end   // 3rd digit
            2'b11: begin nextstate = 2'b00; SEL = 2'b11; CAT = 4'b0001; end   // 4th digit
         endcase
endmodule
```

Binary To 7-Hex & To Message Decoder



```
// New HEX dislpay is active-high!!!!!!
// Same as active low, but flip all 1s and 0s

module binary2seven
(
   input [3:0] BIN, MODE,
   output logic [0:6] SEV
);

   always_comb
      if(MODE == 1'b0) begin
         case ({BIN[3:0]})
            4'b0000: {SEV[0:6]} = 7'b1111110;   //0
            4'b0001: {SEV[0:6]} = 7'b0110000;   //1
            4'b0010: {SEV[0:6]} = 7'b1101101;   //2
            4'b0011: {SEV[0:6]} = 7'b1111001;   //3
            4'b0100: {SEV[0:6]} = 7'b0110011;   //4
            4'b0101: {SEV[0:6]} = 7'b1011011;   //5
            4'b0110: {SEV[0:6]} = 7'b1011111;   //6
            4'b0111: {SEV[0:6]} = 7'b1110000;   //7
            4'b1000: {SEV[0:6]} = 7'b1111111;   //8
            4'b1001: {SEV[0:6]} = 7'b1110011;   //9
            4'b1010: {SEV[0:6]} = 7'b1110111;   //A
            4'b1011: {SEV[0:6]} = 7'b0011111;   //b
            4'b1100: {SEV[0:6]} = 7'b1001110;   //C
            4'b1101: {SEV[0:6]} = 7'b0111101;   //d
            4'b1110: {SEV[0:6]} = 7'b1001111;   //E
            4'b1111: {SEV[0:6]} = 7'b1000111;   //F
         endcase
      end else begin
         case ({BIN[3:0]})         //ABCDEFG
            4'b0000: {SEV[0:6]} = 7'b0000000;   //
            4'b0001: {SEV[0:6]} = 7'b0110111;   // H
            4'b0010: {SEV[0:6]} = 7'b1001111;   // E
            4'b0011: {SEV[0:6]} = 7'b0001110;   // L
            4'b0100: {SEV[0:6]} = 7'b0001110;   // L
            4'b0101: {SEV[0:6]} = 7'b1111110;   // O
            4'b0110: {SEV[0:6]} = 7'b0000000;   //
            4'b0111: {SEV[0:6]} = 7'b1101101;   // 2
            4'b1000: {SEV[0:6]} = 7'b0000000;   //
            4'b1001: {SEV[0:6]} = 7'b0110011;   // Y
            4'b1010: {SEV[0:6]} = 7'b1111110;   // O
            4'b1011: {SEV[0:6]} = 7'b0111110;   // U
            default: {SEV[0:6]} = 7'b0000000;   //
         endcase
      end
endmodule
```

PIN ASSIGMENTS



	tatu	From	To	Assignment Name	Value	Enabled
1	✓		CLK	Location	PIN_P11	Yes
2	✓		ON_OFF	Location	PIN_A7	Yes
3	✓		CLEAR	Location	PIN_B8	Yes
4	✓		CAT[1]	Location	PIN_AA19	Yes
5	✓		CAT[2]	Location	PIN_Y19	Yes
6	✓		CAT[3]	Location	PIN_AB20	Yes
7	✓		HEX[0]	Location	PIN_AA12	Yes
8	✓		HEX[1]	Location	PIN_AA11	Yes
9	✓		HEX[2]	Location	PIN_Y10	Yes
10	✓		HEX[3]	Location	PIN_AB9	Yes
11	✓		HEX[4]	Location	PIN_AB8	Yes
12	✓		HEX[5]	Location	PIN_AB7	Yes
13	✓		HEX[6]	Location	PIN_AB17	Yes
14	✓		CAT[0]	Location	PIN_AB19	Yes
15	✓		SW1	Location	PIN_C11	Yes
16	✓		SW2	Location	PIN_D12	Yes
17	✓		SW0	Location	PIN_C10	Yes
18	?		SW9	Location	PIN_F15	Yes
19	✓		SW3	Location	PIN_C12	Yes

SW9 is for part B (I implemented A & B similarly)

DEMOED IN PERSON

Demoed on 02/05/2024

To: TA (Madison)