Name: _____Servando_Olvera________ ID# ________1001909287_______

Date Submitted: _____04-18-2024____ Time Submitted _____9:00_pm______

CSE 3341 Digital Logic Design II

CSE 5357 Advanced Digital Logic Design

Spring Semester 2024

Lab 6 – Keypad Encoder and Scanner

100 points

Due Date – May 18, 2024, 11:59 PM

Submit on Canvas Assignments

DESIGN REQUIREMENTS

PURPOSE/OUTCOMES

To design, implement using the DE10-Lite + KeyPad + FlipBoard + HexBoard a keypad scanner and encoder to display keystrokes on a multiplexed seven-segment display. By successfully completing this lab you will be able to use the KeyPad and HexBoard for input and output units for the term project.

BACKGROUND

A basic calculator can be partitioned in to four functional units. The arithmetic unit (AU) receives two eight-bit operands, with the left-most bit being a sign-bit, encoded in two's complement and produces an output in two's complement. For the four-function term-project calculator, data will be entered in hexadecimal on the KeyPad and displayed in hexadecimal on the HexBoard. Hence, the KeyPad must be configured to output key strokes in hexadecimal characters. Note, the KeyPad (4x4 keypad) must be plugged in to the Arduino port on the DE 10-Lite.

DESIGN REQUIREMENTS

Your assignment is to configure and test the KeyPad Scanner and Encoder to capture four key strokes and display the hex code on the HexBoard.

1. Implement the BCD KeyPad Scanner and Encoder and display the outputs on the HexBoard.

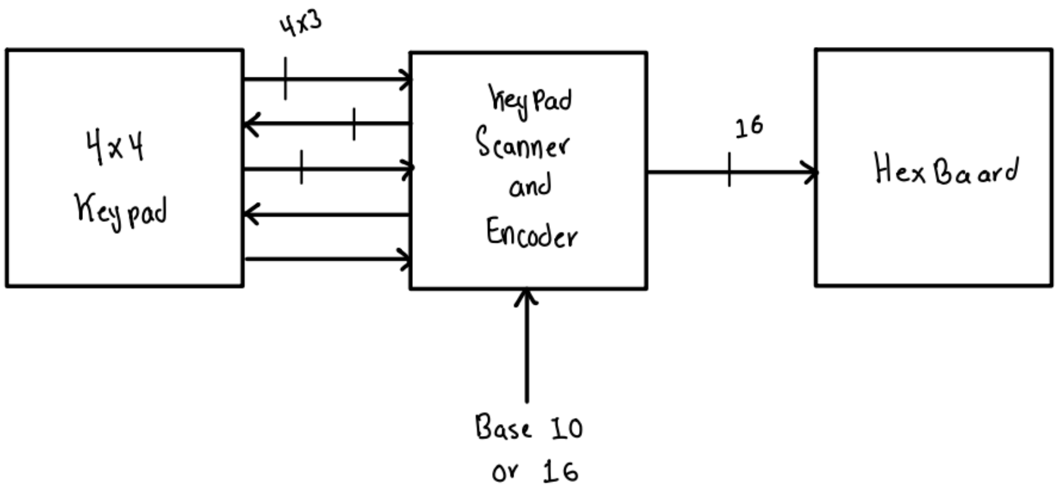
2. Demonstrate your implementation with an appropriate test.

3. Modify the code to capture hexadecimal characters.

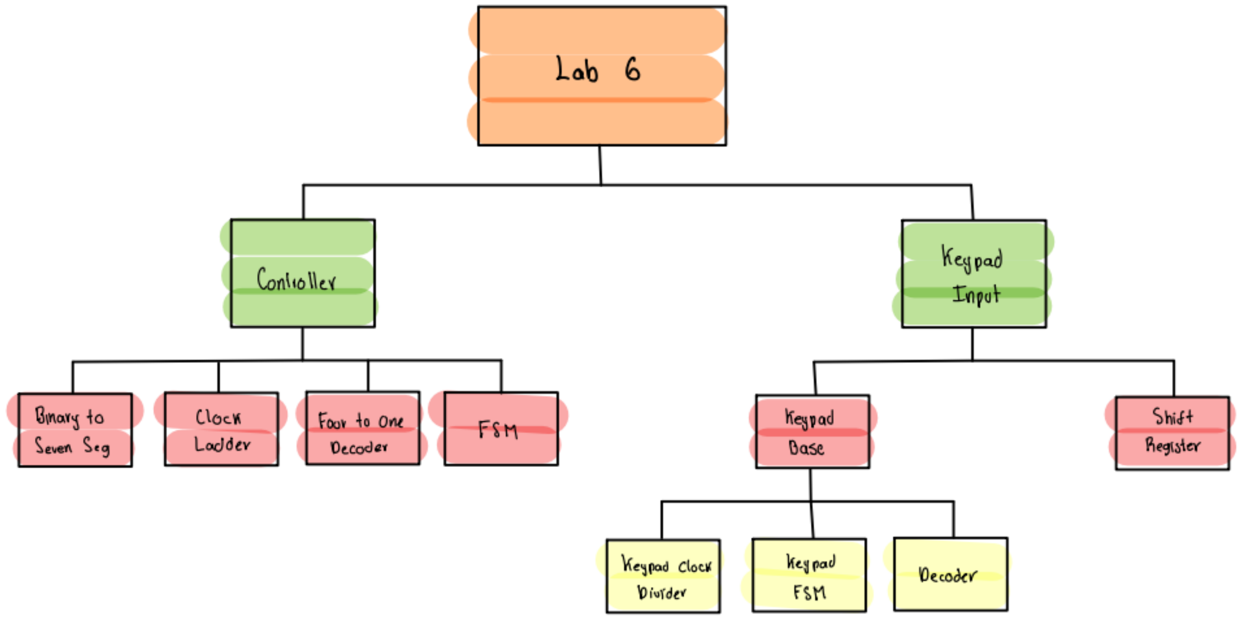
4. Implement the HEX KeyPad Scanner and Encoder and display the outputs on the HexBoard.

5. Demonstrate your implementation with an appropriate test.

ORGANIZATION DIAGRAM

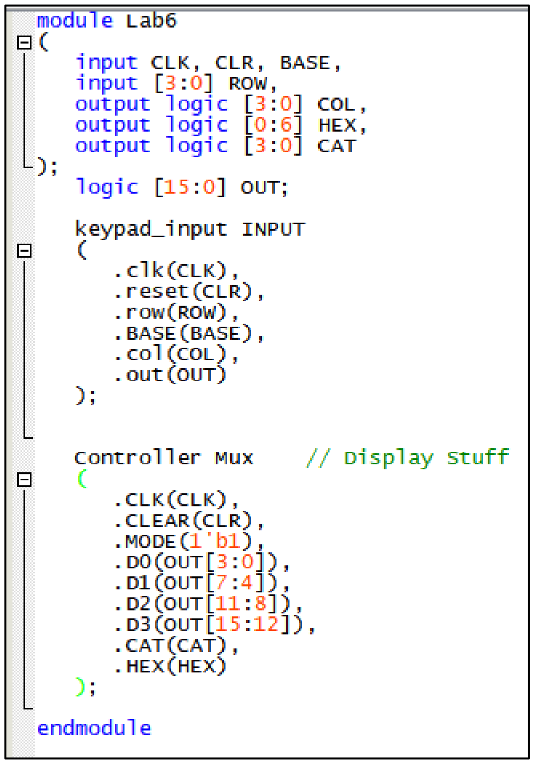


HIRERARCHY DIAGRAM



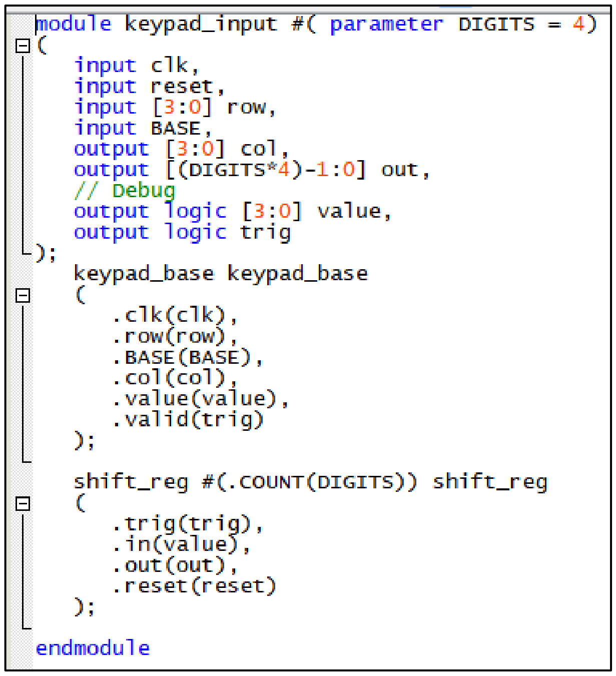
SYSTEM-VERILOG CODE

Part Top Module



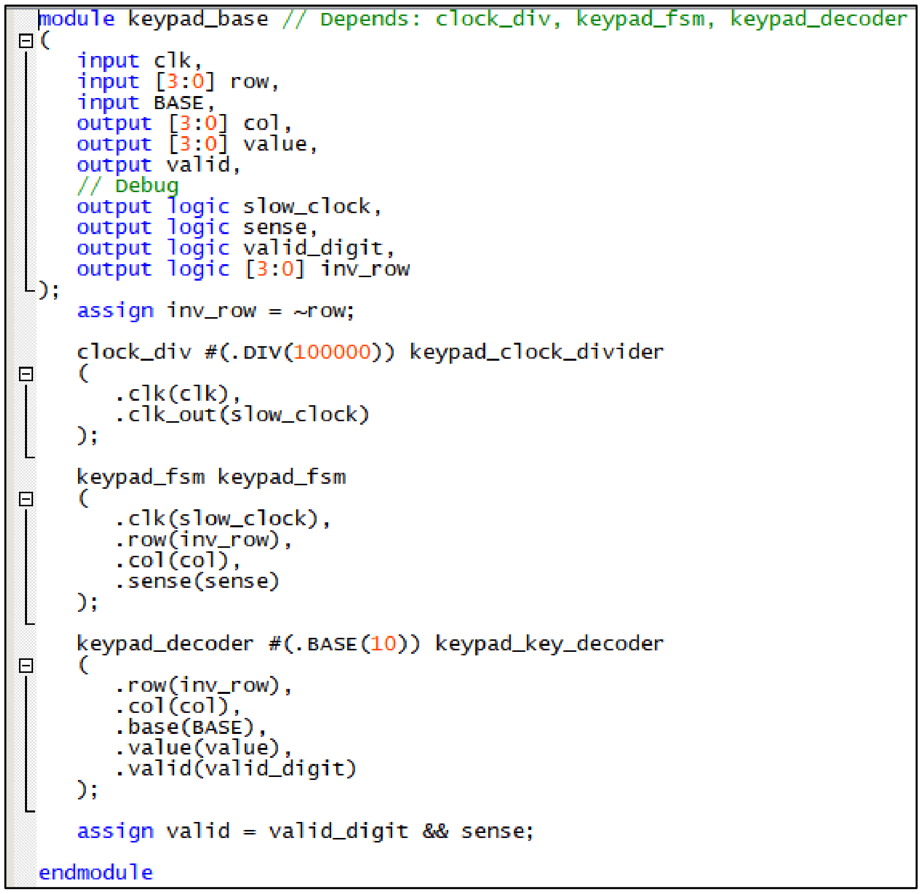
```
module Lab6
(
    input CLK, CLR, BASE,
    input [3:0] ROW,
    output logic [3:0] COL,
    output logic [0:6] HEX,
    output logic [3:0] CAT
);
    logic [15:0] OUT;

    keypad_input INPUT
    (
        .clk(CLK),
        .reset(CLR),
        .row(ROW),
        .BASE(BASE),
        .col(COL),
        .out(OUT)
    );


    Controller Mux    // Display Stuff
    (
        .CLK(CLK),
        .CLEAR(CLR),
        .MODE(1'b1),
        .D0(OUT[3:0]),
        .D1(OUT[7:4]),
        .D2(OUT[11:8]),
        .D3(OUT[15:12]),
        .CAT(CAT),
        .HEX(HEX)
    );

endmodule
```

Keypad Input



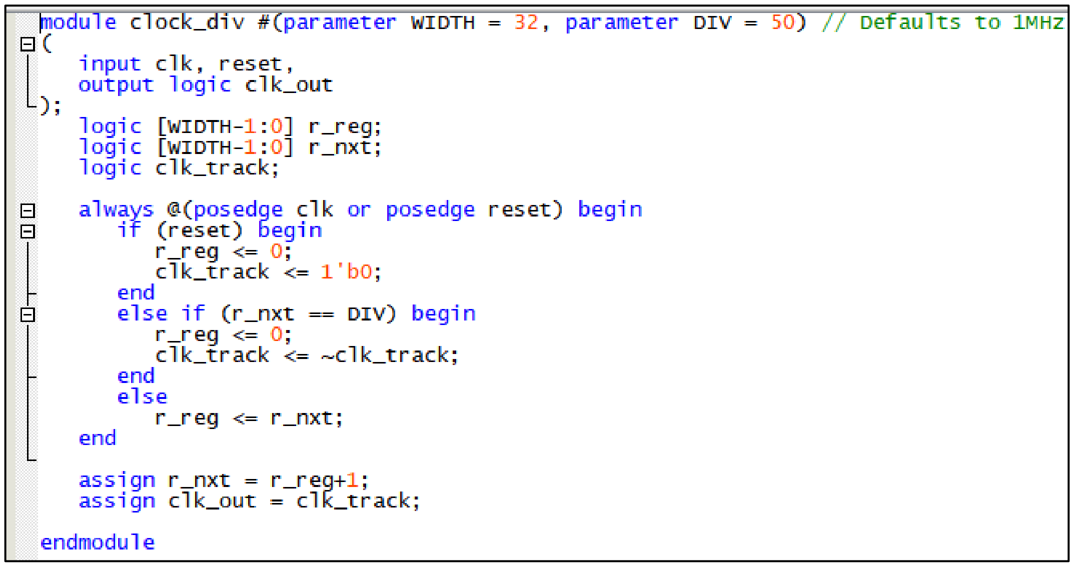
```
module keypad_input #( parameter DIGITS = 4)
(
    input clk,
    input reset,
    input [3:0] row,
    input BASE,
    output [3:0] col,
    output [(DIGITS*4)-1:0] out,
    // Debug
    output logic [3:0] value,
    output logic trig
);
    keypad_base keypad_base
    (
        .clk(clk),
        .row(row),
        .BASE(BASE),
        .col(col),
        .value(value),
        .valid(trig)
    );

    shift_reg #(.COUNT(DIGITS)) shift_reg
    (
        .trig(trig),
        .in(value),
        .out(out),
        .reset(reset)
    );

endmodule
```

Keypad Base Module



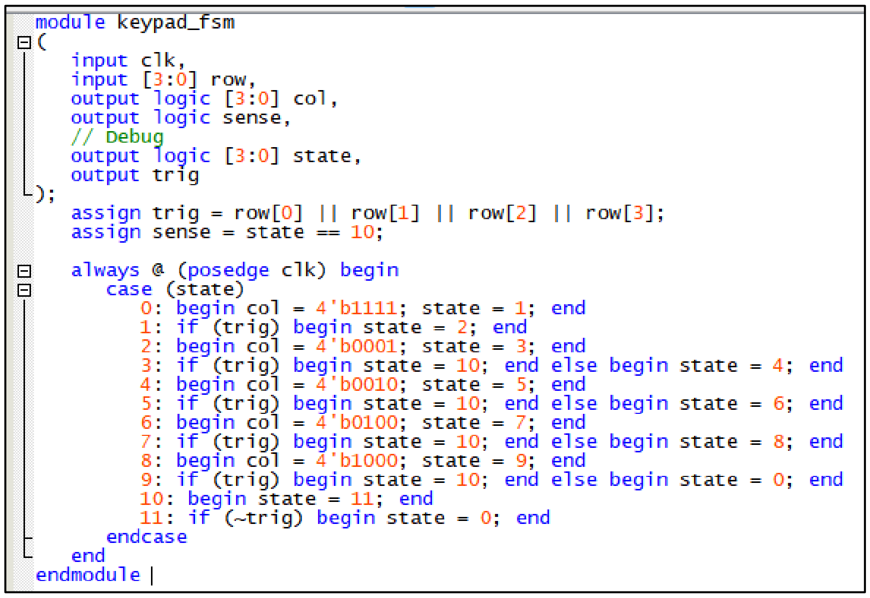
```
module keypad_base // Depends: clock_div, keypad_fsm, keypad_decoder
(
   input clk,
   input [3:0] row,
   input BASE,
   output [3:0] col,
   output [3:0] value,
   output valid,
   // Debug
   output logic slow_clock,
   output logic sense,
   output logic valid_digit,
   output logic [3:0] inv_row
);
   assign inv_row = ~row;

   clock_div #(.DIV(100000)) keypad_clock_divider
   (
      .clk(clk),
      .clk_out(slow_clock)
   );

   keypad_fsm keypad_fsm
   (
      .clk(slow_clock),
      .row(inv_row),
      .col(col),
      .sense(sense)
   );

   keypad_decoder #(.BASE(10)) keypad_key_decoder
   (
      .row(inv_row),
      .col(col),
      .base(BASE),
      .value(value),
      .valid(valid_digit)
   );

   assign valid = valid_digit && sense;

endmodule
```

Clock Divider Module



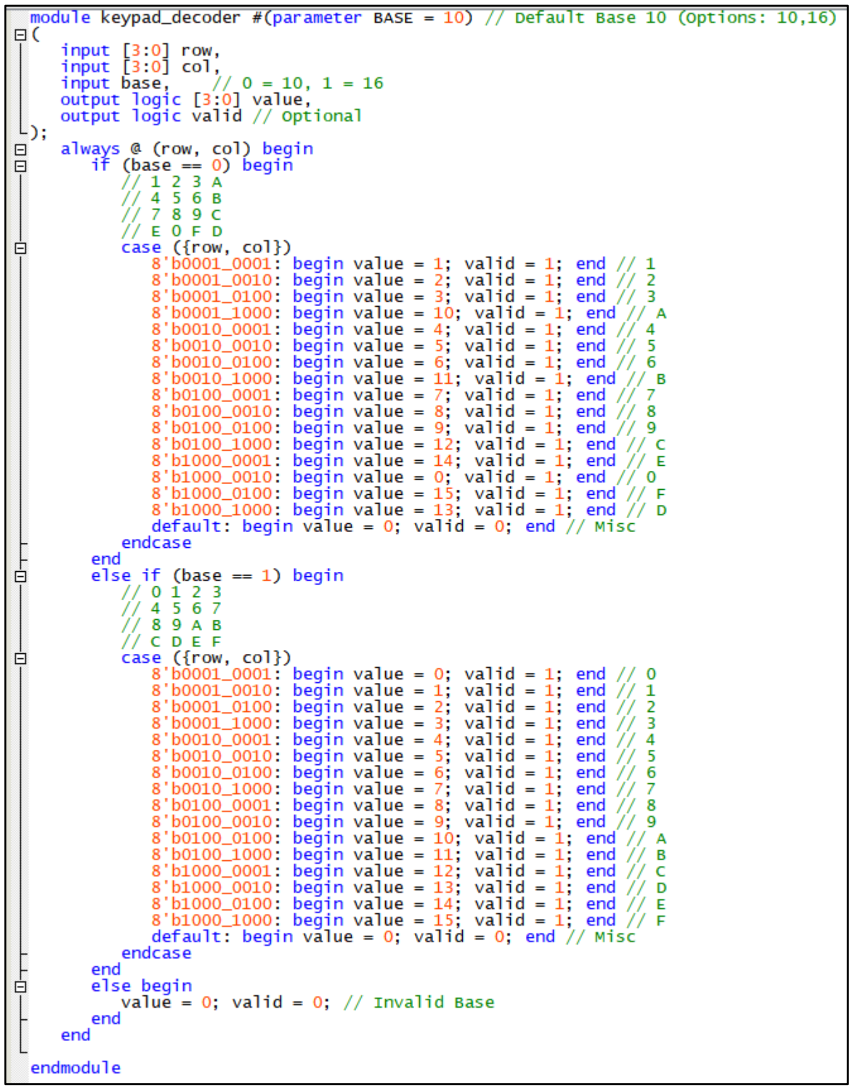
```
module clock_div #(parameter WIDTH = 32, parameter DIV = 50) // Defaults to 1MHz
(
   input clk, reset,
   output logic clk_out
);
   logic [WIDTH-1:0] r_reg;
   logic [WIDTH-1:0] r_nxt;
   logic clk_track;

   always @(posedge clk or posedge reset) begin
      if (reset) begin
         r_reg <= 0;
         clk_track <= 1'b0;
      end
      else if (r_nxt == DIV) begin
         r_reg <= 0;
         clk_track <= ~clk_track;
      end
      else
         r_reg <= r_nxt;
   end

   assign r_nxt = r_reg+1;
   assign clk_out = clk_track;

endmodule
```

Keypad FSM Module



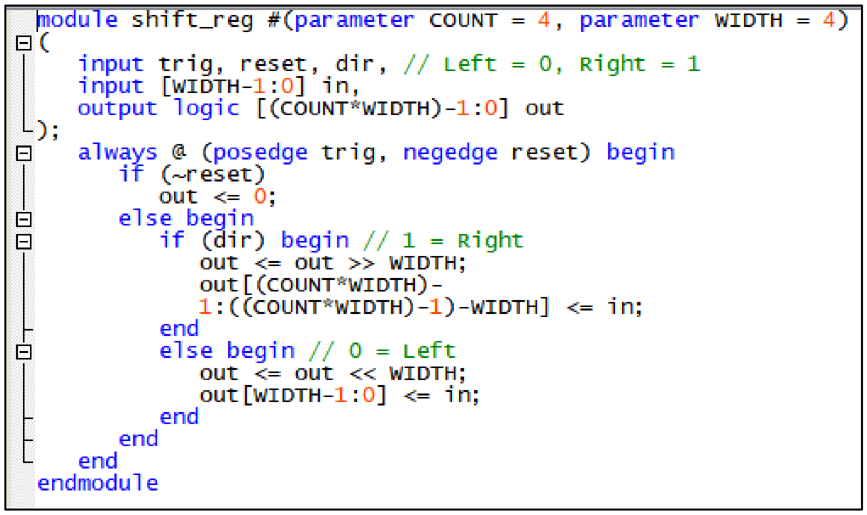
```
module keypad_fsm
(
    input clk,
    input [3:0] row,
    output logic [3:0] col,
    output logic sense,
    // Debug
    output logic [3:0] state,
    output trig
);
    assign trig = row[0] || row[1] || row[2] || row[3];
    assign sense = state == 10;

    always @ (posedge clk) begin
        case (state)
            0: begin col = 4'b1111; state = 1; end
            1: if (trig) begin state = 2; end
            2: begin col = 4'b0001; state = 3; end
            3: if (trig) begin state = 10; end else begin state = 4; end
            4: begin col = 4'b0010; state = 5; end
            5: if (trig) begin state = 10; end else begin state = 6; end
            6: begin col = 4'b0100; state = 7; end
            7: if (trig) begin state = 10; end else begin state = 8; end
            8: begin col = 4'b1000; state = 9; end
            9: if (trig) begin state = 10; end else begin state = 0; end
            10: begin state = 11; end
            11: if (~trig) begin state = 0; end
        endcase
    end
endmodule |
```

Keypad Key Decoder Module



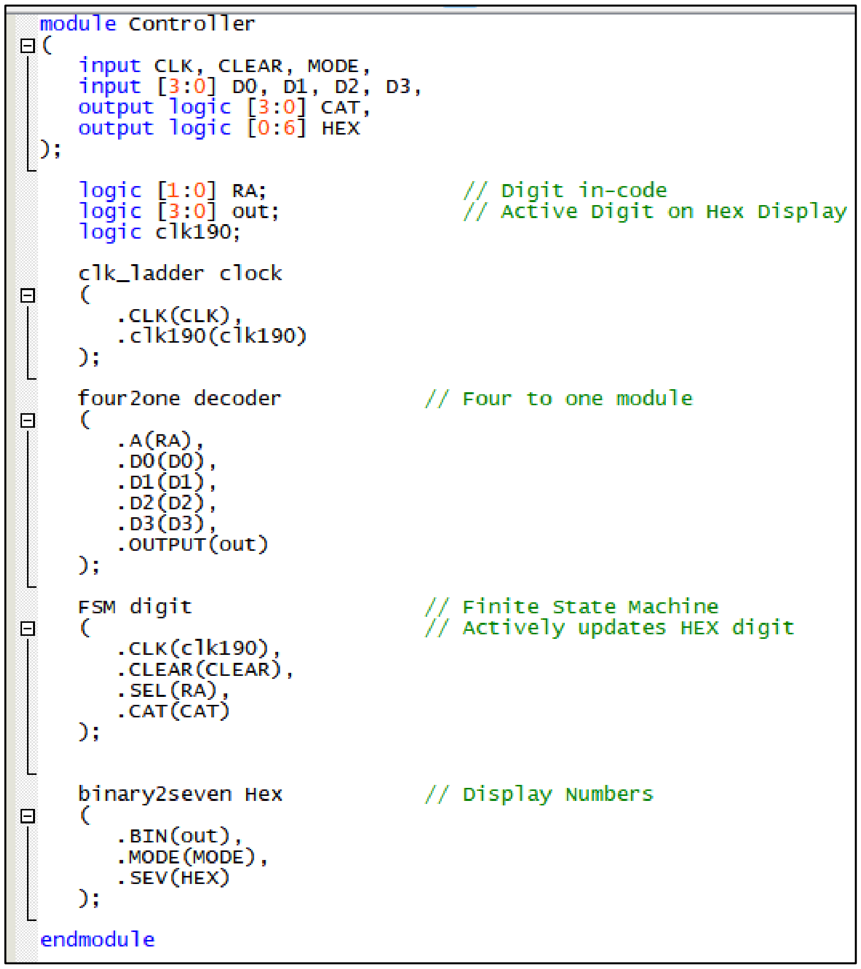
```
module keypad_decoder #(parameter BASE = 10) // Default Base 10 (Options: 10,16)
(
    input [3:0] row,
    input [3:0] col,
    input base,     // 0 = 10, 1 = 16
    output logic [3:0] value,
    output logic valid // Optional
);
    always @ (row, col) begin
        if (base == 0) begin
            // 1 2 3 A
            // 4 5 6 B
            // 7 8 9 C
            // E 0 F D
            case ({row, col})
                8'b0001_0001: begin value = 1; valid = 1; end // 1
                8'b0001_0010: begin value = 2; valid = 1; end // 2
                8'b0001_0100: begin value = 3; valid = 1; end // 3
                8'b0001_1000: begin value = 10; valid = 1; end // A
                8'b0010_0001: begin value = 4; valid = 1; end // 4
                8'b0010_0010: begin value = 5; valid = 1; end // 5
                8'b0010_0100: begin value = 6; valid = 1; end // 6
                8'b0010_1000: begin value = 11; valid = 1; end // B
                8'b0100_0001: begin value = 7; valid = 1; end // 7
                8'b0100_0010: begin value = 8; valid = 1; end // 8
                8'b0100_0100: begin value = 9; valid = 1; end // 9
                8'b0100_1000: begin value = 12; valid = 1; end // C
                8'b1000_0001: begin value = 14; valid = 1; end // E
                8'b1000_0010: begin value = 0; valid = 1; end // 0
                8'b1000_0100: begin value = 15; valid = 1; end // F
                8'b1000_1000: begin value = 13; valid = 1; end // D
                default: begin value = 0; valid = 0; end // Misc
            endcase
        end
        else if (base == 1) begin
            // 0 1 2 3
            // 4 5 6 7
            // 8 9 A B
            // C D E F
            case ({row, col})
                8'b0001_0001: begin value = 0; valid = 1; end // 0
                8'b0001_0010: begin value = 1; valid = 1; end // 1
                8'b0001_0100: begin value = 2; valid = 1; end // 2
                8'b0001_1000: begin value = 3; valid = 1; end // 3
                8'b0010_0001: begin value = 4; valid = 1; end // 4
                8'b0010_0010: begin value = 5; valid = 1; end // 5
                8'b0010_0100: begin value = 6; valid = 1; end // 6
                8'b0010_1000: begin value = 7; valid = 1; end // 7
                8'b0100_0001: begin value = 8; valid = 1; end // 8
                8'b0100_0010: begin value = 9; valid = 1; end // 9
                8'b0100_0100: begin value = 10; valid = 1; end // A
                8'b0100_1000: begin value = 11; valid = 1; end // B
                8'b1000_0001: begin value = 12; valid = 1; end // C
                8'b1000_0010: begin value = 13; valid = 1; end // D
                8'b1000_0100: begin value = 14; valid = 1; end // E
                8'b1000_1000: begin value = 15; valid = 1; end // F
                default: begin value = 0; valid = 0; end // Misc
            endcase
        end
        else begin
            value = 0; valid = 0; // Invalid Base
        end
    end

endmodule
```

Shift Register Module



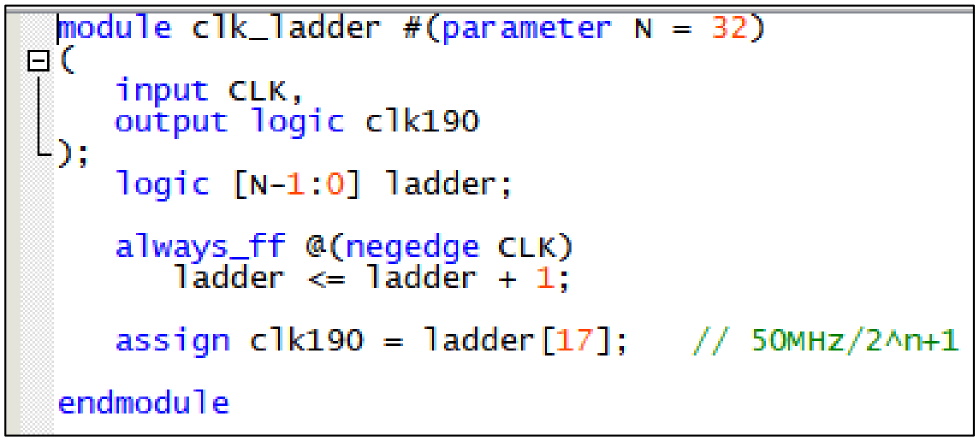
```
module shift_reg #(parameter COUNT = 4, parameter WIDTH = 4)
(
   input trig, reset, dir, // Left = 0, Right = 1
   input [WIDTH-1:0] in,
   output logic [(COUNT*WIDTH)-1:0] out
);
   always @ (posedge trig, negedge reset) begin
      if (~reset)
         out <= 0;
      else begin
         if (dir) begin // 1 = Right
            out <= out >> WIDTH;
            out[(COUNT*WIDTH)-
            1:((COUNT*WIDTH)-1)-WIDTH] <= in;
         end
         else begin // 0 = Left
            out <= out << WIDTH;
            out[WIDTH-1:0] <= in;
         end
      end
   end
endmodule
```

Controller/MUX Module



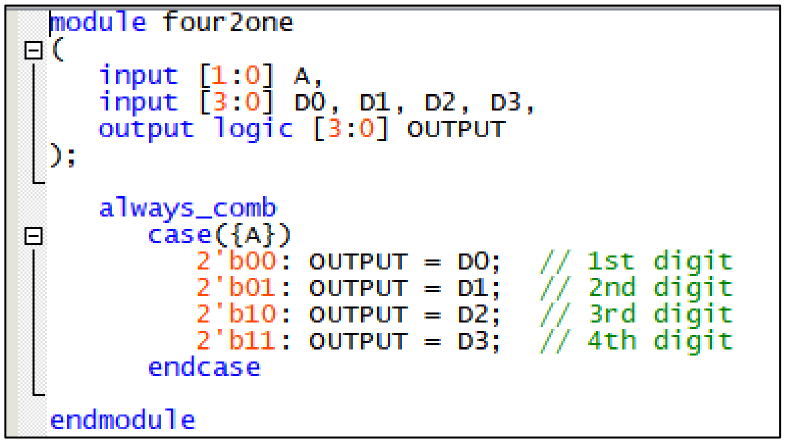
```
module Controller
(
   input CLK, CLEAR, MODE,
   input [3:0] D0, D1, D2, D3,
   output logic [3:0] CAT,
   output logic [0:6] HEX
);

   logic [1:0] RA;                // Digit in-code
   logic [3:0] out;               // Active Digit on Hex Display
   logic clk190;

   clk_ladder clock
   (
      .CLK(CLK),
      .clk190(clk190)
   );

   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
   (
      .BIN(out),
      .MODE(MODE),
      .SEV(HEX)
   );
endmodule
```

Clock Ladder Module



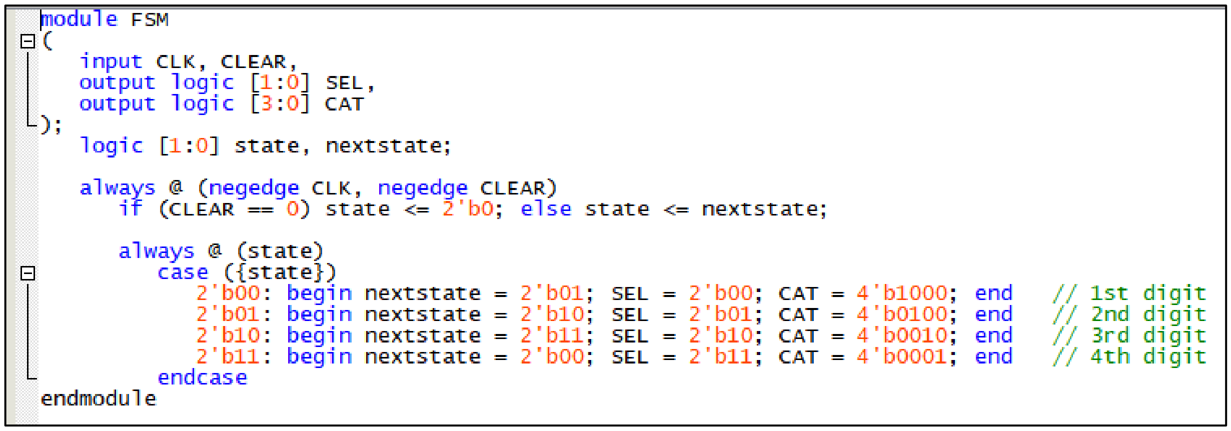
```
module clk_ladder #(parameter N = 32)
(
   input CLK,
   output logic clk190
);
   logic [N-1:0] ladder;

   always_ff @(negedge CLK)
      ladder <= ladder + 1;

   assign clk190 = ladder[17];    // 50MHz/2^n+1

endmodule
```

Four to One Decoder Module



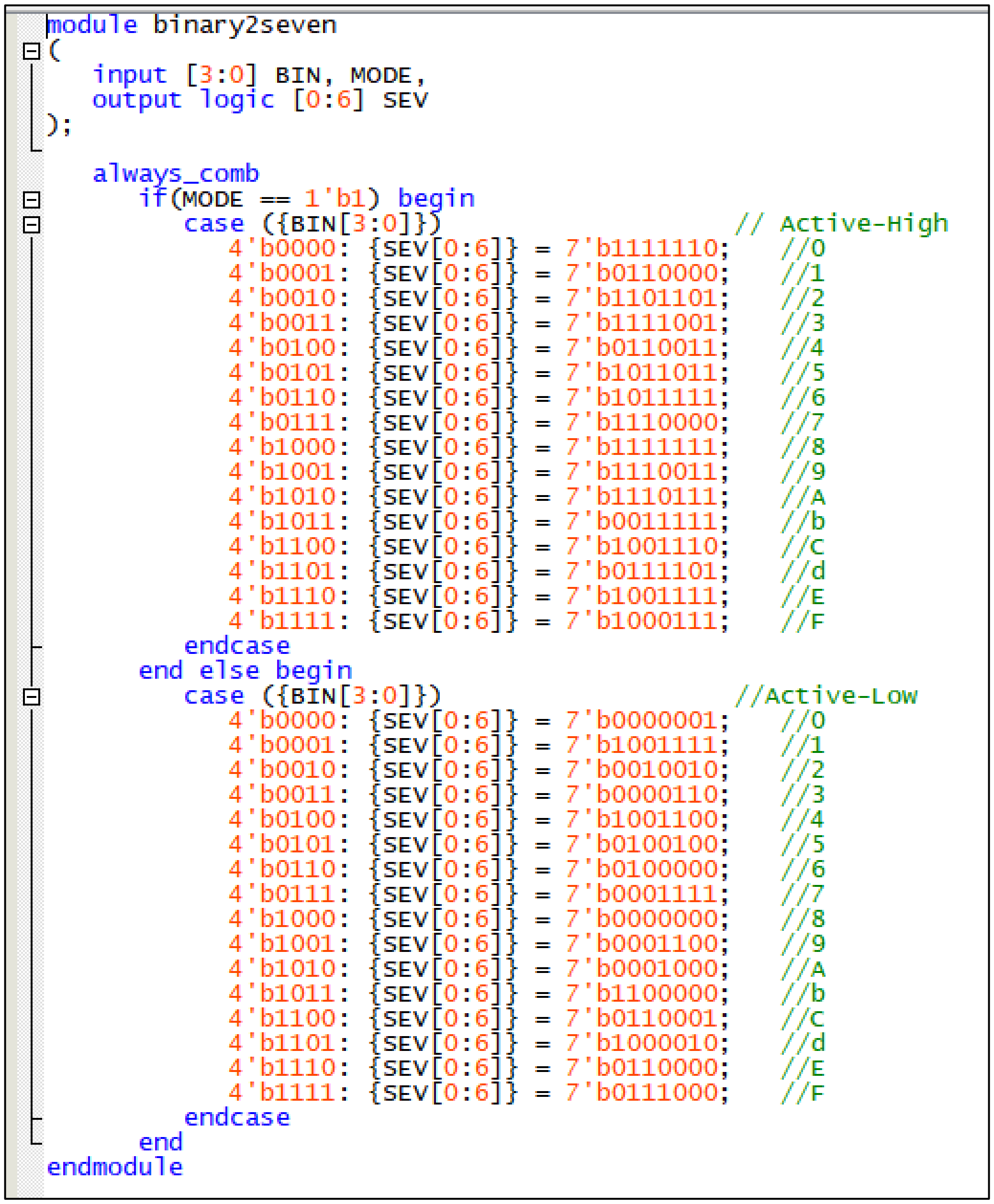
```
module four2one
(
   input [1:0] A,
   input [3:0] D0, D1, D2, D3,
   output logic [3:0] OUTPUT
);

   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



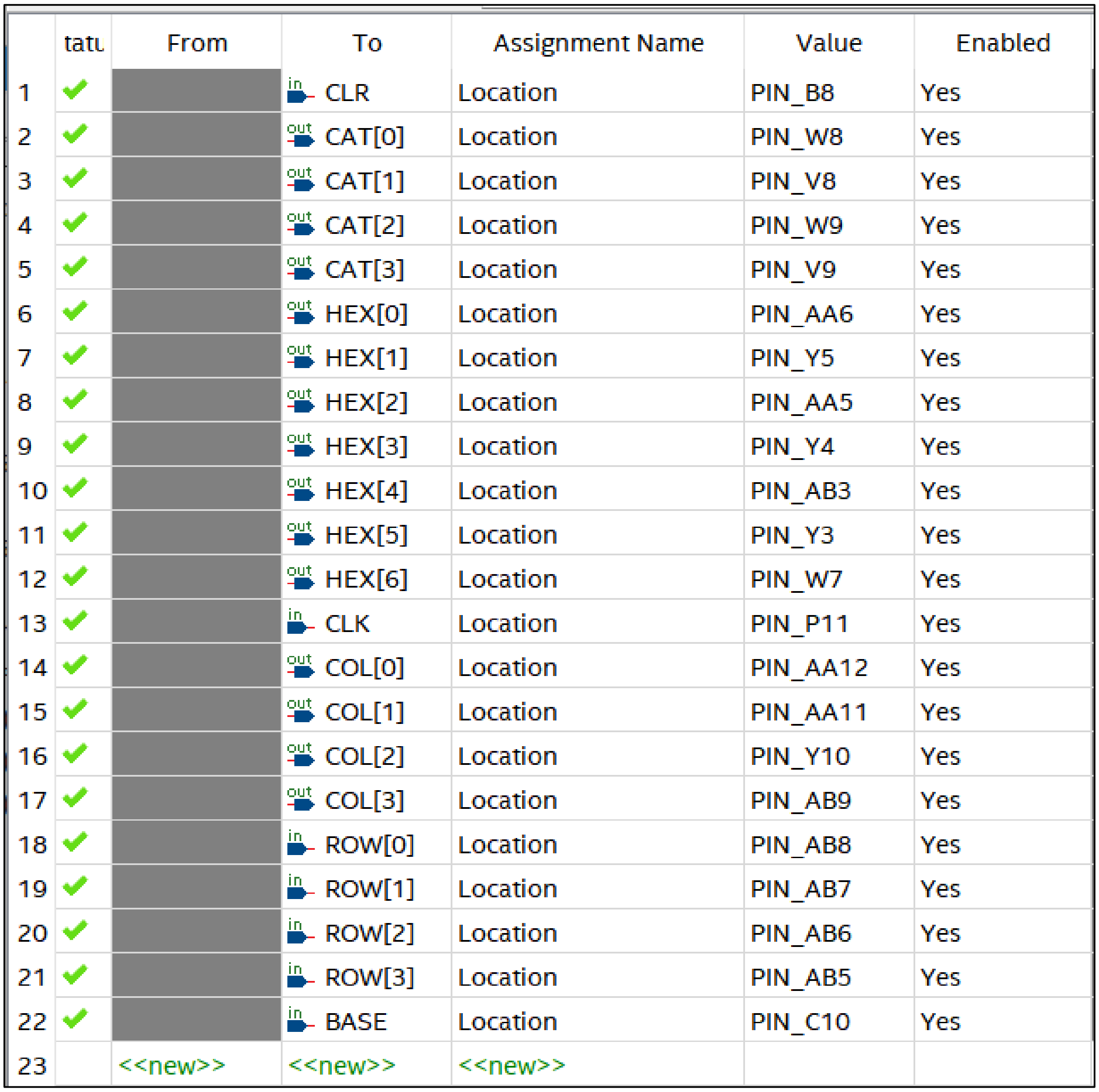
```
module FSM
(
   input CLK, CLEAR,
   output logic [1:0] SEL,
   output logic [3:0] CAT
);
   logic [1:0] state, nextstate;

   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
```

Four to One Decoder Module



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

    always_comb
        if(MODE == 1'b1) begin
            case ({BIN[3:0]})                        // Active-High
                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]})                        //Active-Low
                4'b0000: {SEV[0:6]} = 7'b0000001;    //0
                4'b0001: {SEV[0:6]} = 7'b1001111;    //1
                4'b0010: {SEV[0:6]} = 7'b0010010;    //2
                4'b0011: {SEV[0:6]} = 7'b0000110;    //3
                4'b0100: {SEV[0:6]} = 7'b1001100;    //4
                4'b0101: {SEV[0:6]} = 7'b0100100;    //5
                4'b0110: {SEV[0:6]} = 7'b0100000;    //6
                4'b0111: {SEV[0:6]} = 7'b0001111;    //7
                4'b1000: {SEV[0:6]} = 7'b0000000;    //8
                4'b1001: {SEV[0:6]} = 7'b0001100;    //9
                4'b1010: {SEV[0:6]} = 7'b0001000;    //A
                4'b1011: {SEV[0:6]} = 7'b1100000;    //b
                4'b1100: {SEV[0:6]} = 7'b0110001;    //C
                4'b1101: {SEV[0:6]} = 7'b1000010;    //d
                4'b1110: {SEV[0:6]} = 7'b0110000;    //E
                4'b1111: {SEV[0:6]} = 7'b0111000;    //F
            endcase
        end
endmodule
```

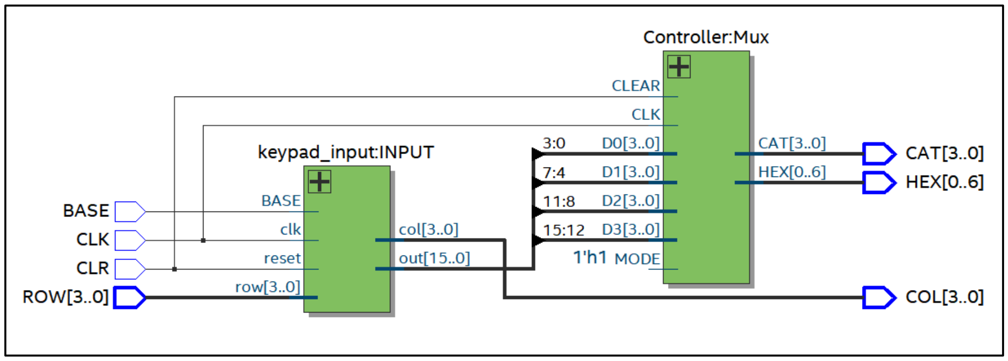
PIN ASSIGMENTS



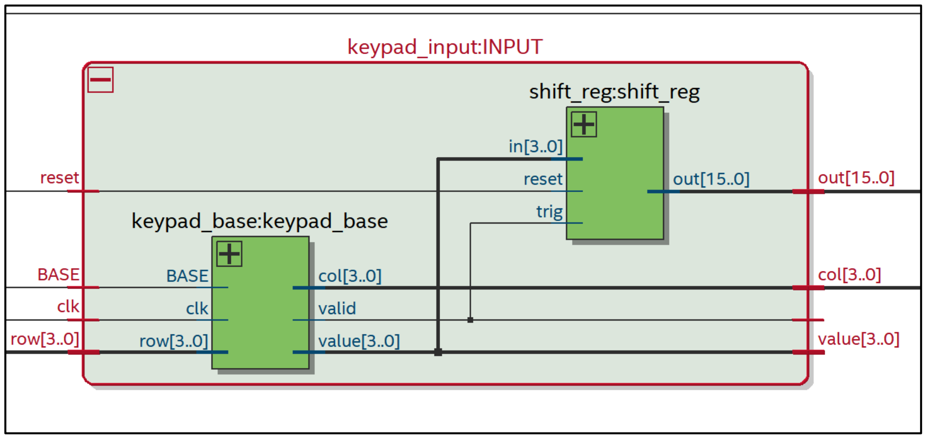
	tatu	From	To	Assignment Name	Value	Enabled
1	✓		CLR	Location	PIN_B8	Yes
2	✓		CAT[0]	Location	PIN_W8	Yes
3	✓		CAT[1]	Location	PIN_V8	Yes
4	✓		CAT[2]	Location	PIN_W9	Yes
5	✓		CAT[3]	Location	PIN_V9	Yes
6	✓		HEX[0]	Location	PIN_AA6	Yes
7	✓		HEX[1]	Location	PIN_Y5	Yes
8	✓		HEX[2]	Location	PIN_AA5	Yes
9	✓		HEX[3]	Location	PIN_Y4	Yes
10	✓		HEX[4]	Location	PIN_AB3	Yes
11	✓		HEX[5]	Location	PIN_Y3	Yes
12	✓		HEX[6]	Location	PIN_W7	Yes
13	✓		CLK	Location	PIN_P11	Yes
14	✓		COL[0]	Location	PIN_AA12	Yes
15	✓		COL[1]	Location	PIN_AA11	Yes
16	✓		COL[2]	Location	PIN_Y10	Yes
17	✓		COL[3]	Location	PIN_AB9	Yes
18	✓		ROW[0]	Location	PIN_AB8	Yes
19	✓		ROW[1]	Location	PIN_AB7	Yes
20	✓		ROW[2]	Location	PIN_AB6	Yes
21	✓		ROW[3]	Location	PIN_AB5	Yes
22	✓		BASE	Location	PIN_C10	Yes
23		<<new>>	<<new>>	<<new>>		

RTL DIAGRAMS

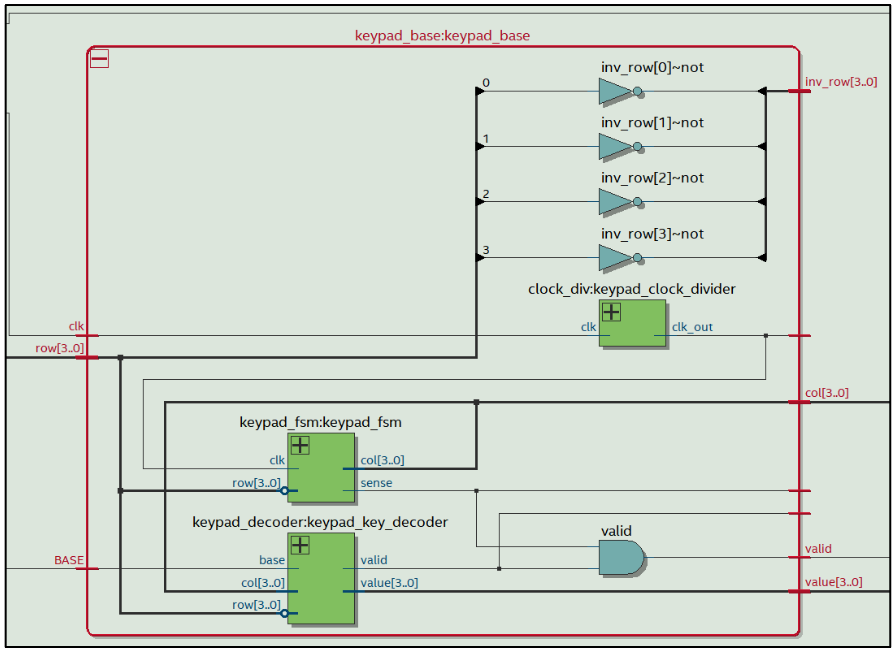
Top Module



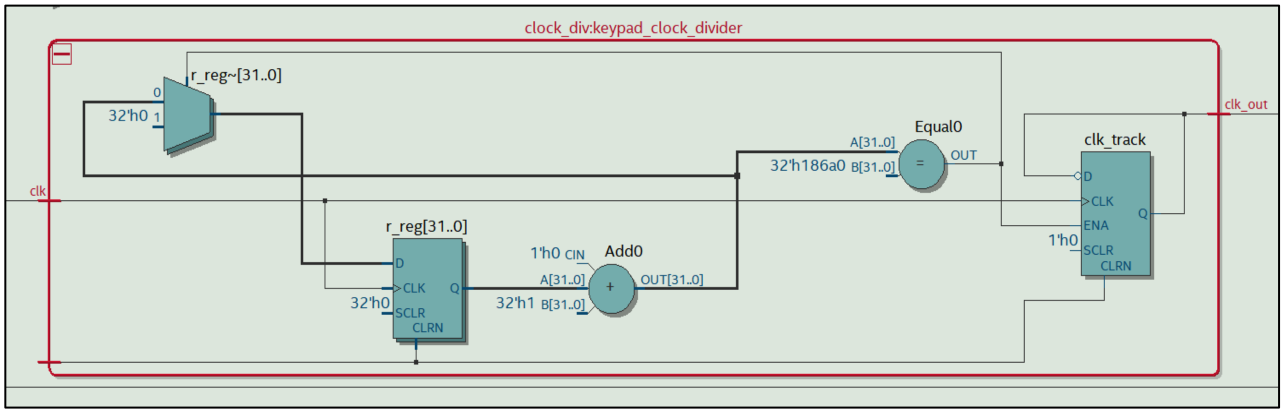
Keypad Input



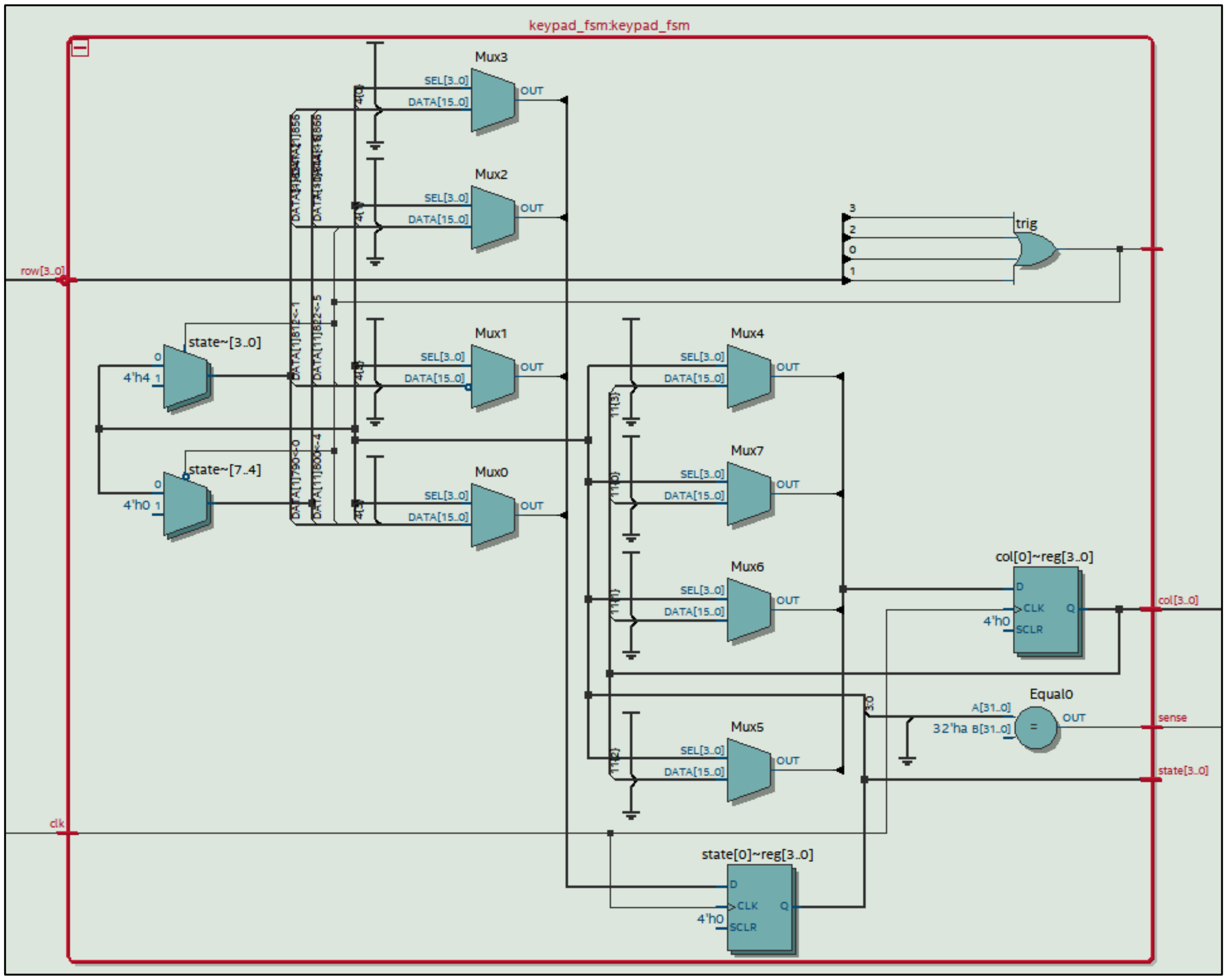
Keypad Base



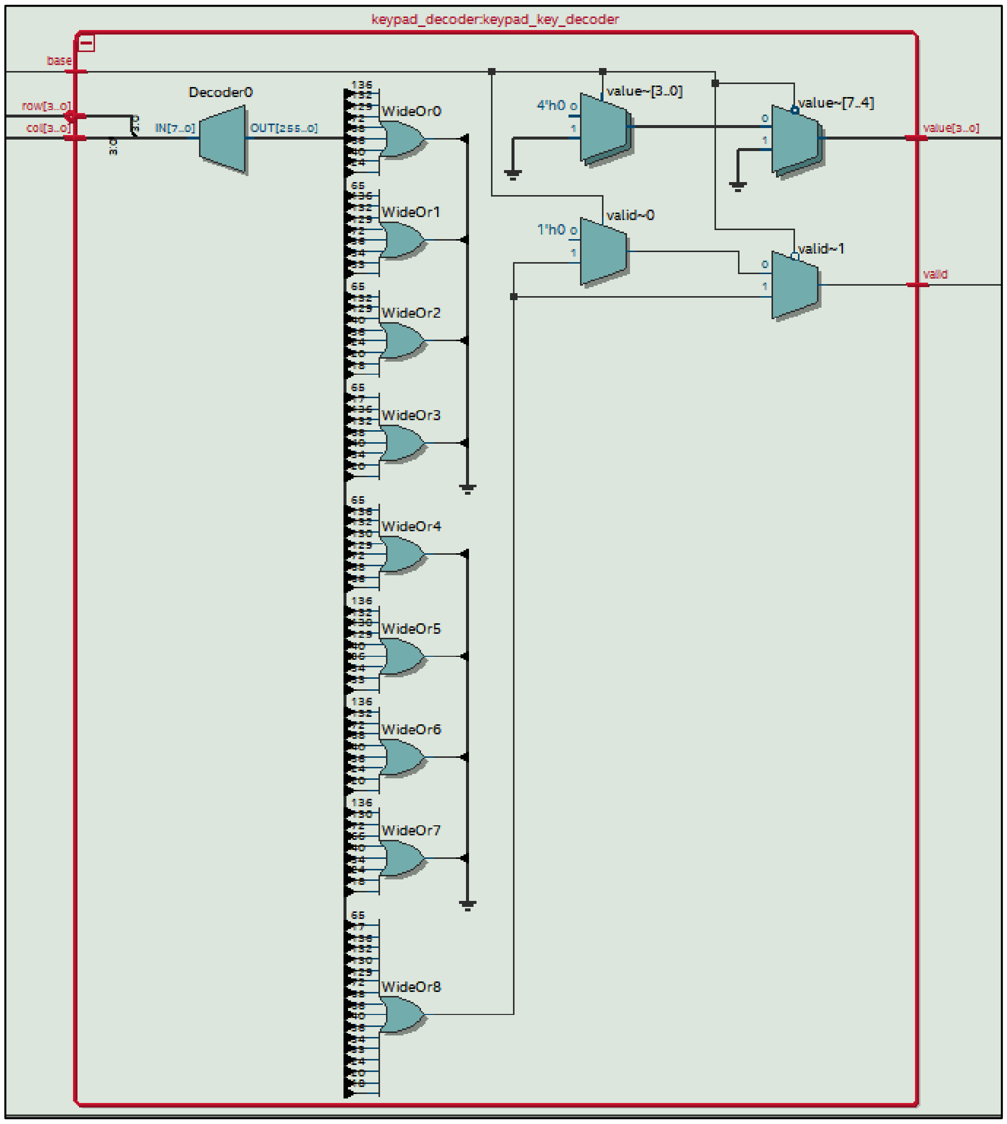
Clock Divider



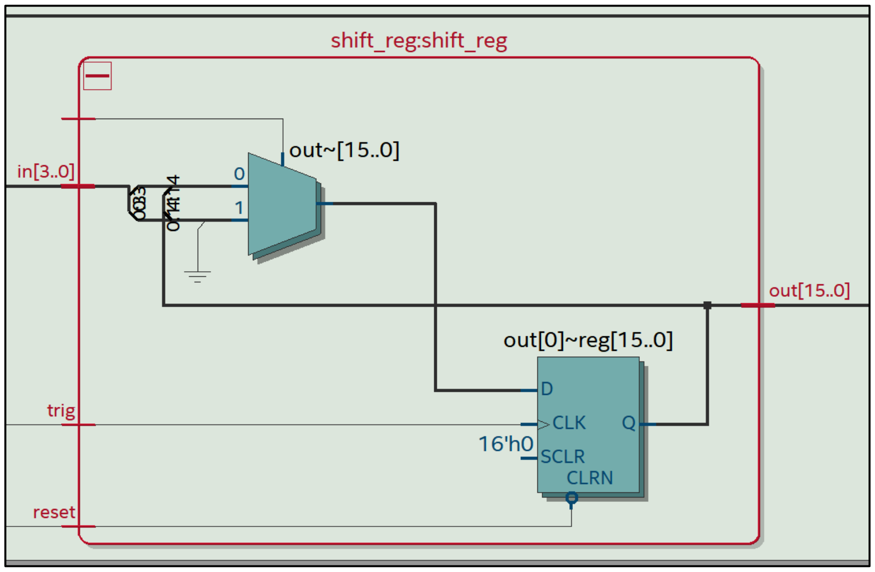
Keypad Finite State Machine



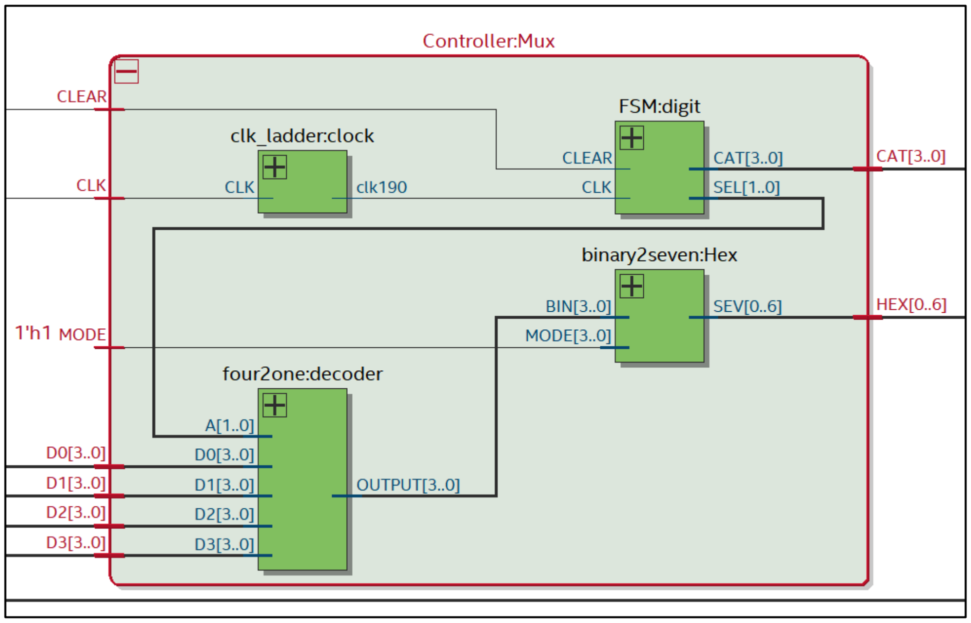
Keypad Key Decoder



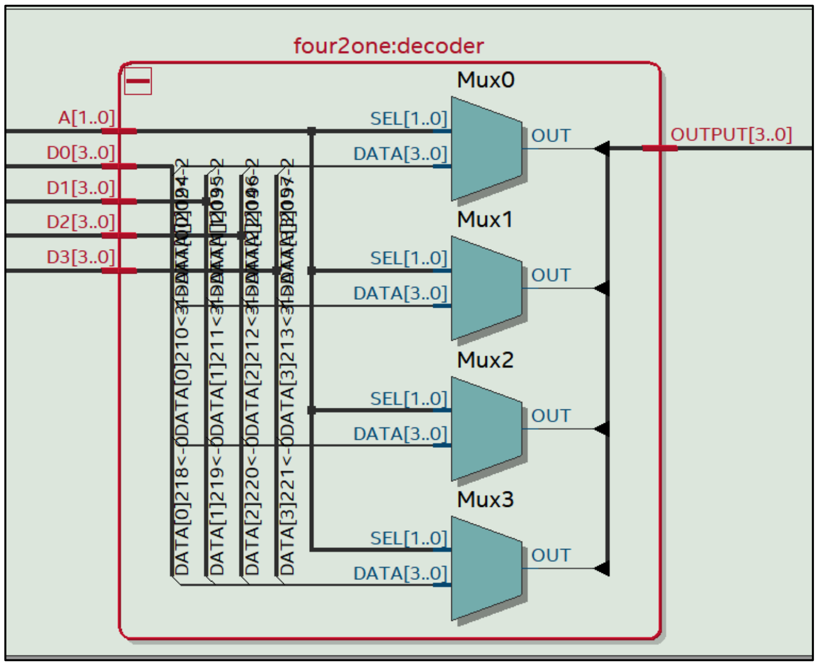
Shift Register



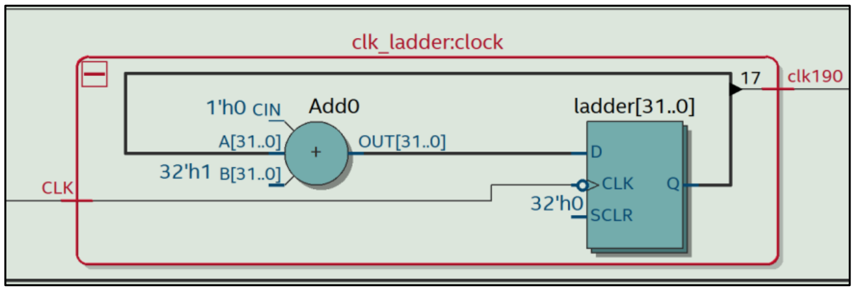
Controller/MUX



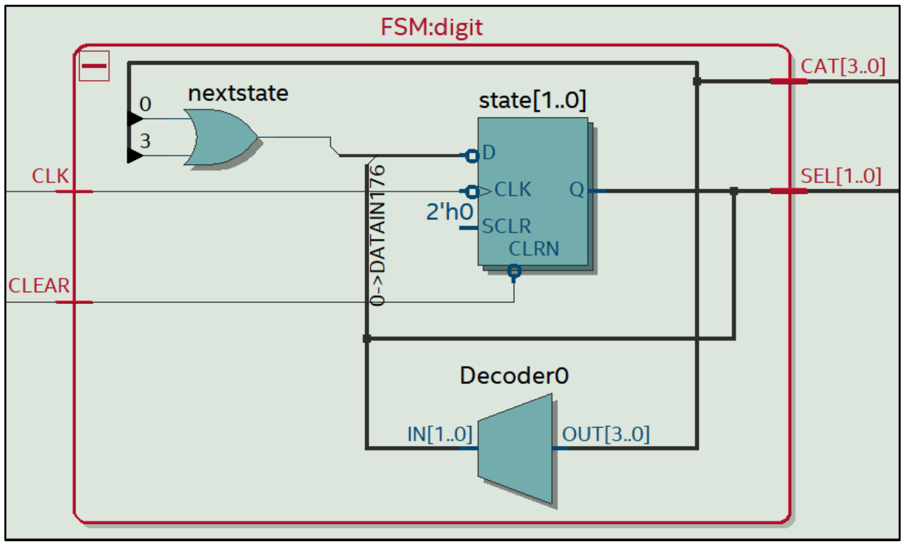
Four to One Decoder



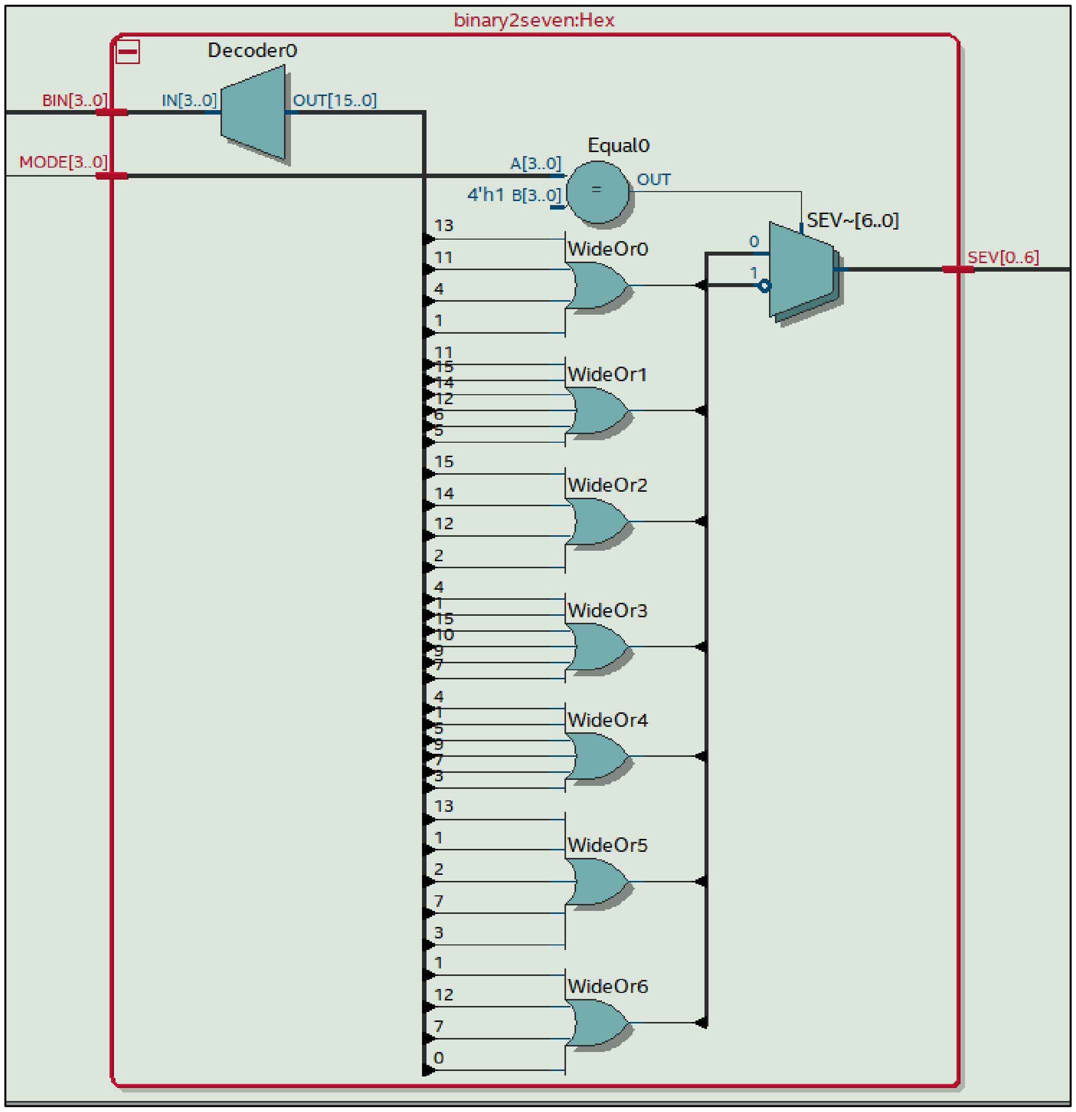
Clock Ladder



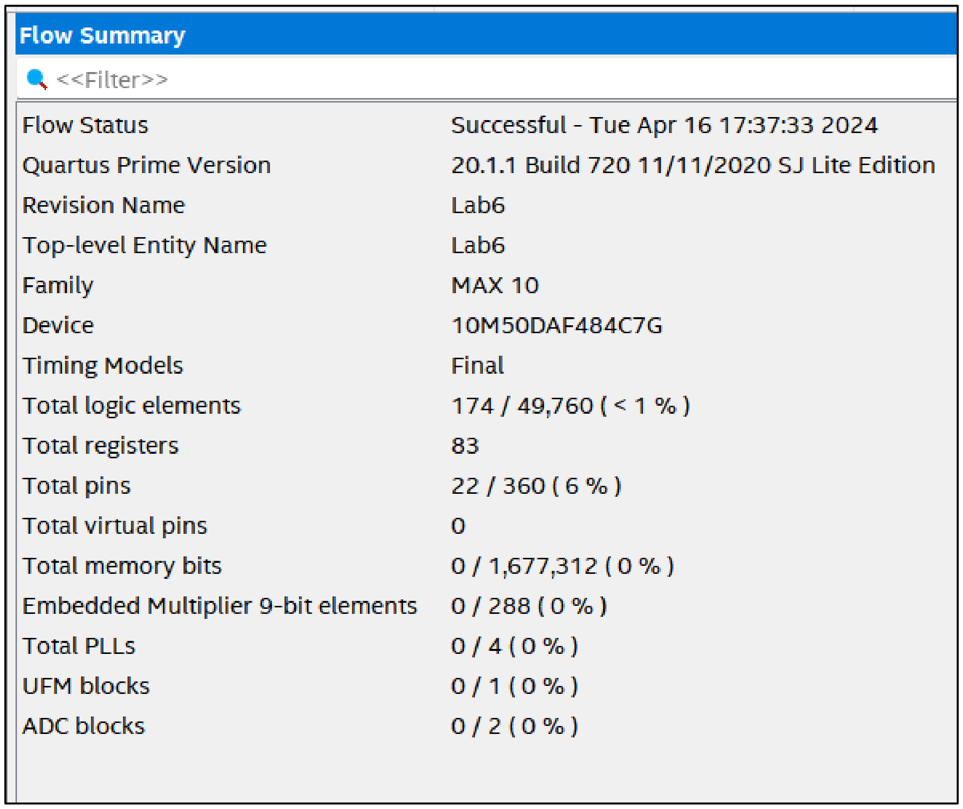
Finite State Machine



Binary To Seven Hex Decoder



Compilation Summary



Flow Summary

<<Filter>>

Flow Status	Successful - Tue Apr 16 17:37:33 2024
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	Lab6
Top-level Entity Name	Lab6
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	174 / 49,760 (< 1 %)
Total registers	83
Total pins	22 / 360 (6 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

DEMOSTRATION OF DCB KEYPAD SCANNER

Demoed In person to TA.

DEMOSTRATION OF HEX KEYPAD SCANNER

Demoed In person to TA.