Name: _____Servando_Olvera_____ ID# _____1001909287_____

Date Submitted: _____04-04-2024____ Time Submitted _____7:00_pm_____

CSE 3341 Digital Logic Design II

CSE 5357 Advanced Digital Logic Design

Spring Semester 2024

Lab 5 – Eight-Bit Divider

150 points

Due Date – April 4, 2024, 11:59 PM

Submit on Canvas Assignments

Note – Late submissions will not be accepted!

USIGNED VERISON

DESIGN REQUIREMENTS

Your assignment is to design a registered eight-bit divider that has the input/output diagram shown shown in class, and incorporates the non-restoring divider also discussed in class. Verilog code for the divider is posted on Canvas in the Reference Materials module.

DESIGN REQUIREMENTS

1. Eight-bit divider
2. Unsigned numbers
3. Verilog implementation
4. Design verification (simulation)
5. DE10-Lite realization

DESIGN PROCESS

1. Design, implement, verify, and realize an unsigned version. (150 points)
2. Design, implement, verify, and realize a signed version. Extra credit (50 points).

DESIGN VERIFICATION

1. Simulate your design to verify its correctness. Use the following values of A and B in your simulations.

    (a) 0FFF ÷ FF Display in hexadecimal on the HEX displays.
    (b) 0FFF ÷ EE
    (c) 00AC ÷ FF
    (d) 0067 ÷ 67
    (e) 0000 ÷ 0A

2. Include a screen shot of your simulation waveforms in your report.
3. Record the simulation results in a table for your report (use hexadecimal)
4. How many clock cycles does it take each case to complete? What would be the corresponding divide times for your fastest clock?

RTL ANALYSIS

1. Generate RTL diagrams using the Quartus Prime Netlist Viewer.
2. Record the compilation summary for your report. How many ALM, registers, and pins does your design require?

DE10-Lite IMPLEMENTATION (each version)

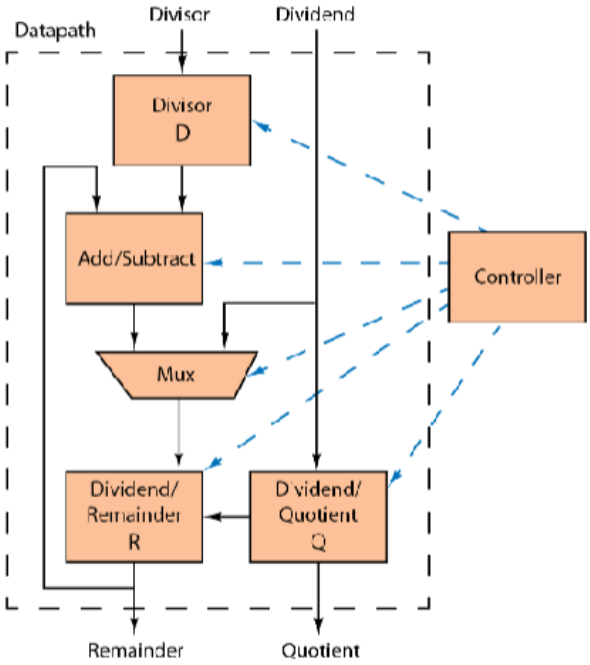
1. Implement your design on the DE10-Lite using the following inputs/outputs using pin assignments of your choice.
Inputs A, B, Load A, Load B, Start, Clock (50 MHz), Reset
Outputs Aout, Bout, Q, R, Done.
Display Aout, Bout, A, B, Q, and R in hexadecimal on the HEX displays.
2. Include a table of your assignments in your report.
3. Program the DE10-Lite with your design.

Note: Since it was required to display A, B, Q & R in the same HEX display I deviated from the inputs/outputs listed above.

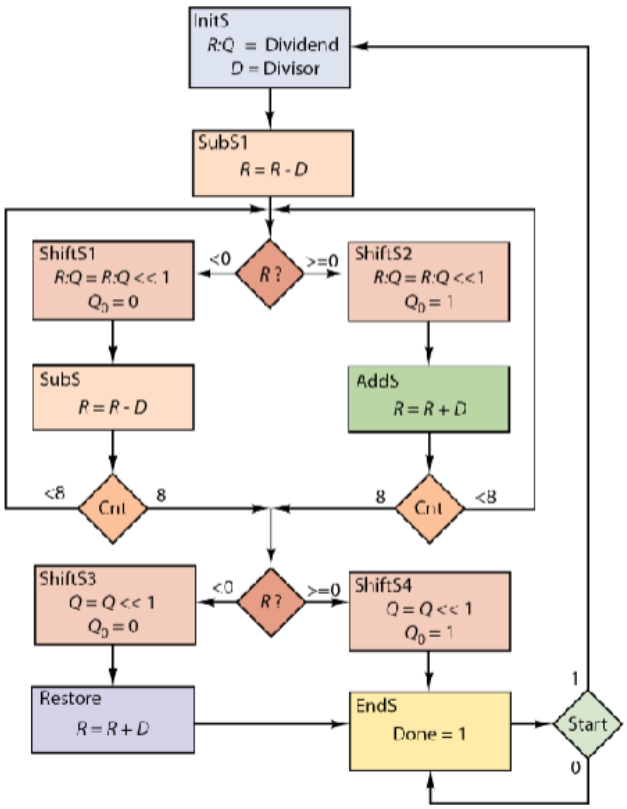
DE10-Lite TESTING

1. Test your implementations by applying the same patterns as above.
2. Record the test results in a table for your report.
3. Take a picture (or video) of your results for your report.

DATA PATH DIAGRAM



CONTROL PATH DIAGRAM



SYSTEM-VERILOG CODE

Top Module

```
module Lab5
(
    input CLK, CLR, LOAD_NEXT,
    input [7:0] X,
    output logic [0:6] HEX,
    output logic [3:0] CAT
);

    logic [15:0] A;    // Dividend
    logic [7:0] B;     // Divisor

    logic [7:0] Quotient;
    logic [7:0] Remainder;
    logic [15:0] Quo_Rem, OUT;
    logic [3:0] load_val;
    logic DONE;

    StateMachine A_B_Q_R_Display  // Chose value to Display
    (                             //
       .CLK(LOAD_NEXT),           // Dividend
       .CLEAR(CLR),               // Divisor
       .A(A),                     // Quotient, Remainder
       .B(B),
       .Q_R(Quo_Rem),
       .display(OUT),
       .ins(load_val)
    );

    NBitRegister DIVIDEND_2       // Upper Half of Dividend
    (
       .D(X) ,
       .CLK(~load_val[0]) ,
       .CLR(CLR),
       .Q(A[15:8])
    );

    NBitRegister DIVIDEND_1       // Lowr half of Dividend
    (
       .D(X) ,
       .CLK(~load_val[1]) ,
       .CLR(CLR),
       .Q(A[7:0])
    );


    NBitRegister DIVISOR          // Divisor
    (
       .D(X) ,
       .CLK(~load_val[2]) ,
       .CLR(CLR),
       .Q(B)
    );

    Divider DIVIDE
    (
       .Dividend(A),
       .Divisor(B),
       .Quotient(Quotient),
       .Remainder(Remainder),
       .CLOCK(CLK),
       .START(~load_val[3]),
       .DONE(DONE)
    );
```

```
    NBitRegister QUOTIENT         // Quotient
    (
       .D(Quotient),
       .CLK(~DONE),
       .CLR(CLR),
       .Q(Quo_Rem[15:8])
    );

    NBitRegister REMAINDER        // Remainder
    (
       .D(Remainder),
       .CLK(~DONE),
       .CLR(CLR),
       .Q(Quo_Rem[7:0])
    );

    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
```

Register Module

```
module NBitRegister #(parameter N = 8)
(
    input [N-1:0] D,
    input CLK, CLR,
    output logic [N-1:0] Q
);

    always @ (negedge CLK, negedge CLR) begin
        if (CLR == 1'b0)
            Q <= 0;                                 //zero out register
        else if (CLK == 1'b0)
            Q <= D;                                 //data input values loaded in
    end
endmodule
```

Display Different Registers State Machine
(Display Dividend or Divisor or Quotient & Remainder at the push of a button)

```
module StateMachine
(
    input CLK, CLEAR,
    input [7:0] B,
    input [15:0] A, Q_R,
    output logic [15:0] display,
    output logic [3:0] ins
);
    logic [2:0] state, nextstate;

    always @ (negedge CLK, negedge CLEAR)
        if (CLEAR == 0) state <= 3'b0;
        else state <= nextstate;


        always @ (state)
            case ({state})
                3'b000: begin nextstate = 3'b001; display = 16'b0; ins = 4'b0000; end
                3'b001: begin nextstate = 3'b010; display = A;     ins = 4'b0001; end
                3'b010: begin nextstate = 3'b011; display = A;     ins = 4'b0010; end
                3'b011: begin nextstate = 3'b100; display = B;     ins = 4'b0100; end
                3'b100: begin nextstate = 3'b100; display = Q_R;   ins = 4'b1000; end
            endcase
endmodule
```

Divider Module

```
module Divider
(
    input [15:0] Dividend,
    input [7:0] Divisor,
    output logic [7:0] Quotient,
    output logic [7:0] Remainder,
    input CLOCK,
    input START,
    output logic DONE
);
    logic [8:0] alu_out;
    logic alu_cy;
    logic [8:0] mux_out;
    logic [8:0] mux_in;
    logic [8:0] R_out;
    logic [7:0] Q_out;
    logic [7:0] D_out;
    logic Rload;
    logic Qload;
    logic Dload;
    logic Rshift;
    logic Qshift;
    logic AddSub;
    logic Qbit;

    assign Remainder = R_out[7:0];
    assign mux_in = {1'b0, Dividend[15:8]};
    assign Quotient = Q_out;

    MUX #(9) Mux1 (alu_out, mux_in, mux_out, Qload);

    shiftreg #(9) Rreg (mux_out, R_out, CLOCK, Rload, Rshift, Q_out[7]);
    shiftreg #(8) Qreg (Dividend[7:0], Q_out, CLOCK, Qload, Qshift, Qbit);
    shiftreg #(8) Dreg (Divisor, D_out, CLOCK, Dload, 1'b0, 1'b0);

    alu #(8) AdSb (R_out, D_out, alu_out, AddSub);

    D_Control DivCtrl (CLOCK, START, alu_out[8], AddSub, Dload, Rload, Qload, Rshift, Qshift, DONE, Qbit);

endmodule
```

Divider Control Module

```
module D_Control
(
    input Clock,         //active-high clock
    input Start,         // start pulse
    input Rsign,         // sign from alu op
    output logic AddSub, // select add/subtract
    output logic Dload,  // enable load D register
    output logic Rload,  // enable load R register
    output logic Qload,  // enable load Q register
    output logic Rshift, // enable R reg shift
    output logic Qshift, // enable Q reg shift
    output logic DONE,   // alorithm done indicator
    output logic Qbit
);

    // State definitions
    parameter Inits = 4'h0;
    parameter Adds = 4'h1;
    parameter Subs1 = 4'h2;
    parameter Subs = 4'h3;
    parameter Shifts1 = 4'h4;
    parameter Shifts2 = 4'h5;
    parameter Shifts3 = 4'h6;
    parameter Shifts4 = 4'h7;
    parameter Restore = 4'h8;
    parameter Ends = 4'h9;

    logic [3:0] State;
    logic [2:0] Count;

    // decode state variable for Moore model outputs
    assign Rload = ((State == Inits) || (State == Subs1) || (State == Subs) || (State == Adds) || (State == Restore)) ? 1'b1 : 1'b0;
    assign Dload = (State == Inits) ? 1'b1 : 1'b0;
    assign Qload = (State == Inits) ? 1'b1 : 1'b0;
    assign AddSub = ((State == Adds) || (State == Restore)) ? 1'b0 : 1'b1;
    assign Rshift = ((State == Shifts1) || (State == Shifts2)) ? 1'b1 : 1'b0;
    assign Qshift = ((State == Shifts1) || (State == Shifts2) || (State == Shifts3) || (State == Shifts4)) ? 1'b1 : 1'b0;
    assign Qbit = ((State == Shifts1) || (State == Shifts3)) ? 1'b0 : 1'b1;
    assign DONE = (State == Ends) ? 1'b1 : 1'b0;

    // counter for number of iterations
    initial State = Inits;

    always @(posedge Clock) begin
       if (State == Inits)
          Count = 0;
       else if ((State == Shifts1) || (State == Shifts2)) begin
          if (Count == 7)
             Count = 0;
          else
             Count = Count + 1;
       end
    end

    // state transitions
    always @(posedge Clock) begin
       case (State)
       Ends: if (Start == 1'b1) State = Inits; else State = Ends;
       Inits: State = Subs1;
       Subs1: if (Rsign == 1'b1) State = Shifts1; else State = Shifts2;
       Subs: if (Count == 0) begin if (Rsign == 1'b0) State = Shifts4; else State = Restore;end
             else if (Rsign == 1'b1) State = Shifts1;
             else State = Shifts2;
       Adds: if (Count == 0) begin if (Rsign == 1'b0) State = Shifts4; else State = Restore; end
             else if (Rsign == 1'b1) State = Shifts1; else State = Shifts2;
       Shifts1: State = Adds;
       Shifts2: State = Subs;
       Shifts3: State = Ends;
       Shifts4: State = Ends;
       Restore: State = Shifts3;
       endcase
    end
endmodule
```

MUX Module

```
module MUX #(parameter N = 8)
(
    input [N-1:0] A, B,      //N-bit inputs
    output logic [N-1:0] Y, //N-bit output
    input S                  //select signal
);

    assign Y = (S == 0) ? A : B;

endmodule
```

Shift Register Module

```
module shiftreg #(parameter N = 8)
(
    input [N-1:0] D,          //parallel inputs
    output logic [N-1:0] Q, //register outputs
    input CLK,                //clock (active high)
    input LD,                 //synchronous load select
    input SH,                 //synchronous shift select
    input SerIn               //serial shift input
);

    always @(posedge CLK) begin
        if (LD == 1'b1)
            Q = D;                      //synchronous load
        else if (SH == 1'b1)
            Q = {Q[N-2:0], SerIn};   //synchronous lect shift
        else
            Q = Q;                      //default hold state
    end

endmodule
```

ALU Register

```
module alu #(parameter N = 8)
(
    input [N:0] A,              //N+1 bit operand
    input [N-1:0] B,            //N bit operand
    output logic [N:0] Result, //N+1 bit result (includes carry)
    input Sel                   //Sel = 0 for add, 1 for sub
);

    assign Result = (Sel == 0) ? (A + B) : (A - B);

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
```

MUX/Controller 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
```

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
```

Clock Ladder Module

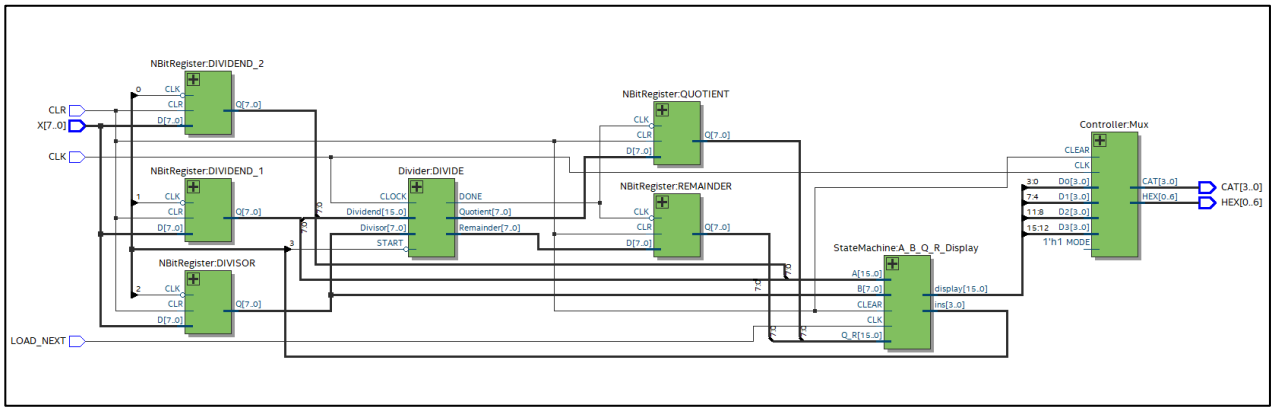
```
module clk_ladder #(parameter N = 32)
(
    input CLK,
    output logic clk190, clk1
);
    logic [N-1:0] ladder;

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

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

endmodule
```

Binary to Seven-Seg Display Decoder Module

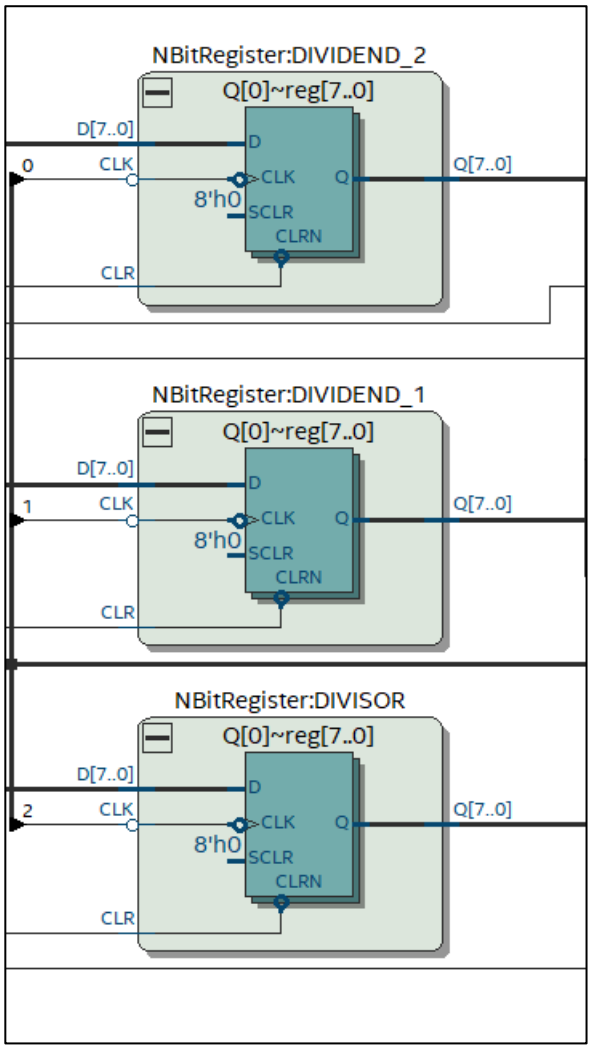
```
module binary2seven
(
   input [3:0] BIN, MODE,
   output logic [0:6] SEV
);

   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
```

RTL DIAGRAMS

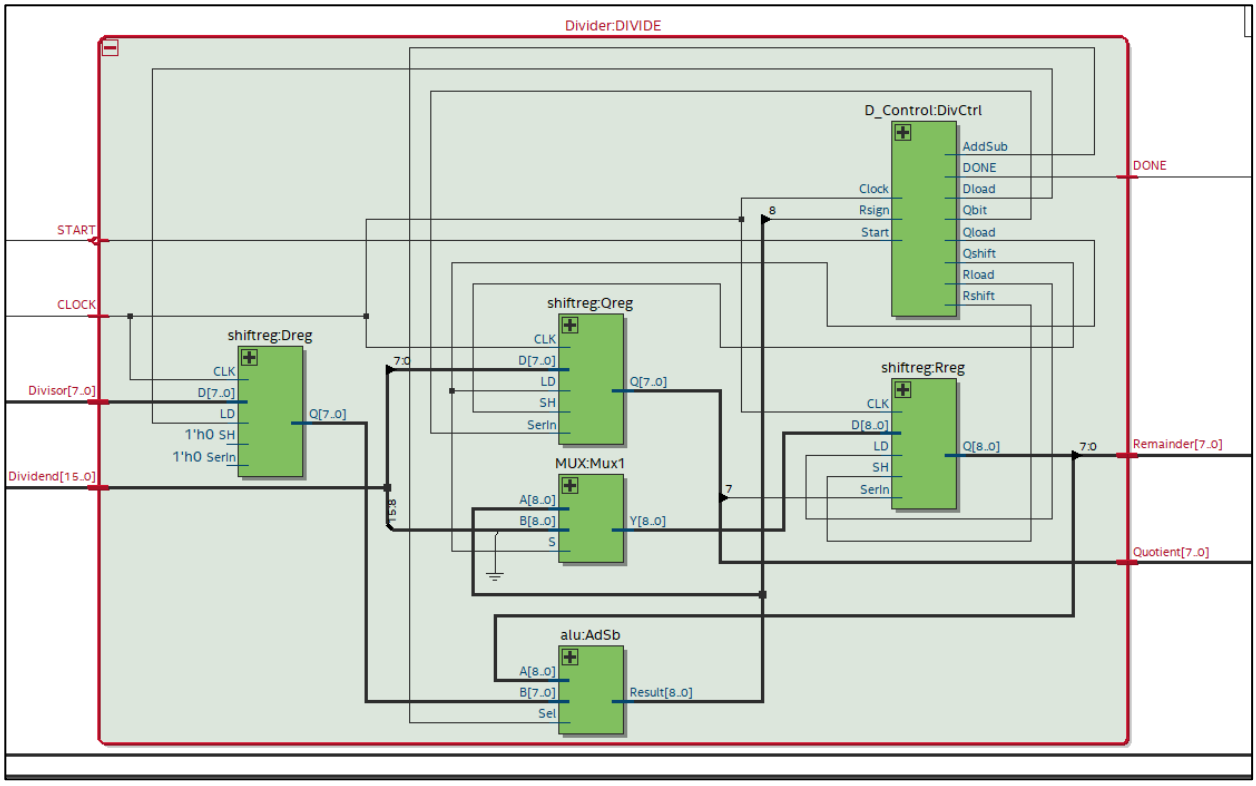
Top Module



Dividend & Divisor Registers

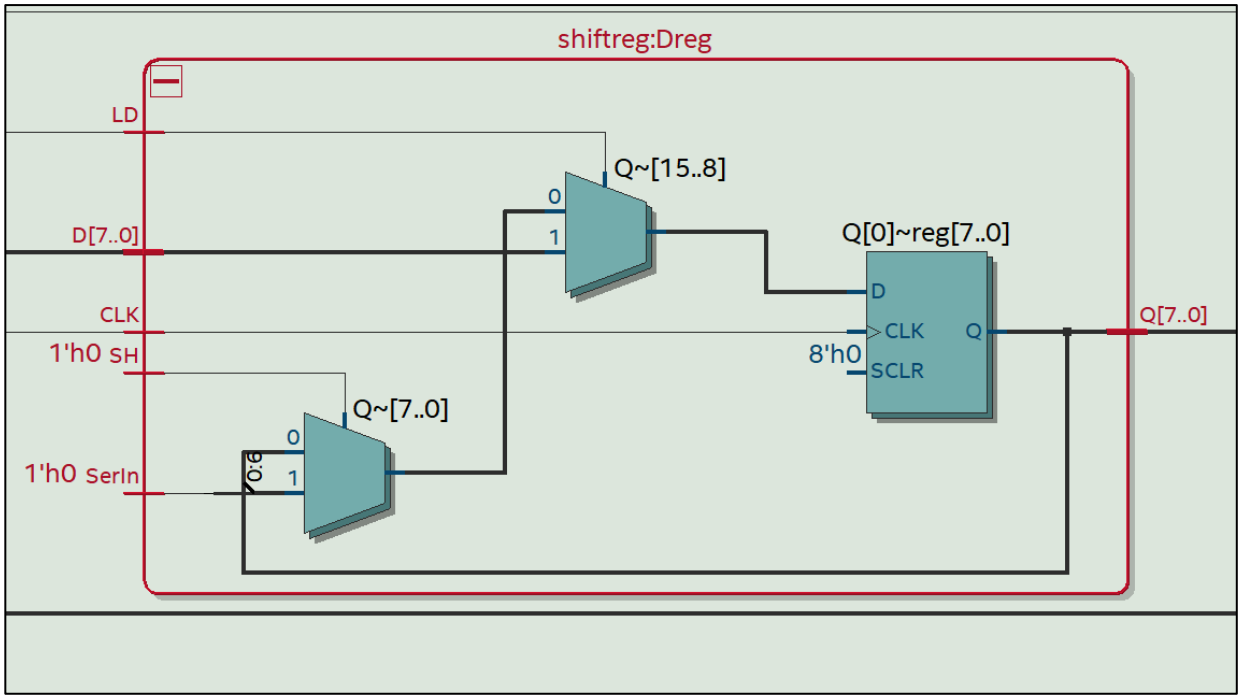


Divider

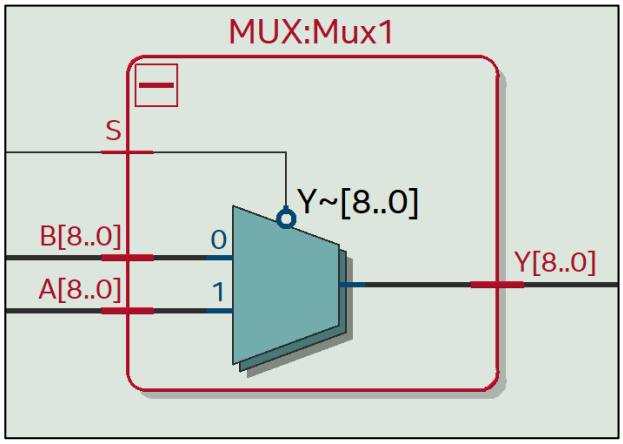


Shift Register

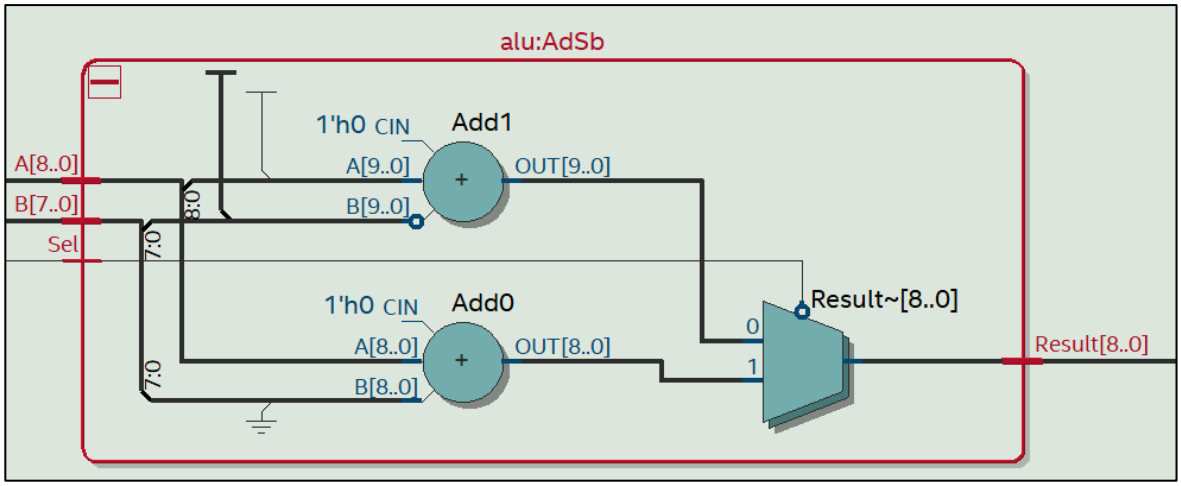
(Same RTL Diagram on Shifting Divisor, Quotient & Remainder)



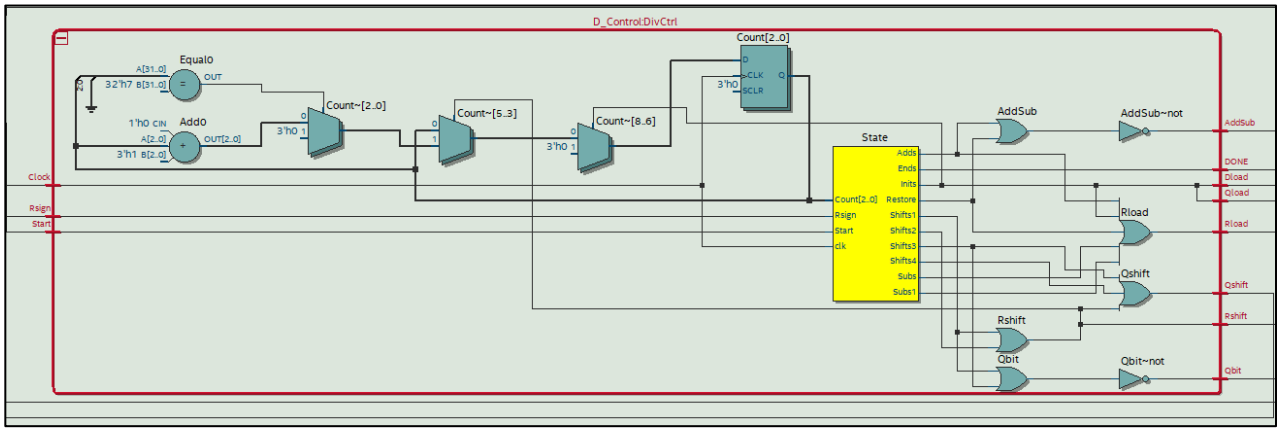
Mux



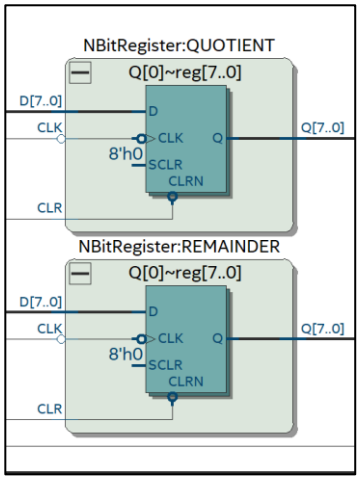
ALU



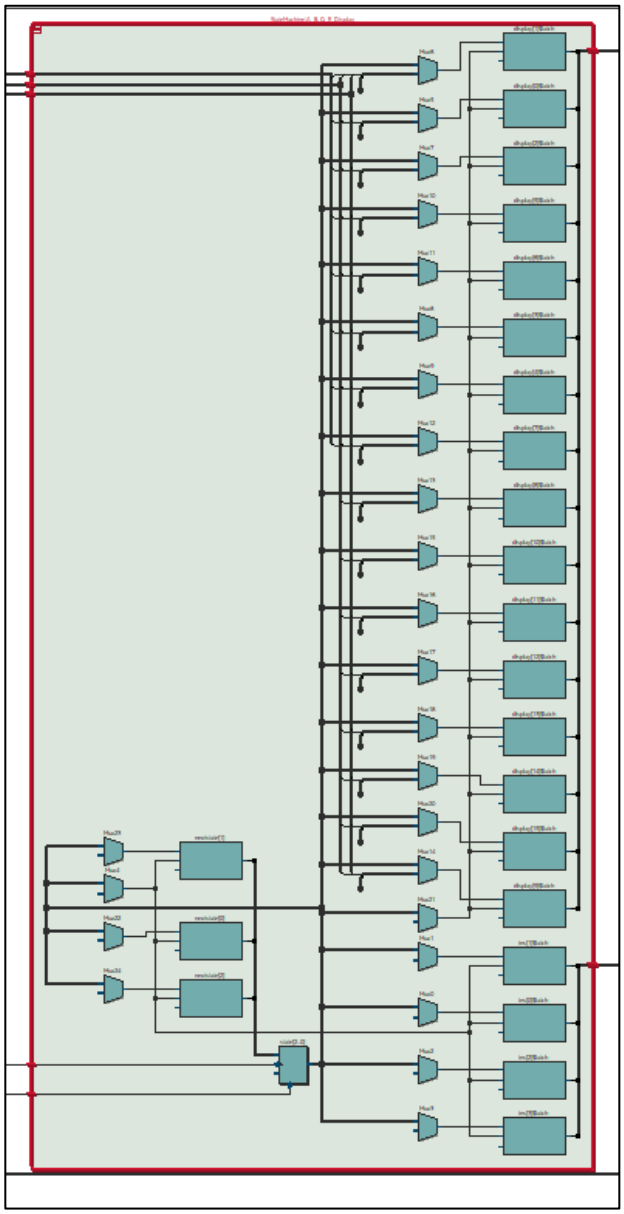
Divider Control Unit



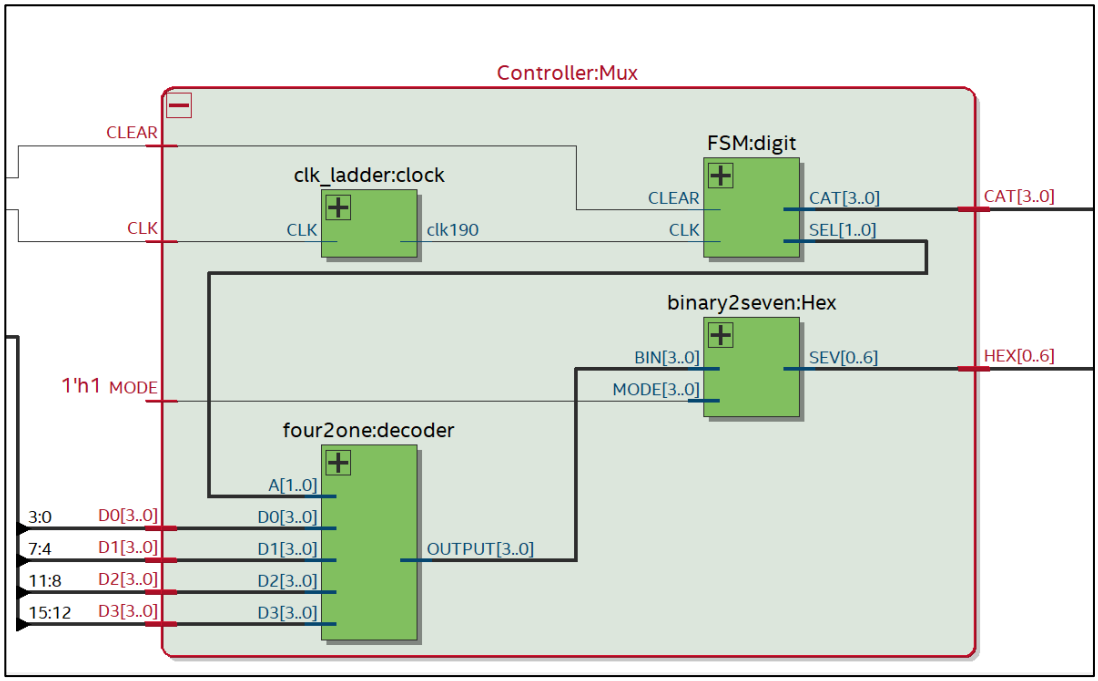
Remainder & Quotient Registers



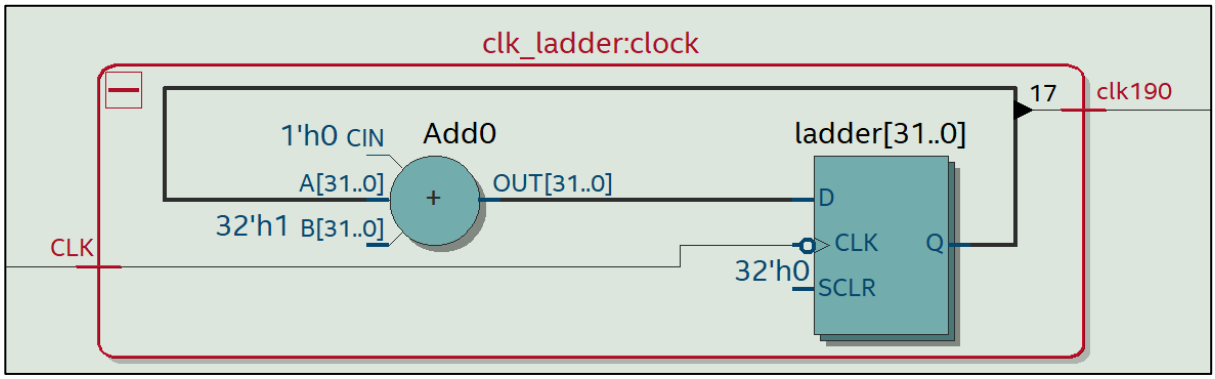
Display Different Registers State Machine



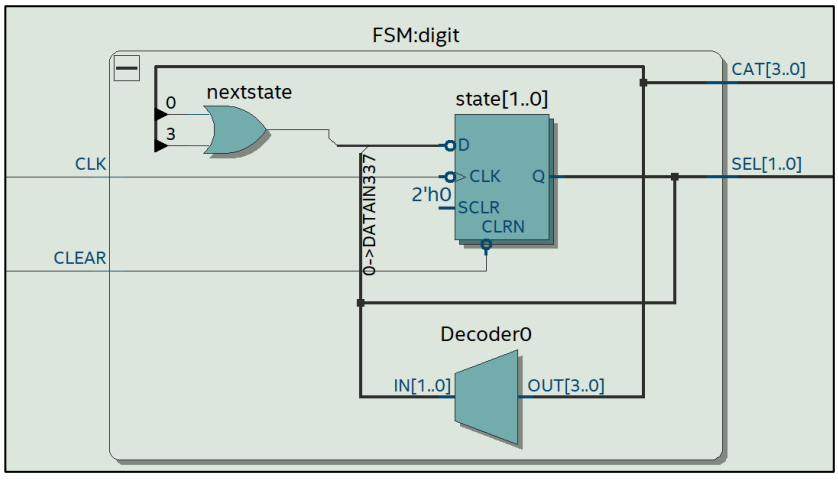
MUX/Controller



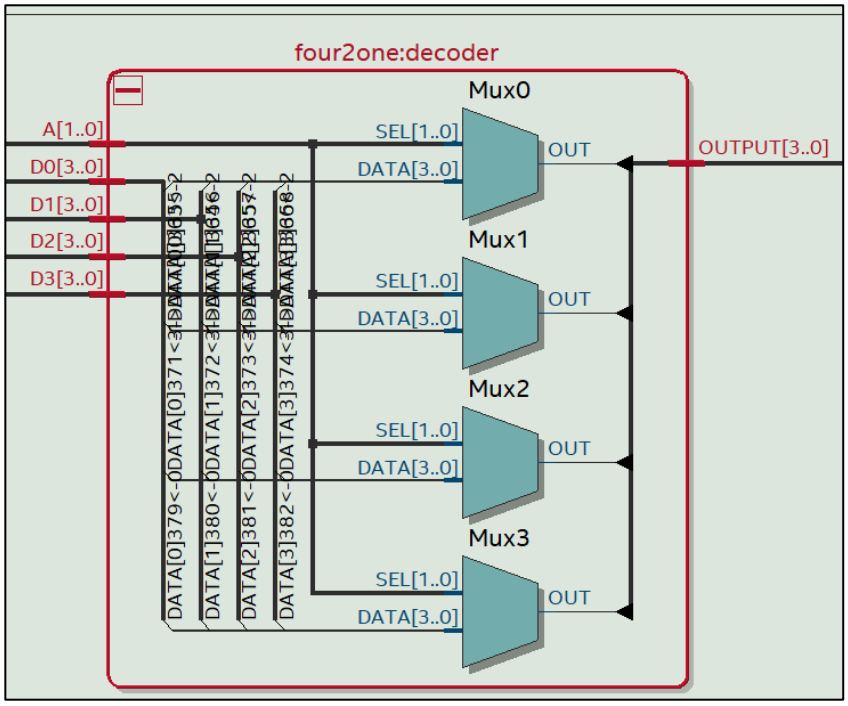
Clock Ladder



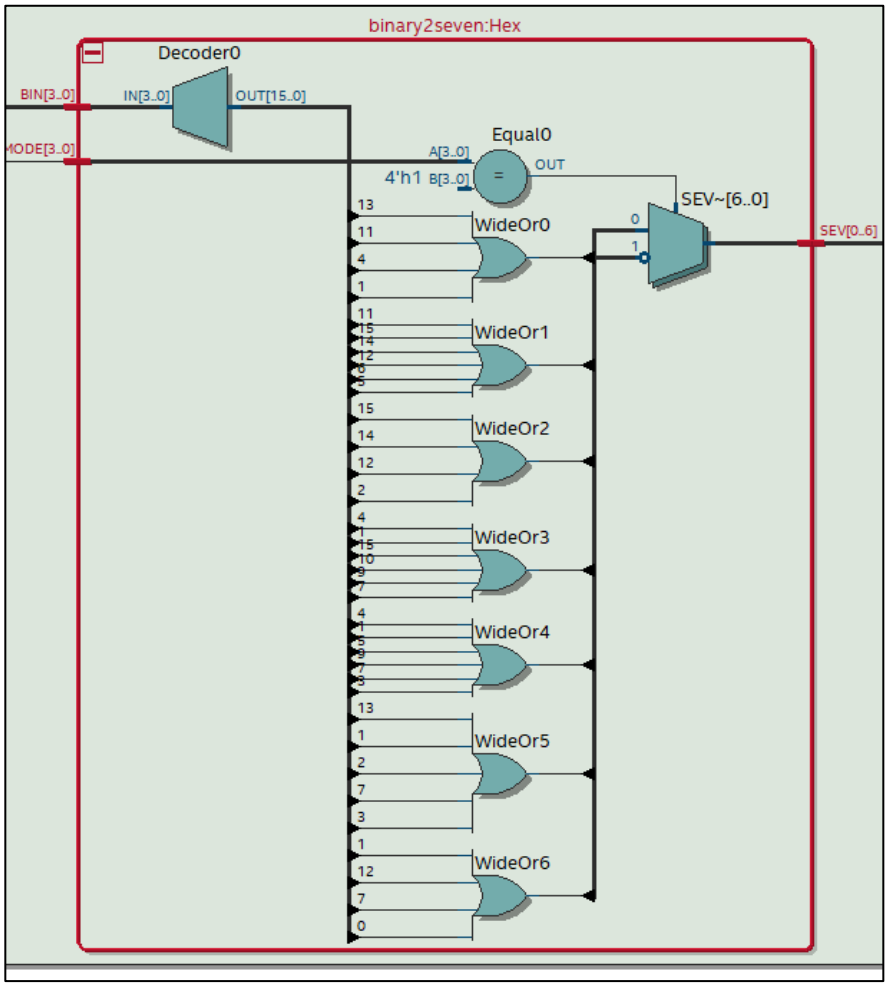
Finite State Machine



Four to One Decoder



Binary to Seven Segment Display Hex



Compilation Summary

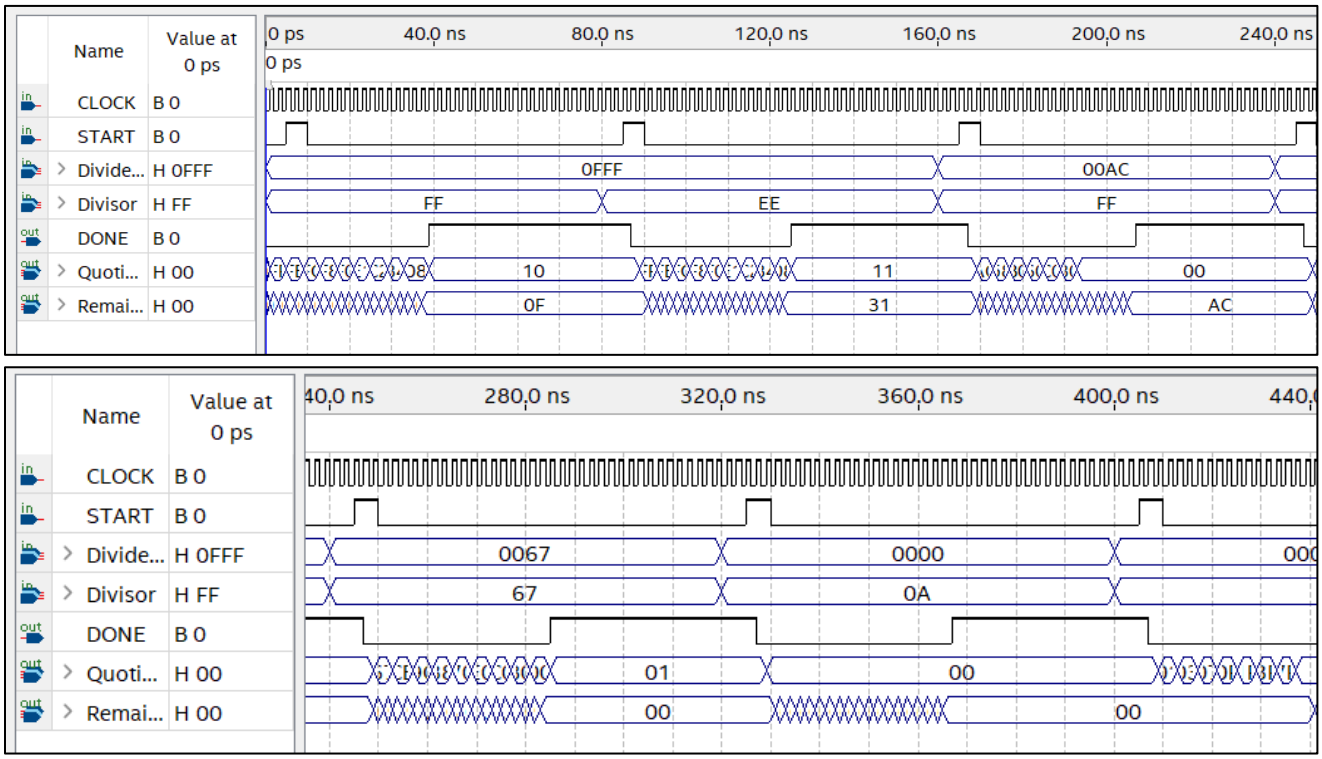
Flow Summary

<<Filter>>

Flow Status	Successful - Wed Apr 03 19:54:31 2024
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	Lab5
Top-level Entity Name	Lab5
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	173 / 49,760 (< 1 %)
Total registers	101
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 %)

# of ALMs = 173
# of REGISTERS = 101
# of PINS = 22

SIMULATION RESULTS WAVEFORM



Test	Quotient (Hex)	Remainder (Hex)	Clock Cycles
a)	10	0F	~ 34
b)	11	31	~ 40
c)	00	AC	~ 42
d)	01	00	~ 40
e)	00	00	~ 42

- What would be the corresponding divide times for your fastest clock?
  Fastest divide time was ~ 34 clock cycles.

$$T = \frac{1}{f} = \frac{1}{50MHz} = 20\ ns$$

$$T_{34\ Cycles} = 20\ ns\ x\ 34 = 860\ ns$$

Fastest divide time was 680 ns

PIN ASSIGMENTS

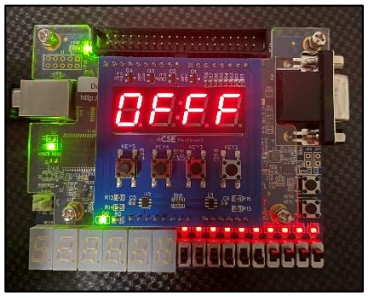
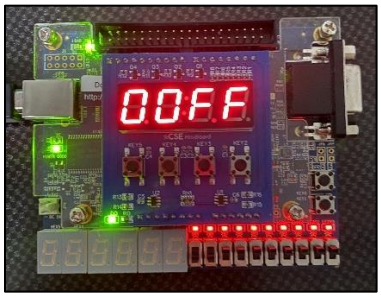
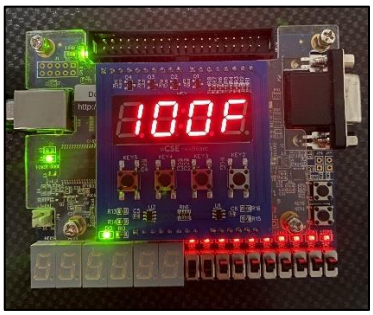
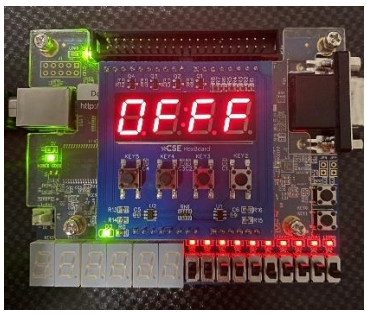
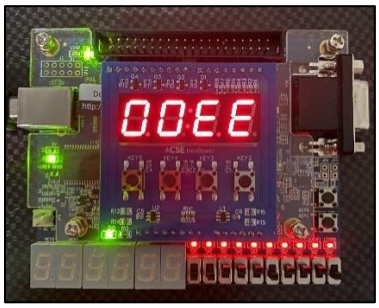
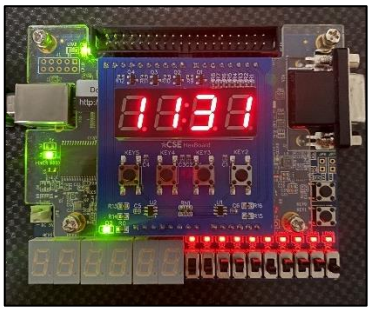
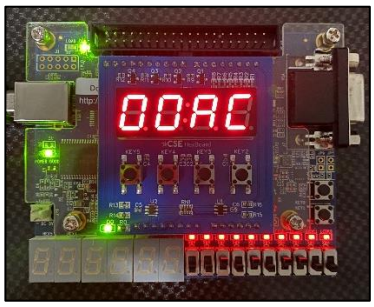
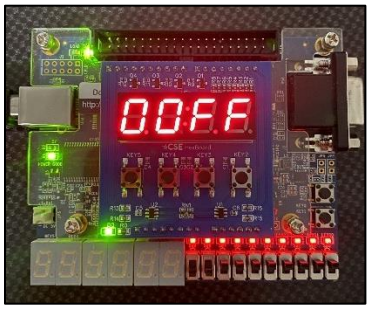
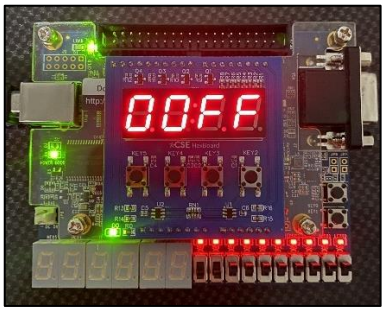
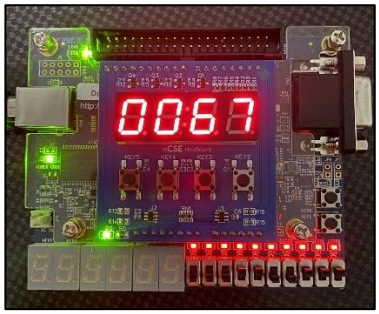
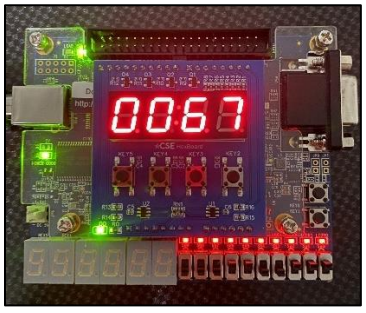
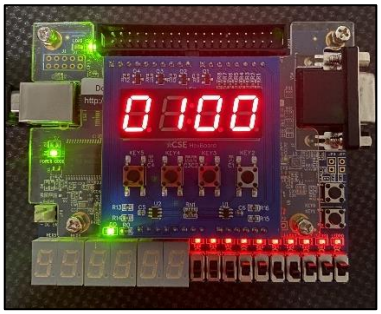
	tatu	From	To	Assignment Name	Value	Enabled
1	✔		in CLR	Location	PIN_B8	Yes
2	✔		out CAT[0]	Location	PIN_AB19	Yes
3	✔		out CAT[1]	Location	PIN_AA19	Yes
4	✔		out CAT[2]	Location	PIN_Y19	Yes
5	✔		out CAT[3]	Location	PIN_AB20	Yes
6	✔		out HEX[0]	Location	PIN_AA12	Yes
7	✔		out HEX[1]	Location	PIN_AA11	Yes
8	✔		out HEX[2]	Location	PIN_Y10	Yes
9	✔		out HEX[3]	Location	PIN_AB9	Yes
10	✔		out HEX[4]	Location	PIN_AB8	Yes
11	✔		out HEX[5]	Location	PIN_AB7	Yes
12	✔		out HEX[6]	Location	PIN_AB17	Yes
13	✔		in X[0]	Location	PIN_C10	Yes
14	✔		in X[1]	Location	PIN_C11	Yes
15	✔		in X[2]	Location	PIN_D12	Yes
16	✔		in X[3]	Location	PIN_C12	Yes
17	✔		in X[4]	Location	PIN_A12	Yes
18	✔		in X[5]	Location	PIN_B12	Yes
19	✔		in X[6]	Location	PIN_A13	Yes
20	✔		in X[7]	Location	PIN_A14	Yes
21	✔		in CLK	Location	PIN_P11	Yes
22	✔		in LOA...EXT	Location	PIN_A7	Yes
23		<<new>>	<<new>>	<<new>>		

DE-10 LITE TEST RESULTS

Unsigned Test Results

Test	Dividend ÷ Divisor (Hex)	Quotient (Hex)	Remainder (Hex)
a)	0FFF ÷ FF	10	0F
b)	0FFF ÷ EE	11	31
c)	00AC ÷ FF	00	AC
d)	0067 ÷ 67	01	00
e)	0000 ÷ 0A	00	00

PHOTOS OF TEST RESULTS

	Dividend	Divisor	{Quotient, Remainder}
a)			
b)			
c)			
d)			
e)