Name: _____Servando_Olvera_____ ID# _____1001909287_____

Date Submitted: _____02-20-2024____ Time Submitted _____9:00_pm_____

CSE 3341 Digital Logic Design II

CSE 5357 Advanced Digital Logic Design

Spring Semester 2024

Lab 2 – Registered High-Speed Adder Subtractor

200 points

Due Date – February 20, 2024, 11:59 PM

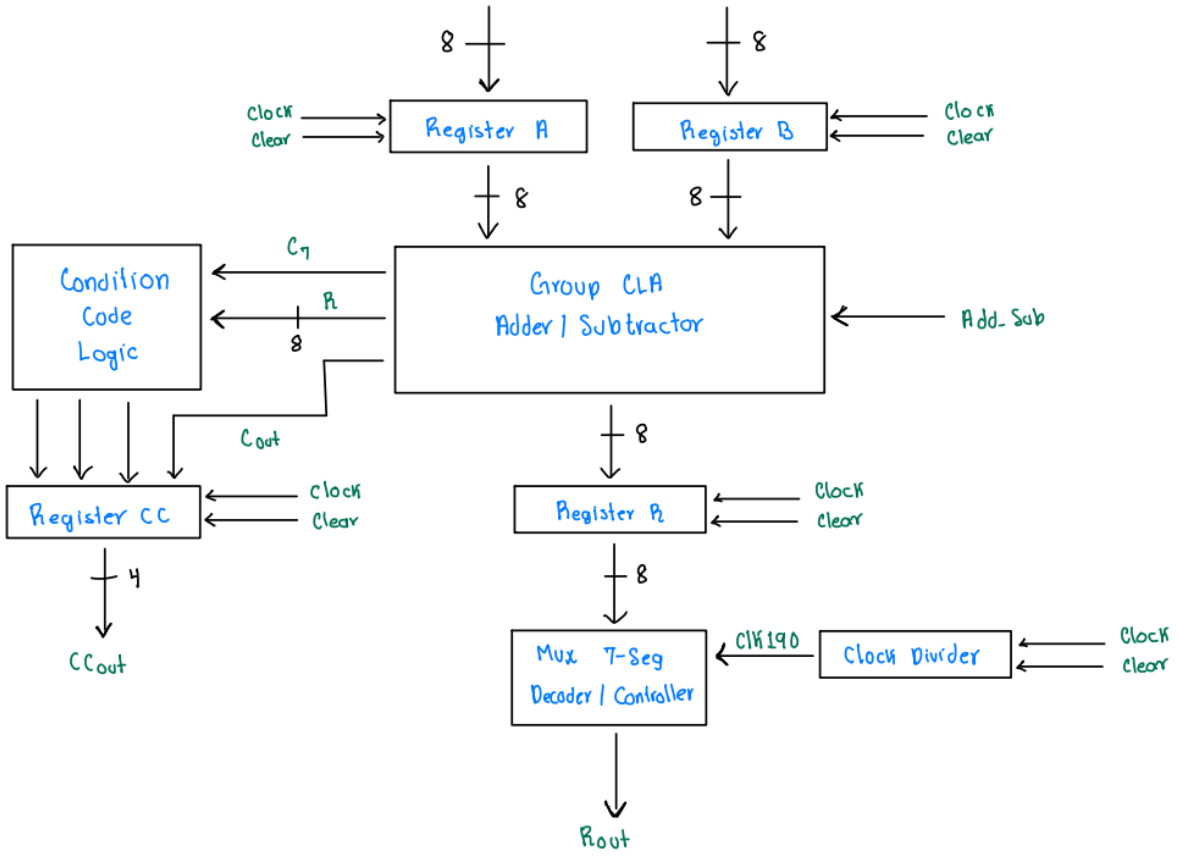
Submit on Canvas Assignments

DESIGN REQUIREMENTS

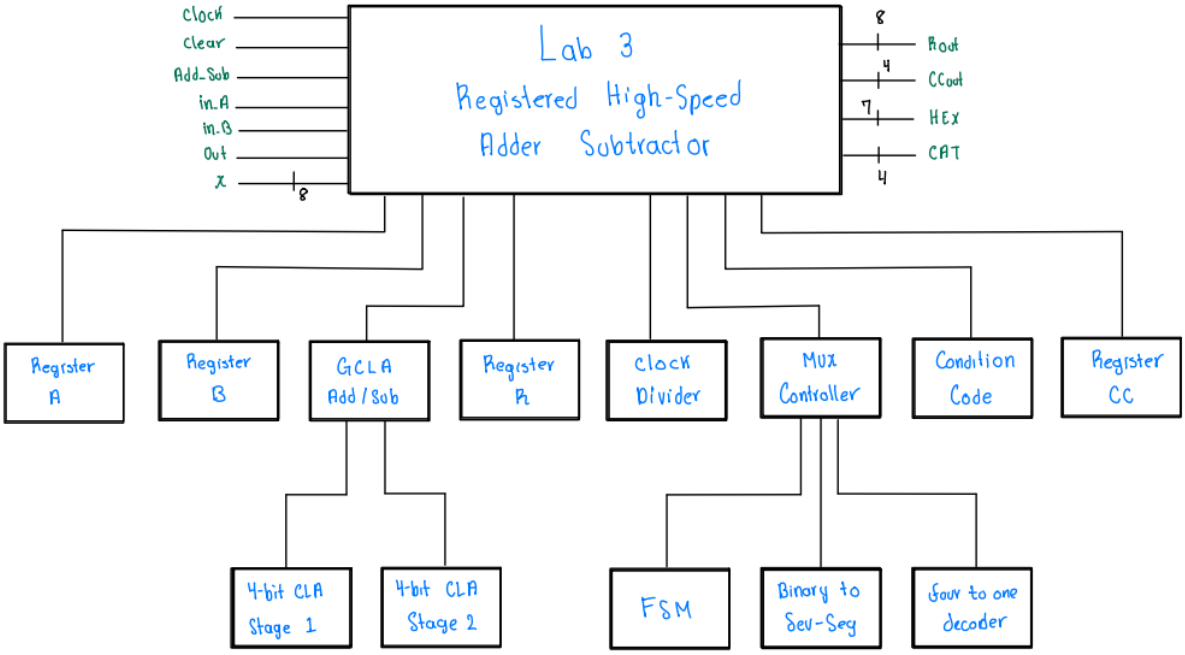
Your assignment is to design an eight-bit registered high-speed adder subtractor that also produces carry-out, overflow, zero, and negative condition code outputs. The adder/subtractor component must use group carry lookahead architecture. You will code your design in SystemVerilog, simulate to verify its correctness, and test its functionality on a DE10-Lite development board. You will also perform a timing analysis on Quartus and compute the add/subtract time of your device.

Assume A, B, and R are eight-bit signed binary numbers using a two's complement number system. CarryOut, OVR, ZERO, and NEG are condition codes determined by the result of the last operation performed. InA and InB load registers A and B, respectively. Out loads register R with the result of the last operation and the register CC with the last condition codes. Clear loads all zeros in all registers.

ORGANIZATION DIAGRAM



HIRERARCHY DIAGRAM



SYSTEM-VERILOG CODE

Part Top Module

```
module Lab3
(
    input CLK, CLEAR, Add_Sub, inA, inB, Out,
    input [7:0] X,
    output logic [7:0] Rout,
    output logic [3:0] CCout,
    output logic [0:6] HEX,
    output logic [3:0] CAT
);

    logic [7:0] Aout, Bout, R;
    logic Cout, C7, OVR, Neg, Zero, clk190;

    NBitRegister regA
    (
        .D(X) ,
        .CLK(inA) ,
        .CLR(CLEAR),
        .Q(Aout)
    );

    NBitRegister regB
    (
        .D(X) ,
        .CLK(inB),
        .CLR(CLEAR),
        .Q(Bout)
    );

    GCLA_AddSub attempt // Group CLA Adder/Subtractor
    (
        .A(Aout),
        .B(Bout),
        .Add_Sub(Add_Sub),
        .R(R),
        .Cout(Cout),
        .C7(C7)
    );

    NBitRegister regR // Register R
    (
        .D(R),
        .CLK(Out),
        .CLR(CLEAR),
        .Q(Rout)
    );


    divideXN #(263158,32) CLK190        // 190 Hz clock
    (                                   // 50e6 / 190 = 263158
        .CLK(CLK),
        .CLEAR(CLEAR),
        .OUT(clk190)
    );

    Controller Mux
    (
        .clk190(clk190),
        .CLEAR(CLEAR),
        .MODE(1'b1),
        .D0(Rout[3:0]),
        .D1(Rout[7:4]),
        .CAT(CAT),
        .HEX(HEX)
    );

    ConditionCode CCLogic
    (
        .R(R),
        .Cout(Cout),
        .C7(C7),
        .OVR(OVR),
        .Neg(Neg),
        .Zero(Zero)
    );

    NBitRegister #(3'd4) regCC  // Register CC
    (
        .D({Cout, OVR, Neg, Zero}) ,
        .CLK(Out),
        .CLR(CLEAR),
        .Q(CCout)
    );

endmodule
```

Register Module

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

    always @ (posedge CLK, negedge CLR) begin
        if (CLR == 1'b0)
            Q <= 0;                                  //zero out register
        else if (CLK == 1'b1)
            Q <= D;                                  //data input values loaded in
    end
endmodule
```

Four Bit Carry Look-Ahead Adder Module

```
module FourBit_CLA
(
    input [3:0] A, B,
    input Cin,
    output logic [3:0] G, P, Sum,
    output logic Cout,
    output logic C7
);

    logic [4:0] C;

    assign G = A & B;              // Generate

    assign P = A ^ B;              // Propagate

    always_comb begin
        C[0] = Cin;
        C[1] = G[0] | P[0] & C[0];
        C[2] = G[1] | P[1] & C[1];
        C[3] = G[2] | P[2] & C[2];
        C[4] = G[3] | P[3] & C[3];
    end

    assign Cout = C[4];                // Output Carry


    assign Sum = A ^ B ^ C;            // Compute Sum
                                       // C gets truncated!!! Might be an issue
    assign C7 = C[3];

endmodule
```

Group Carry Look-Ahead Adder Module

```
module GCLA_AddSub
(
    input [7:0] A, B,
    input Add_Sub,        // Eventually... Feature in the works
    output [7:0] R,
    output Cout,
    output C7
);

    //logic C4;
    logic [7:0] g, p;
    logic P3_0, G3_0, C4, P7_4, G7_4;

    FourBit_CLA GCLA4_1stStage    // Fisrt Stage/ fisrt 4 bits computed
    (
        .A(A[3:0]),
        .B({B[3] ^ Add_Sub, B[2] ^ Add_Sub, B[1] ^ Add_Sub, B[0] ^ Add_Sub}),
        .Cin(Add_Sub),
        .G(g[3:0]),
        .P(p[3:0]),
        .Sum(R[3:0]),
        //.Cout(C4)
    );

    assign P3_0 = p[3] & p[2] & p[1] & p[0];
    assign G3_0 = g[3] | g[2] & p[3] | g[1] & p[3] & p[2] | g[0] & p[3] & p[2] & p[1];
    assign C4   = G3_0 | P3_0 & Add_Sub;

    FourBit_CLA GCLA4_2ndStage    // Second Stage/ second 4 bits computed
    (
        .A(A[7:4]),
        .B({B[7] ^ Add_Sub, B[6] ^ Add_Sub, B[5] ^ Add_Sub, B[4] ^ Add_Sub}),
        .Cin(C4),
        .G(g[7:4]),
        .P(p[7:4]),
        .Sum(R[7:4]),
        //.Cout(Cout),
        .C7(C7)
    );

    assign P7_4 = p[7] & p[6] & p[5] & p[4];
    assign G7_4 = g[7] | g[6] & p[7] | g[5] & p[7] & p[6] | g[4] & p[7] & p[6] & p[5];
    assign Cout = G7_4 | P7_4 & C4;     // C8

endmodule
```

Condition Code Module

```
// Too simple for its own module
module ConditionCode
(
    input [7:0] R,
    input Cout, C7,
    output OVR, Neg, Zero
);

    assign OVR = C7 ^ Cout;     // XOR C[8] with C[7]
    assign Neg = R[7];          // Signed number, so MSB determines if neg

    assign Zero = ~(R[7] | R[6] | R[5] | R[4] | R[3] | R[2] | R[1] | R[0]);

endmodule
```

Binary to Seven Segment Display 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
```

MUX/Controller Module

```
module Controller
(
    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
    (
        .BIN(out),
        .MODE(MODE),
        .SEV(HEX)
    );


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
                default: begin nextstate = 2'b00; SEL = 2'b00; CAT = 4'b000; end  // Don't touch the other digits
            endcase                                                               // Not needed here
endmodule
```

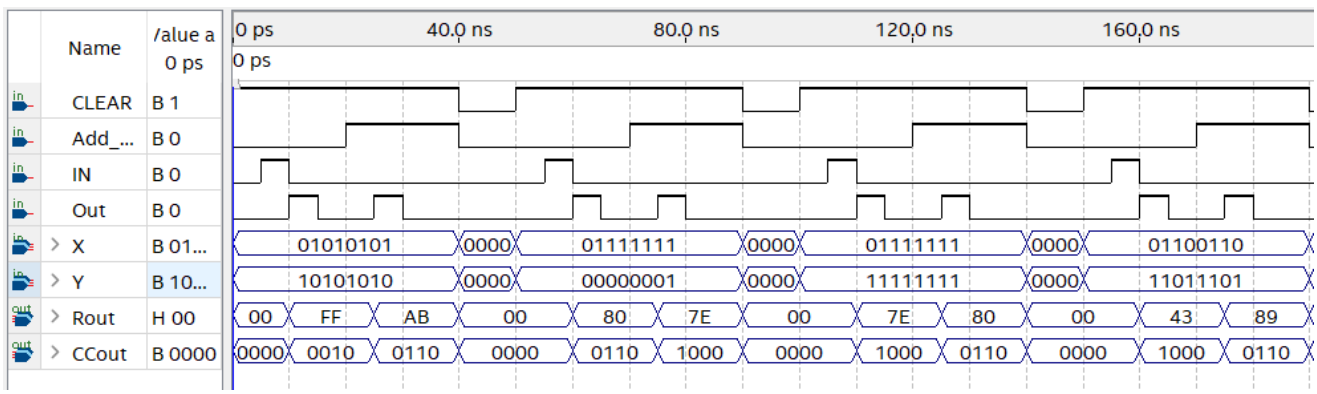
Four to One Decoder Module

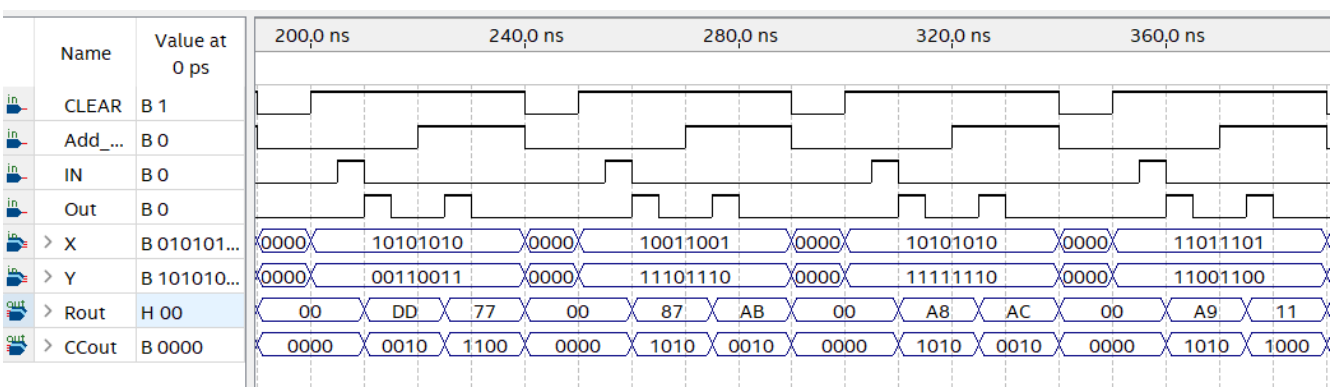
```
module four2one
(
    input [1:0] A,
    input [3:0] D0, D1, D2, D3,
    output logic [3:0] OUTPUT
);

    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
```

Clock Divider Module

```
module divideXN #(parameter N = 10, parameter M = 4)
(
    input CLK, CLEAR,
    output logic [M-1:0] COUNT,                    // COUNT is defined as a M-bit register
    output logic OUT
);
    always_ff @ (negedge CLK, negedge CLEAR)
        if (CLEAR == 1'b0)
            COUNT <= 0;                            // COUNT is loaded with all 0's
        else begin
            if (COUNT == N-2'd2) begin
                OUT <= 1'b1;
                COUNT <= N-1'd1;                   // Once COUNT = N-2 OUT = 1
            end else
                if (COUNT == N-1'd1) begin
                    OUT <= 1'b0;
                    COUNT <= 0;                    // Once COUNT = N-1 OUT=0
                end else begin
                    OUT <= 1'b0;
                    COUNT <= COUNT + 1'b1;         // COUNT is incremented
                end
        end
endmodule
```

SIMULATION RESULTS WAVEFORM



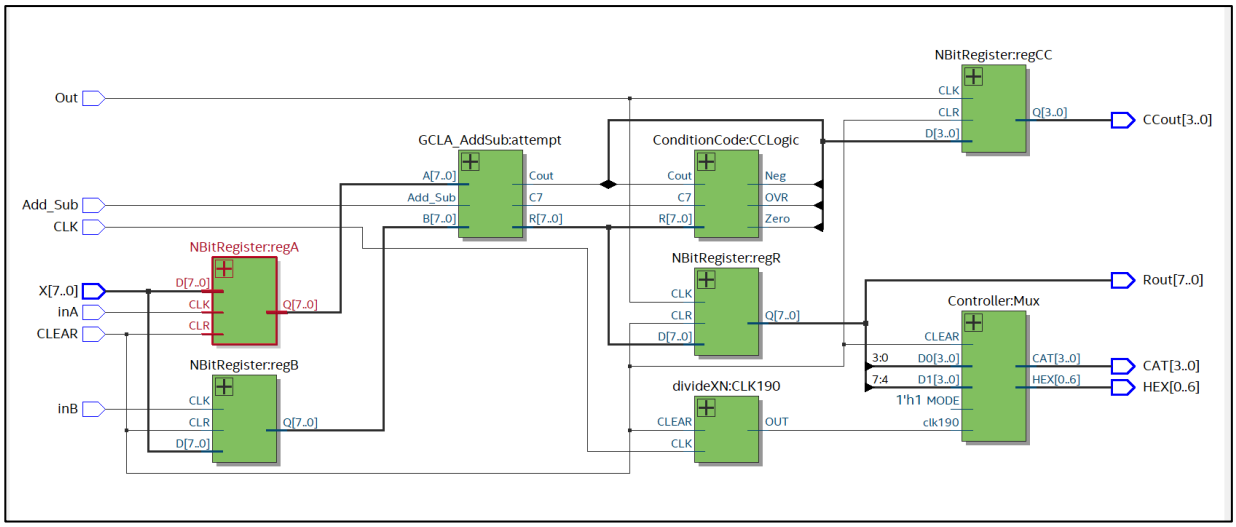


PIN ASSIGMENTS

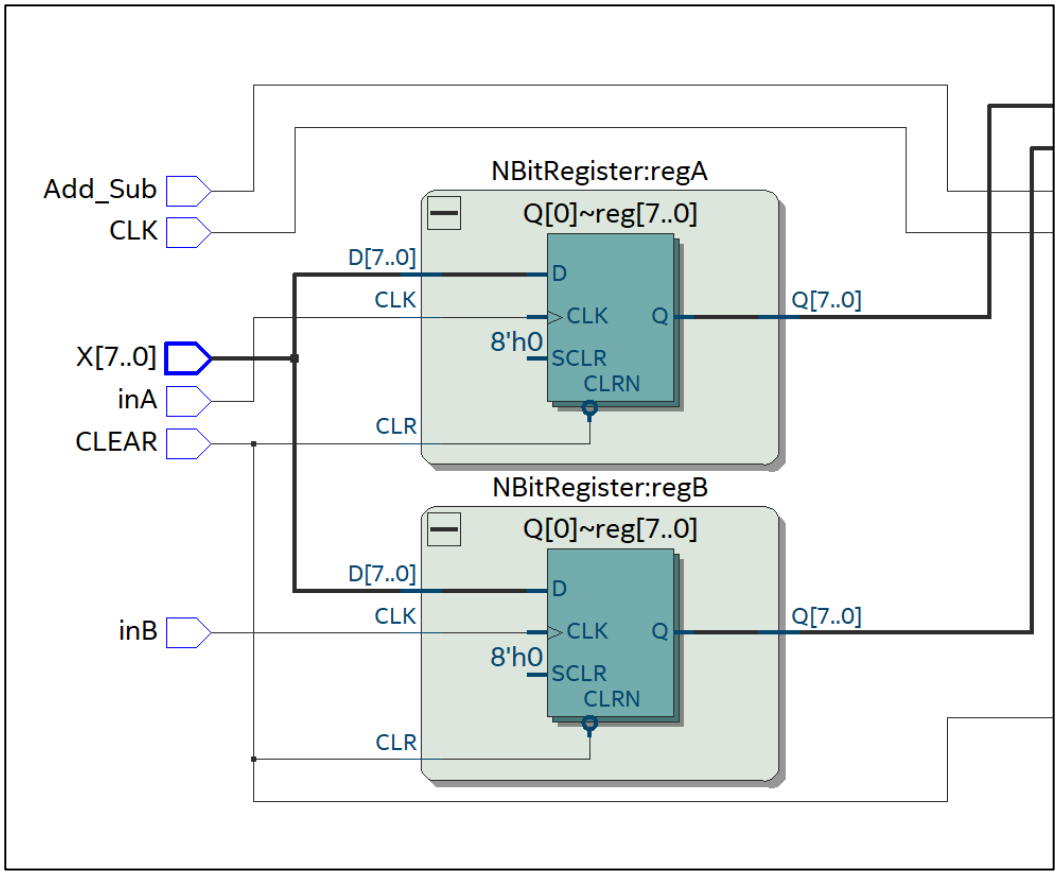
	tatu	From	To	Assignment Name	Value	Enabled	Entity
1	✔		CLEAR	Location	PIN_AB6	Yes	
2	✔		inA	Location	PIN_B8	Yes	
3	✔		inB	Location	PIN_A7	Yes	
4	✔		Out	Location	PIN_AB5	Yes	
5	✔		Add_Sub	Location	PIN_F15	Yes	
6	✔		X[1]	Location	PIN_C11	Yes	
7	✔		X[2]	Location	PIN_D12	Yes	
8	✔		X[3]	Location	PIN_C12	Yes	
9	✔		X[4]	Location	PIN_A12	Yes	
10	✔		X[5]	Location	PIN_B12	Yes	
11	✔		X[6]	Location	PIN_A13	Yes	
12	✔		X[7]	Location	PIN_A14	Yes	
13	✔		CCout[0]	Location	PIN_A8	Yes	
14	✔		CCout[1]	Location	PIN_A9	Yes	
15	✔		CCout[2]	Location	PIN_A10	Yes	
16	✔		CCout[3]	Location	PIN_B10	Yes	
17	✔		X[0]	Location	PIN_C10	Yes	
18	✔		HEX[1]	Location	PIN_AA11	Yes	
19	✔		HEX[2]	Location	PIN_Y10	Yes	
20	✔		HEX[3]	Location	PIN_AB9	Yes	
21	✔		HEX[4]	Location	PIN_AB8	Yes	
22	✔		HEX[5]	Location	PIN_AB7	Yes	
23	✔		HEX[6]	Location	PIN_AB17	Yes	
24	✔		CAT[0]	Location	PIN_AB19	Yes	
25	✔		CAT[1]	Location	PIN_AA19	Yes	
26	✔		CAT[2]	Location	PIN_Y19	Yes	
27	✔		CAT[3]	Location	PIN_AB20	Yes	
28	✔		HEX[0]	Location	PIN_AA12	Yes	
29	✔		CLK	Location	PIN_P11	Yes	
30		<<new>>	<<new>>	<<new>>			

RTL DIAGRAMS

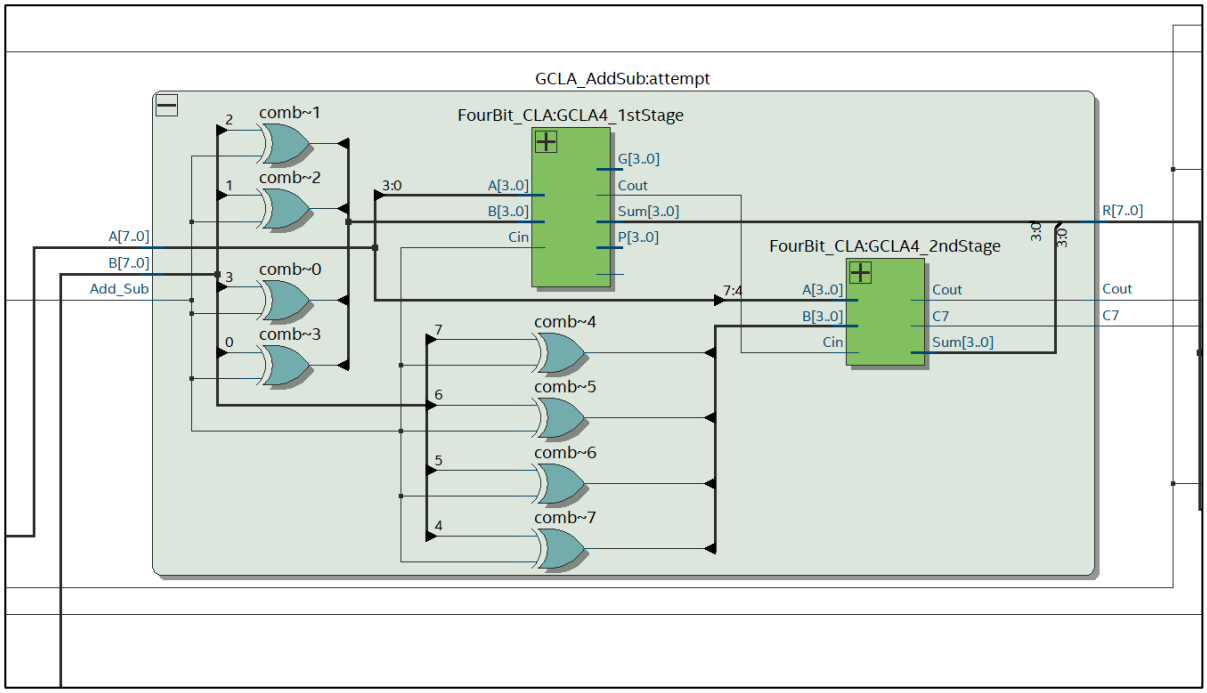
Top Module



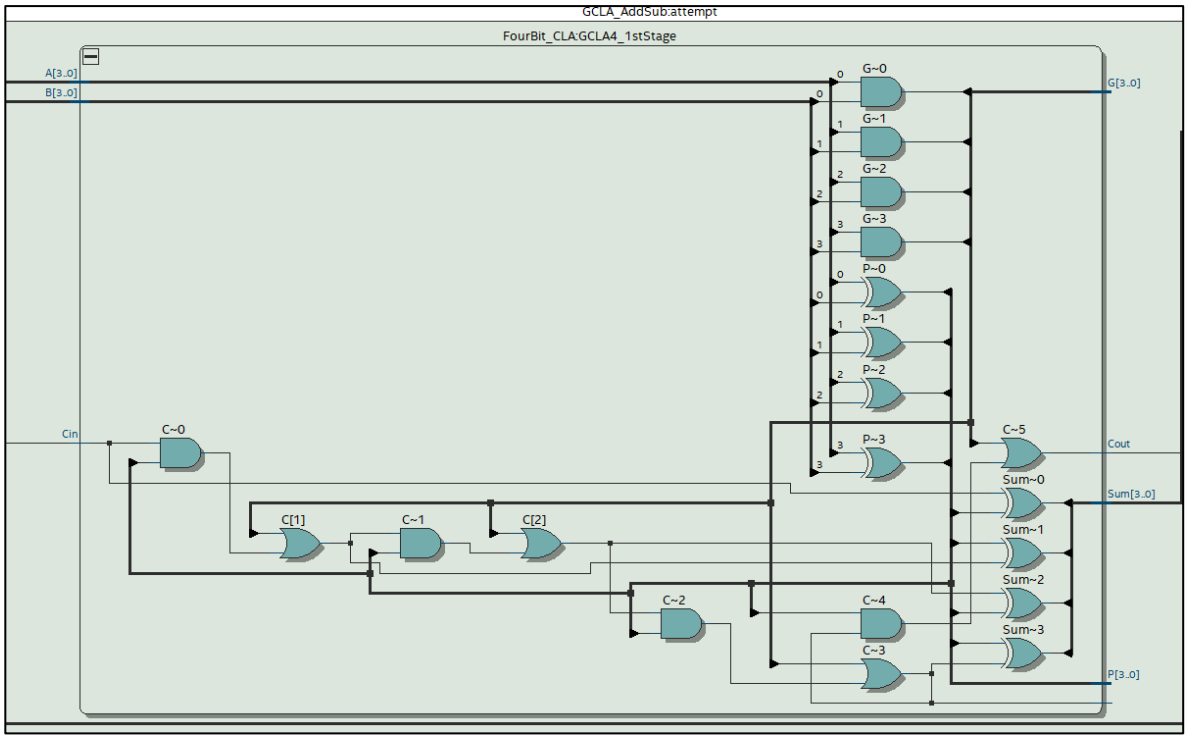
Registers A & B



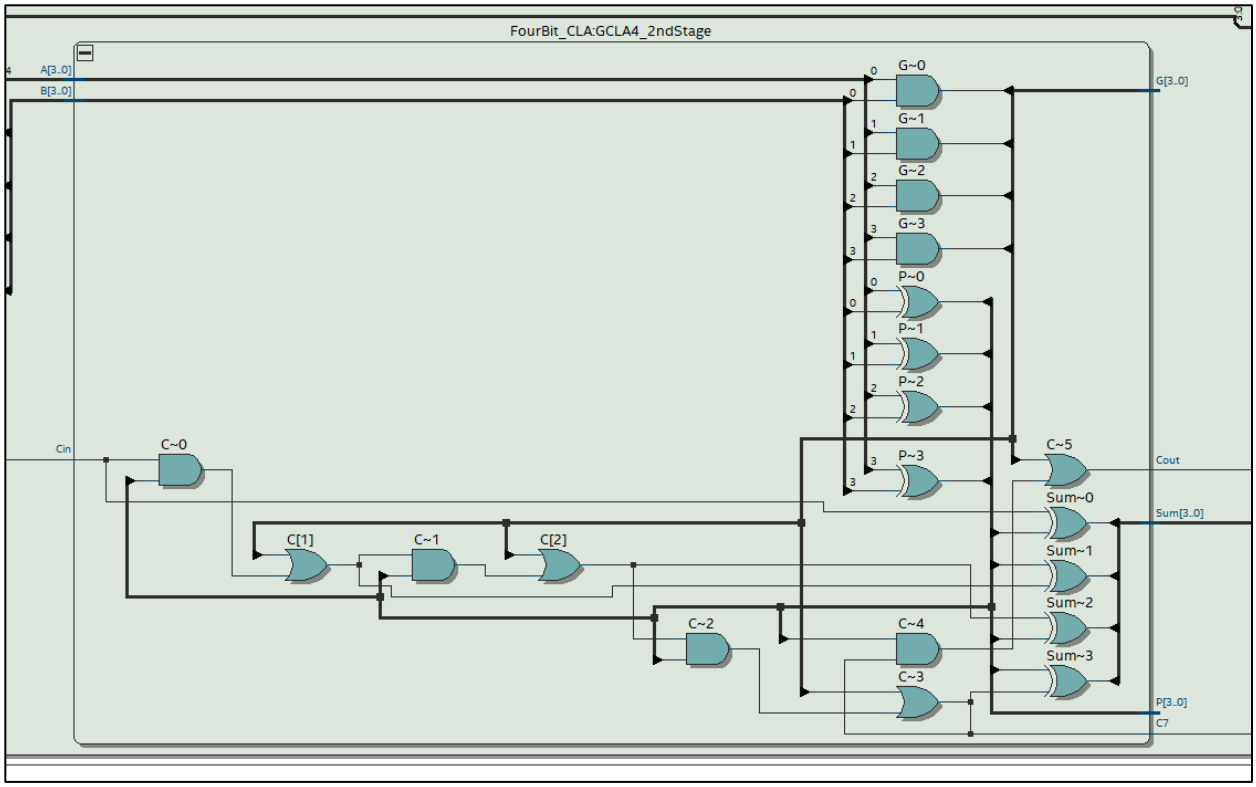
Group Carry Look-Ahead Adder



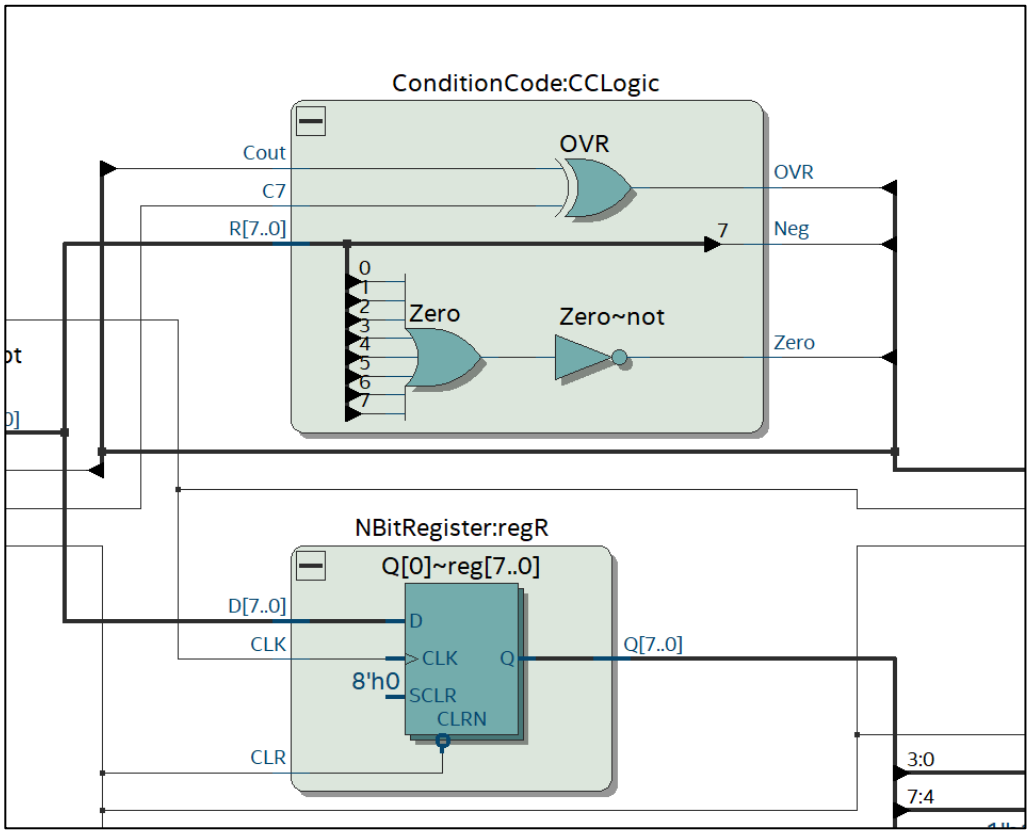
4-Bit Carry Look-Ahead Adder Stage 1



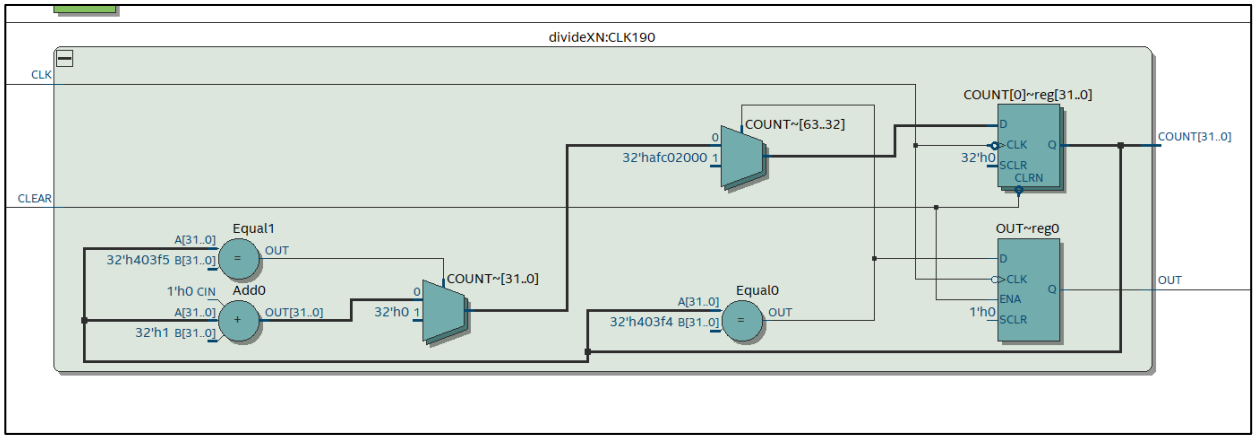
4-Bit Carry Look-Ahead Adder Stage 2



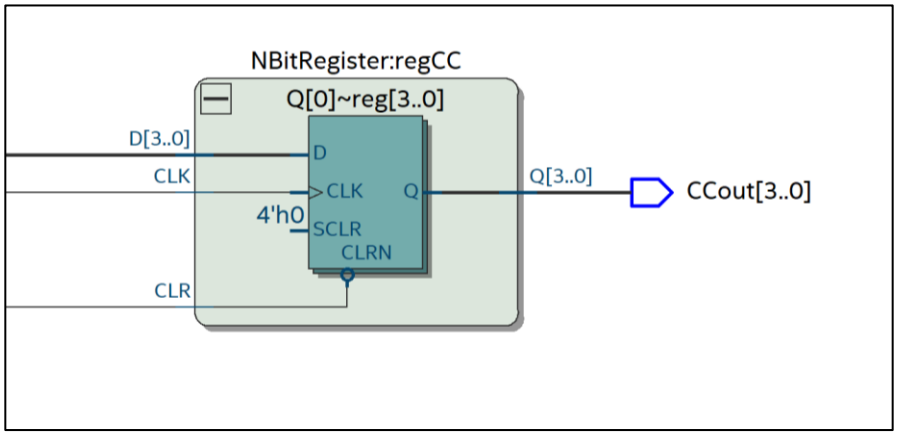
Condition Code & Register R



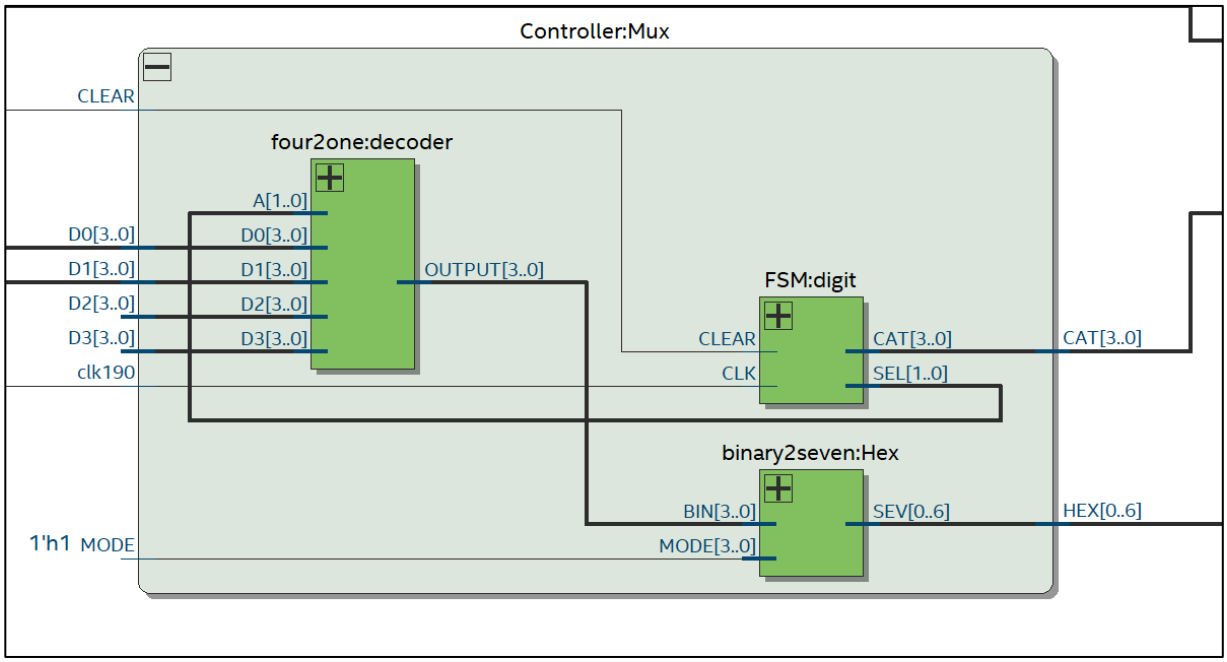
Clock Divider



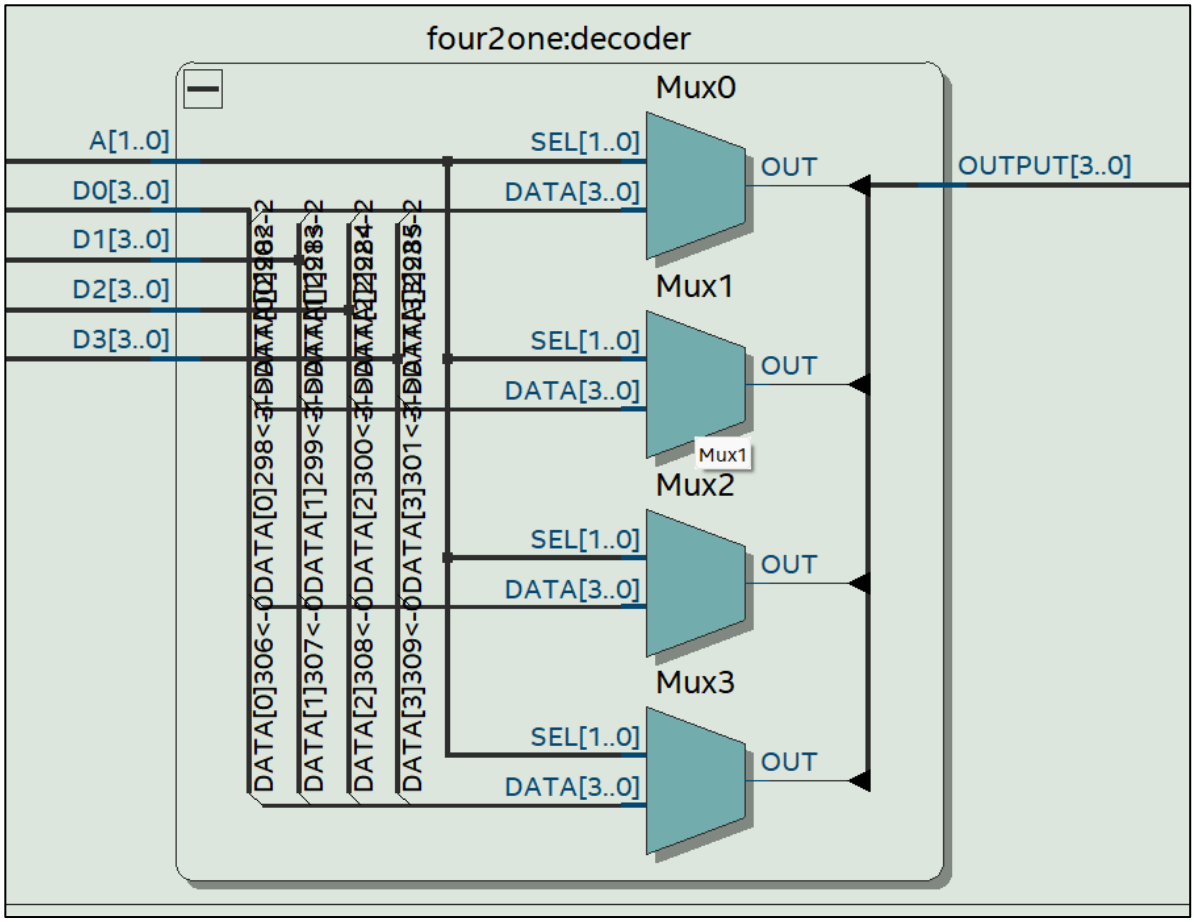
Register CC



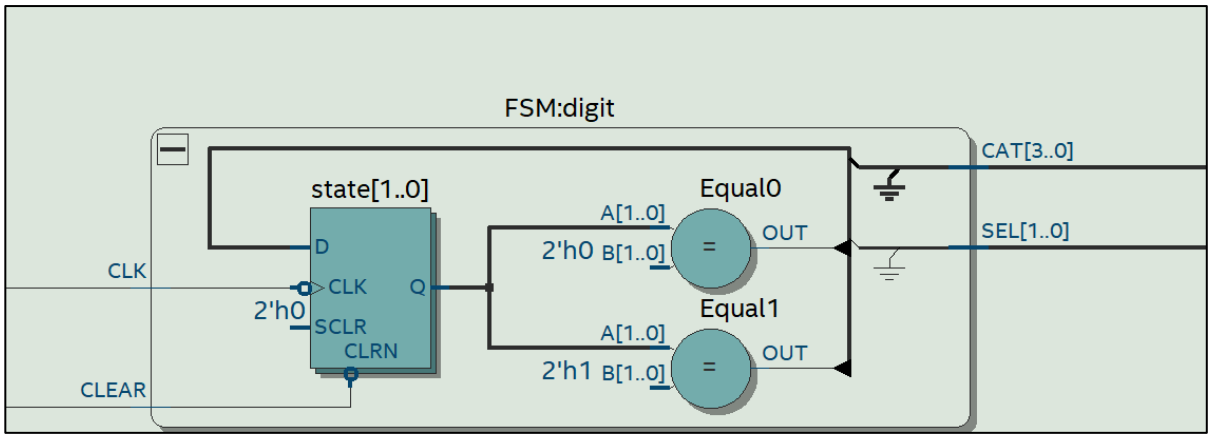
Mux/Controller



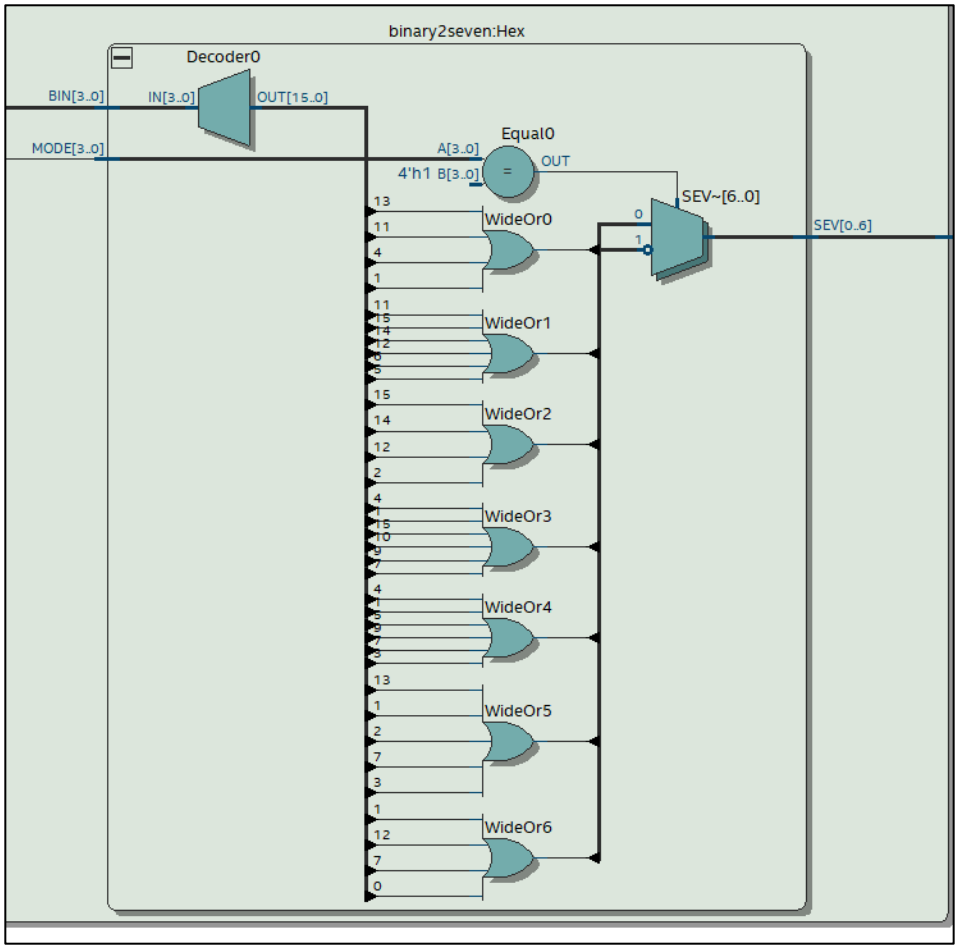
Four to One Decoder



Finite State Machine



Binary To Seven Hex Decoder



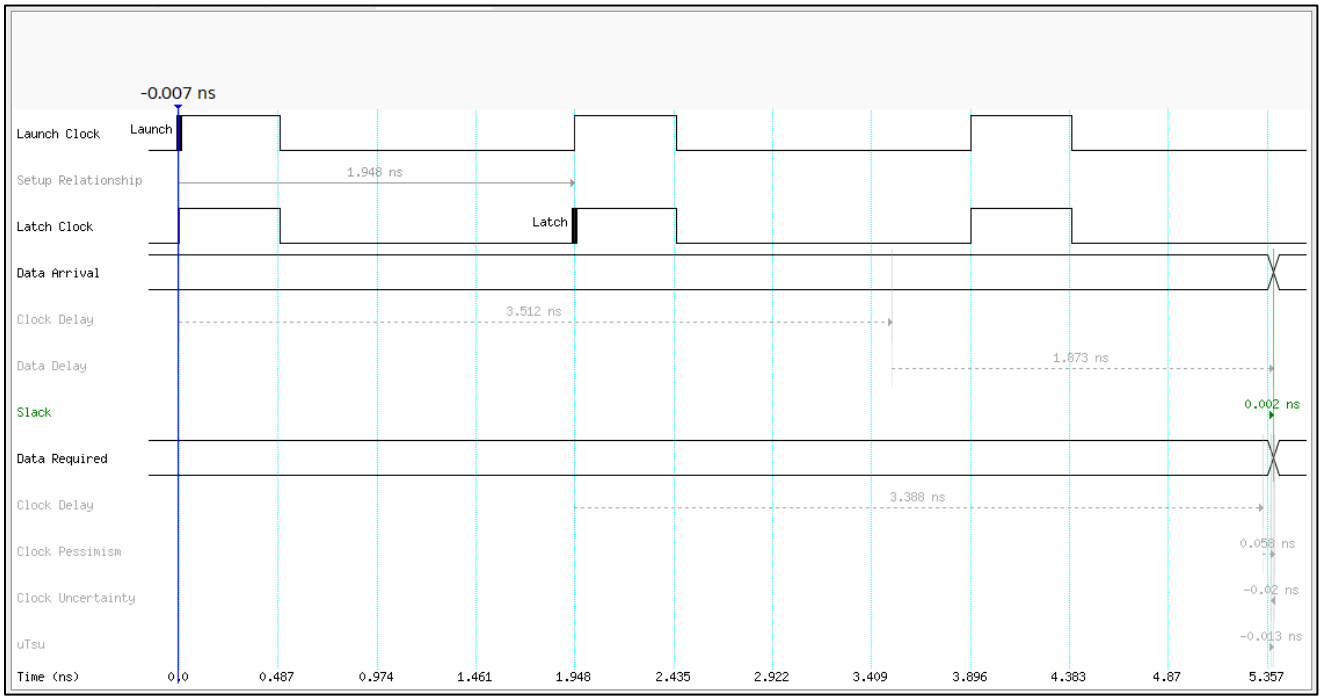
Compilation Summary

Flow Summary

<<Filter>>

Flow Status	Successful - Thu Feb 15 20:44:05 2024
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	Lab3
Top-level Entity Name	Lab3
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	96 / 49,760 (< 1 %)
Total registers	63
Total pins	37 / 360 (10 %)
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 %)

TIMING ANALYSIS DIAGRAMS

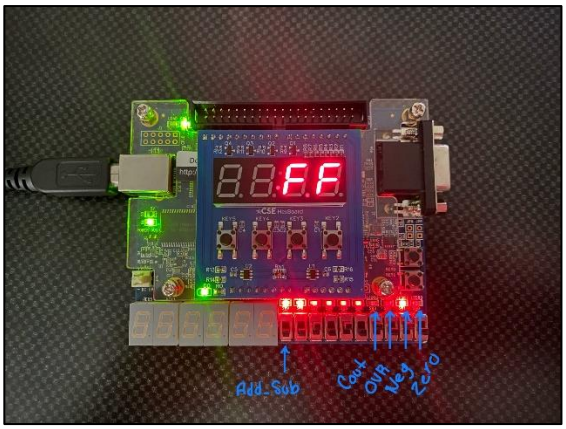


TEST RESULTS

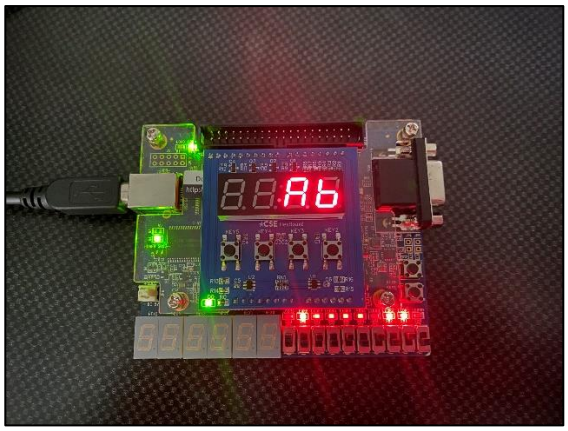
	A	+/-	B	R_HEX	R_Binary	Cout	OVR	Neg	Zero
a	0101 0101	+	1010 1010	FF	1111 1111	0	0	1	0
	0101 0101	−	1010 1010	AB	1010 1011	0	1	1	0
b	0111 1111	+	0000 0001	80	1000 0000	0	1	1	0
	0111 1111	−	0000 0001	7E	0111 1110	1	0	0	0
c	0111 1111	+	1111 1111	7E	0111 1110	1	0	0	0
	0111 1111	−	1111 1111	80	1000 0000	0	1	1	0
d	0110 0110	+	1101 1101	43	0100 0011	1	0	0	0
	0110 0110	−	1101 1101	89	1000 1001	0	1	1	0
e	1010 1010	+	0011 0011	DD	1101 1101	0	0	1	0
	1010 1010	−	0011 0011	77	0111 0111	1	1	0	0
f	1001 1001	+	1110 1110	87	1000 0111	1	0	1	0
	1001 1001	-	1110 1110	AB	1010 1011	0	0	1	0
g	1010 1010	+	1111 1110	A8	1010 1000	1	0	1	0
	1010 1010	−	1111 1110	AC	1010 1100	0	0	1	0
h	1101 1101	+	1100 1100	A9	1010 1001	1	0	1	0
	1101 1101	−	1100 1100	11	0001 0001	1	0	0	0

PHOTOS OF MARKED TEST RESULTS

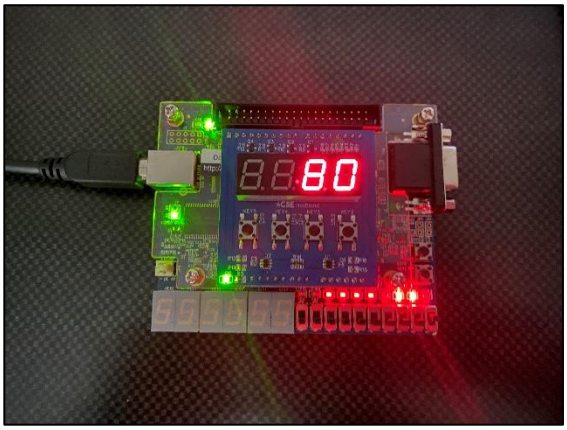
a) **(+)**



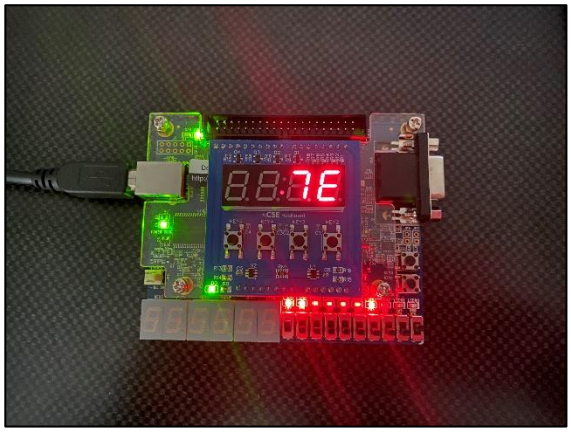
(-)



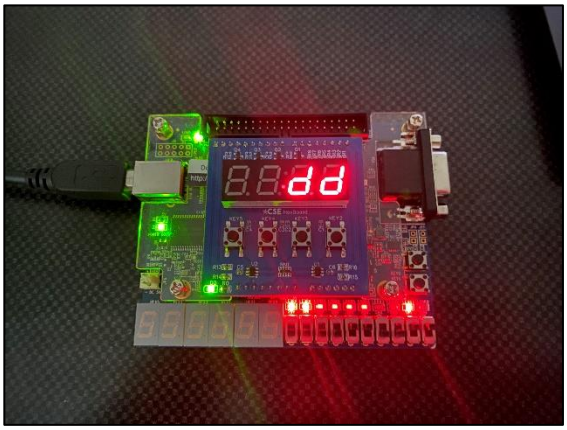
b) **(+)**



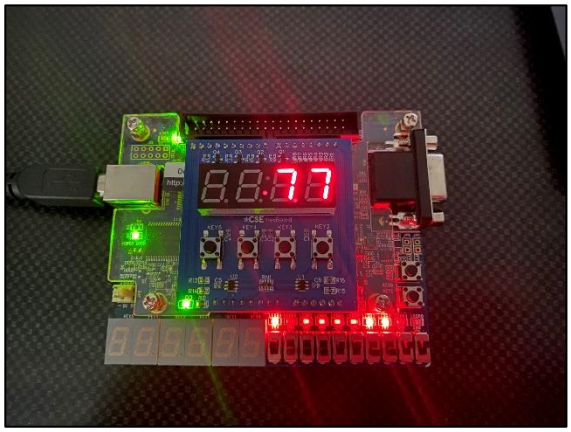
(-)



e) **(+)**



(-)



h) **(+)** **(-)**