Name: _____Servando_Olvera_____ ID# _____1001909287_____

Date Submitted: _____04-27-2024____ Time Submitted _____4:00_pm_____

CSE 3341 Digital Logic Design II

CSE 5357 Advanced Digital Logic Design

Spring Semester 2024

Eight-Bit, Four Function Calculator

250+ points

Due Date – April 30, 2024, 11:59 PM

Submit on Canvas Assignments

DESIGN REQUIREMENTS

PURPOSE/OUTCOMES

To design, implement on the DE10-Lite + KeyPad + HexBoard, and test an eight-bit, four-function (ADD, SUBTRACT, MULTIPLY, DIVIDE) calculator. The calculator can be partitioned in to four components as illustrated in Figure 1. You will perform this project by designing the Control Unit (CU) and integrating it with the Arithmetic Unit (AU), Output Unit (OU), and Input Unit (IU) designed in previous assignments. By successfully completing this project, you will have demonstrated an ability to design, implement, and test a machine that incorporates combinational and sequential logic circuits implemented with field programmable logic arrays (FPGAs) and designed with the SystemVerilog hardware description language (HDL).

REQUIREMENTS

To successfully complete the term project, you must design the CU and integrate it with the IU, AU, and OU and demonstrate it's functionality with the test inputs specified below.

a. 74 + 35
b. 74 – 35
c. –74 – 35
d. 127 + 6
e. 10 x 15
f. 127 x 2
g. –1 x –1
h. –127 x –127
i. 10 ÷ 5
j. 67 ÷ 32

INPUT REQUIREMENTS

1. Operands must be entered in hexadecimal on the CSE KeyPad 4 x 4 keypad shown in Figure 2.
2. Control and operations must be entered using pushbuttons as described below.

NUMBER ENTRY AND OPERATIONS REQUIREMENTS

Inputs consist of a sign and up to three magnitude digits. Enter operands and perform operations as follows.

1. Clear the calculator – press Key0 (Clear All)
2. Capture operand A using the KeyPad
3. Enter operand Ahigh – press Key1
4. Enter operand Alow – press Key1
5. Capture operand B using the KeyPad
6. Enter the operand B – press Key1
7. Enter the operation to be performed – add (Key2), subtract (Key3), multiply (Key4), divide (Key5)

DISPLAY REQUIREMENTS

1. Operands and results must be displayed in hexadecimal on the HexBoard.
2. Results must be displayed in decimal sign-magnitude on the DE10 seven-segment displays after calculations are complete.
3. Operations that produce an oveflow must light LEDR9.
4. Operations that produce a zero (0) result must light LEDR8.

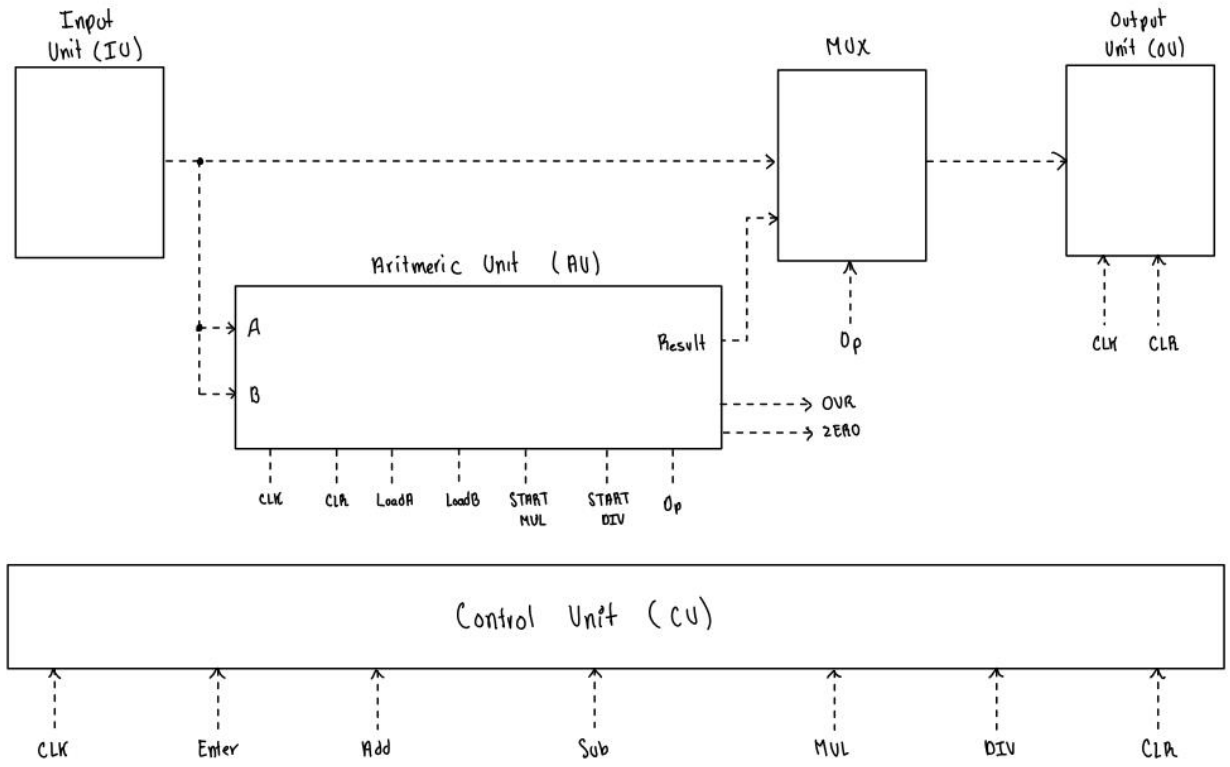
CONTROL UNIT REQUIREMENTS

The control-path, data-path interface is shown in Figure 3. The control signals are produced by a finite state machine described in Figure 4.

DESCRIPTION OF ADDED FEATURES

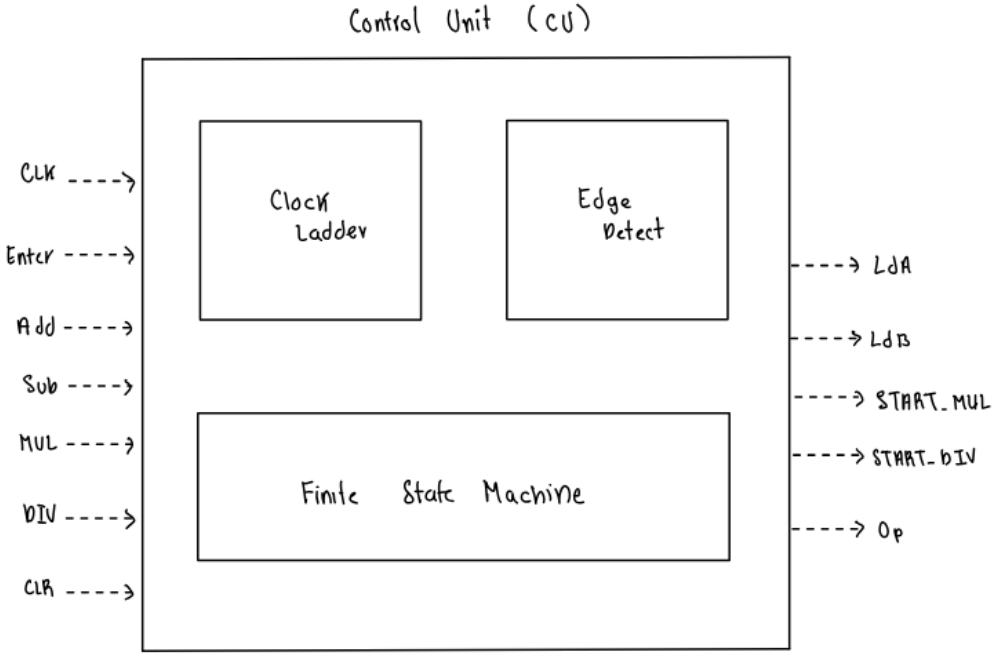
N/A
No added features.

ORGANIZATION DIAGRAM

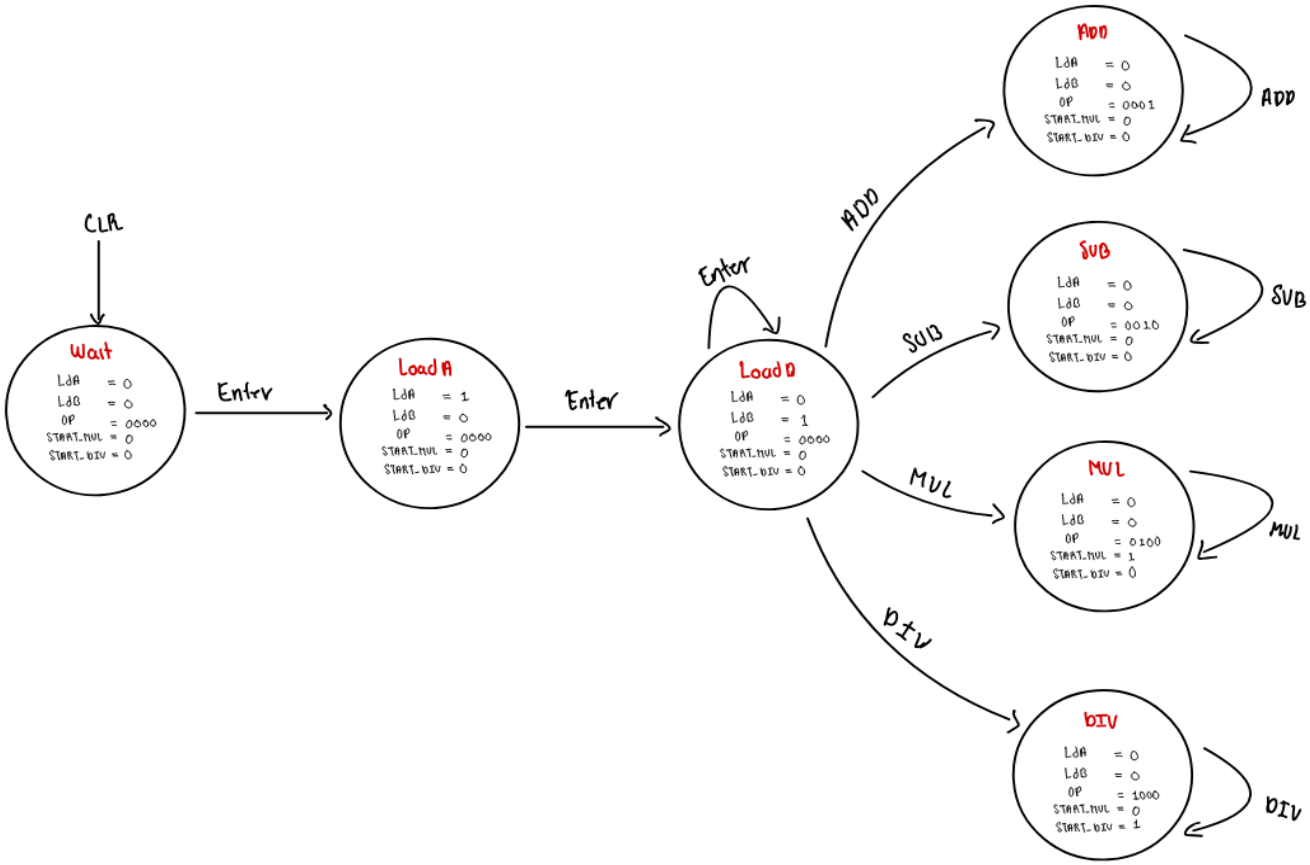


CONTROL UNIT

Inputs/Output



State Diagram

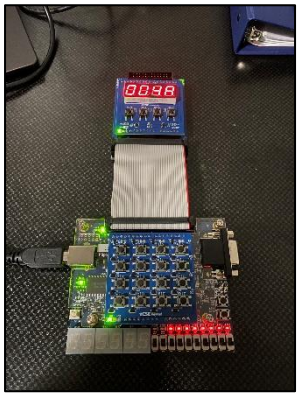
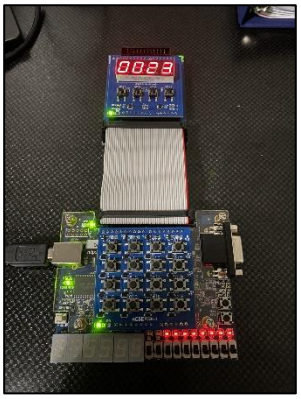
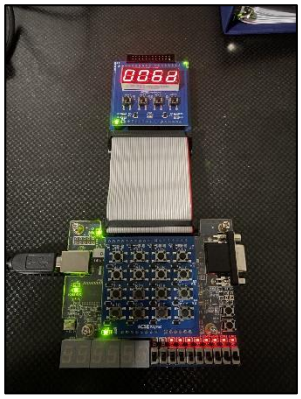
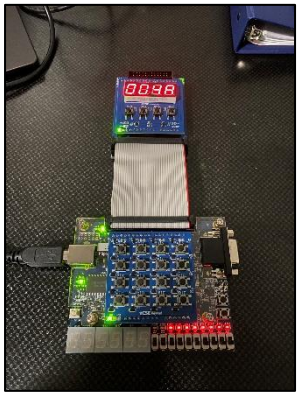
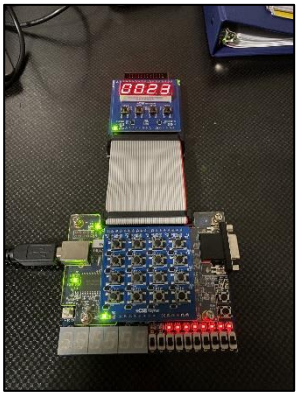
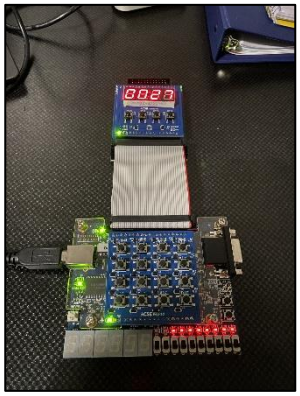
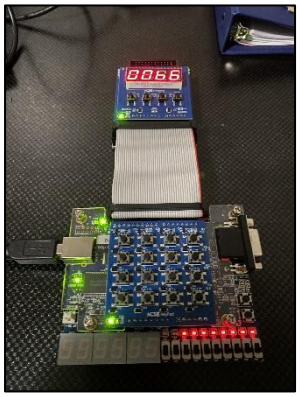
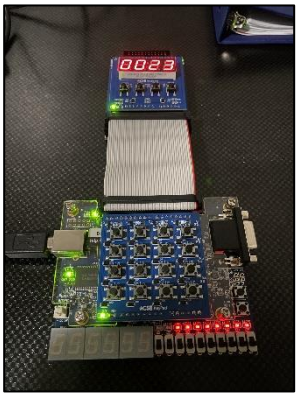
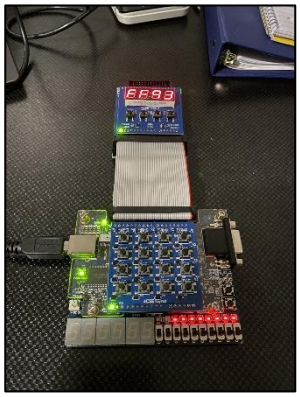
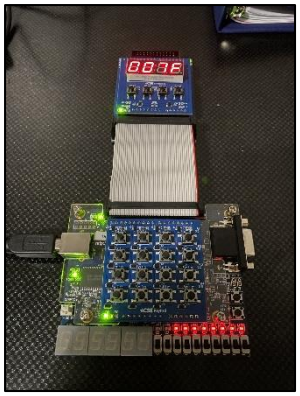
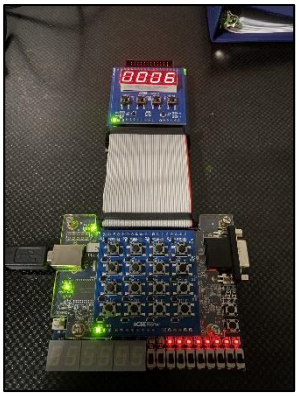


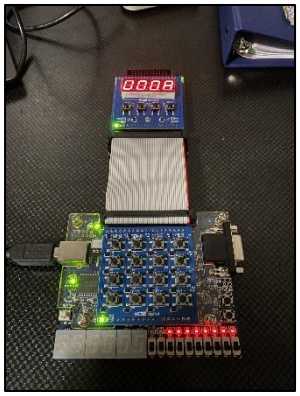
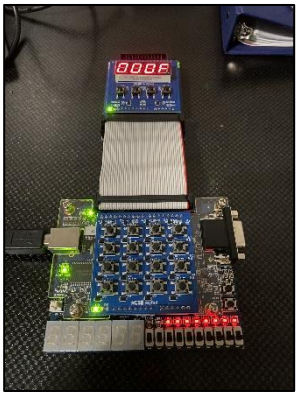
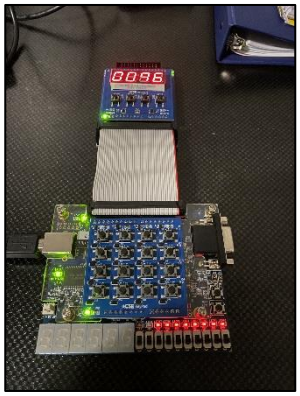
TEST RESULTS

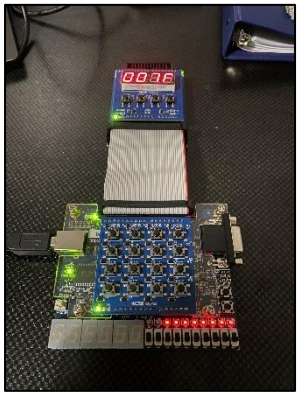
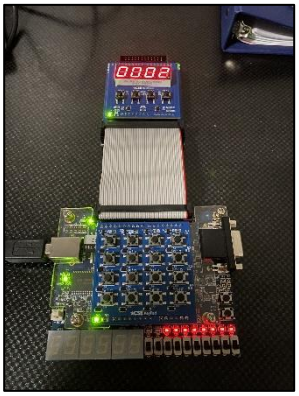
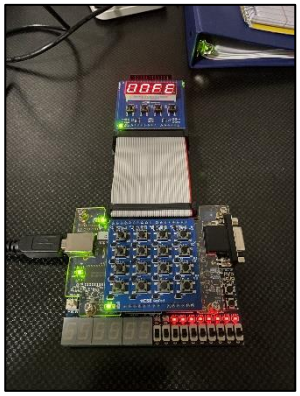
Summary

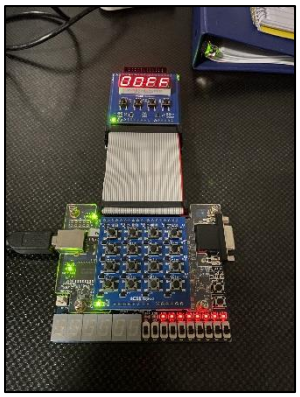
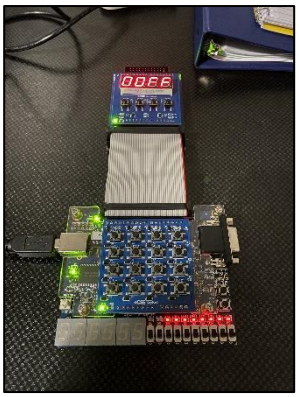
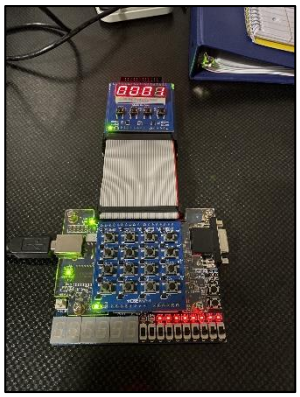
Test Case	Inputs (Decimal)	Expected Result (Decimal)	Expected Result (Hex)	Actual Inputs (Hex)	Actual Result	OVR	ZERO
a.	74 + 35	109	006D	4A + 23	006D	0	0
b.	74 - 35	39	0027	4A - 23	0027	0	0
c.	-74 - 35	-109	FF93	B6 - 23	FF93	0	0
d.	127 + 6	133	0085	7F + 06	FF85	1	0
e.	10 x 15	150	0096	0A x 0F	0096	0	0
f.	127 x 2	254	00FE	7F x 02	00FE	0	0
g.	-1 x -1	1	0001	FF x FF	0001	0	0
h.	-127 x -127	16,129	3F01	81 x 81	3F01	0	0
i.	10 ÷ 5	2	0002	0A ÷ 05	02|00	0	0
j.	67 ÷ 32	2|3	02|01	43 ÷ 20	02|03	0	0

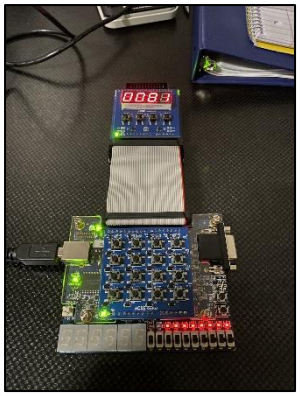
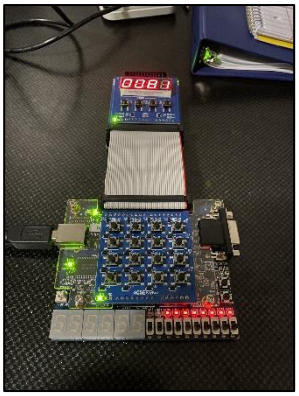
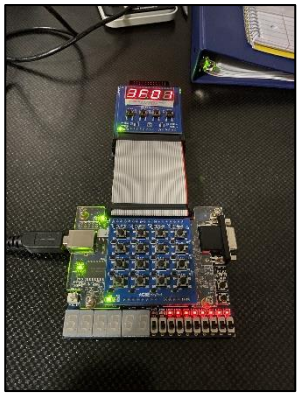
Demonstration (Documented Pictures)

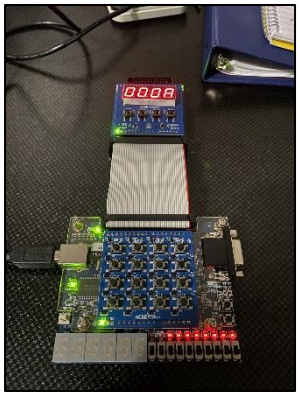
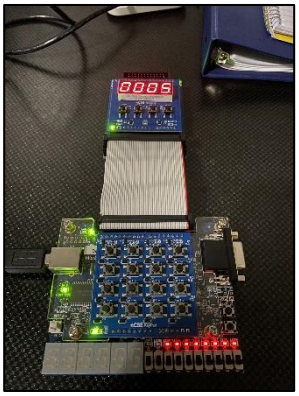
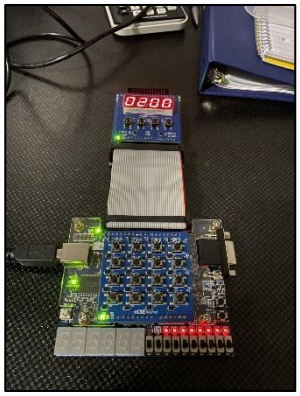
Test Case	Input A	Input B	Op	Result
a.			+	
b.			-	
c.			-	
d.			-	

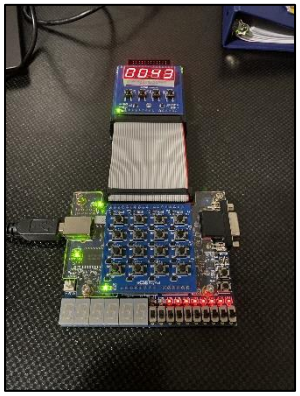
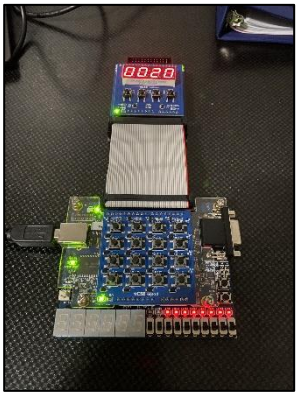
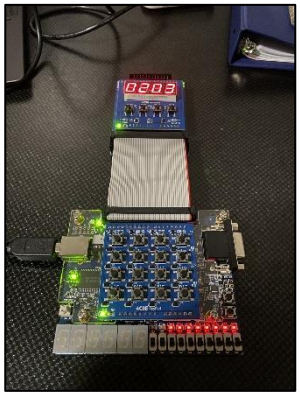
e.   + 

f.   x 

g.   x 

h.   x 

i.   ÷ 

j.   ÷ 

VERILOG CODE

Term Project/ Top Module

```
module TermProject
(
    input CLK, CLR, Enter, Add, Sub, Mul, Div,
    input [3:0] ROW,
    output logic [3:0] COL,
    output logic [0:6] HEX,
    output logic [3:0] CAT,
    output logic OVR, ZERO
);

    logic LdA, LdB, START_MUL, START_DIV;
    logic [3:0] OP;

    CU control_Unit         // Calculator Control Unit
    (
        .CLK(CLK),
        .CLR(CLR),
        .Enter(Enter),
        .Add(Add),
        .Sub(Sub),
        .Mul(Mul),
        .Div(Div),
        .LdA(LdA),
        .LdB(LdB),
        .START_MUL(START_MUL),
        .START_DIV(START_DIV),
        .OP(OP)
    );

    logic [15:0] keyPad;

    keypad_input Input_Unit // Caculator Input Unit
    (
        .clk(CLK),
        .reset(CLR),
        .row(ROW),
        .col(COL),
        .out(keyPad)
    );

    AU Arithmetic_Unit      // Calculator Arithmetic Unit
    (
        .CLK(CLK),
        .CLR(CLR),
        .LdA(LdA),
        .LdB(LdB),
        .START_MUL(START_MUL),
        .START_DIV(START_DIV),
        .X(keyPad),
        .OP(OP),
        .Rout(OUT),
        .OVR(OVR),
        .ZERO(ZERO)

    );

    logic [15:0] OUT;

    Controller Output_Unit     // Display Inputs and Result
    (
        .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
```

Control Unit Module

```
module CU
(
    input CLK, CLR, Enter, Add, Sub, Mul, Div,
    output logic LdA, LdB, START_MUL, START_DIV,
    output logic [3:0] OP
);

    logic [2:0] state, nextstate;
    parameter WAIT = 3'b000, LoadA = 3'b001, LoadB = 3'b011, ADD = 3'b010, SUB = 3'b100, MUL = 3'b101, DIV = 3'b111;

    logic [31:0] ladder;
    logic EnterSync;


    clk_ladder slowerCLK
    (
       .CLK(CLK),
       .ladder(ladder)
    );

    EdgeDetect detect
    (
       .in(Enter),
       .clock(ladder[15]),  // 50MHz/2^(15+1) ~ 763 Hz
       .out(EnterSync)
    );

    always @(negedge ladder[15], negedge CLR) begin
       if (CLR == 0) state <= WAIT;
       else state <= nextstate;
    end

    always_comb
       case ({state})
          WAIT:    begin
                      if(EnterSync == 1'b1) nextstate <= LoadA;
                      else nextstate <= WAIT;
                                            LdA <= 1'b0; LdB <= 1'b0; OP <= 4'b0000; START_MUL <= 1'b0; START_DIV <= 1'b0; end
          LoadA:   begin
                      if(EnterSync == 1'b1) nextstate <= LoadB;
                      else nextstate <= LoadA;
                                            LdA <= 1'b1; LdB <= 1'b0; OP <= 4'b0000; START_MUL <= 1'b0; START_DIV <= 1'b0; end
          LoadB:   begin
                      if(Add == 0)      nextstate <= ADD;
                      else if(Sub == 0) nextstate <= SUB;
                      else if(Mul == 0) nextstate <= MUL;
                      else if(Div == 0) nextstate <= DIV;
                      else              nextstate <= LoadB;
                                            LdA <= 1'b0; LdB <= 1'b1; OP <= 4'b0000; START_MUL <= 1'b0; START_DIV <= 1'b0; end
          ADD:     begin nextstate <= ADD;    LdA <= 1'b0; LdB <= 1'b0; OP <= 4'b0001; START_MUL <= 1'b0; START_DIV <= 1'b0; end
          SUB:     begin nextstate <= SUB;    LdA <= 1'b0; LdB <= 1'b0; OP <= 4'b0010; START_MUL <= 1'b0; START_DIV <= 1'b0; end
          MUL:     begin nextstate <= MUL;    LdA <= 1'b0; LdB <= 1'b0; OP <= 4'b0100; START_MUL <= 1'b1; START_DIV <= 1'b0; end
          DIV:     begin nextstate <= DIV;    LdA <= 1'b0; LdB <= 1'b0; OP <= 4'b1000; START_MUL <= 1'b0; START_DIV <= 1'b1; end
          default: begin nextstate <= WAIT;   LdA <= 1'b0; LdB <= 1'b0; OP <= 4'b0000; START_MUL <= 1'b0; START_DIV <= 1'b0; end
       endcase

endmodule

```

MUX

```
module M_U_X
(
    input [3:0] OP,
    input logic [15:0] X, AddSub_R, Mul_R, Div_R,
    output [15:0] Rout
);

    always_comb begin
       if(OP[0]) Rout = AddSub_R;
       else
       if(OP[1]) Rout = AddSub_R;
       else
       if(OP[2]) Rout = Mul_R;
       else
       if(OP[3]) Rout = Div_R;
       else Rout = X;
    end

endmodule
```

Arithmetic Unit Module

```
module AU
(
    input CLK, CLR, LdA, LdB, START_MUL, START_DIV,
    input [15:0] X,
    input [3:0] OP,
    output logic [15:0] Rout,
    output logic OVR, ZERO
);

    logic [15:0] AddSub_R, P, Mul_R, Div_R;
    logic [15:0] A;
    logic [7:0] B, Quotient, Remainder;
    logic C7, Cout, Halt, DONE;

    NBitRegister #(16) InputA
    (
        .D(X),
        .CLK(~LdA),
        .CLR(CLR),
        .Q(A)
    );

    NBitRegister #(8) InputB
    (
        .D(X),
        .CLK(~LdB),
        .CLR(CLR),
        .Q(B)
    );

    //-----------------------------------------//
        GCLA_AddSub Add_Sub
        (
            .A(A[7:0]),
            .B(B),
            .Add_Sub(OP[1]),
            .R(AddSub_R),
            .Cout(Cout),
            .C7(C7)
        );
    //-----------------------------------------//
        Multiplier Mult
        (
            .Clock(CLK),
            .Reset(START_MUL),
            .Multiplicand(A[7:0]),
            .Multiplier(B),
            .Product(P),
            .Halt(Halt)
        );

        NBitRegister #(16) regR
        (
            .D(P),
            .CLK(~Halt),
            .CLR(CLR),
            .Q(Mul_R)
        );
    //-----------------------------------------//

        Divider DIVIDE
        (
            .Dividend(A),
            .Divisor(B),
            .Quotient(Quotient),
            .Remainder(Remainder),
            .CLOCK(CLK),
            .START(START_DIV),
            .DONE(DONE)
        );

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

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


    M_U_X MUX
    (
        .OP(OP),
        .AddSub_R(AddSub_R),
        .X(X),
        .Mul_R(Mul_R),
        .Div_R(Div_R),
        .Rout(Rout)
    );

    assign OVR = (C7 ^ Cout) && (OP[0] || OP[1]);
    assign ZERO = !Rout && OP;

endmodule
```

Clock Ladder Modified Module

```
1   module clk_ladder #(parameter N = 32)
2   (
3      input CLK,
4      output logic [N-1:0] ladder
5   );
6
7      always_ff @(negedge CLK)
8         ladder <= ladder + 1;
9
10  endmodule
```

Edge Detect Module

```
1   module EdgeDetect
2   (
3      input in, clock,
4      output out
5   );
6
7      logic in_delay;
8
9      always @ (negedge clock)
10        in_delay <= in;
11
12     assign out = in & ~in_delay;
13
14  endmodule
```