

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. 10×15
- f. 127×2
- g. -1×-1
- h. -127×-127
- i. $10 \div 5$
- j. $67 \div 32$

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 overflow must light LEDR9.
4. Operations that produce a zero (0) result must light LEDR8.

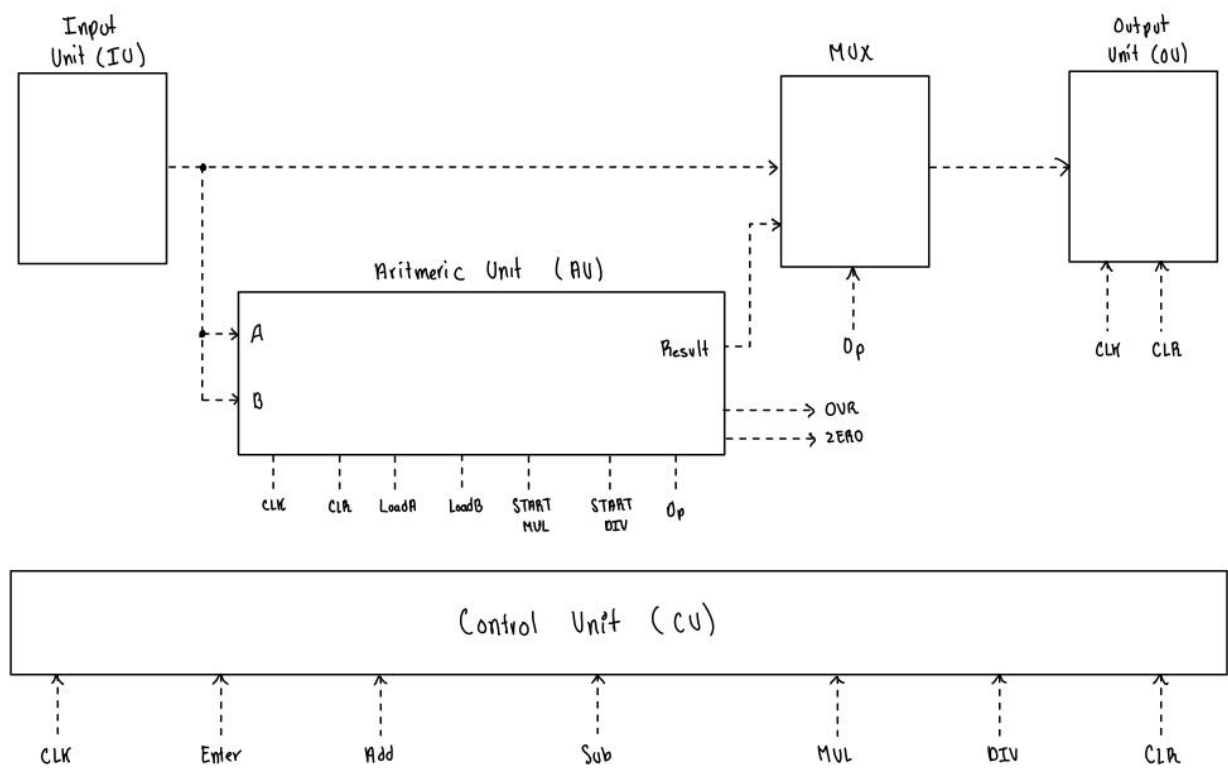
CONTROL UNIT REQUIREMENTS

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

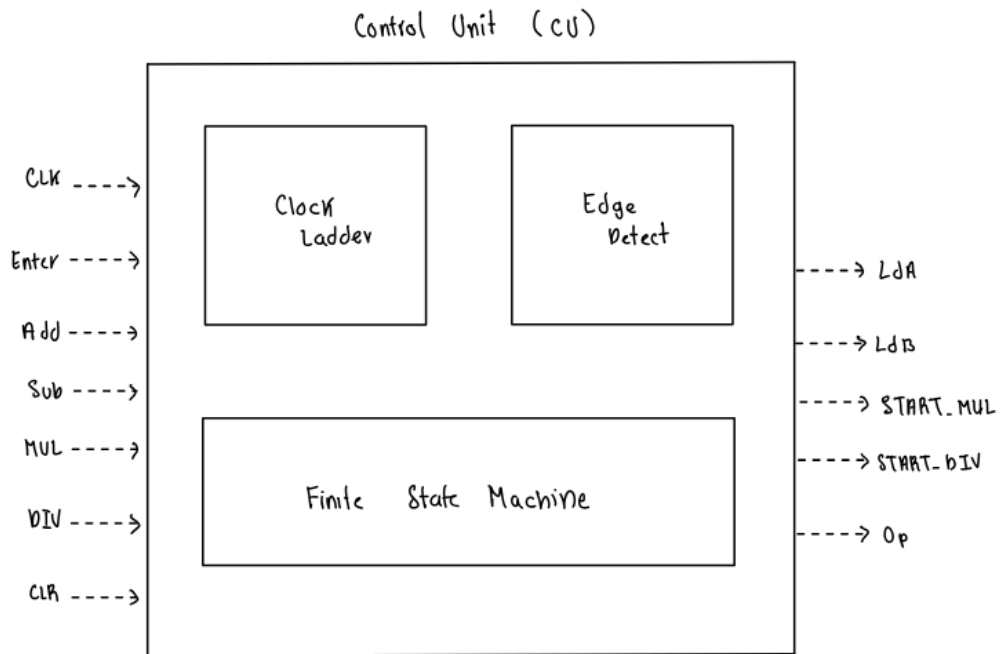
N/A
No added features.

ORGANIZATION DIAGRAM

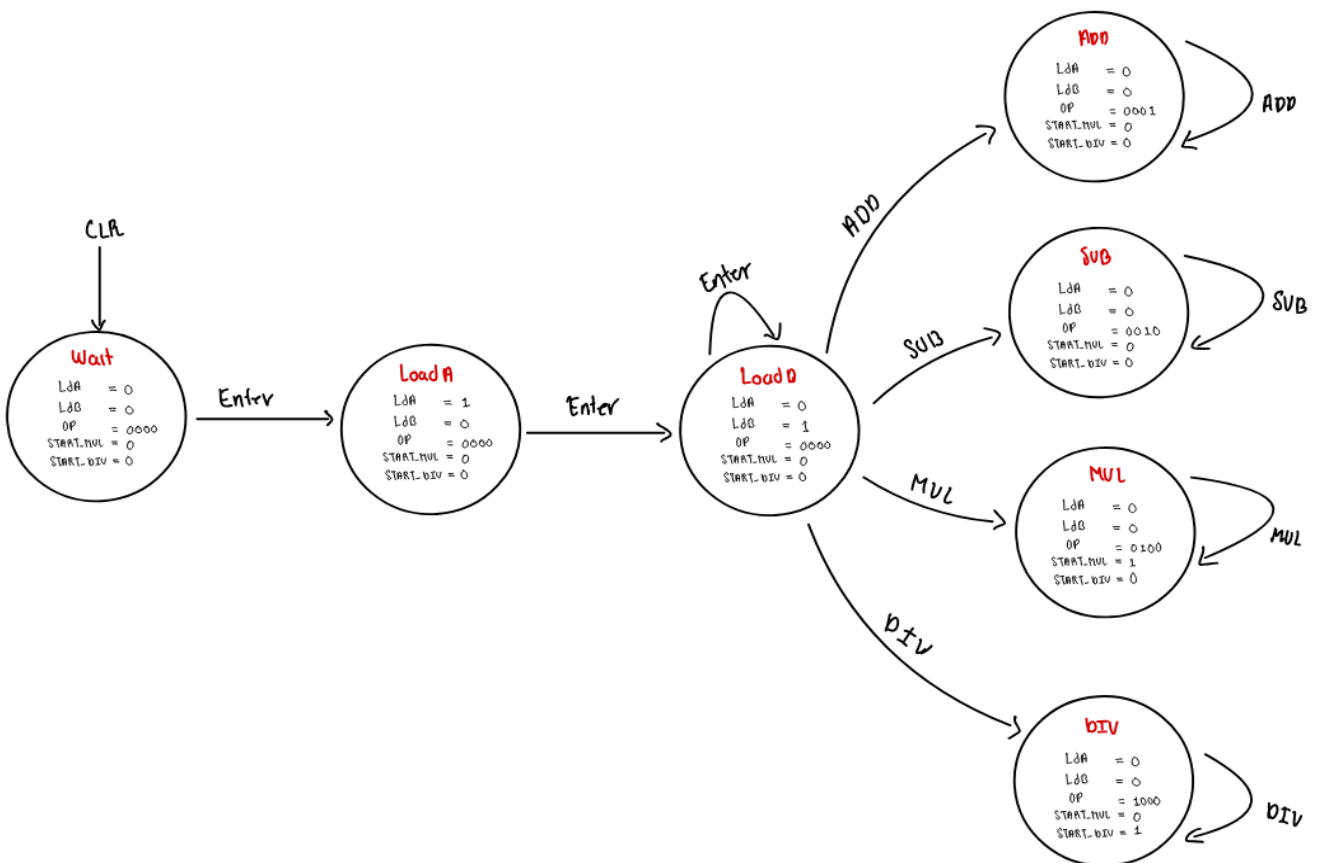


CONTROL UNIT

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.	10×15	150	0096	$0A \times 0F$	0096	0	0
f.	127×2	254	00FE	$7F \times 02$	00FE	0	0
g.	-1×-1	1	0001	$FF \times FF$	0001	0	0
h.	-127×-127	16,129	3F01	81×81	3F01	0	0
i.	$10 \div 5$	2 0	02 00	$0A \div 05$	02 00	0	0
j.	$67 \div 32$	2 3	02 03	$43 \div 20$	02 03	0	0

Demonstration (Documented Pictures)

Test
Case

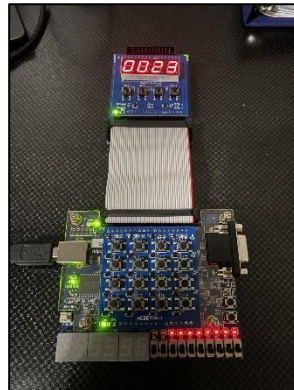
Input A

Input B

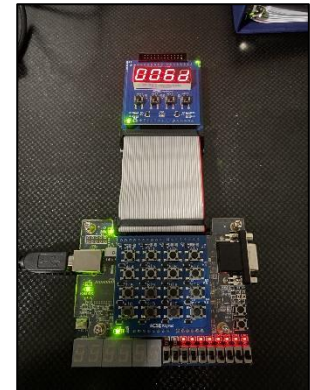
Op

Result

a.



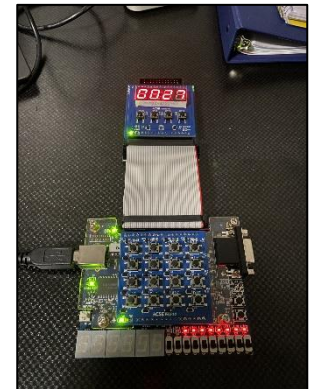
+



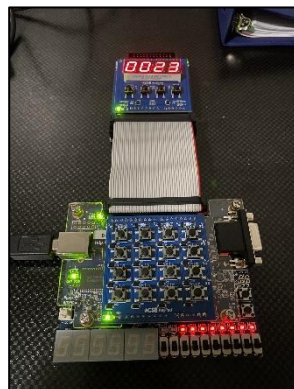
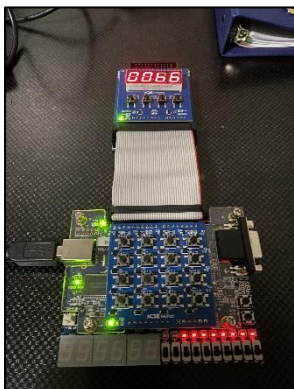
b.



-



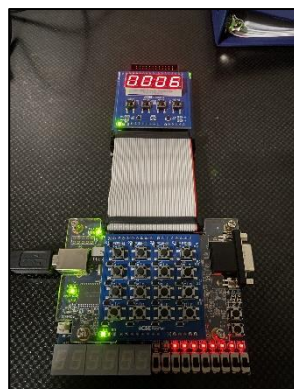
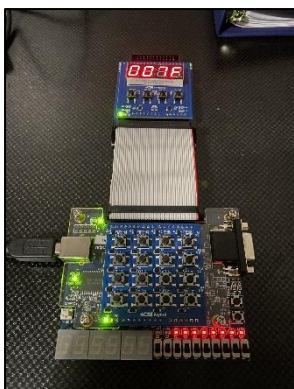
c.



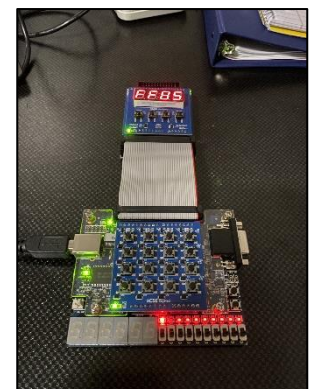
-



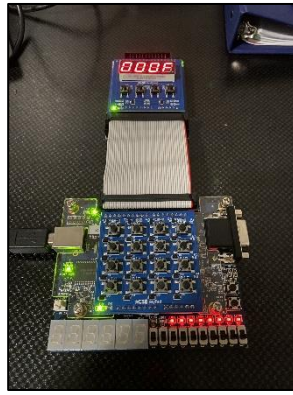
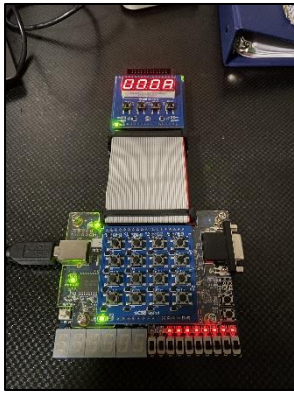
d.



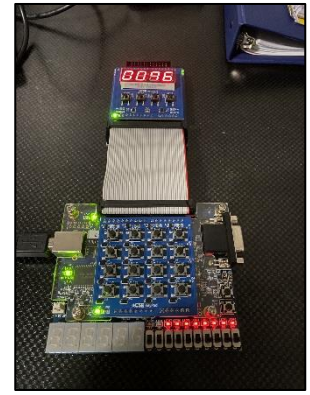
-



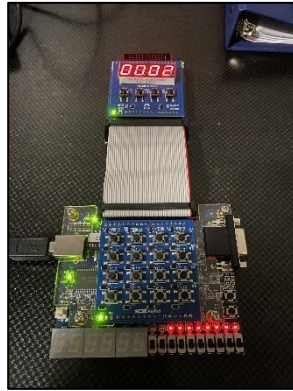
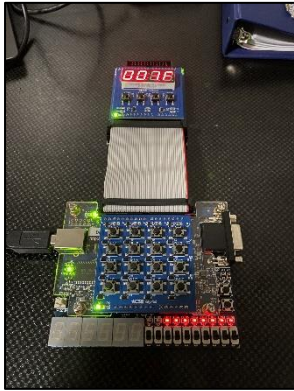
e.



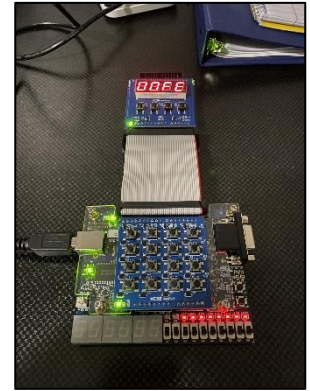
+



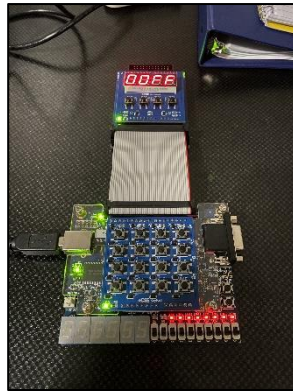
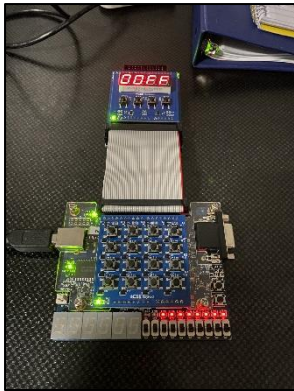
f.



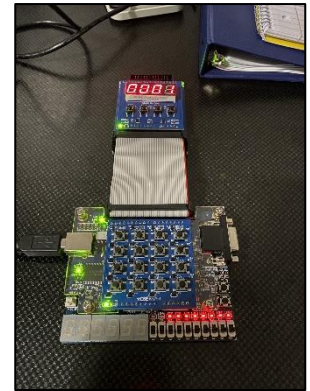
x



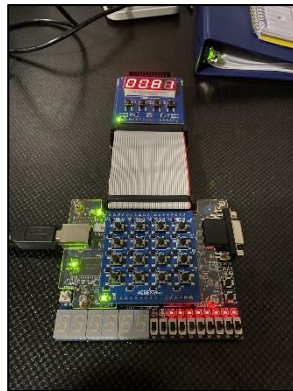
g.



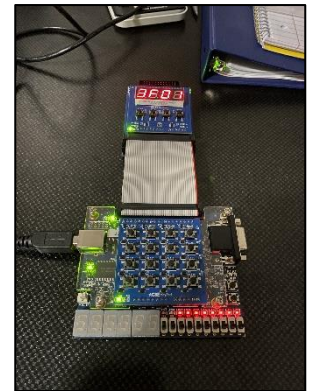
x



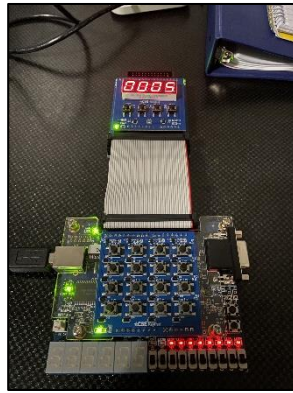
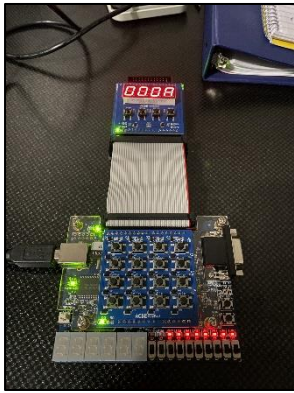
h.



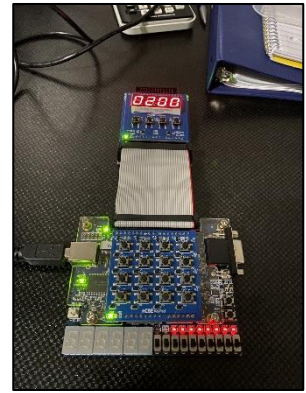
x



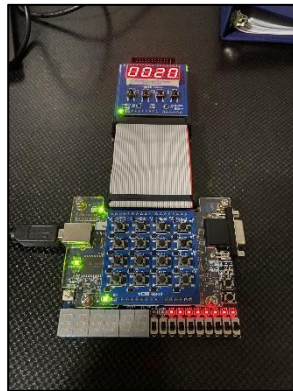
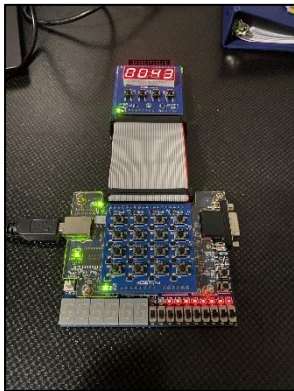
i.



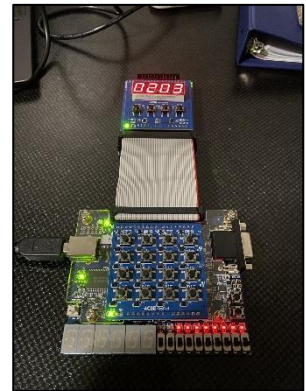
÷



j.



÷



VERILOG CODE

Term Project/ Top Module

```
1  module TermProject
2  (
3      input CLK, CLR, Enter, Add, Sub, Mul, Div,
4      input [3:0] ROW,
5      output logic [3:0] COL,
6      output logic [0:6] HEX,
7      output logic [3:0] CAT,
8      output logic OVR, ZERO
9  );
10
11
12      logic LdA, LdB, START_MUL, START_DIV;
13      logic [3:0] OP;
14
15      CU control_Unit          // Calculator Control Unit
16      (
17          .CLK(CLK),
18          .CLR(CLR),
19          .Enter(Enter),
20          .Add(Add),
21          .Sub(Sub),
22          .Mul(Mul),
23          .Div(Div),
24          .LdA(LdA),
25          .LdB(LdB),
26          .START_MUL(START_MUL),
27          .START_DIV(START_DIV),
28          .OP(OP)
29      );
30
31      logic [15:0] keyPad;
32
33      keypad_input Input_Unit // Calculator Input Unit
34      (
35          .clk(CLK),
36          .reset(CLR),
37          .row(ROW),
38          .col(COL),
39          .out(keyPad)
40      );
41
42      AU Arithmetic_Unit      // Calculator Arithmetic Unit
43      (
44          .CLK(CLK),
45          .CLR(CLR),
46          .LdA(LdA),
47          .LdB(LdB),
48          .START_MUL(START_MUL),
49          .START_DIV(START_DIV),
50          .X(keyPad),
51          .OP(OP),
52          .Rout(OUT),
53          .OVR(OVR),
54          .ZERO(ZERO)
55      );
56
57
58      logic [15:0] OUT;
59
60      Controller Output_Unit  // Display Inputs and Result
61      (
62          .CLK(CLK),
63          .CLEAR(CLR),
64          .MODE(1'b1),
65          .D0(OUT[3:0]),
66          .D1(OUT[7:4]),
67          .D2(OUT[11:8]),
68          .D3(OUT[15:12]),
69          .CAT(CAT),
70          .HEX(HEX)
71      );
72
73  endmodule
```

Control Unit Module

```

1  module CU
2  (
3      input CLK, CLR, Enter, Add, Sub, Mul, Div,
4      output logic LdA, LdB, START_MUL, START_DIV,
5      output logic [3:0] OP
6  );
7
8      logic [2:0] state, nextstate;
9      parameter WAIT = 3'b000, LoadA = 3'b001, LoadB = 3'b011, ADD = 3'b010, SUB = 3'b100, MUL = 3'b101, DIV = 3'b111;
10
11      logic [31:0] ladder;
12      logic EnterSync;
13
14
15      clk_ladder slowerCLK
16      (
17          .CLK(CLK),
18          .ladder(ladder)
19      );
20
21      EdgeDetect detect
22      (
23          .in(Enter),
24          .clock(ladder[15]), // 50MHz/2^(15+1) ~ 763 Hz
25          .out(EnterSync)
26      );
27
28      always @(negedge ladder[15], negedge CLR) begin
29          if (CLR == 0) state <= WAIT;
30          else state <= nextstate;
31      end
32
33      always_comb
34      case ({state})
35          WAIT: begin
36              if (EnterSync == 1'b1) nextstate <= LoadA;
37              else nextstate <= WAIT;
38              LdA <= 1'b0; LdB <= 1'b0; OP <= 4'b0000; START_MUL <= 1'b0; START_DIV <= 1'b0; end
39          LoadA: begin
40              if (EnterSync == 1'b1) nextstate <= LoadB;
41              else nextstate <= LoadA;
42              LdA <= 1'b1; LdB <= 1'b0; OP <= 4'b0000; START_MUL <= 1'b0; START_DIV <= 1'b0; end
43          LoadB: begin
44              if (Add == 0) nextstate <= ADD;
45              else if (Sub == 0) nextstate <= SUB;
46              else if (Mul == 0) nextstate <= MUL;
47              else if (Div == 0) nextstate <= DIV;
48              else nextstate <= LoadB;
49              LdA <= 1'b0; LdB <= 1'b1; OP <= 4'b0000; START_MUL <= 1'b0; START_DIV <= 1'b0; end
50          ADD: begin nextstate <= ADD; LdA <= 1'b0; LdB <= 1'b0; OP <= 4'b0001; START_MUL <= 1'b0; START_DIV <= 1'b0; end
51          SUB: begin nextstate <= SUB; LdA <= 1'b0; LdB <= 1'b0; OP <= 4'b0010; START_MUL <= 1'b0; START_DIV <= 1'b0; end
52          MUL: begin nextstate <= MUL; LdA <= 1'b0; LdB <= 1'b0; OP <= 4'b0100; START_MUL <= 1'b1; START_DIV <= 1'b0; end
53          DIV: begin nextstate <= DIV; LdA <= 1'b0; LdB <= 1'b0; OP <= 4'b1000; START_MUL <= 1'b0; START_DIV <= 1'b1; end
54          default: begin nextstate <= WAIT; LdA <= 1'b0; LdB <= 1'b0; OP <= 4'b0000; START_MUL <= 1'b0; START_DIV <= 1'b0; end
55      endcase
56
57  endmodule
58

```

MUX

```

1  module M_U_X
2  (
3      input [3:0] OP,
4      input logic [15:0] X, AddSub_R, Mul_R, Div_R,
5      output [15:0] Rout
6  );
7
8      always_comb begin
9          if (OP[0]) Rout = AddSub_R;
10         else
11             if (OP[1]) Rout = AddSub_R;
12             else
13                 if (OP[2]) Rout = Mul_R;
14                 else
15                     if (OP[3]) Rout = Div_R;
16                     else Rout = X;
17             end
18         end
19     endmodule

```

Arithmetic Unit Module

```

1  module AU
2  (
3      input CLK, CLR, LdA, LdB, START_MUL, START_DIV,
4      input [15:0] X,
5      input [3:0] OP,
6      output logic [15:0] Rout,
7      output logic OVR, ZERO
8  );
9
10     logic [15:0] AddSub_R, P, Mul_R, Div_R;
11     logic [15:0] A;
12     logic [7:0] B, Quotient, Remainder;
13     logic C7, Cout, Halt, DONE;
14
15     NBitRegister #(16) InputA
16     (
17         .D(X),
18         .CLK(~LdA),
19         .CLR(CLR),
20         .Q(A)
21     );
22
23     NBitRegister #(8) InputB
24     (
25         .D(X),
26         .CLK(~LdB),
27         .CLR(CLR),
28         .Q(B)
29     );
30
31     //-----//
32     GCLA_AddSub Add_Sub
33     (
34         .A(A[7:0]),
35         .B(B),
36         .Add_Sub(OP[1]),
37         .R(AddSub_R),
38         .Cout(Cout),
39         .C7(C7)
40     );
41     //-----//
42     Multiplier Mult
43     (
44         .Clock(CLK),
45         .Reset(START_MUL),
46         .Multiplicand(A[7:0]),
47         .Multiplier(B),
48         .Product(P),
49         .Halt(Halt)
50     );
51
52     NBitRegister #(16) regR
53     (
54         .D(P),
55         .CLK(~Halt),
56         .CLR(CLR),
57         .Q(Mul_R)
58     );
59     //-----//
60
61     Divider DIVIDE
62     (
63         .Dividend(A),
64         .Divisor(B),
65         .Quotient(Quotient),
66         .Remainder(Remainder),
67         .CLOCK(CLK),
68         .START(START_DIV),
69         .DONE(DONE)
70     );
71
72     NBitRegister QUOTIENT // Quotient
73     (
74         .D(Quotient),
75         .CLK(~DONE),
76         .CLR(CLR),
77         .Q(Div_R[15:8])
78     );
79
80     NBitRegister REMAINDER // Remainder
81     (
82         .D(Remainder),
83         .CLK(~DONE),
84         .CLR(CLR),
85         .Q(Div_R[7:0])
86     );
87     //-----//
88
89     M_U_X MUX
90     (
91         .OP(OP),
92         .AddSub_R(AddSub_R),
93         .X(X),
94         .Mul_R(Mul_R),
95         .Div_R(Div_R),
96         .Rout(Rout)
97     );
98
99     assign OVR = (C7 ^ Cout) && (OP[0] || OP[1]);
100    assign ZERO = !Rout && OP;
101
102 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
```