Name: __Temitayo Aderounmu_____ ID# _____1001568524_____

Date Submitted: __04/28/2023_____ Time Submitted _____11:59_____

CSE 4357/5357

Advanced Digital Logic Design

Spring Semester 2023

Term Project

Eight-Bit, Two-Function Calculator

Due Date – May 2, 2023, 11:59 PM

Submit on Canvas Assignments

DESIGN REQUIREMENTS

Design, implement on the DE10-Lite + Keypad (+Hex Board), 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. This project is performed 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. 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 System Verilog hardware description language (HDL).

INPUT REQUIREMENTS

1. Inputs must be entered on the CSE Keypad 4 x 4 keypad as shown in Figure 2.
2. Non-negative numbers must be entered in decimal sign-magnitude using a blank for the sign and may be entered with or without leading zeros for the magnitude. Examples 027 or 27.
3. Negative numbers must be entered in decimal sign-magnitude with * for the sign followed by the magnitude with leading zeros. Examples *027, *100, and *001.

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 the operand and operation – press key A, B, C, or D (see below for key definitions)
4. Capture operand B using the Keypad
5. Enter the operand and start the operation – press #
6. Clear Entry – implement a Clear Entry function (Key1)

DISPLAY REQUIREMENTS

1. Operands must be displayed in decimal sign-magnitude on the DE10 seven-segment displays (HEX3, HEX2, HEX1, and HEX0) upon entry.
2. Results must be displayed in decimal sign-magnitude on the DE10 seven-segment displays after calculations are complete.
3. non-negative results must be displayed without a sign and without leading zeros.
4. Negative results must be displayed with a minus sign (-) followed by the magnitude with or without leading zeros.
5. Operations that produce overflows must be indicated by lighting LEDR9.

ORGANIZATION DIAGRAM

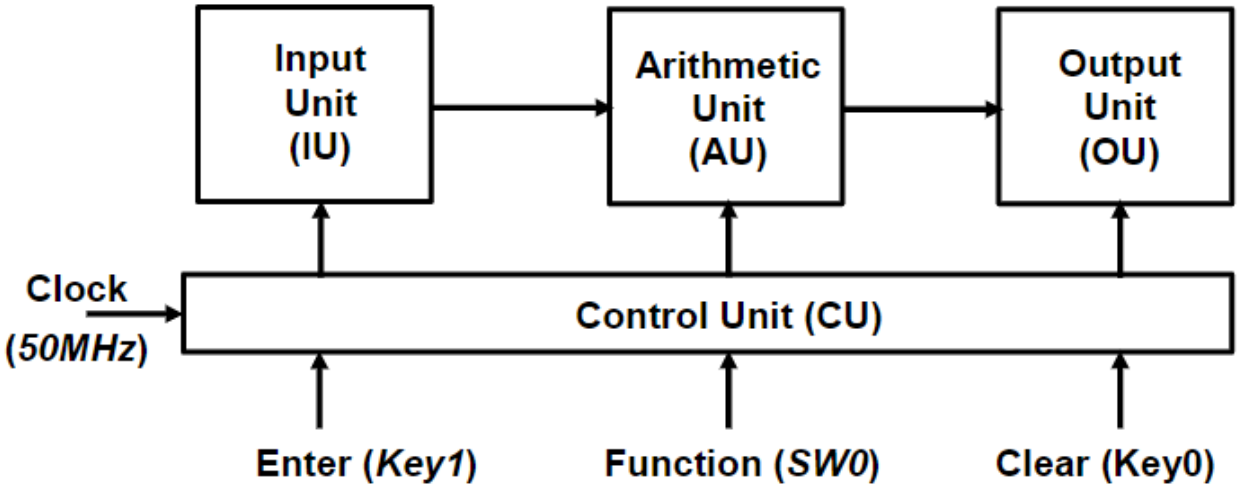

Figure 1 – Calculator functional units

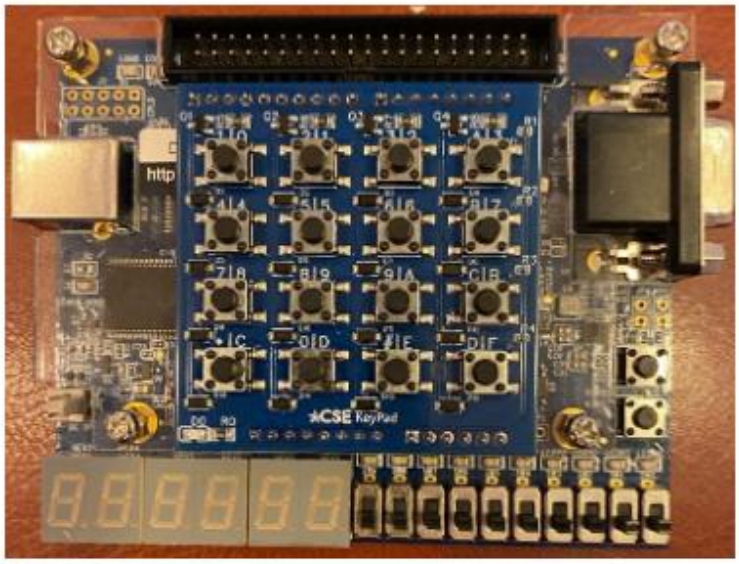

Figure 2 – KeyPad plugged in to DE10-Lite Arduino port

DESCRIPTION OF ADDED FEATURES

I added an invalid output feature that displays [Er] if an invalid input is logged into the input unit.

The explanatory code is showed below in its section.

CONTROL UNIT

The control signals are produced by a finite state machine.

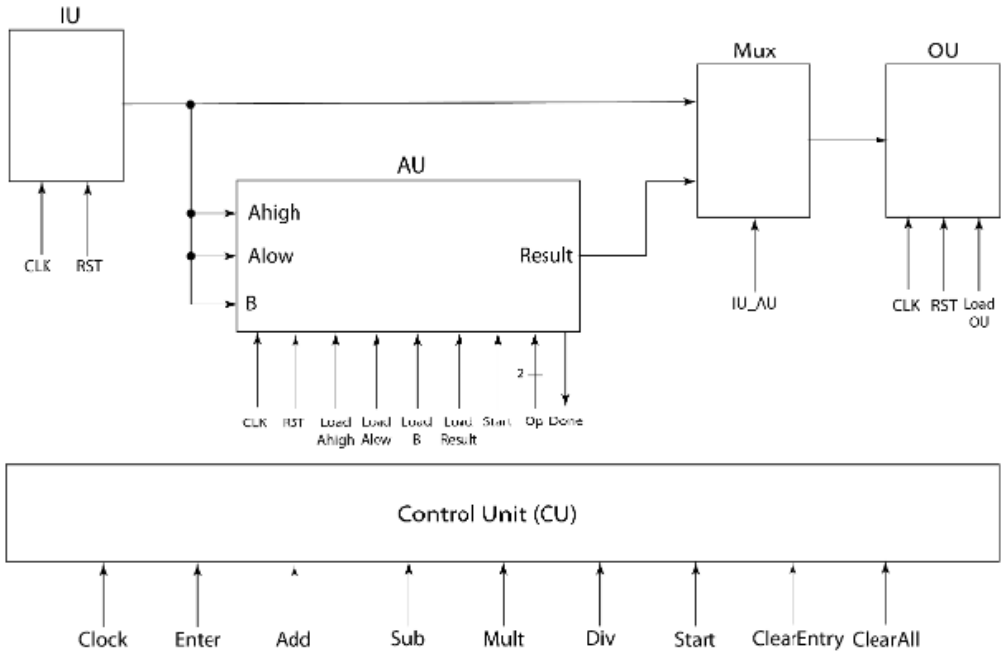

Figure 3 – Control-Path, Data-Path Interface

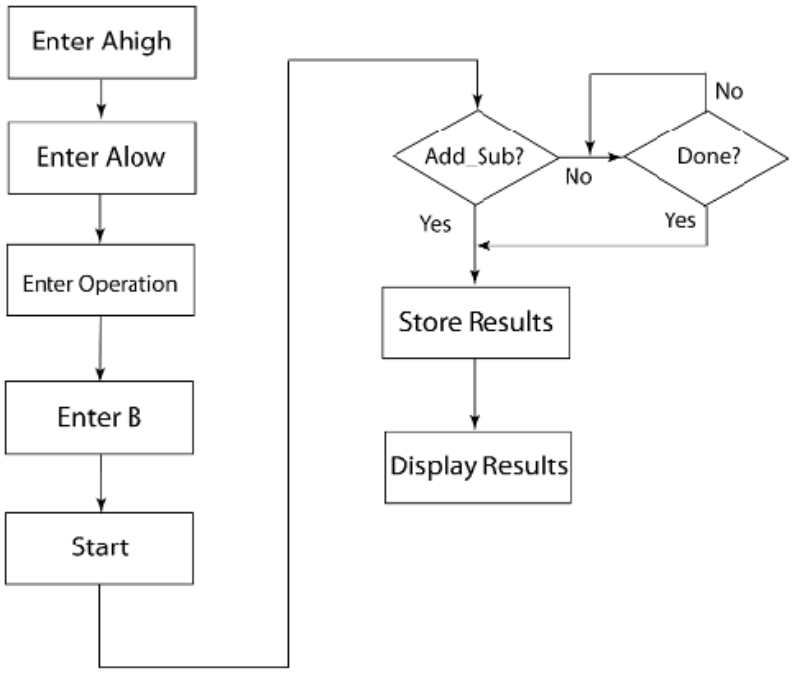

Figure 4 – Control Unit Control Flow

TEST RESULTS

OPERATION	RESULTS
74 + 35	109
74 - 35	39
-74 - 35	-109
127 + 6	-123
10 x 15	150
127 x 2	254
-1 x -1	1
-127 x -127	16129
10 / 5	2
67 / 32	2 R 3

DISCUSSION OF TEST RESULTS

LINK TO VIDEO DEMONSTRATION:

https://youtube.com/J18KUbnBY-OPh-YQ?e

VERILOG MODULES

```
1   //Edge detector circuit
2   module EdgeDetect
3   (
4      input in, clock,
5      output out
6   );
7      reg in_delay;
8      always @ (posedge clock)
9         in_delay <= in;
10     assign out = in & ~in_delay;
11  endmodule
12
```

```
1   //N-bit, two to one MUX
2   module MUX
3   (
4      input [7:0] Rout, LED,      //declare data    input
5      input IUAU,               //declare select input
6      output [7:0] OutIn          //declare output
7   );
8      assign OutIn = IUAU==0 ? LED : Rout; //select input
9   endmodule
10
```

```
//sign to Seven Segment Decoder
module sign2seven (
    input [3:0] BIN,
    input threeLead,
    output reg [0:6] SEV);
        always @ (BIN)
            case ({BIN[3:0]})
                4'b0000: {SEV[0:6]} = 7'b1111111;//0
                4'b0001: if(threeLead) begin {SEV[0:6]} = 7'b1111111; end
                else begin {SEV[0:6]} = 7'b1111110; end//-
            endcase
endmodule
```

```
//Control Unit Finite State Machine
module ControlUnitFSM
(
      input Enter,Clear,
      output reg Reset,LoadALow,LoadAHigh,LoadO,LoadB,LoadR,IUAU
);
      reg [1:0] state,nextstate;
      parameter S0=4'b0000,S1=4'b0001,S2=4'b0010,S3=4'b0011,S4=4'b0100, S5=4'b0101;
      always @ (posedge Enter,negedge Clear)
            if (Clear==1'b0) state <= S0;
            else state <= nextstate;
      always @ (state)
      begin
         case (state)
               S0: begin Reset = 1; LoadALow = 0;  LoadAHigh = 0; LoadO = 0;  LoadB = 0; LoadR = 0; IUAU = 0; nextstate <= S1; end  //Clear
               S1: begin Reset = 1; LoadALow = 1;  LoadAHigh = 0; LoadO = 0;  LoadB = 0; LoadR = 0; IUAU = 0; nextstate <= S2; end  //LoadA Enter
               S2: begin Reset = 0; LoadALow = 0;  LoadAHigh = 1; LoadO = 0;  LoadB = 0; LoadR = 0; IUAU = 0; nextstate <= S3; end  //Load Ahigh Enter
               S3: begin Reset = 0; LoadALow = 0;  LoadAHigh = 0; LoadO = 1;  LoadB = 0; LoadR = 0; IUAU = 0; nextstate <= S4; end  //Load Operation Enter
               S4: begin Reset = 0; LoadALow = 0;  LoadAHigh = 0; LoadO = 0;  LoadB = 1; LoadR = 0; IUAU = 0; nextstate <= S5; end  //Load B Enter
               S5: begin Reset = 0; LoadALow = 0;  LoadAHigh = 0; LoadO = 0;  LoadB = 0; LoadR = 1; IUAU = 1; end  //Load results Enter
            endcase
      end
endmodule

```

```
module hun2seven (
    input [3:0] BIN,
    input signbit,secLead,
    output reg [0:6] SEV);
        always @ (BIN)
            case ({BIN[3:0]})
                4'b0000: if(secLead) begin {SEV[0:6]} = 7'b1111111; end
                else if(signbit && ~(secLead)) begin {SEV[0:6]} = 7'b1111110; end//- 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
endmodule
```

```
//TEMITAYO ADEROUNMU - 1001568524 - MID-TERM PROJECT - EIGHT-BIT, TWO-FUNCTION CALCULATOR
module CALCFUNC
(
    input Enter, Clock, //Pll input clock of 50MHz
    input Clear, //A7, B8,SW0
    input [3:0] row,
    output [3:0] col,
    output [0:6] HEX5,HEX4,HEX3,HEX2,HEX1,HEX0,
    output DONE, DONEM, OVRF //ON LEDR9
);
    wire [3:0] value;
    wire valid,enter_out,Reset,LoadALow, LoadAHigh, LoadO,LoadB,LoadR,IUAU;
    reg AddSub, Mult, Div, Start;
    wire [7:0] Rout, OutIn, LED,TwoComp;
    assign TwoComp = LED;
    //assign OutIn = (IUAU)?Rout:LED;


    EdgeDetect EdgeDetect_inst
    (
        .in(Enter) , // input  in_sig
        .clock(Clock) , // input  clock_sig
        .out(enter_out) // output  out_sig
    );

    ControlUnitFSM ControlUnitFSM_inst
    (
        .Enter(enter_out) ,  // input  Enter_sig
        .Clear(Clear) ,   // input  Clear_sig
        .Reset(Reset) ,   // output  Reset_sig
        .LoadALow(LoadALow) ,   // input  LoadALow_sig
        .LoadAHigh(LoadAHigh) , // input  LoadAHigh_sig
        .LoadO(LoadO) ,
        .LoadB(LoadB) ,   // output  LoadB_sig
        .LoadR(LoadR) ,   // output  LoadR_sig
        .IUAU(IUAU)    // output  IUAU_sig
    );


    CALCIU CALCIU_inst
    (
        .clock(Clock) ,   // input  clock_sig
        .reset(Reset) ,   // input  reset_sig
        .row(row) , // input [3:0] row_sig
        .col(col) , // output [3:0] col_sig
        .value(value) ,
        .valid(valid) ,   // output  valid_sig
        .HEX5(HEX5) ,  // output [0:6] HEX5_sig
        .HEX4(HEX4) ,  // output [0:6] HEX4_sig
        .LED(LED)    // output [7:0] LED_sig
    );


    AU AU_inst
    (
        .IN(TwoComp) , // input [7:0] IN_sig
        .CLOCK(Clock) ,   // input  CLOCK_sig
        .value(value) ,   // input  value_sig
        .LoadALow(LoadALow) ,   // input  LoadALow_sig
        .LoadAHigh(LoadAHigh) , // input  LoadAHigh_sig
        .LoadB(LoadB) ,   // input  LoadB_sig
        .LoadR(LoadR) ,   // input  LoadR_sig
        .Reset(Reset) ,   // input  Reset_sig
        .Clear(Clear) ,   // input  Clear_sig
        .Rout(Rout) , // output [15:0] Rout_sig
        .OVRF(OVRF) ,  // output  OVRF_sig
        .DONE(DONE) ,  // output  DONE_sig
        .DONEM(DONEM)  // output  DONEM_sig
    );
```

```
            .S(IUAU) ,   // input  IUAU_sig
            .Y(OutIn)    // output [7:0] OutIn_sig
        );


        //assign OutIn = (IUAU)?LED:Rout;

        CALCOU CALCOU_inst
        (
            .A(OutIn) , // input [7:0] A_sig
            .HEX3(HEX3) ,  // output [0:6] HEX3_sig
            .HEX2(HEX2) ,  // output [0:6] HEX2_sig
            .HEX1(HEX1) ,  // output [0:6] HEX1_sig
            .HEX0(HEX0)    // output [0:6] HEX0_sig
        );

    endmodule
```

```
module ones2seven (
    input [3:0] BIN,
    input secLead, threeLead,
    output reg [0:6] SEV);
        always @ (BIN)
            case ({BIN[3:0]})
                4'b0000: if(secLead && threeLead) begin {SEV[0:6]} = 7'b1111111; end
                else begin {SEV[0:6]} = 7'b0000001;end//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
endmodule
```

```
module ten2seven (
    input [3:0] BIN,
    input threeLead,signbit,
    output reg [0:6] SEV);
        always @ (BIN)
            case ({BIN[3:0]})
                4'b0000: if(threeLead && signbit)begin {SEV[0:6]} = 7'b1111110; end
                else if (threeLead && ~(signbit)) begin {SEV[0:6]} = 7'b1111111; end
                else begin {SEV[0:6]} = 7'b0000001; end//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
endmodule
```

```
module AU(
    input [7:0] IN,
    input CLOCK,
    //input START,
    input value, LoadALow,LoadAHigh,LoadB,LoadR,Reset,Clear,
    //input OUT,
    //output reg [0:6] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0,
    output [15:0] Rout,
    output OVRF, //ON LEDR9
    output DONE, DONEM
);

    reg   AddSub, Mult, Div, Start;
    wire [15:0] GCout, DIVout, MULTout, SNDout;


    always @(value)
        begin
            case (value)
                  4'b1010: begin AddSub = 0; end
                  4'b1011: begin AddSub = 1; end
                  4'b1100: begin Mult = 1;   end
                  4'b1101: begin Div = 1;   end
                  4'b1111: begin Start = 1;  end
            endcase
        end


        GCLA GCLAinst
        (
            .AB(IN) ,    // input [7:0] AB_sig
            .AddSub(AddSub) ,
            .InA(LoadALow) ,  // input  InA_sig
            .InB(LoadB) ,  // input  InB_sig
            .OUT(LoadR) ,  // input  OUT_sig
            .CLR(Clear) ,  // input  CLR_sig
            .Rout(GCout) , // output [7:0] Rout_sig
            .OVR(OVRF)  // output  OVR_sig
        );


        DIV ddd
        (
            .inpt(IN) , // input [7:0] inpt_sig
            .CLOCK(CLOCK) ,   // input  CLOCK_sig
            .START(Start) ,   // input  START_sig
            .LoadALow(LoadALow) ,   // input  LoadALow_sig
            .LoadAHigh(LoadAHigh) , // input  LoadAHigh_sig
            .LoadB(LoadB) ,   // input  LoadB_sig
            .Reset(Reset) ,   // input  Reset_sig
            .OUT(DIVout) , // output [15:0] OUT_sig
            .DONE(DONE)    // output  DONE_sig
        );


        Multiplier8x8 Multiplier8x8_inst
        (
            .ClockIn(CLOCK) , // input  ClockIn_sig
            .Reset(Reset) ,   // input  Reset_sig
            .MPDin(LoadALow) ,   // input  MPDin_sig
            .MPRin(LoadB) ,   // input  MPRin_sig
            .go(Start) ,   // input  go_sig
            .X(IN) , // input [7:0] X_sig
            .Done(DONEM) , // output  Done_sig
            .ProdOut(MULTout)    // output [15:0] ProdOut_sig
        );

        assign SNDout = (DONE == 1 && Div == 1) ? DIVout : GCout;
        assign Rout = (DONEM == 1 && Mult == 1) ? MULTout : SNDout;


endmodule
```

```
1   //Multiplier. Verilog behavioral model.
2   module Multiplier8x8 (
3       input ClockIn, Reset, MPGin, MPRin, go, //declare inputs
4       input [7:0] X,
5       // input [7:0] Multiplicand,
6       // input [7:0] Multiplier,
7       // output [15:0] Product, //declare outputs
8
9       output reg Done,
10      output [15:0] ProdOut);
11
12      wire Halt;
13      wire [0:7] out5, out4, out3, out2, out1, out0;
14      wire [7:0] LED;
15      integer i;
16      wire [7:0] Multiplicand, Multiplier;
17      wire [15:0] Product,In_Out;
18      reg [7:0] cycle;
19      reg [7:0] RegQ;
20      //Q register
21      reg [15:0] RegM;
22      //M register
23      reg [16:0] RegA; //A register
24      reg [2:0] Count; //3-bit iteration counter
25      wire C0, Start, Add, Shift, Clock, ClockOut;
26      assign Product = {RegA[7:0],RegQ}; //product = A:Q
27      // assign out5 = 8'b11111111;
28      // assign out4 = 8'b11111111;
29      assign LED = 8'b0;
30      //2-bit counter for #iterations
31      always @(posedge Clock)
32      if (Start == 1) Count <= 3'b00; //clear in Start state
33      else if (Shift == 1) Count <= Count + 1; //increment in Shift state
34      assign C0 = Count[2]&Count[1] & Count[0]; //detect count = 7
35      //Multiplicand register (load only)
36      always @(posedge Clock)
37      if (Start == 1)
38      begin
39         for (i = 7; i <= 15; i = i + 1)
40            RegM[i] <= Multiplicand[7];
41            for (i = 0; i <=6; i = i + 1)
42               RegM[i] <= Multiplicand[i];
43      end
44      //Multiplier register (load, shift)
45      always @(posedge Clock)
46      if (Start == 1) RegQ <= Multiplier; //load in Start state
47      else if (Shift == 1) RegQ <= {RegA[0],RegQ[7:1]}; //shift in Shift state
48      //Accumulator register (clear, load, shift)
49      always @(posedge Clock)
50      if (Start == 1) RegA <= 16'b0; //clear in Start state
51      else if (Add == 1 && Count == 7) RegA <= RegA - RegM;
52      //Add or subtract in Add state
53      else if (Add == 1 && Count != 7) RegA <= RegA + RegM;
54      else if (Shift == 1) RegA <= RegA >>> 1; //shift in Shift state
55      always @(posedge Clock, negedge Reset)
56      if (Reset == 0) Done = 1'b0; else if (Halt == 1) Done = 1'b1;
57      else Done = 1'b0;
58      //Instantiate controller module
59      MultControl8x8 Ctrl (Clock, Reset, RegQ[0], C0, Start, Add, Shift, Halt);
60      //Clock cycle counter
61      always @ (posedge Clock, negedge Reset)
62      if (Reset == 1'b0) cycle <= 1'b0;
63      else if (Done == 1'b0) cycle <= cycle + 1'b1; else cycle <= cycle;
64      //Instantiate start/stop toggle
65      OnOffToggle OnOff
66      (
67         .OnOff(go) , // input OnOff_sig
68         .IN(ClockIn) , // input IN_sig
69         .OUT(ClockOut) // output OUT_sig
70      );
71      //Instantiate 5Hz clock
72      FiveHzClock FiveHz_clock
73      (
74         .clock(ClockOut) , // input clock_sig
75         .reset(Reset) , // input reset_sig
76         .FiveHz(Clock) // output FiveHz_sig
```

```
77        );
78        //Instantiate buffer registers
79        NbitRegisterWclear MPDBuffer
80        (
81           .D(X) , // input [N-1:0] D_sig
82           .Q(Multiplicand), // output [N-1:0] Q_sig
83           .CLK(MPDin) , // input CLK_sig
84           .CLR(Reset) // input CLR_sig
85        );
86        NbitRegisterWclear MPRBuffer
87        (
88           .D(X) , // input [N-1:0] D_sig
89           .Q(Multiplier) , // output [N-1:0] Q_sig
90           .CLK(MPRin) , // input CLK_sig
91           .CLR(Reset) // input CLR_sig
92        );
93        NbitRegisterWclear #(16) OutBuffer
94        (
95           .D(In_Out) , // input [N-1:0] D_sig
96           .Q(ProdOut) , // output [N-1:0] Q_sig
97           .CLK(~MPDin|~MPRin|Done) , // input CLK_sig
98           .CLR(Reset) // input CLR_sig
99        );
100       //Instantiate 2-to-1 multiplexer
101       mux #(16) in_out
102       (
103          .A({Multiplicand, Multiplier}) , // input [N-1:0] A_sig
104          .B(Product) , // input [N-1:0] B_sig
105          .Y(In_Out) , // output [N-1:0] Y_sig
106          .S(Halt) // input S_sig
107       );
108       //Instantiate output decoders
109       binary2seven clock_cycle_high
110       (
111          .BIN(cycle[7:4]) , // input [3:0] BIN_sig
112          .SEV(out5) // output [0:6] SEV_sig
113       );
114       binary2seven clock_cycle_low
115       (
116          .BIN(cycle[3:0]) , // input [3:0] BIN_sig
117          .SEV(out4) // output [0:6] SEV_sig
118       );
119       binary2seven ProdOutHigh1
120       (
121          .BIN(ProdOut[15:12]) , // input [3:0] BIN_sig
122          .SEV(out3) // output [0:6] SEV_sig
123       );
124       binary2seven ProdOutHigh0
125       (
126          .BIN(ProdOut[11:8]) , // input [3:0] BIN_sig
127          .SEV(out2) // output [0:6] SEV_sig
128       );
129       binary2seven ProdOutLow1
130       (
131          .BIN(ProdOut[7:4]) , // input [3:0] BIN_sig
132          .SEV(out1) // output [0:6] SEV_sig
133       );
134       binary2seven ProdOutLow0
135       (
136          .BIN(ProdOut[3:0]) , // input [3:0] BIN_sig
137          .SEV(out0) // output [0:6] SEV_sig
138       );
139    endmodule
140
141
```

```
1
2   module DIV(
3      input [7:0] inpt,
4      input CLOCK,
5      input START,
6      input LoadALow,LoadAHigh,LoadB,Reset,
7      //input OUT,
8      //output reg [0:6] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0,
9      output [15:0] OUT,
10     output DONE
11  );
12
13     logic [7:0] HEX01,HEX23,HEX45;
14     logic [3:0] HEX5IN, HEX4IN, HEX3IN, HEX2IN, HEX1IN, HEX0IN;
15     logic [7:0] countcycles;
16     logic [15:0] Dividendw;
17     logic [7:0] Divisorw;
18     logic [15:0] Dividend;
19     logic [7:0] Divisor;
20     logic [7:0] QuotientF;
21     logic [7:0] Quotient;
22     logic [7:0] RemainderF;
23     logic [7:0] Remainder;
24     logic [2:0] Count;
25     logic neqR;
26     logic [8:0] alu_out;
27     logic alu_cy;
28     logic [8:0] mux_out;
29     logic [8:0] mux_in;
30     logic [8:0] R_out;
31     logic [7:0] Q_out;
32     logic [7:0] D_out;
33     logic Rload;
34     logic Qload;
35     logic Dload;
36     logic Rshift;
37     logic Qshift;
38     logic AddSub;
39     logic Qbit;
40
41
42     //assign neqR=1'b0;
43
44
45     NbitRegisterwclear LOADALow
46     (
47        .D(inpt) ,  // input [7:0] DataIn_sig
48        .Q(Dividendw[7:0]),  // output [7:0] DataOut_sig
49        .CLK(LoadALow) ,  // input  CLK_sig
50        .CLR(Reset)    // input  CLR_sig
51     );
52
53
54     NbitRegisterwclear LOADAHigh
55     (
56        .D(inpt) ,  // input [7:0] DataIn_sig
57        .Q(Dividendw[15:8]) ,   // output [7:0] DataOut_sig
58        .CLK(LoadAHigh) , // input  CLK_sig
59        .CLR(Reset) // input  CLR_sig
60     );
61
62     NbitRegisterwclear LOADB
63     (
64
65        .D(inpt) ,  // input [7:0] DataIn_sig
66        .Q(Divisor),   // output [7:0] DataOut_sig
67        .CLK(LoadB) ,  // input  CLK_sig
68        .CLR(Reset)    // input  CLR_sig
69     );
70
71
72     DividendZTwosComp DividendZTwosComp_inst
73     (
74        .BinarySM(Dividendw) ,  // input [15:0] BinarySM_sig
75        .TwosComp(Dividend)  // output [15:0] TwosComp_sig
76     );
```

```
77      /*
78      BinarySMZTwosComp BinarySMZTwosComp_inst
79      (
80         .BinarySM(Divisorw) ,   // input [7:0] BinarySM_sig
81         .TwosComp(Divisor)   // output [7:0] TwosComp_sig
82      );*/
83
84
85
86      assign negR= (Dividendw[15] ^ Divisor[7]);
87      assign countcycles = {5'b0,Count};
88      assign Remainder = R_out[7:0];
89      assign mux_in = {1'b0, Dividend[15:8]};
90      assign Quotient = Q_out;
91      mux #(9) Mux1 (alu_out, mux_in, mux_out, Qload);
92      shiftreg #(9) Rreg (mux_out, R_out, CLOCK, Rload, Rshift, Q_out[7]);
93      shiftreg #(8) Qreg (Dividend[7:0], Q_out, CLOCK, Qload, Qshift, Qbit);
94      shiftreg #(8) Dreg (Divisor, D_out, CLOCK, Dload, 1'b0, 1'b0);
95      alu #(8) AddSub (R_out, D_out, alu_out, AddSub);
96      dcontrol DivCtrl (CLOCK, START, alu_out[8], AddSub, Dload, Rload, Qload, Rshift, Qshift,
    DONE, Count, Qbit);
97
98      ResultZTwosComp ResultZTwosComp_inst
99      (
100        .negR(negR) ,  // input  negR_sig
101        .BinarySM(Quotient) ,   // input [7:0] BinarySM_sig
102        .TwosComp(QuotientF)    // output [7:0] TwosComp_sig
103     );
104
105  /* ResultZTwosComp RemainderTwosComp
106     (
107        .negR(~negR) , // input  negR_sig
108        .BinarySM(Remainder) ,  // input [7:0] BinarySM_sig
109        .TwosComp(RemainderF)   // output [7:0] TwosComp_sig
110     );*/
111
112     assign HEX5IN = (START == 1) ? Dividendw[15:12] : countcycles[7:4]; //START ==0
113     assign HEX4IN = (START == 1) ? Dividendw[11:8] : countcycles[3:0];
114     assign HEX3IN = (START == 1) ? Dividendw[7:4] : QuotientF[7:4];
115     assign HEX2IN = (START == 1) ? Dividendw[3:0] : QuotientF[3:0];
116     assign HEX1IN = (START == 1) ? Divisor[7:4] : Remainder[7:4];
117     assign HEX0IN = (START == 1) ? Divisor[3:0] : Remainder[3:0];
118
119     assign HEX01 = {HEX0IN,HEX1IN};
120     assign HEX23 = {HEX2IN,HEX3IN};
121     assign HEX45 = {HEX4IN,HEX5IN};
122
123     assign OUT = {HEX01,HEX23};
124
```

CODE THAT DEMONSTRATED ERROR ON THE HEX BOARD FOR ADDITIONAL FEATURES TO THE PROJECT.

```
1   //output Er for invalid
2   module validOut (
3       input [3:0] BIN,
4       input valid,
5       output reg [0:6] SEV);
6           always @ (BIN)
7               case ({BIN[3:0]})
8                   4'b1110: if(~valid) begin {SEV[0:6]} = 7'b0110000; end
9                   else begin {SEV[0:6]} = 7'b1111111; end//E
10                  4'b1111: if(~valid) begin {SEV[0:6]} = 7'b1111010; end
11                  else begin {SEV[0:6]} = 7'b1111111; end//r
12              endcase
13  endmodule
```

```
1   //Temitayo Aderounmu - 1001568524 - Assignment 3 - CSE 4357 - Spring 2023
2   //Calculator Output Unit
3   module CALCOU (
4      input [7:0] A,
5      //input clock, reset,
6      output [0:6] HEX3, HEX2, HEX1, HEX0);
7      wire [15:0] bcdout;
8      wire [3:0] ONES, TENS;
9      wire [1:0] HUNDREDS;
10     wire [7:0] R;
11     wire [3:0] SIGN,HUN;
12     wire lastBit,threeLead, secLead;
13     assign SIGN = {3'b0,lastBit};
14     assign HUN = {2'b0,HUNDREDS};
15
16     TwoCompToSign TwoCompToSign_inst
17     (
18        .A(A) ,  // input [7:0] A_sig
19        .R(R) ,  // output [7:0] R_sig
20        .lastBit(lastBit)    // output  lastBit_sig
21     );
22
23     binary2bcd binary2bcd_inst
24     (
25        .A(R) ,  // input [7:0] A_sig
26        .ONES(ONES) ,   // output [3:0] ONES_sig
27        .TENS(TENS) ,   // output [3:0] TENS_sig
28        .HUNDREDS(HUNDREDS)  // output [1:0] HUNDREDS_sig
29     );
30
31     assign threeLead = (HUNDREDS==0)?1:0;
32     assign secLead = (TENS==0)?1:0;
33
34     //Instantiate binary to seven segment decoders for displaying inputs and outputs in HEX
35
36     sign2seven sign2seven_inst
37     (
38        .BIN(SIGN) ,    // input [3:0] BIN_sig
39        .threeLead(threeLead) , // input  threeLead_sig
40        .SEV(HEX3)  // output [0:6] SEV_sig
41     );
42
43     hun2seven hun2seven_inst
44     (
45        .BIN(HUN) , // input [3:0] BIN_sig
46        .signbit(lastBit) ,  // input  signbit_sig
47        .secLead(secLead) ,  // input  secLead_sig
48        .SEV(HEX2)  // output [0:6] SEV_sig
49     );
50
51     ten2seven ten2seven_inst
52     (
53        .BIN(TENS) ,    // input [3:0] BIN_sig
54        .threeLead(threeLead) , // input  threeLead_sig
55        .signbit(lastBit) ,  // input  signbit_sig
56        .SEV(HEX1)  // output [0:6] SEV_sig
57     );
58
59     ones2seven ONES1
60     (
61        .BIN(ONES) ,      // input [3:0] BIN_sig
62        .secLead(secLead) ,  // input  secLead_sig
63        .threeLead(threeLead) , // input  threeLead_sig
64        .SEV(HEX0)       // output [0:6] SEV_sig
65     );
66
67
68
69  endmodule
```

PIN ASSIGNMENTS

PART1:

	Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	Ok		Clear	Location	PIN_B8	Yes			
2	Ok		Clock	Location	PIN_P11	Yes			
3	Ok		OVRF	Location	PIN_B11	Yes			
4	Ok		HEX0[0]	Location	PIN_C14	Yes			
5	Ok		HEX0[1]	Location	PIN_E15	Yes			
6	Ok		HEX0[2]	Location	PIN_C15	Yes			
7	Ok		HEX0[3]	Location	PIN_C16	Yes			
8	Ok		HEX0[4]	Location	PIN_E16	Yes			
9	Ok		HEX0[5]	Location	PIN_D17	Yes			
10	Ok		HEX0[6]	Location	PIN_C17	Yes			
11	Ok		HEX1[0]	Location	PIN_C18	Yes			
12	Ok		HEX1[1]	Location	PIN_D18	Yes			
13	Ok		HEX1[2]	Location	PIN_E18	Yes			
14	Ok		HEX1[3]	Location	PIN_B16	Yes			
15	Ok		HEX1[4]	Location	PIN_A17	Yes			
16	Ok		HEX1[5]	Location	PIN_A18	Yes			
17	Ok		HEX1[6]	Location	PIN_B17	Yes			
18	Ok		HEX2[0]	Location	PIN_B20	Yes			
19	Ok		HEX2[1]	Location	PIN_A20	Yes			
20	Ok		HEX2[2]	Location	PIN_B19	Yes			
21	Ok		HEX2[3]	Location	PIN_A21	Yes			
22	Ok		HEX2[4]	Location	PIN_B21	Yes			
23	Ok		HEX2[5]	Location	PIN_C22	Yes			
24	Ok		HEX2[6]	Location	PIN_B22	Yes			
25	Ok		HEX3[0]	Location	PIN_F21	Yes			
26	Ok		HEX3[1]	Location	PIN_E22	Yes			
27	Ok		HEX3[2]	Location	PIN_E21	Yes			
28	Ok		HEX3[3]	Location	PIN_C19	Yes			
29	Ok		HEX3[4]	Location	PIN_C20	Yes			
30	Ok		HEX3[5]	Location	PIN_D19	Yes			
31	Ok		HEX3[6]	Location	PIN_E17	Yes			
32	Ok		col[0]	Location	PIN_AA12	Yes			
33	Ok		col[1]	Location	PIN_AA11	Yes			
34	Ok		col[2]	Location	PIN_Y10	Yes			
35	Ok		col[3]	Location	PIN_AB9	Yes			
36	Ok		row[0]	Location	PIN_AB8	Yes			
37	Ok		row[1]	Location	PIN_AB7	Yes			
38	Ok		row[2]	Location	PIN_AB6	Yes			
39	Ok		row[3]	Location	PIN_AB5	Yes			

	Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
40	Ok		row[3]	Location	PIN_AB5	Yes			
41	Ok		AddSub	Location	PIN_C10	Yes			
42		<<new>>	<<new>>	<<new>>					