Name: Padmanathan Kannan **Student ID:** 1002011190

Date Submitted: 05-02-2023

CSE 5357

Advanced Digital Logic Design

Spring Semester 2023

Assignment 8 Term Project – Eight-Bit, Four-Function Calculator

Requirements Summary:

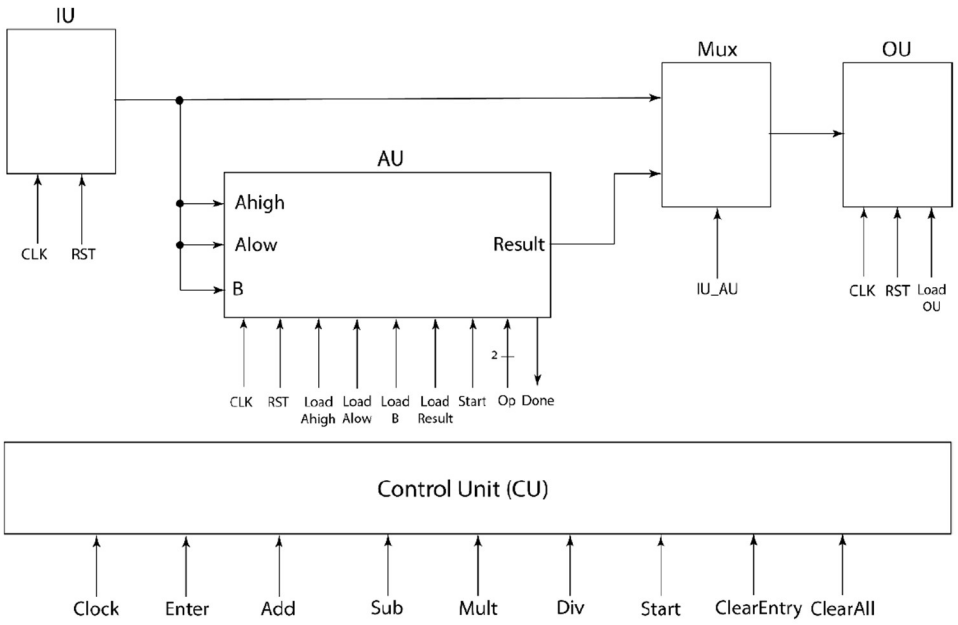
The goal is to create a calculator with four functions by utilizing the Arithmetic Unit, Output Unit, Input Unit, and Control Unit. Users must enter inputs using a 4 x 4 KeyPad, where non-negative numbers should be entered in decimal sign-magnitude, and blank should be used for the sign. Leading zeros are optional. Meanwhile, negative numbers must be in decimal sign-magnitude, and * should be used for the sign followed by the magnitude with leading zeros.

Inputs consist of up to three magnitude digits and a sign. Pressing key 0 clears the calculator. To capture operand A, press the corresponding key on the keypad. Enter the operation and operand by pressing A, B, C, or D. To capture operand B, repeat the process and press the # key to start the operation. Clear Entry function is accessible via Key1.

Upon entry, operands should be displayed on the DE10 seven-segment displays in decimal sign-magnitude format (HEX5, HEX4, HEX3, HEX2, HEX1, and HEX0). Once calculations are complete, results should also be displayed in decimal sign-magnitude on the same displays. Non-negative results should appear without a sign and leading zeros, while negative results must have a minus sign (-) followed by the magnitude with or without leading zeros. LEDR9 should light up to indicate overflow in operations. A finite state machine generates the control signals.

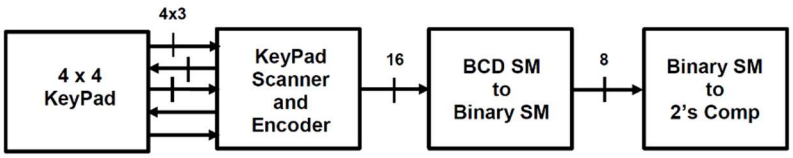
Organization Diagram:

1) **Top Level:**

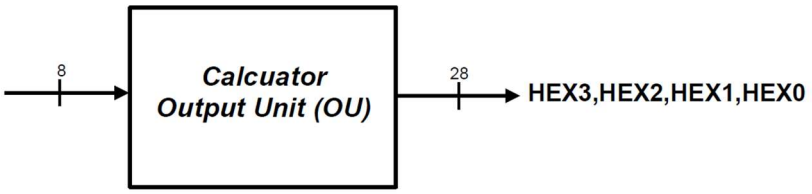


---

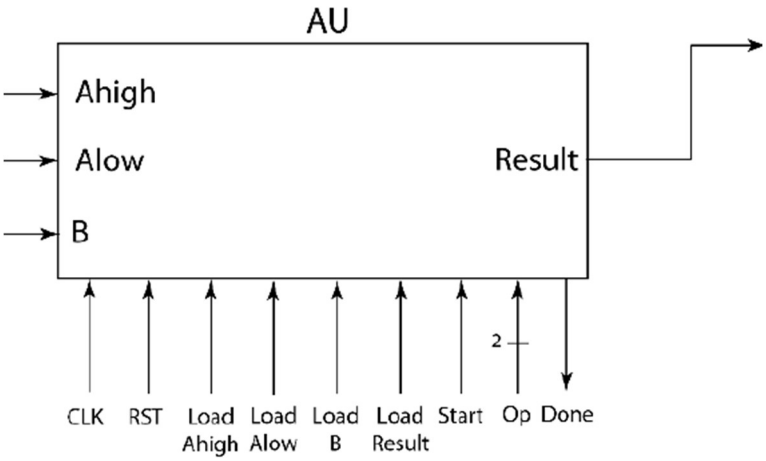
2) **Input Unit:**



3) **Output Unit:**

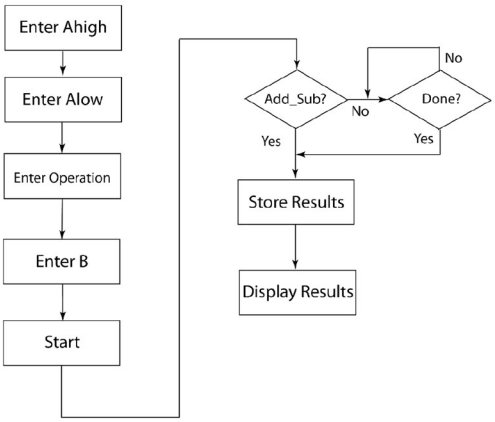


4) **Arithmetic Unit:**



Control Unit i/o and Diagrams:

1) **Flow Diagram:**



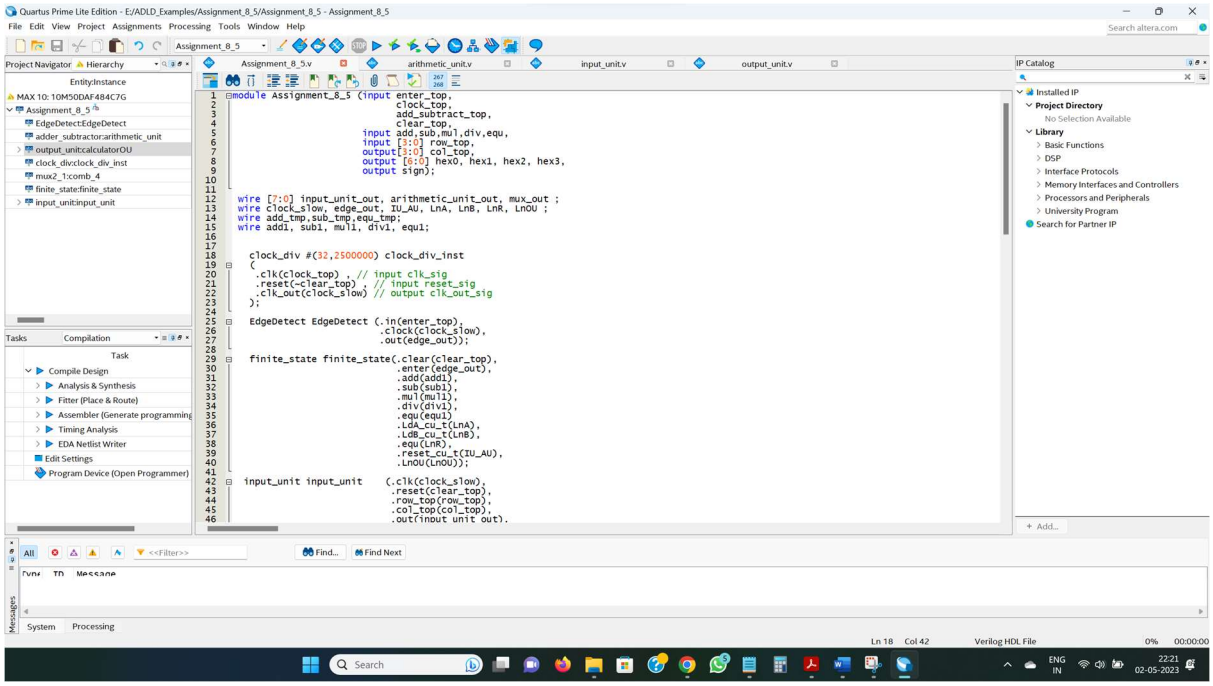
2) **i/o:**

The i/o of finite state machine are **input** clock,clearall,clearentery,add,sub,mult,div,Done,enter, **output** LoadA,LoadB,Loadresult,LoadOU,IUAU,start.

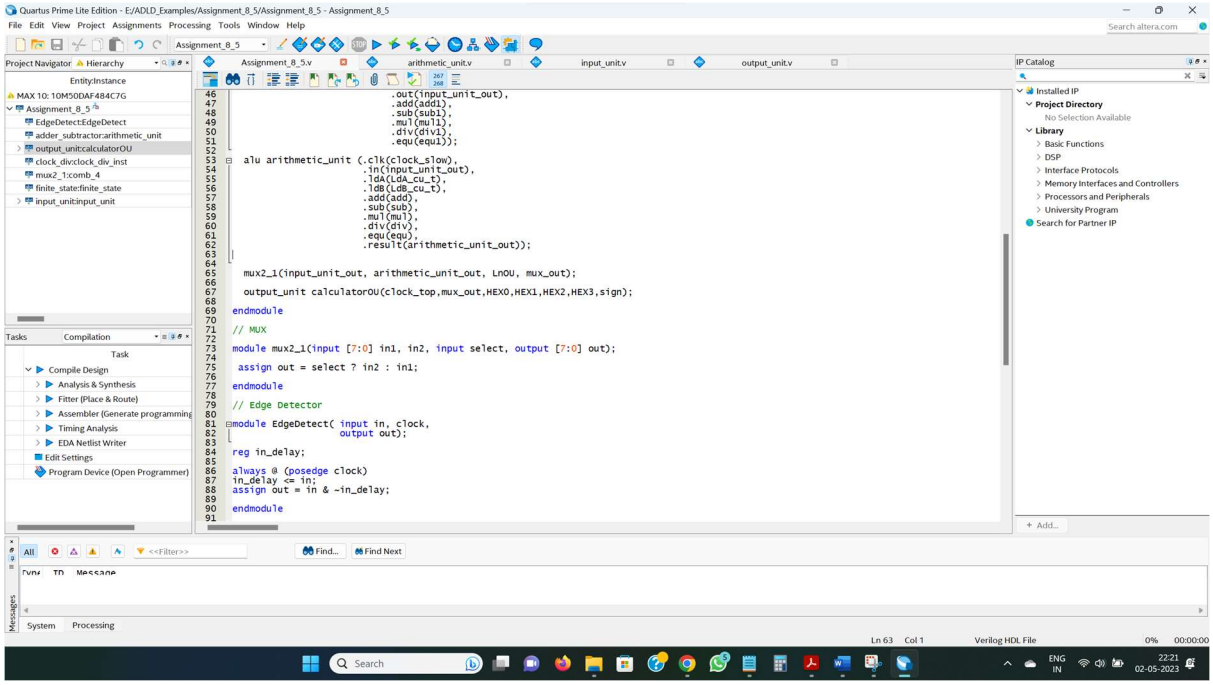
The i/o of top module are **input** enter_top, clock_top, add_subtract_top,clear_top, **input** [3:0] row_top, **output**[3:0] col_top,**output** [6:0] hex0, hex1, hex2, hex3, **output** sign.

Verilog Code:

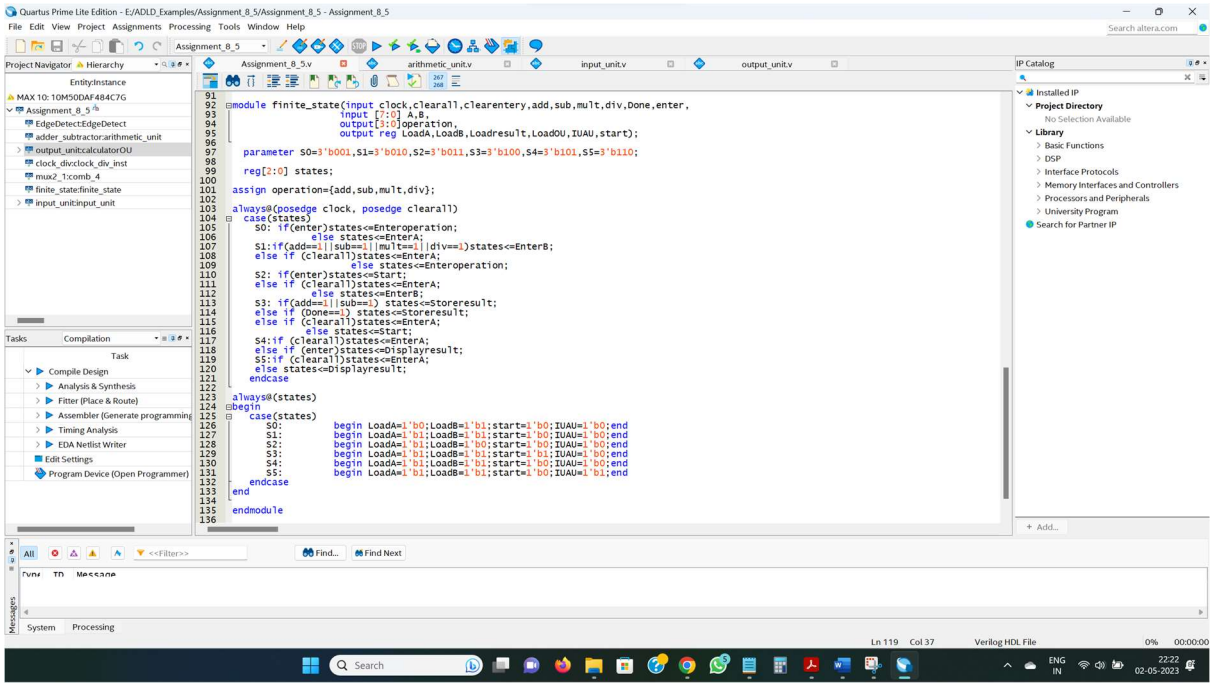
Top module and Control Unit:



```
module Assignment_8_5 (input enter_top,
                       clock_top,
                       add_subtract_top,
                       clear_top,
                input add,sub,mul,div,equ,
                input [3:0] row_top,
                output[3:0] col_top,
                output [6:0] hex0, hex1, hex2, hex3,
                output sign);

wire [7:0] input_unit_out, arithmetic_unit_out, mux_out ;
wire clock_slow, edge_out, IU_AU, LnA, LnB, LnR, LnOU ;
wire add_tmp,sub_tmp,equ_tmp;
wire add1, sub1, mul1, div1, equ1;


  clock_div #(32,2500000) clock_div_inst
  (
   .clk(clock_top) , // input clk_sig
   .reset(~clear_top) , // input reset_sig
   .clk_out(clock_slow) // output clk_out_sig
  );

  EdgeDetect EdgeDetect (.in(enter_top),
                         .clock(clock_slow),
                         .out(edge_out));

  finite_state finite_state(.clear(clear_top),
                            .enter(edge_out),
                            .add(add1),
                            .sub(sub1),
                            .mul(mul1),
                            .div(div1),
                            .equ(equ1)
                            .LdA_cu_t(LnA),
                            .LdB_cu_t(LnB),
                            .equ(LnR),
                            .reset_cu_t(IU_AU),
                            .LnOU(LnOU));

 input_unit input_unit    (.clk(clock_slow),
                           .reset(clear_top),
                           .row_top(row_top),
                           .col_top(col_top),
                           .out(input_unit_out),
```

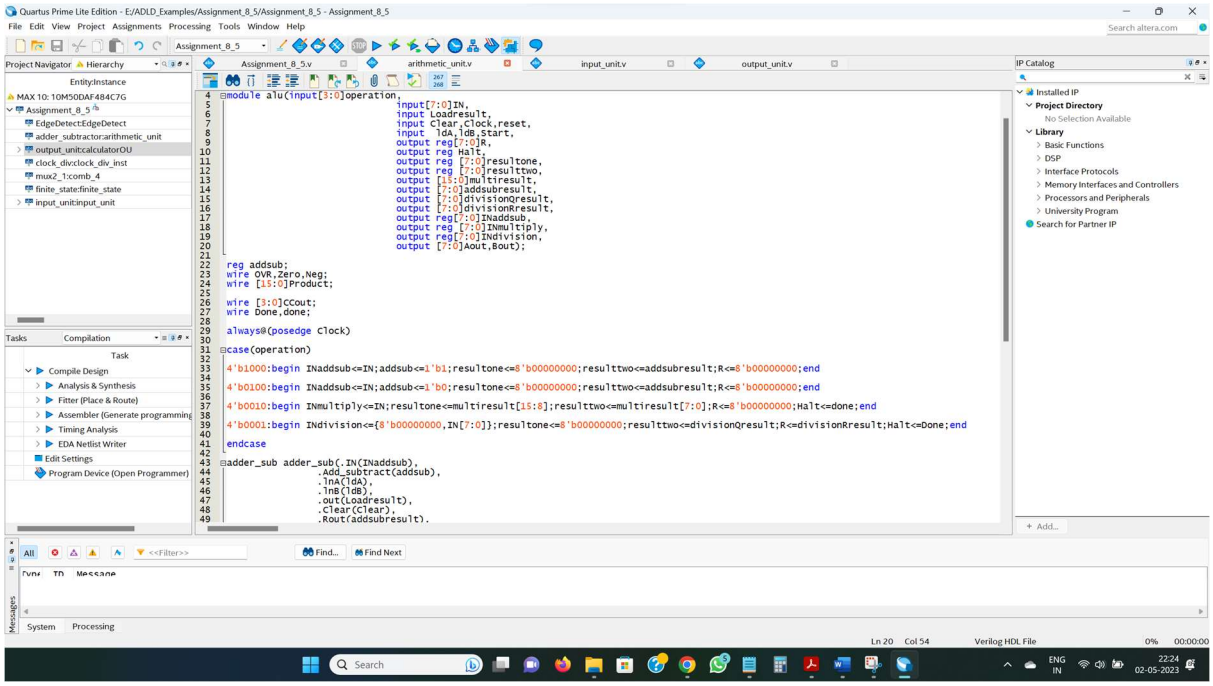


```
                           .out(input_unit_out),
                           .add(add1),
                           .sub(sub1),
                           .mul(mul1),
                           .div(div1),
                           .equ(equ1));

 alu arithmetic_unit (.clk(clock_slow),
                      .in(input_unit_out),
                      .ldA(LdA_cu_t),
                      .ldB(LdB_cu_t),
                      .add(add),
                      .sub(sub),
                      .mul(mul),
                      .div(div),
                      .equ(equ),
                      .result(arithmetic_unit_out));


  mux2_1(input_unit_out, arithmetic_unit_out, LnOU, mux_out);

  output_unit calculatorOU(clock_top,mux_out,HEX0,HEX1,HEX2,HEX3,sign);

endmodule

// MUX

module mux2_1(input [7:0] in1, in2, input select, output [7:0] out);

 assign out = select ? in2 : in1;

endmodule

// Edge Detector

module EdgeDetect( input in, clock,
                   output out);

reg in_delay;

always @ (posedge clock)
in_delay <= in;
assign out = in & ~in_delay;

endmodule
```

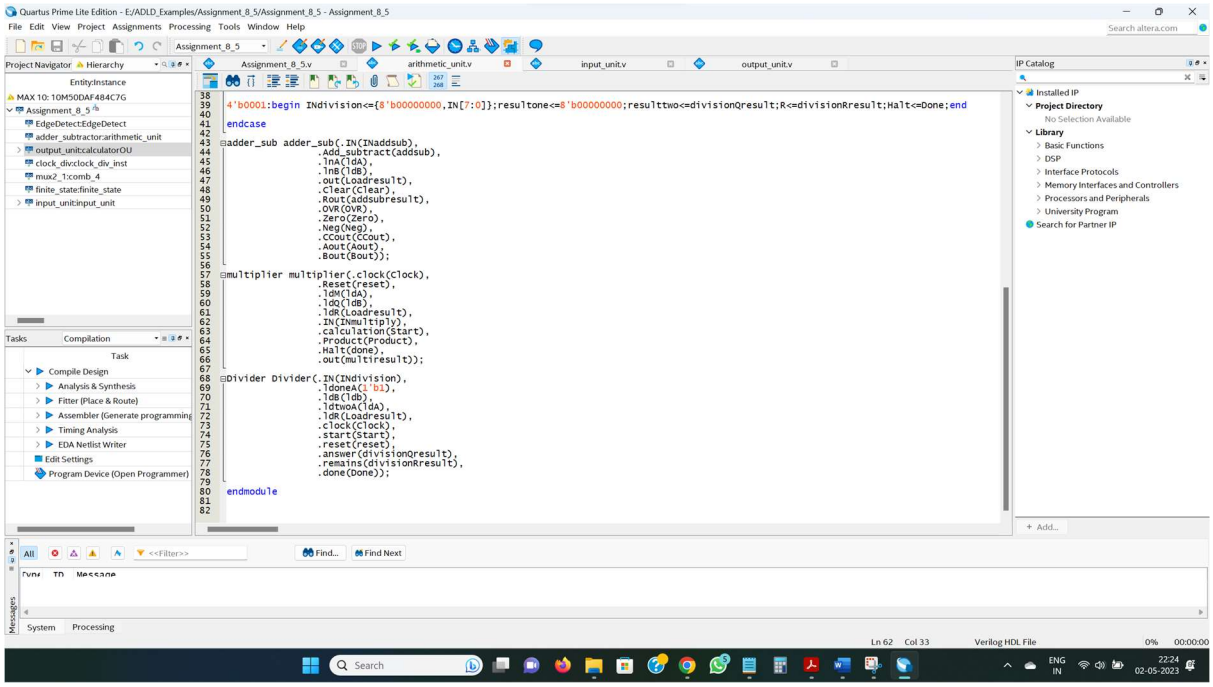


```
module finite_state(input clock,clearall,clearentery,add,sub,mult,div,Done,enter,
                    input [7:0] A,B,
                    output[3:0]operation,
                    output reg LoadA,LoadB,Loadresult,LoadOU,IUAU,start);

  parameter S0=3'b001,S1=3'b010,S2=3'b011,S3=3'b100,S4=3'b101,S5=3'b110;

  reg[2:0] states;

assign operation={add,sub,mult,div};

always@(posedge clock, posedge clearall)
  case(states)
    S0: if(enter)states<=Enteroperation;
              else states<=EnterA;
    S1:if(add==1||sub==1||mult==1||div==1)states<=EnterB;
    else if (clearall)states<=EnterA;
                      else states<=Enteroperation;
    S2: if(enter)states<=Start;
    else if (clearall)states<=EnterA;
          else states<=EnterB;
    S3: if(add==1||sub==1) states<=Storeresult;
    else if (Done==1) states<=Storeresult;
    else if (clearall)states<=EnterA;
          else states<=Start;
    S4:if (clearall)states<=EnterA;
    else if (enter)states<=Displayresult;
    S5:if (clearall)states<=EnterA;
    else states<=Displayresult;
  endcase

always@(states)
begin
   case(states)
      S0:          begin LoadA=1'b0;LoadB=1'b1;start=1'b0;IUAU=1'b0;end
      S1:          begin LoadA=1'b1;LoadB=1'b1;start=1'b0;IUAU=1'b0;end
      S2:          begin LoadA=1'b1;LoadB=1'b0;start=1'b0;IUAU=1'b0;end
      S3:          begin LoadA=1'b1;LoadB=1'b1;start=1'b1;IUAU=1'b0;end
      S4:          begin LoadA=1'b1;LoadB=1'b1;start=1'b0;IUAU=1'b0;end
      S5:          begin LoadA=1'b1;LoadB=1'b0;start=1'b0;IUAU=1'b1;end
   endcase
end

endmodule
```

4 Function Arithmetic Unit:

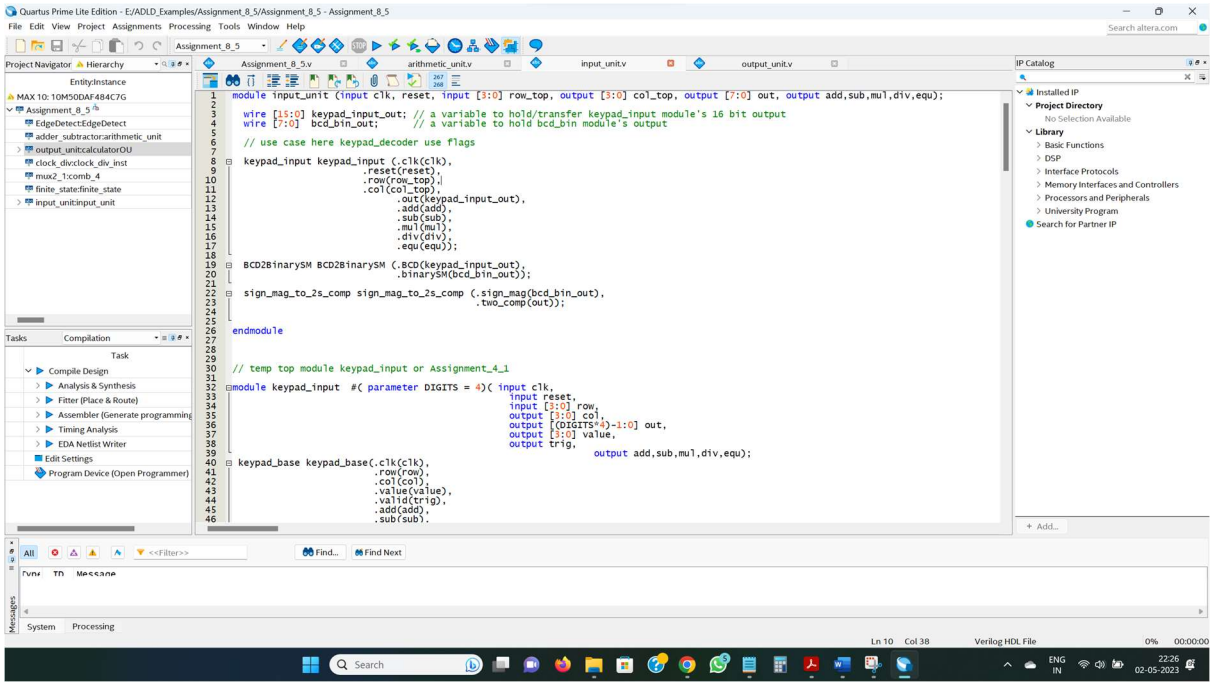


```
module alu(input[3:0]operation,
                        input[7:0]IN,
                        input Loadresult,
                        input Clear,Clock,reset,
                        input  ldA,ldB,Start,
                        output reg[7:0]R,
                        output reg Halt,
                        output reg [7:0]resultone,
                        output reg [7:0]resulttwo,
                        output [15:0]multiresult,
                        output [7:0]addsubresult,
                        output [7:0]divisionQresult,
                        output [7:0]divisionRresult,
                        output reg[7:0]INaddsub,
                        output reg [7:0]INmultiply,
                        output reg[7:0]INdivision,
                        output [7:0]Aout,Bout);

reg addsub;
wire OVR,Zero,Neg;
wire [15:0]Product;

wire [3:0]CCout;
wire Done,done;

always@(posedge Clock)

case(operation)
4'b1000:begin INaddsub<=IN;addsub<=1'b1;resultone<=8'b00000000;resulttwo<=addsubresult;R<=8'b00000000;end

4'b0100:begin INaddsub<=IN;addsub<=1'b0;resultone<=8'b00000000;resulttwo<=addsubresult;R<=8'b00000000;end

4'b0010:begin INmultiply<=IN;resultone<=multiresult[15:8];resulttwo<=multiresult[7:0];R<=8'b00000000;Halt<=done;end

4'b0001:begin INdivision<={8'b00000000,IN[7:0]};resultone<=8'b00000000;resulttwo<=divisionQresult;R<=divisionRresult;Halt<=Done;end

endcase

adder_sub adder_sub(.IN(INaddsub),
                    .Add_subtract(addsub),
                    .lnA(ldA),
                    .lnB(ldB),
                    .out(Loadresult),
                    .Clear(Clear),
                    .Rout(addsubresult).
```

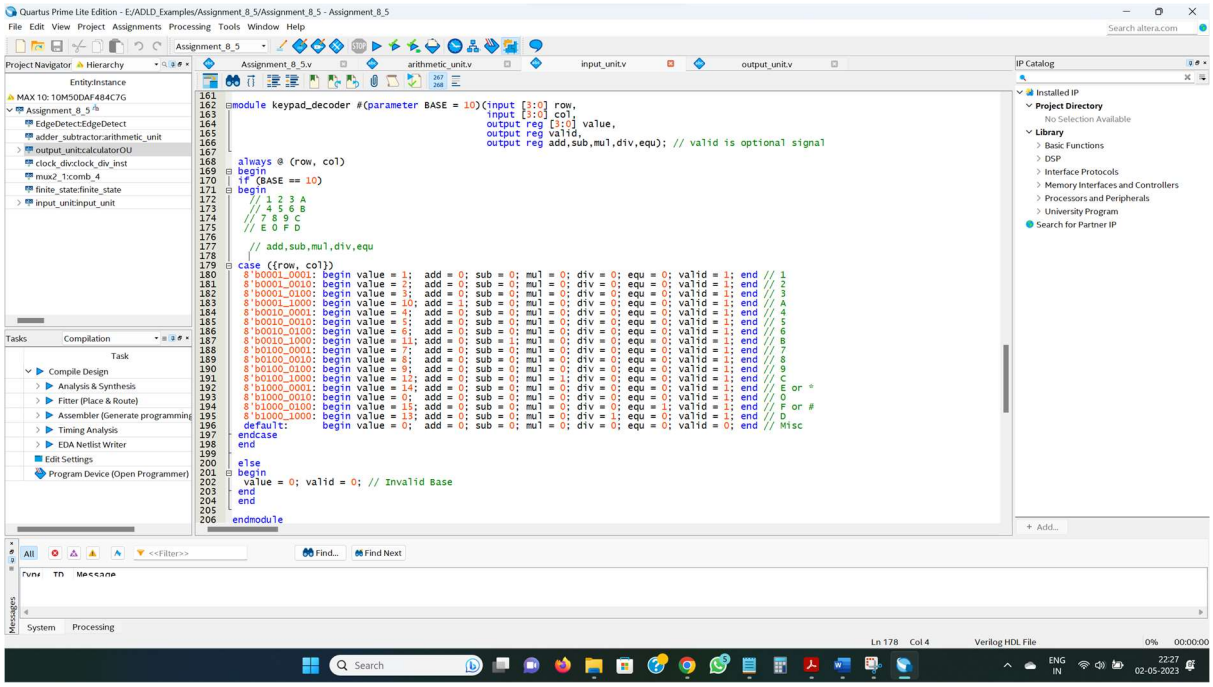


```
4'b0001:begin INdivision<={8'b00000000,IN[7:0]};resultone<=8'b00000000;resulttwo<=divisionQresult;R<=divisionRresult;Halt<=Done;end

endcase

adder_sub adder_sub(.IN(INaddsub),
                    .Add_subtract(addsub),
                    .InA(IdA),
                    .InB(IdB),
                    .out(Loadresult),
                    .Clear(Clear),
                    .Rout(addsubresult),
                    .OVR(OVR),
                    .Zero(Zero),
                    .Neg(Neg),
                    .CCout(CCout),
                    .Aout(Aout),
                    .Bout(Bout));

multiplier multiplier(.clock(Clock),
                      .Reset(reset),
                      .ldM(ldA),
                      .ldQ(ldB),
                      .ldR(Loadresult),
                      .IN(INmultiply),
                      .calculation(Start),
                      .Product(Product),
                      .Halt(done),
                      .out(multiresult));

Divider Divider(.IN(INdivision),
                .ldoneA(1'b1),
                .ldB(ldB),
                .ldtwoA(ldA),
                .ldR(Loadresult),
                .clock(Clock),
                .start(Start),
                .reset(reset),
                .answer(divisionQresult),
                .remains(divisionRresult),
                .done(Done));

endmodule
```

Input Unit (Added the Add,Sub,Mul,Div logic to key A,B,C,D):

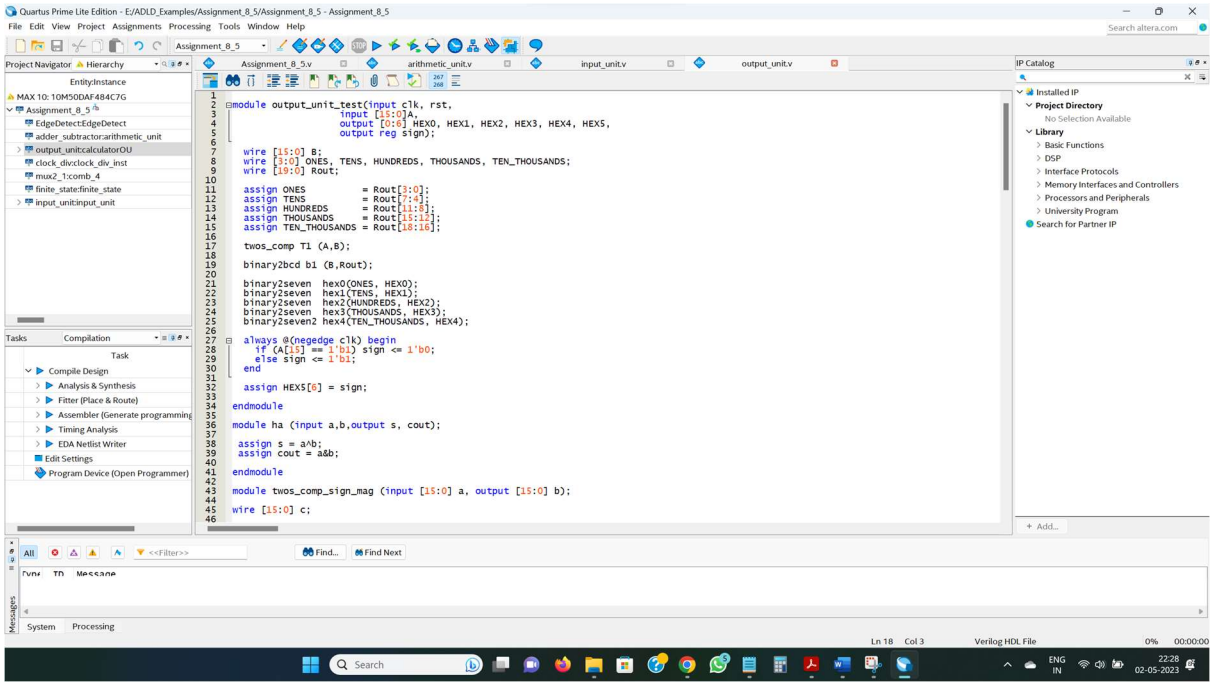


```
module input_unit (input clk, reset, input [3:0] row_top, output [3:0] col_top, output [7:0] out, output add,sub,mul,div,equ);

  wire [15:0] keypad_input_out; // a variable to hold/transfer keypad_input module's 16 bit output
  wire [7:0]  bcd_bin_out;      // a variable to hold bcd_bin module's output

  // use case here keypad_decoder use flags

  keypad_input keypad_input (.clk(clk),
                             .reset(reset),
                             .row(row_top),
                             .col(col_top),
                                   .out(keypad_input_out),
                                   .add(add),
                                   .sub(sub),
                                   .mul(mul),
                                   .div(div),
                                   .equ(equ));

  BCD2BinarySM BCD2BinarySM (.BCD(keypad_input_out),
                             .binarySM(bcd_bin_out));

  sign_mag_to_2s_comp sign_mag_to_2s_comp (.sign_mag(bcd_bin_out),
                                           .two_comp(out));

endmodule


// temp top module keypad_input or Assignment_4_1

module keypad_input  #( parameter DIGITS = 4)( input clk,
                                               input reset,
                                               input [3:0] row,
                                               output [3:0] col,
                                               output [(DIGITS*4)-1:0] out,
                                               output [3:0] value,
                                               output trig,
                                                      output add,sub,mul,div,equ);

keypad_base keypad_base(.clk(clk),
                        .row(row),
                        .col(col),
                        .value(value),
                        .valid(trig),
                        .add(add),
                        .sub(sub),
```



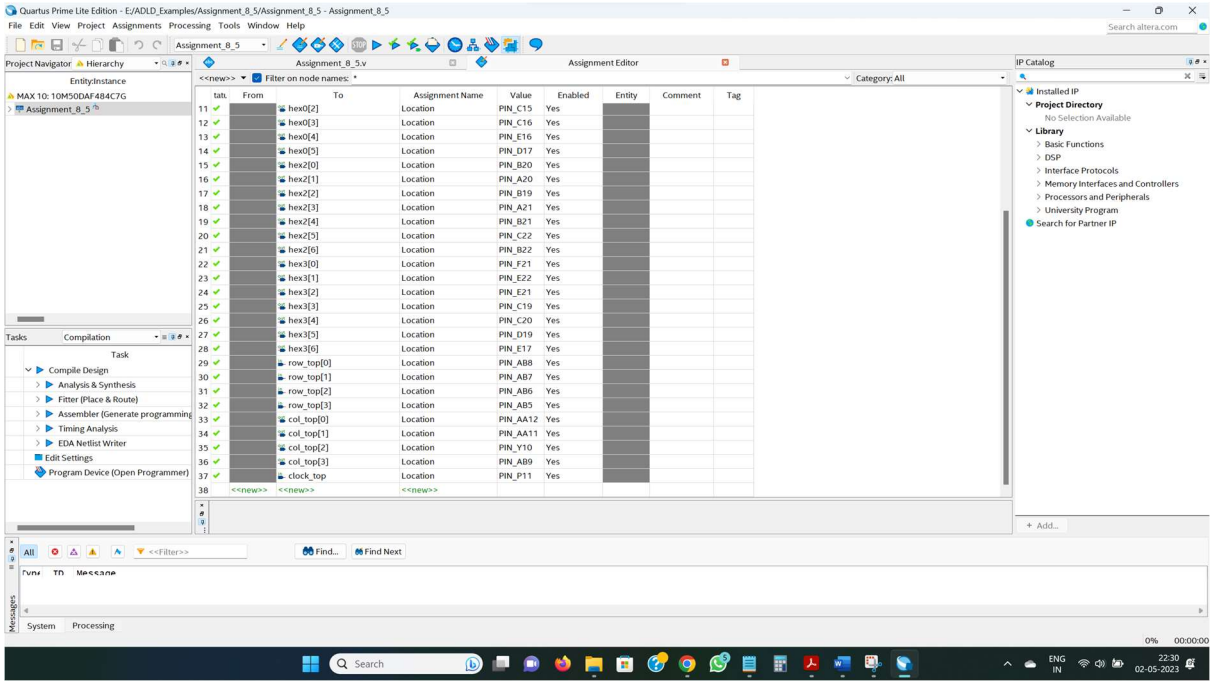
```
161
162 module keypad_decoder #(parameter BASE = 10)(input [3:0] row,
163                                                input [3:0] col,
164                                                output reg [3:0] value,
165                                                output reg valid,
166                                                output reg add,sub,mul,div,equ); // valid is optional signal
167
168  always @ (row, col)
169  begin
170  if (BASE == 10)
171  begin
172     // 1 2 3 A
173     // 4 5 6 B
174    // 7 8 9 C
175    // E 0 F D
176
177     // add,sub,mul,div,equ
178
179  case ({row, col})
180    8'b0001_0001: begin value = 1;  add = 0; sub = 0; mul = 0; div = 0; equ = 0; valid = 1; end // 1
181    8'b0001_0010: begin value = 2;  add = 0; sub = 0; mul = 0; div = 0; equ = 0; valid = 1; end // 2
182    8'b0001_0100: begin value = 3;  add = 0; sub = 0; mul = 0; div = 0; equ = 0; valid = 1; end // 3
183    8'b0001_1000: begin value = 10; add = 1; sub = 0; mul = 0; div = 0; equ = 0; valid = 1; end // A
184    8'b0010_0001: begin value = 4;  add = 0; sub = 0; mul = 0; div = 0; equ = 0; valid = 1; end // 4
185    8'b0010_0010: begin value = 5;  add = 0; sub = 0; mul = 0; div = 0; equ = 0; valid = 1; end // 5
186    8'b0010_0100: begin value = 6;  add = 0; sub = 0; mul = 0; div = 0; equ = 0; valid = 1; end // 6
187    8'b0010_1000: begin value = 11; add = 0; sub = 1; mul = 0; div = 0; equ = 0; valid = 1; end // B
188    8'b0100_0001: begin value = 7;  add = 0; sub = 0; mul = 0; div = 0; equ = 0; valid = 1; end // 7
189    8'b0100_0010: begin value = 8;  add = 0; sub = 0; mul = 0; div = 0; equ = 0; valid = 1; end // 8
190    8'b0100_0100: begin value = 9;  add = 0; sub = 0; mul = 0; div = 0; equ = 0; valid = 1; end // 9
191    8'b0100_1000: begin value = 12; add = 0; sub = 0; mul = 1; div = 0; equ = 0; valid = 1; end // C
192    8'b1000_0001: begin value = 14; add = 0; sub = 0; mul = 0; div = 0; equ = 0; valid = 1; end // E or *
193    8'b1000_0010: begin value = 0;  add = 0; sub = 0; mul = 0; div = 0; equ = 0; valid = 1; end // 0
194    8'b1000_0100: begin value = 15; add = 0; sub = 0; mul = 0; div = 0; equ = 1; valid = 1; end // F or #
195    8'b1000_1000: begin value = 13; add = 0; sub = 0; mul = 0; div = 1; equ = 0; valid = 1; end // D
196    default:      begin value = 0;  add = 0; sub = 0; mul = 0; div = 0; equ = 0; valid = 0; end // Misc
197  endcase
198  end
199
200  else
201  begin
202    value = 0; valid = 0; // Invalid Base
203  end
204  end
205
206  endmodule
```

Output Unit (Updated to include 16 bit inputs and outputs):



```
1
2  module output_unit_test(input clk, rst,
3                          input [15:0]A,
4                          output [0:6] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,
5                          output reg sign);
6
7    wire [15:0] B;
8    wire [3:0] ONES, TENS, HUNDREDS, THOUSANDS, TEN_THOUSANDS;
9    wire [19:0] Rout;
10
11   assign ONES          = Rout[3:0];
12   assign TENS          = Rout[7:4];
13   assign HUNDREDS      = Rout[11:8];
14   assign THOUSANDS     = Rout[15:12];
15   assign TEN_THOUSANDS = Rout[18:16];
16
17   twos_comp T1 (A,B);
18
19   binary2bcd b1 (B,Rout);
20
21   binary2seven  hex0(ONES, HEX0);
22   binary2seven  hex1(TENS, HEX1);
23   binary2seven  hex2(HUNDREDS, HEX2);
24   binary2seven  hex3(THOUSANDS, HEX3);
25   binary2seven2 hex4(TEN_THOUSANDS, HEX4);
26
27   always @(negedge clk) begin
28     if (A[15] == 1'b1) sign <= 1'b0;
29     else sign <= 1'b1;
30   end
31
32   assign HEX5[6] = sign;
33
34 endmodule
35 module ha (input a,b,output s, cout);
36
37
38  assign s = a^b;
39  assign cout = a&b;
40
41 endmodule
42
43 module twos_comp_sign_mag (input [15:0] a, output [15:0] b);
44
45 wire [15:0] c;
46
```

Pin i/o Assignments:



Test Results Video Link:

https://drive.google.com/file/d/1zQWUFONmdfbfuaPhDNyrYuMJfY7/view?usp=sharing45671g

Note: If the above Google Drive Link says "Video is still processing. Try again later.", kindly download the video to watch it.