Course Name:	**Digital Design Using Verilog HDL**	**Semester:**	**IV**
Date of Performance:	**24 /03 / 2025**	**DIV/Batch No:**	**A1**
Student Name:	**Aaditya Chavan**	**Roll No:**	**16014023001**

Miniproject
Title: 4 bit ALU with LCD display

Aim and Objective of the Experiment:

- Develop a 4 bit alu with all the standard ALU functions such as logical instructions and arithmetic instructions
- Display the output on onboard leds for debugging and verification
- Display the final output on onboard LCD

COs to be achieved:

CO1: Understand the language constructs and programming fundamentals of Verilog HDL
CO2. Choose the suitable abstraction level for a particular digital design
CO3. Construct Combinational and sequential circuits in different modeling styles using Verilog
CO4. Analyze and Verify the functionality of digital circuits/systems using test benches

Theory:

The 4-bit ALU (Arithmetic Logic Unit) project provided an in-depth exploration into digital design using Verilog HDL, focusing on implementing fundamental arithmetic and logical operations in hardware.

The objective was to design an ALU capable of performing tasks such as addition, subtraction, bitwise operations, and division/modulus, and to visualize the results on an LCD display module. Through this exercise, I gained hands-on experience in modular design, hierarchical coding practices, timing control for peripheral communication, and testbench-based simulation. Additionally, integrating the ALU with the LCD controller deepened my understanding of interfacing hardware modules and managing control signals in a synchronous system.

Arithmetic Operations:

CONTROL	SELECT LINES	OPERATIONS
1	4'b0000	ADDITION
1	4'b0001	SUBTRACTION
1	4'b0010	MULTIPLICATION
1	4'b0011	INCREMENT
1	4'b0100	DECREMENT
1	4'b1000	DIVISION
1	4'b1001	MODULUS

Logical Operations:

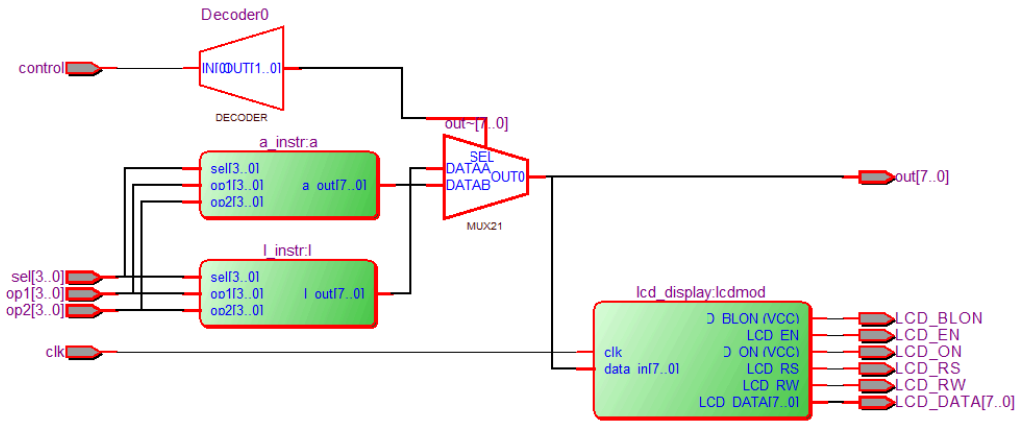
CONTROL	SELECT LINES	OPERATIONS
0	4'b0000	AND
0	4'b0001	OR
0	4'b0010	XOR
0	4'b0011	NOT
0	4'b0100	NAND
0	4'b0101	NOR
0	4'b0110	XNOR
0	4'b0111	SHIFT RIGHT LOGICAL
0	4'b1000	SHIFT LEFT LOGICAL

Problem Statements:

1. Implement a 4 bit ALU

2. implement a lcd display controller to display the output

Code :



top module -

```
module alu4_lcd(
    input wire control,
    input [3:0] sel,
    input [3:0] op1,
    input [3:0] op2,
    input wire clk,
    output reg [7:0] out,
    output wire [7:0] LCD_DATA,
    output wire LCD_RS,
    output wire LCD_RW,
    output wire LCD_EN,
    output wire LCD_ON,
    output wire LCD_BLON
);
```

```
    wire [7:0] out_logic;
    wire [7:0] out_arithmetic;

    l_instr l(
        .sel(sel),
        .op1(op1),
        .op2(op2),
        .l_out(out_logic)
    );

    a_instr a(
        .sel(sel),
        .op1(op1),
        .op2(op2),
        .a_out(out_arithmetic)
    );

    always @(*) begin
        case (control)
            1'b0: out = out_logic;
            1'b1: out = out_arithmetic;
        endcase
    end


    lcd_display lcdmod (
        .clk(clk),
        .data_in(out),
        .LCD_DATA(LCD_DATA),
        .LCD_RS(LCD_RS),
        .LCD_RW(LCD_RW),
        .LCD_EN(LCD_EN),
        .LCD_ON(LCD_ON),
        .LCD_BLON(LCD_BLON)
    );

endmodule
```

Arithmetic instructions module:

```
module a_instr(
        input [3:0] sel,
   input [3:0] op1,
   input [3:0] op2,
   output reg [7:0] a_out

        );


        wire    [7:0] div_out;
        wire    [3:0] mod_out;

        divmod d(
   .op1(op1),
   .op2(op2),
   .quotient(div_out),
   .remainder(mod_out)
);


        always @(*) begin

        case(sel)
                4'b0000: a_out = {4'b0000,op1 + op2};//addition
                4'b0001: a_out = {4'b0000,op1-op2};//subtraction
                4'b0010: a_out = op1*op2;//multiplication
                4'b0011: a_out = op1 + 4'b0001;//increment
                4'b0100: a_out = op1 - 4'b0001;//decrement
                4'b1000: a_out = div_out;//division
                4'b1001: a_out = {4'b0000,mod_out};//modulus
                default: a_out = 8'b0;
        endcase
        end

        endmodule
```

DIVISION MODULE:

```
module divmod(
```

```
    input  [3:0] op1,       // Dividend
    input  [3:0] op2,       // Divisor
    output reg [7:0] quotient, // 4-bit integer + 4-bit fractional part
    output reg [3:0] remainder
);
    integer i;
    reg [3:0] rem;
    reg [7:0] result;

    always @(*) begin
        // Default assignments for all variables
        quotient  = 8'd0;
        remainder = 4'd0;
        result    = 8'd0;
        rem       = 4'd0;
        i         = 0;

        if (op2 == 0) begin
            quotient  = 8'hFF;
            remainder = 4'hF;
        end else begin
            rem = op1;

            // Integer part
            result[7:4] = op1 / op2;
            rem = op1 % op2;

            // Fractional part
            for (i = 3; i >= 0; i = i - 1) begin
                rem = rem << 1;
                if (rem >= op2) begin
                    result[i] = 1;
                    rem = rem - op2;
                end else begin
                    result[i] = 0;
                end
            end

            quotient  = result;
```

```
            remainder = rem;
        end
    end
endmodule
```

logical instructions module:

```
module l_instr(

            input [3:0] sel,
    input [3:0] op1,
    input [3:0] op2,
    output reg [7:0] l_out
            );


            always @(*) begin
            case(sel)
                    4'b0000: l_out = {4'b0000,op1 & op2};//and
                    4'b0001: l_out = {4'b0000,op1 | op2};//or
                    4'b0010: l_out = {4'b0000,op1 ^ op2};//xor
                    4'b0011: l_out = {4'b0000, ~op1};//not
                    4'b0100: l_out = {4'b0000,~(op1 & op2)};//nand
                    4'b0101: l_out = {4'b0000,~(op1 | op2)};//nor
                    4'b0110: l_out = {4'b0000,op1 ^~ op2};//xnor
                    4'b0111: l_out = {4'b0000, op1>>1};//shift right logical
                    4'b1000: l_out = {4'b0000, op1<<1};//shift left logical
                    default: l_out = 8'b0;

            endcase
            end

endmodule
```

LCD Display module:

```
module lcd_display (
    input wire clk,
    input wire [7:0] data_in,
    output reg [7:0] LCD_DATA,
    output reg LCD_RS,
```

```
    output reg LCD_RW,
    output reg LCD_EN,
    output reg LCD_ON,
    output reg LCD_BLON
);
    reg [25:0] count;
    reg [4:0] state;
    reg [7:0] hex_high, hex_low;
    reg [7:0] lcd_string [0:15];
    reg [3:0] char_index;


    function [7:0] to_ascii;
        input [3:0] nibble;
        begin
            if (nibble < 10)
                to_ascii = "0" + nibble;
            else
                to_ascii = "A" + (nibble - 10);
        end
    endfunction

    initial begin
        LCD_ON   = 1;
        LCD_BLON = 1;
    end

    always @(posedge clk) begin
        count <= count + 1;


        hex_high <= to_ascii(data_in[7:4]);
        hex_low  <= to_ascii(data_in[3:0]);


        lcd_string[0]  <= "R";
        lcd_string[1]  <= "e";
        lcd_string[2]  <= "s";
        lcd_string[3]  <= "u";
```

```
        lcd_string[4]  <= "l";
        lcd_string[5]  <= "t";
        lcd_string[6]  <= ":";
        lcd_string[7]  <= " ";
        lcd_string[8]  <= "0";
        lcd_string[9]  <= "x";
        lcd_string[10] <= hex_high;
        lcd_string[11] <= hex_low;
        lcd_string[12] <= " ";
        lcd_string[13] <= " ";
        lcd_string[14] <= " ";
        lcd_string[15] <= " ";

        // Generate enable pulse every ~10ms
        if (count[19:0] == 20'd0) begin
            LCD_EN <= 1;
            LCD_RW <= 0;     // Always writing
            LCD_RS <= 1;     // Sending data
            LCD_DATA <= lcd_string[char_index];
            char_index <= char_index + 1;
            if (char_index == 15)
                char_index <= 0;
        end else begin
            LCD_EN <= 0;
        end
    end
endmodule
```

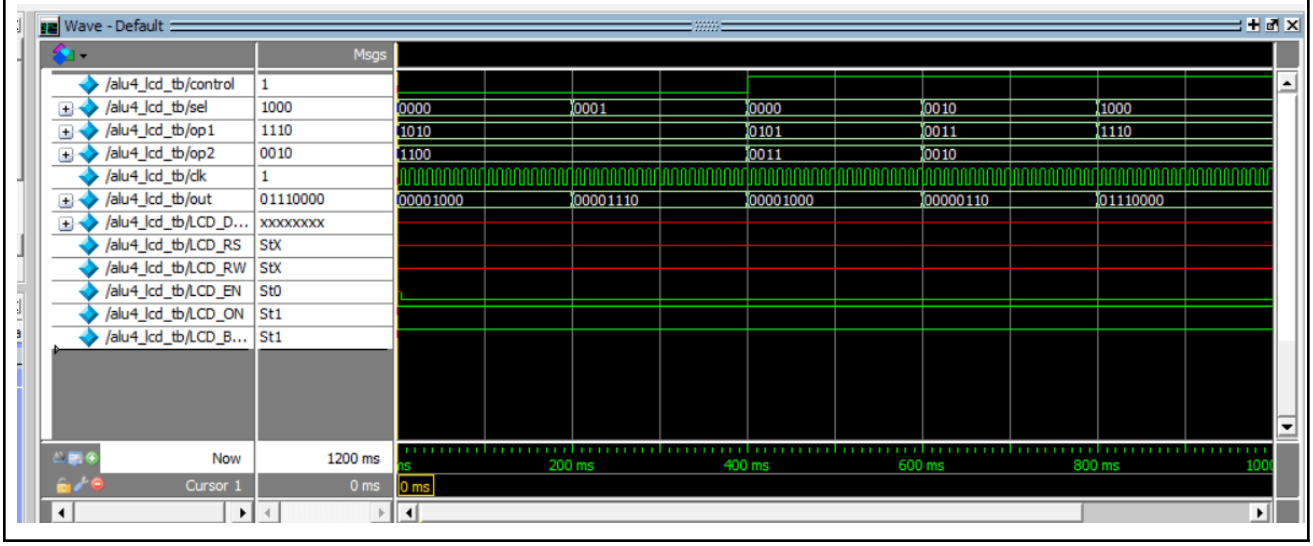
Testbench:

```
`timescale 1ms/1ms

module alu4_lcd_tb;

    // Testbench signals
    reg control;
    reg [3:0] sel;
    reg [3:0] op1;
    reg [3:0] op2;
    reg clk;
```

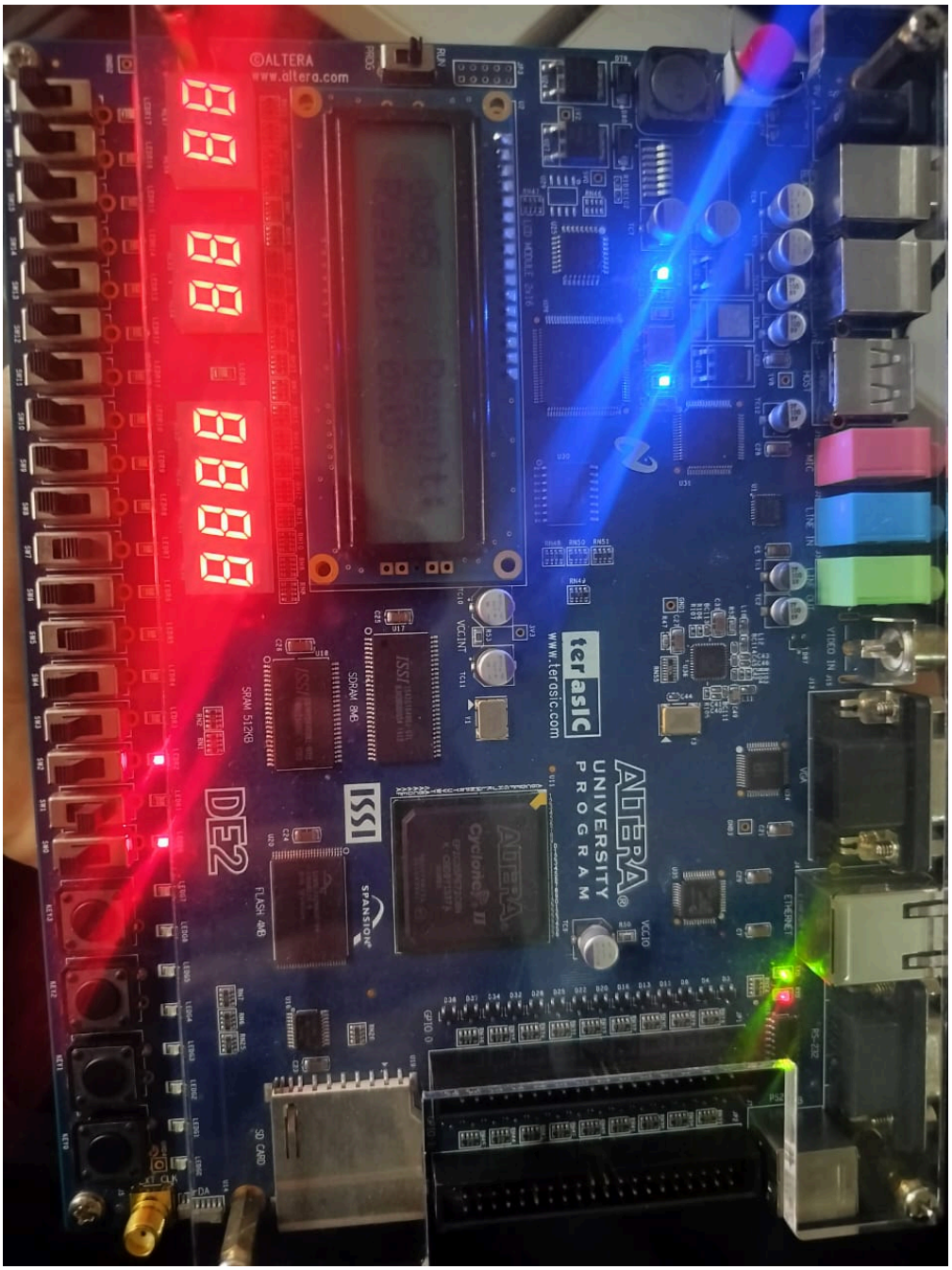
```
wire [7:0] out;

// LCD signal wires
wire [7:0] LCD_DATA;
wire LCD_RS, LCD_RW, LCD_EN, LCD_ON, LCD_BLON;

// Clock generation
initial begin
    clk = 0;
    forever #5 clk = ~clk; // 100 Hz for simulation visibility
end

// Instantiate the ALU
alu4_lcd uut (
    .control(control),
    .sel(sel),
    .op1(op1),
    .op2(op2),
    .out(out),
    .clk(clk), // required for lcd_display inside
    .LCD_DATA(LCD_DATA),
    .LCD_RS(LCD_RS),
    .LCD_RW(LCD_RW),
    .LCD_EN(LCD_EN),
    .LCD_ON(LCD_ON),
    .LCD_BLON(LCD_BLON)
);

initial begin
    $display("Time\tControl\tSel\tOp1\tOp2\tOut\tLCD_EN\tLCD_RS\tLCD_DATA");
    $monitor("%0t\t%b\t%04b\t%04b\t%04b\t%08b\t%b\t%b\t%8b",
            $time, control, sel, op1, op2, out, LCD_EN, LCD_RS, LCD_DATA);

    // Test 1: Logic AND
    control = 0;
    sel = 4'b0000; op1 = 4'b1010; op2 = 4'b1100; #200;

    // Test 2: Logic OR
```

```
        sel = 4'b0001; op1 = 4'b1010; op2 = 4'b1100; #200;

        // Test 3: Arithmetic ADD
        control = 1;
        sel = 4'b0000; op1 = 4'b0101; op2 = 4'b0011; #200;

        // Test 4: Arithmetic MUL
        sel = 4'b0010; op1 = 4'b0011; op2 = 4'b0010; #200;

        // Test 5: Division
        sel = 4'b1000; op1 = 4'b1110; op2 = 4'b0010; #400;

        // End simulation
        $stop;
    end

endmodule
```

Output:



```
add wave -position end  sim:/alu4_lcd_tb/out
add wave -position end  sim:/alu4_lcd_tb/LCD_DATA
add wave -position end  sim:/alu4_lcd_tb/LCD_RS
add wave -position end  sim:/alu4_lcd_tb/LCD_RW
add wave -position end  sim:/alu4_lcd_tb/LCD_EN
add wave -position end  sim:/alu4_lcd_tb/LCD_ON
add wave -position end  sim:/alu4_lcd_tb/LCD_BLON
VSIM 14> run -all
# Time  Control Sel     Op1     Op2     Out
# 0     0       0000    1010    1100    00001000
# 200   0       0001    1010    1100    00001110
# 400   1       0000    0101    0011    00001000
# 600   1       0010    0011    0010    00000110
# 800   1       1000    1110    0010    01110000
# Break in Module alu4_lcd_tb at C:/altera/13.0sp1/my projects/alu_lcd/alu4_lcd_tb.v line 63

VSIM 15>
```



Future Scope of the project

1. Expand the ALU to support 8-bit or 16-bit operations for more complex computation.
2. Implement pipelining to enhance processing speed and throughput.

3. Interface the ALU with a soft-core CPU for testing as part of a full processor datapath.
4. Add real-time debugging features through UART or 7-segment displays.

Conclusion:

In conclusion, the design and implementation of the 4-bit ALU with LCD display integration served as a practical and insightful exercise in digital logic design and hardware description languages. It not only reinforced core concepts of arithmetic and logical operations but also provided valuable experience in modular coding, hardware interfacing, and real-time result visualization. This project lays a strong foundation for building more complex processor components in the future and demonstrates the potential of Verilog-based design in developing efficient and scalable digital systems.

Signature of faculty in charge with date: