**COAL Lab # 02**

**ALU Designing in Verilog**

**Name:** Muhammad Usman

**Class:** BSCS 3C1

**Reg No:** cs211208

**Lab Exercise:**

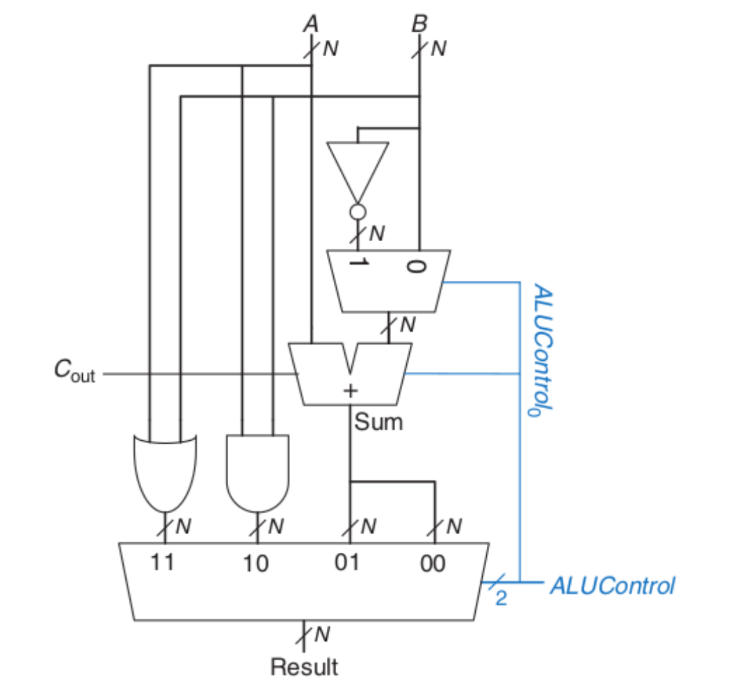
**Literature Review:**

An arithmetic-logic unit is the part of a central processing unit that carries out arithmetic and logic operations on the operands.ALU is a combinational circuit and the part of CPU that carries out arithmetic such as addition, subtraction, multiplication and logical operations such as AND, OR, NOT operations on the operands as per computer instructions.Multiplexer is being used in this circuit to select the specific output on the final result.

**Task:**

Write a Verilog module for the above ALU unit. Make sure all input and output signals have a unique name. Make a test bench to test all the operations by taking inputs of your own choices. Attach the test bench and the output waveforms.

**Block Diagram:**

****

**Boolean Equation:**

A\_and\_B = A & B;

A\_or\_B = A | B;

A\_sum\_B = A + B + control[0];

A\_diff\_B = A + ~(B) + control[0];

**Verilog Code:**

design.v

module ALU(A,B,Q,control);

    input [31:0]A,B;

    input[1:0]control;

    wire Cout;

    output [31:0]Q;

    wire [31:0]A\_or\_B,A\_and\_B,B\_not,A\_sum\_B;

    wire [31:0]S1;

    assign A\_or\_B = A | B;

    assign A\_and\_B = A & B;

    assign B\_not = ~B;

    assign S1 = (control[0] == 1'b1) ? B\_not : B;

    assign {Cout,A\_sum\_B} = A+S1+control[0];

    assign Q = (control == 2'b00) ? A\_sum\_B : (control == 2'b01) ? A\_sum\_B : (control == 2'b10) ? A\_and\_B : A\_or\_B;

endmodule

testbench.v

module tb();

    reg [31:0]A,B;

    reg [1:0]control;

    wire [31:0]Q;

    ALU dut (.A(A),.B(B),.Q(Q),.control(control));

    initial begin

      $dumpfile("dump.vcd");

      $dumpvars(0);

    end

    initial begin

        control <= 2'b00;

        A <= 32'd70;

        B <= 32'd30;

        #100

        control <= 2'b01;

        A <= 32'd100;

        B <= 32'd20;

        #100

        control <= 2'b10;

        A <= 32'd10;

        B <= 32'd50;

        #100

        control <= 2'b11;

        A <= 32'd20;

        B <= 32'd40;

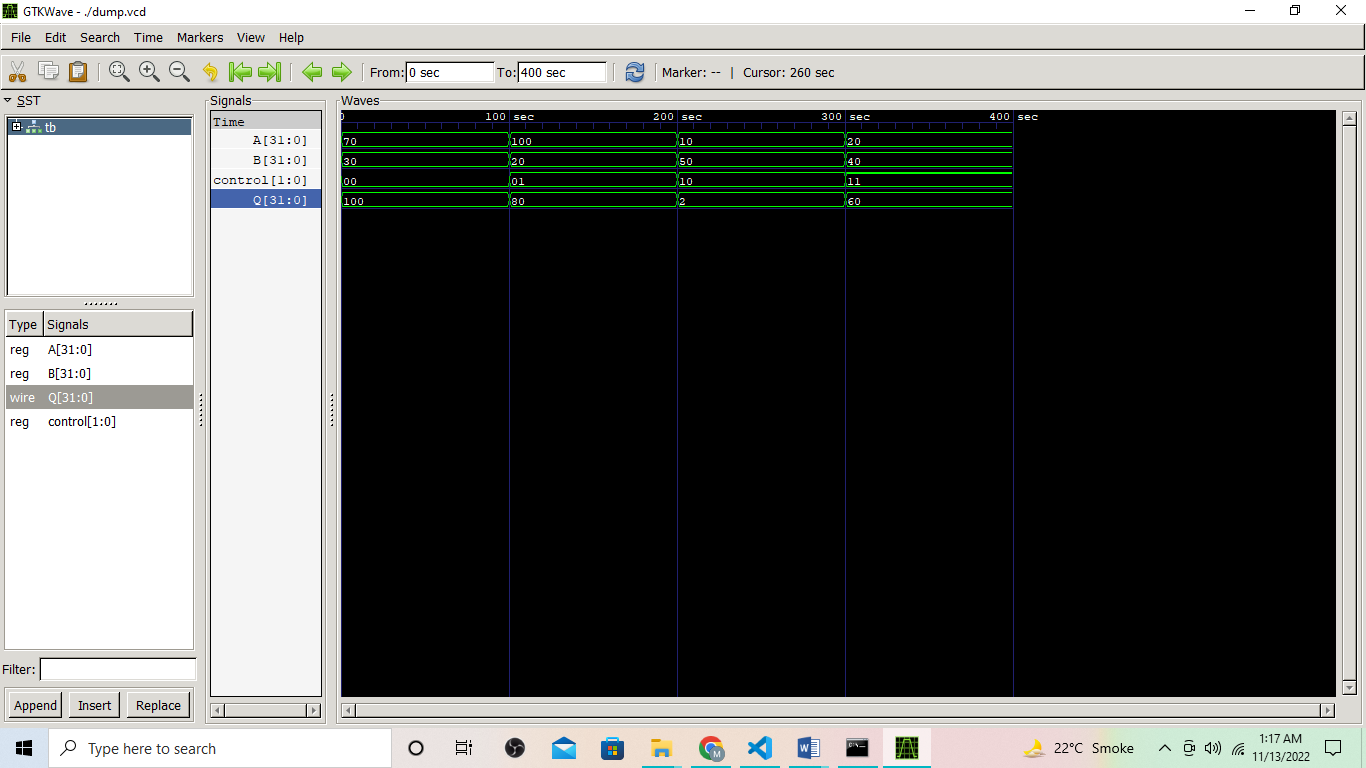
        #100

        $finish;

    end

endmodule

**Waveforms:**



**Post Lab Task:**

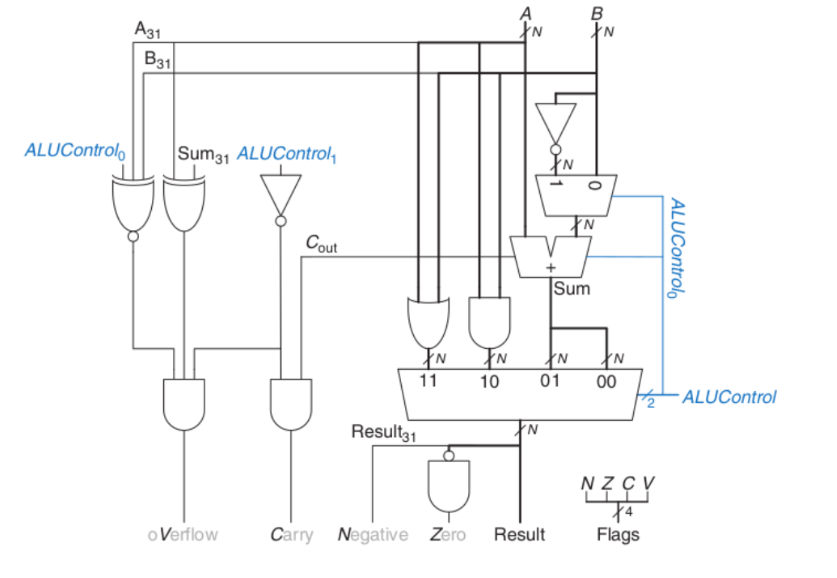
**Literature Review:**

An arithmetic-logic unit is the part of a central processing unit that carries out arithmetic and logic operations on the operands.ALU is a combinational circuit and the part of CPU that carries out arithmetic such as addition, subtraction, multiplication and logical operations such as AND, OR, NOT operations on the operands as per computer instructions.Multiplexer is being used in this circuit to select the specific output on the final result with Four flag outputs with one bits whose purpose is to show to true or false. Zero flag will produce logic level high, if output is zero. Negative flag will produce logic level high, if MSB of result is high. Carry flag will produce logic level high, if carry bit will have logic level high. Overflow condition is when specified output bits cannot store the produced output, so an overflow flag will generate logic level high.

**Task:**

Write a Verilog module for the given ALU figure. Make a test bench to test all the operations by taking inputs of your own choices. Attach the test bench and the output waveforms.

**Block Diagram:**

****

**Boolean Equation:**

A\_and\_B = A & B;

A\_or\_B = A | B;

A\_sum\_B = A + B + control[0];

A\_diff\_B = A + ~(B) + control[0];

result = Final Output;

Zero = &(~result);

Negative = result[31];

Verflow = ~(control[0] ^ A[31] ^ B[31]) & (A[31] ^ A\_sum\_B [31]) & (~control[1]);

Carry = ~control[1] & Cout;

**Verilog Code:**

design.v

module ALU(A,B,result,control,Verflow,Carry,Negative,Zero);

    input [31:0]A,B;

    input[2:0]control;

    output [31:0]result;

    output Verflow,Carry,Negative,Zero;

    wire Cout;

    wire [31:0]A\_or\_B,A\_and\_B,B\_not,A\_sum\_B;

    wire A\_xor\_sum,A\_xnor\_B\_xnor\_control,control\_1\_not;

    wire [31:0]S1;

    // Interim Outputs

    assign A\_or\_B = A | B;

    assign A\_and\_B = A & B;

    assign B\_not = ~B;

    assign S1 = (control[0] == 1'b1) ? B\_not : B;

    assign {Cout,A\_sum\_B} = A+S1+control[0];

    assign A\_xor\_sum = A[31]^A\_sum\_B[31];

    assign A\_xnor\_B\_xnor\_control = ~(A[31]^B[31]^control[0]);

    assign control\_1\_not = ~(control[1]);

    // Flags

    assign Verflow = A\_xnor\_B\_xnor\_control & A\_xor\_sum & control\_1\_not;

    assign Zero = &(~result);

    assign Negative = result[31];

    assign Carry = control\_1\_not & Cout;

    // Final Output

    assign result = (control[1:0] == 2'b00) ? A\_sum\_B : (control[1:0] == 2'b01) ? A\_sum\_B : (control[1:0] == 2'b10) ? A\_and\_B : A\_or\_B ;

endmodule

testbench.v

module tb();

    reg [31:0]A,B;

    reg [1:0]control;

    wire [31:0]result;

    wire Verflow,Carry,Negative,Zero;

    ALU dut (.A(A),.B(B),.result(result),.control(control),.Verflow(Verflow),.Carry(Carry),.Negative(Negative),.Zero(Zero));

    initial begin

      $dumpfile("dump.vcd");

      $dumpvars(0);

    end

    initial begin

        control <= 2'b00;

        A <= 32'b10000000000000000000000000000000;

        B <= 32'b10000000000000000000000000000001;

        #100

        control <= 2'b01;

        A <= 32'd40;

        B <= 32'd40;

        #100

        control <= 2'b10;

        A <= 32'b01010101010101010101010101010101;

        B <= 32'b10101010101010101010101010101010;

        #100

        control <= 2'b11;

        A <= 32'b00001110000000000000000000000001;

        B <= 32'b00000000000000110000000000001111;

        #100

        $finish;

    end

endmodule

**Waveforms:**

