COAL Lab-02

ALU Designing in Verilog

Name: Saad Nisar Butt

Reg. no: cs211246

Class: BS-CS-3C-1

Lab Exercise

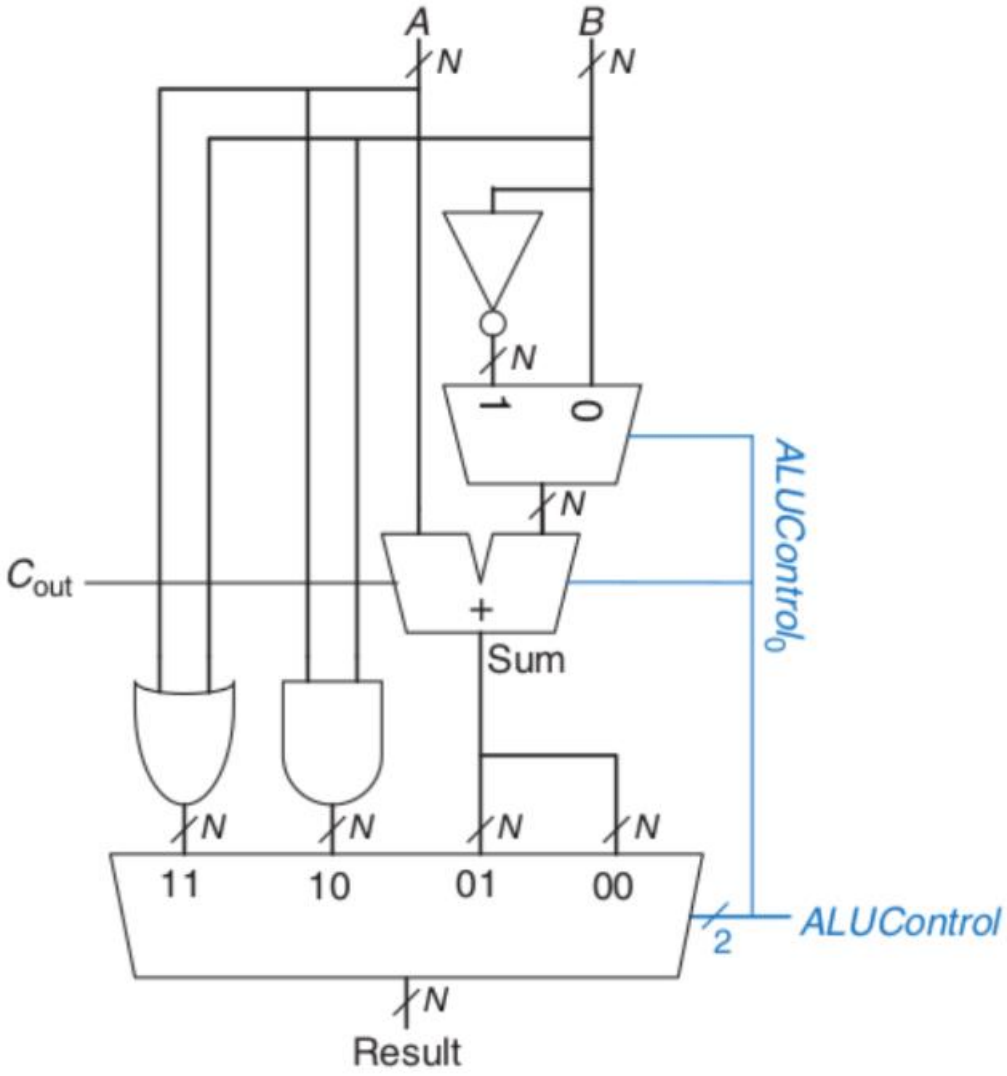
Literature Review:

ALU which stands for Arithmetic Logic Unit is combinational digital circuit and the part of central processing unit 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. In this task, a part of ALU is designed that can perform Addition, Subtraction as arithmetic operations and And, Or operations as logical operations on two inputs. By using a 4x1 multiplexer, a specific output will be generated which will be based upon control signal used as an input.

Task:

Write a Verilog module for the ALU unit shown below. Make sure all input and output signals have a unique name. Make 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 Equations:

Sum = A + B + ALUControl[0]

Difference = A + (~B) + ALUControl[0]

And = A & B

Or = A | B

Verilog Code:

design.v

```
module ALU(A,B,ctrl,Result);

  // Inputs
  input [31:0] A,B;
  input [2:0] ctrl;

  // Outputs
  output [31:0] Result;

  // Interim Wires
  wire [31:0] A_and_B, A_or_B, B_not, A_sum_B;
  wire [31:0] S1;
  wire Cout;

  // Logic Designing
  // And
  assign A_and_B = A & B;
  // Or
  assign A_or_B = A | B;
  // Not
  assign B_not = ~B;
  // 2x1 MUX
  assign S1 = (ctrl[0] == 1'b1) ? B_not : B;
  // Addition / Subtraction
  assign {Cout, A_sum_B} = A + S1 + ctrl[0];
  // Result output through 4x1 MUX
  assign Result = (ctrl[1:0] == 2'b00) ? A_sum_B : (ctrl[1:0] == 2'b01) ? A_sum_B
: (ctrl[1:0] == 2'b10) ? A_and_B : A_or_B;
```

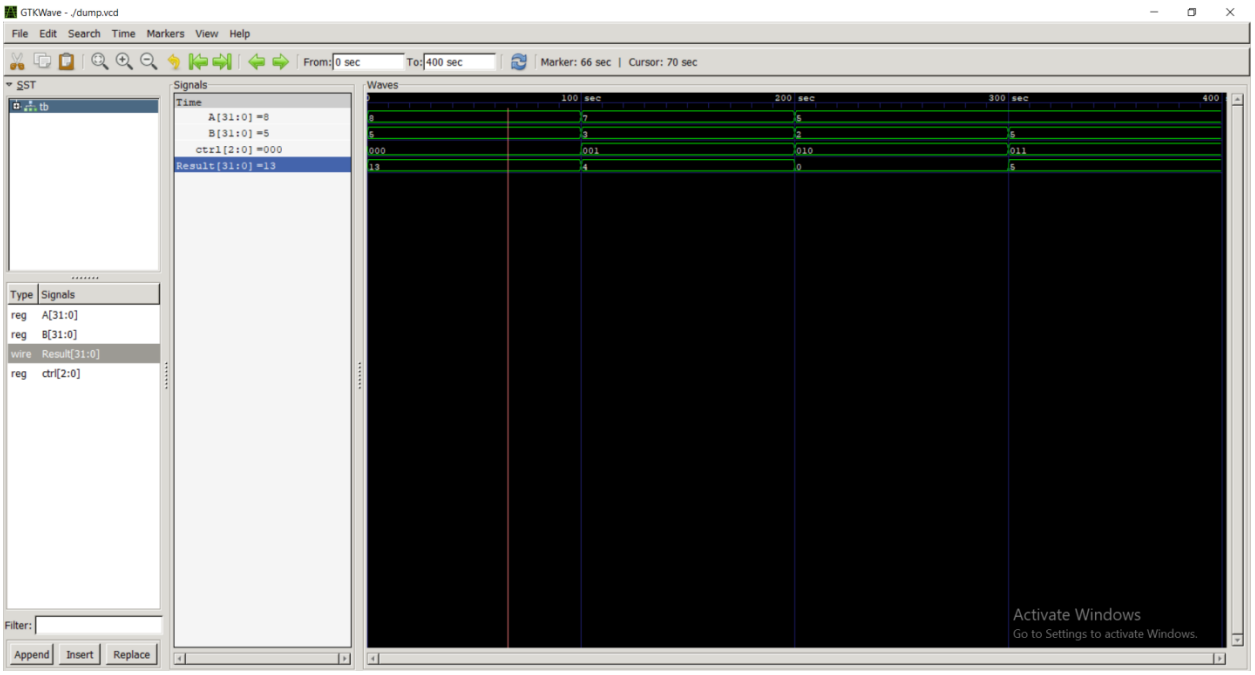
```
endmodule
```

testbench.v

```
module tb();

  reg [31:0]A,B;
  reg [2:0] ctrl;
  wire [31:0] Result;

  // Module Declaratrion
  ALU dut(
    .A(A), .B(B), .ctrl(ctrl), .Result(Result)
  );

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end

  initial begin
    A <= 32'd8;
    B <= 32'd5;
    ctrl <= 3'b000;
    #100;

    A <= 32'd8;
    B <= 32'd5;
    ctrl <= 3'b001;
    #100;

    A <= 32'd8;
    B <= 32'd5;
    ctrl <= 3'b010;
    #100;

    A <= 32'd8;
    B <= 32'd5;
    ctrl <= 3'b011;
    #100;

    A <= 32'd8;
    B <= 32'd5;
    ctrl <= 3'b101;
```

```
    #100;
  end

endmodule
```

Waveforms:



Post Lab Task

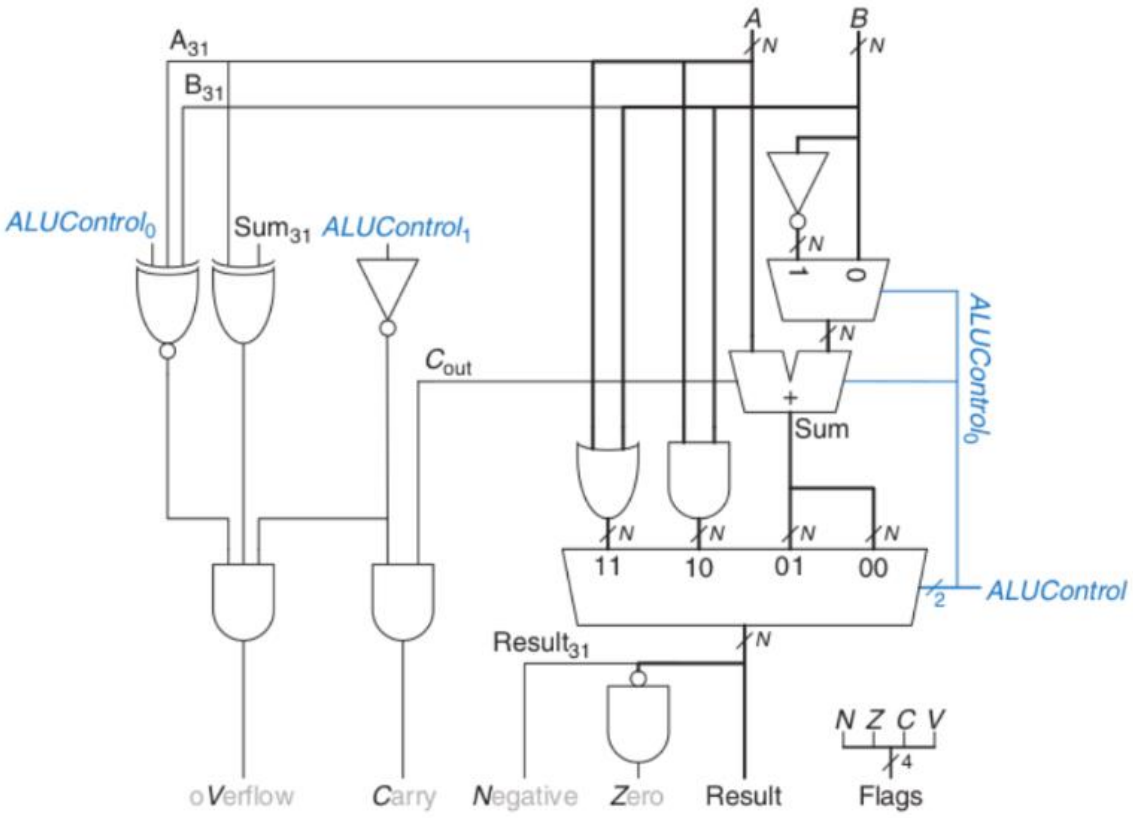
Literature Review:

ALU which stands for Arithmetic Logic Unit is combinational digital circuit and the part of central processing unit 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. In this task, a part of ALU is designed that can perform Addition, Subtraction as arithmetic operations and And, Or operations as logical operations on two inputs. By using a 4x1 multiplexer, a specific output will be generated which will be based upon control signal used as an input. Four flag outputs are here which acts as Boolean operations like 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 ouput, so an overflow flag will generate logic level high if there will be that specific condition.

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 Equations:

Sum = A + B + ALUControl[0]

Difference = A + (~B) + ALUControl[0]

And = A & B

Or = A | B

Result = ALU Output

Z = &(~Result)

N = Result[31]

C = (~ALUControl[1]) & Cout

V = (~(ALUControl[0] ^ A[31] ^ B[31])) & (A[31] ^ Sum[31]) & (~ALUControl[1])

Verilog Code:

design.v

```
module Flags_ALU(A, B, ctrl, Result, Z, N, C, V);

  // inputs
  input [31:0] A, B;
  input [2:0] ctrl;

  // outputs
  output [31:0] Result;
  // Flags for Zero, Negative, Carry and Overflow respectively.
  output Z,N,C,V;

  // interim wires
  wire [31:0] A_and_B, A_or_B, B_not, A_sum_B;
  wire [31:0] S1;
  wire [31:0] not_Result;
  wire Cout, xor_A_Sum, xnor_A_B_ctrl0, ctrl1_not;

  // Logic Designing
  // And
  assign A_and_B = A & B;
  // Or
  assign A_or_B = A | B;
  // Not
  assign B_not = ~B;
  // 2x1 Mux for addition or subtraction
  assign S1 = (ctrl[0] == 1'b1) ? B_not : B;
  // Addition / Subtraction
  assign {Cout, A_sum_B} = A + S1 + ctrl[0];
  // Result output through 4x1 Mux
  assign Result = (ctrl[1:0] == 2'b00) ? A_sum_B :
                  (ctrl[1:0] == 2'b01) ? A_sum_B :
                  (ctrl[1:0] == 2'b10) ? A_and_B : A_or_B;

  // Flags Outputs
  // for zero checking
```

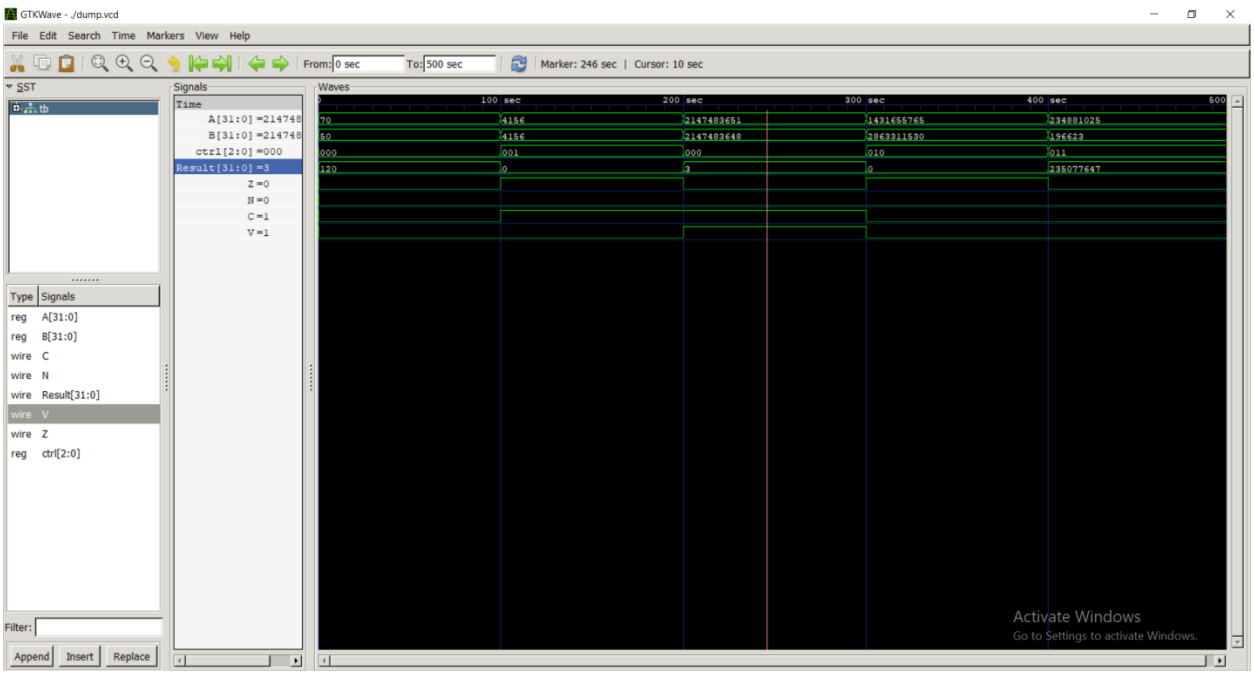
```
  assign not_Result = ~Result;
  assign Z = &(not_Result);
  // for negative checking
  assign N = Result[31];
  //for carry checking
  assign ctrl1_not = (~ctrl[1]);
  assign C = ctrl1_not & Cout;
  // for overflow checking
  assign xor_A_Sum = A_sum_B[31] ^ A[31];
  assign xnor_A_B_ctrl0 = ~(A[31] ^ B[31] ^ ctrl[0]);
  assign V = ctrl1_not & xor_A_Sum & xnor_A_B_ctrl0;

endmodule
```

testbench.v

```
module tb();

  reg [31:0]A,B;
  reg [2:0] ctrl;
  wire [31:0] Result;
  wire Z,N,C,V;

  Flags_ALU dut(
    .A(A), .B(B), .ctrl(ctrl), .Result(Result), .Z(Z),.N(N),.C(C),.V(V)
  );

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end

  initial begin
    A <= 32'b00000000000000001110000011110000;
    B <= 32'b00000001000000001010000010110000;
    ctrl <= 3'b000;
    #100;

    A <= 32'b00000000000000000001000000111100;
    B <= 32'b00000000000000000001000000111100;
    ctrl <= 3'b001;
    #100;
```

```
    A <= 32'b10000000000000000000000000000011;
    B <= 32'b10000000000000000000000000000000;
    ctrl <= 3'b000;

    A <= 32'b01010101010101010101010101010101;
    B <= 32'b10101010101010101010101010101010;
    ctrl <= 3'b010;
    #100;

    A <= 32'b00001111000000000000000000000001;
    B <= 32'b00000000000000011000000000001111;
    ctrl <= 3'b011;
    #100;

  end

endmodule
```

Waveforms:



_____