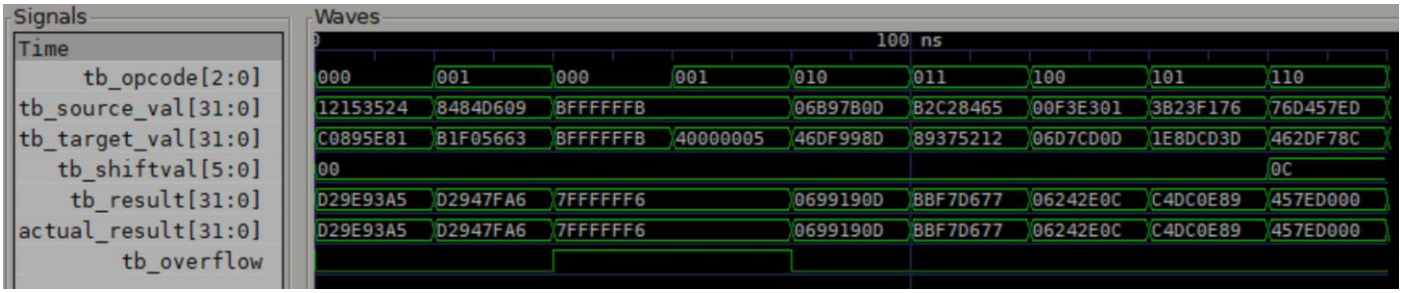
**1차 과제 – 32bit ALU**

**2022104346 정지헌**

**Simulation results:**




**Signals Description**

tb_opcode: ALU input, what the ALU does is determined according to the following:

- 000: add
- 001: sub
- 010: and
- 011: or
- 100: xor
- 101: not A
- 110: shift left by B
- 111: Unassigned

tb_source_val: ALU input, operand A

tb_target_val: ALU input, operand B

tb_shiftval: Testbench inner variable, how many bits have been shifted, in the shift operation

tb_result: ALU output, operation result of the ALU

actual_result: Testbench inner variable, what the ALU is supposed to output

tb_overflow: ALU output, indicates overflow in add and sub operations.

**Source codes**

https://github.com/wjdwlgjs/Hong_Verilog/tree/main/ALU

ALU_tb.v

```
`include "ALU/ALU32Bit.v"
`timescale 1ns/1ns

module ALU_tb();
    reg signed [31:0] tb_source_val;
    reg signed [31:0] tb_target_val;
    reg signed [2:0] tb_opcode;

    wire signed [31:0] tb_result;
    wire tb_overflow;

    reg [5:0] tb_shiftval;

    reg signed [31:0] actual_result;
```

```
    ALU32Bit TestALU(
        .source_val_i(tb_source_val),
        .target_val_i(tb_target_val),
        .opcode_i(tb_opcode),
        .alu_result_o(tb_result),
        .alu_overflow_o(tb_overflow)
    );

    initial begin
        $dumpfile("ALU\\BuildFiles\\ALU_tb.vcd");
        $dumpvars(0, ALU_tb);
        $monitor("$time: %0d, a: %0d, b: %0d, opcode: %b, result: %0d, ans: %0d,
overflow: %b, shiftval: %0d", $time, tb_source_val, tb_target_val, tb_opcode, tb_result,
actual_result, tb_overflow, tb_shiftval);

        tb_shiftval = 0;
        // test add operation
        tb_source_val = $random;
        tb_target_val = $random;
        tb_opcode = 3'b000;
        actual_result = tb_source_val + tb_target_val;

        #20 // test sub operation
        tb_source_val = $random;
        tb_target_val = $random;
        tb_opcode = 3'b001;
        actual_result = tb_source_val - tb_target_val;

        #20 // test add operation with overflow
        tb_source_val = -1073741829;
        tb_target_val = -1073741829;
        tb_opcode = 3'b000;
        actual_result = tb_source_val + tb_target_val;

        #20 // test sub operation with overflow
        tb_source_val = -1073741829;
        tb_target_val = 1073741829;
        tb_opcode = 3'b001;
        actual_result = tb_source_val - tb_target_val;

        #20 // test bitwise AND operation
        tb_source_val = $random;
        tb_target_val = $random;
        tb_opcode = 3'b010;
        actual_result = tb_source_val & tb_target_val;

        #20 // test bitwise OR operation
        tb_source_val = $random;
        tb_target_val = $random;
        tb_opcode = 3'b011;
        actual_result = tb_source_val | tb_target_val;
```

```
        #20 // test bitwise XOR operation
        tb_source_val = $random;
        tb_target_val = $random;
        tb_opcode = 3'b100;
        actual_result = tb_source_val ^ tb_target_val;

        #20 // test not A operation
        tb_source_val = $random;
        tb_target_val = $random;
        tb_opcode = 3'b101;
        actual_result = ~tb_source_val;

        #20 // test shift by B operation
        tb_source_val = $random;
        tb_target_val = $random;
        tb_opcode = 3'b110;
        tb_shiftval = tb_target_val[5:0];
        actual_result = tb_source_val << tb_shiftval;

        #20 // unassigned operation
        tb_source_val = $random;
        tb_target_val = $random;
        tb_opcode = 3'b111;
        actual_result = 0;
    end

endmodule
```

ALU32Bit.v

```
`include "ALU/Submodules/Adder32Bit.v"
`include "ALU/Submodules/MUX32Bit3X8.v"
`include "ALU/Submodules/ConditionalInverter32Bit.v"

module ALU32Bit(
    input [31:0] source_val_i, // A
    input [31:0] target_val_i, // B
    input [2:0] opcode_i,
    output [31:0] alu_result_o,
    output alu_overflow_o
    );

    /*
    opcode table:
    000: add
    001: sub (invert B)
    010: and
    011: or
    100: xor
    101: not (invert A)
    110: shift left
    111: Unassigned
    */
```

```
    // add/subtract
    wire [31:0] flipped_b;
    wire sub; // 1 when opcode == 001 (A sub B operation)
    assign sub = ~opcode_i[2] & ~opcode_i[1] & opcode_i[0];

    ConditionalInverter32Bit BInverter(
        .target_i(target_val_i),
        .inv_sel_i(sub),
        .flip_result_o(flipped_b)
    );

    wire [31:0] add_sub_result;
    wire temp_overflow;

    Adder32Bit AdderSubtractor(
        .a32_i(source_val_i),
        .b32_i(flipped_b),
        .cin32_i(sub),
        .sum32_o(add_sub_result),
        .cout32_o(),
        .adder_overflow_o(temp_overflow)
    );
    assign alu_overflow_o = ~opcode_i[2] & ~opcode_i[1] & temp_overflow;

    // and
    wire [31:0] and_result;
    assign and_result = source_val_i & flipped_b;

    // or
    wire [31:0] or_result;
    assign or_result = source_val_i | flipped_b;

    // xor
    wire [31:0] xor_result;
    assign xor_result = source_val_i ^ flipped_b;

    // not A
    wire [31:0] flipped_a;
    wire flip_a; // 1 when opcode == 101

    assign flip_a = opcode_i[2] & ~opcode_i[1] & opcode_i[0];

    ConditionalInverter32Bit AInverter(
        .target_i(source_val_i),
        .inv_sel_i(flip_a),
        .flip_result_o(flipped_a)
    );

    // shift left
    wire [31:0] shifted_a;
    assign shifted_a = source_val_i << target_val_i[5:0];

    // MUX part
```

```
    MUX32Bit3X8 final_mux(
        .in000_i(add_sub_result),
        .in001_i(add_sub_result),
        .in010_i(and_result),
        .in011_i(or_result),
        .in100_i(xor_result),
        .in101_i(flipped_a),
        .in110_i(shifted_a),
        .in111_i(32'b0),
        .sel_3bit_i(opcode_i),
        .result_3x8mux_o(alu_result_o)
    );

endmodule
```

SubModules/SingleBitAdder.v

```
module SingleBitAdder(
    input a_i,
    input b_i,
    input cin_i,

    output sum_o,
    output cout_o
    );

    assign sum_o = a_i ^ b_i ^ cin_i;
    assign cout_o = a_i & b_i | a_i & cin_i | b_i & cin_i;

endmodule
```

SubModules/Adder32Bit.v

```
`include "ALU/SubModules/SingleBitAdder.v"

module Adder32Bit(
    input [31:0] a32_i,
    input [31:0] b32_i,
    input cin32_i,

    output [31:0] sum32_o,
    output cout32_o,
    output adder_overflow_o
    );

    wire [32:0] carry;

    assign carry[0] = cin32_i;
    assign cout32_o = carry[32];
    assign adder_overflow_o = carry[32] ^ carry[31];

    genvar i;

    generate
```

```
        for (i = 0; i < 32; i = i + 1) begin
            SingleBitAdder AdderInst(
                .a_i(a32_i[i]),
                .b_i(b32_i[i]),
                .cin_i(carry[i]),
                .sum_o(sum32_o[i]),
                .cout_o(carry[i+1])
            );
        end
    endgenerate

endmodule
```

SubModules/ConditionalInverter32Bit.v

```
module ConditionalInverter32Bit(
    // flips the bits of target_i if inv_sel_i == 1;
    input [31:0] target_i,
    input inv_sel_i,

    output [31:0] flip_result_o
    );

    assign flip_result_o = target_i ^ {32{inv_sel_i}};
endmodule
```

SubModules/MUX32Bit2X4.v

```
module MUX32Bit2X4(
    input [31:0] in00_i,
    input [31:0] in01_i,
    input [31:0] in10_i,
    input [31:0] in11_i,
    input [1:0] sel_2bit_i,

    output [31:0] result_2x4mux_o
    );

    wire [3:0] decoded_sel;
    wire [31:0] mid00;
    wire [31:0] mid01;
    wire [31:0] mid10;
    wire [31:0] mid11;

    assign decoded_sel[0] = ~sel_2bit_i[1] & ~sel_2bit_i[0];
    assign decoded_sel[1] = ~sel_2bit_i[1] & sel_2bit_i[0];
    assign decoded_sel[2] = sel_2bit_i[1] & ~sel_2bit_i[0];
    assign decoded_sel[3] = sel_2bit_i[1] & sel_2bit_i[0];

    assign mid00 = in00_i & {32{decoded_sel[0]}};
    assign mid01 = in01_i & {32{decoded_sel[1]}};
    assign mid10 = in10_i & {32{decoded_sel[2]}};
    assign mid11 = in11_i & {32{decoded_sel[3]}};
```

```
    assign result_2x4mux_o = mid00 | mid01 | mid10 | mid11;

endmodule
```

SubModules/MUX32Bit3X8.v

```
`include "ALU/SubModules/MUX32Bit2X4.v"

module MUX32Bit3X8(
    input [31:0] in000_i,
    input [31:0] in001_i,
    input [31:0] in010_i,
    input [31:0] in011_i,
    input [31:0] in100_i,
    input [31:0] in101_i,
    input [31:0] in110_i,
    input [31:0] in111_i,
    input [2:0] sel_3bit_i,

    output [31:0] result_3x8mux_o
    );

    wire [31:0] zero_three_result;
    wire [31:0] four_seven_result;

    MUX32Bit2X4 ZeroToThree(
        .in00_i(in000_i), // 0
        .in01_i(in001_i), // 1
        .in10_i(in010_i), // 2
        .in11_i(in011_i), // 3
        .sel_2bit_i(sel_3bit_i[1:0]),
        .result_2x4mux_o(zero_three_result)
    );

    MUX32Bit2X4 FourToSeven(
        .in00_i(in100_i), // 4
        .in01_i(in101_i), // 5
        .in10_i(in110_i), // 6
        .in11_i(in111_i), // 7
        .sel_2bit_i(sel_3bit_i[1:0]),
        .result_2x4mux_o(four_seven_result)
    );

    assign result_3x8mux_o = zero_three_result & {32{~sel_3bit_i[2]}} |
four_seven_result & {32{sel_3bit_i[2]}};

endmodule
```