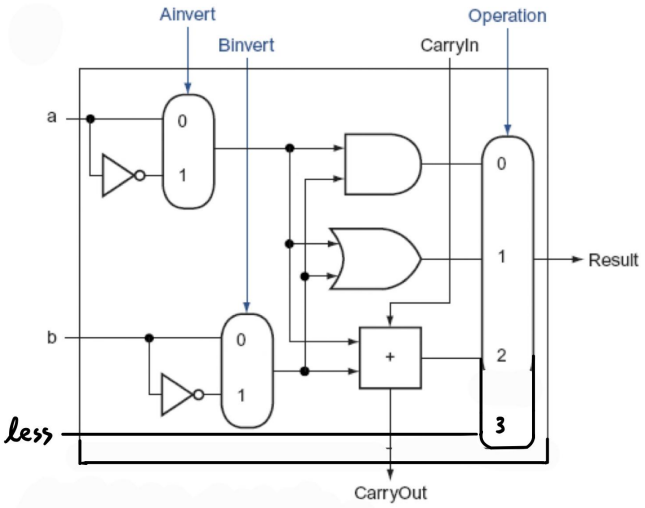
109550073 陳宥安

Computer Organization Report

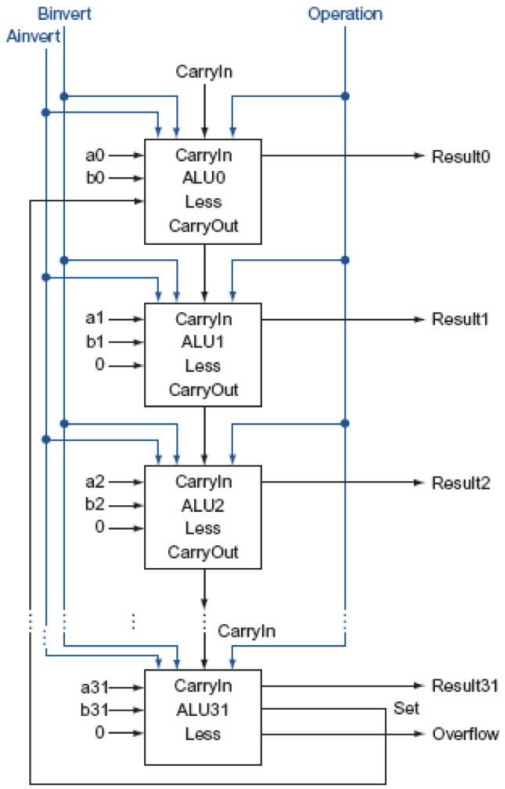
Architecture diagrams:

Goal: Implement a 32-bits ALU with several 1-bit ALU

alu_top.v: 1-bit ALU.



alu.v: 32-bit ALU (Constructed by 32 1-bit ALU).



109550073 陳宥安

Hardware module analysis:

(a) alu_top.v →1-bit ALU

```
module alu_top(
                src1,       //1 bit source 1 (input)
                src2,       //1 bit source 2 (input)
                less,       //1 bit less     (input)
                A_invert,   //1 bit A_invert (input)
                B_invert,   //1 bit B_invert (input)
                cin,        //1 bit carry in (input)
                operation,  //operation      (input)
                result,     //1 bit result   (output)
                cout       //1 bit carry out(output)
                );

input         src1;
input         src2;
input         less;
input         A_invert;
input         B_invert;
input         cin;
input [2-1:0] operation;
output        result;
output        cout;
reg           result;
wire          temp_a, temp_b;
```

```
assign temp_a = A_invert ^ src1;    //MUX1
assign temp_b = B_invert ^ src2;    //MUX2
assign cout = (temp_a&temp_b) | (cin&temp_a) | (cin&temp_b);
always@(*)
begin
    case(operation)
    2'b00:
        result= temp_a & temp_b;
    2'b01:
        result= temp_a | temp_b;
    2'b10:
        result= temp_a ^ temp_b ^ cin;
    2'b11:
        result=less;
    endcase
end


endmodule
```

temp_a and temp_b are results from two MUX, they can be implemented by XOR operation. cout represents carry out bit. Since it is from a full adder, we can use the equation for the carryout of the full adder((a|b)&(a|c)&(b|c)). The case expression deals with different operation codes: 00 for AND, 01 for OR, 10 for ADD, and 11 for Set Less Then.

(b) alu.v → 32-bit ALU

```
module alu(
    clk, // system clock              (input)
    rst_n, // negative reset           (input)
    src1, // 32 bits source 1         (input)
    src2, // 32 bits source 2         (input)
    ALU_control, // 4 bits ALU control input  (input)
    result, // 32 bits result           (output)
    zero, // 1 bit when the output is 0, zero must be set (output)
    cout, // 1 bit carry out          (output)
    overflow // 1 bit overflow           (output)
);

    input          clk;
    input          rst_n;
    input  [32-1:0] src1;
    input  [32-1:0] src2;
    input   [4-1:0] ALU_control;

    output [32-1:0] result;
    output         zero;
    output         cout;
    output         overflow;

    reg    [32-1:0] result;
    reg            zero;
    reg            cout;
    reg            overflow;

    wire   [31:0]   temp_result;
    wire   [31:0]   temp_cout;
    wire           less;
    wire           temp_overflow;
```

32 1-bit ALU is declared as follow:

```
alu_top a0( .src1(src1[0]),   .src2(src2[0]),   .less(less),  .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(ALU_control[2]), .operation(ALU_control[1:0]), .result(temp_result[0]),   .cout(temp_cout[0]));
alu_top a1( .src1(src1[1]),   .src2(src2[1]),   .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[0]),   .operation(ALU_control[1:0]), .result(temp_result[1]),   .cout(temp_cout[1]));
alu_top a2( .src1(src1[2]),   .src2(src2[2]),   .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[1]),   .operation(ALU_control[1:0]), .result(temp_result[2]),   .cout(temp_cout[2]));
alu_top a3( .src1(src1[3]),   .src2(src2[3]),   .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[2]),   .operation(ALU_control[1:0]), .result(temp_result[3]),   .cout(temp_cout[3]));
alu_top a4( .src1(src1[4]),   .src2(src2[4]),   .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[3]),   .operation(ALU_control[1:0]), .result(temp_result[4]),   .cout(temp_cout[4]));
alu_top a5( .src1(src1[5]),   .src2(src2[5]),   .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[4]),   .operation(ALU_control[1:0]), .result(temp_result[5]),   .cout(temp_cout[5]));
alu_top a6( .src1(src1[6]),   .src2(src2[6]),   .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[5]),   .operation(ALU_control[1:0]), .result(temp_result[6]),   .cout(temp_cout[6]));
alu_top a7( .src1(src1[7]),   .src2(src2[7]),   .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[6]),   .operation(ALU_control[1:0]), .result(temp_result[7]),   .cout(temp_cout[7]));
alu_top a8( .src1(src1[8]),   .src2(src2[8]),   .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[7]),   .operation(ALU_control[1:0]), .result(temp_result[8]),   .cout(temp_cout[8]));
alu_top a9( .src1(src1[9]),   .src2(src2[9]),   .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[8]),   .operation(ALU_control[1:0]), .result(temp_result[9]),   .cout(temp_cout[9]));
alu_top a10(.src1(src1[10]),  .src2(src2[10]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[9]),   .operation(ALU_control[1:0]), .result(temp_result[10]),  .cout(temp_cout[10]));
alu_top a11(.src1(src1[11]),  .src2(src2[11]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[10]),  .operation(ALU_control[1:0]), .result(temp_result[11]),  .cout(temp_cout[11]));
alu_top a12(.src1(src1[12]),  .src2(src2[12]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[11]),  .operation(ALU_control[1:0]), .result(temp_result[12]),  .cout(temp_cout[12]));
alu_top a13(.src1(src1[13]),  .src2(src2[13]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[12]),  .operation(ALU_control[1:0]), .result(temp_result[13]),  .cout(temp_cout[13]));
alu_top a14(.src1(src1[14]),  .src2(src2[14]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[13]),  .operation(ALU_control[1:0]), .result(temp_result[14]),  .cout(temp_cout[14]));
alu_top a15(.src1(src1[15]),  .src2(src2[15]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[14]),  .operation(ALU_control[1:0]), .result(temp_result[15]),  .cout(temp_cout[15]));
alu_top a16(.src1(src1[16]),  .src2(src2[16]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[15]),  .operation(ALU_control[1:0]), .result(temp_result[16]),  .cout(temp_cout[16]));
alu_top a17(.src1(src1[17]),  .src2(src2[17]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[16]),  .operation(ALU_control[1:0]), .result(temp_result[17]),  .cout(temp_cout[17]));
alu_top a18(.src1(src1[18]),  .src2(src2[18]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[17]),  .operation(ALU_control[1:0]), .result(temp_result[18]),  .cout(temp_cout[18]));
alu_top a19(.src1(src1[19]),  .src2(src2[19]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[18]),  .operation(ALU_control[1:0]), .result(temp_result[19]),  .cout(temp_cout[19]));
alu_top a20(.src1(src1[20]),  .src2(src2[20]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[19]),  .operation(ALU_control[1:0]), .result(temp_result[20]),  .cout(temp_cout[20]));
alu_top a21(.src1(src1[21]),  .src2(src2[21]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[20]),  .operation(ALU_control[1:0]), .result(temp_result[21]),  .cout(temp_cout[21]));
alu_top a22(.src1(src1[22]),  .src2(src2[22]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[21]),  .operation(ALU_control[1:0]), .result(temp_result[22]),  .cout(temp_cout[22]));
alu_top a23(.src1(src1[23]),  .src2(src2[23]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[22]),  .operation(ALU_control[1:0]), .result(temp_result[23]),  .cout(temp_cout[23]));
alu_top a24(.src1(src1[24]),  .src2(src2[24]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[23]),  .operation(ALU_control[1:0]), .result(temp_result[24]),  .cout(temp_cout[24]));
alu_top a25(.src1(src1[25]),  .src2(src2[25]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[24]),  .operation(ALU_control[1:0]), .result(temp_result[25]),  .cout(temp_cout[25]));
alu_top a26(.src1(src1[26]),  .src2(src2[26]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[25]),  .operation(ALU_control[1:0]), .result(temp_result[26]),  .cout(temp_cout[26]));
alu_top a27(.src1(src1[27]),  .src2(src2[27]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[26]),  .operation(ALU_control[1:0]), .result(temp_result[27]),  .cout(temp_cout[27]));
alu_top a28(.src1(src1[28]),  .src2(src2[28]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[27]),  .operation(ALU_control[1:0]), .result(temp_result[28]),  .cout(temp_cout[28]));
alu_top a29(.src1(src1[29]),  .src2(src2[29]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[28]),  .operation(ALU_control[1:0]), .result(temp_result[29]),  .cout(temp_cout[29]));
alu_top a30(.src1(src1[30]),  .src2(src2[30]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[29]),  .operation(ALU_control[1:0]), .result(temp_result[30]),  .cout(temp_cout[30]));
alu_top a31(.src1(src1[31]),  .src2(src2[31]),  .less(0),     .A_invert(ALU_control[3]), .B_invert(ALU_control[2]), .cin(temp_cout[30]),  .operation(ALU_control[1:0]), .result(temp_result[31]),  .cout(temp_cout[31]));
```

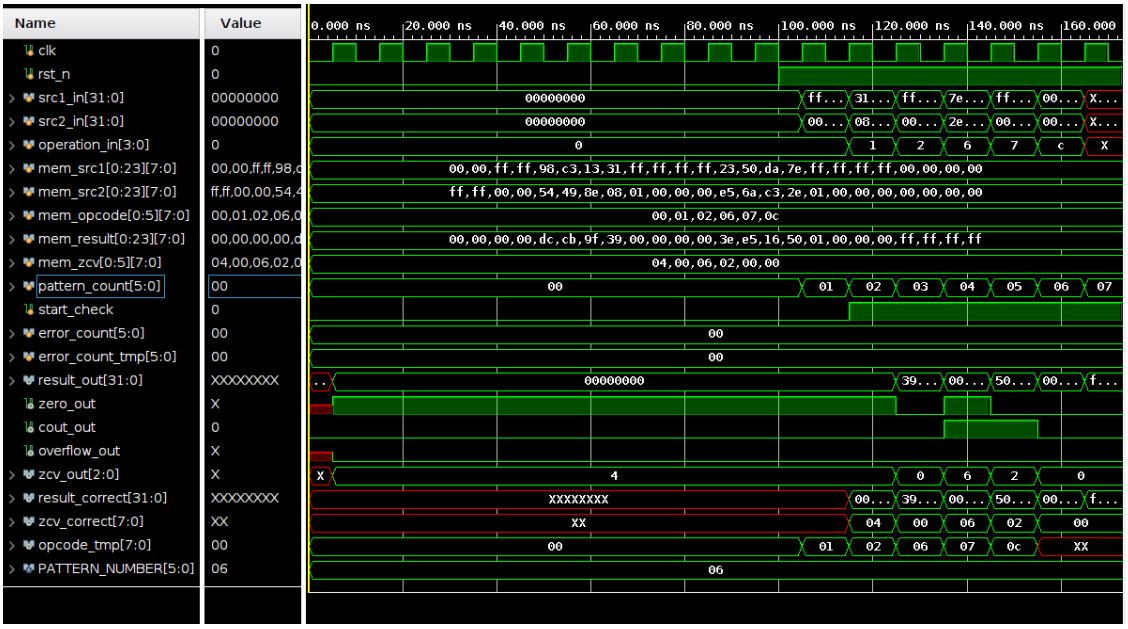
```
    assign temp_overflow = temp_cout[30]^temp_cout[31];
    assign less = (src1[31]^(~src2[31])^temp_cout[30]);

    always@( posedge clk or negedge rst_n )
    begin
        result <= temp_result;
        zero <= temp_result == 0 ? 1 : 0;
        cout <= (ALU_control[1:0] != 2'b10) ? 0 : temp_cout[31];
        overflow <= temp_overflow;
    end

endmodule
```

result is obtained with 32 times of 1-bit ALU operation. temp_overflow is calculated by the temp_cout[30] and temp_cout[31](carry out of NO.31 and 32 bit). Less calculated by src[31]-src2[31], so it can be calculated as above. cout needs to be noticed only when there is ADD operation for 1-bit ALU. zero is judged by whether temp_result is 0 or not.

Experiment result:




```
****************************************************
 Congratulation! All data are correct!
****************************************************
$stop called at time : 175 ns : File "/home/iammrchen/Desktop/CO_LAB1/testbench.v" Line 104
INFO: [USF-XSim-96] XSim completed. Design snapshot 'testbench_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:06 ; elapsed = 00:00:07 . Memory (MB): peak = 8003.516 ; gain = 0.000 ; free physical = 222 ; free virtual = 16986
```

Problems you met and solutions:

When I wrote cout <= temp_cout in alu.v, I got two wrong answers. After checking the course's slides again, I realized that only 10(ADD) operation needs to consider cout. On the other hand, ADD operation was written as “a+b” in alu_top.v in the beginning, and I found this error when I was checking my code. It turns out that overflow can cause

problems.

Summary:

It’s a great opportunity to implement ALU with verilog by ourselves. After this lab, I rainfore more concepts about ALU and correct those errors. Hope that I can get a full score for this lab QAQ.