Question 1:

Verilog description for ALU:

`timescale 1ns/1ns

module ALU(input signed [15:0] inA, inB, output reg [15:0] outW, input inC, input [2:0] opc, output reg zer, neg);

    always @ (inA, inB, opc) begin

        outW = 16'b0;

        zer = 1'b0;

        neg = 1'b0;

        case (opc)

            3'b000: outW = ~inA + 1;

            3'b001: outW = inA + 1;

            3'b010: outW = inA + inB + inC;

            3'b011: outW = inA + (inB >>> 1);

            3'b100: outW = inA & inB;

            3'b101: outW = inA | inB;

            3'b110: outW = {inA[7:0], inB[7:0]};

            default: outW = 16'b0;

        endcase

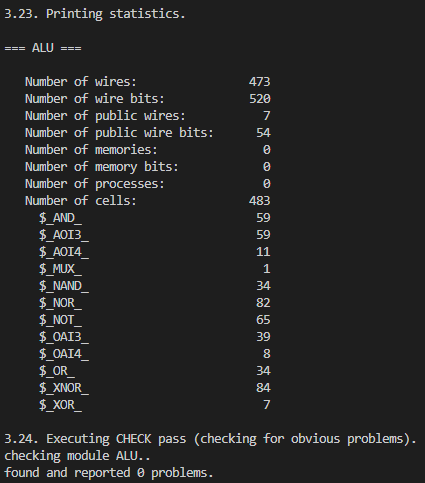
        zer = outW == 16'b0 ? 1'b1: 1'b0;

        neg = outW[15] == 1'b1 ? 1'b1: 1'b0;

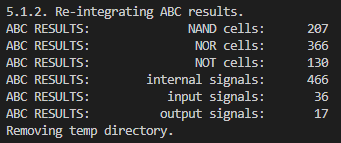
    end

endmodule

Then we synthesized the code using Yosys default cells, and the result is in below:



And then we used our own cells lib for our circuit.



Finally we wrote the Verilog description using Verilog and write a testbench for them.

`timescale 1ns/1ns

module tb();

    reg [15:0] aa, bb;

    reg [2:0] opc;

    reg cc;

    wire [15:0] ww, ww\_y;

    wire zer, neg, zer\_y, neg\_y;

    ALU alu(.inA(aa), .inB(bb), .inC(cc), .outW(ww), .opc(opc), .zer(zer), .neg(neg));

    ALU\_synth alu\_y(.inA(aa), .inB(bb), .inC(cc), .outW(ww\_y), .opc(opc), .zer(zer\_y), .neg(neg\_y));

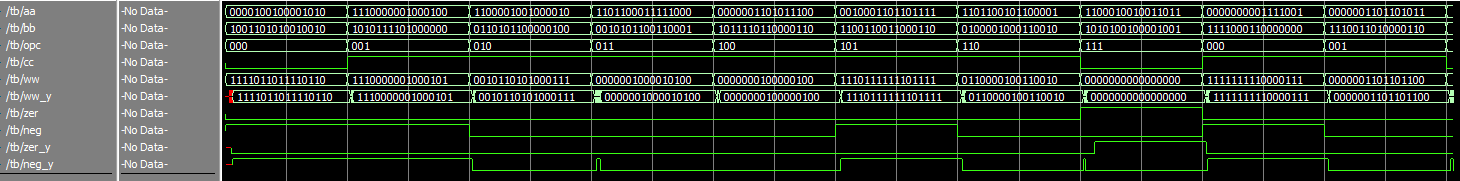
    initial opc = 3'b0;

    initial {aa, bb, cc} = $random;

    initial repeat (10) #1000 {aa, bb, cc} = $random;

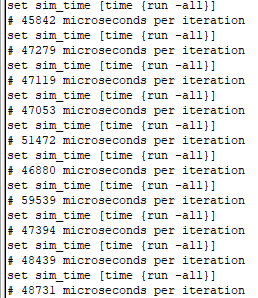
    initial repeat (10) #1000 opc = opc + 3'b001;

endmodule

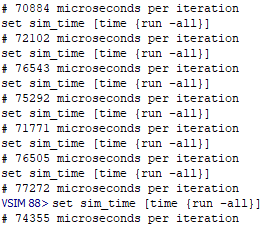
Here’s the wayform. 

We used this command “set sim\_time [time {run -all}]” at modelsim and compared the simulation speed of the two descriptions.

Our description:

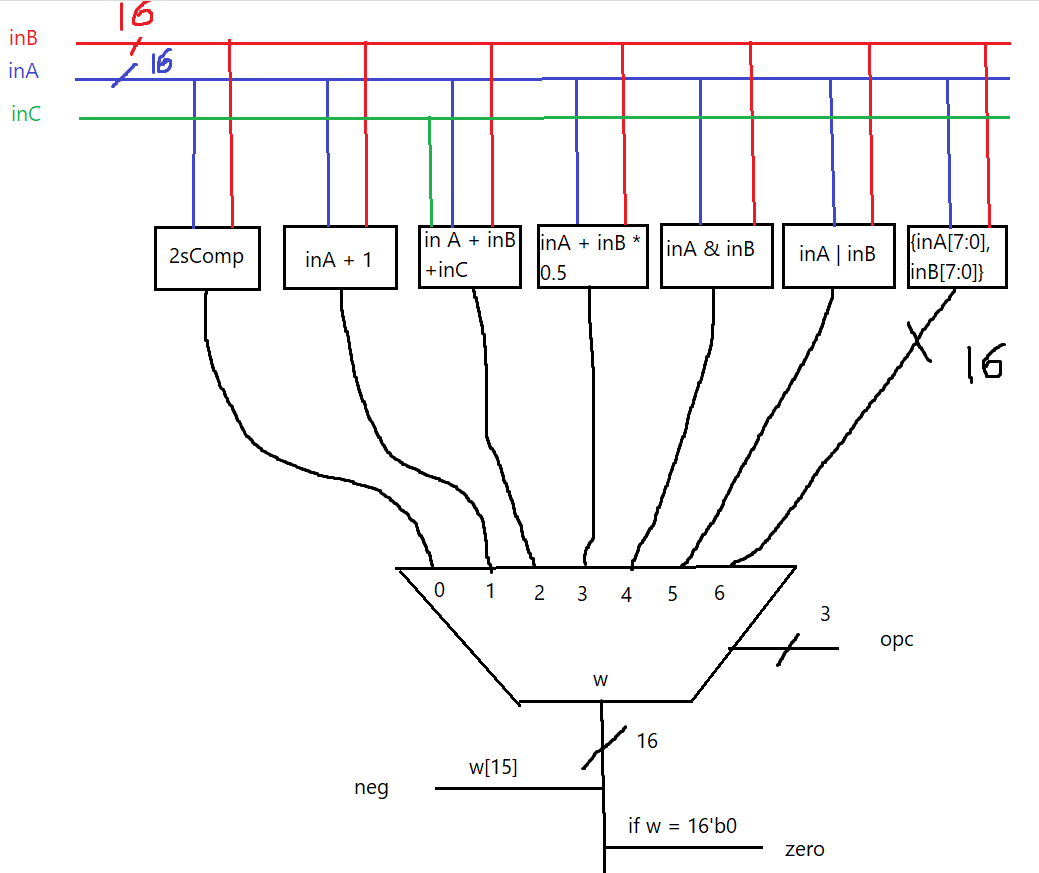


Synthesized description:



Question 2:

Struct level design:



We wrote a new description using assign statement.

`timescale 1ns/1ns

module ALU(input signed [15:0] inA, inB, output reg [15:0] outW, input inC, input [2:0] opc, output reg zer, neg);

    wire [15:0] out [7:0];

    assign out[0] = ~inA + 1;

    assign out[1] = inA + 1;

    assign out[2] = inA + inB + inC;

    assign out[3] = inA + (inB >>> 1);

    assign out[4] = inA & inB;

    assign out[5] = inA | inB;

    assign out[6] = {inA[7:0], inB[7:0]};

    assign outW = opc == 3'b000 ? out[0]:

    opc == 3'b001 ? out[1]:

    opc == 3'b010 ? out[2]:

    opc == 3'b011 ? out[3]:

    opc == 3'b100 ? out[4]:

    opc == 3'b101 ? out[5]:

    opc == 3'b110 ? out[6]:

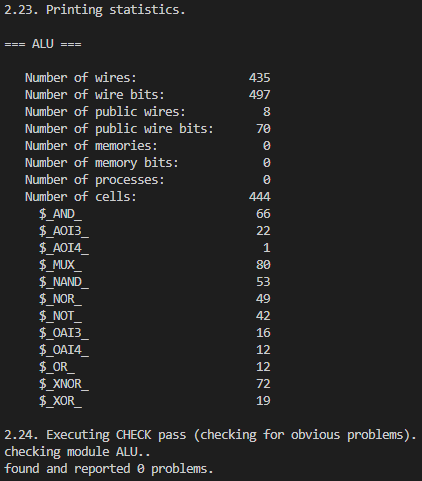
    3'b0;

    assign zer = outW == 16'b0 ? 1 : 0;

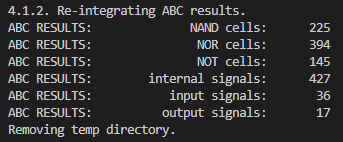
    assign neg = w[15];

endmodule

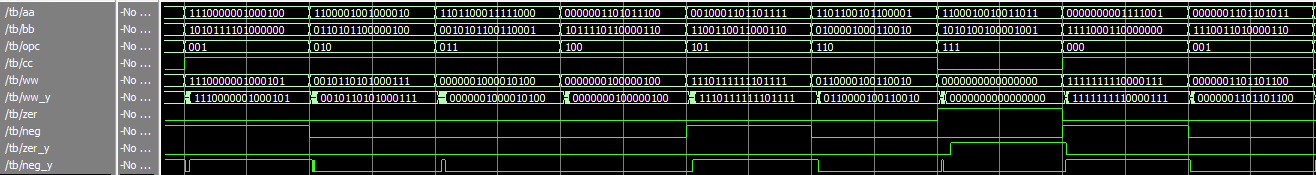
Like question 1 we will start to synthesize it.



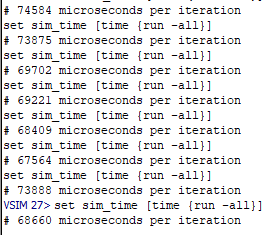
Synthesized with our own cells lib.



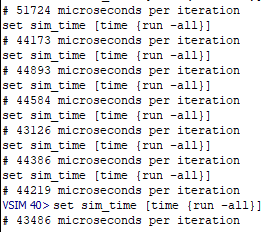
We use the same testbench we used in question 1 and here’s the waveform:



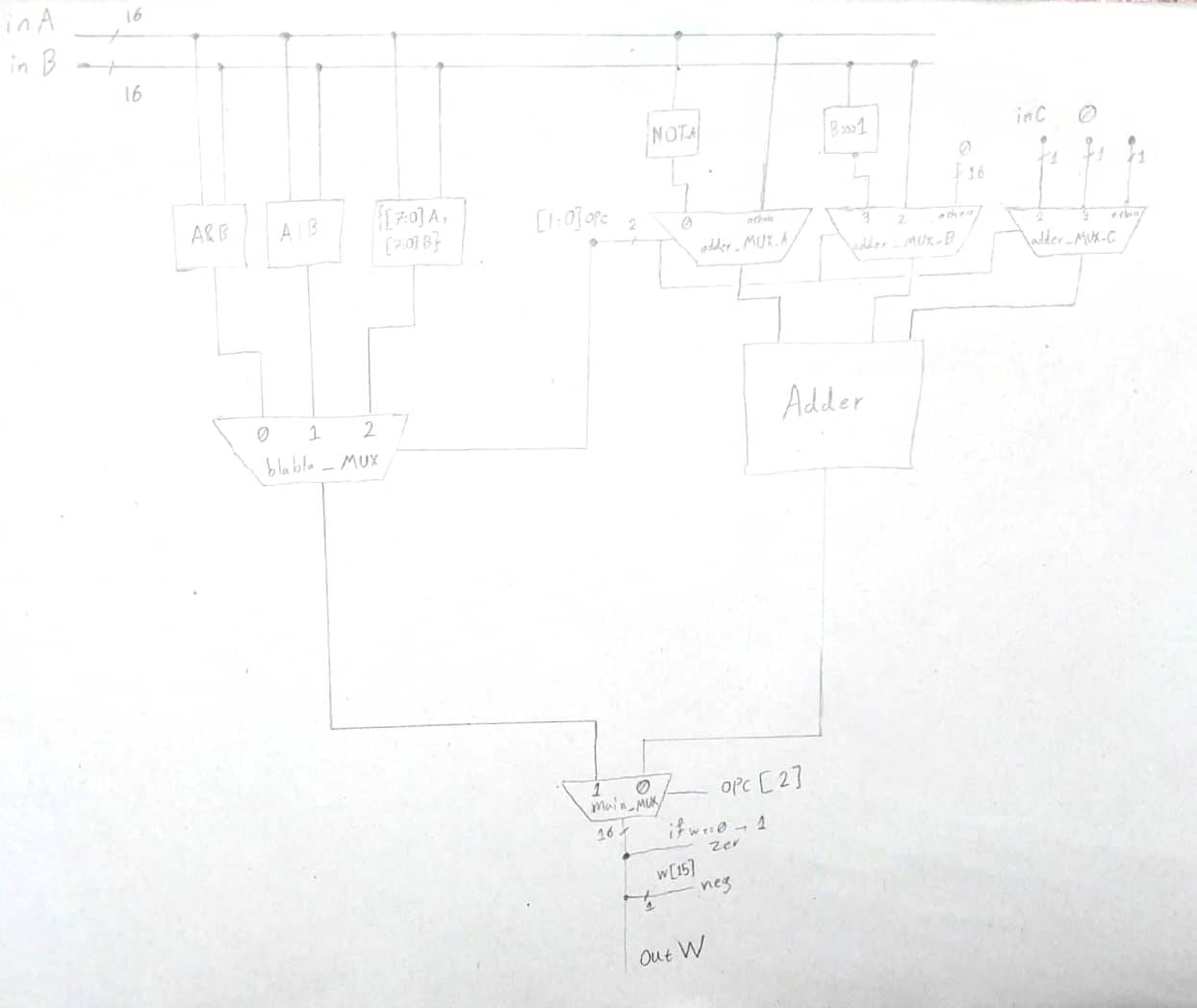
Now we’ll compare simulation speed between our description and synthesized one.

Our description:  


Synthesized description:



Now we have designed another circuit which use less cells than the others. Here’s the design:



And here’s the description of it:

`timescale 1ns/1ns

module ALU\_min(input signed [15:0] inA, inB, output reg [15:0] outW, input inC, input [2:0] opc, output reg zer, neg);

    wire [15:0] blabla\_MUX, adder\_MUX\_A, adder\_MUX\_B, Adder, A\_and\_B, A\_or\_B, merged\_A\_B, NOT\_A, Shifted\_B;

    wire adder\_MUX\_C;

    assign NOT\_A = ~inA;

    assign Shifted\_B = inB >>> 1;

    assign A\_and\_B = inA & inB;

    assign A\_or\_B = inA | inB;

    assign merged\_A\_B = {inA[7:0], inB[7:0]};

    assign blabla\_MUX = (opc[1:0] == 2'b00) ? A\_and\_B:

    (opc[1:0] == 2'b01) ? A\_or\_B:

    (opc[1:0] == 2'b10) ? merged\_A\_B:

    16'b0;

    assign adder\_MUX\_A = (opc[1:0] == 2'b00) ? NOT\_A:

    inA;

    assign adder\_MUX\_B = (opc[1:0] == 2'b11) ? Shifted\_B:

    (opc[1:0] == 2'b10) ? inB:

    16'b0;

    assign adder\_MUX\_C = (opc[1:0] == 2'b10) ? inC:

    (opc[1:0] == 2'b11) ? 1'b0:

    1'b1;

    assign Adder = adder\_MUX\_A + adder\_MUX\_B + adder\_MUX\_C;

    assign outW = (opc[2] == 1'b0) ? Adder:

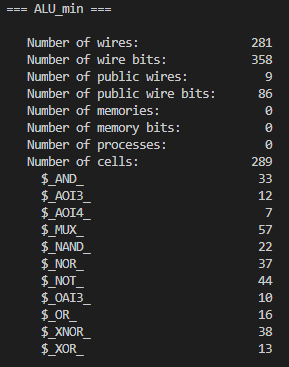
    blabla\_MUX;

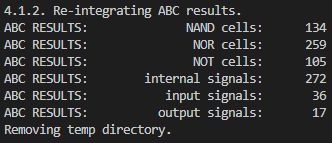
    assign zer = outW == 16'b0 ? 1 : 0;

    assign neg = outW[15];

endmodule

We synthesized it, and here’s the results:

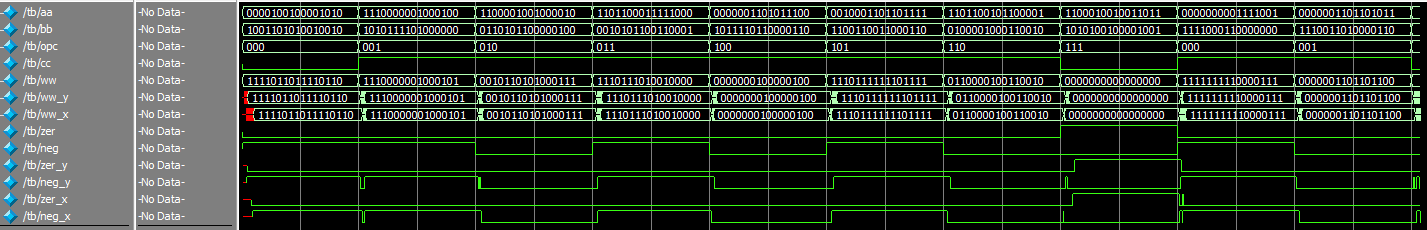




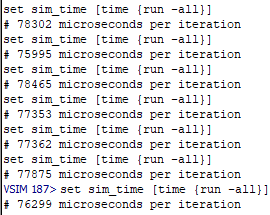
ww = ALU\_min

ww\_y = ALU\_synth

ww\_x = ALU\_min\_synth



Our description:



Synthesized one:

