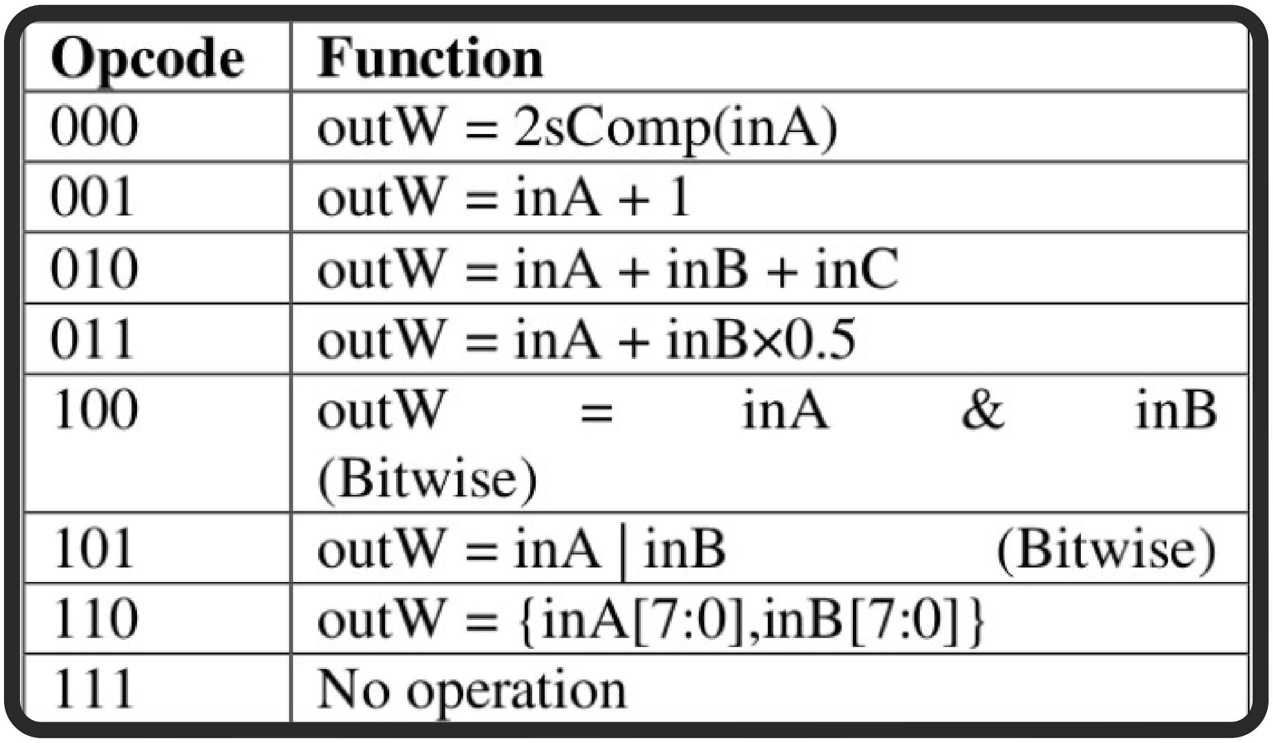
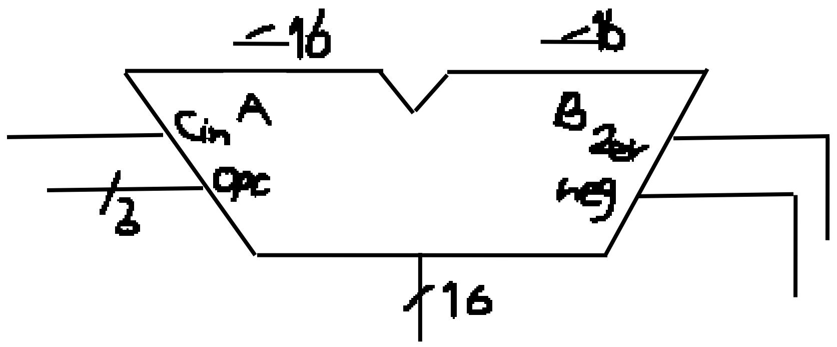
Q1:

In this question, we should make ALU using the table below and always statement.



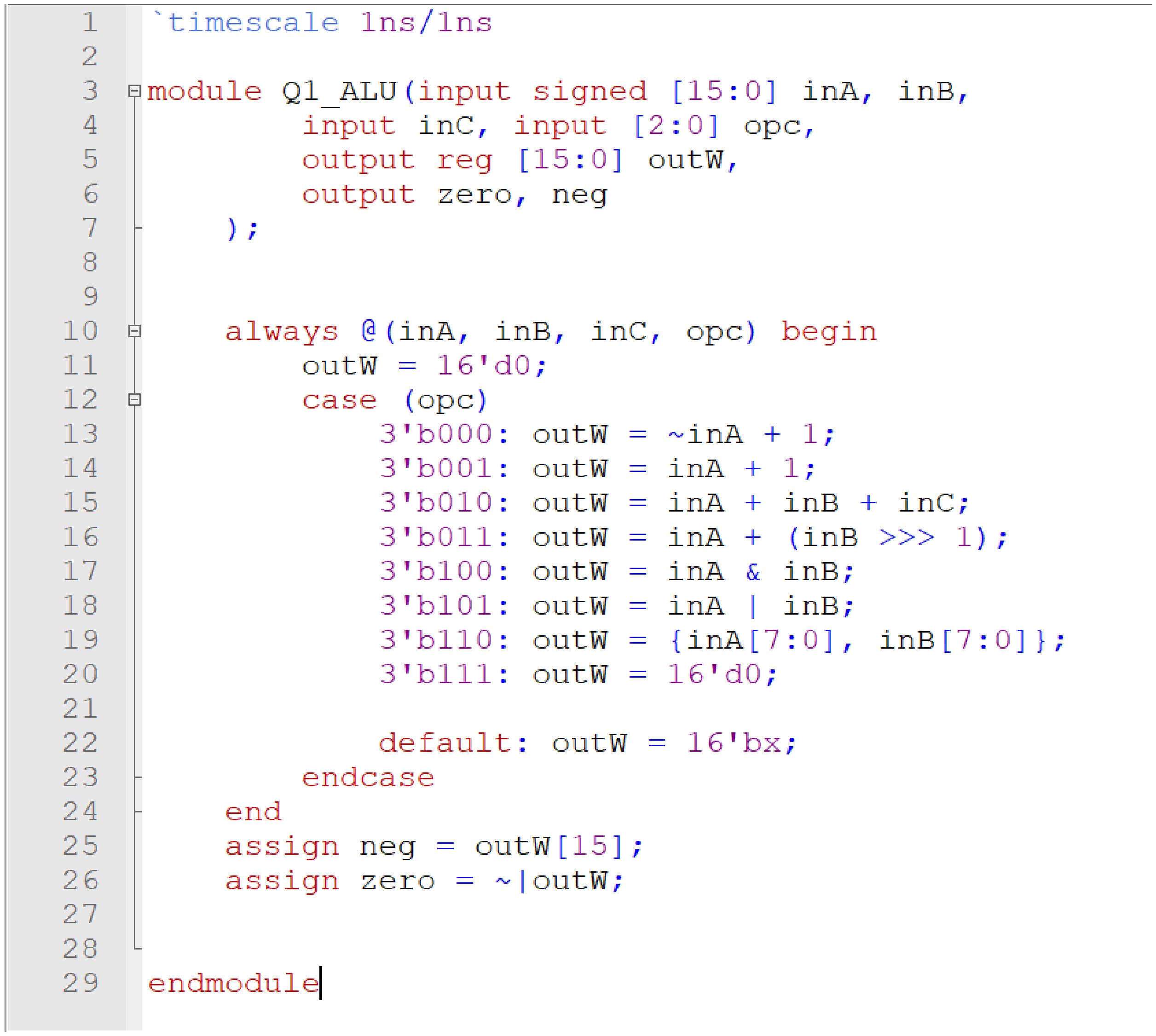
Opcode	Function
000	outW = 2sComp(inA)
001	outW = inA + 1
010	outW = inA + inB + inC
011	outW = inA + inB×0.5
100	outW = inA & inB (Bitwise)
101	outW = inA | inB (Bitwise)
110	outW = {inA[7:0],inB[7:0]}
111	No operation

First, we draw the circuit:



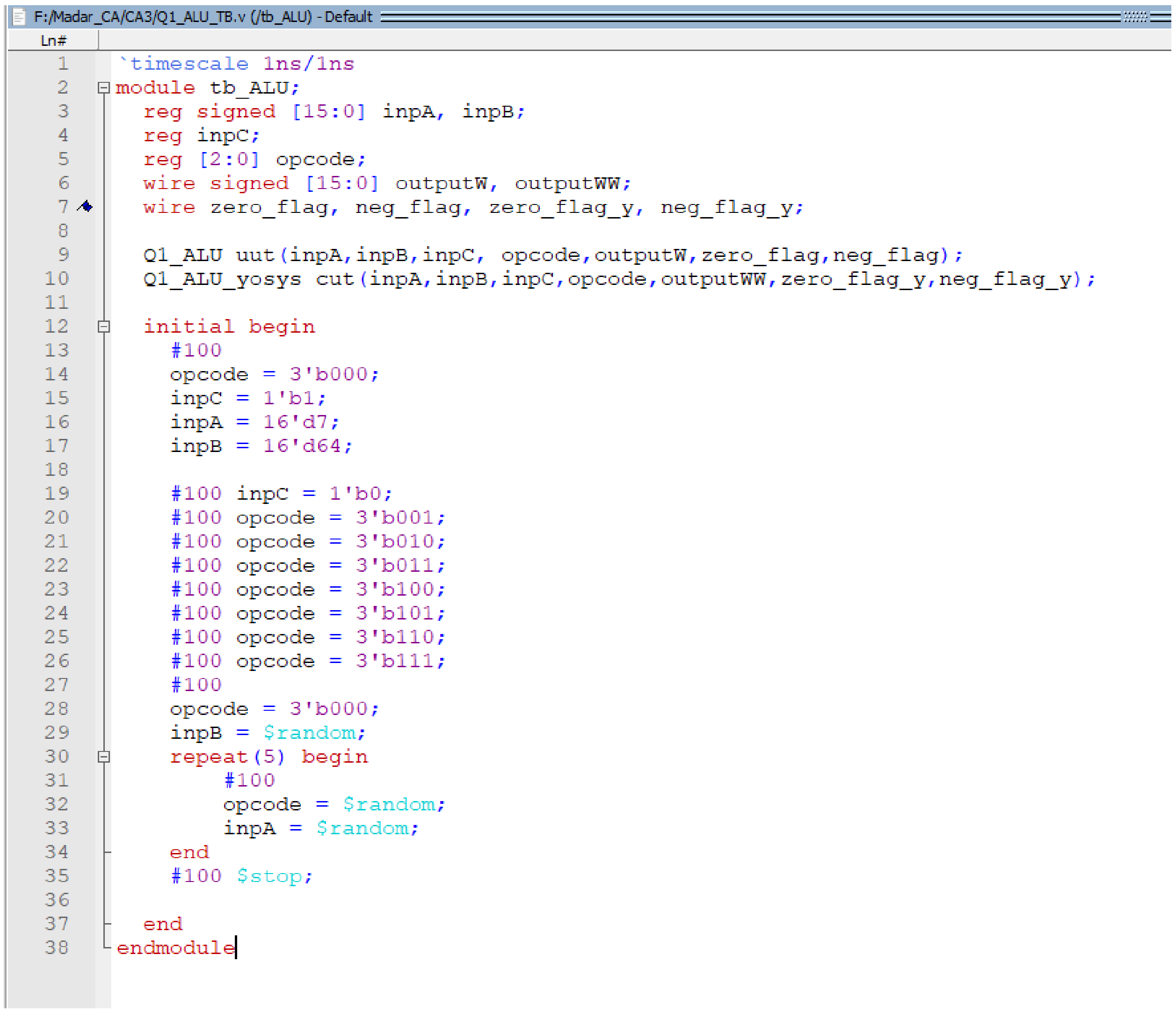
Part a:

Here is the code:



```verilog
`timescale 1ns/1ns

module Q1_ALU(input signed [15:0] inA, inB,
        input inC, input [2:0] opc,
        output reg [15:0] outW,
        output zero, neg
    );


    always @(inA, inB, inC, opc) begin
        outW = 16'd0;
        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]};
            3'b111: outW = 16'd0;

            default: outW = 16'bx;
        endcase
    end
    assign neg = outW[15];
    assign zero = ~|outW;


endmodule
```

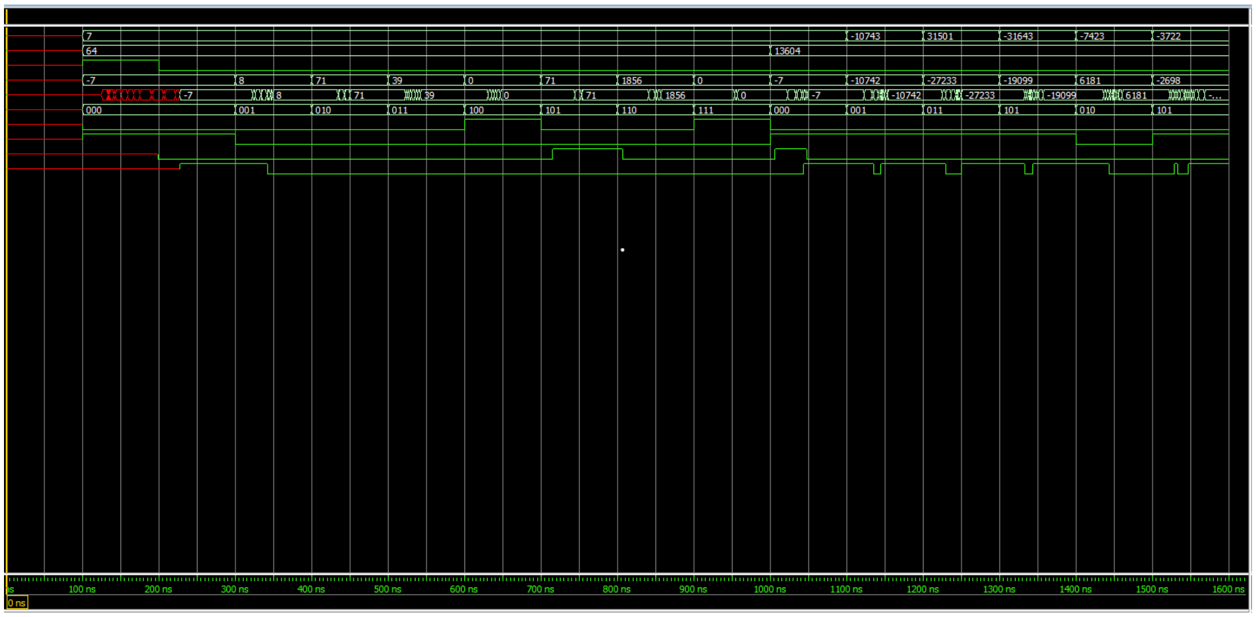
Now we have to use testbench to verify our circuit:



F:/Madar_CA/CA3/Q1_ALU_TB.v (/tb_ALU) - Default

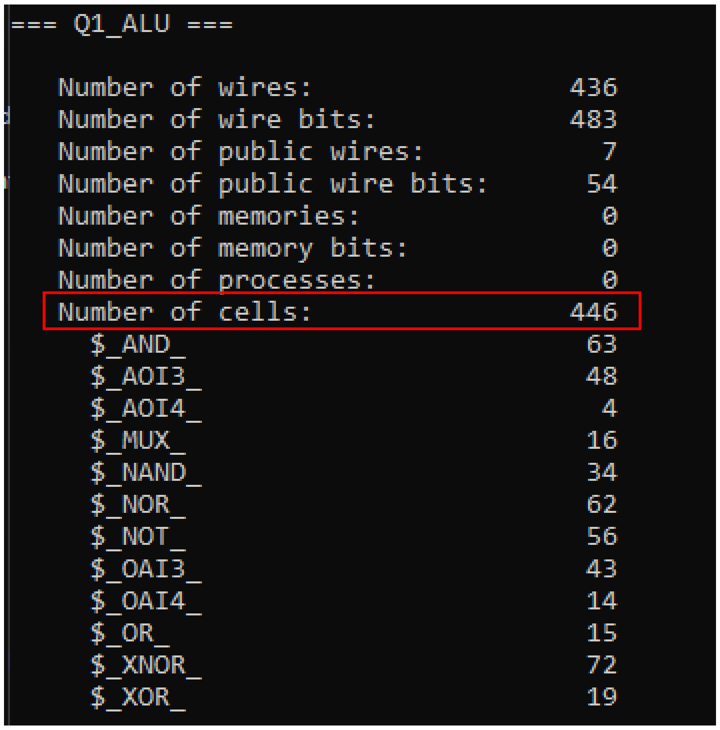
```verilog
`timescale 1ns/1ns
module tb_ALU;
  reg signed [15:0] inpA, inpB;
  reg inpC;
  reg [2:0] opcode;
  wire signed [15:0] outputW, outputWW;
  wire zero_flag, neg_flag, zero_flag_y, neg_flag_y;

  Q1_ALU uut(inpA,inpB,inpC, opcode,outputW,zero_flag,neg_flag);
  Q1_ALU_yosys cut(inpA,inpB,inpC,opcode,outputWW,zero_flag_y,neg_flag_y);

  initial begin
    #100
    opcode = 3'b000;
    inpC = 1'b1;
    inpA = 16'd7;
    inpB = 16'd64;

    #100 inpC = 1'b0;
    #100 opcode = 3'b001;
    #100 opcode = 3'b010;
    #100 opcode = 3'b011;
    #100 opcode = 3'b100;
    #100 opcode = 3'b101;
    #100 opcode = 3'b110;
    #100 opcode = 3'b111;
    #100
    opcode = 3'b000;
    inpB = $random;
    repeat(5) begin
        #100
        opcode = $random;
        inpA = $random;
    end
    #100 $stop;

  end
endmodule
```

And this is the timing digram and the results of it:



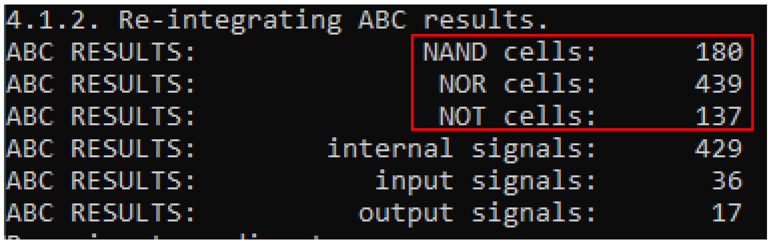
7
64
13604
-10743
31501
-31643
-7423
-3722
-7
8
71
39
0
71
1856
0
-7
-10742
-27233
-19099
6181
-2698
000
001
010
011
100
101
110
111
000
001
011
101
010
101
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1000 ns
1100 ns
1200 ns
1300 ns
1400 ns
1500 ns
1600 ns
0 ns

Part b:
Now we use Yosys to synthsis the code
(Results of synthesis in Yosys without our library.)
Number of cells is 446.



```
=== Q1_ALU ===

   Number of wires:                436
   Number of wire bits:            483
   Number of public wires:           7
   Number of public wire bits:      54
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                446
     $_AND_                         63
     $_AOI3_                        48
     $_AOI4_                         4
     $_MUX_                         16
     $_NAND_                        34
     $_NOR_                         62
     $_NOT_                         56
     $_OAI3_                        43
     $_OAI4_                        14
     $_OR_                          15
     $_XNOR_                        72
     $_XOR_                         19
```

Results of synthesis in Yosys with our library.



```
4.1.2. Re-integrating ABC results.
ABC RESULTS:                NAND cells:      180
ABC RESULTS:                 NOR cells:      439
ABC RESULTS:                 NOT cells:      137
ABC RESULTS:          internal signals:      429
ABC RESULTS:             input signals:       36
ABC RESULTS:            output signals:       17
```

Number of cells: 756

Part c: There is in the zip file. (the files of pre & post synthesis are called (in order) ->Q1_ALU.v, Q1_ALU_yosys)

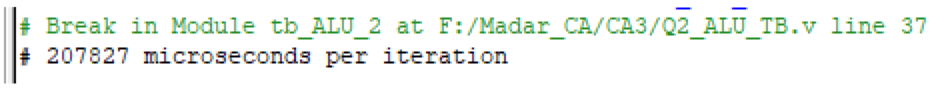
Part d:

We made ALU (in first question) by Verilog using software components and elements like ‘always’, case statement and…so we don’t know with which hardware components Verilog implements this circuit.

But in part c, Yosys simulates the hardware by the library provided. And some delays are considered in it. I guess that because part c ciruit is completely made with hardware and gates, it is slower than the part one circuit.

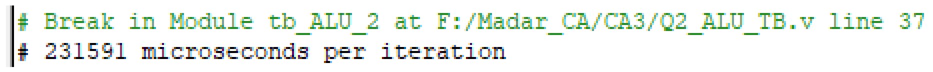
Now, we can calculate the times using command “time {run -all}”. We should first use “vsim” and then the command above.

Pre synthesis:



```
# Break in Module tb_ALU_2 at F:/Madar_CA/CA3/Q2_ALU_TB.v line 37
# 207827 microseconds per iteration
```

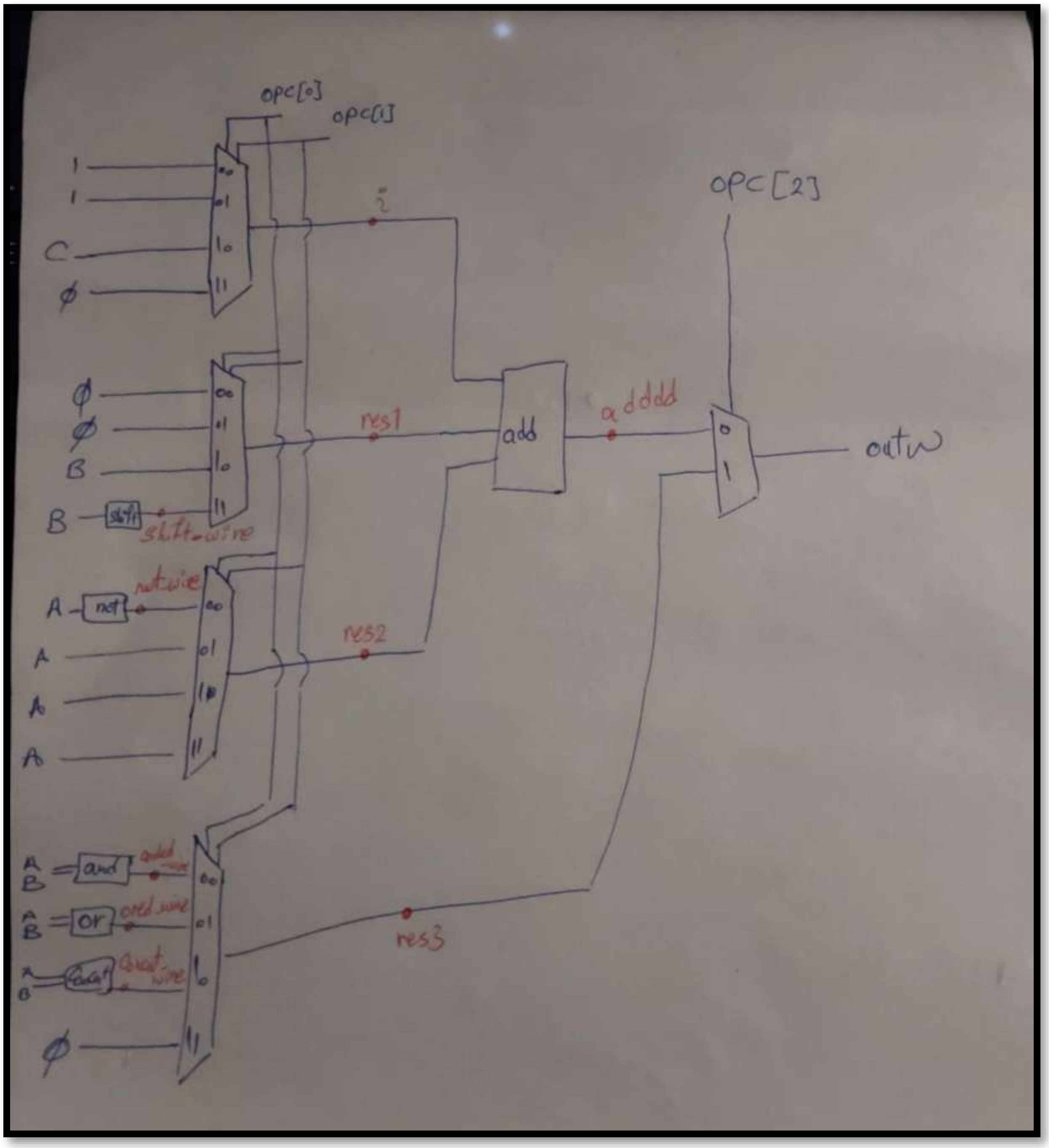
Post synthesis:



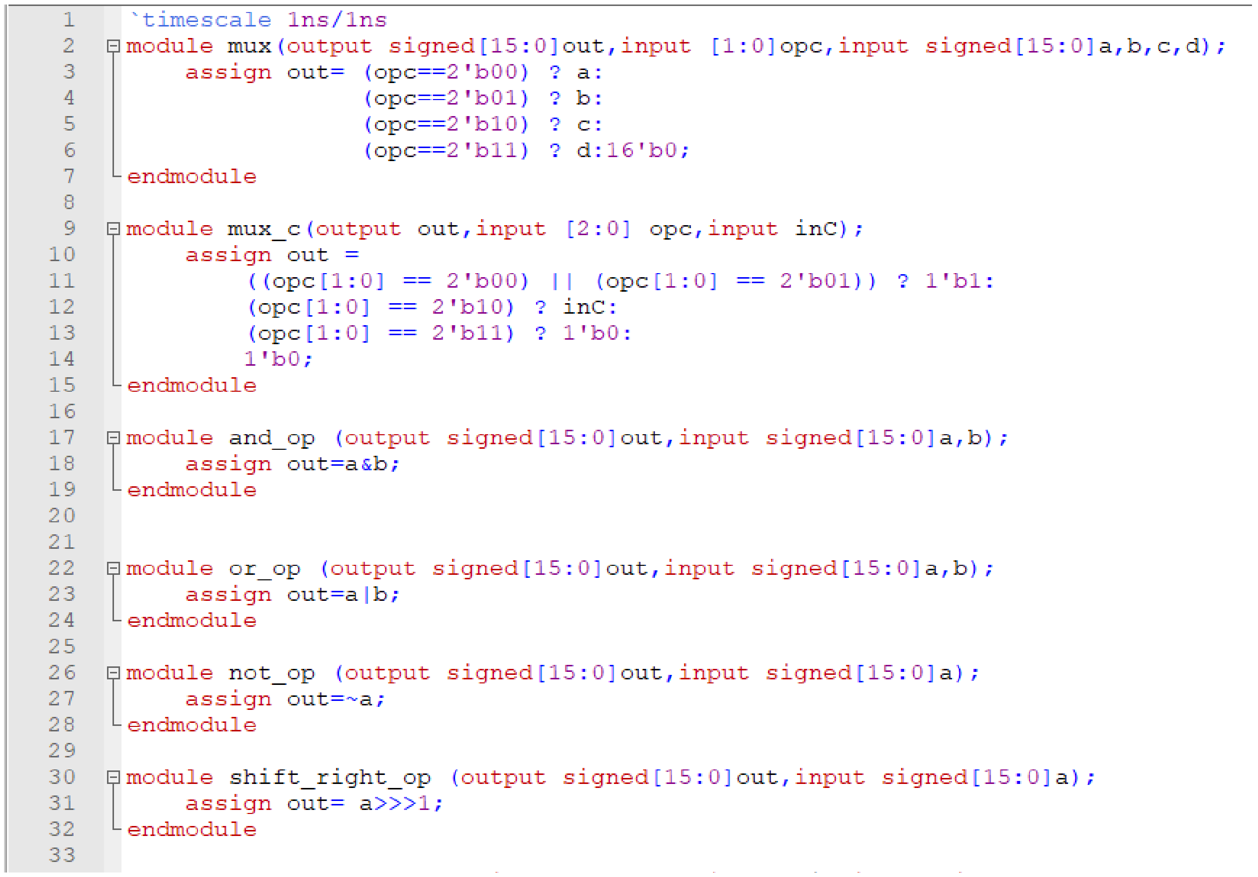
```
# Break in Module tb_ALU_2 at F:/Madar_CA/CA3/Q2_ALU_TB.v line 37
# 231591 microseconds per iteration
```

Q2:

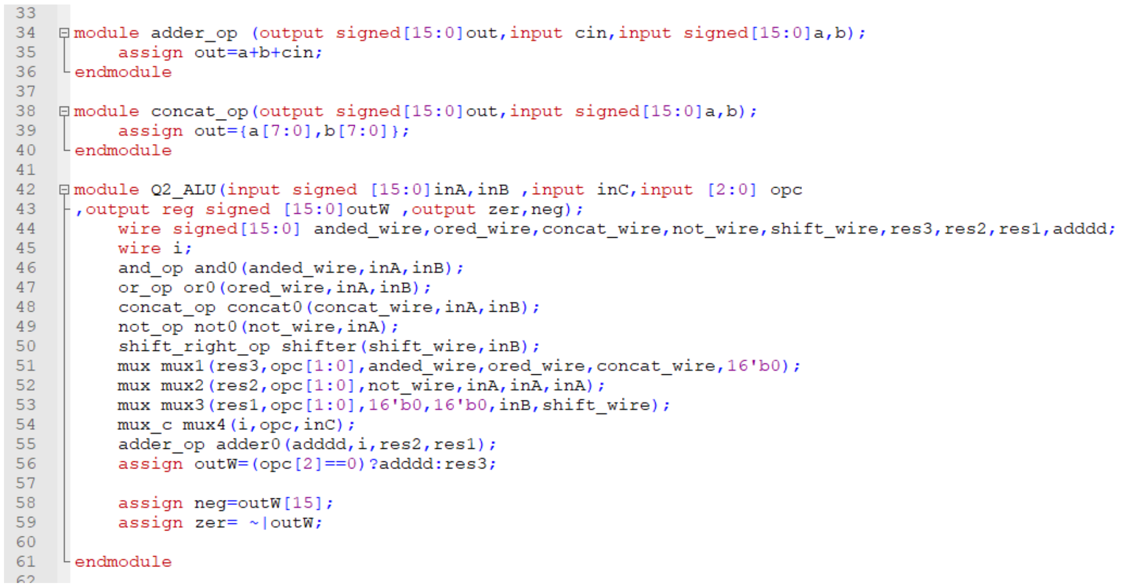
Here is the hardware of this question:



This is the code:

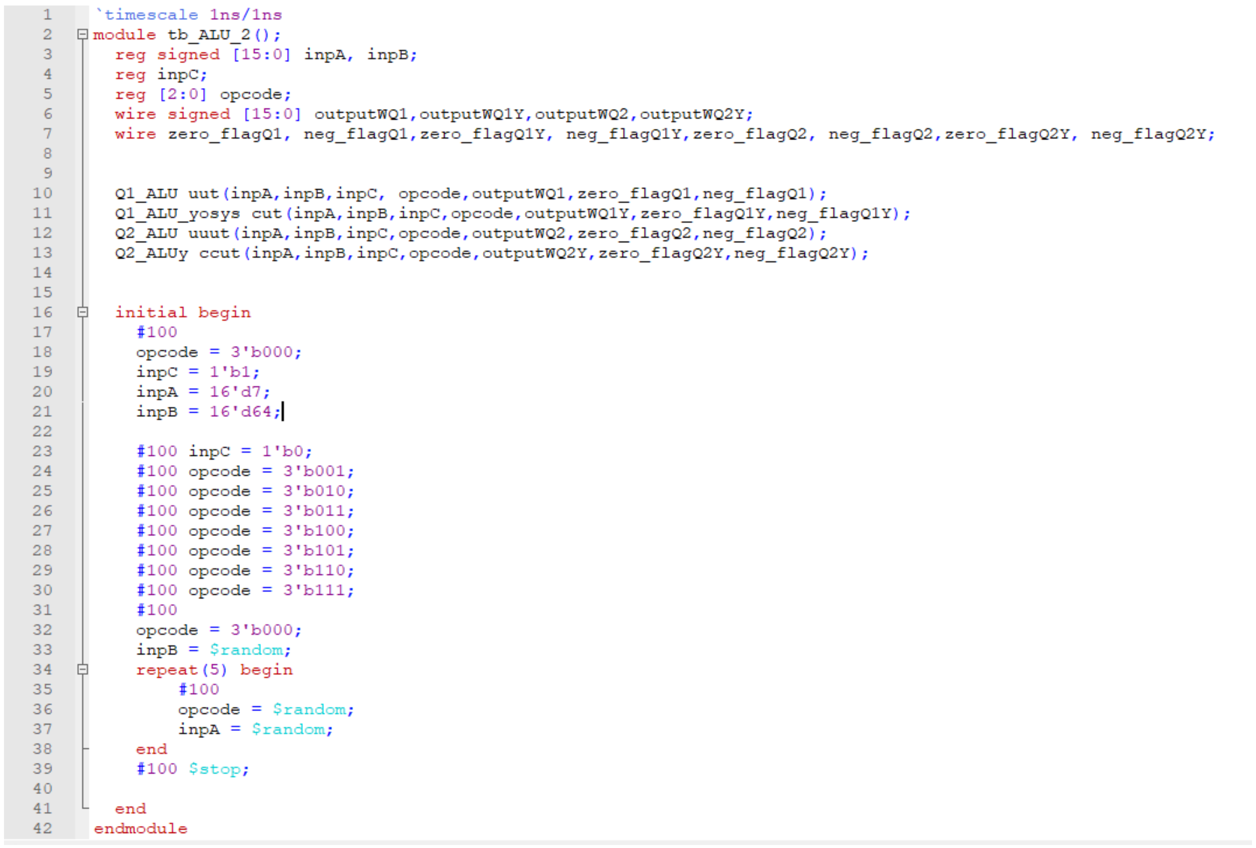


```
`timescale 1ns/1ns
module mux(output signed[15:0]out,input [1:0]opc,input signed[15:0]a,b,c,d);
    assign out= (opc==2'b00) ? a:
                (opc==2'b01) ? b:
                (opc==2'b10) ? c:
                (opc==2'b11) ? d:16'b0;
endmodule

module mux_c(output out,input [2:0] opc,input inC);
    assign out =
        ((opc[1:0] == 2'b00) || (opc[1:0] == 2'b01)) ? 1'b1:
        (opc[1:0] == 2'b10) ? inC:
        (opc[1:0] == 2'b11) ? 1'b0:
        1'b0;
endmodule

module and_op (output signed[15:0]out,input signed[15:0]a,b);
    assign out=a&b;
endmodule


module or_op (output signed[15:0]out,input signed[15:0]a,b);
    assign out=a|b;
endmodule

module not_op (output signed[15:0]out,input signed[15:0]a);
    assign out=~a;
endmodule

module shift_right_op (output signed[15:0]out,input signed[15:0]a);
    assign out= a>>>1;
endmodule

```



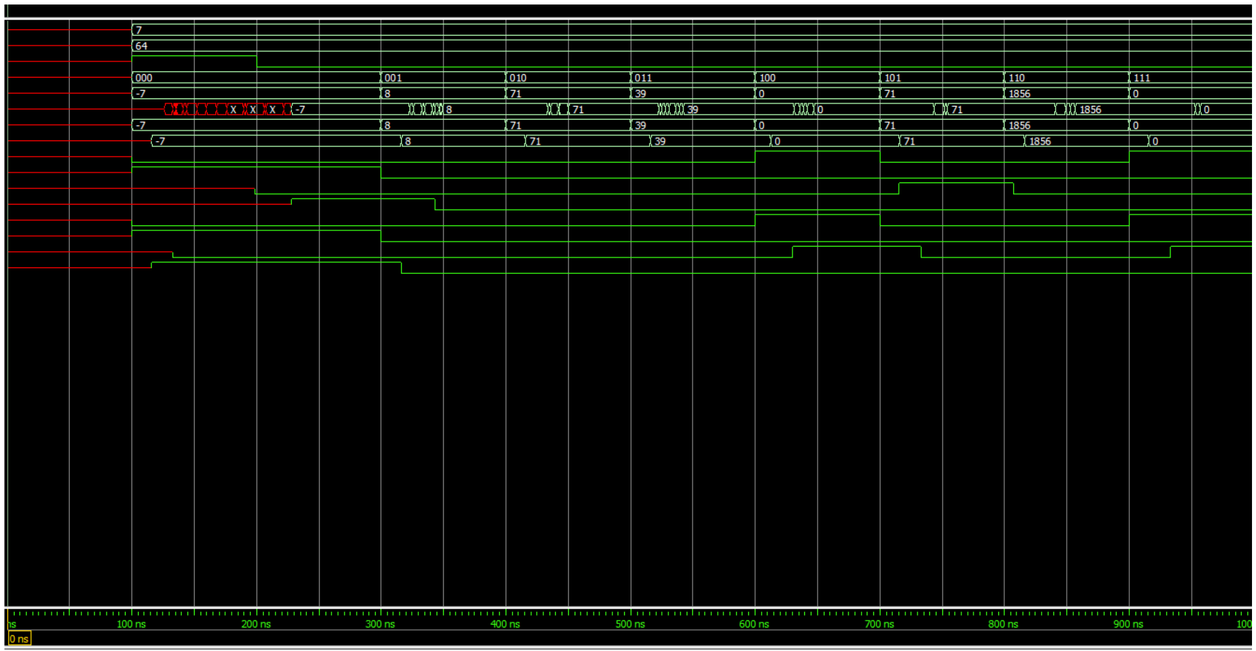
```

module adder_op (output signed[15:0]out,input cin,input signed[15:0]a,b);
    assign out=a+b+cin;
endmodule

module concat_op(output signed[15:0]out,input signed[15:0]a,b);
    assign out={a[7:0],b[7:0]};
endmodule

module Q2_ALU(input signed [15:0]inA,inB ,input inC,input [2:0] opc
,output reg signed [15:0]outW ,output zer,neg);
    wire signed[15:0] anded_wire,ored_wire,concat_wire,not_wire,shift_wire,res3,res2,res1,adddd;
    wire i;
    and_op and0(anded_wire,inA,inB);
    or_op or0(ored_wire,inA,inB);
    concat_op concat0(concat_wire,inA,inB);
    not_op not0(not_wire,inA);
    shift_right_op shifter(shift_wire,inB);
    mux mux1(res3,opc[1:0],anded_wire,ored_wire,concat_wire,16'b0);
    mux mux2(res2,opc[1:0],not_wire,inA,inA,inA);
    mux mux3(res1,opc[1:0],16'b0,16'b0,inB,shift_wire);
    mux_c mux4(i,opc,inC);
    adder_op adder0(adddd,i,res2,res1);
    assign outW=(opc[2]==0)?adddd:res3;

    assign neg=outW[15];
    assign zer= ~|outW;

endmodule
```

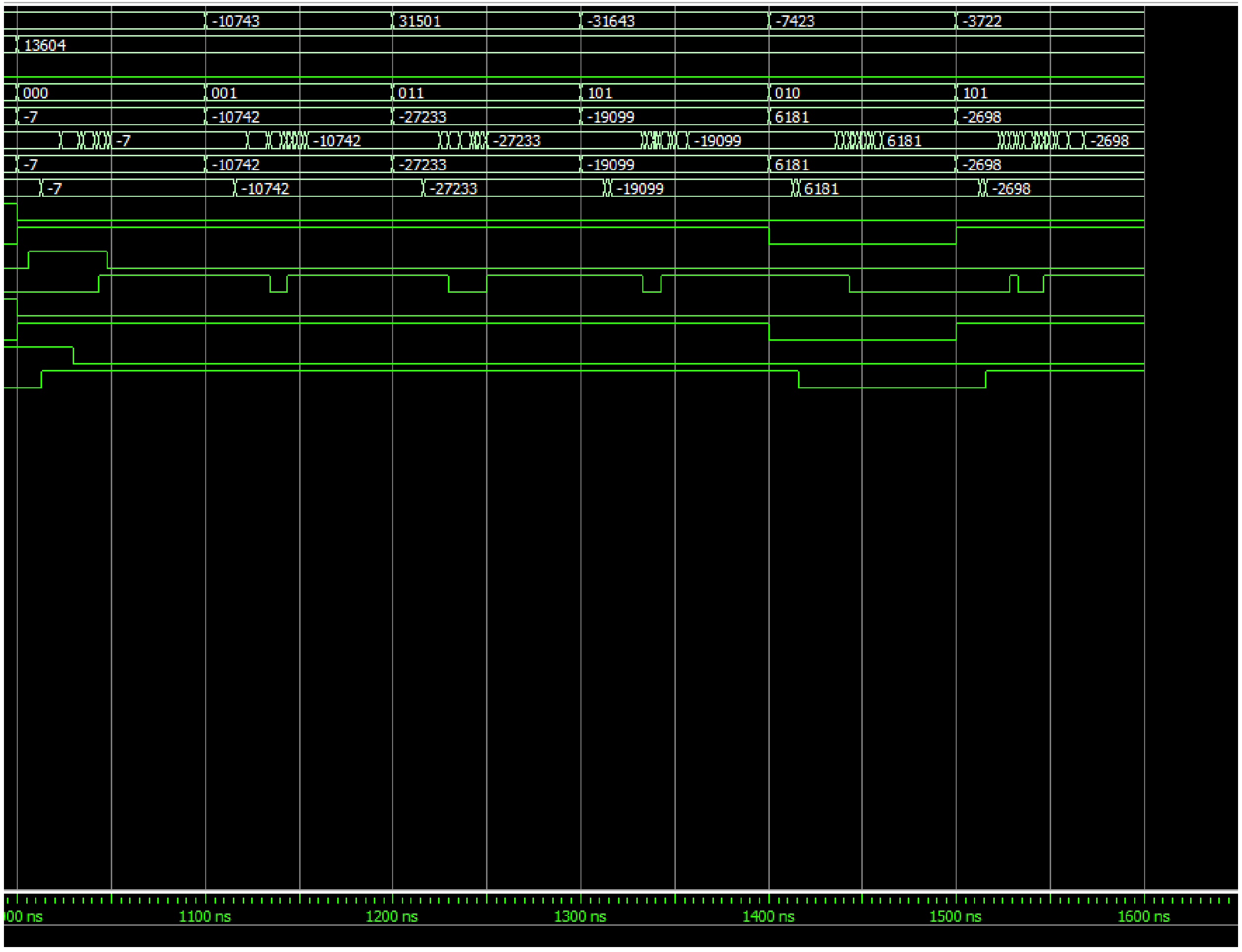
This is the test bench for all parts of this assignment:



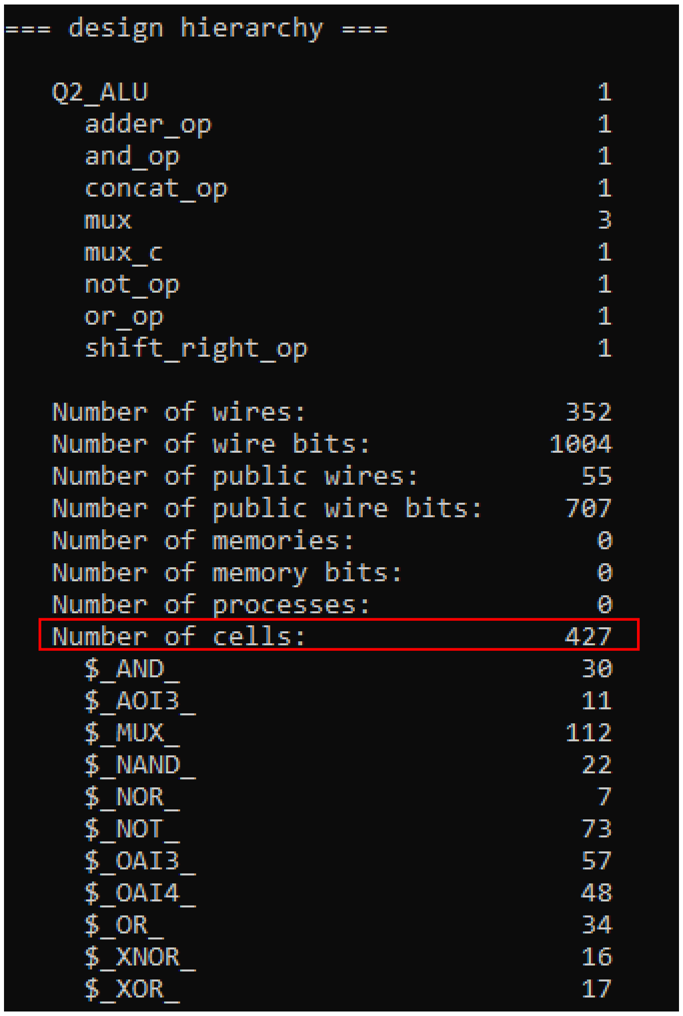
```verilog
`timescale 1ns/1ns
module tb_ALU_2();
  reg signed [15:0] inpA, inpB;
  reg inpC;
  reg [2:0] opcode;
  wire signed [15:0] outputWQ1,outputWQ1Y,outputWQ2,outputWQ2Y;
  wire zero_flagQ1, neg_flagQ1,zero_flagQ1Y, neg_flagQ1Y,zero_flagQ2, neg_flagQ2,zero_flagQ2Y, neg_flagQ2Y;


  Q1_ALU uut(inpA,inpB,inpC, opcode,outputWQ1,zero_flagQ1,neg_flagQ1);
  Q1_ALU_yosys cut(inpA,inpB,inpC,opcode,outputWQ1Y,zero_flagQ1Y,neg_flagQ1Y);
  Q2_ALU uuut(inpA,inpB,inpC,opcode,outputWQ2,zero_flagQ2,neg_flagQ2);
  Q2_ALUy ccut(inpA,inpB,inpC,opcode,outputWQ2Y,zero_flagQ2Y,neg_flagQ2Y);


  initial begin
    #100
    opcode = 3'b000;
    inpC = 1'b1;
    inpA = 16'd7;
    inpB = 16'd64;

    #100 inpC = 1'b0;
    #100 opcode = 3'b001;
    #100 opcode = 3'b010;
    #100 opcode = 3'b011;
    #100 opcode = 3'b100;
    #100 opcode = 3'b101;
    #100 opcode = 3'b110;
    #100 opcode = 3'b111;
    #100
    opcode = 3'b000;
    inpB = $random;
    repeat(5) begin
        #100
        opcode = $random;
        inpA = $random;
    end
    #100 $stop;

  end
endmodule
```

Here is the timing diagram:





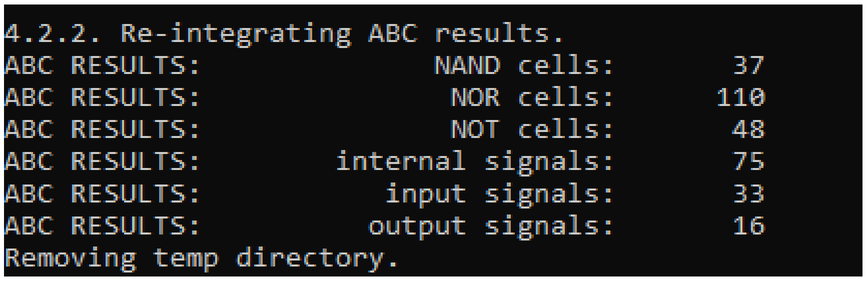
Part b:
Now we use Yosys to synthsis the code.
(Results of synthesis in Yosys without our library).



```
=== design hierarchy ===

   Q2_ALU                          1
     adder_op                      1
     and_op                        1
     concat_op                     1
     mux                           3
     mux_c                         1
     not_op                        1
     or_op                         1
     shift_right_op                1

   Number of wires:              352
   Number of wire bits:         1004
   Number of public wires:        55
   Number of public wire bits:   707
   Number of memories:             0
   Number of memory bits:          0
   Number of processes:            0
   Number of cells:              427
     $_AND_                       30
     $_AOI3_                      11
     $_MUX_                      112
     $_NAND_                      22
     $_NOR_                        7
     $_NOT_                       73
     $_OAI3_                      57
     $_OAI4_                      48
     $_OR_                        34
     $_XNOR_                      16
     $_XOR_                       17
```

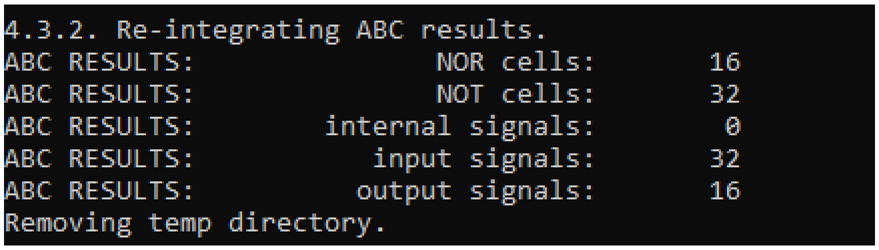
Number of cells is decreased.(from 446 to 427)

Result of gates using the library provided:



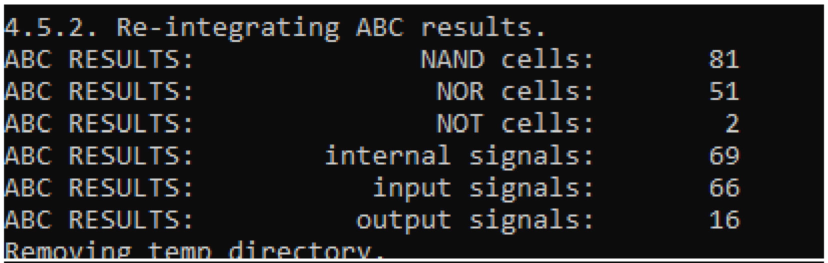
```
4.2.2. Re-integrating ABC results.
ABC RESULTS:                NAND cells:       37
ABC RESULTS:                 NOR cells:      110
ABC RESULTS:                 NOT cells:       48
ABC RESULTS:          internal signals:       75
ABC RESULTS:             input signals:       33
ABC RESULTS:            output signals:       16
Removing temp directory.
```

Adder_op



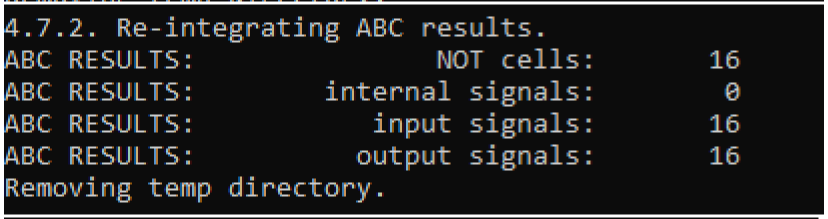
```
4.3.2. Re-integrating ABC results.
ABC RESULTS:                 NOR cells:       16
ABC RESULTS:                 NOT cells:       32
ABC RESULTS:          internal signals:        0
ABC RESULTS:             input signals:       32
ABC RESULTS:            output signals:       16
Removing temp directory.
```

And_op



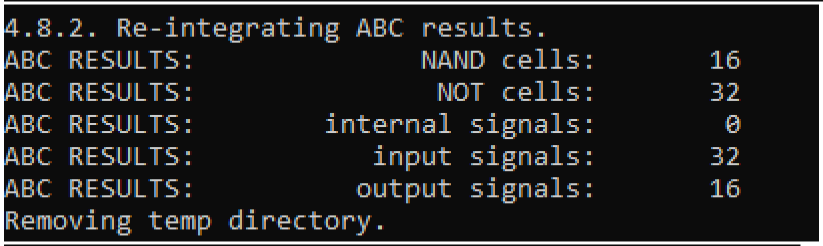
```
4.5.2. Re-integrating ABC results.
ABC RESULTS:                NAND cells:       81
ABC RESULTS:                 NOR cells:       51
ABC RESULTS:                 NOT cells:        2
ABC RESULTS:          internal signals:       69
ABC RESULTS:             input signals:       66
ABC RESULTS:            output signals:       16
Removing temp directory.
```

mux



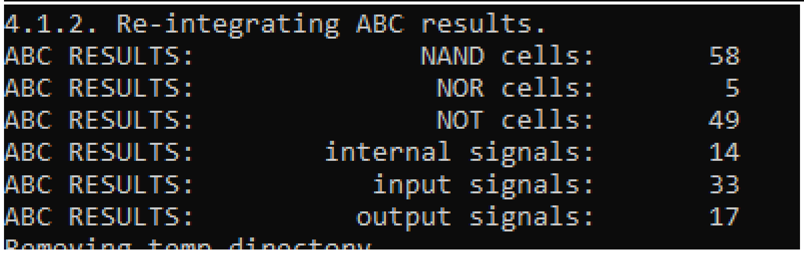
```
4.7.2. Re-integrating ABC results.
ABC RESULTS:                 NOT cells:       16
ABC RESULTS:          internal signals:        0
ABC RESULTS:             input signals:       16
ABC RESULTS:            output signals:       16
Removing temp directory.
```

Mux_c



```
4.8.2. Re-integrating ABC results.
ABC RESULTS:                NAND cells:       16
ABC RESULTS:                 NOT cells:       32
ABC RESULTS:          internal signals:        0
ABC RESULTS:             input signals:       32
ABC RESULTS:            output signals:       16
Removing temp directory.
```

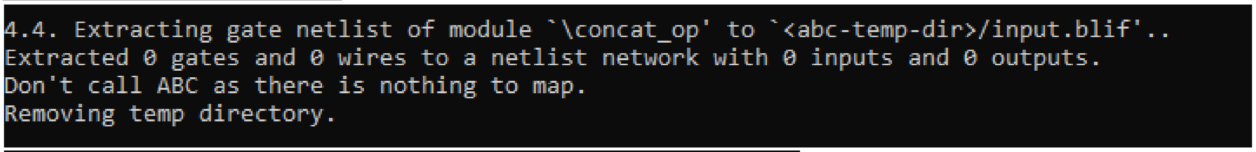
Not_op



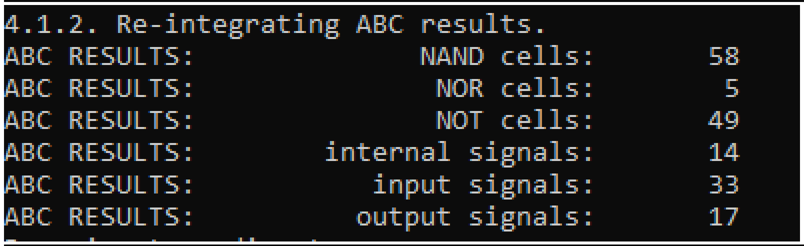
```
4.1.2. Re-integrating ABC results.
ABC RESULTS:                NAND cells:       58
ABC RESULTS:                 NOR cells:        5
ABC RESULTS:                 NOT cells:       49
ABC RESULTS:          internal signals:       14
ABC RESULTS:             input signals:       33
ABC RESULTS:            output signals:       17
Removing temp directory.
```

Or_op

concat

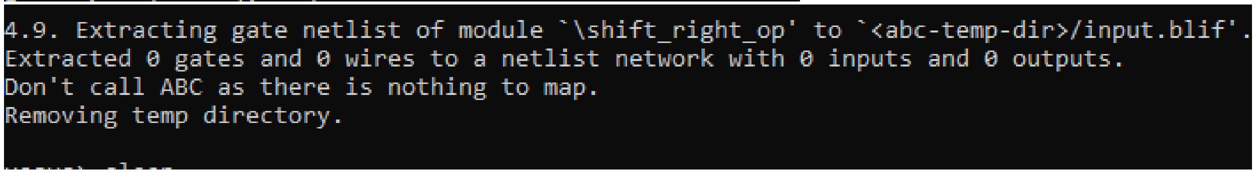


```
4.4. Extracting gate netlist of module `\concat_op' to `<abc-temp-dir>/input.blif'..
Extracted 0 gates and 0 wires to a netlist network with 0 inputs and 0 outputs.
Don't call ABC as there is nothing to map.
Removing temp directory.
```



```
4.1.2. Re-integrating ABC results.
ABC RESULTS:              NAND cells:       58
ABC RESULTS:               NOR cells:        5
ABC RESULTS:               NOT cells:       49
ABC RESULTS:        internal signals:       14
ABC RESULTS:           input signals:       33
ABC RESULTS:          output signals:       17
```

Q2_ALU



```
4.9. Extracting gate netlist of module `\shift_right_op' to `<abc-temp-dir>/input.blif'.
Extracted 0 gates and 0 wires to a netlist network with 0 inputs and 0 outputs.
Don't call ABC as there is nothing to map.
Removing temp directory.
```

Shift_right_op

Number of cells: 556

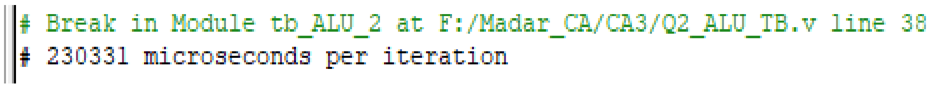
Part c: It is shown above.

Part d:

Like question 1, part a is made by Verilog using software components like always and case statements, we don’t know what hardware implementations does Verilog use for the statements, but yosys simulates the hardware using gates and library provided. I think is that part c is slower because it is completely made by hardware and gates and sth that are limited -> I mean the limited gates that are in the library provided.

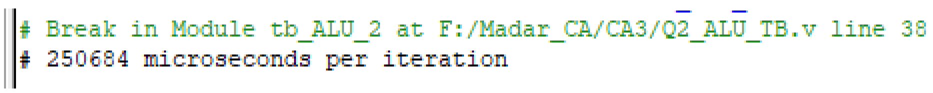
To calculate the time delay: we use again this command while we are in vsim. ”time {run -all}”

Time for part a:



```
# Break in Module tb_ALU_2 at F:/Madar_CA/CA3/Q2_ALU_TB.v line 38
# 230331 microseconds per iteration
```

And this is for part c:



```
# Break in Module tb_ALU_2 at F:/Madar_CA/CA3/Q2_ALU_TB.v line 38
# 250684 microseconds per iteration
```

Question 3:

If we compare the synthesized results of question 1 and 2, we can conclude that question 2 circuit is much more efficient because of the hardware implementation and somehow use brain😊 and some fascinating ways to minimize the circuit.

But it is hard to design these circuits and these are a little slower so there is no free lunch!