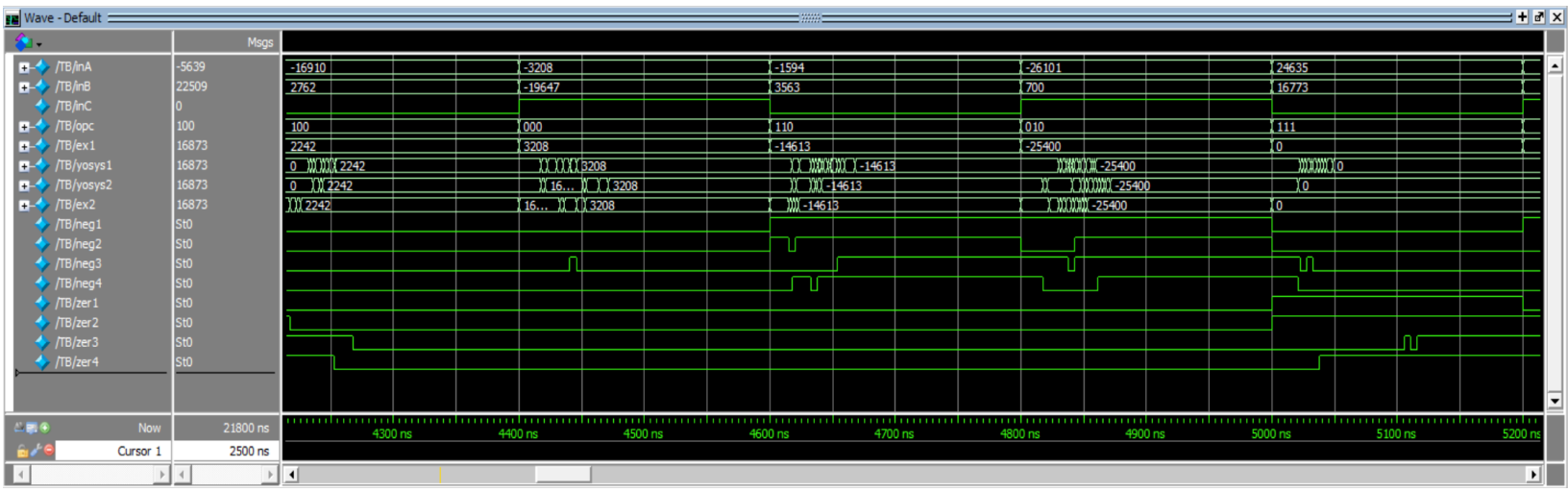
Amin Aghakasiri 810101381

CA3 Report

NOTE: Testbench code and waveform results for all 4 parts is provided in below 2 pictures:

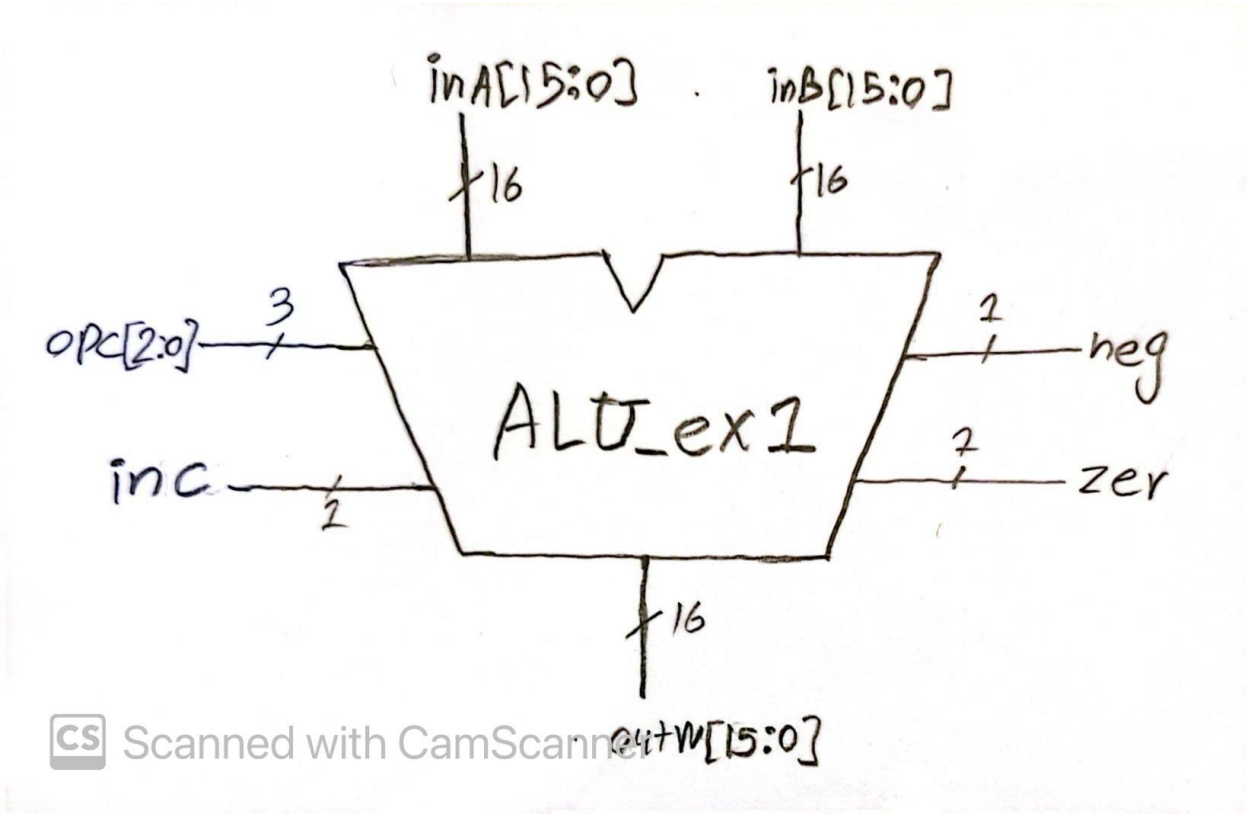
testbench.sv

```
`timescale 1ns/1ns
module TB();
    wire signed [15:0] ex1,ex2,yosys1,yosys2;
    wire zer, neg;
    logic signed [15:0] inA, inB;
    logic inC;
    logic [2:0] opc;
    ALU_ex1 tb1(ex1,neg,zer,inA,inB,inC,opc);
    ALU_yosys1 tb2(yosys1,neg,zer,inA,inB,inC,opc);
    ALU_ex21 tb3(ex2,neg,zer,inA,inB,inC,opc);
    ALU_yosys2 tb4(yosys2,neg,zer,inA,inB,inC,opc);

    initial begin
        opc = 3'b000;
        inC = 1'b1;
        inA = 16'd243;
        inB = 16'd3210;
        inC = 1'b0;

        #200 opc = 3'b000;
        #200 opc = 3'b001;
        #200 opc = 3'b010;
        #200 opc = 3'b011;
        #200 opc = 3'b100;
        #200 opc = 3'b101;
        #200 opc = 3'b110;
        #200 opc = 3'b111;

        repeat(100000) begin
          #200
          inB = $random;
          opc = $random;
          inA = $random;
          inC = $random;
        end
        #200 $stop;
    end

endmodule
```



Question 1:

The handwritten hardware of this question is shown below:



Part a: The testbench and results are shown above.

Part b:

Result using yosys library:

```
=== ALU_ex1 ===

   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
```

Result using provided library:

```
5.1.2. Re-integrating ABC results.
ABC RESULTS:              NAND cells:      183
ABC RESULTS:               NOR cells:      436
ABC RESULTS:               NOT cells:      138
ABC RESULTS:        internal signals:      429
ABC RESULTS:           input signals:       36
ABC RESULTS:          output signals:       17
```

Number of cells: 757

Part c: The results are shown in the first page.

Part d: Since part a is made by Verilog using software components such as always and case statement, we don't know what hardware implementation does Verilog use for the statements, in contrast to part c that we yosys simulates the hardware using gates. Also because of considering delays in part c, the waveform of it have x outputs(unlike part a). For evaluating time, it is obvious that part c is slower because it is completely built by hardware and gates. However, we can calculate the times using command "time {run -all} in modelsim like below:

Time for part a:

```
# Break in Module TB at C:/Users/Amin/Documents/Digtal-Systems/CA2/testbench.v line 36
# 370998 microseconds per iteration
```

Time for part c:

```
# Break in Module TB at C:/Users/Amin/Documents/Digtal-Systems/CA2/testbench.v line 36
# 4257442 microseconds per iteration
```

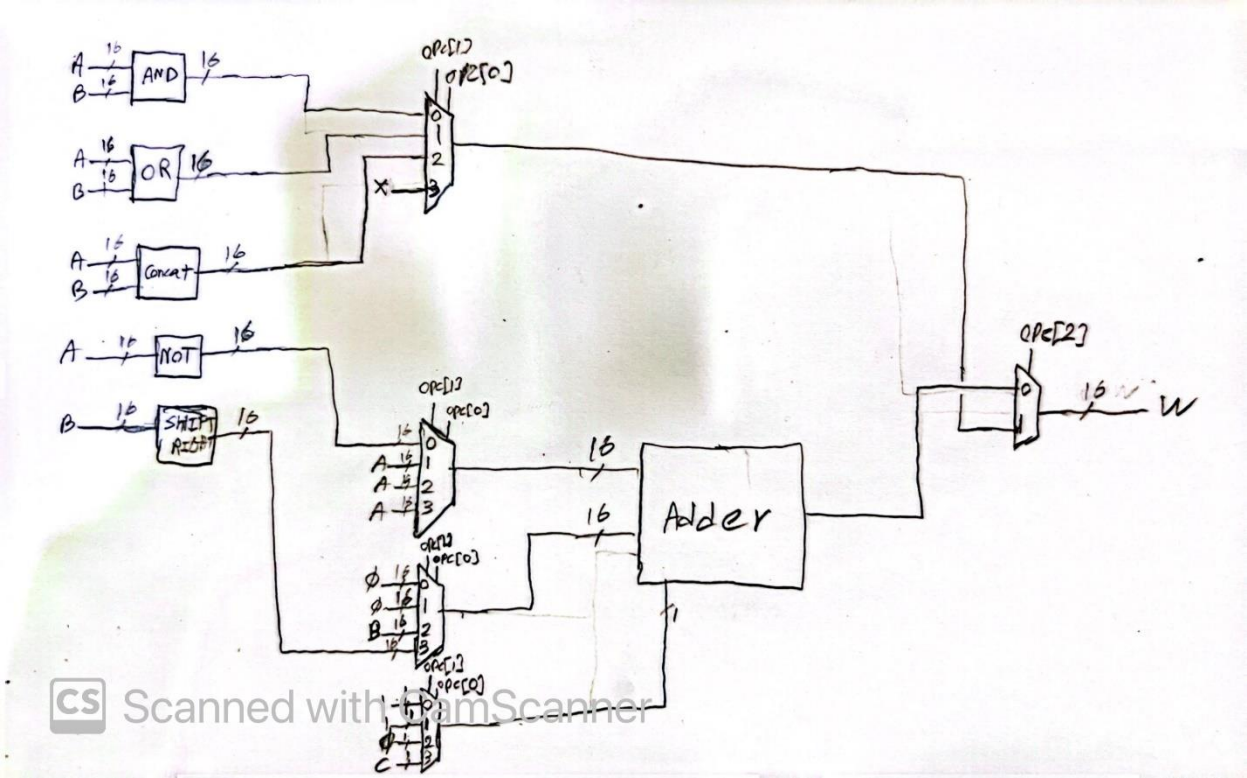
The code of part a:

ex1.sv

```
`timescale 1ns/1ns
module ALU_ex1(
    output logic signed [15:0] outW,
    output neg, zer,
    input signed [15:0] inA, inB,
    input inC,
    input [2:0] opc
);
    always @(inA,inB,inC,opc) begin
        outW = 16'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]};
            3'b111: outW = 16'b0;
            default : outW = 16'bx;
        endcase
    end
    assign  neg = outW[15];
    assign  zer = ~|outW;
endmodule
```

Question 2:

The handwritten hardware of this question is shown below:



Part a: The testbench and results are shown in the first page.

Part b:

Result using yosys library (next page):

```
=== design hierarchy ===

   ALU_ex2                          1
     AB_AND                         1
     AB_CONCAT                      1
     AB_OR                          1
     COMPLEMENT                     1
     FIRST4OPS                      1
     LAST3OPS                       1
     MUX4TO1A                       1
     MUX4TO1B                       1
     MUX4TO1CONTROL                 1
     SHIFT                          1

   Number of wires:               213
   Number of wire bits:           793
   Number of public wires:         50
   Number of public wire bits:    630
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:               293
     $_AND_                        30
     $_AOI3_                       11
     $_MUX_                        80
     $_NAND_                       19
     $_NOR_                        23
     $_NOT_                        56
     $_OAI3_                        9
     $_OR_                         32
     $_XNOR_                       16
     $_XOR_                        17
```

As we can see, the number is so much decreased (from 446 to 293).

Result using provided library (for all modules in my code):

```
4.1.2. 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.
```

AB_AND

```
4.3.2. 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.
```

AB_OR

```
4.5.2. Re-integrating ABC results.
ABC RESULTS:               NOT cells:       16
ABC RESULTS:        internal signals:        0
ABC RESULTS:           input signals:       16
ABC RESULTS:          output signals:       16
```

COMPLEMENT

```
4.6.2. 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
```

FIRST4OPS

```
4.7.2. Re-integrating ABC results.
ABC RESULTS:              NAND cells:       34
ABC RESULTS:               NOR cells:       50
ABC RESULTS:               NOT cells:       34
ABC RESULTS:        internal signals:       19
ABC RESULTS:           input signals:       50
ABC RESULTS:          output signals:       16
```

LAST3OPS

```
4.8.2. Re-integrating ABC results.
ABC RESULTS:              NAND cells:        1
ABC RESULTS:               NOR cells:       49
ABC RESULTS:               NOT cells:        2
ABC RESULTS:        internal signals:        1
ABC RESULTS:           input signals:       34
ABC RESULTS:          output signals:       16
```

MUX4TO1A

```
4.9.2. Re-integrating ABC results.
ABC RESULTS:              NAND cells:       32
ABC RESULTS:               NOR cells:       32
ABC RESULTS:               NOT cells:       16
ABC RESULTS:        internal signals:       53
ABC RESULTS:           input signals:       34
ABC RESULTS:          output signals:       16
```

MUX4TO1B

```
4.10.2. Re-integrating ABC results.
ABC RESULTS:              NAND cells:        2
ABC RESULTS:               NOT cells:        1
ABC RESULTS:        internal signals:        1
ABC RESULTS:           input signals:        3
ABC RESULTS:          output signals:        1
```

MUX4TO1CON

AB_CONCAT

```
Extracted 0 gates and 0 wires to a netlist network with 0 inputs and 0 outputs.
```

SHIFT

```
Extracted 0 gates and 0 wires to a netlist network with 0 inputs and 0 outputs.
```

ALU_ex2(main module):

```
4.4.2. 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
```

Number of cells: 672

Part c: The results are shown in the first page.

Part d: Like question 1, since part a is made by Verilog using software components such as always and case statement, we don’t know what hardware implementation does Verilog use for the statements, in contrast to part c that we yosys simulates the hardware using gates. Also because of considering delays in part c, the waveform of it have x outputs(unlike part a). For evaluating time, it is obvious that part c is slower because it is completely built by hardware and gates.

However, we can calculate the times using command “time {run -all} in modelsim like below:

Time for part a:

```
# Break in Module TB at C:/Users/Amin/Documents/Digtal-Systems/CA2/testbench.v line 36
# 3546125 microseconds per iteration
```

Time for part c:

```
# Break in Module TB at C:/Users/Amin/Documents/Digtal-Systems/CA2/testbench.v line 36
# 4126897 microseconds per iteration
```

Code of part a is show below:

```
`timescale 1ns/1ns
module AB_AND1(output signed [15:0]out,input signed [15:0] inA,input signed [15:0] inB);
    assign out = inA & inB;
endmodule

module AB_OR1(output signed [15:0]out,input signed [15:0] inA,input signed [15:0] inB);
    assign out = inA | inB;
endmodule

module AB_CONCAT1(output signed [15:0]out,input signed [15:0] inA,input signed [15:0] inB);
    assign out = {inA[7:0], inB[7:0]};
endmodule

module COMPLEMENT1(output signed [15:0]out,input signed [15:0] inA);
    assign out = ~inA;
endmodule

module SHIFT1(output signed [15:0]out,input signed [15:0] inB);
    assign out = inB >>> 1;
endmodule


module MUX4TO1A1(output signed [15:0]out,input [2:0] opc,input signed [15:0] inA,input signed [15:0] inA_comp);
    assign out =
        (opc[1:0] == 2'b00) ? inA_comp:
        ((opc[1:0] == 2'b01) || (opc[1:0] == 2'b10) || (opc[1:0] == 2'b11)) ? inA:
        16'bx;
endmodule

module MUX4TO1B1(output signed [15:0]out,input [2:0] opc,input signed [15:0] inB,input signed [15:0] inB_shifted);
    assign out =
        ((opc[1:0] == 2'b00) || (opc[1:0] == 2'b01)) ? 16'b0:
        (opc[1:0] == 2'b10) ? inB:
        (opc[1:0] == 2'b11) ? inB_shifted:
        16'bx;
endmodule

module MUX4TO1CONTROL1(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:
```

```
endmodule

module FIRST4OPS1(output signed [15:0]out,input signed [15:0] inA_new,input signed [15:0] inB_new,input inC_new);
    assign out = inA_new + inB_new + inC_new;
endmodule

module LAST3OPS1(output signed [15:0]out,input [2:0] opc,input signed [15:0] A_B_anded,input signed [15:0] A_B_concat,input signed [15:0] A_B_ored);
    assign out =
        (opc[1:0] == 2'b00) ? A_B_anded:
        (opc[1:0] == 2'b01) ? A_B_ored:
        (opc[1:0] == 2'b10) ? A_B_concat:
        16'bx;
endmodule


module ALU_ex21(
    output logic signed [15:0] outW,
    output neg, zer,
    input signed [15:0] inA, inB,
    input inC,
    input [2:0] opc
);
    wire control_mux;
    wire[15:0] a_mux, b_mux, a_comp, b_shift, a_b_ored, a_b_anded, a_b_concat, first4ops, last3ops;
    AB_AND anding(a_b_anded,inA,inB);
    AB_OR oring(a_b_ored,inA,inB);
    AB_CONCAT concating(a_b_concat,inA,inB);
    COMPLEMENT complementing(a_comp,inA);
    SHIFT shifting(b_shift,inB);
    MUX4TO1A A_mux(a_mux,opc,inA,a_comp);
    MUX4TO1B B_mux(b_mux,opc,inB,b_shift);
    MUX4TO1CONTROL Control_mux(control_mux,opc,inC);
    FIRST4OPS F_OPS(first4ops,a_mux,b_mux,control_mux);
    LAST3OPS L_OPS(last3ops,opc,a_b_anded,a_b_concat,a_b_ored);
    assign outW =
        (opc[2] == 2'b0) ? first4ops:
        (opc[2] == 2'b1) ? last3ops:
        16'b0;
    assign neg = outW[15];
    assign zer = ~|outW;
endmodule
```

Question 3:If we compare the synthesis results in question 1 and 2, we can find out that question 2 ALU is much more efficient because of the hardware implementation and using a little creative and tricky way for optimizing that is sharing the hardware and minimizing it as much as we can. But this way is much harder to design and a bit slower.