Digital systems

Computer Assignment 3

810101532 سید علیرضا میرشفیعی

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:

```
3.23. Printing statistics.

=== ALU ===

   Number of wires:                473
   Number of wire bits:            520
   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:                483
     $_AND_                         59
     $_AOI3_                        59
     $_AOI4_                        11
     $_MUX_                          1
     $_NAND_                        34
     $_NOR_                         82
     $_NOT_                         65
     $_OAI3_                        39
     $_OAI4_                         8
     $_OR_                          34
     $_XNOR_                        84
     $_XOR_                          7

3.24. Executing CHECK pass (checking for obvious problems).
checking module ALU..
found and reported 0 problems.
```

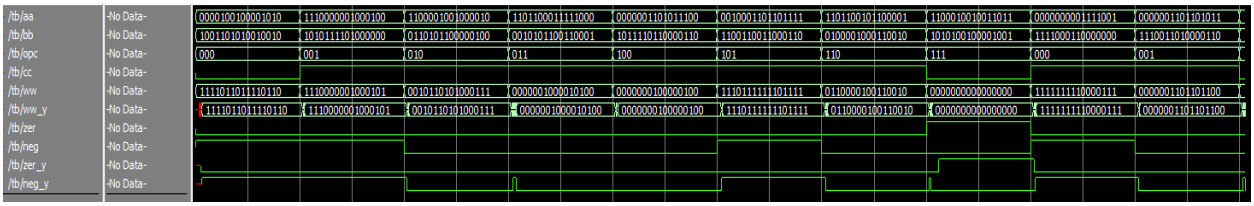
And then we used our own cells lib for our circuit.

```
5.1.2. Re-integrating ABC results.
ABC RESULTS:              NAND cells:      207
ABC RESULTS:               NOR cells:      366
ABC RESULTS:               NOT cells:      130
ABC RESULTS:        internal signals:      466
ABC RESULTS:           input signals:       36
ABC RESULTS:          output signals:       17
Removing temp directory.
```

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:

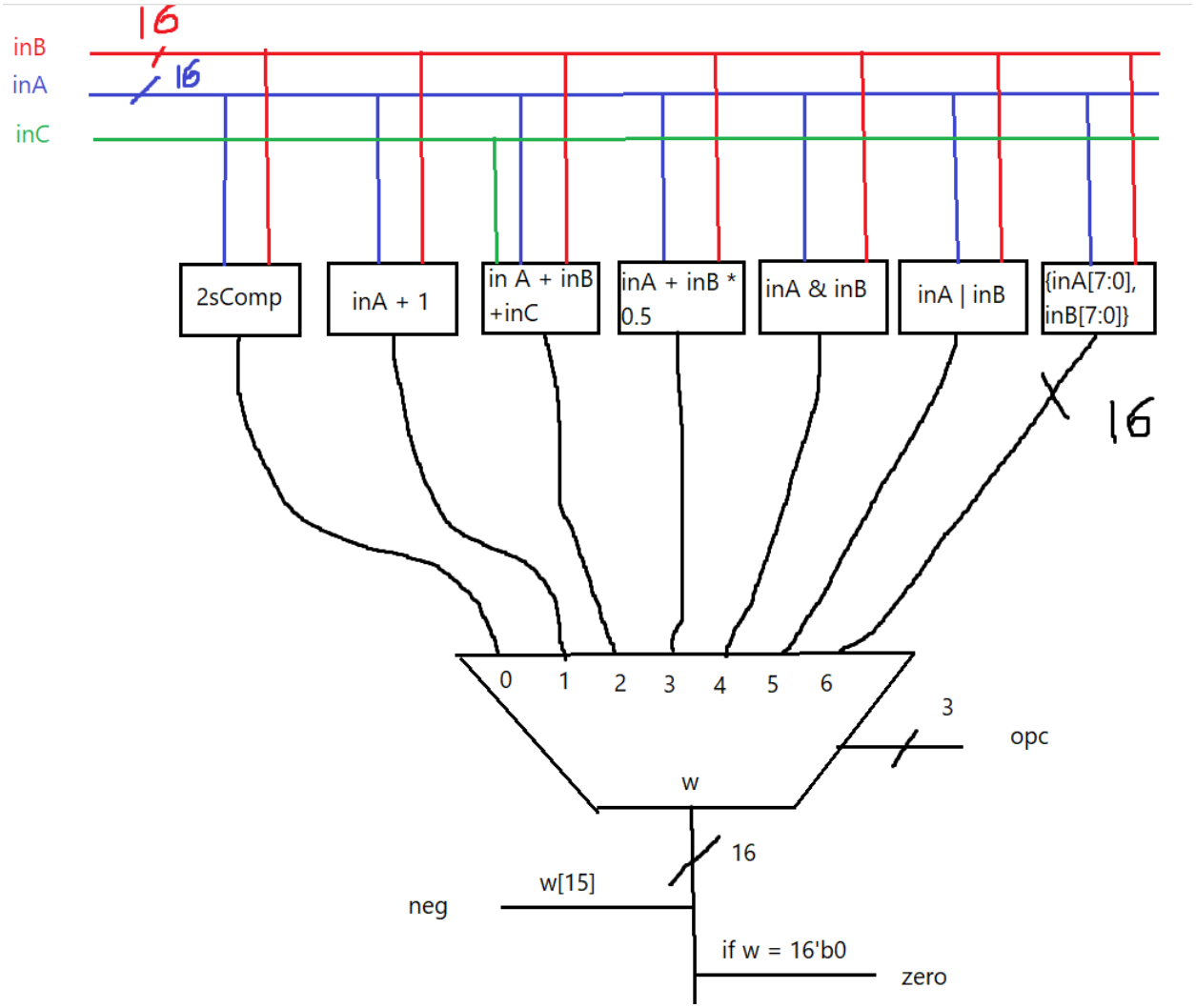
```
set sim_time [time {run -all}]
# 45842 microseconds per iteration
set sim_time [time {run -all}]
# 47279 microseconds per iteration
set sim_time [time {run -all}]
# 47119 microseconds per iteration
set sim_time [time {run -all}]
# 47053 microseconds per iteration
set sim_time [time {run -all}]
# 51472 microseconds per iteration
set sim_time [time {run -all}]
# 46880 microseconds per iteration
set sim_time [time {run -all}]
# 59539 microseconds per iteration
set sim_time [time {run -all}]
# 47394 microseconds per iteration
set sim_time [time {run -all}]
# 48439 microseconds per iteration
set sim_time [time {run -all}]
# 48731 microseconds per iteration
```

Synthesized description:

```
# 70884 microseconds per iteration
set sim_time [time {run -all}]
# 72102 microseconds per iteration
set sim_time [time {run -all}]
# 76543 microseconds per iteration
set sim_time [time {run -all}]
# 75292 microseconds per iteration
set sim_time [time {run -all}]
# 71771 microseconds per iteration
set sim_time [time {run -all}]
# 76505 microseconds per iteration
set sim_time [time {run -all}]
# 77272 microseconds per iteration
VSIM 88> set sim_time [time {run -all}]
# 74355 microseconds per iteration
```

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
```

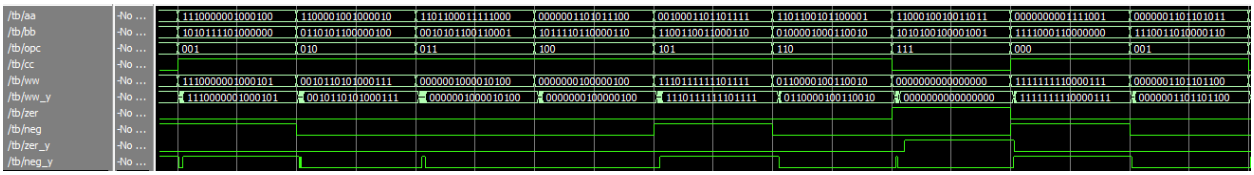
Like question 1 we will start to synthesize it.

```
2.23. Printing statistics.

=== ALU ===

   Number of wires:                435
   Number of wire bits:            497
   Number of public wires:           8
   Number of public wire bits:      70
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                444
     $_AND_                         66
     $_AOI3_                        22
     $_AOI4_                         1
     $_MUX_                         80
     $_NAND_                        53
     $_NOR_                         49
     $_NOT_                         42
     $_OAI3_                        16
     $_OAI4_                        12
     $_OR_                          12
     $_XNOR_                        72
     $_XOR_                         19

2.24. Executing CHECK pass (checking for obvious problems).
checking module ALU..
found and reported 0 problems.
```

Synthesized with our own cells lib.

```
4.1.2. Re-integrating ABC results.
ABC RESULTS:               NAND cells:      225
ABC RESULTS:                NOR cells:      394
ABC RESULTS:                NOT cells:      145
ABC RESULTS:        internal signals:      427
ABC RESULTS:           input signals:       36
ABC RESULTS:          output signals:       17
Removing temp directory.
```

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.

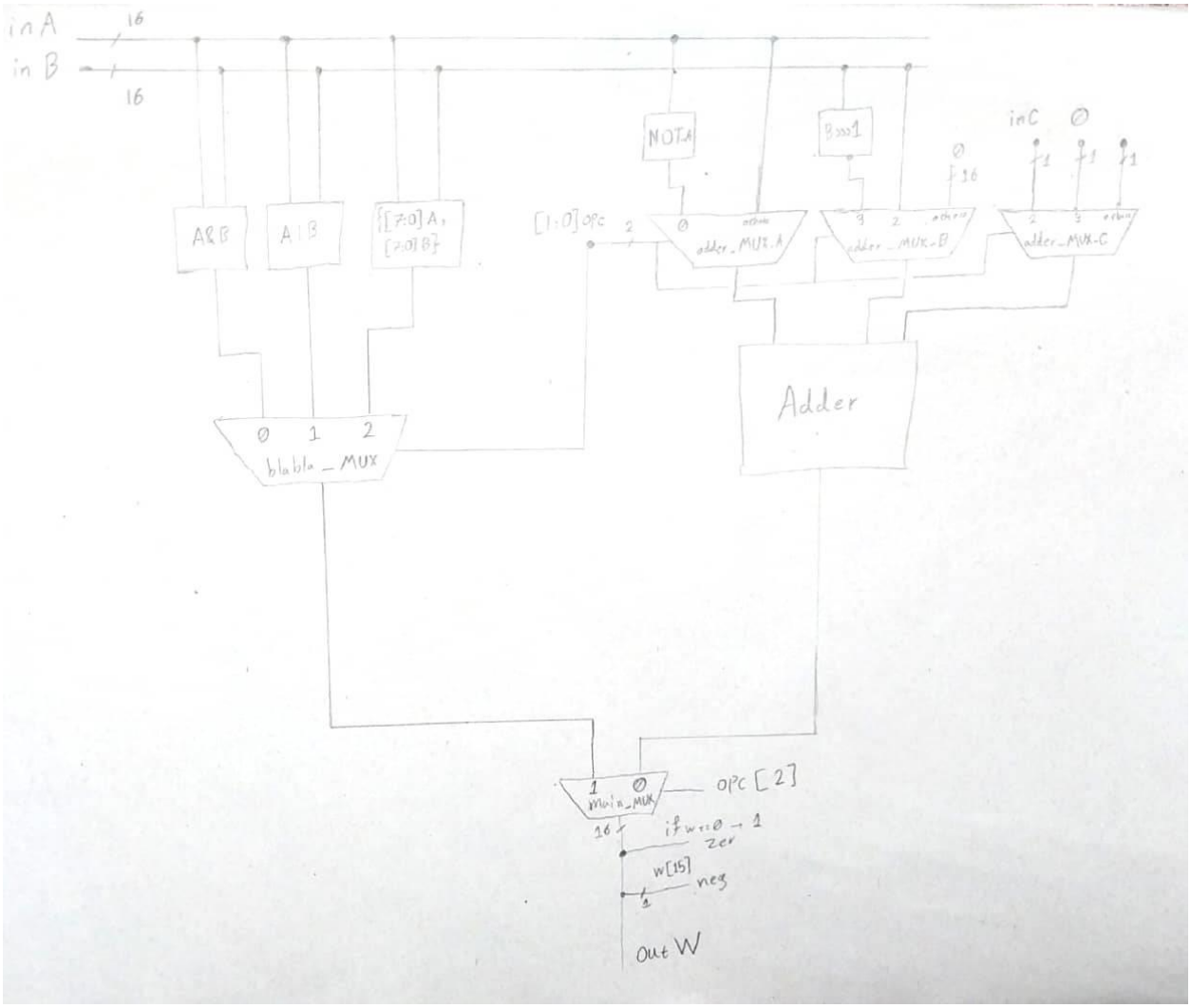
Our description:

```
# 74584 microseconds per iteration
set sim_time [time {run -all}]
# 73875 microseconds per iteration
set sim_time [time {run -all}]
# 69702 microseconds per iteration
set sim_time [time {run -all}]
# 69221 microseconds per iteration
set sim_time [time {run -all}]
# 68409 microseconds per iteration
set sim_time [time {run -all}]
# 67564 microseconds per iteration
set sim_time [time {run -all}]
# 73888 microseconds per iteration
VSIM 27> set sim_time [time {run -all}]
# 68660 microseconds per iteration
```

Synthesized description:

```
# 51724 microseconds per iteration
set sim_time [time {run -all}]
# 44173 microseconds per iteration
set sim_time [time {run -all}]
# 44893 microseconds per iteration
set sim_time [time {run -all}]
# 44584 microseconds per iteration
set sim_time [time {run -all}]
# 43126 microseconds per iteration
set sim_time [time {run -all}]
# 44386 microseconds per iteration
set sim_time [time {run -all}]
# 44219 microseconds per iteration
VSIM 40> set sim_time [time {run -all}]
# 43486 microseconds per iteration
```

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:

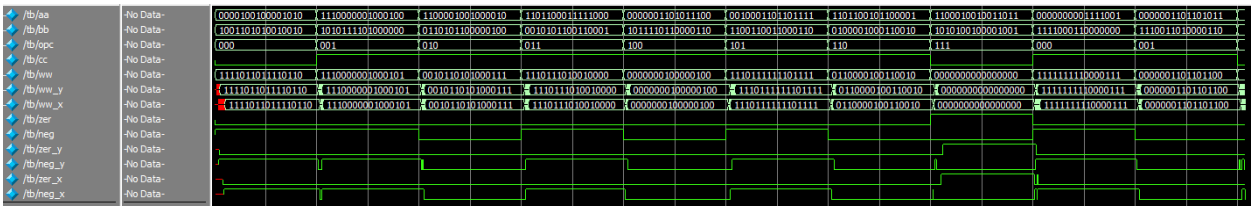
```
=== ALU_min ===

   Number of wires:                281
   Number of wire bits:            358
   Number of public wires:           9
   Number of public wire bits:      86
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                289
     $_AND_                         33
     $_AOI3_                        12
     $_AOI4_                         7
     $_MUX_                         57
     $_NAND_                        22
     $_NOR_                         37
     $_NOT_                         44
     $_OAI3_                        10
     $_OR_                          16
     $_XNOR_                        38
     $_XOR_                         13
```

```
4.1.2. Re-integrating ABC results.
ABC RESULTS:              NAND cells:      134
ABC RESULTS:               NOR cells:      259
ABC RESULTS:               NOT cells:      105
ABC RESULTS:        internal signals:      272
ABC RESULTS:           input signals:       36
ABC RESULTS:          output signals:       17
Removing temp directory.
```

ww = ALU_min

ww_y = ALU_synth

ww_x = ALU_min_synth



Our description:

```
set sim_time [time {run -all}]
# 78302 microseconds per iteration
set sim_time [time {run -all}]
# 75995 microseconds per iteration
set sim_time [time {run -all}]
# 78465 microseconds per iteration
set sim_time [time {run -all}]
# 77353 microseconds per iteration
set sim_time [time {run -all}]
# 77362 microseconds per iteration
set sim_time [time {run -all}]
# 77875 microseconds per iteration
VSIM 187> set sim_time [time {run -all}]
# 76299 microseconds per iteration
```

Synthesized one:

```
# 79658 microseconds per iteration
set sim_time [time {run -all}]
# 82276 microseconds per iteration
set sim_time [time {run -all}]
# 81237 microseconds per iteration
set sim_time [time {run -all}]
# 81498 microseconds per iteration
set sim_time [time {run -all}]
# 79907 microseconds per iteration
set sim_time [time {run -all}]
# 83969 microseconds per iteration
VSIM 219> set sim_time [time {run -all}]
# 78890 microseconds per iteration
```

Question 3:

The synthesized ALU_min had the most delay between all circuits but the least number of cells too. And the simulation speed of ALU_min was more than the others.