ECE 574 – VLSI System Design

Homework n°2

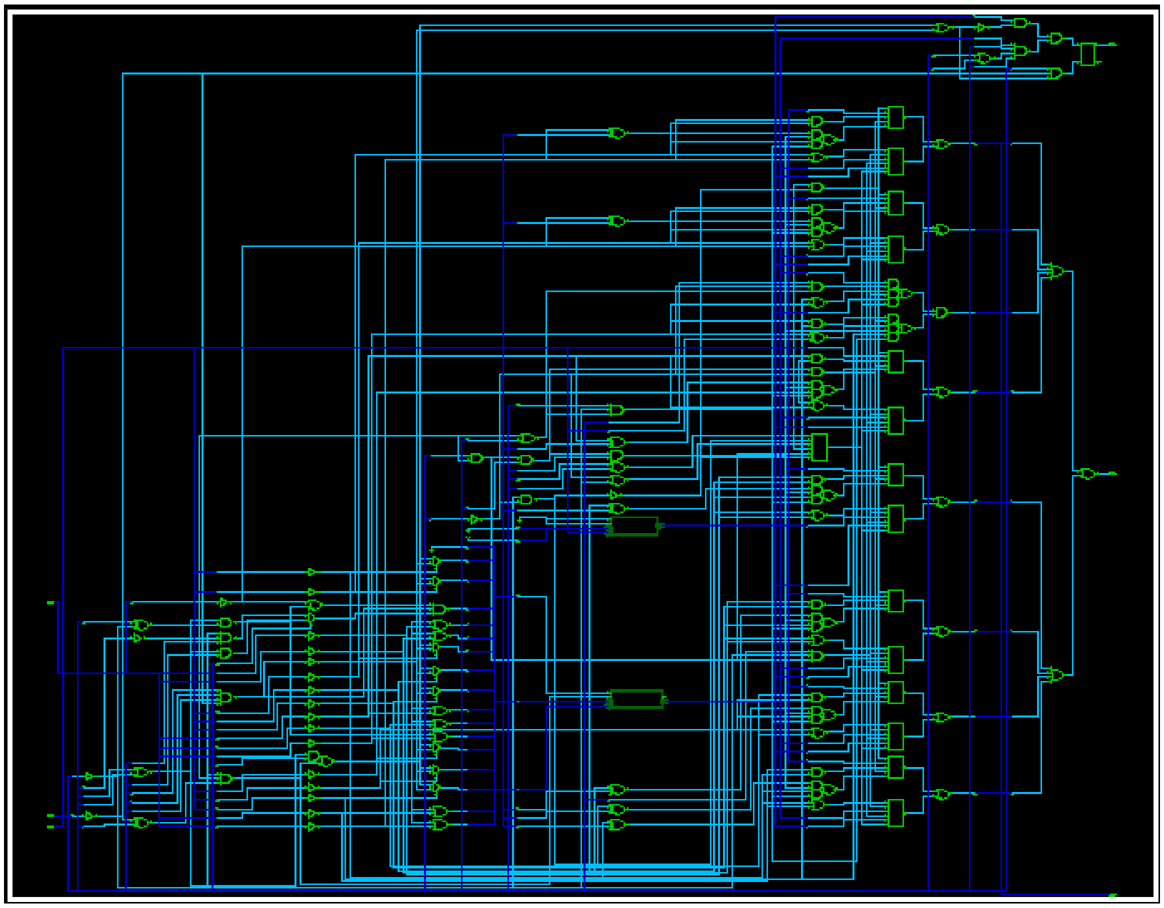
Amine Gaizi
gaizia@oregonstate.edu
Student ID: 934-044-900

1. The Verilog code:

```
//Module Declaration: input and outputs
module alu(
    input[7:0] in_a,       //input a
    input [7:0] in_b,      //input b
    input [3:0] opcode,        //opcode input
    output reg [7:0] alu_out,   //alu output
    output reg   alu_zero,  //logic '1' when alu_output [7:0] is     all zeros
    output reg   alu_carry  //indicates a carry out from AKY
    );

 //Declaration of the opcodes value in Hexadecimal
    parameter c_add = 4'h1;
    parameter c_sub = 4'h2;
    parameter c_inc = 4'h3;
    parameter c_dec = 4'h4;
    parameter c_or = 4'h5;
    parameter c_and = 4'h6;
    parameter c_xor = 4'h7;
    parameter c_shr = 4'h8;
    parameter c_shl = 4'h9;
    parameter c_onescomp = 4'hA;
    parameter c_twoscomp = 4'hB;

    always_comb begin //a
       unique case(opcode) //unique modifier because only one option can match
             c_add: {alu_carry, alu_out} = in_a + in_b; //Arithmetic addition
             c_sub: {alu_carry, alu_out} = in_a - in_b; //Arithmetic susbtracion
             c_inc: {alu_carry, alu_out} = in_a + 1; //Arithmetic incrementation
             c_dec: {alu_carry, alu_out} = in_a - 1; //Arithmetic decrementation
             c_or:
                begin
                   alu_out = in_a | in_b; //Logic OR operation
                   alu_carry = 0; //Reset carry
                end

             c_and:
                begin
                   alu_out = in_a & in_b; //Logic AND operation
                   alu_carry = 0; //Reset carry
                end
             c_xor:
```

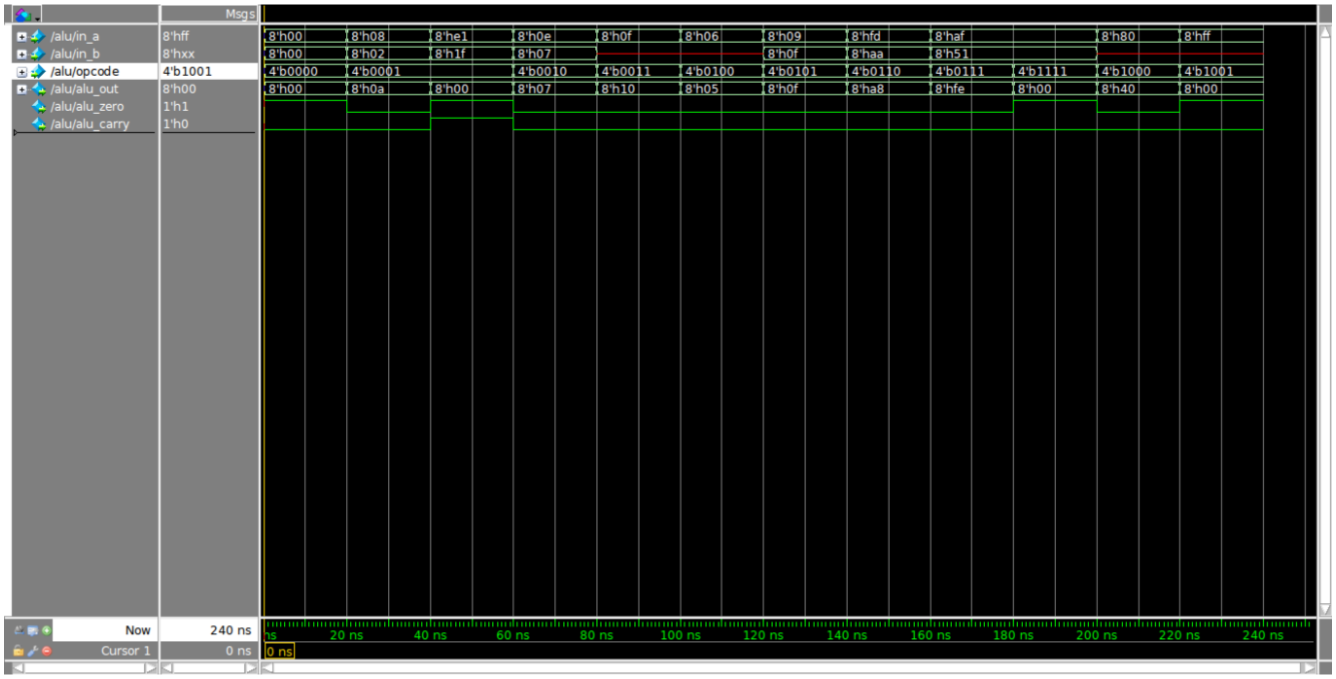
```
            begin
                alu_out = in_a ^ in_b; //Logic XOR operation
                alu_carry = 0; //Reset carry
            end

        c_shr:
            begin
                alu_out = in_a>>1; //Right shifting
                alu_carry = 0; //Reset carry
            end
        c_shl: {alu_carry, alu_out} = 1<<in_a; //Left shifting
        c_onescomp: alu_out = ~in_a;
        c_twoscomp:
            begin
                alu_out = ~in_a; //Invert
                {alu_carry, alu_out} = alu_out+1; //Add 1
            end
        default: {alu_carry, alu_out} = 9'b000000000; //If no match, reset alu_out and alu_carry
    endcase
    alu_zero = ~| alu_out; // NOR operation so: if alu_out = 0, alu_zero=1
  end
endmodule
```

2. Schematic of the designed ALU:

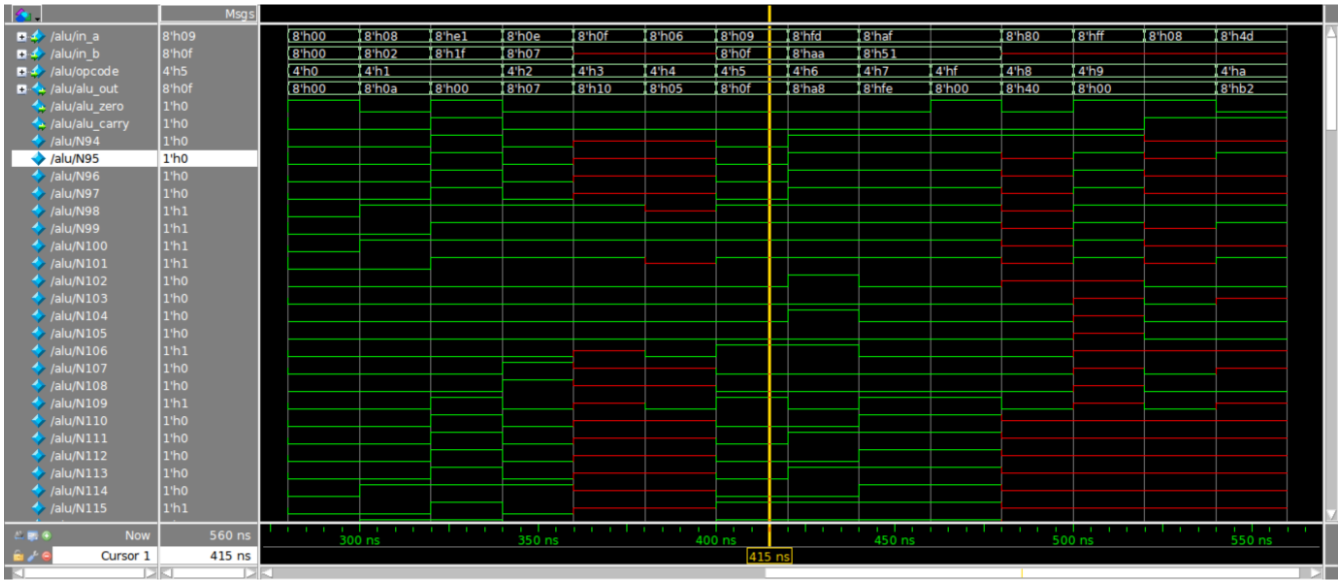


3. Waveform of the code working correctly

a. RTL level


Msgs
/alu/in_a 8'hff
/alu/in_b 8'hxx
/alu/opcode 4'b1001
/alu/alu_out 8'h00
/alu/alu_zero 1'h1
/alu/alu_carry 1'h0
8'h00 8'h08 8'he1 8'h0e 8'h0f 8'h06 8'h09 8'hfd 8'haf 8'h80 8'hff
8'h00 8'h02 8'h1f 8'h07 8'h0f 8'haa 8'h51
4'b0000 4'b0001 4'b0010 4'b0011 4'b0100 4'b0101 4'b0110 4'b0111 4'b1111 4'b1000 4'b1001
8'h00 8'h0a 8'h00 8'h07 8'h10 8'h05 8'h0f 8'ha8 8'hfe 8'h00 8'h40 8'h00
Now 240 ns
Cursor 1 0 ns
ns 20 ns 40 ns 60 ns 80 ns 100 ns 120 ns 140 ns 160 ns 180 ns 200 ns 220 ns 240 ns
0 ns

b. Gate level



c. Test code

```
#undefined start
 force in_a  8'h00
 force in_b  8'h00
 force opcode    4'b0000
```

```
run 20

#add 8+2 = 0xA
force in_a  8'h08
force in_b  8'h02
force opcode    4'b0001
run 20

#add test carry
# 0xE1 + 0x1F = 0x00 and Carry=1
force in_a  8'hE1
force in_b  8'h1F
force opcode    4'b0001
run 20

#susbstract 0x0E - Ox07 = 0x07
force in_a      8'h0E
force in_b      8'h07
force opcode    4'b0010
run 20

#increment 0x0F + 0x01 = 0x10
force in_a      8'h0F
force in_b      8'hXX
force opcode    4'b0011
run 20

#decrement expected result: 0x05
force in_a  8'h06
force in_b  8'hXX
force opcode    4'b0100
run 20

#logic or, 0b00001001 | 0b00001111 = 0b00001111
force in_a  8'b00001001
force in_b  8'b00001111
force opcode    4'b0101
run 20

#logic and
force in_a  8'b11111101
force in_b  8'b10101010
force opcode    4'b0110
run 20

#logic xor
force in_a  8'b10101111
force in_b  8'b01010001
force opcode    4'b0111
run 20

#wrong OP code for XOR -> goes to default
force in_a      8'b10101111
force in_b      8'b01010001
force opcode    4'b1111
run 20

#logic shift right
force in_a  8'b10000000
force in_b  8'hXX
force opcode    4'b1000
run 20

#logic shift left
force in_a  8'b11111111
force in_b  8'hXX
force opcode    4'b1001
run 20

#logic ones compe
force in_a  8'h08
force in_b  8'hXX
force opcode    4'b1001
run 20

#logic twos comp
force in_a  8'h4D
force in_b  8'hXX
force opcode    4'b1010
run 20
```

4. Written answers:

a. Total area used by the ALU

The total area use by the ALU is: 1814.079795 um^2.

b. Number of different cells utilized

Report hierarchy displays:

```
AND2X1              saed90nm_typ
   AND3X1              saed90nm_typ
   AO21X1              saed90nm_typ
   AO22X1              saed90nm_typ
   AO221X1             saed90nm_typ
   AO222X1             saed90nm_typ
   AOI222X1            saed90nm_typ
   INVX0               saed90nm_typ
   LATCHX1             saed90nm_typ
   MUX21X1             saed90nm_typ
   NAND2X0             saed90nm_typ
   NAND3X0             saed90nm_typ
   NAND4X0             saed90nm_typ
   NOR2X0              saed90nm_typ
```

```
OR2X1                saed90nm_typ
OR4X1                saed90nm_typ
XNOR2X1              saed90nm_typ
alu_DW01_addsub_0
   FADDX1            saed90nm_typ
   XOR2X1            saed90nm_typ
alu_DW01_ash_0
   AND2X1            saed90nm_typ
   AO21X1            saed90nm_typ
   INVX0             saed90nm_typ
   NAND2X0           saed90nm_typ
   NAND3X0           saed90nm_typ
   NOR2X0            saed90nm_typ
   OR3X1             saed90nm_typ
```

This totals to 26 different gates utilized.

c. Number of cells:

$$Gate\ equivalent\ count = \frac{Total\ Area}{NAND2X1\ area} = \frac{1814.079795}{5.5296} = 328{,}067$$

The equivalent gate count is 328 gates.

d. Hierarchical block:

The block is an adder substractor ("alu_DW01_addsub"). It is implemented using a full adder block and an XOR logic gate.

e. Delay:

The maximum delay through the ALU is 2.68ns. The beginning point is "opcode[0](input port)", and the endpoint is "alu_zero(output_port)".