Building a 4-bit ALU using VHDL

Due date: February 26th, 2017

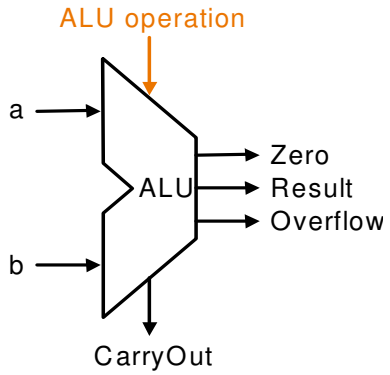
Assignment # 3

Objectives: Arithmetic logic unit (ALU) is a basic building block for processors and many other circuit specific applications. The purpose of this assignment is to build the multifunctional logic unit (ALU) using **Verilog HDL.** This assignment guides student to decompose a large problem into smaller modules and testing them by combining into an integrated circuit.

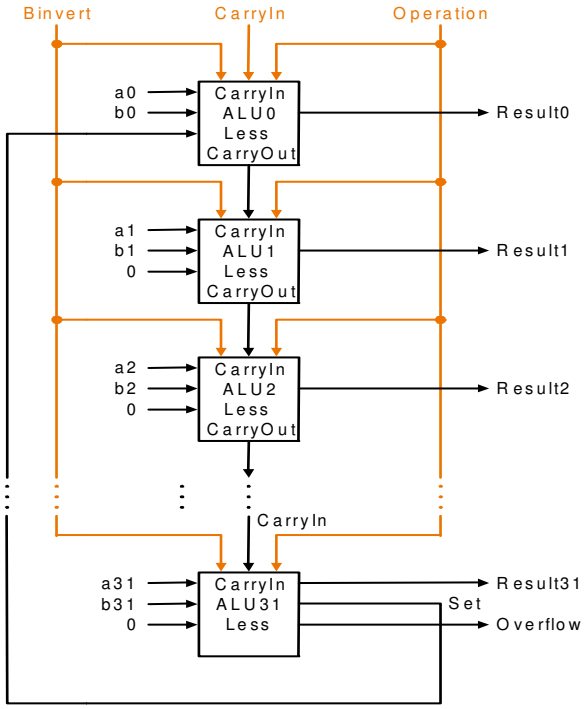
Problem Description: Your task is to create a 4-bit ALU in VHDL. ALU (a digital circuit) performs arithmetic and logical operations. The ALU logic block diagram (1-bit) and its operations shown as following:

ALU control lines

Bneg-ate	Oper-ation	Func-tion
0	00	and
0	01	or
0	10	add
1	10	sub
1	11	slt



You need to integrate the 1-bit ALUs and formulate a 4-bit ALU using the following block diagram:



Specifications

The implemented ALU is capable to take five (5) inputs- two(2) operand bits a and b, two(2) bit selector values and one(1) bit Bnegate value. Selector and Bnegate values specify which of the five functions (listed above) that the ALU performs.

The ALU has four (4) outputs, F (the result), Cout (the carry or borrow output), Zero Flag, which is set to 1 if all bits in ALU results are 0 and overflow bit, which is the xor of carry in to the most significant ALU and carry out from the most significant ALU.

You can only use the given specific modules.

- 4-1 multiplexer.
- 4-bit adder.
- basic gates (NOT, NAND, AND, NOR, OR, XOR).

4-1 multiplexor Verilog HDL code

```
module mux4_to_1 (out, i0, i1, i2, i3, s1,s0);
        output out;
        input i0, i1, i2, i3;
        input s1,s0;
        wire s1n, s0n;
        wire y0, y1, y2, y3;
        not(s1n, s1);
        not(s0n, s0);
        and(y0, i0, s1n, s0n);
        and(y1, i1, s1n, s0);
        and(y2, i2, s1, s0n);
        and(y3, i3, s1, s0);
        or (out, y0, y1, y2, y3);
endmodule
```

4-bit adder Verilog HDL code

```
module fulladd (sum1, c_out1, a1, b1, c_in1);
        output sum1, c_out1;
        input a1,b1,c_in1;
        wire s1,c1,s2;
        xor(s1, a1, b1);
        and (c1, a1, b1);
        xor(sum1, s1, c_in1);
        and (s2, s1,c_in1);
        xor(c_out1, s2,c1);
endmodule
module fulladd4(sum,c_out,a,b,c_in);
        output [3:0] sum;
        output c_out;
        input [3:0] a,b;
        input c_in;
```

```
    wire c1, c2, c3;

    fulladd f0(sum[0], c1, a[0], b[0], c_in);
    fulladd f1(sum[1], c2, a[1], b[1], c1);
    fulladd f2(sum[2], c3, a[2], b[2], c2);
    fulladd f3(sum[3], c_out, a[3], b[3], c3);

endmodule
```

- **You will need to design your own stimulus to input the 4-bit ALU and check its functionalities.**