Project 2: Designing an 8-bit ALU

Adam Sumner - A20283081
Contribution - 100%

ECE 485

November 16th, 2015

1 Introduction

The purpose of this project is to implement an 8-bit ALU in the VHDL hardware-description language. The ALU should be able to perform the following functions:

- ADD
- SUBTRACT
- LESS-THAN
- AND
- NOT
- XOR
- Bit Shift Left
- Arithmetic Bit Shift Right

2 Design

2.1 Architecture

The architecture of the ALU is simple. First and foremost, a `clk` signal is included. This is necessary because during the design process, it was concluded that all operations will be performed and outputted on the positive edge of the `clk`. Also, as per the design specification, two inputs `a` and `b` must be included. These are both signed vectors of 8 bits because this ALU is designed for 8 bit operations. They are signed because operations like addition and subtraction are included in the ALU's capability. Because in the real world negative numbers are often part of computer programs, it was necessary to make sure that the functionality of the ALU would be able to handle them. Due to the fact that signed numbers are being used, two status bits are included. These are `ov` and `zero`. `ov` is the overflow bit that is set when an overflow/underflow occurs, and `zero` is the zero status bit that is set when the output is zero. `op` is the operation bit. This signal is used to select the operation of the ALU to be performed on inputs `a` and `b`. `s` is a selector bit. Because bit shift left and arithmetic bit shift right are included in the functionality of the ALU and the ALU always takes two inputs, this signal is included to select which input to perform the shift operation on. Last an output must be included in the design of the ALU, and this is done by implementing an output `y` of 8 bits. A visual overview of the ALU is shown in Figure 1.

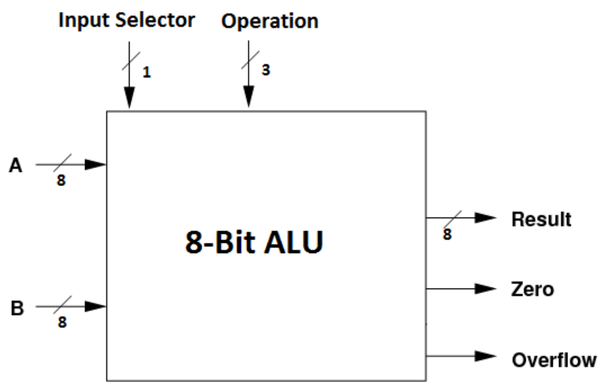

Figure 1: High Level Overview of Architecture

2.2 Behavior

The behavior section of the code in Section 2.3 is at its core a switch case on the operation to be performed. Two bit vectors are first declared, `temp9` and `temp`. `temp9` is used in the calculation for overflow and underflow and `temp` is a temporary storage location of the result so that before its value is outputted from the ALU, its value can be checked for zero to see if the zero status bit should be set or not. A process is defined for the `clk` signal. If the `clk` is at its rising edge, then an operation can be performed.

2.3 Code

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity alu8 is
port(
    clk : in std_logic;
    s : in std_logic;
    ov, zero  : out std_logic;
    a,b : in signed(7 downto 0);
    op  : in unsigned(2 downto 0);
    y : out signed(7 downto 0)

    );
end alu8;

architecture Behavioral of alu8 is

signal temp9: signed(8 downto 0);
signal temp: signed(7 downto 0);

begin
process(clk)
begin
  if(rising_edge(clk)) then
    case op is
      when "000" =>
        temp <= a AND b;
      when "001" =>
        temp <= a OR b;
      when "010" =>
        temp9 <= (a(7)& a) + (b(7) & b);
        if temp9(8) /= temp9(7) then
          ov <= '1'; --overflow!
        end if;
        --unsigned on temp so resize doesn't save the sign
        temp <= signed(resize(unsigned(temp), y'length));
      when "011" =>
        if(a < b) then
          temp <= "11111111";
        else
          temp <= "00000000";
        end if;
      when "100" =>
        temp9 <= (a(7)& a) - (b(7) & b);
        if temp9(8) /= temp9(7) then
          ov <= '1'; --underflow!
        end if;
        --unsigned on temp so resize doesn't save the sign
        temp <= signed(resize(unsigned(temp9), y'length));
      when "101" =>
        temp <= a XOR b;
      when "110" =>
        if s = '1' then
          temp <= SHIFT_LEFT(a,1);
        else
          temp <= SHIFT_LEFT(b,1);
        end if;
      when "111" =>
        if s = '1' then
          temp <= SHIFT_RIGHT(a,1);
        else
          temp <= SHIFT_RIGHT(b,1);
        end if;
      when others =>
        NULL; --default case
    end case;
    if temp = "00000000" then
```

```
        zero <= '1';
      else
        zero <= '0';
      end if;
      y <= temp;
    end if;
end process;

end Behavioral;
```

3 Analysis

4 Simulation Results

5 Conclusion