COEN316 – Lab 1

Name: Christopher Dubuc-Pesetti

Student ID: 40037815

Date submitted: September 30, 2018

"I hereby certify that this submission is my original work and meets the Faculty's Expectations of Originality."

Name: Christopher Dubuc-Pesetti ID: 40037815

Objectives:

The objective of this lab was to design, simulate and synthesize a 32-bit ALU in VHDL. The ALU will be responsible with arithmetic addition and subtraction, as well as logical operations. Overflow and zero detection is also incorporated into the design.

Results & Discussion:

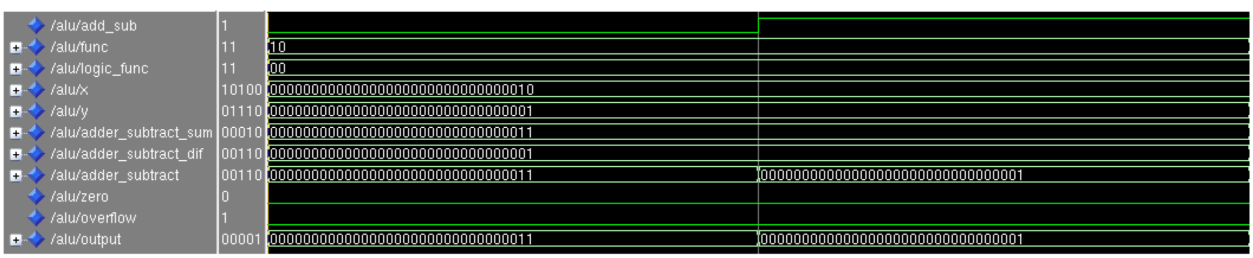
The signed std_logic library was used to complete the ALU.

```vhdl
--sum comb logic
adder_subtract_sum <= signed(x) + signed(y);

--dif comb logic
adder_subtract_dif <= signed(x) - signed(y);

--adder_subtract block
process(adder_subtract_sum,adder_subtract_dif,add_sub)
begin
    if (add_sub = '0') then
        adder_subtract <= std_logic_vector(adder_subtract_sum);
    else
        adder_subtract <= std_logic_vector(adder_subtract_dif);
    end if;
end process;
```

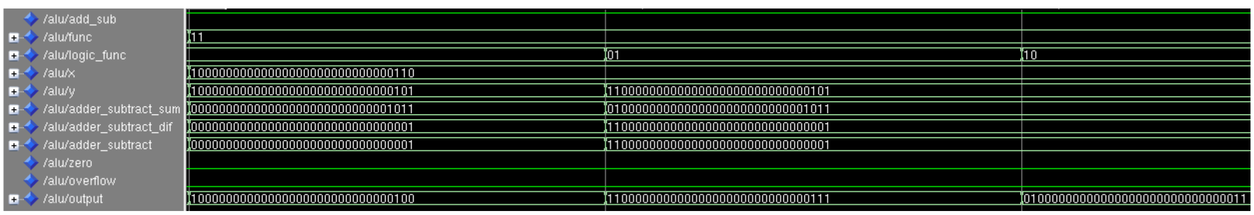
Above shows how the addition and subtraction was performed in VHDL with casting.



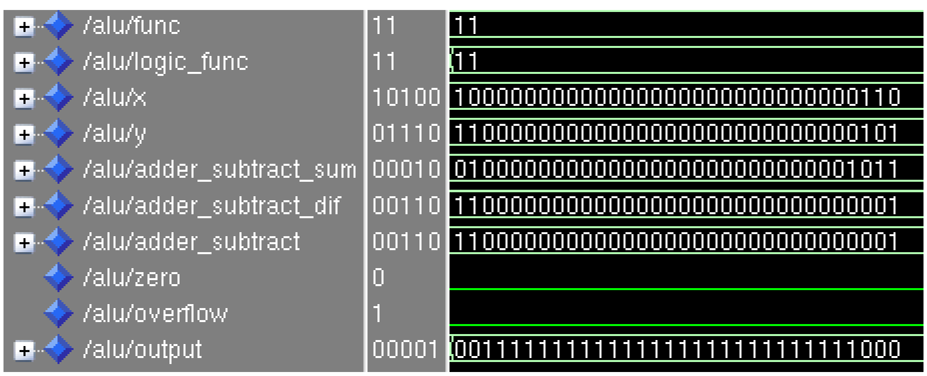
First tests were done to check the addition and subtraction were functioning correctly. First case adds 2 + 1 = 3 when add_sub = ‘0’. Second case performs 2 – 1 = 1 when add_sub = ‘1’.

```vhdl
--logic unit block
process(x,y,logic_func)
variable temp : std_logic_vector(31 downto 0);
begin
    if (logic_func = "00") then --AND
        temp := x AND y;
    elsif (logic_func = "01") then --OR
        temp := x OR y;
    elsif (logic_func = "10") then --XOR
        temp := x XOR y;
    else                    --NOR
        temp := (not (x OR y));
    end if;
    logic_unit <= temp;
end process;
```

Above shows code snippet of how the logical operations are performed using a variable.



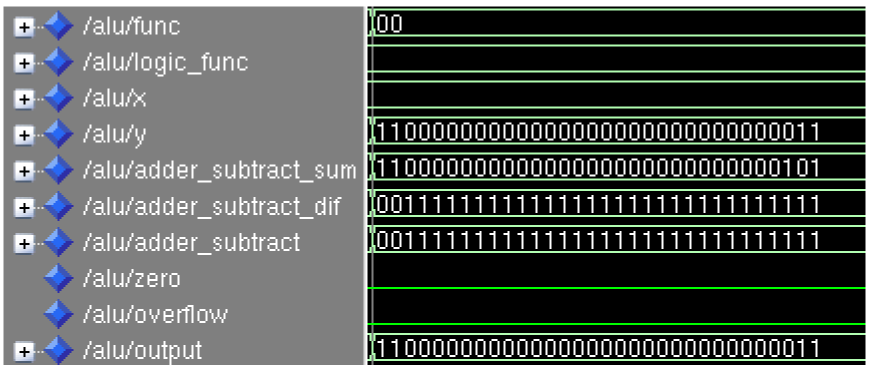
First case above is performing the AND operation, second case the OR operation and third case the XOR operation.



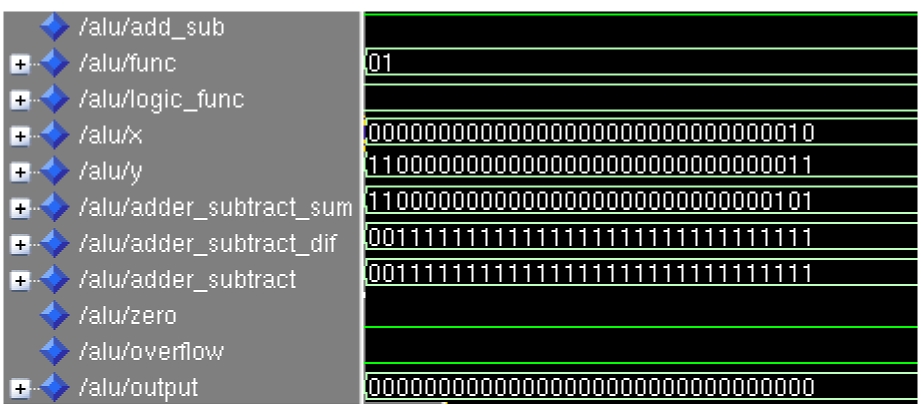
Above shows the NOR operation.

```vhdl
-- 4to1 output mux
process(y,adder_subtract,logic_unit,func,adder_subtract_dif)
variable temp : std_logic_vector(31 downto 0) := (others => '0');
begin
    if (func = "00") then          -- lui
        temp := y;
    elsif (func = "01") then     -- slt
        temp := "0000000000000000000000000000000" & std_logic_vector(adder_subtract_dif(31 downto 31));
    elsif (func = "10") then     -- output of adder_subtract
        temp := adder_subtract;
    else                         -- output of logic_unit
        temp := logic_unit;
    end if;
    --output <= (not temp); -- negated onboard LED output
    output <= temp;
end process;
```

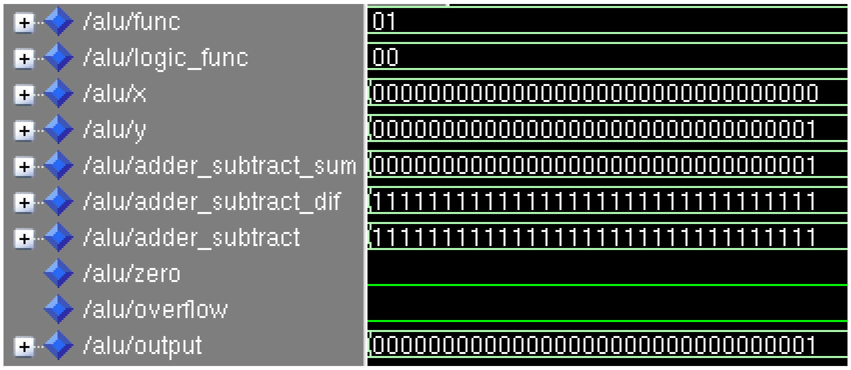
Above is the 4to1 Mux code snippet that details the main operations of the ALU based on the 2-bit func input.



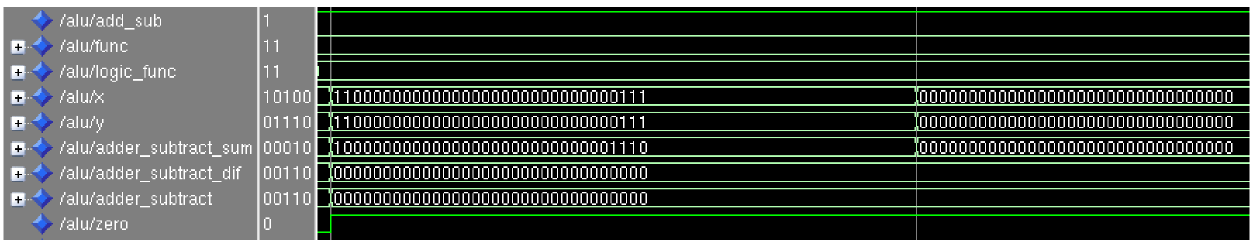
Above is the lui (load upper immediate, output <= y) instruction.



Above is the slt instruction with the MSB being ‘0’.



Above is the slt instruction with the MSB being ‘1’.

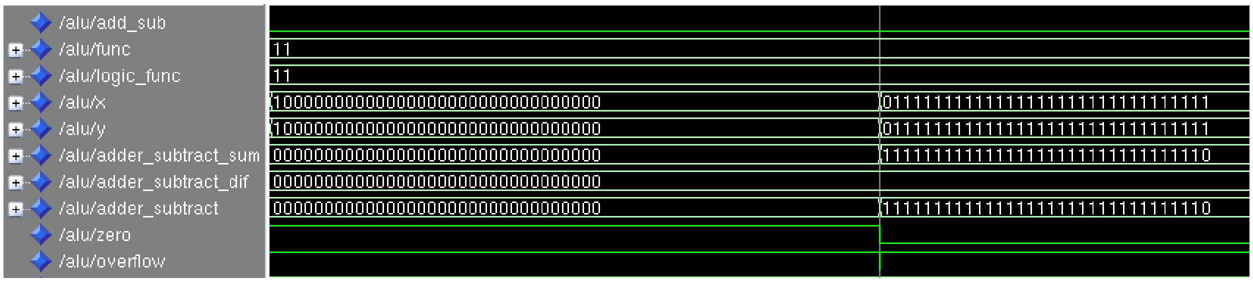


Above is the zero output being set to ‘1’ when two equal numbers being added and subtracted sum to ‘0’.

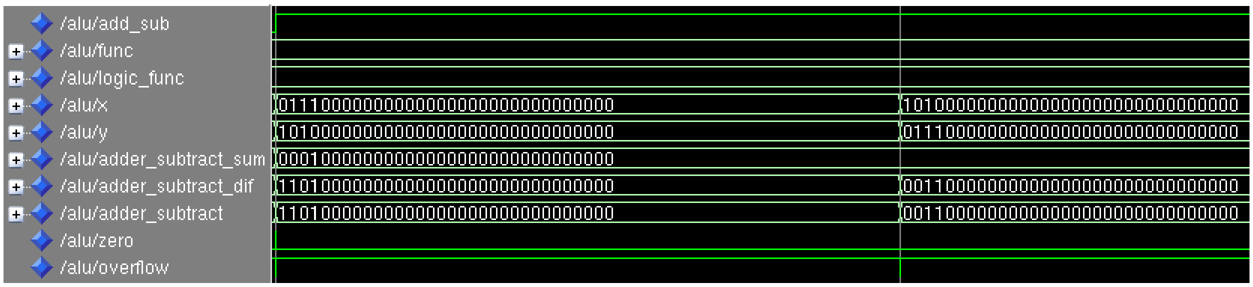
```
-- overflow detection logic
bool_sum <= ((NOT x(31)) AND (NOT y(31)) AND adder_subtract(31)) OR ( x(31) AND y(31) AND (NOT adder_subtract(31)));
bool_dif <= ((x(31)) AND (NOT y(31)) AND (NOT adder_subtract(31))) OR ((NOT x(31)) AND (y(31)) AND (adder_subtract(31)));

-- overflow bit output
process(bool_sum,bool_dif,add_sub)
begin
    if (add_sub = '0' AND bool_sum = '1') then
        overflow <= '1';
    elsif (add_sub = '1' AND bool_dif = '1') then
        overflow <= '1';
    else
        overflow <= '0';
    end if;
end process;
```

Above is the overflow logic code snippet that sets the overflow bit to ‘1’ or ‘0’ based on the Boolean logic determined by a truth table and kmap minimization.



Above are the test cases for overflow detection when summing. This occurs when MSBs of x and y are the same but the MSB of the output is different.



Above are the test cases for the overflow detection when performing subtraction. This occurs when x and y are of opposite sign but the output is opposite the sign of the y input (Example: +7 – (-6) = +13 normally, but if restricted to 4 bits then it would be: +7 – (-6) = -3 by overflow). This example was used to test the overflow in this case, using the highest 4 bits of the inputs and outputs.

The RTL schematic and precision log are attached to the lab report at the back. As for discussion of the precision log, there were 2 warnings. The warnings were to say that 2 unused signals are in the design; this was due to me forgetting to remove them after optimizing some VHDL code.

There was difficulty at first with performing the overflow and zero detection due to some confusion with the arithmetic being done in signed notation but this was overcome after time spent thinking over the problem.

Conclusion:

According to the Modelsim simulation results, the design works according to spec. The ALU was designed, simulated and synthesized properly without major issues.