Embedded Systems
Lab Report #2
Alexander Riveron
3/7/19

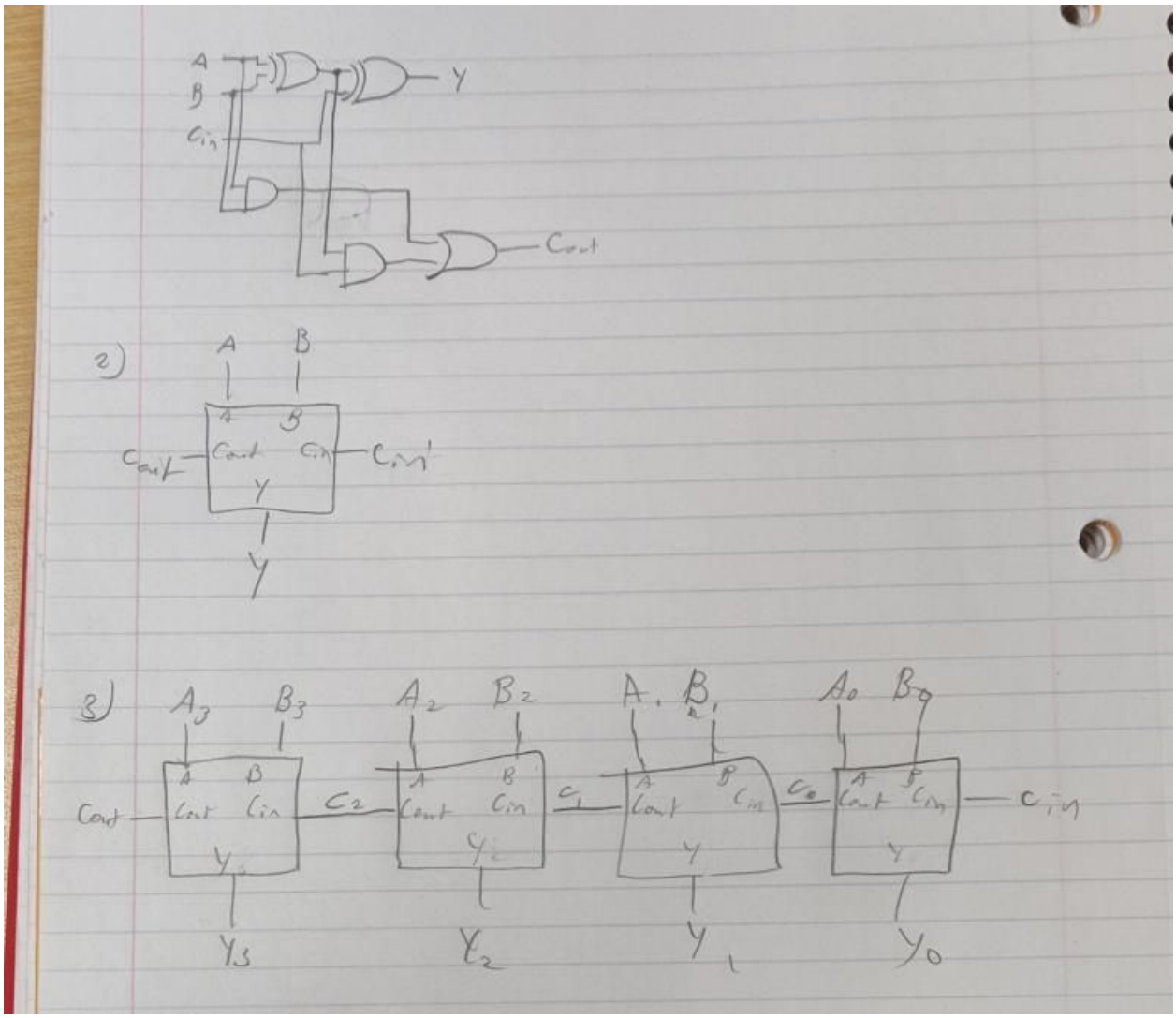
Purpose:

The purpose of this lab is to design a ripple adder from a series of full adders, and to implement an ALU on a Zybo.

Pre Lab

$Y = A \oplus B \oplus C_{in}$

$C_{out} = A \square B + C_{in} \square (B \oplus A)$



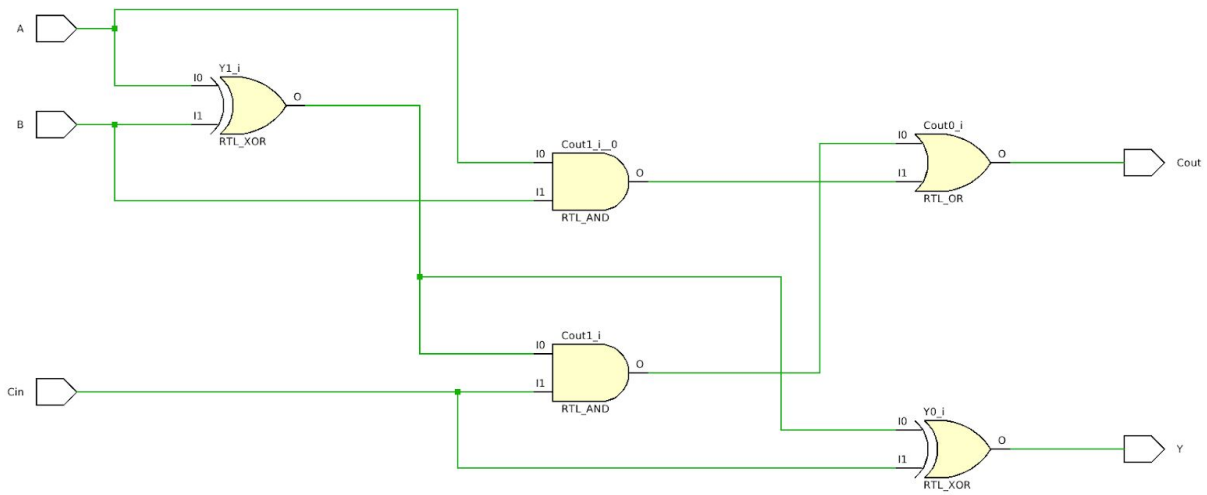
1) The Full adder takes in 3 inputs, A, B, and Cin, and outputs Y and Cout based on the following truth table

Cin	A	B	Y	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

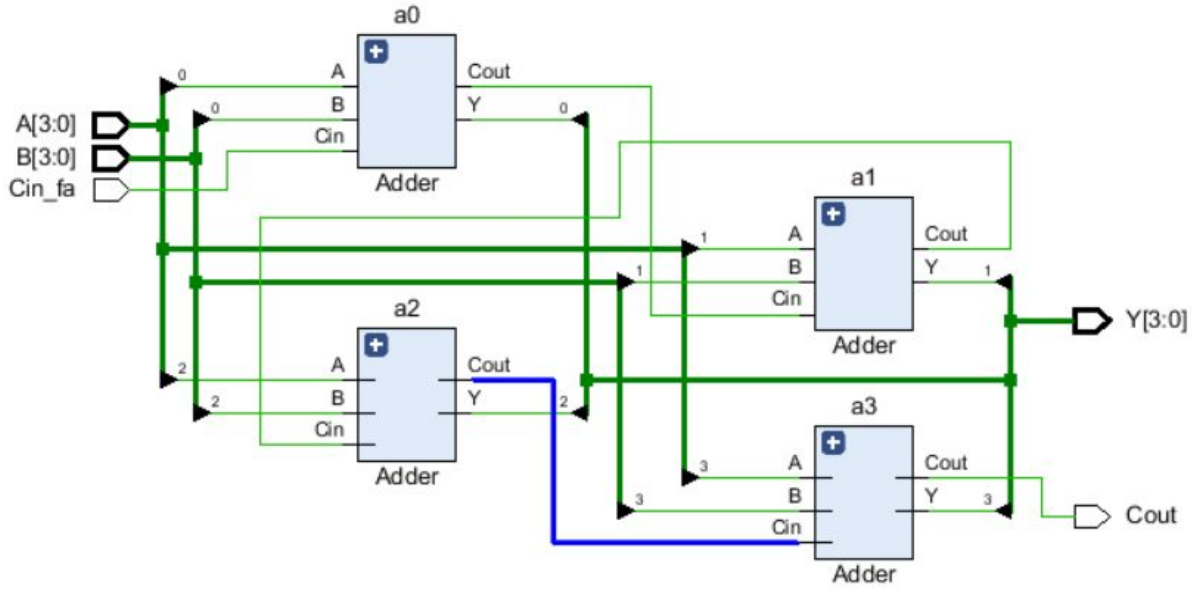
2)

Full Adder:

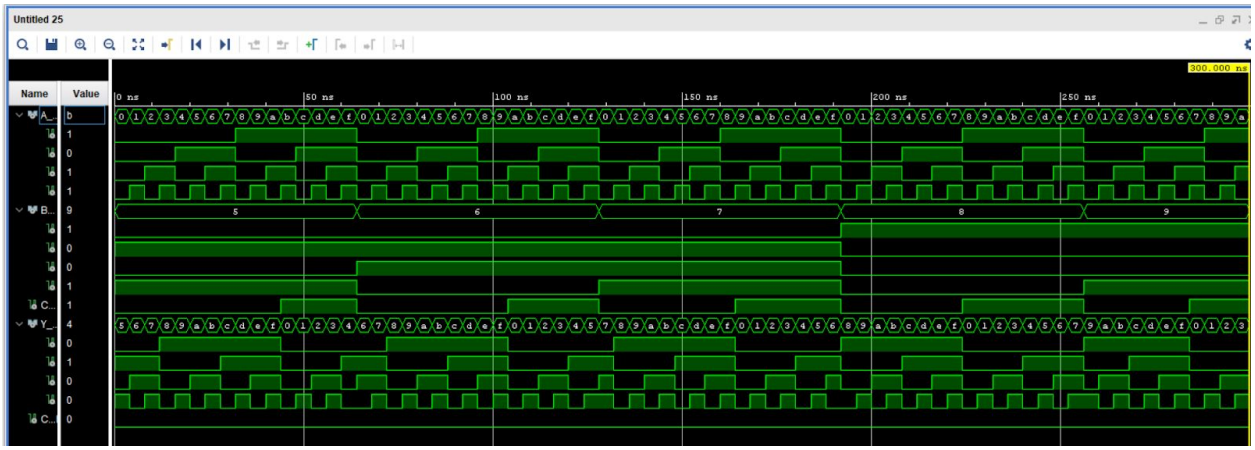
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Adder is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           Cin : in STD_LOGIC;
           Y : out STD_LOGIC;
           Cout : out STD_LOGIC);
end Adder;

architecture Adder of Adder is

signal temp1, temp2, temp3 : STD_LOGIC;

begin
    process (A,B,Cin)
    begin
        Y <= (A xor B) xor Cin;
        --temp1 <= A xor B;
        --temp2 <= temp1 and Cin;
        --temp3 <= A and B;
        Cout <= (((A xor B) and Cin) or (A and B));
        --Cout <= temp2 or temp3;
end process;
end Adder;
```



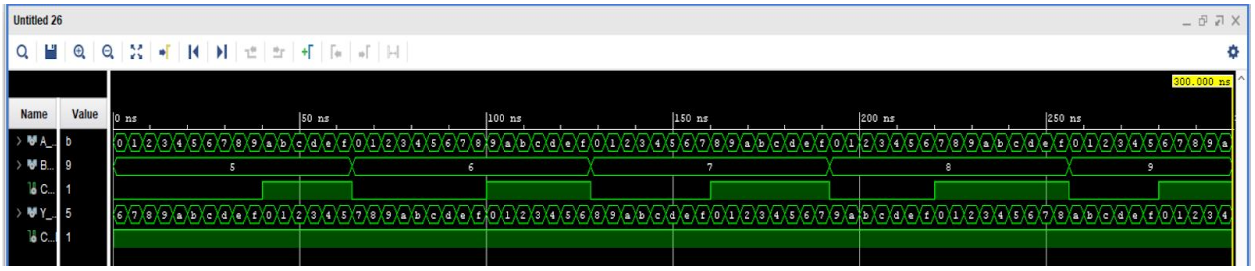
Ripple Carry Adder:



Cin = 0



Cin = 1



When originally testing the full adders, I did not treat Cin as a variable in the main process, doing so caused many issues in the output of the ripple adder. This simple mistake costed me many hours in debugging time, but this is an important lesson learned for future more ambitious projects.

2) Somebody Did the Work Already

The ALU receives 2 inputs that are to be processed based on a third input that determines the operation that is to occur. The ALU is to be ran on a zybo where 4 bit inputs A, B, and opcode values are determined on different switches, and the user determines which input is to be used by pressing a button. Pressing more than one button or for not long enough results in no process due to the debouncer.

my_alu:

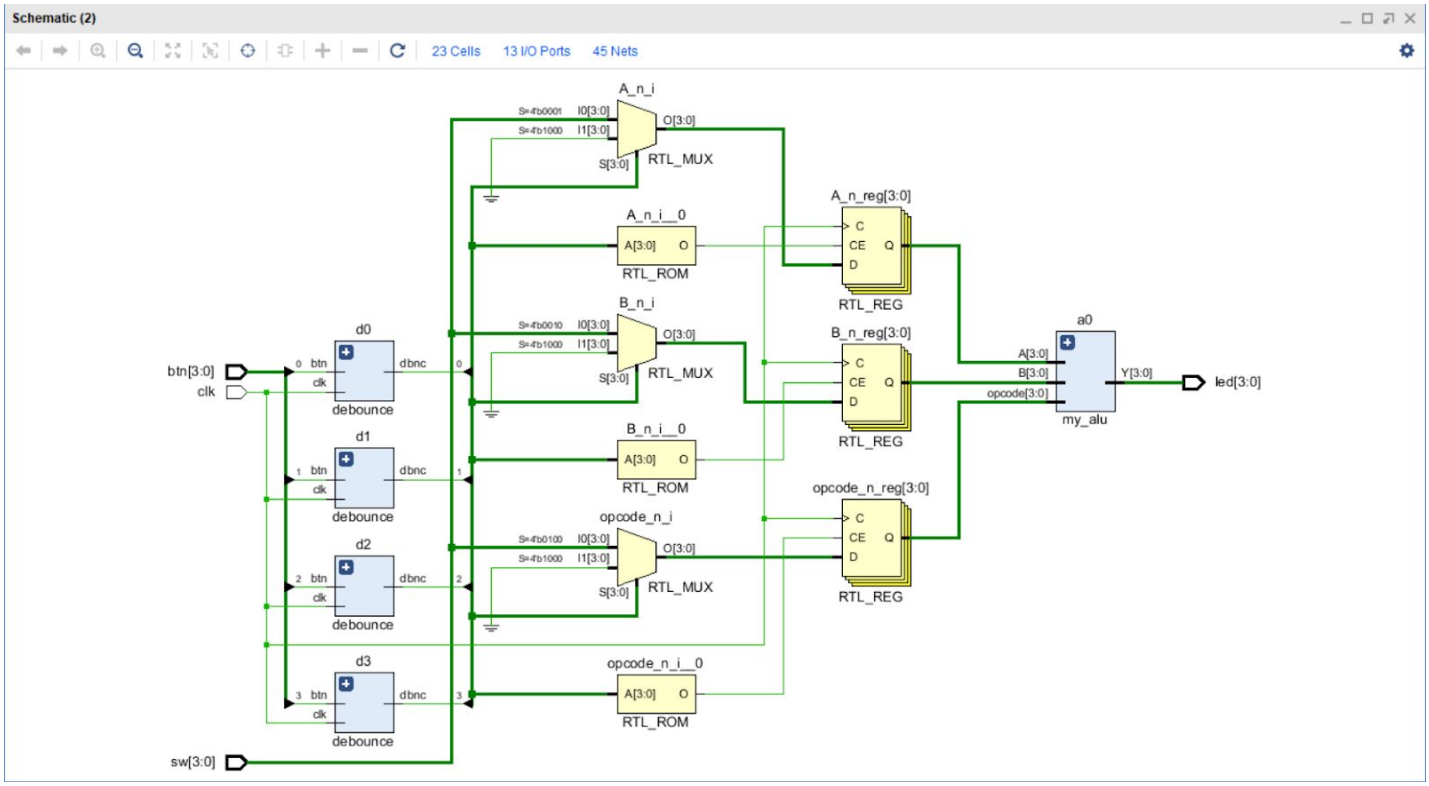
```
entity my_alu is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           opcode : in STD_LOGIC_VECTOR (3 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end my_alu;

architecture Behavioral of my_alu is

begin
process (A,B,Opcode)
    begin

    case (opcode) is
    when "0000" => Y <= A+B;
    when "0001" => Y <= A-B;
    when "0010" => Y <= A+1;
    when "0011" => Y <= A-1;
    when "0100" => Y <= 0-A;
    when "0101" => if (A>B) then
                     Y <= "0001";
                   else
                     Y <= "0000";
                   end if;
    when "0110" => Y <= A(2 downto 0)&'0';
    when "0111" => Y <= '0' & A(3 downto 1);
    when "1000" => Y <= '1' & A(3 downto 1);
    when "1001" => Y <= not A;
    when "1010" => Y <= A and B;
    when "1011" => Y <= A or B;
    when "1100" => Y <= A xor B;
    when "1101" => Y <= A xnor B;
    when "1110" => Y <= A nand B;
    when "1111" => Y <= A nor B;
    end case;
end process;

end Behavioral;
```

alu_tester:

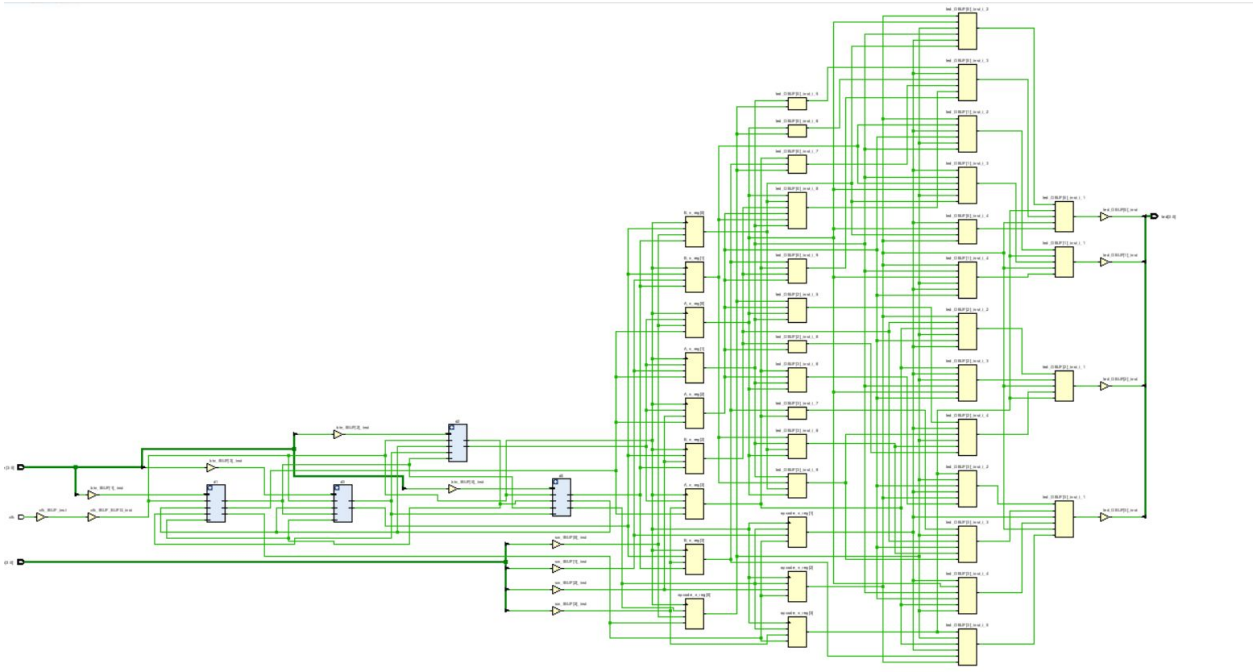
```
entity alu_tester is
    Port ( sw : in STD_LOGIC_VECTOR (3 downto 0);
           btn : in STD_LOGIC_VECTOR (3 downto 0);
           clk : in STD_LOGIC;
           led : out STD_LOGIC_VECTOR (3 downto 0));
end alu_tester;

architecture Behavioral of alu_tester is


component debounce is
    Port ( btn : in STD_LOGIC;
      clk : in STD_LOGIC;
      dbnc : out STD_LOGIC);
 end component;

 component my_alu is
    Port (
           A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           opcode : in STD_LOGIC_VECTOR (3 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

    signal db_bt, A_n, B_n, opcode_n : STD_LOGIC_VECTOR (3 downto 0);
begin

    process(btn, sw, clk)
    begin
    if (rising_edge(clk)) then
        case (db_bt) is
            when "0001" => A_n <= sw;
            when "0010" => B_n <= sw;
            when "0100" => opcode_n <= sw;
            when "1000" => A_n <= "0000";
                B_n <= "0000";
                opcode_n <= "0000";
            when others => A_n <= A_n;
                B_n <= B_n;
                opcode_n <= opcode_n;
```

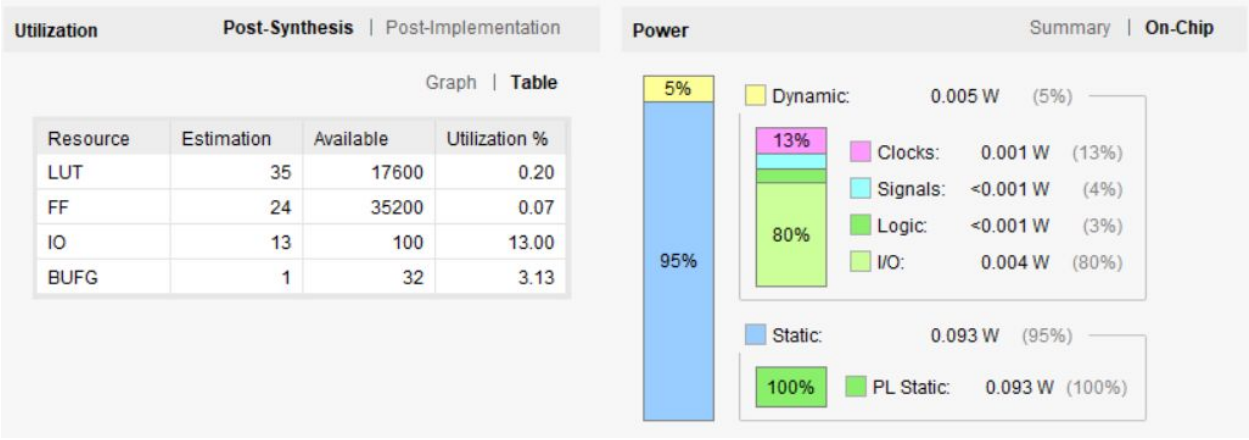
```
        end case;
    end if;
    end process;

    d0: debounce port map (
        btn => btn(0),
        clk => clk,
        dbnc => db_bt(0));
    d1: debounce port map (
            btn => btn(1),
            clk => clk,
            dbnc => db_bt(1));
    d2: debounce port map (
            btn => btn(2),
            clk => clk,
            dbnc => db_bt(2));
    d3: debounce port map (
            btn => btn(3),
            clk => clk,
            dbnc => db_bt(3));
    a0: my_alu port map (
            A => A_n,
            B => B_n,
            opcode => opcode_n,
            Y => led);
end Behavioral;
```

Elaborated Schematic



Synthesis Schematic





The XDC chip required clk, and all 4 bits of the btn, sw, and led to work on the zybo

```
##Clock signal
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L11P_T1_SRCC_35 Sch=sysclk
create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];


##Switches
set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { sw[0] }]; #IO_L19N_T3_VREF_35 Sch=SW0
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { sw[1] }];  #IO_L24P_T3_34 Sch=SW1
set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports { sw[2] }]; #IO_L4N_T0_34 Sch=SW2
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; #IO_L9P_T1_DQS_34 Sch=SW3


##Buttons
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { btn[0] }]; #IO_L20N_T3_34 Sch=BTN0
set_property -dict { PACKAGE_PIN P16   IOSTANDARD LVCMOS33 } [get_ports { btn[1] }]; #IO_L24N_T3_34 Sch=BTN1
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { btn[2] }]; #IO_L18P_T2_34 Sch=BTN2
set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports { btn[3] }]; #IO_L7P_T1_34 Sch=BTN3


##LEDs
set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { led[0] }]; #IO_L23P_T3_35 Sch=LED0
set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { led[1] }]; #IO_L23N_T3_35 Sch=LED1
set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; #IO_0_35=Sch=LED2
set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; #IO_L3N_T0_DQS_AD1N_35 Sch=LED3
```

This lab taught us how VHDL has a library set that is optimally programed to handle simple operations for the user. If we were to design all of the operations along with the ALU, much more unnecessary time for coding and debugging would be needed complete a simple ALU.