POST LAB LAB 2

Embedded Systems Spring 2019

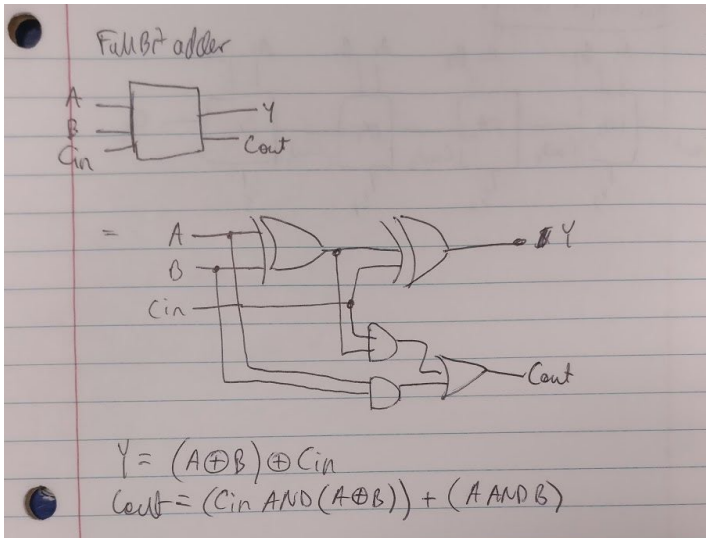
Adam Falkowski

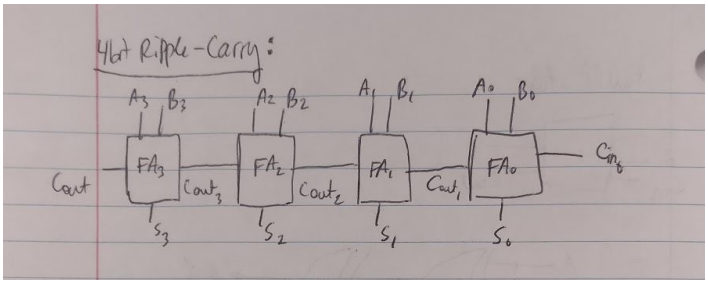
3/7/2019

Purpose:
In this lab we explored Full (ripple) adders and a customizable ALU. We also experimented with IF and case statements as well as different VHDL coding styles, most notable structural and behavioral.

Pre-Lab Questions:

Write the logic equations for a single bit full adder with inputs A, B, Cin, and outputs Y, Cout. Draw a black box diagram for the single bit full adder described above. Draw a block diagram of a 4-bit ripple-carry adder made of single bit full adders, with 4 bit inputs A and B, a single bit input Cin, a 4-bit output S, and a single bit output Cout.



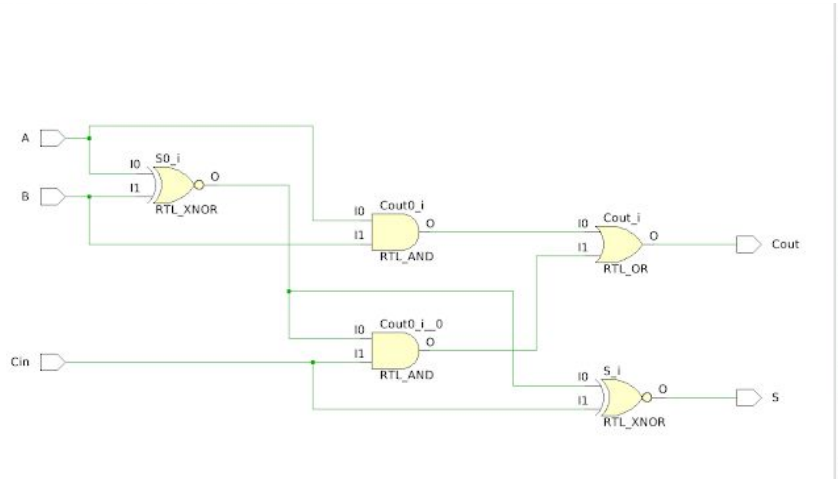


Single Full Bit Adder:
The FB adder should take in 3 inputs: A, B and Cin and outputs two signals, out and Cout. Both the carry in and carry out both work with overflow digits.

VHDL Code:

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


entity Adder is
    Port (
          signal A : in std_logic;
          signal B : in std_logic;
          signal Cin : in std_logic;
          S,Cout : out std_logic
           );
end Adder;

architecture Adder_arc of Adder is

begin

   S <= ((A XNOR B) XNOR Cin);
   Cout <= (A AND B) OR ((A XNOR B) AND Cin);
end Adder_arc;
```

RTL:



Ripple Carry Adder:

Similar function as the full adder except there are now 4 adders cascaded where the cout is connected to the corresponding cin for the next adder.

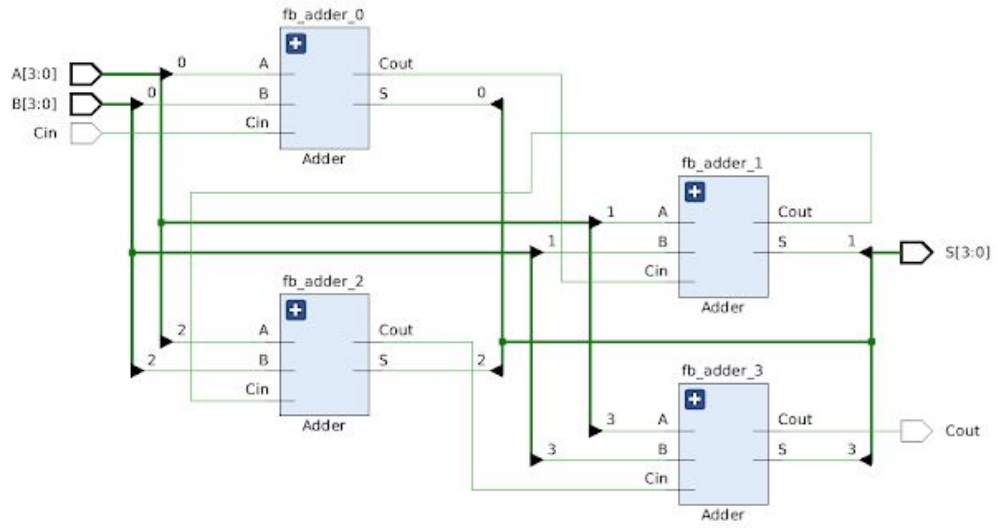
VHDL Code

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;



entity Ripple_Carry_Adder is
    Port (
            signal A, B : in std_logic_vector(3 downto 0);
            signal S : out std_logic_vector(3 downto 0);
            signal Cin: in std_logic;
            signal Cout: out std_logic
           );
end Ripple_Carry_Adder;

architecture Ripple_Carry_Adder_arc of Ripple_Carry_Adder is
    signal c1,c2,c3 : std_logic;

    component Adder is
        port (
            signal A : in std_logic;
            signal B : in std_logic;
            signal Cin : in std_logic;
            S,Cout : out std_logic
            );
    end component;
begin
    fb_adder_0: adder
        port map(
                A => A(0),
                B => B(0),
                Cin => Cin,
                Cout => C1,
                S => S(0)
                );
     fb_adder_1: adder
        port map(
                A => A(1),
                B => B(1),
                Cin => C1,
                Cout => C2,
                S => S(1)
                );
      fb_adder_2: adder
        port map(
```

```
            A => A(2),
            B => B(2),
            Cin => C2,
            Cout => C3,
            S => S(2)
            );
    fb_adder_3: adder
      port map(
            A => A(3),
            B => B(3),
            Cin => C3,
            Cout => Cout,
            S => S(3)
            );

end Ripple_Carry_Adder_arc;
```

RTL:



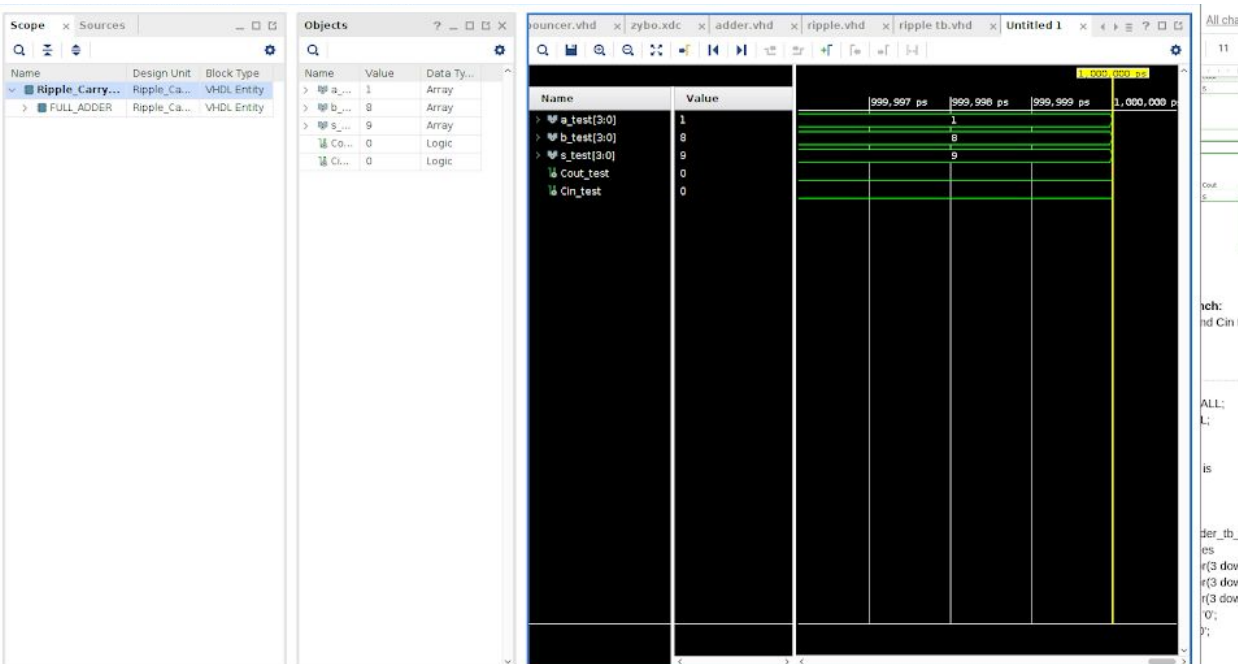
Ripple Carry Adder Test Bench:
I set specific values for A, B and Cin to see if they are added together correctly.

VHDL Code:

```
library IEEE;
```

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


entity Ripple_Carry_Adder_tb is
end Ripple_Carry_Adder_tb;


architecture Ripple_Carry_Adder_tb_arc of Ripple_Carry_Adder_tb is
--test signals and the test values
signal a_test : std_logic_vector(3 downto 0) := "0001";
signal b_test : std_logic_vector(3 downto 0) := "1000";
signal s_test : std_logic_vector(3 downto 0) := "0000";
signal Cout_test : std_logic := '0';
signal Cin_test : std_logic := '0';


component Ripple_Carry_Adder is
        Port (
            signal A, B : in std_logic_vector(3 downto 0);
            signal S : out std_logic_vector(3 downto 0);
            signal Cin: in std_logic;
            signal Cout: out std_logic
           );
end component;
begin
FULL_ADDER: Ripple_Carry_Adder
            port map (
                        A=>a_test,
                        B=>b_test,
                        S => s_test,
                        Cin => Cin_test,
                        Cout => Cout_test
                        );

-- tested by using different values for a_test, b_test, anc C_in to see if S had the proper result


end Ripple_Carry_Adder_tb_arc;
```

Simulation:



ALU:
I created a small custom ALU that was capable of 16 different operations. I also instantiated the previous debounce device to combat any mechanical bouncing. I also used a ALU_tb file to connect all the pieces (called test bench but doesn't act like a test bench; a test file)

VHDL Code:

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


entity Ripple_Carry_Adder_tb is
end Ripple_Carry_Adder_tb;


architecture Ripple_Carry_Adder_tb_arc of Ripple_Carry_Adder_tb is
--test signals and the test values
signal a_test : std_logic_vector(3 downto 0) := "0001";
signal b_test : std_logic_vector(3 downto 0) := "1000";
signal s_test : std_logic_vector(3 downto 0) := "0000";
signal Cout_test : std_logic := '0';
signal Cin_test : std_logic := '0';
```

```
component Ripple_Carry_Adder is
        Port (
            signal A, B : in std_logic_vector(3 downto 0);
            signal S : out std_logic_vector(3 downto 0);
            signal Cin: in std_logic;
            signal Cout: out std_logic
           );
end component;
begin
FULL_ADDER: Ripple_Carry_Adder
            port map (
                        A=>a_test,
                        B=>b_test,
                        S => s_test,
                        Cin => Cin_test,
                        Cout => Cout_test
                        );

-- tested by using different values for a_test, b_test, anc C_in to see if S had the proper result


end Ripple_Carry_Adder_tb_arc;
```

ALU Test File:

VHDL Code:

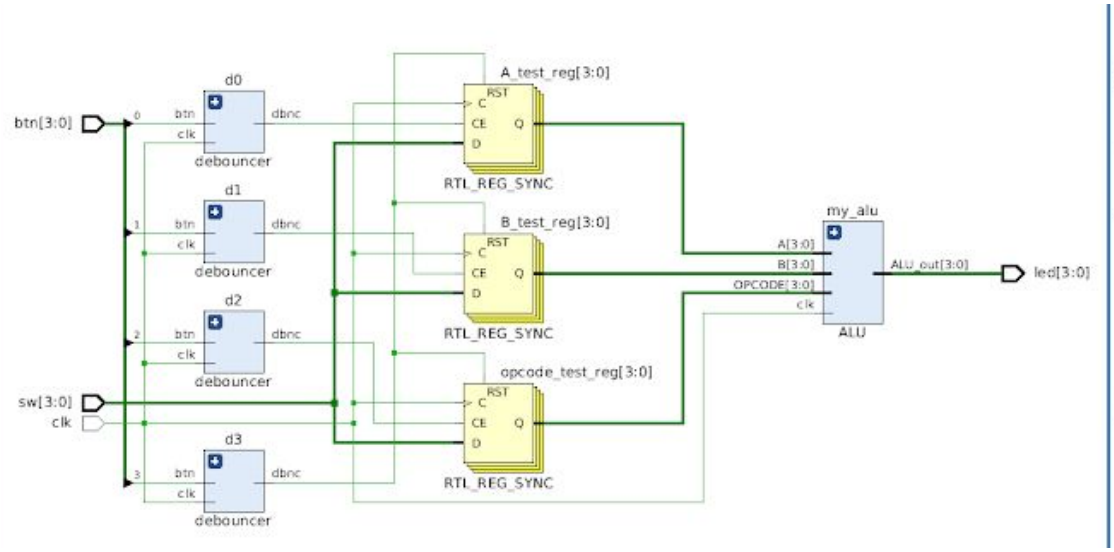
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity ALU_tb is
    Port (
            clk : in std_logic;
            led : out std_logic_vector (3 downto 0);
            btn : in std_logic_vector(3 downto 0);
            sw :in std_logic_vector(3 downto 0)
            );
end ALU_tb;

architecture ALU_arc_tb of ALU_tb is
```

```
signal opcode_test : std_logic_vector(3 downto 0);
signal A_test : std_logic_vector(3 downto 0):= "0001";
signal B_test: std_logic_vector(3 downto 0):= "1000";
signal alu_out_test: std_logic_vector(3 downto 0);
signal button_ad : std_logic_vector (3 downto 0); --button after debounce



        component debouncer
        Port
        (
        btn, clk : in std_logic;
        dbnc :out std_logic
         );
        end component;

        component ALU
        Port (
            A, B : in std_logic_vector(3 downto 0);
            OPCODE: in std_logic_vector (3 downto 0);
            ALU_out : out std_logic_vector(3 downto 0);
            clk :in std_logic
             );
         end component;

begin

         my_alu: ALU
            port map
                (
                clk => clk,
                OPCODE => opcode_test,
                A => a_test,
                B => b_test,
                ALU_out => led
                );
          d0: debouncer
             port map
                (
                btn => btn(0),
                dbnc => button_ad(0),
                clk => clk
                );
```

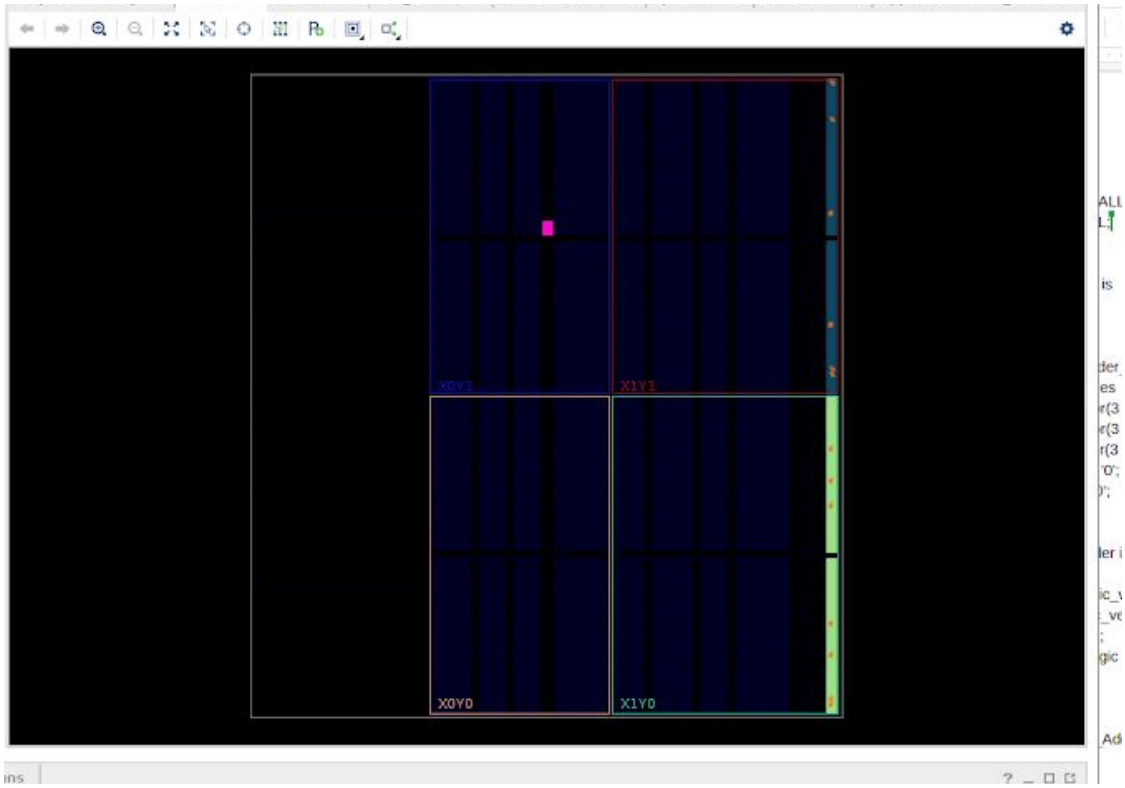
```
        d1: debouncer
          port map
            (
            btn => btn(1),
            dbnc => button_ad(1),
            clk => clk
            );
          d2: debouncer
          port map
            (
            btn => btn(2),
            dbnc => button_ad(2),
            clk => clk
            );
          d3: debouncer
          port map
            (
            btn => btn(3),
            dbnc => button_ad(3),
            clk => clk
            );

process(clk)
begin
if rising_edge(clk) then
   if (button_ad(3) = '1') then --reset statement
      A_test  <= (others => '0');
      B_test  <= (others => '0');
      opcode_test <= (others => '0');

   else
      if (button_ad(2) = '1') then
      opcode_test <= sw;
      end if;
      if (button_ad(1) = '1') then
      B_test <= sw;
      end if;
      if (button_ad(0) ='1') then
      A_test <= sw;
      end if;

   end if;
end if;
```

```
end process;
end ALU_arc_tb;
```

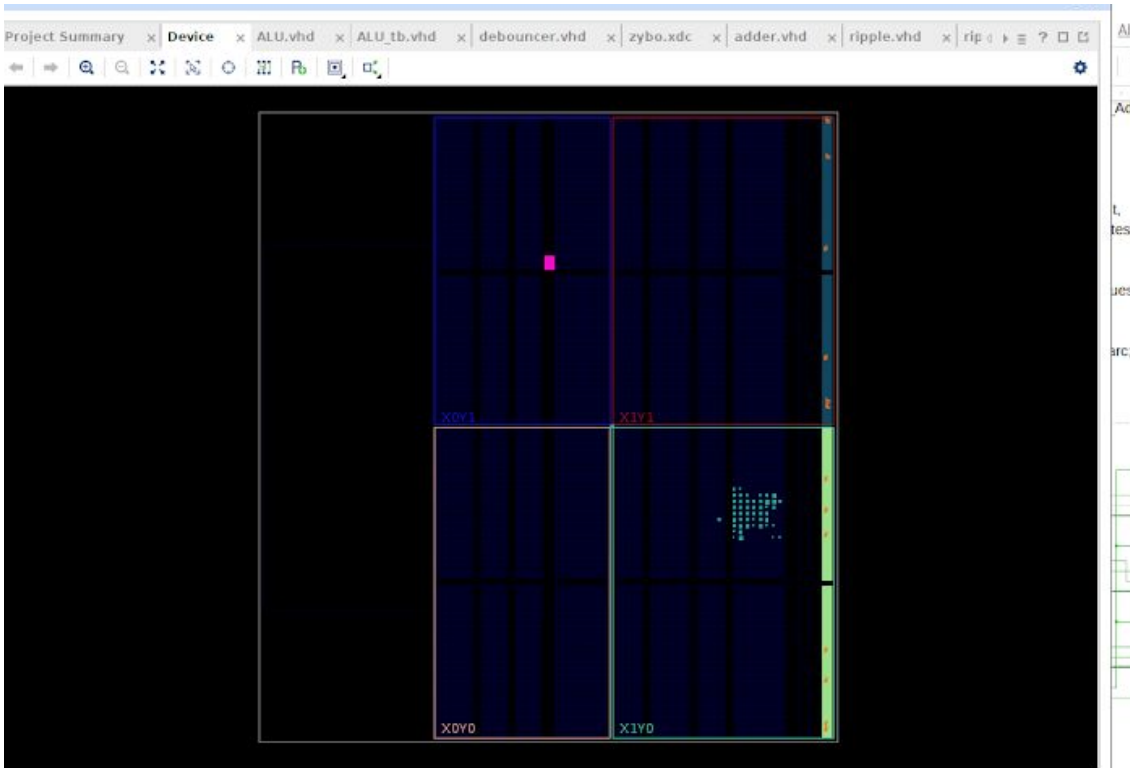
RTL for ALU:



Synthesized design for ALU:



Implemented Design:



XDC file:

```
## This file is a general .xdc for the ZYBO Rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used signals according to the project


##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


##I2S Audio Codec
```

Only the Switches, buttons and leds are uncommented because we only use those components.

Summary:
Lab 2 was a bit easier then lab 1 but nevertheless proved to be challenging. I realized that having your code and simulations right doesn't always guarantee the code will work on the Zybo Board. After Several hours trying to get the code to execute properly on the Zybo board I was able to successfully implement the ALU.