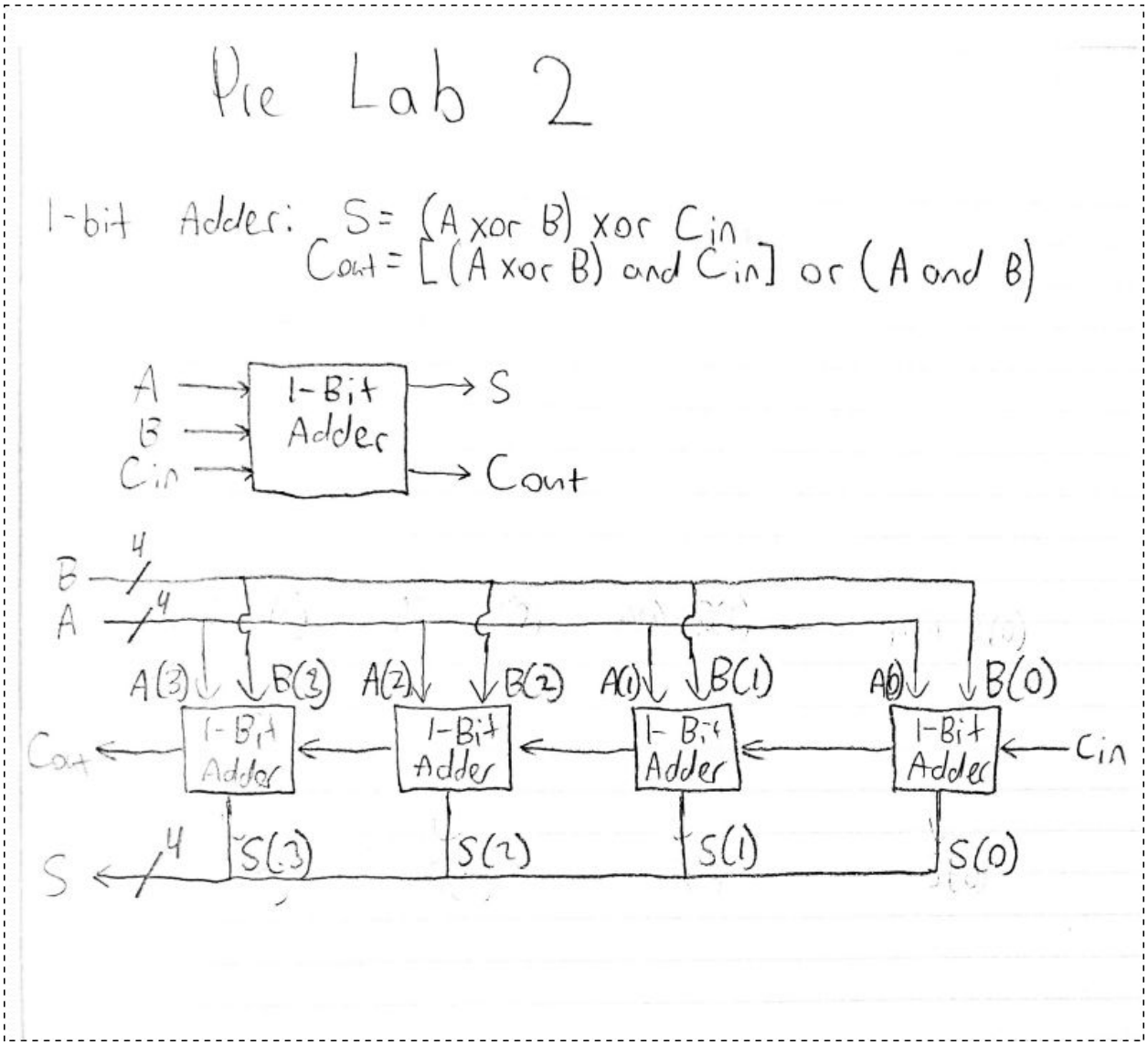
Embedded Systems 1

Lab Report 2

Gavin McKim
3/07/2019

Pre Lab:



Purpose:

The purpose of this lab was to get used to using math library functions in vhdl. In the first part, the lab displayed how these functions worked on a bit wise level by having us create a 1-bit adder. Then we were showed how this 1-bit function can be extended to multiple bits by building a 4-bit adder. After understanding how these functions work, the second part of this lab aims to teach how to use the built in math functions. Understanding how these functions work allows us to avoid having to build them from scratch each time.

1. Back to Digital Logic Design

Theory of Operation:

For the first part of this section, we created a 1-bit adder. This adder had inputs for the two added bits as well as an input for a carry in bit. The carry in bit adds a “+1” to the summation. It outputted the sum of the two inputs as well as a carry out bit. The carry out bit is for when the sum of the two inputs overflow the one bit output. This circuit was implemented using a basic logic circuit containing ANDs, ORs, and XORs. The circuit should properly add the two bit inputs and properly calculate the sum and carry out while also taking into account the carry in bit. Next, we cascaded four of these 1-bit adders to create a 4-bit ripple adder. Each 1-bit adder calculates the sum of the nth-bit of the 4-bit inputs. These sums are concatenated together to find the 4-bit sum. The carry in bit of one block is taken from the carry out of the previous block. The whole circuit also take a carry in input and outputs a carry out. The circuit should properly calculate the carry out and the sum of two 4-bit inputs while also taking into account the carry in bit.

1-Bit Adder Code:

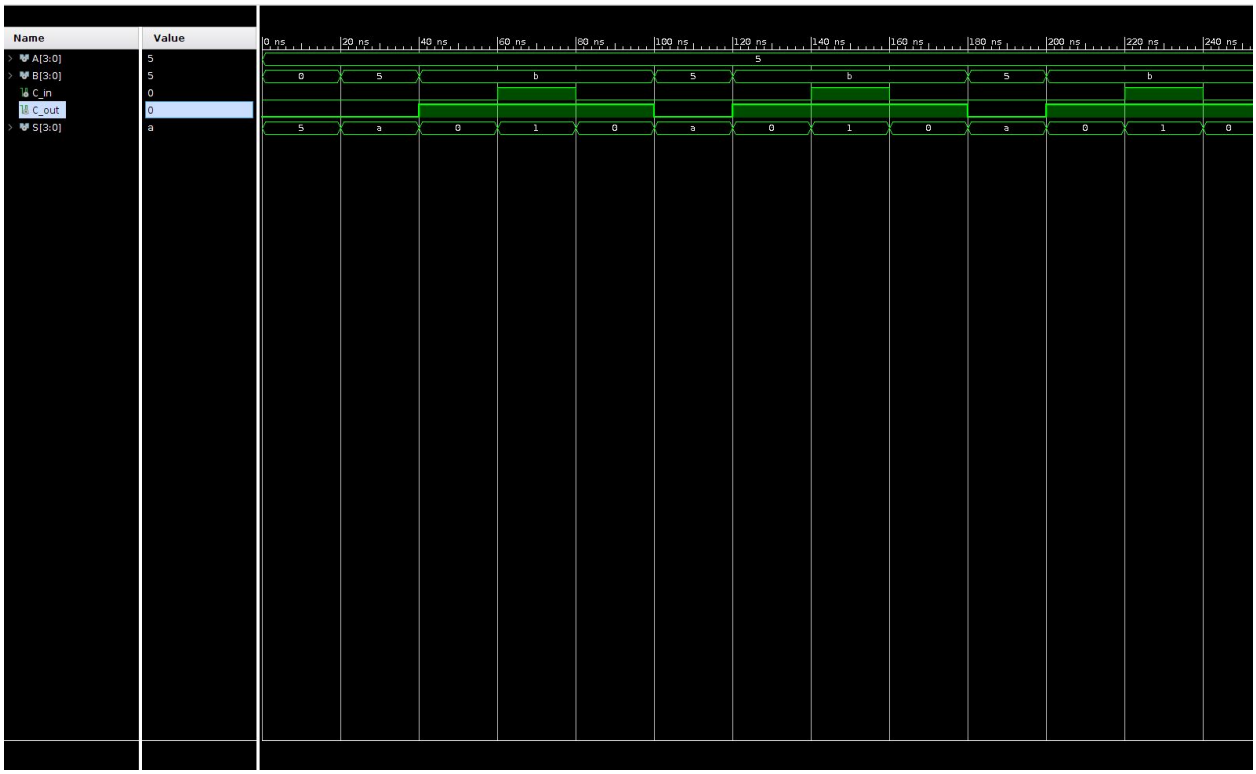
```
1  ----------------------------------------------------------------------
2
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6
7
8  entity adder_one is
9    Port (A, B, C_in : in std_logic;
10         S, C_out : out std_logic );
11 end adder_one;
12
13 architecture Behavioral of adder_one is
14 signal sum : std_logic;
15 begin
16 sum <= A xor B;
17 S <= sum xor C_in;
18 C_out <= (sum and C_in) or (A and B);
19
20 end Behavioral;
```

Ripple_Adder(4-Bit Adder) Code:

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

entity ripple_adder is
  port (
    A_in, B_in : in std_logic_vector(3 downto 0);
    C_in0 : in STD_LOGIC;
    C_out3 : out STD_LOGIC;
    S_out : out std_logic_vector(3 downto 0)
  );
end ripple_adder;

architecture Behavioral of ripple_adder is
  component adder_one is
  port (
    A : in STD_LOGIC;
    B : in STD_LOGIC;
    C_in : in STD_LOGIC;
    S : out STD_LOGIC;
    C_out : out STD_LOGIC
  );
  end component adder_one;

signal C : std_logic_vector(2 downto 0);
begin
adder_one_0: component adder_one
     port map (
      A => A_in(0),
      B => B_in(0),
      C_in => C_in0,
      C_out => C(0),
      S => S_out(0)
    );
adder_one_1: component adder_one
     port map (
      A => A_in(1),
     B => B_in(1),
     C_in => C(0),
     C_out => C(1),
     S => S_out(1)
    );
adder_one_2: component adder_one
     port map (
      A => A_in(2),
     B => B_in(2),
     C_in => C(1),
     C_out => C(2),
     S => S_out(2)
   );
adder_one_3: component adder_one
     port map (
      A => A_in(3),
     B => B_in(3),
     C_in => C(2),
     C_out => C_out3,
     S => S_out(3)
   );
end Behavioral;
```

Testbench Code:

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

entity ripple_adder is
  port (
    A_in, B_in : in std_logic_vector(3 downto 0);
    C_in0 : in STD_LOGIC;
    C_out3 : out STD_LOGIC;
    S_out : out std_logic_vector(3 downto 0)
  );
end ripple_adder;

architecture Behavioral of ripple_adder is
  component adder_one is
  port (
    A : in STD_LOGIC;
    B : in STD_LOGIC;
    C_in : in STD_LOGIC;
    S : out STD_LOGIC;
    C_out : out STD_LOGIC
  );
  end component adder_one;

signal C : std_logic_vector(2 downto 0);
begin
adder_one_0: component adder_one
     port map (
      A => A_in(0),
      B => B_in(0),
      C_in => C_in0,
      C_out => C(0),
      S => S_out(0)
    );
adder_one_1: component adder_one
     port map (
      A => A_in(1),
     B => B_in(1),
     C_in => C(0),
     C_out => C(1),
     S => S_out(1)
    );
adder_one_2: component adder_one
     port map (
      A => A_in(2),
     B => B_in(2),
     C_in => C(1),
     C_out => C(2),
     S => S_out(2)
   );
adder_one_3: component adder_one
     port map (
      A => A_in(3),
     B => B_in(3),
     C_in => C(2),
     C_out => C_out3,
     S => S_out(3)
   );
end Behavioral;

```



2. Somebody did the work already

Theory of Operation:

In this part of the lab, we created a 16-bit Arithmetic Logic Unit(ALU). The ALU can perform 16 functions depending on the value of a 4-bit opcode input signal. The ALU also takes two 4-bit signals as inputs and outputs a function of those inputs based on what opcode was sent. This was implemented using a case statement to determine what function to perform. Ideally it should choose the correct function based on the opcode and then output the correct result. Next, this ALU was implemented in a top level design to be imported to the Zybo board. Each of the three inputs to the ALU was connected to a button. When a button was pressed on a rising edge of the clock, the value represented in the switches would be loaded into the buttons respective signal. The fourth button was used to reset all the inputs to “0000”. The output of the selected function will be displayed using the LED’s. The debouncer code from Lab 1 was also utilized to get an accurate button input from the board. Ideally the top level design should be able to be successfully migrated to the board and be able to perform all functions in the ALU regardless of input.

My_alu Code:

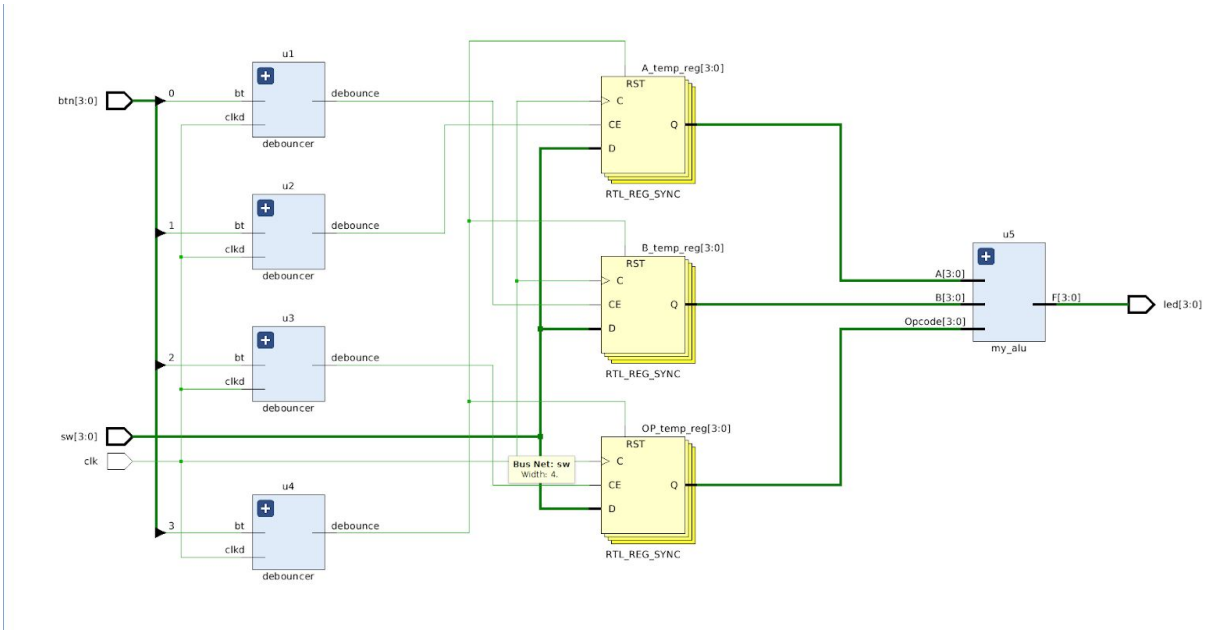
```
----------------------------------------------------------------------------------

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


entity my_alu is
  Port (A, B : in std_logic_vector (3 downto 0);
        Opcode : in std_logic_vector(3 downto 0);
        F : out std_logic_vector(3 downto 0));
end my_alu;

architecture Behavioral of my_alu is

begin
p: process(A, B, Opcode)
begin
    case(Opcode) is
    when "0000" => F <= std_logic_vector(unsigned(A) + unsigned(B));
    when "0001" => F <= std_logic_vector(unsigned(A) - unsigned(B));
    when "0010" => F <= std_logic_vector(unsigned(A) + 1);
    when "0011" => F <= std_logic_vector(unsigned(A) - 1);
    when "0100" => F <= std_logic_vector(0 - unsigned(A));
    when "0101" => if(A > B) then F <= "0001";
                   else F <= "0000";
                   end if;
    when "0110" => F <= std_logic_vector(unsigned(A) sll 1);
    when "0111" => F <= std_logic_vector(unsigned(A) srl 1);
    when "1000" => F <= '1' & A(3 downto 1);
    when "1001" => F <= NOT A;
    when "1010" => F <= A AND B;
    when "1011" => F <= A OR B;
    when "1100" => F <= A XOR B;
    when "1101" => F <= A XNOR B;
    when "1110" => F <= A NAND B;
    when "1111" => F <= A NOR B;
    end case;
end process;
end Behavioral;

```

Alu_tester Code:

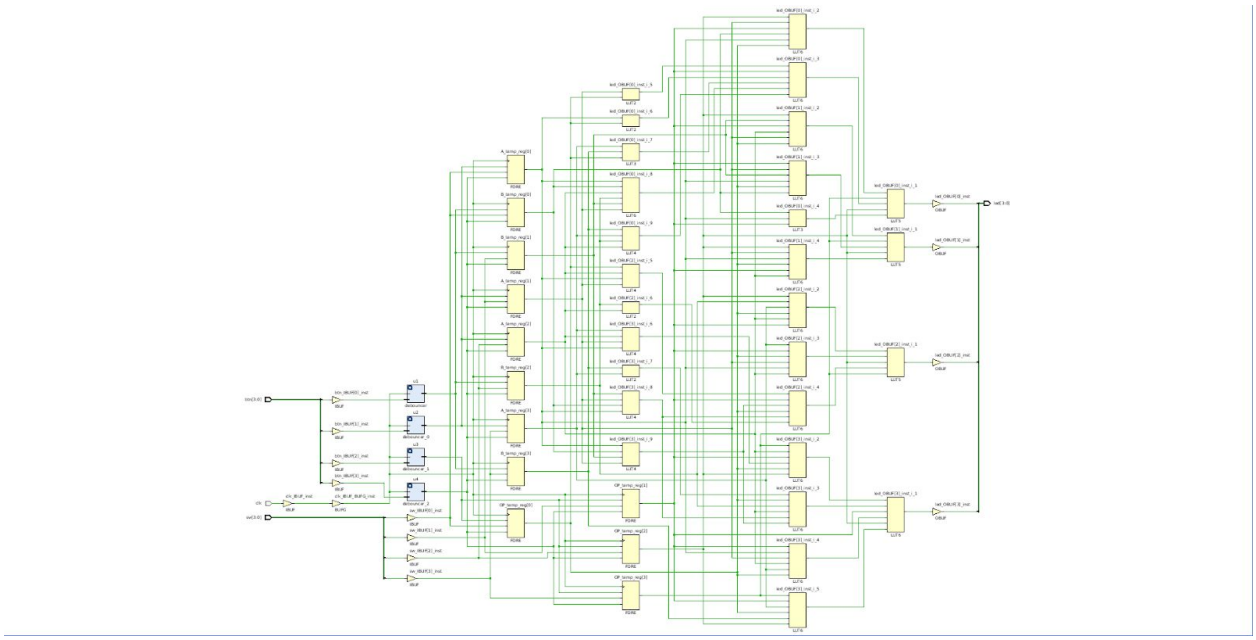
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity alu_tester is
  Port (sw, 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 my_alu is
  Port (A, B : in std_logic_vector (3 downto 0);
        Opcode : in std_logic_vector(3 downto 0);
        F : out std_logic_vector(3 downto 0));
end component;

component debouncer is
 Port ( clkd, bt : in std_logic;
        debounce: out std_logic);
end component;

signal value : std_logic_vector (3 downto 0);
signal dbnc : std_logic_vector (3 downto 0);
signal A_temp, B_temp, OP_temp : std_logic_vector (3 downto 0);
begin

process(clk)
begin
if rising_edge(clk) then
    if(dbnc(3) = '1') then
        A_temp <= (others => '0');
        B_temp <= (others => '0');
        OP_temp <= (others => '0');
    else
        if(dbnc(0) = '1') then B_temp <= sw; end if;
        if(dbnc(1) = '1') then A_temp <= sw; end if;
        if(dbnc(2) = '1') then OP_temp <= sw; end if;
    end if;
end if;
end process;

u1: debouncer
port map ( bt => btn(0),
           clkd => clk,
           debounce => dbnc(0));

u2: debouncer
port map ( bt => btn(1),
           clkd => clk,
           debounce => dbnc(1));

u3: debouncer
port map ( bt => btn(2),
           clkd => clk,
           debounce => dbnc(2));

u4: debouncer
port map ( bt => btn(3),
           clkd => clk,
           debounce => dbnc(3));

```

```
65  u5: my_alu
66  port map ( A => A_temp,
67             B => B_temp,
68             Opcode => OP_temp,
69             F => led);
70
71
72
73  end Behavioral;
```

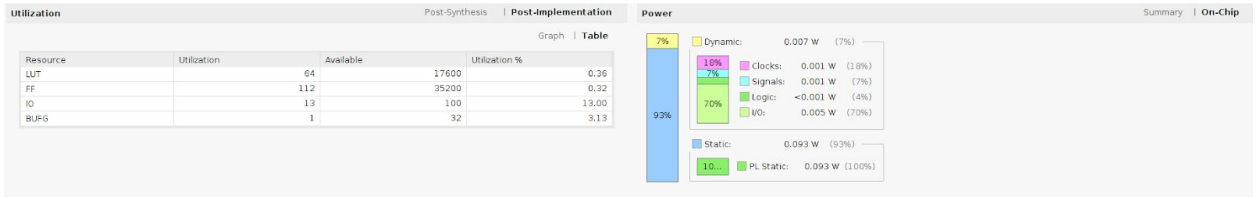
Elaboration Schematic:



Synthesis Schematic:



Power and Utilization:



Utilization — Post-Synthesis | **Post-Implementation** — Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	64	17600	0.36
FF	112	35200	0.32
IO	13	100	13.00
BUFG	1	32	3.13

Power — Summary | **On-Chip**

7%	Dynamic:	0.007 W	(7%)
10%	Clocks:	0.001 W	(18%)
7%	Signals:	0.001 W	(7%)
70%	Logic:	<0.001 W	(4%)
	I/O:	0.005 W	(70%)
93%	Static:	0.093 W	(93%)
10...	PL Static:	0.093 W	(100%)

Constraint File:

```
##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

```

Discussion:

Observations:

Coming into the lab, I already knew how Full Adders worked from previous classes. However, it was interesting seeing how they could be implemented using VHDL. The main thing I learned from this lab is how to use many of the math functions in VHDL. The lab helped me understand what input and output signal types that the operations worked with. For example, knowing when to cast signal to unsigned in order to perform certain operations.

Questions:

The main question this lab leaves is how to use the IP Integrator. When creating the 4-bit adder, I tried to experiment a little bit and use the IP integrator to assemble four 1-bit adders and compile code from the diagram. However, I could not get this design working in my testbench simulation so it would be cool to learn how to use this. I believe the IP integrator will help save a lot of time in the future.