- Lab 2
    - Embedded Systems
    - Lab Report 2
    - Sahana Asokan
    - 3-6-2019
- Purpose
    - The purpose of this lab is to learn more about ALUs. In this specific lab we will be learning about full adders and it will be implemented using basic Boolean logic. Later on in this lab we will describe the behavior of a 16 function ALU.
- Part 1 Adder
    - Theory of Operation
        - The single bit full adder is the first step and is based of a simple binary truth table, we will then use this single bit adder and create a 4-bit ripple carry adder.
        - If designed correctly the 4 bit ripple carry adder will follow the correct logic based of the karnaugh map and the intial single bit adder.
    - Design
        - VHDL Code for the adder

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

entity adder is
Port (A, B, C_in : in std_logic;
 S, C_out: out std_logic);
end adder;

architecture Behavioral of adder is

begin

S <= C_in xor (A xor B); C_out <= (C_in and (A xor B))
or (A and B);

end Behavioral;
```

        - VHDL code for ripple adder

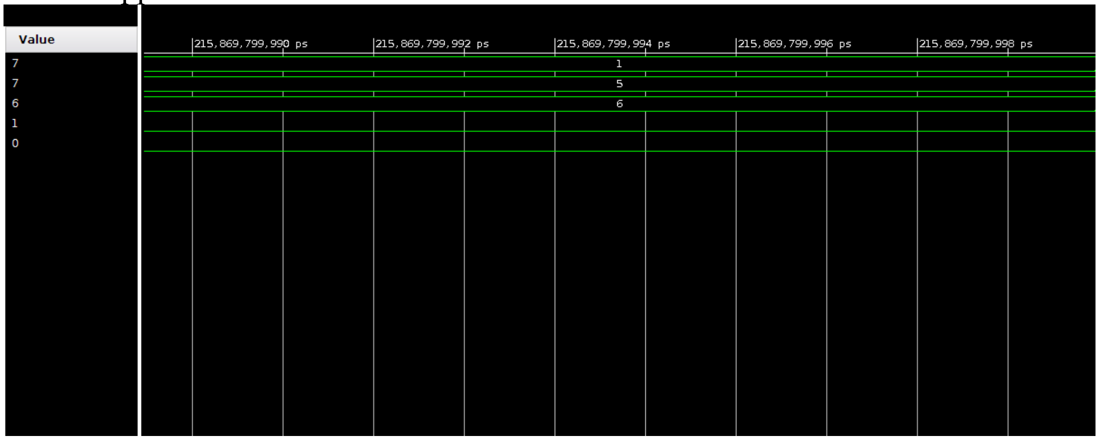
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ripple_adder is
Port (A, B: in std_logic_vector(3 downto 0);
C_in: in std_logic;
S: out std_logic_vector(3 downto 0);
C_output: out std_logic);
end ripple_adder;

architecture Behavioral of ripple_adder is

component adder is
Port (A, B, C_in : in std_logic;
S, C_out: out std_logic);
end component;
```

```
signal A1,A2,A3: std_logic;

begin
Adder0: adder port map(A => A(0),B => B(0),C_in =>
C_in,C_out => A1, S => S(0));
Adder1: adder port map(A => A(1),B => B(1),C_in =>
A1,C_out => A2,S => S(1));
Adder2: adder port map(A => A(2),B => B(2),C_in =>
A2,C_out => A3,S => S(2));
Adder3: adder port map(A => A(3),B => B(3),C_in =>
A3,C_out => C_output,S => S(3));
end Behavioral;
```

- Test Bench for Ripple Adder

```
library
IEEE;
          use IEEE.STD_LOGIC_1164.ALL;
          use IEEE.std_logic_1164.all;
          use IEEE.numeric_std.all;



          entity ripple_addertb is
          end ripple_addertb;



          architecture Behavioral of ripple_addertb is
              signal C_intb      : std_logic;
              signal A_tb, B_tb   : std_logic_vector(3 downto 0);
              signal C_outb     : std_logic;
              signal S_tb         : std_logic_vector(3 downto 0);

              component ripple_adder
              Port (A, B: in std_logic_vector(3 downto 0);
              C_in: in std_logic;
              S: out std_logic_vector(3 downto 0);
              C_output: out std_logic);

              end component;



              begin
                  rippleadder: ripple_adder port map (A=>A_tb, B=>B_tb,
          S=>S_tb, C_in=>C_intb, C_output=>C_outb);
                  process begin
                      A_tb <= "0010";
                      C_intb <= '1';
```

```
        wait for 20 ms;
        B_tb <= "1011";
        wait for 20 ms;
        B_tb <= "0011";
        C_intb <= '0';
        wait for 20 ms;
        A_tb <= "1001";
        C_intb <= '1';
        wait for 20 ms;
        B_tb<="1100";
    end process;
end Behavioral;
```

- Simulation for ripple adder tb
  - 

- 16-Bit Alu
  - Theory of Operation
    - This circuit is implementing a button that will be enabled by the behavior of a clock. This button will be used from the last lab. We will be designed a 16 function ALU following the functions listed in the lab manual. We will also test the ALU by implementing the debouncer from the last lab.
    - If designed correctly, the ALU tester should express the correct switch and led controls in the zybo. Pressing button 3 should clear all 3 signals to 0 Etc.
  - Design
    - VHDL Code for my_alu

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

          entity my_alu is
          Port( clk :in std_logic;
          A,B,opcode :in  std_logic_vector( 3 downto 0 );
          output :out  std_logic_vector( 3 downto 0));
```

```
end my_alu;


architecture Behavioral of my_alu is
begin
alu : process(clk)
begin
    if rising_edge(clk) then
        case Opcode is
when "0000" => output <= std_logic_vector(unsigned(A) +
unsigned(B));
when "0001" => output <= std_logic_vector(unsigned(A) -
unsigned(B));
when "0010" => output <= std_logic_vector(unsigned(A) + 1);
when "0011" => output <= std_logic_vector(unsigned(A) - 1);
when "0100" => output <= std_logic_vector(0 - unsigned(A));

when "0101" => if (A > B) then
    output <= "0001";
    else
    output <= "0000";
    end if;


when "0110" => output <= A(2 downto 0) & '0';
when "0111" => output <= '0' & A(3 downto 1);
when "1000" => output <= A(3) & A(3 downto 1);
when "1001" => output <= not(A);
when "1010" => output <= A and B;
when "1011" => output <= A or B;
when "1100" => output <= A xor B;
when "1101" => output <= A xnor B;
when "1110" => output <= A nand B;
when "1111" => output <= A nor B;
when others => output <= (others => '0');
end case;
end if;
end process alu;

end Behavioral;
```

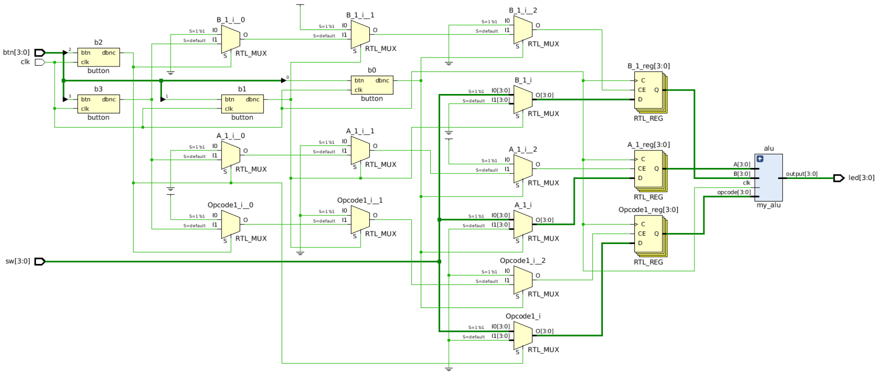
- VHDL Code for alu_tester

```
LibraryIEEE;
use IEEE.STD_LOGIC_1164.ALL;


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

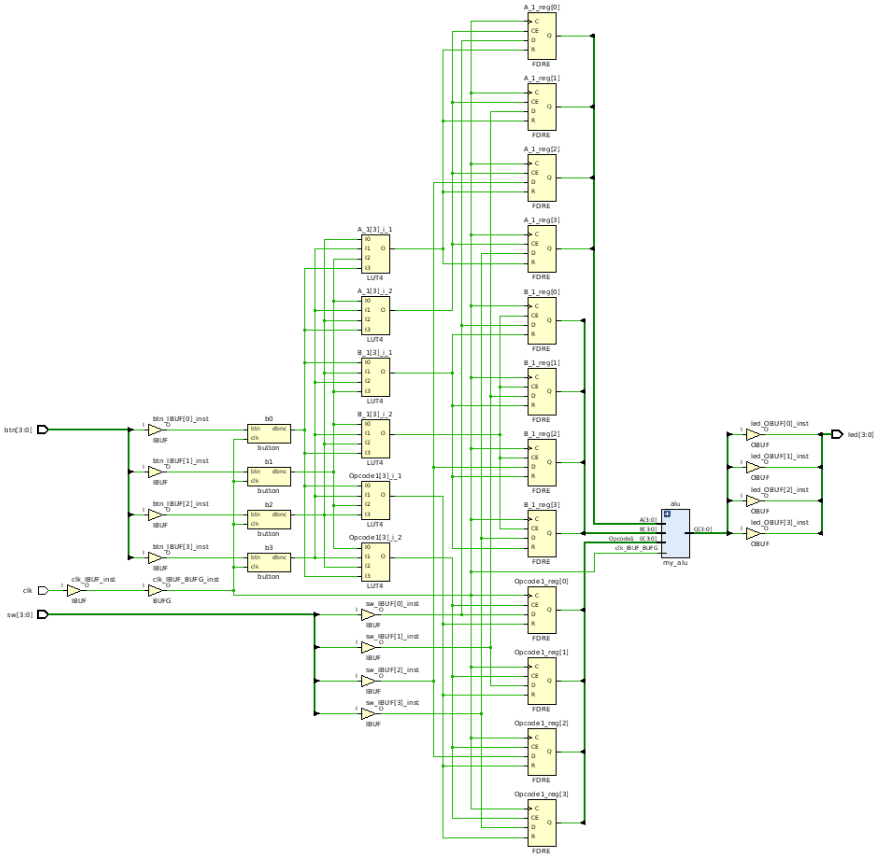
```
architecture Behavioral of alu_tester is
component button
Port( btn,clk: in std_logic;
dbnc: out std_logic);
end component;


component my_alu
Port( clk :in std_logic;
A,B,opcode :in  std_logic_vector( 3 downto 0 );
output :out  std_logic_vector( 3 downto 0));
end component;


signal but1,but2,but3,but4 :  std_logic;
signal A_1,B_1,Opcode1 : std_logic_vector( 3 downto 0);
signal clockout : std_logic_vector(3 downto 0);

begin
b0: button port map(clk => clk,btn => btn(0),dbnc => but1);
b1: button port map(clk => clk,btn => btn(1),dbnc => but2);
b2: button port map(clk => clk,btn => btn(2),dbnc => but3);
b3: button port map(clk => clk, btn => btn(3),dbnc => but4);
alu: my_alu port map(clk => clk,A => A_1,B => B_1,opcode => opcode1,ou
=> led);

process(clk)
begin
if rising_edge(clk) then
        if but1 = '1' then a_1 <= sw;
        elsif but2 = '1' then B_1 <= sw;
        elsif but3 = '1' then Opcode1 <= sw;
        elsif but4 = '1' then
            a_1 <= (others => '0');
            B_1 <= (others => '0');
            Opcode1 <= (others => '0');
    end if;
    end if;
end process;

end Behavioral;
```

- Design

- RTL Schematic for alu_tester
  -



- Synthesis Schematic for alu_tester

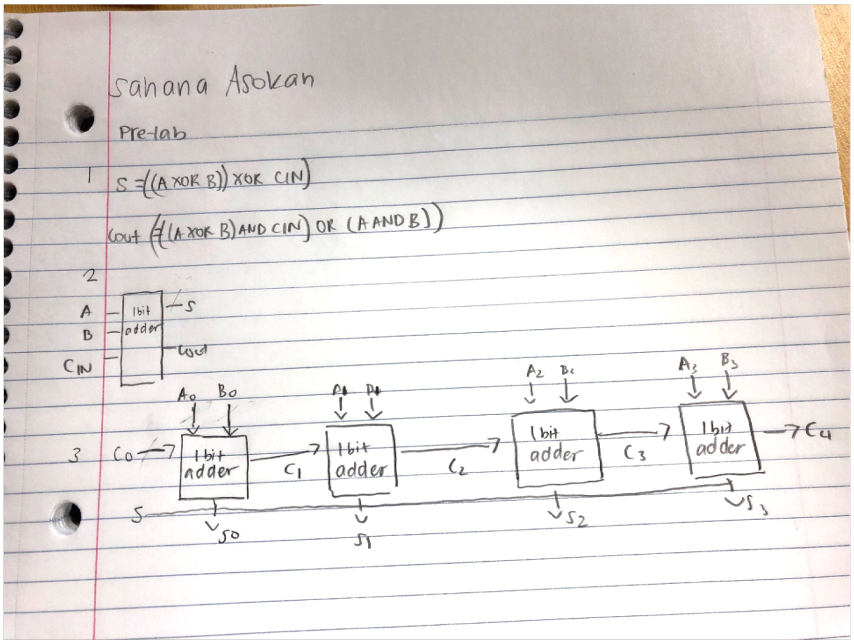


  -

- Discussion
  - Oberservations/Discoveries
    - I learned a lot in this lab. I was having a lot of trouble with my original debouncer code so that is why my overall tester wasn't working.
  - Questions

- I would still like more review in regards to the zybo and implementing different sources. I wish there were more resources in regards to implementing specific logic.

Prelab



Sahana Asokan

Pre-lab

1 S =((A XOR B)) XOR CIN)

Cout =((A XOR B) AND CIN) OR (A AND B))

2
A
B
CIN
1bit adder
S
Cout
3
C0
A0
B0
1 bit adder
C1
A1
B1
1 bit adder
C2
A2
B2
1 bit adder
C3
A3
B3
1 bit adder
C4
S
S0
S1
S2
S3