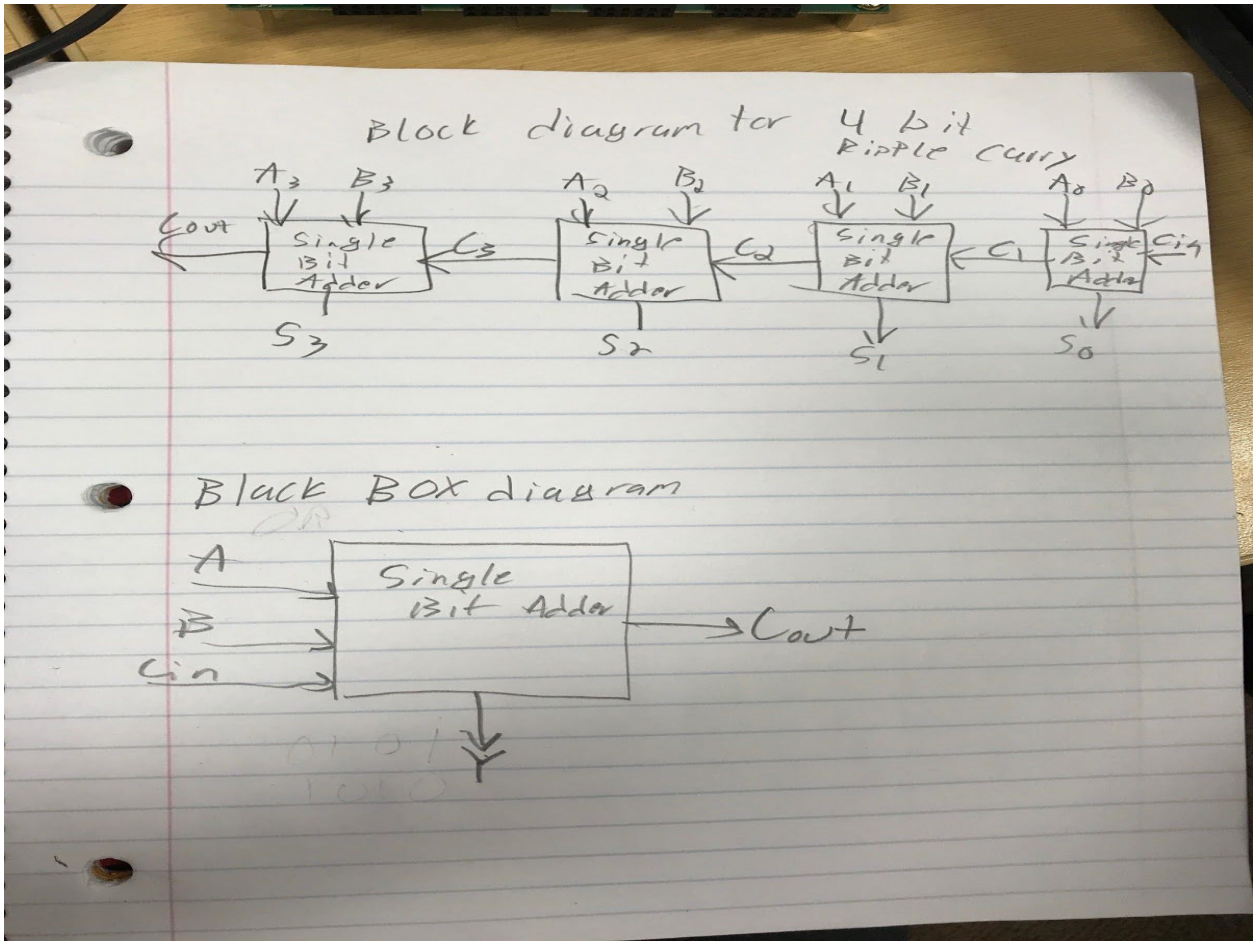
Kevin Honeker
3/7
Lab 2

Pre Lab:

Logic equations for a full adder:Y = A XOR B XOR Cin
Cout = ((A XOR B) AND Cin) OR (A AND B)

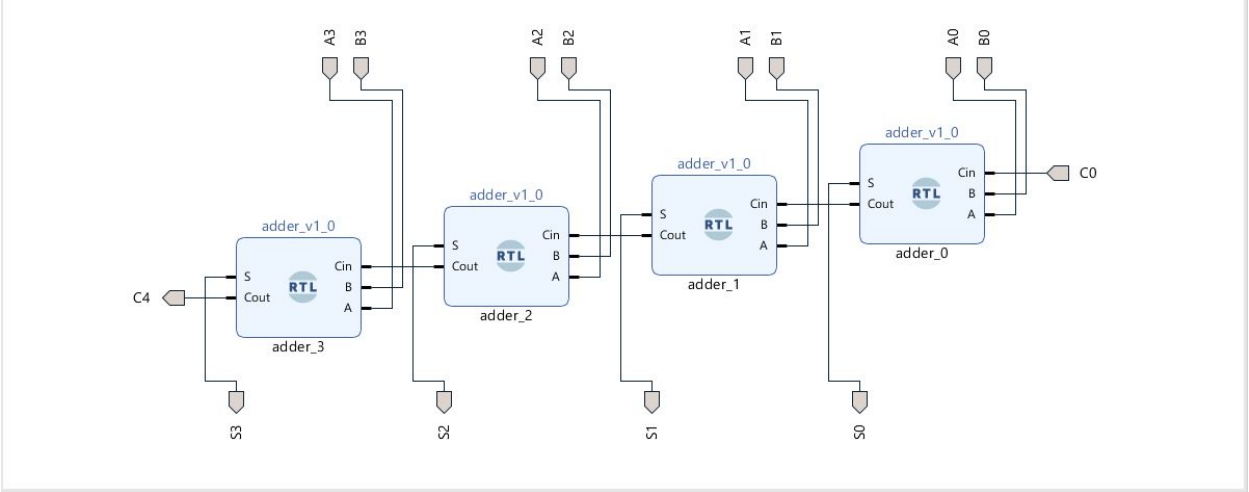


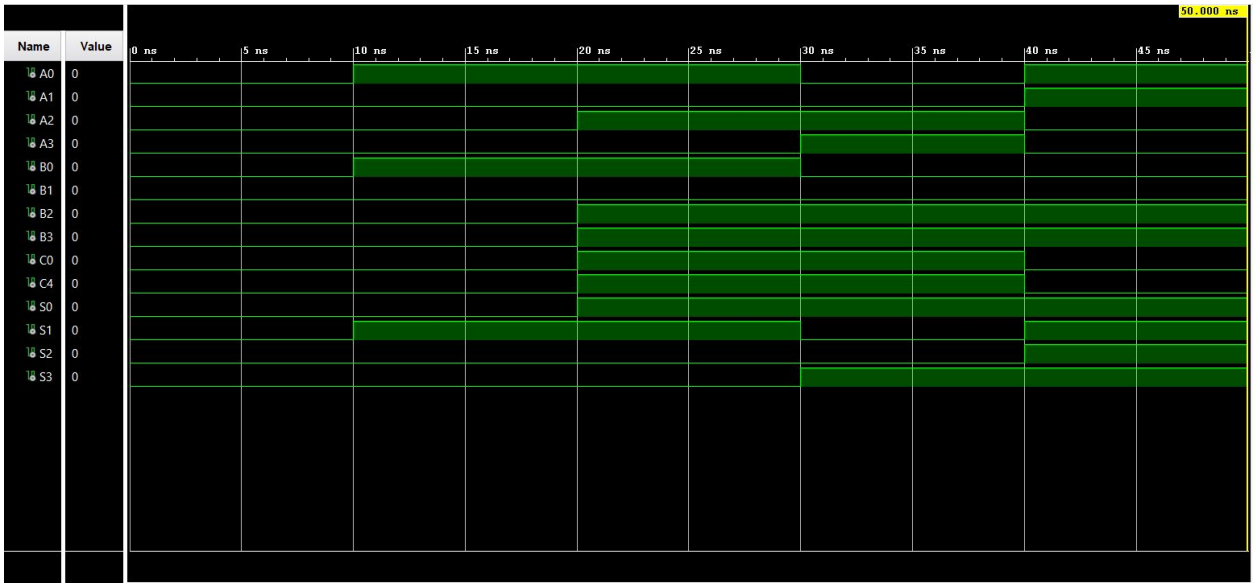
Purpose: the purpose of this lab is to simulate a 4 bit ripple carry adder and a 16 function ALU.

Part 1
Theory of operation: The circuit should be able to perform 4 bit arithmetic This is the truth table for each single bit adder is:

A	B	Cin	S	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

Block Diagram that simulated 4 bit ripple carry adder





VHDL for Single Bit adder:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity adder is
    port (A, B, Cin : in  std_logic;
          Cout, S   : out std_logic);
end adder;

architecture add of adder is
begin

S <= (A XOR B) XOR Cin;
Cout <= ((A XOR B) AND Cin) OR (A AND B);
end add;
```

VHDL for 4 bit ripple carry adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VCOMPONENTS.ALL;
entity ripple_adder is
```

```
  port (
    A0 : in STD_LOGIC;
    A1 : in STD_LOGIC;
    A2 : in STD_LOGIC;
    A3 : in STD_LOGIC;
    B0 : in STD_LOGIC;
    B1 : in STD_LOGIC;
    B2 : in STD_LOGIC;
    B3 : in STD_LOGIC;
    C0 : in STD_LOGIC;
    C4 : out STD_LOGIC;
    S0 : out STD_LOGIC;
    S1 : out STD_LOGIC;
    S2 : out STD_LOGIC;
    S3 : out STD_LOGIC
  );
  attribute CORE_GENERATION_INFO : string;
  attribute CORE_GENERATION_INFO of ripple_adder : entity is
"ripple_adder,IP_Integrator,{x_ipVendor=xilinx.com,x_ipLibrary=BlockDiagram,x_ipName=ripple
_adder,x_ipVersion=1.00.a,x_ipLanguage=VHDL,numBlks=4,numReposBlks=4,numNonXlnxBl
ks=0,numHierBlks=0,maxHierDepth=0,numSysgenBlks=0,numHlsBlks=0,numHdlrefBlks=4,num
PkgbdBlks=0,bdsource=USER,synth_mode=OOC_per_IP}";
  attribute HW_HANDOFF : string;
  attribute HW_HANDOFF of ripple_adder : entity is "ripple_adder.hwdef";
end ripple_adder;

architecture STRUCTURE of ripple_adder is
  component ripple_adder_adder_0_0 is
  port (
    A : in STD_LOGIC;
    B : in STD_LOGIC;
    Cin : in STD_LOGIC;
    Cout : out STD_LOGIC;
    S : out STD_LOGIC
  );
  end component ripple_adder_adder_0_0;
  component ripple_adder_adder_1_0 is
  port (
    A : in STD_LOGIC;
    B : in STD_LOGIC;
    Cin : in STD_LOGIC;
    Cout : out STD_LOGIC;
    S : out STD_LOGIC
```

```
  );
  end component ripple_adder_adder_1_0;
  component ripple_adder_adder_2_0 is
  port (
    A : in STD_LOGIC;
    B : in STD_LOGIC;
    Cin : in STD_LOGIC;
    Cout : out STD_LOGIC;
    S : out STD_LOGIC
  );
  end component ripple_adder_adder_2_0;
  component ripple_adder_adder_3_0 is
  port (
    A : in STD_LOGIC;
    B : in STD_LOGIC;
    Cin : in STD_LOGIC;
    Cout : out STD_LOGIC;
    S : out STD_LOGIC
  );
  end component ripple_adder_adder_3_0;
  signal A0_1 : STD_LOGIC;
  signal A1_1 : STD_LOGIC;
  signal A2_1 : STD_LOGIC;
  signal A3_1 : STD_LOGIC;
  signal B0_1 : STD_LOGIC;
  signal B1_1 : STD_LOGIC;
  signal B2_1 : STD_LOGIC;
  signal B3_1 : STD_LOGIC;
  signal C0_1 : STD_LOGIC;
  signal adder_0_Cout : STD_LOGIC;
  signal adder_0_S : STD_LOGIC;
  signal adder_1_Cout : STD_LOGIC;
  signal adder_1_S : STD_LOGIC;
  signal adder_2_Cout : STD_LOGIC;
  signal adder_2_S : STD_LOGIC;
  signal adder_3_Cout : STD_LOGIC;
  signal adder_3_S : STD_LOGIC;
begin
  A0_1 <= A0;
  A1_1 <= A1;
  A2_1 <= A2;
  A3_1 <= A3;
  B0_1 <= B0;
```

```
  B1_1 <= B1;
  B2_1 <= B2;
  B3_1 <= B3;
  C0_1 <= C0;
  C4 <= adder_3_Cout;
  S0 <= adder_0_S;
  S1 <= adder_1_S;
  S2 <= adder_2_S;
  S3 <= adder_3_S;
adder_0: component ripple_adder_adder_0_0
    port map (
     A => A0_1,
     B => B0_1,
     Cin => C0_1,
     Cout => adder_0_Cout,
     S => adder_0_S
   );
adder_1: component ripple_adder_adder_1_0
    port map (
     A => A1_1,
     B => B1_1,
     Cin => adder_0_Cout,
     Cout => adder_1_Cout,
     S => adder_1_S
   );
adder_2: component ripple_adder_adder_2_0
    port map (
     A => A2_1,
     B => B2_1,
     Cin => adder_1_Cout,
     Cout => adder_2_Cout,
     S => adder_2_S
   );
adder_3: component ripple_adder_adder_3_0
    port map (
     A => A3_1,
     B => B3_1,
     Cin => adder_2_Cout,
     Cout => adder_3_Cout,
     S => adder_3_S
   );
end STRUCTURE;
```

Test Bench

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

entity test_ripple_adder is
end test_ripple_adder;


architecture TB of test_ripple_adder is

component ripple_adder is
port (
  A0 : in STD_LOGIC;
  A1 : in STD_LOGIC;
  A2 : in STD_LOGIC;
  A3 : in STD_LOGIC;
  B0 : in STD_LOGIC;
  B1 : in STD_LOGIC;
  B2 : in STD_LOGIC;
  B3 : in STD_LOGIC;
  C0 : in STD_LOGIC;
  C4 : out STD_LOGIC;
  S0 : out STD_LOGIC;
  S1 : out STD_LOGIC;
  S2 : out STD_LOGIC;
  S3 : out STD_LOGIC
);
end component ripple_adder;

signal A0 : STD_LOGIC;
signal A1 : STD_LOGIC;
signal A2 : STD_LOGIC;
signal A3 : STD_LOGIC;
signal B0 : STD_LOGIC;
signal B1 : STD_LOGIC;
signal B2 : STD_LOGIC;
signal B3 : STD_LOGIC;
signal C0 : STD_LOGIC;
signal C4 : STD_LOGIC;
signal S0 : STD_LOGIC;
signal S1 : STD_LOGIC;
signal S2 : STD_LOGIC;
signal S3 : STD_LOGIC;
```

```
begin

test : process
begin
--wait for 10 ns;
    A0<='0';
    A1<='0';
    A2<='0';
    A3<='0';
    B0<='0';
    B1<='0';
    B2<='0';
    B3<='0';
    C0<='0';
    wait for 10 ns;
    A0<='1';        --A = 0001
    B0<='1';        --B = 0001
    wait for 10 ns;
    A2<='1';        --A = 0101
    B3<='1';        --B = 1101
    B2<='1';
    C0<='1';
    wait for 10 ns;
    A0<='0';
    B0<='0';        --B = 1100
    A3<='1';        --A = 1100
    wait for 10 ns;
    A3<='0';
    A2<='0';
    A1<='1';
    A0<='1';    --  A = 0011
    C0<='0';
    Wait for 10 ns;
    C0<='1';




end process;
DUT: component ripple_adder port map (
  A0 => A0,
  A1 => A1,
  A2 => A2,
```

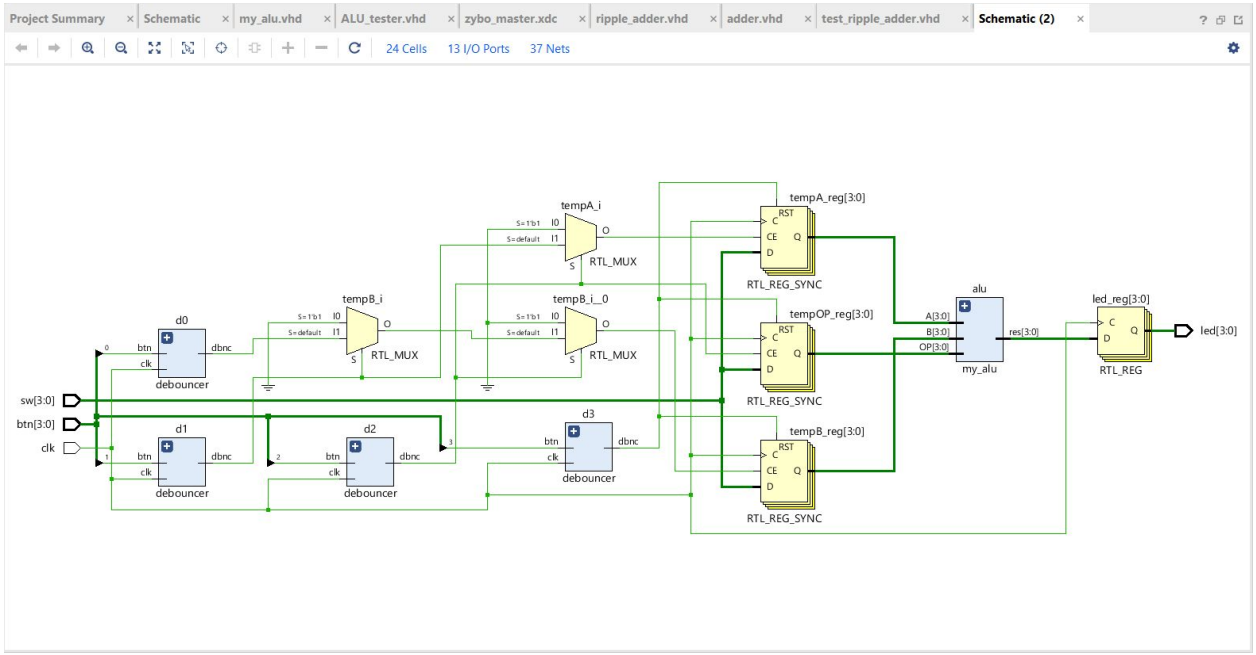
```
  A3 => A3,
  B0 => B0,
  B1 => B1,
  B2 => B2,
  B3 => B3,
  C0 => C0,
  C4 => C4,
  S0 => S0,
  S1 => S1,
  S2 => S2,
  S3 => S3
);


end TB;
```

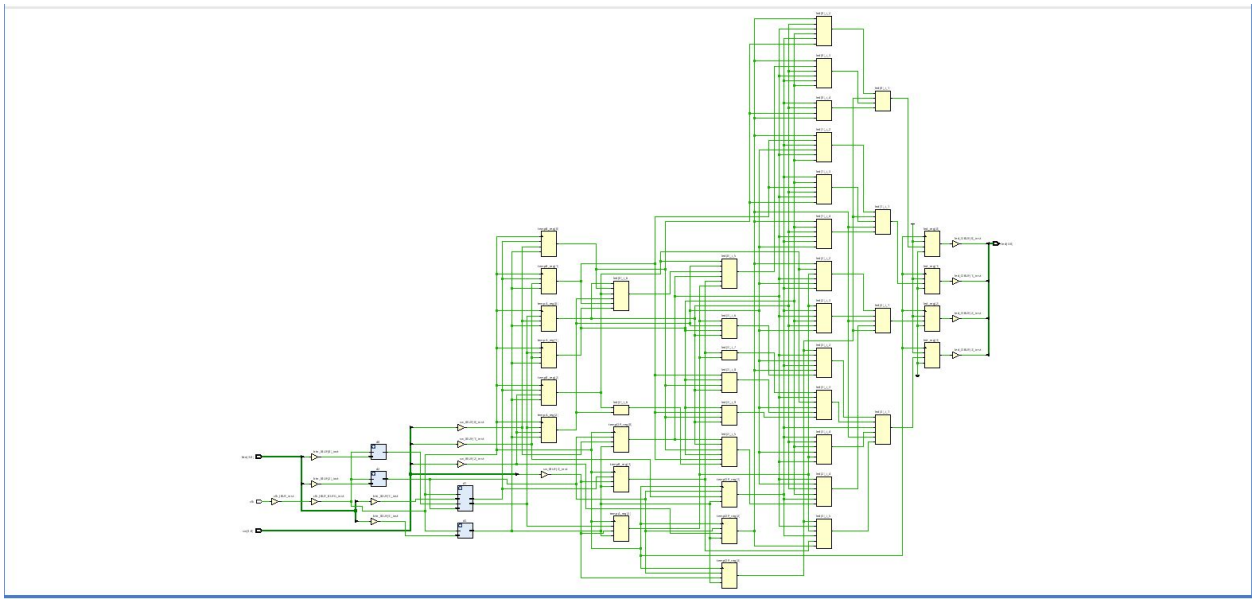
Part 2

Theory of Operation: The circuit should be have as the 16 function 4 bit ALU whose functions are specified in the lab instructions.

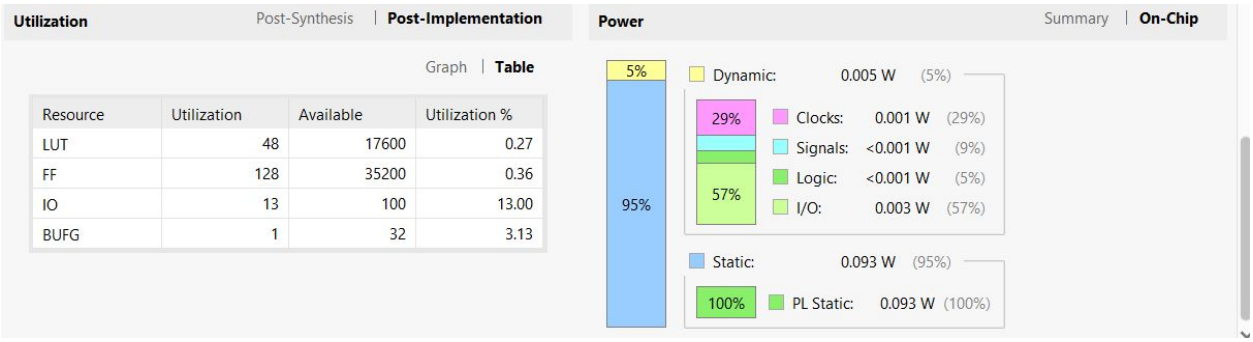
Elaboration Schematic:



Synthesis Schematic



Power Graphs and utilization table



Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	48	17600	0.27
FF	128	35200	0.36
IO	13	100	13.00
BUFG	1	32	3.13

VHDL Code for my_ALU

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

use IEEE.std_logic_unsigned.all;
use IEEE.NUMERIC_STD.ALL;


entity my_alu is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           B : in STD_LOGIC_VECTOR (3 downto 0);
           OP : in STD_LOGIC_VECTOR (3 downto 0);
```

```
        res : out STD_LOGIC_VECTOR (3 downto 0));
end my_alu;

architecture Behavioral of my_alu is

begin
process (OP)
begin
   case OP is
   when "0000" => res <= A+B;
   when "0001" => res <= A-B;
   when "0010" => res <= A+1;
   when "0011" => res <= A-1;
   when "0100" => res <= 0-A;
   when "0101" =>
   if unsigned(A)>unsigned(B) then
   res <= "0001";
   else
   res <= "0000";
   end if;

   when "0110" => res<= A(2 downto 0)&'0';
   when "0111" => res<= '0'&A(3 downto 1);
   when "1000" => res<= '1'&A(3 downto 1);
   when "1001" => res<= not A;
   when "1010" => res<= A AND B;
   when "1011" => res<= A OR B;
   when "1100" => res<= A XOR B;
   when "1101" => res<= A XNOR B;
   when "1110" => res<= A NAND B;
   when "1111" => res<= A NOR B;
   when others => res<= "0000";

   end case;

 end process;
end Behavioral;
```

VHDL Code for ALU_tester

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.NUMERIC_STD.ALL;



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

architecture alu_test of ALU_tester is
signal dbnc0, dbnc1, dbnc2, dbnc3 : std_logic;
signal result, tempA, tempB, tempOP : std_logic_vector(3 downto 0);

component debouncer
     port ( btn, clk : in std_logic;
          dbnc     : out std_logic);
    end component;
component my_alu
    port( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          OP : in STD_LOGIC_VECTOR (3 downto 0);
          res : out STD_LOGIC_VECTOR (3 downto 0));
end component;

begin

 d0 : debouncer
    port map( btn => btn(0),
              clk => clk,
              dbnc => dbnc0);
 d1 : debouncer
    port map (btn => btn(1),
             clk => clk,
             dbnc => dbnc1);
 d2 : debouncer
    port map ( btn => btn(2),
               clk=> clk,
               dbnc => dbnc2);
  d3 : debouncer
    port map (btn => btn(3),
```

```
            clk => clk,
            dbnc => dbnc3);

   alu : my_alu
       port map ( A => tempA,
                  B => tempB,
                  OP => tempOP,
                  res => result);

process (clk)
    begin
       if clk='1' and clk'event then
          if dbnc3 ='1' then
             tempA<="0000";
             tempB<="0000";
             tempOP<="0000";
           elsif dbnc2 = '1' then
             tempOP<=sw;
           elsif dbnc1 = '1' then
             tempA<=sw;
           elsif dbnc0 = '1' then
             tempB<=sw;
           end if;
         led<=result;
        end if;
      end process;

end alu_test;

Zybo master XDC 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
```

The segment of code is the uncomment parts of the xdc file that allowed my designs to work on the Zybo board.

Discussion:
In this lab I learned about the functions and operations that are defined in the numeric_std library. I also learned that Vivado can generate test bench files and design sources from block diagrams