Efe Tarhan

22002840

EE102-02

3.11.2021

LAB 4: ARITHMETIC LOGIC UNIT

A) PURPOSE

The purpose of this experiment is to understand and implement an ALU module by using VHDL. Learning VHDL functions like “entity, port, port map, process” is also a goal of this lab. Implementing component instantation by using “port map” function is understood and being used a lot in this lab.

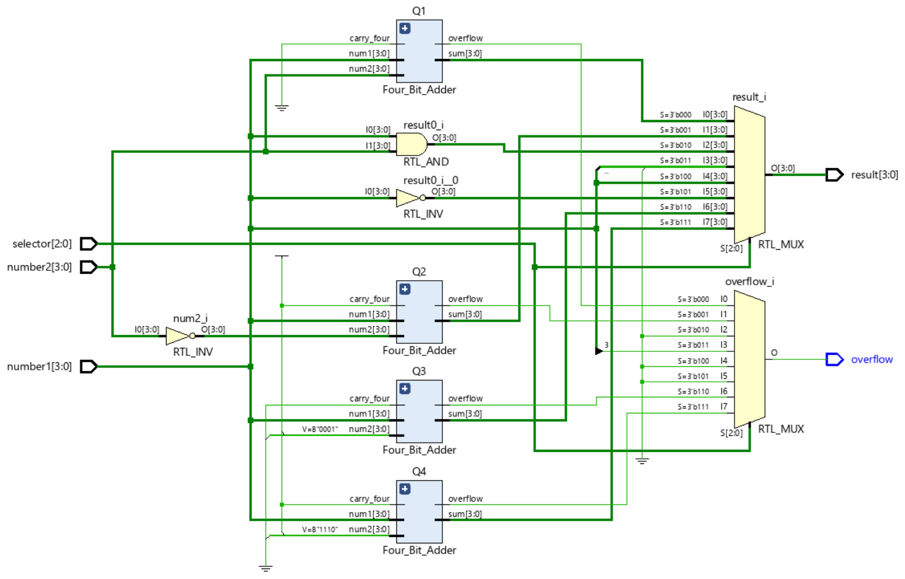
B) DESIGN METHODOLOGY

Main objective of this lab is to design an ALU unit by using two 4 bit numbers. 8 different arithmetic operations will be implemented with the unit and the numbers. AlU will be a combinational logic box which will contain addition, subtraction, bitwise and operation, left bit shifter, and lastly bit rotation. So, if the ALU is considered as a component 3 input, 4 bus first number, 4 bus second number and a 3 bus selector input will be connected to the input side of the component. At the output side, there will be the result of the operation that we choose and also an indicator which shows if there is an overflow caused from substraction error or bit insufficiency.

Selector Code	**Arithmetic Operation**
“000”	**Addition**
“001”	**Subtraction**
“010”	**Bitwise AND**
“011”	**Shift Left**
“100”	**Bit Rotation**
“101”	**One’s Complement**
“110”	**Increment**
“111”	**Decrement**

(Table 1: Multiplexer inputs and their operation counterparts)

First an one bit full adder will be written. In another design file this component will be called with the component and port map functions which will be useful when making a four bit full adder from four one bit full adders. After completing the second design file which is the four bit adder. The last file will be created which will be the ALU. In ALU the operations that require four bit adder (addition, substraction, increment and decrement) will be completed outside the 8 to 1 multiplexer which will choose the operation that the user desire to implement. After doing port map for four of the operations the multiplexer will be prepared.

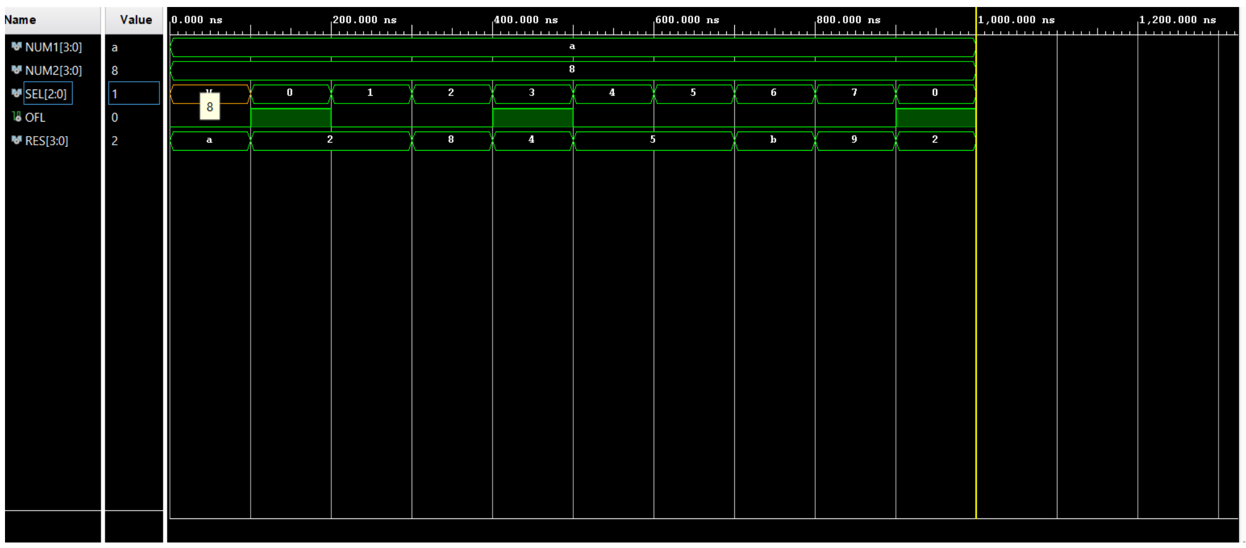


(Image 1: RTL Schmatic of ALU)

After completing the coding part and creating the RTL Schematic, it became more sensible that there also must be a multiplexer for the results that come from the overflows. Hence the last shape of the ALU component that was designed can be seen from the Image 1.

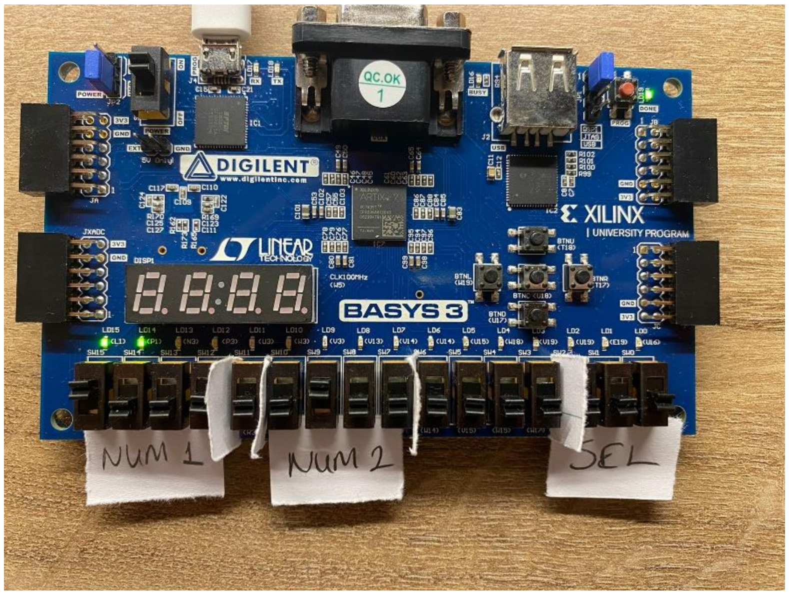
C) RESULTS

The test bench code in the appendix part has written and the simulation was done. The image below is the result of the test bench code of the ALU with first number equal to “1010” and second number is equal to “1000”. The type of the operation has been changed by sending different multiplexer select combinations to see the results.

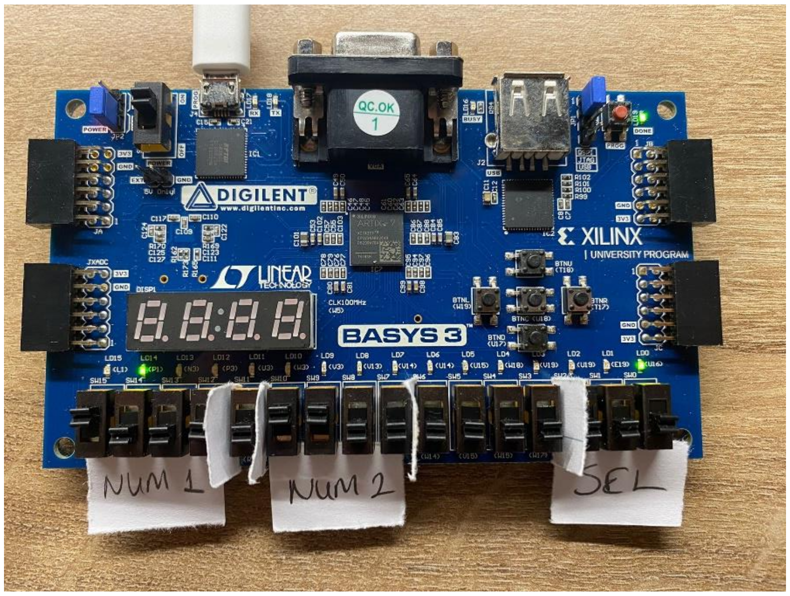


(Image 2: Testbench Simulation of OFL and RES when NUM1 = “1010” and NUM2 = “1000”)

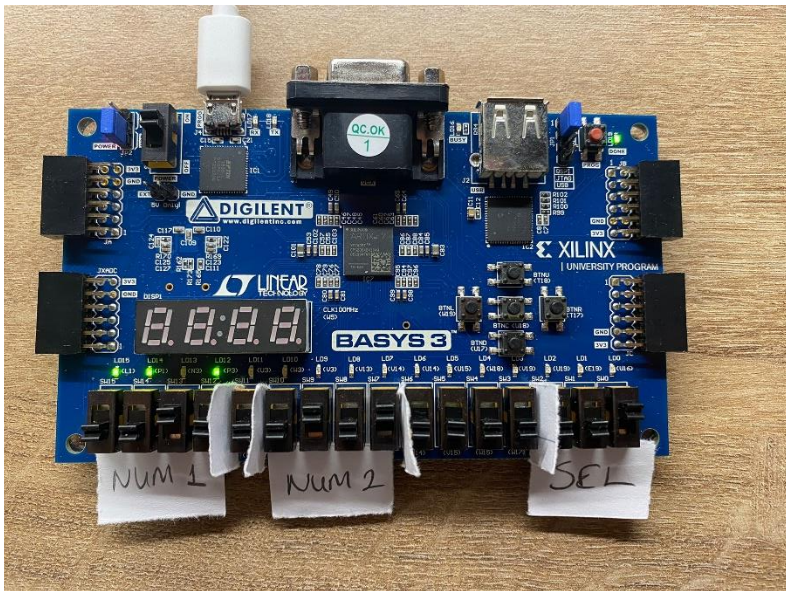
After the simulation process has done without error bitstream of the test code has generated for installation of it on BASYS3. The device is programmed and several input has experimented on the device. Results of the experiments are displayed with the images below



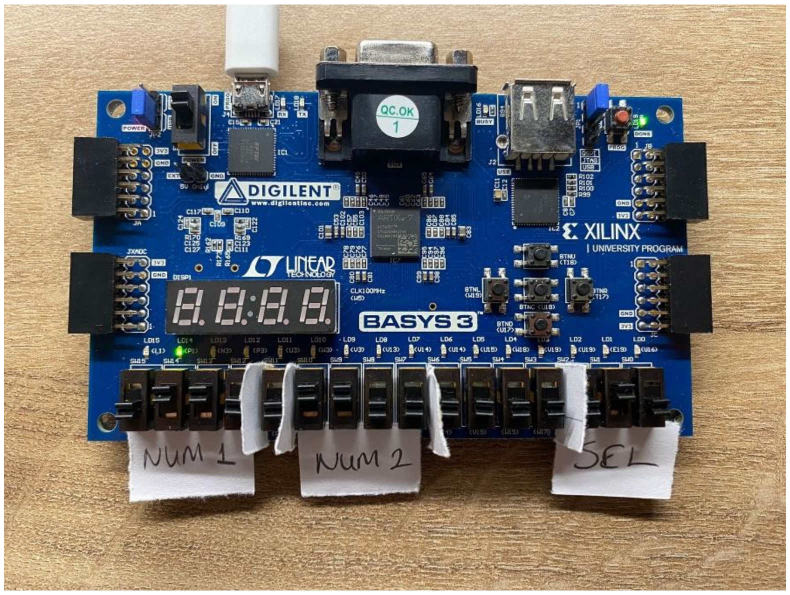
(Image 3: NUM1 = “1000”,NUM2 = “0100”, SEL = “000” then RES = “1100”, OL = ‘0’ : Addition)



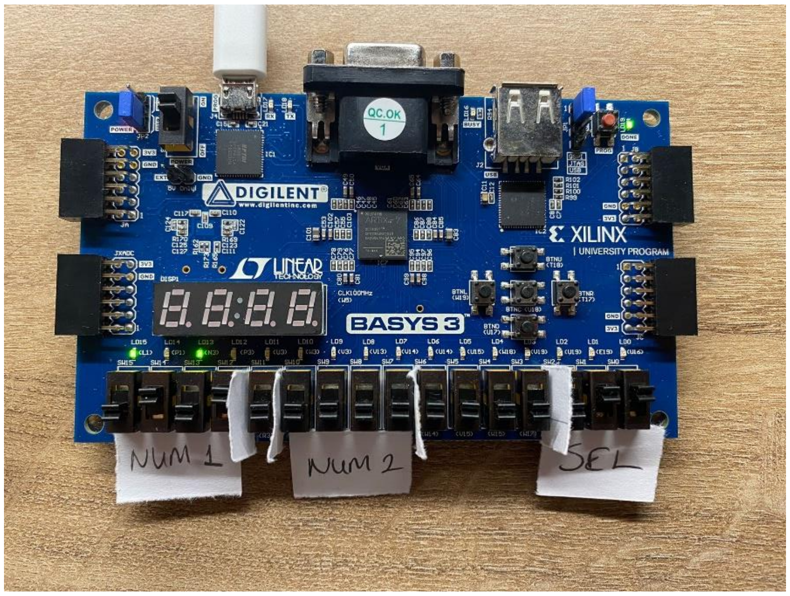
(Image 4: NUM1 = “1000”,NUM2 = “1100”, SEL = “000” then RES = “0100”, OL = ‘1’: Addition)



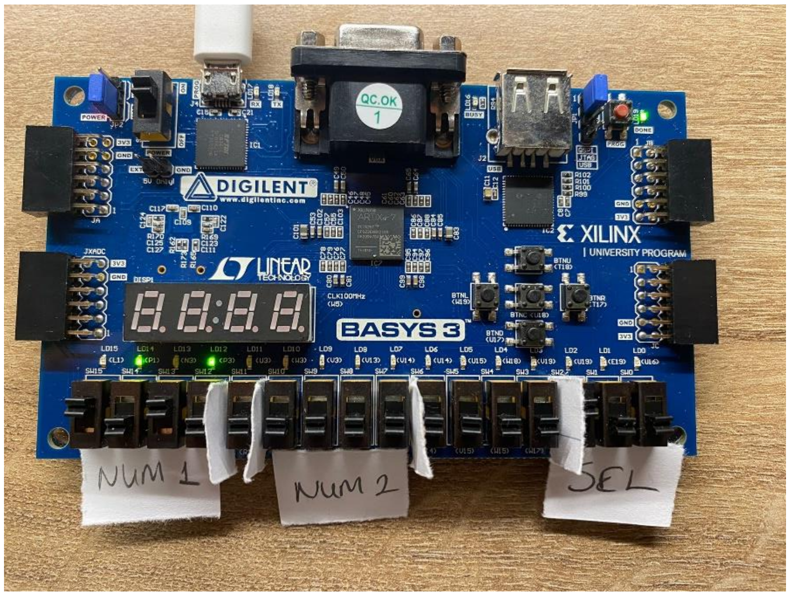
(Image 5: NUM1 = “0010” , NUM2 = “0101”, SEL = “001” then RES = “1101”, OL= ‘0’: Substraction)



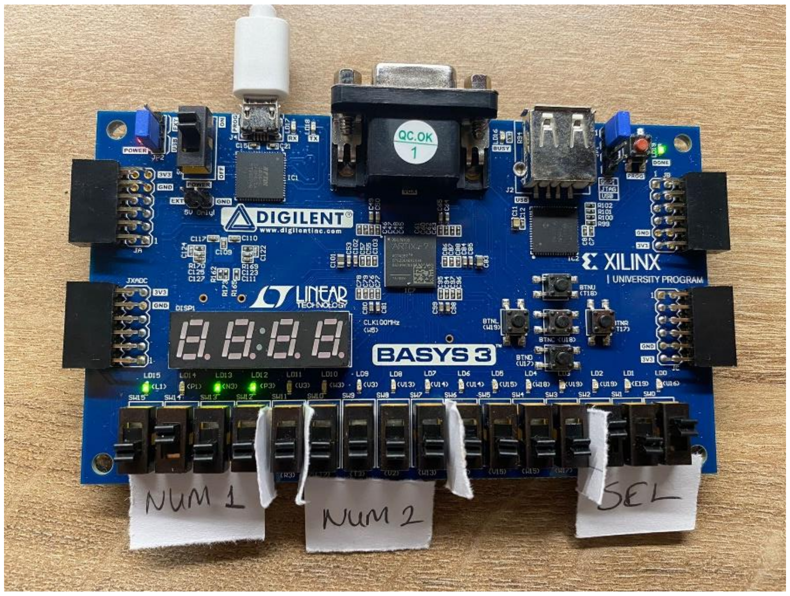
(Image 6: NUM1 = “0110” , NUM2 = “1100”, SEL = “010” then RES = “0100”, OL= ‘0’: Bitwise AND)



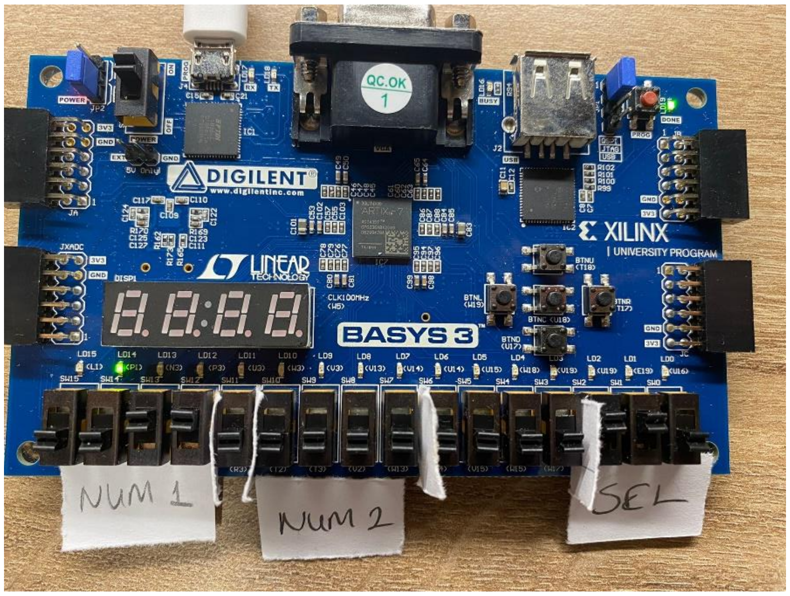
(Image 7: NUM1 = “0101” , NUM2 = “0000”, SEL = “011” then RES = “1010”, OL= ‘0’ : Shift to Left)



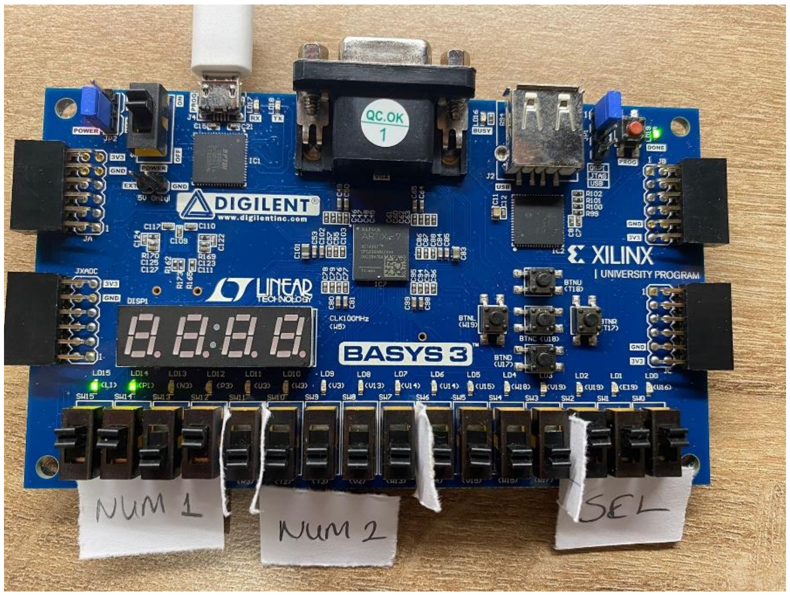
(Image 8: NUM1 = “1010” , NUM2 = “0000”, SEL = “100” then RES = “0101”, OL= ‘0’ : Rotation)



(Image 9: NUM1 = “0100” , NUM2 = “0000”, SEL = “101” then RES = “1011”, OL= ‘0’ : One’s Complement)



(Image 10: NUM1 = “0011” , NUM2 = “0000”, SEL = “110” then RES = “0100”, OL= ‘0’ : Increment by One)



(Image 11: NUM1 = “1101” , NUM2 = “0000”, SEL = “111” then RES = “1100”, OL= ‘0’ : Sıgned Decrement by One)

D) CONCLUSION

Implementation of component instantation process have well experienced in this lab. Several errors and problems occurred during coding but by following the error information debugging has done. RTL Schemtic of the desired ALU has gave more chance to visualize theoric expressions that we’ve been using in the lectures. This lab was crucial before starting to work on the term project because it was important for handiness of the VHDL.

E)APPENDIX

One Bit Adder Code:

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

entity One_Bit_Adder is
    Port ( carry0: in STD_LOGIC;
            bit1 : in STD_LOGIC;
            bit2 : in STD_LOGIC;
            sum : out STD_LOGIC;
            carry1 : out STD_LOGIC);
end One_Bit_Adder;
architecture Behavioral of One_Bit_Adder is
begin
sum <= bit1 xor bit2 xor carry0;
carry1 <= ((bit1 xor bit2) and carry0) or (bit1 and bit2);
end Behavioral;
```

Four Bit Adder Code:

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

library One_bit_adder;
use One_bit_adder.all;

entity Four_Bit_Adder is
    Port ( carry_four : in STD_LOGIC;
                num1 : in STD_LOGIC_VECTOR (3 downto 0);
            num2 : in STD_LOGIC_VECTOR (3 downto 0);
            sum : out STD_LOGIC_VECTOR (3 downto 0);
            overflow: out STD_LOGIC;
            carry_o : inout STD_LOGIC);
end Four_Bit_Adder;

architecture Behavioral of Four_Bit_Adder is
signal carry : STD_LOGIC_VECTOR (3 downto 1);
COMPONENT One_Bit_Adder port(carry0,bit1,bit2 : in STD_LOGIC; sum, carry1 : out STD_LOGIC);
```

```
end COMPONENT;


begin



S0: One_Bit_Adder port map(carry0 => carry_four,bit1 => num1(0),bit2 => num2(0),sum =>
sum(0),carry1 => carry(1));


S1: One_Bit_Adder port map(carry0 => carry(1),bit1 => num1(1),bit2 => num2(1),sum => sum(1),carry1
=> carry(2));


S2: One_Bit_Adder port map(carry0 => carry(2),bit1 => num1(2),bit2 => num2(2),sum => sum(2),carry1
=> carry(3));


S4: One_Bit_Adder port map(carry0 => carry(3),bit1 => num1(3),bit2 => num2(3),sum => sum(3),carry1
=> carry_o);


overflow <= carry_o xor carry(3);
end Behavioral;
```

ALU Code:

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


library Four_Bit_Adder;
use Four_Bit_Adder.all;



entity ALU is
    Port ( number1 : in STD_LOGIC_VECTOR (3 downto 0);
           number2 : in STD_LOGIC_VECTOR (3 downto 0);
           selector : in STD_LOGIC_VECTOR (2 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0);
```

```
        overflow : out STD_LOGIC);
end ALU;

architecture Behavioral of ALU is

COMPONENT Four_Bit_Adder port(num1, num2: in STD_LOGIC_VECTOR (3 downto 0); carry_four :
in STD_LOGIC;sum : out STD_LOGIC_VECTOR (3 downto 0);
overflow : out STD_LOGIC ; carry_o : inout STD_LOGIC);
end COMPONENT;

signal result1, result2, result3, result4 : STD_LOGIC_VECTOR (3 downto 0);
signal overflow1, overflow2, overflow3, overflow4 : STD_LOGIC;
signal null1, null2,null3,null4 : STD_LOGIC;
begin

Q1 : Four_Bit_Adder port map(num1 => number1, num2 => number2, carry_four => '0', sum =>
result1,carry_o => overflow1, overflow=> null1 );
Q2: Four_Bit_Adder port map(num1 => number1, num2 => not(number2), carry_four => '1', sum =>
result2,carry_o => null2 ,overflow=> overflow2);
Q3 : Four_Bit_Adder port map(num1 => number1, num2 => "0001", carry_four => '0', sum =>
result3,carry_o => null3, overflow =>overflow3);
Q4 : Four_Bit_Adder port map(num1 => number1, num2 => "1110", carry_four => '1', sum =>
result4,carry_o => null4, overflow => overflow4);

  Allu :process(number1, number2, selector) is
  begin
    case selector is

      when "000" => --Four Bit Adder--
      result <= result1;
      overflow <= overflow1;

      when "001" => --Four Bit Substractor--
```

```
result <= result2;
overflow <= overflow2;

when "010" => --Four bit AND of number1 and number 2--
result <= number1 and number2;
overflow <= '0';

when "011" => --Four bit Logical Shift--

result(3 downto 0) <= number1(2 downto 0) & '0';
overflow <= number1(3);

when "100" => -- Four Bit Rotation--
result(3) <= number1(0);
result(2) <= number1(1);
result(1) <= number1(2);
result(0) <= number1(3);


overflow <= '0';

when "101" => -- Four Bit One's Complement--

result <= not(number1);
overflow <= '0';

when "110" => --Increment by one--
result <= result3;
overflow <= overflow3;

when "111" => --Decrement by one--
```

```
            result <= result4;
            overflow <= overflow4;

            when others =>
            result <= number1;
            overflow <= '0';
        end case;
    end process;
end Behavioral;
```

<u>**Test Bench:**</u>

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

entity ALU_BENCH is
end ALU_BENCH;

architecture Behavioral of ALU_BENCH is
COMPONENT ALU PORT(number1, number2: in STD_LOGIC_VECTOR (3 downto 0);
           selector : in STD_LOGIC_VECTOR (2 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0);
           overflow : out STD_LOGIC);
END COMPONENT;

signal NUM1 : STD_LOGIC_VECTOR (3 downto 0);
signal NUM2 : STD_LOGIC_VECTOR (3 downto 0);
signal SEL : STD_LOGIC_VECTOR (2 downto 0);

signal OFL : STD_LOGIC;
signal RES : STD_LOGIC_VECTOR (3 downto 0);
```

```
begin

UUT: ALU port map(number1 => NUM1, number2 => NUM2, selector => SEL, result => RES,
overflow => OFL );

start: process
begin
NUM1 <= "1010";
NUM2 <= "1000";

wait for 100ns;
SEL <= "000";

wait for 100ns;
SEL <= "001";

wait for 100ns;
SEL <= "010";

wait for 100ns;
SEL <= "011";

wait for 100ns;
SEL <= "100";

wait for 100ns;
SEL <= "101";

wait for 100ns;
SEL <= "110";

wait for 100ns;
```

```
SEL <= "111";
end process;


end Behavioral;
```