Report: Arithmetic Logic Unit (ALU)

Bilkent University Electrical and Electronics Department
EE102-01 Lab 4

Semih Akkoç - 22103132

November 7, 2022

Purpose:

The intent of this lab was to study the essentials of the arithmetic logic unit and design a system that has the ability to achieve addition, subtraction, bitwise operation, and shift operation. Additionally, getting familiar with this unit will grant a fundamental understanding of the combinational circuit which can perform mathematical operations, and the implementation of this circuit on FPGA using VHDL in a modular fashion.

Methodology:

In order to proceed with this lab work, one needed to determine which eight functions would be implemented. The initial two functions were determined as adder and subtractor from the lab manual, and the rest needed to include at least one bitwise and one shift operation. As far as this report is concerned, these functions have been chosen as follows bitwise NOT, AND, OR, XOR, XNOR, and left shift. For this arithmetic logic unit (ALU), 4-bit unsigned binary numbers were chosen, and the implementation of these circuits was integrated with switches and LEDs to observe it on BASYS3.

Design Specifications:

While implementing this unit 4-bit unsigned binary numbers were used and with select inputs, these previously mentioned eight operations can be performed. ALUs inputs and outputs can be seen in the following table.

```
sel  : in  std_logic_vector 3-bit long (select inputs)
x_alu: in std_logic_vector 4-bit long
y_alu: in std_logic_vector 4-bit long
z_alu: out std_logic_vector 4-bit long (result vector)
carry: out std_logic
```

Subsequently, ALU has been designed in a modular fashion. "ALU.vhd" is the module that switches between selections and results in the outputs of the desired function. There are eight modules under the arithmetic logic unit, and they can be listed as follows.

- FourBitAdder.vhd
- FourBitSubtractor.vhd
- BitwiseNot.vhd
- BitwiseAnd.vhd
- BitwiseOr.vhd
- BitwiseXor.vhd
- BitwiseXnor.vhd
- LogicalLeftShift.vhd

Additionally, to create the adder and subtractor modules, full adder and half adder modules have been implemented. Since there may be an overflow of the operations regarding addition and subtraction, the carry bit has been designed as a signifier shows whether there is an overflow or not. The following table shows how select pins choose functions and how functions operate fundamentally.

Select	Operation	Input	Output
000	4-bit Addition	X,Y	X+Y
001	4-bit Subtraction	X,Y	X-Y
010	Bitwise Not	X	X'
011	Bitwise And	X,Y	X AND Y
100	Bitwise Or	X,Y	X OR Y
101	Bitwise Xor	X,Y	X XOR Y
110	Bitwise Xnor	X,Y	X XNOR Y
111	Logical Left Shift	X	Left Shift X

Results:

The suggested ALU design schematics and simulations of it and photos of the implementation on BASYS3 can be found, and the codes regarding the modules can be found in the appendices part.

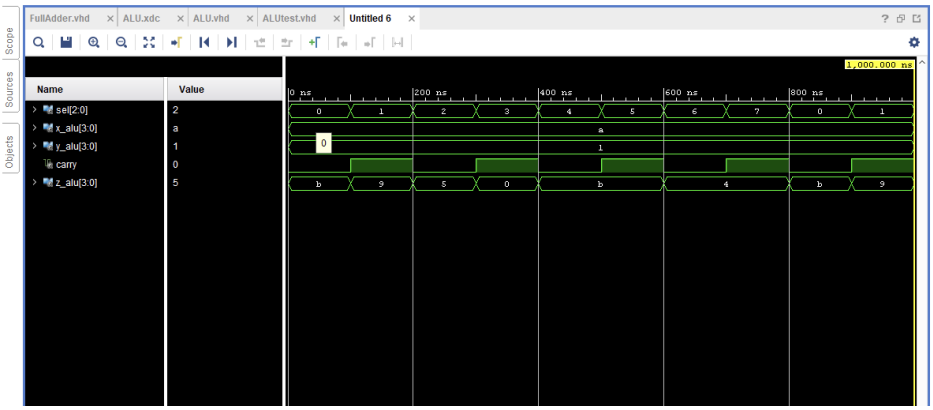


Figure 1: Simulation, testbench of the design.

The following part includes the schematic of the ALU and the other modules which form the arithmetic logic unit. Figure 2. is on the next page.

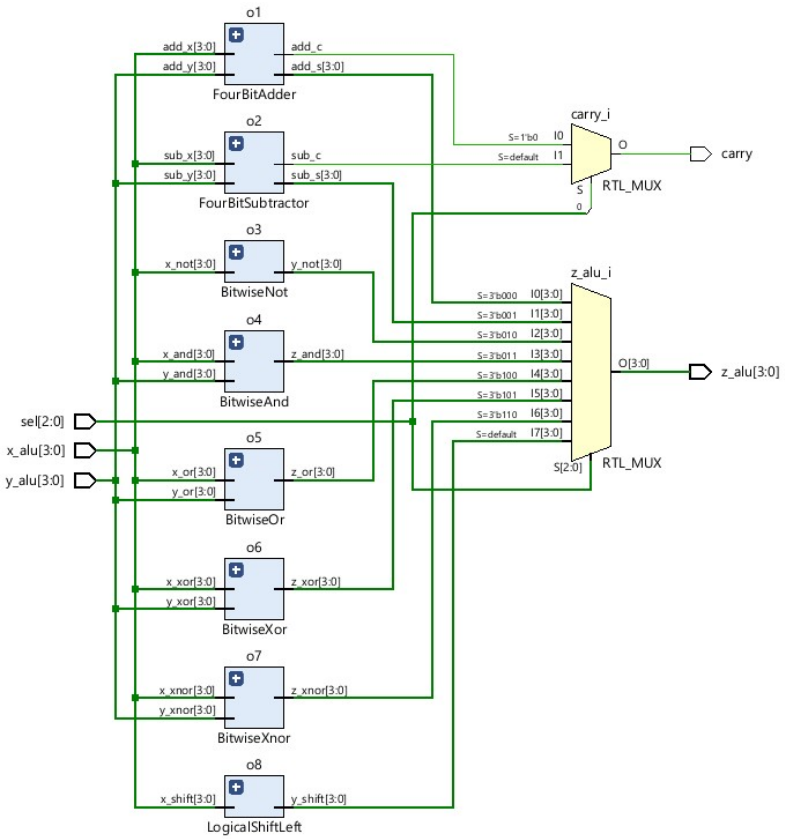
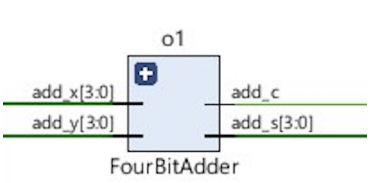
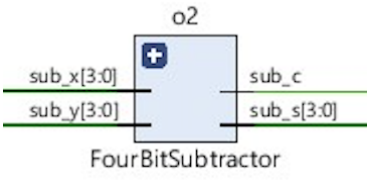


Figure 2: The schematic of the ALU.

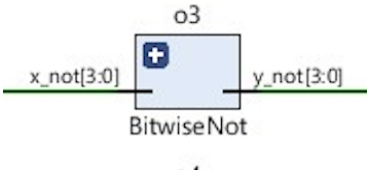
From the figure above, each sub-unit can be observed, and the multiplexer implementation of the selects can be seen as well. Figure 3. is on the next page.



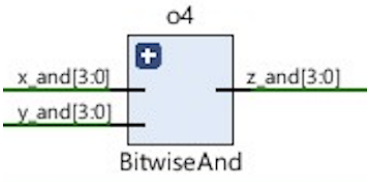
(a) Four-Bit Addition



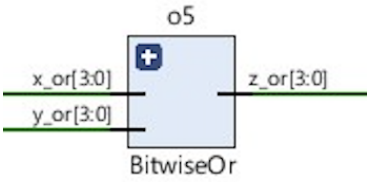
(b) Four-Bit Subtraction



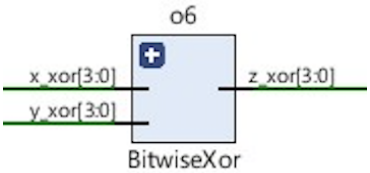
(c) Bitwise Not



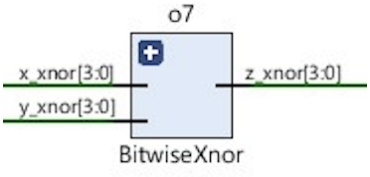
(d) Bitwise And



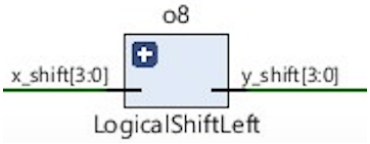
(e) Bitwise Or



(f) Bitwise Xor



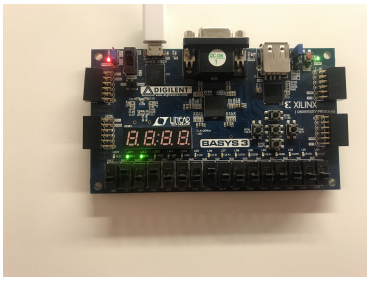
(g) Bitwise Xnor



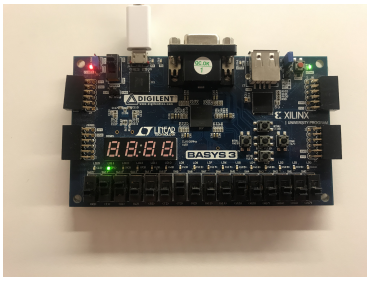
(h) Logical Left Shift

Figure 3: Modules.

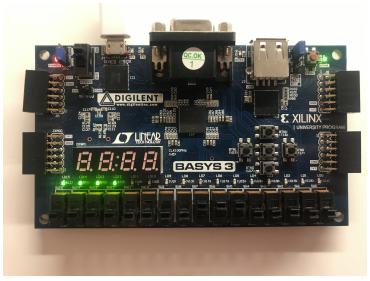
In favor of assigning demonstrating the design on the BASYS3 board following images are chosen to show some of the operations done on the board.



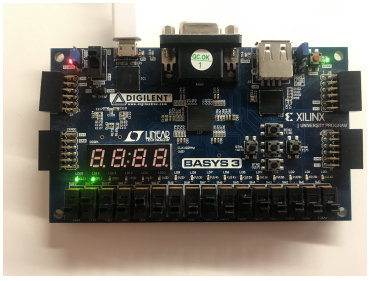
(a) Four-Bit Subtraction



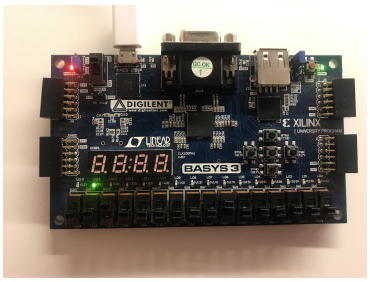
(b) Four-Bit Subtraction



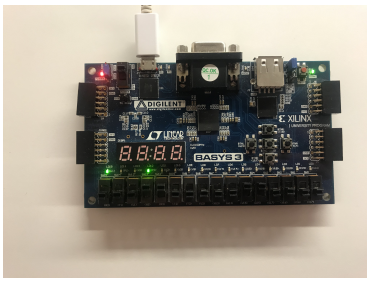
(c) Bitwise Not



(d) Bitwise Or



(e) Logical Left Shift



(f) Bitwise Xor

Figure 4: Some operations on the BASYS3 board.

Conclusion:

The arithmetic logic unit has been implemented using VHDL in a modular fashion. Eight various functions have been used, and with select input, these various operations can be called. All of these operations operated on 4-bit unsigned binary numbers. While implementing these functions 'process' method has been used extensively to implement most the functions. However, especially the modules such as addition and subtraction cannot operate on negative numbers; therefore, further improvements can be made.

Appendices:

Note:

Since all of the unnecessary comments in the codes have been excluded, the remaining pages may seem few. On the contrary, by discarding the redundant comments, the real work done by the student has been brought to light which shows more quality pages than useless ones.

Code 1: (ALU.vhd)

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

entity ALU is
       PORT (
       sel: in std_logic_vector (2 downto 0);
       x_alu: in std_logic_vector (3 downto 0);
       y_alu: in std_logic_vector (3 downto 0);
       carry: out std_logic;
       z_alu: out std_logic_vector (3 downto 0)
          );
end ALU;

architecture Behavioral of ALU is

       component FourBitAdder
   PORT (
       add_x: in std_logic_vector (3 downto 0);
       add_y: in std_logic_vector (3 downto 0);
       add_c: out std_logic;
       add_s: out std_logic_vector (3 downto 0)
              );
       end component;

       component FourBitSubtractor
   PORT (
       sub_x: in std_logic_vector (3 downto 0);
       sub_y: in std_logic_vector (3 downto 0);
       sub_c: out std_logic;
       sub_s: out std_logic_vector (3 downto 0)
              );
```

```
end component;

component LogicalShiftLeft
PORT (
x_shift: in std_logic_vector (3 downto 0);
y_shift: out std_logic_vector (3 downto 0)
              );
end component;

component BitwiseAnd
PORT (
x_and: in std_logic_vector (3 downto 0);
y_and: in std_logic_vector (3 downto 0);
z_and: out std_logic_vector (3 downto 0)
              );
end component;


component BitwiseOr
PORT (
x_or: in std_logic_vector (3 downto 0);
y_or: in std_logic_vector (3 downto 0);
z_or: out std_logic_vector (3 downto 0)
              );
end component;

component BitwiseXnor
PORT (
x_xnor: in std_logic_vector (3 downto 0);
y_xnor: in std_logic_vector (3 downto 0);
z_xnor: out std_logic_vector (3 downto 0)
              );
end component;

component BitwiseXor
PORT (
x_xor: in std_logic_vector (3 downto 0);
y_xor: in std_logic_vector (3 downto 0);
z_xor: out std_logic_vector (3 downto 0)
              );
end component;

component BitwiseNot
```

```
        PORT (
        x_not: in std_logic_vector (3 downto 0);
        y_not: out std_logic_vector (3 downto 0)
                    );
        end component;

        SIGNAL sig0 : std_logic_vector (3 downto 0);
        SIGNAL sig1 : std_logic_vector (3 downto 0);
        SIGNAL sig2 : std_logic_vector (3 downto 0);
        SIGNAL sig3 : std_logic_vector (3 downto 0);
        SIGNAL sig4 : std_logic_vector (3 downto 0);
        SIGNAL sig5 : std_logic_vector (3 downto 0);
        SIGNAL sig6 : std_logic_vector (3 downto 0);
        SIGNAL sig7 : std_logic_vector (3 downto 0);

        SIGNAL sic0 : std_logic;
        SIGNAL sic1 : std_logic;

begin

        o1 : FourBitAdder          PORT MAP (x_alu, y_alu, sic0,
           sig0); -- add
        o2 : FourBitSubtractor  PORT MAP (x_alu, y_alu, sic1, sig1);
           -- subtract
        o3 : BitwiseNot                PORT MAP (x_alu, sig2);
                        -- bitwise NOT
        o4 : BitwiseAnd                PORT MAP (x_alu, y_alu,
           sig3);        -- bitwise AND
        o5 : BitwiseOr                 PORT MAP (x_alu, y_alu,
           sig4);        -- bitwise OR
        o6 : BitwiseXor                PORT MAP (x_alu, y_alu,
           sig5);        -- bitwise XOR
        o7 : BitwiseXnor               PORT MAP (x_alu, y_alu,
           sig6);        -- bitwise XNOR
        o8 : LogicalShiftLeft   PORT MAP (x_alu, sig7);
                        -- logical shift


        with sel select
               z_alu <= sig0 when "000", -- "" or ''
                             sig1 when "001",
                             sig2 when "010",
                             sig3 when "011",
```

```
                                sig4 when "100",
                                sig5 when "101",
                                sig6 when "110",
                                sig7 when others;

        with sel(0) select
                carry <= sic0 when '0',
                                sig1 when others;

end Behavioral;
```

Code 2: (ALUtest.vhd)

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

entity ALUtest is
--  Port ( );
end ALUtest;

architecture Behavioral of ALUtest is

component ALU
    PORT(
        sel: in std_logic_vector (2 downto 0);
        x_alu: in std_logic_vector (3 downto 0);
        y_alu: in std_logic_vector (3 downto 0);
        carry: out std_logic;
        z_alu: out std_logic_vector (3 downto 0)
    );
end component;

SIGNAL sel: std_logic_vector (2 downto 0);
SIGNAL x_alu: std_logic_vector (3 downto 0);
SIGNAL y_alu: std_logic_vector (3 downto 0);
SIGNAL carry: std_logic;
SIGNAL z_alu: std_logic_vector (3 downto 0);

begin

    UUT: ALU PORT MAP(
```

```
    sel=>sel,
    x_alu=>x_alu,
    y_alu=>y_alu,
    carry=>carry,
    z_alu=>z_alu
);

testBench: process
begin

sel<="000";
x_alu<="1010";
y_alu<="0001";

wait for 100ns;
sel<="001";
x_alu<="1010";
y_alu<="0001";

wait for 100ns;
sel<="010";
x_alu<="1010";
y_alu<="0001";

wait for 100ns;
sel<="011";
x_alu<="1010";
y_alu<="0001";

wait for 100ns;
sel<="100";
x_alu<="1010";
y_alu<="0001";

wait for 100ns;
sel<="101";
x_alu<="1010";
y_alu<="0001";

wait for 100ns;
sel<="110";
x_alu<="1010";
y_alu<="0001";
```

```
    wait for 100ns;
    sel<="111";
    x_alu<="1010";
    y_alu<="0001";

    wait for 100ns;
    end process;

end Behavioral;
```

Code 3: (FourBitAdder.vhd)

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

entity FourBitAdder is
    PORT (
        add_x: in std_logic_vector (3 downto 0);
        add_y: in std_logic_vector (3 downto 0);
        add_c: out std_logic;
        add_s: out std_logic_vector (3 downto 0)
    );
end FourBitAdder;

architecture Behavioral of FourBitAdder is
    signal add_signal_y : std_logic_vector (3 downto 0);
    signal add_signal_c : std_logic_vector (2 downto 0);
    component FullAdder
    PORT (
        x_i : in std_logic;
        y_i : in std_logic;
        c_i : in std_logic;
        c_i1: out std_logic;
        s_i : out std_logic
    );
    end component;


begin
```

```
    process is
        begin
            for i in 0 to 3 loop
                add_signal_y(i) <= add_y(i);
                end loop;
                wait;
    end process;

    --add_sign(0) <= '1';

    --add_generate:
    --for i in 0 to 3 generate
    --    addber: FullAdder PORT MAP (add_x(i), add_y(i),
        add_sign(i), add_sign(i+1), add_s(i));
    --end generate;

    addber1: FullAdder PORT MAP (add_x(0), add_signal_y(0), '0'
        , add_signal_c(0), add_s(0));
    addber2: FullAdder PORT MAP (add_x(1), add_signal_y(1),
        add_signal_c(0), add_signal_c(1), add_s(1));
    addber3: FullAdder PORT MAP (add_x(2), add_signal_y(2),
        add_signal_c(1), add_signal_c(2), add_s(2));
    addber4: FullAdder PORT MAP (add_x(3), add_signal_y(3),
        add_signal_c(2), add_c , add_s(3));

    end Behavioral;
```

Code 4: (FourBitSubtractor.vhd)

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


entity FourBitSubtractor is
    PORT (
        sub_x: in std_logic_vector (3 downto 0);
        sub_y: in std_logic_vector (3 downto 0);
        sub_c: out std_logic;
        sub_s: out std_logic_vector (3 downto 0)
    );
end FourBitSubtractor;
```

```
architecture Behavioral of FourBitSubtractor is
    signal sub_signal_y : std_logic_vector (3 downto 0);
    signal sub_signal_c : std_logic_vector (2 downto 0);
    component FullAdder
    PORT (
        x_i: in std_logic;
        y_i: in std_logic;
        c_i: in std_logic;
        c_i1: out std_logic;
        s_i : out std_logic
    );
    end component;


begin

    process is
        begin
            for i in 0 to 3 loop
                sub_signal_y(i) <= not sub_y(i);
                end loop;
                wait;
    end process;

    --sub_sign(0) <= '1';

    --sub_generate:
    --for i in 0 to 3 generate
    --    subber: FullAdder PORT MAP (sub_x(i), sub_y(i),
        sub_sign(i), sub_sign(i+1), sub_s(i));
    --end generate;

    subber1: FullAdder PORT MAP (sub_x(0), sub_signal_y(0), '1'
        , sub_signal_c(0), sub_s(0));
    subber2: FullAdder PORT MAP (sub_x(1), sub_signal_y(1),
        sub_signal_c(0), sub_signal_c(1), sub_s(1));
    subber3: FullAdder PORT MAP (sub_x(2), sub_signal_y(2),
        sub_signal_c(1), sub_signal_c(2), sub_s(2));
    subber4: FullAdder PORT MAP (sub_x(3), sub_signal_y(3),
        sub_signal_c(2), sub_c , sub_s(3));

    end Behavioral;
```

Code 5: (FullAdder.vhd)

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


entity FullAdder is
    PORT (
        x_i: in std_logic;
        y_i: in std_logic;
        c_i: in std_logic;
        c_i1: out std_logic;
        s_i : out std_logic
    );
end FullAdder;

architecture Behavioral of FullAdder is

signal sign : std_logic_vector (2 downto 0);

component HalfAdder
    PORT (
        x: in std_logic;
        y: in std_logic;
        c: out std_logic;
        s: out std_logic
    );
end component;

begin
  -- degenerate:
  -- for i in 0 to 1 generate
  --     h: HalfAdder PORT MAP (x_i, y_i, sign(0), sign(1));
  -- end generate;


    h1 : HalfAdder PORT MAP (x_i, y_i, sign(0), sign(1));
    h2 : HalfAdder PORT MAP (c_i, sign(1), sign(2) ,s_i);

    c_i1 <= sign(0) or sign(2);

end Behavioral;
```

Code 6: (HalfAdder.vhd)

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


entity HalfAdder is
    PORT (
        x: in std_logic;
        y: in std_logic;
        c: out std_logic;
        s: out std_logic
    );
end HalfAdder;

architecture Behavioral of HalfAdder is
begin
    c <= x and y;
    s <= x xor y;
end Behavioral;
```

Code 7: (BitwiseNot.vhd)

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


entity BitwiseNot is
       PORT (
              x_not: in std_logic_vector (3 downto 0);
              y_not: out std_logic_vector (3 downto 0)
              );
end BitwiseNot;

architecture Behavioral of BitwiseNot is

begin

       process is
```

```
        begin
            for i in 0 to 3 loop
                y_not(i) <= NOT x_not(i);
                end loop;
                wait;
    end process;

end Behavioral;
```

Code 8: (BitwiseAnd.vhd)

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

entity BitwiseAnd is
        PORT (
                        x_and: in std_logic_vector (3 downto 0);
                        y_and: in std_logic_vector (3 downto 0);
                        z_and: out std_logic_vector (3 downto 0)
                        );
end BitwiseAnd;

architecture Behavioral of BitwiseAnd is

begin

        process is
        begin
            for i in 0 to 3 loop
                z_and(i) <= x_and(i) AND y_and(i);
                end loop;
                wait;
    end process;

end Behavioral;
```

Code 9: (BitwiseOr.vhd)

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

entity BitwiseOr is
       PORT (
                      x_or: in std_logic_vector (3 downto 0);
                      y_or: in std_logic_vector (3 downto 0);
                      z_or: out std_logic_vector (3 downto 0)
                      );
end BitwiseOr;

architecture Behavioral of BitwiseOr is

begin

       process is
       begin
           for i in 0 to 3 loop
               z_or(i) <= x_or(i) OR y_or(i);
               end loop;
               wait;
    end process;

end Behavioral;
```

Code 10: (BitwiseXor.vhd)

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

entity BitwiseXor is
       PORT (
                      x_xor: in std_logic_vector (3 downto 0);
                      y_xor: in std_logic_vector (3 downto 0);
                      z_xor: out std_logic_vector (3 downto 0)
                             );
end BitwiseXor;

architecture Behavioral of BitwiseXor is

begin
```

```
        process is
        begin
            for i in 0 to 3 loop
                z_xor(i) <= x_xor(i) XOR y_xor(i);
                end loop;
                wait;
    end process;


end Behavioral;
```

Code 11: (BitwiseXnor.vhd)

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

entity BitwiseXnor is
        PORT (
                        x_xnor: in std_logic_vector (3 downto 0);
                        y_xnor: in std_logic_vector (3 downto 0);
                        z_xnor: out std_logic_vector (3 downto 0)
                               );
end BitwiseXnor;

architecture Behavioral of BitwiseXnor is

begin

        process is
        begin
            for i in 0 to 3 loop
                z_xnor(i) <= x_xnor(i) XNOR y_xnor(i);
                end loop;
                wait;
    end process;


end Behavioral;
```

Code 12: (LogicalLeftShift.vhd)

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

entity LogicalShiftLeft is
       PORT (
              x_shift: in std_logic_vector (3 downto 0);
              y_shift: out std_logic_vector (3 downto 0)
              );
end LogicalShiftLeft;

architecture Behavioral of LogicalShiftLeft is

begin

       y_shift(0) <= '0';
       process is
       begin
           for i in 0 to 2 loop
              y_shift(i+1) <= x_shift(i);
              end loop;
              wait;
   end process;

end Behavioral;
```

References:

- "Arithmetic logic unit." Wikipedia, Wikimedia Foundation, 26 Oct. 2022, en.wikipedia.org/wiki/Arithmetic_logic_unit. Accessed 7 Nov. 2022.
- "Bitwise operation." Wikipedia, Wikimedia Foundation, 27 Oct. 2022, en.wikipedia.org/wiki/Bitwise_operation. Accessed 8 Nov. 2022.
- https://github.com/CankutBoraTuncer/Bilkent-EEE102-Labs