Emre Akdemir

22302606

EE102-02

EE-102 LAB 4 REPORT: ARITHMETIC LOGIC UNIT

05/03/2025

Purpose

The aim of this lab was to create an Arithmetic Logic Unit (ALU) using VHDL and embedding it on BASYS3. This unit can do 8 different operations, including addition, subtraction, bitwise XOR, AND, OR, logical left shift, ones complement and rotate operations.

Methodology

Firstly, I looked for the ALU explanation from Wikipedia and decided which functions I should include into ALU. I planned to include addition, subtraction, bitwise XOR, AND, OR, logical left shift, ones complement and rotate operations into my ALU. Inputs of the operations are 4-bit. First, I created a half adder module, then created a full adder using two half adders. To create four-bit addition module, I used 4 full adders. To create subtraction, I also used 4 full adders. The difference from the addition is, I inverted B by taking complement and I set carry in input as “1”. So even the inputs are unsigned, I get a signed number result. Then I created rotate, logical left shift, ones complement, XOR, OR, AND sub modules and the top module, respectively. The operations will be performed by using two or one 4-bit number. After writing all modules, I run the synthesis and implementation. The inputs and outputs were in right place. I checked the outputs using a testbench. Testbench created values starting from 0 to 2047, 11-bit numbers increasing one by one. Output signals were correct for each selection. After these, I embedded the code on BASYS3 and successfully got right results.

Design Specifications

ALU’s are widely used in computer’s Central Processing Unit’s (CPU), making aritmetic operations and logical shifts. In this lab, I designed an ALU which can perform addition, subtraction, bitwise XOR, AND, OR, logical left shift, ones complement and rotate.

Inputs and Outputs

There are three inputs and one output in the design. Switchs starting from pin 0 to pin 10, I divided them into three parts, first 4 switches (from 3 to 0) corresponds to the first input, 3 being most and 0 being least significant number, and switches (from 7 to 4), corresponds to the second input, 7 being most and 4 being least significant number. The selection input, operating which function out of 8 should be performed is set to third part, (from 10 to 8). Output is 5 bit long, which is set as last five pins of the LEDs. The main three inputs are in std_logic_vector format. Others are in std_logic format.

Submodules and Top module

There are 8 sub modules and 1 top module. For the four-bit adder and subtraction, there are 8 full-adder, and 16 half adder modules are also used as sub modules. In main module, there are 8 different signals corresponds to the selection of the user.

When selection is “000”, the ALU performs addition. Inputs are a_i (first four bit) and b_i (second four bit). Output is a_i + b_i.

When selection is “001”, the ALU performs subtraction. Inputs are a_i (first four bit) and b_i (second four bit). Output is a_i - b_i. Even though the inputs are unsigned, the output is signed.

When selection is “010”, the ALU performs rotation. Input is a_i (first four bit) and the output is (a_i(2), a_i(1), a_i(0), a_i(3)).

When selection is “011”, the ALU performs logical left shift. Input is a_i (first four bit) and the output is (a_i(2), a_i(1), a_i(0), ‘0’).

When selection is “100”, the ALU performs bitwise ones complement. Input is a_i (first four bit) and the output is (not(a_i)).

When selection is “101”, the ALU performs bitwise XOR. Inputs are a_i (first four bit) and b_i (second four bit) the output is (a_i(3) xor b_i(3) ,a_i(2) xor b_i(2), a_i(1) xor b_i(1), a_i(0) xor b_i(0)).

When selection is “110”, the ALU performs bitwise OR. Inputs are a_i (first four bit) and b_i (second four bit) the output is (a_i(3) or b_i(3) ,a_i(2) or b_i(2), a_i(1) or b_i(1), a_i(0) or b_i(0)).

When selection is “110”, the ALU performs bitwise AND. Inputs are a_i (first four bit) and b_i (second four bit) the output is (a_i(3) and b_i(3) ,a_i(2) and b_i(2), a_i(1) and b_i(1), a_i(0) and b_i(0)).

Inputs, outputs and I/O porting:

Inputs:

a_i(0-3): Switches (V17, V16, W16,W17)

b_i(0-3): Switches (W15, V15, W14,W13)

selection(0-2): Switches(V2,T3,T2)

Outputs:

output(0-4): Leds(U3,P3,N3,P1,L1)

Results

RTL schematics clearly shows that there are 8 different multiplexers (MUX) are used, multiplexers correspond to selection part of the design. Based on selection, the signal is carried through other multiplexers, and finally as output. The blue boxes correspond to the operations. Based on selection input, that specific operation in the blue box is performed. The coming inputs are splitting into three, being as “a_i”, ”b_i" and “selection”. The final output is one, being as “output”. From left to right, AND, OR, XOR, ones complement, logical left shift, rotate, subtraction and four bit adder operation are aligned.

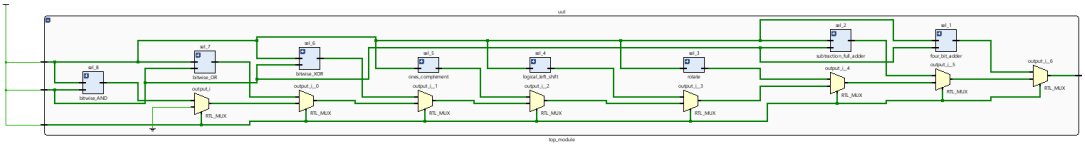

Figure 1: RTL Schematics of the design

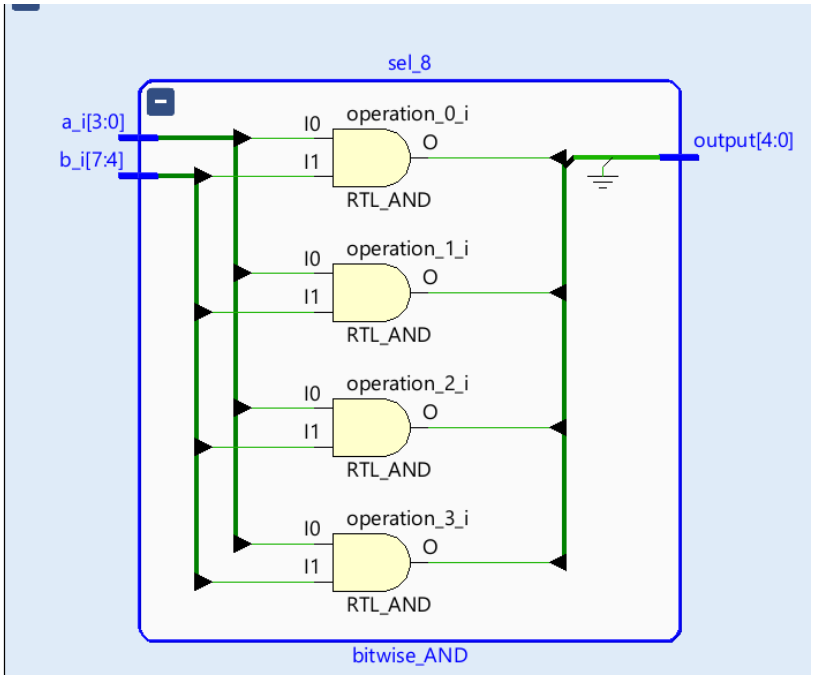

Figure 2: RTL Schematics of bitwise AND

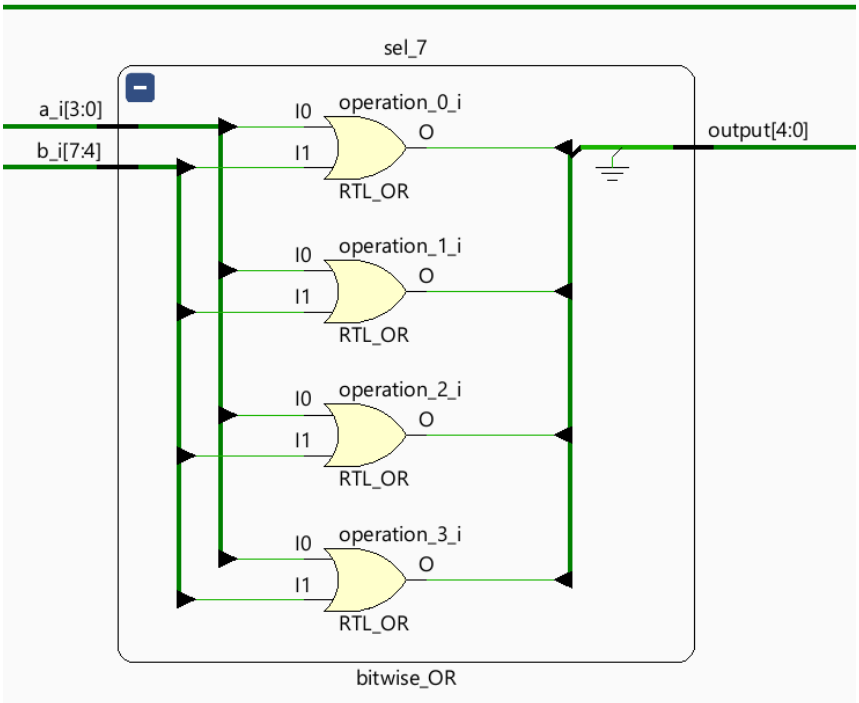

Figure 3: RTL Schematics of bitwise OR

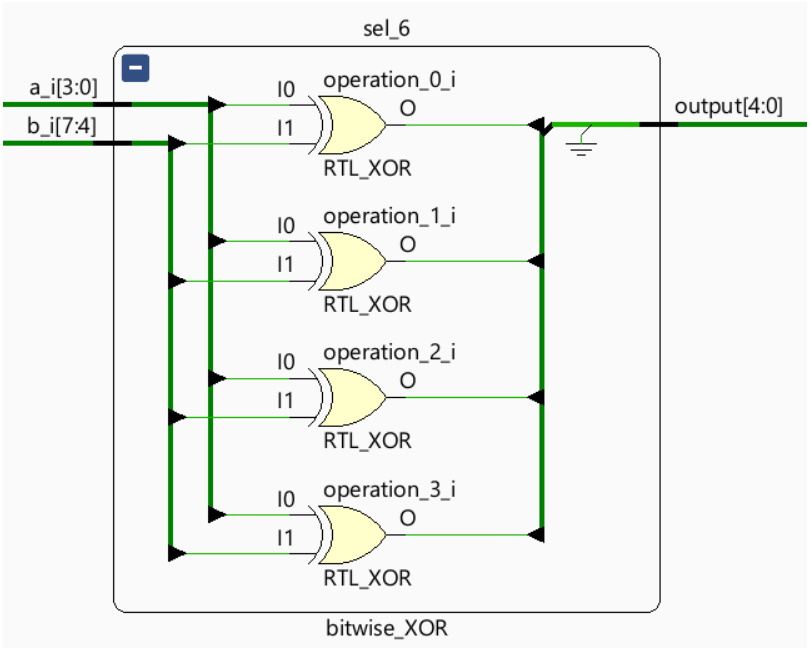

Figure 3: RTL Schematics of bitwise XOR

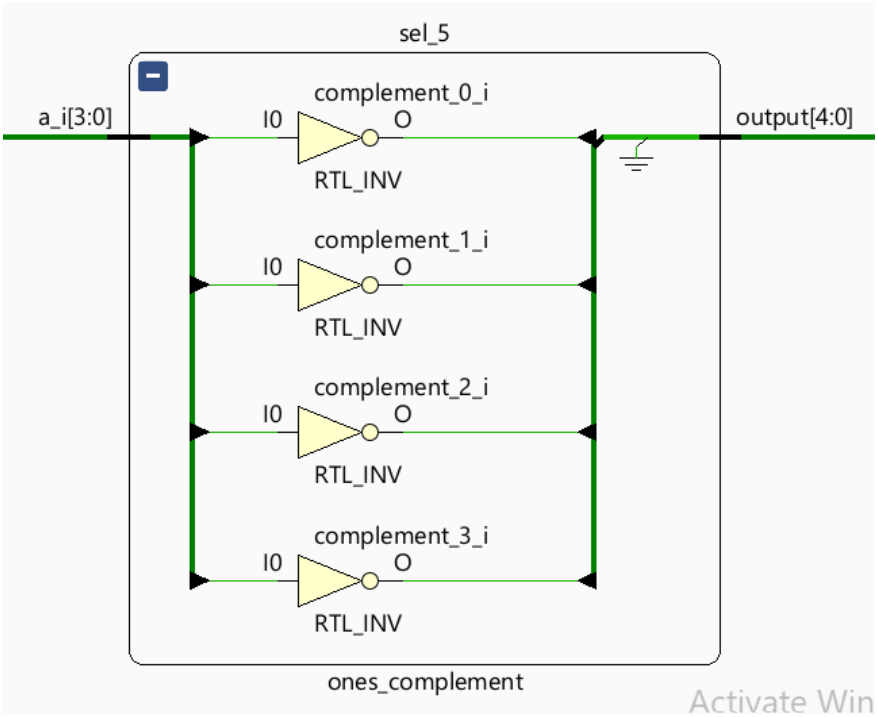

Figure 4: RTL Schematics of bitwise Ones Complement

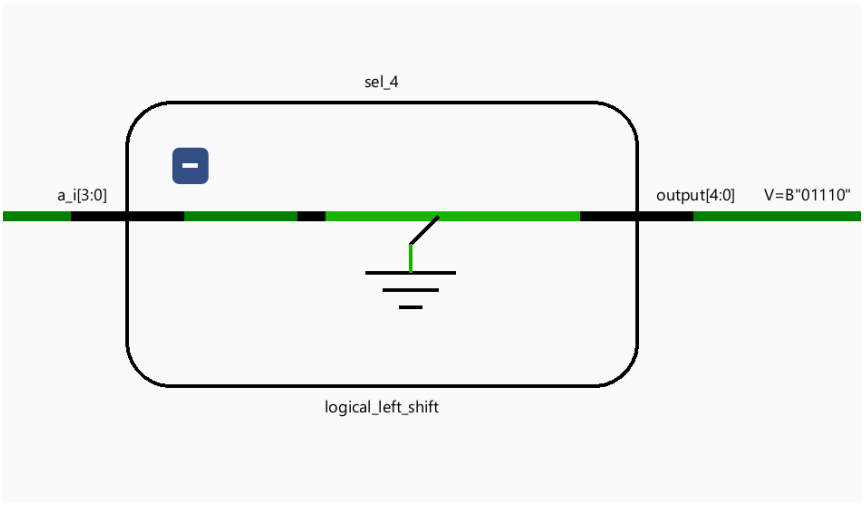

Figure 5: RTL Schematics of Logical Left Shift

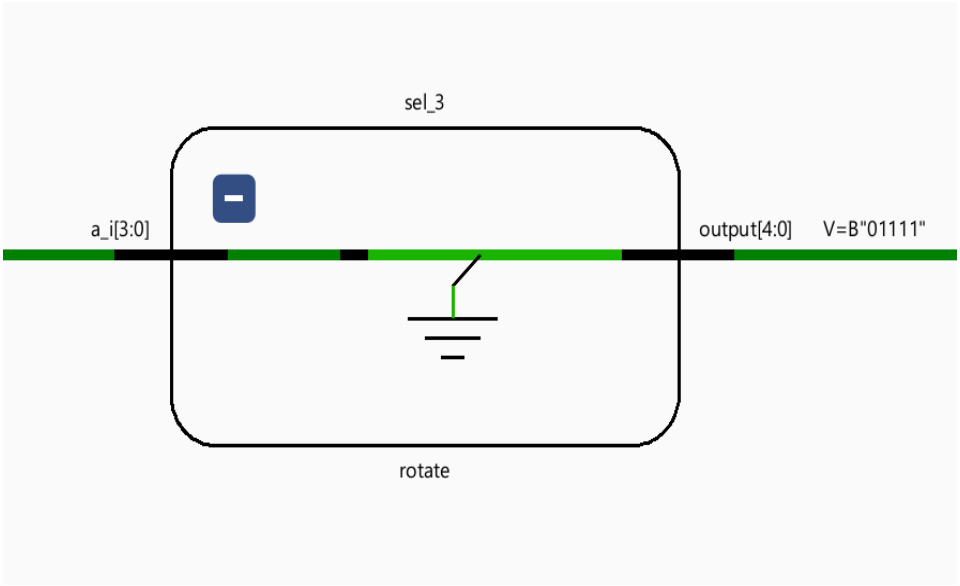

Figure 6: RTL Schematics of Rotate

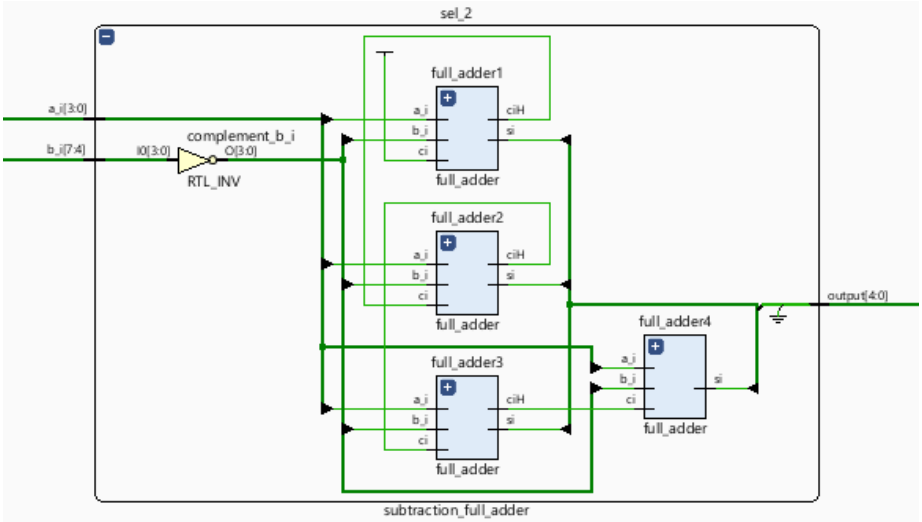

Figure 7: RTL Schematics of Subtraction using Full Adders

Note that I set carry in input as “1” and took the complement of the b_i input.

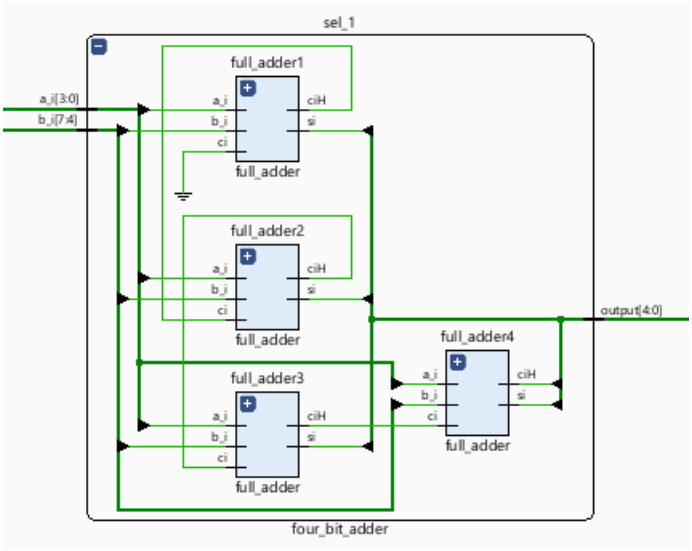

Figure 8: RTL Schematics of 4-bit Adder

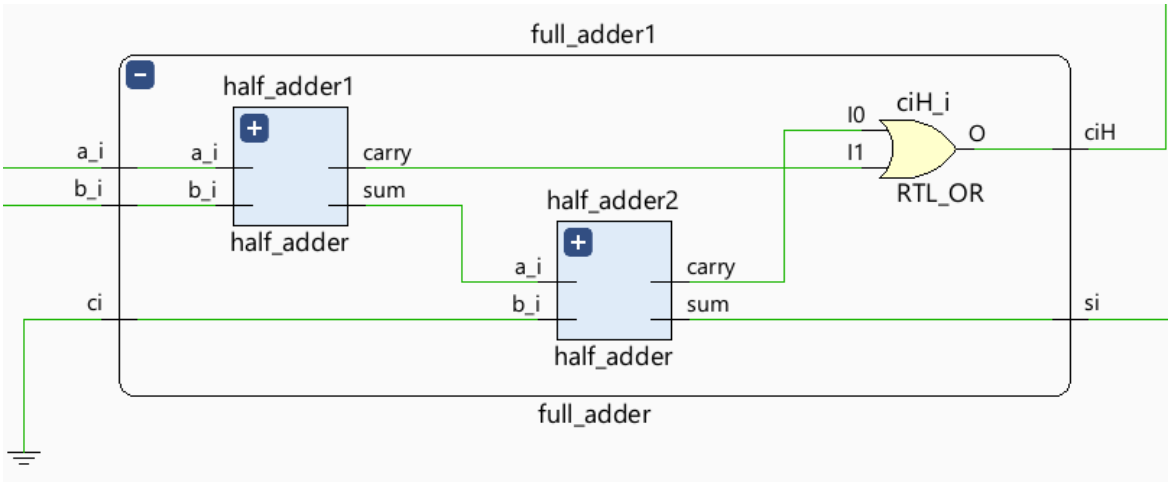

Figure 9: RTL Schematics of Full Adder

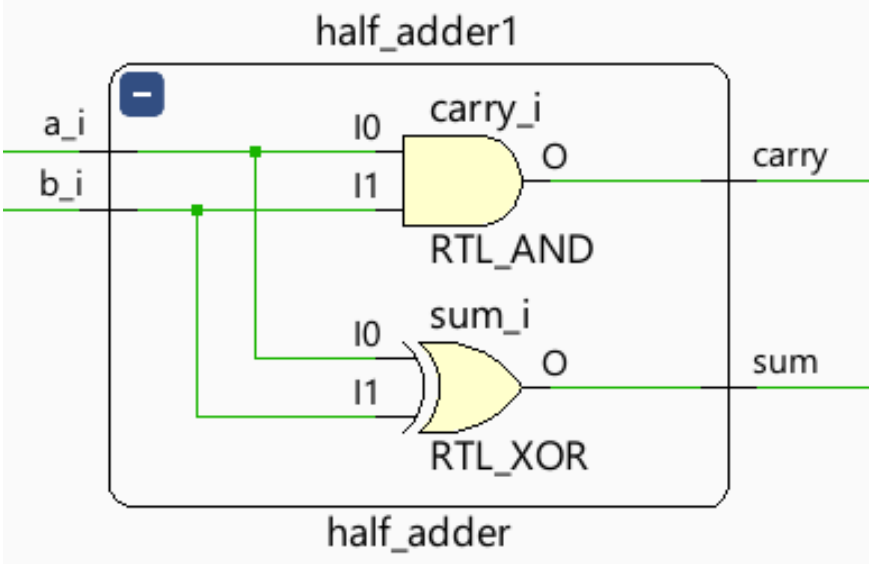

Figure 10: RTL Schematics of Half Adder

Testbench

To test the design, I created a testbench, which generates a 11-bit number by starting from 0 to increasing it up to 2047. Then I divided this number into three signals and denoted them to the input_1 (first four bit), input_2 (second four bit) and selection (last three bit). By doing so, I was able to display all the possible waveforms.

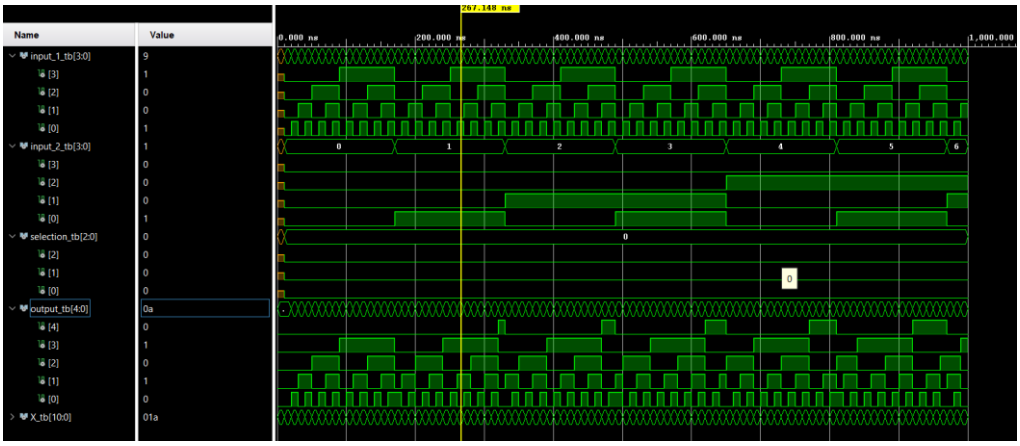

Figure 11: Waveforms when selection is “000” (addition, input_1 is “1001”, input_2 is “0001”, output is “01010”)

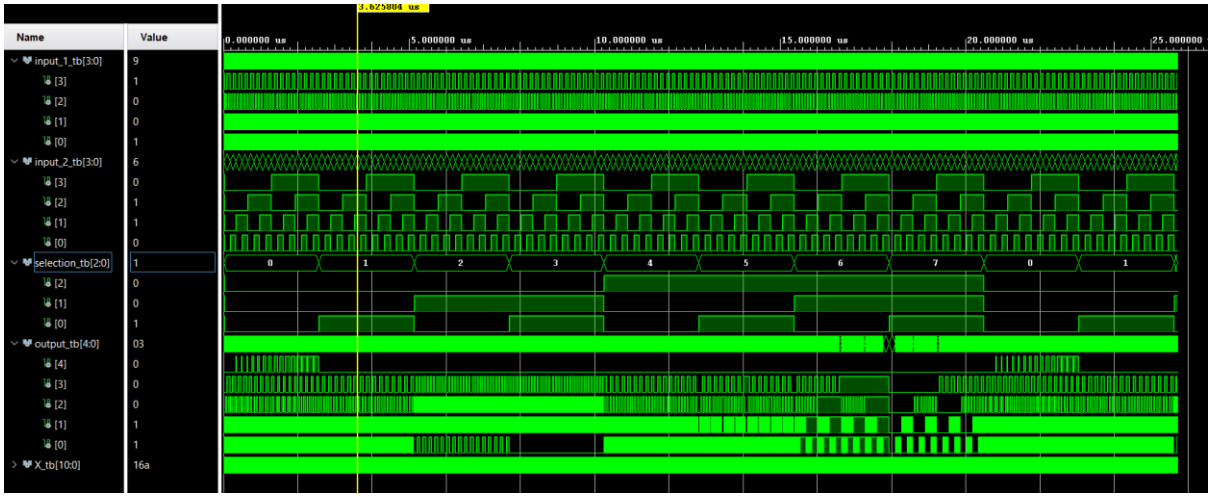

Figure 12: Waveforms when selection is “001” (subtraction, does (input_1 - input_2) input_1 is “1001”, input_2 is “0110”, output is “11000”)

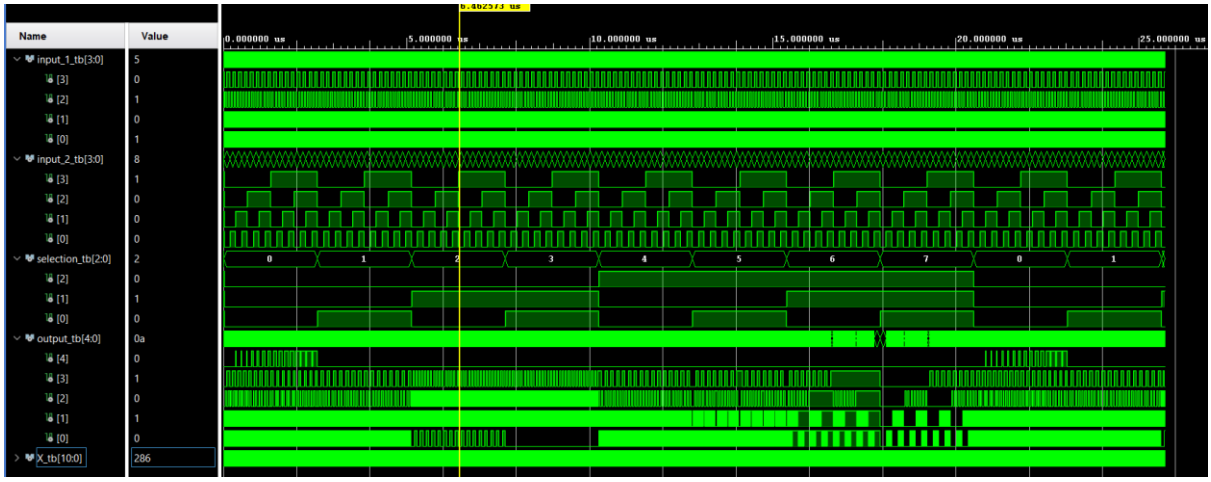

Figure 13: Waveforms when selection is “010” (rotation, input_1 is “1001”, output is “01010”)

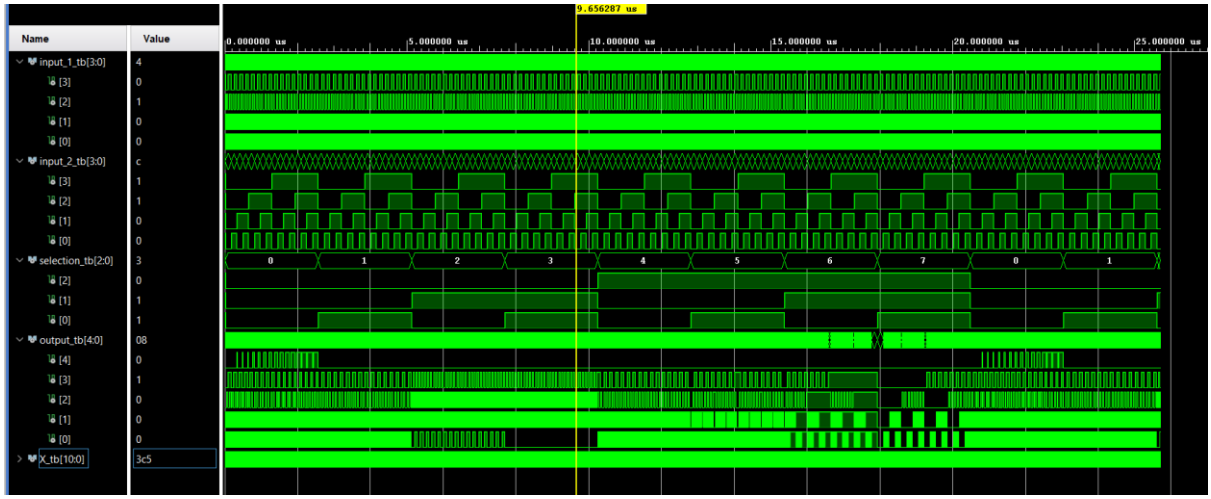

Figure 14: Waveforms when selection is “011” (logical left shift, input_1 is “0010”, output is “00010”)

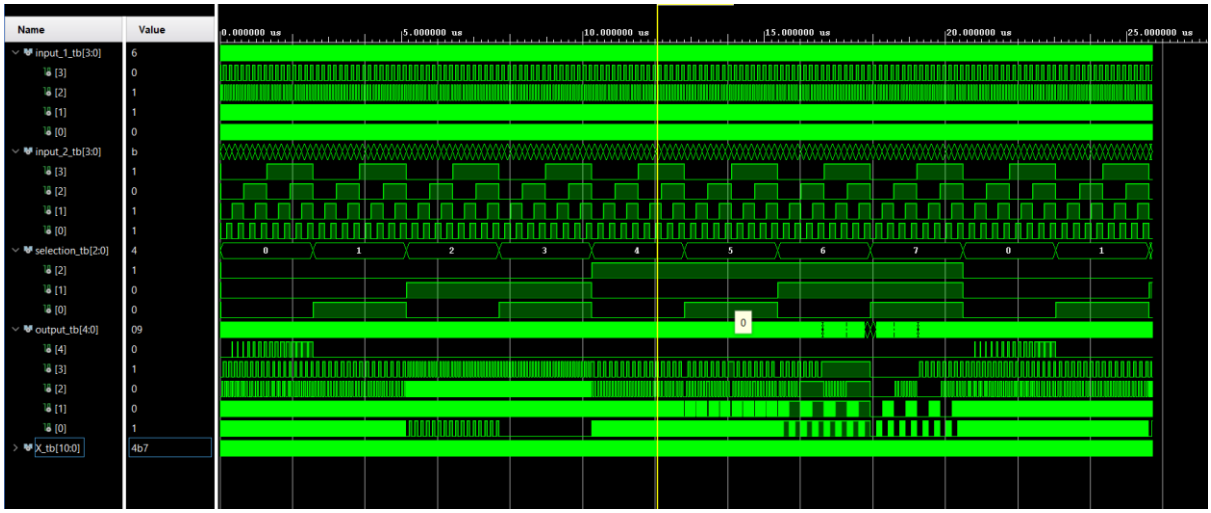

Figure 15: Waveforms when selection is “100” (ones complement, input_1 is “0110”, output is “10010”)

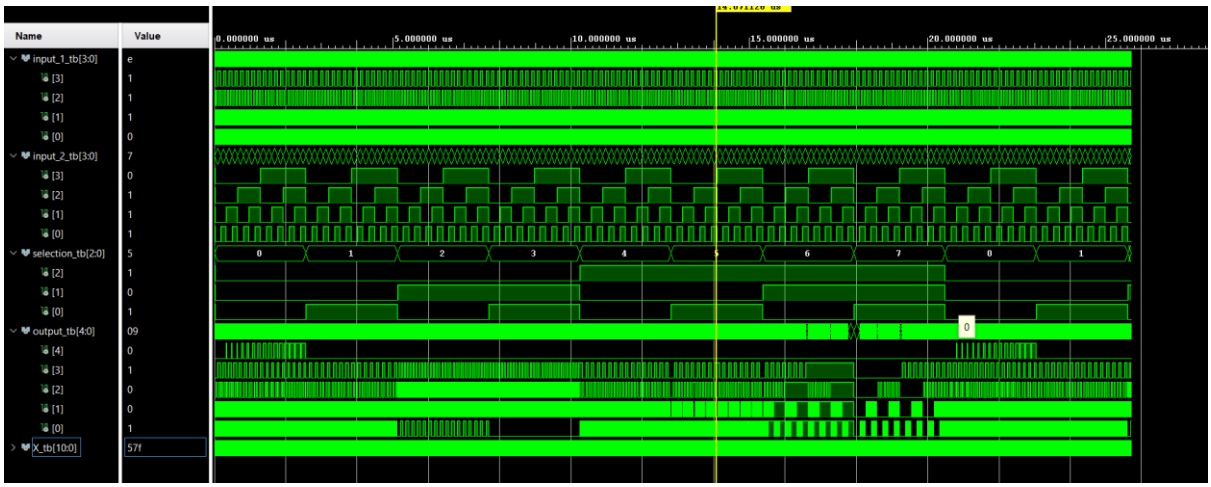

Figure 16: Waveforms when selection is “101” (bitwise XOR, input_1 is “0111”, input_2 is “1110” output is “10010”)

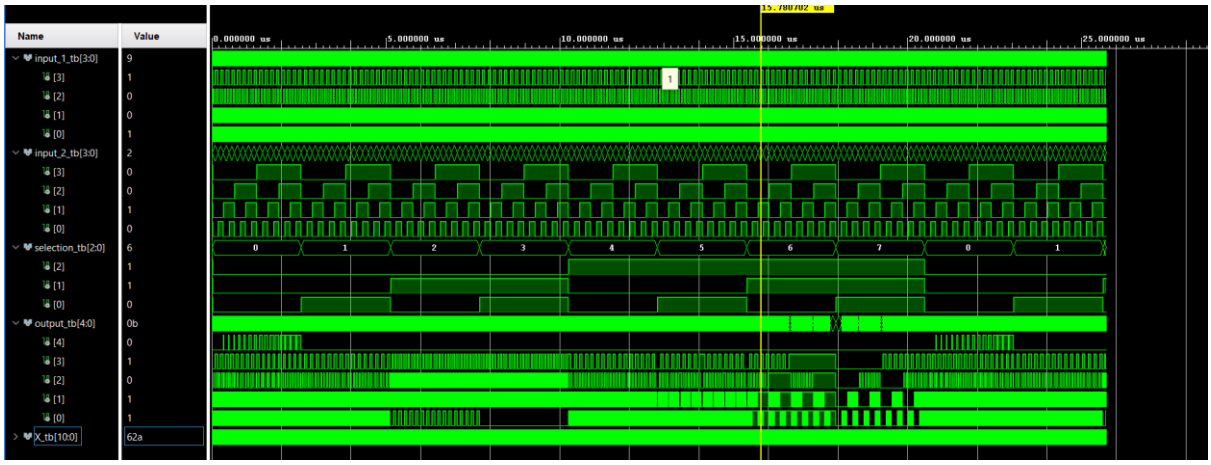

Figure 17: Waveforms when selection is “110” (bitwise OR, input_1 is “1001”, input_2 is “0100” output is “11010”)

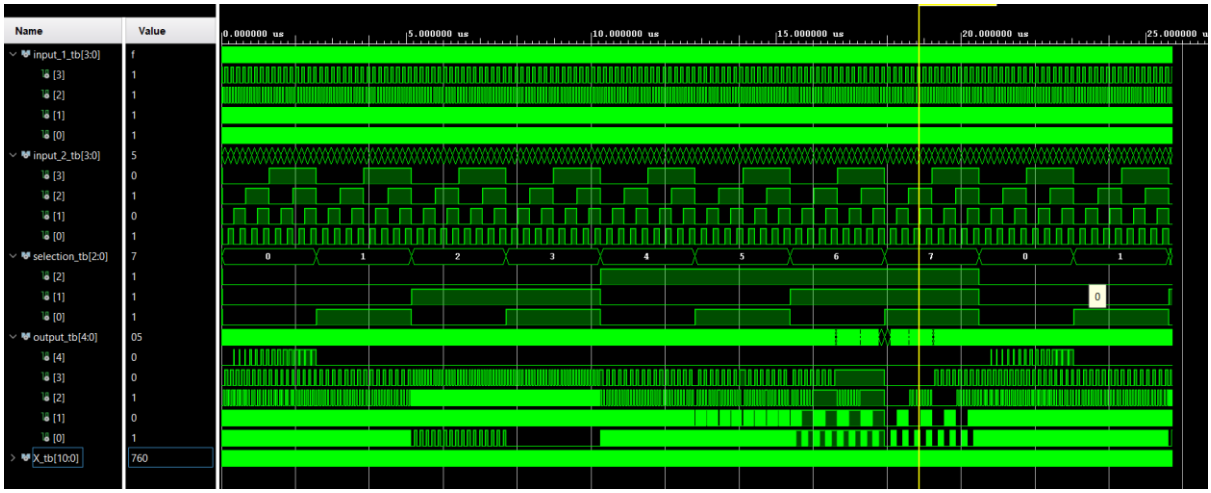

Figure 18: Waveforms when selection is “111” (bitwise AND, input_1 is “1111”, input_2 is “1010” output is “10100”)

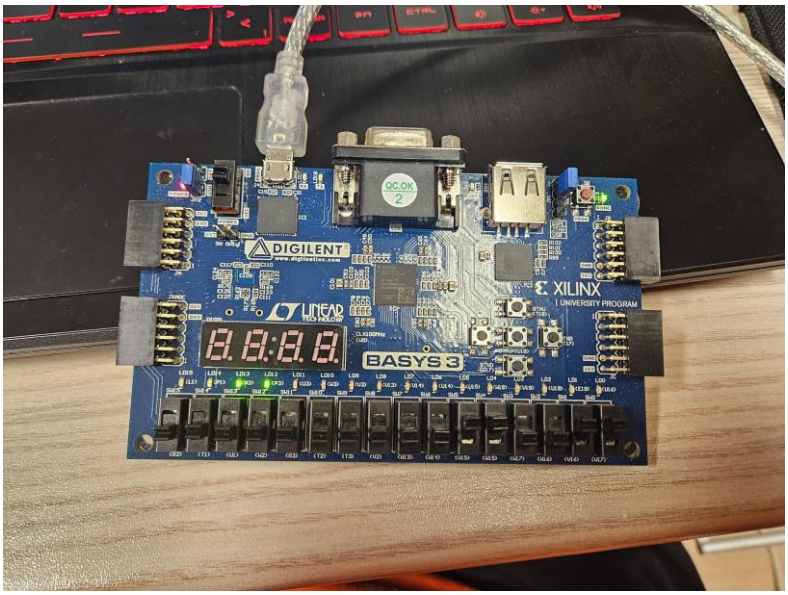

Figure 19: FPGA Board when selection is “000” (addition, input_1 is “1100”, input_2 is “1100” output is “01100”)

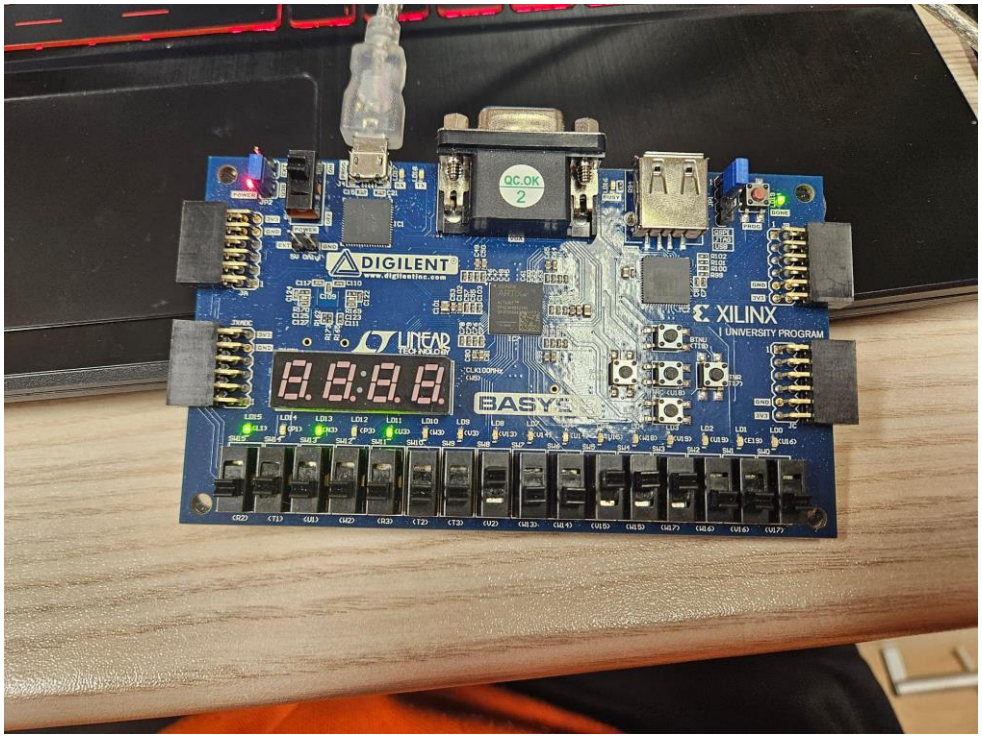

Figure 20: FPGA Board when selection is “001” (subtraction, input_1 is “1000”, input_2 is “0011” output is “10101”(The most significant digit is connected to Carry out, so neglect it. “0101” is the output.))

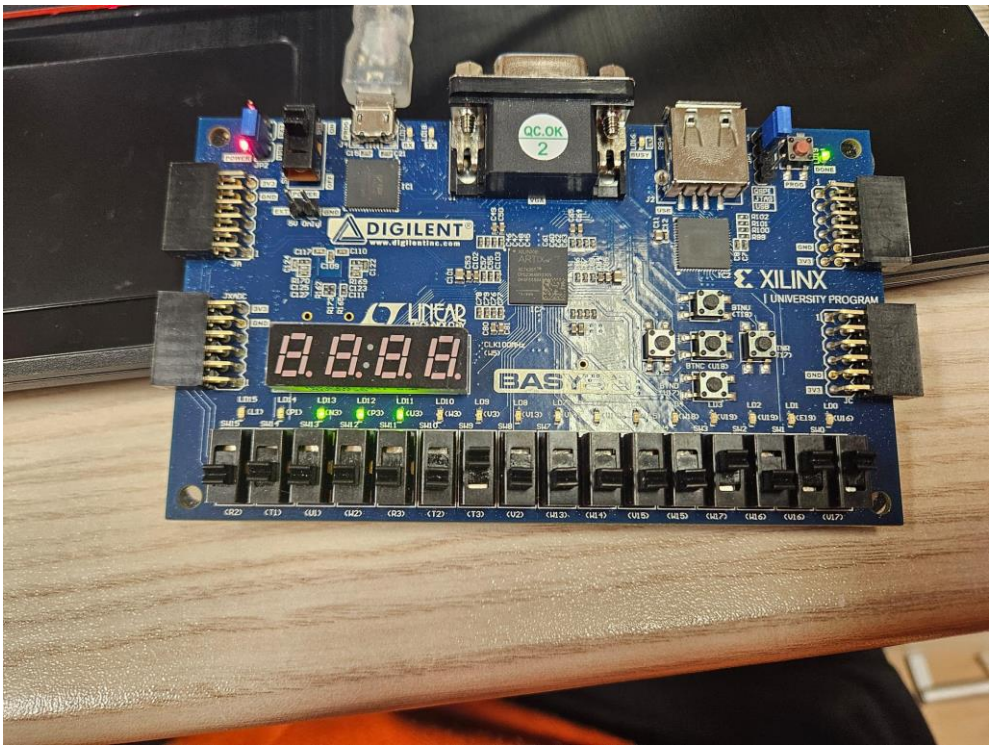

Figure 21: FPGA Board when selection is “010” (rotation, input_1 is “1011”, output is “00111”)

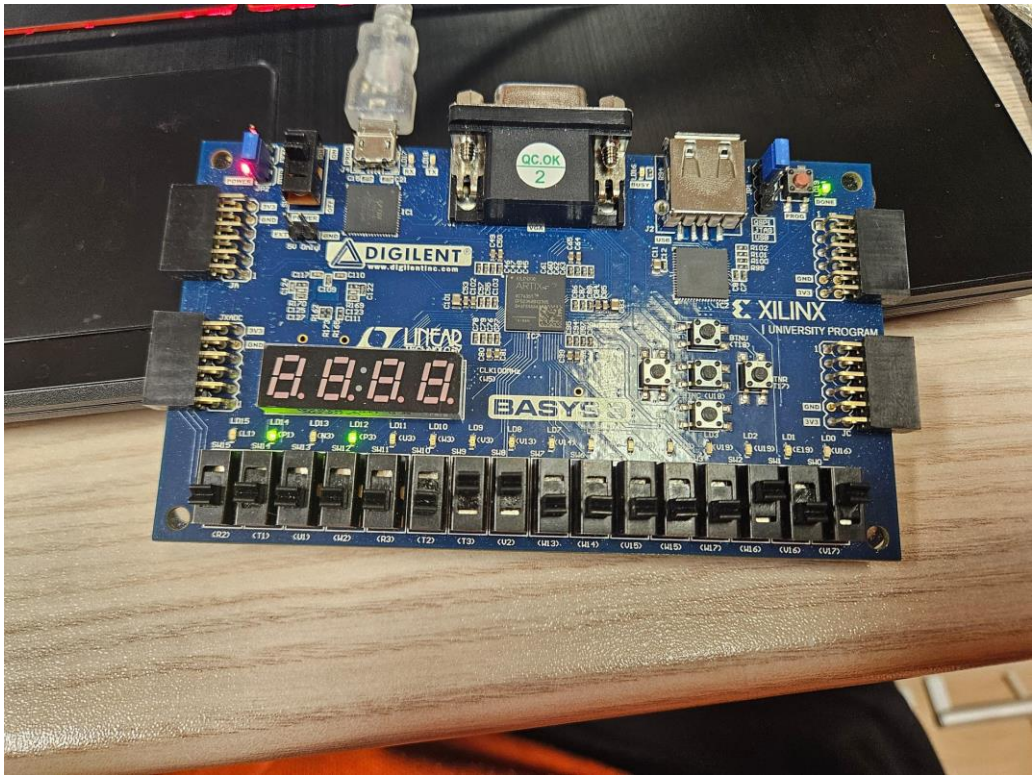

Figure 22: FPGA Board when selection is “011” (logical left shift, input_1 is “0101”, output is “01010”)

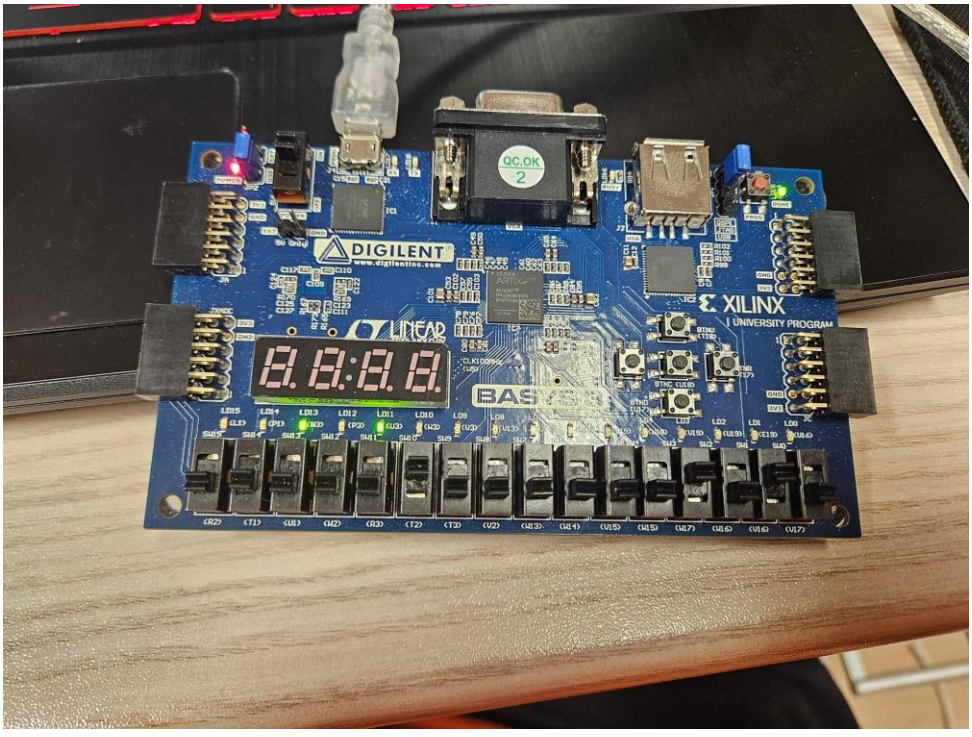

Figure 23: FPGA Board when selection is “100” (ones complement, input_1 is “1010”, output is “00101”)

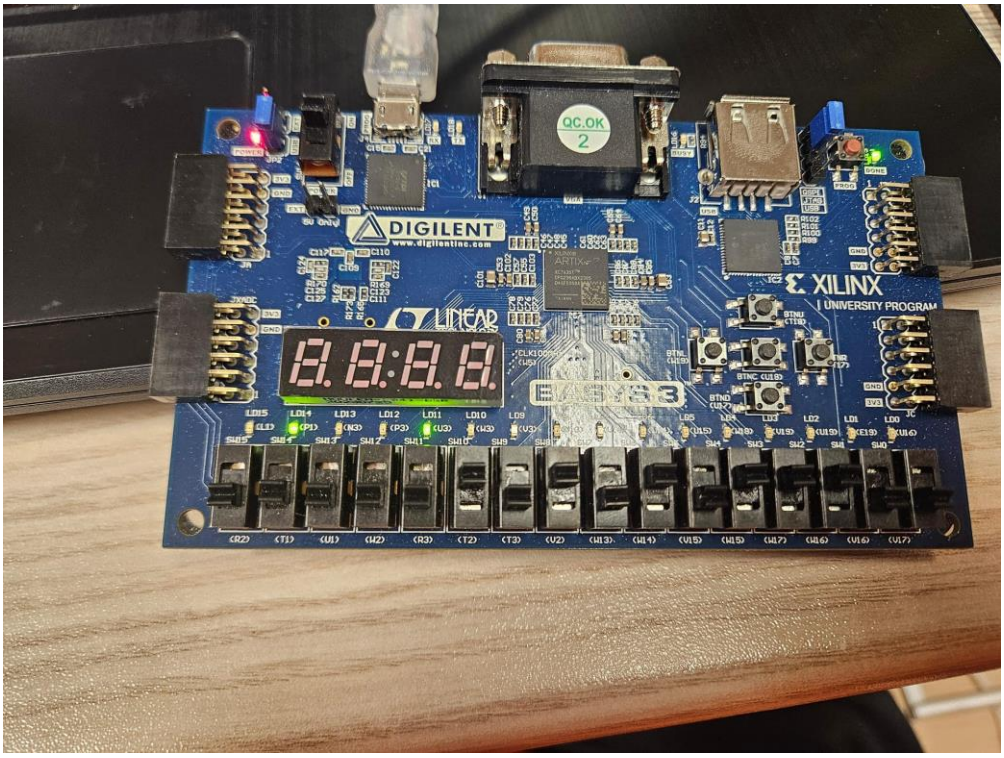

Figure 24: FPGA Board when selection is “101” (bitwise XOR, input_1 is “1100”, input_2 is “0101”, output is “01001”)

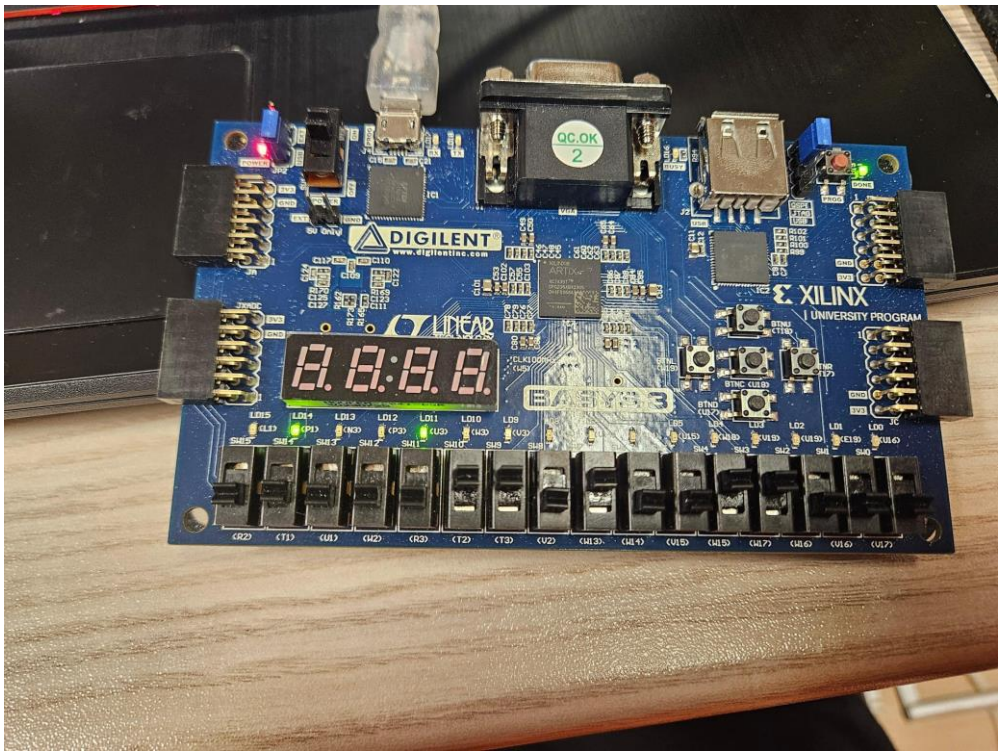

Figure 25: FPGA Board when selection is “110” (bitwise OR, input_1 is “1000”, input_2 is “1001”, output is “01001”)

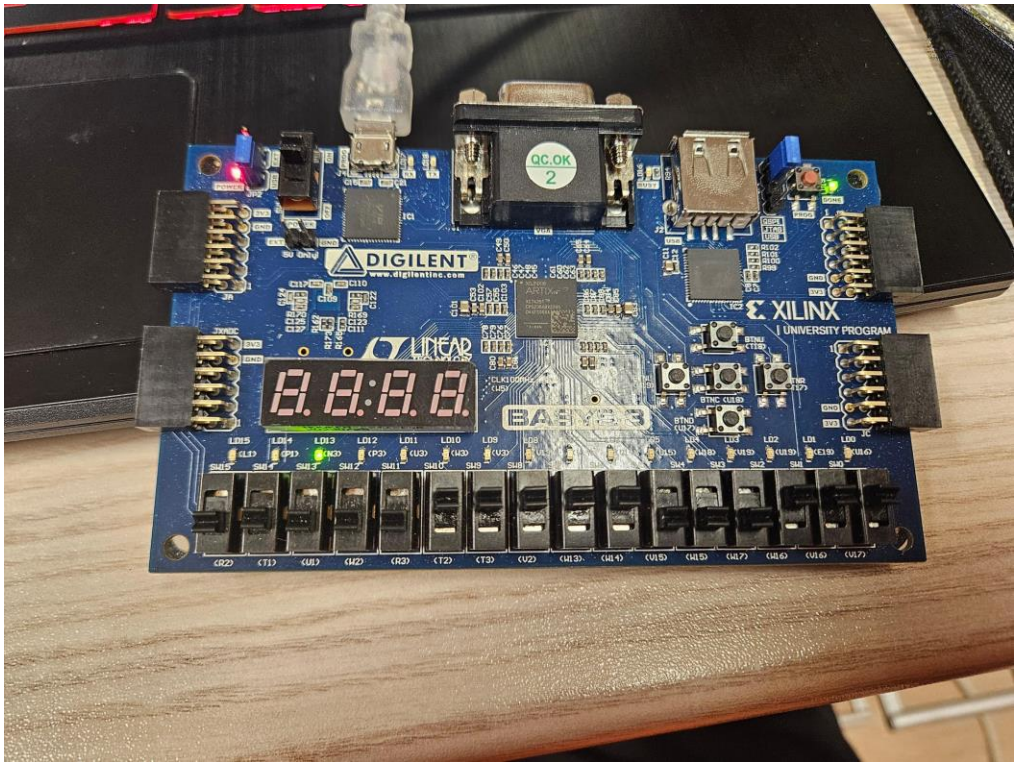

Figure 26: FPGA Board when selection is “111” (bitwise AND, input_1 is “0111”, input_2 is “1100”, output is “00100”)

Conclusion

Aim of this lab was to design an Arithmetic Logic Unit in modular fashion using VHDL, and embed it on BASYS3. ALU has arbitrary 8 different functions including Arithmetic and Logic shift, bitwise gate, addition and subtraction operations. To choose the desired function, user adjusts the switches. Ultimately, the testbench results were as expected, RTL schematics seen right, and operations on BASYS3 gave the correct results. Thanks to this lab, I got used to more modular hierarchy and syntax of the VHDL. Also, I did a lot of troubleshooting by testing after and after trying to fix the errors. As the result, the lab was succesfully done.

Code

Testbench:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity top_module_tb is
end entity;

architecture testbench of top_module_tb is
    -- DUT (Device Under Test) Component Declaration
    component top_module
        port (
            input_1  : in std_logic_vector(3 downto 0);
            input_2  : in std_logic_vector(3 downto 0);
            selection : in std_logic_vector(2 downto 0);
            output   : out std_logic_vector(4 downto 0)
        );
    end component;

    -- Signals
    signal input_1_tb  : std_logic_vector(3 downto 0);
    signal input_2_tb  : std_logic_vector(3 downto 0);
    signal selection_tb : std_logic_vector(2 downto 0);
    signal output_tb   : std_logic_vector(4 downto 0);
    signal X_tb        : std_logic_vector(10 downto 0);

begin
    -- Instantiate the DUT
    uut: top_module
        port map (
            input_1  => input_1_tb,
            input_2  => input_2_tb,
            selection => selection_tb,
            output   => output_tb
        );

    -- Test Process
    process
    begin
        for i in 0 to 2047 loop  -- Iterate through all possible 11-bit values
            X_tb <= std_logic_vector(to_unsigned(i, 11));  -- Assign i to X_tb

            -- Extract values
```

```
        input_1_tb  <= X_tb(3 downto 0);
        input_2_tb  <= X_tb(7 downto 4);
        selection_tb <= X_tb(10 downto 8);

        wait for 10 ns;  -- Wait to simulate propagation delay
        end loop;
   end process;
end architecture;
```

Top Module:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;

entity top_module is
   Port ( input_1 : in  std_logic_vector(3 downto 0);
        input_2 : in  std_logic_vector(7 downto 4);
        selection: in std_logic_vector(2 downto 0);
        output: out std_logic_vector(4 downto 0)
        );
end top_module;

architecture Behavioral of top_module is
   component four_bit_adder is
   Port (
        a_i: in std_logic_vector(3 downto 0);
        b_i: in std_logic_vector(7 downto 4);
        output: out std_logic_vector(4 downto 0)
        );
   end component;

   component subtraction_full_adder is
    Port (
      a_i: in std_logic_vector(3 downto 0);
      b_i: in std_logic_vector(7 downto 4);
      output: out std_logic_vector(4 downto 0)
      );
```

```
end component;

component rotate is
Port ( a_i : in STD_LOGIC_VECTOR(3 downto 0);
     output : out STD_LOGIC_VECTOR (4 downto 0));
end component;

component ones_complement is
   Port ( a_i : in STD_LOGIC_VECTOR (3 downto 0);
     output: out STD_LOGIC_VECTOR (4 downto 0)
    );
end component;

component logical_left_shift is
   Port ( a_i : in STD_LOGIC_VECTOR(3 downto 0);
        output : out STD_LOGIC_VECTOR (4 downto 0)
       );
end component;

component bitwise_XOR is
   Port ( a_i : in STD_LOGIC_VECTOR (3 downto 0);
        b_i : in STD_LOGIC_VECTOR (7 downto 4);
        output : out STD_LOGIC_VECTOR (4 downto 0)
       );
end component;

component bitwise_OR is
   Port ( a_i : in STD_LOGIC_VECTOR (3 downto 0);
        b_i : in STD_LOGIC_VECTOR (7 downto 4);
        output : out STD_LOGIC_VECTOR (4 downto 0)
       );
end component;



component bitwise_AND is
   Port ( a_i : in STD_LOGIC_VECTOR (3 downto 0);
        b_i : in STD_LOGIC_VECTOR (7 downto 4);
        output : out STD_LOGIC_VECTOR (4 downto 0)
       );
end component;
```

```
    signal input_signal_first: std_logic_vector(3 downto 0) := (others => '0');
    signal input_signal_second: std_logic_vector(7 downto 4) := (others => '0');
    signal output_signal_1: std_logic_vector(4 downto 0) := (others => '0');
    signal output_signal_2: std_logic_vector(4 downto 0) := (others => '0');
    signal output_signal_3: std_logic_vector(4 downto 0) := (others => '0');
    signal output_signal_4: std_logic_vector(4 downto 0) := (others => '0');
    signal output_signal_5: std_logic_vector(4 downto 0) := (others => '0');
    signal output_signal_6: std_logic_vector(4 downto 0) := (others => '0');
    signal output_signal_7: std_logic_vector(4 downto 0) := (others => '0');
    signal output_signal_8: std_logic_vector(4 downto 0) := (others => '0');
begin
    input_signal_first <= input_1;
    input_signal_second <= input_2;


    -- Instantiating the 4-bit adder
    sel_1 : four_bit_adder
       port map (
         a_i => input_signal_first,
         b_i => input_signal_second,
         output => output_signal_1
       );

    sel_2 : subtraction_full_adder
        port map (
          a_i => input_signal_first,
          b_i => input_signal_second,
          output => output_signal_2
       );

     sel_3 : rotate
        port map (
           a_i => input_signal_first,
           output => output_signal_3
         );
     sel_4 : logical_left_shift
        port map (
           a_i => input_signal_first,
           output => output_signal_4
         );
     sel_5 : ones_complement
```

```
        port map (
          a_i => input_signal_first,
          output => output_signal_5
         );
    sel_6 : bitwise_XOR
        port map (
          a_i => input_signal_first,
          b_i => input_signal_second,
          output => output_signal_6
         );
    sel_7 : bitwise_OR
        port map (
          a_i => input_signal_first,
          b_i => input_signal_second,
          output => output_signal_7
         );
    sel_8 : bitwise_AND
        port map (
          a_i => input_signal_first,
          b_i => input_signal_second,
          output => output_signal_8
         );
process(selection,output_signal_1,output_signal_2,output_signal_3,output_signal_4,o
utput_signal_5,output_signal_6,output_signal_7,output_signal_8)

begin

   if selection = "000" then
     output <=  output_signal_1;
   elsif selection = "001" then
     output <= output_signal_2;
   elsif selection = "010" then
     output <= output_signal_3;
   elsif selection = "011" then
     output <= output_signal_4;
   elsif selection = "100" then
     output <= output_signal_5;
   elsif selection = "101" then
     output <= output_signal_6;
   elsif selection = "110" then
     output <= output_signal_7;
```

```
   elsif selection = "111" then
     output <= output_signal_8;
   else
     output <= "00000";
   end if;
  end process;
end Behavioral;
```

Four-bit Adder:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity four_bit_adder is
    Port (
          a_i: in std_logic_vector(3 downto 0);
          b_i: in std_logic_vector(7 downto 4);
          output: out std_logic_vector(4 downto 0)
          );
end four_bit_adder;

architecture Behavioral of four_bit_adder is
    component half_adder is
           port (
              a_i : in STD_LOGIC;
              b_i : in STD_LOGIC;
              carry : out STD_LOGIC;
              sum : out STD_LOGIC
           );
        end component;
    component full_adder is
        port (
```

```
            a_i : in STD_LOGIC;
            b_i : in STD_LOGIC;
            ci : in STD_LOGIC;
            ciH : out STD_LOGIC;
            si : out STD_LOGIC
        );
    end component;
    signal c_in : STD_LOGIC := '0';
    signal temp : std_logic_vector (3 downto 0) := (others => '0');

begin
    full_adder1 : full_adder
        port map (
            a_i => a_i(0),
            b_i => b_i(4),
            ci => c_in,
            ciH => temp(0),
            si => output(0)
        );
     full_adder2 : full_adder
        port map (
            a_i => a_i(1),
            b_i => b_i(5),
            ci => temp(0),
            ciH => temp(1),
            si => output(1)
        );
      full_adder3 : full_adder
        port map (
            a_i => a_i(2),
            b_i => b_i(6),
            ci => temp(1),
            ciH => temp(2),
            si => output(2)
        );
      full_adder4 : full_adder
        port map (
            a_i => a_i(3),
            b_i => b_i(7),
            ci => temp(2),
            ciH => temp(3),
            si => output(3)
        );
    output(4) <= temp(3);
end Behavioral;
```

Full Adder:

```
library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_std.all;

use std.textio.all;


-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC_STD.ALL;


-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;


entity full_adder is
   Port ( a_i : in STD_LOGIC;
          b_i : in STD_LOGIC;
          ci : in STD_LOGIC;
          ciH : out STD_LOGIC;
          si : out STD_LOGIC
    );
end full_adder;

architecture Behavioral of full_adder is
   component half_adder is
      port (
         a_i : in STD_LOGIC;
```

```
        b_i : in STD_LOGIC;
        carry : out STD_LOGIC;
        sum : out STD_LOGIC
      );
  end component;
  signal x_1 : STD_LOGIC := '0';
  signal y_1 : STD_LOGIC := '0';
  signal f_0 : STD_LOGIC := '0';
begin
  half_adder1 : half_adder
    port map (
      a_i  => a_i,
      b_i => b_i,
      sum => x_1,
      carry  => y_1
    );

  half_adder2 : half_adder
    port map (
      a_i => x_1,
      b_i => ci,
      sum => si,
      carry => f_0
    );
  ciH <= f_0 or y_1;
end Behavioral;
```

Half Adder:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
```

```
entity half_adder is
   Port ( a_i : in STD_LOGIC;
          b_i : in STD_LOGIC;
          carry : out STD_LOGIC;
          sum : out STD_LOGIC);
end half_adder;

architecture Behavioral of half_adder is

begin
carry <= a_i and b_i;
sum <=  a_i xor b_i;

end Behavioral;
```

Subtraction:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity subtraction_full_adder is
  Port ( a_i: in std_logic_vector(3 downto 0);
         b_i: in std_logic_vector(7 downto 4);
         output: out std_logic_vector(4 downto 0)
        );
end subtraction_full_adder;

architecture Behavioral of subtraction_full_adder is
   component half_adder is
          port (
             a_i : in STD_LOGIC;
             b_i : in STD_LOGIC;
```

```
            carry : out STD_LOGIC;
            sum : out STD_LOGIC
        );
    end component;
  component full_adder is
    port (
      a_i : in STD_LOGIC;
      b_i : in STD_LOGIC;
      ci : in STD_LOGIC;
      ciH : out STD_LOGIC;
      si : out STD_LOGIC
    );
  end component;
  signal c_in : STD_LOGIC := '1';
  signal temp : std_logic_vector (3 downto 0) := (others => '0');
  signal complement_b : std_logic_vector (3 downto 0) := (others => '0');
begin
  complement_b <= not(b_i);
  full_adder1 : full_adder
    port map (
      a_i => a_i(0),
      b_i => complement_b(0),
      ci => c_in,
      ciH => temp(0),
      si => output(0)
    );
   full_adder2 : full_adder
    port map (
      a_i => a_i(1),
      b_i => complement_b(1),
      ci => temp(0),
      ciH => temp(1),
      si => output(1)
    );
   full_adder3 : full_adder
    port map (
      a_i => a_i(2),
      b_i => complement_b(2),
      ci => temp(1),
      ciH => temp(2),
      si => output(2)
```

```
    );
   full_adder4 : full_adder
    port map (
      a_i => a_i(3),
      b_i => complement_b(3),
      ci => temp(2),
      ciH => temp(3),
      si => output(3)
    );
  output(4) <= '0';
end Behavioral;
```

Rotate:

```
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity rotate is
  Port ( a_i : in STD_LOGIC_VECTOR(3 downto 0);
       output : out STD_LOGIC_VECTOR (4 downto 0));
end rotate;

architecture Behavioral of rotate is
  signal rotate: std_logic_vector(3 downto 0);
begin
  rotate(1) <= a_i(0);
  rotate(2) <= a_i(1);
  rotate(3) <= a_i(2);
  rotate(0) <= a_i(3);

  output(0) <= rotate(0);
  output(1) <= rotate(1);
  output(2) <= rotate(2);
  output(3) <= rotate(3);
```

```
   output(4) <= '0';

end Behavioral;
```

Logical Left Shift:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity logical_left_shift is
   Port ( a_i : in STD_LOGIC_VECTOR(3 downto 0);
          output : out STD_LOGIC_VECTOR (4 downto 0));
end logical_left_shift;

architecture Behavioral of logical_left_shift is
   signal shifting: std_logic_vector(3 downto 0);
begin
   shifting(1) <= a_i(0);
   shifting(2) <= a_i(1);
   shifting(3) <= a_i(2);
   shifting(0) <= '0';

   output(0) <= shifting(0);
   output(1) <= shifting(1);
   output(2) <= shifting(2);
   output(3) <= shifting(3);
   output(4) <= '0';

end Behavioral;
```

Ones Complement:

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

```
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ones_complement is
   Port ( a_i : in STD_LOGIC_VECTOR (3 downto 0);
          output: out STD_LOGIC_VECTOR (4 downto 0)
         );
end ones_complement;

architecture Behavioral of ones_complement is
   signal complement: STD_LOGIC_VECTOR (3 downto 0);
begin
   complement(0) <= not a_i(0);
   complement(1) <= not a_i(1);
   complement(2) <= not a_i(2);
   complement(3) <= not a_i(3);

   output(0) <= complement(0);
   output(1) <= complement(1);
   output(2) <= complement(2);
   output(3) <= complement(3);
   output(4) <= '0';
end Behavioral;
```

Bitwise XOR:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
```

```
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity bitwise_XOR is
   Port ( a_i : in STD_LOGIC_VECTOR (3 downto 0);
          b_i : in STD_LOGIC_VECTOR (7 downto 4);
          output : out STD_LOGIC_VECTOR (4 downto 0));
end bitwise_XOR;

architecture Behavioral of bitwise_XOR is
   signal operation: STD_LOGIC_VECTOR (3 downto 0);
begin
   operation(0) <= a_i(0) xor b_i(4);
   operation(1) <= a_i(1) xor b_i(5);
   operation(2) <= a_i(2) xor b_i(6);
   operation(3) <= a_i(3) xor b_i(7);

   output(0) <= operation(0);
   output(1) <= operation(1);
   output(2) <= operation(2);
   output(3) <= operation(3);
   output(4) <= '0';
end Behavioral;
```

Bitwise OR:

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity bitwise_OR is
Port ( a_i : in STD_LOGIC_VECTOR (3 downto 0);
```

```
b_i : in STD_LOGIC_VECTOR (7 downto 4);
output : out STD_LOGIC_VECTOR (4 downto 0));
end bitwise_OR;

architecture Behavioral of bitwise_OR is
signal operation: STD_LOGIC_VECTOR (3 downto 0);
begin
operation(0) <= a_i(0) or b_i(4);
operation(1) <= a_i(1) or b_i(5);
operation(2) <= a_i(2) or b_i(6);
operation(3) <= a_i(3) or b_i(7);

output(0) <= operation(0);
output(1) <= operation(1);
output(2) <= operation(2);
output(3) <= operation(3);
output(4) <= '0';

end Behavioral;
```

Bitwise AND;

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity bitwise_AND is
Port ( a_i : in STD_LOGIC_VECTOR (3 downto 0);
b_i : in STD_LOGIC_VECTOR (7 downto 4);
```

```
output : out STD_LOGIC_VECTOR (4 downto 0));
end bitwise_AND;

architecture Behavioral of bitwise_AND is
signal operation: STD_LOGIC_VECTOR (3 downto 0);
begin
operation(0) <= a_i(0) and b_i(4);
operation(1) <= a_i(1) and b_i(5);
operation(2) <= a_i(2) and b_i(6);
operation(3) <= a_i(3) and b_i(7);

output(0) <= operation(0);
output(1) <= operation(1);
output(2) <= operation(2);
output(3) <= operation(3);
output(4) <= '0';

end Behavioral;
```