EEE 102

LAB 4

Student Name: Robin Umut Kızıl

Student ID: 22003260

Class code and section: EE-102-01

Purpose of Lab:

In this lab, it is aimed to become familiar with the 6-bit ALU structure that can perform 8 different functions (addition, subtraction, logical shift left, arithmetic shift right, logical shift right, comparator (with equal, less than, and greater than outputs), rotate right, and bitwise xor) and to create this structure with vhdl codes on basys 3.

Design Specification:

In vhdl code, ALU structure will be assigned as top module, remaining 8 functions will be assigned as submodules. COMPONENT and PORT MAP structures will be used to connect submodules and top modules. In addition, any operators from the libraries will not be used, they will be created in the code.

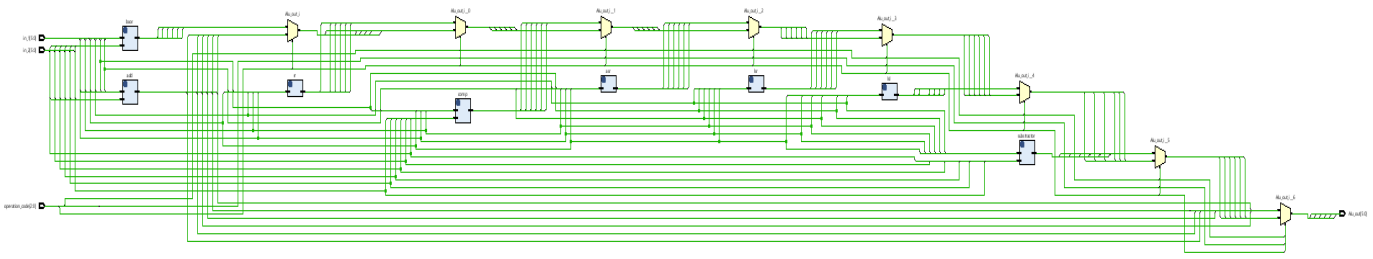


Figure 1: RTL Schematic

Methodology:

First, the ALU top module will be created, then 2 6-bit numbers and the 3-bit number that decides the operation as input, and the 6-bit output number are defined to the ports. 8 out signals are created to match the output of the 8 functions. These signals are linked to the results of the operation_code states with the if else structure. In this way, the signals that will be connected to

the outputs of the functions with the PORT MAP structure in the future have been controlled. Then the functions were created and became submodules of the ALU with the COMPONENT and PORT MAP structures.

Below is a detailed description of how these functions are created.

1. **Addition;**

   To create addition, it is necessary to collect each digit and set the carry s they will give to the next digit. For this, a separate one_bit_adder source code is written, which will be a component to addition. This code has 3 inputs as in1, in2, cin and 2 outputs as sum, cout. Then, this one_bit_adder added to addition as a component and the 6 carry signal created are performed with a 6-bit addition. (The value of carry_in0 is chosen as 0, since there is no carry to enter the first digit.)

2. **Subtraction;**

   For subtraction, 6 separate not-y signals are assigned as priority, these are inverted versions of each digit of the 2nd number. Thus, one compliment of the 2nd number is obtained. Then these newly created signals and 1 number are collected as in the addition function, while carry_in0 = 1. Thus, the 1st number and the 2 compliments of the 2nd number are added together, which means that the 1 and the second number are subtracted.

3. **logical shift left;**

   For logical shift left, all input digits are linked to the output digit one to the left of it. 0 is written in the blank digit on the right.

4. **arithmetic shift right;**

   For arithmetic shift right, all input digits are linked to the output digit to the right of it. The number of bits previously there is written in the blank digit on the right.

5. **logical shift right;**

   For logical shift right, all input digits are linked to the output digit one to the right of it. 0 is written in the blank digit on the left.

6. **comparator;**

For comparison, the previously written Subtraction component is added as a subcomponent here, and then the output is examined by subtracting the 2 input numbers from each other. if the output is '000000' it means that the two numbers are the same. if the MSB of the output is 1 it means that the subtracted number is larger. of the remaining cases, the first number will be greater than the second. this is associated using the if else method.

7. **rotate right;**

For logical shift right, all input digits are linked to the output digit one to the right of it. Right last digit which is disappear is written in the blank digit on the left.

8. **bitwise xor;**

The digits of 2 numbers entered as input are compared with each other using the xor relation. If those two digits are the same, the corresponding digit in the output will be 0, if those two digits are different, the corresponding digit in the output will be 1.

After all functions were assigned as components, the constraint file was created, and then synthesis, implementation, testbench, bitstream and hardware manager processes were checked and code tested on basys 3.

correspondences expressed by the operation control code;

Operational code	corresponding function
“000”	**Addition**
“001”	**Subtraction**
“010”	**logical shift left**
“011”	**arithmetic shift right**
“100”	**logical shift right**
“101”	**comparator**
“110”	**rotate right**
“111”	**bitwise xor**

Table 1: Operational code

Conclusion:

In conclusion, code worked as expected on basys3, functions returned as they should. In this way, it was learned by being familiar with the 6-bit ALU structure and its application on basys3 was seen. That's why this lab is successful.

Appendices A:

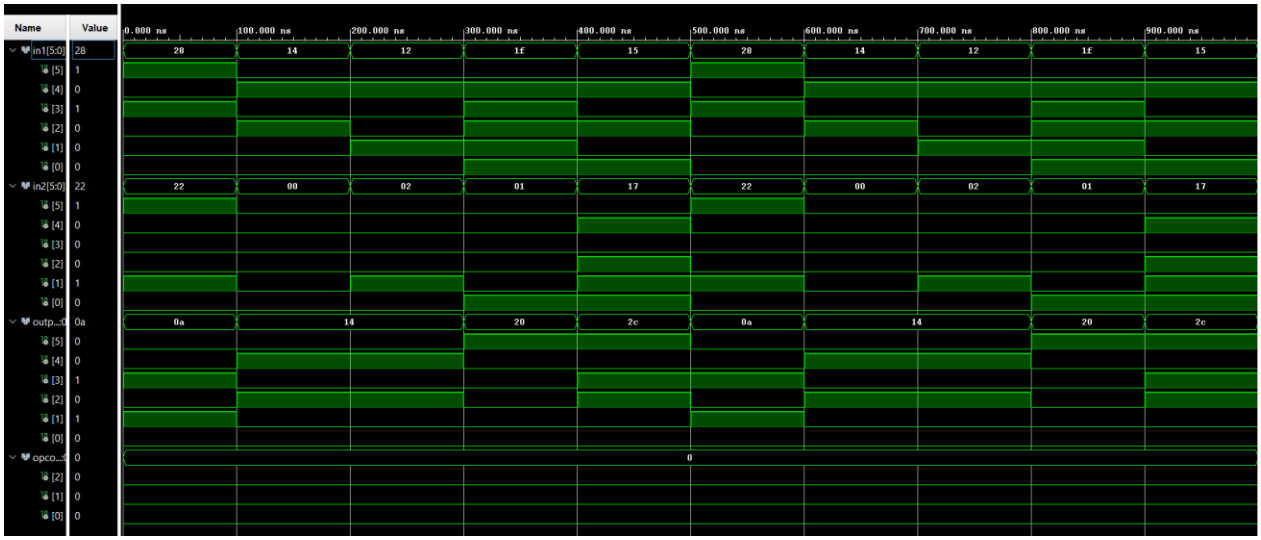
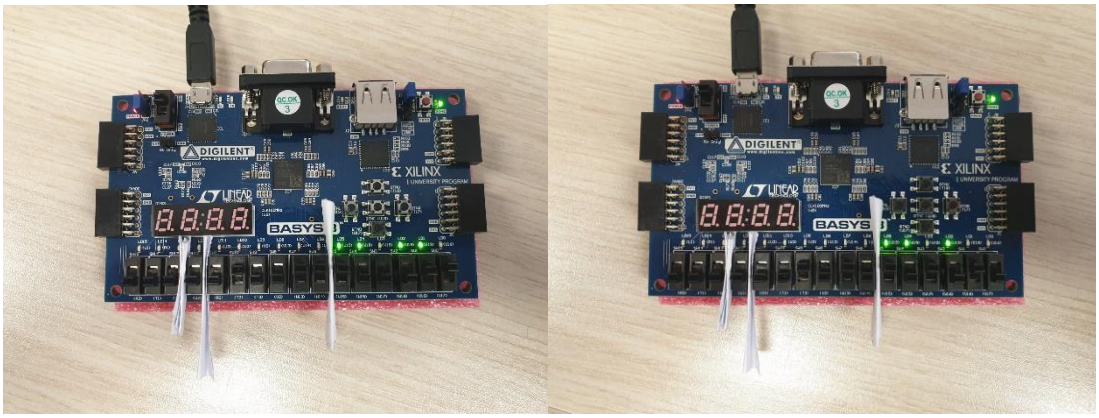
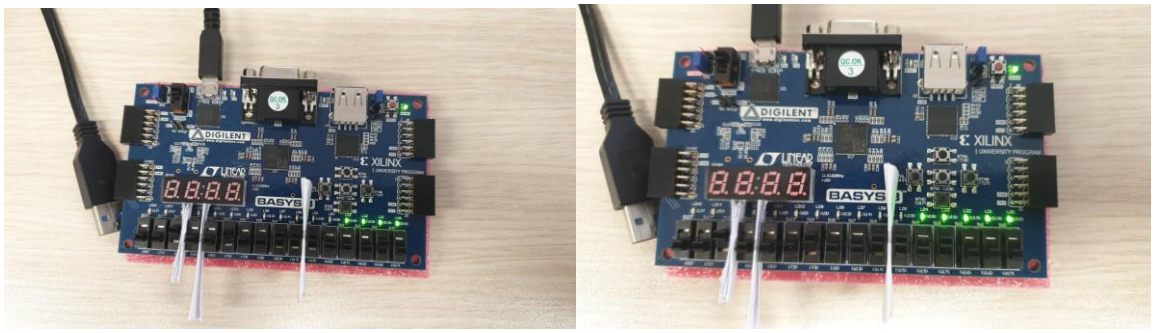


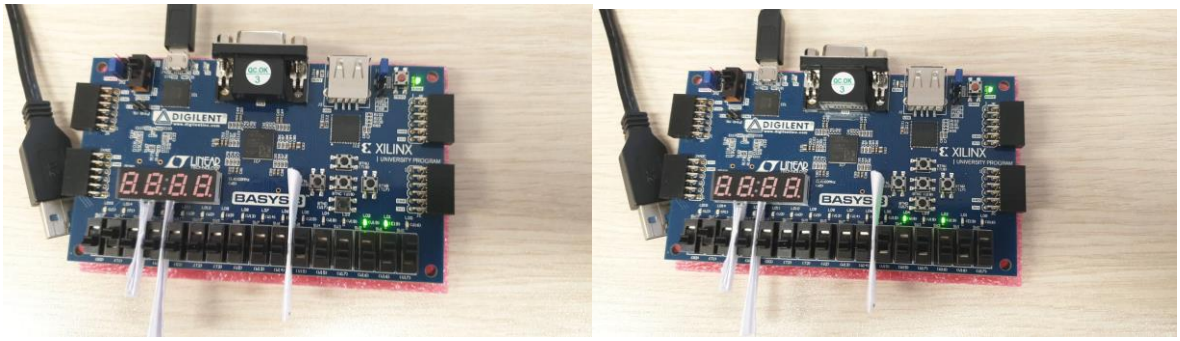
Figure 2: Testbench example of Addition



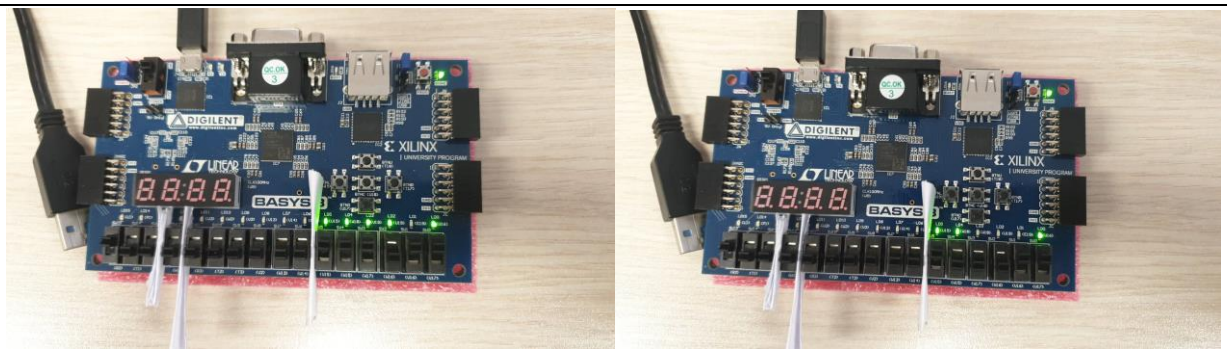
Figures 3 and 4: examples for addition



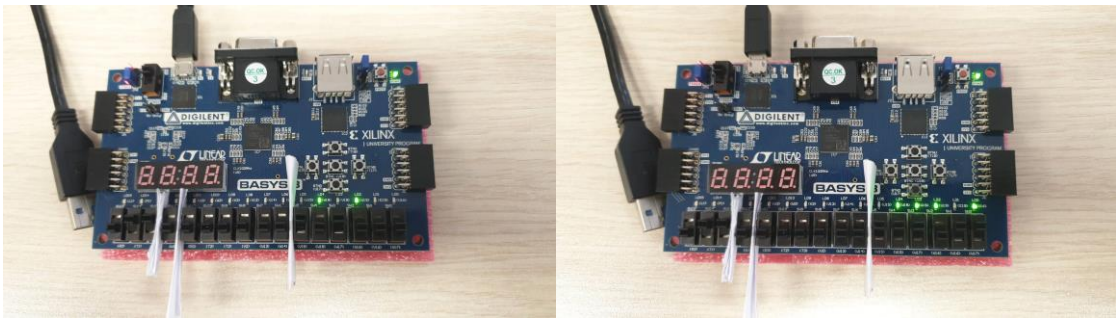
Figures 5 and 6: examples for subtraction



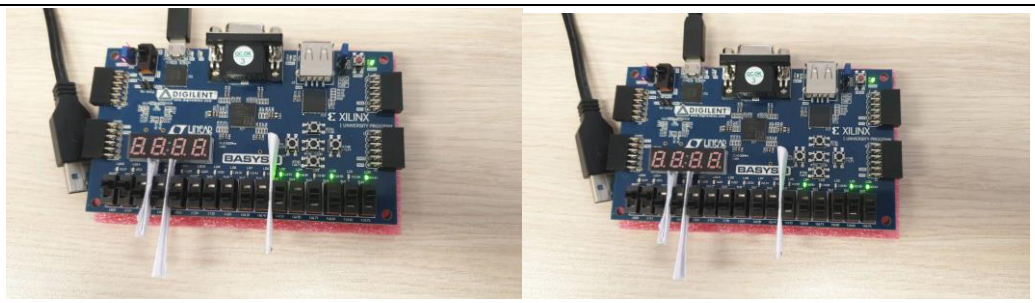
Figures 7 and 8: examples for logical shift left



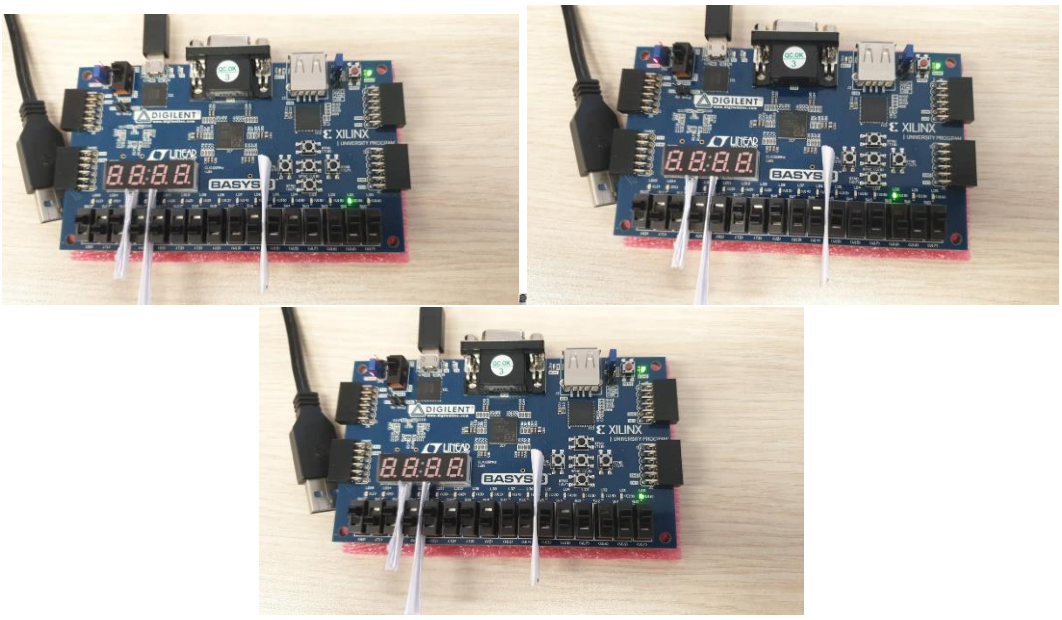
Figures 9 and 10: examples for arithmetic shift right



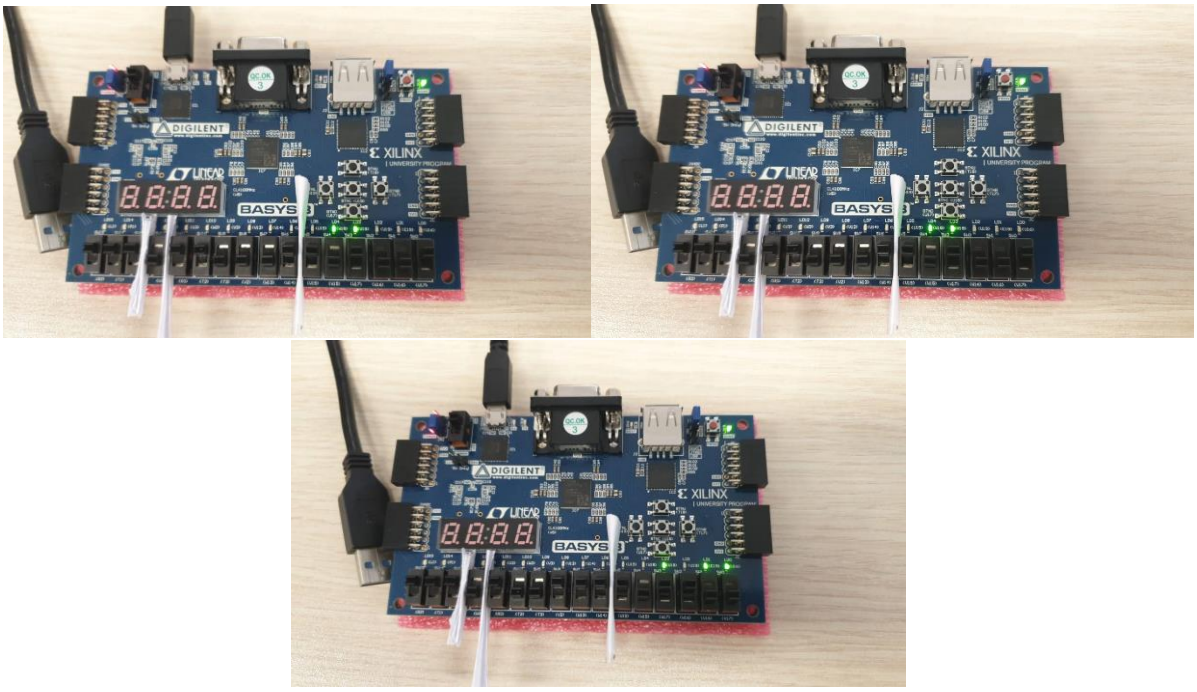
Figures 11 and 12: examples for logical shift right



Figures 13 and 14: examples for rotate right



Figures 15,16 and 17: examples for comparator



Figures 18, 19 and 20: examples for bitwise xor

Appendices B:

ALU Code:

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



entity ALU is
      Port (operation_code: in std_logic_vector(2 downto 0);
          in_1,in_2: in std_logic_vector(5 downto 0);
          Alu_out: out std_logic_vector(5 downto 0)
   );
end ALU;

architecture Behavioral of ALU is

component addition
   port(x,y  : in std_logic_vector(5 downto 0);
       outz : out std_logic_vector(5 downto 0)
   );
end component;

component subtraction
   port(x,y  : in std_logic_vector(5 downto 0);
       outz : out std_logic_vector(5 downto 0)
   );
end component;

component logical_shift_left
   port(x  : in std_logic_vector(5 downto 0);
       y : out std_logic_vector(5 downto 0)
```

```
   );
end component;

component logical_shift_right
   port(x  : in std_logic_vector(5 downto 0);
        y : out std_logic_vector(5 downto 0)
   );
end component;

component arithmetic_shift_right
   port(x  : in std_logic_vector(5 downto 0);
        y : out std_logic_vector(5 downto 0)
   );
end component;

component comparator
   port(x,y  : in std_logic_vector(5 downto 0);
        outz : out std_logic_vector(5 downto 0)
   );
end component;

component rotatee_right
   port(x  : in std_logic_vector(5 downto 0);
        y : out std_logic_vector(5 downto 0)
   );
end component;

component bitwise_xor
   port(x,y  : in std_logic_vector(5 downto 0);
        outz : out std_logic_vector(5 downto 0)
   );
```

```
end component;

signal out1, out2, out3, out4, out5, out6, out7, out8 : std_logic_vector(5 downto 0);

begin

add        : addition port map(x => in_1,y => in_2,outz => out1);
substractor :  subtraction port map(x => in_1, y => in_2, outz => out2);
lsl        : logical_shift_left port map(x => in_1,y => out3);
lsr        : logical_shift_right port map(x => in_1,y => out4);
asr        : arithmetic_shift_right port map(x => in_1,y => out5);
comp       : comparator port map(x => in_1, y => in_2, outz => out6);
rr         : rotatee_right port map(x => in_1,y => out7);
bxor       : bitwise_xor port map(x => in_1,y => in_2,outz => out8);



process(operation_code,in_1,in_2,out1, out2, out3, out4, out5, out6, out7, out8)


   begin
   if operation_code    = "000" then
   Alu_out <= out1;
   elsif operation_code = "001" then
   Alu_out <= out2;
   elsif operation_code = "010" then
   Alu_out <= out3;
   elsif operation_code = "011" then
   Alu_out <= out4;
   elsif operation_code = "100" then
```

```
    Alu_out <= out5;
    elsif operation_code = "101" then
    Alu_out <= out6;
    elsif operation_code = "110" then
    Alu_out <= out7;
    elsif operation_code = "111" then
    Alu_out <= out8;
    else
    Alu_out <= out1;
    end if;
    end process;
end Behavioral;
```

Addition Code:

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

entity addition is
    Port ( x : in STD_LOGIC_VECTOR (5 downto 0);
           y : in STD_LOGIC_VECTOR (5 downto 0);
           outz : out STD_LOGIC_VECTOR (5 downto 0));
end addition;

architecture Behavioral of addition is

component one_bit_adder
    port(x,y,c_in : in std_logic;
         sum,c_out    : out std_logic
    );
```

```
end component;

signal c0,c1,c2,c3,c4,c5,c6,trash : std_logic;



begin

c0 <= '0' ;

st1 : one_bit_adder port map(
   x => x(0),y => y(0),c_in => c0,sum => outz(0),c_out => c1
);
st2 : one_bit_adder port map(
   x => x(1),y => y(1),c_in => c1,sum => outz(1),c_out => c2
);
st3 : one_bit_adder port map(
   x => x(2),y => y(2),c_in => c2,sum => outz(2),c_out => c3
);
st4 : one_bit_adder port map(
   x => x(3),y => y(3),c_in => c3,sum => outz(3),c_out => c4
);
st5 : one_bit_adder port map(
   x => x(4),y => y(4),c_in => c4,sum => outz(4),c_out => c5
);
st6 : one_bit_adder port map(
   x => x(5),y => y(5),c_in => c5,sum => outz(5),c_out => trash
);



end Behavioral;
```

One Bit Adder Code:

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


entity one_bit_adder is
   Port ( x     : in STD_LOGIC;
          y     : in STD_LOGIC;
          c_in  : in STD_LOGIC;
          sum   : out STD_LOGIC;
          c_out : out STD_LOGIC);
end one_bit_adder;

architecture Behavioral of one_bit_adder is

begin

sum   <= x xor y xor c_in;
c_out <= (x and y) or (y and c_in) or (x and c_in);

end Behavioral;
```

Subtraction Code:

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


entity subtraction is
   Port ( x : in STD_LOGIC_VECTOR (5 downto 0);
```

```
        y : in STD_LOGIC_VECTOR (5 downto 0);
        outz : out STD_LOGIC_VECTOR (5 downto 0));
end subtraction;

architecture Behavioral of subtraction is

component one_bit_adder
   port(x,y,c_in : in std_logic;
        sum,c_out    : out std_logic
   );
end component;

component inverter
   port(x : in std_logic;
        y : out std_logic
        );
end component;

signal c0,c1,c2,c3,c4,c5,c6,trash : std_logic;
signal noty0,noty1,noty2,noty3,noty4,noty5 : std_logic;

begin
c0 <= '1';

noty0 <= not y(0);
noty1 <= not y(1);
noty2 <= not y(2);
noty3 <= not y(3);
noty4 <= not y(4);
noty5 <= not y(5);
```

```
st1 : one_bit_adder port map(
   x => x(0),y => noty0,c_in => c0,sum => outz(0),c_out => c1
);
st2 : one_bit_adder port map(
   x => x(1),y => noty1,c_in => c1,sum => outz(1),c_out => c2
);
st3 : one_bit_adder port map(
   x => x(2),y => noty2,c_in => c2,sum => outz(2),c_out => c3
);
st4 : one_bit_adder port map(
   x => x(3),y => noty3,c_in => c3,sum => outz(3),c_out => c4
);
st5 : one_bit_adder port map(
   x => x(4),y => noty4,c_in => c4,sum => outz(4),c_out => c5
);
st6 : one_bit_adder port map(
   x => x(5),y => noty5,c_in => c5,sum => outz(5),c_out => trash
);

end Behavioral;
```

Logical Shift Left Code:

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


entity logical_shift_left is
   Port ( x : in STD_LOGIC_VECTOR (5 downto 0);
          y : out STD_LOGIC_VECTOR (5 downto 0)
```

```
        );
end logical_shift_left;

architecture Behavioral of logical_shift_left is



begin

y(5) <= x(4);
y(4) <= x(3);
y(3) <= x(2);
y(2) <= x(1);
y(1) <= x(0);
y(0) <= '0';



end Behavioral;
```

Logical Shift Right Code:

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



entity logical_shift_right is
    Port ( x : in STD_LOGIC_VECTOR (5 downto 0);
           y : out STD_LOGIC_VECTOR (5 downto 0)
         );
end logical_shift_right;

architecture Behavioral of logical_shift_right is
```

```
begin
y(5) <= '0';
y(4) <= x(5);
y(3) <=x(4);
y(2) <=x(3);
y(1) <=x(2);
y(0) <=x(1);


end Behavioral;
```

Arithmetic Shift Right Code:

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


entity arithmetic_shift_right is
   Port ( x : in STD_LOGIC_VECTOR (5 downto 0);
          y : out STD_LOGIC_VECTOR (5 downto 0)
         );
end arithmetic_shift_right;

architecture Behavioral of arithmetic_shift_right is


begin
y(5) <= x(5);
y(4) <= x(5);
y(3) <= x(4);
```

```
y(2) <= x(3);
y(1) <= x(2);
y(0) <= x(1);

end Behavioral;
```

Comparator Code:

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



entity comparator is
    Port ( x : in STD_LOGIC_VECTOR (5 downto 0);
           y : in STD_LOGIC_VECTOR (5 downto 0);
           outz : out STD_LOGIC_VECTOR (5 downto 0));
end comparator;

architecture Behavioral of comparator is

component subtraction
    port(x,y  : in std_logic_vector(5 downto 0);
         outz : out std_logic_vector(5 downto 0)
    );
end component;



signal comp : STD_LOGIC_VECTOR(5 downto 0);

begin
```

```
substract :  subtraction port map(x => x, y => y, outz => comp);

process(x,y,comp)

   begin
   if comp = "000000" then
   outz <= "000001";

   elsif comp(5) = '1' then
   outz <= "000010";

   else
   outz <= "000100";

   end if;

end process;

end Behavioral;
```

Rotate Right Code:

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

entity rotatee_right is
   Port ( x : in STD_LOGIC_VECTOR (5 downto 0);
          y : out STD_LOGIC_VECTOR (5 downto 0));
end rotatee_right;

architecture Behavioral of rotatee_right is
```

```
begin

y(5) <= x(0);
y(4) <= x(5);
y(3) <= x(4);
y(2) <= x(3);
y(1) <= x(2);
y(0) <= x(1);

end Behavioral;
```

Bitwise xor:

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

entity bitwise_xor is
    Port ( x : in STD_LOGIC_VECTOR (5 downto 0);
           y : in STD_LOGIC_VECTOR (5 downto 0);
           outz : out STD_LOGIC_VECTOR (5 downto 0));
end bitwise_xor;

architecture Behavioral of bitwise_xor is

begin
```

```
outz(0) <= x(0) xor y(0);
outz(1) <= x(1) xor y(1);
outz(2) <= x(2) xor y(2);
outz(3) <= x(3) xor y(3);
outz(4) <= x(4) xor y(4);
outz(5) <= x(5) xor y(5);


end Behavioral;
```

Testbench Code:

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date: 23.04.2023 21:09:50
-- Design Name:
-- Module Name: testbrench - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
```

```
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity testbench is

end testbench;

architecture Behavioral of testbench is

component ALU
   Port (operation_code: in std_logic_vector(2 downto 0);
         in_1,in_2: in std_logic_vector(5 downto 0);
         Alu_out: out std_logic_vector(5 downto 0)
   );
end component;
signal in1, in2,output : std_logic_vector(5 downto 0);
signal opcode : std_logic_vector(2 downto 0 );
begin

testb : ALU port map(operation_code => opcode, in_1 => in1, in_2 => in2, Alu_out => output);

testBench_main  : process
begin

   opcode <= "001";
   in1 <= "101110";
   in2 <= "111010";
```

```
    wait for 100ns;

    opcode <= "001";
    in1 <= "011100";
    in2 <= "011001";
    wait for 100ns;

    opcode <= "001";
    in1 <= "011110";
    in2 <= "010010";
    wait for 100ns;

    opcode <= "001";
    in1 <= "011111";
    in2 <= "011001";
    wait for 100ns;

    opcode <= "001";
    in1 <= "011101";
    in2 <= "011111";
    wait for 100ns;

end process;
end Behavioral;
```