Alp Tekin

22403597

EE102-02

22.10.2025

Bilkent University

Department of Electrical and Electronics Engineering

Lab 4 Report: Arithmetic Logic Unit

Purpose

The aim of the experiment is to design an arithmetic logic unit (ALU) which performs eight functions, including addition, subtraction, at least one bitwise and one shift operation. The purpose is to understand how ALU functions works and gain hands on experience by simulating its behavior and implementing the design on the Basys3 board.

Design Specifications

In this ALU design, I used 11-bit input and 5-bit output. The operation is selected by using the 3-bit input Opcode[2:0]. This 3-bit Opcode input determine which operation is selected as can be seen in Figure 1. The remaining 8 input bits correspond to two 4-bit inputs provided by the user: A_In[3:0] and B_In[3:0]. The main 4-bit output, Result_Out[3:0], represents the result of the selected operation and is displayed by the LEDs on the Basys3 board. The fifth output bit, bit5, only taken into account when the selected operation is addition. For addition, it indicates the carry-out when the sum exceeds 4 bits, otherwise it remains ‘0’.

This Arithmetic Logic Unit (ALU) is designed in a modular structure, consisting eight separate modules. Each module is responsible for a different operation. All submodules are under the top-level module names ALU_Top, which connects the inputs and outputs of each functional block.

The top module selects one of these operations according to given Opcode[2:0] value and produces the corresponding 5-bit result through Result_Out[3:0] and 5bit output. The top-module and eight sub-modules are:

- ALU_Top.vhd
    - adder.vhd
    - subtracter.vhd
    - xor_4bit.vhd
    - bitwise_comparator.vhd
    - complement.vhd
    - rotate_right.vhd
    - parity_checker.vhd
    - logical_shift.vhd

Also in this design, I create two additional submodules which used to implement addition and subtraction for the ALU. The half_adder module performs single-bit addition by generating a sum and a carry output. The full_adder module extends this operation by including a carry-in input. The names of these submodules are listed below.

- full_adder.vhd
- half_adder.vhd

Figure 1 summarizes all operations that the ALU can perform. The table lists the input combinations, the corresponding output expressions and an example for each operation. Here the A3-A0 represent the bits of A_In[3:0] and B3-B0 represent the bits of B_In[3:0]. Every operation except for subtraction uses unsigned inputs and produces unsigned outputs. The subtraction operation takes two unsigned inputs but its result can represent negative value when the first number is smaller than the second. Thus, the subtraction has a 4-bit signed number output. Furthermore, only the addition operation produces a 5-bit result. In all other operations, this bit remains “0” and is not shown in the output in Figure 1.

Selection (Opcode)	**Operation**	**Input(s)**	**Outputs (MSB to LSB)**	**Example**
“000”	Addition	A_In[3:0] B_In[3:0]	A_In + B_In	“1101” + “1011” = “11000”
“001”	Subtraction	A_In[3:0] B_In[3:0]	A_In – B_In	“1000” - “0011” = “0101”
“010”	Rotate Right	A_In[3:0]	A0 A3 A2 A1	“1100” => “0110”
“011”	Complement	A_In[3:0]	NOT A_In	“1101” => “0010”
“100”	Bitwise XOR	A_In[3:0] B_In[3:0]	A3 xor B3 A2 xor B2 A1 xor B1 A0 xor B0	“1100” XOR “0110” = “1010”
“101”	Bitwise XNOR	A_In[3:0] B_In[3:0]	A3 xnor B3 A2 xnor B2 A1 xnor B1 A0 xnor B0	“1101” XNOR “1011” = “1001”
“110”	Logical Shift	A_In[3:0]	“0” & A_In(3 downto 1)	“1100” => “0110”
“111”	Parity Check	A_In[3:0]	“000” (A3 xor A2 xor A1 xor A0)	“1010” => “0001”

Figure 1: Opcode and Operation Behaviors

The ALU then implemented on the Basys3 board using constraint file. The switches and LEDs on the board corresponding to the inputs and outputs are shown here:

A_In(0): V17 switch

A_In(1): V16 switch

A_In(2): W16 switch

A_In(3): W17 switch

B_In(0): W15 switch

B_In(1): V15 switch

B_In(2): W14 switch

B_In(3): W13 switch

Opcode(0): U1 switch

Opcode(1): T1 switch

Opcode(2): R2 switch

Result_Out(0): U16 LED

Result_Out(1): E19 LED

Result_Out(2): U19 LED

Result_Out(3): V19 LED

bit5 (Carry Out – Addition): W18 LED

Methodology

In this experiment, I designed the ALU step by step using VHDL in Vivado. First, I determined the eight arithmetic logic operations such as addition, subtraction, bitwise XOR and logical shift. For addition and subtraction, I also created two submodules: full adder and half adder. I designed these operations as separate modules to create a modular structure. This modular design improves readability and makes debugging easier. After writing each submodule, I created a top-module which consists of all these operations and connects their inputs and outputs. After writing all these modules, I generated the RTL schematics for both the top module and submodules. The RTL schematics can visualize whether the pins are connected properly or not. Before implementing the design on the Basys3 board, I used a testbench to verify whether my design functioned correctly.

To create the testbench code, I wrote several random input combinations for each operation and checked the results accordingly. After that, I created the constraint file which connects my VHDL design to Basys3 board. By generating the bitstream, I implemented the ALU design to Basys3 board. Finally, I tested the circuit using the switches and LEDs to verify the implementation was made successfully.

Results

After completing the design, I generated the RTL schematics in Vivado to observe the internal structure of the ALU. As shown in Figure 2, the schematic of the top module shows the hierarchy between the submodules and verifies that all connections made properly. The blue cells represent the sub-modules and the operation is selected by the multiplexer on the right according to the Opcode[2:0] value. In addition, there is another multiplexer at the bottom right that determines whether the fifth bit (bit5) is active, depending on the selected operation. Note that bit5 is used as the carry-out bit only for the addition operation. The cells from top to bottom represent each submodule of the ALU, including complement, parity checker, rotate right, logical shift, subtraction, bitwise XNOR, bitwise XOR and addition.

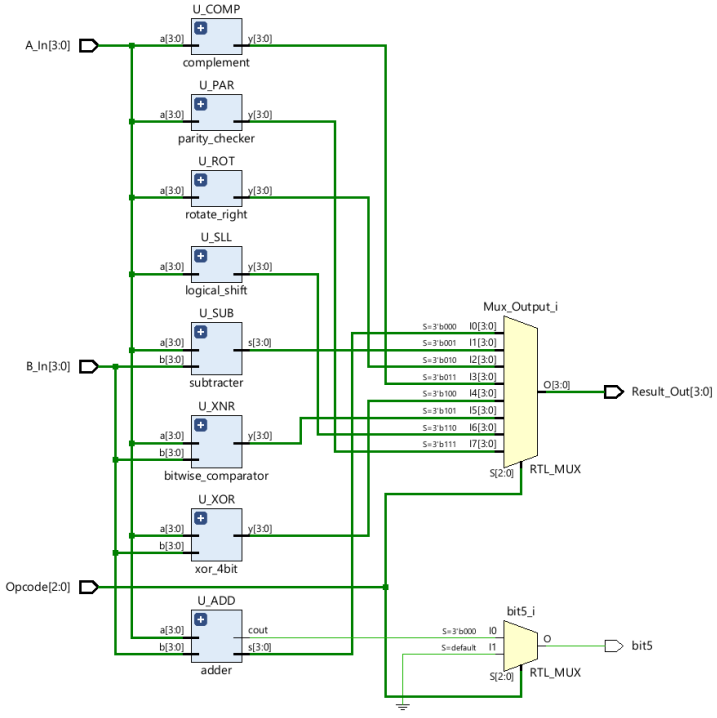

Figure 2: RTL schematic for ALU_Top

Figure 3-12 show the RTL schematics for each submodule below.

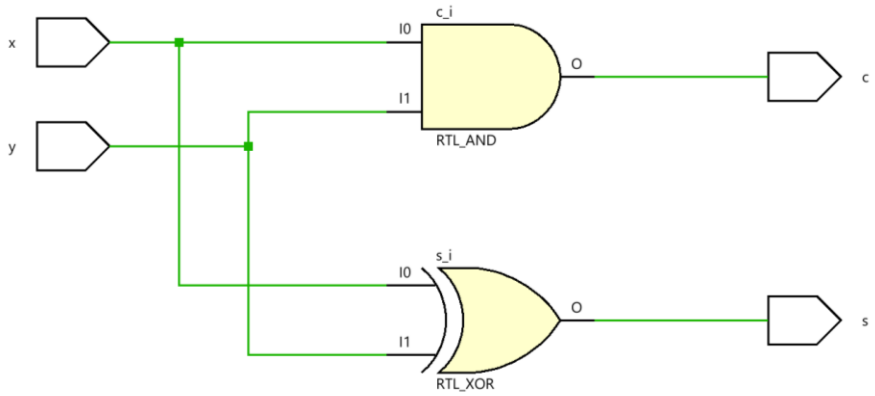

Figure 3: RTL schematic for half_adder

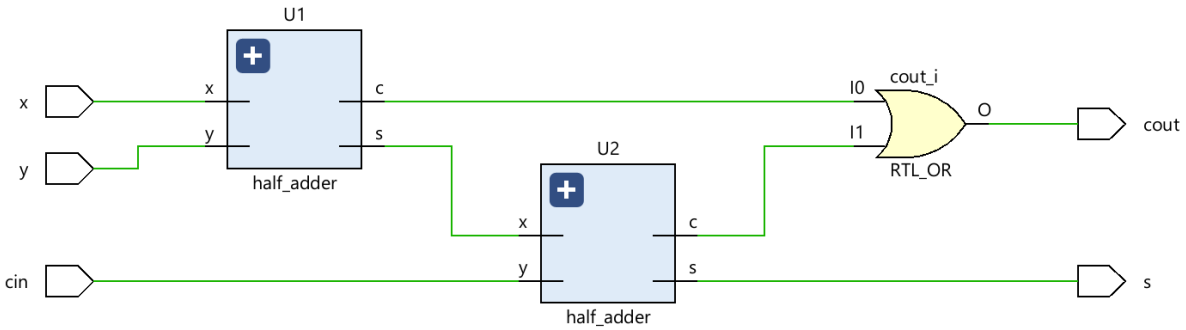

Figure 4: RTL schematic for full_adder

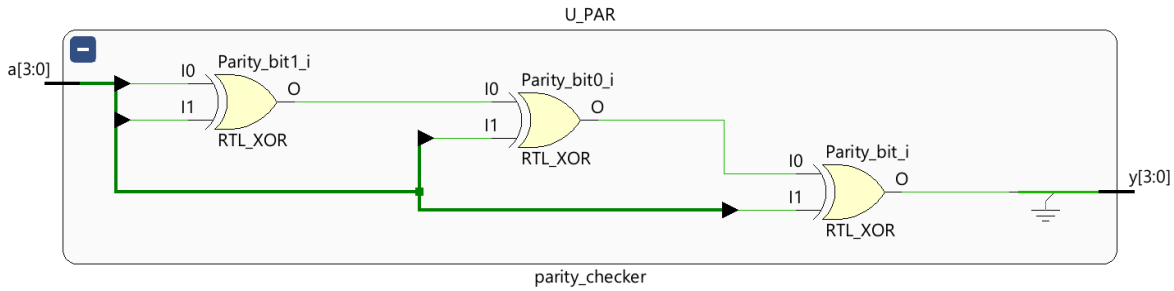

Figure 5: RTL schematic for parity_checker

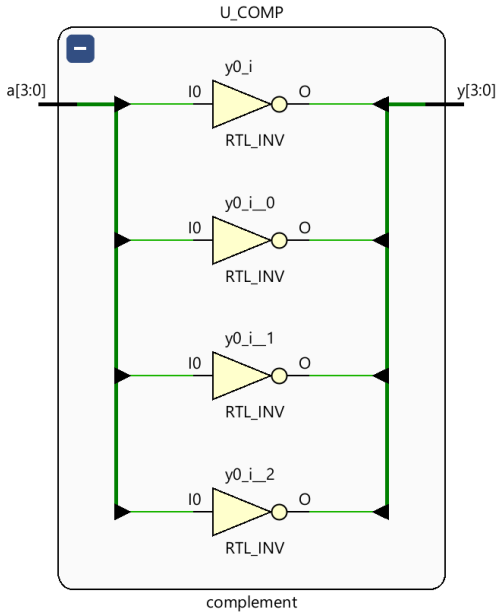

Figure 6: RTL schematic for complement

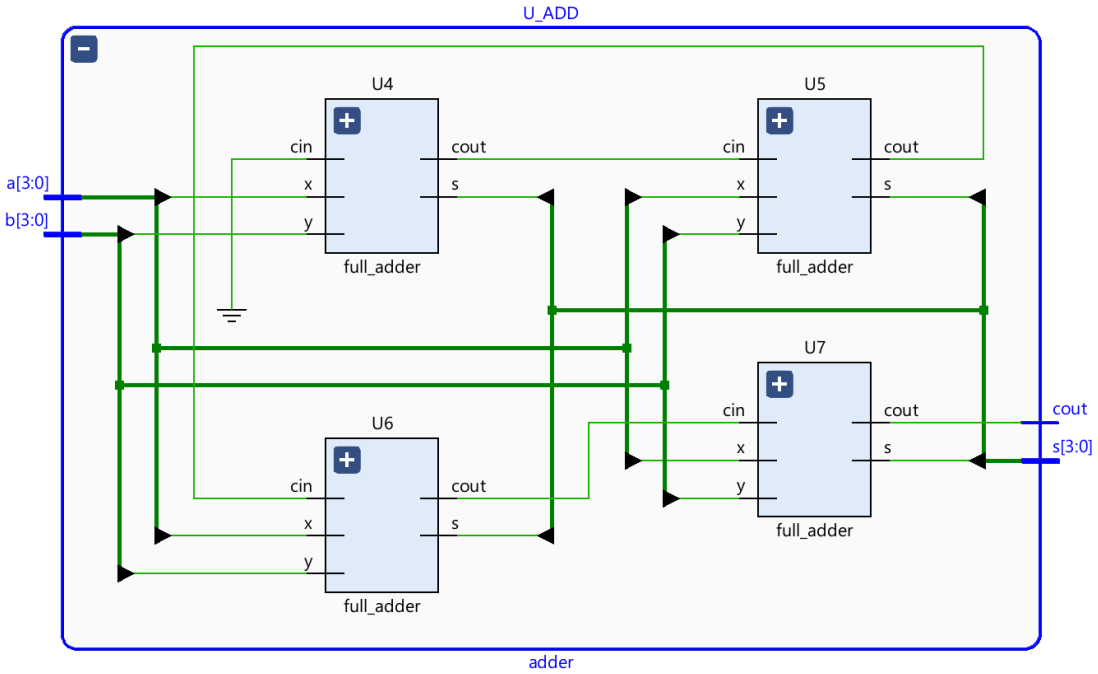

Figure 7: RTL schematic for adder

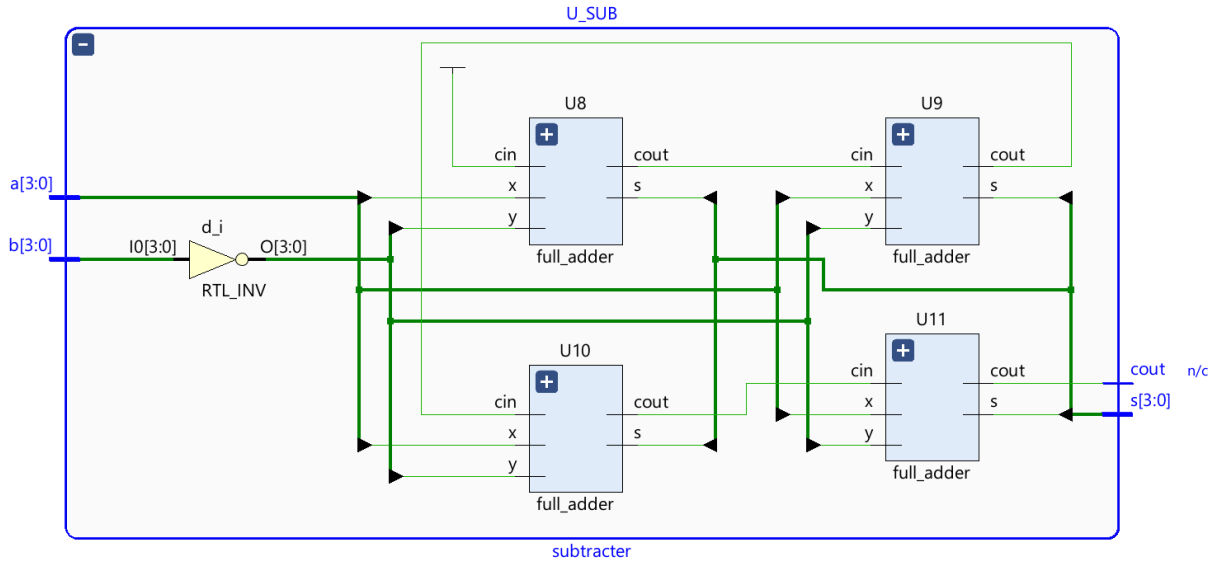

Figure 8: RTL schematic for subtracter

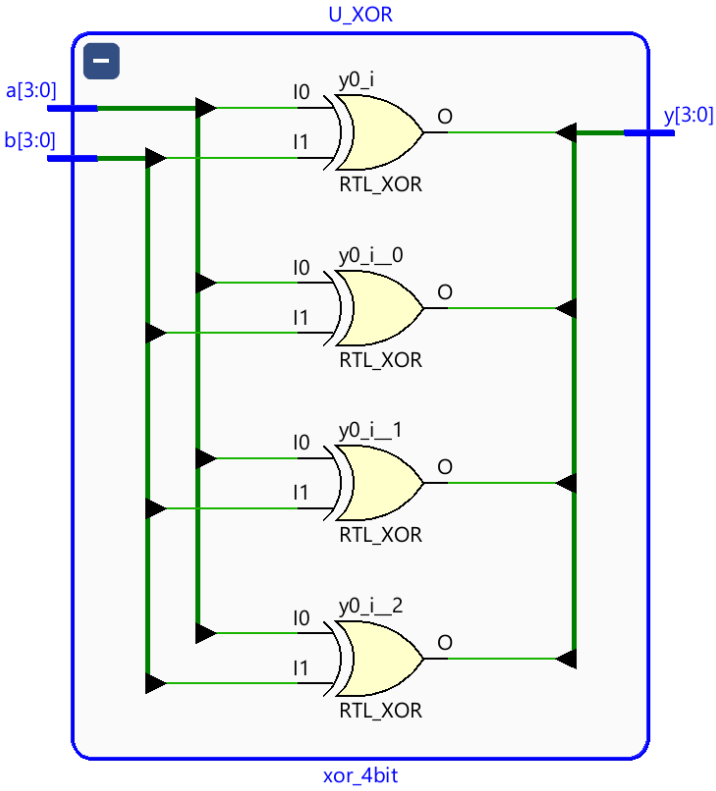

Figure 9: RTL schematic for xor_4bit

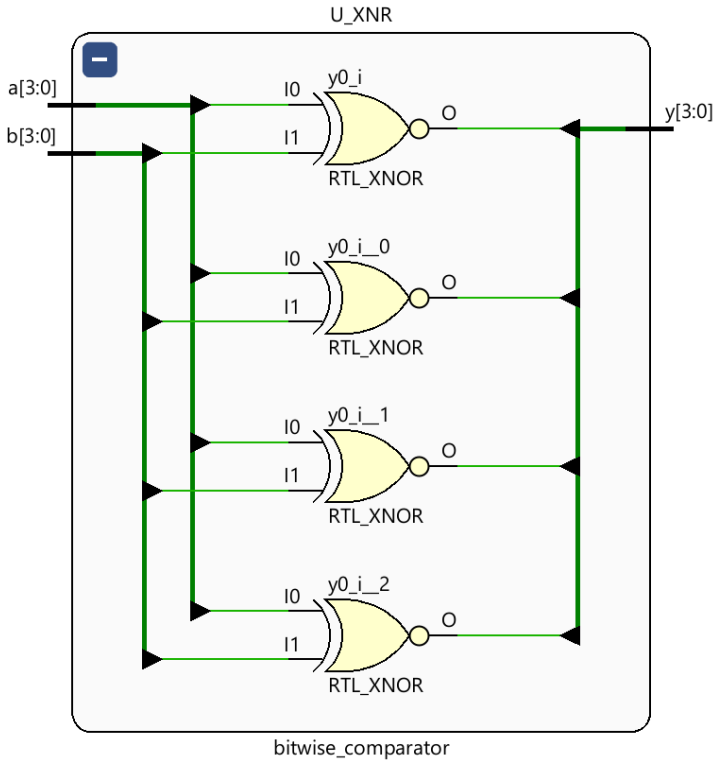

Figure 10: RTL schematic for bitwise_comparator

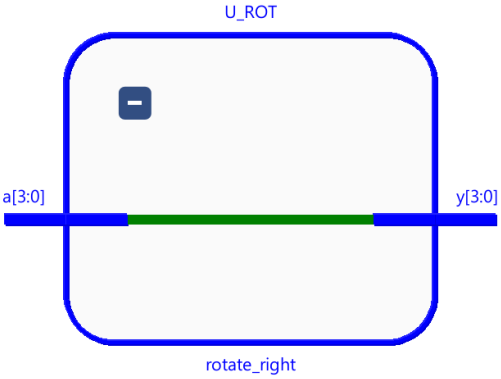

Figure 11: RTL schematic for rotate_right

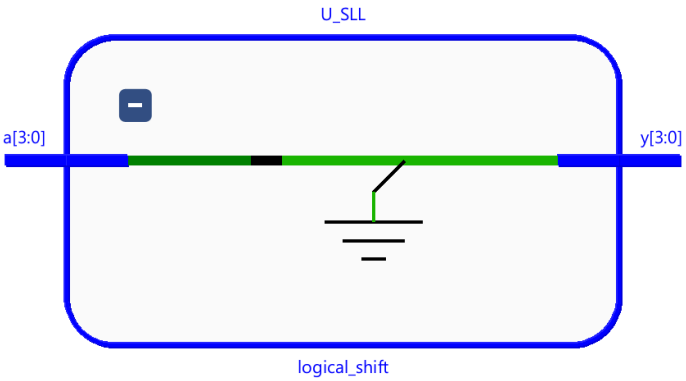

Figure 12: RTL schematic for logical shift

As shown in Figure 11 and 12, the RTL schematics for the logical shift and rotate right submodules look like just a wire. This simplification occurs because these operations are implemented using direct bit assignments in VHDL. Since the operation is simple, it appears only as a wire in the schematic. Also, in the logical shift module, the ground connection is directly tied to the most significant bit to perform the shift.

To check the behavior of the circuit, I used different combinations of inputs for each operation. Figure 13 shows the simulation results for all eight different ALU functions. In the test bench code I observe that the results are corresponds with the expected ones. Thus, the design and test bench simulation were implemented correctly.

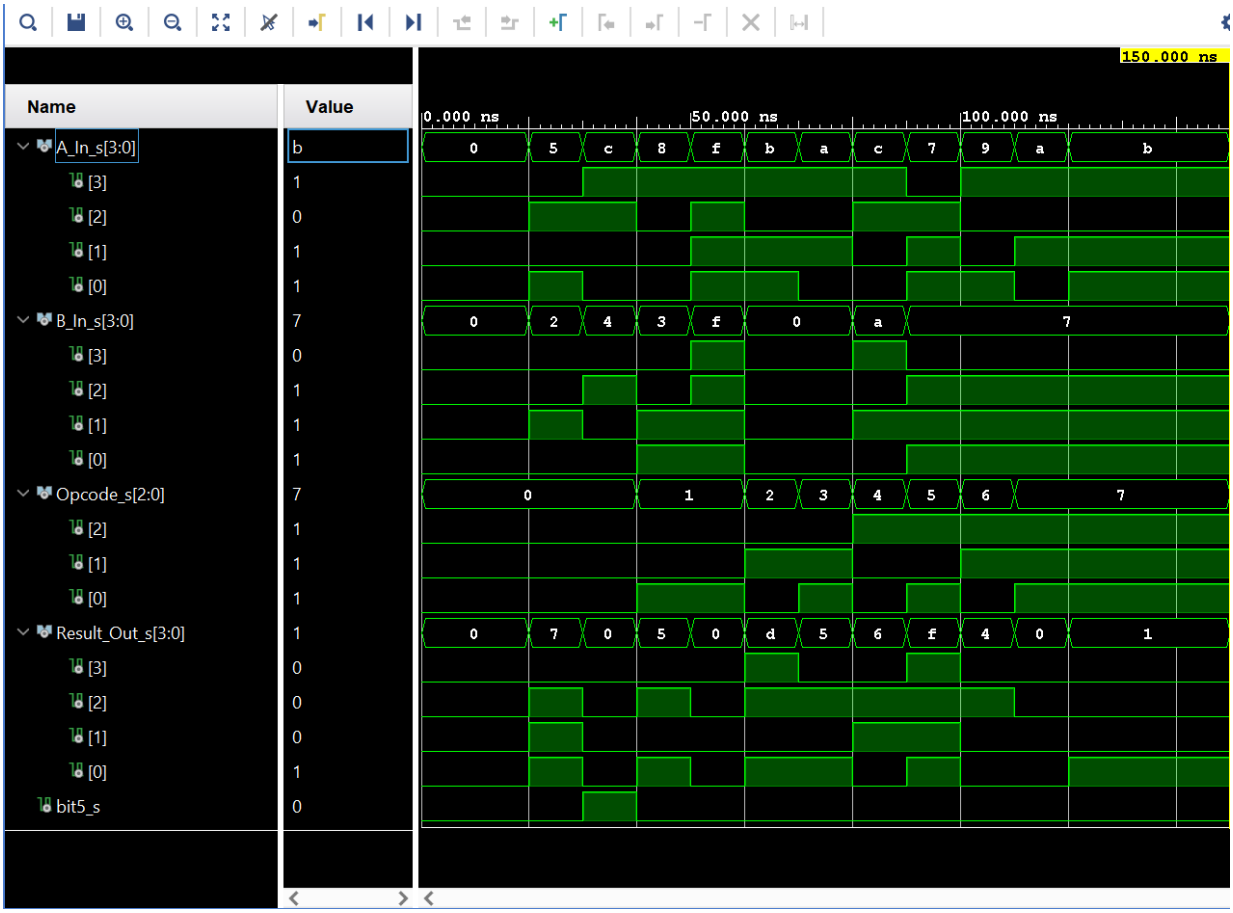

Figure 13: Simulation results of the testbench

After that, the design is implemented on Basys3 board. Figures 14-21 show the outputs of the Basys3 board for different operations and different input combinations. The operation is selected using the three leftmost switches, while the inputs are provided by the rightmost four switches for A_In[3:0] and the next four switches for B_In[3:0]. In Figure 14, I labeled the pin assignments to show which switches correspond to each input signal.

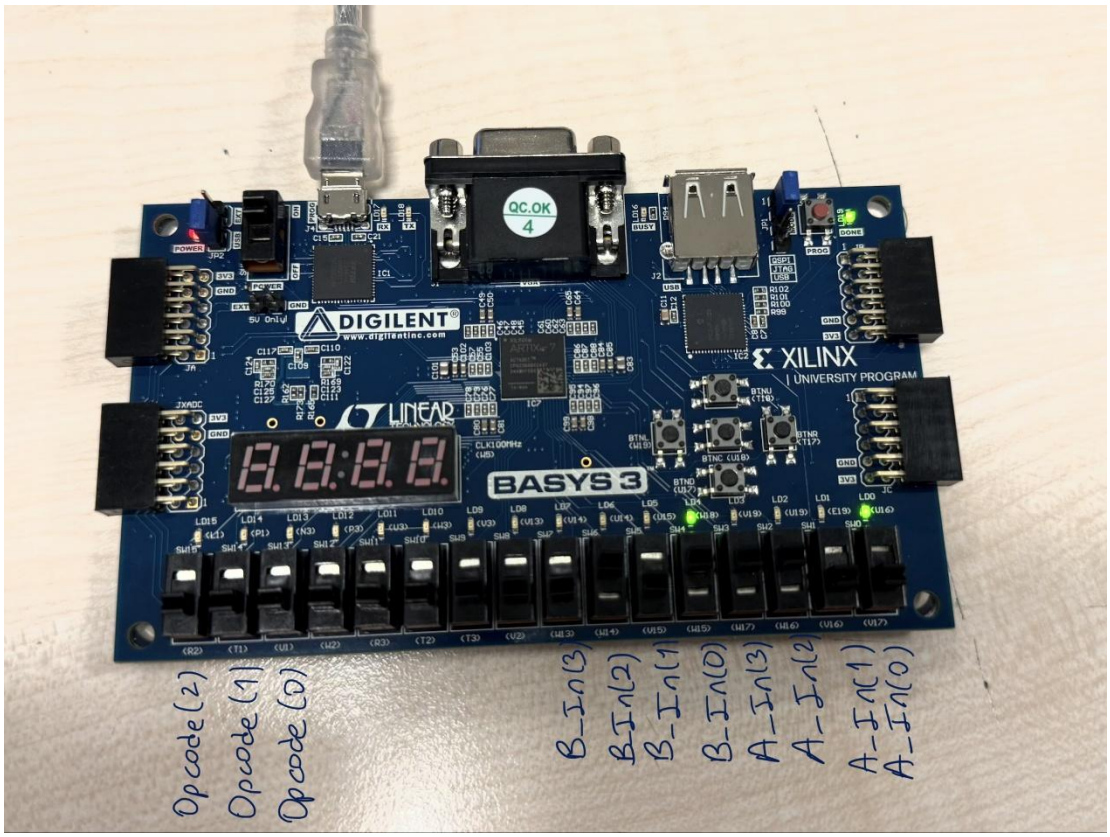

Figure 14: Example of addition operation (Opcode = "000")

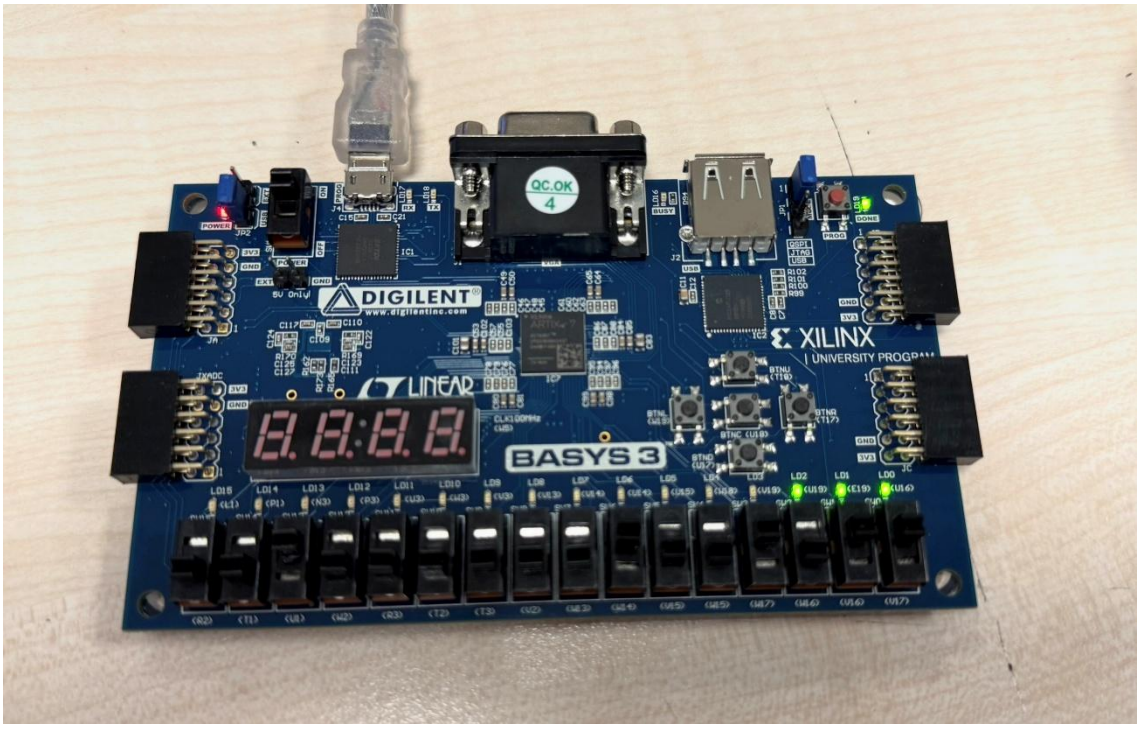

Figure 15: Example of subtraction operation (Opcode = "001")

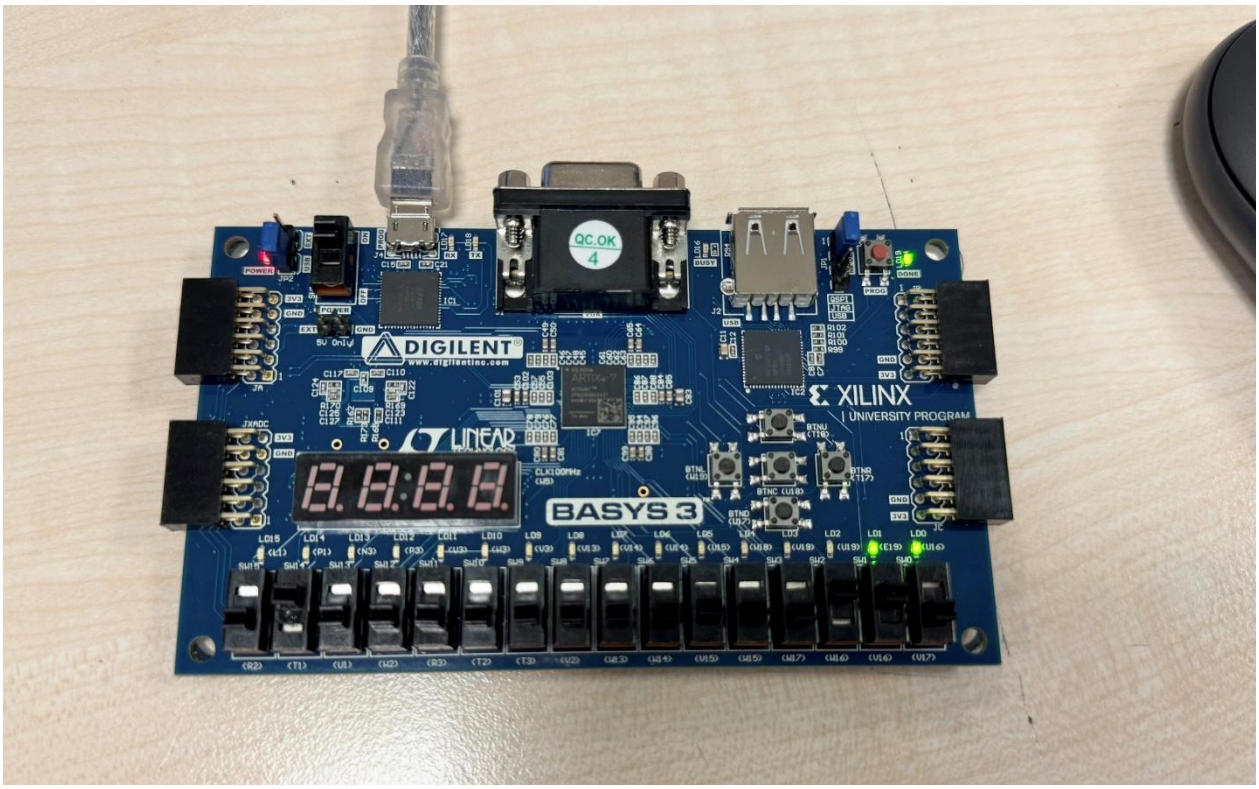

Figure 16: Example of rotate right operation (Opcode = "010")

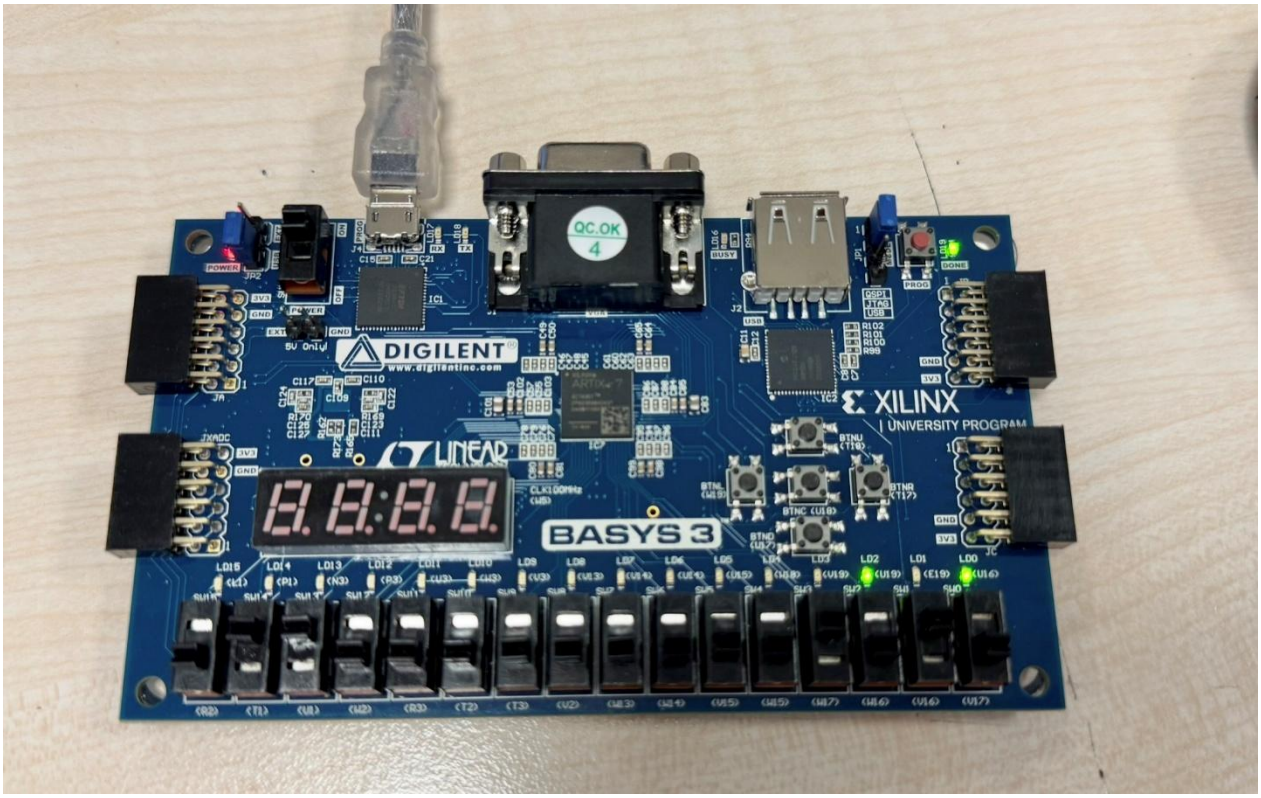

Figure 17: Example of complement operation (Opcode = "011")

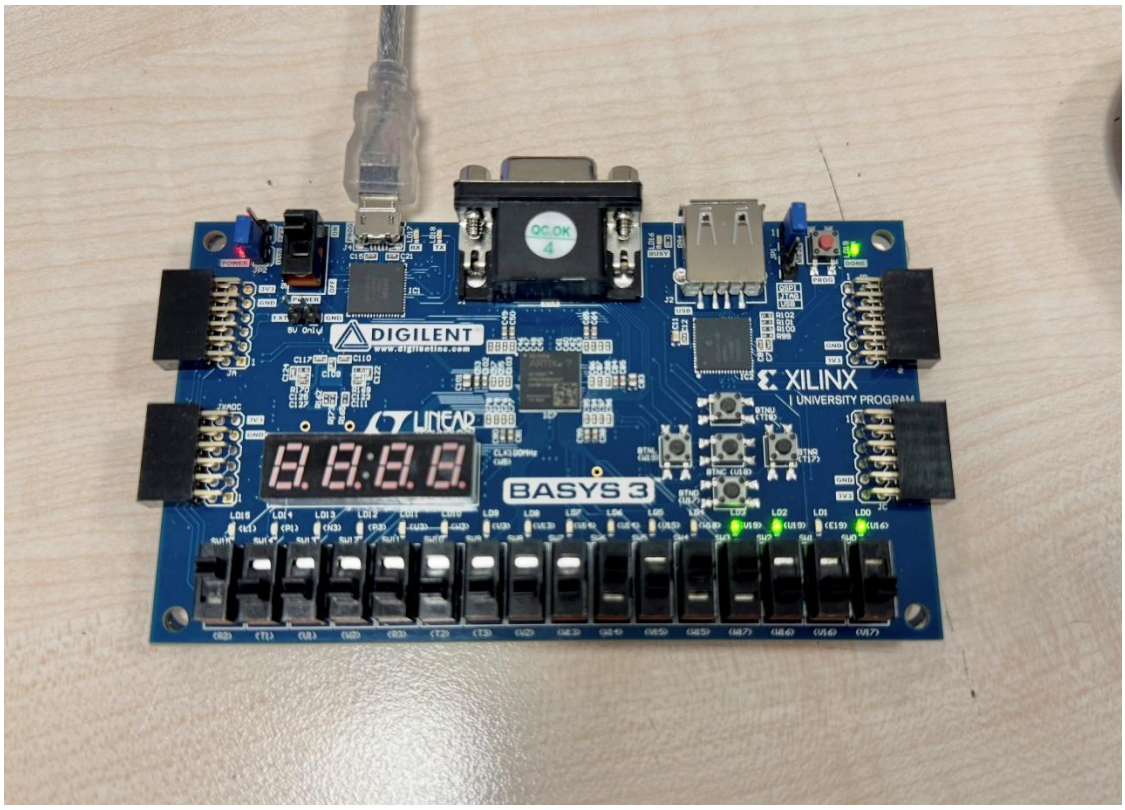

Figure 18: Example of bitwise XOR operation (Opcode = "100")

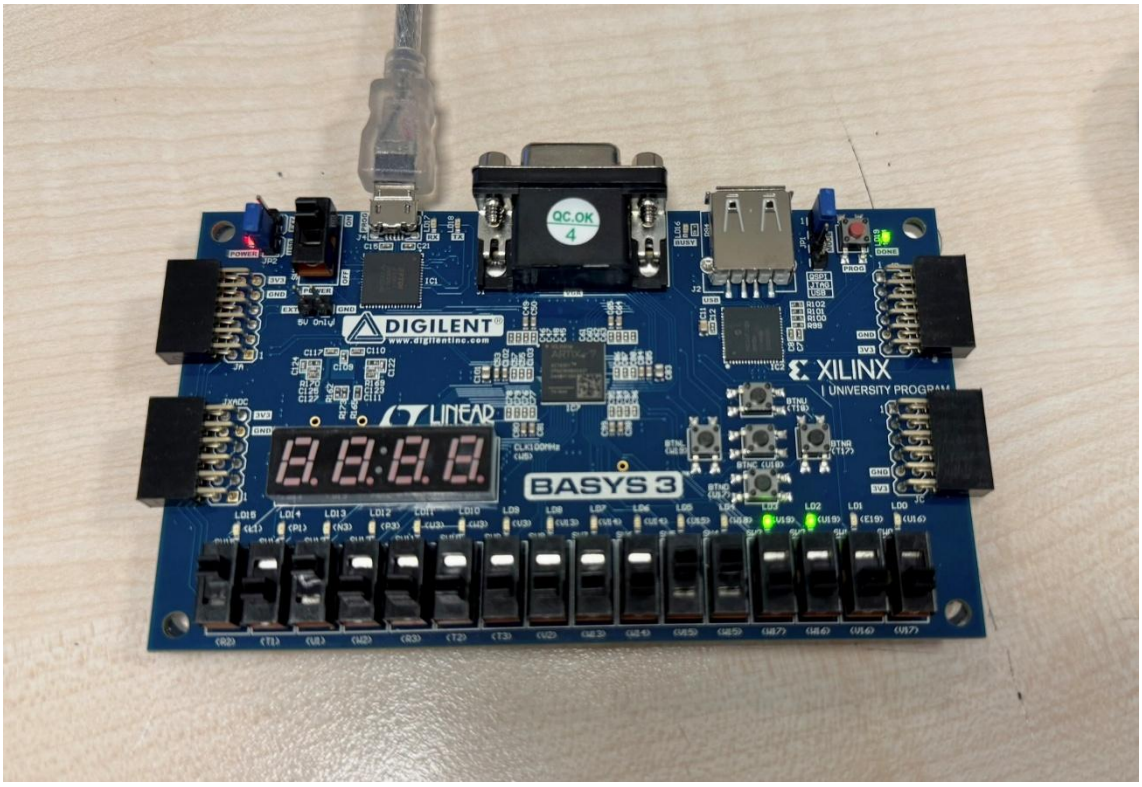

Figure 19: Example of bitwise XNOR operation (Opcode = "101")

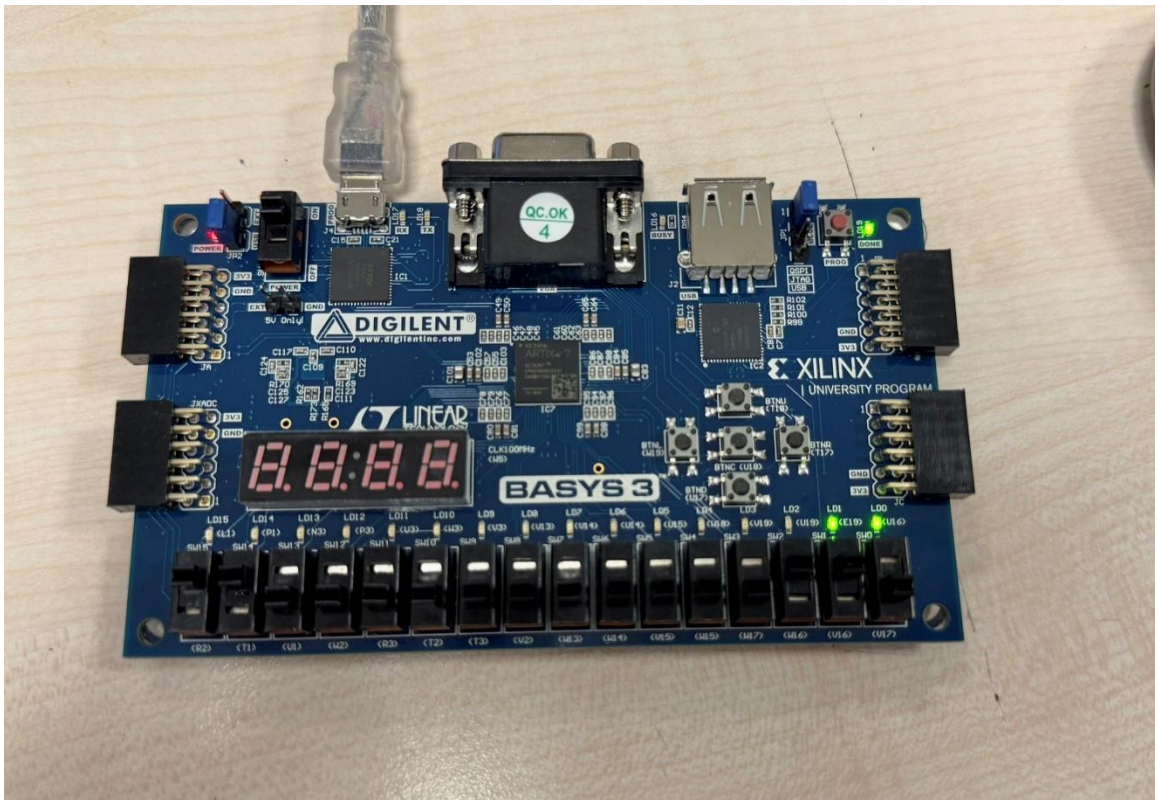

Figure 20: Example of logical shift operation (Opcode = "110")

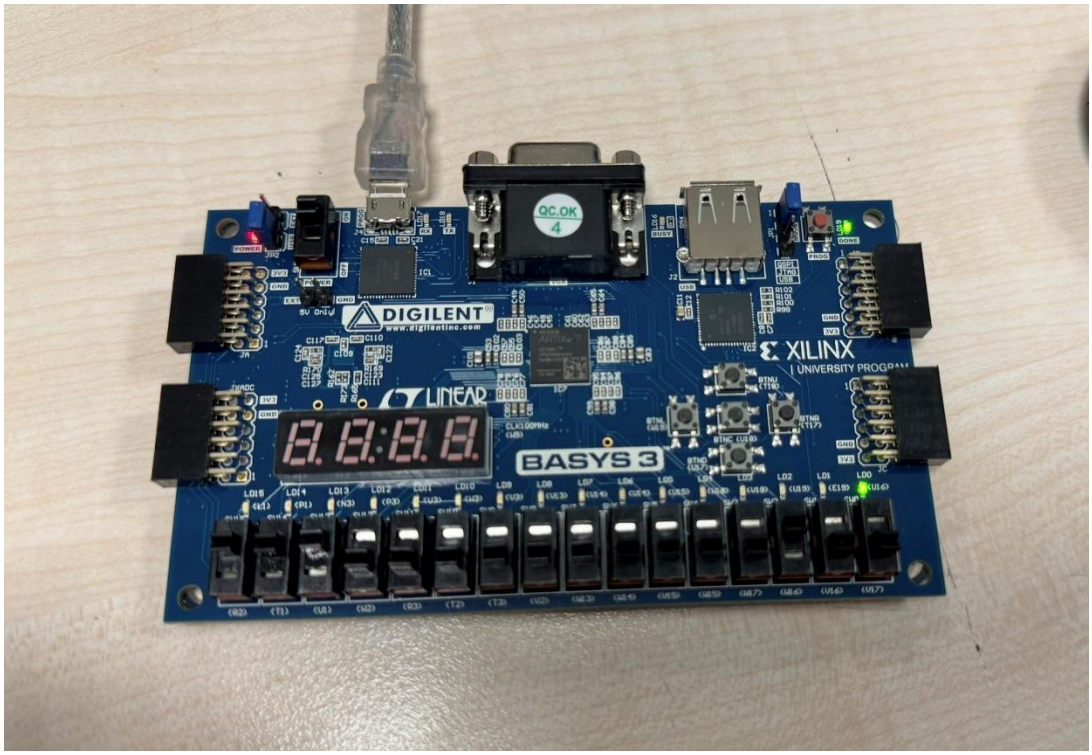

Figure 21: Example of parity check operation (Opcode = "111")

Conclusion

In this experiment, I successfully designed, simulated and implemented the Arithmetic Logic Unit (ALU) on the Basys3 board. The ALU performed eight different operations including addition, subtraction, bitwise and shift functions. Through this lab, I learned how to design arithmetic logic functions using VHDL and implemented them in a modular fashion. I also practice how to write a testbench code, which helps me to gain a practical experience in testing and debugging digital circuits. After that, implementing the design on Basys3 board, I was able to observe outputs through LEDs and verify that the hardware behavior matched the simulation results. Overall, this experiment improved my understanding of digital system design, especially how different arithmetic and logical operations are made and improve my skills to develop and verify design on VHDL. Although this experiment took a significant amount of time due to the complexity of the code and the logic behind it, it was very helpful for me to learn VHDL.

Appendices

ALU_Top.vhd

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


entity ALU_Top is
   port (
      A_In      : in  std_logic_vector(3 downto 0);
      B_In      : in  std_logic_vector(3 downto 0);
      Opcode    : in  std_logic_vector(2 downto 0);
      Result_Out : out std_logic_vector(3 downto 0);
      bit5  : out std_logic
   );
end entity ALU_Top;

architecture Structural of ALU_Top is
   component adder
```

```
    port (a, b: in std_logic_vector(3 downto 0);
     s: out std_logic_vector(3 downto 0); cout: out std_logic);
end component;

component subtracter
    port (a, b: in std_logic_vector(3 downto 0);
     s: out std_logic_vector(3 downto 0); cout: out std_logic);
end component;

component complement
    port (a: in std_logic_vector(3 downto 0);
    y: out std_logic_vector(3 downto 0));
end component;

component rotate_right
    port (a: in std_logic_vector(3 downto 0);
    y: out std_logic_vector(3 downto 0));
end component;

component xor_4bit
    port (a, b: in std_logic_vector(3 downto 0);
     y: out std_logic_vector(3 downto 0));
end component;

component bitwise_comparator -- XNOR
    port (a, b: in std_logic_vector(3 downto 0);
     y: out std_logic_vector(3 downto 0));
end component;

component parity_checker
```

```
    port (a: in std_logic_vector(3 downto 0);
     y: out std_logic_vector(3 downto 0));
  end component;

  component logical_shift
    port (a: in std_logic_vector(3 downto 0);
     y: out std_logic_vector(3 downto 0));
  end component;

  signal S_adder, S_subtracter : std_logic_vector(3 downto 0);
  signal C_adder, C_subtracter : std_logic;

  signal R_COMP, R_ROTATE, R_XOR, R_XNOR, R_SHIFT, R_PARITY : std_logic_vector(3
downto 0);

  signal Mux_Output : std_logic_vector(3 downto 0);

begin

  U_ADD: adder     port map (a => A_In, b => B_In, s => S_adder, cout => C_adder);
  U_SUB: subtracter port map (a => A_In, b => B_In, s => S_subtracter, cout => C_subtracter);

  U_XOR: xor_4bit    port map (a => A_In, b => B_In, y => R_XOR);
  U_XNR: bitwise_comparator port map (a => A_In, b => B_In, y => R_XNOR);

  U_COMP: complement port map (a => A_In, y => R_COMP);
  U_ROT: rotate_right  port map (a => A_In, y => R_ROTATE);

  U_PAR: parity_checker port map (a => A_In, y => R_PARITY);
  U_SLL: logical_shift port map (a => A_In, y => R_SHIFT);
```

```
    with Opcode select
        Mux_Output <= S_adder      when "000",
                      S_subtracter when "001",
                      R_ROTATE     when "010",
                      R_COMP       when "011",
                      R_XOR        when "100",
                      R_XNOR       when "101",
                      R_SHIFT      when "110",
                      R_PARITY     when "111",
                      (others => '0') when others;
    Result_Out <= Mux_Output;
    bit5 <= C_adder      when Opcode = "000" else
                '0';

end architecture Structural;
```

half_adder.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity half_adder is
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           s : out STD_LOGIC;
           c : out STD_LOGIC
      );
end half_adder;
architecture Behavioral of half_adder is

begin
```

```
s <= x xor y;
c <= x and y;
end Behavioral;
```

full_adder.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity full_adder is
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           cin : in STD_LOGIC;
           s : out STD_LOGIC;
           cout : out STD_LOGIC
      );
end full_adder;
architecture Behavioral of full_adder is

Signal s0,c0,c1 : STD_LOGIC;

Component half_adder is
    PORT(
        x, y: in STD_LOGIC;
        s, c: out STD_LOGIC
    );
end component;
begin
U1: half_adder PORT MAP (x,y,s0,c0);
U2: half_adder PORT MAP (s0,cin,s,c1);
cout <= c0 or c1;
end Behavioral;
```

adder.vhd

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

entity adder is
    Port ( a : in std_logic_vector (3 downto 0);
           b : in std_logic_vector (3 downto 0);
           s : out std_logic_vector (3 downto 0);
           cout: out std_logic
    );
end adder;

architecture Behavioral of adder is
Signal c1,c2,c3: std_logic;
Component full_adder is
    PORT(
        x, y, cin: in std_logic;
        s, cout: out std_logic
    );
End Component;
begin
U4: full_adder Port Map(a(0), b(0), '0', s(0), c1);
U5: full_adder Port Map(a(1), b(1), c1, s(1), c2);
U6: full_adder Port Map(a(2), b(2), c2, s(2), c3);
U7: full_adder Port Map(a(3), b(3), c3 , s(3), cout);
end Behavioral;
```

subtracter.vhd

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

```
entity subtracter is
   Port ( a : in std_logic_vector (3 downto 0);
          b : in std_logic_vector (3 downto 0);
          s : out std_logic_vector (3 downto 0);
          cout : out std_logic
   );
end subtracter;

architecture Behavioral of subtracter is
Signal c0,c1, c2, c3: std_logic;
Signal d : std_logic_vector(3 downto 0);
Component full_adder is
   PORT(
      x, y, cin: in std_logic;
      s, cout: out std_logic
   );
   End Component;

begin
c0 <= '1';
d <= not b;
U8: full_adder PORT MAP (a(0), d(0), c0, s(0), c1);
U9: full_adder PORT MAP (a(1), d(1), c1, s(1), c2);
U10: full_adder PORT MAP (a(2), d(2), c2, s(2), c3);
U11: full_adder PORT MAP (a(3), d(3), c3, s(3), cout);

end Behavioral;
```

xor_4bit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity xor_4bit is
    Port ( a : in std_logic_vector (3 downto 0);
           b : in std_logic_vector (3 downto 0);
           y : out std_logic_vector (3 downto 0)
    );
end xor_4bit;
architecture Behavioral of xor_4bit is
begin
    y(0) <= a(0) xor b(0);
    y(1) <= a(1) xor b(1);
    y(2) <= a(2) xor b(2);
    y(3) <= a(3) xor b(3);

end Behavioral;
```

bitwise_comparator.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity bitwise_comparator is
    Port ( a : in std_logic_vector (3 downto 0);
           b : in std_logic_vector (3 downto 0);
           y : out std_logic_vector (3 downto 0)
    );
end bitwise_comparator;
```

```
architecture Behavioral of bitwise_comparator is
begin
    y(0) <= a(0) xnor b(0);
    y(1) <= a(1) xnor b(1);
    y(2) <= a(2) xnor b(2);
    y(3) <= a(3) xnor b(3);
end Behavioral;
```

complement.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity complement is
    Port ( a : in std_logic_vector (3 downto 0);
           y : out std_logic_vector (3 downto 0)
    );
end complement;

architecture Behavioral of complement is
begin
y(0) <= not a(0);
y(1) <= not a(1);
y(2) <= not a(2);
y(3) <= not a(3);
end Behavioral;
```

rotate.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity rotate_right is
    Port ( a : in std_logic_vector (3 downto 0);
           y : out std_logic_vector (3 downto 0)
    );
end rotate_right;
architecture Behavioral of rotate_right is
begin
y <= a(0) & a(3 downto 1);
end Behavioral;
```

parity_checker.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity parity_checker is
    Port ( a : in std_logic_vector (3 downto 0);
           y : out std_logic_vector (3 downto 0)
    );
end parity_checker;

architecture Behavioral of parity_checker is
    signal Parity_bit: std_logic;
begin
    Parity_bit <= a(3) xor a(2) xor a(1) xor a(0);
    y(0) <= Parity_bit;
    y(3 downto 1) <= (others => '0');
```

```
end Behavioral;
```

logical_shift.vhd

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

entity logical_shift is
    Port ( a : in std_logic_vector (3 downto 0);
           y : out std_logic_vector (3 downto 0)
    );
end logical_shift;

architecture Behavioral of logical_shift is
begin
y <= '0' & a(3 downto 1);
end Behavioral;
```

ALU_Top_tb.vhd

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

entity ALU_Top_tb is
end entity ALU_Top_tb;

architecture TestBench of ALU_Top_tb is

    component ALU_Top
        port (
            A_In       : in  std_logic_vector(3 downto 0);
```

```
        B_In       : in  std_logic_vector(3 downto 0);
        Opcode     : in  std_logic_vector(2 downto 0);
        Result_Out : out std_logic_vector(3 downto 0);
        bit5  : out std_logic
    );
  end component;

  signal A_In_s       : std_logic_vector(3 downto 0) := (others => '0');
  signal B_In_s       : std_logic_vector(3 downto 0) := (others => '0');
  signal Opcode_s     : std_logic_vector(2 downto 0) := (others => '0');

  signal Result_Out_s : std_logic_vector(3 downto 0);
  signal bit5_s  : std_logic;



begin

  DUT : ALU_Top
    port map (
      A_In       => A_In_s,
      B_In       => B_In_s,
      Opcode     => Opcode_s,
      Result_Out => Result_Out_s,
      bit5  => bit5_s
    );

  stimulus_process: process
  begin
```

```
wait for 2 * 10 ns;

Opcode_s <= "000";
A_In_s <= "0101";
B_In_s <= "0010";
wait for 10 ns;
-- Result="00111"

A_In_s <= "1100";
B_In_s <= "0100";
wait for 10 ns;
-- Result="10000"

Opcode_s <= "001";
A_In_s <= "1000";
B_In_s <= "0011";
wait for 10 ns;
-- Result="0101"

A_In_s <= "1111";
B_In_s <= "1111";
wait for 10 ns;
-- Result="0000"

Opcode_s <= "010";
A_In_s <= "1011";
B_In_s <= "0000";
wait for 10 ns;
-- Result="1101"
```

```
Opcode_s <= "011";
A_In_s <= "1010";
wait for 10 ns;
-- Result="0101"

Opcode_s <= "100";
A_In_s <= "1100";
B_In_s <= "1010";
wait for 10 ns;
-- Result="0110"

Opcode_s <= "101";
A_In_s <= "0111";
B_In_s <= "0111";
wait for 10 ns;
-- Result="1111"

Opcode_s <= "110";
A_In_s <= "1001";
wait for 10 ns;

-- Result="0100"

Opcode_s <= "111";
A_In_s <= "1010";
wait for 10 ns;
-- Result="0000"

A_In_s <= "1011";
```

```
        wait for 10 ns;
        -- Result="0001"

        wait;

    end process stimulus_process;

end architecture TestBench;
```

constraint.xdc

```
set_property PACKAGE_PIN V17 [get_ports {A_In[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {A_In[0]}]
set_property PACKAGE_PIN V16 [get_ports {A_In[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {A_In[1]}]
set_property PACKAGE_PIN W16 [get_ports {A_In[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {A_In[2]}]
set_property PACKAGE_PIN W17 [get_ports {A_In[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {A_In[3]}]

set_property PACKAGE_PIN W15 [get_ports {B_In[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B_In[0]}]
set_property PACKAGE_PIN V15 [get_ports {B_In[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B_In[1]}]
set_property PACKAGE_PIN W14 [get_ports {B_In[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B_In[2]}]
set_property PACKAGE_PIN W13 [get_ports {B_In[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B_In[3]}]

set_property PACKAGE_PIN U1 [get_ports {Opcode[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Opcode[0]}]
```

```
set_property PACKAGE_PIN T1 [get_ports {Opcode[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Opcode[1]}]
set_property PACKAGE_PIN R2 [get_ports {Opcode[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Opcode[2]}]

set_property PACKAGE_PIN U16 [get_ports {Result_Out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Result_Out[0]}]
set_property PACKAGE_PIN E19 [get_ports {Result_Out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Result_Out[1]}]
set_property PACKAGE_PIN U19 [get_ports {Result_Out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Result_Out[2]}]
set_property PACKAGE_PIN V19 [get_ports {Result_Out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Result_Out[3]}]

set_property PACKAGE_PIN W18 [get_ports {bit5}]
set_property IOSTANDARD LVCMOS33 [get_ports {bit5}]
```

References

https://en.wikipedia.org/wiki/Arithmetic_logic_unit

basys3xdc.pdf

https://nandland.com/learn-vhdl/