EEE102 Lab 4 Report: Arithmetic Logic Unit

Nizam Ercan 22302317 EEE102-2

1- Purpose:

The objective of this lab is to construct an Arithmetic Logic Unit (ALU) capable of performing of eight distinct functions. Using the knowledge acquired from previous labs on VHDL and BASYS-3, we aimed to implement fundamental arithmetic operations like addition and subtraction, alongside functionalities such as shifting and bitwise operations. Through this lab, I reinforced my understanding of VHDL and FPGA-based digital design.

2- Methodology:

My initial step was to select eight functions for implementation in the ALU. The first two functions, addition and subtraction, were determined from the lab manual. Additionally, the remaining functions needed to include at least one bitwise and one shift operation. For this lab, I am selecting incrementing, decrementing, right shift, left shift, nand gate and xnor gate operations. For the ALU design, I am using 4-bit unsigned binary numbers to represent inputs. The implementation of these circuits was integrated with switches and LEDs on Basys3 where a constraints file was written to assign the inputs and outputs to appropriate ports. I used a modular approach for this design where each operation and additionally a full adder was coded in different files and mapped to a top module. Also, a testbench was written to simulate the behavior of the top module to detect some possible errors in the design before implementing it on Basys3.

3- Design Specifications:

There are 3 inputs which are called first_number (4 bits), second_number(4 bits) and selection(3 bits) and 1 output which is called result(4 bits). The inputs are controlled with switches and the output can be observed with the LEDs.

My ALU consists of 8 modules that represents the operations as you can see below:

1- summation.vhd
2- substractor.vhd
3- increment.vhd
4- decrement.vhd
5- right_shift.vhd
6- left_shift.vhd
7- nand_gate.vhd
8- xnor_gate.vhd

For summation and substractor module to work, I used 4 half adders for each. Therefore, an additional sub module full_adder.vhd was also written.

Also, I used the modules substractor.vhd and summation.vhd to write the modules increment.vhd and decrement.vhd respectively as I have already written them before.

The user can select which operations to make by changing the selection input by switches. In table 1, you can see which "selection" input corresponds to which operation with the inputs and outputs of the operations.

"selection" inputs	Operation	Inputs	Outputs
0b000	Addition	first_number, second_number	result
0b001	Subtraction	first_number, second_number	result
0b010	Incrementing	first_number	result
0b011	Decrementing	first_number	result
0b100	Right shift	first_number	result
0b101	Left shift	first_number	result
0b110	nand gate	first_number, second_number	result
0b111	xnor gate	first_number, second_number	result

Table 1: Assigned operations to "selection" inputs

4- Results:

After designing all the modules and coding the testbench in Vivado I obtained the final RTL schematics of my design as you can see in Figure 1. In this design, you can see what inputs each module takes and how the selection input helps us to choose the wanted output since it is implemented as a multiplexer. How each individual module operates with the logic gates inside them is also added to the appendix.

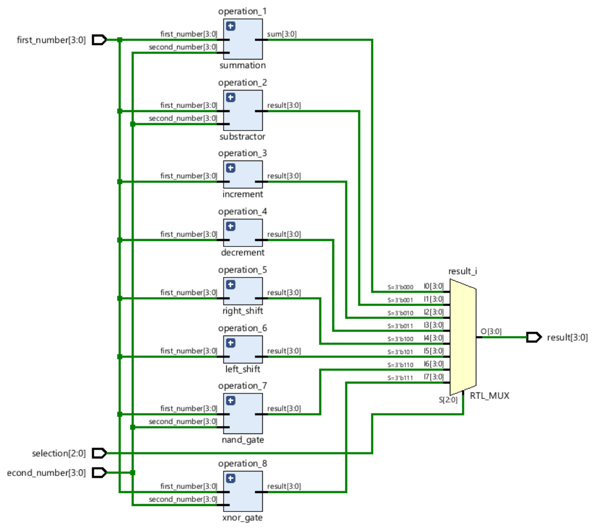


Figure 1: RTL schematics of ALU

For simulating the design before implementing it to Basys3, I checked 2 examples for each operation and you can see the result of this simulation in Figure 2.

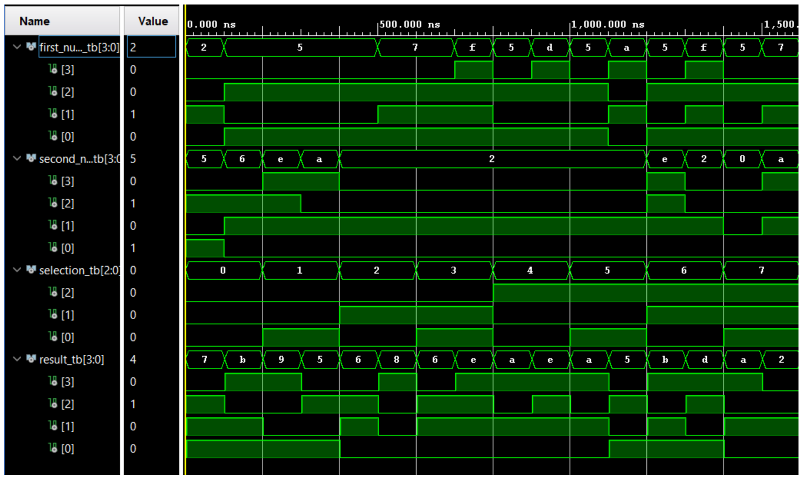


Figure 2: The results of the simulation

We can observe that the design is working without any issue in this simulation which means that it is ready to be implemented on a real FPGA.

As you can see in the constraints file, which is also included in the appendix:

- selection(2) to selection(0) are assigned to “R2”, “T1” and “U1” pins respectively.
- first_number(3) to first_number(0) are assigned to “W17”, “W16”, “V16”, “V17” pins respectively
- second_number(3) to second_number(0) are assigned to “V2”, “W13”, “W14”, “V15” pins respectively.
- result(3) to result(0) are assigned to “V19”, “U19”, “E19”, “U16” pins respectively.

With these pin configurations, the design is synthesized and implemented to Basys3. In figures 3 and 4, you can see two examples from the working Basys3. More examples were added to the appendix.

Example 1:

Selection = “000”

first_number = “0110”

second_number = “0100”

result = “1010”

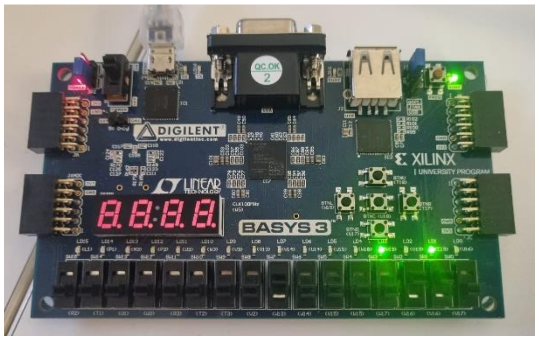


Figure 3: First example

Example 2:

selection = "110"

first_number = "1010"

second_number = "0011"

result = "1101"

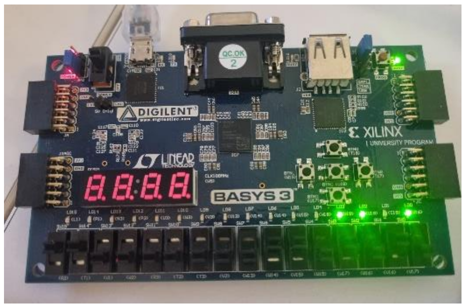


Figure 4: second example

5- Conclusion:

In this lab, I developed an ALU using VHDL and Basys3, embracing a modular design approach. The ALU was designed to support eight distinct functions, providing the capability to perform various operations on 4-bit unsigned binary numbers. While the implemented ALU meets the specified requirements in the lab manual and demonstrates expected functionality in both simulation and FPGA implementation, there are few areas for improvement. Notably, certain operations, including addition and subtraction, are currently limited to unsigned binary numbers only and do not support negative inputs. Also, I am ignoring the possibility of an overflow in the operations summation, subtraction, incrementing and decrementing. Overall, this project provided valuable insights into digital logic design, VHDL coding, and FPGA implementation.

6- Appendix:

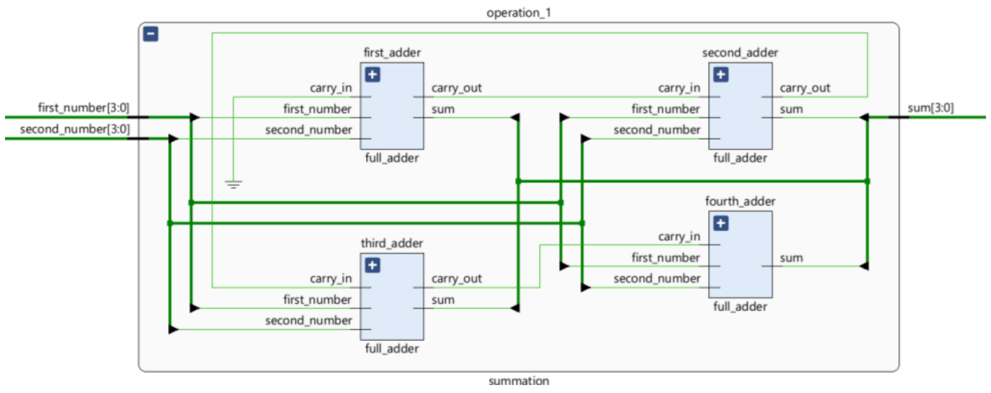


Figure A: summation module

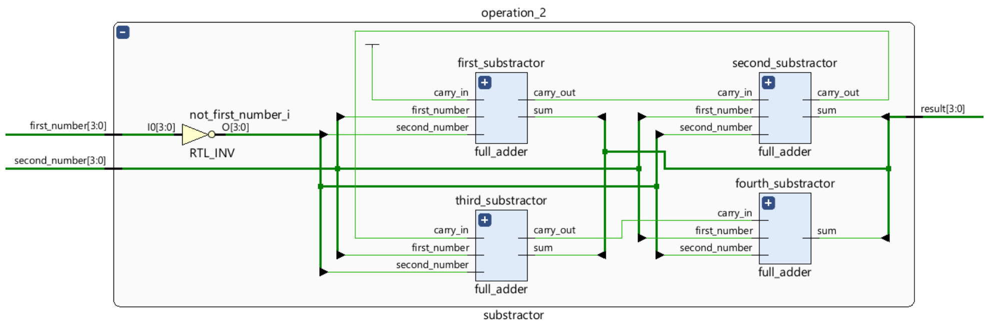


Figure B: substractor module

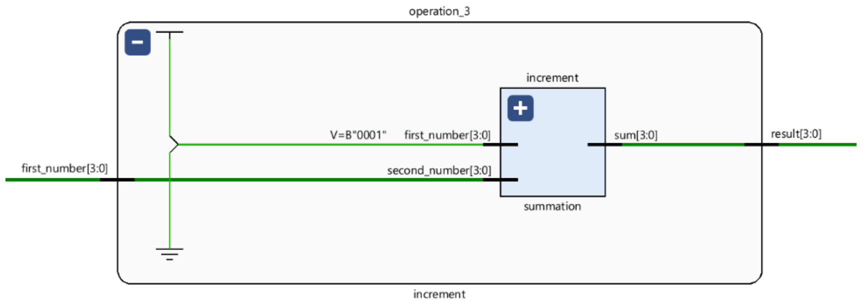


Figure C: increment module

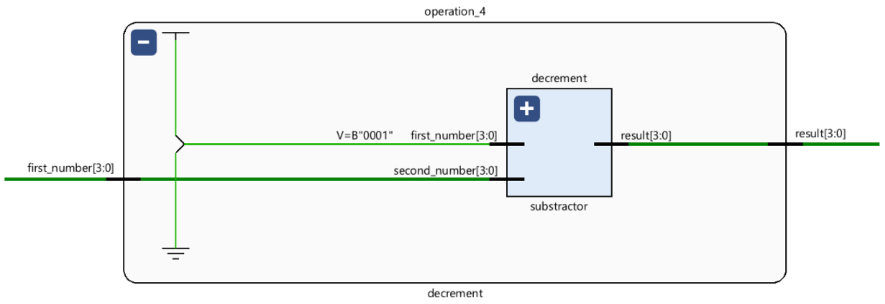
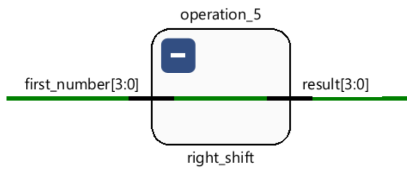


Figure D: decrementing module



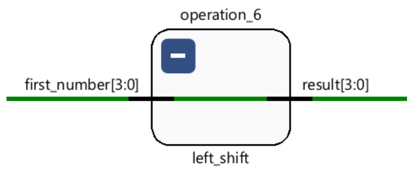


Figure E: left shift and right shift modules

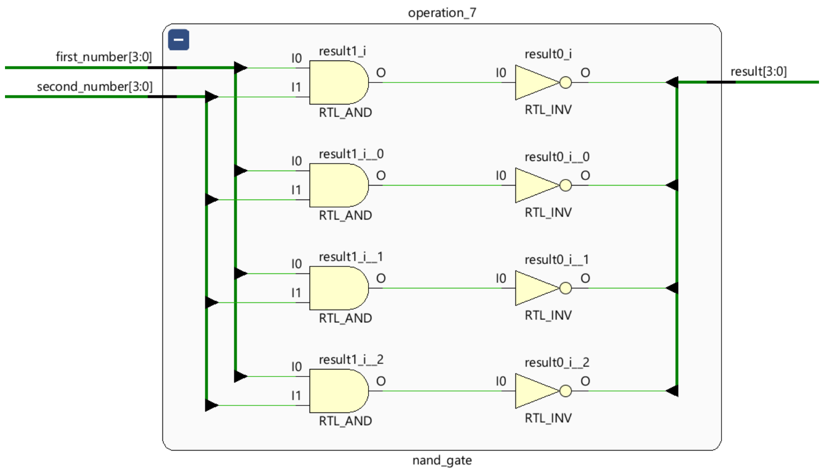


Figure F: nand gate module

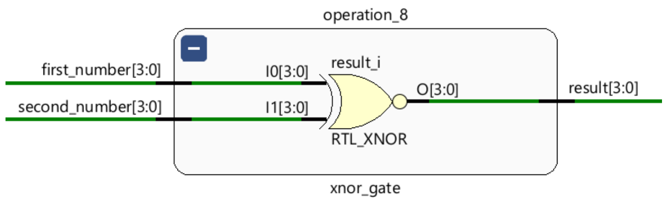


Figure G: xnor gate

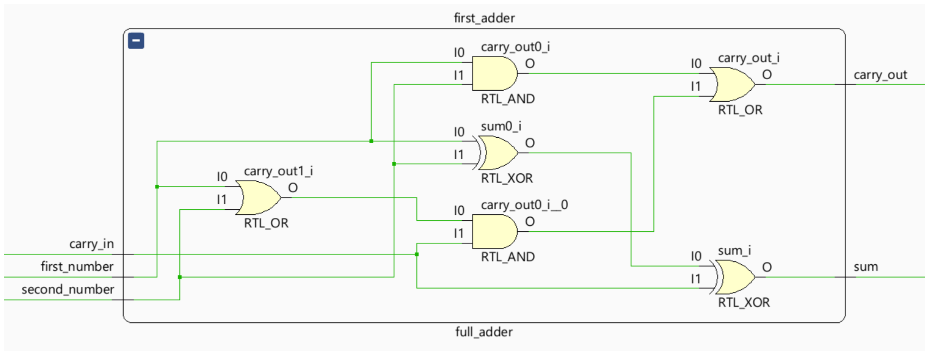


Figure H: full adder module

Example 3:

select = “001”

first_number = “0101”

second_number = “0111”

result = “0010”

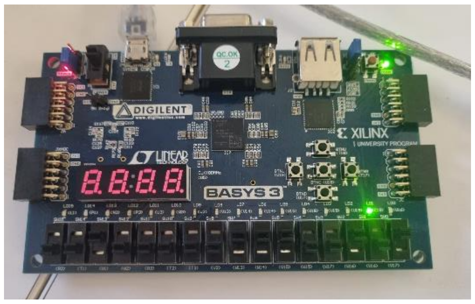


Figure I: example 3

Example 4:

select = “010”

first_number = “0101”

result = “0110”

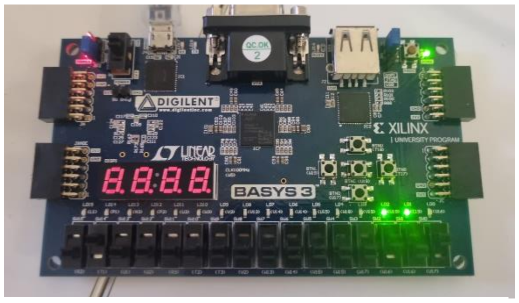


Figure J: fourth example

Example 5:

select = “011”

first_number = “0101”

result = “0100”

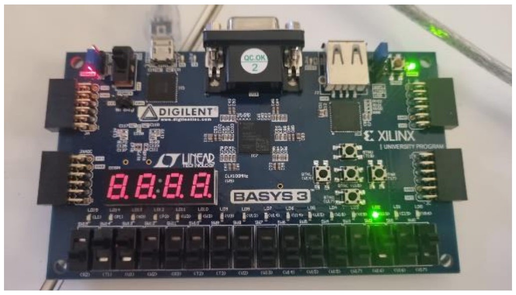


Figure K: fifth example

Example 6:

select = “100”

first_number = “0111”

result = “1011”

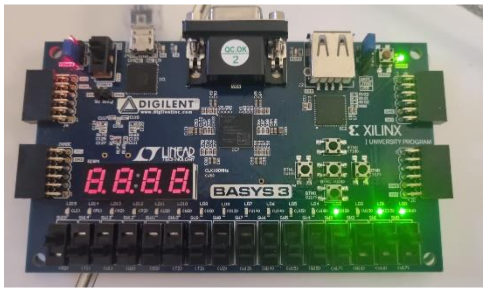


Figure L: sixth example

Example 7:

select = “101”

first_number = “0110”

result = “1100”

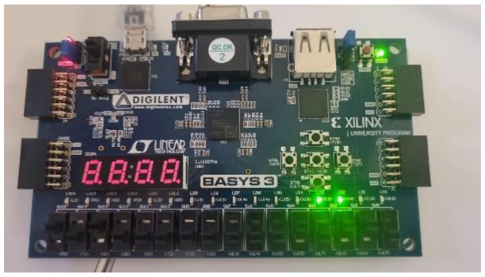


Figure M: seventh example

Example 8:

select = “101”

first_number = “0100”

second_number = “0110”

result = “1100”

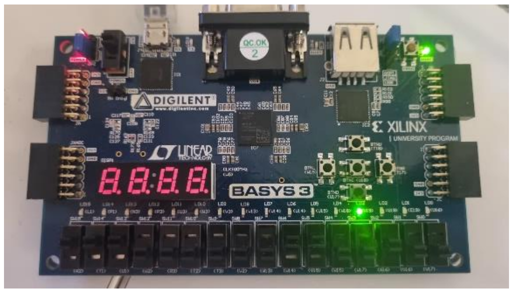


Figure N: eighth example

- Top module: top.vhd:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;



entity top is

    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);

           second_number : in STD_LOGIC_VECTOR (3 downto 0);

           selection : in STD_LOGIC_VECTOR (2 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end top;

architecture Behavioral of top is

component summation is
    Port ( first_number : in STD_LOGIC_vector (3 downto 0);
           second_number : in STD_LOGIC_vector (3 downto 0);
           sum : out STD_LOGIC_vector(3 downto 0)
           );
end component;

component substractor is
    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
           second_number : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component increment is
    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component decrement is
    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component right_shift is
    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end component;
```

```
component left_shift is
    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component nand_gate is
    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
           second_number : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component xnor_gate is
    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
           second_number : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal out1, out2, out3, out4, out5, out6, out7, out8 : std_logic_vector(3
downto 0);
signal first_number_s, second_number_s : std_logic_vector(3 downto 0);

begin

first_number_s <= first_number;
second_number_s <= second_number;

operation_1: summation port map(first_number_s, second_number_s, out1);
operation_2: substractor port map(first_number_s, second_number_s, out2);
operation_3: increment port map(first_number_s, out3);
operation_4: decrement port map(first_number_s, out4);
operation_5: right_shift port map(first_number_s, out5);
```

```
operation_6: left_shift port map(first_number_s, out6);

operation_7: nand_gate port map(first_number_s, second_number_s ,out7);

operation_8: xnor_gate port map(first_number_s, second_number_s, out8);



process is

begin

    case selection is

        when "000" => result <= out1;

        when "001" => result <= out2;

        when "010" => result <= out3;

        when "011" => result <= out4;

        when "100" => result <= out5;

        when "101" => result <= out6;

        when "110" => result <= out7;

        when "111" => result <= out8;

        when others => result <= "0100";

    end case;

    wait for 1ns;

end process;

end Behavioral;
```

- Testbenc: sim.vhd

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

entity top_tb is
end top_tb;

architecture Behavioral of top_tb is

    component top
        Port (
            first_number  : in  STD_LOGIC_VECTOR (3 downto 0);
            second_number : in  STD_LOGIC_VECTOR (3 downto 0);
            selection     : in  STD_LOGIC_VECTOR (2 downto 0);
            result        : out STD_LOGIC_VECTOR (3 downto 0)
        );
    end component;

    signal first_number_tb  : STD_LOGIC_VECTOR (3 downto 0) ;
    signal second_number_tb : STD_LOGIC_VECTOR (3 downto 0) ;
    signal selection_tb     : STD_LOGIC_VECTOR (2 downto 0) ;
    signal result_tb        : STD_LOGIC_VECTOR (3 downto 0) ;

begin

    test_bench: top port map (
        first_number  => first_number_tb,
        second_number => second_number_tb,
        selection     => selection_tb,
        result        => result_tb
    );

    stimulus_proc: process
    begin

        first_number_tb  <= "0010";
        second_number_tb <= "0101";
        selection_tb     <= "000";
        wait for 101 ns;
        first_number_tb  <= "0101";
        second_number_tb <= "0110";
        selection_tb     <= "000";
        wait for 100 ns;
        first_number_tb  <= "0101";
        second_number_tb <= "1110";
        selection_tb     <= "001";
        wait for 100 ns;
        first_number_tb  <= "0101";
        second_number_tb <= "1010";
        selection_tb     <= "001";
        wait for 100 ns;
        first_number_tb  <= "0101";
        second_number_tb <= "0010";
        selection_tb     <= "010";
        wait for 100 ns;
        first_number_tb  <= "0111";
        second_number_tb <= "0010";
        selection_tb     <= "010";
        wait for 100 ns;
        first_number_tb  <= "0111";
        second_number_tb <= "0010";
        selection_tb     <= "011";
        wait for 100 ns;
        first_number_tb  <= "1111";
        second_number_tb <= "0010";
        selection_tb     <= "011";
        wait for 100 ns;
        first_number_tb  <= "0101";
        second_number_tb <= "0010";
        selection_tb     <= "100";
        wait for 100 ns;
        first_number_tb  <= "1101";
        second_number_tb <= "0010";
        selection_tb     <= "100";
        wait for 100 ns;
        first_number_tb  <= "0101";
        second_number_tb <= "0010";
        selection_tb     <= "101";
        wait for 100 ns;
        first_number_tb  <= "1010";
        second_number_tb <= "0010";
        selection_tb     <= "101";
        wait for 100 ns;
        first_number_tb  <= "0101";
        second_number_tb <= "1110";
        selection_tb     <= "110";
        wait for 100 ns;
        first_number_tb  <= "1111";
        second_number_tb <= "0010";
        selection_tb     <= "110";
        wait for 100 ns;
        first_number_tb  <= "0101";
        second_number_tb <= "0000";
        selection_tb     <= "111";
        wait for 100 ns;
        first_number_tb  <= "0111";
        second_number_tb <= "1010";
        selection_tb     <= "111";
        wait for 100 ns;

    end process;

end Behavioral;
```

- Summation: summation.vhd

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

entity summation is
    Port ( first_number : in STD_LOGIC_vector (3 downto 0);
           second_number : in STD_LOGIC_vector (3 downto 0);
           sum : out STD_LOGIC_vector(3 downto 0)
           );
end summation;

architecture Behavioral of summation is

    component full_adder is
        Port ( first_number : in STD_LOGIC;
           second_number : in STD_LOGIC;
           carry_in : in STD_LOGIC;
           carry_out : out STD_LOGIC;
           sum : out STD_LOGIC);
    end component;

signal carry : std_logic_vector(3 downto 0);
signal overflow: std_logic;
signal first_number_s: std_logic_vector(3 downto 0);
signal second_number_s: std_logic_vector(3 downto 0);

begin

first_number_s <= first_number;
second_number_s <= second_number;

first_adder : full_adder port map(first_number_s(0), second_number_s(0), '0',
carry(1), sum(0));
second_adder : full_adder port map(first_number_s(1), second_number_s(1),
carry(1), carry(2), sum(1));
third_adder : full_adder port map(first_number_s(2), second_number_s(2),
carry(2), carry(3), sum(2));
fourth_adder : full_adder port map(first_number_s(3), second_number_s(3),
carry(3), overflow, sum(3));


end Behavioral;
```

- Subtraction: substractor.vhd

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


entity substractor is
    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
           second_number : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end substractor;

architecture Behavioral of substractor is

    component full_adder is
            Port ( first_number : in STD_LOGIC;
               second_number : in STD_LOGIC;
               carry_in : in STD_LOGIC;
               carry_out : out STD_LOGIC;
               sum : out STD_LOGIC);
        end component;


signal carry: std_logic_vector(3 downto 0);
signal not_first_number : std_logic_vector(3 downto 0);
signal overflow : std_logic;

begin

not_first_number <= not first_number;


first_substractor: full_adder port map(second_number(0), not_first_number(0),
carry(0), carry(1), result(0));
```

```
second_substractor: full_adder port map(second_number(1),
not_first_number(1), carry(1), carry(2), result(1));
third_substractor: full_adder port map(second_number(2), not_first_number(2),
carry(2), carry(3), result(2));
fourth_substractor: full_adder port map(second_number(3),
not_first_number(3), carry(3), overflow, result(3));

carry(0) <= '1';


end Behavioral;
```

- Incrementation: increment.vhd

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

entity increment is
    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end increment;

architecture Behavioral of increment is

component summation is
    Port ( first_number : in STD_LOGIC_vector (3 downto 0);
           second_number : in STD_LOGIC_vector (3 downto 0);
           sum : out STD_LOGIC_vector(3 downto 0)
           );
end component;

signal first_number_s : std_logic_vector(3 downto 0);

begin

first_number_s <= first_number;

increment : summation port map("0001", first_number_s, result);


end Behavioral;
```

- Decrementing: decrement.vhd

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


entity decrement is
    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end decrement;

architecture Behavioral of decrement is
```

```
component substractor is
    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
           second_number : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal first_number_s: std_logic_vector(3 downto 0);

begin

first_number_s <= first_number;

decrement: substractor port map("0001", first_number_s, result);

end Behavioral;
```

- Right Shift: right_shift.vhd

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


entity right_shift is
    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end right_shift;

architecture Behavioral of right_shift is

begin

result(0) <= first_number(1);
result(1) <= first_number(2);
result(2) <= first_number(3);
result(3) <= first_number(0);

end Behavioral;
```

- Left shift: left_shift.vhd

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


entity left_shift is
    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end left_shift;

architecture Behavioral of left_shift is

begin
```

```
result(0) <= first_number(3);
result(1) <= first_number(0);
result(2) <= first_number(1);
result(3) <= first_number(2);


end Behavioral;
```

- Nand gate: nand_gate

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

entity nand_gate is
    Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
           second_number : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0));
end nand_gate;

architecture Behavioral of nand_gate is

begin

result(0) <= first_number(0) nand second_number(0);
result(1) <= first_number(1) nand second_number(1);
result(2) <= first_number(2) nand second_number(2);
result(3) <= first_number(3) nand second_number(3);


end Behavioral;
```

- Xnor gate: xnor_gate.vhd

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

    entity xnor_gate is
        Port ( first_number : in STD_LOGIC_VECTOR (3 downto 0);
               second_number : in STD_LOGIC_VECTOR (3 downto 0);
               result : out STD_LOGIC_VECTOR (3 downto 0));
    end xnor_gate;

    architecture Behavioral of xnor_gate is

    begin

        result <= first_number xnor second_number;


    end Behavioral;
```

- Full adder: full_adder.vhd

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


entity full_adder is
    Port ( first_number : in STD_LOGIC;
           second_number : in STD_LOGIC;
           carry_in : in STD_LOGIC;
           carry_out : out STD_LOGIC;
           sum : out STD_LOGIC);
end full_adder;


architecture Behavioral of full_adder is


begin

sum <= first_number xor second_number xor carry_in;
carry_out <= (first_number and second_number) or ((first_number or
second_number) and carry_in);



end Behavioral;
```

- Constraints: constraints.xdc

```
set_property IOSTANDARD LVCMOS33 [get_ports {first_number[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {first_number[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {first_number[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {first_number[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {second_number[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {second_number[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {second_number[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {second_number[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sum[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sum[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sum[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sum[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sum[0]}]
set_property PACKAGE_PIN V17 [get_ports {first_number[0]}]
set_property PACKAGE_PIN V16 [get_ports {first_number[1]}]
set_property PACKAGE_PIN W16 [get_ports {first_number[2]}]
set_property PACKAGE_PIN W17 [get_ports {first_number[3]}]
set_property PACKAGE_PIN V15 [get_ports {second_number[0]}]
set_property PACKAGE_PIN W14 [get_ports {second_number[1]}]
set_property PACKAGE_PIN W13 [get_ports {second_number[2]}]
set_property PACKAGE_PIN V2 [get_ports {second_number[3]}]
set_property PACKAGE_PIN U16 [get_ports {sum[0]}]
set_property PACKAGE_PIN E19 [get_ports {sum[1]}]
set_property PACKAGE_PIN U19 [get_ports {sum[2]}]
set_property PACKAGE_PIN V19 [get_ports {sum[3]}]
set_property PACKAGE_PIN W18 [get_ports {sum[4]}]

set_property IOSTANDARD LVCMOS33 [get_ports {result[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {result[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {result[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {result[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {selection[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {selection[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {selection[0]}]
set_property PACKAGE_PIN R2 [get_ports {selection[2]}]
set_property PACKAGE_PIN T1 [get_ports {selection[1]}]
set_property PACKAGE_PIN U1 [get_ports {selection[0]}]
set_property PACKAGE_PIN U16 [get_ports {result[0]}]
set_property PACKAGE_PIN E19 [get_ports {result[1]}]
set_property PACKAGE_PIN U19 [get_ports {result[2]}]
set_property PACKAGE_PIN V19 [get_ports {result[3]}]
```

7- References:

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