Bilkent University

Electrical and Electronics Department

EE102-01 Lab 4 Report:

“Arithmetic Logic Unit”

23/10/2023

Fatih Mehmet Çetin - 22201689

Purpose:

The purpose of this lab was to understand and implement an arithmetic logic unit (ALU) which is capable of performing at least one bitwise and one shift operation via VHDL. The operations that were chosen in this specific ALU are addition, subtraction, and-gate, logical shift, rotation, inversion, incrementation by one and finally, decrementation by one. Also, several VHDL functions such as entity, port, port map and process were frequently used throughout the experiment.

Methodology:

The initial task was to design an ALU which was able to do 8 different arithmetic operations. After deciding on which operations to use, 3 different design source files were created, which included the VHDL code that describes the chosen operations. For a more organised code structure, the operations were not described in a single design source file but rather they were spread out to three different files (Bit_Adder; Four_Bit_Adder; ALU).

Then, an RTL schematic was created in order to see the connections between the signals more clearly. Then, a testbench file was created to simulate the eight operations and, a waveform output was generated and observed to see if the code was working properly. After that, a constraints file was created and a bitstream for BASYS3 was generated. Finally, the program was uploaded to BASYS3 and was simulated using the switches and LEDs on the FPGA Board.

Design Specifications:

The ALU has two different 4-bit number inputs and a single 4-bit number-output. Also, it has a 3-bit selection input in order to help the user determine which arithmetic operation to use. Basically, what was designed is an 8-to-1 multiplexer. The operations that can be operated in the designed ALU are addition, subtraction, and-gate, logical shift, rotation, inversion, incrementation by one and finally, decrementation by one. The final single bit output of the

ALU is the overflow which indicates whether there has been an error from bit sufficiency or not.

Here you can see the arithmetic operation list sorted by the selection input **(Table 1)**:

Selection Code	Arithmetic Operation
“000”	Addition
“001”	Subtraction
“010”	And-gate
“011”	Logical shift
“100”	Rotation
“101”	Inversion
“110”	Incrementation by one
“111”	Decrementation by one

Table 1: Arithmetic Operation List by Selection Code

The VHDL codes for the design sources are in a modular fashion. “ALU.vhd” is the top module. “Bit_Adder.vhd” and “Four_Bit_Adder.vhd” files are the sub-modules of the top module. “Four_Bit_Adder.vhd” calls “Bit_Adder.vhd” via component and port-map functions and the top module calls “Four_Bit_Adder.vhd” as well.

The operations that require a four-bit adder (addition, subtraction, incrementation, decrementation) will be done outside of the 8-to-1 multiplexer in order to have a more organised code structure. These operations are the Q0, Q1, Q2 and Q3 accordingly in the RTL schematic **(Figure 1)**. Also, bitwise and shift operations (Logical shift, inversion, rotation, incrementation and decrementation) takes only a single 4-bit number as their input. Therefore, the second number does not affect the outcome of those operations.

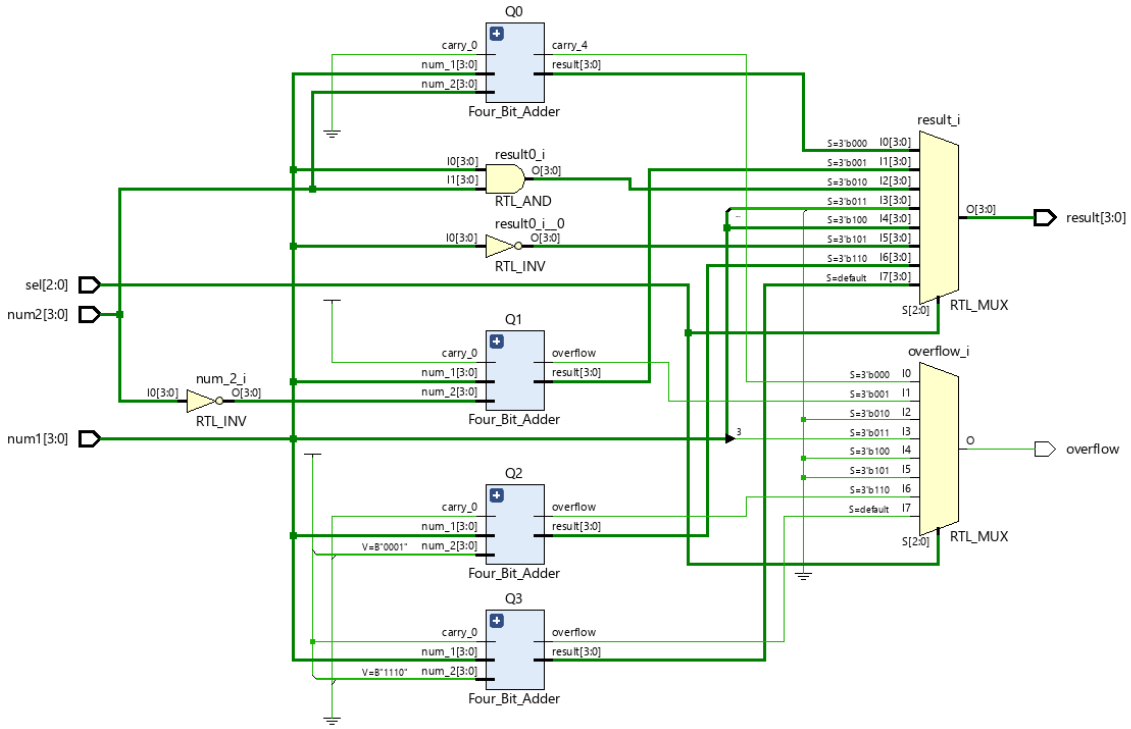

Figure 1: RTL Schematic of ALU.vhd

Results:

After the testbench code was written, the system was ready to be simulated. Here is the simulation of the testbench whose number inputs are “1010” and “1000” **(Figure 2).**

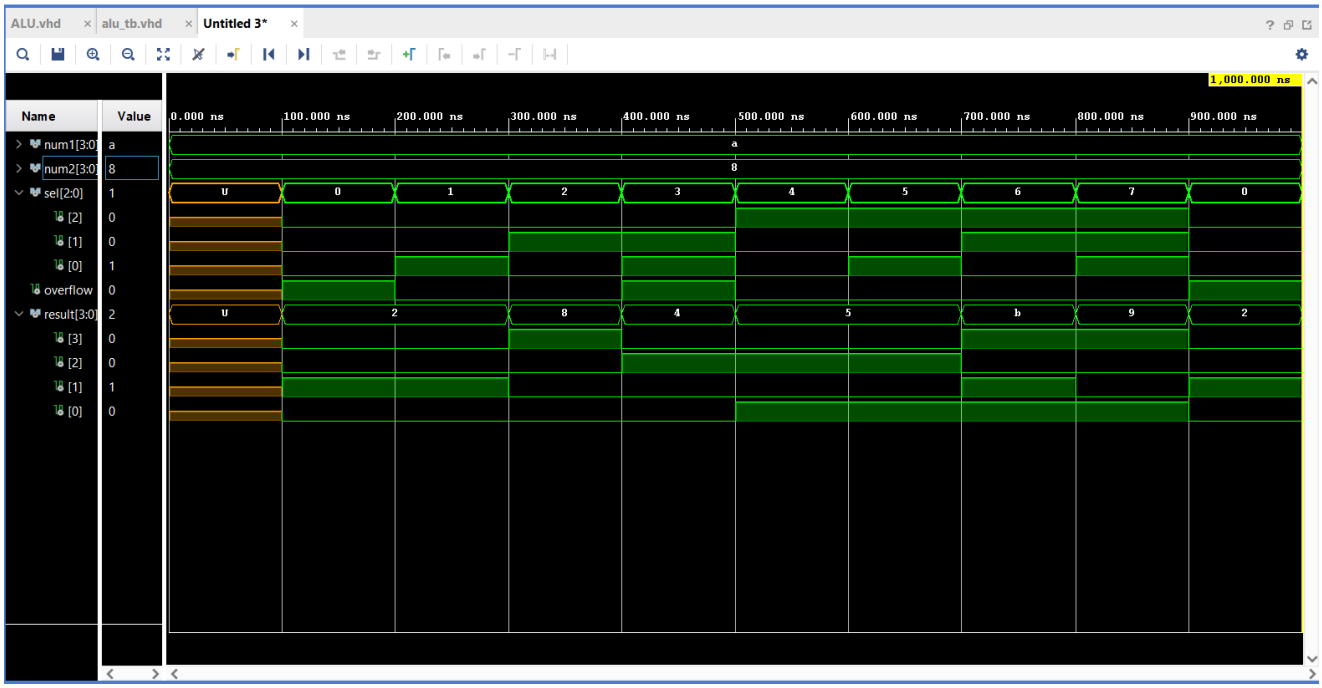

Figure 2: Testbench Simulation of the Program when the number inputs are ”1010” and ”1000”

After completing the simulation successfully, the program was ready to be uploaded to BASYS3. Here you can see the pin configuration **(Figure 3)** of the signals and the experiment results when the two input numbers are “1010” and “1000” **(Figure 4.1 to 4.8)**:

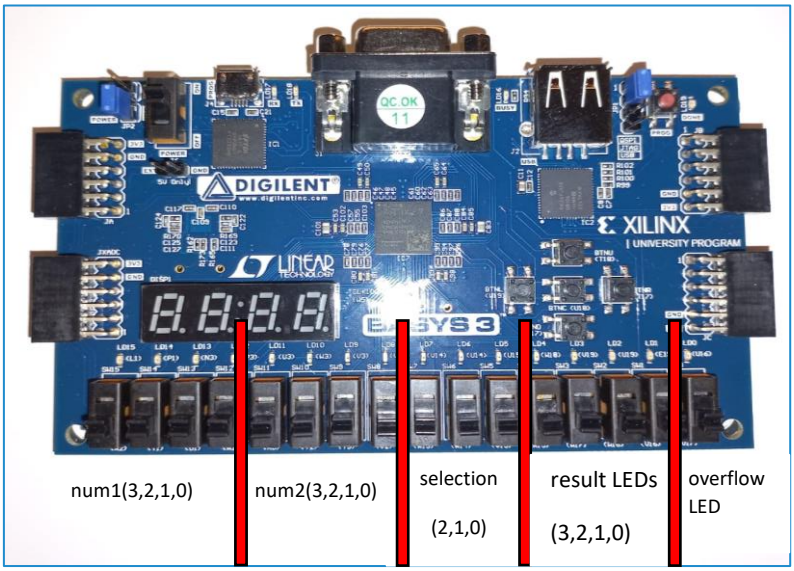

Figure 3: Pin Configuration for BASYS3

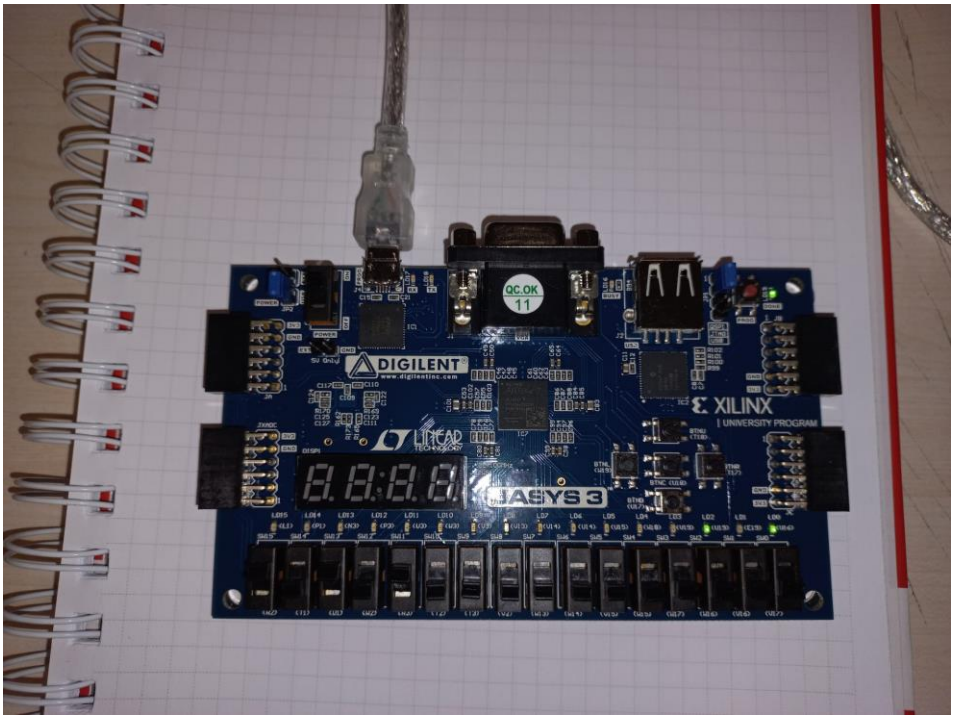

Figure 4.1: num1<=”1010”; num2<=”1000”; sel<=”000”; result<=”0010”; overflow<=”1” --addition--

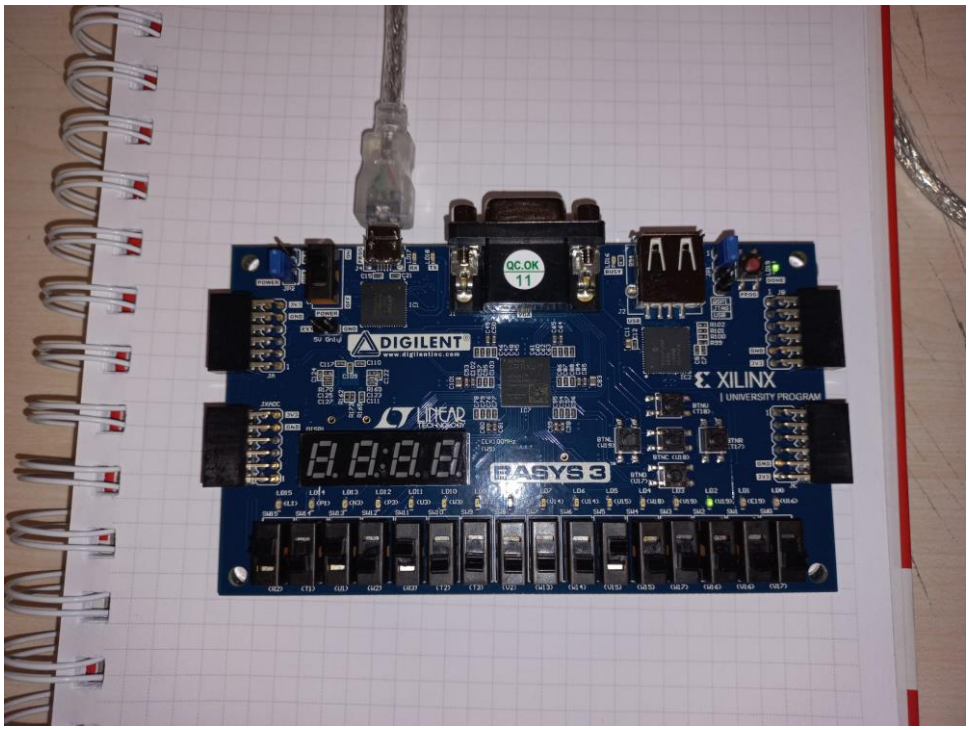

Figure 4.2: num1<=”1010”; num2<=”1000”; sel<=”001”; result<=”0010”; overflow<=”0” --subtraction--

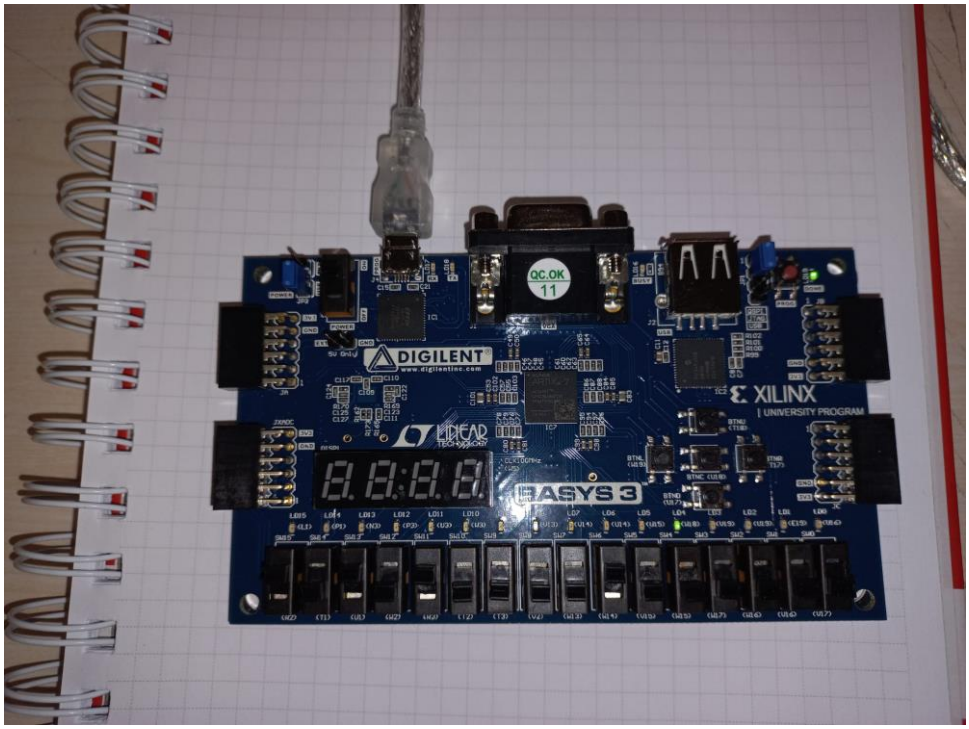

Figure 4.3: num1<=”1010”; num2<=”1000”; sel<=”010”; result<=”1000”; overflow<=”0” --and-gate--

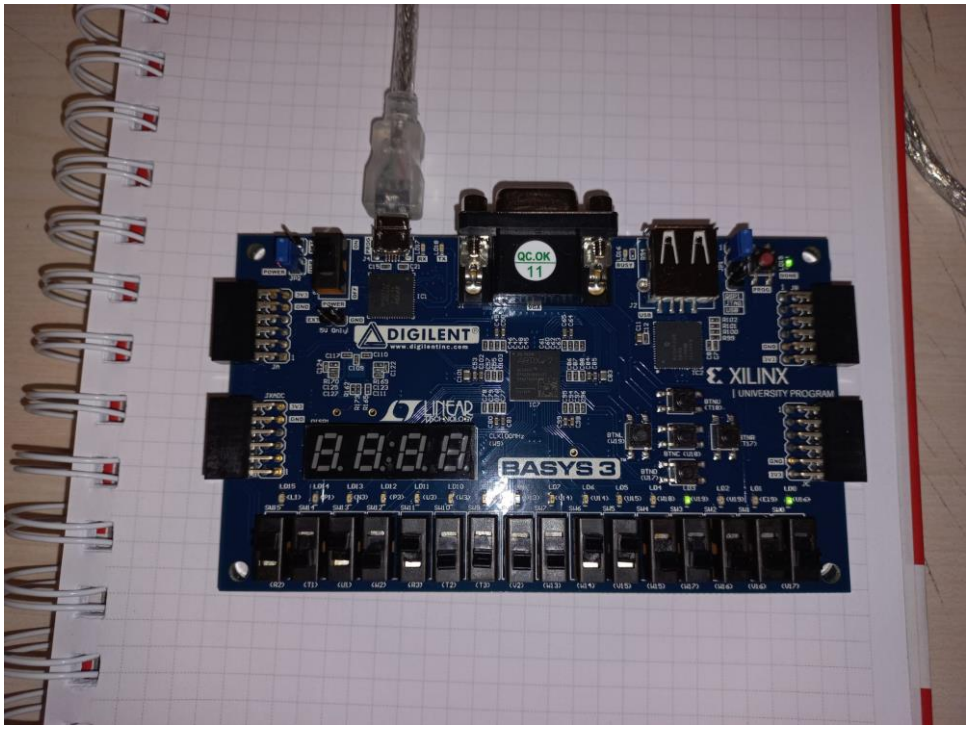

Figure 4.4: num1<=”1010”; num2<=”1000”; sel<=”011”; result<=”0100”; overflow<=”1” --Logical Shift--

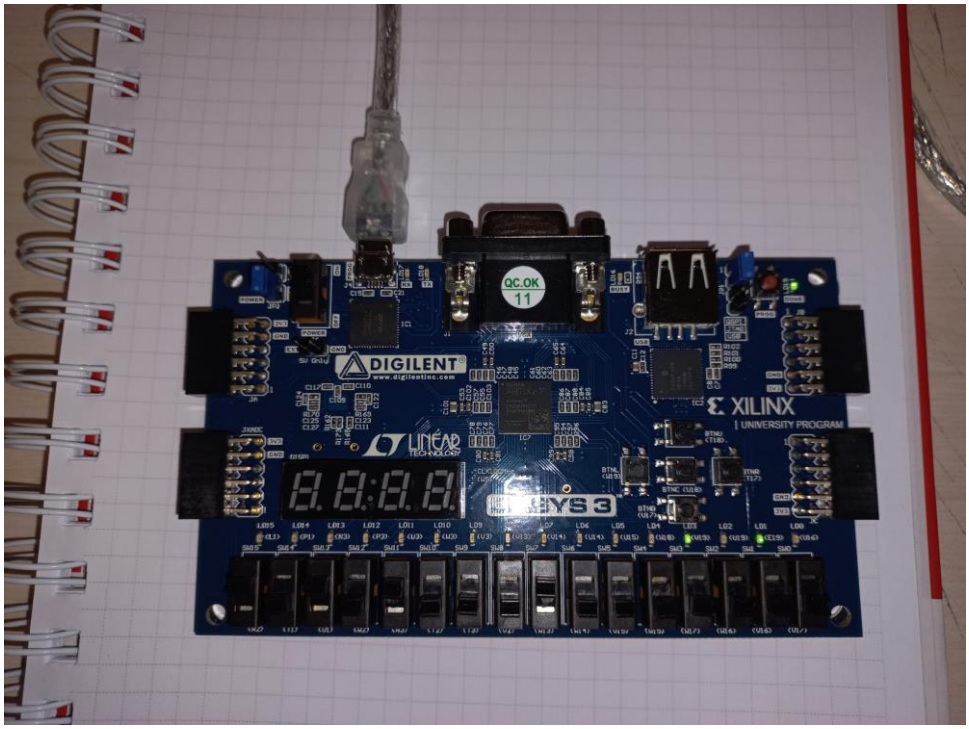

Figure 4.5: num1<=”1010”; num2<=”1000”; sel<=”100”; result<=”0101”; overflow<=”0” --Rotation--

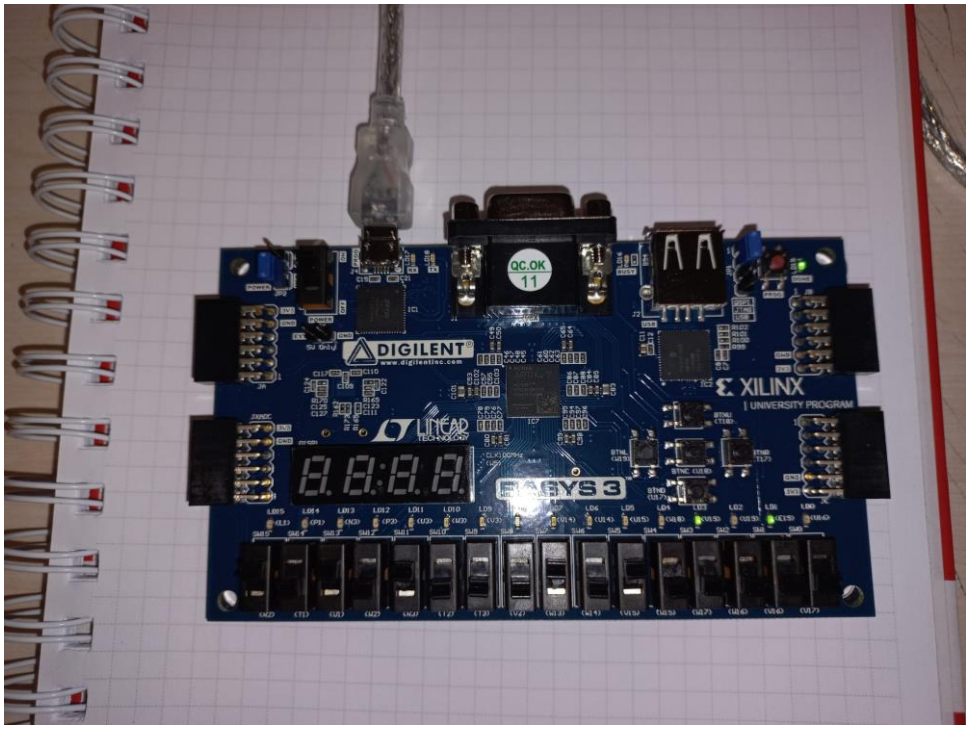

Figure 4.6: num1<=”1010”; num2<=”1000”; sel<=”101”; result<=”0101”; overflow<=”0” --Inversion--

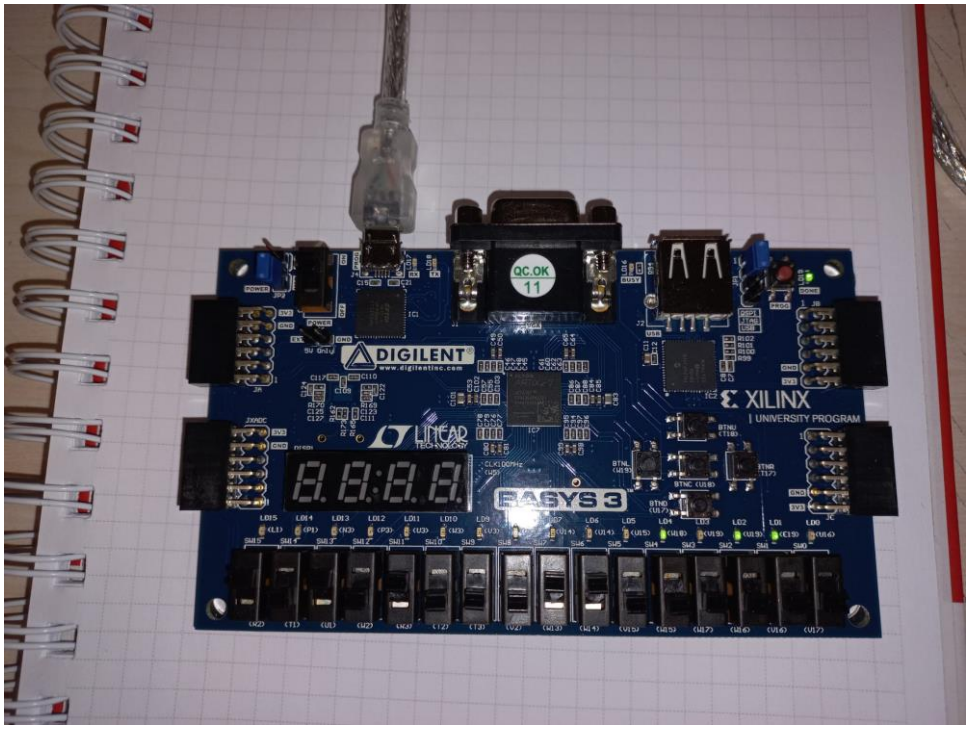

Figure 4.7: num1<=”1010”; num2<=”1000”; sel<=”110”; result<=”1011”; overflow<=”0” --Incrementation--

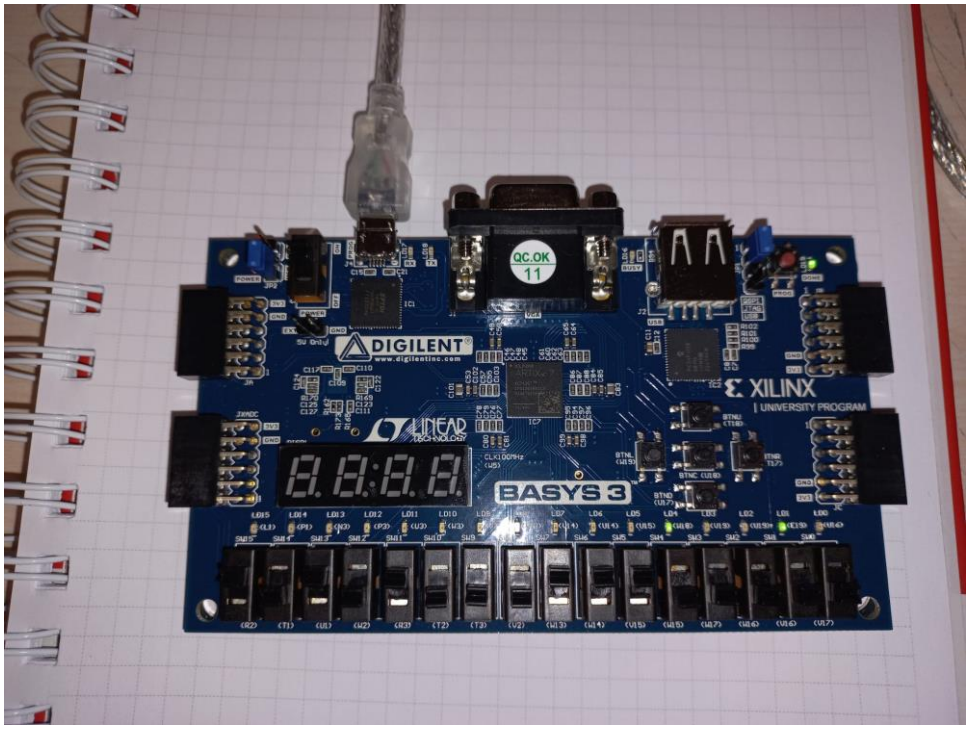

Figure 4.8: num1<=”1010”; num2<=”1000”; sel<=”111”; result<=”1001”; overflow<=”0” --Decrementation--

Conclusion:

The main purpose of the lab was to understand how an ALU works and implement one using several operations. Several VHDL functions such as component, port, port map was used frequently during the experiment. I believe this lab was very crucial in understanding the very basics of Vivado and VHDL. I think my knowledge about FPGA programming went significantly higher after this lab and this will help me a lot during my term project.

Also, one thing I would like to mention is that there was another way of implementing the proposed ALU, and it included create 8 different sub-modules for every single arithmetic operation and calling them to the main module each time the user selects the desired operation. By the observations made in the lab, a significant number of students went down that path. I would like to state that this way of doing the lab has definitely included more room for error and it was certainly a waste of time and energy in my opinion.

Appendices:

RTL Schematics:

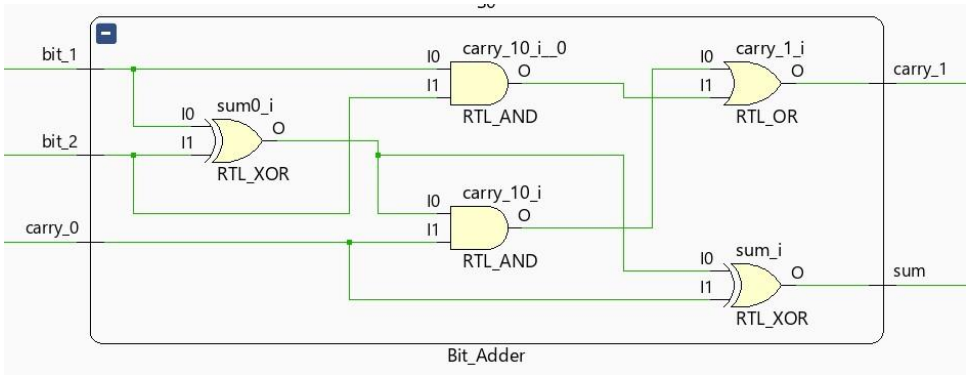

Figure 5: RTL Schematic of Bit_Adder.vhd

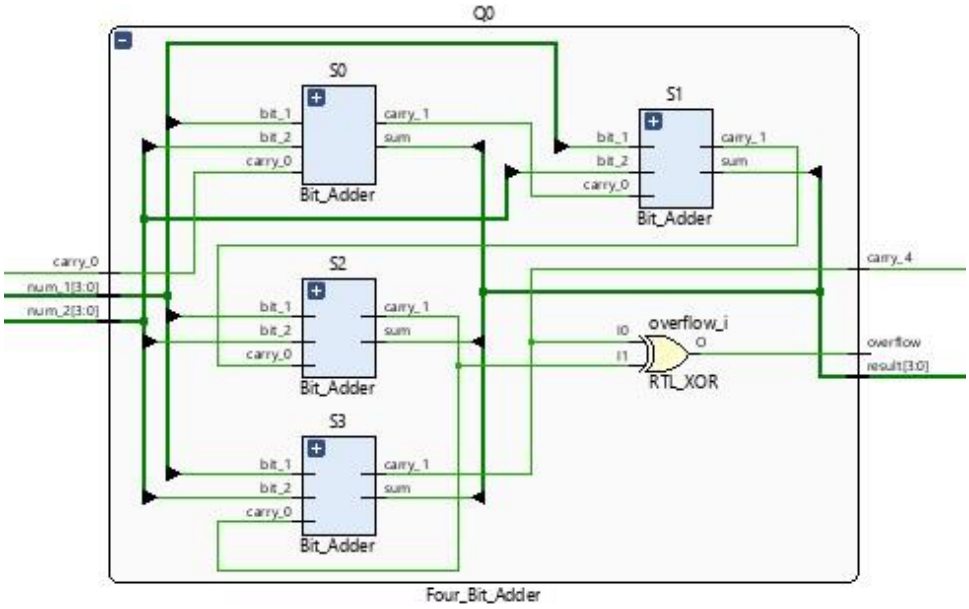

Figure 6: RTL Schematic of Four_Bit_Adder.vhd

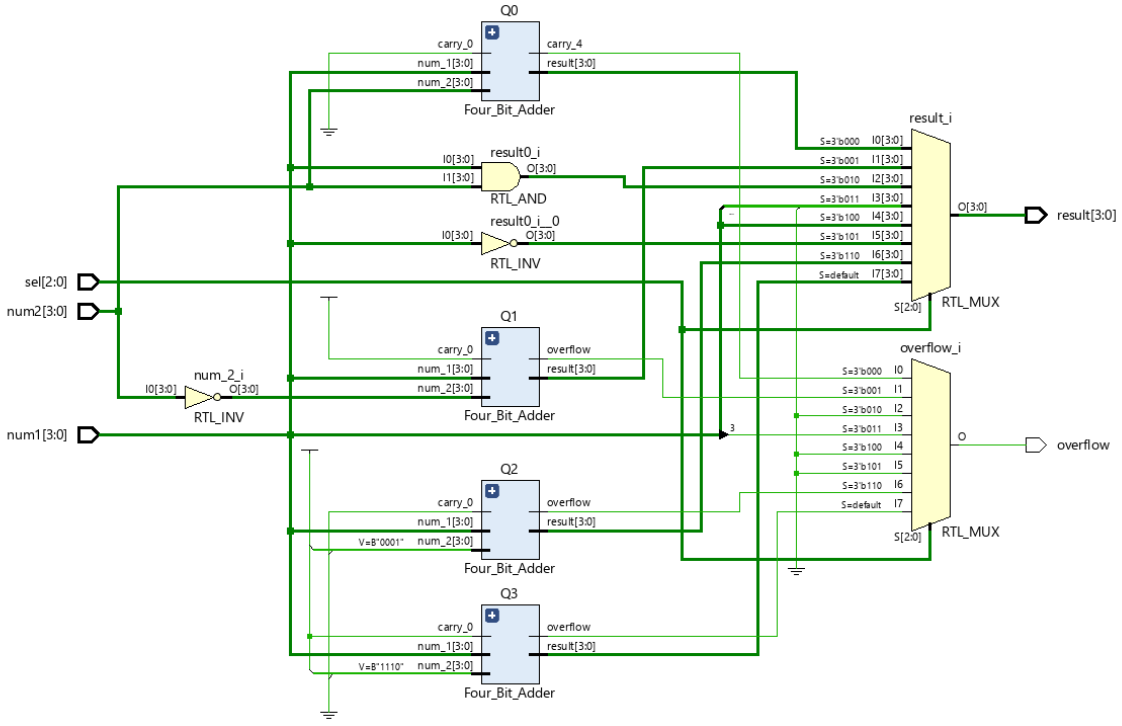

Figure 1: RTL Schematic of ALU.vhd

VHDL CODE:

Bit_Adder.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Bit_Adder is
    Port ( carry_0 : in STD_LOGIC;
           bit_1 : in STD_LOGIC;
           bit_2 : in STD_LOGIC;
           sum : out STD_LOGIC;
           carry_1 : out STD_LOGIC);
end Bit_Adder;

architecture Behavioral of Bit_Adder is

begin
sum <= bit_1 xor bit_2 xor carry_0;
carry_1 <= ((bit_1 xor bit_2) and carry_0) or (bit_1 and bit_2);

end Behavioral;
```

Four_Bit_Adder.vhd

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

library Bit_adder;
use Bit_adder.all;
entity Four_Bit_Adder is
    Port ( carry_0 : in STD_LOGIC;
           num_1 : in STD_LOGIC_VECTOR (3 downto 0);
           num_2 : in STD_LOGIC_VECTOR (3 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0);
```

```
        overflow : out STD_LOGIC;
        carry_4 : inout STD_LOGIC);
end Four_Bit_Adder;

architecture Behavioral of Four_Bit_Adder is

signal carry : STD_LOGIC_VECTOR (3 downto 1);

component Bit_Adder
port(carry_0,bit_1,bit_2 : in STD_LOGIC;
sum, carry_1 : out STD_LOGIC);
end component;

begin

S0: Bit_Adder port map(carry_0 => carry_0, bit_1 => num_1(0),
   bit_2 => num_2(0),sum => result(0),carry_1 => carry(1));
S1: Bit_Adder port map(carry_0 => carry(1), bit_1 => num_1(1),
   bit_2 => num_2(1),sum => result(1),carry_1 => carry(2));
S2: Bit_Adder port map(carry_0 => carry(2), bit_1 => num_1(2),
   bit_2 => num_2(2),sum => result(2),carry_1 => carry(3));
S3: Bit_Adder port map(carry_0 => carry(3), bit_1 => num_1(3),
   bit_2 => num_2(3),sum => result(3),carry_1 => carry_4);

overflow <= carry_4 xor carry(3);

end Behavioral;
```

ALU.vhd

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

```
library Four_Bit_adder;
use Four_Bit_adder.all;
entity ALU is
    Port ( num1 : in STD_LOGIC_VECTOR (3 downto 0);
           num2 : in STD_LOGIC_VECTOR (3 downto 0);
           sel : in STD_LOGIC_VECTOR (2 downto 0);
           result : out STD_LOGIC_VECTOR (3 downto 0);
           overflow : out STD_LOGIC);
end ALU;

architecture Behavioral of ALU is

component Four_Bit_Adder
port(num_1, num_2: in STD_LOGIC_VECTOR (3 downto 0);
carry_0 : in STD_LOGIC;
result : out STD_LOGIC_VECTOR (3 downto 0);
overflow : out STD_LOGIC ;
carry_4 : inout STD_LOGIC);
end component;

signal result1, result2, result3, result4 : STD_LOGIC_VECTOR (3 downto 0);
signal overflow1, overflow2, overflow3, overflow4 : STD_LOGIC;
signal null1, null2,null3,null4 : STD_LOGIC;

begin
Q0 : Four_Bit_Adder port map(num_1 => num1, num_2 => num2, carry_0 => '0',
    result => result1,carry_4 => overflow1, overflow=> null1 );
Q1: Four_Bit_Adder port map(num_1 => num1, num_2 => not(num2), carry_0 => '1',
    result => result2,carry_4 => null2 ,overflow=> overflow2);
Q2 : Four_Bit_Adder port map(num_1 => num1, num_2 => "0001", carry_0 => '0',
    result => result3,carry_4 => null3, overflow =>overflow3);
```

```
Q3 : Four_Bit_Adder port map(num_1 => num1, num_2 => "1110", carry_0 => '1',
   result => result4,carry_4 => null4, overflow => overflow4);

allu:process(num1, num2, sel)is
begin
   case sel is
   when "000" => --Adder--
   result <= result1;
   overflow <= overflow1;

   when "001" => --subtractor--
   result <= result2;
   overflow <= overflow2;

   when "010" => --and-gate--
   result <= num1 and num2;
   overflow <= '0';

   when "011" => --Logical Shift--
   result(3 downto 0) <= num1(2 downto 0) & '0';
   overflow <= num1(3);

   when "100" => -- Four Bit Rotation--
   result(3) <= num1(0);
   result(2) <= num1(1);
   result(1) <= num1(2);
   result(0) <= num1(3);
   overflow <= '0';

   when "101" => --inverse--
   result <= not(num1);
   overflow <= '0';
```

```
    when "110" => --Add 1--
    result <= result3;
    overflow <= overflow3;

    when others => --subtract 1--
    result <= result4;
    overflow <= overflow4;

    end case;
end process;
end Behavioral;
```

Alu_tb.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity alu_tb is
end alu_tb;

architecture Behavioral of alu_tb is

component ALU
port(num1, num2 : in STD_LOGIC_VECTOR (3 downto 0);
sel : in STD_LOGIC_VECTOR (2 downto 0);
result : out STD_LOGIC_VECTOR (3 downto 0);
overflow : out STD_LOGIC);
end component;

signal num1 : STD_LOGIC_VECTOR (3 downto 0);
signal num2 : STD_LOGIC_VECTOR (3 downto 0);
signal sel : STD_LOGIC_VECTOR (2 downto 0);
signal overflow : STD_LOGIC;
```

```
signal result : STD_LOGIC_VECTOR (3 downto 0);

begin
UUT: ALU port map(num1 => num1, num2 => num2, sel => sel, result => result, overflow => overflow );

start: process
begin

num1 <= "1010";
num2 <= "1000";



wait for 100ns;
sel <= "000";

wait for 100ns;
sel <= "001";

wait for 100ns;
sel <= "010";

wait for 100ns;
sel <= "011";

wait for 100ns;
sel <= "100";

wait for 100ns;
sel <= "101";

wait for 100ns;
```

```
sel <= "110";

wait for 100ns;
sel <= "111";

end process;
end Behavioral;
```

Alu_cst.xdc

```
--num1--
set_property PACKAGE_PIN R2 [get_ports {num1[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {num1[3]}]
set_property PACKAGE_PIN T1 [get_ports {num1[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {num1[2]}]
set_property PACKAGE_PIN U1 [get_ports {num1[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {num1[1]}]
set_property PACKAGE_PIN W2 [get_ports {num1[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {num1[0]}]

--num2--
set_property PACKAGE_PIN R3 [get_ports {num2[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {num2[3]}]
set_property PACKAGE_PIN T2 [get_ports {num2[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {num2[2]}]
set_property PACKAGE_PIN T3 [get_ports {num2[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {num2[1]}]
set_property PACKAGE_PIN V2 [get_ports {num2[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {num2[0]}]


--sel--
```

```
set_property PACKAGE_PIN W13 [get_ports {sel[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {sel[2]}]
set_property PACKAGE_PIN W14 [get_ports {sel[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {sel[1]}]
set_property PACKAGE_PIN V15 [get_ports {sel[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {sel[0]}]

--result--
set_property PACKAGE_PIN W18 [get_ports {result[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {result[3]}]
set_property PACKAGE_PIN V19 [get_ports {result[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {result[2]}]
set_property PACKAGE_PIN U19 [get_ports {result[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {result[1]}]
set_property PACKAGE_PIN E19 [get_ports {result[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {result[0]}]

--overflow--
set_property PACKAGE_PIN U16 [get_ports {overflow}]
   set_property IOSTANDARD LVCMOS33 [get_ports {overflow}]
```