Lab 4: Arithmetic Logic Unit

Date: 08.11.2022

Name: Orkun İbrahim Kök

Section: EE102-01

Purpose

This lab aims designing an Arithmetic Logic Unit (ALU) including addition, subtraction and six other operations (at least one bitwise and one shift operation). The overall process can said to be designing a calculator using the Basys3 FPGA.

Design Specifications

I choosed three 3-bit numbers as inputs in my arithmetic logic unit operations (addition, subtraction, multiplication, comparator, orgate, andgate, xnorgate, and bitwise shifting). Also, I used a 8-to-1 MUX in order to choose the alu operation I want. Therefore, it contains 3 select inputs. Due to probability of overflow in some of the operations, I choosed a 4-bit output rather than 3-bit.

Inputs: Input(a) = $a_3a_2a_1$ / Input(b) = $b_3b_2b_1$ / select_inputs(s) = $s_2s_1s_0$

Output: Output(w) = $w_3w_2w_1w_0$

Select Input	Input	Output	Example operation
“000” / Addition	Input(a) & Input(b)	Input(a) + Input(b)	“101” + “100” = “1001”
“001” / Multiplication	Input(a) &Input(b)	Input(a) * Input(b)	“111” * “101” = “0011”
“010” / Subtractor	Input(a) &Input(b)	Input(a) - Input(b)	“101” – “010” = “011”
“011” / Comparator	Input(a) &Input(b)	Input(a) <,=,> Input(b)	“010” & “110” = “0010”
“100” / Bitwise shifter	Input(a)	“0$a_2a_1a_3$”	“010” = “0100”
“101” / Xnor gate	Input(a) &Input(b)	Input(a) xnor Input(b)	“111” XNOR “000” = “0000”
“110” / And gate	Input(a) &Input(b)	Input(a) and Input(b)	“111” AND “111” = “1000”
“111” / Or gate	Input(a) &Input(b)	Input(a) or Input(b)	“000” OR “000” = “0000”

Table 1: ALU Operations

Methodology

I designed an arithmetic operation unit with 8 operations. These operations are:

1. Addition
2. Multiplication
3. Subtraction
4. Comparator
5. Bitwise Shifter

6. XNOR Gate
7. AND Gate
8. Or Gate

For my operations, in order to prohibit overflow causes, I used 4-bit outputs, only subtraction operation has 3-bit output because it is enough for it (Figure 1). While writing the code for calculator operations (addition, multiplication, subtraction), I implemented half adder and full adder sub-modules to maket he operations easy.

Results

After choosing the operations that I wanted to use, I wrote the VHDL codes for corresponding operations. The RTL schematic of ALU including all 8 operations can be seen in Figure 2. Also, each of the RTL schematics for every operation can be found in Appendices part.

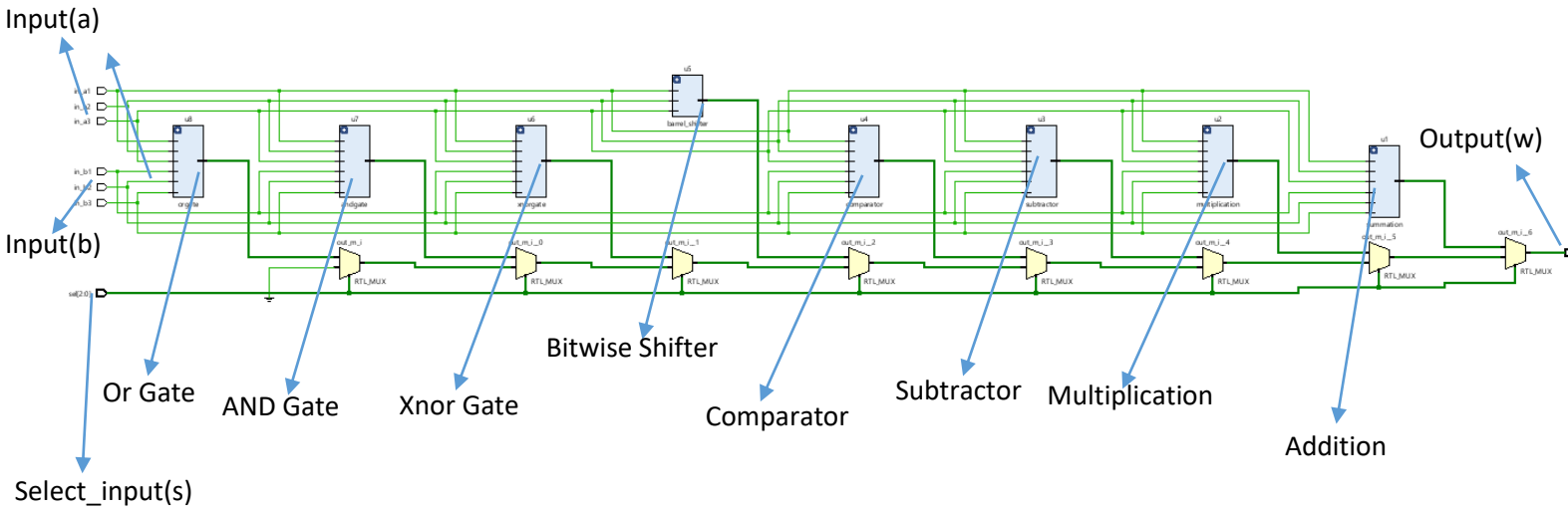

Figure 1: RTL Schematic

As mentioned, these ALU operations are implemented on Basys3 FPGA, Figure 3 shows the corresponding switches and LEDs for input and output values.

Input(a_1): V17 switch

Input(a_2): V16 switch

Input(a_3): W16 switch

Input(b_1): W15 switch

Input(b_2): V15 switch

Input(b_3): W14 switch

Select_input(s_2): R2 switch

Select_input(s_1): T1 switch

Select_input(s_0): U1 switch

Output(w_3): V19 LED

Output(w_2): U19 LED

Output(w_1): E19 LED

Output(w_0): U16 LED

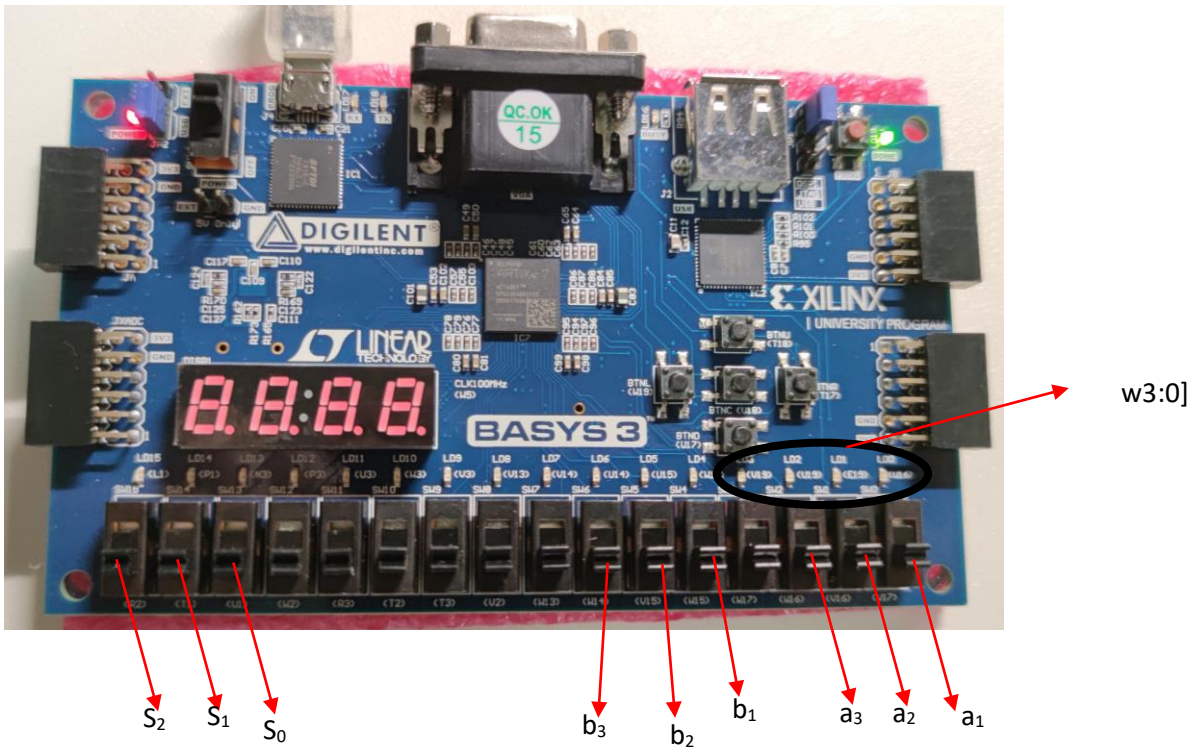

Figure 2: Used input switches and output LEDs on Basys3 FPGA

Because there are so many possibilites while recording all of the combinations for each of the operations, I will only show the combinations where input(a) = “111” and input(b) = 111. (Figure 4)

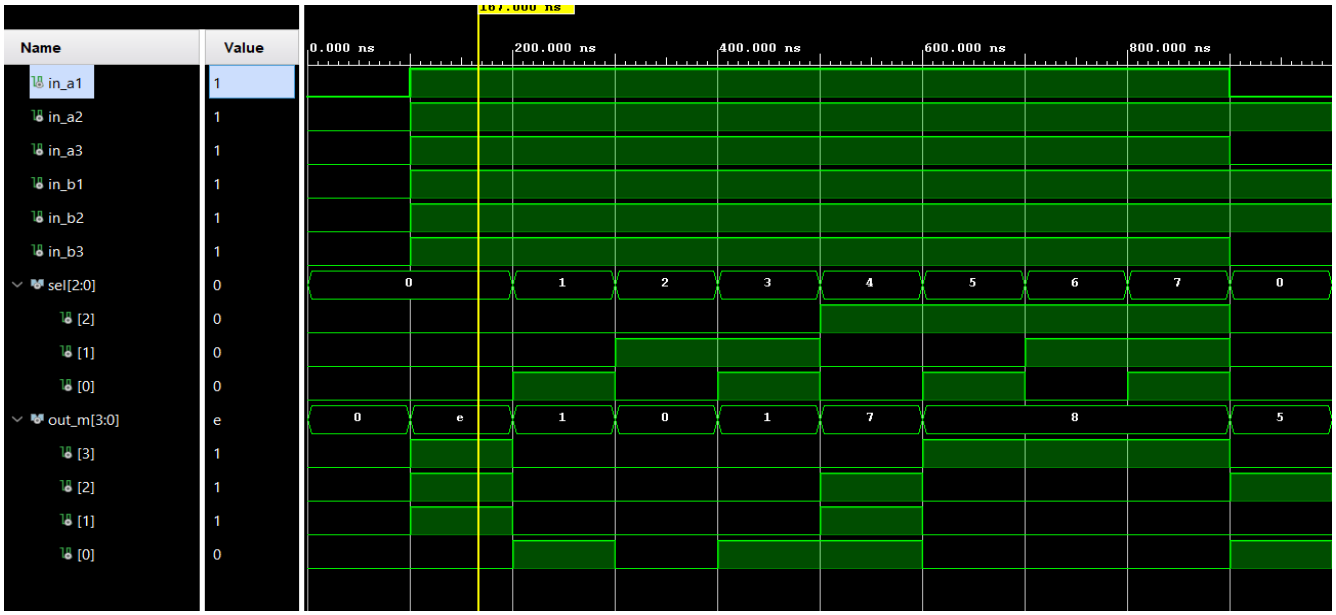

Figure 3: Time diagram and output values for all operations with the input combinations input(a) = “111” and input(b) = “111”

For each operation, I tried some values and recorded the output values, images of the working FPGA can be seen in Appendix part.

Cases

Addition (000 select input):

Input(a) : “110”

Input(b): “011”

Expected Output(w): “1001”

Multiplication (001 select input):

Input(a) : “110”

Input(b): “011”

Expected Output(w): “1001”

Subtraction (010 select input):

Input(a) : “110”

Input(b): “010”

Expected Output(w): “100”

Comparator (011 select input):

Input(a) : “100”

Input(b): “111”

Expected Output(w): “0010”

Bitwise Shifter (100 select input):

Input(a) : “011”

Expected Output(w): “0110”

XNOR gate (101 select input):

Input(a) : “111”

Input(b): “011”

Expected Output(w): “0000”

AND gate (110 select input):

Input(a) : “111”

Input(b): “111”

Expected Output(w): “1000”

OR gate (111 select input):

Input(a) : “000”

Input(b): “000”

Expected Output(w): “0000”

Conclusion

The purpose of the experiment is designing an Arithmetic Logic Unit (ALU) consisting of 8 operations and including addition, subtraction, at least one bitwise and one shift operation. After choosing the eight operations, next step is implementing those operations in Basys3 FPGA by writing the corresponding VHDL codes. For the multiplication part, it is the most difficult operation because it is not taught in the class, so I tried to learn it myself for this lab experiment, however, now I know how to multiply two 3-bit numbers. I tried lots of combinations in the lab and all of the time, the outputs are accurate and correct. Also, using half adders and full adders inside another module seem to be difficult, but I learnt it while doing the experiment, and I learnt the basics of VHDL.

Appendices

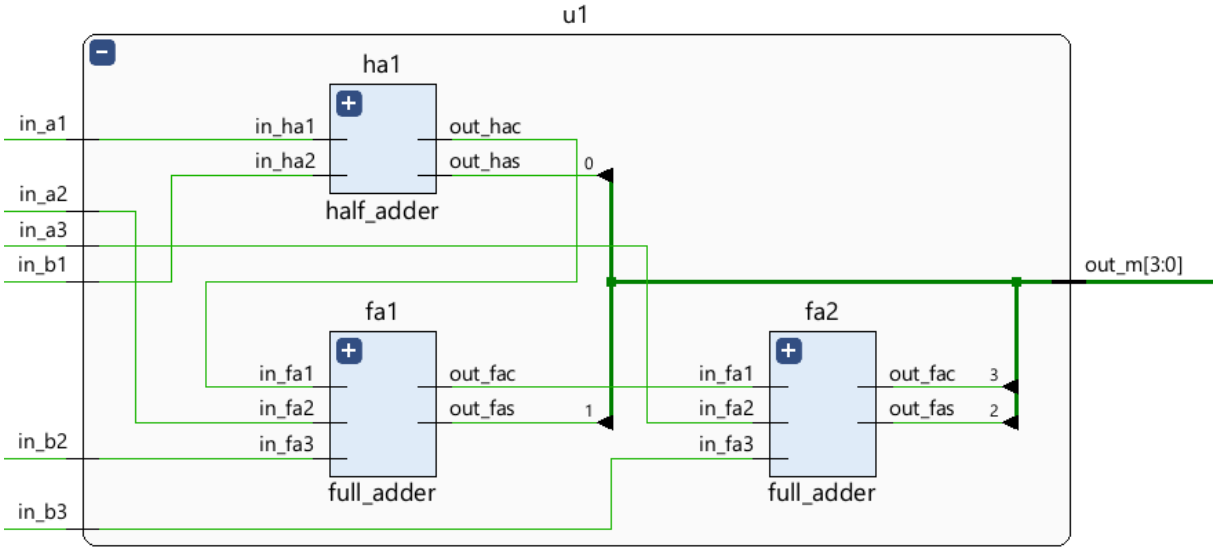

Figure 4.1: Addition Schematic

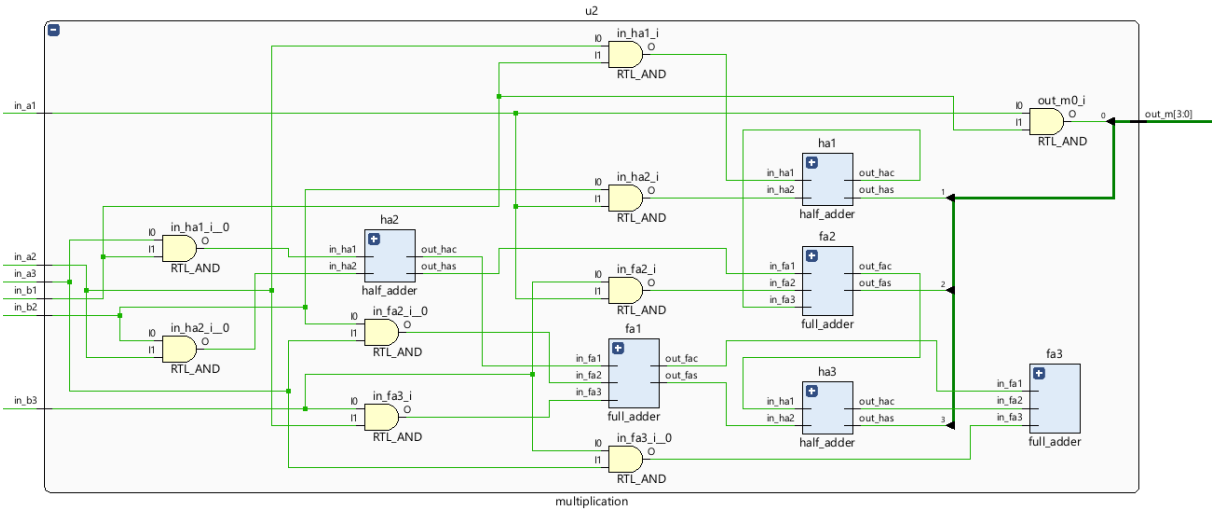

Figure 4.2: Multiplication Schematic

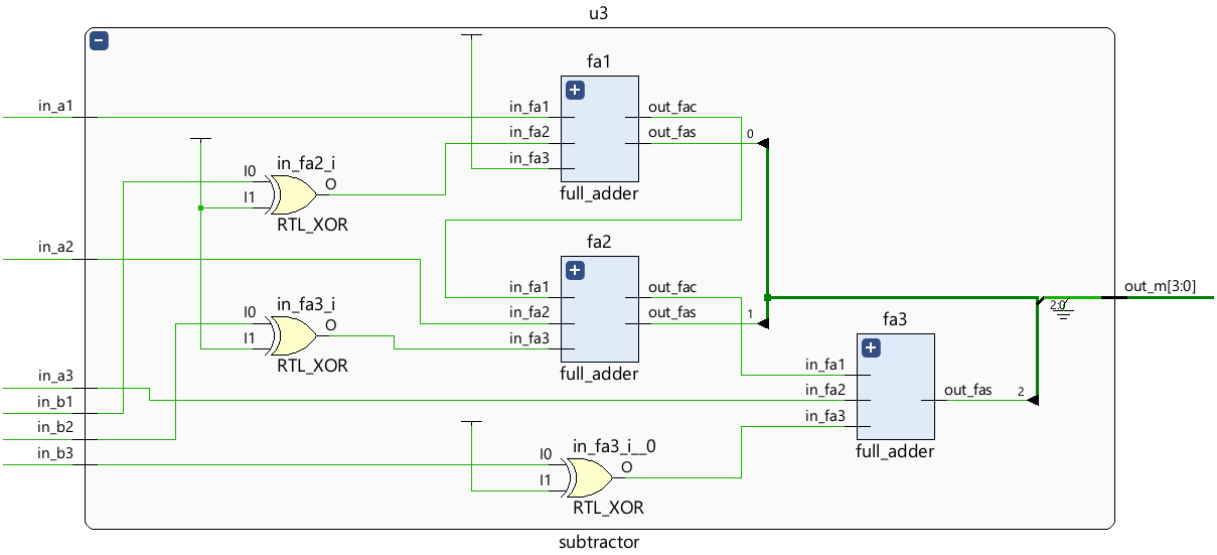

Figure 4.3: Subtractor Schematic

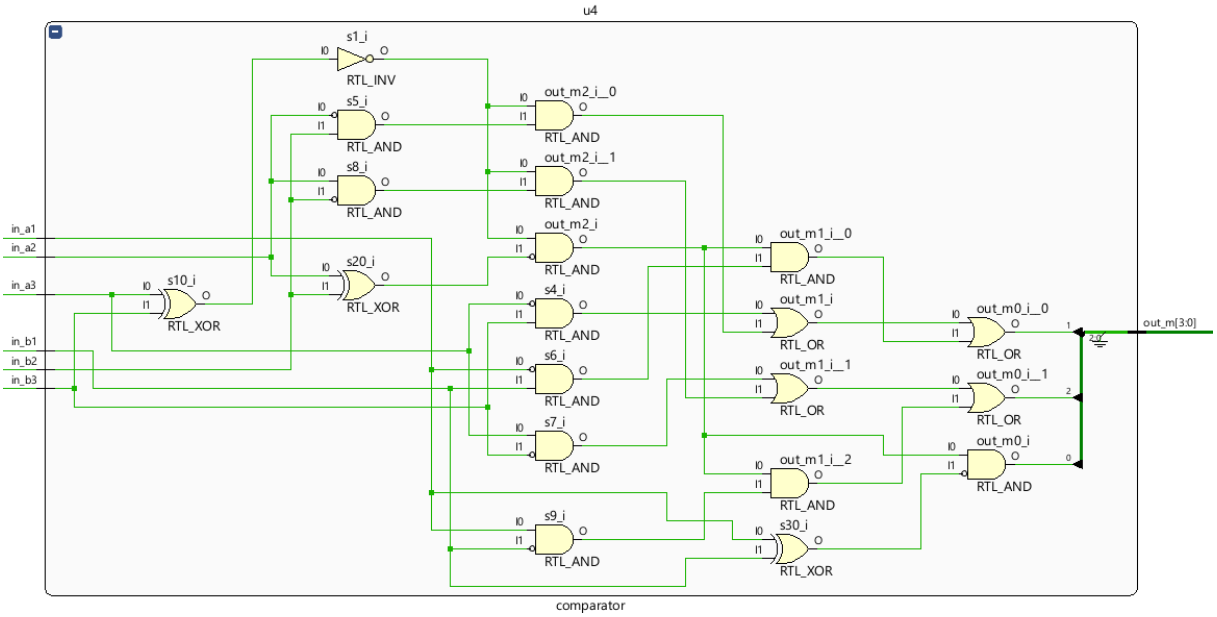

Figure 4.4: Comparator Schematic

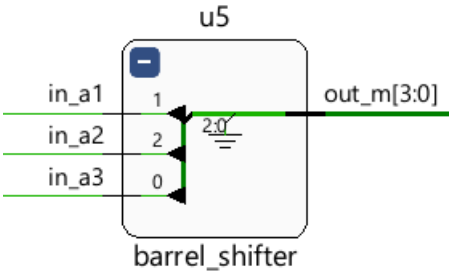

Figure 4.5: Bitwise Shifter Schematic

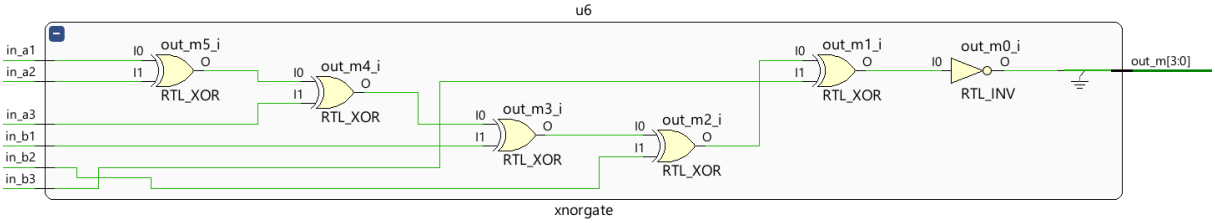

Figure 4.6: XNOR Gate Schematic

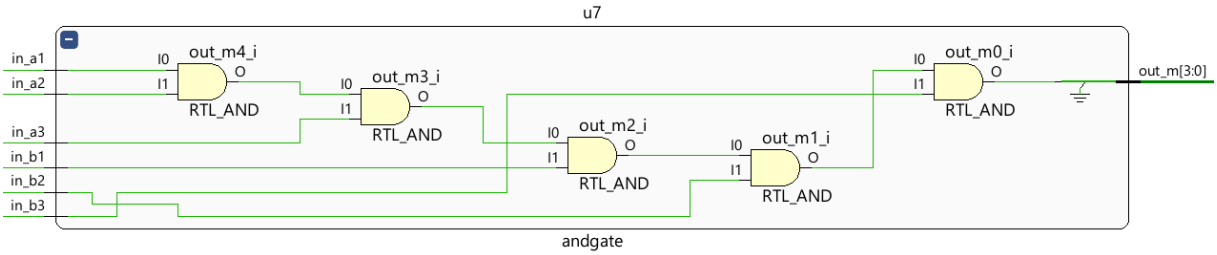

Figure 4.7: AND Gate Schematic

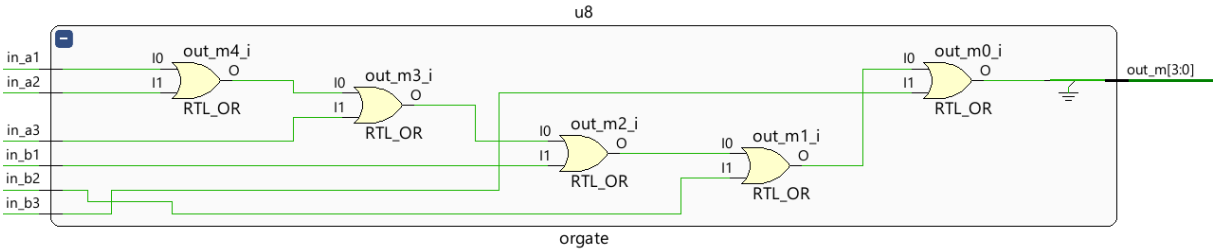

Figure 4.8: OR Gate Schematic

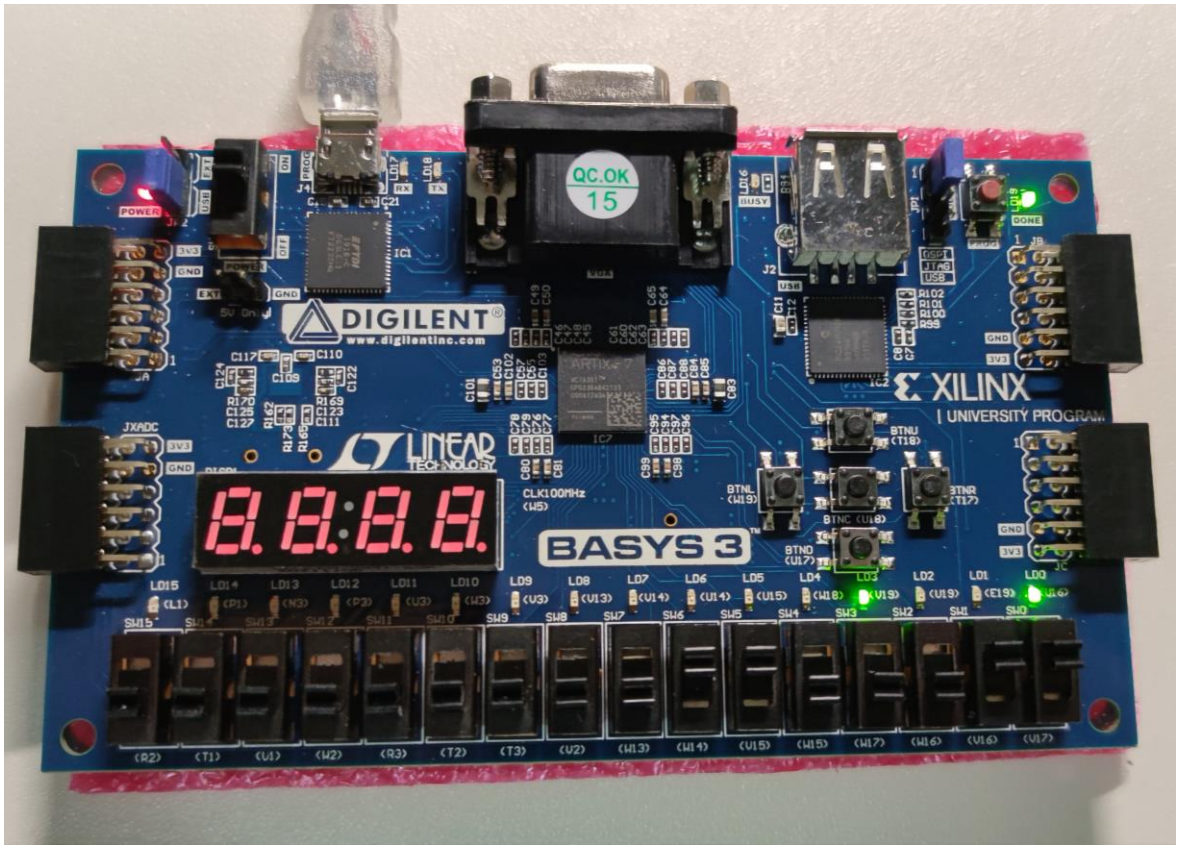

Figure 5.1: Addition operation with input combination input(a) = "011" and input(b) = "001". Output(w) is "1001"

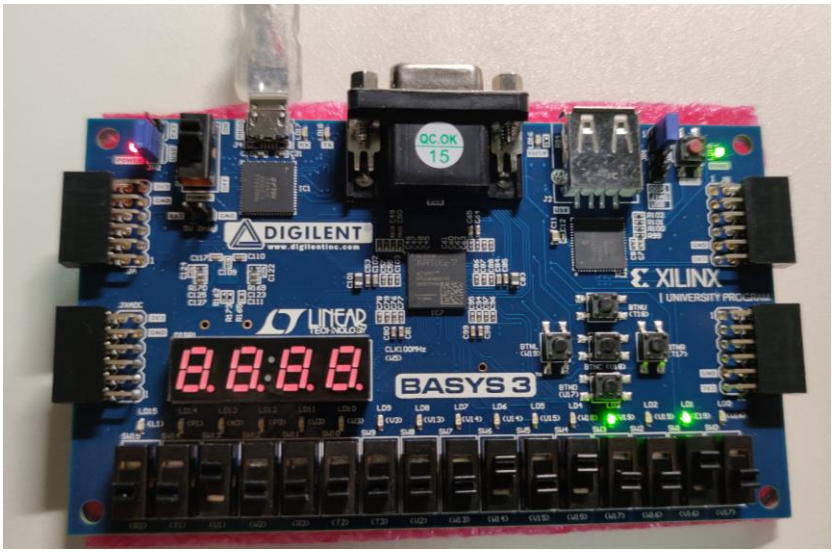

Figure 5.2: Multiplication operation with input combination input(a) = "010" and input(b) = "101". Output(w) is "1010"

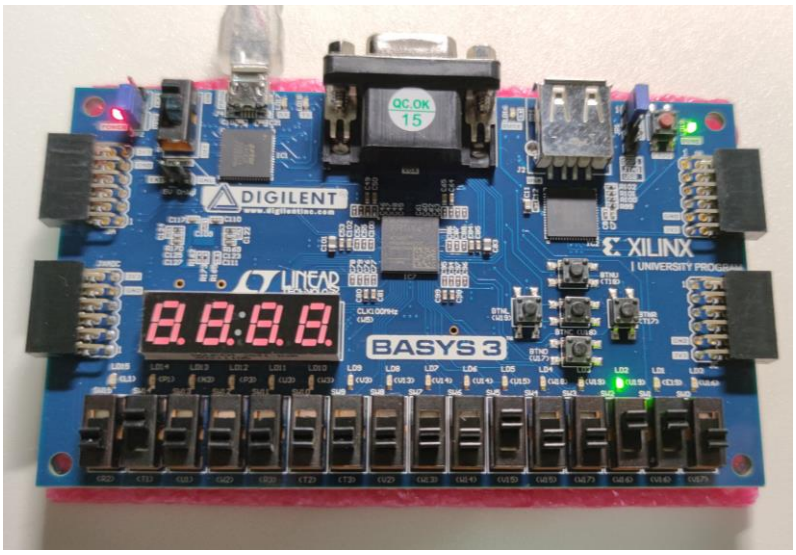

Figure 5.3: Subtraction operation with input combination input(a) = “110” and input(b) = “010”. Output(w) is “100”

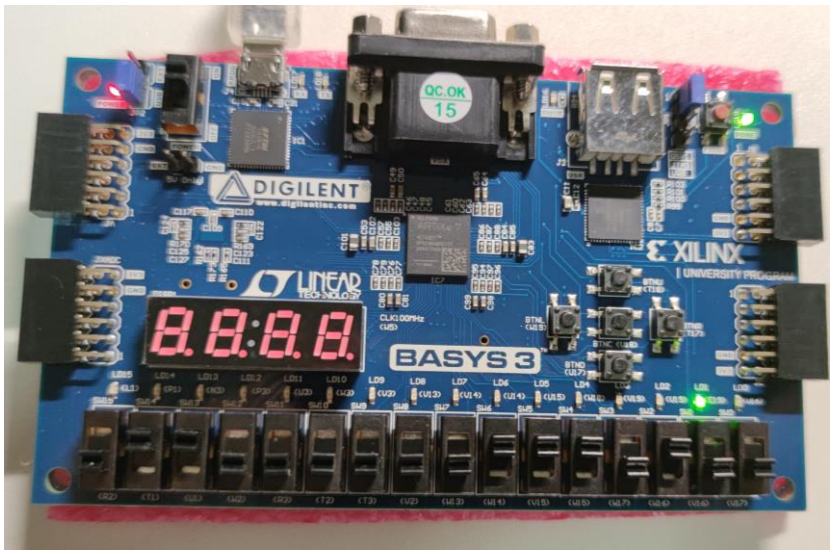

Figure 5.4: Comparator operation with input combination input(a) = “100” and input(b) = “111”. Output(w) is “0010”

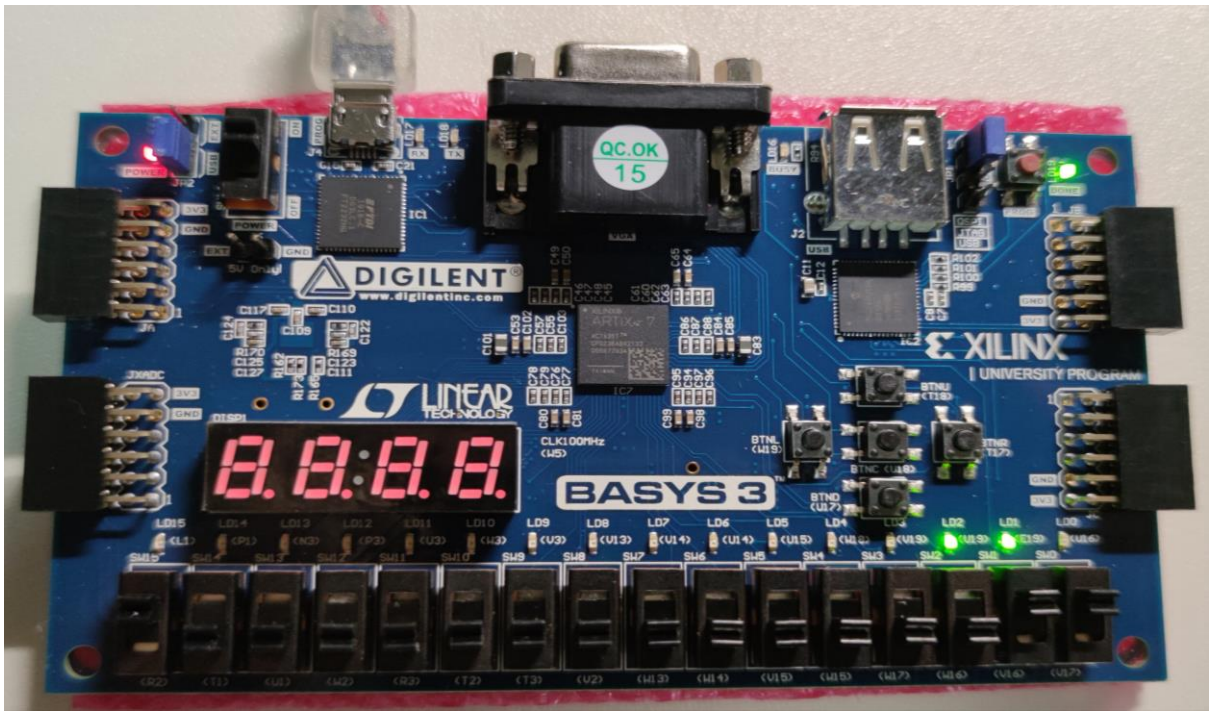

Figure 5.5: Bitwise Shifter operation with input(a) = "011". Output(w) is "0110"

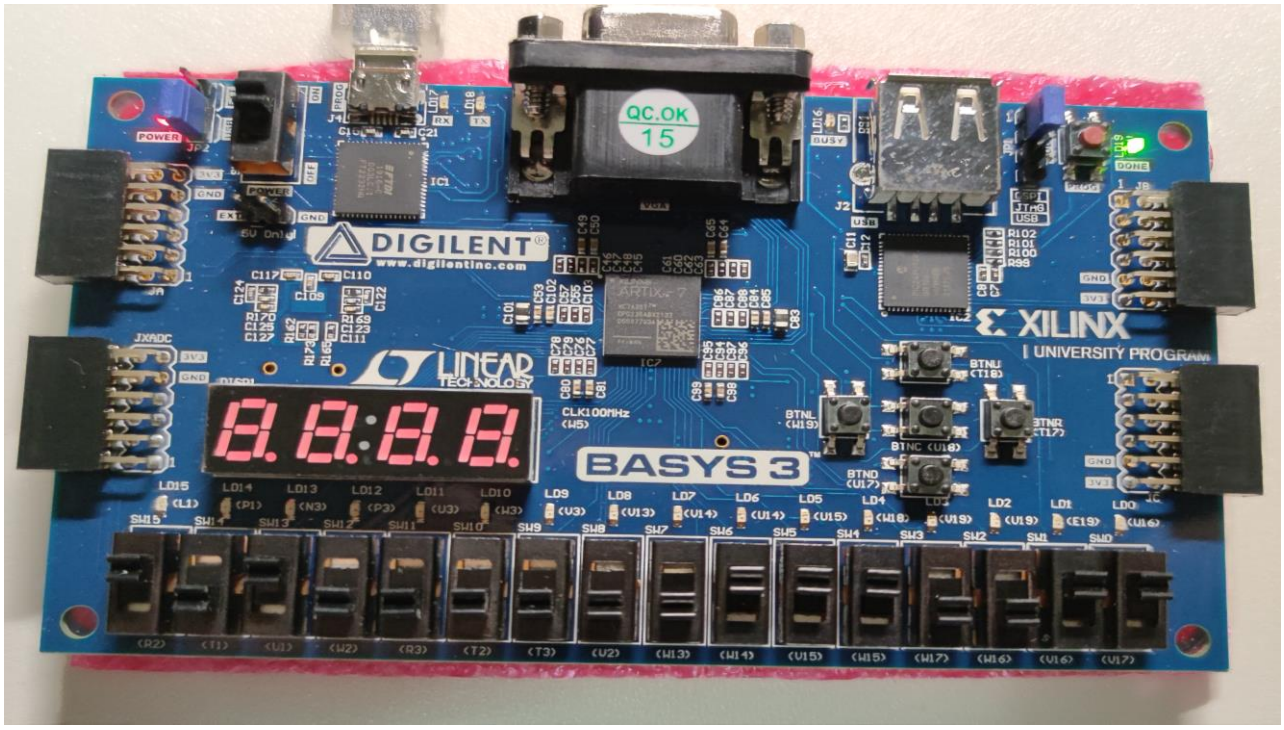

Figure 5.6: XNOR Gate operation with input combination input(a) = "011" and input(b) = "111". Output(w) is "0000"

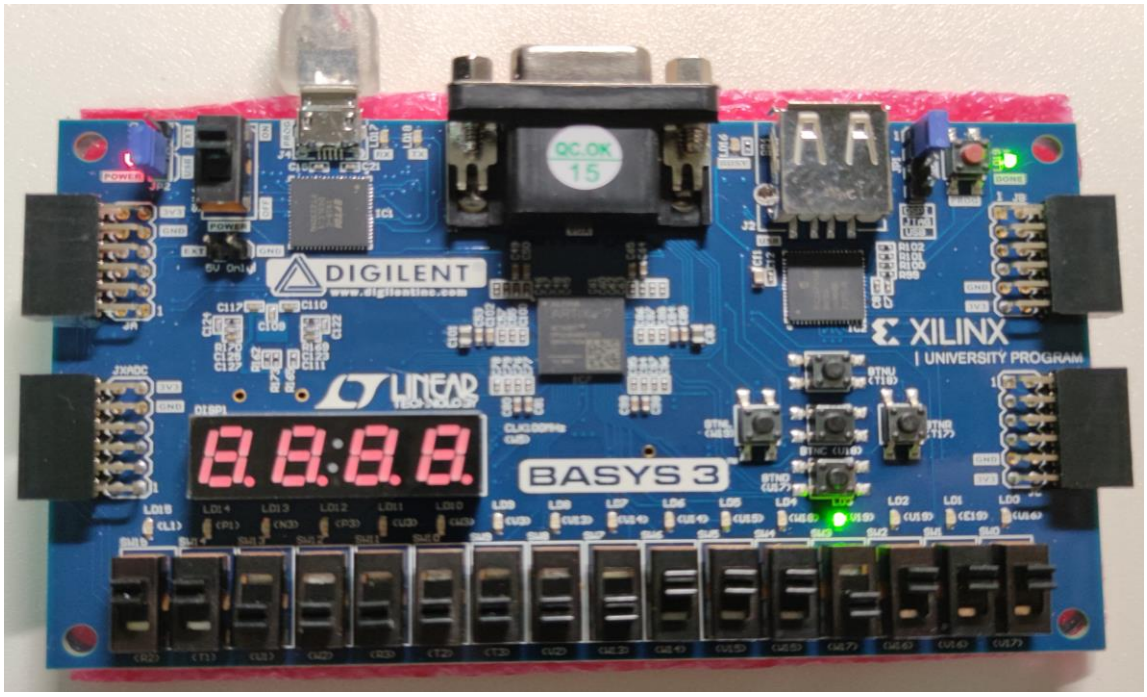

Figure 5.7: AND Gate operation with input combination input(a) = "111" and input(b) = "111". Output(w) is "1000"

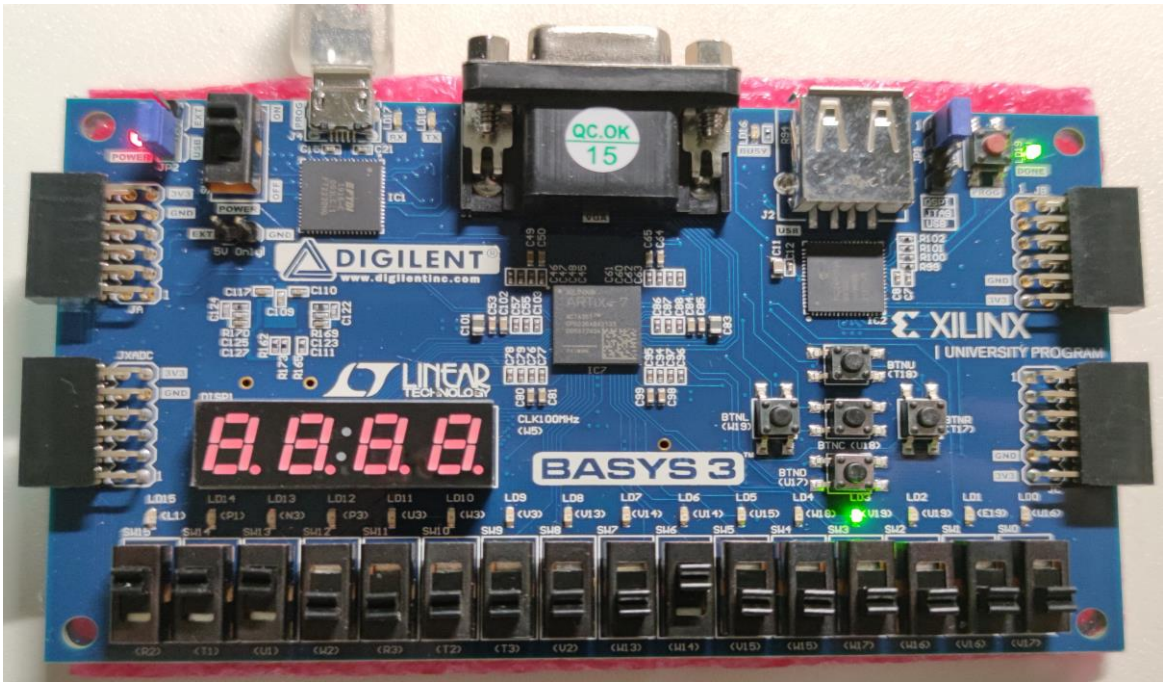

Figure 5.8: OR Gate operation with input combination input(a) = "000" and input(b) = "100". Output(w) is "1000"

alu.vhd

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



entity alu is
    Port(input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in STD_LOGIC;
        select_input: in std_logic_vector(2 downto 0);
        output_w: out std_logic_vector(3 downto 0));
end alu;

architecture Behavioral of alu is
component addition
    Port ( input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in std_logic;
        output_w: out std_logic_vector(3 downto 0));
end component;
component multiplication
        Port ( input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in std_logic;
        output_w: out std_logic_vector(3 downto 0));
end component;
component subtractor
        Port ( input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in std_logic;
        output_w: out std_logic_vector(3 downto 0));
end component;
component comparator
        Port ( input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in std_logic;
        output_w: out std_logic_vector(3 downto 0));
end component;
component bitwise_shifter
```

```
        Port (input_a1,input_a2, input_a3: in std_logic;
        output_w: out STD_LOGIC_vector(3 downto 0));
end component;
component xnorgate
        Port ( input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in std_logic;
        output_w: out std_logic_vector(3 downto 0));
end component;
component andgate
        Port ( input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in std_logic;
        output_w: out std_logic_vector(3 downto 0));
end component;
component orgate
        Port ( input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in std_logic;
        output_w: out std_logic_vector(3 downto 0));
end component;
signal inputa1, inputa2, inputa3, inputb1, inputb2, inputb3: std_logic;
signal output1, output2, output3, output4, output5, output6, output7, output8:
std_logic_vector(3 downto 0);

begin
inputa1 <= input_a1;
inputa2 <= input_a2;
inputa3 <= input_a3;
inputb1 <= input_b1;
inputb2 <= input_b2;
inputb3 <= input_b3;
operation1: addition
port map(input_a1 => inputa1, input_a2 => inputa2, input_a3 => inputa3,
   input_b1 => inputb1,input_b2 => inputb2, input_b3 => inputb3,
   output_w => output1);
```

```
operation2: multiplication
port map(input_a1 => inputa1, input_a2 => inputa2, input_a3 => inputa3,
   inout_b1 => inputb1, input_b2 => inputb2, input_b3 => inputb3,
   output_w => output2);
operation3: subtractor
port map(input_a1 => inputa1, input_a2 => inputa2, input_a3 => inputa3,
   inout_b1 => inputb1, input_b2 => inputb2, input_b3 => inputb3,
   output_w => output3);
operation4: comparator
port map(input_a1 => inputa1, input_a2 => inputa2, input_a3 => inputa3,
   inout_b1 => inputb1, input_b2 => inputb2, input_b3 => inputb3,
   output_w => output4);
operation5: bitwise_shifter
port map(input_a1 => inputa1,input_a2 => inputa2, input_a3 => inputa3,
   output_w => output5);
operation6: xnorgate
port map(input_a1 => inputa1, input_a2 => inputa2, input_a3 => inputa3,
   inout_b1 => inputb1, input_b2 => inputb2, input_b3 => inputb3,
   output_w => output6;
operation7: andgate
port map(input_a1 => inputa1, input_a2 => inputa2, input_a3 => inputa3,
   inout_b1 => inputb1, input_b2 => inputb2, input_b3 => inputb3,
   output_w => output7);
operation8: orgate
port map(input_a1 => inputa1, input_a2 => inputa2, input_a3 => inputa3,
   inout_b1 => inputb1, input_b2 => inputb2, input_b3 => inputb3,
   output_w => output28;
process(select_input, output1, output2, output3, output4, output5, output6, output7,
output8)
begin
```

```
 if select_input = "000" then
    output_w <= output1;
 elsif select_input = "001" then
    output_w <= output2;
 elsif select_input = "010" then
    output_w <= output3;
 elsif select_input = "011" then
    output_w <= output4;
 elsif select_input = "100" then
    output_w <= output5;
 elsif select_input = "101" then
    output_w <= output6;
 elsif select_input = "110" then
    output_w <= output7;
 elsif select_input = "111" then
    output_w <= output8;
 else
    out_m <= "0000";
 end if;
end process;

end Behavioral;
```

addition.vhd

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

entity addition is
    Port ( input_a1,input_a2, input_a3, input_b1, input_b2, input_b3: in STD_LOGIC;
```

```
        output_w: out STD_LOGIC_vector(3 downto 0));
end addition;

architecture Behavioral of addition is
component half_adder
 Port ( input_ha1 : in STD_LOGIC;
    input_ha2 : in STD_LOGIC;
    output_has : out STD_LOGIC;
    output_hac : out STD_LOGIC);
end component;
component full_adder
 Port ( input_fa1 : in STD_LOGIC;
    input_fa2 : in STD_LOGIC;
    input_fa3 : in STD_LOGIC;
    output_fas : out STD_LOGIC;
    output_fac : out STD_LOGIC);
end component;
signal hac,  fac1: std_logic;
begin
    ha1:half_adder
    port map(input_ha1 => input_a1, input_ha2 => input_b1,
        output_has => output_w(0), output_hac => hac);
    fa1:full_adder
 port map(input_fa1 => hac, input_fa2 => input_a2, input_fa3 => input_b2,
    out_fas => output_w(1), output_fac => fac1);
 fa2:full_adder
 port map(input_fa1 => hac, input_fa2 => input_a2, input_fa3 => input_b3,
    out_fas => output_w(2), output_fac => output_w(3));
end Behavioral;
```

multiplication.vhd

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

entity multiplication is
    Port (input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in STD_LOGIC;
        output_w: out STD_LOGIC_vector(3 downto 0));
end multiplication;

architecture Behavioral of multiplication is
component half_adder
 Port ( input_ha1 : in STD_LOGIC;
    input_ha2 : in STD_LOGIC;
    output_has : out STD_LOGIC;
    output_hac : out STD_LOGIC);
end component;
component full_adder
 Port ( input_fa1 : in STD_LOGIC;
    input_fa2 : in STD_LOGIC;
    input_fa3 : in STD_LOGIC;
    output_fas : out STD_LOGIC;
    output_fac : out STD_LOGIC);
end component;
signal sum_1, sum_2: std_logic;
signal carry_1, carry_2, carry_3, carry_4, carry_5: std_logic;
signal dump_2, dump_3: std_logic;
signal select_input: std_logic;
begin
output_w(0) <= input_a1 and input_b1;
```

```
ha1: half_adder port map(input_ha1 => (input_a2 and input_b1), input_ha2 =>( input_b2
and input_a1),
   output_has => output_w(1), output_hac => carry_1);
ha2: half_adder
port map(input_ha1 => (input_a3 and input_b1), input_ha2 =>( input_b2 and input_a2),
   output_has => sum_1, output_hac => carry_2);
fa1: full_adder
port map(input_fa1 => carry_2, input_fa2 =>( input_b2 and input_a3), input_fa3 =>
(input_b3 and input_a2),
   output_fas => sum_2, output_fac => carry_3);
fa2: full_adder
port map(input_fa1 => sum_1, input_fa2 =>( input_b3 and input_a1), input_fa3 => carry_1,
   output_fas => output_w(2), output_fac => carry_4);
ha3: half_adder
port map(input_ha1 => carry_4, input_ha2 =>sum_2,
   output_has => output_w(3), output_hac => carry_5);
fa3: full_adder
port map(input_fa1 => carry_3, input_fa2 =>carry_5, input_fa3 => (input_b3 and input_a3),
   output_fas => dump_2, output_fac => dump_3);


end Behavioral;
```

subtractor.vhd

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


entity subtractor is
   Port ( input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in STD_LOGIC;
      output_w: out STD_LOGIC_vector(3 downto 0));
end subtractor;
```

```
architecture Behavioral of subtractor is
component full_adder
 Port ( input_fa1 : in STD_LOGIC;
    input_fa2 : in STD_LOGIC;
    input_fa3 : in STD_LOGIC;
    output_fas : out STD_LOGIC;
    output_fac : out STD_LOGIC);
end component;
signal dummy,fac_1,fac_2: std_logic;

begin
 fa1:full_adder
 port map(input _fa1 => input_a1, input_fa2 => (input_b1 xor '1'), input_fa3 => '1',
    output_fas => output_w(0), output_fac => fac_1);
 fa2:full_adder
 port map(input_fa1 => fac_1, input_fa2 => input_a2, input_fa3 => (input_b2 xor '1'),
    output_fas => output_w(1), output_fac => fac_2);
 fa3:full_adder
 port map(input_fa1 => fac_2, input_fa2 => input_a3, input_fa3 => (input_b3 xor '1'),
    output_fas => output_w(2), output_fac => dummy);
 output_w(3) <= '0';

end Behavioral;
```

comparator.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity comparator is
```

```
Port ( input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in STD_LOGIC;
   output_w: out STD_LOGIC_vector(3 downto 0));
end comparator;

architecture Behavioral of comparator is
signal s1,s2,s3,s4,s5,s6,s7,s8,s9: std_logic;

begin
s1 <= not(input_a3 xor input_b3);
s2 <= not(input_a2 xor input_b2);
s3 <= not(input_a1 xor input_b1);
s4 <= (not input_a3) and input_b3;
s5 <= (not input_a2) and input_b2;
s6 <= (not input_a1) and input_b1;
s7 <= input_a3 and (not input_b3);
s8 <= input_a2 and (not input_b2);
s9 <= input_a1 and (not input_b1);
output_w(0) <= s1 and s2 and s3; --for a=b
output_w (1) <= s4 or (s1 and s5) or (s1 and s2 and s6); --for a<b
output_w (2) <= s7 or (s1 and s8) or (s1 and s2 and s9); --for a>b
output_w (3) <= '0';
end Behavioral;
```

bitwise_shifter.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity bitwise_shifter is
Port (input_a1, input_a2, input_a3: in STD_LOGIC;
```

```
    output_w: out STD_LOGIC_vector(3 downto 0));
end bitwise_shifter;

architecture Behavioral of bitwise_shifter is
signal bit1, bit2, bit3: std_logic;
begin
bit1<= input_a1;
bit2<= input_a2;
bit3<= input_a3;
output_w <= '0' & bit2 & bit1 & bit3;

end Behavioral;
```

xnorgate.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity xnorgate is
Port ( input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in STD_LOGIC;
    output_w: out STD_LOGIC_vector(3 downto 0):= "0000");

end xnorgate;

architecture Behavioral of xnorgate is

begin
output_w(3) <= (not(input_a1 xor input_a2 xor input_a3 xor input_b1 xor input_b2 xor
input_b3));

end Behavioral;
```

andgate.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity andgate is

Port ( input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in STD_LOGIC;

   output_w: out STD_LOGIC_vector(3 downto 0):= "0000");

end andgate;


architecture Behavioral of andgate is


begin

output_w(3) <= input_a1 and input_a2 and input_a3 and input_b1 and input_b2 and
input_b3;


end Behavioral;
```

orgate.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity orgate is

Port ( input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in STD_LOGIC;

   output_w: out STD_LOGIC_vector(3 downto 0):= "0000");

end orgate;


architecture Behavioral of orgate is


begin

output_w(3) <= input_a1 or input_a2 or input_a3 or input_b1 or input_b2 or input_b3;


end Behavioral;
```

testbench1.vhd

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

entity testbench1 is
--  Port ( );
end testbench1;

architecture Behavioral of testbench1 is
component alu
   Port (input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: in STD_LOGIC;
      Select_input: in std_logic_vector(2 downto 0);
      output_w: out STD_LOGIC_vector(3 downto 0));
end component;
   signal input_a1, input_a2, input_a3, input_b1, input_b2, input_b3: std_logic;
   signal select_input: std_logic_vector(2 downto 0);
   signal output_w: std_logic_vector(3 downto 0);
begin
   operation1: alu
   Port map(input_a1 => input_a1, input_a2 => input_a2, input_a3 => input_a3, input_b1 =>
input b1, input_b2 =>
   input_b2, input_b3 => input_b3, select_input => select_input, output_w =>output_w);
testbench1: process
begin
   select_input <= "000";
   input_a1 <= '0';
   input _a2 <= '0';
   input _a3 <= '0';
   input _b1 <= '0';
```

```
    input _b2 <= '0';
    input _b3 <= '0';
  wait for 100ns;
    select_input <= "000";
    input _a1 <= '1';
    input _a2 <= '1';
    input _a3 <= '1';
    input _b1 <= '1';
    input _b2 <= '1';
    input _b3 <= '1';
  wait for 100ns;
    select_input <= "001";
    input_a1 <= '1';
    input_a2 <= '1';
    input_a3 <= '1';
    input_b1 <= '1';
    input_b2 <= '1';
    input_b3 <= '1';
  wait for 100ns;
    select_input <= "010";
    input_a1 <= '1';
    input_a2 <= '1';
    input_a3 <= '1';
    input_b1 <= '1';
    input_b2 <= '1';
    input_b3 <= '1';
  wait for 100ns;
    select_input <= "011";
    input_a1 <= '1';
```

```
    input_a2 <= '1';
    input_a3 <= '1';
    input_b1 <= '1';
    input_b2 <= '1';
    input_b3 <= '1';
  wait for 100ns;
    select_input <= "100";
    input_a1 <= '1';
    input_a2 <= '1';
    input_a3 <= '1';
    input_b1 <= '1';
    input_b2 <= '1';
    input_b3 <= '1';
  wait for 100ns;
    select_input <= "101";
    input_a1 <= '1';
    input_a2 <= '1';
    input_a3 <= '1';
    input_b1 <= '1';
    input_b2 <= '1';
    input_b3 <= '1';
  wait for 100ns;
    select_input <= "110";
    input_a1 <= '1';
    input_a2 <= '1';
    input_a3 <= '1';
    input_b1 <= '1';
    input_b2 <= '1';
    input_b3 <= '1';
```

```
 wait for 100ns;
   select_input <= "111";
   input_a1 <= '1';
   input_a2 <= '1';
   input_a3 <= '1';
   input_b1 <= '1';
   input_b2 <= '1';
   input_b3 <= '1';
-------------------------------------------------
 wait for 100ns;
   select_input <= "000";
   input_a1 <= '0';
   input_a2 <= '1';
   input_a3 <= '0';
   input_b1 <= '1';
   input_b2 <= '1';
   input_b3 <= '0';
 wait for 100ns;
   select_input <= "001";
   input_a1 <= '0';
   input_a2 <= '1';
   input_a3 <= '0';
   input_b1 <= '1';
   input_b2 <= '1';
   input_b3 <= '0';
 wait for 100ns;
   select_input <= "010";
   input_a1 <= '0';
   input_a2 <= '1';
```

```
    input_a3 <= '0';
    input_b1 <= '1';
    input_b2 <= '1';
    input_b3 <= '0';
  wait for 100ns;
    select_input <= "011";
    input_a1 <= '0';
    input_a2 <= '1';
    input_a3 <= '0';
    input_b1 <= '1';
    input_b2 <= '1';
    input_b3 <= '0';
  wait for 100ns;
    select_input <= "100";
    input_a1 <= '0';
    input_a2 <= '1';
    input_a3 <= '0';
    input_b1 <= '1';
    input_b2 <= '1';
    input_b3 <= '0';

  wait for 100ns;
    select_input <= "101";
    input_a1 <= '0';
    input_a2 <= '1';
    input_a3 <= '0';
    input_b1 <= '1';
    input_b2 <= '1';
    input_b3 <= '0';
```

```
    wait for 100ns;
    select_input <= "110";
    input_a1 <= '0';
    input_a2 <= '1';
    input_a3 <= '0';
    input_b1 <= '1';
    input_b2 <= '1';
    input_b3 <= '0';
 wait for 100ns;
    select_input <= "111";
    input_a1 <= '0';
    input_a2 <= '1';
    input_a3 <= '0';
    input_b1 <= '1';
    input_b2 <= '1';
    input_b3 <= '0';
-------------------------------------------------
--time diagram inputs
 wait for 100ns;
    select_input <= "000";
    input_a1 <= '1';
    input_a2 <= '1';
    input_a3 <= '0';
    input_b1 <= '1';
    input_b2 <= '0';
    input_b3 <= '1';
 wait for 100ns;
    select_input <= "001";
    input_a1 <= '1';
```

```
    input_a2 <= '1';
    input_a3 <= '0';
    input_b1 <= '1';
    input_b2 <= '0';
    input_b3 <= '1';
  wait for 100ns;
    select_input <= "010";
    input_a1 <= '1';
    input_a2 <= '1';
    input_a3 <= '0';
    input_b1 <= '1';
    input_b2 <= '0';
    input_b3 <= '1';
  wait for 100ns;
    select_input <= "011";
    input_a1 <= '1';
    input_a2 <= '1';
    input_a3 <= '0';
    input_b1 <= '1';
    input_b2 <= '0';
    input_b3 <= '1';
  wait for 100ns;
    select_input <= "100";
    input_a1 <= '1';
    input_a2 <= '1';
    input_a3 <= '0';
    input_b1 <= '1';
    input_b2 <= '0';
    input_b3 <= '1';
```

```
 wait for 100ns;
    select_input <= "101";
    input_a1 <= '1';
    input_a2 <= '1';
    input_a3 <= '0';
    input_b1 <= '1';
    input_b2 <= '0';
    input_b3 <= '1';
 wait for 100ns;
    select_input <= "110";
    input_a1 <= '1';
    input_a2 <= '1';
    input_a3 <= '0';
    input_b1 <= '1';
    input_b2 <= '0';
    input_b3 <= '1';
 wait for 100ns;
    select_input <= "111";
    input_a1 <= '1';
    input_a2 <= '1';
    input_a3 <= '0';
    input_b1 <= '1';
    input_b2 <= '0';
    input_b3 <= '1';
 wait for 100ns;
end process;
end Behavioral;
```