Eray Gündoğdu 22101962 7.11.2022

EEE102 LAB04: Arithmetic Logic Unit Design

Purpose:

Aim of this experiment is to get familiar with bit shift operations , bitwise logic operations,arithmetic operations and designing an arithmetic logic unit.

Methadology:

In this experiment,arithmetic logic unit is design to make addition,substraction,AND gate,OR gate,NAND gate,inversion of inputs and right shift operations.There are 2 inputs assigned as “A” and “B”.These inputs are 4 bit inputs.In addition there is a 4 bit output assigned as “Y”,a 3 bit select input assigned as “SEL” and a overflow output for the addition and substraction operations.Select input determins which operation is going to be done.SEL is matched with operations as given below.These inputs belong to the ALU code which is test1 in this experiment.Testbench code is written for running a simulation and values of A and B are given in the testbench such as “1001” .For the addition operation,four bit adder is designed by using four one bit adder.Substraction is done by inverting two variables “B” and “c_in”.This process is done in the test1 file.Shifting operation is done by adding A(0) and A with “&”.The operations are complete.In constraints 1 file,A(0),A(1),A(2),A(3) is assigned to pins W2,U1,T1,R2 respectively. B(0),B(1),B(2),B(3) is assigned to the pins V17,V16,W16,W17 respectively.SEL(0),SEL(1),SEL(2) is assigned to the pins V2,T3,T2.Y(0),Y(1),Y(2),Y(3) is assigned to the pins U16,E19,U19,V19 respectively. After completing constraints file,ALU is checked wheter it is working correct or not by looking at simulation.After the checking process,code is implemented BASYS3.

If SEL is “000” addition is done.

If SEL is “001” substraction is done.

If SEL is “010” AND gate is done.

If SEL is “011” OR gate is done.

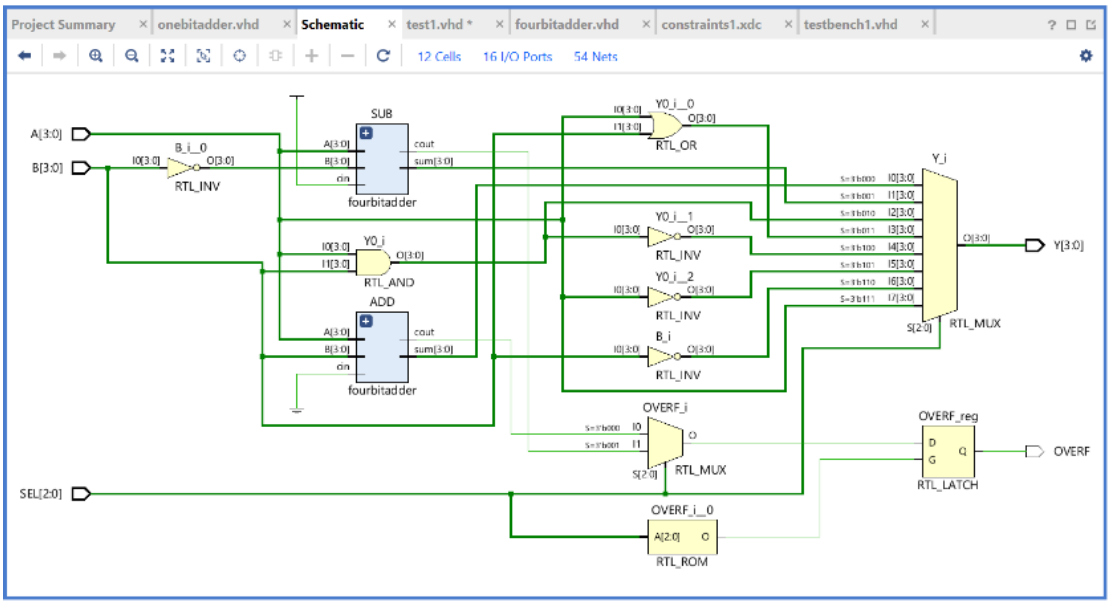
If SEL is “100” NAND is done.

If SEL is “101” NOT A is done.

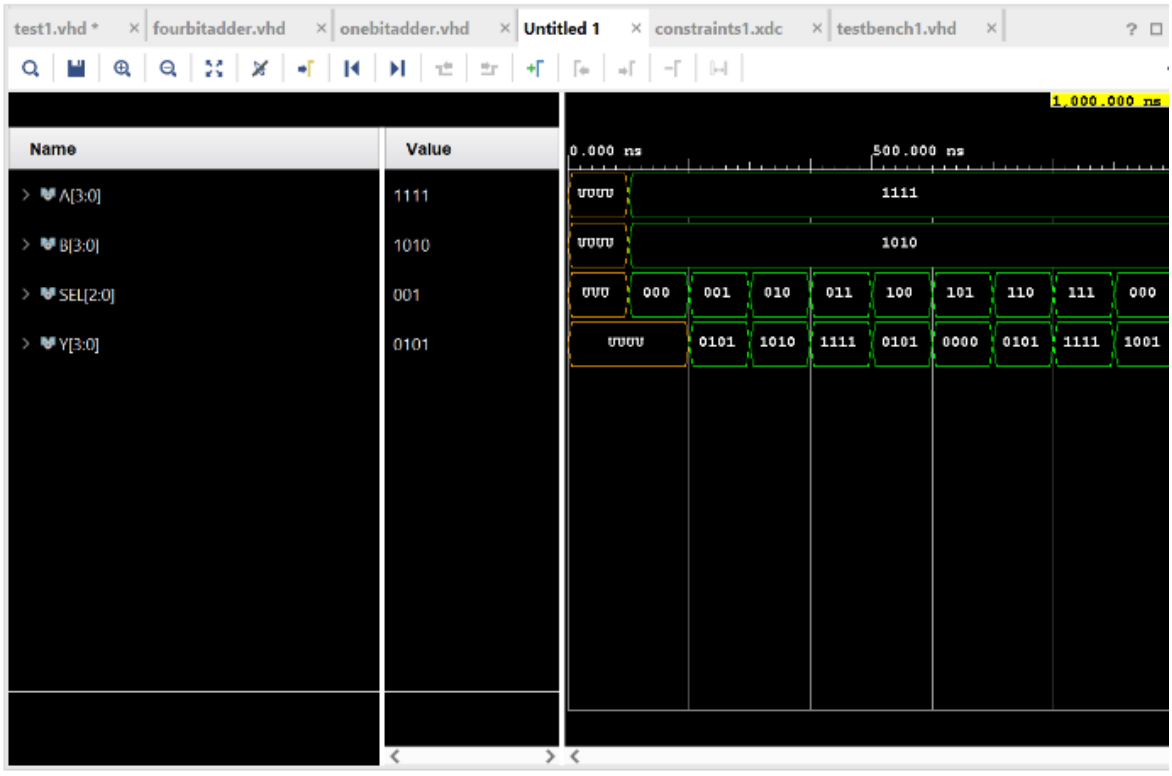
If SEL is “110” NOT B is done.

If SEL is “111” right shift is done.

RTL Schematic:

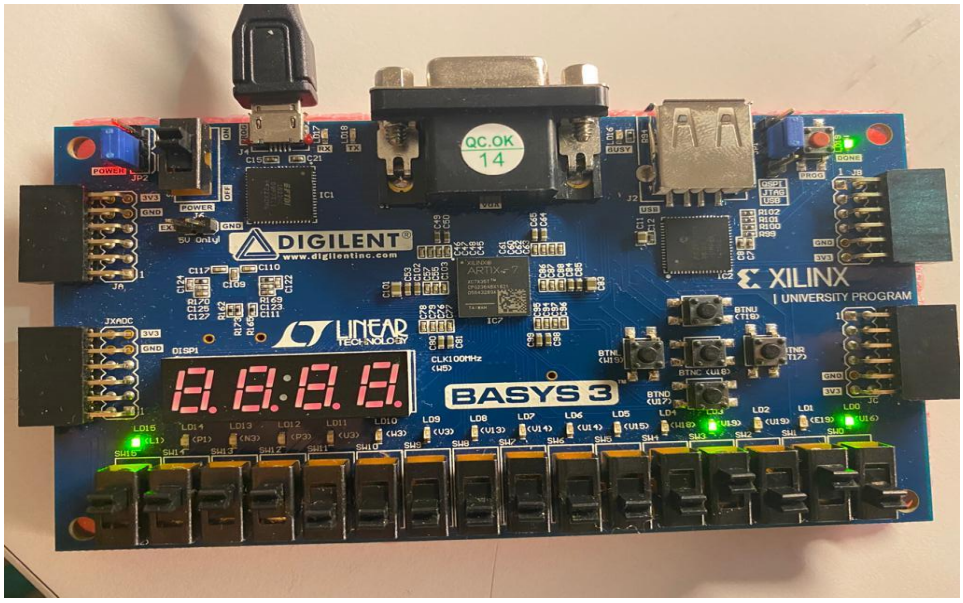


Simulation:

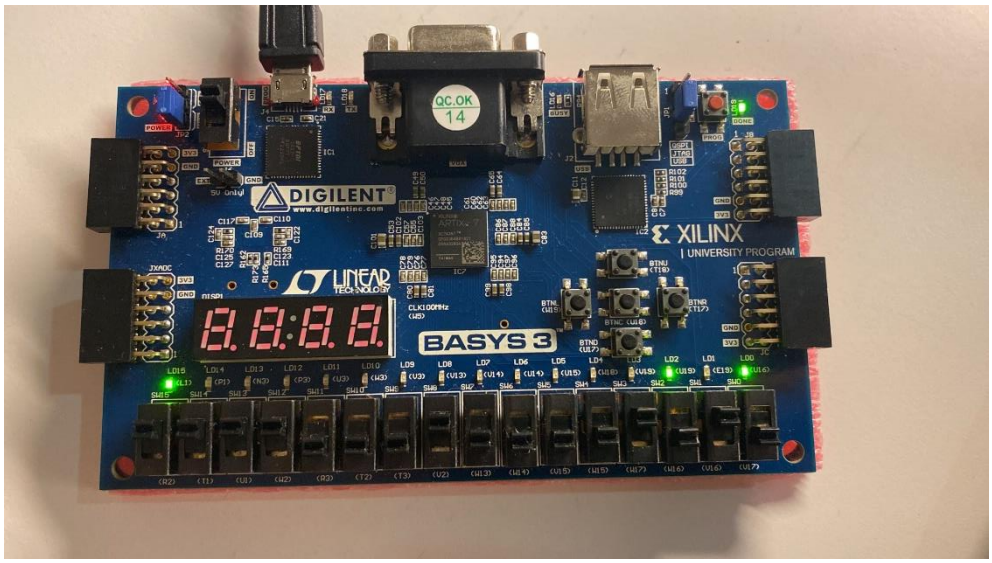


Demonstration of Operations on BASYS3:

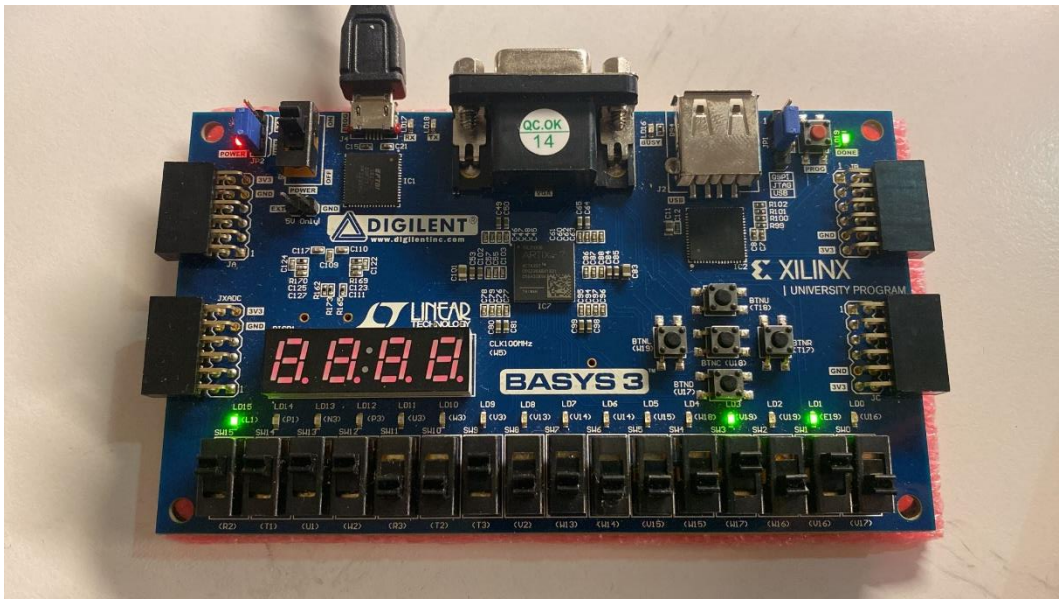
A is assigned as “1111”,B is assigned as “1010”.



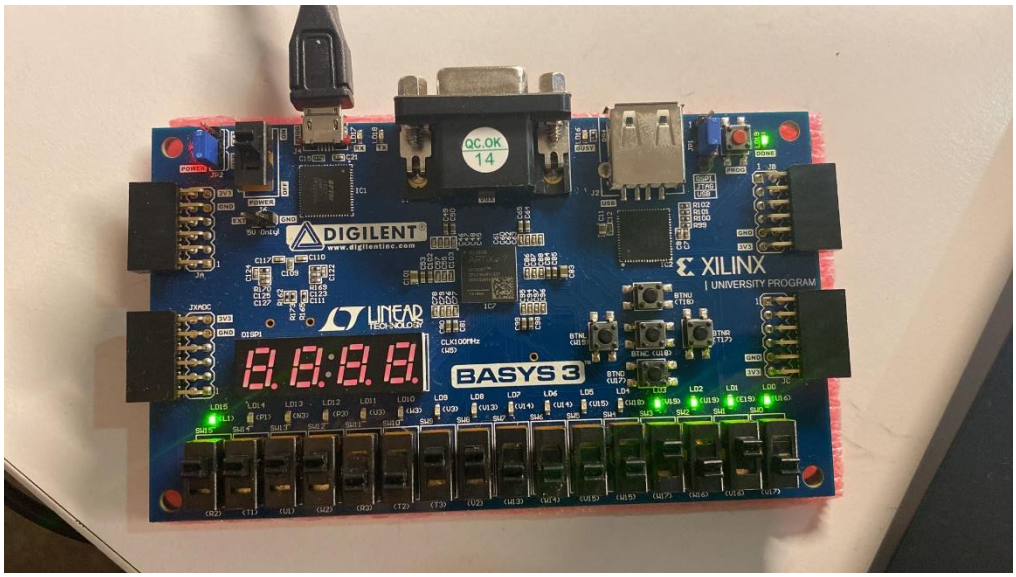
Summation



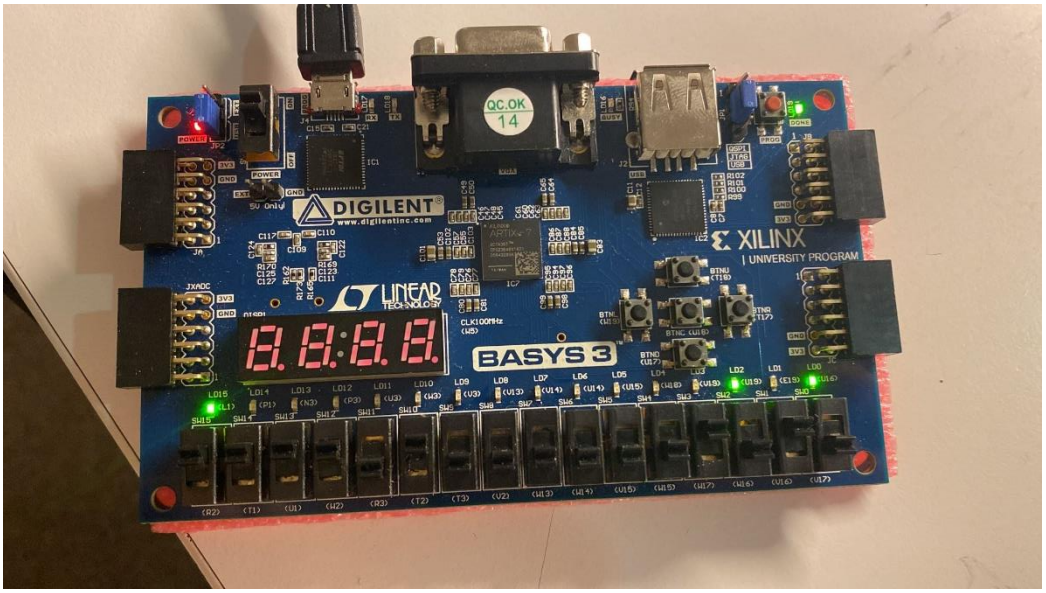
Substraction



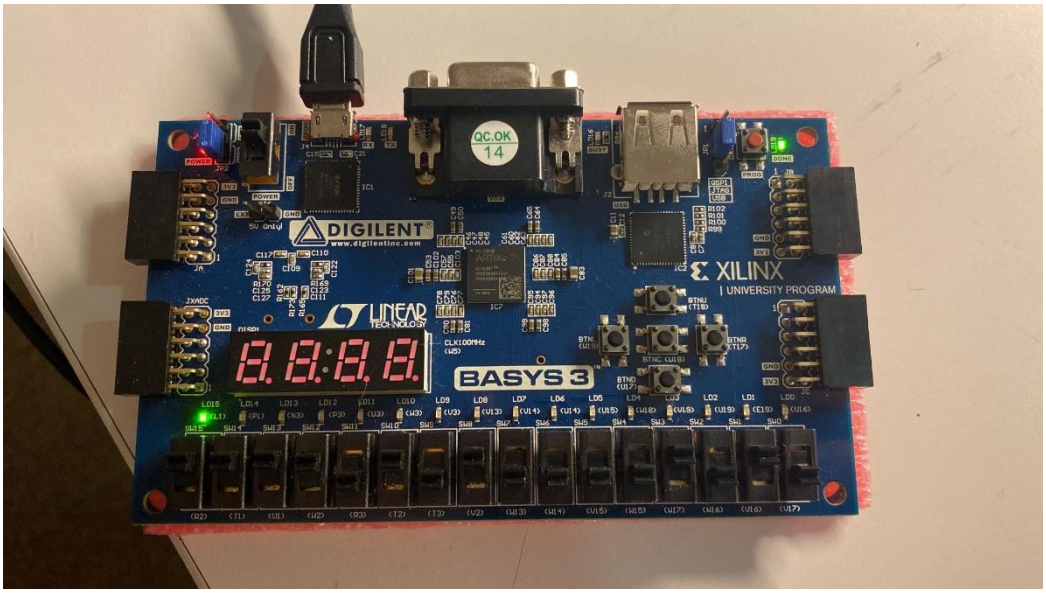
AND gate



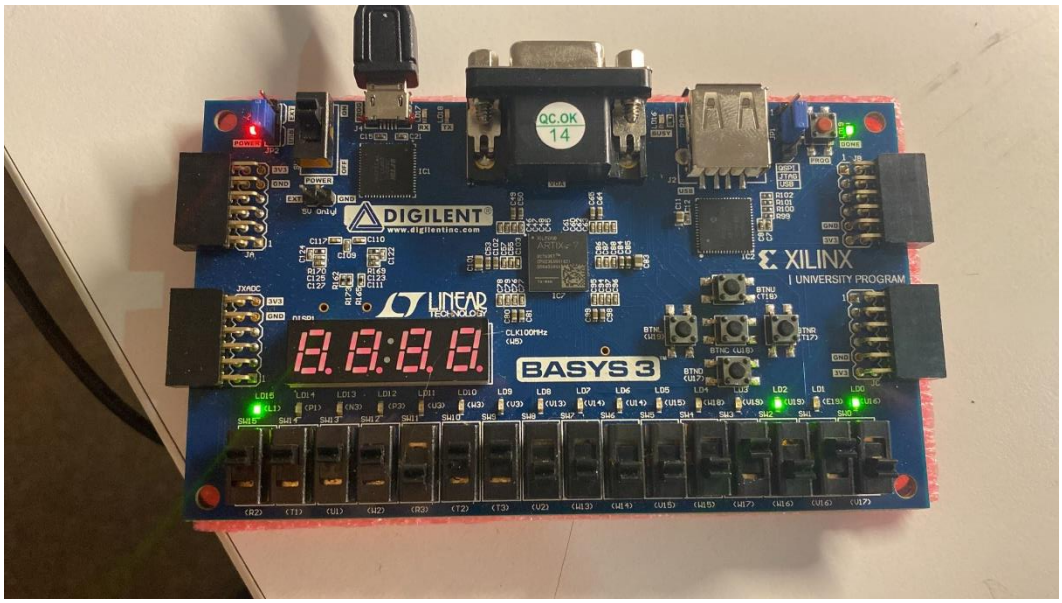
OR gate



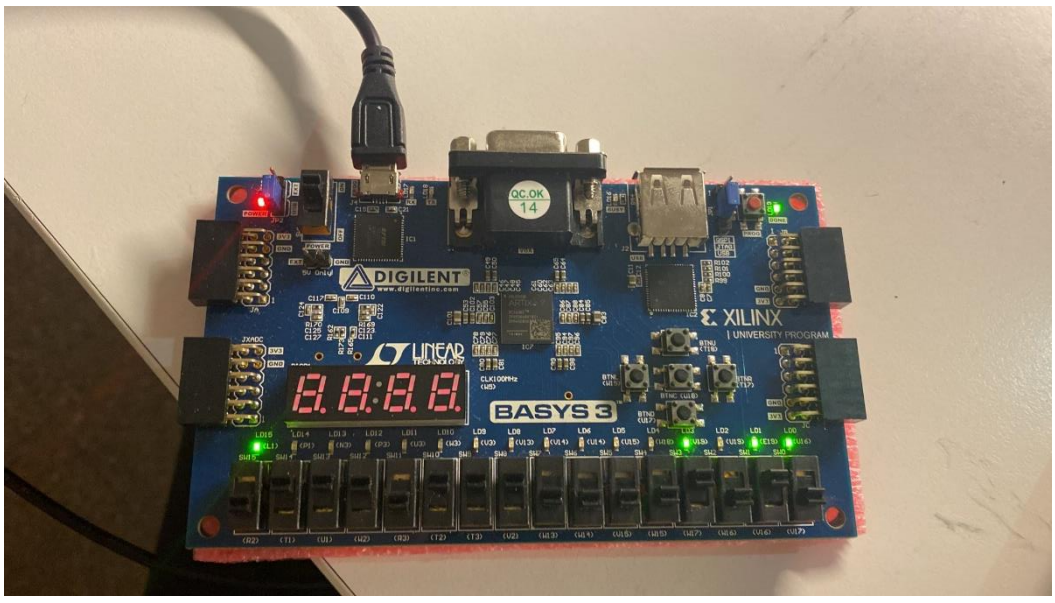
NAND gate



NOT A



NOT B



Right shift operation when A=”0111”

Conclusion:

The experiment was successful since the values in simulation and BASYS LED demonstration are accurate.ALU operates succesfully and ı further got used to bitwise operations and arithmetic operations thanks to this experiment.

Codes:

Code of one bit adder(“onebitadder”):

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity onebitadder is
   Port ( A : in STD_LOGIC;
        B : in STD_LOGIC;
        cin : in STD_LOGIC;
        sum : out STD_LOGIC;
        cout : out STD_LOGIC);
end onebitadder;

architecture Behavioral of onebitadder is

begin
sum <=  A xor B xor cin;



cout <= (A and B) or (cin and A) or (cin and B);

end Behavioral;
```

Code of four bit adder(fourbitadder):

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

```
entity fourbitadder is
Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
     B : in STD_LOGIC_VECTOR (3 downto 0);
     cin : in STD_LOGIC;
     cout : out STD_LOGIC;
     sum : out STD_LOGIC_VECTOR (3 downto 0));
end fourbitadder;

architecture structure of fourbitadder is
COMPONENT onebitadder
Port( A, B, cin          : in  STD_LOGIC;
          sum, cout          : out STD_LOGIC );
end component;
signal s1, s2, s3, s4: STD_LOGIC;
begin

b1: onebitadder port map (A(0), B(0), cin, sum(0), s2);
b2: onebitadder port map (A(1), B(1), s2, sum(1), s3);
b3: onebitadder port map (A(2), B(2), s3, sum(2), s4);
b4: onebitadder port map (A(3), B(3), s4, sum(3), cout);

end structure ;
```

Code of ALU(“test1”):

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity test1 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);
       SEL : in STD_LOGIC_VECTOR (2 downto 0);
```

```
        OVERF: out std_logic);
end test1;

architecture Behavioral of test1 is
signal c_in: std_logic:='0';
signal cout_sum: std_logic:='0';
signal cout_sub: std_logic:='0';
signal s_sum: std_logic_vector(3 downto 0);
signal s_sub: std_logic_vector(3 downto 0);

component fourbitadder is
Port(A : in std_logic_vector(3 downto 0);
B : in std_logic_vector(3 downto 0);
cin : in std_logic;
sum : out std_logic_vector(3 downto 0);
cout : out std_logic);
end component;
begin
--Adder
ADD: fourbitadder port map(A => A,
B => B,
cin => c_in,
sum => s_sum,
cout => cout_sum);
--Substractor
SUB: fourbitadder port map(A =>A,
B => not B,
cin => not c_in,
sum => s_sub,
cout => cout_sub);
```

```
PROCESS (A,B,SEL)
BEGIN
CASE SEL IS
WHEN "000" =>
   Y <=  s_sum;
   OVERF <= cout_sum;
WHEN "001" =>
   Y <= s_sub;
   OVERF <= cout_sub;
WHEN "010" =>
   Y <= A AND B;
WHEN "011" =>
   Y <= A OR B;
WHEN "100" =>
   Y <= A NAND B;
WHEN "101" =>
   Y <= NOT A;
WHEN "110" =>
   Y <= NOT B;
WHEN "111" =>
   Y <= A(0) & A(3 downto 1);
WHEN OTHERS =>
       NULL;
END CASE;
END PROCESS;
end Behavioral;
```

Code of testbench(“testbench1”):

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity testbench1 is
--  Port ( );
end testbench1;

architecture Behavioral of testbench1 is
COMPONENT test1
 PORT( A : IN std_logic_vector(3 downto 0);
 B : IN std_logic_vector(3 downto 0);
 SEL : IN std_logic_vector(2 downto 0);
 Y : OUT std_logic_vector(3 downto 0);
 OVERF : out std_logic);
 END COMPONENT;
 SIGNAL A : std_logic_vector(3 downto 0);
 SIGNAL B : std_logic_vector(3 downto 0);
 SIGNAL SEL : std_logic_vector(2 downto 0);
 SIGNAL Y : std_logic_vector(3 downto 0);


BEGIN
UUT: test1 PORT MAP(
A => A,
B => B,
SEL => SEL,
Y => Y
 );
testbench1 : PROCESS
begin
  wait for 100ns;
```

```
A <= "1111";
B <= "1010";
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;
```

Code of constraints(“constraints1”):

```
set_property PACKAGE_PIN W2 [get_ports {A[0]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {A[0]}]
set_property PACKAGE_PIN U1 [get_ports {A[1]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {A[1]}]
set_property PACKAGE_PIN T1 [get_ports {A[2]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {A[2]}]
set_property PACKAGE_PIN R2 [get_ports {A[3]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {A[3]}]
set_property PACKAGE_PIN V17 [get_ports {B[0]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {B[0]}]
set_property PACKAGE_PIN V16 [get_ports {B[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B[1]}]
set_property PACKAGE_PIN W16 [get_ports {B[2]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {B[2]}]
set_property PACKAGE_PIN W17 [get_ports {B[3]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {B[3]}]
set_property PACKAGE_PIN V2 [get_ports {SEL[0]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {SEL[0]}]
set_property PACKAGE_PIN T3 [get_ports {SEL[1]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {SEL[1]}]
set_property PACKAGE_PIN T2 [get_ports {SEL[2]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {SEL[2]}]
set_property PACKAGE_PIN U16 [get_ports {Y[0]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {Y[0]}]
set_property PACKAGE_PIN E19 [get_ports {Y[1]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {Y[1]}]
```

```
set_property PACKAGE_PIN U19 [get_ports {Y[2]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {Y[2]}]
set_property PACKAGE_PIN V19 [get_ports {Y[3]}]
 set_property IOSTANDARD LVCMOS33 [get_ports {Y[3]}]
set_property PACKAGE_PIN L1 [get_ports {OVERF}]
 set_property IOSTANDARD LVCMOS33 [get_ports {OVERF}]
```