COEN 313 Lab 2

Name:	JUNPENG GAI
ID number:	40009896
Course number:	COEN313 UJ-X Lab
Date performed:	Wednesday, 13 February 2023
Due Date:	Wednesday, 3 March 2023

"I certify that this submission is my original work and meets the Faculty's Expectations of Originality"

Table of Contents

1. Objective

The purpose of this lab is to learn more VHDL, different from lab 1 we will write our own VHDL code for the given concept diagram.

There are 2 ways of implementation:

1. by using port map

2.by using only one line of Boolean expression.

After that, we will compile and simulate the code with 2 different styles and we can perform synthesis analysis and the compare the schematic in the FPGA.

2. Procedure

A	B	C	OUT
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 1Truth table of circuit

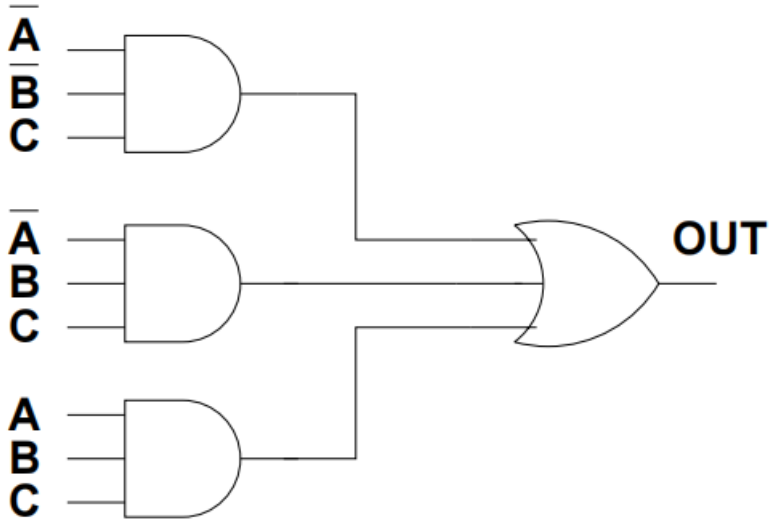

Figure 1 Logic gate implementation

2.1 VHDL simulation for the port map version :

In this simulation we are going to use port map.

I wrote 3 vhdl file, they are and3_gate.vhdl , or3_gate.vhdl and sum.vhdl.

2.1.1 Writing vhdl code for port map version

```
library IEEE; use IEEE.std_logic_1164.all;

entity and3_gate  is
port( and1, and2, and3: in std_logic;
      andout : out std_logic);
end and3_gate;


architecture example of and3_gate is begin
  andout <= and1 and and2 and and3;
end;
```

Figure 2 and3_gate.vhdl

```
library IEEE; use IEEE.std_logic_1164.all;

entity or3_gate  is
port( or1, or2 ,or3 : in std_logic;
      orout : out std_logic);
end or3_gate;


architecture example of or3_gate is begin
  orout <= or1 or or2 or or3;
end;
```

Figure 3 or3_gate.vhdl

```
library IEEE;
use IEEE.std_logic_1164.all;
entity sum is
port( a,b,c     : in std_logic;
      output    : out std_logic);
end sum;


architecture structural of sum is

-- declare a half-adder component


component and3_gate
  port ( and1, and2,and3 :   in std_logic;
         andout : out std_logic);
end component;


component or3_gate
  port ( or1, or2,or3 :   in std_logic;
         orout : out std_logic);
end component;

signal out1, out2,out3,not_a,not_b     : std_logic;

for a1, a2,a3 : and3_gate use entity WORK.and3_gate(example);
for o1 : or3_gate use entity WORK.or3_gate(example);

begin

not_a <= not a;
not_b <= not b;


a1: and3_gate port map(and1 => not_a, and2 => not_b,
                       and3 => c, andout => out1);

a2: and3_gate port map(and1 => not_a, and2 => b,
                       and3 => c, andout => out2);

a3: and3_gate port map(and1 => a, and2 => b,
                       and3 => c, andout => out3);

o1: or3_gate port map(or1 => out1, or2 => out2,
                      or3 => out3, orout => output);



end structural;
```

Figure 4 sum.vhdl

We created 2 components for and3_gate and or3_gate, we will use those 2 architectures in the sum.vhdl file to simulate and implement the circuit in the FPGA. There are 3 input signals and 2 intermedia signals to pass the output from one component to another component.

2.1.2 Modelsim simulation

Here is our do file

```
add wave a
add wave b
add wave c
add wave output



force a 0
force b 0
force c 0
run 2

force a 0
force b 0
force c 1
run 2

force a 0
force b 1
force c 0
run 2

force a 0
force b 1
force c 1
run 2

force a 1
force b 0
force c 0
run 2

force a 1
force b 0
force c 1
run 2

force a 1
force b 1
force c 0
run 2

force a 1
force b 1
force c 1
run 2
```

Table 2 lab2.do file

Therefore, we can use the modelsim to generate the wave:

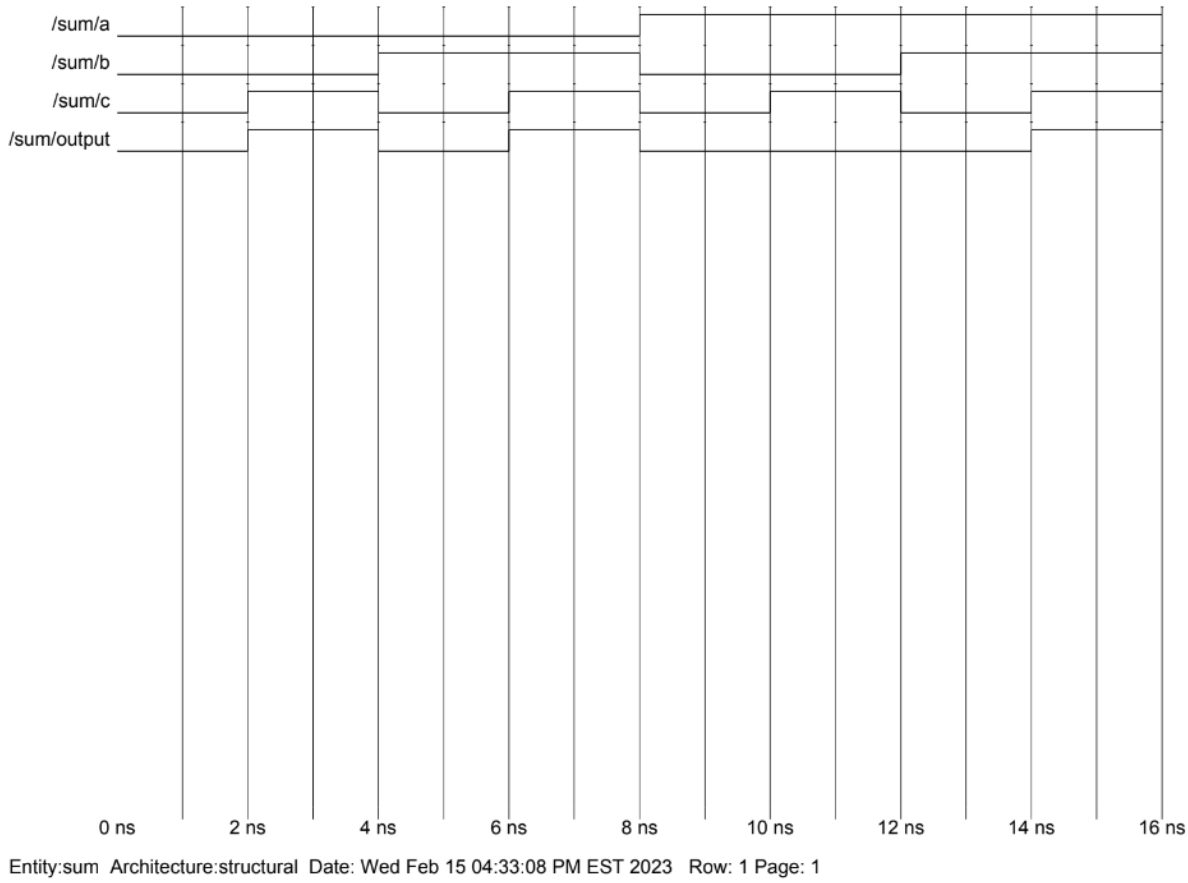

Figure 5 Modelsim result for port map version

Because we have all 2^3=8 combinations in the force file, so in our output we can see they performed as expected from the truth table. Next, we will start to implement the vhdl file to the FPGA board.

2.1.3 Xilin implementation

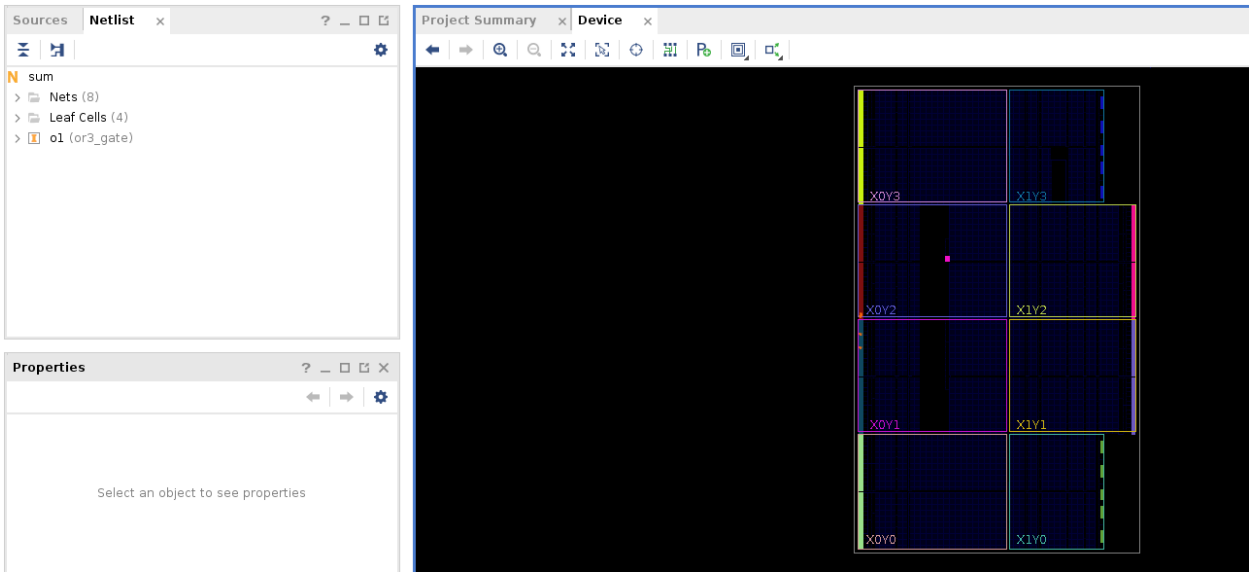

Figure 6 implemented design on device for port map

After we import the file and constrain file to the Xilinx Vivado, above figure shows us the elaboration design for the device in the FPGA board.

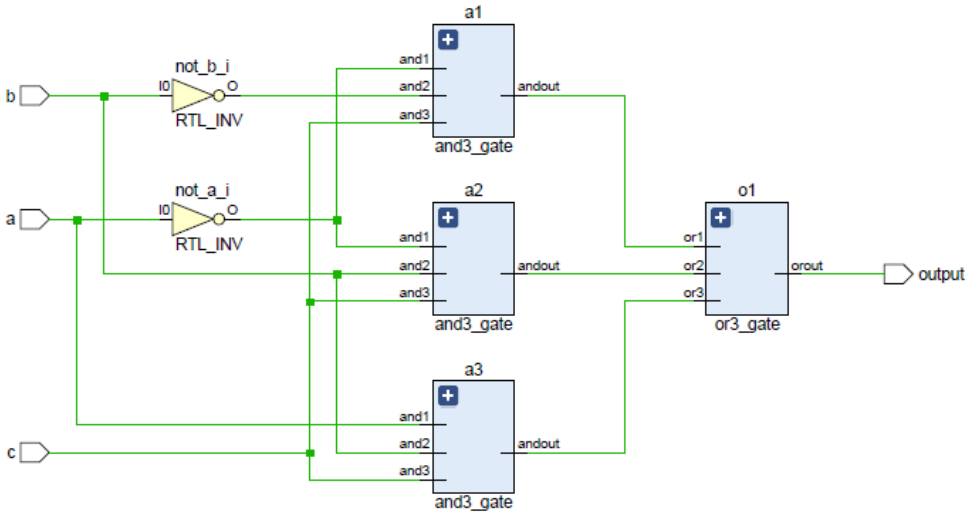

Figure 7 RTL elaborated schematic for port map version

Here is the RTL schematic for the port map version, from the figure we can see that each component is expressed by a concept block. Next step we will implement the design and generate the bit stream and upload it to the FPGA board.

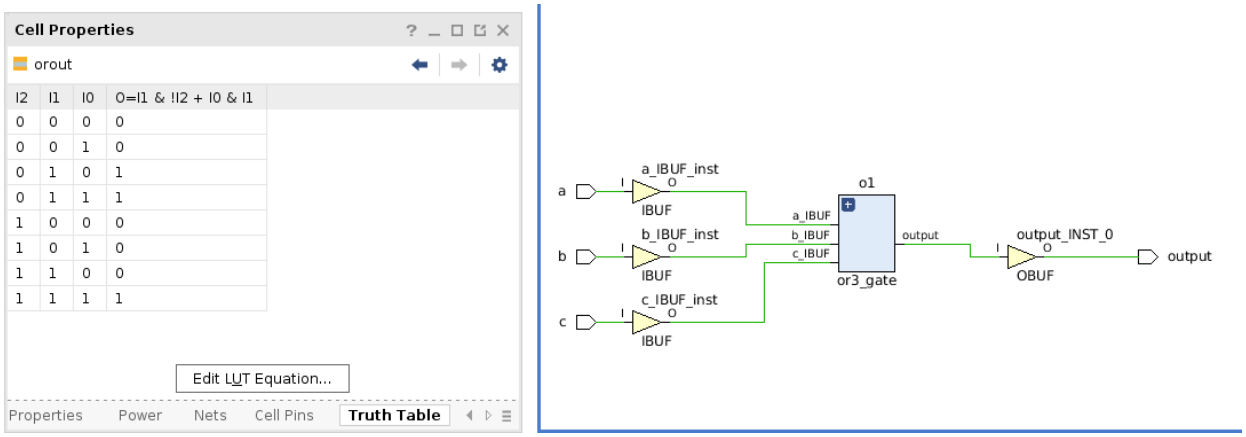

Figure 8 port map version implemented design and truth table

2.2 VHDL simulation for Boolean expression version:

2.2.1 Writing vhdl code for port map version

```
library IEEE;
use IEEE.std_logic_1164.all;

entity sum_of_minterms is
port( a,b,c : in std_logic;
      output    : out std_logic);
end sum_of_minterms;


architecture outputs of sum_of_minterms is
begin

    output <=  (((not a )and (not b) and c) or ((not a) and b and c) or (a and b and c));

end outputs;
```

Figure 9 sum_of_minterms.vhdl

This vhdl file is relative simple, because we use one boolean expression to get output. Xilinx Vivado will optimize the code and generate the RTL elaboration schematic and implement schematic.

2.2.2 Modelsim simulation

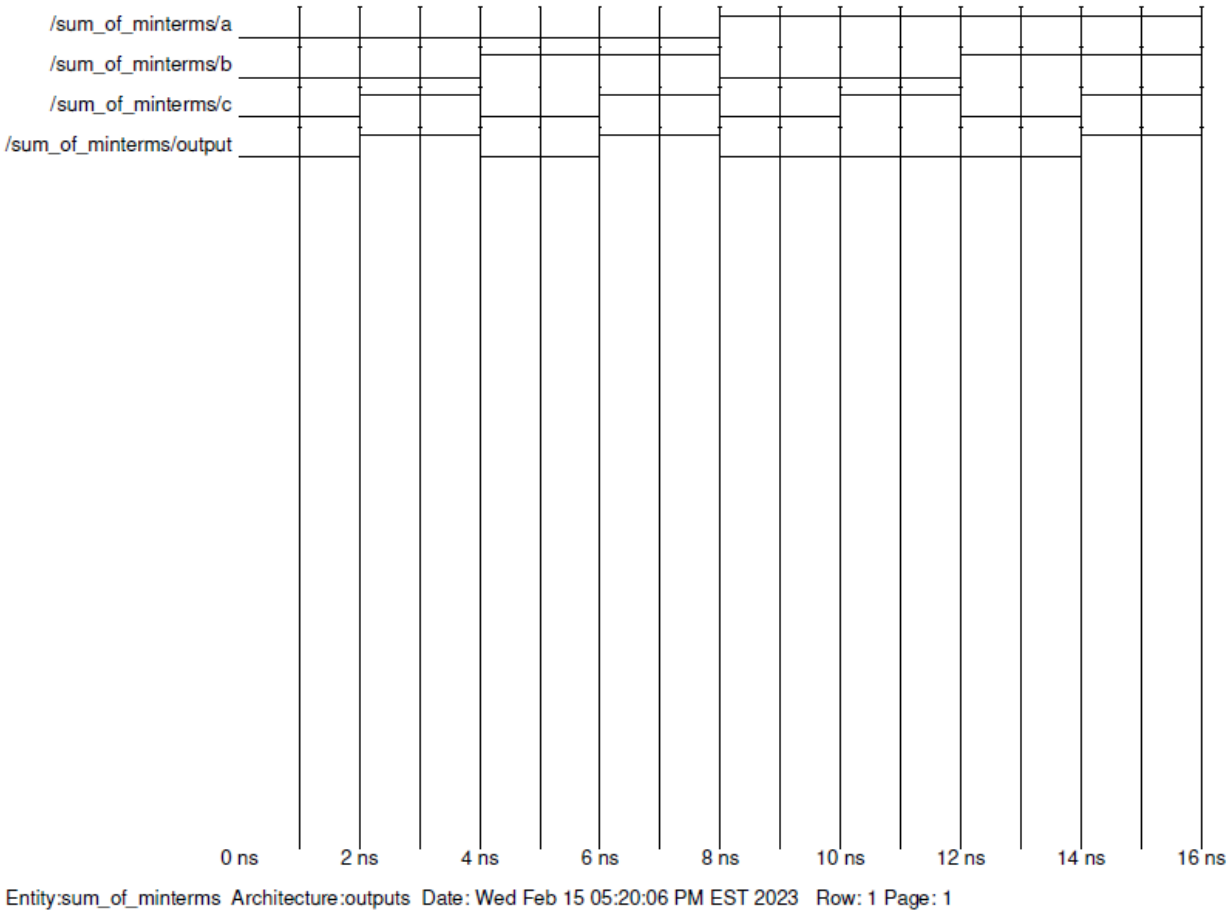

Figure 10 Modelsim result for port map version

We used the same do file to get the input and put it into the Modelsim to simulate the output.The result is shown in the above figure.

2.2.3 Xilinx implementation

After we set up the project, we can generate the RTL elaborated schematic for sum_of_minterms.vhdl file.

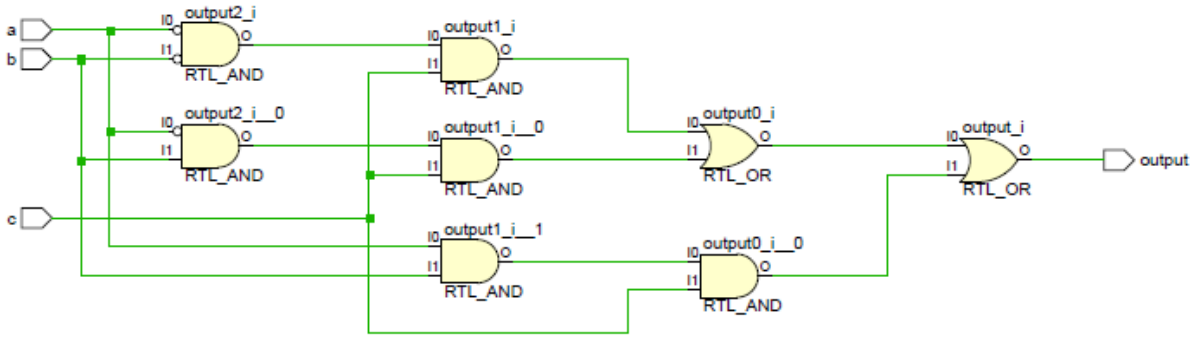

Figure 11 RTL elaborated schematic for boolean expression version

Also, we can get the implemented design, here we can see that even the elaborated schematic is different from the port map version, but we will have the same implementation schematic. That's because the optimized of the Xinlin Vivado, since they have exactly same functionality the implementation will be the same.

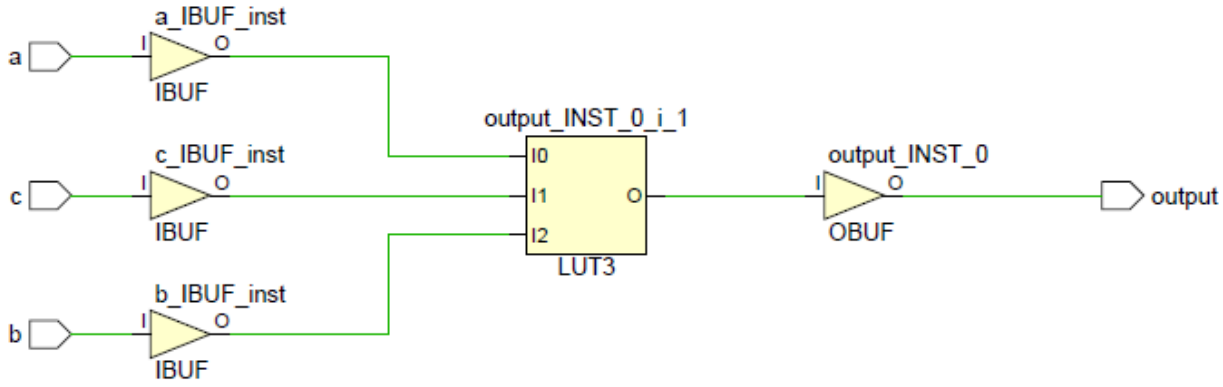

Figure 12 Sum of minterms implemented design

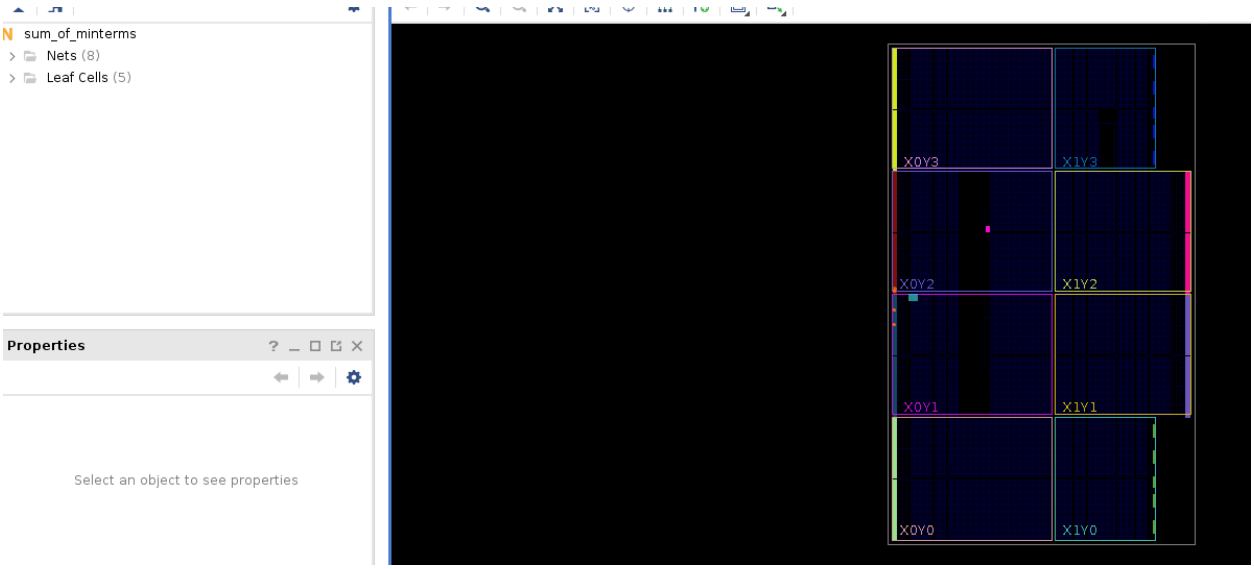

Figure 13 implemented design on device

3. Conclusion

In this lab, for the given circuit we used 2 different method to simulate and implement the design in the Modelsim and Xilinx Vivado.

After doing the simulation for both versions, we can confirm the code is right because the output meets the expectation from the given truth table.2 RTL elaborated schematic are different, port map version looks like a concept diagram while the other version is a logic gate schematic. But finally in the implemented design we have the same implementation in the FPGA board, that is because the optimization of Xilinx Vivado enable to optimize both design of same functionality to be the same implementation in the board.

4. Question

1. ***Rewrite the VHDL code for the sum_of_minterms entity making use of only CSA statements (no port maps). Re-synthesize your design with Vivado and compare the resulting RTL elaborated and Implemented schematic diagrams with that of the original design. Comment on any differences/similarities among the schematics. You do not have to download this version of the circuit to the FPGA board.***

```
library IEEE;
use IEEE.std_logic_1164.all;

entity sum_of_minterms is
port( a,b,c : in std_logic;
      output    : out std_logic);
end sum_of_minterms;


architecture outputs of sum_of_minterms is
begin

    output <=  (((not a )and (not b) and c) or ((not a) and b and c) or (a and b and c));

end outputs;
```

I rewrite the code and do the simulation in the lab section, in the simulation.

The RTL schematic for the port map version, from the **figure 7** we can see that each component is expressed by a concept block. But here we can see that even the elaborated schematic is different from the port map version, but we will have the same implementation schematic. That's because the optimized of the Xinlin Vivado, since they have exactly same functionality the implementation will be the same.

2. ***Determine the Boolean function which the LUT implements in both versions of the VHDL code.***

   Is the function equal to the original sum-of-minterms expression described by the VHDL code?

orout

I2	I1	I0	O=I1 & !I2 + I0 & I1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Figure 14 port map version

Cell Properties

output_INST_0_i_1

I2	I1	I0	O=!I0 & I1 + I1 & I2
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Edit LUT Equation...

Figure 15 rewrite boolean expression version

$$output\_port\_map = I0 \,\&\, I1 + I1 \,\&\, !\, I2$$

$$output\_port\_map = \bar{A}C + BC$$

Because in the port map version

b: I0

c: I1

a: I2

$$outpu\_SumOfMinterms = !I0 \,\&\, I1 + I1 \,\&\, I2$$

$$outpu\_SumOfMinterms = \bar{A}C + BC$$

In the boolean expression version

a: I0

b: I2

c: I1

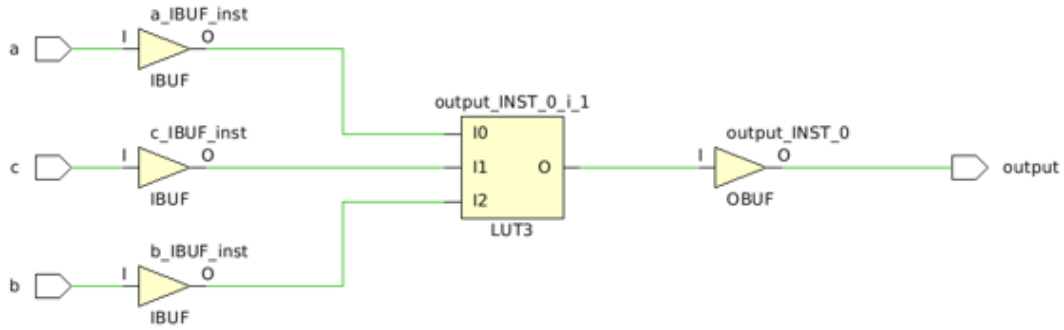

Figure 16 lut equation and schematic

We can see from the truth tables above. I0 I1 I2 are three inputs to the LUT and the LUT equations are exactly the same. Because in 2 version I0 I1 I2 don't stand for the same input so the truth table might look a little bit different.

3. ***Do the two RTL elaborated schematics indicate whether the synthesis tool has performed any logic minimization?***

Yes, it does. From the initial expression:

$$out = \bar{A}\bar{B}C + \bar{A}BC + ABC$$

LUT we get:

$$\boldsymbol{outpu\_SumOfMinterms = !I0 \,\&\, I1 + I1 \,\&\, I2}$$

Or

$$\boldsymbol{output\_port\_map = I0 \,\&\, I1 + I1 \,\&\, !I2}$$

Which equivalent to:

$$\boldsymbol{output_{pormap} = outpu\_SumOfMinterms = \bar{A}C + BC}$$

If we do the Boolean calculation by ourselves:

$out = \bar{A}\bar{B}C + \bar{A}BC + ABC$

$= \bar{A}\bar{B}C + BC$

$= (\bar{A}\bar{B} + B)C$

$= \big((\bar{B} + B) + (\bar{A} + B)\big)C$

$= \bar{A}C + BC$

Therefore, we can find out that indicates the logic minimization have been performed.

4. ***Determine whether the following VHDL code results in RTL elaborated and Implemented schematics which are indicative of logic minimization having been performed:***

No, because here the output is:

$$output = \overline{\overline{(input\_1)}}$$

Basically, it's a double negation. Because it has been the simplest form. It can be concluded that RTL elaborated and implemented schematics can't do further logic minimization.

5. Appendix

5.1 or3_gate.vhdl

```
library IEEE; use IEEE.std_logic_1164.all;

entity or3_gate  is
port( or1, or2 ,or3 : in std_logic;
     orout : out std_logic);
end or3_gate;

architecture example of or3_gate is begin
  orout <= or1 or or2 or or3;
end;
```

5.2 and3_gate.vhdl

```
library IEEE; use IEEE.std_logic_1164.all;

entity and3_gate  is
port( and1, and2, and3: in std_logic;
     andout : out std_logic);
```

```
end and3_gate;

architecture example of and3_gate is begin
  andout <= and1 and and2 and and3;
end;
```

5.3 Sum.vhdl(port map version)

```
library IEEE;
use IEEE.std_logic_1164.all;
entity sum is
port( a,b,c         : in std_logic;
      output        : out std_logic);
end sum;

architecture structural of sum is
component and3_gate
  port ( and1, and2,and3 :   in std_logic;
         andout : out std_logic);
end component;

component or3_gate
  port ( or1, or2,or3 :   in std_logic;
         orout : out std_logic);
end component;
```

```
signal out1, out2,out3,not_a,not_b     : std_logic;

for a1, a2,a3 : and3_gate use entity WORK.and3_gate(example);
for o1 : or3_gate use entity WORK.or3_gate(example);

begin
not_a <= not a;
not_b <= not b;

a1: and3_gate port map(and1 => not_a, and2 => not_b,
                       and3 => c, andout => out1);

a2: and3_gate port map(and1 => not_a, and2 => b,
                       and3 => c, andout => out2);

a3: and3_gate port map(and1 => a, and2 => b,
                       and3 => c, andout => out3);

o1: or3_gate port map(or1 => out1, or2 => out2,
                      or3 => out3, orout => output);
end structural;
```

5.4 Sum.vhdl(one line of boolean expression)

```
library IEEE;
use IEEE.std_logic_1164.all;

entity sum_of_minterms is
port( a,b,c        : in std_logic;
     output        : out std_logic);
end sum_of_minterms;

architecture outputs of sum_of_minterms is
begin
          output <=  (((not a )and (not b) and c) or ((not a) and b and c) or (a and b and c));
end outputs;
```

5.5 sum.xdc

```
# Vivado does not support old UCF syntax
# must use XDC syntax

set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 }   [ get_ports { a }      ] ;
set_property -dict { PACKAGE_PIN L16 IOSTANDARD LVCMOS33 }   [ get_ports { b }    ] ;
set_property -dict { PACKAGE_PIN M13 IOSTANDARD LVCMOS33 }   [ get_ports { c }  ] ;
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 }   [ get_ports
{ output }    ] ;
```