Digital Systems Design II

COEN 313 (FJ-X)

Experiment 2

Andre Hei Wang Law

4017 5600

Performed on October 4, 2021

Due on October 18, 2021

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

Friday, October 15, 2021

4017 5600

1) Objectives

In continuation of the first experiment, the main purpose of the second lab of the course COEN 313 is to become acquainted with structural and concurrent VHDL. Then, students are also expected to become familiar with different VHDL coding styles as well as to gain knowledge in combinational logic minimization such performed by logic synthesis tools introduced during the first experiment.

2) Procedure

With the aide of the pdf files "Digital Logic Simulation and Synthesis Using Modelsim, precision RTL, and Xilinx Ise" and "lab2.pdf", students will implement a two-level Boolean function circuit with structural VHDL design using port map statements of the necessary "AND" and "OR" gates. After simulating and verifying their design with the Modelsim simulator, they will synthesize and implement their VHDL code with Vivado in order to generate the bitstream and program the FPGA board. Overall, it is by writing, implanting, synthesizing and testing their VHDL codes that students will achieve mastery on handling different statement designs as well as VHDL coding styles.

3) Design

Table 1. Boolean Function Truth of Table

A	B	C	OUTPUT
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 2. K-Map of the Boolean Function

A\BC	00	01	11	10
0	0	1	1	0
1	0	0	1	0

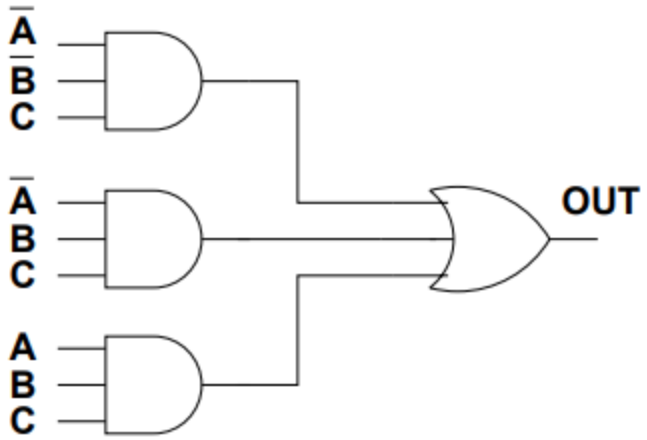

Figure 1: Two-level sum of minterms hardware implementation.

With the given truth table from "lab2.pdf", we are able to determine the k-map of this Boolean function. The simplified output can be calculated as the following:

$$OUTPUT = A'C + BC$$
$$OUTPUT = C(A' + B)$$

To verify the validity of this answer, it is possible to simplify the given Boolean equation sum of minterms form of "OUT=A'B'C + A'BC + ABC" as such: (represented by Figure 1)

$$OUT = A'B'C + A'BC + ABC$$
$$OUT = C(A'B' + A'B) + ABC$$
$$OUT = CA' + ABC$$
$$OUT = C(A' + AB)$$
$$OUT = C(A' + B)$$

It can be noticed that this reduced form is the exact same as the calculated one from the k-map. As such, we can argue that we have correctly implemented the k-map of the Boolean function.

4) Results

During experiment 2, many encountered errors and had countless questions concerning the process of implementing their VHDL codes correctly and properly. In my case, I was able to steadily work on my part until I had issues and unexpected responses from my implementation in Vivado which restricted me of collecting my necessary data in time. As such, some deliverables and results have not been gathered properly while some may contain hypothetical results. However, by at least having these results in hand, it ensures that it will not hinder the questions section and some parts of the results section of this report.

4.3) VHDL Code of the First Versions of the Design.

VHDL Code of an AND Gate:

```
entity and_gate is
port (   A, B, C: in BIT;
         OUTPUT: out BIT);
end and_gate;
architecture and_arch of and_gate is
begin
         OUTPUT <= A and B and C after 5 ns;
end and_arch;
```

VHDL Code of an OR Gate:

```
entity or_gate is
port (   A, B, C: in BIT;
         OUTPUT: out BIT);
end or_gate;
architecture or_arch of or_gate is
begin
         OUTPUT <= A or B or C;
end or_arch;
```

VHDL Code of the Circuit using Structural Design:

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

entity sum_of_minterms is
         port (   A, B, C: in BIT;
                  D: out BIT);
end sum_of_minterms;
```

```
architecture circuit_arch of sum_of_minterms is

component and_gate
        port (    A, B, C: in BIT;
        OUTPUT: out BIT);
end component;

component or_gate
        port (    A, B, C: in BIT;
        OUTPUT:       out BIT);
end component;

signal s1, s2, s3:         BIT;
signal not_A, not_B:   BIT;

for U1, U2, U3: and_gate use entity WORK.and_gate(and_arch);
for U4:            or_gate use entity WORK.or_gate(or_arch);

begin
not_A <= not A;
not_B <= not B;

U1:      and_gate port map(A=>not_A, B=> not_B, C=> C, OUTPUT=> s1);
U2:      and_gate port map(A=> not_A, B=> B, C=> C, OUTPUT=> s2);
U3:      and_gate port map(A=> A, B=> B, C=> C, OUTPUT=> s3);
U4:      or_gate port map(A=> s1, B=> s2, C=> s3, OUTPUT => D);
end circuit_arch;
```

4.4) Testing – Do File.

```
force a 0
force b 0
force c 0
run

force a 0
force b 0
force c 1
run

force a 0
force b 1
force c 0
run

force a 0
force b 1
force c 1

force a 1
force b 0
force c 0
run

force a 1
force b 0
force c 1
run
```

```
force a 1
force b 1
force c 0
run

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

4.5) Testing - Xilinx XDC File.

```
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}];
```

5) Questions

5.1) 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.

Rewritten VHDL Code using only CSA Statements: (Second Version of the Design):

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

entity csa_rewrite is
port (   a: in std_logic_vector(2 downto 0);
         output: out std_logic);
end csa_rewrite;

architecture circuit_rewrite of csa_rewrite is
         signal s1, s2, s3: std_logic;
begin
         s1 <= (not a(0)) and (not a(1)) and a(2) after 12 ns;
         s2 <= (not a(0)) and a(1) and a(2) after 12 ns;
         s3 <= a(0) and a(1) and a(2) after 12 ns;
         output <= s1 or s2 or s3 afer 20 ns;
end circuit_rewrite;
```

Differences: The two RTL elaborated designs are vastly different as the original one had components to separate each signal which is clear and simple to understand. On the other hand, the CSA version circuit had less components, thus it is more efficient, yet it has many elements which made it harder to read and understand at a glance.

Similarities: The Implemented Schematic Diagram and truth table for both is the same as the simplified and optimized design of the original Boolean function is the same no matter the style of code.

5.2) 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?

To begin, it is important to understand that the design of the VHDL code is implemented based on the available logic elements of the Xilinx device. It is the Xilinx Vivado tools that determine the method of implementing the code into Boolean logic. As such, the truth table obtained from the "Cell Properties" window has to correspond to the truth table that the LUT has been programmed to implement which was given in table 1 of "lab2.pdf".

5.3) Do the two RTL elaborated schematics indicate whether the synthesis tool has performed any logic minimization?

The RTL elaborated schematics of both codes won't have signs of logic minimization as this is a pre-optimized schematic. In other words, this design is generic and keeps its normal gate-level design with AND gates, OR gates, etc. It is important to understand that the RTL schematic design is independent of the Xilinx device, making this design its pre-optimized form.

5.4) Determine whether the following VHDL code results in RTL elaborated and implemented schematics which are indicative of logic minimization having been performed:

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   entity keith is
4   port (  input_1: in std_logic;
5           output: out std_logic);
6   end keith;
7   architecture rtl of keith is
8           signal first, second: std_logic;
9           begin
10          first <= not input_1;
11          second <= not first;
12          output <= second;
12  end rtl;
```

The VHDL code is indicative of having an RTL elaborated and implemented schematics in which minimization occurs at lines 10-12 which become "output <= input_1". The reason behind this is that the double negation in lines 10 and 11 negate each and other, thus leaving the output simply equal to input_1.