ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E

DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	1
EXPERIMENT DATE	:	19.03.2021
LAB SESSION	:	FRIDAY - 14.00
GROUP NO	:	G14

GROUP MEMBERS:

150180112 : ÖMER MALİK KALEMBAŞI

150190014 : FEYZA ÖZEN

150190108 : EKİN TAŞYÜREK

SPRING 2021

Contents

1 INTRODUCTION [10 points]

In this experiment, we implemented AND, OR, and NOT gates by using Vivado. With this experiment, we aimed to show the axioms and theorems of boolean algebra.

2 MATERIALS AND METHODS [40 points]

2.1 PART 1

In this part, we used Verilog operators such as '&' (Bitwise AND), '|' (Bitwise OR), and '∼' (Bitwise NOT) operations to implement AND, OR, and NOT Gates.

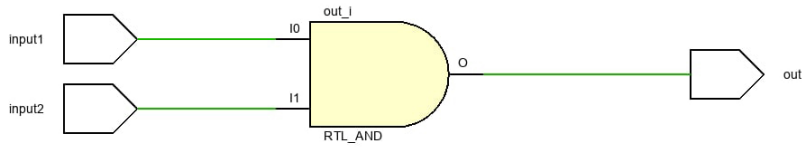

Figure 1: AND Gate RTL Schematic

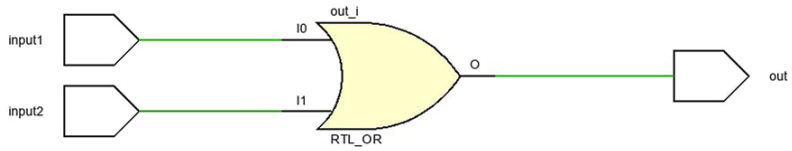

Figure 2: OR Gate RTL Schematic

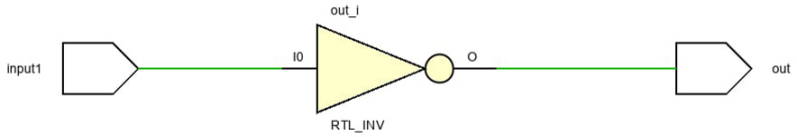

Figure 3: NOT Gate RTL Schematic

2.2 PART 2

In this part, we implemented F_1 and F_2 expressions by using AND, OR, NOT modules which we designed in the first part. We tested our implementations by giving them input combinations and use the combinations in a simulation on Vivado.

$$F_1(a, b) = a + ab = a$$

Proof of the equality:

$a + ab = a$	Original equation
$a.1 + a.b = a$	Identity
$a.(1 + b) = a$	Distributivity
$a.1 = a$	Identity
$a = a$	Identity

$$F_2(a, b) = (a + b)(a + b') = a$$

Proof of the equality:

$(a + b)(a + b') = a$	Original equation
$a + (b.b') = a$	Distributivity
$a + 0 = a$	Complement
$a = a$	Identity

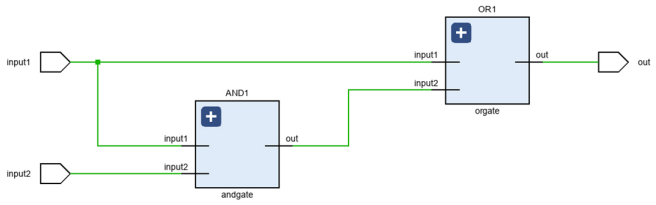

Figure 4: F_1 RTL Schematic

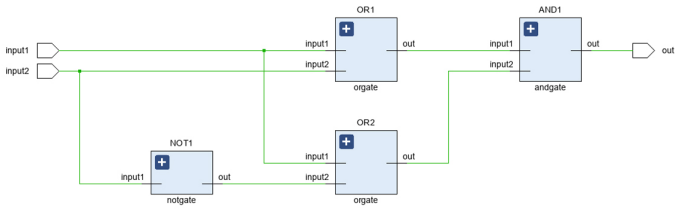

Figure 5: F_2 RTL Schematic

2.3 PART 3

In this part, we determined the dual of the theorem and then, implemented the functions for both sides of the dual theorem using AND, OR, NOT modules that we designed in the first part.

Theorem : $(a + ab = a)$

The dual of a Boolean expression is the expression one obtains by interchanging addition and multiplication and interchanging 0's and 1's.

Dual of $a + ab = a$ is $a.(a + b) = a$.

$$\begin{aligned} a.(a+b) &= a && \text{Original equation} \\ a.a + a.b &= a && \text{Distributivity} \\ a + a.b &= a && \text{Idempotency} \end{aligned}$$

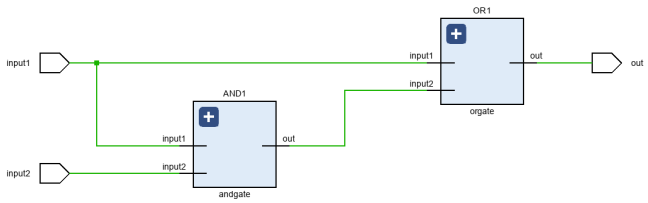

Figure 6: $a + ab = a$ RTL Schematic

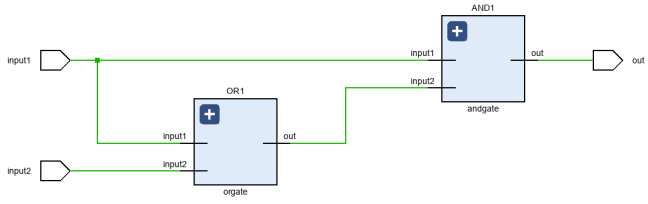

Figure 7: $a.(a + b) = a$ RTL Schematic

2.4 PART 4

In this part, we determined the complement of F_3, then implemented the circuit using AND, OR, NOT modules that we designed in the first part.

$$F_3(a, b, c) = ab + a'c$$

Complementary of function $F = ab + a'c$ is $F' = (a' + b').(a + c')$, De Morgan theorem was used to calculate this expression.

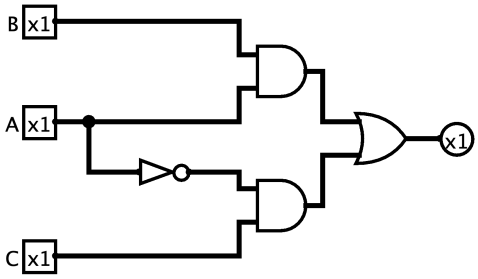

Figure 8: F_3 Circuit

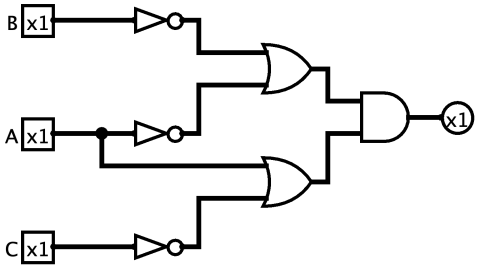

Figure 9: F_3' Circuit

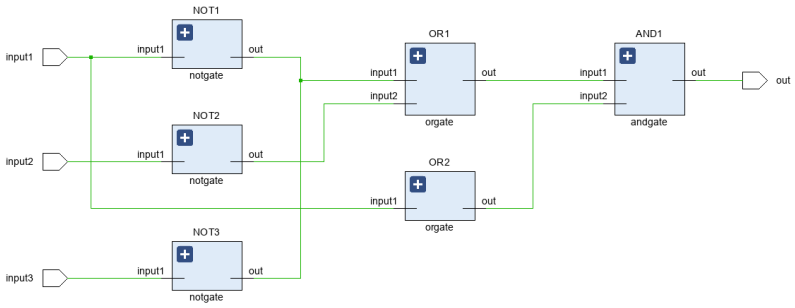

Figure 10: F_3' RTL Schematic

2.5 PART 5

In this part, we implemented the simplified expression of F_4 using AND, OR, NOT modules that we designed in the first part.

$$F_4(a, b, c, d) = U_1(1, 2, 5, 6, 9, 10, 13, 14)$$

ab \ cd	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

Figure 11: Karnaugh Map of F_4

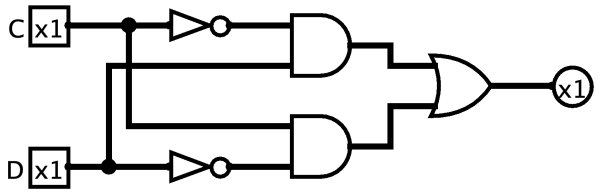

Figure 12: F_4 Circuit

According to the table, the simplified equation is $F_4 = c.d' + c'.d$

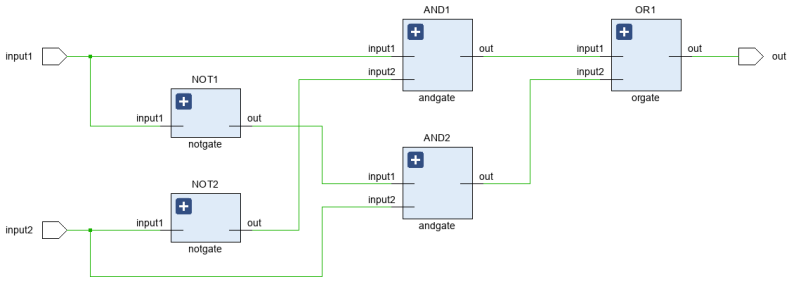

Figure 13: F_4 RTL Schematic

3 RESULTS [15 points]

3.1 PART 1

In this part, we implemented AND, OR, and NOT gates. The simulation results show that we made them correctly.

3.1.1 AND GATE

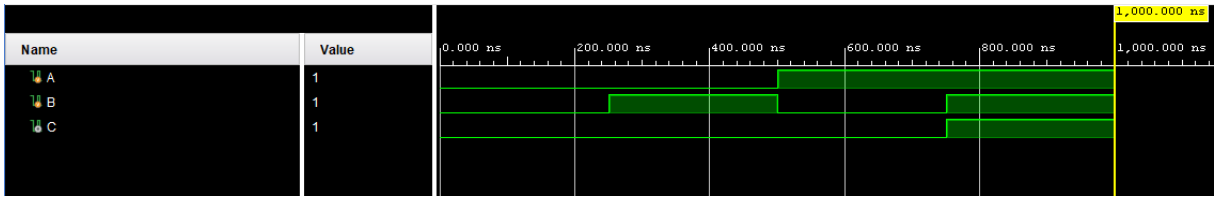

Figure 14: AND gate simulation

3.1.2 OR GATE

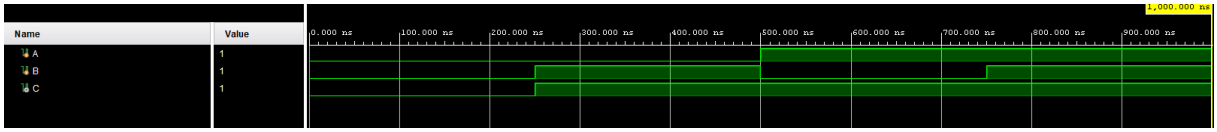

Figure 15: OR gate simulation

3.1.3 NOT GATE

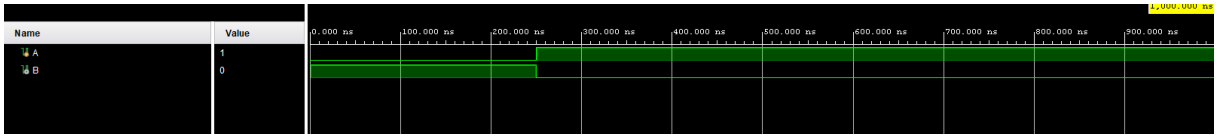

Figure 16: NOT gate simulation

3.2 PART 2

In this part, we showed that F_1 and F_2 are equal to a by implementing them using AND, OR, NOT gate we designed in the first part. In F_1, when a is 0, the output is 0 regardless of what b is. Also, when b is 1, the output is 1 regardless of what b is. That shows the validness of the theorem. The results are the same for both F_1 and F_2.

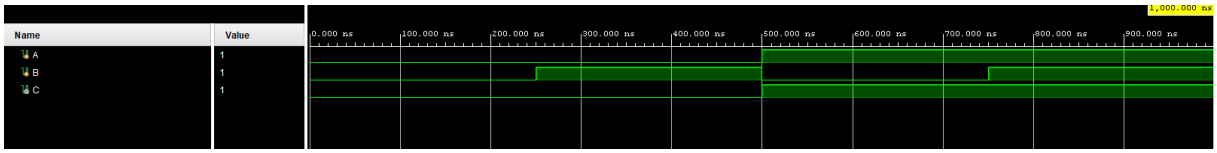

Figure 17: F_1 simulation

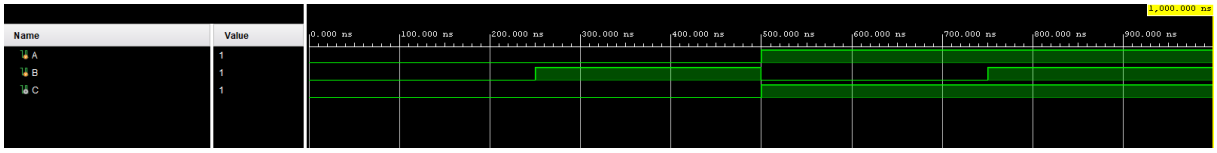

Figure 18: F_2 simulation

3.3 PART 3

In this part, we showed that the dual of an expression is the same as the original expression. Theorem is $(a + ab = a)$. When a is 0 and b regardless of b, the output is equal to 0. When a is 1 and b regardless of b, the output is equal to 1. Dual of the theorem is $a.(a + b) = a$. When a is 0 and b regardless of b, the output is equal to 0. When a is 1 and b regardless of b, the output is equal to 1. That proves our expectations about the equality of dual expressions.

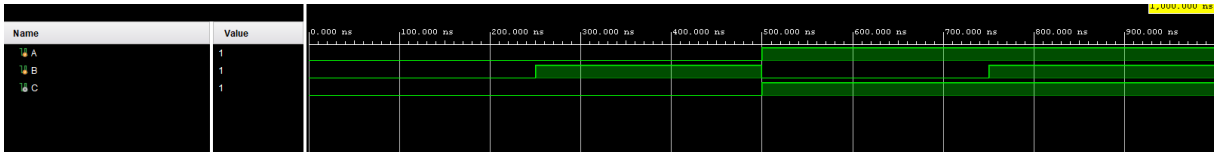

Figure 19: F_3 simulation

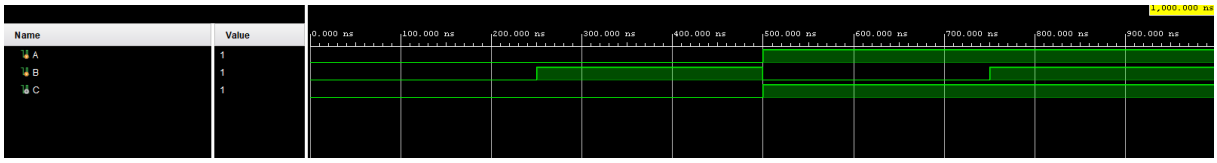

Figure 20: F_3' simulation

3.4 PART 4

In this part, we showed that the simulation results are the same as the truth table we made.

a	b	c	F_3	F'_3
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	1	0

Figure 21: Truth table for F_3 and F_3'

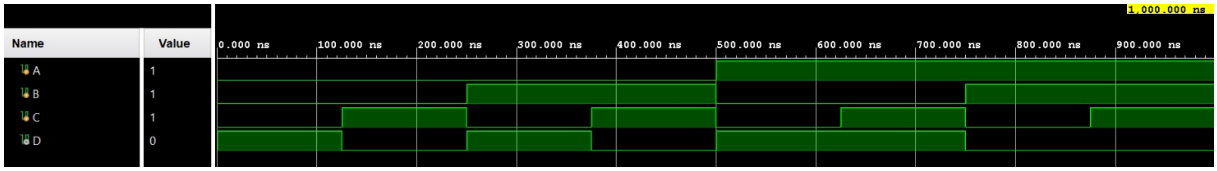

Figure 22: F_3' simulation

3.5 PART 5

In this part, we used the simplified expression that we got from the Karnaugh map. When c and d are 0, the output is 0. When c is 0 and d is 1, the output is 1. When c is 1 and d is 0, the output is 1. When c and d are 1, the output is 0. The simulation results are the same as what we expected.

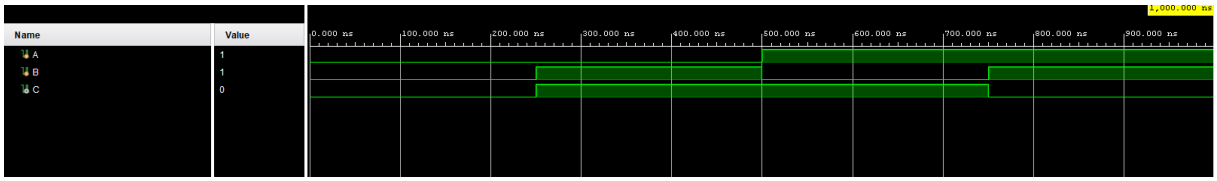

Figure 23: F_4 simulation

4 DISCUSSION [25 points]

In Part 1, we implemented AND, OR, NOT modules to use in the following parts. A, B are the input variables and C is the output variable.

$C = A\&B$ equation used for AND gate
$C = A|B$ equation used for OR gate
$C = \sim A$ equation used for NOT gate

In Part 2, we had two expressions:$F_1(a, b) = a+ab = a$ and $F_2(a, b) = (a+b)(a+b0) = a$. Firstly, we proved the equalities using the axioms of Boolean algebra. Then, using the gate modules we implemented in Part 1, we designed the expressions as logic circuits. Our code looked like;

```
andgate AND1(.input1(input1), .input2(input2), .out(araKablo1));
orgate OR1(.input1(input1), .input2(araKablo1), .out(out));
```

We defined the result of the AND gate as araKablo1 (a.b) and used it as a input in the OR gate. The result of our OR gate was as expected: $a + a.b$. After implementing the expressions, we prepared test cases for them with every possible input and checked their correctness.

In Part 3, we have the equality $a+ab = a$. First, we determined its dual as $a.(a+b) = a$. We proved it using Boolean algebra and checked its correctness. Then, again using the gate modules from the Part 1, we implemented the circuit. Our code looked like;

```
orgate OR1(.input1(input1), .input2(input2), .out(araKablo1));
andgate AND1(.input1(input1), .input2(araKablo1), .out(out));
```

The OR gate summed our two inputs (a+b). The AND gate multiplied the result of the OR gate with our first input(a) and the final result was a(a+b) . After implementing the expressions, we prepared test cases for them with every possible input and checked their correctness.

In Part 4, we had the function $F_3(a, b, c) = ab + a'.c$. Firstly, we calculated its complementary (F'3) using De Morgan theorem as $F_3' = (a' + b').(a + c')$. Afterwards, we validate our expressions using a truth table. Then, again using the gate modules from the Part 1, we implemented the circuit. Our code looked like;

```
notgate NOT1(.input1(input1), .out(araKablo1));
notgate NOT2(.input1(input2), .out(araKablo2));
notgate NOT3(.input1(input3), .out(araKablo1));
orgate OR1(.input1(araKablo1), .input2(araKablo2), .out(araKablo4));
orgate OR2(.input1(input1), .input2(araKablo3), .out(araKablo5));
andgate AND1(.input1(araKablo4), .input2(araKablo5), .out(out));
```

To use the results of the gates in another gate, we defined wires. After implementing the expression, we prepared test cases for it with every possible input and checked its correctness.

In Part 5, we were given a logical function: $F(a, b, c, d) = 1(1, 2, 5, 6, 9, 10, 13, 14)$. First, we simplified the function using a Karnaugh map. Our simplified expression was $F = c.d' + c'.d$. Then, again using the gate modules from the Part 1, we implemented the circuit. Our code looked like;

```
notgate NOT1(.input1(input1), .out(araKablo1));
notgate NOT2(.input1(input2), .out(araKablo2));
andgate AND1(.input1(input1), .input2(araKablo2), .out(araKablo3));
andgate AND2(.input1(araKablo1), .input2(input2), .out(araKablo4));
orgate OR1(.input1(araKablo3), .input2(araKablo4), .out(out));
```

Since our simplified expression had only c and d variables in it, we didn't use the a and b variables and defined only two inputs. After implementing the expression, we prepared test cases for it with every possible input and checked its correctness.

5 CONCLUSION [10 points]

We faced lots of difficulties while making this experiment. We tried to install Xilinx Vivado but it was a problem that to use the software in macOS. Then we tried to learn how to use the software and how to code. It was a problem that creating files separate for each module, at the beginning of the experiment. We had to restart the software to start to experiment again a few times. We did not face any difficulty while coding the modules. But test files were quite hard to code. Especially we could not learn correct syntax fast.

In conclusion, we learned how to use Xilinx Vivado and how to code in Verilog. Then, we learned how to code basic logic gate modules such as AND, OR, NOT and implement them into logical expressions.