ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E

DIGITAL CIRCUITS LABORATORY
HOMEWORK REPORT

HOMEWORK NO : 1

GROUP NO : G5

GROUP MEMBERS:

150180025 : HAMZA OĞUZ ŞENSES

150200108 : ALP TÜRKBAYRAK

SPRING 2021

Contents

FRONT COVER

CONTENTS

1 INTRODUCTION **1**

2 Preliminary **1**

2.1 First Question 1

2.2 Second Question 4

3 Experiment **6**

3.1 Part 1 6

3.2 Part 2 15

3.3 Part 3 16

3.4 Part 4 17

3.5 Part 5 18

3.6 Part 6 19

3.7 Part 7 20

3.8 Part 8 21

3.9 Part 9 22

3.10 Part 10 23

3.11 Part 11 24

4 DISCUSSION **26**

4.1 Preliminaries 26

4.2 Part 1 26

4.3 Part 2 26

4.4 Part 3 26

4.5 Part 4 26

4.6 Part 5 26

4.7 Part 6 26

4.8 Part 7 27

4.9 Part 8 27

4.10 Part 9 27

4.11 Part 10 27

4.12 Part 11 27

5 CONCLUSION **28**

REFERENCES **29**

1 INTRODUCTION

In this assignment, we designed and simulated combinatorial logic circuits and various versions of full adders, and finally a subtractor using Verilog.

2 Preliminary

The solutions of the questions in the preliminary part are given below.

2.1 First Question

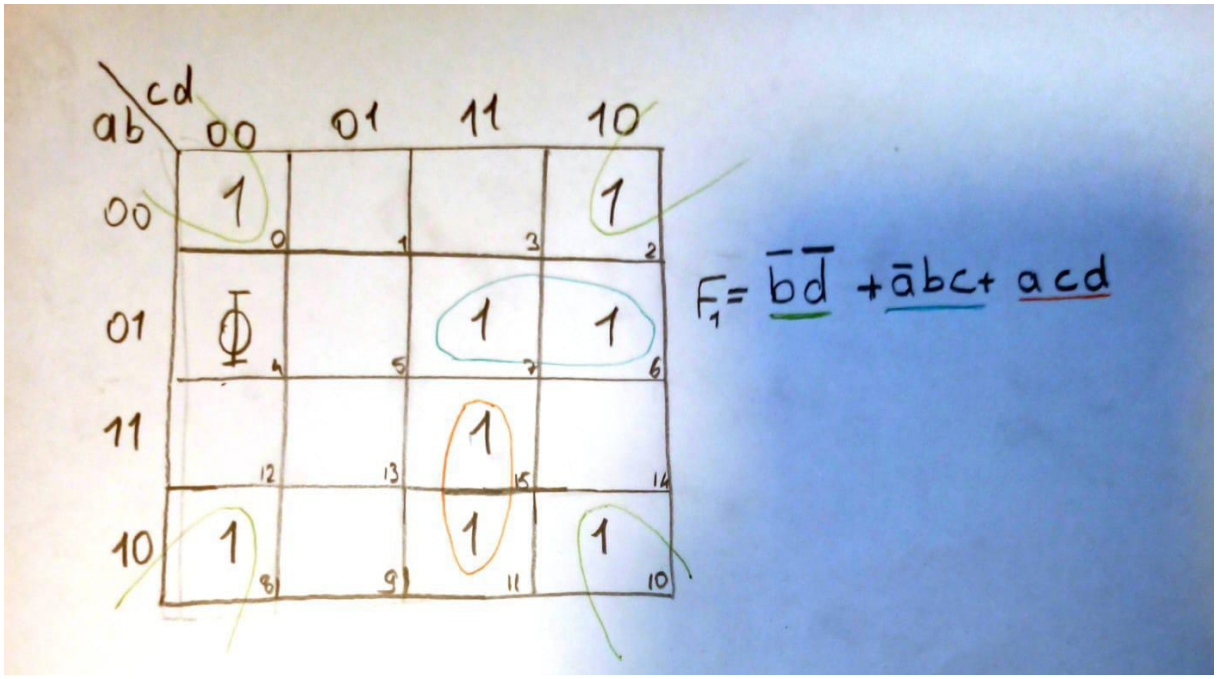
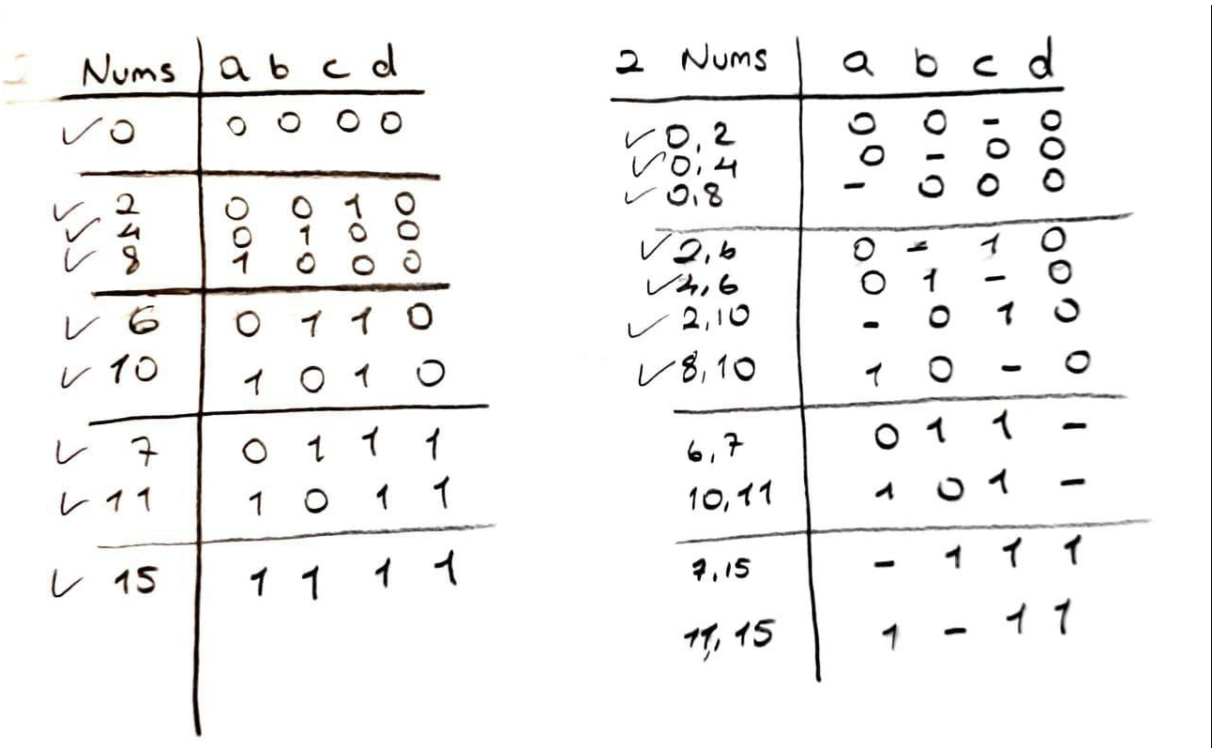


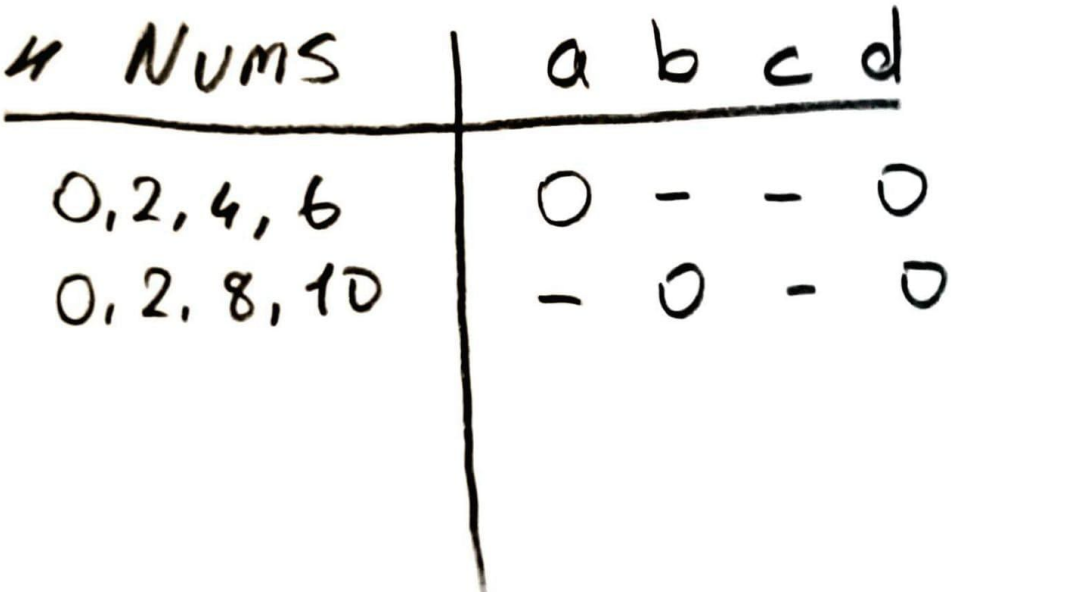
Figure 1: Question 1.a



Nums	a	b	c	d
0	0	0	0	0
2	0	0	1	0
4	0	1	0	0
8	1	0	0	0
6	0	1	1	0
10	1	0	1	0
7	0	1	1	1
11	1	0	1	1
15	1	1	1	1

2 Nums	a	b	c	d
0,2	0	0	-	0
0,4	0	-	0	0
0,8	-	0	0	0
2,6	0	-	1	0
4,6	0	1	-	0
2,10	-	0	1	0
8,10	1	0	-	0
6,7	0	1	1	-
10,11	1	0	1	-
7,15	-	1	1	1
11,15	1	-	1	1

Figure 2: Question 1.b



4 Nums	a	b	c	d
0,2,4,6	0	-	-	0
0,2,8,10	-	0	-	0

Figure 3: Question 1.b

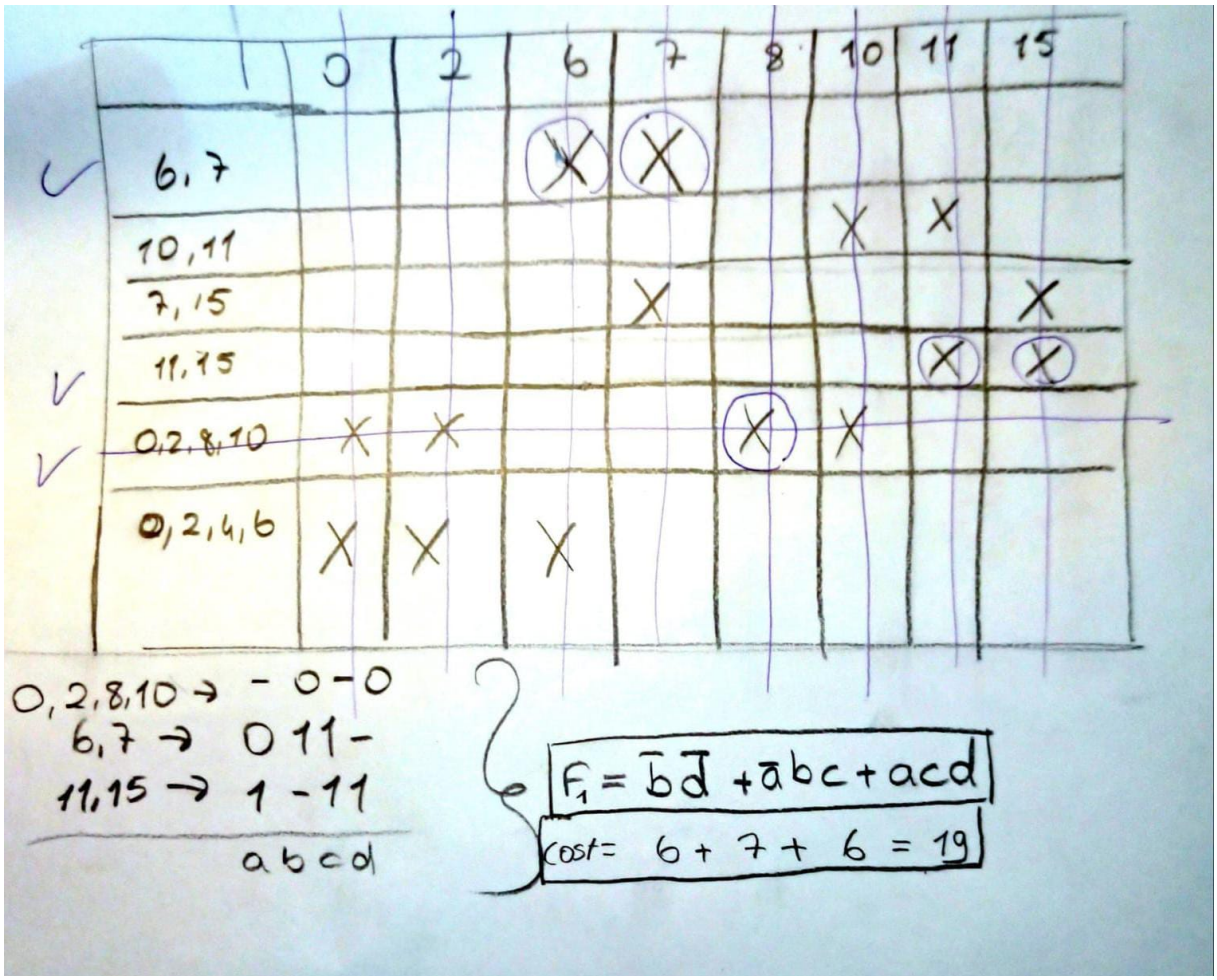


Figure 4: Question 1.c

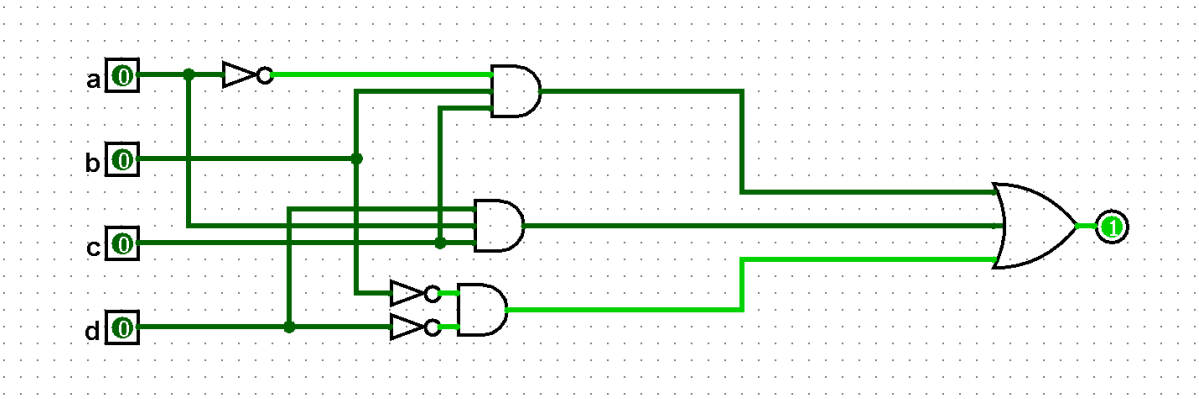


Figure 5: Question 1.d

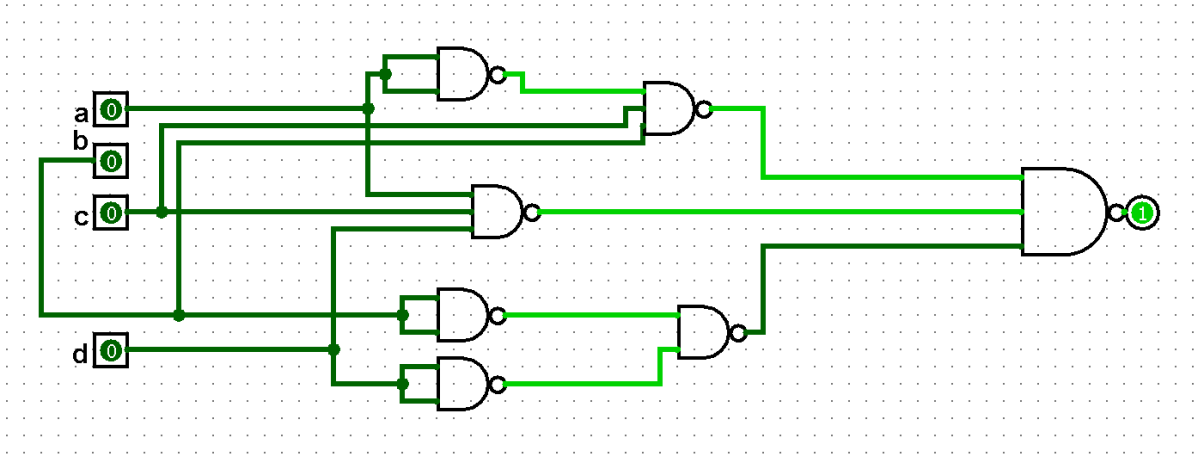


Figure 6: Question 1.e

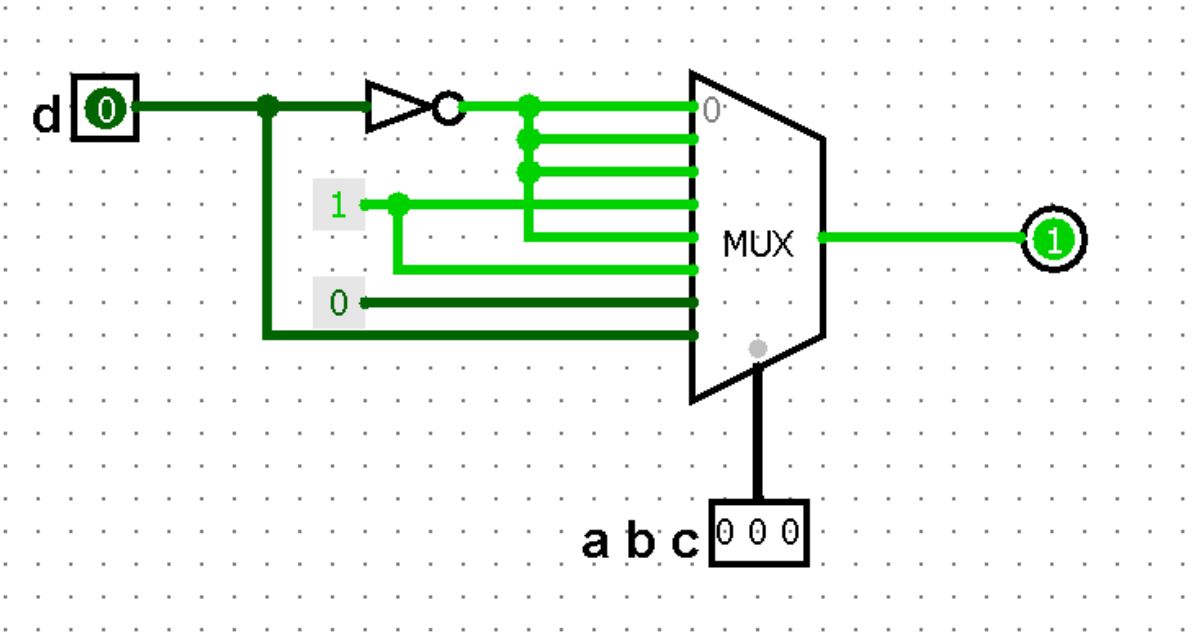


Figure 7: Question 1.f

2.2 Second Question

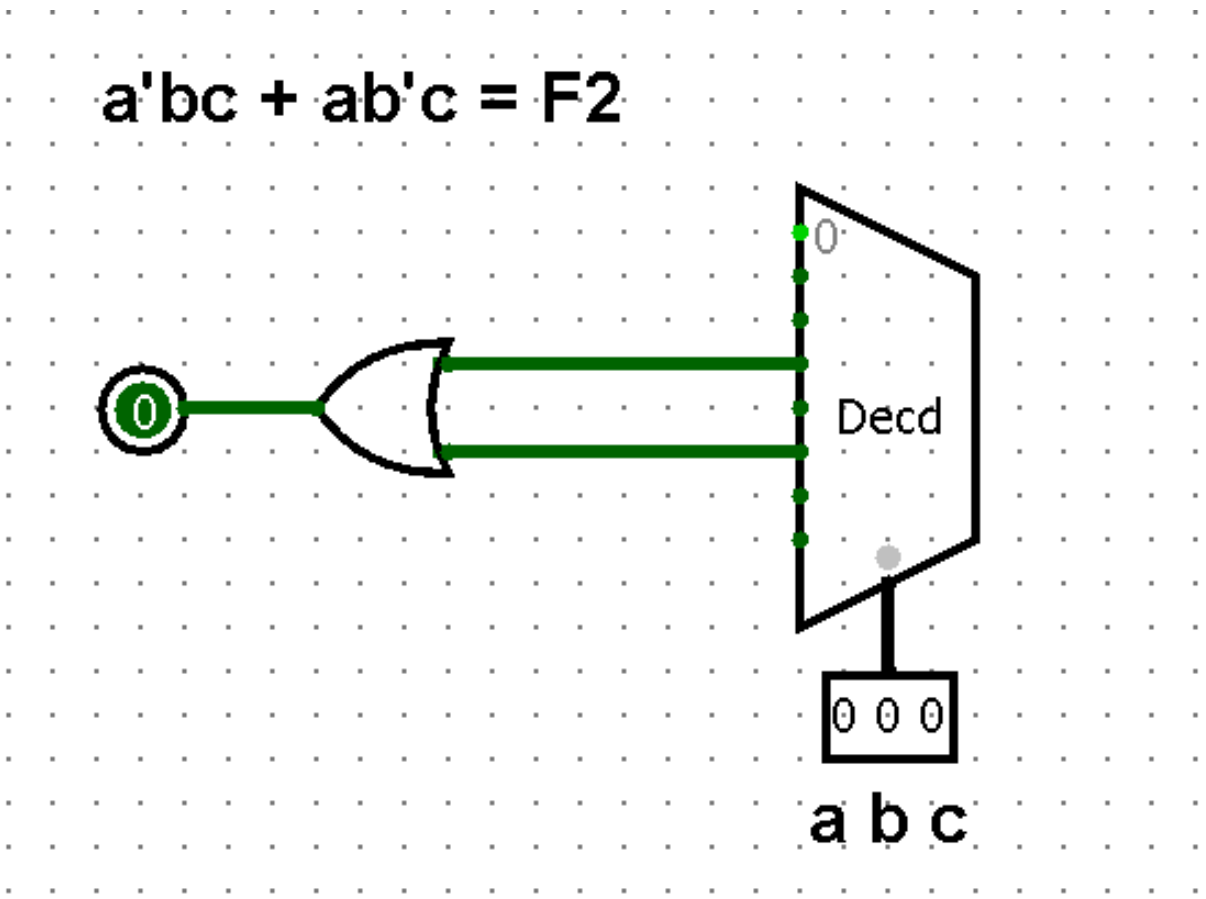


Figure 8: Question 2

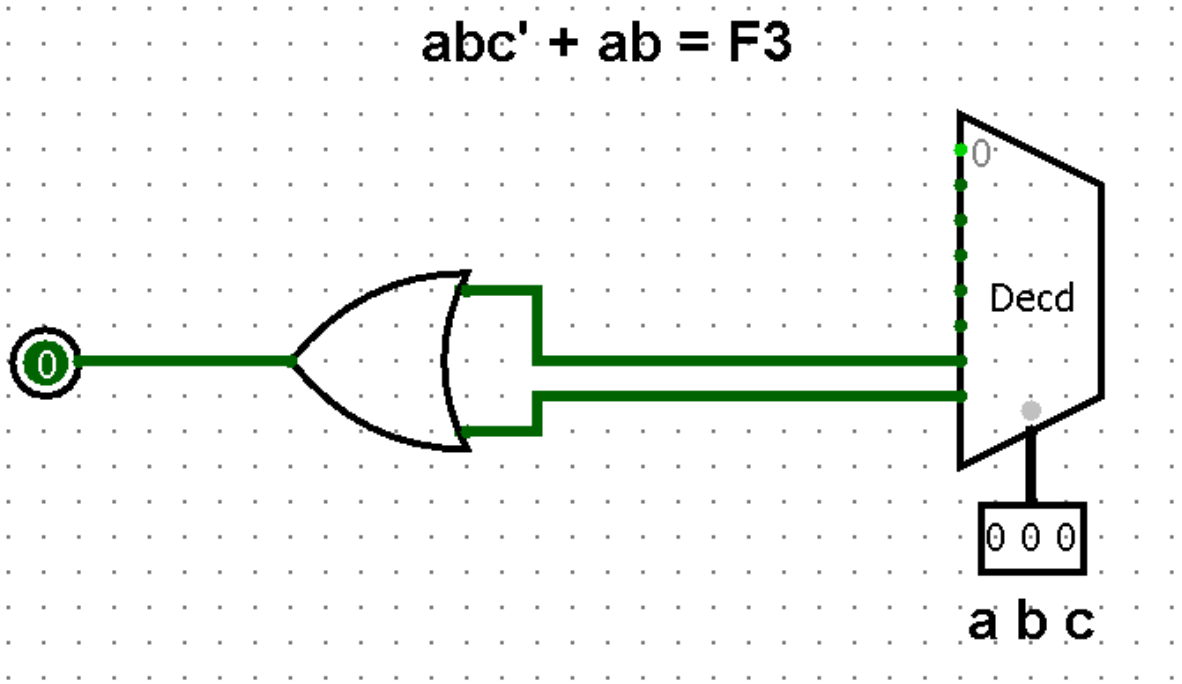


Figure 9: Question 2

3 Experiment

After designing each and every gate/IC, we ran a simulation to test their integrity. The results of these simulations are below.

3.1 Part 1

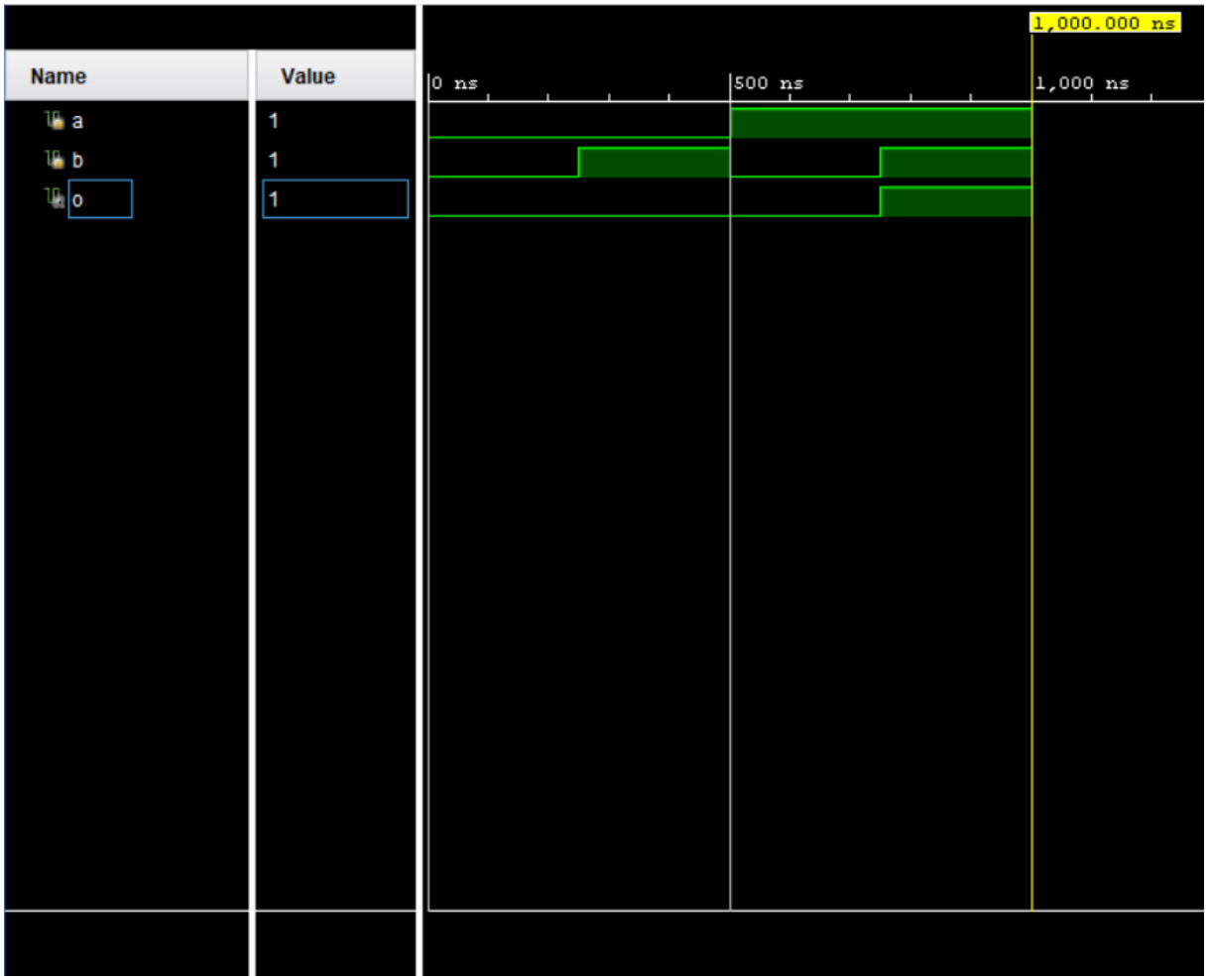


Figure 10: AND Gate Simulation

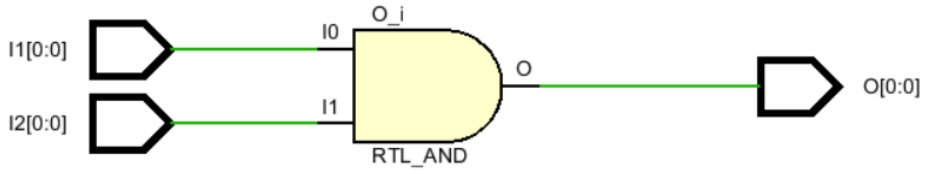


Figure 11: AND Gate design

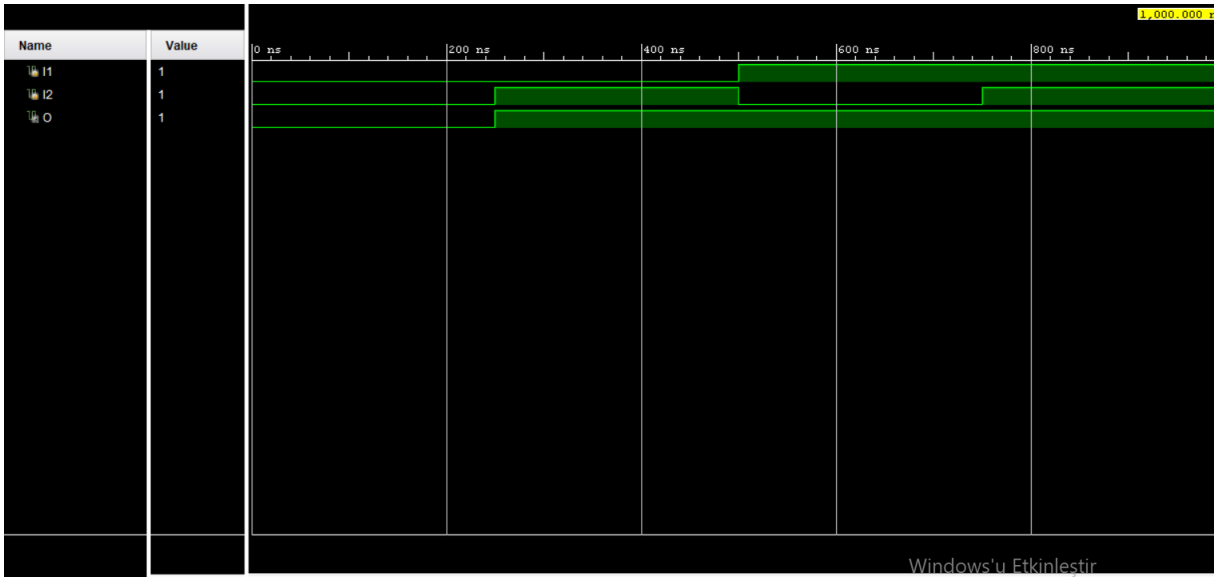


Figure 12: OR Gate Simulation

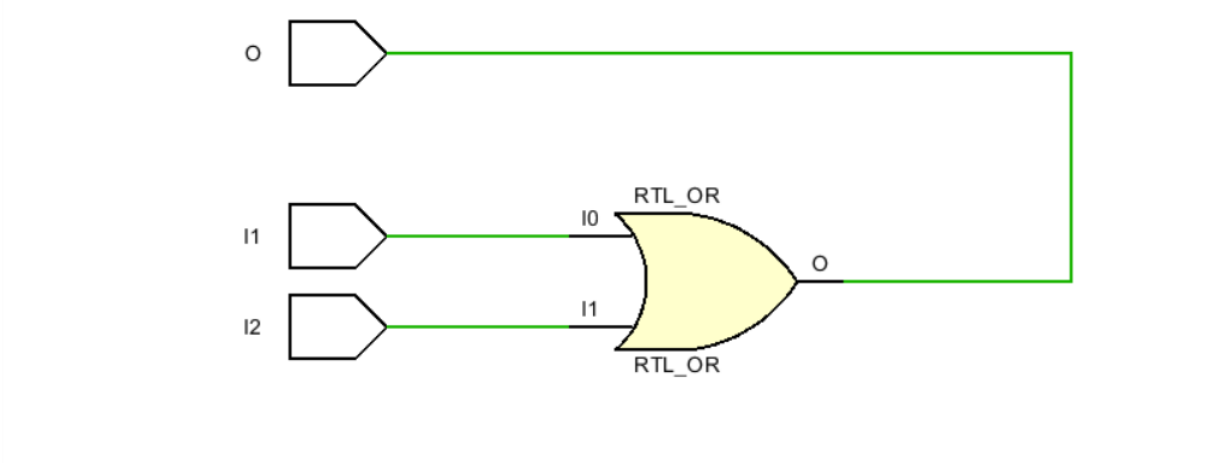


Figure 13: OR Gate Design

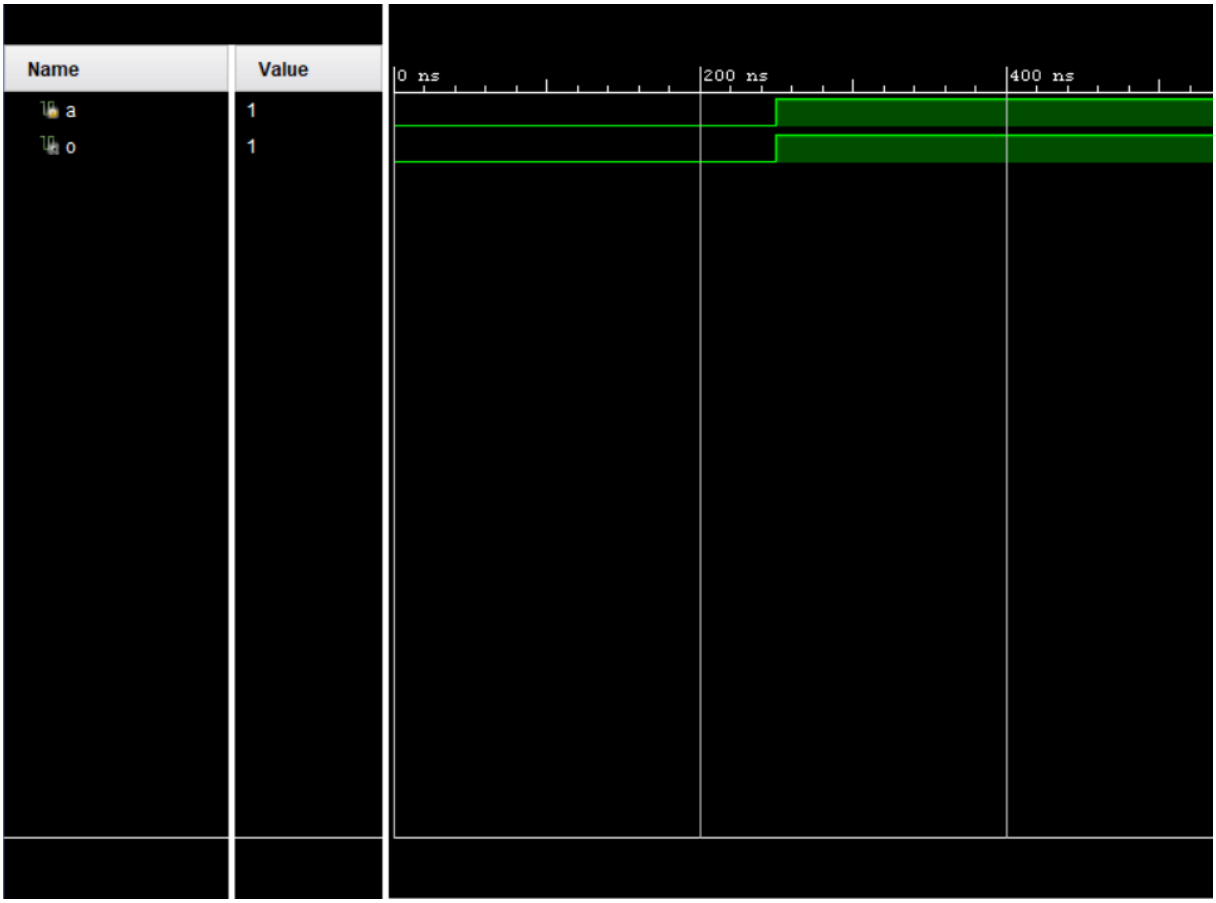


Figure 14: NOT Gate Simulation

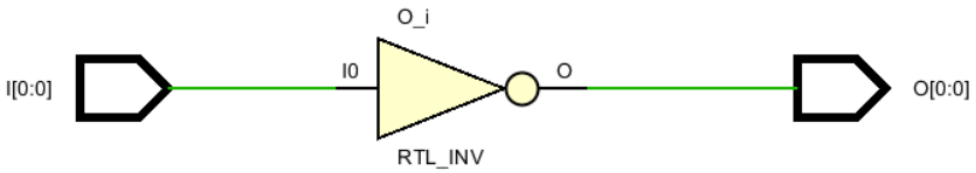


Figure 15: NOT Gate Design

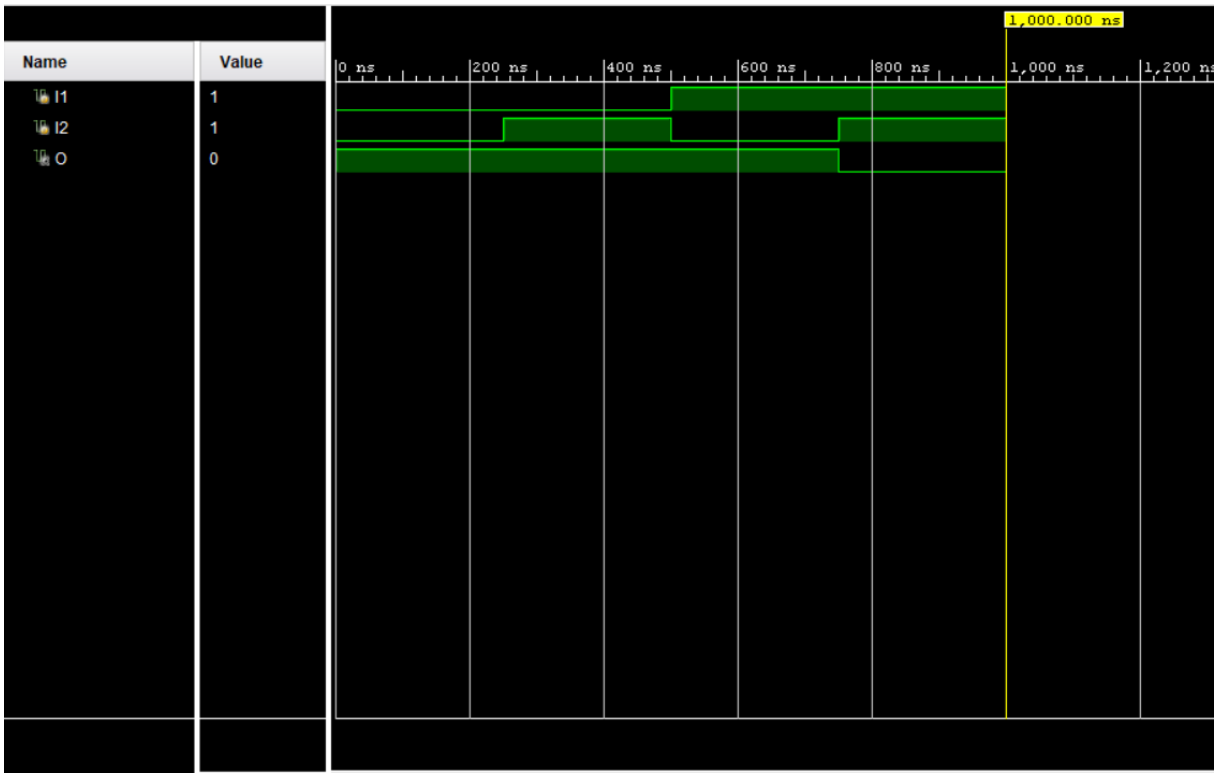


Figure 16: NAND Gate Simulation

Figure 17: NAND Gate Design

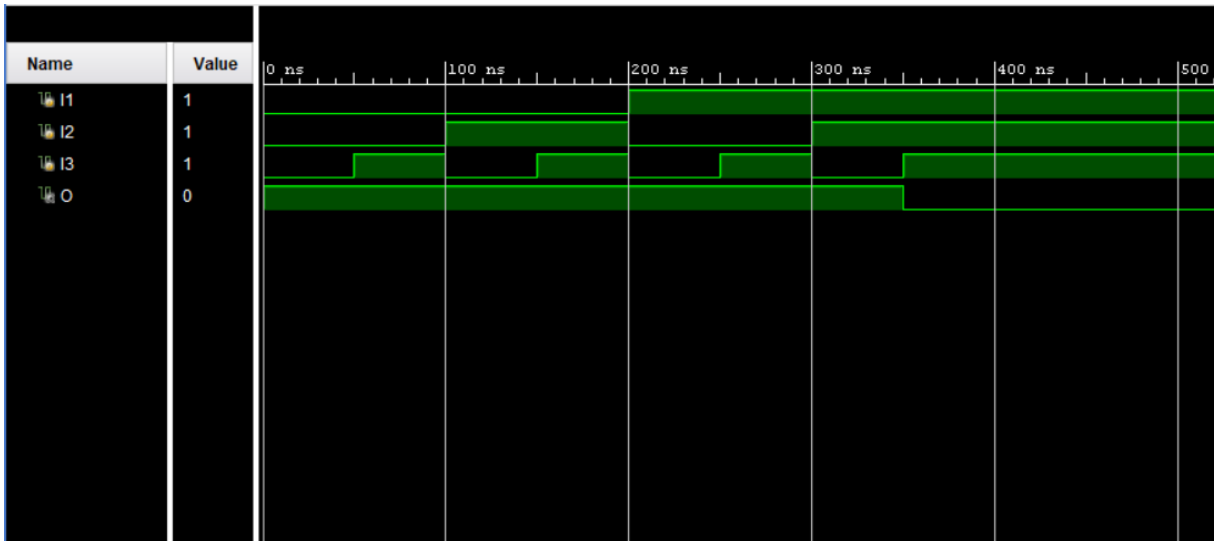


Figure 18: 3-input NAND Gate Simulation

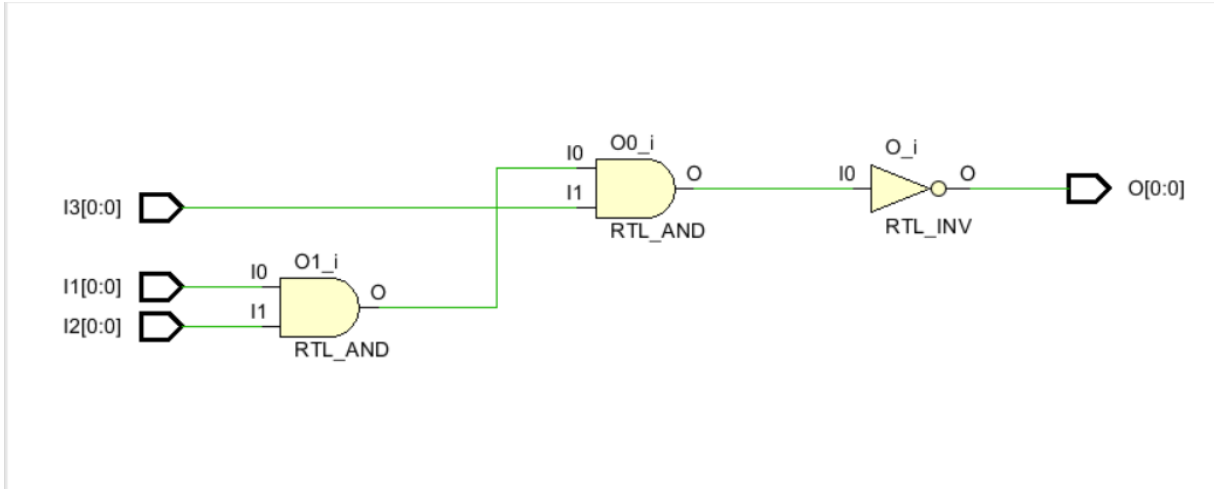


Figure 19: 3-input NAND Gate Design

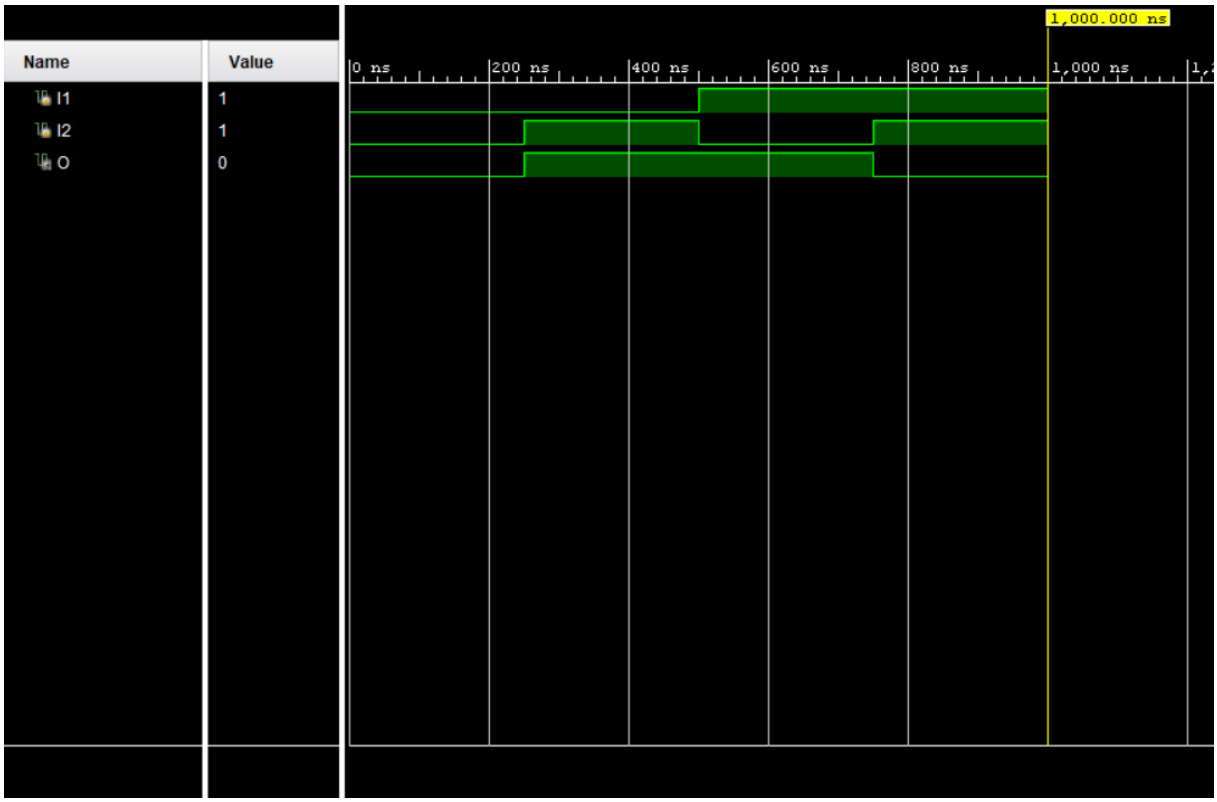


Figure 20: XOR Gate Simulation

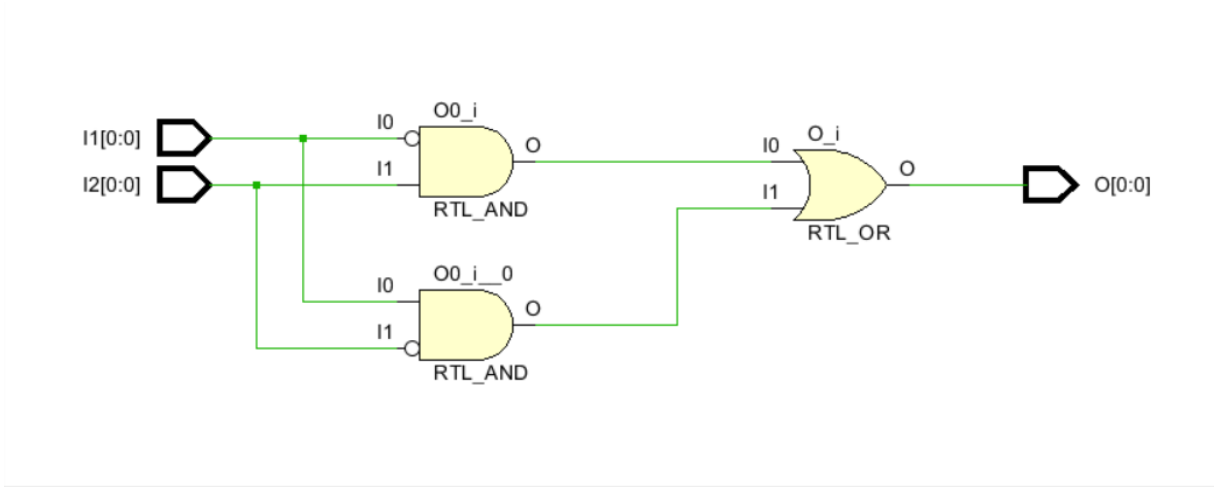


Figure 21: XOR Gate Design

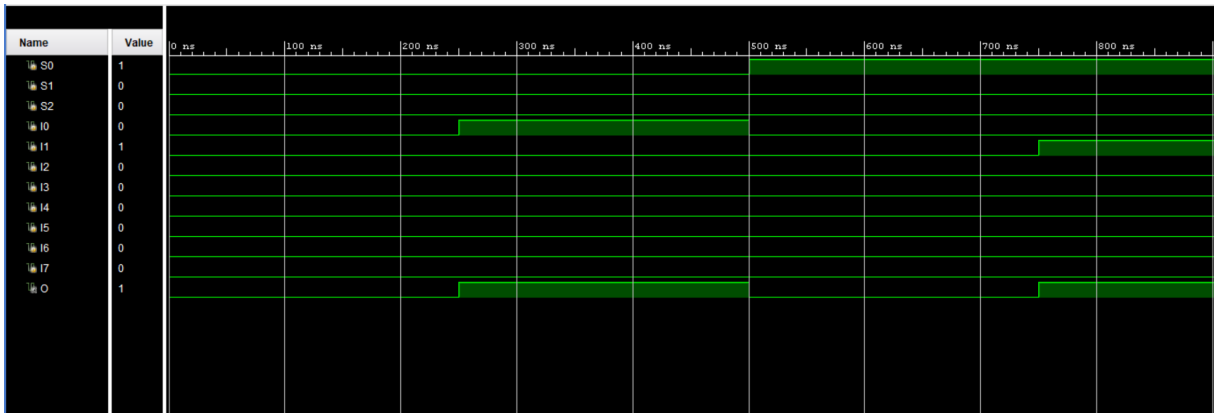


Figure 22: 8:1 MUX Simulation

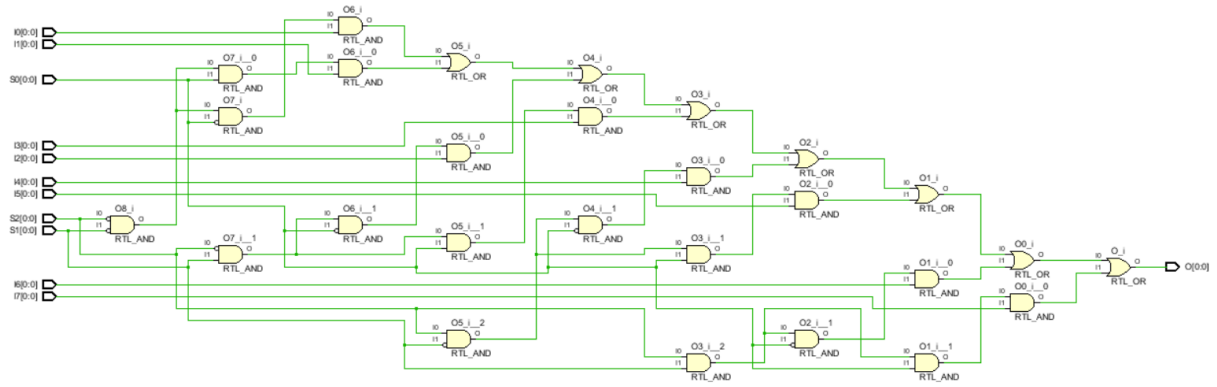


Figure 23: 8:1 MUX Gate

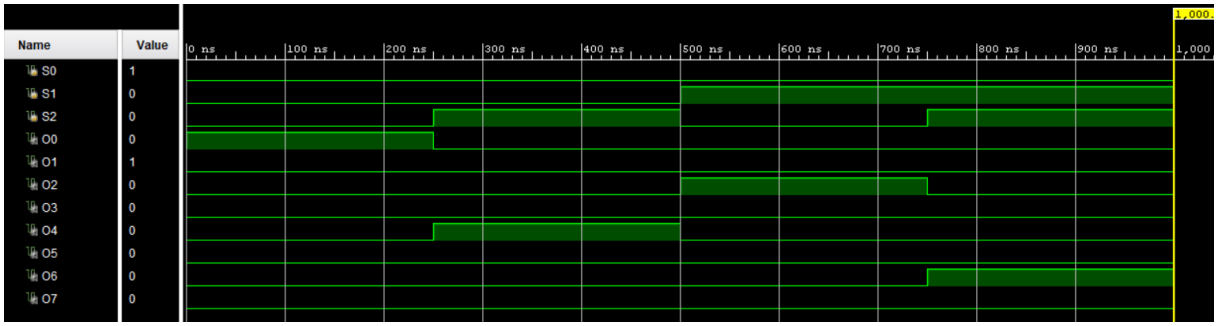


Figure 24: 3:8 DECODER Gate Simulation

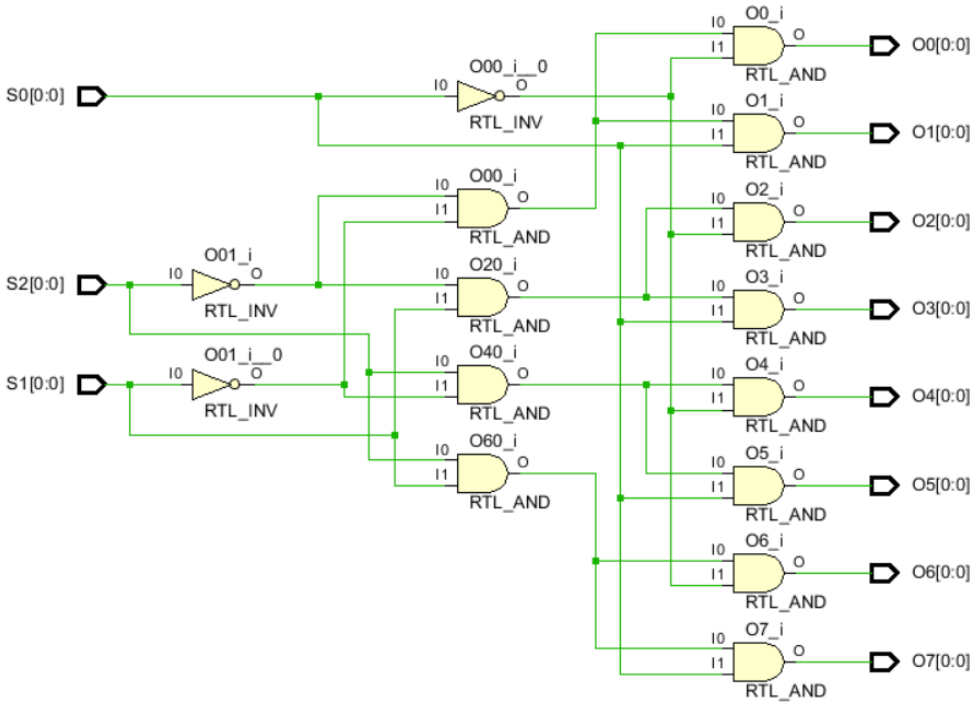


Figure 25: 3:8 DECODER Gate Design

3.2 Part 2

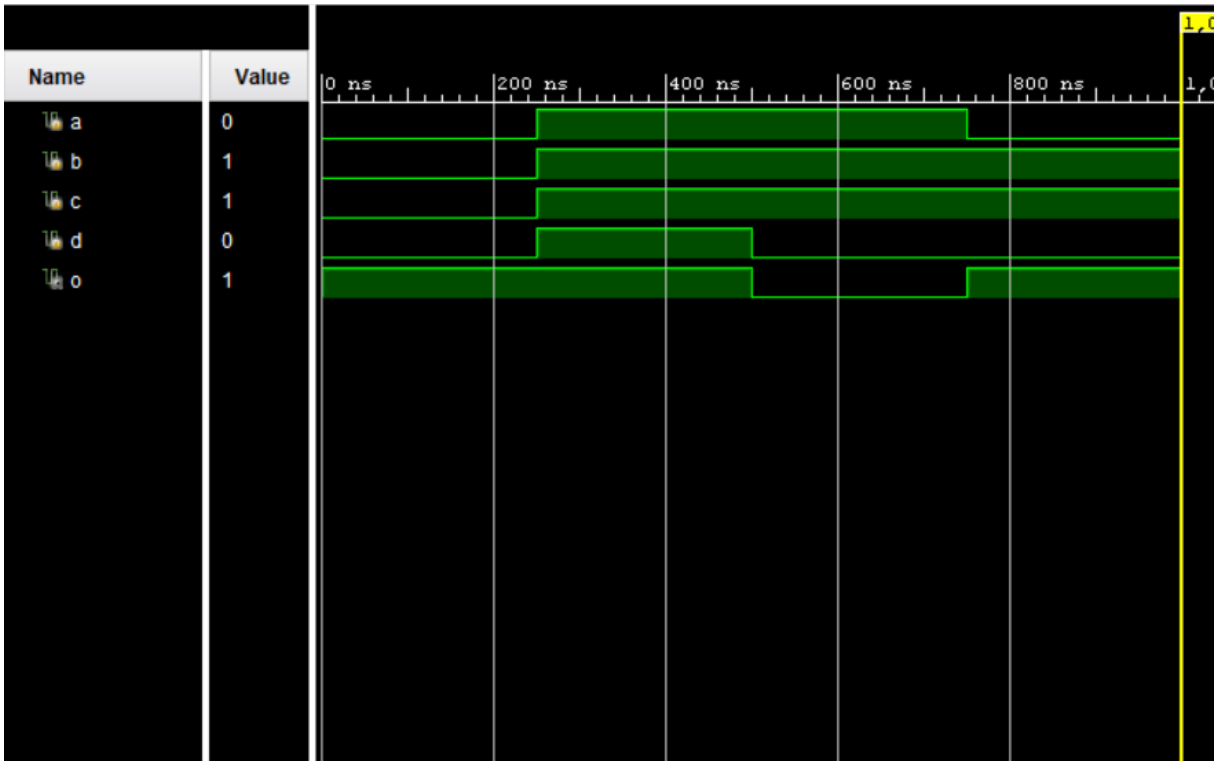


Figure 26: Part 2 Simulation

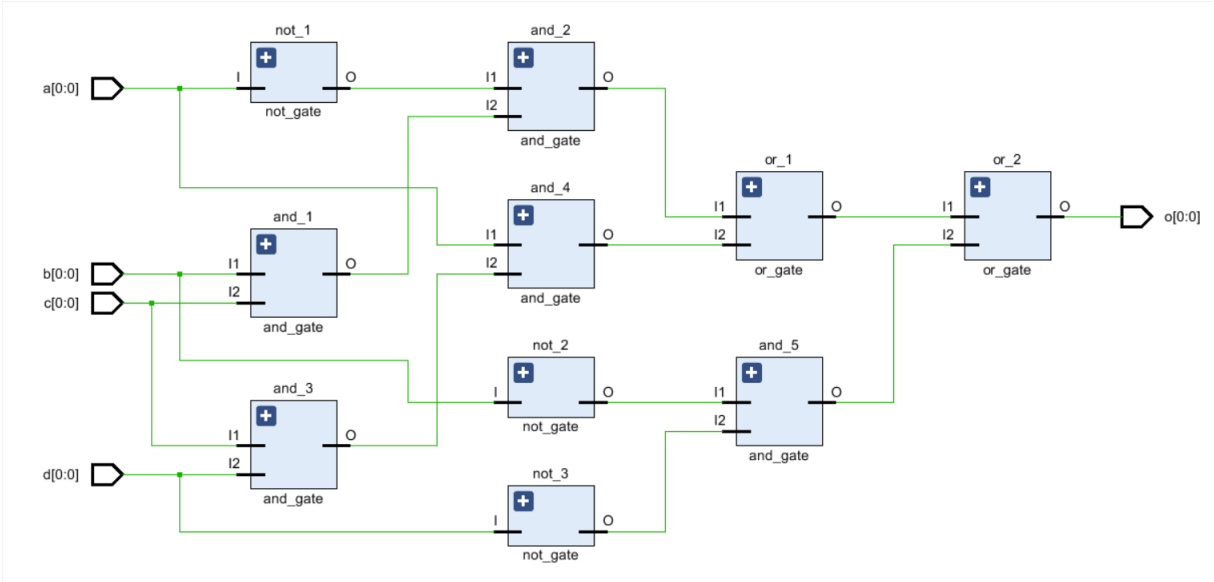


Figure 27: Part 2 Design

3.3 Part 3

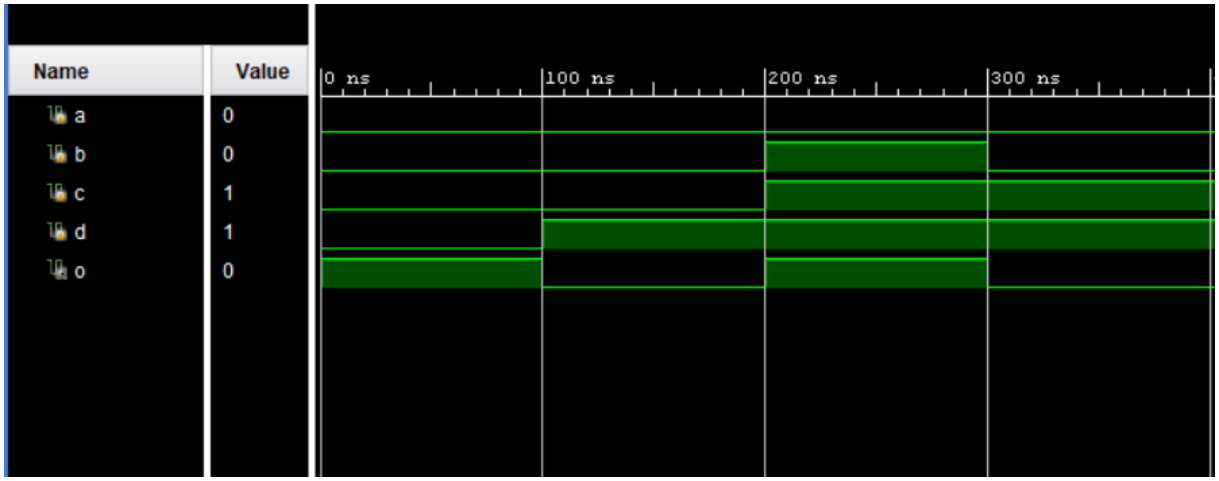


Figure 28: Part 3 Simulation

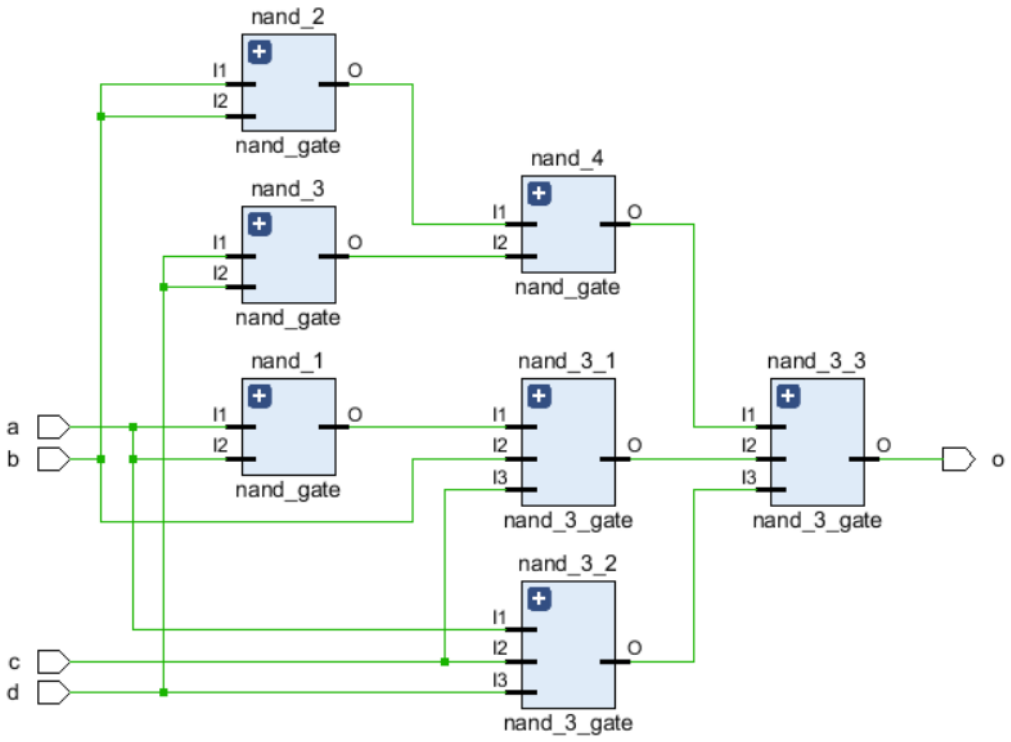


Figure 29: Part 3 Design

3.4 Part 4

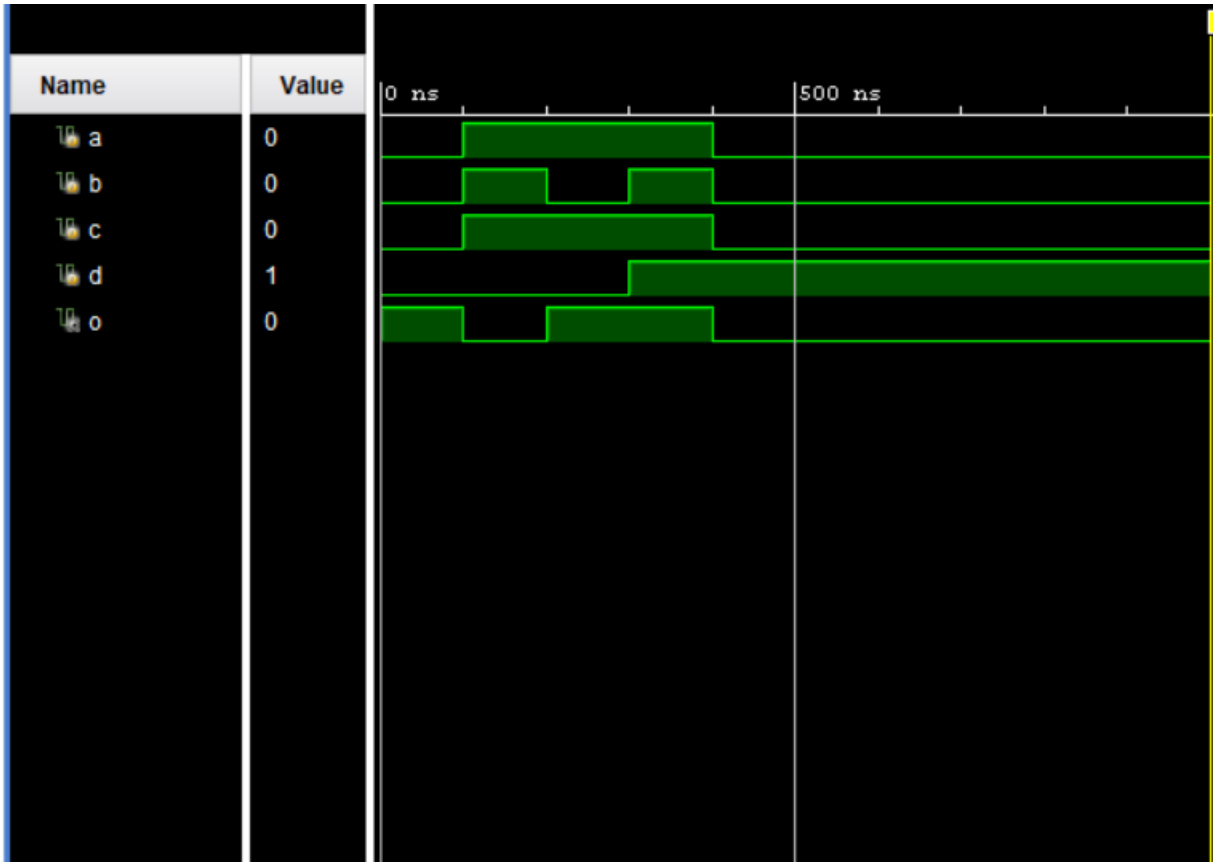


Figure 30: Part 4 Simulation

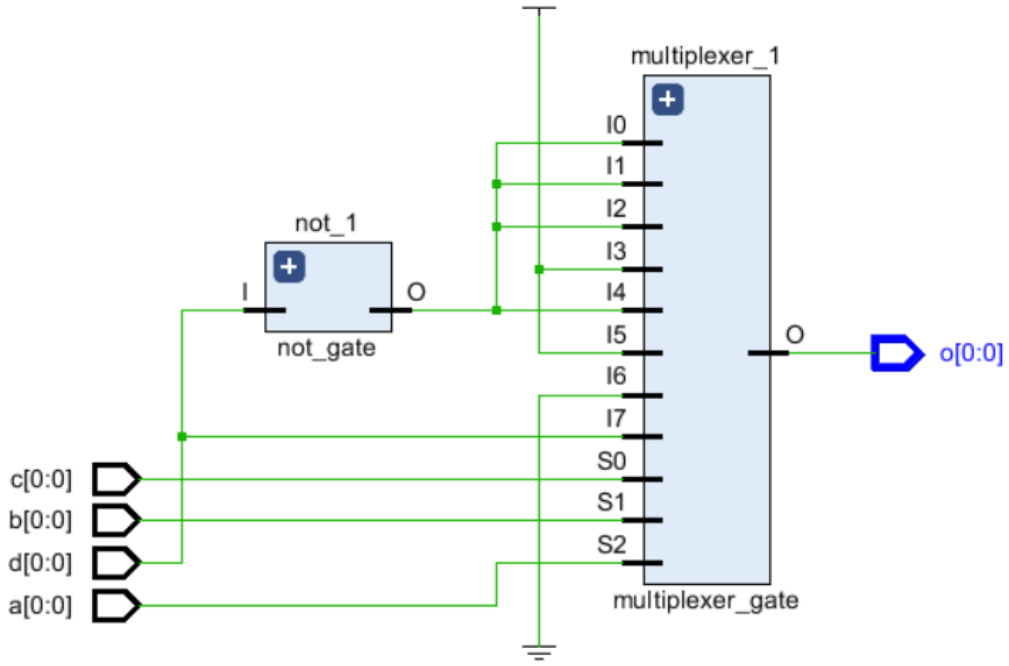


Figure 31: Part 4 Design

3.5 Part 5

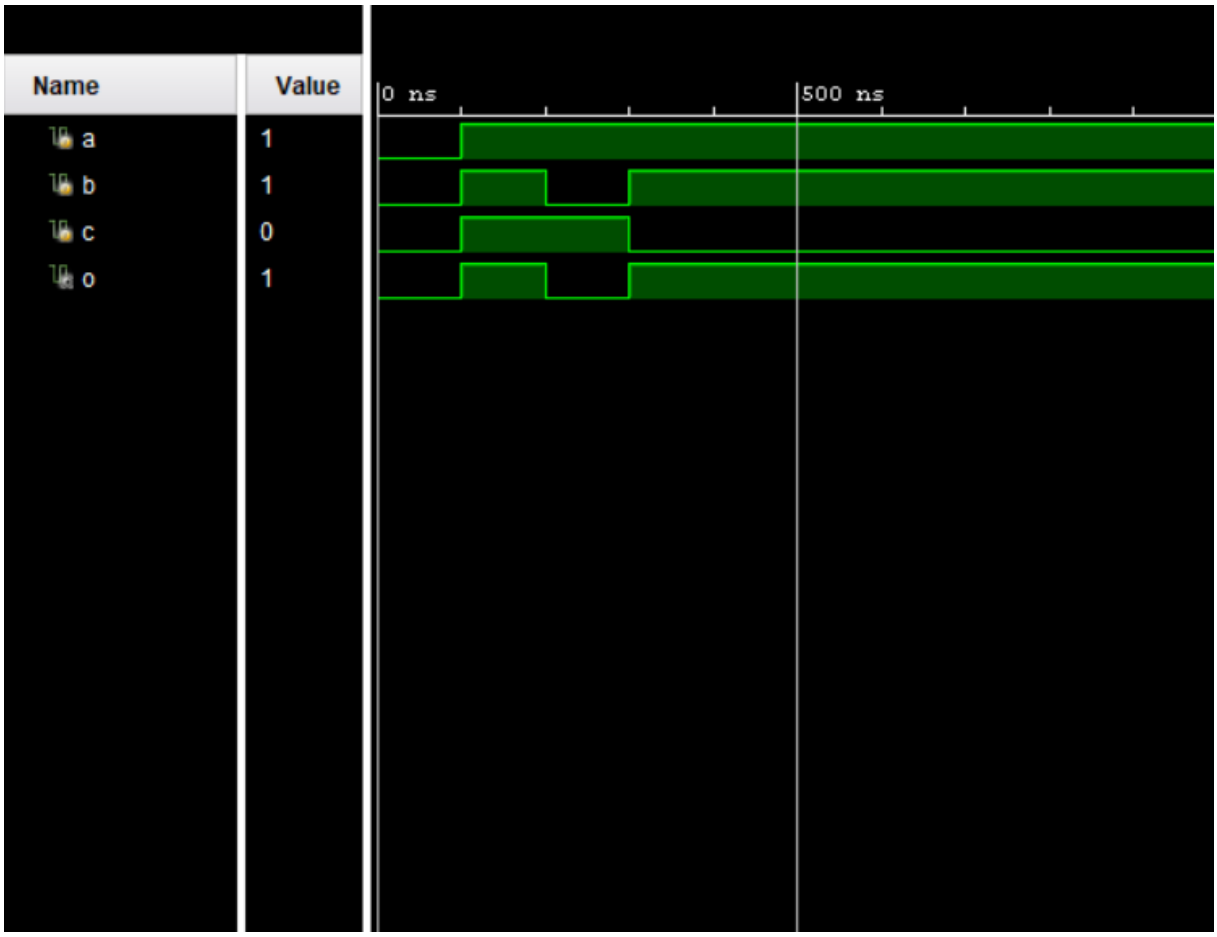


Figure 32: Part 5 Simulation

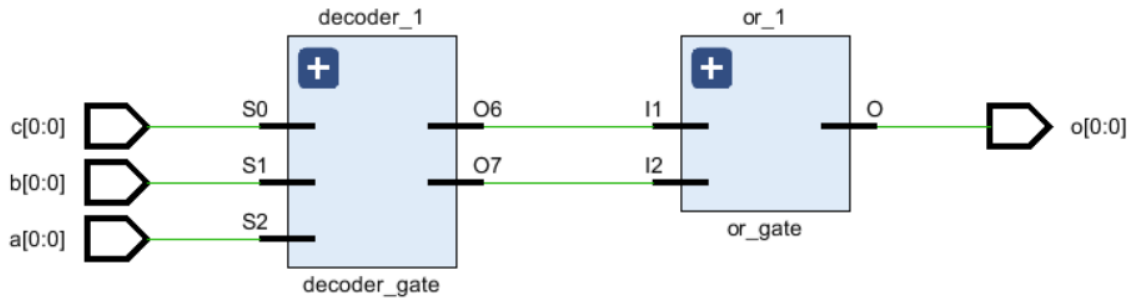


Figure 33: Part 5 Design

3.6 Part 6

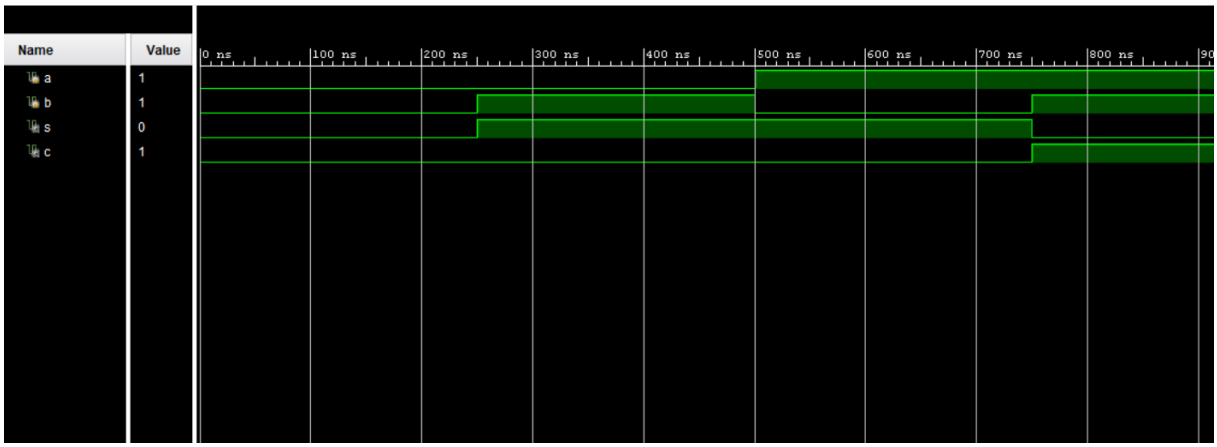


Figure 34: Part 6 Simulation

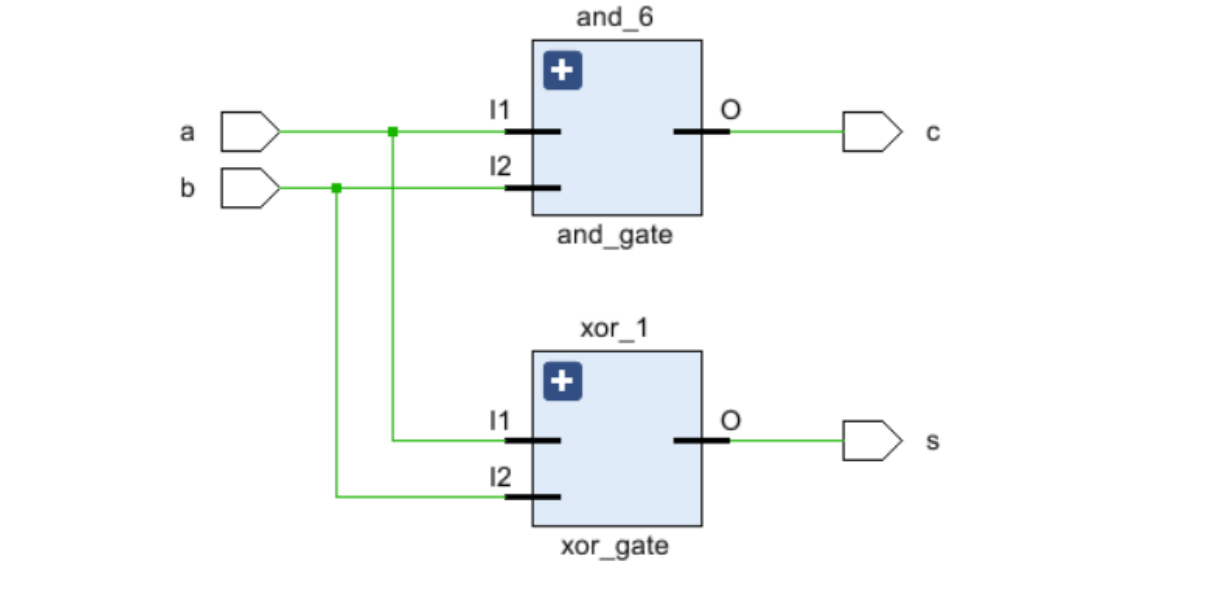


Figure 35: Part 6 Design

3.7 Part 7

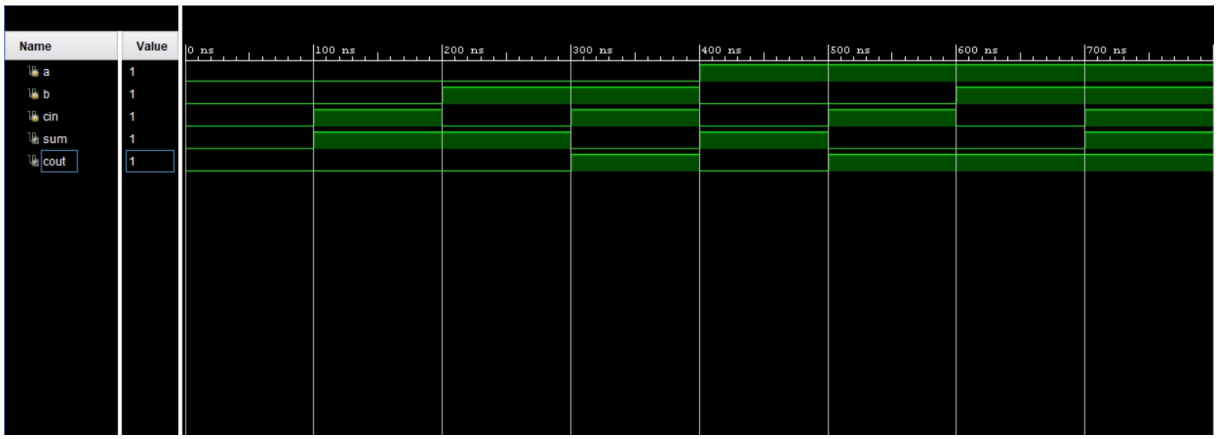


Figure 36: Part 7 Simulation

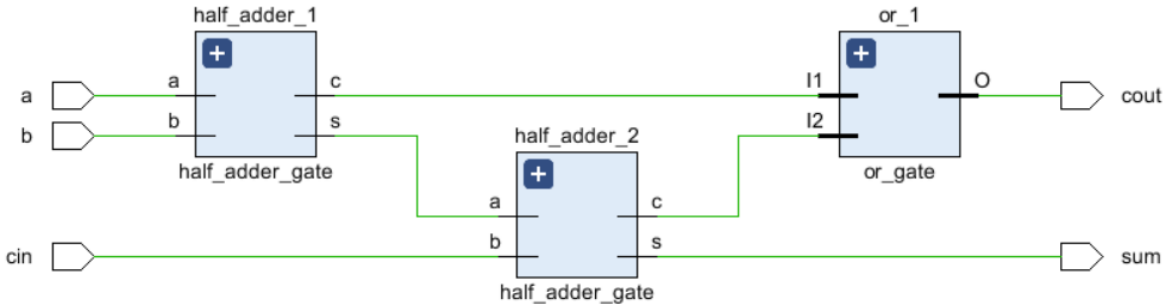


Figure 37: Part 7 Design

3.8 Part 8

Figure 38: Part 8 Simulation

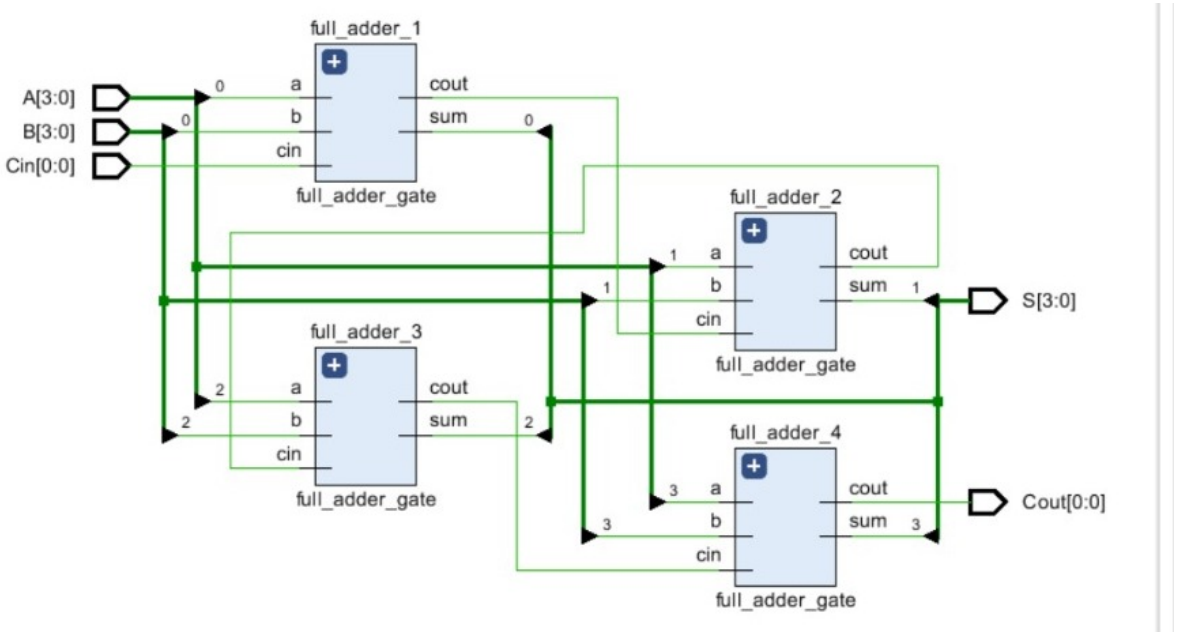


Figure 39: Part 8 Design

3.9 Part 9

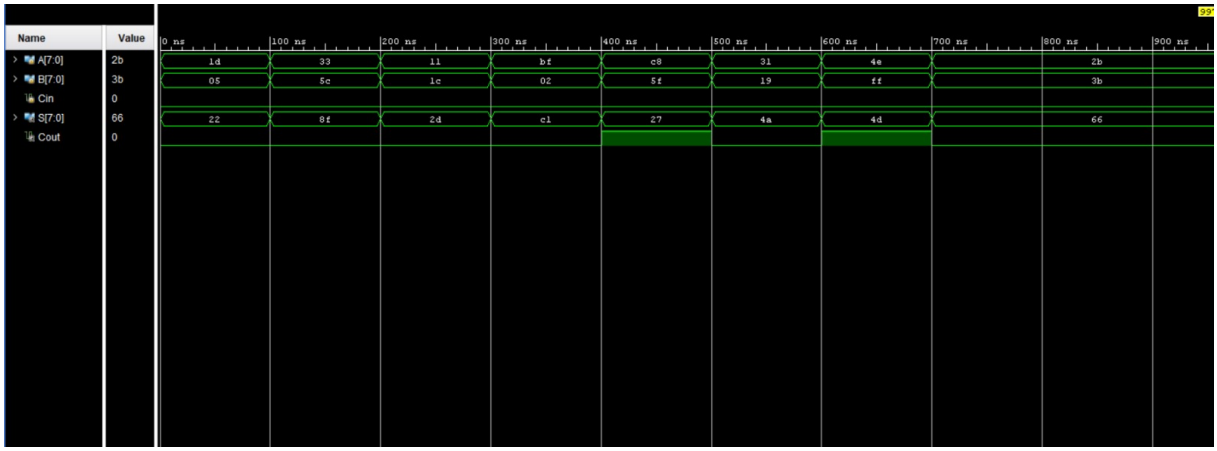


Figure 40: Part 9 Simulation

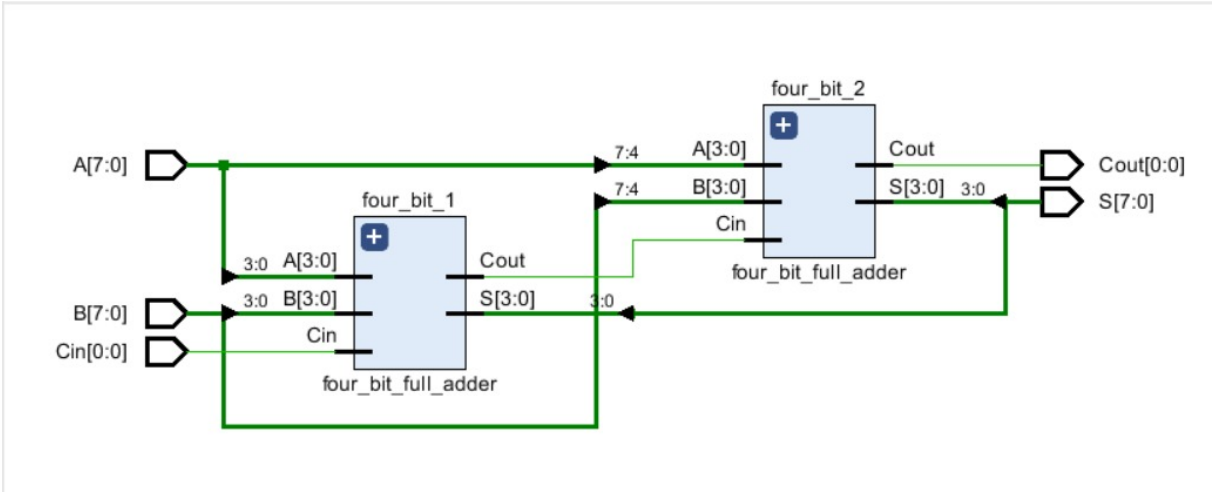


Figure 41: Part 9 Design

3.10 Part 10

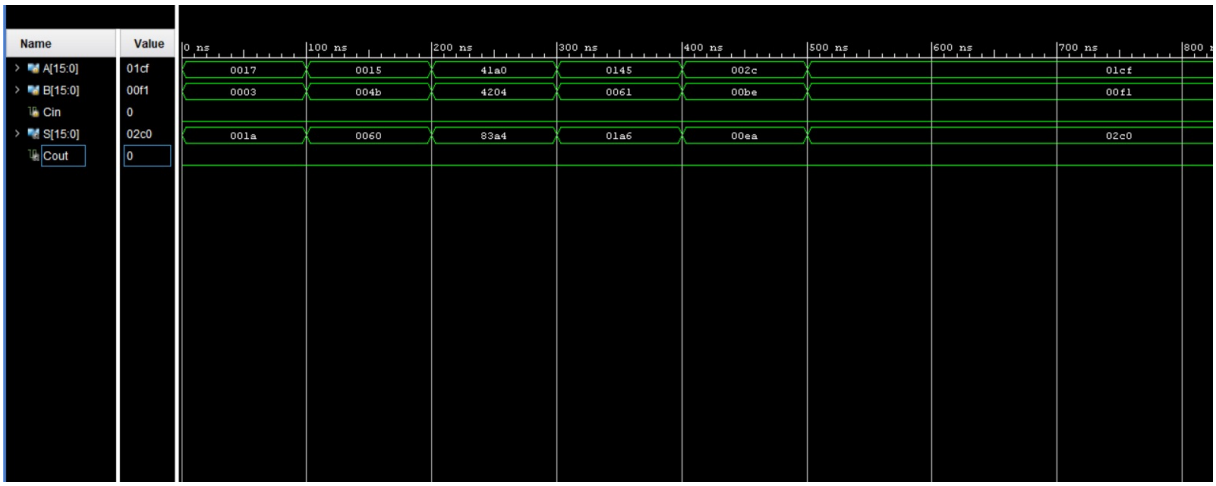


Figure 42: Part 10 Simulation

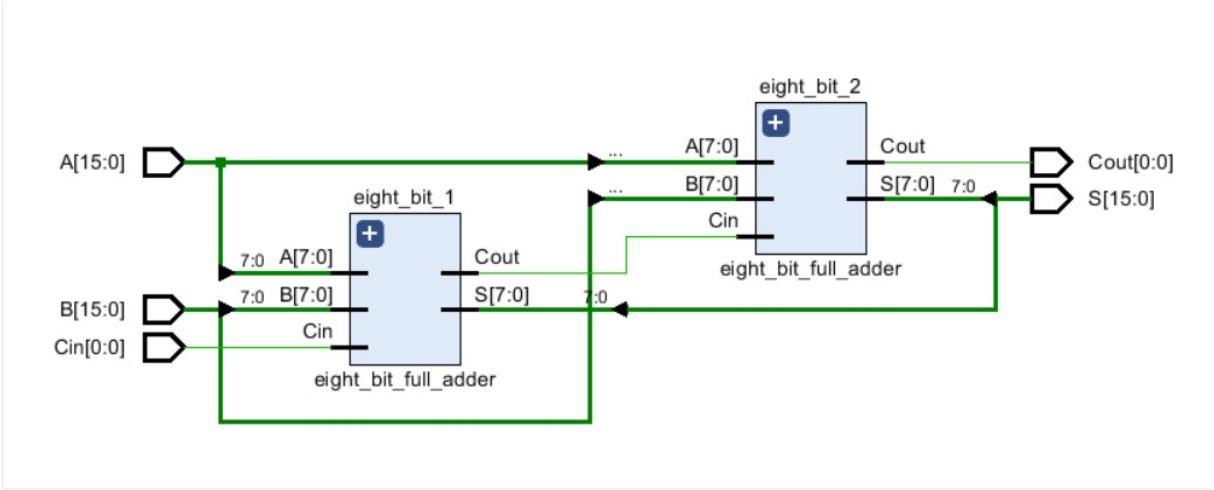


Figure 43: Part 10 Design

3.11 Part 11

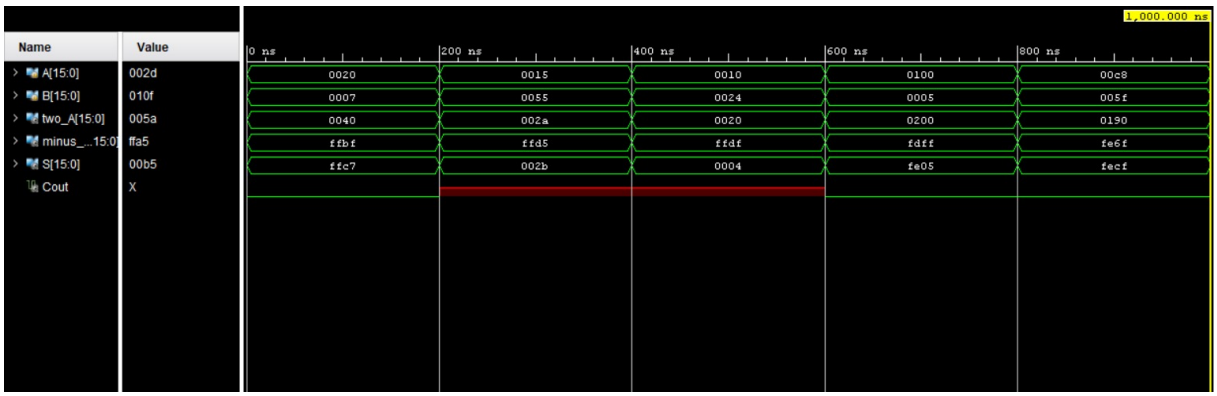


Figure 44: Part 11 Simulation

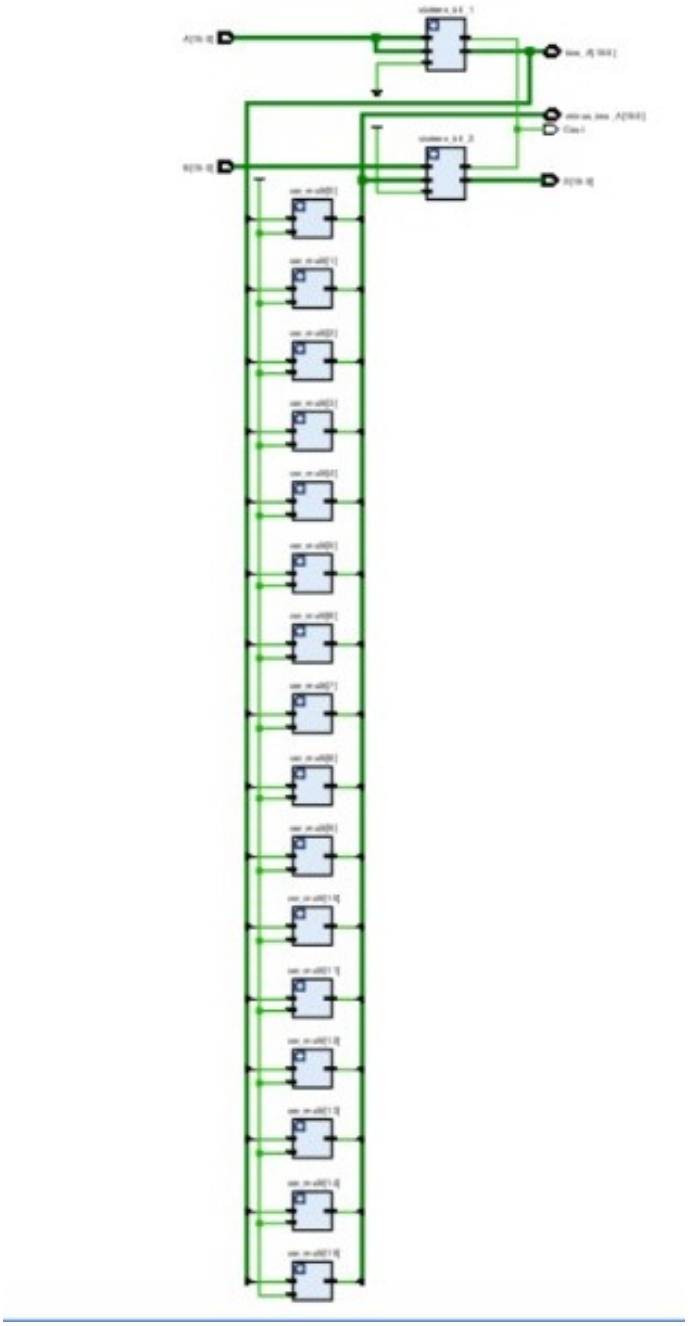


Figure 45: Part 11 Design

4 DISCUSSION

4.1 Preliminaries

For preliminaries section, in question 1, we drew a K-Map to find the prime implicants of F1. Alternatively, for part 1.b, we used the Quine-McCluskey method to do the same. After that, we constructed a prime implicant chart to find the minimum cost equation. Then, we drew the circuits for part 1.c, 1.d, 1.e and 1.f using Logisim. For question 2, Logisim was used again to draw circuits.

4.2 Part 1

For Part 1, we designed basic gate modules using simple bit-wise operations.

4.3 Part 2

For Part 2, we implemented the circuit we drew in preliminaries question 1.d section using modules we designed in Part 1.

4.4 Part 3

For Part 3, we implemented the circuit we drew in preliminaries question 1.d section using only the NAND gate module we designed in Part 1.

4.5 Part 4

For Part 4, we implemented the circuit we drew in preliminaries question 1.f section using the modules we designed in Part 1. Multiplexer was set up specifically with the inputs and the circuit in mind.

4.6 Part 5

For Part 5, we implemented the circuit we drew in preliminaries question 2 section using the modules we designed in Part 1. Decoder was set up specifically with the outputs and the circuit in mind.

4.7 Part 6

For Part 6, we implemented a new half-adder module using XOR and AND gate modules we designed. XOR's output represents the sum and AND's output represents the carry.

4.8 Part 7

For Part 7, we implemented a new full-adder module using half-adder and OR gate modules we designed. One half-adder adds cin and sum of the other while the other half adder adds the two inputs. Then their cout outputs are fed to the OR gate to get the Cout and Sum.

4.9 Part 8

For Part 8, we implemented a new 4-bit full-adder module using full-adder modules we designed. Input bits are fed one by one to the full-adders while cout of the LSB is fed to the next bit's calculating full-adder as cin.

4.10 Part 9

For Part 9, we implemented a new 8-bit full-adder module using 4-bit full-adder modules we designed. Input bits are fed four by four to the 4-bit full-adders while cout of the first adder is fed to the other adder's cin input.

4.11 Part 10

For Part 10, we implemented a new 16-bit full-adder module using 8-bit full-adder modules we designed. Input bits are fed eight by eight to the 18-bit full-adders while cout of the first adder is fed to the other adder's cin input.

4.12 Part 11

For Part 11, we implemented a new 16-bit subtractor module using 16-bit full-adder and XOR modules we designed. To calculate 2A, a 16-bit full-adder was used to add A to A. The result was named 2A. To take the complement of 2A, all of the bits were XOR'ed with 1. This is 1's complement. To take 2's complement, the other 16-bit full-adder's cin was fed as constant 1. The other input of the second 16-bit full-adder is B. Therefore the whole operation is: B + -(A + A)

5 CONCLUSION

For this homework, we had to work separately, unlike we used to do on laboratory sessions. This situation made it so that we had to work more in coordination for each part and we had to edit each other's codes. In that way, it was a great experience in terms of getting used to teamwork and reading code of your teammates.

Learning how to use Verilog/Vivado turned out to be the biggest challenge of this homework. Through research and, more importantly, trial and error, we figured the basics and went forward from there.

Some important things we learned by using Verilog/Vivado is how easy it is to code digital circuits instead of draw them using software such as Logisim. It is clear that for more complex circuitry, Verilog/VHDL is the path to choose as it is so easy to design a module and reuse it for creating more complex modules if need be (such as it was the case when converting half-adder to full-adder to 4-bit full-adder to 8-bit full-adder to 16-bit full-adder to adder-subtractor).

We learned how to run basic simulations taking advantage of Vivado's not so user-friendly interface options which took a lot of time getting used to and reading errors.

REFERENCES