ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E
DIGITAL CIRCUITS LABORATORY HOMEWORK REPORT

HOMEWORK NO	:	1
HOMEWORK DATE	:	28.03.2022 - 10.04.2022
LAB SESSION	:	FRIDAY - 14.00
GROUP NO	:	G08

GROUP MEMBERS:

150200705	:	HELİN ASLI & AKSOY
150180089	:	HARUN & ÇİFCİ

SPRING 2022

Contents

1 INTRODUCTION [10 points]

In this experiment we have aimed to implement combinational logic circuits and Full adder using the Verilog. In this experiment we did not use ‘’ (Bitwise AND), ‘—’ (Bitwise OR), ‘’ (Bitwise NOT), and ‘’ (Concatenate) ‘ˆ ’, ‘+’, ‘-’, ‘*’, ‘/’ and bitwise left and right shift operations. This experiment is an introduction to learn how to design and use a module. We simulated all modules in our design.

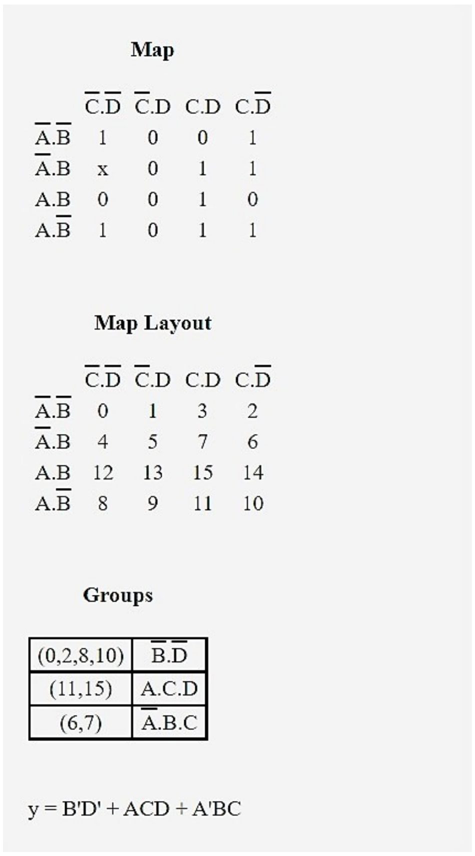
2 PRELIMINARY

We have answered all questions and provide everything that are indicated in the “Report” section of the related experiment in “Experiments Booklet”.

2.1 Question 1

We have found all prime implicants by using Karnaugh Map.

2.1.1 1.a



Map

	$\overline{C}.\overline{D}$	$\overline{C}.D$	$C.D$	$C.\overline{D}$
$\overline{A}.\overline{B}$	1	0	0	1
$\overline{A}.B$	x	0	1	1
$A.B$	0	0	1	0
$A.\overline{B}$	1	0	1	1

Map Layout

	$\overline{C}.\overline{D}$	$\overline{C}.D$	$C.D$	$C.\overline{D}$
$\overline{A}.\overline{B}$	0	1	3	2
$\overline{A}.B$	4	5	7	6
$A.B$	12	13	15	14
$A.\overline{B}$	8	9	11	10

Groups

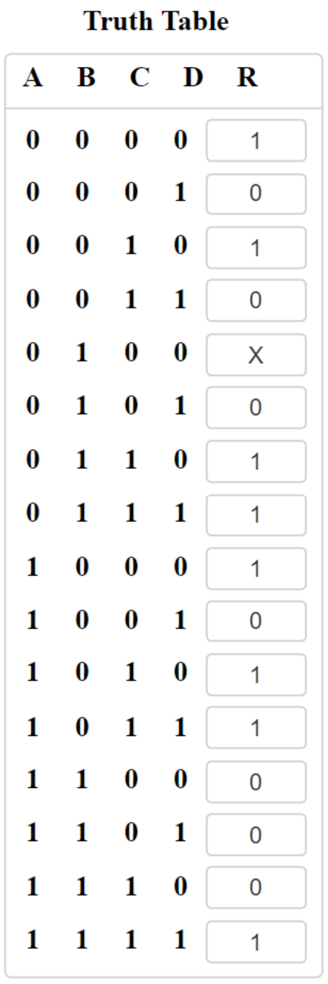
(0,2,8,10)	$\overline{B}.\overline{D}$
(11,15)	$A.C.D$
(6,7)	$\overline{A}.B.C$

y = B'D' + ACD + A'BC

Figure 1: Karnaugh Map

2.1.2 1.b

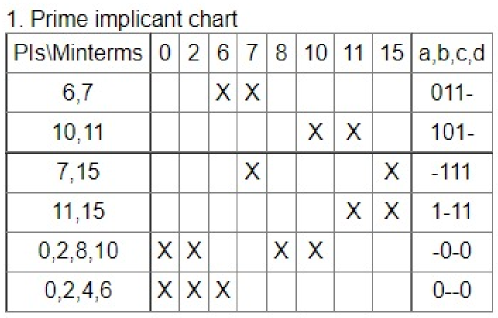
We have found all prime implicants by using Quine-Mccluskey Method.



Truth Table

A	B	C	D	R
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	X
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Figure 2: Truth table



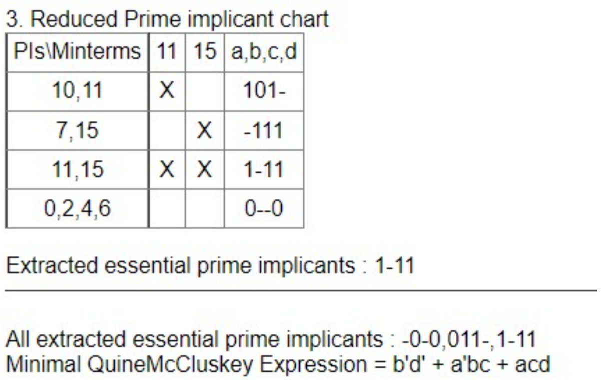
1. Prime implicant chart

PIs\Minterms	0	2	6	7	8	10	11	15	a,b,c,d
6,7			X	X					011-
10,11						X	X		101-
7,15				X				X	-111
11,15							X	X	1-11
0,2,8,10	X	X			X	X			-0-0
0,2,4,6	X	X	X						0--0

Figure 3: Prime implicant chart

2.1.3 1.c

We have created prime implicant chart and found the expression with the minimum cost with 2 units of cost for each variable and 1 unit of cost for complement of a variable.



3. Reduced Prime implicant chart

PIs\Minterms	11	15	a,b,c,d
10,11	X		101-
7,15		X	-111
11,15	X	X	1-11
0,2,4,6			0--0

Extracted essential prime implicants : 1-11

All extracted essential prime implicants : -0-0,011-,1-11
Minimal QuineMcCluskey Expression = b'd' + a'bc + acd

Figure 4: Essential Prime Implicants

2.1.4 1.d

We have designed and drew the lowest cost expression using NOT, AND, and OR gates.

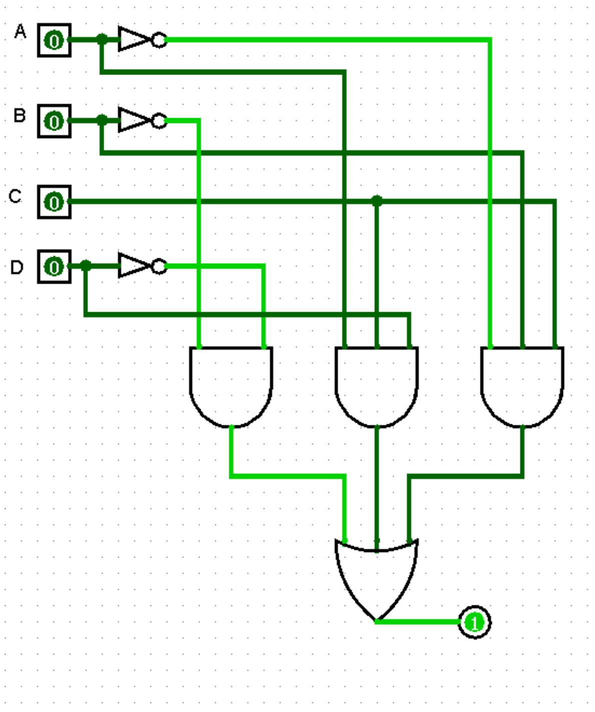


Figure 5: The truth table of F1(a, b) = a + a · b

2.1.5 1.e

We have designed and drew the lowest cost expression using only NAND gates.

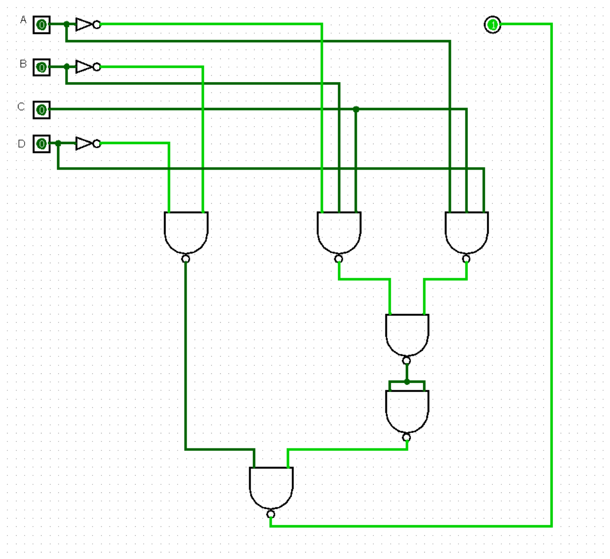


Figure 6: The truth table of F1(a, b) = a + a · b

2.1.6 1.f

We have designed and drew the lowest cost expression using a single 8:1 Multiplexer, AND, OR and NOT gates.

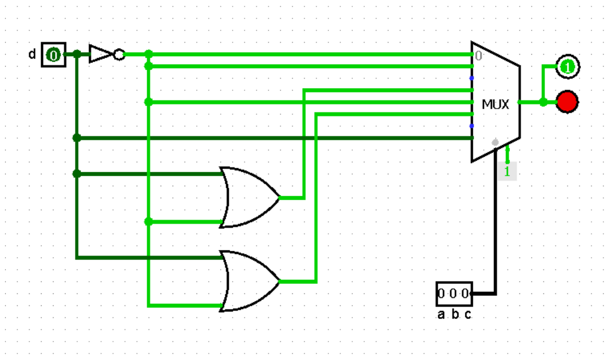


Figure 7: The truth table of F1(a, b) = a + a · b

2.2 Question 2

We have designed and drew the functions F2 and F3 given in below Equations Equation 2 and Equation 3 using ONE single 3:8 decoder, 2-input OR gates. F2(a, b, c) = a'bc + ab'c and F3(a, b, c) = abc' + ab

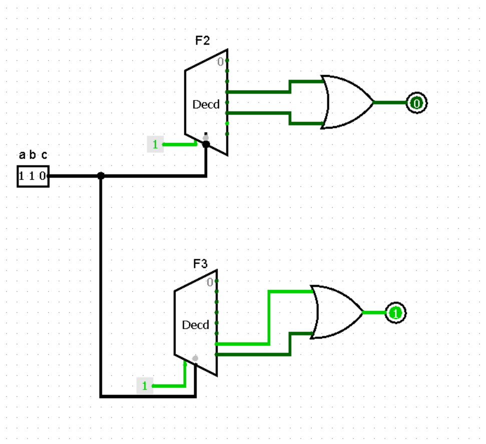


Figure 8: Schema of F2 and F3

2.3 Question 3

We recalled signed and unsigned addition for binary numbers in 2's complement notation. We recalled signed and unsigned subtraction for binary numbers in 2's complement notation. We recalled signed and unsigned subtraction for binary numbers in 2's complement notation.

3 EXPERIMENT

3.1 PART 1

In this part, we have implemented AND, OR, NOT, XOR, NAND, 8:1 Multiplexer and 3:8 Decoder modules. After implemented those modules we run the simulations for various input combinations to validate our designs.

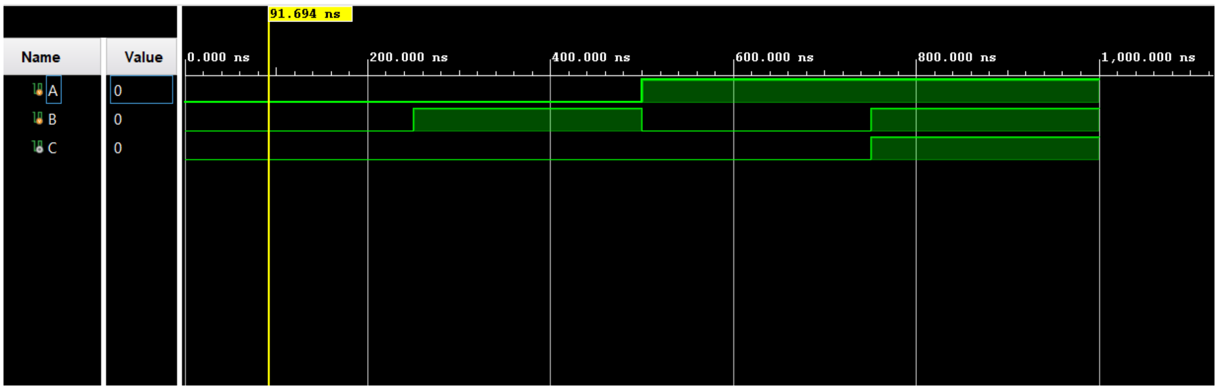


Figure 9: Simulation for AND gate

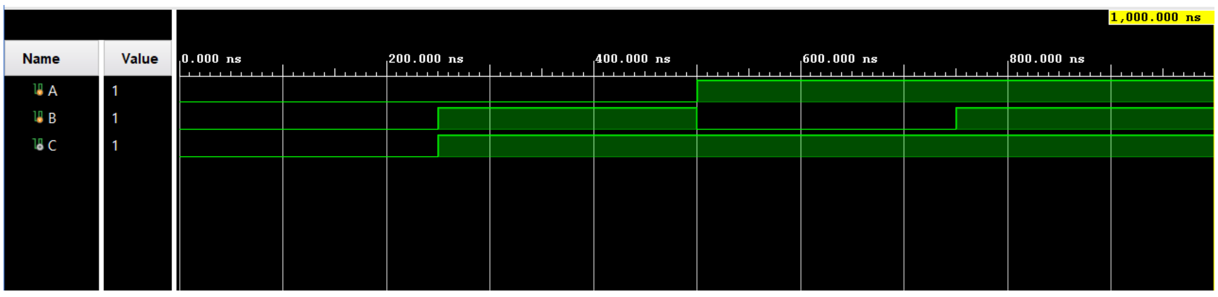


Figure 10: Simulation for OR gate

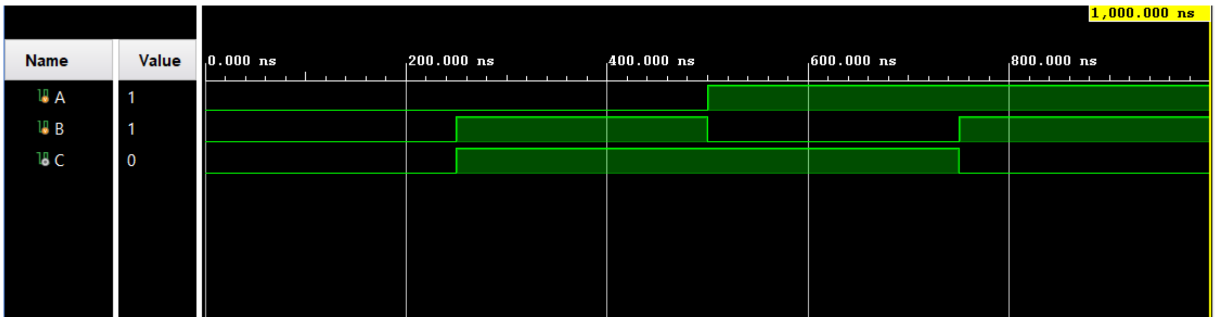


Figure 11: Simulation for XOR gate

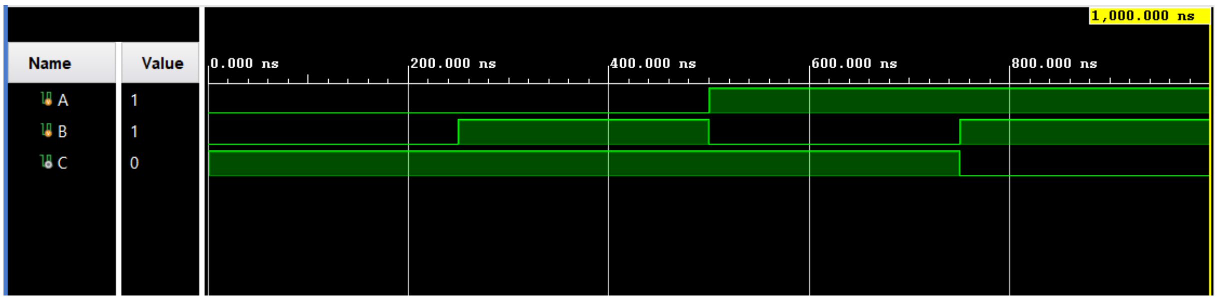


Figure 12: Simulation for NAND gate

3.2 PART 2

We have implemented the circuit that we have designed in Preliminary 1.d. section using NOT, AND and OR modules in Verilog.

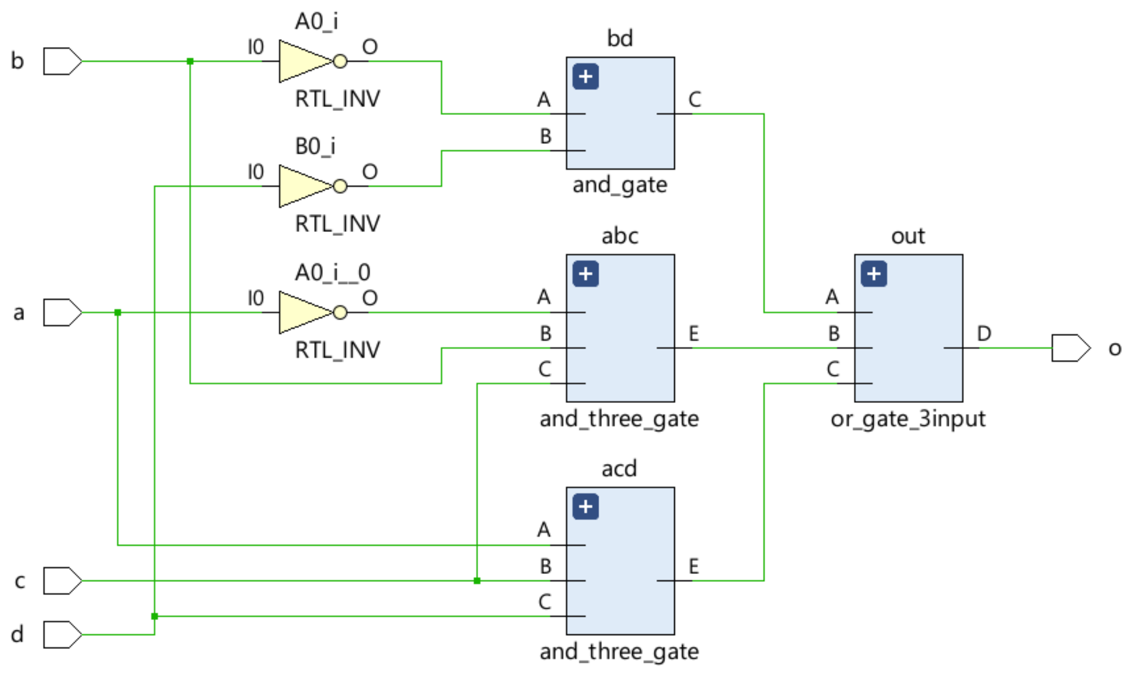


Figure 13: Schema of circuit designed for Part 2

3.3 PART 3

We have implemented the circuit that we have designed in Preliminary 1.e. section using only NAND modules in Verilog.

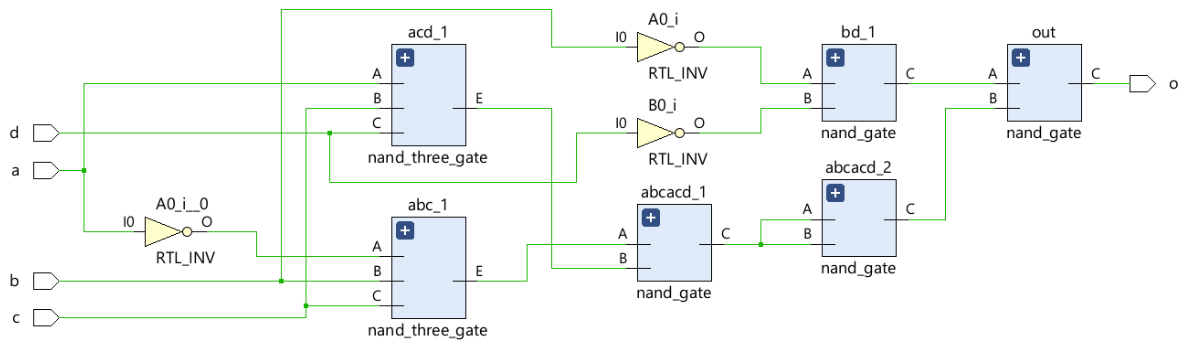


Figure 14: Schema of circuit designed for Part 3

3.4 PART 4

We have implemented the circuit that we have designed in Preliminary 1.f. section using 8:1 multiplexer, NOT, AND and OR modules in Verilog.

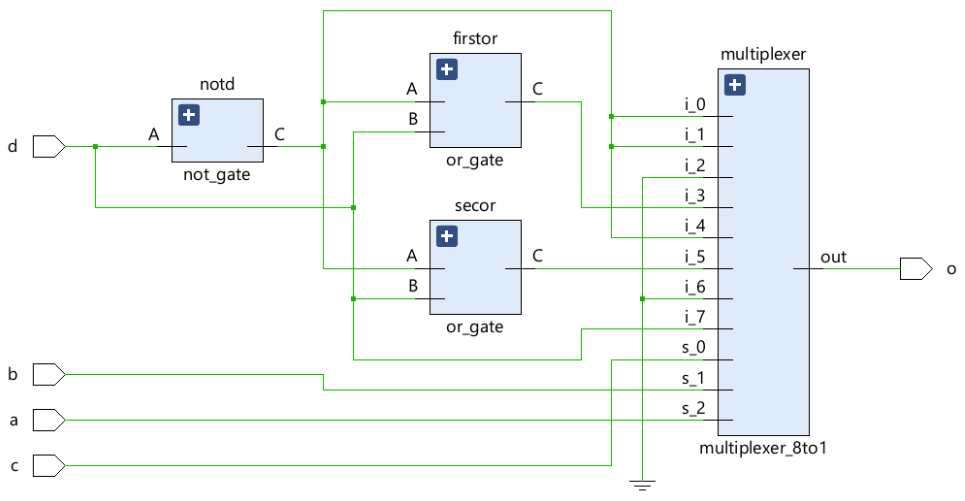


Figure 15: Schema of circuit designed for Part 4

3.5 PART 5

We have implemented the circuit that we have designed in Preliminary 2 section using 3:8 decoder and OR modules in Verilog.

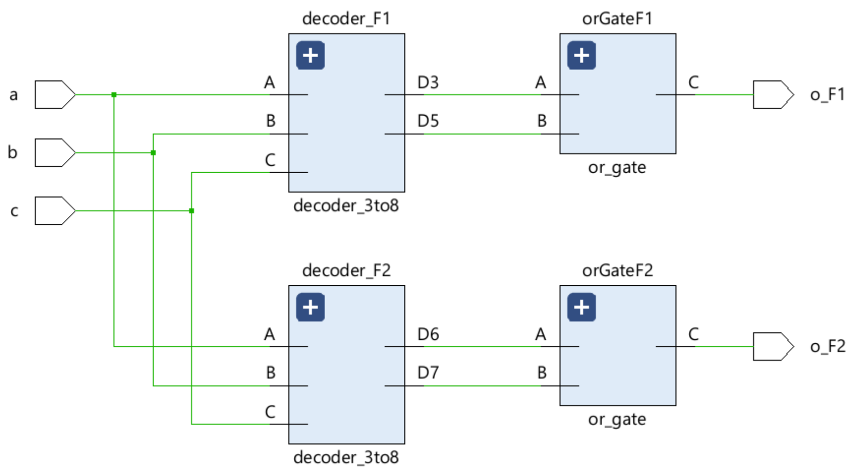


Figure 16: Schema of circuit designed for Part 5

3.6 PART 6

In this part, we have implemented 1-Bit Half Adder module by using AND, OR, NOT, XOR modules which we have designed in the first part. Then, we hav simulated it for each different combination of input.

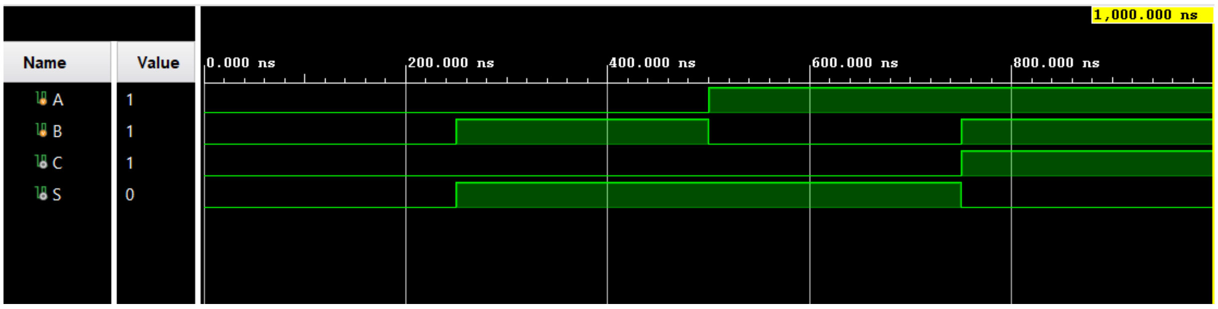


Figure 17: Simulation of designed Half-Adder

3.7 PART 7

In this part, we have implemented 1-Bit Full Adder by using half adder and OR modules which we have designed in the previous parts. Then, we have simulated it for each different combination of input.

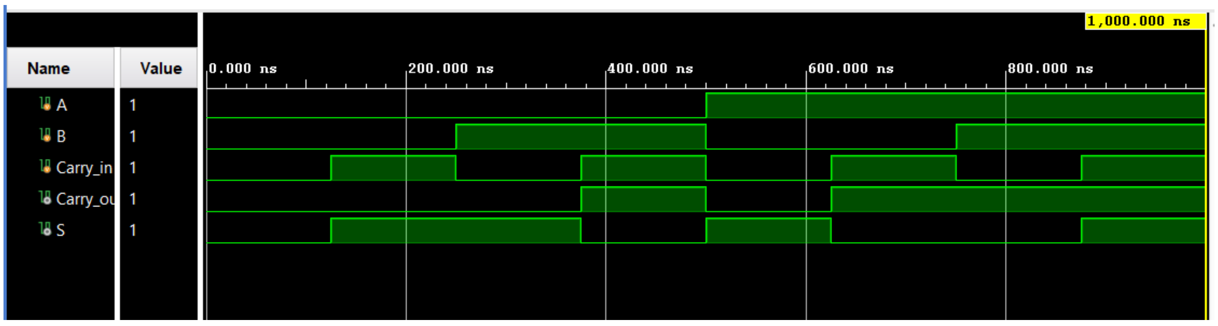


Figure 18: Simulation of designed Full-Adder

3.8 PART 8

In this part, we have implemented a 4-Bit Full Adder by using 1-Bit Full Adder modules which we have designed in the previous part. Then, we have simulated it for '8+1', '2+7', '4+5', '11+10', '14+5', '15+9', '6+3', and '8+12' operations.

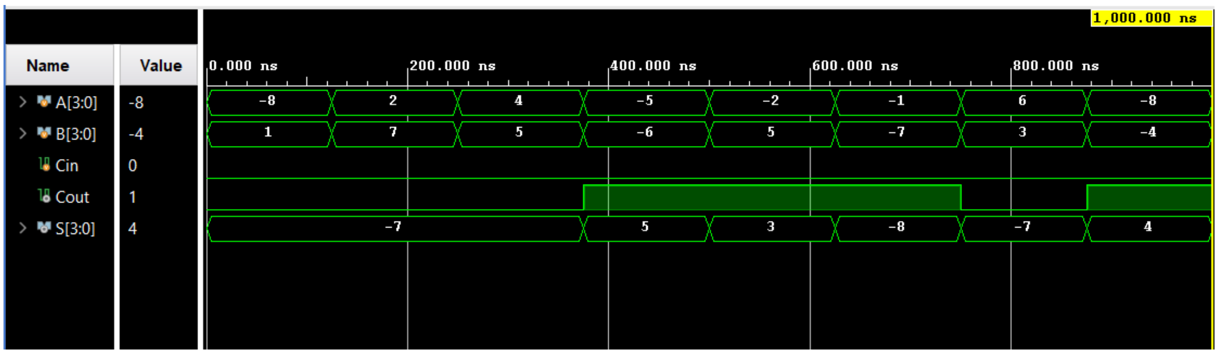


Figure 19: Simulation of designed 4-Bit Full-Adder

3.9 PART 9

In this part, we have implemented a 8-Bit Full Adder by using 1-Bit Full Adder modules which we have designed in the previous part. Then, we have simulated it for '29+5', '51+92', '17+28', '191+2', '200+95', '49+25', '78+255', and '43+59' operations.

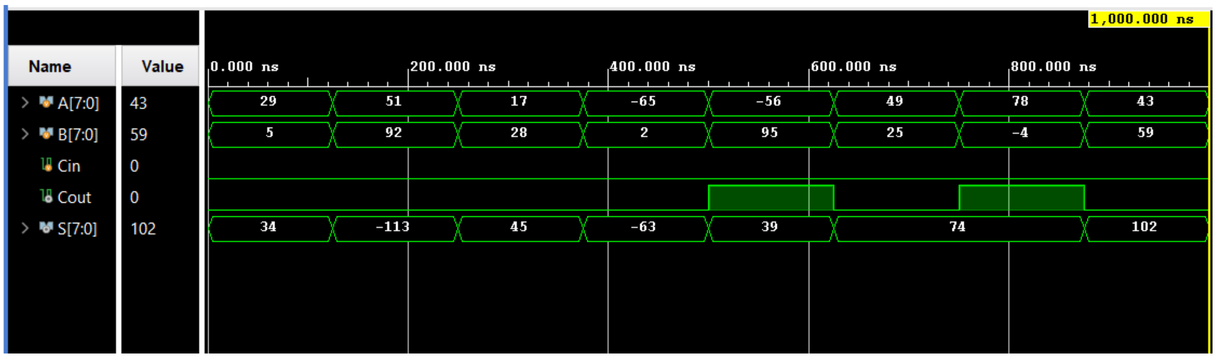


Figure 20: Simulation of designed 8-Bit Full-Adder

3.10 PART 10

In this part, we have implemented a 16-Bit Adder-Subtractor by using 8-Bit Full Adder and XOR modules which we have designed in the previous part. Then, we have simulated it for '23+3', '21+75', '16800+16900', '69834+66500', '325+97', '44+190', '463+241', and '86+572' operations

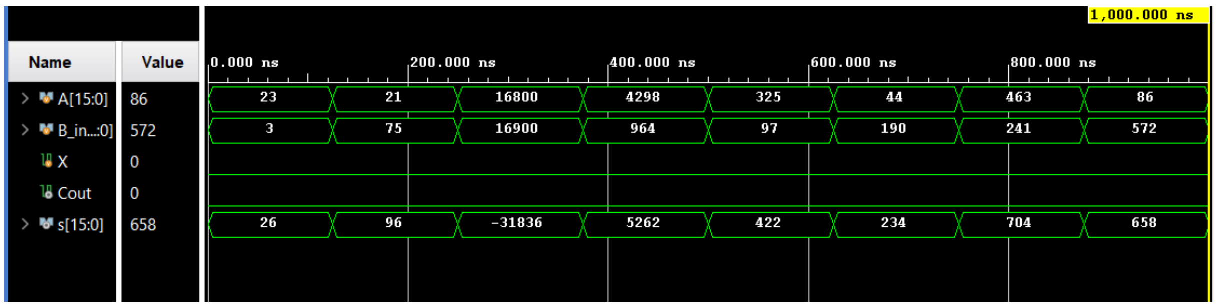


Figure 21: Simulation of designed 16-Bit Adder-Subtractor

3.11 PART 11

In this part, we have implemented a module which calculates the B - 2A by using 16-Bit Adder-Subtractor, Adder, NOT, XOR, AND, and OR modules. Then, we have simulated it for 'A=32, B=7', 'A=21, B= 85', 'A=16, B=36', 'A=256, B=5', 'A=200,B=95', 'A=45, B=135', 'A=36, B=255', and 'A=25, B=65' inputs.

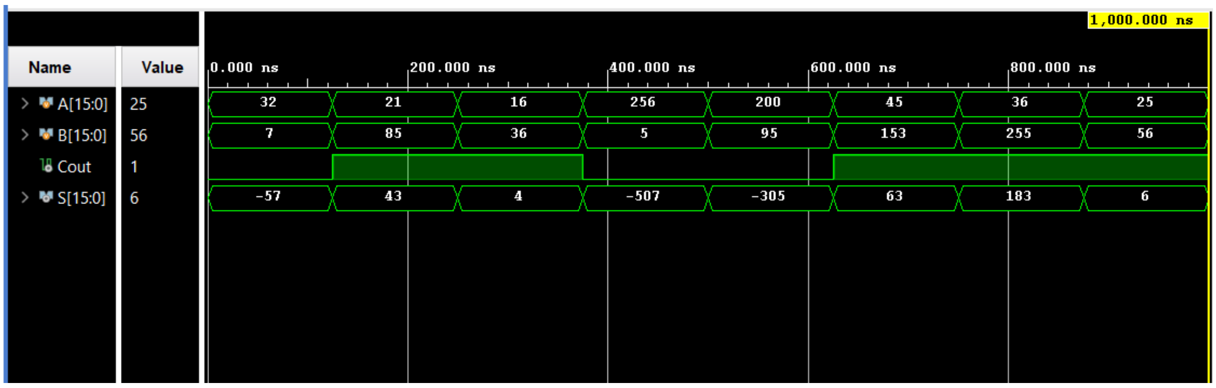


Figure 22: Simulation of designed B-2A Module

4 DISCUSSION [25 points]

While implementing the 16-Bit Adder-Subtractor we have faced with problems though the logic we formed seems efficient. We overcame with that problem by implementing 16-Bit Full Adder first. Then we have created 16-Bit Adder-Subtractor with 16-Bit Full Adder. And we have used them in Part 11.

5 CONCLUSION [10 points]

The main difficulty that we have faced was with the editor and syntax. But once we overcame with it, it was easy to build whole project module by module. While implementing more complex modules we have learned how to simulate and debug. Also recalled how to create Half-Adder,Full-Adders(1-4-8-16) and Adder-Subtractor.