ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E

DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

HOMEWORK NO	:	1
HOMEWORK DUE	:	10.04.2022
LAB SESSION	:	FRIDAY - 14.00
GROUP NO	:	G25

GROUP MEMBERS:

150210710 : FATİH BASKIN

150190102 : ELVAN TEKE

SPRING 2022

Contents

1 INTRODUCTION

In this homework, we simplified given expressions in Preliminaries using Karnaugh diagrams and Quine-McCluskey method. then we created abstract circuity by drawing and started to implement those wanted circuits in Verilog.

As stated in homework introduction, we had to design every gate as modules in Verilog. In Part 1, we designed AND, OR, NOT, XOR, NAND, 8:1 Multiplexer and 3:8 Decoder modules. In Parts 2, 3, 4 and 5 we designed circuits in Verilog which were given in Preliminaries.

In Part 6, we had designed Half Adder and in Part 7, Full Adder modules. With using the Full Adder in Part 7, we designed 4 Bit, 8 Bit Adders and 16 Bit Adder/Subtractor circuits. In Part 11, by using Adder/Subtractor in Part 10, we created a circuit which performs B - 2A operation.

2 MATERIALS AND METHODS

2.1 Preliminaries

2.1.1 Function F_1

$F_1(a, b, c, d) = \cup_1(0, 2, 6, 7, 8, 10, 11, 15) + \cup_\phi(4)$

2.1.1.a) Finding prime implicants using Karnaugh diagram.

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

Prime implicants are: $\bar{b}\bar{d}$, $\bar{a}\bar{d}$, $\bar{a}bc$, bcd, acd, $a\bar{b}c$

2.1.1.b) Finding prime implicants using Quine-McCluskey method.

0	0 0 0 0	✓	0,2	0 0 - 0	✓	0,2,4,6	0 - - 0	
2	0 0 1 0	✓	0,4	0 - 0 0	✓	0,2,8,10	- 0 - 0	
4	0 1 0 0	✓	0,8	- 0 0 0	✓			
8	1 0 0 0	✓	2,6	0 - 1 0	✓			
6	0 1 1 0	✓	2,10	- 0 1 0	✓			
10	1 0 1 0	✓	4,6	0 1 - 0	✓			
7	0 1 1 1	✓	8,10	1 0 - 0	✓			
11	1 0 1 1	✓	6,7	0 1 1 -				
15	1 1 1 1	✓	10,11	1 0 1 -				
			7,15	1 0 1 -				
			11,15	1 0 1 -				

Prime implicants are the cases of 0 1 1 - , 1 0 1 - , - 1 1 1 , 1 - 1 1 , 0 - - 0 and - 0 - 0 which translates into $\bar{a}bc$, $a\bar{b}c$, bcd, acd, $\bar{a}\bar{d}$, $\bar{b}\bar{d}$ respectively.

2.1.1.c) Prime implicant chart and minimum cost with 2 units of cost for each variable and 1 unit of cost for complement of a variable.

	0	2	6	7	8	10	11	15	**Cost**
a′d′	x	x	x						6
b′d′	x	x			x	x			6
a′bc			x	x					7
ab′c						x	x		7
acd							x	x	6
bcd				x				x	6

Figure 1: Minterm 8 is a distinguished point.

	6	7	11	15	**Cost**
a′d′	x				6
a′bc	x	x			7
ab′c			x		7
acd			x	x	6
bcd		x		x	6

Figure 2: There are no distinguished points, most sensible implicants are $\bar{a}bc$ and acd.

Considering the cost, cheapest implementation is by using $\bar{b}\bar{d}$, $\bar{a}bc$ and acd with cost of 19.

2.1.1.d) Designing the lowest cost expression using NOT, AND, and OR gates.

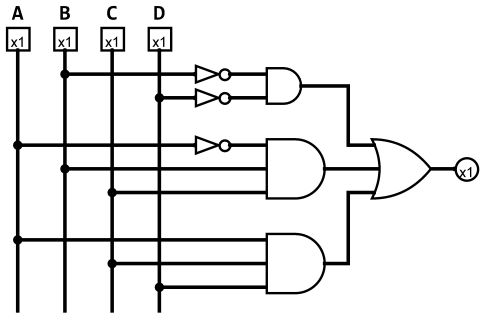

Figure 3: Lowest cost implementation F_1 using NOT, AND, OR gates.

2.1.1.e) Designing the lowest cost expression using NAND gates.

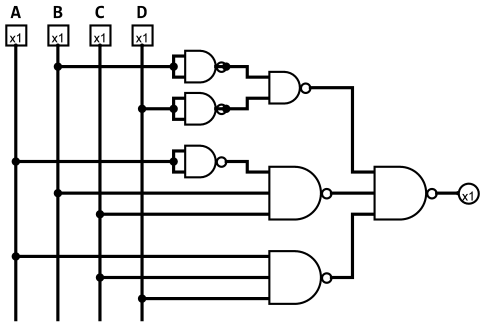

Figure 4: Lowest cost implementation F_1 using NAND gates.

2.1.1.f) Designing the lowest cost expression using single 8:1 Multiplexer, AND, OR, NOT gates.

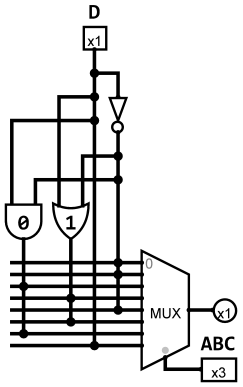

Figure 5: Lowest cost implementation F_1 using 8:1 Multiplexer, AND, OR, NOT gates.

2.1.2 F_2 and F_3

$F_2(a, b, c) = a\bar{b}c + \bar{a}bc$ 101 and 011 outputs 5 and 3
$F_3(a, b, c) = ab\bar{c} + ab$ 111 and 11- outputs 6 and 7

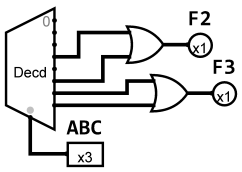

Figure 6: Implementation of F_2 and F_3 using 3:8 Decoder and two input OR gates.

2.1.3 Preliminary 3, Binary Addition and Subtraction

We were expected to study on binary addition and subtraction for this part for both signed and unsigned numbers. We did study binary addition and subtraction to be able to do parts 8, 9, 10 and 11.

2.2 Part 1

In this part, we were expected to implement AND, OR, NOT, XOR, NAND, 8:1 Multiplexer and 3:8 Decoder modules which we would use in the following parts.

2.2.1 NOT Gate

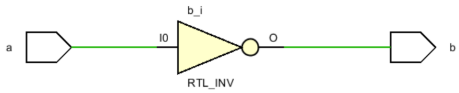

Figure 7: Screenshot of the RTL design of NOT gate.

2.2.2 Two Input AND Gate

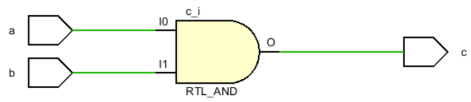

Figure 8: Screenshot of the RTL design of two input AND gate.

2.2.3 Three Input AND Gate

We were not required to make three input AND gate but for sake of simplicity we had designed three input AND gate using the two input AND gate module.

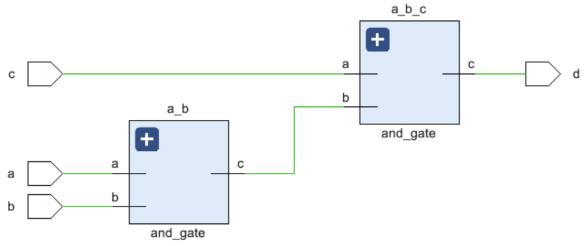

Figure 9: Screenshot of the RTL design of three input AND gate.

2.2.4 Two Input OR Gate

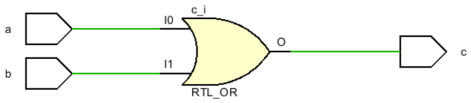

Figure 10: Screenshot of the RTL design of two input OR gate.

2.2.5 Three Input OR Gate

We were not required to make three input OR gate but for sake of simplicity we had designed three input OR gate using the two input OR gate module.

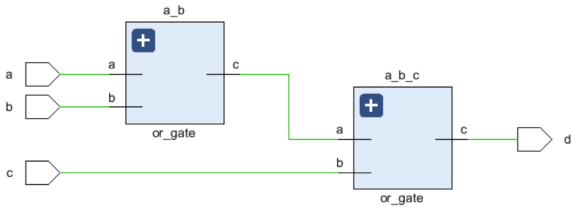

Figure 11: Screenshot of the RTL design of three input OR gate.

2.2.6 Two Input XOR Gate

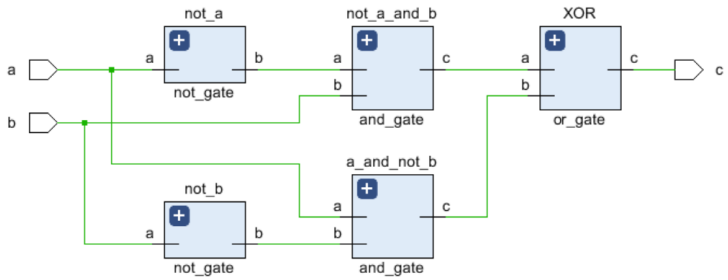

Figure 12: Screenshot of the RTL design of two input XOR gate.

2.2.7 Two Input NAND Gate

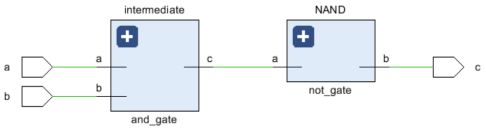

Figure 13: Screenshot of the RTL design of two input NAND gate.

2.2.8 Three Input NAND Gate

We were not required to make three input NAND gate but for sake of simplicity we had designed three input NAND gate using the two input NAND gate module.

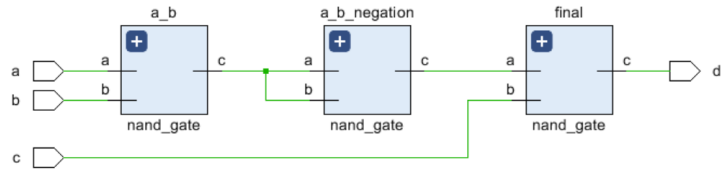

Figure 14: Screenshot of the RTL design of three input NAND gate.

2.2.9 2:1 Multiplexer

We were not required to make 2:1 Multiplexer but for sake of simplicity we had designed it using AND, OR, NOT gates.

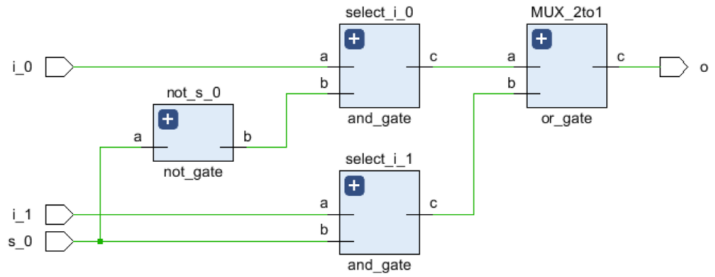

Figure 15: Screenshot of the RTL design of 2:1 Multiplexer.

2.2.10 4:1 Multiplexer

We were not required to make 4:1 Multiplexer but for sake of simplicity we had designed it using 2:1 Multiplexer we had designed earlier.

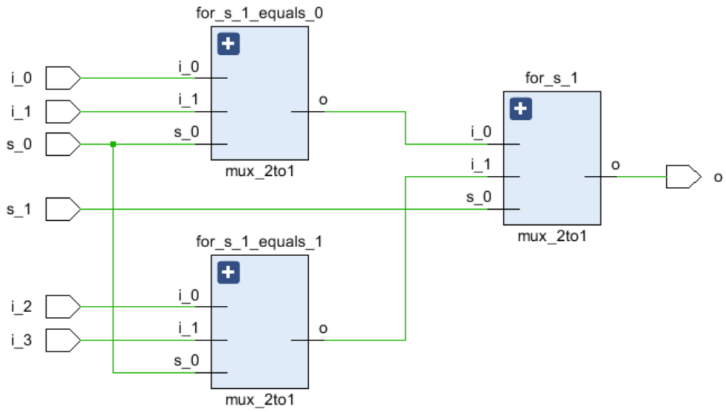

Figure 16: Screenshot of the RTL design of 4:1 Multiplexer.

2.2.11 8:1 Multiplexer

To simply implement this circuit, we had designed 4:1 and 2:1 Multiplexers and used them to create 8:1 Multiplexer.

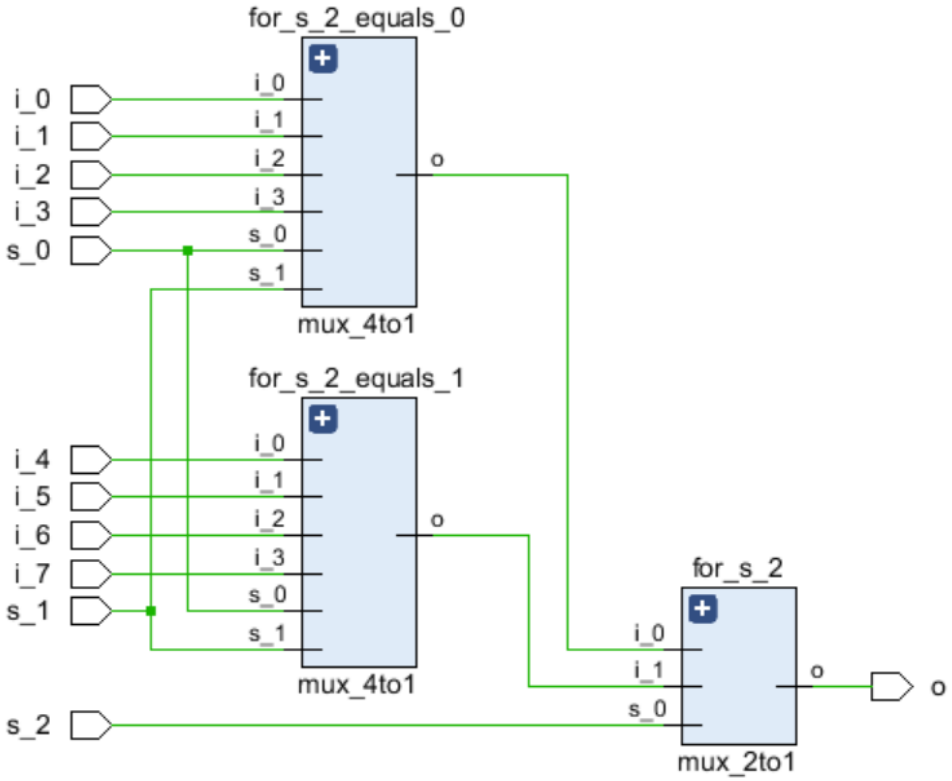

Figure 17: Screenshot of the RTL design of 8:1 Multiplexer.

2.2.12 3:8 Decoder

We gave implemented decoder by ANDing i0, i1, i2 and their complements. We used previously designed three input AND gates for here.

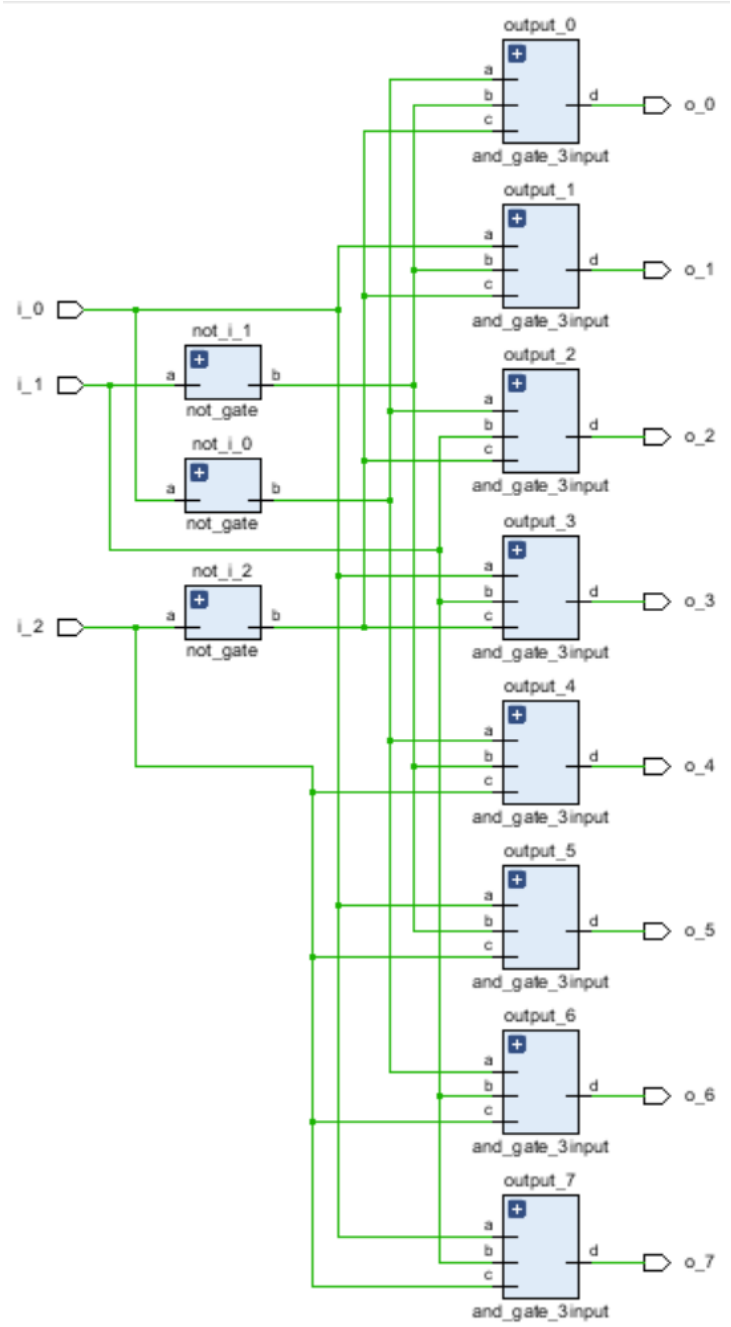

Figure 18: Screenshot of the RTL design of 3:8 Decoder.

2.3 Part 2

We were expected to implement the circuit that we had designed in Preliminary 1.d. section using NOT, AND, and OR modules in Verilog.

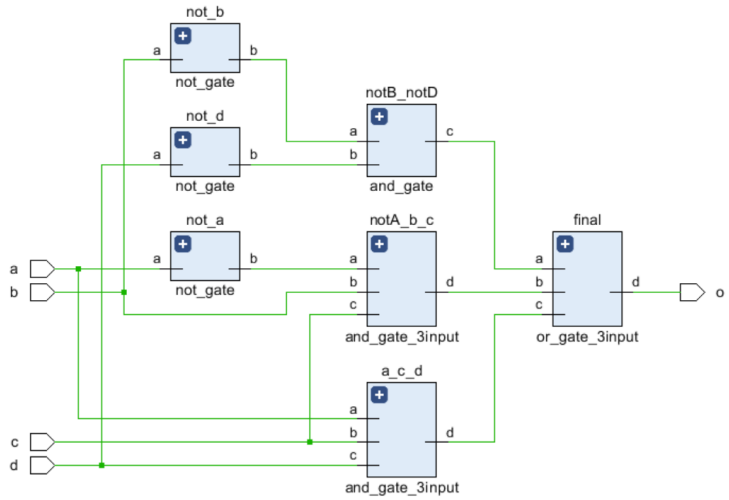

Figure 19: Screenshot of the RTL design of Part 2.

2.4 Part 3

We were expected to implement the circuit that we had designed in Preliminary 1.e. section using NAND modules in Verilog.

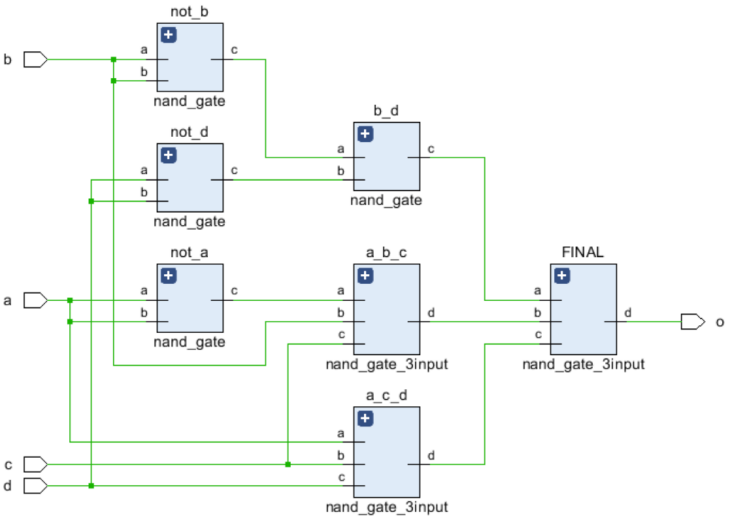

Figure 20: Screenshot of the RTL design of Part 3.

2.5 Part 4

We were expected to implement the circuit that we had designed in Preliminary 1.f. section using only 8:1 Multiplexer, AND, NOT and OR gates.

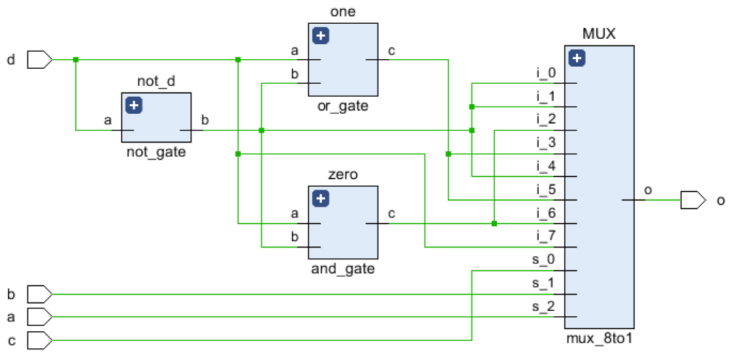

Figure 21: Screenshot of the RTL design of Part 4.

2.6 Part 5

We were expected to implement the circuit that we had designed in Preliminary 2 section using only 3:8 Decoder and two two-input-OR-gates.

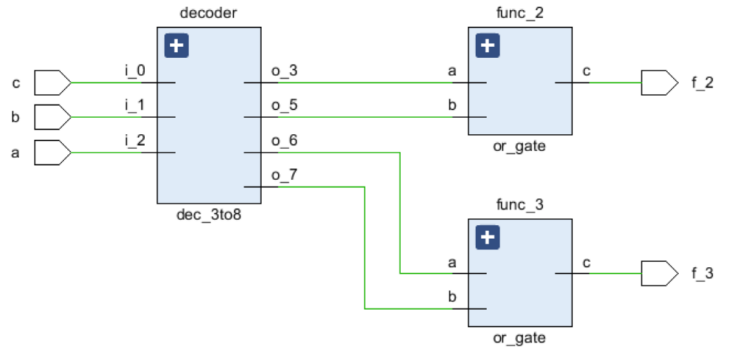

Figure 22: Screenshot of the RTL design of Part 5.

2.7 Part 6

We were expected to implement 1-Bit Half Adder module by using AND, OR, NOT, XOR modules.

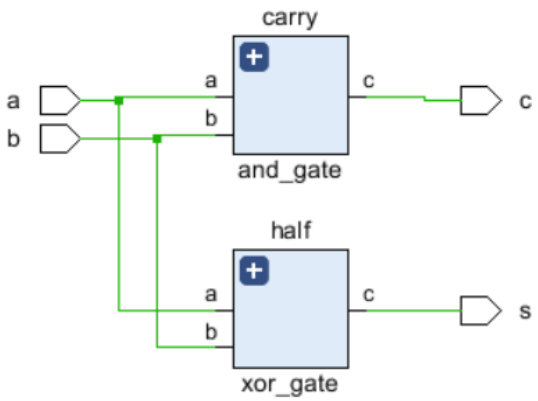

Figure 23: Screenshot of the RTL design of Part 6.

2.8 Part 7

We were expected to implement 1-Bit Full Adder by using half adder and OR modules.

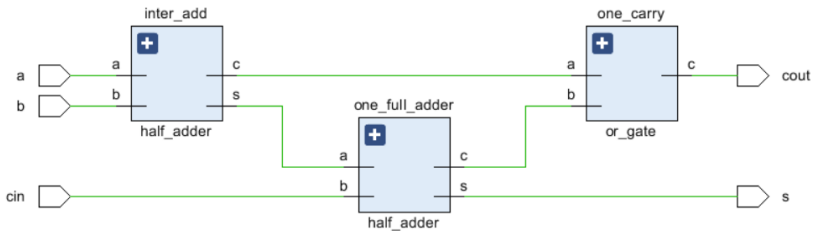

Figure 24: Screenshot of the RTL design of Part 7.

2.9 Part 8

We were expected to implement 4-Bit Full Adder by using 1-Bit Full Adder modules.

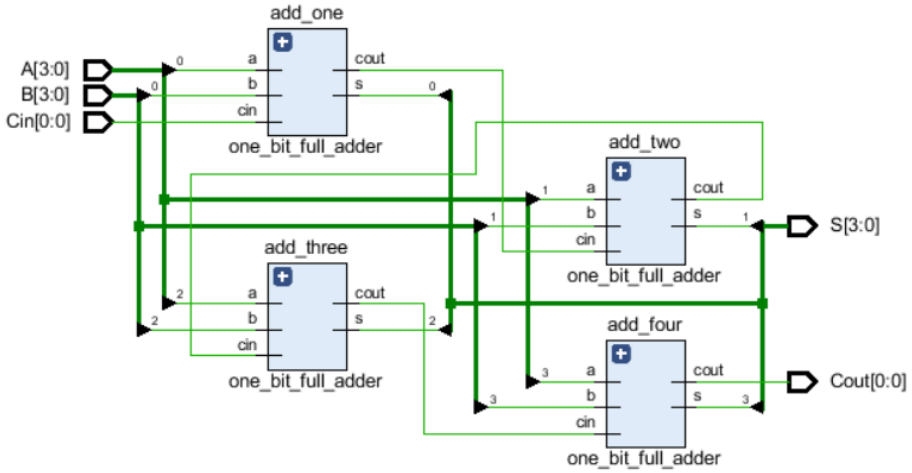

Figure 25: Screenshot of the RTL design of Part 8.

2.10 Part 9

We were expected to implement 8-Bit Full Adder by using 1-Bit Full Adder modules.

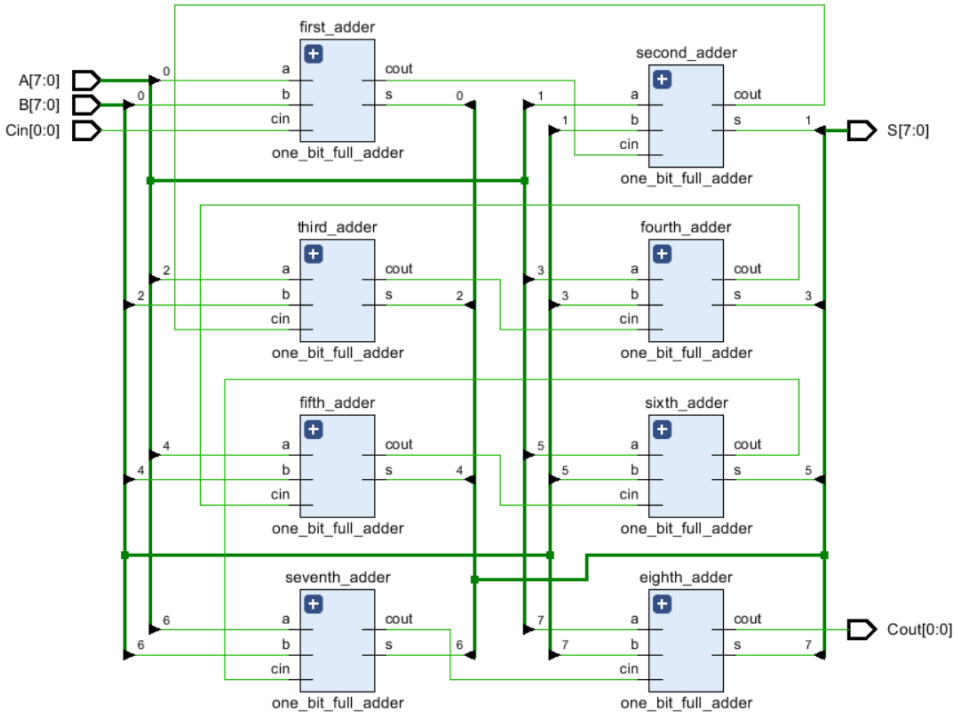

Figure 26: Screenshot of the RTL design of Part 9.

2.11 Part 10

We were expected to implement 16-Bit Adder-Subtractor by using 8-Bit Full Adder and XOR modules.

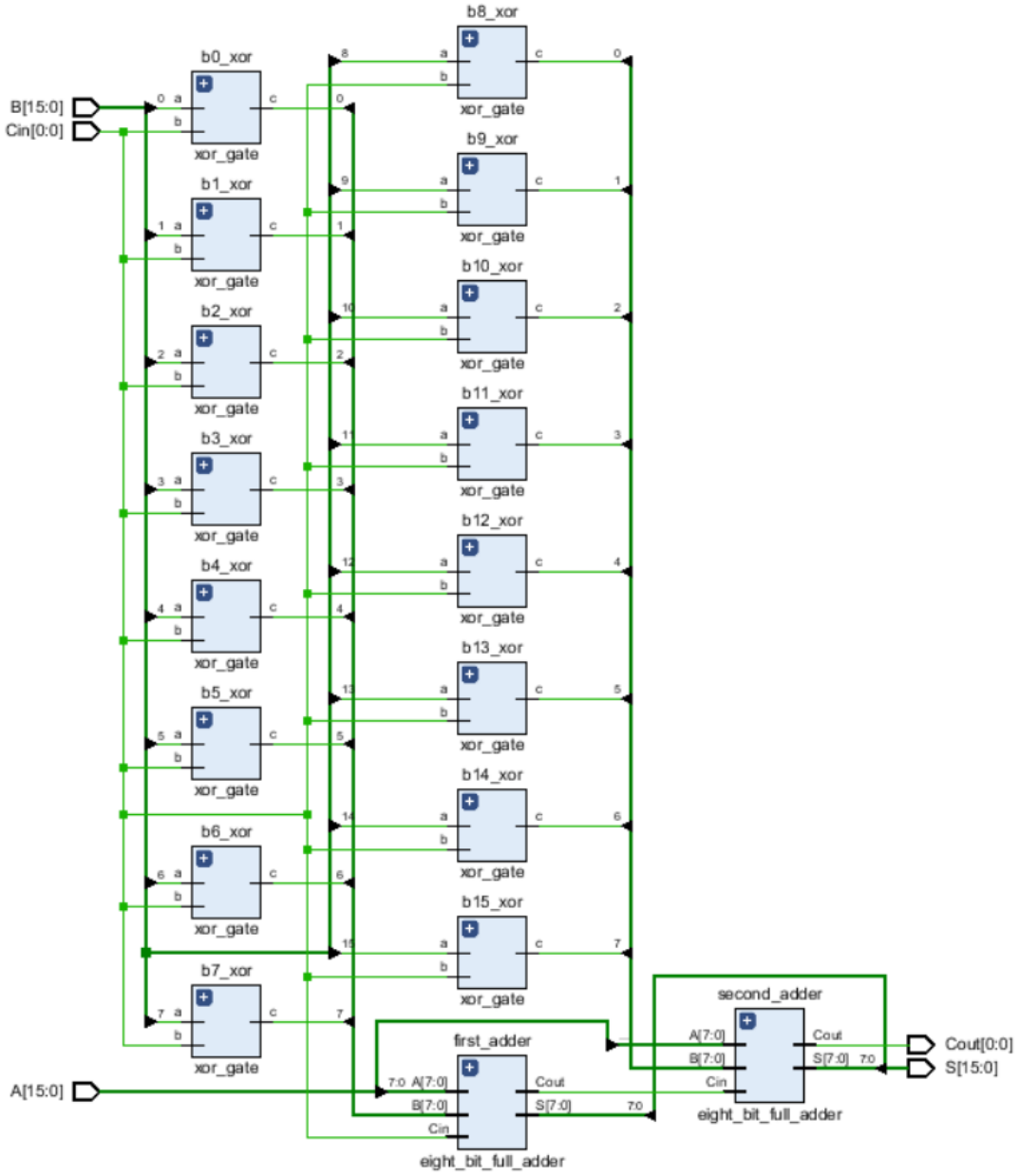

Figure 27: Screenshot of the RTL design of Part 10.

2.12 Part 11

In this part, we were expected to design a module which calculates the B - 2A by using 16-Bit Adder-Subtractor, Adder, NOT, XOR, AND, and OR modules. We achieved 2A by shifting A 1 bit to left.

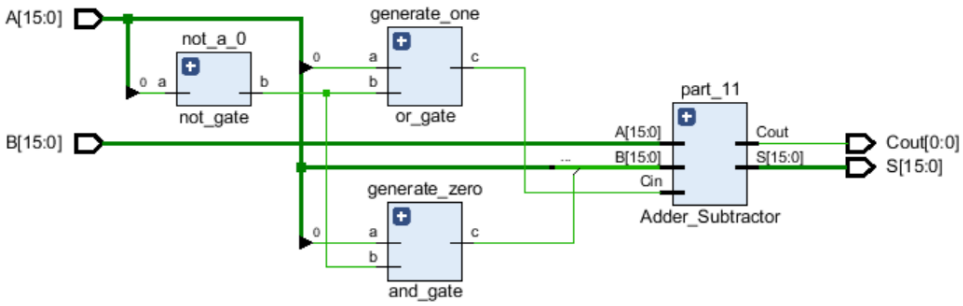

Figure 28: Screenshot of the RTL design of Part 11.

3 RESULTS

3.1 Part 1

3.1.1 NOT Gate

Simulation for the NOT gate. A is input and B is output

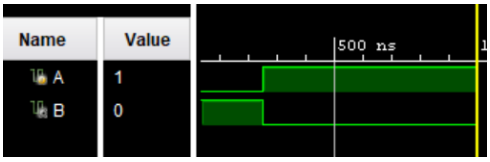

Figure 29: Screenshot of the simulation of NOT gate we designed. Circuit behaves as expected.

3.1.2 Two Input AND Gate

Simulation for the two input AND gate. A and B are inputs and C is output.

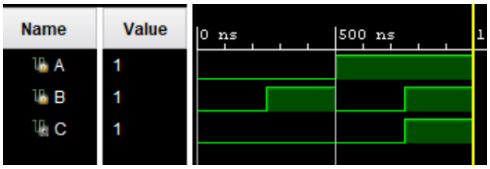

Figure 30: Screenshot of the simulation of two input AND gate we designed. Circuit behaves as expected.

3.1.3 Three Input AND Gate

Simulation for the three input AND gate. A, B and C are inputs and D is output.

Figure 31: Screenshot of the simulation of three input AND gate we designed. Circuit behaves as expected.

3.1.4 Two Input OR Gate

Simulation for the two input OR gate: A and B is input and C is output

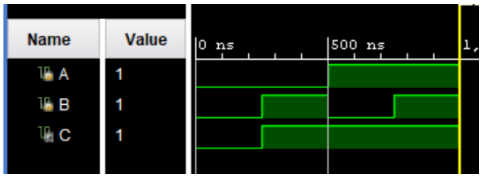

Figure 32: Screenshot of the simulation of two input OR gate we designed. Circuit behaves as expected.

3.1.5 Three Input OR Gate

Simulation for the three input OR gate. A, B and C are inputs and D is output.

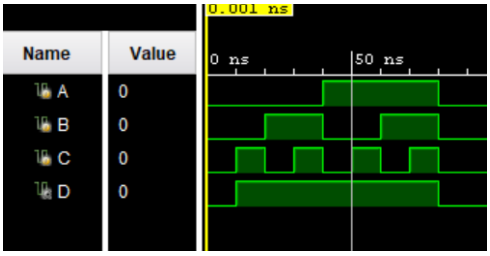

Figure 33: Screenshot of the simulation of three input OR gate we designed. Circuit behaves as expected.

3.1.6 Two Input XOR Gate

Simulation for the two input XOR gate. A and B are inputs and C is output.

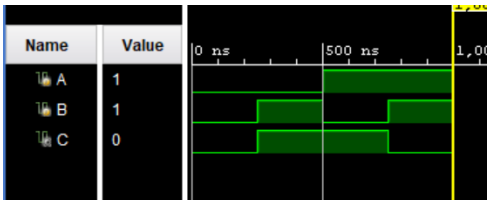

Figure 34: Screenshot of the simulation of three input XOR gate we designed. Circuit behaves as expected.

3.1.7 Two Input NAND Gate

Simulation for the two input NAND gate: A and B is input and C is output

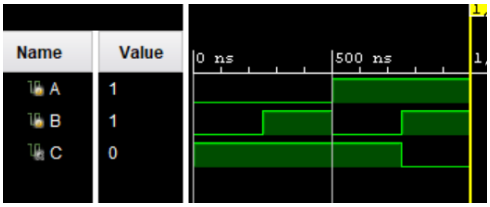

Figure 35: Screenshot of the simulation of two input NAND gate we designed. Circuit behaves as expected.

3.1.8 Three Input NAND Gate

Simulation for the three input NAND gate. A, B and C are inputs and D is output.

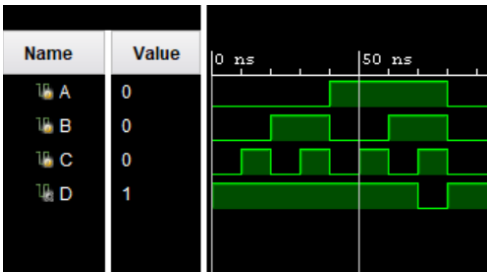

Figure 36: Screenshot of the simulation of three input NAND gate we designed. Circuit behaves as expected.

3.1.9 2:1 Multiplexer

Simulation for the 2:1 Multiplexer. I0 and I1 are inputs, S0 is selector and O is output.

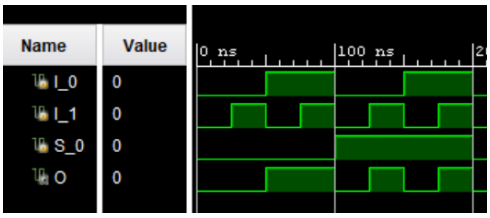

Figure 37: Screenshot of the simulation of 2:1 Multiplexer we designed. Circuit behaves as expected.

3.1.10 4:1 Multiplexer

Simulation for the 4:1 Multiplexer. I0, I1, I2 and I3 are inputs, S0 and S1 are selectors and O is output.

Figure 38: Screenshot of the simulation of 4:1 Multiplexer we designed. Circuit behaves as expected.

3.1.11 8:1 Multiplexer

Simulation for the 8:1 Multiplexer. I0, I1, I2, I3, I4, I5, I6 and I7 are inputs, S0, S1 and S2 are selectors and O is output.

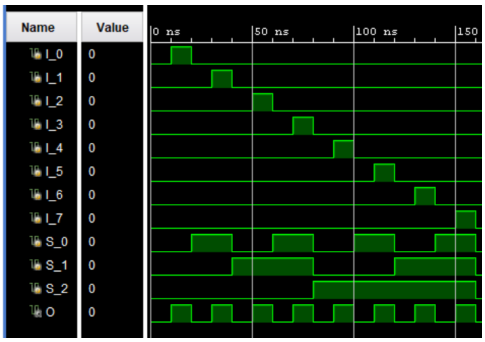

Figure 39: Screenshot of the simulation of 8:1 Multiplexer we designed. Circuit behaves as expected.

3.1.12 3:8 Decoder

Simulation for the 3:8 Decoder. i0, i1 and i2 are inputs and o1, o2, o3, o4, o5, o6, o7 are outputs.

Figure 40: Screenshot of the simulation of 3:8 Decoder we designed. Circuit behaves as expected.

3.2 Part 2

$F_1(a, b, c, d) = \cup_1(0, 2, 6, 7, 8, 10, 11, 15) + \cup_\phi(4)$

Simulation for the circuit we had designed for Part 2. According to function's expression, we should get output as 1 in cases of 0000, 0020, 0110, 0111, 1000, 1010, 1011, 1111. Don't care condition for 0100. Inputs are A, B, C and D and output is O.

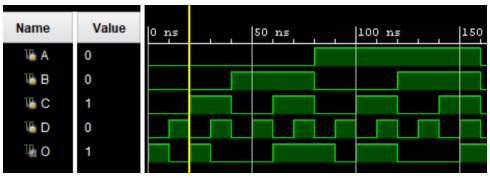

Figure 41: Screenshot of the simulation of Part 2 circuit we designed. Circuit behaves as expected. Don't care condition is O in case of 0100

3.3 Part 3

Simulation for the circuit we had designed for Part 3. We should get same result as Part 2. Inputs are A, B, C and D and output is O.

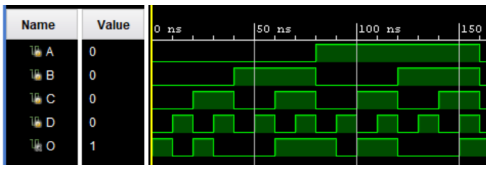

Figure 42: Screenshot of the simulation of Part 3 circuit we designed. Behaves as similar to Part 2

3.4 Part 4

Simulation for the circuit we had designed for Part 4. We should get same result as Part 2. Inputs are A, B, C and D and output is O.

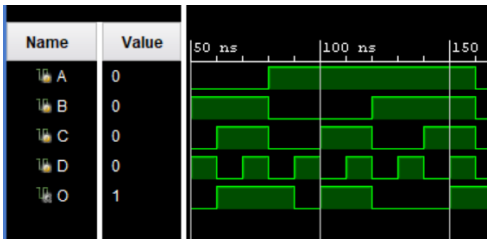

Figure 43: Screenshot of the simulation of Part 4 circuit we designed. Behaves as similar to Part 2

3.5 Part 5

$F_2(a, b, c) = a\bar{b}c + \bar{a}bc$ 101 and 011 outputs 5 and 3

$F_3(a, b, c) = ab\bar{c} + ab$ 111 and 11- outputs 6 and 7

We should expect output of the function 2 as 1 in cases of 101 and 001. Output of the function 3 should be 1 in cases of 110 and 111.

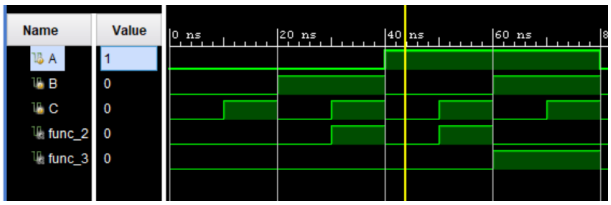

Figure 44: Screenshot of the simulation of Part 5 circuit we designed. Behaves as expected.

3.6 Part 6

Simulation for the 1-bit Half Adder. A and B are inputs, Sum and Carry are outputs.

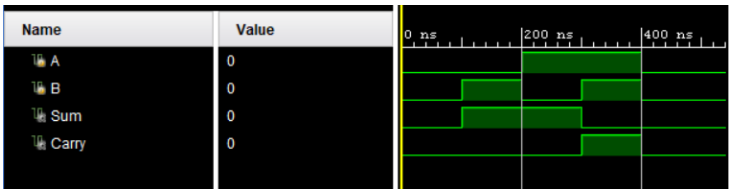

Figure 45: Screenshot of the simulation of 1-bit Half Adder circuit we designed. Behaves as expected.

3.7 Part 7

Simulation for the 1-bit Full Adder. A, B and Cin are inputs, Sum and Cout are outputs.

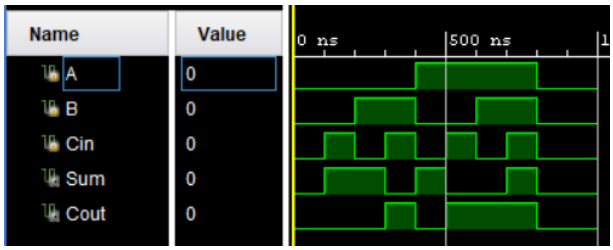

Figure 46: Screenshot of the simulation of 1-bit Full Adder circuit we designed. Behaves as expected.

3.8 Part 8

Simulation for the 4-bit Full Adder. A, B and Cin are inputs, Sum and Cout are outputs. In overflow cases, Cout should be 1.

A	B	Result	A	B	Result
8	1	9	14	5	19 (overflow=3)
4	5	9	11	10	21 (overflow=5)
2	7	9	15	9	24 (overflow=8)
6	3	9	8	12	20 (overflow=4)

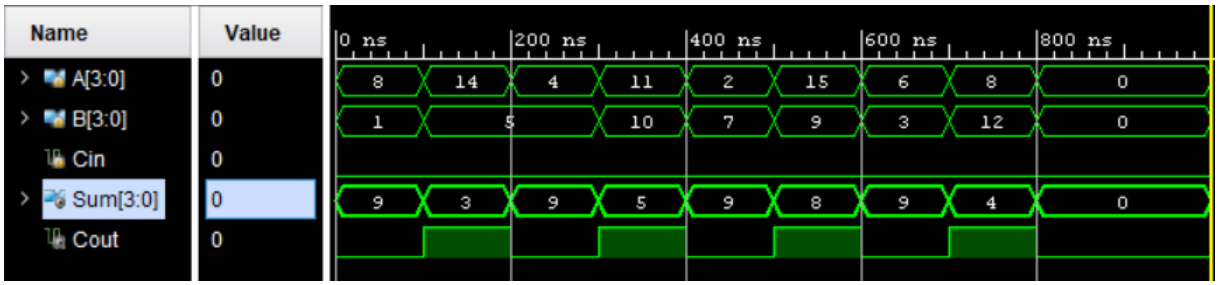

Figure 47: Screenshot of the simulation of 4-bit Full Adder we designed. Behaves as expected.

3.9 Part 9

Simulation for the 8-bit Full Adder. A, B and Cin are inputs, Sum and Cout are outputs. In overflow cases, Cout should be 1.

A	B	Result	A	B	Result
29	5	34	51	92	143
17	28	45	191	2	193
49	25	74	43	59	102
200	95	295 (overflow=39)	78	255	333 (overflow=77)

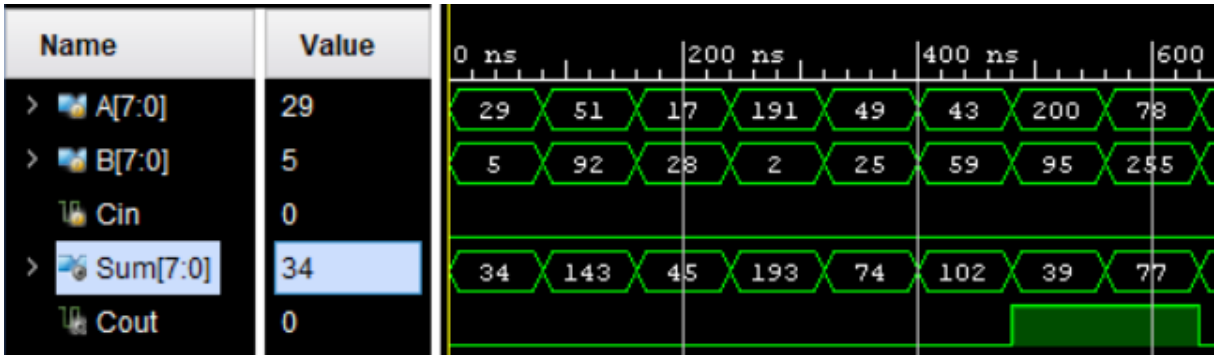

Figure 48: Screenshot of the simulation of 8-bit Full Adder we designed. Behaves as expected.

3.10 Part 10

Simulation for the 16-bit Adder-Subtractor. A, B and Cin are inputs, Sum and Cout are outputs. In overflow cases, Cout should be 1.

A	B	Result	A	B	Result
23	3	26	21	75	96
16800	16900	33700 (ovf=932)	69834 (ovf=4298)	66500 (ovf=964)	5262
45	135	180	36	255	291
25	65	90			

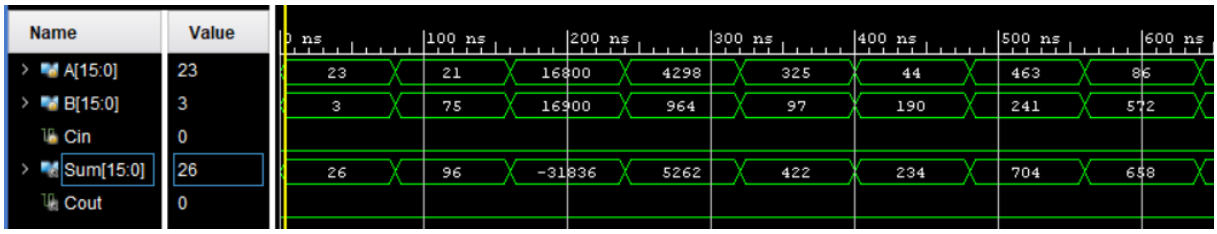

Figure 49: Screenshot of the simulation of 16-bit Adder-Subtractor we designed. Behaves as expected.

3.11 Part 11

Part 11 circuit calculates B - 2A. A, B and Cin are inputs, Sum and Cout are outputs.

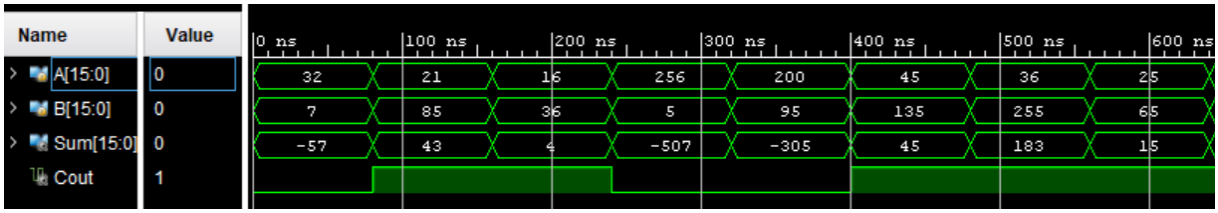

Figure 50: Screenshot of the simulation of 16-bit B - 2A circuit. Behaves as expected.

A	B	Result	A	B	Result	A	B	Result	A	B	Result
32	7	-57	21	85	43	16	36	4	256	5	-507
200	95	-305	45	135	45	36	255	183	25	65	15

4 DISCUSSION

In this homework we created some combinational circuits in accordance to homework description. Firstly in Part 1 we created some basic logic gates such as NOT gate, two input AND gate, three input AND gate, two input OR gate, three input OR gate, two input NAND gate, three input NAND gate, two input XOR gate as modules in Verilog then with using them we created 8:1 Multiplexer and 3:8 Decoder. We checked whether they work correctly by running simulations in Vivado.

Then from Part 2 to Part 5, we implemented the circuits of given expressions in Preliminaries by writing their code in Verilog and created those circuits by using the modules we defined in Part 1. Then we ran simulations and checked whether they match the correct input-output combinations.

From Part 6 to Part 11, we created arithmetic circuits. In Part 6, an half adder, then using that half adder in Part 7 we created a Full Adder. With that Full Adder in Parts 8 and 9 we created 4-bit Full Adder and 8-bit Full Adder respectively. Then in part 10, we created 16-bit Adder/Subtractor circuit by using the 8-bit Full Adder and XOR gate which we defined them earlier. Then lastly in Part 11, we created a circuit which does B - 2A operation in 16-bits. We created 2A by shifting it to left 1 bit. Then we connected those B and 2A as inputs of 16-bit Adder/Subtractor. To perform subtraction we have given Carry input as 1. We have tested all arithmetic operations and results were as expected.

5 CONCLUSION

In this homework, we have learnt how to use Verilog Hardware Description Language to create combinational logic circuits. It was a struggle to learn it at the beginning but we got used to it. The problem with this homework was we had to use every gate as a module and sometimes assigning correct wires to correct ports were problematic but we got over it.

REFERENCES