

**ISTANBUL TECHNICAL UNIVERSITY**  
**COMPUTER ENGINEERING DEPARTMENT**

**BLG 242E**  
**DIGITAL CIRCUITS LABORATORY**  
**HOMEWORK REPORT**

**HOMEWORK NO : 1**

**LAB SESSION : FRIDAY - 16.30**

**GROUP NO : 18**

**GROUP MEMBERS:**

150200916 : Denis Iurie Davidoglu

150220770 : Onur Baylam

**SPRING 2023**

# Contents

<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
<b>2</b>	<b>MATERIALS AND METHODS</b>	<b>1</b>
2.1	Preliminary . . . . .	1
2.2	Experiment . . . . .	10
<b>3</b>	<b>CONCLUSION</b>	<b>17</b>

# 1 INTRODUCTION

In this homework, preliminary part revises basic digital circuits topics, such as 2's complement notation; minimization of logic functions using Karnaugh diagram and Quine-McCluskey method; implementing the same function with different logic components, such as universal NAND gates, multiplexer, decoder. In the experiment part, basic logic circuits are implemented and simulated in Verilog HDL, in the order of increasing complexity, with the final module being 16-Bit Adder-Subtractor.

## 2 MATERIALS AND METHODS

### 2.1 Preliminary

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

For the given function  $F_1$ , we apply given operations below:

- a) Find its prime implicants using Karnaugh diagram.
- In order to obtain prime implicants of function  $F_1$ , we cluster all **true points** (1s) which are in the adjacent cells in the Karnaugh diagram into same group. Afterwards, we retain the constant literals and remove the changing ones. Moreover, if the truth value of a constant variable is '1', we use **true form** of that variable to write the relevant prime implicant. Otherwise, we use its **complement form**.

**NOTE:** We assume **don't care terms** ( $\phi$ ) to be **true** (1) to create larger groups.

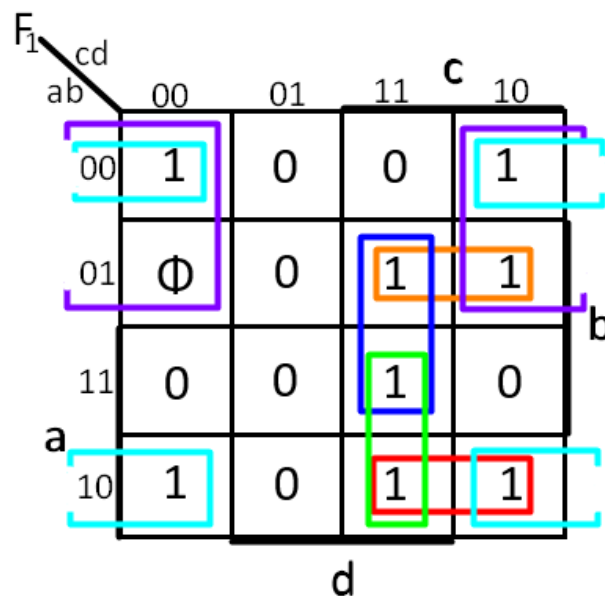


Figure 1: Karnaugh map of the function  $F_1$

Set of all prime implicants:

**a'bc**   **bcd**   **ab'c**   **acd**   **a'd'**   **b'd'**

Figure 2: Prime implicants of the function  $F_1$

b) Find its prime implicants using Quine-McCluskey method.

Step 1		Step 2		Step 3	
Num	abcd	Num	abcd	Num	abcd
0	0000✓	2, 0	00-0✓	10, 8, 2, 0	-0-0
2	0010✓	8, 0	-000✓	6, 4, 2, 0	0-0
8	1000✓	4, 0	0-00✓		
4	0100✓	6, 2	0-10✓		
6	0110✓	10, 2	-010✓		
10	1010✓	10, 8	10-0✓		
7	0111✓	6, 4	01-0✓		
11	1011✓	7, 6	011-		
15	1111✓	11, 10	101-		
		15, 7	-111		
		15, 11	1-11		

The rows without checkmark are unmatched and thus are the prime implicants:  $\bar{a}bc, a\bar{b}c, bcd, acd, \bar{b}\bar{d}, \bar{a}\bar{d}$ .

c) Create prime implicant chart and find the expression with the minimum cost with 2 units of cost for each variable and 1 unit of cost for complement of a variable.

First of all, we label each prime implicant with a symbol (letter). Afterwards, we calculate the cost of each prime implicant.

	<b>a'bc</b>	<b>bcd</b>	<b>ab'c</b>	<b>acd</b>	<b>a'd'</b>	<b>b'd'</b>
<b>Symbols:</b>	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>	<b>F</b>
<b>Costs:</b>	<b>7</b>	<b>6</b>	<b>7</b>	<b>6</b>	<b>6</b>	<b>6</b>
<b>Covered Points:</b>	<b>6,7</b>	<b>7,15</b>	<b>10,11</b>	<b>11,15</b>	<b>0,2,6</b>	<b>0,2,8,10</b>

Figure 3: Prime implicants with symbols, costs and covered points

**NOTE:** Since there is no need to cover **don't care terms** ( $\phi$ ). We do not need to include them to the prime implicant chart.

**Prime Implicant Chart:**

PI	0	2	6	7	8	10	11	15	Cost
A			X	X					7
B				X				X	6
C						X	X		7
D							X	X	6
E	X	X	X						6
F	X	X			X	X			6

Figure 4: Prime implicant chart

**Step 1:** In the prime implicant chart, **F** is an essential prime implicant which covers the distinguished point **8**. Therefore, we need to select prime implicant **F** to be able to cover point **8**.

We mark F to indicate that it will be included to the minimal covering sum.

PI	0	2	6	7	8	10	11	15	Cost
A			X	X					7
B				X				X	6
C						X	X		7
D							X	X	6
E	X	X	X						6
F	X	X			X	X			6

Figure 5: Prime implicant **F** is selected

The new form of the prime implicant chart:

PI	6	7	11	15	Cost
A	X	X			7
B		X		X	6
C			X		7
D			X	X	6
E	X				6

Figure 6: New form of the prime implicant chart

**Step 2:** In the prime implicant chart, **D** covers **C** and the cost of **D** is smaller than the cost of **C**. So, we can remove **C** from the chart.

PI	6	7	11	15	Cost
A	X	X			7
B		X		X	6
<del>C</del>			<del>X</del>		<del>7</del>
D			X	X	6
E	X				6

Figure 7: **C** is removed from the chart

The new form of the prime implicant chart:

PI	6	7	11	15	Cost
A	X	X			7
B		X		X	6
D			X	X	6
E	X				6

Figure 8: New form of the prime implicant chart

**Step 3:** In the new form of the chart, **D** is an essential prime implicant which covers the distinguished point **11**. Therefore, we select **D** and remove it from the chart.

PI	6	7	11	15	Cost
A	X	X			7
B		X		X	6
<del>D</del>			X	X	6
E	X				6


Figure 9: Prime implicant **D** is selected

The new form of the prime implicant chart:

PI	6	7	Cost
A	X	X	7
B		X	6
E	X		6

Figure 10: New form of the prime implicant chart

**Step 4:** To cover all true points with minimum cost, we need to select prime implicant **A**. As a result, we select **A** and form our minimal covering sum.



PI	6	7	Cost
<b>A</b>	X	X	<b>7</b>
B		X	6
E	X		6

Figure 11: Prime implicant **A** is selected

Minimal Covering Sum:

$$F_1(a, b, c, d) = \boxed{a'bc} + \boxed{acd} + \boxed{b'd'}$$

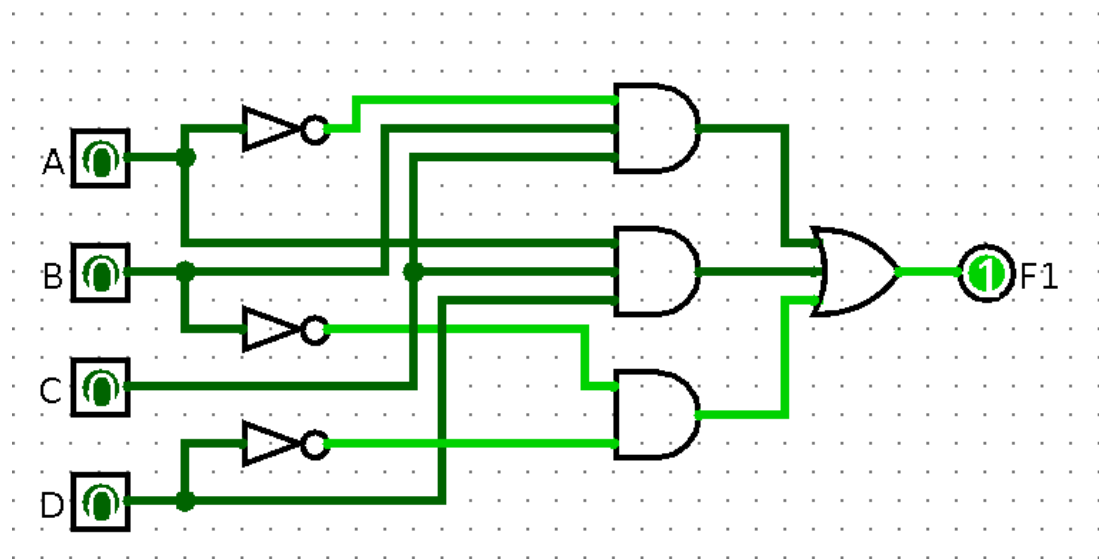
Symbols:                      A                      D                      F

**Total cost:**                      7                      +                      6                      +                      6                      = 19

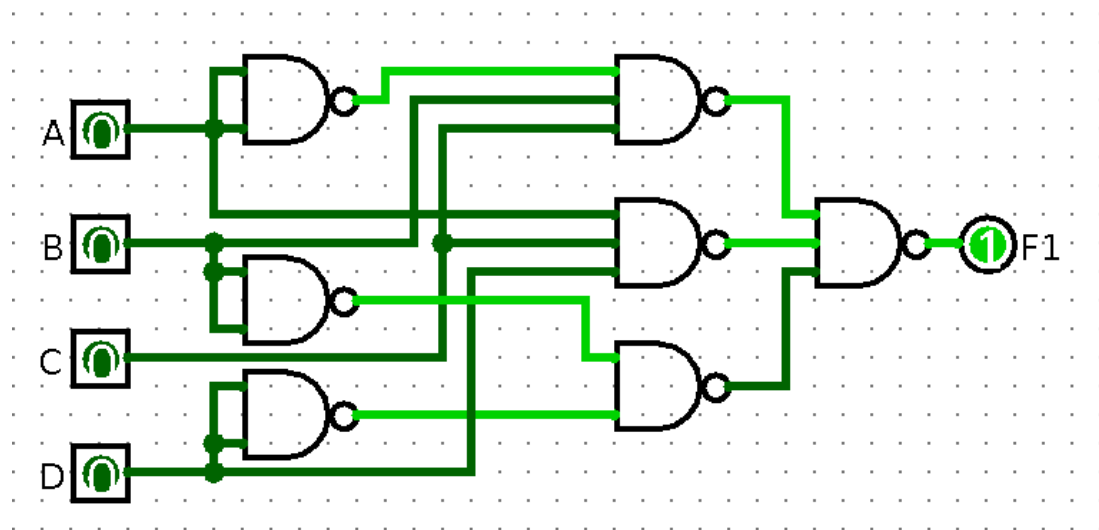
Figure 12: The expression with the minimum cost for function  $F_1$



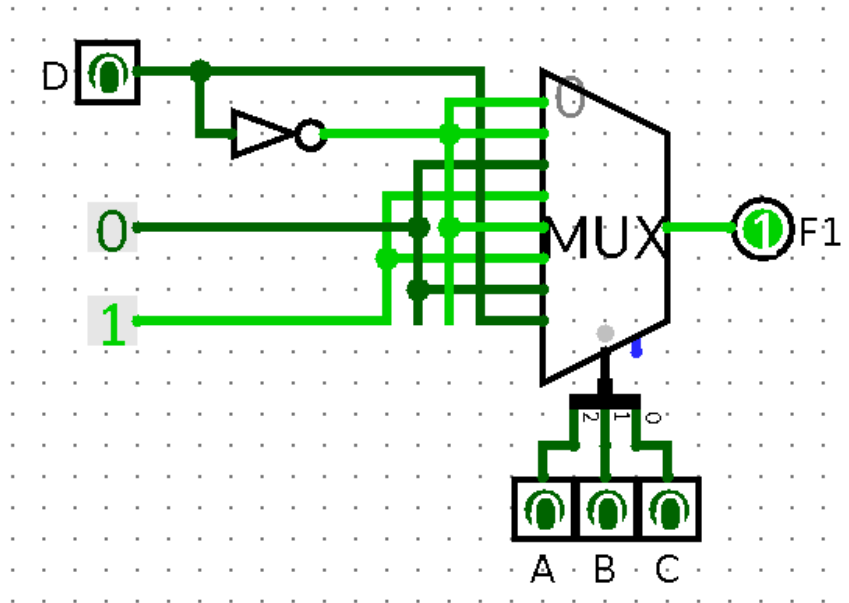
d) Design and draw the lowest cost expression using NOT, AND, and OR gates.



e) Design and draw the lowest cost expression using only NAND gates.

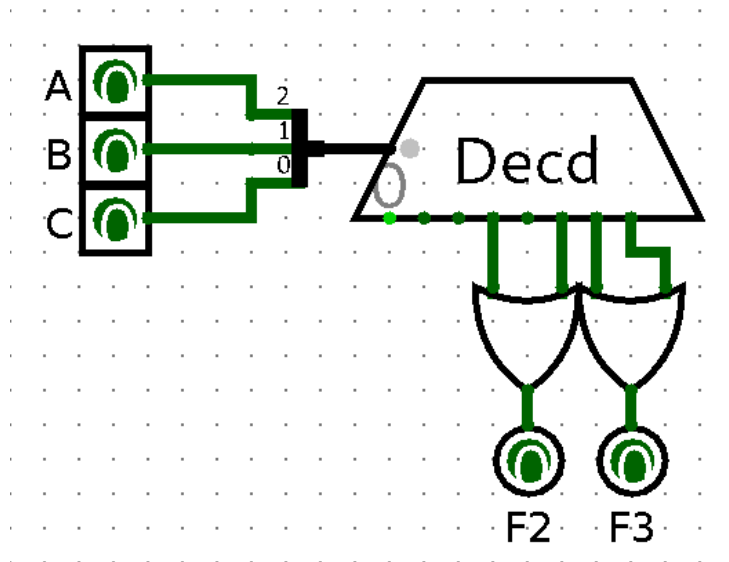


- f) Design and draw the lowest cost expression using a single 8:1 Multiplexer, AND, OR and NOT gates.



2. Design and draw the functions  $F_2$  and  $F_3$  given using ONE single 3:8 decoder, 2-input OR gates.

$$F_2(a, b, c) = \bar{a}bc + a\bar{b}c, F_3(a, b, c) = ab\bar{c} + ab$$



3.
  - Recall signed and unsigned addition for binary numbers in 2's complement notation.
  - Recall signed and unsigned subtraction for binary numbers in 2's complement notation.

In 2's complement notation, both addition and subtraction use the same table:

<i>a</i>	<i>b</i>	<i>sum</i> <i>carry</i>	
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Addition is performed without any changes to the numbers, both for unsigned and signed. Only the most significant bit is interpreted differently, being reserved for sign in the signed representation, and used normally in the unsigned:

$$\begin{array}{r}
 \begin{array}{r}
 \phantom{0}^1 \\
 001000010 \\
 + 000011011 \\
 \hline
 001011101
 \end{array}
 \qquad
 \begin{array}{r}
 \phantom{0}^1 \phantom{0}^1 \phantom{0}^1 \phantom{0}^1 \phantom{0}^1 \phantom{0}^1 \\
 011001001 \\
 + 011111111 \\
 \hline
 111001000
 \end{array}
 \end{array}$$

Subtraction, on the other hand, firstly requires transforming a number into 2's complement, which is done by flipping the bits and adding one. For example, to compute  $(01010011)_2 - (00000101)_2$ , instead of performing subtraction, addition with the 2's complement of the subtrahend can be done:

$$(\overline{00000101})_2 + 1 = (11111010)_2 + 1 = (11111011)_2$$

$$\begin{array}{r}
 \phantom{0}^1 \phantom{0}^1 \phantom{0}^1 \phantom{0}^1 \phantom{0}^1 \phantom{0}^1 \\
 001010011 \\
 + 011111011 \\
 \hline
 101001110
 \end{array}$$

The extra bit that arises after addition will be ignored and not counted as part of result; however, it can be used to interpret which number was bigger, if there was overflow or borrow, if the result can be represented, all depending on the type of operation and whether the numbers were signed or unsigned.

## 2.2 Experiment

- Part 1: Simulation results of AND, OR, NOT, XOR, NAND, 8:1 Multiplexer and 3:8 Decoder modules

**AND Module Simulation Result:** This module operates as an AND gate.



Figure 13: AND module simulation result

**OR Module Simulation Result:** This module operates as an OR gate.

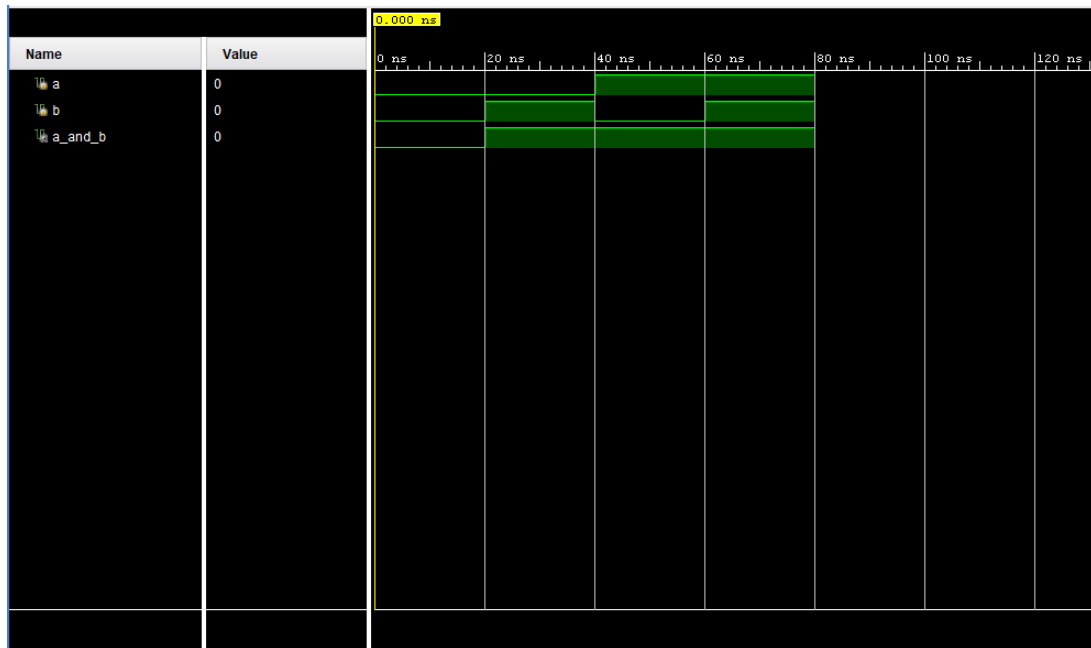


Figure 14: OR module simulation result

**NOT Module Simulation Result:** This module operates as a NOT gate.



Figure 15: NOT module simulation result

**XOR Module Simulation Result:** This module operates as an XOR gate.

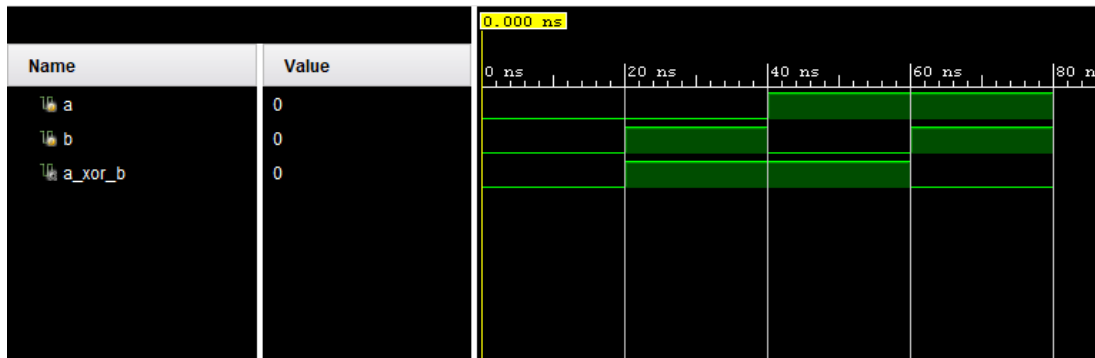


Figure 16: XOR module simulation result

**NAND Module Simulation Result:** This module operates as a NAND gate.



Figure 17: NAND module simulation result

**8:1 Multiplexer Module Simulation Result:** This module operates as an 8:1 Multiplexer.



Figure 18: 8:1 Multiplexer module simulation result

**3:8 Decoder Module Simulation Result:** This module operates as a 3:8 Decoder.

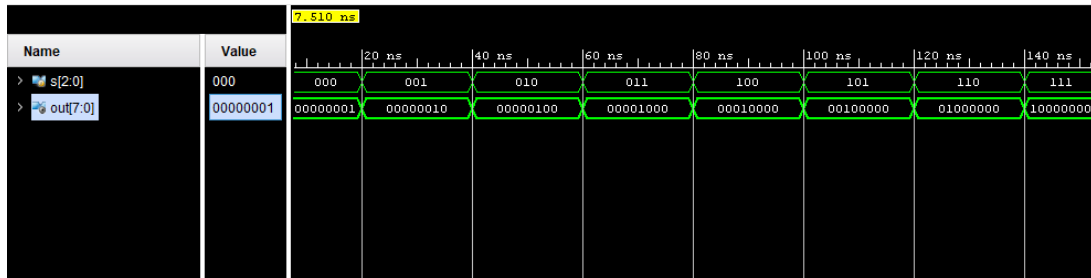


Figure 19: 3:8 Decoder module simulation result

- **Part 2: Simulation result of  $F_1$  in Preliminary 1.d.**

Simulation result of the function  $F_1$  designed in Preliminary 1.d. section using NOT, AND, and OR modules.

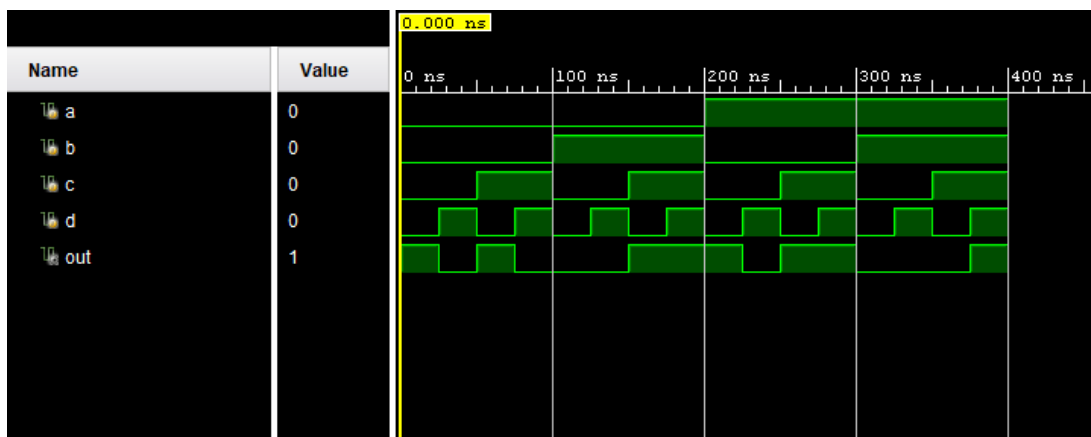


Figure 20: Preliminary 1.d.  $F_1$  simulation result

- **Part 3: Simulation result of  $F_1$  in Preliminary 1.e.**

Simulation result of the function  $F_1$  designed in Preliminary 1.e. section using only NAND modules.

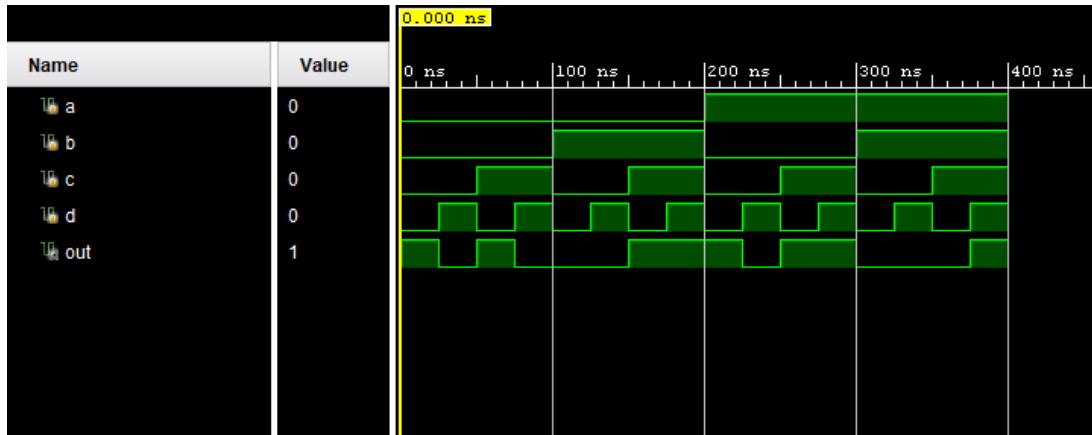


Figure 21: Preliminary 1.e.  $F_1$  simulation result

- **Part 4: Simulation result of  $F_1$  in Preliminary 1.f.**

Simulation result of the function  $F_1$  designed in Preliminary 1.f. section using a 8:1 multiplexer, AND, OR and NOT gates.

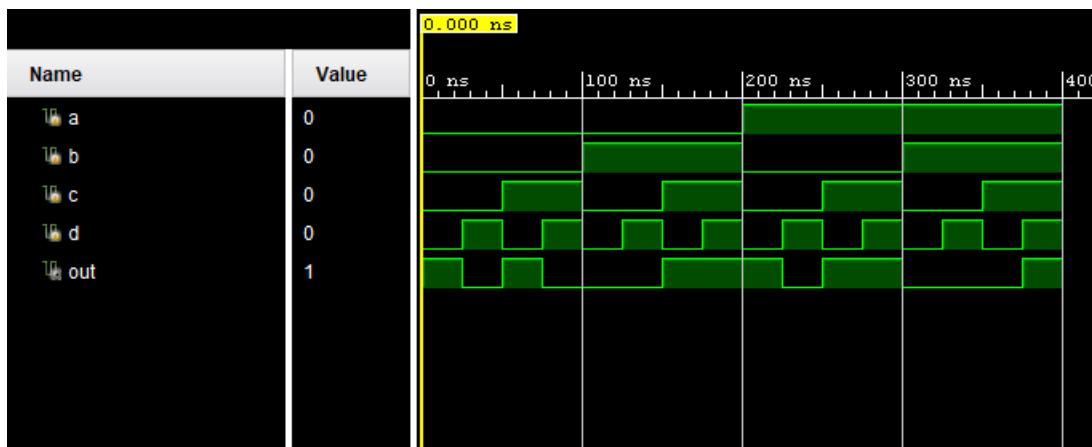


Figure 22: Preliminary 1.f.  $F_1$  simulation result



- **Part 5: Simulation results of  $F_2$  and  $F_3$  in Preliminary 2.**

Simulation result of the function  $F_2$  and  $F_3$  designed in Preliminary 2 section using 3:8 Decoder and OR modules.

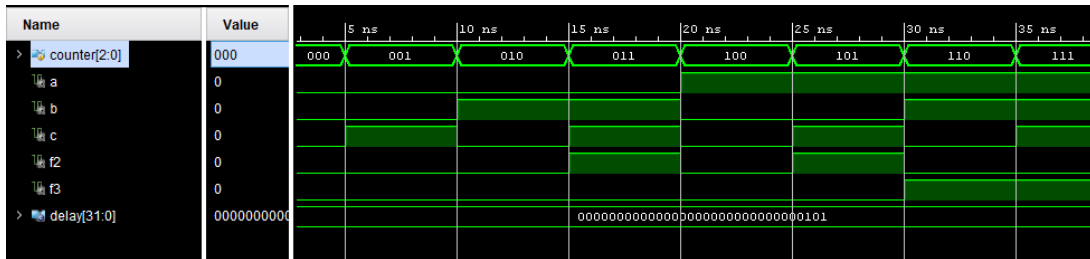


Figure 23: Preliminary 2.  $F_2$  and  $F_3$  simulation result

- **Part 6: Simulation result of 1-Bit Half Adder.**

Simulation result of 1-Bit Half Adder using AND, OR, NOT, XOR modules.

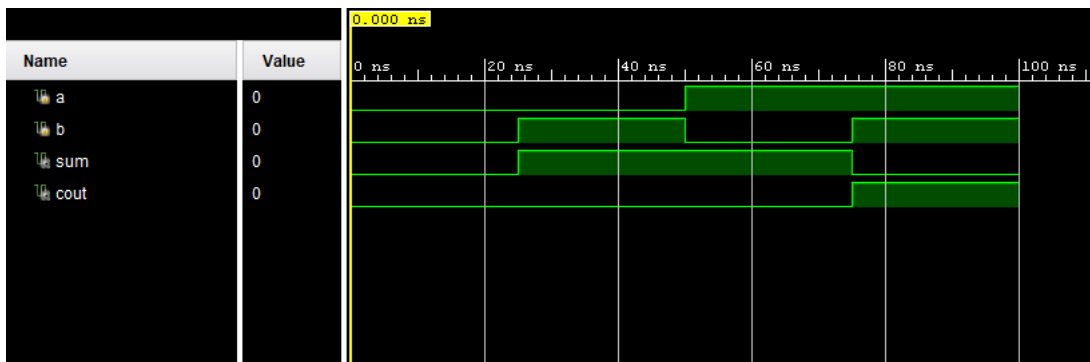


Figure 24: 1-Bit Half Adder Simulation Result

- **Part 7: Simulation result of 1-Bit Full Adder.**

Simulation result of 1-Bit Full Adder using half adder and OR modules.

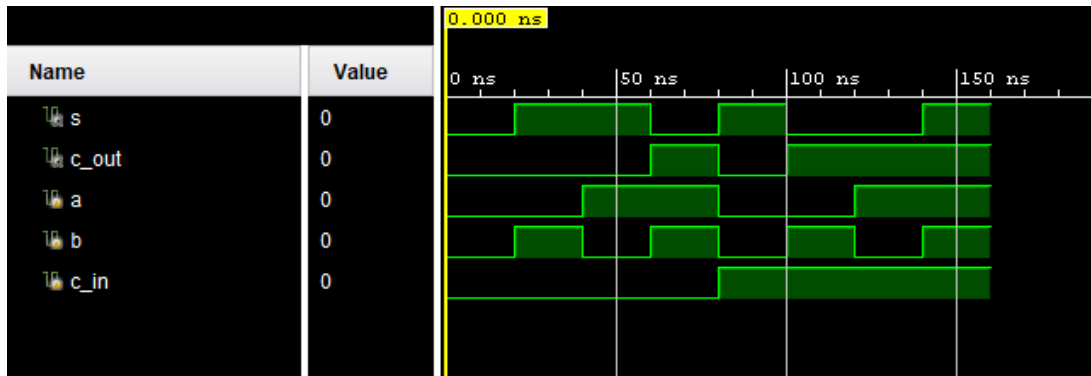


Figure 25: 1-Bit Full Adder Simulation Result

- **Part 8: Simulation result of 4-Bit Full Adder.**

Simulation result of 4-Bit Full Adder using 1-Bit Full Adder modules.

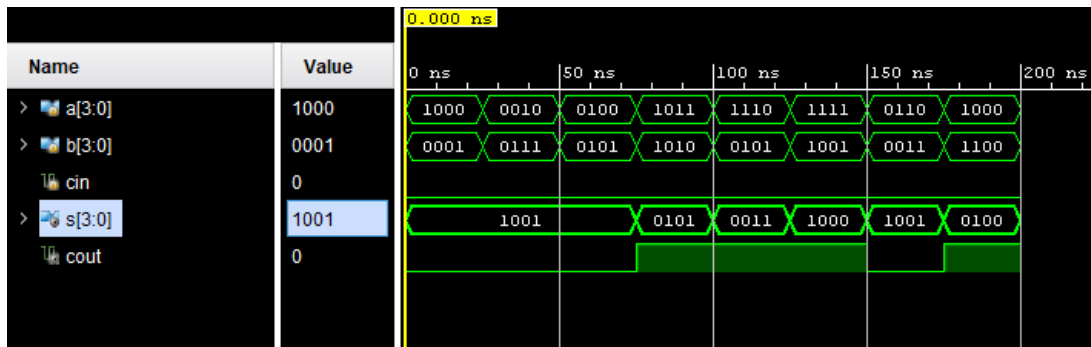


Figure 26: 4-Bit Full Adder Simulation Result

- **Part 9: Simulation result of 8-Bit Full Adder.**

Simulation result of 8-Bit Full Adder by using 1-Bit Full Adder modules.

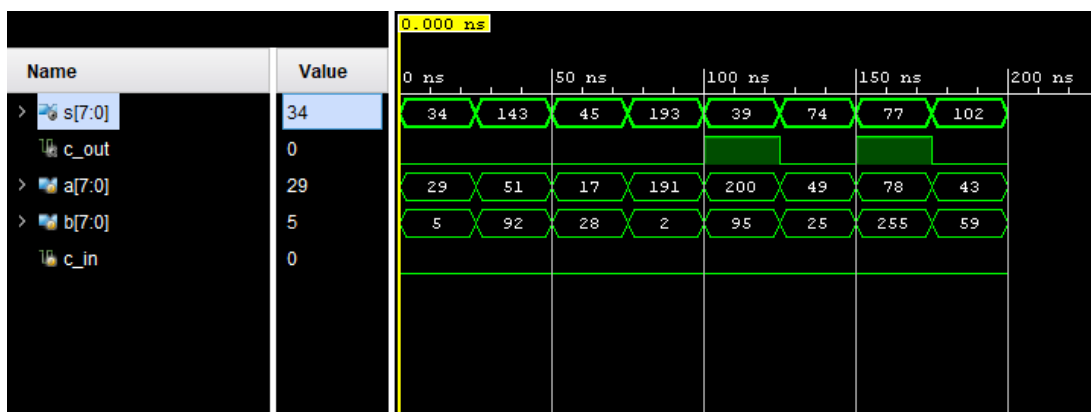


Figure 27: 8-Bit Full Adder Simulation Result

- **Part 10: Simulation result of 16-Bit Adder-Subtractor**

Simulation result of 16-Bit Adder-Subtractor by using 8-Bit Full Adder and XOR modules.

**NOTE:** The circuit operates on unsigned 16-bit binary numbers.

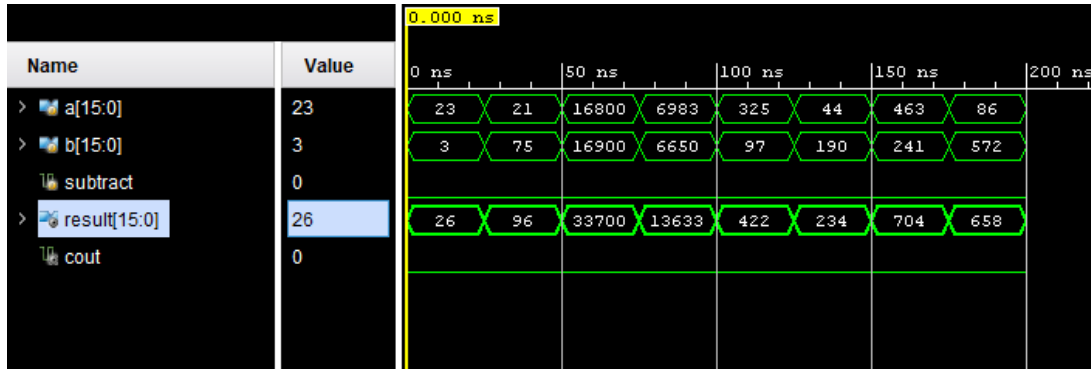


Figure 28: 16-Bit Adder-Subtractor Simulation Result

- **Part 11: Simulation result of the circuit which calculates B-2A**

Simulation result of B-2A by using 16-Bit Adder-Subtractor, Adder, NOT, XOR, AND, and OR modules.

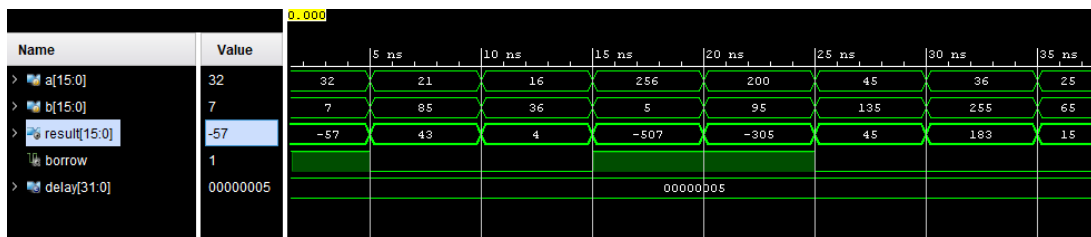


Figure 29: Simulation result of B-2A calculator module

### 3 CONCLUSION

In this homework, we have applied our knowledge from the digital circuits course by deriving least cost expressions with human-friendly Karnaugh maps and a more algorithmic Quine-McCluskey method; facing the challenge to use only one kind of logic components, which happens in real life; learned to use Verilog HDL instead of Logisim for simulating circuits; built a fully working 16-bit adder-subtractor, whose functionality can be tracked down to the basic two-input gates. While using Verilog HDL, simulating circuits and interpreting results was harder than designing the circuit in the first place because as the circuit gets more complex, the number of possible outcomes increases and one should be clever enough to provide proper tests with edge cases, instead of brute-forcing all input combinations. It is no surprise that the name Verilog is a portmanteau of

the words "verification" and "logic". Although challenging with features unusual to most programming languages, it is very powerful for creating virtual hardware and verifying real devices. With the vast options for exporting the results of a simulation, such as exporting to waveform viewing programs, it dominates over Logisim or, even worse, truth table-generating websites. One disadvantage is that the circuit described is not available as a schematic. Nonetheless, we felt more like engineers as we designed and simulated digital circuits with this toolbox.