ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E
DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	1
EXPERIMENT DATE	:	24.03.2021
LAB SESSION	:	FRIDAY - 10.30
GROUP NO	:	G1

GROUP MEMBERS:

150170908	:	MUHAMMAD RIZA FAIRUZZABADI
070170364	:	AİŞE HÜMEYRA BOZ

SPRING 2021

Contents

1 INTRODUCTION

In this project, we were required to complete two distinct tasks, preliminary and experiment. First in preliminary part, we found a incompletely specified function's prime implicates using both Karnaugh diagram and Quine-McCluskey method. Then by taking into account of each variable's costs, we simplified the equation and found the minimum cost.We designed and drew the lowest cost expression with NOT, AND, OR gates then, transformed it to only NAND gates.Finally in preliminary part, we drew the lowest cost expression using a 8:1 Multiplexer and NOT gates. We designed and drew the second and third functions using 3:8 decoder, 2-input OR gates. In addition, we implemented the previous equations using Vivado Design Suite by preparing modules. After, we simulated each one of them and validated their truths.

2 MATERIALS AND METHODS

2.1 MATERIALS

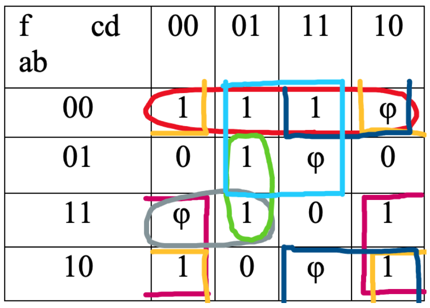
Tools Used

- Vivado Design Suite - Xilinx
- Latex (overleaf.com)
- Logisim

Firstly we designed our preliminary logic circuits in Logisim and then we design, implement and program the logic circuits to alongside their NOT,AND,OR gates modules in Vivado Design Suite, lastly we used overleaf.com to prepare the report document in LaTex.

2.2 PRELIMINARY

In the first question, we found F1(a, b, c, d) = 1(0, 1, 3, 5, 8, 10, 13, 14) + (2, 7, 11, 12) equation's prime implicants using Karnaugh diagram and Quine-McCluskey method. In Karnaugh map, 's are assumed to be 1 when we are finding the set of all prime implicants in Karnaugh map in SOP form. Karnaugh map can be seen below and its prime implicants can be detected since there cannot be any more simplification among them. Prime implicants are named as $A = a'.b', B = b'.d', C = a'.d, D = b'.c, E = a.d', F = b.c'.d, G = a.b.c'$.



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

Figure 1: Function 1 represented by Karnaugh Map

In second part, we found the same expression's prime implicants using Quine McKluskey method. We obtained the same prime implicants as in the first question that can be observed below.

Num	a, b, c, d	
0	0000	✓
1	0001	✓
2	0010	✓
8	1000	✓
3	0011	✓
5	0101	✓
10	1010	✓
12	1100	✓
7	0111	✓
11	1011	✓
13	1101	✓
14	1110	✓

Num	a, b, c, d	
0,1	000-	✓
0,2	00-0	✓
0,8	-000	✓
1, 3	00-0	✓
1, 5	0-01	✓
2, 3	001-	✓
2, 10	-010	✓
8,10	10-0	✓
8, 12	1-00	✓
3, 7	0-11	✓
3,11	-011	✓
5, 7	01-1	✓
5, 13	-101	
12, 13	110-	
12, 14	11-0	✓
10, 11	101-	✓
10,14	1-10	✓

Num	a, b, c, d
0,1,2,3	00--
0,2,1,3	00--
0,2,8,10	-0-0
0,8,2,10	-0-0
1,3,5,7	0--1
1,5,3,7	0--1
2,3,10,11	-01-
2,10,3,11	-01-
8,10,12,14	1--0
8,12,10,14	1--0

Figure 2: Function 1 represented by Quine McKluskey

Next, we created a prime implicant chart and found the expression with minimum cost. It was given as 2 units of cost for each variable and 1 unit of cost for complement of a variable.

Step.1: In this chart, 14 is the distinguished point. As E is essential prime implicants, its rows and columns are removed from chart and they are marked to be shown in the final set.

Step.2: In this chart, F covers G. Since cost of F is same as cost of G, G is removed

	0	1	3	5	8	10	13	14	Cost
A	X	X	X						6
B	X				X	X			6
C		X	X	X					5
D			X			X			5
E					X	X		X	5
F				X			X		7
G							X		7

Figure 3: Prime Implicant Chart - First Step

	0	1	3	5	13	Cost
A	X	X	X			6
B	X					6
C		X	X	X		5
D			X			5
F				X	X	7
G					X	7

Figure 4: Prime Implicant Chart - Second Step

from chart. Similarly, A covers B. Since cost of A is same as cost of B, B is removed from chart. Step.3: In this chart, 0,13 are distinguished points. Therefore A and F are

	0	1	3	5	13	Cost
A	X	X	X			6
C		X	X	X		6
D			X			5
F				X	X	7

Figure 5: Prime Implicant Chart - Last Step

selected and marked. Then all points are selected and an expression consisting A, F, E is the lowest cost expression.

$$F = (a'.b' + a.d' + b.c'.d)$$

Then, we designed and drew the lowest cost expression using gates in a variety. First one is with NOT, AND, OR and second one is comprised of NAND gates. While, last one is with one 8:1 Multiplexer and NOT gates.

Lastly in the second question, we were given two logical expressions of $F(a, b, c)= abc' + a'c$ and $F(a, b, c)= ab'c' + bc$ were asked to design and draw the functions using ONE single 3:8 Decoder. To make it so, we need to take the operands of abc', a'c, ab'c' and bc of both expressions from the decoder. From the Decoder Truth Table in Figure 7,

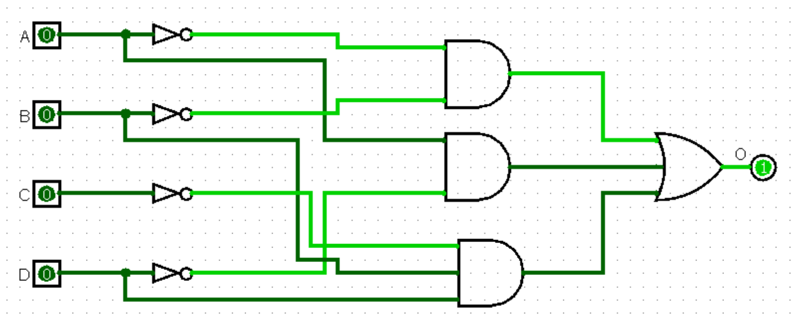


Figure 6: Lowest cost expression using NOT, AND, and OR gates

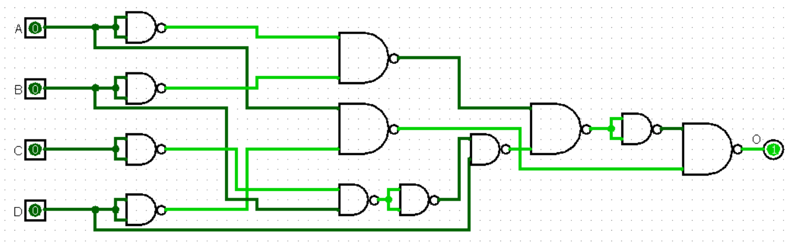


Figure 7: Lowest cost expression using only NAND gates

it can be seen that abc' and ab'c' can be selected through O4 and O5 outputs. However, to get a'c and bc we need to take all outputs that consist the operands (O1 and O4, O4 and O7) and get them through OR gates before being operated alongside their pairs. The 3:8 Decoder Design is as shown in Figure 8.

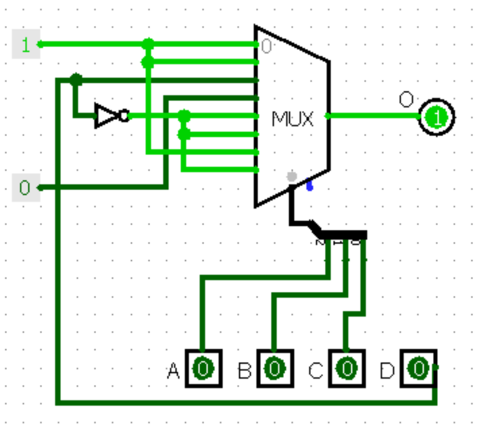


Figure 8: Lowest cost expression using 8:1 Multiplexer and NOT gates

A	**B**	**C**	**D**	**F**	
0	0	0	0	1	1
0	0	0	1	1	
0	0	1	0	1	1
0	0	1	1	1	
0	1	0	0	0	D
0	1	0	1	1	
0	1	1	0	0	0
0	1	1	1	0	
1	0	0	0	1	D'
1	0	0	1	0	
1	0	1	0	1	D'
1	0	1	1	0	
1	1	0	0	1	1
1	1	0	1	1	
1	1	1	0	1	D'
1	1	1	1	0	

Figure 9: Multiplexer Truth Table

2.3 PART 1

In the first part of this experiment we were asked to implement the circuit we have designed in Preliminary 1.d. in Verilog. We used NOT AND OR modules which we implemented in the previous experiment. For AND module, we chose A and B as inputs and C as an output. Then we assigned C as A&B using assign statement. For NOT module, A was the input and B was the output. We assigned B as $B = \sim A$. In the last module which is OR, we assigned C as the output and A and B as the inputs. Through assign statement, we obtained C = $A\ |B$. We implemented the design as module with name P1. Firstly, it takes A, B, C, and D wires as inputs, and O as outputs. Then, we first took the negates of every inputs by instantiating NOT modules for each one of them, and the negated values are stored/loaded as nota, notb, notc, and notd, respectively. We then implemented the required AND operations by instantiating AND modules 4 times. First we did the $a'b'$ equation by setting nota and notb as input and NANB as output

A	**B**	**C**	**X**	O7	O6	O5	O4	O3	O2	O1	O0
0	0	0	A' B' C'								1
0	0	1	A' B' C							1	
0	1	0	A' B C'						1		
0	1	1	A' B C					1			
1	0	0	A B' C'				1				
1	0	1	A B' C			1					
1	1	0	A B C'		1						
1	1	1	A B C	1							

Figure 10: Decoder Truth Table

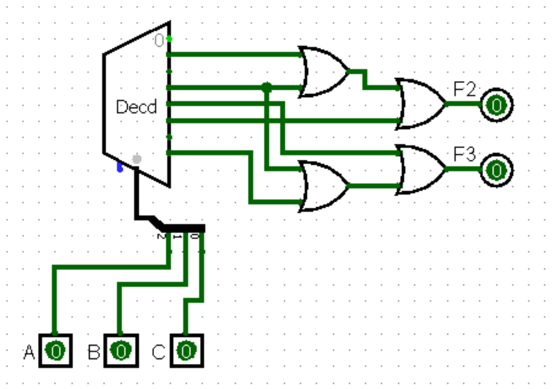


Figure 11: 3:8 Decoder Design

arguments. In second instantiation we did the ad' operation by setting A and notd as input and NDA as output arguments. Thirdly we did the $bc'd$ equation by first getting bc' value by instantiating AND with notc and B as input an BNCD as output arguments, then we calculate that newly taken BNC value with D within the 4th AND instantiation, giving us the output as BNCD. Lastly, we implement the OR operation by instantiating the OR modules twice, since our module only takes two input. First we calculate $a'b' + ad'$ by setting NANB and NDA as input and O1 as output arguments. We then calculate that O1 with BNCD in the last instantiation (or11), giving us output O which is the result of the equation $a'b' + ad' + bc'd$. Following the implementation, we created a new simulation module inside our simulation.v file, naming it Part1. Within it, we declared 4 reg data types i1, i2, i3 and i4 as outputs, and wire o as outputs. We then instantiate/call the module P1 with those inputs and outputs. Inside initial block, we set i1, i2, i3, and i4 with all of their possible values, which can be seen in the expression's truth table in Figure 14. We set the delay to 50 ns. The result of the simulation is shown in Figure 13. As we can we see, the outputs in the simulation are the same with the ones in the truth table, indicating its correctness. The elaborated design schematics of Part 1 can be seen in Figure 12.

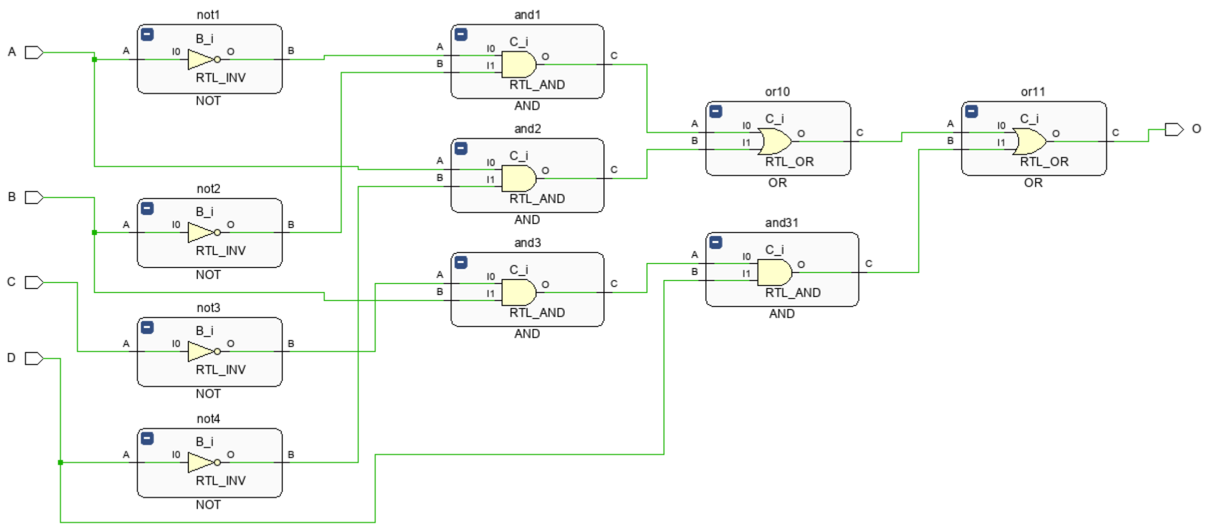


Figure 12: Elaborated Design Schematic of Part 1

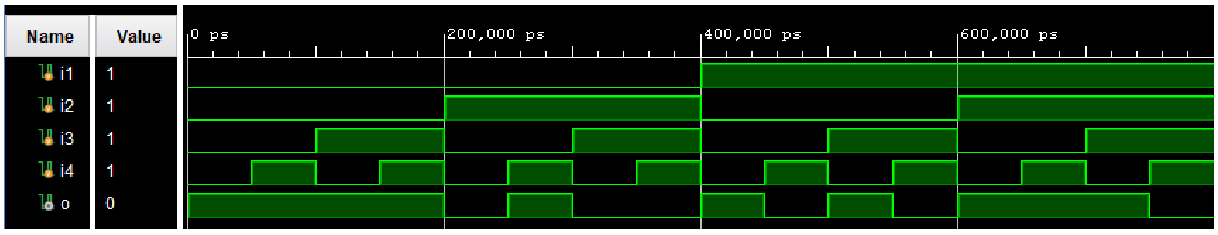


Figure 13: P1 Simulation with Delay 50

2.4 PART 2

In the second part of the experiment, we were tasked to design and implement the logic circuits for the given expression in Preliminary 1.e. section using only NAND modules in Verilog and simulate it to validate its correctness. To implement the expression, we add a new module along with the previously created NOT, AND, OR modules which is called NAND. We set A,B as inputs and C as an output.Then, assigned C as complement of A AND B. In addition to that, we created a new module that is called P2 for this specific question part. Since it was required to use only NAND gates, we instantiated just NAND module. For P2 module, we at first set A,B,C,D as inputs and O as an output. We instantiate NAND operations first and put A,B,C,D as inputs for each operation and obtained nota,notb,notc,notd for outputs. In order to obtain $(a'b')'$ value, we set nota,notb as inputs and NANDAB as an output. In order to get $((ad')'$ value, we set nota,notb as inputs and NANDAB as an output.Then, we instantiate again NAND module and set notc, B and NANDBNC as arguments, showing $(bc')'$. We repeated the process by setting NANDBNC, NANDBNC and notNANDBNC as arguments, showing $(bc'$ as getting the complementary of the expression. Then, we instantiate NAND module put notNANDBNC, D as inputs and NANDBNCD as an output, getting $((bc'd)'$. Then we started to combine our results, starting from NANDAB, NANDBNCD as inputs and o1 as an output $((a'b')'(bc'd)')'$. We obtained the complementary of this expression, $(a'b')'(bc'd)'$. Lastly, we put NANDo1, NANDAND, O as arguments obtaining a'b'+bc'd+ad'. We implemented our simulation module which has the same operations and variables as part 1 that can be observed in Figure 16. To validate its correctness, we added a truth table along with simulation. The elaborated design schematics of Part 2 can be seen in Figure 15.

A	**B**	**C**	**D**	**F**
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Figure 14: Truth table of the expression

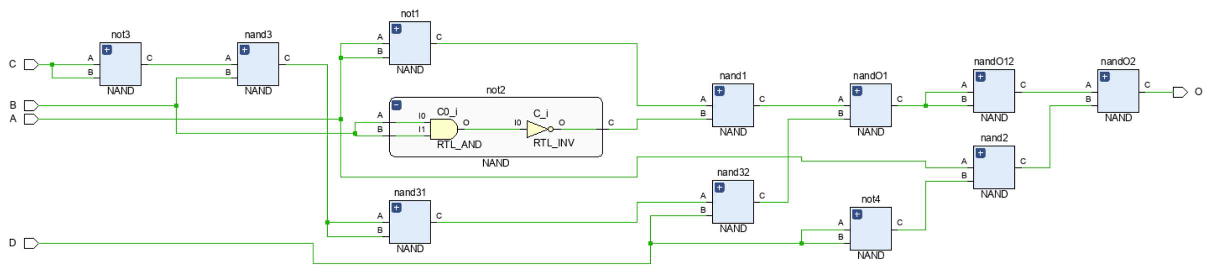


Figure 15: Elaborated Design Schematic of Part 2

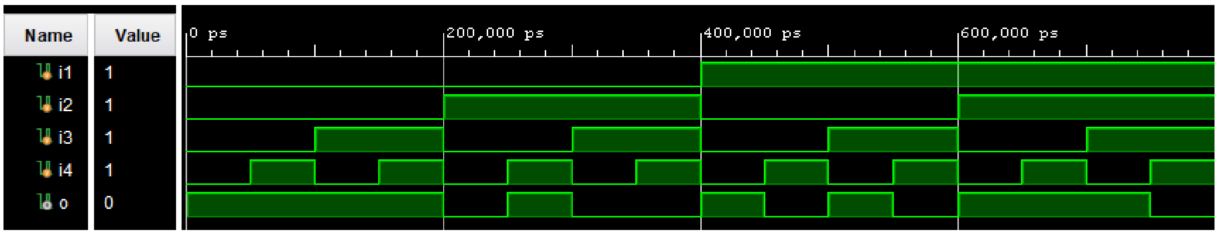


Figure 16: P2 Simulation with Delay 50

2.5 PART 3

In the third part of the experiment, we were required to implement the circuit that we have designed in Preliminary 1.f, which is the lowest cost expression of F1, in 8:1 Multiplexer and NOT gates in Verilog. The initial design of the mentioned Multiplexer is shown in Figure 8. For this we created two separate modules inside modules.v. First one is to implement the general MUX circuit, and the second one is to implement the extras we have in our expression, which is having an extra input D that is taken not as selector but as Multiplexer input. In the first module, which is MUX, we first set I0-I7 as input wire for MUX inputs, alongside A, B, C and D. Then we declared O as output wire as well. Here, we assign O according to this formula $O = (I0\ A\ B\ C)|(I1\ A\ BC)|(I2\ AB\ C)+(I3\ ABC)+(I4A\ B\ C)+(I5A\ BC)+(I6AB\ C)+(I7ABC)$ which will set O to whichever Multiplexer input is supposed to be selected. In the second module, which we named P3, we set A, B, C and D as input wires. We then instantiate the NOT module to get the negate of D as notd, and lastly we lastly designed MUX module with the appropriate arguments (1,1,D,0,NOTD,NOTD,1,NOTD,A,B,C,O). After the implementation, we created another simulation module inside simulation.v file, naming it Part3. Its whole contents (the test values, inputs and outputs) are the same as the previous parts' simulation modules. The result of the simulation is shown in Figure 18. As we can we see, the outputs in the simulation are the same with the ones in the truth table in Figure 9, indicating its correctness. The elaborated design schematics of Part 3 can be seen in Figure 17.

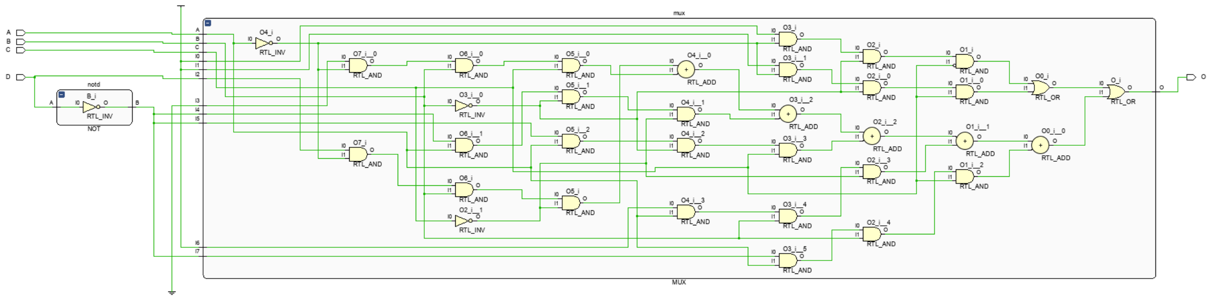


Figure 17: Elaborated Design Schematic of Part 3

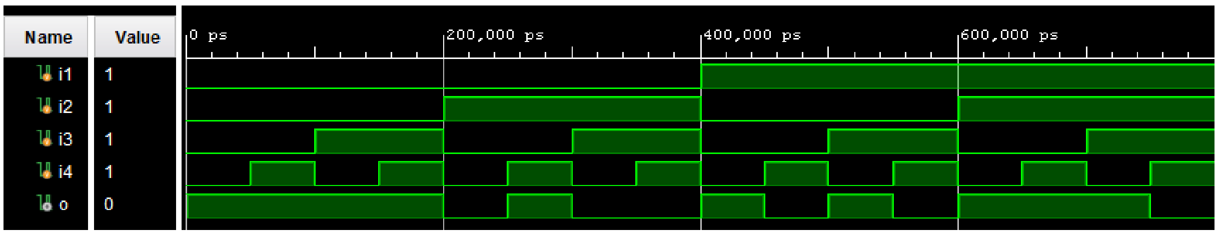


Figure 18: P3 Simulation with Delay 50

2.6 PART 4

In the last, fourth part of the experiment we were tasked to implement the circuit we designed in Preliminary 2 section, which is a 3:8 Decoder. The initial design is shown in Figure 11, alongside its truth table in Figure 10. Just like in the previous part, we also implemented the circuit in two modules, one for general Decoder operation and one for the extras within our expression. In the first DEC module, we set A, B and C as input wires, and we set O1 - O7 as output wires. We then assigned O1 to O7 with their respective pairs according to Truth Table in Figure 10, as shown below.

$$O0 = \tilde{A}\&\tilde{B}\&\tilde{C}$$
$$O1 = \tilde{A}\&\tilde{B}\&C$$
$$O2 = \tilde{A}\&B\&\tilde{C}$$
$$O3 = \tilde{A}\&B\&C$$
$$O4 = A\&\tilde{B}\&\tilde{C}$$
$$O5 = A\&\tilde{B}\&C$$
$$O7 = A\&B\&C$$
$$A.(A+B) = A.B + A.B'$$

Then in the other module which we named P4, we set A, B, and C as input wires, and F2, F3 as output wires. In this module, we instantiated previous DEC module, which through that we give A, B, and C as its input arguments, which then will results in the outputs O1 - O7. We then use those newly acquired O1 - O7 variables' values in the implementation of the expression. For the first expression we instantiated OR as or1 with O1 and O3 as input, giving us OR1 as output. Then we use calculate that OR1 with O5 in the next OR instantiating, resulting in F2. Then for the other expression we did the same instantiations but with O3 and O7 and O4. That concludes the implementation of the module. Lastly, for testing purposes we again created one other simulation module inside the file, named Part4, Unlike the previous ones, here we only declared 3 reg data types i1, i2 and i3, alongside 2 output wires F2 and F3. As we did in the previous simulations, we instantiated/called the P4 module with the inputs and outputs we recently declared/set, then we set every possible values of i1, i2 and i3 inside the initial block for testing. The result of the simulation can be seen in 20. It gives us the correct outputs for each expression, indicating its correctness. Note that we did not connect nor read the output of other decoder values which are not connected to the gates, we only seek for F2 and F3 outputs. The elaborated design schematic of the circuit is shown in Figure 19.

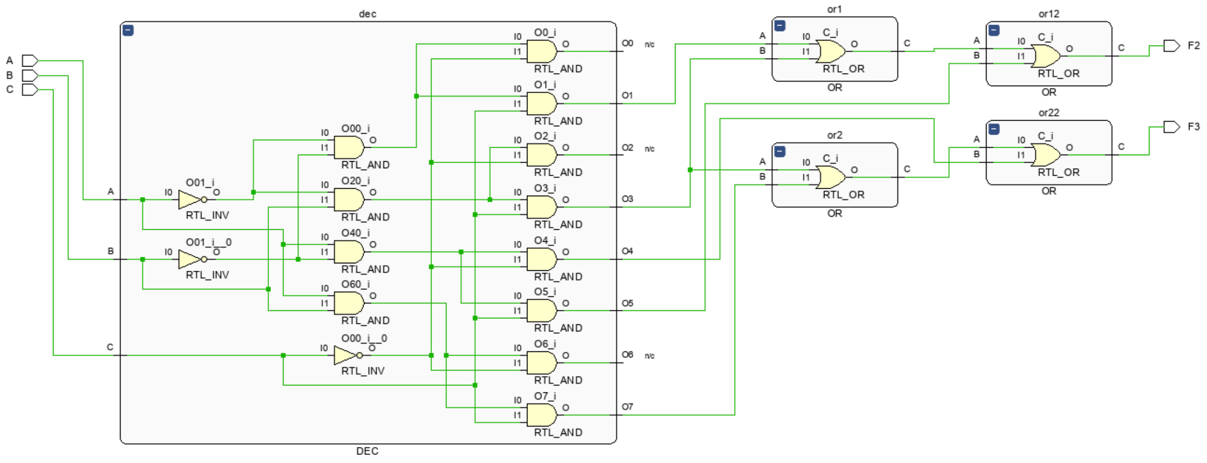


Figure 19: Elaborated Design Schematic of Part 4

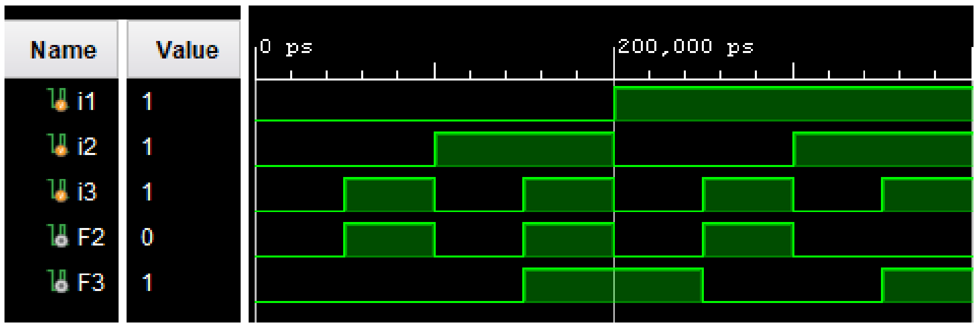


Figure 20: P4 Simulation with Delay 50

3 RESULTS

We completed every task we were asked in the experiment. At first we solved the questions in the preliminary part by hand, but then we got to see their results clearly through simulations. Since we draw and design the expressions in Logisim, it was not hard to validate their correctness through simulations. We added required tables, images, circuits to the specific places in in preliminary part. For the experiment we supplemented our images namely elaborated design and simulations to the report for each part.

4 DISCUSSION

In first part, we implemented the circuit we designed in preliminary part using Verilog. Also, we added comments to our code in order for it to be understood by the reader. In materials and methods section, we did indicate the elaborate design schematic of modules alongside with simulations. In first part, as it was asked, we used AND,OR,NOT modules that can be used later as well and obtained the result. Likewise, in part 2 we did have the same result but with NAND gates. We also have created the function tables/truth tables of every expressions again in their corresponding parts. Some of them are also usable to check the correctness of the implementation by comparing them with simulation results. In third part, we created MUX module that can be utilized in the future. We assigned its internal structure to to assign part. Then for our question, we created another module and utilized NOT and MUX modules. In MUX module, we placed the values we were asked, simulated and validated them. In the last part, we created a separate DEC module, assigned output values. Again in another module, we implemented DEC and two-input OR modules then simulated.It was inconvenient to set as top in each time we need our modules. So, we uncommented the part we used during the experiment.

5 CONCLUSION

We successfully implemented the modules and simulated them, but apart from last week we learnt how to form a multiplexer and decoder modules. Since we knew their internal expressions, it was not hard to complete the rest of the module. We also realized the importance of forming a separate module for each question. There were several differences of the arrangements of gates and multiplexer/decoder. One of the most challenging part for us was since we have countless figures, it was hard to keep their position as we wish so we referred them by their number. All in all, we can conclude that the experiment was beneficial for us.

REFERENCES

[1] LogicLab. Logic lab. *An example journal*, 22(4):10–16, February 2020.

[2] Overleaf documentation https://tr.overleaf.com/learn.