ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E

DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	2
EXPERIMENT DATE	:	26.03.2021
LAB SESSION	:	FRIDAY - 14.00
GROUP NO	:	G14

GROUP MEMBERS:

150180112 : ÖMER MALİK KALEMBAŞI

150190014 : FEYZA ÖZEN

150190108 : EKİN TAŞYÜREK

SPRING 2021

Contents

1 INTRODUCTION [10 points]

In this experiment, we aimed to find the expression with the lowest cost for combinational logic circuits and implement them. We designed basic logic gates modules to implement more complex circuits as modules, like 8:1 Multiplexer and 3:8 Decoder. Finally, we used these complex modules to implement an equation. We tested all of our implementations by simulating on Vivado.

2 MATERIALS AND METHODS [40 points]

2.1 PART 1

In this part we used Karnaugh map to find prime implicants of equation $F_1(a,b,c,d) = U_1(0,1,3,5,8,10,13,14) + \cup_\Phi(2,7,11,12)$. Then we checked the correctness of the Karnaugh map preparing charts according to the Quine-McCluskey method. After, we drew a prime implicant chart to find the lowest cost expression of F_1. Our lowest cost expression was $F_1 = a'b' + ad' + bc'd$. Finally, we designed simplified expression on Logism, we took RTL schematics on Vivado and simulated it according various input combinations.

CD / AB	00	01	11	10
00	0 1	1 1	3 1	2 ϕ
01	4 0	5 1	7 ϕ	6 0
11	12 ϕ	13 1	15 0	14 1
10	8 1	9 0	11 ϕ	10 1

CD / AB	00	01	11	10
00	0 1	1 1	3 1	2 ϕ
01	4 0	5 1	7 ϕ	6 0
11	12 ϕ	13 1	15 0	14 1
10	8 1	9 0	11 ϕ	10 1

Figure 1: Karnaugh Diagram of F_1 (1.a)

Prime Implicants:

(0,1,2,3) $a'b'$
(0,2,8,10) $b'd'$
(1,3,5,7) $a'd$
(8,10,12,14) ad'
(2,3,10,11) $b'c$
(5,13) $bc'd$
(12,13) abc'

	a	b	c	d
0	0	0	0	0
1	0	0	0	1
ϕ2	0	0	1	0
8	1	0	0	0
3	0	0	1	1
5	0	1	0	1
10	1	0	1	0
ϕ12	1	1	0	0
ϕ7	0	1	1	1
ϕ11	1	0	1	1
13	1	1	0	1
15	1	1	1	0

	a	b	c	d	
0,1	0	0	0	-	✓
0,2	0	0	-	0	✓
0,8	-	0	0	0	✓
1,3	0	0	-	0	✓
1,5	0	-	0	1	✓
2,3	0	0	1	-	✓
2,10	-	0	1	0	✓
8,10	1	0	-	0	✓
8,12	1	-	0	0	✓
3,7	0	-	1	1	✓
3,11	-	0	1	1	✓
5,7	0	1	-	1	✓
5,13	-	1	0	1	
10,11	1	0	1	-	✓
10,14	1	-	1	0	✓
12,13	1	1	0	-	
12,14	1	1	-	0	✓

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

Figure 2: Quine-McCluskey Method Applied on F_1 (1.b)

Prime Implicants:

(0,1,2,3) $a'b'$
(0,2,8,10) $b'd'$
(1,3,5,7) $a'd$
(8,10,12,14) ad'
(2,3,10,11) $b'c$
(5,13) $bc'd$
(12,13) abc'

	0	1	3	5	8	10	13	14	Cost
a'b'	✗	✗	✗						6
b'd'	✗				✗	✗			6
a'd		✗	✗	✗					5
b'c			✗			✗			5
ad'					✗	✗		✗	5
bc'd				✗			✗		7
abc'							✗		7

	0	1	3	5	8	10	13	14	Cost
a'b'	✗	✗	✗						✓ 6
b'd'	✗				✗	✗			6
a'd		✗	✗	✗					5
b'c			✗			✗			5
ad'					✗	✗		✗	✓ 5
bc'd				✗			✗		✓ 7
abc'							✗		7

Figure 3: Prime Implicant Chart of F_1 (1.c)

Prime Implicants:

(0,1,2,3) $a'b'$
(1,3,5,7) $a'd$
(5,13) $bc'd$

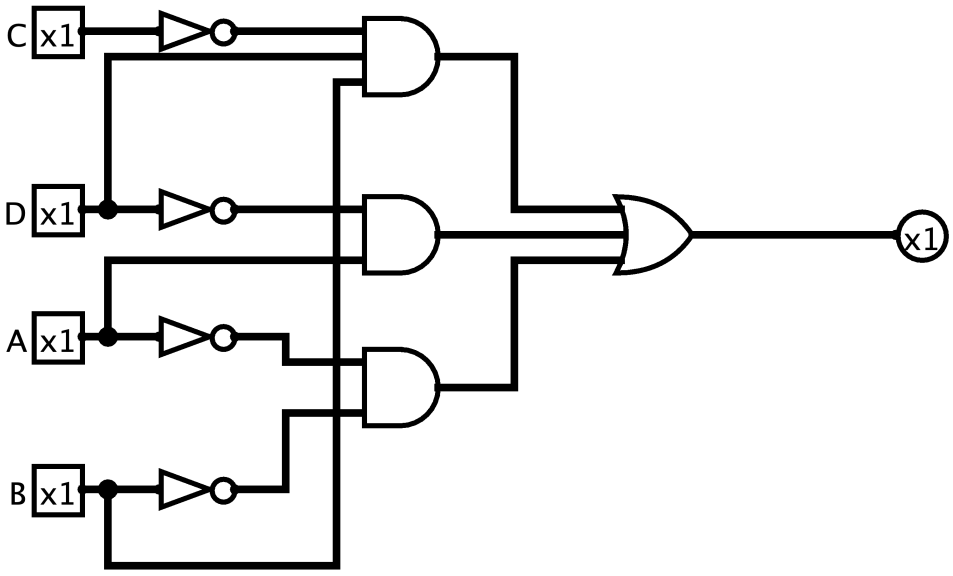

Figure 4: Circuit of F_1 (1.d)

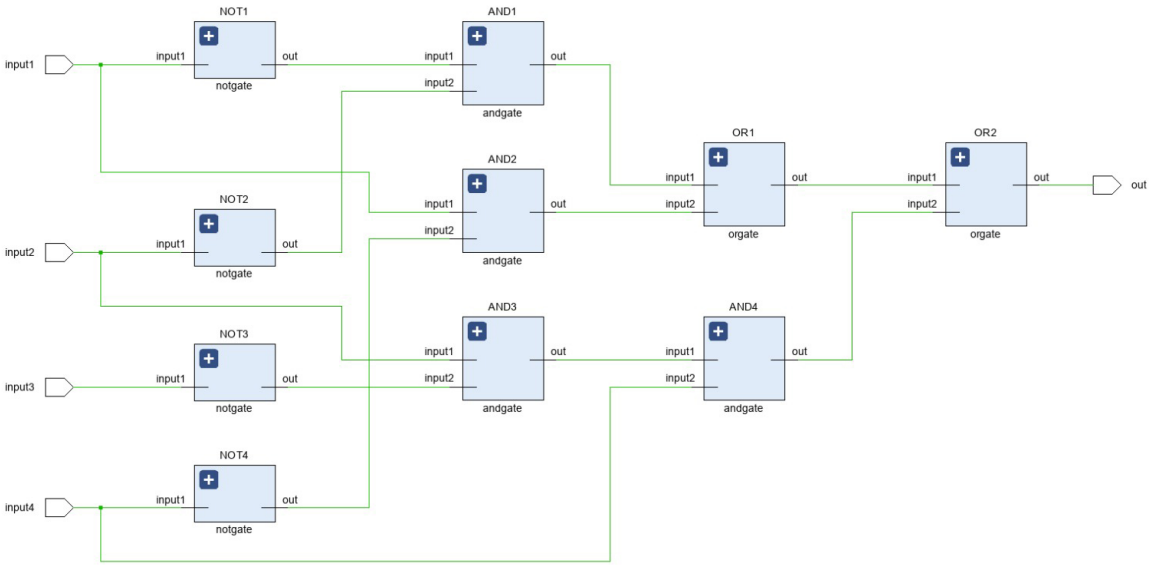

Figure 5: F_1 RTL Schematic

2.2 PART 2

In this part we wrote NAND gate module to use in this part and later in our implementations. Then we drew and designed lowest cost expression of F_1 using only NAND gates. Lowest cost expression of $F_1 = a'b' + ad' + bc'd$. Finally, we took RTL schematic of circuit F_1 with using only NAND gates.

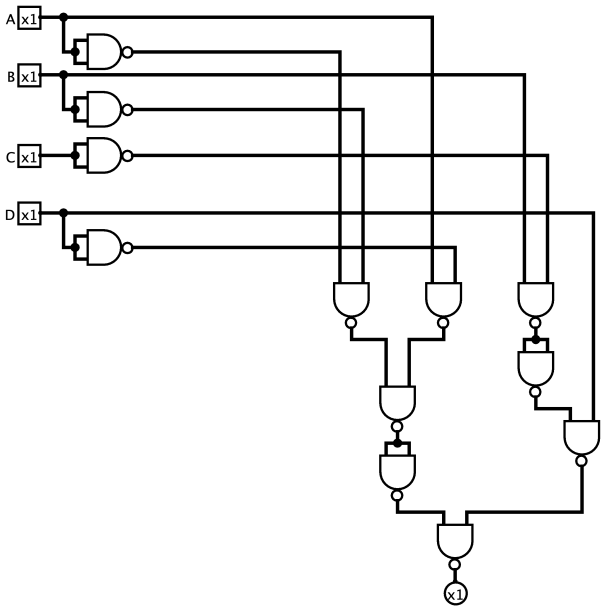

Figure 6: Circuit of F_1 (Only NAND Gates)

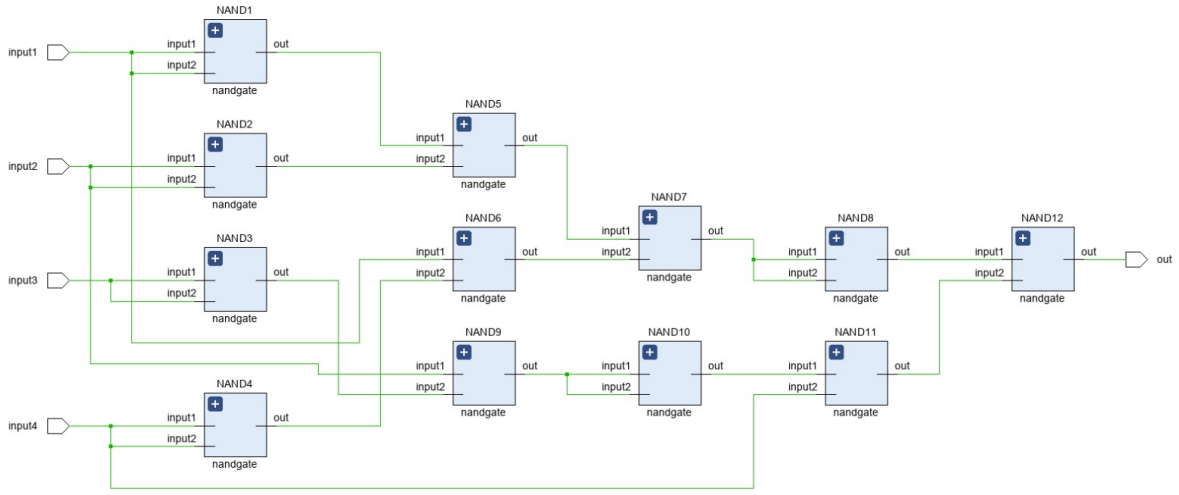

Figure 7: F_1 RTL Schematic (Only NAND Gates)

2.3 PART 3

In this part we wrote and designed 8:1 Multiplexer by using AND, OR, NOT gates modules to use it later in our implementations. Then we designed the circuit with single 8:1 multiplexer and only NOT gates according to lowest cost expression of equation $F_1(a, b, c, d) = U_1(0, 1, 3, 5, 8, 10, 13, 14) + \cup_\Phi(2, 7, 11, 12)$, which is $F_1 = a'b' + ad' + bc'd$.

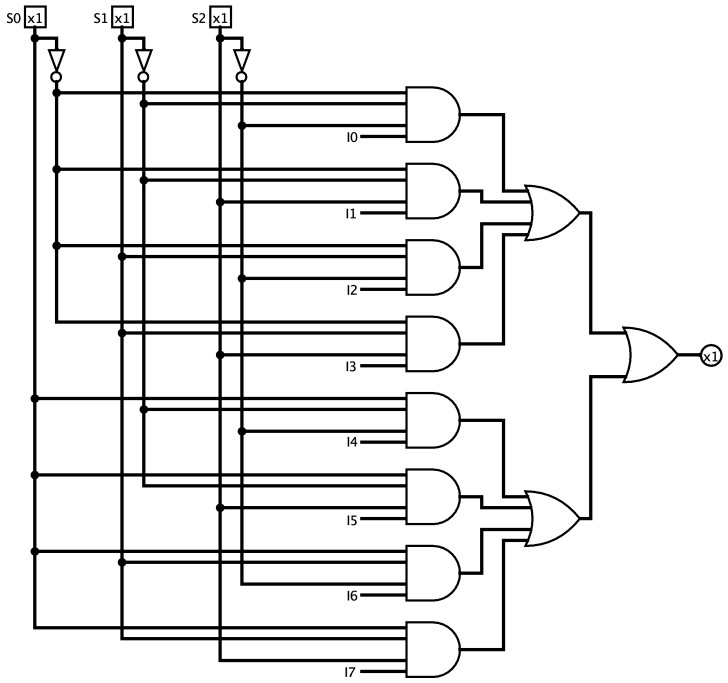

Figure 8: 8:1 Multiplexer

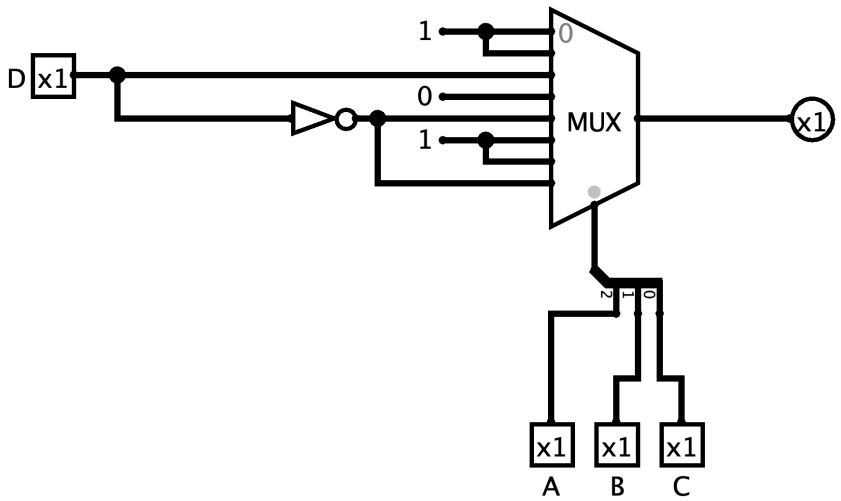

Figure 9: F_1 Circuit (8:1 Multiplexer and NOT Gates)

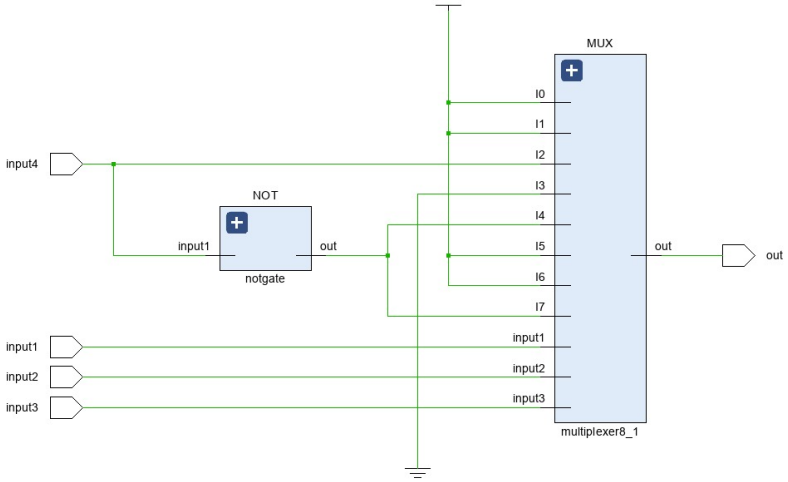

Figure 10: F_1 RTL Schematic (8:1 Multiplexer and NOT Gates)

2.4 PART 4

In this part, we designed 3:8 Decoder by using AND, OR, NOT gates and implemented it as a module. Then we designed and drew equations $F_2(a, b, c) = abc' + a'c$ and $F_3(a, b, c) = ab'c' + bc$. In this section, we had been designed expressions using single 3:8 Decoder and OR gates only. After we designed circuits in Logism, we implemented them on Vivado and took RTL schematics of them.

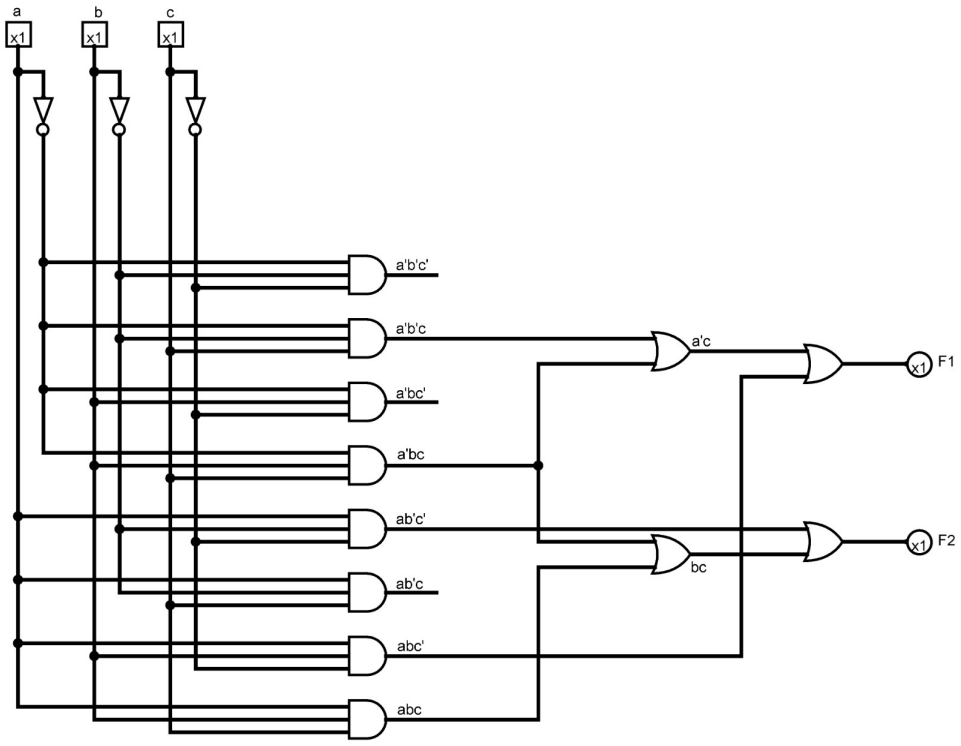

Figure 11: F_2 and F_3 Circuit (Open Circuit of 3:8 Decoder)

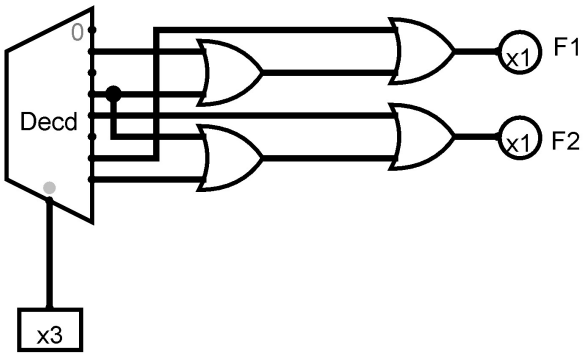

Figure 12: F_2 and F_3 Circuit (Closed Circuit of 3:8 Decoder)

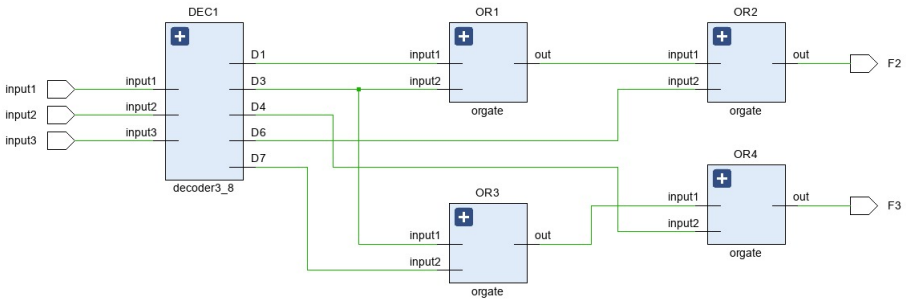

Figure 13: F_2 and F_3 RTL Schematic (3:8 Decoder and OR Gate)

3 RESULTS [15 points]

3.1 PART 1

In this part we wrote AND, OR, NOT gates as modules to use them later in our implementations. We drew a Karnaugh diagram and charts according to Quine-McCluskey method to find lowest cost expression of an equation. Then we design the lowest cost logic circuit. Finally, we showed simulation results to validate our design. As shown in Figure 13, the results are correct.

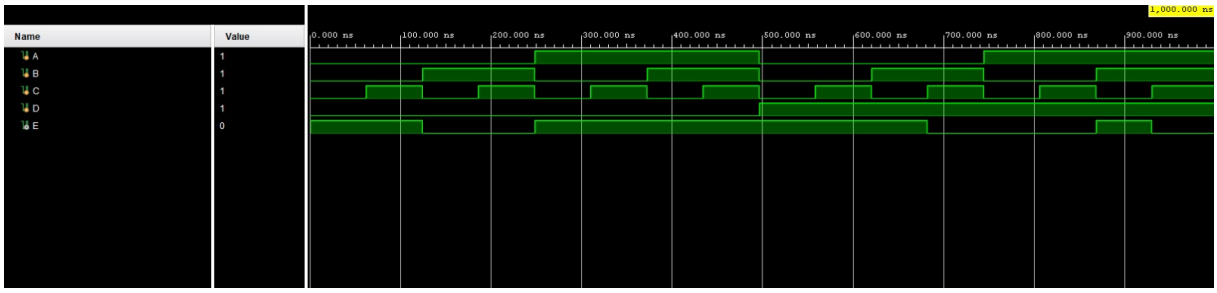

Figure 14: RTL Schematic

3.2 PART 2

In this section, we designed the lowest cost expression of the equation using only NAND gates. This section showed us that any logic circuit can be designed using only NAND gates. Finally, we ran the simulation with various input combinations to check validity.

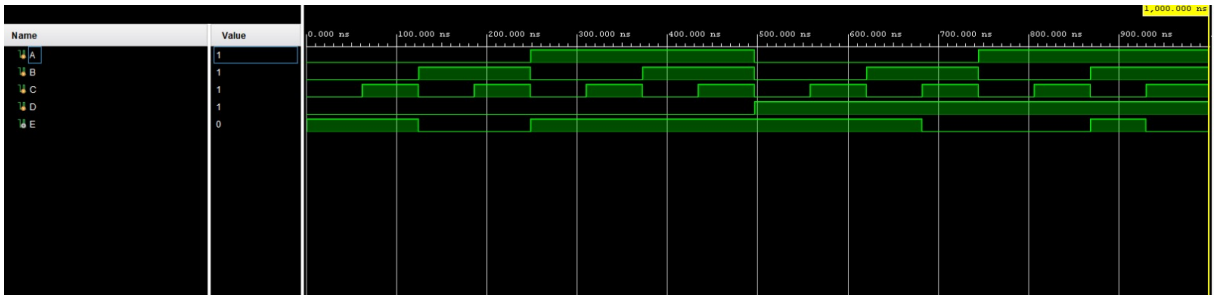

Figure 15: RTL Schematic

3.3 PART 3

In this part, we showed that 8:1 Multiplexer can be designed using AND, OR, NOT gates. With 8:1 Multiplexer, according to values of three inputs A, B, and C; Multiplexer select the output and showed it as a result. In our design, we connected inputs A, B, C to Multiplexer to select which input becomes output and we connected input D as I_2 and input D' as input I_4 and I_7 for Multiplexer. Finally, we ran the simulation and validated our design.

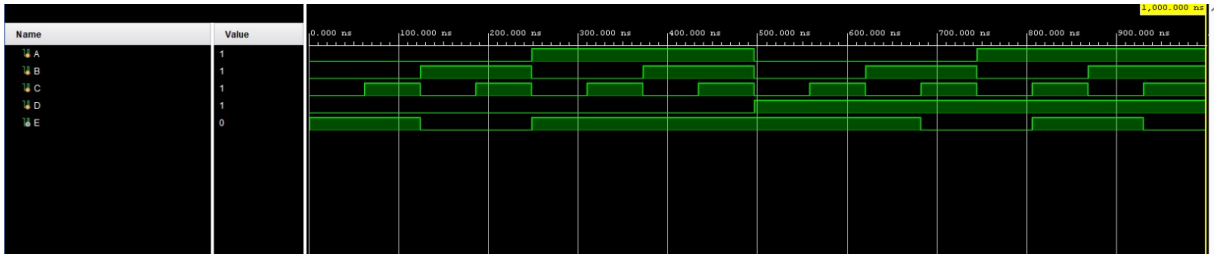

Figure 16: RTL Schematic

3.4 PART 4

In this part, we showed that 3:8 Decoder can be designed using AND, OR, NOT gates. With 3:8 Decoder, according to the values of three inputs a, b, c; we implemented 8 outputs and connected related outputs with OR gates. Finally, we ran the simulation and validated our design.

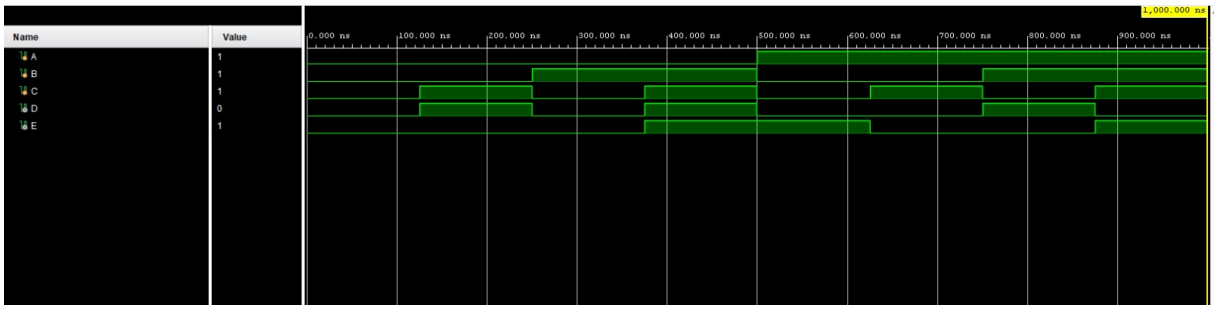

Figure 17: RTL Schematic

4 DISCUSSION [25 points]

In Part 1, we implemented AND, OR, NOT modules to use in the following parts. A, B are the input variables and C is the output variable.

$C = A\&B$ equation used for AND gate

$C = A|B$ equation used for OR gate

$C = \sim A$ equation used for NOT gate

We had an expression: $F_1(a, b, c, d) = U_1(0, 1, 3, 5, 8, 10, 13, 14) + \cup_\Phi(2, 7, 11, 12)$. Firstly, we found the prime implicants of the function using a Karnaugh map. After, we prepared tables with using Quine-McCluskey method to check the correctness of the prime implicants. Then we drew the cost table to simplify prime implicants. Our simplified and lowest cost expression was $F_1 = a'b' + ad' + bc'd$. Finally, using the gate modules we implemented in earlier, we designed the lowest cost expression as logic circuits. Our code looked like;

```
notgate NOT1(.input1(input1), .out(araKablo1));
notgate NOT2(.input1(input2), .out(araKablo2));
notgate NOT3(.input1(input3), .out(araKablo3));
notgate NOT4(.input1(input4), .out(araKablo4));
andgate AND1(.input1(araKablo1), .input2(araKablo2), .out(araKablo5));
andgate AND2(.input1(input1), .input2(araKablo4), .out(araKablo6));
```

```
andgate AND3(.input1(input2), .input2(araKablo3), .out(araKablo7));
andgate AND4(.input1(araKablo7), .input2(input4), .out(araKablo8));
orgate OR1(.input1(araKablo5), .input2(araKablo6), .out(araKablo9));
orgate OR2(.input1(araKablo9), .input2(araKablo8), .out(out));
```

After implementing the expression, we prepared test cases for it with every possible input and checked its correctness.

In Part 2, we had same expression: $F_1(a, b, c, d) = U_1(0, 1, 3, 5, 8, 10, 13, 14) + \cup_\Phi(2, 7, 11, 12)$. We designed the lowest cost logic circuit using only NAND gates. We implemented NAND modules to use in the following parts. Our code looked like;

```
nandgate NAND1(.input1(input1), .input2(input1), .out(araKablo1));
nandgate NAND2(.input1(input2), .input2(input2), .out(araKablo2));
nandgate NAND3(.input1(input3), .input2(input3), .out(araKablo3));
nandgate NAND4(.input1(input4), .input2(input4), .out(araKablo4));
nandgate NAND5(.input1(araKablo1), .input2(araKablo2), .out(araKablo5));
nandgate NAND6(.input1(input1), .input2(araKablo4), .out(araKablo6));
nandgate NAND7(.input1(araKablo5), .input2(araKablo6), .out(araKablo7));
nandgate NAND8(.input1(araKablo7), .input2(araKablo7), .out(araKablo8));
nandgate NAND9(.input1(input2), .input2(araKablo3), .out(araKablo9));
nandgate NAND10(.input1(araKablo9), .input2(araKablo9), .out(araKablo10));
nandgate NAND11(.input1(araKablo10), .input2(input4), .out(araKablo11));
nandgate NAND12(.input1(araKablo8), .input2(araKablo11), .out(out));
```

After implementing the expressions, we prepared test cases for them with every possible input and checked their correctness.

In Part 3, We used AND, OR, NOT gates modules that we implemented previously to write module for 8:1 Multiplexer. Our Multiplexer module looked like: In Multiplexer module, we defined input1, input2 and input3 as our basic inputs to identify which input $(I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7)$ becomes output. Then we designed logic circuit of our simplified equation $F_1 = a'b' + ad' + bc'd$, using 8:1 Multiplexer and NOT gates. Here we took input a, b, c as multiplexers identifier inputs and took input d as I_2. Some of results was changed according to input d, so we connected them to multiplexer as variable d (I_2) or d complement" (I_4, I_7). We connected other inputs binary 0 I_3 or $1(I_0, I_1, I_5, I_6)$. Our code looked like:

```
notgate NOT(.input1(input4), .out(araKablo1));
```

```
multiplexer8_1 MUX(.input1(input1), .input2(input2), .input3(input3),
.I0(1), .I1(1), .I2(input4), .I3(0), .I4(araKablo1), .I5(1), .I6(1),
.I7(araKablo1),  .out(out));
```

Finally, we prepared test cases for Part 3 with every possible input and checked their correctness.

In Part 4, we had two expression;

$$F_2(a, b, c) = abc' + a'c$$

$$F_3(a, b, c) = ab'c' + bc$$

In this section, we designed and wrote a module for 3:8 Decoder using AND, OR, NOT gates modules and we implemented Decoder for use later in our works. We defined three inputs (input1, input2, input3) and 8 outputs ($D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7$) for our 3:8 Decoder. Here we connected a, b, c as inputs to the Decoder. According to the Decoder's structure we took abc'as output D_6, $a'b'c$ as output D_1, $a'bc$ as output D_3. First, we connected D_1 and D_3 with or gate, after connected this to D_6. Thus, we had expression F_2:
$abc' + a'b'c + a'bc$ (original expression)
$abc' + a'c(b'b)$ (disributivity)
$abc' + a'c$ (inverse)(identity)
$= F_2(a, b, c) = abc' + a'c$

Same as previous lines, we took $ab'c'$ as output D_4, $a'bc$ as output $D3$, abc as output D_7. First, we connected D_3 and D_7 with or gate, after connected this to D_4. Thus, we had expression F_3:
$ab'c' + a'bc + abc$ (original expression)
$ab'c' + bc(a'a)$ (distributivity)
$ab'c' + bc$ (inverse)(identity)
$= F3(a, b, c) = ab'c' + bc$

Our code looked like:

```
decoder3_8 DEC1(.input1(input1), .input2(input2), .input3(input3),
.D0(araKablo1), .D1(araKablo2), .D2(araKablo3), .D3(araKablo4), .D4(araKablo5),
.D5(araKablo6), .D6(araKablo7), .D7(araKablo8));
orgate OR1(.input1(araKablo2), .input2(araKablo4), .out(araKablo9));
orgate OR2(.input1(araKablo9), .input2(araKablo7), .out(F2));
orgate OR3(.input1(araKablo4), .input2(araKablo8), .out(araKablo10));
orgate OR4(.input1(araKablo10), .input2(araKablo5), .out(F3));
```

Finally, we prepared test cases for part 4 with every possible input and checked their correctness of F2 and F3.

5 CONCLUSION [10 points]

We used multiplexer and decoder in this homework and strengthened our knowledge about them. It was so difficult and complicated to code a multiplexer using 2-input gates in Verilog. Then we learned that we can use 2 or more input gates. In conclusion, we used AND, OR, NOT, NAND gates and complicated circuits like multiplexer and decoder. Then used the simulation tool and checked the correctness of our results.