ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E

DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	1
EXPERIMENT DATE	:	19.03.2021
LAB SESSION	:	FRIDAY - 10.30
GROUP NO	:	G17

GROUP MEMBERS:

150180024 : ŞULE BEYZA KARADAĞ

150190710 : SENİHA SERRA BOZKURT

150190024 : AHMET FURKAN KAVRAZ

SPRING 2021

Contents

1 INTRODUCTION [10 points]

We recalled the axioms and theorems of Boolean Algebra and validated these axioms and theorems using the Verilog.

2 MATERIALS AND METHODS [40 points]

2.1 Preliminary

In the preliminary part, we proved the given equations, functions and their duals by using the axioms of Boolean Algebra. Next to each step, we wrote the name of the axiom we applied in the previous step.

Proof of normal expressions:

F(a, b) = a + ab = a

a . 1 + a . b = a	Identity
a . (1 + b) = a	Distributivity
a . (1) = a	Dominance
a = a	Identity

F(a, b) = (a + b) . (a + b') = a

a + (b . b') = a	Distributivity
a + (0) = a	Inverse
a = a	Identity

Proof of dual expressions:

F(a, b) = a . (a + b) = a

(a + 0) . (a + b) = a	Identity
a + (b . 0) = a	Distributivity
a + 0 = a	Dominance
a = a	Identity

F(a,b) = (a . b) + (a . b') = a

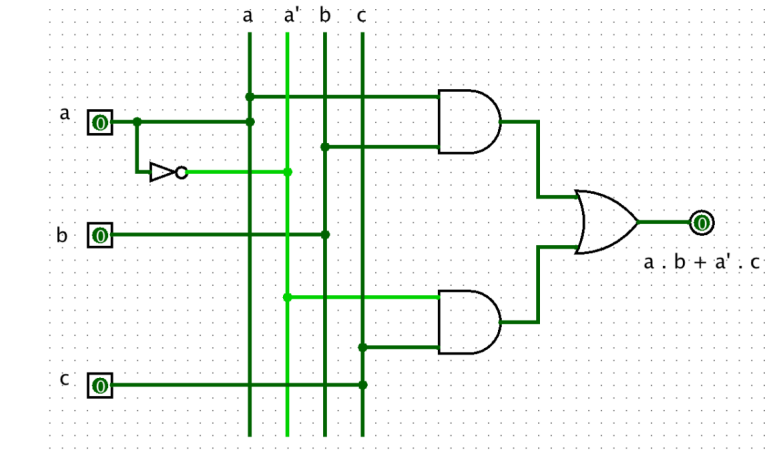
a + (b . b') = a	Distributivity
a + (0) = a	Inverse
a = a	Identity

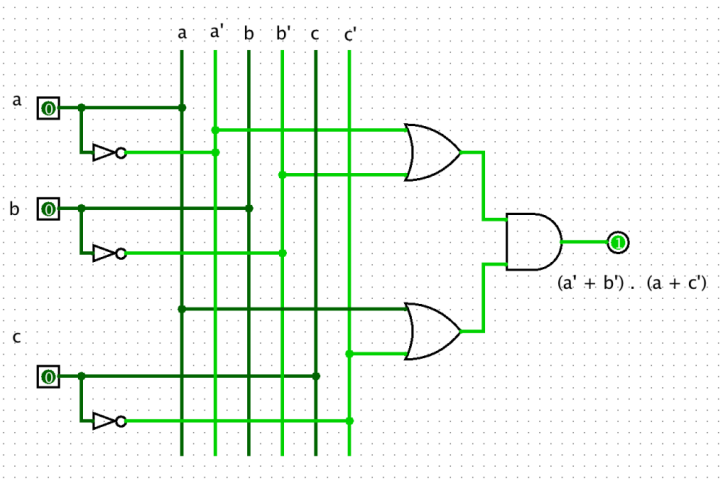
Calculation of complementary expression (F') for the function F by using De Morgan theorem:

F = a . b + a' . c

F' = (a . b + a' . c)'	DeMorgan's Law
= (a . b)' . (a' . c)'	DeMorgan's Law
= (a' + b') . (a + c')	DeMorgan's Law

And its circuits are:

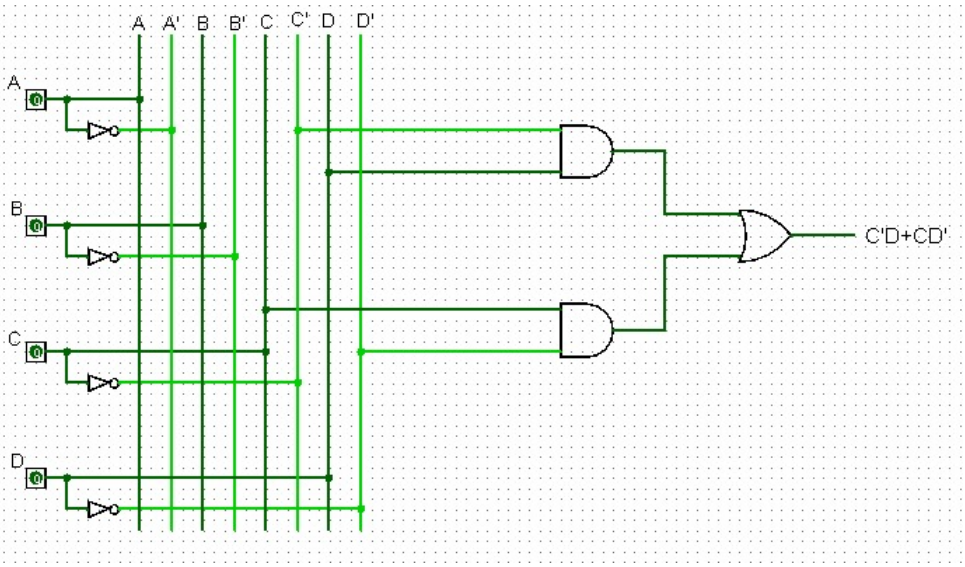




Karnaugh Map:

We simplified the given logical function by using Karnaugh Map and obtained the expression of the function. Then, we draw the logic circuit according to obtained expression = C'.D + C.D'

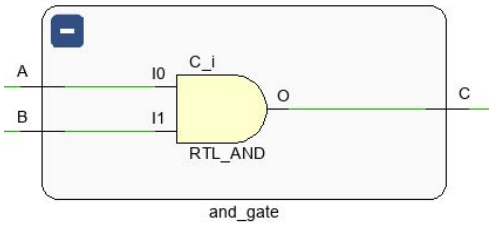
CD / AB	00	01	11	10
00	0 (0)	1 (1)	0 (3)	1 (2)
01	0 (4)	1 (5)	0 (7)	1 (6)
11	0 (12)	1 (13)	0 (15)	1 (14)
10	0 (8)	1 (9)	0 (11)	1 (10)



2.2 Part 1

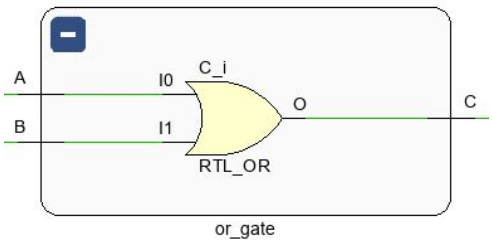
In this part, the requested to implement AND, OR, NOT modules which we will use in the following experiments are implemented. Names of the modules are same with the corresponding gates.

```
module and_gate(
    input A,
    input B,
    output C
    );
    assign C = A & B;
endmodule
```



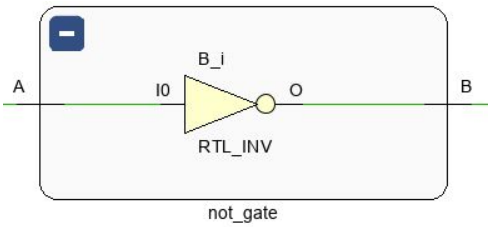
AND gate is created with assigning result of applying "AND" operation to inputs to output.

```
module or_gate(
    input A,
    input B,
    output C
    );
    assign C = A | B;
endmodule
```



OR gate is created with assigning result of applying "OR" operation to inputs to output.

```
module not_gate(
    input A,
    output B
    );
    assign B = ~A;
endmodule
```



NOT gate is created with assigning result of "NOT" operation of input to output.

2.3 Part 2

In this part, we designed and implemented the logic circuits for the given expressions below by using **AND**, **OR**, **NOT** modules which we designed in the first part. Then, we simulated it for each different combination of input and validated the correctness of our implementation.

```
module F1(
    input wire a,
    input wire b,
    output wire c
    );
    wire var1;
    and_gate and1(a, b, var1);
    or_gate or1(a, var1, c);
endmodule

module F2(
    input wire a,
    input wire b,
    output wire c
    );
    wire var1, var2, var3;
    or_gate or1(a, b, var1);

    not_gate not1(b,var2);
    or_gate or2(a, var2, var3);

    and_gate and1(var1, var3, c);
endmodule
```

	a	b	F1
0	0	0	0
1	0	1	0
2	1	0	1
3	1	1	1

	a	b	F2
0	0	0	0
1	0	1	0
2	1	0	1
3	1	1	1

We implement the design of F1 and F2 with the help of "And", "Or" and "Not" Gates.

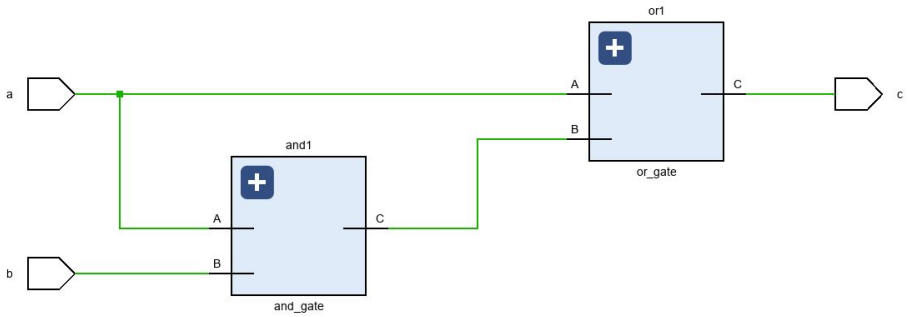
In order to obtain F_1:

1. We connected a and b to AND gate to obtain var1.
2. We connected a and var1 to OR gate to obtain c.

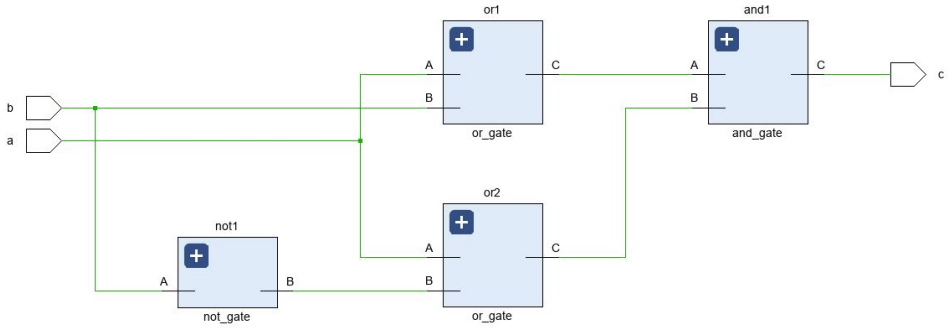
In order to obtain F_2:

1. We connected a and b to OR gate to obtain var1.
2. We connected b to NOT gate to obtain var2.

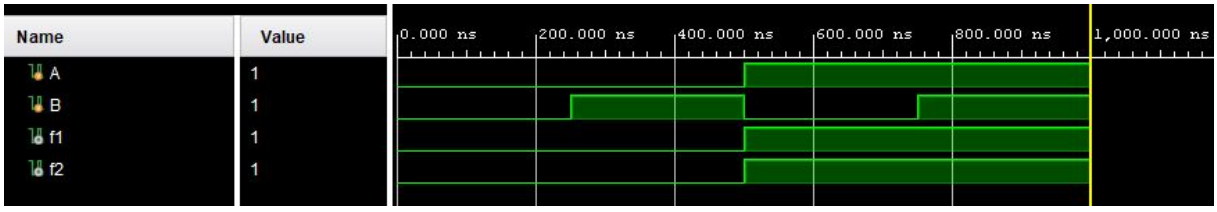
3. We connected a and var2 to OR gate to obtain var3.
4. We connected var1 and var3 to AND gate to obtain c.



Elaborate design of F1 function.



Elaborate design of F2 function.



Simulation results of F1 and F2 functions.

2.4 Part 3

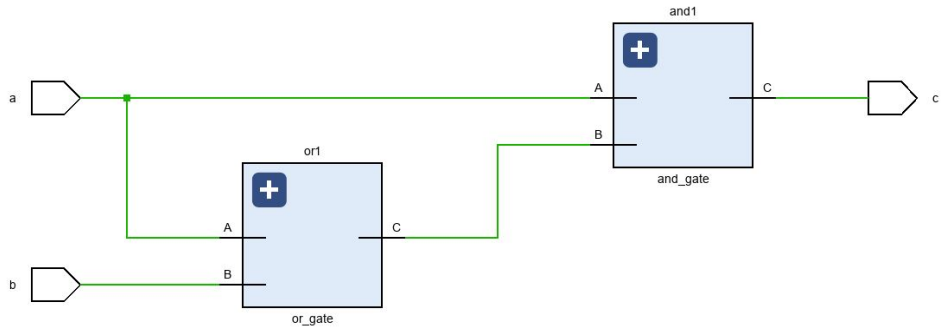
The theorem is given as: a + a . b = a First, we determined the dual of the given theorem and then, implemented the functions for both sides of the dual theorem using AND, OR, NOT modules which we designed in the first part. We validated the truth of the theorem by comparing the changes in the outputs using simulation. Dual of F1: a . (a + b) = a

```
module dual_of_F1(
    input wire a,
    input wire b,
    output wire c
    );
    wire var1;
    or_gate or1(a, b, var1);
    and_gate and1(a, var1, c);
endmodule
```

	a	b	F1-dual
0	0	0	0
1	0	1	0
2	1	0	1
3	1	1	1

In order to get dual of F_1:

1. We connected a and b to OR gate to obtain (a + b), which is also var1
2. We connected a and var1 to AND gate to obtain a . (a + b), which is also c.



Elaborate design of Dual of F1 function.

Simulation results of Dual of F1 function.

2.5 Part 4

F_3(a, b, c) = a . b + a' . c is given. Firstly, we determined the complement of the given function (F3). Then, we implemented the circuit which realizes the complementary function (F3'). We validated our implementation by using the truth table and simulated it for each different combination of input.

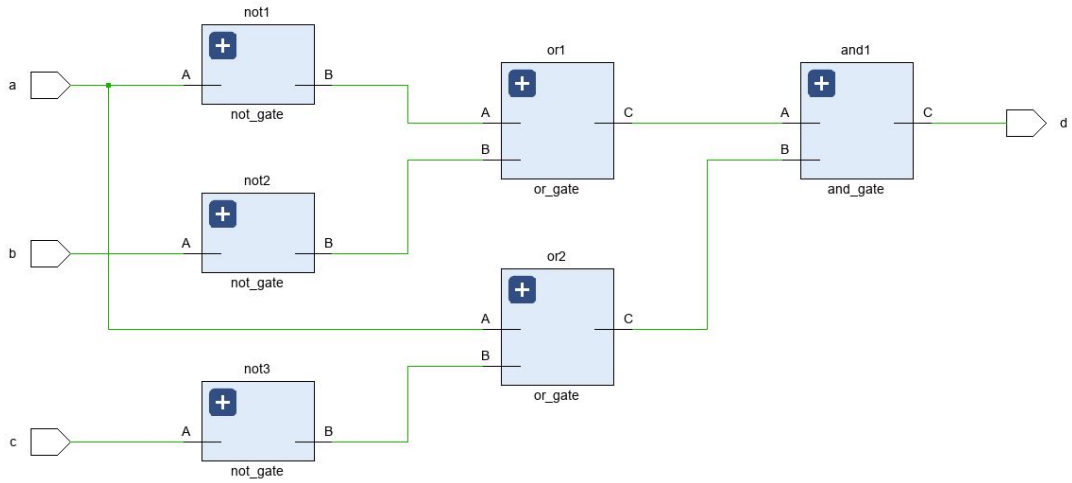
```
module comp_F3(
    input wire a,
    input wire b,
    input wire c,
    output wire d
    );
    wire not_a, not_b, not_c, var1, var2;
    not_gate not1(a, not_a);
    not_gate not2(b,not_b);
    not_gate not3(c, not_c);

    or_gate or1(not_a, not_b, var1);
    or_gate or2(a, not_c, var2);

    and_gate and1(var1, var2, d);
endmodule
```

	a	b	c	F3
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

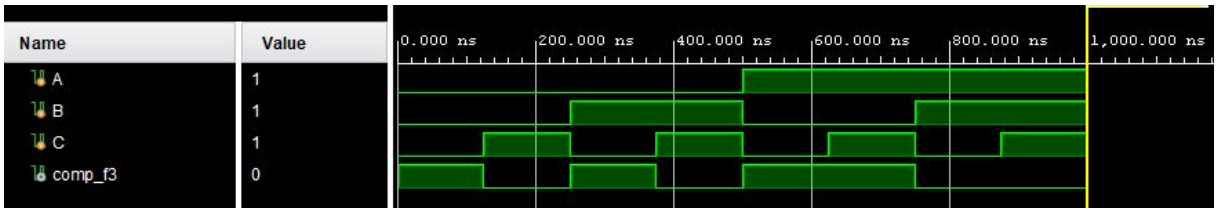
	a	b	c	F3'
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0

In order to get complement of F_3:

1. We connected a to NOT gate to obtain a'.
2. We connected b to NOT gate to obtain b'.
3. We connected c to NOT gate to obtain c'.
4. We connected a' and b' to OR gate to obtain (a'+ b'), which is = var1.
5. We connected a and c' to OR gate to obtain (a + c'), which is = var2.
6. Finally, we connected var1 (a'+ b') and var2(a + c') to AND gate to obtain (a' + b') . (a + c'), which is the complement of F_3.



Elaborate design of Complement of F3 function.



Simulation results of Complement of F3 function.

2.6 Part 5

A basic logical function (F4) is defined as follows.

F4(a, b, c, d) = U_1(1, 2, 5, 6, 9, 10, 13, 14)

First, we simplified given logical function and implemented the simplified expression using AND, OR, NOT modules which we designed in the first part.

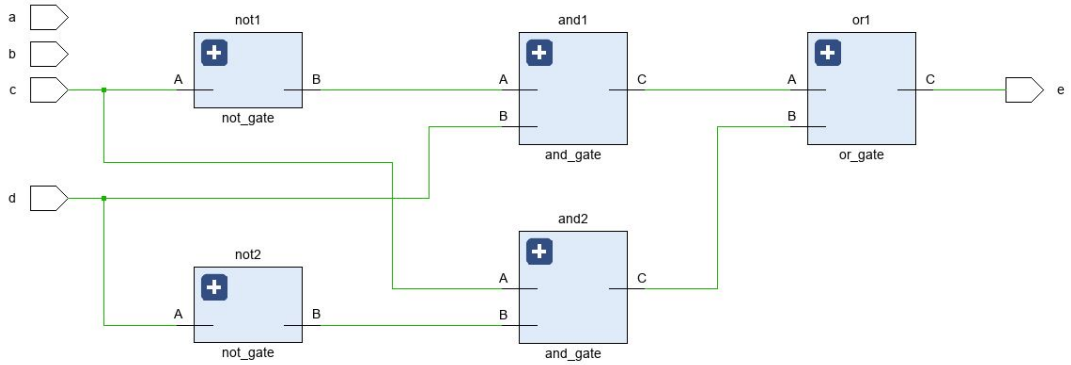
The logical function is **c'd + cd'**. We validated our implementation by observing the outputs for each possible input.

```
module F4(
    input wire a,
    input wire b,
    input wire c,
    input wire d,
    output wire e
    );
    wire not_c, not_d, var1, var2;
    not_gate not1(c, not_c);
    not_gate not2(d,not_d);

    and_gate and1(not_c, d, var1);
    and_gate and2(c, not_d, var2);

    or_gate or1(var1, var2, e);
endmodule
```

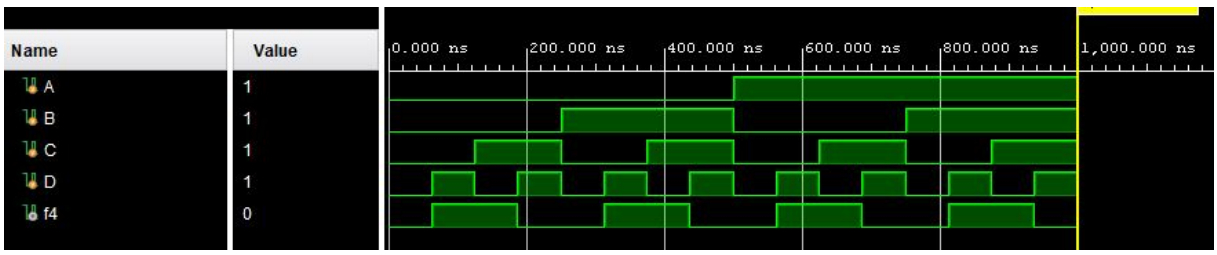
	a	b	c	d	F4
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	0

In order to get F_4:

1. We connected c to NOT gate to obtain not_c.
2. We connected d to NOT gate to obtain not_d.
3. We connected not_c and d to AND gate to obtain (c' . d), which is also var1.
4. We connected c and not_d to AND gate to obtain (c . d'), which is also var2.
5. We connected var1 and var2 to OR gate to obtain (c' . d) + c . d'), which is also e.



Elaborate design of F4 function.



Simulation results of F4 function.

3 RESULTS [15 points]

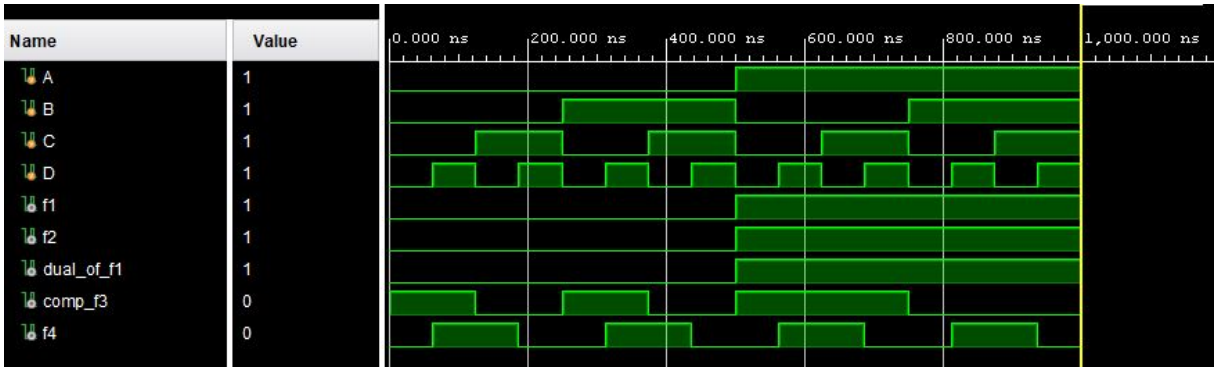
The simulation results we obtained after completing each step of the experiments resulted the same with their truth tables drawn above. This concludes the fact that our experiment ended up being consistent.

```
module test();

    reg A;
    reg B;
    reg C;
    reg D;
    wire f1;
    wire f2;
    wire dual_of_f1;
    wire comp_f3;
    wire f4;

    F1          uut1  (A,B,f1);
    F2          uut2  (A,B,f2);
    dual_of_F1  uut3  (A,B,dual_of_f1);
    comp_F3     uut4  (A,B,C,comp_f3);
    F4          uut5  (A,B,C,D,f4);


    initial begin
        A = 0;   B = 0; C = 0;  D = 0;  #62.5;
        A = 0;   B = 0; C = 0;  D = 1;  #62.5;
        A = 0;   B = 0; C = 1;  D = 0;  #62.5;
        A = 0;   B = 0; C = 1;  D = 1;  #62.5;
        A = 0;   B = 1; C = 0;  D = 0;  #62.5;
        A = 0;   B = 1; C = 0;  D = 1;  #62.5;
        A = 0;   B = 1; C = 1;  D = 0;  #62.5;
        A = 0;   B = 1; C = 1;  D = 1;  #62.5;
        A = 1;   B = 0; C = 0;  D = 0;  #62.5;
        A = 1;   B = 0; C = 0;  D = 1;  #62.5;
        A = 1;   B = 0; C = 1;  D = 0;  #62.5;
        A = 1;   B = 0; C = 1;  D = 1;  #62.5;
        A = 1;   B = 1; C = 0;  D = 0;  #62.5;
        A = 1;   B = 1; C = 0;  D = 1;  #62.5;
        A = 1;   B = 1; C = 1;  D = 0;  #62.5;
        A = 1;   B = 1; C = 1;  D = 1;  #62.5;
    end
endmodule
```

You can see the simulation inputs and their given times to complete each step of their part of the simulation next to this column. To carry out each step of the experiment, we commented unused inputs and used the rest of the inputs, and arranged the time slots of each, accordingly.

- First, for **Part-1**, we used regs A and B to implement AND, OR and NOT gates.
- After that, in **Part-2** we used the gates we formed in Part-1 and formed the functions F_1 and F_2 using regs A and B.
- Then, in **Part-3**, we formed the dual of F_1 function.
- In **Part-4**, we used the regs A, B and C and formed the function F_3'
- Finally, in **Part-5**, we used ALL the regs and simulated the function. We prooved the statement that the function is equal to = C'D + CD'.

The final result is the formed XOR gate between C and D inputs.



4 DISCUSSION [25 points]

In the preliminary part, we proved the given equations (a + a.b = a and also (a + b). (a + b') = a) and their duals by using the axioms of Boolean Algebra.
After that, we calculated the complement of the equation a.b + a'c with the help of De-Morgan's theorem and we draw the corresponding circuits using Logisim.
Later, we used the given function in minterm form, simplified it and drew the circuit.

Also, we find by using Karnaugh Map the least number of gates required to form the final experiment, in part-5, which is C XOR D. So this is the circuit with lowest cost.

During the experiment, we first formed the gates and used the corresponding inputs to simulate all of the parts.
We designed all the elaborated form of given functions in each part and then simulated the implementations. Then, comparing the results with drawn truth tables, we concluded that our results are coherent.

While trying to simulate all the modules, one after each other, we realized that more inputs are necessary, so 2^{input} combinations are required to form what we have done with the truth tables. So we increased the number of inputs and also input combinations. Then we realized that the spared time slot is not enough for 2^{input} combinations, so we decreased the time spared to each binary input combination.

5 CONCLUSION [10 points]

We learned how to use Vivado and write modules in Verilog. The language was like the interpretation of the circuit models we drew back in the previous term in Digital Circuits (BLG 231E) course using Logisim, so as the circuit models deriven from the vivado were similiar to those. So we included both the circuits obtained from vivado and from logisim, to compare the results, which you can see above. Learned to write in LaTex, and all the pdf formatting.
The difficulties we faced are learning the programs, languages and group work.