ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E
DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	1
EXPERIMENT DATE	:	31.03.2023
LAB SESSION	:	FRIDAY - 10.30
GROUP NO	:	G8

GROUP MEMBERS:

150200919	:	Abdullah Jafar Mansour Shamout
150220762	:	Muhammed Yusuf Mermer

SPRING 2023

Contents

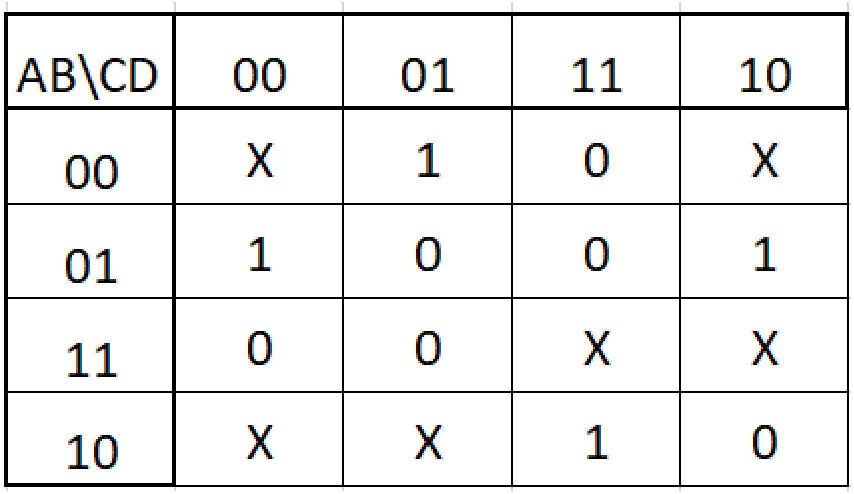
1 INTRODUCTION [10 points]

In this experiment we learned how to find the expression with the lowest cost using combinational logic circuits and implemented them. We found them using Karnaugh map and Quine-McClusky's methods then using and implicant chart we found the lowest cost implementation and implemented them. We implemented the expression in two ways. Using AND, NOT, OR gates, and using an 8-1 mux with NOT gates.

2 MATERIALS AND METHODS [40 points]

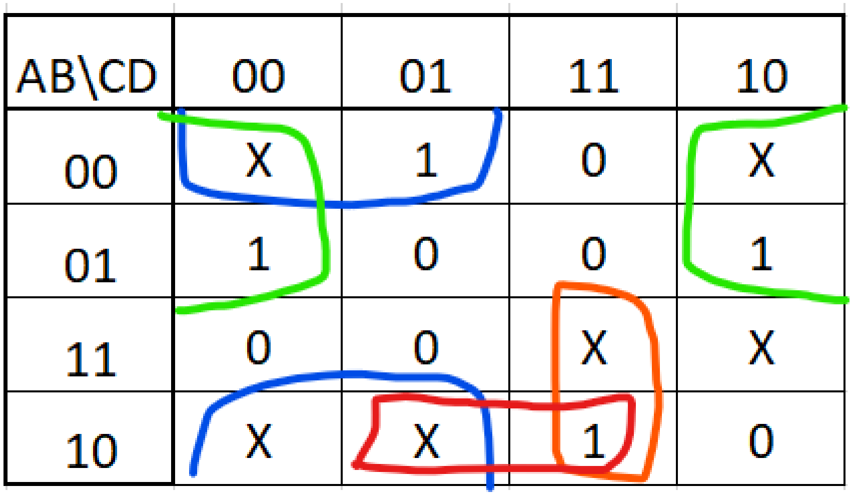
2.1 Preliminary 1.a

To find with K-Map, at first we find prime implicant for implicants.
Prime implicant is biggest implicants for that group of 1's.



AB\CD	00	01	11	10
00	X	1	0	X
01	1	0	0	1
11	0	0	X	X
10	X	X	1	0

Figure 1: K-Map

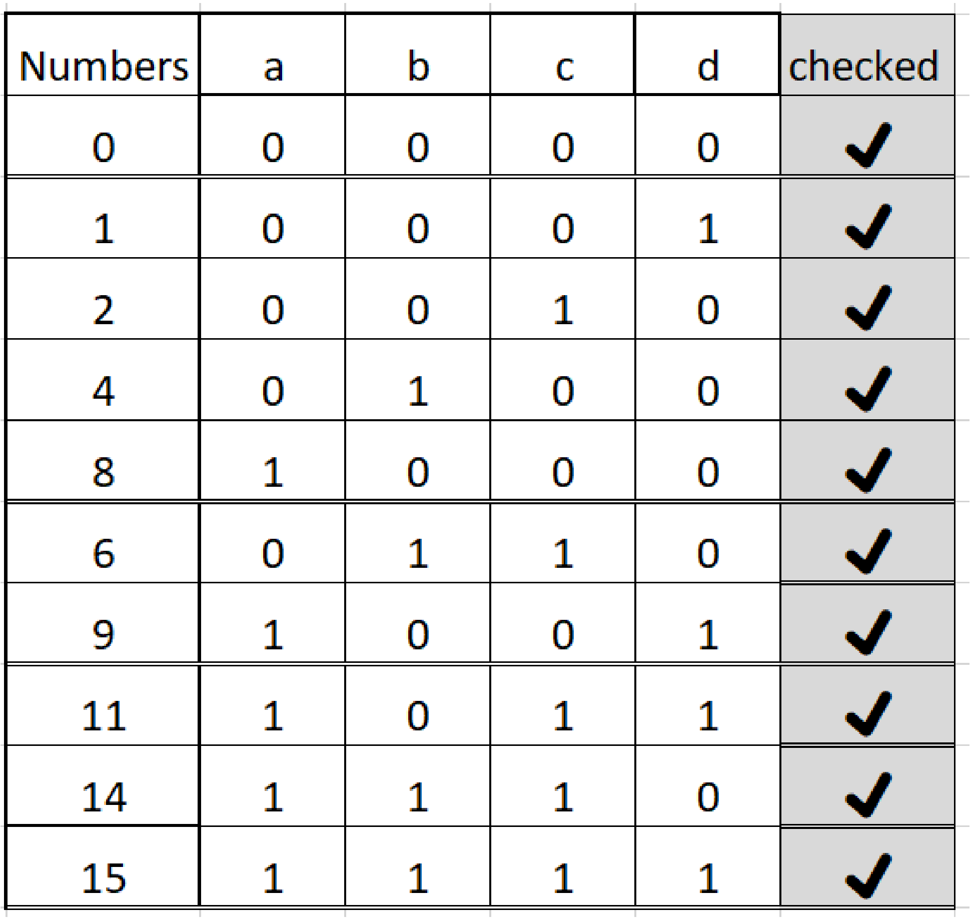


AB\CD	00	01	11	10
00	X	1	0	X
01	1	0	0	1
11	0	0	X	X
10	X	X	1	0

Figure 2: K-Map With All Prime Implicants

2.2 Preliminary 1.b

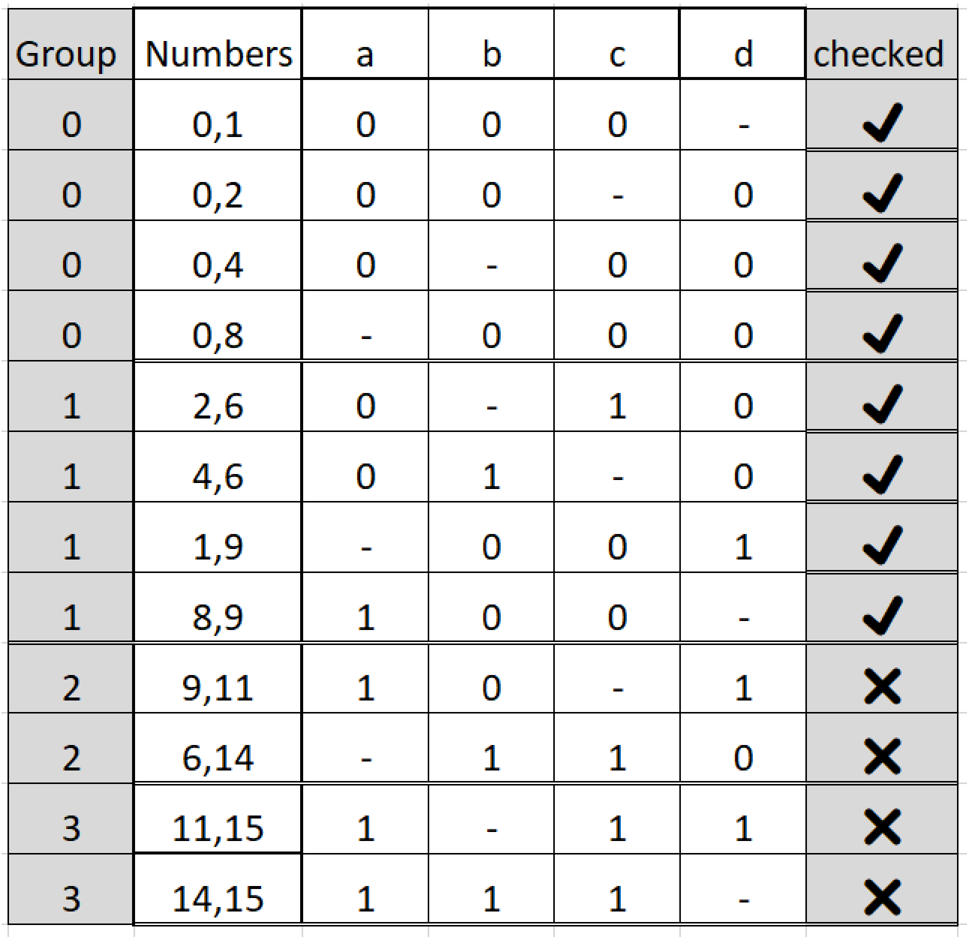
To find with Quine Mccluskey method, at first we need to write all implicants(including don't care condition) to table (like binary digits) as in the figure:



Numbers	a	b	c	d	checked
0	0	0	0	0	✓
1	0	0	0	1	✓
2	0	0	1	0	✓
4	0	1	0	0	✓
8	1	0	0	0	✓
6	0	1	1	0	✓
9	1	0	0	1	✓
11	1	0	1	1	✓
14	1	1	1	0	✓
15	1	1	1	1	✓

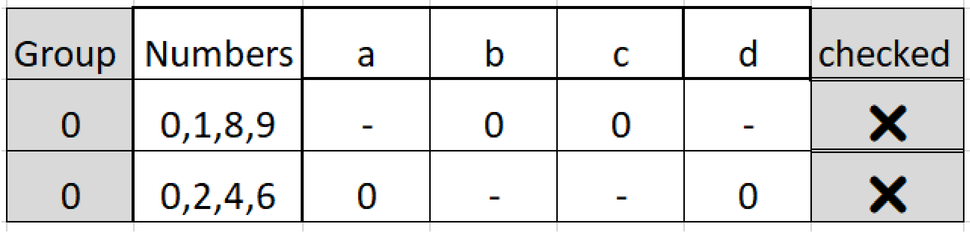
Figure 3: input combinations

After this operation, we made pairs from different groups which have only one digit difference. And we checked previous rows if they are used in table:



Group	Numbers	a	b	c	d	checked
0	0,1	0	0	0	-	✓
0	0,2	0	0	-	0	✓
0	0,4	0	-	0	0	✓
0	0,8	-	0	0	0	✓
1	2,6	0	-	1	0	✓
1	4,6	0	1	-	0	✓
1	1,9	-	0	0	1	✓
1	8,9	1	0	0	-	✓
2	9,11	1	0	-	1	✗
2	6,14	-	1	1	0	✗
3	11,15	1	-	1	1	✗
3	14,15	1	1	1	-	✗

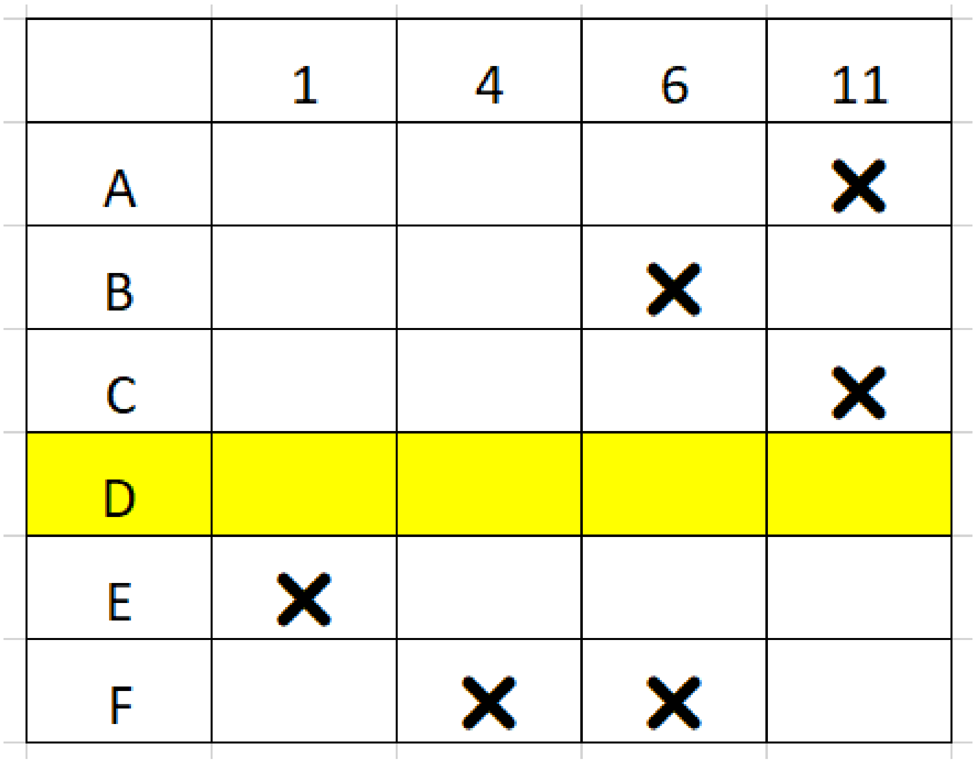
Figure 4: groups



Group	Numbers	a	b	c	d	checked
0	0,1,8,9	-	0	0	-	✗
0	0,2,4,6	0	-	-	0	✗

Figure 5: groups

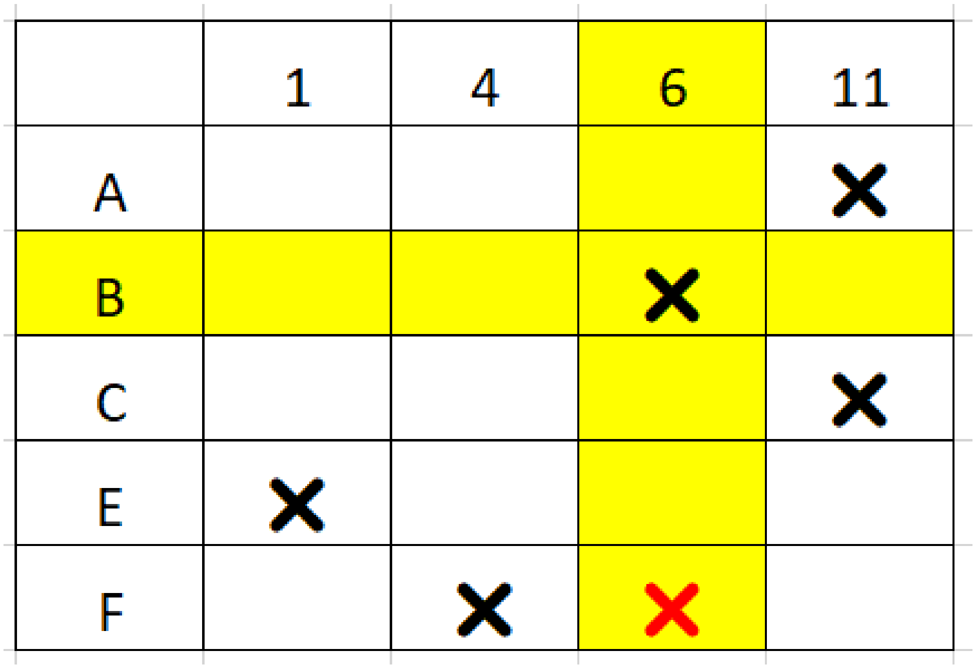
after that to find the cost we made a chart with the left groups and compared their coverage of the 1's. Here we see that D doesnt cover any so we delete it



	1	4	6	11
A				✗
B			✗	
C				✗
D				
E	✗			
F		✗	✗	

Figure 6: cost1

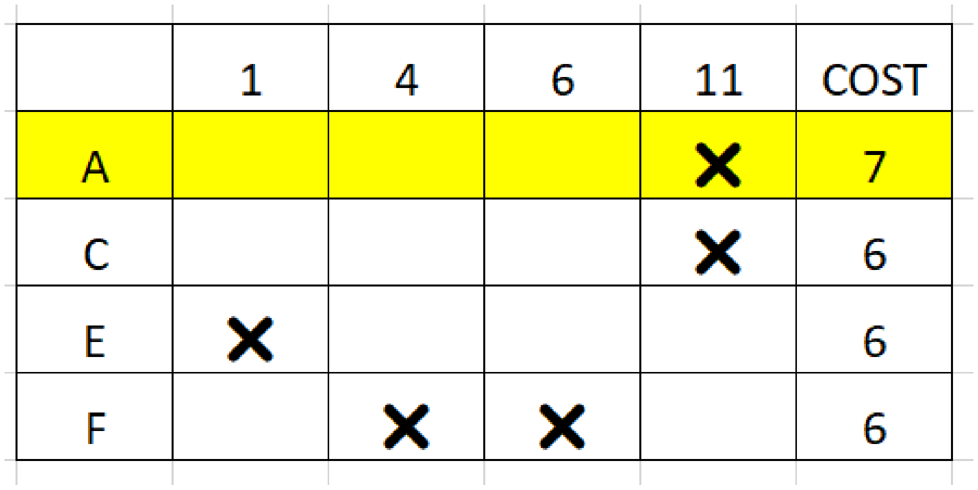
Here we see that F is the only one that covers 4 and it covers 6 too so B is redundant so we can delete it.



	1	4	6	11
A				✗
B			✗	
C				✗
E	✗			
F		✗	✗	

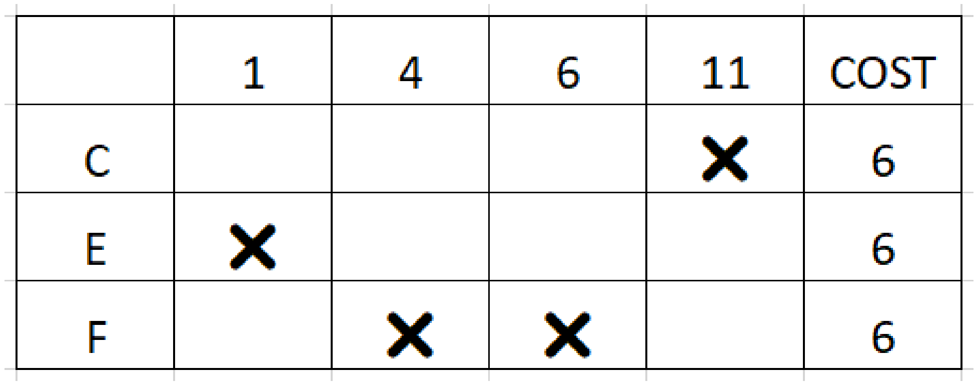
Figure 7: cost2

If we compare costs now we see 11 is covered by C for a cheaper cost than A so we delete A and we are left with C, E, F.



	1	4	6	11	COST
A				X	7
C				X	6
E	X				6
F		X	X		6

Figure 8: cost3



	1	4	6	11	COST
C				X	6
E	X				6
F		X	X		6

Figure 9: cost4

Thus the lowest cost is 18 which has the expression: ACD + B'C' + A'D' we can see the circuitry of this part in the results section

2.3 Preliminary 2

In this part of the question we need to implement given expression using 8 to multiplexer and not gates. We already got the simplest and most cost effective equations via finding essential prime implicants (You can see how we got the equation from Preliminary 1).

To make a multiplexer, at first we need to determine selectors. For this question, we connected Selector input 2 to input A, Selector input 1 to input B, Selector input 0 to input C.

At every combinations of this A, B and C inputs, multiplexer's only one input will be reflected to the output. When we write combinations of A, B, C into our simplified equation, we see that for every combinations, we got equation in terms of 0,1 or D value.

Therefore, one can understand that we can connect inputs directly to 0, 1, d or d' to the multiplexer according to the values of a,b,c. Here is the table for all a,b,c combinations and the the value we get from the equation.

a	b	c	Value
0	0	0	1
0	0	1	d'
0	1	0	d'
0	1	1	d'
1	0	0	1
1	0	1	d
1	1	0	0
1	1	1	d

Table 1: Truth Table for a,b,c Combinations

Then we can connect this values to multiplexer as:

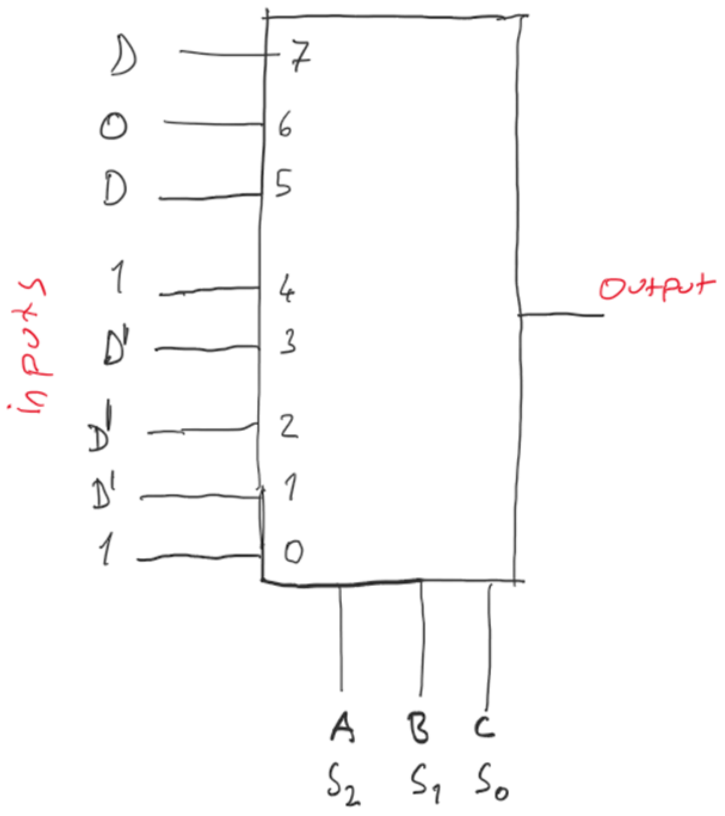


Figure 10: multiplexer design

3 RESULTS [15 points]

3.1 Experiment part 1

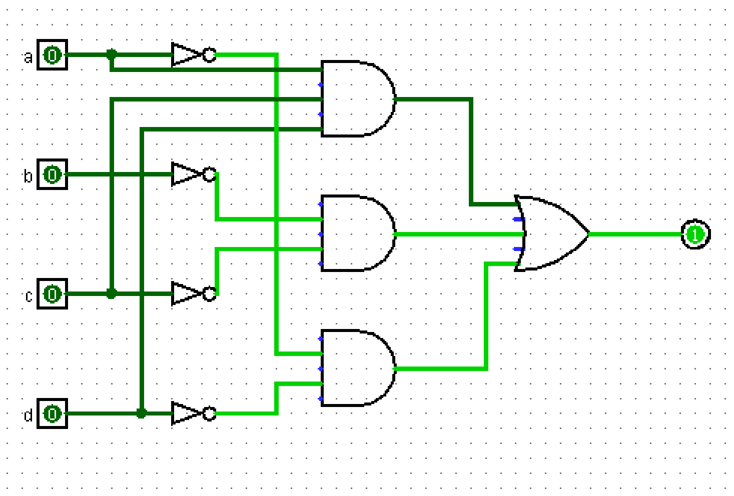


Figure 11: circuit

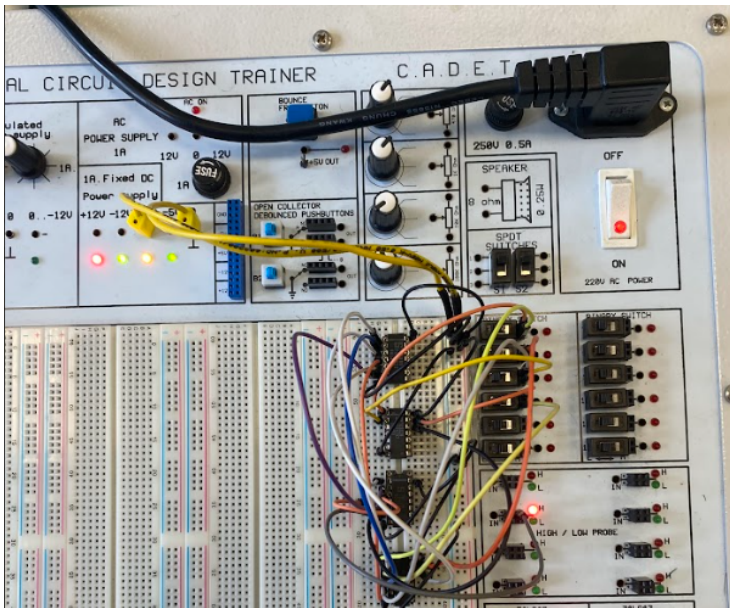
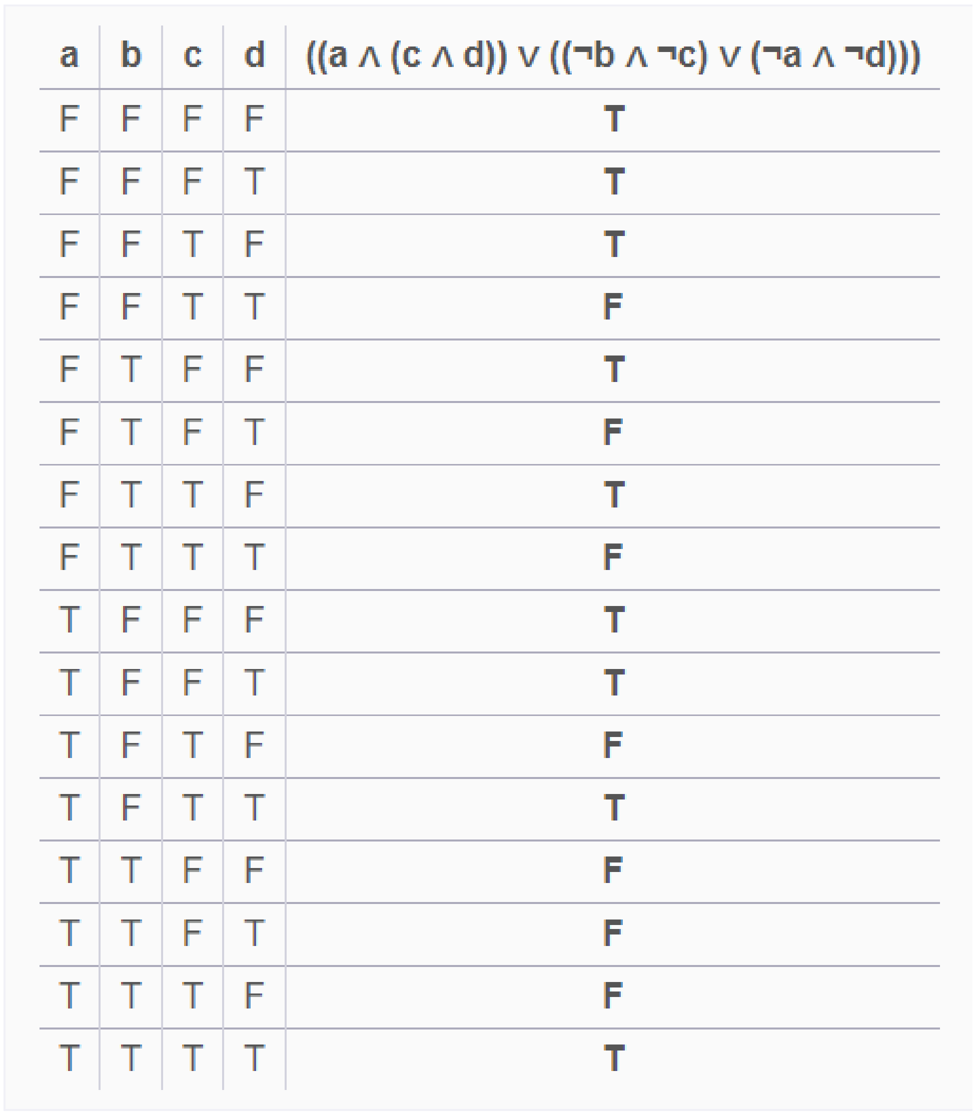


Figure 12: exp1 circuit IRL

Click here to see all the lab photos of the input combination both for experiment 1 and experiment 2.



a	b	c	d	((a ∧ (c ∧ d)) ∨ ((¬b ∧ ¬c) ∨ (¬a ∧ ¬d)))
F	F	F	F	**T**
F	F	F	T	**T**
F	F	T	F	**T**
F	F	T	T	**F**
F	T	F	F	**T**
F	T	F	T	**F**
F	T	T	F	**T**
F	T	T	T	**F**
T	F	F	F	**T**
T	F	F	T	**T**
T	F	T	F	**F**
T	F	T	T	**T**
T	T	F	F	**F**
T	T	F	T	**F**
T	T	T	F	**F**
T	T	T	T	**T**

Figure 13: exp1 truth table

3.2 Experiment part 2

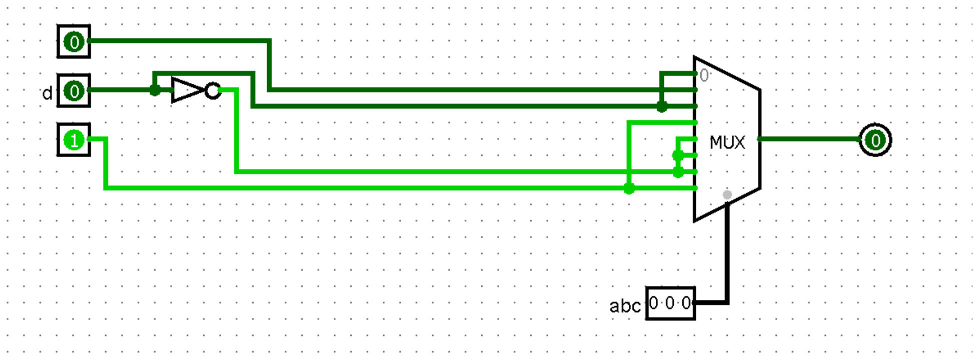


Figure 14: circuit

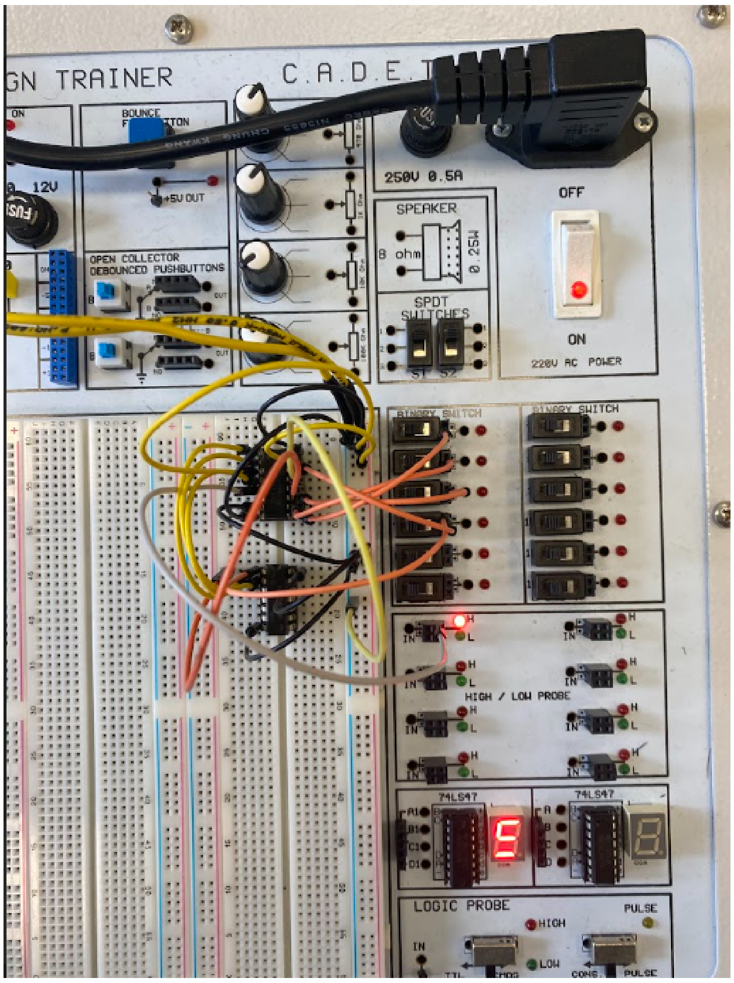


Figure 15: exp1 circuit IRL

Click here to see all the lab photos of the input combination both for experiment 1 and experiment 2.

a	b	c	Value
0	0	0	1
0	0	1	d’
0	1	0	d’
0	1	1	d’
1	0	0	1
1	0	1	d
1	1	0	0
1	1	1	d

Table 2: Truth Table for a,b,c Combinations

4 DISCUSSION [25 points]

4.1 Experiment part 1

For the implementation of experiment 1, we used 3-input and 2-input AND gates to create the minterms then for the OR gate we used a 3-input OR gate in our model circuit, however when we tried to implement it IRL we did not have a single 3-input OR gate, so we used two 2-input OR gates to create the same effect. By comparing the truth-table in our results section to the model that we built using logisim and the implementation that we did in the lab. We found out that all of the designes were correct and work properly. Regarding the dont-cares (X) in our function and their outputs,

- for 0: binary: 0000: as we can see in the circuit and the simplified expression, the terms b'c' and a'd' would result in a 1 so after we use OR we get a 1 in our output
- for 2: binary: 0010: as we can see in the circuit and the simplified expression, the term a'd' would result in a 1 so after we use OR we get a 1 in our output
- for 8: binary: 1000: as we can see in the circuit and the simplified expression, the term b'c' would result in a 1 so after we use OR we get a 1 in our output
- for 9: binary: 1001: as we can see in the circuit and the simplified expression, the term b'c would result in a 1 so after we use OR we get a 1 in our output
- for 14: binary: 1110: as we can see in the circuit and the simplified expression, there are no terms that would result in this combination of giving a 1 so the output would be 0
- for 15: binary: 1111: as we can see in the circuit and the simplified expression, the term acd would result in a 1 so after we use OR we get a 1 in our output

by comparing those results and our karnaugh map we can see that it does match our map as the (X) don't care in position 14 is not in any group

4.2 Experiment part 2

For the implementation, we used only NOT and multiplexer as requested. For NOT gate we used 7404 HEX inverter and for multiplexer we used 74151 8-input multiplexer. As always, first we connected Vcc to 5V and Gnd to 0V for both circuits. Beside of this connections, we also connected 0v to E' for the multiplexer, otherwise multiplexer will not work. Then we connected d to NOT gate to get d'. After that, we connected S2 to

A, S1 to B and S0 to C. For the inputs, we connected them as in the combinations of the truth table given above in results part. 3.2

Then we tried our implementation for all the input combinations for a, b, c and d. As it can be seen in the photos , our circuit works properly according the truth table 3.1.

5 CONCLUSION [10 points]

In this laboratory section, we did 2 experiments. First we find expression from given function using first K-Map and then Quine MC-Cluskey methods. We found least costly expression. Then using this expression:

- At first we implemented this equation using AND, OR, NOT gates.
- Then in the second experiment we used only NOT and multiplexer to implement same equations.

In the both experiment we made connection without any problem and got the correct results at the very first trial.