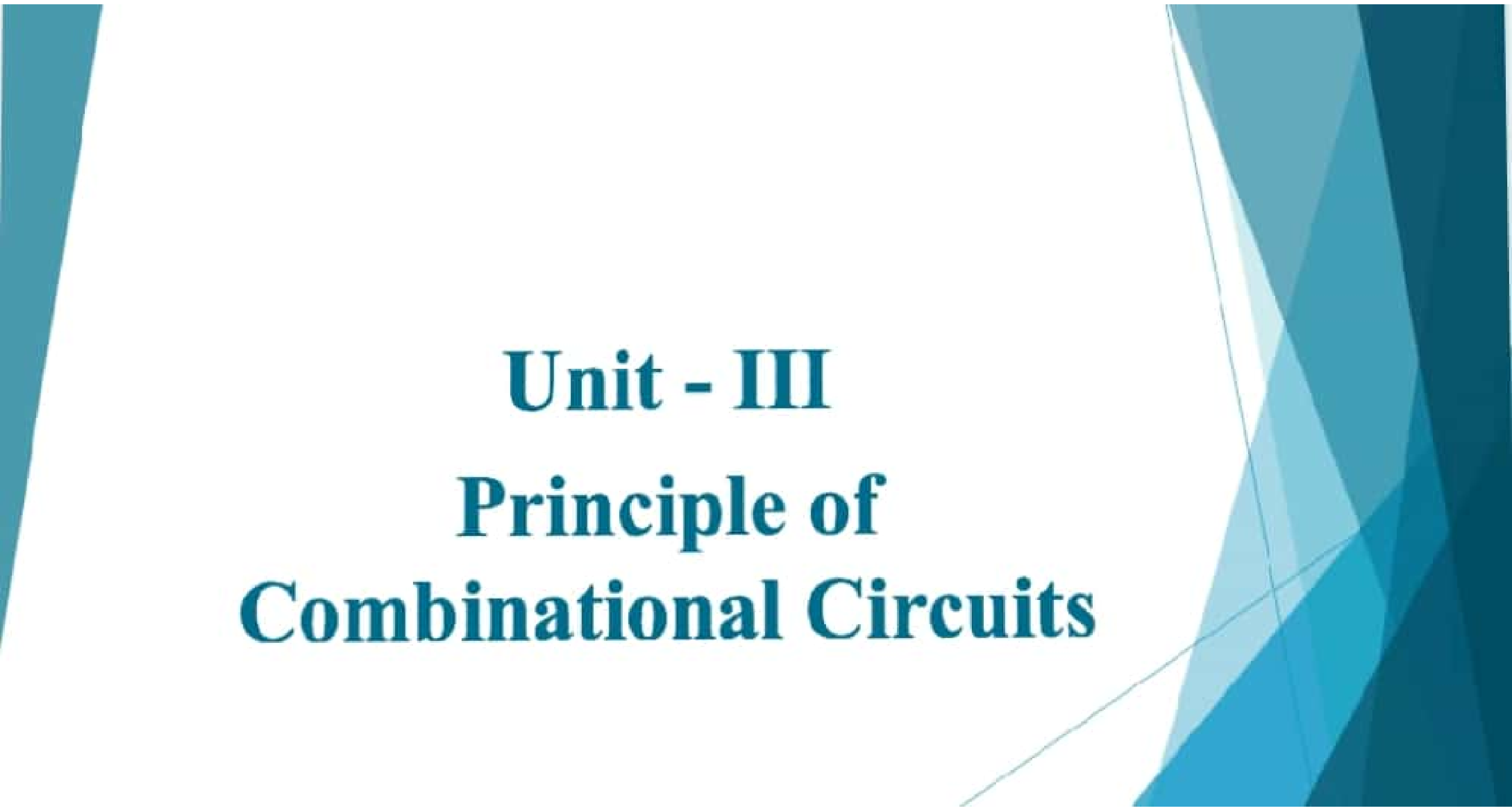
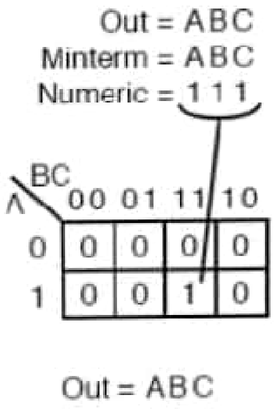


Unit - III

Principle of Combinational Circuits

Minterm

- A *minterm* is a Boolean expression resulting in **1** for the output of a single cell, and **0**s for all other cells in a Karnaugh map (K-map), or truth table. If a minterm has a single 1 and the remaining cells as **0**s, it would appear to cover a minimum area of **1**s.



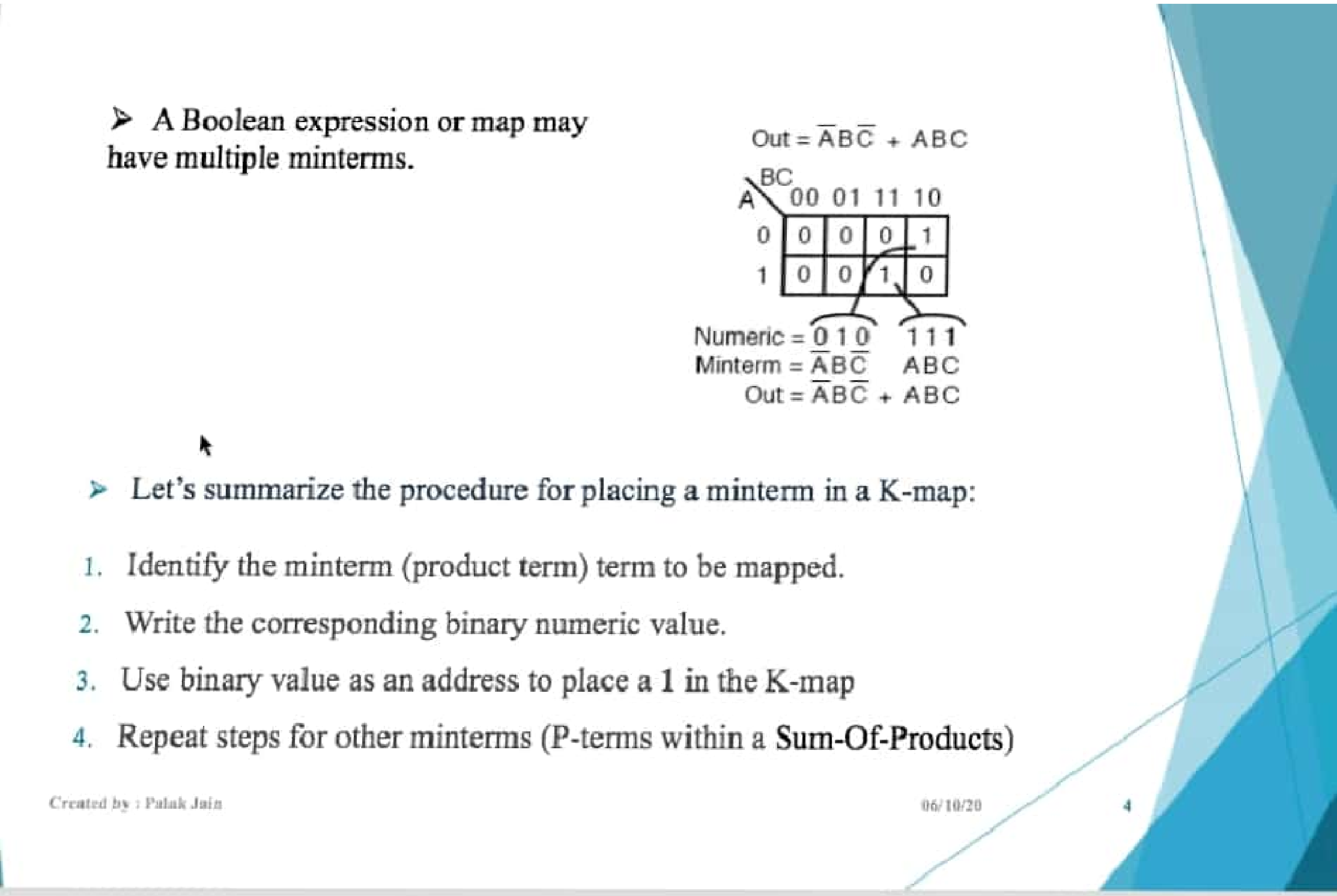
Out = ABC
Minterm = ABC
Numeric = 1 1 1

A \ BC	00	01	11	10
0	0	0	0	0
1	0	0	1	0

Out = ABC

This illustration shows the minterm ABC, a single product term and minterm is represented by 1.

The cell 111 corresponds to the minterm ABC



- A Boolean expression or map may have multiple minterms.

Out = $\overline{A}B\overline{C} + ABC$

A \ BC	00	01	11	10
0	0	0	0	1
1	0	0	1	0

Numeric = $\overline{0}1\overline{0}$ 111

Minterm = $\overline{A}B\overline{C}$ ABC

Out = $\overline{A}B\overline{C} + ABC$

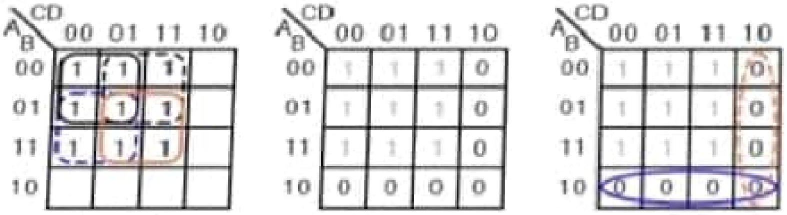
- Let's summarize the procedure for placing a minterm in a K-map:

1. Identify the minterm (product term) term to be mapped.
2. Write the corresponding binary numeric value.
3. Use binary value as an address to place a 1 in the K-map
4. Repeat steps for other minterms (P-terms within a **Sum-Of-Products**)

Example:

Let us revisit a previous problem involving an SOP minimization. Produce a Product-Of-Sums solution. Compare the POS solution to the previous SOP.

$$
\begin{aligned}
\text{Out} = {} & \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}CD \\
& + \overline{A}B\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD \\
& + AB\overline{C}\,\overline{D} + AB\overline{C}D + ABCD
\end{aligned}
$$



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

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

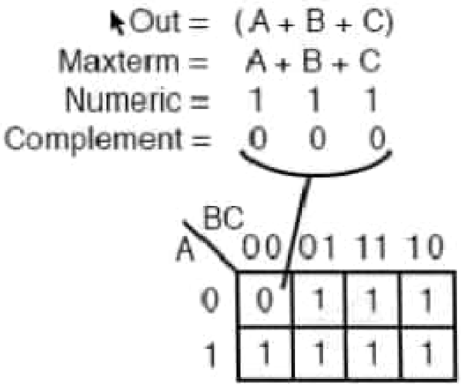
AB \ CD	00	01	11	10
00	1	1	1	0
01	1	1	1	0
11	1	1	1	0
10	0	0	0	0

$\text{Out} = \overline{A}\,\overline{C} + \overline{A}D + B\overline{C} + BD$

$\text{Out} = (\overline{A}+B)\,(\overline{C}+D)$

Maxterm

- A *maxterm* is a Boolean expression resulting in a **0** for the output of a single cell expression, and **1**s for all other cells in the Karnaugh map, or truth table. The illustration above left shows the maxterm **(A+B+C)**, a single sum term, as a single **0** in a map that is otherwise **1**s.

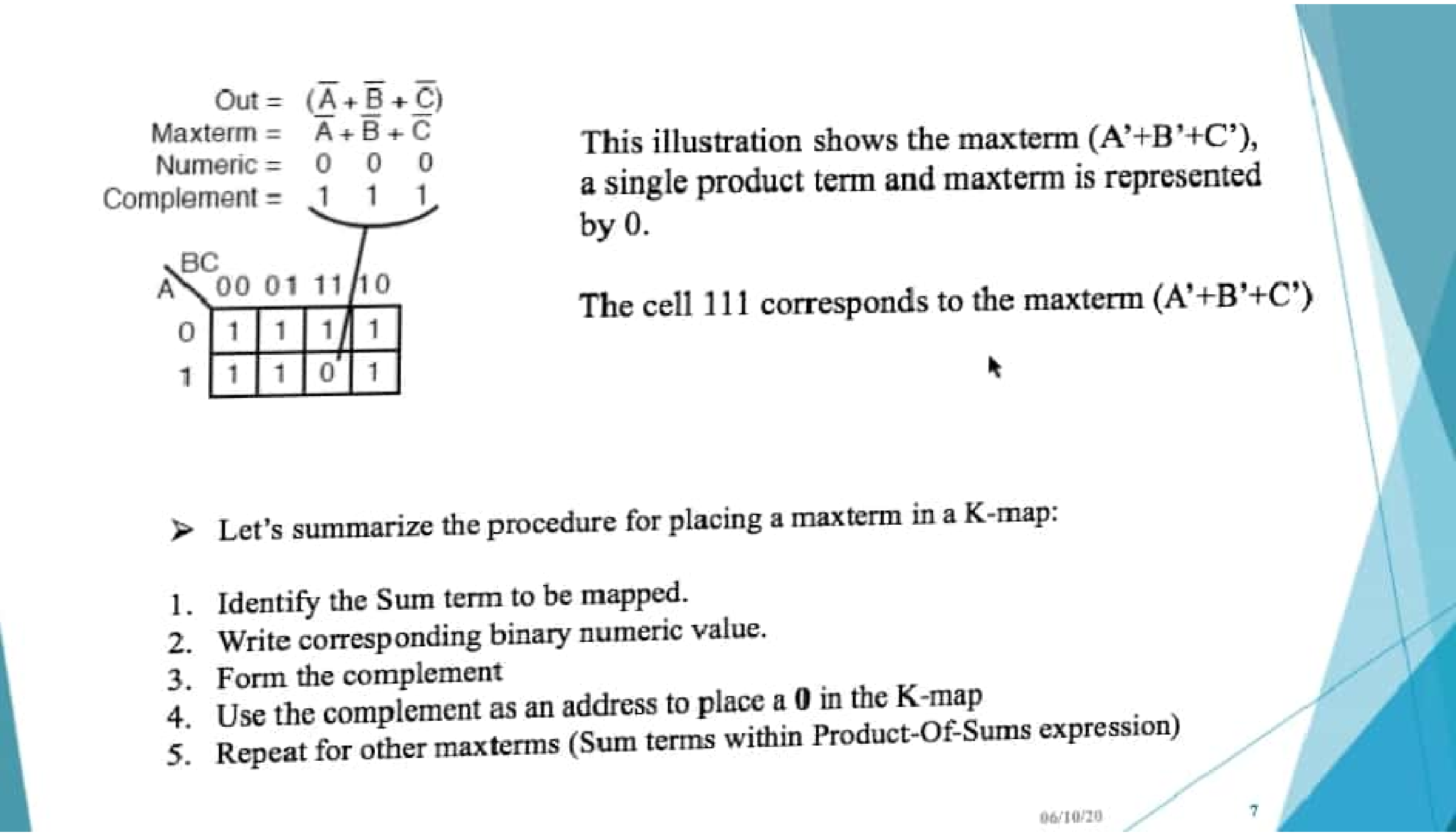


Out = (A + B + C)
Maxterm = A + B + C
Numeric = 1 1 1
Complement = 0 0 0

A \ BC	00	01	11	10
0	0	1	1	1
1	1	1	1	1

This illustration shows the maxterm (A+B+C), a single product term and maxterm is represented by 0.

The cell 000 corresponds to the maxterm (A+B+C)



$$\begin{aligned}\text{Out} &= (\overline{A}+\overline{B}+\overline{C})\\ \text{Maxterm} &= \overline{A}+\overline{B}+\overline{C}\\ \text{Numeric} &= 0\quad 0\quad 0\\ \text{Complement} &= \underbrace{1\quad 1\quad 1}\end{aligned}$$

A \ BC	00	01	11	10
0	1	1	1	1
1	1	1	0	1

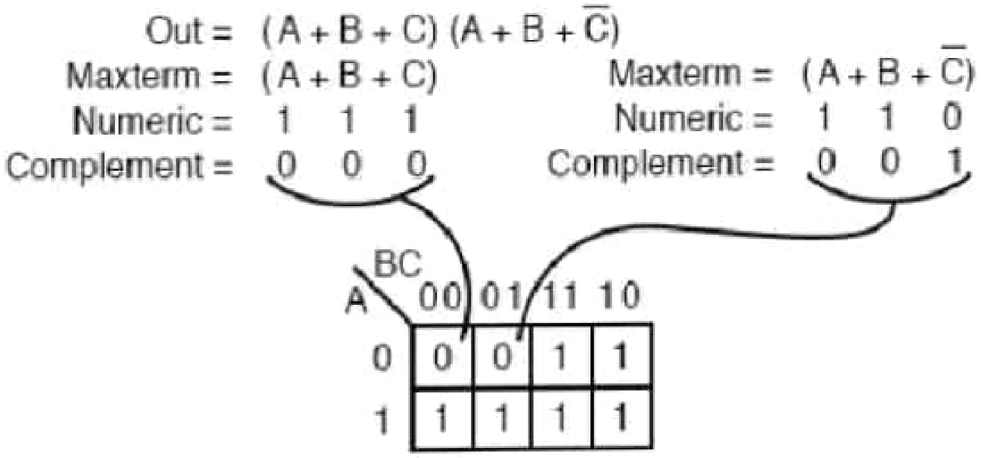
This illustration shows the maxterm (A'+B'+C'), a single product term and maxterm is represented by 0.

The cell 111 corresponds to the maxterm (A'+B'+C')

➢ Let's summarize the procedure for placing a maxterm in a K-map:

1. Identify the Sum term to be mapped.
2. Write corresponding binary numeric value.
3. Form the complement
4. Use the complement as an address to place a **0** in the K-map
5. Repeat for other maxterms (Sum terms within Product-Of-Sums expression)

- A Boolean expression or map may have multiple maxterms.



Out = $(A + B + C)\,(A + B + \overline{C})$

Maxterm = $(A + B + C)$

Numeric = 1 1 1

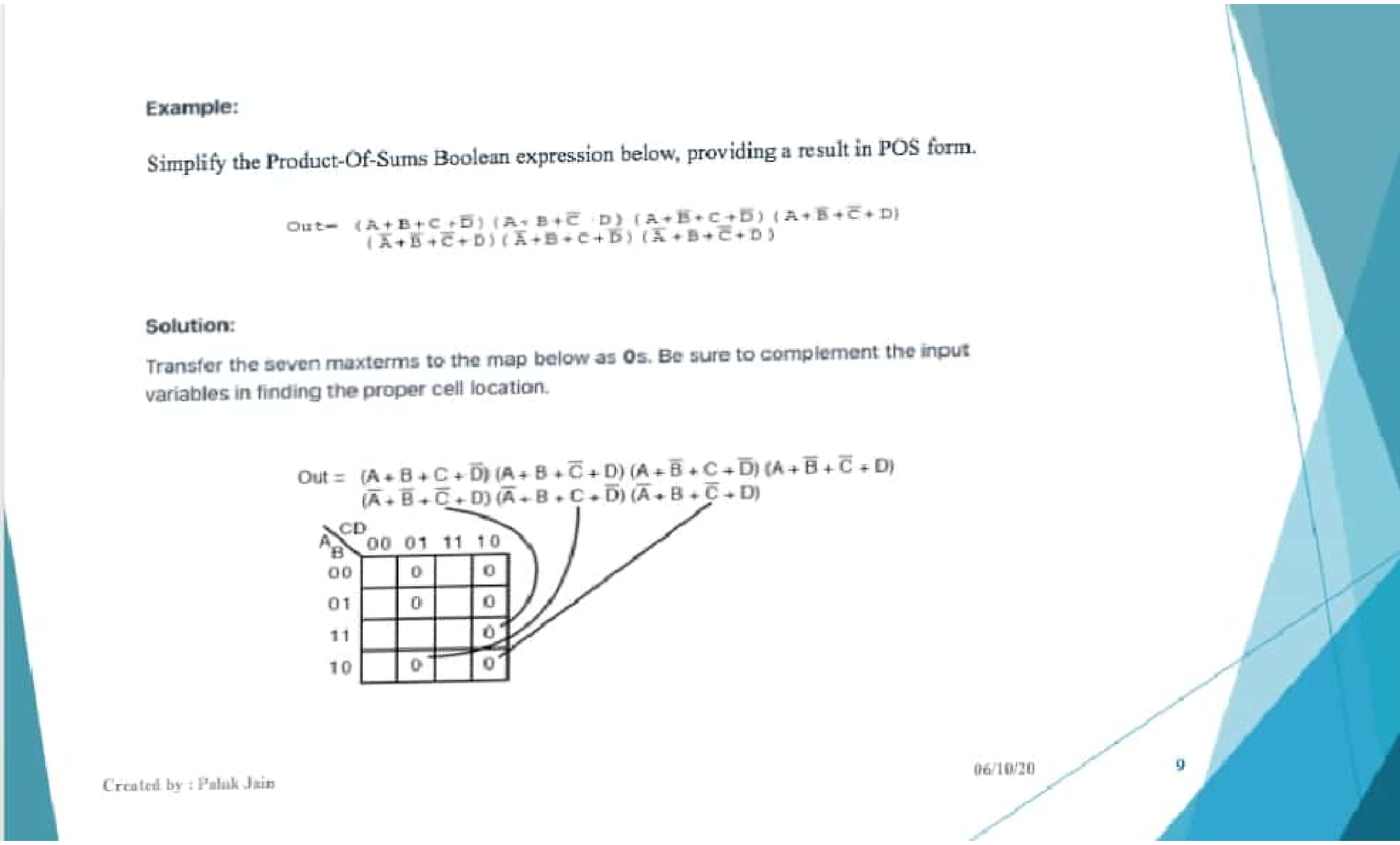
Complement = 0 0 0

Maxterm = $(A + B + \overline{C})$

Numeric = 1 1 0

Complement = 0 0 1

A \ BC	00	01	11	10
0	0	0	1	1
1	1	1	1	1



Example:

Simplify the Product-Of-Sums Boolean expression below, providing a result in POS form.

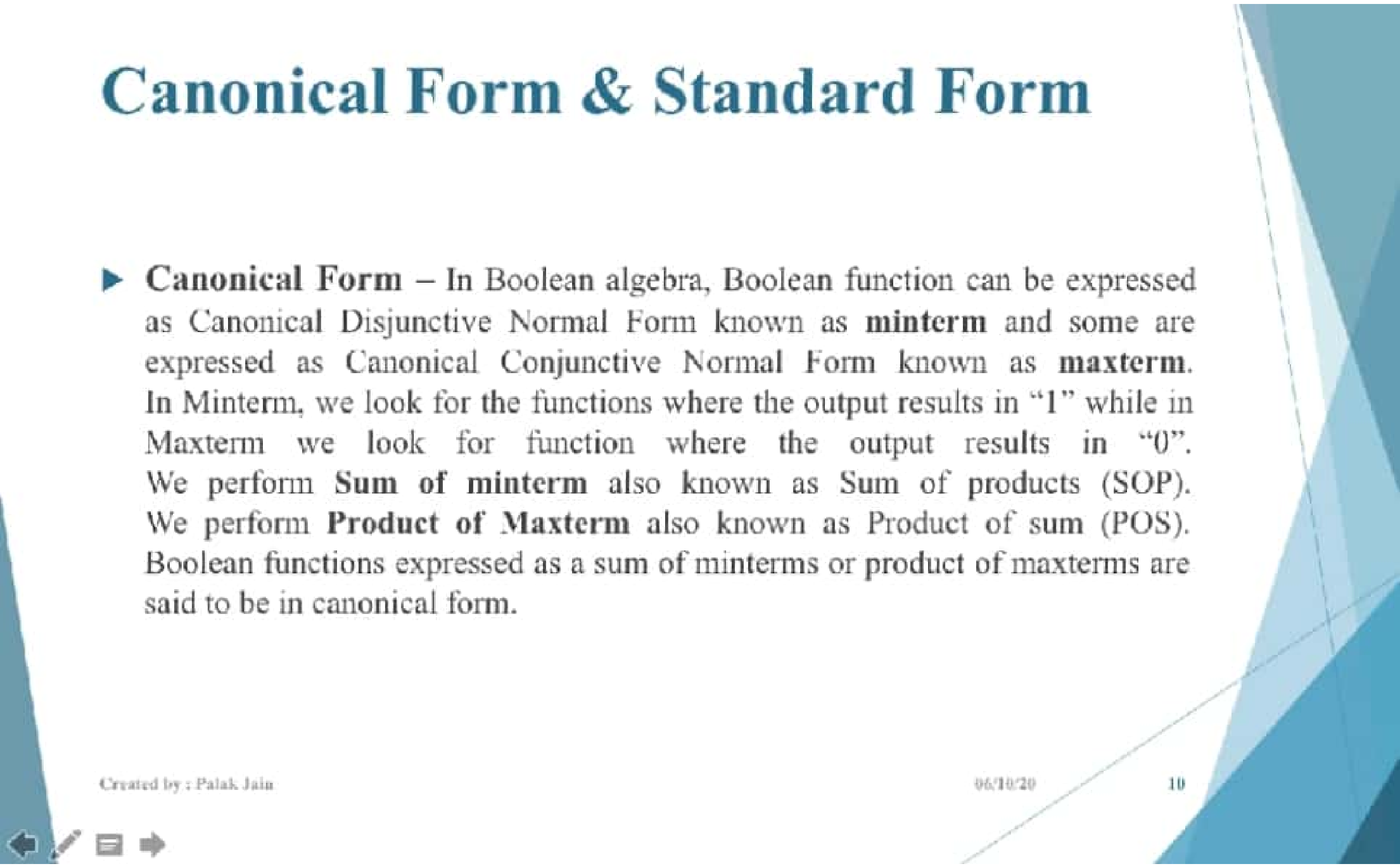
$$Out= (A+B+C+\overline{D})(A+B+\overline{C}+D)(A+\overline{B}+C+\overline{D})(A+\overline{B}+\overline{C}+D)$$
$$(\overline{A}+\overline{B}+\overline{C}+D)(\overline{A}+B+C+\overline{D})(\overline{A}+B+\overline{C}+D)$$

Solution:

Transfer the seven maxterms to the map below as **0**s. Be sure to complement the input variables in finding the proper cell location.

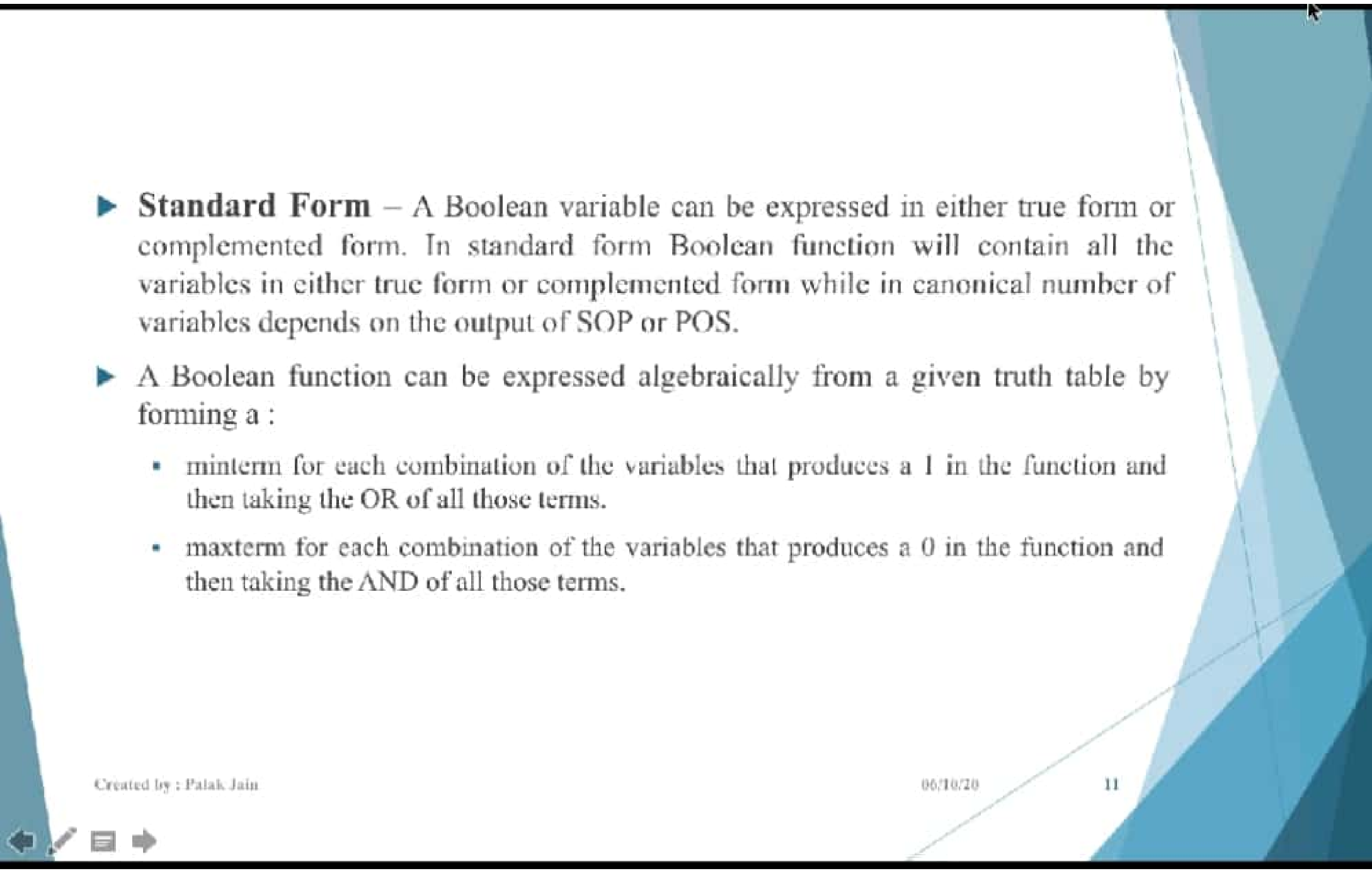
$$Out = (A+B+C+\overline{D})(A+B+\overline{C}+D)(A+\overline{B}+C+\overline{D})(A+\overline{B}+\overline{C}+D)$$
$$(\overline{A}+\overline{B}+\overline{C}+D)(\overline{A}+B+C+\overline{D})(\overline{A}+B+\overline{C}+D)$$

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

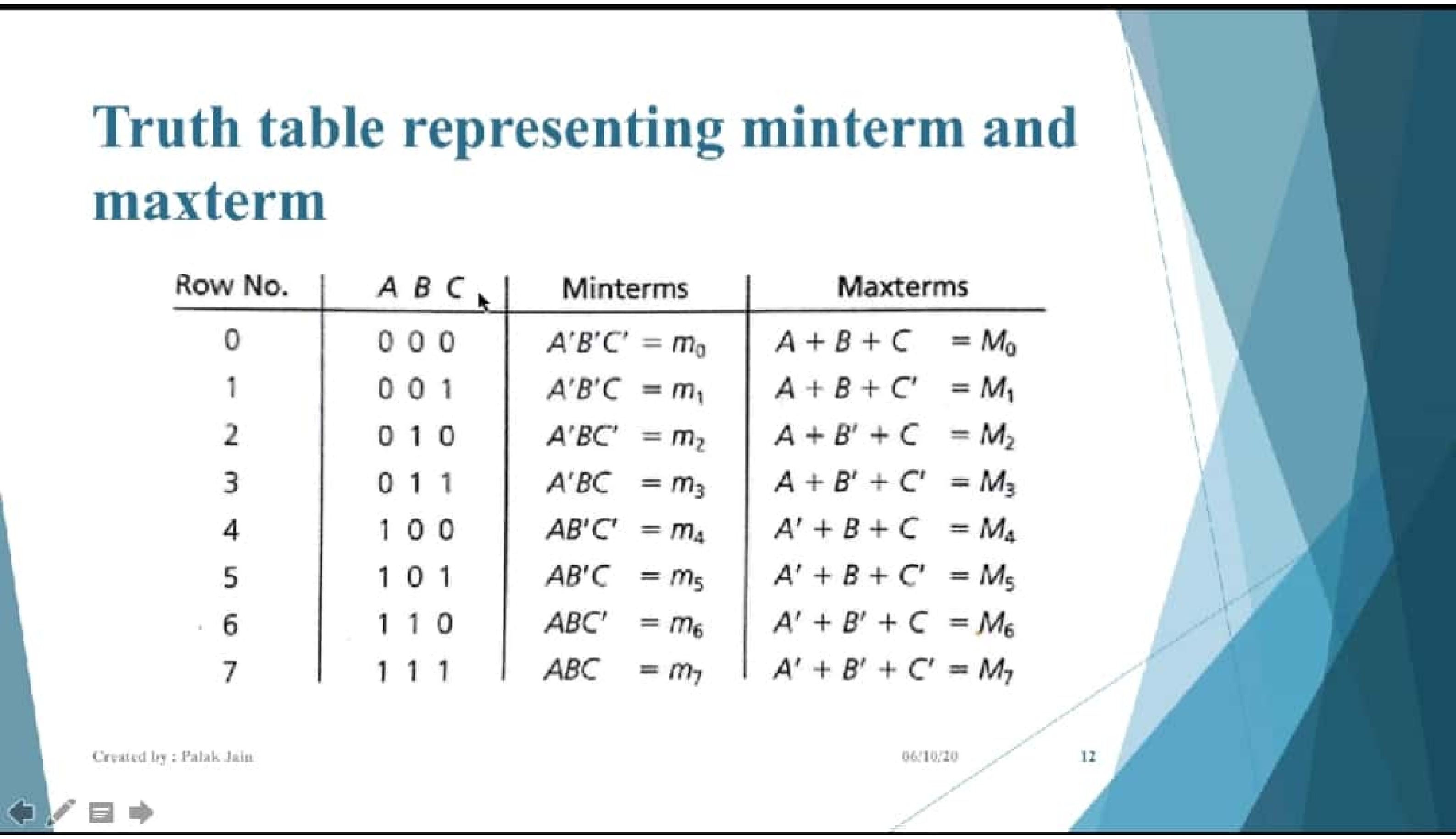


Canonical Form & Standard Form

- **Canonical Form** – In Boolean algebra, Boolean function can be expressed as Canonical Disjunctive Normal Form known as **minterm** and some are expressed as Canonical Conjunctive Normal Form known as **maxterm**. In Minterm, we look for the functions where the output results in "1" while in Maxterm we look for function where the output results in "0". We perform **Sum of minterm** also known as Sum of products (SOP). We perform **Product of Maxterm** also known as Product of sum (POS). Boolean functions expressed as a sum of minterms or product of maxterms are said to be in canonical form.

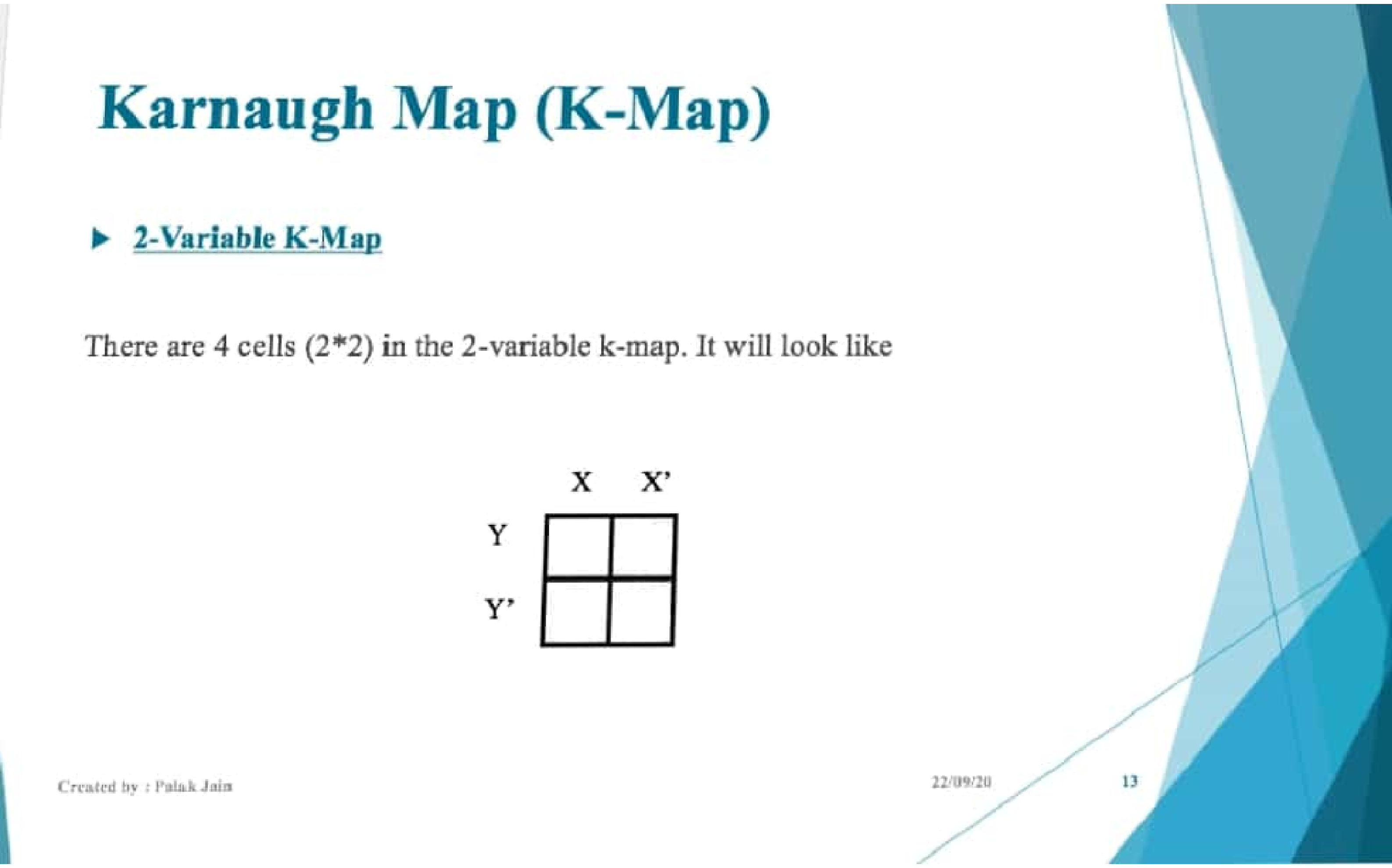


- **Standard Form** – A Boolean variable can be expressed in either true form or complemented form. In standard form Boolean function will contain all the variables in either true form or complemented form while in canonical number of variables depends on the output of SOP or POS.
- A Boolean function can be expressed algebraically from a given truth table by forming a :
  - minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms.
  - maxterm for each combination of the variables that produces a 0 in the function and then taking the AND of all those terms.



Truth table representing minterm and maxterm

Row No.	$A\ B\ C$	Minterms	Maxterms
0	0 0 0	$A'B'C' = m_0$	$A + B + C = M_0$
1	0 0 1	$A'B'C = m_1$	$A + B + C' = M_1$
2	0 1 0	$A'BC' = m_2$	$A + B' + C = M_2$
3	0 1 1	$A'BC = m_3$	$A + B' + C' = M_3$
4	1 0 0	$AB'C' = m_4$	$A' + B + C = M_4$
5	1 0 1	$AB'C = m_5$	$A' + B + C' = M_5$
6	1 1 0	$ABC' = m_6$	$A' + B' + C = M_6$
7	1 1 1	$ABC = m_7$	$A' + B' + C' = M_7$

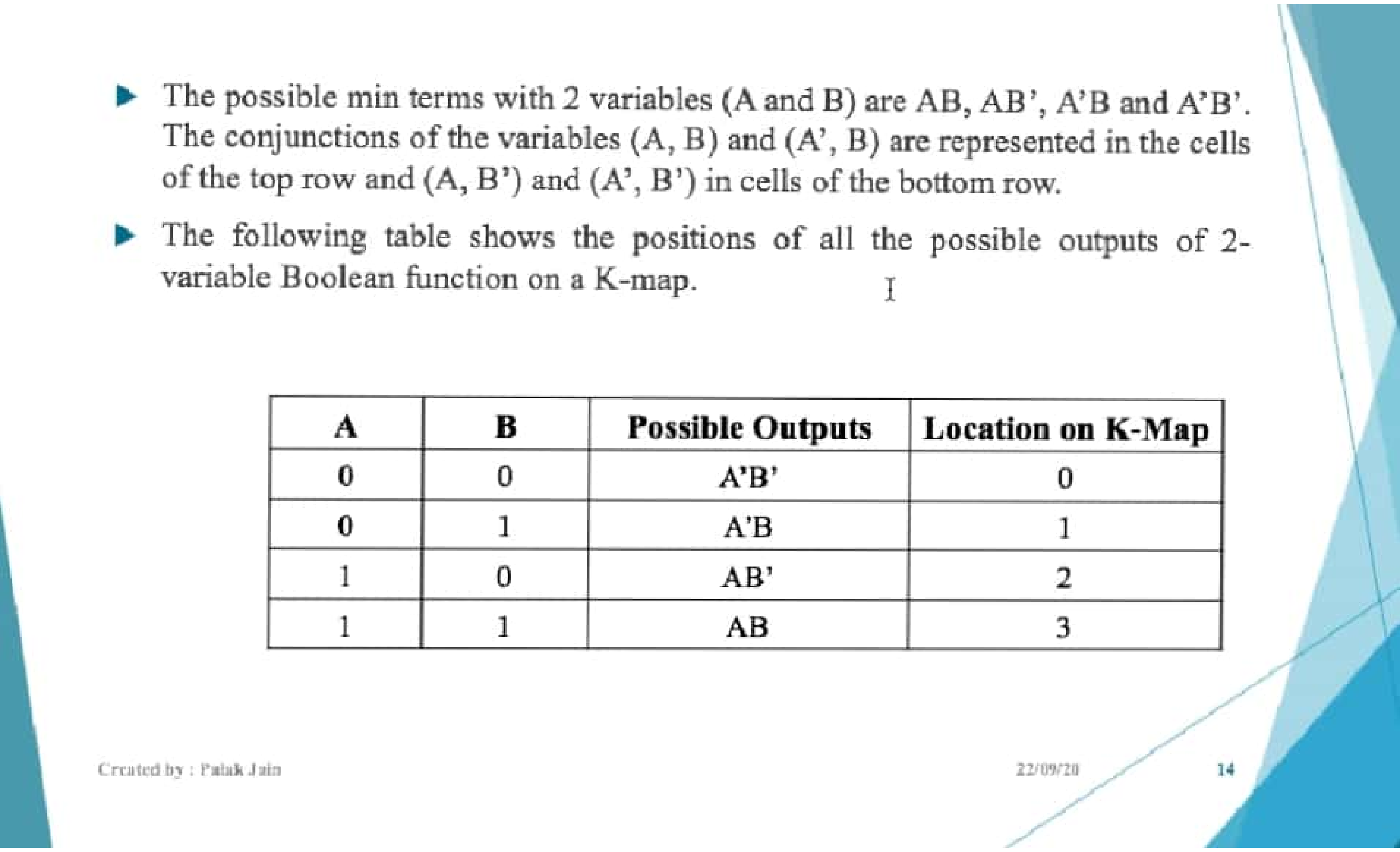


Karnaugh Map (K-Map)

- **2-Variable K-Map**

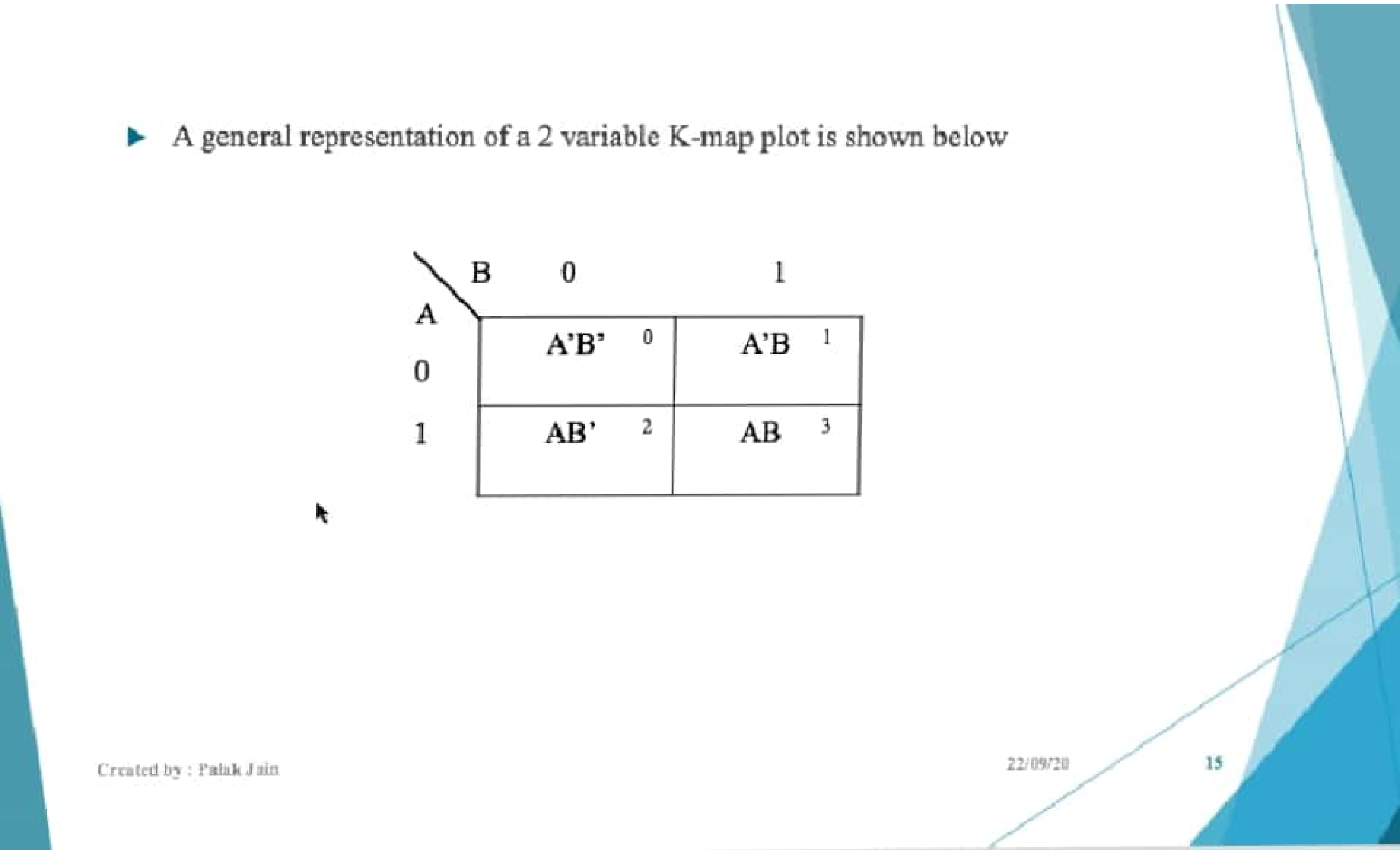
There are 4 cells (2*2) in the 2-variable k-map. It will look like

	X	X’
Y		
Y’		



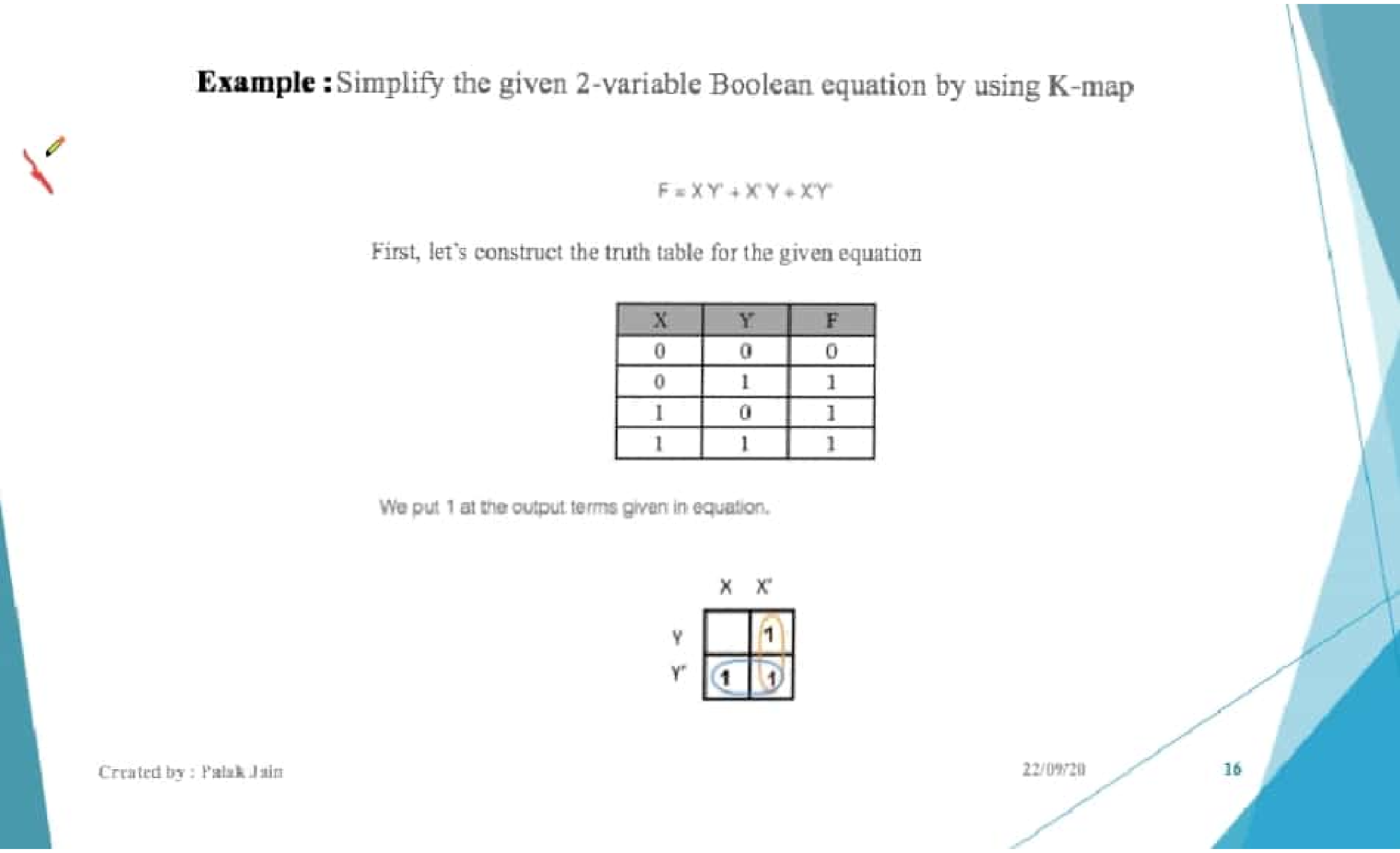
- The possible min terms with 2 variables (A and B) are AB, AB', A'B and A'B'. The conjunctions of the variables (A, B) and (A', B) are represented in the cells of the top row and (A, B') and (A', B') in cells of the bottom row.
- The following table shows the positions of all the possible outputs of 2-variable Boolean function on a K-map.

A	B	Possible Outputs	Location on K-Map
0	0	A'B'	0
0	1	A'B	1
1	0	AB'	2
1	1	AB	3



- A general representation of a 2 variable K-map plot is shown below

A \ B	0	1
0	A’B’ 0	A’B 1
1	AB’ 2	AB 3



Example : Simplify the given 2-variable Boolean equation by using K-map

$$F = XY' + X'Y + X'Y'$$

First, let's construct the truth table for the given equation

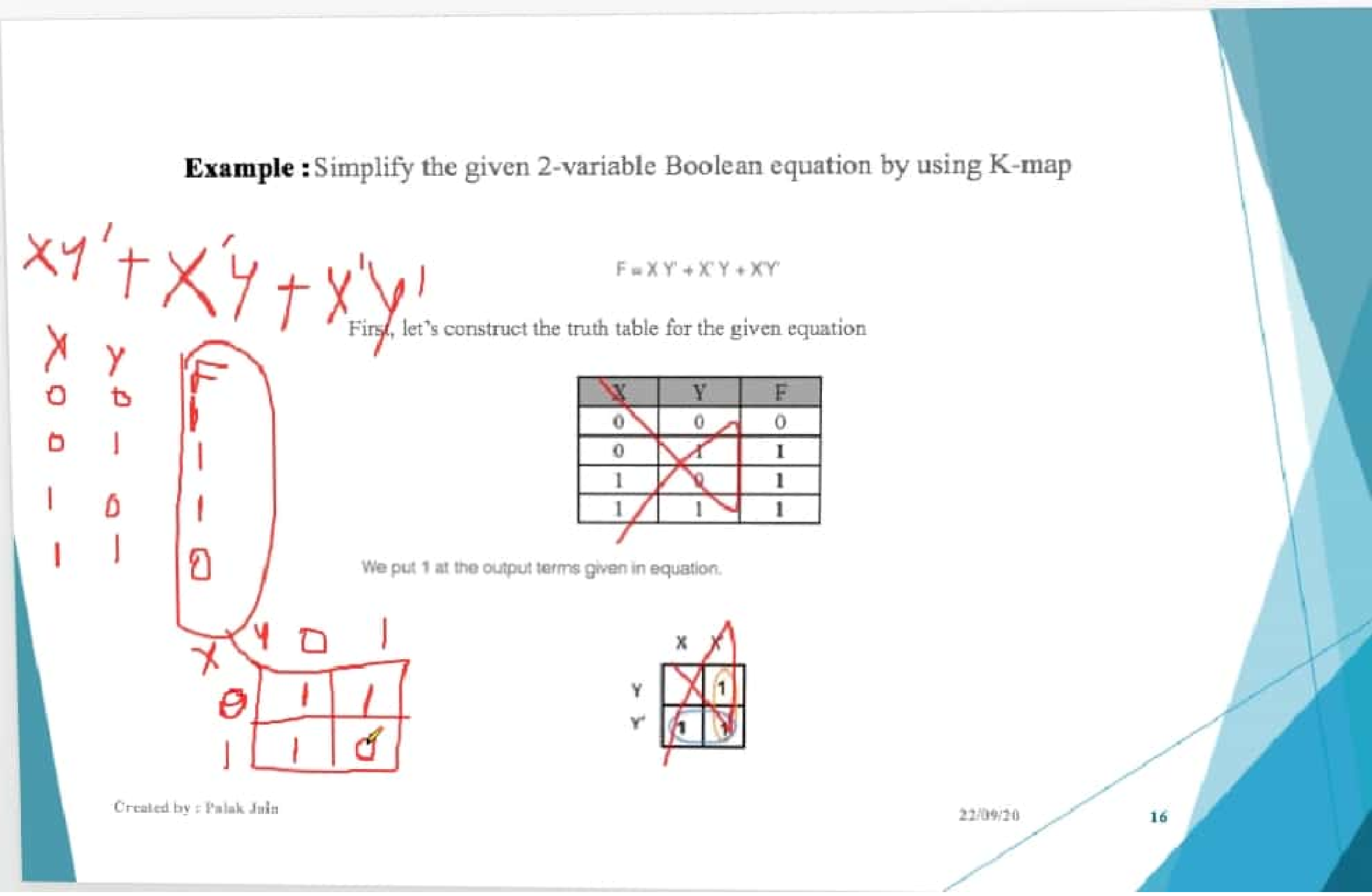
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	1

We put 1 at the output terms given in equation.

	X	X'
Y		1
Y'	1	1

Created by : Palak Jain

22/09/20



Example : Simplify the given 2-variable Boolean equation by using K-map

$xy' + x'y + x'y'$

$F = XY' + X'Y + XY'$

First, let's construct the truth table for the given equation

X	Y	F
0	0	0
0	1	1
1	0	1
1	1	1

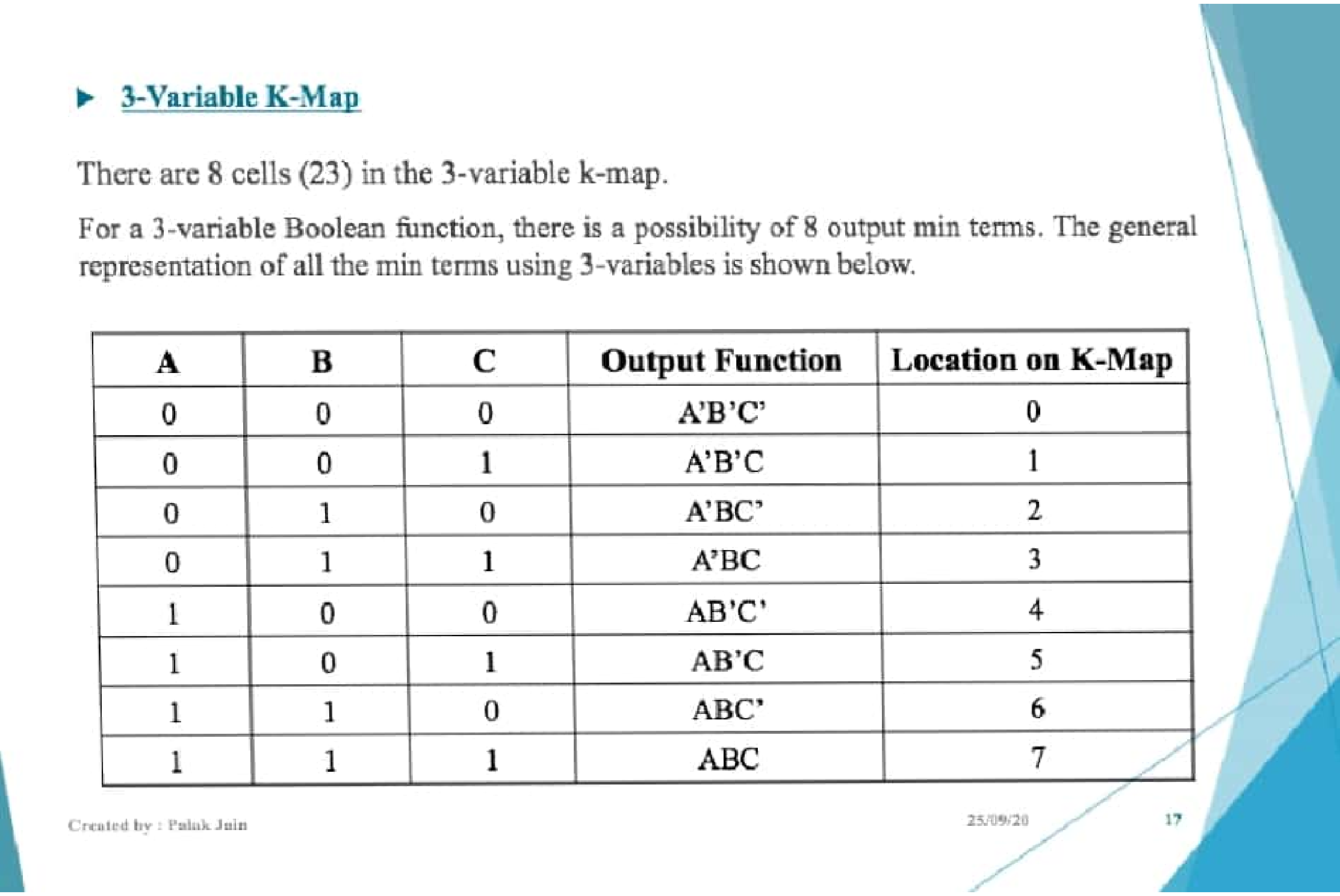
X	Y	F
0	0	1
0	1	1
1	0	1
1	1	0

We put 1 at the output terms given in equation.

X \ Y	0	1
0	1	1
1	1	0

	X	X'
Y		1
Y'	1	1

Created by : Palak Jain

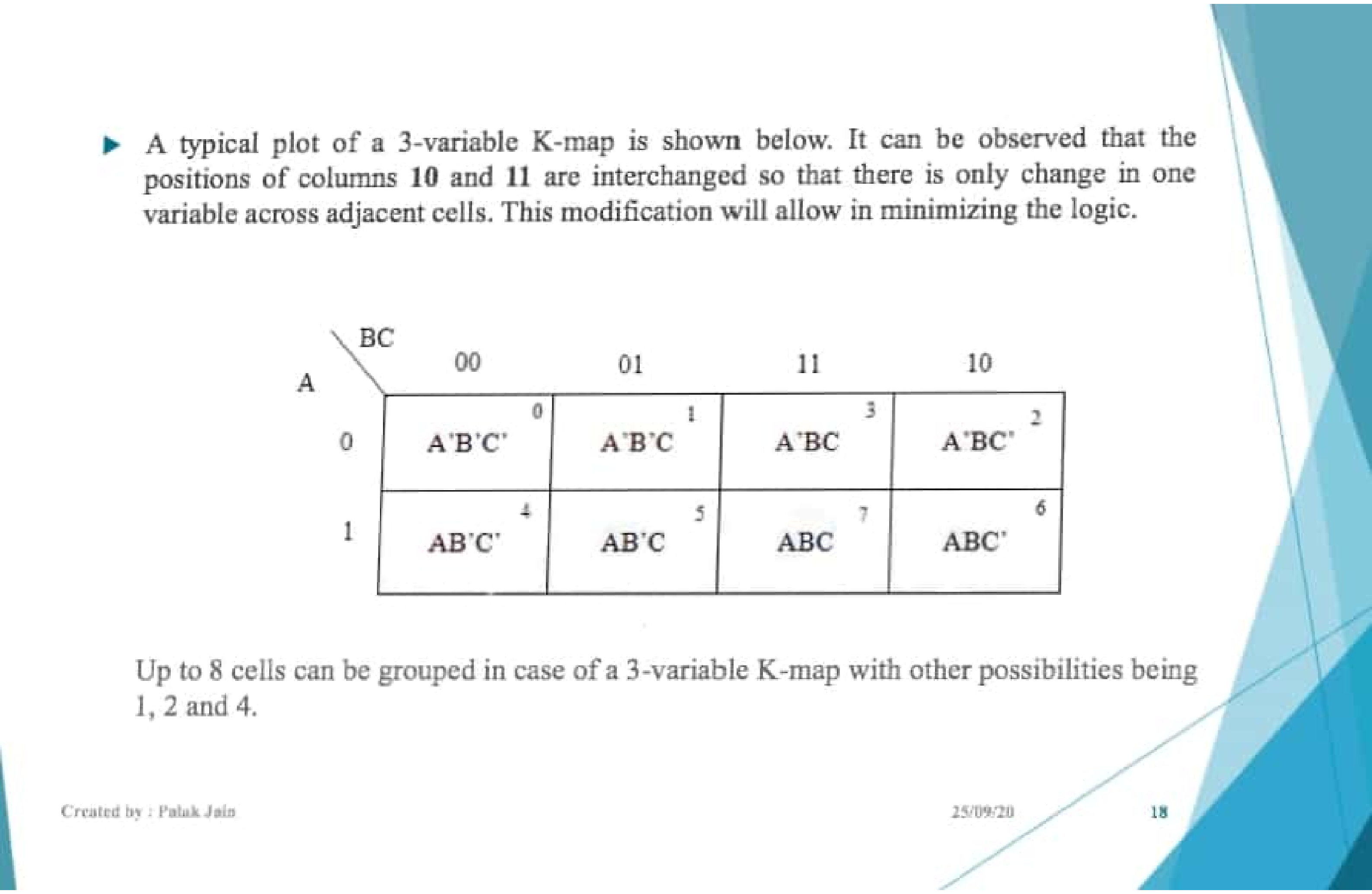


3-Variable K-Map

There are 8 cells (23) in the 3-variable k-map.

For a 3-variable Boolean function, there is a possibility of 8 output min terms. The general representation of all the min terms using 3-variables is shown below.

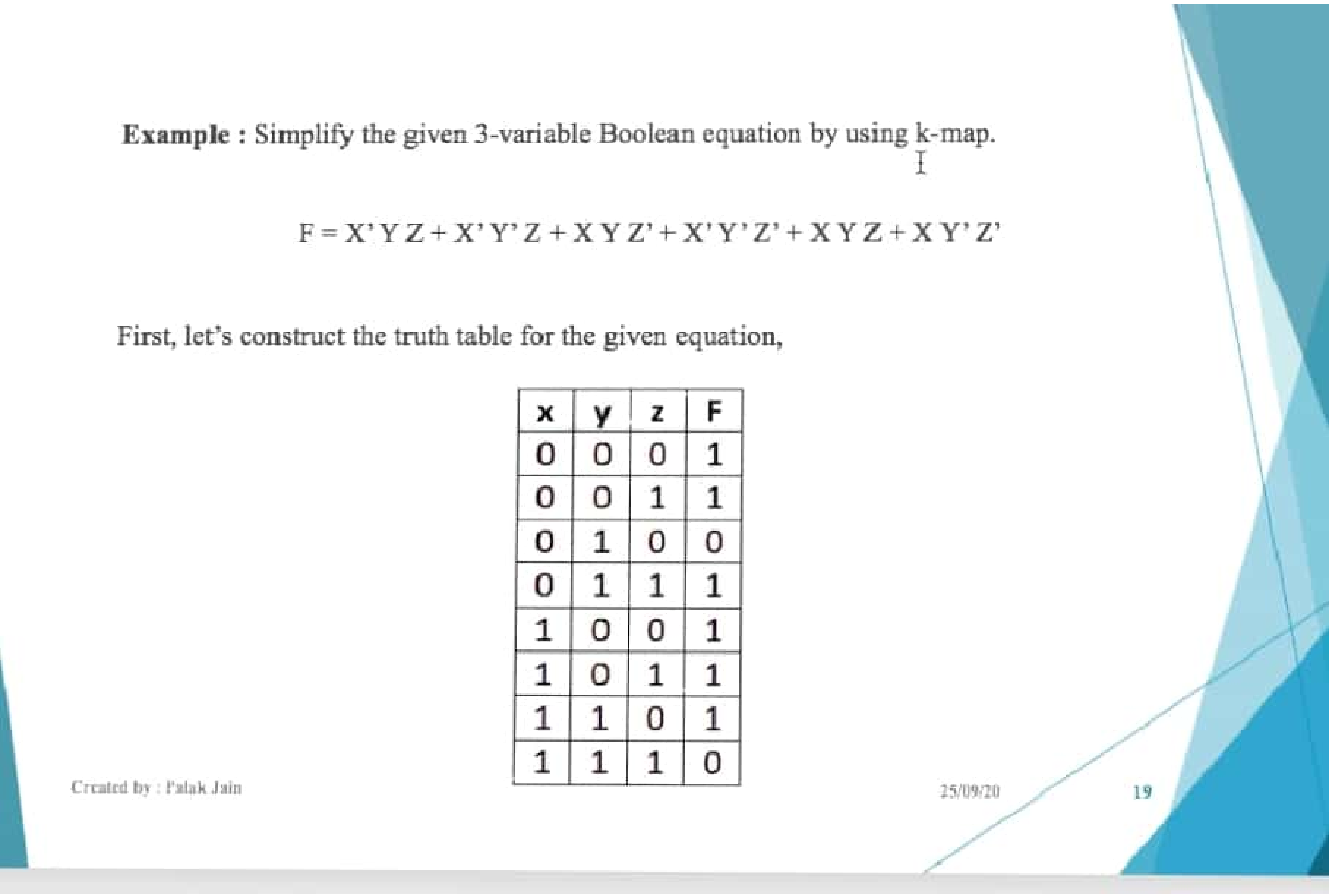
A	B	C	Output Function	Location on K-Map
0	0	0	A'B'C'	0
0	0	1	A'B'C	1
0	1	0	A'BC'	2
0	1	1	A'BC	3
1	0	0	AB'C'	4
1	0	1	AB'C	5
1	1	0	ABC'	6
1	1	1	ABC	7



- A typical plot of a 3-variable K-map is shown below. It can be observed that the positions of columns **10** and **11** are interchanged so that there is only change in one variable across adjacent cells. This modification will allow in minimizing the logic.

BC A	00	01	11	10
0	0 A'B'C'	1 A'B'C	3 A'BC	2 A'BC'
1	4 AB'C'	5 AB'C	7 ABC	6 ABC'

Up to 8 cells can be grouped in case of a 3-variable K-map with other possibilities being 1, 2 and 4.

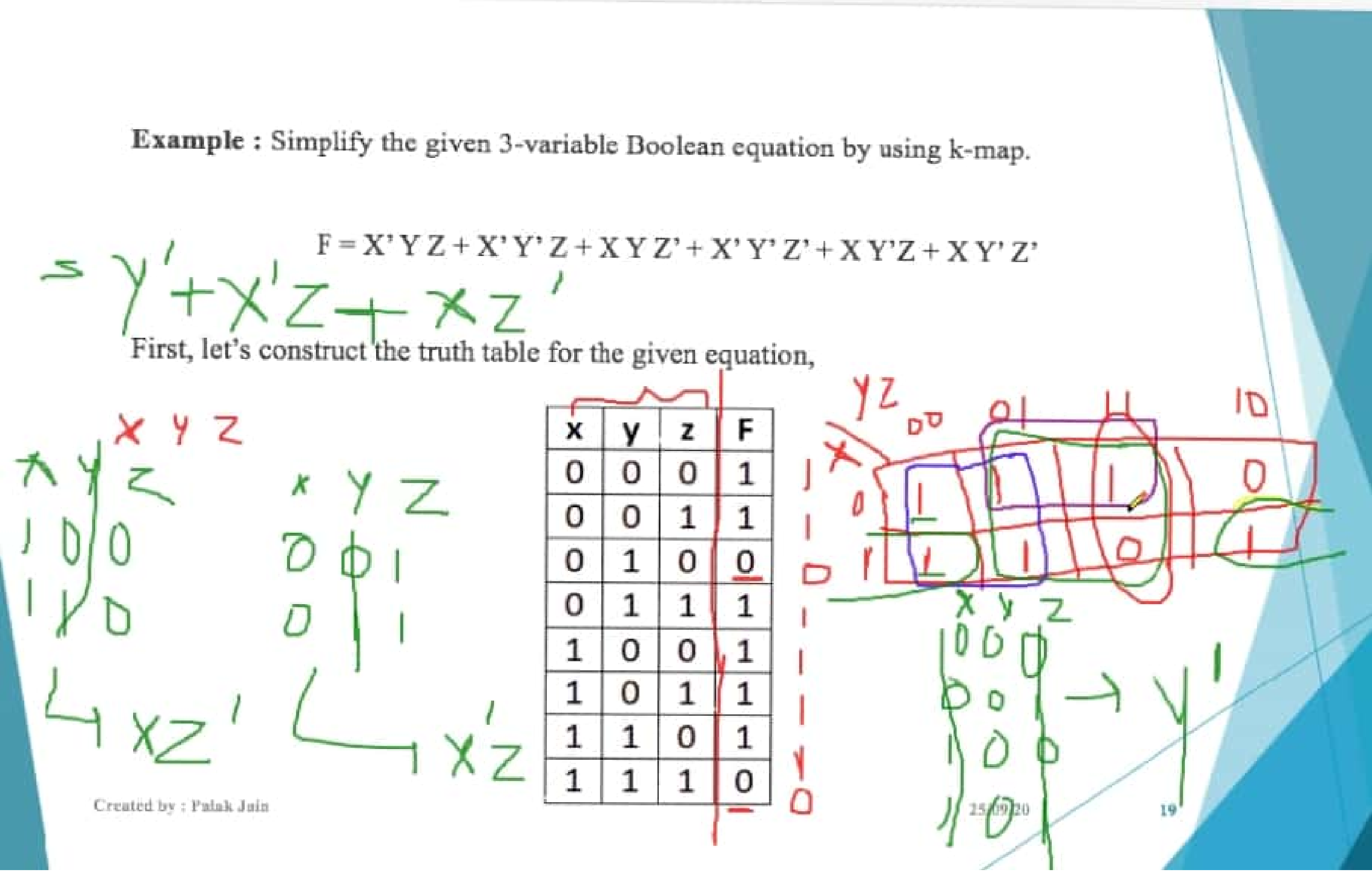


Example : Simplify the given 3-variable Boolean equation by using k-map.

$$F = X'YZ + X'Y'Z + XYZ' + X'Y'Z' + XYZ + XY'Z'$$

First, let's construct the truth table for the given equation,

x	y	z	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



Example : Simplify the given 3-variable Boolean equation by using k-map.

$$F = X'YZ + X'Y'Z + XYZ' + X'Y'Z' + XY'Z + XY'Z'$$

$$= Y' + X'Z + XZ'$$

First, let's construct the truth table for the given equation,

x	y	z	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

1 1 0 1 1 1 1 0

X \ YZ	00	01	11	10
0	1	1	1	0
1	1	1	0	1

x	y	z
1	0	0
1	1	0

→ XZ'

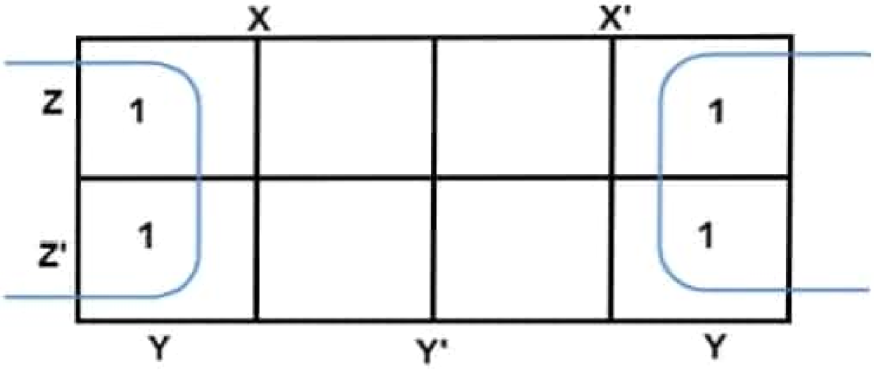
x	y	z
0	0	1
0	1	1

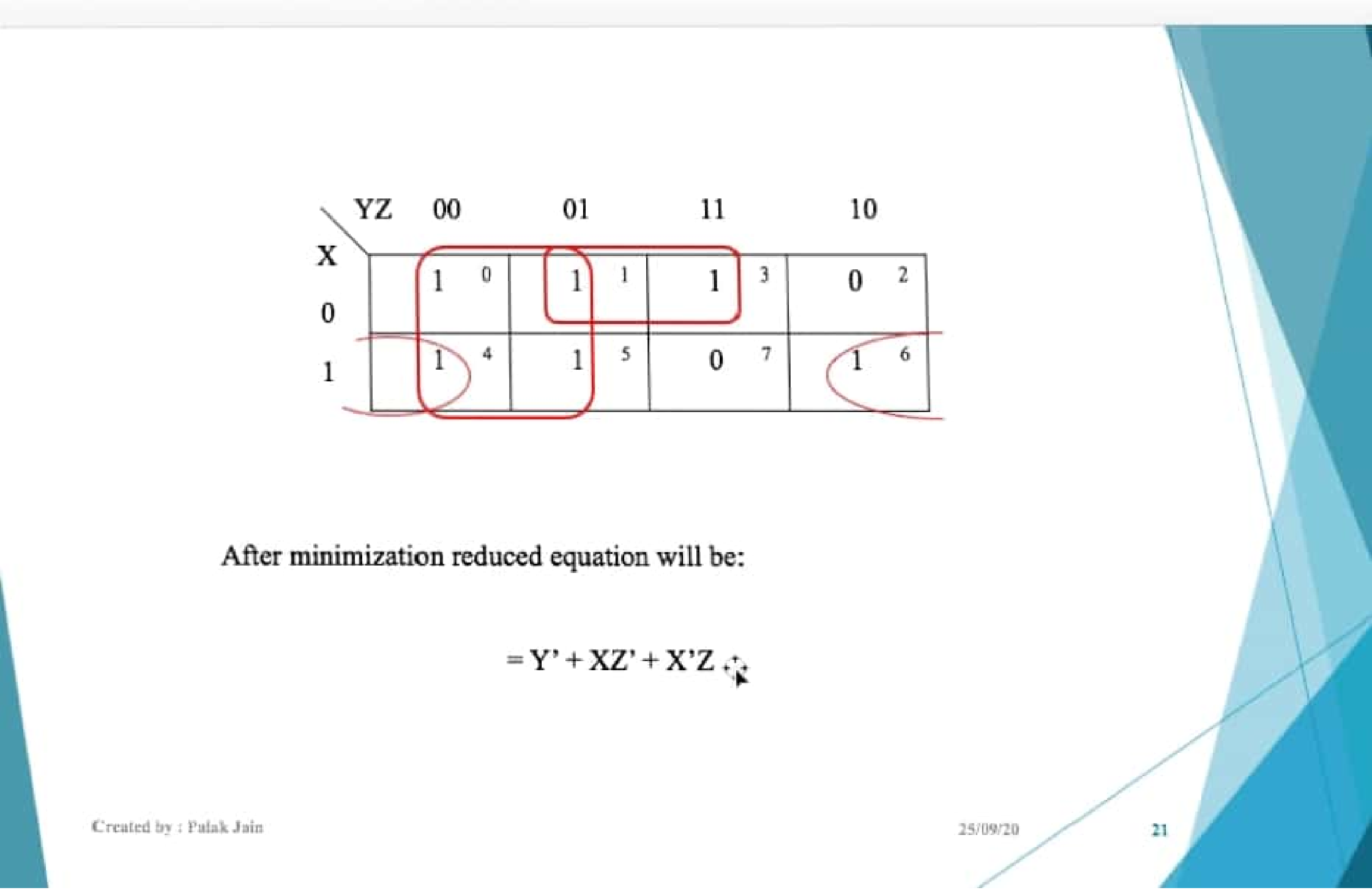
→ $X'Z$

x	y	z
0	0	0
0	0	1
1	0	0
1	0	1

→ Y'

- There are 8 cells (23) in the 3-variable k-map. It will look like (see below image).
- The largest group size will be 8 but we can also form the groups of size 4 and size 2, by possibility. In the 3 variable Karnaugh map, we consider the left most column of the k-map as the adjacent column of rightmost column. So the size 4 group is formed as shown below.

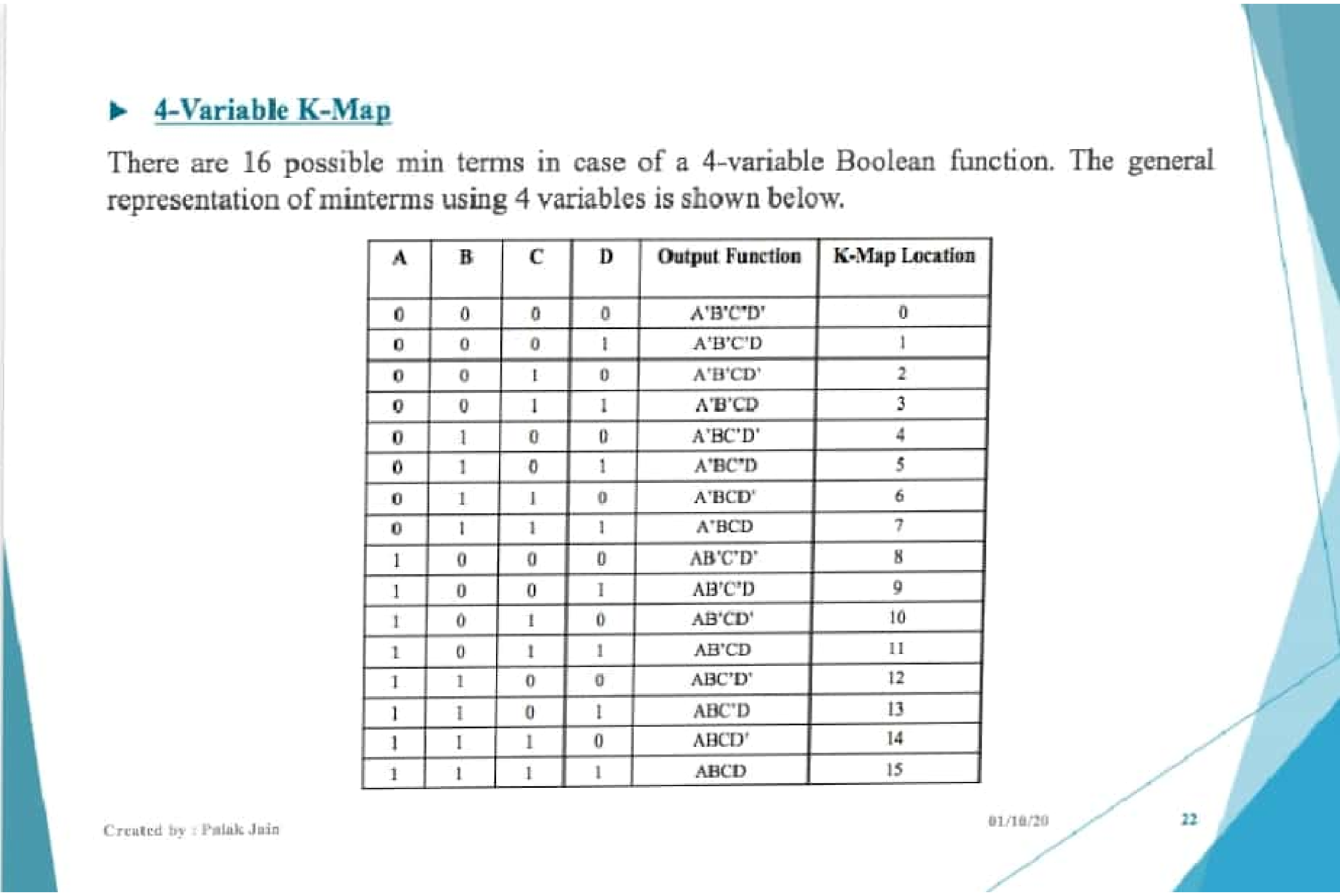




X \ YZ	00	01	11	10
0	1^{0}	1^{1}	1^{3}	0^{2}
1	1^{4}	1^{5}	0^{7}	1^{6}

After minimization reduced equation will be:

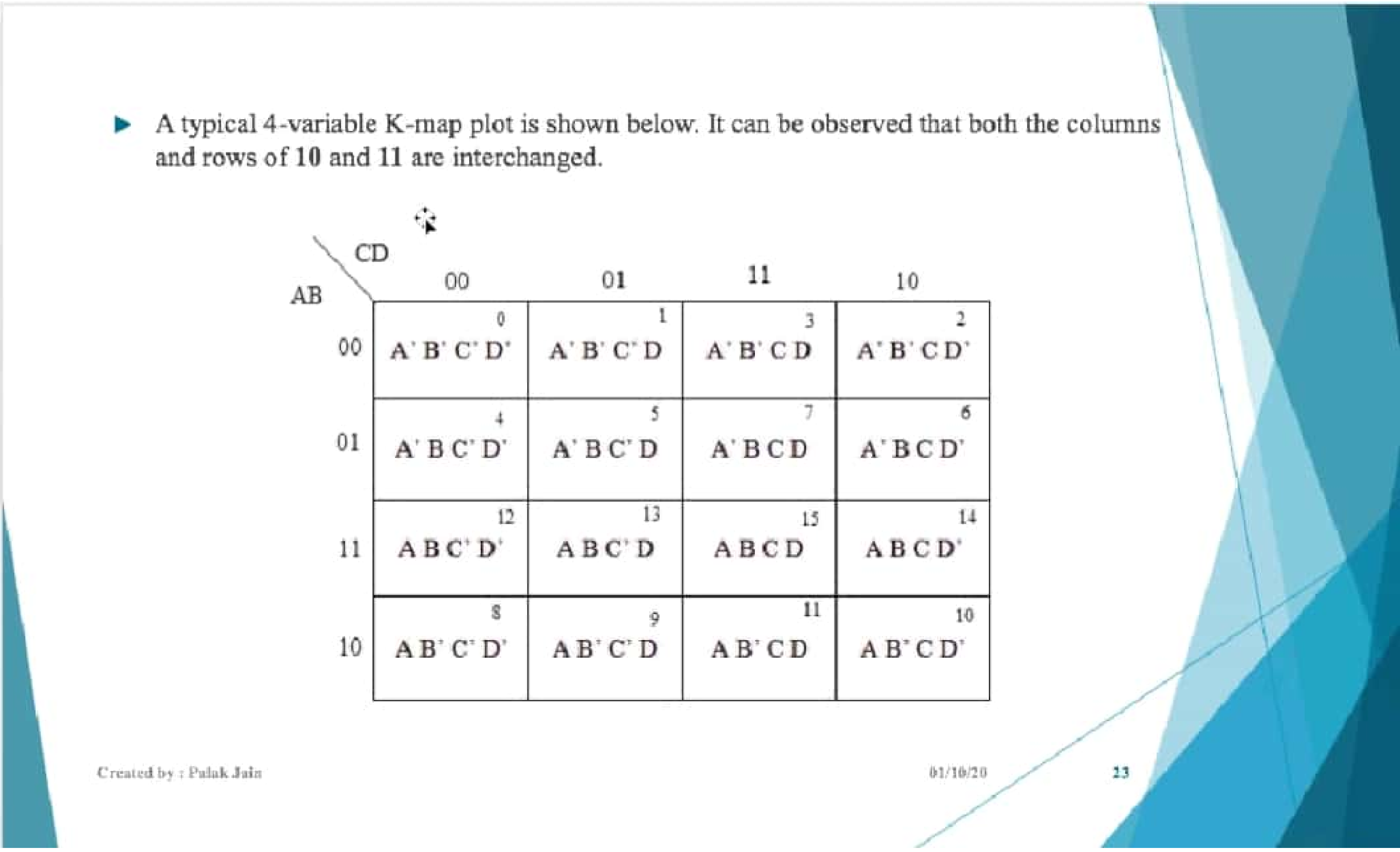
$$= Y' + XZ' + X'Z$$



4-Variable K-Map

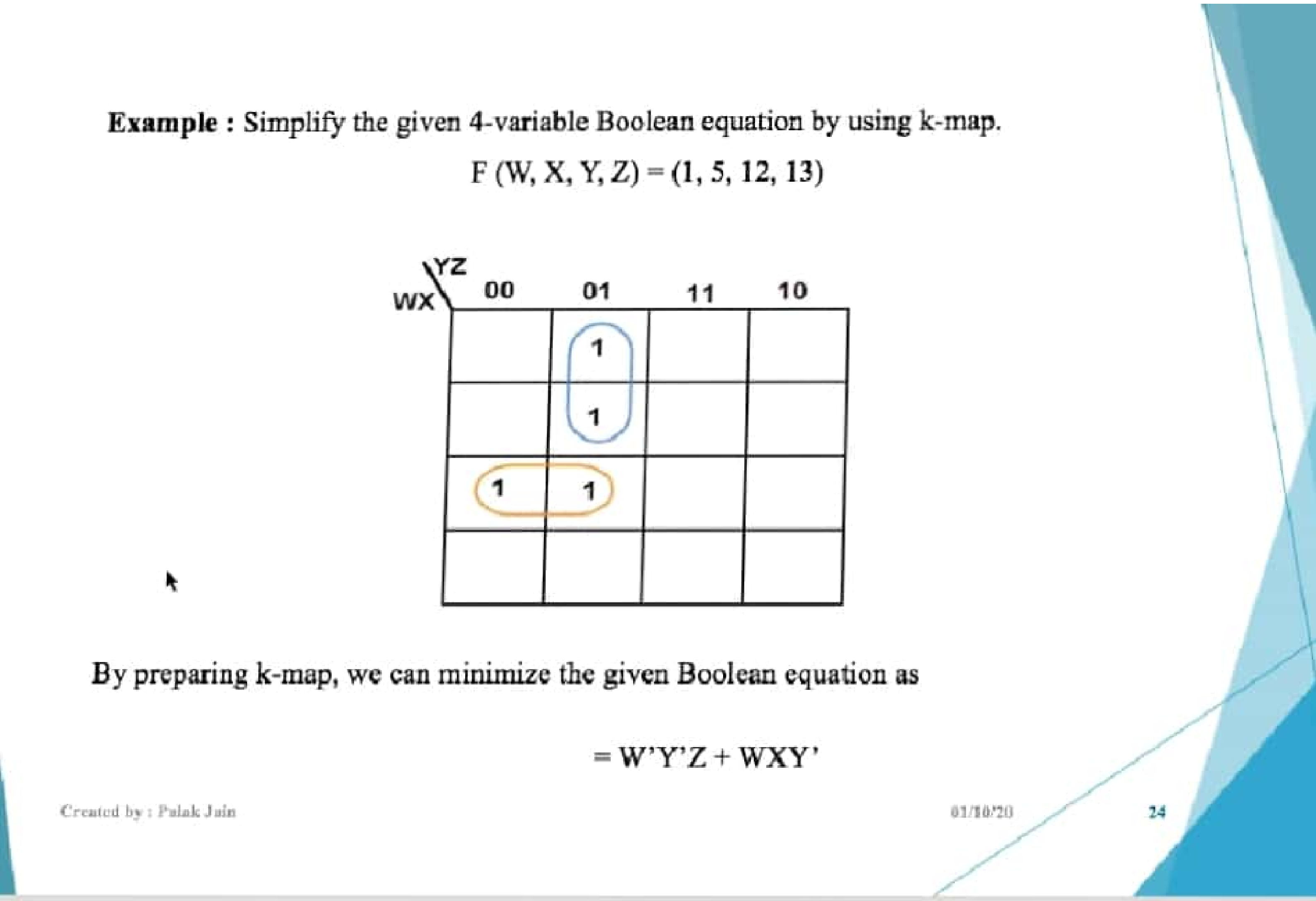
There are 16 possible min terms in case of a 4-variable Boolean function. The general representation of minterms using 4 variables is shown below.

A	B	C	D	Output Function	K-Map Location
0	0	0	0	A'B'C"D'	0
0	0	0	1	A'B'C'D	1
0	0	1	0	A'B'CD'	2
0	0	1	1	A'B'CD	3
0	1	0	0	A'BC'D'	4
0	1	0	1	A'BC"D	5
0	1	1	0	A'BCD'	6
0	1	1	1	A'BCD	7
1	0	0	0	AB'C'D'	8
1	0	0	1	AB'C'D	9
1	0	1	0	AB'CD'	10
1	0	1	1	AB'CD	11
1	1	0	0	ABC'D'	12
1	1	0	1	ABC'D	13
1	1	1	0	ABCD'	14
1	1	1	1	ABCD	15



- A typical 4-variable K-map plot is shown below. It can be observed that both the columns and rows of **10** and **11** are interchanged.

AB \ CD	00	01	11	10
00	0 A' B' C' D'	1 A' B' C' D	3 A' B' C D	2 A' B' C D'
01	4 A' B C' D'	5 A' B C' D	7 A' B C D	6 A' B C D'
11	12 A B C' D'	13 A B C' D	15 A B C D	14 A B C D'
10	8 A B' C' D'	9 A B' C' D	11 A B' C D	10 A B' C D'



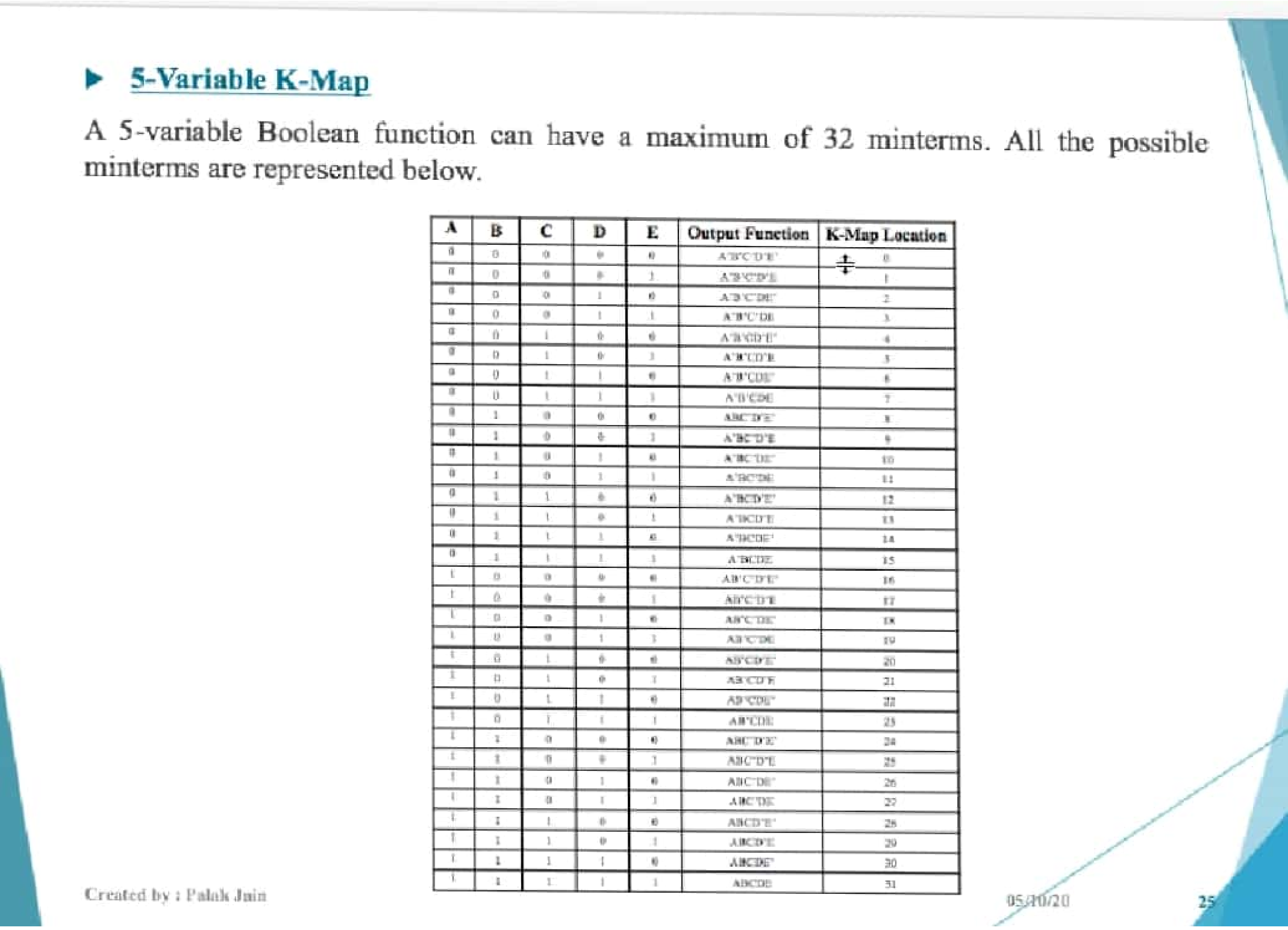
Example : Simplify the given 4-variable Boolean equation by using k-map.

$$F (W, X, Y, Z) = (1, 5, 12, 13)$$

WX \ YZ	00	01	11	10
		1		
		1		
	1	1		

By preparing k-map, we can minimize the given Boolean equation as

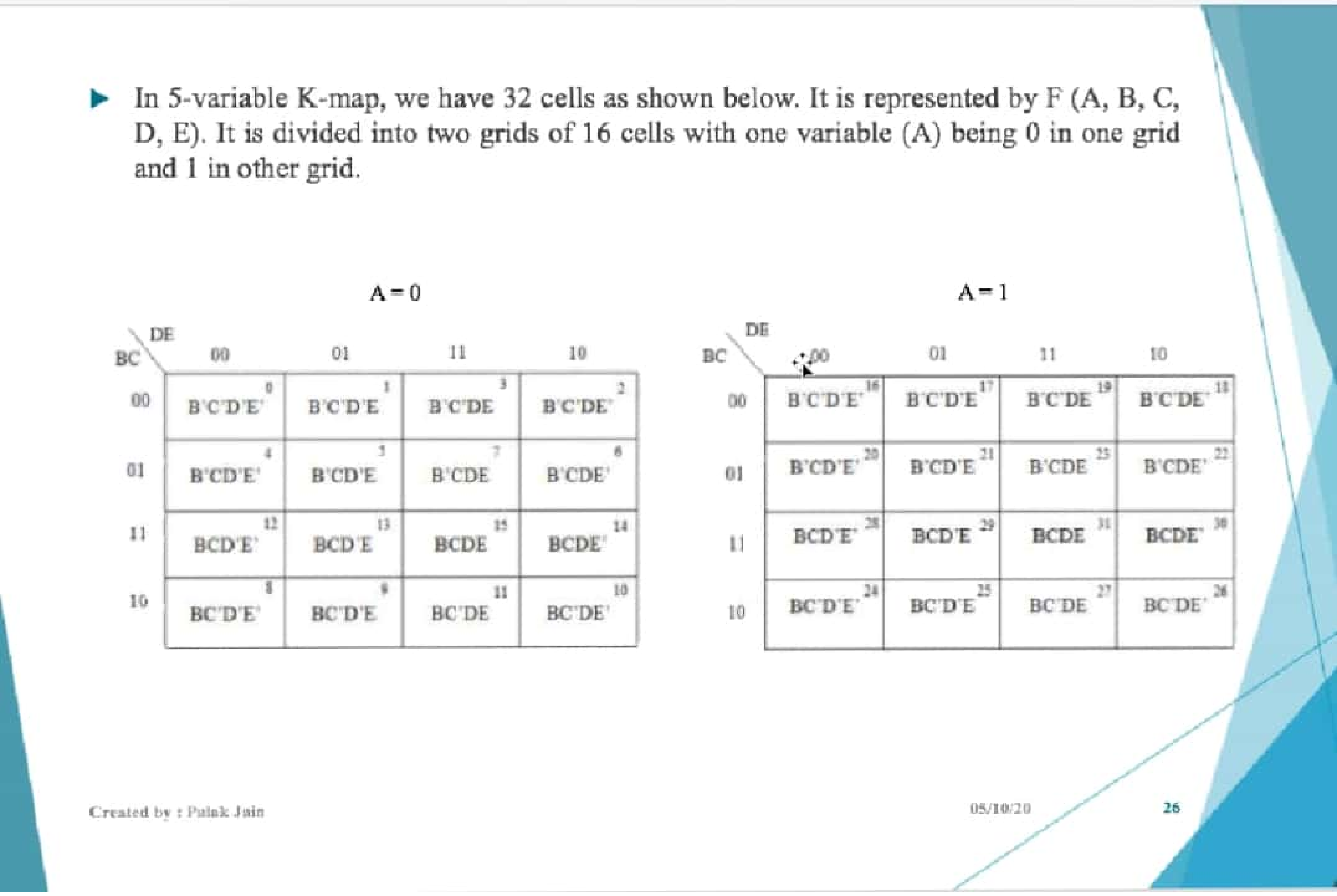
$$= W'Y'Z + WXY'$$



5-Variable K-Map

A 5-variable Boolean function can have a maximum of 32 minterms. All the possible minterms are represented below.

A	B	C	D	E	Output Function	K-Map Location
0	0	0	0	0	A'B'C'D'E'	0
0	0	0	0	1	A'B'C'D'E	1
0	0	0	1	0	A'B'C'DE'	2
0	0	0	1	1	A'B'C'DE	3
0	0	1	0	0	A'B'CD'E'	4
0	0	1	0	1	A'B'CD'E	5
0	0	1	1	0	A'B'CDE'	6
0	0	1	1	1	A'B'CDE	7
0	1	0	0	0	ABC'D'E'	8
0	1	0	0	1	A'BC'D'E	9
0	1	0	1	0	A'BC'DE'	10
0	1	0	1	1	A'BC'DE	11
0	1	1	0	0	A'BCD'E'	12
0	1	1	0	1	A'BCD'E	13
0	1	1	1	0	A'BCDE'	14
0	1	1	1	1	A'BCDE	15
1	0	0	0	0	AB'C'D'E'	16
1	0	0	0	1	AB'C'D'E	17
1	0	0	1	0	AB'C'DE'	18
1	0	0	1	1	AB'C'DE	19
1	0	1	0	0	AB'CD'E'	20
1	0	1	0	1	AB'CD'E	21
1	0	1	1	0	AB'CDE'	22
1	0	1	1	1	AB'CDE	23
1	1	0	0	0	ABC'D'E'	24
1	1	0	0	1	ABC'D'E	25
1	1	0	1	0	ABC'DE'	26
1	1	0	1	1	ABC'DE	27
1	1	1	0	0	ABCD'E'	28
1	1	1	0	1	ABCD'E	29
1	1	1	1	0	ABCDE'	30
1	1	1	1	1	ABCDE	31



- In 5-variable K-map, we have 32 cells as shown below. It is represented by F (A, B, C, D, E). It is divided into two grids of 16 cells with one variable (A) being 0 in one grid and 1 in other grid.

A = 0

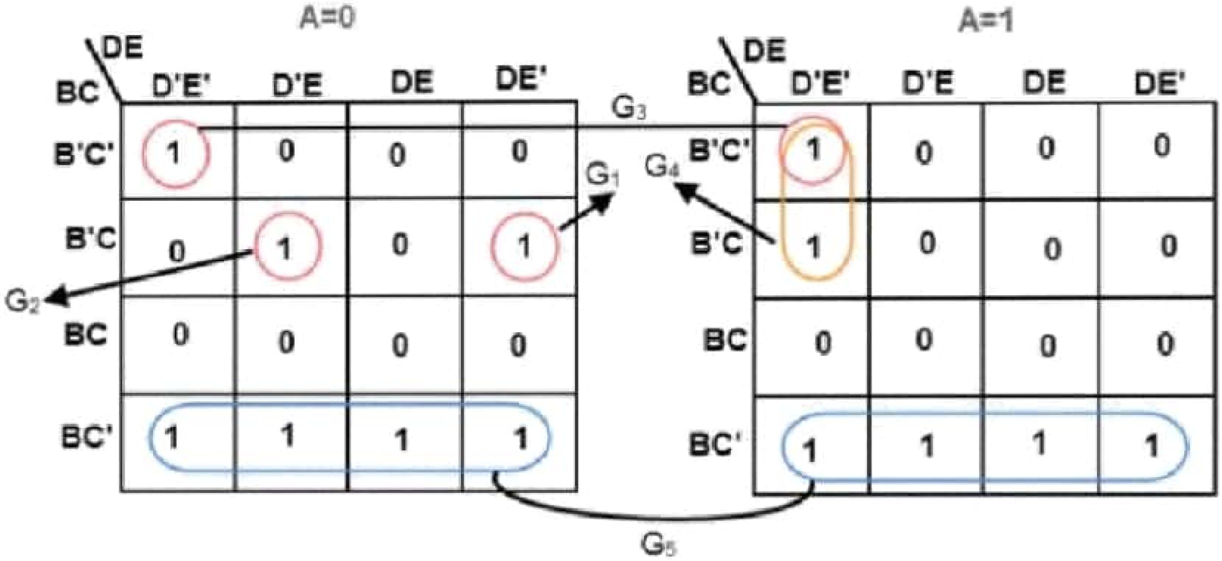
BC \ DE	00	01	11	10
00	0 B'C'D'E'	1 B'C'D'E	3 B'C'DE	2 B'C'DE'
01	4 B'CD'E'	5 B'CD'E	7 B'CDE	6 B'CDE'
11	12 BCD'E'	13 BCD'E	15 BCDE	14 BCDE'
10	8 BC'D'E'	9 BC'D'E	11 BC'DE	10 BC'DE'

A = 1

BC \ DE	00	01	11	10
00	16 B'C'D'E'	17 B'C'D'E	19 B'C'DE	18 B'C'DE'
01	20 B'CD'E'	21 B'CD'E	23 B'CDE	22 B'CDE'
11	28 BCD'E'	29 BCD'E	31 BCDE	30 BCDE'
10	24 BC'D'E'	25 BC'D'E	27 BC'DE	26 BC'DE'

Example : Simplify the given 5-variable Boolean equation by using k-map.

$$f(A, B, C, D, E) = \sum m\,(0, 5, 6, 8, 9, 10, 11, 16, 20, 42, 25, 26, 27)$$

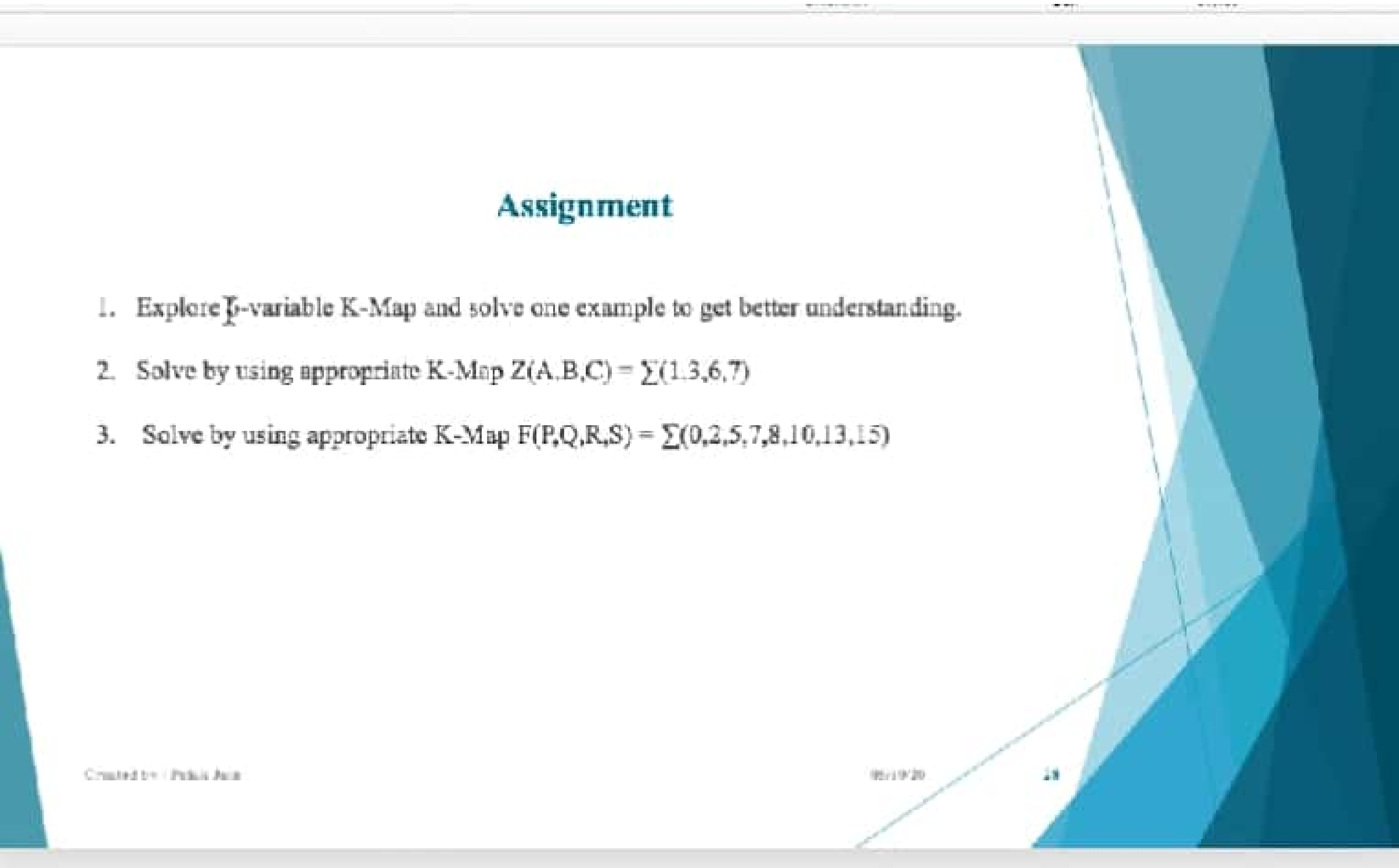


A=0

BC \ DE	D'E'	D'E	DE	DE'
B'C'	1	0	0	0
B'C	0	1	0	1
BC	0	0	0	0
BC'	1	1	1	1

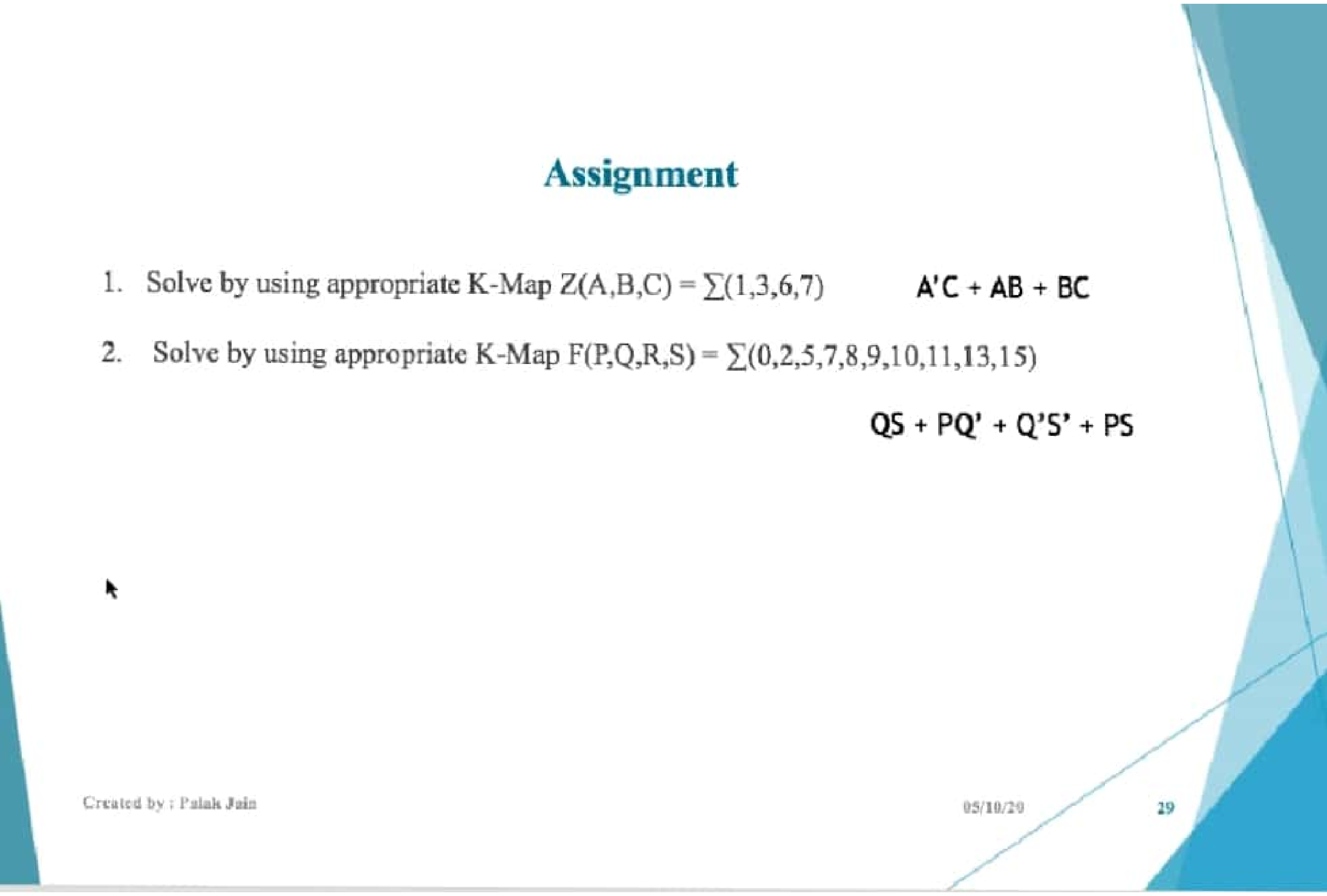
A=1

BC \ DE	D'E'	D'E	DE	DE'
B'C'	1	0	0	0
B'C	1	0	0	0
BC	0	0	0	0
BC'	1	1	1	1



Assignment

1. Explore 5-variable K-Map and solve one example to get better understanding.
2. Solve by using appropriate K-Map Z(A,B,C) = ∑(1,3,6,7)
3. Solve by using appropriate K-Map F(P,Q,R,S) = ∑(0,2,5,7,8,10,13,15)

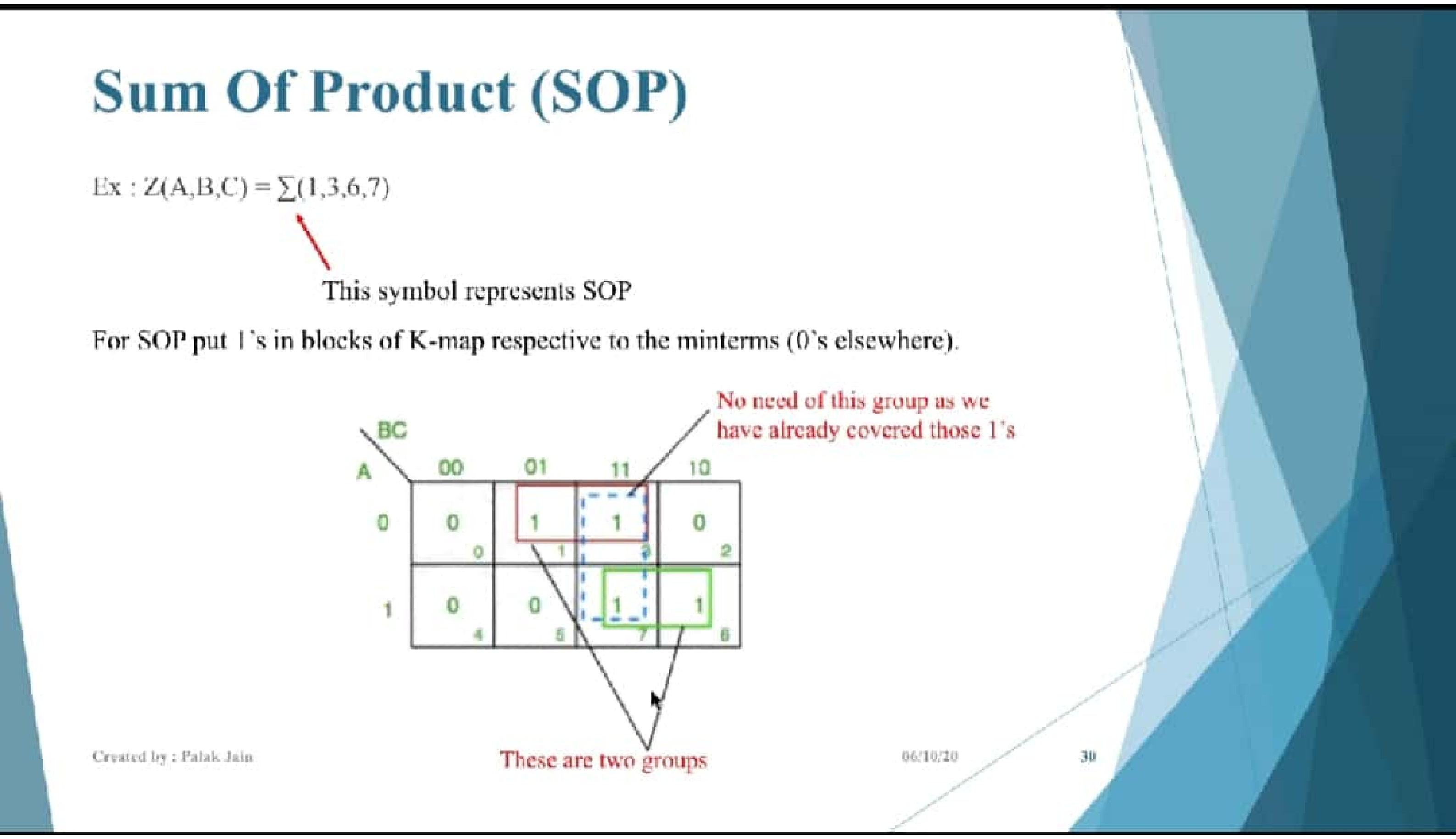


Assignment

1. Solve by using appropriate K-Map $Z(A,B,C) = \sum(1,3,6,7)$ $A'C + AB + BC$

2. Solve by using appropriate K-Map $F(P,Q,R,S) = \sum(0,2,5,7,8,9,10,11,13,15)$

$QS + PQ' + Q'S' + PS$



Sum Of Product (SOP)

Ex : Z(A,B,C) = ∑(1,3,6,7)

This symbol represents SOP

For SOP put 1's in blocks of K-map respective to the minterms (0's elsewhere).

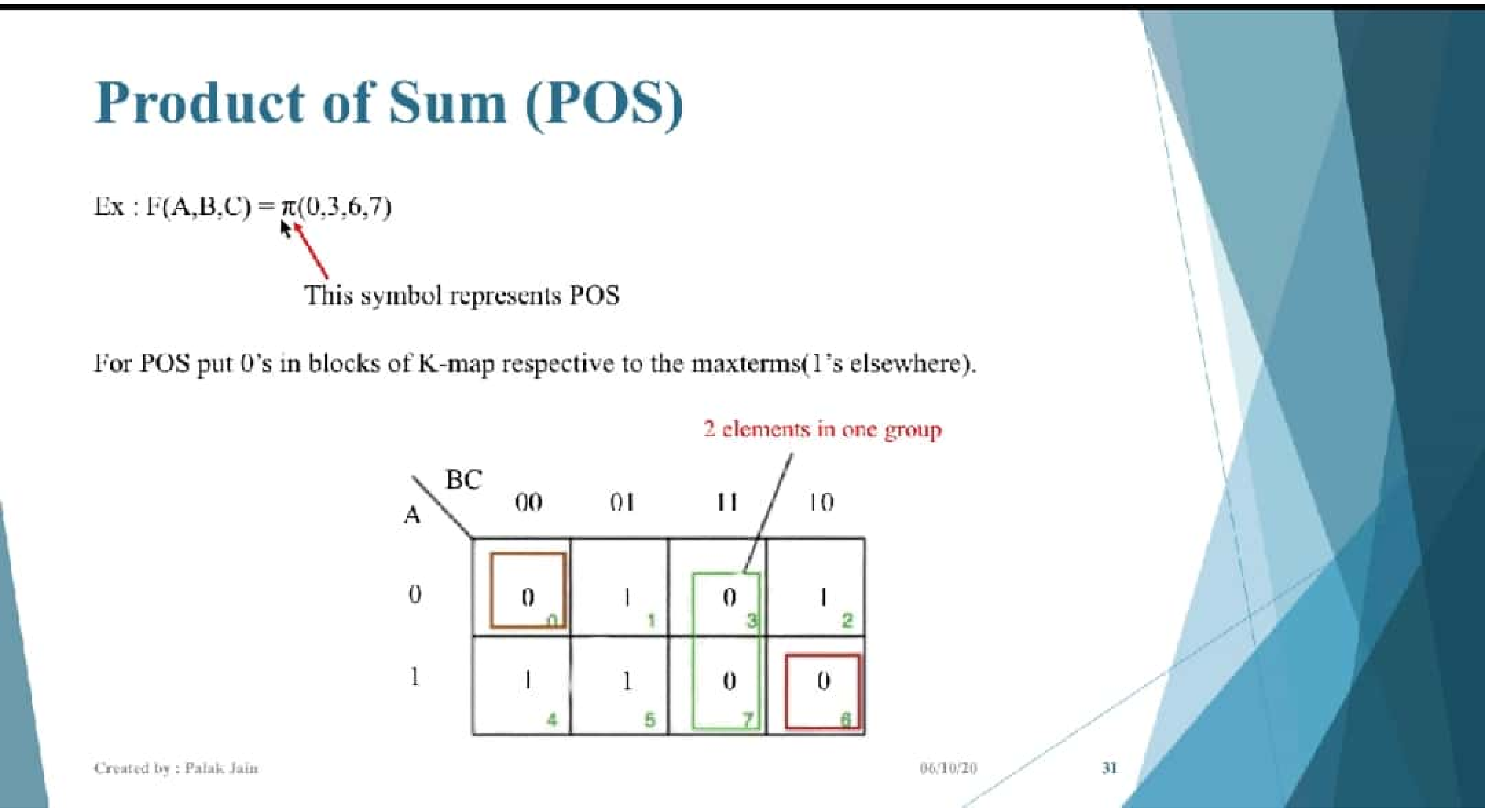
BC A	00	01	11	10
0	0 0	1 1	1 3	0 2
1	0 4	0 5	1 7	1 6

No need of this group as we have already covered those 1's

These are two groups

Created by : Palak Jain

06/10/20



Product of Sum (POS)

Ex : F(A,B,C) = π(0,3,6,7)

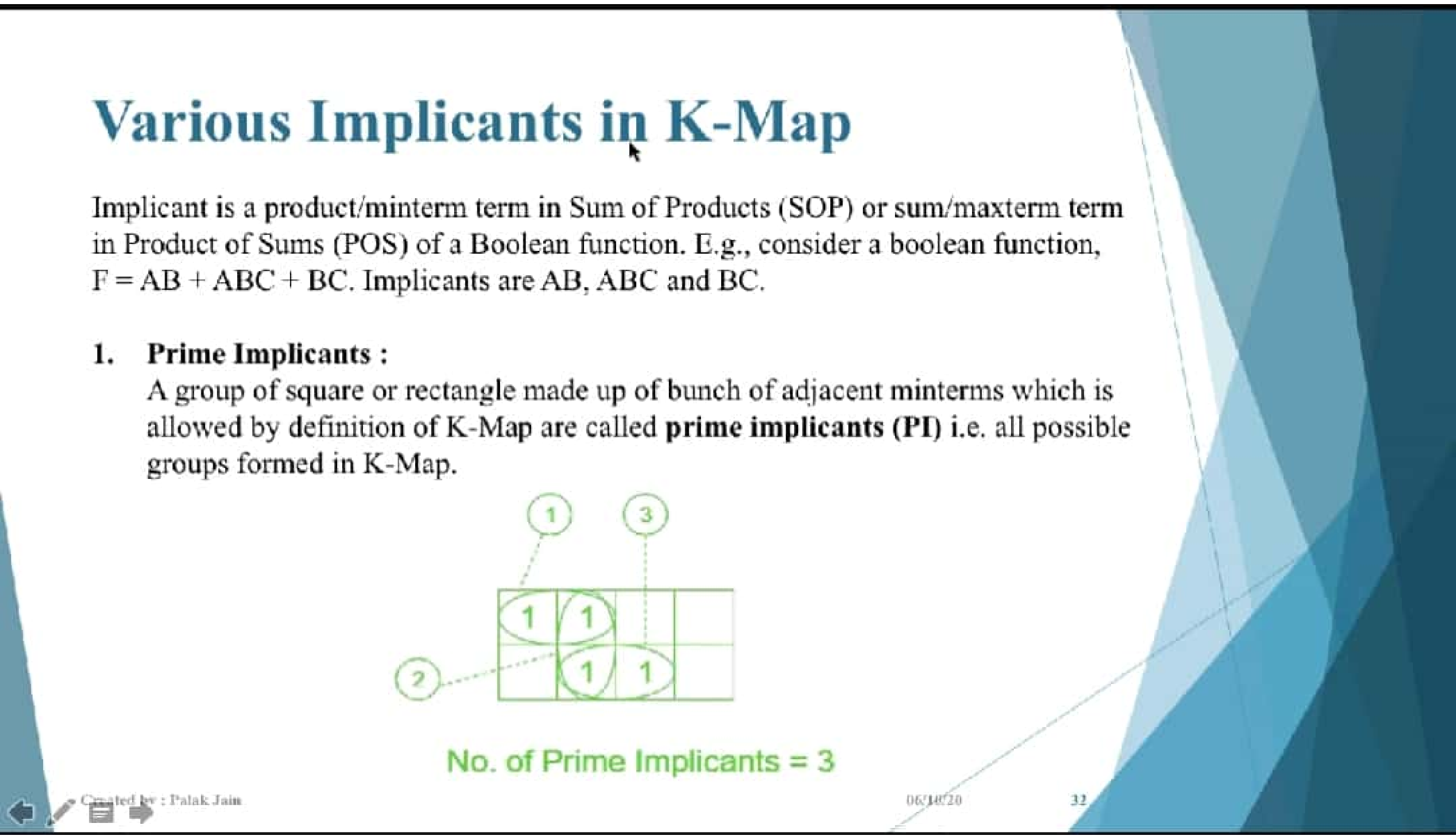
This symbol represents POS

For POS put 0's in blocks of K-map respective to the maxterms(1's elsewhere).

2 elements in one group

A \ BC	00	01	11	10
0	0 0	1 1	0 3	1 2
1	1 4	1 5	0 7	0 6

Created by : Palak Jain

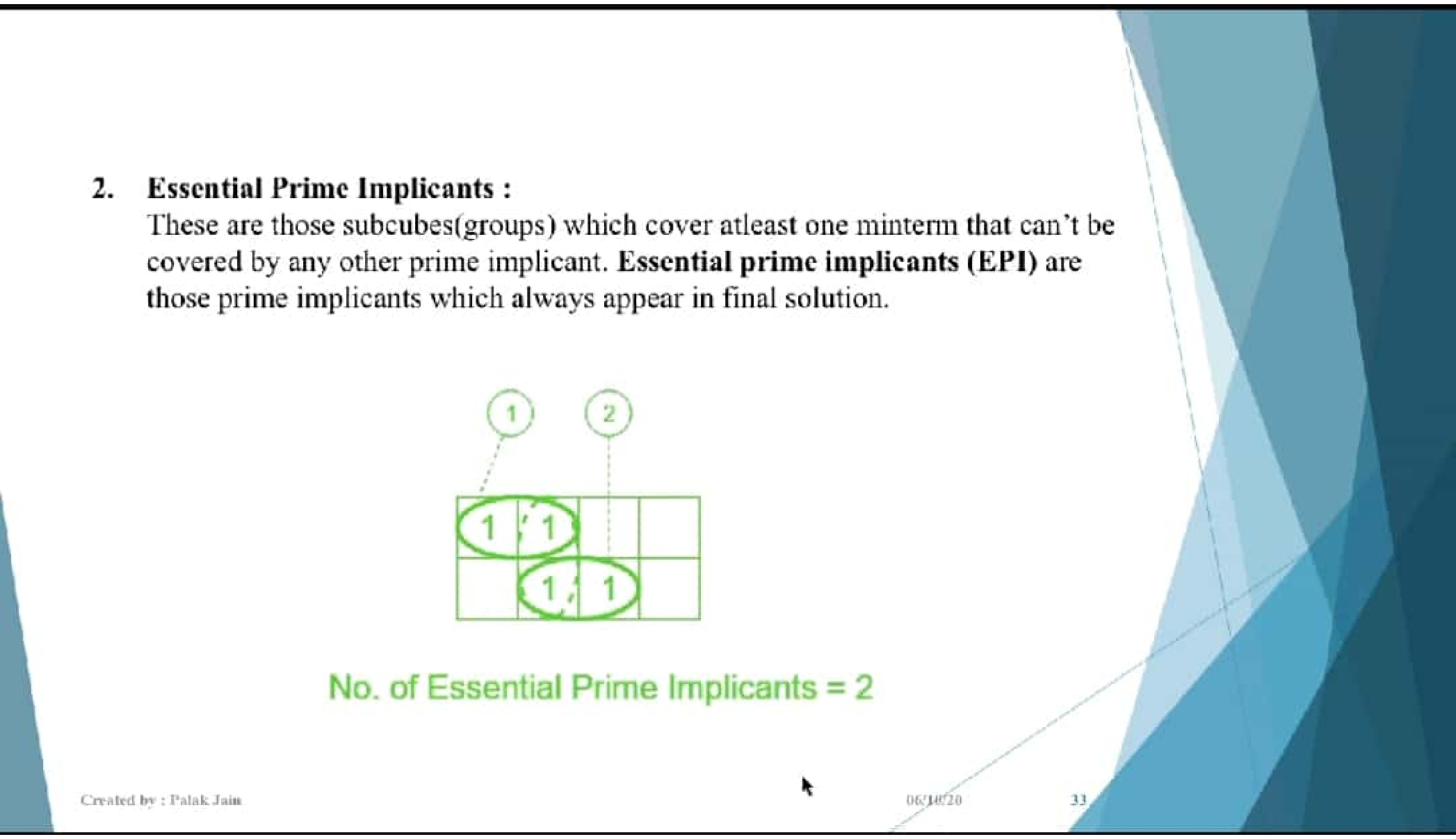


Various Implicants in K-Map

Implicant is a product/minterm term in Sum of Products (SOP) or sum/maxterm term in Product of Sums (POS) of a Boolean function. E.g., consider a boolean function, F = AB + ABC + BC. Implicants are AB, ABC and BC.

1. **Prime Implicants :**
   A group of square or rectangle made up of bunch of adjacent minterms which is allowed by definition of K-Map are called **prime implicants (PI)** i.e. all possible groups formed in K-Map.

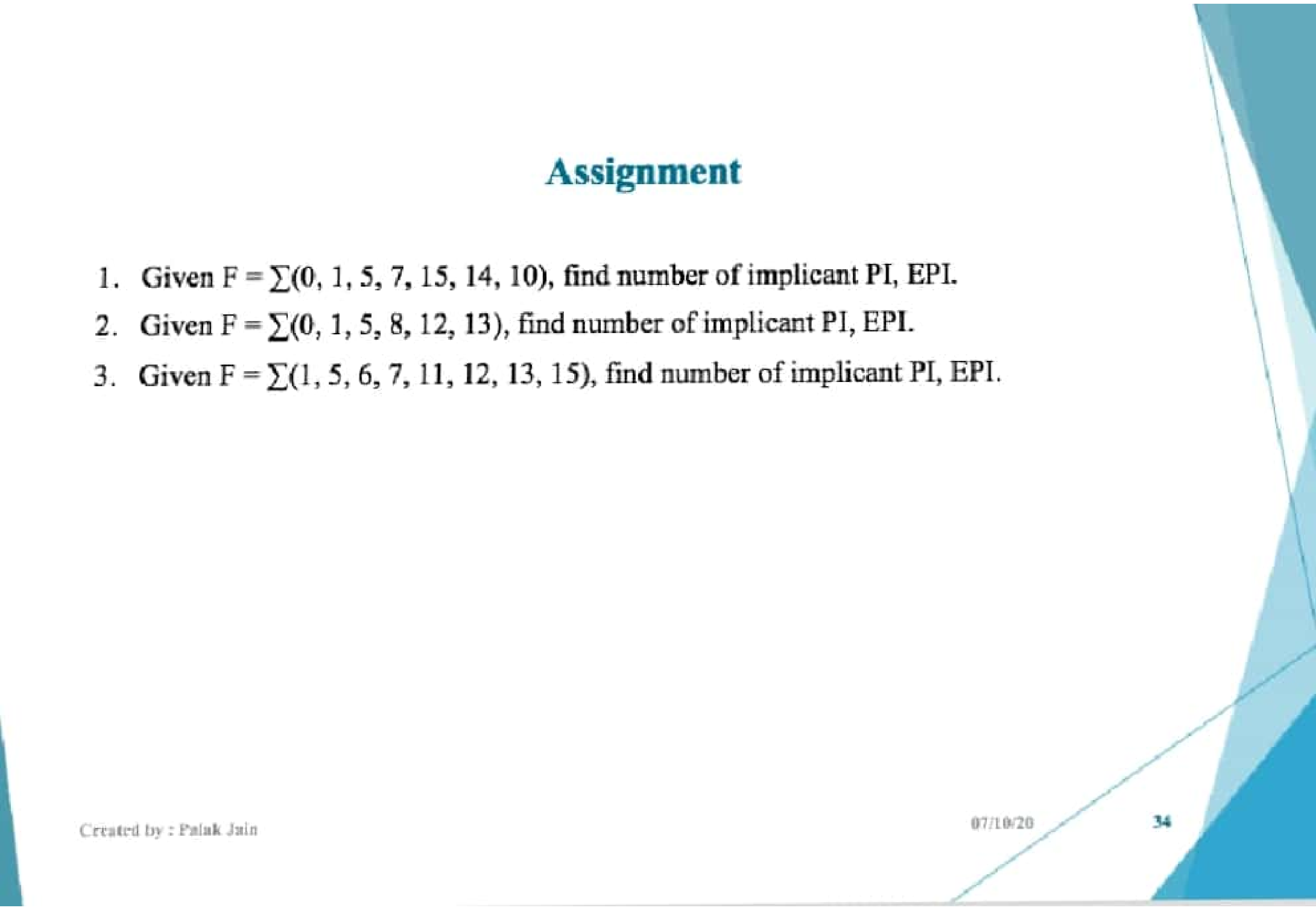
No. of Prime Implicants = 3



2. **Essential Prime Implicants :**
These are those subcubes(groups) which cover atleast one minterm that can't be covered by any other prime implicant. **Essential prime implicants (EPI)** are those prime implicants which always appear in final solution.

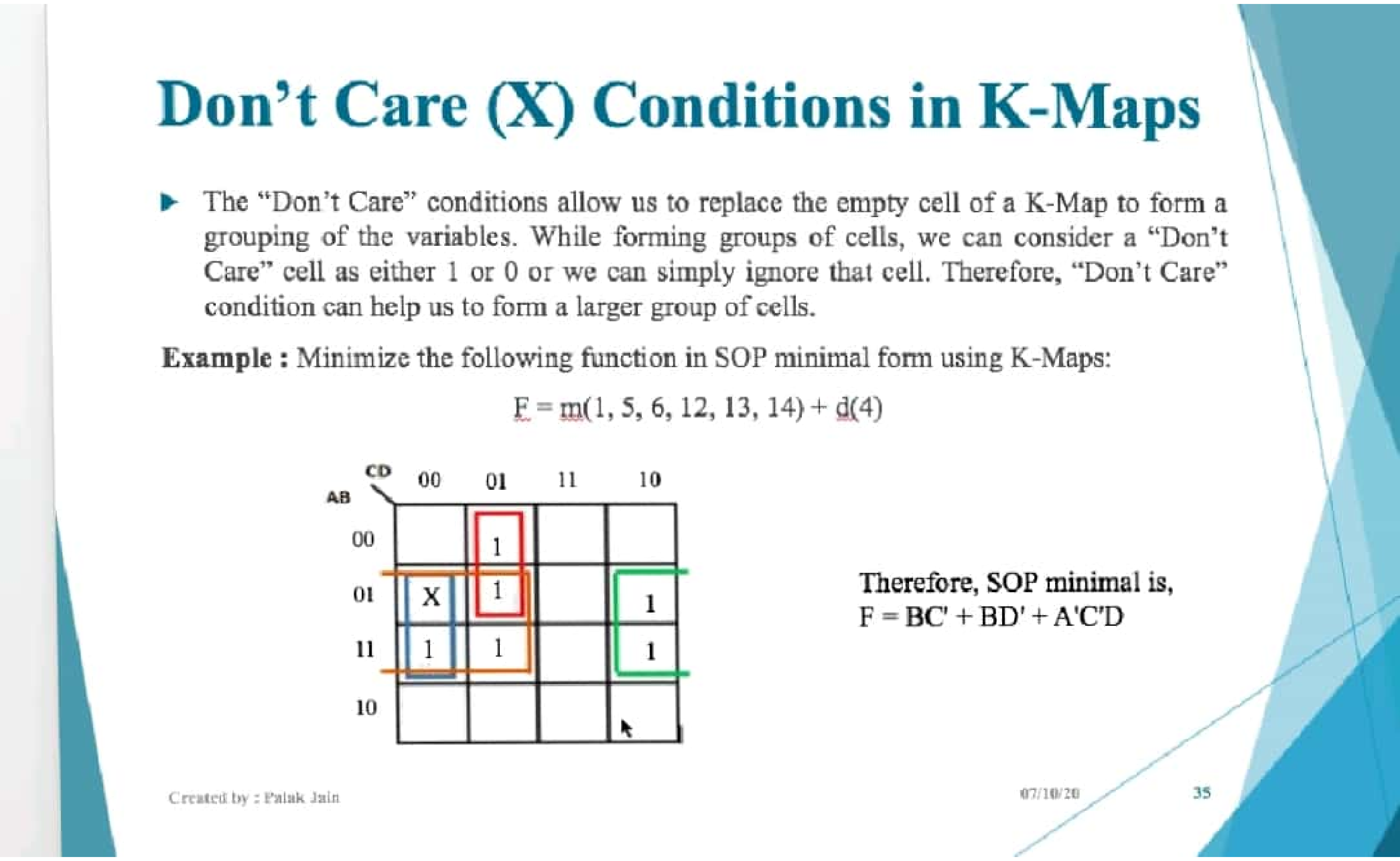
1	1		
	1	1	

No. of Essential Prime Implicants = 2



Assignment

1. Given F = ∑(0, 1, 5, 7, 15, 14, 10), find number of implicant PI, EPI.
2. Given F = ∑(0, 1, 5, 8, 12, 13), find number of implicant PI, EPI.
3. Given F = ∑(1, 5, 6, 7, 11, 12, 13, 15), find number of implicant PI, EPI.



Don’t Care (X) Conditions in K-Maps

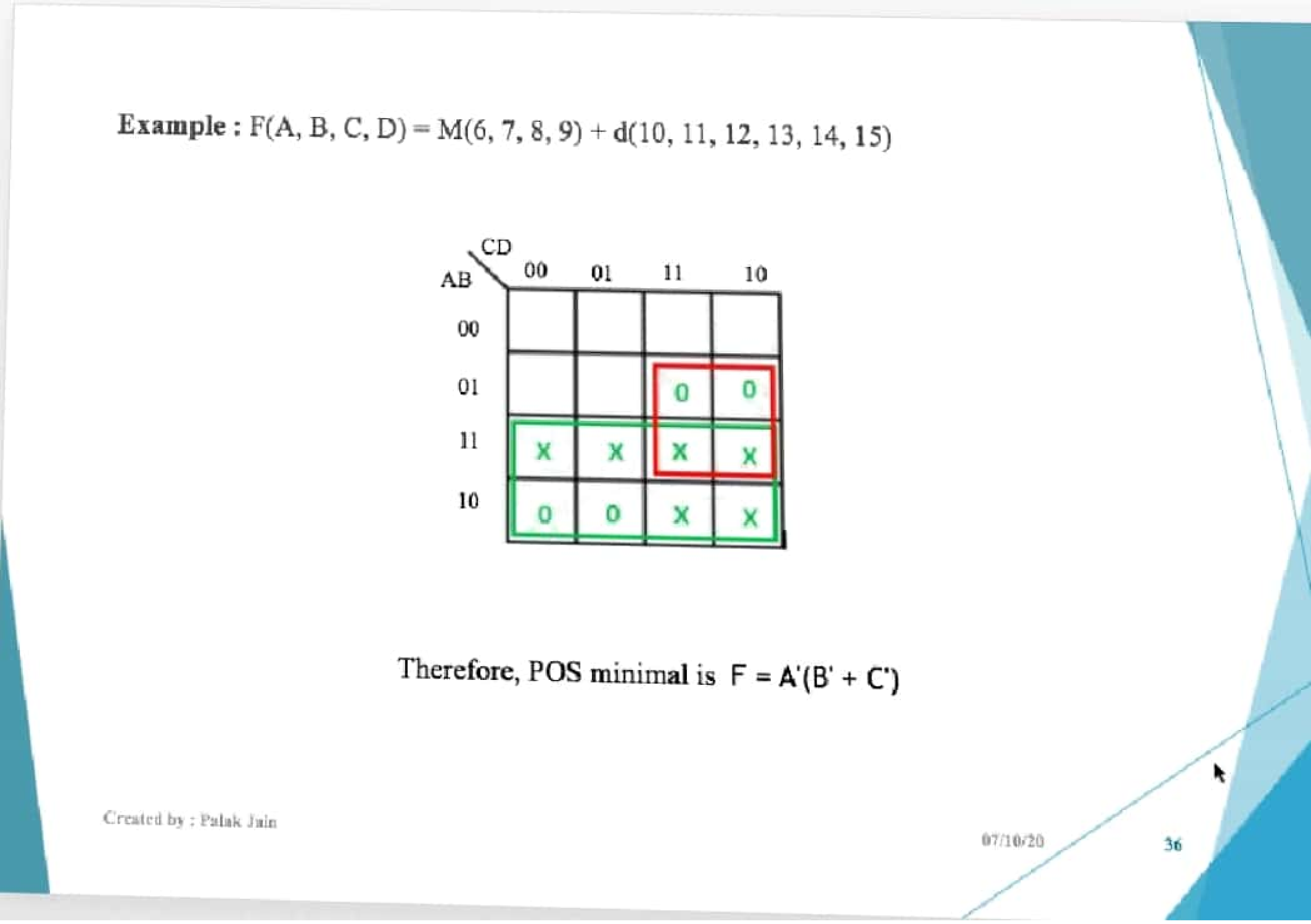
- The “Don’t Care” conditions allow us to replace the empty cell of a K-Map to form a grouping of the variables. While forming groups of cells, we can consider a “Don’t Care” cell as either 1 or 0 or we can simply ignore that cell. Therefore, “Don’t Care” condition can help us to form a larger group of cells.

Example : Minimize the following function in SOP minimal form using K-Maps:

$$F = m(1, 5, 6, 12, 13, 14) + d(4)$$

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

Therefore, SOP minimal is,
F = BC' + BD' + A'C'D



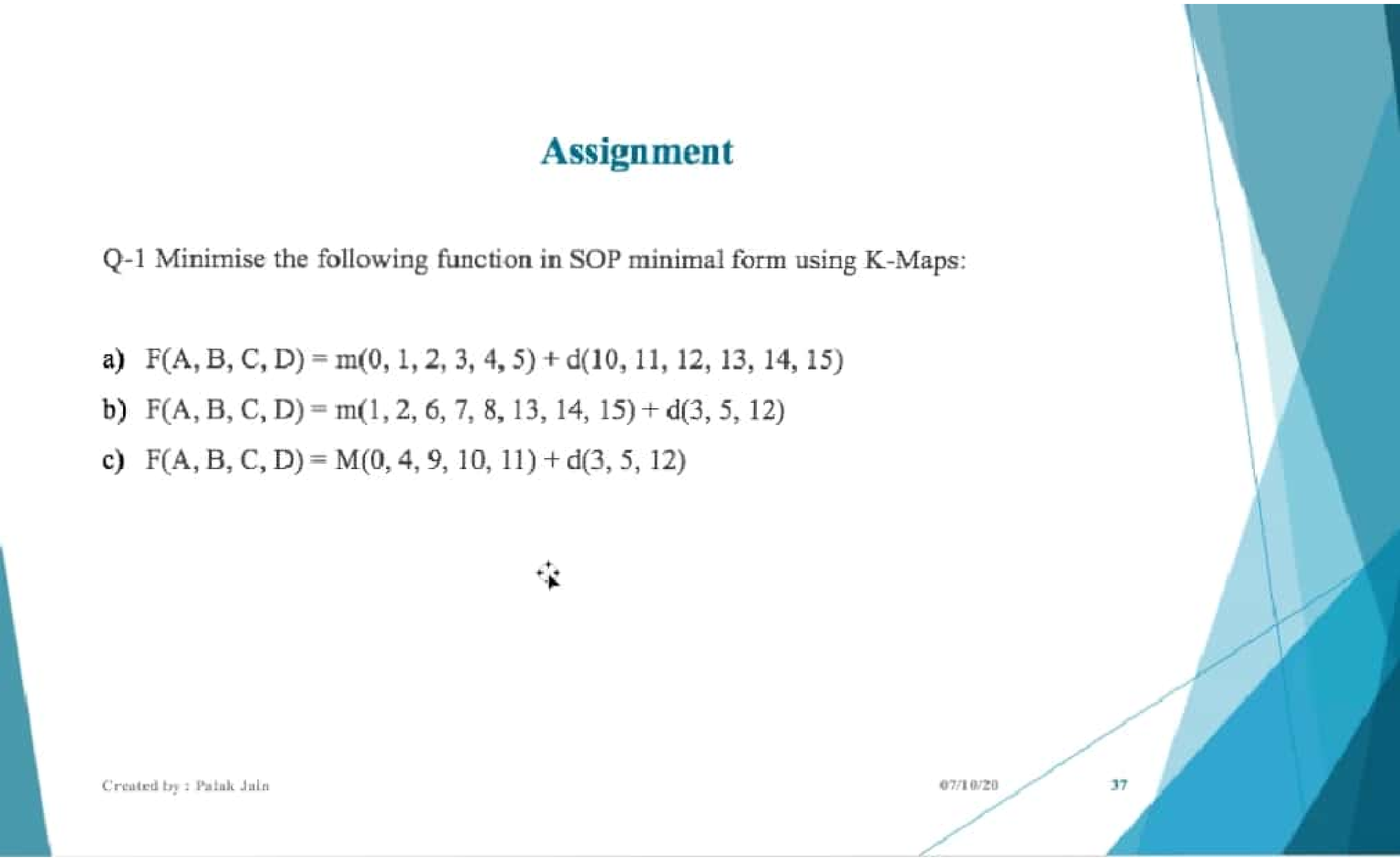
Example : $F(A, B, C, D) = M(6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)$

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

Therefore, POS minimal is $F = A'(B' + C')$

Created by : Palak Jain

07/10/20



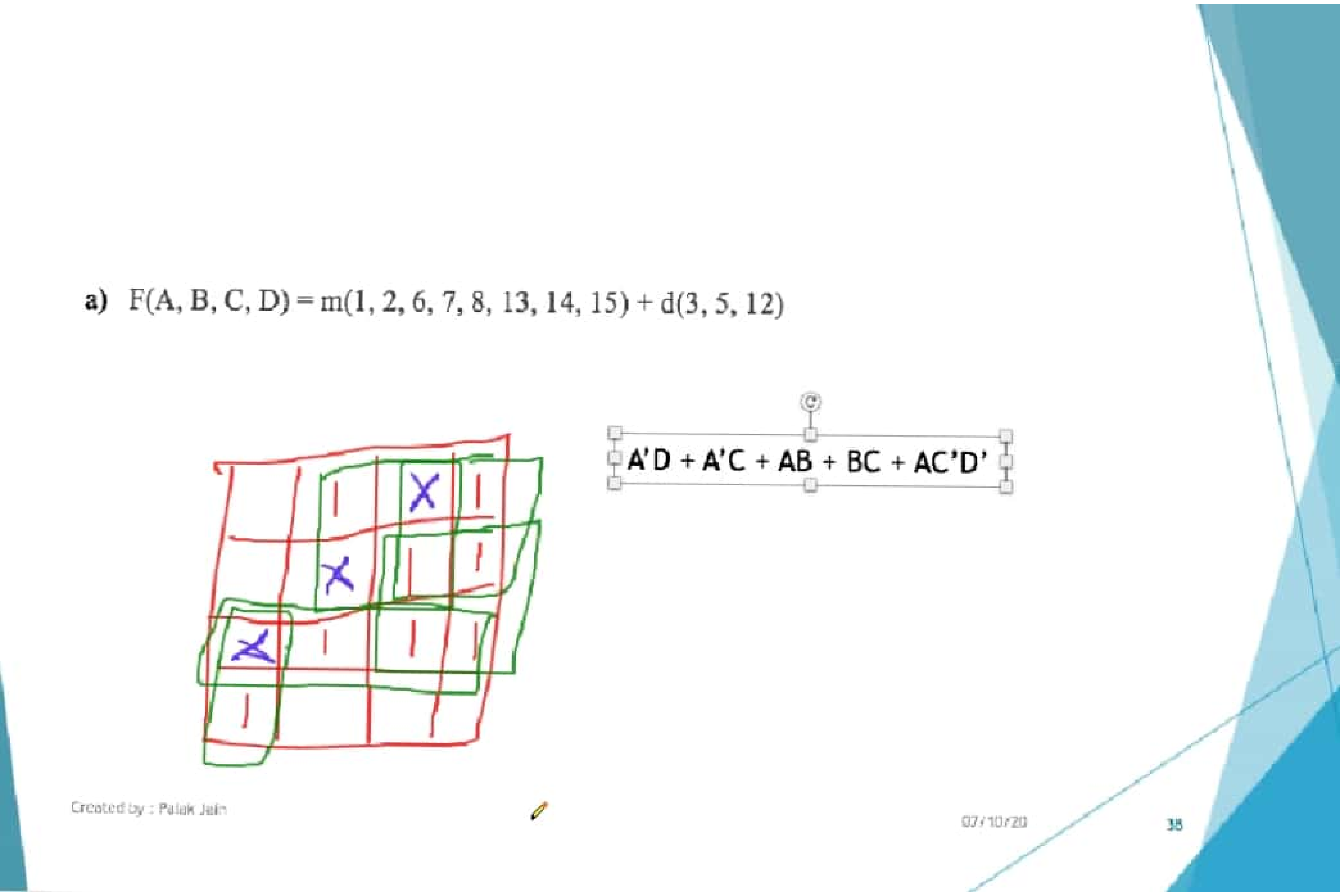
Assignment

Q-1 Minimise the following function in SOP minimal form using K-Maps:

a) F(A, B, C, D) = m(0, 1, 2, 3, 4, 5) + d(10, 11, 12, 13, 14, 15)

b) F(A, B, C, D) = m(1, 2, 6, 7, 8, 13, 14, 15) + d(3, 5, 12)

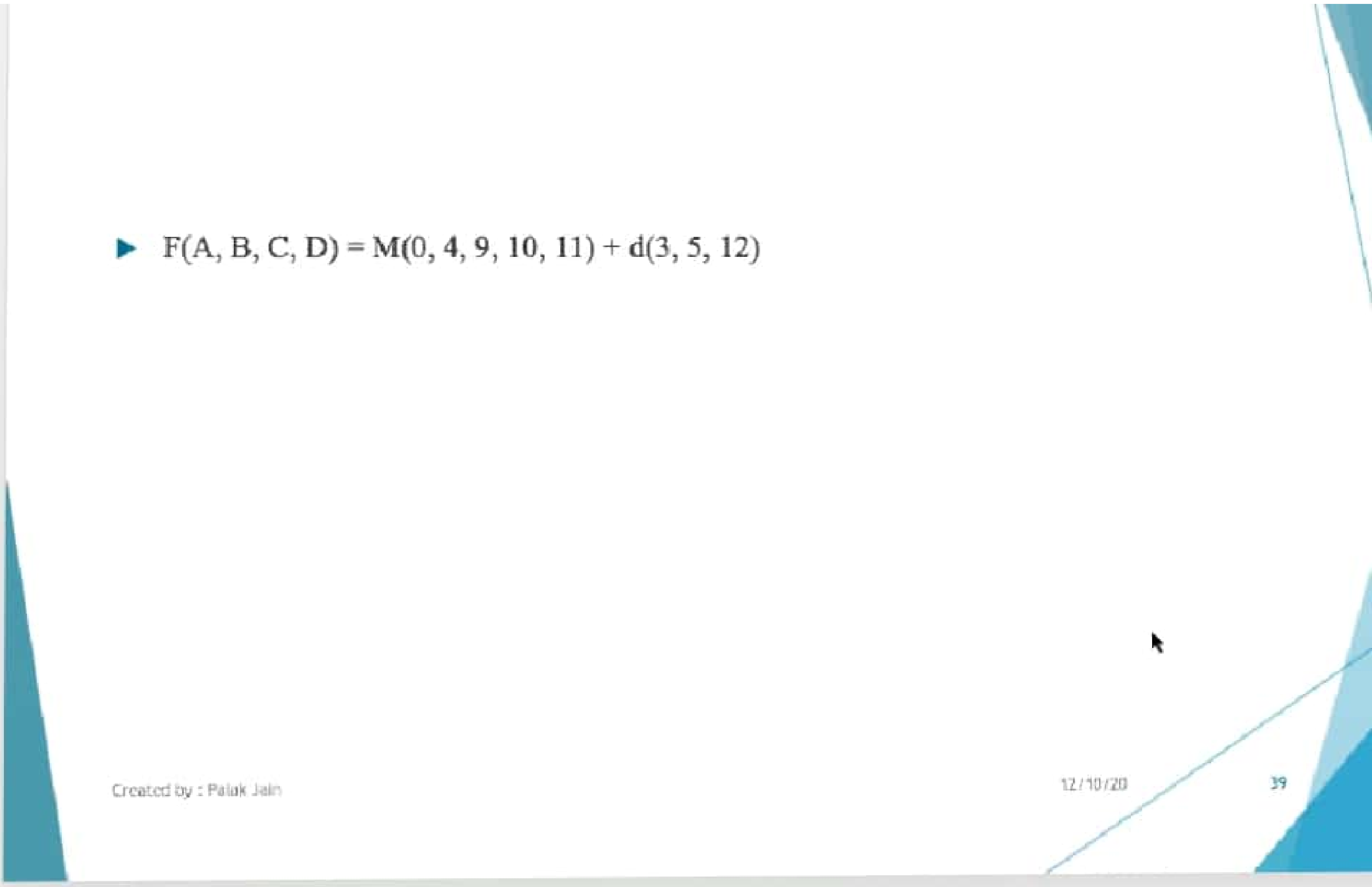
c) F(A, B, C, D) = M(0, 4, 9, 10, 11) + d(3, 5, 12)



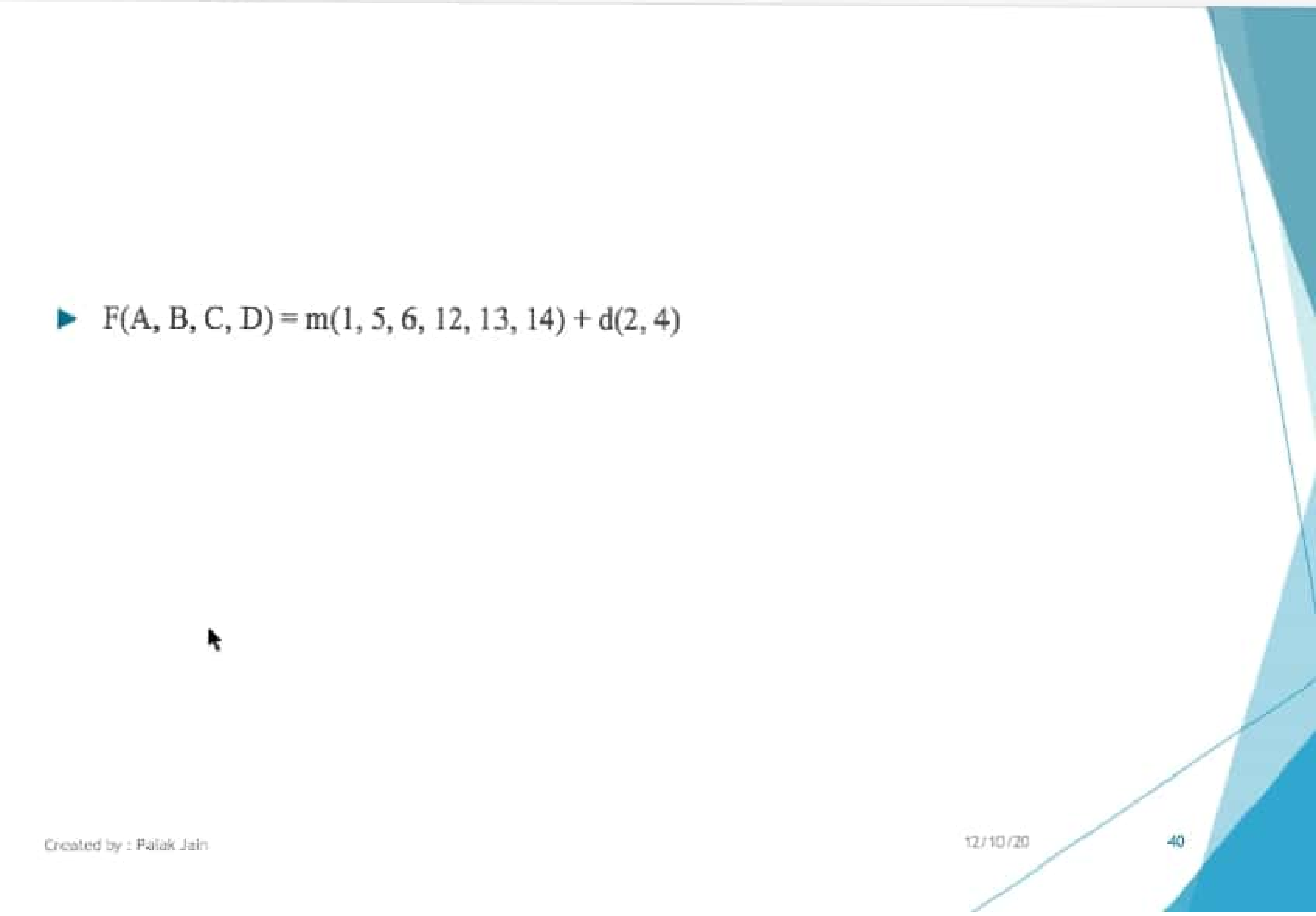
a) F(A, B, C, D) = m(1, 2, 6, 7, 8, 13, 14, 15) + d(3, 5, 12)

	1	X	1
	X	1	1
X	1	1	1
1			

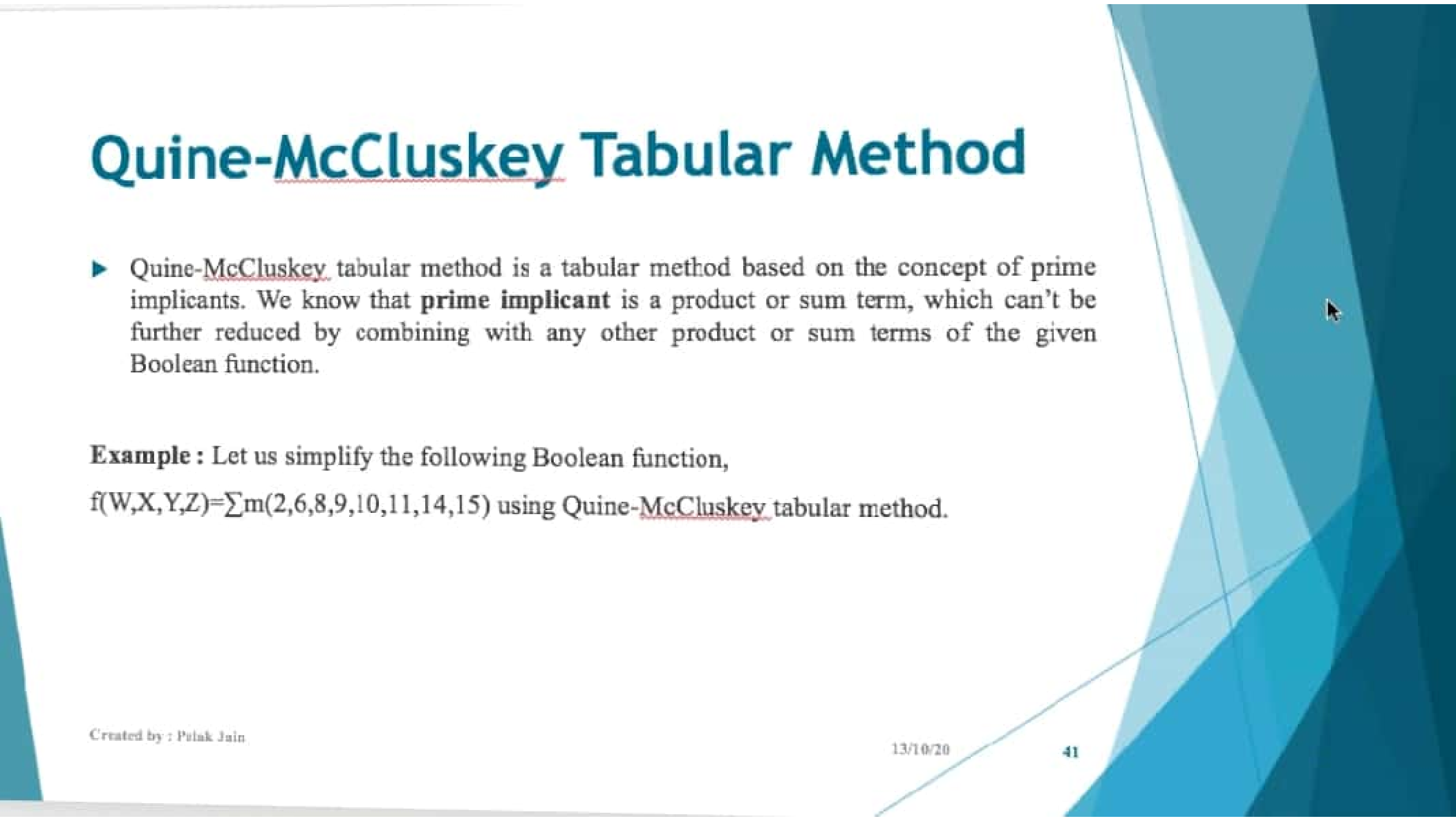
A'D + A'C + AB + BC + AC'D'



- $F(A, B, C, D) = M(0, 4, 9, 10, 11) + d(3, 5, 12)$



- F(A, B, C, D) = m(1, 5, 6, 12, 13, 14) + d(2, 4)

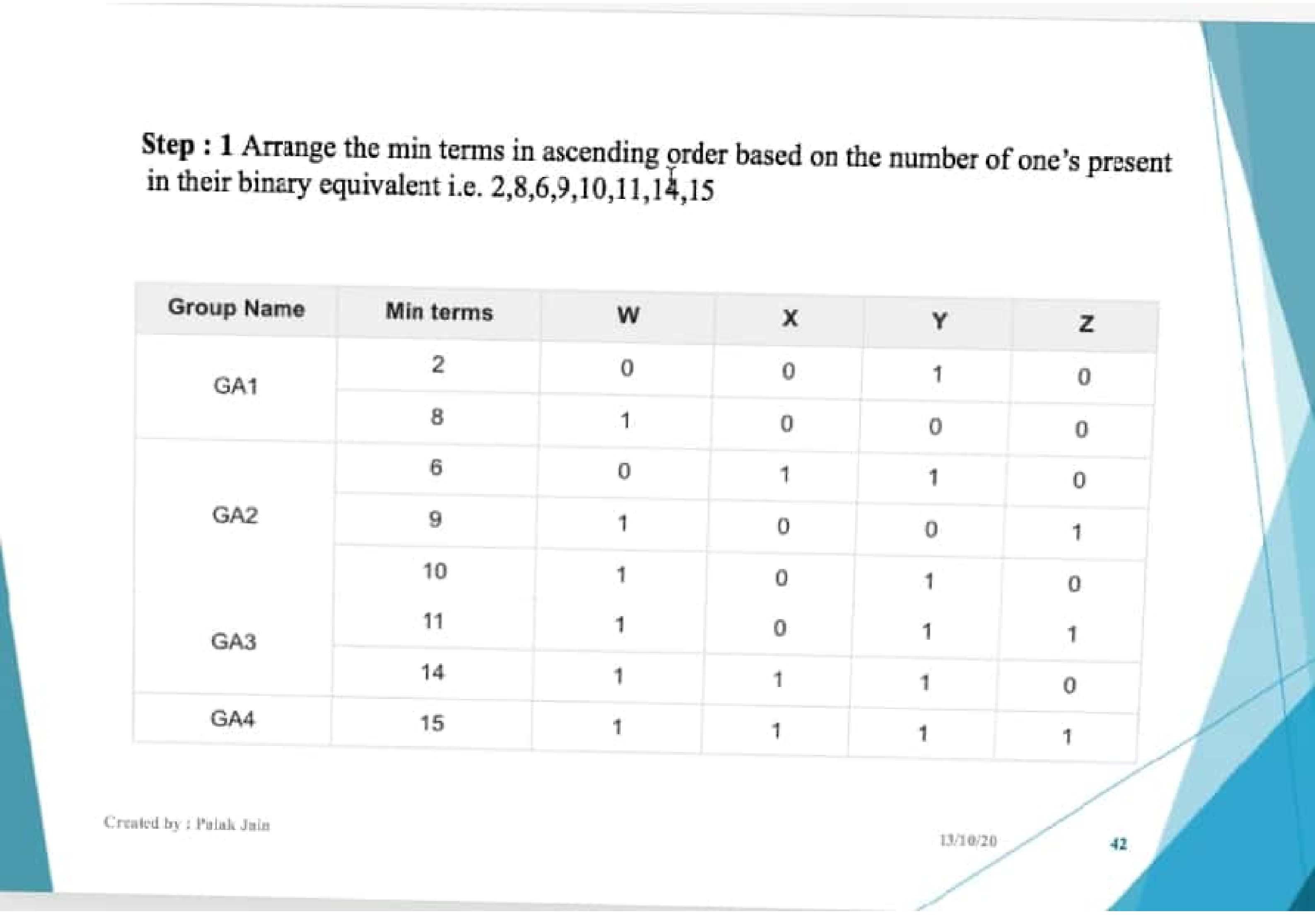


Quine-McCluskey Tabular Method

- Quine-McCluskey tabular method is a tabular method based on the concept of prime implicants. We know that **prime implicant** is a product or sum term, which can't be further reduced by combining with any other product or sum terms of the given Boolean function.

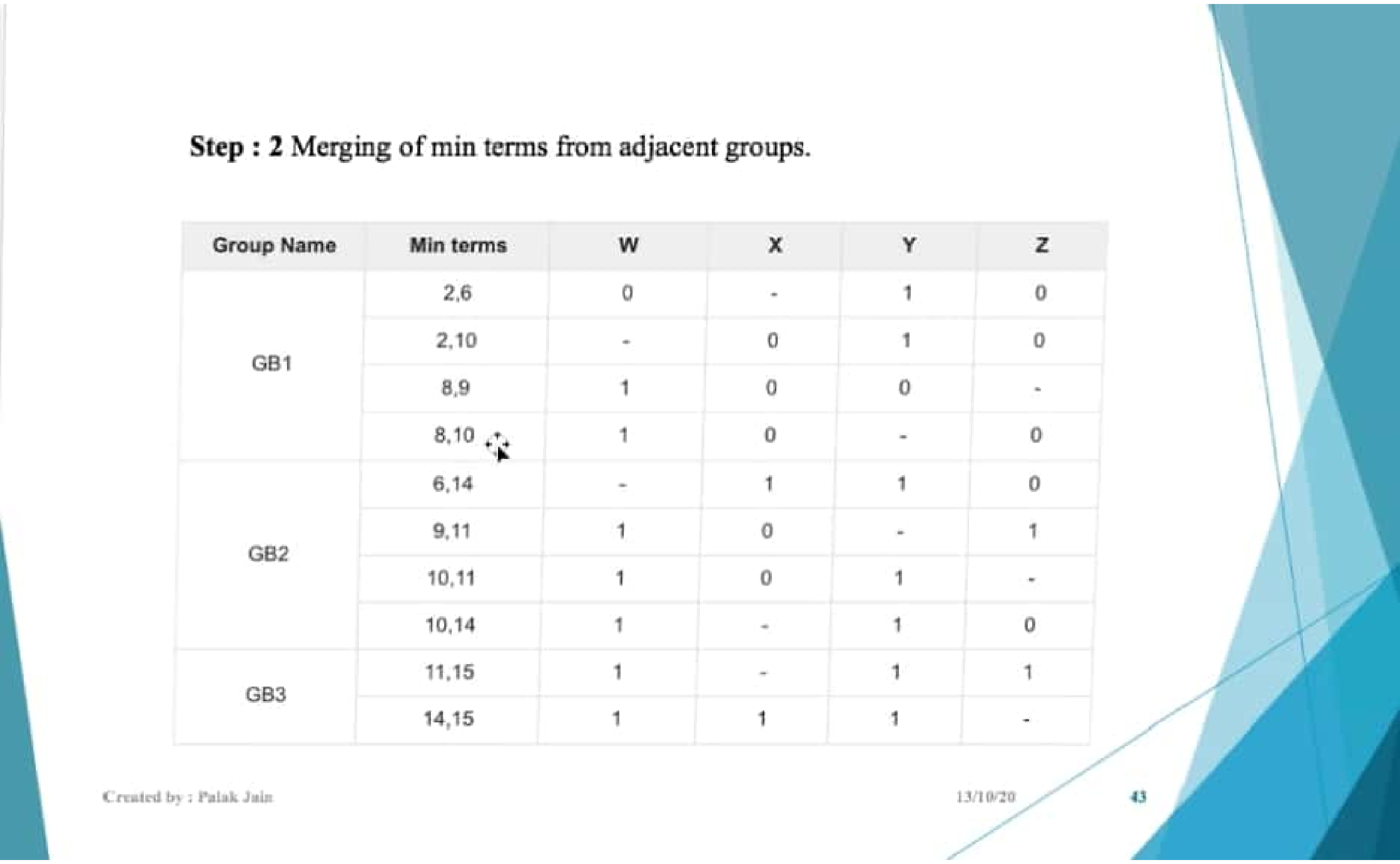
Example : Let us simplify the following Boolean function,

f(W,X,Y,Z)=∑m(2,6,8,9,10,11,14,15) using Quine-McCluskey tabular method.



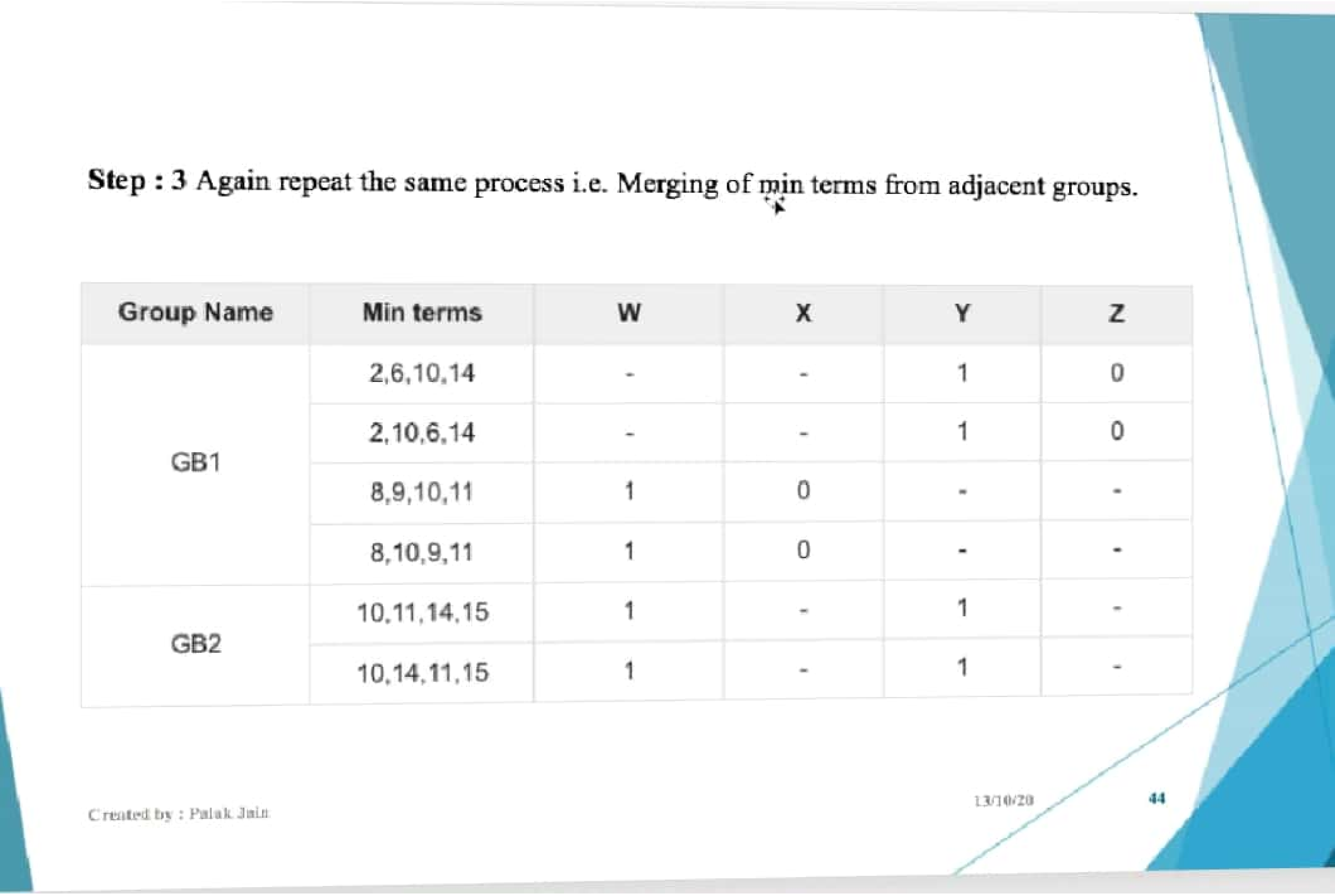
Step : 1 Arrange the min terms in ascending order based on the number of one's present in their binary equivalent i.e. 2,8,6,9,10,11,14,15

Group Name	Min terms	W	X	Y	Z
GA1	2	0	0	1	0
	8	1	0	0	0
GA2	6	0	1	1	0
	9	1	0	0	1
	10	1	0	1	0
GA3	11	1	0	1	1
	14	1	1	1	0
GA4	15	1	1	1	1



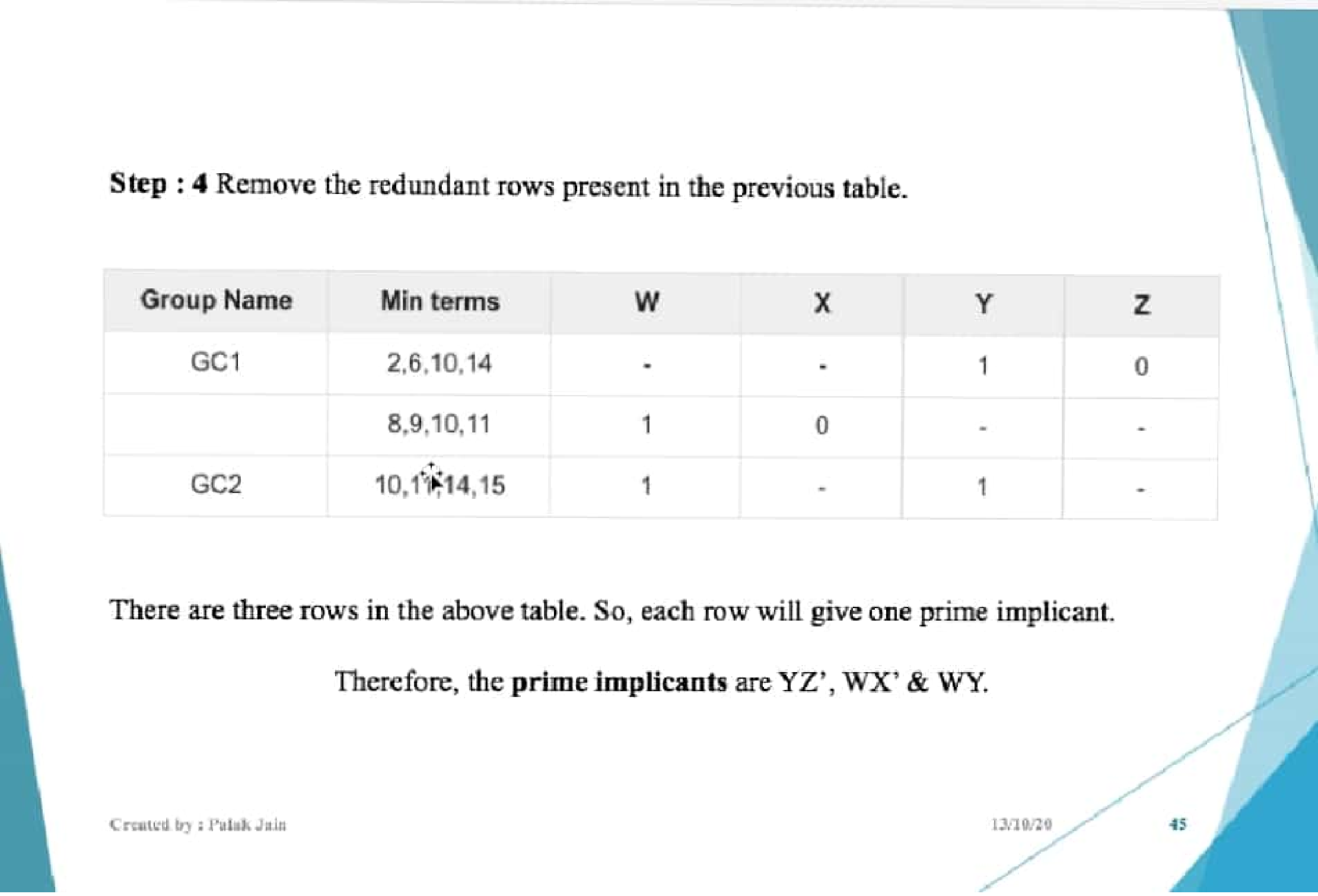
Step : 2 Merging of min terms from adjacent groups.

Group Name	Min terms	W	X	Y	Z
GB1	2,6	0	-	1	0
	2,10	-	0	1	0
	8,9	1	0	0	-
	8,10	1	0	-	0
GB2	6,14	-	1	1	0
	9,11	1	0	-	1
	10,11	1	0	1	-
	10,14	1	-	1	0
GB3	11,15	1	-	1	1
	14,15	1	1	1	-



Step : 3 Again repeat the same process i.e. Merging of min terms from adjacent groups.

Group Name	Min terms	W	X	Y	Z
GB1	2,6,10,14	-	-	1	0
	2,10,6,14	-	-	1	0
	8,9,10,11	1	0	-	-
	8,10,9,11	1	0	-	-
GB2	10,11,14,15	1	-	1	-
	10,14,11,15	1	-	1	-

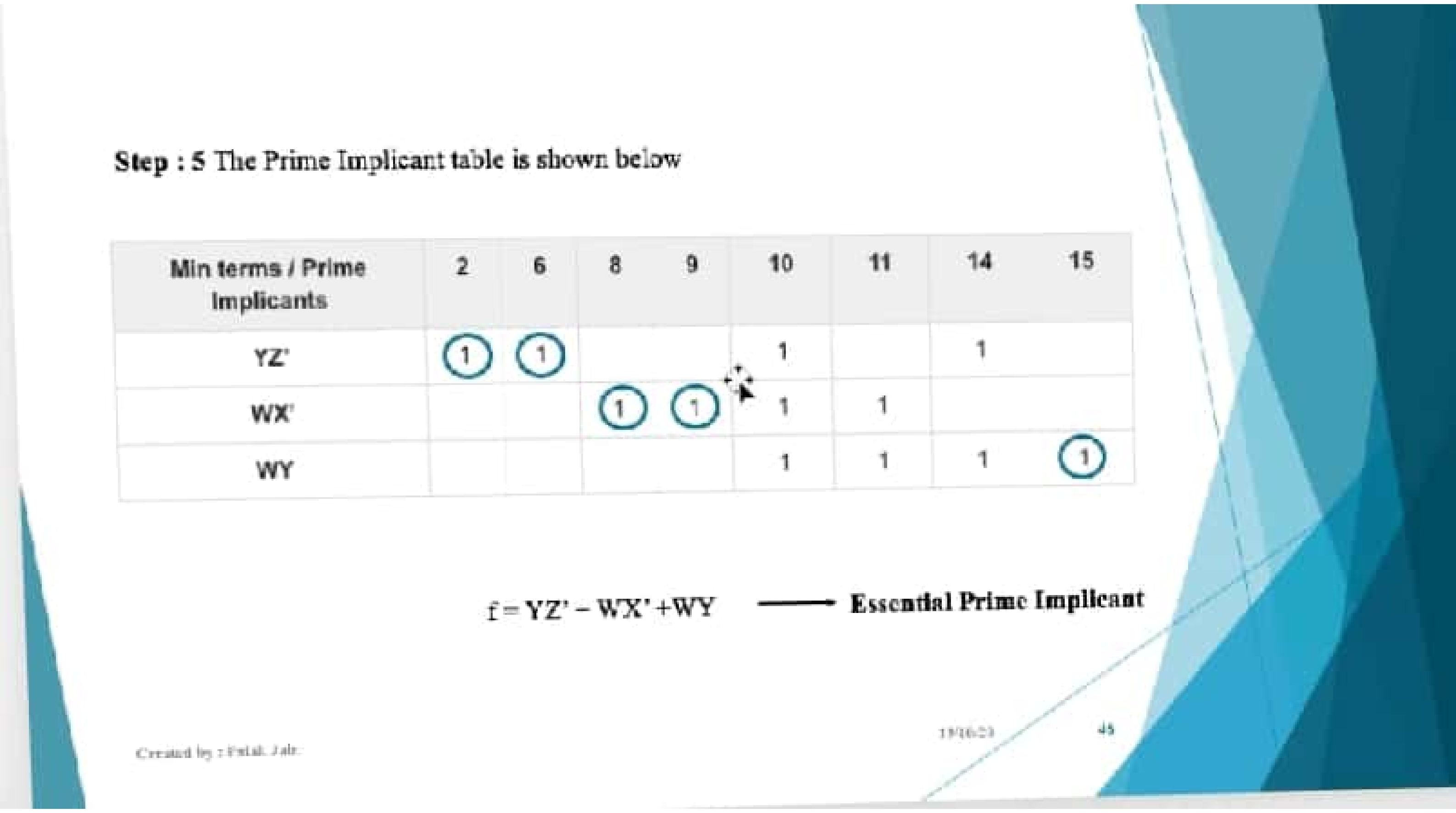


Step : 4 Remove the redundant rows present in the previous table.

Group Name	Min terms	W	X	Y	Z
GC1	2,6,10,14	-	-	1	0
	8,9,10,11	1	0	-	-
GC2	10,11,14,15	1	-	1	-

There are three rows in the above table. So, each row will give one prime implicant.

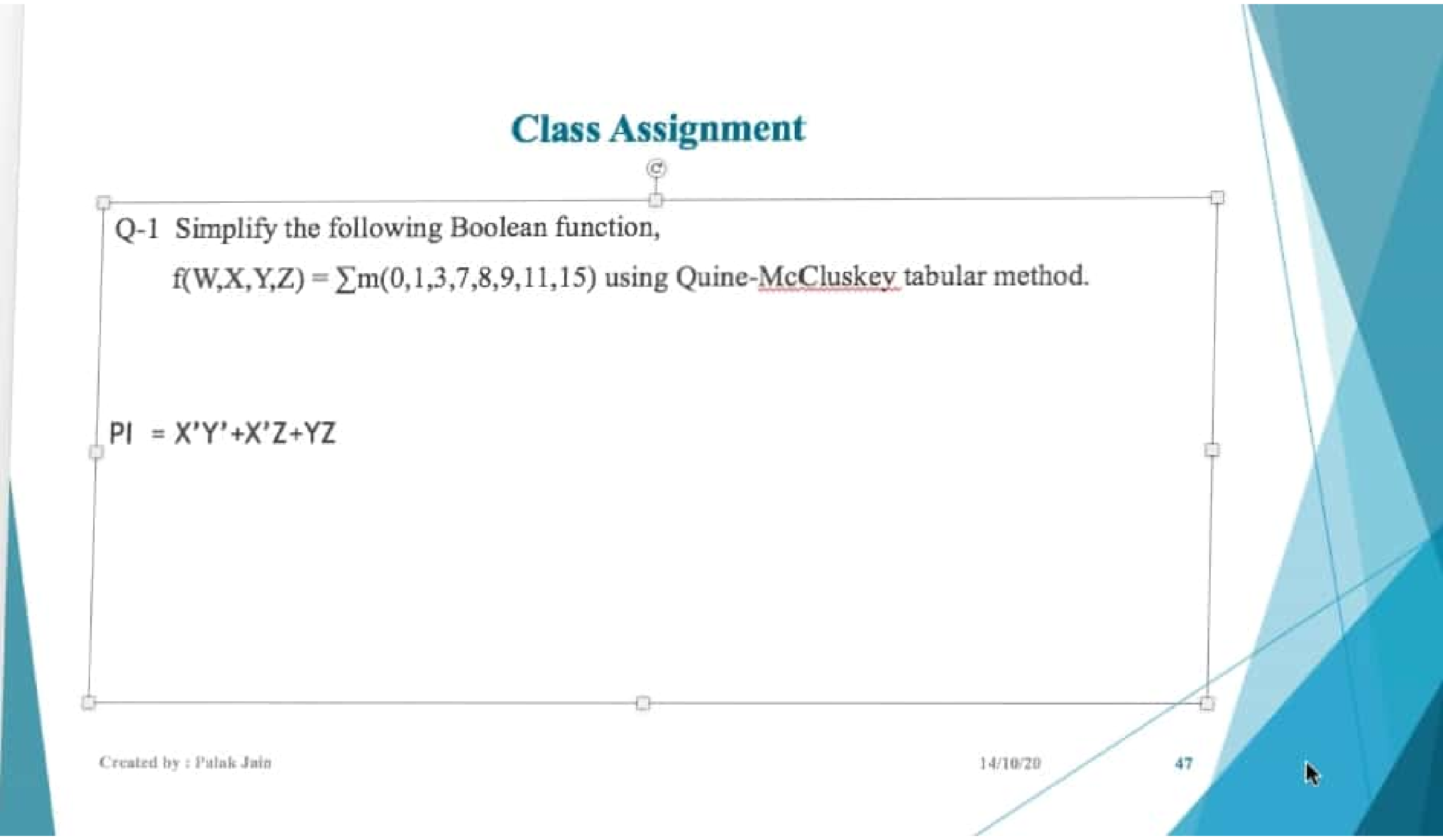
Therefore, the **prime implicants** are YZ', WX' & WY.



Step : 5 The Prime Implicant table is shown below

Min terms / Prime Implicants	2	6	8	9	10	11	14	15
YZ'	①	①			1		1	
WX'			①	①	1	1		
WY					1	1	1	①

$f = YZ' - WX' + WY$ ——— **Essential Prime Implicant**

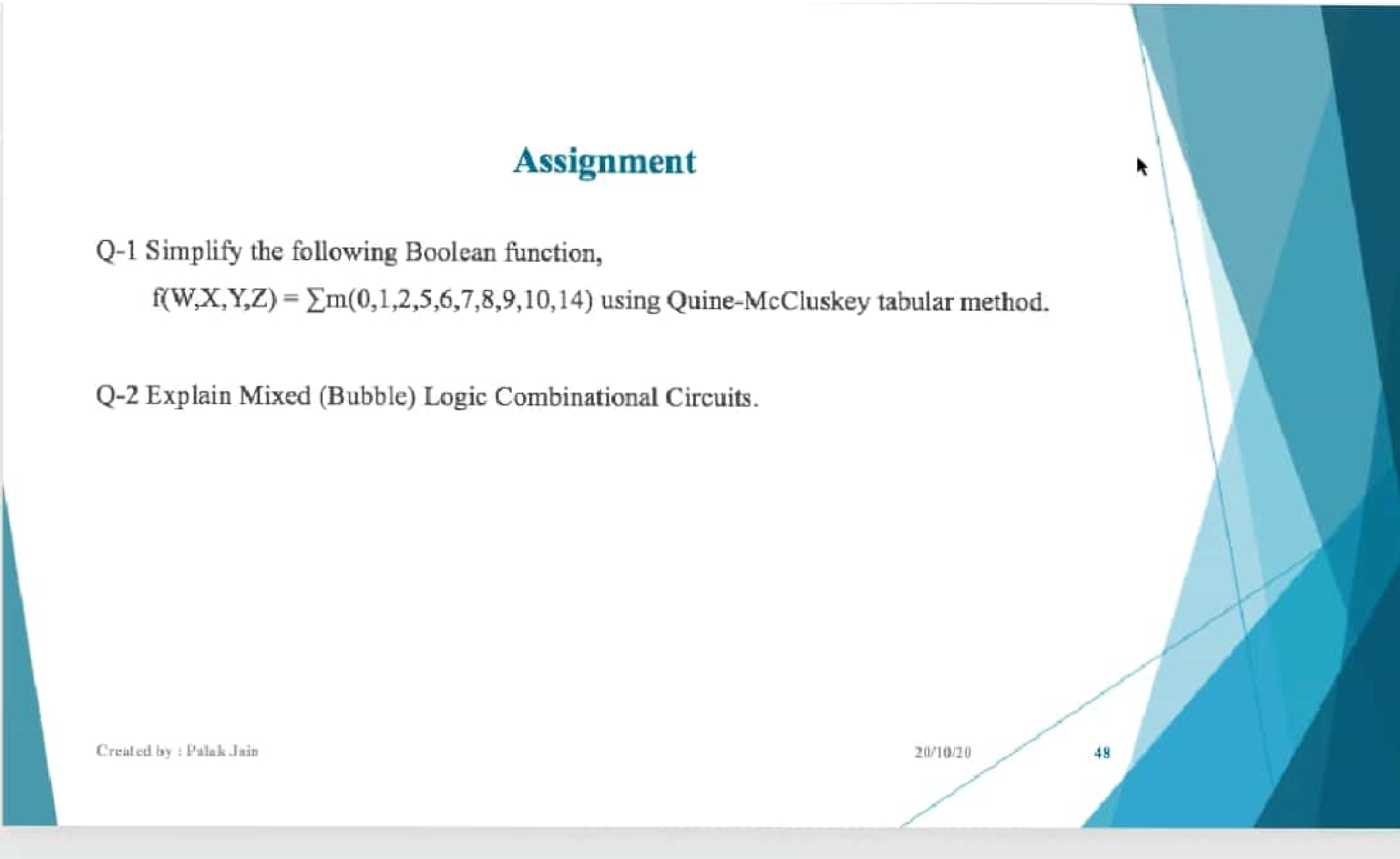


Class Assignment

Q-1 Simplify the following Boolean function,

$f(W,X,Y,Z) = \sum m(0,1,3,7,8,9,11,15)$ using Quine-McCluskey tabular method.

PI = X'Y'+X'Z+YZ

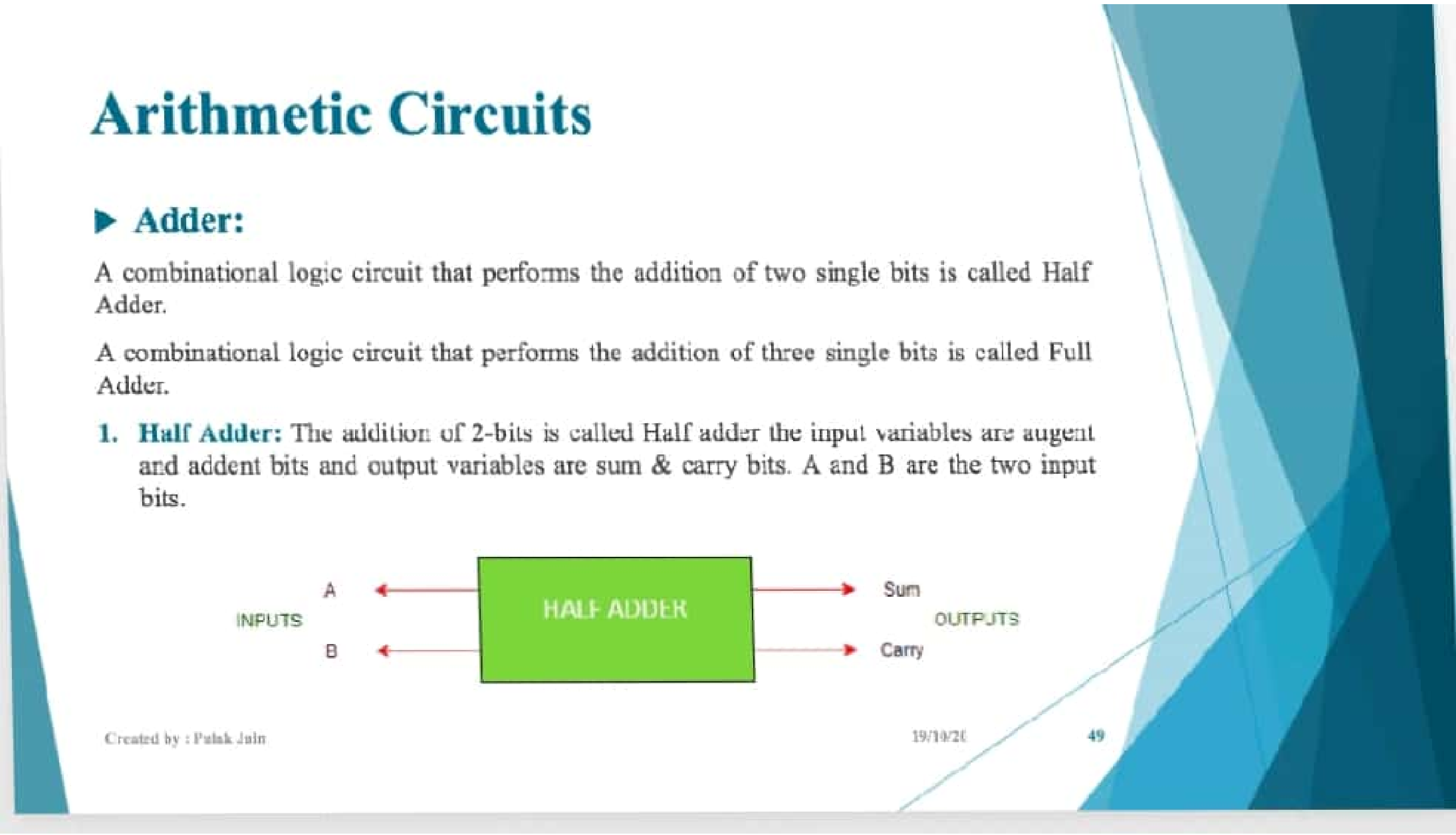


Assignment

Q-1 Simplify the following Boolean function,

$f(W,X,Y,Z) = \sum m(0,1,2,5,6,7,8,9,10,14)$ using Quine-McCluskey tabular method.

Q-2 Explain Mixed (Bubble) Logic Combinational Circuits.



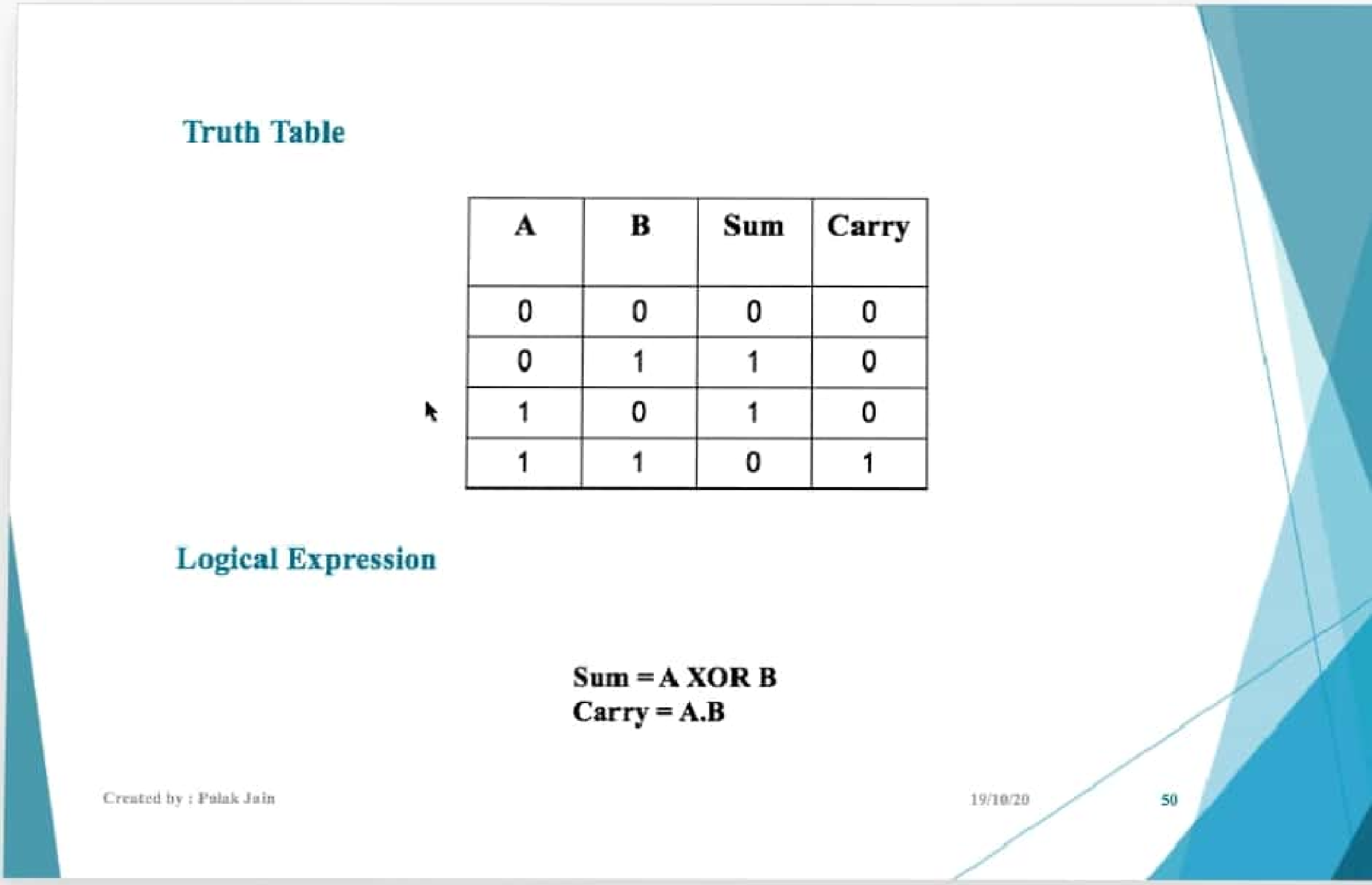
Arithmetic Circuits

▶ Adder:

A combinational logic circuit that performs the addition of two single bits is called Half Adder.

A combinational logic circuit that performs the addition of three single bits is called Full Adder.

1. **Half Adder:** The addition of 2-bits is called Half adder the input variables are augent and addent bits and output variables are sum & carry bits. A and B are the two input bits.

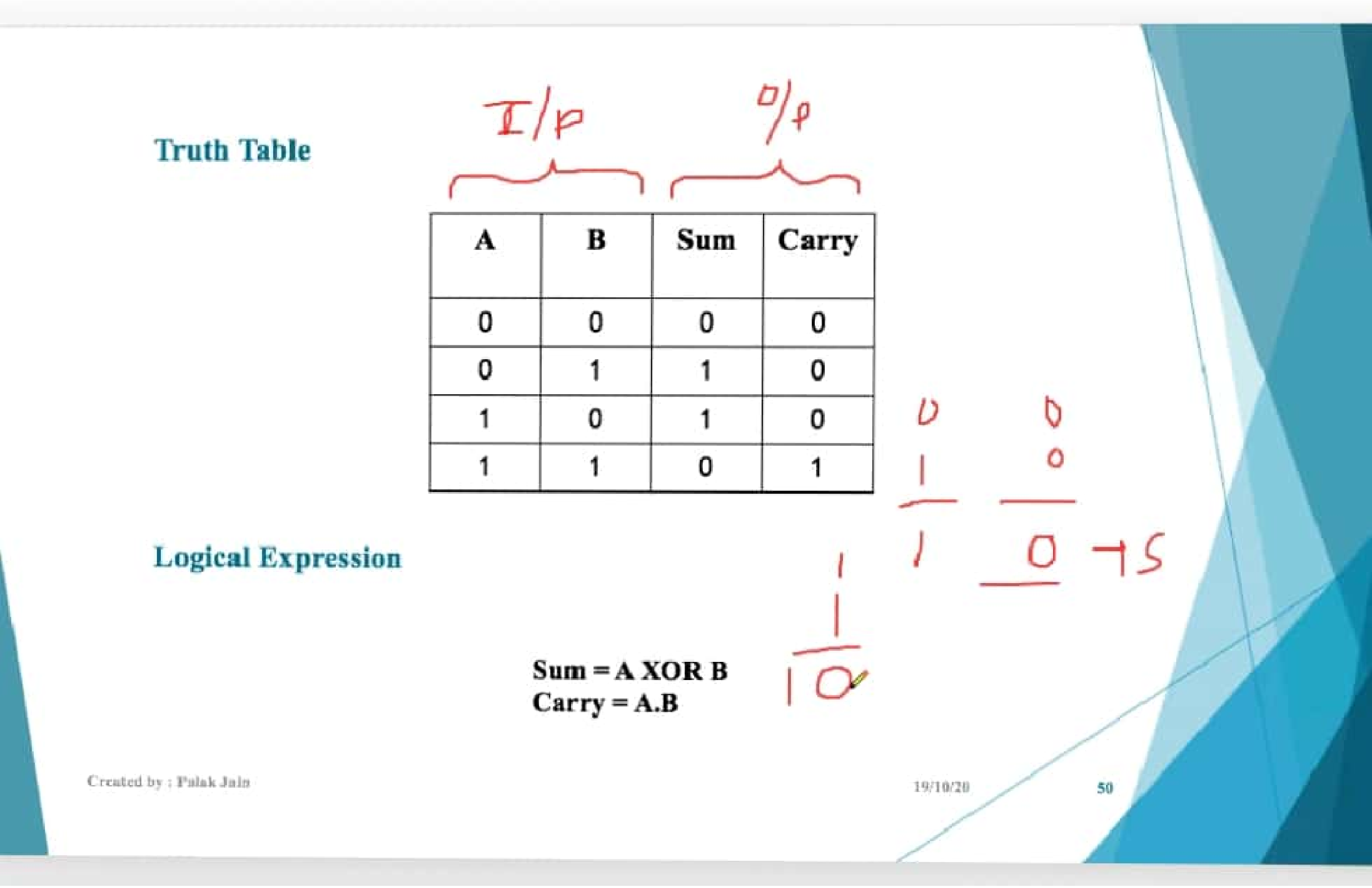


Truth Table

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Logical Expression

Sum = A XOR B
Carry = A.B



Truth Table

$\underbrace{\qquad\qquad}_{}$ I/p $\qquad$ $\underbrace{\qquad\qquad}_{}$ O/p

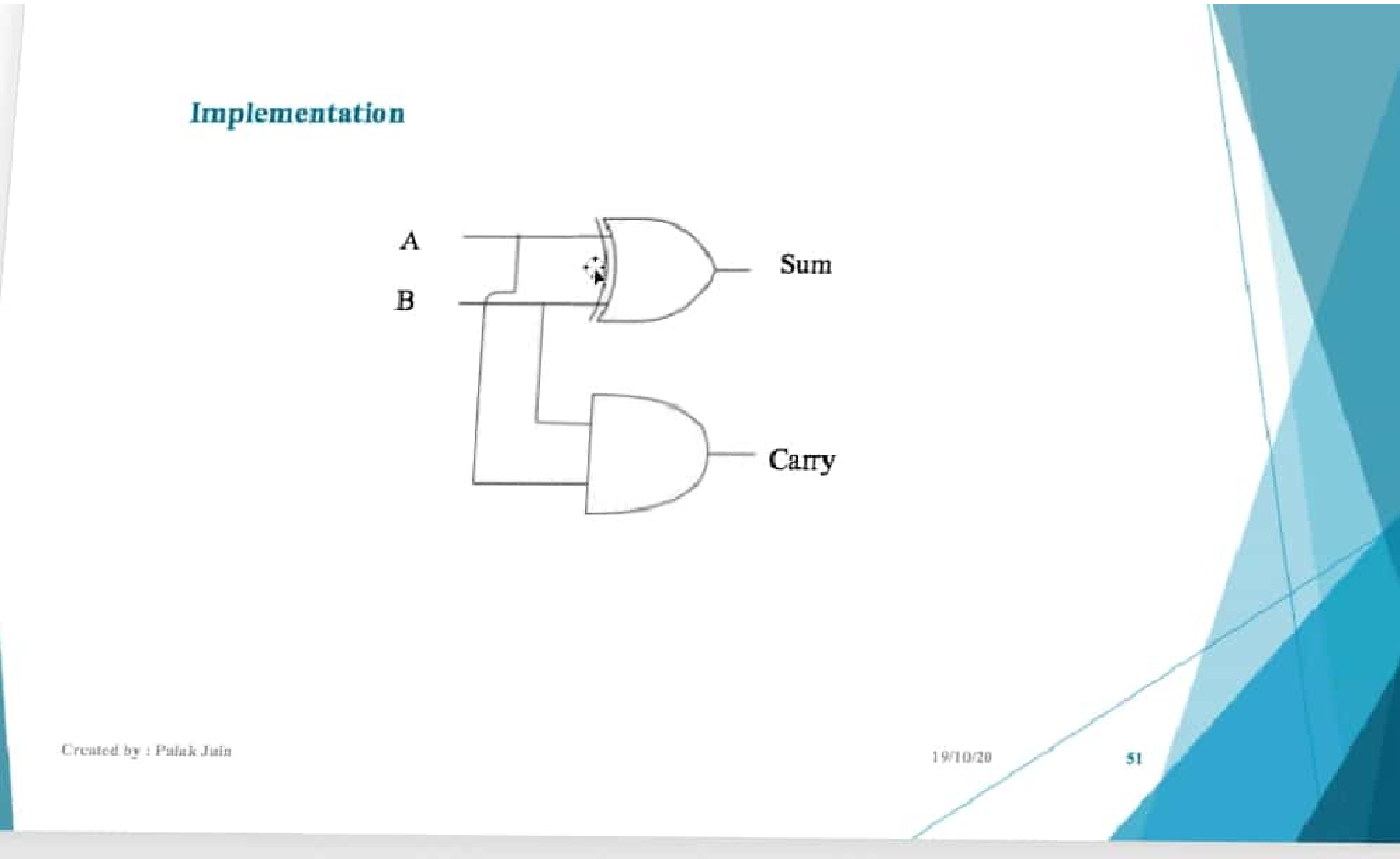
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$\begin{array}{r}0\\1\\\hline 1\end{array} \qquad \begin{array}{r}0\\0\\\hline 0\end{array} \to S$

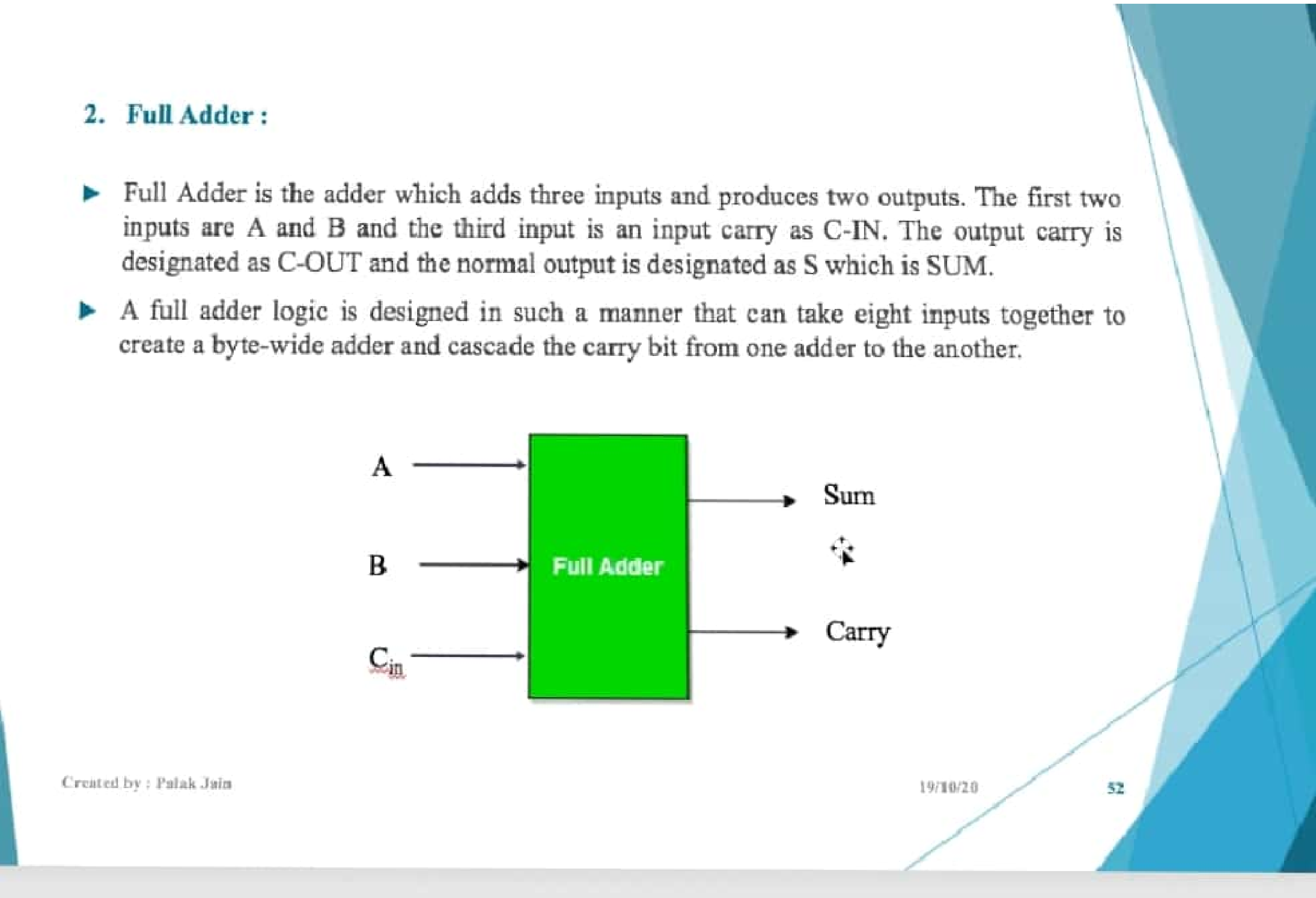
$\begin{array}{r}1\\1\\\hline 10\end{array}$

Logical Expression

Sum = A XOR B
Carry = A.B

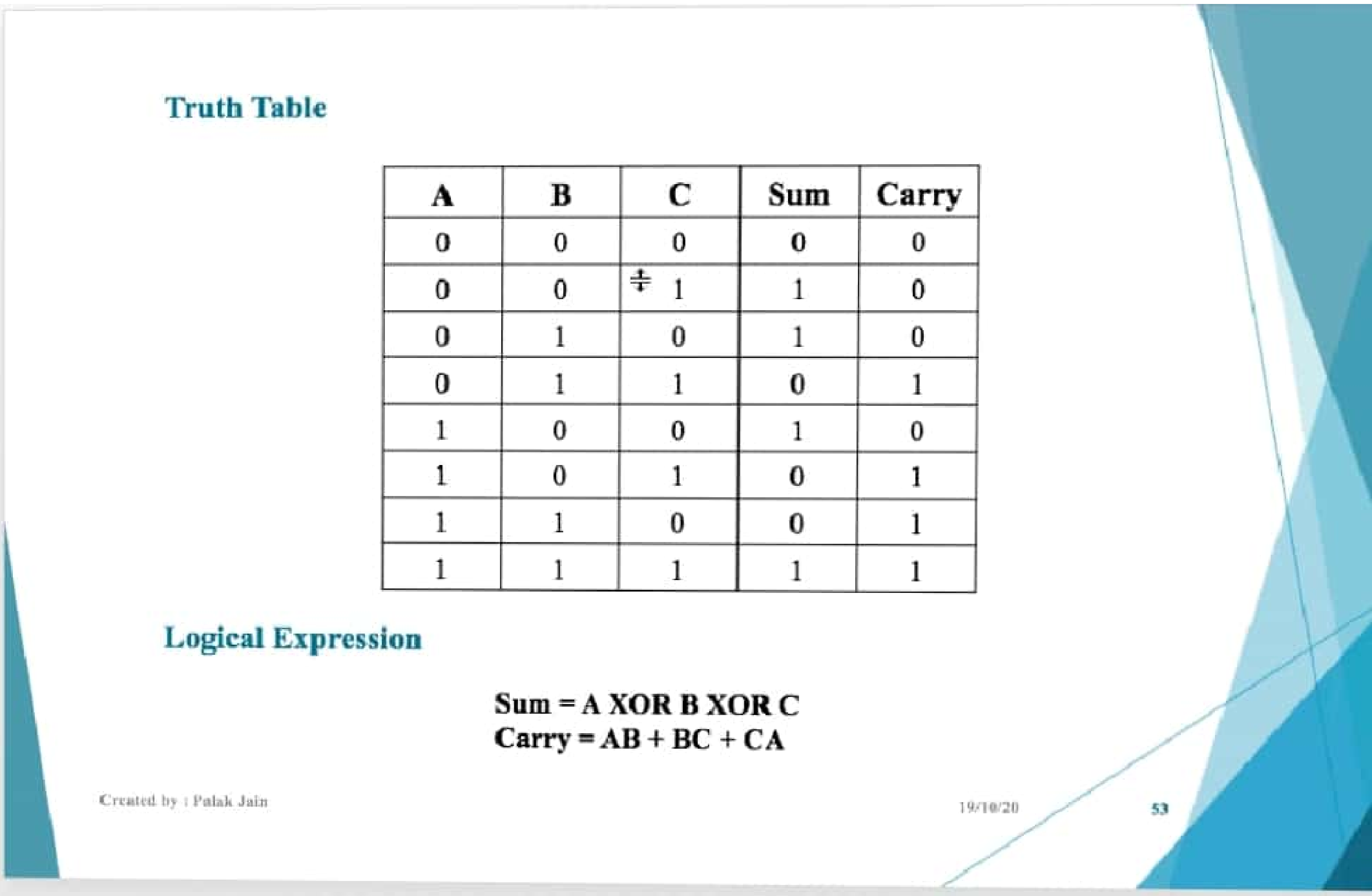


Implementation



2. Full Adder :

- Full Adder is the adder which adds three inputs and produces two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. The output carry is designated as C-OUT and the normal output is designated as S which is SUM.
- A full adder logic is designed in such a manner that can take eight inputs together to create a byte-wide adder and cascade the carry bit from one adder to the another.

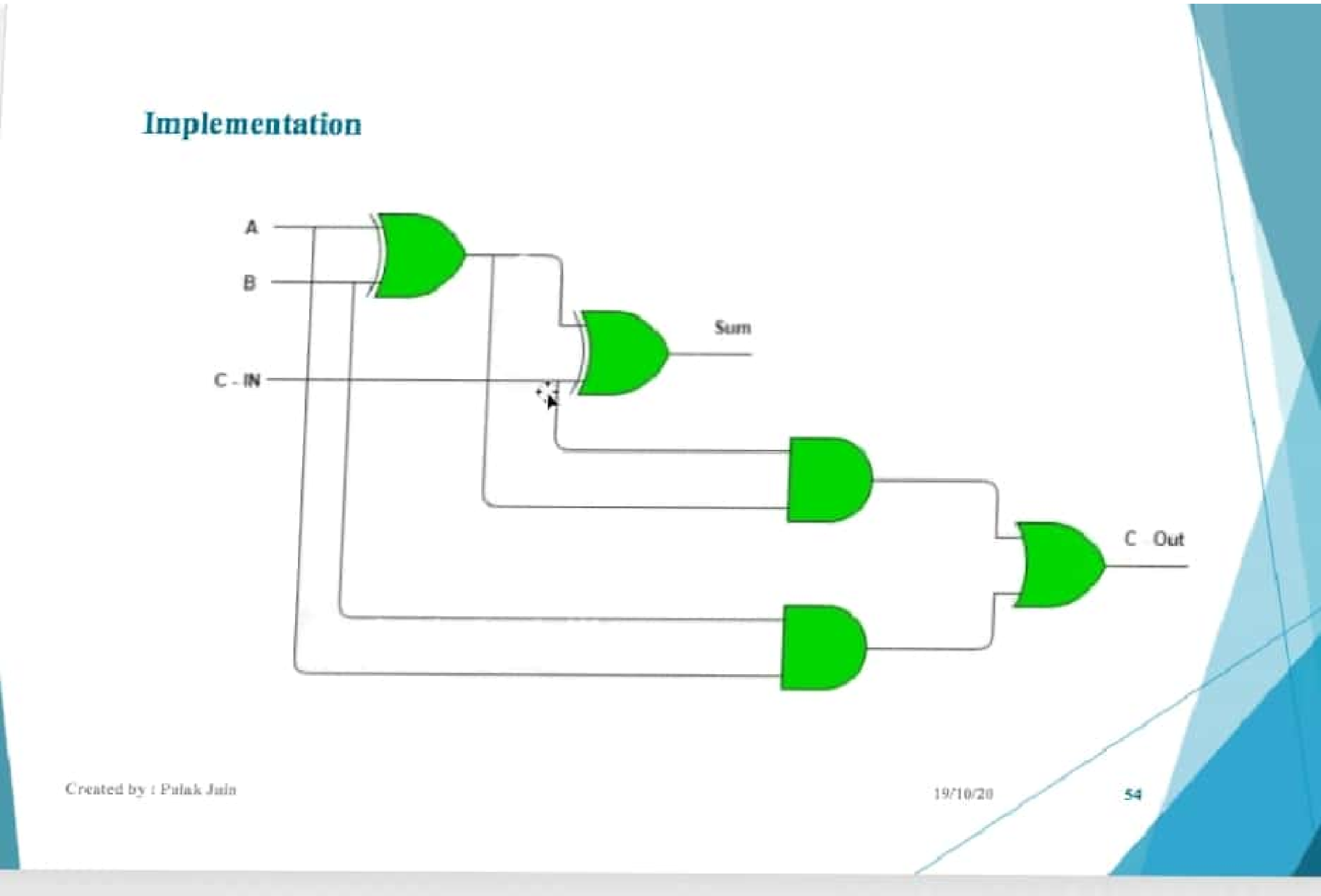


Truth Table

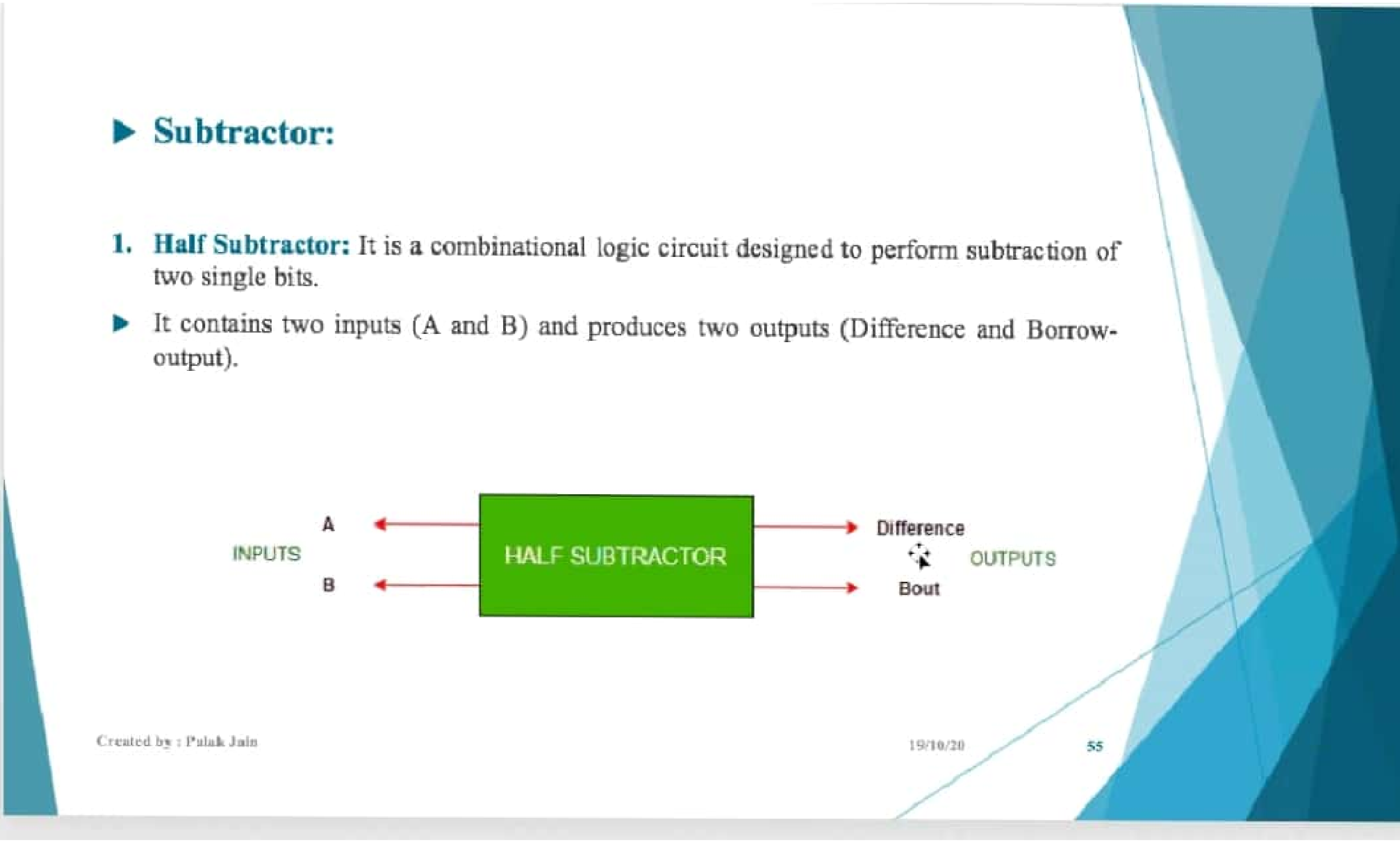
A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Logical Expression

Sum = A XOR B XOR C
Carry = AB + BC + CA



Implementation



Subtractor:

1. **Half Subtractor:** It is a combinational logic circuit designed to perform subtraction of two single bits.

- It contains two inputs (A and B) and produces two outputs (Difference and Borrow-output).

INPUTS

A

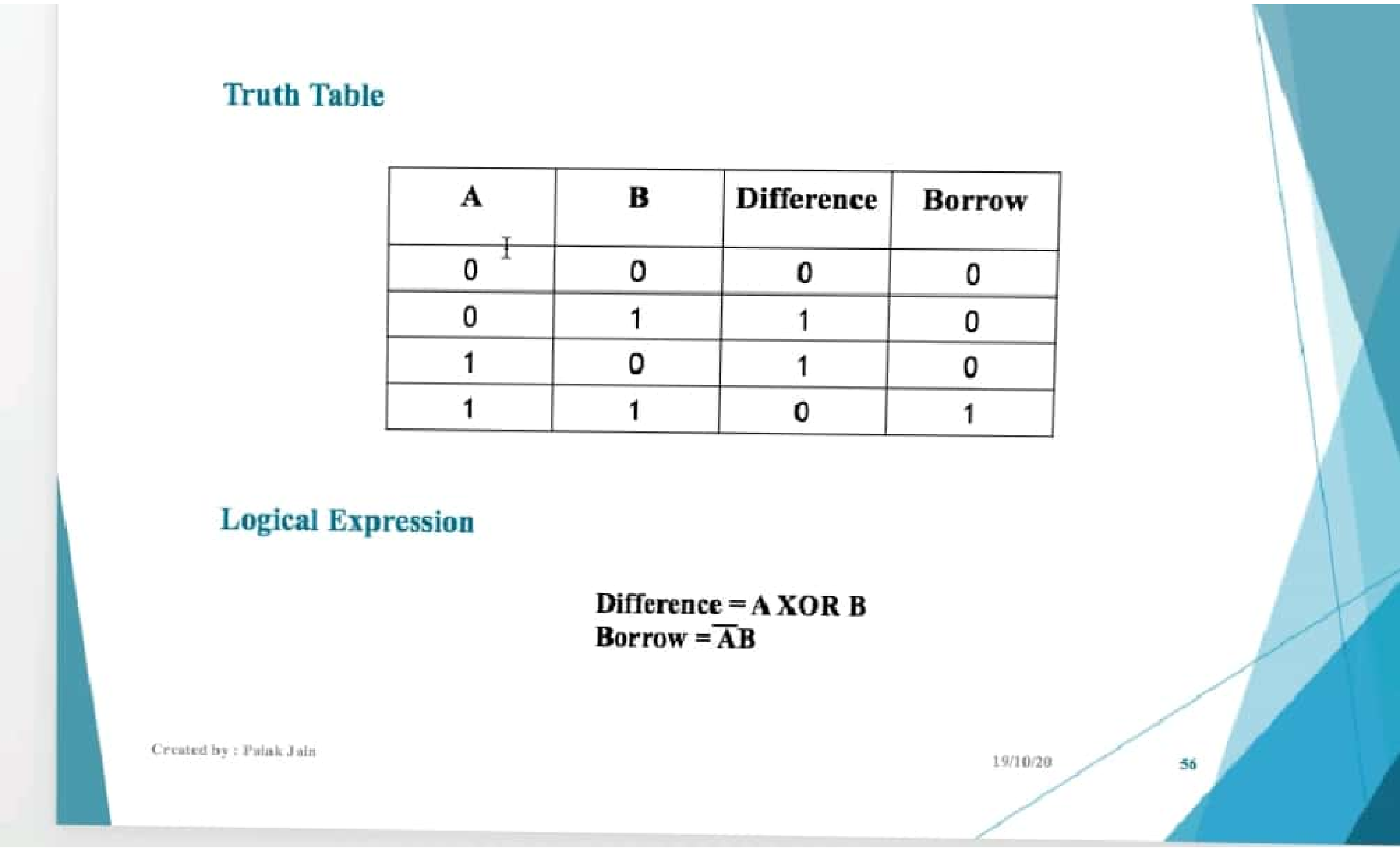
B

HALF SUBTRACTOR

Difference

Bout

OUTPUTS

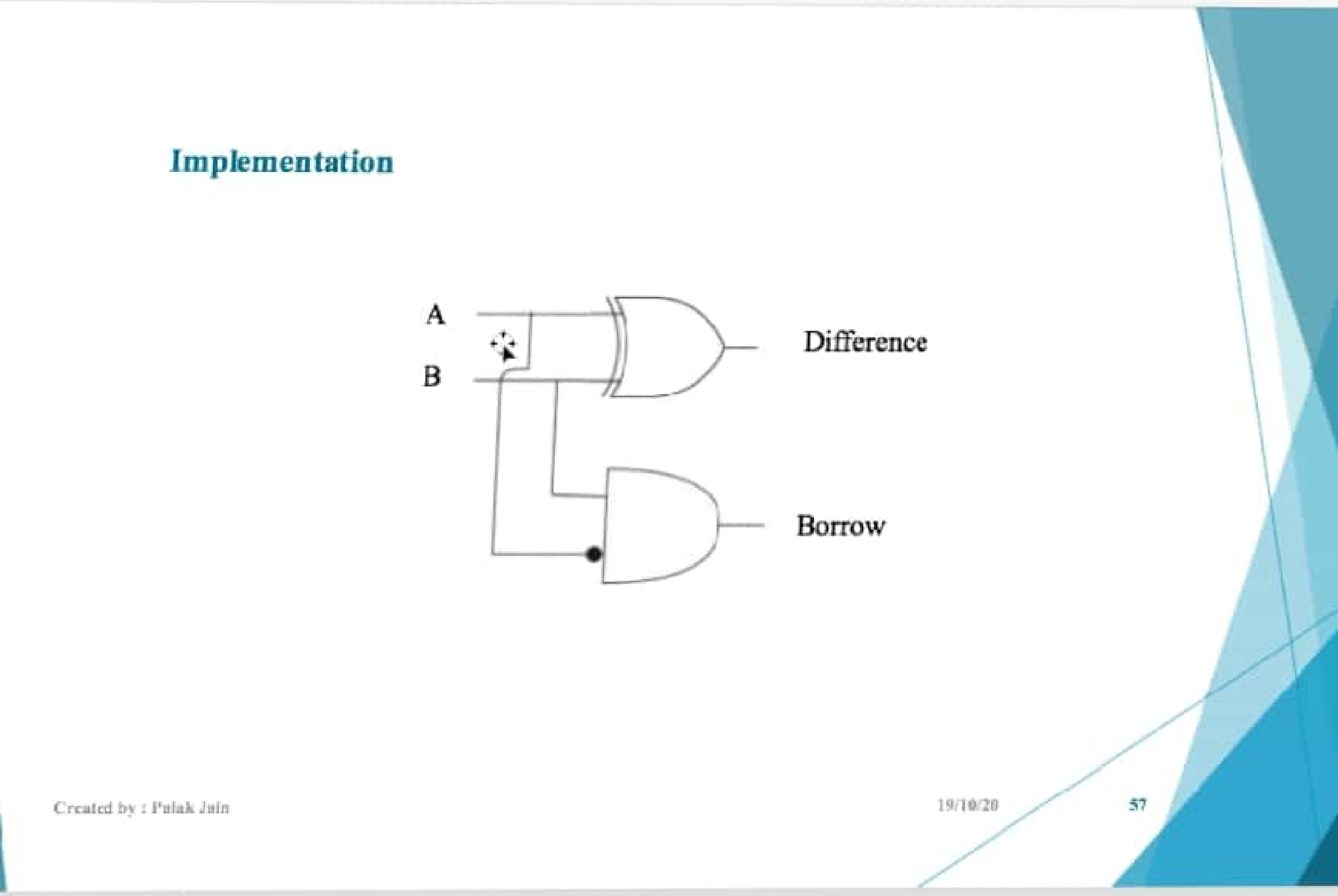


Truth Table

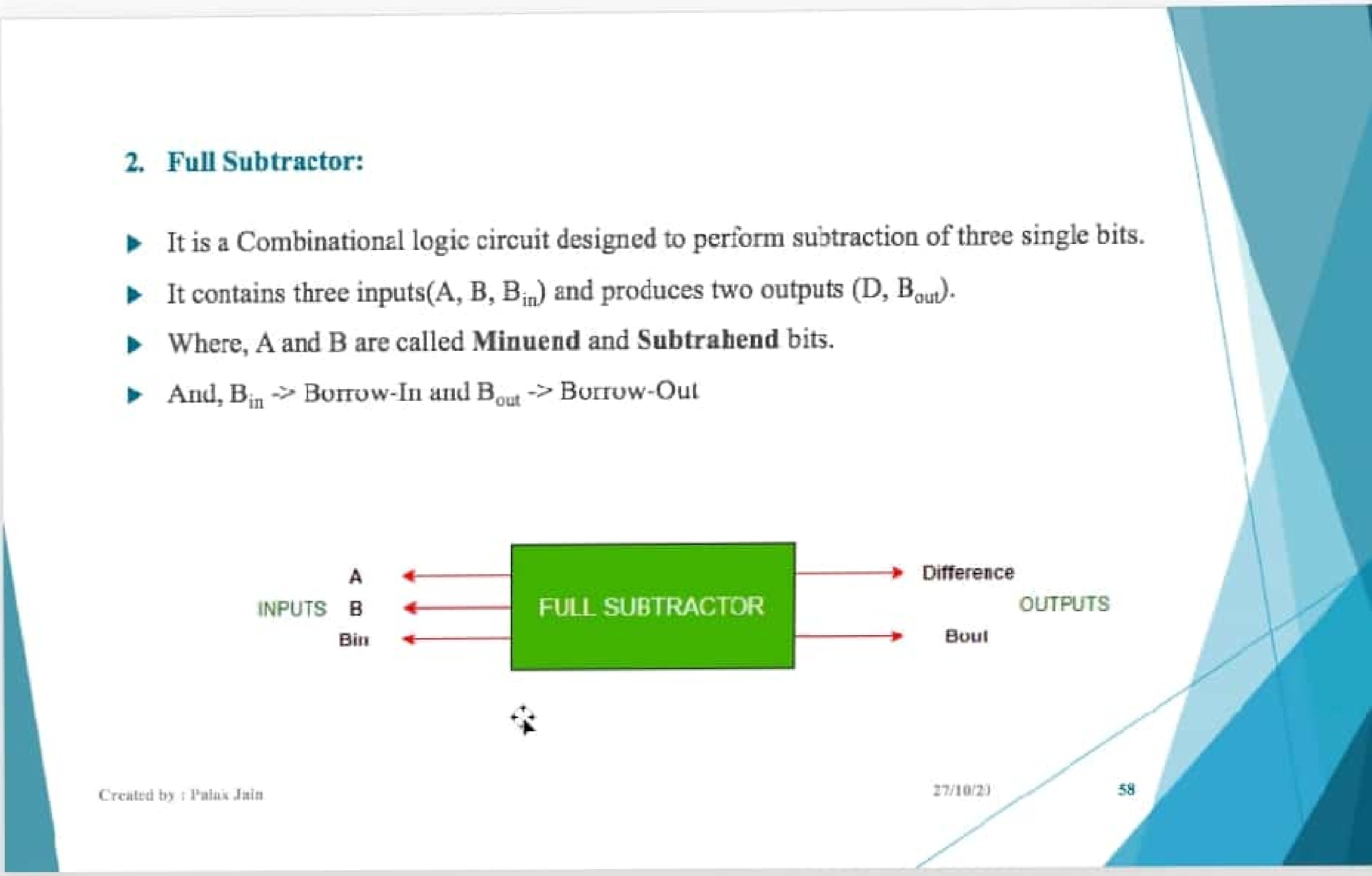
A	B	Difference	Borrow
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Logical Expression

Difference = A XOR B
Borrow = $\overline{A}$B



Implementation



2. Full Subtractor:

- It is a Combinational logic circuit designed to perform subtraction of three single bits.
- It contains three inputs(A, B, B_{in}) and produces two outputs (D, B_{out}).
- Where, A and B are called **Minuend** and **Subtrahend** bits.
- And, B_{in} -> Borrow-In and B_{out} -> Borrow-Out

INPUTS

A

B

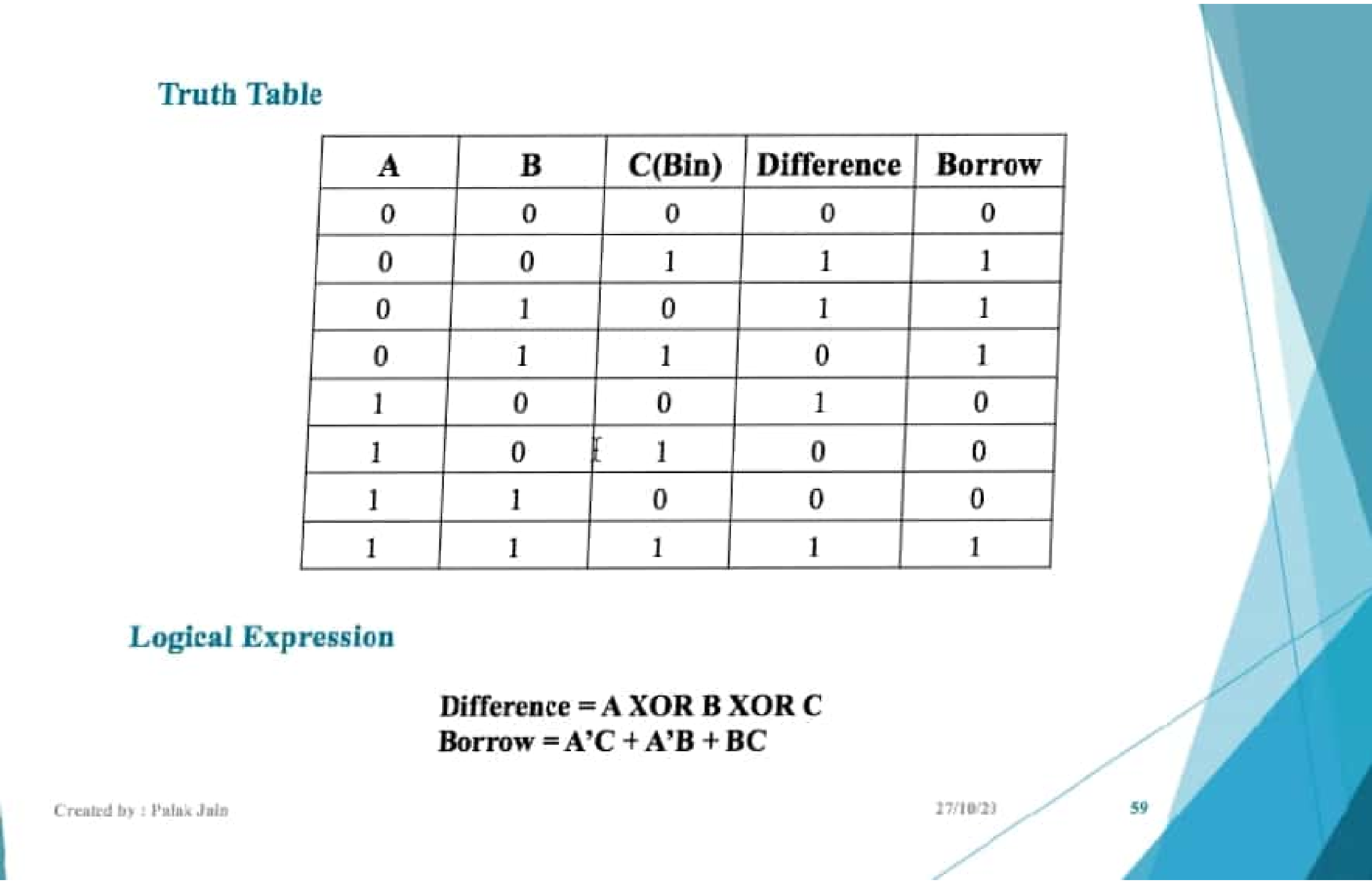
Bin

FULL SUBTRACTOR

Difference

Bout

OUTPUTS

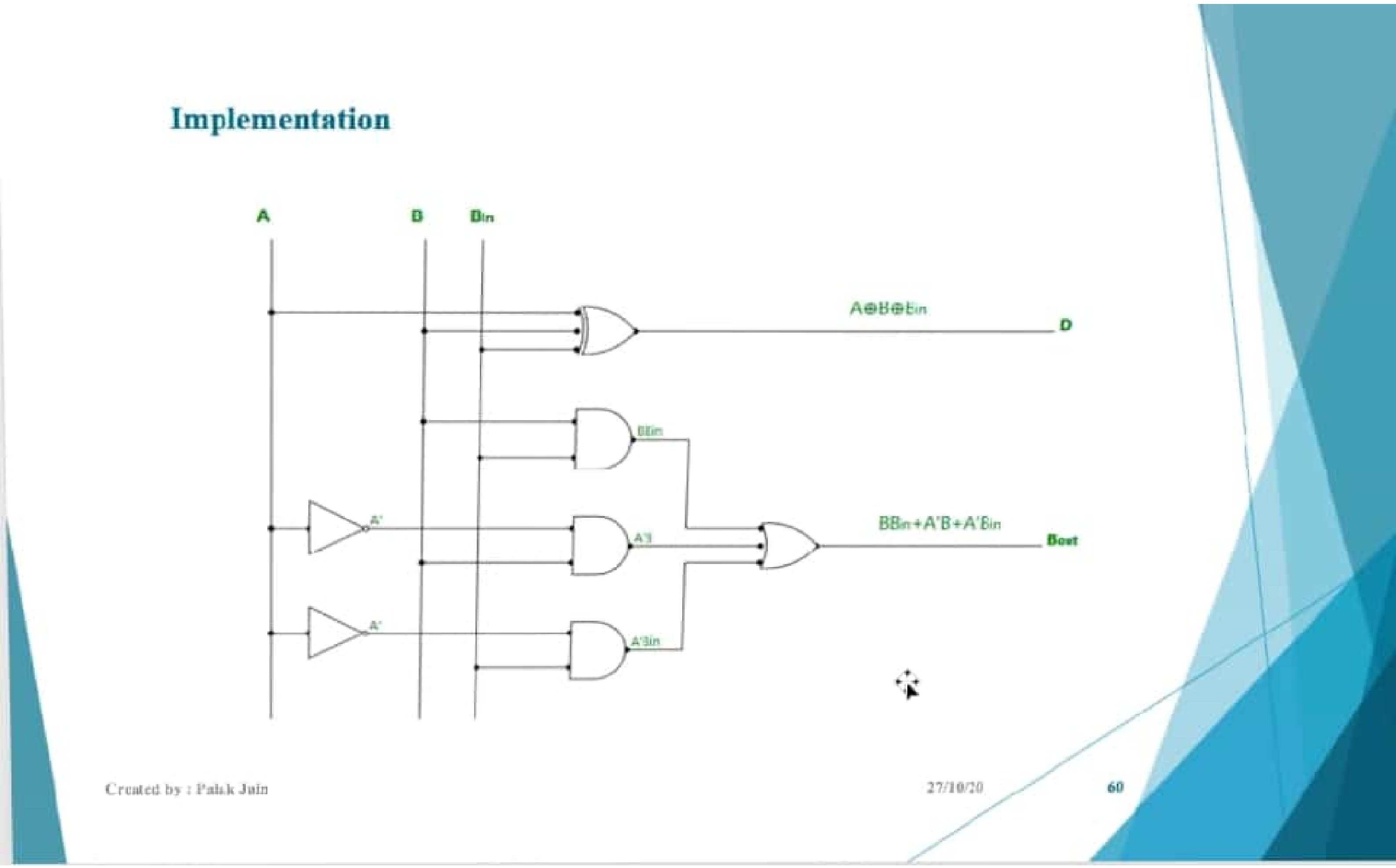


Truth Table

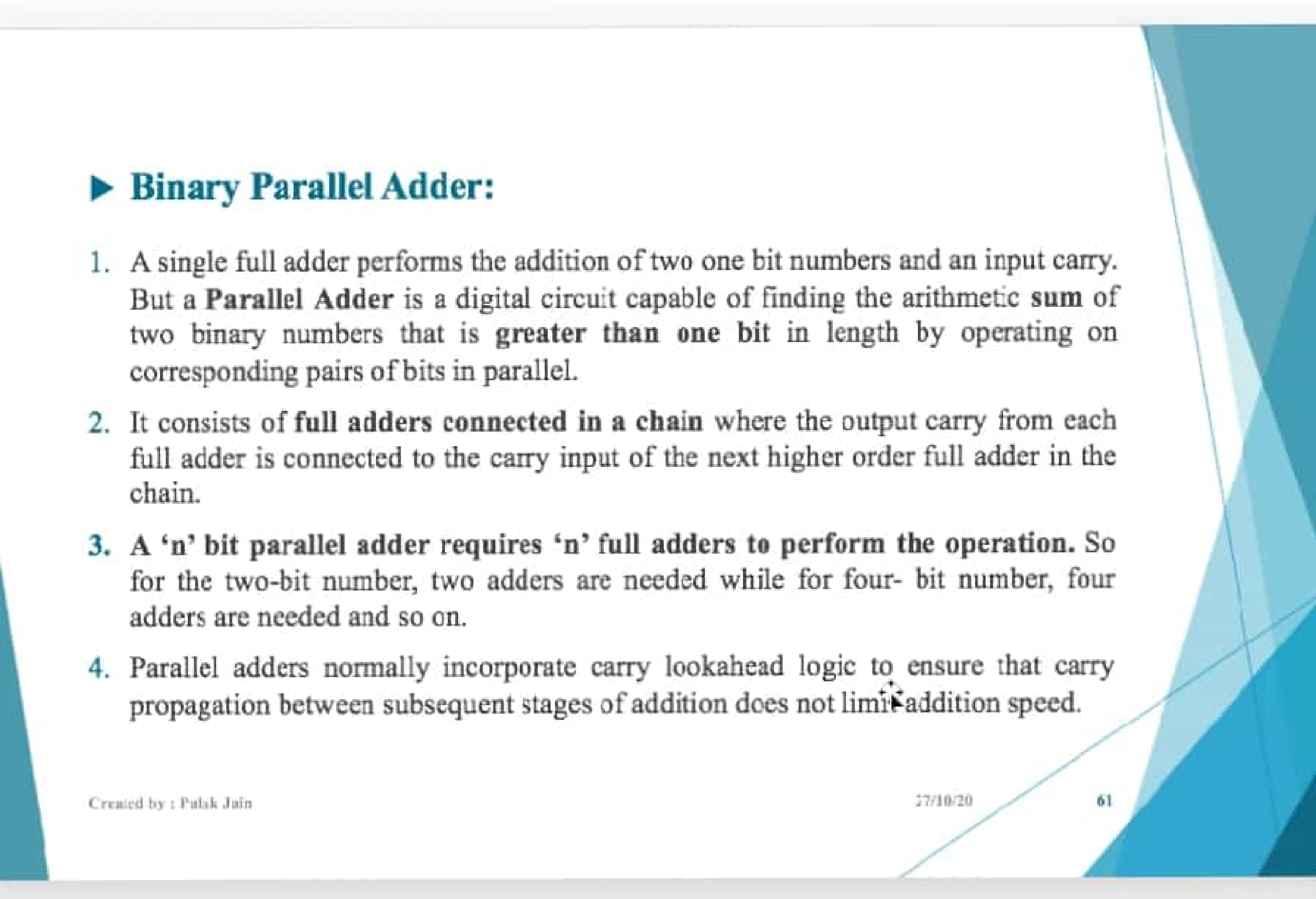
A	B	C(Bin)	Difference	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Logical Expression

Difference = A XOR B XOR C
Borrow = A'C + A'B + BC

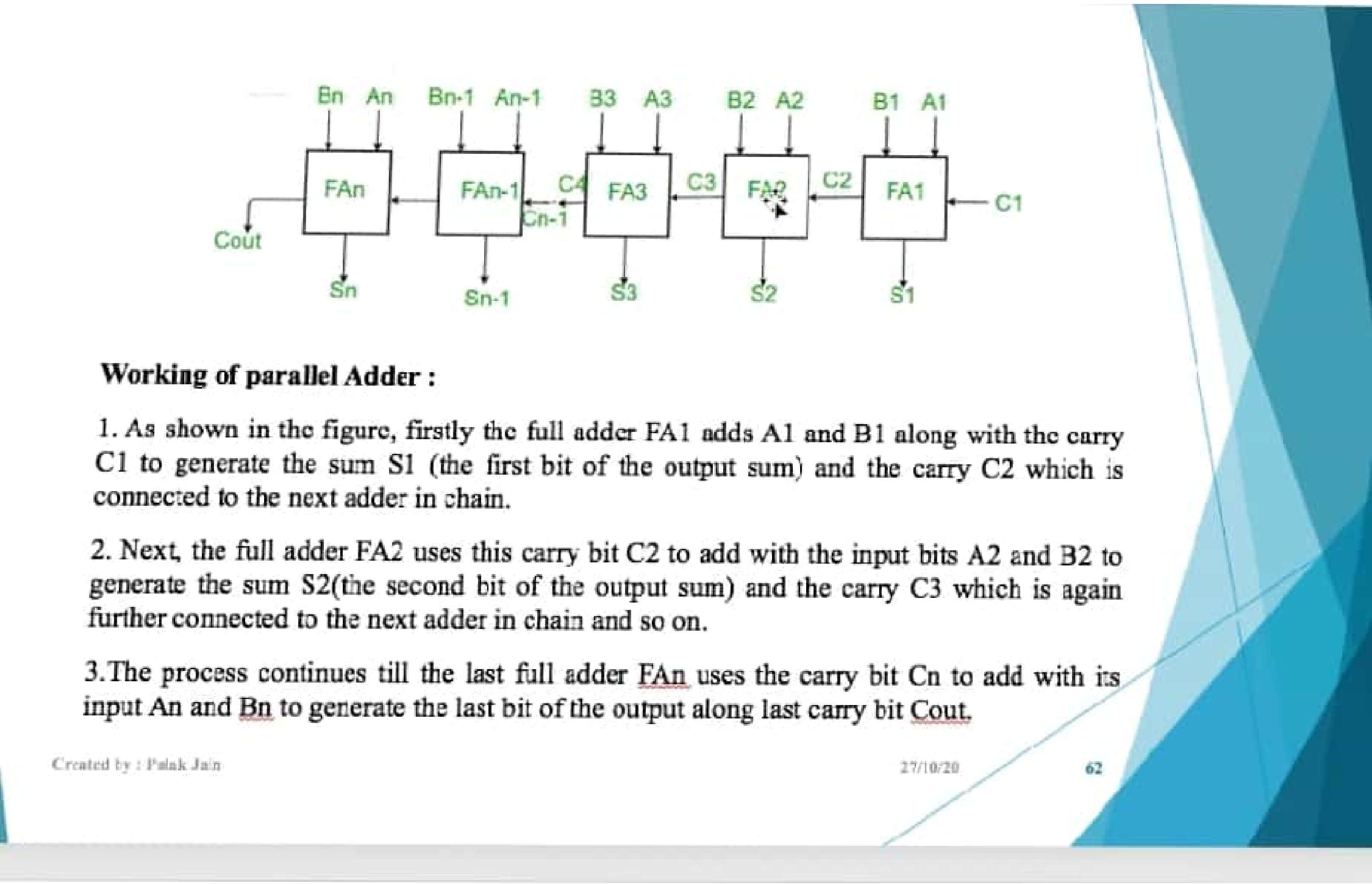


Implementation



Binary Parallel Adder:

1. A single full adder performs the addition of two one bit numbers and an input carry. But a **Parallel Adder** is a digital circuit capable of finding the arithmetic **sum** of two binary numbers that is **greater than one bit** in length by operating on corresponding pairs of bits in parallel.
2. It consists of **full adders connected in a chain** where the output carry from each full adder is connected to the carry input of the next higher order full adder in the chain.
3. **A 'n' bit parallel adder requires 'n' full adders to perform the operation.** So for the two-bit number, two adders are needed while for four- bit number, four adders are needed and so on.
4. Parallel adders normally incorporate carry lookahead logic to ensure that carry propagation between subsequent stages of addition does not limit addition speed.

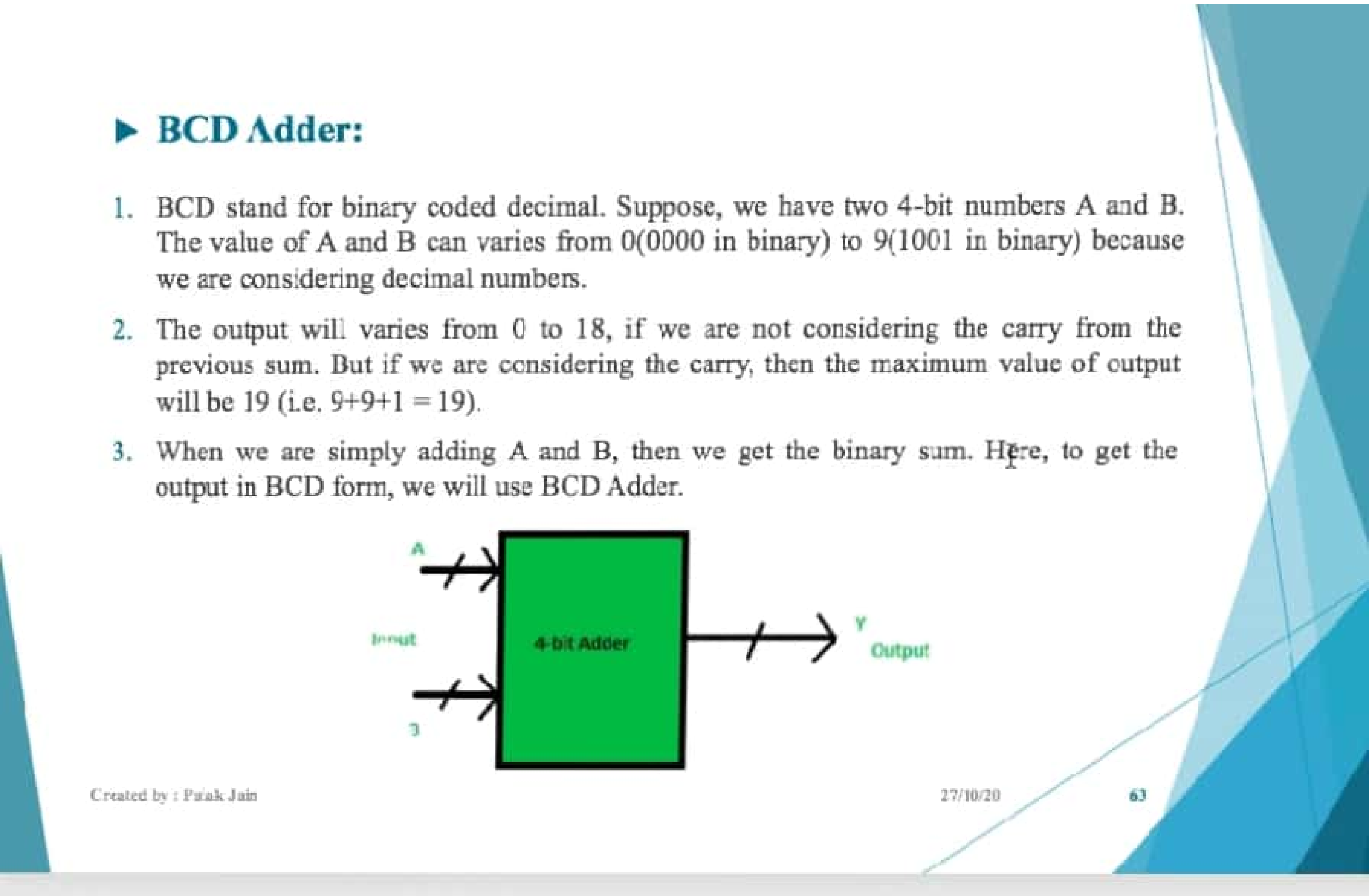


Working of parallel Adder :

1. As shown in the figure, firstly the full adder FA1 adds A1 and B1 along with the carry C1 to generate the sum S1 (the first bit of the output sum) and the carry C2 which is connected to the next adder in chain.

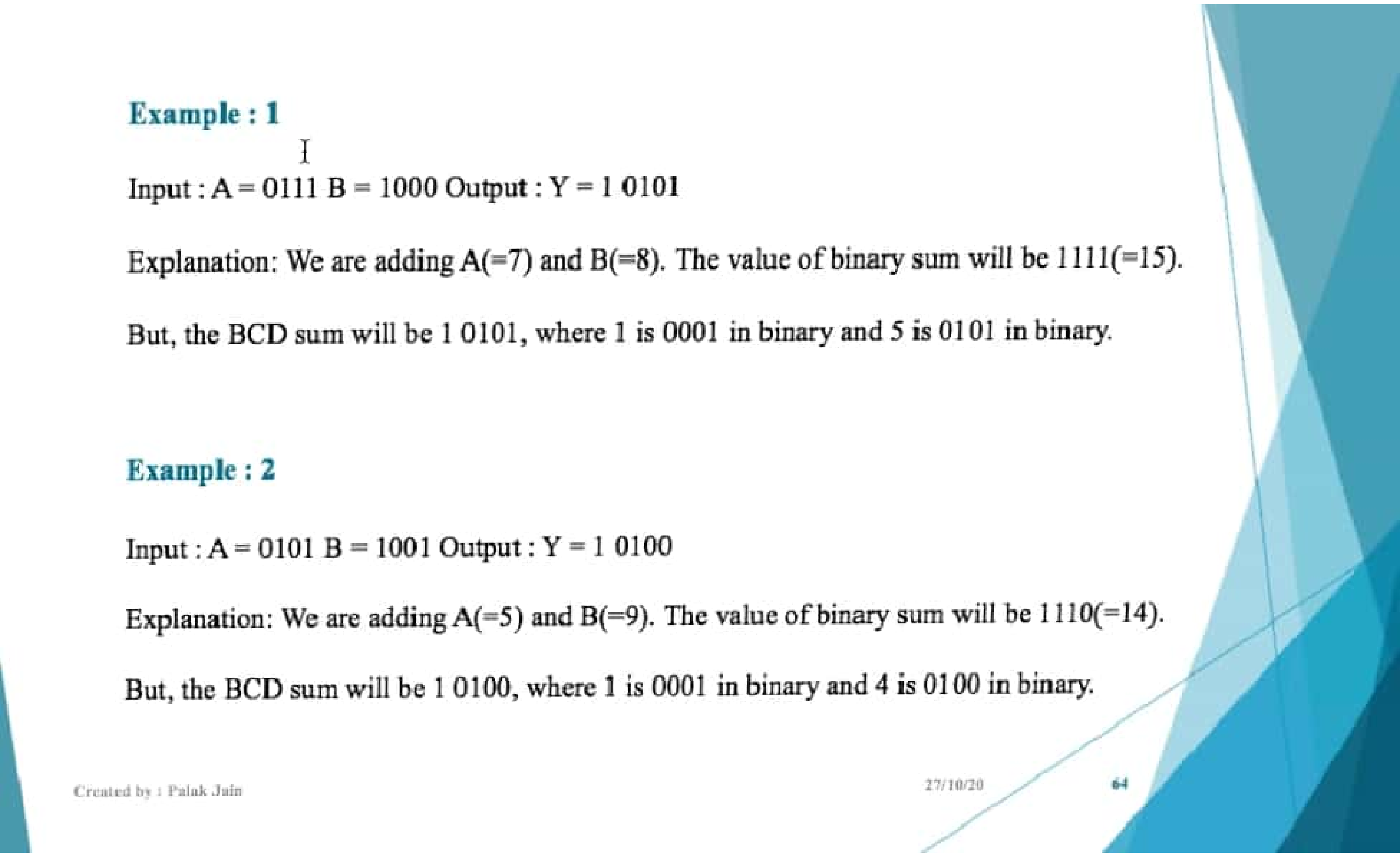
2. Next, the full adder FA2 uses this carry bit C2 to add with the input bits A2 and B2 to generate the sum S2(the second bit of the output sum) and the carry C3 which is again further connected to the next adder in chain and so on.

3.The process continues till the last full adder FAn uses the carry bit Cn to add with its input An and Bn to generate the last bit of the output along last carry bit Cout.



▶ BCD Adder:

1. BCD stand for binary coded decimal. Suppose, we have two 4-bit numbers A and B. The value of A and B can varies from 0(0000 in binary) to 9(1001 in binary) because we are considering decimal numbers.
2. The output will varies from 0 to 18, if we are not considering the carry from the previous sum. But if we are considering the carry, then the maximum value of output will be 19 (i.e. 9+9+1 = 19).
3. When we are simply adding A and B, then we get the binary sum. Here, to get the output in BCD form, we will use BCD Adder.



Example : 1

Input : A = 0111 B = 1000 Output : Y = 1 0101

Explanation: We are adding A(=7) and B(=8). The value of binary sum will be 1111(=15).

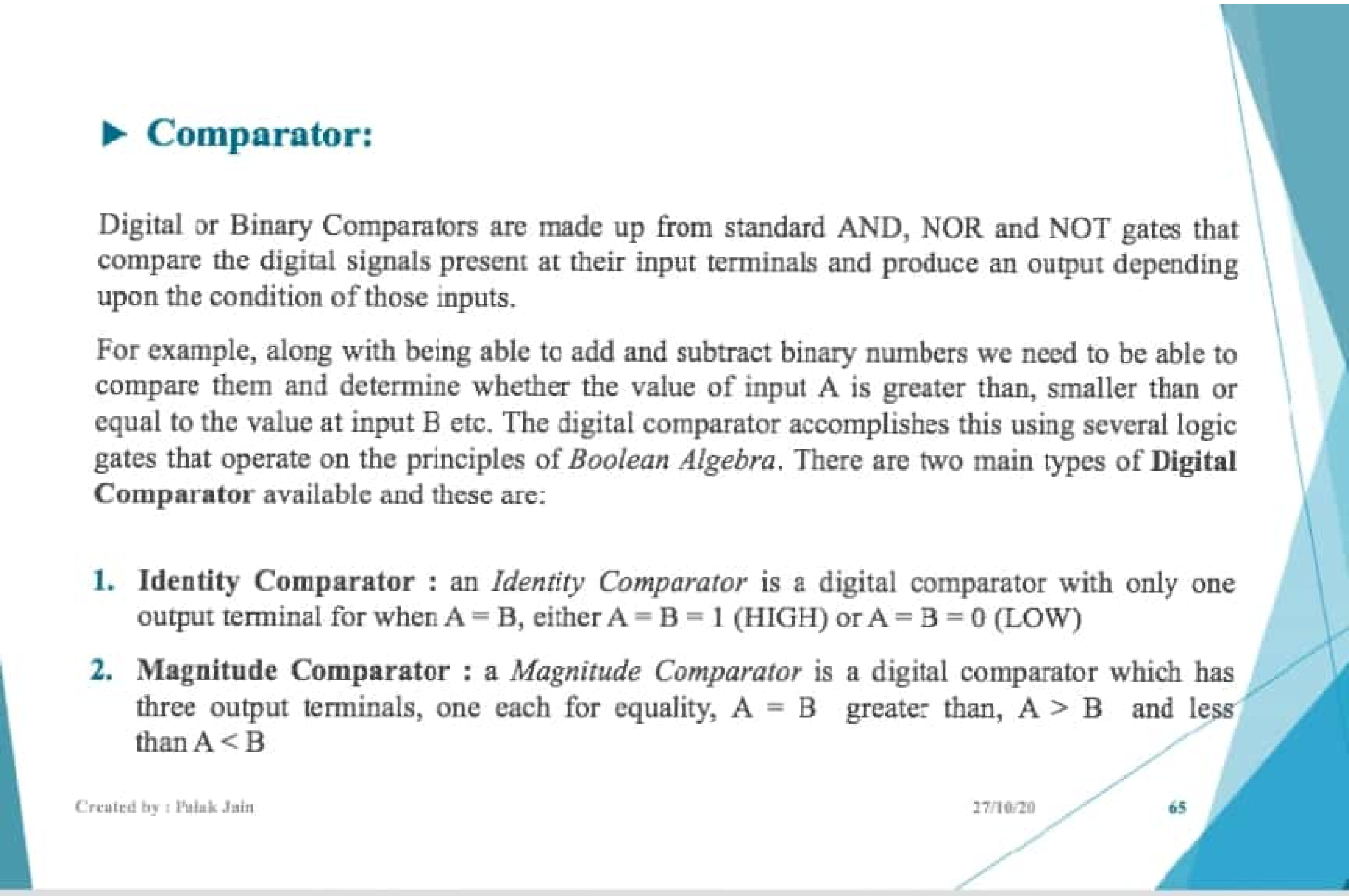
But, the BCD sum will be 1 0101, where 1 is 0001 in binary and 5 is 0101 in binary.

Example : 2

Input : A = 0101 B = 1001 Output : Y = 1 0100

Explanation: We are adding A(=5) and B(=9). The value of binary sum will be 1110(=14).

But, the BCD sum will be 1 0100, where 1 is 0001 in binary and 4 is 0100 in binary.



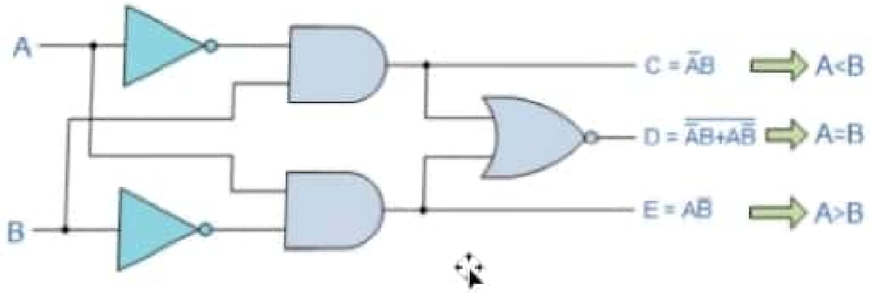
▶ Comparator:

Digital or Binary Comparators are made up from standard AND, NOR and NOT gates that compare the digital signals present at their input terminals and produce an output depending upon the condition of those inputs.

For example, along with being able to add and subtract binary numbers we need to be able to compare them and determine whether the value of input A is greater than, smaller than or equal to the value at input B etc. The digital comparator accomplishes this using several logic gates that operate on the principles of *Boolean Algebra*. There are two main types of **Digital Comparator** available and these are:

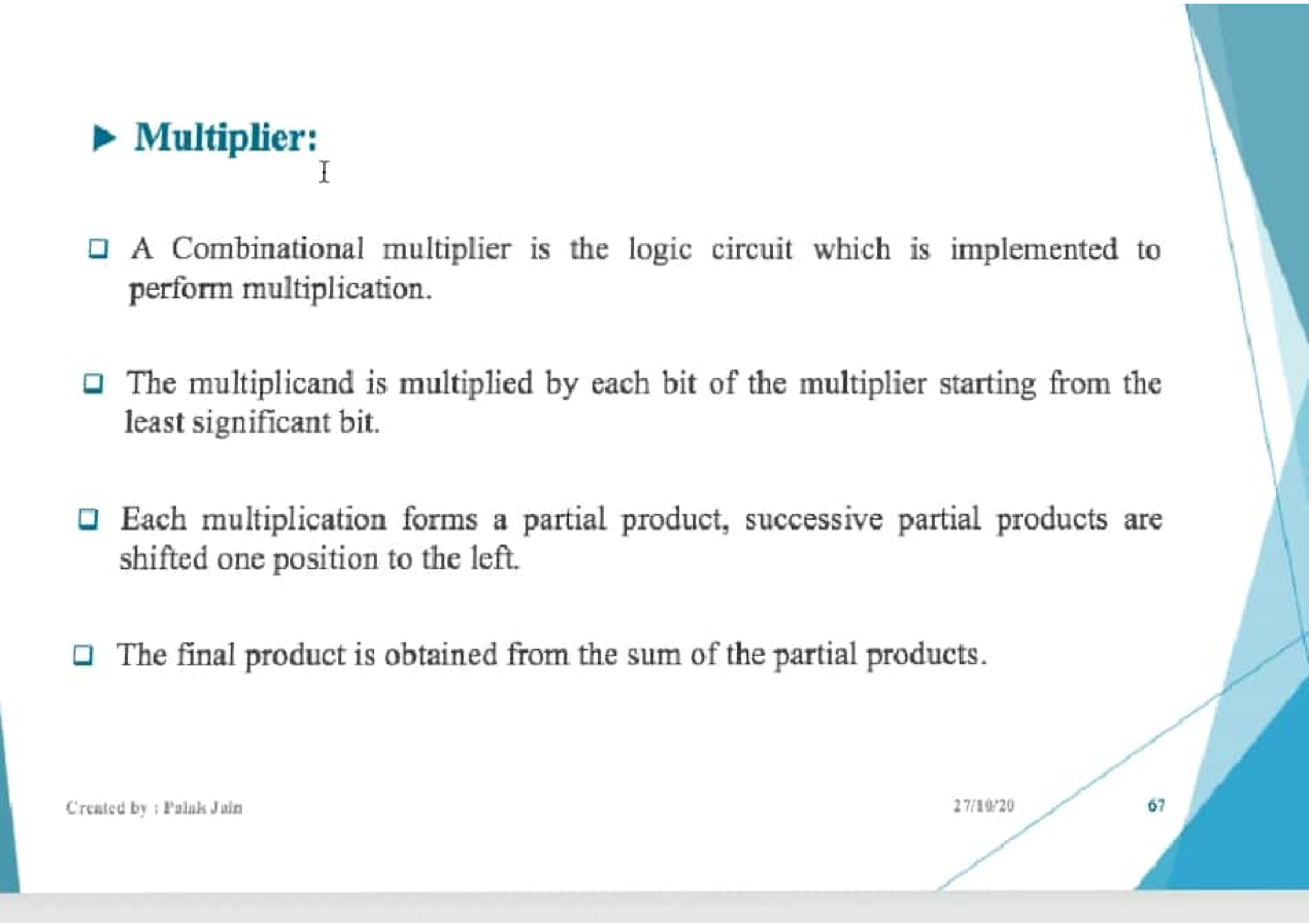
1. **Identity Comparator :** an *Identity Comparator* is a digital comparator with only one output terminal for when A = B, either A = B = 1 (HIGH) or A = B = 0 (LOW)
2. **Magnitude Comparator :** a *Magnitude Comparator* is a digital comparator which has three output terminals, one each for equality, A = B greater than, A > B and less than A < B

1-bit Digital Comparator Circuit



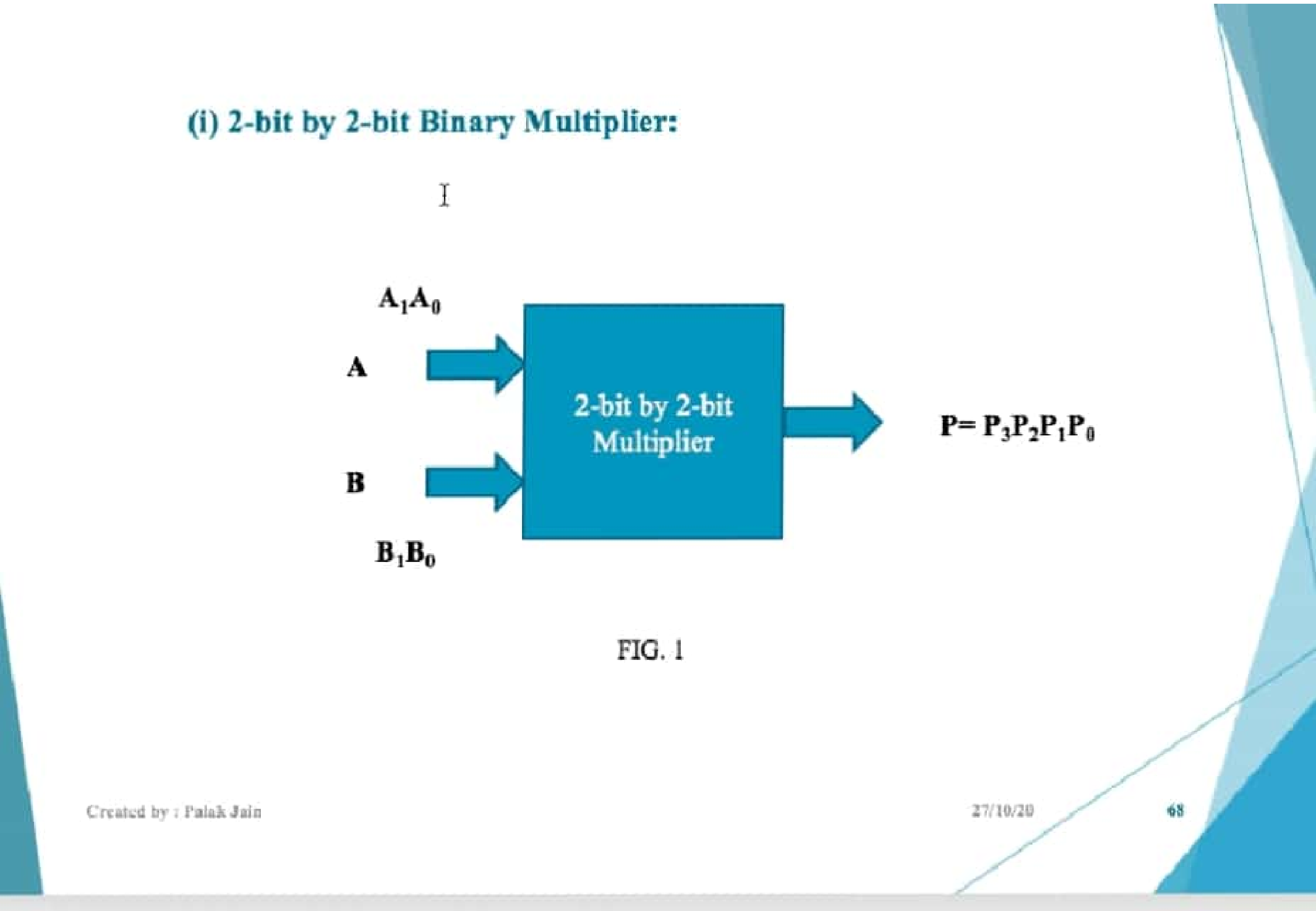
Digital Comparator Truth Table

Inputs		Outputs		
B	A	A > B	A = B	A < B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0



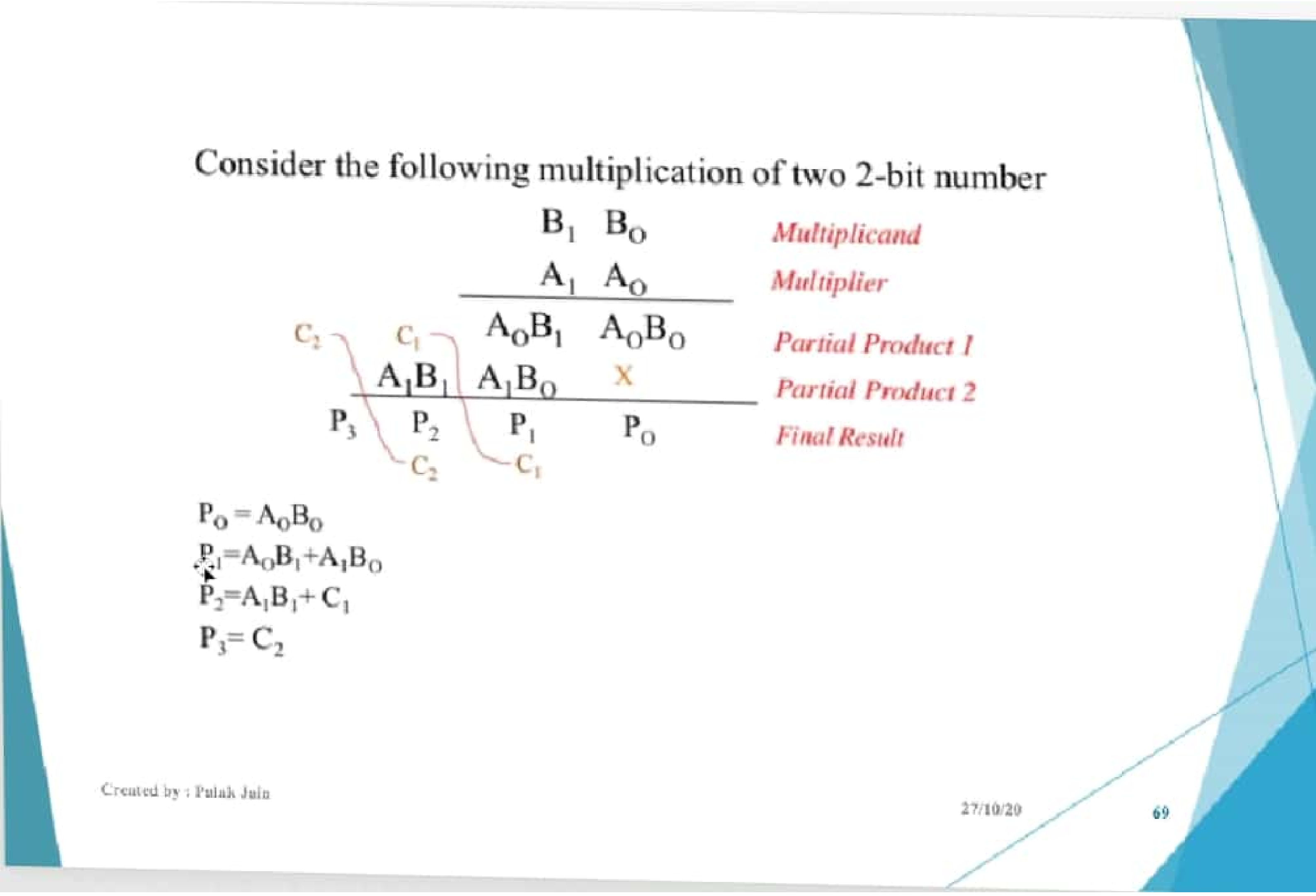
▶ Multiplier:

- □ A Combinational multiplier is the logic circuit which is implemented to perform multiplication.
- □ The multiplicand is multiplied by each bit of the multiplier starting from the least significant bit.
- □ Each multiplication forms a partial product, successive partial products are shifted one position to the left.
- □ The final product is obtained from the sum of the partial products.



(i) 2-bit by 2-bit Binary Multiplier:

FIG. 1



Consider the following multiplication of two 2-bit number

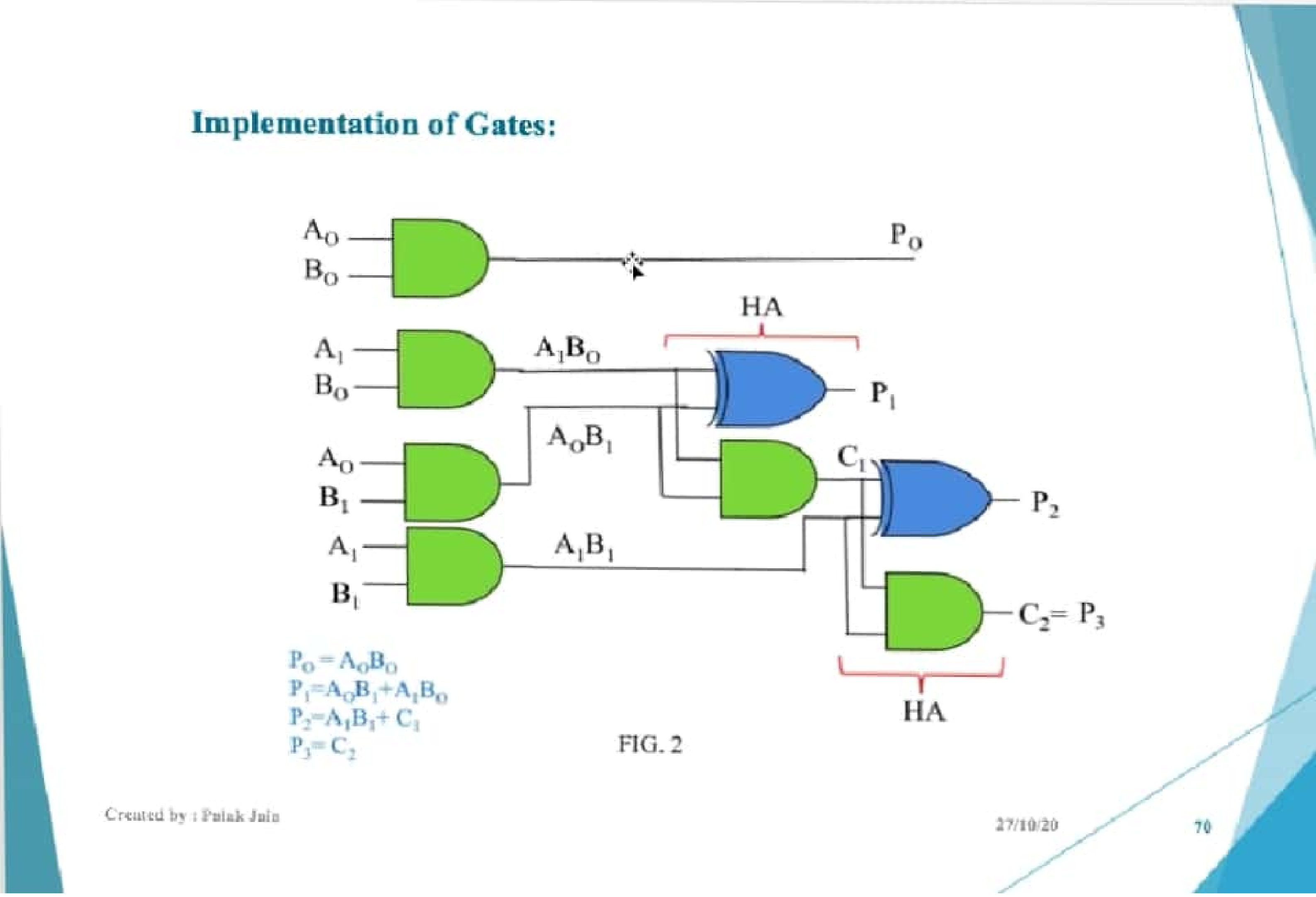
$$\begin{array}{ccccl} & & B_1 & B_0 & \textit{Multiplicand} \\ & & A_1 & A_0 & \textit{Multiplier} \\ \cline{3-4} C_2 & C_1 & A_0B_1 & A_0B_0 & \textit{Partial Product 1} \\ & A_1B_1 & A_1B_0 & X & \textit{Partial Product 2} \\ \hline P_3 & P_2 & P_1 & P_0 & \textit{Final Result} \\ & C_2 & C_1 & & \end{array}$$

$P_0 = A_0B_0$

$P_1 = A_0B_1 + A_1B_0$

$P_2 = A_1B_1 + C_1$

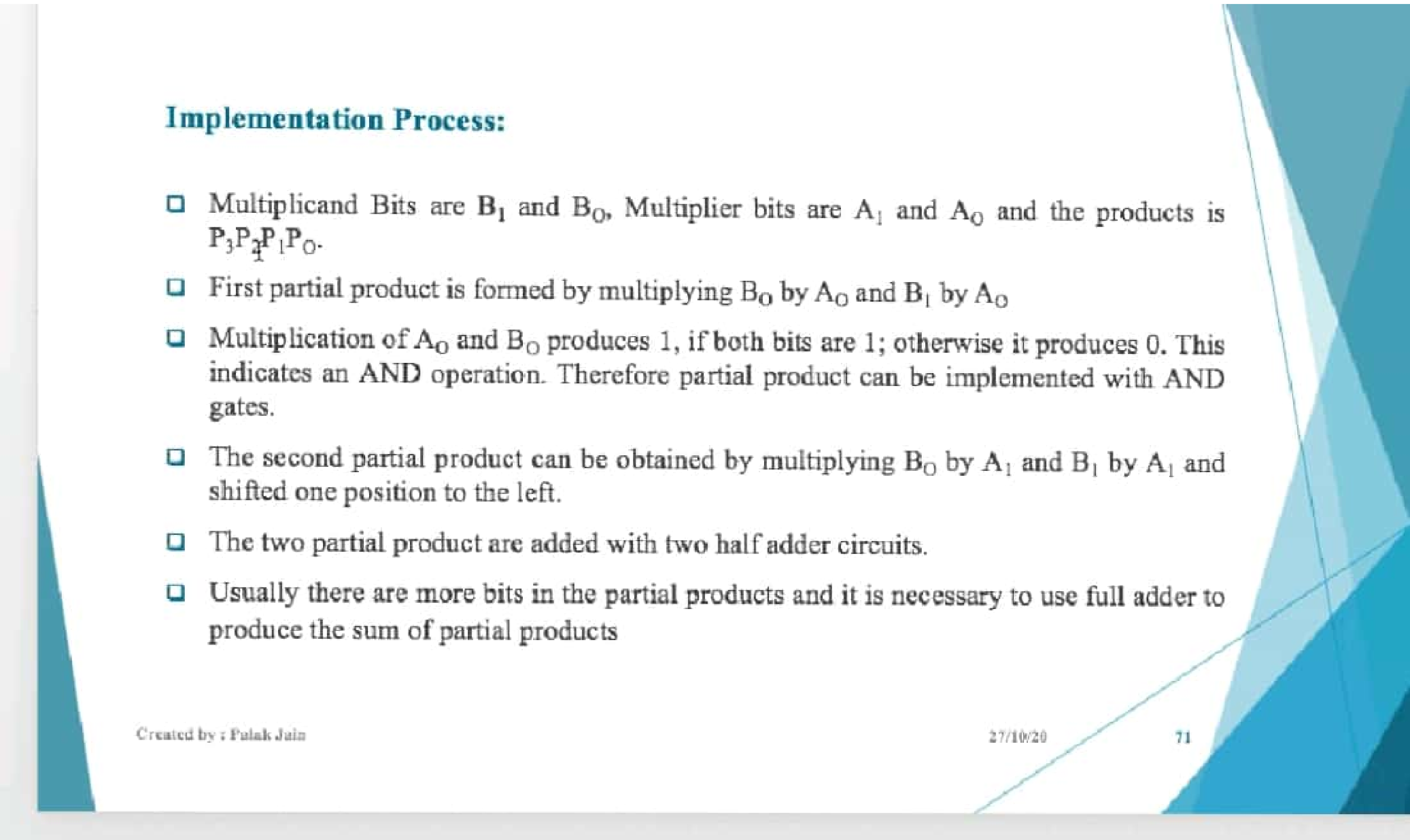
$P_3 = C_2$



Implementation of Gates:

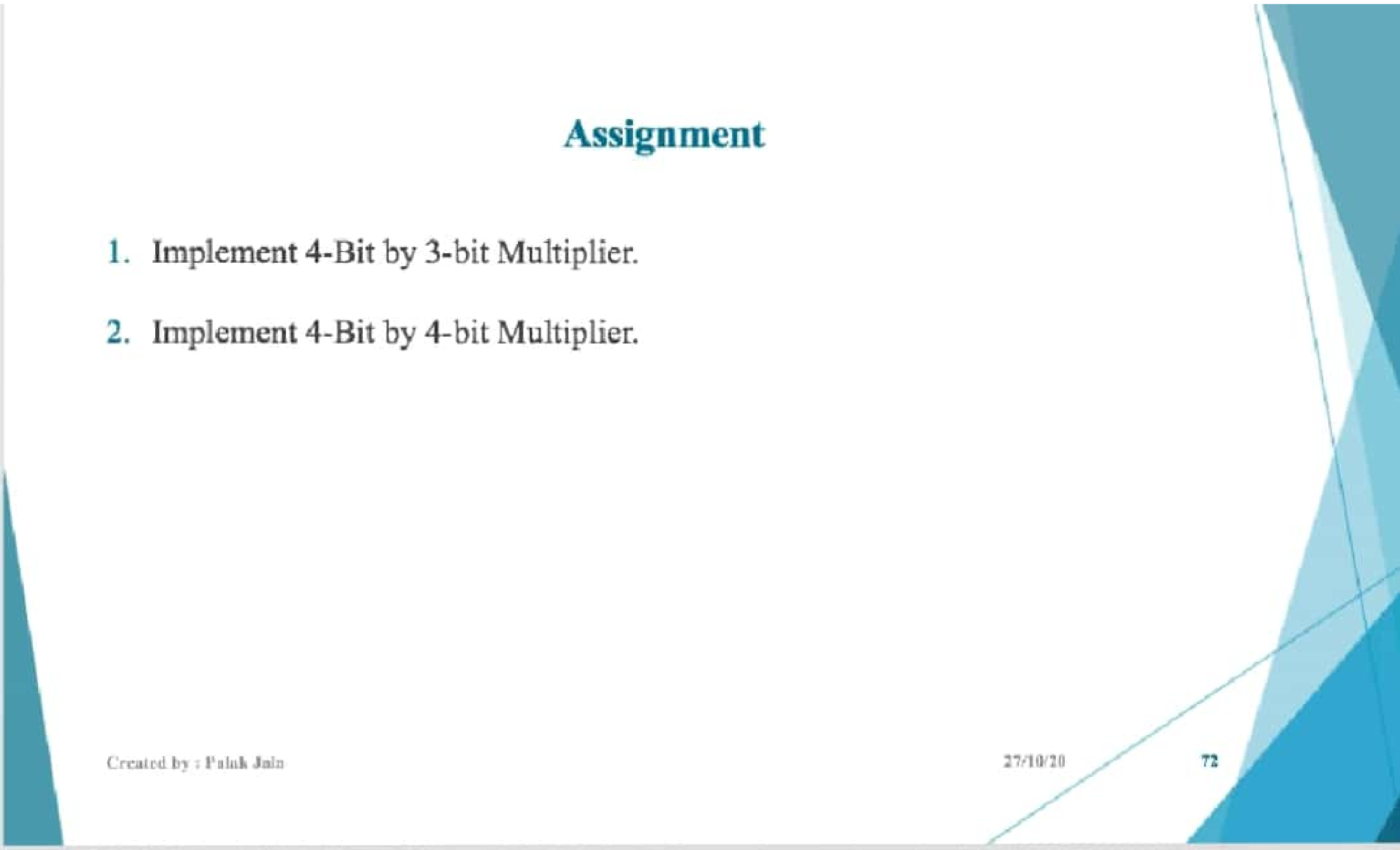
$P_O = A_OB_O$
$P_1 = A_OB_1 + A_1B_O$
$P_2 = A_1B_1 + C_1$
$P_3 = C_2$

FIG. 2



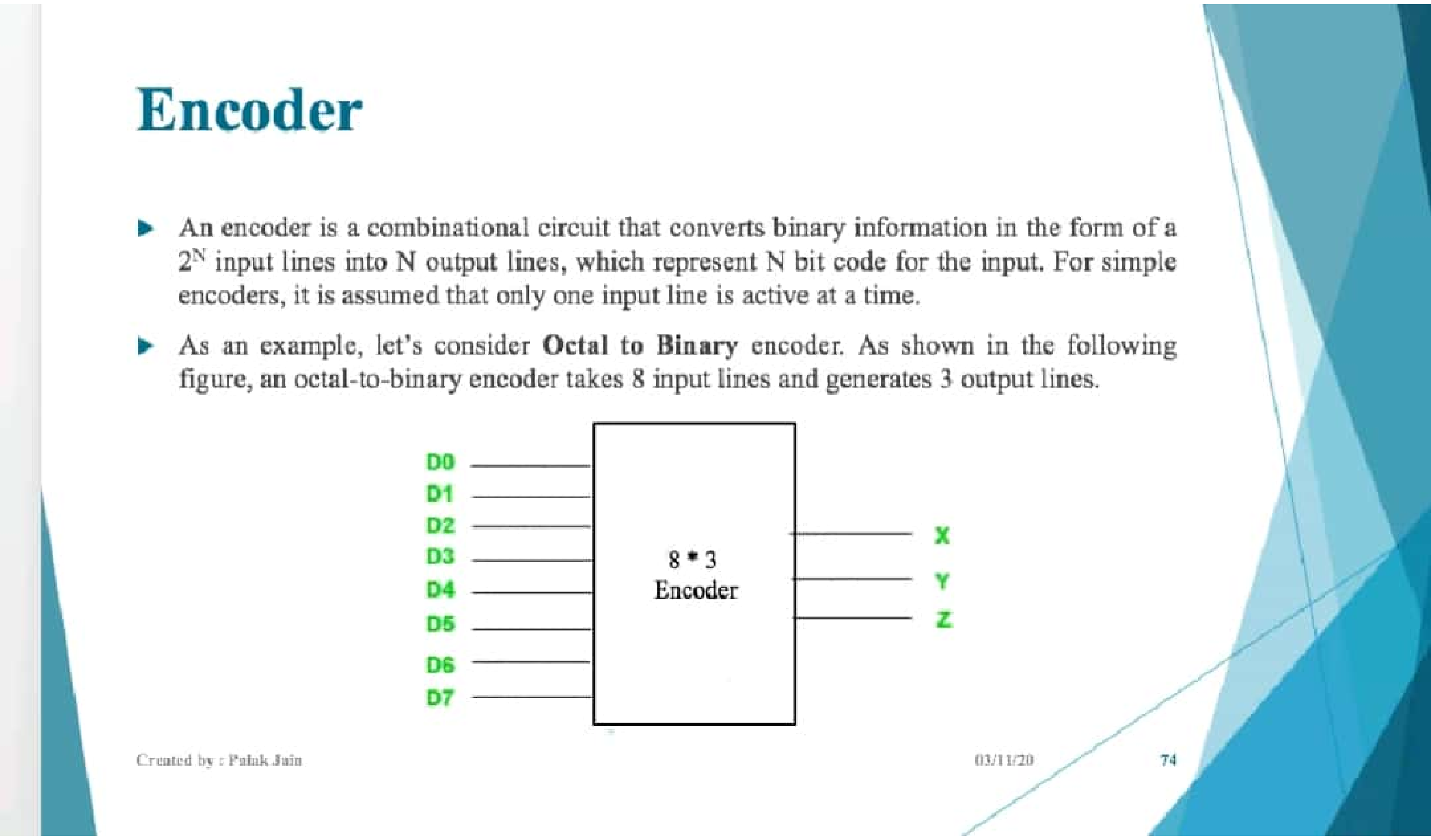
Implementation Process:

- Multiplicand Bits are B_1 and B_O, Multiplier bits are A_1 and A_O and the products is $P_3P_2P_1P_O$.
- First partial product is formed by multiplying B_O by A_O and B_1 by A_O
- Multiplication of A_O and B_O produces 1, if both bits are 1; otherwise it produces 0. This indicates an AND operation. Therefore partial product can be implemented with AND gates.
- The second partial product can be obtained by multiplying B_O by A_1 and B_1 by A_1 and shifted one position to the left.
- The two partial product are added with two half adder circuits.
- Usually there are more bits in the partial products and it is necessary to use full adder to produce the sum of partial products



Assignment

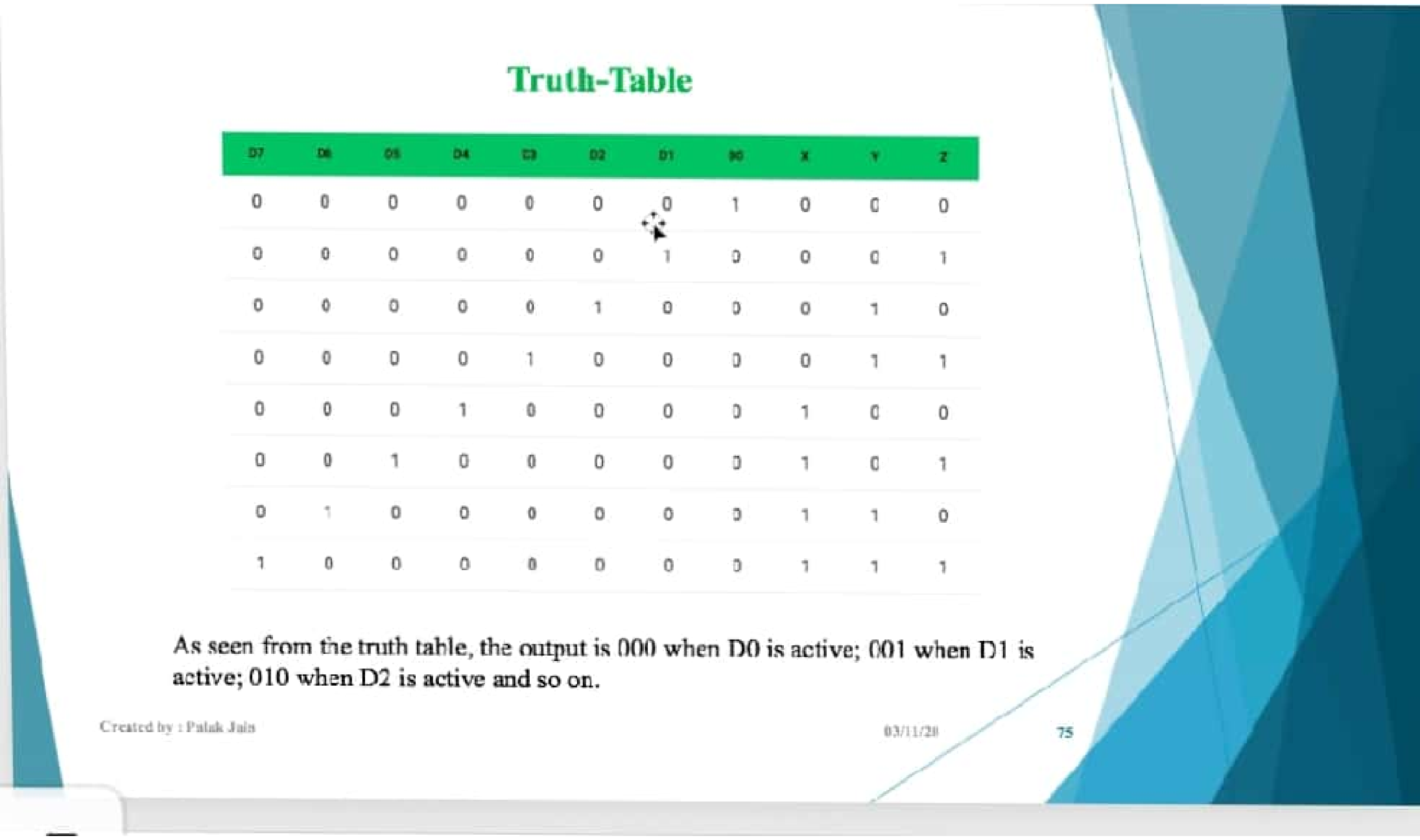
1. Implement 4-Bit by 3-bit Multiplier.
2. Implement 4-Bit by 4-bit Multiplier.



Encoder

- An encoder is a combinational circuit that converts binary information in the form of a 2^N input lines into N output lines, which represent N bit code for the input. For simple encoders, it is assumed that only one input line is active at a time.
- As an example, let's consider **Octal to Binary** encoder. As shown in the following figure, an octal-to-binary encoder takes 8 input lines and generates 3 output lines.

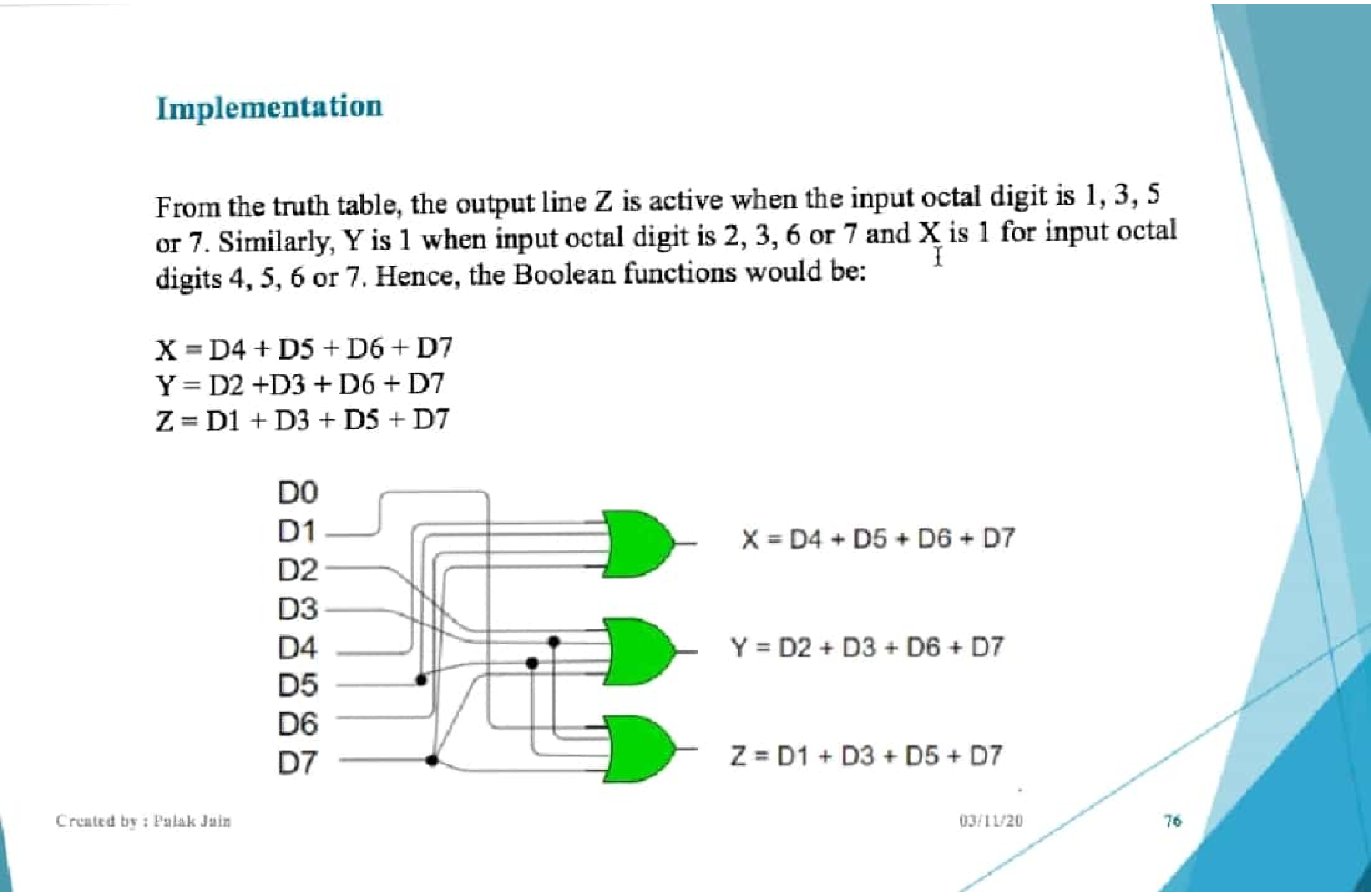
Created by : Palak Jain



Truth-Table

D7	D6	D5	D4	D3	D2	D1	D0	X	Y	Z
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

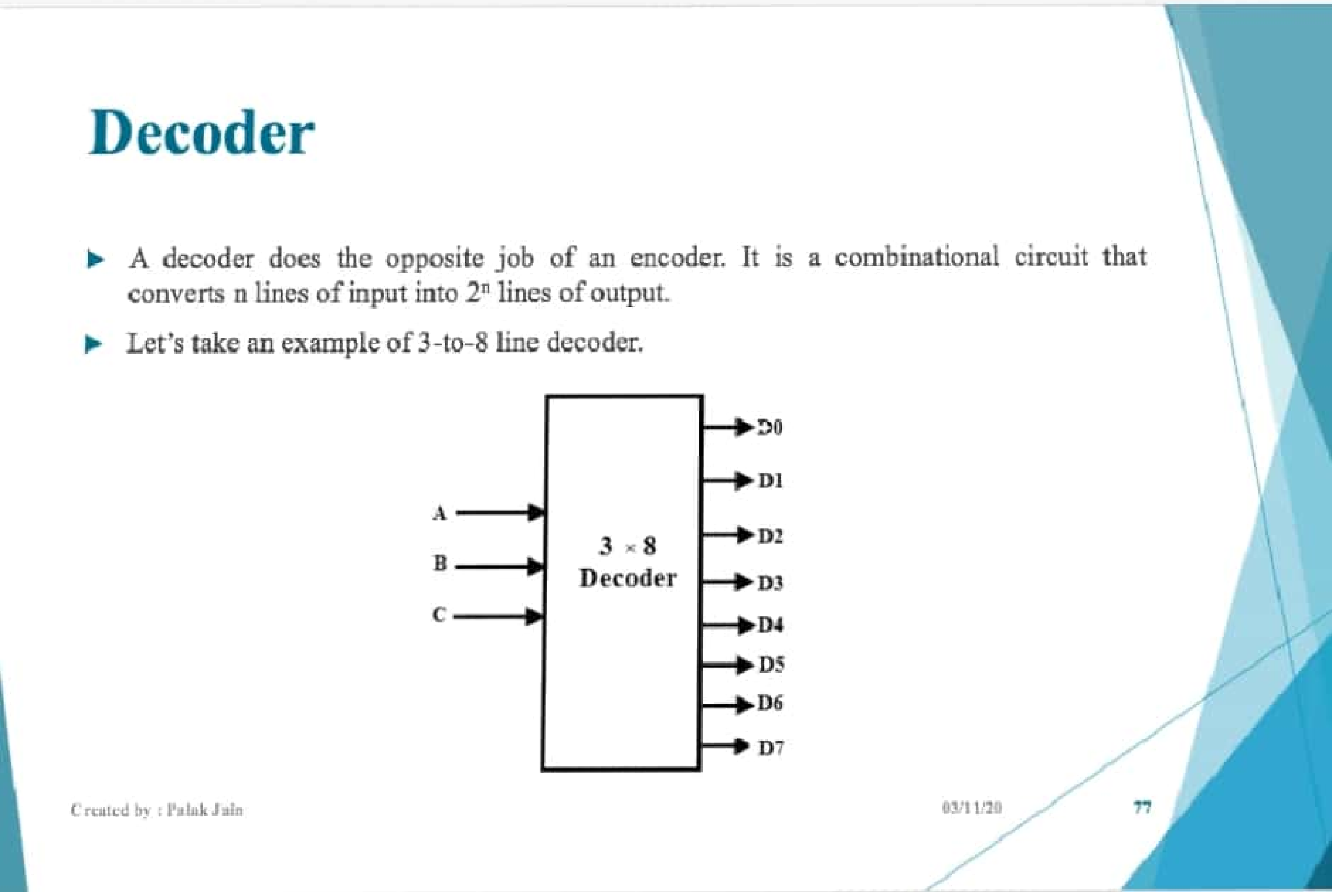
As seen from the truth table, the output is 000 when D0 is active; 001 when D1 is active; 010 when D2 is active and so on.



Implementation

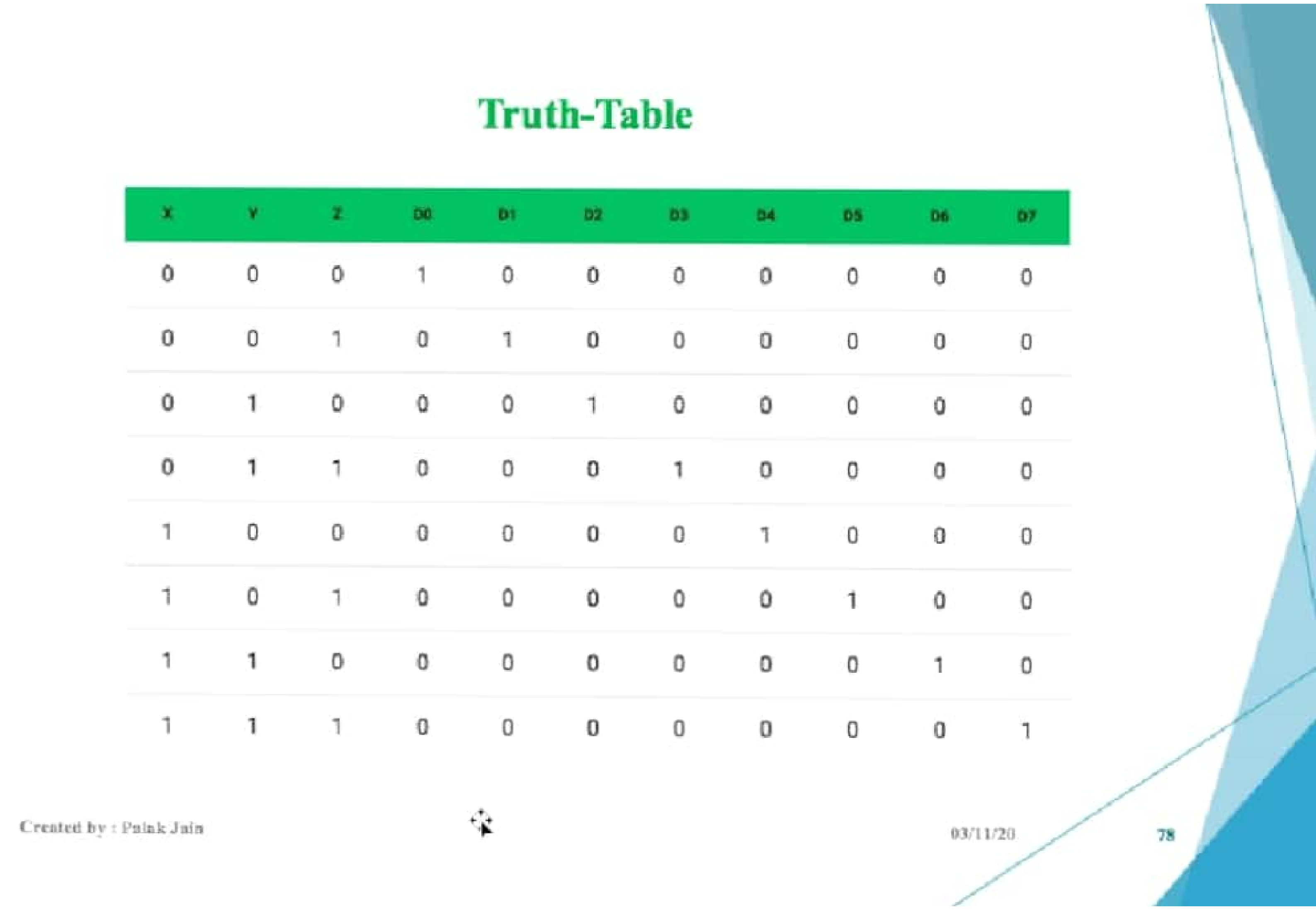
From the truth table, the output line Z is active when the input octal digit is 1, 3, 5 or 7. Similarly, Y is 1 when input octal digit is 2, 3, 6 or 7 and X is 1 for input octal digits 4, 5, 6 or 7. Hence, the Boolean functions would be:

X = D4 + D5 + D6 + D7
Y = D2 +D3 + D6 + D7
Z = D1 + D3 + D5 + D7



Decoder

- A decoder does the opposite job of an encoder. It is a combinational circuit that converts n lines of input into 2^n lines of output.
- Let's take an example of 3-to-8 line decoder.



Truth-Table

X	Y	Z	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Implementation

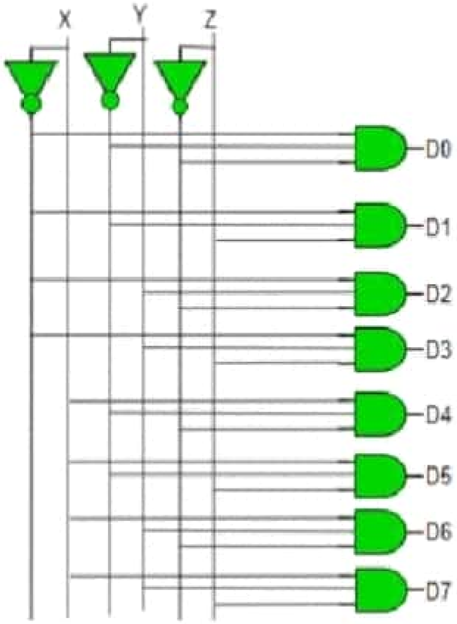
D0 is high when X = 0, Y = 0 and Z = 0.

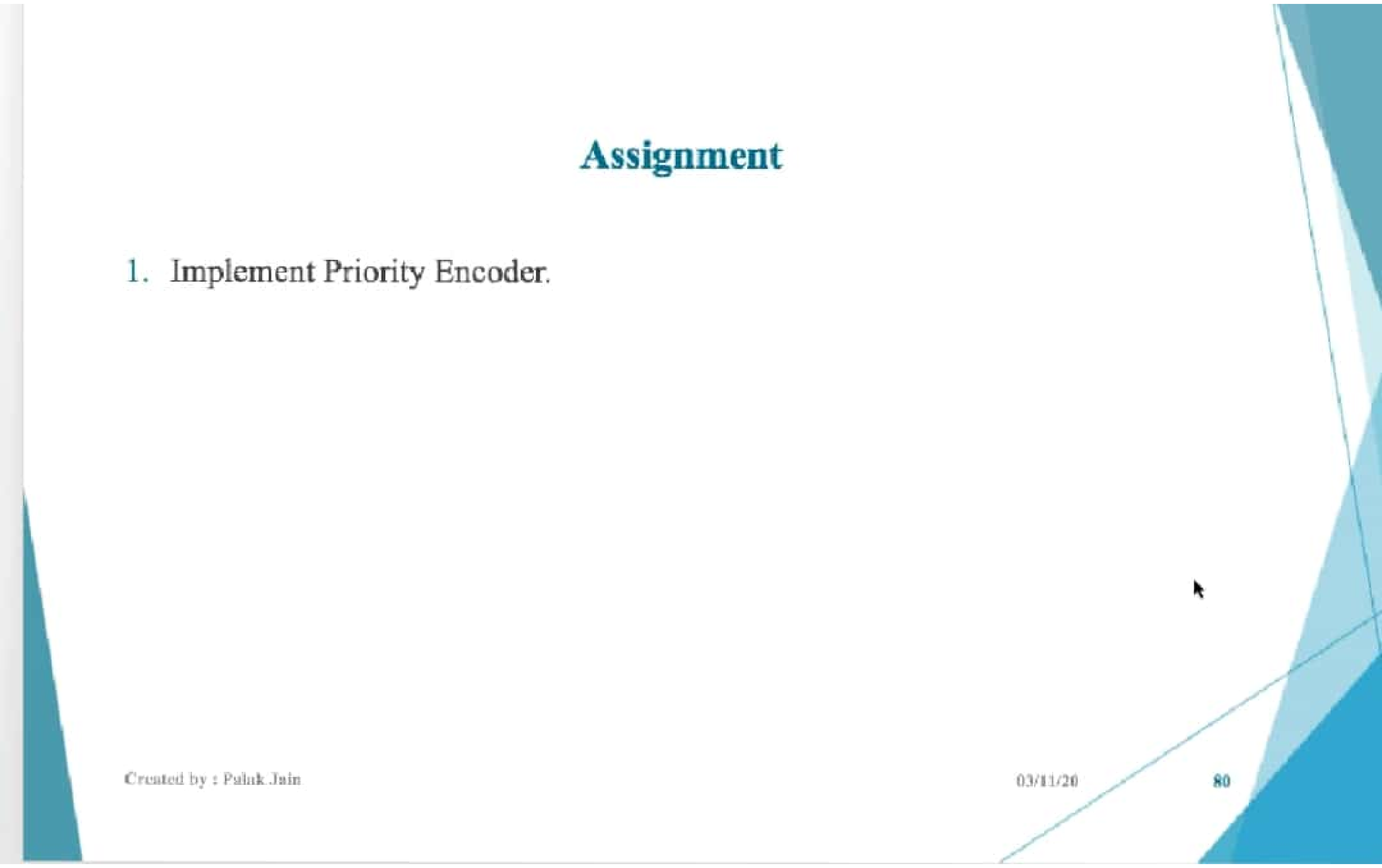
Hence,

D0 = X' Y' Z'

Similarly,

D1 = X' Y' Z
D2 = X' Y Z'
D3 = X' Y Z
D4 = X Y' Z'
D5 = X Y' Z
D6 = X Y Z'
D7 = X Y Z





Assignment

1. Implement Priority Encoder.

CS 236 (PCC): DIGITAL LOGIC DESIGN

Cr. Hrs. 5 (3 +0+2)

	L	T	P
Credit	3	0	2
Hours	3	0	4

Course Outcome: At the end of the course the student will be able to:

CO1: Demonstrate the principles of number system, binary codes and logic families.

CO2: Analyze and design combinational circuits using standard gates and minimization methods.

CO3: Efficiently optimize and minimize logic function using k-maps.

CO4: Design common digital circuit such as - decoders, multiplexers, encoder, demultiplexer etc.

CO5: Analyze and design sequential circuit such as flip-flops, counters, registers etc.

Unit-I

Computer Number Systems and Codes: Number Systems and their conversion, Negative Numbers representation, Codes; Binary Coded Decimal number (BCD), Excess-3 BCD Code, Gray Codes representation.

Logic families: Characteristics of digital ICs, Diode-Transistor Logic (DTL) Transistor- Transistor Logic (TTL) TTL output structures: Totem pole output, Darlington Output, Open-Collector Outputs. Wired Logic, Tri-State Logic, Emitter-Coupled Logic, Metal-Oxide Semiconductor (MOS) Logic, Complementary metal oxide semiconductor (CMOS) Logic.

Unit-II

Logical Operations, Logic Gates, and Boolean Algebra: Truth Table, Logical Operations and logic gates, Logic Circuits, Realizing Circuits From Boolean Expressions, Derived Logical Functions and Gates: The NAND Gate, The NOR Gate, The Exclusive-OR or XOR Gate, The

Exclusive-NOR, or XNOR Gate, Boolean Algebra, Boolean Algebra Theorems, De Morgan's Theorems, Duality Theorem, Universal Gates, Deriving the XOR Function, Reducing Boolean Expressions by Algebraic reduction.

Unit-III

Principles of Combinational Logic Circuits: Minterm and Maxterm designations, Canonical Forms, Karnaugh Map: Karnaugh Map upto six variables. Prime implicant (PI), Essential Prime implicant (EPI), Simplification of Boolean expressions using K-map in POS and SOP form, Incompletely Specified Functions (Don't Care Terms), Quine-McCluskey Minimization Method, Mixed (Bubble) logic Combinational Circuits. Arithmetic Circuits: Adders, Subtractor, 2-bit Full-Adder/Subtractor, Binary Parallel Adder, BCD Adder, Multiplier, Digital comparator, Decoders, Encoders, Priority Encoder, Multiplexers, Implementation of Boolean Function with Multiplexer, Demultiplexer.

Unit-IV

Sequential Logic Circuits: Latches, Flip-flops: SR (Set-Reset) Flip-Flop, Edge-Detector Circuits, Master-Slave S-R Flip-Flop, J-K flip-flop, Master-Slave J-K Flip-flop, D Flip-Flop, T Flip-flop, Conversions of flip-flops, Mealy and Moore Machines. Counters: Asynchronous (Ripple) Counters, Propagation Delay in Ripple Counter, Asynchronous Counters with Mod Numbers, Synchronous (Parallel) Counters, Design of Synchronous Counters.

Registers: Serial- in/serial- out, Serial- in/parallel- out, Parallel- in/serial-out, Parallel- in/parallel- out, Bi-directional shift register, Shift-registers counters (Ring Counter, Johnson Counter).

Lab/ Practicals

- Design and implement various logic gates such as derived and universal logic gates.
- Design and implement combinational circuits such as adders, Subtractor, encoder, decoder, multiplexer, demultiplexer, comparators.
- Design and implement sequential circuits such as flip flops, counters, and registers.

Text Books/References

1. "Digital Logic and Computer Design", M. Morris Mano, Prentice-Hall.
2. "Digital Fundamentals", Thomas L. Floyd., Pearson Education.