WEEK 3-2017

Combinational circuit II

WEEK 3-2017

Combinational circuit II

Dear students,

This year the School will be offering mentoring sessions to the following courses. These mentoring sessions commence next week (week 4) and available throughout the semester. Please make use of this opportunity to clarify your doubts face-to-face with the available academic staff during these sessions. The sessions will be available during the following times in room 552, Hilmer building (E10). These are open sessions and **DO NOT** require prior appointment. Please feel free to drop in!

Day	Courses
Monday 4-6pm	ELEC1111, ELEC2134, ELEC3104
Tuesday 4-6pm	ELEC1111, ELEC2141, ELEC3106
Wednesday 4-6pm	ELEC1111, ELEC2134, ELEC3115

I am reaching out to you all on the possibility that you could do a last-minute call out to some of your students before **EOD Wednesday March 14th, 2018** to remind them about applications for the ***Projects and Pitch Fair*** event being held on **March 20th, 2018** at the MCIC. I realize this is a little last minute so if you can advertise in any way that would be great! **Applications close: 11:59PM THIS Wednesday MARCH 14th, 2018**

This year we have made this event even bigger by inviting some of our Industry Partners to be present as well as students and current Student-Led Project groups. New student projects/start-up ideas are invited to ***Pitch*** to a panel of judges and then we will follow by having our industry partners go on stage and be able to get that quick pitch exposure by letting the students know what opportunities in terms of career, internships etc. they can offer!

There is $2500 in prize funding available to be awarded!

I have attached some information at the bottom for you to send out as an email to students, as well as a quick reference PowerPoint slide to show in your lecture core courses!

If students have any questions, please give them this email to contact: engprojects@unsw.edu.au

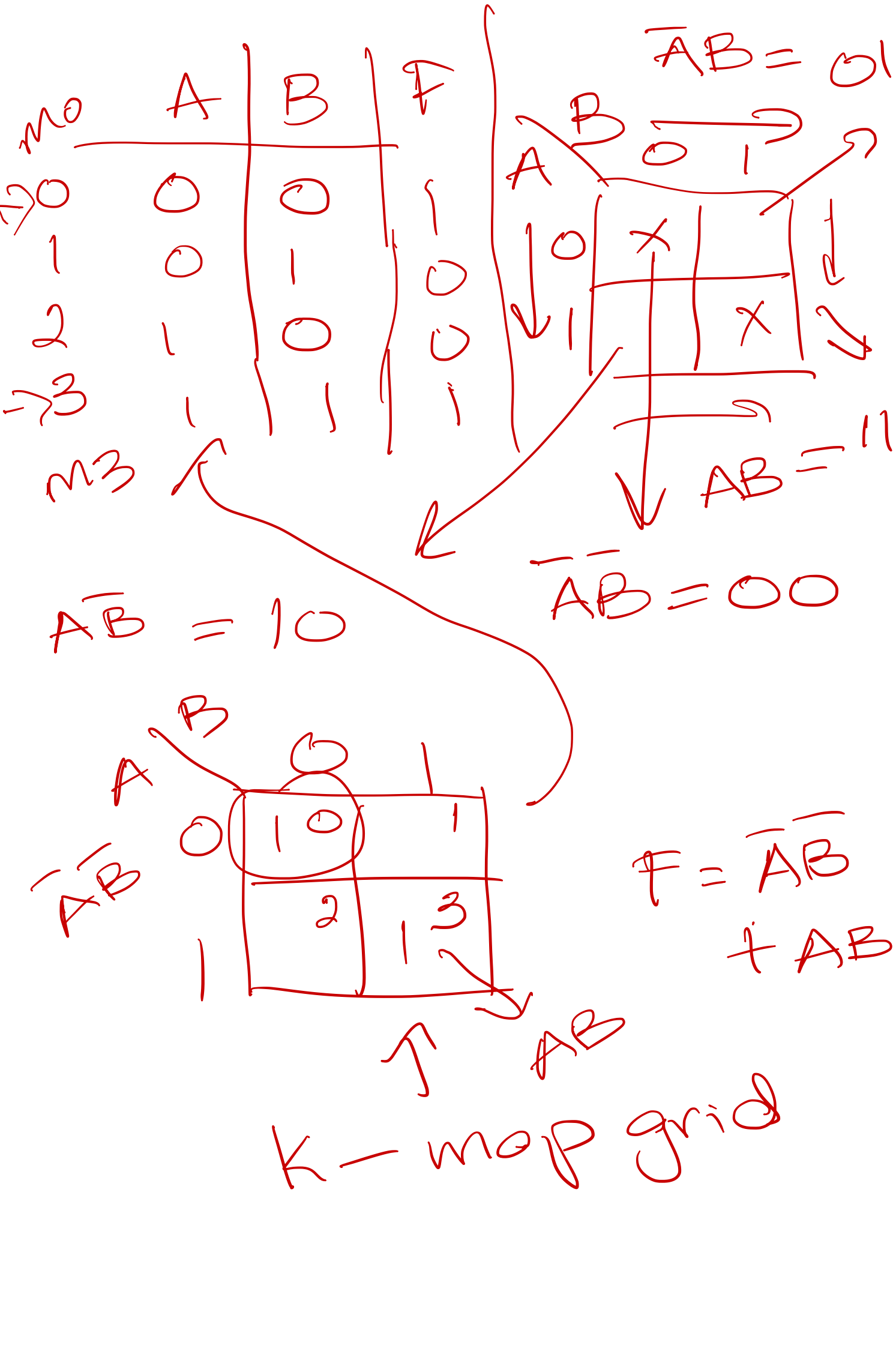
Thanks so much for your support and have a great evening!

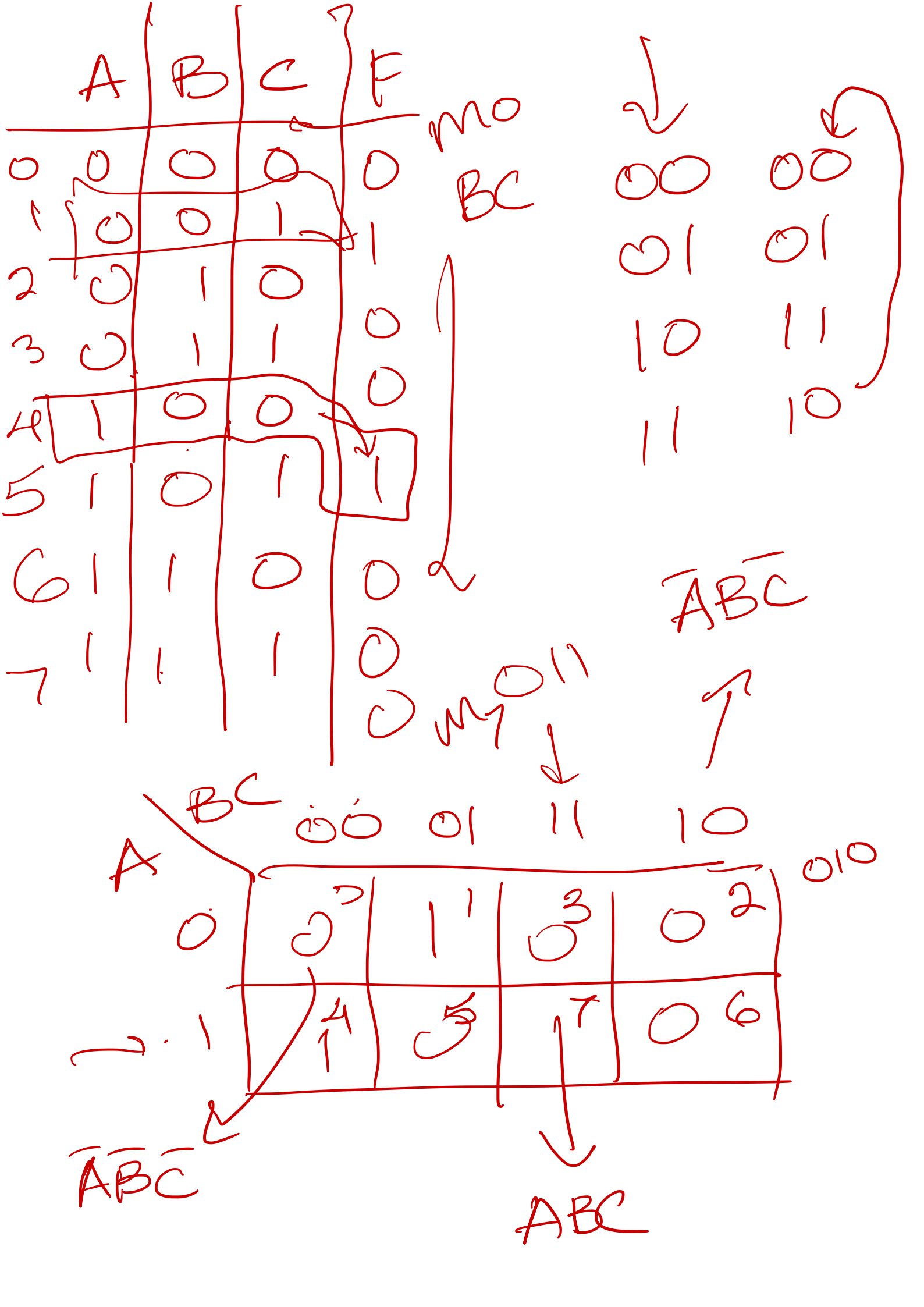
Two-level circuit optimization

- **Boolean functions directly dictates the logic circuit implementation.**
- **Important to device a way that leads to the simplest logic circuit implementation.**
- **Although Boolean expressions can be simplified by algebraic manipulation, the procedure is awkward and it is hard to confirm if the simplest expression is obtained.**
- **A map method is often used to arrive at optimized Boolean equations.**

Karnaugh map or K-map

- The map method is commonly referred as Karnaugh map or K-map.
- K-map provides systematic and straight forward procedure to simplify Boolean functions up to four variables.
- Consists of squares each of which corresponds to each minterm (a row of a truth table) of the Boolean function.
- The Boolean function is identified in the map by those squares for which it has value 1.


m0 A B F
0 0 0 1
1 0 1 0
2 1 0 0
3 1 1 1
m3
$\bar{A}B = 01$
B
A
0 1
0 X
1 X
AB = 11
$\bar{A}\bar{B} = 00$
$A\bar{B} = 10$
B
A
0 1
0 1 0 1
1 2 3
$\bar{A}\bar{B}$
AB
$F = \bar{A}\bar{B} + AB$
K-map grid


A B C F
0 0 0 0 0
1 0 0 1 1
2 0 1 0 0
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
m0
BC
00 00
01 01
10 11
11 10
m1
m3
ABC
ABC
BC
00 01 11 10
A
0 0 1 3 2
1 4 5 7 6
010
ABC
ABC

Karnaugh map or K-map

- The optimized expression obtained from K-map is always expressed in sum of products or product of sums forms.
- Thus, the expression is realizable using two-level logic circuit implementation.
- A two-variable K-map has the following structure

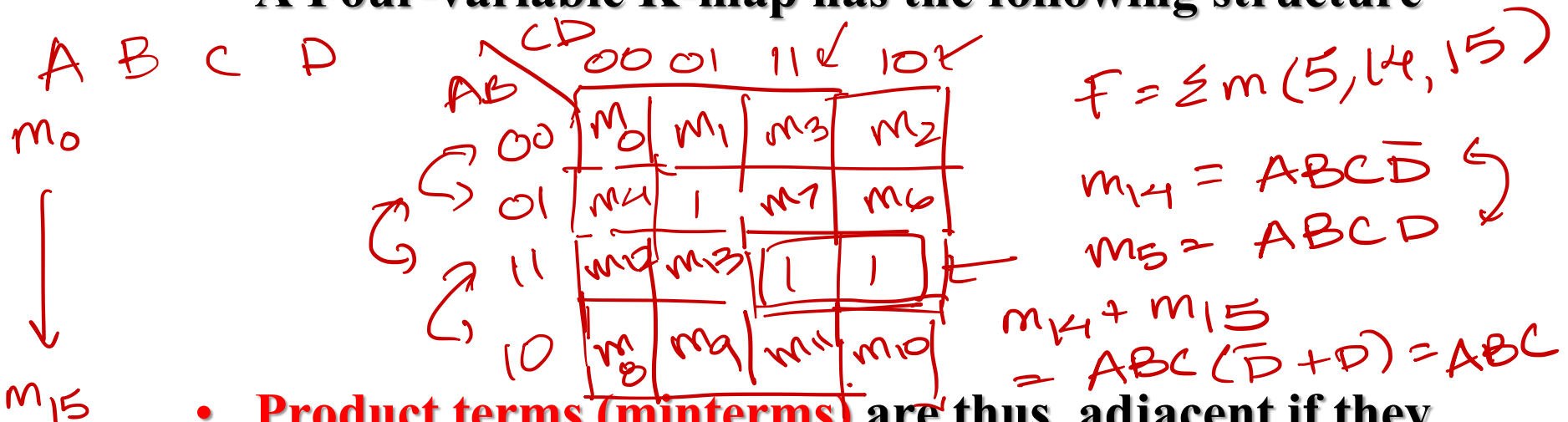
Karnaugh map or K-map

- A three-variable K-map has the following structure

- Note that the adjacent squares to east, west, north and south of a given square represent binary combinations that differ by only one bit. That is gray code format.
- Two squares can be adjacent even if they do not physically appear adjacent in the K-map. For example, m_0 and m_2, m_4 and m_6, in a three variable K-map.

Karnaugh map or K-map

- **A Four-variable K-map has the following structure**



- **Product terms (minterms) are thus adjacent if they differ only in one literal, which is complemented in one and uncomplemented in another.**
- **m_0, m_4,m_{12},m_8 are adjacent to m_2,m_6,m_{14},m_{10}**
- **m_0,m_1,m_3,m_2 are adjacent to m_8, m_9,m_{11}, m_{10}**
- **Two adjacent squares, representing minterms of a function, can be combined to form a product term with one less variable.**

Steps in using K-maps

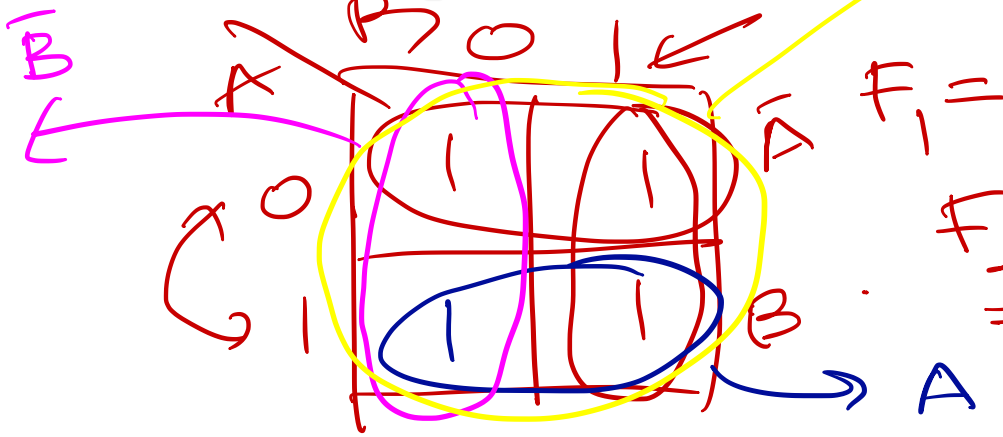
1. **To enter the function on the K-map**

   The function may be given in the form of truth table, a sum of minterms or a sum of product expressions.

   This is done by placing '1' in the squares for which the function has a logical '1' for the minterm corresponding to the square.

$F_3 = \bar{A}B + AB = (\bar{A}+A)B = B$

Example 1: $F = \sum m(0,1,3)$ or $F = \bar{A} + AB$



$F_1 = \bar{A}\bar{B} + \bar{A}B = \bar{A}$

$F_2 = B$

$F = \bar{A} + B$

Steps in using K-maps

2. **To identify collections of squares on the map to be considered for the simplified expression.**

   - **The collection of squares with logical 1 forms a rectangle that represents a simplified expression.**

   - **The goal is to find the fewest of such rectangles that cover all the squares marked with 1s**

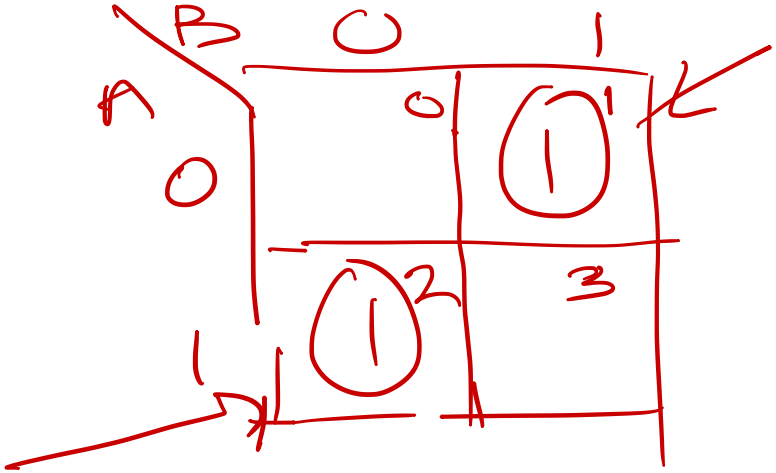
Example 1: F = $\sum m(0,1,3)$ or F = $\overline{A} + AB$

Steps in using K-maps

3. **To read off the sum of products expression corresponding to the constructed rectangles in the map.**

Example 1: $F = \sum m(0, 1, 3)$ or $F = \overline{A} + AB$

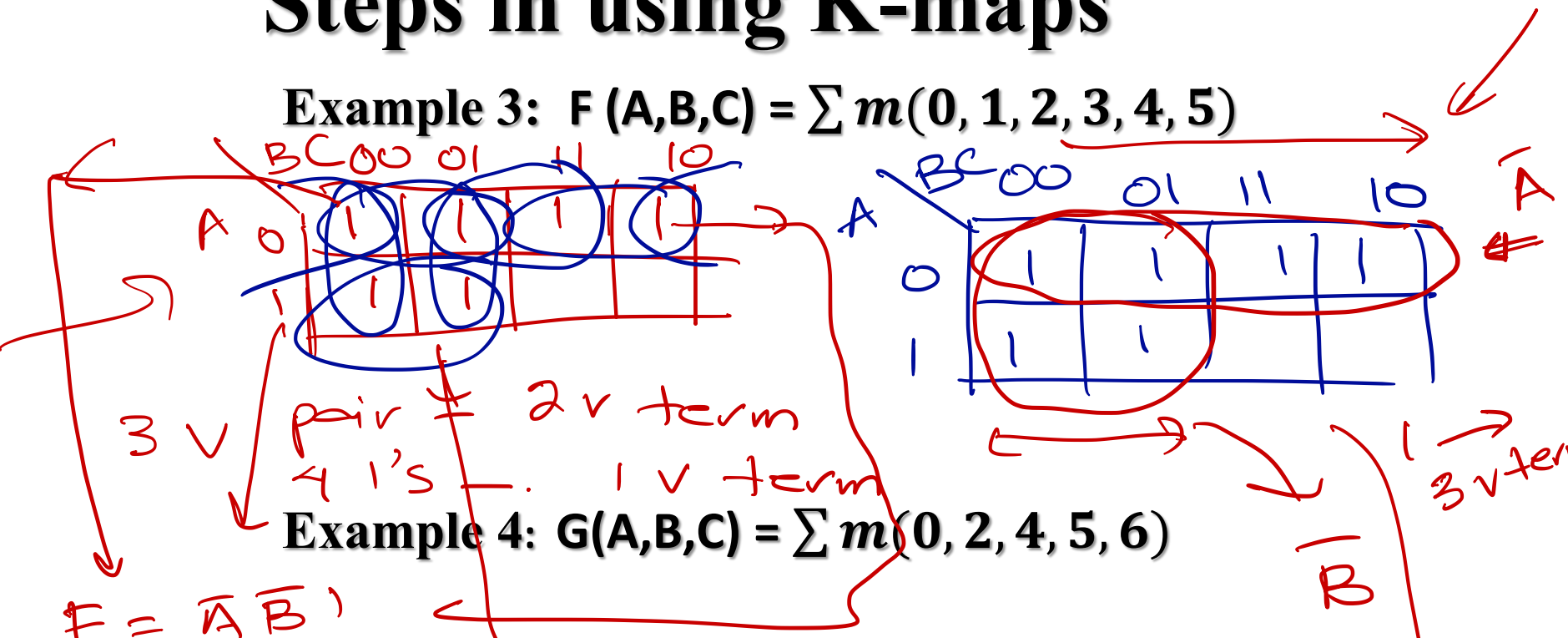
Example 2: $G = \sum m(1, 2)$



$G = \overline{A}B + A\overline{B}$

Steps in using K-maps

Example 3: $F(A,B,C) = \sum m(0,1,2,3,4,5)$



Example 4: $G(A,B,C) = \sum m(0,2,4,5,6)$

$F = \bar{A}\bar{B}^1 + \bar{A}B^2 + AB^3$

$\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + \bar{A}B\bar{C}$

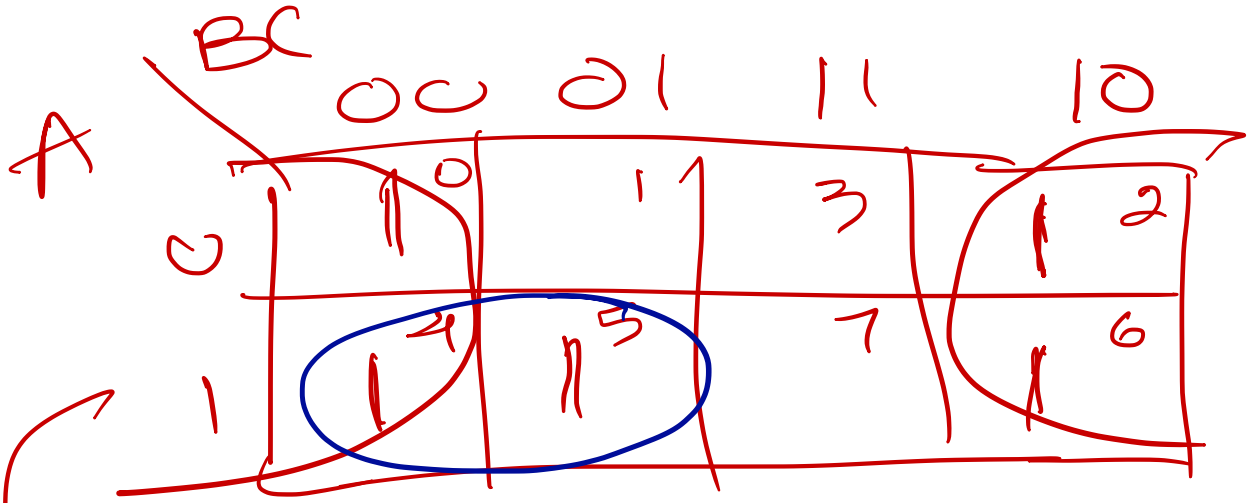
$= \bar{A}\bar{B}(\bar{C}+C) + \bar{A}BC(C+\bar{C})$

$= \bar{A}\bar{B} + \bar{A}B = \bar{A}(\bar{B}+B) = \bar{A}$

$6+2+3 = 11$

$F = \frac{\bar{A}+\bar{B}}{2}$

$G(A,B,C) = \sum m(0,2,4,5,6)$



Any combos of 4 1's?

$\bar{C} + A\bar{B} = G$

$H = A\bar{C} + \bar{A}C + AB$ ←

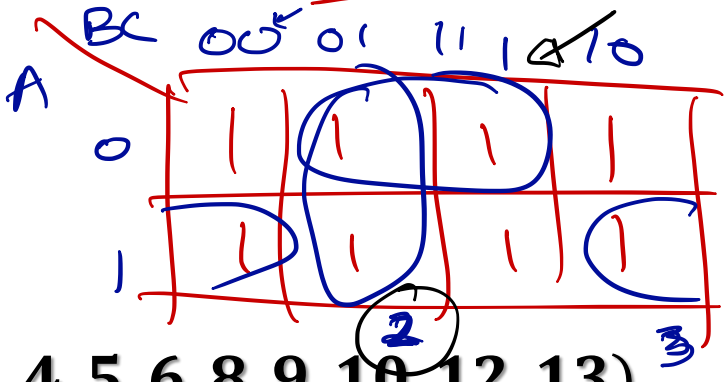
6
3
3
12

6+3+3
= 12

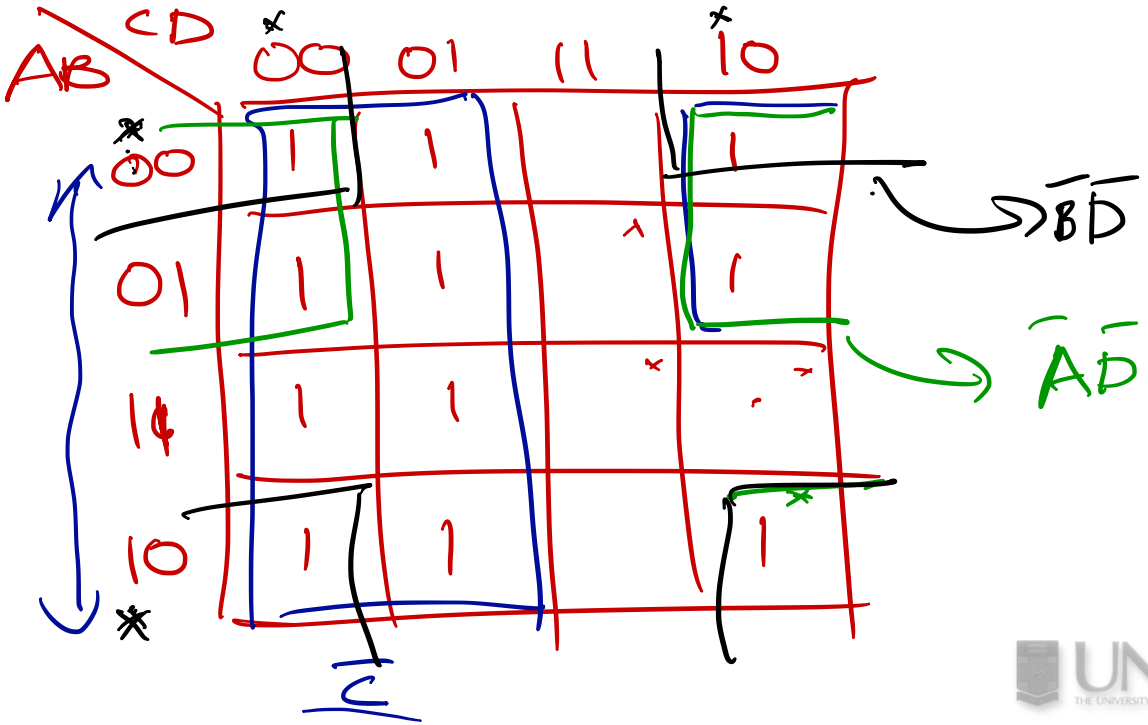
Steps in using K-maps

Example 5: H (A,B,C) = $\sum m(1,3,4,5,6)$

$H = \bar{A}C + \bar{B}C + A\bar{C}$



Example 6: F(A,B,C,D) = $\sum m(0,1,2,4,5,6,8,9,10,12,13)$

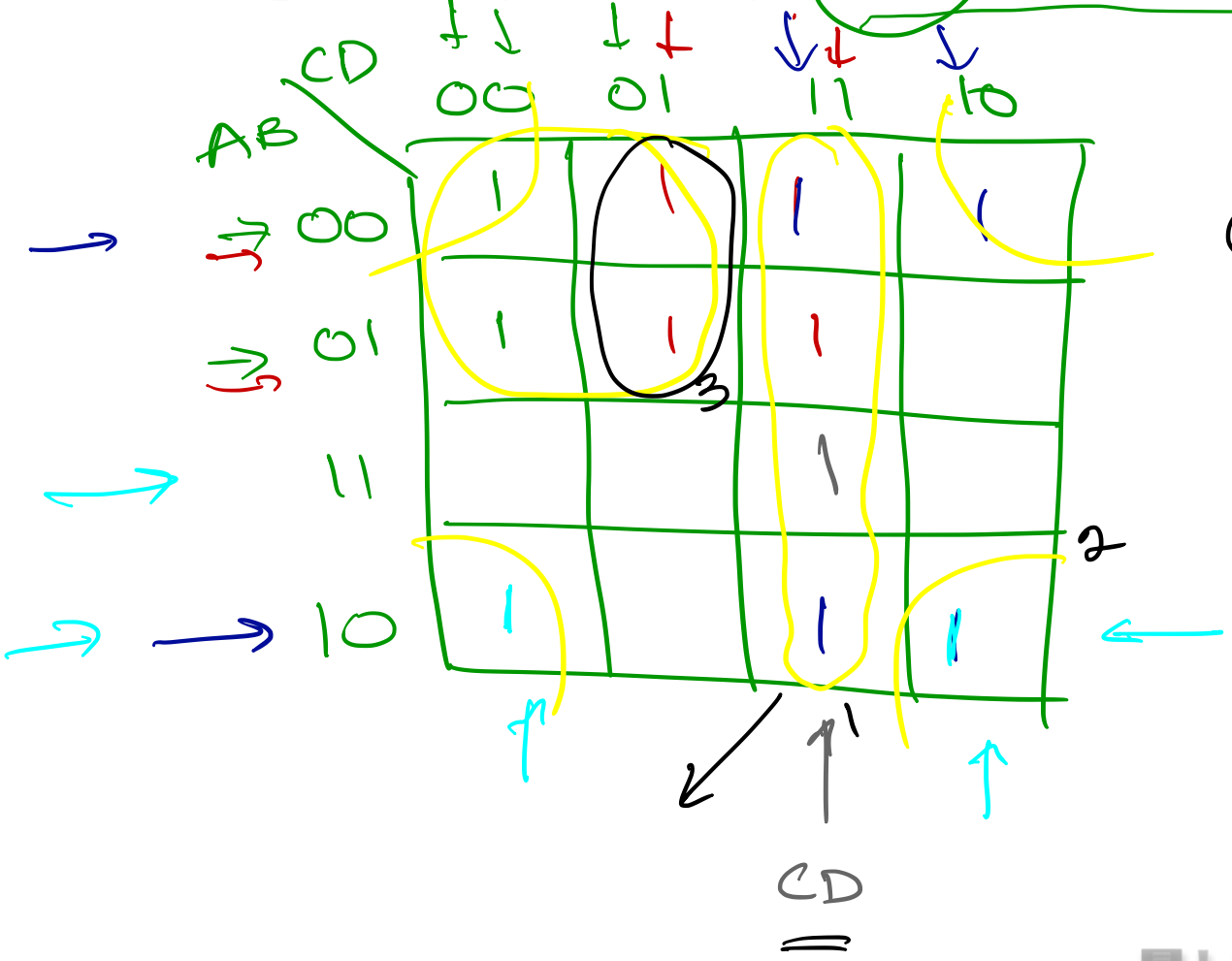


$F = \bar{C} + \bar{A}\bar{D} + \bar{B}\bar{D}$

$F = \bar{C} + \bar{D}(\bar{A}+\bar{B})$

Steps in using K-maps

Example 7: $G(A,B,C,D) = \bar{A}\bar{C}\bar{D} + \bar{A}D + \bar{B}C + CD + A\bar{B}\bar{D}$



$G = CD + \bar{B}\bar{D} + \bar{A}\bar{C}$

Map manipulation

- **Map manipulation can be made systematic if the following terms are introduced.**

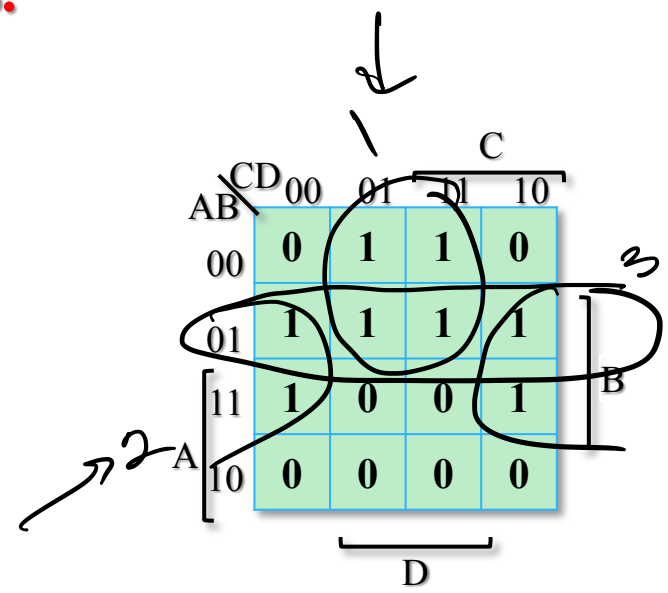
- ***Implicant*: A product term is an implicant of a function if the function has the value 1 for all miniterms of the product term.**
- ***In other words, all rectangles on a map made up of squares containing 1s correspond to implicants.***

- ***Prime implicant*: If the removals of any literal from an implicant P results in a product term that is not an implicant of the function, then P is a prime implicant.**
- ***In other words, prime implicant corresponds to a rectangle containing as many squares as possible.***

Map manipulation

- ***Essential prime implicant*: If a miniterm of a function is only included in one prime implicant, then that prime implicant is said to be *essential*.**
- ***In other words, essential prime implicant is a prime implicant containing at least a square with a 1 that is not included in any other rectangle or prime implicants.***

Example1:

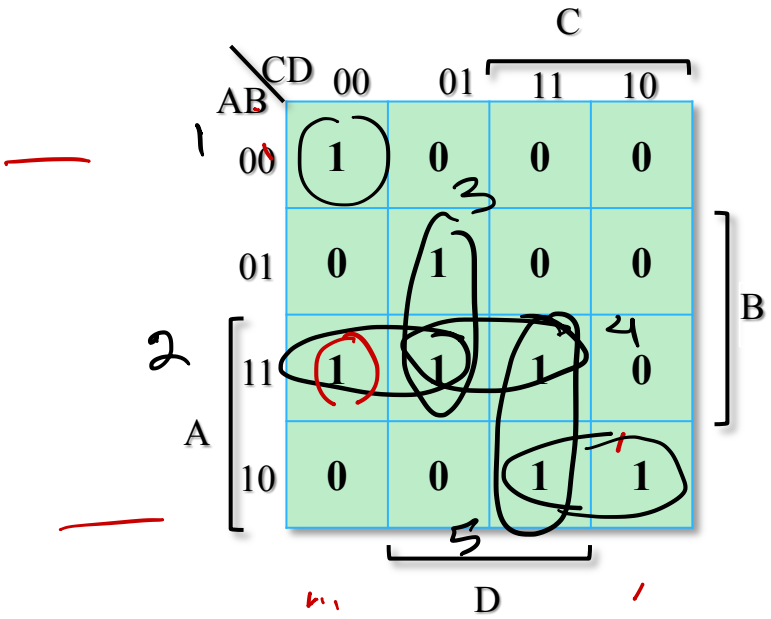


Map manipulation

Example2:

Prime

4
5



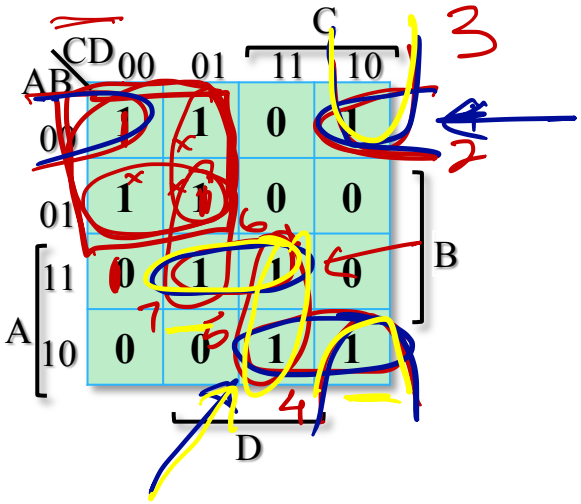
Essential

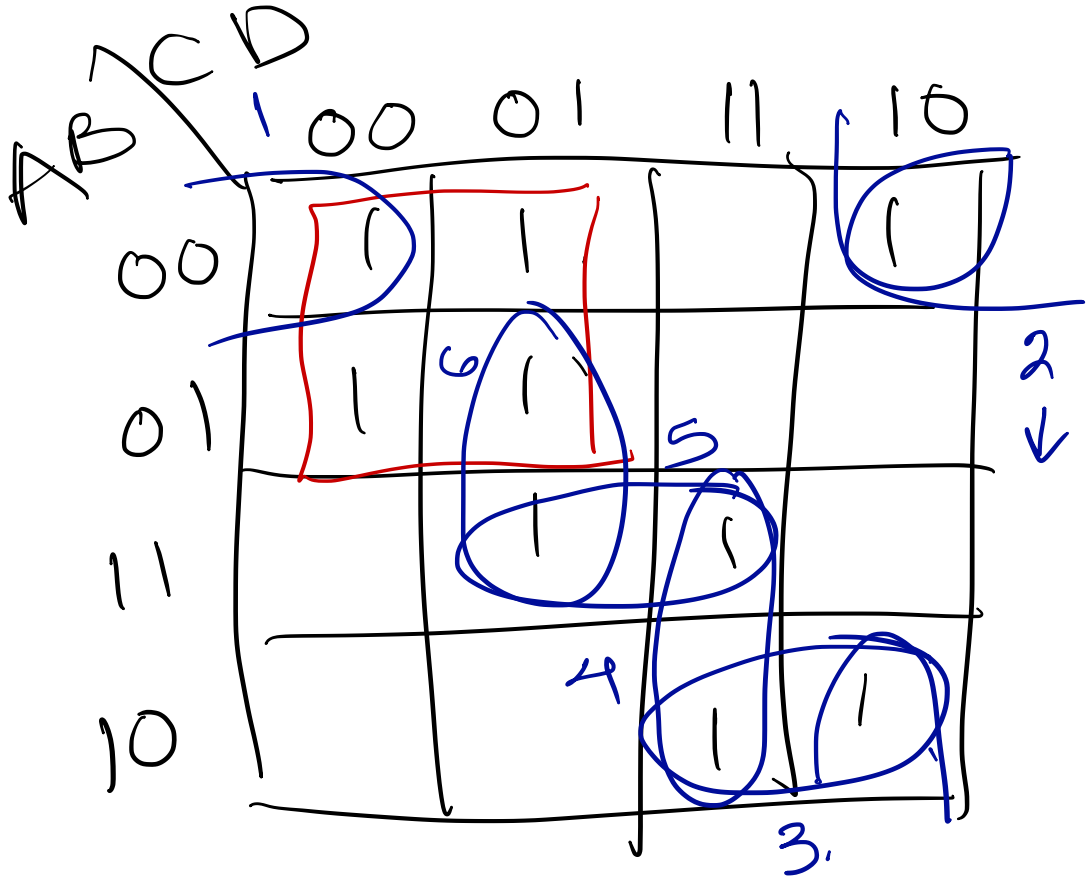
1
2
3
6

Map manipulation

- ***Selection rule***: **Minimize the overlap among prime implicants as much as possible.**

Example: F(A,B,C,D) = $\sum m(0, 1, 2, 4, 5, 10, 11, 13, 15)$

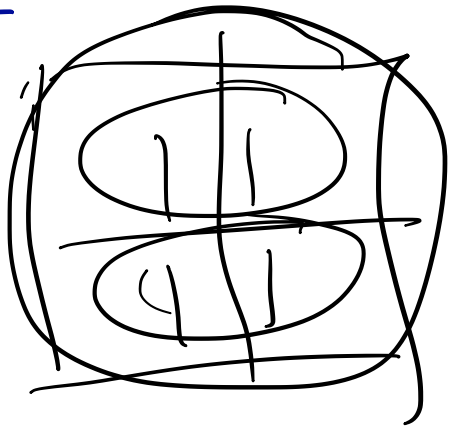




5

3

1 or 2



6

4

2

imp ✗

prime imp

ess. prime imp ✓

5

4 6

2x2 3x3

Dear students,

This year the School will be offering mentoring sessions to the following courses. These mentoring sessions commence next week (week 4) and available throughout the semester. Please make use of this opportunity to clarify your doubts face-to-face with the available academic staff during these sessions. The sessions will be available during the following times in room 552, Hilmer building (E10). These are open sessions and **DO NOT** require prior appointment. Please feel free to drop in!

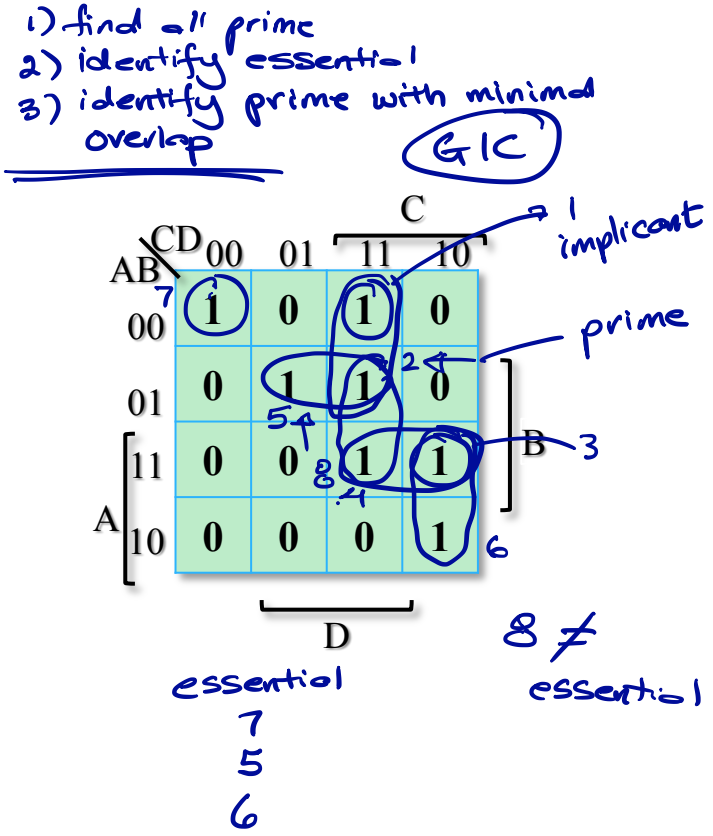
Day	Courses
Monday 4-6pm	ELEC1111, ELEC2134, ELEC3104
Tuesday 4-6pm	ELEC1111, ELEC2141, ELEC3106
Wednesday 4-6pm	ELEC1111, ELEC2134, ELEC3115

Optimization Algorithm

- **Find *all* prime implicants**
- **Include *all* essential prime implicants in the solution**
- **Include other prime implicants to cover all minterms not yet covered**
- **Try to minimize the overlap between the prime implicants**

Algorithm example

Example: F(A,B,C,D) = $\sum m(0, 3, 5, 9, 10, 14, 15)$



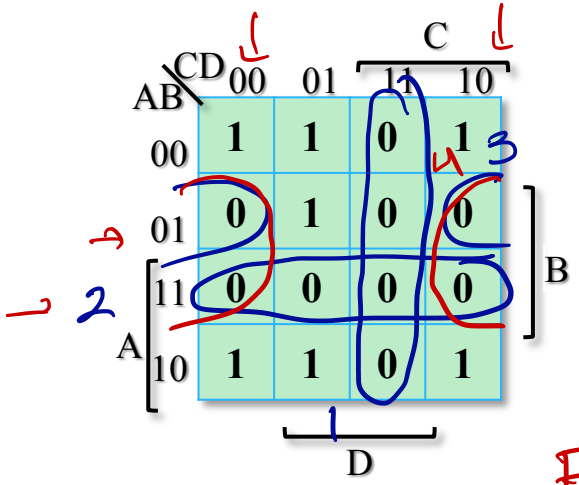
Product-of-Sums Optimization

- **Find a SOP expression for the complement of the function by grouping 0's instead of 1's**
- **Invert the function back and apply DeMorgan's Theorem**

Product-of-Sums Optimization

- **Simplify the following in a POS form**

Example: $F(A,B,C,D) = \sum m(0,1,2,5,8,9,10)$



1 – CD

2 – AB

4 – $B\bar{D}$

SOP

$\bar{F} = CD + AB + B\bar{D}$

$F = (\bar{C}+\bar{D})(\bar{A}+\bar{B})(\bar{B}+D)$

POS

Don’t care conditions

- It may happen that the output of a boolean function for a particular combinations of input variable is unspecified.

- When a particular input combination never occurs, then output is unspecified. For example, in a BCD code, input combinations from 1010 to 1111 will never occur.

- When a particular input combination is expected to occur, but we do not care about its output.

- The unspecified minterms of the function are referred as “Don’t care conditions”.

heater = 1 temp < 15

A = 1

= 0 temp > 15

cooling = 1 B = 1 temp > 20

= 0 temp < 20

~~temp~~ 15 < temp < 20

heater = 0

cooling = 0

A temp < 15	— 1	A > 15 — 0
B temp > 20	— 1	B < 20 — 0

temp < 15	~~temp~~ 15 < temp < 20
A = 1	A = 0
B = 0	B = 0

temp > 20	temp < 15 & temp > 20	
A = 0	A = 1	X Don't
B = 1	B = 1	X care

Don’t care conditions

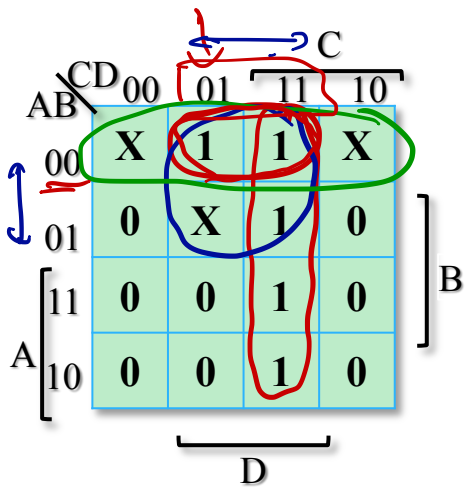
- These conditions can be used on a map to provide further simplification of the function.
- Don’t care conditions are represented by “X” in K-maps or truth tables
- In choosing adjacent squares to simplify the function in a map, the don’t care minterms may be used with either 1’s or 0’s squares.

Don’t care conditions

- These conditions can be used on a map to provide further simplification of the function.

- Example: F(A,B,C,D) = $\sum m(1, 3, 7, 11, 15)$

  d(A,B,C,D) = $\sum m(0, 2, 5)$



Don't care conditions

F = CD + $\bar{A}\bar{B}$D

F = CD + $\bar{A}$D

F = CD + $\bar{A}\bar{B}$

Multiple-Level Optimization

- ***Multiple-level circuits*** **are circuits with more than two gate levels**
- **Multiple-level circuits can have reduced gate input cost compared to two-level circuits, but tend to have longer propagation delay**
- **Multiple-level optimization is performed by applying transformations to circuits**
- **No systematic procedure exists – may not get optimum solution**

Factoring

- ***Factoring*** **is finding a factored form from either SOP or POS expressions**

- **Example: gate input count = 17**

$$G = ABC + ABD + E + ACF + ADF$$
$$= AB(C + D) + E + AF(C + D)$$
$$= (AB + AF)(C + D) + E$$
$$= A(B + F)(C + D) + E$$

GIC = 9

- **Savings of almost half the gate inputs!**

Decomposition

- ***Decomposition*** **is the expression of a function as a set of new functions**

- **Example: gate input count = 26**

$$G = A\bar{C}E + A\bar{C}F + A\bar{D}E + A\bar{D}F + BCD\bar{E}\bar{F}$$

26

- **Factor first:**

$$G = A(\bar{C}E + \bar{C}F + \bar{D}E + \bar{D}F) + BCD\bar{E}\bar{F}$$
$$= A(\bar{C} + \bar{D})(E + F) + BCD\bar{E}\bar{F}$$

Decomposition

- **After factoring:**

$$G = A(\bar{C} + \bar{D})(E + F) + BCD\bar{E}\bar{F}$$

- **Define:**

$X_1 = CD$ **GIC = 2**

$X_2 = E + F$ **GIC = 2**

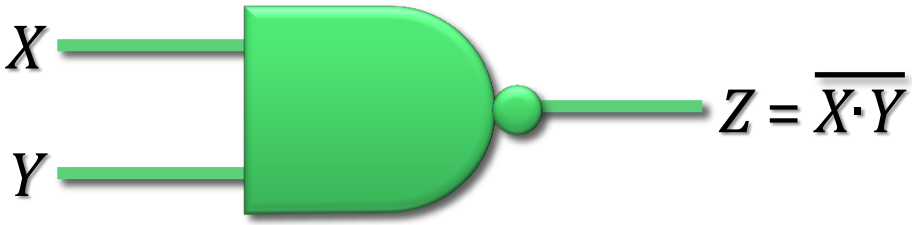
- **Rewrite *G* as:**

$$G = A\bar{X}_1X_2 + BX_1\bar{X}_2$$

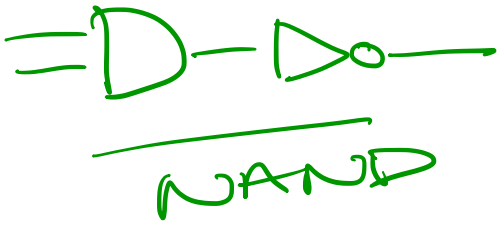
GIC = 14

- **Savings of 12 gate inputs!**

Other Gate Types – NAND Gate

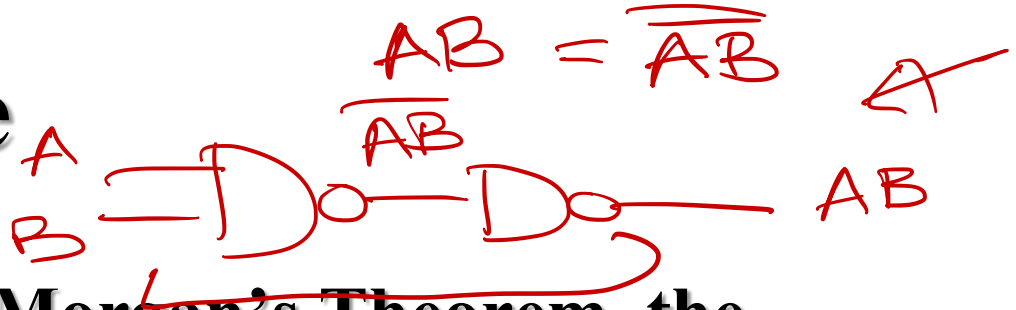


X	Y	$Z = \overline{X \cdot Y}$
0	0	1
0	1	1
1	0	1
1	1	0

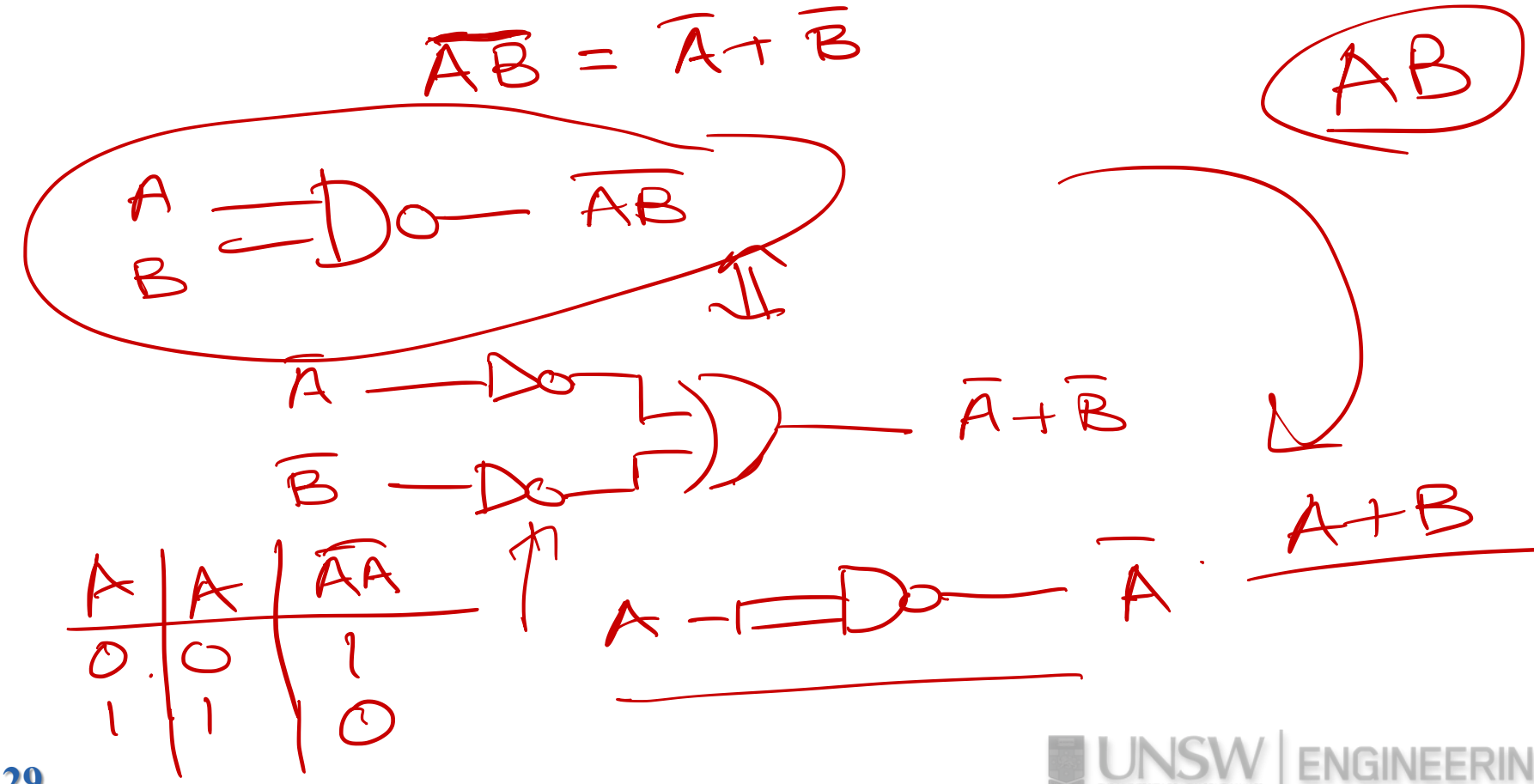


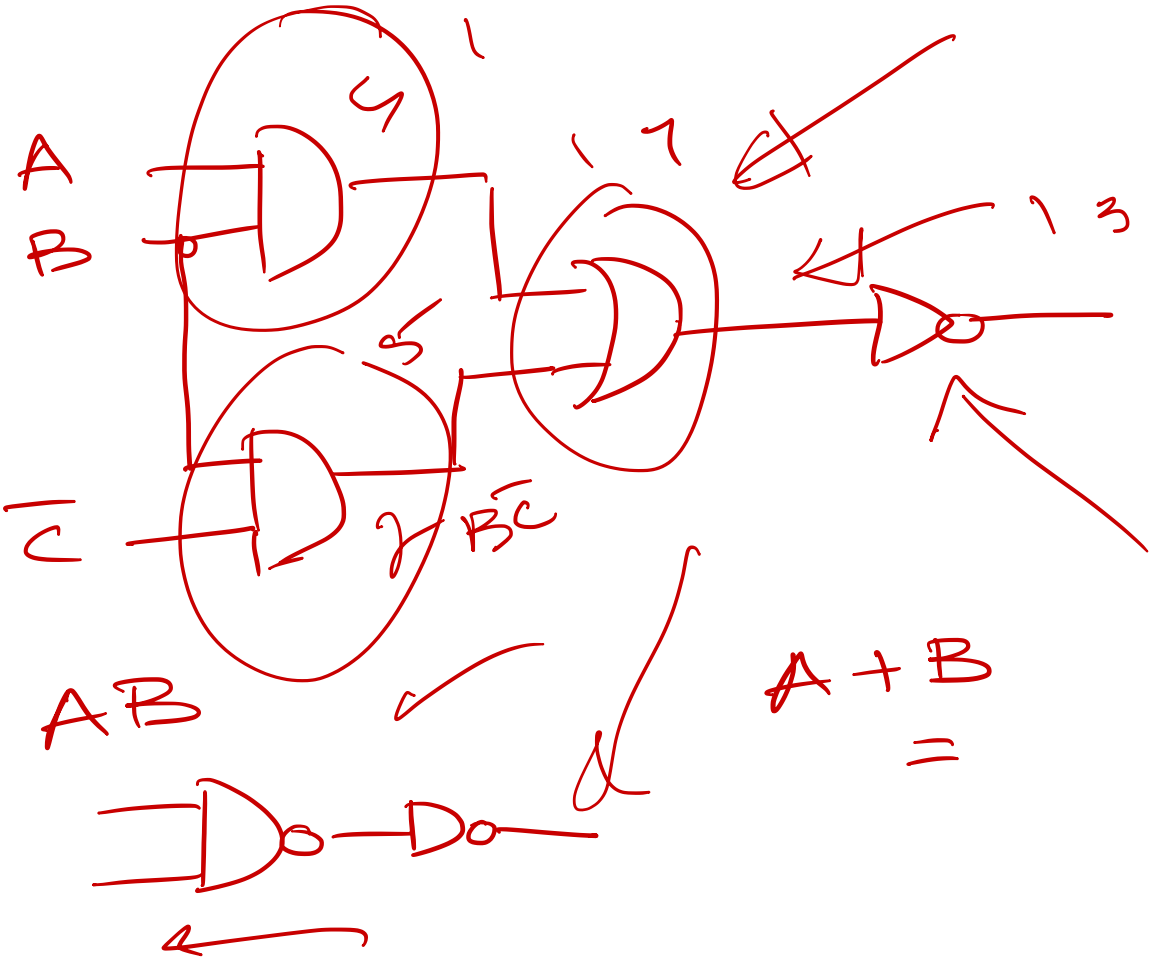
- *NAND* represents NOT-AND, i.e. the AND function with a NOT applied to the result

NAND Gate



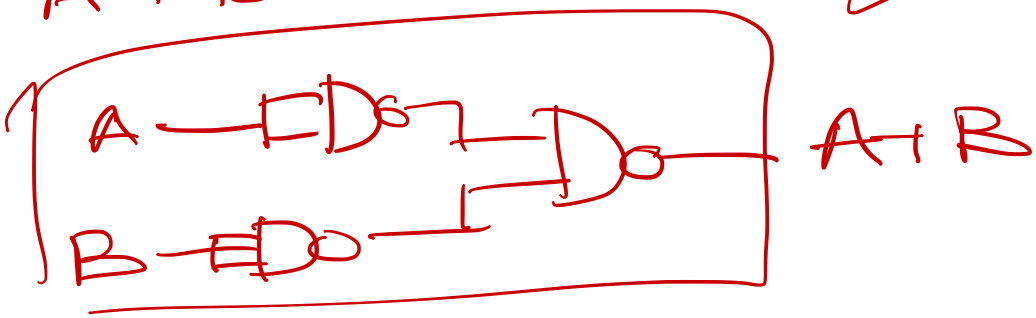
- **By applying DeMorgan's Theorem, the NAND function can also be expressed as:**

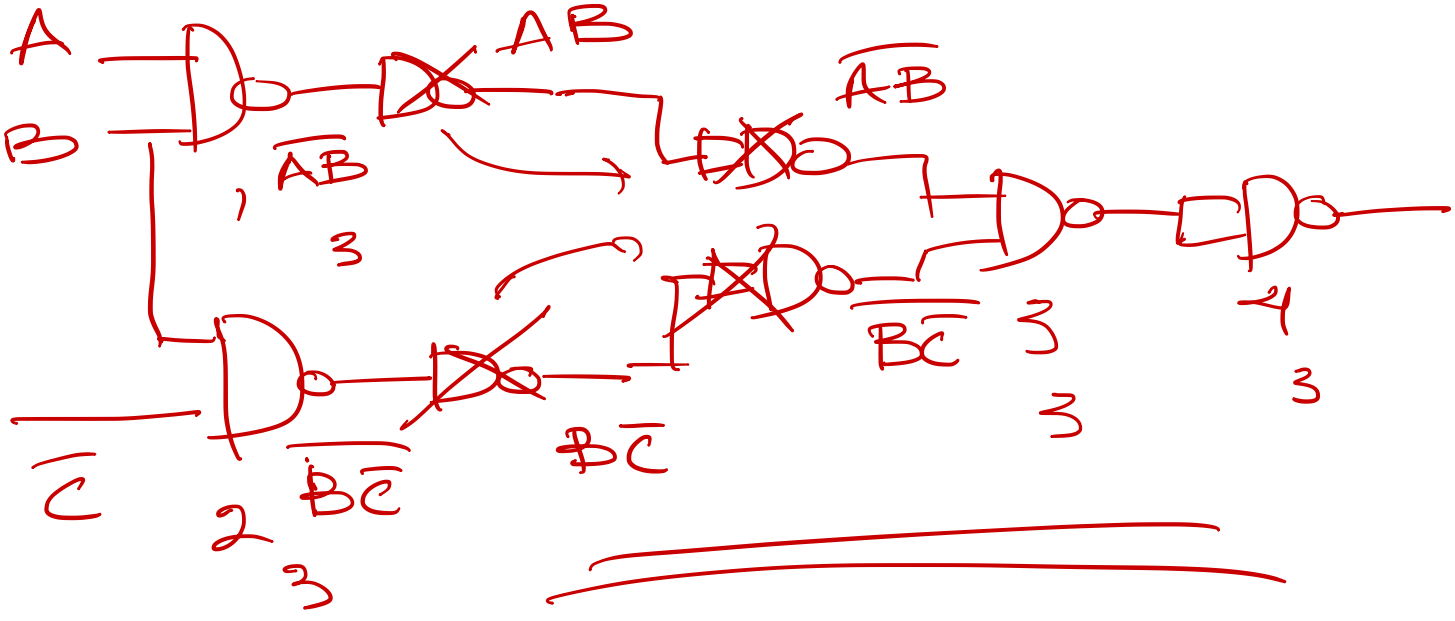




$\overline{AB} = \bar{A} + \bar{B}$

$\overline{\bar{A}\bar{B}} = \bar{\bar{A}} + \bar{\bar{B}} = A + B$



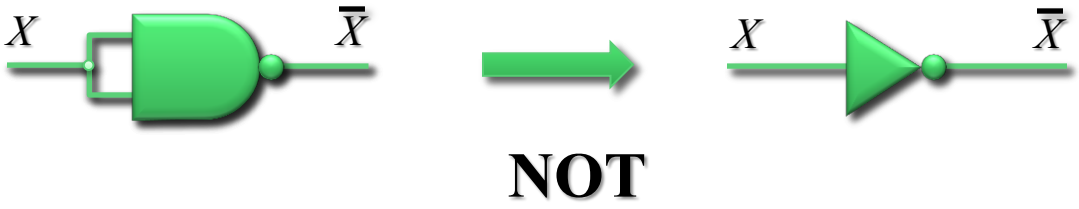


NAND Gate

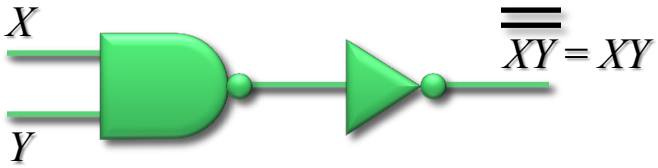
- **The NAND gate is the natural implementation for CMOS technology in terms of chip area and speed**

- **The NAND gate is a *universal gate* – a gate type that can implement any Boolean function**

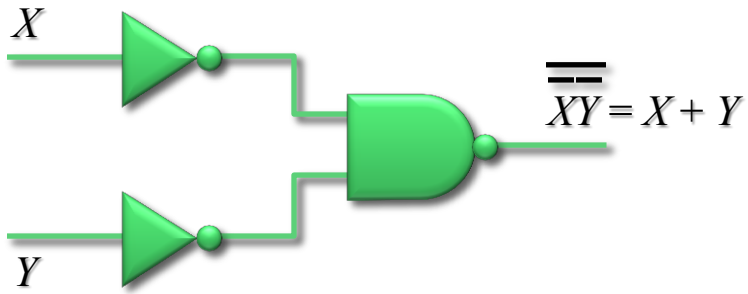
NAND as a Universal Gate



NOT

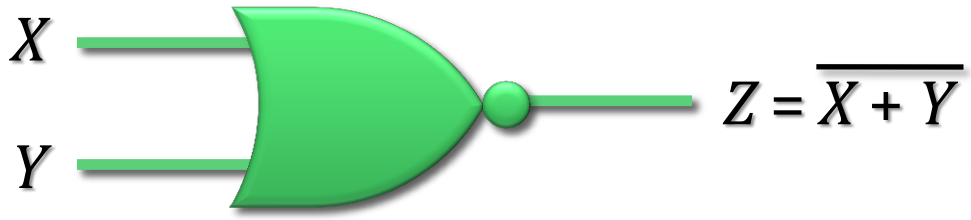


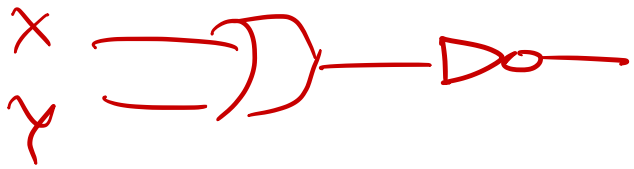
AND



OR

NOR Gate



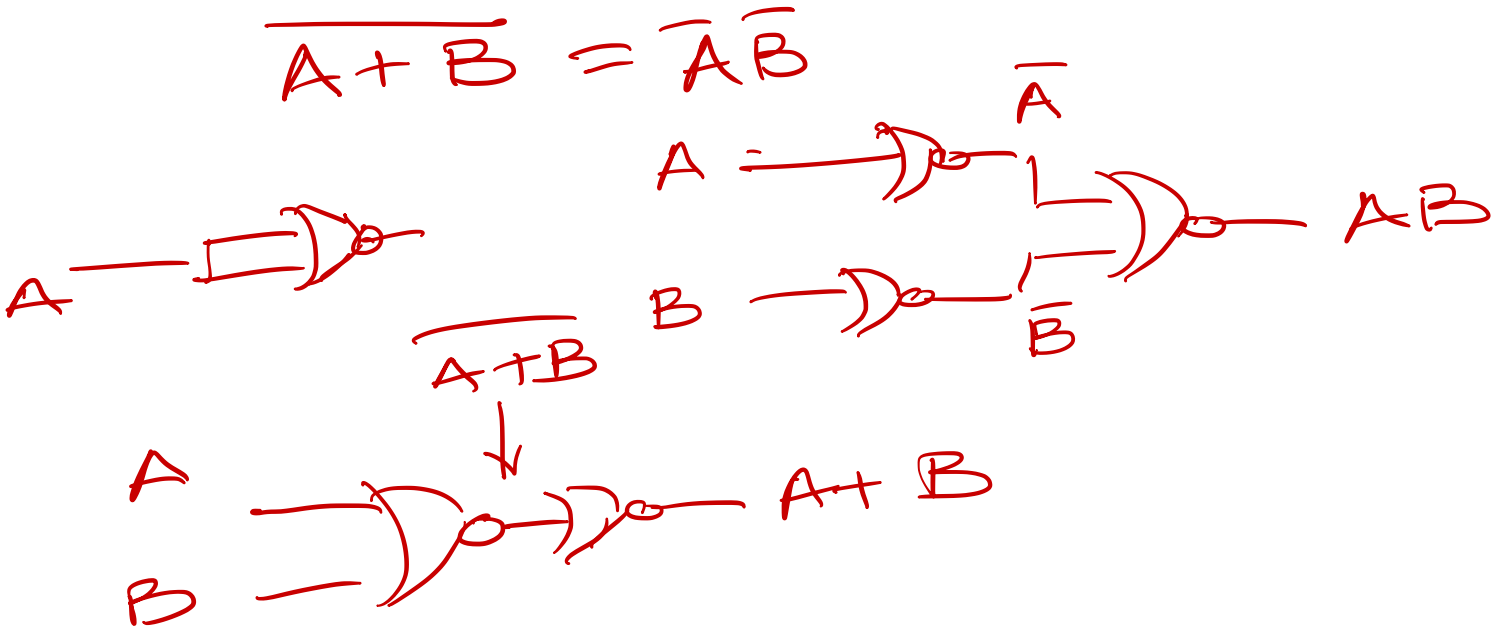


X	Y	$Z = \overline{X + Y}$
0	0	1
0	1	0
1	0	0
1	1	0

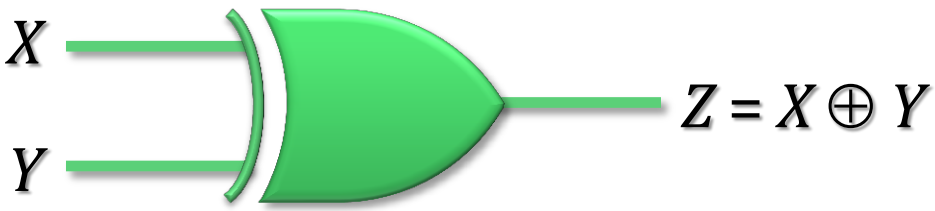
- ***NOR*** **represents NOT-OR, i.e. the OR function with a NOT applied to the result**
- **The NOR gate is another type of a universal gate**

NOR Gate

- **By applying DeMorgan's Theorem, the NOR function can also be expressed as:**



Exclusive-OR Gate



X	Y	Z = X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

- The *Exclusive-OR* (*XOR*) gate outputs 1 if one of its inputs are 1, but not both
- The XOR is denoted by ⊕ and is defined as:

$$X \oplus Y = \bar{X}Y + X\bar{Y}$$

XOR Identities

- **The following identities apply to the XOR operation:**

1. $X \oplus 0 = X$
2. $X \oplus 1 = \overline{X}$
3. $X \oplus X = 0$
4. $X \oplus \overline{X} = 1$
5. $X \oplus \overline{Y} = \overline{X \oplus Y}$
6. $\overline{X} \oplus Y = \overline{X \oplus Y}$

XOR Identities

- **The XOR operation is commutative:**

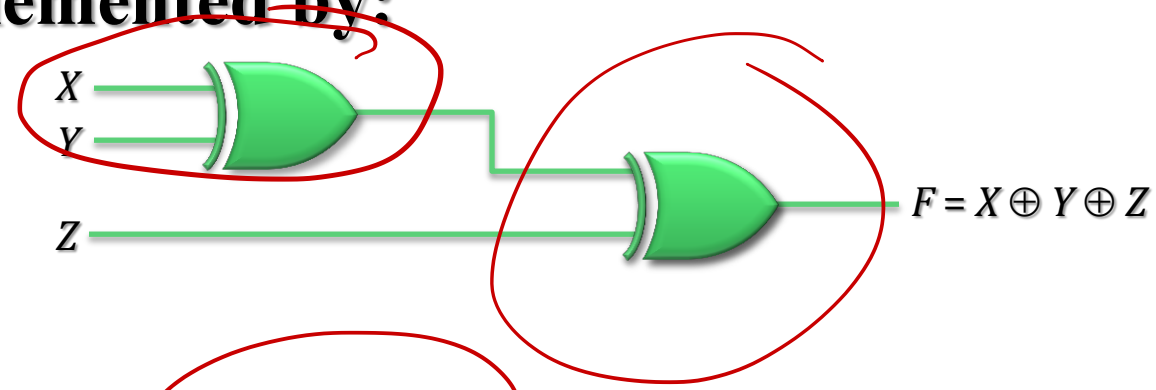
$$A \oplus B = B \oplus A$$

- **And associative:**

$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$$

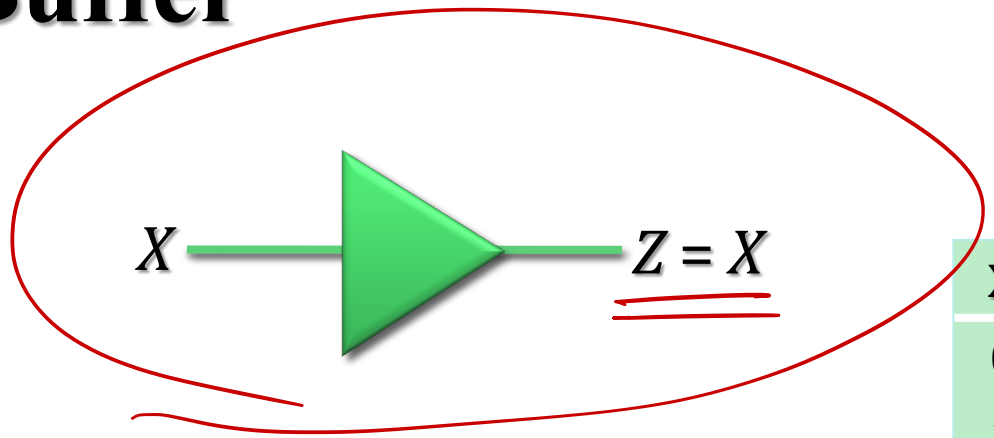
Cascading XOR Gates

- An XOR gate can only have two inputs
- The XOR operation on three or more variables is known as the *odd function*, implemented by:



- Similarly, XNOR operation on three or more variables is known as the *even function* and is implemented by adding an inverter to the output of the odd function

Buffer



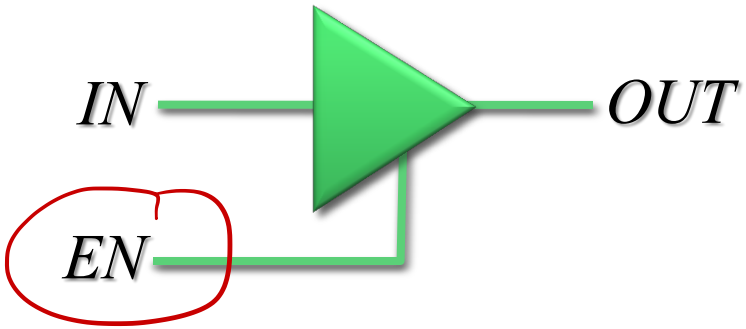
X	Z = X
0	0
1	1

- A *Buffer* is a gate with the function:

$$F = X$$

- It acts as an electrical amplifier used to improve circuit voltage levels and increase the speed of circuit operation

3-State Buffer



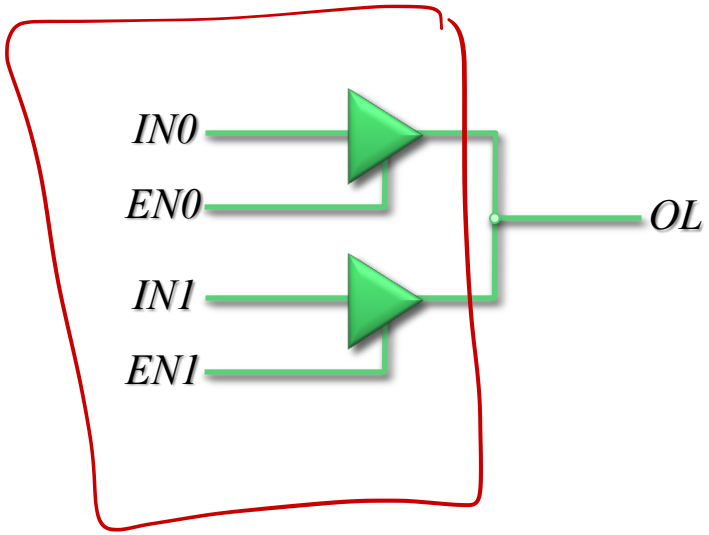
EN	IN	OUT
0	X	Z
1	0	0
1	1	1

- ***3-State Buffer*** **adds a third logic value – High-Impedance – denoted as *Hi-Z* or just *Z***

- **The input *EN* (enable) acts as the control line of the buffer**

High-Impedance Outputs

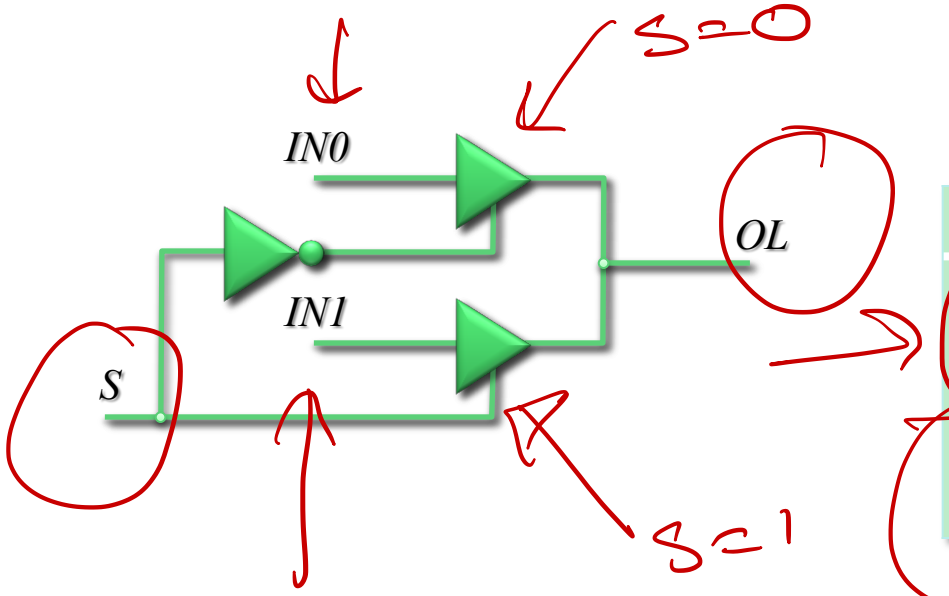
- A Hi-Z value behaves as an open-circuit, i.e. the output appears to be disconnected
- Two or more Hi-Z capable gates can hence be connected to the same output line:



EN1	EN0	IN1	IN0	OL
0	0	X	X	
0	1	X	0	
0	1	X	1	
1	0	0	X	
1	0	1	X	
1	1	0	0	
1	1	1	1	
1	1	0	1	
1	1	1	0	

3-State Buffers as a Multiplexer

- **Must ensure no electric clashes in the circuit**



S	IN1	IN0	OL
0	X	0	0
0	X	1	1
1	0	X	0
1	1	X	1

- **This circuit acts as a multiplexer with control input *S* (select)**