



# **ECE-332:437**

## **DIGITAL SYSTEMS DESIGN (DSD)**

### **Fall 2016 – Lecture 3**

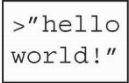


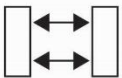
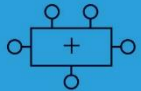

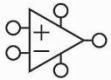

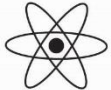
Nagi Naganathan  
September 15, 2016

# Topics to cover today – September 15, 2016

- Lecture 3
  - Logic Gates
  - Boolean Equations
  - Boolean Algebra
  - Combinational Logic

# Chapter 2 :: Topics

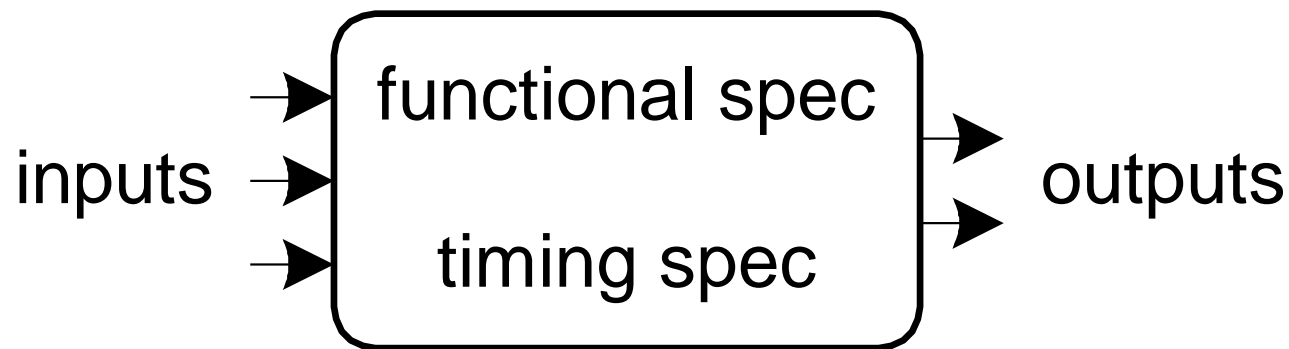
- Introduction
- Boolean Equations
- Boolean Algebra
- From Logic to Gates
- Multilevel Combinational Logic
- X's and Z's, Oh My
- Karnaugh Maps
- Combinational Building Blocks
- Timing

Application Software	
Operating Systems	
Architecture	
Micro-architecture	
Logic	
Digital Circuits	
Analog Circuits	
Devices	
Physics	

# Introduction

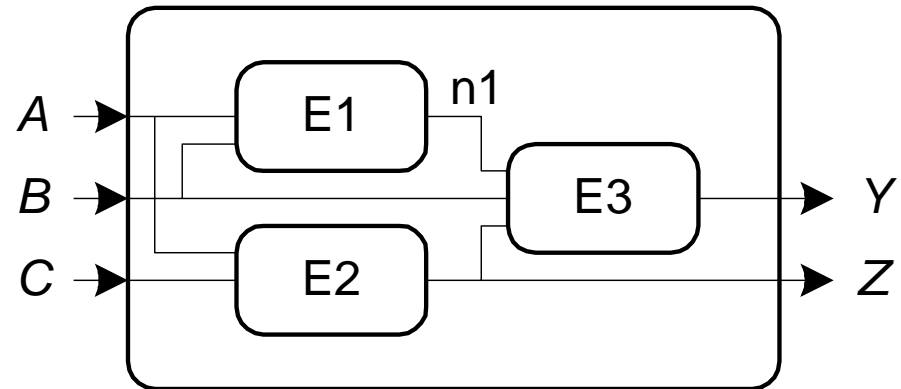
A logic circuit is composed of:

- Inputs
- Outputs
- Functional specification
- Timing specification



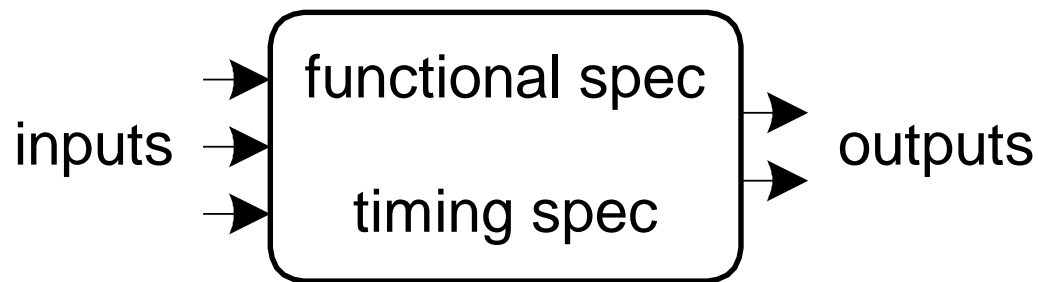
# Circuits

- Nodes
  - Inputs:  $A, B, C$
  - Outputs:  $Y, Z$
  - Internal:  $n1$
- Circuit elements
  - $E1, E2, E3$
  - Each a circuit



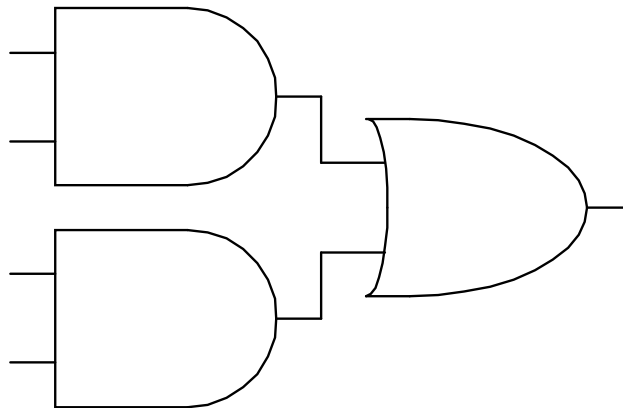
# Types of Logic Circuits

- **Combinational Logic**
  - Memoryless
  - Outputs determined by current values of inputs
- **Sequential Logic**
  - Has memory
  - Outputs determined by previous and current values of inputs



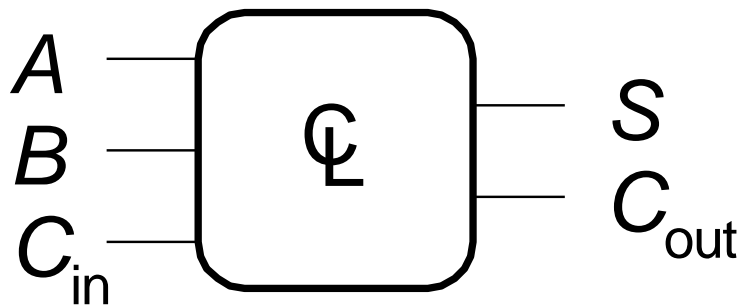
# Rules of Combinational Composition

- Every element is combinational
- Every node is either an input or connects to *exactly one* output
- The circuit contains no cyclic paths
- **Example:**



# Boolean Equations

- Functional specification of outputs in terms of inputs
- Example:**  $S = F(A, B, C_{in})$   
 $C_{out} = F(A, B, C_{in})$



$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$



# Some Definitions

- Complement: variable with a bar over it  
 $\bar{A}, \bar{B}, \bar{C}$
- Literal: variable or its complement  
 $A, \bar{A}, B, \bar{B}, C, \bar{C}$
- Implicant: product of literals  
 $ABC, \bar{A}C, BC$
- Minterm: product that includes all input variables  
 $ABC, \bar{A}\bar{B}\bar{C}, ABC$
- Maxterm: sum that includes all input variables  
 $(A+\bar{B}+C), (\bar{A}+B+\bar{C}), (\bar{A}+\bar{B}+C)$



# Sum-of-Products (SOP) Form

- All equations can be written in SOP form
- Each row has a **minterm**
- A minterm is a product (AND) of literals
- Each minterm is TRUE for that row (and only that row)
- Form function by ORing minterms where the output is TRUE
- Thus, a sum (OR) of products (AND terms)

<i>A</i>	<i>B</i>	<i>Y</i>	minterm	minterm name
0	0	0	$\overline{A} \overline{B}$	$m_0$
0	1	1	$\overline{A} B$	$m_1$
1	0	0	$A \overline{B}$	$m_2$
1	1	1	$A B$	$m_3$

$$Y = F(A, B) =$$

# Sum-of-Products (SOP) Form

- All equations can be written in SOP form
- Each row has a **minterm**
- A minterm is a product (AND) of literals
- Each minterm is TRUE for that row (and only that row)
- Form function by ORing minterms where the output is TRUE
- Thus, a sum (OR) of products (AND terms)

<i>A</i>	<i>B</i>	<i>Y</i>	minterm	minterm name
0	0	0	$\overline{A} \overline{B}$	$m_0$
0	1	1	$\overline{A} B$	$m_1$
1	0	0	$A \overline{B}$	$m_2$
1	1	1	$A B$	$m_3$

$$Y = F(A, B) =$$

# Sum-of-Products (SOP) Form

- All equations can be written in SOP form
- Each row has a **minterm**
- A minterm is a product (AND) of literals
- Each minterm is TRUE for that row (and only that row)
- Form function by ORing minterms where the output is TRUE
- Thus, a sum (OR) of products (AND terms)

A	B	Y	minterm	minterm name
0	0	0	$\overline{A} \overline{B}$	$m_0$
0	1	1	$\overline{A} B$	$m_1$
1	0	0	$A \overline{B}$	$m_2$
1	1	1	$A B$	$m_3$

$$Y = F(A, B) = \overline{A}B + AB = \Sigma(1, 3)$$

# Product-of-Sums (POS) Form

- All Boolean equations can be written in POS form
- Each row has a **maxterm**
- A maxterm is a sum (OR) of literals
- Each maxterm is FALSE for that row (and only that row)
- Form function by ANDing the maxterms for which the output is FALSE
- Thus, a product (AND) of sums (OR terms)

<b>A</b>	<b>B</b>	<b>Y</b>	<b>maxterm</b>	<b>maxterm name</b>
0	0	0	$A + B$	$M_0$
0	1	1	$A + \overline{B}$	$M_1$
1	0	0	$\overline{A} + B$	$M_2$
1	1	1	$\overline{A} + \overline{B}$	$M_3$

$$Y = F(A, B) = (A + B)(A + \overline{B}) = \Pi(0, 2)$$

# Boolean Equations Example

- You are going to the cafeteria for lunch
  - You won't eat lunch ( $\bar{E}$ )
  - If it's not open ( $\bar{O}$ ) or
  - If they only serve corndogs ( $C$ )
- Write a truth table for determining if you will eat lunch ( $E$ ).

$O$	$C$	$E$
0	0	
0	1	
1	0	
1	1	

# Boolean Equations Example

- You are going to the cafeteria for lunch
  - You won't eat lunch ( $\bar{E}$ )
  - If it's not open ( $\bar{O}$ ) or
  - If they only serve corndogs ( $C$ )
- Write a truth table for determining if you will eat lunch ( $E$ ).

$O$	$C$	$E$
0	0	0
0	1	0
1	0	1
1	1	0

# SOP & POS Form

- SOP – sum-of-products

$O$	$C$	$E$	minterm
0	0		$\overline{O} \overline{C}$
0	1		$\overline{O} C$
1	0		$O \overline{C}$
1	1		$O C$

- POS – product-of-sums

$O$	$C$	$E$	maxterm
0	0		$O + C$
0	1		$O + \overline{C}$
1	0		$\overline{O} + C$
1	1		$\overline{O} + \overline{C}$



# SOP & POS Form

- SOP – sum-of-products

$O$	$C$	$E$	minterm
0	0	0	$\overline{O} \overline{C}$
0	1	0	$\overline{O} C$
1	0	1	$O \overline{C}$
1	1	0	$O C$

$$E = O\overline{C}$$

$$= \Sigma(2)$$

- POS – product-of-sums

$O$	$C$	$E$	maxterm
0	0	0	$O + C$
0	1	0	$O + \overline{C}$
1	0	1	$\overline{O} + C$
1	1	0	$\overline{O} + \overline{C}$

$$E = (O + C)(O + \overline{C})(\overline{O} + \overline{C})$$

$$= \Pi(0, 1, 3)$$

# Boolean Algebra

- Axioms and theorems to **simplify** Boolean equations
- Like regular algebra, but simpler: variables have only two values (1 or 0)
- **Duality** in axioms and theorems:
  - ANDs and ORs, 0's and 1's interchanged

# Boolean Axioms

Axiom		Dual		Name
A1	$B = 0 \text{ if } B \neq 1$	A1'	$B = 1 \text{ if } B \neq 0$	Binary field
A2	$\overline{0} = 1$	A2'	$\overline{1} = 0$	NOT
A3	$0 \bullet 0 = 0$	A3'	$1 + 1 = 1$	AND/OR
A4	$1 \bullet 1 = 1$	A4'	$0 + 0 = 0$	AND/OR
A5	$0 \bullet 1 = 1 \bullet 0 = 0$	A5'	$1 + 0 = 0 + 1 = 1$	AND/OR

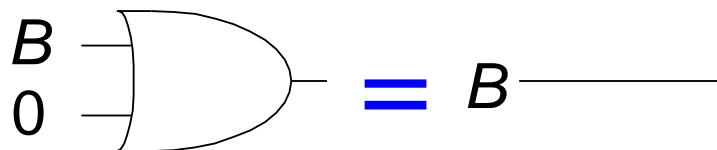
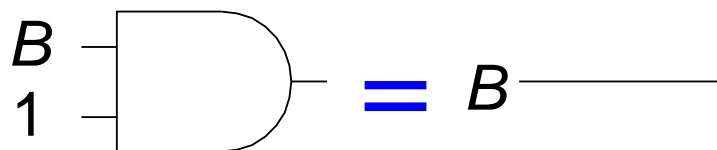
Theorem		Dual		Name
T1	$B \bullet 1 = B$	T1'	$B + 0 = B$	Identity
T2	$B \bullet 0 = 0$	T2'	$B + 1 = 1$	Null Element
T3	$B \bullet B = B$	T3'	$B + B = B$	Idempotency
T4		$\overline{\overline{B}} = B$		Involution
T5	$B \bullet \overline{B} = 0$	T5'	$B + \overline{B} = 1$	Complements

# T1: Identity Theorem

- $B \cdot 1 = B$
- $B + 0 = B$

# T1: Identity Theorem

- $B \cdot 1 = B$
- $B + 0 = B$



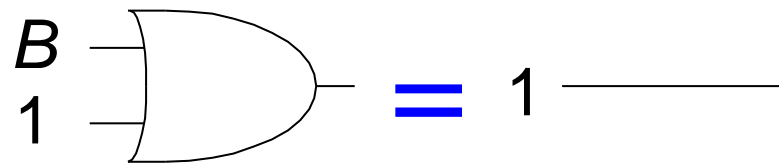
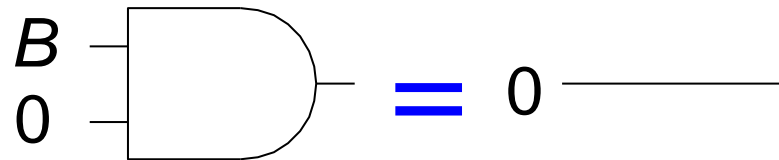
# T2: Null Element Theorem

- $B \cdot 0 = 0$
- $B + 1 = 1$



# T2: Null Element Theorem

- $B \cdot 0 = 0$
- $B + 1 = 1$



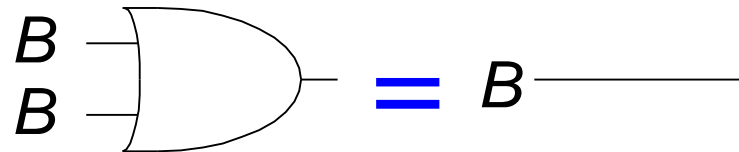
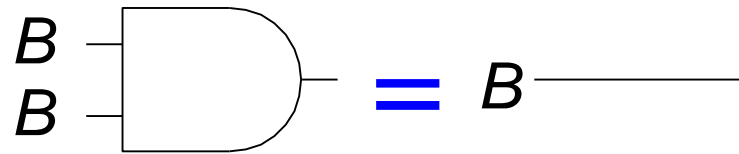
# T3: Idempotency Theorem

- $B \cdot B = B$
- $B + B = B$



# T3: Idempotency Theorem

- $B \cdot B = B$
- $B + B = B$

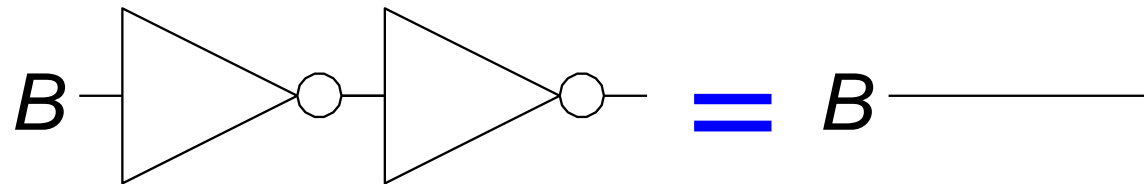


# T4: Identity Theorem

- $\overline{\overline{B}} = B$

# T4: Identity Theorem

- $\overline{\overline{B}} = B$

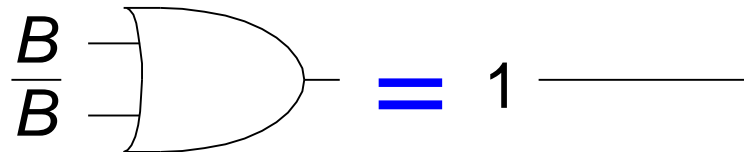
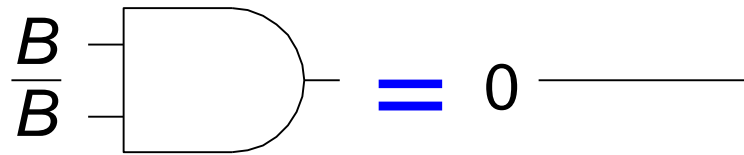


# T5: Complement Theorem

- $B \cdot \bar{B} = 0$
- $B + \bar{B} = 1$

# T5: Complement Theorem

- $B \cdot \bar{B} = 0$
- $B + \bar{B} = 1$



# Boolean Theorems Summary

	Theorem		Dual	Name
T1	$B \bullet 1 = B$	T1'	$B + 0 = B$	Identity
T2	$B \bullet 0 = 0$	T2'	$B + 1 = 1$	Null Element
T3	$B \bullet B = B$	T3'	$B + B = B$	Idempotency
T4		$\overline{\overline{B}} = B$		Involution
T5	$B \bullet \overline{B} = 0$	T5'	$B + \overline{B} = 1$	Complements

# Boolean Theorems of Several Vars

Theorem		Dual		Name
T6	$B \bullet C = C \bullet B$	T6'	$B + C = C + B$	Commutativity
T7	$(B \bullet C) \bullet D = B \bullet (C \bullet D)$	T7'	$(B + C) + D = B + (C + D)$	Associativity
T8	$(B \bullet C) + (B \bullet D) = B \bullet (C + D)$	T8'	$(B + C) \bullet (B + D) = B + (C \bullet D)$	Distributivity
T9	$B \bullet (B + C) = B$	T9'	$B + (B \bullet C) = B$	Covering
T10	$(B \bullet C) + (B \bullet \overline{C}) = B$	T10'	$(B + C) \bullet (B + \overline{C}) = B$	Combining
T11	$(B \bullet C) + (\overline{B} \bullet D) + (C \bullet D)$ $= B \bullet C + \overline{B} \bullet D$	T11'	$(B + C) \bullet (\overline{B} + D) \bullet (C + D)$ $= (B + C) \bullet (\overline{B} + D)$	Consensus
T12	$\overline{B_0 \bullet B_1 \bullet B_2 \dots}$ $= (\overline{B_0} + \overline{B_1} + \overline{B_2} \dots)$	T12'	$\overline{B_0 + B_1 + B_2 \dots}$ $= (\overline{B_0} \bullet \overline{B_1} \bullet \overline{B_2})$	De Morgan's Theorem

**Note:** T8' differs from traditional algebra: OR (+) distributes over AND ( $\bullet$ )



# Simplifying Boolean Equations

## Example 1:

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



# Simplifying Boolean Equations

## Example 1:

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

$$= B(A + \bar{A}) \quad \text{T8}$$

$$= B(1) \quad \text{T5'}$$

$$= B \quad \text{T1}$$

# Simplifying Boolean Equations

## Example 2:

$$Y = A(AB + ABC)$$

# Simplifying Boolean Equations

## Example 2:

$$Y = A(AB + ABC)$$

$$= A(AB(1 + C)) \quad \text{T8}$$

$$= A(AB(1)) \quad \text{T2'}$$

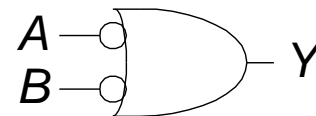
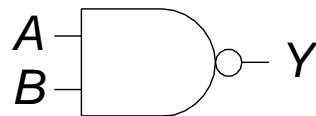
$$= A(AB) \quad \text{T1}$$

$$= (AA)B \quad \text{T7}$$

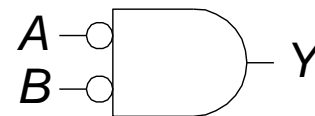
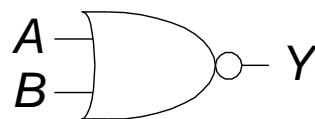
$$= AB \quad \text{T3}$$

# DeMorgan's Theorem

- $Y = \overline{AB} = \overline{A} + \overline{B}$



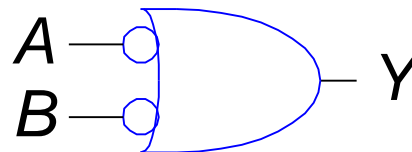
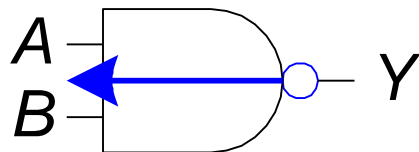
- $Y = \overline{A + B} = \overline{A} \cdot \overline{B}$



# Bubble Pushing

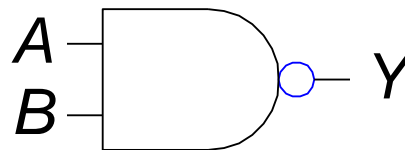
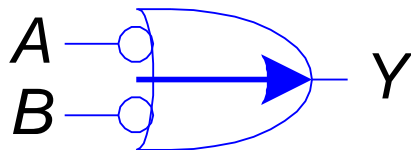
- **Backward:**

- Body changes
- Adds bubbles to inputs



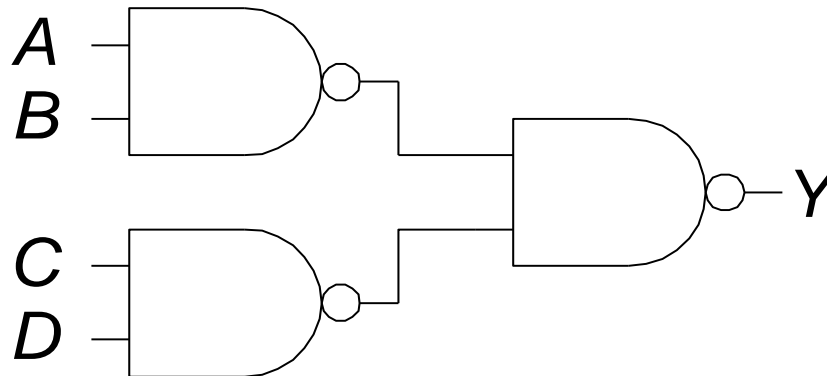
- **Forward:**

- Body changes
- Adds bubble to output



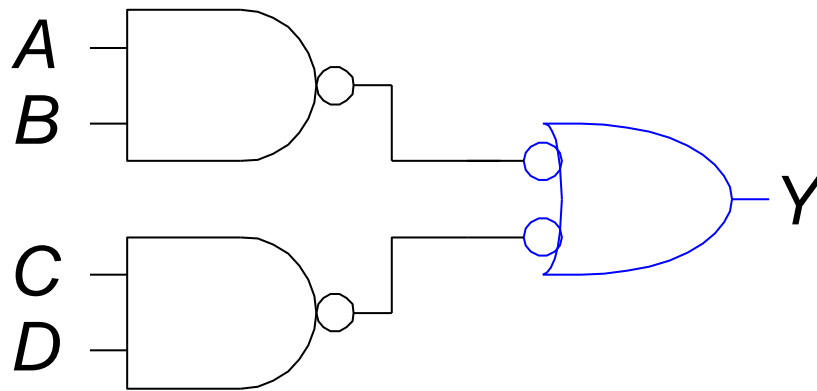
# Bubble Pushing

- What is the Boolean expression for this circuit?



# Bubble Pushing

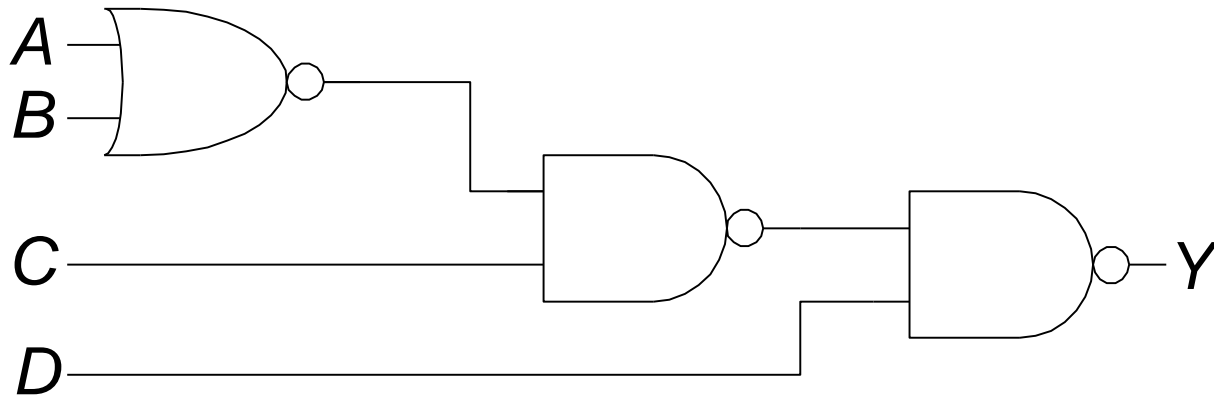
- What is the Boolean expression for this circuit?



$$Y = AB + CD$$

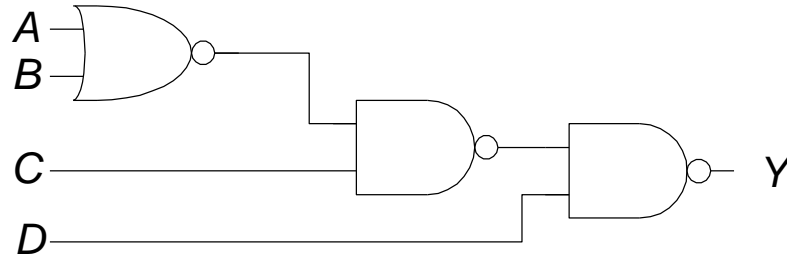
# Bubble Pushing Rules

- Begin at output, then work toward inputs
- Push bubbles on final output back
- Draw gates in a form so bubbles cancel

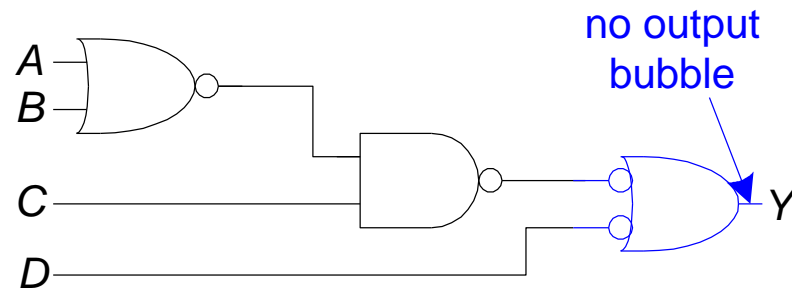




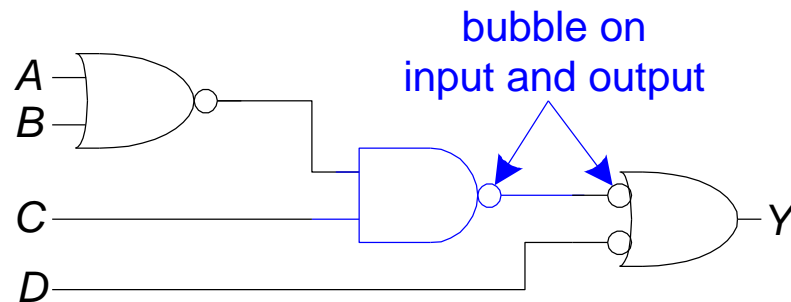
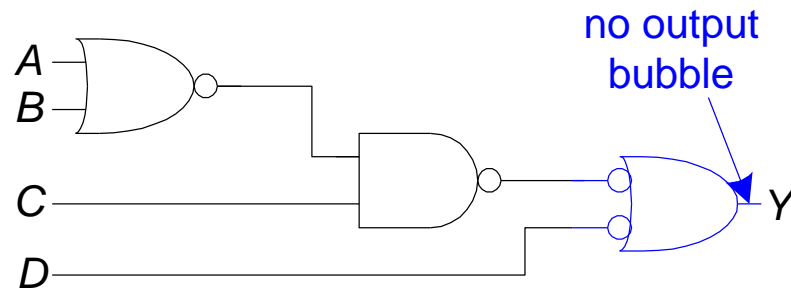
# Bubble Pushing Example



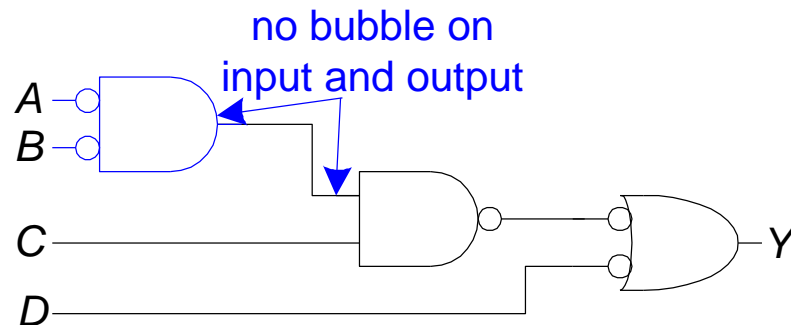
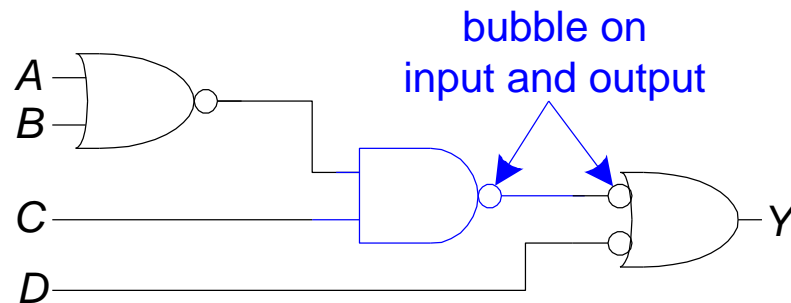
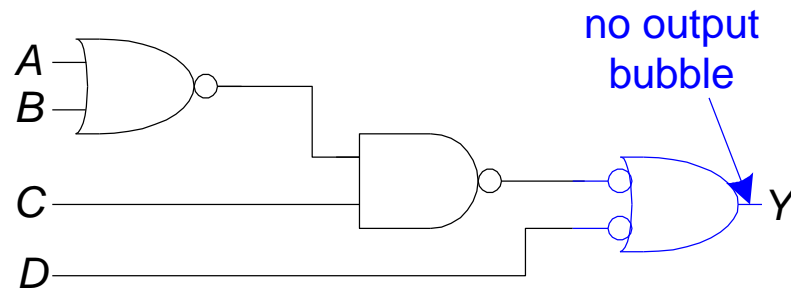
# Bubble Pushing Example



# Bubble Pushing Example



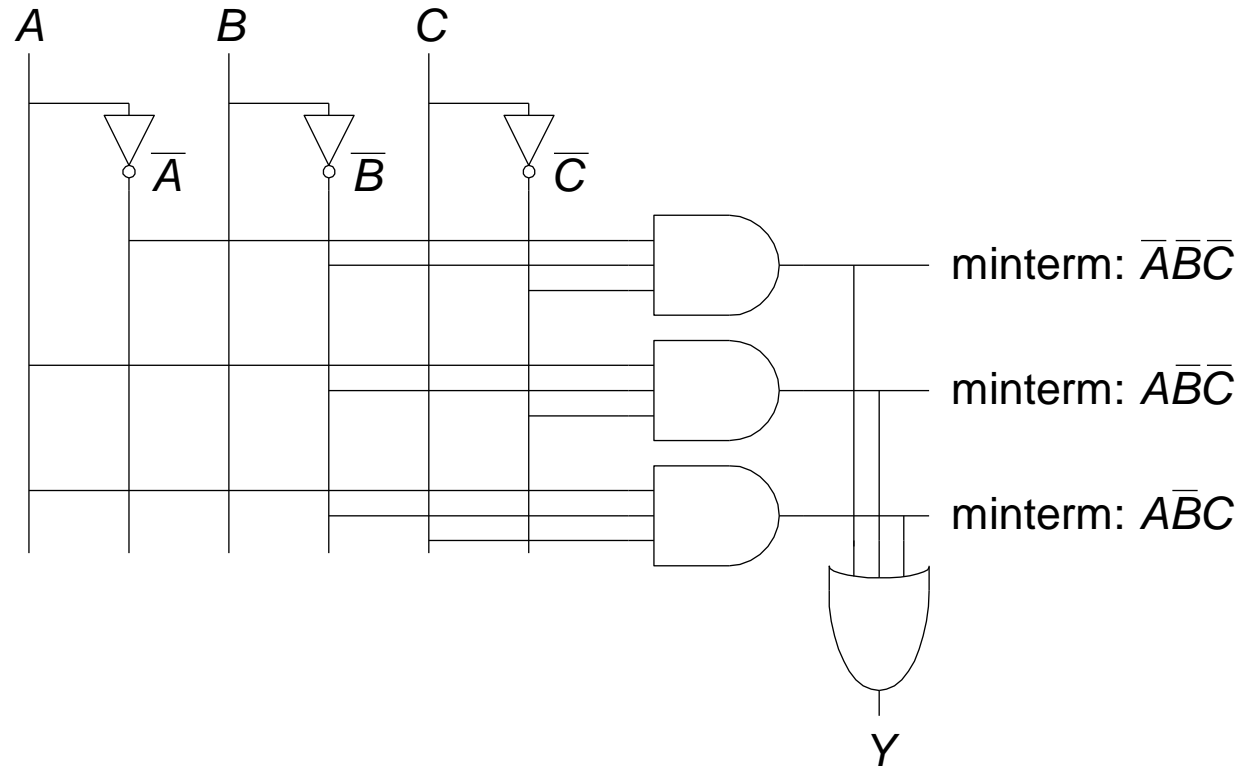
# Bubble Pushing Example



$$Y = \overline{A}\overline{B}C + \overline{D}$$

# From Logic to Gates

- Two-level logic: ANDs followed by ORs
- Example:  $Y = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}C$



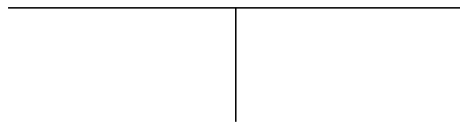
# Circuit Schematics Rules

- Inputs on the left (or top)
- Outputs on right (or bottom)
- Gates flow from left to right
- Straight wires are best

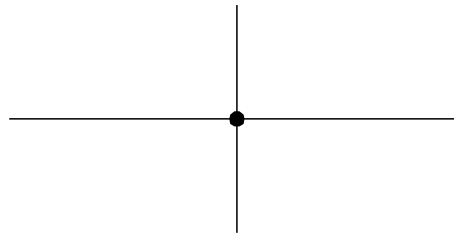
# Circuit Schematic Rules (cont.)

- Wires always connect at a T junction
- A dot where wires cross indicates a connection between the wires
- Wires crossing *without* a dot make no connection

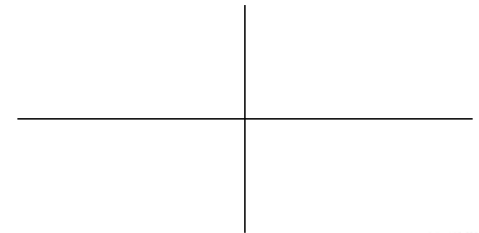
wires connect  
at a T junction



wires connect  
at a dot



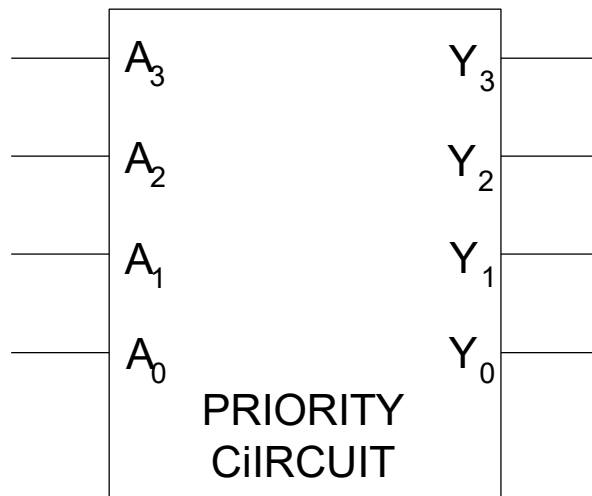
wires crossing  
without a dot do  
not connect



# Multiple-Output Circuits

- Example: Priority Circuit**

Output asserted  
corresponding to  
most significant  
TRUE input



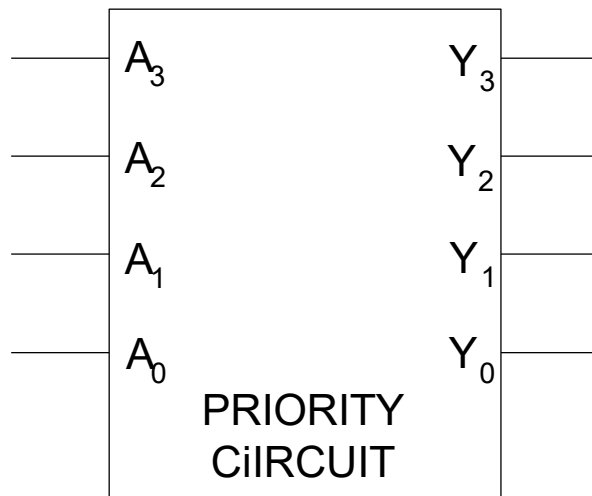
$A_3$	$A_2$	$A_1$	$A_0$	$Y_3$	$Y_2$	$Y_1$	$Y_0$
0	0	0	0				
0	0	0	1				1
0	0	1	0			1	
0	0	1	1			1	
0	1	0	0		1		
0	1	0	1		1		
0	1	1	0		1		
0	1	1	1		1		
1	0	0	0	1			
1	0	0	1	1			
1	0	1	0	1			
1	0	1	1	1			
1	1	0	0	1			
1	1	0	1	1			
1	1	1	0	1			
1	1	1	0	1			
1	1	1	1	1			
1	1	1	1	1			



# Multiple-Output Circuits

- Example: Priority Circuit**

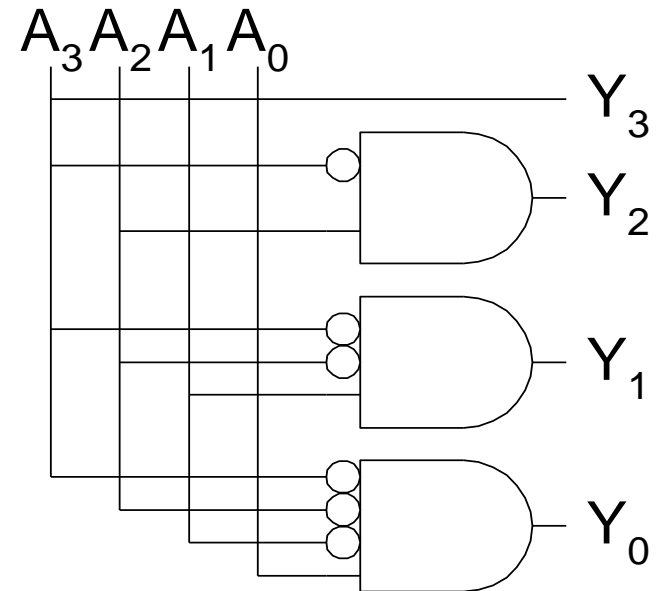
Output asserted  
corresponding to  
most significant  
TRUE input



$A_3$	$A_2$	$A_1$	$A_0$	$Y_3$	$Y_2$	$Y_1$	$Y_0$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	0	0	0

# Priority Circuit Hardware

$A_3$	$A_2$	$A_1$	$A_0$	$Y_3$	$Y_2$	$Y_1$	$Y_0$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0



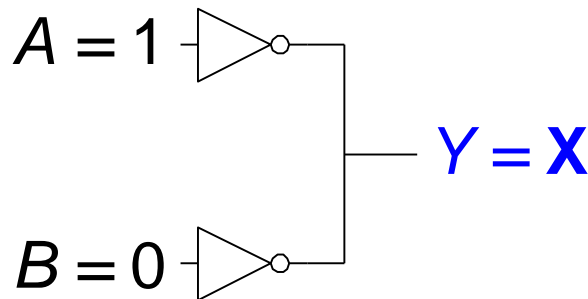
# Don't Cares

$A_3$	$A_2$	$A_1$	$A_0$	$Y_3$	$Y_2$	$Y_1$	$Y_0$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

$A_3$	$A_2$	$A_1$	$A_0$	$Y_3$	$Y_2$	$Y_1$	$Y_0$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	X	0	0	1	0
0	1	X	X	0	1	0	0
1	X	X	X	1	0	0	0

# Contention: X

- Contention: circuit tries to drive output to 1 **and** 0
  - Actual value somewhere in between
  - Could be 0, 1, or in forbidden zone
  - Might change with voltage, temperature, time, noise
  - Often causes excessive power dissipation

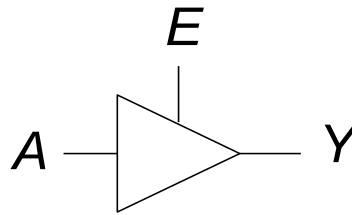


- **Warnings:**
  - Contention usually indicates a **bug**.
  - **X** is used for “don’t care” and contention - look at the context to tell them apart

# Floating: Z

- Floating, high impedance, open, high Z
- Floating output might be 0, 1, or somewhere in between
  - A voltmeter won't indicate whether a node is floating

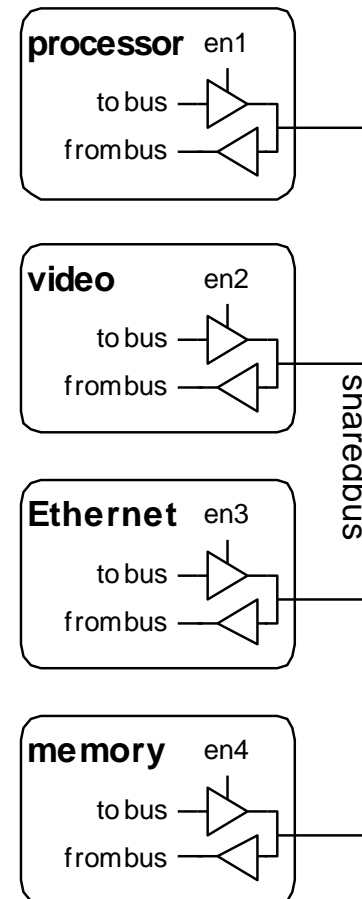
## Tristate Buffer



$E$	$A$	$Y$
0	0	Z
0	1	Z
1	0	0
1	1	1

# Tristate Busses

- Floating nodes are used in tristate busses
  - Many different drivers
  - Exactly one is active at once



# Part 2

# Karnaugh Maps

# Boolean Function Minimization

- Simpler expression leads to minimum number of gates
- Cheaper, Faster, Smaller
- Algebraic methods
  - Using theorems
  - Harder
- Karnaugh Map (K-map)
  - Pictorial/Graphical representation and similar to Venn Diagram
  - Easier due to pattern matching
  - Limited to 5 variables
  - Quine-Mccluskey for large combinational circuits



# What are Karnaugh<sup>1</sup> maps?

---

- Karnaugh maps provide an alternative way of simplifying logic circuits
- Instead of using Boolean algebra simplification techniques, you can transfer logic values from a Boolean statement or a truth table into a Karnaugh map
- The arrangement of 0's and 1's within the map helps you to visualize the logic relationships between the variables and leads directly to a simplified Boolean statement

<sup>1</sup>Named for the American electrical engineer Maurice Karnaugh

# Karnaugh Maps (K-map)

---

- **A K-map is a collection of squares**
  - Each square represents a minterm
  - The collection of squares is a graphical representation of a Boolean function
  - Adjacent squares differ in the value of one variable
  - Alternative algebraic expressions for the same function are derived by recognizing patterns of squares
- **The K-map can be viewed as**
  - A reorganized version of the truth table
  - A topologically-warped Venn diagram as used to visualize sets in algebra of sets

# Some Uses of K-Maps

---

- **Provide a means for:**
  - **Finding optimum or near optimum**
    - **SOP and POS standard forms, and**
    - **two-level AND/OR and OR/AND circuit implementations**
- for functions with small numbers of variables**
- **Visualizing concepts related to manipulating Boolean expressions, and**
- **Demonstrating concepts used by computer-aided design programs to simplify large circuits**

# K-map Definitions

- Complement: variable with a bar over it

$\bar{A}, \bar{B}, \bar{C}$

- Literal: variable or its complement

$A, \bar{A}, B, \bar{B}, C, \bar{C}$

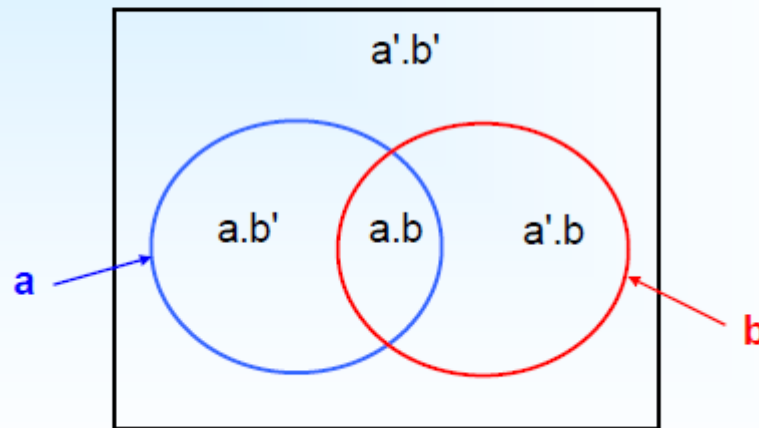
- Implicant: product of literals

$ABC, \bar{A}C, BC$

- **Prime implicant:** implicant corresponding to the largest circle in a K-map

# Venn Diagrams

- Venn diagram to represent the space of minterms.
- Example of 2 variables (4 minterms):



# Venn Diagrams

- Each set of minterms represents a Boolean function. Examples:

$$\{a.b, a.b'\} \rightarrow a.b + a.b' = a.(b+b') = a$$

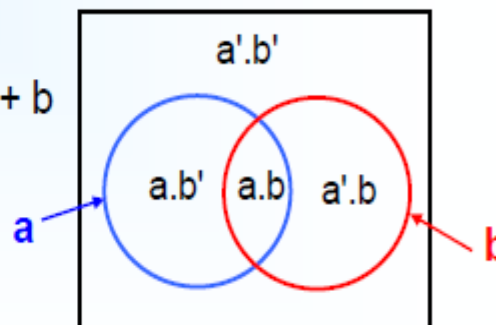
$$\{a'.b, a.b\} \rightarrow a'.b + a.b = (a'+a).b = b$$

$$\{a.b\} \rightarrow a.b$$

$$\{a.b, a.b', a'.b\} \rightarrow a.b + a.b' + a'.b = a + b$$

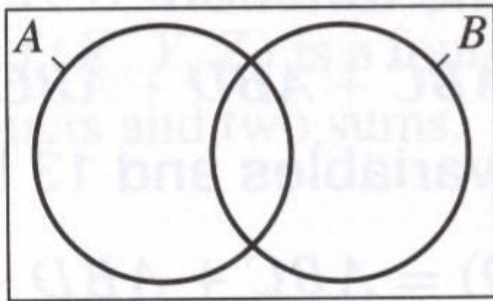
$$\{\} \rightarrow 0$$

$$\{a'.b', a.b, a.b', a'.b\} \rightarrow 1$$

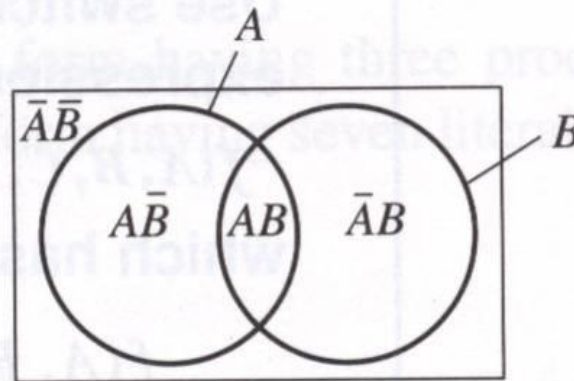


# Karnaugh maps

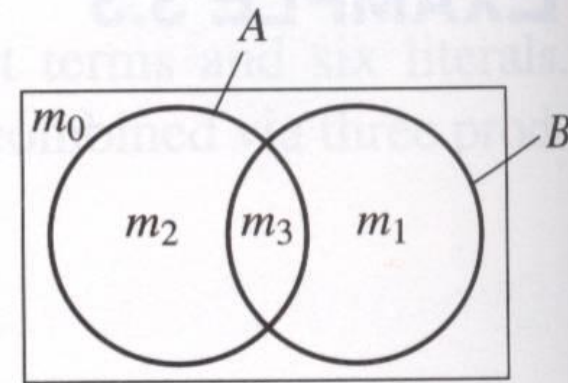
- Karnaugh map (also K-map) is a graphic tool, pictorial representation of truth table
  - Extension of the concepts of truth table, Venn diagram, minterm



(a)



(b)



(c)

- Transition from Venn diagram to minterm

# K-Map – Example

## Two-Variable K-Maps:

The 2-variable map is a table of 2 rows by 2 columns. The 2 rows represent the two values of the first input variable A, while the two columns represent the two values of the second input variable B.

Thus, all entries (squares) in the first row correspond to input variable  $A=0$ , while entries (squares) of the second row correspond to  $A=1$ .

Likewise, all entries of the first column correspond to input variable  $B = 0$ , while entries of the second column correspond to  $B=1$ .

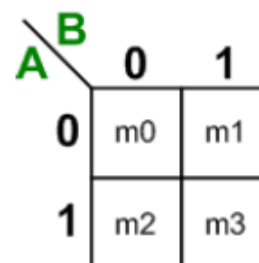
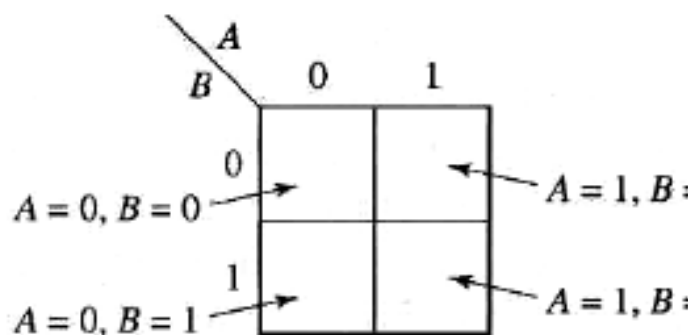
Thus, each map entry (or square) corresponds to a unique value for the input variables A and B.



# K-Map – Example

The Karnaugh map of a function specifies the value of the function for every combination of values of the independent variables

A two-variable Karnaugh map



## 2-variable K-map

### Code Distance:

Let's first define the concept of Code Distance. The distance between two binary code-words is the number of bit positions in which the two code-words have different values.

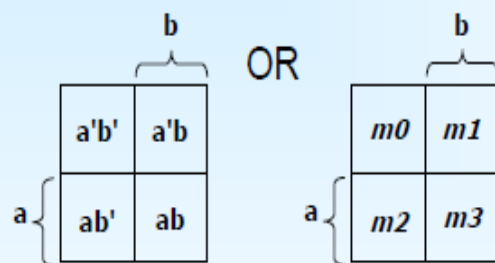
For example, the distance between the code words **1001** and **0001** is **1** while the distance between the code-words **0011** and **0100** is **3**.

This definition of code distance is commonly known as the *Hamming distance* between two codes.

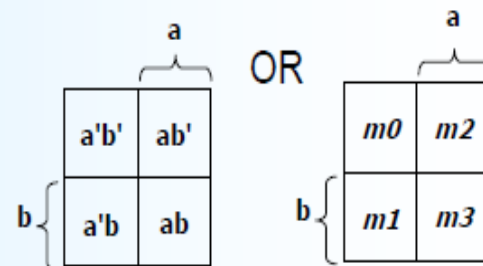
## 2-variable K-map

- Alternative layouts of a 2-variable ( $a, b$ ) K-map

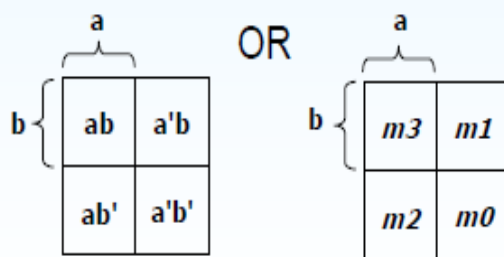
Alternative 1:



Alternative 2:



Alternative 3:



and others...

# K-Map

The cells are arranged as above, but we write them empty, like this:

**2-input:**

a \ b	0	1
0		
1		

**3-input:**

a \ bc	00	01	11	10
0				
1				

**4-input:**

ab \ cd	00	01	11	10
00				
01				
11				
10				

Note that the numbers are *not* in binary order, but are arranged so that only a single bit changes between neighbours.

# K-Map

- Easier way of deriving Boolean expressions

## Example

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

A Karnaugh map sets out the minterms pictorially.

Example for 3 variables

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

Like a truth table

- each 1 represents the presence of that minterm in the CSOP form

Not like a truth table

- set out differently (each column/row differs in 1 variable only from its neighbours).

A \ BC	00	01	11	10
0	1	2	4	3
1	5	6	8	7

- numbers correspond to rows in a truth table

# K-map Rules

- Every 1 in a K-map must be circled at least once
- Each circle must span a power of 2 (i.e. 1, 2, 4) squares in each direction
- Each circle must be as large as possible
- A circle may wrap around the edges of the K-map
- A “don't care” (X) is circled only if it helps minimize the equation

# 2-variable K-map

Here is a two-input truth table for a digital circuit:

Row	Inputs		Output
	A	B	F
Row # 0	0	0	0
Row # 1	0	1	1
Row # 2	1	0	1
Row # 3	1	1	1

The corresponding K-map is:

		B	
		0	1
A	0	Row # 0: 0	Row # 1: 1
	1	Row # 2: 1	Row # 3: 1

# 2-variable K-map

Two K-map squares are considered *adjacent* if the input codes they represent have a *Hamming distance of 1*.

A K-map square with a function value of 1 will be referred to as a *1-Square*.

A K-map square with a function value of 0 will be referred to as a *0-Square*.

**Step 1:** Draw the map according to the number of input variables of the function.

**Step 2:** Fill “1’s” in the squares for which the function is true.

**Step 3:** Form as big group of *adjacent 1-squares* as possible. There are some rules for this which you will learn with bigger maps.

**Step 4:** Find the common literals for each group and write the simplified expression in SOP.

## Example:

Consider the given truth table of two variable function. Obtain the simplified function using K-map.

A	B	F
0	0	0
0	1	0
1	0	1
1	1	1



## 2-variable K-map

First draw a 2-variable K-map. The function  $F$  is true when  $AB'$  ( $m_2$ ) is true and when  $AB$  ( $m_3$ ) is true, so a 1 is placed inside the square that belongs to  $m_2$  and a 1 is placed inside the square that belongs to  $m_3$ .

		<b>B</b>	
		0	1
<b>A</b>	0		
	1	1	1

Since both of the 1-squares have different values for variable  $B$  but the same value for variable  $A$ , which is 1, i.e., wherever  $A = 1$  then  $F = 1$  thus  $F = A$ .

This simplification is justified by algebraic manipulation as follows:

$$F = m_2 + m_3 = AB' + AB = A(B' + B) = A$$

To understand how combining squares simplifies Boolean functions, the basic property possessed by the **adjacent squares** must be recognized.

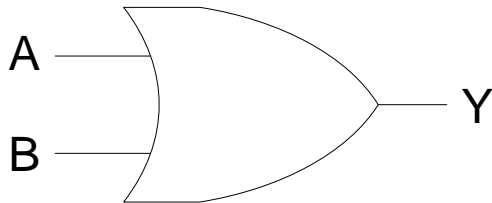
In the above example, the two 1-squares are adjacent with the same value for variable  $A$  ( $A=1$ ) but different values for variable  $B$  (one square has  $B=0$ , while the other has  $B=1$ ).

This reduction is possible since both squares are adjacent and the net expression is that of the common variable ( $A$ ).

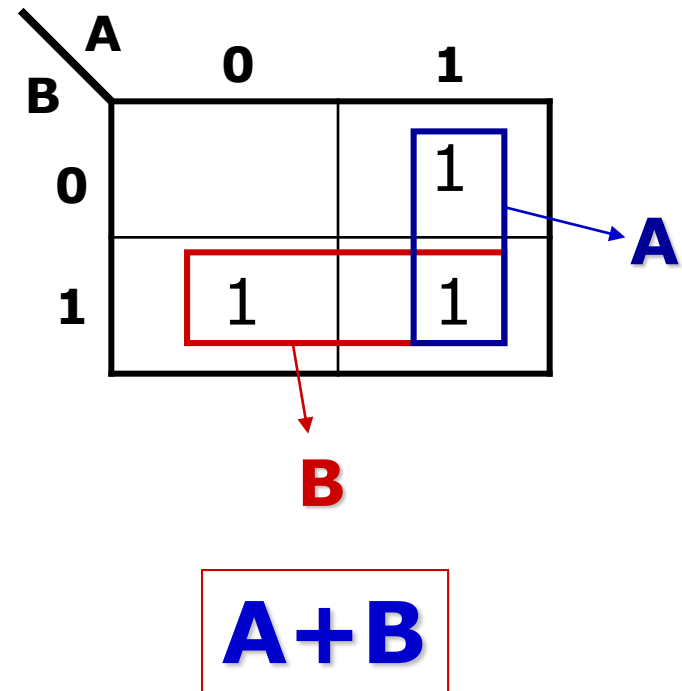
# Example

---

2-variable Karnaugh maps are trivial but can be used to introduce the methods you need to learn. The map for a 2-input OR gate looks like this:



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1




# K-Map – 2 variables

In this example:

$$F = m_1 + m_2 + m_3 = m_1 + m_2 + (m_3 + m_3)$$

$$F = (m_1 + m_3) + (m_2 + m_3) = A + B$$

 **Rule:** A 1-square can be member of more than one group.

If we exchange the places of **A** and **B**, then minterm positions will also change. Thus, **m<sub>1</sub>** and **m<sub>2</sub>** will be exchanged as well.

		B	
		0	1
A	0	m0	m1
	1	m2	m3

		A	
		0	1
B	0	m0	m2
	1	m1	m3

In an *n*-variable map each square is *adjacent* to “*n*” other squares, e.g., in a 2-variable map each square is adjacent to two other squares as shown below:

		B	
		0	1
A	0	✓	✓
	1		

		B	
		0	1
A	0		
	1	✓	✓

		B	
		0	1
A	0	✓	
	1	✓	

		B	
		0	1
A	0		✓
	1		✓

Examples of non-adjacent squares are shown below:

		B	
		0	1
A	0	✗	
	1		✗

		B	
		0	1
A	0		✗
	1	✗	

# K-Map – 2 variables

## 2-input Example

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

	B	0	1
A \	0	1	1
	1	1	

$$Y = B' + A'$$

(This is NAND)

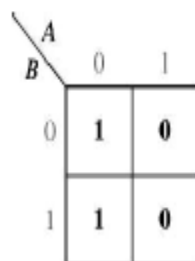
Direct from truth table:  $Y = A'B' + A'B + AB'$

# K-Map – Example

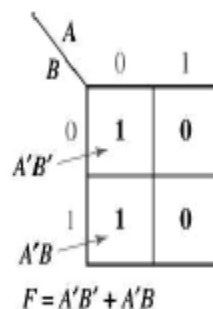
- Each entry of a Karnaugh map is a minterm
- We can read the minterms from the map just like we can read them from the truth table

$A B$	$F$
0 0	1
0 1	1
1 0	0
1 1	0

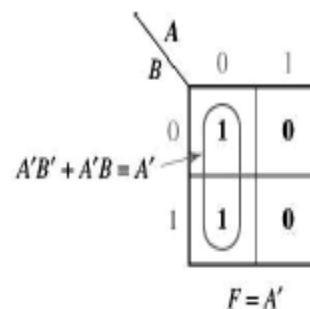
(a)



(b)



(c)



(d)

# Karnaugh Maps (K-Maps)

- Boolean expressions can be minimized by combining terms
- K-maps minimize equations graphically
- $PA + P\bar{A} = P$

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Y C	AB			
	00	01	11	10
0	1	0	0	0
1	1	0	0	0

Y C	AB			
	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}B\bar{C}$	$AB\bar{C}$	$A\bar{B}\bar{C}$
1	$\bar{A}\bar{B}C$	$\bar{A}BC$	$ABC$	$A\bar{B}C$

# 3-Input K-Map

Y C \ AB		00	01	11	10
C	0	$ABC$	$\bar{A}BC$	$AB\bar{C}$	$A\bar{B}\bar{C}$
	1	$\bar{A}\bar{B}C$	$\bar{A}B\bar{C}$	$ABC$	$A\bar{B}C$

Truth Table

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

K-Map

Y C \ AB		00	01	11	10
C	0				
	1				

# 3-Input K-Map

Y C \ AB		AB			
		00	01	11	10
C	0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}B\bar{C}$	$A\bar{B}\bar{C}$	$ABC$
	1	$\bar{A}\bar{B}C$	$\bar{A}BC$	$ABC$	$A\bar{B}C$

Truth Table

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

K-Map

Y C \ AB		AB			
		00	01	11	10
C	0	0	1	1	0
	1	0	1	0	0

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



Karnaugh maps consist of a set of  $2^2$  squares where 2 is the number of variables in the Boolean expression being minimized.

A	B	F	Minterm	Maxterm
0	0	0	$A'B'$	$A + B$
0	1	1	$A'B$	$A + B'$
1	0	1	$AB'$	$A' + B$
1	1	1	$AB$	$A' + B'$

**Truth Table**

A \ B	0	1
0	0	1
1	1	1

**2 Variable K-Map**

# Karnaugh Maps..... (Contd.)

- For three and four variable expressions  
Maps with  $2^3 = 8$  and  $2^4 = 16$  cells are used.  
Each cell represents a MINTERM or a MAXTERM

	00	01	11	10
0				
1				

$2^3 = 8$  Cells

	00	01	11	10
00				
01				
11				
10				

4 Variable K-Map  $2^4 = 16$  Cells

# 3 variable K-Map

## Three-Variable K-Maps:

There are eight minterms for a Boolean function with three-variables. Hence, a three-variable map consists of 8 squares.

		BC			
		00	01	11	10
A	0	m0	m1	m3	m2
	1	m4	m5	m7	m6

All entries (squares) in the first row correspond to input variable  $A=0$ , while entries (squares) of the second row correspond to  $A=1$ .

Likewise, all entries of the first column correspond to input variable  $B = 0, C = 0$ , all entries of the second column correspond to input variable  $B = 0, C = 1$ , all entries of the third column correspond to input variable  $B = 1, C = 1$ , while entries of the fourth column correspond to  $B=1, C = 0$ .

# 3 variable K-Map

## Variations of Three-Variable Map:

The figure shows variations of the three-variable map. Note that the minterm corresponding to each square can be obtained by substituting the values of variables ABC in order.

		AB			
		00	01	11	10
C	0	m0	m2	m6	m4
	1	m1	m3	m7	m5

		AC			
		00	01	11	10
B	0	m0	m1	m5	m4
	1	m2	m3	m7	m6

AB	C	
	0	1
00	m0	m1
01	m2	m3
11	m6	m7
10	m4	m5

BC	A	
	0	1
00	m0	m4
01	m1	m5
11	m3	m7
10	m2	m6

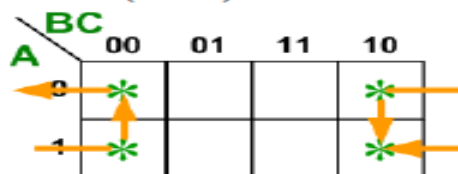
# 3 variable K-Map


There are cases where two squares in the map are considered to be adjacent even though they do not physically touch each other.


In the figure of 3-variable map,  $m_0$  is adjacent to  $m_2$  and  $m_4$  is adjacent to  $m_6$  because the minterms differ by only one variable. This can be verified algebraically:

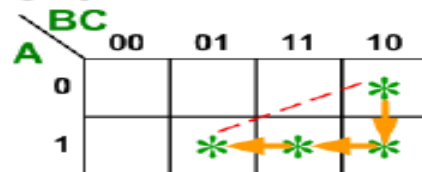
$$m_0 + m_2 = A'B'C' + A'BC' = A'C' (B' + B) = A'C'$$


$$m_4 + m_6 = AB'C' + ABC' = AC' (B' + B) = AC'$$



 **Rule:** Groups may only consist of 2, 4, 8, 16,... squares (always power of 2). For example, groups may not consist of 3, 6 or 12 squares.

 **Rule:** Members of a group must have a closed loop adjacency, i.e., L-Shaped 4 squares do not form a valid group.



 **Notes:**

1. Each square is adjacent to 3 other squares.
2. One square is represented by a minterm (i.e. a product term containing all 3 literals).
3. A group of 2 adjacent squares is represented by a product term containing only 2 literals, i.e., 1 literal is dropped.
4. A group of 4 adjacent squares is represented by a product term containing only 1 literal, i.e., 2 literals are dropped.

# 3 variable K-Map

- There are 8 minterms for 3 variables (a, b, c).  
Therefore, there are 8 cells in a 3-variable K-map.

		b			
		bc			
		00	01	11	10
a	0	$a'b'c'$	$a'b'c$	$a'bc$	$a'bc'$
a	1	$ab'c'$	$ab'c$	$abc$	$abc'$

OR

		b			
		bc			
		00	01	11	10
a	0	$m_0$	$m_1$	$m_3$	$m_2$
a	1	$m_4$	$m_5$	$m_7$	$m_6$

Above arrangement ensures that minterms of adjacent cells *differ by only ONE literal*.  
(Other arrangements which satisfy this criterion may also be used.)

Note Gray code sequence

# 3 variable K-Map

- There is **wrap-around** in the K-map:
  - $a'.b'.c'$  ( $m0$ ) is adjacent to  $a'.b.c'$  ( $m2$ )
  - $a.b'.c'$  ( $m4$ ) is adjacent to  $a.b.c'$  ( $m6$ )



	bc	00	01	11	10
a	0	$m0$	$m1$	$m3$	$m2$
	1	$m4$	$m5$	$m7$	$m6$

Each cell in a 3-variable K-map has 3 adjacent neighbours. In general, each cell in an  $n$ -variable K-map has  $n$  adjacent neighbours. For example,  $m0$  has 3 adjacent neighbours:  $m1$ ,  $m2$  and  $m4$ .

# Minimization Steps (SOP Expression)

Form groups of adjacent 1's. Make groups as large as possible.

Group size must be a power of two.  
i.e. Group of

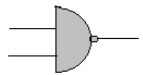
- 8 (OCTET)
- 4 (QUAD),
- 2 (PAIR) or
- 1 (Single)

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



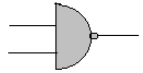
# Minimization Steps (SOP Expression with 4 var.)

*The process has following steps:*



Draw the K-Map for given function as shown

Enter the function values into the K-Map by placing 1's and 0's into the appropriate Cells



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

# Karnaugh Maps - Rules of Simplification (SOP Expression)

- Groups may not include any cell containing a zero

A \ B	0	1
0	0	
1	1	

WRONG ✗

A \ B	0	1
0	0	
1	1	1

RIGHT ✓

# Karnaugh Maps - Rules of Simplification (SOP Expression)

- Groups may be horizontal or vertical, but not diagonal .

		A	
		0	1
B	0	0	1
	1	1	0

WRONG X

		A		
		0	1	
B	0	0	1	✓
	1	1	1	

RIGHT ✓

# Karnaugh Maps - Rules of Simplification (SOP Expression)

- Groups must contain 1, 2, 4, 8, or in general  $2^n$  cells.
- That is if  $n = 1$ , a group will contain two 1's since  $2^1 = 2$ .
  - If  $n = 2$ , a group will contain four 1's since  $2^2 = 4$ .

A \ B	0	1
	0	1
0	1	1
1	0	0

Group of 2

RIGHT ✓

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

Group of 3

WRONG ✗

A \ B	0	1
	0	1
0	1	1
1	1	1

Group of 4

RIGHT ✓

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

Group of 5

WRONG ✗

# Karnaugh Maps - Rules of Simplification (SOP Expression)

- Each group should be as large as possible.

$\backslash AB$	00	01	11	10
C 0	1	1	1	1
1	0	0	1	1

RIGHT ✓

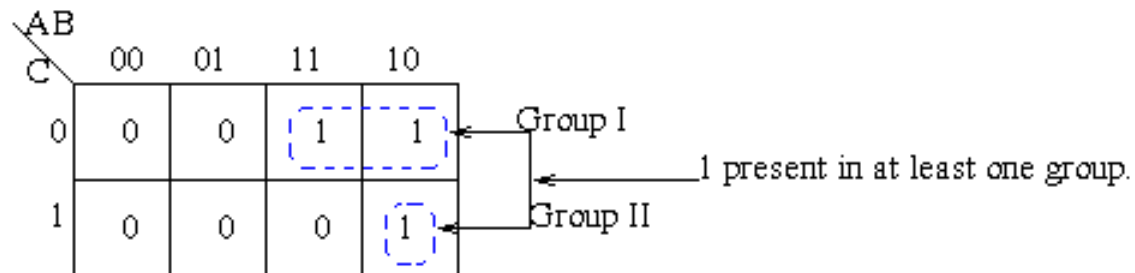
$\backslash AB$	00	01	11	10
C 0	1	1	1	1
1	0	0	1	1

WRONG ✗

(Note that no Boolean laws broken,  
but not sufficiently minimal)

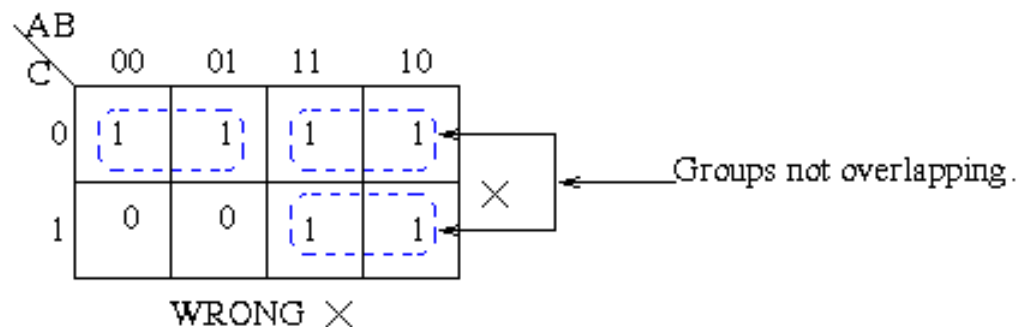
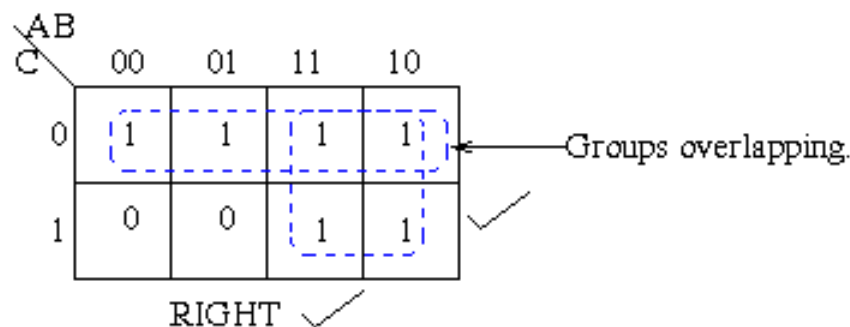
# Karnaugh Maps - Rules of Simplification (SOP Expression)

- Each cell containing a 1 must be in at least one group.



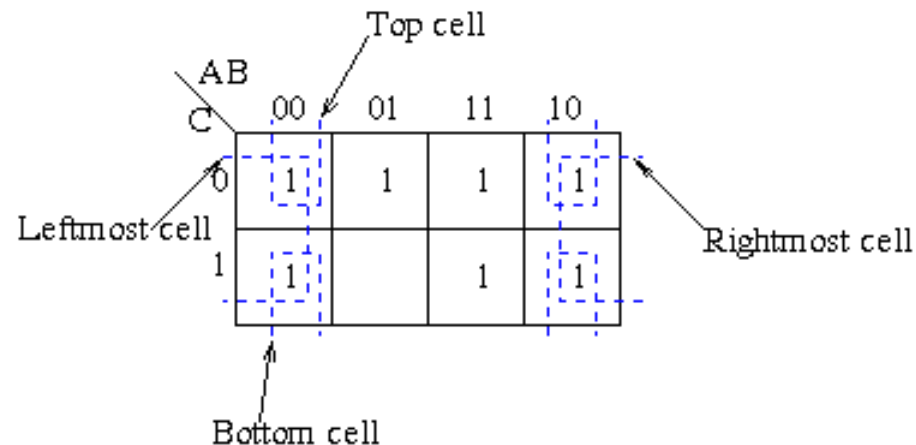
# Karnaugh Maps - Rules of Simplification (SOP Expression)

- Groups may overlap.



# Karnaugh Maps - Rules of Simplification (SOP Expression)

- Groups may wrap around the table.
- The **leftmost** cell in a row may be grouped with the rightmost cell and
- The top cell in a column may be grouped with the bottom cell.





# Karnaugh Maps - Rules of Simplification

## (SOP Expression)

- There should be as few groups as possible, as long as this does not contradict any of the previous rules.

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

RIGHT ✓

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

WRONG ✗

## Karnaugh Maps - Rules of Simplification

### (SOP Expression)

1. No 0's allowed in the groups.
2. No diagonal grouping allowed.
3. Groups should be as large as possible.
4. Only power of 2 number of cells in each group.
5. Every 1 must be in at least one group.
6. Overlapping allowed.
7. Wrap around allowed.
8. Fewest number of groups are considered.
9. Redundant groups ignored

# Minterms

- A **minterm** is a special product of literals, in which each input variable appears exactly once.
- A function with  $n$  variables has  $2^n$  minterms (since each variable can appear complemented or not)
- A three-variable function, such as  $f(x,y,z)$ , has  $2^3 = 8$  minterms:

$x'y'z'$	$x'y'z$	$x'yz'$	$x'yz$
$xy'z'$	$xy'z$	$xyz'$	$xyz$

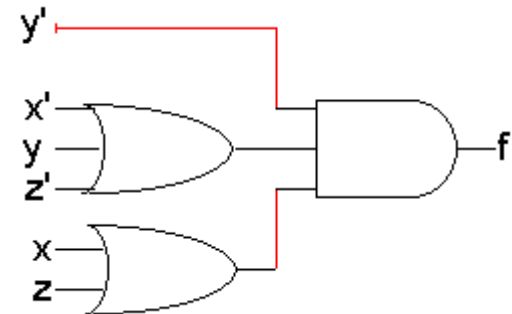
- Each minterm is true for exactly one combination of inputs:

Minterm	Is true when...	Shorthand
$x'y'z'$	$x=0, y=0, z=0$	$m_0$
$x'y'z$	$x=0, y=0, z=1$	$m_1$
$x'yz'$	$x=0, y=1, z=0$	$m_2$
$x'yz$	$x=0, y=1, z=1$	$m_3$
$xy'z'$	$x=1, y=0, z=0$	$m_4$
$xy'z$	$x=1, y=0, z=1$	$m_5$
$xyz'$	$x=1, y=1, z=0$	$m_6$
$xyz$	$x=1, y=1, z=1$	$m_7$

# The dual idea: products of sums

---

- Just to keep you on your toes...
  - A **product of sums (POS)** expression contains:
    - Only AND (product) operations at the "outermost" level
    - Each term must be a sum of literals
- $$f(x,y,z) = y' (x' + y + z') (x + z)$$
- Product of sums expressions can be implemented with two-level circuits
    - literals and their complements at the "0th" level
    - *OR gates* at the first level
    - a single *AND gate* at the second level
  - Compare this with sums of products



# Maxterms

- A **maxterm** is a *sum* of literals, in which each input variable appears exactly once.
- A function with  $n$  variables has  $2^n$  maxterms
- The maxterms for a three-variable function  $f(x,y,z)$ :

$$\begin{array}{cccc} x' + y' + z' & x' + y' + z & x' + y + z' & x' + y + z \\ x + y' + z' & x + y' + z & x + y + z' & x + y + z \end{array}$$

- Each maxterm is *false* for exactly one combination of inputs:

Maxterm	Is <i>false</i> when...	Shorthand
$x + y + z$	$x=0, y=0, z=0$	$M_0$
$x + y + z'$	$x=0, y=0, z=1$	$M_1$
$x + y' + z$	$x=0, y=1, z=0$	$M_2$
$x + y' + z'$	$x=0, y=1, z=1$	$M_3$
$x' + y + z$	$x=1, y=0, z=0$	$M_4$
$x' + y + z'$	$x=1, y=0, z=1$	$M_5$
$x' + y' + z$	$x=1, y=1, z=0$	$M_6$
$x' + y' + z'$	$x=1, y=1, z=1$	$M_7$

# Minterms and maxterms are related

---

- Any minterm  $m_i$  is the complement of the corresponding maxterm  $M_i$

Minterm	Shorthand	Maxterm	Shorthand
$x'y'z'$	$m_0$	$x + y + z$	$M_0$
$x'y'z$	$m_1$	$x + y + z'$	$M_1$
$x'yz'$	$m_2$	$x + y' + z$	$M_2$
$x'yz$	$m_3$	$x + y' + z'$	$M_3$
$xy'z'$	$m_4$	$x' + y + z$	$M_4$
$xy'z$	$m_5$	$x' + y + z'$	$M_5$
$xyz'$	$m_6$	$x' + y' + z$	$M_6$
$xyz$	$m_7$	$x' + y' + z'$	$M_7$

- For example,  $m_4' = M_4$  because  $(xy'z')' = x' + y + z$

# Product of maxterms form

- Every function can be written as a *unique product of maxterms*
- If you have a truth table for a function, you can write a product of maxterms expression by picking out the rows of the table where the function output is 0. (Be careful if you're writing the actual literals!)

x	y	z	f(x,y,z)	f'(x,y,z)
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

$$\begin{aligned}
 f &= (x' + y + z)(x' + y + z')(x' + y' + z') \\
 &= M_4 M_5 M_7 \\
 &= \text{✌} M(4,5,7)
 \end{aligned}$$

$$\begin{aligned}
 f' &= (x + y + z)(x + y + z')(x + y' + z) \\
 &\quad (x + y' + z')(x' + y' + z) \\
 &= M_0 M_1 M_2 M_3 M_6 \\
 &= \text{✌} M(0,1,2,3,6)
 \end{aligned}$$

f' contains all the maxterms not in f

# Converting between standard forms

---

- We can convert a sum of minterms to a product of maxterms

From before  $f = \sum m(0,1,2,3,6)$

and  $f' = \sum m(4,5,7)$

$$= m_4 + m_5 + m_7$$

complementing  $(f')' = (m_4 + m_5 + m_7)'$

so  $f = m_4' m_5' m_7'$  [ DeMorgan's law ]  
 $= M_4 M_5 M_7$  [ By the previous page ]  
 $= \text{✌} M(4,5,7)$

- In general, just replace the minterms with maxterms, using maxterm numbers that don't appear in the sum of minterms:

$$f = \sum m(0,1,2,3,6)$$

$$= \text{✌} M(4,5,7)$$

- The same thing works for converting from a product of maxterms to a sum of minterms



# Switching Algebra – T10

- $X \cdot Y + X \cdot Y' = X$

- 

	WX	00	01	11	10
YZ	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

Minterm 5 and Minterm 13:

$$X \cdot Y' \cdot Z$$

Minterm 1 and Minterm 9:

$$X' \cdot Y' \cdot Z$$

$$Y' \cdot Z$$

Each merge removes one literal

$2^i$  cells  $\rightarrow (n - i)$  literals

Corresponding product terms:

covers only 1: variables

covers only 0: complement of variables

covers both 0 and 1: not included

# More Examples

- **Don't cares:** the output does not matter for these input combinations, or they never appear as valid input

$$F = \Sigma(4, 12, 13, 14, 15) + d(0, 5, 8)$$

Minimal Sum:

$$F = A \cdot B + C' \cdot D'$$

or

$$F = A \cdot B + C' \cdot B$$

Minimal Product:

$$F = B \cdot (A + C')$$

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

# K-Map – 3 Variables

## K-mapping & Minimization Steps

Step 1: generate K-map

- Put a 1 in all specified minterms
- Put a 0 in all other boxes (optional)

Step 2: group all adjacent 1s without including any 0s

- All groups (aka *prime implicants*) must be rectangular and contain a “power-of-2” number of 1s
  - 1, 2, 4, 8, 16, 32, ...
- An essential group (aka *essential prime implicant*) contains at least 1 minterm not included in any other groups
  - A given minterm may be included in multiple groups

Step 3: define product terms using variables common to all minterms in group

Step 4: sum all essential groups plus a minimal set of remaining groups to obtain a minimum SOP

# K-Map – 3 Variables

- $Z = \sum_{A,B,C}(1,3,6,7)$

- Recall SOP minterm implementation

- 8 gates
- 27 gate I/O

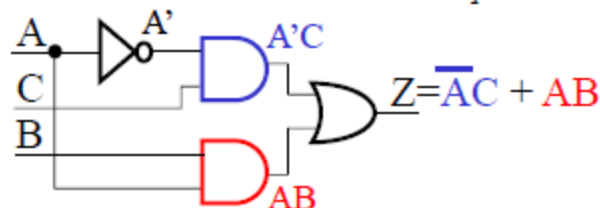
- K-map results

- 4 gates
- 11 gate I/O

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

essential prime implicants

Note: this group not needed since 1s are already covered



A	B	C	Z	Row value
0	0	0	0	0
0	0	1	1	1
0	1	0	0	2
0	1	1	1	3
1	0	0	0	4
1	0	1	0	5
1	1	0	1	6
1	1	1	1	7

# K-Map – 3 Variable

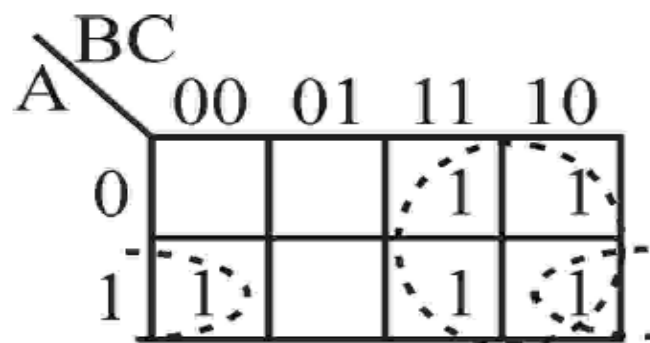
## K-map Minimization Goals

- Larger groups:
  - Smaller product terms
    - Fewer variables in common
  - Smaller AND gates
    - In terms of number of inputs
- Fewer groups:
  - Fewer product terms
    - Fewer AND gates
    - Smaller OR gate
      - ✓ In terms of number of inputs
- Alternate method:
  - Group 0s
    - Could produce fewer and/or smaller product terms
  - Invert output
    - Use NOR instead of OR gate

# K-Map – 3 variables

## 3-input Example

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



$$Y = AC' + B$$

Direct from truth table:  $Y = A'BC' + A'BC + AB'C' + ABC' + ABC$

# Karnaugh Maps: A Geometric Approach

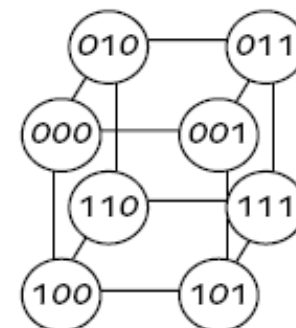
**K-Map:** a truth table arranged so that terms which differ by exactly one variable are adjacent to one another so we can see potential reductions easily.

Truth Table

C	A	B	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

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

It's cyclic. The left edge is adjacent to the right edge. (It's really just a flattened out cube).



# Finding Subcubes

We can identify clusters of “irrelevant” variables by circling adjacent subcubes of 1s. A subcube is just a lower dimensional cube.

$C \backslash AB$	00	01	11	10
0	0	0	1	1
1	0	1	1	0

$\begin{matrix} \backslash AB \\ CD \backslash \end{matrix}$	00	01	11	10
00	0	0	1	0
01	0	1	1	1
11	0	1	0	1
10	1	0	0	1

The best strategy is generally a greedy one.

- Circle the largest N-dimensional subcube ( $2^N$  adjacent 1's)
- Continue circling the largest remaining subcubes (even if they overlap previous ones).
- Circle smaller and smaller subcubes until no 1s are left.



# Write Down Equations

Write down a product term for the portion of each cluster/subcube that is invariant.

$C \backslash AB$	00	01	11	10
0	0	0	1	1
1	0	1	1	0

$$Y = \bar{C}A + CB$$

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

$$Y = ABC\bar{C} + \bar{A}BD + A\bar{B}D + \bar{B}C\bar{D}$$

# Recap: K-map Minimization

- 1) Copy truth table into K-Map
- 2) Identify subcubes,  
selecting the largest available subcube at each step, even if it involves some overlap with previous cubes, until all ones are covered. (Try: 4x4, 2x4 and 4x2, 1x4 and 4x1, 2x2, 2x1 and 1x2, finally 1x1)
- 3) Write down the minimal SOP realization

Truth Table

C	B	A	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

The circled terms are called *implicants*. An implicant not completely contained in another implicant is called a *prime implicant*.

C\BA	00	01	11	10
0	0	1	1	0
1	0	0	1	1

$$Y = \bar{C}A + CB$$

# Grouping the minterms together

- The most difficult step is grouping together all the 1s in the K-map.
  - Make **rectangles** around groups of one, two, four or eight 1s.
  - All of the 1s in the map should be included in at least one rectangle.
  - Do not include any of the 0s.

		y	
x	0	1	0
	1	1	1
		z	

- Each group corresponds to one product term. For the simplest result:
  - Make as few rectangles as possible, to minimize the number of products in the final expression.
  - Make each rectangle as large as possible, to minimize the number of literals in each term.
  - It's all right for rectangles to overlap, if that makes them larger.

# Reading the MSP from the K-map

- Finally, you can find the MSP.
  - Each rectangle corresponds to one product term.
  - The product is determined by finding the common literals in that rectangle.

			y	
	0	1	0	0
x	0	1	1	1
		z		

			y	
	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
x	$xy'z'$	$xy'z$	$xyz$	$xyz'$
		z		

- For our example, we find that  $xy + y'z + xz = y'z + xy$ . (This is one of the additional algebraic laws from last time.)

- Simplify the sum of minterms  $m_1 + m_3 + m_5 + m_6$ .

			y	
	$m_0$	$m_1$	$m_3$	$m_2$
x	$m_4$	$m_5$	$m_7$	$m_6$
		z		

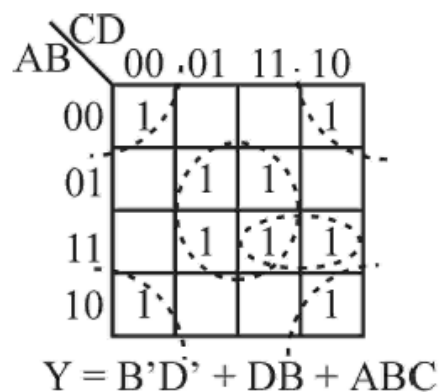
- Here is the filled in K-map, with all groups shown.
  - The magenta and green groups overlap, which makes each of them as large as possible.
  - Minterm  $m_6$  is in a group all by its lonesome.

				y
	0	1	1	0
x	0	1	0	1
			z	

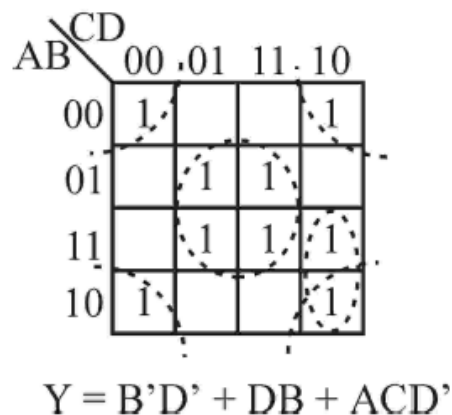
- The final MSP here is  $x'z + y'z + xyz'$ .

# K-Map – 4 variables

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



or



# 4-input K-map

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>Y</i>
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

<i>Y</i> <i>CD</i> \ <i>AB</i>		00	01	11	10
		00	01	11	10
00		1	0	0	1
01		0	1	0	1
11		1	1	0	0
10		1	1	0	1



# 4-input K-map

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

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

$$Y = \bar{A}\bar{C} + \bar{A}BD + A\bar{B}\bar{C} + \bar{B}D$$

# K-maps with Don't Cares

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>Y</i>
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

<i>Y</i> <i>CD</i> \ <i>AB</i>		00	01	11	10
00					
01					
11					
10					

# K-maps with Don't Cares

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>Y</i>
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

		<i>AB</i>			
		00	01	11	10
<i>Y</i> <i>CD</i>	00	1	0	X	1
	01	0	X	X	1
	11	1	1	X	X
	10	1	1	X	X

# K-maps with Don't Cares

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

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

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

# 4-input K-map

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>Y</i>
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

<i>Y</i>		<i>AB</i>			
<i>CD</i>		00	01	11	10
	00				
	01				
	11				
	10				

# 4-input K-map

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>Y</i>
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

<i>Y</i> \ <i>CD</i> \ <i>AB</i>					
		00	01	11	10
<i>CD</i>	00	1	0	0	1
	01	0	1	0	1
	11	1	1	0	0
	10	1	1	0	1

# 4-input K-map

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

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

$$Y = \bar{A}C + \bar{A}BD + A\bar{B}\bar{C} + \bar{B}D$$

# K-maps with Don't Cares

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>Y</i>
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

		<i>AB</i>			
		00	01	11	10
<i>Y</i> <i>CD</i>	00				
	01				
	11				
	10				



# K-maps with Don't Cares

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>Y</i>
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

		<i>AB</i>			
		00	01	11	10
<i>Y</i> <i>CD</i>	00	1	0	X	1
	01	0	X	X	1
	11	1	1	X	X
	10	1	1	X	X

# K-maps with Don't Cares

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	X
1	0	1	1	X
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

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

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

# Minimization Steps (SOP Expression)

1's can be part of more than one group.

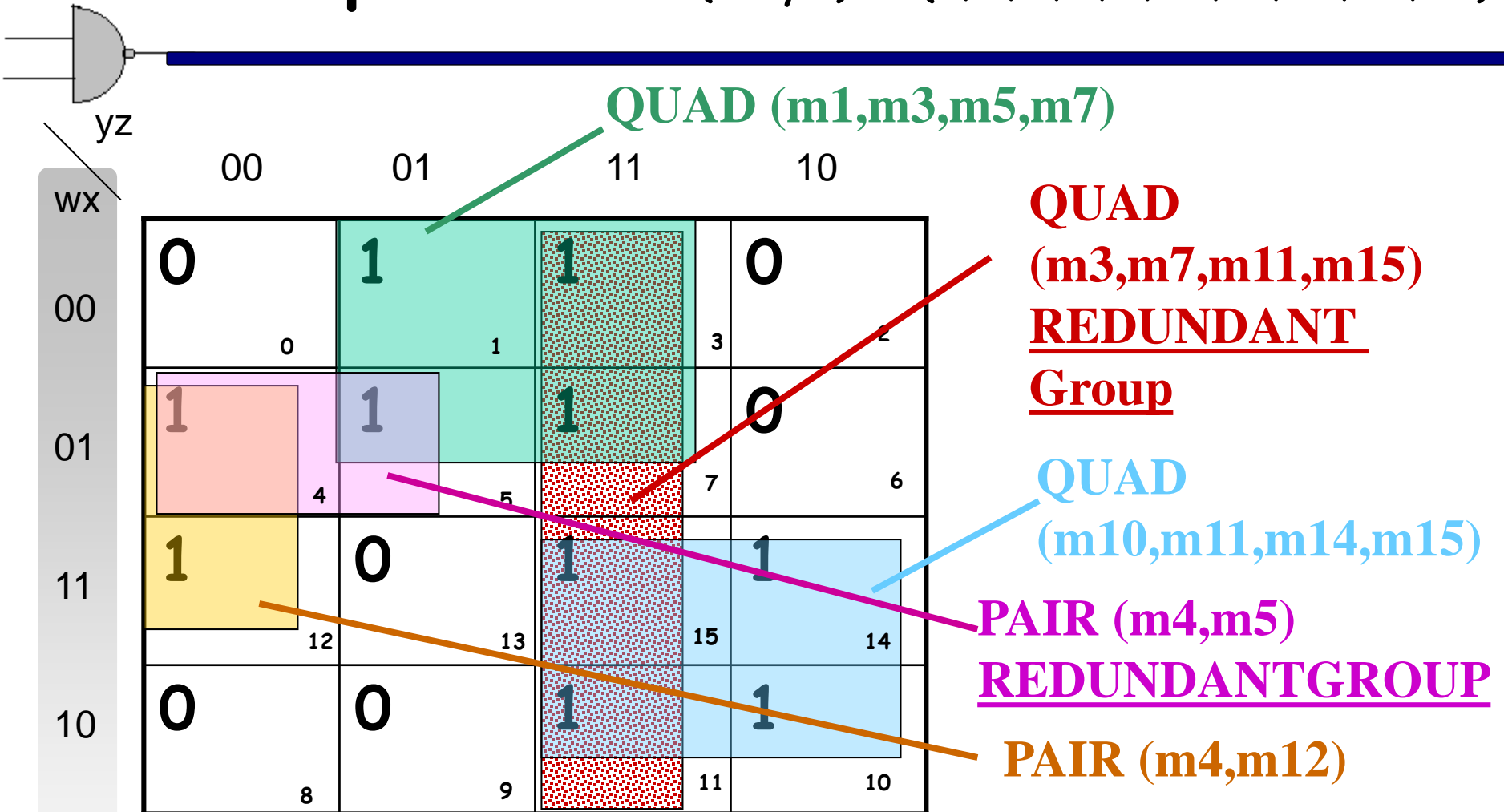
Select the least number of groups that cover all the 1's.

Eliminate Redundant Groups

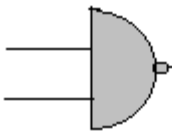
Ensure that every 1 is in a group.

		yz			
		00	01	11	10
wx	00	0 0	1 1	1 3	0 2
	01	1 4	1 5	1 7	0 6
	11	1 12	0 13	1 15	1 14
	10	0 8	0 9	1 11	1 10

**Example:** Reduce  $f(wxyz) = \Sigma(1,3,4,5,7,10,11,12,14,15)$



**Minimized Expression :**  $xy'z' + wy + w'z$



## OCTET REDUCTION ( Group of 8:)

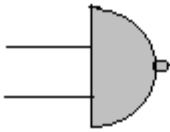
- The term gets reduced by 3 literals i.e. 3 variables change within the group of 8 ( Octets )

W X  
 $\underline{0} \ \underline{0}$   
W.X  
 $\underline{0} \ 1$   
W.X  
 $1 \ 1$   
W.X  
 $1 \ \underline{0}$   
W.X

YZ	$\underline{0} \ \underline{0}$ Y.Z	$\underline{0} \ 1$ Y.Z	$1 \ 1$ Y. Z	$1 \ \underline{0}$ Y. Z
$\underline{0} \ \underline{0}$ W.X	1	1	0	0
$\underline{0} \ 1$ W.X	1	1	0	0
$1 \ 1$ W.X	1	1	0	0
$1 \ \underline{0}$ W.X	1	1	0	0

**OCTET**

(m0,m1,m4,m5,m8,  
m9, m12,m13)

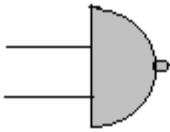


# OCTET REDUCTION ( Group of 8:)

YZ		0 0		0 1		1 1		1 0	
W X		$\overline{Y} \cdot \overline{Z}$		$\overline{Y} \cdot Z$		$Y \cdot Z$		$Y \cdot \overline{Z}$	
0 0	$\overline{W} \cdot \overline{X}$	0	1	1			0		
0 1	$\overline{W} \cdot X$	0	1	1			0		
1 1	$W \cdot X$	0	1	1			0		
1 0	$W \cdot \overline{X}$	0	1	1			0		

**OCTET**

(m1,m3,m5,m7,m9,  
m11, m13,m15)

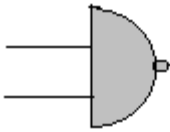


# OCTET REDUCTION ( Group of 8:)

W X \ Y Z		0 0 Y.Z		0 1 Y.Z		1 1 Y.Z		1 0 Y.Z	
		0 0 W.X		0 1 W.X		1 1 W.X		1 0 W.X	
0 0	W.X	1	0	0	1	0	3	1	2
0 1	W.X	1	4	0	5	0	7	1	6
1 1	W.X	1	12	0	13	0	15	1	14
1 0	W.X	1	8	0	9	0	11	1	10

MAP ROLLING

**OCTET**  
(m0,m2,m4,m6,  
m8, m10, m12,m14)



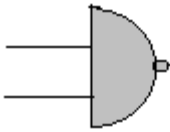
## OCTET REDUCTION ( Group of 8:)

W X \ Y Z		$\begin{smallmatrix} 0 & 0 \\ \hline Y & Z \end{smallmatrix}$	$\begin{smallmatrix} 0 & 1 \\ \hline Y & Z \end{smallmatrix}$	$\begin{smallmatrix} 1 & 1 \\ \hline Y & Z \end{smallmatrix}$	$\begin{smallmatrix} 1 & 0 \\ \hline Y & Z \end{smallmatrix}$
		$\begin{smallmatrix} 0 & 0 \\ \hline W & X \end{smallmatrix}$	$\begin{smallmatrix} 0 & 1 \\ \hline W & X \end{smallmatrix}$	$\begin{smallmatrix} 1 & 1 \\ \hline W & X \end{smallmatrix}$	$\begin{smallmatrix} 1 & 0 \\ \hline W & X \end{smallmatrix}$
		0	1	3	2
		4	5	7	6
		12	13	15	14
		8	9	11	10

**OCTET**

**(m4,m5,m6,m7,m12,  
m13, m14,m15)**





# OCTET REDUCTION ( Group of 8:)

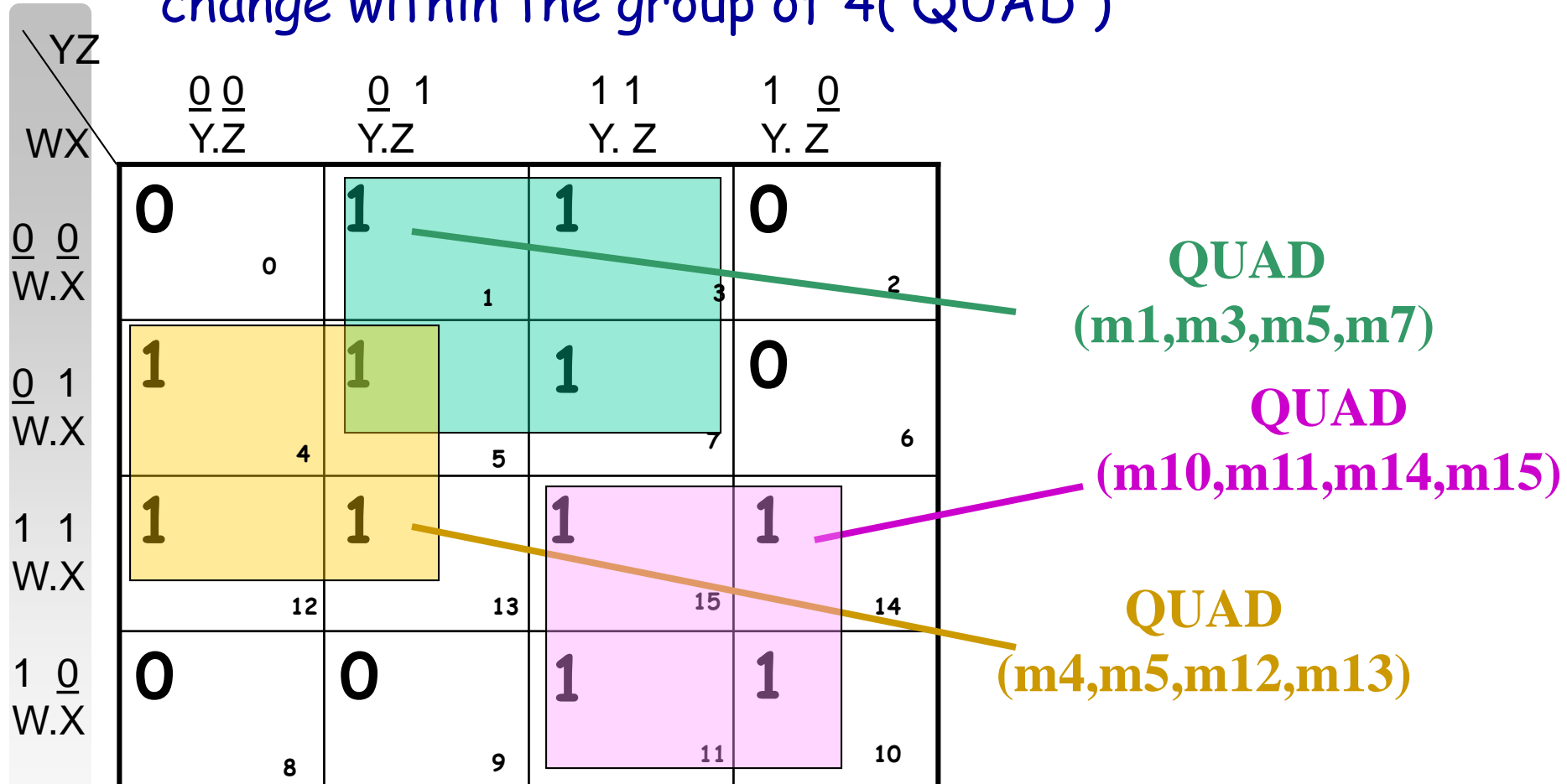
W X \ Y Z		0 0	0 1	1 1	1 0
		Y.Z	Y.Z	Y.Z	Y.Z
0 0	W.X	1 0	1 1	1 3	1 2
0 1	W.X	0 4	0 5	0 7	0 6
1 1	W.X	0 12	0 13	0 15	0 14
1 0	W.X	1 8	1 9	1 11	1 10

MAP ROLLING

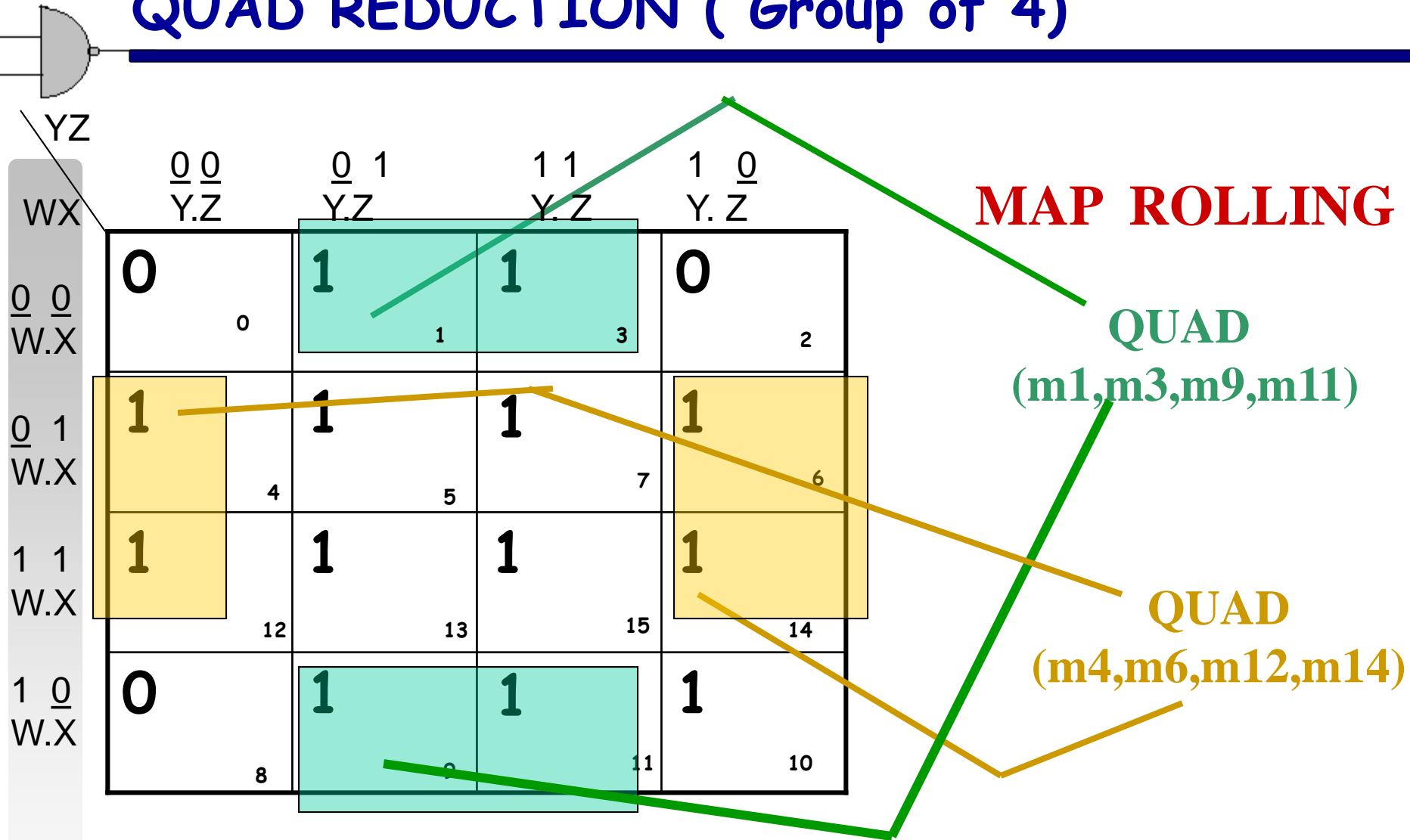
**OCTET**  
(m0,m1,m2,m3  
M8,m9,m10,m11)

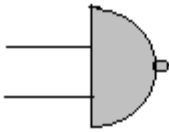
# QUAD REDUCTION ( Group of 4)

- The term gets reduced by 2 literals i.e. 2 variables change within the group of 4( QUAD )



# QUAD REDUCTION ( Group of 4)



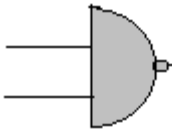


# QUAD REDUCTION ( Group of 4)

## CORNER ROLLING

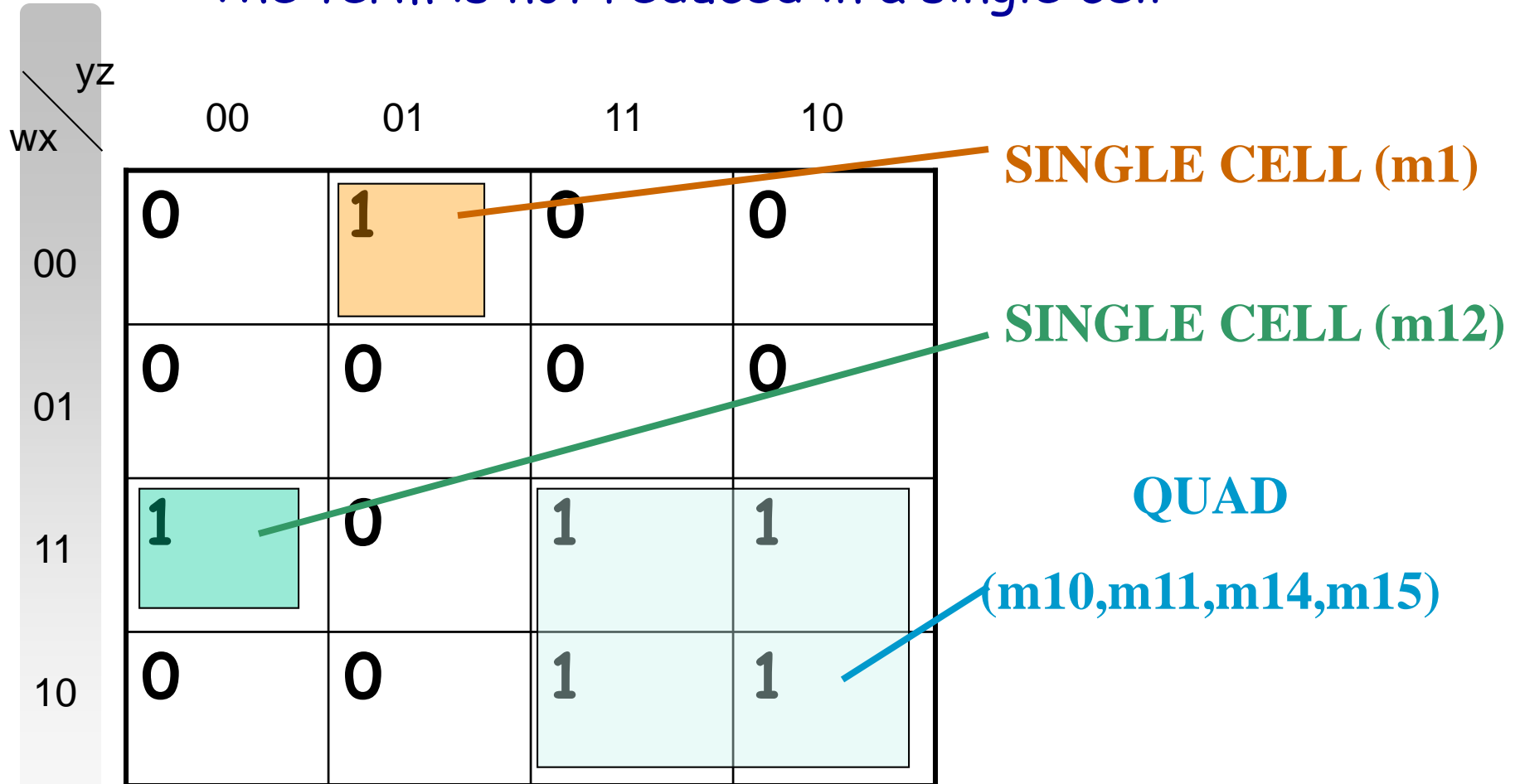
YZ		0 0 Y.Z		0 1 Y.Z		1 1 Y.Z		1 0 Y.Z	
WX									
0 0 W.X	1 0	0 1		0 3		1 2			
0 1 W.X	0 4	0 5		0 7		0 6			
1 1 W.X	0 12	0 13		0 15		0 14			
1 0 W.X	1 8	0 9		0 11		1 10			

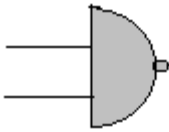
**QUAD**  
(m0,m2,m8,m10)



# SINGLE CELL REDUCTION

- The term is not reduced in a single cell





# PAIR REDUCTION ( Group of 2)

- The term gets reduced by 1 literals i.e. 1 variables change within the group of 2 (PAIR)

change within the group of 2 (PAIR )

YZ WX	$\frac{0}{Y} \frac{0}{Z}$ Y.Z	$\frac{0}{Y} \frac{1}{Z}$ Y.Z	$\frac{1}{Y} \frac{1}{Z}$ Y.Z	$\frac{1}{Y} \frac{0}{Z}$ Y.Z
$\frac{0}{W} \frac{0}{X}$	1 0	0 1	0 3	1 2
$\frac{0}{W} \frac{1}{X}$	0 4	1 5	1 7	0 6
$\frac{1}{W} \frac{1}{X}$	0 12	0 13	0 15	0 14
$\frac{1}{W} \frac{0}{X}$	0 8	0 9	0 11	0 10

MAP ROLLING PAIR (m0,m2)

PAIR (m5,m7)

# 4-Input K-Map

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

		AB			
		00	01	11	10
CD	00				
	01				
	11				
	10				

# 4-Input K-Map

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

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



# **Lecture 4**

## **Combinatorial Building Blocks**