Minimization of Boolean Expression (Contd…)

- Two method can by applied to reduce the Boolean expression –

i) Algebraic
ii) Using Karnaugh Map (K-Map).

Minimization of Boolean Expression (Contd…)

➢ Algebraic Method

- The different Boolean rules and theorems are used to simplify the Boolean expression in this method.

Minimization of Boolean Expression (Contd…)

Solved Problem

Minimize the following Boolean Expression:

1. a’bc + ab’c’ + ab’c + abc’ +abc

   = a’bc + ab’ + ab

   = a’bc + a

2. AB’CD’ + AB’CD + ABCD’ + ABCD

   = AB’C + ABC

   = AC

Minimization of Boolean Expression (Contd…)

Exercise

A. Minimize the following Boolean Expression:

1. X'Y'Z' + X'YZ' + XY'Z' + XYZ'
2. a(b + b'c + b'c')

B. Prove algebraically that

1. (x+y+z)(x'+y+z)=y+z
2. A+A'B'=A+B'

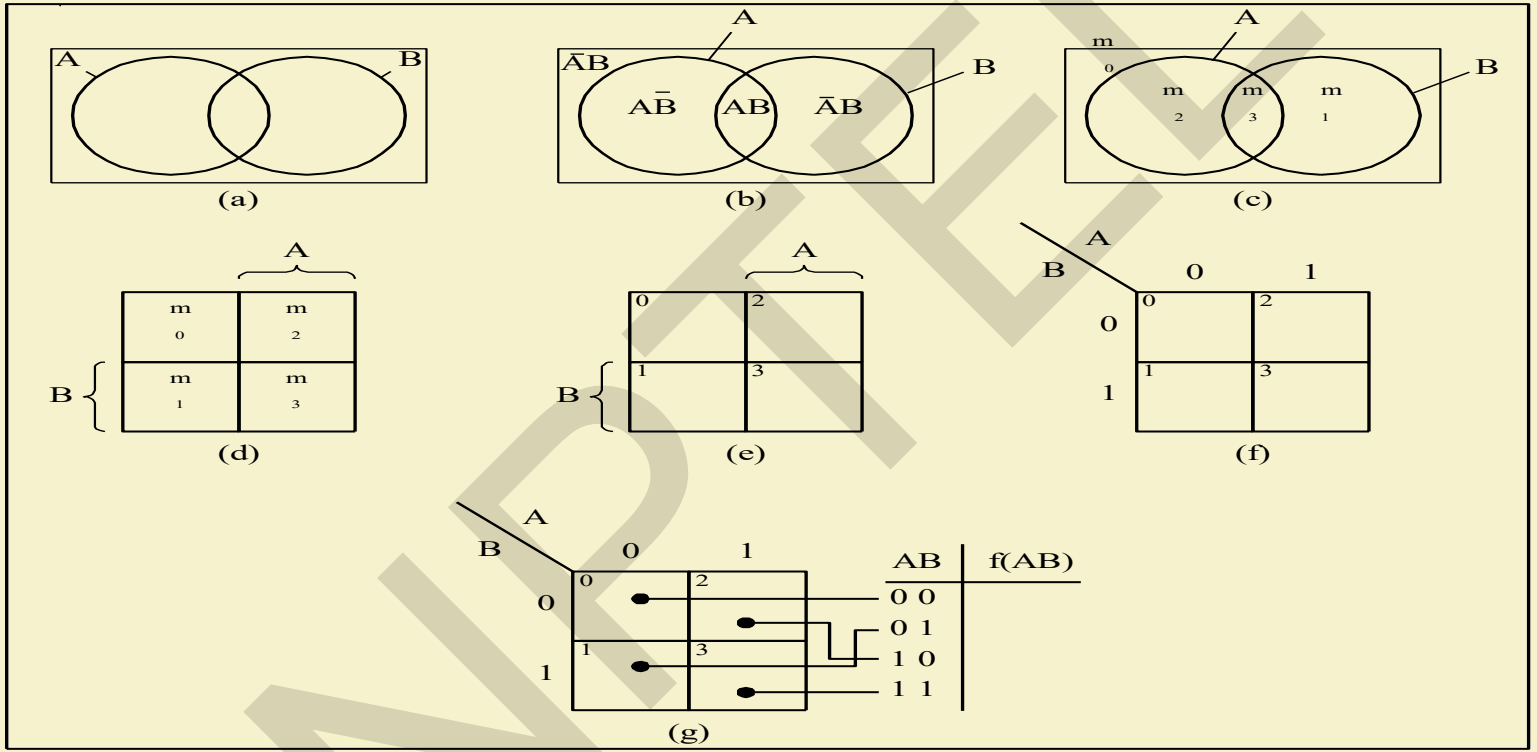
Minimization of Boolean Expression (Contd…)

Karnaugh Map

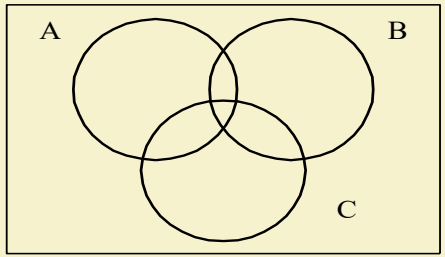
The Karnaugh map (K-map for short), Maurice Karnaugh's 1953 refinement of Edward Veitch's 1952 Veitch diagram, is a method to simplify Boolean algebra expressions.

- K-Maps are a convenient way to simplify Boolean Expressions.
- They can be used for up to 4 or 5 variables.
- They are a visual representation of a truth table.

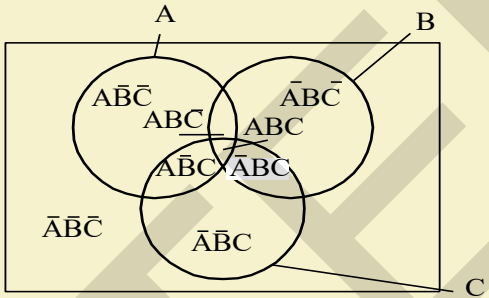
Venn diagram and equivalent K-map for two variables



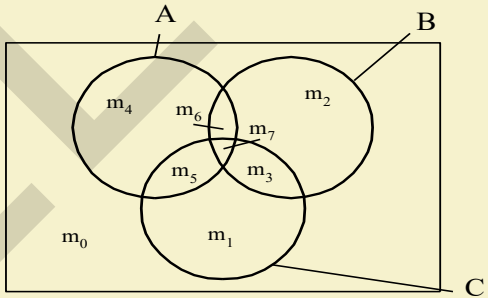
Venn diagram and equivalent K-map for three variables



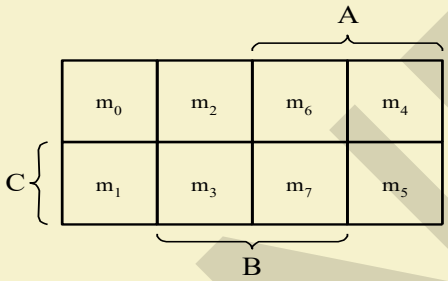
(a)



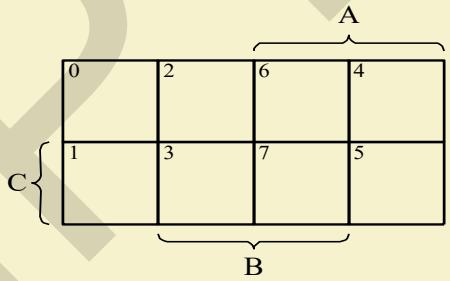
(b)



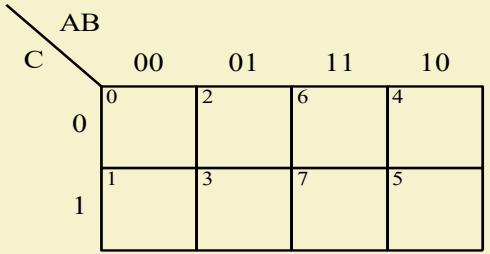
(c)



(d)

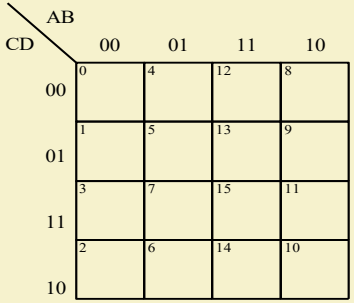


(e)

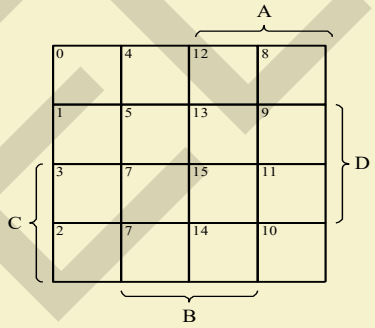


(f)

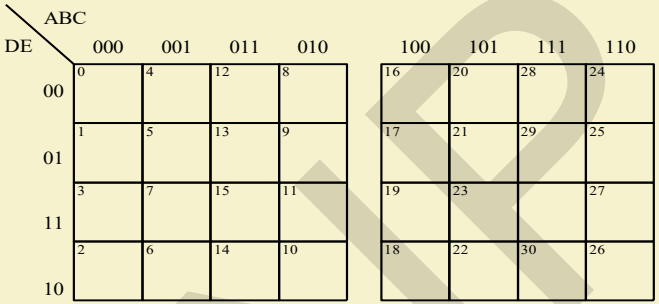
K-maps for four and five variables



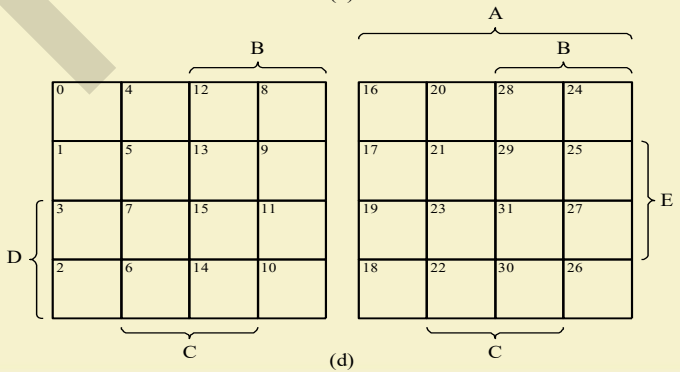
(a)



(b)

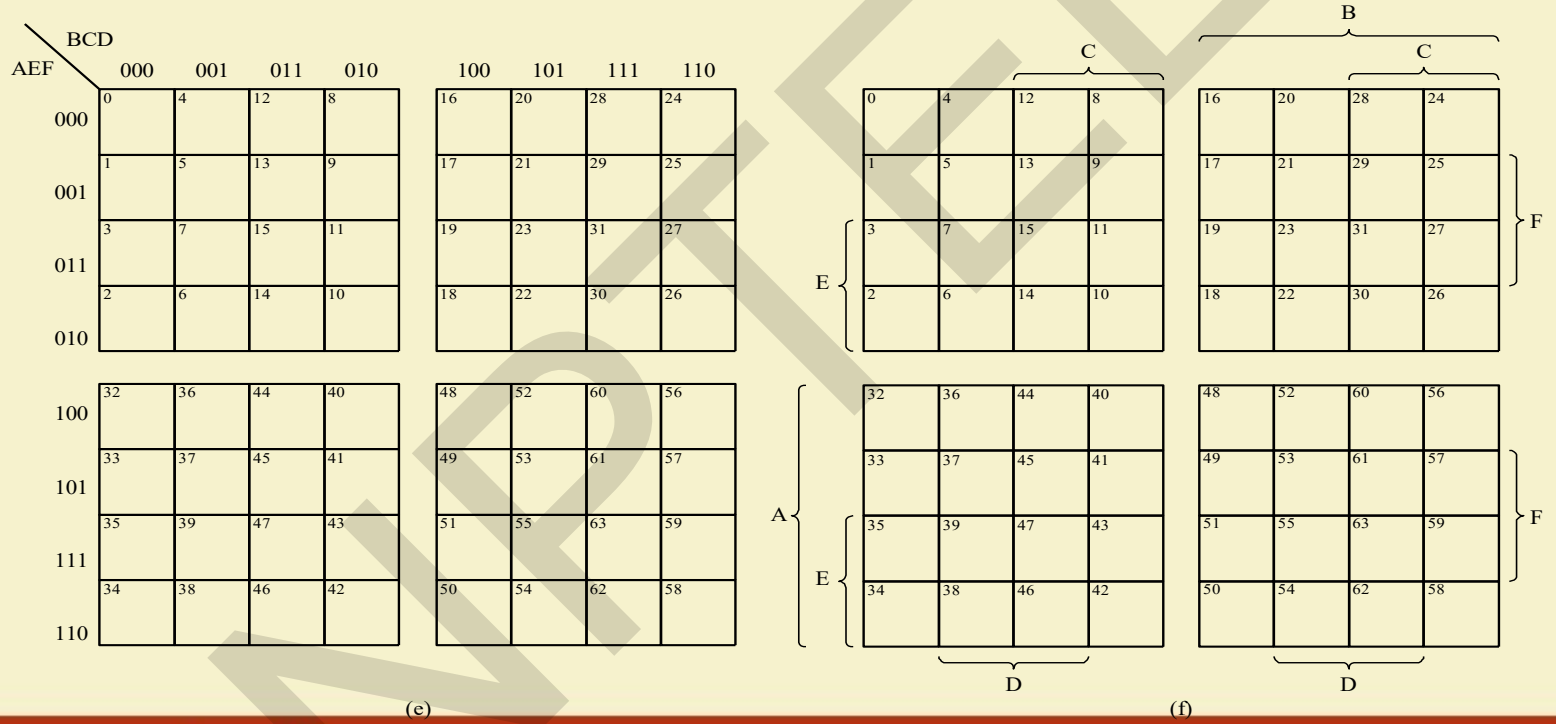


(c)



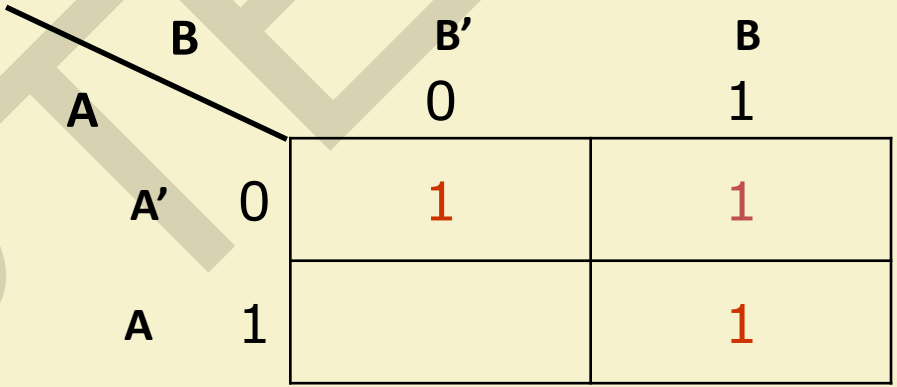
(d)

K-maps for six variables



Truth table to K-Map (2 variable minterm)

A	B	P
0	0	1
0	1	1
1	0	0
1	1	1



The expression is:

$$\bar{A}.\bar{B} + A.\bar{B} + A.B$$

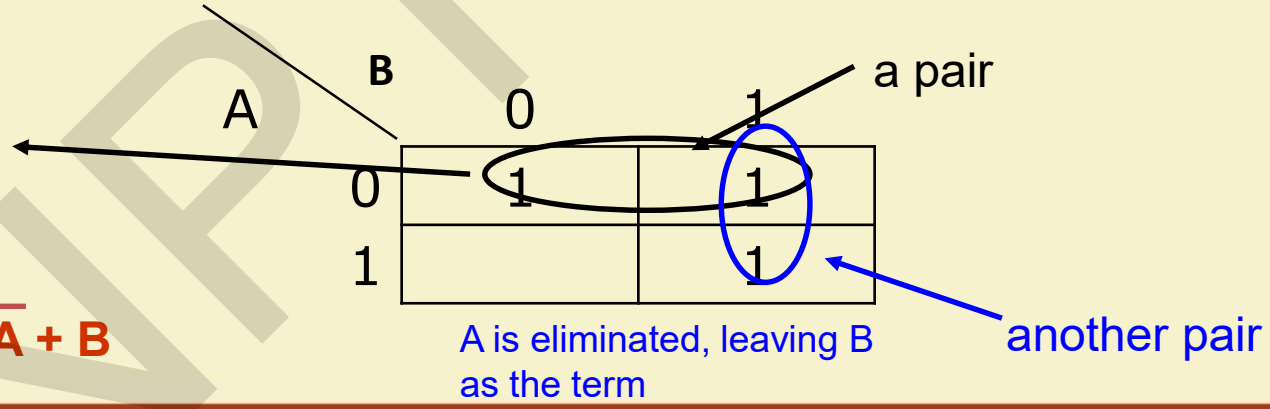
minterms are represented by a 1 in the corresponding location in the K map.

K-Maps (2 Variables k-map contd…)

- Adjacent 1's can be "paired off"
- Any variable which is both a 1 and a zero in this pairing can be eliminated
- Pairs may be adjacent horizontally or vertically

The expression is:
A'.B' + A'.B + A.B

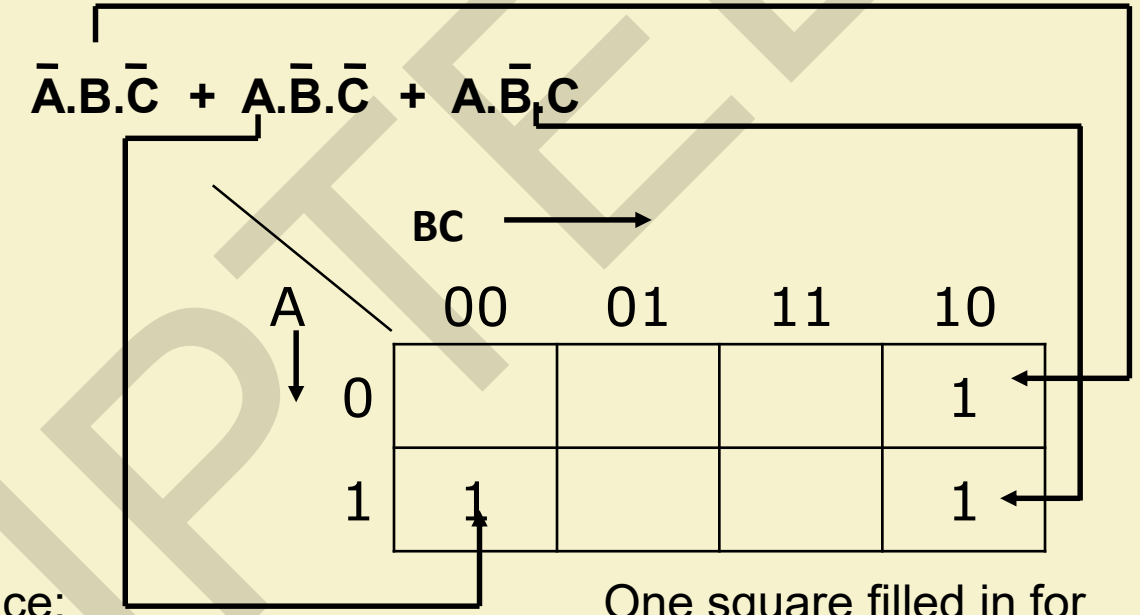
B is eliminated, leaving $\bar{A}$ as the term



After reduction the expression becomes $\bar{A} + B$

- Three Variable K-Map

A	B	C	P
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

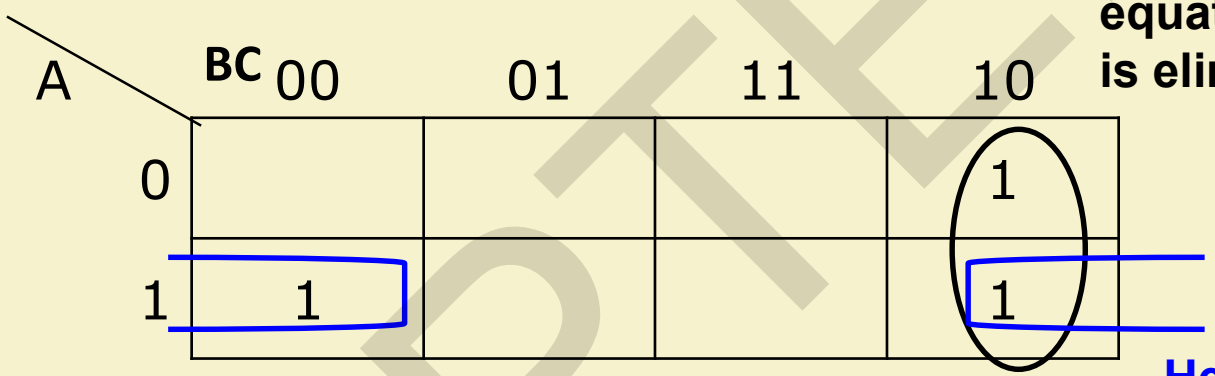


Notice the code sequence:
00 01 11 10 – a Gray code.

One square filled in for each minterm.

Three Variable K-Map (Contd…)

Grouping the Pairs



equates to B.$\overline{C}$ as A is eliminated.

Our truth table simplifies to A.$\overline{C}$ + B.$\overline{C}$ as before.

Here, we can "wrap around" and this pair equates to A.$\overline{C}$ as B is eliminated.

Three Variable K-Map (Contd…)

Expression is ABC+ABC'+A'BC+ABC'

Groups of 4 in a block can be used to eliminate two variables:

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	1
1	1	1	1

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

Groups of 4

QUAD = A'BC+A'BC'+ABC+ABC'
= A'B+AB
=B

Karnaugh Maps - Four Variable K-Map

AB \ CD	00	01	11	10
00	$\bar{A}.\bar{B}.\bar{C}.\bar{D}$	$\bar{A}.\bar{B}.\bar{C}.D$	$\bar{A}.\bar{B}.C.D$	$\bar{A}.\bar{B}.C.\bar{D}$
01	$\bar{A}.B.\bar{C}.\bar{D}$	$\bar{A}.B.\bar{C}.D$	$\bar{A}.B.C.D$	$\bar{A}.B.C.\bar{D}$
11	$A.B.\bar{C}.\bar{D}$	$A.B.\bar{C}.D$	$A.B.C.D$	$A.B.C.\bar{D}$
10	$A.\bar{B}.\bar{C}.\bar{D}$	$A.\bar{B}.\bar{C}.D$	$A.\bar{B}.C.D$	$A.\bar{B}.C.\bar{D}$

K-Map

Reduction Rule

To reduce the Boolean expression, first we have to mark pairs, quads and octets.

Pair – remove one variable

Quad – remove two variables

Octet – remove three variables

Imp – To get the optimum reduction, priority is given to octet first, then quad and then pair.

Karnaugh Maps - Four Variable K-Map

Octet Reduction

AB \ CD	C'D'[00]	C'D[01]	CD[11]	CD'[10]
A'B'[00]	1	1	1	1
A'B[01]	1	1	1	1
AB[11]				
AB'[10]				

A'B'[00]	1	1		
A'B[01]	1	1		
AB[11]	1	1		
AB'[10]	1	1		

Karnaugh Maps - Four Variable K-Map

Octet Reduction

AB \ CD	C'D'[00]	C'D[01]	CD[11]	CD'[10]
A'B'[00]	1	1	1	1
A'B[01]				
AB[11]				
AB'[10]	1	1	1	1

A'B'[00]	1			1
A'B[01]	1			1
AB[11]	1			1
AB'[10]	1			1

Karnaugh Maps - Four Variable K-Map

Quad Reduction

AB \ CD	C'D'[00]	C'D[01]	CD[11]	CD'[10]
A'B'[00]	1	1		1
A'B[01]	1	1		1
AB[11]				
AB'[10]				

A'B'[00]	1			
A'B[01]	1			
AB[11]	1			
AB'[10]	1			

Karnaugh Maps - Four Variable K-Map

Quad Reduction

AB \ CD	C'D'[00]	C'D[01]	CD[11]	CD'[10]
A'B'[00]	1			1
A'B[01]	1			1
AB[11]				
AB'[10]				

A'B'[00]	1			1
A'B[01]				
AB[11]				
AB'[10]	1			1

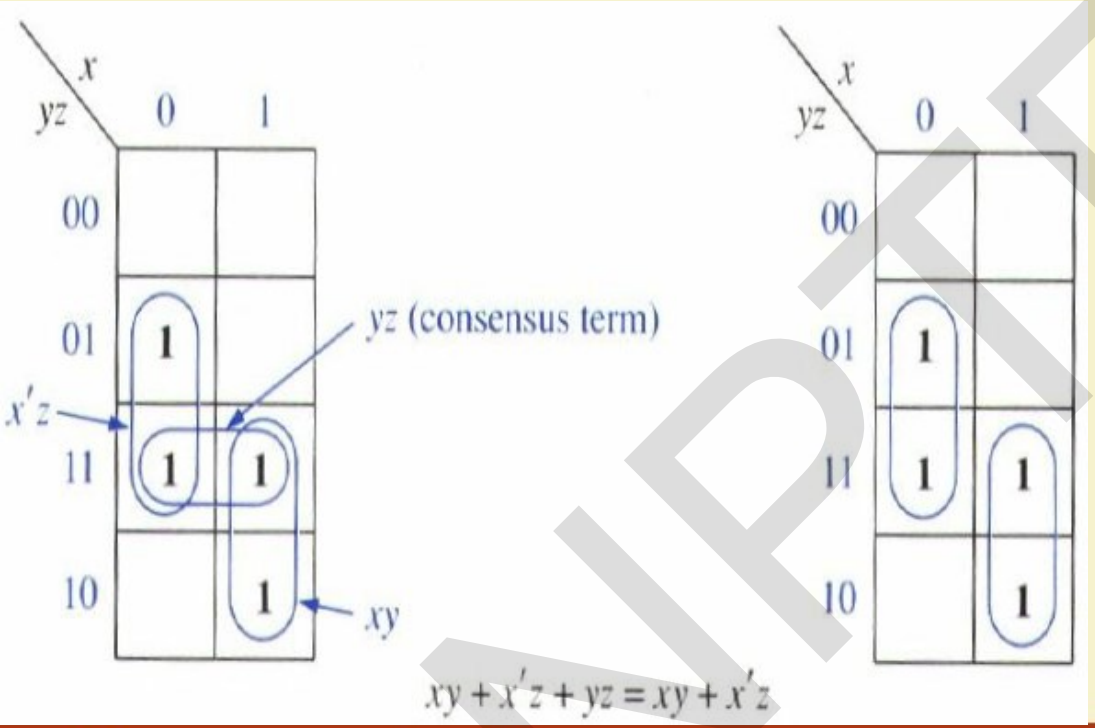
Karnaugh Maps - Four Variable K-Map

Quad Reduction

AB \ CD	C'D'[00]	C'D[01]	CD[11]	CD'[10]
A'B'[00]	1			1
A'B[01]	1			1
AB[11]				
AB'[10]				

A'B'[00]	1			1
A'B[01]				
AB[11]				
AB'[10]	1			1

Redundant Term

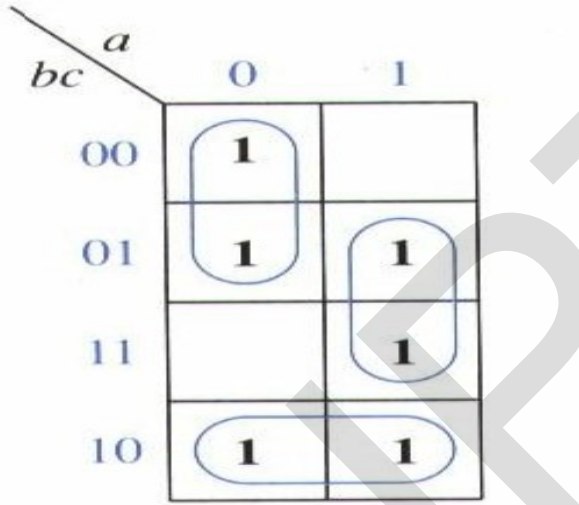


Term *yz* is redundant

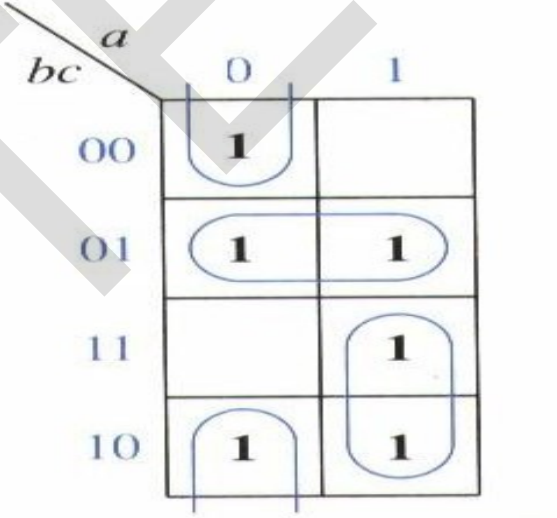
xy + x'z + yz
= xy + x'z +(x+x')yz
= xy + x'z + xyz + x'yz
= (xy + xyz) + (x'z+ x'yz)
= xy(1 + z) + x'z(1+ y)
= xy + x'z

More than one solution

$$F = \sum m(0,1,2,5,6,7)$$

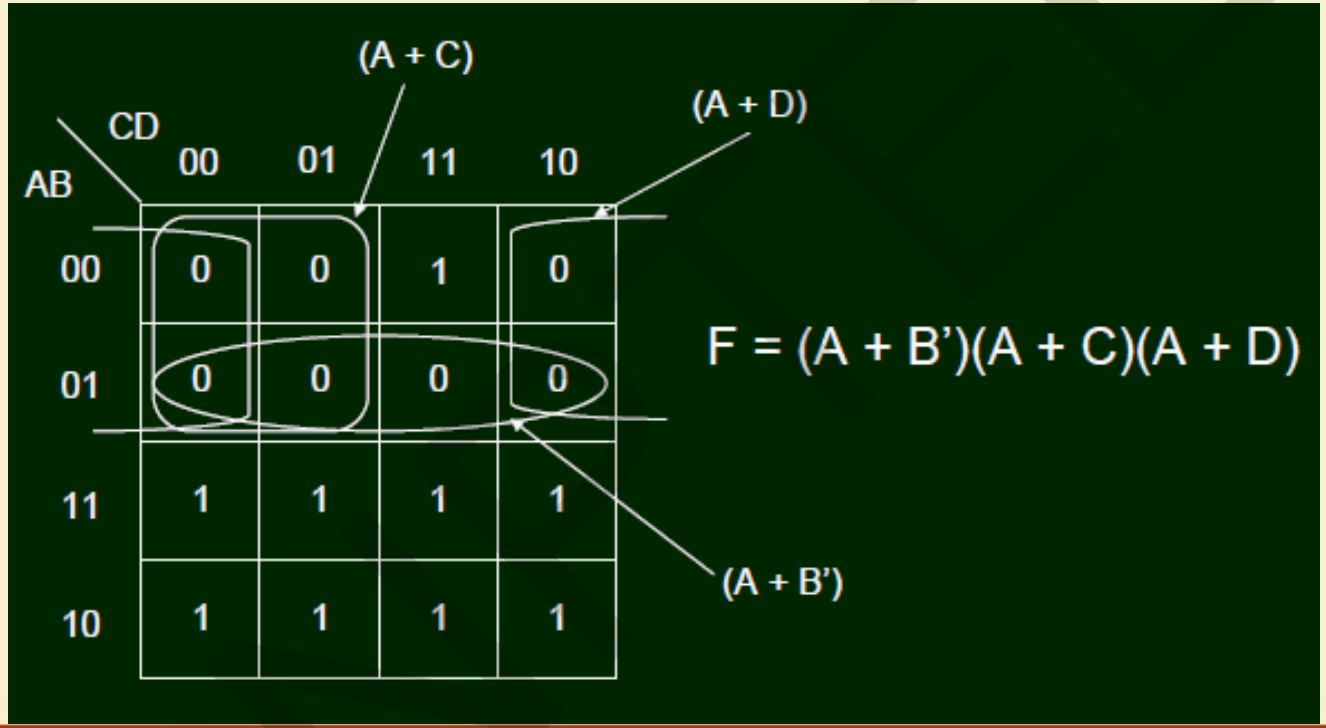


$F = a'b' + bc' + ac$



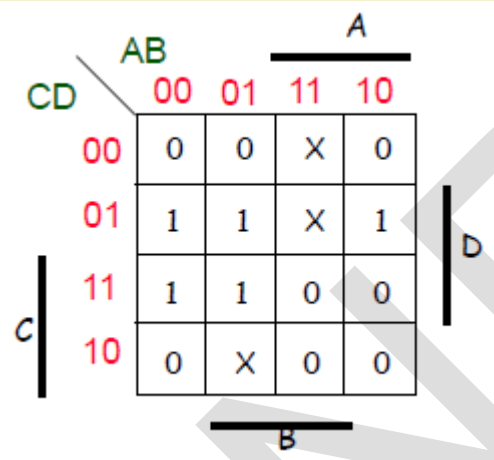
$F = a'c' + b'c + ab$

Four Variable: POS Karnaugh Map



Karnaugh maps: Don’t cares

- In some cases, outputs are undefined
- We “don’t care” if the logic produces a 0 or a 1
- This knowledge can be used to simplify functions.

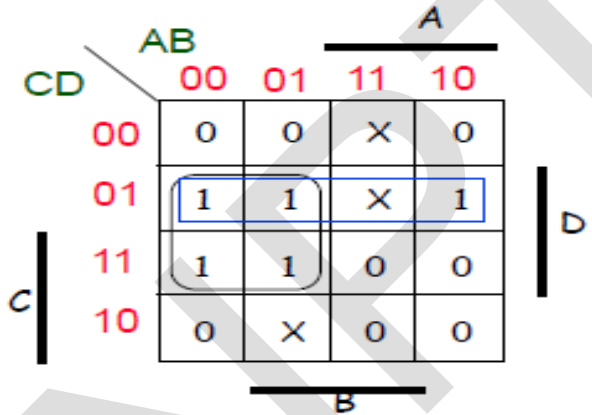


- Treat X’s like either 1’s or 0’s
- Very useful
- OK to leave some X’s uncovered

Karnaugh maps: Don’t cares

- $f(A,B,C,D) = \Sigma\, m(1,3,5,7,9) + d(6,12,13)$
  - without don't cares
    - f =

$A'D + C'D$



A	B	C	D	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	X
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	X
1	1	0	1	X
1	1	1	0	0
1	1	1	1	0

Don’t Care Conditions

- In some situations, we don’t care about the value of a function for certain combinations of the variables.
  - these combinations may be impossible in certain contexts
  - or the value of the function may not matter in when the combinations occur
- In such situations we say the function is *incompletely specified* and there are multiple (completely specified) logic functions that can be used in the design.
  - so we can select a function that gives the simplest circuit
- When constructing the terms in the simplification procedure, we can choose to either cover or not cover the don’t care conditions.

Map Simplification with Don't Cares

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

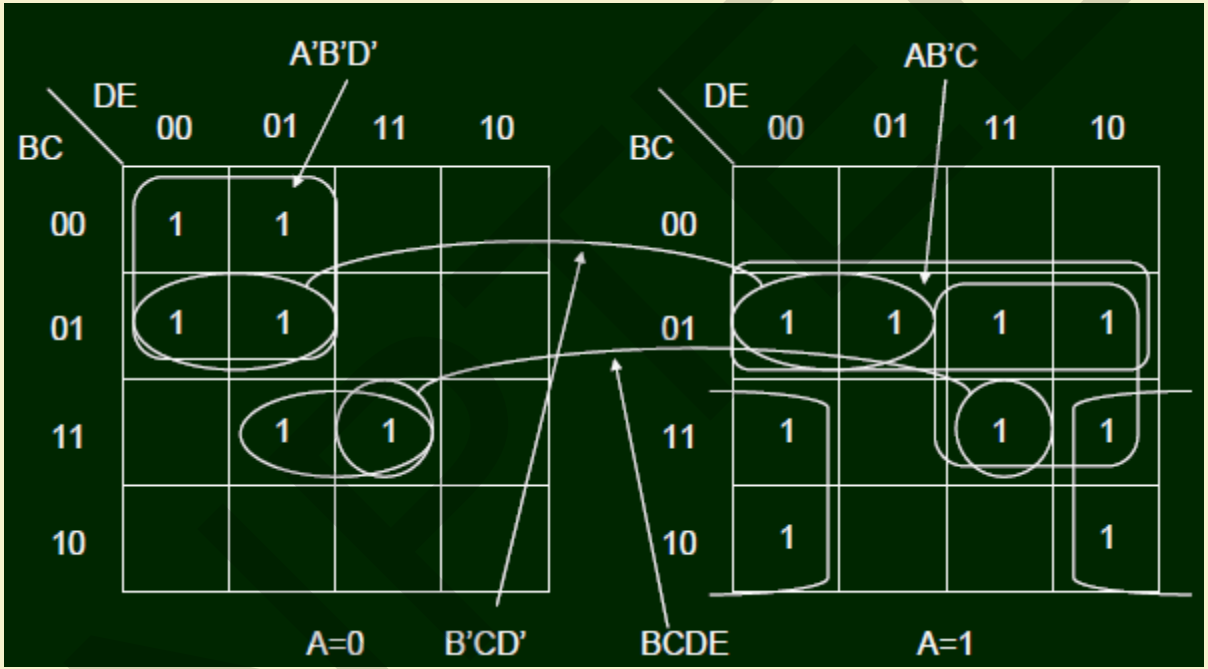
$F=A'C'D+B+AC$

Alternative covering.

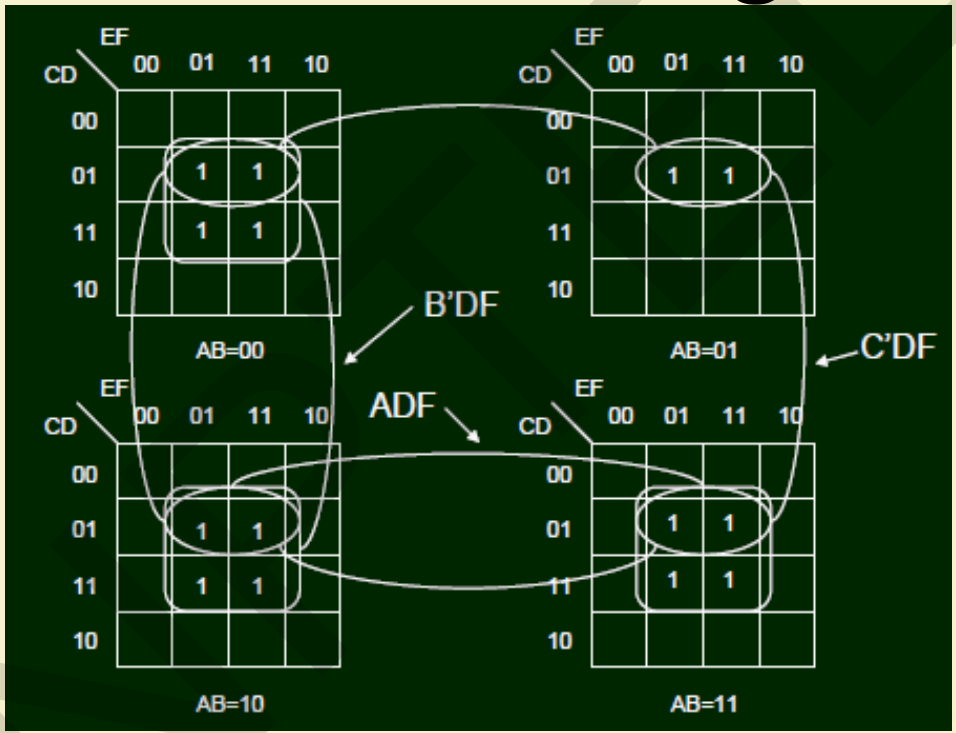
AB \ CD	00	01	11	10
00	0	1	0	0
01	x	x	x	1
11	1	1	1	x
10	x	0	1	1

$F=A'B'C'D+ABC'+BC+AC$

Five Variable Karnaugh Map

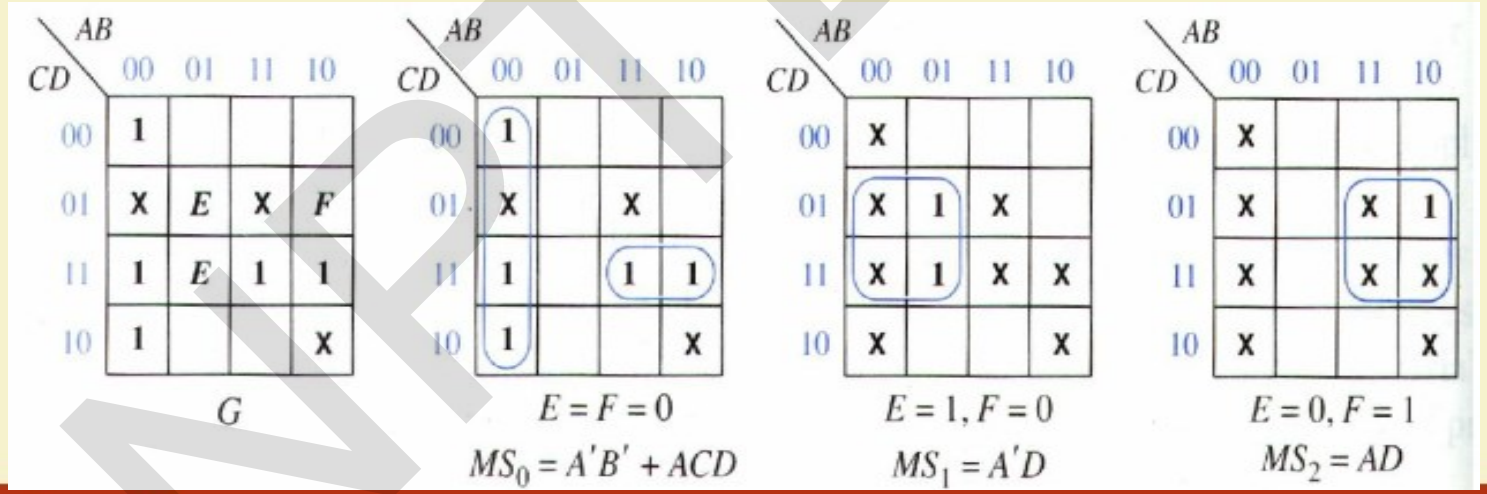


Six Variable Karnaugh Map



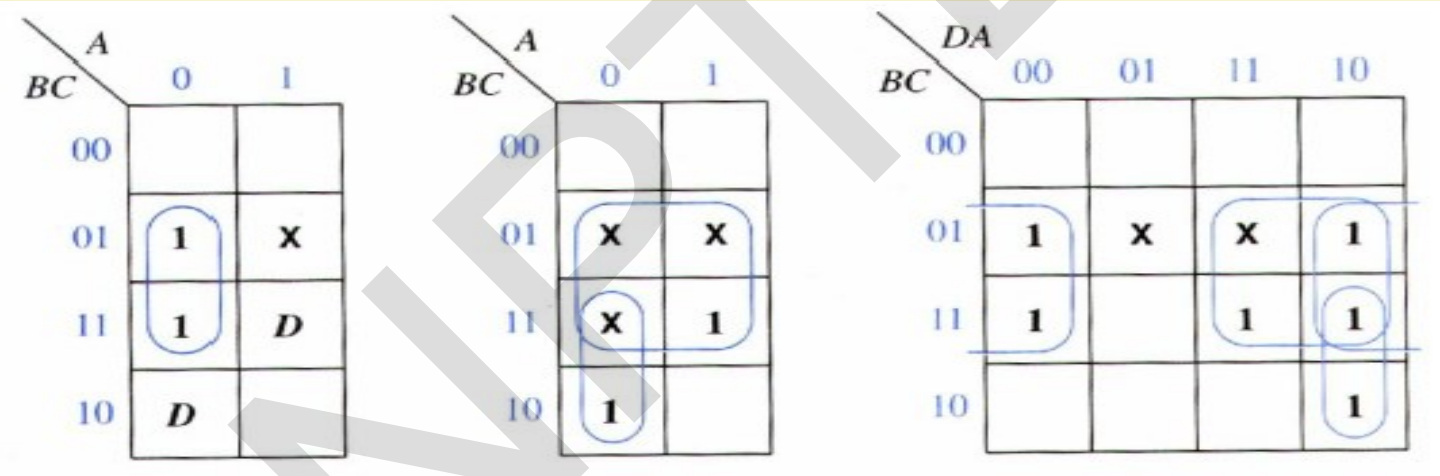
Simplification using Map-Entered Variables

- Extend K-Map for more variables
- When variable ‘E’ appears in a square, if E=1, then the corresponding minterm is present in G.
- $G(A,B,C,D,E,F) = m_0 + m_2 + m_3 + Em_5 + Em_7 + Fm_9 + m_{11} + m_{15} + (\text{don't cares})$



An Example

- F(A, B, C, D) = A'B'C + A'BC + A'BC'D + ABCD + don't care (AB'C)
- Consider D as a map-entered variable
- When D = 0, F = A'C
- When D = 1, F = C + A'B
- F = A'C + D(C+ A'B) = A'C + CD + A'BD



K-Map: Another View

Implicants

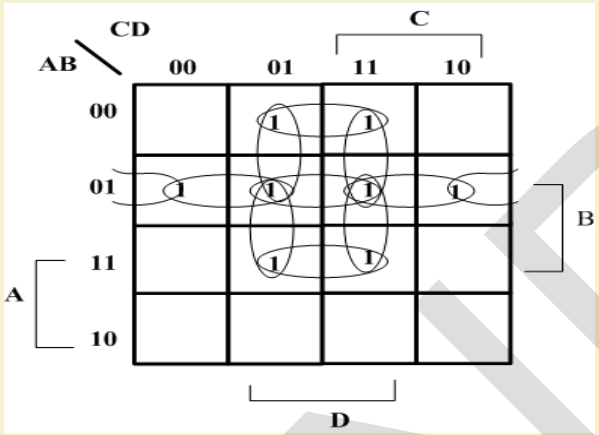
- Must cover all 1s of the function on K-map
  - Each 1 can be used multiple times in generating the terms of the expression
- When we group 1's on a K-map, generating a term, that term is an *implicant* of the function
  - Prime Implicants
  - Essential Prime Implicants

Prime Implicant

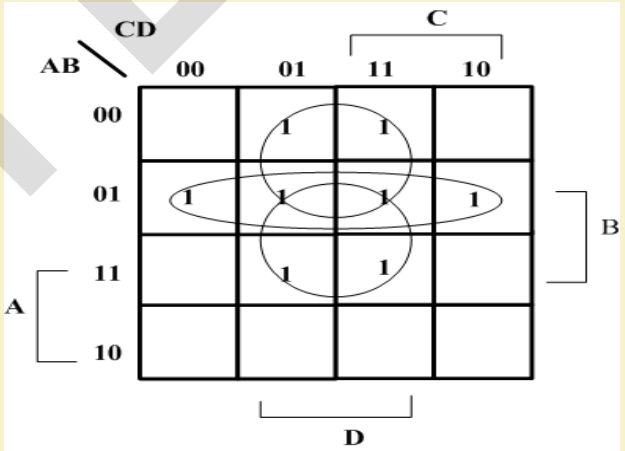
- If removal of an any literal from an implicant P results in a product term that is not an implicant of the function, then P is a prime implicant.
- Let's take a look at this.

Implicants

- Implicants with 2 1s with 4 1s



A'B'D
A'BD
ABD
A'BC'
A'BC
A'BD'
A'C'D
A'CD
BC'D
BCD



A'D
A'B
BD

Which are prime implicants?

- Consider
  - A'B'D
    - Remove A' → B'D THUS meets previous statement
- However, on an n-variable map, the set of prime implicants corresponds to the set of rectangles made up of 2^m squares containing 1s with each rectangle containing as many squares as possible.
- Thus, A'B'D is not a prime implicant. It is an implicant, just not a prime implicant.

Prime implicants of the function

- Have
  - A’D A’B and BD
  - Removal of any literal from any of these terms results in an implicant that is not implicant of the function.
  - They also cover all 1s of the function and are not contained in some larger implicant
  - These are the prime implicants of the function

Some general statements on PI

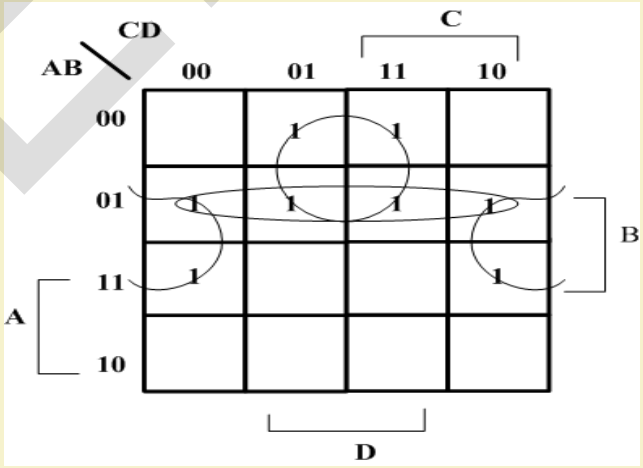
- A single 1 on a map is a prime implicant if is not adjacent to any other 1 of the function.
- Two adjacent 1s on a map represent a prime implicant, provided that they are not within a rectangle of 4 or more squares containing 1s.
- Four 1’s that are an implicant are a prime implicant if they are not within a group of 8.

Essential Prime Implicant

- A prime implicant that contains a 1 that is not covered by any other prime implicant of the function is an essential prime implicant.
  IT MUST BE INCLUDED IN ANY MINIMAL REPRESENTATION OF THE FUNCTION.

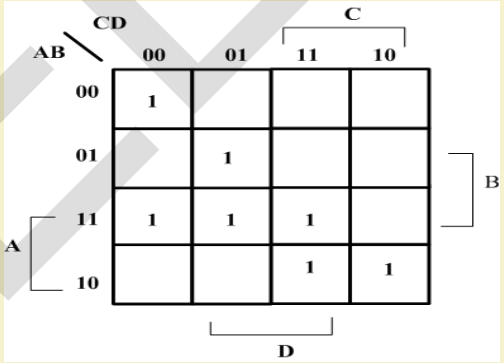
Example

- Consider the example
- Prime Implicants
  - A'D
  - BD'
  - A'B
- Essential Prime Implicants
  - A'D
  - BD'
- So F=A'D + BD'

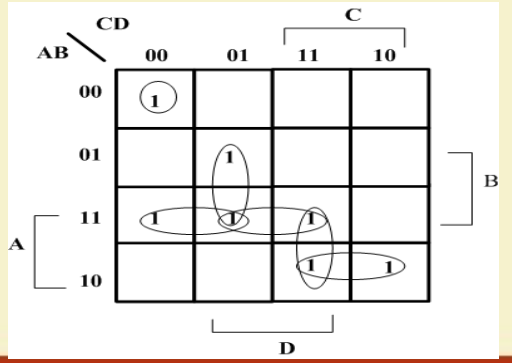


Using non-essential prime impicants

- Consider the example

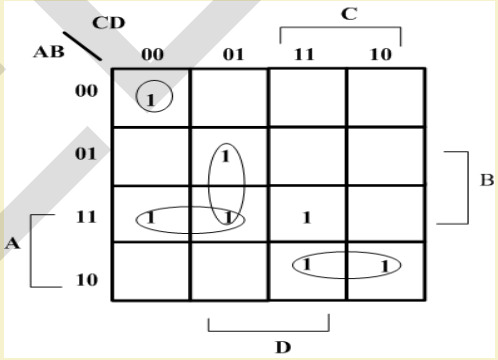


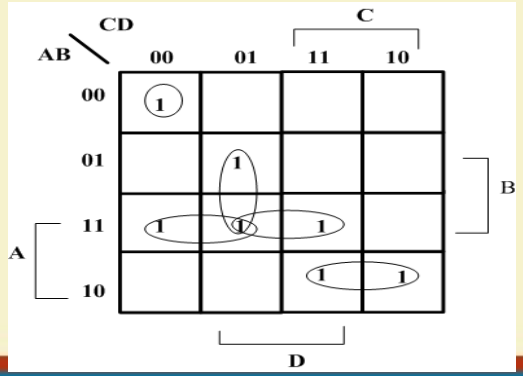
- Having Prime Implicants



continue

- Include those that are essential
- Leaves just a single 1 uncovered
- Have the 2 choices to use for covering it, both of equal size (i.e. same # of literals)
- Choose either
- Choose as shown getting
- F = A'B'C'D' +BC'D+ABC' + AB'C + ABD



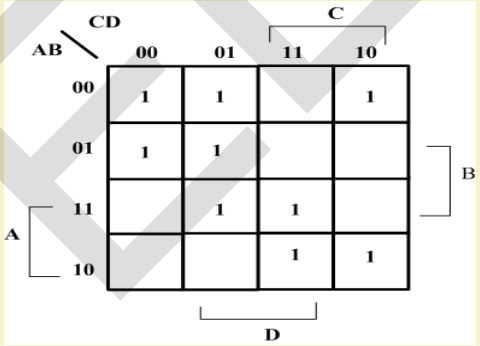


Selecting non-essential

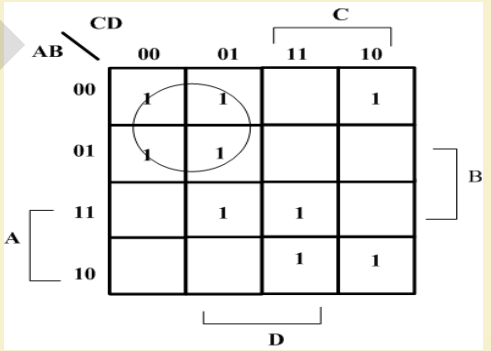
- Selection Rule: Minimize the overlap among prime implicants as much as possible.
- Make sure each prime implicant selected includes at least one minterm not included in any other prime implicant selected.

Another problem

- Problem:

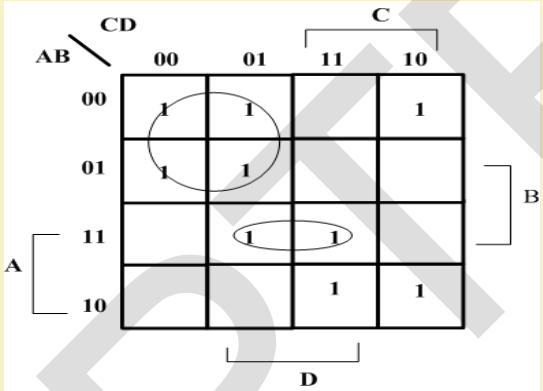


- Cover largest group of 1s

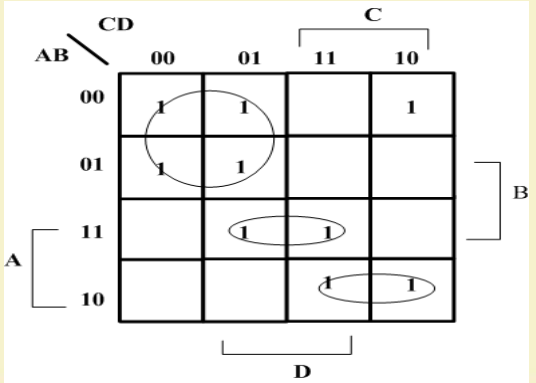


Select 1s not covered

- First

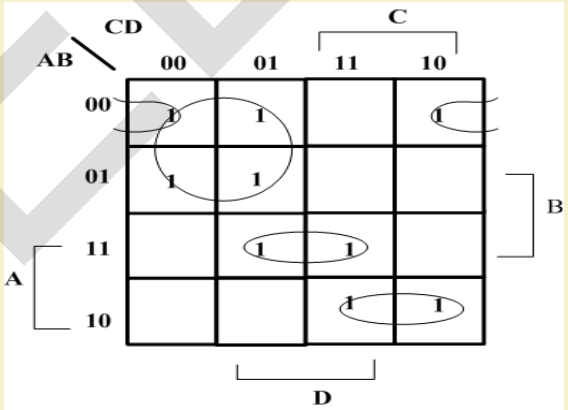


- Second



Just 1 more to go

- have a choice
  - Both are equal
  - Choose 1
- Get
- F = A’C’+ABD+AB’C + A’B’D’
- Equal in cost to
- F = A’C’+ABD+AB’C + B’CD’

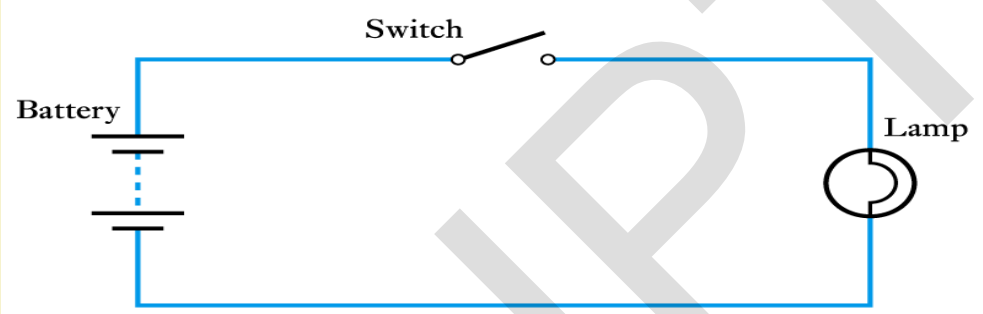


Logic Gates

Santanu Chattopadhyay
Electronics and Electrical Communication Engineering

Binary Quantities and Variables

- A **binary quantity** is one that can take only 2 states



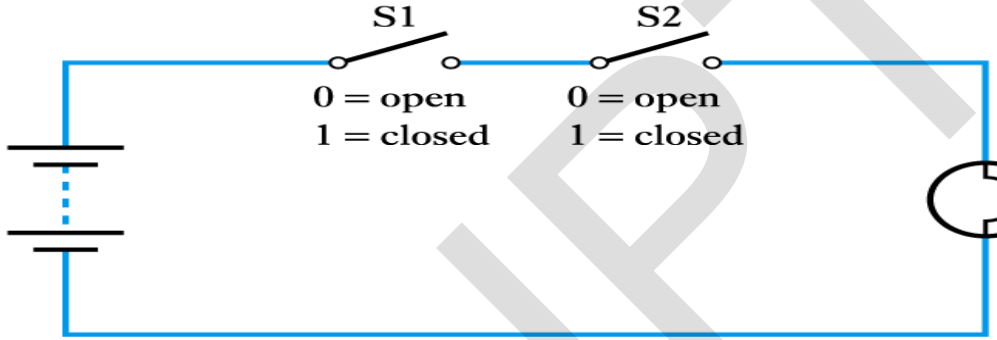
A simple binary arrangement

S	L
OPEN	OFF
CLOSED	ON

S	L
0	0
1	1

A truth table

- A binary arrangement with two switches in series



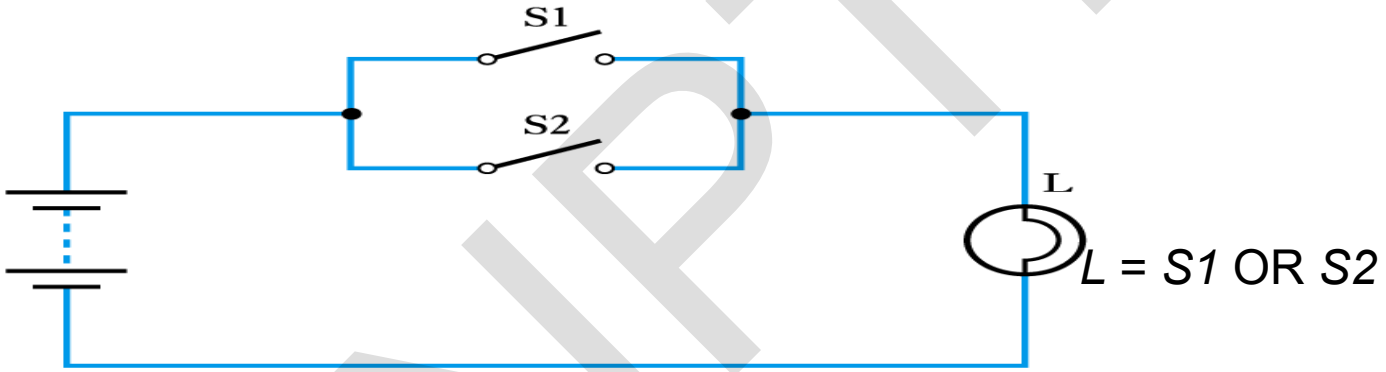
(a) Circuit

S1	S2	L
0	0	0
0	1	0
1	0	0
1	1	1

L = *S1* AND *S2*

(b) Truth table

- A binary arrangement with two switches in parallel

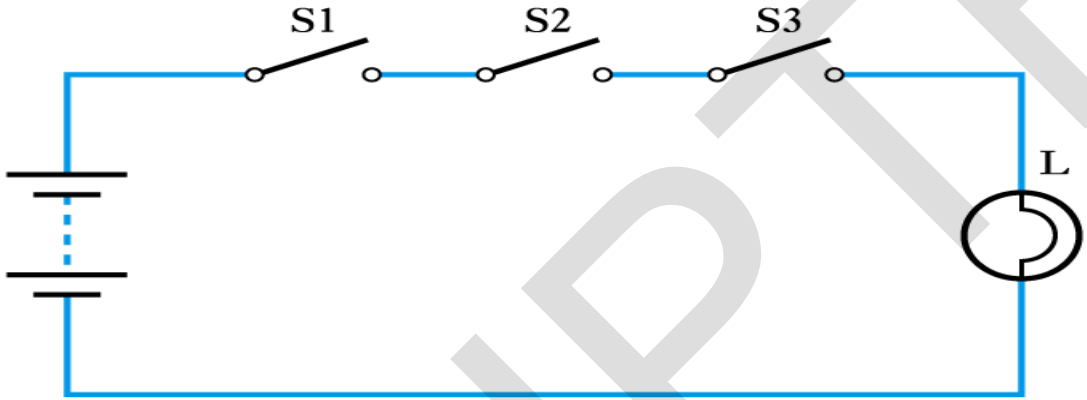


(a) Circuit

S1	S2	L
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table

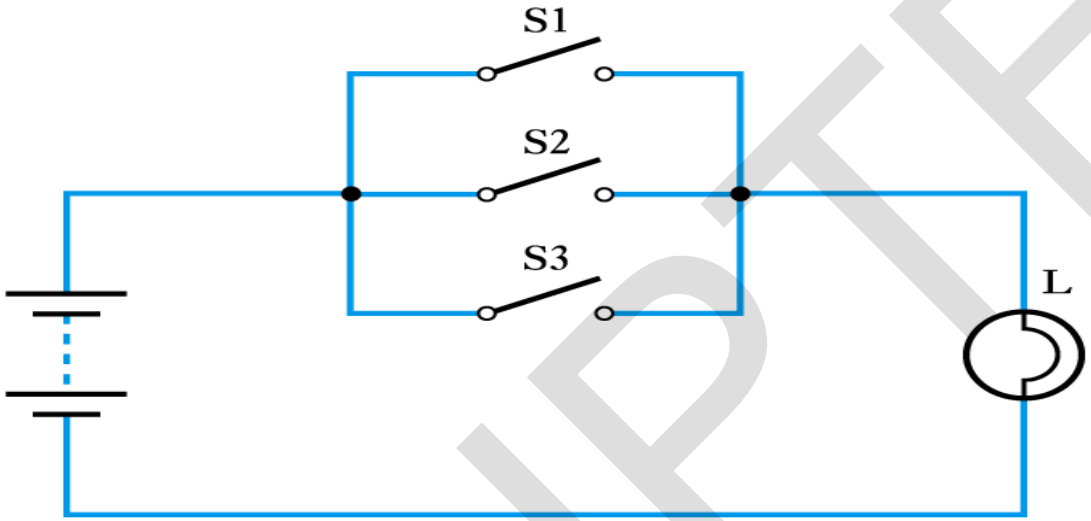
- Three switches in series



S1	S2	S3	L
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

L = *S1* AND *S2* AND *S3*

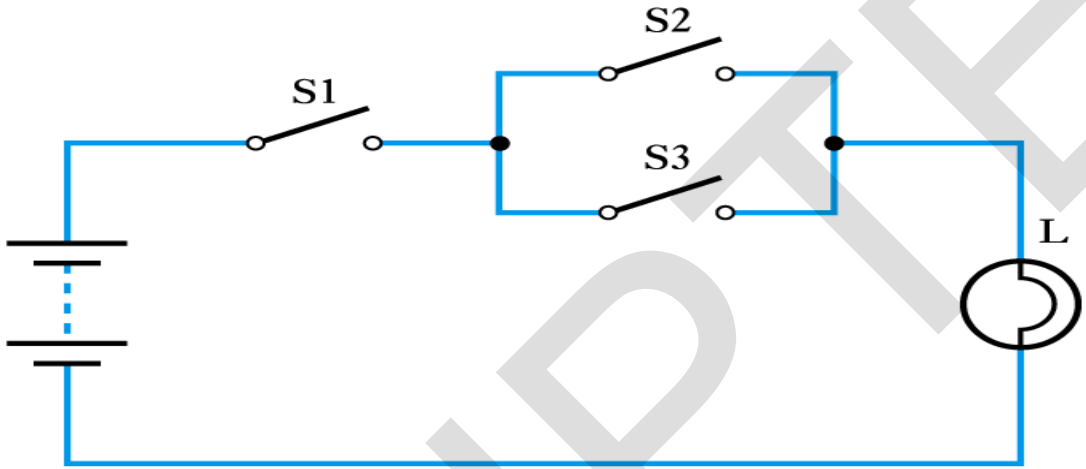
- Three switches in parallel



S1	S2	S3	L
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

L = *S1* OR *S2* OR *S3*

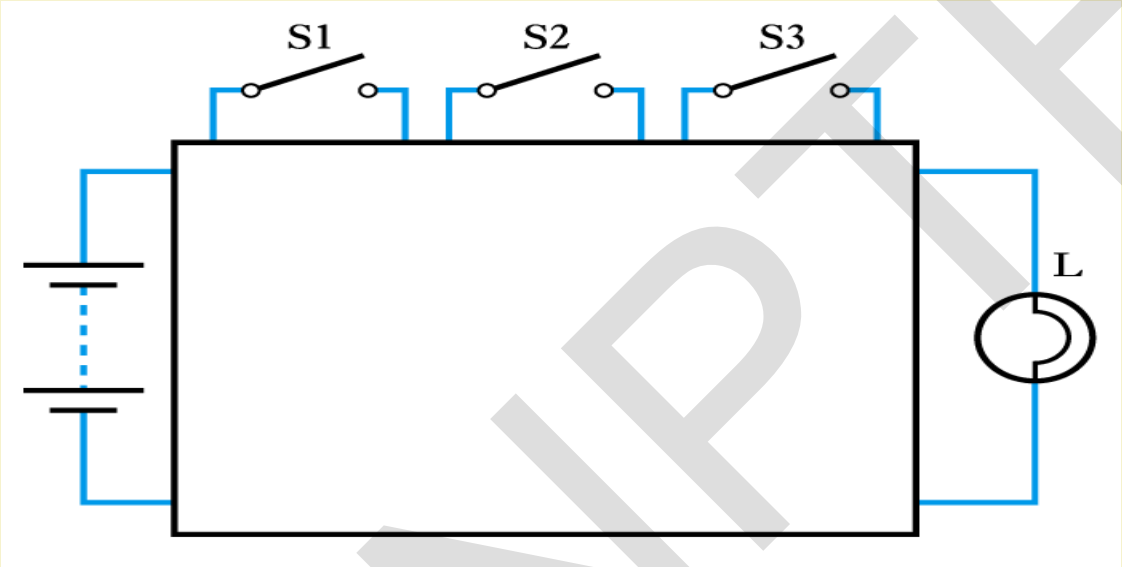
- A series/parallel arrangement



S1	S2	S3	L
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

L = *S1* AND (*S2* OR *S3*)

- Representing an unknown network



Logic Gates

- The building blocks used to create digital circuits are **logic gates**
- There are three elementary logic gates and a range of other simple gates
- Each gate has its own **logic symbol** which allows complex functions to be represented by a logic diagram
- The function of each gate can be represented by a **truth table** or using **Boolean notation**

- **The AND gate**

(a) Circuit symbol

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

(b) Truth table

$$C = A \cdot B$$

(c) Boolean expression

- **The OR gate**

(a) Circuit symbol

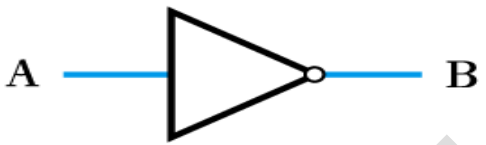
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table

$C = A + B$

(c) Boolean expression

- **The NOT gate (or inverter)**



(a) Circuit symbol

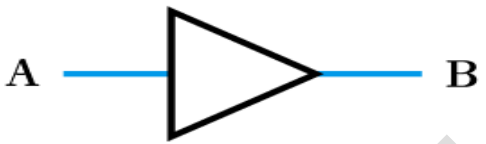
A	B
0	1
1	0

(b) Truth table

$$B = \overline{A}$$

(c) Boolean expression

- **A logic buffer gate**



(a) Circuit symbol

A	B
0	0
1	1

(b) Truth table

$B = A$

(c) Boolean expression

- **The NAND gate**

(a) Circuit symbol

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

(b) Truth table

$$C = \overline{A \cdot B}$$

(c) Boolean expression

- **The NOR gate**

(a) Circuit symbol

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

(b) Truth table

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

(c) Boolean expression

- **The Exclusive OR gate**

(a) Circuit symbol

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

(b) Truth table

$$C = A \oplus B$$

(c) Boolean expression

- **The Exclusive NOR gate**

(a) Circuit symbol

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

(b) Truth table

$$C = \overline{A \oplus B}$$

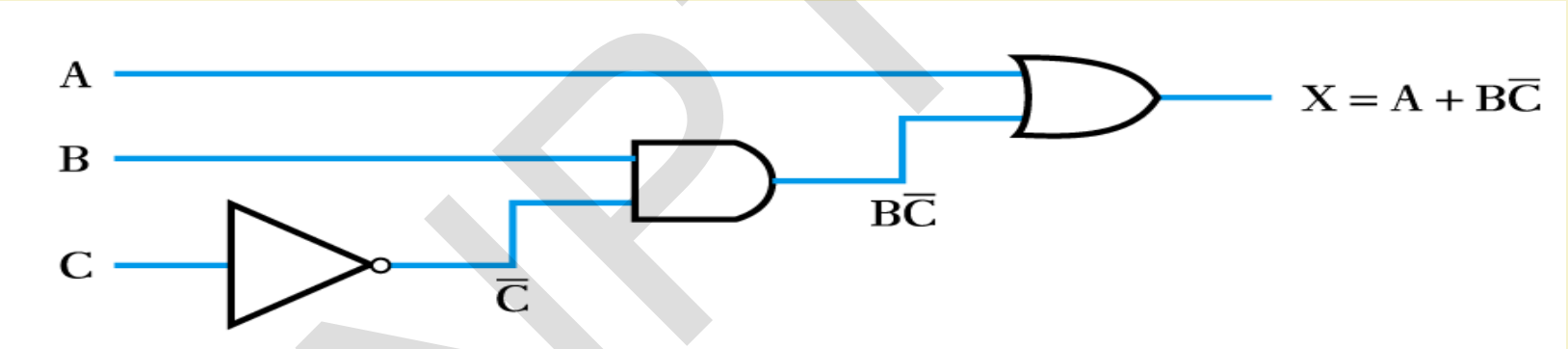
(c) Boolean expression

Combinational Logic

- Digital systems may be divided into two broad categories:
  - **combinational logic**
    - where the outputs are determined solely by the current states of the inputs
  - **sequential logic**
    - where the outputs are determined not only by the current inputs but also by the sequence of inputs that led to the current state

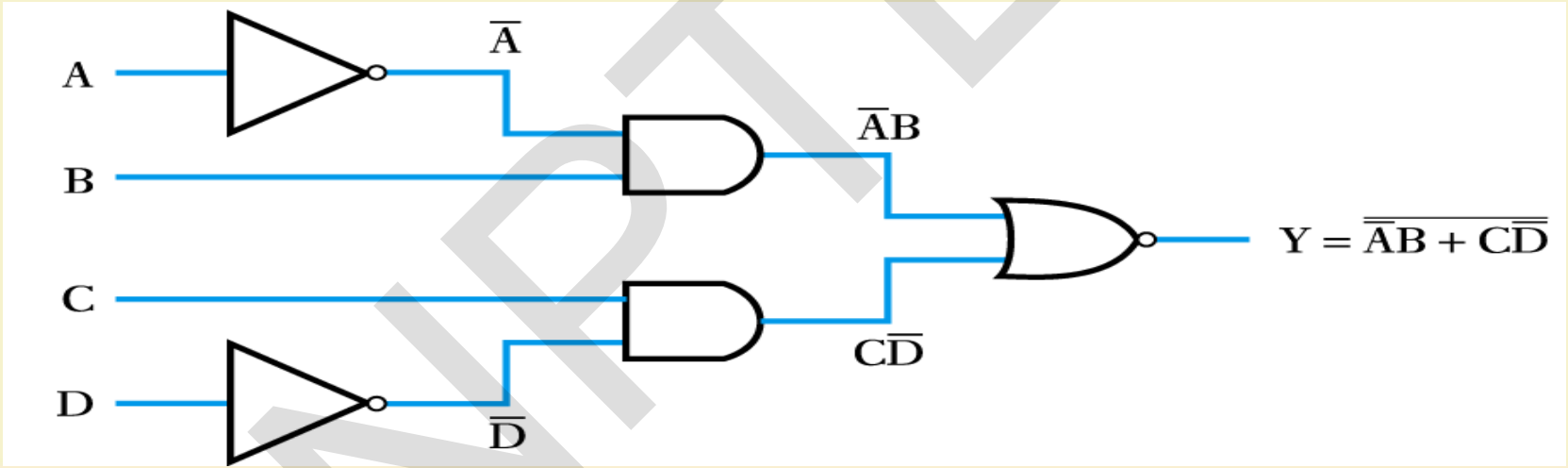
- **Implementing a function from a Boolean expression**

  Implement the function $X = A + B\overline{C}$

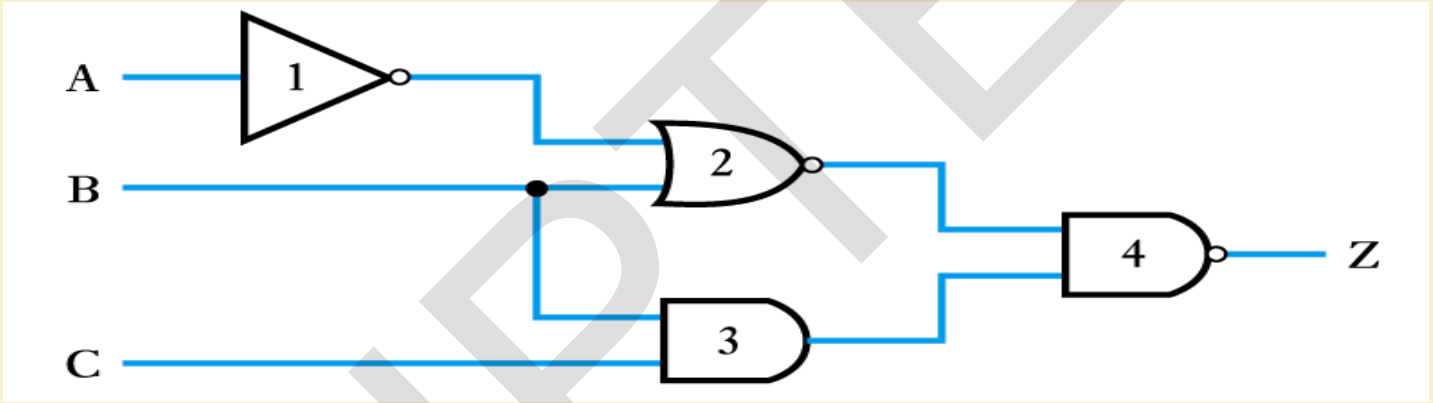


- **Implementing a function from a Boolean expression**

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

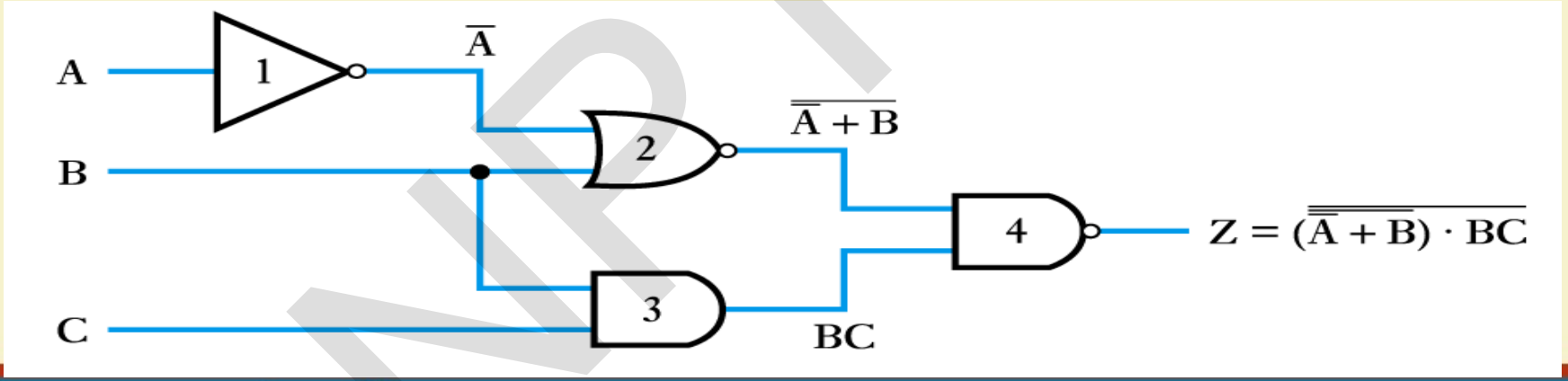


- **Generating a Boolean expression from a logic diagram**



Example (continued)

– work progressively from the inputs to the output adding logic expressions to the output of each gate in turn



- **Implementing a logic function from a description**

The operation of the Exclusive OR gate can be stated as:

"The output should be true if either of its inputs are true, but not if both inputs are true."

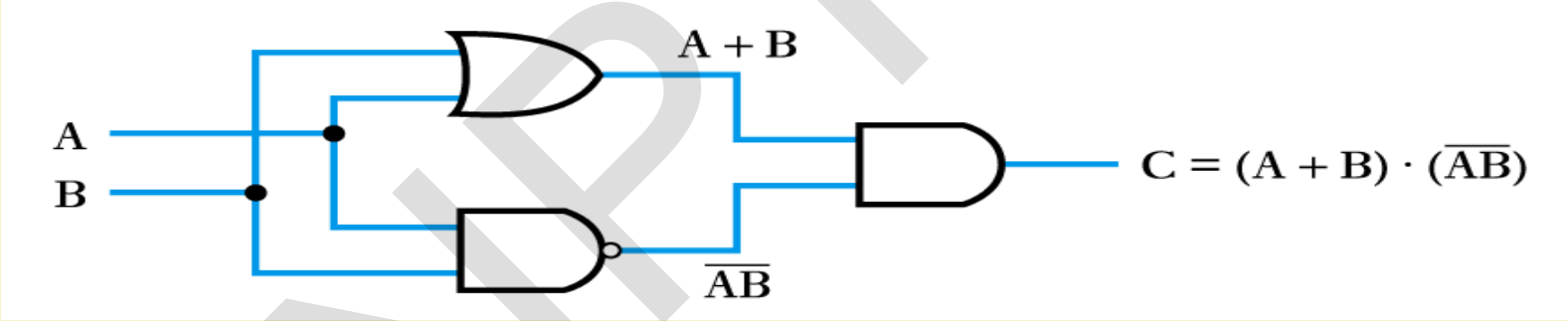
This can be rephrased as:

"*The output is true if A* OR *B is true*, AND *if A* AND *B are* NOT *true*."

We can write this in Boolean notation as $X = (A + B) \bullet \overline{(AB)}$

Example (continued)

The logic function $X = (A + B) \bullet \overline{(AB)}$



- **Implementing a logic function from a truth table**

Implement the function of the following truth table

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

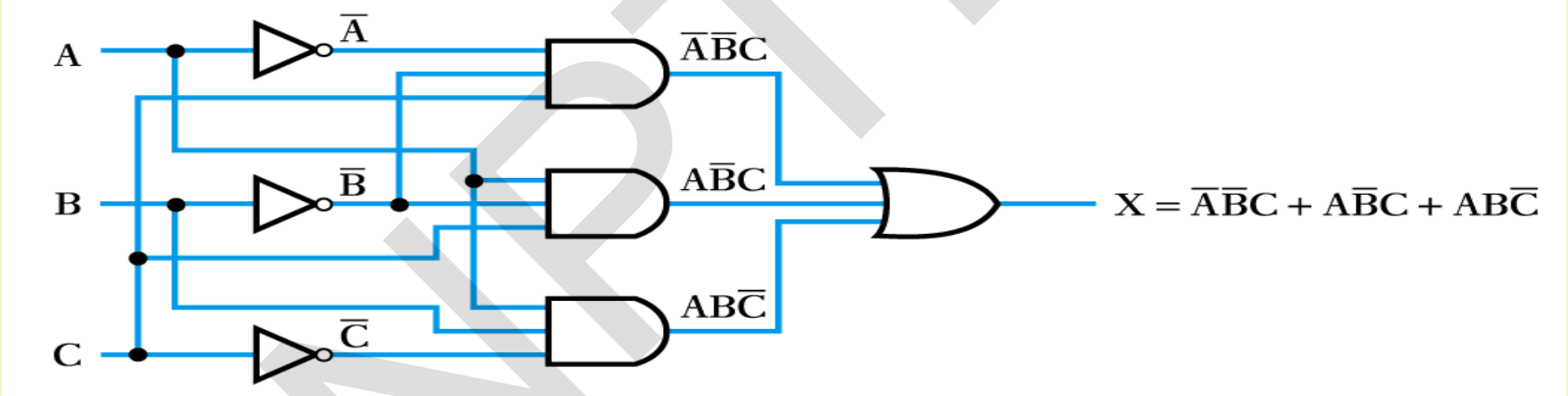
- first write down a Boolean expression for the output
- then implement as before
- in this case

$$X = \bar{A}\,\bar{B}C + A\bar{B}C + AB\bar{C}$$

Example (continued)

The logic function $X = \overline{A}\,\overline{B}C + A\overline{B}C + AB\overline{C}$

can then be implemented as before

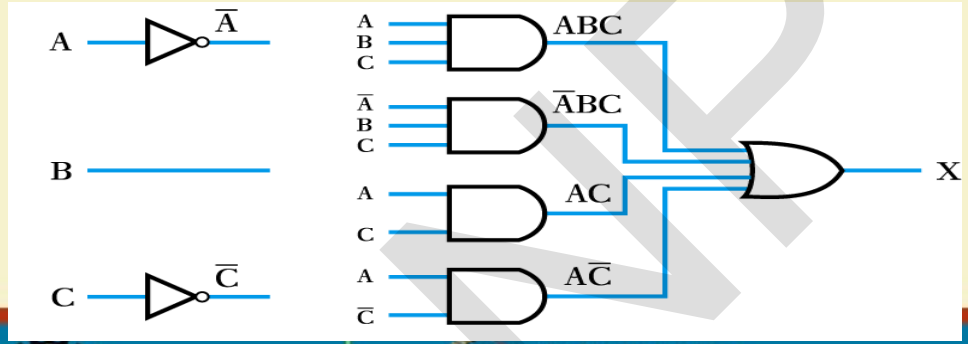


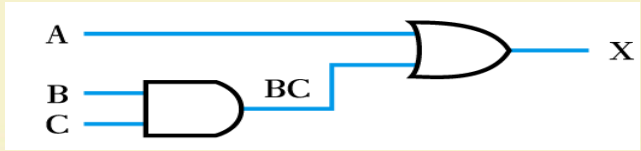
- In some cases it is possible to *simplify* logic expressions using the rules of Boolean algebra

$$X = ABC + \overline{A}BC + AC + A\overline{C}$$

can be simplified to $X = BC + A$

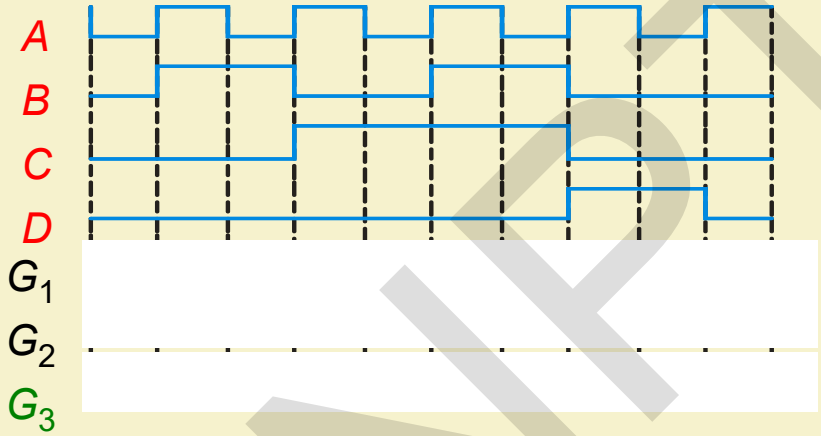
hence the following circuits are equivalent

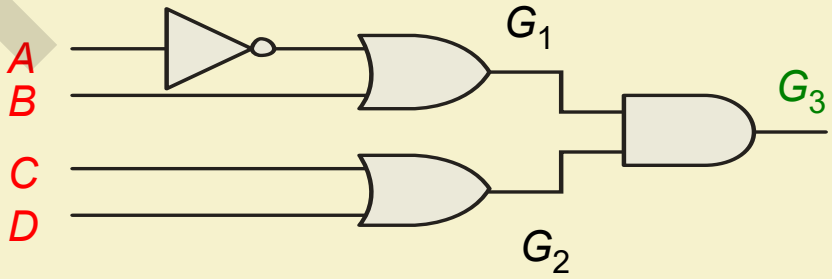




Pulsed Waveforms

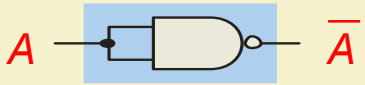
For combinational circuits with pulsed inputs, the output can be predicted by developing intermediate outputs and combining the result. For example, the circuit shown can be analyzed at the outputs of the OR gates:



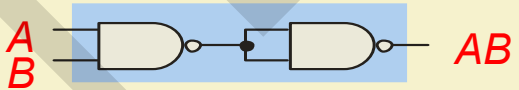


Universal Gates

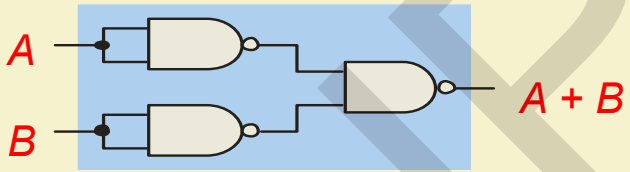
NAND gates are sometimes called **universal** gates because they can be used to produce the other basic Boolean functions.



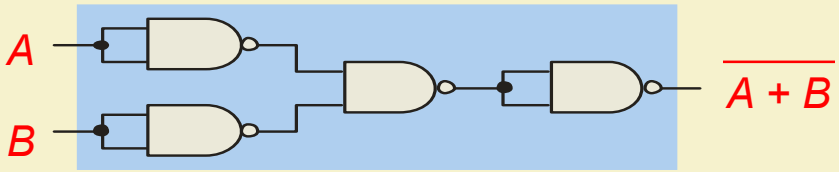
Inverter



AND gate



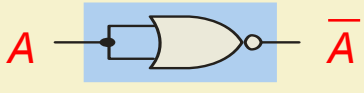
OR gate



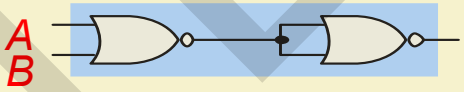
NOR gate

Universal Gates

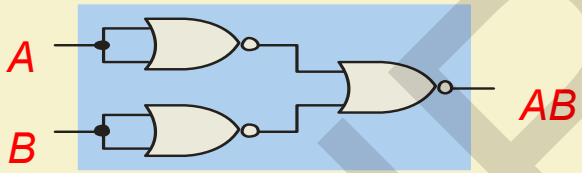
NOR gates are also **universal** gates and can form all of the basic gates.



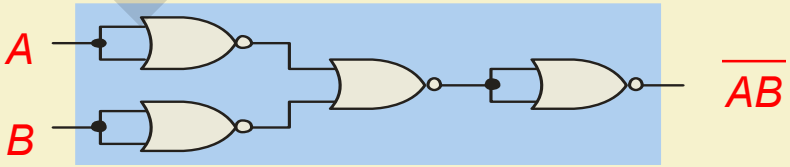
Inverter



OR gate



AND gate



NAND gate

Other Universal Possibilities ...

- {AND, OR, INVERT}
- {XOR, AND}
- {XOR, OR}
- ...