- **Using Karnaugh maps**
- **Second Canonical form**
- **Conversion to NAND or to NOR gate representation**

Incompletely specified functions - ‘Don’t care’ terms

•Sometimes, the output for certain inputs will not matter (e.g. certain input combinations may never occur).

•Such outputs are known as ‘**don’t care**’ terms and can be made either 0 or 1 depending which aids the minimisation process.

Truth table

D	H	L	F
0	0	0	1
0	0	1	0
0	1	0	×
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	×
1	1	1	0

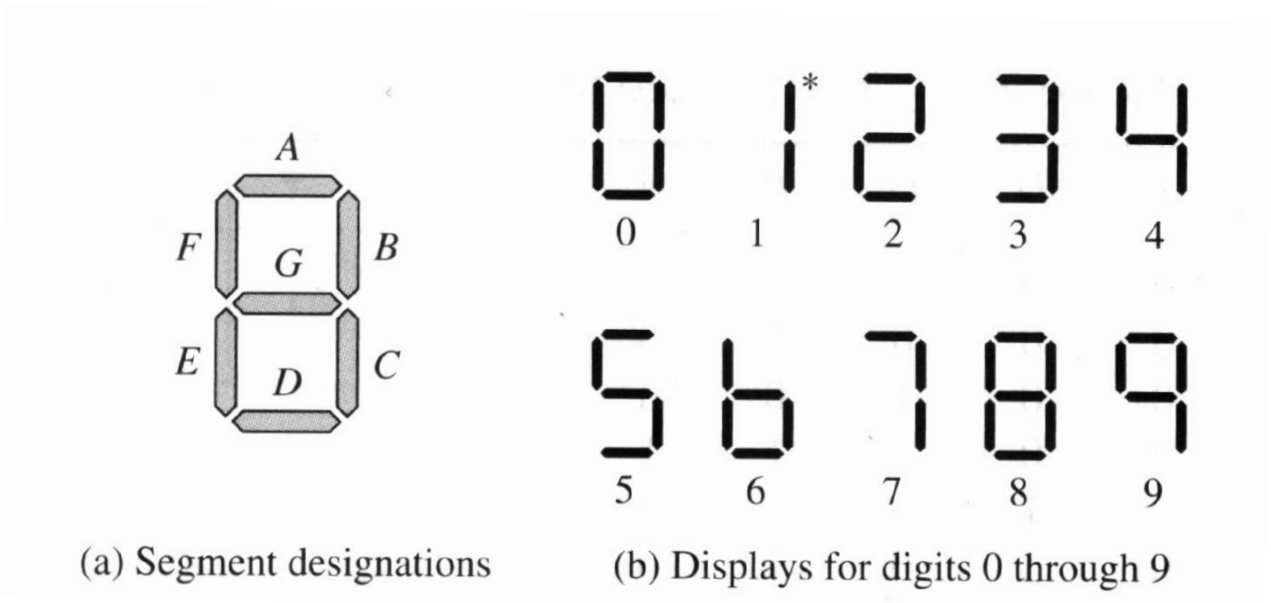
These input combinations do not occur in practice

Don’t care terms

Incompletely specified functions - ‘Don’t care’ terms

Example: BCD decoder for seven-segment display.

In four-bit BCD there are only 10 out of 16 combinations which are useful for representing decimal digits. The outputs of the six higher combinations should be treated as ‘don’t care’ ones.



(a) Segment designations (b) Displays for digits 0 through 9

Incompletely specified functions - 'Don't care' terms

Example: BCD decoder for seven-segment display.

Decimal	Binary				
	W	X	Y	Z	
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	
8	1	0	0	0	
9	1	0	0	1	
	1	0	1	0	Don't care .....
	1	0	1	1	Don't care .....
	1	1	0	0	Don't care .....
	1	1	0	1	Don't care .....
	1	1	1	0	Don't care .....
	1	1	1	1	Don't care .....

These combinations are never used

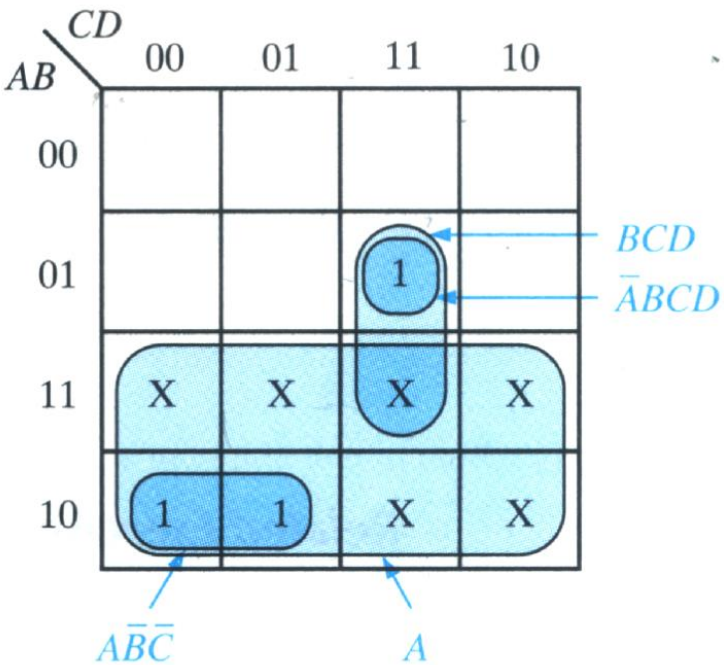
Using ‘Don’t care’ terms

Truth table

Inputs $ABCD$	Output Y
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	0
0 1 0 0	0
0 1 0 1	0
0 1 1 0	0
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

Don’t cares

- **One can choose the ‘don’t care’ term to be ‘0’ or ‘1’, depending what would help to minimise the expression (to wrap more terms in Karnaugh map).**
- **No need to use all the ‘don’t care’ terms.**



Without “don’t cares” $Y = \bar{A}\bar{B}C + \bar{A}BCD$

With “don’t cares” $Y = A + BCD$

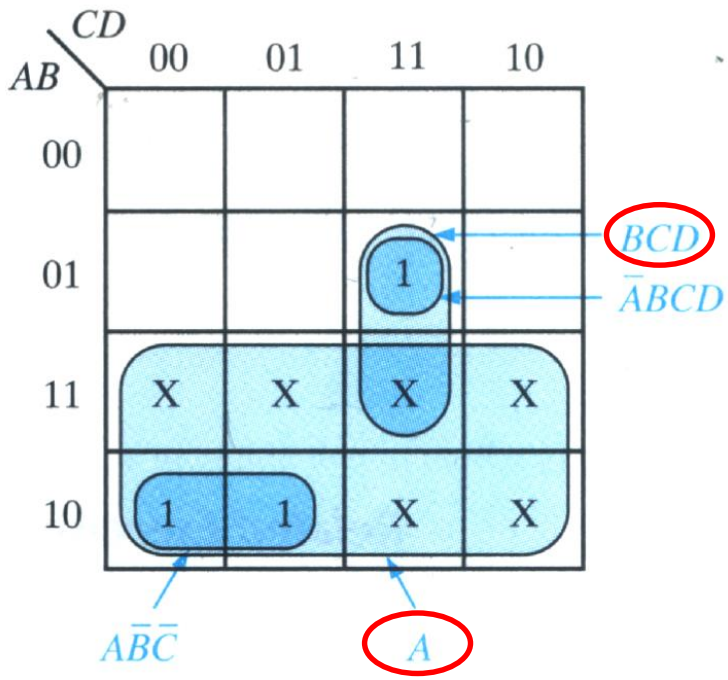
Using ‘Don’t care’ terms

Truth table

Inputs $ABCD$	Output Y
0 0 0 0	0
0 0 0 1	0
0 0 1 0	0
0 0 1 1	0
0 1 0 0	0
0 1 0 1	0
0 1 1 0	0
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

Don’t cares

One can choose the don’t care term to be ‘0’ or ‘1’, depending what would help to minimise the expression (to wrap more terms in Karnaugh map)



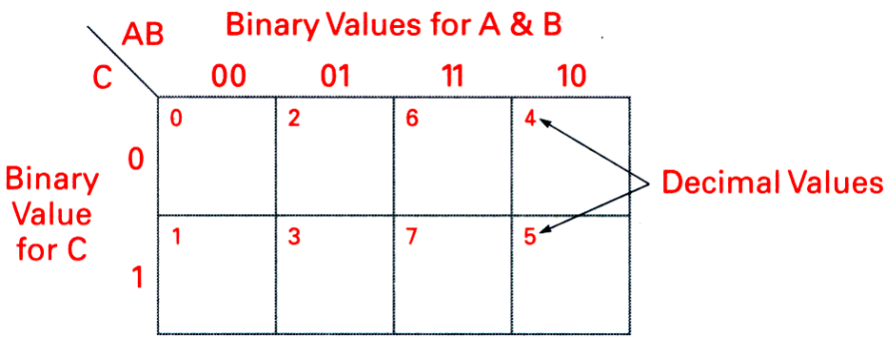
Without “don’t cares” $Y = \bar{A}\bar{B}C + \bar{A}BCD$

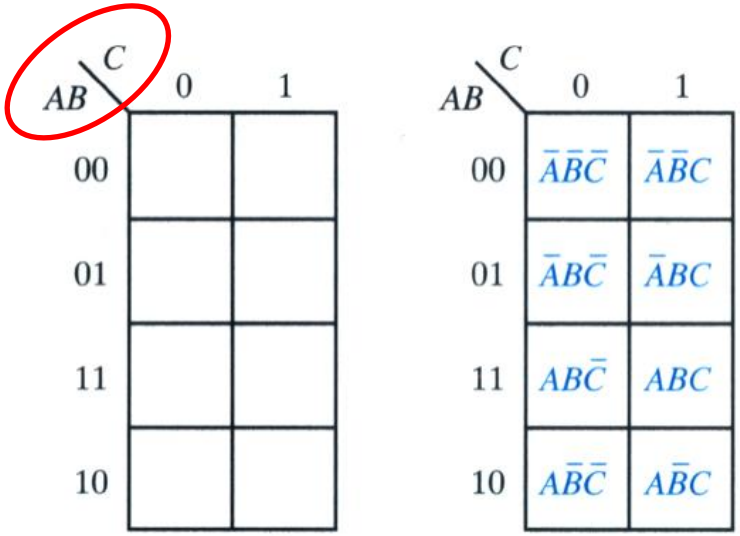
With “don’t cares” $Y = A + BCD$

Using Karnaugh maps

Karnaugh maps can be transposed.

But the order of rows/columns must be preserved (only one variable changes between the adjacent terms).





Using Karnaugh maps

Karnaugh maps can be transposed

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

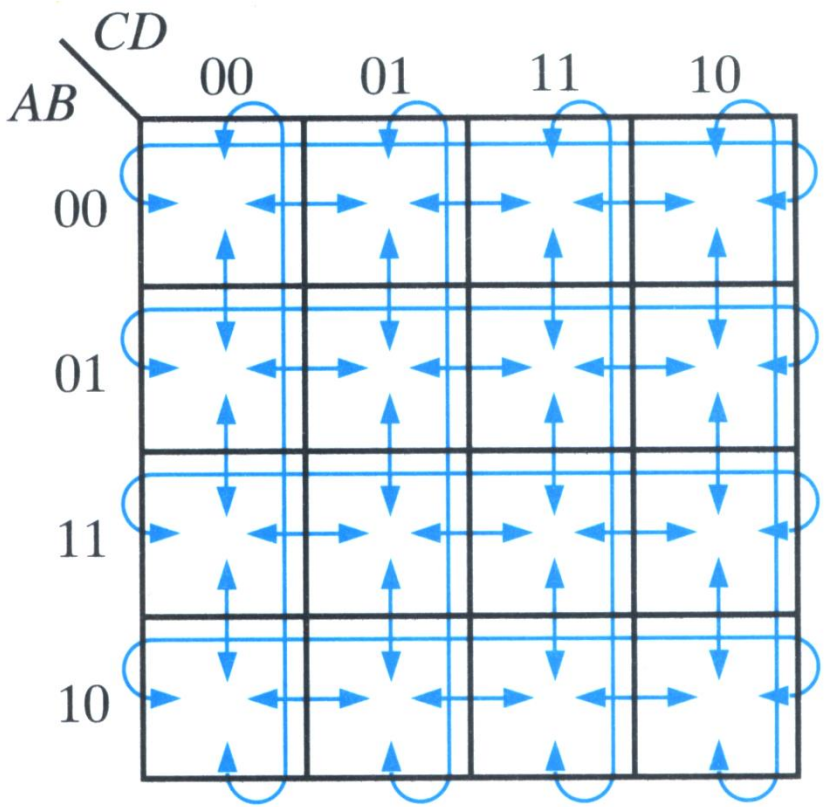
Decimal Values

AB \ CD	00	01	11	10
00				
01				
11				
10				

AB \ CD	00	01	11	10
00	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}CD$	$\bar{A}\bar{B}C\bar{D}$
01	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}B\bar{C}D$	$\bar{A}BCD$	$\bar{A}BC\bar{D}$
11	$AB\bar{C}\bar{D}$	$AB\bar{C}D$	$ABCD$	$ABC\bar{D}$
10	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}D$	$A\bar{B}CD$	$A\bar{B}C\bar{D}$

Using Karnaugh maps: keep the right sequence of rows/columns

Adjacencies in the Karnaugh map: only one bit is changing between the neighbouring cells



Using Karnaugh maps

Looping cells: aim to wrap together as much cells as possible

(a)

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

(b)

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

(c)

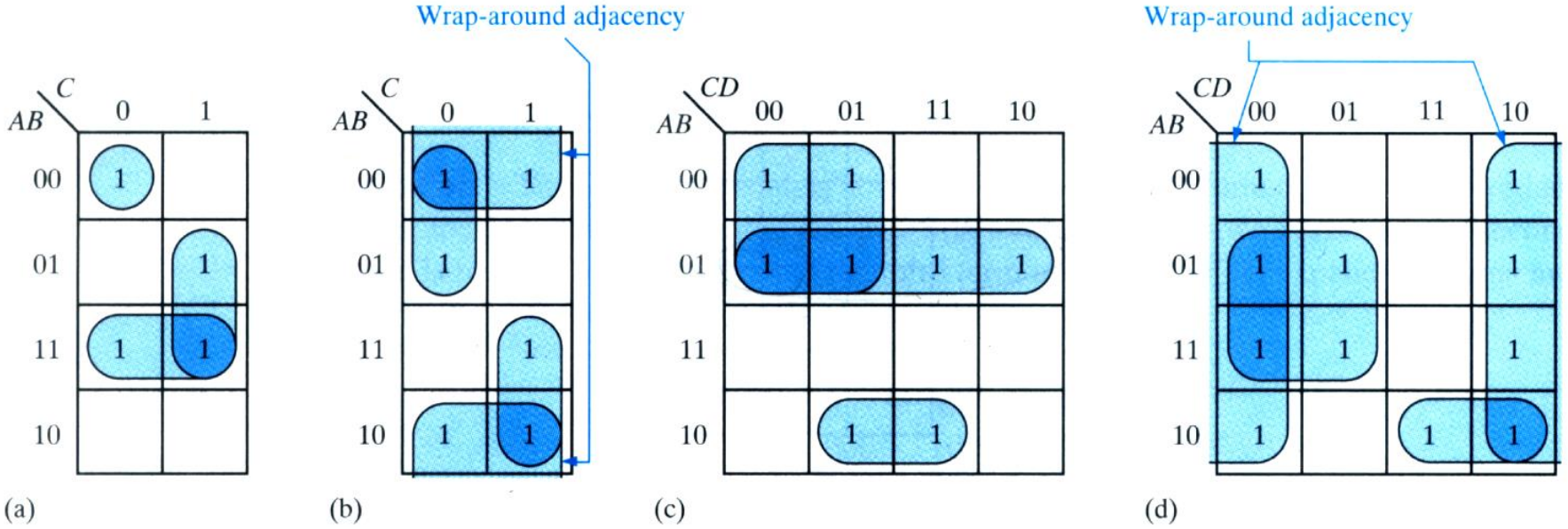
AB \ CD	00	01	11	10
00	1	1		
01	1	1	1	1
11				
10		1	1	

(d)

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

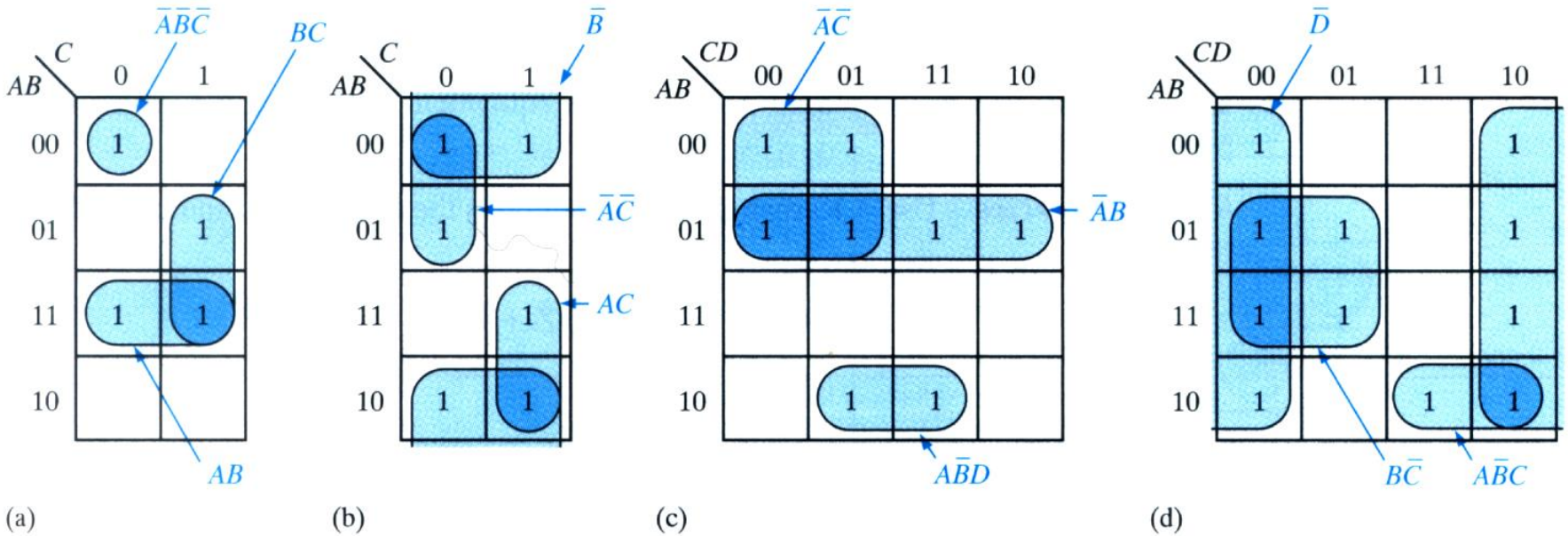
Using Karnaugh maps

Note in which cases you need to use the wrap-arounds



Using Karnaugh maps

- **The changing variable disappears: explore both horizontal and vertical directions where appropriate**
- **Consider each term only once**



Second canonical form: product of sum

We will use the same truth table as for the conveyor example for 1st canonical form.
But this time we will look at 'zero' outputs.

a	*b*	*m*	*s*	*M*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

maxterms

a + b + m + s
a + b + m + s'
a + b + m' + s
a + b + m' + s'
a + b' + m + s
a + b' + m + s'
a + b' + m' + s

a' + b + m + s
a' + b + m + s'
a' + b + m' + s

a' + b' + m + s
a' + b' + m + s'
a' + b' + m' + s

product of sums

M = (a+b+m+s)(a+b+m+s')...

... (a'+b'+m'+s)

Second canonical form: product of sum

M = (a+b+m+s)(a+b+m+s') ... (a'+b'+m'+s) =

Code the maxterms with binaries: **0000 0001 ... 1110**

Replace the binaries with decimal equivalents: **0 1 ... 14**

Finally arrive with short form of logic expression :

M = Π(0,1,2,3,4,5,6,8,9,10,12,13,14)

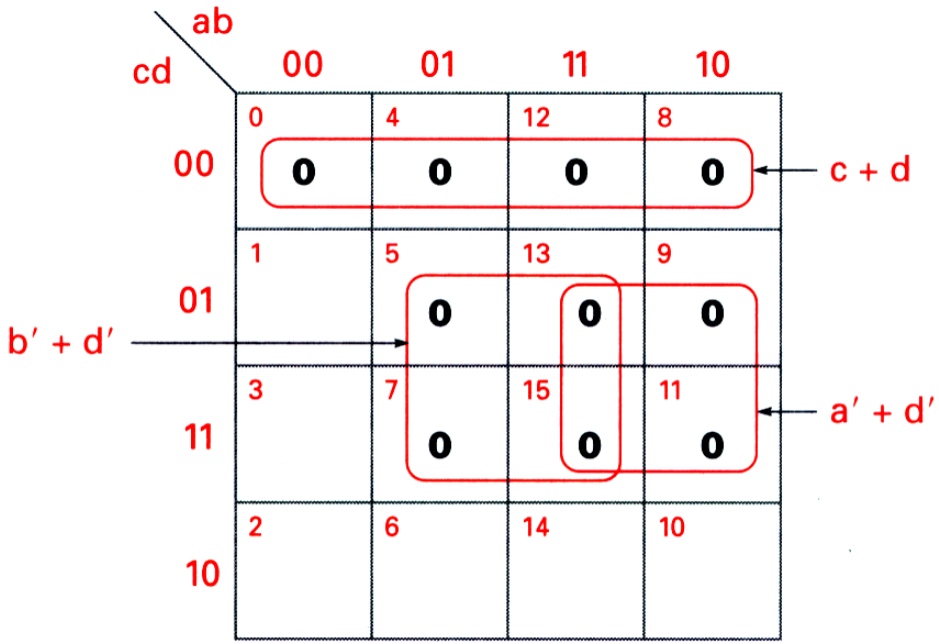
Second canonical form or product of sum (POS)

Simplifying maxterm expressions

Loading and grouping maxterms is exactly the same as loading and grouping minterms, except that

- **'0's are loaded instead of '1's**
- **terms are sums not products**

$$G = f(a, b, c, d) = \Pi(0, 4, 5, 7, 8, 9, 11, 12, 13, 15)$$



$$G = (c + d)(a' + d')(b' + d')$$

***Exercises*:**

1. A logic circuit has inputs A, B, C and D. The output of circuit is given by

$E = \Sigma (1, 3, 4, 5, 7, 10, 12, 13)$

Find the minimum product of sum (2nd canonical) for E.

Answer: E = (A'+B'+C')(A'+B+D')(A+C'+D)(B+C+D) or E = (A'+B+C)(A+B+D)(A'+C'+D')(B'+C'+D)

2. Construct the Karnaugh maps and find the minimum POS and SOP expressions for each of the following logic functions:

(a) Z = W'X'Y' + W'XY' + WX'Y + WXY

(b) D = A'B'C' + AB'C' + A'B'C + AB'C + A'BC

(c) E = A'BCD' + ABC'D' + ABC'D + ABCD'

Answers:

SOPs: (a) Z=W'Y'+WY; (b) D=B'+A'C; (c) E=ABC'+BCD'.

POSs: (a) (W+Y')(W'+Y); (b) (A'+B')(B'+C); (c) E=B(A+C)(C'+D').

Conversion between canonical forms

Conversion can be easily done by using the reference set I:

$$\mathbf{I} = (0, 1, 2, 3, 4, 5, 6, 7)$$

If the minterm set (related to 1-st canonical form or SOP) is **m**, and the maxterm (related to 2-nd canonical form or POS) is **M** then:

$$\mathbf{M = I - m}$$

Example: $\mathbf{F} = \sum\overbrace{(0, 5, 6, 7)}^{\mathbf{m}}$

$\mathbf{M = I - m =}$ (0, 1, 2, 3, 4, 5, 6, 7) - (0, 5, 6, 7) = (1, 2, 3, 4)

hence $\mathbf{F} = \boldsymbol{\pi}\underbrace{(1, 2, 3, 4)}_{\mathbf{M}}$

We have seen that the same logical task (conveyor example) can be formulated either as 1st or as 2nd canonical form.

Why different canonical forms?

(a) To chose the simplest circuit design – **minimal number of gates**.

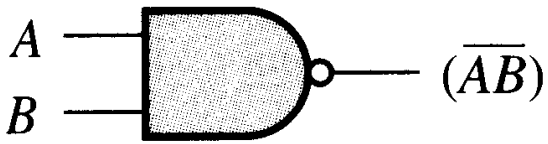
(b) Engineer often have a task to optimise a logic circuit to work not only with minimal gates number but also for **particular type of gate.**

Conversion to one type of gates

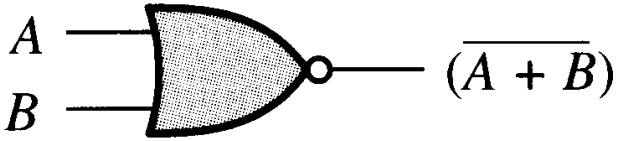
One of the important implications of De Morgan's theorems is that any logic function can be implemented solely with **NAND gates** (AND + inverter)

or

with **NOR gates** (OR + inverter).

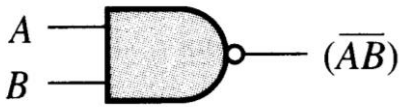


(a) NAND gate



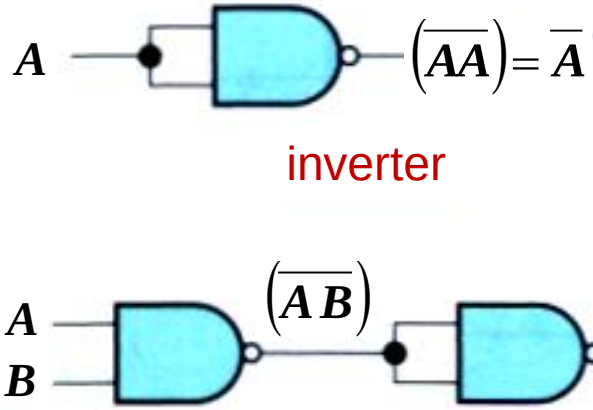
(b) NOR gate

Conversion to one type of gates

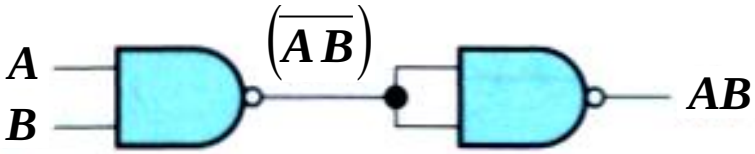


NAND gate

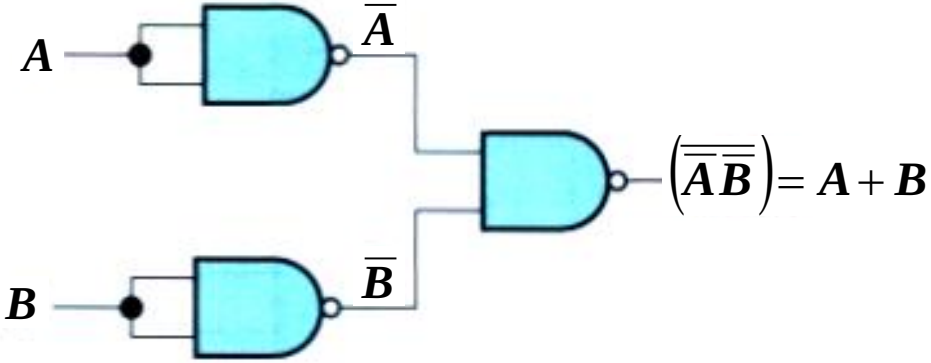
Any logic operation can be represented as a combination of NAND gates:



inverter

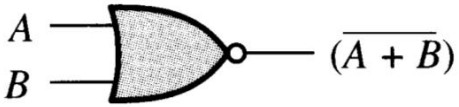


AND gate



OR gate

Conversion to one type of gates



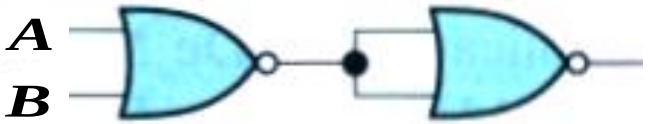
NOR gate

Any logic operation can be represented as a combination of NOR gates:

inverter

$$\left(\overline{A+A}\right)=\overline{A}$$

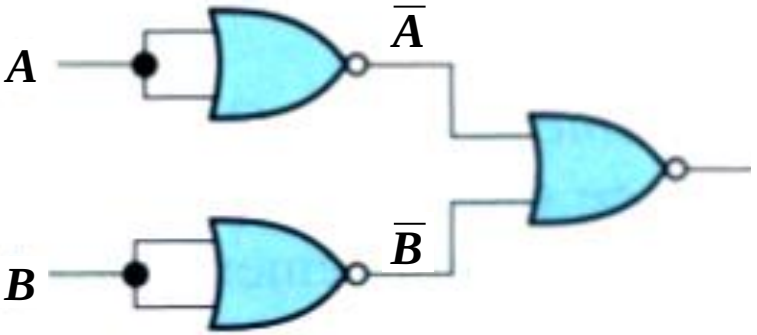
OR gate



$$\overline{A+B}$$

$$\left(\overline{\overline{A+B}+\overline{A+B}}\right)=$$

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

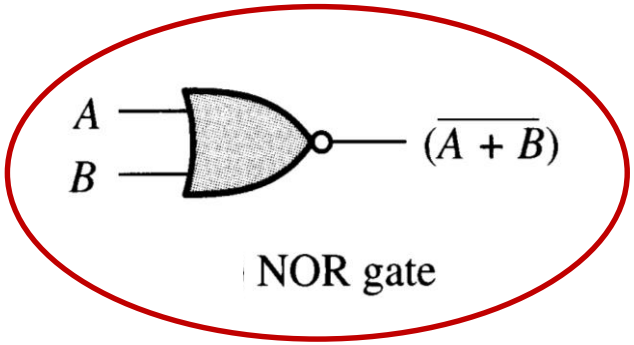


$$\left(\overline{\overline{A}+\overline{B}}\right)=AB$$

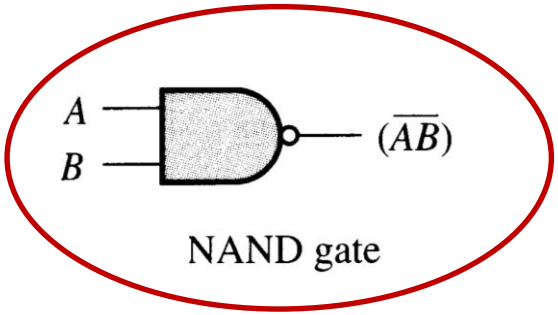
AND gate

Conversion to one type of gates

How is this done in practice and which type of the gate is preferable?



or



?

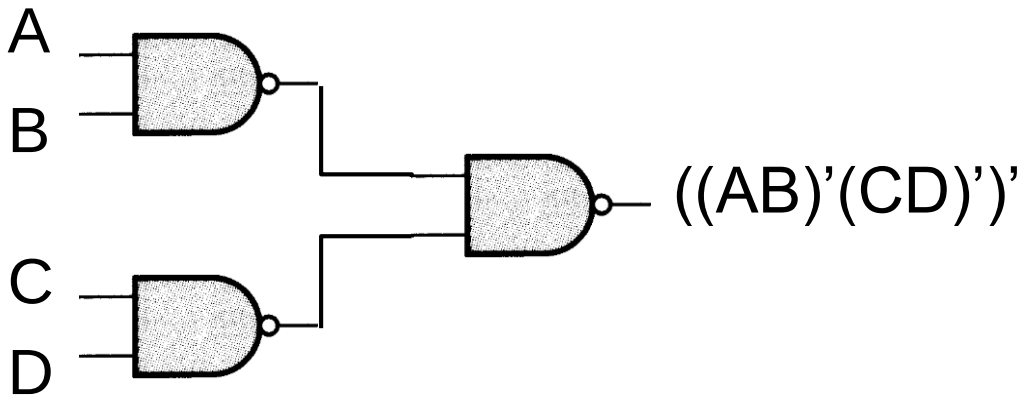
Conversion to one type of gates: SOP

First canonical form or sum of products (**SOP**) can be conveniently represented with **NAND** gates only:

$$F = AB + CD$$

Applying De Morgan's theorem: $F' = (AB)'(CD)'$ Applying De Morgan's theorem again:

$$F'' = F = ((AB)'(CD)')'$$



Conversion to one type of gates: POS

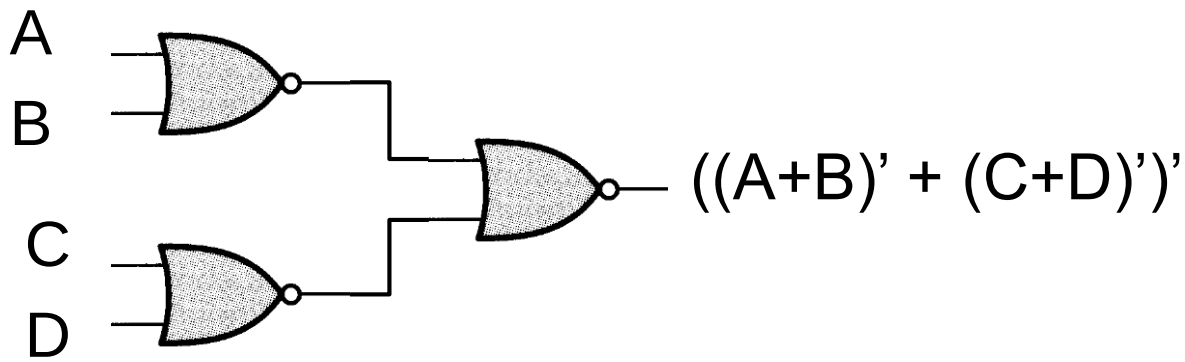
Second canonical form or product of sum (**POS**) can be conveniently represented with **NOR** gates only:

$$F = (A+B)(C+D)$$

Applying De Morgan's theorem: $F' = (A+B)' + (C+D)'$

Applying De Morgan's theorem again:

$$F'' = F = ((A+B)' + (C+D)')'$$



Self- assessment

- What characteristics define a combinational logic circuit?
- What are product terms and sum terms?
- What are two forms that can be used to describe the operation of a combinational logic circuit?
- How does the number of columns and rows in a truth table relate to the number of input and output variables?
- What is meant by a canonical form, and how are they related to the outputs from a truth table?
- What is the effect of minimising of a canonical form?
- Why can Karnaugh maps be used for ‘visually’ minimising Boolean expressions?