BME2322 – Logic Design

The Instructors:

Dr. Görkem SERBES (C317)

gserbes@yildiz.edu.tr

https://avesis.yildiz.edu.tr/gserbes/

Lab Assistants:

Nihat AKKAN

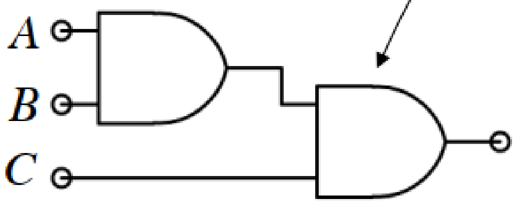
nakkan@yildiz.edu.tr

https://avesis.yildiz.edu.tr/nakkan

LECTURE 5

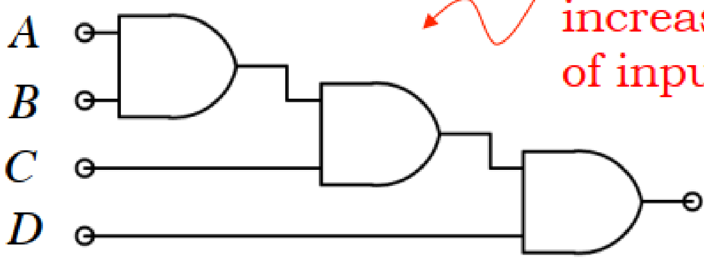
ANDs and ORs with > 2 Inputs

Replace 2-input AND gates with 2-input OR gates to create large fan-in OR gates.



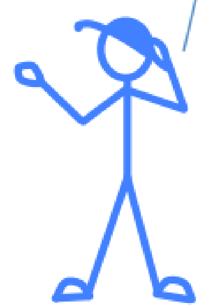
$Z = A \cdot B \cdot C = (A \cdot B) \cdot C$

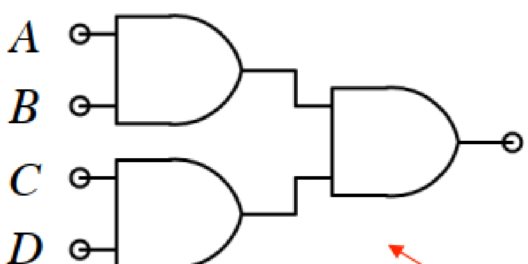
Chain: Propagation delay increases <u>linearly</u> with number of inputs



$Z = ((A \cdot B) \cdot C) \cdot D$

Which one should I use?



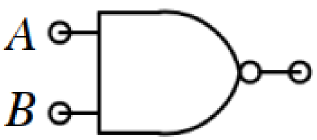


$Z = (A \cdot B) \cdot (C \cdot D)$

Tree: Propagation delay increases <u>logarithmically</u> with number of inputs

More Building Blocks

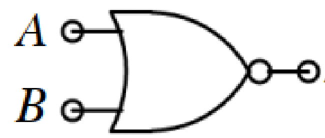
NAND (not AND)



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

A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

NOR (not OR)

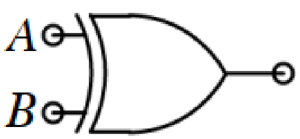


$Z = \overline{A + B}$

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	0

In a CMOS gate, rising inputs lead to falling outputs and vice-versa, so CMOS gates are naturally inverting. Want to use NANDs and NORs in CMOS designs… But NAND and NOR operations are not associative, so wide NAND and NOR gate can't use a chain or tree strategy. Stay tuned for more on this!

XOR (exclusive OR)



$Z = A \oplus B$

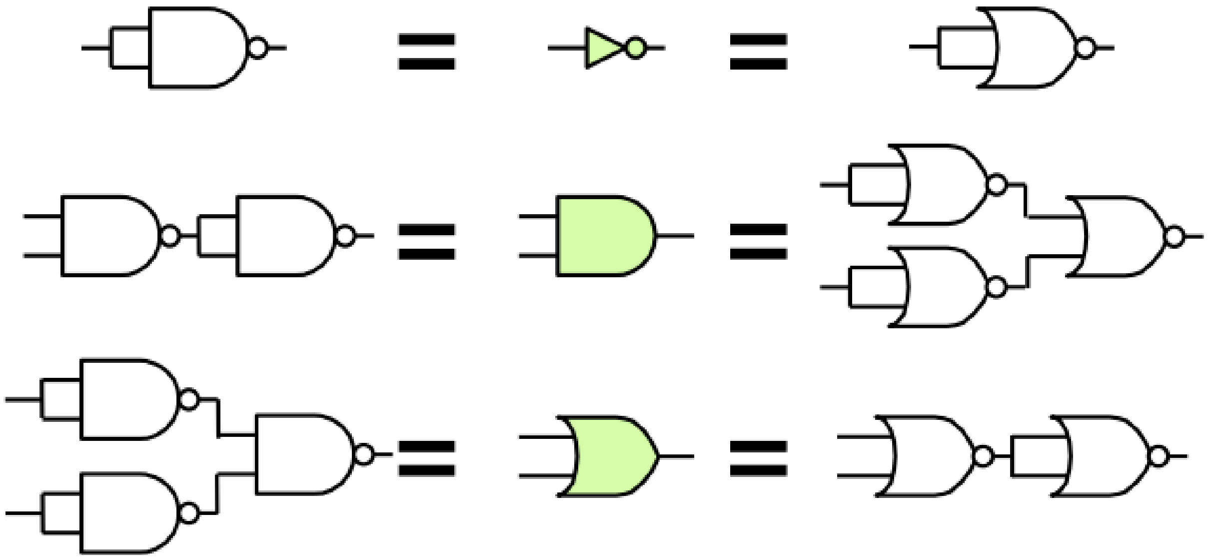
A	B	Z
0	0	0
0	1	1
1	0	1
1	1	0

XOR is very useful when implementing parity and arithmetic logic. Also used as a "programmable inverter": if A=0, Z=B; if A=1, Z=~B

Wide fan-in XORs can be created with chains or trees of 2-input XORs.

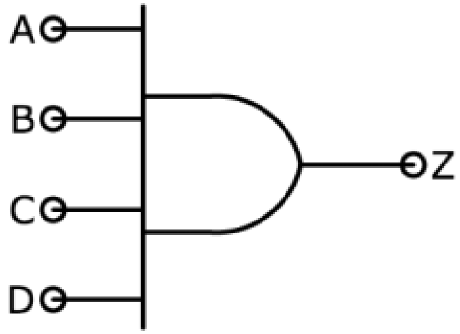
Universal Building Blocks

NANDs and NORs are <u>universal:</u>

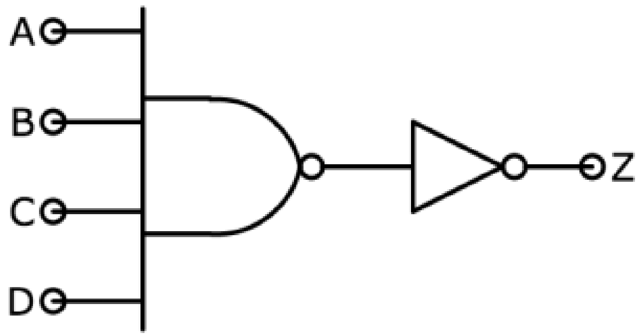


Any logic function can be implemented using only NANDs (or, equivalently, NORs). Good news for CMOS technologies!

Which one to chose?

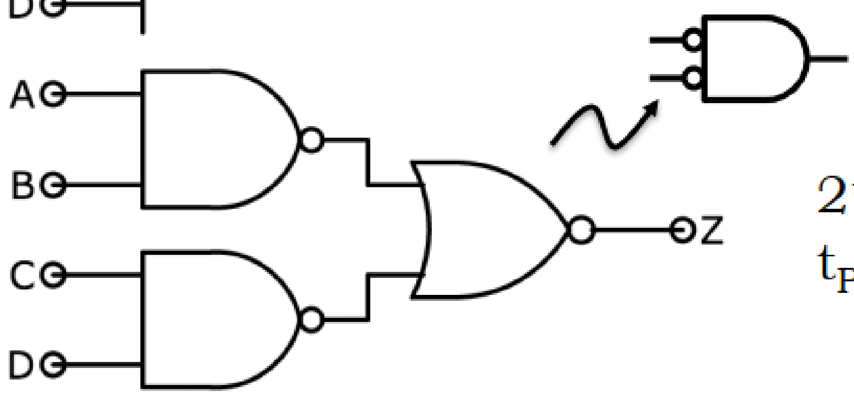


AND4:
t_{PD} = 160 ps, size = 20 μ^2



NAND4 + INV:
t_{PD} = 90 ps, size = 27 μ^2

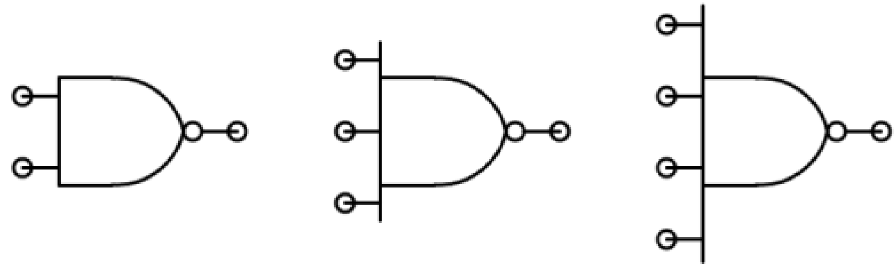
Demorgan's Laws: $\overline{A} \cdot \overline{B} = \overline{A + B}$
$\overline{A} + \overline{B} = \overline{A \cdot B}$



2*NAND2 + NOR2:
t_{PD} = 80 ps, size = 30 μ^2

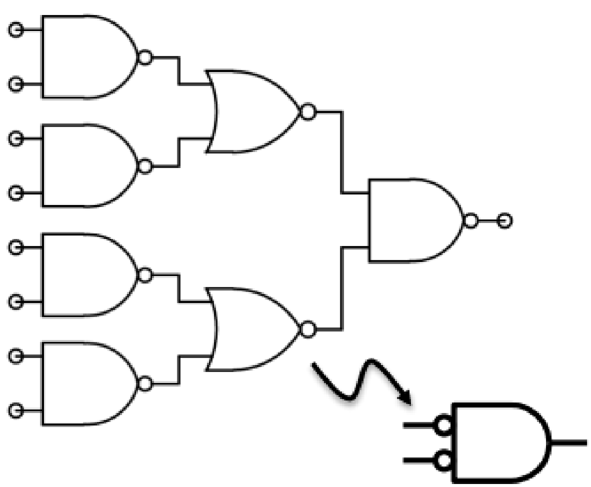
Wide NANDs and NORs

Most logic libraries include 2-, 3- and 4-input devices:

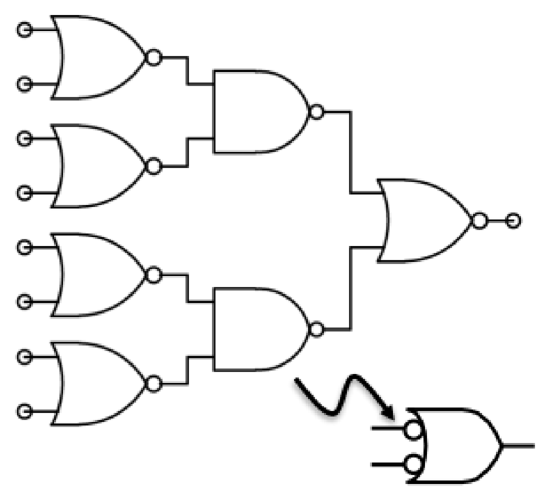


But for a large number of inputs, the series connections of too many MOSFETs can lead to very large effective R.
Design note: use trees of smaller devices…

8-input
NAND



8-input
NOR

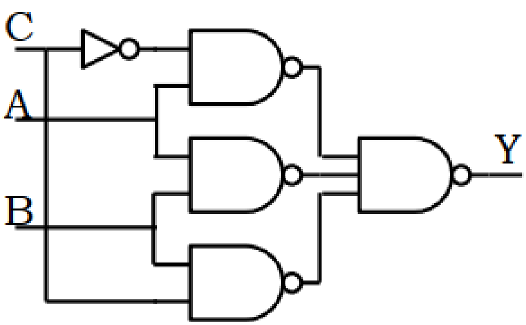


CMOS Sum-of-products Implementation

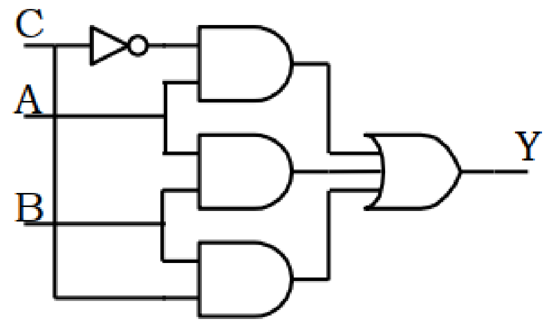
"Pushing Bubbles"

NAND-NAND

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



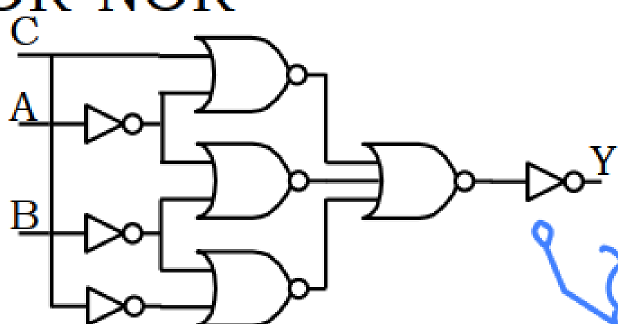
≡

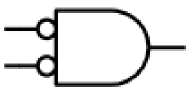


$A\bar{C} + AB + BC$

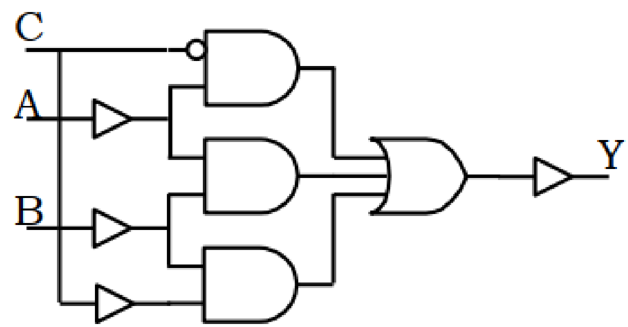
NOR-NOR

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



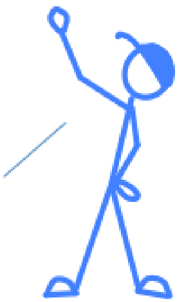


≡



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

You might think all these extra inverters would make this structure less attractive. However, quite the opposite is true.



Simplification of Boolean Functions: Two Methods

- **Algebraic method** by using Identities & Theorem
- **Graphical method** by using Karnaugh Map method
  - The K-map method is easy and straightforward.
  - A K-map for a function of n variables consists of 2^n cells, and,
  - in every row and column, two adjacent cells should differ in the value of only one of the logic variables.

Logic Simplification

Can we implement the same function with fewer gates? Before trying we'll add a few more tricks in our bag.

BOOLEAN ALGEBRA:

OR rules:	a + 1 = 1, a + 0 = a, a + a = a
AND rules:	a1 = a, a0 = 0, aa = a
Commutative:	a + b = b + a, ab = ba
Associative:	(a + b) + c = a + (b + c), (ab)c = a(bc)
Distributive:	a(b+c) = ab + ac, a + bc = (a+b)(a+c)
Complements:	$a+\overline{a}=1, \quad a\overline{a}=0$
Absorption:	$a+ab=a, \quad a+\overline{a}b=a+b \quad a(a+b)=a, \quad a(\overline{a}+b)=ab$
Reduction:	$\boxed{ab+\overline{a}b=b,} \quad (a+b)(\overline{a}+b)=b$
DeMorgan's Law:	$\overline{a}+\overline{b}=\overline{ab}, \quad \overline{a}\overline{b}=\overline{a+b}$

Boolean Minimization

Let's (again!) simplify

$$Y = \overline{C}\,\overline{B}A + CB\overline{A} + CBA + \overline{C}BA$$

Using the identity

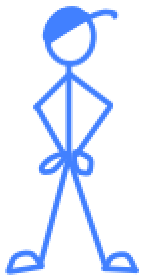
$$\alpha A + \alpha\overline{A} = \alpha(A + \overline{A}) = \alpha \cdot 1 = \alpha$$

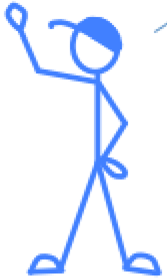
For any expression α and variable A:

$$Y = \overline{C}\,\overline{B}A + CB\overline{A} + CBA + \overline{C}BA$$

$$Y = \overline{C}\,\overline{B}A + CB + \overline{C}BA$$

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





Truth Tables with "Don't Cares"

One way to reveal the opportunities for a more compact implementation is to rewrite the truth table using "don't cares" (-- or X) to indicate when the value of a particular input is irrelevant in determining the value of the output.

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

C	B	A	Y	
0	X	0	0	
0	X	1	1	$\longrightarrow \overline{C}A$
1	0	X	0	
1	1	X	1	$\longrightarrow CB$
X	0	0	0	
X	1	1	1	$\longrightarrow BA$

Note: Some input combinations (e.g., 000) are matched by more than one row in the "don't care" table. It would be a bug if all the matching rows didn't specify the same output value!

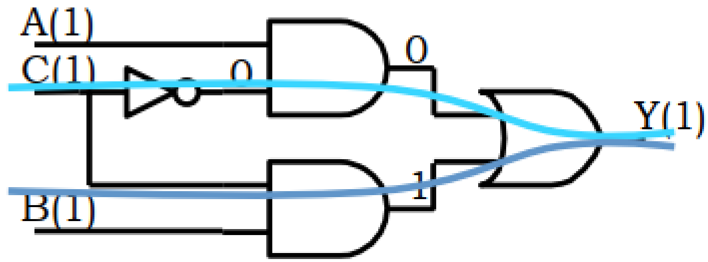
The Case for a Non-minimal SOP

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

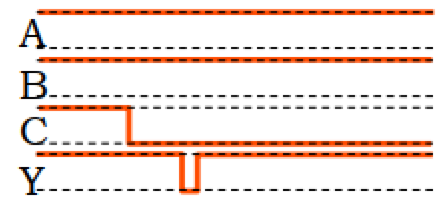
$\overline{C}A$

CB

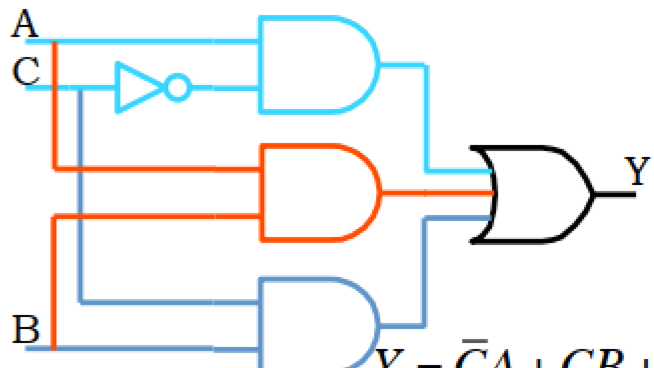
BA



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



NOTE: The steady state behavior of these circuits is identical. They differ in their transient behavior.



$$Y = \overline{C}A + CB + AB$$

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.

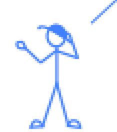
Truth Table

C	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

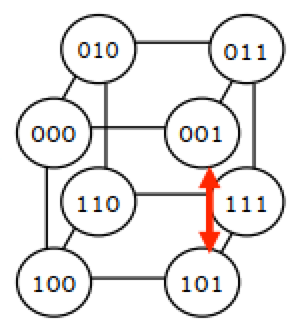
Here's the layout of a 3-variable K-map filled in with the values from our truth table:

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

Why did he shade that row Gray?



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

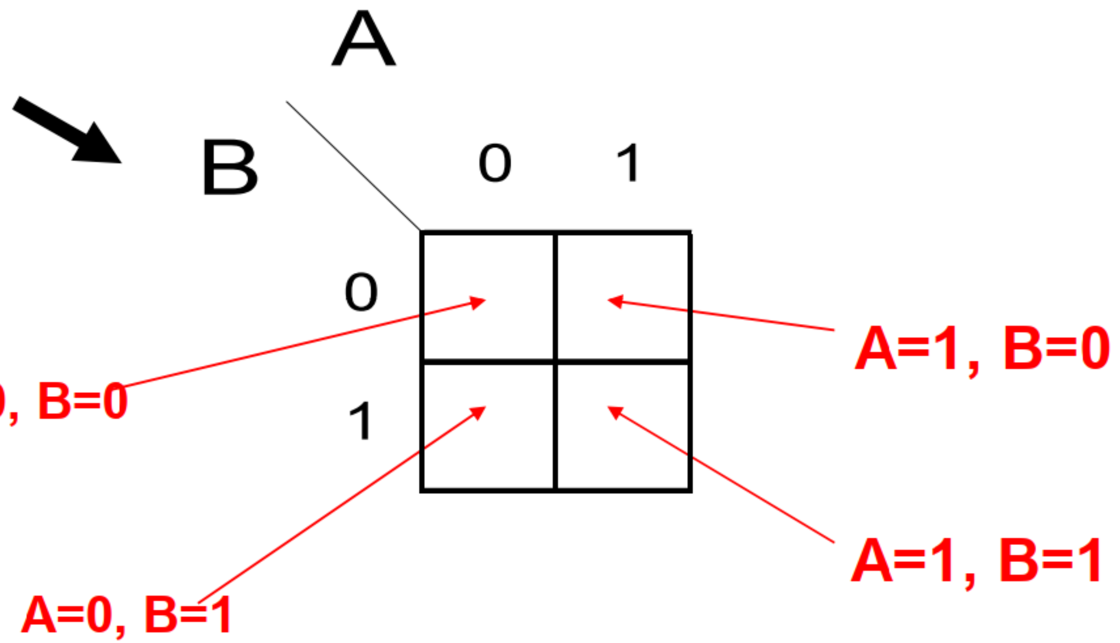


Karnaugh Map Advantages

- Minimization can be done more systematically
- Much simpler to find minimum solutions
- Easier to see what is happening (graphical)
- Almost always used instead of boolean minimization.

2-Variable Karnaugh Map

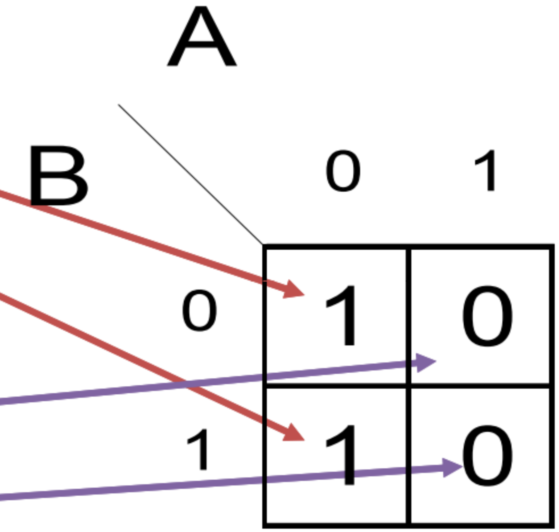
A	B	F
0	0	
0	1	
1	0	
1	1	



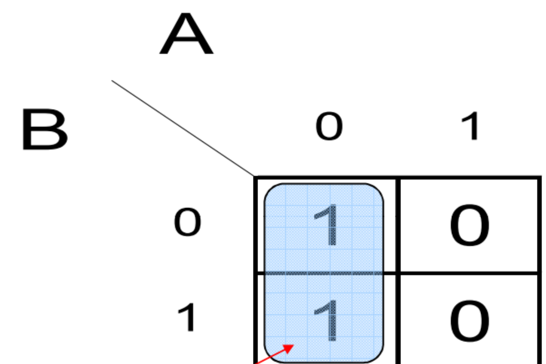
A different way to draw a truth table: by folding it

Some Examples

A	B	index	F
0	0	0	1
0	1	1	1
1	0	2	0
1	1	3	0



A	B	index	F
0	0	0	1
0	1	1	1
1	0	2	0
1	1	3	0



A = 0

F = A’

K-Map of three variable

A	B	C	index	F
0	0	0	0	1
0	0	1	1	0
0	1	0	2	1
0	1	1	3	0
1	0	0	4	1
1	0	1	5	1
1	1	0	6	1
1	1	1	7	1

F=∑m(0,2,4,5,6,7)

BC / A	B'C'	B'C	BC	BC'
A'	1 (0)	0 (1)	0 (3)	1 (2)
A	1 (4)	1 (5)	1 (7)	1 (6)

F= A + C'

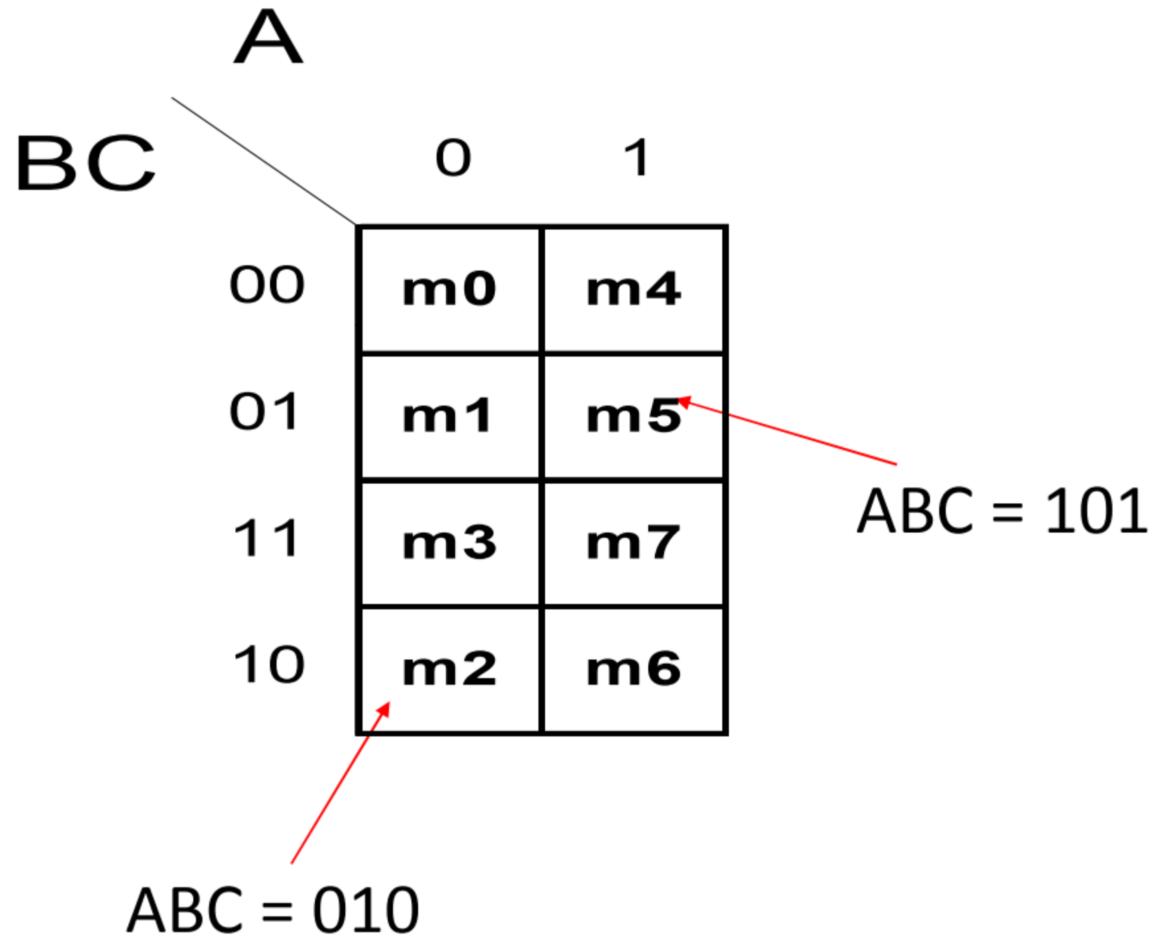
Group of 4 m(4,5,7,6)

Group of 4 m(0,2,4,6)

Another Approach for 3 variable K-Map

Note the order of the B C variables:

0 0
0 1
1 1
1 0



Minterm Expansion to K-Map

$$F = \sum m(1, 3, 4, 6)$$

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

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

Minterms are the 1's, everything else is 0

Remember Minterms

- Boolean function can be expressed algebraically from a given truth table
  - Forming sum of ALL the minterms that produce 1 in the function

Example : Consider the function defined by the truth table

F(X,Y,Z)= X'Y'Z' + X'YZ' + XY'Z + XYZ

$$= m_0 + m_2 + m_5 + m_7$$

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

X	Y	Z	m	F
0	**0**	**0**	m_0	**1**
0	0	1	m_1	0
0	**1**	**0**	m_2	**1**
0	1	1	m_3	0
1	0	0	m_4	0
1	**0**	**1**	m_5	**1**
1	1	0	m_6	0
1	**1**	**1**	m_7	**1**

Extending K-maps to 4-variable Tables

4-variable K-map F(A,B,C,D):

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

Again it's cyclic. The left edge is adjacent to the right edge, and the top is adjacent to the bottom.

For functions of 5 or 6 variables, we'd need to use the 3rd dimension to build a 4x4x4 K-map. But then we're out of dimensions…

Finding Implicants

An implicant

- is a rectangular region of the K-map where the function has the value 1 (i.e., a region that will need to be described by one or more product terms in the sum-of-products)
- has a width and length that must be a power of 2: 1, 2, 4
- can overlap other implicants
- is a prime implicant if it is not completely contained in any other implicant.

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

$A\overline{C}$

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

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

$\overline{A}C$

$\overline{B}$

- can be uniquely identified by a single product term. The larger the implicant, the smaller the product term.

Finding Prime Implicants

We want to find all the prime implicants. The right strategy is a greedy one.

- Find the uncircled prime implicant with the greatest area
  - Order: 4x4 ⇒ 2x4 or 4x2 ⇒ 4x1 or 1x4 or 2x2 ⇒ 2x1 or 1x2 ⇒ 1x1
  - Overlap is okay
- Circle it
- Repeat until all prime implicants are circled

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

Write Down Equations

Picking just enough prime implicants to cover all the 1's in the KMap, combine equations to form minimal sum-of-products.

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

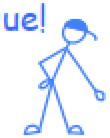
$$Y = A\overline{C} + BC$$

We're done!

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

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

Minimal SOP is not necessarily unique!

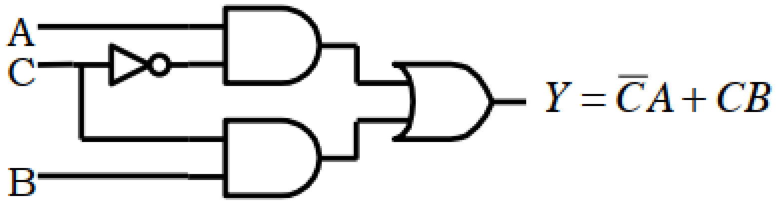


\AB CD\	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	0	0	1

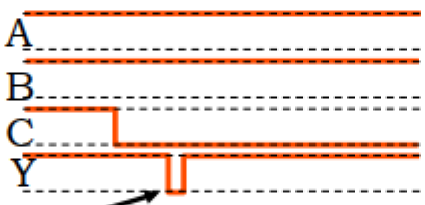
$$Y = D + B\overline{C} + A\overline{B} + \overline{B}C$$

Prime Implicants, Glitches & Leniency

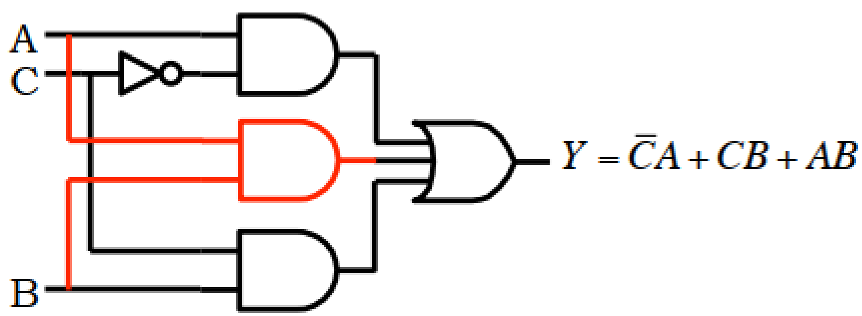
This circuit produces a glitch on Y when A=1, B=1, C: 1→0



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

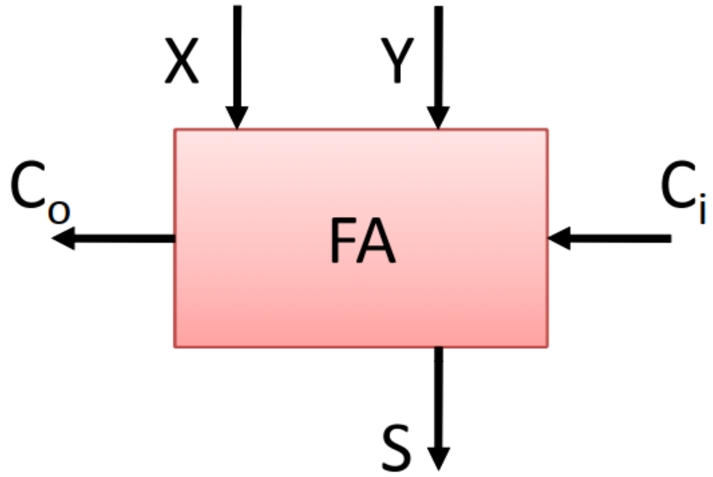


To make the circuit lenient, include product terms for ALL prime implicants.



Example 1

Ci	X	Y	index	S	Co
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	2	1	0
0	1	1	3	0	1
1	0	0	4	1	0
1	0	1	5	0	1
1	1	0	6	0	1
1	1	1	7	1	1



$$S = \sum m(1, 2, 4, 7)$$

$$Co = \sum m(3, 5, 6, 7)$$

Example 1

$$S = \sum m(1, 2, 4, 7)$$

XY \ Ci	0	1
00	0	1
01	1	0
11	0	1
10	1	0

$$S = Ci'X'Y + Ci'XY' + CiX'Y' + CiXY$$

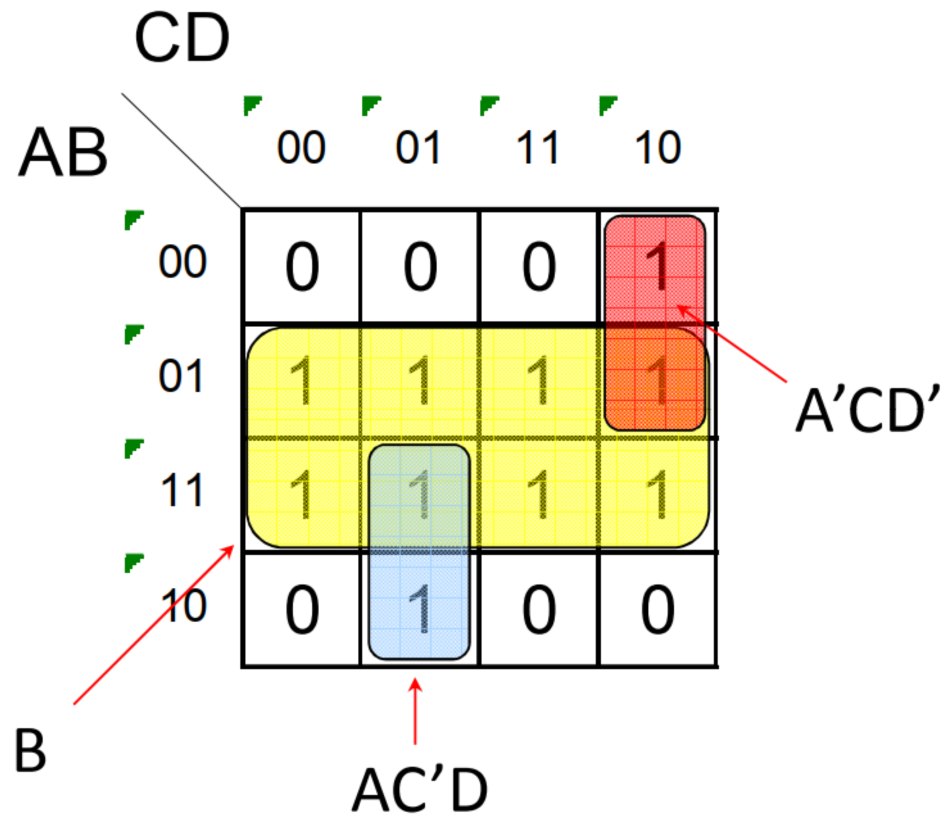
$$Co = \sum m(3, 5, 6, 7)$$

XY \ Ci	0	1
00	0	0
01	0	1
11	1	1
10	0	1

$$Co = XY + CiX + CiY$$

Example 2

AB \ CD	00	01	11	10
00	**m0**	**m1**	**m3**	**m2**
01	**m4**	**m5**	**m7**	**m6**
11	**m12**	**m13**	**m15**	**m14**
10	**m8**	**m9**	**m11**	**m10**



F = AC'D + A'CD' + B

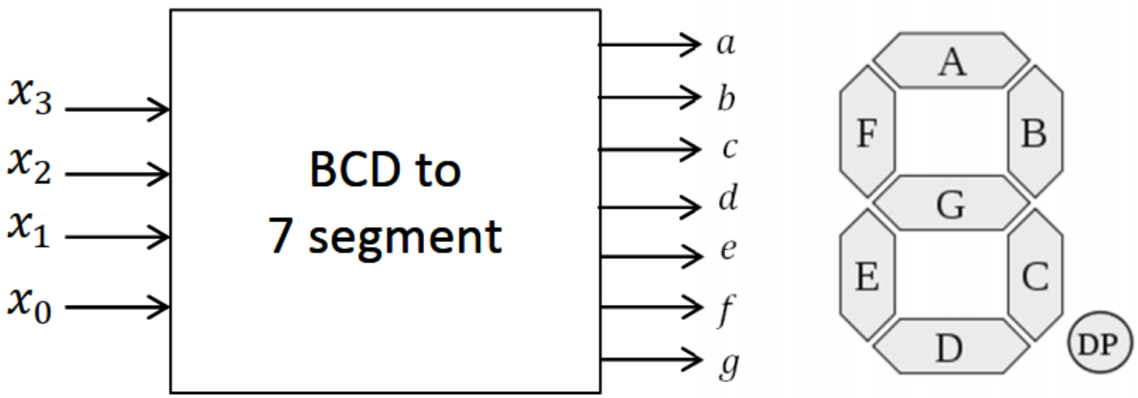
Don’t Care

- A **don't-care term** is an input to a function that the designer does not care about
- Because that input would never happen
- Example:
  - BCD number (0-9, A-F) are 4 bits, don’t care about input A-F
  - Suppose a system have 5 type of input
    - Unfortunately we can’t have 2 input line
    - Make 3 input line and last 3 sequence as don’t care
    - S0, S1, S2,S3,S4, X,X,X == > 000, 001….,111

BCD to 7- Segment Decoder Example

Some combinations of input signal values could **never occur**, or, when they occur, the output signal values do not matter **(don't care)**. The corresponding minterms can be used, or not, in order to optimize the final circuit.

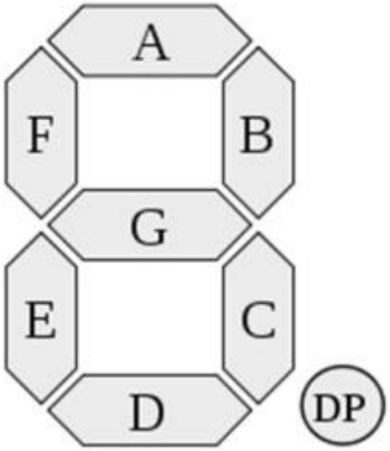
EXAMPLE: BCD to 7-segment decoder.



BCD to 7- Segment Decoder Example

EXAMPLE: BCD to 7-segment decoder.

x3	x2	x1	x0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	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							



BCD to 7- Segment Decoder Example

x3	x2	x1	x0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	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							

$$b = \bar{x}_3.\bar{x}_2 + \bar{x}_2.\bar{x}_1 + \bar{x}_3.\bar{x}_1.\bar{x}_0 + \bar{x}_3.x_1.x_0$$

x_3 x_2 x_1 x_0	b	c
0 0 0 0	1	1
0 0 0 1	1	1
0 0 1 0	1	0
0 0 1 1	1	1
0 1 0 0	1	1
0 1 0 1	0	1
0 1 1 0	0	1
0 1 1 1	1	1
1 0 0 0	1	1
1 0 0 1	1	1
1 0 1 0	1	0
1 0 1 1	1	1
1 1 0 0	1	1
1 1 0 1	0	1
1 1 1 0	0	1
1 1 1 1	1	1

$$b = \bar{x}_2 + \bar{x}_1.\bar{x}_0 + x_1.x_0$$

Dealing With Don’t Cares

$F = \Sigma m(1, 3, 7) + \Sigma d(0, 5)$

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

A’B’C+AB’C+
A’BC+ABC
= C

F = C

Circle the x’s that help get bigger groups of 1’s
Don’t circle the x’s that don’t

BCD to 7- Segment Decoder Example

$a = \bar{x}_3.x_1 + \bar{x}_3.x_2.x_0 + x_3.\bar{x}_2.\bar{x}_1$

$b = \bar{x}_3.\bar{x}_2 + \bar{x}_2.\bar{x}_1 + \bar{x}_3.\bar{x}_1.\bar{x}_0 + \bar{x}_3.x_1.x_0$

$c = \bar{x}_2.\bar{x}_1 + \bar{x}_3.x_0 + \bar{x}_3.x_2$

$d = \bar{x}_2.\bar{x}_1.\bar{x}_0 + \bar{x}_3.\bar{x}_2.x_1 + \bar{x}_3.x_1.\bar{x}_0 + \bar{x}_3.x_2.\bar{x}_1.x_0$

$e = \bar{x}_2.\bar{x}_1.\bar{x}_0 + \bar{x}_3.x_1.\bar{x}_0$

$f = \bar{x}_3.\bar{x}_1.\bar{x}_0 + \bar{x}_3.x_2.\bar{x}_1 + \bar{x}_3.x_2.\bar{x}_0 + x_3.\bar{x}_2.\bar{x}_1$

$g = \bar{x}_3.\bar{x}_2.x_1 + \bar{x}_3.x_2.\bar{x}_1 + \bar{x}_3.x_2.\bar{x}_0 + x_3.\bar{x}_2.\bar{x}_1$

$a = x_1 + x_2.x_0 + x_3$

$b = \bar{x}_2 + \bar{x}_1.\bar{x}_0 + x_1.x_0$

$c = \bar{x}_1 + x_0 + x_2$

$d = \bar{x}_2.\bar{x}_0 + \bar{x}_2.x_1 + x_1.\bar{x}_0 + x_2.\bar{x}_1.x_0$

$e = \bar{x}_2.\bar{x}_0 + x_1.\bar{x}_0$

$f = \bar{x}_1.\bar{x}_0 + x_2.\bar{x}_1 + x_2.\bar{x}_0 + x_3$

$g = \bar{x}_2.\bar{x}_0 + x_2.\bar{x}_1 + x_1.\bar{x}_0 + x_3$

35	***total***	***26***
24	AND	15
7	OR	7
4	INV	4

5-Variable Karnaugh Map

DE \ BC	00	01	11	10
00	m0	m4	m12	m8
01	m1	m5	m13	m9
11	m3	m7	m15	m11
10	m2	m6	m14	m10

This is the A=0 plane

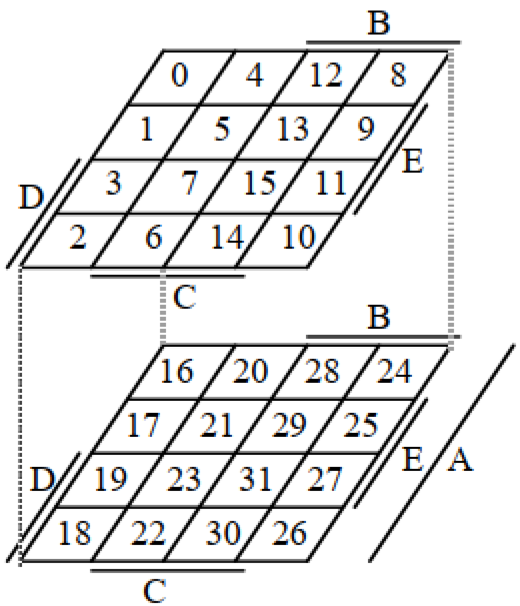
DE \ BC	00	01	11	10
00	m16	m20	m28	m24
01	m17	m21	m29	m25
11	m19	m23	m31	m27
10	m18	m22	m30	m26

This is the A=1 plane

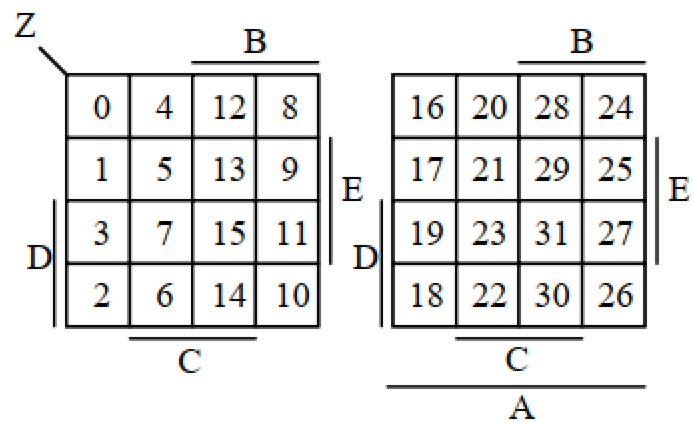
The planes are adjacent to one another (one is above the other in 3D)

Alternate Version

- Five-Variable Maps.

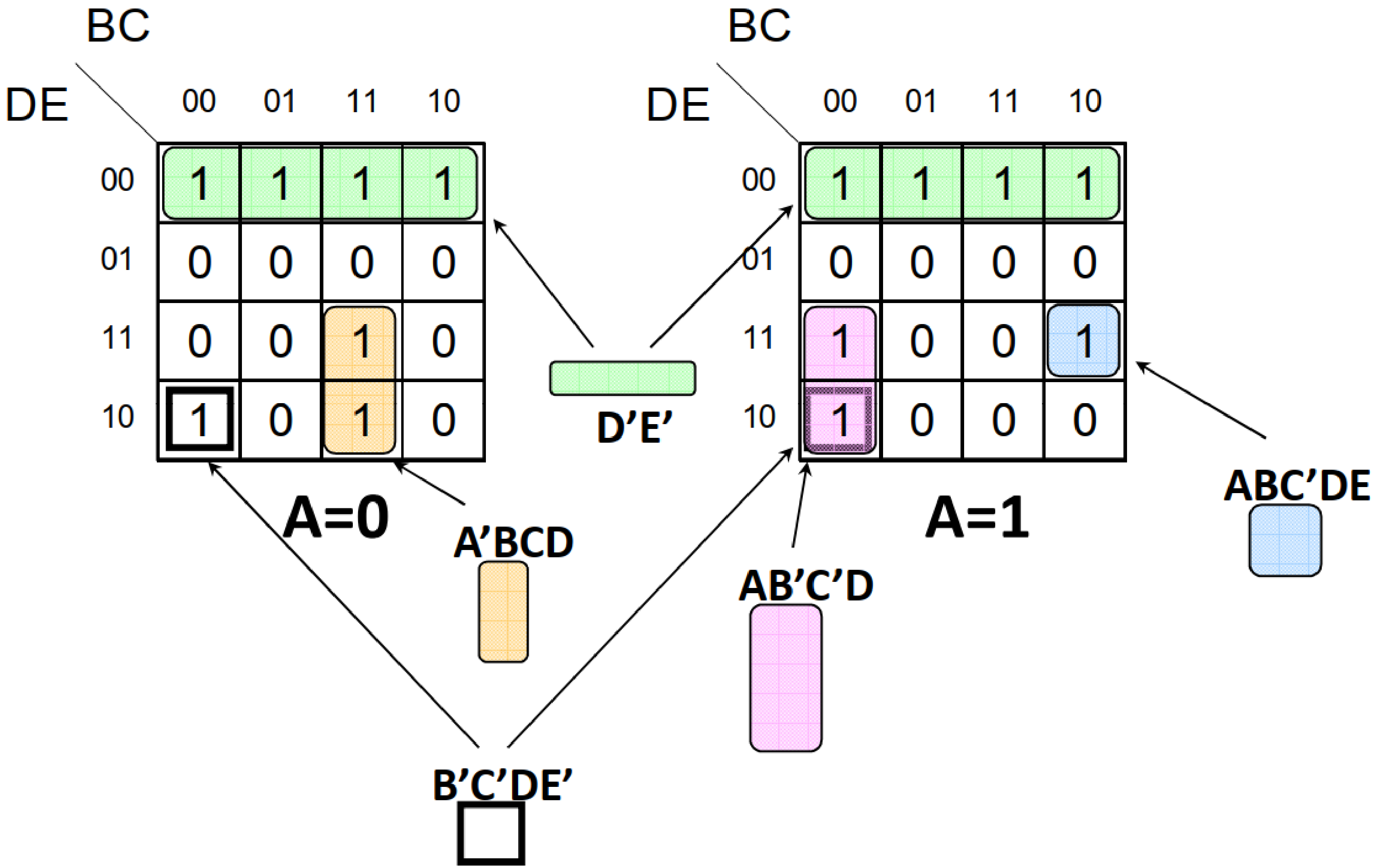


Five-Variable Map Structure



Alternate Version of Five-Variable Map

5-Variable Karnaugh Map Example



6-Variable Karnaugh Map

AB=00

EF \ CD	00	01	11	10
00	m0	m4	m12	m8
01	m1	m5	m13	m9
11	m3	m7	m15	m11
10	m2	m6	m14	m10

AB=01

EF \ CD	00	01	11	10
00	m16	m20	m28	m24
01	m17	m21	m29	m25
11	m19	m23	m31	m27
10	m18	m22	m30	m26

AB=10

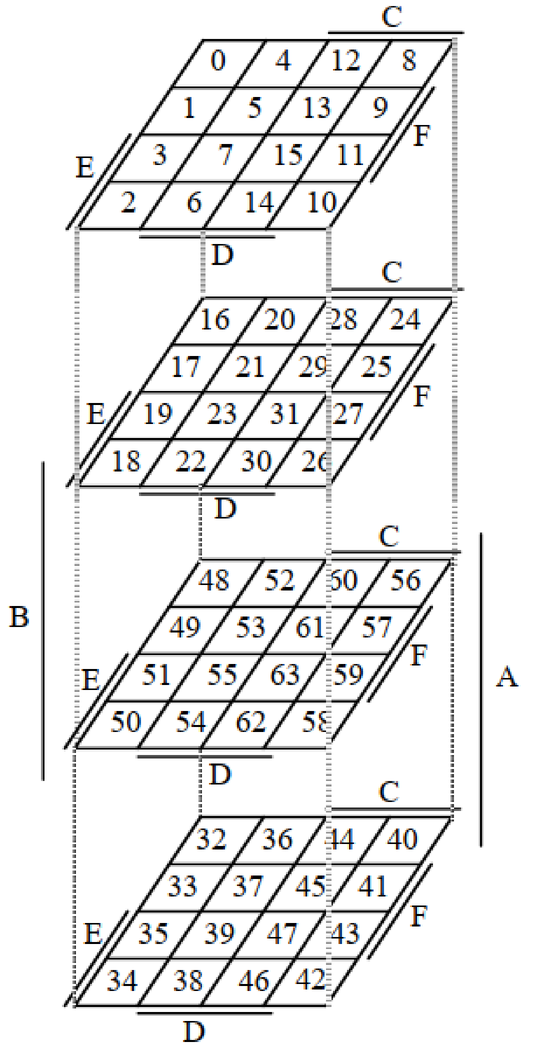
EF \ CD	00	01	11	10
00	m32	m36	m44	m40
01	m33	m37	m45	m41
11	m35	m39	m47	m43
10	m34	m38	m46	m42

AB=11

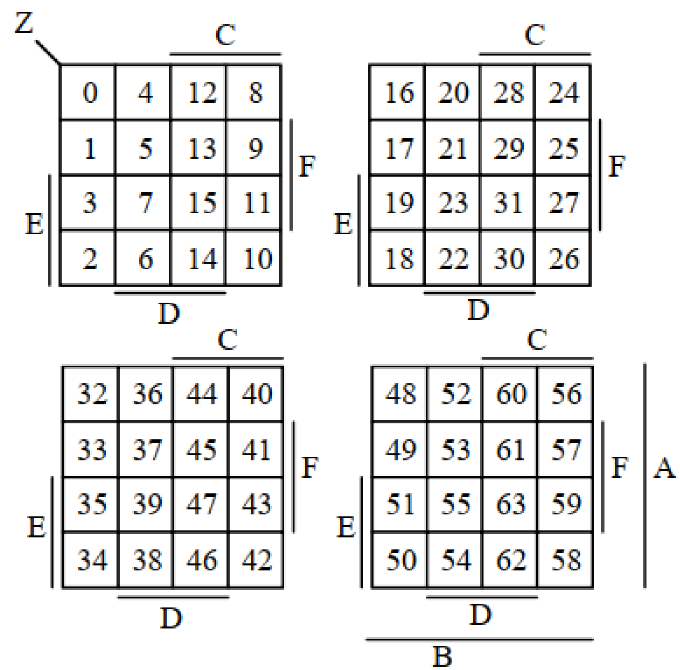
EF \ CD	00	01	11	10
00	m48	m52	m60	m56
01	m49	m53	m61	m57
11	m51	m55	m63	m59
10	m50	m54	m62	m58

Alternate Version

- Six-Variable Maps



Six Variable Map Structure



Alternate Version of Six-Variable Map