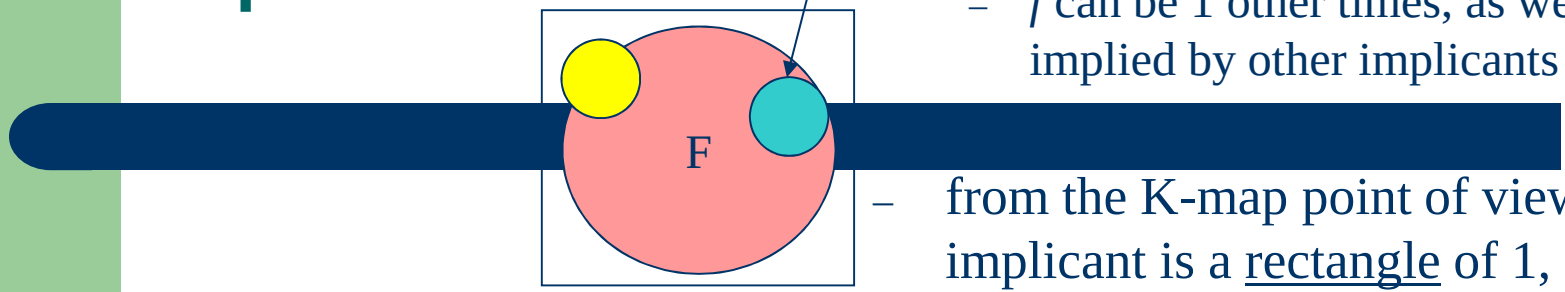
Lecture 8 – Gate-Level Minimization – Part II

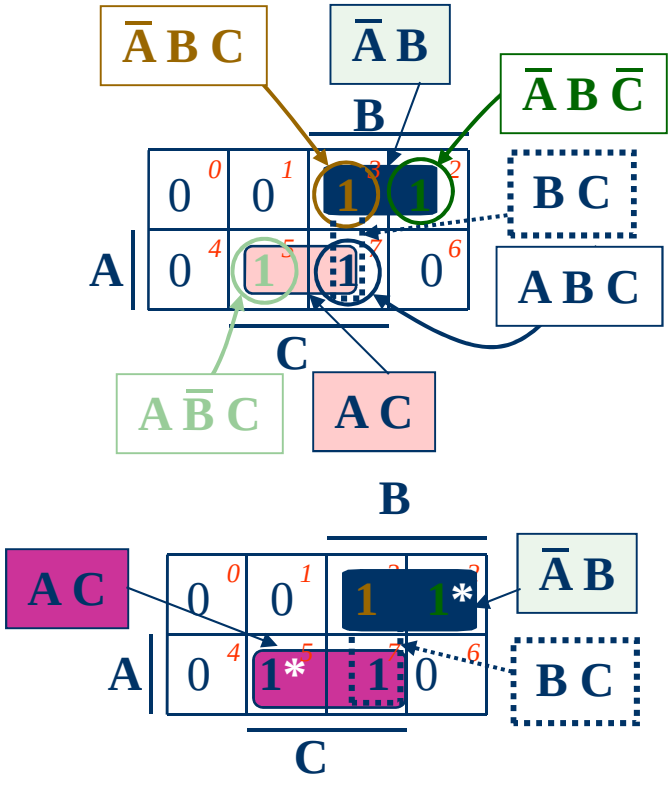
Rami Abielmona
University of Ottawa
February 06, 2015
ITI 1100 B
Digital Systems I

Presentation Outline

- Implicants
- POS Form
- Seven-Segment Decoder
- Implementation using NAND/NOR
- Other Implementations
- Key terms

Implicants





Given a logic function $z = F(x_0, x_1, x_2, \ldots)$

Implicant = product term that,
if equal to 1, implies $f = 1$

- whenever the implicant is $1 \Rightarrow f = 1$
- f can be 1 other times, as well: may be implied by other implicants

- from the K-map point of view, an implicant is a rectangle of 1, 2, 4, 8, … (any power of 2) 1's

Prime Implicant = cannot be totally covered by another implicant
(e.g., **AC, BC, A'B**)

Each "1" which is covered by a single implicant is marked with a star (*)

Essential Prime Implicant = a prime implicant that contains at least one "1*"

A **minimum cover** of a logic function has to contain all its prime implicants

Prime implicants

When grouping square:

• a group should contain a maximum of adjacent squares

→Known as *PRIME IMPLICANT*

• each group represents one term in the function

• a function should contain a minimum set of terms

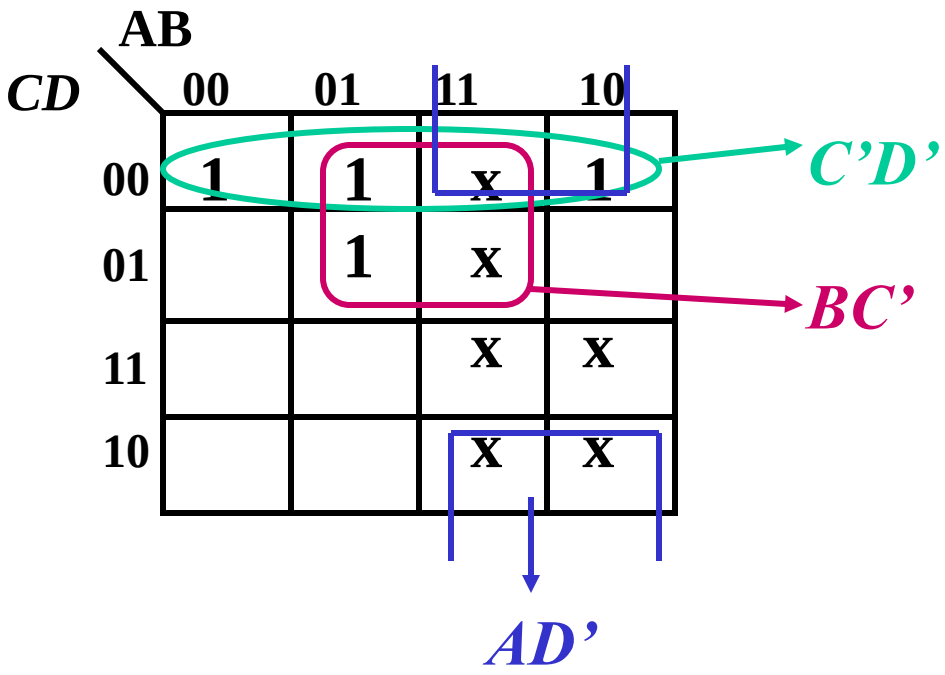
when selecting groups :

- Select only those groups that have at least one square that is not covered by another group:

→ Known as *Essential Prime Implicant*

- Do not select a group that has all its squares covered by other groups: →Known as *Optional prime implicant*

b) Obtain Sum of Products for F



$$F = BC' + C'D'$$

c) Obtain product of Sums : 2 STEPS

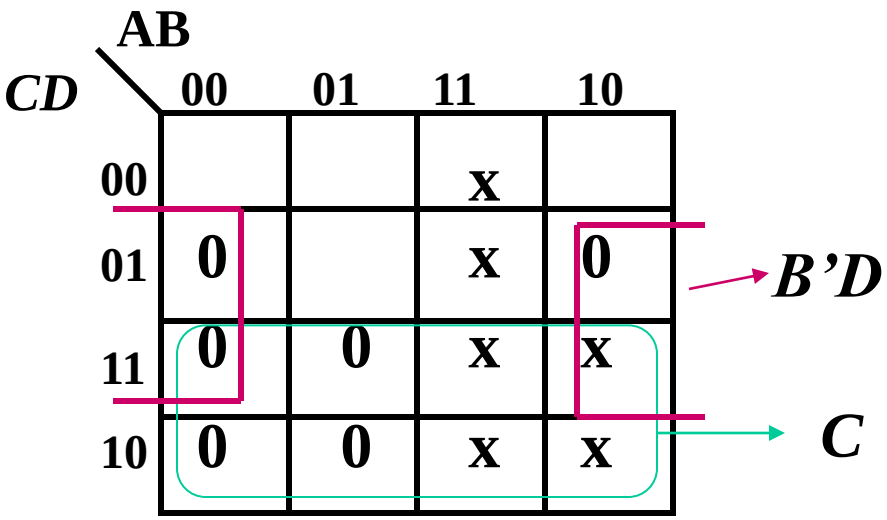
1 - use Minterms to simplify and obtain F'

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

2 - complement F' to get the Product of Sum form

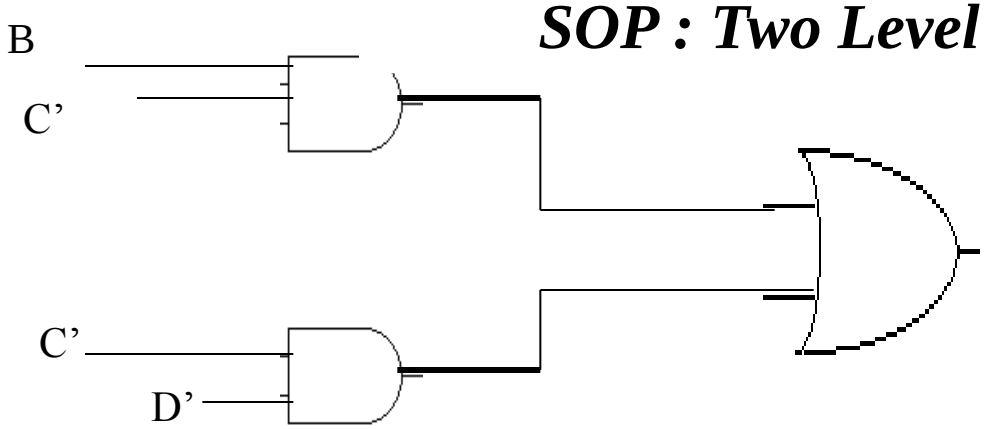
$$F'' = (B'D + C)' = (B''+D').C'$$

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



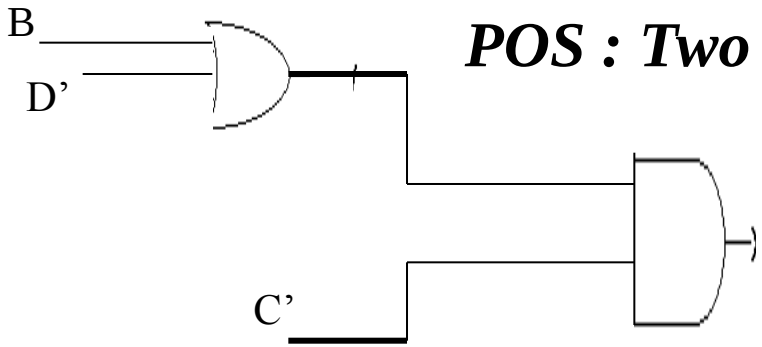
Two Level Implementations

SOP : Two Level Implementation



$$F = BC' + C'D'$$

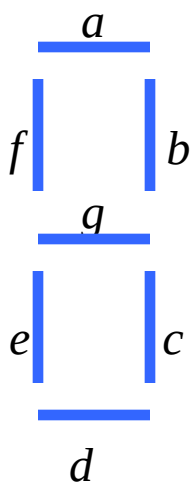
POS : Two Level Implementation



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

Seven Segment Decoder -Example

a BCD to Seven Segment Decoder inputs data in BCD form and converts it to a seven segment output

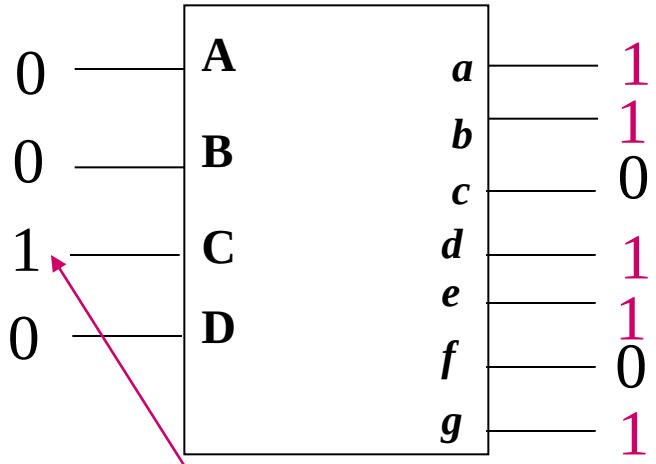


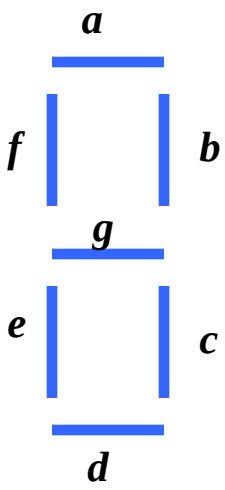
(a) Segment designation

(b) Numerical designation for display

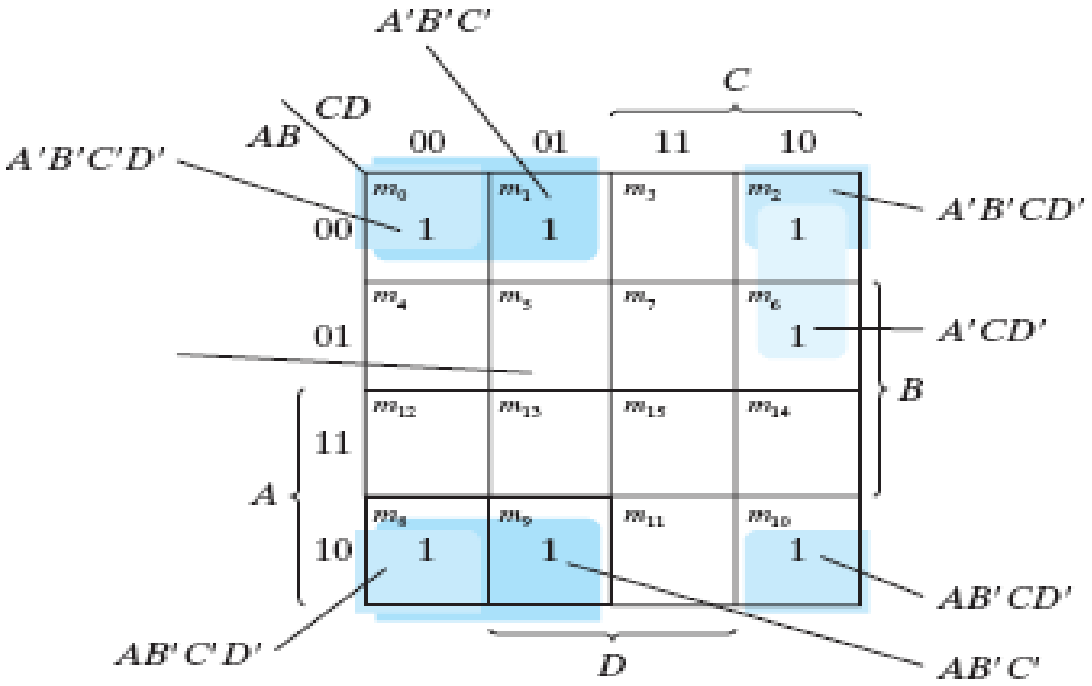
A- BCD to Seven Segment Decoder

A	B	C	D	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	1	0	1	1
1	0	1	0	x	x	x	x	x	x	x
1	0	1	1	x	x	x	x	x	x	x
1	1	0	0	x	x	x	x	x	x	x
1	1	0	1	x	x	x	x	x	x	x
1	1	1	0	x	x	x	x	x	x	x
1	1	1	1	x	x	x	x	x	x	x





Don’t care terms



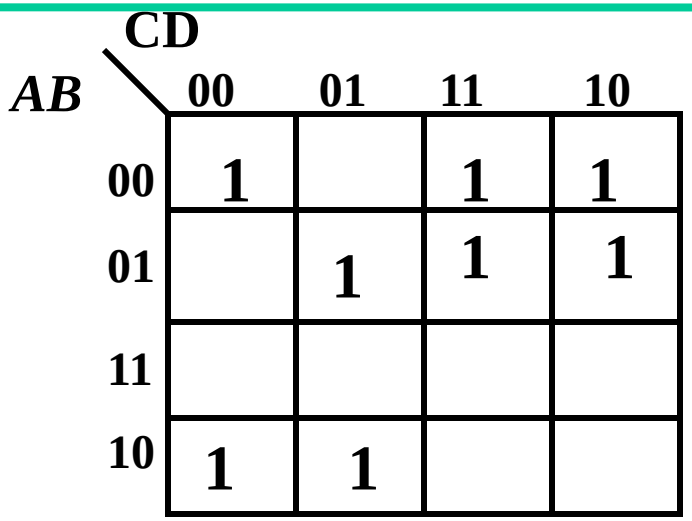
Note: $A'B'C'D' + A'B'CD' = A'B'D'$

$AB'C'D' + AB'CD' = AB'D'$

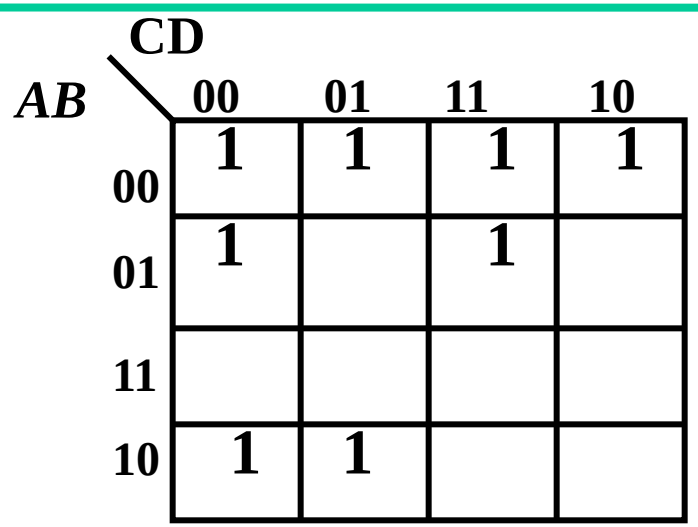
$A'B'D' + AB'D' = B'D'$

$A'B'C' + AB'C' = B'C'$

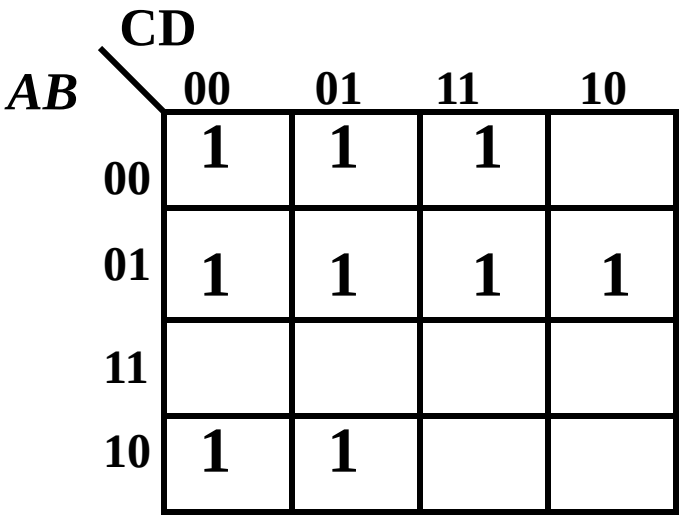
K-MAP


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

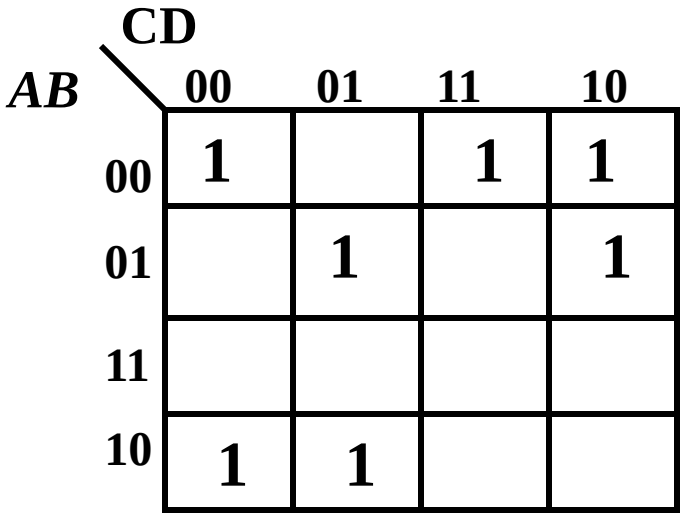
$a =$


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

$b =$

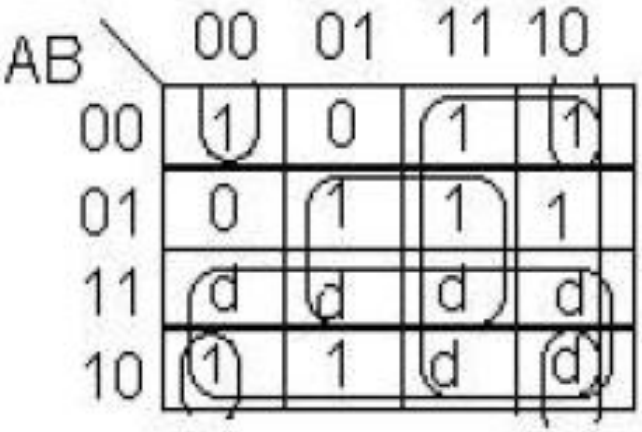

CD
AB
00 01 11 10
00 1 1 1
01 1 1 1 1
11
10 1 1

$c =$


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

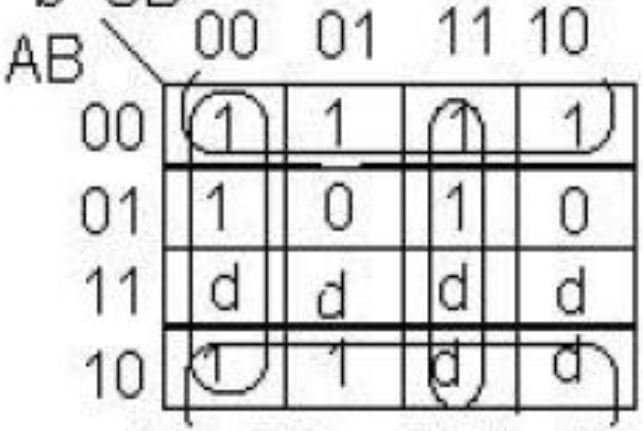
$d =$

a



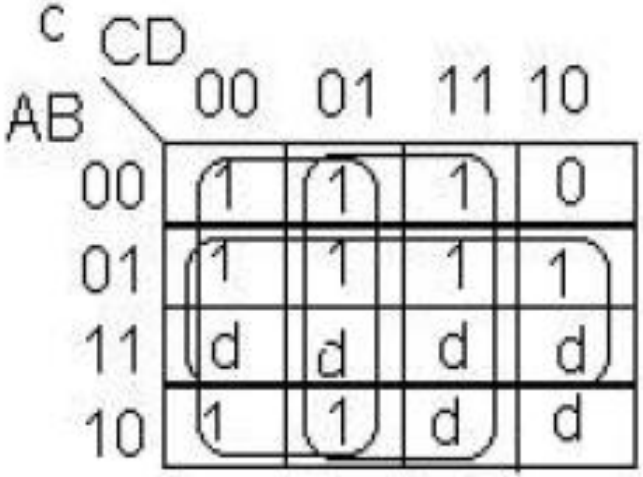
a = A + BD + B'D'+ C

b



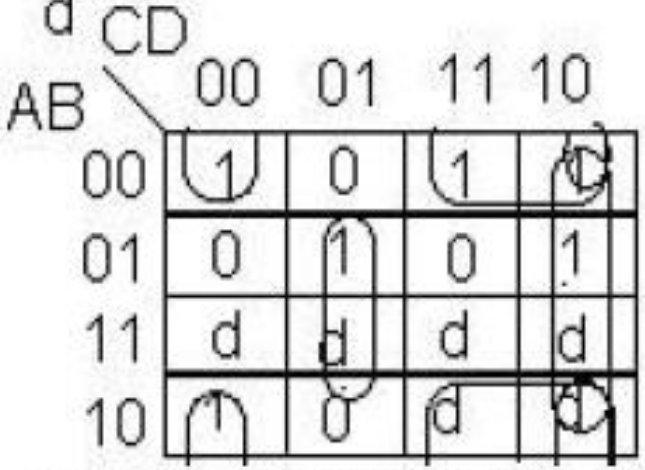
b = CD + C'D' + B'

c

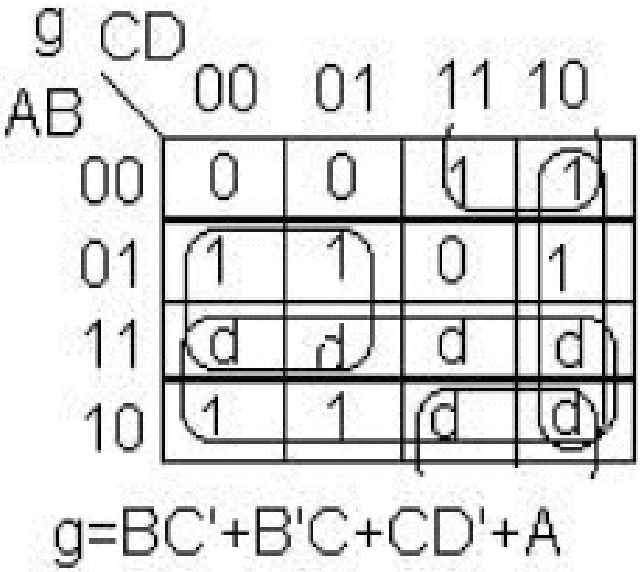


c= B + C'+ D

d



d=BC'D+B'C+B'D'+CD'


g
CD
AB
00 01 11 10
00 0 0 1 1
01 1 1 0 1
11 d d d d
10 1 1 d d
g=BC'+B'C+CD'+A

Implementations using NAND & NOR Gates

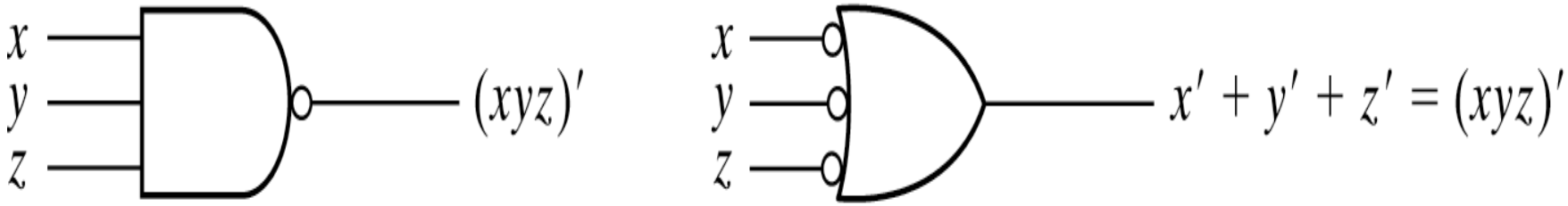
• Digital circuits are frequently constructed with NAND or NOR gates rather with AND and OR gates.

→ Both NAND and NOR gates are very valuable as any design can be realized using either one.

•It is easier to build digital circuits using all NAND or NOR gates than to combine AND,OR, and NOT gates.

•NAND/NOR gates are typically faster and cheaper to produce.

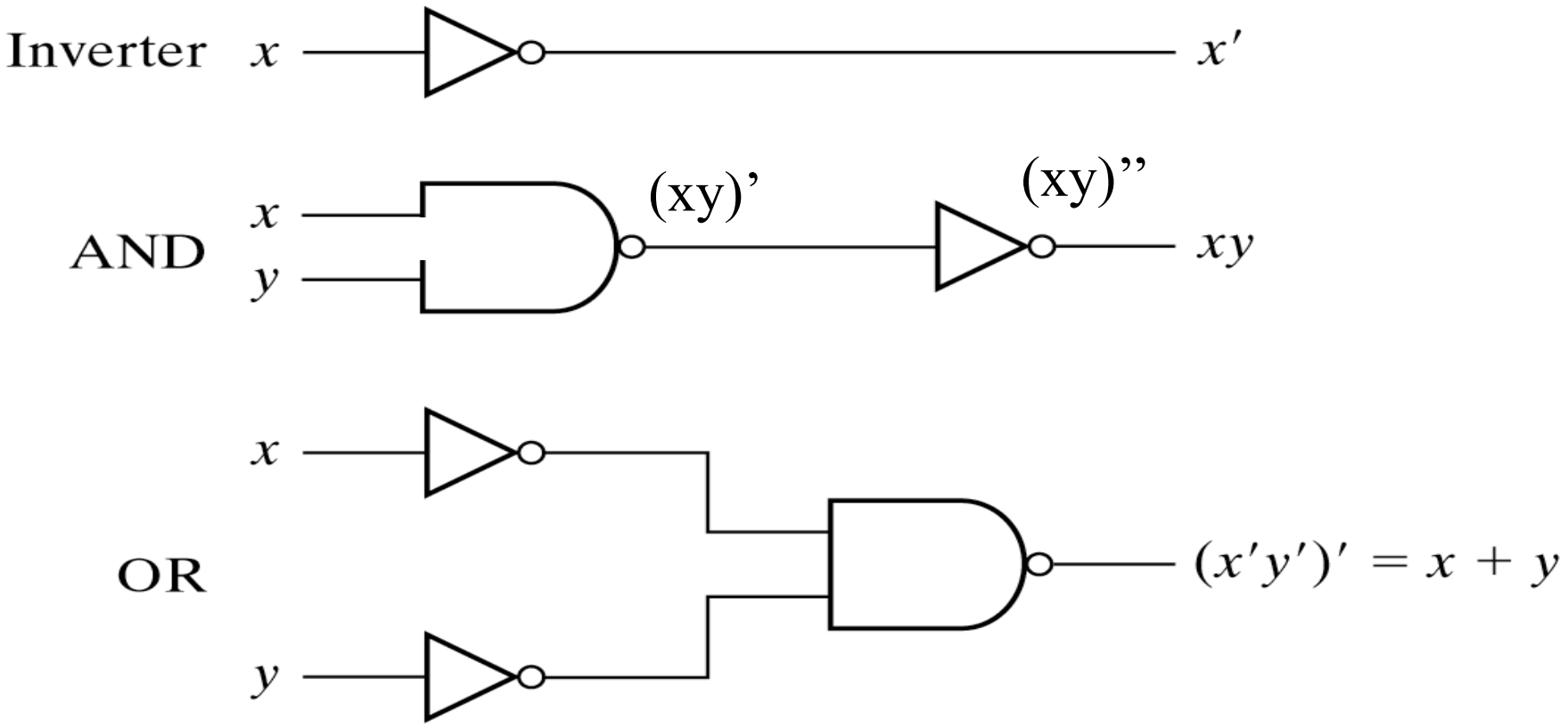
Logic Operations with NAND Gates



(a) AND–invert (b) Invert–OR

Two Graphic Symbols for NAND Gate

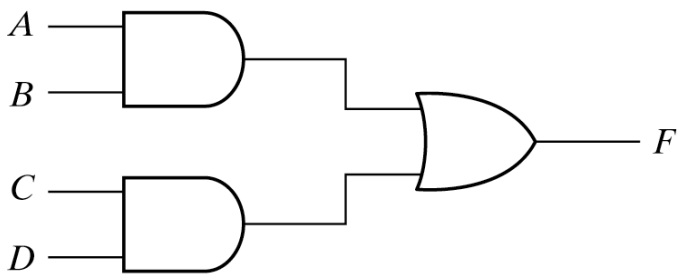
Logic Operations with NAND Gates



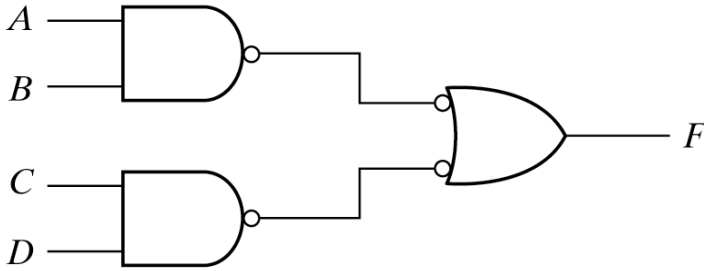
Logic Operations with NAND Gates

NAND gates Implementations

(AB)'' + (CD)''= ((AB)'(CD)')'

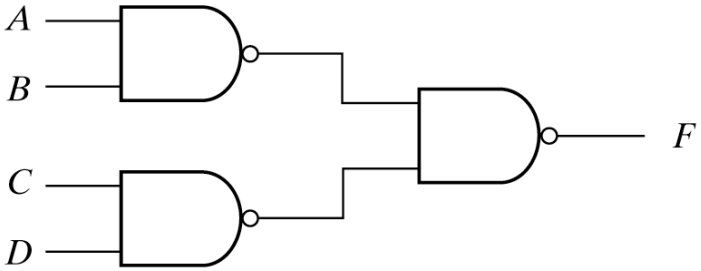


a) Two level with AND-OR



b) Two level with NAND& Invert- OR

SOP



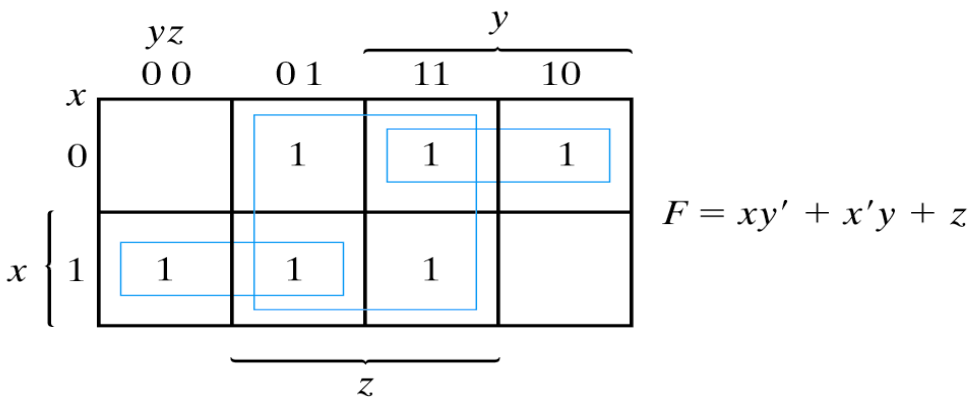
c) Two level with NAND gates
(use this in exam)

Three Ways to Implement F = AB + CD

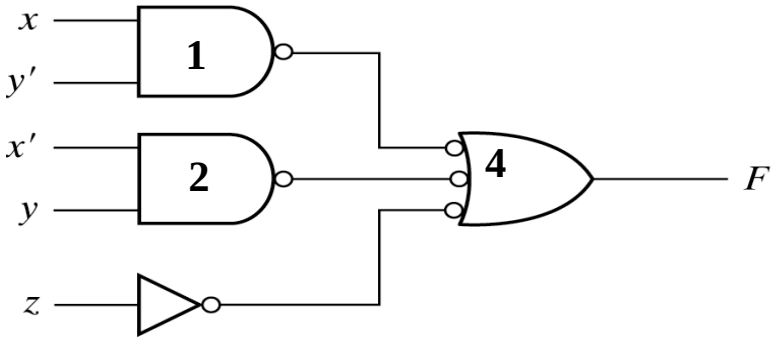
NAND gates implementations -Examples

1: (xy')' 2: (x'y)' 4: ((xy')')' + ((x'y)')' + (z')'= xy' + x'y +z

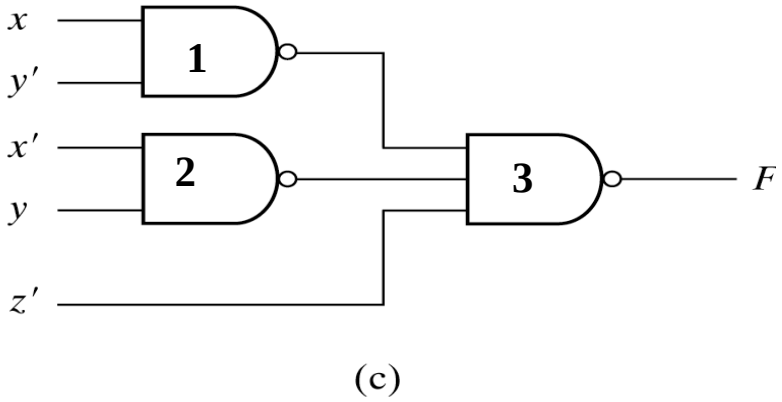
3: ((xy')'(x'y)'(z'))'= (xy')" + (x'y)"+ (z')' = xy' + x'y +z



(a)



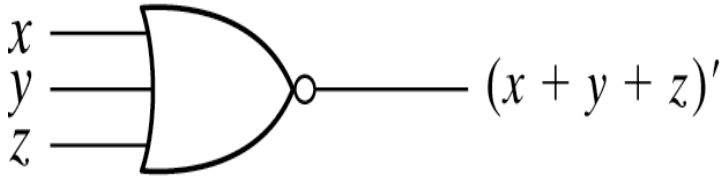
(b)



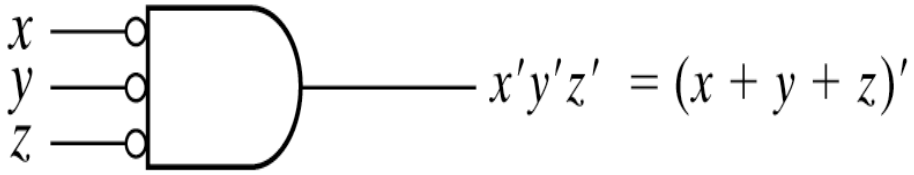
(c)

Using NAND gates to implement SOP

Logic Operations with NOR Gates



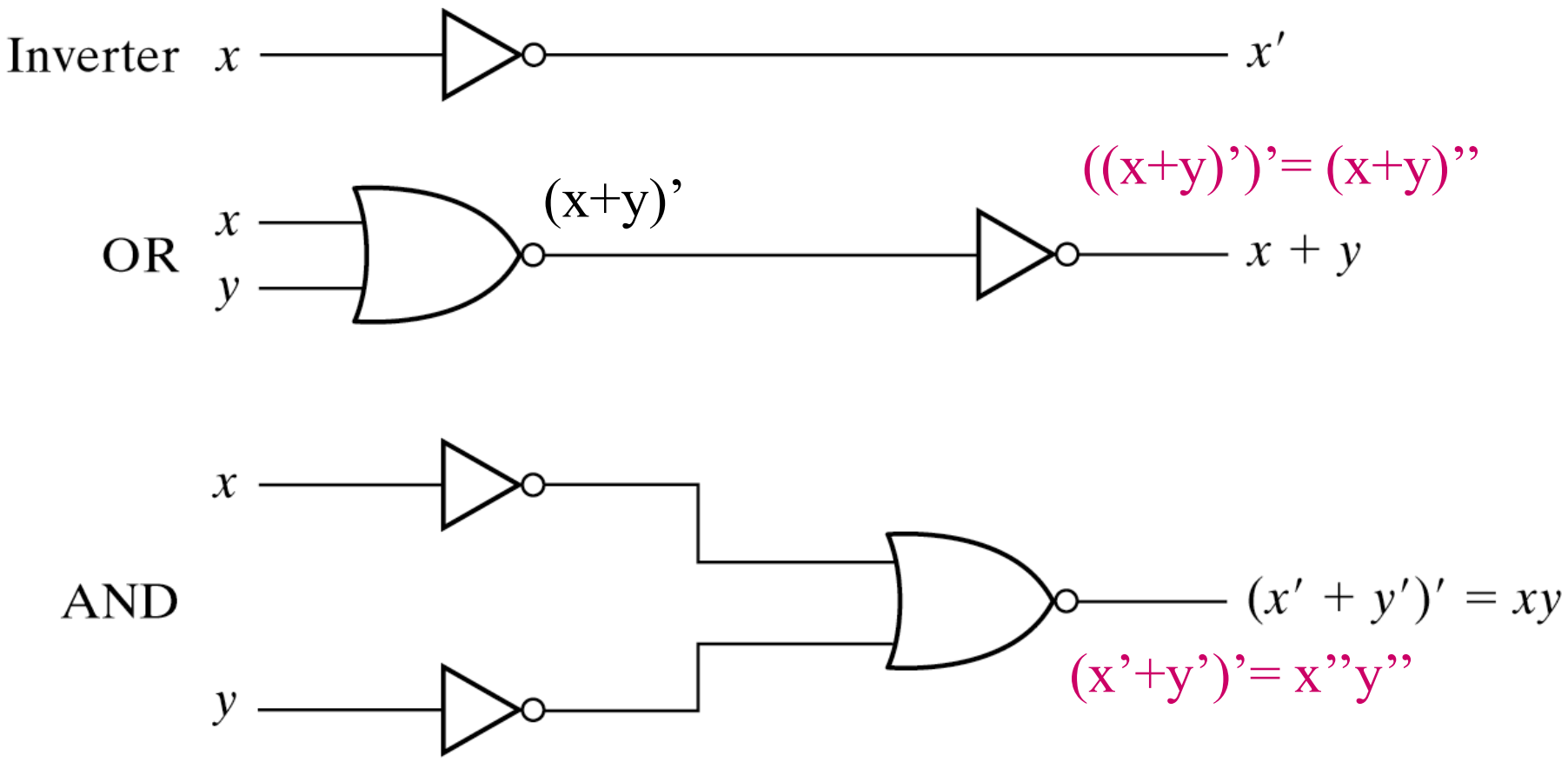
(a) OR–invert



(a) Invert–AND

Two Graphic Symbols for NOR Gate

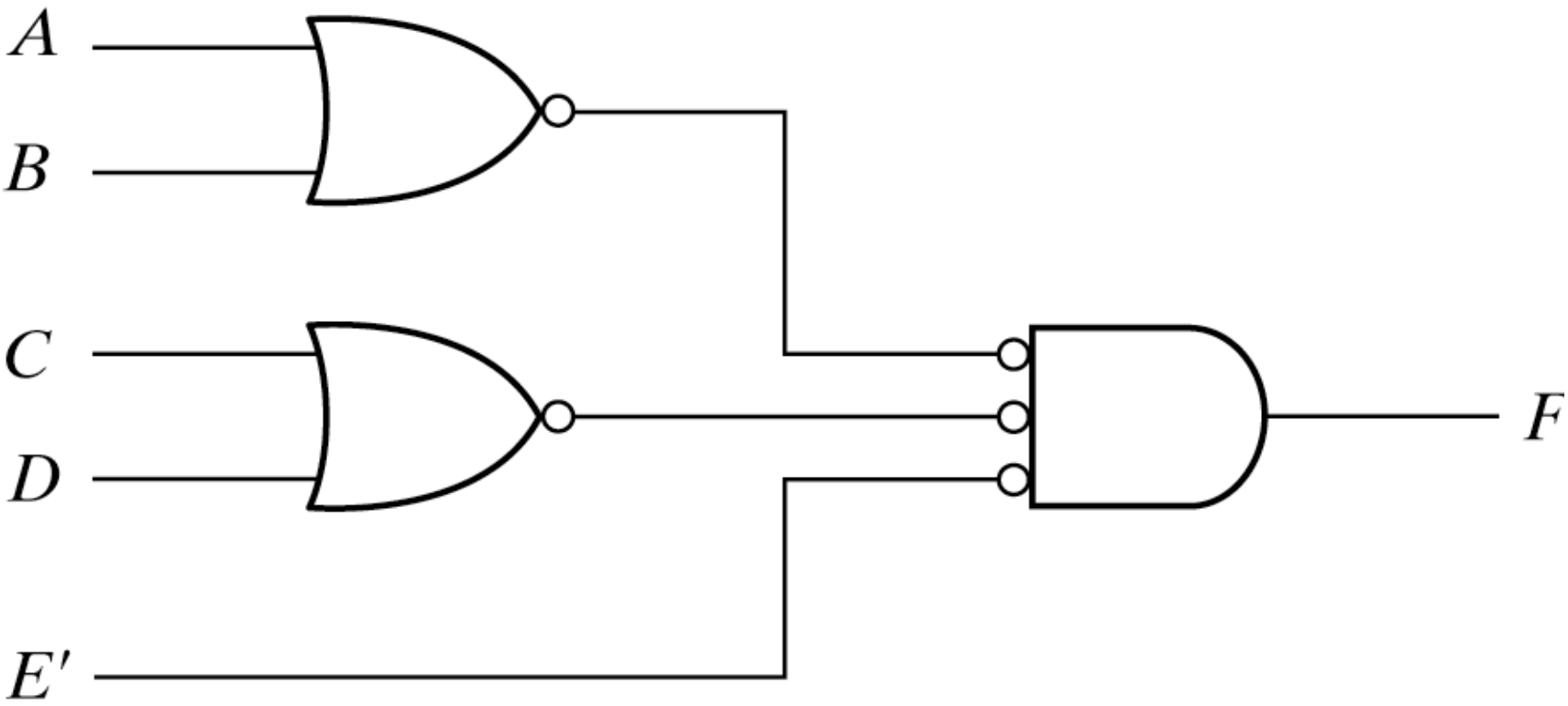
Logic Operations with NOR Gates



Logic Operations with NOR Gates

NOR gates Implementation -Examples

POS with NOR



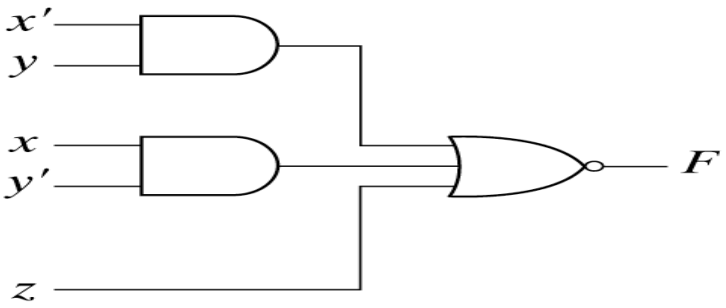
Implementing $F = (A + B)(C + D)E$

Other implementation examples

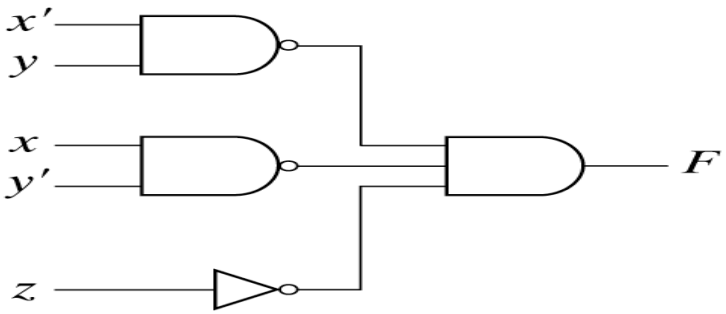
x \ yz	00	01	11	10
0	1	0	0	0
1	0	0	0	1

$$F = x'y'z' + xyz'$$
$$F' = x'y + xy' + z$$

(a) Map simplification in sum of products.

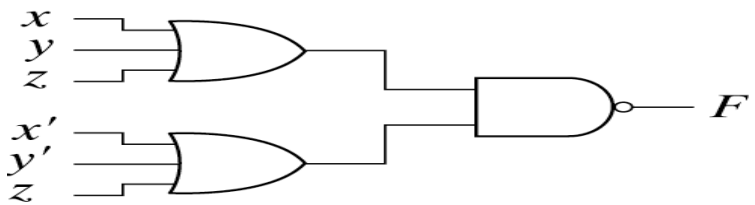


AND-NOR

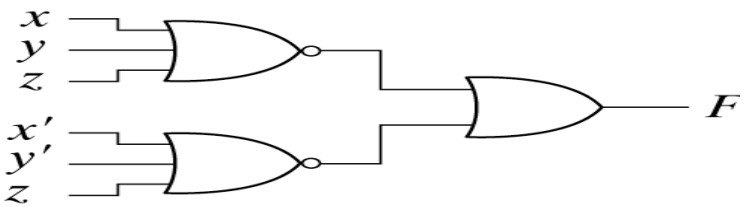


NAND-AND

(b) $F = (x'y + xy' + z)'$



OR-NAND



NOR-OR

(c) $F = [(x + y + z)(x' + y' + z)]'$

Other Two-level Implementations

Key Terms and Review Points

- Essential Prime Implicant
- Implicant
- Optional Prime Implicant
- Prime Implicant
- Minimum Cover
- NAND/NOR Implementation

- References: Dr. Karmouch and Dr. Groza ITI 1100 Slides