DIGITAL LOGIC

Chapter 3 part2: Two Level Implementation

2023 Fall

Today's Agenda

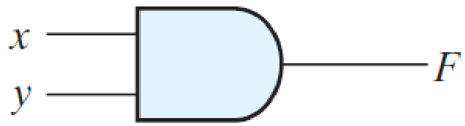
- Recap
- Context
  - NAND and NOR Implementation
  - Other Two-Level Implementations
  - Exclusive-OR Function
- Reading: Textbook, Chapter 3.6-3.9

Recap

- Lecture3
  - K-map
    - sum of minterms
      - for any boolean function
    - 2, 3, 4 variables
      - 2^n squares, gray code structure
  - Minimization
    - sum of product
      - group 1-minterms => F
    - product of sum
      - group 0-minterms => F'
      - then Demorgan's Law => F = (F')'
    - sum of minterms, ∑(2, 3, 4, 5)
      - 2, 3, 4, 5 are 1-minterms
    - product of maxterms ∏(0, 1, 6, 7)
      - 0, 1, 6, 7 are 0-minterms (Mi = mi')
  - Implicant
    - Prime Implicant (PI)
    - Essential Prime Implicant (EPI)
    - Grouping tips
      - EPI first, then PI
  - don't care
    - output is X, d(1,3,4)
      - 1, 3, 4 can be 0- or 1-minterm
      - grouped only if neccessary

Recall: Logic Gates

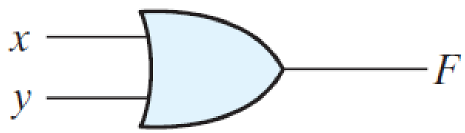
AND



$F = x \cdot y$

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

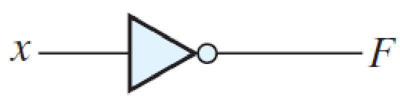
OR



$F = x + y$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

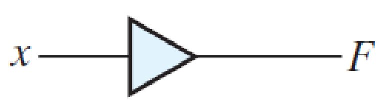
Inverter



$F = x'$

x	F
0	1
1	0

Buffer

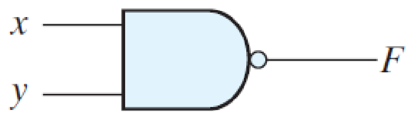


$F = x$

x	F
0	0
1	1

Recall: Logic Gates

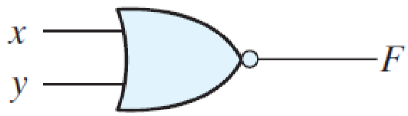
NAND



$F = (xy)'$

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

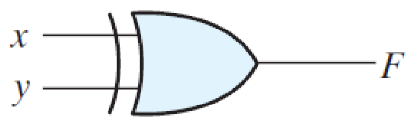
NOR



$F = (x + y)'$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

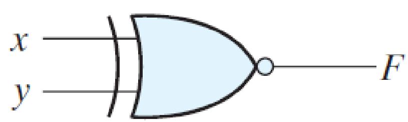
Exclusive-OR (XOR)



$$F = xy' + x'y = x \oplus y$$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR or equivalence



$$F = xy + x'y' = (x \oplus y)'$$

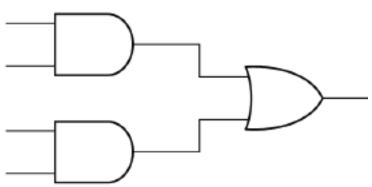
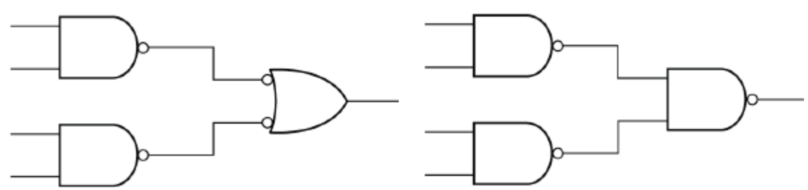
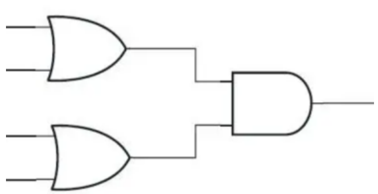
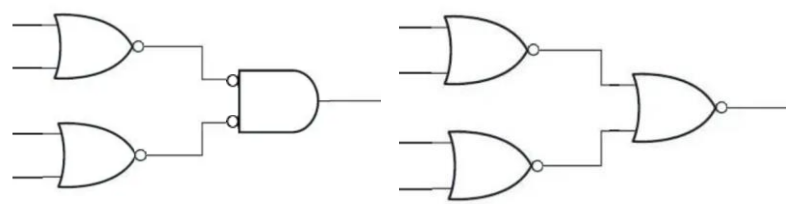
x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

Outline

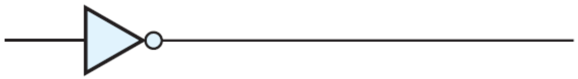
- **NAND Implementation**
- NOR Implementation
- Other Two-Level Implementations
- Exclusive-OR Function

Universal Gates

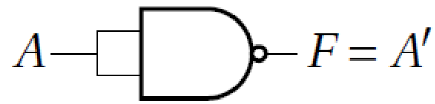
- NAND gates and NOR gates are called universal gates or universal building blocks.
  - Any type of gates or logic functions can be implemented by these gates.
  - NAND and NOR gates are easier to fabricate thus are requently used.

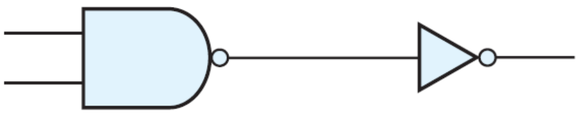
	Standard form	Universal Gate implementation
Sum-of-products	AND-OR 	NAND-NAND 
Product-of-sums	OR-AND 	NOR-NOR 

NAND circuits

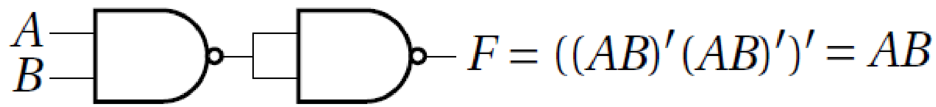


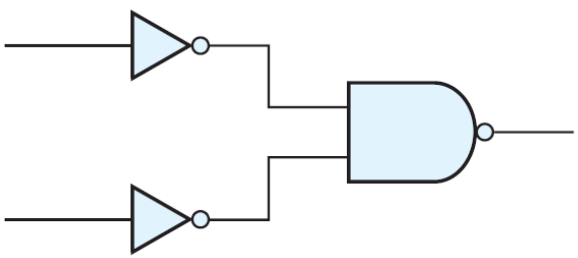
Inverter



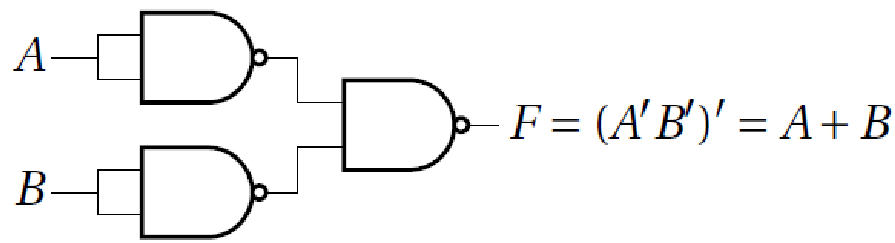


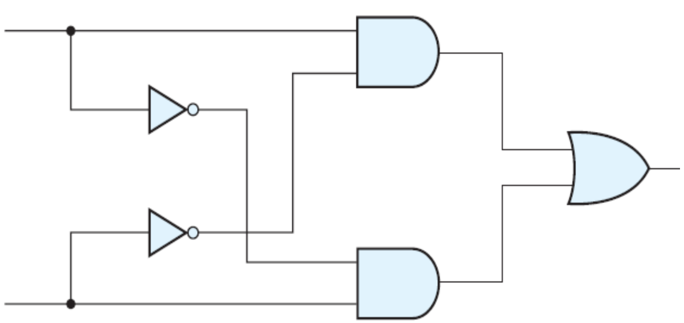
AND



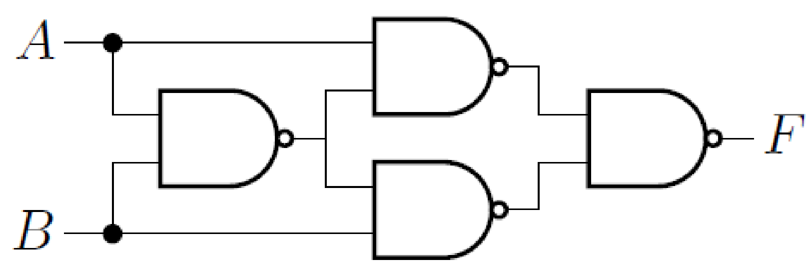


OR





XOR

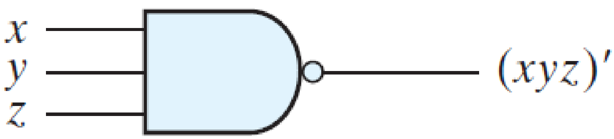


$$F = ((A(AB)')'((AB)'B)')'$$
$$= A(AB)' + (AB)'B = AB' + A'B = A \oplus B$$

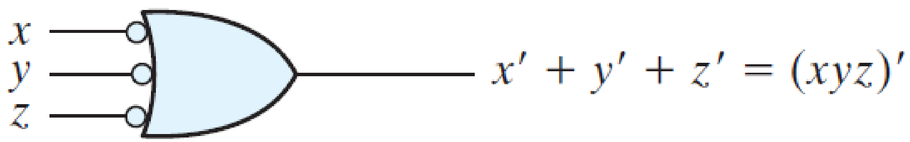
hint: $AB' + A'B = AB' + A'B + AA' + BB'$

NAND circuits

- To facilitate the conversion to NAND logic, it is convenient to define an alternative graphic symbol for the gate.
- AND-invert and Invert-OR are both NAND gates



(a) AND-invert



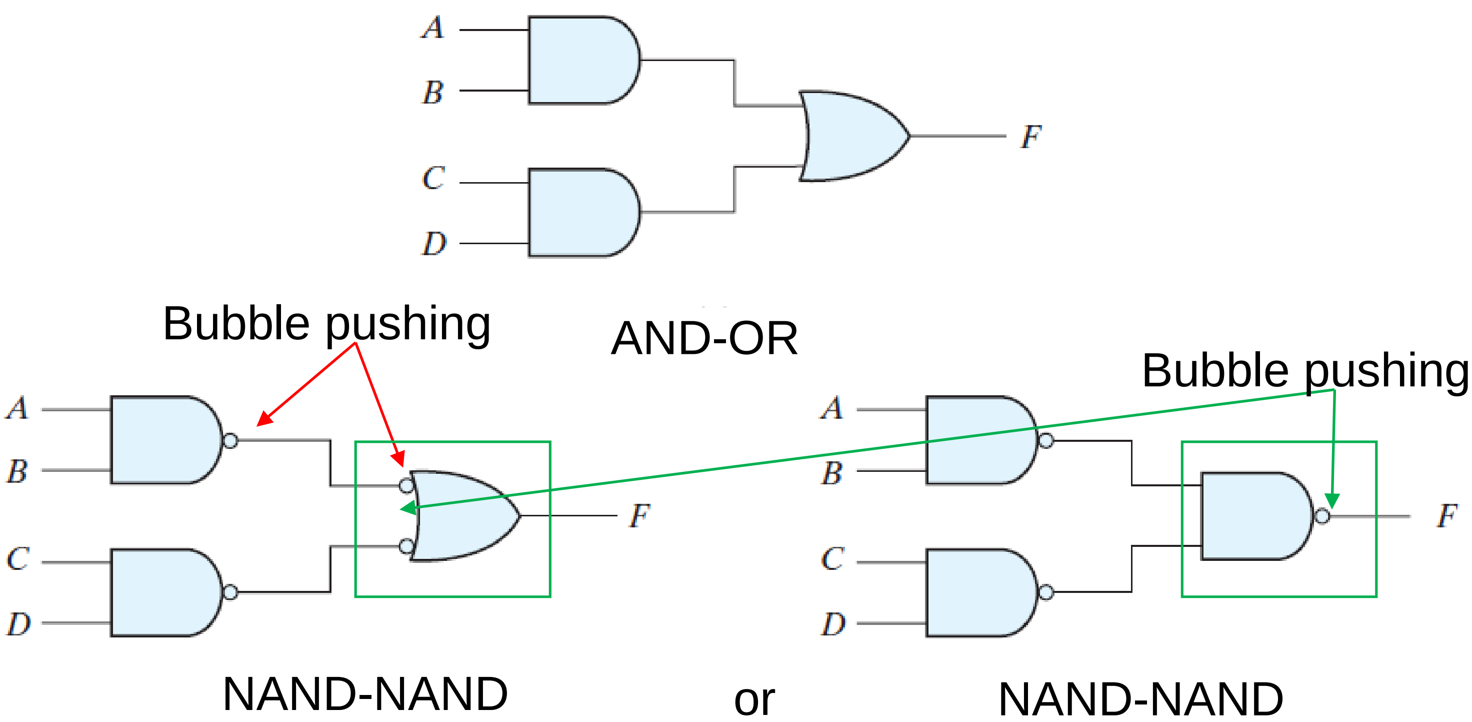
(b) Invert-OR

NAND-NAND Implementation

- A Boolean function can be implemented with two levels of NAND gates
  1. Starting point → Simplify the function in the form of **sum-of-products** (AND-OR circuit).
  2. Transfer it to 2-level NAND-NAND expression.
     - algebraically (**DeMorgan's Law**)
     - or graphically (**Bubble pushing**)
  3. Draw the corresponding NAND gate implementation. A 1-input NAND gate can be replaced by an inverter.

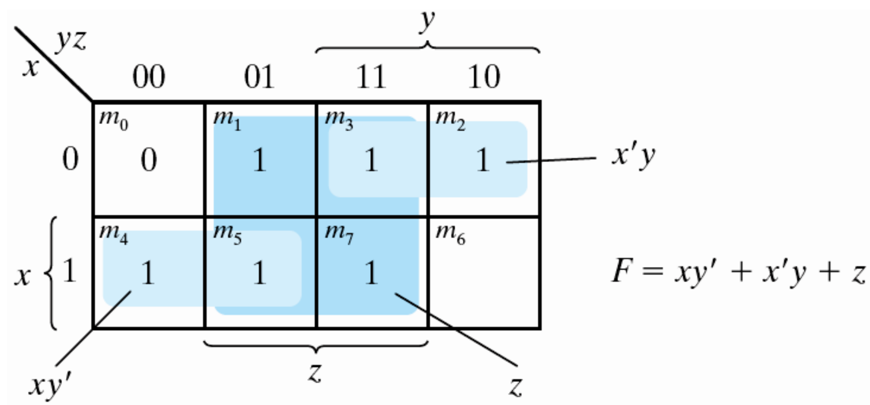
NAND-NAND Example1

- F(A,B,C,D) = AB + CD
  - Starting point: sum of products form → done
  - F = AB+CD = ((AB+CD)’)’ = ((AB)' (CD)’)’ →DeMorgan’s
  - Implementations: AND-OR / NAND-NAND (AND-Inv / Inv-OR)

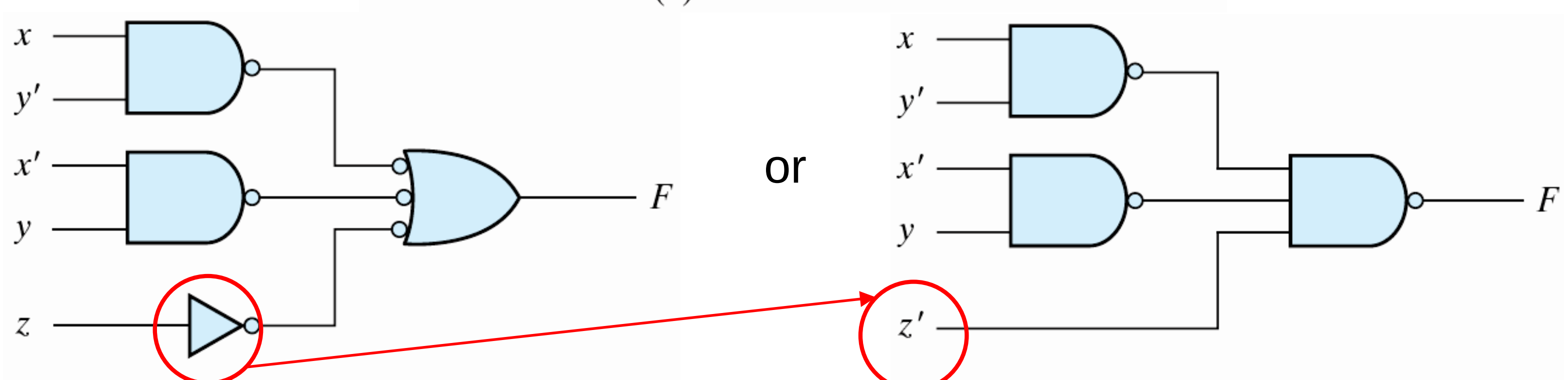


NAND-NAND Example2

- Example: Implement the following Boolean function with NAND gates
- F(x,y,z) = ∑(1,2,3,4,5,7)



(a)



(b) (c)

NAND-NAND Example3

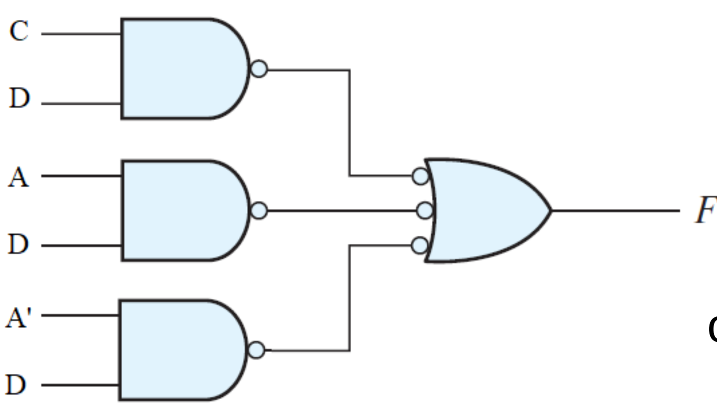
- Exercise: Implement the following Boolean function with NAND gates
- $F(A,B,C,D) = A'B'C'D+CD+AC'D$

NAND-NAND Example3

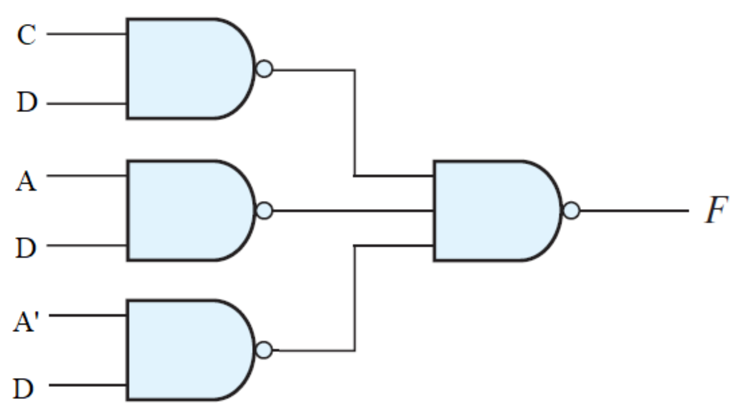
- $F(A,B,C,D) = A'B'C'D+CD+AC'D$
  $= A'B'C'D+(A+A')(B+B')CD+A(B+B')C'D$
  $=A'B'C'D+ABCD+AB'CD+A'BCD+A'B'CD+ABC'D+AB'C'D$
  $=\sum(1,3,7,9,11,13,15)$
- $F = CD+AD+A'D$
  $=[(CD+AD+A'D)']' = [(CD)'(AD)'(A'D)']'$

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

C B A D



or



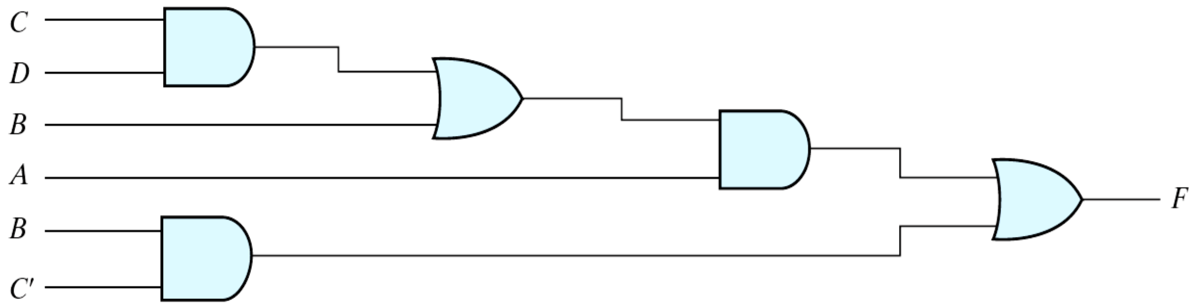
Multilevel NAND Implementation

- Multilevel-NAND circuits conversion procedure
  - Convert all AND to NAND with AND-Invert graphic symbols
  - Convert all OR to NAND with Invert-OR graphic symbols
  - Check all the bubbles (inverter) in the diagram and insert possible inverter to keep the original function

- Example: F(A,B,C,D) = A(CD+B)+BC'
  - AND-OR logic → NAND-NAND logic
    - For every bubble that is not compensated by another small circle along the same line, insert an inverter.

AND → AND + inverter

OR: inverter + OR = NAND



(a) AND–OR gates

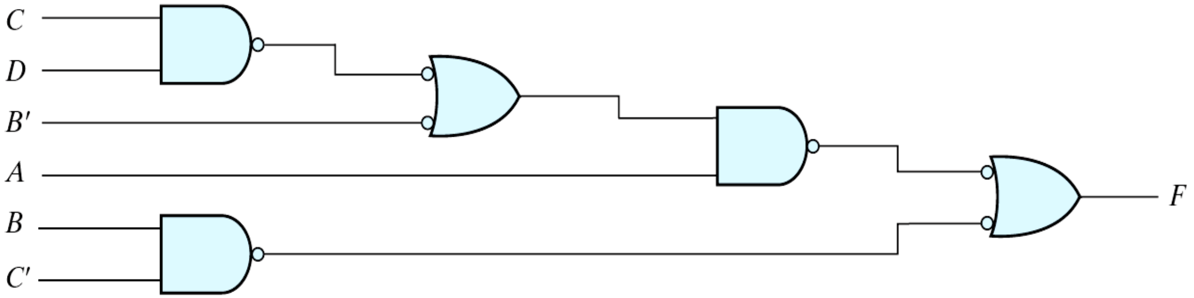
Multilevel NAND Implementation

- Multilevel-NAND circuits conversion procedure
  - Convert all AND to NAND with AND-Invert graphic symbols
  - Convert all OR to NAND with Invert-OR graphic symbols
  - Check all the bubbles (inverter) in the diagram and insert possible inverter to keep the original function

- Example: F(A,B,C,D) = A(CD+B)+BC’
  - AND-OR logic → NAND-NAND logic
    - For every bubble that is not compensated by another small circle along the same line, insert an inverter.

AND → AND + inverter

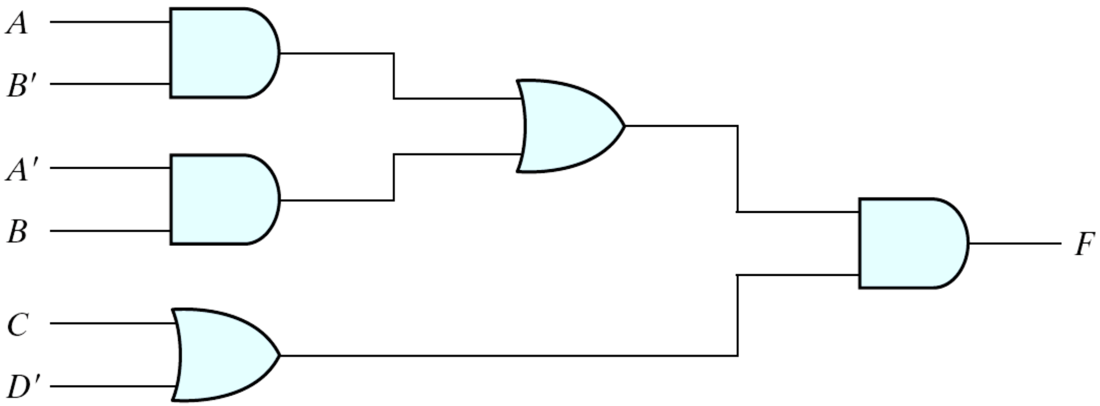
OR: inverter + OR = NAND



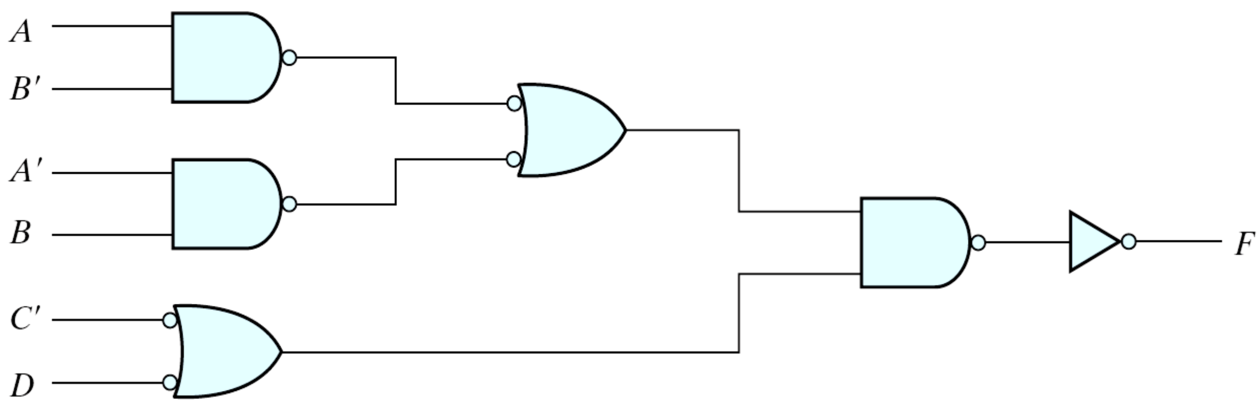
(b) NAND gates

Multilevel NAND Implementation

- Exercise: Implementing F = (AB′ +A′B)(C+ D′)



(a) AND–OR gates



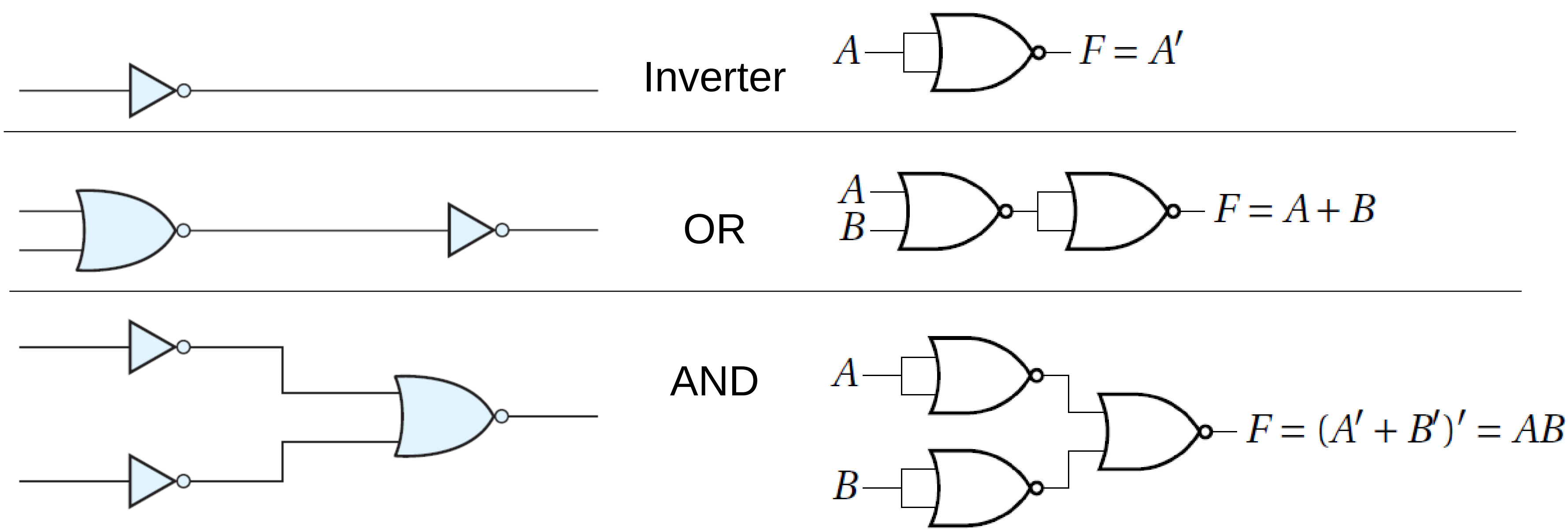
(b) NAND gates

Outline

- NAND Implementation
- **NOR Implementation**
- Other Two-Level Implementations
- Exclusive-OR Function

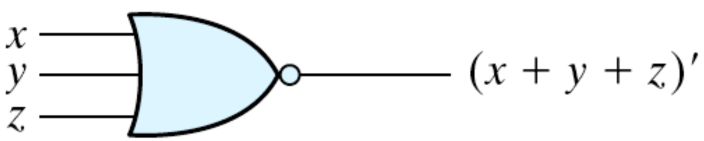
NOR-NOR Implementation

- NOR-NOR is the dual of the NAND-NAND implementation
  - All procedures and rules for NOR logic are the duals of the corresponding which developed for NAND logic.

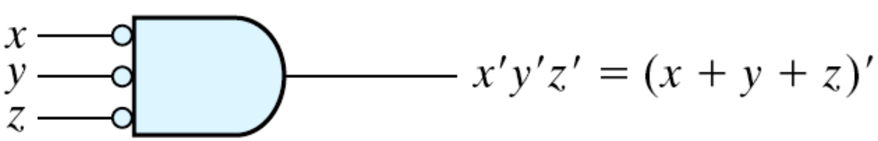


NOR-NOR Implementation

- To facilitate the conversion to NOR logic, it is convenient to define an alternative graphic symbol for the gate.



(a) OR-invert



(b) Invert-AND

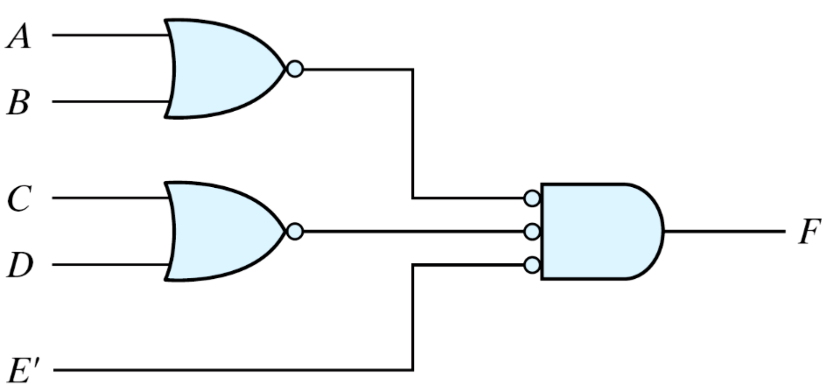
NOR-NOR Implementation

- Procedure of NOR-NOR implementation
  - Starting point → Simplify the function in the form of **product-of-sum** (OR-AND circuit).
  - Transfer it to 2-level NOR-NOR expression.
    - algebraically (**DeMorgan's Law**)
    - or, graphically (**Bubble pushing**)
  - Draw the corresponding NOR gate implementation. A 1-input NOR gate can be replaced by an inverter.

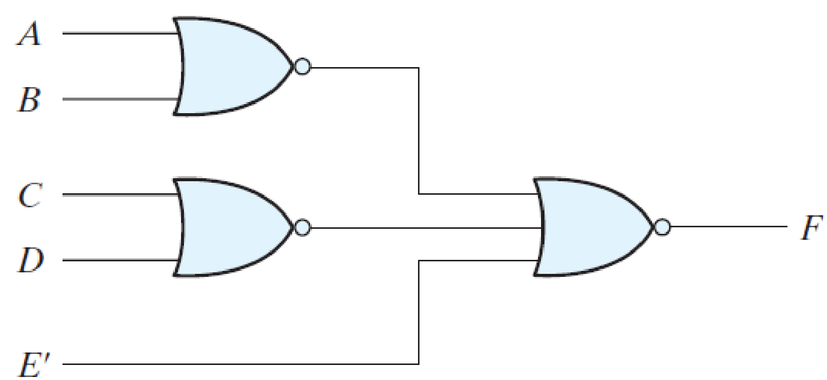
sum-of-product (AND-OR) => NAND-NAND
product-of-sum (OR-AND) => NOR-NOR

NOR-NOR Example1

- Example: Implement the following Boolean function with NAND gates
- F = (A + B)(C + D)E
  - Starting point: product of sums form → done
  - F = (A+B)(C+D)E = ((A+B)(C+D)E)’)’
  - = ((A+B)’+(C+D)’+E’)’→DeMorgan’s
  - Implementations:



or



NOR-NOR Example2

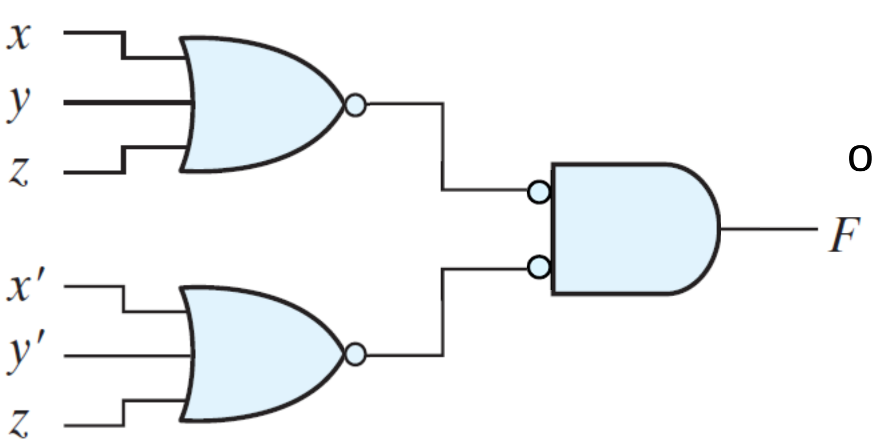
- Example $F(x,y,z) = \sum(1,2,3,4,5,7)$

$x \backslash yz$	00	01	11	10
0	m_0 0	m_1 1	m_3 1	m_2 1
1	m_4 1	m_5 1	m_7 1	m_6 0

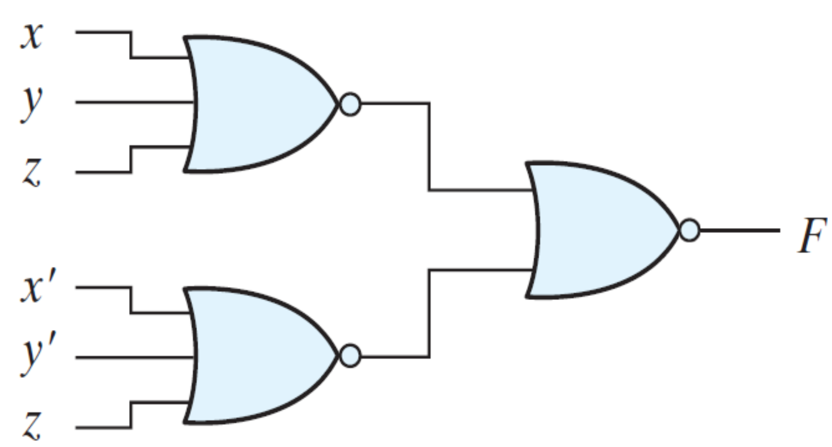
$F' = x'y'z' + xyz'$

$F = (F')' = (x'y'z' + xyz')'$

$\quad = ((x+y+z)' + (x'+y'+z))'$

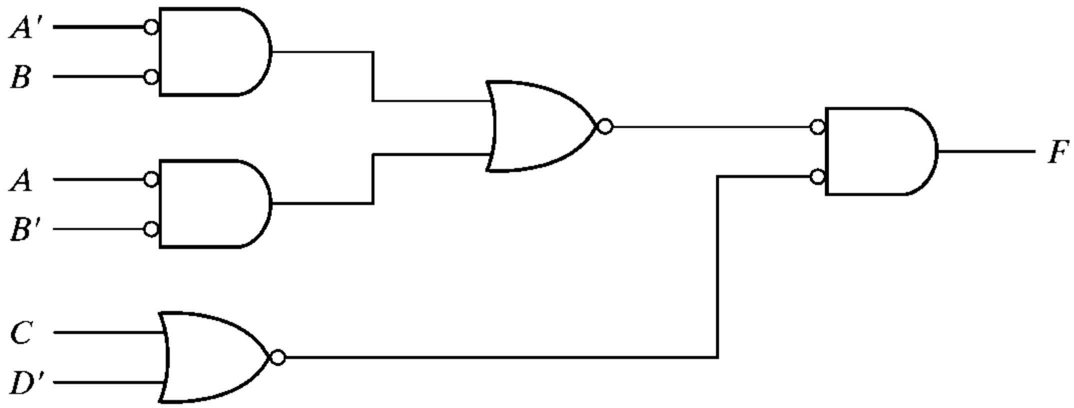


or

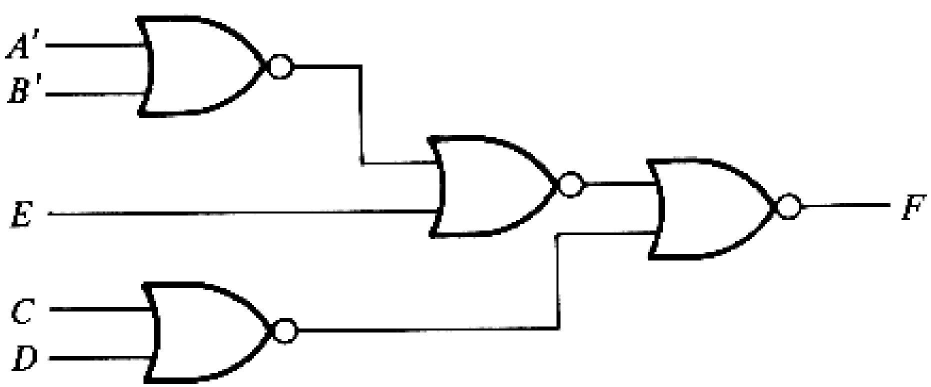


Multilevel NOR Implementation

- Change the OR gates to NOR gates with OR-invert graphic symbols and the AND gate to a NOR gate with an invert-AND graphic symbol.
- Example:
- F = (AB’ +A’B)(C + D')

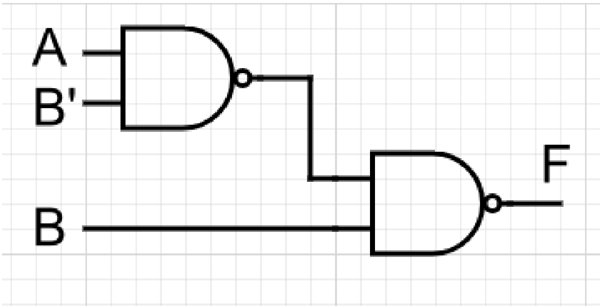


- F =(AB+E)(C+D)



Quiz

1. Simplify the Boolean function
   F(x,y,z) = x'y + yz + x’y’z’ into
   sum of product form, using map method.
2. Simplify the previous function into product of sum form.
3. True/False: any logic circuit can be implemented with NOR gates
4. Which of the following Boolean function is correct for the logic diagram?
   a) 1
   b) 0
   c) B’
   d) A



5. Simplify the following Boolean function F, together with the don’t-care conditions d. F(A,B,C,D) = ∑(0,6,8,12,14), d(A,B,C,D) = ∑(2,4,10)

Outline

- NAND Implementation
- NOR Implementation
- **Other Two-Level Implementations**
- Exclusive-OR Function

Two-Level Forms

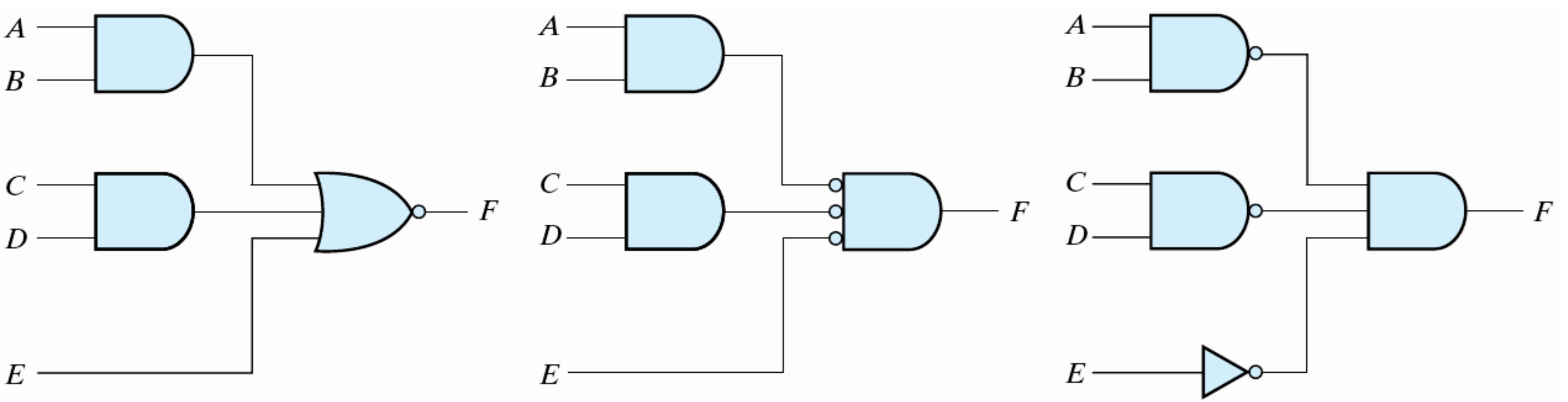
- AND/NAND/OR/NOR have 16 possible combinations of two-level forms
- Eight of them **degenerate** to a single operation
  - AND-AND => AND
  - OR-OR => OR
  - AND-NAND => NAND
  - OR-NOR => NOR
  - NAND-NOR =>AND
  - NOR-NAND => OR
  - NAND-OR => NAND
  - NOR-AND => NOR

Two-Level Forms

- Eight are **non-degenerate** forms
- AND-OR => standard sum-of-products
- NAND-NAND => standard sum-of-products
- OR-AND => standard product-of-sums
- NOR-NOR => standard product-of-sums
- NAND-AND/AND-NOR => AND-OR-INVERT (AOI)
  - **complement** of sum-of-products
- OR-NAND/NOR-OR => OR-AND-INVERT (OAI)
  - **complement** of product-of-sums

AND-OR-Invert Implementation

- NAND-AND = AND-NOR = AOI
  - F(A,B,C,D,E)=(AB+CD+E)’
  - F’(A,B,C,D,E)=AB+CD+E (sum of products)
  - An AND–OR implementation requires an expression in sum-of-products form.
  - The AND–OR–INVERT implementation is similar, except for the inversion.

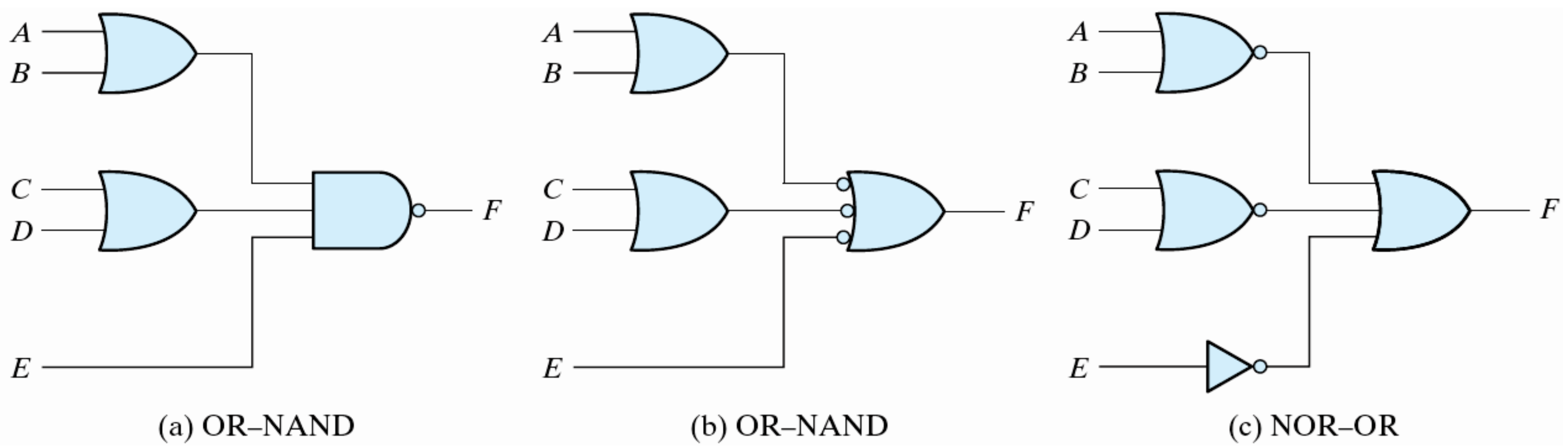


(a) AND–NOR (b) AND–NOR (c) NAND–AND

Combine 0's in K-map to simplify F' in product-of-sums and then invert the results

OR-AND-Invert Implementation

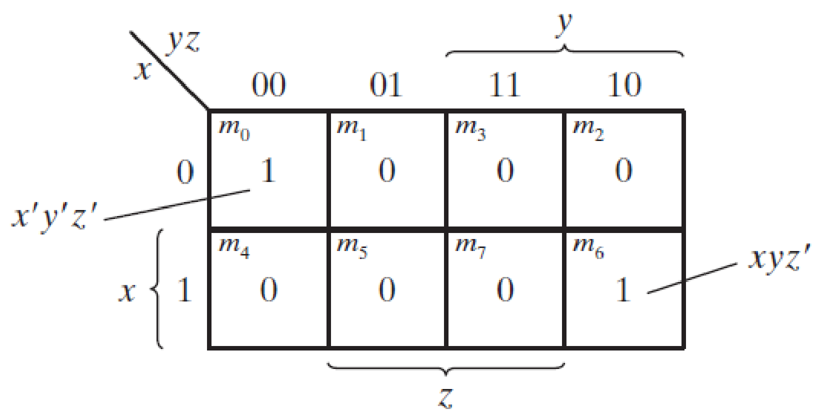
- OR-NAND = NOR-OR = OAI
  - F(A,B,C,D,E)=((A+B)(C+D)E)'
  - F' = (A+B)(C+D)E   (product of sums)
  - The AND–OR–INVERT implementation requires an expression in product-of-sums form.



Combine 1's in K-map to simplify F' in product-of-sums and then invert the results

AOI & OAI Example

- Example



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

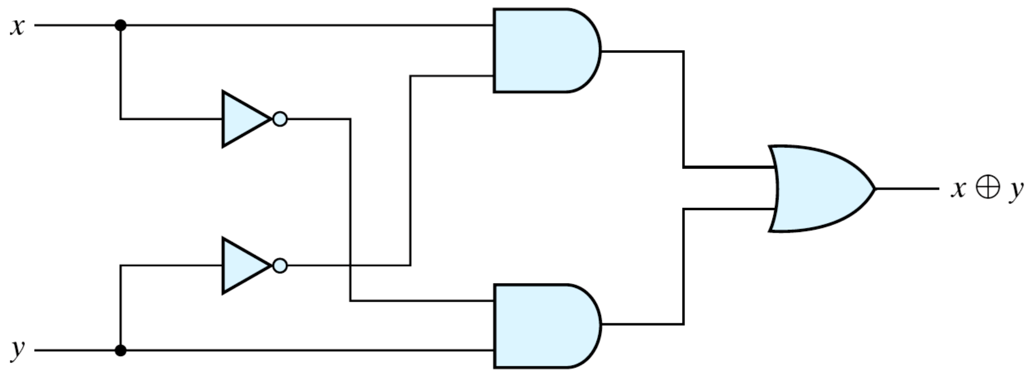
- AND-OR
  - F=x'y'z' + xyz'
- NAND-NAND
  - F=((x'y'z')'(xyz')')'
- OR-AND
  - F'=x'y+xy'+z → F=z'(x'+y)(x+y')
- NOR-NOR
  - F'=x'y+xy'+z → F=(z+(x'+y)'+(x+y')')'
- AOI
  - F'=x'y+xy'+z → F=(x'y+xy'+z)'
- OAI
  - F=x'y'z'+xyz' → F'=(x+y+z)(x'+y'+z) → F=((x+y+z)(x'+y'+z))'

Exclusive-OR Function

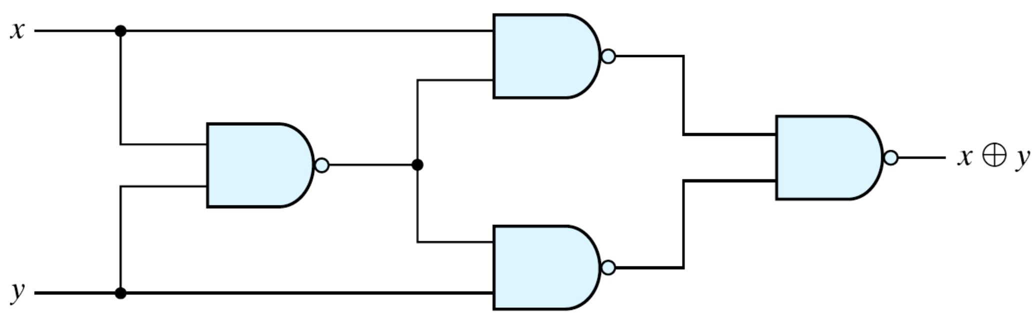
- Exclusive-OR (XOR)
  - $x \oplus y = xy' + x'y$
- Exclusive-NOR (XNOR or equivalency)
  - $(x \oplus y)' = xy + x'y'$
- Some identities
  - $x \oplus 0 = x$
  - $x \oplus 1 = x'$
  - $x \oplus x = 0$
  - $x \oplus x' = 1$
  - $x \oplus y' = x' \oplus y = (x \oplus y)'$
- Commutative and associative
  - $A \oplus B = B \oplus A$
  - $(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$

Exclusive-OR Implementations

- Implementations
  - (x'+y')x + (x'+y')y = xy'+x'y = x⊕y



(a) With AND–OR–NOT gates



(b) With NAND gates

Outline

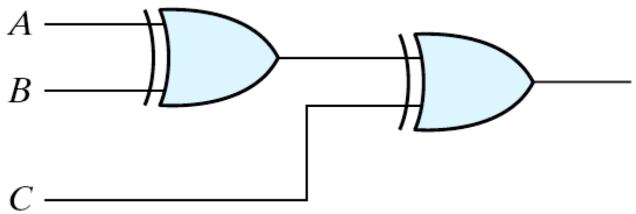
- NAND Implementation
- NOR Implementation
- Other Two-Level Implementations
- **Exclusive-OR Function**

Odd function

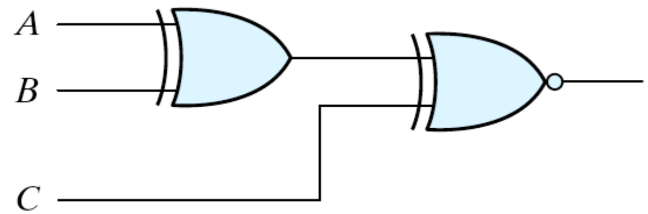
- The XOR operation with three or more variables can be converted into an ordinary Boolean function by replacing the $\oplus$ with its equivalent Boolean expression.
- $A \oplus B \oplus C = (AB' + A'B)C' + (AB + A'B')C$

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

  $= \sum(1, 2, 4, 7)$
- Odd function (XOR) → if **odd** number of variables are equal to 1, then F = 1.
- Even function (XNOR) → if **even** number of variables are equal to 1, then F = 1.



(a) 3-input odd function



(b) 3-input even function

Recall: Error-Detecting Code

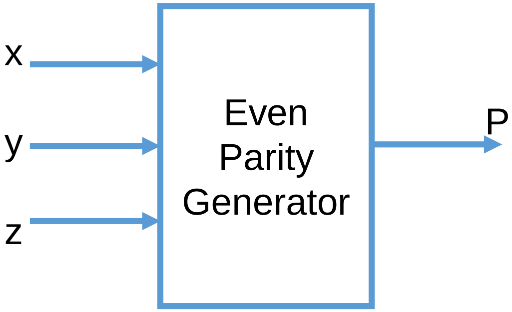
- Error-Detecting Code
  - An eighth bit is sometimes added to the ASCII character to indicate its parity.
  - A parity bit (校验位) is an extra bit included with a message to make the total number of 1's either even or odd.
- Example:

	With even parity	With odd parity
ASCII A = 1000001	$\boxed{0}$1000001	$\boxed{1}$1000001

Even-Parity-Generator Truth Table

Three-Bit Message			Parity Bit
x	*y*	*z*	*P*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

XOR functions can be used for parity generator and parity checker

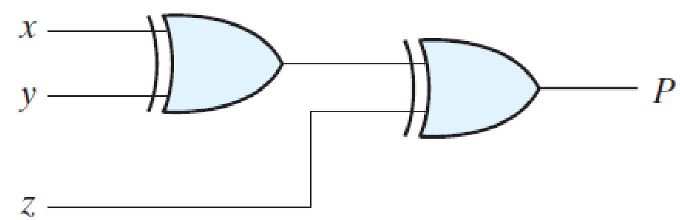


Parity Generation and Checking

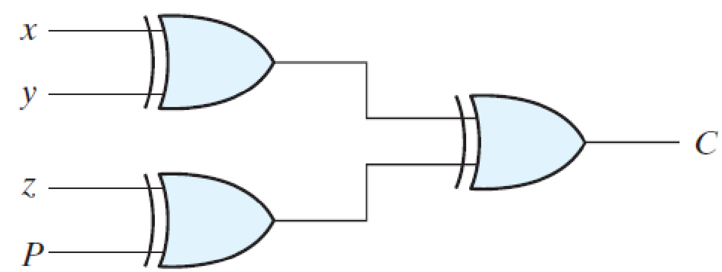
- $P = xy'z'+x'yz'+xyz+x'y'z$

  $= \sum(1, 2, 4, 7)$ – odd function

x	y	z	Parity bit p
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



even parity generator



even parity checker

- Parity Generation and Checking
  - A even parity bit: $P = x \oplus y \oplus z$
  - Even Parity check: $C = x \oplus y \oplus z \oplus P$
    - C=1: one bit error or an odd number of data bit error
    - C=0: correct or an even # of data bit error