DIGITAL DESIGN AND COMPUTER ORGANIZATION

Gate Level Minimization: NAND and NOR Implementation

Department of Computer Science and Engineering

DIGITAL DESIGN AND COMPUTER ORGANIZATION

NAND and NOR implementation

Department of Computer Science and Engineering

Gate-Level Minimization

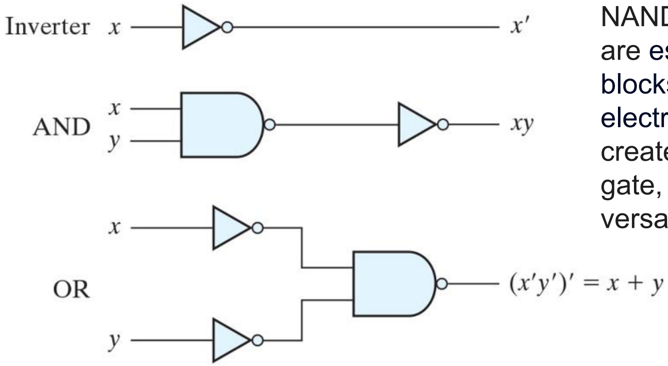
NAND and NOR implementation(Universal Gates)

- A convenient way to implement a Boolean function with NAND gates is to obtain the simplified Boolean function in terms of Boolean operators and then convert the function to NAND logic.

- The conversion of an algebraic expression from AND, OR, and complement to NAND can be done by simple circuit manipulation techniques that change AND–OR diagrams to NAND diagrams.

Gate-Level Minimization

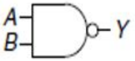
NAND and NOR implementation(T1-3.6)



Universal logic gates, NAND and NOR, are essential building blocks in digital electronics. They can create any other logic gate, making them highly versatile for circuit design

Logic operations with NAND gates.

NAND

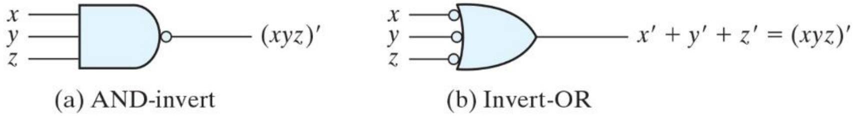


$Y = \overline{AB}$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Gate-Level Minimization

NAND and NOR implementation



Two graphic symbols for a **three-input NAND gate.**

Gate-Level Minimization
NAND and NOR implementation

- The AND-invert symbol has been defined previously and consists of an AND graphic symbol followed by a small circle negation indicator referred to as a bubble.

- It is possible to represent a NAND gate by an OR graphic symbol that is preceded by a bubble in each input.

- The **invert-OR symbol for the NAND gate follows DeMorgan’s theorem** and the convention that the negation indicator (bubble) denotes complementation.

Gate-Level Minimization
NAND and NOR implementation(2 level Implementation)

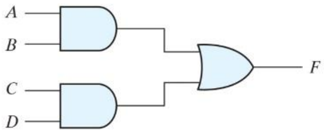
The implementation of Boolean functions with NAND gates requires that the functions be in sum-of-products form.

Implement the function F = AB + CD with

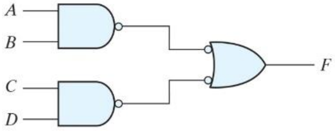
- And ,OR gate
- NAND gates only(AND-Invert only)
- AND -Invert and Invert-OR

Gate-Level Minimization

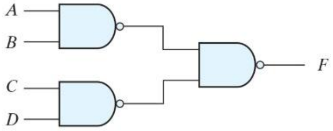
NAND and NOR implementation(2 level Implementation)



(a)



(b)



(c)

Three ways to implement $F = AB + CD$.

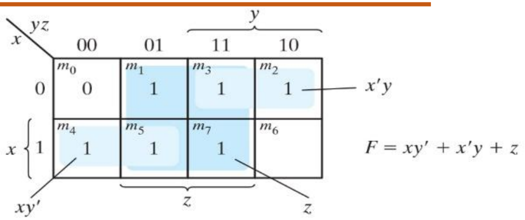
Gate-Level Minimization

NAND and NOR implementation(2 level Implementation)

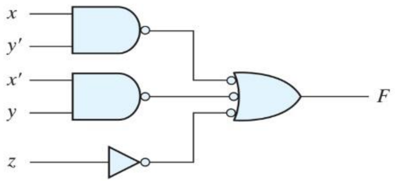
Implement the following Boolean function with NAND gates:

$F(x, y, z) = (1, 2, 3, 4, 5, 7)$

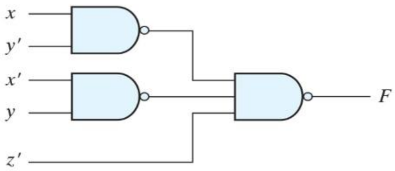
Gate-Level Minimization and Combinational logic NAND and NOR implementation



(a)



(b)



(c)

Solution to Solved Example

Gate-Level Minimization and Combinational logic NAND and NOR implementation

The procedure for obtaining the logic diagram from a Boolean function is as follows:

1. Simplify the function and express it in sum-of-products form.
2. Draw a NAND gate for each product term of the expression that has at least two literals. The inputs to each NAND gate are the literals of the term. This procedure produces a group of first-level gates.
3. Draw a single gate using the AND-invert or the invert-OR graphic symbol in the second level, with inputs coming from outputs of first-level gates.
4. A term with a single literal requires an inverter in the first level. However, if the single literal is complemented, it can be connected directly to an input of the second level NAND gate

Gate-Level Minimization and Combinational logi

NAND and NOR implementation (multilevel NAND gate)

The standard form of expressing Boolean functions results in a two-level implementation. There are occasions, however, when the design of **digital systems results in gating structures with three or more levels**

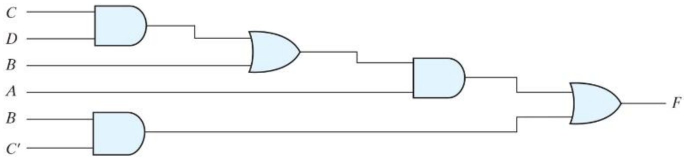
The most common procedure in the design of multilevel circuits is to express the Boolean function in terms of AND, OR, and complement operations. The function can then be implemented with AND and OR gates. After that, if necessary, it can be converted into an all-NAND circuit.

Implementing $F = A(CD + B) + BC'$ using NAND gate(AND invert, Invert OR) . Implement the same using AND-OR gates

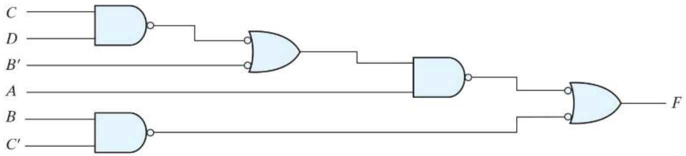
Draw the Circuit .

Gate-Level Minimization

NAND and NOR implementation(multilevel NAND gate)



(a) AND–OR gates



(b) NAND gates

Implementing $F = A(CD + B) + BC'$.

Gate-Level Minimization and Combinational logi

NAND and NOR implementation(multilevel NAND gate)

Although it is possible to remove the parentheses and reduce the expression into a standard sum-of-products form, we choose to implement it as a multilevel circuit for illustration

The general procedure for converting a multilevel AND–OR diagram into an all-NAND diagram using mixed notation is as follows:

1. Convert all AND gates to NAND gates with AND-invert graphic symbols.

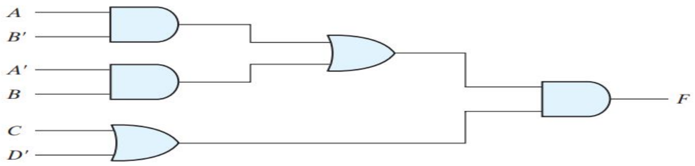
2. Convert all OR gates to NAND gates with invert-OR graphic symbols.

3. Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter (a one-input NAND gate) or complement the input literal.

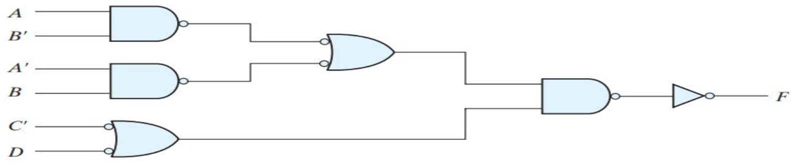
Implement F = (AB` + A`B)(C + D`) using AND-OR gates, NAND gates

Gate-Level Minimization and Combinational logi

NAND and NOR implementation(multilevel NAND gate)



(a) AND–OR gates



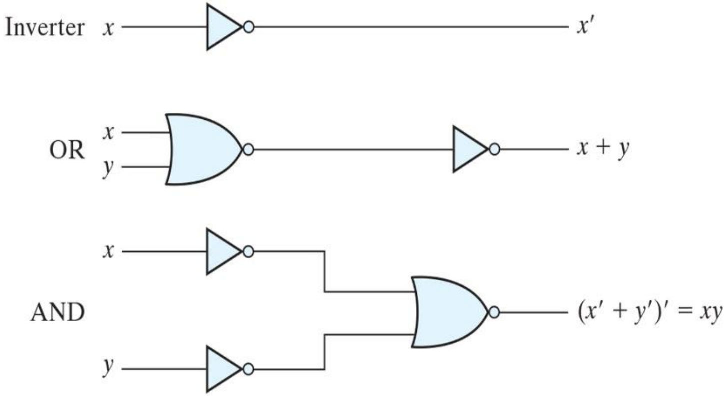
(b) NAND gates

FIGURE 3.21
Implementing $F = (AB' + A'B)(C + D')$

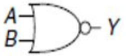
Gate-Level Minimization and Combinational logic NOR implementation

The NOR operation is the dual of the NAND operation. Therefore, all procedures and rules for NOR logic are the **duals** of the corresponding procedures and rules developed for NAND logic. The NOR gate is another universal gate that can be used to implement any Boolean function

Gate-Level Minimization and Combinational logic NOR implementation



NOR



$Y = \overline{A+B}$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Logic operations with NOR gates.

Gate-Level Minimization and Combinational logic NOR implementation

Logic operations with NOR gates

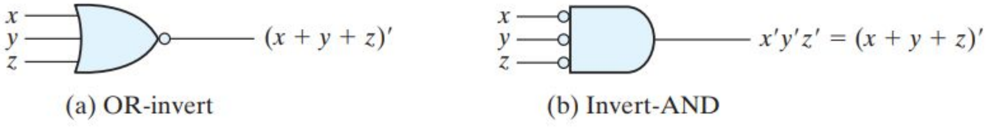


FIGURE 3.23
Two graphic symbols for the NOR gate

The procedure for converting a multilevel AND–OR diagram to an all-NOR diagram. we must convert each OR gate to an OR-invert symbol and each AND gate to an invert-AND symbol

Gate-Level Minimization and Combinational logic NOR implementation

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

Implementing F = (A\`B + AB\`)(C + D\`) with NOR gates

Gate-Level Minimization and Combinational logic NAND and NOR implementation

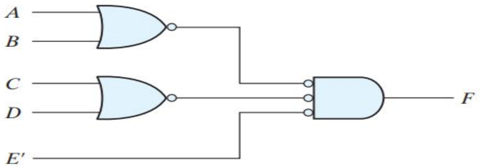


FIGURE 3.24
Implementing $F = (A + B)(C + D)E$

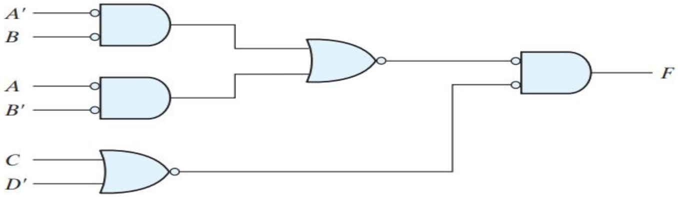


FIGURE 3.25
Implementing $F = (AB' + A'B)(C + D')$ with NOR gates

Exclusive-OR Function (T1: 3.8)

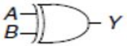
The exclusive-OR (XOR), denoted by the symbol $\oplus$, is a logical operation that performs the following Boolean operation:

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

The exclusive-OR is equal to 1 if only x is equal to 1 or if only y is equal to 1 (i.e., x and y differ in value), but not when both are equal to 1 or when both are equal to 0. The exclusive-NOR, also known as equivalence, performs the following Boolean operation:

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

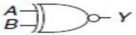
XOR



$Y = A \oplus B$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

XNOR



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

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Exclusive-OR Function

The following identities apply to the exclusive-OR operation:

$$x \oplus 0 = x$$
$$x \oplus 1 = x'$$
$$x \oplus x = 0$$
$$x \oplus x' = 1$$
$$x \oplus y' = x' \oplus y = (x \oplus y)'$$

Any of these identities can be proven with a truth table or by replacing the $\oplus$ operation by its equivalent Boolean expression. Also, it can be shown that the exclusive-OR operation is both commutative and associative; that is,

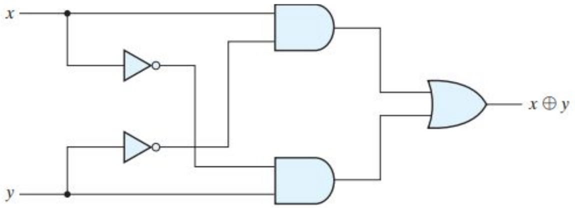
$$A \oplus B = B \oplus A$$

and

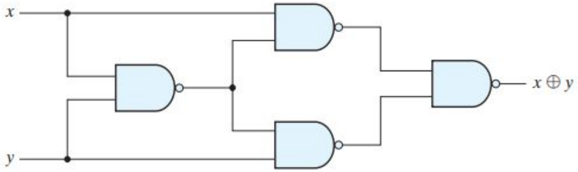
$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$$

Exclusive-OR Implementation

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



(a) Exclusive-OR with AND–OR–NOT gates

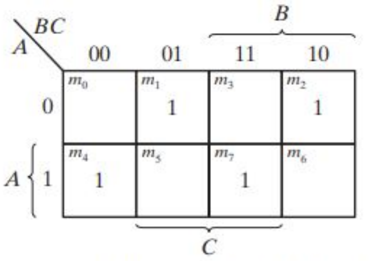


(b) Exclusive-OR with NAND gates

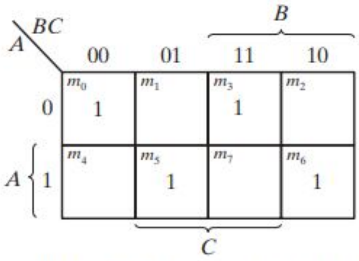
Odd function - XOR for 3 variables

- Multiple-variable exclusive-OR operation is defined as an odd function.
- XOR – 3 variables: A XOR B XOR C
- The Boolean expression clearly indicates that the three-variable exclusive-OR function is equal to 1 if only one variable is equal to 1 or if all three variables are equal to 1.
- Multiple variable XOR – ODD Function: The odd function is identified from the four minterms whose binary values have an odd number of 1's- ODD parity.
- [illegible]

Odd function - XOR for 3 variables



(a) Odd function $F = A \oplus B \oplus C$

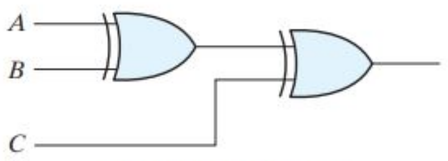


(b) Even function $F = (A \oplus B \oplus C)'$

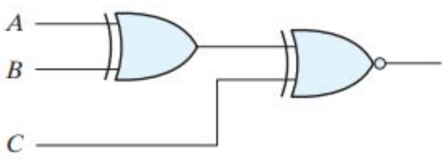
FIGURE 3.31

Map for a three-variable exclusive-OR function

Odd function - XOR for 3 variables



(a) 3-input odd function



(b) 3-input even function

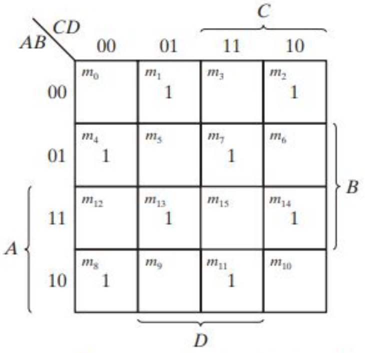
FIGURE 3.32
Logic diagram of odd and even functions

Odd function - XOR for 4 variables

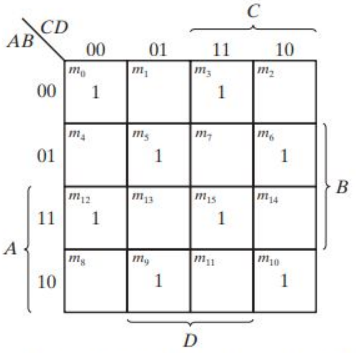
- There are 16 minterms for a four-variable Boolean function.
- Half of the minterms have binary numerical values with an odd number of 1's;
- The other half of the minterms have binary numerical values with an even number of 1's

Odd function - XOR for 4 variables

$$\begin{aligned} A \oplus B \oplus C \oplus D &= (AB' + A'B) \oplus (CD' + C'D) \\ &= (AB' + A'B)(CD + C'D') + (AB + A'B')(CD' + C'D) \\ &= \Sigma(1, 2, 4, 7, 8, 11, 13, 14) \end{aligned}$$



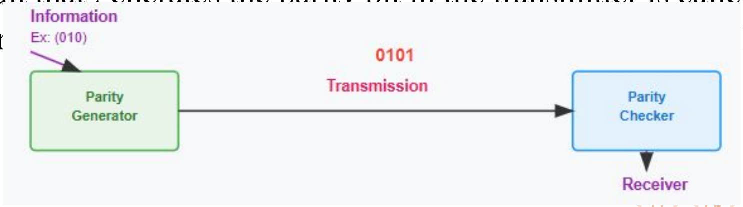
(a) Odd function $F = A \oplus B \oplus C \oplus D$



(b) Even function $F = (A \oplus B \oplus C \oplus D)'$

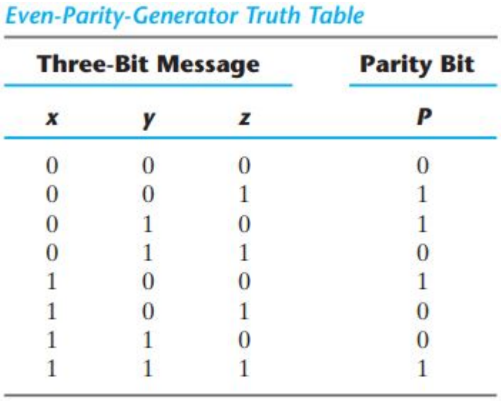
Parity Generation and Checking

- Exclusive-OR functions are very useful in systems requiring error detection and correction.
- A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even. The message, including the parity bit, is transmitted and then checked at the receiving end for errors.
- The circuit that generates the parity bit in the transmitter is called a parity generator led a parity checker.



Parity Generation and Checking

Parity bit P must be generated to make the number of 1's even if even parity is considered.



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

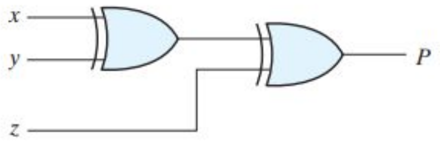
Parity Generation and Checking

- A parity bit is an extra bit included with a binary message to make the number of 1’s either odd or even. The message, including the parity bit, is transmitted and then checked at the receiving end for errors

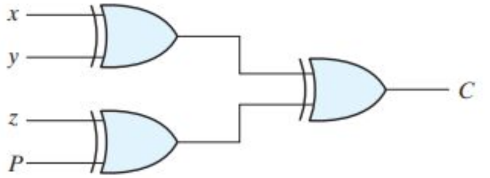
- An error is detected if the checked parity does not correspond with the one transmitted. The circuit that generates the parity bit in the transmitter is called a parity generator. The circuit that checks the parity in the receiver is called a parity checker.

Parity Generation and Checking

Logic diagram of a parity generator and checker.



(a) 3-bit even parity generator

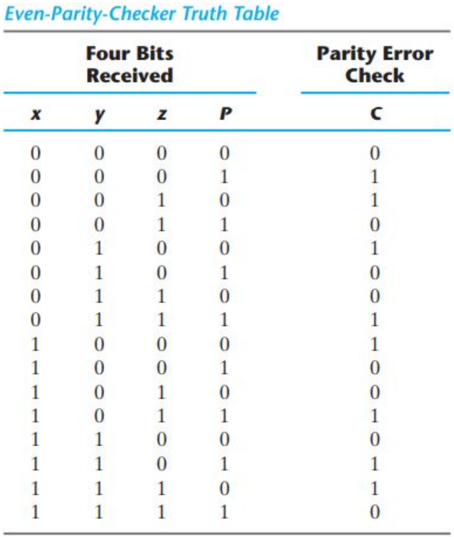


(b) 4-bit even parity checker

Parity Generation and Checking

Parity check
A XOR B XOR C XOR P

Advantage- use same gates for parity generator and checker



Even-Parity-Checker Truth Table

Four Bits Received				Parity Error Check
x	*y*	*z*	*P*	*C*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Applications

- Alarm Systems

Scenario:
Industrial machines often have safety alarms that trigger if certain unsafe conditions occur (e.g., overheat, high pressure, door open).

Why NAND/NOR?

Safety logic can be built with a single type of universal gate:

- NAND can be configured to act as OR to trigger alarms when any unsafe condition is present.
- NAND can also be configured to act as AND to ensure alarms only trigger if multiple sensor fail simultaneously.

Applications

Processor Design (Control Logic)

Scenario:In CPUs, the control unit decides which micro-operation to perform based on opcode and status flags.

Why NAND/NOR?
At the hardware level, all combinational control logic can be implemented using only NAND or only NOR gates — reducing design complexity in IC fabrication.

THANK YOU

Team DDCO

Department of Computer Science