Ollscoil na Gaillimhe
University of Galway

CT101 Computing Systems

Dr. Bharathi Raja Chakravarthi
Lecturer-above-the-bar
Email: bharathi.raja@universityofgalway.ie

University *of* Galway.ie

Ollscoil na Gaillimhe
University of Galway

NAND and NOR Implementation

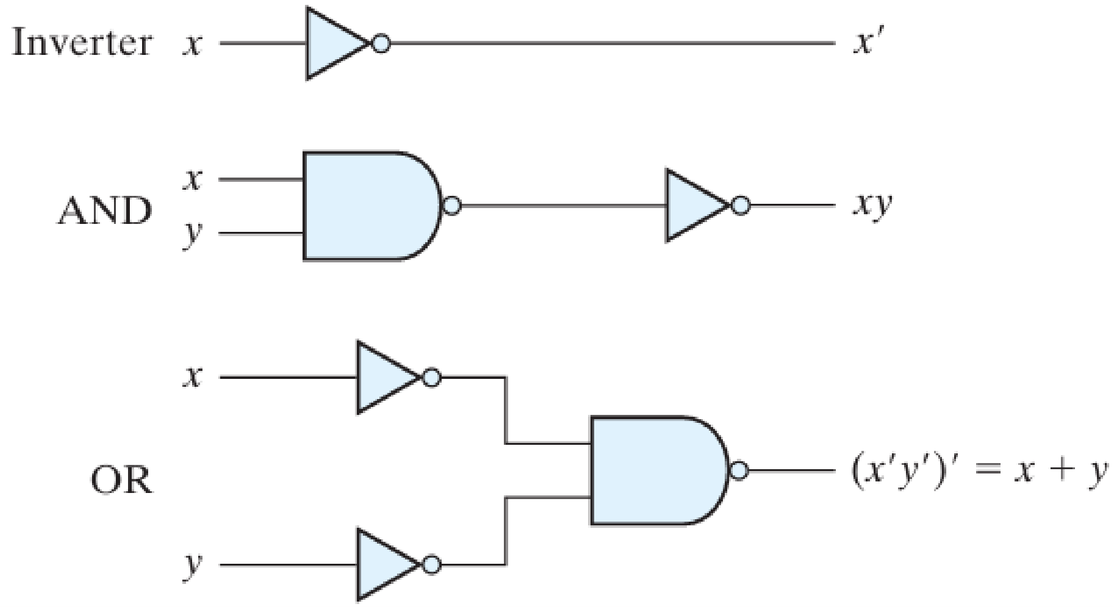
University *of* Galway.ie

Introduction

- Digital circuits are frequently constructed with NAND or NOR gates rather than with AND and OR gates.
- NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families.
- Rules and procedures have been developed for the conversion from Boolean functions given in terms of **AND, OR, and NOT** into equivalent **NAND and NOR** logic diagrams.

NAND Circuits

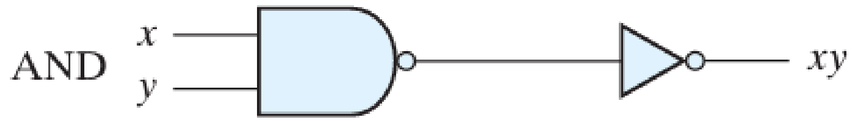
- The NAND gate is said to be a **universal gate** because any logic circuit can be implemented with it.
- To show that any Boolean function can be implemented with NAND gates, we need only show that the logical operations of AND, OR, and complement can be obtained with NAND gates alone.
- This is indeed shown in Figure.



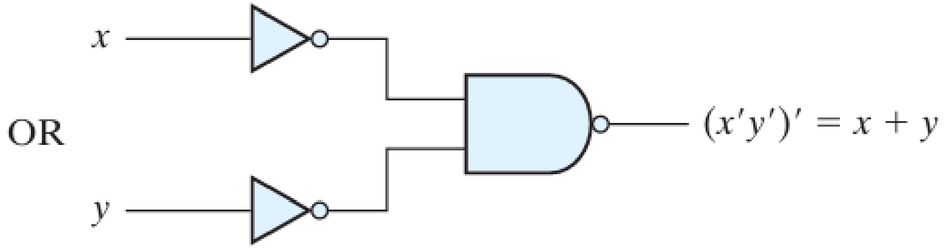
NAND Circuits

- The complement operation is obtained from a one-input NAND gate that behaves exactly like an inverter.

- The AND operation requires two NAND gates.
- The first produces the NAND operation and the second inverts the logical sense of the signal.



- The OR operation is achieved through a NAND gate with additional inverters in each input.



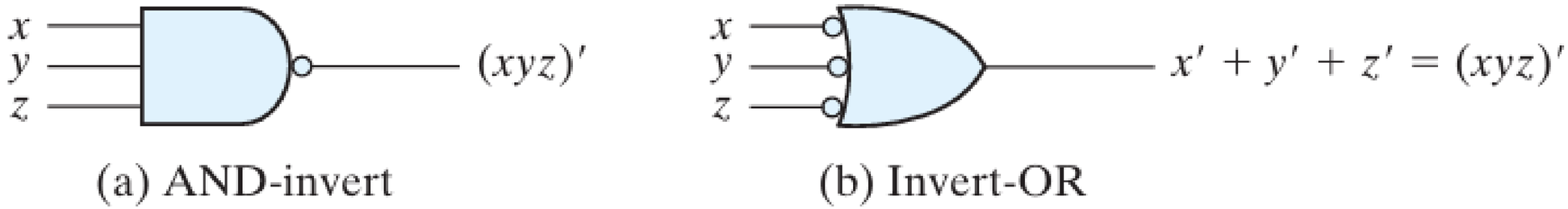
NAND Circuits

- 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.

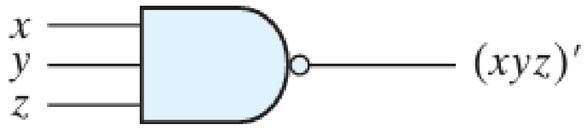
NAND Circuits

- To facilitate the conversion to NAND logic, defining an **alternative graphic symbol** for the gate is convenient.
- Two equivalent graphic symbols for the NAND gate are shown in Figure.



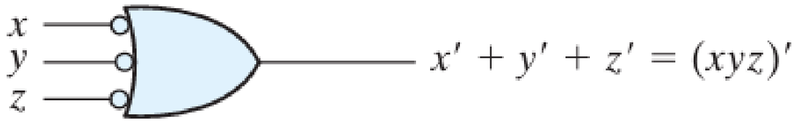
(a) AND-invert (b) Invert-OR

NAND Circuits



(a) AND-invert

- 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.



(b) Invert-OR

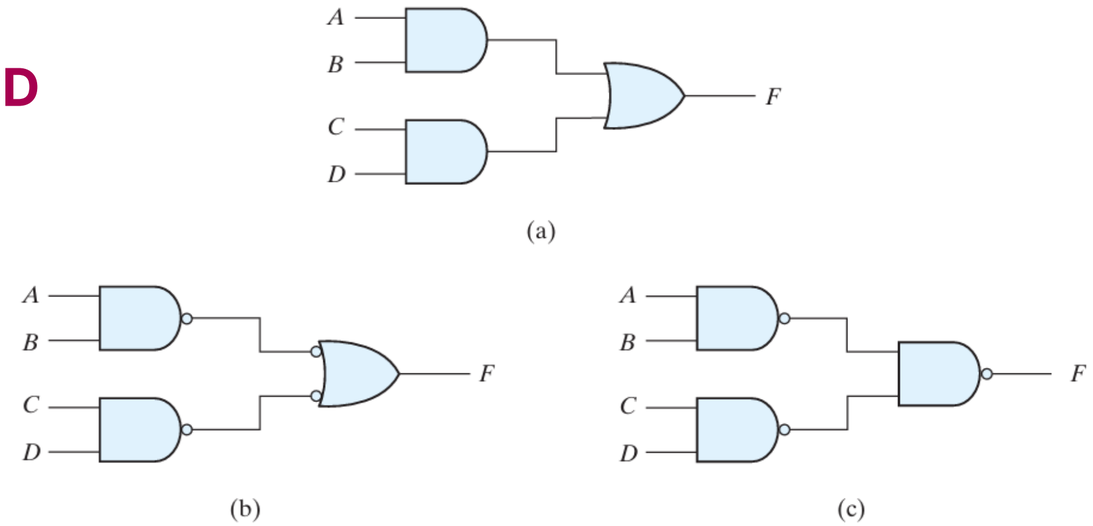
- The **invert-OR symbol** for the NAND gate follows DeMorgan's theorem and the convention that the negation indicator (bubble) denotes complementation.

- The two graphic symbols' representations are useful in analyzing and designing NAND circuits.
- When both symbols are mixed in the same diagram, the circuit is said to be in mixed notation.

Two-Level Implementation

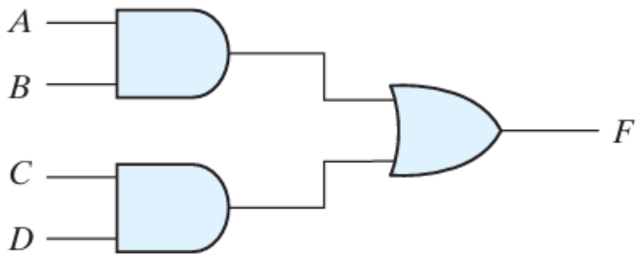
- The implementation of Boolean functions with NAND gates requires that the functions be in sum-of-products form.
- To see the relationship between a sum-of-products expression and its equivalent NAND implementation, consider the logic diagrams drawn in Figure below.
- All three diagrams are equivalent and implement the function

F = AB + CD



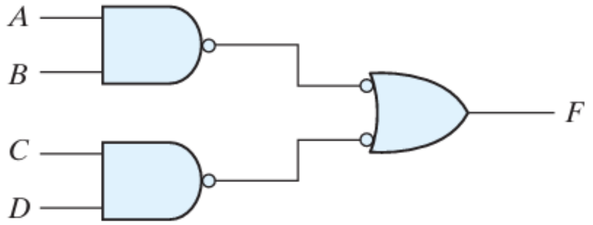
Two-Level Implementation

- The function **F = AB + CD** is implemented in Fig. (a) with AND and OR gates.



(a)

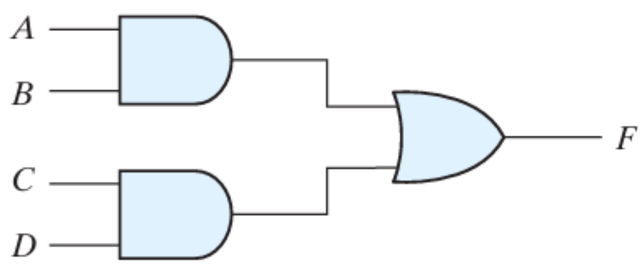
- In Fig. (b), the AND gates are replaced by NAND gates and the OR gate is replaced by a NAND gate with an **OR-invert graphic** symbol.



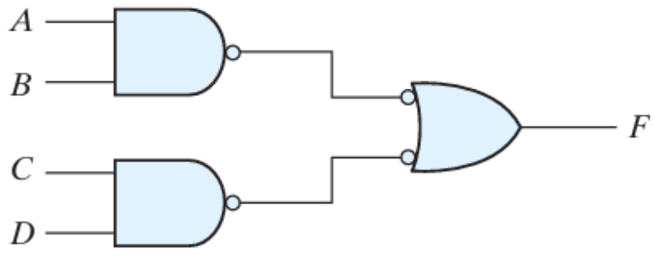
(b)

Two-Level Implementation

- Remember that a bubble denotes complementation and two bubbles along the same line represent double complementation, so both can be removed.
- Removing the bubbles on the gates of (b) produces the circuit of (a).
- Therefore, the two diagrams implement the same function and are equivalent.

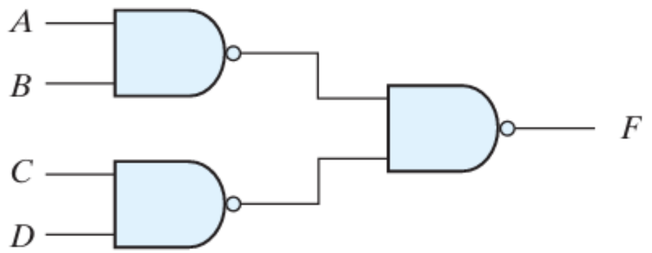


(a)



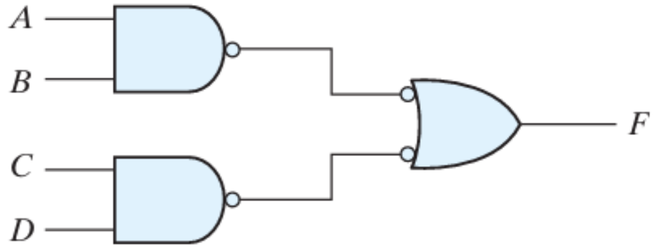
(b)

Two-Level Implementation



(c)

- In Figure (c), the output NAND gate is redrawn with the AND-invert graphic symbol.
- In drawing NAND logic diagrams, the circuit shown in either Figure (b) or (c) is acceptable.
- The one in Figure (b) is in mixed notation and represents a more direct relationship to the Boolean expression it implements.

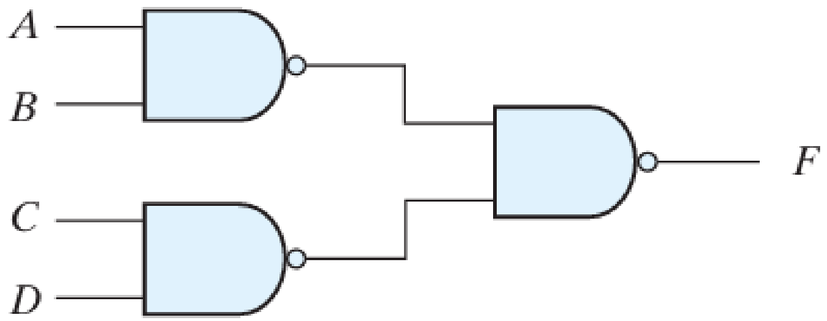


(b)

Two-Level Implementation

- The NAND implementation in Figure (c) can be verified algebraically.
- The function it implements can easily be converted to the sum- of products form by DeMorgan's theorem:

F = ((AB)(CD)) = AB + CD



(c)

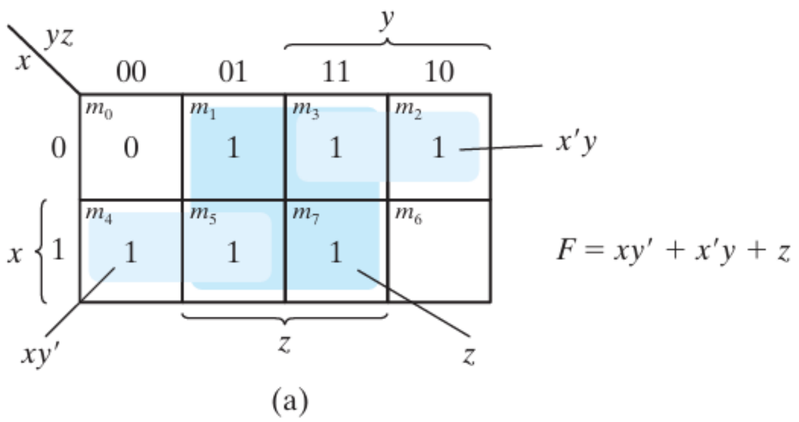
Example

Implement the following Boolean function with NAND gates:

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

- The first step is to simplify the function **into the sum-of-products** form.
- This is done by means of the map of Figure (a), from which the simplified function is obtained:

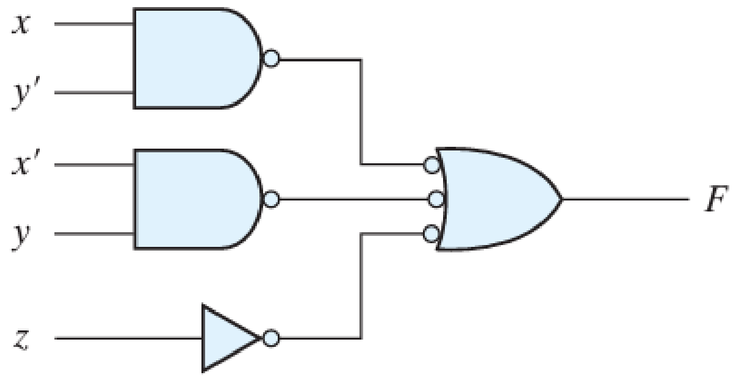
F = xy + xy + z



(a)

Example

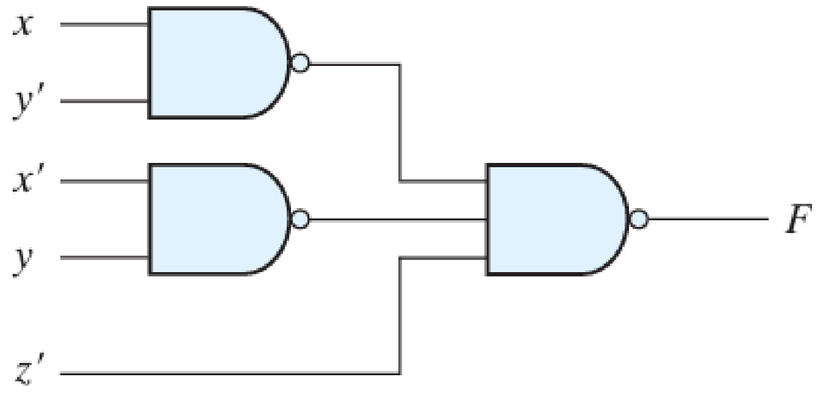
- The two-level NAND implementation is shown in Figure (b) in mixed notation.
- Note that input z must have a one-input NAND gate (an inverter) to compensate for the bubble in the second-level gate.



(b)

Example

- An alternative way of drawing the logic diagram is given in Figure (c).
- Here, all the NAND gates are drawn with the same graphic symbol.
- The inverter with input z has been removed, but the input variable is complemented and denoted by z.



(c)

Example

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. Create a NAND gate for each product term with at least two literals. Use the literals as inputs to these gates, forming the first level.

3. Draw a single gate using the AND-invert or invert-OR graphic symbol for the second level, with inputs connected from outputs of first-level gates.

4. If a term has a single literal, add 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.

Multilevel NAND Circuits

- The standard form of expressing Boolean functions results in a two-level implementation.
- In some cases, the design of digital systems may require gating structures with three or more logic 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.

Multilevel NAND Circuits

- The function can then be implemented with AND and OR gates.
- After that, if necessary, it can be converted into an all-NAND circuit.
- Consider, for example, the Boolean function

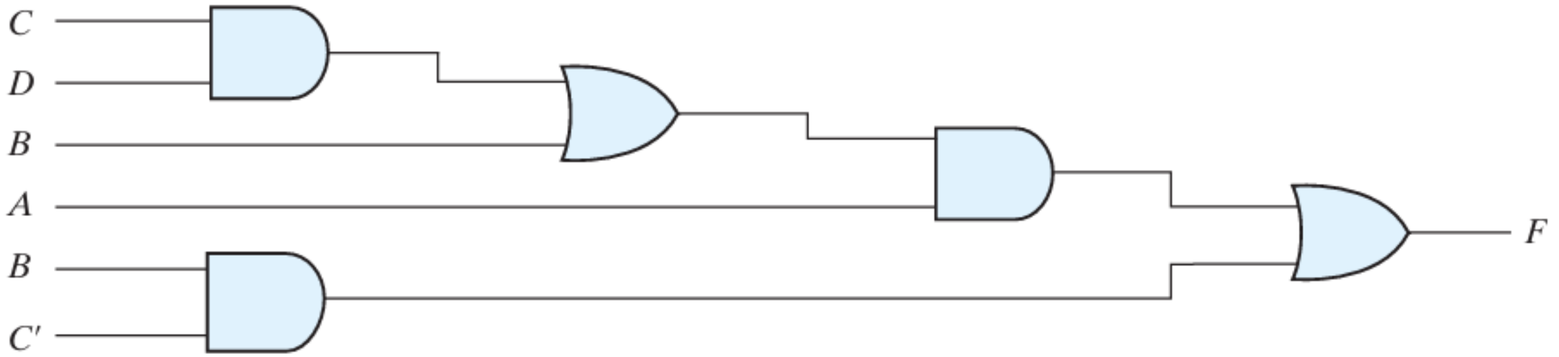
$$F = A\,(CD + B) + BC'$$

Multilevel NAND Circuits

- 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 AND–OR implementation is shown in Figure below. There are four levels of gating in the circuit.

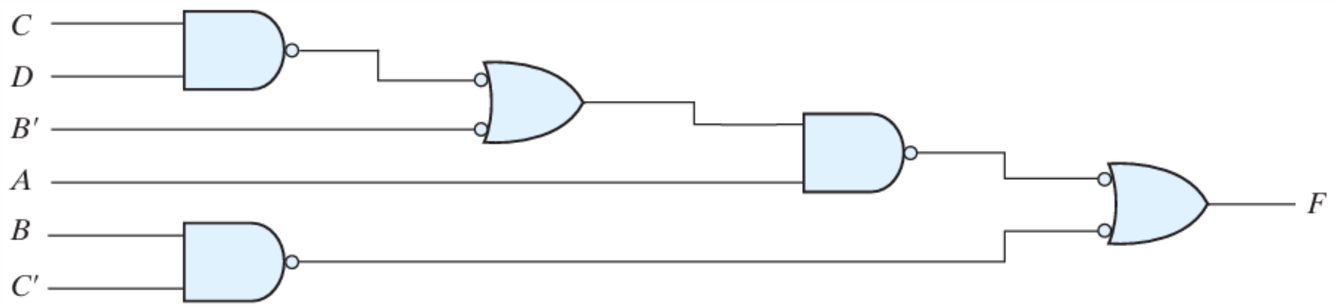
- The first level has two AND gates.
- The second level has an OR gate followed by an AND gate in the third level and an OR gate on the fourth level.



(a) AND–OR gates

Multilevel NAND Circuits

- A logic diagram with a pattern of alternating levels of AND and OR gates can easily be converted into a NAND circuit with the use of mixed notation, shown in Figure below.



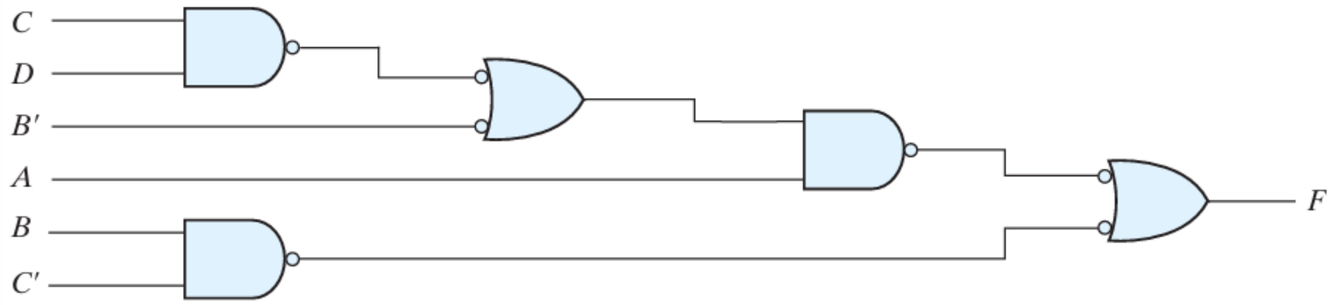
(b) NAND gates

- The procedure is to change every AND gate to an AND-invert graphic symbol and every OR gate to an invert-OR graphic symbol.

Multilevel NAND Circuits

- The **NAND** circuit performs the same logic as the **AND–OR** diagram as long as there are **two bubbles** along the same line.

- The bubble associated with input B causes extra complementation, which must be compensated for by changing the input literal to B.



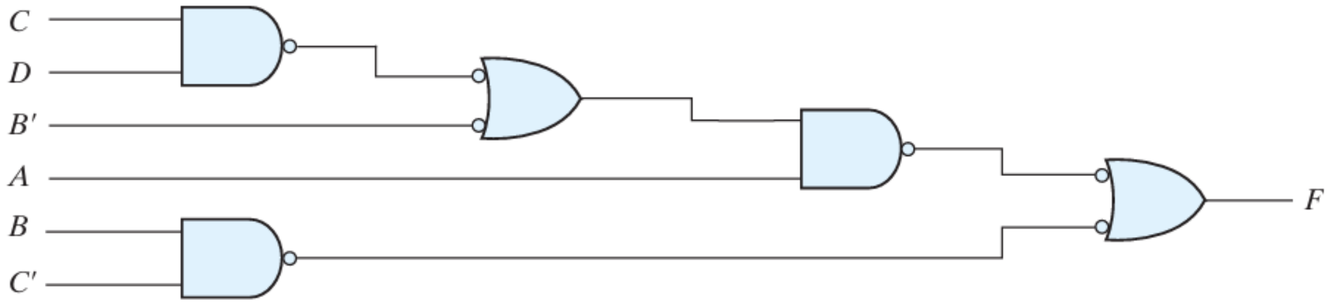
(b) NAND gates

Multilevel NAND Circuits

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, do one of the following:

    - Insert an inverter (a one-input NAND gate).
    - Complement the input literal associated with the bubble.



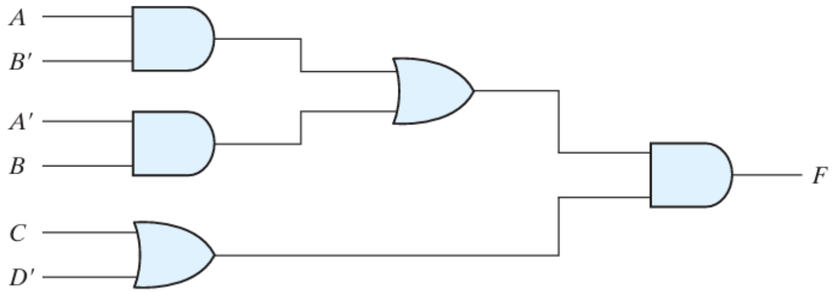
(b) NAND gates

Multilevel NAND Circuits

Another example, consider the multilevel Boolean function

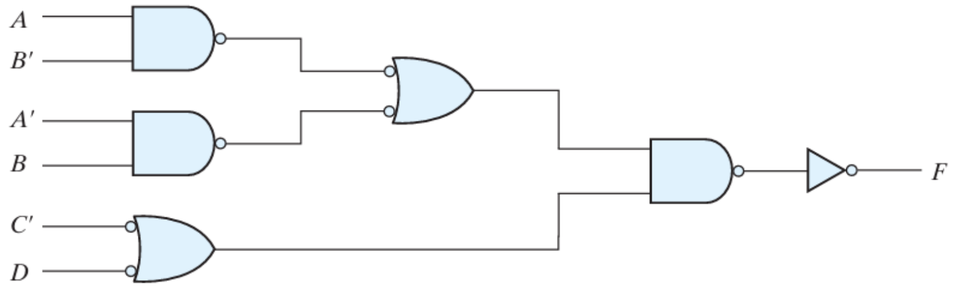
F = (AB + AB)(C + D)

- The AND–OR implementation of above function is shown in Figure with three levels of gating.



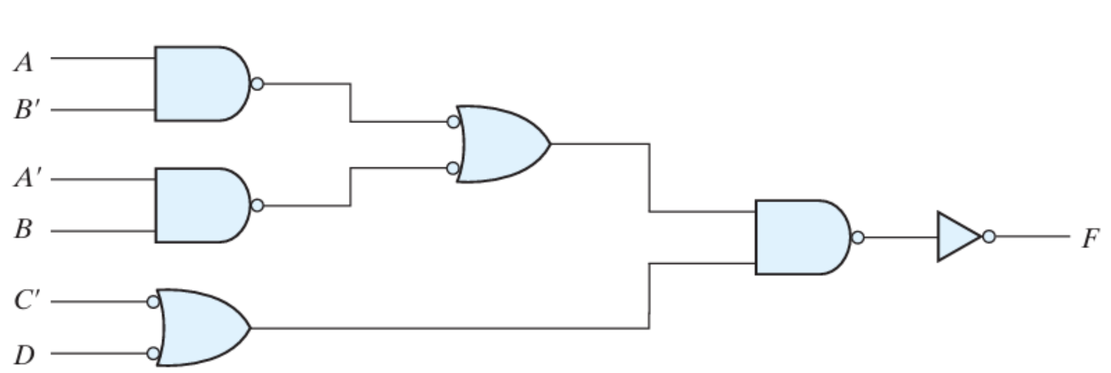
(a) AND–OR gates

- The conversion to NAND with mixed notation is presented in Figure of the diagram.



(b) NAND gates

Multilevel NAND Circuits



(b) NAND gates

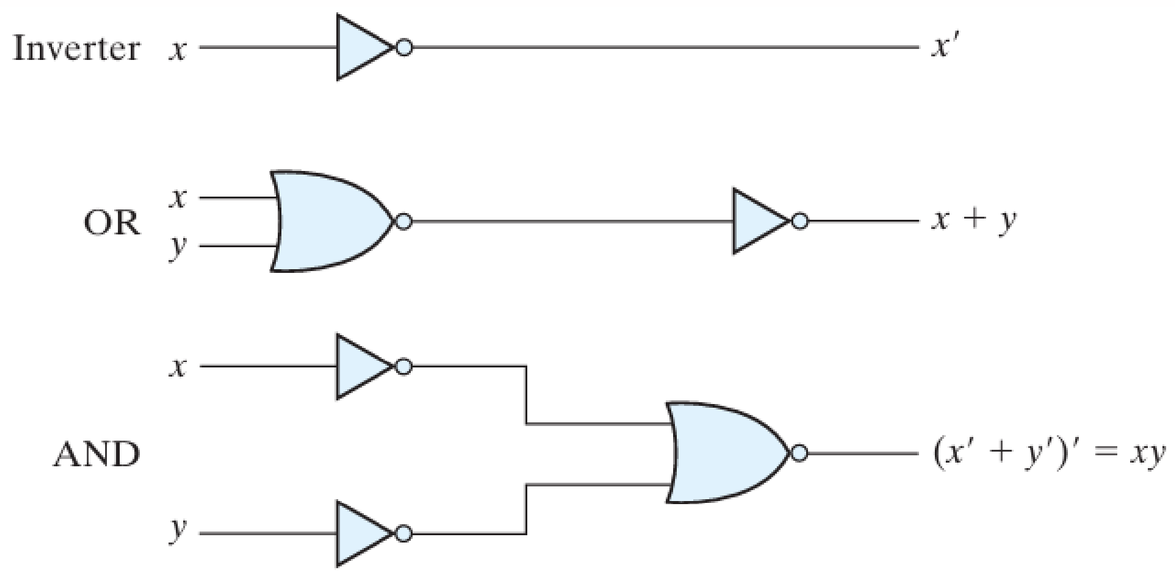
- The two additional bubbles associated with inputs C and D cause these two literals to be complemented by C and D.
- The bubble in the output NAND gate complements the output value, so we need to insert an inverter gate at the output in order to complement the signal again and get the original value back.

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.

NOR Implementation

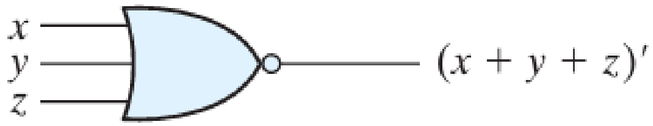
- The implementation of the complement, **OR, and AND operations with NOR** gates is shown in Figure below.
- The complement operation is obtained from a one input NOR gate that behaves exactly like an inverter.
- The OR operation requires **two NOR gates**, and the **AND** operation is **obtained with a NOR** gate that has inverters in each input.



NOR Implementation

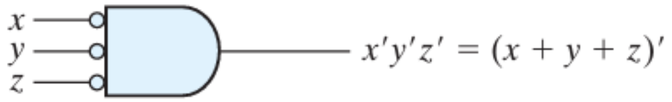
Two graphic symbols for the mixed notation:

- The OR-invert symbol defines the NOR operation as an OR followed by a complement.



(a) OR-invert

- The invert-AND symbol complements each input and then performs an AND operation.



(b) Invert-AND

- The two symbols designate the same NOR operation and are logically identical because of **DeMorgan's theorem**.

NOR Implementation

- To transform a two-level implementation with NOR gates, you need to first simplify the function into product-of-sums form.
- This simplified expression is derived from the map by combining the **0's** and applying complements as needed.
- Once you have the product-of-sums expression, you can implement it with a two-level structure consisting of OR gates and an AND gate.

NOR Implementation

- To convert this **OR-AND** diagram into a **NOR** diagram, follow these steps:

    1. Change all OR gates to NOR gates using the OR-invert graphic symbols.
    2. Change the AND gate to a NOR gate using the invert-AND graphic symbol.

- This transformation allows you to create a circuit using **NOR** gates while maintaining the logic of the original **product-of-sums** expression.

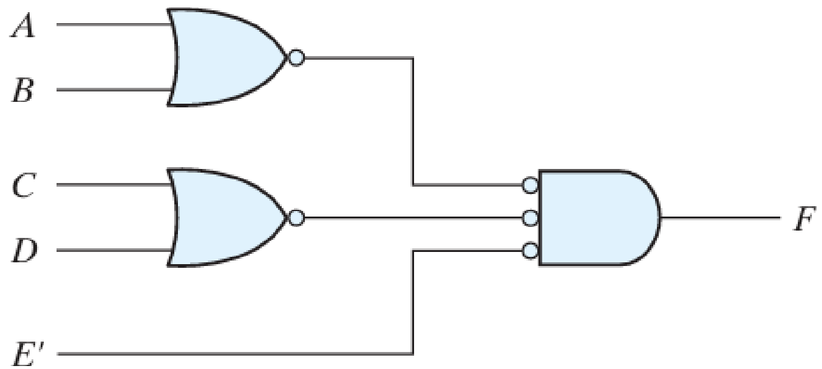
- It's a useful technique when working with NOR gate-based digital logic design.

NOR Implementation

- A **single literal term** going into the **second-level** gate must be complemented.
- The Below Figure shows the NOR implementation of a function expressed as a product of sums:

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

- The OR–AND pattern can easily be detected by the removal of the bubbles along the same line.
- Variable E is complemented to compensate for the third bubble at the input of the second-level gate.



NOR Implementation

- The procedure of converting a multilevel AND-OR diagram to an all-NOR diagram is analogous to the transformation for NAND gates.

- To achieve this for NOR gates, follow these steps:

    1. Change each OR gate to an OR-invert symbol.
    2. Change each AND gate to an invert-AND symbol.

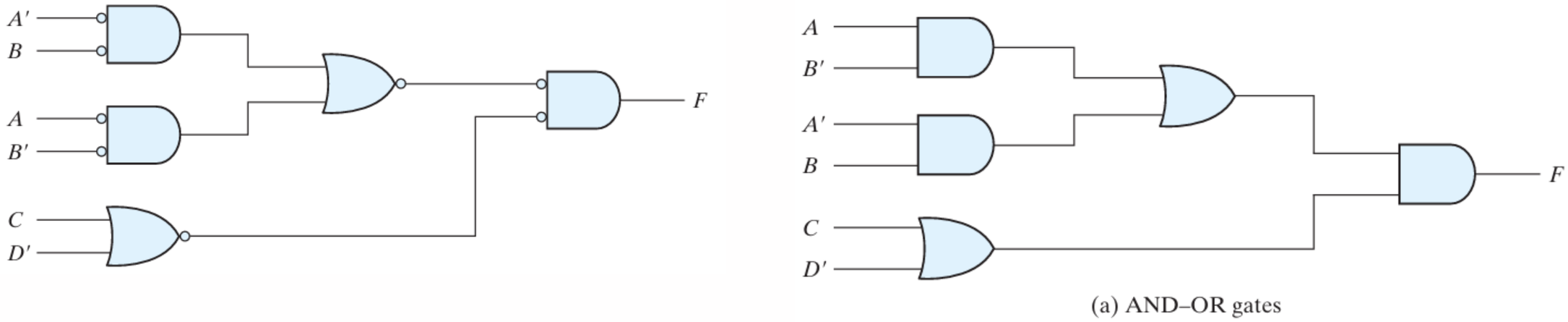
- While making these conversions, be vigilant about bubbles.
- Any bubble that is not compensated by another bubble along the same line needs an inverter, or the complementation of the input literal.

NOR Implementation

- The transformation of the AND–OR diagram of Fig (a) into a NOR diagram is shown in the below Figure.

- The Boolean function for this circuit is

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

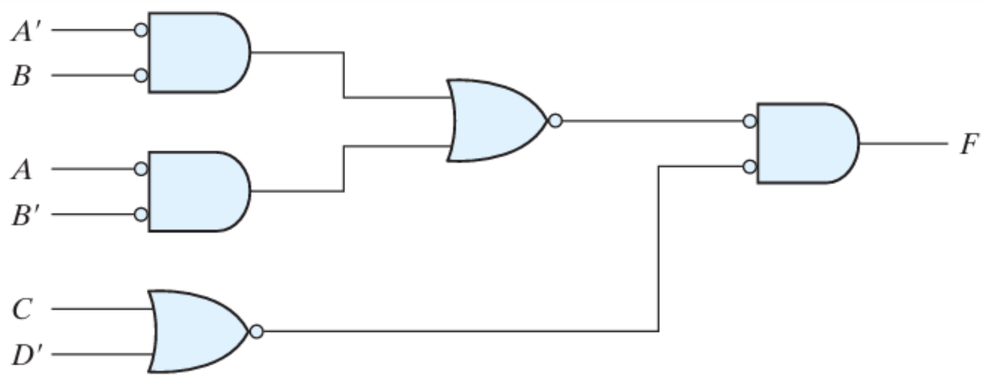


(a) AND–OR gates

NOR Implementation

- The equivalent **AND–OR** diagram can be **recognized from** the **NOR** diagram **by removing all the bubbles**.

- To compensate for the bubbles in four inputs, it is necessary to complement the corresponding input literals.



References

- Computer Organization and Architecture Designing for Performance Tenth Edition by William Stallings
- Digital Design With an Introduction to the Verilog HDL FIFTH EDITION by M Morris, M. and Michael, D., 2013.

Ollscoil na Gaillimhe
University of Galway

Thank *you*

University *of* Galway.ie