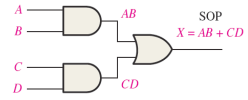
AND-OR

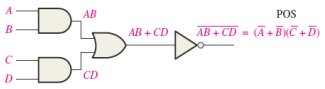
An AND-OR circuit directly implements an SOP expression



AND-OR-INVERT LOGIC

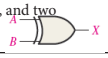
$$X = (\overline{A} + \overline{B})(\overline{C} + \overline{D}) = \overline{(AB)(CD)} = \overline{\overline{(AB)(CD)}} = \overline{\overline{AB} + \overline{CD}} = \overline{AB + CD}$$

When the output of an AND-OR circuit is complemented (inverted), it results in an AND-OR Invert circuit.



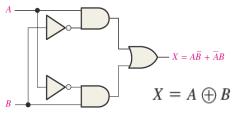
EXCLUSIVE-OR

Although this circuit is considered a type of logic gate with its own unique symbol, it is actually a combination of two AND gates, one OR gate, and two



Truth table for an exclusive-OR.

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

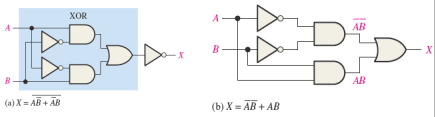


EXCLUSIVE-NOR LOGIC

$$X = \overline{A\overline{B} + \overline{A}B} = \overline{(A\overline{B})}\,\overline{(\overline{A}B)} = (\overline{A} + B)(A + \overline{B}) = \overline{A}\,\overline{B} + AB$$

Notice that the output X is HIGH only when the two inputs, A and B, are at the same level.

The exclusive-NOR can be implemented by simply inverting the output of an exclusive-OR, as shown in Figure 5–6(a), or by directly implementing the expression A B + AB, as shown in part (b).

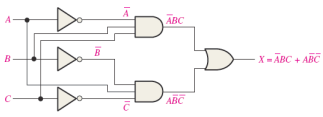


Two equivalent ways of implementing the exclusive-NOR.

FROM A TRUTH TABLE TO A LOGIC CIRCUIT

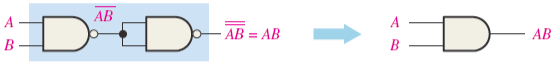
Inputs A	Inputs B	Inputs C	Output X	Product Term
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC$
1	0	0	1	$A\overline{B}\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	0	

$X = \overline{A}BC + A\overline{B}\overline{C}$



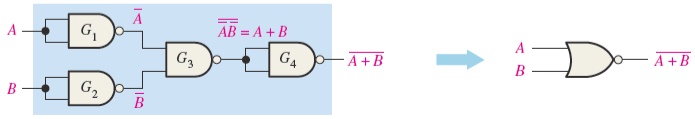
THE NAND GATE AS A UNIVERSAL ELEMENT

(a) One NAND gate used as an inverter



(b) Two NAND gates used as an AND gate

(c) Three NAND gates used as an OR gate

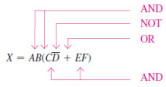


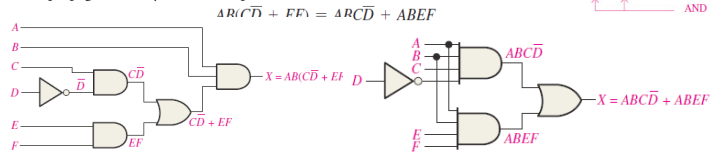
(d) Four NAND gates used as a NOR gate

FROM A BOOLEAN EXPRESSION TO A LOGIC CIRCUIT

Unless an intermediate term, such as CD + EF, is required as an output for some other purpose, it is usually best to reduce a circuit to its SOP form in order to reduce the overall propagation delay time. The expression is converted to SOP as follows:

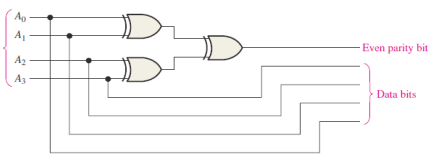
$$AB(C\overline{D} + EF) = ABC\overline{D} + ABEF$$





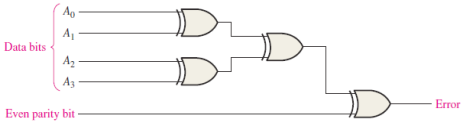
EVEN-PARITY GENERATOR

A parity bit is added to a binary code in order to provide error detection. For even parity, a parity bit is added to the original code to make the total number of 1s in the code even. The circuit in Figure 5–7 produces a 1 output when there is an odd number of 1s on the inputs in order to make the total number of 1s in the output code even. A 0 output is produced when there is an even number of 1s on the inputs.



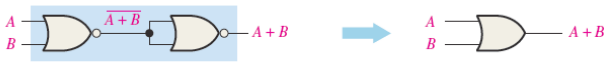
EVEN-PARITY CHECKER

Produces a 1 output when there is an error in the five-bit code and a 0 when there is no error.

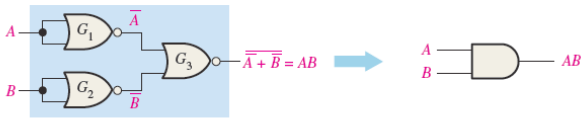


THE NOR GATE AS A UNIVERSAL LOGIC ELEMENT

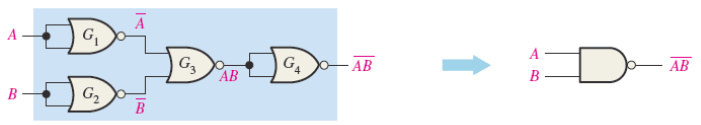
(a) One NOR gate used as an inverter



(b) Two NOR gates used as an OR gate



(c) Three NOR gates used as an AND gate



(d) Four NOR gates used as a NAND gate

NAND LOGIC

As you have learned, a NAND gate can function as either a NAND or a negative-OR because, by DeMorgan's theorem,

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

NAND — negative-OR

Consider the NAND logic in Figure 5–20. The output expression is developed in the following steps:

$$\begin{aligned} X &= \overline{(\overline{AB})(\overline{CD})} \\ &= \overline{(\overline{A} + \overline{B})(\overline{C} + \overline{D})} \\ &= (\overline{\overline{A} + \overline{B}}) + (\overline{\overline{C} + \overline{D}}) \\ &= \overline{\overline{AB}} + \overline{\overline{CD}} \\ &= AB + CD \end{aligned}$$

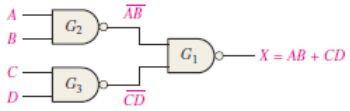
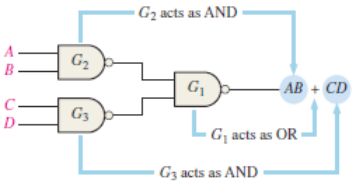


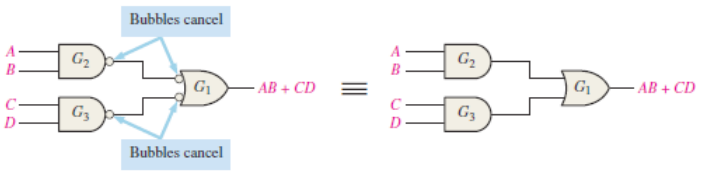
FIGURE 5–20 NAND logic for $X = AB + CD$.

As you can see in Figure 5–20, the output expression, $AB + CD$, is in the form of two AND terms ORed together. This shows that gates G_2 and G_3 act as AND gates and that gate G_1 acts as an OR gate, as illustrated in Figure 5–21(a). This circuit is redrawn in part (b) with NAND symbols for gates G_2 and G_3 and a negative-OR symbol for gate G_1.

Notice in Figure 5–21(b) the bubble-to-bubble connections between the outputs of gates G_2 and G_3 and the inputs of gate G_1. *Since a bubble represents an inversion, two*



(a) Original NAND logic diagram showing effective gate operation relative to the output expression

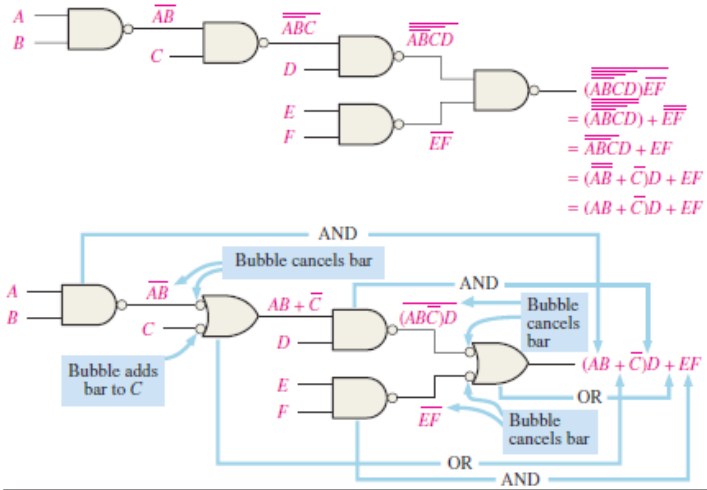


(b) Equivalent NAND/Negative-OR logic diagram (c) AND-OR equivalent

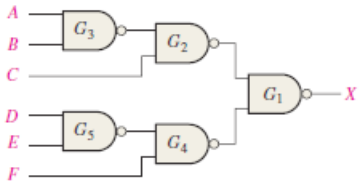
FIGURE 5–21 Development of the AND-OR equivalent of the circuit in Figure 5–20.

NAND LOGIC DIAGRAMS USING DUAL SYMBOLS

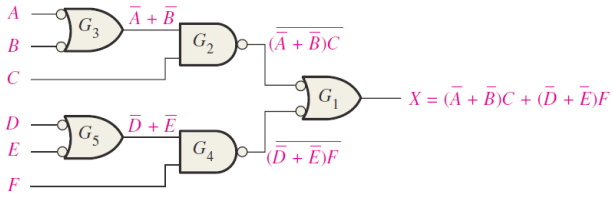
All logic diagrams using NAND gates should be drawn with each gate represented by either a NAND symbol or the equivalent negative-OR symbol to reflect the operation of the gate within the logic circuit. The NAND symbol and the negative-OR symbol are called dual symbols. When drawing a NAND logic diagram, always use the gate symbols in such a way that every connection between a gate output and a gate input is either bubble-tobubble or nonbubble-to-nonbubble. In general, a bubble output should not be connected to a nonbubble input or vice versa in a logic diagram.



NAND USING DUAL SYMBOLS EXAMPLE PROBLEM

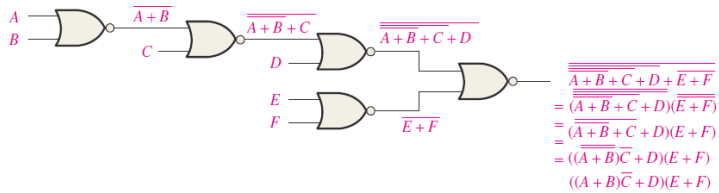


Redraw the logic diagram with the use of equivalent negative-OR symbols as shown. Writing the expression for X directly from the indicated logic operation of each gate gives:

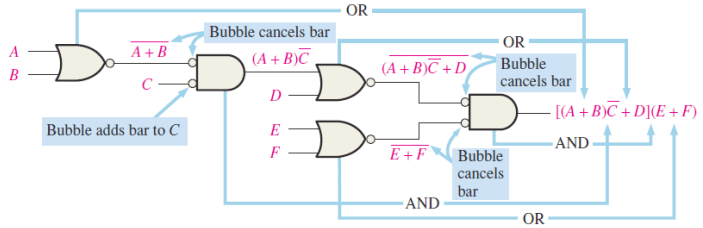


NOR LOGIC DIAGRAM USING DUAL SYMBOLS

As with NAND logic, the purpose for using the dual symbols is to make the logic diagram easier to read and analyze, as illustrated in the NOR logic circuit in Figure 5–28. When the circuit in part (a) is redrawn with dual symbols in part (b), notice that all output-to-input connections between gates are bubble-to-bubble or nonbubble-to-nonbubble. Again, you can see that the shape of each gate symbol indicates the type of term (AND or OR) that it produces in the output expression, thus making the output expression easier to determine and the logic diagram easier to analyze.

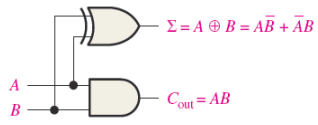


(a) Final output expression is obtained after several Boolean steps.



(b) Output expression can be obtained directly from the function of each gate symbol in the diagram.

THE HALF-ADDER



Notice that the output carry (Cout) is a 1 only when both A and B are 1s; therefore, Cout can be expressed as the AND of the input variables. Cout = AB

Now observe that the sum output (©) is a 1 only if the input variables, A and B, are not equal. The sum can therefore be expressed as the exclusive-OR of the input variables.

Half-adder truth table.

A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Σ = sum
C_{out} = output carry
A and B = input variables (operands)

$$\Sigma = A \oplus B$$

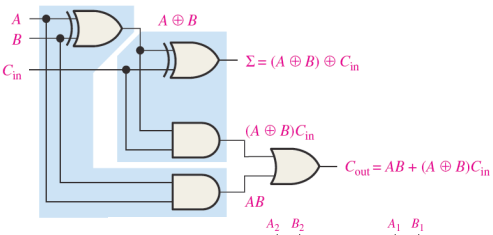
THE FULL-ADDER

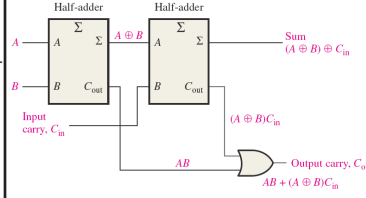
$$\Sigma = (A \oplus B) \oplus C_{in}$$

TABLE 6–2
Full-adder truth table.

A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

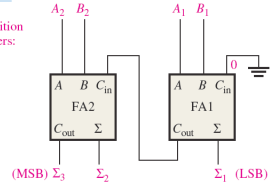
C_{in} = input carry, sometimes designated as *CI*
C_{out} = output carry, sometimes designated as *CO*
Σ = sum
A and B = input variables (operands)



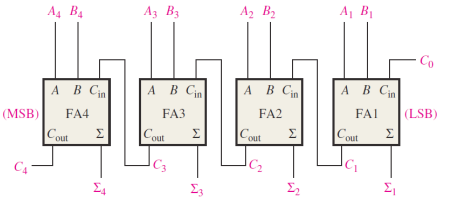


General format, addition of two 2-bit numbers:

$$\begin{array}{r} A_2A_1 \\ +\ B_2B_1 \\ \hline \Sigma_3\Sigma_2\Sigma_1 \end{array}$$

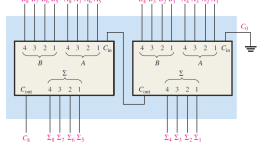


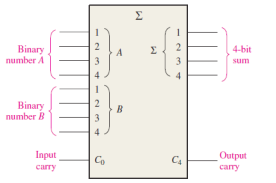
FOUR-BIT PARALLEL ADDERS

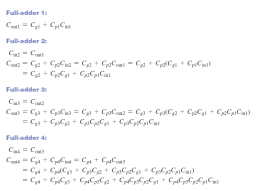


Truth table for each stage of a 4-bit parallel adder.

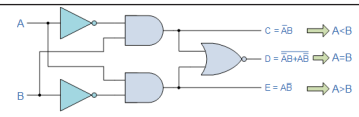
C_{n-1}	A_n	B_n	Σ_n	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



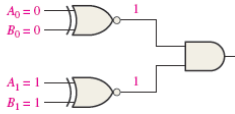




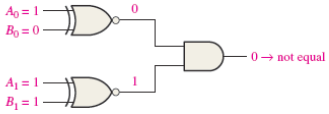
COMPARATORS



10 and 10

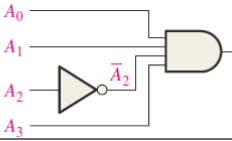


(b) 11 and 10

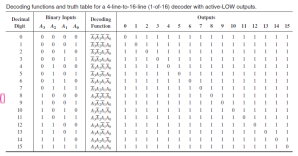


BASIC BINARY DECODER

Decoding logic for producing a HIGH output when 1011 is on the inputs.



$$X = A_3\bar{A}_2A_1A_0$$



LOGIC DIAGRAM FOR A 4-INPUT MULTIPLEXER.

The basic multiplexer has several data-input lines and a single output line. It also has data-select inputs, which permit digital data on any one of the inputs to be switched to the output line.

Data selection for a 1-of-4-multiplexer.

Data-Select Inputs		
S_1	S_0	Input Selected
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3