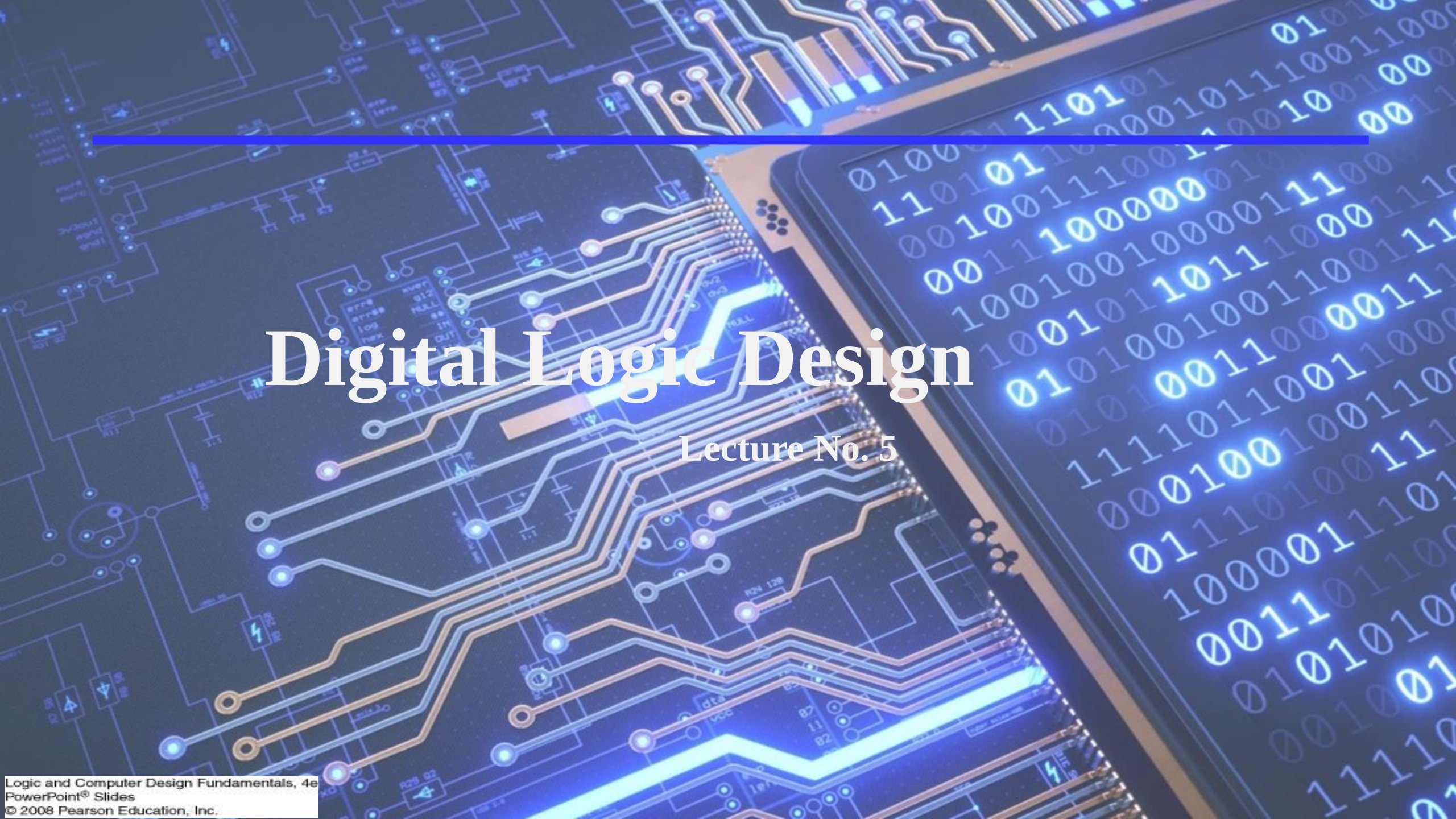


Digital Logic Design

Lecture No. 5

Overview

- Boolean Functions and their implementation
- Definition of Combinational logic circuit.
- Examples of making truth table and circuit diagram from Boolean equation.
- Dual vs complement
- Practice of questions reducing to particular number of literals.
- Implementing functions with only OR and NOT gates.
- Implementing functions with only AND and NOT gates.

Boolean Functions and their Implementation

- Any Boolean function can be represented by using several logic gates by interconnecting them.
- The implementation of Boolean functions by using logic gates involves connecting one logic gate’s output to another gate’s input.
- These logic gates are the building blocks of combinational logic circuits.

Combinational Logic Circuits

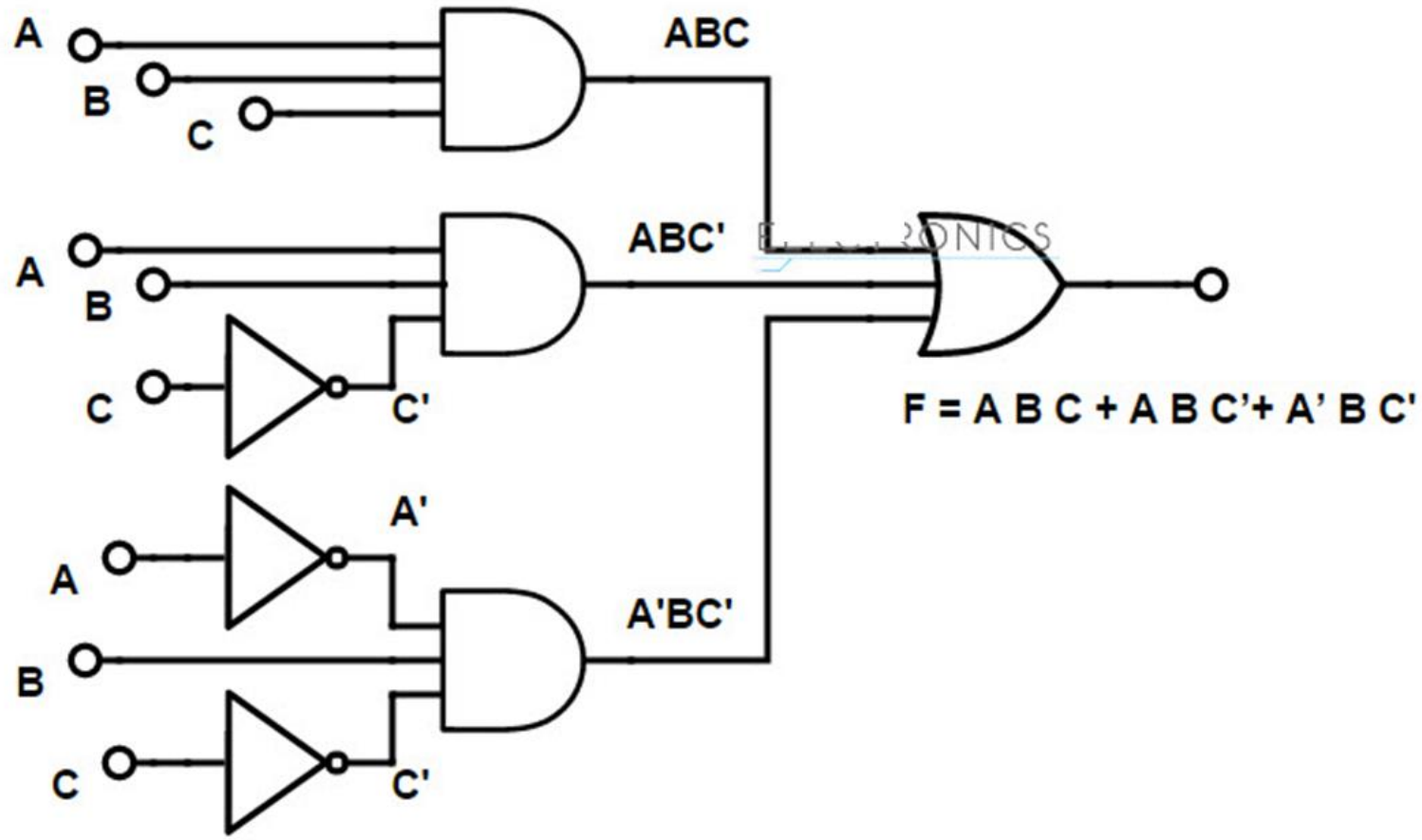
- **Combinational Logic Circuits** are memoryless digital **logic circuits** whose output at any instant in time depends only on the **combination** of its inputs.
- Unlike Sequential **Logic Circuits** whose outputs are dependant on both their present inputs and their previous output state giving them some form of Memory.
- Combinational logic circuits have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output.

Combinational Logic Circuits

- The three main ways of specifying the function of a combinational logic circuit are:
- 1. Boolean Algebra – This forms the algebraic expression showing the operation of the logic circuit for each input variable either True or False that results in a logic “1” output.
- 2. Truth Table – A truth table defines the function of a logic gate by providing a concise list that shows all the output states in tabular form for each possible combination of input variable that the gate could encounter.
- 3. Logic Diagram – This is a graphical representation of a logic circuit that shows the wiring and connections of each individual logic gate, represented by a specific graphical symbol, that implements the logic circuit.

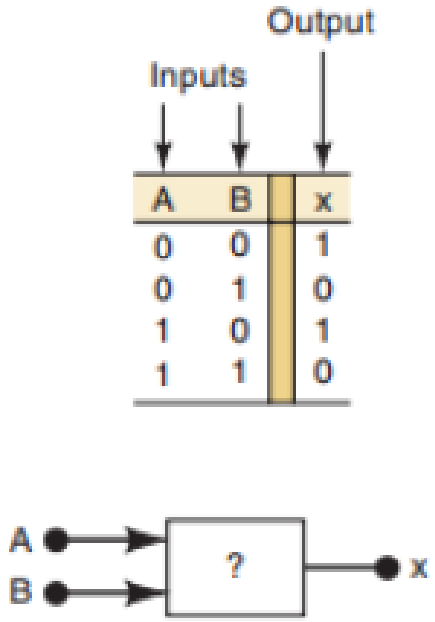
Implement the Boolean function by using basic logic gates:

- F = A B C + A B C’+ A’ B C’



Truth Table

- A truth table is a means for describing how a logic circuit's output depends on the logic levels present at the circuit's inputs.



A	B	x
0	0	1
0	1	0
1	0	1
1	1	0

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

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

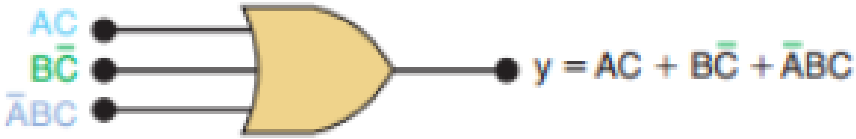
Truth table

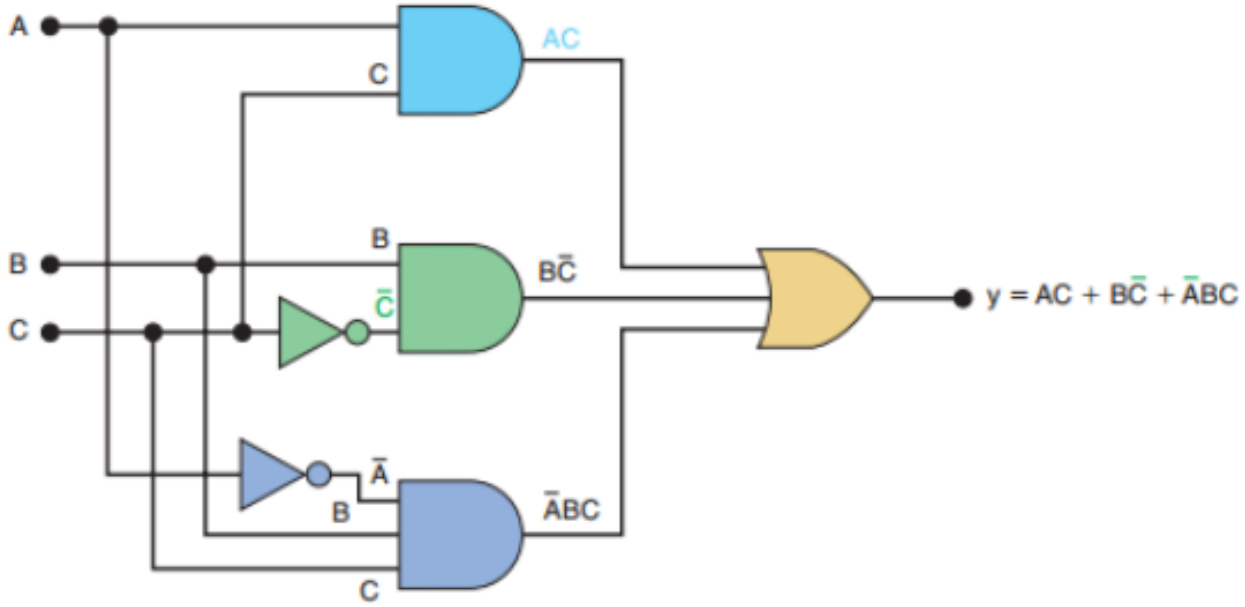
- F = A B C + A B C’+ A’ B C’

A	B	C	$\overline{A}$	$\overline{B}$	$\overline{C}$	F
0	0	0				
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

Implementing circuit diagram from Boolean expression

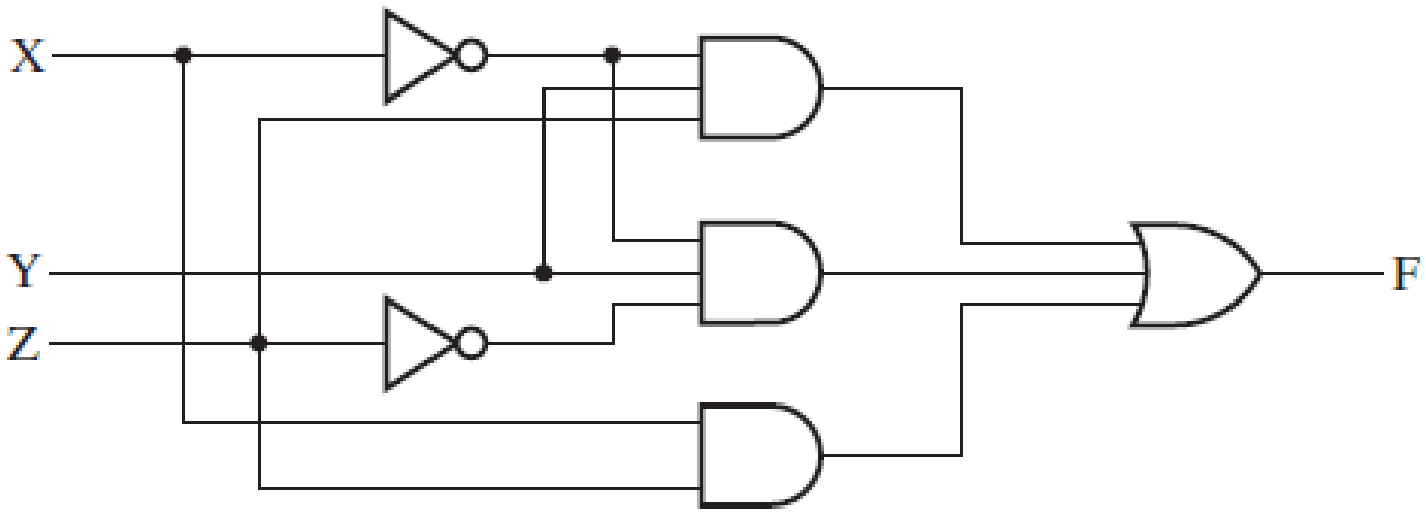
Suppose that we want to construct a circuit diagram whose output is $Y=AC + B\overline{C} + \overline{A}BC$





Example 2

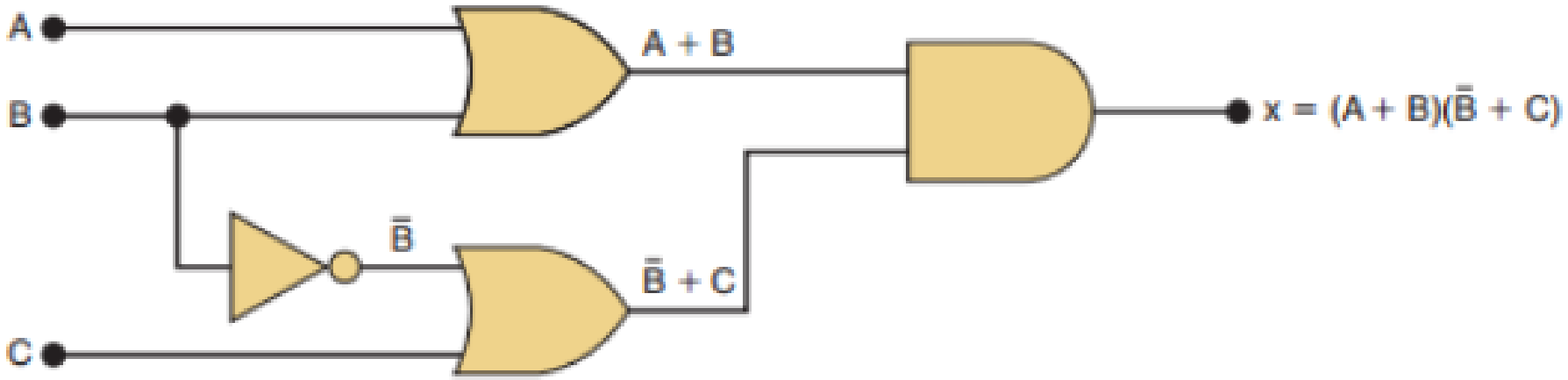
$F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$



(a) $F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$

Example 3

- $X = (A + B)(\overline{B} + C)$



Dual vs Complement

- Boolean duals are generated by simply replacing ANDs with ORs and ORs with ANDs.
- The complement of an expression is the negation of the variables WITH the replacement of ANDs with ORs and vice versa.
- Consider: A+B
- Complement: A'B'
- Dual: AB

Expression Simplification

- An application of Boolean algebra
- Simplify to contain the smallest number of literals (complemented and uncomplemented variables):

$$A\,B + \overline{A}\,C\,D + \overline{A}\,B\,D + \overline{A}\,C\,\overline{D} + A\,B\,C\,D$$

$$= AB + ABCD + \overline{A}\,C\,D + \overline{A}\,C\,\overline{D} + \overline{A}\,B\,D$$

$$= AB + AB(CD) + \overline{A}\,C\,(D + \overline{D}) + \overline{A}\,B\,D$$ Absorption theorem(A.AB=A)

$$= AB + \overline{A}\,C + \overline{A}\,B\,D = B(A + \overline{A}D) + \overline{A}C$$ Distributive (A+BC)=(A+B)(A+C)

$$= B\,(A + D) + \overline{A}\,C$$ 5 literals

Example 2

Simplify the expression $z = \overline{(\overline{A} + C) \cdot (B + \overline{D})}$ to one having only single variables inverted.

Solution

Using theorem (17), and treating $(\overline{A} + C)$ as x and $(B + \overline{D})$ as y, we have

$$z = \overline{(\overline{A} + C)} + \overline{(B + \overline{D})}$$

We can think of this as breaking the large inverter sign down the middle and changing the AND sign $(\cdot)$ to an OR sign $(+)$. Now the term $\overline{(\overline{A} + C)}$ can be simplified by applying theorem (16). Likewise, $\overline{(B + \overline{D})}$ can be simplified:

$$\begin{aligned} z &= \overline{(\overline{A} + C)} + \overline{(B + \overline{D})} \\ &= (\overline{\overline{A}} \cdot \overline{C}) + \overline{B} \cdot \overline{\overline{D}} \end{aligned}$$

Here we have broken the larger inverter signs down the middle and replaced the $(+)$ with a $(\cdot)$. Canceling out the double inversions, we have finally

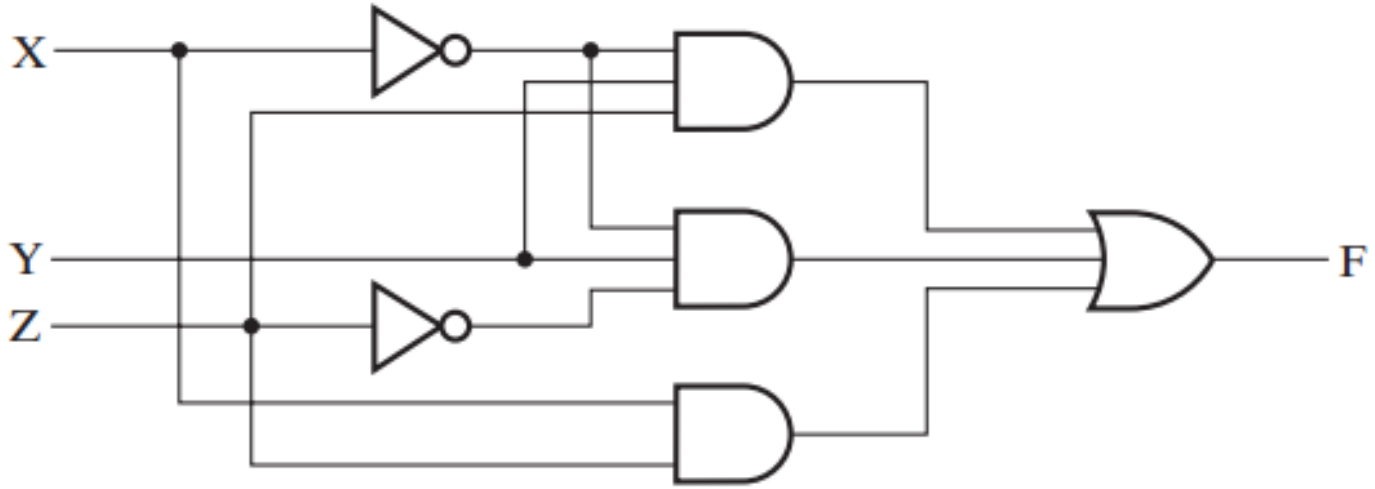
$$z = A\overline{C} + \overline{B}D$$

Example 3

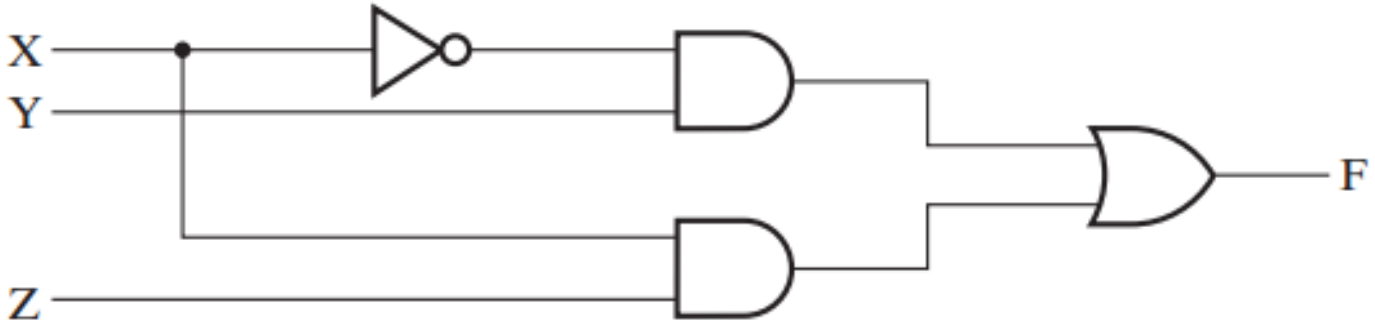
- $F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$

$$
\begin{aligned}
F &= \overline{X}YZ + \overline{X}Y\overline{Z} + XZ \\
&= \overline{X}Y(Z + \overline{Z}) + XZ && \text{by identity 14} \\
&= \overline{X}Y \cdot 1 + XZ && \text{by identity 7} \\
&= \overline{X}Y + XZ && \text{by identity 2}
\end{aligned}
$$

Example 3: Circuit Diagram



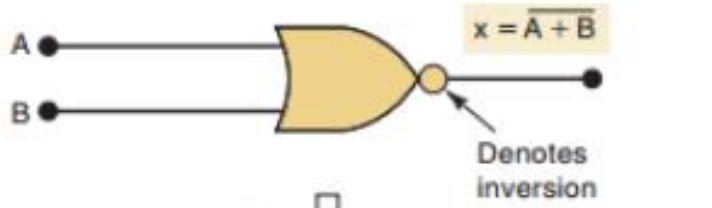
(a) $F = \overline{X}YZ + \overline{X}Y\overline{Z} + XZ$



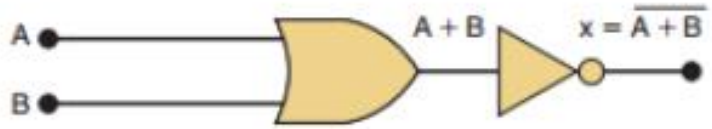
(b) $F = \overline{X}Y + XZ$

NOR Gate

- The NOR gate operates like and OR gate followed by an INVERTER. Its symbol is same as OR gate except that it has a small circle on the output which represents the inversion operation.



(a)



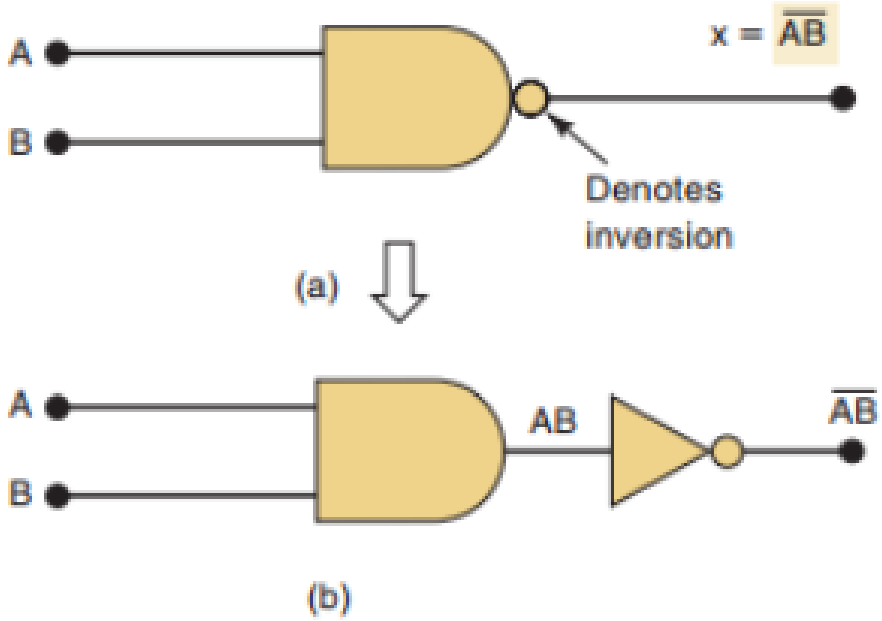
(b)

		OR	NOR
A	B	A + B	$\overline{A + B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

(c)

NAND Gate

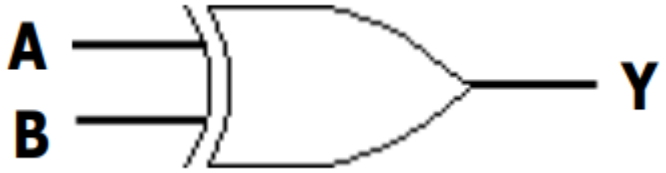
- The NAND gate operates like AND gate followed by an INVERTER. Its symbol is same as AND gate except that it has a small circle on the output which represents the inversion operation.



		AND	NAND
A	B	AB	$\overline{AB}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

(c)

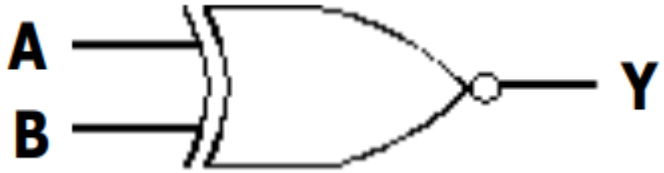
XOR Gate



- This is a XOR gate.
- XOR gates set (1) their output when exactly one of the inputs is set (1).
- The switching algebra symbol for this operation is $\oplus$, i.e. $1 \oplus 1 = 0$ and $1 \oplus 0 = 1$.

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

XNOR Gate



- This is a XNOR gate.
- This functions as an exclusive-NOR gate, or simply the complement of the XOR gate.
- The switching algebra symbol for this operation is $\odot$, i.e. $1 \odot 1 = 1$ and $1 \odot 0 = 0$.

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

Universal gates

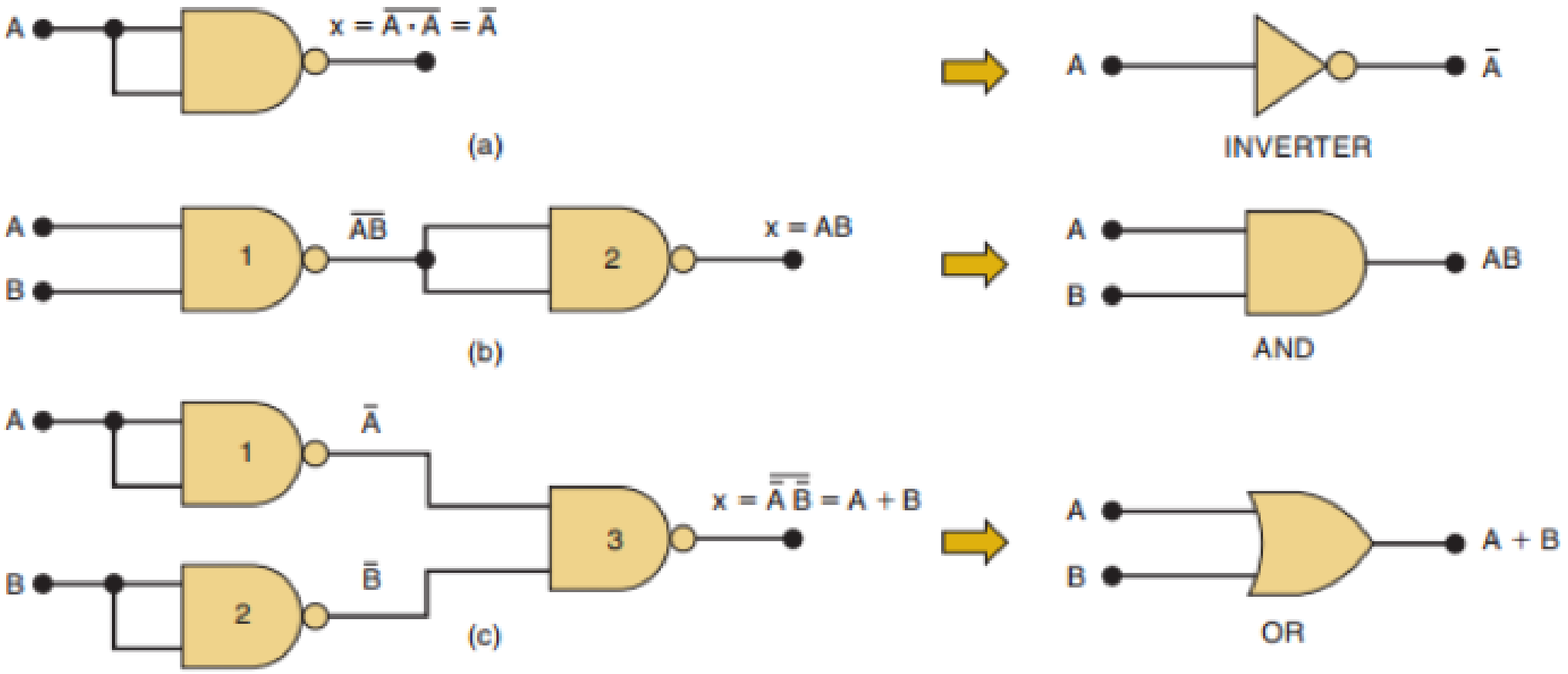
- A universal gate is a gate which can implement any Boolean function without need to use any other gate type.

- The NAND and NOR gates are universal gates.

NAND as Universal Gate

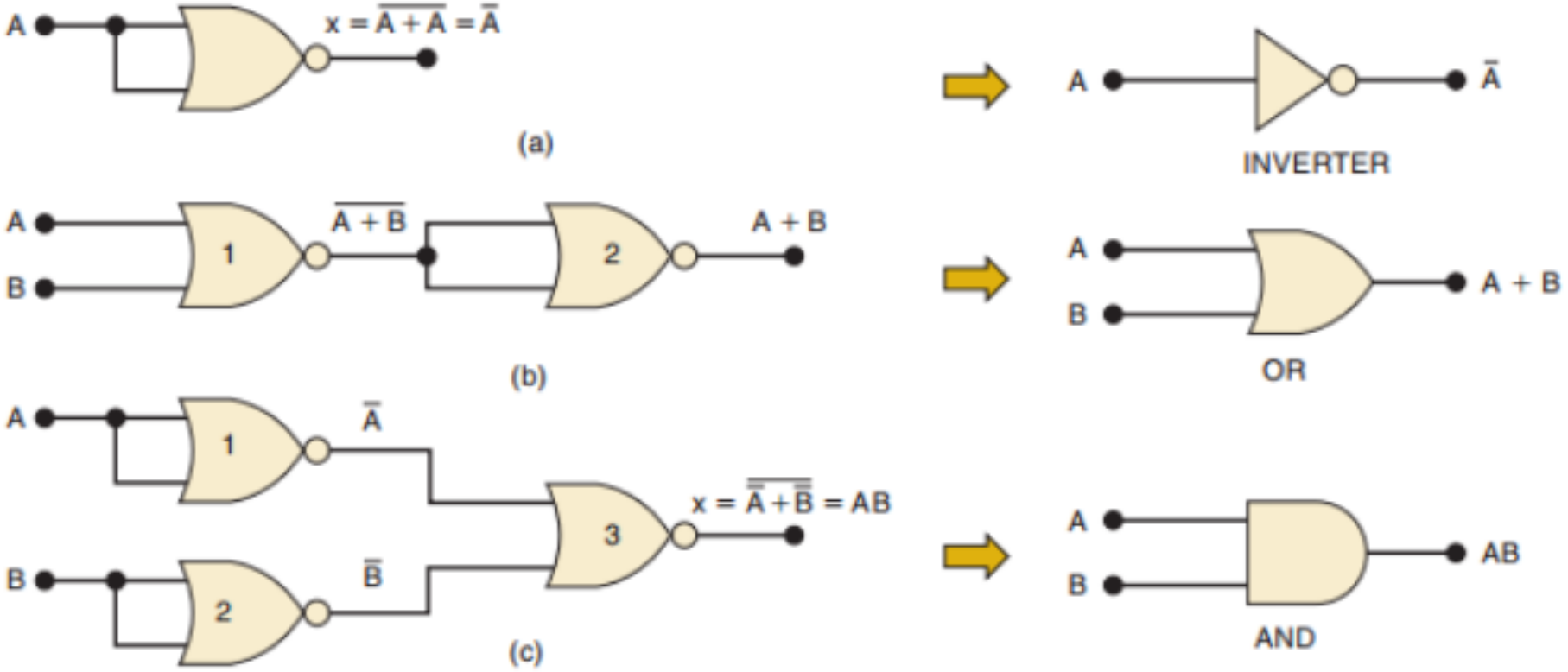
- All Boolean expressions consist of various combinations of the basic operations of OR, AND, and NOT.
- It is possible, however, to implement any logic expression using only NAND gates and no other types of gate.
- This is because NAND gates, in the proper combination, can be used to perform each of the Boolean operations OR, AND, and NOT.

NAND as Universal Gate



NOR as Universal Gate

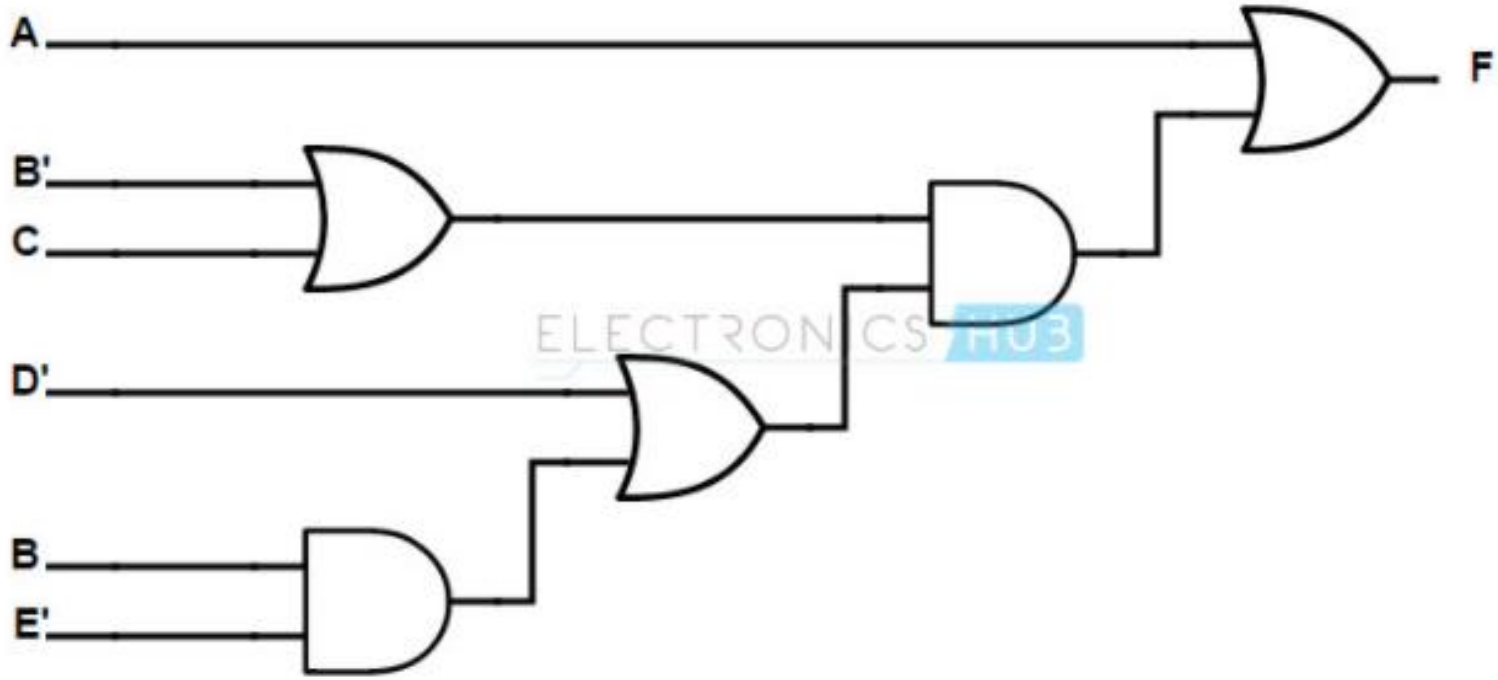
- In a same manner as NAND gate, NOR gates can be arranged to implement any of Boolean operations.



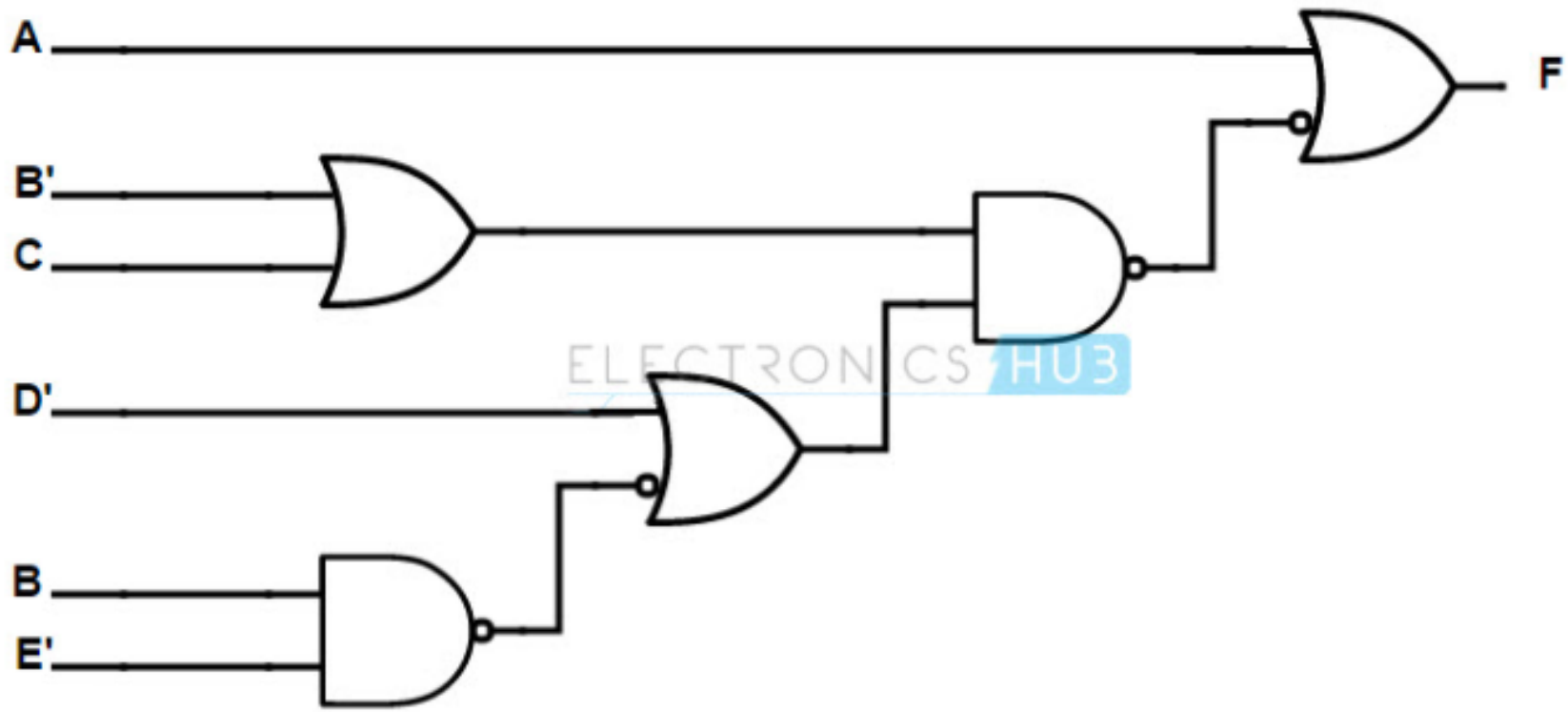
Implement the Boolean function by using a NAND gate

- F (A, B, C, D, E) = A + (B’ + C) (D’ + BE’)

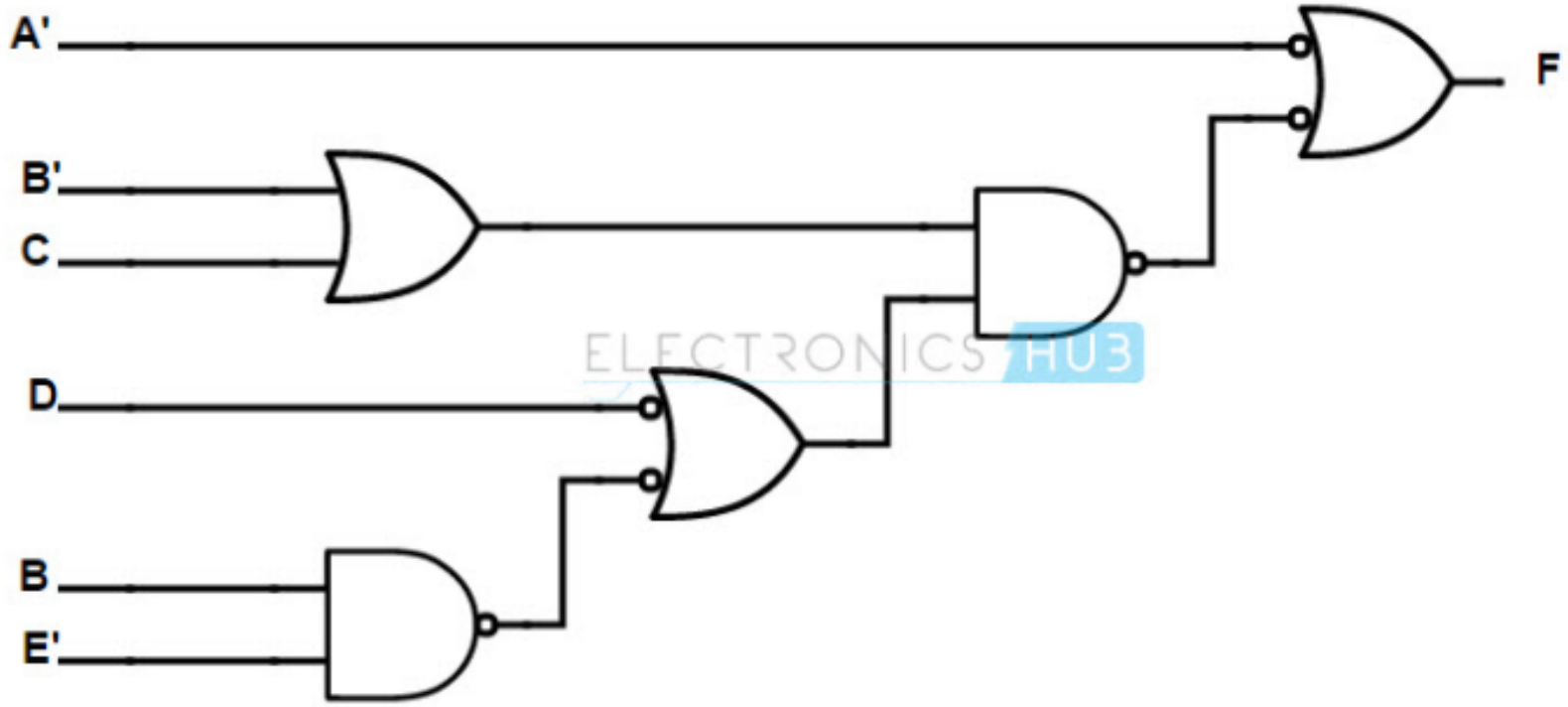
First, the given Boolean function or equation should be represented using AND-OR gates. The AND-OR implementation is shown below.



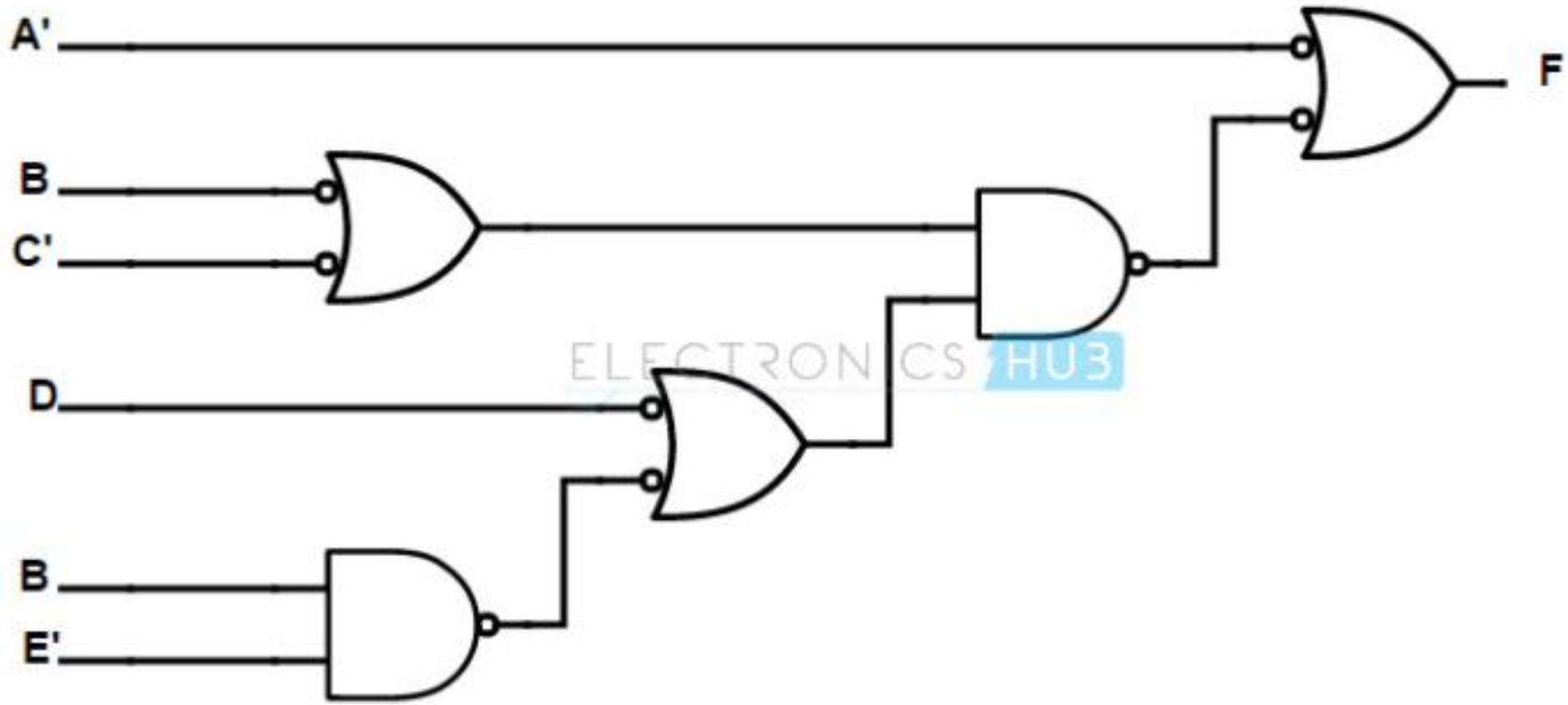
In order to convert the AND gates into NAND gates, a bubble (complement) is introduced at the output of the AND gate. To compensate the bubble, the input of the next gate is also introduced with a bubble. The implementation is shown below.



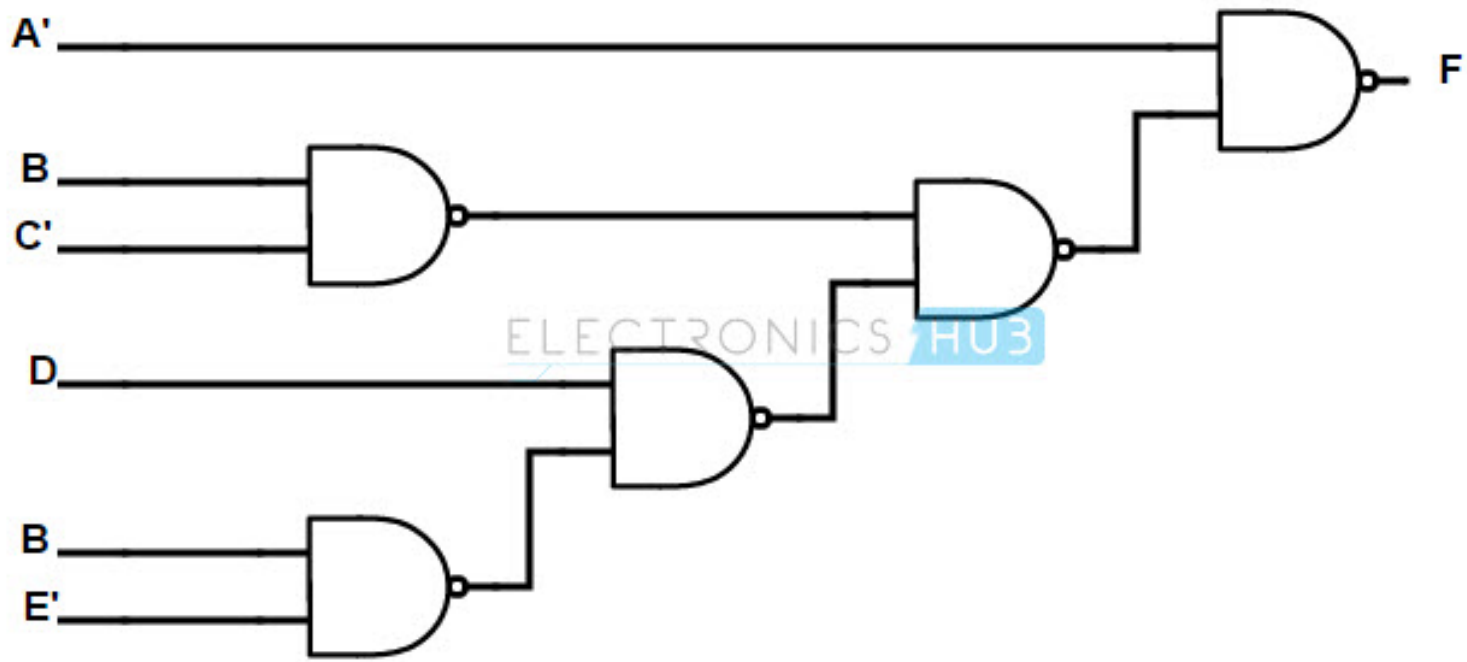
To impose uniformity at the input, if a gate has one input with a bubble, the other input is also introduced with a bubble. Again, in order to compensate the bubble, the output of the preceding gate is introduced with bubble or complement the literal. The same is shown in the following figure.



If an OR gate is not having any bubble at either of the inputs, bubbles are introduced and are appropriately compensated as shown in the figure below.



An OR gate with two complemented inputs is equivalent to a NAND gate (according to DeMorgan's Law A'+B' = (AB)'). Hence, replacing the OR gate, which is having two complemented inputs, with NAND gate, we get the final structure of the implementation of the Boolean function using NAND gates. The final implementation is shown below.



Implementation of Boolean function using NOR gate

Implement the Boolean function by using NOR logic gate.

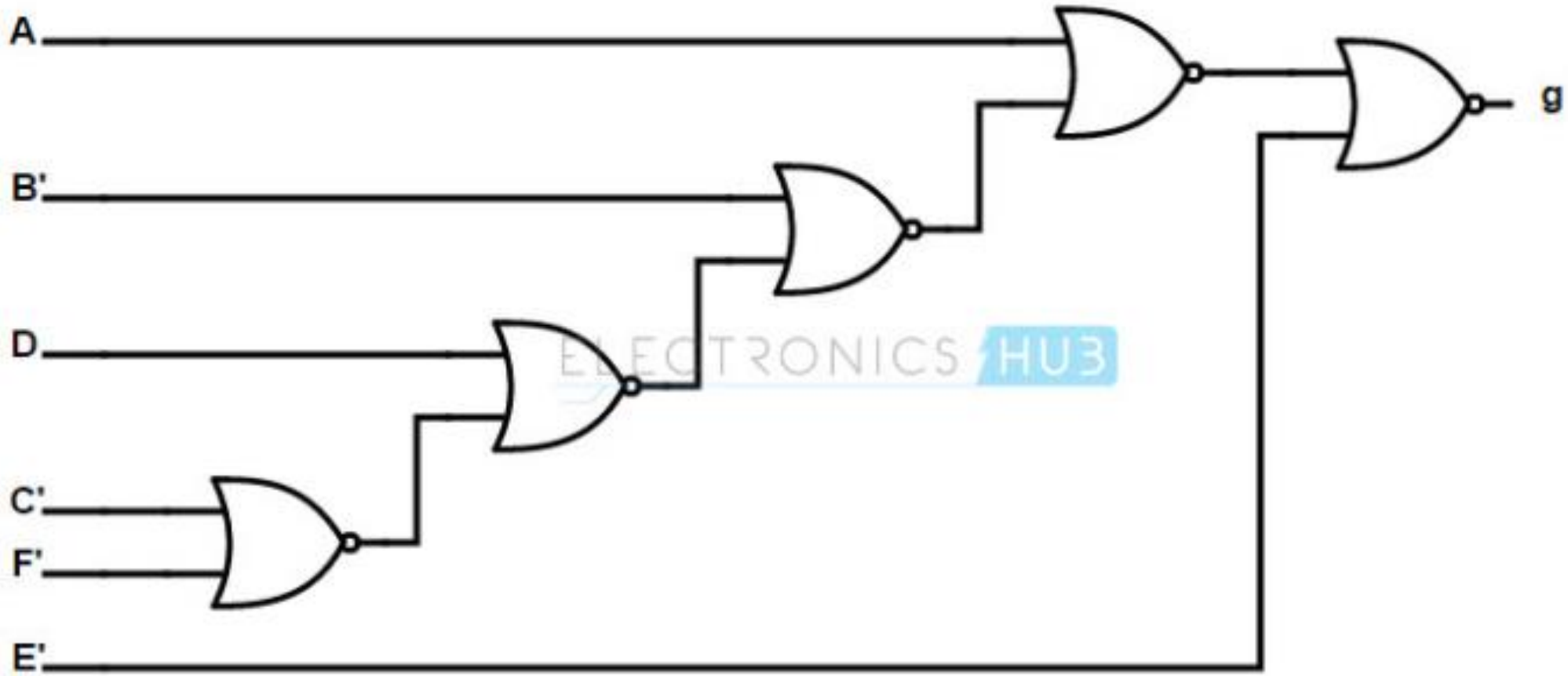
$$g\ (A, B, C, D, E, F) = (A\ E) + (B\ D\ E) + (B\ C\ E\ F)$$

We can solve the given equation as

$$g\ (A, B, C, D, E, F) = AE + BDE + BCEF$$

$$= (A + BD + BCF)\ E$$

$$= (A + B\ (D + CF))\ E$$



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

Why to use Universal gates?

- Both NAND and NOR gates are very valuable as any design can be realized using either one.
- It is easier to build an IC chip using all NAND or NOR gates than to combine AND,OR, and NOT gates.
- NAND/NOR gates are typically faster at switching and cheaper to produce.