Chapter 1
Boolean Algebra and Combinational Logic

Abstract This chapter introduces to the idea of digitally representing analog quantities and goes step by step through the main concepts of the Boolean algebra: variables, functions, truth tables, operations, and properties. The chapter is quite detailed and accompanied by many examples and exercises in order to provide a precise framework of the fundamentals of digital design. It includes the theorems which constitute the foundation for the application of the Boolean algebra to logic networks, with a precise focus on their application for combinational network design.

1.1 Analog and Discrete Variables

In every field of human knowledge, information's observation, memorization, elaboration, and communication is something everyone has to deal with. The definition of the word *"information"* may sound obvious, since this term is commonly used in everyday language, but for our aim we need a definition that leaves no space to any ambiguous interpretation. We hence refer to R. V. L. Hartley (1888–1970), one of the fathers of *Information Theory*, who helps us with the following definition:

Information is a reduction of uncertainty.

From this sentence, it is clear that information is associated with "before" and "after," in relation to an event having a probability to happen; thanks to this probability, an observer reduces his uncertainty related to the event itself. From this definition, it can be easily derived that information can be conveyed through the employment of physical quantities variables, changing in time or space. For example, we can refer to information transmitted by a computer screen through images, defined as variations of luminosity and color in time and space, or the information transmitted by an earphone, through a sound, defined as variations of air pressure over time.

To the aim of studying information processing, we will not refer directly to a physical quantity variable, but rather to its numeric representation, indicated with *"G,"* and its variation over time, indicated with *"T."*

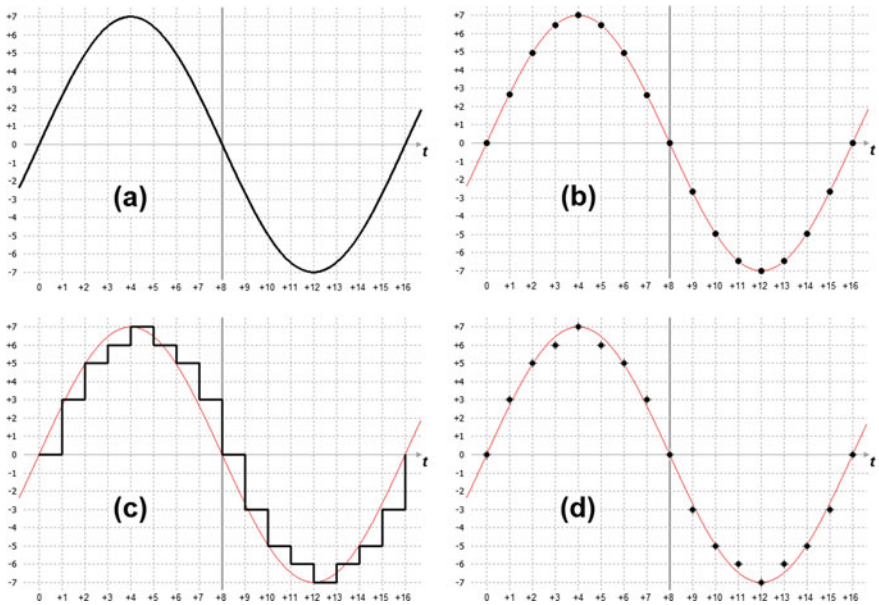
This approach allows us to divide the representation into two families: if G can vary continuously between a value and another one, assuming all the infinite intermediate

G. Donzellini et al., *Introduction to Digital Systems Design*,
https://doi.org/10.1007/978-3-319-92804-3_1

values, we are employing an "analog" representation. If, instead, G can assume only a limited number of values, we are employing a "discrete" (or "digital") representation, becoming "binary" (or "Boolean") if the numbering system uses only two symbols.

Sometimes, the distinction between a physical quantity variable and its numeric representation may create confusion, but we are interested only in the latter. Light, for example, may be described both through a discrete representation (*photons*) and a continuous one (*electromagnetic waves*), but identifying the "true" representation, if it exists, is beyond our scope: we gladly leave this objective to philosophers. For an engineer, what matters is using the most appropriate tool to solve the problem under analysis.

In the picture, four possible representations obtained using discrete or continuous values for G and T are provided [red lines are given for reference]:



(a) continuous quantity variable changing over time;
(b) continuous quantity variable sampled over time;
(c) quantity variable quantized in amplitude and continuous over time;
(d) quantity variable quantized in amplitude and sampled over time.

In the analog case, the main tool at our disposal is math, involving real numbers and functions defined over them: i.e., algebra and infinitesimal calculus. The analog representation is effective at dealing with natural physical quantities variables at the macroscopic level.

This does not mean that natural quantities' variables are analog, but only that in this case the analog representation is the most suitable and effective. In fact, if we go deeper toward the microscopic level, matter reveals its discrete nature (atoms

and particles) and the analog representation is not necessarily the most convenient. Generally, a device able to convert continuous quantities variables into digital ones, and vice versa, is necessary. These devices are called *Analog to Digital Converters* (ADC) and *Digital to Analog Converters* (DAC) respectively, but they are beyond the scope of this book.

The digital case may be faced with discrete math, generally more complex than real numbers math. If we limit to the case of G assuming only two values, i.e., the binary case, we can refer to the *Boolean algebra* that takes its name from its creator, the Irish logician and mathematician George Boole (1815–1864). *Boolean algebra* will be sufficient for our scope, that is, putting the basis of the combinational logic and digital systems.

Binary variables are usually indicated with $\{0, 1\}$ or also with other symbols, like $\{-1, +1\}$, $\{L, H\}$ (*Low* and *High*) or $\{T, F\}$ (*True* or *False*), depending on the context.

In the digital field, we will make use of both the representation over continuous time, called *"asynchronous,"* and the one over discrete time, called *"synchronous."* A logic network is called *synchronous* if its parts operate simultaneously, according to a common synchronization signal; it is instead defined as *asynchronous* if its parts operate in an autonomous mode among each other.

The binary (digital) representation possesses advantages and disadvantages with respect to the analog one:

- an analog value is a pure mathematical abstraction, since it requires infinite precision to be expressed.
- a discrete value (binary) is easily storable, since it requires a finite number (two) of the physical variable values to be memorized.
- the management and processing of binary variables are less sensitive to a possible damaging of the signals that occur during its processing and transmission. In fact, if the damage is not large enough to alter the distinction between the two signal levels *(high/low)*, there is no damage in the information carried by signals.
- the precision of the system can be easily controlled by choosing the number of bits that code the information.
- devices processing digital information, namely digital systems, are simpler to design, though the practical realization requires a higher number of circuital components.

1.2 Boolean Variables

Let X be a certain discreet variable. We will call *Boolean variable* any discreet variable that can assume only two values. These values are denoted as follows:

$$\begin{array}{ll} X = 0 & \text{false} \\ X = 1 & \text{true} \end{array}$$

In the following, the values 0, 1 will be used.

1.3 Boolean Functions

If we have the Boolean variables X_1, X_2, ..., X_n, the following:

$$f\ (X_1, X_2, \ldots, X_n)$$

is called a Boolean function, and it can assume only the values 0 and 1. This function associates a Boolean value to every element in its domain.

The domain of a function of n-variables is composed of all the 2^n combinations of their values. Therefore, domain's elements are countable. Two functions are equivalent if they assume the same value for any combination of their variables' values.

1.4 Truth Tables

The Principle of Perfect Induction (that is, carrying out all the calculations) makes it possible to prove the value of f for all the 2^n points of the domain. The function is represented in the truth table.

Let's assume a three-variable function X_1, X_2, X_3. We can construct a table with all the values assumed by f:

X_1	X_2	X_3	f
0	0	0	
0	0	1	values
0	1	0	
0	1	1	assumed
1	0	0	
1	0	1	by
1	1	0	
1	1	1	f

Observation: To write the $2^3 = 8$ elements of the domain, we begin at the farthest right column (X_3), from the top and alternate between one 0 and one 1. In the next column, we alternate between two 0s and two 1s, while in the column after that, four 0s and four 1s and so on, doubling the number of 0s and 1s with each new column.

Examples:

Derive the **truth tables** from the *verbal definitions*:

1. U is true if C is true or if B and A are both true.

2. Z is true if the number of ones in the inputs M, G, D is equal to two.

1.

C	B	A	U
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

2.

M	G	D	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

1.5 Definition of Boolean Algebra

Boolean algebra provides the necessary tools to calculate and interpret information presented in binary form. Boolean algebra is an *algebraic system* (a set of elements to which a set of operations is associated), defined by:

- The set of values {0,1};
- The operations *OR*, *AND*, and *NOT*;
- The equivalence operator "=", along with the properties reflexive, symmetric, and transitive.

The three operations are defined as follows:

Operation:	OR (logical sum)	AND (logical product)	NOT (negation)
Algebraic symbols:	$X + Y$ $X \vee Y$ $X \cup Y$ X or Y	$X \cdot Y = XY$ $X \wedge Y$ $X \cap Y$ X and Y	$\overline{X}$ $!X$ $-X$ not(X)

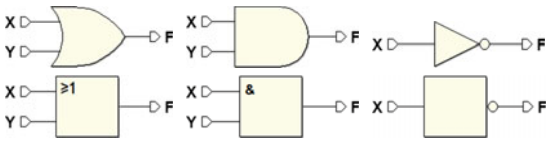
Truth table:

X	Y	$X + Y$
0	0	0
0	1	1
1	0	1
1	1	1

X	Y	$X \cdot Y$
0	0	0
0	1	0
1	0	0
1	1	1

X	$\overline{X}$
0	1
1	0

Circuit diagram symbols:



1.6 The Fundamental Properties of Boolean Algebra

Conventions

- X, Y, Z, X_1, X_2, X_3, ..., X_n, are considered Boolean variables.
- The parentheses establish the calculation priorities as in regular algebra.
- AND is prioritized over OR (e.g., $X + YZ = X + (YZ)$).

This is also analogous to regular algebra. All the properties can be demonstrated through Perfect Induction, that is, by verifying the validity of each combination of values assumed by the variables that make up the expression.

Example: $X \cdot 0 = 0$ is verified through the truth table:

X	0	$X \cdot 0$
0	0	0
1	0	0

Duality Principle

If a given expression is valid, its dual expression is also valid. The dual expression is obtained by switching the OR with the AND and the 0 constants with the 1 constants from the original expression. For example:

$$\begin{array}{rr} & X + 1 = 1 \\ \text{(dual:)} & X \cdot 0 = 0 \end{array}$$

$$\begin{array}{rr} & X + 0 = X \\ \text{(dual:)} & X \cdot 1 = X \end{array}$$

Idempotent Law

$$\begin{array}{rr} & X + X = X \\ \text{(dual:)} & X \cdot X = X \end{array}$$

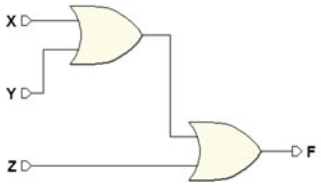
Commutative Law

$$\begin{array}{rr} & X + Y = Y + X \\ \text{(dual:)} & X \cdot Y = Y \cdot X \end{array}$$

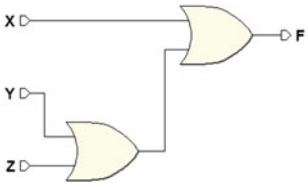
Associative Law

$$\begin{array}{rr} & (X + Y) + Z = X + (Y + Z) = X + Y + Z \\ \text{(dual:)} & (X \cdot Y) \cdot Z = X \cdot (Y \cdot Z) = X \cdot Y \cdot Z. \end{array}$$

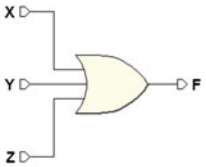
The associative law makes it possible to extend fundamental operations to more than two variables. The circuit symbols for the first expression are:



$(X + Y) + Z$



$X + (Y + Z)$



$X + Y + Z$

As there is no distinction between the first and second circuits, it makes sense to generally define an OR of three or more inputs. The same holds true for the AND, so it is really the property of Associativity that allows us to make sense of OR and AND gates with more than two inputs.

We can redefine the OR and AND operations with n inputs:

- An OR with n inputs gives a 0 as output only if all the n inputs are 0, otherwise it gives a 1 as output.
- An AND with n inputs gives a 1 as output only if all the n inputs are 1, otherwise it gives a 0 as output.

Distributivity

Factoring law	$(X + Y) \cdot (X + Z) = X + (Y \cdot Z)$
Distributive law (dual:)	$(X \cdot Y) + (X \cdot Z) = X \cdot (Y + Z)$

Proof of the factoring law:

$$\begin{aligned}(X + Y) \cdot (X + Z) &= X \cdot X + X \cdot Z + X \cdot Y + Y \cdot Z = \\ &= X + X \cdot Z + X \cdot Y + Y \cdot Z \\ &= X \cdot (1 + Y) + X \cdot Z + Y \cdot Z \\ &= X + X \cdot Z + Y \cdot Z \\ &= X \cdot (1 + Z) + Y \cdot Z \\ &= X + (Y \cdot Z)\end{aligned}$$

It would also be possible to demonstrate this law through Perfect Induction (i.e., verifying all the possible combinations for X, Y, Z):

X	Y	Z	$Y \cdot Z$	$X + Y \cdot Z$	$X + Y$	$X + Z$	$(X + Y)(X + Z)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

It is clear that columns $X + Y \cdot Z$ and $(X + Y)(X + Z)$ are equal.

Complementation

$$\begin{aligned} & X + \overline{X} = 1 \\ \text{(dual:)} \quad & X \cdot \overline{X} = 0 \end{aligned}$$

Absorption

First form:

$$\begin{aligned} & X + X \cdot Y = X \\ \text{(dual:)} \quad & X \cdot (X + Y) = X \end{aligned}$$

Second form:

$$\begin{aligned} & X + (\overline{X} \cdot Y) = X + Y \\ \text{(dual:)} \quad & X \cdot (\overline{X} + Y) = X \cdot Y \end{aligned}$$

Proof:

$$\begin{aligned} X + X \cdot Y &= X \cdot (1 + Y) = X \cdot 1 = X \\ X \cdot (X + Y) &= X \cdot X + X \cdot Y = X + X \cdot Y = X \\ X + \overline{X} \cdot Y &= X + X \cdot Y + \overline{X} \cdot Y = X + Y(X + \overline{X}) = X + Y \\ X \cdot (\overline{X} + Y) &= X \cdot \overline{X} + X \cdot Y = X \cdot Y \end{aligned}$$

Logic Adjacency

$$\begin{aligned} & Y\,X + Y\,\overline{X} = Y \\ \text{(dual:)} \quad & (Y + X) \cdot (Y + \overline{X}) = Y \end{aligned}$$

Proof:

$$\begin{aligned} Y\,X + Y\,\overline{X} &= Y \cdot (X + \overline{X}) = Y \cdot 1 = Y \\ (Y + X) \cdot (Y + \overline{X}) &= Y + (X \cdot \overline{X}) = Y + 0 = Y \end{aligned}$$

Consensus

$$\begin{aligned} & X \cdot Y + Y \cdot Z + Z \cdot \overline{X} = X \cdot Y + Z \cdot \overline{X} \\ \text{(dual:)} \quad & (X + Y)(Y + Z)(Z + \overline{X}) = (X + Y)(Z + \overline{X}) \end{aligned}$$

Proof:

$$
\begin{aligned}
&X \cdot Y + Y \cdot Z + Z \cdot \overline{X} = \\
&\qquad = X \cdot Y + Y \cdot (X + \overline{X}) \cdot Z + Z \cdot \overline{X} = \\
&\qquad = (X \cdot Y + X \cdot Y \cdot Z) + (Z \cdot \overline{X} \cdot Y + Z \cdot \overline{X}) = \\
&\qquad = X \cdot Y + Z \cdot \overline{X} \\
&(X + Y)(Y + Z)(Z + \overline{X}) = \\
&\qquad = (X + Y)[(X + Y + Z)(\overline{X} + Y + Z)](Z + \overline{X}) = \\
&\qquad = [(X + Y)(X + Y + Z)][(Z + \overline{X} + Y)(Z + \overline{X})] = \\
&\qquad = (X + Y)(Z + \overline{X})
\end{aligned}
$$

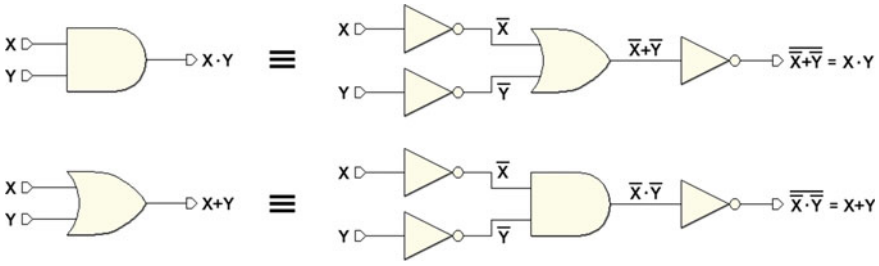
Involution

Also known as *Double Complement law*: $\overline{\overline{X}} = X$.

Duality or De Morgan's Theorem

A logical product of two variables can be substituted by the negation of their logical sum. Dual: a logical sum of two variables can be substituted by the negation of their logical product:

$$
\begin{aligned}
X \cdot Y &= \overline{\overline{X} + \overline{Y}} \\
\text{(dual:)} \quad X + Y &= \overline{\overline{X} \cdot \overline{Y}}
\end{aligned}
$$



This theorem is important: it allows us to obtain an AND through an OR gate and vice versa. The theorem tells us that either one of the two functions is superfluous according to the definition of Boolean algebra.

Generalized De Morgan's Theorem

The theorem applies to any number of variables:

$$
\begin{aligned}
X_1 \cdot X_2 \cdot \ldots \cdot X_n &= \overline{\overline{X_1} + \overline{X_2} + \ldots + \overline{X_n}} \\
\text{(dual :)} \quad X_1 + X_2 + \ldots + X_n &= \overline{\overline{X_1} \cdot \overline{X_2} \cdot \ldots \cdot \overline{X_n}}.
\end{aligned}
$$

1.7 Other Operations

In this paragraph, we define other operations in Boolean algebra: NAND, NOR, and EXOR.

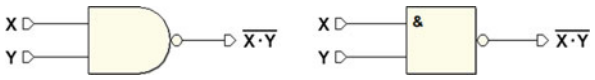
NAND

The NAND operation is equivalent to an AND whose output is negated:

$$X \text{ nand } Y = \overline{(X \cdot Y)}$$

X	Y	$(X \text{ nand } Y)$
0	0	1
0	1	1
1	0	1
1	1	0

Circuital symbols:



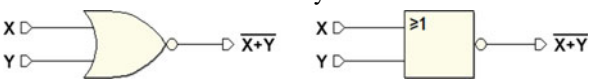
NOR

The NOR operation is equivalent to an OR whose output is negated:

$$X \text{ nor } Y = \overline{(X + Y)}$$

X	Y	$(X \text{ nor } Y)$
0	0	1
0	1	0
1	0	0
1	1	0

Circuital symbols:



NAND and NOR are *commutative* but not *associative*.

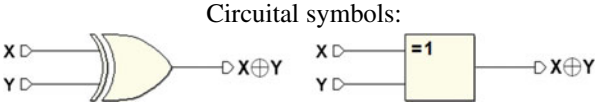
XOR (Exclusive OR)

The XOR operation is said "anticoincidence" (it provides 1 when the inputs are different):

$$X \oplus Y = X \text{ xor } Y = X\,\overline{Y} + \overline{X}\,Y$$

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

Circuital symbols:



The XOR is *commutative* and *associative*. If we negate its output, we obtain the "coincidence" function (equivalence of inputs):

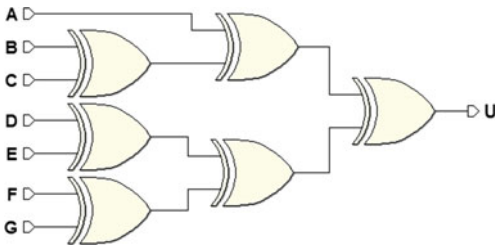
$$\overline{X \oplus Y} = X\,Y + \overline{X}\,\overline{Y}$$

Generalized XOR

Is a multiple inputs XOR, written thus:

$$X_1 \oplus X_2 \oplus \ldots \oplus X_n = \begin{cases} 1 \text{ if there is an odd number of inputs} = 1 \\ 0 \text{ if there is an even number of inputs} = 1 \end{cases}$$

They are made with the typical structure of the XOR *tree*:



1.8 Functionally Complete Operation Sets

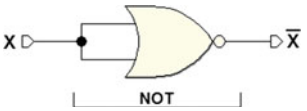
We have seen that Boolean algebra is based on a set of two elements {0, 1} and a set of operations: OR, AND, NOT. Also, De Morgan's Theorem shows that one of the two AND or OR operations can be considered superfluous and the sets of {OR, NOT} or {AND, NOT} are a sufficient basis to construct all of Boolean algebra. Let's broaden the subject by discussing other sets of operations that allows to construct Boolean algebra (named, for this reason, *Functionally Complete Operation Sets*):

1. {AND, OR, NOT}
2. {NOR}
3. {NAND}
4. {OR, NOT}
5. {AND, NOT}
6. {EXOR, AND}
6. {EXOR, OR}

Note: in practice, only {NOR} and {NAND} sets are used.

{NOR} Set

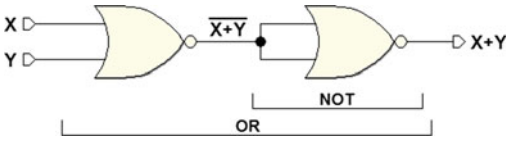
We can obtain OR and NOT from NOR gates. If we connect a NOR as in the figure below, we obtain a NOT. Given that the X and Y inputs are connected together, we obtain the following from the NOR table:



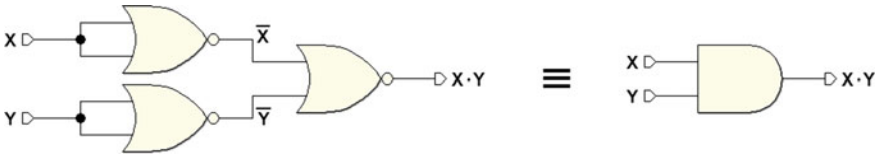
In fact:

X	Y	X nor Y
0	0	1
1	1	0

However, we obtain the OR gate by negating the NOR output with a NOT:

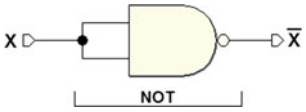


To obtain the AND, we apply De Morgan: $X \cdot Y = \overline{\overline{X} + \overline{Y}}$. We have:



{NAND} Set

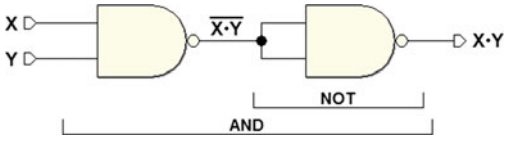
Similar to the above, the NOT is obtained as follows, taking into account the two lines of the NAND table where the two X and Y inputs are equal:



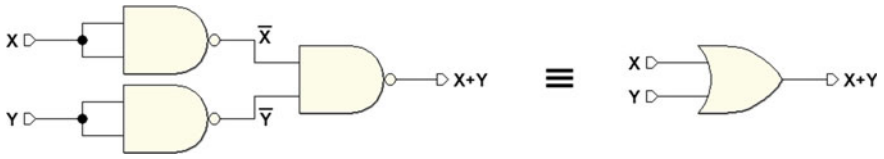
In fact:

X	Y	X nand Y
0	0	1
1	1	0

Therefore, to obtain the AND, it is sufficient to connect the NAND to a NOT made with a NAND.



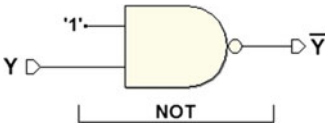
Finally, by De Morgan, we obtain the OR:



Take note: there is another way to obtain the NOT by the NAND. By connecting one of the inputs to the constant $X = 1$:

X	Y	X nand Y
0	0	1
1	1	0

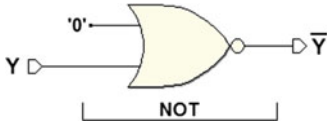
we obtain:



Similarly, if we posit $X = 0$ for the NOR we get:

X	Y	X nor Y
0	0	1
0	1	0

we obtain:



{OR, NOT} Set

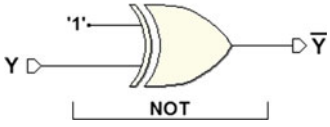
The AND is obtained by De Morgan's Theorem.

{AND, NOT} Set

The OR is obtained by De Morgan's Theorem.

{XOR, AND} Set

The NOT is obtained by the XOR as follows:



From the XOR truth table, we get:

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

positing $X = 1$:

X	Y	$X \oplus Y$
1	0	1
1	1	0

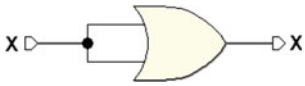
Note: If we change the constant $X = 1$ in 0, we get the identity. Therefore, we obtain an inverting/identity function, "programmable" by the input X.

{XOR, OR} Set

The NOT is obtained through the XOR and the AND by using De Morgan's Theorem.

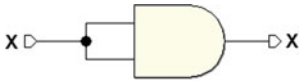
Identity

Identity can be obtained in the following ways:

from **OR**: 

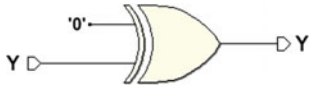
In fact:

X	Y	$X + Y$
0	0	0
1	1	1

from **AND**: 

In fact:

X	Y	$X \cdot Y$
0	0	0
1	1	1

from **XOR**: 

In fact:

X	Y	$X \oplus Y$
0	0	0
0	1	1

1.9 Shannon's Expansion Theorem

First Form

A Boolean function can be broken down this way:

$$f(X_1, X_2, X_3, \ldots, X_n) = \overline{X_1} \cdot f(0, X_2, X_3, \ldots, X_n) + X_1 \cdot f(1, X_2, X_3, \ldots, X_n)$$

The first of the two terms obtained is equivalent to the starting function but only when $X_1 = 0$, so it is conditioned by $\overline{X_1}$. Likewise, the second term, which applies to $X_1 = 1$, is conditioned by X_1.

Now that we have seen the process, we can extract all the variables of the function:

$$\begin{aligned} f(X_1, X_2, X_3, \ldots, X_n) = {} & \overline{X_1} \cdot \overline{X_2} \cdot f(0, 0, X_3, \ldots, X_n) + \\ & \overline{X_1} \cdot X_2 \cdot f(0, 1, X_3, \ldots, X_n) + \\ & X_1 \cdot \overline{X_2} \cdot f(1, 0, X_3, \ldots, X_n) + \\ & X_1 \cdot X_2 \cdot f(1, 1, X_3, \ldots, X_n) = \\ = {} & \ldots \end{aligned}$$

In the end, every $f(0, 1, \ldots)$-type entry will turn out to be a (0 or 1) constant. From an n-variable function, we obtain 2^n product terms in OR, where each term consists of all the direct and negated variables.

As an example, let's break down a $f(C, B, A)$ into three variables:

$$\begin{aligned} f(C, B, A) = {} & \overline{C}\ \overline{B}\ \overline{A} \cdot f(0, 0, 0) + \\ & \overline{C}\ \overline{B}\ A \cdot f(0, 0, 1) + \\ & \overline{C}\ B\ \overline{A} \cdot f(0, 1, 0) + \\ & \overline{C}\ B\ A \cdot f(0, 1, 1) + \\ & C\ \overline{B}\ \overline{A} \cdot f(1, 0, 0) + \\ & C\ \overline{B}\ A \cdot f(1, 0, 1) + \\ & C\ B\ \overline{A} \cdot f(1, 1, 0) + \\ & C\ B\ A \cdot f(1, 1, 1) \end{aligned}$$

The expanded form of the function is called *sum of products*, or *first canonical form*, or *AND–OR form*.

Example

We want to derive the analytical expression from a Boolean function $f(C, B, A)$ defined through the truth table using the first form of the theorem.

C	B	A	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$$\Rightarrow \quad \begin{aligned} f = {} & \overline{C}\ \overline{B}\ \overline{A} \cdot f(0, 0, 0) + \\ & \overline{C}\ \overline{B}\ A \cdot f(0, 0, 1) + \\ & \overline{C}\ B\ \overline{A} \cdot f(0, 1, 0) + \\ & \overline{C}\ B\ A \cdot f(0, 1, 1) + \\ & C\ \overline{B}\ \overline{A} \cdot f(1, 0, 0) + \\ & C\ \overline{B}\ A \cdot f(1, 0, 1) + \\ & C\ B\ \overline{A} \cdot f(1, 1, 0) + \\ & C\ B\ A \cdot f(1, 1, 1) \end{aligned} = \begin{aligned} & \overline{C}\ \overline{B}\ \overline{A} \cdot 0 + \\ & \overline{C}\ \overline{B}\ A \cdot 1 + \\ & \overline{C}\ B\ \overline{A} \cdot 0 + \\ & \overline{C}\ B\ A \cdot 0 + \\ & C\ \overline{B}\ \overline{A} \cdot 0 + \\ & C\ \overline{B}\ A \cdot 1 + \\ & C\ B\ \overline{A} \cdot 1 + \\ & C\ B\ A \cdot 0 \end{aligned} = \begin{aligned} & \overline{C}\ \overline{B}\ A + \\ & C\ \overline{B}\ A + \\ & C\ B\ \overline{A} \end{aligned}$$

Beginning by the definition, we substitute all the $f(...)$-type constants with the real value of the function, taken directly from the truth table. We observe that in the logical sum, the terms with 0 in AND can be omitted since they are always 0. We then simplify the remaining terms corresponding to the lines with output 1 and eliminate the product for the constant. The remaining expression is what we're looking for. It analytically expresses the behavior of the function in the *first canonical form*.

Second Form

The second form allows us to break down a function f into a product of sums:

$$\begin{aligned} f(X_1, X_2, X_3, \ldots, X_n) = {} & (X_1 + f(0, X_2, X_3, \ldots, X_n)) \cdot \\ & (\overline{X_1} + f(1, X_2, X_3, \ldots, X_n)) \end{aligned}$$

The first sum term is equivalent to the starting function after substituting $X_1 = 0$ and applies if $X_1 = 0$ (otherwise the whole term is 1). The second sum term is equivalent

to the starting function after substituting $X_1 = 1$ and applies if $X_1 = 1$ (otherwise the whole term is 1). In other words, for a certain value of X_1, one of the two terms is always 1, while the other assumes the value of the function.

If we go through the break-down procedure until we exhaust all the f argument variables, we obtain 2^n terms in AND. Each term is composed of the logical sum of all the direct or negated variables and the value of the function for that specific combination:

$$\begin{aligned} f(X_1, X_2, X_3, \ldots, X_n) = &(X_1 + X_2 + f(0, 0, X_3, \ldots, X_n)) \cdot \\ &(X_1 + \overline{X_2} + f(0, 1, X_3, \ldots, X_n)) \cdot \\ &(\overline{X_1} + X_2 + f(1, 0, X_3, \ldots, X_n)) \cdot \\ &(\overline{X_1} + \overline{X_2} + f(1, 1, X_3, \ldots, X_n)) = \\ = &\ldots \end{aligned}$$

Consider a three-variable function $f(C, B, A)$; we obtain 2^3 OR terms in AND:

$$\begin{aligned} f(C, B, A) = &(C + B + A + f(0, 0, 0)) \cdot \\ &(C + B + \overline{A} + f(0, 0, 1)) \cdot \\ &(C + \overline{B} + A + f(0, 1, 0)) \cdot \\ &(C + \overline{B} + \overline{A} + f(0, 1, 1)) \cdot \\ &(\overline{C} + B + A + f(1, 0, 0)) \cdot \\ &(\overline{C} + B + \overline{A} + f(1, 0, 1)) \cdot \\ &(\overline{C} + \overline{B} + A + f(1, 1, 0)) \cdot \\ &(\overline{C} + \overline{B} + \overline{A} + f(1, 1, 1) \end{aligned}$$

A function expanded this way takes on the *second canonical form*, or *product of sums*, or *OR–AND form*. Any Boolean function can be expressed this way. For a function with n-variables, we obtain 2^n factors to multiply.

Example

Through the second form of Shannon's Expansion Theorem, we derive the analytical expression of a Boolean function $f(C, B, A)$, given the truth table that describes it.

C	B	A	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$\Rightarrow$

$$\begin{aligned} f = &(C + B + A + f(0, 0, 0)) \cdot \\ &(C + B + \overline{A} + f(0, 0, 1)) \cdot \\ &(C + \overline{B} + A + f(0, 1, 0)) \cdot \\ &(C + \overline{B} + \overline{A} + f(0, 1, 1)) \cdot \\ &(\overline{C} + B + A + f(1, 0, 0)) \cdot \\ &(\overline{C} + B + \overline{A} + f(1, 0, 1)) \cdot \\ &(\overline{C} + \overline{B} + A + f(1, 1, 0)) \cdot \\ &(\overline{C} + \overline{B} + \overline{A} + f(1, 1, 1) \end{aligned}$$

$=$

$$\begin{aligned} f = &(C + B + A + 1) \cdot \\ &(C + B + \overline{A} + 1) \cdot \\ &(C + \overline{B} + A + 0) \cdot \\ &(C + \overline{B} + \overline{A} + 0) \cdot \\ &(\overline{C} + B + A + 0) \cdot \\ &(\overline{C} + B + \overline{A} + 1) \cdot \\ &(\overline{C} + \overline{B} + A + 1) \cdot \\ &(\overline{C} + \overline{B} + \overline{A} + 1) \end{aligned}$$

We have applied the definition of the second form of the theorem by substituting all the $f(...)$-type constants with the values of the truth table. Given that the constant term 1 in a logical sum absorbs the other variables and that adding 0 is redundant, the function's final expression is:

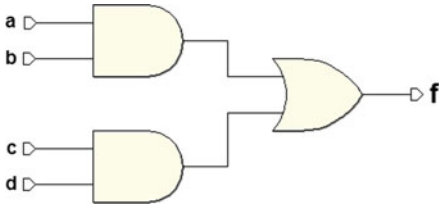
$$f = (C + \overline{B} + A) \cdot (C + \overline{B} + \overline{A}) \cdot (\overline{C} + B + A)$$

The expression expresses how the function behaves in the *second canonical form.*

1.10 Level of Boolean Expressions

The level is the maximum number of cascading operations made on the input variables. For example:

$f = a + b$ is a one-level expression
$f = ab + c$ is a two-level expression
$f = ab + cd$ is a two-level expression



Take note: the levels are important for technical reasons. The more levels there are, the longer the delays; we will focus mainly on syntheses of two-level networks. When the number of levels has to be computed, we suppose all the input variables and their complemented forms to be available. Thus, the expression $f = \overline{a}b + \overline{c}d$ is a two-level Boolean expression.

1.11 Literals

Literals are the number of input variables that make up a Boolean expression (not to be confused with the number of variables).
For example: if $f(a, b)$ is a logical function with two binary variables a and b:

$f = a + b$ has 2 literals
$f = ab + \overline{a}b$ has 4 literals.

1.12 Minterms

If an AND term in a Boolean expression contains all the direct or negated variables in the entire expression, it is called a *fundamental product*, or *minterm*. For example:

$$f(X_1, X_2, X_3) = X_1 \cdot X_2 \cdot \overline{X_3} \qquad \text{is a minterm.}$$

An n-variable function has 2^n minterms since every variable in the function must be part of a minterm, in its direct or negated form. Note that among all the possible combinations of variables, there is only one for which a certain minterm equals 1 (e.g., $X_1 \cdot X_2 \cdot \overline{X_3} = 1$ if and only if $X_1 = 1$, $X_2 = 1$, $X_3 = 0$).

1.13 Maxterms

If an OR term in a Boolean expression contains all the direct or negated variables in the entire expression, it is called a *fundamental sum*, or *maxterm*. As above, if there are n-variables, there are 2^n maxterms. For example:

$$f(X_1, X_2, X_3) = X_1 + \overline{X_2} + X_3 \qquad \text{is a maxterm.}$$

Remember that there is only one combination of variables for which a certain maxterm equals zero (e.g., $X_1 + \overline{X_2} + X_3 = 0$ if and only if $X_1 = 0$, $X_2 = 1$, $X_3 = 0$).

1.14 Implicants

Given the Boolean expressions f and g, g is an implicant of f: g implies f ($g \Rightarrow f$) or f covers g ($f \supset g$) if f always $= 1$ when $g = 1$.

$$\text{In this example: } f(X, Y, Z) = XY + Z \text{ we have } \begin{array}{r} XY \Rightarrow f \\ Z \Rightarrow f \end{array}$$

Every time Z and/or XY equal 1, f also equals 1. XY and Z are therefore implicants of f. X does not imply f: in fact if X equals 1 f does not necessarily equal 1.

1.15 Prime Implicants

g is a *prime* implicant of f if:

- $g \Rightarrow f$ ($f \supset g$);

- g is not covered by another implicant with fewer literals.

In other words, an implicant is prime if it equals 1 when no other implicant equals 1. An implicant, which *is not* prime, can be removed from the expression since it is superfluous. In the example:

$$f = XY + X + Z$$

X and Z are prime implicants while XY is a non-prime implicant of f. In fact, to have $XY = 1$ it is necessary that $X = 1$, but if $X = 1$ then $f = 1$ anyway, since X already implies f ($X \supset XY$).

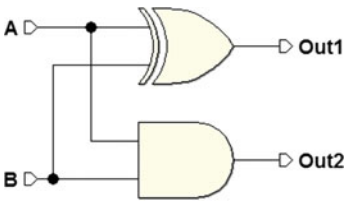
1.16 Combinational Networks

A *combinational network* is defined as a logical circuit whose output depends only on the combination of its inputs. In Chap. 5, we will discuss *sequential networks* whose output does not only depend on the values of the inputs in that time but also on the "inputs history." In other words, we will see that these networks have memory capacity.

A combinational network can be described in terms of a Boolean function. Note some examples of combinational networks.

1.16.1 Example: Logical Network Analysis

We want to analyze the following circuit, obtaining its truth table:



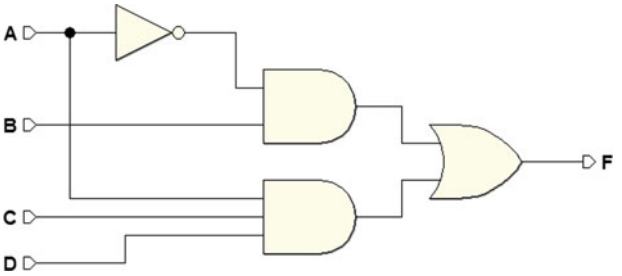
It is composed of two known gates, so we can complete the truth table directly:

A	B	$Out1$	$Out2$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

We will find this circuit again in Chap. 2, being an arithmetic circuit.

1.16.2 Example: Two-Level Logical Network Analysis

As above, we will analyze the circuit, compiling its truth table:



By analyzing all the circuit paths, we must determine the output value F for all the combination of A, B, C, and D. It is useful to include the intermediate outputs in the truth table. The result of the analysis is:

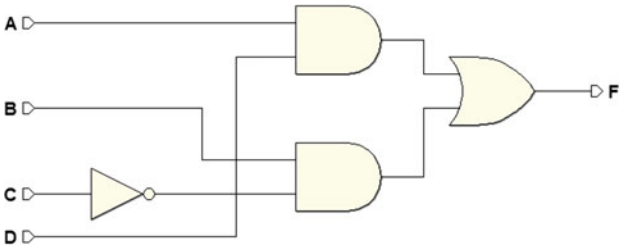
A	B	C	D	$\overline{A}$	$\overline{A}\,B$	$A\,C\,D$	F
0	0	0	0	1	0	0	0
0	0	0	1	1	0	0	0
0	0	1	0	1	0	0	0
0	0	1	1	1	0	0	0
0	1	0	0	1	1	0	1
0	1	0	1	1	1	0	1
0	1	1	0	1	1	0	1
0	1	1	1	1	1	0	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	0	1	1
1	1	0	0	0	0	0	0
1	1	0	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	1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

1.16.3 Example: Circuit Schematic of a Logical Network (1)

We want to draw the logical schematic of a circuit, given its Boolean expression:

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

The result is:

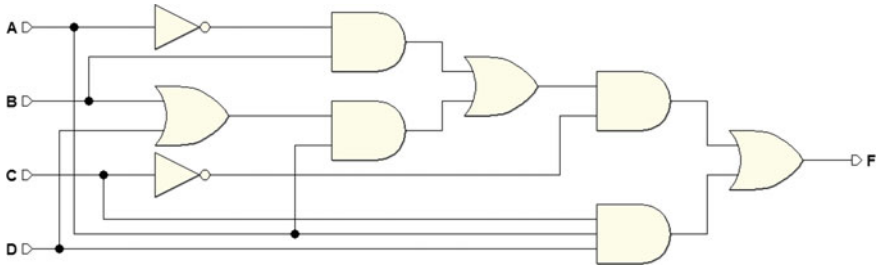


1.16.4 Example: Circuit Schematic of a Logical Network (2)

If we draw the logical schematic of the circuit, given this Boolean expression:

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

The result is a five-level network:



For practice, if we derive the truth table of this circuit and the one of the previous example, we see that the truth tables are identical! Perfect Induction shows that the two networks are equivalent. We can also prove that they are equivalent through the properties of Boolean algebra, as follows:

$$\begin{aligned}
F &= A\,D\,C + \overline{C}\,(\overline{A}\,B + A\,(B + D)) = \\
&= A\,D\,C + \overline{C}\,(\overline{A}\,B + A\,B + A\,D) = \\
&= A\,D\,C + \overline{C}\,(B\,(\overline{A} + A) + A\,D) = \\
&= A\,D\,C + \overline{C}\,(B + A\,D) = A\,D\,C + A\,D\,\overline{C} + \overline{C}\,B = \\
&= A\,D\,(C + \overline{C}) + \overline{C}\,B = A\,D + \overline{C}\,B.
\end{aligned}$$

1.16.5 Example: Defining the Behavior of a Logical Network

We want to define the truth table of a combinational network with three inputs A, B, and C: output F must assume value 1 when the number of input 1s is odd.

First, we prepare the truth table (bottom left). Then, line by line, we count the 1*s* and write the value of F based on the given definition. For example, in the last line

we count three (an odd number of) 1s so we insert a 1 in the output column. In the end, we get the table at the right.

A	B	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	.

$\rightarrow$

A	B	C	F
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

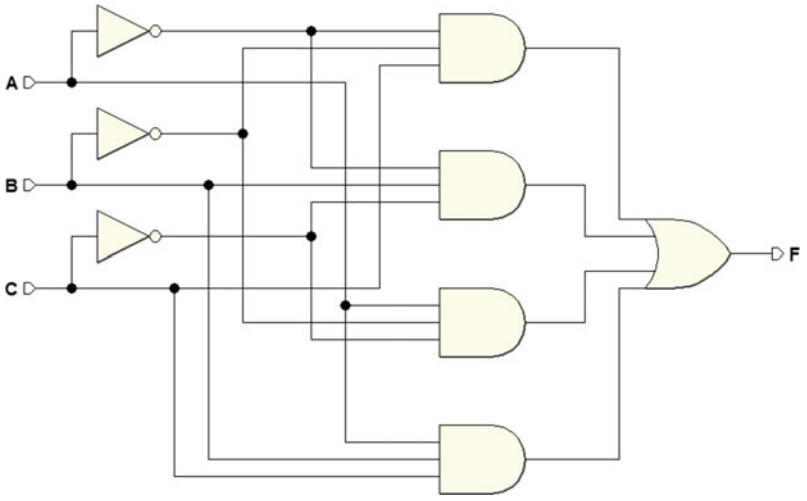
1.16.6 Example: Circuit Schematic from the Truth Table

Let's examine the table derived from the last example and start by writing the Boolean expression of the function using the first form (AND–OR) of Shannon's Theorem. Then, we'll draw its circuit schematic.

We obtain the following from the table:

$$F = \overline{A}\ \overline{B}\ C + \overline{A}\ B\ \overline{C} + A\ \overline{B}\ \overline{C} + A\ B\ C$$

The expression defines four three-input ANDs that merge into one single four-input OR. After the components of this schematic are designed and interconnected, we will connect inputs A, B, and C to the ANDs, taking any negations into account. In the end, we get:



1.16.7 Example: Controlling a Heating System

Designing the control circuit of a heating system.

The system is composed of a thermostat, a heater, and a switch. T, I, and C are Boolean variables: the first two are inputs, the last, an output.

Defining the Variables

Given t_0 as a certain threshold temperature, the thermostat indicates that we are above or below it. Thus, we suppose that:

$$T = 0 \qquad (\text{if } t \geq t_0) \qquad T = 1 \qquad (\text{if } t < t_0)$$

The heater could be ON or OFF:

$$C = 0 \qquad (\text{heater OFF}) \qquad C = 1 \qquad (\text{heater ON})$$

The same goes for the switch, which could be ON or OFF:

$$I = 0 \qquad (\text{switch OFF}) \qquad I = 1 \qquad (\text{switch ON})$$

Defining the Network Operation

For the heater to be ON ($C = 1$) it must be $t < t_0$ ($T = 1$) and the switch must be ON ($I = 1$).

Let's translate this *verbal description* into the truth table:

- if I is OFF $\Rightarrow$ heater is OFF ($C = 0$)
- if I is ON but ($t \geq t_0$) $\Rightarrow$ heater is OFF ($C = 0$)
- if I is ON and ($t < t_0$) $\Rightarrow$ heater is ON ($C = 1$)

$\Rightarrow$

I	T	C
0	0	0
0	1	0
1	0	0
1	1	1

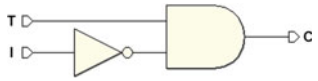
Synthesis

Synthesis is the process that allows us to find the logical expression (and the network schematic) from a truth table. The resulting logical expression is $C = I \cdot T$, and the circuit representation is:

Note: the values assigned to the three variables to represent the different physical conditions are arbitrary. For example, we would be able to define I and C as before and T as follows:

$$\begin{array}{l} T = 0 \text{ se } t < t_0 \\ T = 1 \text{ se } t \geq t_0 \end{array} \qquad \text{gives us:} \qquad \begin{array}{cc|c} I & T & C \\ \hline 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{array}$$

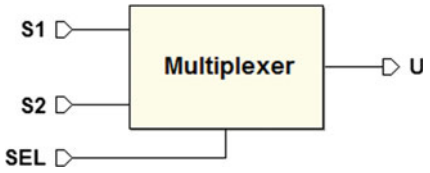
Notice that $C = 1$ if and only if $I = 1$ and $T = 0$. Here, the logical expression is $C = I \cdot \overline{T}$. The new circuit representation is:



In the two cases, both $C = I \cdot T$ and $C = I \cdot \overline{T}$ are minterms. There is one only combination of I and T whose product is $C = 1$. In general, to synthesize a network with just one 1 in the output, we take the minterm of the unique combination of variables that give the output 1. We do this by inserting the variables that equal 1 in that specific combination into the *direct* form and those that equal 0 into the *negated* form.

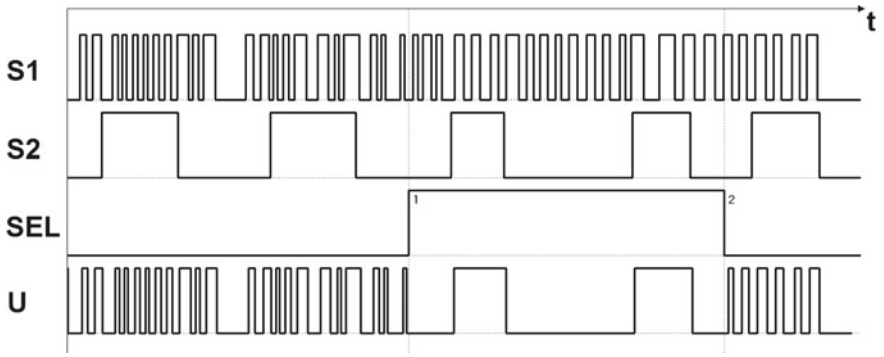
1.16.8 Example: Two Channels Multiplexer (Selector)

This is a system that provides a Boolean output variable (U) that copies one of the two possible inputs ($S1$, $S2$), depending on the value of a control variable (SEL).



Before defining the problem in Boolean terms, let's draw the timing diagram of the possible $S1s$, $S2s$, and $SELs$ over time and see how U should change.

Let's represent $S1$ and $S2$ as *digital signals*, that is, sequences of $0s$ and $1s$ that vary over time and encode information such as a phone call or a TV program, according to a certain standard. Without concerning ourselves with the type of code, let's use SEL to select which of the signals will be routed to output U.



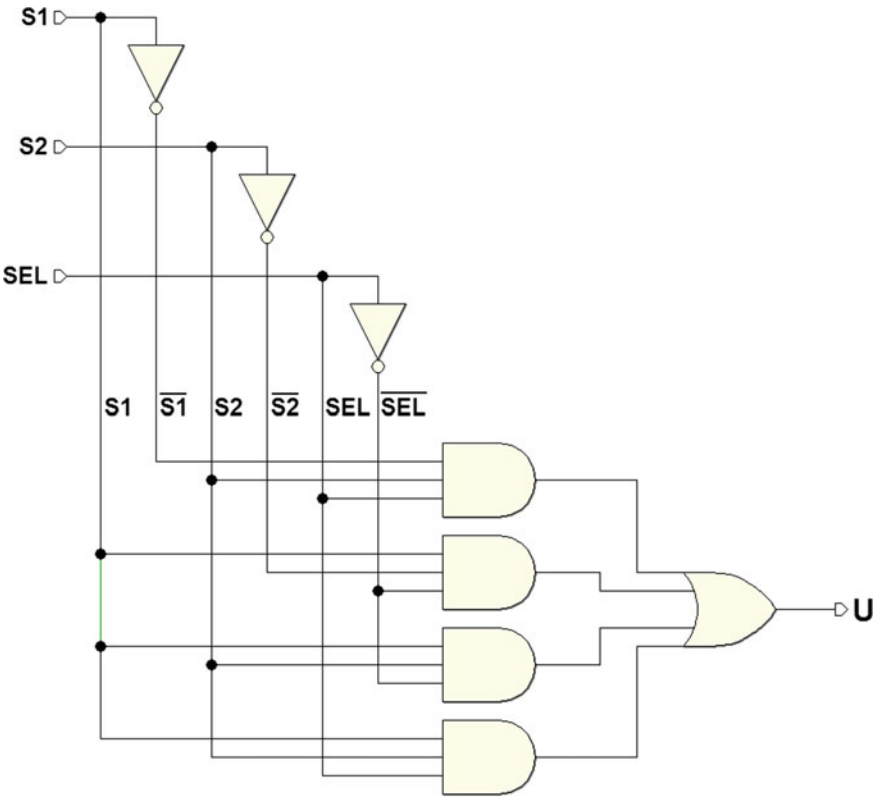
From the image above, we see that U needs to assume the values of $S1$ if $SEL = 0$, or of $S2$ if $SEL = 1$. Thus, it is possible to define the truth table. Notice that the table is valid "moment by moment," and does not describe the history of the network operations:

SEL	$S1$	$S2$	U	
0	0	0	0	
0	0	1	0	
0	1	0	1	(a)
0	1	1	1	(b)
1	0	0	0	
1	0	1	1	(c)
1	1	0	0	
1	1	1	1	(d)

Let's apply the first form of Shannon's Expansion Theorem: meaning, let's do an AND–OR canonical synthesis. For all the $1s$ in the output column, let's write the corresponding minterms, merging them in a OR:

$$\begin{aligned} U = {} & \overline{SEL} \cdot S1 \cdot \overline{S2} + \quad \text{(a)} \\ & \overline{SEL} \cdot S1 \cdot S2 + \quad \text{(b)} \\ & SEL \cdot \overline{S1} \cdot S2 + \quad \text{(c)} \\ & SEL \cdot S1 \cdot S2 \qquad \text{(d)} \end{aligned}$$

This is the circuit schematic, including the NOTs (three suffice):



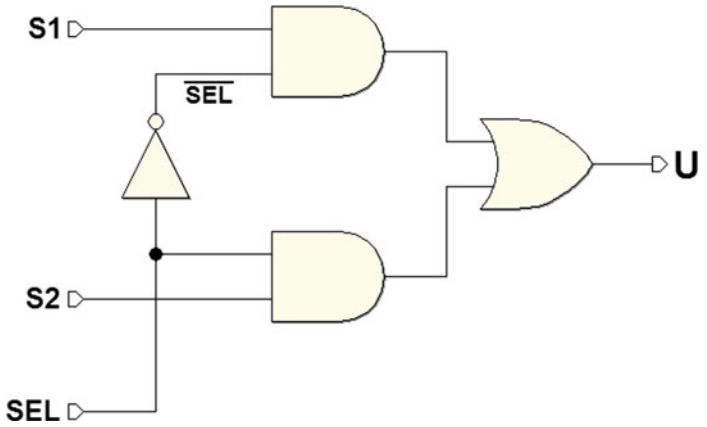
Notice that every time we see one of the four combinations that the table assigns at 1, the corresponding minterm (and only that) generates a 1. The OR makes it so that the output U goes to 1 for all four combinations above.

Minimizing

The AND/OR canonical form that we derived can be minimized by using the properties of Boolean algebra. We need to minimize the *number of literals* of the expression that describes the network. Let's see the following:

$$\begin{aligned} U &= \overline{SEL} \cdot S1 \cdot \overline{S2} \; + \; \overline{SEL} \cdot S1 \cdot S2 \; + \; SEL \cdot \overline{S1} \cdot S2 \; + \; SEL \cdot S1 \cdot S2 \; = \\ &= (S1 \cdot \overline{S2} \; + \; S1 \cdot S2) \cdot \overline{SEL} \; + \; (\overline{S1} \cdot S2 \; + \; S1 \cdot S2) \cdot SEL \; = \\ &= ((\overline{S2} + S2) \cdot S1) \cdot \overline{SEL} \; + \; ((\overline{S1} + S1) \cdot S2) \cdot SEL \; = \\ &= S1 \cdot \overline{SEL} + S2 \cdot SEL \end{aligned}$$

Take into account that $(S1 + \overline{S1}) = 1$ and $(\overline{S2} + S2) = 1$. Originally the expression had 12 literals; now it only has four. See the logical schematic below. The network's complexity is markedly reduced:



The final expression:

$$U = S1 \cdot \overline{SEL} + S2 \cdot SEL$$

can also be reinterpreted intuitively. It simply says that output U copies $S1$ when SEL is 0 (thus $\overline{SEL} = 1$), while it copies $S2$ when SEL is 1.

Note: we went through the analytical process of minimization for practice. To reduce the complexity of a Boolean function, we can use simpler methods like the "maps" that we will see in Chap. 2.

1.17 Exercises

1. Negate the following term and transform it into a four-term logical sum.

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

2. Negate the following term and then transform it into a single product term.

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

3. Using the theorems of Boolean algebra, minimize the following logical expression:

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

4. Using the theorems of Boolean algebra, it is possible to prove that the following expression equals 1 only when A and B are contemporaneously at 1 or when D is at 1 and C is contemporaneously at 0:

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

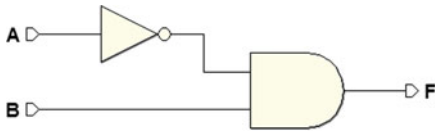
5. Minimize the following logical function with the criteria of Boolean algebra.

$$Y = \overline{A}\,(A + B) + \overline{C} + B\,C$$

6. Using De Morgan's theorem, minimize the following logical function:

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

7. Design the circuit that implements the expression $\overline{A}\,B + A\,\overline{B}\,\overline{C}\,(B + C)$, and verify that it is equivalent to the following network:



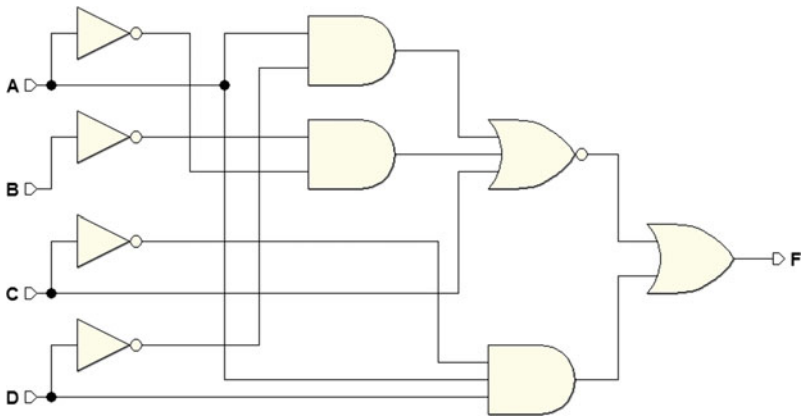
8. Design the circuits corresponding to the following logical expressions:

   (a) $C = (A + B) + \overline{A\,B}$
   (b) $D = A\,(B + C)\,\overline{C}$
   (c) $E = \overline{A}\,D\,\overline{(C + \overline{D})}$
   (d) $G = \overline{A}\,B\,\overline{C} + D\,(C + A\,\overline{B}\,C)$

9. Do the following conversions between the canonical forms:

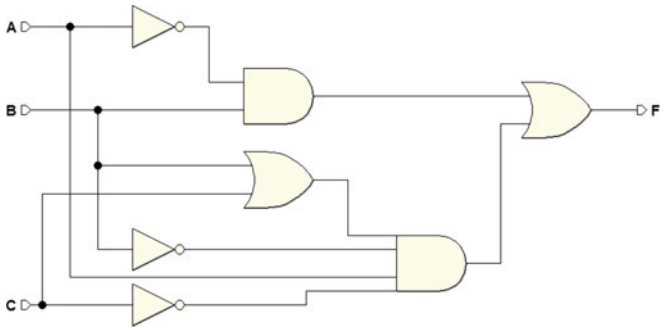
   (a) $F = \overline{A}\,B\,\overline{C} + \overline{A}\,B\,C + A\,\overline{B}\,C + A\,B\,\overline{C}$ to the OR–AND form.
   (b) $G = (\overline{A} + B + C)(\overline{A} + \overline{B} + \overline{C})(A + B + C)$ to the AND–OR form.

10. Analytically derive the logical function of the following circuit:



1.18 Solutions

1. $\overline{\overline{A}\,B\,\overline{C}\,D} = \overline{\overline{A}} + \overline{B} + \overline{\overline{C}} + \overline{D} = A + \overline{B} + C + \overline{D}$
2. $\overline{\overline{A} + \overline{B} + C} = \overline{\overline{A}}\,B\,\overline{C} = A\,B\,\overline{C}$
3. $A + AB + CB + C\overline{B} = A(1 + B) + C(B + \overline{B}) = A + C$
4. We need to demonstrate that the expression given equals the one below:
$$F = A\,B + \overline{C}\,D$$
By minimizing the expression given we obtain:
$$\begin{aligned} F &= \overline{\overline{A}\,C\,B + \overline{A}\,C\,\overline{B} + \overline{A}\,\overline{D} + \overline{B}\,C\,D + \overline{B}\,C + \overline{B}\,\overline{D}} = \\ &= \overline{\overline{A}\,C\,(B + \overline{B}) + \overline{A}\,\overline{D} + \overline{B}\,C\,(D + 1) + \overline{B}\,\overline{D}} = \\ &= \overline{\overline{A}\,C + \overline{A}\,\overline{D} + \overline{B}\,C + \overline{B}\,\overline{D}} = \\ &= \overline{\overline{A}\,(C + \overline{D}) + \overline{B}\,(C + \overline{D})} = \\ &= \overline{(\overline{A} + \overline{B}) \cdot (C + \overline{D})} = \\ &= \overline{(\overline{A} + \overline{B})} + \overline{(C + \overline{D})} \;=\; A\,B + \overline{C}\,D \end{aligned}$$
5. We obtain:
$$\begin{aligned} Y &= \overline{A}\,(A + B) + \overline{C} + B\,C = \\ &= \overline{A}\,A + \overline{A}\,B + \overline{C} + B\,C \;=\; \overline{A}\,B + \overline{C} + B\,\overline{C} + B\,C = \\ &= \overline{A}\,B + \overline{C} + B\,(\overline{C} + C) \;=\; \overline{A}B + \overline{C} + B = \\ &= B + \overline{C} \end{aligned}$$
6. We obtain:
$$Y = \overline{A + A\overline{B} + CD} = \overline{A(1 + \overline{B}) + CD} = \overline{A + CD} = \overline{A}(\overline{C} + \overline{D})$$
7. The network is as follows:

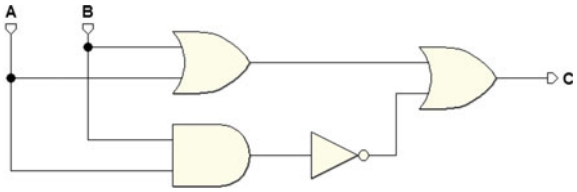


Minimizing the expression of F:

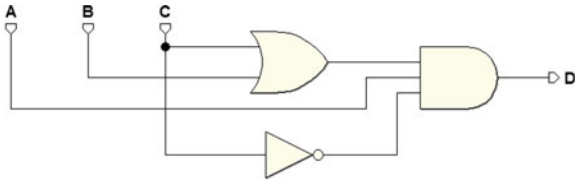
$$\begin{aligned} F &= \overline{A}\,B + A\,\overline{B}\,\overline{C}\,(B + C) = \overline{A}\,B + A\,\overline{B}\,\overline{C}\,B + A\,\overline{B}\,\overline{C}\,C = \\ &= \overline{A}\,B + A\,\overline{C}\,(\overline{B}\,B) + A\,\overline{B}\,(\overline{C}\,C) = \overline{A}\,B + A\,\overline{C}\,(0) + A\,\overline{B}\,(0) = \\ &= \overline{A}\,B \end{aligned}$$

8. These are the circuits corresponding to the expressions:

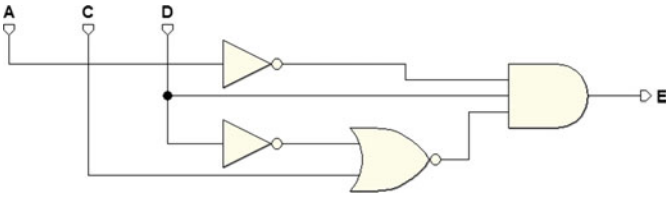
(a) $C = (A + B) + \overline{A\,B}$:



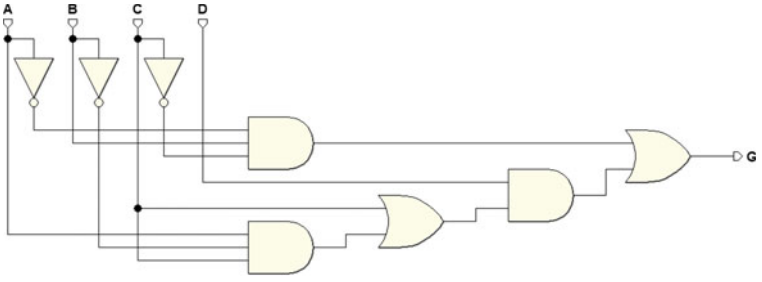
(b) $D = A\,(B + C)\,\overline{C}$:



(c) $E = \overline{A}\,D\,\overline{(C + \overline{D})}$:



(d) $G = \overline{A}\,B\,\overline{C} + D\,(C + A\,\overline{B}\,C)$:



9. (a) We derive the truth table and represent the negated output.

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

From this table, we derive the AND/OR expression of the negated function and we apply De Morgan's Theorem.

$$\begin{aligned}\overline{F} &= \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C + A\,B\,C + A\,\overline{B}\,\overline{C} \\ &= \overline{(A+B+C)(A+B+\overline{C})(\overline{A}+\overline{B}+\overline{C})(\overline{A}+B+C)} =\end{aligned}$$

Finally, we once more negate the whole expression.

$$F = (A+B+C)(A+B+\overline{C})(\overline{A}+\overline{B}+\overline{C})(\overline{A}+B+C).$$

(b) We complete the truth table, from which we will then directly derive the AND–OR canonical synthesis.

A	B	C	G
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$$G = \overline{A}\,\overline{B}\,C + \overline{A}\,B\,\overline{C} + \overline{A}\,B\,C + A\,\overline{B}\,C + A\,B\,\overline{C}$$

10. $F = A\,\overline{C}\,D + \overline{(A\,\overline{D} + \overline{A}\,\overline{B} + C)}$.