THE ESSENTIALS OF

Computer Organization and Architecture

THIRD EDITION

Linda Null
Julia Lobur

Chapter 3

Boolean Algebra and Digital Logic

Chapter 3 Objectives

- Understand the relationship between Boolean logic and digital computer circuits.
- Learn how to design simple logic circuits.

3.1 Introduction

- In the latter part of the nineteenth century, George Boole incensed philosophers and mathematicians alike when he suggested that logical thought could be represented through mathematical equations.
  - *How dare anyone suggest that human thought could be encapsulated and manipulated like an algebraic formula?*
- Computers, as we know them today, are implementations of Boole's *Laws of Thought*.
  - John Atanasoff and Claude Shannon were among the first to see this connection.

3.1 Introduction

- In the middle of the twentieth century, computers were commonly known as “thinking machines” and “electronic brains.”
  - Many people were fearful of them.
- Nowadays, we rarely ponder the relationship between electronic digital computers and human logic. Computers are accepted as part of our lives.
  - Many people, however, are still fearful of them.
- In this chapter, you will learn the simplicity that constitutes the essence of the machine.

3.2 Boolean Algebra

- Boolean algebra is a mathematical system for the manipulation of variables that can have one of two values.
  - In formal logic, these values are "true" and "false."
  - In digital systems, these values are "on" and "off," 1 and 0, or "high" and "low."
- Boolean expressions are created by performing operations on Boolean variables.
  - Common Boolean operators include AND, OR, and NOT.

3.2 Boolean Algebra

- A Boolean operator can be completely described using a truth table.
- The truth table for the Boolean operators AND and OR are shown at the right.
- The AND operator is also known as a Boolean product. The OR operator is the Boolean sum.

X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

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

3.2 Boolean Algebra

- The truth table for the Boolean NOT operator is shown at the right.
- The NOT operation is most often designated by a prime mark ($\mathbf{x'}$). It is sometimes indicated by an overbar ($\bar{\mathbf{x}}$) or an "elbow" ($\neg\mathbf{x}$).

NOT x

x	x'
0	1
1	0

3.2 Boolean Algebra

- A Boolean function has:
  - At least one Boolean variable,
  - At least one Boolean operator, and
  - At least one input from the set {0,1}.
- It produces an output that is also a member of the set {0,1}.

Now you know why the binary numbering system is so handy in digital systems.

3.2 Boolean Algebra

- The truth table for the Boolean function:

$$F(x,y,z) = xz' + y$$

is shown at the right.

- To make evaluation of the Boolean function easier, the truth table contains extra (shaded) columns to hold evaluations of subparts of the function.

$$F(x,y,z) = xz' + y$$

x	y	z	z'	xz'	xz'+ y
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

3.2 Boolean Algebra

- As with common arithmetic, Boolean operations have rules of precedence.
- The NOT operator has highest priority, followed by AND and then OR.
- This is how we chose the (shaded) function subparts in our table.

$F(x,y,z) = xz' + y$

x	y	z	z'	xz'	xz' + y
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

3.2 Boolean Algebra

- Digital computers contain circuits that implement Boolean functions.
- The simpler that we can make a Boolean function, the smaller the circuit that will result.
  – Simpler circuits are cheaper to build, consume less power, and run faster than complex circuits.
- With this in mind, we always want to reduce our Boolean functions to their simplest form.
- There are a number of Boolean identities that help us to do this.

3.2 Boolean Algebra

- Most Boolean identities have an AND (product) form as well as an OR (sum) form. We give our identities using both forms. Our first group is rather intuitive:

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0 + x = x$
Null Law	$0x = 0$	$1 + x = 1$
Idempotent Law	$xx = x$	$x + x = x$
Inverse Law	$xx' = 0$	$x + x' = 1$

3.2 Boolean Algebra

- Our second group of Boolean identities should be familiar to you from your study of algebra:

Identity Name	AND Form	OR Form
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x + (y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$

3.2 Boolean Algebra

- Our last group of Boolean identities are perhaps the most useful.
- If you have studied set theory or formal logic, these laws are also familiar to you.

Identity Name	AND Form	OR Form
Absorption Law	$x(x+y) = x$	$x + xy = x$
DeMorgan's Law	$(xy)' = x' + y'$	$(x+y)' = x'y'$
Double Complement Law	$(x)'' = x$	

3.2 Boolean Algebra

- We can use Boolean identities to simplify:

$$F(x,y,z) = xy + x'z + yz$$

$$
\begin{aligned}
F(x,y,z) &= xy + x'z + yz \\
&= xy + x'z + yz(1) && \text{(Identity)} \\
&= xy + x'z + yz(x + x') && \text{(Inverse)} \\
&= xy + x'z + (yz)x + (yz)x' && \text{(Distributive)} \\
&= xy + x'z + x(yz) + x'(zy) && \text{(Commutative)} \\
&= xy + x'z + (xy)z + (x'z)y && \text{(Associative twice)} \\
&= xy + (xy)z + x'z + (x'z)y && \text{(Commutative)} \\
&= xy(1 + z) + x'z(1 + y) && \text{(Distributive)} \\
&= xy(1) + x'z(1) && \text{(Null)} \\
&= xy + x'z && \text{(Identity)}
\end{aligned}
$$

3.2 Boolean Algebra

- Sometimes it is more economical to build a circuit using the complement of a function (and complementing its result) than it is to implement the function directly.
- DeMorgan’s law provides an easy way of finding the complement of a Boolean function.
- Recall DeMorgan’s law states:

$$(xy)' = x' + y' \quad \text{and} \quad (x + y)' = x'y'$$

3.2 Boolean Algebra

- DeMorgan’s law can be extended to any number of variables.
- Replace each variable by its complement and change all ANDs to ORs and all ORs to ANDs.
- Thus, we find the the complement of:

$$F(x,y,z) = (xy)+(x'y)+(xz')$$

is:

$$\begin{aligned} F'(x,y,z) &= ((xy)+(x'y)+(xz'))' \\ &= (xy)'(x'y)'(xz')' \\ &= (x'+y')(x+y')(x'+z) \end{aligned}$$

3.2 Boolean Algebra

- Through our exercises in simplifying Boolean expressions, we see that there are numerous ways of stating the same Boolean expression.
  - These “synonymous” forms are *logically equivalent.*
  - Logically equivalent expressions have identical truth tables.
- In order to eliminate as much confusion as possible, designers express Boolean functions in *standardized* or *canonical* form.

3.2 Boolean Algebra

- There are two canonical forms for Boolean expressions: sum-of-products and product-of-sums.
  - Recall the Boolean product is the AND operation and the Boolean sum is the OR operation.
- In the sum-of-products form, ANDed variables are ORed together.
  - For example: $F(x,y,z) = xy + xz + yz$
- In the product-of-sums form, ORed variables are ANDed together:
  - For example: $F(x,y,z) = (x+y)(x+z)(y+z)$

3.2 Boolean Algebra

- It is easy to convert a function to sum-of-products form using its truth table.
- We are interested in the values of the variables that make the function true (=1).
- Using the truth table, we list the values of the variables that result in a true function value.
- Each group of variables is then ORed together.

$F(x,y,z) = xz' + y$

x	y	z	xz' + y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

3.2 Boolean Algebra

- The sum-of-products form for our function is:

$$F(x,y,z) = xz' + y$$

$$F(x,y,z) = (x'yz') + (x'yz) + (xy'z') + (xyz') + (xyz)$$

x	y	z	xz' + y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

We note that this function is not in simplest terms. Our aim is only to rewrite our function in canonical sum-of-products form.

3.3 Logic Gates

- We have looked at Boolean functions in abstract terms.
- In this section, we see that Boolean functions are implemented in digital computer circuits called gates.
- A gate is an electronic device that produces a result based on two or more input values.
  - In reality, gates consist of one to six transistors, but digital designers think of them as a single unit.
  - Integrated circuits contain collections of gates suited to a particular purpose.

3.3 Logic Gates

- The three simplest gates are the AND, OR, and NOT gates.

X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

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

NOT X

X	X′
0	1
1	0

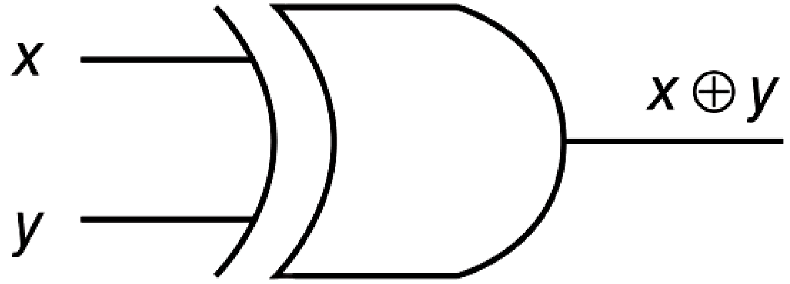
- They correspond directly to their respective Boolean operations, as you can see by their truth tables.

3.3 Logic Gates

- Another very useful gate is the exclusive OR (XOR) gate.
- The output of the XOR operation is true only when the values of the inputs differ.

X XOR **Y**

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0



Note the special symbol ⊕ for the XOR operation.

3.3 Logic Gates

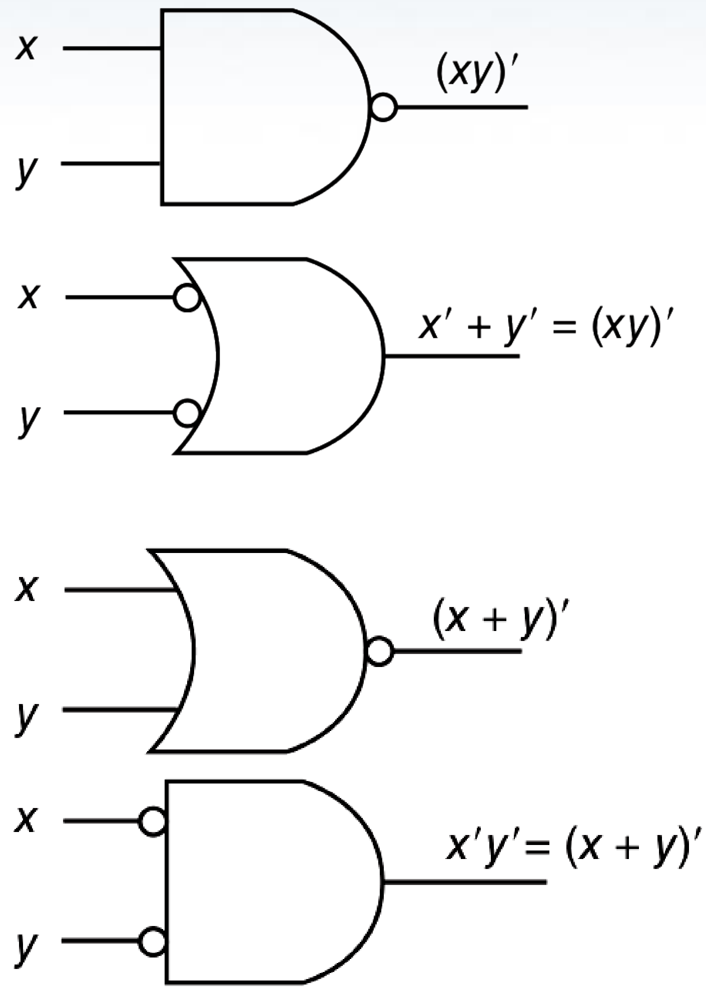
- NAND and NOR are two very important gates. Their symbols and truth tables are shown at the right.

X NAND Y

X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

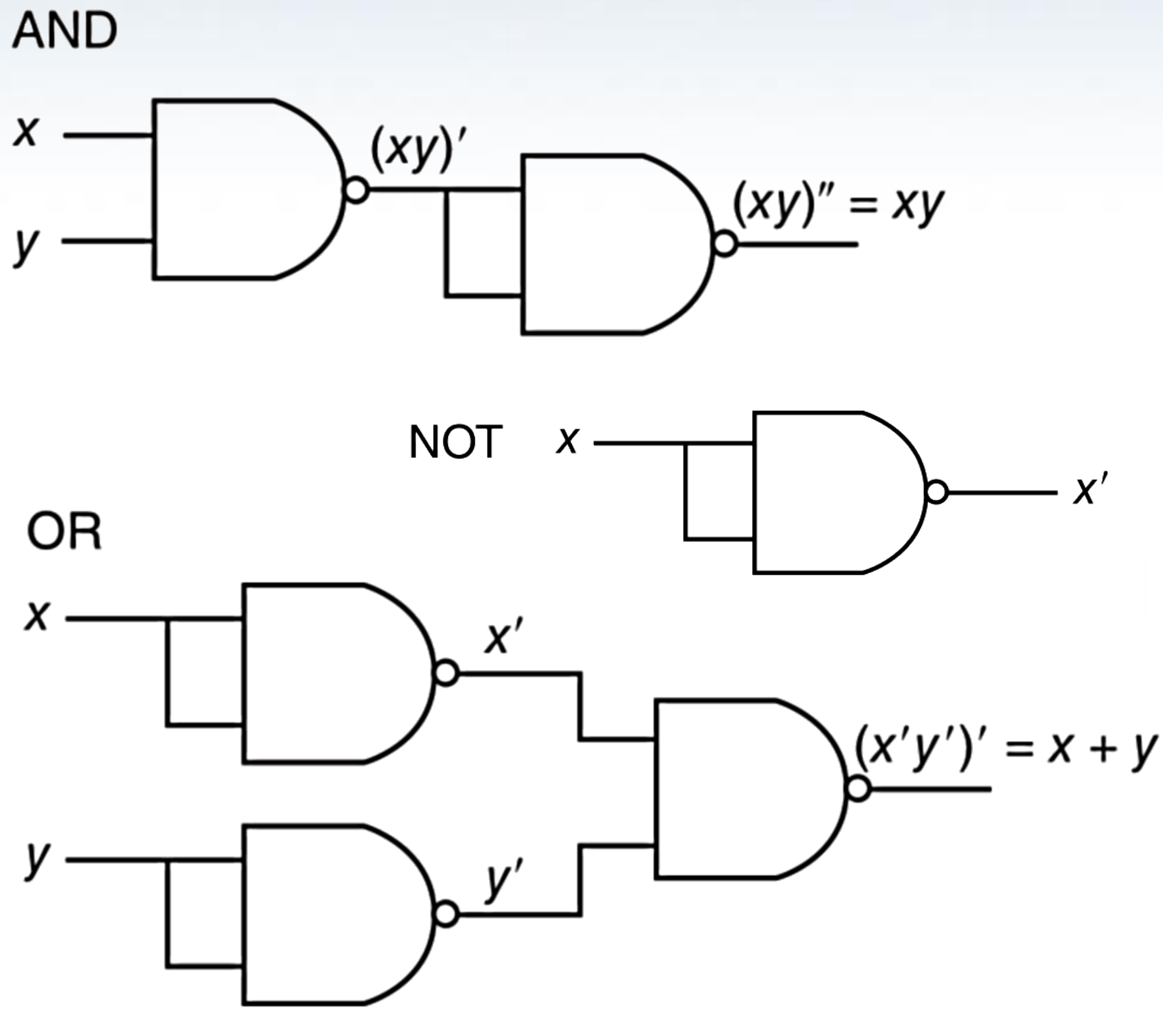
X NOR Y

X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0



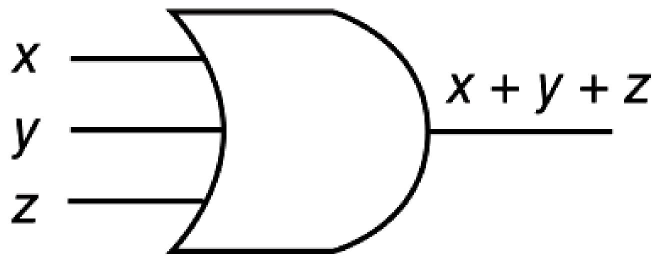
3.3 Logic Gates

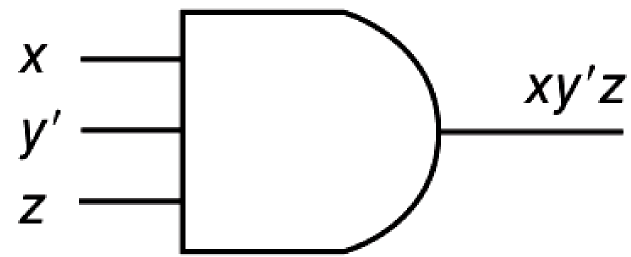
- NAND and NOR are known as *universal gates* because they are inexpensive to manufacture and any Boolean function can be constructed using only NAND or only NOR gates.

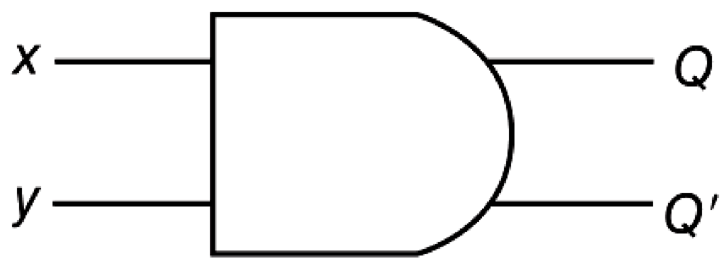


3.3 Logic Gates

- Gates can have multiple inputs and more than one output.
  - A second output can be provided for the complement of the operation.
  - We'll see more of this later.

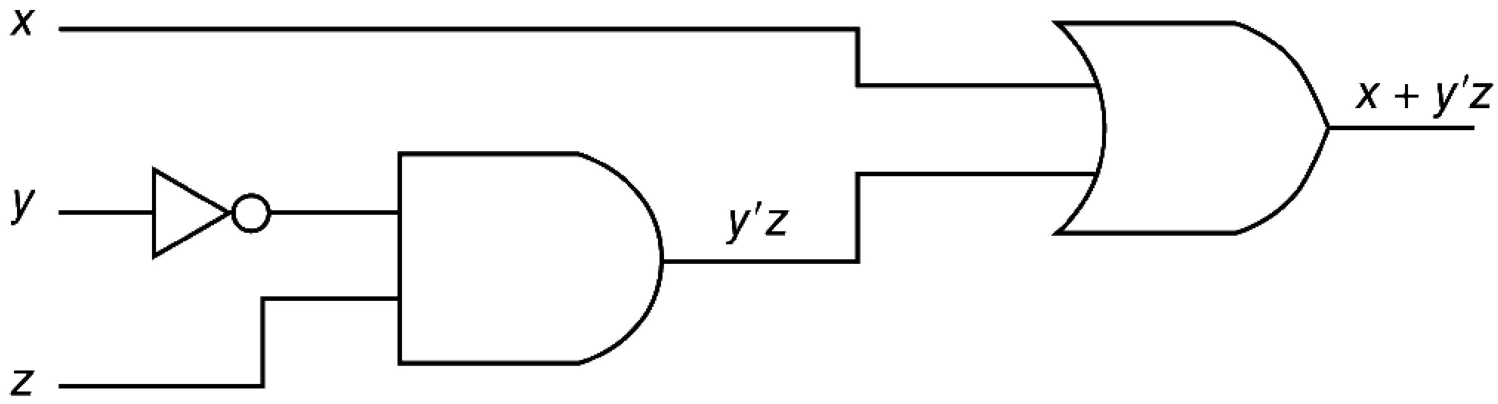






3.4 Digital Components

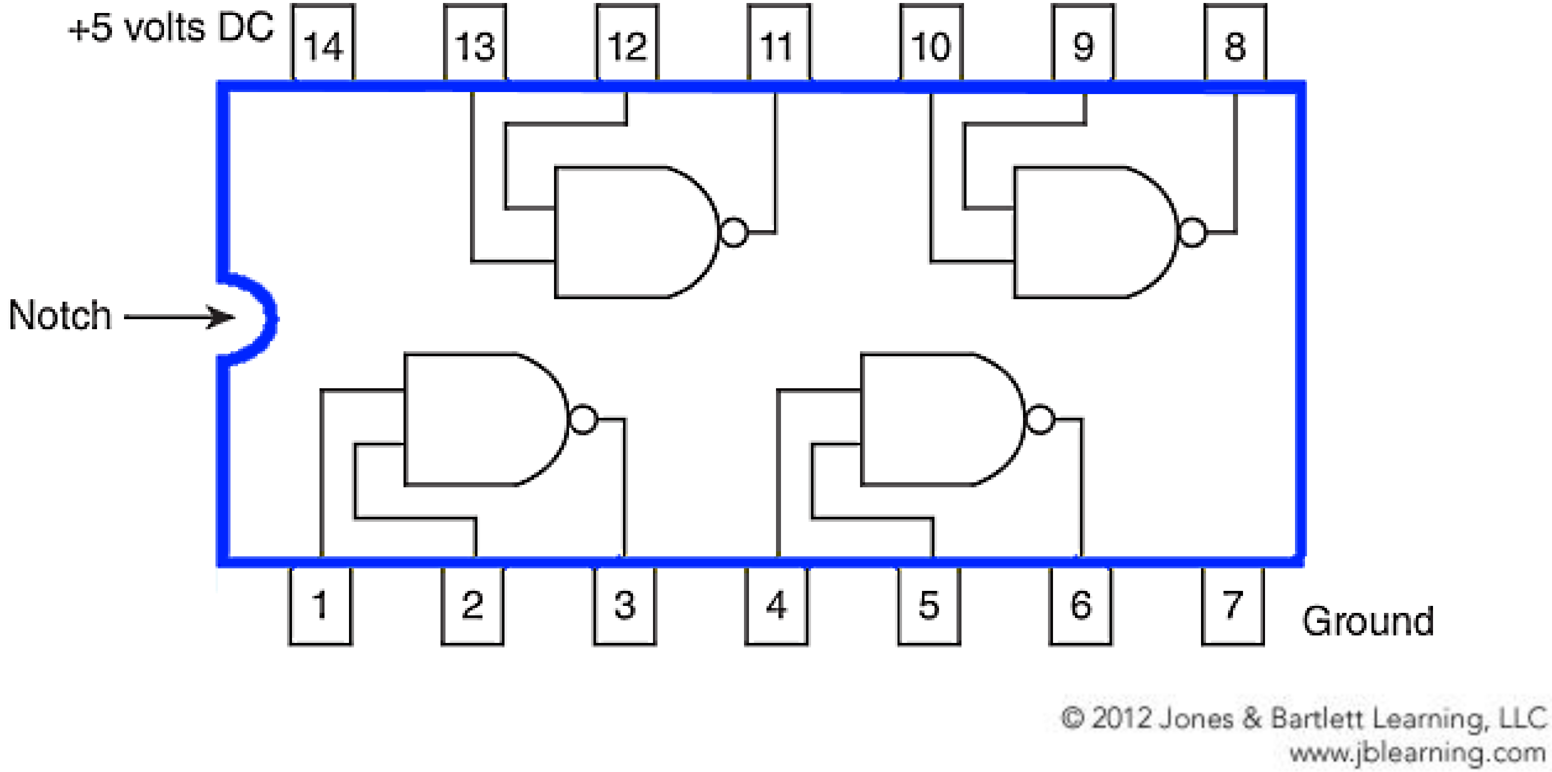
- The main thing to remember is that combinations of gates implement Boolean functions.
- The circuit below implements the Boolean function $F(x,y,z) = x + y'z$:



We simplify our Boolean expressions so that we can create simpler circuits.

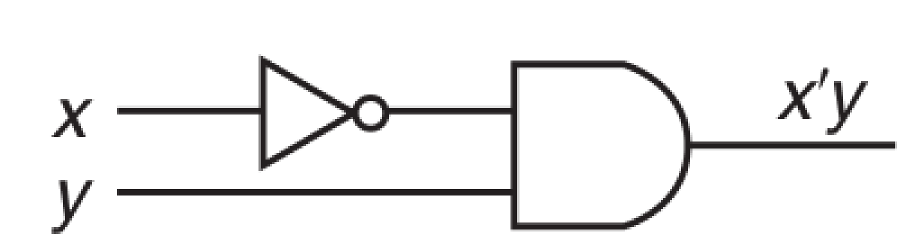
3.4 Digital Components

- Standard digital components are combined into single integrated circuit packages.
- Boolean logic can be used to implement the desired functions.

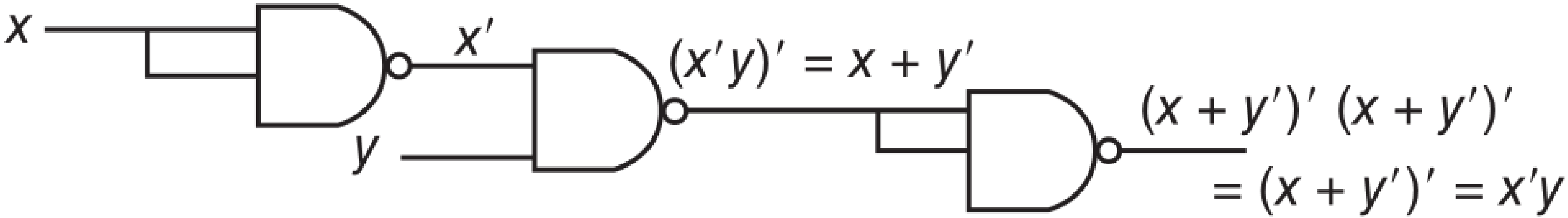


3.4 Digital Components

- The Boolean circuit:

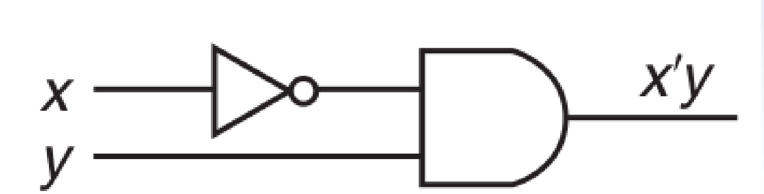


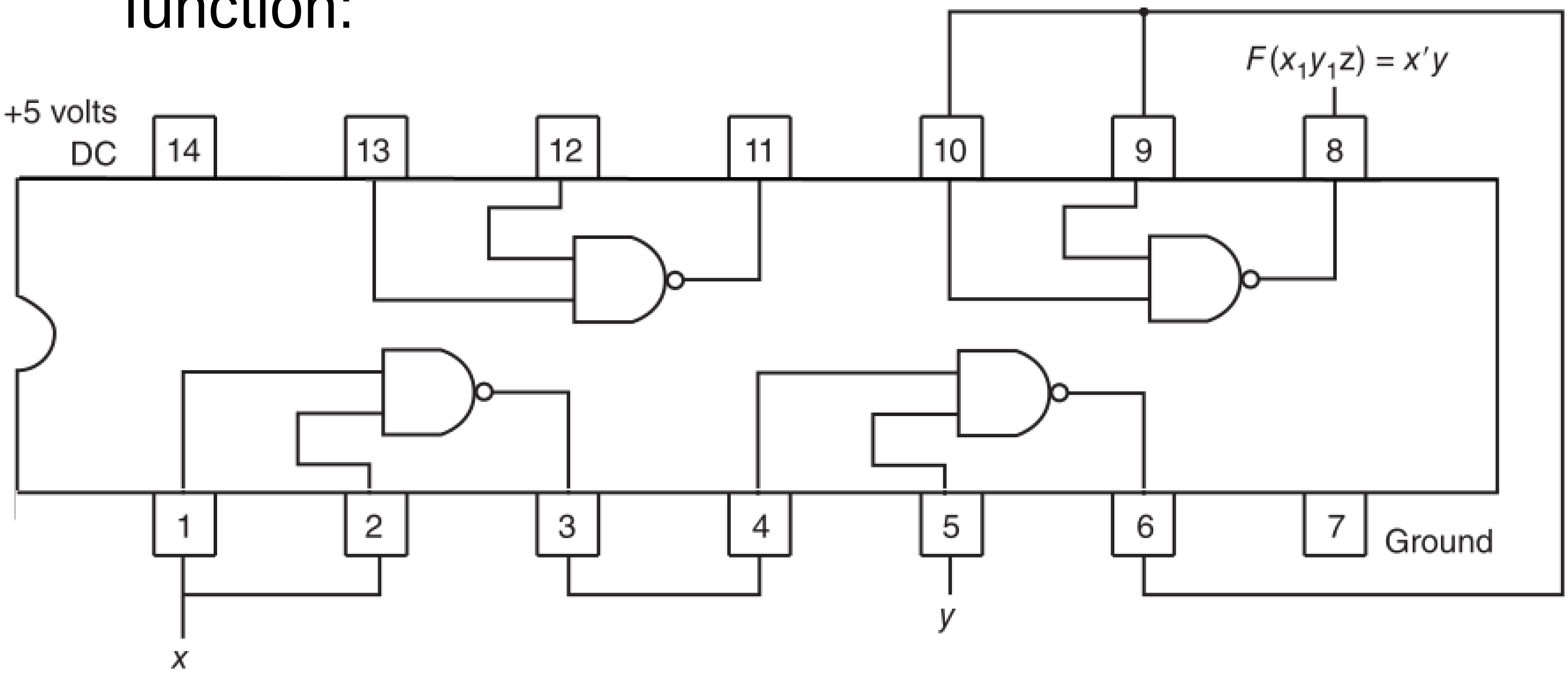
- Can be rendered using only NAND gates as:



3.4 Digital Components

- So we can wire the pre-packaged circuit to implement our function:





3.4 Digital Components

- Boolean logic is used to solve practical problems.
- Expressed in terms of Boolean logic practical problems can be expressed by truth tables.
- Truth tables can be readily rendered into Boolean logic circuits.

3.4 Digital Components

- Suppose we are to design a logic circuit to determine the best time to plant a garden.
- We consider three factors (inputs):
- (1) time, where 0 represents day and 1 represents evening;
- (2) moon phase, where 0 represents not full and 1 represents full; and
- (3) temperature, where 0 represents 45°F and below, and 1 represents over 45°F.
- We determine that the best time to plant a garden is during the evening with a full moon.

3.4 Digital Components

- This results in the following truth table:

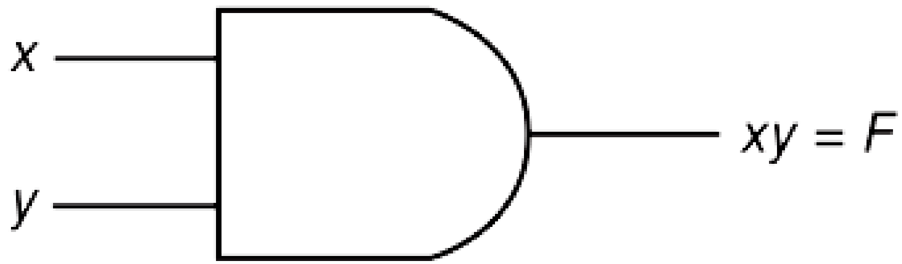
Time (x)	Moon (y)	Temperature (z)	Plant?
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

3.4 Digital Components

- From the truth table, we derive the circuit:

Time (x)	Moon (y)	Temperature (z)	Plant?
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

$$F(x,y,z) = xyz' + xyz = xy$$



Boolean Logic Conclusion

- Computers are implementations of Boolean logic.
- Boolean functions are completely described by truth tables.
- Logic gates are small circuits that implement Boolean operators.
- The basic gates are AND, OR, and NOT.
  - The XOR gate is very useful in parity checkers and adders.
- The “universal gates” are NOR, and NAND.