

# Chapter 2

## Logic Gates

### 2.1 The Three Basic Logic Gates

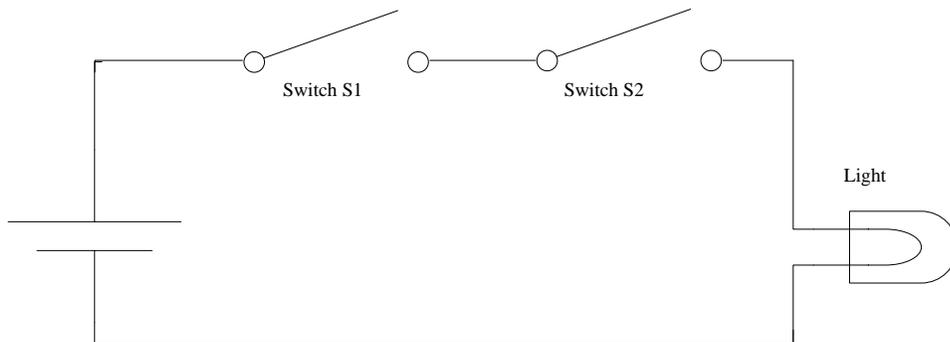
Much of a computer's hardware is comprised of digital logic circuits. Digital logic circuits are built from just a handful of primitive elements, called *logic gates*, combined in various ways.

In a digital logic circuit, only two values may be present. The values may be  $-5$  and  $+5$  volts. Or the values may be  $0.5$  and  $3.5$  volts. Or the values may be... you get the picture. To allow consideration of all of these possibilities, we will say that digital logic circuits allow the presence of two logical values: 0 and 1.

So, signals in a digital logic circuit take on the values of 0 or 1. Logic gates are devices which compute functions of these binary signals.

#### The AND Gate

Consider the circuit below which consists of a battery, a light, and two switches in series:



When will the light turn on? It should be clear that the light will turn on only if both switch S1 *and* switch S2 are shut.

It is quite likely that you encounter the *and* operation in some shape or form hundreds of times each day. Consider the simple action of withdrawing funds from your checking account at an ATM. You will only be able to complete the transaction if you have a checking account *and* you have money in it. The ATM will only permit the transaction if you have your ATM card *and* you enter your correct 4-digit PIN. To enter the correct PIN, you have to enter the first digit correctly *and* enter the second digit correctly *and* enter the third digit correctly *and* enter the fourth digit correctly.

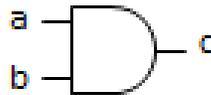
Returning to the circuit above, we can represent the light's operation using a table:

S1	S2	Light
open	open	off
open	closed	off
closed	open	off
closed	closed	on

The switch is a binary device: it can be open or closed. Let's represent these two states as 0 and 1. Likewise, the light is a binary device with two states: off and on, which we will represent as 0 and 1. Rewriting the table above with this notation, we have:

S1	S2	Light
0	0	0
0	1	0
1	0	0
1	1	1

This table, which displays the output for all possible combinations of the input, is termed the *truth table* for the AND operation. In a computer, this *and* functionality is implemented with a circuit called an AND gate. The simplest AND gate has two inputs and one output and is represented pictorially by the symbol:



where the inputs have been labeled  $a$  and  $b$ , and the output has been labeled  $c$ . If both inputs are 1 then the output is 1. Otherwise, the output is 0.

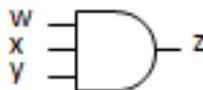
We represent the *and* operation by using either the multiplication symbol (i.e., “ $\cdot$ ”) or by writing the inputs together. Thus, for the AND gate shown above, we would write the output  $c$  as  $c = a \cdot b$  or as  $c = ab$ . This would be pronounced: “ $c = a$  and  $b$ .”

The truth table for the AND gate is shown below. The output  $c = ab$  is equal to 1 *if and only if* (iff)  $a$  is 1 *and*  $b$  is 1. Otherwise, the output is 0.

$a$	$b$	$c$
0	0	0
0	1	0
1	0	0
1	1	1

(Some textbooks use the symbol  $\wedge$  for the AND operation; i.e., some textbooks would use the notation  $c = a \wedge b$ . We will not use this notation in this text, but you may encounter it in the future.)

AND gates can have more than one input (however, an AND gate always has just a single output). Let's consider a three-input AND gate:



The expression for  $z$  is  $z = w \cdot x \cdot y$ . In words, the output  $z = 1$  if and only if  $w = 1$  and  $x = 1$  and  $y = 1$ . Otherwise  $z = 0$ .

## Example

Write the truth table for the 3-input AND gate.

Solution:

We begin by recognizing that we have three inputs and one output, so, labeling the inputs as  $w$ ,  $x$  and  $y$ , and the output as  $z$ , the heading of the truth table will be:

$w$	$x$	$y$	$z$

A truth table conveys the value of the output for all possible input combinations. One possible input combination is  $w = 1$ ,  $x = 0$  and  $y = 1$ . Another possible input combination is  $w = 0$ ,  $x = 1$  and  $y = 1$ . Each of these possibilities will be represented as a line in the truth table, with the output  $z$  indicated for each input combination.

So, we must consider: How many different input combinations will exist if we have three inputs? Note that since each input combination will be presented on one line of the truth table, the foregoing question can be restated as: *How many lines will be in a truth table with 3 inputs?*

The answer is provided by the **Multiplication Rule**, which states that if an operation consists of  $k$  steps and the first step can be performed in  $n_1$  ways and the second step can be performed in  $n_2$  and the third step can be performed in  $n_3$  ways, and so forth, then the entire operation can be performed in  $n_1 n_2 n_3 \cdots n_k$  different ways.

Applying the multiplication rule to our specific example, the “steps” correspond to the inputs, of which there are three (i.e.,  $k = 3$ ). The first input can take on two different values, 0 or 1, so  $n_1 = 2$ . Likewise, the second and third inputs can each take on two different values, so  $n_2 = 2$ ,  $n_3 = 2$ .

So, how can we present the truth table in an organized manner?

The answer is to start by considering the three inputs to be a

$w$	$x$	$y$	$z$
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Finally, we recall that for this three input AND gate, the output  $z = 1$  if and only if  $w = 1$  and  $x = 1$  and  $y = 1$ . Otherwise  $z = 0$ . Thus, the truth table is:

$w$	$x$	$y$	$z$
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	0
1	1	1	1

---

In general, a logic gate with  $k$  inputs will have  $2^k$  possible input combinations.

### Example

How many lines will be in the truth table for a 6-input AND gate?

Solution:

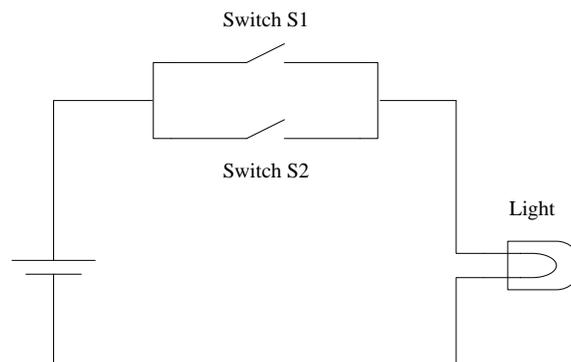
---

Note that the  $k$ -input AND gate can be described in three different ways:

- 
- 
- 

### The OR Gate

Now consider the circuit shown below, that has 2 switches in parallel.



It is evident that the light will turn on when either switch S1 is shut **or** switch S2 is shut **or** both are shut.

It is quite likely that you encounter the *or* operation in some shape or form hundreds of times each day. Consider the simple action of sitting on your couch at home at two in the morning studying for your Digital Logic class. Your phone will ring if you get a call from Alice *or* from Bob. Your home's security alarm will go off if the front door opens *or* the back door opens. You will drink a cup of coffee if you are drowsy *or* you are thirsty.

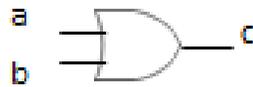
We can represent the light's operation using a table

S1	S2	Light
open	open	off
open	closed	on
closed	open	on
closed	closed	on

Changing the words *open* and *off* to 0 and the words *shut* and *on* to 1 and the table becomes:

S1	S2	Light
0	0	0
0	1	1
1	0	1
1	1	1

This is the truth table for the OR operation. This *or* functionality is implemented with a circuit called an OR gate. The simplest OR gate has two inputs and one output and is represented pictorially by the symbol:



If either or both inputs are 1, the output is 1. Otherwise, the output is 0.

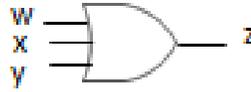
We represent the *or* operation by using the addition symbol. Thus, for the OR gate above, we would write the output  $c$  as  $c = a + b$ . This would be pronounced: " $c = a$  or  $b$ ."

The truth table for the OR gate is shown below. The output is 1 if  $a$  is 1 *or*  $b$  is 1; otherwise, the output is 0.

$a$	$b$	$c$
0	0	0
0	1	1
1	0	1
1	1	1

(Some textbooks use the symbol  $\vee$  for the OR operation; i.e., some textbooks would use the notation  $c = a \vee b$ . We will not use this notation, but you may encounter it in the future.)

OR gates can have more than one input (however, an OR gate always has just a single output). A three-input OR gate is shown below:



### Example

For the three-input OR gate shown above:

- (a) Write a mathematical expression for  $z$ .
- (b) Describe the operation of the gate in words.
- (c) Write the truth table for the gate.

Solution:

- (a)
- (b)
- (c)

---

### The NOT Gate

The last of our basic logic gates is the NOT gate. The NOT gate always has one input and one output. If the input is 1, the output is 0. If the input is 0, the output is 1. This operation—changing the value of the binary input—is called *complementation*, *negation* or *inversion*. The mathematical symbol for negation is an apostrophe. If the input to a NOT gate is  $P$ , the output, termed the *complement*, is denoted as  $P'$ .

The pictorial symbol for a NOT gate is intended to depict an amplifier followed by a bubble, shown below. Sometimes the NOT operation is represented by just the bubble, without the amplifier. We will see this later.



The truth table for the NOT gate is shown below:

$P$	$P'$
0	1
1	0

(Some textbooks use the over-bar symbol or a preceding tilde symbol for the NOT operation; i.e., some textbooks would use the notation  $\overline{P}$  or  $\sim P$ . We will not use this notation, but you may encounter it in your future studies.)

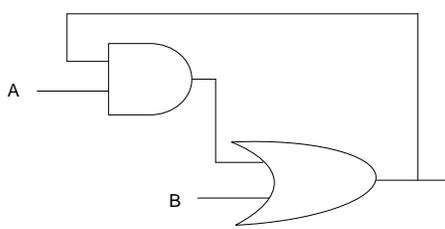
## 2.2 Combinational Logic Circuits

A combinational logic circuit is a digital logic circuit whose output is determined only by the current values of the inputs (with no dependence on past input values). A combinational circuit is built up by combining our three basic logic gates with the following provisos:

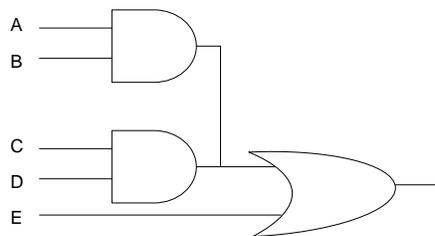
- The output of a gate may not *eventually* feed back to that same gate.
- The output lines from 2 different gates cannot be combined.

### Example

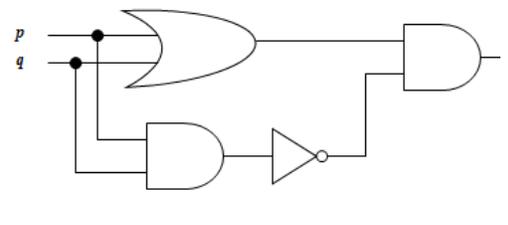
Which of the circuits depicted below are combinational logic circuits? In each case, explain your reasoning.



(a)



(b)



(c)

Solution

It is very convenient to describe combinational logic circuits in terms of *Boolean expressions* which are expressions that can only take on two values: 0 or 1. You have already seen several Boolean expressions. For example, if the inputs to an OR gate are  $X$  and  $Y$ , the output is  $X + Y$ . The terms  $X$ ,  $Y$  and  $X + Y$  are all Boolean expressions because they can only take on only the two values 0 or 1.

Similarly, if the inputs to an AND gate are  $X$  and  $Y$ , the output is  $XY$ . The term  $XY$  is a Boolean expression. If  $X$  enters a NOT gate, the output is  $X'$ .  $X'$  is a Boolean expression.

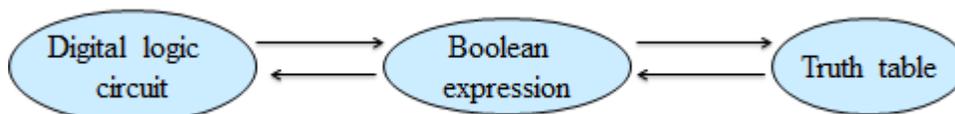
Note that since the OR operation uses the ordinary plus sign, the output of an OR gate—the term  $X + Y$ —is called a sum term. Similarly, since the AND operation uses the ordinary multiplication sign or no symbol at all, the output of an AND gate—the term  $XY$  or  $X \cdot Y$ —is called a product term.

We mentioned in the last section that logic gates can be represented in three different ways: as a pictorial circuit (hereafter termed a digital logic circuit), as a truth table or as a mathematical expression (specifically, a Boolean expression). Similarly, a combinational circuit, which is built up by combining our three basic logic gates, can also be represented in the same three ways: as a digital logic circuit, as a truth table, or as a Boolean expression. Our goal is to become comfortable shifting between these three representations.

Specifically, we would like to be able to perform the following:

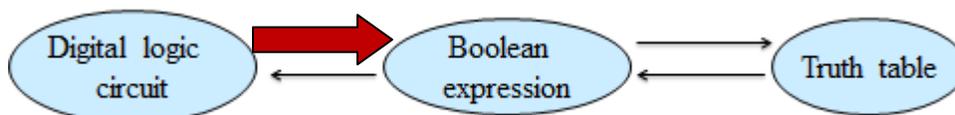
- Given a digital logic circuit, write the corresponding Boolean expression for the circuit.
- Given a Boolean expression for a digital logic circuit, construct the corresponding truth table
- Given a Boolean expression, draw a digital logic circuit that represents this expression.
- Given a truth table, construct a Boolean expression for the truth table.

Pictorially, we would like to be able to perform the transitions outlined below:



### From the Digital Logic Circuit to the Boolean Expression

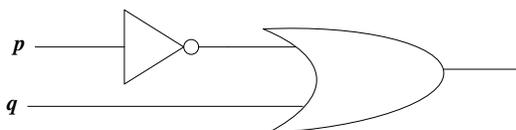
Given a digital logic circuit, we would like to determine the Boolean expression that governs the circuit:



The Boolean expression corresponding to a digital logic circuit can be determined by evaluating the effect of the logic gates on the input expressions.

### Example

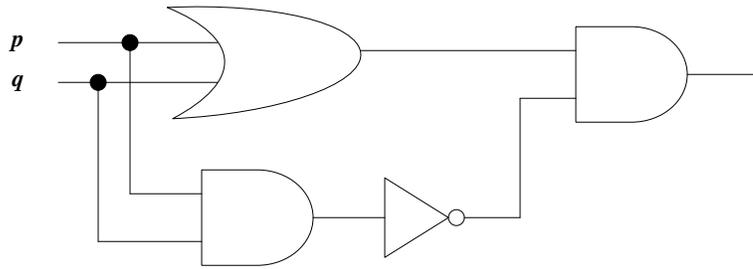
Write the Boolean expression for the output of the logic circuit shown below.



Solution:

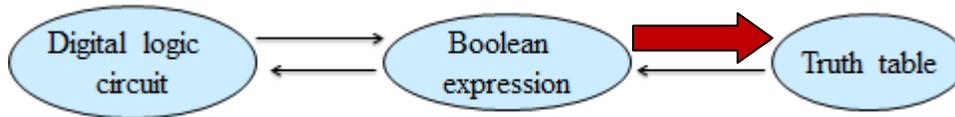
## Example

Write the Boolean expression for the output of the logic circuit shown below.



Solution:

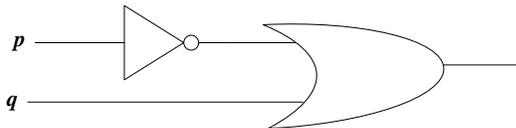
## From the Boolean Expression to the Truth Table



To develop the truth table for a complex Boolean expression, it is usually easiest to develop successive columns displaying the truth values for component (sub-parts) of the expression, gradually building toward the final column representing the truth values for the full expression.

## Example

Earlier we saw that the logic circuit shown below



has the Boolean expression \_\_\_\_\_. Construct a truth table for this logic circuit.

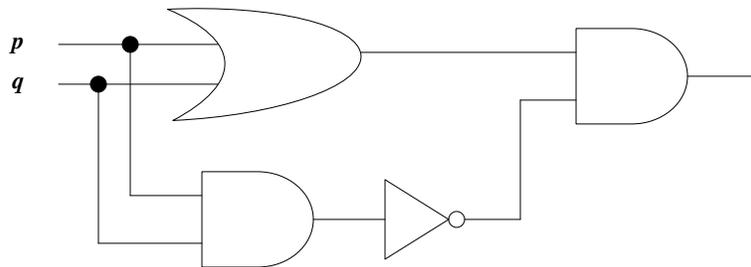
Solution:

---

Note that sometimes it is easy to construct the truth table directly from the logic circuit (by tracing through the circuit for each possible combination of input values) without first deriving the Boolean expression. Nevertheless, except for simple circuits, it is usually easier (and less prone to errors) to go via the Boolean expression as shown above.

### Example

Earlier we saw that the logic circuit shown below

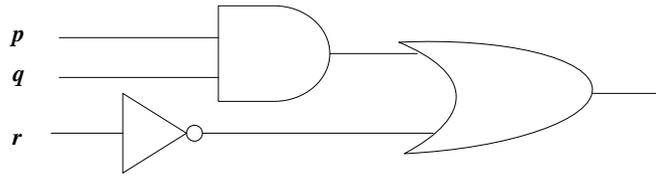


has the Boolean expression \_\_\_\_\_ . Construct a truth table for this logic circuit.

Solution:

## Example

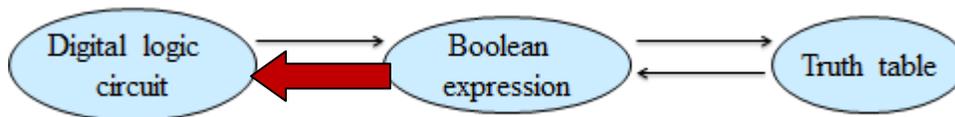
Write the Boolean expression for the logic circuit shown below and then construct the truth table for it.



Solution:

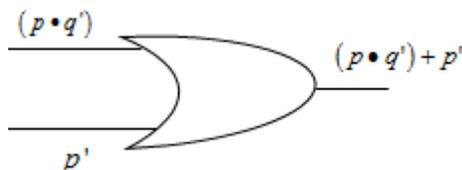
---

## From the Boolean Expression to the Digital Logic Circuit

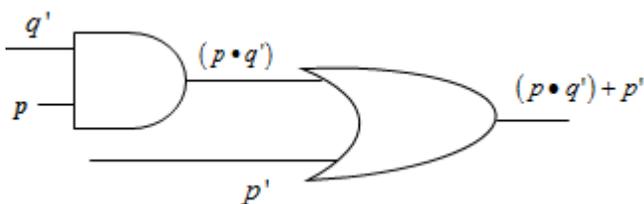


Given a Boolean expression, we would like to be able to draw a digital logic circuit that will implement the expression. The basic approach to drawing the circuit is to write the expression on the right side of the page and then draw the circuit from right to left, by working in from the outermost part of the expression.

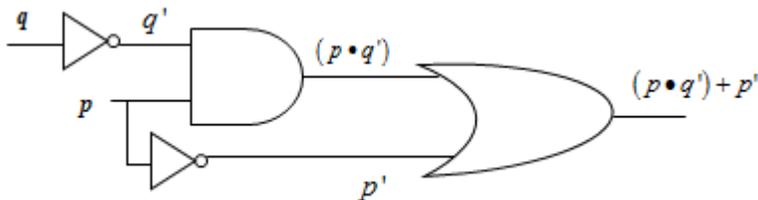
For example, suppose we wished to construct a digital logic circuit that will implement the Boolean expression  $(p \cdot q') + p'$ . At the highest level, we are OR-ing two terms:  $(p \cdot q')$  and  $p'$ . Constructing a circuit with these inputs we have:



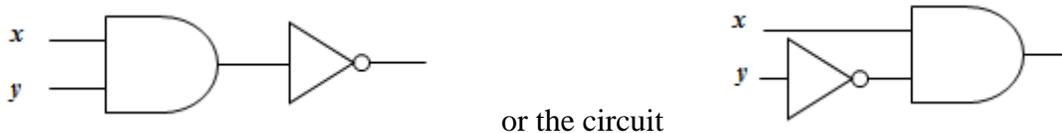
Now, let's continue to deconstruct this by looking at the top term,  $(p \cdot q')$ . This is the AND-ing of  $p$  and  $q'$ . Thus, continuing the drawing from right-to-left we have:



Finally, we consider the negations.



It is worth pausing to note the “order of precedence” for Boolean operations. If we have an expression with ANDs, ORs and NOTs, what is the correct order for performing the operations? To make this question more concrete, if we have the expression  $xy'$  does this correspond to the circuit



or the circuit

The order of precedence is:

- 
- 
- 
- 

Thus, the correct circuit for  $xy'$  is the circuit shown on \_\_\_\_\_.

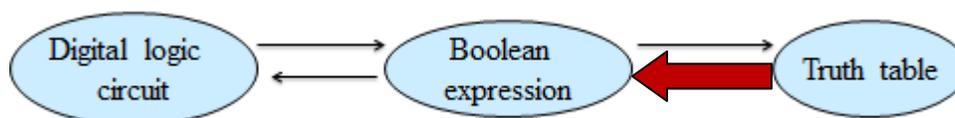
**Example**

Draw a digital logic circuit that represents the Boolean expression  $x + (y \cdot (x \cdot z)')$

Solution: We develop the circuit incrementally:

---

## From the Truth Table to the Boolean Expression



Lastly, we would like to develop a Boolean expression from a truth table. But, before doing this, we first introduce some terminology.

In a Boolean expression, any appearance of a variable (or its complement) is called a **literal**.

### Example

How many literals are in this expression:  $ab' + bc'd + a'd + e$

Solution:

---

One or more literals connected by an AND operator is called a **product term**.

### Example

How many product terms are in this expression:  $ab' + bc'd + a'd + e$

Solution:

A **minterm** is a product term that includes all of the input variables of a problem (either complemented or uncomplemented).

### Example

Suppose our circuit has 4 inputs:  $a, b, c$  and  $d$ . Which of the following are minterms?

$abcd$

$abc$

$a'bcd'$

$b'c'd'$

Solution:

Each line in a truth table corresponds to a minterm. For example, consider the truth table:

$p$	$q$	$r$	$(p \cdot q) + r'$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Each line of the truth table corresponds to a minterm consisting of three literals:

- The first literal is either  $p$  or  $p'$ . Specifically, the first literal is  $p$  if the value of  $p$  is 1 in this row, and is  $p'$  if the value of  $p$  is 0 in this row.
- The second literal is either  $q$  or  $q'$ . Specifically, the second literal is  $q$  if the value of  $q$  is 1 in this row, and is  $q'$  if the value of  $q$  is 0 in this row.
- The third literal is either  $r$  or  $r'$ . Specifically, the third literal is  $r$  if the value of  $r$  is 1 in this row, and is  $r'$  if the value of  $r$  is 0 in this row.

So, for example, the first line of the truth table, for which  $p = 0, q = 0$  and  $r = 0$  corresponds to the minterm  $p'q'r'$ .

### Example

In the truth table above, what is the minterm corresponding to the third row? The seventh row?

Solution:

With this terminology, we are ready to examine the procedure for developing a Boolean expression from a truth table. First, we will just mechanically run through the procedure. Then, once you appreciate the process involved, we will delve into why this procedure works.

To develop a Boolean expression from a truth table:

Step 1. Identify the rows of the truth table that have an output of 1.

Step 2. For each such row, write the minterm corresponding to that row.

Step 3. We will now have one minterm corresponding to each row in the truth table that had an output of 1. The Boolean expression for the truth table consists of the OR-ing of all of these minterms together.

### Example

Construct a Boolean expression for the truth table shown, and sketch the digital logic circuit to implement the truth table.

$p$	$q$	output
0	0	0
0	1	1
1	0	0
1	1	1

Solution: