Digital Logic Design

Tutorials and Laboratory Exercises

John F. Passafiume
Michael Douglas

Digital Logic Design

Tutorials and Laboratory Exercises

John F. Passafiume
Michael Douglas

Georgia Institute of Technology

1817

HARPER & ROW, PUBLISHERS, New York
Cambridge, Philadelphia, San Francisco,
London, Mexico City, São Paulo, Singapore, Sydney

Sponsoring Editor: John Willig
Project Editor: Nora Helfgott
Text Design Adaptation: Betty Sokol
Cover Design: Betty Sokol
Cover Photo: Stock, Boston
Text Art: Reproduction Drawings Limited
Production: Delia Tedoff
Compositor: ComCom Division of Haddon Craftsmen, Inc.
Printer and Binder: The Murray Printing Company

Library of Congress Cataloging in Publication Data

Passafiume, John F.
Digital logic design.

Bibliography: p.
1. Logic circuits. 2. Logic circuits—Laboratory manuals. I. Douglas, Michael, 1960– . II. Title.
TK7868.L6P38 1985 621.3819'5835 84-10790
ISBN 0-06-045028-2

84 85 86 87 9 8 7 6 5 4 3 2 1

Contents

HCV27/12=09

chapter 1

Review of Fundamental Concepts and An Introduction to the TTL Family

Objectives

1. A review of the basic logic gates and their functions.
2. An introduction to the TTL family—functional, electrical, and physical characteristics.
3. An introduction to circuit wiring and testing techniques.

1.1 Fundamental Digital Concepts

The student is expected to have had experience with boolean algebra and the basic logic gates. As a refresher we will review the five basic logic gates: AND, OR, INVERT, NAND, and NOR.

1.1.1 The AND Gate

The AND gate provides a true output only when all inputs are true. The nature of an AND gate is illustrated in Table 1.1. In this and the following truth tables, true is represented by a 1 and false is represented by a 0. This truth table shows the output F of a two-input AND gate for each possible combination of the inputs A and B. The American National Standards Institute (ANSI) symbol for the AND gate is shown in Figure 1.1.

In boolean algebra, the AND function is usually indicated by a dot between the

Table 1.1 Truth Table for the AND Function

A	*B*	*F*
0	0	0
0	1	0
1	0	0
1	1	1

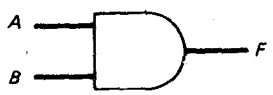


Figure 1.1 AND Gate

Table 1.2 Truth Table for the OR Function

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Figure 1.2 OR Gate

inputs (logical multiplication), or by implicit multiplication, as in algebra, by placing the input variables next to each other. For example:

$$F = A \text{ AND } B$$

can be written as:

$$F = A \cdot B$$
$$F = AB$$
$$F = (A)(B)$$

1.1.2 The OR Gate

The OR gate provides a true output when any of its inputs are true. The nature of an OR gate is illustrated in Table 1.2. This truth table shows the output F of a two-input OR gate for each possible combination of the inputs A and B. The ANSI symbol for the OR gate is shown in Figure 1.2.

In boolean algebra, the OR function is usually indicated by a plus sign (logical addition) between the inputs. For example:

$$F = A \text{ OR } B$$

can be written as:

$$F = A + B$$

1.1.3 The NOT Gate or Inverter

The output of an inverter is simply the complement of the input. The nature of an inverter is illustrated in Table 1.3, which shows the output of an inverter for the two possible inputs. The ANSI symbol for the inverter is shown in Figure 1.3.

Table 1.3 Truth Table for the NOT Function

A	F
0	1
1	0

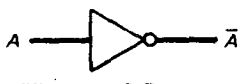


Figure 1.3 Inverter

Table 1.4 Truth Table for the NAND Function

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

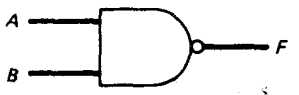


Figure 1.4 NAND Gate

In boolean algebra, the NOT function is usually indicated by a bar over the variable or a following prime. For example:

$$F = \text{NOT } A$$

can be written as:

$$F = \overline{A}$$
$$F = A'$$

The use of the prime should be limited to typewriters where the bar cannot be generated. The bar is much more readable, especially in more complex expressions.

1.1.4 The NAND Gate

The output of the NAND gate is simply the complement of the output of the AND gate. Thus the NAND gate provides a false output only when all inputs are true. The nature of the NAND gate is illustrated in Table 1.4. This truth table shows the output F of a two-input NAND gate for all possible combinations of the inputs A and B. The ANSI symbol for the NAND gate is shown in Figure 1.4.

In boolean algebra, the NAND function is not a primitive operator but a combination of the AND and invert functions. A NAND operation is simply the AND function with its output inverted: (N)ot (AND). This can be illustrated in boolean algebra as:

$$F = \overline{AB}$$
$$F = \overline{A \cdot B}$$

1.1.5 The NOR Gate

The output of the NOR is the complement of the output of the OR gate. Therefore, the output of the NOR gate is illustrated in Table 1.5. This truth table shows the output F of a two-input NOR gate for all possible combinations of the inputs A and B. The ANSI symbol for the NOR gate is shown in Figure 1.5.

In boolean algebra, the NOR function is not a primitive operator but a combina-

Table 1.5 Truth Table for the NOR Function

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

Figure 1.5 NOR Gate

tion of the OR and invert functions. A NOR operation is simply the OR function with its output inverted: (N)ot (OR). This can be illustrated in boolean algebra as:

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

1.2 Introduction to the TTL Family

So far we have talked of the five basic gates in terms of true and false, logic symbols, and truth tables. But how are these logic gates used in actual circuits? Logic gates can be built in a variety of ways. Some of the first computing devices used mechanical relays to implement logic gates. More recent forms include metal-oxide semiconductor (CMOS, PMOS, NMOS), emitter-coupled logic (ECL), resistor-transistor logic (RTL), diode-transistor logic (DTL), and transistor-transistor logic (TTL). TTL is the most popular logic family today and will be used in this course.

1.2.1 TTL Characteristics

The members of the TTL family are each identified by a unique part number. TTL part numbers are in the form 74XXX, where the XXX is a two- or three-digit number uniquely identifying the particular type of chip. For example, the part number for the TTL chip containing four two-input AND gates is 7408. The chip containing four two-input OR gates is the 7432.

In addition to the 74 series, there is a 54 series of chips that is functionally equivalent to the corresponding 74 series of chips but meet more stringent military specifications.

In the TTL family, a high logic level is represented by a voltage in the range of 2.0v–5.0v, and a low logic level by a voltage in the range of 0.0v–0.8v. Any voltage greater than 0.8v and less than 2.0v is undefined. Typically, a high logic level represents a 1 or TRUE and a low logic level represents a 0 or FALSE. These levels are referred to as *active high* logic levels. With *active low* logic levels, a low logic level represents a 1 or TRUE and a high logic level represents a 0 or FALSE.

1.2.2 TTL Subfamilies

Within the TTL family, several subgroupings are present. For example, chips are available with one of three different output circuits: totem-pole, open-collector, and three-state.

Totem-pole outputs derive their name from the appearance of the output circuit. A pull-up transistor and pull-down transistor are "stacked" on top of each other to decrease switching time and reduce the number of external components required to connect gates. Never connect the outputs of totem-pole gates together; the connection may damage the gates. We will deal primarily with totem-pole outputs in this course.

Open-collector outputs have no pull-up transistor. Instead, an external pull-up resistor is required for the gate to assume a high logic level. This makes open-collector gates ideal for interface applications since the pull-up voltage can usually be up to 15 or 30 volts. Also, the outputs of open-collector gates can be connected together to form hard-wired logic.

Three-state outputs can assume one of three states: a high logic level, a low logic

level, and a third state that appears as an open circuit—as if nothing is there. This third state is often used in computers where many devices must access a single data bus.

There are several different internal constructions (affecting the speed and power consumption of the chip) that serve to further classify the TTL family. In addition to the standard 74 series, there is a low-power, low-speed series, 74L; a high-power, high-speed series, 74H; another high-power yet even faster series using Schottky diodes, 74S; and, finally, a very low-power, yet faster-than-standard TTL series using Schottky diodes, the 74LS series.

1.3 Wiring and Testing a Circuit

In later labs, more and more complex circuits will be designed and wired. Following the basic guidelines provided here will make the wiring and debugging effort much simpler.

The first requirement in implementing a circuit after it has been designed is obtaining the necessary parts. The proper TTL chips to perform the logic functions required can be found by looking in Appendix B or in a TTL data book. For example, if a two-input NAND gate is needed, we find the proper TTL part number is 7400.

Once the part numbers are obtained, it is wise to write the pin numbers for each input and output used on the chip on your logic diagram. This will make wiring much simpler and reduce the chance for error. In addition to gate connections, each TTL chip will require two additional connections: a +5v power-supply connection and a ground connection. Typically, pin 14 is connected to +5v and pin 7 is connected to ground. Be careful though; there are exceptions.

It is very important to properly identify pin numbers before wiring a circuit. A chip can be ruined by simply inserting it in a circuit backwards. Figure 1.6 shows the pin numbering of a typical TTL chip. With the chip oriented as shown, pin 1 will always be the top left pin.

1.3.1 Wiring the Circuit

Circuits will be wired on a logic design station. These design stations generally provide a breadboard for wiring the circuit, a power supply, switches for inputs, LEDs for output monitoring, and, finally, some sort of debounced switch inputs. The

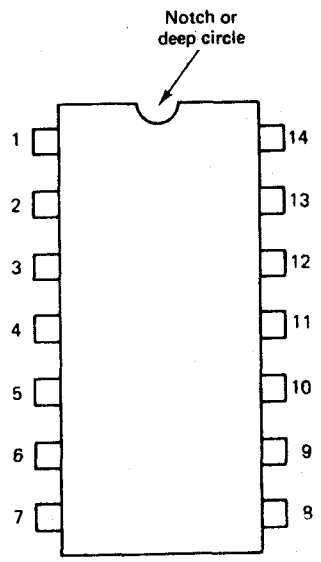


Figure 1.6 Pin Numbering

use and features of the logic designer you will use will be covered during the first lab session.

Wiring a circuit, especially a complex one, can be very frustrating. To make the process easier and reduce the chance for error, here are a few simple guidelines that should be followed.

1. Always start with the power OFF on the logic designer.
2. Insert the required chips in the breadboard. (Make sure the chips are oriented correctly.)
3. Connect all power (Vcc) and ground pins to +5v and ground respectively.
4. Connect wires between pins, switches, LEDs, and so on, as labeled on your logic diagram.
5. Visually inspect your circuit and correct any mistakes.
6. Turn on the power and test the circuit. The use of test equipment such as the logic probe, oscilloscope, and volt meter will be covered in the lab.

Lab Exercise

Objective

This lab is intended to introduce the student to the use of the lab equipment: the logic designer, the test equipment, and, of course, the TTL chips.

Procedure

This lab begins with a brief lecture introducing the procedures of the lab and the use of the lab equipment.

1. AND gate operation.
   (a) Look up the TTL part number for a two-input AND gate, and obtain this chip from your lab instructor.
   (b) Connect the inputs of one of the AND gates to the switches on the logic designer and the output of that gate to an LED on the logic designer.
   (c) Verify proper operation of the AND gate for all input combinations by using the input switches. When the LED is lit, this indicates a high logic level or 1. When the LED is off, this indicates a low logic level or 0.
   (d) Record the TTL part number, and produce a truth table based on the results of (c).
2. OR gate operation. (Repeat (1) for a two-input OR gate.)
3. NAND gate operation. (Repeat (1) for a two-input NAND gate.)
4. NOR gate operation. (Repeat (1) for a two-input NOR gate.)
5. What level does an unconnected input appear to be (high or low)? (*Hint:* Repeat one of the previous operations with one input left unconnected.) What reasoning did you use to determine your answer?

Review Questions

1.1 Generate the truth table for:
   (a) A four-input NAND gate
   (b) A four-input NOR gate

1.2 Can a NAND gate be used as an INVERTER? If so, illustrate how with a two-input NAND gate.

1.3 Can a NOR gate be used as an INVERTER? If so, illustrate how with a two-input NOR gate.

1.4 Assuming POSITIVE logic, what do each of the following voltage levels represent (1 or 0)?
   (a) 0.3v
   (b) 2.4v
   (c) 5.0v

(d) 0.0v
(e) 1.9v

1.5 Find the TTL part number and the number of gates per chip for each of the following logic gates:
(a) Inverters
(b) Two-input OR gates
(c) Four-input NAND gates
(d) Two-input NOR gates
(e) Two-input AND gates

1.6 What pin numbers must be connected to power supply and ground for each of the following TTL chips:
(a) 7400
(b) 7420
(c) 7476
(d) 74151

1.7 Which of the following TTL subfamilies is fastest in switching speed?
(a) 74XX
(b) 74SXX
(c) 74LSXX

1.8 Which of the following TTL subfamilies is lowest in power consumption?
(a) 74XX
(b) 74LSXX
(c) 74SXX

1.9 Show that the distributive law of OR over AND holds true by means of a truth table; that is, show that:

$$X + (Y \cdot Z) = (X + Y) \cdot (X + Z)$$

Note that this is not valid for ordinary algebra.

1.10 Show that the distributive law of AND over OR holds true by means of a truth table; that is, show that:

$$X \cdot (Y + Z) = (X \cdot Y) + (X \cdot Z)$$

Note that this rule holds for ordinary algebra.

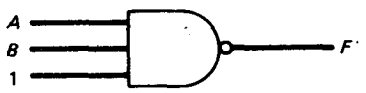
1.11 Use a truth table to prove the absorption rule; that is,

$$X + X \cdot Y = X$$

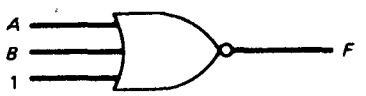
1.12 Use a truth table to prove the following theorem.

$$X \cdot (X + Y) = X$$

1.13 Given the following TTL gates and connections, give an expression for the output function F.
(a)



(b)



1.14 Which of the following families has the fastest switching speed?
(a) ECL
(b) CMOS
(c) 74SXX
(d) DTL

1.15 Which of the following families has the lowest standby current drain?
(a) ECL
(b) CMOS
(c) TTL
(d) DTL

1.16 Which of the following TTL subfamilies is typically lowest in cost?
(a) 74XX
(b) 74SXX
(c) 74LSXX

1.17 What would be the result of connecting two inverters in tandem? Why would this be a useful thing to do in a digital circuit?

1.18 What is the disadvantage of leaving the inputs of a logic gate unconnected?

chapter

2 Basic Two-Level Circuits

Objectives

1. An introduction to standard two-level forms: sum of products (SOP) and product of sums (POS).
2. An introduction to methods for implementing simple SOP and POS equations.

2.1 Sum of Products Form

Assume the existence of a simple digital network and that the operation of this network is defined by Table 2.1. This network has three inputs: A, B, and C, and an output F. The operation of this network can readily be determined by observing the truth table. The output F can be seen to equal 1 in four cases:

1. When $A = 0$ and $B = 1$ and $C = 0$ or
2. When $A = 0$ and $B = 1$ and $C = 1$ or
3. When $A = 1$ and $B = 0$ and $C = 0$ or
4. When $A = 1$ and $B = 1$ and $C = 1$

Assuming the true state of a variable to be 1 and the complemented state of a variable to be 0, the preceding observation can be converted almost directly into the following boolean expression:

$$F = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + ABC \tag{2.1}$$

This form of an equation, a series of product (AND) terms connected by addition (OR), is referred to as a *sum of products* (SOP) equation. This form is the most easily derived from a truth table.

Table 2.1

A	B	C	F
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	0
1	1	1	1

2.1.1 SOP Circuit Implementation

Once the proper SOP expression has been derived, the next step is the design of a logic diagram from which a circuit can be built to implement the digital network.

An expression in SOP form is easily converted into a logic diagram. The result of an SOP equation is the OR of the results of several AND functions. Thus the output of the logic network should also be the result of an OR operation on the results of several AND functions. The logic network to implement Equation 2.1 is illustrated in Figure 2.1.

As shown here, the output F is the result of an OR operation on the results of several AND functions—a direct parallel to the original SOP equation. Also note the two-level nature of the logic diagram. Ignoring the input inverters, we see that the longest path is always just two gates: the AND then the OR. Circuits in this form are referred to as two-level circuits.

In this diagram, the inverters providing the complements of the input variables are shown explicitly. Often these input inverters are not shown and the inputs to each gate simply labeled with the proper value. This simplifies the logic diagram and makes the inputs to the gates more readily apparent. Figure 2.2 (page 13) shows this format. When it is assumed that both a variable and its complement are available, as in Figure 2.2, we refer to these as *double-rail* inputs. Assume double-rail inputs are available on all logic diagrams you produce unless specifically stated otherwise.

2.1.2 Normal Form

An SOP equation in which each product (AND) term contains all input variables in either true or complemented form is said to be in *normal* or *canonical* form. Thus Equation 2.1 is in normal form since each product term contains all three input variables (A, B, and C) in either true or complemented form.

An equation in normal form will have one product term for each entry in the truth table for which the output is 1. For example, Equation 2.1 contains four product terms,

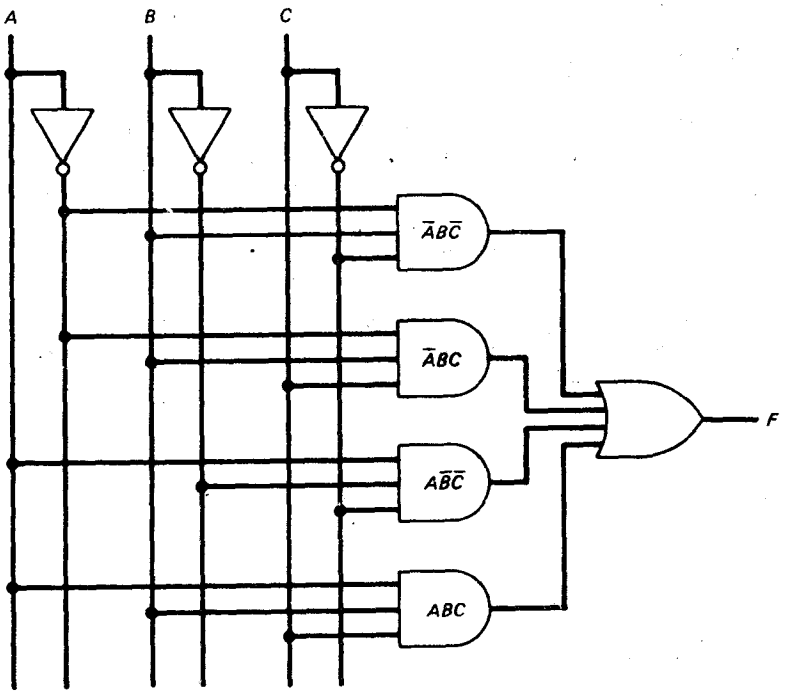


Figure 2.1 SOP Implementation of Table 2.1

and in the truth table there are four entries for which the output equals 1. If there were just three entries for which the output equaled 1, then the normal form equation for that truth table would contain just three product terms.

An equation in normal form is not necessarily the simplest or only equation to evaluate a particular boolean function. For example, the following boolean equation also evaluates the function defined by Table 2.1:

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

This equation is not in normal form since not all product terms contain all input variables. The first term is missing input C, and the last term is missing input A.

Though there are typically several equations that will evaluate a given function, there is just one equation in normal form that will. Thus, to remove ambiguity, a function is often expressed in normal form. To convert an SOP equation to normal form, terms with missing input variables must be "multiplied" by the missing variable. Since multiplication (ANDing) by 1 will not change a product term, we could rewrite the previous equation as:

$$F = \overline{A}B(1) + A\overline{B}\,\overline{C} + (1)BC$$

From the laws of boolean algebra we know that ORing any variable with its complement always equals 1:

$$(X + \overline{X}) = 1$$

Thus we can rewrite the previous equation as:

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

This provides the missing input variables for the product terms. Simplifying the previous equation yields:

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

Since the first and last terms are the same, one can be removed leaving the normal form equation

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

which is the same as the original Equation 2.1.

2.1.3 Minterms

Looking at Table 2.1 we see that each combination of input variables forms a binary value between 0 and $2^N - 1$ where N equals the number of input variables. In the previous example, N is 3; thus possible values range from 0 through 7 (000–111). Since each possible combination of input variables forms a unique binary value, a particular input combination can be represented by the decimal equivalent of the binary value formed by the inputs. This value is referred to as the *minterm* number.

For example, what input combination does minterm 5 ($m5$) represent? Assuming the three input variables used previously:

5 decimal = 101 binary => $A\overline{B}C$