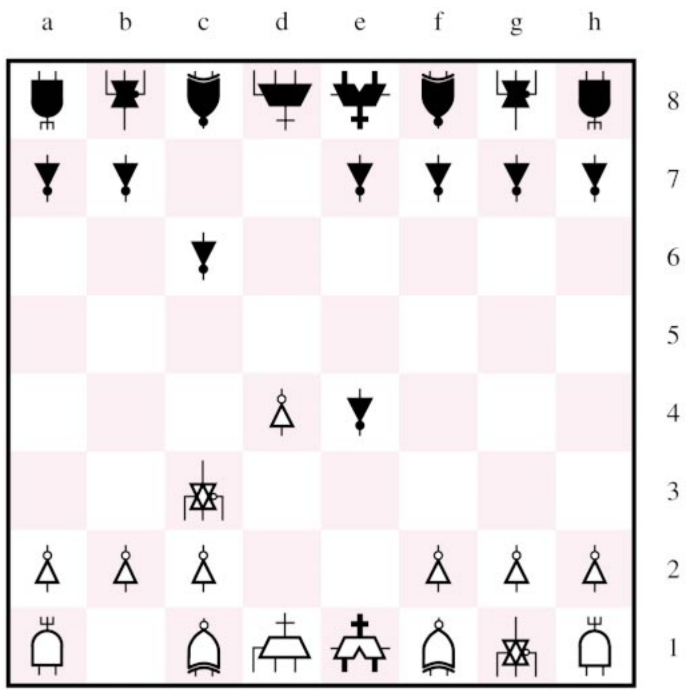
chapter

3

Implementation Technology



3. Nb1–c3, d5xe4

In section 1.2 we said that logic circuits are implemented using transistors and that a number of different technologies exist. We now explore technology issues in more detail.

Let us first consider how logic variables can be physically represented as signals in electronic circuits. Our discussion will be restricted to binary variables, which can take on only the values 0 and 1. In a circuit these values can be represented either as levels of voltage or current. Both alternatives are used in different technologies. We will focus on the simplest and most popular representation, using voltage levels.

The most obvious way of representing two logic values as voltage levels is to define a *threshold* voltage; any voltage below the threshold represents one logic value, and voltages above the threshold correspond to the other logic value. It is an arbitrary choice as to which logic value is associated with the low and high voltage levels. Usually, logic 0 is represented by the low voltage levels and logic 1 by the high voltages. This is known as a *positive logic* system. The opposite choice, in which the low voltage levels are used to represent logic 1 and the higher voltages are used for logic 0 is known as a *negative logic* system. In this book we use only the positive logic system, but negative logic is discussed briefly in section 3.4.

Using the positive logic system, the logic values 0 and 1 are referred to simply as "low" and "high." To implement the threshold-voltage concept, a range of low and high voltage levels is defined, as shown in Figure 3.1. The figure gives the minimum voltage, called V_{SS}, and the maximum voltage, called V_{DD}, that can exist in the circuit. We will assume that V_{SS} is 0 volts, corresponding to electrical ground, denoted *Gnd*. The voltage V_{DD} represents the power supply voltage. The most common level for V_{DD} is 5 volts, but 3.3 volts is also popular. In this chapter we will usually assume that $V_{DD} = 5$ V. Figure 3.1 indicates that voltages in the range *Gnd* to $V_{0,max}$ represent logic value 0. The name $V_{0,max}$ means the maximum voltage level that a logic circuit must recognize as low. Similarly, the range from $V_{1,min}$ to V_{DD} corresponds to logic value 1, and $V_{1,min}$ is the minimum voltage level that a logic circuit must interpret as high. The exact levels of $V_{0,max}$ and $V_{1,min}$

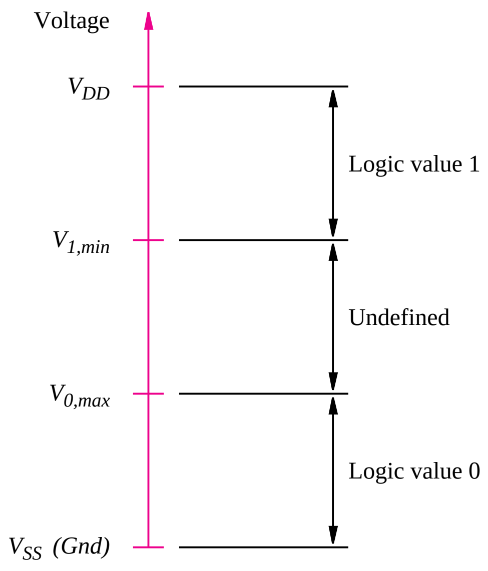


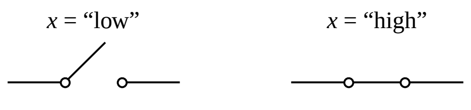
Figure 3.1 Representation of logic values by voltage levels.

depend on the particular technology used; a typical example might set $V_{0,max}$ to 40 percent of V_{DD} and $V_{1,min}$ to 60 percent of V_{DD}. The range of voltages between $V_{0,max}$ and $V_{1,min}$ is undefined. Logic signals do not normally assume voltages in this range except in transition from one logic value to the other. We will discuss the voltage levels used in logic circuits in more depth in section 3.8.3.

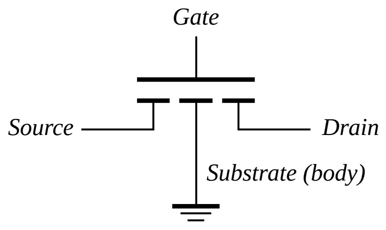
3.1 Transistor Switches

Logic circuits are built with transistors. A full treatment of transistor behavior is beyond the scope of this text; it can be found in electronics textbooks, such as [1] and [2]. For the purpose of understanding how logic circuits are built, we can assume that a transistor operates as a simple switch. Figure 3.2*a* shows a switch controlled by a logic signal, *x*. When *x* is low, the switch is open, and when *x* is high, the switch is closed. The most popular type of transistor for implementing a simple switch is the *metal oxide semiconductor field-effect transistor (MOSFET)*. There are two different types of MOSFETs, known as *n-channel*, abbreviated *NMOS*, and *p-channel*, denoted *PMOS*.

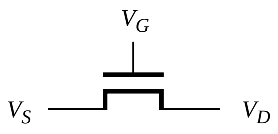
Figure 3.2*b* gives a graphical symbol for an NMOS transistor. It has four electrical terminals, called the *source*, *drain*, *gate*, and *substrate*. In logic circuits the substrate (also



(a) A simple switch controlled by the input *x*



(b) NMOS transistor



(c) Simplified symbol for an NMOS transistor

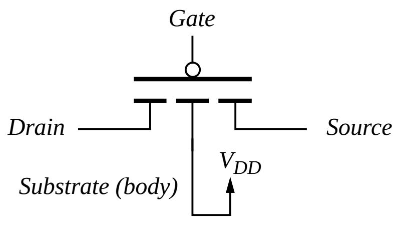
Figure 3.2 NMOS transistor as a switch.

called *body*) terminal is connected to *Gnd*. We will use the simplified graphical symbol in Figure 3.2*c*, which omits the substrate node. There is no physical difference between the source and drain terminals. They are distinguished in practice by the voltage levels applied to the transistor; by convention, the terminal with the lower voltage level is deemed to be the source.

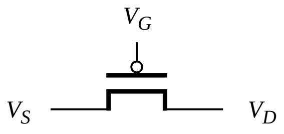
A detailed explanation of how the transistor operates will be presented in section 3.8.1. For now it is sufficient to know that it is controlled by the voltage V_G at the gate terminal. If V_G is low, then there is no connection between the source and drain, and we say that the transistor is *turned off*. If V_G is high, then the transistor is *turned on* and acts as a closed switch that connects the source and drain terminals. In section 3.8.2 we show how to calculate the resistance between the source and drain terminals when the transistor is turned on, but for now assume that the resistance is 0 Ω.

PMOS transistors have the opposite behavior of NMOS transistors. The former are used to realize the type of switch illustrated in Figure 3.3*a*, where the switch is open when the control input *x* is high and closed when *x* is low. A symbol is shown in Figure 3.3*b*. In logic circuits the substrate of the PMOS transistor is always connected to V_{DD}, leading to the simplified symbol in Figure 3.3*c*. If V_G is high, then the PMOS transistor is turned off and acts like an open switch. When V_G is low, the transistor is turned on and acts as a closed switch that connects the source and drain. In the PMOS transistor the source is the node with the higher voltage.

(a) A switch with the opposite behavior of Figure 3.2(*a*)



(b) PMOS transistor



(c) Simplified symbol for an PMOS transistor

Figure 3.3 PMOS transistor as a switch.

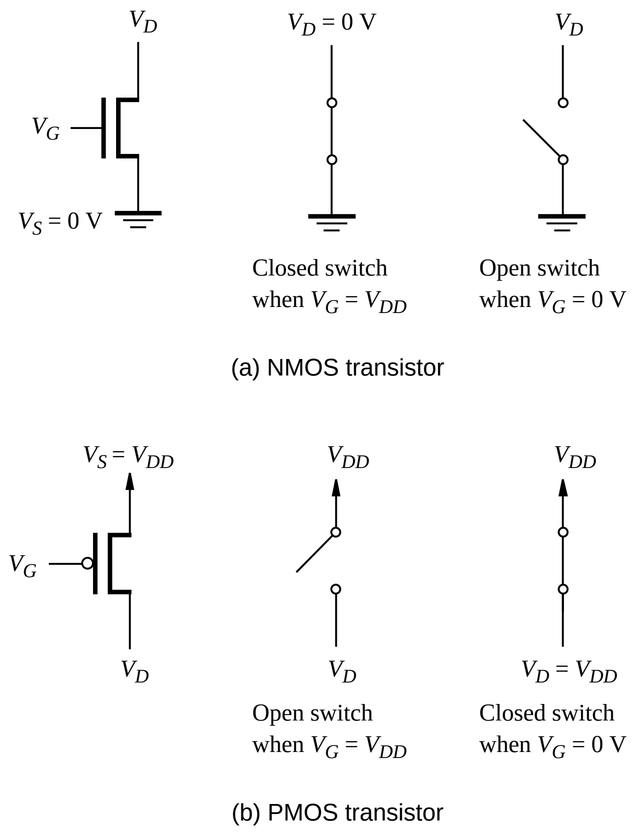


Figure 3.4 NMOS and PMOS transistors in logic circuits.

Figure 3.4 summarizes the typical use of NMOS and PMOS transistors in logic circuits. An NMOS transistor is turned on when its gate terminal is high, while a PMOS transistor is turned on when its gate is low. When the NMOS transistor is turned on, its drain is *pulled down* to *Gnd*, and when the PMOS transistor is turned on, its drain is *pulled up* to V_{DD}. Because of the way the transistors operate, an NMOS transistor cannot be used to pull its drain terminal completely up to V_{DD}. Similarly, a PMOS transistor cannot be used to pull its drain terminal completely down to *Gnd*. We discuss the operation of MOSFETs in considerable detail in section 3.8.

3.2 NMOS Logic Gates

The first schemes for building logic gates with MOSFETs became popular in the 1970s and relied on either PMOS or NMOS transistors, but not both. Since the early 1980s, a combination of both NMOS and PMOS transistors has been used. We will first describe how logic circuits can be built using NMOS transistors because these circuits are easier to understand.

Such circuits are known as NMOS circuits. Then we will show how NMOS and PMOS transistors are combined in the presently popular technology known as *complementary MOS*, or *CMOS*.

In the circuit in Figure 3.5*a*, when $V_x = 0$ V, the NMOS transistor is turned off. No current flows through the resistor *R*, and $V_f = 5$ V. On the other hand, when $V_x = 5$ V, the transistor is turned on and pulls V_f to a low voltage level. The exact voltage level of V_f in this case depends on the amount of current that flows through the resistor and transistor. Typically, V_f is about 0.2 V (see section 3.8.3). If V_f is viewed as a function of V_x, then the circuit is an NMOS implementation of a NOT gate. In logic terms this circuit implements the function $f = \overline{x}$. Figure 3.5*b* gives a simplified circuit diagram in which the connection to the positive terminal on the power supply is indicated by an arrow labeled V_{DD} and the connection to the negative power-supply terminal is indicated by the *Gnd* symbol. We will use this simplified style of circuit diagram throughout this chapter.

The purpose of the resistor in the NOT gate circuit is to limit the amount of current that flows when $V_x = 5$ V. Rather than using a resistor for this purpose, a transistor is normally used. We will discuss this issue in more detail in section 3.8.3. In subsequent diagrams a dashed box is drawn around the resistor *R* as a reminder that it is implemented using a transistor.

Figure 3.5*c* presents the graphical symbols for a NOT gate. The left symbol shows the input, output, power, and ground terminals, and the right symbol is simplified to show only

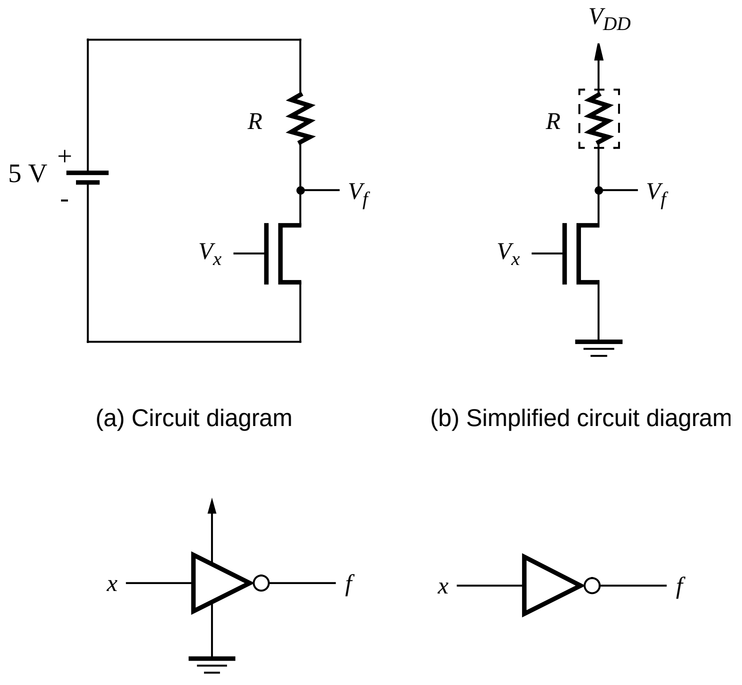
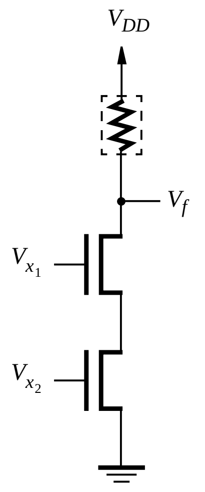


Figure 3.5 A NOT gate built using NMOS technology.

the input and output terminals. In practice only the simplified symbol is used. Another name often used for the NOT gate is *inverter*. We use both names interchangeably in this book.

In section 2.1 we saw that a series connection of switches corresponds to the logic AND function, while a parallel connection represents the OR function. Using NMOS transistors, we can implement the series connection as depicted in Figure 3.6*a*. If $V_{x_1} = V_{x_2} = 5$ V, both transistors will be on and V_f will be close to 0 V. But if either V_{x_1} or V_{x_2} is 0, then no current will flow through the series-connected transistors and V_f will be pulled up to 5 V. The resulting truth table for f, provided in terms of logic values, is given in Figure 3.6*b*. The realized function is the complement of the AND function, called the *NAND* function, for NOT-AND. The circuit realizes a NAND gate. Its graphical symbols are shown in Figure 3.6*c*.

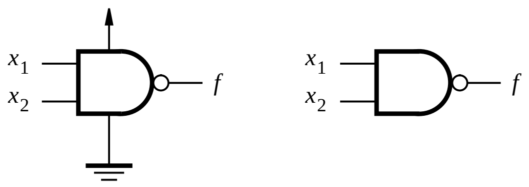
The parallel connection of NMOS transistors is given in Figure 3.7*a*. Here, if either $V_{x_1} = 5$ V or $V_{x_2} = 5$ V, then V_f will be close to 0 V. Only if both V_{x_1} and V_{x_2} are 0 will V_f be pulled up to 5 V. A corresponding truth table is given in Figure 3.7*b*. It shows that the circuit realizes the complement of the OR function, called the *NOR* function, for NOT-OR. The graphical symbols for the NOR gate appear in Figure 3.7*c*.



(a) Circuit

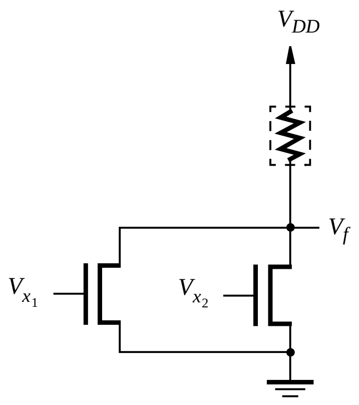
x_1	x_2	f
0	0	1
0	1	1
1	0	1
1	1	0

(b) Truth table



(c) Graphical symbols

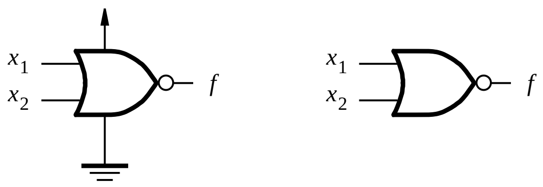
Figure 3.6 NMOS realization of a NAND gate.



(a) Circuit

x_1	x_2	f
0	0	1
0	1	0
1	0	0
1	1	0

(b) Truth table



(c) Graphical symbols

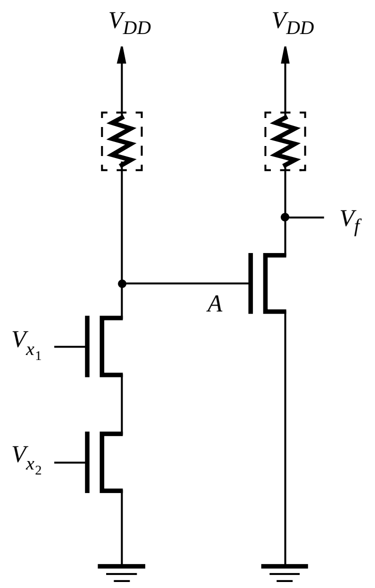
Figure 3.7 NMOS realization of a NOR gate.

Instead of the NAND and NOR gates just described, the reader would naturally be interested in the AND and OR gates that were used extensively in the previous chapter. Figure 3.8 indicates how an AND gate is built in NMOS technology by following a NAND gate with an inverter. Node A realizes the NAND of inputs x_1 and x_2, and f represents the AND function. In a similar fashion an OR gate is realized as a NOR gate followed by an inverter, as depicted in Figure 3.9.

3.3 CMOS Logic Gates

So far we have considered how to implement logic gates using NMOS transistors. For each of the circuits that has been presented, it is possible to derive an equivalent circuit that uses PMOS transistors. However, it is more interesting to consider how both NMOS and PMOS transistors can be used together. The most popular such approach is known as CMOS technology. We will see in section 3.8 that CMOS technology offers some attractive practical advantages in comparison to NMOS technology.

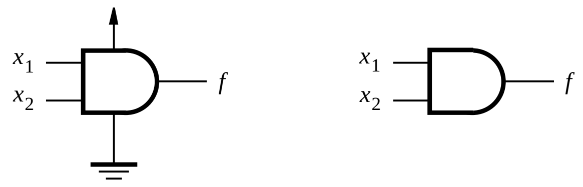
In NMOS circuits the logic functions are realized by arrangements of NMOS transistors, combined with a pull-up device that acts as a resistor. We will refer to the part of the circuit



(a) Circuit

x_1	x_2	f
0	0	0
0	1	0
1	0	0
1	1	1

(b) Truth table

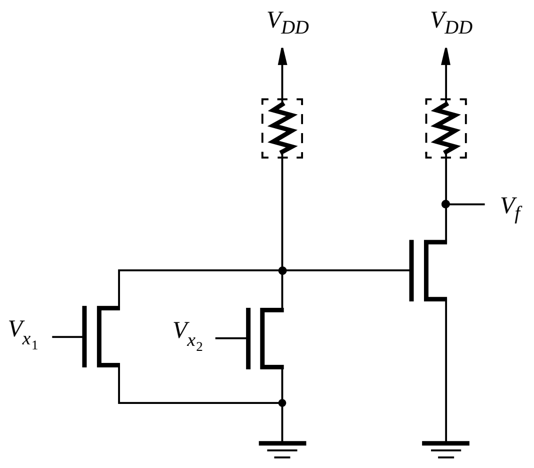


(c) Graphical symbols

Figure 3.8 NMOS realization of an AND gate.

that involves NMOS transistors as the *pull-down network (PDN)*. Then the structure of the circuits in Figures 3.5 through 3.9 can be characterized by the block diagram in Figure 3.10. The concept of CMOS circuits is based on replacing the pull-up device with a *pull-up network (PUN)* that is built using PMOS transistors, such that the functions realized by the PDN and PUN networks are complements of each other. Then a logic circuit, such as a typical logic gate, is implemented as indicated in Figure 3.11. For any given valuation of the input signals, either the PDN pulls V_f down to *Gnd* or the PUN pulls V_f up to V_{DD}. The PDN and the PUN have equal numbers of transistors, which are arranged so that the two networks are *duals* of one another. Wherever the PDN has NMOS transistors in series, the PUN has PMOS transistors in parallel, and vice versa.

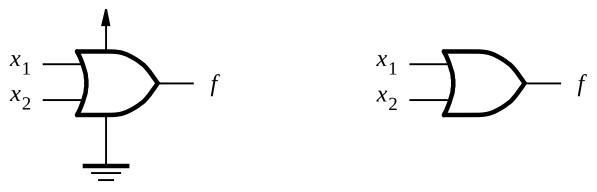
The simplest example of a CMOS circuit, a NOT gate, is shown in Figure 3.12. When $V_x = 0$ V, transistor T_2 is off and transistor T_1 is on. This makes $V_f = 5$ V, and since T_2 is



(a) Circuit

x_1	x_2	f
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table



(c) Graphical symbols

Figure 3.9 NMOS realization of an OR gate.

off, no current flows through the transistors. When $V_x = 5$ V, T_2 is on and T_1 is off. Thus $V_f = 0$ V, and no current flows because T_1 is off.

A key point is that no current flows in a CMOS inverter when the input is either low or high. This is true for all CMOS circuits; no current flows, and hence no power is dissipated under steady state conditions. This property has led to CMOS becoming the most popular technology in use today for building logic circuits. We will discuss current flow and power dissipation in detail in section 3.8.

Figure 3.13 provides a circuit diagram of a CMOS NAND gate. It is similar to the NMOS circuit presented in Figure 3.6 except that the pull-up device has been replaced by the PUN with two PMOS transistors connected in parallel. The truth table in the figure specifies the state of each of the four transistors for each logic valuation of inputs x_1 and x_2. The reader can verify that the circuit properly implements the NAND function. Under static conditions no path exists for current flow from V_{DD} to *Gnd*.

The circuit in Figure 3.13 can be derived from the logic expression that defines the NAND operation, $f = \overline{x_1 x_2}$. This expression specifies the conditions for which $f = 1$; hence it defines the PUN. Since the PUN consists of PMOS transistors, which are turned on when their control (gate) inputs are set to 0, an input variable x_i turns on a transistor if

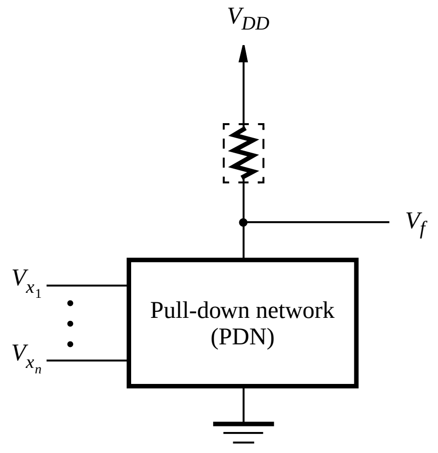


Figure 3.10 Structure of an NMOS circuit.

$x_i = 0$. From DeMorgan's law, we have

$$f = \overline{x_1 x_2} = \overline{x}_1 + \overline{x}_2$$

Thus $f = 1$ when *either* input x_1 or x_2 has the value 0, which means that the PUN must have two PMOS transistors connected in parallel. The PDN must implement the complement of f, which is

$$\overline{f} = x_1 x_2$$

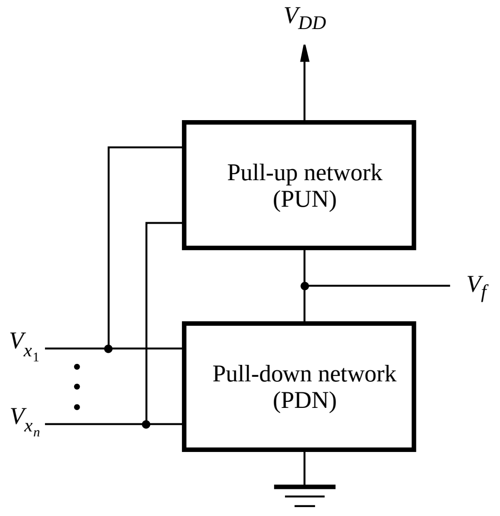
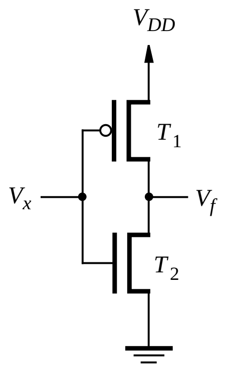


Figure 3.11 Structure of a CMOS circuit.



x	T_1	T_2	f
0	on	off	1
1	off	on	0

(a) Circuit (b) Truth table and transistor states

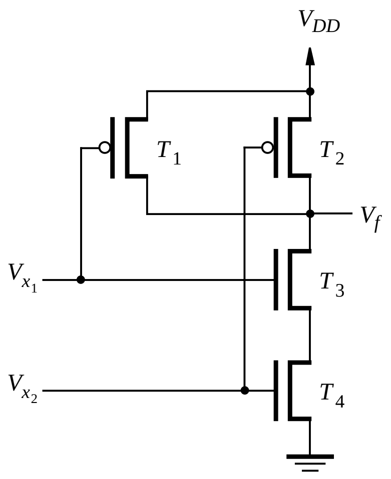
Figure 3.12 CMOS realization of a NOT gate.

Since $\overline{f} = 1$ when *both* x_1 and x_2 are 1, it follows that the PDN must have two NMOS transistors connected in series.

The circuit for a CMOS NOR gate is derived from the logic expression that defines the NOR operation

$$f = \overline{x_1 + x_2} = \overline{x}_1\overline{x}_2$$

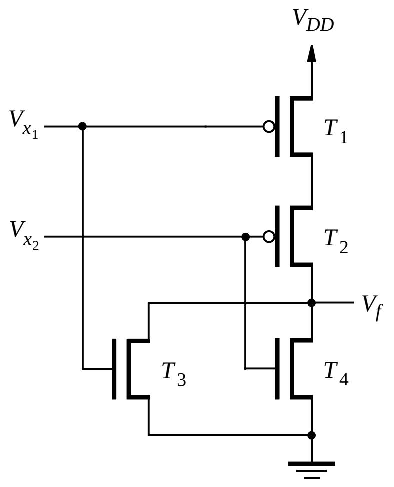
Since $f = 1$ only if both x_1 and x_2 have the value 0, then the PUN consists of two PMOS transistors connected in series. The PDN, which realizes $\overline{f} = x_1 + x_2$, has two NMOS transistors in parallel, leading to the circuit shown in Figure 3.14.



x_1	x_2	T_1	T_2	T_3	T_4	f
0	0	on	on	off	off	1
0	1	on	off	off	on	1
1	0	off	on	on	off	1
1	1	off	off	on	on	0

(a) Circuit (b) Truth table and transistor states

Figure 3.13 CMOS realization of a NAND gate.



(a) Circuit

x_1	x_2	T_1	T_2	T_3	T_4	f
0	0	on	on	off	off	1
0	1	on	off	off	on	0
1	0	off	on	on	off	0
1	1	off	off	on	on	0

(b) Truth table and transistor states

Figure 3.14 CMOS realization of a NOR gate.

A CMOS AND gate is built by connecting a NAND gate to an inverter, as illustrated in Figure 3.15. Similarly, an OR gate is constructed with a NOR gate followed by a NOT gate.

The above procedure for deriving a CMOS circuit can be applied to more general logic functions to create *complex gates*. This process is illustrated in the following two examples.

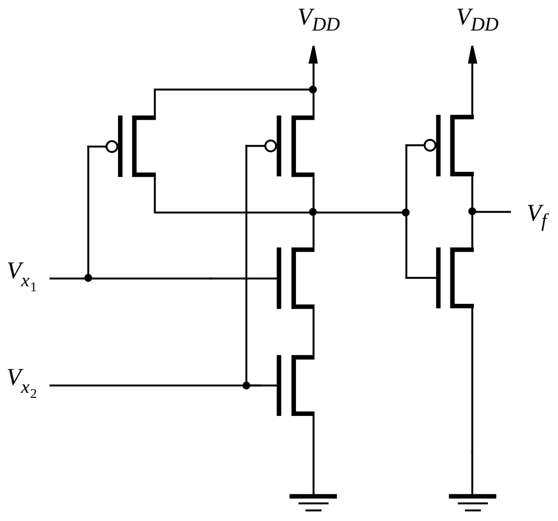


Figure 3.15 CMOS realization of an AND gate.

Example 3.1 Consider the function

$$f = \overline{x}_1 + \overline{x}_2\overline{x}_3$$

Since all variables appear in their complemented form, we can directly derive the PUN. It consists of a PMOS transistor controlled by x_1 in parallel with a series combination of PMOS transistors controlled by x_2 and x_3. For the PDN we have

$$\overline{f} = \overline{\overline{x}_1 + \overline{x}_2\overline{x}_3} = x_1(x_2 + x_3)$$

This expression gives the PDN that has an NMOS transistor controlled by x_1 in series with a parallel combination of NMOS transistors controlled by x_2 and x_3. The circuit is shown in Figure 3.16.

Example 3.2 Consider the function

$$f = \overline{x}_1 + (\overline{x}_2 + \overline{x}_3)\overline{x}_4$$

Then

$$\overline{f} = x_1(x_2x_3 + x_4)$$

These expressions lead directly to the circuit in Figure 3.17.

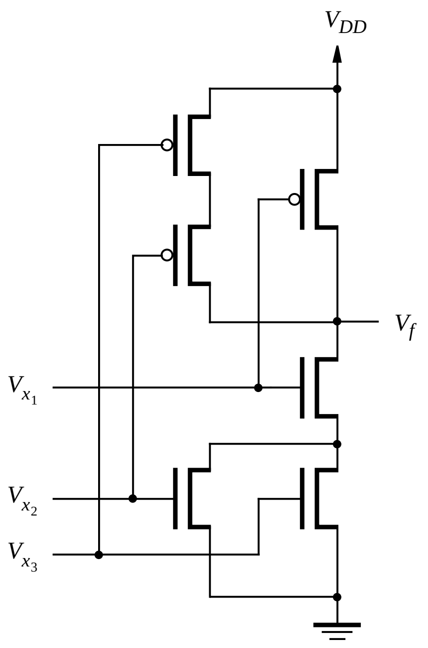


Figure 3.16 The circuit for Example 3.1.

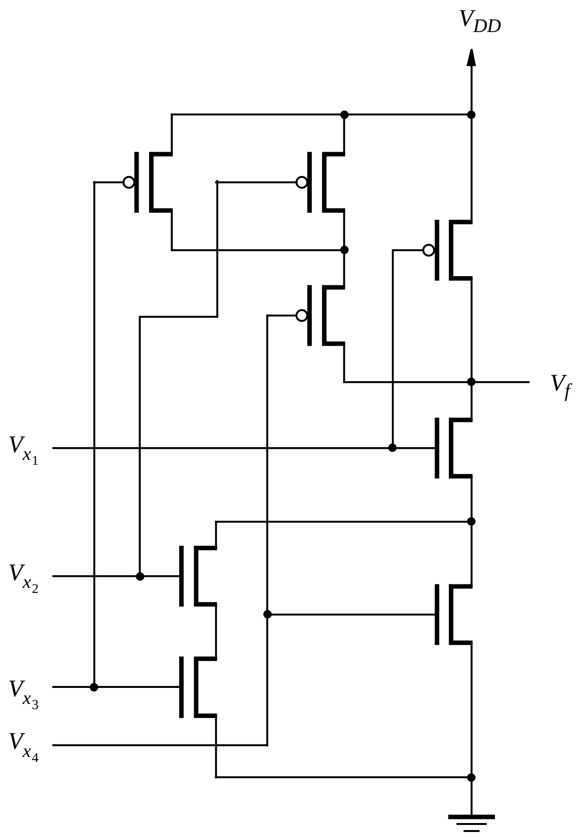


Figure 3.17 The circuit for Example 3.2.

The circuits in Figures 3.16 and 3.17 show that it is possible to implement fairly complex logic functions using combinations of series and parallel connections of transistors (acting as switches), without implementing each series or parallel connection as a complete AND (using the structure introduced in Figure 3.15) or OR gate.

3.3.1 Speed of Logic Gate Circuits

In the preceding sections we have assumed that transistors operate as ideal switches that present no resistance to current flow. Hence, while we have derived circuits that realize the functionality needed in logic gates, we have ignored the important issue of the speed of operation of the circuits. In reality transistor switches have a significant resistance when turned on. Also, transistor circuits include capacitors, which are created as a side effect of the manufacturing process. These factors affect the amount of time required for signal values to propagate through logic gates. We provide a detailed discussion of the speed of logic circuits, as well as a number of other practical issues, in section 3.8.

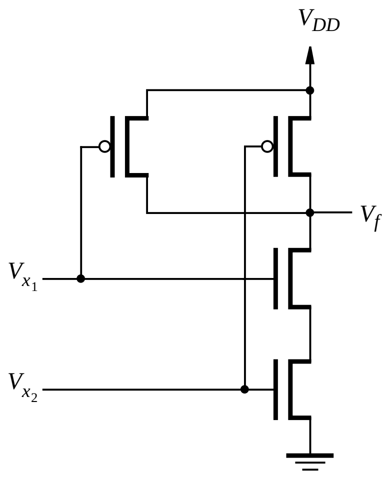
3.4 Negative Logic System

At the beginning of this chapter, we said that logic values are represented as two distinct ranges of voltage levels. We are using the convention that the higher voltage levels represent logic value 1 and the lower voltages represent logic value 0. This convention is known as the positive logic system, and it is the one used in most practical applications. In this section we briefly consider the negative logic system in which the association between voltage levels and logic values is reversed.

Let us reconsider the CMOS circuit in Figure 3.13, which is reproduced in Figure 3.18*a*. Part (*b*) of the figure gives a truth table for the circuit, but the table shows voltage levels instead of logic values. In this table, L refers to the low voltage level in the circuit, which is 0 V, and H represents the high voltage level, which is V_{DD}. This is the style of truth table that manufacturers of integrated circuits often use in data sheets to describe the functionality of the chips. It is entirely up to the user of the chip as to whether L and H are interpreted in terms of logic values such that $L = 0$ and $H = 1$, or $L = 1$ and $H = 0$.

Figure 3.19*a* illustrates the positive logic interpretation in which $L = 0$ and $H = 1$. As we already know from the discussions of Figure 3.13, the circuit represents a NAND gate under this interpretation. The opposite interpretation is shown in Figure 3.19*b*. Here negative logic is used so that $L = 1$ and $H = 0$. The truth table specifies that the circuit represents a NOR gate in this case. Note that the truth table rows are listed in the opposite order from what we normally use, to be consistent with the L and H values in Figure 3.18*b*. Figure 3.19*b* also gives the logic gate symbol for the NOR gate, which includes small triangles on the gate's terminals to indicate that the negative logic system is used.

As another example, consider again the circuit in Figure 3.15. Its truth table, in terms of voltage levels, is given in Figure 3.20*a*. Using the positive logic system, this circuit



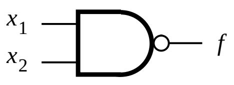
(a) Circuit

V_{x_1}	V_{x_2}	V_f
L	L	H
L	H	H
H	L	H
H	H	L

(b) Voltage levels

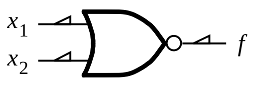
Figure 3.18 Voltage levels in the circuit in Figure 3.13.

x_1	x_2	f
0	0	1
0	1	1
1	0	1
1	1	0



(a) Positive logic truth table and gate symbol

x_1	x_2	f
1	1	0
1	0	0
0	1	0
0	0	1



(b) Negative logic truth table and gate symbol

Figure 3.19 Interpretation of the circuit in Figure 3.18.

represents an AND gate, as indicated in Figure 3.20*b*. But using the negative logic system, the circuit represents an OR gate, as depicted in Figure 3.20*c*.

It is possible to use a mixture of positive and negative logic in a single circuit, which is known as a *mixed logic system*. In practice, the positive logic system is used in most applications. We will not consider the negative logic system further in this book.

3.5 STANDARD CHIPS

In Chapter 1 we mentioned that several different types of integrated circuit chips are available for implementation of logic circuits. We now discuss the available choices in some detail.

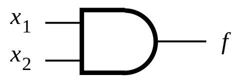
3.5.1 7400-SERIES STANDARD CHIPS

An approach used widely until the mid-1980s was to connect together multiple chips, each containing only a few logic gates. A wide assortment of chips, with different types of logic gates, is available for this purpose. They are known as 7400-series parts because the chip part numbers always begin with the digits 74. An example of a 7400-series part is given in Figure 3.21. Part (*a*) of the figure shows a type of package that the chip is provided in, called a *dual-inline package (DIP)*. Part (*b*) illustrates the 7404 chip, which comprises six NOT gates. The chip's external connections are called *pins* or *leads*. Two pins are used to connect to V_{DD} and *Gnd*, and other pins provide connections to the NOT gates. Many

V_{x_1}	V_{x_2}	V_f
L	L	L
L	H	L
H	L	L
H	H	H

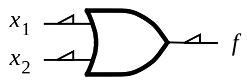
(a) Voltage levels

x_1	x_2	f
0	0	0
0	1	0
1	0	0
1	1	1



(b) Positive logic

x_1	x_2	f
1	1	1
1	0	1
0	1	1
0	0	0



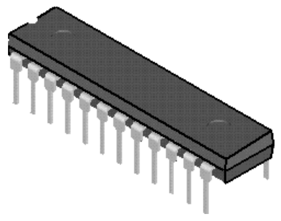
(c) Negative logic

Figure 3.20 Interpretation of the circuit in Figure 3.15.

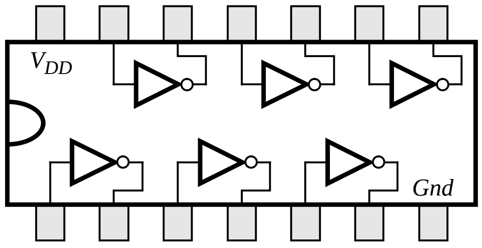
7400-series chips exist, and they are described in the data books produced by manufacturers of these chips [3–7]. Diagrams of some of the chips are also included in several textbooks, such as [8–12].

The 7400-series chips are produced in standard forms by a number of integrated circuit manufacturers, using agreed-upon specifications. Competition among various manufacturers works to the designer's advantage because it tends to lower the price of chips and ensures that parts are always readily available. For each specific 7400-series chip, several variants are built with different technologies. For instance, the part called 74LS00 is built with a technology called transistor-transistor logic (TTL), which is described in Appendix E, whereas the 74HC00 is fabricated using CMOS technology. In general, the most popular chips used today are the CMOS variants.

As an example of how a logic circuit can be implemented using 7400-series chips, consider the function $f = x_1x_2 + \overline{x}_2x_3$, which is shown in the form of a logic diagram in



(a) Dual-inline package



(b) Structure of 7404 chip

Figure 3.21 A 7400-series chip.

Figure 2.26. A NOT gate is required to produce $\overline{x}_2$, as well as 2 two-input AND gates and a two-input OR gate. Figure 3.22 shows three 7400-series chips that can be used to implement the function. We assume that the three input signals x_1, x_2, and x_3 are produced as outputs by some other circuitry that can be connected by wires to the three chips. Notice that power and ground connections are included for all three chips. This example makes use of only a portion of the gates available on the three chips, hence the remaining gates can be used to realize other functions.

Because of their low logic capacity, the standard chips are seldom used in practice today, with one exception. Many modern products include standard chips that contain buffers. Buffers are logic gates that are usually used to improve the speed of circuits. An example of a buffer chip is depicted in Figure 3.23. It is the 74244 chip, which comprises eight *tri-state buffers*. We describe how tri-state buffers work in section 3.8.8. Rather than showing how the buffers are arranged inside the chip package, as we did for the NOT gates in Figure 3.21, we show only the pin numbers of the package pins that are connected to the buffers. The package has 20 pins, and they are numbered in the same manner as shown for Figure 3.21; *Gnd* and V_{DD} connections are provided on pins 10 and 20, respectively. Many other buffer chips also exist. For example, the 162244 chip has 16 tri-state buffers. It is part of a family of devices that are similar to the 7400-series chips but with twice as many gates in each chip. These chips are available in multiple types of packages, with the most popular being a *small-outline integrated circuit (SOIC)* package. An SOIC package has a similar shape to a DIP, but the SOIC is considerably smaller in physical size.

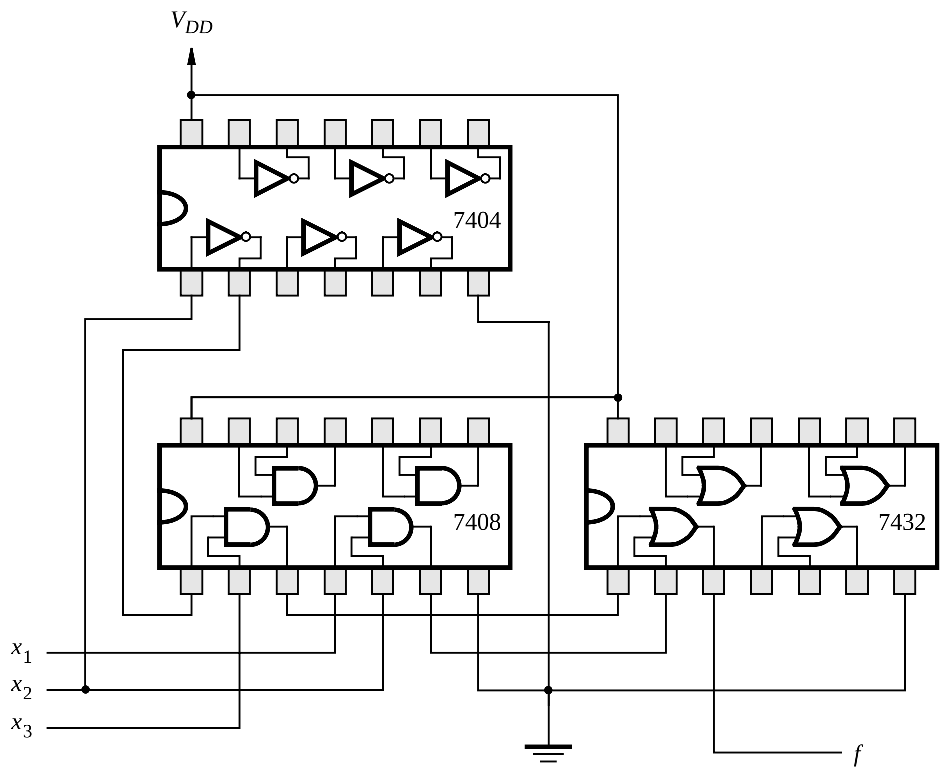


Figure 3.22 An implementation of $f = x_1x_2 + \bar{x}_2x_3$.

As integrated circuit technology has improved over time, a system of classifying chips according to their size has evolved. The earliest chips produced, such as the 7400-series chips, comprise only a few logic gates. The technology used to produce these chips is referred to as *small-scale integration (SSI)*. Chips that include slightly more logic circuitry, typically about 10 to 100 gates, represent *medium-scale integration (MSI)*. Until the mid-1980s chips that were too large to qualify as MSI were classified as *large-scale integration*

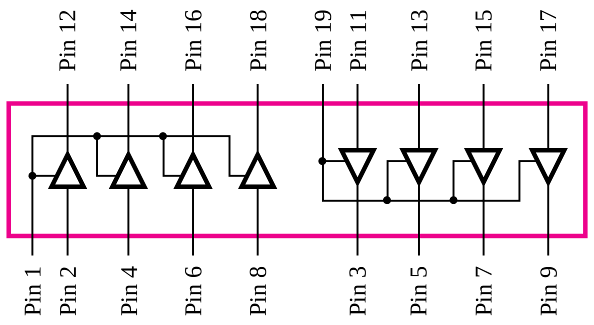


Figure 3.23 The 74244 buffer chip.

(LSI). In recent years the concept of classifying circuits according to their size has become of little practical use. Most integrated circuits today contain many thousands or millions of transistors. Regardless of their exact size, these large chips are said to be made with *very large scale integration (VLSI)* technology. The trend in digital hardware products is to integrate as much circuitry as possible onto a single chip. Thus most of the chips used today are built with VLSI technology, and the older types of chips are used rarely.

3.6 PROGRAMMABLE LOGIC DEVICES

The function provided by each of the 7400-series parts is fixed and cannot be tailored to suit a particular design situation. This fact, coupled with the limitation that each chip contains only a few logic gates, makes these chips inefficient for building large logic circuits. It is possible to manufacture chips that contain relatively large amounts of logic circuitry with a structure that is not fixed. Such chips were first introduced in the 1970s and are called *programmable logic devices (PLDs)*.

A PLD is a general-purpose chip for implementing logic circuitry. It contains a collection of logic circuit elements that can be customized in different ways. A PLD can be viewed as a "black box" that contains logic gates and programmable switches, as illustrated in Figure 3.24. The programmable switches allow the logic gates inside the PLD to be connected together to implement whatever logic circuit is needed.

3.6.1 PROGRAMMABLE LOGIC ARRAY (PLA)

Several types of PLDs are commercially available. The first developed was the *programmable logic array (PLA)*. The general structure of a PLA is depicted in Figure 3.25. Based on the idea that logic functions can be realized in sum-of-products form, a PLA

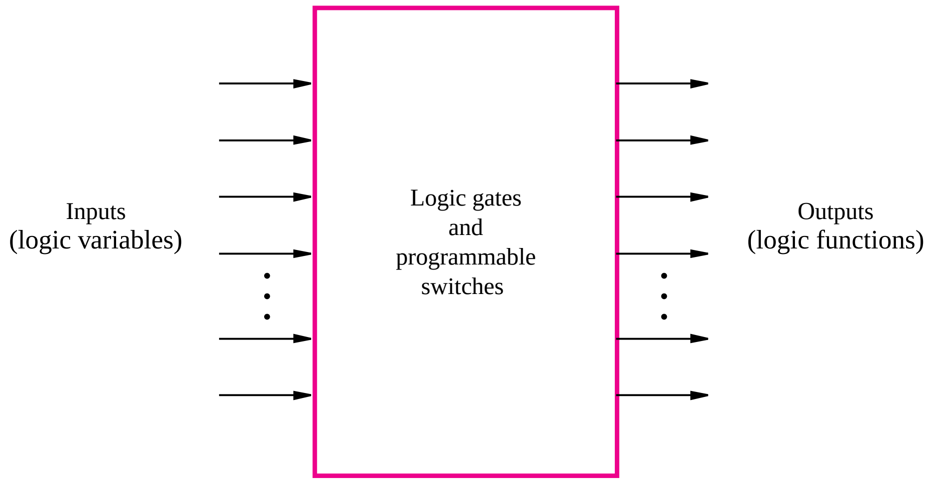


Figure 3.24 Programmable logic device as a black box.

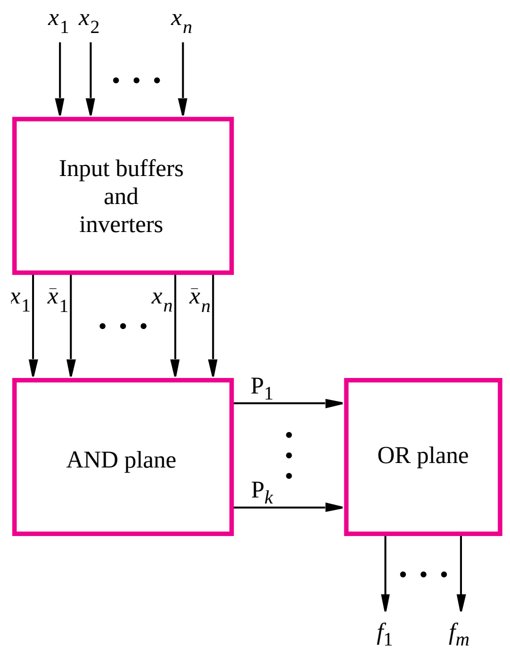

x_1 x_2 x_n
Input buffers and inverters
x_1 $\bar{x}_1$ x_n $\bar{x}_n$
AND plane
P_1
P_k
OR plane
f_1 f_m

Figure 3.25 General structure of a PLA.

comprises a collection of AND gates that feeds a set of OR gates. As shown in the figure, the PLA's inputs $x_1, \ldots, x_n$ pass through a set of buffers (which provide both the true value and complement of each input) into a circuit block called an *AND plane*, or *AND array*. The AND plane produces a set of product terms $P_1, \ldots, P_k$. Each of these terms can be configured to implement any AND function of $x_1, \ldots, x_n$. The product terms serve as the inputs to an *OR plane*, which produces the outputs $f_1, \ldots, f_m$. Each output can be configured to realize any sum of $P_1, \ldots, P_k$ and hence any sum-of-products function of the PLA inputs.

A more detailed diagram of a small PLA is given in Figure 3.26, which shows a PLA with three inputs, four product terms, and two outputs. Each AND gate in the AND plane has six inputs, corresponding to the true and complemented versions of the three input signals. Each connection to an AND gate is programmable; a signal that is connected to an AND gate is indicated with a wavy line, and a signal that is not connected to the gate is shown with a broken line. The circuitry is designed such that any unconnected AND-gate inputs do not affect the output of the AND gate. In commercially available PLAs, several methods of realizing the programmable connections exist. Detailed explanation of how a PLA can be built using transistors is given in section 3.10.

In Figure 3.26 the AND gate that produces P_1 is shown connected to the inputs x_1 and x_2. Hence $P_1 = x_1x_2$. Similarly, $P_2 = x_1\bar{x}_3$, $P_3 = \bar{x}_1\bar{x}_2x_3$, and $P_4 = x_1x_3$. Programmable connections also exist for the OR plane. Output f_1 is connected to product terms P_1, P_2, and P_3. It therefore realizes the function $f_1 = x_1x_2 + x_1\bar{x}_3 + \bar{x}_1\bar{x}_2x_3$. Similarly, output

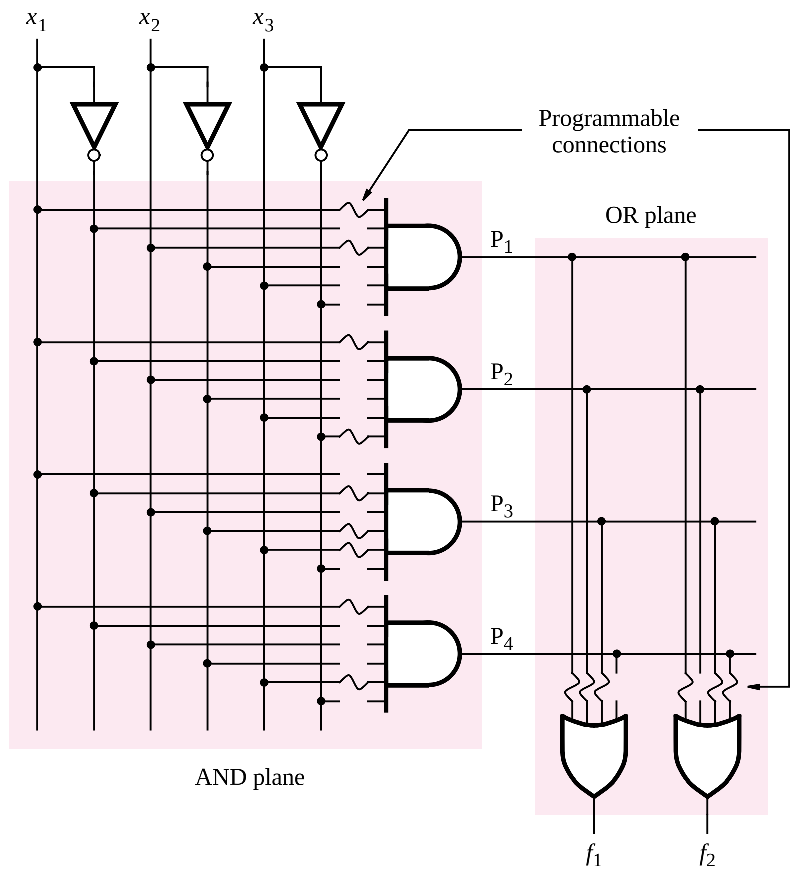


Figure 3.26 Gate-level diagram of a PLA.

$f_2 = x_1x_2 + \overline{x}_1\overline{x}_2x_3 + x_1x_3$. Although Figure 3.26 depicts the PLA programmed to implement the functions described above, by programming the AND and OR planes differently, each of the outputs f_1 and f_2 could implement various functions of x_1, x_2, and x_3. The only constraint on the functions that can be implemented is the size of the AND plane because it produces only four product terms. Commercially available PLAs come in larger sizes than we have shown here. Typical parameters are 16 inputs, 32 product terms, and eight outputs.

Although Figure 3.26 illustrates clearly the functional structure of a PLA, this style of drawing is awkward for larger chips. Instead, it has become customary in technical literature to use the style shown in Figure 3.27. Each AND gate is depicted as a single horizontal line attached to an AND-gate symbol. The possible inputs to the AND gate are drawn as vertical lines that cross the horizontal line. At any crossing of a vertical and horizontal line, a programmable connection, indicated by an X, can be made. Figure 3.27 shows the programmable connections needed to implement the product terms in Figure 3.26. Each OR gate is drawn in a similar manner, with a vertical line attached to an OR-gate symbol.

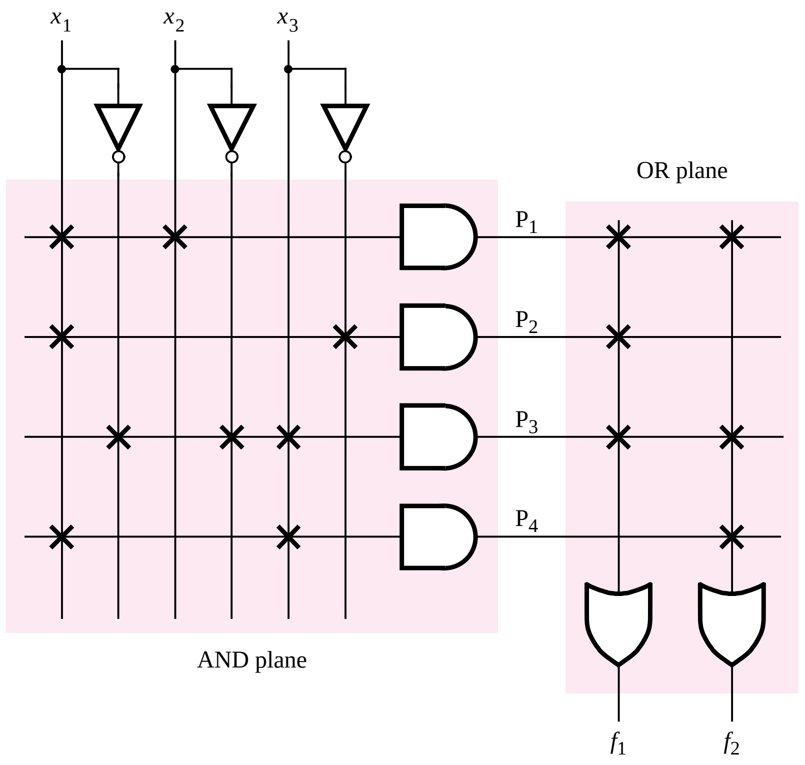


Figure 3.27 Customary schematic for the PLA in Figure 3.26.

The AND-gate outputs cross these lines, and corresponding programmable connections can be formed. The figure illustrates the programmable connections that produce the functions f_1 and f_2 from Figure 3.26.

The PLA is efficient in terms of the area needed for its implementation on an integrated circuit chip. For this reason, PLAs are often included as part of larger chips, such as microprocessors. In this case a PLA is created so that the connections to the AND and OR gates are fixed, rather than programmable. In section 3.10 we will show that both fixed and programmable PLAs can be created with similar structures.

3.6.2 Programmable Array Logic (PAL)

In a PLA both the AND and OR planes are programmable. Historically, the programmable switches presented two difficulties for manufacturers of these devices: they were hard to fabricate correctly, and they reduced the speed-performance of circuits implemented in the PLAs. These drawbacks led to the development of a similar device in which the AND plane is programmable, but the OR plane is fixed. Such a chip is known as a *programmable array logic (PAL)* device. Because they are simpler to manufacture, and thus less expensive than PLAs, and offer better performance, PALs have become popular in practical applications.

An example of a PAL with three inputs, four product terms, and two outputs is given in Figure 3.28. The product terms P_1 and P_2 are hardwired to one OR gate, and P_3 and P_4 are hardwired to the other OR gate. The PAL is shown programmed to realize the two logic functions $f_1 = x_1x_2\overline{x}_3 + \overline{x}_1x_2x_3$ and $f_2 = \overline{x}_1\overline{x}_2 + x_1x_2x_3$. In comparison to the PLA in Figure 3.27, the PAL offers less flexibility; the PLA allows up to four product terms per OR gate, whereas the OR gates in the PAL have only two inputs. To compensate for the reduced flexibility, PALs are manufactured in a range of sizes, with various numbers of inputs and outputs, and different numbers of inputs to the OR gates. An example of a commercial PAL is given in Appendix E.

So far we have assumed that the OR gates in a PAL, as in a PLA, connect directly to the output pins of the chip. In many PALs extra circuitry is added at the output of each OR gate to provide additional flexibility. It is customary to use the term *macrocell* to refer to the OR gate combined with the extra circuitry. An example of the flexibility that may be provided in a macrocell is given in Figure 3.29. The symbol labeled *flip-flop* represents a memory element. It stores the value produced by the OR gate output at a particular point in time and can hold that value indefinitely. The flip-flop is controlled by the signal called *clock*. When *clock* makes a transition from logic value 0 to 1, the flip-flop stores the value at its *D* input at that time and this value appears at the flip-flop's Q output. Flip-flops are used for implementing many types of logic circuits, as we will show in Chapter 7.

In section 2.7.2 we discussed a 2-to-1 multiplexer circuit. It has two data inputs, a select input, and one output. The select input is used to choose one of the data inputs as

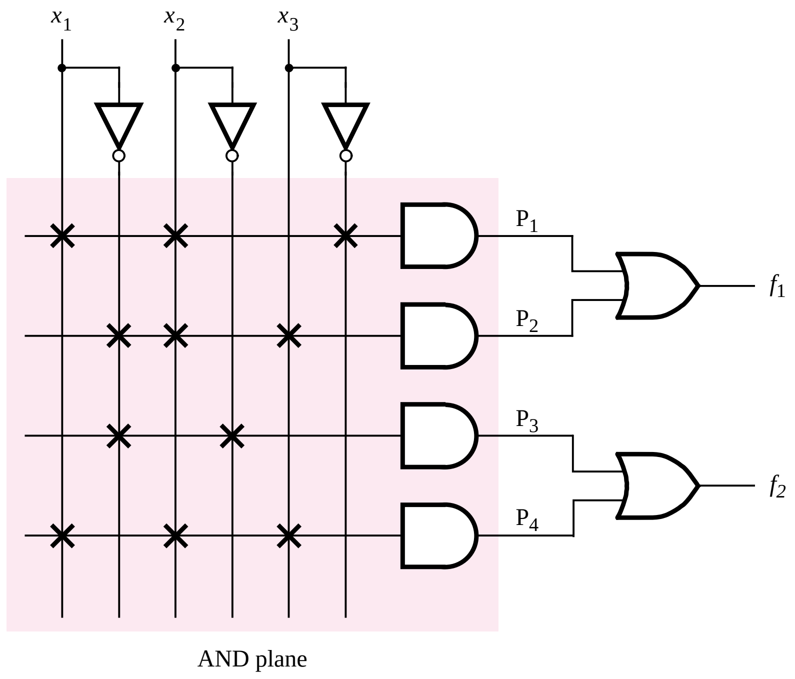


Figure 3.28 An example of a PAL.

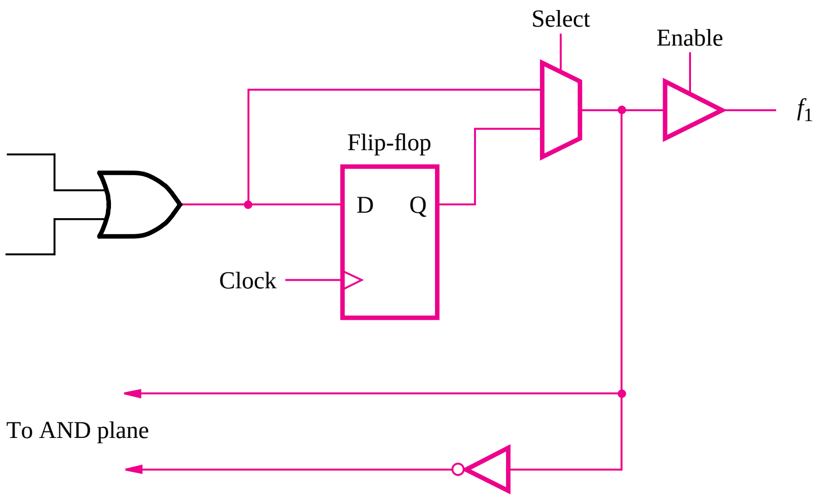


Figure 3.29 Extra circuitry added to OR-gate outputs from Figure 3.28.

the multiplexer's output. In Figure 3.29 a 2-to-1 multiplexer selects as an output from the PAL either the OR-gate output or the flip-flop output. The multiplexer's select line can be programmed to be either 0 or 1. Figure 3.29 shows another logic gate, called a tri-state buffer, connected between the multiplexer and the PAL output. We discuss tri-state buffers in section 3.8.8. Finally, the multiplexer's output is "fed back" to the AND plane in the PAL. This feedback connection allows the logic function produced by the multiplexer to be used internally in the PAL, which allows the implementation of circuits that have multiple stages, or levels, of logic gates.

A number of companies manufacture PLAs or PALs, or other, similar types of *simple PLDs (SPLDs)*. A partial list of companies, and the types of SPLDs that they manufacture, is given in Appendix E. An interested reader can examine the information that these companies provide on their products, which is available on the World Wide Web (WWW). The WWW locator for each company is given in Table E.1 in Appendix E.

3.6.3 PROGRAMMING OF PLAS AND PALS

In Figures 3.27 and 3.28, each connection between a logic signal in a PLA or PAL and the AND/OR gates is shown as an X. We describe how these switches are implemented using transistors in section 3.10. Users' circuits are implemented in the devices by *configuring*, or *programming*, these switches. Commercial chips contain a few thousand programmable switches; hence it is not feasible for a user of these chips to specify manually the desired programming state of each switch. Instead, CAD systems are employed for this purpose. We introduced CAD tools in Chapter 2 and described methods for design entry and simulation of circuits. For CAD systems that support targeting of circuits to PLDs, the tools have the capability to automatically produce the necessary information for programming each of the switches in the device. A computer system that runs the CAD tools is connected by a cable to a dedicated *programming unit*. Once the user has completed the design of a circuit, the CAD tools generate a file, often called a *programming file* or *fuse map*, that specifies the state that each switch in the PLD should have, to realize correctly the designed circuit. The PLD is placed into the programming unit, and the programming file is transferred from the computer system. The programming unit then places the chip into a special *programming mode* and configures each switch individually. A photograph of a programming unit is shown in Figure 3.30. Several adaptors are shown beside the main unit; each adaptor is used for a specific type of chip package.

The programming procedure may take a few minutes to complete. Usually, the programming unit can automatically "read back" the state of each switch after programming, to verify that the chip has been programmed correctly. A detailed discussion of the process involved in using CAD tools to target designed circuits to programmable chips is given in Appendices B, C, and D.

PLAs or PALs used as part of a logic circuit usually reside with other chips on a printed circuit board (PCB). The procedure described above assumes that the chip can be removed from the circuit board for programming in the programming unit. Removal is made possible by using a socket on the PCB, as illustrated in Figure 3.31. Although PLAs and PALs are available in the DIP packages shown in Figure 3.21*a*, they are also available in another popular type of package, called a *plastic-leaded chip carrier (PLCC)*, which is depicted in Figure 3.31. On all four of its sides, the PLCC package has pins that "wrap around" the edges of the chip, rather than extending straight down as in the case of a DIP. The socket that houses the PLCC is attached by solder to the circuit board, and the PLCC is held in the socket by friction.

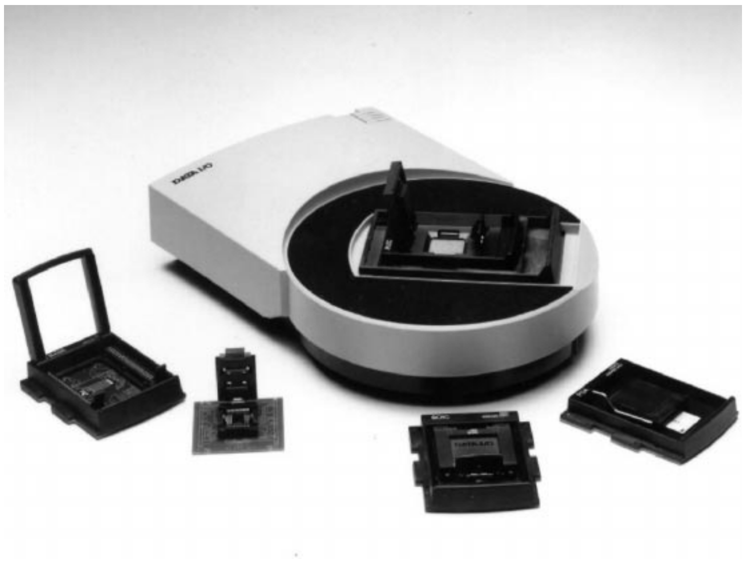


Figure 3.30 A PLD programming unit (courtesy of Data IO Corp.).

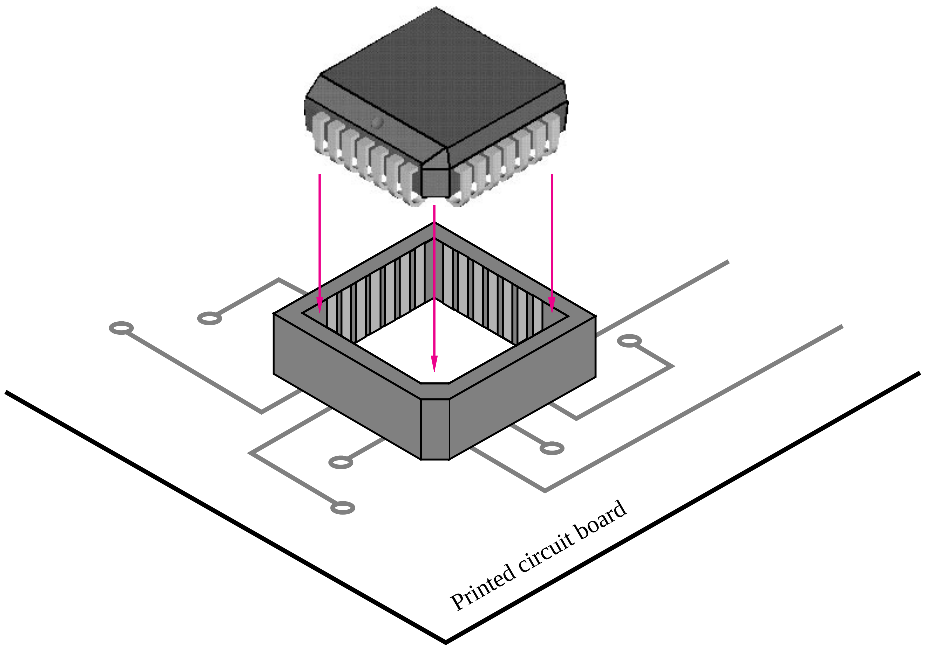


Figure 3.31 A PLCC package with socket.

Instead of relying on a programming unit to configure a chip, it would be advantageous to be able to perform the programming while the chip is still attached to its circuit board. This method of programming is called *in-system programming (ISP)*. It is not usually provided for PLAs or PALs, but is available for the more sophisticated chips that are described below.

3.6.4 Complex Programmable Logic Devices (CPLDs)

PLAs and PALs are useful for implementing a wide variety of small digital circuits. Each device can be used to implement circuits that do not require more than the number of inputs, product terms, and outputs that are provided in the particular chip. These chips are limited to fairly modest sizes, typically supporting a combined number of inputs plus outputs of not more than 32. For implementation of circuits that require more inputs and outputs, either multiple PLAs or PALs can be employed or else a more sophisticated type of chip, called a *complex programmable logic device (CPLD)*, can be used.

A CPLD comprises multiple circuit blocks on a single chip, with internal wiring resources to connect the circuit blocks. Each circuit block is similar to a PLA or a PAL; we will refer to the circuit blocks as *PAL-like blocks*. An example of a CPLD is given in Figure 3.32. It includes four PAL-like blocks that are connected to a set of *interconnection wires*. Each PAL-like block is also connected to a subcircuit labeled *I/O block*, which is attached to a number of the chip's input and output pins.

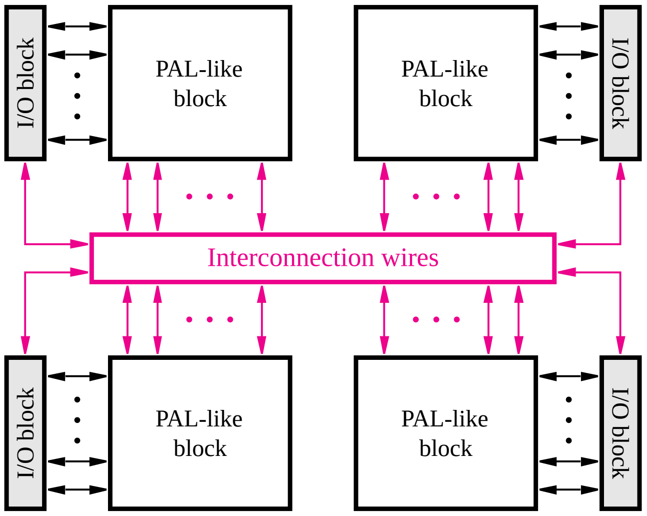


Figure 3.32 Structure of a complex programmable logic device (CPLD).

Figure 3.33 shows an example of the wiring structure and the connections to a PAL-like block in a CPLD. The PAL-like block includes 3 macrocells (real CPLDs typically have about 16 macrocells in a PAL-like block), each consisting of a four-input OR gate (real CPLDs usually provide between 5 and 20 inputs to each OR gate). The OR-gate output is connected to another type of logic gate that we have not yet introduced. It is called an Exclusive-OR (XOR) gate. We discuss XOR gates in section 3.9.1. The behavior of an XOR gate is the same as for an OR gate except that if both of the inputs are 1, the XOR gate produces a 0. One input to the XOR gate in Figure 3.33 can be programmably connected to 1 or 0; if 1, then the XOR gate complements the OR-gate output, and if 0, then the XOR gate has no effect. In many CPLDs the XOR gates can be used in other ways also, which we will see in Example 4.19, in Chapter 4. The macrocell also includes a flip-flop, a multiplexer, and a tri-state buffer. As we mentioned in the discussion for Figure 3.29, the flip-flop is used to store the output value produced by the OR gate. Each tri-state buffer (see section 3.8.8) is connected to a pin on the CPLD package. The tri-state buffer acts as a switch that allows each pin to be used either as an output from the CPLD or as an input. To use a pin as an output, the corresponding tri-state buffer is enabled, acting as a switch that is turned on. If the pin is to be used as an input, then the tri-state buffer is disabled, acting as a switch that is turned off. In this case an external source can drive a signal onto the pin, which can be connected to other macrocells using the interconnection wiring.

The interconnection wiring contains programmable switches that are used to connect the PAL-like blocks. Each of the horizontal wires can be connected to some of the vertical wires that it crosses, but not to all of them. Extensive research has been done to decide how many switches should be provided for connections between the wires. The number of switches is chosen to provide sufficient flexibility for typical circuits without wasting

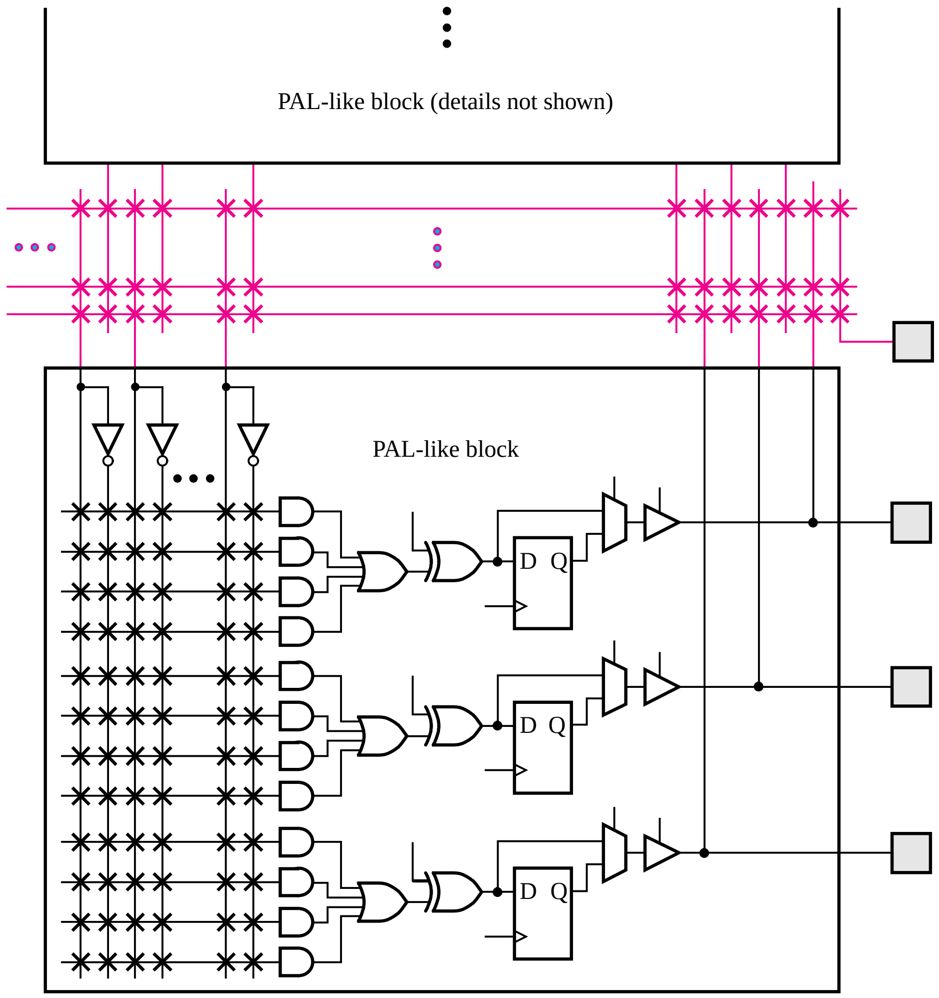
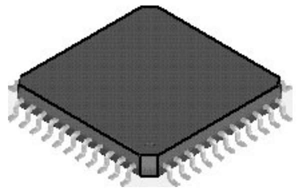


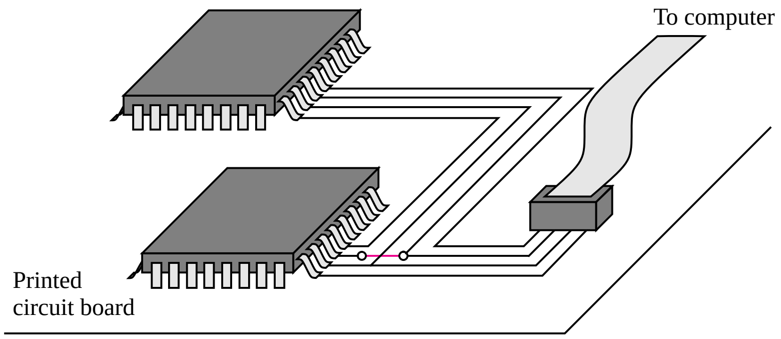
Figure 3.33 A section of the CPLD in Figure 3.32.

many switches in practice. One detail to note is that when a pin is used as an input, the macrocell associated with that pin cannot be used and is therefore wasted. Some CPLDs include additional connections between the macrocells and the interconnection wiring that avoids wasting macrocells in such situations.

Commercial CPLDs range in size from only 2 PAL-like blocks to more than 100 PAL-like blocks. They are available in a variety of packages, including the PLCC package that is shown in Figure 3.31. Figure 3.34*a* shows another type of package used to house CPLD chips, called a *quad flat pack (QFP)*. Like a PLCC package, the QFP package has pins on all four sides, but whereas the PLCC's pins wrap around the edges of the package, the QFP's pins extend outward from the package, with a downward-curving shape. The QFP's pins



(a) CPLD in a quad flat pack (QFP) package



(b) JTAG programming

Figure 3.34 CPLD packaging and programming.

are much thinner than those on a PLCC, which means that the package can support a larger number of pins; QFPs are available with more than 200 pins, whereas PLCCs are limited to fewer than 100 pins.

Most CPLDs contain the same type of programmable switches that are used in SPLDs, which are described in section 3.10. Programming of the switches may be accomplished using the same technique described in section 3.6.3, in which the chip is placed into a special-purpose programming unit. However, this programming method is rather inconvenient for large CPLDs for two reasons. First, large CPLDs may have more than 200 pins on the chip package, and these pins are often fragile and easily bent. Second, to be programmed in a programming unit, a socket is required to hold the chip. Sockets for large QFP packages are very expensive; they sometimes cost more than the CPLD device itself. For these reasons, CPLD devices usually support the ISP technique. A small connector is included on the PCB that houses the CPLD, and a cable is connected between that connector and a computer system. The CPLD is programmed by transferring the programming information generated by a CAD system through the cable, from the computer into the CPLD. The circuitry on the CPLD that allows this type of programming has been standardized by the IEEE and is usually called a *JTAG port*. It uses four wires to transfer information between the computer and the device being programmed. The term *JTAG* stands for Joint Test Action

Group. Figure 3.34*b* illustrates the use of a JTAG port for programming two CPLDs on a circuit board. The CPLDs are connected together so that both can be programmed using the same connection to the computer system. Once a CPLD is programmed, it retains the programmed state permanently, even when the power supply for the chip is turned off. This property is called *nonvolatile* programming.

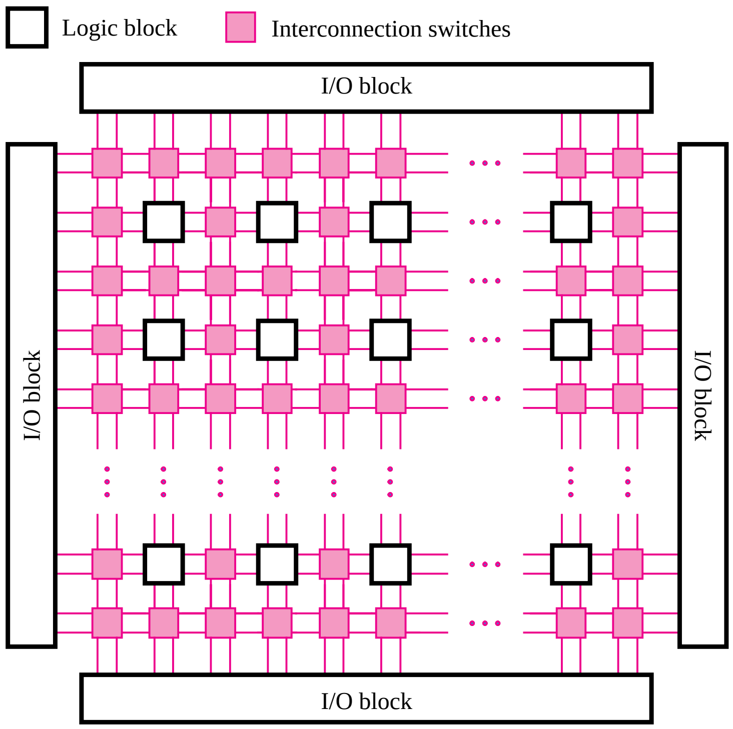
CPLDs are used for the implementation of many types of digital circuits. In industrial designs that employ some type of PLD device, CPLDs are used in about half the cases (SPLDs are used in only a small fraction of recently produced designs). A number of companies offer competing CPLDs. Appendix E lists, in Table E.2, the names of the major companies involved and shows the company's WWW locator. The reader is encouraged to examine the product information that each company provides on its Web pages. One example of a commercially available CPLD is described in detail in Appendix E. This CPLD family, manufactured by Altera and called the MAX 7000, is used in several examples presented later in the book.

3.6.5 FIELD-PROGRAMMABLE GATE ARRAYS

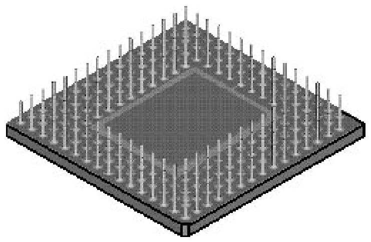
The types of chips described above, 7400 series, SPLDs, and CPLDs, are useful for implementation of a wide range of logic circuits. Except for CPLDs, these devices are rather small and are suitable only for relatively simple applications. Even for CPLDs, only moderately large logic circuits can be accommodated in a single chip. For cost and performance reasons, it is prudent to implement a desired logic circuit using as few chips as possible, so the amount of circuitry on a given chip and its functional capability are important. One way to quantify a circuit's *size* is to assume that the circuit is to be built using only simple logic gates and then estimate how many of these gates are needed. A commonly used measure is the total number of two-input NAND gates that would be needed to build the circuit; this measure is often called the number of *equivalent gates*.

Using the equivalent-gates metric, the size of a 7400-series chip is simple to measure because each chip contains only simple gates. For SPLDs and CPLDs the typical measure used is that each macrocell represents about 20 equivalent gates. Thus a typical PAL that has eight macrocells can accommodate a circuit that needs up to about 160 gates, and a large CPLD that has 1000 macrocells can implement circuits of up to about 20,000 equivalent gates.

By modern standards, a logic circuit with 20,000 gates is not large. To implement larger circuits, it is convenient to use a different type of chip that has a larger logic capacity. A *field-programmable gate array (FPGA)* is a programmable logic device that supports implementation of relatively large logic circuits. FPGAs are quite different from SPLDs and CPLDs because FPGAs do not contain AND or OR planes. Instead, FPGAs provide *logic blocks* for implementation of the required functions. The general structure of an FPGA is illustrated in Figure 3.35*a*. It contains three main types of resources: logic blocks, I/O blocks for connecting to the pins of the package, and interconnection wires and switches. The logic blocks are arranged in a two-dimensional array, and the interconnection wires are organized as horizontal and vertical *routing channels* between rows and columns of logic blocks. The routing channels contain wires and programmable switches that allow the logic blocks to be interconnected in many ways. Figure 3.35*a* shows two locations for



(a) General structure of an FPGA



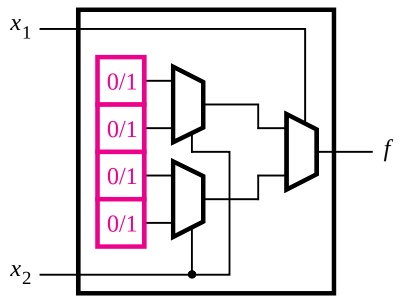
(b) Pin grid array (PGA) package (bottom view)

Figure 3.35 A field-programmable gate array (FPGA).

programmable switches; the blue boxes adjacent to logic blocks hold switches that connect the logic block input and output terminals to the interconnection wires, and the blue boxes that are diagonally between logic blocks connect one interconnection wire to another (such as a vertical wire to a horizontal wire). Programmable connections also exist between the I/O blocks and the interconnection wires. The actual number of programmable switches and wires in an FPGA varies in commercially available chips.

FPGAs can be used to implement logic circuits of more than a few hundred thousand equivalent gates in size. Two examples of FPGAs, called the Altera FLEX 10K and the Xilinx XC4000, are described in Appendix E. FPGAs are available in a variety of packages, including the PLCC and QFP packages described earlier. Figure 3.35*b* depicts another type of package, called a *pin grid array (PGA).* A PGA package may have up to a few hundred pins in total, which extend straight outward from the bottom of the package, in a grid pattern. Yet another packaging technology that has emerged is known as the *ball grid array (BGA).* The BGA is similar to the PGA except that the pins are small round balls, instead of posts. The advantage of BGA packages is that the pins are very small; hence more pins can be provided on the package.

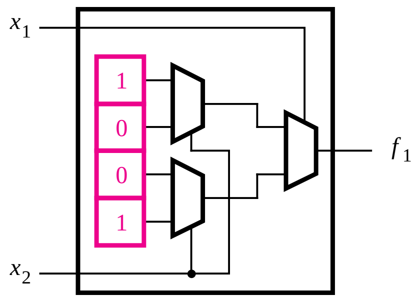
Each logic block in an FPGA typically has a small number of inputs and one output. A number of FPGA products are on the market, featuring different types of logic blocks. The most commonly used logic block is a *lookup table (LUT)*, which contains *storage cells* that are used to implement a small logic function. Each cell is capable of holding a single logic value, either 0 or 1. The stored value is produced as the output of the storage cell. LUTs of various *sizes* may be created, where the size is defined by the number of inputs. Figure 3.36*a* shows the structure of a small LUT. It has two inputs, x_1 and x_2, and one



(a) Circuit for a two-input LUT

x_1	x_2	f_1
0	0	1
0	1	0
1	0	0
1	1	1

(b) $f_1 = \bar{x}_1\bar{x}_2 + x_1x_2$



(c) Storage cell contents in the LUT

Figure 3.36 A two-input lookup table (LUT).

output, f. It is capable of implementing any logic function of two variables. Because a two-variable truth table has four rows, this LUT has four storage cells. One cell corresponds to the output value in each row of the truth table. The input variables x_1 and x_2 are used as the select inputs of three multiplexers, which, depending on the valuation of x_1 and x_2, select the content of one of the four storage cells as the output of the LUT. We introduced multiplexers in section 2.7.2 and will discuss storage cells in Chapter 10.

To see how a logic function can be realized in the two-input LUT, consider the truth table in Figure 3.36*b*. The function f_1 from this table can be stored in the LUT as illustrated in Figure 3.36*c*. The arrangement of multiplexers in the LUT correctly realizes the function f_1. When $x_1 = x_2 = 0$, the output of the LUT is driven by the top storage cell, which represents the entry in the truth table for $x_1x_2 = 00$. Similarly, for all valuations of x_1 and x_2, the logic value stored in the storage cell corresponding to the entry in the truth table chosen by the particular valuation appears on the LUT output. Providing access to the contents of storage cells is only one way in which multiplexers can be used to implement logic functions. A detailed presentation of the applications of multiplexers is given in Chapter 6.

Figure 3.37 shows a three-input LUT. It has eight storage cells because a three-variable truth table has eight rows. In commercial FPGA chips, LUTs usually have either four or five inputs, which require 16 and 32 storage cells, respectively. In Figure 3.29 we showed that PALs usually have extra circuitry included with their AND-OR gates. The same is true for FPGAs, which usually have extra circuitry, besides a LUT, in each logic block. Figure 3.38 shows how a flip-flop may be included in an FPGA logic block. As discussed for Figure 3.29, the flip-flop is used to store the value of its D input under control of its *clock* input. Examples of logic blocks in commercial FPGAs are presented in Appendix E.

For a logic circuit to be realized in an FPGA, each logic function in the circuit must be small enough to fit within a single logic block. In practice, a user's circuit is automatically

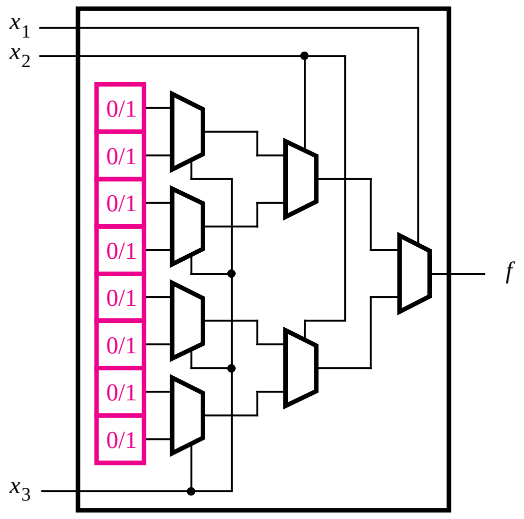


Figure 3.37 A three-input LUT.

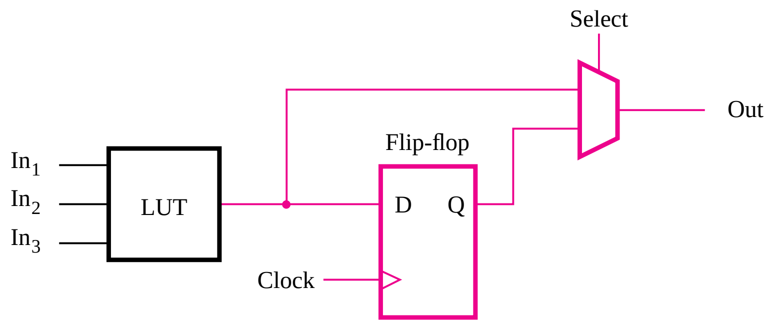


Figure 3.38 Inclusion of a flip-flop in an FPGA logic block.

translated into the required form by using CAD tools (see section 4.12). When a circuit is implemented in an FPGA, the logic blocks are programmed to realize the necessary functions and the routing channels are programmed to make the required interconnections between logic blocks. FPGAs are configured by using the ISP method, which we explained in section 3.6.4. The storage cells in the LUTs in an FPGA are *volatile*, which means that they lose their stored contents whenever the power supply for the chip is turned off. Hence the FPGA has to be programmed every time power is applied. Often a small memory chip that holds its data permanently, called a *programmable read-only memory (PROM)*, is included on the circuit board that houses the FPGA. The storage cells in the FPGA are loaded automatically from the PROM when power is applied to the chips.

A small FPGA that has been programmed to implement a circuit is depicted in Figure 3.39. The FPGA has two-input LUTs, and there are four wires in each routing channel. The figure shows the programmed states of both the logic blocks and wiring switches in a section of the FPGA. Programmable wiring switches are indicated by an X. Each switch shown in blue is turned on and makes a connection between a horizontal and vertical wire. The switches shown in black are turned off. We describe how the switches are implemented by using transistors in section 3.10.1. The truth tables programmed into the logic blocks in the top row of the FPGA correspond to the functions $f_1 = x_1x_2$ and $f_2 = \overline{x}_2x_3$. The logic block in the bottom right of the figure is programmed to produce $f = f_1 + f_2 = x_1x_2 + \overline{x}_2x_3$.

3.6.6 Using CAD Tools to Implement Circuits in CPLDs and FPGAs

In section 2.8 we suggested that the reader should work through Tutorial 1, in Appendix B, to gain some experience using real CAD tools. Tutorial 1 covers the steps of design entry and functional simulation. Now that we have discussed some of the details of the implementation of circuits in chips, the reader may wish to experiment further with the CAD tools. In Tutorial 2, section C.3, we illustrate how to download a circuit from a computer into a CPLD or FPGA.

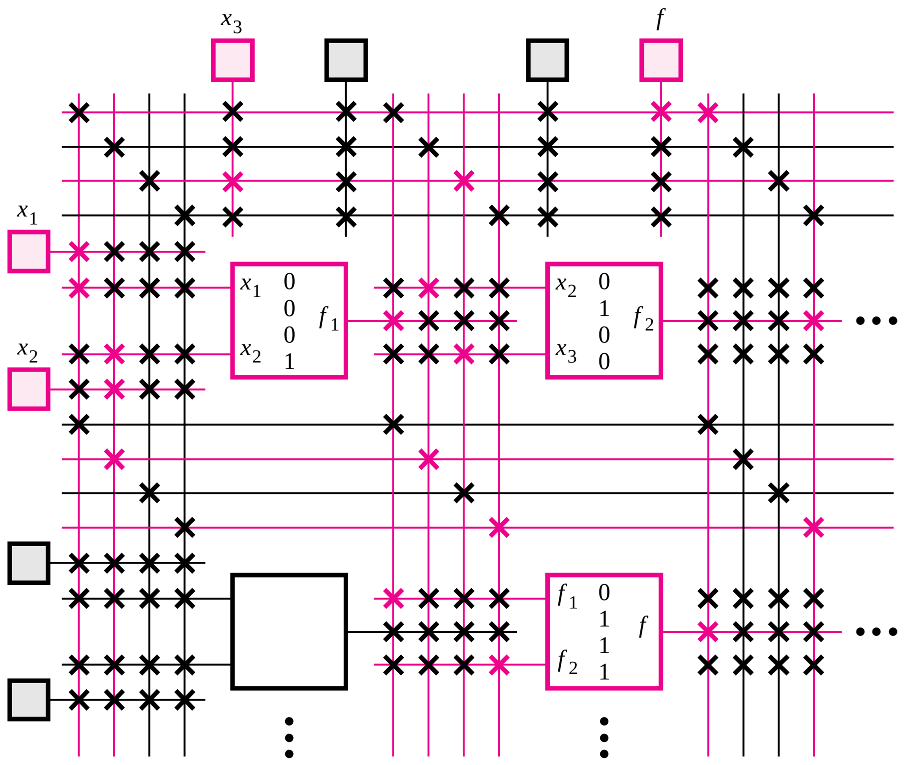


Figure 3.39 A section of a programmed FPGA.

3.7 Custom Chips, Standard Cells, and Gate Arrays

The key factor that limits the size of a circuit that can be accommodated in a PLD is the existence of programmable switches. Although these switches provide the benefit of user programmability, they consume a significant amount of space on the chip. They also result in a reduction in the speed of operation of circuits. In this section we will introduce some integrated circuit technologies that do not contain programmable switches.

Chips that provide the largest number of logic gates and the highest speed are so-called *custom chips*. Whereas a PLD is prefabricated, containing logic gates and programmable switches that are programmed to realize a user's circuit, a custom chip is created from scratch. The designer of a custom chip has complete flexibility to decide the size of the chip, the number of transistors the chip contains, the placement of each transistor on the chip, and the way the transistors are connected together. The process of defining exactly where on the chip each transistor and wire is situated is called *chip layout*. For a custom chip the designer may create any layout that is desired. Because it may contain more than a million transistors, a custom chip requires a large amount of design effort and therefore is expensive. Consequently, custom chips are used only when a very large number of transistors is needed and high-speed performance is important. Also, the product being designed must be expected to sell in sufficient quantities to recoup the expense. Two examples of products that are usually realized with custom chips are microprocessors and memory chips.

Some of the design effort incurred for a custom chip can be avoided by using a technology known as *standard cells*. Chips made using this technology are often called *application-specific integrated circuits (ASICs)*. This technology is illustrated in Figure 3.40, which depicts a small portion of a chip. The rows of logic gates may be connected by wires that are created in the *routing channels* between the rows of gates. In general, many types of logic gates may be used in such a chip. The available gates are prebuilt and are stored in a library that can be accessed by the designer. In Figure 3.40 the wires are drawn in two colors. This scheme is used because metal wires can be created on integrated circuits in multiple *layers*, which makes it possible for two wires to cross one another without creating a short circuit. The blue wires represent one layer of metal wires, and the black wires are a different layer. Each blue square represents a hard-wired connection (called a *via*) between a wire on one layer and a wire on the other layer. In current technology it is possible to have eight or more layers of metal wiring. Some of the metal layers can be placed on top of the transistors in the logic gates, resulting in a more efficient chip layout.

Like a custom chip, a standard-cell chip is created from scratch according to a user's specifications. The circuitry shown in Figure 3.40 implements the two logic functions that we realized in a PLA in Figure 3.26, namely, $f_1 = x_1x_2 + x_1\overline{x}_3 + \overline{x}_1\overline{x}_2x_3$ and $f_2 = x_1x_2 + \overline{x}_1\overline{x}_2x_3 + x_1x_3$. Because of the expense involved, a standard-cell chip would never be created for a small circuit such as this one, and thus the figure shows only a portion of a much larger chip. The layout of individual gates (standard cells) is predesigned and fixed. The chip layout can be created automatically by CAD tools because of the regular arrangement of the logic gates (cells) in rows. A typical chip has many long rows of logic gates with a large number of wires between each pair of rows. The I/O blocks around the periphery connect to the pins of the chip package, which is usually a QFP, PGA, or BGA package.

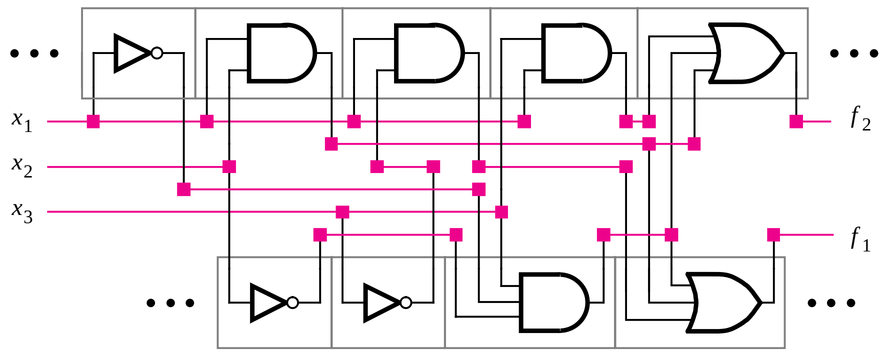


Figure 3.40 A section of two rows in a standard-cell chip.

Another technology, similar to standard cells, is the *gate-array* technology. In a gate array parts of the chip are prefabricated, and other parts are custom fabricated for a particular user's circuit. This concept exploits the fact that integrated circuits are fabricated in a sequence of steps, some steps to create transistors and other steps to create wires to connect the transistors together. In gate-array technology, the manufacturer performs most of the fabrication steps, typically those involved in the creation of the transistors, without considering the requirements of a user's circuit. This process results in a silicon wafer (see Figure 1.1) of partially finished chips, called the gate-array *template*. Later the template is modified, usually by fabricating wires that connect the transistors together, to create a user's circuit in each finished chip. The gate-array approach provides cost savings in comparison to the custom-chip approach because the gate-array manufacturer can amortize the cost of chip fabrication over a large number of template wafers, all of which are identical. Many variants of gate-array technology exist. Some have relatively large logic cells, while others are configurable at the level of a single transistor.

An example of a gate-array template is given in Figure 3.41. The gate array contains a two-dimensional array of logic cells. The chip's general structure is similar to a standard-cell chip except that in the gate array all logic cells are identical. Although the types of logic cells used in gate arrays vary, one common example is a two- or three-input NAND gate. In some gate arrays empty spaces exist between the rows of logic cells to accommodate the wires that will be added later to connect the logic cells together. However, most gate arrays do not have spaces between rows of logic cells, and the interconnection wires are fabricated on top of the logic cells. This design is possible because, as discussed for Figure 3.40, metal wires can be created on a chip in multiple layers. This technology is known

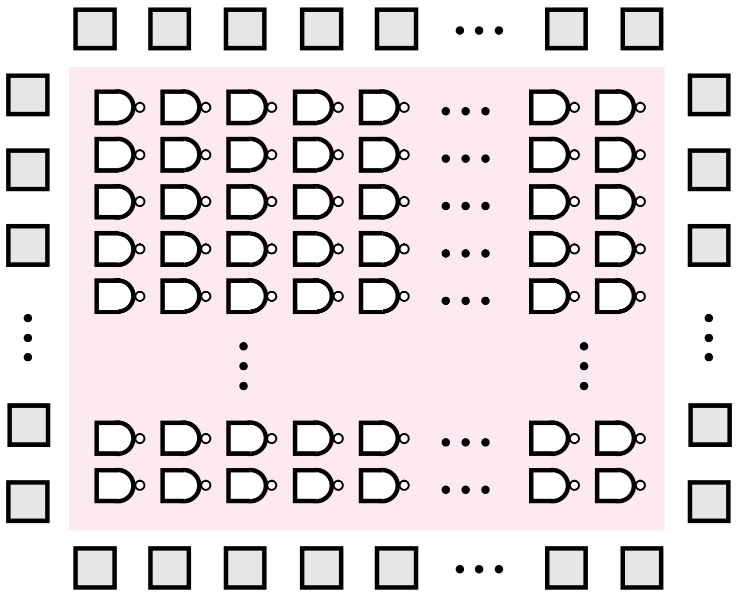


Figure 3.41 A sea-of-gates gate array.

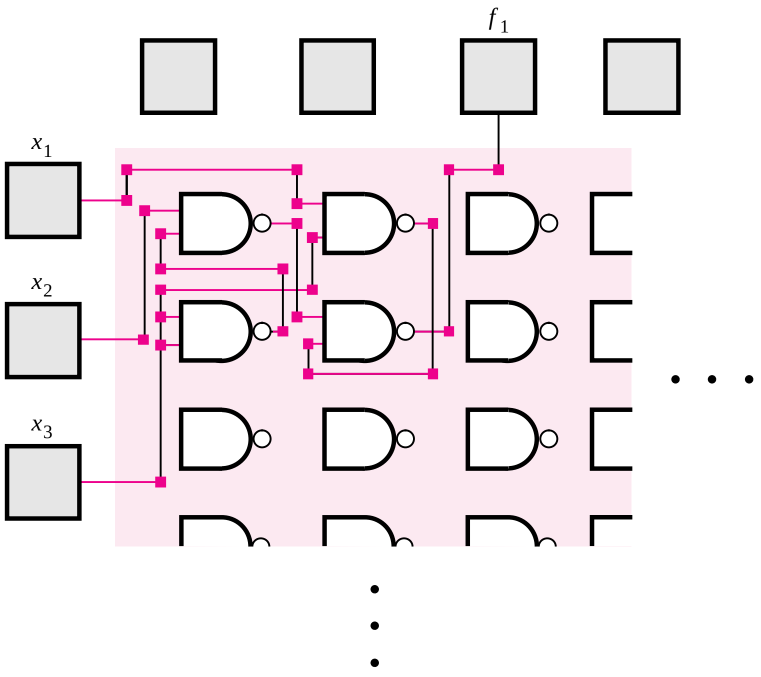


Figure 3.42 The logic function $f_1 = x_2\overline{x}_3 + x_1x_3$ in the gate array of Figure 3.41.

as the *sea-of-gates* technology. Figure 3.42 depicts a small section of a gate array that has been customized to implement the logic function $f = x_2\overline{x}_3 + x_1x_3$. It is easy to verify that this circuit with only NAND gates is equivalent to the AND-OR form of the circuit. We will describe a process for deriving this equivalence in section 4.6.

3.8 Practical Aspects

So far in this chapter, we have described the basic aspects of logic gate circuits and given examples of commercial chips. In this section we provide more detailed information on several aspects of digital circuits. We describe how transistors are fabricated in silicon and give a detailed explanation of how transistors operate. We discuss the robustness of logic circuits and discuss the important issues of signal propagation delays and power dissipation in logic gates.

3.8.1 MOSFET Fabrication and Behavior

To understand the operation of NMOS and PMOS transistors, we need to consider how they are built in an integrated circuit. Integrated circuits are fabricated on silicon wafers.