

1.9 Printed-Circuit Boards

printed-circuit board (PCB)

printed-wiring board (PWB)

PCB traces

mil

fine-line

surface-mount technology (SMT)

multichip module (MCM)

An IC is normally mounted on a *printed-circuit board (PCB)* [or *printed-wiring board (PWB)*] that connects it to other ICs in a system. The multilayer PCBs used in typical digital systems have copper wiring etched on multiple, thin layers of fiberglass that are laminated into a single board about 1/16 inch thick.

Individual wire connections, or *PCB traces* are usually quite narrow, 10 to 25 mils in typical PCBs. (A *mil* is one-thousandth of an inch.) In *fine-line* PCB technology, the traces are extremely narrow, as little as 4 mils wide with 4-mil spacing between adjacent traces. Thus, up to 125 connections may be routed in a one-inch-wide band on a single layer of the PCB. If higher connection density is needed, then more layers are used.

Most of the components in modern PCBs use *surface-mount technology (SMT)*. Instead of having the long pins of DIP packages that poke through the board and are soldered to the underside, the leads of SMT IC packages are bent to make flat contact with the top surface of the PCB. Before such components are mounted on the PCB, a special “solder paste” is applied to contact pads on the PCB using a stencil whose hole pattern matches the contact pads to be soldered. Then the SMT components are placed (by hand or by machine) on the pads, where they are held in place by the solder paste (or in some cases, by glue). Finally, the entire assembly is passed through an oven to melt the solder paste, which then solidifies when cooled.

Surface-mount component technology, coupled with fine-line PCB technology, allows extremely dense packing of integrated circuits and other components on a PCB. This dense packing does more than save space. For very high-speed circuits, dense packing goes a long way toward minimizing adverse analog phenomena, including transmission-line effects and speed-of-light limitations.

To satisfy the most stringent requirements for speed and density, *multichip modules (MCMs)* have been developed. In this technology, IC dice are not mounted in individual plastic or ceramic packages. Instead, the IC dice for a high-speed subsystem (say, a processor and its cache memory) are bonded directly to a substrate that contains the required interconnections on multiple layers. The MCM is hermetically sealed and has its own external pins for power, ground, and just those signals that are required by the system that contains it.

1.10 Digital-Design Levels

Digital design can be carried out at several different levels of representation and abstraction. Although you may learn and practice design at a particular level, from time to time you’ll need to go up or down a level or two to get the job done. Also, the industry itself and most designers have been steadily moving to higher levels of abstraction as circuit density and functionality have increased.

The lowest level of digital design is device physics and IC manufacturing processes. This is the level that is primarily responsible for the breathtaking advances in IC speed and density that have occurred over the past decades. The effects of these advances are summarized in *Moore's Law*, first stated by Intel founder Gordon Moore in 1965: that the number of transistors per square inch in an IC doubles every year. In recent years, the rate of advance has slowed down to doubling about every 18 months, but it is important to note that with each doubling of density has also come a doubling of speed.

Moore's Law

This book does not reach down to the level of device physics and IC processes, but you need to recognize the importance of that level. Being aware of likely technology advances and other changes is important in system and product planning. For example, decreases in chip geometries have recently forced a move to lower logic-power-supply voltages, causing major changes in the way designers plan and specify modular systems and upgrades.

In this book, we jump into digital design at the transistor level and go all the way up to the level of logic design using HDLs. We stop short of the next level, which includes computer design and overall system design. The "center" of our discussion is at the level of functional building blocks.

To get a preview of the levels of design that we'll cover, consider a simple design example. Suppose you are to build a "multiplexer" with two data input bits, A and B, a control input bit S, and an output bit Z. Depending on the value of S, 0 or 1, the circuit is to transfer the value of either A or B to the output Z. This idea is illustrated in the "switch model" of Figure 1-7. Let us consider the design of this function at several different levels.

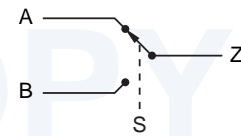


Figure 1-7
Switch model for multiplexer function.

Although logic design is usually carried out at higher level, for some functions it is advantageous to optimize them by designing at the transistor level. The multiplexer is such a function. Figure 1-8 shows how the multiplexer can be designed in "CMOS" technology using specialized transistor circuit structures

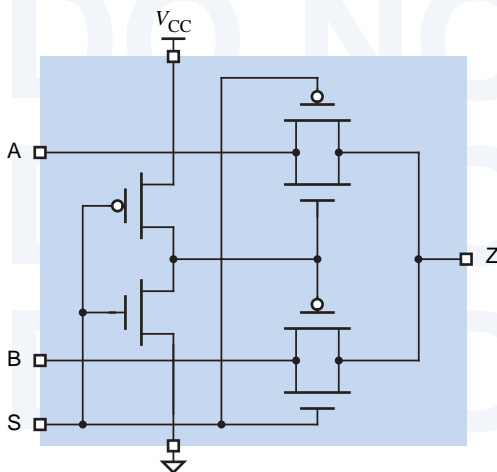


Figure 1-8
Multiplexer design using CMOS transmission gates.

Table 1-1
Truth table for the
multiplexer function.

S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

called “transmission gates,” discussed in Section 3.7.1. Using this approach, the multiplexer can be built with just six transistors. Any of the other approaches that we describe require at least 14 transistors.

In the traditional study of logic design, we would use a “truth table” to describe the multiplexer’s logic function. A truth table list all possible combinations of input values and the corresponding output values for the function. Since the multiplexer has three inputs, it has 2^3 or 8 possible input combinations, as shown in the truth table in Table 1-1.

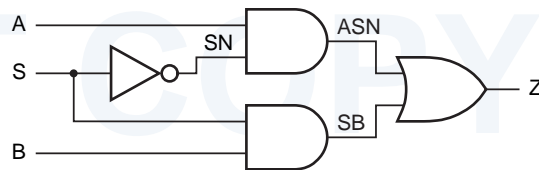
Once we have a truth table, traditional logic design methods, described in Section 4.3, use Boolean algebra and well understood minimization algorithms to derive an “optimal” two-level AND-OR equation from the truth table. For the multiplexer truth table, we would derive the following equation:

$$Z = S' \cdot A + S \cdot B$$

This equation is read “Z equals not S and A or S and B.” Going one step further, we can convert the equation into a corresponding set of logic gates that perform the specified logic function, as shown in Figure 1-9. This circuit requires 14 transistors if we use standard CMOS technology for the four gates shown.

A multiplexer is a very commonly used function, and most digital logic technologies provide predefined multiplexer building blocks. For example, the 74x157 is an MSI chip that performs multiplexing on two 4-bit inputs simultaneously. Figure 1-10 is a logic diagram that shows how we can hook up just one bit of this 4-bit building block to solve the problem at hand. The numbers in color are pin numbers of a 16-pin DIP package containing the device.

Figure 1-9
Gate-level logic diagram
for multiplexer function.



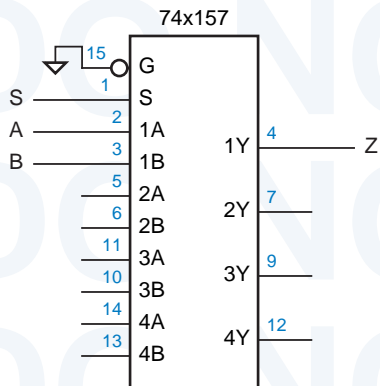


Figure 1-10
Logic diagram for a multiplexer using an MSI building block.

We can also realize the multiplexer function as part of a programmable logic device. Languages like ABEL allow us to specify outputs using Boolean equations similar to the one on the previous page, but it's usually more convenient to use “higher-level” language elements. For example, Table 1-2 is an ABEL program for the multiplexer function. The first three lines define the name of the program module and specify the type of PLD in which the function will be realized. The next two lines specify the device pin numbers for inputs and output. The “WHEN” statement specifies the actual logic function in a way that's very easy to understand, even though we haven't covered ABEL yet.

An even higher level language, VHDL, can be used to specify the multiplexer function in a way that is very flexible and hierarchical. Table 1-3 is an example VHDL program for the multiplexer. The first two lines specify a standard library and set of definitions to use in the design. The next four lines specify only the inputs and outputs of the function, and purposely hide any details about the way the function is realized internally. The “architecture” section of the program specifies the function's behavior. VHDL syntax takes a little getting used to, but the single “when” statement says basically the same thing that the ABEL version did. A VHDL “synthesis tool” can start with this

```

module chap1mux
title 'Two-input multiplexer example'
CHAP1MUX device 'P16V8'

A, B, S    pin 1, 2, 3;
Z          pin 13 istype 'com';

equations

WHEN S == 0 THEN Z = A; ELSE Z = B;

end chap1mux

```

Table 1-2
ABEL program for the multiplexer.

Table 1-3
VHDL program for
the multiplexer.

```

library IEEE;
use IEEE.std_logic_1164.all;

entity Vchap1mux is
    port ( A, B, S: in  STD_LOGIC;
          Z:      out STD_LOGIC );
end Vchap1mux;

architecture Vchap1mux_arch of Vchap1mux is
begin
    Z <= A when S = '0' else B;
end Vchap1mux_arch;

```

behavioral description and produce a circuit that has this behavior in a specified target digital-logic technology.

By explicitly enforcing a separation of input/output definitions (“entity”) and internal realization (“architecture”), VHDL makes it easy for designers to define alternate realizations of functions without having to make changes elsewhere in the design hierarchy. For example, a designer could specify an alternate, structural architecture for the multiplexer as shown in Table 1-4. This architecture is basically a text equivalent of the logic diagram in Figure 1-9.

Going one step further, VHDL is powerful enough that we could actually define operations that model functional behavioral at the transistor level (though we won’t explore such capabilities in this book). Thus, we could come full circle by writing a VHDL program that specifies a transistor-level realization of the multiplexer equivalent to Figure 1-8.

Table 1-4
“Structural” VHDL
program for the
multiplexer.

```

architecture Vchap1mux_gate_arch of Vchap1mux is
signal SN, ASN, SB: STD_LOGIC;
begin
    U1: INV (S, SN);
    U2: AND2 (A, SN, ASN);
    U3: AND2 (S, B, SB);
    U4: OR2 (ASN, SB, Z);
end Vchap1mux_gate_arch;

```

1.11 The Name of the Game

board-level design

Given the functional and performance requirements for a digital system, the name of the game in practical digital design is to minimize cost. For *board-level designs*—systems that are packaged on a single PCB—this usually means minimizing the number of IC packages. If too many ICs are required, they won’t all fit on the PCB. “Well, just use a bigger PCB,” you say. Unfortunately, PCB sizes are usually constrained by factors such as pre-existing standards (e.g., add-in