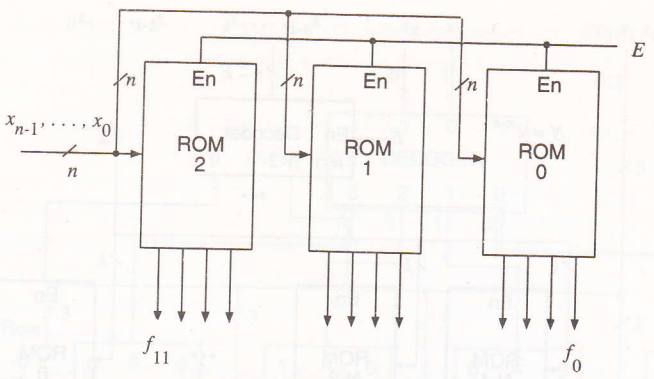

Figure 12.10 ROM-based implementation of large number of switching functions.

module is not more expensive than another standard module with the same number of pins.

2. The cost of an integrated circuit is very dependent on the number of chips of the same kind produced. Programmable modules are used in many applications, so they can be mass produced and customized for the particular application at the last stage of production or in the field.
3. The cost of an integrated circuit depends significantly on the cost of designing the chip. Programmable modules have a relatively low design cost due to their regularity.
4. The cost of a system depends heavily on the number of chips used. Due to the complex functions they can realize, programmable modules reduce considerably the number of chips required.
5. A field-programmable module allows for easier modification of the functions it implements than other approaches.

In VLSI designs, the use of programmable modules results in a low design cost because of their regularity and because they can be easily adapted from modules in a library.

On the other hand, the disadvantage of using programmable modules is that they might produce a slower network and require a larger silicon area than other approaches.

The main differences among PLAs and ROMs are as follows.

1. The PLA results in a more compact implementation because, instead of storing the whole table of the functions, it implements reduced sum of products (product of sums) expressions. This difference is due to having a decoder for ROM and an AND array for PLA. If the number of products (sums) required is significantly smaller than the maximum 2^n, then this reduction in size can be large.
2. The PLA implementation is limited to a set of functions that can be represented by a set of sums of products (products of sums) with no more than r products (sums). No such a limitation exists for the ROM.
3. PLAs are more difficult to program than ROMs. This results from the fact that, to achieve the compactness that is possible with a PLA, it is necessary to design a two-level AND-OR (OR-AND) network that has a small number of AND (OR) gates.

12.5 FIELD-PROGRAMMABLE GATE ARRAYS

A **field-programmable gate array** (FPGA) is a VLSI module that can be programmed to implement a digital system consisting of tens of thousands of gates. In contrast to PLA/PSA

modules that implement two-level combinational and canonical sequential networks, FPGAs allow the realization of multilevel networks and complex systems on a single chip.

A FPGA module consists of an array of three kinds of programmable (configurable) elements:

- **logic blocks,** either combinational and/or sequential;
- **interconnection points** (switches); and
- **input/output blocks.**

In addition, there are wires grouped in horizontal and vertical **channels.** Figure 12.11 illustrates the organization of an FPGA module at the chip level.

Each logic module can be programmed to implement several switching functions of a few variables, and to control the use of the flip-flops in the module. Logic modules used in FPGAs are implemented either as a look-up table (LUT) or a multiplexer. A LUT of k inputs can be programmed as a truth table of a switching function of up to k variables. A 2^n-input multiplexer, as discussed in Chapter 9, can be used to implement any switching function of n variables. The interconnect points and input/output blocks can be programmed to achieve the desired connections.

There are three basic approaches in providing programmability of FPGAs:

- On-chip control latches (memory cells) that are set with bit patterns to define the chip configuration. This type is called SRAM-FPGA because the set of control latches can be considered as a static random access memory, as defined in Chapter 14. These FPGAs are volatile, that is, the programming information is not preserved after the chip is powered down.
- Antifuse-programmed devices that are programmed electrically to provide connections that define the chip configuration. The programming is done by permanently closing some of the antifuse switches. Thus, unlike SRAM-FPGAs, these devices cannot be

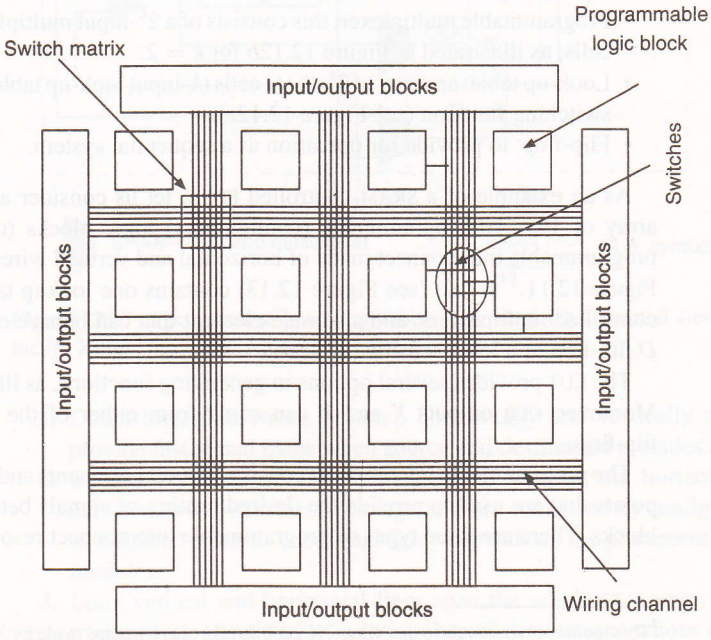

Figure 12.11 Organization of an FPGA chip.

reprogrammed (like mask-programmed ROMs and PLAs). However, these nonvolatile FPGAs are faster than the SRAM-type devices. One important advantage of antifuses is their very small size, allowing a large number of interconnections on a chip.

- Using several electrically programmable devices (EPROMs and EEPROMs) and a shared interconnect mechanism on a single chip. In contrast to SRAM-based FPGAs, EPROM and EEPROM technologies do not require external permanent memory to preserve chip configuration. On the other hand, they require more complex chip fabrication process and use larger cells.

We discuss only the SRAM-type of FPGAs, due to their high reprogrammability.

SRAM-FPGAs

In SRAM-FPGAs, each programmable element is controlled by a memory cell. Memory cells are loaded during the programming phase with binary values that represent the desired values of control signals that define the chip configuration. These signals remain constant until another configuration is loaded. That is, the bit pattern stored in the on-chip volatile reconfiguration memory defines the structure and behavior of the chip—the chip "personality." This approach, although requiring an external nonvolatile memory for storage of configuration patterns, offers dynamic flexibility. That is, the same FPGA module can be used for different purposes. For example, FPGA modules on a board can be initially configured to perform testing of components on the board; after testing, the FPGAs can be reprogrammed to perform functions as required by the board application. Loading a configuration pattern into an FPGA typically takes several milliseconds.

The following are typical elements of a SRAM-FPGA:

- Programmable switch: a SRAM cell attached to the gate of a transistor acts as a switch (see Figure 12.12*a*) that is used to provide connections between logic/storage blocks' inputs/outputs and interconnecting lines.
- Programmable multiplexer: this consists of a 2^k-input multiplexer controlled by k SRAM cells, as illustrated in Figure 12.12*b* for $k = 2$.
- Look-up table: an array of 2^k SRAM cells (k-input look-up table) implements a k-variable switching function (see Figure 12.12*c*).
- Flip-flop: to provide for operation as a sequential system.

As an example of a SRAM-controlled FPGA, let us consider a module consisting of an array of 10×10 programmable (configurable) logic blocks (CLBs), connected using a programmable interconnect made of horizontal and vertical wires organized as depicted in Figure 12.11.[1] A CLB (see Figure 12.13) contains one lookup table (LUT), several SRAM-controlled multiplexers, and a storage element that can behave either as an edge-sensitive D flip-flop or a level-sensitive D latch.

The LUT provides several options in generating functions, as illustrated in Figure 12.14.[2] Moreover, CLB outputs X and Y can come from either of the LUT outputs or from the flip-flop.

The programmable interconnect consists of metal segments and programmable switching points that are used to provide the desired routing of signals between CLBs as well as I/O blocks. There are three types of programmable interconnect resources (see Figure 12.15):

---

[1]This module corresponds to the Xilinx XC2000 family of FPGA chips made by Xilinx, Inc., San Jose, CA.

[2]The configurations in Figure 12.14 are not obtained directly from Figure 12.13.

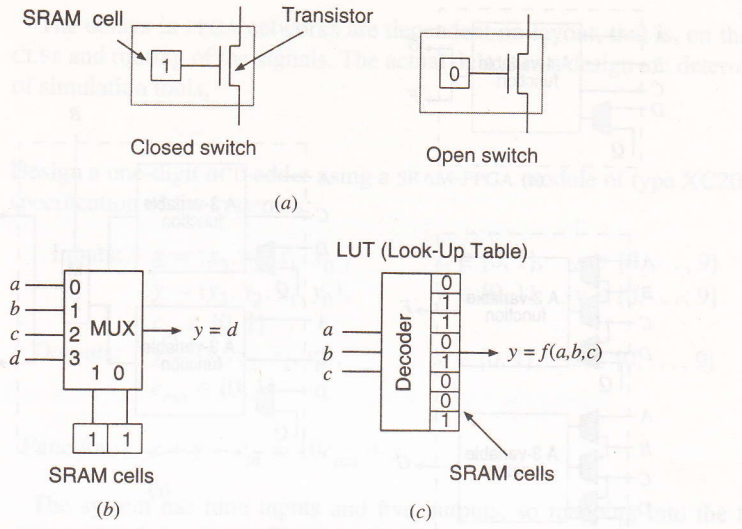

Figure 12.12 SRAM-FPGA programmable components. (*a*) Switch. (*b*) 4-input multiplexer. (*c*) Look-up table (LUT).

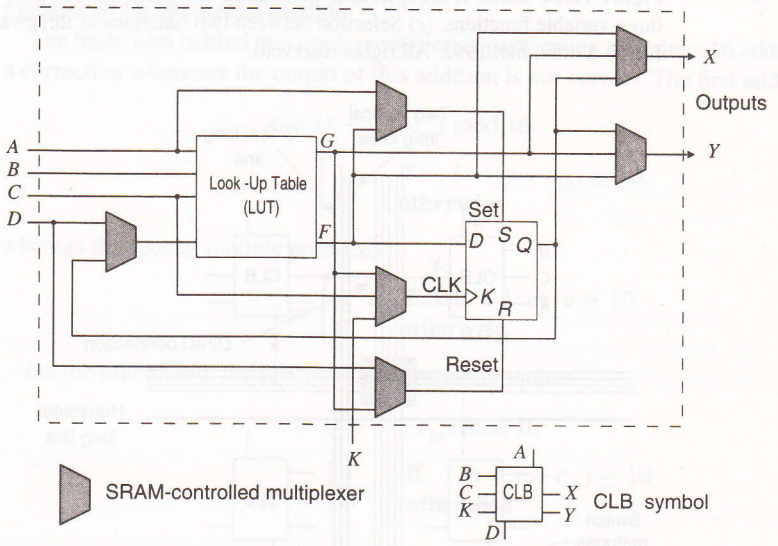

Figure 12.13 A programmable (configurable) logic block (CLB), partial view (courtesy of Xilinx, Inc.© Xilinx, Inc. 1992. All rights reserved).

1. Direct interconnections between horizontally and vertically adjacent CLBs. These provide fast signal paths when source and destination modules are adjacent.
2. General-purpose interconnects consist of vertical and horizontal wiring segments between switch matrices. The segments are connected by configuring switch matrices in a desired pattern. The signal delay depends on the placement of source/destination modules.
3. Long vertical and horizontal lines span the whole CLB array, providing means for transmitting signals to a large number of destinations whose delay is critical (e.g., clock signal).

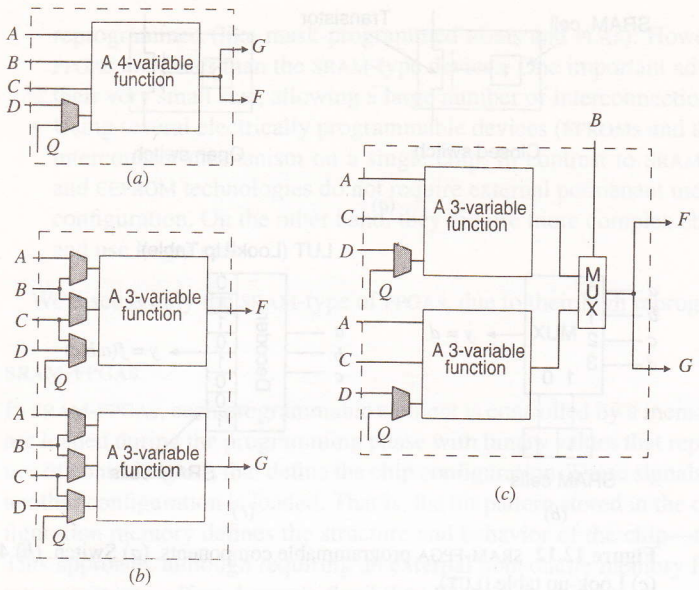

Figure 12.14 SRAM-FPGA options in generating functions. (*a*) One four-variable function. (*b*) Two three-variable functions. (*c*) Selection between two functions of three variables (courtesy of Xilinx, Inc.© Xilinx, Inc. 1992. All rights reserved).

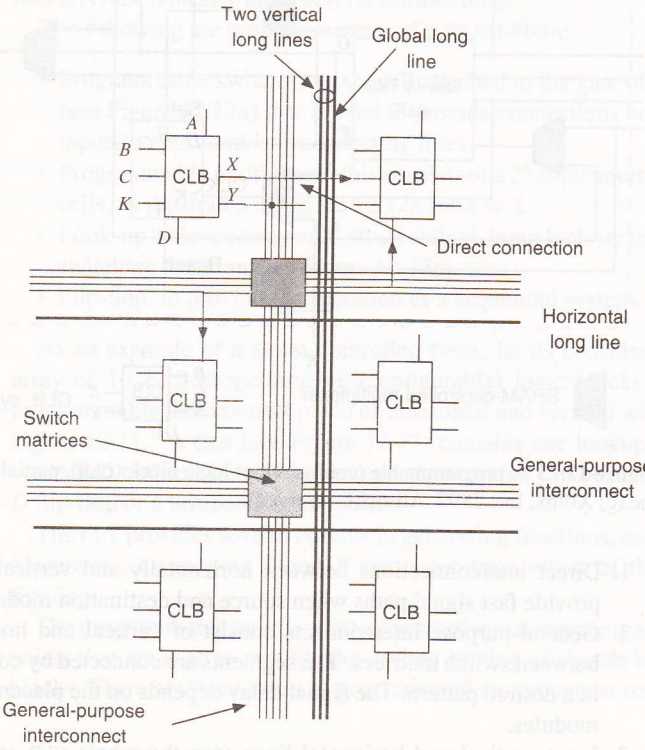

Figure 12.15 Programmable interconnect (courtesy of Xilinx, Inc.© Xilinx, Inc. 1992. All rights reserved).

The delays in FPGA networks are dependent on layout, that is, on the placement of the CLBs and routing of the signals. The actual delays in a design are determined with the help of simulation tools.

EXAMPLE 12.5

Design a one-digit BCD adder using a SRAM-FPGA module of type XC2000. The high-level specification of the system is:

Inputs: $\underline{x} = (x_3, x_2, x_1, x_0)$, $x_j \in \{0, 1\}$, $x \in \{0, \ldots, 9\}$
$\underline{y} = (y_3, y_2, y_1, y_0)$, $y_j \in \{0, 1\}$, $y \in \{0, \ldots, 9\}$
$c_{in} \in \{0, 1\}$

Outputs: $\underline{s} = (s_3, s_2, s_1, s_0)$, $s_j \in \{0, 1\}$, $s \in \{0, \ldots, 9\}$
$c_{out} \in \{0, 1\}$

Function: $x + y + c_{in} = 10c_{out} + s$

The system has nine inputs and five outputs, so mapping into the FPGA requires decomposition into several CLBs. Many decompositions are possible. We rely on a scheme that uses two binary adders and a special module, as shown in Figure 12.16. Note that the second adder is actually a four-bit adder that adds either 0 or 6, depending on the value of t generated by the special module.

The basic idea behind this implementation is performing a modulo-16 addition, and then a correction whenever the output of this addition is not correct. The first adder produces

$$v = (x + y + c_{in}) \bmod 16$$

$$u = \begin{cases} 1 & \textbf{if} \quad (x + y + c_{in}) \geq 16 \\ 0 & \textbf{otherwise} \end{cases}$$

whereas the special module produces

$$t = \begin{cases} 1 & \textbf{if} \quad u = 1 \ \textbf{or} \ v \geq 10 \\ 0 & \textbf{otherwise} \end{cases}$$

On the other hand, the function of the system requires

$$s = (x + y + c_{in}) \bmod 10$$

$$c_{out} = \begin{cases} 1 & \textbf{if} \quad (x + y + c_{in}) \geq 10 \\ 0 & \textbf{otherwise} \end{cases}$$

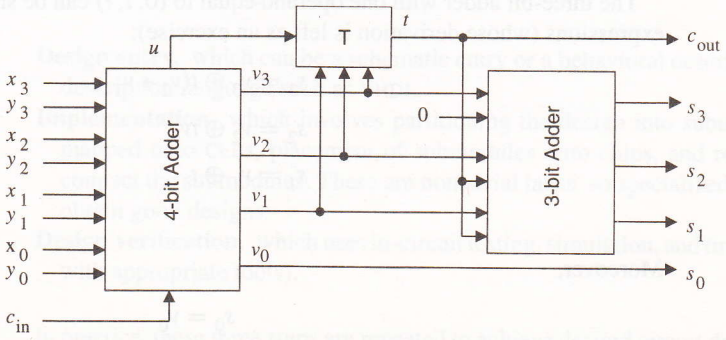

Figure 12.16 Implementation of a BCD adder module.

riable function. (*b*) Two
bles (courtesy of Xilinx,

To obtain these outputs we consider three cases:

Case 1. If $(x + y + c_{in}) \leq 9$ then

$$s = v$$
$$c_{out} = 0$$

so the second adder should add 0 and $c_{out} = 0$. This is accomplished because $t = 0$.

Case 2. If $10 \leq (x + y + c_{in}) \leq 15$, the first adder produces

$$v = x + y + c_{in}$$
$$u = 0$$

Getting the correct s requires subtracting 10 from v and making $c_{out} = 1$. That is,

$$\begin{aligned} s &= v - 10 \\ &= x + y + c_{in} + 6 - 16 \\ &= (x + y + c_{in} + 6) \bmod 16 \end{aligned}$$
$$c_{out} = 1$$

which is achieved by the second adder because $t = 1$.

Case 3. If $(x + y + c_{in}) \geq 16$, the first adder produces

$$v = (x + y + c_{in} - 16)$$
$$u = 1$$

Then,

$$\begin{aligned} s &= x + y + c_{in} - 10 \\ &= v + 6 \end{aligned}$$
$$c_{out} = 1$$

The second adder should again add 6 and make $c_{out} = 1$, which happens because $t = 1$.

The condition $u = 1$ or $v \geq 10$ corresponds to the switching expression

$$t = u + v_3 v_2 + v_3 v_1$$

The three-bit adder with one operand equal to $(0, t, t)$ can be simplified to the following expressions (whose derivation is left as an exercise):

$$s_3 = v_3 \oplus t(v_2 + v_1)$$
$$s_2 = v_2 \oplus t v_1'$$
$$s_1 = v_1 \oplus t$$

Moreover,

$$s_0 = v_0$$
$$c_{out} = t$$

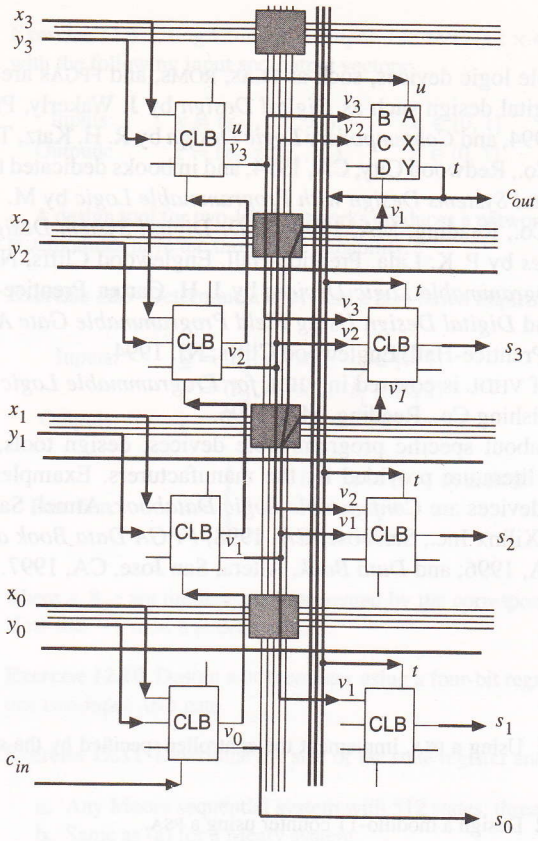

Figure 12.17 FPGA implementation of BCD adder module.

We now map this design onto the FPGA by implementing the four-bit adder as a carry-ripple adder (first column in Figure 12.17). In the second column, the upper module implements t, whereas the next three modules implement the simplified three-bit adder. ■

The design of a system based on FPGAs involves intensive use of CAD tools and module libraries. The basic design steps are:

Design entry, which can be a schematic entry or a behavioral description in a hardware description language such as VHDL.

Implementation, which involves partitioning the design into submodules that can be mapped onto CLBs, placement of submodules onto chips, and routing of signals to connect the submodules. These are nontrivial tasks, so specialized tools are needed to obtain good designs.

Design verification, which uses in-circuit testing, simulation, and timing analysis (again, with appropriate tools).

In practice, these three steps are repeated to achieve desired circuit delay and complexity features.