CHAPTER 13

PROGRAMMABLE LOGIC DEVICE ARCHITECTURES*[†]

OUTLINE

- 13-1 Digital Systems Family Tree
- 13-2 Fundamentals of PLD Circuitry
- 13-3 PLD Architectures
- 13-4 The GAL 16V8 (Generic Array Logic)
- 13-5 The Altera EPM7128S CPLD
- 13-6 The Altera FLEX10K Family
- 13-7 The Altera Cyclone Family

*Diagrams of the GAL 16V8 device presented in this chapter have been reproduced through the courtesy of Lattice Semiconductor Corporation, Hillsboro, Oregon.

[†]Diagrams of the MAX7000S and FLEX10K family devices presented in this chapter have been reproduced through the courtesy of Altera Corporation, San Jose, California.

OBJECTIVES

Upon completion of this chapter, you will be able to:

- Describe the different categories of digital system devices.
- Describe the different types of PLDs.
- Interpret PLD data book information.
- Define PLD terminology.
- Compare the different programming technologies used in PLDs.
- Compare the architectures of different types of PLDs.
- Compare the features of the Altera MAX7000S and FLEX10K families of PLDs.

INTRODUCTION

Throughout the chapters of this book you have been introduced to a wide variety of digital circuits. You now know how the building blocks of digital systems work and can combine them to solve a wide variety of digital problems. More complicated digital systems, such as microcomputers and digital signal processors, have also been briefly described. The defining difference between microcomputer/DSP systems and other digital systems is that the former follow a programmed sequence of instructions that the designer specifies. Many applications require faster response than a microcomputer/DSP architecture can accommodate and in these cases, a conventional digital circuit must be used. In today's rapidly advancing technology market, most conventional digital systems are not being implemented with standard logic device chips containing only simple gates or MSI-type functions. Instead, programmable logic devices, which contain the circuitry necessary to create logic functions, are being used to implement digital systems. These devices are not programmed with a list of instructions, like a computer or DSP. Instead, their internal hardware is configured by electronically connecting and disconnecting points in the circuit.

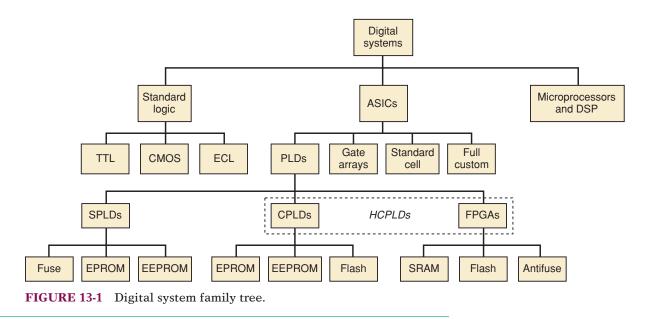
Why have PLDs taken over so much of the market? With programmable devices, the same functionality can be obtained with one IC rather than using several individual logic chips. This characteristic means less board space, less power required, greater reliability, less inventory, and overall lower cost in manufacturing.

In the previous chapters you have become familiar with the process of programming a PLD using either AHDL or VHDL. At the same time, you have learned about all the building blocks of digital systems. The PLD implementations of digital circuits up to this point have been presented as a "black box." We have not been concerned with what was going on inside the PLD to make it work. Now that you understand all the circuitry inside the black box, it is time to turn the lights on in there and look at how it works. This will allow you to make the best decisions when selecting and applying a PLD to solve a problem. This chapter will take a look at the various types of hardware available to design digital systems. We will then introduce you to the architectures of various families of PLDs.

13-1 DIGITAL SYSTEMS FAMILY TREE

While the major goal of this chapter is to investigate PLD architectures, it is also useful to look at the various hardware choices available to digital system designers because it should give us a little better perception of today's digital hardware alternatives. The desired circuit functionality can generally be achieved by using several different types of digital hardware. Throughout this book, we have described both standard logic devices as well as how programmable logic devices can be used to create the same functional blocks. Microcomputers and DSP systems can also often be applied with the necessary sequence of instructions (i.e., the application's program) to produce the desired circuit function. The design engineering decisions must take into account many factors, including the necessary speed of operation for the circuit, cost of manufacturing, system power consumption, system size, amount of time available to design the product, etc. In fact, most complex digital designs include a mix of different hardware categories. Many trade-offs between the various types of hardware have to be weighed to design a digital system.

A digital system family tree (see Figure 13-1) showing most of the hardware choices that are currently available can be useful in sorting out the many categories of digital devices. The graphical representation in the figure does not show all the details—some of the more complex device types have many additional subcategories, and older, obsolete device types have been omitted for clarity. The major digital system categories include standard logic, application-specific integrated circuits (ASICs) and microprocessor/ digital signal processing (DSP) devices.



The first category of **standard logic** devices refers to the basic functional digital components (gates, flip-flops, decoders, multiplexers, registers, counters, etc.) that are available as SSI and MSI chips. These devices have been used for many years (some more than 30 years) to design complex digital systems. An obvious drawback is that the system may literally consist of hundreds of such chips. These inexpensive devices can still be useful if our design is not very complex. As discussed in Chapter 8, there are three major families of standard logic devices: TTL, CMOS, and ECL. TTL is a mature technology consisting of numerous subfamilies that have been developed over many years of use. Very few new designs apply TTL logic, but many, many digital systems still contain TTL devices. CMOS is the most popular standard logic device family today, primarily due to its low power consumption. ECL technology, of course, is applied for higher-speed designs. Standard logic devices are still available to the digital designer, but if the application is very complex, a lot of SSI/MSI chips will be needed. That solution is not very attractive for our design needs today.

The microprocessor/digital signal processing (DSP) category is a much different approach to digital system design. These devices actually contain the various types of functional blocks that have been discussed throughout this text. With microcomputer/DSP systems, devices can be controlled electronically, and data can be manipulated by executing a program of instructions that has been written for the application. A great deal of flexibility can be achieved with microcomputer/DSP systems because all you have to do is change the program. The major downfall with this digital system category is speed. Using a hardware solution for your digital system design is always faster than a software solution.

The third major digital system category is called **application-specific integrated circuits (ASICs)**. This broad category represents the modern hardware design solution for digital systems. As the acronym implies, an integrated circuit is designed to implement a specific desired application. Four subcategories of ASIC devices are available to create digital systems: programmable logic devices, gate arrays, standard-cell, and full-custom.

Programmable logic devices (PLDs), sometimes referred to as fieldprogrammable logic devices (FPLDs), can be custom-configured to create any desired digital circuit, from simple logic gates to complex digital systems. Many examples of PLD designs have been given in earlier chapters. This ASIC choice for the designer is very different from the other three subcategories. With a relatively small capital investment, any company can purchase the necessary development software and hardware to program PLDs for their digital designs. On the other hand, to obtain a gate array, standard-cell or full-custom ASIC requires that most companies contract with an IC foundry to fabricate the desired IC chip. This option can be extremely expensive and usually requires that your company purchase a large volume of parts to be cost effective.

Gate arrays are ULSI circuits that offer hundreds of thousands of gates. The desired logic functions are created by the interconnections of these prefabricated gates. A custom-designed mask for the specific application determines the gate interconnections, much like the stored data in a mask-programmed ROM. For this reason, they are often referred to as mask-programmed gate arrays (MPGAs). Individually, these devices are less expensive than PLDs of comparable gate count, but the custom programming process by the chip manufacturer is very expensive and requires a great deal of lead time.

Standard-cell ASICs use predefined logic function building blocks called cells to create the desired digital system. The IC layout of each cell has been designed previously, and a library of available cells is stored in a computer database. The needed cells are laid out for the desired application, and the interconnections between the cells are determined. Design costs for

standard-cell ASICs are even higher than for MPGAs because all IC fabrication masks that define the components and interconnections must be custom designed. Greater lead time is also needed for the creation of the additional masks. Standard cells do have a significant advantage over gate arrays. The cell-based functions have been designed to be much smaller than equivalent functions in gate arrays, which allows for generally higher-speed operation and cheaper manufacturing costs.

Full-custom ASICs are considered the ultimate ASIC choice. As the name implies, all components (transistors, resistors, and capacitors) and the interconnections between them are custom-designed by the IC designer. This design effort requires a significant amount of time and expense, but it can result in ICs that can operate at the highest possible speed and require the smallest die (individual IC chip) area. Smaller IC die sizes allow for many more die to fit on a silicon wafer, which significantly lowers the manufacturing cost for each IC.

More on PLDs

This chapter is mainly about PLDs, so we will look a little further down that branch of the family tree. The development of PLD technology has advanced continuously since the first PLDs appeared more than 30 years ago. The early devices contained the equivalent of a few hundred gates, and now we have parts available that contain a few million gates. The old devices could handle a few inputs and a few outputs with limited logic capabilities. Now there are PLDs that can handle hundreds of inputs and outputs. Original devices could be programmed only once and, if the design changed, the old PLD would have to be removed from the circuit and a new one, programmed with the updated design, would have to be inserted in its place. With newer devices, the internal logic design can be changed on the fly, while the chip is still connected to a printed circuit board in an electronic system.

Generally, PLDs can be described as being one of three different types: simple programmable logic devices (SPLDs), complex programmable logic devices (CPLDs), or field programmable gate arrays (FPGAs). There are several manufacturers with many different families of PLD devices, so there are many variations in architecture. We will attempt to discuss the general characteristics for each of the types, but be forewarned: the differences are not always clear-cut. The distinction between CPLDs and FPGAs is often a little fuzzy, with the manufacturers constantly designing new, improved architectures and frequently muddying the waters for marketing purposes. Together, CPLDs and FPGAs are often referred to as high-capacity programmable logic devices (HCPLDs). The programming technologies for PLD devices are actually based on the various types of semiconductor memory. As new types of memory have been developed, the same technology has been applied to the creation of new types of PLD devices.

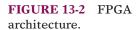
The amount of logic resources available is the major distinguishing feature between SPLDs and HCPLDs. Today, SPLDs are devices that typically contain the equivalent of 600 or fewer gates, while HCPLDs have thousands and hundreds of thousands of gates available. Internal programmable signal interconnect resources are much more limited with SPLDs. SPLDs are generally much less complicated and much cheaper than HCPLDs. Many small digital applications need only the resources of an SPLD. On the other hand, HCPLDs are capable of providing the circuit resources for complete complex digital systems, and larger, more sophisticated HCPLD devices are designed every year.

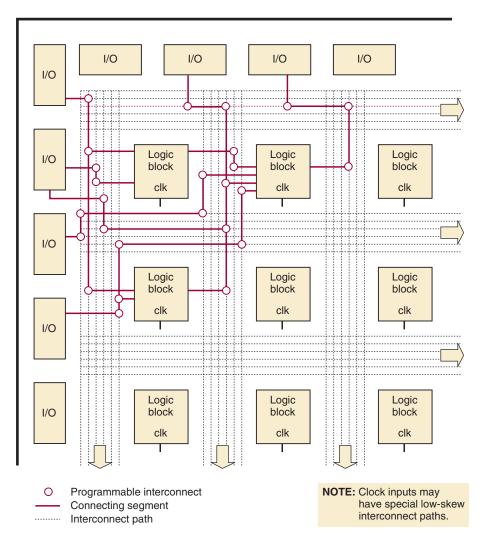
The SPLD classification includes the earliest PLD devices. The amount of logic resources contained in the early PLDs may be relatively small by today's

standards, but they represented a significant technological step in their ability to create easily a custom IC that can replace several standard logic devices. Over the years, numerous semiconductor advances have created different SPLD types. The first PLD type to gain the interest of circuit designers was programmed by literally burning open selected fuses in the programming matrix. The fuses that were left intact in these one-time programmable (OTP) devices provided the electrical connections for the AND/OR circuits to produce the desired functions. This logic device was based on the fuse links in PROM memory technology (see Section 12-7) and was most commonly referred to as a programmable logic array (PLA). PLDs didn't really gain widespread acceptance with digital designers until the late 1970s, when a device called a programmable array logic (PAL) was introduced. The programmable fuse links in a PAL are used to determine the input connections to a set of AND gates that are wired to fixed OR gates. With the development of the ultraviolet erasable PROM came the EPROM-based PLDs in the mid 1980s, followed soon by PLDs using electrically erasable (EEPROM) technology.

CPLDs are devices that typically combine an array of PAL-type devices on the same chip. The logic blocks themselves are programmable AND/fixed-OR logic circuits with fewer product terms available than most PAL devices. Each logic block (often called a macrocell) can typically handle many input variables, and the internal programmable logic signal routing resources tend to be very uniform throughout the chip, producing consistent signal delays. When more product terms are needed, gates may be shared between logic blocks, or several logic blocks can be combined to implement the expression. The flip-flop used to implement the register in the macrocell can often be configured for D, JK, T (toggle), or SR operation. Input and output pins for some CPLD architectures are associated with a specific macrocell, and typically additional macrocells are buried (that is, not connected to a pin). Other CPLD architectures may have independent I/O blocks with built-in registers that can be used to latch incoming or outgoing data. The programming technologies used in CPLD devices are all nonvolatile and include EPROM, EEPROM, and flash, with EEPROM being the most common. All three technologies are erasable and reprogrammable.

FPGAs also have a few fundamental characteristics that are shared. They typically consist of many relatively small and independent programmable logic modules that can be interconnected to create larger functions. Each module can usually handle only up to four or five input variables. Most FPGA logic modules utilize a look-up table (LUT) approach to create the desired logic functions. A look-up table functions just like a truth table in which the output can be programmed to create the desired combinational function by storing the appropriate 0 or 1 for each input combination. The programmable signal routing resources within the chip tend to be quite varied, with many different path lengths available. The signal delays produced for a design depend on the actual signal routing selected by the programming software. The logic modules also contain programmable registers. The logic modules are not associated with any I/O pin. Instead, each I/O pin is connected to a programmable input/output block that, in turn, is connected to the logic modules with selected routing lines. The I/O blocks can be configured to provide input, output, or bidirectional capability, and built-in registers can be used to latch incoming or outgoing data. A general architecture of FPGAs is shown in Figure 13-2. All of the logic blocks and input/output blocks can be programmed to implement almost any logic circuit. The programmable interconnections are accomplished via lines that run through the rows and columns in the channels between the logic blocks. Some FPGAs include large blocks of RAM memory; others do not.





The programming technologies used in FPGA devices include SRAM, flash, and antifuse, with SRAM being the most common. SRAM-based devices are volatile and therefore require the FPGA to be reconfigured (programmed) when it is powered-up. The programming information that defines how each logic block functions, which I/O blocks are inputs and outputs, and how the blocks are interconnected is stored in some type of external memory that is downloaded to the SRAM-based FPGA when power is applied. Antifuse devices are one-time programmable and are therefore nonvolatile. Antifuse memory technology is not currently used for memory devices but, as its name implies, it is the opposite of fuse technology. Instead of opening a fuse link to prevent a signal connection, an insulator layer between interconnects has an electrical short created to produce a signal connection. Antifuse devices are programmed in a device programmer either by the end-user or by the factory or distributor.

Differences in architecture between CPLDs and FPGAs, among different HCPLD manufacturers, and among different families of devices from a single manufacturer can affect the efficiency of design implementation for a particular application. You may ask, "Does the architecture of this PLD family provide the best fit for my application?" It is very difficult, however, to predict which architecture may be the best choice to use for a complex digital system. Only a portion of the available gates can be utilized. Who knows how many equivalent gates will be needed for a large design? The basic design of the signal routing resources can affect how much of the PLD's logic resources can be utilized. The segmented interconnects often found in FPGAs can produce shorter delays between adjacent logic blocks, but they may also produce longer delays between the blocks that are further apart than would be produced by the continuous type of interconnect found in most CPLDs. There is no easy answer to your question, but every HCPLD manufacturer will give you an answer anyway: their product is best!

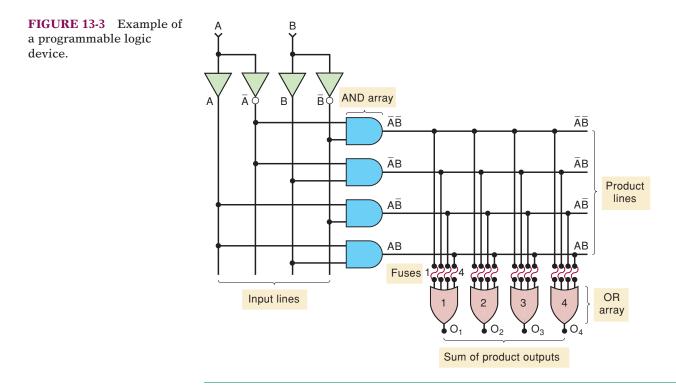
As you can see, the field of PLDs is quite diverse and it is constantly changing. You should now have the basic knowledge of the various types and technologies necessary to interpret PLD data sheets and learn more about them.

REVIEW QUESTIONS

- 1. What are the three major categories of digital systems?
- 2. What is the major disadvantage of a microprocessor/DSP design?
- 3. What does ASIC stand for?
- 4. What are the four types of ASICs?
- 5. What are HCPLDs?
- 6. What are two major differences between CPLDs and FPGAs?
- 7. What does volatility refer to?

13-2 FUNDAMENTALS OF PLD CIRCUITRY

A simple PLD device is shown in Figure 13-3. Each of the four OR gates can produce an output that is a function of the two input variables, A and B. Each output function is programmed with the fuses located between the AND gates and each of the OR gates.



Each of the inputs *A* and *B* feed both a noninverting buffer and an inverting buffer to produce the true and inverted forms of each variable. These are the *input lines* to the AND gate array. Each AND gate is connected to two different input lines to generate a unique product of the input variables. The AND outputs are called the *product lines*.

Each of the product lines is connected to one of the four inputs of each OR gate through a fusible link. With all of the links initially intact, each OR output will be a constant 1. Here's the proof:

$$O_1 = AB + AB + AB + AB$$

= $\overline{A}(\overline{B} + B) + A(\overline{B} + B)$
= $\overline{A} + A = 1$

Each of the four outputs O_1 , O_2 , O_3 , and O_4 can be *programmed* to be any function of *A* and *B* by selectively blowing the appropriate fuses. PLDs are designed so that a blown OR input acts as a logic 0. For example, if we blow fuses 1 and 4 at OR gate 1, the O_1 output becomes

$$O_1 = 0 + \overline{A}B + A\overline{B} + 0 = \overline{A}B + A\overline{B}$$

We can program each of the OR outputs to any desired function in a similar manner. Once all of the outputs have been programmed, the device will permanently generate the selected output functions.

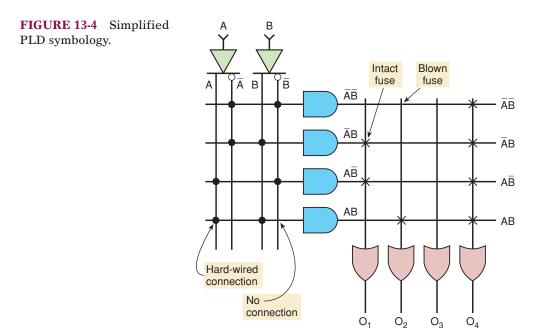
PLD Symbology

The example in Figure 13-3 has only two input variables and the circuit diagram is already quite cluttered. You can imagine how messy the diagram would be for PLDs with many more inputs. For this reason, PLD manufacturers have adopted a simplified symbolic representation of the internal circuitry of these devices.

Figure 13-4 shows the same PLD circuit as Figure 13-3 using the simplified symbols. First, notice that the input buffers are represented as a single buffer with two outputs, one inverted and one noninverted. Next, note that a *single line* is shown going into the AND gate to represent all four inputs. Each time the row line crosses a column represents a separate input to the AND gate. The connections from the input variable lines to the AND gate inputs are indicated as dots. A dot means that this connection to the AND gate input is hard-wired (i.e., one that cannot be changed). At first glance, it looks like the input variables are connected to each other. It is important to realize that this is *not* the case because the single row line represents *multiple* inputs to the AND gate.

The inputs to each of the OR gates are also designated by a single line representing all four inputs. An X represents an intact fuse connecting a product line to one input of the OR gate. The absence of an X (or a dot) at any intersection represents a blown fuse. For OR gate inputs, blown fuses (unconnected inputs) are assumed to be LOW, and for AND gate inputs, blown fuses are HIGH. In this example, the outputs are programmed as

$$O_1 = \overline{A}B + A\overline{B}$$
$$O_2 = AB$$
$$O_3 = 0$$
$$O_4 = 1$$



REVIEW QUESTIONS

- 1. What is a PLD?
- 2. What would output O_1 be in Figure 13-3 if fuses 1 and 2 were blown?
- 3. What does an X represent on a PLD diagram?
- 4. What does a dot represent on a PLD diagram?

13-3 PLD ARCHITECTURES

The concept of PLDs has led to many different architectural designs of the inner circuitry of these devices. In this section, we will explore some of the basic differences in architecture.

PROMs

The architecture of the programmable circuits in the previous section involves programming the connections to the OR gate. The AND gates are used to decode all the possible combinations of the input variables, as shown in Figure 13-5(a). For any given input combination, the corresponding row is activated (goes HIGH). If the OR input is connected to that row, a HIGH appears at the OR output. If the input is not connected, a LOW appears at the OR output. Does this sound familiar? Refer back to Figure 12-9. If you think of the input variables as address inputs and the intact/blown fuses as stored 1s and 0s, you should recognize the architecture of a PROM.

Figure 13-5(b) shows how the PROM would be programmed to generate four specified logic functions. Let's follow the procedure for output $O_3 = AB + \overline{C}\overline{D}$.

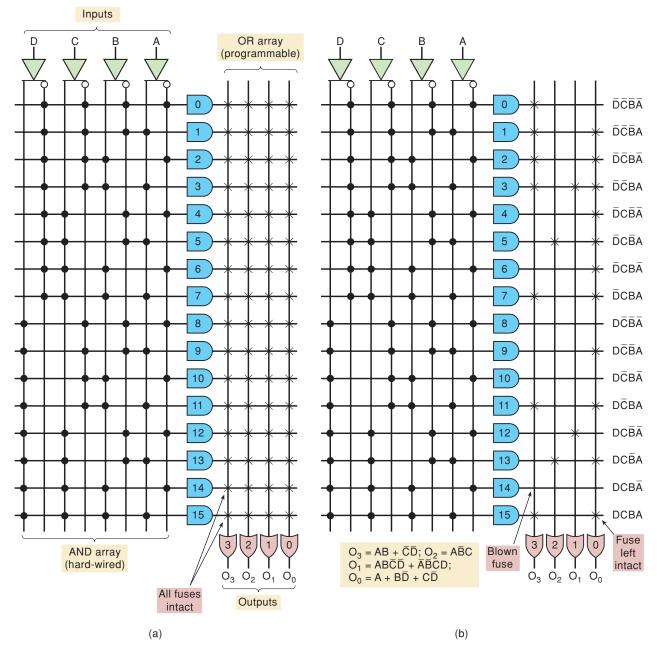


FIGURE 13-5 (a) PROM architecture makes it suitable for PLDs; (b) fuses are blown to program outputs for given functions.

The first step is to construct a truth table showing the desired O_3 output level for all possible input combinations (Table 13-1).

Next, write down the AND products for those cases where the output is to be a 1. The O_3 output is to be the OR sum of these products. Thus, only the fuses that connect these product terms to the inputs of OR gate 3 are to be left intact. All others are to be blown, as indicated in Figure 13-5(b). This same procedure is followed to determine the status of the fuses at the other OR gate inputs.

The PROM can generate any possible logic function of the input variables because it generates every possible AND product term. In general, any application that requires every input combination to be available is a good candidate for a PROM. However, PROMs become impractical when a large

```
TABLE 13-1
```

D	С	В	Α	O 3		
0	0	0	0	1	\rightarrow	$\overline{D}\overline{C}\overline{B}\overline{A}$
0	0	0	1	1	\rightarrow	$\overline{D}\overline{C}\overline{B}A$
0	0	1	0	1	\rightarrow	$\overline{D}\overline{C}B\overline{A}$
0	0	1	1	1	\rightarrow	$\overline{D}\overline{C}BA$
0	1	0	0	0		
0	1	0	1	0		
0	1	1	0	0		
0	1	1	1	1	\rightarrow	$\overline{D}CBA$
1	0	0	0	0		
1	0	0	1	0		
1	0	1	0	0		
1	0	1	1	1	\rightarrow	$D\overline{C}BA$
1	1	0	0	0		
1	1	0	1	0		
1	1	1	0	0		
1	1	1	1	1	\rightarrow	DCBA

number of input variables must be accommodated because the number of fuses doubles for each added input variable.

Calling a PROM a PLD is really just a semantics issue. You already knew that a PROM is programmable and it is a logic device. This is just a way of using a PROM and thinking of its purpose as implementing SOP logic expressions rather than storing data values in memory locations. The real problem is translating the logic equations into the fuse map for a given PROM. A generalpurpose logic compiler designed to program SPLDs has a list of PROM devices that it can support. If you choose to use any old scavenged EPROM as a PLD, you may need to generate your own bit map (like they used to do it), which is very tedious.

Programmable Array Logic (PAL)

The PROM architecture is well suited for those applications where every possible input combination is required to generate the output functions. Examples are code converters and data storage (look-up) tables that we examined in Chapter 12. When implementing SOP expressions, however, they do not make very efficient use of circuitry. Each combination of address inputs must be fully decoded, and each expanded product term has an associated fuse that is used to OR them together. For example, notice how many fuses were required in Figure 13-5 to program the simple SOP expressions and how many product terms are often not used. This has led to the development of a class of PLDs called programmable array logic (PAL). The architecture of a PAL differs slightly from that of a PROM, as shown in Figure 13-6(a).

The PAL has an AND and OR structure similar to a PROM but in the PAL, inputs to the AND gates are programmable, whereas the inputs to the OR gates are hard-wired. This means that every AND gate can be programmed to generate any desired product of the four input variables and their complements. Each OR gate is hard-wired to only four AND outputs. This limits each output function to four product terms. If a function requires

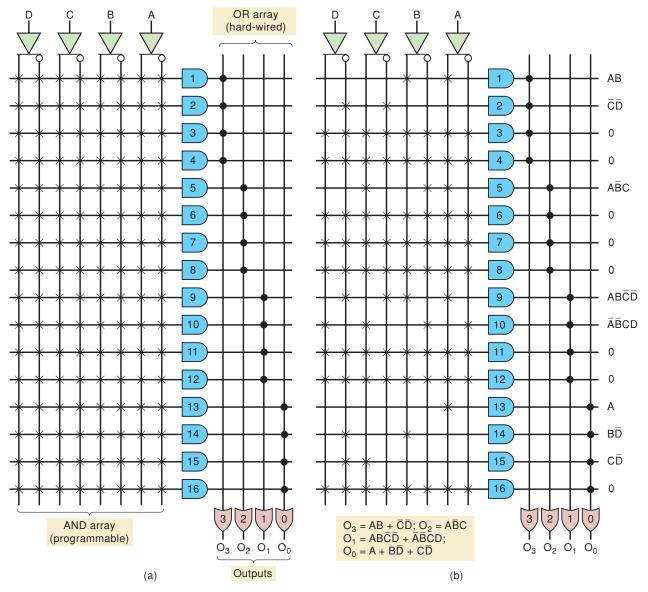


FIGURE 13-6 (a) Typical PAL architecture; (b) the same PAL programmed for the given functions.

more than four product terms, it cannot be implemented with this PAL; one having more OR inputs would have to be used. If fewer than four product terms are required, the unneeded ones can be made 0.

Figure 13-6(b) shows how this PAL is programmed to generate four specified logic functions. Let's follow the procedure for output $O_3 = AB + \overline{CD}$. First, we must express this output as the OR sum of four terms because the OR gates have four inputs. We do this by putting in 0s. Thus, we have

$$O_3 = AB + \overline{C}\overline{D} + 0 + 0$$

Next, we must determine how to program the inputs to AND gates 1, 2, 3, and 4 so that they provide the correct product terms to OR gate 3. We do this term by term. The first term, *AB*, is obtained by leaving intact the fuses that connect inputs *A* and *B* to AND gate 1 and by blowing all other fuses on that line.

Likewise, the second term, \overline{CD} , is obtained by leaving intact only the fuses that connect inputs \overline{C} and \overline{D} to AND gate 2. The third term is a 0. A constant 0 is produced at the output of AND gate 3 by leaving all of its input fuses intact. This would produce an output of $A\overline{ABBCCDD}$, which, as we know, is 0. The fourth term is also 0, so the input fuses to AND gate 4 are also left intact.

The inputs to the other AND gates are programmed similarly to generate the other output functions. Note especially that many of the AND gates have all of their input fuses intact because they need to generate 0s.

An example of an actual PAL integrated circuit is the PAL16L8, which has 10 logic inputs and eight output functions. Each output OR gate is hardwired to seven AND gate outputs, and so it can generate functions that include up to seven terms. An added feature of this particular PAL is that six of the eight outputs are fed back into the AND array, where they can be connected as inputs to any AND gate. This makes it very useful in generating all sorts of combinational logic.

The PAL family also contains devices with variations of the basic SOP circuitry we have described. For example, most PAL devices have a tristate buffer driving the output pin. Others channel the SOP logic circuit to a D FF input and use one of the pins as a clock input to clock all of the output flip-flops synchronously. These devices are referred to as *registered PLDs* because the outputs pass through a register. An example is the PAL16R8, which has up to eight registered outputs (which can also serve as inputs) plus eight dedicated inputs.

Field Programmable Logic Array (FPLA)

The field programmable logic array (FPLA) was developed in the mid-1970s as the first nonmemory programmable logic device. It used a programmable AND array as well as a programmable OR array. Although the FPLA is more flexible than the PAL architecture, it has not been as widely accepted by engineers. FPLAs are used mostly in state-machine design where a large number of product terms are needed in each SOP expression.

REVIEW QUESTIONS 1. Verify that the correct fuses are blown for the O_2 , O_1 , and O_0 functions in Figure 13-5(b). 2. A PAL has a hard-wired _____ array and a programmable _____ array. 3. A PROM has a hard-wired _____ array and a programmable _____ array. 4. How would the equation for the output of O_1 in Figure 13-5(b) change if all the fuses from AND gate 14 were left intact?

13-4 THE GAL 16V8 (GENERIC ARRAY LOGIC)

The GAL 16V8, introduced by Lattice Semiconductor, has an architecture that is very similar to the PAL devices described in the previous section. Standard, low-density PALs are one-time programmable. The GAL chip, on the other hand, uses an EEPROM array (located at row and column intersections in Figure 13-7) to control the programmable connections to the AND matrix, allowing them to be erased and reprogrammed at least 100 times. In addition to the AND and OR gates used to produce the sum of product functions, the GAL 16V8 contains optional flip-flops for register and counter applications, tristate buffers for the outputs, and control multiplexers used

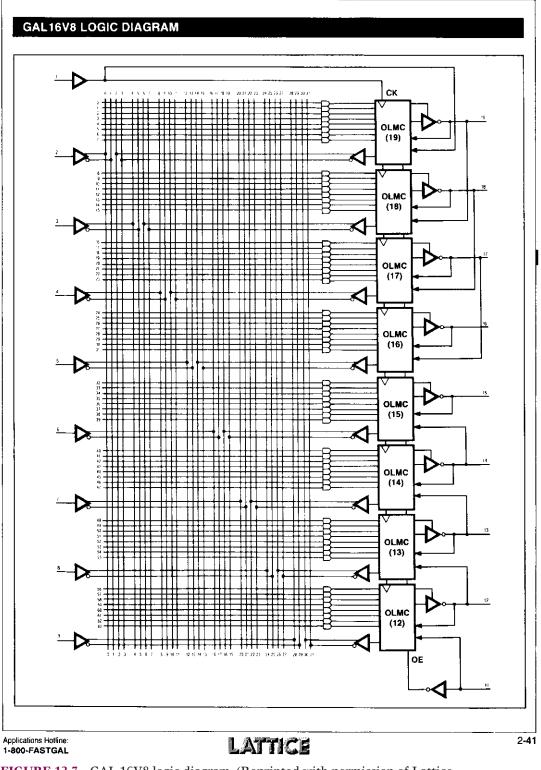


FIGURE 13-7 GAL 16V8 logic diagram. (Reprinted with permission of Lattice Semiconductor.)

to select the various modes of operation. Consequently, it can be used as a generic, pin-compatible replacement for most PAL devices. Specific locations in the memory array are designated to control the various programmable connections in the chip. Fortunately, it is not necessary to delve into the addresses of each bit location in the matrix. The programming software takes care of these details in a user-friendly manner.

The complete logic diagram of the GAL 16V8 is shown in Figure 13-7. This device has eight dedicated input pins (pins 2–9), two special function inputs (pins 1 and 11), and eight pins (12–19) that can be used as inputs or outputs. The major components of the GAL devices are the input term matrix; the AND gates, which generate the products of input terms; and the output logic macrocells (OLMCs). Notice that the eight inputs (pins 2–9) are each connected directly to a column of the input term matrix. The complement of each of these inputs is also connected to a column of the matrix. These pins must always be specified as inputs when programming the 16V8. A logic level and its complement are also fed from each OLMC back to a column of the input matrix. This accounts for the 32 input variables (columns in the input MND gates.

The flexibility of the GAL 16V8 lies in its programmable output logic macrocell. Eight different products (outputs of AND gates) are applied as inputs to each of the eight output logic macrocells. Within each OLMC, the products are ORed together to generate the sum of products (SOP). Recall from Chapter 4 that any logic function can be expressed in SOP form. Within the OLMC, the SOP output may be routed to the output pin to implement a combinational circuit, or it may be clocked into a D flip-flop to implement a registered output circuit.

To understand the detailed operation of the OLMC, refer to Figure 13-8. The figure shows the structure of OLMC(n), where *n* is a number from 12 to 19. Notice that seven of the products are unconditionally connected to the OR

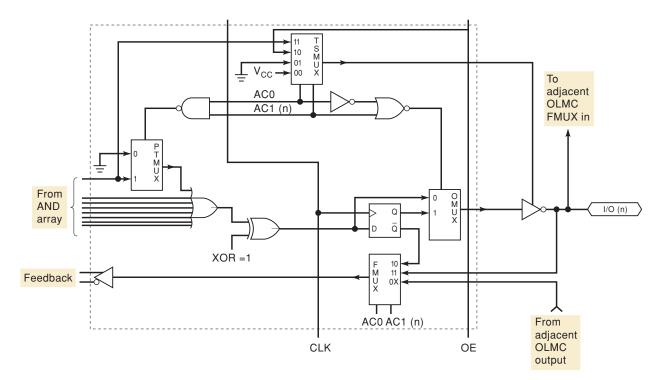


FIGURE 13-8 Output logic macrocell for the GAL 16V8. (Reprinted with permission of Lattice Semiconductor.)

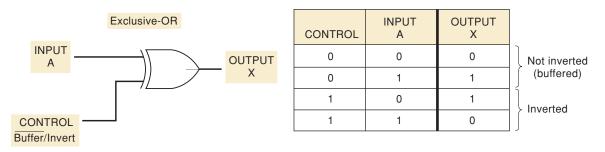


FIGURE 13-9 Using XOR to complement selectively.

gate inputs. The eighth product term is connected to a two-input product term multiplexer (PTMUX), which drives the eighth input to the OR gate. The eighth product term also connects to one input of a four-input multiplexer (TSMUX). The output of TSMUX enables the tristate inverter that drives the output pin [I/O(n)]. The output multiplexer (OMUX) is a two-input MUX that selects between the combinational output (OR gate) and the registered output (the D flip-flop). A fourth MUX selects the logic signal that is fed back to the input matrix. This is called the *feedback* multiplexer (FMUX).

Each of these multiplexers is controlled by programmable bits (AC1 and AC0) in the EEPROM matrix. This is the way that the OLMC configuration can be altered by the programmer. Another programmable bit is the input to the XOR gate. This provides the programmable output polarity feature. Recall that an XOR gate can be used to complement a logic signal selectively, as shown in Figure 13-9. When the control line is a logic 0, the XOR will pass the logic level at input *A* with no inversion. When the control bit is a logic 1, the XOR will invert the signal so that $X = \overline{A}$. In Figure 13-8, the programmable bit (labeled XOR) is a logic 1 under normal positive logic conditions. This inverts the output of the OR gate, which is inverted again when it passes through the tristate inverting buffer on the output.

We can understand the various configuration options by studying the possible inputs to each multiplexer. The TSMUX controls the tristate buffer's enable input. If the V_{CC} input is selected, the output is always enabled, like a standard combinational logic gate. If the grounded input is selected, the tristate output of the inverter is always in its high-impedance state (allowing the I/O pin to be used as an input). Another input to the MUX that may be selected comes from the *OE* input, which is pin 11. This allows the output to be enabled or disabled by an external logic signal applied to pin 11. The last possible input selection is a product term from the eighth AND gate. This allows an AND combination of terms from the input matrix to enable or disable the output.

The FMUX selects the signal that is fed back into the input matrix. In this case, there are three possible selections. Selecting the *MUX* input that is connected to an adjacent stage or the *MUX* input connected to its own OLMC I/O pin allows an existing output state to be fed back to the input matrix in some of the modes of operation. This feature gives the GAL 16V8 the ability to implement sequential circuits such as the cross-coupled NAND gate latch circuit described in Chapter 5. This feedback option also allows an I/O pin to be used as a dedicated input as opposed to an output. One of these two feedback paths is chosen, depending on the MODE that the chip is programmed for. The third option, selecting the output from the D flip-flop, allows the present state of the flip-flop (which can be used to determine the next state) to be fed back to the input matrix. This allows synchronous sequential circuits, such as counters and shift registers, to be implemented.

With all of these options, it would seem that there must be a long list of possible configurations. In actual practice, all these configuration decisions are made by the software. Actually, the GAL 16V8 has only three different modes: (1) *simple mode*, which is used to implement simple SOP combinational logic without tristate outputs; (2) *complex mode*, which implements SOP combinational logic with tristate outputs that are enabled by an AND product expression; and (3) *registered mode*, which allows individual OLMCs to operate in a combinational configuration with tristate outputs (similar to the complex mode) or in a synchronous mode with clocked D FFs synchronized to a common clock signal.

The GAL 16V8 is an inexpensive and versatile PLD chip, but what if a design requires more hardware resources than is contained in the 16V8? It may be possible to split the design into smaller blocks that can be implemented in several 16V8 chips. Fortunately, there are other members of the GAL family to choose from. Another popular, general-purpose PLD is the GAL 22V10. This device has 10 output pins and 12 input pins in an architecture that is similar but not identical to the GAL 16V8. Groups of product terms are logically summed with an OR gate, which feeds an OLMC. Unlike the 16V8, however, each OR gate in the 22V10 does not combine the same number of product terms. The number of terms ranges from eight all the way up to 16. To take advantage of the extra terms, you must assign the larger Boolean expressions to the correct output pin. The D flip-flops contained in the OLMCs also have asynchronous reset and synchronous preset capabilities. A newer version of the 22V10—the ispGAL 22V10—is now available. This device is said to be insystem programmable (ISP). Instead of requiring a programmer, as is needed to program PALs and standard GAL chips, a cable from the PC is connected directly to a special set of pins on the ISP device to do the programming.

REVIEW QUESTIONS

- 1. Name two advantages of GAL devices over PAL devices.
- 2. Name the three modes of operation for a GAL 16V8.

13-5 THE ALTERA EPM7128S CPLD

We will investigate the architecture of the EPM7128S, an EEPROM-based device in the Altera MAX7000S CPLD family. This device is found on several educational development boards, including the Altera UP2, DeVry eSOC, and RSR PLDT-2. The block diagram for this family is shown in Figure 13-10. The major structures in the MAX7000S are the **logic array blocks (LABs)** and the **programmable interconnect array (PIA)**. A LAB contains a set of 16 macrocells and looks very similar to a single SPLD device. Each macrocell consists of a programmable AND/OR circuit and a programmable register (flip-flop). The macrocells in a single LAB can share logic resources such as common product terms or unused AND gates. The number of macrocells contained in one of the MAX7000S family devices depends on the part number. As shown in Table 13-2, the EPM7128S has 128 macrocells arranged in eight LABs.

Logic signals are routed between LABs via the PIA. The PIA is a global bus that connects any signal source to any destination within the device. All inputs to the MAX7000S device and all macrocell outputs feed the PIA. Up to 36 signals can feed each LAB from the PIA. Only signals needed to produce the required functions for any LAB are fed into that LAB.

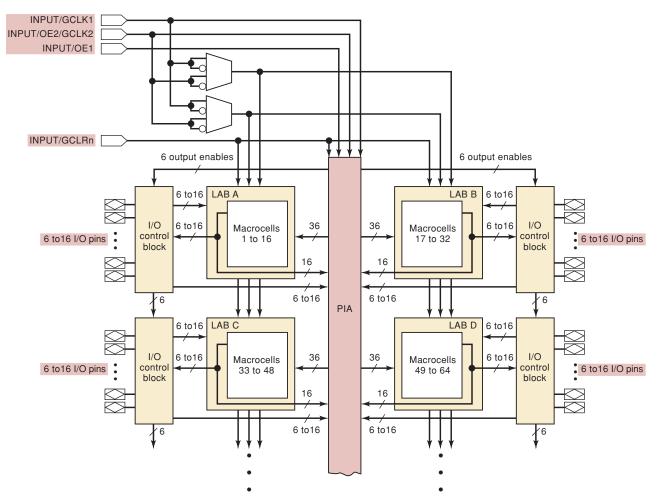


FIGURE 13-10 MAX7000S family block diagram. (Courtesy of Altera Corporation.)

I/O pins in the MAX7000S family are connected to specific macrocells. The number of I/O pins available to the user depends on the device package. An EPM7128S in a 160-pin PQFP package has 12 I/Os per LAB plus four additional input-only pins, for a total of 100 pins. On the other hand, in an 84-pin PLCC package, which is included on the above-mentioned development boards, there are eight I/Os per LAB plus the four extras, for a total of 68 I/O pins. The EPM7128S is an in-system programmable (ISP) device. The ISP feature utilizes a joint test action group (JTAG) interface that requires four specific pins to be dedicated to the programming interface and are therefore not available for general user I/O. The target PLD can be programmed in-system via the JTAG pins by connecting them to the parallel port of a PC with driver gates, as shown

Feature	EPM7032S	EPM7064S	EPM7128S	EPM7160S	EPM7192S	EPM7256S
Usable gates	600	1250	2500	3200	3750	5000
Macrocells	32	64	128	160	192	256
LABs	2	4	8	10	12	16
Maximum number of user I/O pins	36	68	100	104	124	164

 TABLE 13-2
 Altera MAX7000S family device features.

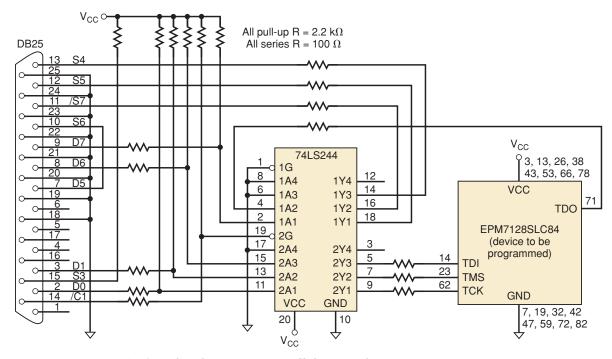


FIGURE 13-11 JTAG interface between PC parallel port and EPM7128SLC84.

in Figure 13-11. The JTAG signals are named TDI (test data in), TDO (test data out), TMS (test mode select), and TCK (test clock). This brings the user I/O pin total for an EPM7128SLC84 (an EPM7128S in an 84-pin PLCC package) down to 64 pins. All 68 pins, however, can be used for user I/O if the EPM7128SLC84 is programmed in a PLD programmer instead of in-system. When the design is compiled, you must indicate whether or not the device will use a JTAG interface. In either case, you can see that some macrocells will not be connected directly to user I/O pins. These macrocells can be utilized by the compiler for internal (buried) logic.

The four input-only pins found on devices in the MAX7000S family can be configured as specific high-speed control signals or as general user inputs. GCLK1 is the primary global clock input for all macrocells in the device. It is used to clock all registers synchronously in a design. It is located on pin 83 on an EPM7128SLC84 (see Figure 13-12). Pin 2 on this device is GCLK2 (secondary global clock). As an alternative, this pin may be used as a secondary global output enable (OE2) for any macrocells designated to have a tristate output. The primary tristate enable, OE1, is located on pin 84. The last of the four global control signals is GCLRn on pin 1. This active-LOW input can control the asynchronous clear on any macrocell register. How these pins are to be used for a specific application is assigned in MAX+PLUS II or Quartus II either automatically by the compiler or manually by the designer during the design process.

The I/O control blocks (see Figure 13-10) configure each I/O pin for input, output, or bidirectional operation. All I/O pins in the MAX7000S family have a tristate output buffer that is (1) permanently enabled or disabled, (2) controlled by one of the two global output enable pins, or (3) controlled by other inputs or functions generated by other macrocells. When an I/O pin is configured as an input, the associated macrocell can be used for buried logic. During in-system programming, the I/O pins will be made tristate and internally pulled up to eliminate board conflicts.



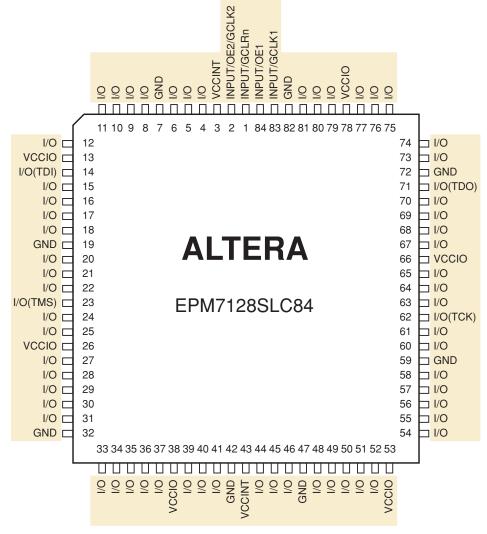


Figure 13-13 shows the block diagram for a MAX7000S macrocell. Each macrocell can produce either a combinational or a registered output. The register (flip-flop) contained in a macrocell will be bypassed to produce a combinational output. The programmable sum of product circuit looks very much like that found in a GAL chip. Each macrocell can produce five product terms. While this is fewer than was found in the simpler GAL chips discussed earlier, it is often sufficient for most logic functions. If more product terms are needed, the compiler will automatically program a macrocell to borrow up to five product terms from each of three adjacent macrocells in the same LAB. This parallel logic expander option can provide a total of 20 product terms. The borrowed gates are no longer usable by the macrocell from which they are borrowed. Another expansion option, available in each LAB, is called shared logic expanders. Instead of adding more product terms, this option allows a common product term to be produced once and then used by several macrocells within the LAB. Only one product term per macrocell can be used in this fashion, but with 16 macrocells per LAB, this makes up to 16 common product terms available. The compiler automatically optimizes the allocation of available product terms within a LAB according to the logic requirements of the design. Using either expander option does incur a small amount of additional propagation delay.

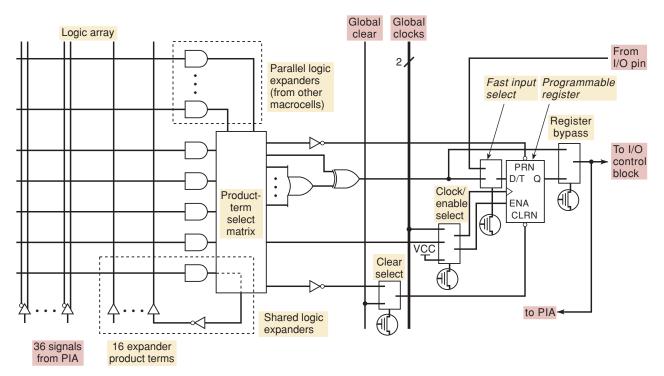


FIGURE 13-13 MAX7000S family macrocell. (Courtesy of Altera Corporation.)

For registered functions, each macrocell flip-flop can be programmed individually to implement D, T, JK, or SR operation. Each programmable register can be clocked in three different modes: (1) with a global clock signal, (2) with a global clock signal when the flip-flop is enabled, or (3) with an array clock signal produced by a buried macrocell or a (nonglobal) input pin. In the EPM7128S, either of the two global clock pins (GCLK1 or GCLK2) can be used to produce the fastest clock-to-Q performance. Either clock edge can be programmed to trigger the flip-flops. Each register can be preset asynchronously or cleared with an active-HIGH or active-LOW product term. Each register may also be cleared with the active-LOW global clear pin (GCLRn). A fast data input path from an I/O pin to the registers, bypassing the PIA, is also available. All registers in the device will be reset automatically at power-up.

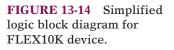
MAX7000S devices have a power-saving option that allows the designer to program each individual macrocell for either high-speed (turbo bit turned on) or low-power (turbo bit turned off) operation. Because most logic applications require only a small fraction of all gates to operate at maximum frequency, this feature may produce a significant savings in total system power consumption. Speed-critical paths in the design can run at maximum speed, while the remaining signal paths can operate at reduced power.

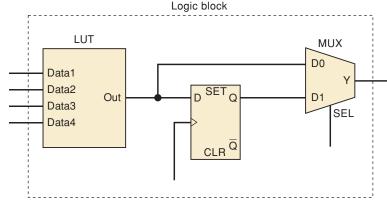
REVIEW QUESTIONS	1. What is a macrocell?
	2. What is an ISP device?
	3. What special control functions are provided with the four input-only pins on a MAX7000S device?
	4. What system advantage is achieved by slowing down selected macrocells in a MAX7000S device?

13-6 THE ALTERA FLEX10K FAMILY

The Altera FLEX10K family of programmable logic devices has a very different architecture. Instead of the programmable AND/fixed-OR gate array used in the MAX7000S devices, this family is based on a look-up table (LUT) architecture. The look-up table produces logic functions by storing the function's output results in an SRAM-based memory. It functions essentially like the truth table for the logic function. SRAM technology for PLDs programs much faster than EEPROM-based devices, and it also results in a very high density of storage cells that are used to program the larger PLDs. SRAMbased PLDs that use the LUT architecture are generally classified in the industry as field programmable gate arrays. Unlike most FPGAs, however, Altera has chosen to utilize a programmable signal routing design for the FLEX10K family that looks more like an enhanced version of the PIA found in the CPLD MAX7000S family. As a result, the FLEX10K family has architectural characteristics that are a combination of the two HCPLD classifications. Based on the high-density architecture of the logic cells, the FLEX10K devices are generally classified as FPGAs.

Let us examine the concept of a look-up table. The LUT is the portion of the programmable logic block that produces a combinational function (see Figure 13-14). This function can be used as the output of the logic block or it may be registered (controlled by the internal MUX). The look-up table itself consists of a set of flip-flops that store the desired truth table for our function. LUTs are usually rather small, typically handling four input variables, and so our truth table would have a total of 16 combinations. We will need a flip-flop to store each of the 16 function values (see Figure 13-15). Up to four input variables in our example LUT will be connected to the data inputs on the decoder block using programmable interconnects. The input combination that is applied will determine which of the 16 flip-flops will be selected to feed the output via the tristate buffers. The look-up table is basically a 16×1 SRAM memory block. All we have to do to create any desired function (of up to four input variables) is to store the appropriate set of 0s and 1s in the LUT's flip-flops. That is essentially what is done to program this type of PLD. Because the flip-flops are volatile (they are SRAM), we need to load the LUT memory for the desired functions whenever the PLD is powered-up. This process is called configuring the PLD. Other portions of the device are also programmed in the same fashion using other SRAM memory bits to store the programming information. This is the basic programming technique for the logic blocks, called logic elements (LEs), found in the FLEX10K devices.





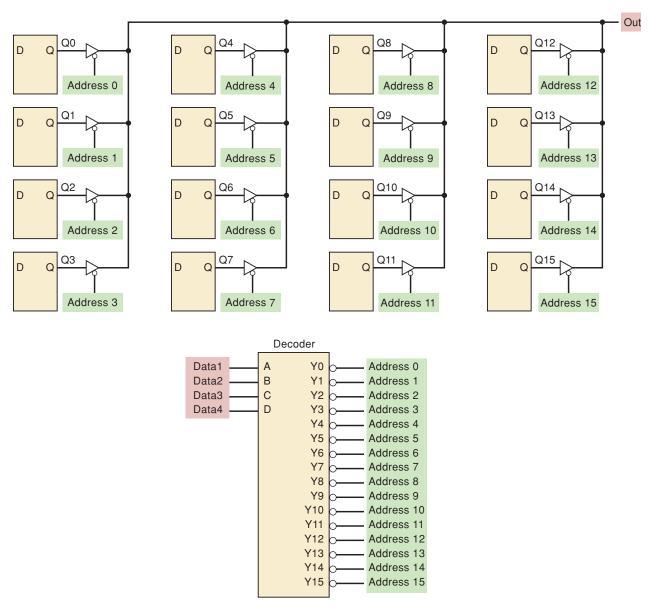


FIGURE 13-15 Functional block diagram for an LUT.

Figure 13-16 shows the block diagram for a FLEX10K logic element. It contains the LUT and programmable register, as well as cascade- and carry-expansion circuitry, programmable control functions, and local and global bus interconnections. The programmable flip-flop can be configured for D, T, JK, or SR operation and will be bypassed for combinational functions. The flip-flop control signals (clock, clear, and preset) can be driven selectively by global inputs, general-purpose I/O pins, or any internally created functions. The LE can produce two outputs to drive local (LAB) and global (FastTrack) interconnects on the chip. This allows the LUT and the register in one LE to be used for unrelated functions. Two types of high-speed data paths—cascade chains and carry chains—connect adjacent LEs without using local interconnects. The cascade-chain expansion allows the FLEX10K architecture to create functions with more than four input variables. Adjacent LUTs can be paralleled together, with each additional LUT providing four more input variables. The carry chain provides a fast carry-forward function between

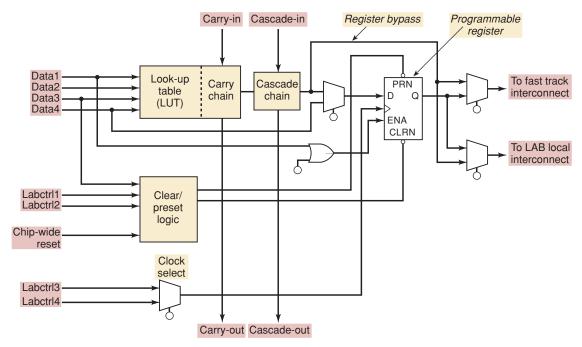
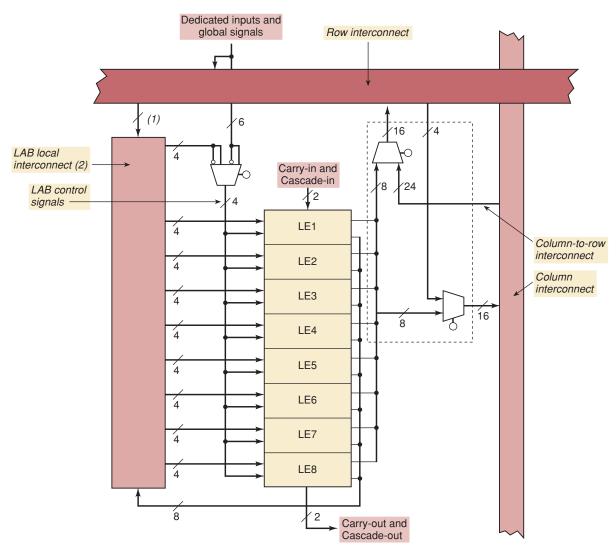
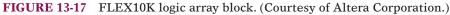


FIGURE 13-16 FLEX10K logic element. (Courtesy of Altera Corporation.)

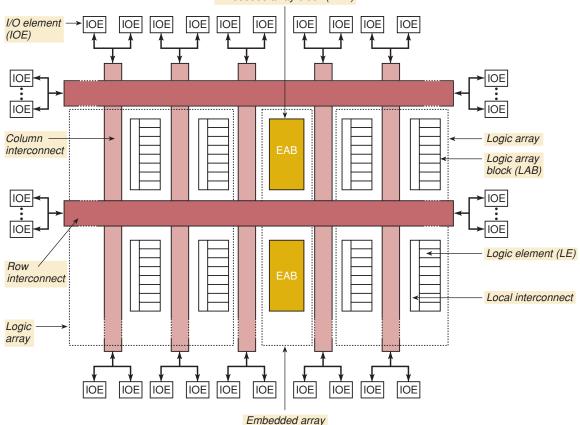




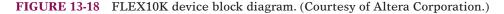
LEs, which allows for efficient implementation of functions that build on other functions such as those found in counters, adders, and comparators. In these functions, the upper bits depend on the lower bits. Without an expansion feature like the carry chain, the propagation delays can become quite long for larger circuits. Cascade-chain and carry-chain logic can be created automatically by the compiler software or manually by the designer during design entry. Propagation delays will increase by a small amount when using the expansion options. The MAX+PLUS II or Quartus II Timing Analyzer calculates these added delays for a given design. Intensive use of carry and cascade chains can reduce routing flexibility and should therefore be limited to speed-critical portions of a design.

The logic array block for the FLEX10K family contains eight logic elements and the local interconnect for that LAB (see Figure 13-17). Signals from one LE to another within an LAB are routed with the local interconnect. The row and column interconnects, which Altera has named a FastTrack interconnect, provide the signal pathways between LABs. Each LAB has four control signals available to all eight LEs. Two can be used for register clocks and the other two are for preset or clear.

The overall block diagram for a FLEX10K device is shown in Figure 13-18. In addition to the logic array blocks and FastTrack interconnects that we have already described, the devices contain I/O elements (IOEs) and embedded array blocks (EABs). The IOEs each contain a bidirectional I/O buffer and a register that can be used for either input or output data storage. Each EAB



Embedded array block (EAB)



EPF10K10	EPF10K20	EPF10K30	EPF10K40	EPF10K50	EPF10K70	EPF10K100	EPF10K120	EPF10K250
10,000	20,000	30,000	40,000	50,000	70,000	100,000	120,000	250,000
31.000	63 000	69.000	93.000	116.000	118 000	158 000	211 000	310,000
,	,	,	,	,	,	,	,	,
576	1,152	1,728	2,304	2,880	3,744	4,992	6,656	12,160
72	144	216	288	360	468	624	832	1,520
3	6	6	8	10	9	12	16	20
150	189	246	189	310	358	406	470	470
	10,000 31,000 576 72 3	10,000 20,000 31,000 63,000 576 1,152 72 144 3 6	10,000 20,000 30,000 31,000 63,000 69,000 576 1,152 1,728 72 144 216 3 6 6	10,000 20,000 30,000 40,000 31,000 63,000 69,000 93,000 576 1,152 1,728 2,304 72 144 216 288 3 6 6 8	10,000 20,000 30,000 40,000 50,000 31,000 63,000 69,000 93,000 116,000 576 1,152 1,728 2,304 2,880 72 144 216 288 360 3 6 6 8 10	10,000 20,000 30,000 40,000 50,000 70,000 31,000 63,000 69,000 93,000 116,000 118,000 576 1,152 1,728 2,304 2,880 3,744 72 144 216 288 360 468 3 6 6 8 10 9	10,000 20,000 30,000 40,000 50,000 70,000 100,000 31,000 63,000 69,000 93,000 116,000 118,000 158,000 576 1,152 1,728 2,304 2,880 3,744 4,992 72 144 216 288 360 468 624 3 6 6 8 10 9 12	10,000 20,000 30,000 40,000 50,000 70,000 100,000 120,000 31,000 63,000 69,000 93,000 116,000 118,000 158,000 211,000 576 1,152 1,728 2,304 2,880 3,744 4,992 6,656 72 144 216 288 360 468 624 832 3 6 6 8 10 9 12 16

TABLE 13-3 A	ltera FLEX10K	family device	features.
--------------	---------------	---------------	-----------

provides a flexible block of 2048 bits of RAM storage for various internal memory applications. Combining multiple EABs on one chip can create larger blocks of RAM. An EAB can also be used to create large combinational functions by implementing an LUT.

The FLEX10K family contains several different sizes of parts, as shown in Table 13-3. The Altera UP2 educational development board also contains an EPF10K70 device in a 240-pin package. As you can see in the table, this device has a lot of logic resources available!

REVIEW QUESTIONS 1. What is a look-up table? 2. What advantage does SRAM programming technology have over EEPROM? 3. What disadvantage does SRAM programming technology have compared to EEPROM?

4. What are EABs? What can they be used for?

13-7 THE ALTERA CYCLONE FAMILY

New families of HCPLD devices are continually being developed. The architectures of these new families provide various combinations of enhancements in logic and signal routing resources, in density (higher number of logic elements), in the amount of embedded memory, in the number of available user I/O pins, higher speeds, and lower costs. Another Altera family that may be of interest to us is the Cyclone family. The UP3 educational development board from Altera contains a Cyclone EP1C6 device. In a Cyclone device, logic functions are implemented in LEs (logic elements) that contain a four-input LUT (look-up table) and a programmable register (D flip-flop) similar to those found in FLEX10K devices. The Cyclone LE contains advanced features to provide more efficient logic utilization than with the FLEX10K. The Cyclone LE, for example, has been enhanced to more efficiently create **TABLE 13-4**AlteraCyclone family devicefeatures.

Feature	EP1C3	EP1C4	EP1C6	EP1C12	EP1C20
LEs	2,910	4,000	5,980	12,060	20,060
M4K RAM blocks	13	17	20	52	64
Total RAM bits	59,904	78,336	92,160	239,616	294,912
PLLs	1	2	2	2	2
Maximum number of I/O pins	104	301	185	249	301

digital applications that use adder/subtractors, asynchronous loading of the programmable register, and shift registers. The logic array blocks in Cyclone devices consist of 10 LEs and a local interconnect. This family also contains blocks of 4K bits of RAM memory that can be configured as dual-port or single-port memory with words up to 36 bits wide. A global clock network with eight global clock lines provides clocks for all I/O elements, LEs, and memory blocks. Internal phase-lock loops (PLLs) provide clock frequency multiplication and division and clock signal phase shifting. The features of the Cyclone family devices are compared in Table 13-4. Cyclone devices have the capability to interface with other digital circuits using multiple I/O standards, but they do not support 5-V I/O. Cyclone family devices are not supported by MAX+PLUS II design software.

SUMMARY

- 1. Programmable logic devices (PLDs) are the key technology in the future of digital systems.
- 2. PLDs can reduce parts inventory, simplify prototype circuitry, shorten the development cycle, reduce the size and power requirements of the product, and allow the hardware of a circuit to be upgraded easily.
- 3. The major digital system categories are standard logic, applicationspecific integrated circuits (ASICs), and microprocessor/digital signal processing (DSP) devices.
- 4. ASIC devices may be programmable logic devices (PLDs), gate arrays, standard cells, or full-custom devices.
- 5. PLDs are the least expensive type of ASIC to develop.
- 6. Simple PLDs (SPLDs) contain the equivalent of 600 or fewer gates and are programmed with fuse, EPROM, or EEPROM technology.
- 7. High-capacity PLDs (HCPLDs) have two major architectural categories: complex programmable logic devices (CPLDs) and field programmable gate arrays (FPGAs).
- 8. The most common CPLD programming technologies are EEPROM and flash, both of which are nonvolatile.
- 9. The most common FPGA programming technology is SRAM, which is volatile.
- 10. The GAL 16V8 is one of the simplest PLDs available but is still widely used and demonstrates the basic principles behind all PLDs.
- 11. The Altera EPM7128S CPLD contains 128 macrocells, each of which contains a programmable AND/OR circuit and a programmable register.
- 12. The EPM7128SLC84 can have up to 68 inputs and outputs.

- 13. The MAX7000S family of CPLDs is in-system programmable (ISP).
- 14. The Altera FLEX10K and Cyclone families of devices use a look-up table (LUT) architecture in an SRAM technology.
- 15. SRAM programming technology is volatile, meaning that the devices must be reconfigured at power-up.

IMPORTANT TERMS

standard logic microprocessor digital signal processing (DSP) application-specific integrated circuit (ASIC) programmable logic device (PLD) gate array standard-cell ASIC full-custom ASIC simple PLD (SPLD) complex PLD (CPLD) field programmable gate array (FPGA) high-capacity PLD (HCPLD) one-time programmable (OTP) programmable array logic (PAL) macrocell look-up table (LUT) logic array block (LAB) programmable interconnect array (PIA) logic element (LE)

PROBLEMS

SECTION 13-1

- 13-1. Describe each of the following major digital system categories:
 - (a) Standard logic
 - (b) ASICs
 - (c) Microprocessor/DSP
- 13-2.*Name three factors that are generally considered when making design engineering decisions.
- 13-3. Why is a microprocessor/DSP system called a software solution for a design?
- 13-4.*What major advantage does a hardware design solution have over a software solution?
- 13-5. Describe each of the following four ASIC subcategories:
 - (a) PLDs
 - (b) Gate arrays
 - (c) Standard-cell
 - (d) Full-custom
- 13-6.*What are the major advantages and disadvantages of a full-custom ASIC?
- 13-7. Name the six PLD programming technologies. Which is one-time programmable? Which is volatile?
- 13-8.*How is the programming of SRAM-based PLDs different from other programming technologies?

SECTION 13-5

13-9. Describe the functions of each of the following architectural structures found in the Altera MAX7000S family:

^{*}Answers to problems marked with an asterisk can be found in the back of the text.

- (a) LAB
- (b) PIA
- (c) Macrocell
- 13-10.* What two ways can be used to program the MAX7000S family devices?
- 13-11. What standard device interface is used for in-system programming in the MAX7000S family?
- 13-12.*What are the four input-only pins on the EPM7128SLC84 (by pin number and function)?
- 13-13. What is the advantage of using one of the global clock inputs for registered operation?

SECTION 13-6

13-14.* What is the fundamental architectural difference between the MAX 7000S and FLEX10K families? What is the programming technology used by each family? Which family is nonvolatile? Which family contains more logic resources?

ANSWERS TO SECTION REVIEW QUESTIONS

SECTION 13-1

1. Standard logic, ASICs, microprocessor 2. Speed 3. Application-specific integrated circuit 4. Programmable logic devices, gate arrays, standard cells, full custom 5. High-capacity programmable logic device 6. (1) Logic blocks: programmable AND/fixed-OR CPLD versus look-up table FPGA (2) Signal routing resources: uniform CPLD versus varied FPGA 7. Volatility refers to whether a PLD (or memory device) loses stored information when it is powered-down.

SECTION 13-2

1. An IC that contains a large number of gates whose interconnections can be modified by the user to perform a specific function. 2. $O_1 = A$ 3. An intact fuse 4. A hard-wired connection

SECTION 13-3

2. Hard-wired OR; programmable AND 3. Hard-wired AND; programmable OR 4. $O_1 = AB\overline{C}\overline{D} + \overline{A}\overline{B}CD + \overline{A}BCD = AB\overline{C}\overline{D} + \overline{A}CD$

SECTION 13-4

1. Erasable and reprogrammable; has an OLMC 2. Simple, complex, registered

SECTION 13-5

1. A macrocell is the programmable logic block in MAX7000S CPLDs consisting of a programmable AND/OR circuit and a programmable register (flip-flop). 2. An ISP PLD device is in-system programmable, which means that it can be programmed while connected in the circuit. 3. Global clocks, tristate output enables, asynchronous clear 4. Power consumption may be decreased by slowing down macrocells.

SECTION 13-6

 A look-up table is typically a 16-word by 1-bit SRAM array used to store the desired output logic levels for a simple logic function.
 SRAM programs faster and has a higher logic cell density than EEPROM.
 SRAM is volatile and must be reconfigured upon power-up of the device.
 Embedded array blocks provide RAM storage on the PLD.