CHAPTER 10: *IC Technology*

10.1 Introduction

In Chapter 1, we introduced the idea that embedded system design includes the use of three classes of technologies: processor technology, IC technology and design technology. Chapters 2–7 have focused mostly on processor technology, since one should understand how to build a processing system first, before learning what IC technologies are available to implement such a system, and before learning what design technologies are available to help build the system more rapidly. In this chapter, we provide an overview of three key IC technologies.

Several earlier chapters focused on an embedded system's structure. A system's structural representation describes the numbers and types of processors, memories, and buses, with which we implement the system's functionality. In this chapter, we focus on mapping that structure to a physical implementation. A system's physical implementation describes the mapping of the structure to actual chips, known as integrated circuits (ICs). A given structure can be mapped to one of several alternative physical implementations, each representing different design trade-offs. In fact, different parts of a structure may be mapped to different physical implementations. We might think of the structural representation as food menu for a banquet meal, and the physical implementation as the meal itself. A wedding banquet might call for a menu of chicken and vegetables, whereas a sports team banquet might call for spaghetti. Thus, we see trade-offs made in choosing the structure. Each meal itself can be prepared in different ways (e.g., the vegetables could be fresh or frozen). Thus, we see further trade-offs in choosing the physical implementation.

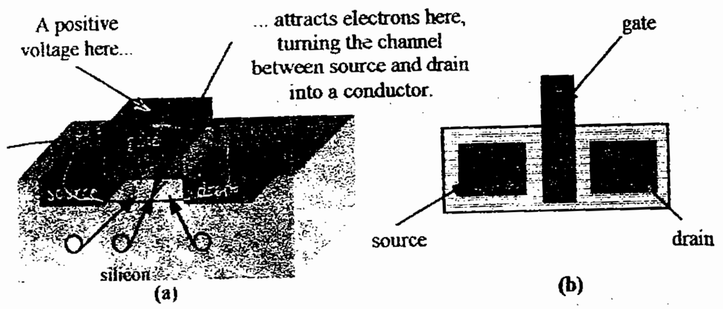
A positive voltage here...
... attracts electrons here, turning the channel between source and drain into a conductor.
gate
source
drain
silicon
(a)
(b)

Figure 10.1: (a) a CMOS transistor (nMOS), (b) top-down view.

We will consider three major categories of physical implementations, or IC technologies: custom, semi-custom and programmable. We should mention that the term "technology" in the context of ICs is often used to instead refer to a particular manufacturing process technology, describing the type and generation of manufacturing equipment being used to build the IC. For example, a chip may be manufactured using a CMOS 0.3-micron process technology. Our use of the term *IC technology* here refers instead to different categories of ICs; each category can be implemented using any manufacturing process.

One should recall from Chapter 1 that processor technologies and IC technologies are independent of one another. Any type of processor can be mapped to any type of IC. Furthermore, a single IC may implement part of a processor, an entire processor, or as is commonly the case today, multiple processors.

Let us begin our discussion of IC technology by again examining a basic transistor. A simplified version of a complementary metal-oxide-semiconductor (CMOS) transistor is shown in Figure 10.1(a). It consists of three terminals: the source, drain, and gate. The source and drain regions lie within the silicon itself, created by implanting ions into those regions. The gate, made from polysilicon, sits between the source and drain but above the silicon, separated from the silicon by a thin layer of insulator, silicon dioxide. The voltage at the gate controls whether current can flow between the source and the drain, while the insulator prevents current from flowing through the gate itself. For an nMOS transistor, if a high enough voltage is applied to the gate, electrons are attracted from throughout the silicon substrate into the channel between the source and the drain, creating a field that allows current conduction between source and drain. On the other hand, if 0 V is applied to the gate, then the channel cannot conduct.

Notice that the transistor has three layers. The source and drain regions lie within the silicon substrate; these regions are known as p-diffusion or n-diffusion, depending on whether we are building an nMOS or pMOS transistor. The silicon dioxide insulating layer lies on top of the substrate, and is typically referred to as oxide. The gate region lies on top of the silicon dioxide, and is made from a substance known as polysilicon.

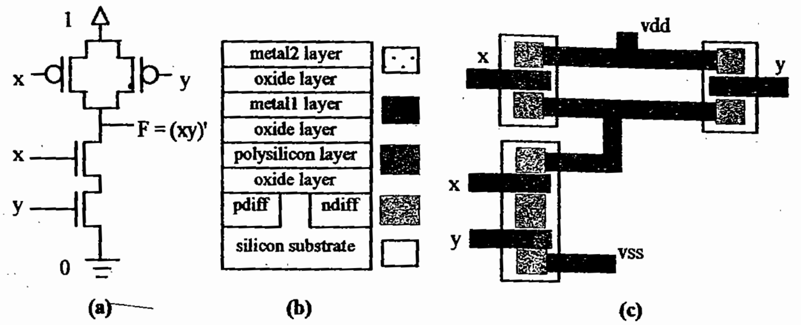

Figure 10.2: Depicting circuits in silicon: (a) a NAND circuit schematic, (b) layers, (c) top-down view of the NAND circuit on an IC.

When drawing tens or hundreds of transistors, the three-dimensional view of Figure 10.1(a) quickly becomes cumbersome to create and is really unnecessary. Instead, we can use a top-down two-dimensional view, wherein we first assign a unique pattern to represent each layer. Thus, the transistor of Figure 10.1(a) could be represented using the top-down view shown in Figure 10.1(b). The oxide layer is implicit, since it must always exist below the polysilicon.

Transistors are not very useful unless they are connected with one another, and so we'll need to introduce at least two layers of metal, which we'll call metal 1 and metal 2, to serve as connections. These layers will need to be insulated from each other and from the polysilicon, requiring two more oxide layers. Figure 10.2(b) depicts the ordering of the various layers we've introduced so far. This figure only depicts the ordering of layers, and doesn't show the connections that must also exist between higher and lower layers.

Note that we always need at least two layers of metal, since otherwise we will be unable to implement all but the most trivial of circuits. Think of trying to build a system of freeways without being allowed to build any bridges and without being able to cross roads going to different places, and you'll understand why at least two metal levels are necessary. Manufacturing processes that use even more than two levels of metal are common.

Now that we have layers for representing transistors and their connections, we can build a simple circuit on an IC. Suppose that we want to build the simple NAND circuit that was introduced in Chapter 2, and is redrawn in Figure 10.2(a) for convenience, consisting of two nMOS and two pMOS transistors. We'll use a top-down view, and use the patterns shown in Figure 10.2(b) for each layer. Figure 10.2(c) shows the top-down view of the NAND circuit, with black representing metal1. Take some time to see if you can see the correspondence between (a) and (c).

Let's consider how a circuit on an IC is actually manufactured. We'll begin by revisiting the simple transistor of Figure 10.1 and considering how this transistor would actually be manufactured. Since the transistor consists of three layers, we might mistakenly assume that we could manufacture this transistor in three steps. In such an idealized manufacturing

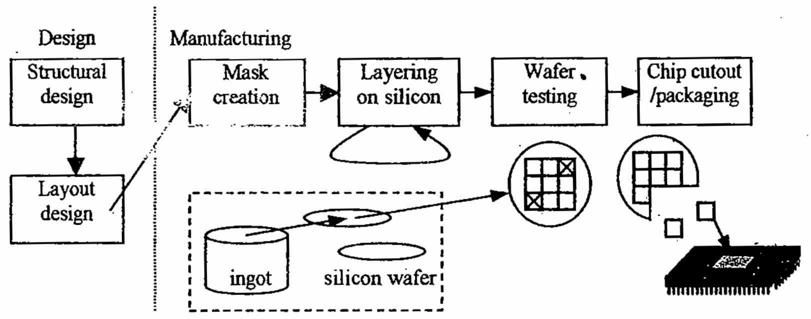

Figure 10.3: IC manufacturing steps.

process, we would first inject into the substrate the ions necessary to create the source and drain regions. Second, we would lay the silicon dioxide over the channel. Third, we would place the gate's polysilicon on top of the silicon dioxide.

Unfortunately, IC manufacturing is not quite so simple. Many steps are necessary to create each layer. For example, in a common manufacturing process, creating the silicon dioxide layer under the gate actually consists of several steps. First, we grow silicon dioxide on top of the entire IC by exposing the IC to extreme heat and gas, akin to growing rust on metal. Second, we cover the silicon dioxide with a substance called *photoresist*, which becomes soluble when exposed to ultraviolet light. Third, we pass ultraviolet light through a mask, which is designed to cast a shadow on the photoresist wherever we want silicon dioxide to stay; the remaining photoresist will be exposed to light and thus become soluble. This process is called *photolithography*. Fourth, we wash away the soluble photoresist with a solvent, thus exposing regions of silicon dioxide. Fifth, we etch away the exposed silicon dioxide with chemicals. Sixth, we remove the remaining photoresist to expose the regions of silicon dioxide that we wanted in the first place.

A similar process is repeated for each layer of the IC. An IC may have about 20 layers. So we see that there may be hundreds of steps, involving hundreds of masks, required to manufacture an IC.

Fortunately, embedded system designers need not worry too much about the details of the IC manufacturing process. Instead, they may only have to provide the input to the IC manufacturing process, which is a layout. A *layout* specifies the placement of every transistor and every wire connecting those transistors on the desired IC. The top-down view that we used for depicting a circuit on an IC was in fact a layout. A layout is akin to a map showing the placement of every city and every highway connecting those cities. From a layout, an IC manufacturer can derive the appropriate set of masks and thus manufacture the IC.

Figure 10.3 illustrates IC manufacturing steps. During the design phase, a designer creates a structural design and then generates a layout for that design. The design phase may take many months. Once fully satisfied, the designer provides the layout to an IC manufacturer, also known as a fabrication plant, "fab," or foundry. Because the layout is often

provided to the manufacturer on a magnetic tape commonly used for storing large quantities of digital data, providing the layout to a manufacturer is commonly referred to as "tape-out." Because part of the manufacturing process involves spinning molten silicon, generating ICs is also referred to as a "silicon spin." IC manufacturing may take months.

Manufacturing consists of several main steps. The first step is to create a set of masks corresponding to the layout. Hundreds of masks may be required. The second step is to use each of these masks to create the various layers on the silicon surface, consisting of several substeps per mask. We point out that this layering process doesn't just create a single IC, but rather numerous ICs at once. The reason is that ICs are built on a silicon wafer. A silicon wafer is a thin polished circle, sliced from a cylinder of silicon, like a pepperoni slice intended for a pizza is sliced from a cylinder of sausage. A silicon wafer may be tens of centimeters in diameter, whereas an IC is usually less than one centimeter on a side, thus meaning that a wafer can hold tens of ICs (perhaps 100). Thus, the masks actually contain tens of identical regions, so that tens of ICs are being created simultaneously on a silicon wafer, as shown in the figure. Think of this the next time that you watch a movie where everyone is trying to get their hands on some "prototype chip" that must be found lest the world be destroyed; if there's one chip, there's probably 50 or 100 more that were made on the same wafer, lying around somewhere! (Never mind — just enjoy the movie).

The third step is to test the ICs on the wafer. ICs determined to be bad are marked, literally, so that they will be thrown away later. The machines that perform such testing are known appropriately as *testers*. They use probes that contact the pads, or input and output ports, of a particular IC on the wafer. They then apply streams of input sequences and look for the appropriate output sequences. These testers are very expensive devices, and their cost per IC pin has actually increased. Unfortunately, with all the steps required to build an IC and because of the extremely small sizes of the transistors and wires involved, bad ICs are quite common. Yield is a measure of the percentage of good ICs versus bad ICs containing errors.

Finally, the last step is to cut out each IC and mount the good ones in an IC package, which of course gets tested again.

Now that we have a better idea of how ICs implement circuits and how ICs are manufactured, we can survey the three main IC technologies: full-custom, semi-custom, and programmable logic device IC technology. Figure 10.4 provides an overview of the designer's tasks for each of the technologies. Full-custom provides the best size and performance but is costly to design and manufacture, while programmable-logic devices involve the simplest design process at the expense of size and performance. Semi-custom represents a compromise between these two extremes.

10.2 Full-Custom (VLSI) IC Technology

In a full-custom IC technology, the designer creates the complete layout — a task often called *physical design* or *VLSI design* (where VLSI stands for very large scale integrated circuit). The designer must design or obtain a transistor-level circuit for every processor and memory. After this point, there are several key physical design tasks necessary to obtain a good layout:

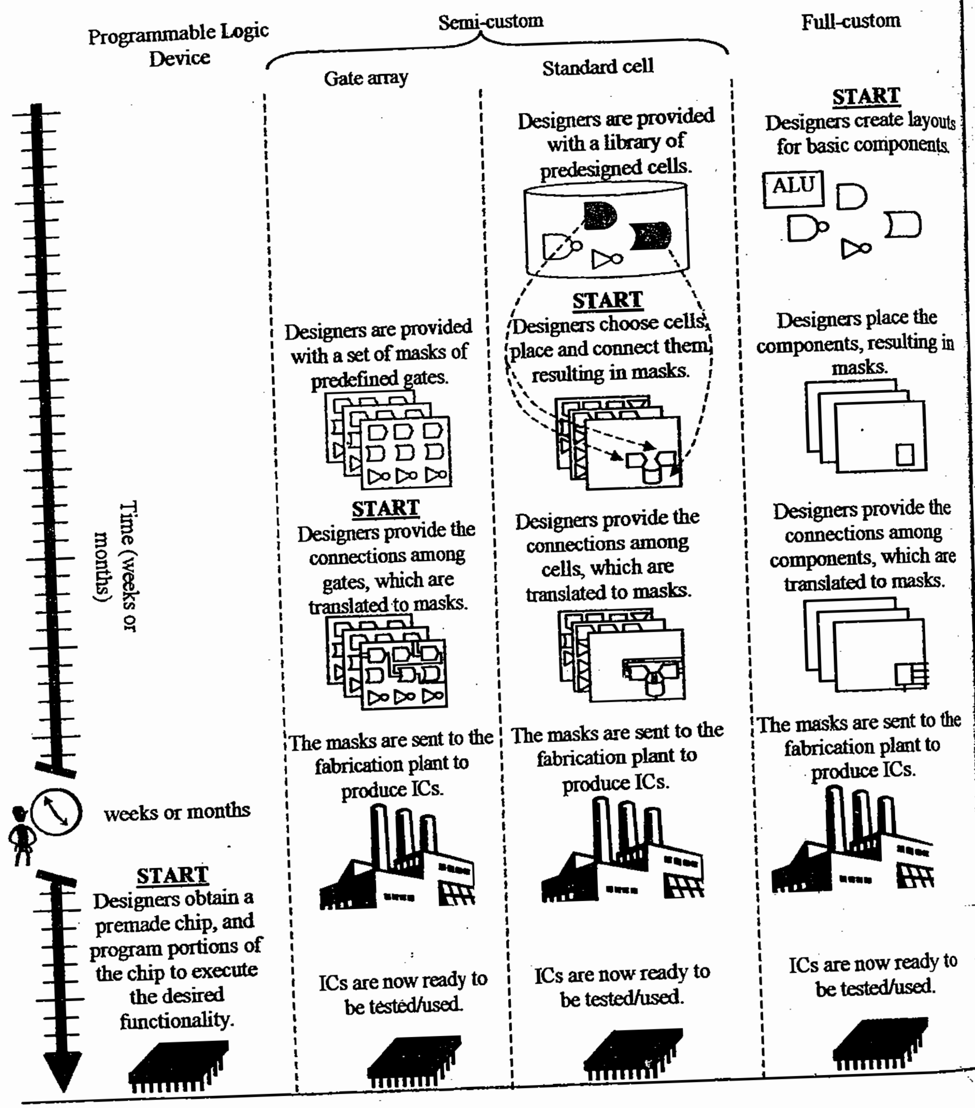
Programmable Logic Device
Semi-custom
Full-custom
Gate array
Standard cell
Time (weeks or months)
START
Designers are provided with a library of predesigned cells.
START
Designers create layouts for basic components.
ALU
Designers are provided with a set of masks of predefined gates.
START
Designers choose cells, place and connect them, resulting in masks.
Designers place the components, resulting in masks.
START
Designers provide the connections among gates, which are translated to masks.
Designers provide the connections among cells, which are translated to masks.
Designers provide the connections among components, which are translated to masks.
The masks are sent to the fabrication plant to produce ICs.
The masks are sent to the fabrication plant to produce ICs.
The masks are sent to the fabrication plant to produce ICs.
weeks or months
START
Designers obtain a premade chip, and program portions of the chip to execute the desired functionality.
ICs are now ready to be tested/used.
ICs are now ready to be tested/used.
ICs are now ready to be tested/used.

Figure 10.4: The three IC technologies.

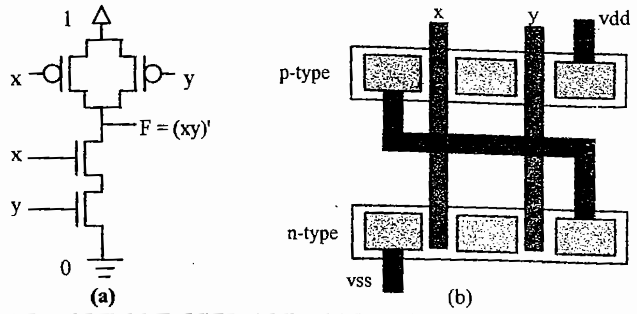

Figure 10.5: A more compact NAND circuit: (a) NAND circuit schematic, (b) compacted layout.

- *Placement*: the task of placing and orienting every transistor somewhere on the IC.
- *Routing*: the task of running wires between the transistors, without intersecting other wires or transistors.
- *Sizing*: the task of deciding how big each wire and transistor will be. Larger wires and transistors provide better performance but consume more power and require more silicon area.

A good layout is typically defined by characteristics like speed and size. Speed is the longest path from input to output, or from register to register, typically measured in nanoseconds. Size is the total silicon area necessary to implement the complete circuit. Both of these features are usually improved when the circuit is highly compacted, namely, when transistors that are connected are placed close together and hence their connecting wires are shorter. Consider for example the NAND layout of Figure 10.2(c). In that example, we did not pay attention to creating a compact layout. Figure 10.5(b) shows a compacted version of the NAND circuit. Notice how much less area is wasted in this compacted version. However, such compaction must obey certain *design rules*. For example, two transistors must be spaced apart a minimum distance lest they electrically interfere with one another.

In the past, many transistor circuits were converted by hand into compact layouts. Such *circuit design* was a common job. However, ICs can now hold so many transistors, numbering in the hundreds of millions, that laying out complete ICs by hand would require an absurd amount of time. Thus, hand layout is usually used only for relatively small, critical components, like the ALU of a microprocessor, or for basic components like logic gates that will be heavily reused.

Instead of hand layout, most layout today is done using automated layout tools, known as *physical design* tools. These tools typically include powerful optimization algorithms that run for hours or days seeking to improve the speed and size of a layout.

The advantages of full-custom IC technology include its excellent efficiency with respect to power, performance, and size. Interconnected transistors can be placed near each other and thus be connected by very short wires, yielding good performance and power. Furthermore, only those transistors necessary for the circuit being designed appear on the IC, resulting in no wasted area due to unused transistors.

The main disadvantages of full-custom IC technology are its high NRE cost and long time-to-market. These disadvantages stem from having to design a complete layout, which even with the aid of tools can be time-consuming and error-prone. Furthermore, masks for every IC layer must be created, increasing NRE cost and delaying time-to-market. In addition, errors discovered after manufacturing the IC are common, often requiring several respins.

10.3 Semi-Custom (ASIC) IC Technology

As mentioned above, creating a full-custom layout can be quite challenging. A designer using a semi-custom IC technology has this burden partially relieved, since rather than creating a full-custom layout, the designer connects pre-layed-out building blocks. The common name for such a semi-custom IC is an *application specific integrated circuit* (ASIC). The term *application specific* was likely chosen to contrast with general-purpose processor ICs, since for many years a processor was implemented as its own IC. ASICs in contrast implemented a circuit specific to a particular application (i.e., a single-purpose processor). Today, however, a single ASIC may implement a combination of general-purpose and single-purpose processors. Needless to say, there is much confusion related to use of the term *ASIC* today. Thus, we prefer the term *semi-custom IC*.

The two main types of semi-custom IC technologies are gate array and standard cell. With either type, the main advantages versus full-custom are reduced NRE cost and faster time-to-market, since less layout and mask creation must be performed. The main disadvantage is reduced performance, power, and size efficiency. However, relative to programmable IC technology (yet to be discussed), semi-custom is extremely efficient in terms of performance, power, and size. Because of its good efficiency coupled with reduced NRE costs, semi-custom is the most popular IC technology today.

Gate Array Semi-Custom IC Technology

In a gate array IC technology, all of the logic gates of the IC have already been layed out with their placement on the IC known, leaving the designer with the task of connecting the gates (routing) in a manner implementing the desired circuit. Note that gate here refers to a logic gate (e.g., AND, OR) rather than a terminal of a CMOS transistor. A simplified gate array layout is shown Figure 10.6(a).

Because the IC's gates are placed beforehand, many of them may go unused, since we may not need all instances of each type of gate in our particular circuit. Furthermore, routing wires between gates may be quite long since the gate placement was decided before knowing what connections would be made.

Standard Cell Semi-Custom IC Technology

In standard cell IC technology, common logic functions, or *cells*, have already been

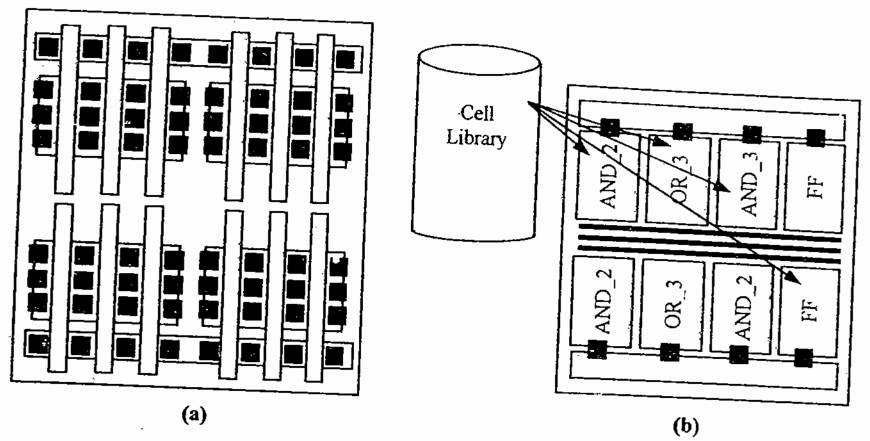

Figure 10.6: Semi-custom IC technology: (a) gate array, (b) standard cell.

compactly layed out. Examples of cells include a NAND gate, a NOR gate, a 2×1 multiplexor, and a combination of AND-OR-INVERT gates. The transistors within a cell are already layed out, but the placement of cells has not been determined. A designer thus must decide which cells to use, where to place them, and how to route among them. A standard cell layout is shown in Figure 10.6(b).

Standard cell therefore requires more NRE cost and longer time-to-market than gate array, since there is more layout remaining to be performed and all masks must still be made. However, NRE and time-to-market is still much less than full-custom, since the intricate layout within each cell is already completed. In addition, efficiency is very good compared to gate array, since only those cells needed are actually used, and their placement can be made so as to reduce interconnect. Furthermore, each cell may implement more complex functions than in gate arrays, leading to more compact designs.

A compromise between gate array and standard cell semi-custom ICs is known as a *cell array*, or cell-based array. A cell array is pretty much what we'd expect it to be based on its name. Cells, which you'll remember can be more complex than gates, have already been layed out, and have also already been placed. Thus, the designer need only connect the cells together.

10.4 Programmable Logic Device (PLD) IC Technology

The time required to manufacture an IC is measured in months, typically two to three months. While we may accept this time once we're ready to manufacture our final system, we probably can't wait so long to obtain a prototype of our system. Furthermore, the NRE cost to manufacture an IC (i.e., creating a layout and masks) may be too expensive to amortize over the number of ICs we plan to manufacture if that number is small. In addition, manufacturing an IC is risky, since we may discover after such manufacturing that an IC doesn't work properly in its target system, either due to manufacturing problems or due to an incorrect initial design. Thus, we never know how many respins will be necessary before we get a working IC; a recent study stated that the industry average was 3.5 spins. Therefore, we would like an IC technology that allows us to implement our system's structure on an IC, but that doesn't require us to manufacture that IC. Instead, we want an IC that we can program in the field, with the field being our lab or office. The term *program* here does not refer to writing software that executes on a microprocessor, but rather to configuring logic circuits and interconnection switches to implement a desired structural circuit.

Programmable logic device (PLD) technology satisfies this goal. A *PLD* is a pre-manufactured IC that we can purchase and then configure to implement our desired circuit.

An early example of a PLD was a programmable logic array (PLA), introduced in the early 1970s. A *PLA* was a small PLD with two levels of logic, a programmable AND array and a programmable OR array. Every PLA input and its complement was connected to every AND gate. So if a PLA had 10 inputs, every AND gate had 20 inputs. Any of these connections could be broken, meaning that each AND gate could generate any product term. Likewise, each OR gate could generate any sum of AND gate outputs. A PAL (programmable array logic) is another PLD type that eliminates the programmability of the OR array to reduce size and delay. PLAs and PALs are often referred to as simple PLDs, or *SPLDs*.

As IC capacity grew over the years, SPLDs could not simply be extended by adding more inputs, since the number of required connections to the AND array inputs would grow too high. Thus, the new capacity was taken advantage of instead by integrating numerous SPLDs on a single chip and adding programmable interconnect between them, resulting in what is known as a complex PLD, or *CPLD*. CPLDs often contain latches to enable implementation of sequential circuits also. Figure 10.7 illustrates a sample architecture for a CPLD. The top half of the figure is an SPLD that can implement any function of the chip's input signals as well as any SPLD output signal. The bottom half represents another identical SPLD. The array on the left consists of vertical lines that can be programmed to connect with any of the horizontal lines, so that any signal's true or complemented value can be fed into any gate. The output of each SPLD feeds into an IO cell. The IO cell can be programmed to pass the latched or unlatched, true or complemented, output to the CPLD's external output, and/or to the programmable array on the left as input to SPLDs.

While able to implement more complex circuits than SPLDs, CPLDs suffer from the problem of not scaling well as their sizes increase. For example, supposed the CPLD architecture of Figure 10.7 had 4 inputs and 2 outputs. Then there would be 6 signals in the programmable array, plus 6 more for those signals' complements, thus requiring 12-input AND gates. Likewise, suppose there were 12 inputs and 6 outputs. Then there would be 18 + 18 signals, requiring 36-input AND gates. Notice such an architecture doesn't scale well.

The logical solution is to build devices that are more modular in nature. In particular, there is no need to connect every input signal and every output signal to every AND gate. A more flexible approach can be used in which a subset of inputs and outputs are input to each SPLD. This more modular, more scalable approach to PLD design resulted in architectures

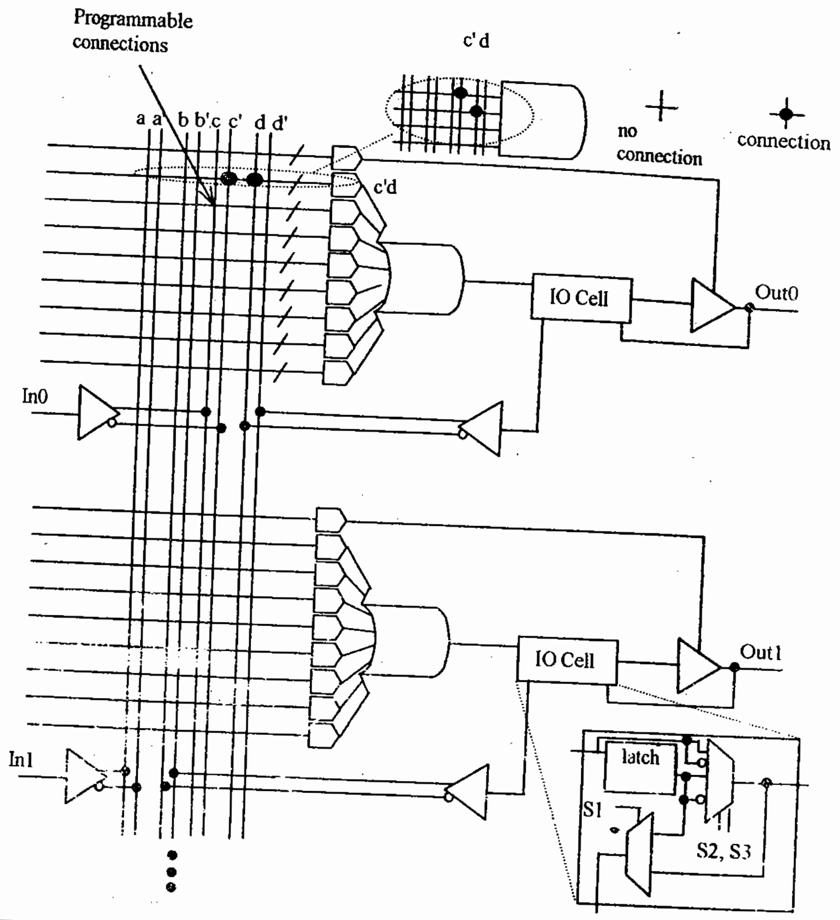

Figure 10.7: A CPLD architecture.

known as field-programmable gate arrays (FPGAs). An *FPGA* consists of arrays of programmable logic blocks connected by programmable interconnect blocks. The name FPGA was intended to contrast these devices with traditional gate arrays, which need masks to create the interconnections between the already layed-out gates. FPGAs in contrast, have their interconnections, as well as logic blocks programmed in the field, meaning in the designer's lab. However, most FPGA architectures do not have arrays of gates anywhere to be found, and thus the name FPGA can be somewhat misleading.

Programming is done by setting bits within the logic or interconnect blocks. Those bits are stored using nonvolatile (EPROM, EEPROM) or volatile (SRAM) memory technology. Another nonvolatile technique used in PLDs is an antifuse, which, as the name implies, behaves opposite to a fuse — an antifuse is originally an open circuit but takes on low resistance when programmed.

10.5 Summary

Creating an IC circuit layout and manufacturing an IC from this layout are complex, expensive, and time-consuming processes. Embedded systems designers can choose from different IC technologies, in order to reduce NRE cost and time-to-market by trading off with other design metrics, like size, performance, and power. Full-custom IC technology is the most expensive technology in terms of NRE cost and time-to-market but yields the most efficient circuits. Semi-custom IC technology, or ASICs, involve use of predesigned basic components, thus reducing NRE cost and time-to-market, but still providing good efficiency. Programmable logic devices are premanufactured and thus eliminate the need for the designer to go through the manufacturing stage, greatly reducing NRE cost and time-to-market, but are significantly inferior to custom or semi-custom IC in terms of size, power, performance, and unit cost. The designer may choose to use PLDs early in the design process, switching to ASICs and even full-custom later in the process when the design has stabilized.

10.6 References and Further Reading

- Sebastian Smith, M.J. *Application-Specific Integrated Circuits*. Reading, MA: Addison Wesley, 1997.

10.7 Exercises

10.1 Using the NAND gate (shown in figure Figure 10.1) as building block, (a) draw the circuit schematic for the function $F = xz + yz'$, and (b) draw the top-down view of the circuit on an IC (make your layout as compact as possible.)

10.2 Draw (a) the transistor-level circuit schematic for a two-input multiplexor, and (b) the top-down view of the circuit on an IC (make your layout as compact as possible.)

10.3 Implement the function $F = xz + yz'$ using the gate array structure given in Figure 10.6(a).

CHAPTER 11: *Design Technology*

11.1 Introduction

We have described how to design embedded systems from processors, memories, and interfaces, and in Chapter 10 we described the various IC technologies available for implementing such systems. Recall that in Chapter 1, we pointed out that IC transistor capacity is growing faster than the ability of designers to produce transistors in their designs. This difference in growth rate has resulted in the well-known productivity gap. Thus, there has been tremendous interest over the past few decades in developing design technologies that will enable designers to produce transistors more rapidly. These technologies have been developed for both software and for hardware, but the recent developments in hardware design technology deserves special attention since they've brought us to a new era in embedded system design.

Design is the task of defining a system's functionality and converting that functionality into a physical implementation, while satisfying certain constrained design metrics and optimizing other design metrics. Design is hard. Just getting the functionality right is tricky because embedded system functionality can be very complex, with millions of possible environment scenarios that must be responded to properly. For example, consider an elevator controller, and in particular the many possible combinations of buttons being pressed, the elevator moving, the doors being open, and so on. Not only is getting the functionality right