9

Programmable Logic Devices

Logic devices constitute one of the three important classes of devices used to build digital electronics systems, memory devices and microprocessors being the other two. Memory devices such as ROM and RAM are used to store information such as the software instructions of a program or the contents of a database, and microprocessors execute software instructions to perform a variety of functions, from running a word-processing program to carrying out far more complex tasks. Logic devices implement almost every other function that the system must perform, including device-to-device interfacing, data timing, control and display operations and so on. So far, we have discussed those logic devices that perform fixed logic functions decided upon at the manufacturing stage. Logic gates, multiplexers, demultiplexers, arithmetic circuits, etc., are some examples. Sequential logic devices such as flip-flops, counters, registers, etc., to be discussed in the following chapters, also belong to this category of logic devices. In the present chapter, we will discuss a new category of logic devices called *programmable logic devices* (PLDs). The function to be performed by a programmable logic device is undefined at the time of its manufacture. These devices are programmed by the user to perform a range of functions depending upon the logic capacity and other features offered by the device. We will begin with a comparison of fixed and programmable logic, and then follow this up with a detailed description of different types of PLDs in terms of operational fundamentals, salient features, architecture and typical applications. A brief introduction to the devices offered by some of the major manufacturers of PLDs and PLD programming languages is given towards the end of the chapter.

9.1 Fixed Logic Versus Programmable Logic

As outlined in the introduction, there are two broad categories of logic devices, namely fixed logic devices and programmable logic devices. Whereas a fixed logic device such as a logic gate or a multiplexer or a flip-flop performs a given logic function that is known at the time of device manufacture, a programmable logic device can be configured by the user to perform a large variety of

Digital Electronics: Principles, Devices and Applications Anil K. Maini
 ISBN: 978-0-470-03214-5

logic functions. In terms of the internal schematic arrangement of the two types of device, the circuits or building blocks and their interconnections in a fixed logic device are permanent and cannot be altered after the device is manufactured.

A *programmable logic device* offers to the user a wide range of logic capacity in terms of digital building blocks, which can be configured by the user to perform the intended function or set of functions. This configuration can be modified or altered any number of times by the user by reprogramming the device. Figure 9.1 shows a simple logic circuit comprising four three-input AND gates and a four-input OR gate. This circuit produces an output that is the sum output of a full adder. Here, A and B are the two bits to be added, and C is the carry-in bit. It is a fixed logic device as the circuit is unalterable from outside owing to fixed interconnections between the various building blocks.

Figure 9.2 shows the logic diagram of a simple programmable device. The device has an array of four six-input AND gates at the input and a four-input OR gate at the output. Each AND gate can handle three variables and thus can produce a product term of three variables. The three variables (A, B and C in this case) or their complements can be programmed to appear at the inputs of any of the four AND gates through fusible links called antifuses. This means that each AND gate can produce the desired three-variable product term. It may be mentioned here that an antifuse performs a function that is opposite to that performed by a conventional electrical fuse. A fuse has a low initial resistance and permanently breaks an electrically conducting path when current through it exceeds a certain limiting value. In the case of an antifuse, the initial resistance is very high and it is designed to create a low-resistance electrically conducting path when voltage across it exceeds a certain level. As a result, this circuit can be programmed to generate any three-variable sum-of-products Boolean function having four minterms by activating the desired fusible links. For example, the circuit could be programmed to produce the sum output resulting from the addition of three bits (the sum output in the case of a full adder) or to produce difference outputs resulting from subtraction of two bits with a borrow-in (the difference output in the case of a full subtractor).

We can visualize that the logic circuit of Fig. 9.2 has a programmable AND array at the input and a fixed OR gate at the output. Incidentally, this is the architecture of programmable logic devices called programmable array logic (PAL). Practical PAL devices have a much larger number of programmable AND gates and fixed OR gates to have enhanced logic capacity and performance capability. PAL devices are discussed in detail in the latter part of the chapter.

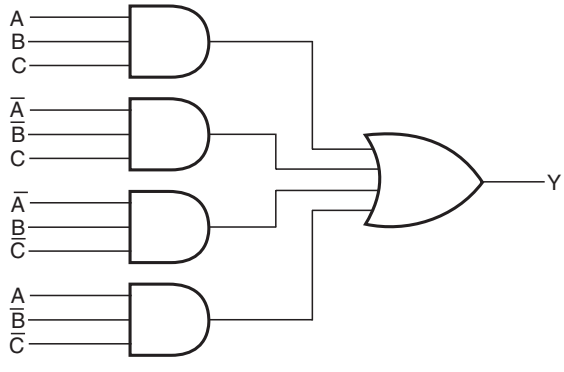


Figure 9.1 Fixed logic circuit.

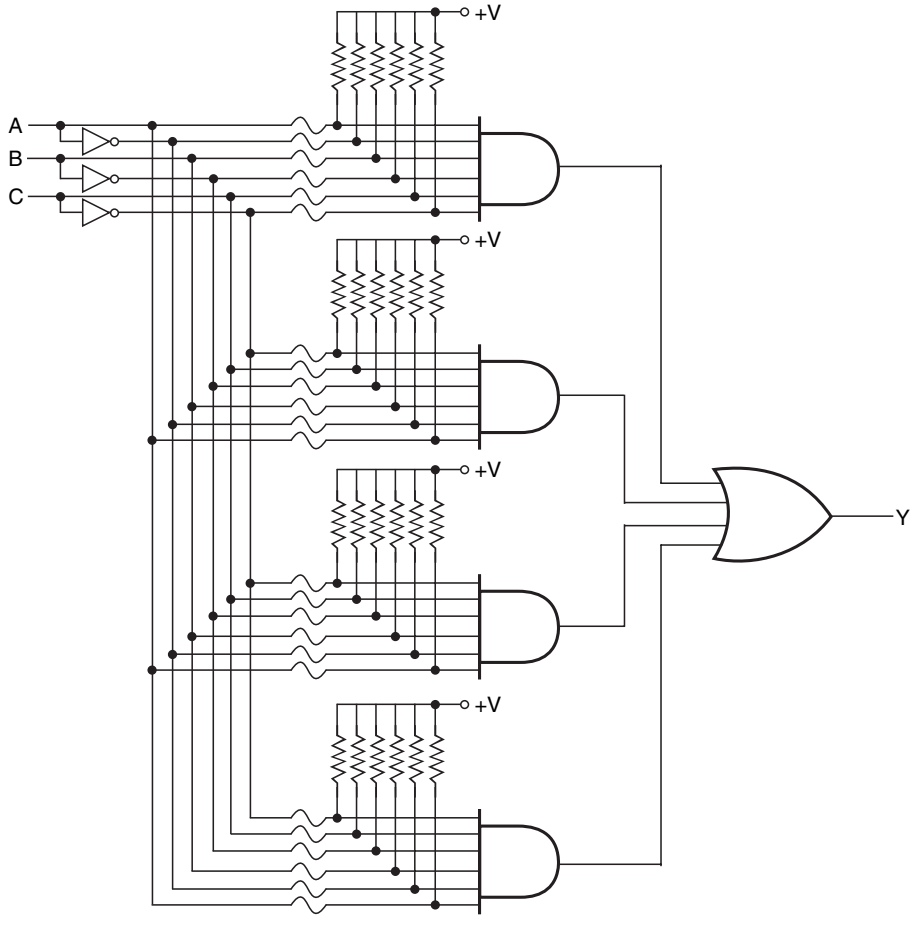


Figure 9.2 Simple programmable logic circuit.

9.1.1 Advantages and Disadvantages

1. If we want to build a fixed logic device to perform a certain specific function, the time required from design to the final stage when the manufactured device is actually available for use could easily be several months to a year or so. PLD-based design requires much less time from design cycle to production run.
2. In the case of fixed logic devices, the process of design validation followed by incorporation of changes, if any, involves substantial nonrecurring engineering (NRE) costs, which leads to an enhanced cost of the initial prototype device. In the case of PLDs, inexpensive software tools can be used for quick validation of designs. The programmable feature of these devices allows quick incorporation of changes and also a quick testing of the device in an actual application environment. In this case, the device used for prototyping is the same as the one that would qualify for use in the end equipment.

3. In the case of programmable logic devices, users can change the circuit as often as they want to until the design operates to their satisfaction. PLDs offer to the users much more flexibility during the design cycle. Design iterations are nothing but changes to the programming file.
4. Fixed logic devices have an edge for large-volume applications as they can be mass produced more economically. They are also the preferred choice in applications requiring the highest performance level.

9.2 Programmable Logic Devices – An Overview

There are many types of programmable logic device, distinguishable from one another in terms of architecture, logic capacity, programmability and certain other specific features. In this section, we will briefly discuss commonly used PLDs and their salient features. A detailed description of each of them will follow in subsequent sections.

9.2.1 Programmable ROMs

PROM (Programmable Read Only Memory) and EPROM (Erasable Programmable Read Only Memory) can be considered to be predecessors to PLDs. The architecture of a programmable ROM allows the user to hardware-implement an arbitrary combinational function of a given number of inputs. When used as a memory device, n inputs of the ROM (called address lines in this case) and m outputs (called data lines) can be used to store $2^n m$-bit words. When used as a PLD, it can be used to implement m different combinational functions, with each function being a chosen function of n variables. Any conceivable n-variable Boolean function can be made to appear at any of the m output lines. A generalized ROM device with n inputs and m outputs has 2^n hard-wired AND gates at the input and m programmable OR gates at the output. Each AND gate has n inputs, and each OR gate has 2^n inputs. Thus, each OR gate can be used to generate any conceivable Boolean function of n variables, and this generalized ROM can be used to produce m arbitrary n-variable Boolean functions. The AND array produces all possible minterms of a given number of input variables, and the programmable OR array allows only the desired minterms to appear at their inputs. Figure 9.3 shows the internal architecture of a PROM having four input lines, a hard-wired array of 16 AND gates and a programmable array of four OR gates. A cross ($\times$) indicates an intact (or unprogrammed) fusible link or interconnection, and a dot ($\bullet$) indicates a hard-wired interconnection. PROMs, EPROMs and EEPROMs (Electrically Erasable Programmable Read Only Memory) can be programmed using standard PROM programmers. One of the major disadvantages of PROMs is their inefficient use of logic capacity. It is not economical to use PROMs for all those applications where only a few minterms are needed. Other disadvantages include relatively higher power consumption and an inability to provide safe covers for asynchronous logic transitions. They are usually much slower than the dedicated logic circuits. Also, they cannot be used to implement sequential logic owing to the absence of flip-flops.

9.2.2 Programmable Logic Array

A *programmable logic array* (PLA) device has a programmable AND array at the input and a programmable OR array at the output, which makes it one of the most versatile PLDs. Its architecture differs from that of a PROM in the following respects. It has a programmable AND array rather than a hard-wired AND array. The number of AND gates in an m-input PROM is always equal to 2^m. In the case of a PLA, the number of AND gates in the programmable AND array for m input variables

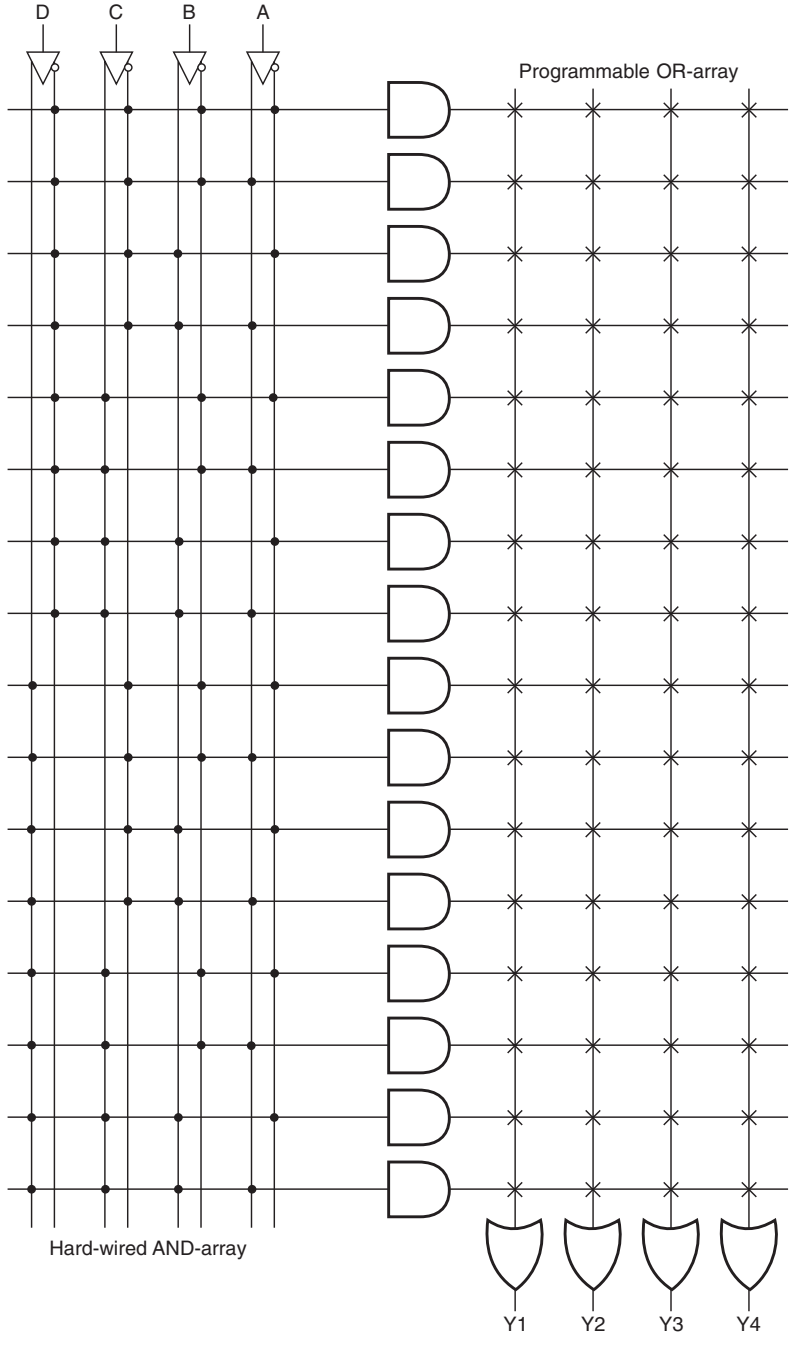


Figure 9.3 Internal architecture of a PROM.

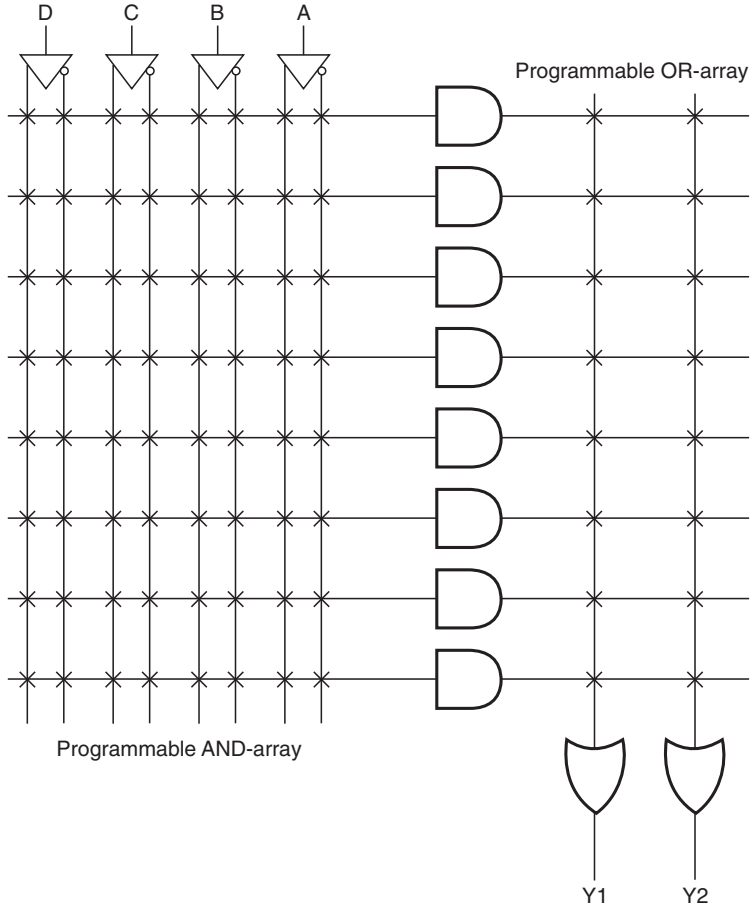


Figure 9.4 Internal architecture of a PLA device.

is usually much less than 2^m, and the number of inputs of each of the OR gates equals the number of AND gates. Each OR gate can generate an arbitrary Boolean function with a maximum of minterms equal to the number of AND gates. Figure 9.4 shows the internal architecture of a PLA device with four input lines, a programmable array of eight AND gates at the input and a programmable array of two OR gates at the output. A PLA device makes more efficient use of logic capacity than a PROM. However, it has its own disadvantages resulting from two sets of programmable fuses, which makes it relatively more difficult to manufacture, program and test.

9.2.3 Programmable Array Logic

Programmable array logic (PAL) architecture has a programmable AND array at the input and a fixed OR array at the output. The programmable AND array of a PAL device is similar to that of a PLA device. That is, the number of programmable AND gates is usually smaller than the number required

to generate all possible minterms of the given number of input variables. The OR array is fixed and the AND outputs are equally divided between available OR gates. For instance, a practical PAL device may have eight input variables, 64 programmable AND gates and four fixed OR gates, with each OR gate having 16 inputs. That is, each OR gate is fed from 16 of the 64 AND outputs. Figure 9.5 shows the internal architecture of a PAL device that has four input lines, an array of eight AND gates at the input and two OR gates at the output, to introduce readers to the arrangement of various building blocks inside a PAL device and allow them a comparison between different programmable logic devices.

9.2.4 Generic Array Logic

A *generic array logic* (GAL) device is similar to a PAL device and was invented by Lattice Semiconductor. It differs from a PAL device in that the programmable AND array of

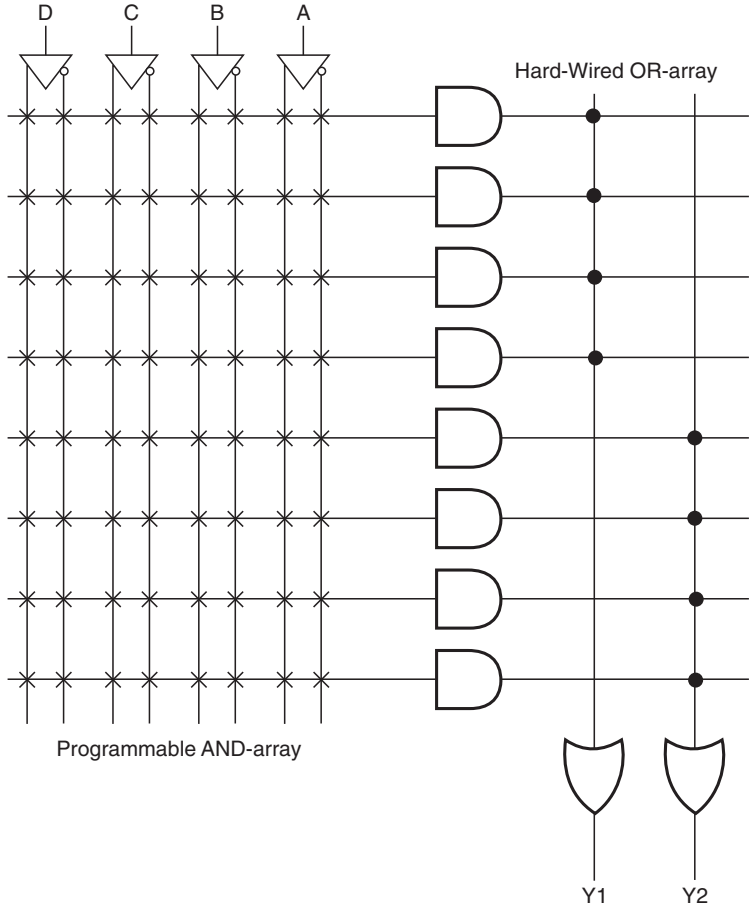


Figure 9.5 Internal architecture of a PAL device.

a GAL device can be erased and reprogrammed. Also, it has reprogrammable output logic. This feature makes it particularly attractive at the device prototyping stage, as any bugs in the logic can be corrected by reprogramming. A similar device called PEEL (Programmable Electrically Erasable Logic) was introduced by the International CMOS Technology (ICT) Corporation.

9.2.5 Complex Programmable Logic Device

Programmable logic devices such as PLAs, PALs, GALs and other PAL-like devices are often grouped into a single category called *simple programmable logic devices* (SPLDs) to distinguish them from the ones that are far more complex. A *complex programmable logic device* (CPLD), as the name suggests, is a much more complex device than any of the programmable logic devices discussed so far. A CPLD may contain circuitry equivalent to that of several PAL devices linked to each other by programmable interconnections. Figure 9.6 shows the internal structure of a typical CPLD. Each of the four logic blocks is equivalent to a PLD such as a PAL device. The number of logic blocks in a CPLD could be more or less than four. Each of the logic blocks has programmable interconnections. A switch matrix is used for logic block to logic block interconnections. Also, the switch matrix in a CPLD may or may not be fully connected. That is, some of the possible connections between logic block outputs and inputs may not be supported by a given CPLD. While the complexity of a typical PAL device may be of the order of a few hundred logic gates, a CPLD may have a complexity equivalent to tens of thousands of logic gates. When compared with FPGAs, CPLDs offer predictable timing characteristics owing to their less flexible internal architecture and are thus ideal for critical control applications and other applications where a high performance level is required. Also, because of their relatively much lower power consumption and lower cost, CPLDs are an ideal solution for battery-operated portable applications such as mobile phones, digital assistants and so on. A CPLD can be programmed either by using a PAL programmer or by feeding it with a serial data stream from a PC after soldering it on the PC board. A circuit on the CPLD decodes the data stream and configures it to perform the intended logic function.

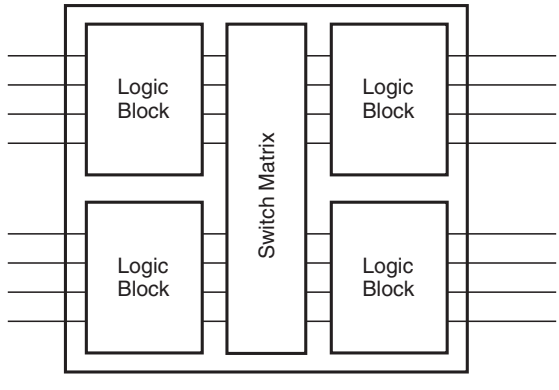


Figure 9.6 CPLD architecture.

9.2.6 Field-Programmable Gate Array

A *field-programmable gate array* (FPGA) uses an array of logic blocks, which can be configured by the user. The term 'field-programmable' here signifies that the device is programmable outside the factory where it is manufactured. The internal architecture of an FPGA device has three main parts, namely the array of logic blocks, the programmable interconnects and the I/O blocks. Figure 9.7 shows the architecture of a typical FPGA. Each of the I/O blocks provides an individually selectable input, output or bidirectional access to one of the general-purpose I/O pins on the FPGA package. The logic blocks in an FPGA are no more complex than a couple of logic gates or a look-up table feeding a flip-flop. The programmable interconnects connect logic blocks to logic blocks and also I/O blocks to logic blocks.

FPGAs offer a much higher logic density and much larger performance features compared with CPLDs. Some of the contemporary FPGA devices offer a logic complexity equivalent to that of eight million system gates. Also, these devices offer features such as built-in hard-wired processors,

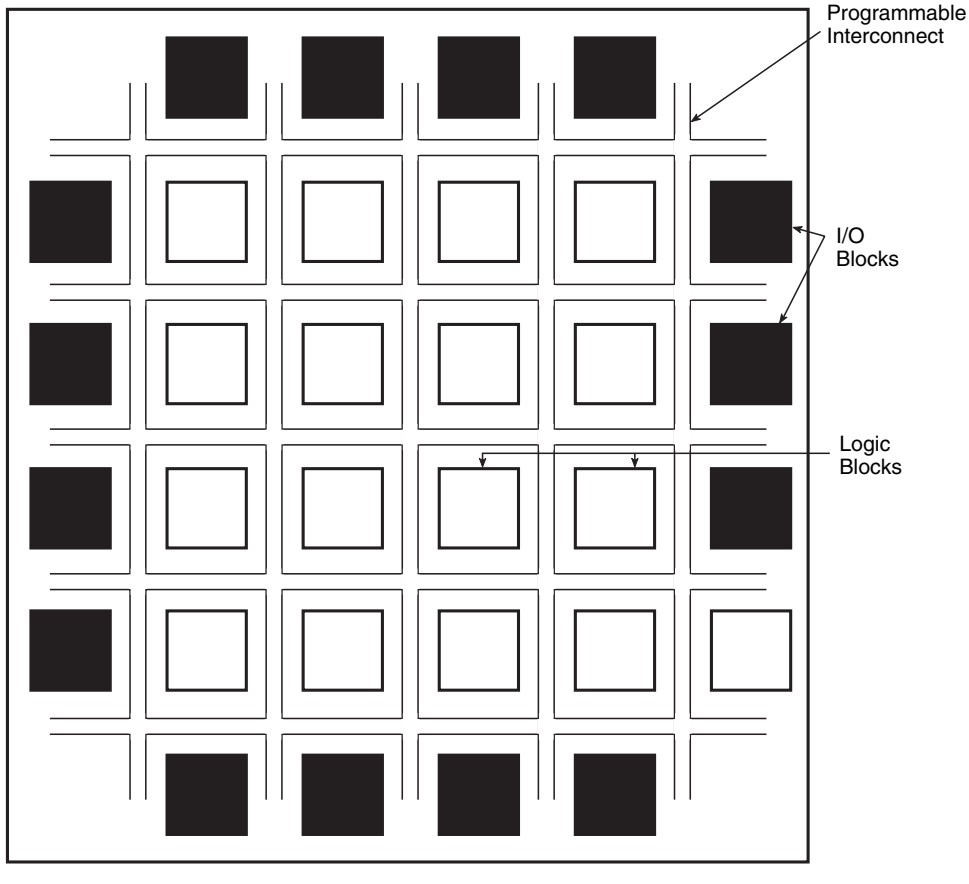


Figure 9.7 FPGA architecture.

large memory, clock management systems and support for many of the contemporary device-to-device signalling technologies. FPGAs find extensive use in a variety of applications, which include data processing and storage, digital signal processing, instrumentation and telecommunications.

FPGAs are also programmed like CPLDs after they are soldered onto the PC board. In the case of FPGAs, the programmed configuration is usually volatile and therefore needs to be reloaded whenever power is applied or a different functionality is required.

9.3 Programmable ROMs

A *read only memory* (ROM) is essentially a memory device that can be used to store a certain fixed set of binary information. As outlined earlier, these devices have certain inherent links that can be made or broken depending upon the type of fusible link to store any user-specified binary information in the device. While, in the case of a conventional fusible link, relevant interconnections are broken to program the device, in the case of an antifuse the relevant interconnections are made to do the same job. This is illustrated in Fig. 9.8. Figure 9.8(a) shows the internal logic diagram of a 4×2 PROM. The figure shows an unprogrammed PROM. Figures 9.8(b) and (c) respectively show the use of a fuse and an antifuse to produce output-1 = AB. Note that in the case of a fuse an unprogrammed interconnection is a 'make' connection, whereas in the case of an antifuse it is a 'break' connection.

Once a given pattern is formed, it remains as such even if power is turned off and on. In the case of PROMs, the user can erase the data already stored on the ROM chip and load it with fresh data. Memory-related issues of ROMs are discussed in detail in Chapter 15 on microcomputer fundamentals. In the present section, we will discuss the use of a PROM as a programmable logic device for implementation of combinational logic functions, which is one of the most widely exploited applications of PROMs. A PROM in general has n input lines and m output lines and is designated as a $2^n \times m$ PROM. Looking at the internal architecture of a PROM device, it is a combinational circuit with the AND gates wired as a decoder and having OR gates equal to the number of outputs. A PROM with five input lines and four output lines, for instance, would have the equivalent of a 5×32 decoder at the input that would generate 32 possible minterms or product terms. Each of these four OR gates would be a 32-input gate fed from 32 outputs of the decoder through fusible links.

Figure 9.9 shows the internal architecture of a 32×4 PROM. We can see that the input side is hard-wired to produce all possible 32 product terms corresponding to five variables. All 32 product terms or minterms are available at the inputs of each of the OR gates through programmable interconnections. This allows the users to have four different five-variable Boolean functions of their choice. Very complex combinational functions can be generated with PROMs by suitably making or breaking these links.

To sum up, for implementing an n-input or n-variable, $m-$output combinational circuit, one would need a $2^n \times m$ PROM. As an illustration, let us see how a PROM can be used to implement the following Boolean function with two outputs given by the equations

$$F_1(A, B, C) = \Sigma 0, 2 \tag{9.1}$$

$$F_2(A, B, C) = \Sigma 1, 4, 7 \tag{9.2}$$

Implementation of this Boolean function would require an 8×2 PROM. The internal logic diagram of the PROM in this case, after it is programmed, would be as shown in Fig. 9.10. Note that, in the programmed PROM of Fig. 9.10, an unprogrammed interconnection indicated by a cross ($\times$) is a 'make' connection.

It may be mentioned here that in practice a PROM would not be used to implement as simple a Boolean function as that illustrated above. The purpose here is to indicate to readers how a PROM

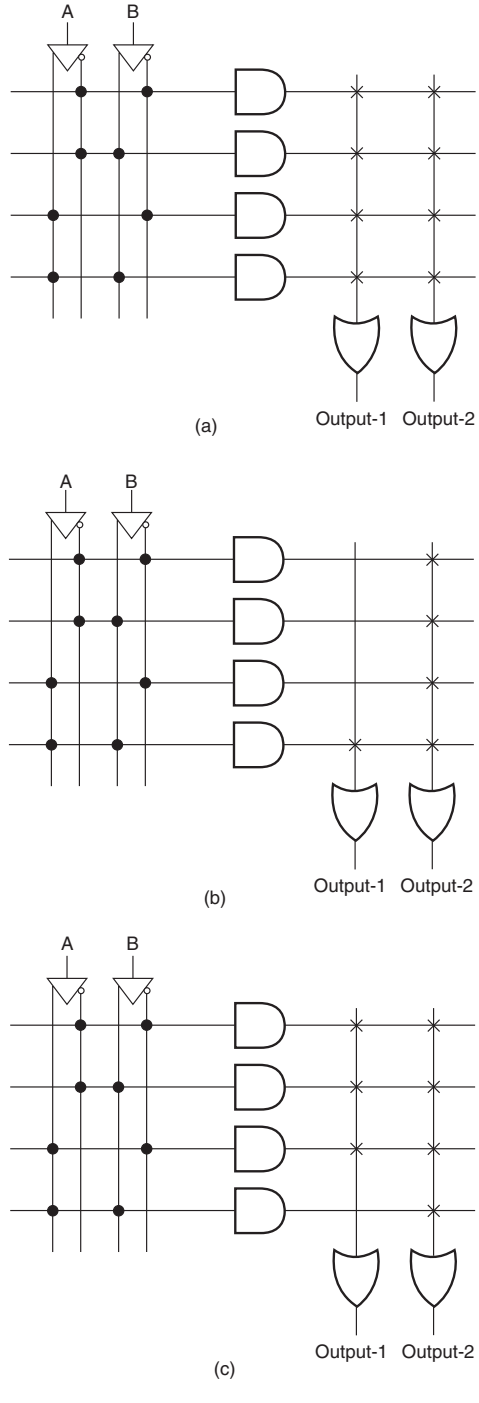


Figure 9.8 Use of fuse and antifuse.

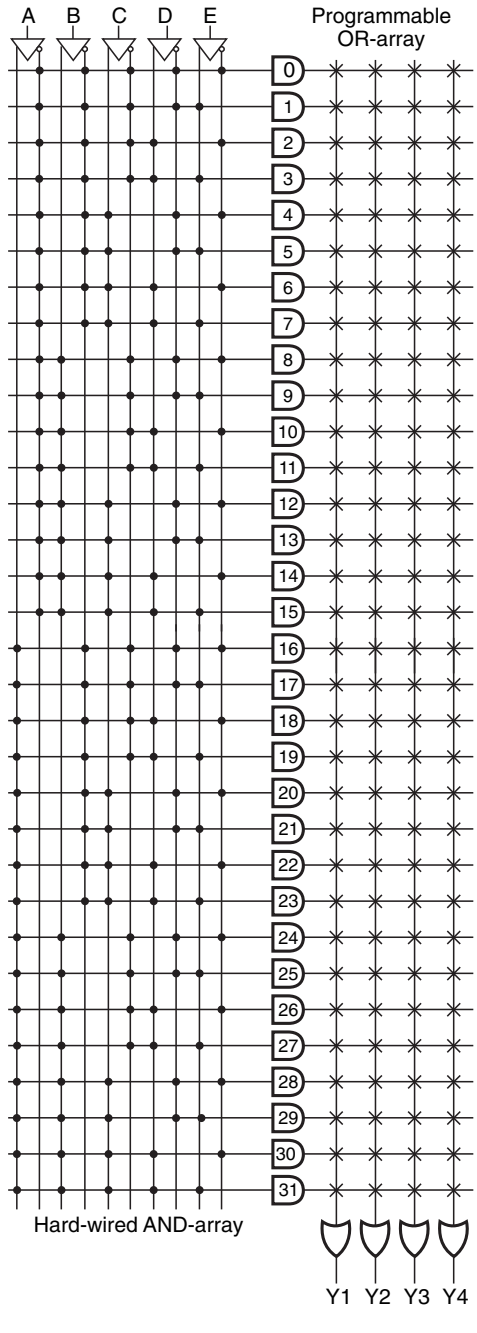


Figure 9.9 Internal architecture of a 32×4 PROM.

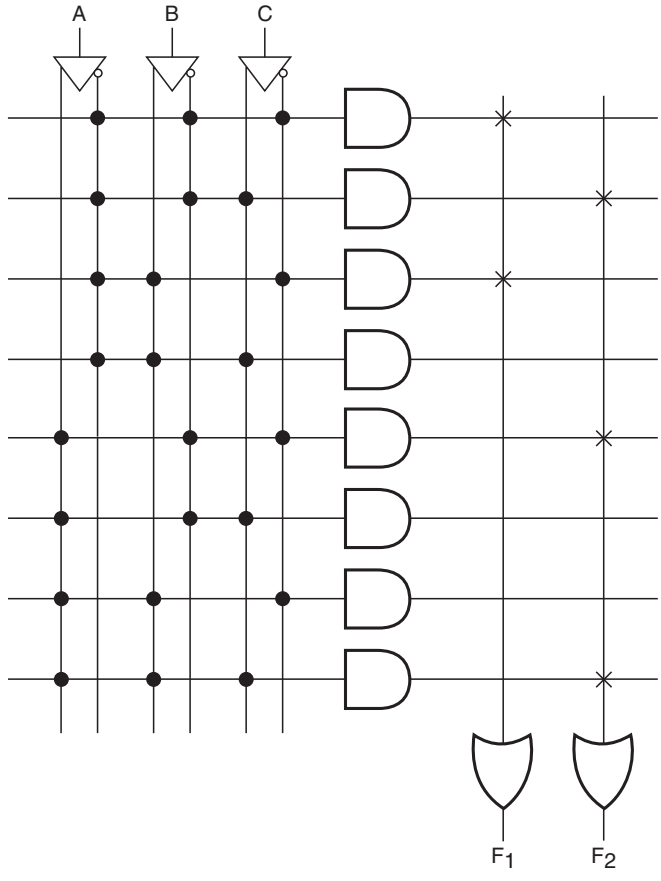


Figure 9.10 8×2 PROM internal logic diagram to implement given Boolean function.

implements a Boolean function. In actual practice, PROMs would be used only in the case of very complex Boolean functions.

Another noteworthy point is that, when it comes to implementing Boolean functions with PROMs, it is not economical to use PROM for those Boolean functions that have a large number of 'don't care' conditions. In the case of a PROM, each 'don't care' condition would have either all 0s or all ls. In other words, the space on the chip is not optimally utilized. Other programmable logic devices such as a PLA or PAL are more suitable in such situations.

Example 9.1

Determine the size of the PROM required for implementing the following logic circuits:

(a) a binary multiplier that multiplies two four-bit numbers;
(b) a dual 8-to-1 multiplexer with common selection inputs;
(c) a single-digit BCD adder/subtractor with a control input for selection of operation.

Solution

(a) The number of inputs required here would be eight. The result of multiplication would be in eight bits. Therefore, the size of the PROM $= 2^8 \times 8 = 256 \times 8$.

(b) The number of inputs $= 8 + 8 + 3 = 19$ (the number of selection inputs $= 3$). The number of outputs $= 2$. Therefore, the size of the PROM $= 2^{19} \times 2 = 512\text{K} \times 2$.

(c) The number of inputs $= 4$ (augend bits) $+ 4$ (addend bits) $+ 1$ (carry-in) $+ 1$ (control input) $= 10$. The number of outputs $= 4$ (sum or subtraction output bits) $+ 1$ (carry or borrow bit) $= 5$. The size of the PROM $= 2^{10} \times 5 = 1024 \times 5 = 1\text{K} \times 5$.

9.4 Programmable Logic Array

A *programmable logic array* (PLA) enables logic functions expressed in sum-of-products form to be implemented directly. It is similar in concept to a PROM. However, unlike a PROM, the PLA does not provide full decoding of the input variables and does not generate all possible minterms. While a PROM has a fixed AND gate array at the input and a programmable OR gate array at the output, a PLA device has a programmable AND gate array at the input and a programmable OR gate array at the output. In a PLA device, each of the product terms of the given Boolean function is generated by an AND gate which can be programmed to form the AND of any subset of inputs or their complements. The product terms so produced can be summed up in an array of programmable OR gates. Thus, we have a programmable OR gate array at the output. The input and output gates are constructed in the form of arrays with input lines orthogonal to product lines and product lines orthogonal to output lines.

Figure 9.11 shows the internal architecture of a PLA device with four input lines, eight product lines and four output lines. That is, the programmable AND gate array has eight AND gates. Each of the AND gates here has eight inputs, corresponding to four input variables and their complements. The input to each of the AND gates can be programmed to be any of the possible 16 combinations of four input variables and their complements. Four OR gates at the output can generate four different Boolean functions, each having a maximum of eight minterms out of 16 minterms possible with four variables. The logic diagram depicts the unprogrammed state of the device. The internal architecture shown in Fig. 9.11 can also be represented by the schematic form of Fig. 9.12. PLAs usually have inverters at the output of OR gates to enable them to implement a given Boolean function in either AND-OR or AND-OR-INVERT form.

Figure 9.13 shows a generalized block schematic representation of a PLA device having n inputs, m outputs and k product terms, with n, m and k respectively representing the number of input variables, the number of OR gates and the number of AND gates. The number of inputs to each OR gate and each AND gate are k and $2n$ respectively.

A PLA is specified in terms of the number of inputs, the number of product terms and the number of outputs. As is clear from the description given in the preceding paragraph, the PLA would have a total of $2\text{K}n + \text{K}m$ programmable interconnections. A ROM with the same number of input and output lines would have $2^n \times \text{m}$ programmable interconnections.

A PLA could be either mask programmable or field programmable. In the case of a mask-programmable PLA, the customer submits a program table to the manufacturer to produce a custom-made PLA having the desired internal paths between inputs and outputs. A *field-programmable logic array* (FPLA) is programmed by the users themselves by means of a hardware programmer unit available commercially.

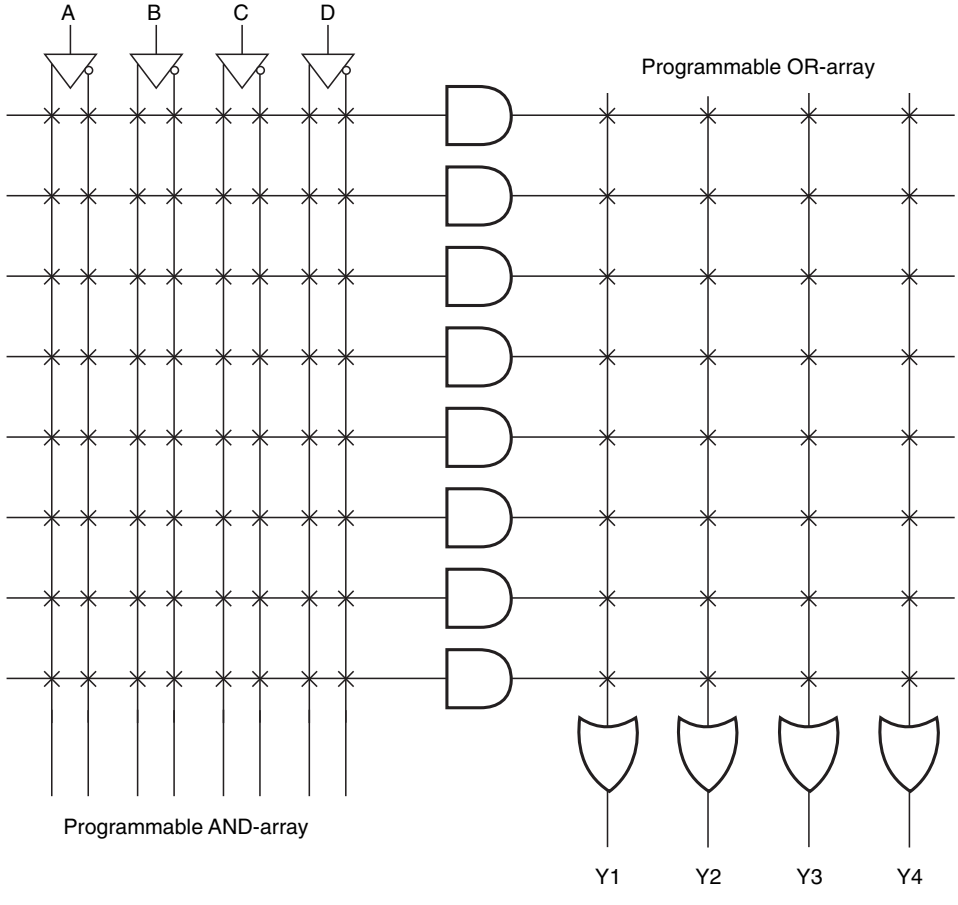


Figure 9.11 Internal architecture of a PLA device.

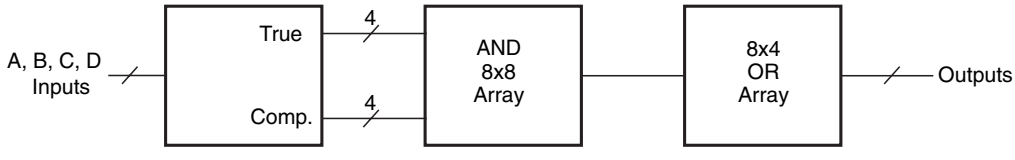


Figure 9.12 Alternative representation of PLA architecture.

While implementing a given Boolean function with a PLA, it is important that each expression is simplified to a minimum number of product terms which would minimize the number of AND gates required for the purpose. Since all input variables are available to different AND gates, simplification of Boolean functions to reduce the number of literals in various product terms is not important. In fact,

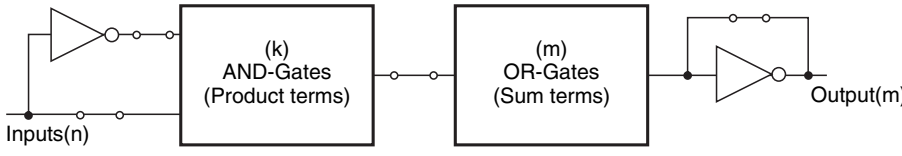


Figure 9.13 Generalized representation of PLA architecture.

each of the Boolean functions and their complements should be simplified. What is desirable is to have fewer product terms and product terms that are common to other functions. We would recall that PLAs offer the flexibility of implementing Boolean functions in both AND-OR and AND-OR-INVERT forms.

Example 9.2

Show the logic arrangement of both a PROM and a PLA required to implement a binary full adder.

Solution

The truth table of a full adder is given in Table 9.1. The Boolean expressions for sum S and carry-out C_o can be written as follows:

$$S = \Sigma 1, 2, 4, 7 \tag{9.3}$$

$$C_o = \Sigma 3, 5, 6, 7 \tag{9.4}$$

Figure 9.14 shows the implementation with an 8×2 PROM.

If we simplify the Boolean expressions for the sum and carry outputs, we will find that the expression for the sum output cannot be simplified any further, and also that the expression for carry-out can be simplified to three product terms with fewer literals. If we examine even the existing expressions, we find that we would need seven AND gates in the PLA implementation. And if we use the simplified expressions, even then we would require the same number of AND gates. Therefore, the simplification here would not help as far as its implementation with a PLA is concerned. Figure 9.15 shows the implementation of a full adder with a PLA device.

Table 9.1 Truth table for example 9.2.

A	B	Carry-in (C_i)	Sum (S)	Carry-out (C_o)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

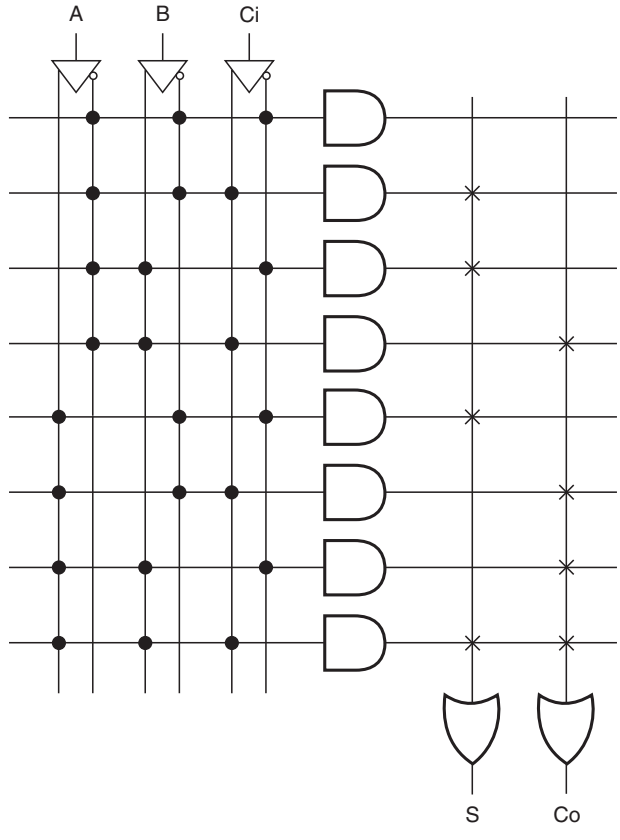


Figure 9.14 Solution to problem 9.2 using a PROM.

Example 9.3

We have two two-bit binary numbers A_1A_0 and B_1B_0. Design a PLA device to implement a magnitude comparator to produce outputs for A_1A_0 being 'equal to', 'not equal to', 'less than' and 'greater than' B_1B_0.

Solution

Table 9.2 shows the function table with inputs and desired outputs. The Boolean expressions for the desired outputs are given in the following equations:

$$\text{Output 1(equal to)} = \overline{A_1}.\overline{A_0}.\overline{B_1}.\overline{B_0} + \overline{A_1}.A_0.\overline{B_1}.B_0 + A_1.A_0.B_1.B_0 + A_1.\overline{A_0}.B_1.\overline{B_0} \tag{9.5}$$

Output 2 (not equal to)

$$\begin{aligned} &= \overline{A_1}.\overline{A_0}.\overline{B_1}.B_0 + \overline{A_1}.\overline{A_0}.B_1.\overline{B_0} + \overline{A_1}.\overline{A_0}.B_1.B_0 + \overline{A_1}.A_0.\overline{B_1}.\overline{B_0} + \overline{A_1}.A_0.B_1.\overline{B_0} + \overline{A_1}.A_0.B_1.B_0 + A_1.\overline{A_0}.\overline{B_1}.\overline{B_0} \\ &\quad + A_1.\overline{A_0}.\overline{B_1}.B_0 + A_1.\overline{A_0}.B_1.B_0 + A_1.A_0.\overline{B_1}.\overline{B_0} + A_1.A_0.\overline{B_1}.B_0 + A_1.A_0.B_1.\overline{B_0} \end{aligned} \tag{9.6}$$

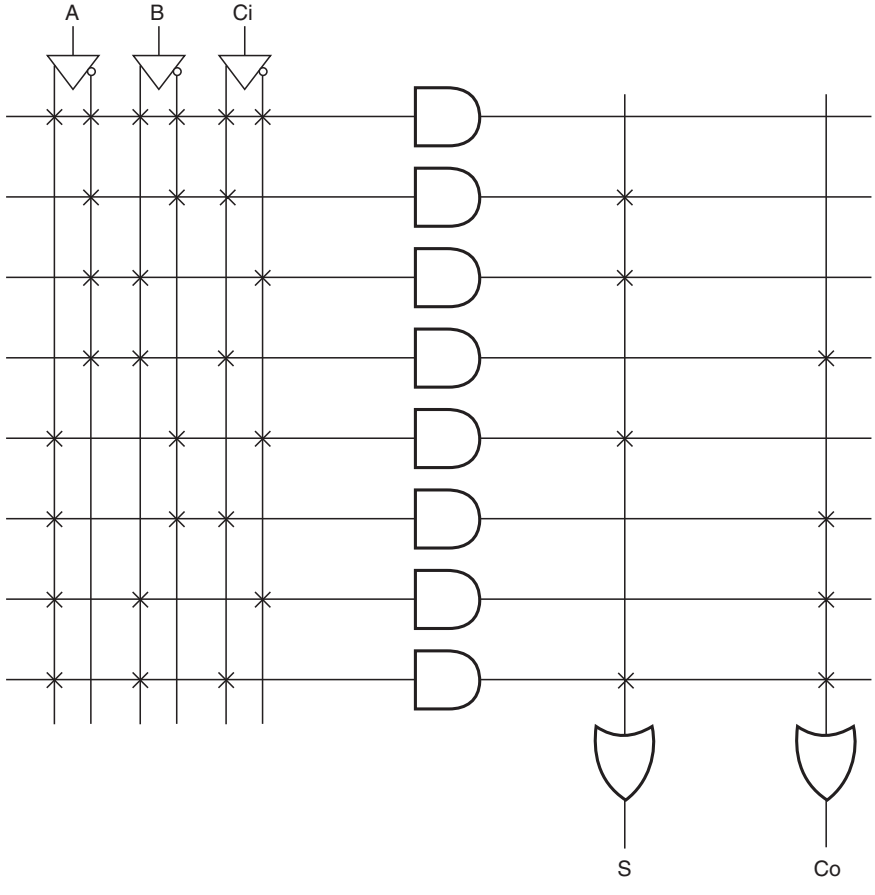


Figure 9.15 Solution to problem 9.2 using a PLA.

Output 3 (less than)

$$= \overline{A_1}.\overline{A_0}.\overline{B_1}.B_0 + \overline{A_1}.\overline{A_0}.B_1.\overline{B_0} + \overline{A_1}.\overline{A_0}.B_1.B_0 + \overline{A_1}.A_0.B_1.\overline{B_0} + \overline{A_1}.A_0.B_1.B_0 + A_1.\overline{A_0}.B_1.B_0 \quad (9.7)$$

Output 4 (greater than)

$$= \overline{A_1}.A_0.\overline{B_1}.\overline{B_0} + A_1.\overline{A_0}.\overline{B_1}.\overline{B_0} + A_1.\overline{A_0}.\overline{B_1}.B_0 + A_1.A_0.\overline{B_1}.\overline{B_0} + A_1.A_0.\overline{B_1}.B_0 + A_1.A_0.B_1.\overline{B_0} \quad (9.8)$$

Figures 9.16(a) to (d) show the Karnaugh maps for the four outputs. The minimized Boolean expressions can be written from the Karnaugh maps as follows:

$$\text{Output 1(equal to)} = \overline{A_1}.\overline{A_0}.\overline{B_1}.\overline{B_0} + \overline{A_1}.A_0.\overline{B_1}.B_0 + A_1.A_0.B_1.B_0 + A_1.\overline{A_0}.B_1.\overline{B_0} \quad (9.9)$$

$$\text{Output 2(not equal to)} = \overline{A_1}.B_1 + A_1.\overline{B_1} + \overline{A_0}.B_0 + A_0.\overline{B_0} \quad (9.10)$$

Table 9.2 Function table for example 9.3.

A_1	A_0	B_1	B_0	Output 1	Output 2	Output 3	Output 4
0	0	0	0	1	0	0	0
0	0	0	1	0	1	1	0
0	0	1	0	0	1	1	0
0	0	1	1	0	1	1	0
0	1	0	0	0	1	0	1
0	1	0	1	1	0	0	0
0	1	1	0	0	1	1	0
0	1	1	1	0	1	1	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	0	1
1	0	1	0	1	0	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	1	0	1
1	1	0	1	0	1	0	1
1	1	1	0	0	1	0	1
1	1	1	1	1	0	0	0

$$\text{Output 3(less than)} = \overline{A_1}.B_1 + \overline{A_1}.\overline{A_0}.B_0 + \overline{A_0}.B_1.B_0 \tag{9.11}$$

$$\text{Output 4(Greater than)} = A_1.\overline{B_1} + A_1.A_0.\overline{B_0} + A_0.\overline{B_1}.\overline{B_0} \tag{9.12}$$

Examination of minimized Boolean expressions (9.9) to (9.12) reveals that there are 12 different product terms to be accounted for. Therefore, a PLA device with 12 AND gates will meet the requirement. Also, since there are four outputs, we need to have four OR gates at the output. Figure 9.17 shows the programmed PLA device. Note that, in the programmed PLA device, an unprogrammed interconnection indicated by a cross ($\times$) is a 'make' connection.

9.5 Programmable Array Logic

The *programmable array logic* (PAL) device is a variant of the PLA device. As outlined in Section 9.2, it has a programmable AND gate array at the input and a fixed OR gate array at the output. The idea to have a fixed OR gate array at the output and make the device less complex originated from the fact that there were many applications where the product-term sharing capability of the PLA was not fully utilized and thus wasted. The PAL device is a trademark of Advanced Micro Devices Inc. PAL devices are however less flexible than PLA devices. The flexibility of a PAL device can be enhanced by having different output logic configurations including the availability of both OR (also called active HIGH) and NOR (also called active LOW) outputs and bidirectional pins that can act both as inputs and outputs, having clocked flip-flops at the outputs to provide what is called registered outputs. These features allow the device to be used in a wider range of applications than would be possible with a device with fixed input and output allocations. The mask-programmed version of PAL is known as the HAL (Hard Array Logic) device. A HAL device is pin-to-pin compatible with its PAL counterpart.

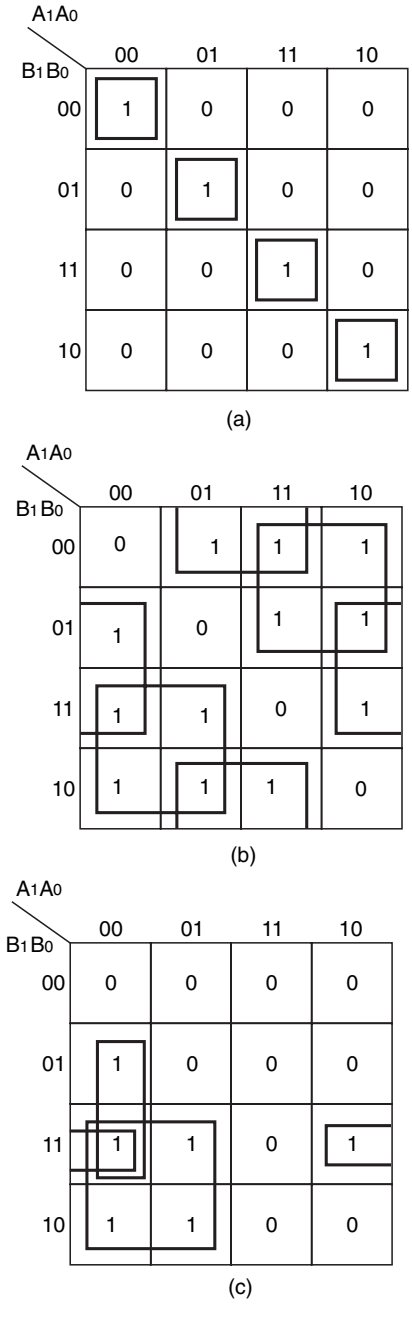


Figure 9.16 Karnaugh maps (example 9.3).

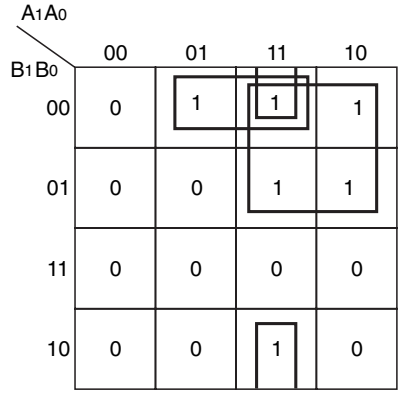


Figure 9.16 (*continued*).

9.5.1 PAL Architecture

Figure 9.18 shows the block schematic representation of the generalized architecture of a PAL device. As we can see from the arrangement shown, the device has a programmable AND gate array that is fed with various input variables and their complements. Programmable input connections allow any of the input variables or their complements to appear at the inputs of any of the AND gates in the array. Each of the AND gates generates a minterm of a user-defined combination of input variables and their complements. As an illustration, Fig. 9.19 gives an example of the generation of minterms.

Outputs from the programmable AND array feed an array of hard-wired OR gates. Here, the output of each of the AND gates does not feed the input of each of the OR gates. Each OR gate is fed from a subset of AND gates in the array. This implies that the sum-of-product Boolean functions generated by each of the OR gates at the output will have only a restricted number of minterms depending upon the number of AND gates from which it is being fed. Outputs from the PAL device, as is clear from the generalized form of representation shown in Fig. 9.18, are available both as OR outputs as well as complemented (or NOR) outputs.

Practical PAL devices offer various output logic arrangements. One of them, of course, is the availability of both OR and NOR outputs as mentioned in the previous paragraph. Another feature available with many PAL devices is that of registered outputs. In the case of registered outputs, the OR gate output drives the D-input of a *D*-type flip-flop, which is loaded with the data on either the LOW-to-HIGH or the HIGH-to-LOW edge of a clock signal. Yet another feature is the availability of bidirectional pins, which can be used both as outputs and inputs. This facility allows the user to feed a product term back to the programmable AND array. It helps particularly in those multi-output function logic circuits that share some common minterms. Some of the common output logic arrangements available with PAL devices are shown in Fig. 9.20.

Some PAL devices offer an EX-OR gate following the OR gate at each output. One of the inputs to the EX-OR gate is programmable, which allows the user to configure it as either an inverter or a noninverting buffer or as a two-input EX-OR gate. This feature is particularly useful while implementing parity and arithmetic operations.

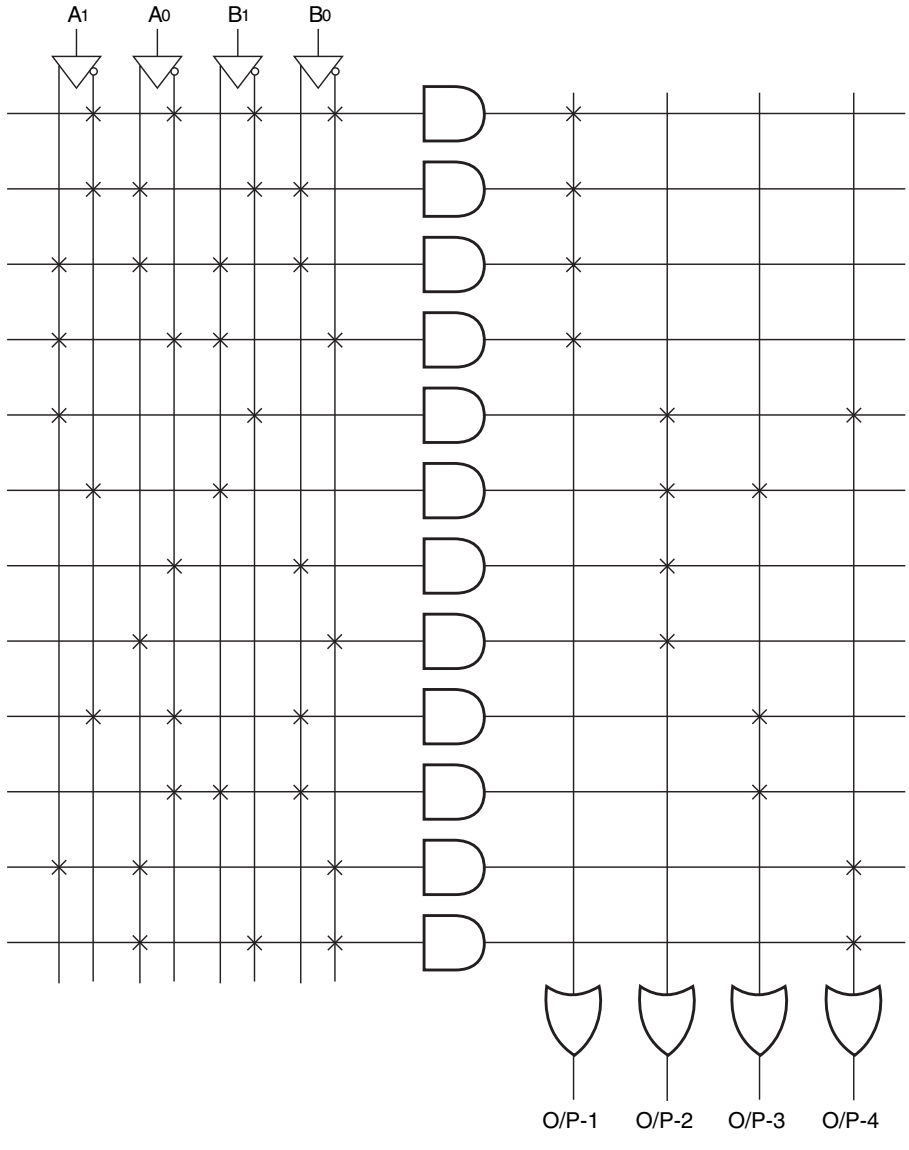


Figure 9.17 Programmed PLA device (example 9.3).

9.5.2 PAL Numbering System

The standard PAL numbering system uses an alphanumeric designation comprising a two-digit number indicating the number of inputs followed by a letter that tells about the architecture/type of logic output. Table 9.3 gives an interpretation of different letter designations in use. Another number following the

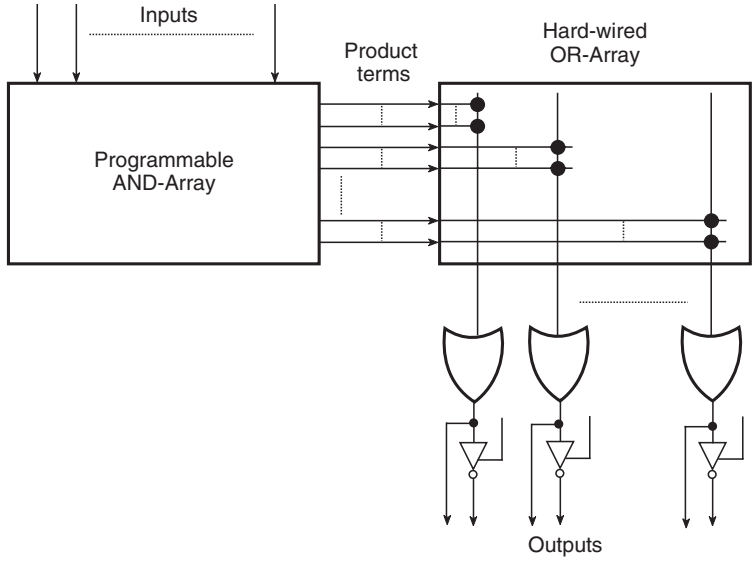


Figure 9.18 Generalized PAL device.

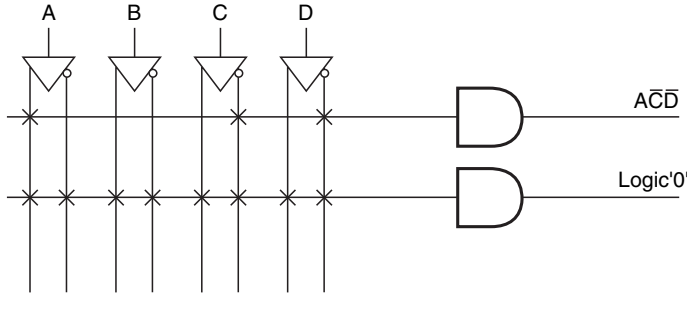


Figure 9.19 Programmability of inputs in a PAL device.

letter indicates the number of outputs. In the case of PAL devices offering a combination of different types of logic output, the rightmost number indicates the number of the output type implied by the letter used in the designation. For example, a PAL device designated PAL-16L8 will have 16 inputs and eight active LOW outputs. Another PAL device designated PAL-16R4 has 16 inputs and four registered outputs. Also, the number of inputs as given by the number designation includes dedicated inputs, user-programmable inputs accessible from combinational I/O pins and any feedback inputs

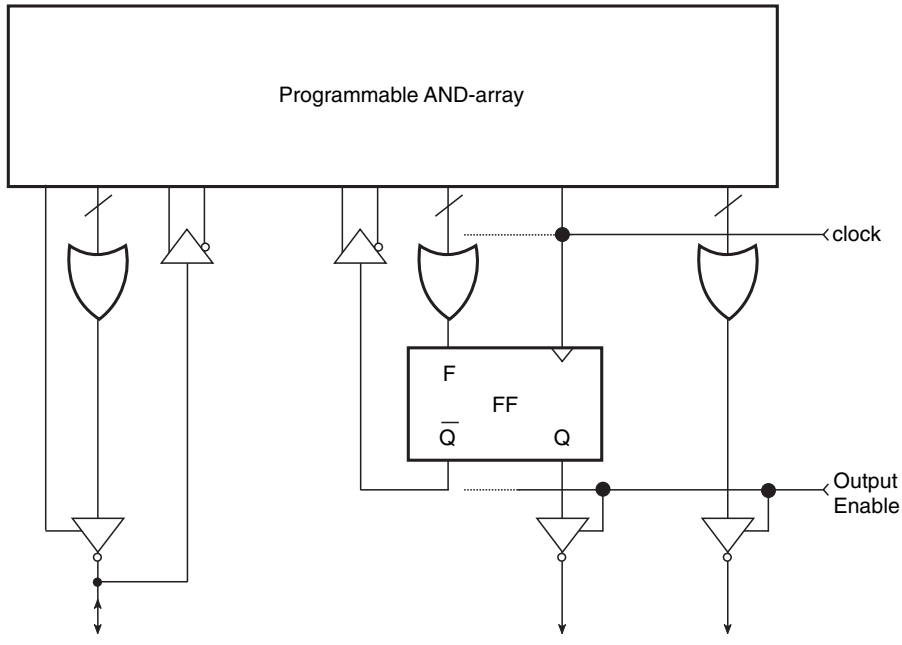


Figure 9.20 Output logic arrangements in a PAL device.

Table 9.3 PAL numbering system.

Architecture – Combinational devices		Architecture – Registered devices	
Code Letter	Description	Code letter	Description
H	Active HIGH outputs	R	Registered outputs
L	Active LOW outputs	X	EXCLUSIVE-OR gates
P	Programmable output polarity	RP	Registered polarity Programmable
C	Complementary outputs	RS	Registered-term steering
XP	EXCLUSIVE-OR gate-Programmable	V	Versatile varied Product terms
S	Product term steering	RX	Registered EX-OR
		MA	Macrocell

from combinational and registered outputs. For example, PAL-16L8 has 10 dedicated inputs and six inputs accessible from I/O pins.

In addition to the numbering system described above, an alphanumeric designation on the extreme left may be used to indicate the technology used. 'C' stands for CMOS, '10H' for 10KH ECL and '100' for 100K ECL. TTL is represented by a blank. A letter on the extreme right may be used to

indicate the power level, with 'L' and 'Q' respectively indicating low and quarter power levels and a blank representing full power.

Example 9.4

Table 9.4 shows the function table of a converter. Starting with the Boolean expressions for the four outputs (P, Q, R, S), minimize them using Karnaugh maps and then hardware-implement this converter with a suitable PLD with PAL architecture.

Solution

From the given function table, we can write the Boolean expressions for the four outputs as follows:

$$P = \overline{A}.B.\overline{C}.D + \overline{A}.B.C.\overline{D} + \overline{A}.B.C.D + A.\overline{B}.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.D \tag{9.13}$$

$$Q = \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D \tag{9.14}$$

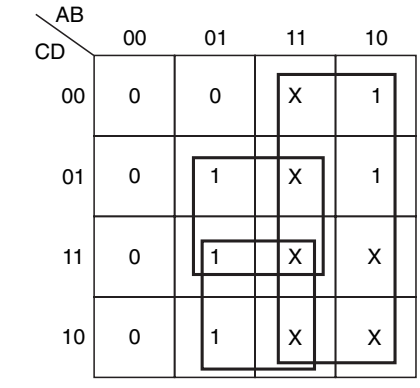
$$R = \overline{A}.\overline{B}.C.\overline{D} + \overline{A}.\overline{B}.C.D + \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D + \overline{A}.B.C.\overline{D} + \overline{A}.B.C.D \tag{9.15}$$

$$S = \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.\overline{B}.C.\overline{D} + \overline{A}.B.C.D + A.\overline{B}.\overline{C}.\overline{D} \tag{9.16}$$

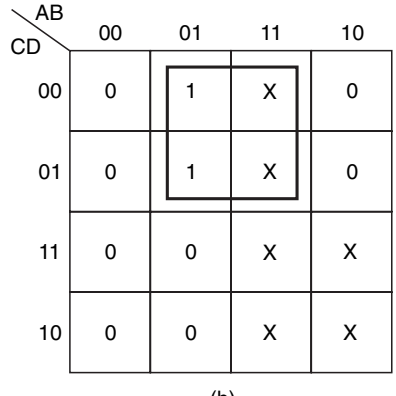
Karnaugh maps for the four outputs P,Q,R and S are respectively shown in Figs 9.21(a) to (d). The minimized Boolean expressions are given by the equations

Table 9.4 Function table in example 9.4.

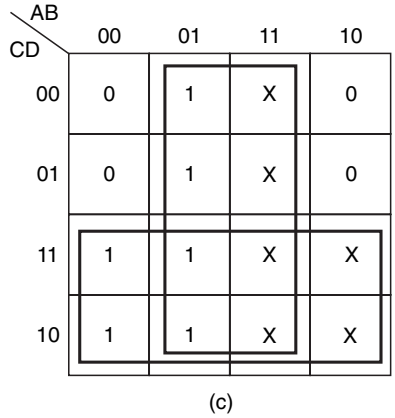
A	B	C	D	P	Q	R	S
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X



(a)



(b)



(c)

Figure 9.21 Karnaugh maps (example 9.4).

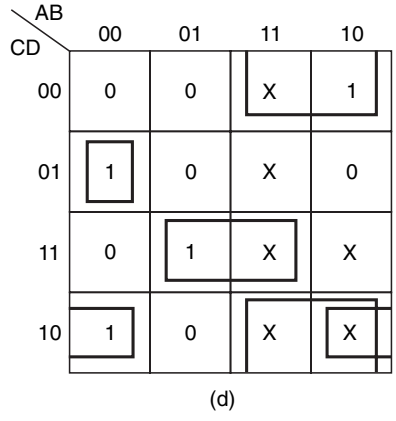


Figure 9.21 (*continued*).

$$P = B.D + B.C + A \tag{9.17}$$

$$Q = B.\overline{C} \tag{9.18}$$

$$R = B + C \tag{9.19}$$

$$S = \overline{A}.\overline{B}.\overline{C}.D + B.C.D + A.\overline{D} + \overline{B}.C.\overline{D} \tag{9.20}$$

The next step is to choose a suitable PAL device. Since there are four output functions, we will need a PAL device with at least four OR gates at the output. Since each of the OR gates is to be hard wired to only a subset of programmable AND arrays, and also because one of the output functions has four product terms, we will need an AND array of 16 AND gates. Since there are four input variables, we need each AND gate in the array to have eight inputs to cater for four variables and their complements. To sum up, we choose a PAL device that has eight inputs, 16 AND gates in the programmable AND array and four OR gates at the output. Each OR gate has four inputs.

Figure 9.22 shows the architecture of the programmed PAL device. We can see that the P output has only three product terms. The fourth input to the relevant OR gate needs to be applied a logic '0' input. This is achieved by feeding the inputs of the corresponding AND gate with all four variables and their complements. Logic 0s, wherever required, are implemented in the same manner. Note that, in the programmed PAL device of Fig. 9.22, an unprogrammed interconnection indicated by a cross ($\times$) is a 'make' connection.

9.6 Generic Array Logic

Generic array logic (GAL) is characterized by a reprogrammable AND array, a fixed OR array and a reprogrammable output logic. It is similar to a PAL device, with the difference that the AND

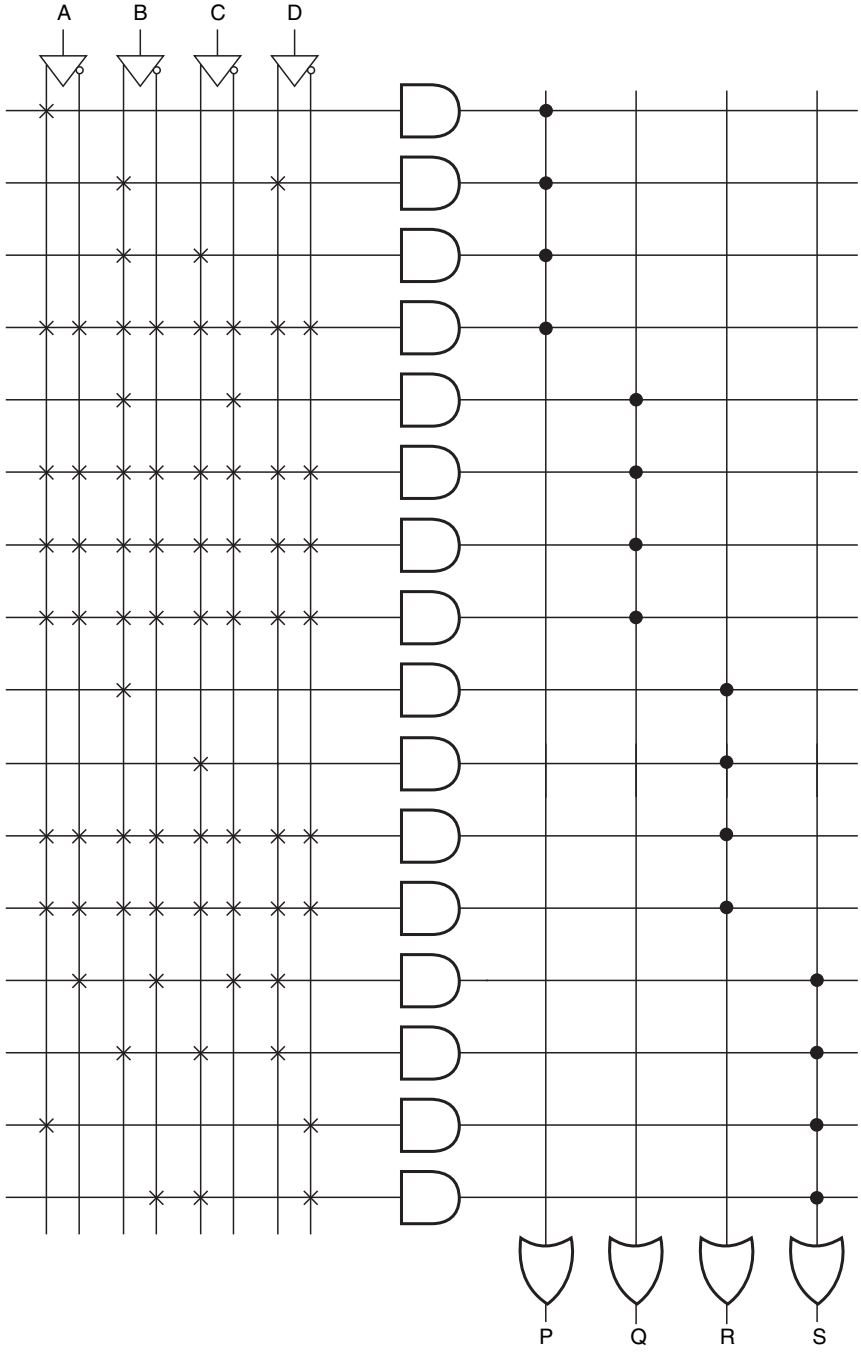


Figure 9.22 Programmed PAL (example 9.4).

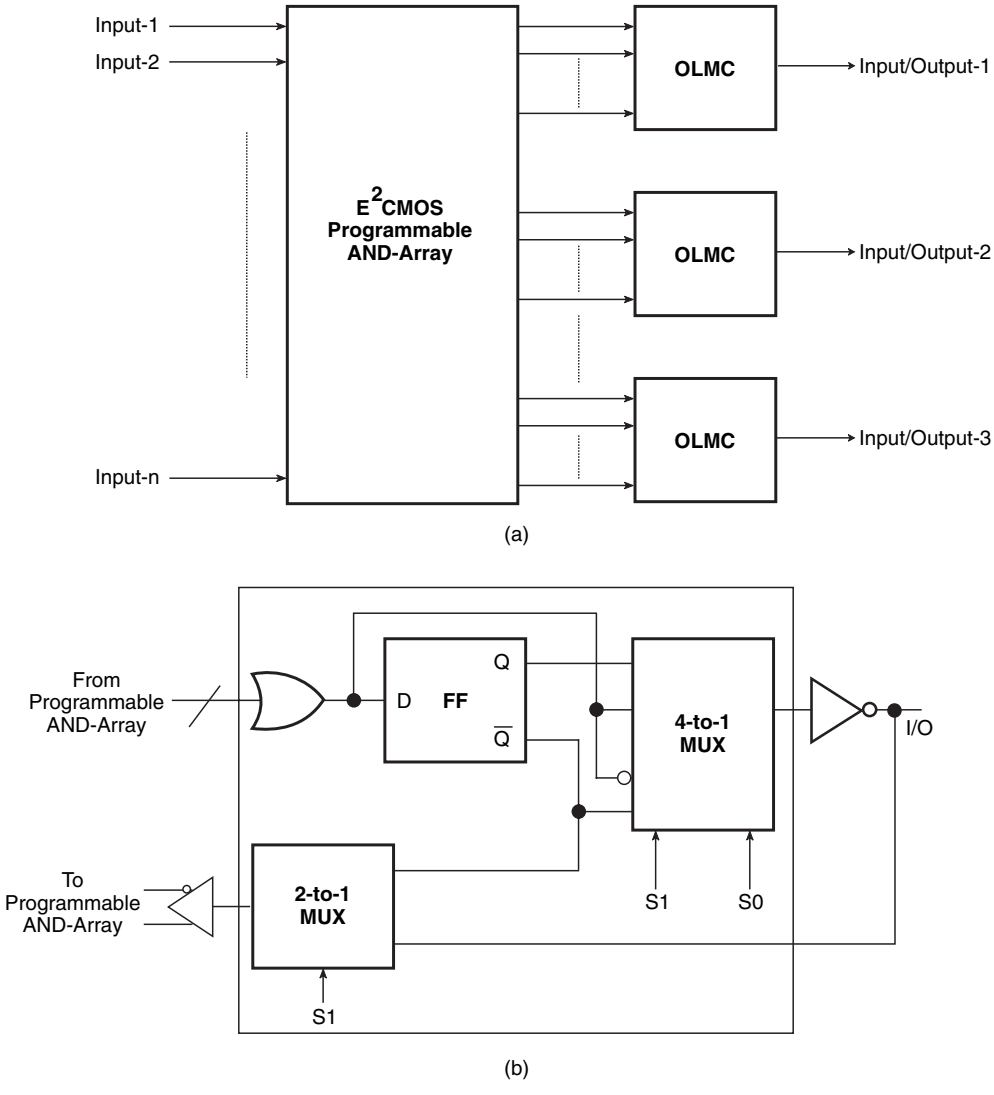


Figure 9.23 (a) Generic array logic generalized block schematic and (b) architecture of an OLMC.

array is not just programmable as is the case in a PAL device but is reprogrammable. That is, it can be reprogrammed any number of times. This has been made possible by the use of electrically erasable PROM cells for storing the programming pattern. The other difference is in the use of reprogrammable output logic, which provides more flexibility to the designer. GAL devices employ output logic macrocells (OLMCs) at the output, which allows the designer to configure the outputs either as combinational outputs or registered outputs.

Figures 9.23(a) and (b) respectively show the block schematic representation of a GAL device and the architecture of a typical OLMC used with GAL devices. The OLMC of the type shown in Fig. 9.23(b) can be configured to produce four different outputs depending upon the selection inputs. These include the following:

1. $S_1S_0 = 00$: registered mode with active LOW output.
2. $S_1S_0 = 01$: registered mode with active HIGH output.
3. $S_1S_0 = 10$: combinational mode with active LOW output.
4. $S_1S_0 = 11$: combinational mode with active HIGH output.

We can see that two of the four inputs to the 4-to-1 multiplexer are combinational outputs, and the other two are the registered outputs. Also, of the two combinational outputs, one is an active HIGH output while the other is an active LOW output. The same is the case with registered outputs. Of the four inputs to the multiplexer, the one appearing at the output depends upon selection inputs. The 2-to-1 multiplexer ensures that the final output is also available as feedback to the programmable AND array.

9.7 Complex Programmable Logic Devices

If we examine the internal architecture of simple programmable logic devices (SPLDs) such as PLAs and PALs, we find that it is not practical to increase their complexity beyond a certain level. This is because the size of the programmable plane (such as the programmable AND plane in a PLA or PAL device) increases too rapidly with increase in the number of inputs to make it a practically viable device. One way to increase the logic capacity of simple programmable logic devices is to integrate multiple SPLDs on a single chip with a programmable interconnect between them. These devices have the same basic internal structure that we see in the case of SPLDs and are grouped together in the category of complex programmable logic devices (CPLDs). Typically, CPLDs may offer a logic capacity equivalent to that of about 50 SPLDs. Programmable logic devices with much higher logic capacities would require a different approach rather than simple extension of the concept of SPLDs.

9.7.1 Internal Architecture

As outlined in the previous paragraph, a CPLD is nothing but the integration of multiple PLDs, a programmable interconnect matrix and an I/O control block on a single chip. Each of the identical PLDs is referred to as a *logic block* or *function block.* Figure 9.24 shows the architecture of a typical CPLD. As is evident from the block schematic arrangement, the programmable interconnect matrix is capable of connecting the input or output of any of the logic blocks to any other logic block. Also, input and output pins connect directly to both the interconnect matrix as well as logic blocks.

Logic blocks may further comprise smaller logic units called macrocells, where each of the macrocells is a subset of a PLD-like logic block. Figure 9.25 shows the structure of a logic block along with its interconnections with the programmable interconnect matrix and I/O block. The horizontal grey-coloured bars inside the logic block constitute an array of macrocells. Typically, each macrocell comprises a set of product terms generated by a subset of the programmable AND array and feeding a configurable output logic. The output logic typically comprises an OR gate, an EX-OR gate and a flip-flop. The flip-flop in the case of most contemporary CPLDs is configurable as a *D*-type, *J-K*, *T*, or *R-S* flip-flop or can even be transparent. Also, the OR gate can be fed with any or all of the product terms generated within the macrocell. Most contemporary CPLDs also offer an architecture where the

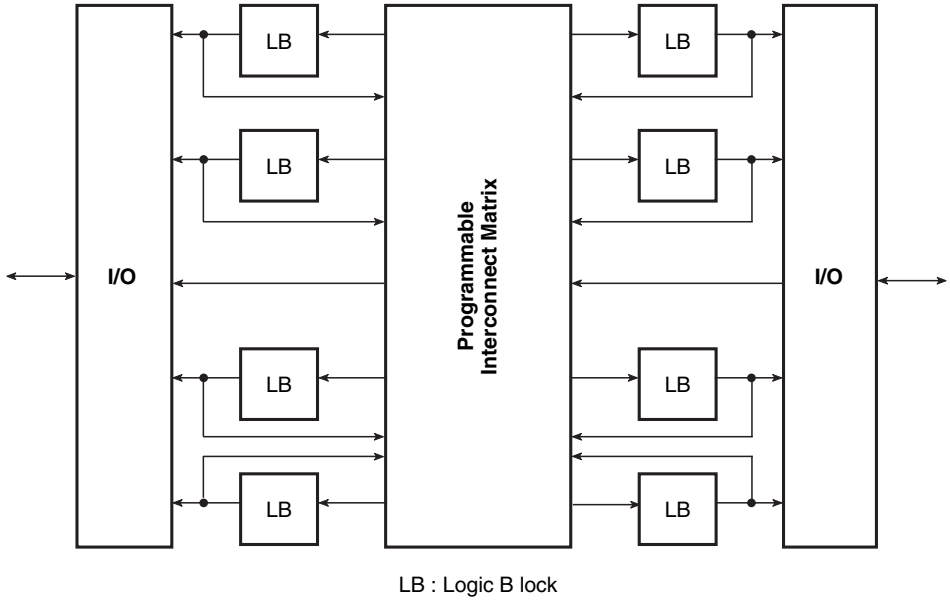


Figure 9.24 CPLD architecture.

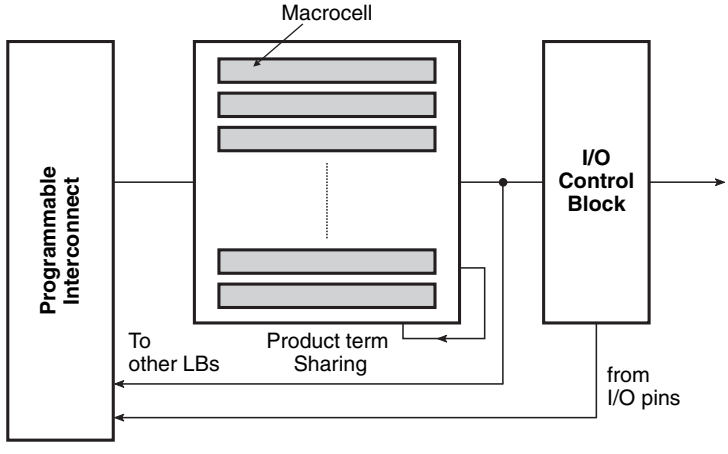


Figure 9.25 Logic block structure.

OR gate can also be fed with some additional product terms generated within other macrocells of the same logic block. For example, a logic block in the case of the MAX-7000 series of CPLDs from Altera offers this product-term flexibility, where the OR gate of each macrocell can have up to 15

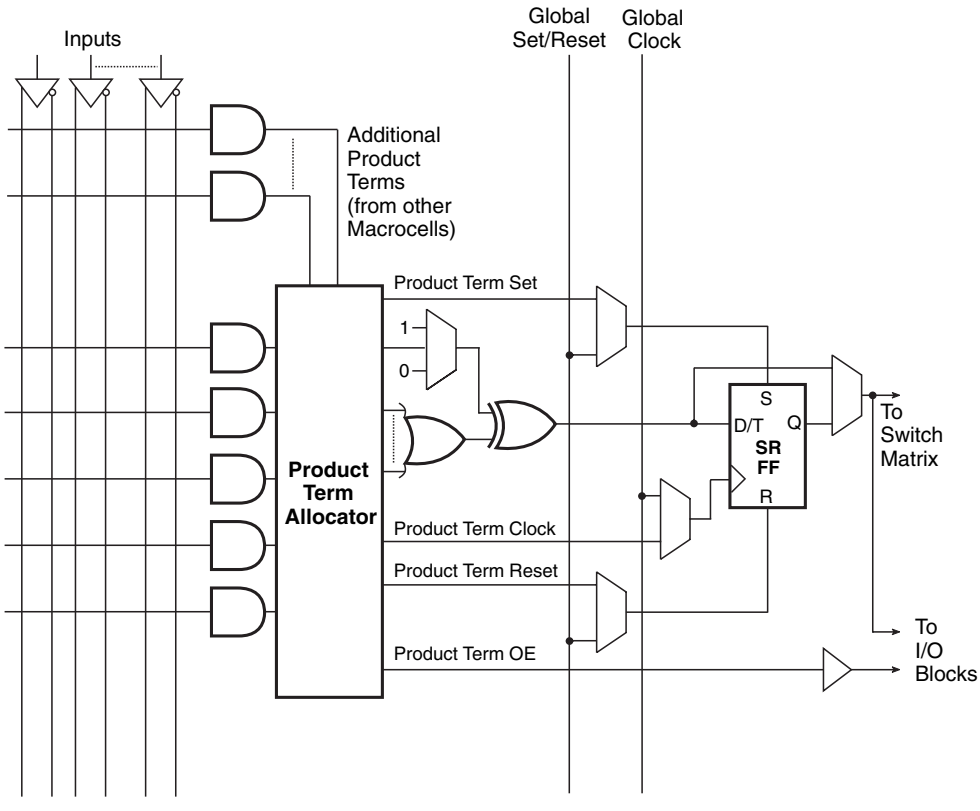


Figure 9.26 Macrocell architecture.

additional product terms from other macrocells in the same logic block, apart from a maximum of five product terms from within the same macrocell.

Figure 9.26 shows the logic diagram of a macrocell typical of macrocells in the logic blocks of most contemporary CPLDs. The diagram is self-explanatory. There may be minor variations in devices from different manufacturers. For example, macrocells in the XC-7000 series CPLDs from Xilinx have two OR gates fed from a two-bit arithmetic logic unit (ALU) and its output feeds a configurable flip-flop.

9.7.2 Applications

Owing to their less flexible internal architecture leading to predictable timing performance, high speed and a range of logic capacities, CPLDs find extensive use in a wide assortment of applications. These include the implementation of random glue logic in prototyping small gate arrays, implementing critical control designs such as graphics controllers, cache control, UARTs, LAN controllers and many more.

CPLDs are fast replacing SPLDs in complex designs. Complex designs using a large number of SPLDs can be replaced with a CPLD-based design with a much smaller number of devices. This is particularly attractive in portable applications such as mobile phones, digital assistants and so on.

CPLD architecture particularly suits those designs that exploit wide AND/OR gates and do not require a large number of flip-flops.

The reprogramming feature of CPLDs makes the incorporation of design changes very easy. With the availability of CPLDs having an in-circuit programming feature, it is even possible to reconfigure the hardware without power down. Changing protocol in a communication circuit could be one such example. One of the most significant advantages of CPLD architecture comes from its simple SPLD-like structure, which allows the design to partition naturally into SPLD-like blocks. This leads to a much more predictable timing or speed performance than would be possible if the design were split into many pieces and mapped into different areas of the chip.

9.8 Field-Programmable Gate Arrays

As outlined earlier, it is not practical to increase the logic capacity with a CPLD architecture beyond a certain point. The highest-capacity general-purpose logic chips available today are the traditional gate arrays, which comprise an array of prefabricated transistors. The chip can be customized during fabrication as per the user's logic design by specifying the metal interconnect pattern. These chips are also referred to as *mask-programmable gate arrays* (MPGAs). These, however, are not field-programmable devices. A field-programmable gate array (FPGA) chip is the user-programmable equivalent of an MPGA chip.

9.8.1 Internal Architecture

An FPGA consists of an array of uncommitted configurable logic blocks, programmable interconnects and I/O blocks. The basic architecture of an FPGA was shown earlier in Fig. 9.7 when presenting an overview of programmable logic devices. As outlined earlier, the basic difference between a CPLD and an FPGA lies in their internal architecture. CPLD architecture is dominated by a relatively smaller number of programmable sum-of-products logic arrays feeding a small number of clocked flip-flops, which makes the architecture less flexible but with more predictable timing characteristics. On the other hand, FPGA architecture is dominated by programmable interconnects, and the configurable logic blocks are relatively simpler. Logic blocks within an FPGA can be as small as the macrocells in a PLD, called fine-grained architecture, or larger and more complex, called coarse-grained architecture. However, they are never as large as the entire PLD like the logic blocks of a CPLD. This feature makes these devices far more flexible in terms of the range of designs that can be implemented with these devices.

Contemporary FPGAs have an on-chip presence of higher-level embedded functions and embedded memories. Some of them even come with an on-chip microprocessor and related peripherals to constitute what is called a complete 'system on a programmable chip'. Virtex-II Pro and Virtex-4 FPGA devices from Xilinx are examples. These devices have one or more PowerPC processors embedded within the FPGA logic fabric.

Figure 9.27 shows a typical logic block of an FPGA. It consists of a four-input look-up table (LUT) whose output feeds a clocked flip-flop. The output can either be a registered output or an unregistered LUT output. Selection of the output takes place in the multiplexer. An LUT is nothing but a small one-bit wide memory array with its address lines representing the inputs to the logic block and a one-bit output acting as the LUT output. An LUT with n inputs can realize any logic function of n inputs by programming the truth table of the desired logic function directly into the memory.

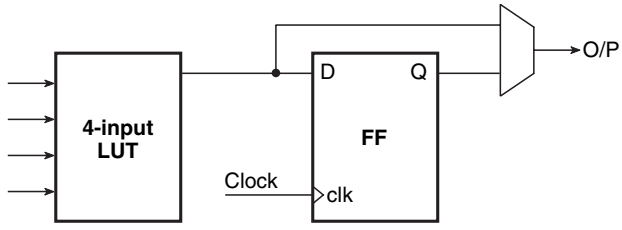


Figure 9.27 Logic block of a typical FPGA.

Logic blocks can have more than one LUT and flip-flops also to give them the capability of realizing more complex logic functions. Figure 9.28 shows the architecture of one such logic block. The architecture shown in Fig. 9.28 is that of a logic block of the XC4000 series of FPGAs from Xilinx. This logic block has two four-input LUTs fed with logic block inputs and a third LUT that can be used in conjunction with the two LUTs to offer a wide range of functions. These include two separate logic functions of four inputs each, a single logic function of up to nine inputs and many more. The logic block contains two flip-flops.

Figure 9.29 shows another similar LUT-based architecture that uses multiple LUTs and flip-flops. The architecture shown in Fig. 9.29 is that of a logic block called a programmable function unit

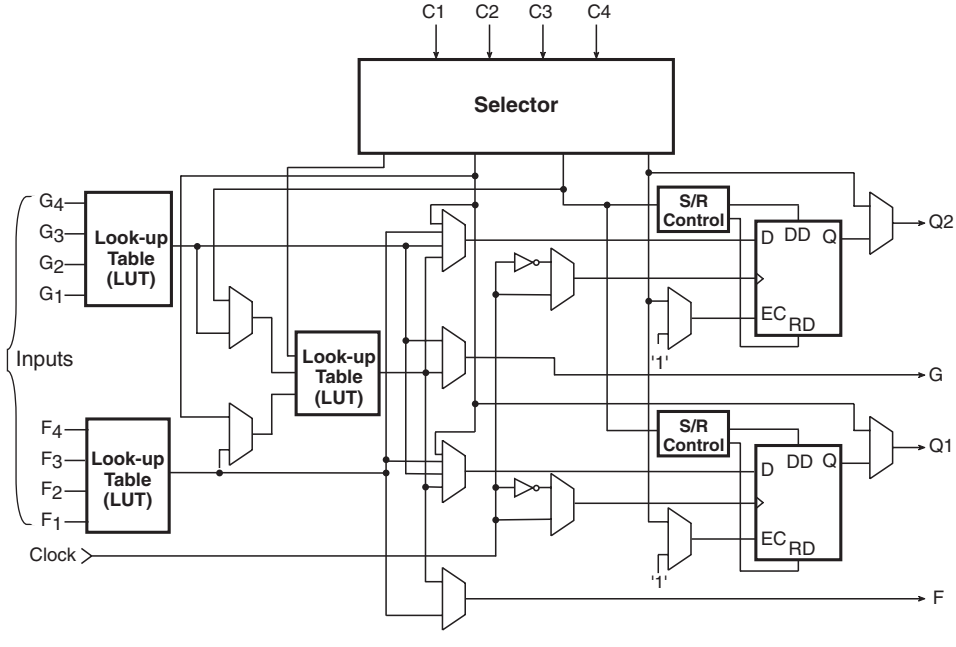


Figure 9.28 Logic block architecture of the XC4000 FPGA from Xilinx.

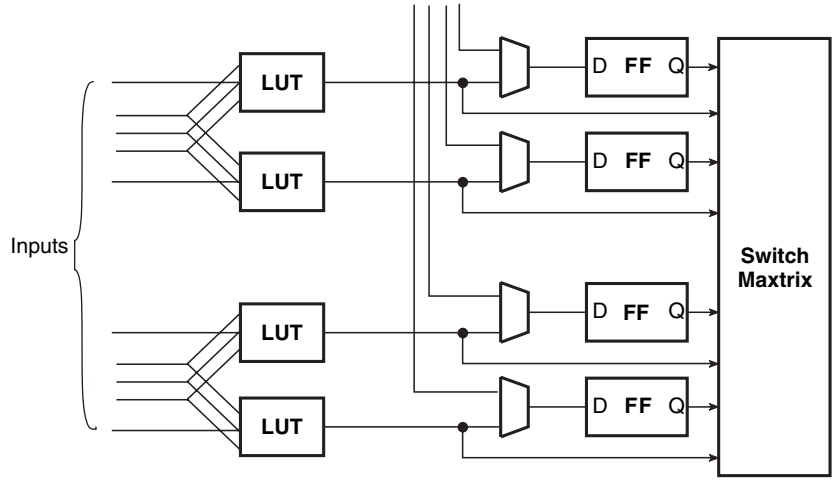


Figure 9.29 Logic block architecture of an AT&T FPGA.

(PFU) by the manufacturer of AT&T FPGA devices. This logic block can be configured either as four four-input LUTs or two five-input LUTs or one six-input LUT.

9.8.2 Applications

In the early days of their arrival on the scene, FPGAs began as competitors to CPLDs for applications such as glue logic for PCBs. With increase in their logic capacity and capability, the availability of a large embedded memory, higher-level embedded functions such as adders and multipliers, the emergence of hybrid technologies combining the logic blocks and interconnects of traditional FPGAs with embedded microprocessors and the facility of full or partial in-system reconfiguration have immensely widened the scope of applications of FPGAs. FPGAs today offer a complete system solution on a single chip, although very complex systems might be implemented with more than one FPGA device.

Some of the major application areas of FPGA devices include digital signal processing, data storage and processing, software-defined radio, ASIC prototyping, speech recognition, computer vision, cryptography, medical imaging, defence systems, bioinformatics, computer hardware emulation and reconfigurable computing. Reconfigurable computing, also called customized computing, involves the use of programmable parts to execute software rather than compiling the software to be run on a regular CPU. This has been made possible by in-system reconfiguration, which allows the internal design to be altered on-the-fly.

9.9 Programmable Interconnect Technologies

The programmable features of every PLD, be it simple programmable logic devices (SPLDs) such as PLAs, PALs and GALs or complex programmable logic devices (CPLDs) or even field-programmable gate arrays (FPGAs), come from their programmable interconnect structure. Interconnect technologies

that have evolved over the years for programming PLDs include fuses, EPROM or EEPROM floating-gate transistors, static RAM and antifuses.

Each one of these is briefly described in the following paragraphs.

9.9.1 Fuse

A fuse is an electrical device that has a low initial resistance and is designed permanently to break an electrically conducting path when current through it exceeds a specified limit. It uses bipolar technology and is nonvolatile and one-time programmable. It was the first user-programmable switch developed for use in PLAs. They were earlier used in smaller PLDs and are now being rapidly replaced by newer technologies.

9.9.2 Floating-Gate Transistor Switch

This interconnect technology is based on the principle of placing a floating-gate transistor between two wires in such a way as to facilitate a WIRE-AND function. This concept is used in EPROM and EEPROM devices, and that is why the floating-gate transistor is sometimes referred to as an EPROM or EEPROM transistor. Figure 9.30 shows the use of floating-gate transistor interconnects in the AND plane of a CPLD or SPLD. All those inputs that are required to be part of a particular product term are activated to drive the product wire to a logic '0' level through the EPROM transistor. For inputs that are not part of the product term, relevant transistors are switched off.

This technology is commonly used in SPLDs and CPLDs. A floating-gate transistor based switch matrix, however, requires a large number of interconnects and therefore transistors. For example, a CPLD with 128 macrocells with four inputs and one output each would require as many as 65 536 interconnects for 100 % routability. A large number of interconnects also adds to the propagation delay.

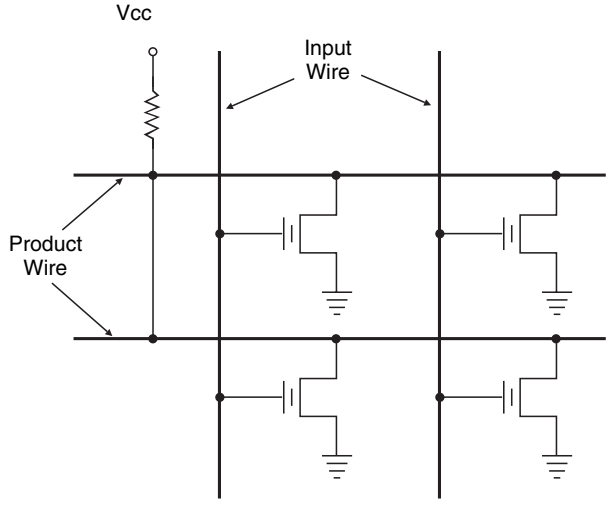


Figure 9.30 Floating-gate transistor interconnect.

The use of multiplexers can reduce this number significantly and can also address the problem of increased propagation delay. An MUX-based interconnect matrix is being used in CPLDs. CPLD type XPLA3 from Xilinx is an example.

9.9.3 Static RAM-Controlled Programmable Switches

Static RAM (SRAM) is basically a semiconductor memory, and the word 'static' implies that it is a nonvolatile memory. That is, the memory retains its contents as long as power is on. A SRAM with m address lines and n data lines is referred to as a $2^m \times n$ memory and is capable of storing 2^m n-bit words. Figure 9.31 shows the basic SRAM cell comprising six MOSFET switches, with four of them connected as cross-coupled inverters. A basic SRAM cell can store one bit of information. The reading operation is carried out by precharging both the bit lines (BL and $\overline{BL}$) to logic '1' and then asserting the WL line. The writing operation is done by giving the desired logic status to the BL line and its complement to the $\overline{BL}$ line and then asserting the WL line.

Figure 9.32 shows the use of SRAM-controlled switches. SRAMs are used to control not only the gate nodes but also the select inputs of multiplexers that drive the logic block inputs. The figure illustrates the routing scheme for feeding the output of one logic block to the input of another via SRAM-controlled pass transistor switches and a SRAM-controlled multiplexer. It may be mentioned here that a SRAM-controlled programmable interconnect matrix does not necessarily use both pass transistors and multiplexers. Whether it uses pass transistors or multiplexers or both is product specific.

9.9.4 Antifuse

An antifuse is an electrical device with a high initial resistance and is designed permanently to create an electrically conducting path typically when voltage across it exceeds a certain level. Antifuses

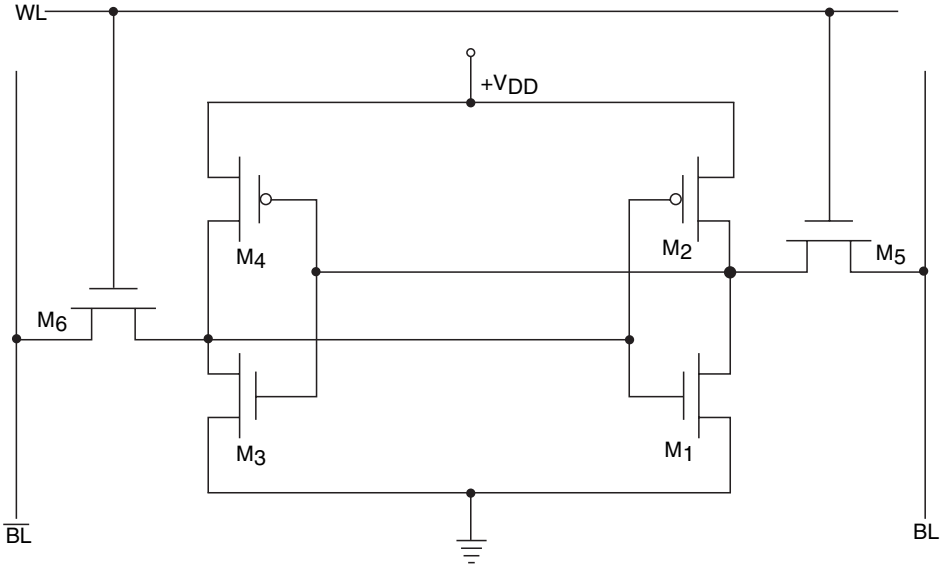


Figure 9.31 SRAM cell.

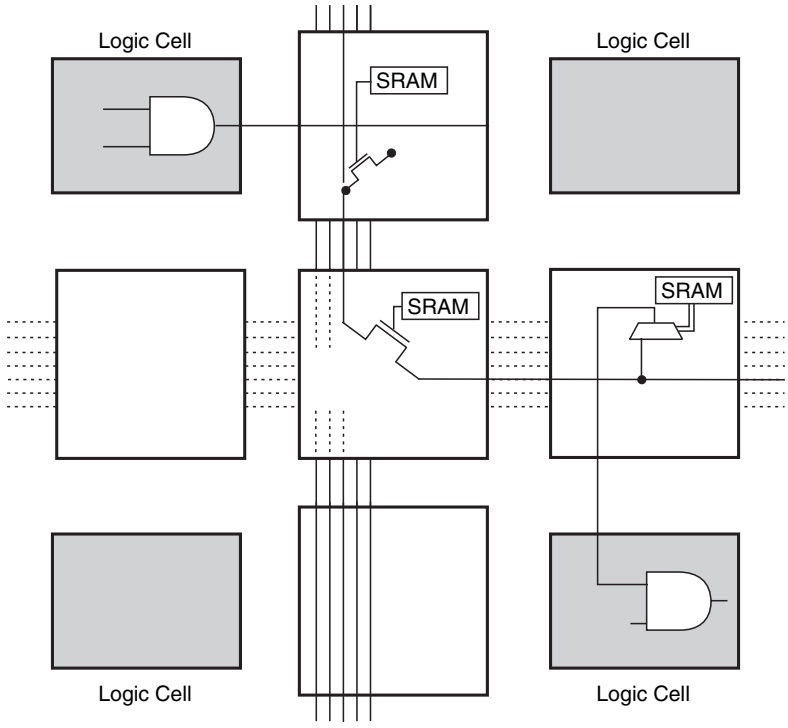


Figure 9.32 SRAM-controlled interconnect.

use CMOS technology, which is one of the main reasons for their wide use in PLDs, FPGAs in particular. A typical antifuse consists of an insulating layer sandwiched between two conducting layers. In the unprogrammed state, the insulating layer isolates the top and bottom conducting layers. When programmed, the insulating layer is transformed into a low-resistance link. Typically, metal is used for conductors and amorphous silicon for the insulator. The application of high voltage across amorphous silicon permanently transforms it into a polycrystalline silicon–metal alloy having a low resistance. There are other antifuse structures too, such as that used in the Actel antifuse. This antifuse, known as PLICE, uses polysilicon and n+ diffusion as conductors and ONO as insulator. Figure 9.33(a) shows the construction. This type of antifuse is usually triggered by a small current of the order of a few milliamperes. The high current density produced in the thin insulating layer produces heat, thus melting the insulating layer and creating an irreversible resistive silicon link.

Antifuses are widely used as programmable interconnects in PLDs [Fig. 9.33(b)]. Antifuse PLDs are one-time programmable, in contrast to SRAM-controlled interconnect-based PLDs, which are reprogrammable. It may be mentioned here that the reprogrammable feature helps the designers fix logic bugs or add new functions. Antifuse PLDs have advantages of nonvolatility and usually higher speeds. Antifuses may also be used in PROMs. In that case, each bit contains both a fuse and an antifuse. The device is programmed by triggering one of the two.

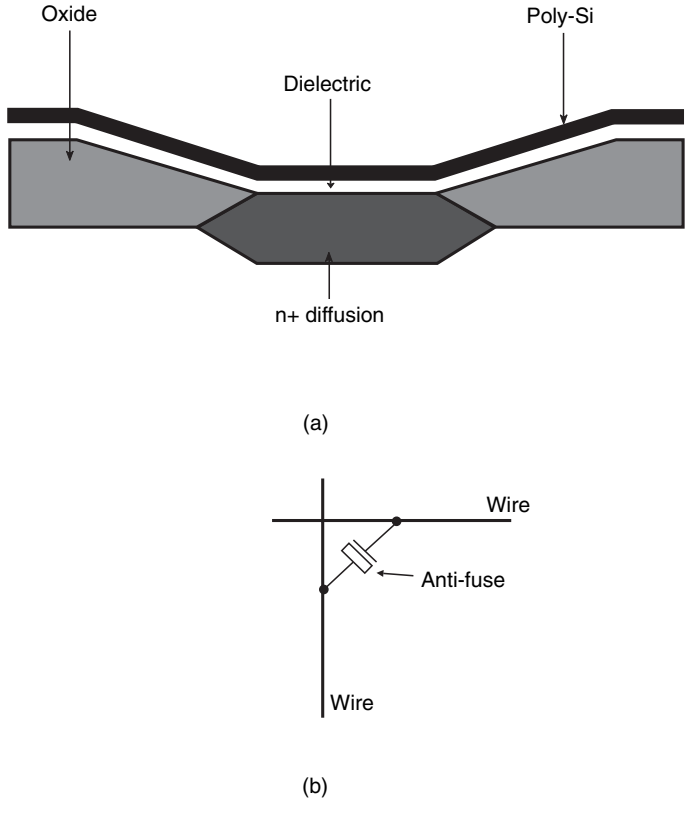


Figure 9.33 (a) Actel's antifuse and (b) the antifuse as a programmable interconnect.

9.10 Design and Development of Programmable Logic Hardware

In this section, we will briefly discuss the various steps involved in the design and development of programmable logic hardware. Figure 9.34 shows a block diagram representation of the sequence of steps involved, in the order in which they are executed.

The process begins with a description of behavioural aspects and the architecture of the intended hardware. This is done by writing a source code in a high-level *hardware description language* (HDL) such as VHDL or Verilog. This step is known as *design entry*. Although schematic capture is also an option for design entry, it has been replaced with language-based tools owing to the designs becoming more and more complex, and also owing to advances in language-based tools.

The most important difference between a hardware and software design is as follows. While software developers tend to think sequentially, hardware designers must think and program in parallel. All input signals are processed in parallel as they travel through a series of macrocells and associated interconnects towards their destination. As a result, statements of HDL create structures, which are executed at the same time. It may be mentioned here that the transfer of information from macrocell to macrocell is synchronized to another signal such as a clock.

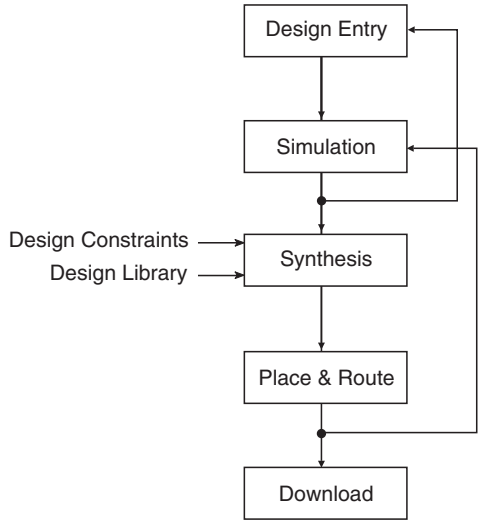


Figure 9.34 Programmable logic design and development process.

The design entry step is either followed by or interspersed with periodic functional *simulation*. The simulator executes the design for a given set of inputs and confirms that the logic is functionally correct.

Hardware compilation comes next. It involves two steps. The first step is *synthesis*, and the result of that is a hardware representation called a netlist. The netlist is device independent and its contents do not depend on the parameters of the PLD to be programmed. It is usually stored in a standard format called the electronic design interchange format (EDIF). The second step, called *place and route*, involves mapping of the logical structure described in the netlist onto actual logic blocks, interconnects and inputs/outputs. The place and route process produces a bit stream, which is nothing but the binary data that must be loaded into CPLD/FPGA to make the chip execute the intended hardware design. It may be mentioned here that each device family has its own proprietary bit stream format.

9.11 Programming Languages

During the PLD development cycle, from design entry to the generation of a bit stream that can be loaded onto the chip using some kind of electronic programming system, two types of software program are needed to perform two different functions.

The first is a *hardware description language* (HDL), which is needed at the design entry stage. HDL is a software programming language that is used to model or describe the intended operation of a piece of hardware. In the present case, this is the function that the PLD chip is intended to perform after it is programmed. It may be worth mentioning here that modern computer languages, including both hardware description languages and high-level programming languages, almost invariably contain declarative and executable statements, and the hardware description languages are particularly rich in the former. If we compare the results of a high-level programming language such as C++ and an HDL, it will be an executable program in the case of the former and declarative in the case of the latter. Hardware description languages that have evolved over the years include ABEL-HDL,

VHDL (VHSIC HDL), Verilog and JHDL (Java HDL). VHSIC stands for Very High-Speed Integrated Circuit.

The second type of software program is a computer program, called a logic compiler, that is used to transform a source code written in HDL into a bit stream. Logic compilers are available from manufacturers or third-party vendors. In the paragraphs to follow, we will briefly describe each of the hardware description languages mentioned above.

9.11.1 ABEL-Hardware Description Language

ABEL-HDL from DATA I/O was intended for relatively simpler PLD circuit designs that could be implemented on SPLDs. ABEL allows the designers to describe the digital circuit designs expressed in the form of truth tables, Boolean functions, state diagrams or any combination of these. It also allows the designer to optimize the design through design validation without specifying a device. In other words, ABEL-HDL facilitates writing hardware-independent programs, and it is only after the design verification and optimization have taken place that the PLD device is chosen. The source code written in the ABEL environment is in standard format to have interface compatibility with other tools.

9.11.2 VHDL-VHSIC Hardware Description Language

VHDL is the most widely used hardware description language used for the purpose of describing complex digital circuit designs that would be implemented on CPLDs and FPGAs. VHDL was originally developed to document the behaviour of ASICs used by various manufacturers in their equipment. It was subsequently followed by the development of logic simulation and synthesis tools that could read VHDL files and output a definition of the physical implementation of the circuit. With modern synthesis tools capable of extracting various digital building blocks such as counters, RAMs, arithmetic blocks, etc., and implementing them as specified by the user, the same VHDL code could be synthesized differently for optimum performance.

VHDL is a strongly typed language. One of the key features of VHDL is that it allows the behaviour of the intended hardware to be described and then verified before the design is translated into actual hardware with the help of synthesis tools. Another feature of VHDL that makes it attractive for digital system design is that it allows description of a concurrent system.

9.11.3 Verilog

Verilog, like VHDL, supports design, design validation and subsequent implementation of analogue, digital and mixed signal circuits at various levels of abstraction. Verilog-based design consists of a hierarchy of modules whose behaviour is defined by concurrent and sequential statements. Sequential statements are placed inside a ‘begin/end’ block and sequential statements contained inside the block are executed sequentially. All concurrent statements and all ‘begin/end’ blocks in the design are executed in parallel. A subset of statements in Verilog is synthesizable. Therefore, if in a given design the different modules use only synthesizable statements, the design can be translated into a netlist, which can further be translated into a bit stream.

Verilog has some similarities and dissimilarities with C-language. It has a similar preprocessor, similar major control keywords like ‘if’, ‘while’, etc., and also a similar formatting mechanism in the printing routines and language operators. Dissimilarities include the use of ‘begin/end’ instead of curly

braces to define a block of code, and also that Verilog does not have structures, pointers and recursive subroutines. Also, the definition of constants in Verilog requires bit width along with their base.

9.11.4 Java HDL

Java HDL (JHDL) was developed in the Configurable Computing Laboratory of Brigham Young University (BYU). It is a low-level hardware description language that primarily uses an object-oriented approach to build circuits. It was developed primarily for the design of FPGA-based hardware, and developers have paid particular attention to supporting the Xilinx series of FPGA chips.

9.12 Application Information on PLDs

In this section, we will look at salient features of some of the commonly used programmable logic devices including SPLDs such as PALs/GALs, CPLDs and FPGAs covering a wide spectrum of devices from leading international manufacturers. Other application-relevant information such as internal architecture, pin connection diagram, etc., is also given for some of the more popular type numbers.

9.12.1 SPLDs

Some of the famous companies that offer SPLDs include Advanced Micro Devices (AMD), Altera, Philips-Signetics, Cypress, Lattice Semiconductor Corporation and ICT. A large range of SPLD products are available from these companies. All of these SPLDs share some common features in terms of the nature of the programmable logic planes, configurable output logic, etc. However, each of these logic devices does offer some unique features that make it particularly attractive for some applications. Some of the widely exploited SPLDs include the 16XX series (16L8, 16R8,16R6 and 16R4) and 22V10 from AMD and EP610 from Altera. These devices are also widely second-sourced by many companies. The Plus 16XX series from Philips is 100 % pin and functional compatible with the 16XX series. 16R8 in the 16XX series and 22V10 PAL devices are industry standards and are widely second-sourced. We will discuss 16XX and 22V10 in a little more detail in the following paragraphs.

The 16XX family of PAL devices employs the familiar sum-of-products implementation comprising a programmable AND array and a fixed OR array. The family offers four PAL-type devices including 16L8, 16R8, 16R6 and 16R4.

Each of the devices in the 16XX family is characterized by a certain number of combinational and registered outputs available to the designer. The devices have three-state output buffers on each output pin, which can be programmed for individual control of all outputs. Other features include the availability of programmable bidirectional pins and output registers. These devices are capable of replacing an equivalent of four or more SSI/MSI integrated circuits. The I/O configuration of the four devices in the 16XX family is summarized in Table 9.5. Figures 9.35(a) to (d) give the basic architecture/pin connections of 16L8, 16R8, 16R6 and 16R4 respectively.

As outlined earlier, many companies offer 22V10 PAL devices. These are available in both bipolar and CMOS technologies. One such contemporary device is GAL 22V10 from Lattice Semiconductor Corporation. As inherent in the type number, the device offers a maximum of 22 inputs and 10 outputs. The outputs are versatile. That is, each one of them can be configured by the user to be either a combinational or registered output. Also, the outputs can be configured to be either active HIGH or active LOW.

Table 9.5 Input/output configuration of the 16XX family.

Device number	Dedicated inputs	Combinational outputs	Registered outputs
16L8	10	8 (6 I/O)	0
16R8	8	0	8
16R6	8	2 I/O	6
16R4	8	4 I/O	4

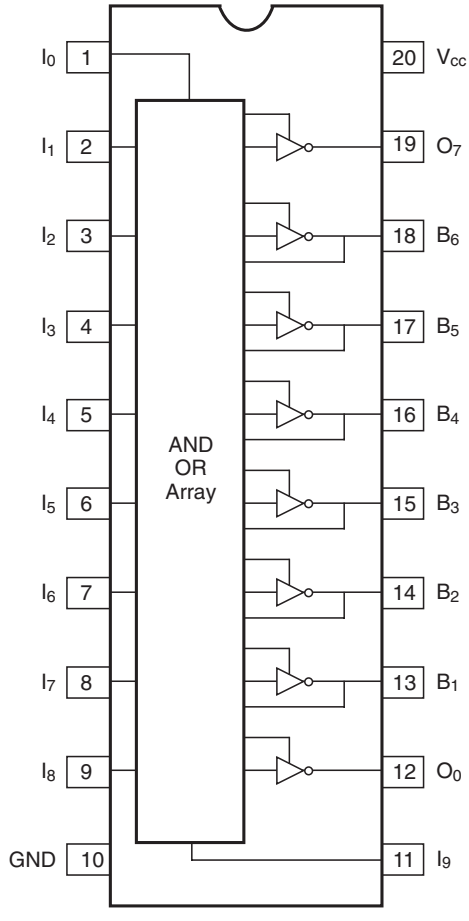
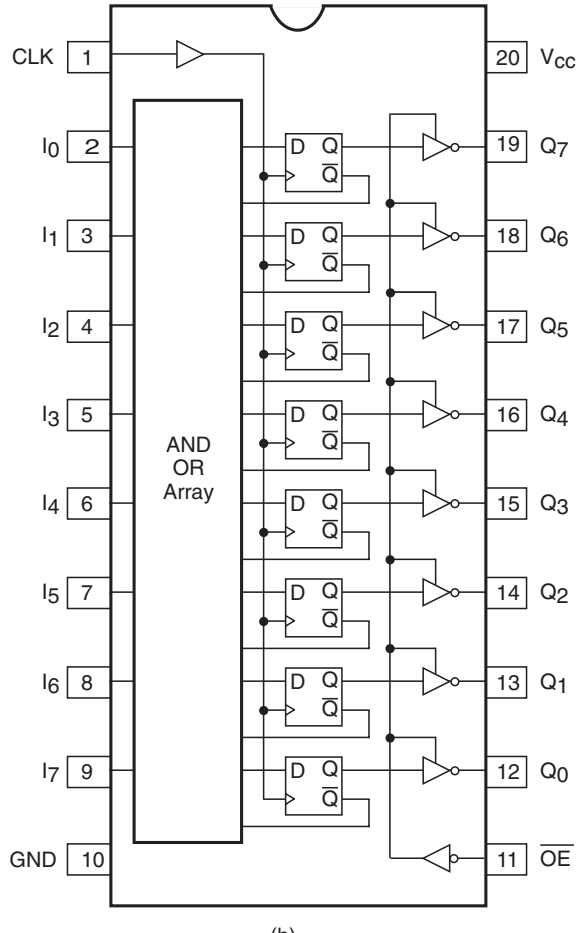


Figure 9.35 Basic architecture/pin connections of the 16XX-series PAL devices.



(b)

Figure 9.35 (*continued*).

GAL 22V10 uses E^2CMOS (electrically erasable CMOS) technology which allows the device to be reprogrammable through the use of an electrically erasable (E^2) floating-gate technology and consume much less power compared with bipolar 22V10 devices owing to the use of advanced CMOS technology. The device specifies 100 erase/write cycles, a 50–75 % saving in power consumption compared with bipolar equivalents and a maximum propagation delay of 4 ns. Each of the output logic macrocells offers two primary functional modes, which include combinational I/O and registered modes. The type of mode (whether combinational I/O or registered) and the output polarity (whether active HIGH or active LOW) are decided by the selection inputs S_0 and S_1, which are normally controlled by the logic compiler. For S_1S_0 equal to 00, 01, 10 and 11, outputs are active LOW registered, active LOW combinational, active HIGH registered and active HIGH combinational respectively.

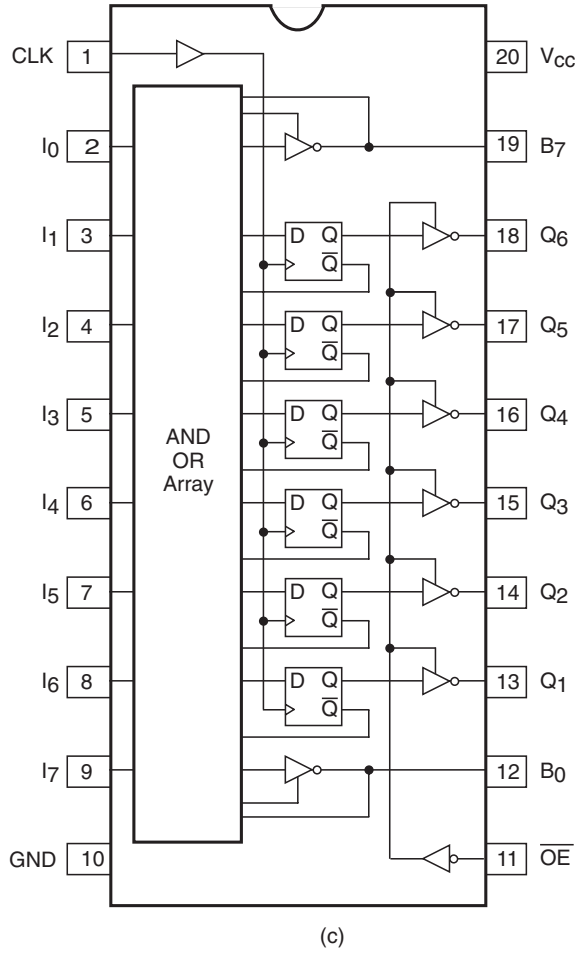


Figure 9.35 (*continued*).

Figure 9.36 shows the basic architecture and pin connection diagram of GAL 22V10. The internal architecture of the output logic macrocell (OLMC) shown as a block in Fig. 9.36 is given in Fig. 9.37.

9.12.2 CPLDs

Major CPLD manufacturers include Altera, Lattice Semiconductor Corporation, Advanced Micro Devices, ICT, Cypress and Xilinx. A large variety of CPLD devices are available from these companies. In the following paragraphs, some of the popular type numbers of CPLDs offered by some of these companies are examined in terms of their characteristic features.

We will begin with CPLDs from Altera. Altera offers three families of CPLDs. These include MAX-5000, MAX-7000 and MAX-9000. MAX-5000 uses an older technology and is used in applications

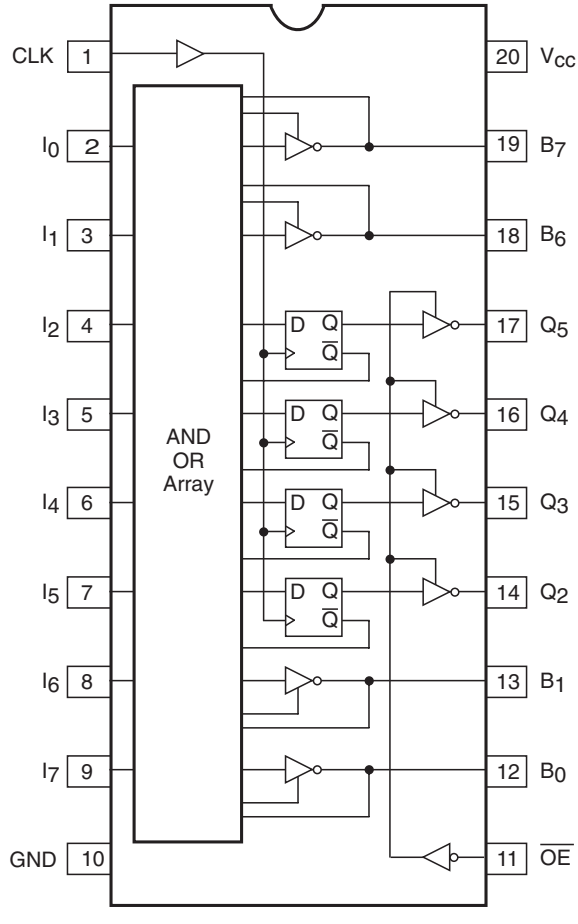


Figure 9.35 (*continued*).

where the designer is looking for cost-effective solutions. The MAX-7000 series of CPLDs are the most widely used ones. MAX-9000 is similar to MAX-7000 except for its higher logic capacity. MAX-7000 series devices use advanced CMOS technology and (E^2PROM)-based architecture and offer densities from 32 to 512 macrocells with pin-to-pin propagation delays as small as 3.5 ns. MAX-7000 devices support in-system programmability and are available with 5.0, 3.3 and 2.5 V core operating voltages. There are three types of device in the MAX-7000 series. These include MAX-7000S, MAX-7000AE and MAX-7000B. Three types are pin-to-pin compatible when used in the same package. Figure 9.38 shows the basic architecture of the MAX-7000 series of CPLDs.

AMD offers the Mach-1 to Mach-5 series of CPLDs. While Mach-1 and Mach-2 are configured around 22V10 PALs, Mach-3 and Mach-4 use 34V16 PALs. Mach-5 is similar to the Mach-4 CPLD except that it offers higher speed performance. All Mach devices use E^2PROM technology.

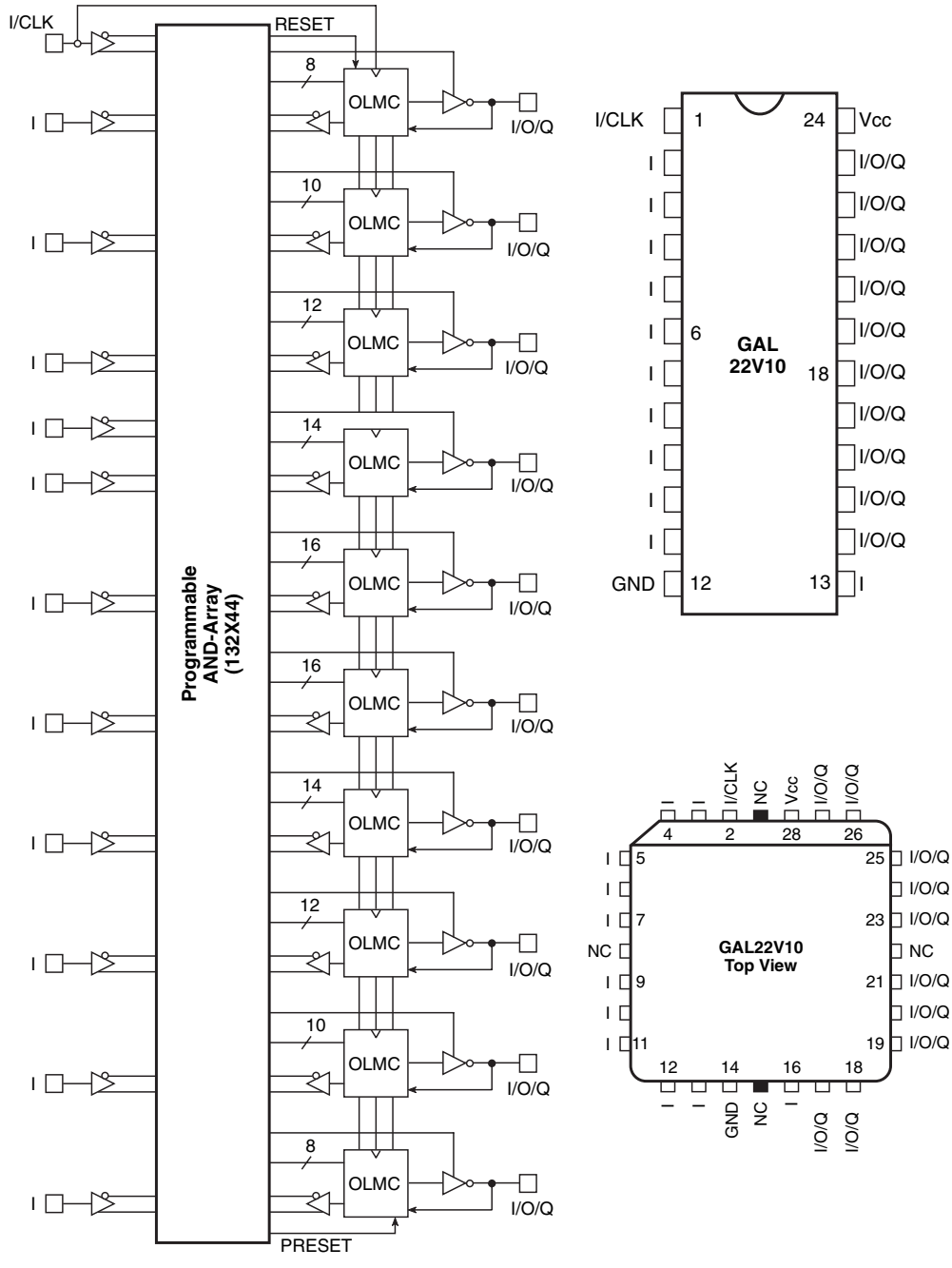


Figure 9.36 Basic architecture and pin connections of 22V10.

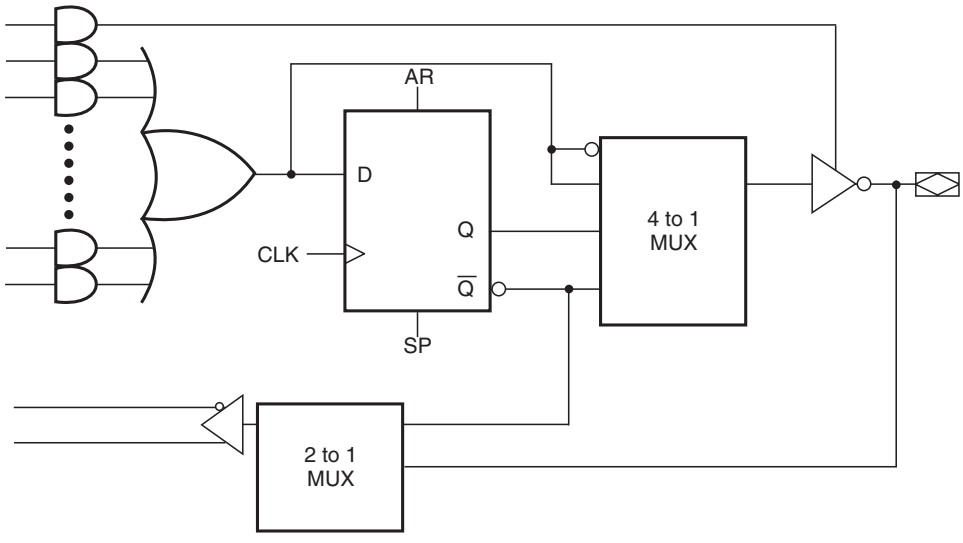


Figure 9.37 Architecture of an output GAL 22V10 logic macrocell.

Figure 9.39 shows the basic architecture of Mach-4 CPLDs. The number of 34V16-like PALs used varies from 6 to 16. Each 34V16-like PAL block consists of a maximum of 34 inputs and 16 outputs. The 34 inputs include 16 outputs that are fed back. All connections in the case of Mach-4 CPLDs, from one PAL block to another and also from a PAL block back to itself, are routed through a central switching matrix, on account of which all connections travel through the same path. This feature gives more predictable time delays in circuits implemented on Mach-4 devices.

Lattice offers the pLSI and ispLSI 1000-series, 2000-series and 3000-series of CPLDs. ispLSI devices are similar to pLSI devices except that they are in-system programmable. The three series of devices differ mainly in logic capacities and speed performance. The logic capacity in the case of the 1000-series CPLDs ranges from about 1200 to 4000 gates, and the pin-to-pin propagation delay is of the order of 10 ns. The ispLSI-1016 CPLD is one such device from the 1000-series of devices. It has a logic capacity of 2000 PLD gates and a pin-to-pin propagation delay of 7.5 ns. The device has four dedicated inputs, 32 universal I/O pins and 96 registers. It uses high-performance E^2CMOS technology, because of which it offers reprogrammability of the logic as well as the interconnects to provide truly reconfigurable systems.

The 2000-series devices have a logic capacity of 600–2000 equivalent gates that offer a higher ratio of macrocells to I/O pins. With a pin-to-pin propagation delay of 5.5 ns, they offer a higher speed performance compared with 1000-series devices. Of the three device families, the 3000-series has the highest logic capacity (up to 5000 equivalent gates). The propagation delay is in the range 10–15 ns. The 3000-series of devices offers some enhancements over the other two series of CPLDs to support more recent design approaches.

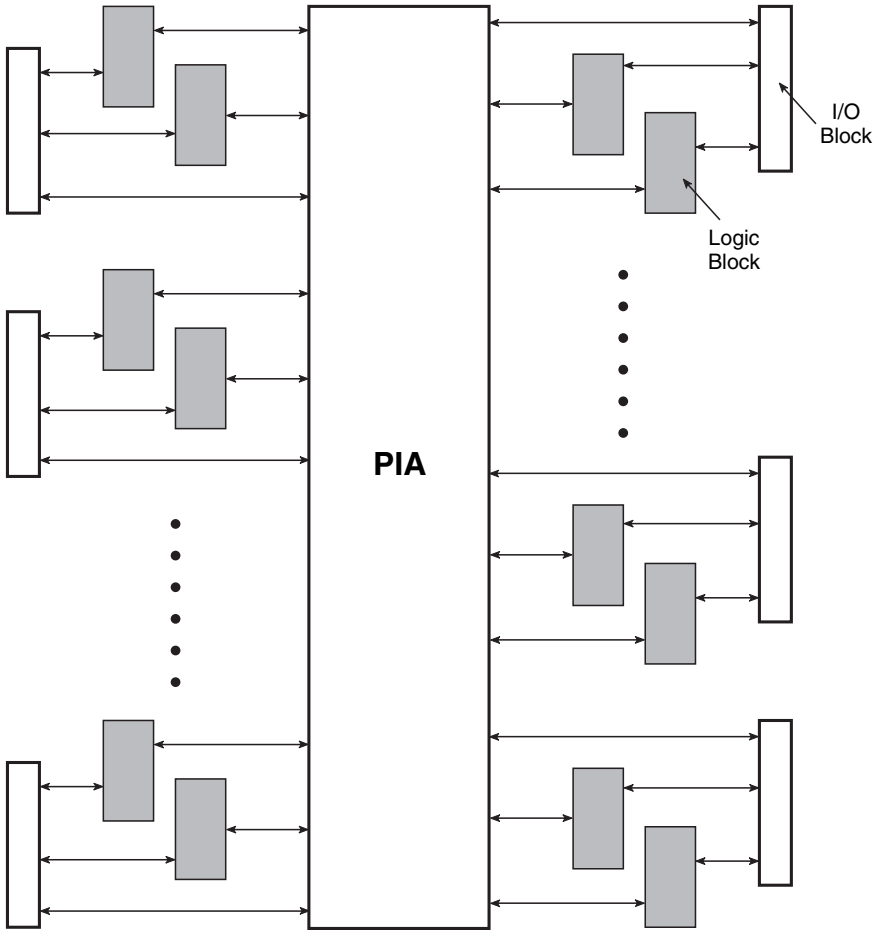


Figure 9.38 MAX-7000 series CPLD architecture.

FLASH-370 from Cypress is yet another popular class of CPLDs. FLASH-370 CPLDs use FLASH E^2PROM technology. Devices are not in-system programmable. One of the salient features of these devices is that they provide more inputs/outputs than the competing products featuring a linear relationship between the number of macrocells and the number of bidirectional I/O pins. FLASH-370 has a typical CPLD architecture as shown in Fig. 9.40, with multiple PAL-like blocks and a programmable interconnect matrix to connect them.

Xilinx, although mainly known for their range of FPGAs, offer CPLDs too. Major families of CPLDs from Xilinx include the XC-7000, CoolRunner and XC-9500 in-system programmable family of devices. The XC-7000 family of CPLDs further comprises two major series, namely XC-7200 and XC-7300. XC-7300 is an enhanced version of XC-7200 in terms of gate capacity and speed

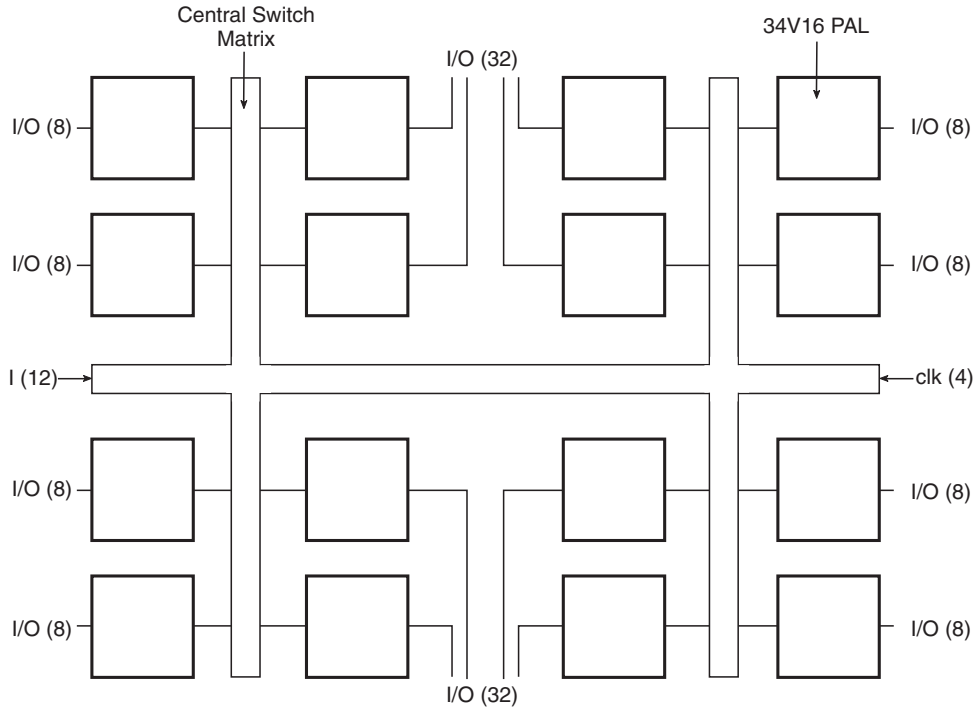


Figure 9.39 Mach-4 CPLD architecture.

performance. XC-7200 offers a logic capacity of 600–1500 gates with a speed performance of 25 ns pin-to-pin propagation delay. XC-7300 offers a gate capacity of up to 3000 gates. Each device in the XC-7000 family contains SPLD-like logic blocks, with each block having nine macrocells. A notable difference between the XC-7000 family of CPLDs and their counterparts from other manufacturers is that each macrocell has two OR gates whose outputs feed a two-bit arithmetic logic unit (ALU), which in turn can generate any function of its two inputs. The ALU output feeds a configurable flip-flop.

The CoolRunner family of CPLDs is characterized by high speed (5 ns pin-to-pin propagation delay) and low power consumption (100 μA of standby current). The family includes the XPLAE series of devices, available in 32, 64 and 128 macrocell versions, the XPLA2-series, which is SRAM-based and available in 320 and 920 macrocell capacities, and the XPLA3 series, available in 32, 64, 128, 256 and 384 macrocell versions.

The XC-9500 family of devices comprises the XC-9536, XC-9572, XC-95108, XC-95144, XC-95216 and XC-95288 series of CPLDs. The family offers a logic capacity ranging from 800 gates (in the case of XC-9536) to 6400 gates (in the case of XC-95288), with a propagation delay varying from 5 ns (in the case of XC-9536) to 15 ns (in the case of XC-95288). Architectural features of the XC-9500 family of CPLDs provide in-system programmability with a minimum of 10 000 program/erase cycles. Other features include output slew rate control and user-programmable ground pins, which help reduce system noise.

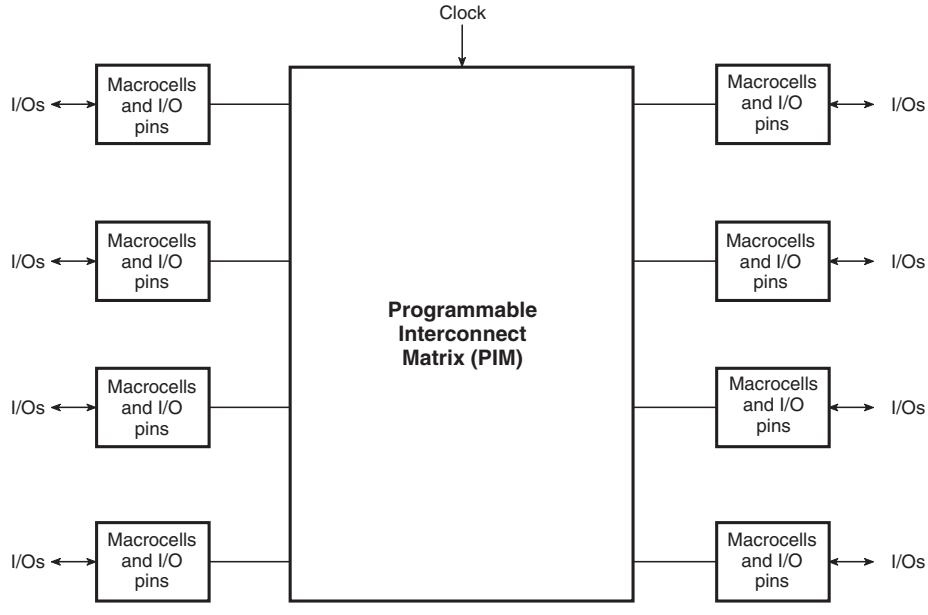


Figure 9.40 FLASH-370 CPLD architecture.

9.12.3 FPGAs

There are two broad categories of FPGAs, namely SRAM-based FPGAs and antifuse-based FPGAs. While Xilinx and Altera are the major players in the former category, antifuse-based devices are offered mainly by Xilinx, Actel, Quicklogic and Cypress. FPGAs were introduced by Xilinx with the XC-2000 series of devices, which have been subsequently followed up by the XC-3000 series, XC-4000 series and XC-5000 series of devices. Of all these, the XC-4000 series is the most widely used one. These are all SRAM-based. Xilinx has also introduced an antifuse-based FPGA family of FPGAs called XC-8100.

The basic architecture of the XC-4000 family is built around a two-dimensional array of configurable logic blocks (CLBs) that can be interconnected by horizontal and vertical routing channels and are surrounded by a perimeter of programmable input/output blocks (IOBs). CLBs provide the functional elements for constructing the user-desired logic function, and IOBs provide the interface between the package pins and internal signal lines. These devices are reconfigurable and are in-system programmable. Table 9.6 gives salient features of the XC-4000X and XC-4000E series of FPGAs.

Altera offers the FLEX-8000 and FLEX-10000 series of FPGAs. FLEX-8000 is SRAM-based. It combines the fine-grained architecture and high register count characteristics of FPGAs with the high speed and predictable interconnect timing delays of CPLDs. The basic logic element comprises a four-input look-up table (LUT) that provides combinational capability and a programmable register that provides sequential capability. Table 9.7 outlines salient features of the FLEX-8000 series of devices.

The FLEX-10000 series offers all the features of FLEX-8000 series devices, with the addition of variable-sized blocks of SRAM called embedded array blocks (EABs). Each of the EABs can be

Table 9.6 Salient features of the XC-4000X and XC-4000E series of FPGAs.

Device number	Logic cells	Maximum logic gates (no RAM)	CLB matrix	Number of CLBs	Number of flip-flops	Maximum user I/Os
XC4002XL	152	1 600	8×8	64	256	64
XC4003E	238	3 000	10×10	100	360	80
XC4005E/XL	466	5 000	14×14	196	616	112
XC4006E	608	6 000	16×16	256	768	128
XC4008E	770	8 000	18×18	324	936	144
XC4010E/XL	950	10 000	20×20	400	1120	160
XC4013E/XL	1368	13 000	24×24	576	1536	192
XC4020E/XL	1862	20 000	28×28	784	2016	224
XC4025E	2432	25 000	32×32	1024	2560	256
XC4028EX/	3078	28 000	32×32	1024	2560	256
XC4036EX/XL	3078	36 000	36×36	1296	3168	288
XC4044XL	3800	44 000	40×40	1600	3840	320
XC4052	4598	52 000	44×44	1936	4576	352
XC4062XL	5472	62 000	48×48	2304	5376	384
XC4085	7448	85 000	56×56	3136	7168	448

Table 9.7 Salient features of the FLEX-8000 series of devices.

Device number	Usable Gates	Flip-flops	Logic Array Blocks (LAB)	Logic Elements (LE)	Maximum User I/O PIns
EPF 8282A/AV	2 500	282	26	208	78
EPF 8452A	4 000	452	42	336	120
EPF 8636A	6 000	636	63	504	136
EPF 8820A	8 000	820	84	672	152
EPF 81188A	12 000	1188	126	1008	184
EPF 81500A	16 000	1500	162	1296	208

configured to serve as an SRAM block with a variable aspect ratio of 256×8, 512×4, $1K \times 2$ or $2K \times 1$.

AT&T offers SRAM-based FPGAs that are similar in architecture to those offered by Xilinx. The overall structure is called an optimized reconfigurable cell array (ORCA). The basic logic block is referred to as a programmable function unit (PFU). Similarities with the Xilinx-4000 series FPGAs include arithmetic circuitry being a part of the PFU and PFU configurability as a RAM. The PFU can be configured as either four four-input LUTs or as two five-input LUTs or as one six-input LUT. When configured as four-input LUTs, it is essential that the various LUT inputs come from the same PFU input. Although on the one hand this reduces the functionality of the PFU, on the other hand it significantly reduces the associated wiring cost.

Actel FPGAs use antifuse technology. Actel offers three main families of FPGA devices, namely Act-1, Act-2 and Act-3. All three series of devices have similar features. The structure is similar to that

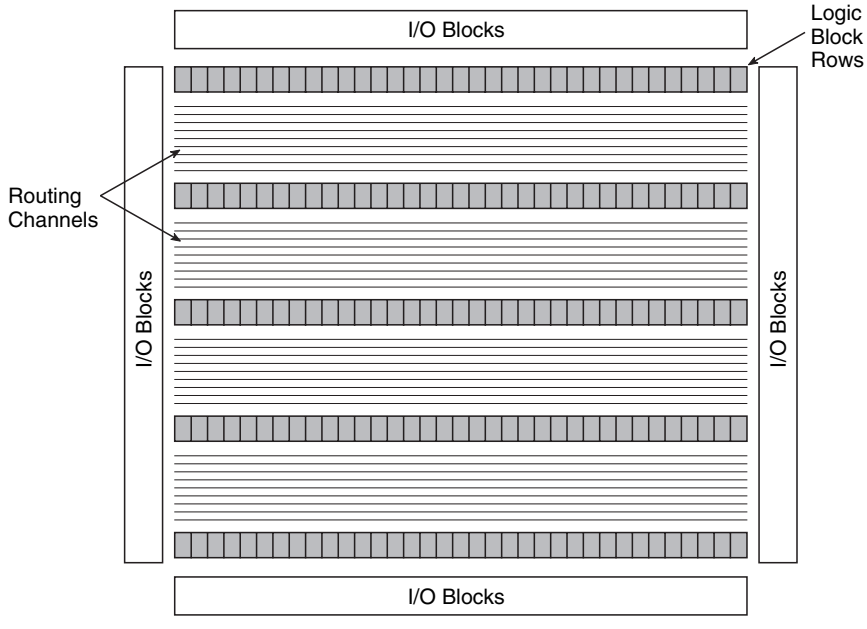


Figure 9.41 Actel FPGA.

of traditional gate arrays comprising logic blocks arranged in horizontal rows with horizontal routing channels between adjacent rows, as shown in Fig. 9.41(a). Actel chips also have vertical wires that overlay the logic blocks to provide signal paths that span multiple rows. These are not shown in Fig. 9.41(a). The logic block is not LUT based. Instead, it comprises an AND gate and an OR gate feeding a multiplexer circuit block, as shown in Fig. 9.41(b). The multiplexer circuit, along with the two gates, can realize a large range of logic functions. In the case of Act-3 FPGAs, 50 % of the logic blocks also contain a flip-flop.

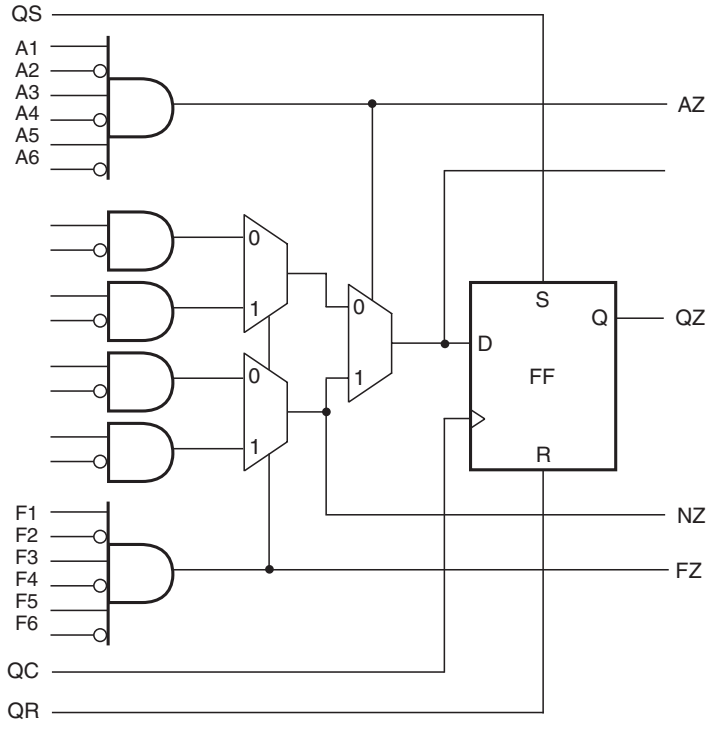


Figure 9.42 Quicklogic FPGA logic block.

Quicklogic also offers antifuse-based FPGAs, like Actel. They offer two families of devices, namely pASIC and pASIC-2. pASIC-2 is an enhanced version of pASIC. The overall structure is array based like the Xilinx FPGAs. The logic blocks are similar to those used in the Actel FPGAs, although more complex than their Actel counterparts. Also, each logic block contains a flip-flop. Figure 9.42 shows the architecture.

Review Questions

1. How does a programmable logic device differ from a fixed logic device? What are the primary advantages of using programmable logic devices?
2. Distinguish between a programmable logic array (PLA) device and a programmable array logic (PAL) device in terms of architecture and capability to implement Boolean functions.
3. How does a generic array logic (GAL) device differ from its PAL counterpart? Do they differ in their internal architecture? If yes, then how?
4. What are complex programmable logic devices (CPLDs)? Briefly outline salient features of these devices and application areas where these devices fit the best.
5. How does the architecture of a typical FPGA device differ from that of a CPLD? In what way does the architecture affect the timing performance in the two cases?

6. What are the various interconnect technologies used for the purpose of programming PLDs? Briefly describe each one of them.
7. What is a hardware description language? What are the requirements of a good HDL? Briefly describe the salient features of VHDL and Verilog.
8. What do you understand by the following as regards programmable logic devices?

   (a) combinational and registered outputs;
   (b) configurable output logic cell;
   (c) reprogrammable PLD;
   (d) in-system programmability.

Problems

1. Figure 9.43 shows a portion of the internal logic diagram of a certain PAL device that uses antifuse interconnect technology. In the diagram shown, a cross (×) represents an unprogrammed interconnect and the absence of a cross (×) at an intersection of input and product lines represents programmed interconnects; a dot (•) represents a hard-wired interconnect. Write (a) the Boolean expression for Y and (b) the Boolean expression for Y if the interconnect technology were fuse based.

   (a) $Y = \overline{A}.B + A.\overline{B}$; (b) the same as in the case of (a)

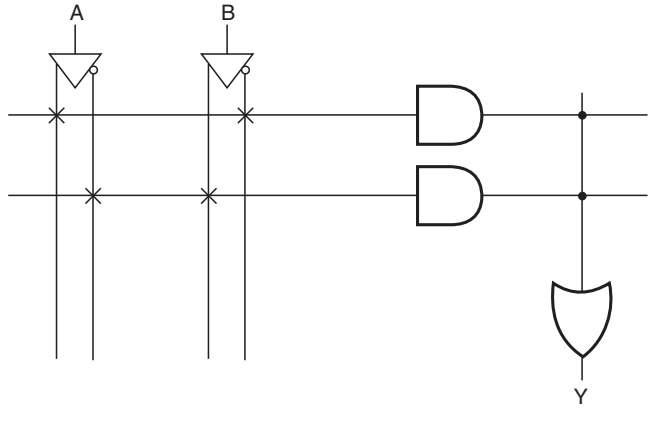


Figure 9.43 Problem 1.

2. Determine the size of PROM required for implementing the following logic circuits.

   (a) 16-to-1 multiplexer;
   (b) four-bit binary adder.

   (a) 1M×1; (b) 512×5

3. Determine the number of programmable interconnections in the following programmable logic devices.

   (a) 1K × 4 PROM;
   (b) PLA device with four input variables, 32 AND gates and four OR gates;
   (c) PAL device with eight input variables, 16 AND gates and four OR gates.

   (a) 4096; (b) 384; (c) 256

4. A and B are two binary variables. The objective is to design a magnitude comparator to produce $A = B$, $A < B$ and $A > B$ outputs. Design a suitable PLD with a PAL-like architecture using anti-fuse based interconnects.

   Fig. 9.44

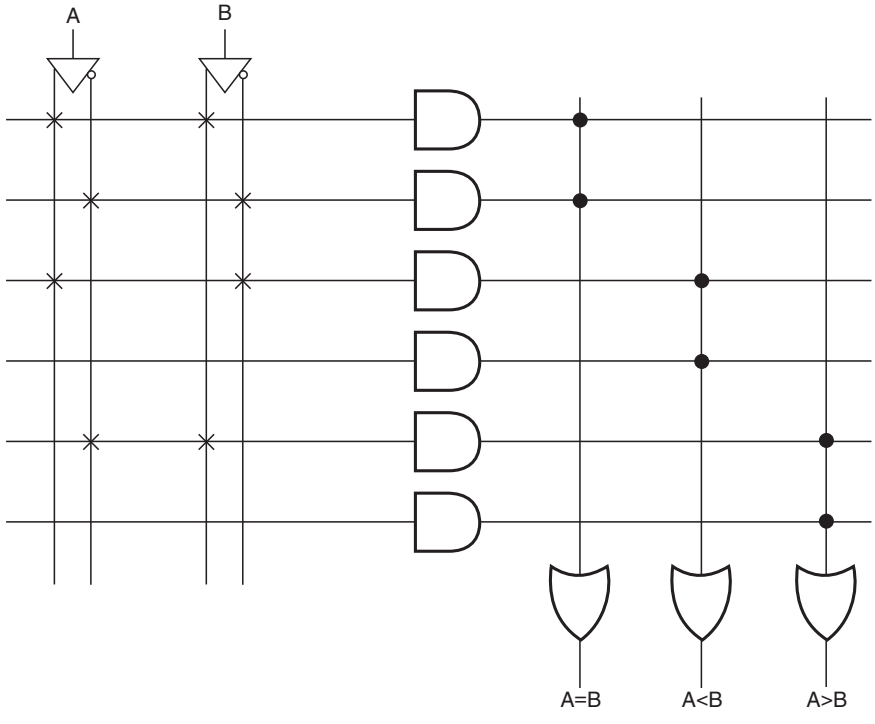


Figure 9.44 Answer to problem 4.

5. Figure 9.45 shows a programmed PAL device using fuse-based interconnects. Examine the logic diagram and determine the logic block implemented by the PLD. A cross (×) represents an unprogrammed interconnection and a dot (•) represents a hard-wired interconnection.

   Full adder

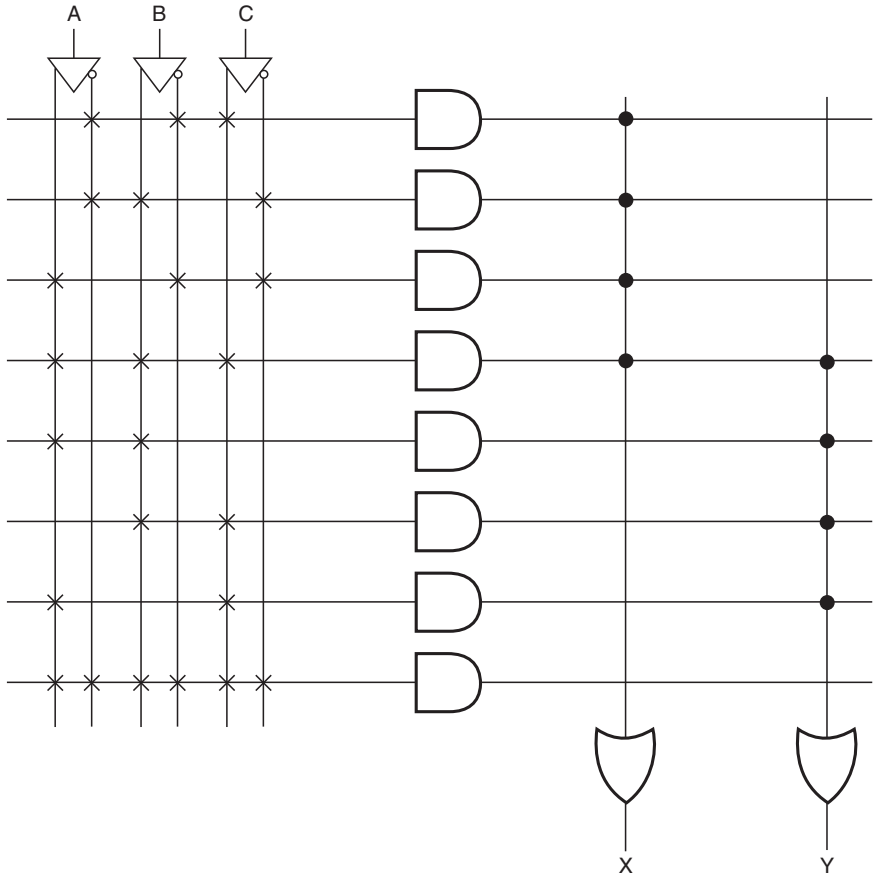


Figure 9.45 Problem 5.

Further Reading

1. Seals, R. C. and Whapshott, G. F. (1997) *Programmable Logic: PLDs and FPGAs*, McGraw-Hill, USA.
2. Dueck, R. (2003) *Digital Design with CPLD Applications and VHDL*, Thomson Delmar Learning, New York, USA.
3. Chartrand, L. (2003) *Digital Fundamentals: Experiments and Concepts with CPLD*, Thomson Delmar Learning, New York, USA.
4. Oldfield, J. and Dorf, R. (1995) *Field Programmable Gate Arrays*, John Wiley & Sons, Inc., New York, USA.
5. Trimberger, S. (Ed.) (1994) *Field Programmable Gate Array Technology*, Kluwer Academic Publishers, MA, USA.
6. Brown, S., Francis, R., Rose, J. and Vranesic, Z. (1992) *Field Programmable Gate Arrays*, Kluwer Academic Publishers, MA, USA.