4 Programmable Logic Devices

The need for getting designs done quickly has led to the creation and evolution of Programmable Logic devices. The idea began from Read Only Memories (ROM) that were just an organized array of gates and has evolved into System On Programmable Chips (SOPC) that use programmable devices, memories and configurable logic all on one chip.

This chapter shows the evolution of basic array structures like ROMs into complex CPLD (Complex Programmable Logic Devices) and FPGAs (Field Programmable Gate Array). This topic can be viewed from different angles, like logic structure, physical design, programming technology, transistor level, software tools, and perhaps even from historic and comerical aspects. However our treatment of this subject is more at the structural level. We discuss gate level structures of ROMs, PLAs, PALs, CPLDs, and FPGAs. The material is at the level needed for understanding configuration and utilization of CPLDs and FPGAs in digital designs.

4.1 Read Only Memories

We present structure of ROMs by showing the implementation of a 3-input 4-output logic function. The circuit with the truth table shown in Figure 4.1 is to be implemented.

4.1.1 Basic ROM Structure

The simplest way to implement the circuit of Figure 4.1 is to form its minterms using AND gates and then OR the appropriate minterms for formation of the four circuit outputs. The circuit requires eight 3-input AND gates and four OR

gates that can take up-to eight inputs. It is easiest to draw this structure in an array format as shown in Figure 4.2.

m:	a	b	c	w	x	y	z
0:	0	0	0	0	0	0	1
1:	0	0	1	1	1	0	0
2:	0	1	0	1	0	1	1
3:	0	1	1	1	0	0	1
4:	1	0	0	0	0	0	1
5:	1	0	1	0	1	1	1
6:	1	1	0	0	0	1	0
7:	1	1	1	0	0	0	0

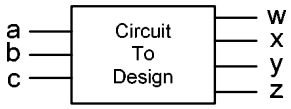

Figure 4.1 A Simple Combinational Circuit

The circuit shown has an array of AND gates and an array of OR gates, that are referred to as the AND-plane and the OR-plane. In the AND-plane all eight minterms for the three inputs, *a*, *b*, and *c* are generated. The OR plane uses only the minterms that are needed for the outputs of the circuit. See for example minterm 7 that is generated in the AND-plane but not used in the OR-plane. Figure 4.3 shows the block diagram of this array structure.

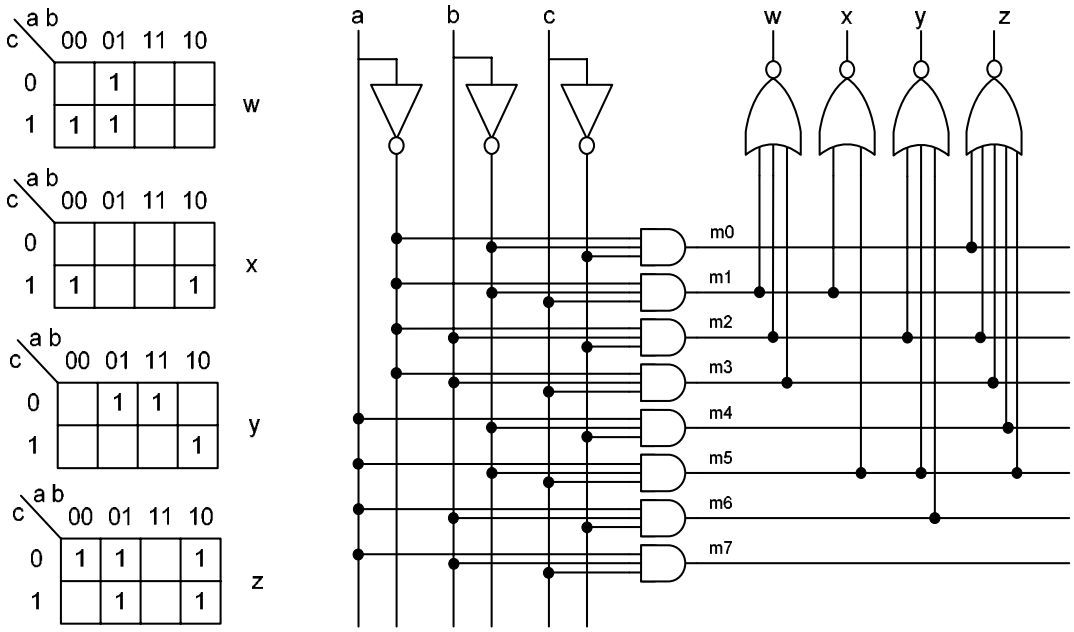

Figure 4.2 AND-OR Implementation

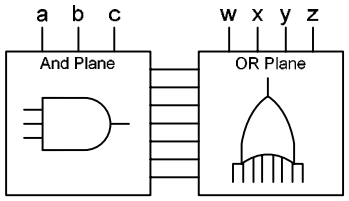

Figure 4.3 AND and OR Planes

4.1.2 NOR Implementation

Since realization of AND and OR gates in most technologies are difficult and generally use more delays and chip area than NAND or NOR implementations, we implement our example circuit using NOR gates. Note that a NOR gate with complemented outputs is equivalent to an OR, and a NOR gate with complemented inputs is equivalent to an AND gate. Our all NOR implementation of Figure 4.4 uses NOR gates for generation of minterms and circuit outputs. To keep functionality and activity levels of inputs and outputs intact, extra inverters are used on the circuit inputs and outputs. These inverters are highlighted in Figure 4.4. Although NOR gates are used, the left plane is still called the AND-plane and the right plane is called the OR-plane.

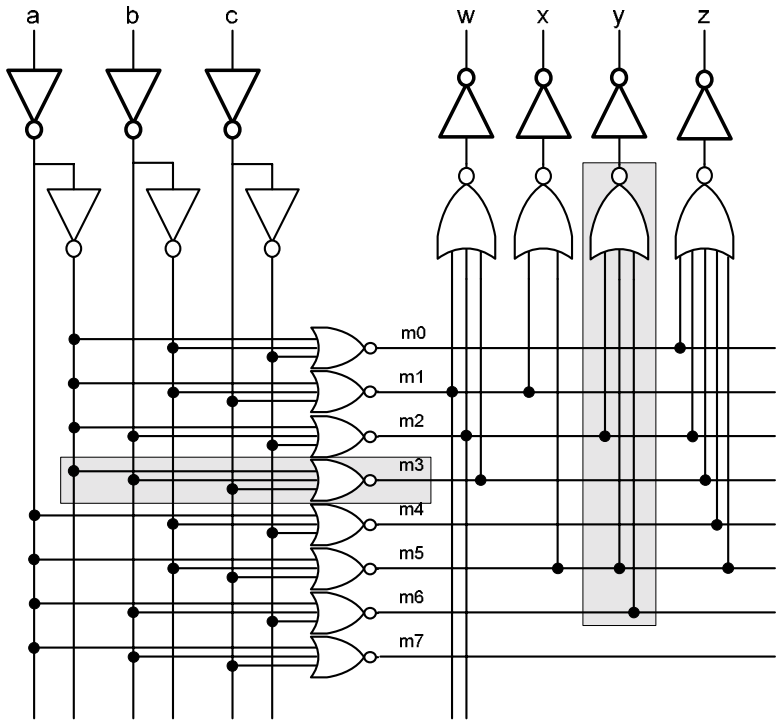

Figure 4.4 All NOR Implementation

4.1.3 Distributed Gates

Hardware implementation of the circuit of Figure 4.4 faces difficulties in routing wires and building gates with large number of inputs. This problem becomes more critical when we are using arrays with tens of inputs. Take for example, a circuit with 16 inputs, which is very usual for combinational circuits. Such a circuit has 64k (2^{16}) minterms. In the AND-plane, wires from circuit inputs must be routed to over 64,000 NOR gates. In the OR-plane, the NOR gates must be large enough for every minterm of the function (over 64,000 minterms) to reach their inputs.

Such an implementation is very slow because of long lines, and takes too much space because of the requirement of large gates. The solution to this problem is to distribute gates along array rows and columns.

In the AND-plane, instead of having a clustered NOR gate for all inputs to reach to, the NOR gate is distributed along the rows of the array. In Figure 4.4, the NOR gate that implements minterm 3 is highlighted. Distributed transistor-level logic of this NOR gate is shown in Figure 4.5. This figure also shows a symbolic representation of this structure.

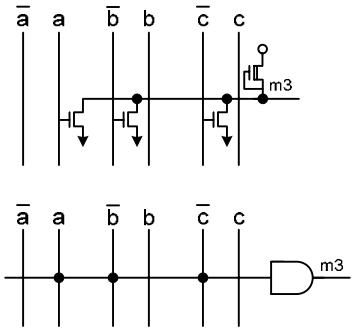

Figure 4.5 Distributed NOR of the AND-plane

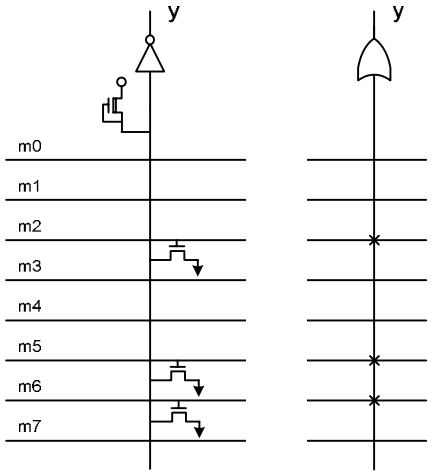

Figure 4.6 Distributed NOR Gate of Output *y*

Likewise, in the OR-plane, instead of having large NOR gates for the outputs of the circuit, transistors of output NOR gates are distributed along the corresponding output columns. Figure 4.6 shows the distributed NOR structure of the *y* output of circuit of Figure 4.4. A symbolic representation of this structure is also shown in this figure.

As shown in Figure 4.5 and Figure 4.6, distributed gates are symbolically represented by gates with single inputs. In each case, connections are made on the inputs of the gate. For the AND-plane, the inputs of the AND gate are *a*, *b*, and *c* forming minterm 3, and for the OR gate of Figure 4.6, the inputs of the gate are *m2*, *m5* and *m6*. The reason for the difference in notations of connections in the AND-plane and the OR-plane (dots versus crosses) becomes clear after the discussion of the next section.

4.1.4 Array Programmability

For the *a*, *b* and *c* inputs, the structure shown in Figure 4.4 implements *w*, *x*, *y* and *z* functions. In this implementation, independent of our outputs, we have generated all minterms of the three inputs. For any other functions other than *w*, *x*, *y* and *z*, we would still generate the same minterms, but use them differently. Hence, the AND-plane with which the minterms are generated can be wired independent of the functions realized. On the contrary, the OR-plane can only be known when the output functions have been determined.

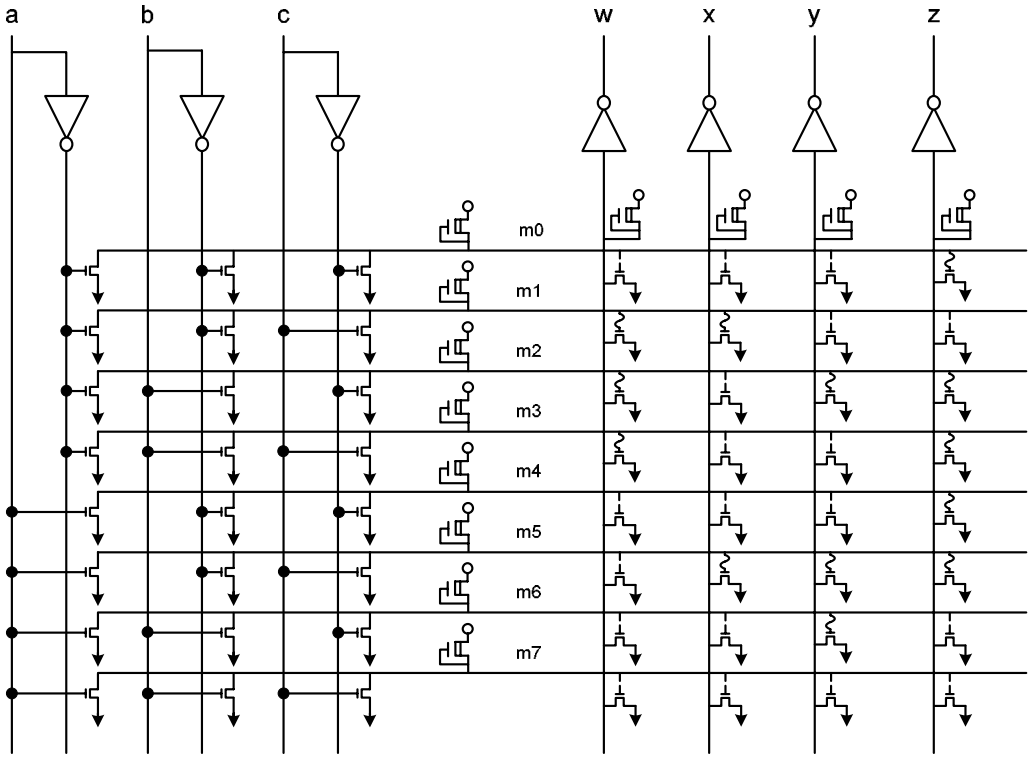

Figure 4.7 Fixed AND-plane, Programmable OR-plane

We can therefore generate a general purpose array logic with all minterms in its AND-plane, and capability of using any number of the minterms for any of the array outputs in its OR-plane. In other words, we want a fixed AND-plane and a programmable (or configurable) OR-plane. As shown in Figure 4.7, transistors for the implementation of minterms in the AND-plane are fixed, but in the OR-plane there are fusible transistors on every output column for every minterm of the AND-plane. For realization of a certain function on an output of this array, transistors corresponding to the used minterms are kept, and the rest are blown to eliminate contribution of the minterm to the output function.

Figure 4.7 shows configuration of the OR-plane for realizing outputs shown in Figure 4.1. Note for example that for output *y*, only transistors on rows *m2*, *m5*, and *m6* are connected and the rest are fused off.

Instead of the complex transistor diagram of Figure 4.7, the notation shown in Figure 4.8 is used for representing the programmability of the configurable arrays. The dots in the AND-plane indicate permanent connections, and the crosses in the OR-plane indicate programmable or configurable connections.

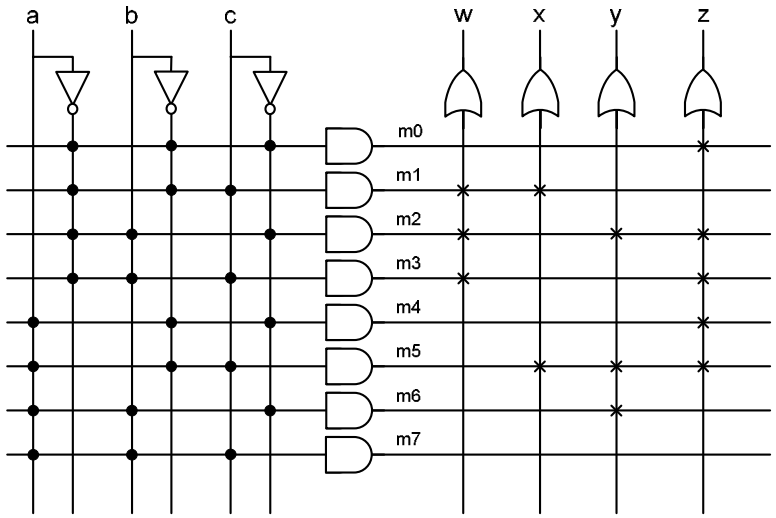

Figure 4.8 Fuse Notation for Configurable Arrays

4.1.5 Memory View

Let us look at the circuit of Figure 4.8 as a black box of three inputs and four outputs. In this circuit, if an input value between 0 and 7 is applied to the *abc* inputs, a 4-bit value is read on the four circuit outputs. For example *abc=011* always reads *wxyz=1001*.

If we consider *abc* as the address inputs and *wxyz* as the data read from *abc* designated address, then the black box corresponding to Figure 4.8 can be regarded as a memory with an address space of 8 words and data of four bits wide. In this case, the fixed AND-plane becomes the memory decoder, and the programmable OR-plane becomes the memory array (see Figure 4.9). Because

this memory can only be read from and not easily written into, it is referred to as Read Only Memory or ROM.

The basic ROM is a one-time programmable logic array. Other variations of ROMs offer more flexibility in programming, but in all cases they can be read more easily than they can be written into.

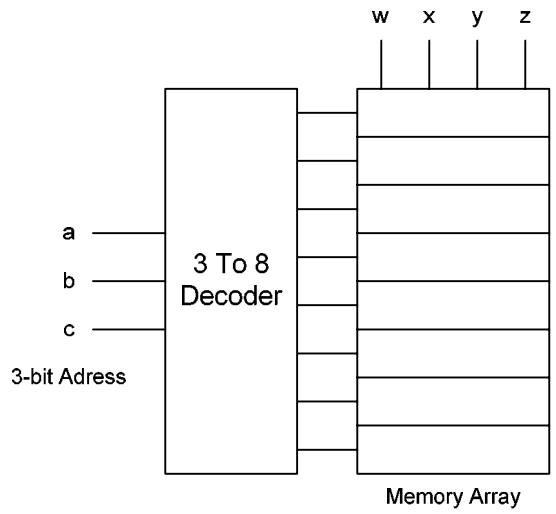

Figure 4.9 Memory View of ROM

4.1.6 ROM Variations

The acronym, ROM is generic and applies to most read only memories. What is today implied by ROM may be ROM, PROM, EPROM, EEPROM or even flash memories. These variations are discussed here.

ROM. ROM is a mask-programmable integrated circuit, and is programmed by a mask in IC manufacturing process. The use of mask-programmable ROMs is only justified when a large volume is needed. The long wait time for manufacturing such circuits makes it a less attractive choice when time-to-market is an issue.

PROM. Programmable ROM is a one-time programmable chip that, once programmed, cannot be erased or altered. In a PROM, all minterms in the AND-plane are generated, and connections of all AND-plane outputs to OR-plane gate inputs are in place. By applying a high voltage, transistors in the OR-plane that correspond to the minterms that are not needed for a certain output are burned out. Referring to Figure 4.7, a fresh PROM has all transistors in its OR-plane connected. When programmed, some will be fused out permanently. Likewise, considering the diagram of Figure 4.8, an un-programmed PROM has X's in all wire crossings in its OR-plane.

EPROM. An Erasable PROM is a PROM that once programmed, can be completely erased and reprogrammed. Transistors in the OR-plane of an EPROM have a normal gate and a floating gate as shown in Figure 4.10. The non-floating gate is a normal NMOS transistor gate, and the floating-gate is surrounded by insulating material that allows an accumulated charge to remain on the gate for a long time.

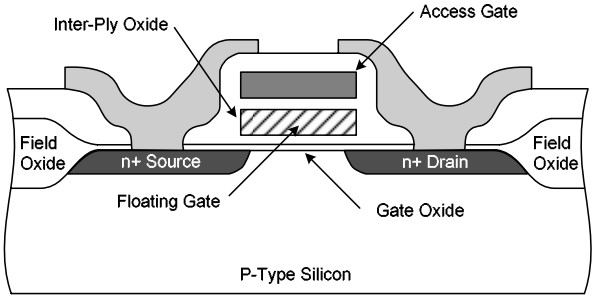

Figure 4.10 Floating Gate

When not programmed, or programmed as a '**1**', the floating gate has no extra charge on it and the transistor is controlled by the non-floating gate (access gate). To fuse-out a transistor, or program a '**0**' into a memory location, a high voltage is applied to the access gate of the transistor which causes accumulation of negative charge in the floating-gate area. This negative charge prevents logic **1** values on the access gate from turning on the transistor. The transistor, therefore, will act as an unconnected transistor for as long as the negative charge remains on its floating-gate.

To erase an EPROM it must be exposed to ultra-violate light for several minutes. In this case, the insulating materials in the floating-gates become conductive and these gates start loosing their negative charge. In this case, all transistors return to their normal mode of operation. This means that all EPROM memory contents become **1**, and ready to be reprogrammed.

Writing data into an EPROM is generally about a 1000 times slower than reading from it. This is while not considering the time needed for erasing the entire EPROM.

EEPROM. An EEPROM is an EPROM that can electrically be erased, and hence the name: Electrically Erasable Programmable ROM. Instead of using ultra-violate to remove the charge on the non-floating gate of an EPROM transistor, a voltage is applied to the opposite end of the transistor gate to remove its accumulated negative charge. An EEPROM can be erased and reprogrammed without having to remove it. This is useful for reconfiguring a design, or saving system configurations. As in EPROMs, EEPROMs are non-volatile memories. This means that they save their internal data while not powered. In order for memories to be electrically erasable, the insulating material surrounding the floating-gate must be much thinner than those of the EPROMS. This makes the number of times EEPROMs can be reprogrammed much less than that of EPROMs and in the order of 10 to 20,000. Writing into a byte of an EEPROM is about 500 times slower than reading from it.

Flash Memory. Flash memories are large EEPROMs that are partitioned into smaller fixed-size blocks that can independently be erased. Internal to a system, flash memories are used for saving system configurations. They are used in digital cameras for storing pictures. As external devices, they are used for temporary storage of data that can be rapidly retrieved.

Various forms of ROM are available in various sizes and packages. The popular 27xxx series EPROMs come in packages that are organized as byte addressable memories. For example, the 27256 EPROM has 256K bits of memory that are arranged into 32K bytes. This package is shown in Figure 4.11.

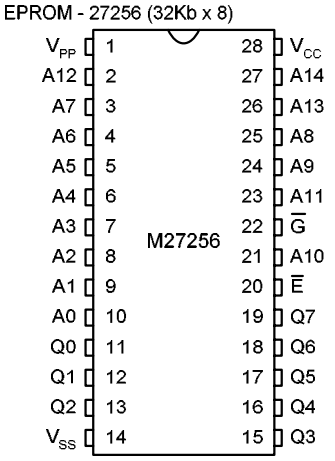

Figure 4.11 27256 EPROM

The 27256 EPROM has a Vpp pin that is used for the supply input during read-only operations and is used for applying programming voltage during the programming phase. The 15 address lines address 256K of 8-bit data that are read on to *O7* to *O0* outputs. Active low *CS* and *OE* are for three-state control of the outputs and are used for cascading EPROMs and/or output bussing.

EPROMs can be cascaded for word length expansion, address space expansion or both. For example, a 1Meg 16-bit word memory can be formed by use of a four by two array of 27256s.

4.2 Programmable Logic Arrays

The price we are paying for the high degree of flexibility of ROMs is the large area occupied by the AND-plane that forms every minterm of the inputs of the ROM. PLAs (Programmable Logic Arrays) constitutes an alternative with less flexibility and less use of silicon. For this discussion we look at ROMs as logic circuits as done in the earlier parts of Section 4.1, and not the memory view of the later parts of this section.

For illustrating the PLA structure, we use the 3-input, 4-output example circuit of Figure 4.1. The AND-OR implementation of this circuit that is shown

in Figure 4.2 led to the ROM structure of Figure 4.8, in which minterms generated in the AND-plane are used for function outputs in the OR-plane.

An easy step to reduce the area used by the circuit of Figure 4.8 is to implement only those minterms that are actually used. In this example, since minterm 7 is never used, the last row of the array can be completely eliminated. In large ROM structures, there will be a much larger percentage of unused minterms that can be eliminated. Furthermore, if instead of using minterms, we minimize our output functions and only implement the regained product terms we will be able to save even more rows of the logic array.

Figure 4.12 shows Karnaugh maps for minimization of *w*, *x*, *y* and *z* outputs of table of Figure 4.1. In this minimization sharing product terms between various outputs is particularly emphasized.

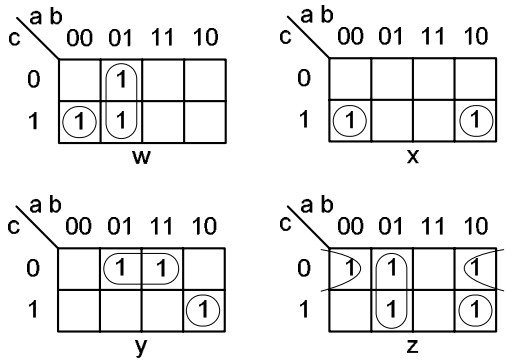

Figure 4.12 Minimizing Circuit of Figure 4.1

Resulting Boolean expressions for the outputs of circuit described by the tables of Figure 4.12 are shown in Figure 4.13. Common product terms in these expressions are vertically aligned.

$$w = \bar{a}b + \bar{a}\bar{b}c$$

$$x = \quad \bar{a}\bar{b}c + a\bar{b}c$$

$$y = \quad\quad a\bar{b}c + b\bar{c}$$

$$z = \bar{a}b + \quad a\bar{b}c \quad + \bar{b}\bar{c}$$

Figure 4.13 Minimized Boolean Expressions

Implementation of *w*, *x*, *y* and *z* functions of *a*, *b* and *c* inputs in an array format using minimized expressions of Figure 4.13 is shown in Figure 4.14. This array uses five rows that correspond to the product terms of the four output functions. Comparing this with Figure 4.8, we can see that we are using less number of rows by generating only the product terms that are needed and not every minterm.

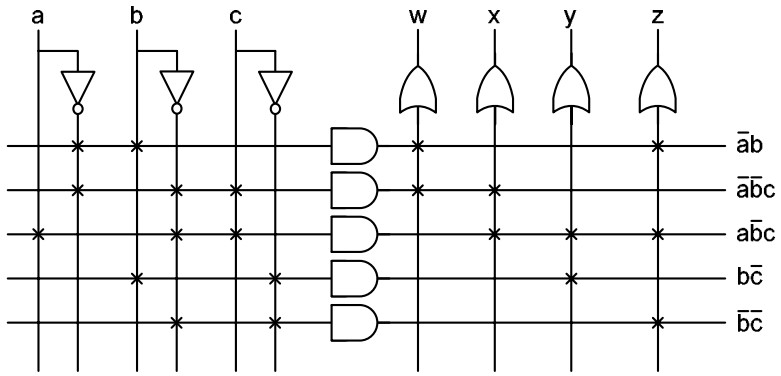

Figure 4.14 PLA Implementation

The price we are paying for the area gained in the PLA implementation of Figure 4.14 is that we now have to program both AND and OR planes. In Figure 4.14 we use X's in both planes where in Figure 4.8 dots are used in the fixed AND-plane and X's in the programmable OR-plane.

While ROM structures are used for general purpose configurable packages, PLAs are mainly used as structured array of hardware for implementing on-chip logic. A ROM is an array logic with fixed AND-plane and programmable OR-plane, and PLA is an array with programmable AND-plane and programmable OR-plane.

A configurable array logic that sits between a PLA and a ROM is one with a programmable AND-plane and a fixed OR-plane. This logic structure was first introduced by Monilitic Memories Inc. (MMI) in the late 1970s and because of its similarity to PLA was retuned to PAL or Programmable Array Logic.

The rationale behind PALs is that outputs of a large logic function generally use a limited number of product terms and the capability of being able to use all product terms for all function outputs is in most cases not utilized. Fixing the number of product terms for the circuit outputs significantly improves the speed of PALs.

4.2.1 PAL Logic Structure

In order to illustrate the logical organization of PALs, we go back to our 3-input, 4-output example of Figure 4.1. Figure 4.15 shows PAL implementation of this circuit. This circuit uses *w*, *x*, *y* and *z* expressions shown in Figure 4.13. Recall that these expressions are minimal realizations for the outputs of our example circuit and are resulted from the k-maps of Figure 4.12.

The PAL structure of Figure 4.15 has a programmable AND-plane and a fixed OR-plane. Product terms are formed in the AND-plane and three such terms are used as OR gate inputs in the OR-plane. This structure allows a maximum of three product terms per output.

Implementing expressions of Figure 4.13 is done by programming fuses of the AND-plane of the PAL. The *z* output uses all three available product terms and all other outputs use only two.

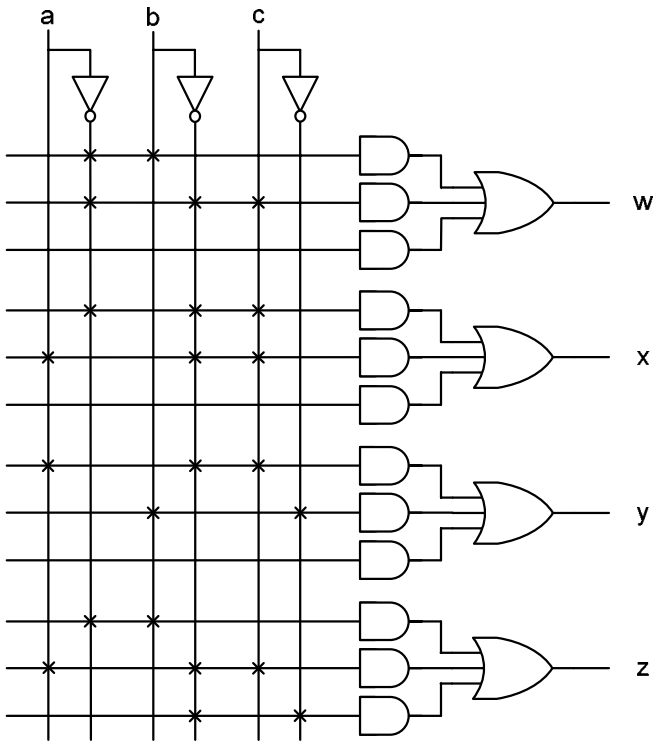

Figure 4.15 PAL Implementation

4.2.2 Product Term Expansion

The limitation on the number of product terms per output in a PAL device can be overcome by providing feedbacks from PAL outputs back into the AND-plane. These feedbacks are used in the AND-plane just like regular inputs of the PAL. Such a feedback allows ORing a subset of product terms of a function to be fed back into the array to further be ORed with the remaining product terms of the function.

Consider for example, PAL implementation of expression *w* shown below:

$$w = \overline{a} \cdot b \cdot \overline{c} + \overline{a} \cdot \overline{b} \cdot c + a \cdot \overline{b} + a \cdot b \cdot c$$

Let us assume that this function is to be implemented in a 3-input PAL with three product terms per output and with outputs feeding back into the AND-plane, as shown in Figure 4.16.

The partial PAL shown in this figure allows any of its outputs to be used as a circuit primary output or as a partial sum-of-products to be completed by ORing more product terms to it. For implementation of expression *w*, the first three product terms are generated on the o_1. The structure shown does not allow the last product term ($a \cdot b \cdot c$) to be ORed on the same output. Therefore, the feedback from this output is used as an input into the next group of product terms. The circled X connection in this figure causes o_1 to be used as

an input into the o_2 group. The last product terms ($a.b.c$) is generated in the AND-plane driving the o_2 output and is ORed with o_1 using the OR-gate of the o_2 output. Expression w is generated on o_2. Note that the feedback of o_2 back into the AND-plane does exist, but not utilized.

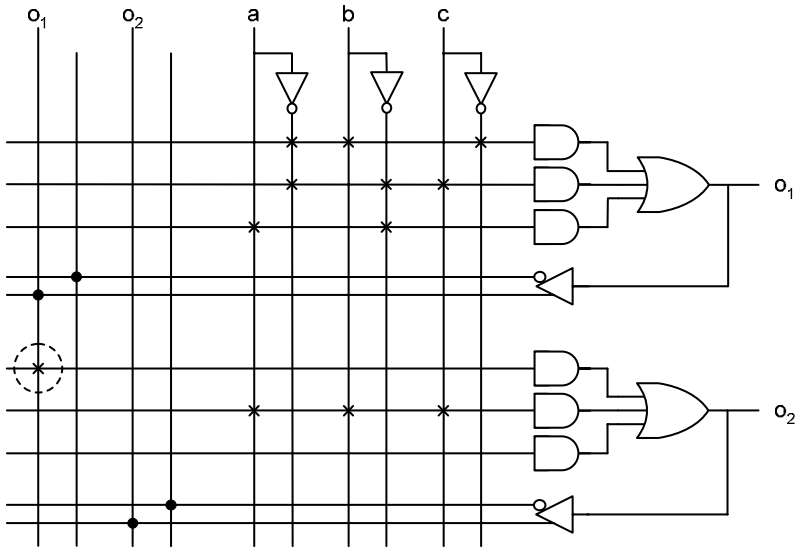

Figure 4.16 A PAL with Product Term Expandability

4.2.3 Three-State Outputs

A further improvement to the original PAL structure of Figure 4.15 is done by adding three-state controls to its outputs as shown in the partial structure of Figure 4.17.

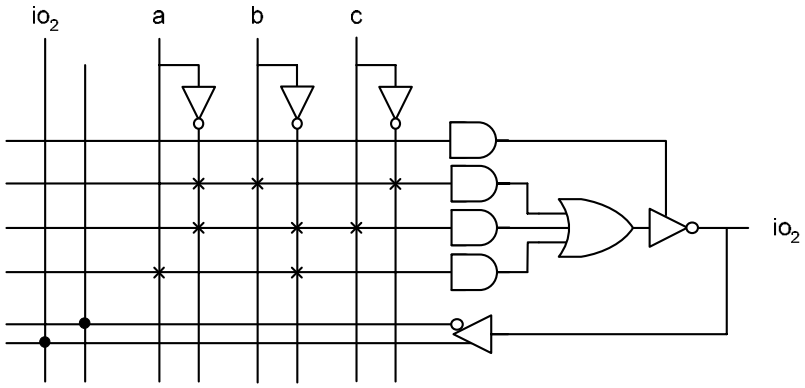

Figure 4.17 PAL Structure with Three Output Control

In addition to the feedback from the output, this structure has two more advantages. First, the pin used as output or partial sum-of-products terms can

also be used as input by turning off the three-state gate that drives it. Note that the lines used for feeding back outputs into the AND-plane in Figure 4.16, become connections from the io_2 input into the AND-plane. The second advantage of this structure is that when *io2* is used as output it becomes a three-state output that is controlled by a programmable product term.

Instead of using a three-state inverting buffer, an XOR gate with three-state output and a fusible input (see Figure 4.18) provides output polarity control when the bi-directional io_2 port is used as output

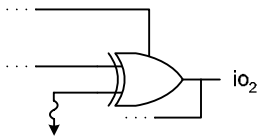

Figure 4.18 Output Inversion Control

4.2.4 Registered Outputs

A major advantage of PALs over PLAs and ROMs is the capability of incorporating registers into the logic structure. Where registers can only be added to the latter two structures on their inputs and outputs, registers added to PAL arrays become more integrated in the input and output of the PAL logic.

As an example structure, consider the registered output of Figure 4.19. The input/output shown can be used as a registered output with three-state, as a two-state output, as a registered feedback into the logic array, or as an input into the AND-plane.

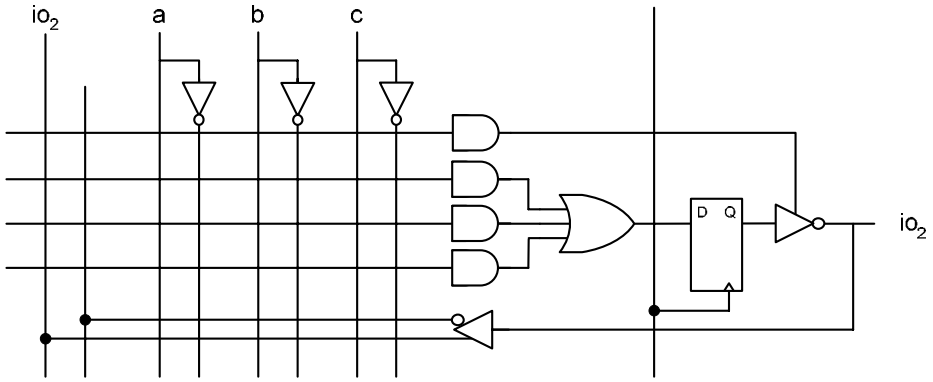

Figure 4.19 Output Inversion Control

A further enhancement to this structure provides logic for bypassing the output flip-flop when its corresponding I/O pin is being used as output. This way, PAL outputs can be programmed as registered or combinational pins.

Other enhancements to the register option include the use of asynchronous control signals for the flip-flop, direct feedback from the flip-flop into the array,

and providing a programmable logic function for the flip-flop output and the feedback line.

4.2.5 Commercial Parts

PAL is a trademark of American Micro Devices Inc. More generically, these devices are referred to as PLDs or programmable logic devices. A variation of the original PAL or a PLD that is somewhat different from a PAL is GAL (Generic Array Logic). The inventor of GAL is the Lattice Semiconductor Inc. GALs are electrically erasable; otherwise have a similar logical structure to PALs. By ability to bypass output flip-flops, GALs can be configured as combinational or sequential circuits. To familiarize readers with some actual parts, we discuss one of Altera's PLD devices.

Altera Classic EPLD Family.
Altera Corporation's line of PLDs is its Classic EPLD Family. These devices are EPROM based and have 300 to 900 usable gates depending on the specific part. These parts come in 24 to 68 pin packages and are available in dual in-line package (DIP), plastic J-lead chip carrier (PLCC), pin-grid array (PGA), and small-outline integrated circuit (SOIC) packages. The group of product terms that are ORed together are referred to as a Macrocell, and the number of Macrocells varies between 16 and 48 depending on the device. Each Macrocell has a programmable register that can be programmed as a D, T, JK and SR flip-flop with individual clear and clock controls.

These devices are fabricated on CMOS technology and are TTL compatible. They can be used with other manufacturers PAL and GAL parts. The EP1810 is the largest of these devices that has 900 usable gates, 48 Macrocells, and a maximum of 64 I/O pins. Pin-to-pin logic delay of this part is 20 ns and it can operate with a maximum frequency of 50 MHz. The architecture of this and other Altera's Classic EPLDs includes Macrocells, programmable registers, output enable or clock select, and a feedback select.

Macrocells. Classic macrocells, shown in Figure 4.20, can be individually configured for both sequential and combinatorial logic operation. Eight product terms form a programmable-AND array that feeds an OR gate for combinatorial logic implementation. An additional product term is used for asynchronous clear control of the internal register; another product term implements either an output enable or a logic-array-generated clock. Inputs to the programmable-AND array come from both the true and complement signals of the dedicated inputs, feedbacks from I/O pins that are configured as inputs, and feedbacks from macrocell outputs. Signals from dedicated inputs are globally routed and can feed the inputs of all device macrocells. The feedback multiplexer controls the routing of feedback signals from macrocells and from I/O pins.

The eight product terms of the programmable-AND array feed the 8-input OR gate, which then feeds one input to an XOR gate. The other input to the XOR gate is connected to a programmable bit that allows the array output to be inverted. This gate is used to implement either active-high or active-low logic, or De Morgan's inversion to reduce the number of product terms needed to implement a function.

Programmable Registers. To implement registered functions, each macrocell register can be individually programmed for D, T, JK, or SR operation. If necessary, the register can be bypassed for combinatorial operation. Registers have an individual asynchronous clear function that is controlled by a dedicated product term. These registers are cleared automatically during power-up. In addition, macrocell registers can be individually clocked by either a global clock or any input or feedback path to the AND array. Altera's proprietary programmable I/O architecture allows the designer to program output and feedback paths for combinatorial or registered operation in both active-high and active-low modes.

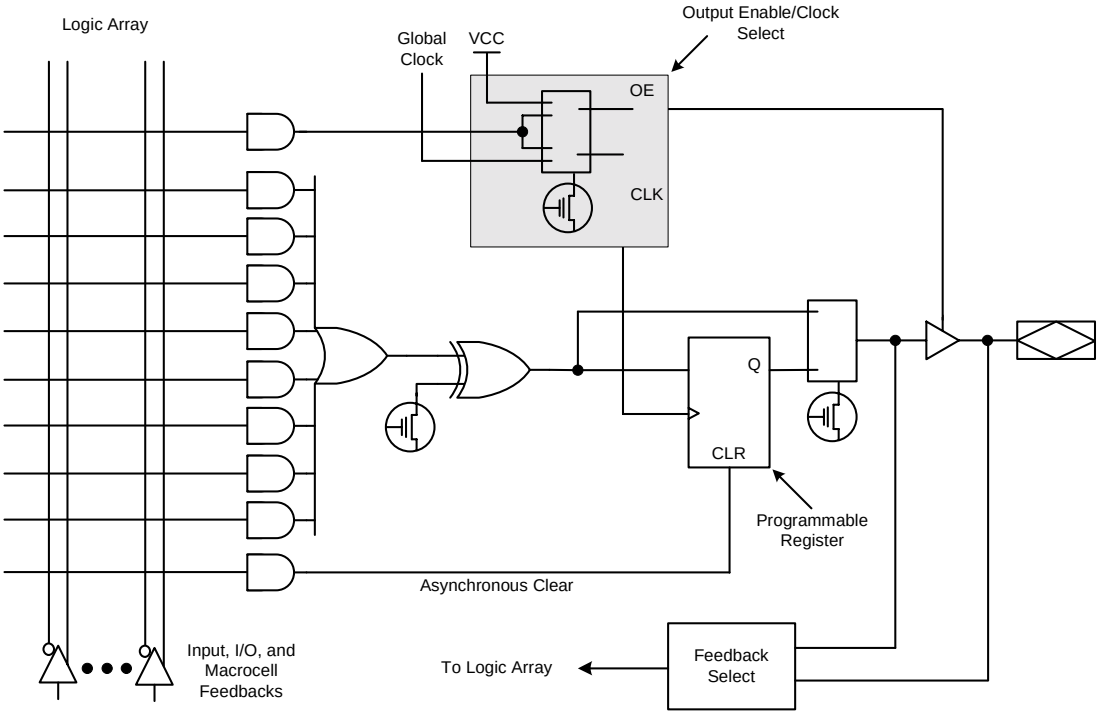

Figure 4.20 Altera's Classic Mecrocell

Output Enable / Clock Select. The box shown in the upper part of Figure 4.20 allows two modes of operations for output and clocking of a Classic macrocell. Figure 4.21 shows these two operating modes (Modes 0 and 1) that are provided by the output enable/clock (OE/CLK) select. The OE/CLK select, which is controlled by a single programmable bit, can be individually configured for each macrocell.

In Mode 0, the tri-state output buffer is controlled by a single product term. If the output enable is high, the output buffer is enabled. If the output enable is low, the output has a high-impedance value. In Mode 0, the macrocell flip-flop is clocked by its global clock input signal.

In Mode 1, the output enable buffer is always enabled, and the macrocell register can be triggered by an array clock signal generated by a product term. This mode allows registers to be individually clocked by any signal on the AND

array. With both true and complement signals in the AND array, the register can be configured to trigger on a rising or falling edge. This product-term-controlled clock configuration also supports gated clock structures.

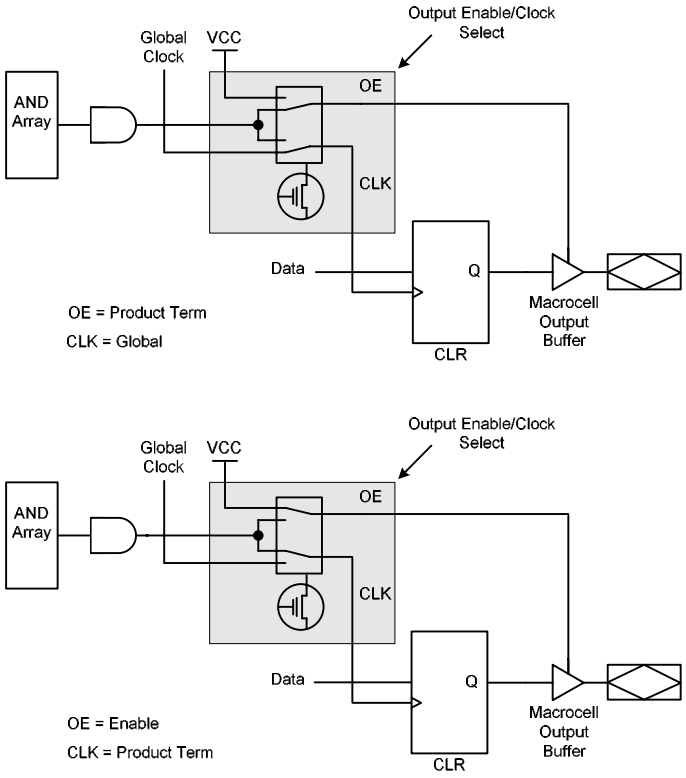

Figure 4.21 Macrocell OE/CLK Select (Upper: Mode 0, Lower: Mode 1)

Feedback Select. Each macrocell in a Classic device provides feedback selection that is controlled by the feedback multiplexer. This feedback selection allows the designer to feed either the macrocell output or the I/O pin input associated with the macrocell back into the AND array. The macrocell output can be either the Q output of the programmable register or the combinatorial output of the macrocell. Different devices have different feedback multiplexer configurations. See Figure 4.22.

EP1810 macrocells can have either of two feedback configurations: quadrant or dual. Most macrocells in EP1810 devices have a quadrant feedback configuration; either the macrocell output or I/O pin input can feed back to other macrocells in the same quadrant. Selected macrocells in EP1810 devices have a dual feedback configuration: the output of the macrocell feeds back to other macrocells in the same quadrant, and the I/O pin input feeds back to all macrocells in the device. If the associated I/O pin is not used, the macrocell output can optionally feed all macrocells in the device. In this case, the output of the macrocell passes through the tri-state buffer and uses the feedback path between the buffer and the I/O pin.

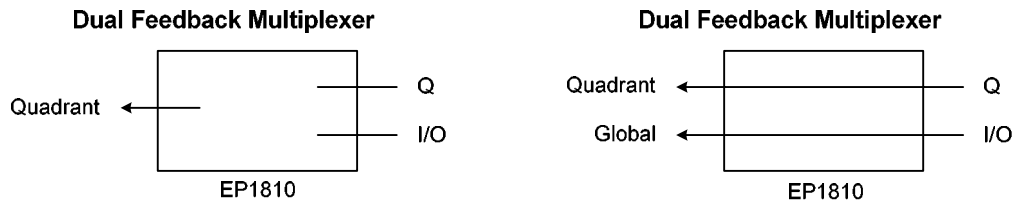

Figure 4.22 Classic Feedback Multiplexer Configurations

Altera "Classic EPLD Family" datasheet describes other features of EP1810 and other Altera's EPLDs. This document has an explanation of device timings of these EPLDs.

4.3 Complex Programmable Logic Devices

The next step up in the evolution and complexity of programmable devices is the CPLD, or Complex PLD. Extending PLDs by making their AND-plane larger and having more macrocells in order to be able to implement larger and more complex logic circuits would face difficulties in speed and chip area utilization. Therefore, instead of simply making these structures larger, CPLDs are created that consist of multiple PLDs with programmable wiring channels between the PLDs. Figure 4.23 shows the general block diagram of a CPLD.

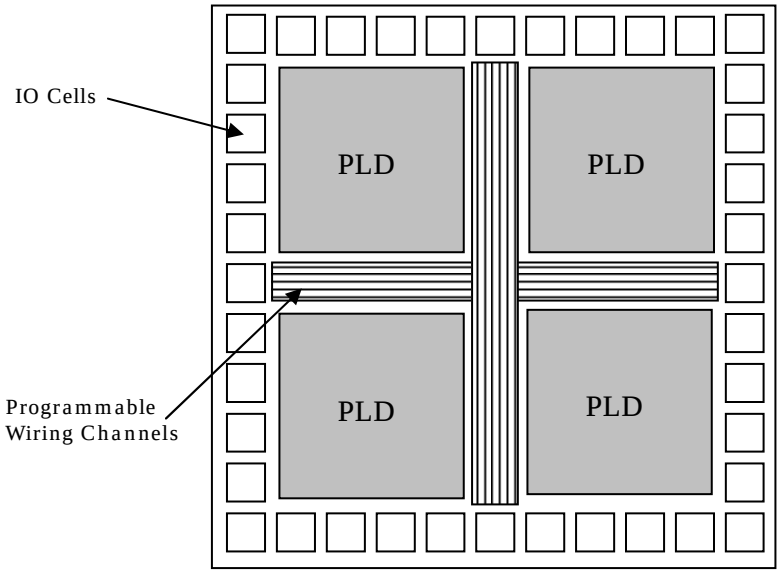

Figure 4.23 CPLD Block Diagram

The approach taken by different manufacturers for implementation of their CPLDs are different. As a typical CPLD we discuss Altera's EPM7128S that is a member of this manufacturer's MAX 7000 Programmable Device Family.

4.3.1 Altera's MAX 7000S CPLD

A member of Altera's MAX 7000 S-series is the EPM7128S CPLD. This is an EEPROM-based programmable logic device with in-system programmability feature through its JTAG interface. Logic densities for the MAX family of CPLDs range from 600 to 5,000 usable gates and the EPM7128S is a mid-rage CPLD in this family with 2,500 usable gates. Note that these figures are 2 to 4 times larger than those of the PLDs from Altera.

The EPM7128s is available in plastic J-lead chip carrier (PLCC), ceramic pin-grid array (PGA), plastic quad flat pack (PQFP), power quad flat pack (RQFP), and 1.0-mm thin quad flat pack (TQFP) packages. The maximum frequency of operation of this part is 147.1 MHz, and it has a propagation delay of 6 ns. This part can operate with 3.3 V or 5.0 V.

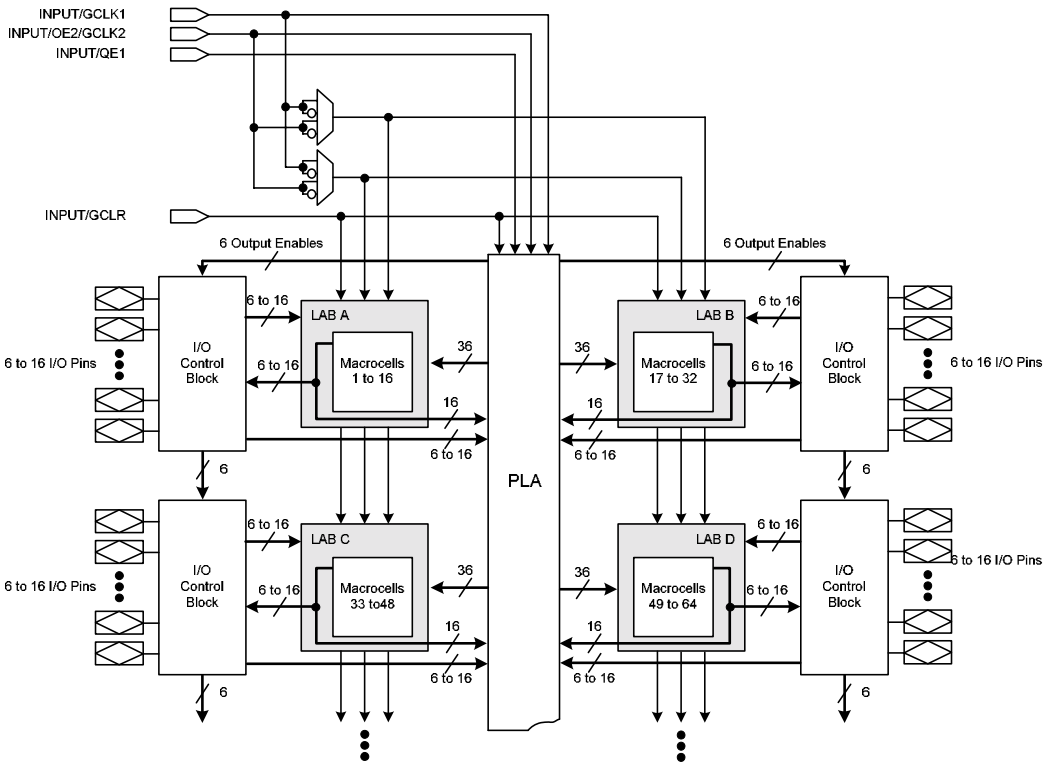

Figure 4.24 Altera's CPLD Architecture

This CPLD has 8 PLDs that are referred to as Logic Array Blocks (LABs). Each LAB has 16 macrocells, making the total number of its macrocells 128. The LABs are linked by a wiring channel that is referred to as the Programmable Interconnect Array (PIA). The macrocells include hardware for expanding product terms by linking several macrocells. The overall architecture of this part is shown in Figure 4.24. In what follows, blocks shown in this figure will be briefly described.

Logic Array Blocks. The EPM7128S has 8 LABs (4 shown in Figure 4.24) that are linked by the PIA global wiring channel. In general, a LAB has the same structure as a PLD described in the previous section. Multiple LABs are linked together via the PIA global bus that is fed by all dedicated inputs, I/O pins, and macrocells. Signals included in a LAB are 36 signals from the PIA that are used for general logic inputs, global controls that are used for secondary register functions, and direct input paths from I/O pins to the registers.

Macrocells. The MAX 7000 macrocell can be individually configured for either sequential or combinatorial logic operation. The macrocell consists of three functional blocks: the logic array, the product-term select matrix, and the programmable register. The macrocell for EPM7128S is shown in Figure 4.25.

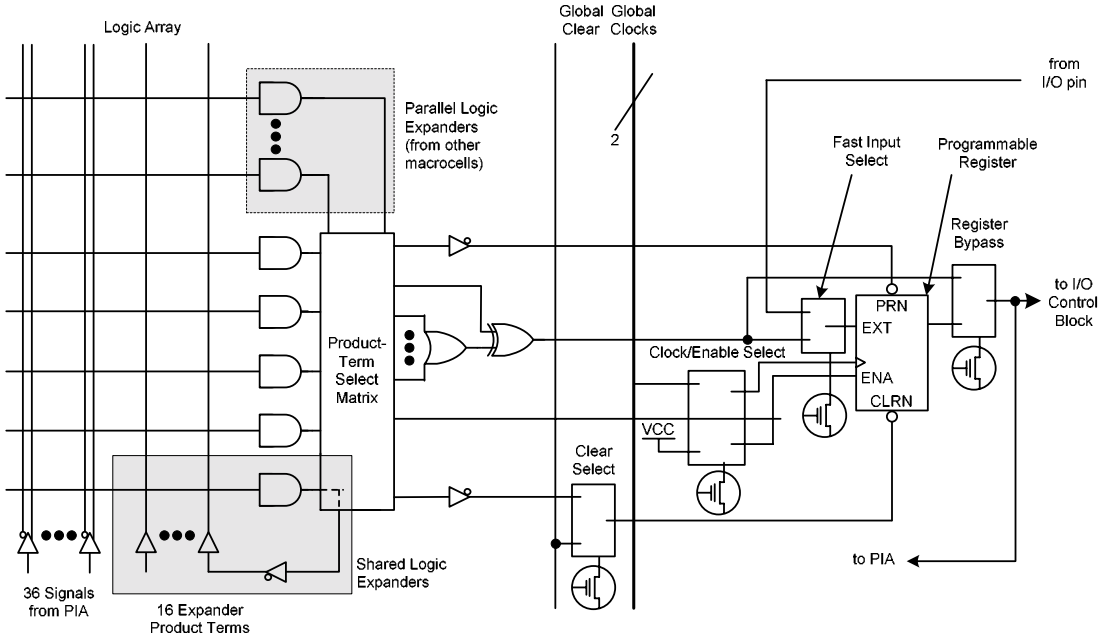

Figure 4.25 MAX 7000 EPM7128S Macrocell

Combinatorial logic is implemented in the logic array, which provides five product terms per macrocell. The product-term select matrix allocates these product terms for use as either primary logic inputs (to the OR and XOR gates) to implement combinatorial functions, or as secondary inputs to the macrocell's register clear, preset, clock, and clock enable control functions. Two kinds of expander product terms ("expanders") are available to supplement macrocell logic resources: Shareable expanders, which are inverted product terms that are fed back into the logic array, and Parallel expanders, which are product terms borrowed from adjacent macrocells.

For registered functions, each macrocell flip-flop can be individually programmed to implement D, T, JK, or SR operation with programmable clock control. The flip-flop can be bypassed for combinatorial operation. Each programmable register can be clocked by a global clock signal and enabled by an active-high clock enable, and by an array clock implemented with a product term. Each register also supports asynchronous preset and clear functions. As

shown in Figure 4.25, the product-term select matrix allocates product terms to control these operations.

Expander Product Terms. Although most logic functions can be implemented with the five product terms available in each macrocell, the more complex logic functions require additional product terms. Another macrocell can be used to supply the required logic resources; however, the MAX 7000 architecture also allows both shareable and parallel expander product terms ("expanders") that provide additional product terms directly to any macrocell in the same LAB.

Each LAB has 16 shareable expanders that can be viewed as a pool of uncommitted single product terms (one from each macrocell) with inverted outputs that feed back into the logic array. Each shareable expander can be used and shared by any or all macrocells in the LAB to build complex logic functions. Figure 4.26 shows how shareable expanders can feed multiple macrocells.

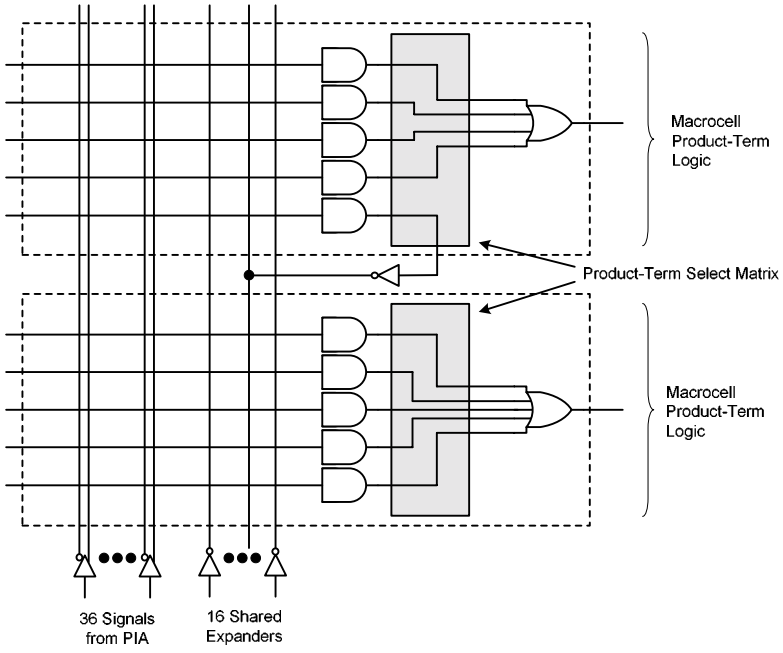

Figure 4.26 MAX 7000 Sharable Expanders

Parallel expanders are unused product terms that can be allocated to a neighboring macrocell. Parallel expanders allow up to 20 product terms to directly feed the macrocell OR logic, with five product terms provided by the macrocell and 15 parallel expanders provided by neighboring macrocells in the LAB.

Two groups of 8 macrocells within each LAB (e.g., macrocells 1 through 8 and 9 through 16) form two chains to lend or borrow parallel expanders. A macrocell borrows parallel expanders from lower numbered macrocells. For example, Macrocell 8 can borrow parallel expanders from Macrocell 7, from Macrocells 7 and 6, or from Macrocells 7, 6, and 5. Within each group of 8, the lowest-numbered macrocell can only lend parallel expanders and the highest-

numbered macrocell can only borrow them. Figure 4.27 shows how parallel expanders can be borrowed from a neighboring macrocell.

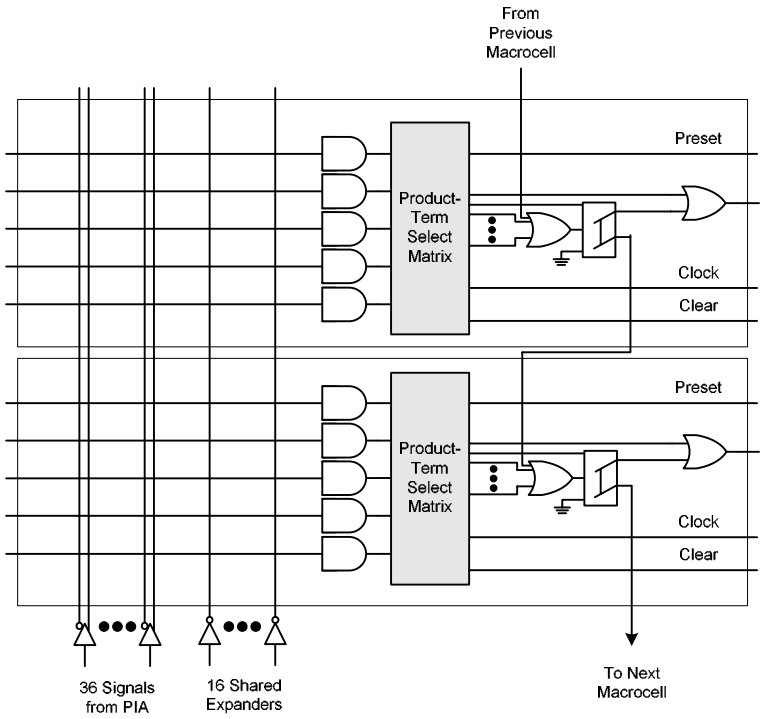

Figure 4.27 MAX 7000 Parallel Expanders

Programmable Interconnect Array. Logic is routed between LABs via the programmable interconnect array (PIA). This global bus is a programmable path that connects any signal source to any destination on the device. All MAX 7000 dedicated inputs, I/O pins, and macrocell outputs feed the PIA, which makes the signals available throughout the entire device. Only the signals required by each LAB are actually routed from the PIA into the LAB.

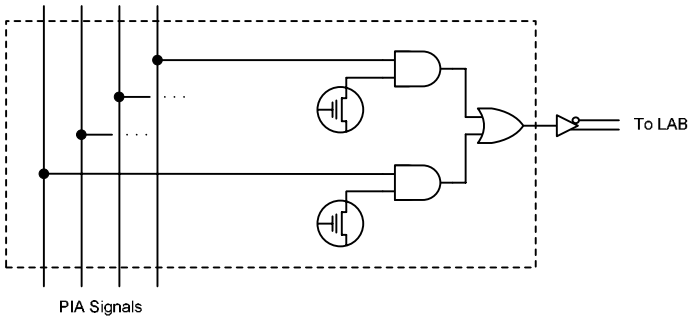

Figure 4.28 PIA Routing in MAX 7000 Devices

Figure 4.28 shows how the PIA signals are routed into the LAB. An EEPROM cell controls one input to a 2-input AND gate, which selects a PIA

signal to drive into the LAB. The PIA has a fixed delay that eliminates skew between signals and makes timing performance easy to predict.

I/O Control Blocks. The I/O control block allows each I/O pin to be individually configured for input, output, or bidirectional operation. All I/O pins have a tri-state buffer that is individually controlled by one of the global output enable signals or directly connected to ground or VCC. Figure 4.29 shows the I/O control block for the EPM7128S of the MAX 7000 family. The I/O control block shown here has six global output enable signals that are driven by the true or complement of two output enable signals, a subset of the I/O pins, or a subset of the I/O macrocells.

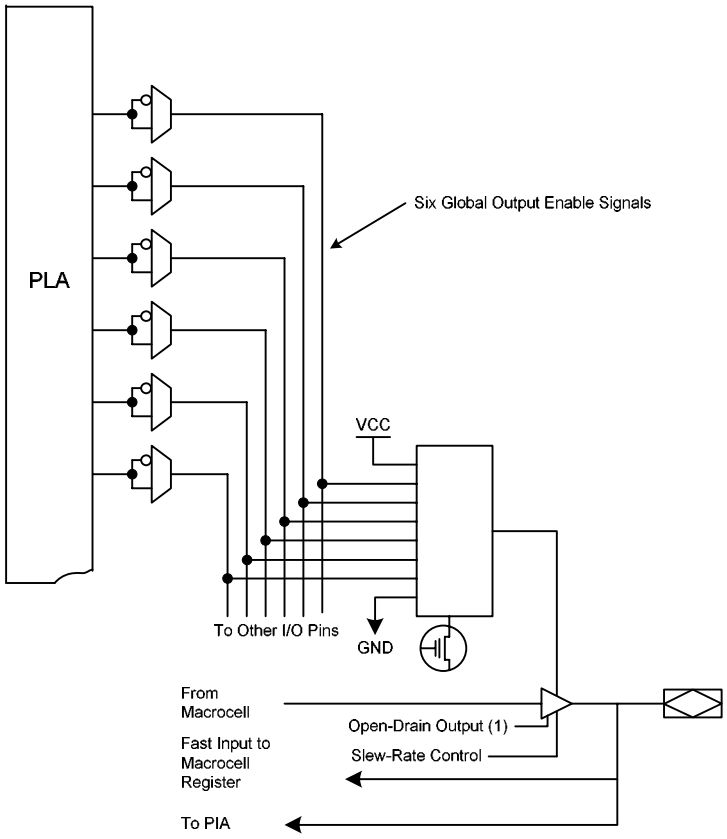

Figure 4.29 I/O Control Block for EPM7128S

When the tri-state buffer control is connected to ground, the output is tri-stated (high impedance) and the I/O pin can be used as a dedicated input. When the tri-state buffer control is connected to VCC, the output is enabled. The MAX 7000 architecture provides dual I/O feedback, in which macrocell and pin feedbacks are independent. When an I/O pin is configured as an input, the associated macrocell can be used for buried logic.

Because of the logic nature of this book, the above discussion concentrated on the logical architecture of the EPM7128S member of the MAX 7000S family. Other details of this part including its timing parameters, programming alternatives, and its In-System Programmability (ISP) features can be found in

Altera's *"MAX 7000 Programmable Logic Device Family"* datasheet. In addition to the EPM7128S device that we discussed, this datasheet has details about other members of the MAX 7000 CPLD family.

4.4 Field Programmable Gate Arrays

A more advanced programmable logic than the CPLD is the Field Programmable Gate Array (FPGA). An FPGA is more flexible than CPLD, allows more complex logic implementations, and can be used for implementation of digital circuits that use equivalent of several Million logic gates.

An FPGA is like a CPLD except that its logic blocks that are linked by wiring channels are much smaller than those of a CPLD and there are far more such logic blocks than there are in a CPLD. FPGA logic blocks consist of smaller logic elements. A logic element has only one flip-flop that is individually configured and controlled. Logic complexity of a logic element is only about 10 to 20 equivalent gates. A further enhancement in the structure of FPGAs is the addition of memory blocks that can be configured as a general purpose RAM. Figure 4.30 shows the general structure of an FPGA.

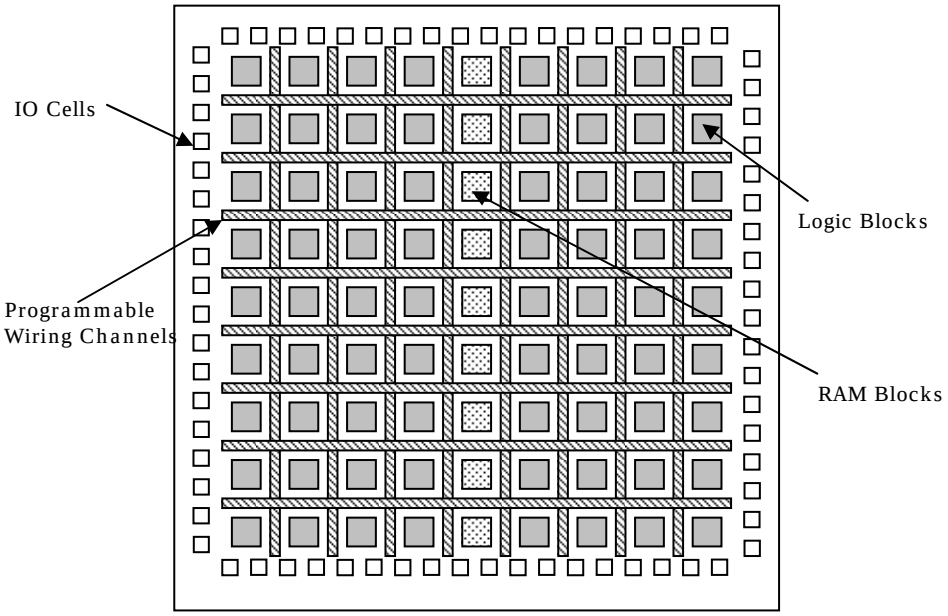

Figure 4.30 FPGA General Structure

As shown in Figure 4.30, an FPGA is an array of many logic blocks that are linked by horizontal and vertical wiring channels. FPGA RAM blocks can also be used for logic implementation or they can be configured to form memories of various word sizes and address space. Linking of logic blocks with the I/O cells and with the memories are done through wiring channels. Within logic blocks, smaller logic elements are linked by local wires.

FPGAs from different manufacturers vary in routing mechanisms, logic blocks, memories and I/O pin capabilities. As a typical FPGA, we will discuss Altera's EPF10K70 that is a member of this manufacturer's FLEX 10K Embedded Programmable Logic Device Family.

4.4.1 Altera's FLEX 10K FPGA

A member of Altera's FLEX 10K family is the EPF10K70 FPGA. This is a SRAM-based FPGA that can be programmed through its JTAG interface. This interface can also be used for FPGAs logic boundary-scan test. Typical gates of this family of FPGAs range from 10,000 to 250,000. This family has up to 40,960 RAM bits that can be used without reducing logic capacity.

Altera's FLEX 10K devices are based on reconfigurable CMOS SRAM elements, the Flexible Logic Element MatriX (FLEX) architecture is geared for implementation of common gate array functions. These devices are reconfigurable and can be configured on the board for the specific functionality required. At system power-up, they are configured with data stored in an Altera serial configuration device or provided by a system controller. Altera offers the EPC1, EPC2, EPC16, and EPC1441 configuration devices, which configure FLEX 10K devices via a serial data stream. Configuration data can also be downloaded from system RAM or from Altera's BitBlaster™ serial download cable or ByteBlasterMV™ parallel port download cable. After a FLEX 10K device has been configured, it can be reconfigured in-circuit by resetting the device and loading new data. Reconfiguration requires less than 320 ms. FLEX 10K devices contain an interface that permits microprocessors to configure FLEX 10K devices serially or in parallel, and synchronously or asynchronously. The interface also enables microprocessors to treat a FLEX 10K device as memory and configure the device by writing to a virtual memory location.

The EPF10K70 has a total of 70,000 typical gates that include logic and RAM. There are a total of 118,000 system gates. The entire array contains 468 Logic Array Blocks (LABs) that are arranged in 52 columns and 9 rows. The LABs are the "Logic Blocks" shown in Figure 4.30. Each LAB has 8 Logic Elements (LEs), making the total number of its LEs 3,744. In the middle of the FPGA chip, a column of 9 Embedded Array Blocks (EABs), each of which has 2,048 bits, form the 18,432 RAM bits of this FPGA. The EPF10K70 has 358 user I/O pins.

FLEX 10K Blocks. The block diagram of a FLEX 10K is shown in Figure 4.31. Each group of LEs is combined into an LAB; LABs are arranged into rows and columns. Each row also contains a single EAB. The LABs and EABs are interconnected by the FastTrack Interconnect. IOEs are located at the end of each row and column of the FastTrack Interconnect.

FLEX 10K devices provide six dedicated inputs that drive the flip-flops' control inputs to ensure the efficient distribution of high-speed, low-skew (less than 1.5 ns) control signals. These signals use dedicated routing channels that provide shorter delays and lower skews than the FastTrack Interconnect. Four of the dedicated inputs drive four global signals. These four global signals can also be driven by internal logic, providing an ideal solution for a clock divider or an internally generated asynchronous clear signal that clears many registers in the device.

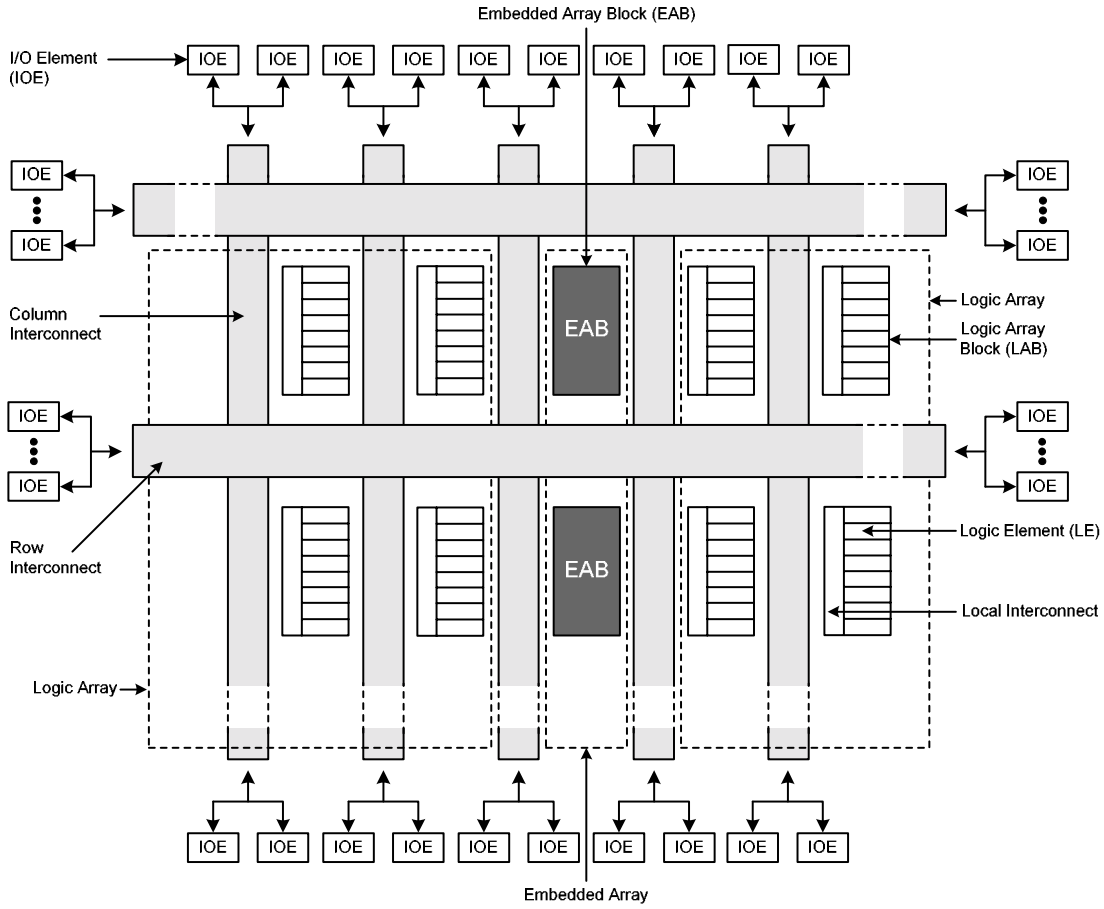

Figure 4.31 FLEX 10K Block Diagram

Signal interconnections within FLEX 10K devices and to and from device pins are provided by the FastTrack Interconnect, a series of fast, continuous row and column channels that run the entire length and width of the device.

Each I/O pin is fed by an I/O element (IOE) located at the end of each row and column of the FastTrack Interconnect. Each IOE contains a bidirectional I/O buffer and a flip-flop that can be used as either an output or input register to feed input, output, or bidirectional signals. When used with a dedicated clock pin, these registers provide exceptional performance. As inputs, they provide setup times as low as 1.6 ns and hold times of 0 ns; as outputs, these registers provide clock-to-output times as low as 5.3 ns. IOEs provide a variety of features, such as JTAG BST support, slew-rate control, tri-state buffers, and open-drain outputs.

Embedded Array Block. Each device contains an embedded array to implement memory and specialized logic functions, and a logic array to implement general logic. The embedded array consists of a series of EABs (EPF10K70 has 9 EABs). When implementing memory functions, each EAB provides 2,048 bits, which can be used to create RAM, ROM, dual-port RAM, or first-in first-out (FIFO) functions. When implementing logic, each EAB can contribute 100 to

600 gates towards complex logic functions, such as multipliers, microcontrollers, state machines, and DSP functions. EABs can be used independently, or multiple EABs can be combined to implement larger functions. Figure 4.32 shows the architecture of EABs and their interconnect busses. The EPF10K70 has 26 inputs to the LAB local interconnect channel from the row.

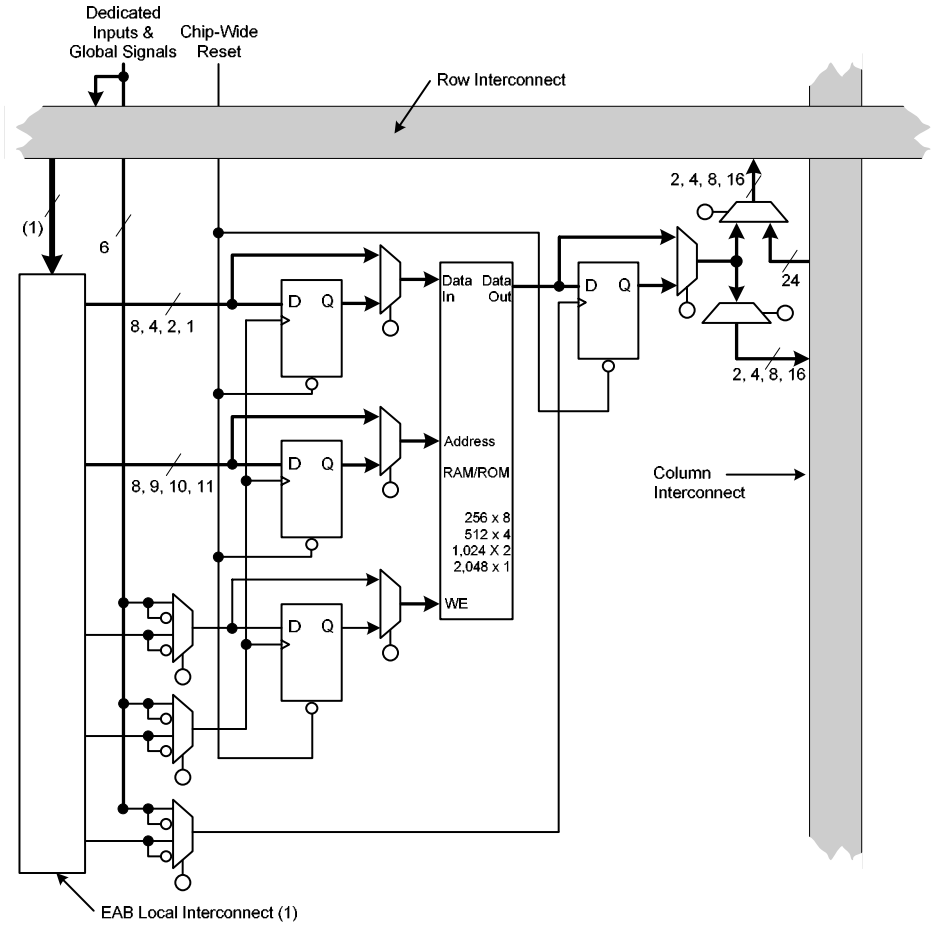

Figure 4.32 EAB Architecture and its Interconnects

Logic functions are implemented by programming the EAB with a read-only pattern during configuration, creating a large look-up table. With tables, combinatorial functions are implemented by looking up the results, rather than by computing them. This implementation of combinatorial functions can be faster than using algorithms implemented in general logic, a performance advantage that is further enhanced by the fast access times of EABs. The large capacity of EABs enables designers to implement complex functions in one logic level. For example, a single EAB can implement a 4×4 multiplier with eight inputs and eight outputs.

EABs can be used to implement synchronous RAM that generates its own WE signal and is self-timed with respect to the global clock. A circuit using the EAB's self-timed RAM need only meet the setup and hold time specifications of

the global clock. When used as RAM, each EAB can be configured in any of the following sizes: 256×8, 512×4, 1,024×2, or 2,048×1. Larger blocks of RAM are created by combining multiple EABs.

Different clocks can be used for the EAB inputs and outputs. Registers can be independently inserted on the data input, EAB output, or the address and WE inputs. The global signals and the EAB local interconnect can drive the WE signal. The global signals, dedicated clock pins, and EAB local interconnect can drive the EAB clock signals. Because the LEs drive the EAB local interconnect, the LEs can control the WE signal or the EAB clock signals.

Each EAB is fed by a row interconnect and can drive out to row and column interconnects. Each EAB output can drive up to two row channels and up to two column channels; the unused row channel can be driven by other LEs.

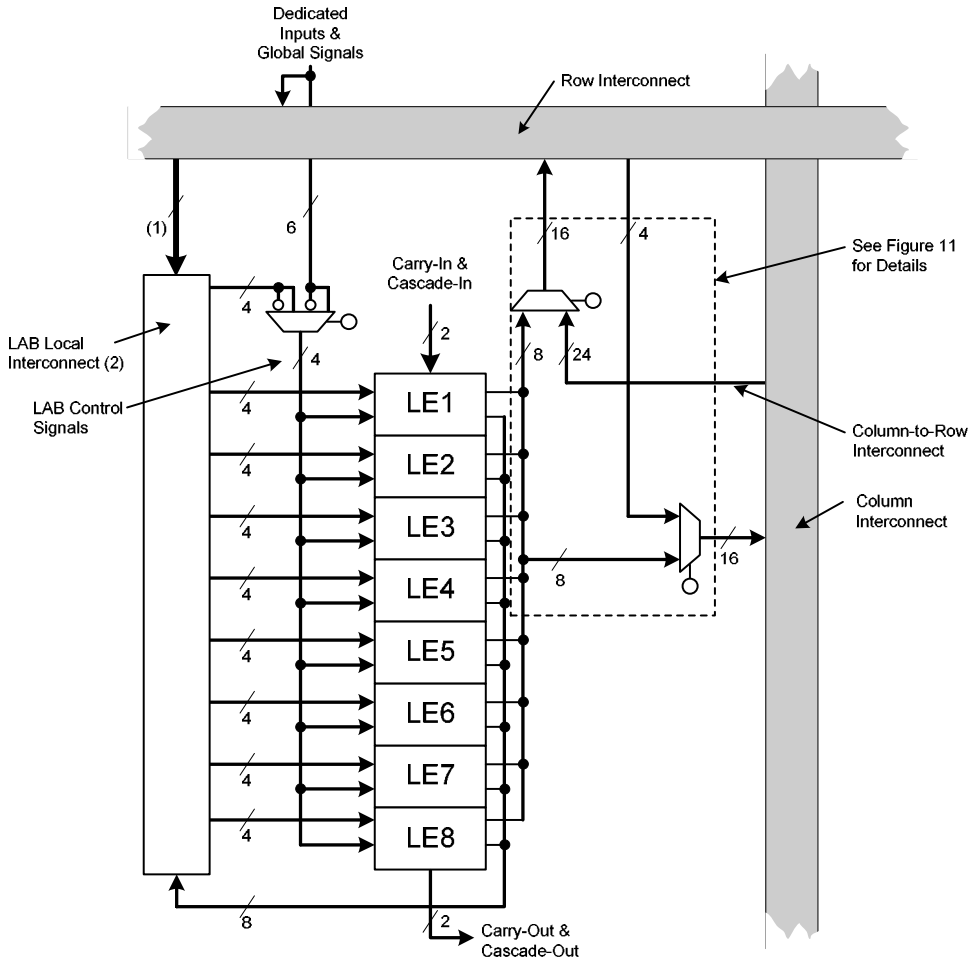

Figure 4.33 FLEX 10K LAB Architecture

Logic Array Block. Refering to Figure 4.31, the logic array of FLEX 10K consists of logic array blocks (LABs). Each LAB contains eight LEs and a local interconnect. An LE consists of a 4-input look-up table (LUT), a programmable

flip-flop, and dedicated signal paths for carry and cascade functions. Each LAB represents about 96 usable gates of logic.

Each LAB (see Figure 4.33) provides four control signals with programmable inversion that can be used in all eight LEs. Two of these signals can be used as clocks; the other two can be used for clear/preset control. The LAB clocks can be driven by the dedicated clock input pins, global signals, I/O signals, or internal signals via the LAB local interconnect. The LAB preset and clear control signals can be driven by the global signals, I/O signals, or internal signals via the LAB local interconnect. The global control signals are typically used for global clock, clear, or preset signals because they provide asynchronous control with very low skew across the device. If logic is required on a control signal, it can be generated in one or more LEs in any LAB and driven into the local interconnect of the target LAB. In addition, the global control signals can be generated from LE outputs.

Logic Element. The LE is the smallest unit of logic in the FLEX 10K architecture. Each LE contains a four-input LUT, which is a function generator that can compute any function of four variables. In addition, each LE contains a programmable flip-flop with a synchronous enable, a carry chain, and a cascade chain. Each LE drives both the local and the FastTrack Interconnect. See Figure 4.34.

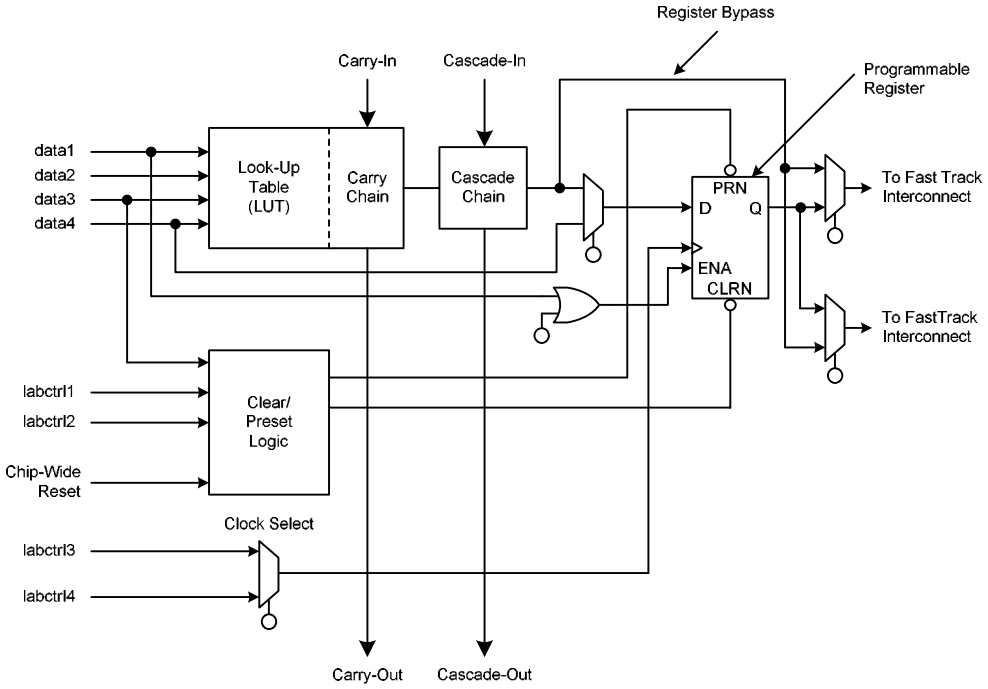

Figure 4.34 Logic Element Structure

The programmable flip-flop in the LE can be configured for D, T, JK, or SR operation. The clock, clear, and preset control signals on the flip-flop can be driven by global signals, general-purpose I/O pins, or any internal logic. For

combinatorial functions, the flip-flop is bypassed and the output of the LUT drives the output of the LE.

The LE has two outputs that drive the interconnect; one drives the local interconnect and the other drives either the row or column FastTrack Interconnect. The two outputs can be controlled independently. For example, the LUT can drive one output while the register drives the other output. This feature, called register packing, can improve LE utilization because the register and the LUT can be used for unrelated functions.

The FLEX 10K architecture provides two types of dedicated high-speed data paths that connect adjacent LEs without using local interconnect paths: carry chains and cascade chains. The carry chain supports high-speed counters and adders; the cascade chain implements wide-input functions with minimum delay. Carry and cascade chains connect all LEs in an LAB and all LABs in the same row. Intensive use of carry and cascade chains can reduce routing flexibility. Therefore, the use of these chains should be limited to speed-critical portions of a design.

The FLEX 10K LE can operate in the following four modes: Normal mode, Arithmetic mode, Up/down counter mode, and Clearable counter mode. Each of these modes uses LE resources differently. In each mode, seven available inputs to the LE—the four data inputs from the LAB local interconnect, the feedback from the programmable register, and the carry-in and cascade-in from the previous LE—are directed to different destinations to implement the desired logic function. Three inputs to the LE provide clock, clear, and preset control for the register. The architecture provides a synchronous clock enable to the register in all four modes.

The FLEX 10K architecture, shown in Figure 4.34, includes a “Clear/Preset Logic” block, which provides controls for the LE flip-flop. Logic for the programmable register’s clear and preset functions is controlled by the DATA3, LABCTRL1, and LABCTRL2 inputs to the LE. The clear and preset control structure of the LE asynchronously loads signals into a register. Either LABCTRL1 or LABCTRL2 can control the asynchronous clear. Alternatively, the register can be set up so that LABCTRL1 implements an asynchronous load. The data to be loaded is driven to DATA3; when LABCTRL1 is asserted, DATA3 is loaded into the register.

FastTrack Interconnect. In the FLEX 10K architecture, connections between LEs and device I/O pins are provided by the FastTrack Interconnect, shown in Figure 4.35. This is a series of continuous horizontal and vertical routing channels that traverse the device. This global routing structure provides predictable performance, even in complex designs.

The FastTrack Interconnect consists of row and column interconnect channels that span the entire device. Each row of LABs is served by a dedicated row interconnect. The row interconnect can drive I/O pins and feed other LABs in the device. The column interconnect routes signals between rows and can drive I/O pins.

A row channel can be driven by an LE or by one of three column channels. These four signals feed dual 4-to-1 multiplexers that connect to two specific row channels. These multiplexers, which are connected to each LE, allow column channels to drive row channels even when all eight LEs in an LAB drive the row interconnect.

Each column of LABs is served by a dedicated column interconnect. The column interconnect can then drive I/O pins or another row's interconnect to route the signals to other LABs in the device. A signal from the column interconnect, which can be either the output of an LE or an input from an I/O pin, must be routed to the row interconnect before it can enter an LAB or EAB. Each row channel that is driven by an IOE or EAB can drive one specific column channel.

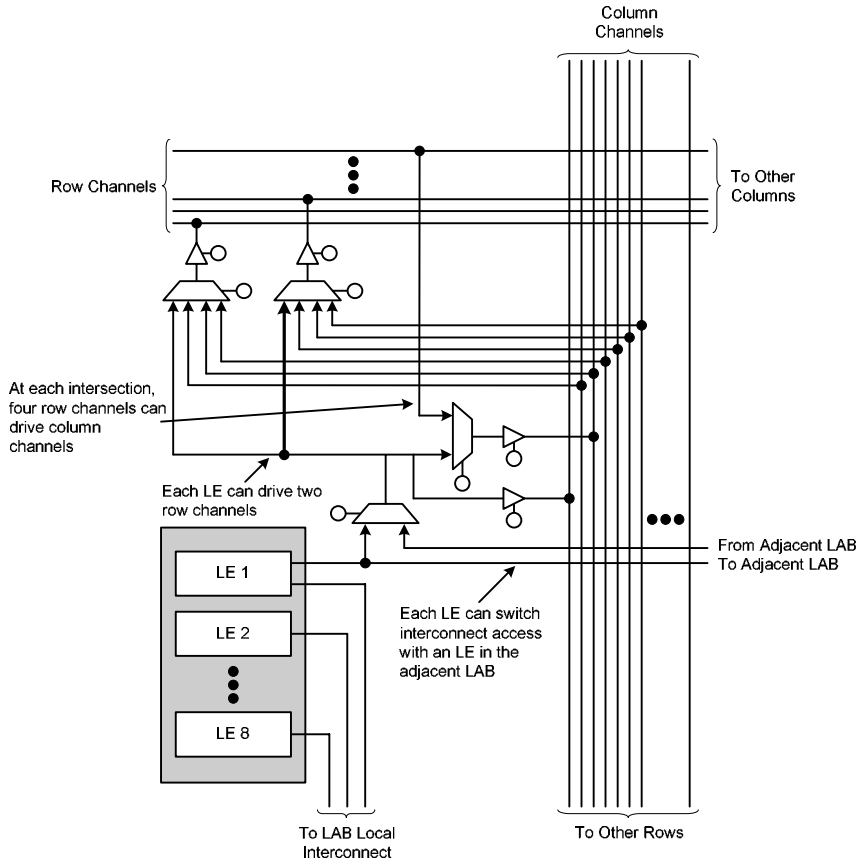

Figure 4.35 FastTrack Interconnect

Access to row and column channels can be switched between LEs in adjacent pairs of LABs. For example, an LE in one LAB can drive the row and column channels normally driven by a particular LE in the adjacent LAB in the same row, and vice versa. This routing flexibility enables routing resources to be used more efficiently. EPF10K70 has 8 rows, 312 channels per row, 52 columns, and 24 interconnects per column.

I/O Element. An I/O element (IOE) of FLEX 10K (see the top-level architecture of Figure 4.31) contains a bidirectional I/O buffer and a register that can be used either as an input register for external data that requires a fast setup time, or as an output register for data that requires fast clock-to-output performance. In some cases, using an LE register for an input register will result in a faster setup time than using an IOE register. IOEs can be used as input, output, or bidirectional pins. For bidirectional registered I/O implementation, the output

register should be in the IOE, and the data input and output enable register should be LE registers placed adjacent to the bidirectional pin.

When an IOE connected to a row (as shown in Figure 4.36), is used as an input signal it can drive two separate row channels. The signal is accessible by all LEs within that row. When such an IOE is used as an output, the signal is driven by a multiplexer that selects a signal from the row channels. Up to eight IOEs connect to each side of each row channel.

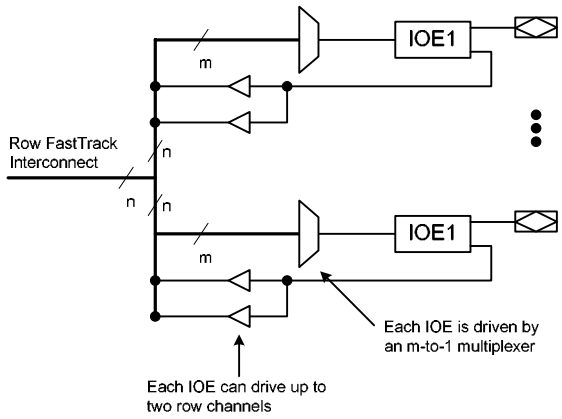

Figure 4.36 FLEX 10K Row-to-IOE Connections

When an IOE connected to a column (as shown in Figure 4.37) is used as an input, it can drive up to two separate column channels. When an IOE is used as an output, the signal is driven by a multiplexer that selects a signal from the column channels. Two IOEs connect to each side of the column channels. Each IOE can be driven by column channels via a multiplexer. The set of column channels that each IOE can access is different for each IOE.

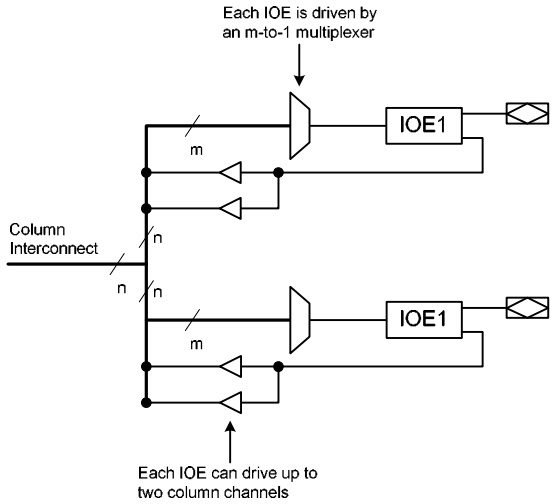

Figure 4.37 FLEX 10K Column-to-IOE Connections

In this section we have shown FPGA structures by using Altera's EPF10K70 that is a member of the FLEX 10K family as an example. The focus of the above

discussion was on the logic structure on this programmable device, and many of the timing and logical configuration details have been eliminated. The "*FLEX 10K Embedded Programmable Logic Device Family*" datasheet is a detailed document about this and other FLEX 10K members. Interested readers are encouraged to study this document for advanced features and details of logical configurations of this FPGA family.

4.5 Summary

In an evolutionary fashion, this chapter showed how a simple idea like the ROM have evolved into FPGA programmable chips that can be used for implementation of complete systems that include several processors, memories and even some analog parts. The first part of this chapter discussed generic structures of programmable devices, and in the second part, when describing more complex programmable devices, Altera devices were used as examples. We focused on the structures and tried to avoid very specific manufacturer's details. This introduction familiarizes readers with the general concepts of the programmable devices and enables them to better understand specific manufacturer's datasheets.