CHAPTER 11
Programmable Logic Devices

Programmable logic is the means by which a large segment of engineers implement their custom logic, whether that logic is a simple I/O port or a complex state machine. Most programmable logic is implemented with some type of HDL that frees the engineer from having to derive and minimize Boolean expressions each time a new logical relationship is designed. The advantages of programmable logic include rapid customization with relatively limited expense invested in tools and support.

The widespread availability of flexible programmable logic products has brought custom logic design capabilities to many individuals and smaller companies that would not otherwise have the financial and staffing resources to build a fully custom IC. These devices are available in a wide range of sizes, operating voltages, and speeds, which all but guarantees that a particular application can be closely matched with a relevant device. Selecting that device requires some research, because each manufacturer has a slightly different specialty and range of products.

Programmable logic technology advances rapidly, and manufacturers are continually offering devices with increased capabilities and speeds. After completing this chapter and learning about the basic types of devices that are available, it is recommended that you to browse through the latest manufacturers' data sheets to get updated information. Companies such as Altera, Atmel, Cypress, Lattice, QuickLogic, and Xilinx provide detailed data sheets on their web sites and also tend to offer bundled development software for reasonable prices.

11.1 CUSTOM AND PROGRAMMABLE LOGIC

Beyond using discrete 7400 ICs, custom logic is implemented in larger ICs that are either manufactured with custom masks at a factory or programmed with custom data images at varying points after fabrication. Custom ICs, or *application specific integrated circuits* (ASICs), are the most flexible option because, as with anything custom, there are fewer constraints on how application specific logic is implemented. Because custom ICs are tailored for a specific application, the potential exists for high clock speeds and relatively low unit prices. The disadvantages to custom ICs are long and expensive development cycles and the inability to make quick logic changes. Custom IC development cycles are long, because a design must generally be frozen in a final state before much of the silicon layout and circuit design work can be completed. Engineering charges for designing a custom mask set (not including the logic design work) can range from $50,000 to well over $1 million, depending on the complexity. Once manufactured, the logic can't simply be altered, because the logic configuration is an inherent property of the custom design. If a bug is found, the time and money to alter the mask set can approach that of the initial design itself.

Programmable logic devices (PLDs) are an alternative to custom ASICs. A PLD consists of general-purpose logic resources that can be connected in many permutations according to an engineer's logic design. This programmable connectivity comes at the price of additional, hidden logic that makes logic connections within the chip. The main benefit of PLD technology is that a design can be rapidly loaded into a PLD, bypassing the time consuming and expensive custom IC development process. It follows that if a bug is found, a fix can be implemented very quickly and, in many cases, reprogrammed into the existing PLD chip. Some PLDs are one-time programmable, and some can be reprogrammed in circuit.

The disadvantage of PLDs is the penalty paid for the hidden logic that implements the programmable connectivity between logic gates. This penalty manifests itself in three ways: higher unit cost, slower speeds, and increased power consumption. Programmable gates cost more than custom gates, because, when a programmable gate is purchased, that gate plus additional connectivity overhead is actually being paid for. Propagation delay is an inherent attribute of all silicon structures, and the more structures that are present in a path, the slower the path will be. It follows that a programmable gate will be slower than a custom gate, because that programmable gate comes along with additional connectivity structures with their own timing penalties. The same argument holds true for power consumption.

Despite the downside of programmable logic, the technology as a whole has progressed dramatically and is extremely popular as a result of competitive pricing, high performance levels, and, especially, quick time to market. Time to market is an attribute that is difficult to quantify but one that is almost universally appreciated as critical to success. PLDs enable a shorter development cycle, because designs can be prototyped rapidly, the bugs worked out, and product shipped to a customer before some ASIC technologies would even be in fabrication. Better yet, if a bug is found in the field, it may be fixable with significantly less cost and disruption. In the early days of programmable logic, PLDs could not be reprogrammed, meaning that a bug could still force the recall of product already shipped. Many modern reprogrammable PLDs allow hardware bugs to be fixed in the field with a software upgrade consisting of a new image that can be downloaded to the PLD without having to remove the product from the customer site.

Cost and performance are probably the most debated trade-offs involved in using programmable logic. The full range of applications in which PLDs or ASICs are considered can be broadly split into three categories as shown in Fig. 11.1. At the high end of technology, there are applications in which an ASIC is the only possible solution because of leading edge timing and logic density requirements. In the mid range, clock frequencies and logic complexity are such that a PLD is capable of solving the problem, but at a higher unit cost than an ASIC. Here, the decision must be made between flexibility and time to market versus lowest unit cost. At the low end, clock frequencies and logic density requirements are far enough below the current state of silicon technology that a PLD may meet or even beat the cost of an ASIC.

It may sound strange that a PLD with its overhead can ever be less expensive than a custom chip. The reasons for this are a combination of silicon die size and volume pricing. Two of the major factors in the cost of fabricating a working silicon die are its size and manufacturing yield. As a die gets smaller, more of them can be fabricated at the same time on the same wafer using the same resources. IC manufacturing processes are subject to a certain yield, which is the percentage of working dice obtained from an overall lot of dice. Some dice develop microscopic flaws during manufacture that make them unusable. Yield is a function of many variables, including the reliability of uniformly manufacturing a certain degree of complexity given the prevailing state of technology at a point in time. From these two points, it follows that a silicon chip will be less expensive to manufacture if it is both small and uses a technology process that is mature and has a high yield.

At the low end of speed and density, a small PLD and a small ASIC may share the same mature technology process and the same yield characteristics, removing yield as a significant variable in

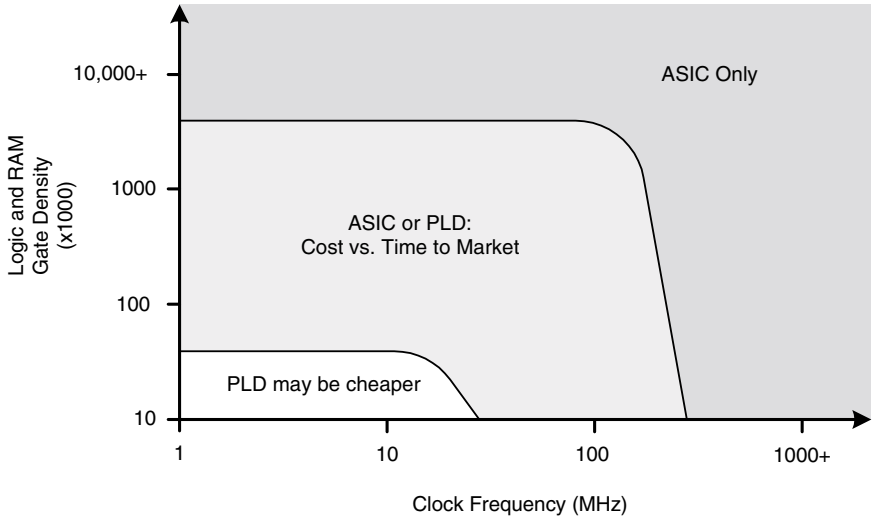

FIGURE 11.1 PLDs vs. ASICs circa 2003.

their cost differential. Likewise, raw packaging costs are likely to be comparable because of the maturation of stable packaging materials and technologies. The cost differential comes down to which solution requires the smaller die and how the overhead costs of manufacturing and distribution are amortized across the volume of chips shipped.

Die size is function of two design characteristics: how much logic is required and how many I/O pins are required. While the size of logic gates has decreased by orders of magnitude over time, the size of I/O pads, the physical structures that packaging wires connect to, has not changed by the same degree. There are nonscalable issues constraining pad size, including physical wire bonding and current drive requirements. I/O pads are often placed on the perimeter of a die. If the required number of I/O pads cannot be placed along the existing die's perimeter, the die must be enlarged even if not required by the logic. ICs can be considered as being balanced, logic limited, or pad limited. A balanced design is optimal, because silicon area is being used efficiently by the logic and pad structures. A logic-limited IC's silicon area is dominated by the internal logic requirements. At the low end being presently discussed, being logic limited is not a concern because of the current state of technology. Pad-limited designs are more of a concern at the low end, because the chip is forced to a certain minimum size to support a minimum number of pins.

Many low-end logic applications end up being pad limited as the state of silicon process technology advances and more logic gates can be squeezed into ever smaller areas. The logic shrinks, but the I/O pads do not. Once an IC is pad limited, ASIC and CPLD implementations may use the same die size, removing it as a cost variable. This brings us back to the volume pricing and distribution aspects of the semiconductor business. If two silicon manufacturers are fabricating what is essentially the same chip (same size, yield, and package), who will be able to offer the lowest final cost? The comparison is between a PLD vendor that turns out millions of the exact same chip each year versus an ASIC vendor that can manufacture only as many of your custom chips that you are willing to buy. Is an ASIC's volume 10,000 units per year? 100,000? One million? With all other factors being equal, the high-volume PLD vendor has the advantage, because the part being sold is not custom but a mass-produced generic product.

11.2 GALS AND PALS

Among the most basic types of PLDs are Generic Array Logic™ (GAL) devices.* GALs are enhanced variants of the older Programmable Array Logic™ (PAL) architecture that is now essentially obsolete. The term PAL is still widely used, but people are usually referring to GAL devices or other PLD variants when they use the term. PALs became obsolete, because GALs provide a superset of their functionality and can therefore perform all of the functions that PALs did. GALs are relatively small, inexpensive, easily available, and manufactured by a variety of vendors (e.g., Cypress, Lattice, and Texas Instruments).

It can be shown through Boolean algebra that any logical expression can be represented as an arbitrarily complex sum of products. Therefore, by providing a programmable array of AND/OR gates, logic can be customized to fit a particular application. GAL devices provide an extensive programmable array of wide AND gates, as shown in Fig. 11.2, into which all the device's input terms are fed. Both true and inverted versions of each input are made available to each AND gate. The outputs of groups of AND gates (products) feed into separate OR gates (sums) to generate user-defined Boolean expressions.

Each intersection of a horizontal AND gate line and a vertical input term is a programmable connection. In the early days of PLDs, these connections were made by fuses that would literally have to be blown with a high voltage to configure the device. Fuse-based devices were not reprogrammable;

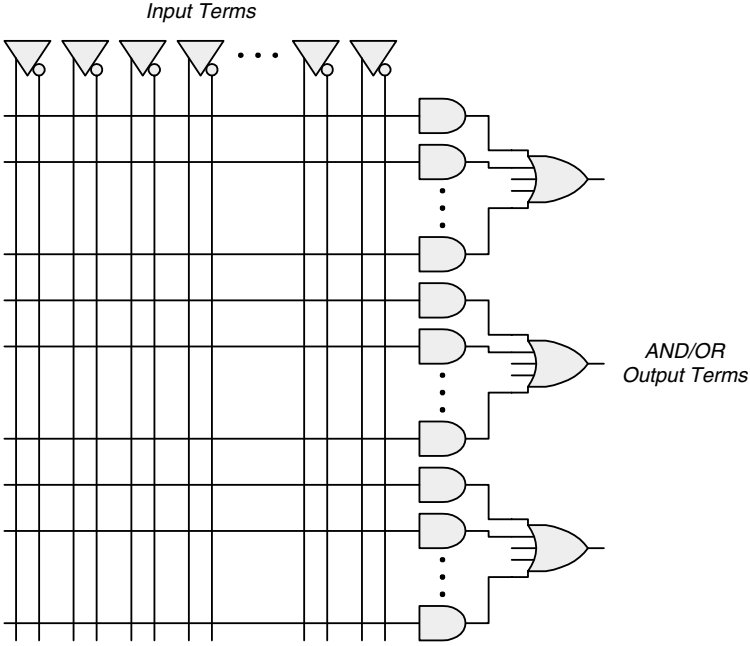

FIGURE 11.2 GAL/PAL AND/OR structure.

* GAL, Generic Array Logic, PAL, and Programmable Array Logic are trademarks of Lattice Semiconductor Corporation.

once a microscopic fuse is blown, it cannot be restored. Today's devices typically rely on EEPROM technology and CMOS switches to enable nonvolatile reprogrammability. However, fuse-based terminology remains in use for historical reasons. The default configuration of a connection emulates an intact fuse, thereby connecting the input term to the AND gate input. When the connection is blown, or programmed, the AND input is disconnected from the input term and pulled high to effectively remove that input from the Boolean expression. Customization of a GAL's programmable AND gate is conceptually illustrated in Fig. 11.3.

With full programmability of the AND array, the OR connections can be hard wired. Each GAL device feeds a differing number of AND terms into the OR gates. If one or more of these AND terms are not needed by a particular Boolean expression, those unneeded AND gates can be effectively disabled by forcing their outputs to 0. This is done by leaving an unneeded AND gate's inputs unprogrammed. Remember that inputs to the AND array are provided in both true and complement versions. When all AND connections are left intact, multiple expressions of the form $A\&\overline{A} = 0$ result, thereby forcing that gate's output to 0 and rendering it nonparticipatory in the OR function.

The majority of a GAL's logic customization is performed by programming the AND array. However, selecting flip-flops, OR/NOR polarities, and input/output configurations is performed by programming a configurable I/O and feedback structure called a *macrocell*. The basic concept behind a macrocell is to ultimately determine how the AND/OR Boolean expression is handled and how the macrocell's associated I/O pin operates. A schematic view of a GAL macrocell is shown in Fig. 11.4, although some GALs may contain macrocells with slightly different capabilities. Multiplexers determine the polarity of the final OR/NOR term, regardless of whether the term is registered and whether the feedback signal is taken directly at the flop's output or at the pin. Configuring the macrocell's output enable determines how the pin behaves.

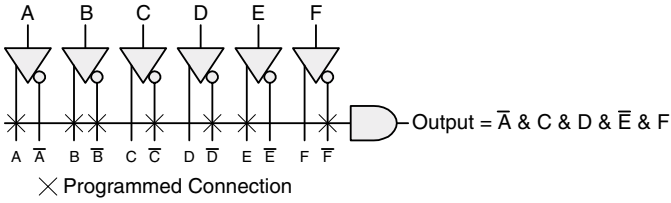

FIGURE 11.3 Programming AND input terms.

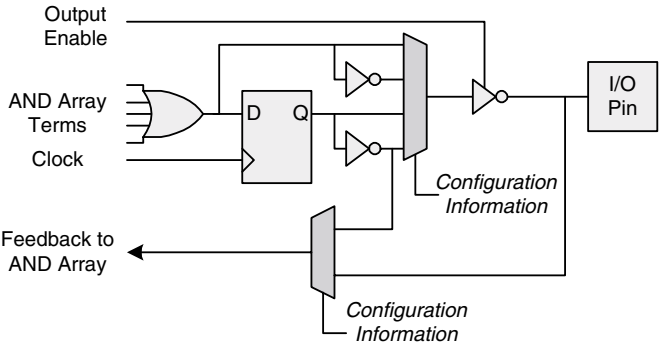

FIGURE 11.4 GAL macrocell.

There are two common GAL devices, the 16V8 and the 22V10, although other variants exist as well. They contain eight and ten macrocells each, respectively. The 16V8 provides up to 10 dedicated inputs that feed the AND array, whereas the 22V10 provides 12 dedicated inputs. One of the 22V10's dedicated inputs also serves as a global clock for any flops that are enabled in the macrocells. Output enable logic in a 22V10 is evaluated independently for each macrocell via a dedicated AND term. The 16V8 is somewhat less flexible, because it cannot arbitrarily feed back all macrocell outputs depending on the device configuration. Additionally, when configured for registered mode where macrocell flops are usable, two dedicated input pins are lost to clock and output enable functions.

GALs are fairly low-density PLDs by modern standards, but their advantages of low cost and high speed are derived from their small size. Implementing logic in a GAL follows several basic steps. First, the logic is represented in either graphical schematic diagram or textual (HDL) form. This representation is converted into a netlist using a translation or synthesis tool. Finally, the netlist is fitted into the target device by mapping individual gate functions into the programmable AND array. Given the fixed AND/OR structure of a GAL, fitting software is designed to perform logic optimizations and translations to convert arbitrary Boolean expressions into sum-of-product expressions. The result of the fitting process is a programming image, also called a *fuse map*, that defines exactly which connections, or fuses, are to be programmed and which are to be left at their default state. The programming image also contains other information such as macrocell configuration and other device-specific attributes.

Modern PLD development software allows the back-end GAL synthesis and fitting process to proceed without manual intervention in most cases. The straightforward logic flow through the programmable AND array reduces the permutations of how a given Boolean expression can be implemented and results in very predictable logic fitting. An input signal propagates through the pin and pad structure directly into the AND array, passes through just two gates, and can then either feed a macrocell flop or drive directly out through an I/O pin. Logic elements within a GAL are close to each other as a result of the GAL's small size, which contributes to low internal propagation delays. These characteristics enable the GAL architecture to deliver very fast timing specifications, because signals follow deterministic paths with low propagation delays.

GALs are a logic implementation technology with very predictable capabilities. If the desired logic cannot fit within the GAL, there may not be much that can be done without optimizing the algorithm or partitioning the design across multiple devices. If the logic fits but does not meet timing, the logic must be optimized, or a faster device must be found. Because of the GAL's basic fitting process and architecture, there isn't the same opportunity of tweaking the device as can be done with more complex PLDs. This should not be construed as a lack of flexibility on the part of the GAL. Rather, the GAL does what it says it does, and it is up to the engineer to properly apply the technology to solve the problem at hand. It is the simplicity of the GAL architecture that is its greatest strength.

Lattice Semiconductor's GAL22LV10D-4 device features a worst-case input-to-output combinatorial propagation delay of just 4 ns.* This timing makes the part suitable for address decoding on fast microprocessor interfaces. The same 22V10 part features a 3-ns t_{CO} and up to 250-MHz operation. The t_{CO} specification is a pin-to-pin metric that includes the propagation delays of the clock through the input pin and the output signal through the output pin. Internally, the actual flop itself exhibits a faster t_{CO} that becomes relevant for internal logic feedback paths. Maximum clock frequency specifications are an interesting aspect of all PLDs and some consideration. These specifications are best-case numbers usually obtained with minimal logic configurations. They may define

* GAL22LV10D, 22LV10_04, Lattice Semiconductor, 2000, p. 7.

the highest toggle rate of the device's flops, but synchronous timing analysis dictates that there is more to f_{MAX} than the flop's t_{SU} and t_{CO}. Propagation delay of logic and connectivity between flops is of prime concern. The GAL architecture's deterministic and fast logic feedback paths reduces the added penalty of internal propagation delays. Lattice's GAL22LV10D features an internal clock-to-feedback delay of 2.5 ns, which is the combination of the actual flop's t_{CO} plus the propagation delay of the signal back through the AND/OR array. This feedback delay, when combined with the flop's 3-ns t_{SU}, yields a practical f_{MAX} of 182 MHz when dealing with most normal synchronous logic that contains feedback paths (e.g., a state machine).

11.3 CPLDS

Complex PLDs, or CPLDs, are the mainstream macrocell-based PLDs in the industry today, providing logic densities and capabilities well beyond those of a GAL device. GALs are flexible for their size because of the large programmable AND matrix that defines logical connections between inputs and outputs. However, this anything-to-anything matrix makes the architecture costly to scale to higher logic densities. For each macrocell that is added, both matrix dimensions grow as well. Therefore, the AND matrix increases in a square function of the I/O terms and macrocells in the device. CPLD vendors seek to provide a more linear scaling of connectivity resources to macrocells by implementing a segmented architecture with multiple fixed-size GAL-style logic blocks that are interconnected via a central switch matrix as shown in Fig. 11.5. Like a GAL, CPLDs are typically manufactured with EEPROM configuration storage, making their function nonvolatile. After programming, a CPLD will retain its configuration and be ready for operation when power is applied to the system.

Each individual logic block is similar to a GAL and contains its own programmable AND/OR array and macrocells. This approach is scalable, because the programmable AND/OR arrays remain fixed in size and small enough to fabricate economically. As more macrocells are integrated onto the same chip, more logic blocks are placed onto the chip instead of increasing the size of individual logic blocks and bloating the AND/OR arrays. CPLDs of this type are manufactured by companies including Altera, Cypress, Lattice, and Xilinx.

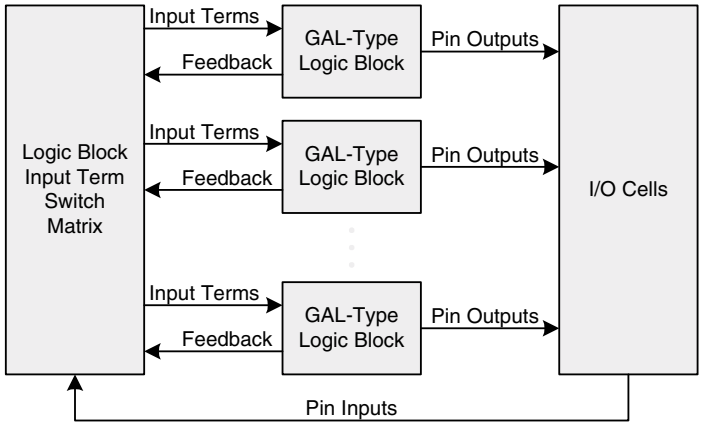

FIGURE 11.5 Typical CPLD architecture.

Generic user I/O pins are bidirectional and can be configured as inputs, outputs, or both. This is in contrast to the dedicated power and test pins that are necessary for operation. There are as many potential user I/O pins as there are macrocells, although some CPLDs may be housed in packages that do not have enough physical pins to connect to all the chip's I/O sites. Such chips are intended for applications that are logic limited rather than pad limited.

Because the size of each logic block's AND array is fixed, the block has a fixed number of possible inputs. Vendor-supplied fitting software must determine which logical functions are placed into which blocks and how the switch matrix connects feedback paths and input pins to the appropriate block. The switch matrix does not grow linearly as more logic blocks are added. However, the impact of the switch matrix's growth is less than what would result with an ever expanding AND matrix. Each CPLD family provides a different number of switched input terms to each logic block.

The logic blocks share many characteristics with a GAL, as shown in Fig. 11.6, although additional flexibility is added in the form of product term sharing. Limiting the number of product terms in each logic block reduces device complexity and cost. Some vendors provide just five product terms per macrocell. To balance this limitation, which could impact a CPLD's usefulness, product term sharing resources enable one macrocell to borrow terms from neighboring macrocells. This borrowing usually comes at a small propagation delay penalty but provides necessary flexibility in handling complex Boolean expressions with many product terms. A logic block's macrocell contains a flip-flop and various configuration options such as polarity and clock control. As a result of their higher logic density, CPLDs contain multiple global clocks that individual macrocells can choose from, as well as the ability to generate clocks from the logic array itself.

Xilinx is a vendor of CPLD products and manufactures a family known as the XC9500. Logic blocks, or function blocks in Xilinx's terminology, each contain 18 macrocells, the outputs of which feed back into the switch matrix and drive I/O pins as well. XC9500 CPLDs contain multiples of 18 macrocells in densities from 36 to 288 macrocells. Each function block gets 54 input terms from the switch matrix. These input terms can be any combination of I/O pin inputs and feedback terms from other function blocks' macrocells.

Like a GAL, CPLD timing is very predictable because of the deterministic nature of the logic blocks' AND arrays and the input term switch matrix. Xilinx's XC9536XV-3 features a maximum pin-to-pin propagation delay of 3.5 ns and a t_{CO} of 2.5 ns.* Internal logic can run as fast as 277 MHz with feedback delays included, although complex Boolean expressions likely reduce this f_{MAX} because of product term sharing and feedback delays through multiple macrocells.

CPLD fitting software is typically provided by the silicon vendor, because the underlying silicon architectures are proprietary and not disclosed in sufficient detail for a third party to design the necessary algorithms. These tools accept a netlist from a schematic tool or HDL synthesizer and automatically divide the logic across macrocells and logic blocks. The fitting process is more complex

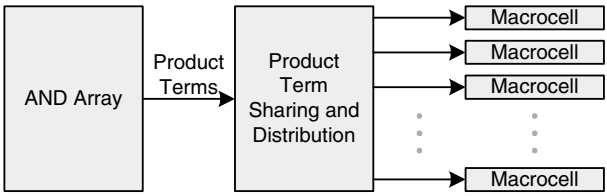

FIGURE 11.6 CPLD logic block.

* XC9536XV, DS053 (v2.2), Xilinx, August 2001, p. 4.

than for a GAL; not every term within the CPLD can be fed to each macrocell because of the segmented AND array structure. Product term sharing places restrictions on neighboring macrocells when Boolean expressions exceed the number of product terms directly connected to each macrocell. The fitting software first reduces the netlist to a set of Boolean expressions in the form that can be mapped into the CPLD and then juggles the assignment of macrocells to provide each with its required product terms. Desired operating frequency influences the placement of logic because of the delay penalties of sharing product terms across macrocells. These trade-offs occur at such a low level that human intervention is often impractical.

CPLDs have come to offer flexibility advantages beyond just logic implementation. As digital systems get more complex, logic IC supply voltages begin to proliferate. At one time, most systems ran on a single 5-V supply. This was followed by 3.3-V systems, and it is now common to find systems that operate at multiple voltages such as 3.3 V, 2.5 V, 1.8 V, and 1.5 V. CPLDs invariably find themselves designed into mixed-voltage environments for the purposes of interface conversion and bus management. To meet these needs, many CPLDs support more than one I/O voltage standard on the same chip at the same time. I/O pins are typically divided into banks, and each bank can be independently selected for a different I/O voltage.

Most CPLDs are relatively small in logic capacity because of the desire for very high-speed operation with deterministic timing and fitting characteristics at a reasonable cost. However, some CPLDs have been developed far beyond the size of typical CPLDs. Cypress Semiconductor's Delta39K200 contains 3,072 macrocells with several hundred kilobits of user-configurable RAM.* The architecture is built around clusters of 128 macrocell logic groups, each of which is similar in nature to a conventional CPLD. In a similar way that CPLDs add an additional hierarchical connectivity layer on top of multiple GAL-type logic blocks, Cypress has added a layer on top of multiple CPLD-type blocks. Such large CPLDs may have substantial benefits for certain applications. Beyond several hundred macrocells, however, engineers have tended to use larger and more scalable FPGA technologies.

11.4 FPGAS

CPLDs are well suited to applications involving control logic, basic state machines, and small groups of read/write registers. These control path applications typically require a small number of flops. Once a task requires many hundreds or thousands of flops, CPLDs rapidly become impractical to use. Complex applications that manipulate and parse streams of data often require large quantities of flops to serve as pipeline registers, temporary data storage registers, wide counters, and large state machine vectors. Integrated memory blocks are critical to applications that require multiple FIFOs and data storage buffers. *Field programmable gate arrays* (FPGAs) directly address these data path applications.

FPGAs are available in many permutations with varying feature sets. However, their common defining attribute is a fine-grained architecture consisting of an array of small logic cells, each consisting of a flop, a small lookup table (LUT), and some supporting logic to accelerate common functions such as multiplexing and arithmetic carry terms for adders and counters. Boolean expressions are evaluated by the LUTs, which are usually implemented as small SRAM arrays. Any function of four variables, for example, can be implemented in a 16×1 SRAM when the four variables serve as the index into the memory. There are no AND/OR arrays as in a CPLD or GAL. All Boolean functions

* Delta39K ISR CPLD Family, Document #38-03039 Rev. *.C, Cypress Semiconductor, December 2001, p. 1.

are implemented within the logic cells. The cells are arranged on a grid of routing resources that can make connections between arbitrary cells to build logic paths as shown in Fig. 11.7. Depending on the FPGA type, special-purpose structures are placed into the array. Most often, these are configurable RAM blocks and clock distribution elements. Around the periphery of the chip are the I/O cells, which commonly contain one or more flops to enable high-performance synchronous interfaces. Locating flops within I/O cells improves timing characteristics by minimizing the distance, and hence the delay between each flop and its associated pin. Unlike CPLDs, most FPGAs are based on SRAM technology, making their configurations volatile. A typical FPGA must be reprogrammed each time power is applied to a system. Major vendors of FPGAs include Actel, Altera, Atmel, Lattice, QuickLogic, and Xilinx.

Very high logic densities are achieved by scaling the size of the two-dimensional logic cell array. The primary limiting factor in FPGA performance becomes routing because of the nondeterministic nature of a multipath grid interconnect system. Paths between logic cells can take multiple routes, some of which may provide identical propagation delays. However, routing resources are finite, and conflicts quickly arise between competing paths for the same routing channels. As with CPLDs, FPGA vendors provide proprietary software tools to convert a netlist into a final programming image. Depending on the complexity of the design (e.g., speed and density), routing an FPGA can take a few minutes or many hours. Unlike a CPLD with deterministic interconnection resources, FPGA timing can vary dramatically, depending on the quality of the logic placement. Large, fast designs require iterative routing and placement algorithms.

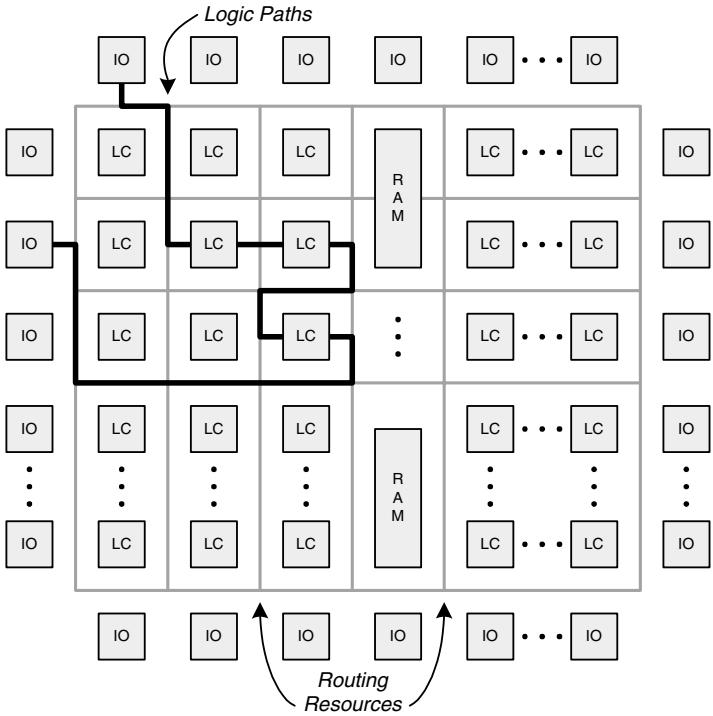

FIGURE 11.7 FPGA logic cell array.

Human intervention can be critical to the successful routing and timing of a complex FPGA design. *Floorplanning* is the process by which an engineer manually partitions logic into groups and then explicitly places these groups into sections of the logic cell array. Manually locating large portions of the logic restricts the routing software to optimizing placement of logic within those boundaries and reduces the number of permutations that it must try to achieve a successful result.

Each vendor's logic cell architecture differs somewhat, but mainly in how support functions such as multiplexing and arithmetic carry terms are implemented. For the most part, engineers do not have to consider the minute details of each logic cell structure, because the conversion of logic into the logic cell is performed by a combination of the HDL synthesis tool and the vendor's proprietary mapping software. In extreme situations, wherein a very specific logic implementation is necessary to squeeze the absolute maximum performance from a specific FPGA, optimizing logic and architecture for a given logic cell structure may have benefits. Engaging in this level of technology-specific optimization, however, can be very tricky and lead to a house-of-cards scenario in which everything is perfectly balanced for a while, and then one new feature is added that upsets the whole plan. If a design appears to be so aggressive as to require fine-tuned optimization, and faster devices cannot be obtained, it may be preferable to modify the architecture to enable more mainstream, abstracted design methodologies.

Notwithstanding the preceding comments, there are high-level feature differences among FPGAs that should be evaluated before choosing a specific device. Of course, it is necessary to pick an FPGA that has sufficient logic and I/O pins to satisfy the needs of the intended application. But not all FPGAs are created equal, despite having similar quantities of logic. While the benefits of one logic structure over another can be debated, the presence or absence of critical design resources can make implementation of a specific design possible or impossible. These resources are clock distribution elements, embedded memory, embedded third-party cores, and multifunction I/O cells.

Clock distribution across a synchronous system must be done with minimal skew to achieve acceptable timing. Each logic cell within a FPGA holds a flop that requires a clock. Therefore, an FPGA must provide at least one global clock signal distributed to each logic cell with low skew across the entire device. One clock is insufficient for most large digital systems because of the proliferation of different interfaces, microprocessors, and peripherals. Typical FPGAs provide anywhere from 4 to 16 global clocks with associated low-skew distribution resources. Most FPGAs do allow clocks to be routed using the general routing resources that normally carry logic signals. However, these paths are usually unable to achieve the low skew characteristics of the dedicated clock distribution network and, consequently, do not enable high clock speeds.

Some FPGAs support a large number of clocks, but with the restriction that not all clocks can be used simultaneously in the same portion of the chip. This type of restriction reduces the complexity of clock distribution on the part of the FPGA vendor because, while the entire chip supports a large number of clocks in total, individual sections of the chip support a smaller number. For example, an FPGA might support 16 global clocks with the restriction that any one quadrant can support only 8 clocks. This means that there are 16 clocks available, and each quadrant can select half of them for arbitrary use. Instead of providing 16 clocks to each logic cell, only 8 need be connected, thus simplifying the FPGA circuitry.

Most FPGAs provide *phase locked loops* (PLLs) or *delay locked loops* (DLLs) that enable the intentional skewing, division, and multiplication of incoming clock signals. PLLs are partially analog circuits, whereas DLLs are fully digital circuits. They have significant overlap in the functions that they can provide in an FPGA, although engineers may debate the merits of one versus the other. The fundamental advantage of a PLL or DLL within an FPGA is its ability to improve I/O timing (e.g., t_{CO}) by effectively removing the propagation delay between the clock input pin and the signal output pin, also known as *deskewing*. As shown in Fig. 11.8, the PLL or DLL aligns the incoming clock to a feedback clock with the same delay as observed at the I/O flops. In doing so, it shifts the incoming

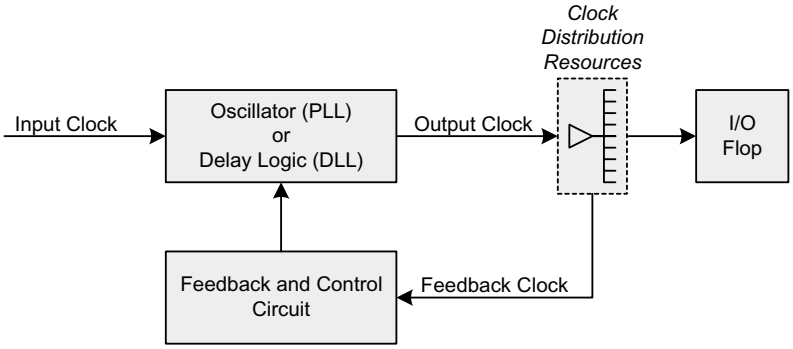

FIGURE 11.8 PLL/DLL clock deskew function within FPGA.

clock so that the causal edge observed by the I/O flops occurs at nearly the same time as when it enters the FPGA's clock input pin. PLLs and DLLs are discussed in more detail in a later chapter.

Additional circuitry enables some PLLs and DLLs to emit a clock that is related to the input frequency by a programmable ratio. The ability to multiply and divide clocks is a benefit to some system designs. An external board-level interface may run at a slower frequency to make circuit implementation easier, but it may be desired to run the internal FPGA logic as a faster multiple of that clock for processing performance reasons. Depending on the exact implementation, multiplication or division can assist with this scheme.

RAM blocks embedded within the logic cell array are a critical feature for many applications. FIFOs and small buffers figure prominently in a variety of data processing architectures. Without on-chip RAM, valuable I/O resources and speed penalties would be given up to use off-chip memory devices. To suit a wide range of applications, RAMs need to be highly configurable and flexible. A typical FPGA's RAM block is based on a certain bit density and can be used in arbitrary width/depth configurations as shown in Fig. 11.9 using the example of a 4-kb RAM block. Single- and dual-port modes are also very important. Many applications, including FIFOs, benefit from a dual-ported

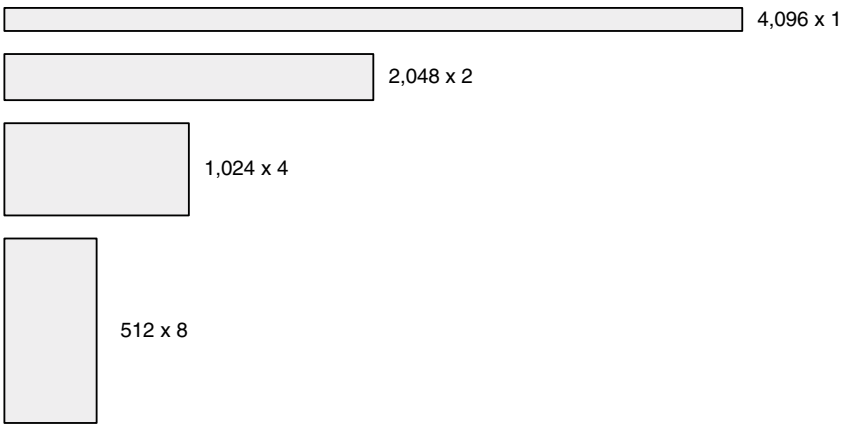

FIGURE 11.9 Configurable FPGA 4 kb RAM block.

RAM block to enable simultaneous reading and writing of the memory by different logic blocks. One state machine may be writing data into a RAM, and another may be reading other data out at the same time. RAM blocks can have synchronous or asynchronous interfaces and may support one or two clocks in synchronous modes. Supporting two clocks in synchronous modes facilitates dual-clock FIFO designs for moving data between different clock domains.

Some FPGAs also allow logic cell LUTs to be used as general RAM in certain configurations. A 16×1 four-input LUT can serve as a 16×1 RAM if supported by the FPGA architecture. It is more efficient to use RAM blocks for large memory structures, because the hardware is optimized to provide a substantial quantity of memory in a small area of silicon. However, LUT-based RAM is beneficial when a design requires many shallow memory structures (e.g., a small FIFO) and all the large RAM blocks are already used. Along with control logic, 32 four-input LUTs can be used to construct a 16×32 FIFO. If a design is memory intensive, it could be wasteful to commit one or more large RAM blocks for such a small FIFO.

Embedding third-party logic cores is a feature that can be useful for some designs, and not useful at all for others. A disadvantage of FPGAs is their higher cost per gate than custom ASIC technology. The main reason that engineers are willing to pay this cost premium is for the ability to implement custom logic in a low-risk development process. Some applications involve a mix of custom and predesigned logic that can be purchased from a third party. Examples of this include buying a microprocessor design or a standard bus controller (e.g., PCI) and integrating it with custom logic on the same chip. Ordinarily, the cost per gate of the third-party logic would be the same as that of your custom logic. On top of that cost is the licensing fee charged by the third party. Some FPGA vendors have decided that there is sufficient demand for a few standard logic cores to offer specific FPGAs that embed these cores into the silicon in a fixed configuration. The benefit of doing so is to drop the per-gate cost of the core to nearly that of a custom ASIC, because the core is hard wired and requires none of the FPGA's configuration overhead.

FPGAs with embedded logic cores may cost more to offset the licensing costs of the cores, but the idea is that the overall cost to the customer will be reduced through the efficiency of the hard-wired core implementation. Microprocessors, PCI bus controllers, and high-speed serdes components are common examples of FPGA embedded cores. Some specific applications may be well suited to this concept.

I/O cell architecture can have a significant impact on the types of board-level interfaces that the FPGA can support. The issues revolve around two variables: synchronous functionality and voltage/current levels. FPGAs support generic I/O cells that can be configured for input-only, output-only, or bidirectional operation with associated tri-state buffer output enable capability. To achieve the best I/O timing, flops for all three functions—input, output, and output-enable—should be included within the I/O cell as shown in Fig. 11.10. The timing improvement obtained by locating these three flops in the I/O cells is substantial. The alternative would be to use logic cell flops and route paths from the logic cell array directly to the I/O pin circuitry, increasing the I/O delay times. Each of the three I/O functions is provided in both registered and unregistered options using multiplexers to provide complete flexibility in logic implementation.

More advanced bus interfaces run at double data rate speeds, requiring more advanced I/O cell structures to achieve the necessary timing specifications. Newer FPGAs are available with I/O cells that specifically support DDR interfaces by incorporating two sets of flops, one for each clock edge as shown in Fig. 11.11. When configured for DDR mode, each of the three I/O functions is driven by a pair of flops, and a multiplexer selects the appropriate flop output depending on the phase of the clock. A DDR interface runs externally to the FPGA on both edges of the clock with a certain width. Internally, the interface runs at double the external width on only one edge of the same clock frequency. Therefore, the I/O cell serves as a 2:1 multiplexer for outputs and a 1:2 demultiplexer for inputs when operating in DDR mode.

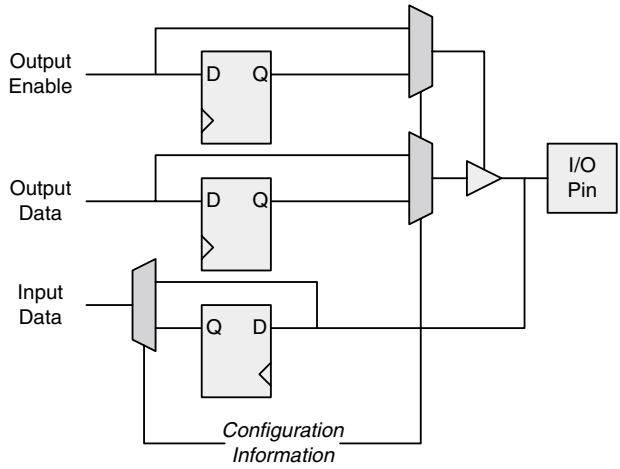

FIGURE 11.10 FPGA I/O cell structure.

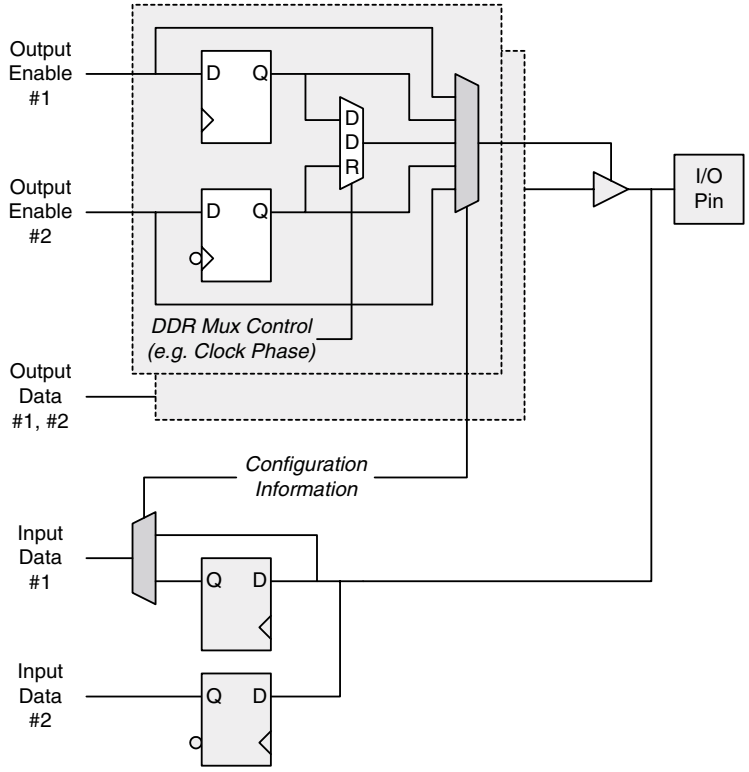

FIGURE 11.11 FPGA DDR I/O cell structure.

Aside from synchronous functionality, compliance with various I/O voltage and current drive standards is a key feature for modern, flexible FPGAs. Like CPLDs that support multiple I/O banks, each of which that can drive a different voltage level, FPGAs are usually partitioned into I/O banks as well, for the same purpose. In contrast with CPLDs, many FPGAs support a wider variety of I/O standards for greater design flexibility.