J. Webster (ed.), *Wiley Encyclopedia of Electrical and Electronics Engineering*

PROGRAMMABLE LOGIC DEVICES

Programmable logic devices (*PLD*) are electrical components whose functions are defined by the user. PLDs are typically used for rapid application prototyping, reconfigurable systems, and implementing designs requiring fast time to market. They provide a cost effective solution for generating low volume application-specific hardware, and a wide range of programmable devices are available with varying characteristics. Devices may be one-time programmable (*OTP*), electrically erasable, erasable with UV light, or in-system reconfigurable. Programmable devices are also available in a wide range of logic densities from programmable-array logic devices (*PALs*) used to implement small and medium scale integration to field programmable gate arrays (*FPGAs*) implementing tens of thousands of gates for very large scale integration (*VLSI*) applications. The level of functionality in the logic blocks of various programmable devices varies from the fine-grain programmable-array logic device to the coarse-grain logic-array blocks in complex programmable-logic devices (*CPLD*).

Reconfigurable devices are often used for system emulation and prototyping. A network of soft-programmable logic provides a reconfigurable test bed to emulate and verify designs at the system level before fabrication. Unlike costly prototypes that require a long time to fabricate, an emulator composed of programmable-logic devices provides a functional model in the few hours required to retarget the netlist and route the programmable-logic devices. Field-programmable circuit boards (*FPCBs*) provide flexible routing among various system components. The system components may include several high-density programmable-logic devices for large system emulation or may possibly contain state-of-the-art devices which may or may not have simulation models. System verification with components that do not have simulation models becomes difficult in software, making hardware emulation necessary (1).

This article begins with a description of various programmable-logic devices and discusses the advantages and disadvantages between the device architectures. Later it discusses applications for field-programmable-logic devices, such as system prototyping, application-specific integrated circuits, in-circuit reprogrammable systems, and dynamically reconfigurable designs. The article then concludes with a discussion of future PLDs.

Programmable-Logic Device Architectures

Programmable-logic devices are available in numerous sizes and structures. Programmable-logic arrays provide a means for implementing high-performance, small-scale integrated circuits. Field-programmable gate arrays connect configurable logic elements through segmented programmable interconnects and may be reconfigurable or one-time programmable. Complex programmable-logic devices connect configurable logic elements through continuous programmable interconnects providing uniform delays for every net in the design. Field-programmable interconnect components are a dense network of segmented programmable interconnects used to provide flexible routing resources among various system components.

Programmable-Logic Arrays. Programmable-logic arrays (*PLAs*) consist of a configurable array of AND gates which feed a programmable array of OR gates (2). Electrical connections in a PLA are implemented through a programmable fuse, a UV-erasable PROM, or an electrically erasable PROM (*EEPROM*) structure.

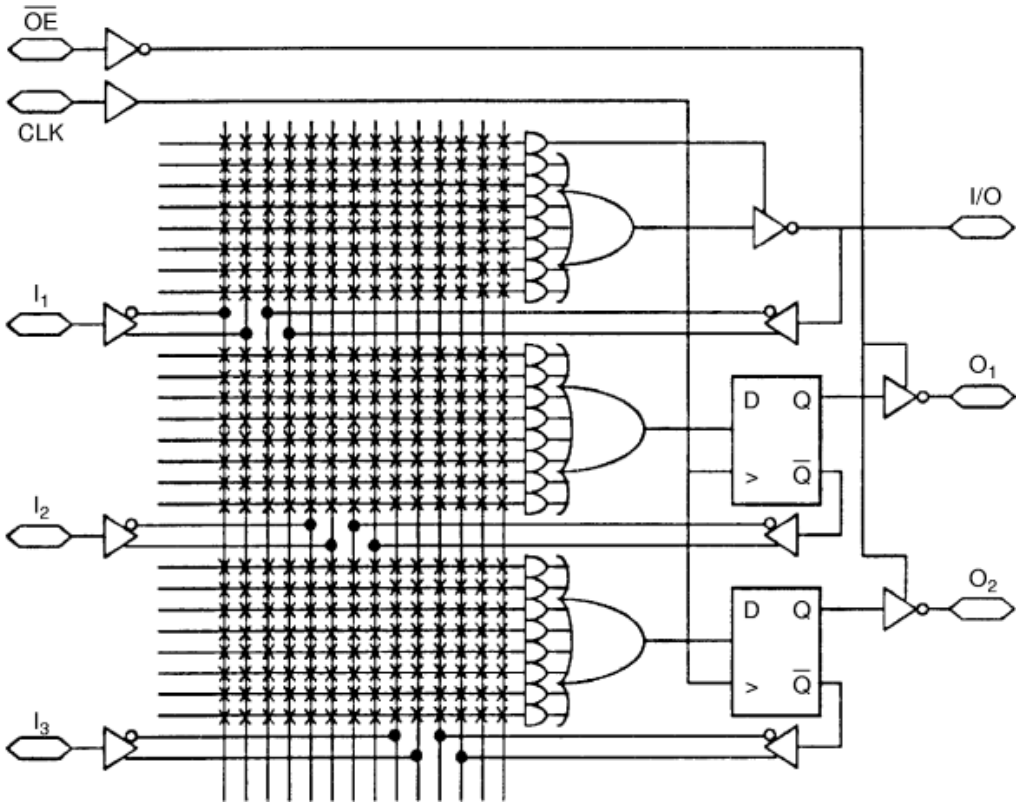

Fig. 1. Programmable Array Logic devices are used to implement small and medium scale integrated circuits. The fixed OR gate responds faster than the programmable OR matrix in the PAL device.

The AND/OR structure of the PLA device can be described as an implementation of sum-of-products Boolean equations. PLA devices are used to implement simple circuits which require high system performance. Many PLA devices include latched outputs in addition to combinatorial outputs and may provide bidirectional I/O pins.

Like the PLA device, a programmable-array logic device (*PAL*) is a fine-grain logic structure which implements sum-of-product terms with a programmable AND array and a fixed-OR gate (3). The fixed-OR gate responds faster than the programmable OR matrix on the PLA providing improved performance at the cost of reduced logic flexibility.

Figure 1 shows a generic PAL matrix including two latched outputs and a pin which can be a combinatorial output, an additional input pin, or a bidirectional pad. Output signals in a PAL device are typically fed back into the array, providing the resources for counters and other state machines. The input and feedback lines are permanently connected to their own specific column-routing wire. The programmable AND array is implemented with a programmable switch located at the point where each horizontal wire feeding an OR gate crosses a column wire (the X's in Fig. 1). These programmable switches implement a wired-AND function on the signal feeding the OR gate with each column signal to which it is connected through a closed programmable switch (3). The programmable AND array generates custom product terms which are then summed together using the fixed-OR gate. This design approach works well because all combinatorial logical expressions can be decomposed into a sum-of-products form.

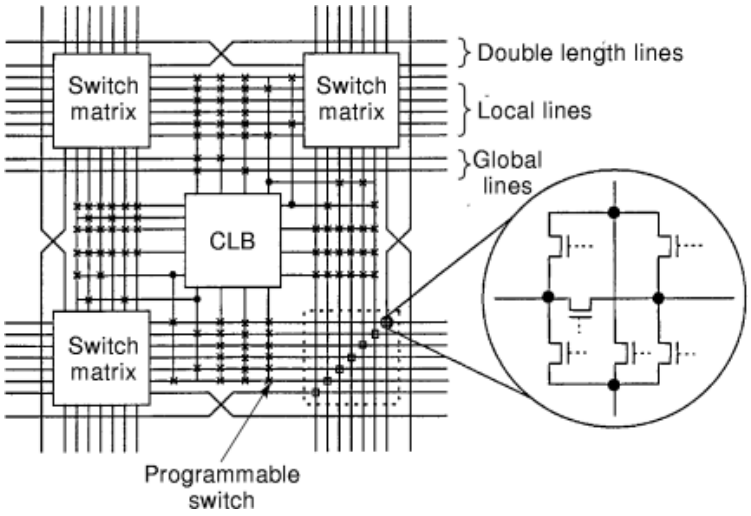

Fig. 2. The routing resources surrounding a CLB consist of local, double length, and global lines. The switch matrix provides full connectivity resources at the intersection of row and column signals.

The simple, two-level structures of the PLA and PAL devices provide high-speed implementation of simple logic circuits. To clarify the terminology, the term programmable-logic device often refers to a set containing only the PAL and PLA device structures (2), whereas some authors refer to the term programmable-logic device as any logic device whose function is defined by the user (3,4). For the purposes of this discussion, the latter definition is used.

Reconfigurable, Field-Programmable Gate Array. An FPGA is an array of configurable logic blocks interconnected through a network of programmable, segmented, routing resources (5). There are several types of FPGAs available in a variety of architectures, all providing varying levels of logic densities, I/O resources, and system performance. One-time programmable FPGAs provide higher logic densities and better system performance because of lower interconnect impedance (6). Reconfigurable SRAM-based devices may perform more slowly, but they support design modifications and in-system reconfiguration. A reconfigurable FPGA is discussed to provide insight into an FPGA structure.

A Xilinx XC4000 series FPGA uses static RAM memory to hold the configuration data for the device, providing a coarse-grained, high-density reconfigurable-logic array. The FPGA consists of an array of configurable-logic blocks (*CLB*) connected by segmented programmable routing (7). Figure 2 shows the interconnect surrounding a configurable-logic block for the XC4000 series FPGA.

There are three types of interconnects in the XILINX 4000 series FPGA: local lines, double-length lines, and global lines. The CLB inputs connect to the rows and columns of the local lines through a programmable switch, and the outputs connect to both a row and a column set of local lines. The local lines are connected to each other and to the double-length lines through the switch matrices where each column and row of local and double-length lines cross. Each row line is connected to a column line through a structure of six pass transistors, allowing any wire segment entering the structure to be connected to any number of other wiring segments. This switch matrix, combined with the ability to wire any CLB input to any of its adjacent local lines, provides the FPGA with a complete network of routing resources (7). The double-length lines connect to alternating switch matrices providing faster routing resources for signals which must travel a longer distance across the FPGA. CLB inputs and outputs may access these signals only by routing them to a local line which further reduces loading on the wire. Signals that must traverse the full chip may be routed on global lines.

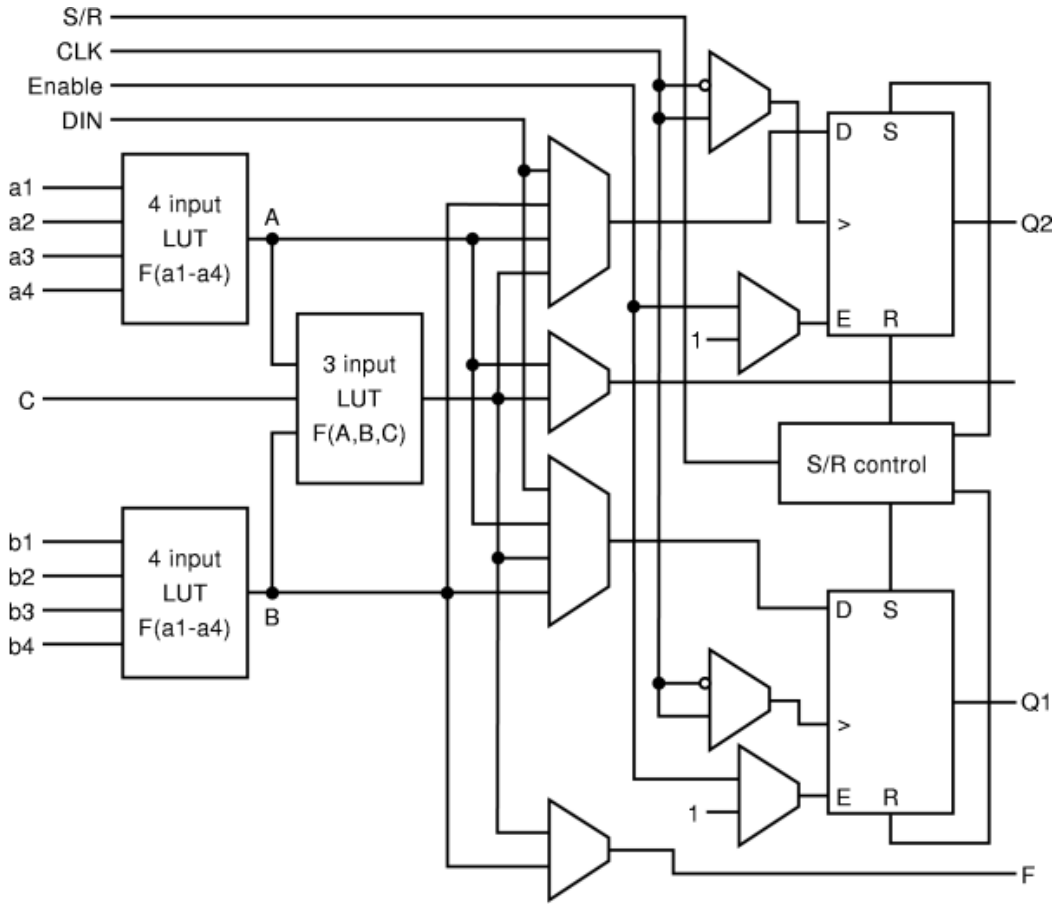

Fig. 3. The configurable logic block contains resources to implement a 16-by-1 bit RAM cell in each 4-input lookup table. Additional resources include fast internal carry paths for implementing two bit adders.

The global lines are driven by a CLB using a tristate buffer, and the CLB inputs have access to the global lines that run past it. Each global line has a bidirectional, tristate buffer at its midpoint to regenerate the signal. Alternatively, the tristate buffer can also be used to logically disconnect the two wires to provide two global lines which run half the length of the chip (7).

The function generators in the CLB shown in Fig. 3 are lookup tables (*LUTs*), programmed during configuration to provide any function of the applied input signals. The Xilinx 4000 series CLB contains two 4-bit LUTs, one 3-bit LUT, and two D-latches. The select lines for the various multiplexers throughout the circuit are controlled by the configuration bits programmed into an SRAM. Each CLB provides two latched outputs and two combinatorial outputs. The S/R signal (shown in Fig. 3) provides the input to a programmable set/reset control block. This control block sends the necessary set and reset signals to the two D-latches based on the configuration bits and the input signal. A separate global reset signal clears the D-latches during power-up or system reset.

The CLB has several special configurations to provide additional functions at higher speeds or increased logic density. The Xilinx XC4000 device family provides the routing resources to access the configuration bits in the two lookup tables. In this configuration, each CLB may be routed to implement either two 16×1 bit or one

32×1 bit memory array. The read access time to these on-chip memory structures are about 5 ns, and the write time is about 10 ns. The XC4000 CLB also contains special hardware to implement a two-bit adder with fast internal carry logic. In this configuration, the FPGA can implement adders that complete their computation in less time and with fewer CLBs than the structure described in Fig. 3 (7).

The structure of the logic block used in FPGA design is a topic of considerable study. The logic block for lookup-table-based FPGAs can be optimized to reduce the required chip area (8) or may be designed to optimize performance (9). These studies use benchmark circuits and various logic-block structures for their analysis. The results show that logic blocks with four-input lookup tables achieve the highest functionality per unit area. Furthermore, the added area required for decomposable lookup tables, multiple function block outputs, and flip-flops within each function block all increase the overall functionality per unit area of the FPGA (2). Analysis of FPGA performance for similar logic-block structures shows that five-input or six-input lookup tables provide designs with optimum performance (9).

Each signal pin on the package of the FPGA has an input/output block (*IOB*) associated with it. Each block may be programmed as a combinatorial input, combinatorial output, latched input, latched output, bidirectional pad, or a tristate output. Furthermore, the pins required during device programming may be used as I/O pins once the configuration is complete. Embedded in the IOBs are test structures compatible with the IEEE Standard 1149.1 for chip and board-level, boundary-scan testing (10,11).

The segmented, programmable interconnect of the FPGA reduces system performance because the RC loading on each wire is large with respect to conventional wires. This loading results from the fact that the segmented, programmable interconnect is comprised of series pass transistors, each adding resistance and capacitance to the net. This loading reduces FPGA performance to a degree where system performance is about three times slower than masked, programmable gate arrays (2). A second major problem with FPGA technology is the relatively low logic density compared with masked programmable gate array or custom standard cell layout. Programmable interconnect requires more area than standard wire routing channels, and additional space is required for the programming circuitry and configuration data (2).

Complex Programmable-Logic Device. Complex programmable logic devices (*CPLDs*) contain continuous routing tracks for programmable interconnects as opposed to the segmented routing resources used in FPGA technology. As a result, the CPLD has greater predictability of signal delays before routing and less signal skew after (5). Segmented routing provides larger routing resources for interblock routing, so many CPLD structures use a coarser grain logic structure to reduce the amount of routing resources required between logic modules.

Many CPLDs further reduce interconnect loading effects by controlling the programmable connections with multiplexers rather than pass transistors. This ensures that signals maintain good rise and fall time along with good system performance, provided that the propagation delay of the multiplexers is on the order of the loading delay in an equivalent, segmented-signal route.

Altera FLEX Family. An example of a complex programmable-logic device is the coarse-grained, high-density Altera FLEX 8000 (4). The configurable-logic structures, called logic elements (*LE*), are similar to CLBs in field-programmable gate arrays. Each logic element has four data input signals connected to a programmable interconnect (discussed later), two inputs (carry-in and cascade-in) from the previous LE in the array, one output signal connected to the programmable interconnect, and two output signals connected to the next LE in the array (carry-out and cascade-out). The four data signals drive a lookup table which may be configured as one 4-input lookup table or as two 3-input lookup tables. The output function may be combinatorial or latched by an available D-latch controlled by three additional LE inputs (clock, clear, and preset).

Unlike the FPGA CLB, the logic elements are grouped into eight-bit clusters called logic-array blocks (*LAB*) shown in Fig. 4. The cascade and carry outputs in the last logic element in a LAB connect to the cascade and carry inputs to the first logic element in the next LAB of the logic array. Because the logic elements are connected in groups of eight, functions, such as eight-bit arithmetic that cannot be performed in one CLB can

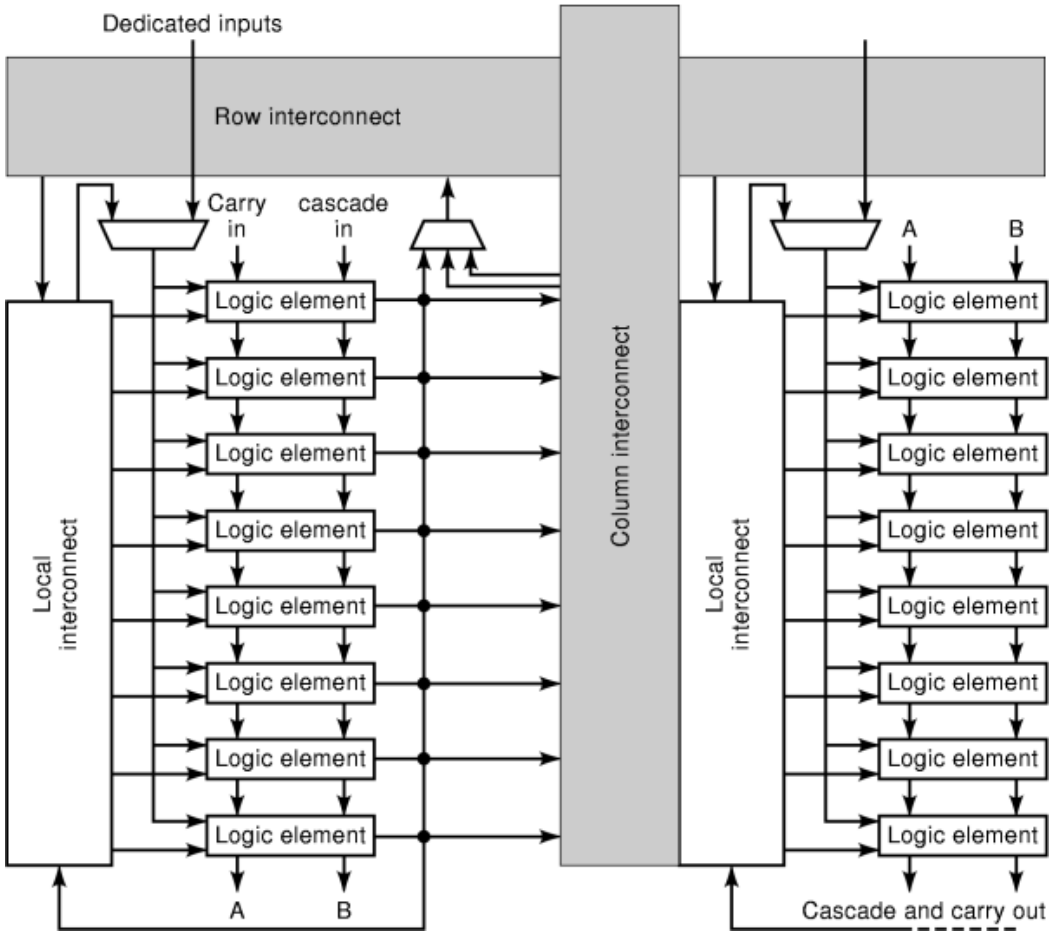

Fig. 4. The logic elements and fast local interconnect make up the logic array blocks in the course-grained FLEX family CPLD. Logic array blocks are connected through the multiplexer controlled row and column routing resources.

be performed in one LAB and operate much faster because of the high-speed, local interconnect provided for carry circuits (4).

The interconnect in the CPLD is continuous as opposed to the segmented interconnect used in field-programmable gate arrays. In the FLEX 8000 device, input signals may enter a LAB only through the row interconnect tracks. Signals on a particular column may connect to a row interconnect track, but a row signal may not connect to a column interconnect track unless it is fed through a logic element. Signal connections in this scheme are not bidirectional, so multiplexers may be used in place of pass transistors to make logic connections. Any logic element in a LAB may output a signal either on a row or a column interconnect track.

Signals enter and leave the device through pins connected to I/O elements (*IOEs*) which have a programmable structure that may be configured as a combinatorial output, a combinatorial input, a latched output, a latched input, or a bidirectional input/output signal. IOEs connect to both row and column interconnects and each IOE may drive or receive data from two row or two column signal tracks (4).

Vantis/Advanced Micro Devices MACH Family. A second CPLD architecture is the MACH family by Advanced Micro Devices (12). The MACH device contains several *PAL* blocks connected through a routing

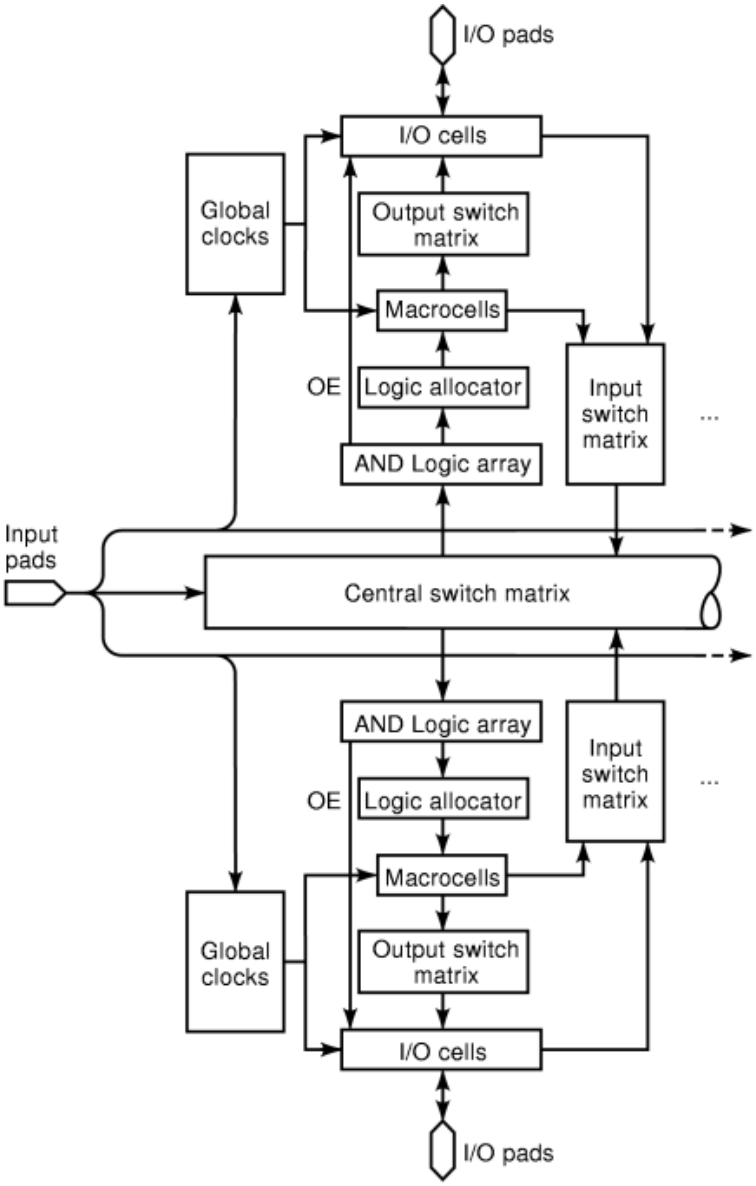

Fig. 5. Data computed in the logic allocator of the MACH CPLD must be routed through the input switch matrix before it may reenter the AND logic block to maintain consistent wiring delays.

matrix. This routing matrix uses continuous routing and does not suffer the interconnect loading effects of the FPGA-based segmented routing previously mentioned. The architecture of the MACH (shown in Fig. 5) is an array of PAL structures interconnected by programmable routing resources called the central switch matrix. The programmable switches are implemented through electrically erasable PROMs.

Each PAL block consists of a product-term array, a logic allocator, macrocells, an output switch matrix, an input switch matrix, I/O cells, and a clock generator. The product-term array is the programmable AND

array (discussed earlier), which receives input from the central switch matrix and sends the various product terms to the logic allocator. The macrocells receive the terms from the logic allocator and provide combinatorial or latched output signals to the input switch matrix and the output switch matrix. The output switch matrix provides connections to route the macrocells to any of the I/O cells in the PAL block. I/O cells may be configured to provide output from the switch matrix, input from the pin to the input switch matrix, latched input from the pin to the input switch matrix, or may be configured as bidirectional pins. The input switch matrix routes signals to the central switch matrix from either the I/O cells or the macrocells. To ensure consistent and predictable delays any feedback paths in the PAL block must be routed out of the macrocells, through the input switch matrix, and back to the central switch matrix, before it may reenter the PAL block (12). The clock generator provides low-skew, global-system clocks to all PAL arrays.

Field-Programmable Interconnect Component. Programmable interconnect provides a medium for interconnecting several components through configurable wires. The most important aspects of a programmable interconnect component are small signal impedance and large I/O count. The greater the I/O count, the more resources are available for routing components together. If the routing impedance is too great, however, system performance is significantly affected by the device. One example of a programmable interconnect device is the field programmable interconnect component (*FPIC*) by Aptix Co. (1) This device contains a high density of passive programmable interconnects providing an array of 936 bidirectional signal pins. SRAM cells control the network of routing tracks and I/O rails in the component so that any signal pin can be logically connected to any other through a series of pass transistors.

Timing. The routing paths provided by the FPIC components behave as passive *RC* networks. The resistance through a pass transistor is significantly higher then the resistance of a wire in a standard routing channel. Furthermore, the effective resistance of an individual pass transistor depends on the voltage of the signal passed. As the source voltage approaches the gate voltage of the pass transistor, the drain-to-source resistance increases. The higher resistance and capacitance caused by the pass transistors in programmable interconnects significantly reduce overall system performance.

The Aptix development system provides the user with the means to specify critical paths to the routing software. Then the router will try to generate a route which minimizes the resistance and capacitance on that particular net. This software also estimates the *RC* loading on the signal paths, once the part has been routed (1).

The routing channels must be treated as a distributed *RC* network as opposed to a lumped *RC* load. This effectively creates a propagation delay through the component in addition to impacting signal rise and fall times. To reduce the effects of the FPIC loading, devices providing a high output current should be used to generate signals, and devices with a high input impedance should be used as receivers.

Regenerative Feedback Repeaters. To reduce the *RC* loading caused by pass-transistor interconnects, a signal can be buffered with the tristate bidirectional buffer shown in Fig. 6. Two problems with this method of signal regeneration are that these buffers may consume a large area and their use adds the buffer propagative delay to the overall delay of the signal. Though they may increase the response of the signal and reduce the quadratic delay of the signal with respect to the line length to a more linear relationship, this form of buffering still adds some inherent delay to the system. A better approach is to use a signal amplifier in parallel with the signal path, such as the regenerative feedback repeater (13) shown in Fig. 7.

During a low-to-high logic transition, the NAND gate will switch on once the voltage on the signal reaches its threshold voltage. This turns on the *PMOS* device, and the node has a path to power providing a fast rise time. This signal amplification continues for the length of time T_d required for the rising edge of the signal to propagate through the buffer chain and turn off the NAND gate. Likewise, during a high-to-low transition, the NOR gate switches on, turning on the NMOS device, which quickly pulls the node low until the falling edge propagates through the inverter chain and the NOR gate is shut off (T_d).

Regenerative feedback repeaters impose a maximum signal frequency on the system and do not respond well in the presence of signal hazards. Any signal propagating along the wire with a short pulse width is

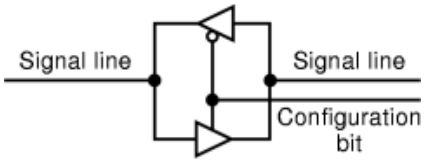

Fig. 6. Bidirectional tri-state buffers are used to regenerate a signal under a heavy load, or may be used to electrically isolate the signal lines.

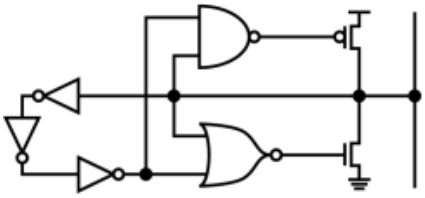

Fig. 7. Regenerative feedback repeaters are useful for amplifying a signal without adding an additional propagation delay. However, they are sensitive to signal hazards.

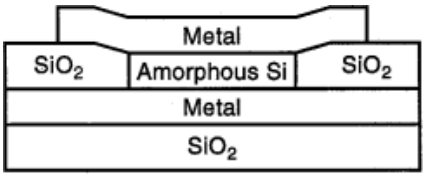

Fig. 8. During programming, an antifuse programmable switch is closed by providing a current through the amorphous Silicon, causing the metal to migrate through the region. The programming current reduces the resistance of a closed antifuse from 100 M ohms to less than a hundred ohms.

lengthened to T_d. Very small pulse widths are filtered out by the repeaters, and signals with a pulse width of about one-half T_d can cause metastability in the repeater. Problems with short pulse widths cannot be solved by making T_d arbitrarily small. If T_d is less than the propagative delay to the next gate, the driving gate shuts off before the receiving gate has turned on, possibly causing oscillations in the signal line. For these reasons, these repeaters are not applicable for FPGA devices. However, in FPIC devices where propagative delays tend to be larger and signal hazards are typically less frequent, regenerative feedback repeaters can significantly improve system performance.

One-Time, Programmable-Logic Devices. As mentioned, the resistance through the pass transistor can strongly affect system performance. One-time programmable devices (*OTP*) provide a dense network of programmable switches with very low resistance. Unlike other programmable devices, however, these connections are permanent. This write-once technology is particularly useful for low-cost, low-volume, high-performance, application-specific integrated circuits (*ASIC*).

The fuse and the antifuse are two examples of OTP connections. The fuse contains a thin metal contact at each programmable interconnect site. When the device is programmed, a large programming current is applied to the contacts where an open circuit is desired, burning out the fuse and disconnecting the wires.

The antifuse programmable switch is similar to the fuse, but contacts begin in the open state and are closed during programming. The resistance of the antifuse programmable switch is about 100 MΩ in the open state and about 50 to 80 Ω in the closed state (6). The structure of the antifuse is shown in Fig. 8.

The antifuse consists of a thin layer of amorphous silicon between the crossing layers of interconnecting metal. During programming, a relatively high voltage (10 V) is placed across the two metal conductors inducing a current through the amorphous silicon. The current causes metal from the contacts to migrate into the amorphous silicon, resulting in a low resistance via between the two metals. These one-time programmable switches require very little space compared to transistor-based programmable switches, but some space is needed for the high-voltage transistors required to handle the large programming voltages and currents. OTP devices provide better performance than the pass-transistor-based devices due to lower interconnect impedance.

Applications

Programmable-logic devices are typically used for rapid prototyping, design implementation, and reconfigurable systems. PALs are typically used in place of several small-scale integrated (*SSI*) circuits to reduce the area required for circuit boards. FPGAs or CPLDs may be used to further reduce the area required for implementing a design in systems where speed is not critical. When implementing designs or developing prototypes requiring multiple PLDs, partitioning the logic is an important factor in determining system performance and required routing resources.

In-system reconfiguration of the SRAM-based PLDs provide a means for altering the functional behavior of a system and can implement a dense network of logic where several tasks operating at different times may be mapped to the same hardware. This design methodology may also implement reconfigurable computers where a processor's physical hardware can be altered from instruction to instruction.

Field-Programmable Circuit Boards. Field-programmable circuits boards (*FPCB*) provide flexible routing resources for interconnecting multiple devices. These boards are typically used for prototyping or implementing large systems. FPCBs consist of a matrix of programmable-logic and other devices wired together or connected through a field-programmable interconnect component.

The FPIC-based field-programmable circuit boards provide the user with a medium for testing systems that include components which cannot be translated to a programmable-logic device. The Aptix MP3 System Explorer (14) uses three FPIC devices to provide a programmable interconnect to two grids of holes for the insertion of components with DIP packaging (100 mil, 300 mil, 400 mil, and 600 mil). These FPICs also provide interconnects to the I/O pins of two Xilinx XC4000pg223 devices. In addition to programmable logic, these FPGAs are *wired* to two 72-pin I/O headers used to connect the target system to the MP3 emulator. Additional FPGAs may be added to the emulator through a socket adapter that connects to the board's hole array for DIP components. A PAL is used to provide eight global lines to the system. Using the MP3 system, prototypes may be rapidly developed without the need to fabricate printed circuit boards. This type of emulation is particularly useful for design verification where components are used which do not have an available model for simulation.

A second example of a field-programmable circuit board is the Aptix AP4FPCB (1). This board is typically used to provide programmable interconnects for a large number of programmable logic devices to implement or emulate large designs. This field-programmable circuit board has the potential to interconnect up to 21 FPGAs and is typically used for emulators in which the design is mapped to several FPGAs. The programmable interconnect of the Aptix AP4 board (shown in Fig. 9) is provided by four field-programmable, interconnect components (described earlier). Each FPIC in this system has 157 bidirectional lines that run to the I/O pins on each of the four core FPGAs and 45 lines that run to the I/O FPGA for its quadrant. The FPICs are connected to each other by one hundred lines that run between each adjacent device, and ninety lines that run between the FPICs diagonally across on the board. In this way, any two FPGAs can be routed to each other by traversing at most two FPICs.

The core FPGAs are any Xilinx XC4000 series device in the Xilinx 191-pin package. A 96-pin DIN connector is interfaced to the FPICs through the four I/O FPGAs (XC4005pg156) previously mentioned. A central FPGA (XC4003pg120) is programmed with a design provided by Aptix to deskew and distribute five signals to the

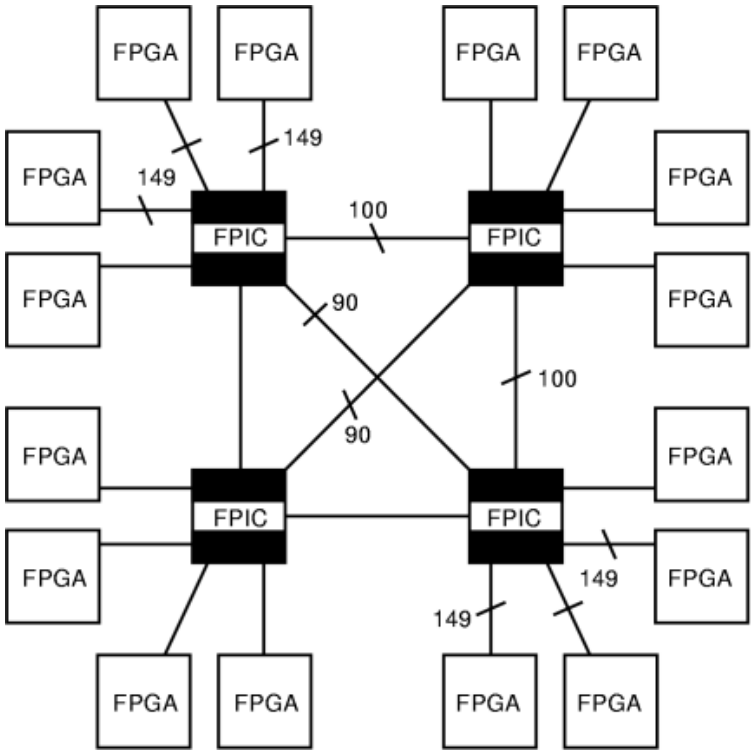

Fig. 9. The AP4-FPCB is used for implementing large systems in multiple field programmable gate arrays. The FPIC devices provide the programmable interconnect between the FPGAs and other system components.

clock inputs of every FPGA in the system. The programmable routing resources to one cluster of FPGAs may be used for a small grid of holes used to insert DIP packaged components similar to those described in the MP3 FPCB.

Prototyping. The rapid evolution of integrated electronics demands that design teams create systems with increasing complexity in shorter periods of time. Design verification often becomes a bottleneck in product development cycles (1). Software development requires gate level simulation, behavioral model simulation, or a prototype of the product under development to verify software modules. Low-level simulation of a system for software development provides an accurate timing environment for verifying the system hardware and software under development. This form of simulation, however, is computationally intensive, and running a large quantity of vectors on a complex system can often take a prohibitively long time. For this reason, a simplified behavioral model is often used as a platform for software development. These behavioral models are created from the behavioral descriptions of the system and are developed in parallel with the hardware implementation as opposed to extracting a functional model from the actual implementation. This can often lead to minor undetectable inconsistencies between the two systems, resulting in the development of software on an erroneous behavioral model. Prototyping provides a middle road between these two extremes, and programmable-logic devices provide a reconfigurable test bed for such prototyping implementations. Though PLD implementation does not preserve critical timing information in synchronous systems, a prototype of the actual system can be obtained by mapping the original design onto programmable-logic devices and running at slower speeds. This allows functional verification and provides an accurate functional model for software verification. The larger the system to be implemented, the faster a prototype runs with respect to a timing-specific software simulation.

For designs the size of a 32-bit RISC processor, the prototype often runs instructions thousands of times faster then a software simulation.

System-level designs often use components supplied by vendors who do not provide simulation models for their products. Under these circumstances, development teams must create their own simulation models of the components for system verification. Developing such models involves spending more critical time in the verification phase of the design process and these models may be prone to error. The use of programmable interconnect in a PLD emulator provides a means of connecting vendor-supplied components used in the final system implementation during the verification process.

Aptix field-programmable circuit boards provide programmable routing resources for connecting multiple FPGAs and custom ASIC circuits for system prototyping. The Aptix development system provides software which partitions a large design into portions small enough to fit in the FPGAs provided on a field-programmable circuit board and automatically route the signals necessary to connect the partitioned design through the FPIC devices. Critical paths in the design may be indicated in the netlist prior to partitioning, and the algorithm keeps an entire critical path on one FPGA, if possible. Software partitioning may save development time, but it typically results in slower system performance. Slow system performance in an emulator used for prototyping and design verification may not initially cause problems. However, if it is used extensively in code development, the system could prove cumbersome.

Manual partitioning of a design results in better system performance. There are several factors to consider when partitioning logic. For instance, in a microprocessor, a bit-slice partitioning scheme typically contains a control module and nearly identical pieces of a datapath, each containing its own slice of the processor that is a set number of bits wide. For example, a bit-slice of a 32-bit processor partitioned into four datapath bit slices would each contain an eight-bit ALU, an eight-bit register file, an eight-bit shifter, and the various supporting logic. The advantage to bit-slice partitioning is that the routing resources required to connect the various components are typically less than for other partitioning algorithms. The disadvantage to bit-slice (or vertical) partitioning is reduced system performance. In a bit-slice design, critical timing paths, such as the ALU carry chain, must traverse several components through the top-level interconnect, resulting in greater delays and lower system performance.

An alternative to bit-slice partitioning is a horizontal partitioning scheme in which the logic is partitioned along pipeline stages. This partitioning typically results in faster system performance because all critical paths are contained within one component. The disadvantages to this partitioning strategy is that a great deal of top-level interconnect is required, and the sizes of the various design components are not symmetrical.

Mapping the design to programmable logic should be an automated process to ensure that the emulated logic is equivalent to the original design. If logic synthesis is used to generate the original design from a hardware description language, retargeting the netlist to the programmable hardware will not be a problem because it involves only synthesizing the models to a different logic family. When synthesis is not used, however, this task may become considerably more difficult. If the design is implemented in a differential logic family (such as cascade voltage switch or current-mode logic), automated netlist retargeting becomes a very complex task. A logical inversion in a differential logic family is achieved simply by switching the differential pair at the input or output of a logic gate. The single-ended netlist retargeted to the programmable logic must have an inverter inserted at each differential inversion. Recognizing when an inversion takes place in a differential netlist can be quite difficult. For example, simple renaming of a differential pair, such as A[1:0] becoming A[0:1], indicates a differential pair inversion, and an inverter must be added to the netlist as it is retargeted to the programmable logic.

An example of an emulated system is the translation of the F-RISC/G GaAs/AlGaAs RISC processor to an Aptix field-programmable circuit board (15). The F-RISC processor is partitioned into a 24-chip, bit-slice design because of the low yield of the process and is interconnected through a high-performance, multichip-module package. The module contains one instruction decoder chip, four 8-bit data path chips, two cache controller chips, one clock deskew chip, and sixteen cache memory chips (16).

The netlist for each chip in the core processor of the actual architecture is mapped to one of the core FPGAs on the Aptix board. The emulator cannot verify timing-specific or analog designs, so the deskew chip was excluded from the emulator. The cache memory chips were not translated to PLDs but were implemented using standard off-the-shelf memory components for reasons of cost and board space. The logic arrays and the memory are interconnected using the programmable interconnect provided by the FPICs.

The emulator provides full functional verification of the processor at the gate level. Several design flaws involving interrupts and traps were detected by the emulator, which had been missed using the software simulator for verification. Code development began using a software behavioral model of the processor. The compiled code was executed on this behavioral model, and the results were saved for verification purposes. Most discrepancies between the behavioral model results and the emulator results were design flaws. However, one such disagreement found in the results was a problem with the behavioral model. In identifying this problem, the emulator prevented the continuation of code development on a faulty system model.

In-System Programming. One major advantage of an SRAM-based device is in-system reconfigurability. In such applications, an external storage device (such as an EPROM) is used to hold the configuration data for the PLD. Design changes involving system upgrades or correcting design flaws requires only alterations to the configurational data used to program the PLD. Replacing an EPROM containing configurational data is often less costly than replacing integrated circuits, and systems may be designed to provide hardware modifications from remote locations (2).

A second application of reconfigurable systems is to reduce the overall amount of hardware required by a system through time-multiplexing mutually exclusive tasks on the available reprogrammable hardware. The configuration EPROM is programmed with the desired FPGA configurations, and an external controller pages from configuration to configuration in the EPROM, reprogramming the PLD when a new hardware task is required. In-system programming can significantly reduce the amount of hardware required to implement a particular application at the cost of reduced system performance.

The audio studio system by Metalithic Systems is an application of in-system programming using FPGAs. This unit is implemented with two XC3090 field-programmable gate arrays. A mini-RISC processor implemented in one of the device operates as the system controller. The second FPGA contains the logic for a vector processor (performing multiply-accumulates and linear interpolation) and lookup tables for a synthesizer engine. The various computations for the 128-channel system implemented in the reconfigurable FPGAs would require an estimated 288 conventional digital signal processing (*DSP*) chips (17).

Reconfigurable computing is another application of in-circuit reconfigurable devices. Three limitations of standard computer processors are that their set of instructions is fixed, their operands are a specific size, and with the exception of the recent MMX technology, they are limited to one instruction per execution unit per clock cycle (18). Reconfigurable processors, however, may be programmed to optimize performance for specific operations and then may be reconfigured upon completing the task. Using these systems, software programs can be compiled into application-specific hardware where execution is considerably faster (2). The Spyder reconfigurable superscalar coprocessor is an example of a generic reconfigurable coprocessor. Three configurable execution pipelines are connected to the processor's memory through two register files. Once the application-specific hardware is designed and routed to the programmable logic making up the execution units, the microcode used to control the pipelines is written in high-level programming language similar to C++. Then the application program running on the host computer is written to send the necessary data to the reconfigurable coprocessor and later processes the results. The feasibility of a compiler which transforms a single high-level program into the necessary code for the application program, the control store, and the functional description of the execution units is a topic of future study (18).

Dynamically Reconfigurable Logic. Programmable logic for system implementation provides short design time at the cost of reduced logic density and system performance. In-system reconfiguration can increase the effective gate count of a reprogrammable PLD at the cost of additional system performance. Each time the PLD must be reconfigured, processing on the device must be stopped, and the new configuration data must be

downloaded before processing can continue. The time required to reprogram a PLD depends on the size of the device, the method of programming, and the speed at which it can be programmed with configurational data. Completely reprogramming a PLD may take anywhere from a hundredth to a tenth of a second (19). In systems running hardware algorithms requiring frequent reconfiguration, one may find that the reconfigurational time is the limiting factor in system performance.

Dynamically reconfigurable systems allow reprogramming unused logic while computing continues in active circuits. For large systems, where logic is implemented on several programmable logic devices, reconfiguration of devices not currently used in processing is possible if the programmable devices (*CPLD*s or *FPGA*s) have in-system configuration algorithms that work with the *IEEE* 1149.1 *JTAG* interface port. This IEEE serial testing standard allows JTAG-complaint devices (both programmable and nonprogrammable/fixed-function parts) to be daisy-chained, and programming data and instructions can be serially loaded into the target device while all the other devices continue operating normally. Recent advances in FPGA technology have introduced dynamically reconfigurable FPGAs. These devices are capable of dynamically reconfiguring inactive logic while active circuits within the device continue processing.

A dynamically reconfigurable system typically begins with a scheduling algorithm to determine which functions in the implementation are mutually time-exclusive. Then these functions are mapped into the same routing resources and are scheduled by the system scheduling controller to be swapped into the hardware. The scheduling controller must determine which function to route to the hardware, based on a system state or input.

The Xilinx XC6200 family of field-programmable gate arrays is an example of a dynamically reconfigurable FPGA (20). The configurational data for these devices are not scanned in serially but rather provide parallel addressing of the configurational RAM similar to that of computer memory. Each CLB and its local routing resources are controlled by a local block of memory locations. The XC6200 FPGA is a fine-grain, programmable-logic device containing simple multiplexer-based functional units with a single D-latch, three inputs, and one output. Previous discussion described fine-grain logic structures with lower functionality per unit area and slower performance. A benefit to this fine-grain logic structure is that small tasks may be mapped more easily to the hardware surrounding active circuits.

When circuit switching occurs between two tasks, the current state of the circuit swapping out must be saved, and the previous state of the circuit swapping in must be restored. The Xilinx XC6200 FPGA provides for this necessity by mapping the state of the D-latch in each functional unit to the configurational data. When circuit switching occurs, the state of the latches may be read or written as necessary. However, state information in any latch created by the combinatorial logic available in the functional units is lost during device reconfiguration. The additional complexity of dynamic reconfiguration limits effective gate count and logic complexity. Thus dynamically reconfigurable devices are smaller and less complex than standard reconfigurable FPGAs.

Conclusion

Field-programmable logic devices are typically used for prototyping, reconfigurable systems, and design implementation where fast time to market is necessary. As the density and performance of PLDs continue to increase, programmable-logic devices will replace ASIC and mask-programmable gate arrays in more applications. Vendors of programmable-logic devices estimate that complex programmable-logic devices soon will be available with 250,000 gate equivalents (21).

A beneficial characteristic of programmable-logic devices is their high I/O counts. This provides an inherently parallel structure useful for I/O processing, bus interfacing, and for hardware algorithms which do not depend on multiply-accumulate steps. DSP is a popular application of system implementation in PLDs because the short design cycle permit products to reach the market sooner. One-time programmable devices are

typically used for DSP applications where performance is a crucial factor. An analysis of FPGA structures for DSP applications shows that an XOR-based functional unit with five inputs and two outputs provides the optimum logic density and performance for most DSP applications (22). This study indicates that application-specific, programmable-logic devices may soon come under development.

BIBLIOGRAPHY

1. Aptix, *Programmable Interconnect System Data Book*, San Jose, CA: Aptix Co., 1993.
2. S. Brown *et al. Field-Programmable Gate Arrays*, Norwell, MA: Kluwer Academic Publishers, 1992.
3. Advanced Micro Devices Co., *PAL Device Data Book and Design Guide*, Advanced Micro Devices Co., Sunny Vale, CA, 1995.
4. Altera Co., *Flex 8000 Handbook*, Altera Co., San Jose, CA, 1994.
5. D. Amos Interconnect trade-offs: CPLD vs FPGA, *Electronic Engineering*, **67**: 81–84, 1995.
6. K. E. Gordon R. J. Wong Conducting Filament Of The Programmed Metal Electrode Amorphous Silicon Antifuse, *IEEE IEDM*, **93-27:** 6.2.1–6.2.4, 1993.
7. Xilinx Inc., *The XC4000 Data Book*, Xilinx Inc., San Jose, CA, August 1992.
8. J. R. Rose *et al.* Architecture of field-programmable gate arrays: The effect of logic block functionality on area efficiency, *IEEE J. Solid State Circuits*, **25**: 1217–1225, 1990.
9. S. Singh *et al.* The effect of logic block architecture on FPGA performance, *IEEE J. Solid State Circuits*, **27**: 281–287, 1992.
10. IEEE Standard 1149.1-1990.
11. IEEE Standard 1149.1-1990 Supplement A.
12. Advanced Micro Devices, *MACH 3 and 4 Family Data Book*, Advanced Micro Devices, Sunnyvale, CA, 1994.
13. I. Dobbelaere M. Horowitz A. El-Gamal Regenerative feedback repeaters for programmable interconnections, *IEEE J. Solid-State Circuits*, **30**: 1246–1253, 1995.
14. Aptix Co., *MP3 System Explorer Data Sheet*, California: Aptix Co., San Jose, CA, December 1994.
15. S. Carlough *et al.* Design verification and emulation of a multichip high-speed GaAs RISC processor using soft-programmable logic, *IEEE Int. ASIC Conf.*, Austin, TX, 1995, pp. 164–166.
16. H. J. Greub FRISC-E: A 250-MIPS hybrid microprocessor, *IEEE Circuits and Devices*, **6**: 16–25, 1990.
17. F. Granville Composing music on the PC: A new gig for reconfigurable computing, *EDN*, **41**: 37, 1996.
18. C. Iseli E. Sanchez Spyder: A SURE (superscalar and reconfigurable) processor, *J. Supercomput.*, **9**: 231–252, 1995.
19. P. Lysaght J. Stockwood A simulation tool for dynamically reconfigurable field programmable gate arrays, *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, **4**: 381–390, 1996.
20. Xilinx Co., *XC6200 Field Programmable Gate Arrays*, Xilinx Inc., San Jose, CA, January 1997.
21. J. H. Mayer CPLDs Push Performance, Density Limits, *Computer Design*, February Supplement: **36**: 22–23, 1997.
22. M. Agarwala P. Balsara An architecture For A DSP field-programmable gate array, *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, **3**: 136–141, 1995.

STEVEN R. CARLOUGH
PETE M. CAMPBELL
SAMUEL A. STEIDL
ATUL GARG
CLIFF A. MAIER
HANS J. GREUB
JOHN F. MCDONALD
MATTHEW W. ERNEST
Rensselaer Polytechnic Institute