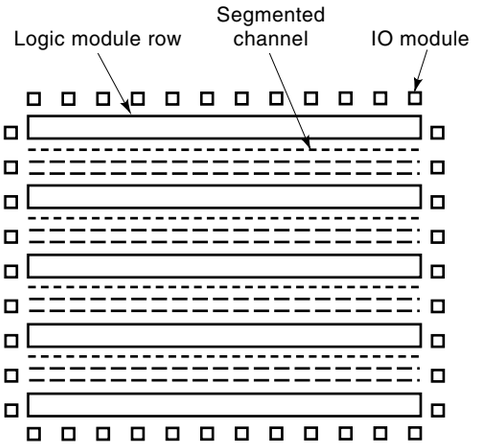

Figure 1. Row-based architecture consists of rows of logic modules separated by horizontal routing channels. The routing tracks in horizontal routing channels are segmented. Vertical routing resources are relatively limited compared with horizontal routing resources. IO modules are at the boundary.

CAD FOR FIELD PROGRAMMABLE GATE ARRAYS

Field-programmable gate arrays (FPGA) are one of the most popular electronic devices that circuit designers use. Because of the high complexity of circuit designs, software tools have become indispensable to the circuit designer in implementing circuits on FPGAs. This article discusses the internal mechanism of computer-aided design (CAD) software tools used by circuit designers to implement circuits on FPGAs.

FPGAs were first introduced into the market in the mid-1980s to combine the field programmability of programmable logic devices and the high density of gate arrays. Compared to the traditional application-specific integrated circuit (ASIC) technology, FPGAs have the advantage of rapid customization with negligible nonrecurring engineering cost. The advantage of rapid turnaround with relatively low cost has led to increasing usage of FPGAs for a wide variety of applications, including rapid system prototyping, small volume production, logic emulation, and special-purpose reconfigurable computing.

Conceptually, an FPGA device can be visualized as composed of three types of basic components embedded in a two-dimensional grid: logic modules, routing resources, and IO modules. A logic module can be customized to realize various logic functions for different circuit designs. IO modules are located around the periphery of an FPGA device. Routing resources consist of routing segments in both vertical and horizontal directions. Usually, adjacent routing segments in the same direction are grouped together to form routing channels. Interconnections between logic modules are realized by routing nets through the routing channels. Row-based and symmetrical array architectures are two popular architectures used in commercial FPGA products. In row-based architecture (see Fig. 1), logic modules are grouped into rows separated by horizontal channels. Compared to the horizontal routing resources, vertical routing segments are much more limited. In symmetrical array architecture (see Fig. 2), routing channels are distributed evenly in both horizontal and vertical directions. Logic modules are surrounded by the adjacent routing channels.

Customization of logic modules and routing segments for implementing a particular circuit design is realized by programming a selected set of switches. A switch can be programmed into either a conductive state (on) or an insulative state (off). Physically, a switch can be implemented using an anti-fuse, or a pass transistor controlled by a static random-access memory (SRAM) cell, or other technologies. An FPGA device is reprogrammable if the device can be programmed multiple times. SRAM-based FPGAs are an example of reprogrammable FPGAs. Conversely, an FPGA device is one-time programmable if the device can be programmed only once. Anti-fuse-based FPGAs are one-time programmable. More description on architectural and physical details of FPGAs can be found in several references (1–3) (see PROGRAMMABLE LOGIC ARRAYS).

The density of a state-of-the-art FPGA device is over 100K gates and continues to increase rapidly. It is practically not feasible to design circuits on FPGAs without using sophisticated CAD software tools. While there are FPGAs of different

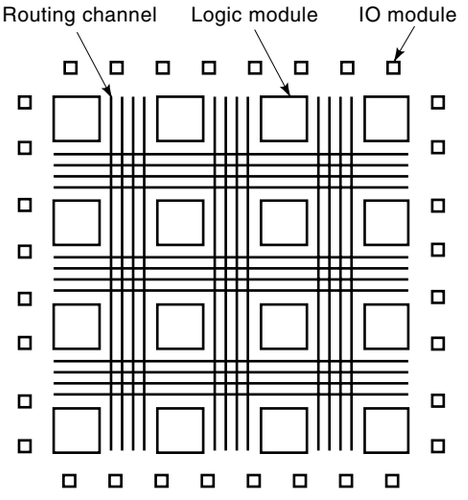

Figure 2. Symmetrical-array architecture consists of islands of logic modules surrounded by routing channels in both vertical and horizontal directions. Because of silicon area limitation, the intersecting vertical and horizontal channels in general are not fully connected.

architectures in both industry and academic research, the flows of the CAD software tool for any FPGA designs are similar and consist of several basic steps, as illustrated in Figure 3:

- *Design Entry.* Specify a circuit design by using schematic capture or hardware design languages (such as VHDL, Verilog).

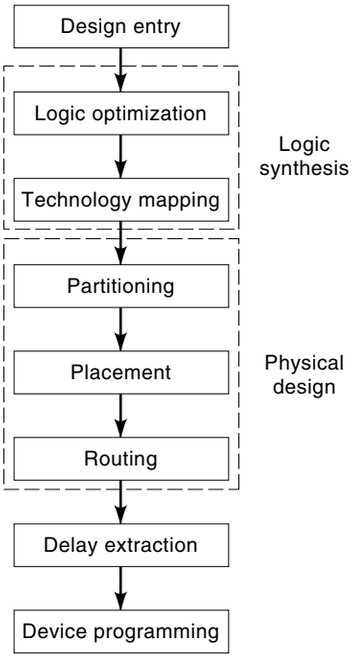

Figure 3. A typical CAD flow for FPGAs goes through the following steps: design entry, logic synthesis, physical design, delay extraction, and device programming. Logic synthesis and physical design steps each can be divided into several substeps, as outlined by the dashed lines.

- *Logic Optimization.* Transform the circuit network into another equivalent circuit network which is more suitable for the subsequent technology mapping step.
- *Technology Mapping.* Transform the technology-independent circuit network into a network of library cells of the target FPGA architecture so that the transformed network is functionally equivalent to the original circuit network.
- *Partitioning.* Partition the network of library cells into several subcircuits so that each subcircuit can be fit into a given set of resources of FPGAs.
- *Placement.* Assign cells of the circuit network to logic and IO modules on an FPGA device.
- *Routing.* Assign nets of the circuit design to the routing segments on an FPGA device. Select the set of switches that need to be programmed into the on state.
- *Delay Extraction.* Compute the routing delay with the physical routing information. Routing delay data will be used for post-layout circuit timing calculation and analysis.
- *Device Programming.* Program the selected switches into on state.

In the literature, logic optimization and technology mapping steps are also called *logic synthesis* and the software types for performing these steps are normally called *front-end tools.* On the other hand, the tasks of partitioning, placement, and routing are called *physical design* and programs for solving these problems are called *back-end tools.*

This flow for FPGA designs is very similar to that used in traditional ASIC technologies. However, the algorithms used for solving the problems encountered in the FPGA design flow can be very different from the algorithms used in ASIC technologies. Very often, it is necessary to develop FPGA-specific algorithms in order to obtain effective as well as efficient solutions.

The reason for having FPGA-specific algorithms is mainly because the resources in FPGAs are fixed and limited, and the architectural details of logic modules and routing resources vary significantly in different FPGA products. Strictly limited and fixed resources in FPGA devices post many constraints on feasible solutions. In comparison with the CAD problems in ASIC designs, the CAD problems in FPGA designs are generally more constraint driven than optimization driven. Finding a feasible solution for an FPGA CAD problem is usually more difficult than finding a feasible solution for an ASIC CAD problem. Practically, it is often acceptable to use as many logic modules and IO pins as available in an FPGA device as long as the utilization is under the resource limits and the solution is routable.

Currently, FPGA architectures are still in constant evolution. There is not yet a universal architecture that is used for different FPGA products. FPGA CAD algorithms, especially physical design algorithms, strongly depend on the architectural details. It is generally necessary to develop architecture-specific algorithms for solving CAD problems in various stages of FPGA design flow in order to fully take advantage of architectural features in different FPGA products.

In addition to these algorithmic differences, the primary advantage of quick turnaround of FPGAs dictates that CAD tools for FPGAs must run much faster than the CAD tools for

ASIC. Thus, more restrictions are imposed on the efficiency of critical algorithms for solving FPGA CAD problems.

This article will focus on the discussion of major FPGA-specific CAD problems in technology mapping, partitioning, placement, and routing. To keep the article concise, algorithmic details for solving the FPGA-specific CAD problems are omitted. In most cases, only the basic ideas of the algorithms are described and the references to the literatures that contain detailed descriptions are given.

General background information on logic synthesis and physical layout can be found in the literature (see CAD FOR MANUFACTURABILITY OF INTEGRATED CIRCUITS).

TECHNOLOGY MAPPING

Details of technology mapping algorithms vary for different architectures. The basic strategy of most FPGA technology mapping algorithms, however, consists of two basic steps: decomposition and covering. In the decomposition step, logic gates in the original circuit networks are decomposed into a different set of logic gates so that the transformed network is more suitable for achieving the optimization objectives such as area or timing. In the subsequent covering step, logic gates in the circuit are covered by cells in the library of the target FPGA device where each cell can be implemented by using a logic module.

The differences in FPGA technology mapping from the conventional approach result from the fact that the number of distinct logic functions that can be implemented with a logic module in most FPGAs is much larger than the typical library size for conventional ASIC technologies. It is therefore not practical to follow the conventional approach of enumerating all possible functions to determine the optimal selection of library cells. Logic modules in FPGAs can be broadly classified into two categories: lookup table (LUT) based and non-LUT based. Techniques used in technology mapping, especially in the covering step, are different for these two types of logic modules (4).

LUT-Based Logic Modules

A K-input LUT-based logic module can implement a total of 2^{2^K} distinct logic functions each with no more than K inputs. Examples of commercial FPGAs that use LUTs for logic modules include Actel's ES6500, Altera's Flex, Lucent's ORCA, and Xilinx's XC4000 product families. For values of K greater than 3, the size of the library for the library-based covering approach becomes impractically large. Many specialized algorithms have been developed to address the LUT-based FPGA technology mapping problem (5).

An important optimization objective for LUT-based technology mapping is to minimize the area, that is, the number of LUTs used for covering a circuit network. One fast and effective approach for LUT area minimization is to formulate the decomposition and covering problems as the bin-packing problem (6). The bin-packing problem is to pack a set of objects of given sizes into the minimum number of bins of fixed capacity. The bin packing problem is NP-hard, but simple, fast, and effective heuristic algorithms for solving the problem exist. The technology mapping results generated by using this approach are significantly better than the conventional approach in terms of both run time and area.

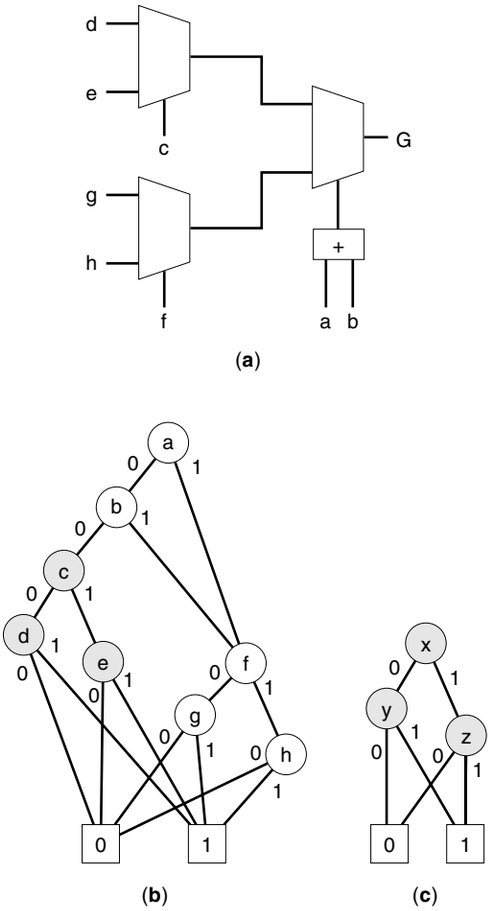

Figure 4. A multiplexor-based logic module in (a) can be represented by a binary-decision diagram (BDD) shown in (b). The BDD for a subcircuit represented by a BDD in (c) is isomorphic to a subgraph of logic module BDD in (b), as indicated by the shaded nodes.

Another important optimization objective is circuit performance. During logic synthesis steps, a commonly used performance metric is the maximum circuit level, i.e., the maximum number of cells on any path from a primary input to a primary output in a combinational circuit. It has been shown that the problem of minimizing the maximum circuit level for combinational circuits in the covering step can be solved optimally using the network flow technique (7). Furthermore, algorithms have been developed to achieve often a practically desirable balance between area and performance (5).

Non-LUT-Based Logic Modules

A K-input non-LUT-based logic module cannot implement every logic function with no more than K inputs. An example of a non-LUT-based logic module is the multiplexer-based logic module used in Actel's ACT FPGA families (see Fig. 4).

In the covering step for non-LUT-based FPGAs, an important operation is to determine whether a cover of the logic gate can be implemented by personalizing a non-LUT logic module. This problem is also known as the *Boolean matching* problem (8). For the logic module shown in Figure 4, the number of distinct logic functions implementable by a logic mod-

ule is more than 700, and thus makes it impractical to apply a conventional library enumeration approach.

A specialized technique for non-LUT-based logic module Boolean matching is based on a reduced ordered binary decision diagram (BDD) technique (9). Given a subcircuit logic function F and a logic module function G, BDDs for F and G, denoted as BDD_F and BDD_G, respectively, are constructed. Boolean matching of F on G is performed by detecting whether BDD_F is isomorphic to any subgraph of BDD_G. Figure 4 illustrates BDDs for a logic function $F = \bar{x}y + xz$ and the logic module shown in Fig. 4, $G = \overline{(a + b)}(\bar{c}d + ce) + (a + b)(\bar{f}g + fh)$. Function F can be implemented by G because BDD_F is isomorphic to a subgraph of BDD_G as induced by the shaded nodes in BDD_G. Technology mapping and fast Boolean matching algorithms using the BDD isomorphism approach have been developed for multiplexer-based logic modules (10,11).

PARTITIONING

In the partitioning step, a circuit is partitioned into a collection of subcircuits. Depending on the number of FPGA devices involved, FPGA partitioning could be either multiple-FPGA partitioning or single-FPGA partitioning.

In multiple-FPGA partitioning, a circuit is partitioned between multiple FPGA devices so that each subcircuit can fit into a single FPGA device. An example where multiple-FPGA partitioning is necessary is a logic emulation system. A logic emulation system verifies the functionality of a circuit design by implementing the circuit design on FPGAs running at a slower clock speed. Typically, a system-level design is too large for a single FPGA device and therefore must be implemented using multiple FPGAs.

Single-FPGA partitioning partitions a circuit within a single FPGA device and is most commonly used for hierarchical architecture FPGAs. In a hierarchical architecture FPGA device, routing resources between logic modules are not uniformly distributed. Instead, logic modules are grouped into clusters where each cluster contains a number of logic modules. Routing resources between clusters are normally much limited compared to the routing resources within a single cluster. Hierarchical architecture has the advantage of smaller device die size than a flat architecture for the same device density, and therefore is most popular for supporting high-density FPGA devices. In the physical design flow for a hierarchical architecture FPGA, a circuit is usually first partitioned into subcircuits so that each subcircuit can fit into a single cluster. Then, subcircuits are placed and routed within individual clusters.

Similar to the conventional partitioning problems, the most basic objective of FPGA partitioning is interconnection minimization between subcircuits. However, compared to the conventional partitioning problems, FPGA partitioning needs to satisfy more constraints in order to obtain a feasible partitioning solution. Finding a feasible partitioning solution is more difficult and important than in conventional partitioning problems. This is because the resources in an FPGA device, especially logic modules and IO ports (or the routing resources between clusters within a single FPGA device), are strictly limited. Consequently, FPGA partitioning problems are more resource-constraint driven than conventional partitioning problems.

Two most essential constraints for both multiple-FPGA and single-FPGA partitioning are IO constraint and capacity constraint. Capacity constraints for an FPGA device can be very complex. Driven by the demand of supporting system-level circuit designs, the FPGA device is becoming larger in terms of density as well as more heterogenous in terms of the types of resources. It is not uncommon to find a commercial FPGA device that contains different logic modules, complex IO modules, various speed grade clocks, embedded memory arrays, and dedicated resources designed for supporting special functions (e.g., wide input gates). Different types of resources on an FPGA device have different upper limits, and a feasible partitioning must satisfy the limitations for each of the different resources. To further complicate the capacity constraint, a logic function can be implemented by using different resources in an FPGA device. For example, a 2-input gate can be implemented using either a 2-input LUT or a 3-input LUT logic module. Consequently, the capacity constraint of an FPGA device cannot be accurately captured by simple measurements such as gate count upper bounds. In addition to the limitation on each type of resource, capacity constraints for an FPGA device need to take into account multiple choices of logic function implementation.

FPGA partitioning algorithms implemented in commercial CAD tools are usually based on traditional move-based approaches, such as Fiduccia–Mattheyses algorithm (12), with modification to incorporate FPGA-specific constraints into the algorithms. Starting with an initial feasible partitioning solution that satisfies the capacity constraints, the algorithms maintain the feasibility by allowing only the moves that do not violate capacity constraints. Initial feasible partitioning solution is usually not difficult to find if the device utilization is not close to the limitation. However, for partitioning problem where device utilization is approaching the resource limitations, finding an initial feasible partitioning solution can be challenging.

PLACEMENT

In the placement step, each cell in the circuit netlist is assigned to a module on an FPGA device. The two most important issues for FPGA placement are routability and performance. Because of the fixed routing resources available on an FPGA device, routability is usually treated as a constraint in the placement process. Net length minimization, which is usually the most important optimization objective for conventional placement problems, is only of secondary importance in FPGA placement. Circuit performance in FPGA placement is also typically treated as a set of timing constraints as specified by the circuit designer. Placement algorithms that consider timing constraints are called *timing-driven placement algorithms* in the literature.

Similar to the placement algorithms for ASIC technology, the placement steps in FPGAs consist of initial placement followed by placement optimization. Initial placement normally concentrates on general objectives, such as net length minimization, and uses constructive algorithms such as min-cut placement in order to achieve fast run time and reasonable quality. During placement optimization, initial placement results are further improved to ensure that the routing resource constraints are satisfied and other objectives, such as timing,

are optimized. Despite similarity to the ASIC placement approach, there exist several FPGA-specific issues that most FPGA placement algorithms need to address, especially during placement optimization, which is very often based on simulated annealing techniques.

Global Routing for Channel Density Computation

The numbers of routing tracks in routing channels are fixed for an FPGA device. A necessary condition for any feasible placement solutions is that the channel density in every channel cannot exceed the number of routing tracks available in the channel. Minimization of net length will tend to cause local congestion and produce a placement solution that is very difficult for subsequent global routing algorithm to generate a feasible routing. In order to calculate the channel density accurately, simulated annealing-based placement algorithms need to perform global routing iteratively for every move. Therefore, in addition to producing high-quality routing solutions, run time becomes a critical requirement for designing FPGA global routing algorithms. Such closely interleaved global routing and placement in FPGAs is different from the placement algorithms used in standard cell architectures, where channel heights can be adjusted and, therefore, global routing does not need to be embedded within the placement process.

Fast Interconnection Delay Estimation

Interconnection delay estimation for timing-driven placement for FPGAs is also very different from ASIC. Normally, an FPGA device contains routing tracks of various lengths in order to achieve delicate balance between routability and performance. Simple interconnection delay estimation models based on net length or fanout are no longer accurate enough for use within timing-driven placement algorithms. On the other hand, more accurate interconnection delay computation methods, such as the distributed *RC* model, are too computationally expensive for incorporating into simulated annealing-based placement algorithms. Therefore, special techniques for fast and sufficiently accurate interconnection delay estimation are essential for timing-driven FPGA placement. Fast interconnection delay estimation techniques have been successfully developed and used for channel-based FPGA architectures (13).

Clock Skew

Controlling clock skew is a critical issue in synchronous circuit designs, especially for high-speed system level designs. As long as other higher priority constraints are satisfied, it is always desirable to reduce clock skew to further improve circuit performance and fault-tolerant margin. FPGA architectures allow further clock skew reduction during placement. A typical FPGA device usually contains several clock networks. Clock pins on sequential elements such as flip flops are connected to the selected clock networks through programmable switches. Figure 5 illustrates connections between clock pin (CLK) and two clock networks (CLK1 and CLK2) in a row-based FPGA architecture. The clock pin CLK can be connected to either CLK1 or CLK2, depending on circuit designs. Different sets of logic modules chosen for circuit placement in an FPGA device lead to different capacitance load distributions on a clock network. Consequently, the clock skews on clock networks may vary with different placements and FPGA-specific placement algorithms can take advantage of this fact to further reduce clock skew where desired (14).

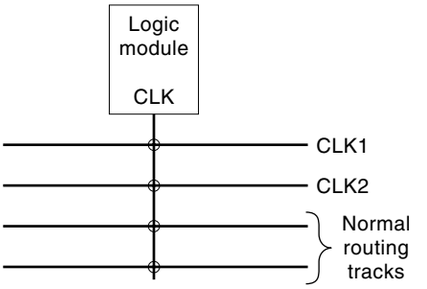

Figure 5. A clock pin on a logic module can be connected to one of two clock network tracks (CLK1, CLK2) in the adjacent routing channels. The connection is established by turning on appropriate switches, represented by the circles.

ROUTING

Because of the high complexity involved in the routing problem, FPGA routing normally is performed in two phases: *global routing* and *detailed routing*. Global routing assigns each net a routing path by selecting a set of routing channels, but does not choose specific routing tracks and switches for each net. The goal of global routing is to create a problem that can facilitate the subsequent detailed router to select routing segments. Since routability is the most important issue, minimization of channel density is normally the optimization objective in FPGA global routing. Similar to the approach for conventional ASIC technologies, FPGA global routing problems are normally formulated as minimum steiner tree problems and solved by using steiner tree minimization algorithms. However, there exist two FPGA-specific issues in global routing. The first one is run time. As mentioned in the previous placement section, since global routing is embedded within the placement optimization process, run time of the FPGA global router is more restricted than the global routers used for ASIC technologies. The second issue is routability estimation. For an FPGA architecture where channel intersection areas are not fully populated with switches, routability in the intersection areas cannot be accurately measured with channel densities. Instead, connectivity architectural details within the channel intersection areas need to be considered in order to estimate the routability more accurately (15).

The task of detailed routing is to assign each net to specific routing segments in the channels as restricted by the global router. Design of detailed routing algorithms depends heavily on FPGA routing architectures. Detailed routing algorithms for row-based and symmetrical-array-based architectures are significantly different.

Detailed Routing for Row-Based Architectures

Routing channels in row-based architectures are segmented. A routing track in the segmented channel is divided into several routing segments with various lengths by placing

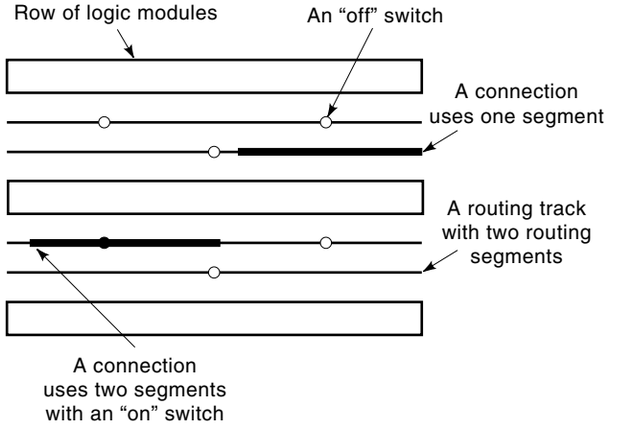

Figure 6. Routing in segmented channels. Switches in "off" and "on" states are represented by open and solid circles, respectively. Switches can be turned on to connect adjacent routing wire segments on the same track in order to route longer connections.

switches between the adjacent routing segments (Fig. 6). Routing track segmentation is designed based on the net connection distribution statistics collected from a large pool of real circuit designs to achieve a delicate balance between routability and performance. Where desirable, two adjacent routing segments on the same track can be connected by turning on the switch in between to form a longer routing segment that can be used to complete a longer net connection. Most of the vertical routing segments are attached to the logic modules and provide routing resources similar to the feed throughs found in the standard cell architecture. Intersecting vertical and horizontal routing segments are fully populated with switches so that any vertical routing segments can be connected to any intersecting horizontal routing segments as necessary. Therefore, the detailed routing problem in row-based architectures is reduced to solving segmented channel routing problems.

Because switches can introduce significant delay to interconnections due to the relatively high fuse resistance, the number of switches allowed for completing a net connection is usually restricted in order to achieve high circuit performance. In a K-segment channel routing, the maximum number of segments used for routing any net connection is limited to K. For K equal to 1, the segmented channel routing problem can be solved efficiently by using a bipartite matching technique. However, for K greater than 1, segmented channel routing becomes an NP-complete problem (16), except for several special segmentations which, unfortunately, are not used in most commercial FPGA products. Efficient and effective heuristic algorithms have been developed in the commercial tools to solve the general segmented channel routing problem.

Detailed Routing for Symmetrical-Array Architectures

The intersecting vertical and horizontal routing tracks in a symmetrical-array-based architecture usually are not fully populated with switches. Consequently, the detailed routing problem for symmetrical-array-based architectures cannot be reduced to solving individual channel routing problems. A commonly followed approach is to explore the connectivity within the routing channels specified by the global router by using a search technique, such as maze router. The search approach is practically feasible due to the coarse granularity of the architecture, where the number of tracks in each channel is less than the number of tracks found in a segmented channel in a row-based architecture. Moreover, the tracks in symmetrical-array architectures are not as finely segmented as the segmentation found in the row-based architectures. The search space therefore is significantly limited. To improve the routability with the limited search space, the competition on the critical routing segments between different nets must be considered in the routing process. The critical routing segments contended by different nets can be identified based on the number of distinct nets that may use the routing segments for routing (17).

Table 1. FPGA Logic Synthesis Tools Vendors and Their Products

CAD Tool Vendor	CAD Tool Product Name
Cadence Designs Systems San Jose, CA	FPGA Designer
Exemplar Logic Alameda, CA	Galileo Logic Explorer
Synopsys Mountain View, CA	FPGA Compiler
Synplicity Mountain View, CA	Synplify

COMMERCIAL CAD SOFTWARE

Front-end logic optimization and technology mapping algorithms normally do not have strong dependence on FPGA architecture details. A small set of basic technology mapping algorithms can be used to support different FPGA products from different FPGA vendors. Consequently, front-end software tools used in FPGA designs are normally from independent CAD software vendors, instead of from FPGA companies. Table 1 lists several major CAD software companies that develop and market FPGA synthesis tools that can support various FPGA architectures (18). In addition to commercial tools, a number of FPGA logic synthesis tools developed at universities are in the public domain. Table 2 lists several such logic synthesis tools.

Unlike synthesis and technology mapping algorithms, FPGA place and route algorithms are strongly tied to the architecture details of the individual FPGA product. Interactive evaluation between the physical design algorithms and architecture details during a new FPGA product development is

Table 2. FPGA Logic Synthesis Tools Developed at Universities

Institute	CAD Tool Name
UC Berkeley	MIS-pga
UCLA	FlowMAP/RASP
University of Toronto	Chortle

critical to the success of product development. Currently, most FPGA vendors develop physical design software tools in-house, and provide proprietary place and route tools together with silicon products to their customers.

FUTURE TRENDS IN FPGA CAD RESEARCH AND DEVELOPMENT

The goal of CAD tools is to help circuit designers use FPGA devices efficiently and effectively, and to help FPGA device architects design new FPGA architectures. Research and development of FPGA CAD tools therefore must be driven by the needs of FPGA users and designers. In this section we discuss several areas that are important for future FPGA CAD tool development.

Run Time Reduction

Currently, the capacity of an FPGA device can far exceed 100K gates and is rapidly increasing. As FPGA devices become larger, the run time of CAD tools for completing an FPGA circuit design is getting longer, especially in the physical design stage. Making matters worse, the increase in run time of current CAD tools is greater than the increase in silicon gate capacity. It is no longer unusual to take more than a day to complete a design of a 100K FPGA device with current CAD tools. If the run time of CAD tools continues to increase at a faster rate than the increase in silicon capacity, the competitive advantage of fast turnaround provided by FPGAs will diminish. In order to maintain the fast turnaround advantage, it is necessary to reduce CAD tool run time, especially in the physical design stage.

Support of Different FPGA Architectures

The demand for flexible CAD tools that are able to support different FPGA architectures is driven by two issues. The first is that new FPGA architectures continue to emerge to accommodate the requirements of new applications and technologies, and designing new FPGA architectures requires CAD tool support for architectural evaluation. The second issue is that developing new CAD tools is a time-consuming and hard to predict process, and very often this process is the bottleneck in the new FPGA product development.

In order to address these issues, CAD tools should consist of a number of modular, independent point tools that can be easily modified and integrated to form a complete design flow to support new FPGA architecture development. The flexibility of integration of point tools is supported by carefully designed device and netlist databases that are used to transfer data between individual point tools. Each of the point tools must be able to support common features in different FPGA architectures and be flexible enough to support new architectural features.

Innovative Algorithms

Innovative algorithms are always in demand as FPGA architectures continue to evolve. For example, a new trend in FPGA architecture design is to integrate specialized functional modules implemented in ASIC together with FPGA in a single device. It is also becoming common to provide embedded memory arrays, especially on large capacity FPGA devices, in order to support system-level designs that require both logic and memory. New CAD algorithms for logic synthesis and physical design may need to be developed in order to effectively integrate different functionalities on a single FPGA device.

Another example where new algorithms are desirable is in hierarchical architectures. As FPGA capacity continues to increase, hierarchical FPGA architectures are more efficient compared with flattened architectures for achieving an appropriate balance between area, performance, and routability. Algorithms such as partitioning and clustering that were previously developed within other contexts will need to be modified in order to accommodate the special requirements of hierarchical FPGA architectures.

BIBLIOGRAPHY

1. S. D. Brown et al., *Field-Programmable Gate Arrays,* the Netherlands, Kluwer Academic Publishers, 1992.
2. S. M. Trimberger, (ed.), *Field-Programmable Gate Array Technology,* the Netherlands, Kluwer Academic Publishers, 1994.
3. A. El Gamal (ed.), Special section on field-programmable gate arrays, *Proc. IEEE,* **81** (7): 1993.
4. R. Murgai, R. Brayton, and A. Sangiovanni-Vincentelli, *Logic Synthesis for Field-Programmable Gate Arrays,* New York: Kluwer Academic Publishers, 1995.
5. J. Cong and Y. Ding, Combinational logic synthesis for LUT based field programmable gate arrays, *ACM Trans. Des. Autom. Electron. Sys.,* **1** (2): 145–204, 1996.
6. R. Francis, J. Rose, and Z. G. Vranesic, Chortle-crf: Fast technology mapping for lookup table-based FPGAs, *Proc. 28th Des. Autom. Conf.,* San Francisco, CA, pp. 227–233, 1991.
7. J. Cong and Y. Ding, An optimal technology mapping algorithm for delay optimization in look-up-table based FPGA designs. *Proc. IEEE Int. Conf. Comput.-Aided Des.,* pp. 48–53, 1992.
8. B. Luca and G. De Micheli, A survey of Boolean matching techniques for library binding, *ACM Trans. Des. Autom. Electron. Syst.* **2** (3): 1996.
9. R. E. Bryant, Graph-based algorithms for Boolean function manipulation, *IEEE Trans. Comput.,* **C-35**: 677–691, 1986.
10. A. Bedarida, S. Ercolani, and G. De Micheli, A new technology mapping algorithm for the design and evaluation of fuse/antifuse-based field-programmable gate arrays, *1st Int. ACM / SIGDA Workshop FPGAs,* pp. 103–108, 1992.
11. K. Zhu and D. F. Wong, Fast Boolean matching for field-programmable gate arrays, *Proc. Eur. Des. Autom. Conf.,* pp. 352–357, 1993.
12. C. M. Fiduccia and R. M. Mattheyses, A linear-time heuristic for improving network partitions, *Proc. ACM / IEEE Des. Autom. Conf.,* pp. 175–181, 1982.
13. M. Chew and J. C. Lien, Fast delay estimation in segmented channel FPGAs, *2nd Int. ACM / SIGDA Workshop Field-Programmable Gate Arrays,* Section 8, 1994.
14. K. Zhu and D. F. Wong, Clock skew minimization during FPGA placement, *IEEE Trans. Comput.-Aided. Des. Integr. Circuits Syst.,* **CAD-16**: 376–385, 1997.
15. Y.-W. Chang et al., A new global routing algorithm for FPGAs, *Proc. IEEE / ACM Int. Conf. Comput.-Aided Des.,* San Jose, CA, pp. 380–385, 1994.
16. J. Greene et al., Segmented channel routing, *Proc. 27th ACM / IEEE Des. Autom. Conf.,* pp. 567–572, 1990.

17. S. Brown, J. Ross, and Z. G. Vranesic, A detailed router for field-programmable gate arrays, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* **CAD-11**: 620–627, 1992.

18. S. Schulz, Logic synthesis and silicon compilation tools, *Integr. Syst. Des.,* 1996.

KAI ZHU
Actel Corporation

D. F. WONG
University of Texas at Austin