

How Much Logic Should Go in an FPGA Logic Block?

VAUGHN BETZ
JONATHAN ROSE
University of Toronto

The logic blocks of most FPGAs contain clusters of lookup tables and flip-flops, yet little is known about good choices for key parameters: How many lookup tables should a cluster contain, how should FPGA routing flexibility change as cluster size changes, and how many inputs should programmable routing provide each cluster?

ALL FIELD-PROGRAMMABLE GATE arrays contain both programmable logic blocks and programmable routing. The nature of the logic block strongly influences an FPGA's speed and density. FPGAs are approximately 10 times less dense and three times slower than mask-programmed gate arrays. Thus, we have strong motivation to explore new logic blocks that help close this gap—for example, logic blocks composed of groups, or clusters, of lookup tables and flip-flops.

Most SRAM-based FPGAs use logic blocks based on lookup tables (LUTs). A LUT-based logic block can implement any function of its inputs. Accordingly, we normally describe lookup tables by their number of inputs. A lookup table with more inputs can implement more logic, and hence we need fewer logic blocks to implement a circuit. Using fewer blocks saves routing area by reducing connections between logic blocks. However, LUT complexity grows exponentially with the number of inputs, so using a lookup table with a large number of inputs as a logic block is impractical.

Instead of creating a larger logic block by increasing the number of inputs, we can simply group several LUTs together and interconnect them with local routing. The resulting block is a logic cluster.¹ Figure 1 shows a circuit implemented in an FPGA in which each logic cluster contains two 4-

input lookup tables. Notice that many connections can be made via the local interconnect within a cluster. Because this local interconnect can be faster than the general-purpose routing between logic blocks, cluster-based logic blocks can improve FPGA speeds. Moreover, an FPGA in which every cluster contains several LUTs needs fewer logic blocks to implement a circuit than an FPGA in which each logic block is a single LUT. Thus, clusters reduce the size of the placement and routing problem considerably. Since placement and routing is usually the most time-consuming step in mapping a design to an FPGA, clusters can significantly reduce design compilation time. As FPGAs grow larger, it is important to keep compilation time from growing too large, or key FPGA advantages—rapid prototyping and design—will be lost.

A more complex issue, the area impact of grouping multiple LUTs into a logic cluster, is our focus here. On the one hand, grouping related LUTs into a single logic block reduces the number of connections between logic blocks, saving routing area. Since general-purpose interconnect consumes most of the die area in SRAM-based FPGAs, this is a significant area savings. On the other hand, in the logic clusters we study, the area required by local routing within the clusters grows quadratically with their size. For suf-

ficiently large clusters, then, the area used by local interconnect will exceed the area saved in general interconnect.

Here, we explore three questions concerning area-efficient design of cluster-based logic blocks. First, how many distinct inputs should the FPGA routing provide to a cluster of LUTs? Reducing the number of inputs to a logic block saves routing area. If the number is too low, however, many circuits will be unable to use all the LUTs in a cluster, and area will be wasted. Second, as the number of LUTs in a logic cluster changes, how should the FPGA's routing architecture change? Finally, how many LUTs should we include in a cluster? Recent FPGAs from Xilinx, Altera, Lucent Technologies, and Actel have grouped several LUTs into logic clusters, but little work investigating these questions has been published. Although Aggarwal and Lewis² investigated a strictly hierarchical FPGA, to our knowledge the work presented here is the first to investigate the use of logic blocks with two levels of hierarchy within an otherwise flat FPGA architecture.

Cluster-based logic blocks

Previous research³ has shown that a four-input lookup table is the most area-efficient LUT. Since most commercial FPGAs use LUTs of this size, all the logic clusters we study here are groups of 4-input LUTs.

Although a 4-input LUT enables FPGAs to perform combinational functions, we cannot implement sequential circuits unless our logic blocks also contain flip-flops. Figure 2 shows how most commercial FPGAs combine a LUT and a flip-flop to create a logic block that can perform both combinational and sequential functions. We call the structure in Figure 2 a basic logic element, or BLE.

A complete logic block, or logic cluster, consists of several BLEs, plus the local routing required to interconnect them, as shown in Figure 3. The logic cluster has two parameters: N , the number of BLEs, and I , the number of inputs. As the figure shows, not all the LUT inputs (there are $4 \times N$) are accessible from outside the cluster. Instead, the FPGA provides only I external inputs to the logic cluster—multiplexers within the cluster allow arbitrary connections of these inputs to the BLE inputs. The same multiplexers also connect to each BLE output, allowing the output of any BLE within

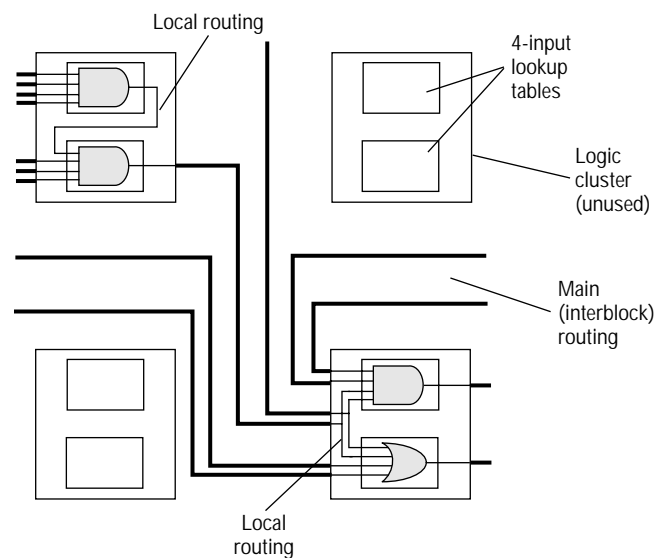


Figure 1. Circuit implementation in an FPGA with a cluster size of two lookup tables.

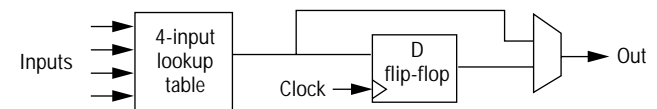


Figure 2. Basic logic element (BLE).

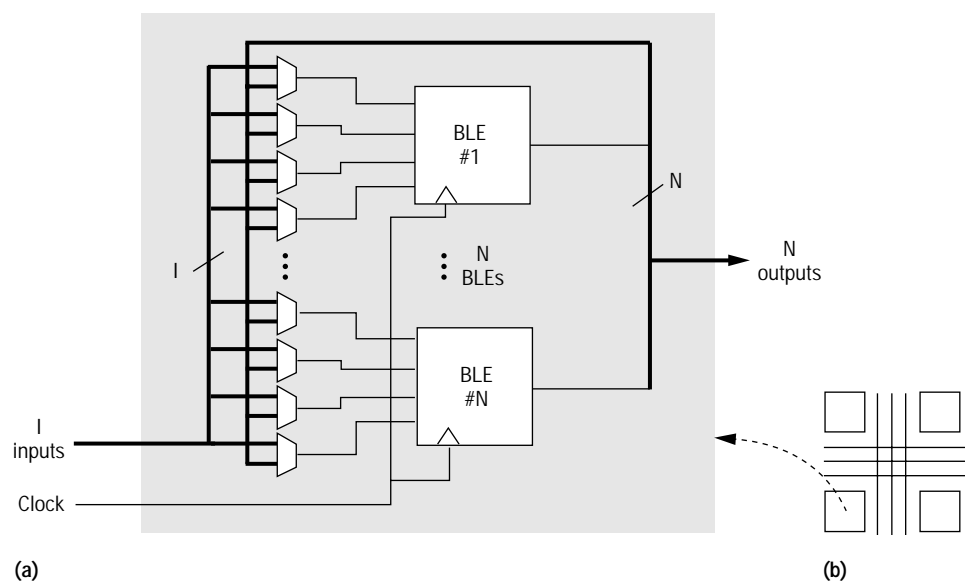


Figure 3. Logic cluster structure (a): FPGA (b).

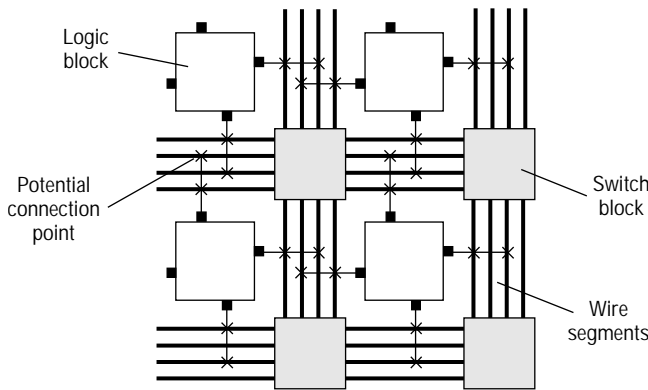


Figure 4. An island-style FPGA.

the cluster to connect to any BLE input. All N outputs of the logic cluster can also connect to the main FPGA routing for use by other logic clusters.

Because each BLE input can connect to any cluster input or any BLE output, we call these logic clusters fully connected. For example, the logic block clusters in the Altera 8000 and 10000 FPGAs are fully connected, and the cluster in the Xilinx 5200 FPGA is almost fully connected.

It is simpler to write CAD tools for fully connected logic clusters than for clusters with less flexible local interconnect. Determining if a group of BLEs can be implemented in a single cluster is simple: If the BLEs need I or fewer distinct inputs, they can all go into one cluster. Also, in a fully connected logic cluster, all cluster inputs and outputs are logically equivalent. That is, all the inputs are functionally identical, and all the outputs are functionally identical. This means that a net that is input to a cluster can connect to any cluster input, and a net driven by a cluster output can connect to any cluster output. Therefore, the router has a great deal of flexibility in routing intercluster nets.

Experimental methodology

Our goal was to determine the cluster parameters that lead to the most area-efficient FPGA architecture. There are no detailed analytic models of FPGA architectures and circuitry, so we must evaluate architectures experimentally.

In our experiments, we implemented a set of 20 benchmark circuits into each FPGA architecture of interest and measured how much area the circuits require in each architecture. For each circuit, we used an automatic CAD flow similar to that used by typical FPGA users: technology mapping, placement, and routing. We took considerable care to use high-quality CAD tools that fully exploited each architecture; low-quality tools or tools that favor particular architectures can lead to inaccurate conclusions. The benchmark circuits are 20 of the largest MCNC (Microelectronics Center of North Carolina) circuits, ranging in size from 500 to 3,690

BLEs.⁴ These circuit sizes are typical of the designs implemented in current commercial FPGAs.

FPGA architecture assumptions. A logic block's area efficiency depends not only on the number of transistors required to implement the block itself but also on the number of transistors required to route the connections between blocks. Consequently, to determine FPGA area efficiency, we must choose an FPGA routing architecture as well as a logic block architecture. All of our experiments assumed an island-style FPGA; both Xilinx and Lucent Technologies FPGAs employ this type of architecture. As shown in Figure 4, an island-style FPGA consists of an array of logic blocks surrounded by channels of wire segments. Input and output pins are distributed around each logic block's perimeter, and programmable switches connect these pins to wire segments in the adjacent routing channels. At every routing channel intersection, there is a switch block,⁵ a set of programmable switches that allows wiring segments to connect to form longer connections.

To be as realistic as possible, we set F_s , the number of wiring segments to which each segment can connect at a switch block,⁵ to 3, the F_s value in most commercial FPGAs. We defined two other important architectural parameters: W is the number of wiring segments in each routing channel—that is, the channel capacity. F_c is the number of wiring segments to which a logic block input or output pin can connect in an adjacent channel.⁵ In Figure 4, for example, W is 4 and F_c is 2.

CAD flow. Figure 5 illustrates the CAD flow we used in these experiments. First, we performed technology-independent logic optimization of each circuit using the SIS synthesis package,⁶ which attempts to simplify the logic and remove redundant circuitry. Next, we used the FlowMap algorithm⁷ to map the technology of each circuit into a netlist of 4-input LUTs and flip-flops. FlowMap takes a circuit description in terms of basic gates and implements it using only 4-input LUTs and flip-flops, the only logic resources available in the FPGAs we studied. Our VPack program¹ then mapped this netlist into logic clusters with the specified values of N and I . Thus, at this point, we had described the circuit as a set of interconnected logic blocks of the exact type that exist in the FPGA we were targeting.

Finally, we used our VPR tool⁸ to place and route the circuit. Placement consists of choosing a position for each logic block that minimizes the length of the wires needed to interconnect the circuitry. Routing consists of choosing the wires that will make each connection. As Figure 5 shows, VPR repeatedly routes the circuit with different channel capacities until it finds the minimum number of wire segments per channel needed to successfully route the circuit. At this

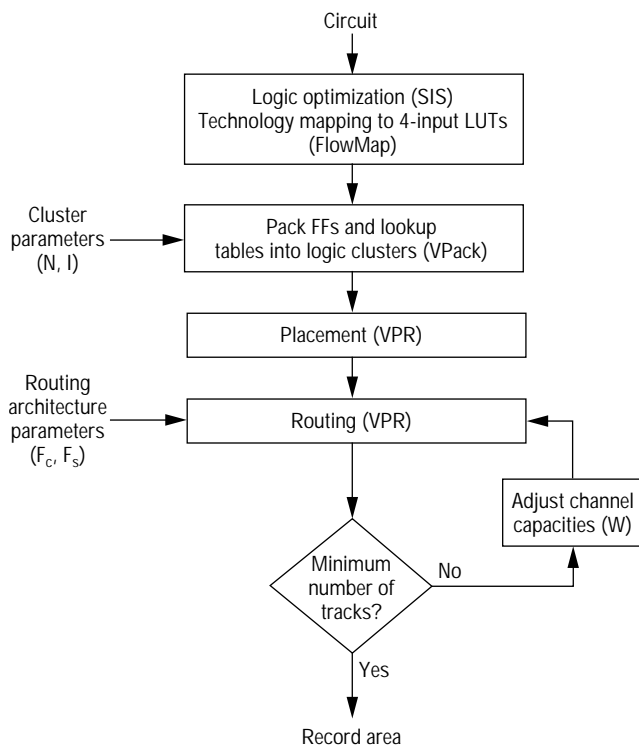


Figure 5. Architecture evaluation CAD flow.

point, we had enough information to use our area model to evaluate the architecture's area efficiency.

Area model. We based the area model on the number of minimum-width transistors required to implement a benchmark circuit in each FPGA architecture (counting larger transistors as several minimum-width transistors). To allow averaging of results from different-size circuits, we used a normalized area metric: number of transistors used per BLE in a circuit. We have developed a detailed model of the number of transistors required to implement both logic clusters and FPGA routing in an SRAM-based FPGA. This model tries to build an FPGA with as few transistors as possible without unduly compromising speed.

Experimental results

From our experiments, we obtained answers to the three questions we asked at the outset—namely, what values of I , F_c , and N lead to the most area-efficient FPGA architectures?

Cluster inputs vs. cluster size. Our first question was how many distinct inputs, I , the FPGA routing should provide to a cluster of size N . Since the number of transistors required to implement each multiplexer shown in Figure 3 grows linearly with I (for large I), we would like to make I as

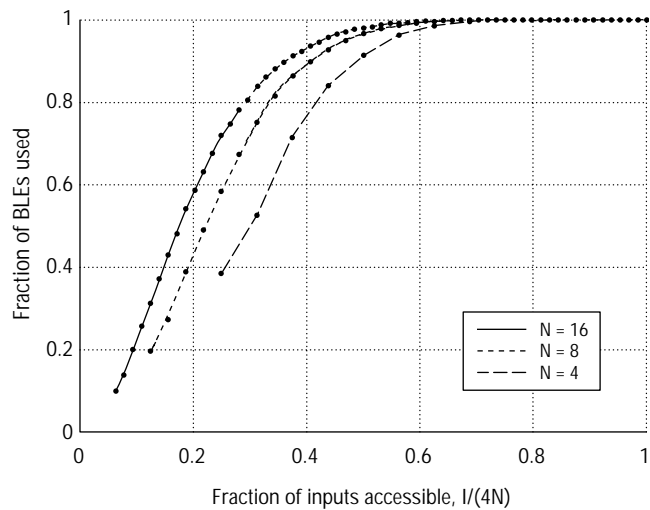


Figure 6. Logic utilization versus number of logic cluster inputs (20-benchmark average).

small as possible. On the other hand, if I is too small, many of the BLEs in a logic cluster may become essentially unusable, reducing logic utilization and wasting area. To find the minimum value of I that allows good cluster utilization, we ran benchmark circuits through the first two steps in Figure 5, technology mapping and cluster packing. Then we measured the resulting logic utilization for different values of I . We define logic utilization as the average number of BLEs per cluster that a circuit can use, divided by N .

Figure 6 shows how the average logic utilization of our 20 benchmarks varies with I for three different logic cluster sizes. The horizontal axis shows the number of distinct inputs to the cluster relative to the total number of BLE inputs in a cluster— $I/(4N)$. For very low values of I , logic utilization is very low, as one would expect. It is interesting, however, that when I is only 50% to 60% of the total number of BLE inputs, logic utilization is essentially 100%. Clearly, it is possible to pack BLEs together so that they have many common inputs and can reuse locally generated outputs. The relative amount of input sharing and output reuse increases slightly with logic cluster size, causing the curves in Figure 6 to shift to the left as cluster size increases.

In Figure 7 (next page), the solid line shows the value of I required to achieve 98% logic utilization as cluster size varies. The dashed line shows how the average number of logic cluster inputs actually used varies with cluster size. Although there are $4N$ BLE inputs in a logic cluster of size N , the number of inputs required to achieve 98% logic utilization is only about $2N + 2$. Furthermore, the average number of logic cluster inputs actually used grows even more slowly. On average, a cluster of size 1 uses 3.5 of its inputs, while a cluster of size 16 uses only 19.7 of its inputs. That is, while

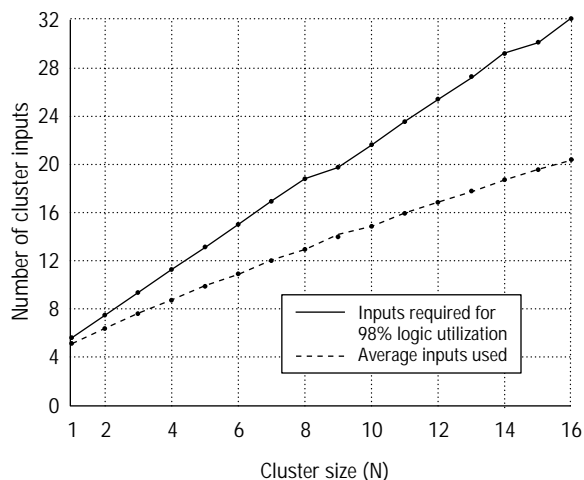


Figure 7. Variation in inputs required and inputs used with cluster size (20-benchmark average).

logic per cluster increases by a factor of 16, the average number of connections that must be routed to each cluster increases by a factor of only 5.6.

Our results show that commercial FPGAs can be more aggressive in reducing the value of I . For example, Altera Flex 8000 FPGAs use logic clusters with $N = 8$ and $I = 24$; our results indicate that $I = 18$ suffices for a cluster of this size. Similarly, the Xilinx 5200 FPGA uses a logic cluster with $N = 4$ and makes all 16 LUT inputs accessible, but our results suggest 10 inputs are sufficient. Reducing I in this manner simplifies the cluster input multiplexers and reduces the number of logic block pins that must be connected to the FPGA routing, resulting in considerable area savings.

Routing flexibility vs. cluster size. Before we can apply the experimental CAD flow to see how area efficiency varies with cluster size, we must choose F_c , the number of routing tracks to which each logic block pin can connect. On the one hand, using a smaller value of F_c reduces the number of programmable switches in the FPGA routing, which improves area efficiency. On the other hand, smaller values of F_c make an FPGA less routable so that larger channel capacities (W) will be required to successfully route circuits. This reduces area efficiency by increasing the routing area. The goal is to choose a value of F_c that balances these competing objectives and achieves good area efficiency.

For a cluster of size 1, a good value of F_c is W ; in other words, each logic block pin can connect to any routing track in an adjacent channel. For larger clusters, however, setting F_c to W provides far more routing flexibility than necessary, wasting area. Recall that full connectivity of a logic cluster means that a net that must connect to a logic block input can connect to any of the I inputs. Similarly, a net that must

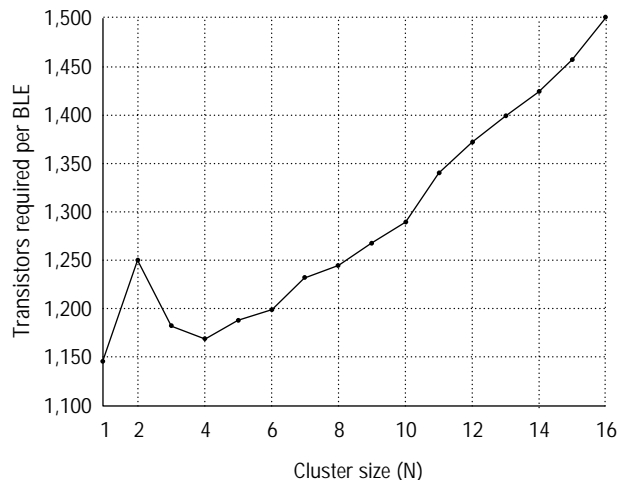


Figure 8. Area efficiency versus cluster size (20-benchmark average).

connect to a logic block output can connect to any of the N outputs. As N increases, keeping F_c fixed at W provides an excessive number of ways to connect to each logic block. For example, a cluster of size 1 has four inputs and one output. If $F_c = W$, there are $4W$ ways to connect to a cluster input and W ways to connect to the cluster output. A cluster of size 16, on the other hand, has 32 inputs and 16 outputs, so there are $32W$ ways to connect to a cluster input and $16W$ ways to connect to a cluster output if $F_c = W$.


We found that a more appropriate level of routing flexibility results when F_c is set to W/N , and all the experiments in the next section used this value. This choice of F_c means that each of the W routing tracks can be driven by one output pin on each logic block. This ensures that the FPGA can readily use all routing tracks in a channel to interconnect blocks.

Area efficiency vs. cluster size. We are now in a position to examine which cluster size leads to the most area-efficient FPGA. In these experiments, we chose the number of inputs to a cluster of size N to be the minimum value that allows VPack to achieve 98% logic utilization. This value of I allows essentially full utilization of our logic clusters. At the same time, it minimizes the complexity of the cluster input multiplexers and the number of logic block pins to be connected to the main FPGA routing. We ran the 20 benchmark circuits through the experimental flow described earlier and determined the area they required after placement and routing in each architecture.

Figure 8 shows how area efficiency varies with cluster size. Notice that all clusters with sizes between 1 and 8 have area efficiencies within a 10% range. Clearly, with proper choices of I and F_c , any cluster in this range provides reasonable area efficiency, except perhaps a cluster of size 2.

As we increase cluster size from 1 to 2, area efficiency worsens: A cluster of size 1 requires no local routing (it is a single BLE), whereas a cluster of size 2 does. The addition of this local routing to the FPGA requires a considerable number of transistors, and at a cluster size of 2, the number of connections between clusters has not decreased enough to compensate. Further increases of cluster size, to 3 and 4, improve area efficiency because now the local routing reduces the amount of routing required between logic blocks more significantly. As cluster size rises past 10, area efficiency rapidly degrades. The complexity of the local routing grows quadratically with cluster size, and for sufficiently large clusters, swamps area improvements gained by reducing the routing required between logic blocks.

WE DRAW THREE MAIN CONCLUSIONS from our work. First, the number of distinct inputs required by a logic cluster grows fairly slowly with cluster size. A cluster of size N requires approximately $2N + 2$ distinct inputs (for $N \leq 16$). Second, because all input and output pins of a cluster are logically equivalent, we can significantly reduce the number of routing tracks to which each pin can connect as we increase cluster size. Finally, the area efficiency of logic blocks containing between one and eight BLEs is within a 10% range, so any logic block in this range is a reasonable choice.

Cluster-based logic blocks have two significant advantages over single BLE logic blocks: Larger clusters reduce the size of the placement problem and tend to increase FPGA speed. Since a cluster-based logic block with appropriate values of N , I , and F_c has an area efficiency comparable to that of a single BLE logic block, an FPGA can gain these advantages without any area penalty. 

Acknowledgments

The Information Technology Research Centre of Ontario, the Natural Sciences and Engineering Research Council of Canada, and the Walter C. Sumner Foundation supported the work described in this article.

References

1. V. Betz and J. Rose, "Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size," *Proc. IEEE Custom Integrated Circuits Conf.*, IEEE Computer Society Press, Los Alamitos, Calif., 1997, pp. 551-554.
2. A. Aggarwal and D. Lewis, "Routing Architectures for Hierarchical Field Programmable Gate Arrays," *Proc. Int'l Conf. Computer Design*, IEEE CS Press, 1994, pp. 475-478.
3. J. Rose et al., "Architecture of Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," *IEEE J. Solid State Circuits*, Oct. 1990, pp. 1217-1225.
4. S. Yang, *Logic Synthesis and Optimization Benchmarks, Version 3.0*, tech. report, Microelectronics Center of North Carolina, Research Triangle Park, N.C., 1991.
5. S. Brown et al., *Field-Programmable Gate Arrays*, Kluwer Academic, Norwell, Mass., 1992.
6. E.M. Sentovich et al., *SIS: A System for Sequential Circuit Analysis*, Tech. Report No. UCB/ERL M92/41, Univ. of California, Berkeley, 1992.
7. J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *IEEE Trans. CAD*, Jan. 1994, pp. 1-12.
8. V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," *Proc. Int'l Workshop Field-Programmable Logic and Applications*, 1997, Springer-Verlag, Berlin, pp. 213-222.



Vaughn Betz is working toward his PhD in electrical engineering at the University of Toronto. His research interests are FPGA architecture and CAD. Betz received his BSEE degree from the University of Manitoba and his MSEE degree from the University of Illinois at Urbana-Champaign. He is an IEEE member.



Jonathan Rose is an associate professor of electrical and computer engineering at the University of Toronto. Previously, he worked on a next-generation FPGA architecture as a senior research scientist at Xilinx. Still earlier, he was a research associate in the Computer Systems Laboratory at Stanford University. He is a cofounder of the ACM FPGA Symposium, and his research covers all aspects of FPGAs and field-programmable systems. Rose received the PhD in electrical engineering from the University of Toronto. He is an IEEE member.

Send questions or comments about this article to the authors at University of Toronto, Dept. of Electrical and Computer Engineering, 10 King's College Rd., Toronto, ON, Canada M5S 3G4; [vaughn, jayar]@eecg.toronto.edu.