

# How Much Logic Should Go in an FPGA Logic Block?

Vaughn Betz and Jonathan Rose  
Department of Electrical and Computer Engineering, University of Toronto  
Toronto, Ontario, Canada M5S 3G4  
{vaughn, jayar}@eecg.utoronto.ca  
(416) 978-1653

## Abstract

The logic blocks of most modern FPGAs contain clusters of look-up tables and flip flops, yet little is known about good choices for several key architectural parameters related to these clusters. There are three basic questions: how many look-up tables should a cluster contain, how should the flexibility of FPGA routing change as the cluster size changes, and how many inputs should the programmable routing provide to each cluster? We first show that logic clusters require fewer inputs from the routing than current commercial FPGAs provide. Secondly, we show that for best area-efficiency the flexibility of FPGA routing should be significantly reduced as the cluster size is increased. Finally, we find that clusters containing between 1 and 8 look-up tables all provide reasonable area-efficiency, as long as the number of cluster inputs and the FPGA routing flexibility are chosen appropriately.

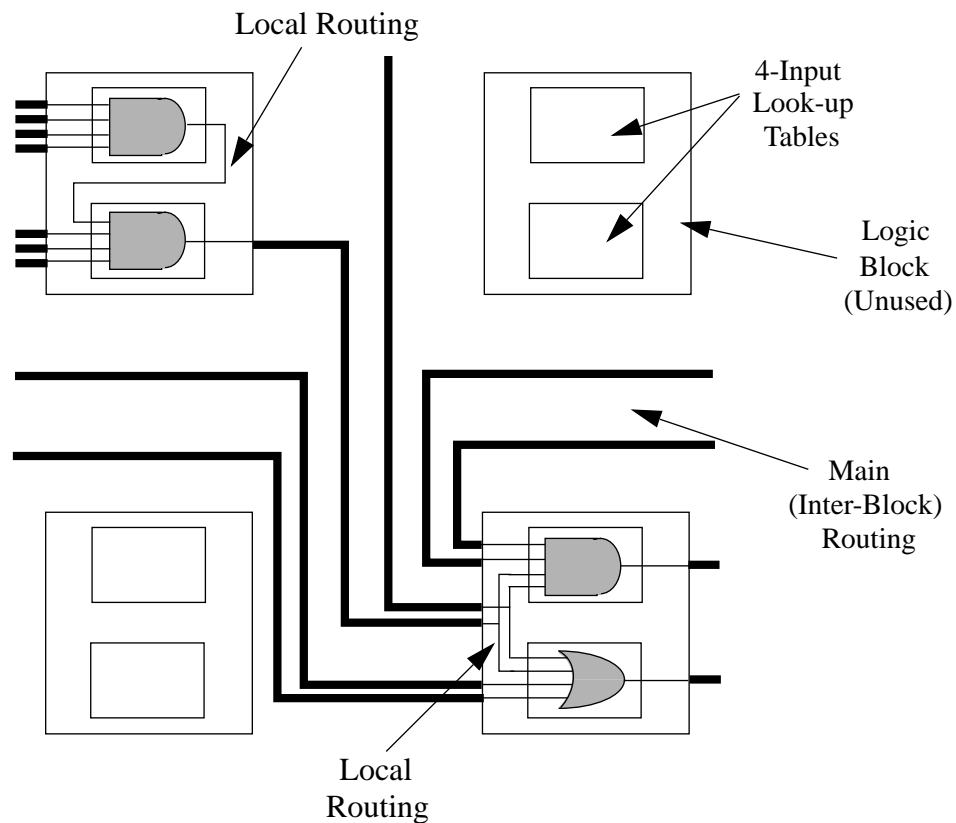
## 1 Introduction

All Field-Programmable Gate Arrays (FPGAs) contain both programmable logic blocks and programmable routing. The logic block employed strongly influences both the speed and density of an FPGA. Since FPGAs are approximately 10 times less dense and three times slower than mask-programmed gate arrays, we are motivated to explore new logic blocks which help close this density and speed gap. This paper investigates logic blocks composed of groups, or clusters, of look-up tables and flip flops.

Most SRAM-based FPGAs use logic blocks based on look-up tables (LUTs). A look-up table is a logic block that can implement *any* function of its inputs; accordingly, look-up tables are normally described by their number of inputs. A look-up table with more inputs can implement more logic, and hence one needs fewer logic blocks to implement a circuit. This saves routing area, as there are fewer connections to route between logic blocks. However, look-up table complexity

grows exponentially with the number of inputs, so it is impractical to use a LUT with a large number of inputs as a logic block.

Instead of creating a larger logic block by increasing the number of inputs to a LUT, we can simply group several LUTs together and provide local routing to interconnect them. We call the resulting logic block a *logic cluster* [1]. Figure 1 shows a circuit implemented in an FPGA in which each logic cluster contains two four-input look-up tables. Notice that many connections can be made via the local interconnect within a cluster. As this local interconnect can be made faster than the general-purpose routing between logic blocks, cluster-based logic blocks can improve FPGA speeds. As well, an FPGA in which every logic block contains several LUTs will need fewer logic blocks to implement a circuit than an FPGA in which each logic block is a single LUT. This reduces 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, cluster-based logic blocks can significantly reduce design compile time. As FPGAs grow larger, it is important to keep this compile time from growing too large or one of the key advantages of



**Figure 1:** Implementation of a circuit in an FPGA with a cluster size of 2.

FPGAs, rapid prototyping and design spins, will be lost.

The area impact of grouping multiple LUTs into a logic cluster is more complex, and is the focus of this paper. On the one hand, grouping related LUTs together into a single logic block reduces the number of connections to be routed between logic blocks, which saves routing area. Since the 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 the local routing within a cluster grows quadratically with cluster size. For sufficiently large clusters, then, the area used by this local interconnect will exceed the area saved in the general interconnect.

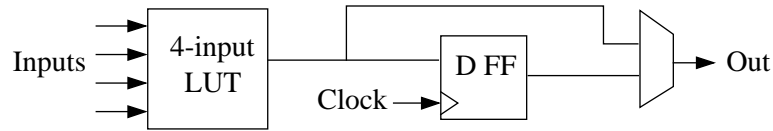
We explore three questions concerning the design of cluster-based logic blocks. First, how many distinct inputs should the FPGA routing provide to a cluster of LUTs? Reducing the number of inputs to a logic block saves routing area, but if the number of inputs is too low many circuits will be unable to use all the LUTs in a logic cluster, wasting area. Secondly, how should the routing architecture of an FPGA be altered as the number of LUTs in a logic cluster changes? Finally, how many LUTs should be included in a cluster to create the most area-efficient logic block? Recent FPGAs from Xilinx, Altera, Lucent Technologies and Actel have all grouped several LUTs together into logic clusters, but there has been little published work investigating any of these three questions.

## 2 Cluster-Based Logic Blocks

The logic blocks of most SRAM-based FPGAs are based on look-up tables. Previous research [2] has shown that a 4-input look-up table (4-LUT) is the most area-efficient LUT, and most commercial FPGAs use LUTs of this size, so all the logic clusters we study are groups of 4-LUTs.

While a 4-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.

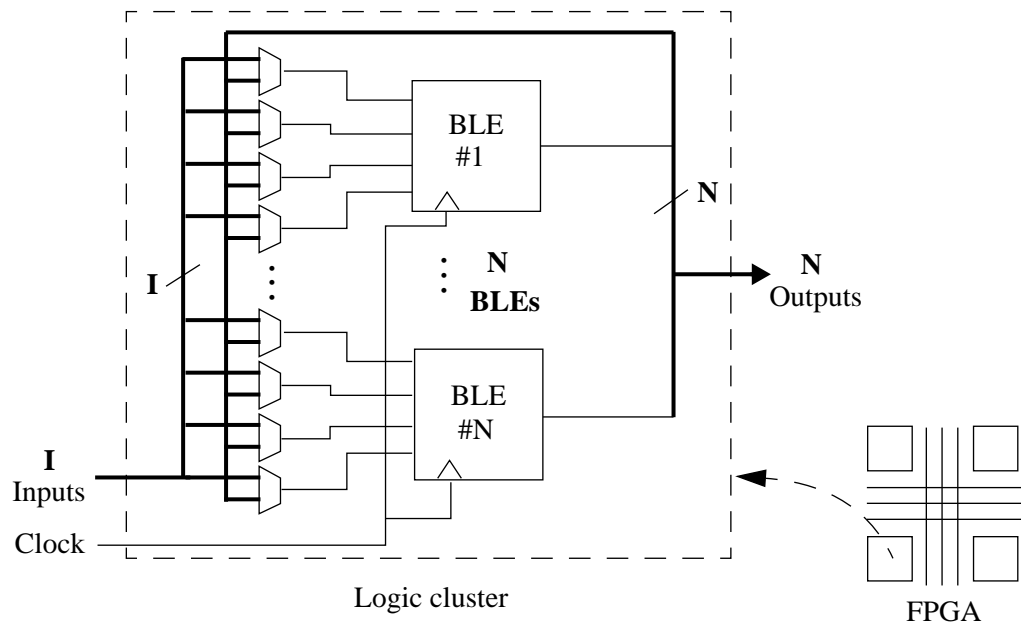
A complete logic block consists of several BLEs, plus the local routing required to interconnect them, as shown in Figure 3. We call the entire logic block a *logic cluster*. We describe a logic cluster with two parameters,  $N$  and  $I$ .  $N$  is the number of BLEs per cluster, while  $I$  is the number



**Figure 2:** Basic Logic Element (BLE)

of inputs to the cluster. As Figure 3 shows, not all of the LUT inputs (of which there are  $4 \times N$ ) are accessible from outside the logic cluster. Instead, only  $I$  external inputs are provided to the logic cluster -- multiplexers within the logic block allow arbitrary connections of these cluster inputs to the BLE inputs. The same multiplexers also connect to each of the BLE outputs, allowing the output of any BLE within the cluster to be connected to any of the BLE inputs. All  $N$  outputs of the logic cluster can also be connected to the main FPGA routing for use by other logic clusters.

Notice that each of the BLE inputs can be connected to any of the cluster inputs or any of the BLE outputs. We therefore call these logic clusters *fully connected*. It is simpler to write CAD tools for fully-connected logic clusters than it is to write tools for clusters with less flexible local interconnect. For example, determining if a group of BLEs can be implemented in a single cluster is simple -- if the BLEs need no more distinct inputs than the number of cluster inputs ( $I$ ), they can all go in one cluster. As well, in a fully-connected logic cluster all the cluster inputs and all the cluster outputs are *logically-equivalent*. That is, all of the inputs are functionally identical, and all



**Figure 3:** Logic cluster structure.

of the outputs are functionally identical. This means that a net which is an input to a cluster can be connected to *any* of the cluster inputs, and a net which is driven by a cluster output can be connected to *any* of the cluster outputs. Therefore the router has a great deal of flexibility in how it routes inter-cluster nets. The logic block cluster used in the Altera 8K and 10K FPGAs is fully connected, and the logic block cluster used in the Xilinx 5200 FPGA is nearly fully connected.

While a strictly hierarchical FPGA was investigated in [3], to our knowledge this is the first work to investigate the use of logic blocks with a two-level hierarchy within an otherwise flat FPGA architecture.

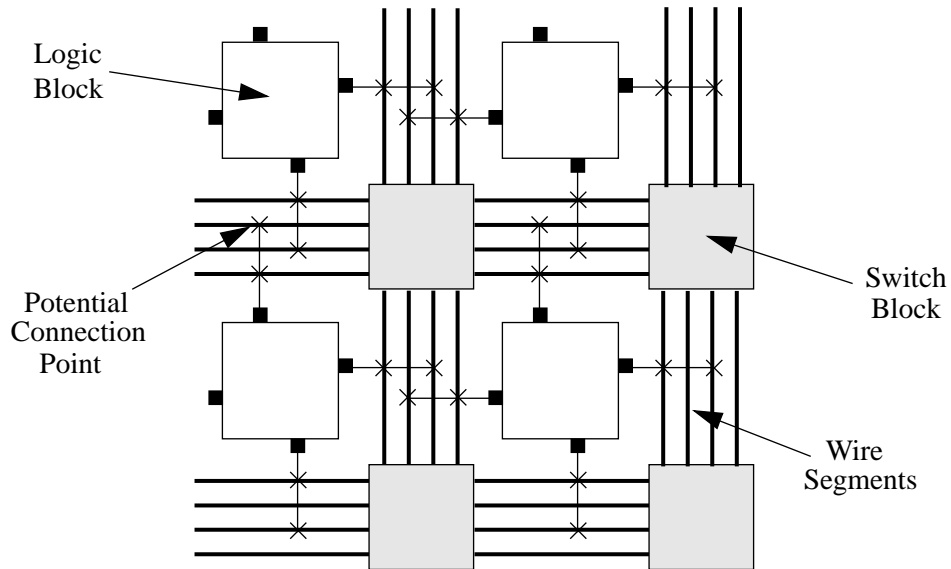
### **3 Experimental Methodology**

Our goal is 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.

We implement a set of twenty benchmark circuits into each FPGA architecture of interest, and measure how much area the circuits require in each architecture. We implement each circuit using an automatic CAD flow similar to that used by typical FPGA users: technology-mapping, placement and routing. We took considerable care to ensure that the CAD tools were of high quality and fully exploited each FPGA architecture, as low-quality tools, or tools that favor some architectures, can lead to inaccurate conclusions. The benchmark circuits used are 20 of the largest MCNC circuits [4]; they range in size from 500 to 3690 BLEs. These circuit sizes are typical of the designs being implemented in current commercial FPGAs.

#### **3.1 FPGA Architecture Assumptions**

The area-efficiency of a logic block depends not only on the number of transistors required to implement the logic block itself, but also on the number of transistors required to route all the connections between blocks. Consequently, we must choose an FPGA routing architecture, as well as a logic block architecture, in order to determine the area-efficiency of an FPGA. All the experiments in this work assume 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. The input and output pins of each logic block are distributed around its perimeter, and programmable switches are used to connect these pins to wire segments in the adjacent routing channels. At every intersection of routing



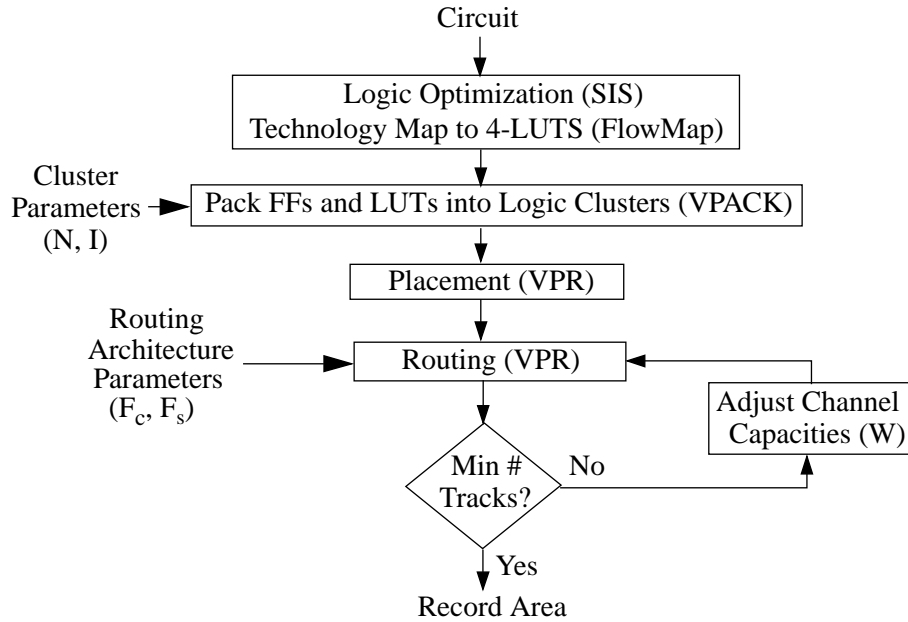
**Figure 4:** An island-style FPGA.

channels, there is a switch block [5], which is a set of programmable switches that allows wiring segments to be connected together in order to form longer connections.

To be as realistic as possible, we set the number of wiring segments to which each segment can be connected at a switch block,  $F_s$  [5], to 3, as this is the value used in most commercial FPGAs. There are two other important architectural parameters to which we will refer in the following sections. We define  $W$  to be the number of wiring segments in each routing channel, i.e. the channel capacity. We define  $F_c$  [5] to be the number of wiring segments to which a logic block input or output pin can connect in an adjacent channel. In Figure 4, for example,  $W$  is four and  $F_c$  is two.

### 3.2 CAD Flow

Figure 5 illustrates the CAD flow used in these experiments. First, the SIS [6] synthesis package is used to perform technology-independent logic optimization of each circuit. That is, SIS attempts to simplify the logic and remove redundant circuitry. Next, each circuit is technology-mapped into 4-LUTs and flip flops by FlowMap [7]. In other words, FlowMap takes a description of a circuit in terms of basic gates and implements it using only 4-LUTs and flip flops, as these are the only logic resources available in the FPGAs we study. Our VPACK program [1] then maps this netlist of 4-LUTs and flip flops into logic clusters with the specified values of  $N$  and  $I$ . At this point, then, the circuit is described as a set of interconnected logic blocks of the exact type that



**Figure 5:** Architecture Evaluation Flow.

exist in the FPGA we’re targeting. Finally, the VPR placement and routing tool [8] is used to place and route the circuit. Placement consists of choosing a position for each logic block within the FPGA so that the length of the wires needed to interconnect the circuitry is minimized, while routing consists of choosing which wires within the FPGA will be used to make each connection.

As Figure 5 shows, the circuit is repeatedly routed with different channel capacities until VPR finds the minimum number of wire segments per channel required to successfully route the circuit. At this point we have enough information to use our area model to evaluate the architecture’s area-efficiency.

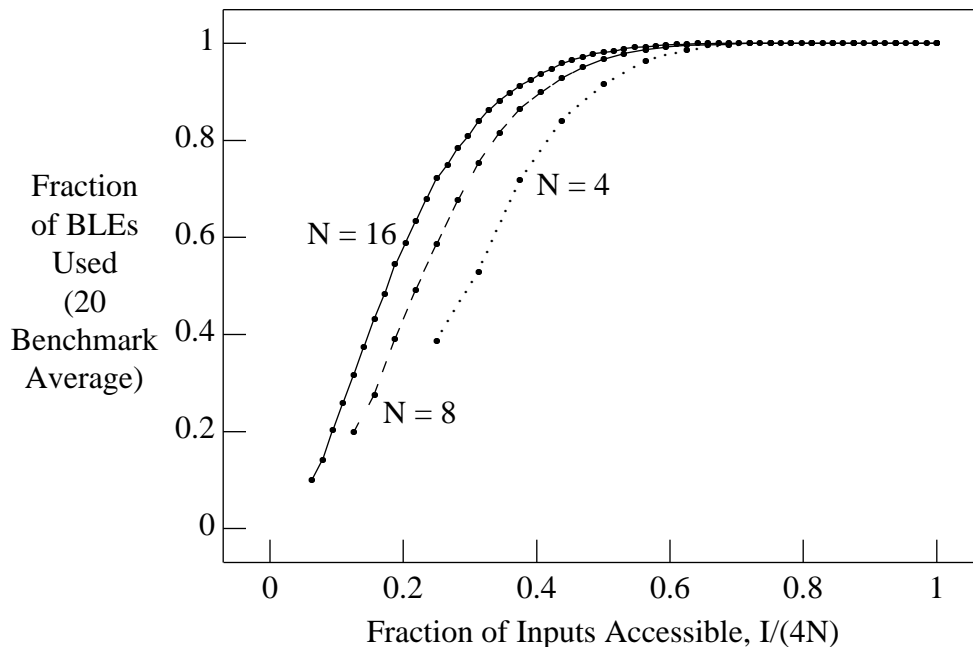
### 3.3 Area Model

The area model is based on counting the number of minimum-width transistors required to implement a benchmark circuit in each FPGA architecture (larger transistors are counted as several minimum width transistors). To allow averaging of results from circuits of different sizes, we use 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 [9]. This model tries to build an FPGA using as few transistors as possible without unduly compromising speed.

## 4 Experimental Results: Cluster Inputs Required vs. Cluster Size

As discussed in the introduction, the first question we wish to answer is how many distinct inputs,  $I$ , should be provided to a cluster of size  $N$ . Since the number of transistors required to implement each of the multiplexers shown in Figure 3 grows linearly with  $I$  (for large  $I$ ), we would like to make  $I$  as small as possible. On the other hand, if  $I$  is made too small, many of the BLEs in a logic cluster may become essentially unusable, reducing logic utilization and wasting area. We find the minimum value of  $I$  that allows good cluster utilization by running benchmark circuits through the first two steps shown in Figure 5, technology-mapping and cluster packing, and measuring the resulting logic utilization for different values of  $I$ . We define logic utilization to be the average number of BLEs per cluster that a circuit is able to use divided by the total number of BLEs per cluster,  $N$ .

Figure 6 shows how the average logic utilization of our 20 benchmarks varies with  $I$  for three different logic cluster sizes. The horizontal axis is the number of distinct inputs to the cluster relative to the total number of BLE inputs in a cluster, i.e.  $I/(4N)$ . For very low values of  $I$ , the 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, the logic utilization is essentially 100%. Clearly it is possible to pack BLEs together so that they have many common inputs and can reuse locally gener-



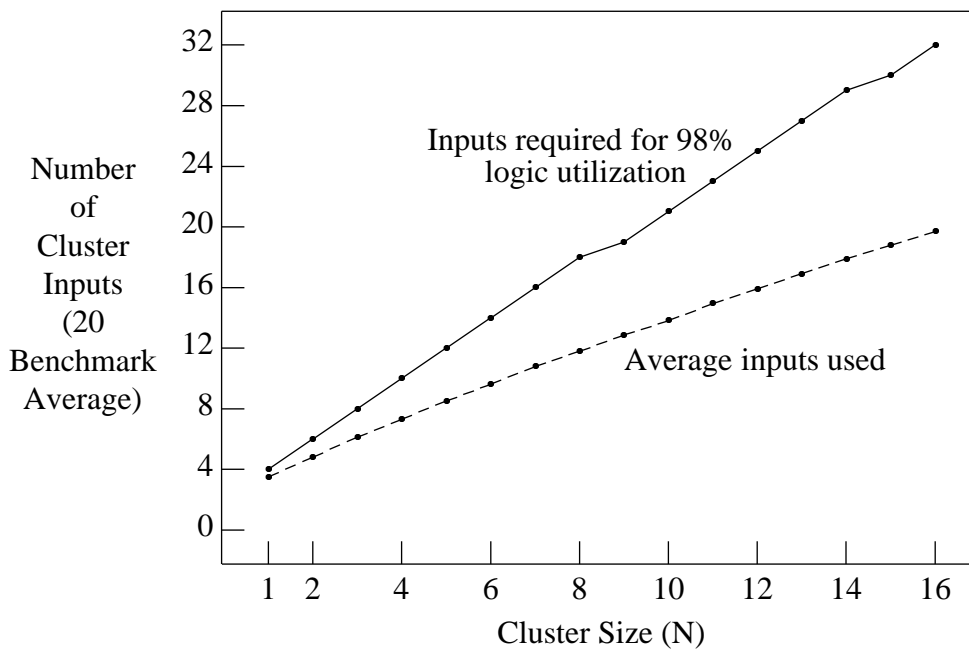
**Figure 6:** Logic utilization vs. number of logic cluster inputs.



ated 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.

The solid line in Figure 7 shows the value of  $I$  required to achieve 98% logic utilization as the cluster size,  $N$ , is varied, while the dashed line shows how the average number of logic cluster inputs that are 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 that are actually used grows even more slowly. On average, a cluster of size 1 uses 3.5 of its inputs, while an cluster of size 16 uses only 19.7 of its inputs. In other words, while the logic per cluster has increased by a factor of 16, the average number of connections that must be routed to each cluster has increased by a factor of only 5.6.

Our results indicate that commercial FPGAs can be more aggressive in reducing the value of  $I$ . For example, the Altera Flex 8K FPGAs use logic clusters with  $N = 8$  and  $I = 24$ , while 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, while 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.



**Figure 7:** Variation in inputs required and inputs used with cluster size.

## 5 Experimental Results: Routing Flexibility vs. Cluster Size

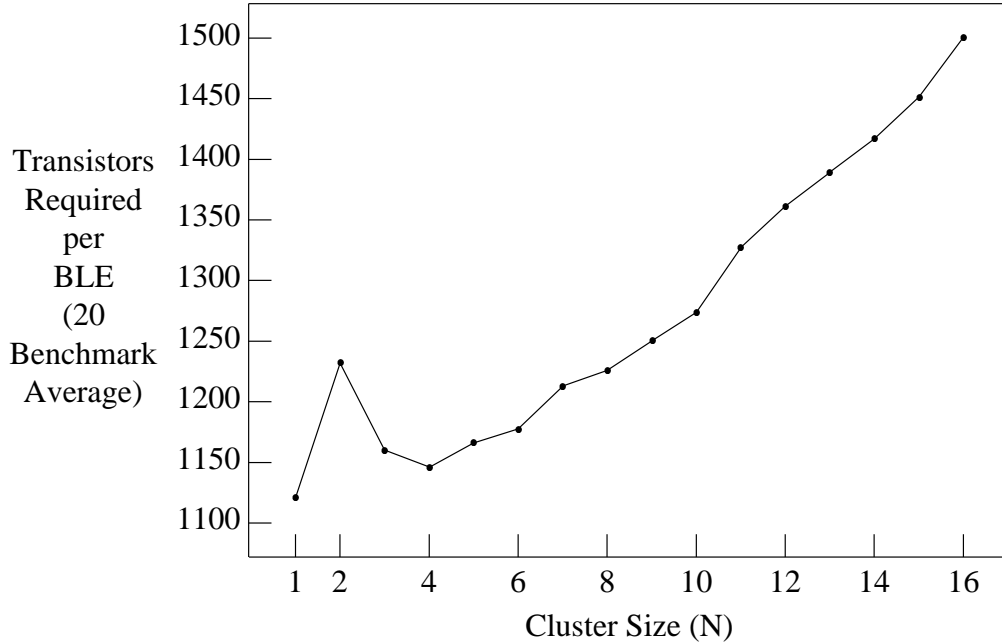
Before we can apply the experimental flow of Section 3 to see how area-efficiency varies with cluster size, we must choose  $F_c$ , the number of routing tracks to which each logic block pin can connect. On the one hand, using a smaller value of  $F_c$  reduces the number of programmable switches in 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 two competing objectives and achieves good area-efficiency.

For a cluster of size 1, a good value of  $F_c$  is  $W$ ; i.e. each logic block pin can be connected to any routing track in an adjacent channel. For larger clusters, however, setting  $F_c$  to  $W$  provides far more routing flexibility than is required, wasting area. Recall that the full connectivity of a logic cluster means that a net which must connect to a logic block input can be connected to *any* of the  $I$  inputs. Similarly, a net that must connect to a logic block output can connect to *any* of the  $N$  outputs. As  $N$  increases then, keeping  $F_c$  fixed at  $W$  provides an excessive number of ways to connect to each logic block. For example, a cluster of size one has 4 inputs and one output. If  $F_c = W$ , then, 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 have experimentally 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 use 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, ensuring that all the routing tracks in a channel can be readily used to interconnect blocks.

## 6 Experimental Results: Area-Efficiency vs. Cluster Size

We are now in a position to examine which cluster size leads to the most area-efficient FPGA. Throughout this section, the number of inputs,  $I$ , to a cluster of size  $N$  is chosen to be the minimum value that allows VPACK to achieve 98% logic utilization. This value of  $I$  allows our logic clusters to be essentially fully utilized, while minimizing the complexity of the cluster input multiplexers and the number of logic block pins to be connected to the main FPGA routing. We ran 20 benchmark circuits through the experimental flow described in Section 3, and determined the area



**Figure 8:** Area-efficiency versus cluster size.

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-efficiency within a 10% range. Clearly, with proper choices of  $I$  and  $F_c$  any cluster in this range, except perhaps a cluster of size 2, provides reasonable area-efficiency.

As one increases the cluster size from 1 to 2, area-efficiency worsens because 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 it has not yet reduced the number of connections to route between clusters enough to compensate. Further increases of cluster size, to  $N = 3$  and 4, improve area-efficiency because the local routing is able to more significantly reduce the amount of routing required between logic blocks. As the cluster size rises past  $N = 10$ , area-efficiency rapidly degrades. The complexity of the input multiplexers in a logic cluster grows quadratically with cluster size, and for sufficiently large clusters this swamps the area improvements gained by reducing the routing required between logic blocks.

## 7 Conclusions

There are three main conclusions to be drawn from this work. First, the number of distinct inputs required by a logic cluster grows fairly slowly with cluster size,  $N$ . A cluster of size  $N$

requires approximately  $2N + 2$  distinct inputs (for  $N \leq 16$ ). Secondly, because all the input and output pins of a cluster are logically equivalent, one can significantly reduce the number of routing tracks to which each logic cluster pin can connect,  $F_c$ , as one increases the cluster size. Finally, the area-efficiency of logic blocks containing between 1 and 8 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 the FPGA speed. Since a cluster-based logic block with appropriate values of  $N$ ,  $I$ , and  $F_c$  has area-efficiency comparable to that of a single BLE logic block, an FPGA can gain these advantages without any area penalty.

## References

- [1] V. Betz and J. Rose, "Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size," *IEEE Custom Integrated Circuits Conf.*, 1997, pp. 551 - 554.
- [2] J. Rose, R. J. Francis, D. Lewis and P. Chow, "Architecture of Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," *IEEE Journal of Solid State Circuits*, Oct. 1990, pp. 1217 - 1225.
- [3] A. Aggarwal and D. Lewis, "Routing Architectures for Hierarchical Field Programmable Gate Arrays," *Int. Conf. on Computer Design*, 1994, pp. 475 - 478.
- [4] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0," *Tech. Report*, Microelectronics Center of North Carolina, 1991.
- [5] S. Brown, R. Francis, J. Rose and Z. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.
- [6] E. M. Sentovich et al, "SIS: A System for Sequential Circuit Analysis," *Tech. Report No. UCB/ERL M92/41*, University 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," *Int. Workshop on Field Programmable Logic and Applications*, 1997, pp. 213 - 222.
- [9] V. Betz, *Ph.D. Dissertation*, in preparation, University of Toronto.