IEEE Transactions on VLSI Systems, Vol. 6, No. 3, pp. 400-408, Sept. 1998. 1999 IEEE Circuits and Systems Society TVLSI Best Paper.

Mesh Routing Topologies for Multi-FPGA Systems

Scott Hauck
Department of ECE
Northwestern University
Evanston, IL 60208 USA
hauck@ece.nwu.edu

Gaetano Borriello
Department of CSE
University of Washington
Seattle, WA 98195 USA
gaetano@cs.washington.edu

Carl Ebeling
Department of CSE
University of Washington
Seattle, WA 98195 USA
ebeling@cs.washington.edu

Abstract

There is currently great interest in using fixed arrays of FPGAs for logic emulators, custom computing devices, and software accelerators. An important part of designing such a system is determining the proper routing topology to use to interconnect the FPGAs. This topology can have a great effect on the area and delay of the resulting system. Crossbar, Hierarchical Crossbar, and Mesh interconnection schemes have all been proposed for use in FPGA-based systems. In this paper we examine Mesh interconnection schemes, and propose several constructs for more efficient topologies. These reduce inter-chip delays by more than 60% over the basic 4-way Mesh.

Introduction

Field-programmable gate arrays (FPGAs) are devices that can be programmed and reprogrammed to implement complex digital logic. They have much greater capacity than other programmable logic devices (PLDs), but unlike mask-programmable gate arrays and other high-density components they are in-circuit reprogrammable, allowing their configurations to be changed by a completely electrical process.

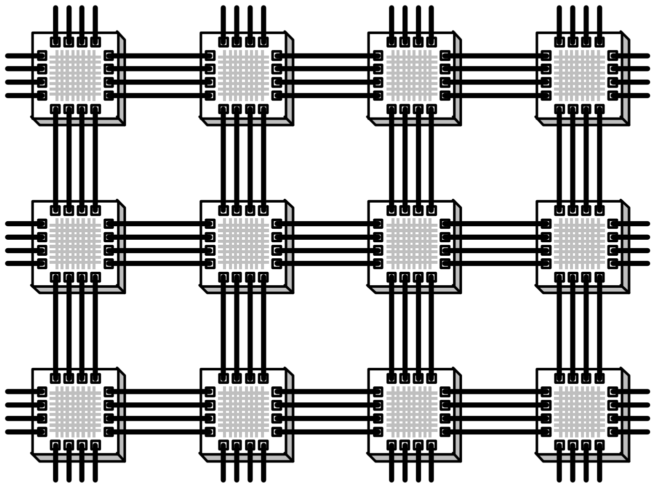


Figure 1. Basic mesh topology.

There is currently tremendous interest in the development of computing platforms from standard FPGAs. These multi-FPGA systems harness multiple FPGAs, connected in a fixed pattern, to implement complex logic structures. The circuit mapped onto the FPGAs need not be standard hardware equations, but can even be operations from algorithms and general computations. While these FPGA-based custom-computing machines may not challenge the performance of microprocessors for all applications, for computations of the right form a multi-FPGA system can offer extremely high performance, surpassing any other programmable solution. Although a custom hardware implementation will be able to beat the power of any generic programmable system, and thus there must always be a faster solution than a multi-FPGA system, the fact is that few applications will ever merit the expense of creating application-specific solutions. An FPGA-based computing machine, which can be reprogrammed like a standard workstation, offers the highest realizable performance for many different applications. In a sense it is a hardware supercomputer, surpassing traditional machine architectures for certain applications. This potential has been realized by many different research machines. The Splash system [6] has provided performance on genetic string matching that is almost 200 times greater than all other supercomputer implementations. The DECPeRLe-1 system [21] has demonstrated world-record performance for many other applications, including RSA cryptography. Many other truly world-class implementations have been created [4, 5, 9 - 11, 13, 15 - 18].

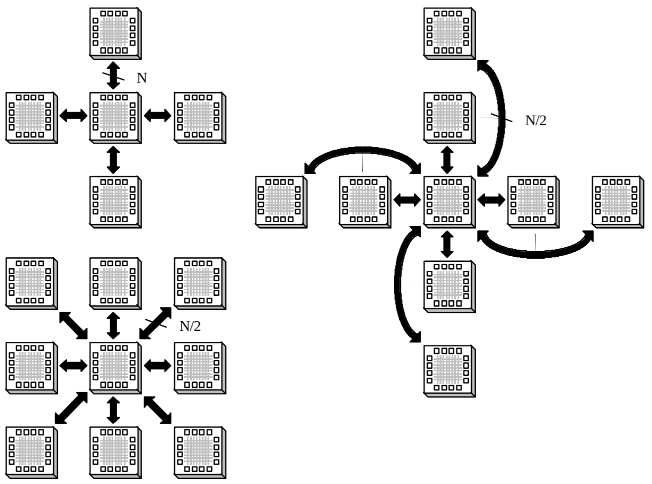


Figure 2. 4-way (top left), 8-way (bottom left), and 1-hop (right) mesh routing topologies.

One of the applications of multi-FPGA systems with the greatest potential is logic emulation. The designers of a

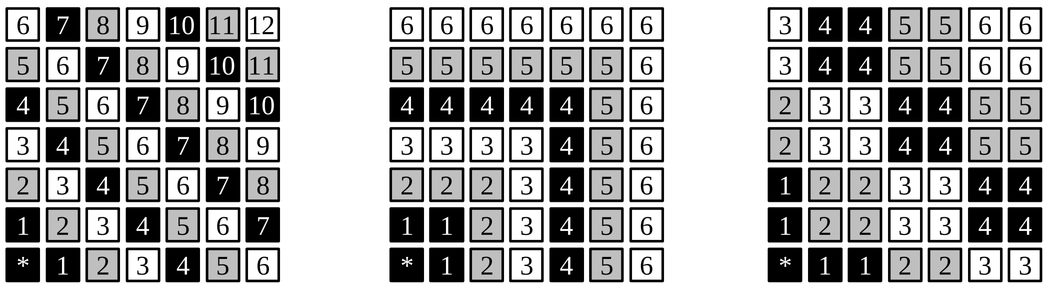


Figure 3. Distance to reach FPGAs in 4-way (left), 8-way (center), and 1-hop (right) topologies. Distances are from the FPGA in the lower-left corner, and represent the number of chip-to-chip connections needed to reach that chip.

custom chip need to verify that the circuit they have designed actually behaves as desired. Software simulation and prototyping have been the traditional solutions to this problem. However, as chip designs become more complex, software simulation is only able to test an ever decreasing portion of the chip's functionality, and it is quite expensive in time and money to debug by repeated prototype fabrications. The solution is logic emulation, the mapping of the circuit under test onto a multi-FPGA system. Since the logic is implemented in the FPGAs in the system, the emulation can run at near real-time, yielding test cycles several orders of magnitude faster than software simulation, yet with none of the time delays and inflexibility of prototype fabrications. These benefits have led many of the advanced microprocessor manufacturers to include logic emulation in their validation methodologies.

Multi-FPGA systems have great potential for logic emulation and reconfigurable computing tasks. An important aspect shared by all of these systems is that they use multiple FPGAs, preconnected in a fixed routing structure, to perform their tasks. While FPGAs can be routed and rerouted in their target systems, the pins moving signals between FPGAs are fixed by the routing structure on the implementation board. Some of the topology concerns in small arrays can be removed by FPICs [1, 12]. FPICs are devices similar to FPGAs but without any logic functions, and are able to produce nearly arbitrary routing between their I/O pins. However, large FPGA systems with FPICs for routing will still need to fix the topology for inter-FPIC routing.

There are many different possible multi-FPGA system topologies [7]. The routing structure used in a multi-FPGA system has a large impact not only on system speed, but also on capacity and system extendibility. Crossbar topologies provide a predictable routing delay, but they sacrifice scalability and chip utilization. Hierarchical crossbar structures [20] have less predictable routing delays, since signals may have to pass through many FPGAs, but have improved scalability. Mesh connections are scaleable, and may have better utilization than the other structures, but have even worse routing predictability. Although mesh topologies have been criticized due to perceived pin limitations, new techniques such as Virtual Wires [2, 19] and future high-I/O FPGA packages make meshes a very viable alternative.

Determining the proper routing topology for a multi-FPGA system is a complex problem. The necessary first step is to determine the best way to use a given routing topology, so that an honest comparison between different topologies can be performed. In this paper, we examine mesh topologies, and present several constructs for more efficient structures. We then provide a quantitative study of the effects of these constructs, and examine their impact on automatic mapping software. Architectural studies of multi-FPGA systems based on crossbars [3] and hierarchical crossbars [20] can be found elsewhere.

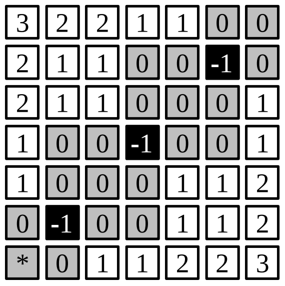


Figure 4. Distances of routes in 1-hop minus distance in 8-way topologies. The number is the number of extra connections necessary to route in an 8-way instead of a 1-hop topology.

Mesh Routing Structures

The obvious structure to use for a mesh topology is a 4-way interconnection (Figure 1), with an FPGA connected to its direct neighbors to the north, south, east and west. All the pins on the north side of an FPGA are connected to the south edge of the FPGA directly to the north, and the east edge is connected similarly. In this way the individual FPGAs are stitched together into a single, larger structure, with the Manhattan distance measure that is representative of most FPGAs carried over to the complete array structure. In this and other topologies an inter-FPGA route incurs a cost in I/O pin and internal FPGA routing resources. The

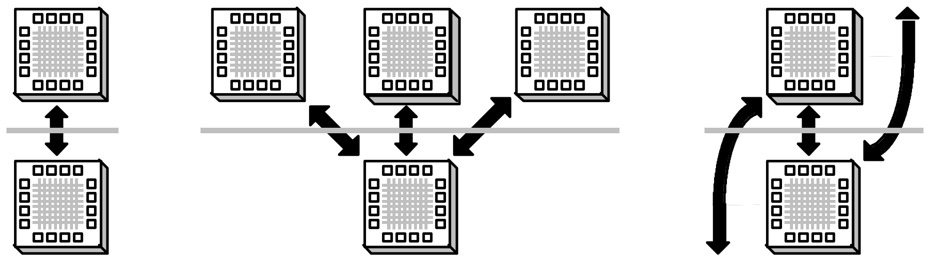


Figure 5. Bisection bandwidth in 4-way (left), 8-way (center), and 1-hop (right) topologies. Three times as many connections cross the bisecting line in the 8-way and 1-hop topologies than in the 4-way topology, though each connection contains half as many wires. This results in a 50% increase in bisection bandwidth.

rest of this paper will attempt to reduce usage of each of these resources. In this way, we not only optimize the delay in the system by shortening routes, but we also reduce the area needed to map a circuit.

I/O Pin Usage Optimization

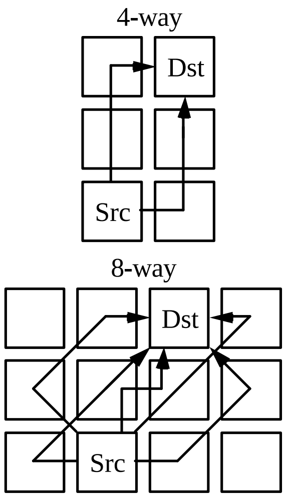
In order to reduce the average number of I/O pins needed to route signals, we can increase the number of neighbors connected to an FPGA. Instead of the simple 4-way connection pattern (Figure 2 top left), we can adopt an 8-way topology. In the 8-way topology (Figure 2 bottom left) an FPGA is not only connected to those FPGAs horizontally and vertically adjacent, but also to those FPGAs diagonally adjacent. A second alternative is to go to a 1-hop topology (Figure 2 right), where an FPGA is connected to the two closest FPGAs directly above, below, to the right, and to the left. One could also consider 2-hop, 3-hop, and longer connection patterns. However, these topologies greatly increase PCB routing complexity, and wiring delays become significant.

We assume here that all connected FPGAs within a specific topology have the same number of wires connecting them, though a topology could easily bias for or against specific connections. Since the 8-way and 1-hop topologies have twice as many neighbors as the 4-way topology, and FPGAs have a fixed number of pins, each pair of connected FPGAs in these topologies have half as many wires connecting them as in the 4-way case.

Switching from a 4-way to an 8-way or 1-hop topology has an impact on two major factors: average I/O pin usage, and bandwidth. Figure 3 shows one quadrant of a mesh under the three different topologies, with each box corresponding to an FPGA. The number indicates how many I/O connections are required to reach that FPGA from the source FPGA at the lower-left corner (shown with an asterisk). A *connection* is a pair of pins on different FPGAs interconnected by a single board trace. As we can see, both the 8-way and 1-hop topologies can reach more FPGAs within a given number of I/O connections than can the 4-way topology. In fact, if we consider an entire mesh instead of a single quadrant, the 8-way can reach twice as many FPGAs as the 4-way within a given number of I/O connections, and the 1-hop topology can reach three times as many FPGAs as a 4-way topology (although within a single I/O connection there are only twice as many neighbors in the 1-hop as in the 4-way, at longer distances it reaches three times as many). On average a route in a 1-hop topology requires about 40% less I/O pins than a route in a 4-way topology. Another interesting observation is that the 1-hop topology can reach almost all the same FPGAs as the 8-way topology can in the same number of I/O connections (see Figure 4). The only exceptions to this are the odd numbered FPGAs along the diagonals from the source in the 8-way topology. What this means is that there is little benefit in using a combined 8-way & 1-hop topology, since the 1-hop topology gives almost all the benefit of the 8-way topology.

One would expect the I/O pin usage optimization to come at the price of lower bandwidth in the mesh. However, this turns out not to be the case. The standard method for measuring bandwidth in a network is to determine the minimum bisection bandwidth. The minimum bisection bandwidth is computed by measuring the bandwidth between all possible splittings of the network into two equal-sized halves, and determining the minimum such bandwidth. This number is important, since if routes are randomly scattered throughout a topology half of the routes will have to use part of this bandwidth, and thus twice the bisection bandwidth is an upper bound on the bandwidth in the system for random routes. In the mesh topologies we have presented, the minimum bisection bandwidth can be found by splitting the mesh vertically or horizontally into two halves. As shown in Figure 5, cutting each row or column in a 4-way mesh splits one pair of connected neighbors, while in an 8-way and 1-hop topology it splits 3 pairs. Since there are three times as many neighbor pairs split, though each pair has half the bandwidth (remember that the 4-way topology has half the number of neighbors, so each pair of neighbors is connected by twice the wires), the 8-way and 1-hop topologies thus have 50% more bisection bandwidth than the 4-way mesh.

An alternative way to view bandwidth is point-to-point bandwidth, the total amount of bandwidth between any two chips in the system. If we simply ask how much bandwidth



8-way vs. 4-way

1.5	1.25	2.0	2.0	2.0
1.5	1.25	1.75	1.75	2.0
1.5	1.25	1.5	1.75	2.0
0.5	0.75	1.25	1.25	1.25
*	0.5	1.5	1.5	1.5

1-hop vs. 4-way

3.0	2.0	2.0	2.0	2.0
1.5	2.0	2.0	2.0	2.0
1.0	1.25	2.0	2.0	2.0
0.5	0.5	1.25	2.0	2.0
*	0.5	1.0	1.5	3.0

Figure 6. Example of the point-to-point fast bandwidth calculation in 4-way (top left) and 8-way (bottom left) meshes. The 8-way routes are only allowed to use three I/O connections, the number of I/O connections necessary to reach the destination in the 4-way topology. Also included is a complete relative bandwidth summary of 8-way vs. 4-way (center) and 1-hop vs. 4-way (right) topologies.

is available from one specific FPGA to another, then (barring missing connections at mesh edges) all meshes have exactly the same point-to-point bandwidth. This is because there are independent paths ("independent" implying two routes don't share individual I/O connections, though they may move through the same FPGAs) from every wire leaving any source FPGA to any destination FPGA. A more interesting issue is that of fast bandwidth, bandwidth that does not require excessively long routing. Specifically, we realize that since a 4-way mesh has twice as many connections to each of its neighbors than an 8-way or 1-hop topology, the 4-way topology can send twice as many signals to that destination using a single I/O connection as can the other topologies. By extrapolation, we would expect that the 4-way topology has more bandwidth to those FPGAs two or more I/O connections away than the other topologies. However, if we allow each topology to use the same number of I/O connections (specifically, the minimum number of I/O connections necessary to reach that FPGA in a 4-way topology), the 8-way and 1-hop topologies actually have greater fast bandwidth. As shown in Figure 6 left, if we route to an FPGA two steps north and one step east, it requires three I/O connections in the 4-way topology, and there are two independent paths between the FPGAs. If we allow the 8-way topology to use three I/O connections, it actually has five independent paths between the two FPGAs (Figure 6 bottom left). Since each path in an 8-way topology has half the bandwidth as a path in a 4-way topology, the 8-way has 25% more fast bandwidth between these FPGAs. Figure 6 shows a complete comparison for a quadrant of the mesh, with the numbers given representing the ratio of 8-way vs. 4-way fast bandwidth (center), and 1-hop vs. 4-way fast bandwidth (right). The ratio numbers are for bandwidth between each FPGA and the FPGA at lower left. As can be seen, in all cases except the FPGAs directly adjacent vertically, horizontally, or diagonally, the 8-way and 1-hop topologies have greater fast bandwidth than the 4-way topology, up to a factor of three.

Thus, as we have shown, the 1-hop topology reduces average I/O pin usage by 40%, increases minimum bisection bandwidth by 50%, and has greater point-to-point fast bandwidth than the 4-way topology to almost all other FPGAs, up to three times as great.

Internal Routing Resource Usage Optimization

In this section we describe FPGA pin interconnection patterns that minimize FPGA internal routing resource usage. While most of the optimizations described here apply equally well to 4-way, 8-way, and 1-hop topologies, we'll concentrate on 4-way topologies for simplicity. Also, we'll abstract the individual FPGAs into a grid of internal routing resources, with the grid width equal to the distance between adjacent FPGA pins. Finally, we optimize for random-logic applications such as logic emulators and software accelerators. Mappings of systolic or otherwise highly regular and structured circuits may require different topologies.

As described earlier, the obvious way to build a 4-way topology is to connect all the pins on the east side of an FPGA to the pins on the west side of the FPGA directly to the east (the north edge is connected similarly). The problem with this construct is demonstrated in Figure 7 top. The figure shows the amount of internal routing resources needed to route from the top center of the FPGA at right to the other locations in that FPGA, and the FPGAs one and two steps to the right. Chip-to-chip resource costs are not

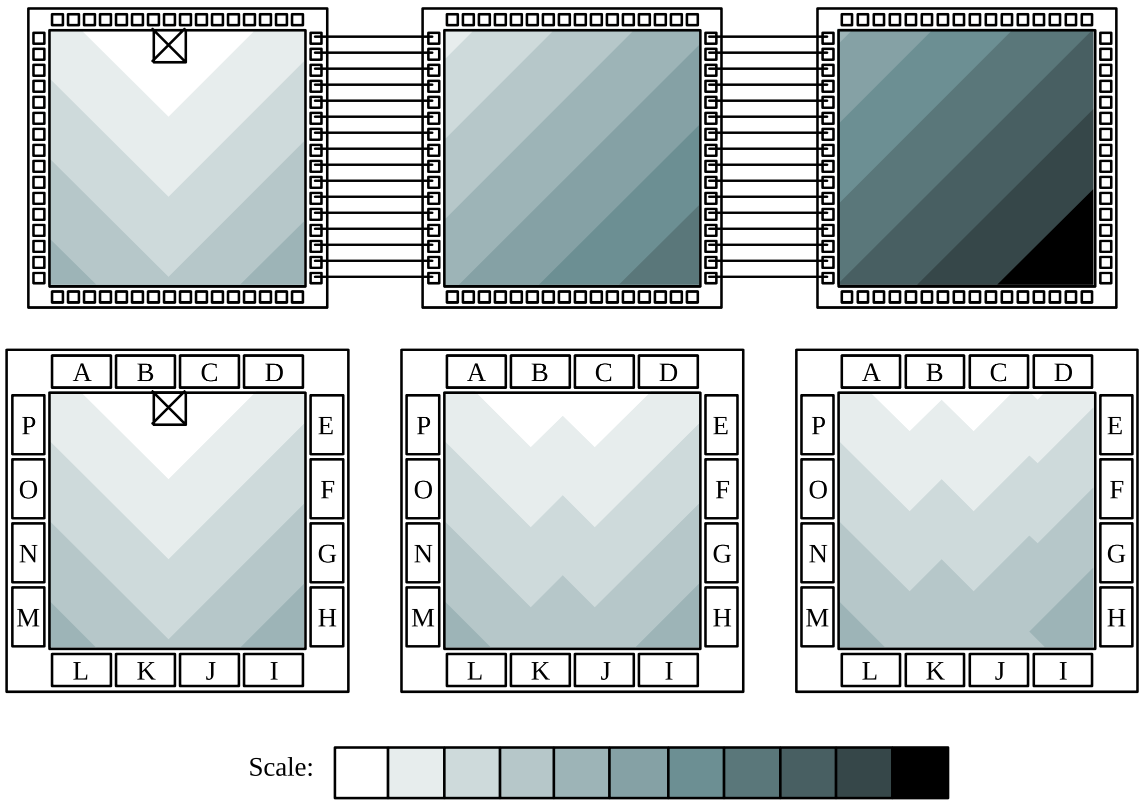


Figure 7. Distances from the X in leftmost FPGA in normal (top) and Superpin (bottom) connection pattern, on a scale from white (shortest) to black (longest). Superpin connections are given by letters, with Superpins with the same letter connected together in adjacent FPGAs. Distances represent internal routing resources, since all routes use the minimum number of chip-to-chip connections, and thus the chip-to-chip connection cost is fixed.

included, since all routes use the minimum number of chip-to-chip connections, and thus the chip-to-chip costs are fixed. Because the pins connecting an FPGA to its neighbor to the east are on the opposite side of the chip from the pins connected to the neighbor to the west, and pins connected to the south are opposite to pins connected to the north, a signal moving through several FPGAs must traverse the length or width of the intervening FPGAs. Thus, as shown in Figure 7 top, moving from the FPGA at left to the FPGA at right requires a large amount of internal routing resources.

An alternative is to scatter the pins connecting pairs of FPGAs around the edges of the FPGAs. We form groups called Superpins, and within a Superpin is one pin connected to each of that FPGA's neighbors. Thus, a Superpin in a 4-way topology has four pins, and a Superpin in an 8-way or 1-hop topology has eight pins. Within a Superpin, pins that are likely to be routed together in a mapping are grouped together. Specifically, long-distance signals will usually require pins going in opposite directions to be connected together in intermediate FPGAs. Signals which move between directly connected FPGAs will not be affected by any ordering of pins within a Superpin, since they never use more than one pin on the same FPGA. Thus, around the edge of an FPGA in a 4-way topology we order the pins N, S, E, W, N, S, E, W…, and in a 1-hop the pins are ordered NN, SS, EE, WW, N, S, E, W, NN, SS, EE, WW…, where the pin NN is connected to an FPGA two steps north of the source FPGA. In an 8-way topology a long-distance route that doesn't connect together pins going in opposite directions will instead probably connect signals 45 degrees off of opposite (e.g. S and NW or NE). Thus, the connection pattern N, S, SW, NE, E, W, NW, SE, S, N, NE, SW, W, E, SE, NW, N, S… is used, since it puts opposite pins together, and pins 45 degrees off of opposite are at most 2 pins distant.

As shown in Figure 7 bottom, if we connect the Superpins together in the obvious manner, with Superpins in one FPGA connected to the corresponding Superpins in neighboring FPGAs, we get significant routing resource usage improvements. The Superpin topology almost removes incremental routing resource usage in intermediate FPGAs.

We can do better than the Superpin strategy just presented by realizing that not only can we scatter the connections between neighboring FPGAs around their edges, but we can also scatter the connections to specific sides of these FPGAs around its neighbor's edges. Put differently, instead

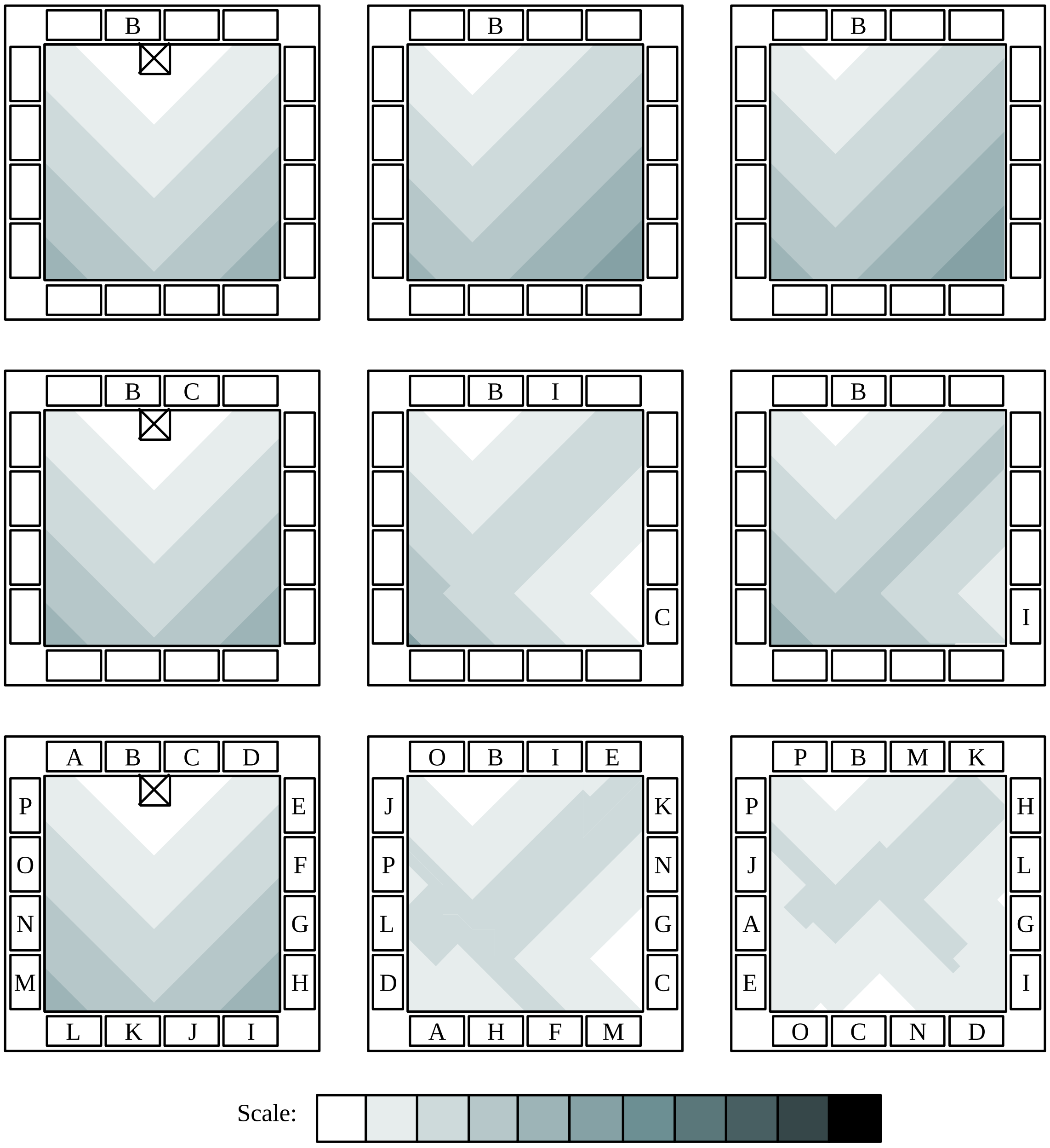


Figure 8. Intuition behind permuted Superpins. A single connection (at top) gives most of the benefit of full unpermuted Superpins. By changing connection **C** to the lower-right corner (middle), more short routes are achieved. Note that connection **I** is simply connection **C** for the middle FPGA. Bottom shows full permuted Superpins, with even shorter routes in further FPGAs. The scale ranges from white (shortest) to black (longest).

of connecting Superpins in one FPGA to corresponding Superpins in adjacent FPGAs, we can instead permute these connections to improve routing resource usage. As shown in Figure 8 top, simply making the connection labeled "**B**" gives most of the benefit of the complete unpermuted Superpin pattern given in Figure 7. Thus, connecting "**C**" as we did in Figure 7 will give little extra benefit, since the short routes the "**C**" connection creates will lead to locations that already have short routes due to the "**B**" connection. If we instead connect "**C**" in the first FPGA to a location on the lower right edge of the adjacent FPGA (Figure 8 middle), we create short routes to locations that only had long routes through "**B**". By continuing this approach, we route Superpin connections so that not only are there short routes from one location in one FPGA to its direct neighbors, but we permute the Superpins such that all locations in the source FPGA have short routes to all locations in all other FPGAs (Figure 8 bottom).

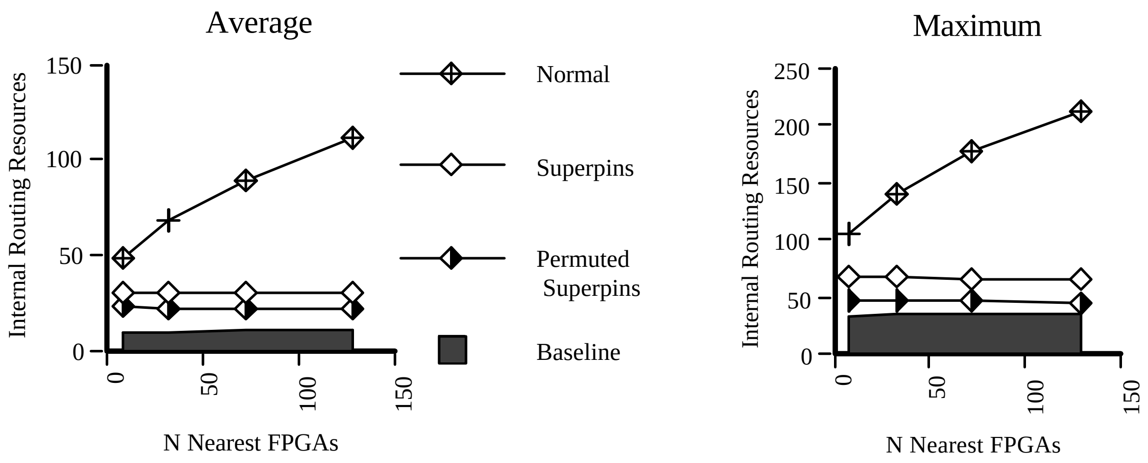


Figure 9. Average (left) and maximum (right) internal routing resource usage from each location in the source FPGA to all locations in the N nearest destination FPGAs in a 1-hop topology.

An interesting observation is that by having two (identical) permutations in series in Figure 8 bottom, we in fact use fewer internal routing resources to reach locations in FPGAs two steps away (the rightmost FPGA) than we need for locations in adjacent FPGAs (middle FPGA in Figure 8 bottom). This effect does diminish with more permutations in series, so that average internal routing resource usage begins to increase again further away from the source, as the incremental cost of routing resources in intermediate FPGAs dominates the gain of additional permutations.

We have presented elsewhere [7] a lower bound on the quality of permutations. Unfortunately, we do not have a simple, deterministic construction method for finding optimum permutations. However, it is fairly easy to write a simple simulated annealing program for permutations which gives very good results. Our admittedly inefficient and unoptimized annealer is less than 500 lines of C code, and has consistently found permutations within a few percent of the lower bounds. Although the runtimes are up to a few days on a Sparc 10, these times are very reasonable for the design of a fixed multi-FPGA system, something that will be done infrequently, and which takes weeks or months to complete.

A quantitative comparison of the internal routing resource usage under the Superpin and Permuted Superpin constructs, all within a 1-hop topology, is presented in Figure 9. These graphs represent the average and maximum resource usage from every point in a source FPGA to every point in the nearest N neighbor FPGAs in the system (FPGAs are represented by grids as described earlier, with 36 pins on a side). An interesting observation is that while the Superpins have a great impact, almost totally removing incremental resource usage in intermediate FPGAs, the Permutations only decrease resource usage by about 28%. One explanation for this is the theoretic lower bound ("Baseline") shown above. This lower bound comes from the observation that in any 1-hop topology, a route must use at least enough routing resources to go from the

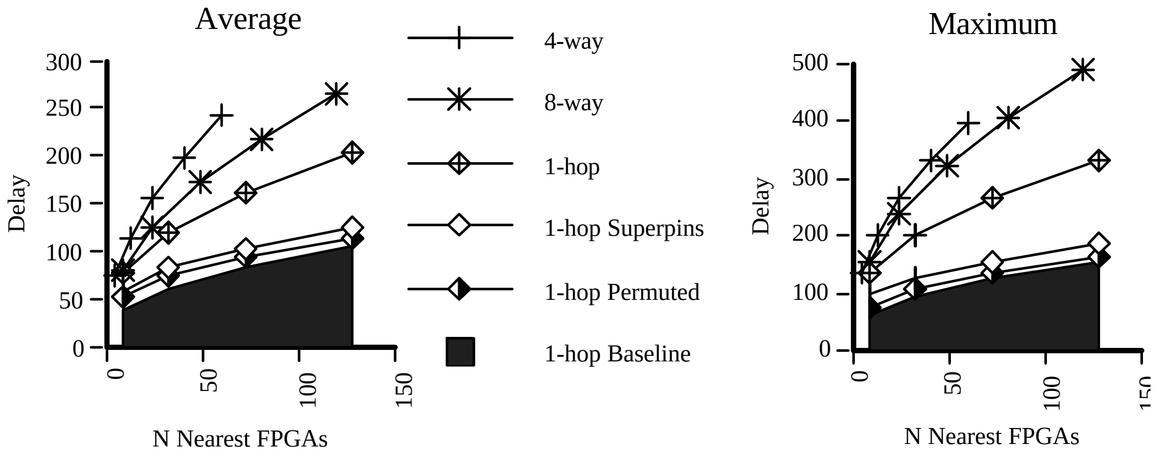


Figure 10. Graphs of average (left) and maximum (right) delay from each location in the source FPGA to all locations in the N nearest destination FPGAs.

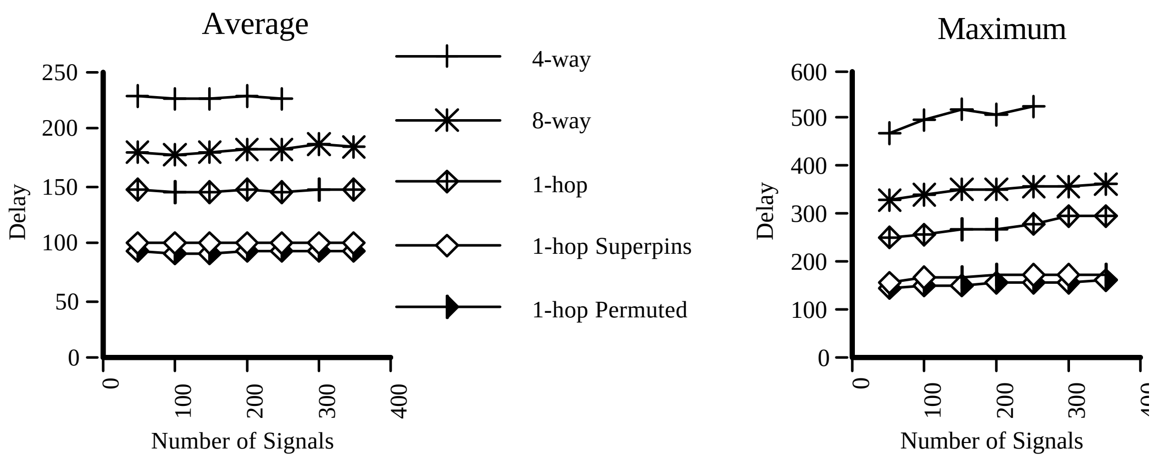


Figure 11. Average and maximum distance of signals under several topologies in a 5x5 array. The maximum values increase with the number of signals routed because we get more random variation with larger sample sizes from a random distribution.

route starting point to the nearest pin on the source FPGA, plus at least one routing resource in each intermediate FPGA, plus enough resources to go between the route ending point and the closest pin on the destination FPGA. As shown, the greatest conceivable improvement over the standard Superpin pattern (assuming we stay within a 1-hop topology) is approximately 60%, and the permutations achieve almost half of this potential. However, when designing a multi-FPGA system, the benefits of permutations must be carefully weighed against the increased board layout complexity.

Overall Comparisons

We can make an overall comparison of all the topological improvements, both I/O pin and internal routing resource optimization, by examining the inter-FPGA routing delays. As shown in Figure 10, we present the average and maximum delay from every point in a source FPGA to every point in the nearest N neighbor FPGAs in the system. The FPGAs are represented as grids with 36 pins on a side, and the delay incurred in using an FPGA's I/O pin is 30 times greater than a single internal routing resource. These numbers are similar to delays found in the Xilinx 3000 series [22]. Note that this approximates possibly quadratic delays in internal routing resources as a linear function. As shown, an 8-way topology decreases delays by 22% over the standard 4-way topology, while a 1-hop topology decreases delays by 38%. By using the permuted Superpin pattern, the delays are decreased by an additional 25%, reducing overall delays by a total of 63%.

While the above numbers give an idea of how the features decrease routing costs at different distances, they ignore the fact that we do not just route a single signal, but in fact have many signals fighting for the same resources. To measure these conflicts, we have used the router developed for the Triptych FPGA project [14], which can be retargeted to different domains by altering a routing resource template. This router optimizes both area utilization and delay, making it a good experimental platform for this domain. As before, we abstracted the individual FPGAs to a Manhattan grid, and allowed signals to share edges in this grid. Thus, this model ignores internal routing conflicts. However, these conflicts would have the greatest impact on those topologies that use the most routing resources, especially resources nearest the FPGA center. Thus, ignoring these conflicts will in general decrease the benefit of better topologies, since they use less resources, and the resources they use are primarily at the chip edge. We also do not include signals that begin and end on the same FPGA, because these are unaffected by the inter-chip topologies.

The first two graphs (Figure 11) show the average and maximum cost for signals in each of the routing topologies, assuming a random distribution of sources and sinks of signals across a 5 by 5 array of FPGAs. Note that the same random data sets are used for all topologies at a given size, since this means that all topologies will be subjected to similar routing conditions. Again, moving between chips costs 30 times as much as a single step inside an FPGA, and the FPGAs have 36 pins on a side. The horizontal axis for both graphs is the total number of signals routed in a given trial, and eight trials are averaged together to generate each point. Trials were run at 50 signal increments until the router failed to route all signals.

There is a question of how well some of the topologies handle multi-signal buses and other situations where several signals move between the same sources and sinks. Specifically, one might expect the permutation topologies to have trouble with buses, since while there is a short path between most sources and sinks, there are few if any parallel paths. To determine if this is true, the second set of

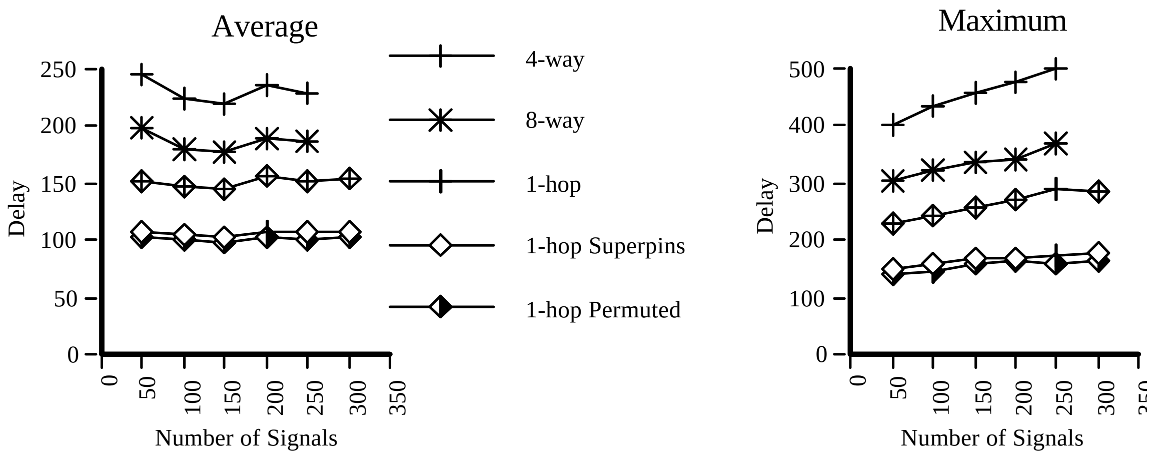


Figure 12. Average and maximum distance of signals under several topologies in a 5x5 array, with all signals in 5-signal buses. The maximum values increase with the number of signals routed because we get more random variation with larger sample sizes from a random distribution. The dip in the average values is from greater random variation due to the smaller number of random points in these examples, since with 5 signal bundles only one-fifth the number of random points are chosen.

graphs (Figure 12) is for a population of 5 signal bundles with random sources and sinks, though signals within a bundle share the same source and sink.

The most striking aspect of the previous graphs is how little congestion seems to affect routing distance. Although samples were taken in 50-signal increments until the router failed, there seems to be little resulting extra routing necessary. Although the graphs of maximum lengths do show increases, this is mostly due to the fact that a larger number of elements from a random distribution will tend to include greater extremes. The graphs for buses are less flat than the other trials, but this is due to the fact that each bus data set has one fifth as many random points, increasing the variance. More importantly, the benefits shown in our original graphs (Figure 10) are demonstrated in actual routing experiments. The 8-way topology is approximately 21% better than the 4-way topology, and the 1-hop is 36% better. Superpins improve the 1-hop topology by about 31%, with permutations saving an additional 5%. Also, in the first set of graphs the 8-way and 1-hop topologies successfully route 40% more signals than the 4-way topology, demonstrating the increase in minimum bisection bandwidth.

Automatic Mapping Tools

Since many multi-FPGA systems will not be used for hand mappings, but instead depend on automatic mapping tools, it is important that a routing topology not only decrease routing costs, but do so in a way that automatic tools can exploit. Since our previous comparisons involved using an automatic routing tool in the congestion examples, and since these experiments yielded distances equivalent to our previous average distance measurements, it is fairly clear that routing tools can exploit our improved topologies. As described in [8], we have developed a pin assignment tool (similar to a detailed router) for inter-FPGA routing, and the only impact of the improved topologies on this tool is the loss of a slight speed optimization opportunity. Partitioning tools are also easily adapted, since the locality needed for meshes is still the primary concern, though the number of closest neighbors is increased. Thus, automatic mappings tools for standard 4-way meshes should be able to be easily extended to the topologies presented here. More details on multi-FPGA system software can be found elsewhere [7].

Conclusions

We have presented several techniques for decreasing routing costs in mesh interconnection schemes: 1-hop interconnections, Superpins, and Permutations. Through the retargeting of an automatic routing tool, we have demonstrated an overall improvement of more than a factor of 2. While better mesh topologies may be feasible, especially if we allow permutations to operate on individual signals instead of Superpins, theoretical lower bounds (the baseline in Figure 9) prove that there is little room for improvement. Real improvements might come from increasing the direct neighbors of an FPGA from 8 to 12 (a 3-D, 1-hop mesh) or more, but the Superpin and Permutation techniques would still be applicable.

The major open question is whether any mesh system makes sense, or if trees, hypercubes, crossbars, or some other general topology is a better choice. However, if this paper is any indication, the best implementation of a given topology may not be obvious, requiring a close look at individual candidate topologies before overall topological comparisons can be completed.

Acknowledgments

The authors are grateful for significant help from Larry McMurchie, without whom the routing trials conducted in this paper would not have been possible. This paper has benefited from the patience of several people, most particularly Elizabeth Walkup, Donald Chinn, and Ross Ortega. This research was funded in part by the Defense Advanced Research Projects Agency under Contract N00014-J-91-4041. Gaetano Borriello and Carl Ebeling were supported in part by NSF Presidential Young Investigator Awards.

List of References

[1] Aptix Corporation, *Data Book*, San Jose, CA, February 1993.

[2] J. Babb, R. Tessier, A. Agarwal, "Virtual Wires: Overcoming Pin Limitations in FPGA-based Logic Emulators", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 142-151, 1993.

[3] P. K. Chan, M. D. F. Schlag, "Architectural Tradeoffs in Field-Programmable-Device-Based Computing Systems", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 152-161, 1993.

[4] C. P. Cowen, S. Monaghan, "A Reconfigurable Monte-Carlo Clustering Processor (MCCP)", *IEEE Workshop on FPGAs for Custom Computing Machines,* pp. 59-65, 1994.

[5] S. A. Cuccaro, C. F. Reese, "The CM-2X: A Hybrid CM-2 / Xilinx Prototype", *IEEE Workshop on FPGAs for Custom Computing Machines,* pp. 121-130, 1993.

[6] M. Gokhale, B. Holmes, A. Kopser, D. Kunze, D. Lopresti, S. Lucas, R. Minnich, P. Olsen, "Splash: A Reconfigurable Linear Logic Array", *International Conference on Parallel Processing,* pp. 526-532, 1990.

[7] S. Hauck, *Multi-FPGA Systems*, Ph.D. Thesis, University of Washington, Department of Computer Science & Engineering, 1995.

[8] S. Hauck, G. Borriello, "Pin Assignment for Multi-FPGA Systems", University of Washington, Dept. of Computer Science & Engineering Technical Report #94-04-01, April 1994.

[9] D. T. Hoang, "Searching Genetic Databases on Splash 2", *IEEE Workshop on FPGAs for Custom Computing Machines,* pp. 185-191, 1993.

[10] H. Högl, A. Kugel, J. Ludvig, R. Männer, K. H. Noffz, R. Zoz, "Enable++: A Second Generation FPGA Processor", *IEEE Symposium on FPGAs for Custom Computing Machines,* 1995.

[11] N. Howard, A. Tyrrell, N. Allinson, "FPGA Acceleration of Electronic Design Automation Tasks", in W. R. Moore, W. Luk, Eds., *More FPGAs*, Oxford, England: Abingdon EE&CS Books, pp. 337-344, 1994.

[12] I-Cube, Inc., "The FPID Family Data Sheet", Santa Clara, CA, February 1994.

[13] E. Lemoine, D. Merceron, "Run Time Reconfiguration of FPGA for Scanning Genomic DataBases", *IEEE Symposium on FPGAs for Custom Computing Machines,* 1995.

[14] L. E. McMurchie, C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs", *ACM/SIGDA International Symposium on Field Programmable Gate Arrays,* pp. 111-117, 1995.

[15] L. Moll, J. Vuillemin, P. Boucard, "High-Energy Physics on DECPeRLe-1 Programmable Active Memory", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays,* pp. 47-52, 1995.

[16] S. Monaghan, C. P. Cowen, "Reconfigurable Multi-Bit Processor for DSP Applications in Statistical Physics", *IEEE Workshop on FPGAs for Custom Computing Machines,* pp. 103-110, 1993.

[17] H. Schmit, D. Thomas, "Implementing Hidden Markov Modelling and Fuzzy Controllers in FPGAs", *IEEE Symposium on FPGAs for Custom Computing Machines,* 1995.

[18] S. D. Scott, A. Samal, S. Seth, "HGA: A Hardware-Based Genetic Algorithm", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays,* pp. 53-59, 1995.

[19] C. Selvidge, A. Agarwal, M. Dahl, J. Babb, "TIERS: Topology IndependEnt Pipelined Routing and Scheduling for VirtualWire™ Compilation", *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 25-31, 1995.

[20] J. Varghese, M. Butts, J. Batcheller, "An Efficient Logic Emulation System", *IEEE Transactions on VLSI Systems*, Vol. 1, No. 2, pp. 171-174, June 1993.

[21] J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, P. Boucard, "Programmable Active Memories: Reconfigurable Systems Come of Age", *IEEE Transactions on VLSI Systems*, 1995.

[22] *The Programmable Gate Array Data Book*, San Jose, CA: Xilinx, Inc., 1992.