International Conference on Computer Design, pp. 170-177, 1994.

Mesh Routing Topologies For Multi-FPGA Systems

Scott Hauck, Gaetano Borriello, Carl Ebeling
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

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. Tree, Bipartite Graph, 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

In the time since they were introduced, FPGAs have moved from being viewed simply as a method of implementing random logic in circuit boards to being a flexible implementation medium for many types of systems. Logic emulation tasks, in which ASIC designs are simulated on large FPGA-based structures [Butts92], have greatly increased simulation speeds. Software subroutines have been hand-optimized to FPGAs to speed up inner loops of programs [Bertin93], and work has been done to automate this process [Wazlowski93]. FPGA-based circuit implementation boards have been built for easier project construction in electronics education [Chan92]. An important aspect shared by all of these systems is that they do not use single FPGAs, but harness multiple FPGAs, preconnected in a fixed routing structure, to perform their tasks. While FPGAs themselves 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. Field-Programmable Interconnects (FPICS)[Aptix93], crossbar chips that can perform arbitrary routing between their pins, may remove some of the topology concerns from small arrays. However, large FPGA systems with FPICs for routing will still need to fix the topology for inter-FPIC routing.

Several routing hierarchies have been used in recent FPGA-based systems. A Mesh structure, where the FPGAs are laid out in a two-dimensional grid, with an FPGA connected only to its four nearest neighbors, is employed in the PAM [Bertin93] software accelerator. A board being developed at MIT incorporates Mesh connections with routing similar to Superpins (described later in this paper)[Tessier93]. The Quickturn RPM [Butts92] logic emulator uses a hierarchy of partial crossbars, or Bipartite Graphs (However, the hierarchy is actually similar to a Tree, making the RPM a Tree of Bipartite Graphs). The BORG board [Chan92], designed for use in electronics education, uses a Bipartite Graph, where half the FPGAs are used for logic, and half for routing only.

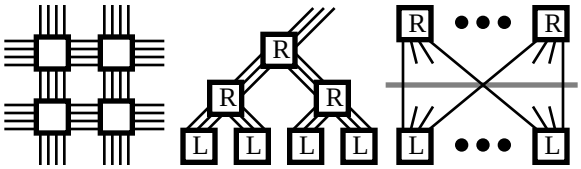

Figure 1. Some proposed FPGA routing structures. Mesh (left), Tree (center), Bipartite Graph (right). FPGAs labelled "L" are used only for logic, labelled "R" are used only for routing, and unlabelled are used for both logic & routing

The routing structure used to interconnect individual chips in an FPGA array has a large impact not only on system speed, but also on required FPGA area, and system extendibility. A Bipartite Graph, with half of its FPGAs used for routing only, provides a predictable routing delay between logic FPGAs, since every inter-FPGA signal moves through exactly one routing FPGA. However, it sacrifices scalability and chip utilization. Tree structures have less predictable routing delays, since signals may have to pass through many FPGAs, but have improved scalability. Mesh connections are scalable, and may have better utilization than the other structures, but have even worse routing predictability. Although Mesh topologies seem to be falling out of favor due to perceived pin limitations, new techniques such as Virtual Wires [Babb93] and future high-I/O FPGA packages make Meshes a very viable alternative.

Determining the proper routing topology for an FPGA array is a complex problem. In fact, the multiprocessor community has been struggling with similar questions for

years, debating the best interconnection topology for their routing networks. The necessary first step is to determine the best way to use a given routing topology, so that the best interconnect structure for each topology can be compared to determine an overall best. 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.

Mesh Routing Structures

The obvious structure to use for a Mesh topology is a 4-way interconnection, with an FPGA connected to its direct neighbors to the north, south, east and west, and all the pins on the north side of an FPGA are connected to the south edge of the FPGA directly to the north (the east edge is connected similarly). This is shown in figure 1 left. 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 both I/O pin and internal FPGA routing resources. The 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.

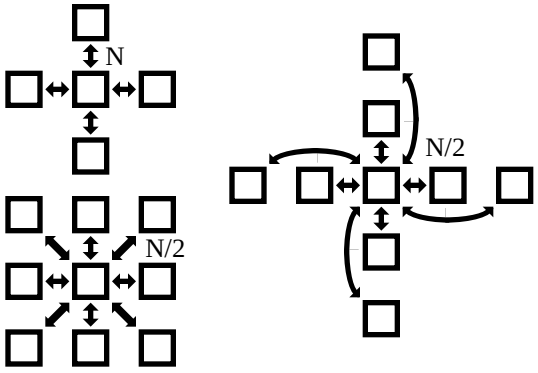

Figure 2. 4-way (top left), 8-way (bottom left), and 1-hop (right) Mesh interconnection patterns.

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 FPGAs directly adjacent, as well as those one step removed, vertically and horizontally are connected together. 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, both of these issues are beyond the scope of this paper.

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.

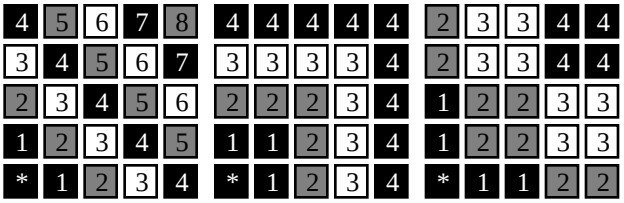

Figure 3. Distance (in number of I/O pins required) to reach FPGAs in 4-way (left), 8-way (center), and 1-hop (right) topologies. Distances are from the FPGAs in the lower-left corners.

Switching from a 4-way to an 8-way or 1-hop topology has two major impacts: 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, and the number indicating how many I/O pins are required to reach that FPGA from the source FPGA at the lower-left corner (shown with an asterisk). As we can see, both the 8-way and 1-hop topologies can reach more FPGAs within a given number of I/O pins 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 can within a given number of I/O pins, and the 1-hop topology can reach three times as many FPGAs as a 4-way topology (while within a single I/O pin there are only twice as many neighbors as the 4-way, at longer distances it reaches three times or more). 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 pins. The only exception 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.

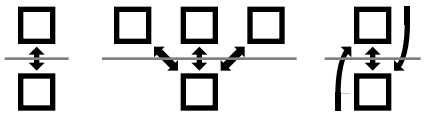

Figure 4. Bisection bandwidth in 4-way (left), 8-way (center), and 1-hop (right) topologies. Three times as many connections cross the bisecting line in the 8-way and 1-hop topologies than in the 4-way topology, though each connection contains half as many wires. This results in a 50% increase in bisection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

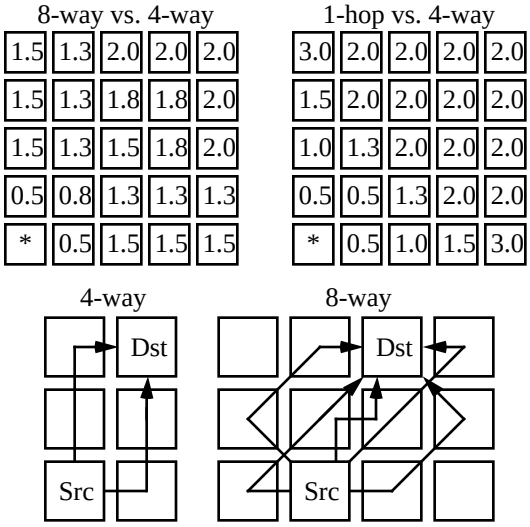

Figure 5. Example of the point-to-point fast bandwidth calculation in 4-way (left) and 8-way (left center) Meshes. The 8-way routes are only allowed to use three I/O pins, the number of I/O pins necessary to reach the destination in the 4-way topology. Also included is a complete relative bandwidth summary of 8-way vs. 4-way (right center) and 1-hop vs. 4-way (right) topologies.

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. This means splitting the network into two equal groups such that the amount of bandwidth going from one group to the other is minimized. 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 4, cutting each column or row 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 between them (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. If we simply ask how much bandwidth 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 pins, 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. 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 pin 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 pins away than the other topologies. However, if we allow each topology to use the same number of I/O pins (specifically, the minimum number of I/O pins 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 5 left, if we route to an FPGA two steps north and one step east, it requires three I/O pins 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 pins, it actually has five independent paths between the two FPGAs (figure 5 left center). 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 5 right 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 (right center), and 1-hop vs. 4-way fast bandwidth (far 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 two or more.

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 twice as great or more.

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.

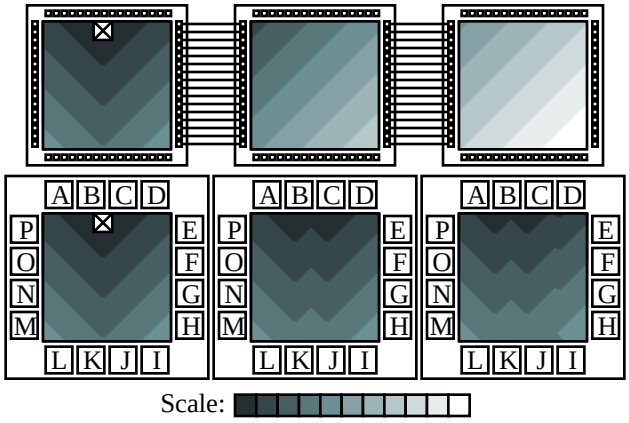

Figure 6. Distances from the X in leftmost FPGA in normal (top) and Superpin (bottom) connection pattern, on a scale ranging from black (shortest) to white (longest) shown at bottom. Superpin connections are given by letters, with Superpins with the same letter connected together in adjacent FPGAs.

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 6 top. Because the pins connecting an FPGA to its neighbor to the east are on the opposite FPGA edge 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 6 top, moving from the top of 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 two 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. 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 6 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 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 7 top, simply making the connection labeled "B" gives most of the benefit of the complete unpermuted Superpin pattern given in figure 6. Thus, connecting "C" as we did in figure 6 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 7 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 7 bottom). Note that by having two (identical) permutations in series in figure 7 bottom, we in fact use less 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 7 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.

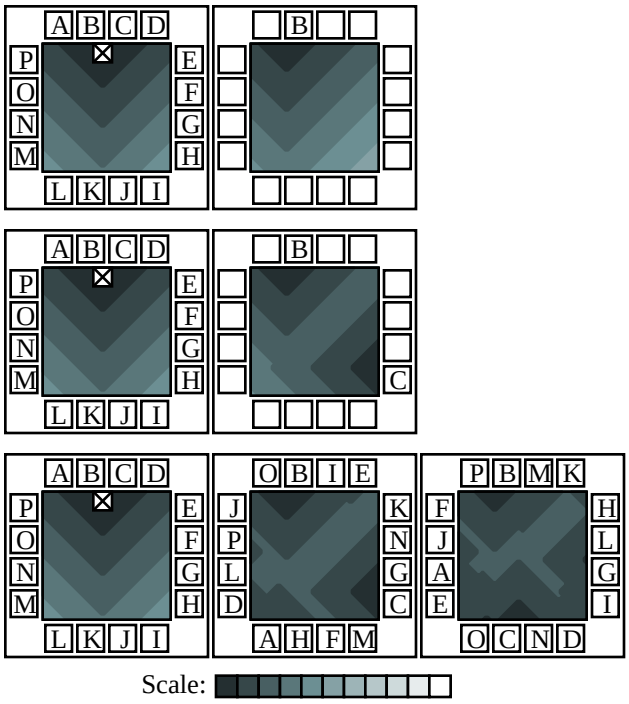

Figure 7. Intuition behind permuted Superpins. A single connection (at top) gives most of the benefit of full unpermuted Superpins. By changing connection C to lower-right corner (middle), more short routes are achieved. Bottom shows full permuted Superpins, with even shorter routes in further FPGAs. Scale ranges from black (shortest) to white (longest) shown at bottom.

It is possible to generate even better topologies by using several different permutations in a single system. As described in [Hauck94a], having different permutations in different directions (i.e. one for paths leading east, another for paths leading south) takes advantage of reconvergent paths (i.e. the path leading east then south arrives at the same FPGA as the path leading south then east), having the short routes along one path leading to different locations than the short routes along another path. However, in our experience several different permutations yield at most a 1% improvement in routing costs, and some of these benefits would probably not be realizable by automatic mapping software.

In [Hauck94a] we presented 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 loose 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 FPGA array, 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 8. These graphs represent the average and maximum resource usages from every point in a source FPGA (FPGAs are represented by grids as described earlier, with 36 pins on a side) to every point in the nearest N neighbor FPGAs in the system. 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 usages by about 28%. One reason 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 at least use enough routing resources to go from the 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 inter-FPGA routing delays. As shown in figure 9, 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 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. Note that this approximates possibly

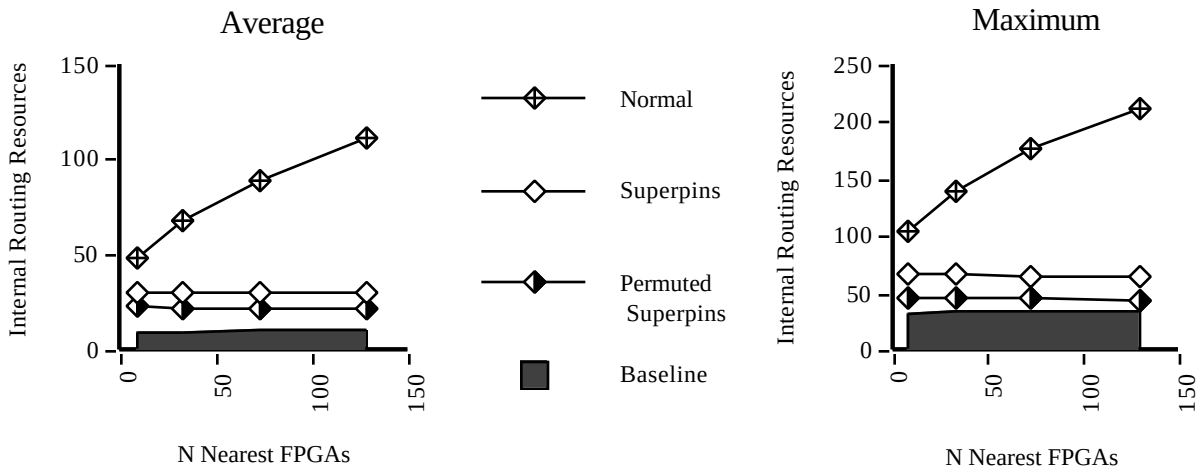

Figure 8. Average (left) and maximum (right) internal routing resource usage from each location in source FPGA to all locations in N nearest destination FPGAs in a 1-hop topology.

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 a good 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 must in fact deal with many signals fighting for the same resources. To measure this resource conflict, we have used the router we developed for the Triptych FPGA project [McMurchie94], 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. While real FPGAs usually do allow multiple signals to move through an area one pin width on a side, which corresponds to an edge in the FPGA grid, there might be conflicts our model will not represent. 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 10) 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

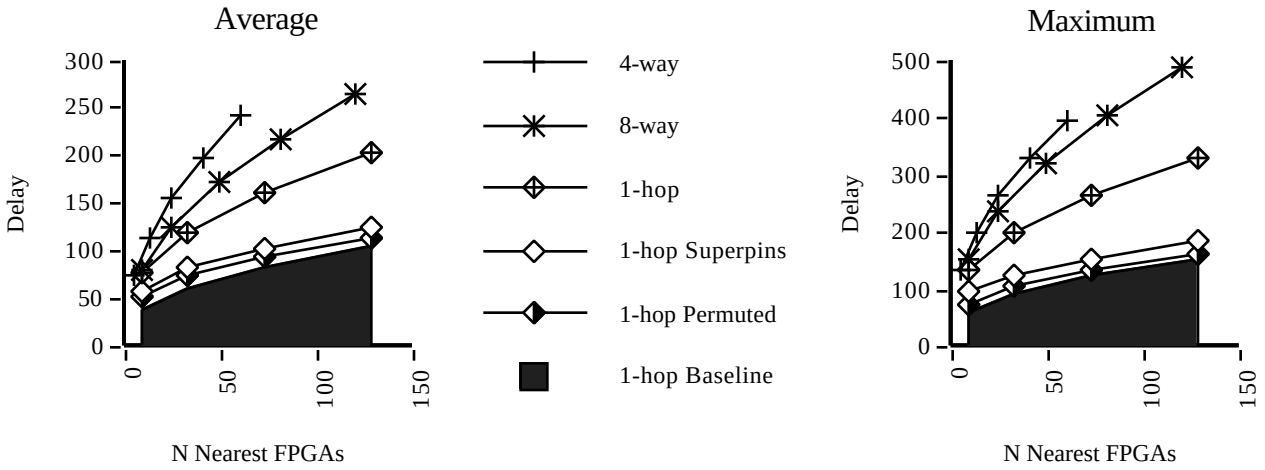

Figure 9. Average (left) and maximum (right) delay from each location in source FPGA to all locations in N nearest destination FPGAs.

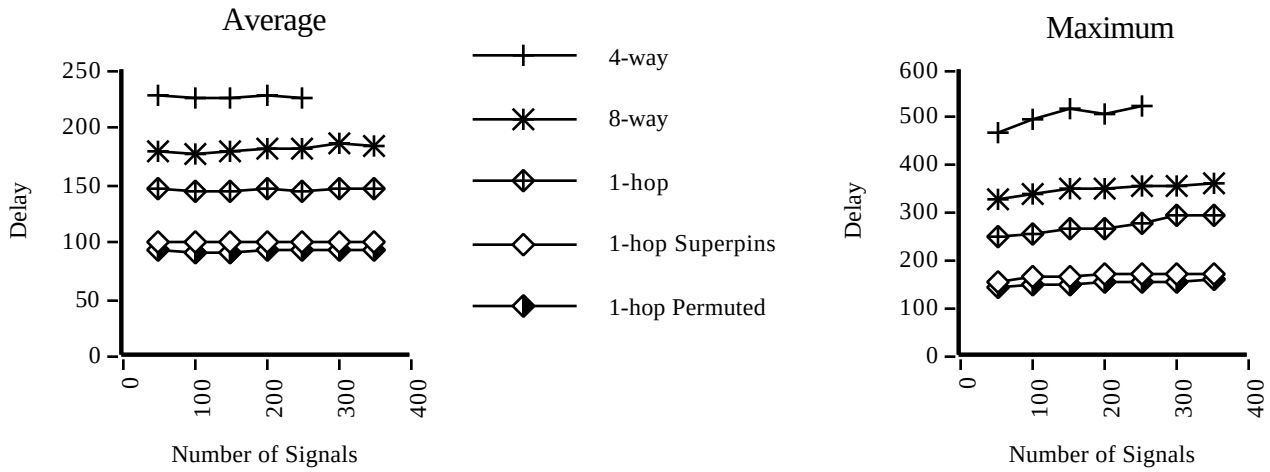

Figure 10. Graphs of average and maximum distance of signals under several topologies in a 5x5 array.

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 shown. Trials were run at 50 signal increments until the router failed to find routes for all signals.

There is a question of how well some of the topologies handle 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 graphs 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 may be 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 probably 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 9) 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

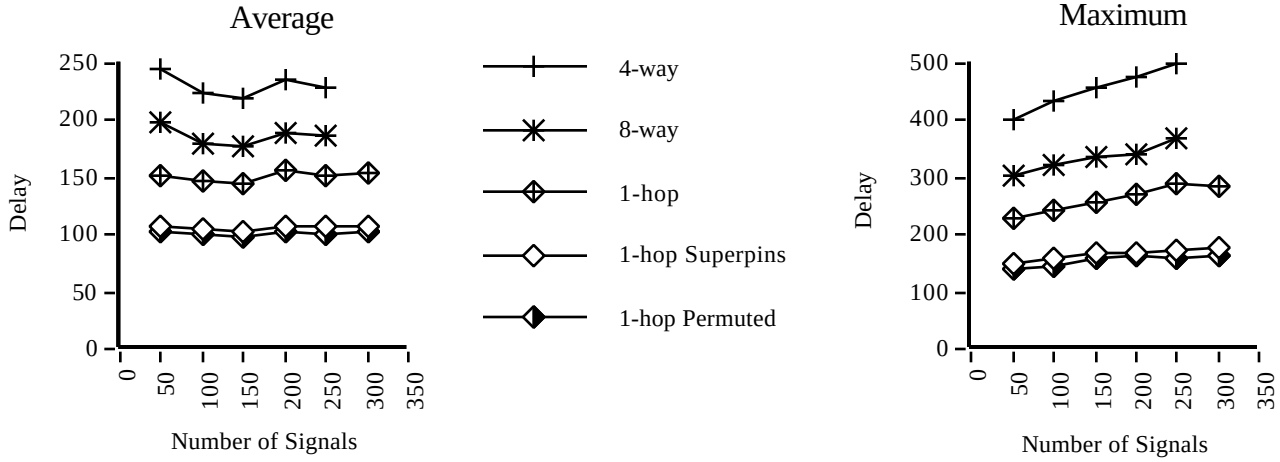

Figure 11. Same as figure 10, but with all signals in 5-signal busses.

bisection bandwidth.

Automatic Mapping Tools

Since most FPGA arrays 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 that it does 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. We have developed a pin assignment tool (similar to a detailed router) for inter-FPGA routing [Hauck94b], and the only impact of the improved topologies on this tool is the loss of a slight speed optimization opportunity. Decomposition and partitioning tools are also easily adapted, since the locality needed for Mesh circuits 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.

Conclusions and Future Work

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 8) prove that there is little room for improvement. Real improvements might come from increasing the direct neighbors of an FPGA from 8 to 26 (a 3-D 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, Bipartite graphs, 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. Also, good retargetable partitioners capable of doing a reasonable optimization for very varied topologies, as well as a good benchmark set for multi-FPGA systems, are necessary for this effort.

Acknowledgments

This paper has benefited from the patience of several people, most particularly Elizabeth Walkup, Donald Chinn, Soha Hassoun, 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. Scott Hauck was support by an AT&T Bell Laboratories Fellowship.

References

[Aptix93] Aptix Data Book, Aptix Corporation, San Jose, CA, 1993.

[Babb93] 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.

[Bertin93] P. Bertin, D. Roncin, J. Vuillemin, "Programmable Active Memories: a Performance Assessment", *Research on Integrated Systems: Proceedings of the 1993 Symposium*, pp.88-102, 1993.

[Butts92] M. Butts, J. Batcheller, J. Varghese, "An Efficient Logic Emulation System", *Proceedings of ICCD*, pp. 138-141, 1992.

[Chan93] 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.

[Chan92] P. K. Chan, M. Schlag, M. Martin, "BORG: A Reconfigurable Prototyping Board Using Field-Programmable Gate Arrays", *Proceedings of the 1st International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, pp. 47-51, 1992.

[Hauck94a] S. Hauck, G. Borriello, C. Ebeling, "Mesh Routing Topologies for FPGA Arrays", *2nd International Workshop on Field-Programmable Gate Arrays*, Berkeley, 1994.

[Hauck94b] S. Hauck, G. Borriello, C. Ebeling, "Pin Assignment for Multi-FPGA Systems", *1994 IEEE Workshop on FPGAs for Custom Computing Machines*, 1994.

[McMurchie94] L. McMurchie, C. Ebeling, G. Borriello, "An Iterative, Performace-Driven Router for FPGAs", University of Washington, Dept. of Computer Science & Engineering Internal Report, 1994.

[Tessier94] R. Tessier, J. Babb, M. Dahl, S. Hanono, A. Agarwal, "The Virtual Wire Emulation System: A Gate-Efficient ASIC Prototyping Environment", FPGA '94, Berkeley, 1994.

[Wazlowski93] M. Wazlowski, L. Agarwal, T. Lee, A. Smith, E. Lam, P. Athanas, H. Silverman, S. Ghosh, "PRISM-II Compiler and Architecture", *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 9-16, 1993.