ACM/SIGDA 2nd International Workshop on Field-Programmable Gate Arrays, Berkeley, February, 1994.

Mesh Routing Topologies For FPGA Arrays

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 routing costs by more than 50% 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, Chan93]. 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], chips that 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.

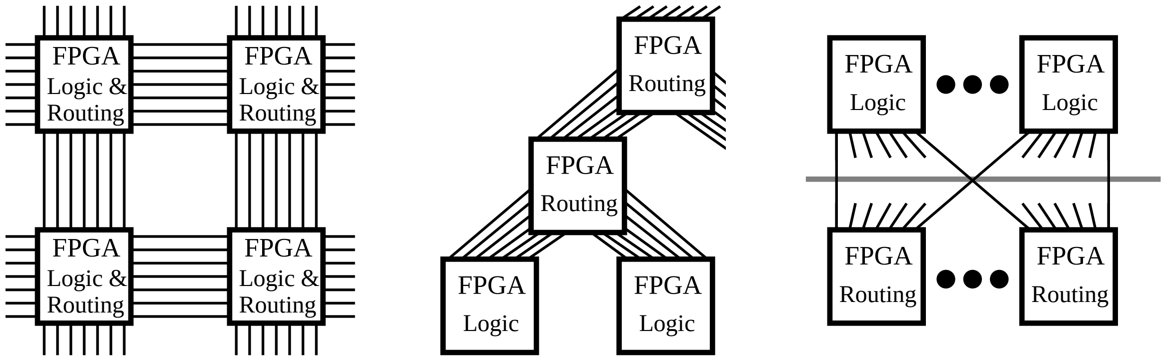


Figure 1. Some proposed FPGA routing structures. Mesh (left), Tree (center), Bipartite Graph (right).

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 heirarchy is actually similar to a Tree, making the RPM a Tree of Bipartite Graphs). The BORG board [Chan92, Chan93], designed for use in electronics education, uses a Bipartite Graph where half the FPGAs are used for logic, and half for routing only.

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] 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 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

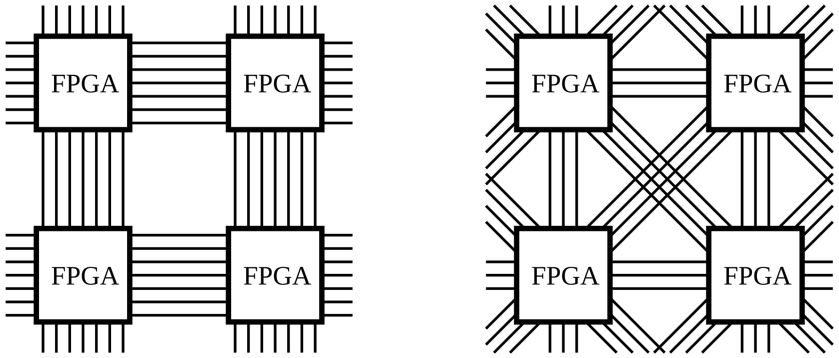


Figure 2. 4-way (left) and 8-way (right) Mesh interconnections.

The obvious structure to use for a Mesh topology is a 4-way interconnection, where all the pins on the north side of an FPGA are connected to the south edge of the FPGA directly to the north. 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 such a system, the inter-chip routing incurs both a delay cost, affecting the speed of the entire system, and an area cost, affecting the amount of FPGA area required to map the resulting circuits. In both area and delay, the cost function includes a penalty for each chip IO, and a penalty for the distance moved inside each FPGA. An overall cost function for both area and delay in a 4-way Mesh is given in equation 1, where n is the number of FPGAs away a sink is from the source of a signal (i.e. n=1 means the destination is adjacent, n=2 means there is one intervening FPGA). The last terms $2*(FPGAwidth/2)$ represent the routing necessary inside the source and destination FPGAs. In the rest of this section we will introduce several modifications to the 4-way Mesh structure which will decrease most of the terms in this equation. One important note is that the terms in the equation are not independent. Specifically, by decreasing the amount of in-chip routing resources used for inter-chip routing, we can map circuits more densely, meaning that the average n will decrease. However, since this simply means that any improvements we make may have an even larger gain than we predict, we can safely ignore this fact in the following discussion.

Eqn 1. $Cost = (ChipIOCost)*n + (RoutingCost)*\left[(n-1)*(FPGAwidth) + 2*(FPGAwidth/2)\right]$

The simplest improvement to make is to connect an FPGA not only with its 4 nearest neighbors, but also with the 4 diagonal neighbors as well. This serves to decrease the distance between FPGAs, since a route that uses a single diagonal pin connection in an 8-way Mesh usually requires both a horizontal and a vertical pin connection in a 4-way Mesh. We will assume here that we use the same number of wires to connect an FPGA to each of its neighbors, though a topology could easily give more or less bandwidth to diagonal neighbors. While the symmetric 8-way Mesh halves the bandwidth to any particular direct neighbor, it doubles the number of nearest neighbors. For routes that are more than one step away from the source FPGA, an 8-way connection pattern provides more possible paths, and can even increase bandwidth. For example, to move two steps north in a 4-way Mesh, a minimum path can only move directly north. In contrast, in an 8-way Mesh the route could first move north, northeast, or northwest, and still take a minimum number of steps. In this case, there is 50% more potential bandwidth between these two FPGAs in the 8-way Mesh, since there are three times as many independent paths, with half as much bandwidth on each path.

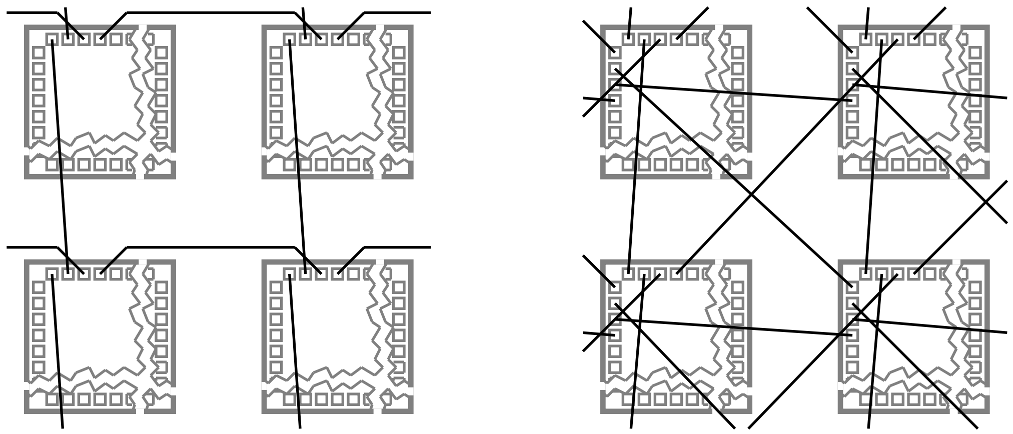


Figure 3. A single Superpin in a 4-way (left) and 8-way (right) Mesh. The 4-way portion, starting at the upper left corner and moving clockwise, is (S, N, W, E). The 8-way portion is (W, E, SE, NW, N, S, SW, NE).

A problem that both the simple 4-way and 8-way Mesh connections have is that in order to move between non-adjacent FPGAs, a route must cross the width or height of the intervening FPGAs. This is because all of the connections to an FPGA's northern neighbor are clustered in a group along the FPGA's north edge, while the connections to the southern neighbor are clustered on the south edge. An alternative to this is to intermingle the connections, making sure that a signal entering on a given pin will have a nearby pin exiting in the direction it wishes to go. In a 4-way Mesh, a signal entering from a given direction might want to leave in any other direction, though most often in the opposite direction. Thus, when we look at the destinations of pins around the FPGA's edge (moving clockwise), a pattern of (S, N, W, E, S, N, W, E...) helps makes sure that a signal moving between distant FPGAs will have to use very little routing resources on the intermediate FPGAs. The 8-way Mesh is somewhat more complex, because a signal entering from a given direction will usually want to exit along one of the three most opposite directions (for example a signal entering from the north will usually want to exit from the south, southeast, or southwest), though most likely through the direction exactly opposite. Again looking clockwise around the FPGA, the pattern (N, S, SW, NE, E, W, NW, SE, S, N, NE, SW, W, E, SE, NW...) ensures that the desired direction is at most 2 pins away, and the opposite direction is 1 pin away. One interesting thing to note with both of these patterns is that the pins come in groups of adjacent pins (4 pins in a 4-way Mesh, 8 in an 8-way) where each direction is represented exactly once. In the following discussion, we will break the chip pins into these groups, and refer to them as *Superpins*. Also, since the distance between pins inside a Superpin is small (only 1 or 2 steps between pins that are likely to be connected in an 8-way Mesh), but to move to another Superpin is much longer (5-11 steps between pins likely to be connected in an 8-way Mesh), we will measure distances in terms of Superpins. A route that stays within a Superpin, either on a terminal or intermediate FPGA, has 0 cost, while a route n Superpins away incurs a cost of n. This will simplify some of the following discussion, though we will not use this assumption in the quantitative comparisons, where it might skew the results.

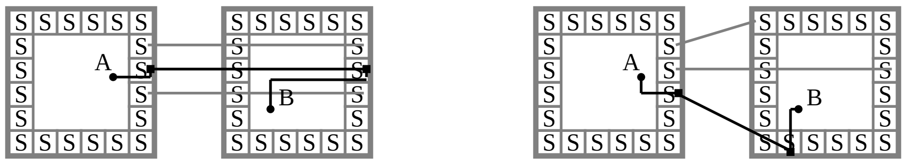


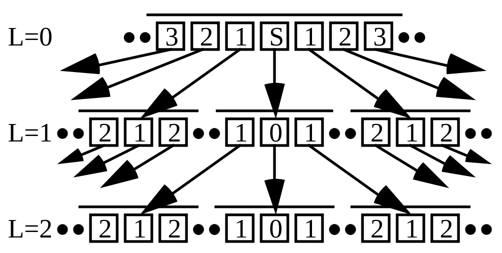
Figure 4. Example of Superpin permutation. The straightforward Superpin connections (some shown in gray at left), and a portion of a permutation at right.

The next important observation is that in the simple Superpin structure described above, pins in an FPGA's nth Superpin were only connected to the neighboring FPGAs' nth Superpins. This is shown at left in figure 4. As can be seen, all Superpins close together in one FPGA connect to Superpins that are close together in adjacent FPGAs. Thus, there are many paths leading from a point in one FPGA to about the same place in the next FPGA. There is a lot of redundancy in this scheme, which sacrifices some optimization opportunities. For example, if the nearest Superpin to a signal takes it to the east edge of a neighbor, then the two next closest Superpins could go to the northwest and southwest corners of the neighboring FPGA (see figure 4 right). Thus, a signal that must go to the west edge of the neighboring FPGA could move a short distance on the local FPGA instead of a much longer route on the destination FPGA, reducing overall routing costs. A good permutation of Superpin connections might also allow short steps distributed across several FPGAs in series to replace any large movements required in the destination FPGA of a long route. Thus, the greater the number of intervening FPGAs, the greater the number of different short paths open to a signal, resulting in ever smaller amounts of internal FPGA routing needed in longer paths.

As implied in the earlier paragraph, a “permutation” of Superpins is simply a connection of Superpins between adjacent FPGAs. One way to think about this is that a permutation forms a one-to-one relation between Superpins on adjacent FPGAs. If we number FPGA Superpins clockwise from the upper-left corner, then the permutation which connects Superpin i in one FPGA to Superpin i in an adjacent FPGA would correspond to the Superpin connection scheme at left in figure 4. A better permutation would have adjacent Superpins in one FPGA connected to non-adjacent Superpins in the other FPGA.

Before we discuss how to construct good pin permutations, it is helpful to develop some bounds on how good a permutation can be. We rate permutations based on the average of the minimum distances of each Superpin on the local FPGA to each Superpin on a destination FPGA, measured in Superpin steps. We do not add any cost for traversing pins, since it is assumed that we will always make the minimum number of chip crossings, and since the permutations do not affect this crossing number. For example, two connected FPGAs with two Superpins each would have an average cost of 0.5, since half of the Superpin cross-product pairs are directly connected (0 Superpin steps away), and the other half are one step away. Note that where the Superpin steps are taken does not matter to our goodness metric. For example, if the best route between a location on FPGA A to a location on FPGA B traverses FPGA C, and uses 2 Superpin steps on A, 1 on B, and 3 on C, the overall distance is considered to be 6.

We can generate a lower bound on how good a permutation can be by assuming each FPGA has an infinite number of Superpins. We then determine how many Superpins could possibly be reached in exactly N Superpin steps, given that we are moving through L-1 intermediate FPGAs. L=0 indicates that we remain on the source FPGA. We use an FPGA with an infinite number of Superpins because in a finite FPGA, reconverging paths may cause less pins to be reached in a certain distance than is possible in theory. An example of this entire process is in figure 5. Note that we assume that all Superpin steps are taken along the outside edge of the FPGA. While this ignores the fact that the opposite edge of the chip can be reached more quickly by moving directly across the chip than by moving around the outside, these paths will never be minimal in the permutations we use, and thus can be safely ignored. We can generate a formula for the resulting function by realizing that to reach a Superpin in exactly N steps, you must reach a Superpin in the previous FPGA in i steps, $0 \leq i \leq N$, and then move across to the destination FPGA. Since a Superpin in the previous FPGA reached in i < N steps leads to two N-step neighbors (one clockwise, the other counterclockwise), but a Superpin with i = N leads only to one N-step Superpin (the one directly connected to it), we have the formula $F(N,L) = F(N,L-1) + 2\sum_{i=0}^{N-1} F(i,L-1)$. Note that this is equivalent to $F(N,L) = F(N,L-1) + F(N-1,L-1) + F(N-1,L)$. The boundary cases are $F(0, L) = 1$, $F(j,0) = 2 | j > 0$.



L \ N	0	1	2	3	4	5	6
0	1	2	2	2	2	2	2
1	1	4	8	12	16	20	24
2	1	6	18	38	66	102	146
3	1	8	32	88	192	360	608
4	1	10	50	170	450	1,002	1,970
5	1	12	72	292	912	2,364	5,336

Figure 5. Diagram of the lower bound calculation (left), and a table of the values (right). The numbers in the pin locations at left indicate the distance of the Superpin from the source S, with the arrows indicating direct connections. These pins form an exponentially expanding fanout tree of possible short paths.

To determine the lower bound for a permutation on FPGAs with finite numbers of Superpins, we know that no permutation can reach more pins in N steps than the formula given above, and may reach less due to reconverging paths. Thus, for an FPGA with M Superpins, we get a lower bound by assuming that the first F(0, L) pins are reached in 0 steps, F(1, L) are reached in 1 step, and so on until all pins are assigned. The weighted average of these distances is the optimum value for Superpin permutations for that array size. Note that for specific permutations we would have to calculate distances from each source Superpin to each destination Superpin, but since all source pins will have the same optimum distances, the average from one source pin is identical to the average from all source pins. For a specific example, the optimums for an FPGA with 18 Superpins is 4.5 for L=0, 1.944 for L=1, and 1.556 for L=2. While this lower bound is not tight for single permutations (provable by brute force search for an 8 Superpin FPGA, where no single permutation is optimum for both L=1 and L=2, though optimums for each separately exist), in our experience permutations exist that either equal or come within a few percent of this value.

While the previous discussion gives a lower bound along a single path of permutations, in order to extend them to a two-dimensional Mesh there are a couple observations to be made. First of all, a permutation must not only work well for signals going from a source FPGA A to a destination B, but also for the reverse route from B to A. However, it turns out that the inverse of a permutation (the permutation seen by a route moving backwards through the original permutation) is exactly as good as the original permutation. This is due to the fact that the measure of a permutation is the average distance from all sources to all sinks, which is identical to the average distance from all sinks to all sources, which is the measure of the inverse's goodness.

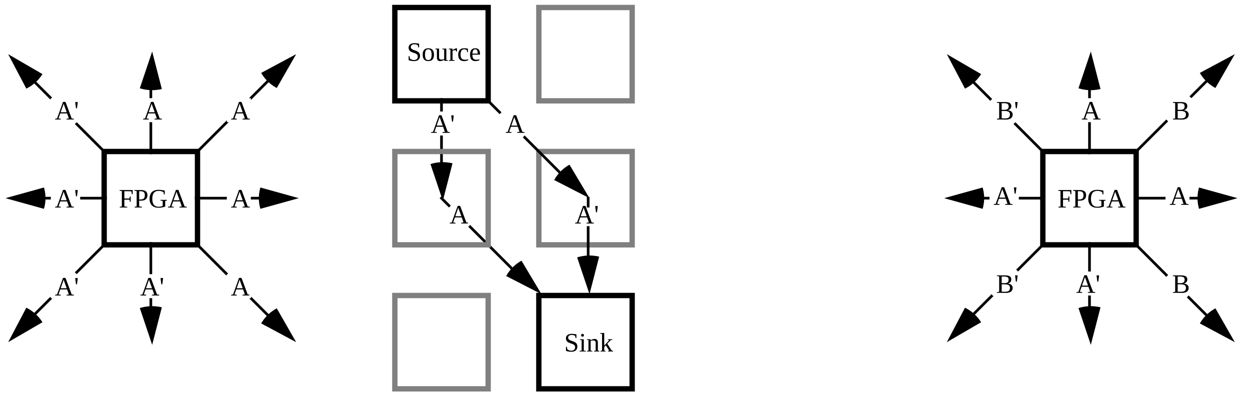


Figure 6. Permutations in 8-way Meshes. The pattern at left, with a single permutation A, and its inverse A', causes some paths to use both the permutation and the inverse, as shown at center. The pattern at right, with permutations A and B, avoids this problem. The two permutations are chosen to be independent, so that permutation A and inverse A' work well with both permutation B and inverse B', and vice-versa.

The next issue is that in a two-dimensional Mesh, paths do not necessarily travel in a single direction, but may be required to change direction to reach their destinations. However, as illustrated in figure 6, using a single permutation in a Mesh may make a route pass through both a permutation and its inverse. Note that this doesn't result in a total cancellation of the permutation's benefit, since a permutation followed by its inverse has at least the benefit of the inverse permutation. This can be seen by realizing that any signal could just take direct connections through the first permutation, without sideways steps in the source FPGA, and then take the minimal path through the inverse permutation, allowing sideways steps in the middle and end FPGAs. Because we do not penalize for crossing chip boundaries, the average distance through a permutation and its inverse is thus at most the average distance through the inverse permutation only. There exists a single-permutation topology that avoids the problem of paths with both a permutation and its inverse, but requires different routing at different locations (i.e. while the permutation leading south from one FPGA may be A, another FPGA might have A' leading south). Two permutations, one for the horizontal and vertical moves, and another for the diagonals, can also fix the inversion problem while keeping the same pattern in every FPGA (see figure 6 right).

The final observation is that for some destinations, there is more than one path between two FPGAs that moves through the minimum number of intermediate FPGAs. For example, to move two FPGAs north, a route can move through either the FPGA directly to the north, northeast, or northwest. We can use this fact to our advantage by choosing combinations of permutations such that the shorter routes through one intermediate FPGA lead to where longer routes through other intermediate FPGAs end. Thus, every destination will have a short path through some intermediate FPGA, yielding a better set of permutations overall. In this way, if there are P paths between FPGAs, then there could conceivably be P times as many pins reached in N steps as predicted by the above lower bound. Note that in this case, it would actually be advantageous to have different permutations on the two diagonals leaving an FPGA. This is so that a path leading northwest then northeast would have a different permutation order from a path leading northeast then northwest. Three permutations are sufficient, because no minimal path will move by both a horizontal (East or West) and a vertical (North or South) edge, since a single diagonal step would replace these two steps.

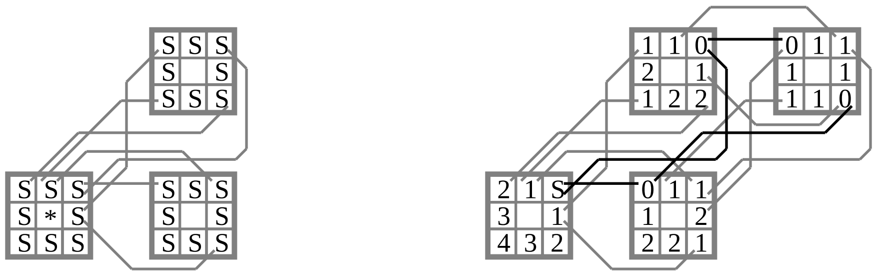


Figure 7. Example of multiple paths in a two-dimensional permuted Mesh decreasing routing costs. A portion of the permutations leading northeast and east from the asterixed FPGA is shown at left. Each FPGA has 8 Superpins. At right is the resulting distances from the upper right Superpin (labeled "S") in the lower left FPGA. Notice that in the FPGA at top right there are two different Superpins 0 steps away. This is because the paths leading through different permutation orders lead to different points. These paths are in black.

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, and has consistently found permutations within a few percent of the lower bounds given earlier. 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.

Comparisons

The best method for determining how much of an impact the constructs introduced in this paper have would be to develop an integrated toolset for partitioning, pin assignment, placement and routing, optimized for each of the topologies. Then, by mapping a set of circuits representative of those expected in the target system, we could exactly compare the different structures. Unfortunately, not all of the necessary tools are available, and there is no good, generally accepted set of benchmarks for multiple-FPGA systems.

Our approach to topology comparison is twofold. The first method is to determine how long the expected inter-chip routes are for individual signals under the different topologies. This data is presented in figure 8. Here, we determine both the average and the maximum distance for every location in a source FPGA to every location in the nearest N destination FPGAs, where N does not include the source FPGA. The system is assumed to be in an infinite plane of FPGAs to avoid edge effects, which should not be significant. The individual FPGAs are loosely based on the Xilinx 3090-125. Specifically, there are 36 pins on each side of the FPGA, and we assume that the internal FPGA routing resources obey a linear Manhattan cost metric, that the sum of the distances traveled in the X and the Y dimension is directly proportional to the cost. When assigning costs both to a step within an FPGA, and to a chip crossing, we must ensure that the ratio between the two is reasonable. For these experiments a chip crossing costs 30 times more than an internal FPGA step, which is similar to the delays encountered in a Xilinx 3090-125.

There are several observations to be made. First, although switching from a 4-way to an 8-way Mesh improves both metrics, using Superpins has by far the greatest effect. In fact, the cumulative averages for 4- and 8-way Meshes without Superpins are in most cases greater than the maximums of the same Meshes with Superpins. Rather surprising is how little benefit the permutations give. The reason for this is not so much that the permutations are ineffective, but that the Superpins are very effective. To illustrate this, we include a baseline on the graph which is a lower bound on all possible 8-way Mesh connection schemes. It is based on the fact that under any 8-way Mesh connection scheme, a path moving M FPGAs away must at least move to the closest pin on the source FPGA, cross M-1 FPGA boundaries (taking 1 step inside each of the intermediate FPGAs), and move between the destination and the pin nearest to it. As the graph shows, the cumulative average of an 8-way Mesh with Superpins is always within 21 internal FPGA steps of the optimum. The permutations do manage to reduce this difference in cumulative average to between 10 and 15 steps. As the graphs in figure 9 show, this results in about a 25% decrease in total internal routing resource usage. Also, although earlier we discussed systems with one, two, and three permutations, only one line is included in the graphs for all permutations. This is because the results for the different cases are virtually indistinguishable, differing by at most a percentage point.

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, 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. For our experiments, 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.

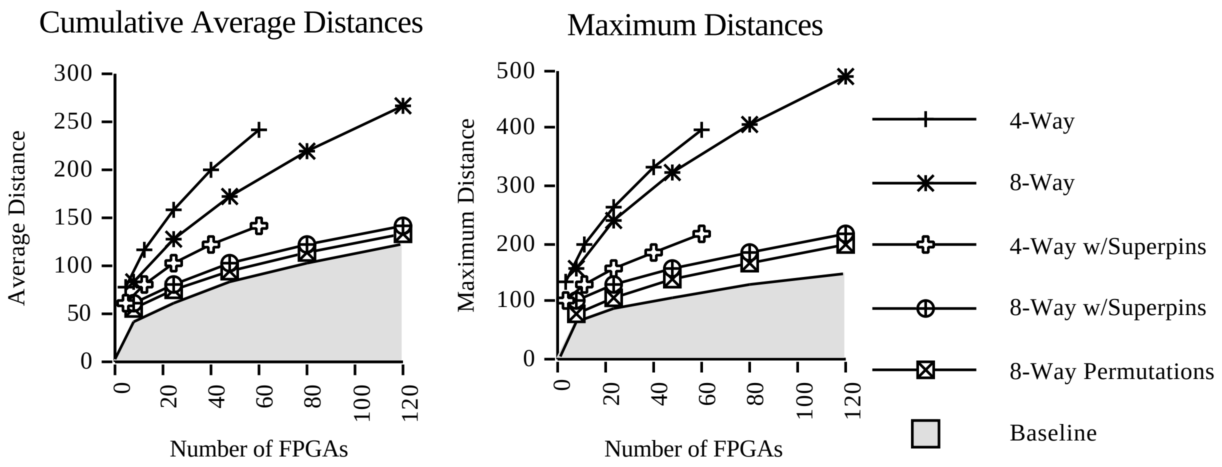


Figure 8. Graphs of the cumulative average and maximum distances of individual signals under several topologies.

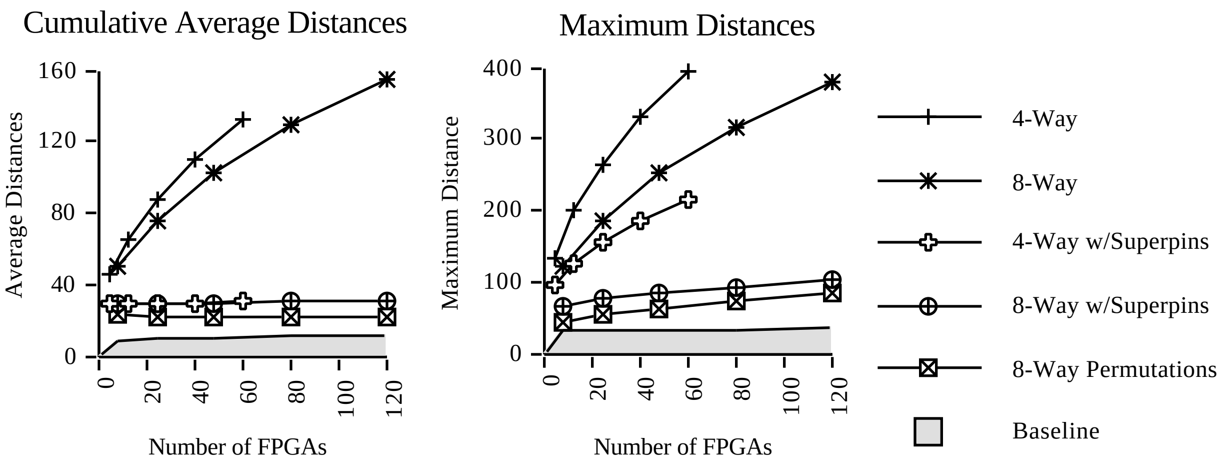


Figure 9. Same as figure 8, but with all pin costs removed.

The first two graphs 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 several trials are averaged together to generate the curves shown. To model the signal locality one hopes to exploit when mapping circuits to a Mesh, a second set of graphs perform the same comparison as the previous graphs, but with the signal lengths biased towards shorter distances. Half of the routes are to adjacent FPGAs, one quarter to FPGAs one step apart, and one quarter are two or three steps apart. Note that this locality is based on an 8-way connection pattern, and although 4-way patterns will actually thus see less locality, this is a reflection of a 4-way pattern's lesser number of neighbors. Finally, 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 third 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.

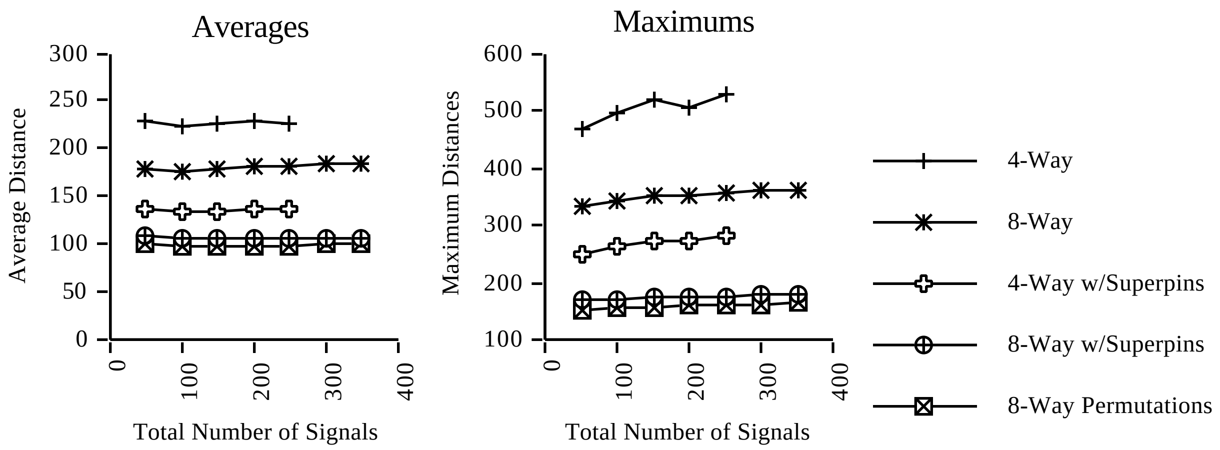


Figure 10. Graphs of average and maximum distance of signals under several topologies in a 5x5 array.

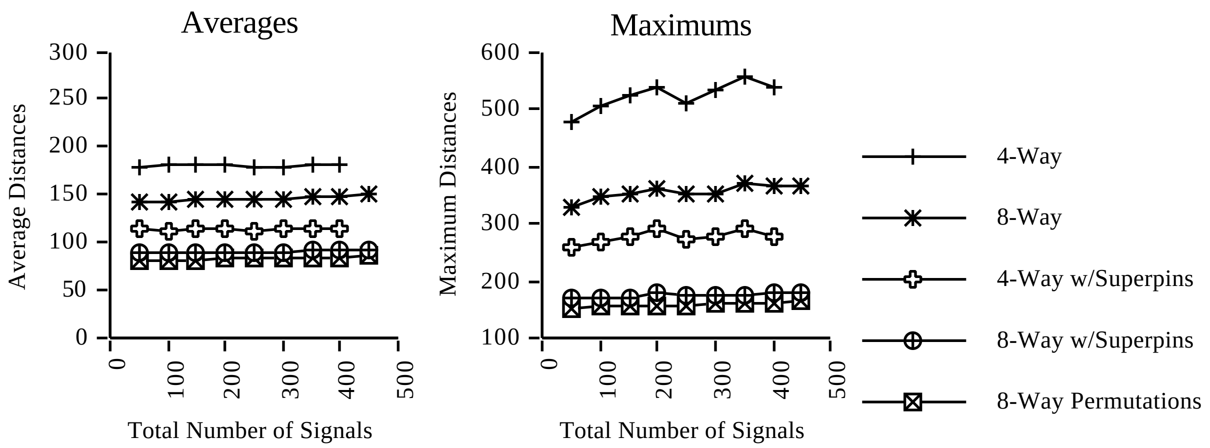


Figure 11. Same as figure 10, but with signal locality.

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 cumulative averages 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 graph (figure 8) are born out in actual routing experiments, with Superpins yielding about a 40% decrease in routing cost, while moving from 4-way to 8-way interconnections is only a 20% improvement. Permuting these Superpin signals is still beneficial, by about 7%. These factors are almost identical to those predicted in figure 8.

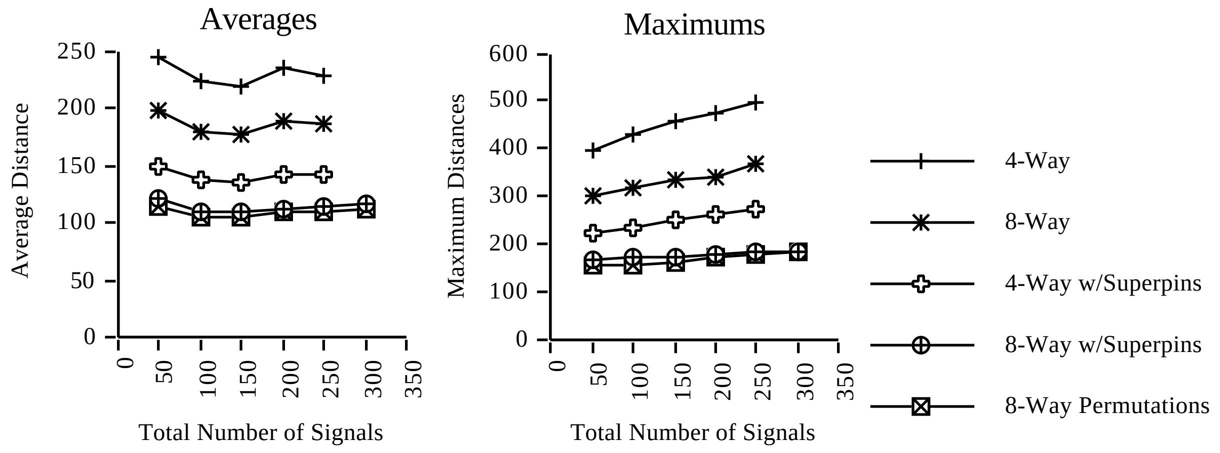


Figure 12. Same as figure 10, but with all signals in 5-signal busses.

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 comparison 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. 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. Placements inside individual FPGAs is the last concern. In the simple 4-way Mesh, to obtain a good placement we primarily seek to put logic connected to signals from FPGAs to the north along the north edge of the FPGA, connections to the south along the south edge, east on the east, and west on the west. A secondary concern is that inter-chip signals should be somewhat spread along an edge, so that competition for individual pins is reduced. When we consider mapping to a permuted 8-way Mesh with Superpins, one can see that it is unnecessary to place logic along any specific edge, since short paths exist from any edge in a source FPGA to each edge on other FPGAs. Thus, one can simply place inter-chip signals near any edge, not necessarily any specific one. The secondary concern, that inter-chip signals should be somewhat spread out across the edges, may still be important. Thus, instead of complicating the task of automatic mapping, these topologies actually simplify placement, do not particularly impact other tools, and improve overall routing costs.

Conclusions and Future Work

We have presented three techniques for decreasing routing costs in Mesh interconnection schemes: 8-way 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 of 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, 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.

References

Aptix Data Book, Aptix Corporation, San Jose, CA, 1993.

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.

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.

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

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.

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.

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.

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.