Chapter 7. Mesh Routing Topologies

Introduction

As discussed in Chapter 3, 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 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. FPICs (Chapter 2) 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.

Chapter 5 has demonstrated that there are many different possible multi-FPGA system topologies. 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 scaleability and chip utilization. Hierarchical crossbar structures 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 [Babb93, Selvidge95] 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. 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 an honest comparison between different topologies can be performed. In this chapter, 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 [Chan93b] and hierarchical crossbars [Varghese93] can be found elsewhere.

Mesh Routing Structures

The obvious structure to use for a mesh topology is a 4-way interconnection (Figure 45), 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 rest of this chapter 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.

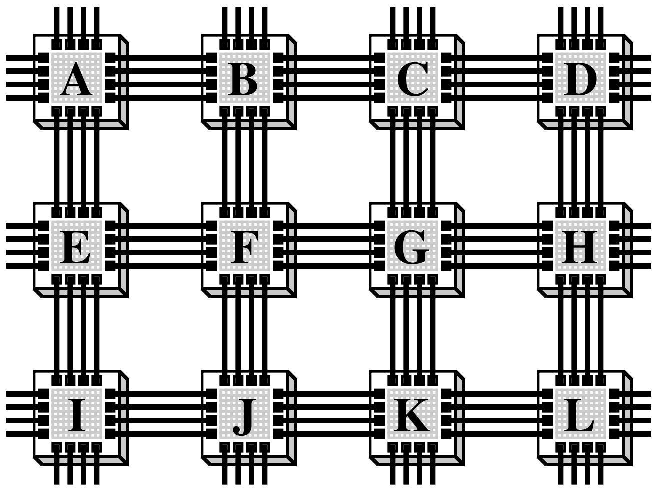


Figure 45. Basic mesh topology.

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 46 top left), we can adopt an 8-way topology. In the 8-way topology (Figure 46 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 46 right), where FPGAs directly adjacent vertically and horizontally, as well as those one step removed, 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.

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.

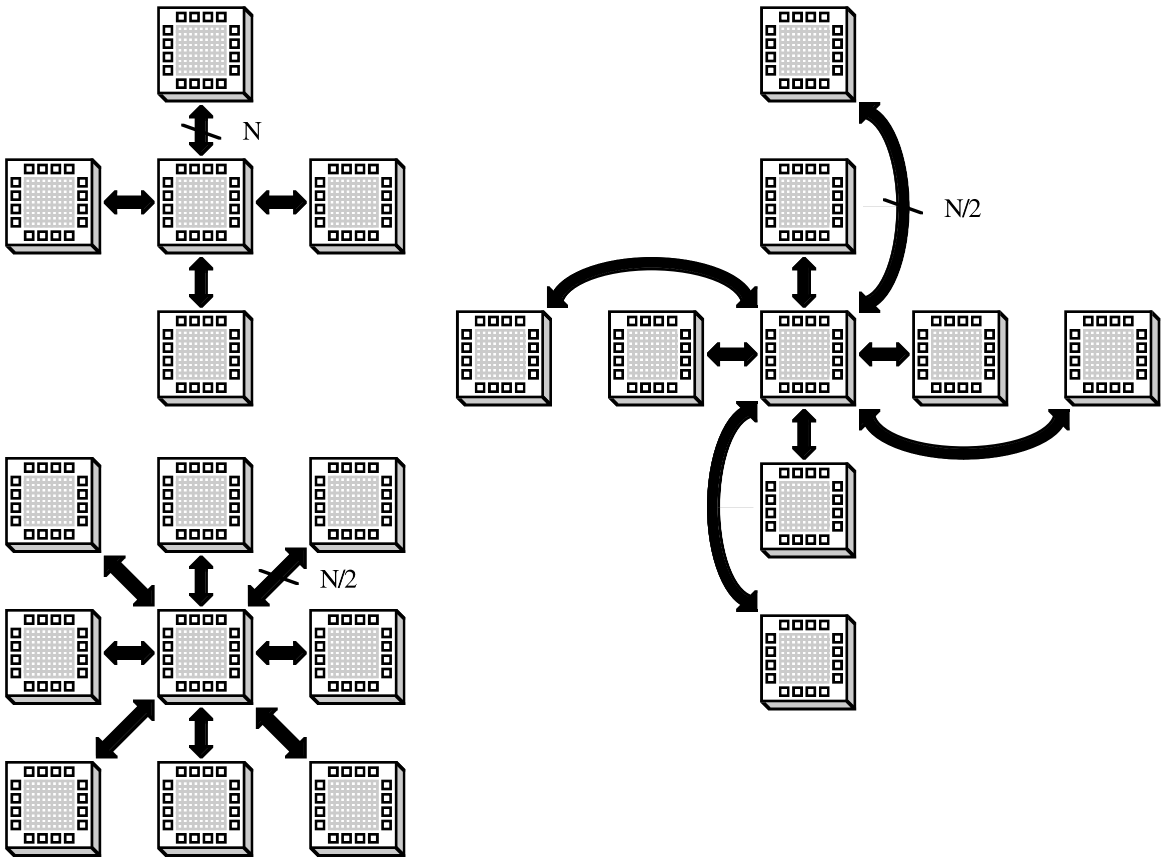


Figure 46. 4-way (top left), 8-way (bottom left), and 1-hop (right) mesh routing topologies.

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 47 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). 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 (while within a single I/O connection 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 connections. 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.

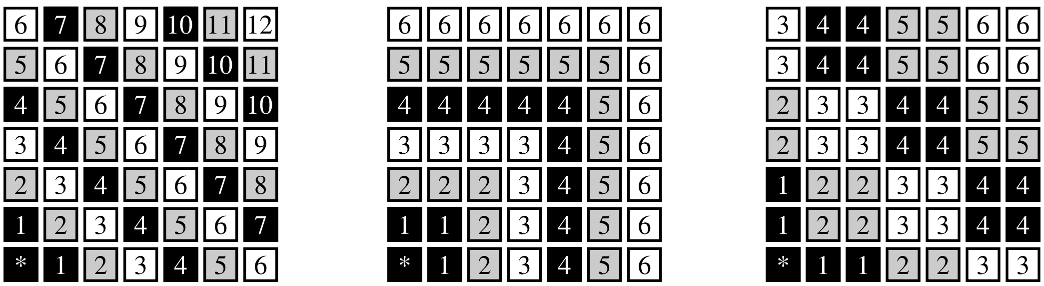


Figure 47. Distance (in number of I/O connections required) to reach FPGAs in 4-way (left), 8-way (center), and 1-hop (right) topologies. Distances are from the FPGA in the lower-left corner.

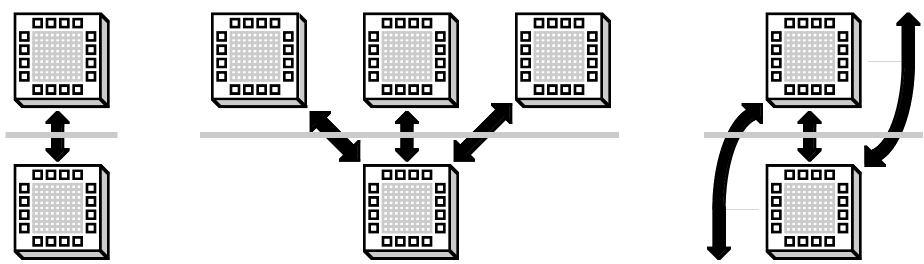
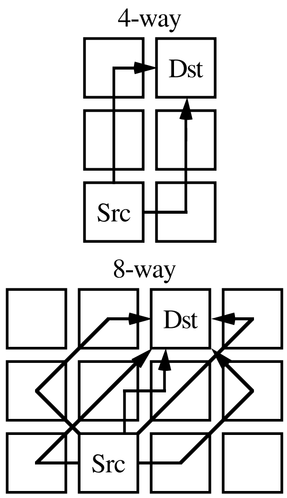


Figure 48. Bisection bandwidth in 4-way (left), 8-way (center), and 1-hop (right) topologies. Three times as many connections cross the bisecting line in the 8-way and 1-hop topologies than in the 4-way topology, though each connection contains half as many wires. This results in a 50% increase in bisection bandwidth.

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 48, 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.



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 49. 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.

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 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. 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 49 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 49 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 49 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 (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 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 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 50 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 50 top, moving from the top of the FPGA at left to the FPGA at right requires a large amount of internal routing resources.

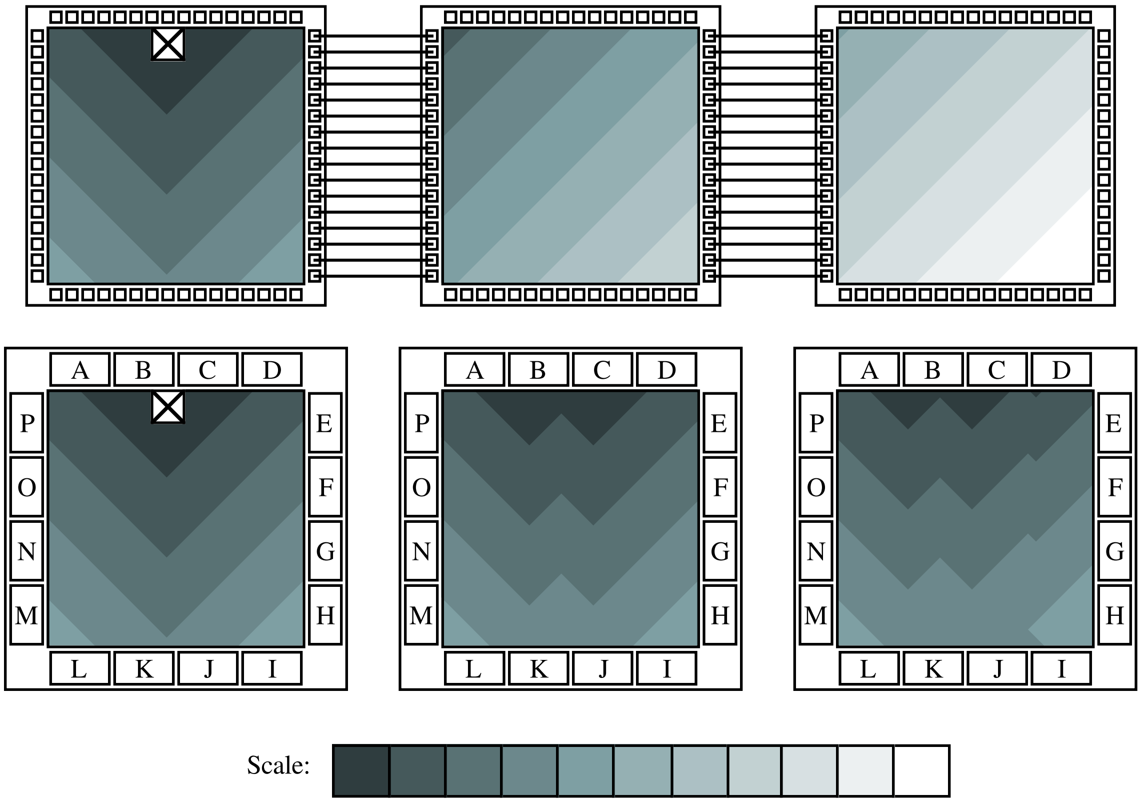


Figure 50. Distances from the X in leftmost FPGA in normal (top) and Superpin (bottom) connection pattern, on a scale from black (shortest) to white (longest). Superpin connections are given by letters, with Superpins with the same letter connected together in adjacent FPGAs.

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. 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.

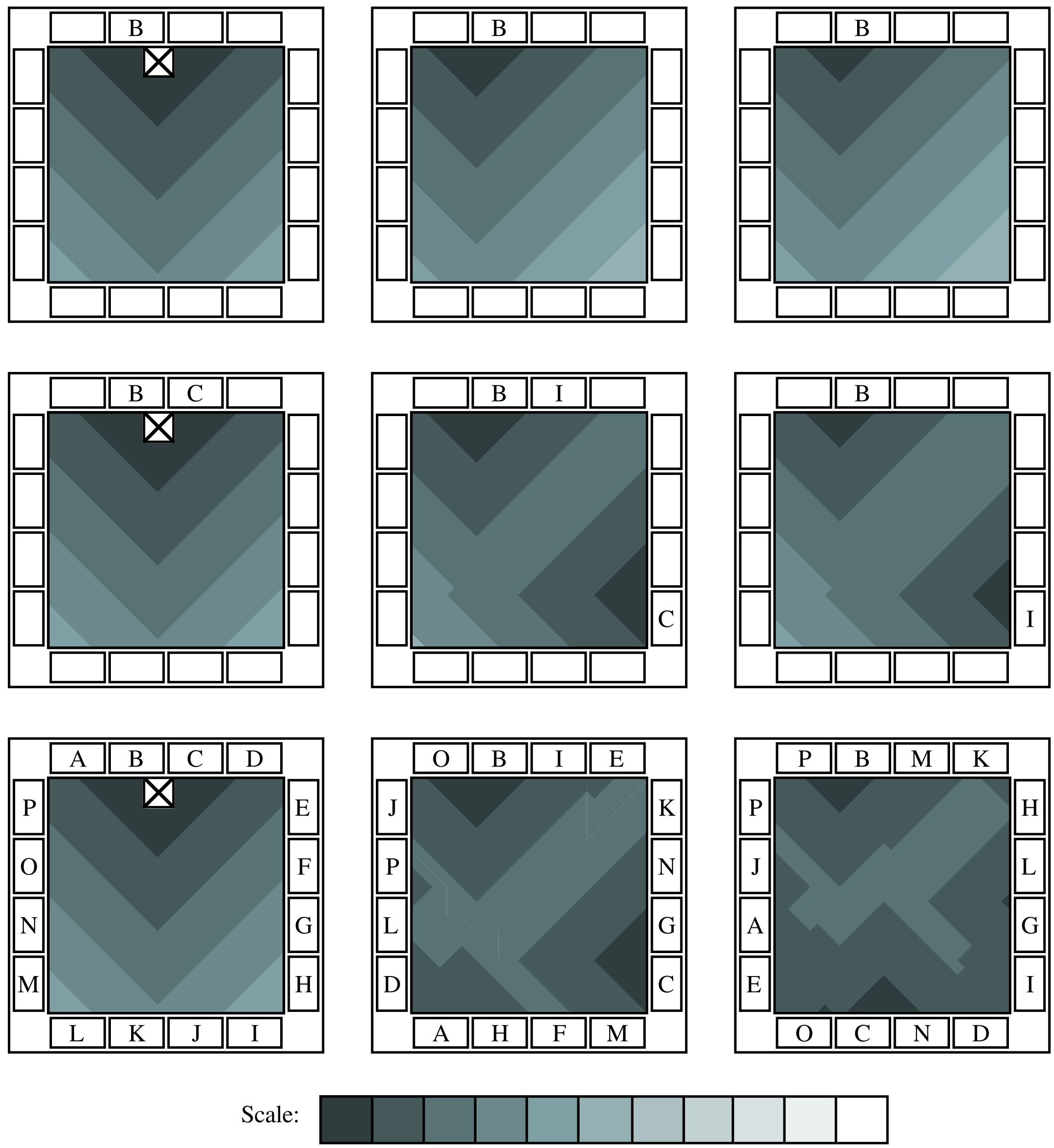


Figure 51. Intuition behind permuted Superpins. A single connection (at top) gives most of the benefit of full unpermuted Superpins. By changing connection **C** to the lower-right corner (middle), more short routes are achieved. Note that connection **I** is simply connection **C** for the middle FPGA. Bottom shows full permuted Superpins, with even shorter routes in further FPGAs. The scale ranges from black (shortest) to white (longest).

As shown in Figure 50 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 51 top, simply making the connection labeled "B" gives most of the benefit of the complete unpermuted Superpin pattern given in Figure 50. Thus, connecting "C" as we did in Figure 50 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 51 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 51 bottom).

An interesting observation is that by having two (identical) permutations in series in Figure 51 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 51 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.

It is possible to generate even better topologies by using several different permutations in a single system. As described later in this chapter, having different permutations in different directions takes advantage of reconvergent paths by having the short routes along one path lead 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.

Later in this chapter we present 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.

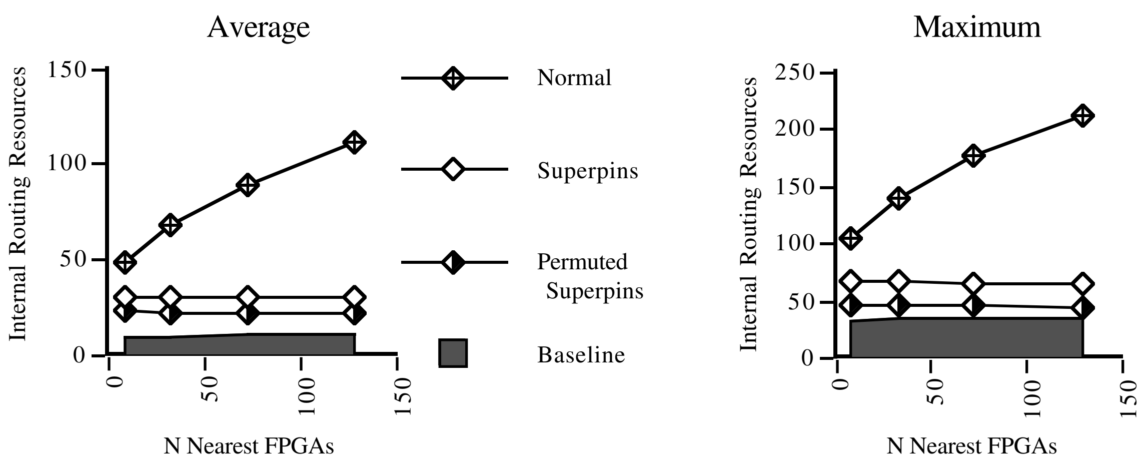
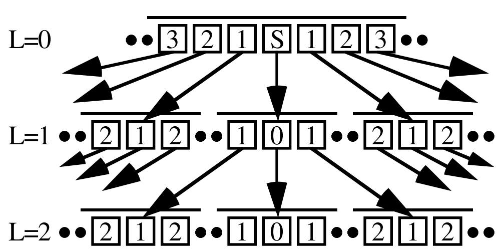


Figure 52. Average (left) and maximum (right) internal routing resource usage from each location in the source FPGA to all locations in the N nearest destination FPGAs in a 1-hop topology.

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 52. 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 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 use at least 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.

Bounds on Superpin Permutation Quality

As implied earlier, 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. Given this model, we can 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 inter-FPGA connections, 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.



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 53. 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.

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 *L* FPGAs away (that is, we travel through *L-1* intermediate FPGAs plus the destination FPGA). *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 shown in Figure 53. 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 $F(N,L) = F(N,L-1) + 2*\sum_{i=0}^{N-1} F(i,L-1)$, where $F(N, L)$ is the number of locations reachable in exactly N steps when moving L FPGAs away. 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 $F(0,L) = 1$, $F(j,0) = 2 | j > 0$.

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 FPGA size and routing distance. 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 for L=1 is (0*1 + 1*4 + 2*8 + 3*5)/18 = 1.944, for L=2 is (0*1 + 1*6 + 2*11)/18 = 1.556, and for L=3 is (0*1 + 1*8 + 2*9)/18 = 1.444. 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 the lower bound.

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, 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, the inverse of a permutation (the permutation seen by a route moving backwards through the original permutation) is in fact 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.

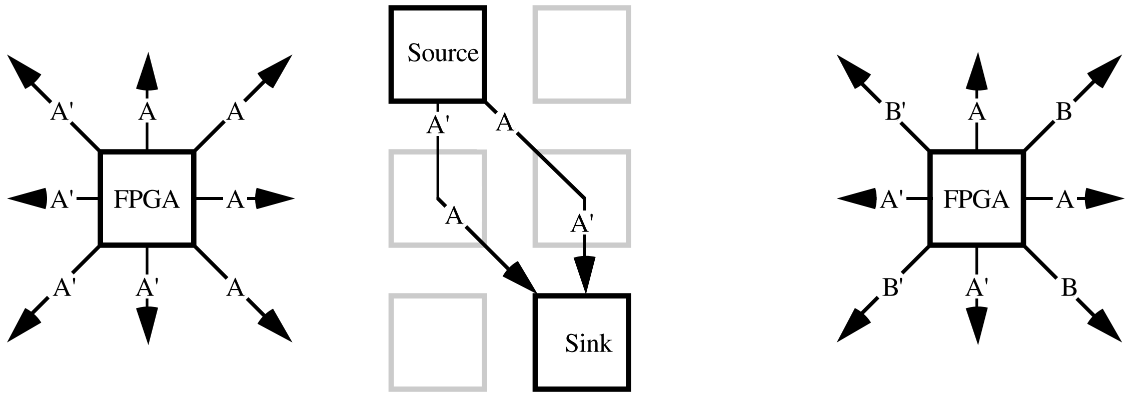


Figure 54. Permutations in 8-way meshes. The pattern at left, with a single permutation A, and its inverse A', causes some paths to use both the permutation and the inverse, as shown at center. The pattern at right, with permutations A and B, avoids this problem. The two permutations are chosen to be 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 54, 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 8-way topology that avoids the problem of paths with both a permutation and its inverse, but requires different routing from 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 54 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 in an 8-way mesh, 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 (see Figure 55). 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 in an 8-way mesh 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.

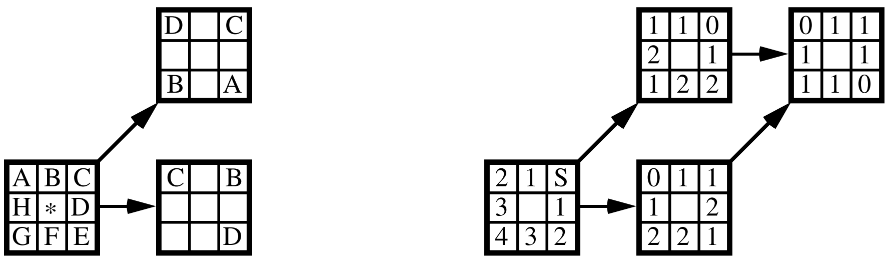


Figure 55. Example of how multiple paths in a two-dimensional permuted mesh decrease routing costs. A portion of the permutations leading northeast and east from the asterixed FPGA is shown at left. Each FPGA has 8 Superpins. At right is the resulting distances from the upper right Superpin (labeled "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.

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 56, 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. 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%.

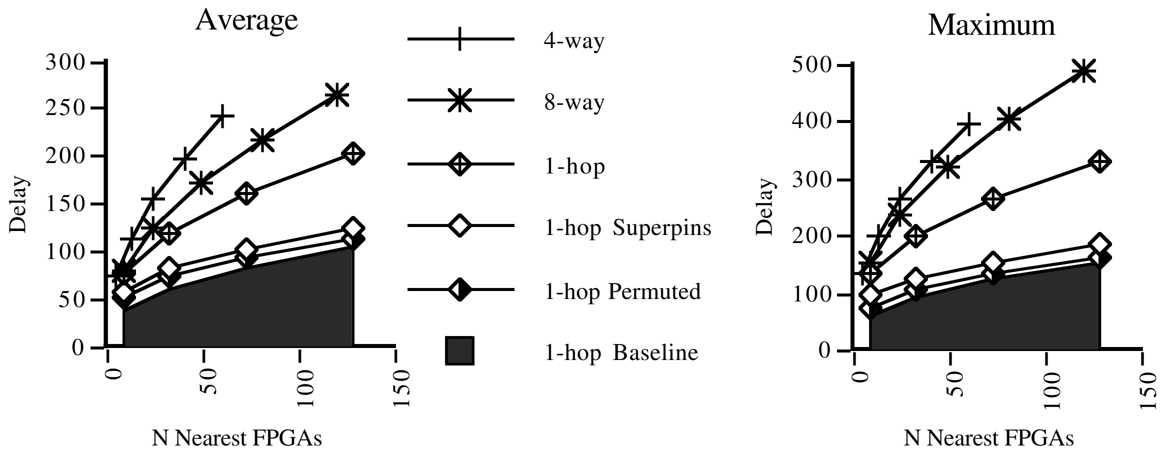


Figure 56. Graphs of average (left) and maximum (right) delay from each location in the source FPGA to all locations in the N nearest destination FPGAs.

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 [McMurchie95], 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 57) 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 graphs (Figure 58) is for a population of 5 signal bundles with random sources and sinks, though signals within a bundle share the same source and sink.

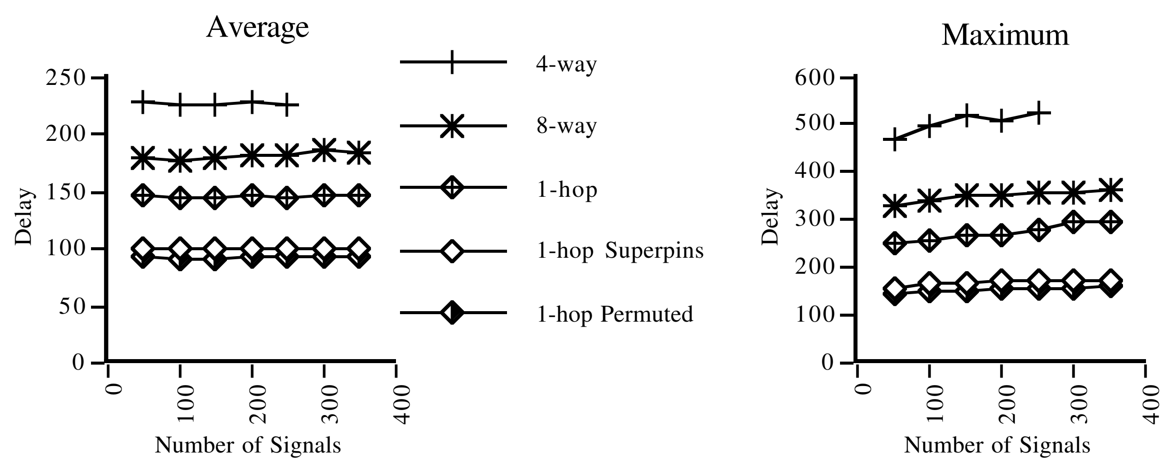


Figure 57. Average and maximum distance of signals under several topologies in a 5x5 array.

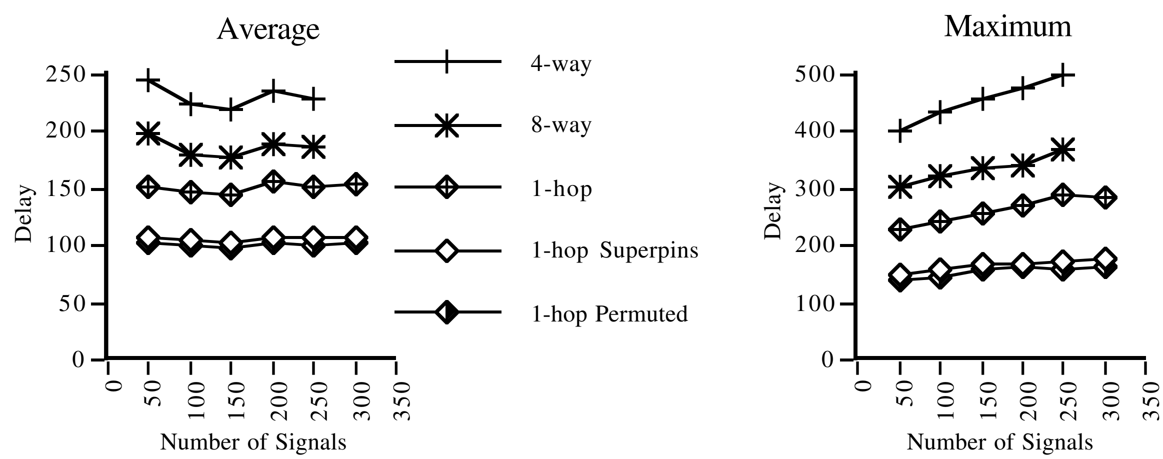


Figure 58. Average and maximum distance of signals under several topologies in a 5x5 array, with all signals in 5-signal buses.

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 56) 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 Chapter 12, 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 meshs 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 later in this thesis, starting with Chapter 9.

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 52) 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, crossbars, or some other general topology is a better choice. However, if this chapter 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.