IEEE Transactions on Computer-Aided Design of Integrated Circuits & Systems, Vol. 16, No. 9, pp. 956-964, September, 1997.

Pin Assignment for Multi-FPGA Systems

Scott Hauck

Department of ECE

Northwestern University

Evanston, IL 60208 USA

hauck@ece.nwu.edu

Gaetano Borriello

Department of CSE

University of Washington

Seattle, WA 98195 USA

gaetano@cs.washington.edu

Abstract

Multi-FPGA systems have tremendous potential, providing a high-performance computing substrate for many different applications. One of the keys to achieving this potential is a complete, automatic mapping solution that creates high-quality mappings in the shortest possible time. In this paper we consider one step in this process, the assignment of inter-FPGA signals to specific I/O pins on the FPGAs in a multi-FPGA system. We show that this problem can neither be handled by pin assignment methods developed for other applications nor standard routing algorithms. Although current mapping systems ignore this issue, we show that an intelligent pin assignment method can achieve both quality and mapping speed improvements over random approaches. Intelligent pin assignment methods already exist for multi-FPGA systems, but are restricted to topologies where logic-bearing FPGAs cannot be directly connected.

In this paper we provide three new algorithms for the pin assignment of multi-FPGA systems with arbitrary topologies. We compare these approaches on several mappings to

current multi-FPGA systems, and show that the force-directed approach produces better mappings, in significantly shorter time, than any of the other approaches.

Introduction

There is currently tremendous interest in the development of computing platforms from standard FPGAs. These systems harness multiple FPGAs, connected in a fixed pattern, to implement complex logic structures. In order to use such a system effectively, an automatic mapping system must be developed to translate hardware specifications into programming files for the individual FPGAs in the system. This mapping system differs from the standard CAD algorithms used for other technologies because of two major constraints: First, the connections between chips are fixed by the architecture of the multi-FPGA system, and thus any inter-chip routing must conform to this topology. Second, a multi-FPGA system is ready to use seconds after the mapping has been developed. Thus, the time to create a mapping is just as important as the resulting quality.

The process of mapping logic onto a multi-FPGA system usually goes through the following stages: First, logic synthesis and technology mapping convert the logic into functions that fit the logic blocks in the FPGAs. Second, partitioning and global placement assigns the logic to specific FPGAs in the system. Third, global routing and pin assignment route the inter-FPGA signals. Fourth, single-chip placement and routing tools develop complete mappings for the individual FPGAs in the system.

Figure 1. Two views of the inter-FPGA routing problem: as a complex graph including internal resources (left), and a more abstract graph with FPGAs as nodes (right).

One way of performing the global routing and pin assignment step for multi-FPGA systems is to use standard single-FPGA routing algorithms. In this approach the FPGAs are viewed as complex entities, explicitly modeling both internal routing resources and individual IOBs connected by board traces (Figure 1 left). The routing algorithm would be used to determine both which intermediate FPGAs to use for long distance routing (i.e., a signal from FPGA **A** to **D** would be assigned to use either FPGA **B** or **C**), as well as which individual IOBs to route through. Unfortunately, although the logic has already been assigned to FPGAs during partitioning and global placement, the assignment of logic into individual logic blocks will not be done until the final step, single chip placement and routing. Thus, since there is no specific source or sink for the individual routes, standard routing algorithms cannot be applied.

The approach generally taken is to abstract entire FPGAs into single nodes in the routing graph, with the arcs between the nodes representing bundles of traces. This solves the unassigned source and sink problem mentioned above, since partitioning has already assigned the logic to specific FPGAs. Unfortunately, the routing algorithm can no longer determine which individual traces a signal should use, since those details have been abstracted away. It is this problem, the assignment of a mapping's logic pins to FPGA IOBs, that this paper addresses. The techniques developed in this paper, when combined with solutions we have developed for other multi-FPGA

mapping and architecture issues [11, 10], can be combined to create a complete multi-FPGA system solution for rapid-prototyping and other domains [9].

Problem Definition

In the pin assignment problem for multi-FPGA systems, we start with a multi-FPGA system and a logic mapping to be fit onto this system. The logic mapping consists of *partitions* of logic, interconnected by external *signals*, as well as *logic pins* which represent the connection of a partition to a signal. Similarly, the multi-FPGA system consists of FPGA *chips*, interconnected by *traces* on the circuit board, as well as *IOBs* which represent the connection of a trace to a chip. There is also a pre-existing assignment of partitions to chips, where each partition must be assigned to a single chip, and each chip can be assigned at most one partition.

The pin assignment process for multi-FPGA systems assigns all signals to traces, and logic pins to IOBs. Specifically, each signal **S**, which is connected to partitions **P1**…**Pn**, must be assigned to exactly one trace **T**. This trace must be connected to chips **C**={**C1**…**Cm**} such that each partition **P**∈{**P1**…**Pn**} is assigned to a chip **c**∈**C**. Each trace can be assigned at most one signal. By assigning signals to traces, we also fix the placement of logic pins to IOBs, constraining the placement of the FPGAs in the system. Note that under this definition a signal can only move between FPGAs that are directly connected (i.e. cannot take multiple hops from the source to the destination). This means that long distance signals must be split into multiple shorter interconnected signals, where each short signal represents the passage of the signal between connected FPGAs.

Figure 2. Terminology for the pin assignment problem.

We judge the quality of a pin assignment by its impact on the routing resource usage in the FPGAs in the system after the logic mapping has been completely mapped to the multi-FPGA system, with a good pin assignment requiring a minimum of routing resources. In this paper, we estimate the routing resource usage in a multi-FPGA system by the cumulative source to sink delay of all signals in all FPGAs in the system. This metric will penalize those assignments that require excessive usage of routing resources in the FPGA, which will decrease performance and reduce the amount of logic that can be fit into the system.

Existing Pin Assignment Approaches

The problem of pin assignment has been studied in many other domains, including routing channels in ASICs [3], general routing in cell-based designs [24, 5], and custom printed circuit boards (PCBs) [17]. Unfortunately, these approaches are unsuitable to the multi-FPGA pin assignment problem. First, these algorithms seek to minimize the length of connections between cells or chips, while in a multi-FPGA system the connections between chips are fixed. The standard approaches also assume that pin assignment has no effect on the quality of the logic element implementations, while in the multi-FPGA problem the primary impact of a pin assignment is on the quality of the mapping to the logic elements (individual FPGAs in this case). Because of these differences, there is no obvious way to adapt existing pin assignment approaches for other technologies to the multi-FPGA system problem.

There are existing methods of pin assignment for certain multi-FPGA topologies. These approaches require that all logic-bearing FPGAs are connected only through FPICs[1], crossbar chips, or routing-only FPGAs. It is assumed that the routing chips can handle any assignment of signals to their IOBs equally well, and thus the only consideration for pin assignment is the routeability of the logic-bearing chips. Since no logic-bearing chips are directly connected we can place these FPGAs independently. These placements induce a pin assignment onto the routing chips. Since the routing chips can handle any pin connection pattern equally well, the routing chips need only be configured to implement this pattern, and the mapping to the multi-FPGA system is complete. While most such approaches require a separate global routing step to determine which routing chips to send signals through [2, 13, 16], it is possible to handle some carefully constructed topologies without global routing [4].

The existing pin assignment methods are adequate for systems where logic-bearing FPGAs are never directly connected. However, the majority of multi-FPGA systems have some or all of their logic-bearing FPGAs directly connected [9], and thus cannot be handled by existing techniques. Today's systems address this problem by simply creating a random pin assignment, or by requiring manual assignment by the user. For random assignments, the global routing step decides which FPGAs to route through based purely on inter-chip concerns, and then pin assignment randomly assigns the signals to traces connecting the appropriate FPGAs.

By using a random pin assignment, the system gives up a potential optimization opportunity, degrading both routing resource usage and mapping time. A random pin assignment is likely to

[1] An FPIC [1, 12] is a purely routing device, allowing arbitrary interconnections between its I/O pins.

positioning. This will also make it harder to find a routing for the individual FPGAs, slowing to four times as long simply because of the poor pin assignment.

multi-FPGA topologies. These algorithms take into consideration the structure of the logic traces. The first methods demonstrate how exact placement and routing information for the then demonstrates how less exact information can be used, maintaining the pin assignment quality

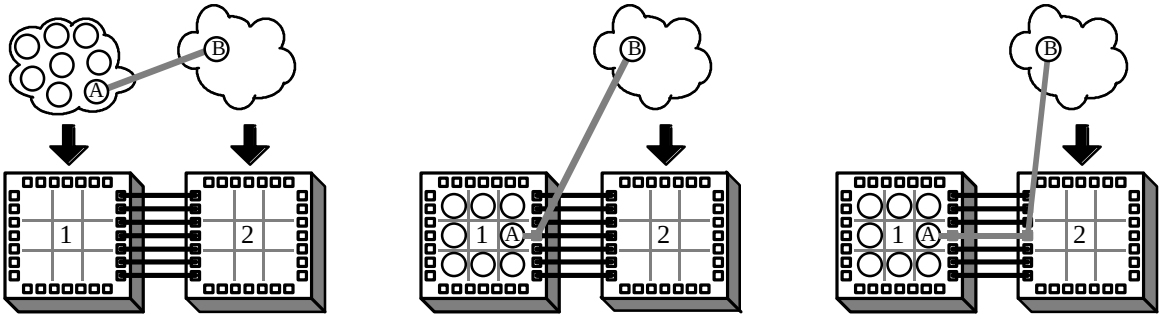

Figure 3. Example of pin assignment by sequential placement. Partitioning and global placement assigns logic to different FPGAs (left), putting connected logic blocks **A** and **B** in adjacent FPGAs. The mapping to FPGA **1** is then placed (center), with the restriction that the signal from **A** to **B** must go on a trace from **1** to **2**. After placement, the assignment of this signal propagates to the adjacent FPGAs (right), forcing a specific pin assignment for the signal from **A** to **B**. FPGA **2** would then be placed.

Sequential Placement Pin Assignment

One approach to pin assignment is to allow the FPGA placement tool to determine its own assignment. This requires that the placement tool allow the user to restrict the locations to which a logic pin can be assigned (a feature available in tools such as the Xilinx APR and PPR placement and routing tools [22]). With such a system, inter-chip signals are restricted to only those IOBs that are connected to the proper destinations. Once the placement tool determines the pin assignment for one FPGA, this assignment is propagated to the attached FPGAs (see Figure 3). Unfortunately, this limits the number of placement runs that can be performed in parallel. Specifically, the assignment from one FPGA is propagated to adjacent FPGAs only after that FPGA has been placed, and thus no two adjacent FPGAs can be placed simultaneously. Since the placement and routing steps can be the most time-consuming steps in the mapping process, achieving the greatest parallelism in this task can be critical.

An algorithm for achieving the highest parallelism during placement, while allowing the placement tool to determine the pin assignment, is to find a minimum graph coloring of the multi-FPGAs system (FPGAs are nodes, board traces are edges). Since the structure of a multi-FPGA system is usually predefined, the coloring can be precomputed, and thus the inefficiency of finding a graph coloring is not important. Then, all the FPGAs assigned the same color can be placed at the same time, since adjacent FPGAs cannot be assigned the same color. For example, in a four-way mesh (every FPGA is connected to the FPGA directly adjacent horizontally and vertically), the FPGAs could be placed in a checkerboard pattern, with half handled in the first iteration, and half in the second (Figure 4 left).

One problem with the maximum parallelism (or “checkerboard”) method is that while the pin assignment for the FPGAs placed first will be very good, since the placer is mostly free in its placement choices, other FPGAs may be placed fairly poorly. For example, in the mesh shown in Figure 4 left half of the FPGAs have their pin assignments dictated by the placement of their neighbors, and thus there is no attention paid to the logic within these FPGAs.

There is an alternative to the checkerboard algorithm, which trades longer mapping run times for better results. The idea is to make sure that FPGAs are not mapped completely independently of one another. In the first step, a single FPGA is placed. The FPGA that is most constrained by the system’s architecture (i.e., by external interfaces) is normally chosen, since it should benefit most from avoiding extra pin constraints. In succeeding steps, neighbors of previously placed FPGAs are chosen to be placed next, with as many FPGAs placed as possible without simultaneously placing interconnected FPGAs. For example, in a 4-way mesh, where the first FPGA routed is in the upper-left corner, the mapping process would proceed in a diagonal wave from upper-left to lower-right (Figure 4 right). In this way, mappings “grow” from a single “seed” assignment, and will be more related to one another than in the checkerboard approach, hopefully easing the mapping tasks for all of the FPGAs.

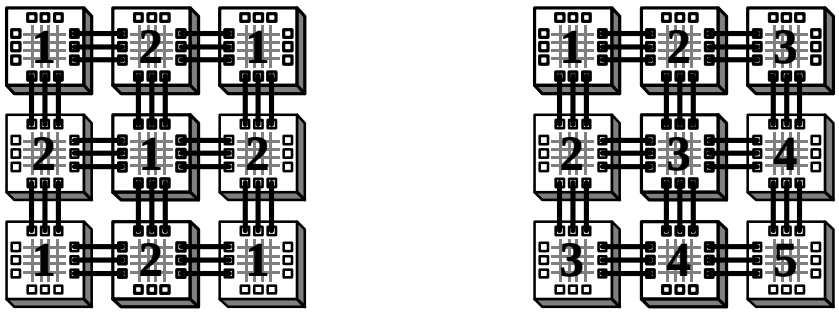

Figure 4. Checkerboard (left) and wavefront (right) pin assignment placement orders.

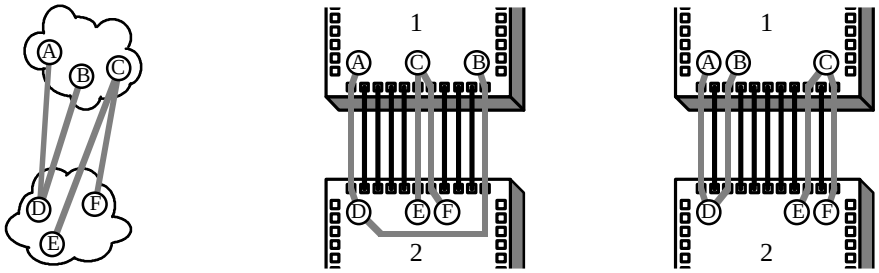

Figure 5. An example of the quality problems with sequential placement based pin assignments. The logic at left must be assigned to FPGAs **1** and **2**. If FPGA **1** is placed first the signals leading to **C** will be grouped together, but **A** and **B** may be separated. Changing the placement order will group together **A** and **B**, but could separate **E** and **F**. A better placement is shown at right.

Unfortunately, even this "wavefront" placement order may not generate good pin assignments. While the individual FPGA placements attempt to find local optimums, global concerns are largely ignored. For example, while signals that are used together in an FPGA will be placed together in the mapping, signals used together in an unplaced neighbor may be scattered (Figure 5). There is also a significant performance overhead for the checkerboard and wavefront methods (hereafter referred to jointly as "sequential placement methods") since they sequentialize the very time-consuming step of placement for different FPGAs.

There are some topologies that cannot be handled by sequential placement approaches. Specifically, when the traces that connect FPGAs are allowed to connect more than two FPGAs there is the potential for conflicts to arise which can cause failures with sequential placement methods. For example, consider the topology in Figure 6. Assume that we wish to map a circuit with one signal connected between **A** and **C**, seven between **A** and **B**, and seven between **B** and **C**. In this situation at least one signal connected between each pair of FPGAs must be placed on

a three-destination trace. However, regardless of the order, the first FPGA placed can use all of the three-destination traces for its own signals, causing the assignment to fail.

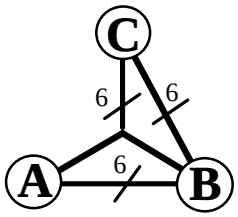

Figure 6. Example of a multi-FPGA topology that cannot be handled by sequential placement techniques, but which can be found in current systems [20, 7, 14, 6, 21].

Force-Directed Pin Assignment

As we have shown, pin assignment via sequential placement can be slow, cannot optimize globally, and may not work at all for some topologies. What is necessary is a more global approach which optimizes the entire mapping, while avoiding sequentializing the placement step. In this section, we will present one such approach, and give a quantitative comparison with the approaches presented earlier.

Intuitively, the best approach to pin assignment would be to place all FPGAs simultaneously, with the placement runs communicating with each other to balance the pin assignment demands of each FPGA. In this way a global optimum could be quickly reached. Unfortunately, the communication necessary to perform this task would be prohibitive. The force-directed approach discussed here is similar to simultaneous placement, but will perform the assignment on a single machine within a single process. Obviously, with the placement of a single FPGA consuming considerable CPU time, complete placement of all FPGAs simultaneously on a single processor is impractical, and thus simplification of the problem is key to a workable solution.

Force-directed pin assignment uses force-directed placement of the individual FPGAs [18]. In force-directed placement, the signals that connect logic in a mapping are replaced by springs between the signal's source and each sink, and the placement process consists of seeking a minimum net force placement of the logic. To find this minimum net force configuration, and thus minimize wirelength in the resulting mapping, the software randomly chooses a logic block and moves it to its minimum net force location. This greedy process continues until a local optimum is found, at which point the software accepts that configuration.

Force-directed placement may seem to be a poor choice for pin assignment, and is generally felt to be inferior to simulated annealing for placement. Two reasons for this are the difficulty force-directed placement has optimizing for goals other than wirelength, and the inaccuracy of the spring approximation to routing costs. However, force-directed placement can handle all of the optimization tasks involved in pin assignment, and we exploit properties of the spring metric in order to efficiently handle multi-FPGA systems. Also, force-directed placement is much faster than simulated annealing, and the time to create a mapping is an important issue for multi-FPGA systems.

As implied earlier, we will not simply place individual FPGAs, but will in fact use force-directed placement simultaneously on all FPGAs in the system. To do this efficiently we make two alterations to the basic force-directed algorithm:

1.) We do not restrict logic functions to non-shared, discrete locations, but instead allow them to be freely placed into any location in the FPGA with no regard to congestion or physical logic block boundaries (however, logic pins are constrained to exact IOBs, and logic pins cannot share a single IOB location).

2.) We assume that all logic functions are always at their optimal positions.

While the second alteration is simply a change in the movement order of nodes, the first change could cause a significant loss in accuracy. However, the resulting degradation of the local optimum will be more than made up for by the ability to seek the global optimum. Also, while this allows logic functions to share physical locations, something that cannot be done in valid FPGA mappings, force-directed placement is used only to determine pin assignments, and the logic functions will be placed later by standard FPGA placement tools. Thus, the final placements will have logic functions assigned to unique, discrete logic block locations. Note that a similar assumption has proven to be a reasonable tradeoff in the placement of standard cells via simulated annealing [19].

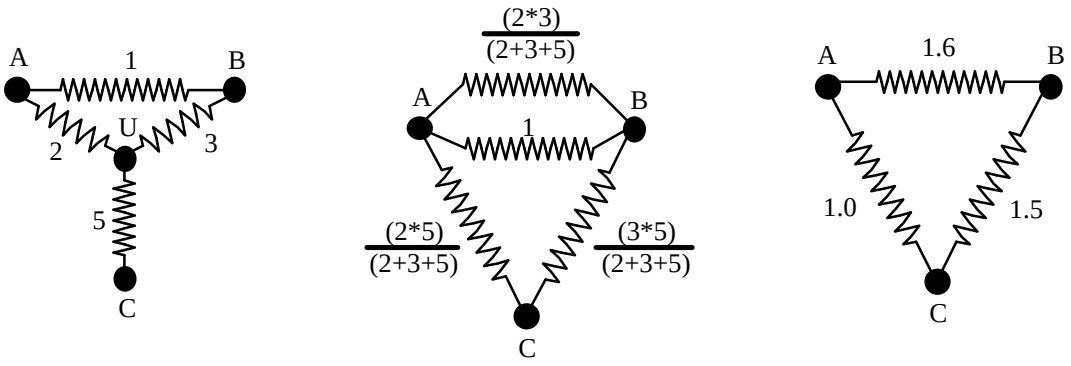

Figure 7. Example of spring simplification rules. The source circuit at left has logic node **U** replaced at center, and any springs created in parallel with others are merged at right.

By making the two assumptions just given, we now have the opportunity to greatly simplify the mapping process. We can examine the system of springs built for the circuit mapping, and use the laws of physics to remove nodes corresponding to logic functions, leaving only logic pins. As shown in appendix A, as well as in the example of Figure 7, the springs connected between an internal logic node and its neighbors can be replaced with a set of springs connected between the

node's neighbors while maintaining the exact same forces on the remaining nodes. By repeatedly applying these simplification rules to the logic nodes in the system, we end up with a mapping consisting only of logic pins, with spring connections that act identically to the complete mapping they replace. In this way, we simplify the problem enough to allow the pin assignment of a large system of FPGAs to be performed efficiently.

$$\sum_{i=1}^{n} SpringConst_i \times (pos_i - pos_{node}) = \left(\sum_{i=1}^{n} SpringConst_i \times pos_i \right) - \left(\sum_{i=1}^{n} SpringConst_i \right) \times pos_{node}$$

Equation 1. Reformulation of the standard force equation.

There are some details of the force-directed algorithm that need to be considered here. First, efficiency in the individual move calculations is critical. To determine the best destination of a node during a move it is necessary to calculate the change in net force for assigning the signal to each trace that goes between the FPGAs the signal connects. This is necessary because there may not be any relationship between the location of an IOB on one FPGA and the location of the IOB the trace goes to on another FPGA (in fact, it has been shown that there is some advantage to be gained by scattering IOB connections [11]). As shown in Equation 1, we can speed up the process of determining the net force at each location by reformulating the standard force equation (at left) so that the information from all springs connected to a node can be combined at the beginning of a move. Then, the calculation of the net force at each location requires only a small amount of computation.

A final extension necessary is software support to avoid the problems found in the sequential approaches on some topologies. Just as the topology in Figure 6 caused problems for the other approaches, it could cause difficulties with the force-directed approach. The solution is to ensure that a signal is never moved into a trace unless it connects to the FPGAs the signal moves

between, and any signal already at that location can be moved to another position under the same restrictions.

As mentioned earlier, the force-directed approach is capable of handling most of the optimization goals necessary for pin assignment. First, since little information about the individual FPGA architectures is necessary beyond the location of the IOBs, it is very easy to port this software to different FPGAs. In fact, our current system allows different FPGA architectures and sizes to be intermingled within a single multi-FPGA system. We can also accommodate crossbar chips and FPICs, chips where minimizing wirelength is unnecessary, by setting the spring constant of all springs within such chips to zero.

Another important consideration is support for predefined pin assignments for special signals. Specifically, signals such as clocks, reset signals, fixed global communication lines, and host interfaces need to be assigned to specific IOB locations. Handling these constraints in the pin assignment tool simply requires permanently locking these signals into their proper locations.

Experimental Results

To compare the various pin assignment approaches, we tested each approach on mappings for four different current systems. These include a systolic DNA comparison circuit for Splash [15], a telecommunications circuit for the NTT board [23], a calorimeter circuit for DECPeRLe-1 [21], and a RISC processor configured for logic simulation on the Marc-1 system [14]. The pin assignment systems compared are: “Random”, which randomly assigns connections to traces; “Checkerboard” and “Wavefront”, which use sequential placement runs to do pin assignment; and “Force”, which uses the force-directed pin assignment technique. The results include both mapping time (Table 1) and resulting wirelength (Table 2). “Multi” is CPU time on SPARC-10s,

and includes the time to perform pin assignment as well as individual FPGA place and routes, and assumes enough machines are available to achieve maximum parallelism. "Uni", the time it takes to complete all of these tasks on a single machine, is also included. The reason the "Multi" category is included is that most users of multi-FPGA systems use multiple machines to perform the placement and routing of the individual FPGAs in the system. In this way, significant performance improvements can be obtained without requiring any special parallelization.

System		Splash		NTT		DECPeRLe-1		Marc-1	
Mapping		DNA Comparator		Telecom		Calorimeter		Logic Sim. Processor	
Force	Multi	33:38		8:26		28:47		24:57	
	Uni	12:04:25		19:12		2:42:51		3:36:32	
Wave	Multi	5:35:40	(9.980)	20:50	(2.470)	2:11:33	(4.570)	Failure	
	Uni	13:24:58	(1.111)	22:36	(1.177)	3:10:02	(1.167)		
Checker	Multi	1:07:00	(1.992)	12:22	(1.466)	1:51:24	(3.870)	Failure	
	Uni	12:37:36	(1.046)	21:55	(1.142)	2:55:24	(1.077)		
Random	Multi	1:00:18	(1.793)	8:18	(0.984)	30:47	(1.070)	26:38	(1.068)
	Uni	17:45:56	(1.471)	20:59	(1.093)	2:56:44	(1.085)	4:36:32	(1.277)

Table 1. Mapping time comparison table. Numbers in parentheses are normalized to the force-directed algorithm's results. "Multi" is the time on multiple processors, while "Uni" is the time on a single machine.

System	Splash	NTT	DECPeRLe-1	Marc-1
Force	143690.9	7292.2	45159.4	102728.6
Wave	146424.8 (1.019)	7345.6 (1.007)	45661.2 (1.011)	Failure
Checker	149860.5 (1.043)	7714.6 (1.058)	46522.5 (1.030)	Failure
Random	156038.2 (1.086)	7854.8 (1.077)	48713.4 (1.079)	108236.1 (1.054)

Table 2. Routing resource usage comparison table. Numbers in parentheses are normalized to the force-directed algorithm's results.

The sequential placement techniques only sequentialize the placement step, while routing is allowed to be performed in parallel with subsequent placements and routings. Also, these placement attempts are begun as soon as possible, so that for example in the checkerboard algorithm, we do not need to wait to begin placing FPGAs of one color until all FPGAs of the previous color are placed, but instead can proceed with a placement once all of its direct neighbors in the previous color(s) have been completed. The random approach is assumed to be able to generate a pin assignment in zero time, so its time value is simply the longest time to place and route any one of the individual FPGAs. Each of these assumptions is the best case for that specific algorithm.

The wirelength is determined by summing up all source to sink delays for all signal in the FPGAs. This is based on the assumption that the delay of a route is proportional to its routing resource usage. While our metric is not exact, since some portions of a multi-destination route will be included more than once, it gives a reasonable approximation of the routing resource usage of the different systems.

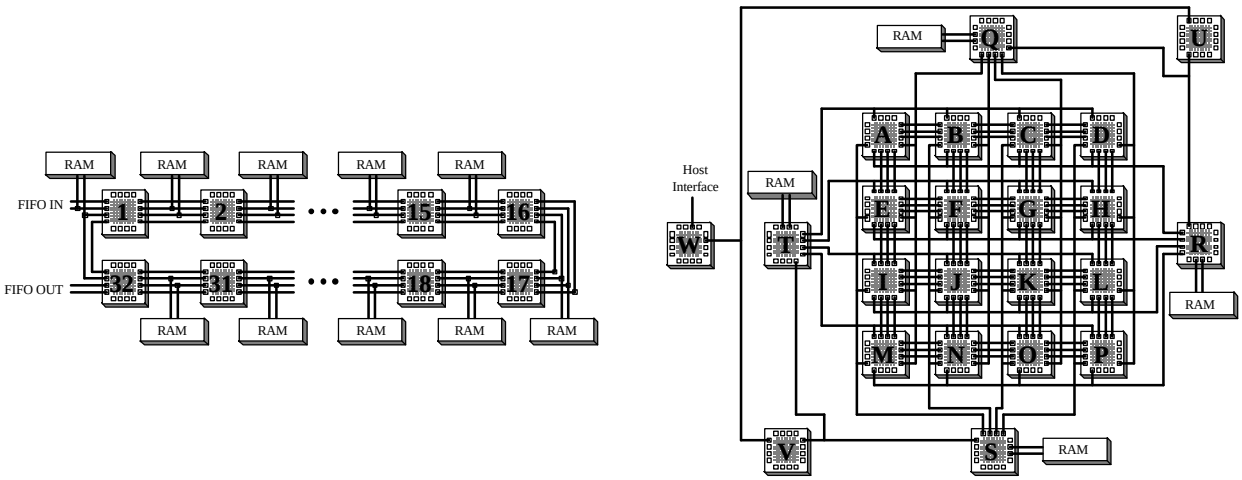

Figure 8. The SPLASH system [15] (left), and the DECPeRLe-1 board [21] (right).

The Splash system [8] is a linear array of Xilinx 3090 FPGAs (Figure 8 left), with memories attached to some of the communication wires. As expected, Random does poorly on the average wirelength, with Checkerboard and Wavefront each doing successively better. However, the Force-Directed approach does the best, since it is able to optimize the mapping globally. More important is the fact that the Force-Directed approach is actually much faster than all the other approaches, including the Random approach, in both parallel and uniprocessor times. The reason for this is that the poor pin assignment generated by the Random approach significantly increases the routing time for some of its FPGAs, up to a factor of more than 4 times as long. Note that the time to perform the force-directed pin assignment does not dominate for either this or any other mapping. For the DNA mapping it is only 7% of the overall (parallel) runtime, for Telecom it is 10%, and for Calorimeter it is 7%, though for the Marc-1 mapping it is 27%. The percentage of the runtime for the uniprocessor case is even lower, never being more that 4% of the total.

The results are almost identical for the DECPeRLe-1 board [21], a four by four mesh of Xilinx 3090 FPGAs with seven additional support FPGAs (Figure 8 right), and the NTT board [23], a five by five mesh of 3042 Xilinx FPGAs with wrap-around connections on the top and bottom

edges (Figure 9 left). However, the Checkerboard approach takes much longer on the DECPeRLe-1 board than it did on the Splash board, making it only about 15% faster than the Wavefront approach. The reason is that the DECPeRLe-1 board, which has 23 FPGAs, is only 19-colorable, leaving very little parallelism for the Checkerboard algorithm to exploit.

The Marc-1 board [14] is a complex system of Xilinx 4005 FPGAs, as shown in Figure 9 right. This board includes the troublesome construct from Figure 6, a topology that makes mapping failures possible in sequential placement approaches regardless of placement order. We attempted to use these sequential placement approaches four times each on this system, but each time it failed because of resource conflicts. However, the force-directed approach easily handled this topology, and generated a good assignment. Note that while the force-directed result is somewhat less of an improvement than the other examples led us to expect, this may be primarily due to the fact that only 51% of the logic pins are assignable, with the others fixed by memory interfaces and other fixed global signals.

As shown by the previous examples, the force-directed algorithm always produces better results than the other approaches, and almost always does so in significantly less time as well. The key to this is the spring reduction rules, which were based on the assumption that during force-directed placement the logic functions need not be restricted to fixed, discrete logic block locations. As mentioned earlier, this assumption is not true to the actual placement requirements, and thus we lose some accuracy in local optimization. However, this allows the algorithm to optimize for global properties, something the other approaches cannot do, and thus achieve better overall results. Although inaccurate, this assumption is reasonable in practice. Note that after the force-

directed pin assignment is completed, standard single chip placements are run, and these placements require that each logic function be placed in a unique, discrete logic block location.

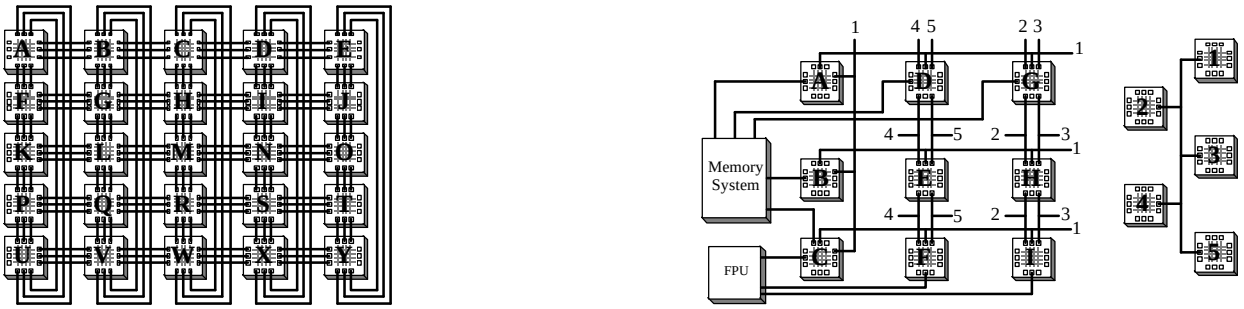

Figure 9. The NTT board [23] (left), and the Marc-1 topology [14] (right).

Example	Force	Wave	Checker	Random
Splash	1.034	1.054	1.079	1.123
NTT	0.988	0.995	1.045	1.064
DECPeRLe-1	1.208	1.222	1.245	1.303
Marc-1	1.126	failure	failure	1.186
Geom. Means:	1.086	N/A	N/A	1.166

Table 3. Ratio of pin assignment results to unconstrained pin assignment distances.

While the comparisons demonstrate the advantages of the force-directed algorithm over the simple approaches described earlier, it would be useful to determine how close to optimal is the force-directed approach. As an estimate, we considered the case where we do not perform pin assignment, and ignore the constraint that logic pins connected to a single signal must be assigned to IOBs connected to a single trace. Thus, the individual FPGAs in the system are placed and routed independently, and we only require that logic pins be assigned to IOBs connected to the correct neighbor. Although this unconstrained placement may not be optimal, since a very intelligent pin assignment may be able to improve the quality of subsequent placements, we believe that these results will be relatively close to the optimal. In Table 3 we present

comparisons between the unconstrained results and each of the algorithms discussed here. As can be seen, on average the force-directed pin assignment achieves results within 8.6% of unconstrained, which is 8% better than the random approach. In one case the force-directed results are even better than the unconstrained results. Because of these results, we believe that not much improvement (at least in routing resource usage) can be achieved beyond what the force-directed algorithm already provides.

Conclusions and Future Work

In this paper we have considered the problem of pin assignment for multi-FPGA systems. During pin assignment, the mapping tools for a multi-FPGA system decide which specific IOBs on the FPGAs will be used to route a given signal. This choice can have a quality impact, consuming varying amounts of routing resources, and a mapping speed impact, with a poor pin assignment complicating the subsequent placement and routing tasks and thus slowing down these tools. In a multi-FPGA system, where the system is ready to use as soon as a mapping is completed, the time to perform a mapping is as important as the quality of the results.

As we showed in this paper, pin assignment methods for other technologies are unusable for the multi-FPGA system case. The primary problem is that in a multi-FPGA system the traces between the IOBs on the FPGAs are fixed in a potentially arbitrary fashion. Pin assignment techniques for other technologies assume that the interconnection pattern is variable and attempt to minimize the wirelength of a given pin assignment. This is obviously inappropriate for a multi-FPGA system where the inter-chip topology is fixed. There are existing techniques for pin assignment of multi-FPGA systems without directly connected logic-bearing FPGAs. However, most multi-FPGA systems do not fit this model, and thus require a new technique. Many current

systems ignore the issue of pin assignment and simply randomly assign signals to traces, or require manual user pin assignment. As we showed, this random pin assignment can degrade the mapping quality, requiring the mapping to use more internal routing resources and the single chip routing tools to take significantly longer to create a mapping.

This paper has described several new methods for handling the general multi-FPGA system pin assignment problem. First, we introduced sequential placement approaches, which use the power of existing placement tools to create intelligent pin assignments. While these approaches do not work on all topologies, and slow down the mapping process because they serialize some FPGA placements, they are capable of providing some quality improvement over the purely random approaches. This yielded quality improvements of 2% to 6%, but with up to a factor of 6 slowdown of the mapping process.

We have also described another new pin assignment method, force-directed pin assignment. This is a technique for the pin assignment of multi-FPGA systems which can handle an arbitrary multi-FPGA topology, including routing-only chips, and can produce improvements in both the time to complete a mapping as well as the quality of the results. This provides a 7% quality improvement over the random approach, with mapping time reductions of up to 45%, with an average reduction of 17%. This approach always produces higher quality mappings than all other approaches, and almost always does it in less time than any other technique.

In conclusion, force-directed pin assignment is a valuable technique that shortens the mapping time for multi-FPGA systems while improving routing resource utilization. By adapting to arbitrary topologies, and handling systems with directly-connected logic-bearing FPGAs (the majority of current systems), this tool handles a need currently served by no other approach.

Because of the quality and speed improvements, as well as the very limited opportunities for further quality gains (since our results are quite close to the lower bound), we expect this technique to become an integral part of multi-FPGA mapping software.

Acknowledgments

This paper has benefited greatly from the patience and support of many people, including Patrice Bertin, Duncan Buell, David Lewis, Daniel Lopresti, Brian Schott, Mark Shand, Russ Tessier, Herve Touati, Daniel Wong, and Kazuhisa Yamada. This research was funded in part by the Defense Advanced Research Projects Agency under Contract N00014-J-91-4041. Scott Hauck was supported by an AT&T Fellowship.

List of References

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

[2] M. Butts, J. Batcheller, "Method of Using Electronically Reconfigurable Logic Circuits", *U.S. Patent 5,036,473*, July 30, 1991.

[3] Y. Cai, D. F. Wong, "Optimal Channel Pin Assignment", *IEEE Transaction on Computer-Aided Design*, Vol. 10, No. 11, pp. 1413-1424, Nov. 1991.

[4] 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.

[5] J. Cong, "Pin Assignment with Global Routing for General Cell Designs", *IEEE Transaction on Computer-Aided Design*, Vol. 10, No. 11, pp. 1401-1412, Nov. 1991.

[6] T. H. Drayer, W. E. King IV, J. G. Tront, R. W. Conners, "MORRPH: A MOdular and Reprogrammable Real-time Processing Hardware", *IEEE Symposium on FPGAs for Custom Computing Machines*, 1995.

[7] S. S. Erdogan, A. Wahab, "Design of RM-nc: A Reconfigurable Neurocomputer for Massively Parallel-Pipelined Computations", *International Joint Conference on Neural Networks*, Vol. 2, pp. 33-38, 1992.

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

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

[10] S. Hauck, G. Borriello, "An Evaluation of Bipartitioning Techniques", *Chapel Hill Conference on Advanced Research in VLSI*, pp. 383-402, March, 1995. Journal version to appear in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[11] S. Hauck, G. Borriello, C. Ebeling, "Mesh Routing Topologies for Multi-FPGA Systems", *International Conference on Computer Design*, pp. 170-177, October, 1994. Journal version to appear in *IEEE Transactions on VLSI Systems*.

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

[13] M. Slimane-Kadi, D. Brasen, G. Saucier, "A Fast-FPGA Prototyping System That Uses Inexpensive High-Performance FPIC", *ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, 1994.

[14] D. M. Lewis, M. H. van Ierssel, D. H. Wong, "A Field Programmable Accelerator for Compiled-Code Applications", *International Conference on Computer Design*, pp. 491-496, 1993.

[15] D. P. Lopresti, "Rapid Implementation of a Genetic Sequence Comparator Using Field-Programmable Logic Arrays", *Advanced Research in VLSI 1991: Santa Cruz*, pp. 139-152, 1991.

[16] W.-K. Mak, D. F. Wong, "On Optimal Board-Level Routing for FPGA-based Logic Emulation", *Design Automation Conference*, 1995.

[17] T. Pförtner, S. Kiefl, R. Dachauer, "Embedded Pin Assignment for Top Down System Design", *European Design Automation Conference*, pp. 209-214, 1992.

[18] K. Shahookar, P. Mazumder, "VLSI Cell Placement Techniques", *ACM Computing Surveys*, Vol. 23, No. 2, pp. 145-220, June 1991.

[19] W. Swartz, C. Sechen, "New Algorithm for the Placement and Routing of Macro Cells", *International Conference on Computer-Aided Design*, pp. 336-339, 1990.

[20] D. A. Thomae, T. A. Petersen, D. E. Van den Bout, "The Anyboard Rapid Prototyping Environment", *Advanced Research in VLSI 1991: Santa Cruz*, pp. 356-370, 1991.

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

[22] *The Programmable Logic Data Book*, San Jose, CA: Xilinx, Inc., 1994.

[23] K. Yamada, H. Nakada, A. Tsutsui, N. Ohta, "High-Speed Emulation of Communication Circuits on a Multiple-FPGA System", *2nd International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, 1994.

[24] X. Yao, M. Yamada, C. L. Liu, "A New Approach to the Pin Assignment Problem", *Design Automation Conference*, pp. 566-572, 1988.

Appendix A. Spring Replacement Rules

As discussed earlier, we wish to reduce the complexity of the force-directed placement algorithm by replacing all springs touching non-logic pin nodes with equivalent springs involving only logic pin nodes. To do this, we iteratively remove individual internal (logic function) nodes from the system, and replace the attached springs with new springs between the removed node's neighbors. Since we are using a Manhattan distance metric, the forces along the X and Y dimension are independent, and are given in Equation 2.

$$F_x = C \times (X_1 - X_2) \quad F_y = C \times (Y_1 - Y_2)$$

Equation 2. Spring forces in the X and Y direction.

C is the spring constant, and (X_1,Y_1) and (X_2,Y_2) are the locations of the nodes connected by the spring. There are two simplification rules necessary for our purposes: parallel spring combining, and node removal. For parallel springs, springs which connect the same two endpoints, the springs can be merged into a single spring whose spring constant is the sum of the parallel springs.

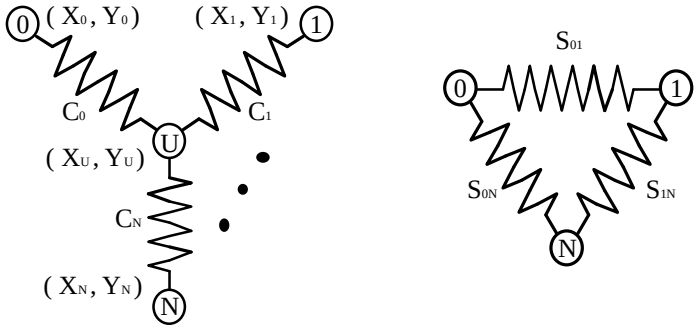

Figure 10. Spring system before logic node **U** is removed (left), and after (right).

In node removal, we remove an internal node, and replace the springs connected to that node with springs connected between the node's neighbors. As shown in Figure 10, we have node **U** with *N+1* neighbors labeled **0**…**N** (note that if **U** has only one neighbor, the node and the spring can simply be removed, since it will never exert force on its neighbor). Node *i* is located at (X_i,Y_i), and is connected to node **U** by a spring with spring constant C_i. As discussed earlier, we assume that internal nodes are always placed at their optimal location, which is the point where the net force on the node is zero. Thus, we can calculate the location of node **U** as shown in Equation 3 and Equation 4 (note that from now on we will work with only the X coordinates of all nodes. Similar derivations can be found for the Y coordinates as well).

$$0 = \sum_{i=0}^{n} C_i \times (X_i - X_U) = \left(\sum_{i=0}^{n} C_i \times X_i \right) - X_U \times \sum_{j=0}^{n} C_j$$

Equation 3. Total forces on node **U**, assuming **U** is placed at its optimal location.

$$X_U = \frac{\sum_{i=0}^{n} C_i \times X_i}{\sum_{j=0}^{n} C_j}$$

Equation 4. Optimal X position of node **U** expressed in terms of its neighbors' positions.

To replace the springs connected to node **U**, we must make sure the new springs provide the same force as the old springs. So, we start with the force equation from Equation 2, and substitute in the location found in Equation 4. The results are Equation 5 and Equation 6.

$$F_k = C_k \times (X_U - X_k) = \frac{C_k \times \left(\sum_{i=0}^{n} C_i \times X_i \right)}{\sum_{j=0}^{n} C_j} - C_k \times X_k$$

Equation 5. Substitution of Equation 4 into Equation 2.

$$F_k = \sum_{i=0}^{n} \left(\frac{C_k \times C_i}{\sum_{j=0}^{n} C_j} \times (X_i - X_k) \right)$$

Equation 6. Simplification of Equation 5.

From Equation 6 it is now clear how to replace the springs incident on node **U**. We can replace all of these springs, and insert a new spring between each pair of neighbors of node **U**. The new spring between nodes *I* and *K* will have a spring constant S_{ik} as given in Equation 7.

$$S_{ik} = \frac{C_k \times C_i}{\sum_{j=0}^{n} C_j}$$

Equation 7. Spring constants for spring replacement, derived from Equation 6.