Experimental Evaluation of Mesh and Partial Crossbar Routing Architectures for Multi-FPGA Systems

Mohammed A. S. Khalid and Jonathan Rose

Department of Electrical and Computer Engineering, University of Toronto
10 King's College Road, Toronto, Ontario, CANADA M5S 3G4
{khalid, jayar}@eecg.toronto.edu

Abstract

Multi-FPGA systems (MFSs) are used as custom computing machines, logic emulators and rapid prototyping vehicles. A key aspect of these systems is their programmable routing architecture, the manner in which wires, FPGAs and Field-Programmable Interconnect Devices (FPIDs) are connected.

In this paper we present an experimental study for evaluating and comparing two commonly used routing architectures for multi-FPGA systems: 8-way mesh and partial crossbar. A set of 15 large benchmark circuits are mapped into these architectures, using a customized set of partitioning, placement and inter-chip routing tools. Particular attention was paid to the development of appropriate inter-chip routing algorithms for each architecture. The architectures are compared on the basis of cost (the total number of pins required in the system) and speed (determined by post inter-chip routing critical path delay). The results show that the 8-way mesh architecture has high cost, poor routability and speed while the partial crossbar architecture gives relatively low cost, good routability and speed.

Using our experimental approach, we also explore a key architecture parameter associated with the partial crossbar architecture, and its impact on the routability and speed of the architecture. We briefly describe an inter-chip router for the partial crossbar architecture, called PCROUTE, that gives excellent routability and speed results for real benchmark circuits.

1 Introduction

Field-Programmable Gate Arrays (FPGAs) are widely used for implementing digital circuits because they offer moderately high levels of integration and rapid turnaround time [Brow92]. Multi-FPGA systems (MFSs), which are collections of FPGAs and memory joined by programmable connections as illustrated in Figure 1, are used when the logic capacity of a single FPGA is insufficient, and when a quickly re-programmed system is desired. The typical uses are for logic emulation [Quic96] [Apti96], rapid prototyping [Van92] [Gall94] [Alte94] [Lewi97] and reconfigurable custom computing machines [Lewi97] [Vuil96] [Arno92] [Cass93] [Dray95].

The routing architecture of an MFS is the way in which the FPGAs, fixed wires, and field programmable interconnect devices (FPIDs) are connected. The routing architecture has a strong effect on the speed, cost and routability of the system. Many architectures have been proposed and built [FCCM] [Butt92] [Van92] [Apti96] [Lewi97] and some research work has been done to empirically evaluate and compare different architectures [Khal97] [Kim96].

In this paper we evaluate and compare two popular architectures, the 8-way mesh and the partial crossbar, using an empirical approach. The architectures are compared on the basis of cost (the total number of pins required in the system) and speed. The speed comparisons are based on post inter-chip routing critical path delay of real benchmark circuits, which, to our knowledge, is the first time such detailed timing information has been used in the study of board-level MFS architectures.

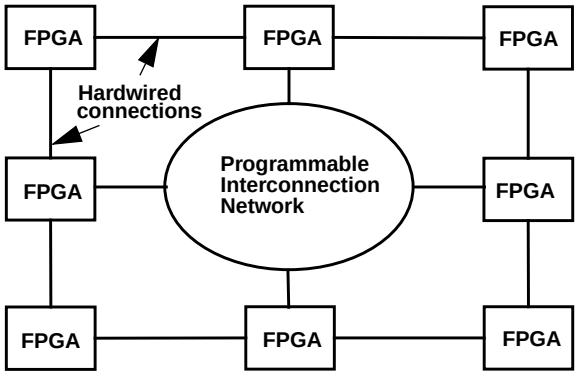

Figure 1 - A Generic Multi-FPGA System

We explore the effect of a key parameter of the partial crossbar architecture, the number of pins per subset (referred to as P_t in the sequel), on routability, cost, and speed. We briefly describe an inter-chip router for the partial crossbar architecture, called PCROUTE, that gives excellent routability and speed results for real benchmark circuits. PCROUTE is equivalent or better in quality compared to other partial crossbar routers that have been proposed so far [Mak97a] [Mak97b] [Lin97] [Slim94].

Previous work has been done evaluating mesh [Hauc94] and other architectures [Chan93]. Although this work provides some theoretical insight into these architectures, empirical studies that evaluate the implementation of real circuits on different architectures provide a more clear picture of the 'goodness' of each architecture relative to the others [Khal97] [Kim96].

This paper is organized as follows: In Section 2 we describe the experimental evaluation procedure and the evaluation metrics used, and give details on the suite of large benchmark circuits used in this experimental work. In Section 3 we cover the architectural issues and assumptions that arise when mapping real circuits to the 8-way mesh and partial crossbar architectures. We also briefly describe architecture-specific inter-chip routing algorithms employed. Experimental results and their analysis is presented in Section 4, and we conclude in Section 5.

2 Experimental Overview

To evaluate the two routing architectures considered in this paper, we used the experimental procedure illustrated in Figure 2. Each benchmark circuit was partitioned, placed and routed into each architecture. Section 2.1 describes the general toolset used in this flow. The cost and delay metrics that we use to evaluate architectures are described in Section 2.2. A description of the 15 benchmark circuits is given in Section 2.3.

2.1 General CAD Flow

As illustrated in Figure 2, we start with a (technology mapped) netlist of 4-LUTs and flip flops of the circuit. The circuit is partitioned into a minimum number of sub-circuits using a multi-way partitioning tool which accepts as constraints the specific FPGA logic capacity and pin count. For all the experiments presented in this paper we used a Xilinx 4013 FPGA, which consists of 1152 4-LUTs, 1152 flip flops, and 192 usable I/O pins [Xili97]. We also assume that each MFS architecture will be implemented on a single board. Multi-way partitioning is accomplished using a recursive bi-partitioning procedure. The partitioning tool is called 'part' and was originally developed for the Transmogrifier-1 rapid prototyping system [Gall94]. It is based on the Fiduccia Mattheyses partitioning algorithm [Fidu82] with an extension for timing-driven pre-clustering [Shih92]. While it is more accurate to do architecture-driven partitioning for the mesh architecture, we believe that recursive bi-partitioning followed by FPGA placement will help in providing enough 'locality' in the post-placement netlist for the mesh architecture.

The next step is system-level placement of each sub-circuit onto a specific FPGA. Given the number of sub-circuit and the netlist of interconnections, each sub-circuit is assigned to a specific FPGA in the MFS. The objective is to place highly connected sub-circuits into adjacent FPGAs (if the architecture has some notion of adjacency) so that the routing resources needed for inter-FPGA connections are minimized.

Given the sub-circuit interconnection netlist and their placement on FPGAs in the MFS, the next step is to route each inter-FPGA net using the most suitable routing path. In the context of MFSs this means that the routing path chosen should be the shortest path (use the minimum number of *hops*) and it should cause the least possible congestion for subsequent nets to be routed. A *hop* is defined as a wire that connects two chips on a board. If the routing attempt is successful, it means that the circuit fits in the specified architecture.

If the routing attempt fails, the partitioning step is repeated after reducing the number of I/O pins per FPGA specified to the partitioner. This usually increases the number of FPGAs needed, and helps routability by decreasing the demand from each FPGA, and providing more "route-through" pins in the FPGAs which facilitate routing. For example, consider a benchmark circuit consisting of 4374 LUTs, 1728 flip flops, and 357 I/Os, mapped to a 8-way mesh. The first mapping attempt partitioned the circuit into 8 sub-circuits (i.e. separate

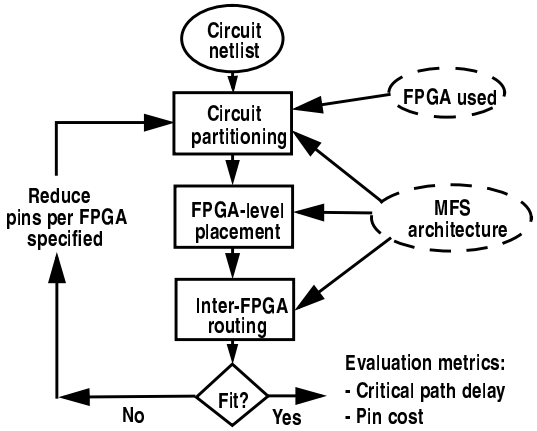

Figure 2 - Experimental Evaluation Procedure for Multi-FPGA Systems

FPGAs). We then placed them on a 2 X 4 mesh of FPGAs followed by inter-FPGA routing. Only 60% of the inter-FPGA nets were routed. The mapping procedure was repeated by reducing the number of pins per FPGA specified to the partitioner, until 100% of the inter-FPGA nets were routed. The circuit was routable on a 3 X 4 array and the number of FPGA I/O pins specified for the partitioner was 100 (out of 192).

For some circuits the routing attempt may fail even after increasing the array size. For such cases, we abandon the fitting attempt when the logic utilization becomes very low after partitioning (15% or less).

Note that in an actual MFS, the inter-FPGA routing step is followed by pin assignment, placement and routing within individual FPGAs. We need not perform these tasks because we are just interested in knowing the MFS size needed to fit the circuit. Our previous research has shown that we can afford to assign pins randomly for each FPGA without jeopardizing routability and speed [Khal95]. During recursive bi-partitioning, we restrict the logic utilization of each FPGA to be less than or equal to 70% to avoid placement and routability problems for individual FPGAs. Thus we ensure that if inter-FPGA routing attempt succeeds, it is almost guaranteed that the subsequent pin assignment, placement, and routing steps will be successful for each FPGA in the MFS.

The inter-chip routing problem is unique for each architecture and this requires an architecture-specific router. We attempted to develop a generic router (FPSROUTE) that can be used for different architectures, but it did not give satisfactory results [Khal97]. Each architecture has unique features that can be exploited by the routing tool to give superior results.

2.2 Evaluation metrics

To compare the two routing architectures we implement benchmark circuits on each and contrast the pin cost and post-routing critical path delay, as described below.

2.2.1 Pin cost

The cost of an MFS is likely a direct function of the number of FPGAs and FPIDs: If the routing architecture is

inefficient, it will require more FPGAs and FPIDs to implement the same amount of logic as a more efficient MFS. While it is difficult to calculate the price of specific FPIDs and FPGAs, we assume that the total cost is proportional to the total number of pins on all of these devices. Since the exact number of FPGAs and FPIDs varies for each circuit implementation (in our procedure above, we allow the MFS to grow until routing is successful), we calculate, for each architecture, the total number of pins required to implement each circuit. We refer to this as the *pin cost* metric for the architecture.

2.2.2 Post-Routing Critical Path Delay

The speed of an MFS, for a given circuit, is determined by the critical path delay obtained after a circuit has been placed and routed at the inter-chip level. We call this the *post-routing critical path delay*. We have developed an MFS static timing analysis tool (MTA) for calculating the post routing critical path delay for a given circuit and MFS architecture.

The operation and modeling used in the MTA are described briefly as follows: It first calculates the critical path delay of the un-partitioned design using a widely used method called the *block oriented technique* [Joup87]. It then reads the inter-FPGA netlist and the routing path for each inter-FPGA net, as provided by the inter-chip router, and the MFS architecture description. From this information the circuit is annotated with the inter-chip delays, from which the critical path delay can be calculated.

In the delay annotation step, the delay values given in Table 1 (obtained from data sheets [Xili97] and [Icub97] and some design experience) are used.

Item	Delay (ns)
Intra-FPGA CLB-to-CLB routing delay	2.5
FPGA input pad delay	1.4
FPGA output pad delay	3.2
CLB delay (without using H-LUT)	1.3
CLB delay (via H-LUT)	2.2
FPID crossing delay (including pad delays)	10
PCB trace delay	3
FPGA Route Through Delay	10

Table 1 - Delays used in Timing Analyzer Model

Note that since we do not perform individual FPGA place and route, we approximate the CLB-to-CLB delay as a constant. The value of 2.5 ns for CLB-to-CLB routing delay is roughly half the delay on a long line for XC4013E-1 FPGA. This is a pessimistic estimate. Although using a single delay value is not accurate, it still gives us a good estimate of the post-routing critical path delay of an MFS because it is dominated by off chip delay values.

2.3 Benchmark Circuits

A total of fifteen large benchmark circuits were used in our experimental work. An extensive effort was expended to collect this suite of large benchmark circuits. The details about each benchmark circuit are shown in Table 2. The table gives the circuit name, size (in 4-LUTs, D flip flops, and I/O count), rough description of the functionality, the source of the circuit and the manner in which it was synthesized. Four circuits were obtained from MCNC [Yang91], two from PREP [Prep], and the remaining nine were developed at the University of Toronto (UofT). The circuits from MCNC were available in the XNF [Xili97] gate-level netlist format required by our front end tools. All the circuits from PREP and UofT were originally available as VHDL or Verilog HDL models and were synthesized into XNF netlists using Exemplar [Exem94] and Synopsys Behavioral Compiler [Knap96] and/or Design Compiler [Syno97] synthesis tools. We show these details of the benchmark circuits because we feel that the MCNC circuits that have been used so far in MFS architecture studies are insufficient in terms of size and variety to 'stress' different architectures and the mapping tools used. Specifically, we found that they are easier to partition and map compared to the other real circuits that we use in this work. Since our benchmark set includes a variety of real circuits, this will make our architectural conclusions more accurate and useful compared to the architectural studies that use synthetic netlists [Hauc94] or use only a small number of large circuits [Kim96]. Also, inter-chip routing tools can be evaluated in a realistic manner using a variety of real benchmark circuits rather than synthetic netlists [Mak97a] [Mak97b] [Lin97].

3 Routing Architecture Description and Routing Algorithms

In this Section we describe the partial crossbar and 8-way mesh architectures. For each architecture, we briefly describe an architecture-specific inter-chip router.

3.1 Architectural Issues Assumptions for the 8-way Mesh

The most simple mesh topology is a 4-way mesh where each FPGA is connected to its horizontally and vertically adjacent neighbours. A variation of this basic mesh topology

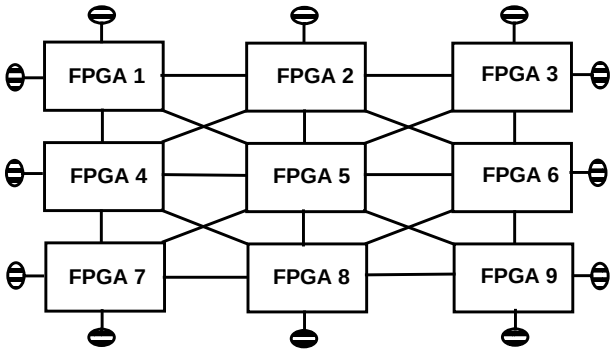

Figure 3 - The 8-way Mesh Architecture

Circuit	Size	Function	Source, Synthesis tool used (if applicable)
s35932	4374 LUTs, 1728 FFs, 357 I/Os	Sequential circuit	MCNC
s38417	6097 LUTs, 1463 FFs, 134 I/Os	Sequential circuit	MCNC
s38584	4396 LUTs 1451 FFs, 292 I/Os	Sequential circuit	MCNC
mips64	2900 LUTs 440 FFs, 260 I/Os	Scaled down version of MIPS R4000	PREP, Verilog model synthesized using Exemplar
spla	3423 LUTs 0 FFs, 62 I/Os	Combinational Circuit	MCNC
cspla	2039 LUTs 0 FFs, 62 I/Os	Clone of spla	UofT, Generated using GEN[Hutt96]
mac64	2560 LUTs 64 FFs, 133 I/Os	64-bit multiply-accumulate ckt.	UofT, Verilog model synthesized using Synopsys
sort8	1540 LUTs 200 FFs, 20 I/Os	8-bit HW sort engine	UofT, Verilog model synthesized using Synopsys
fir16	5366 LUTs 1040 FFs, 60 I/Os	16-bit, 8-stage FIR filter	UofT, Verilog model synthesized using Synopsys
gra	2494 LUTs 1156 FFs, 144 I/Os	Graphics acceleration circuit	UofT, circuit generated using tmcc[Gall95]
fpsdes	3484 LUTs 1008 FFs, 69 I/Os	Fastest pseudo DES circuit	UofT, Verilog model synthesized using Synopsys
spsdes	2452 LUTs 982 FFs, 69 I/Os	Smallest pseudo DES circuit	UofT, Verilog model synthesized using Synopsys
ochip64	3617 LUTs 5810 FFs, 84 I/Os	Output chip for ATM switching chip set	UofT, VHDL model synthesized using Exemplar
ralu32	2553 LUTs 584 FFs, 98 I/Os	32-bit register file, ALU, and control logic	PREP, VHDL model synthesized using Synopsys
iir16	3149 LUTs 522 FFs, 52 I/Os	16-bit IIR filter	UofT, VHDL model synthesized using Synopsys

Table 2 - The Benchmark Circuits Used

is the 8-way mesh [Hauc94] as shown in Figure 3. Each FPGA is connected to its horizontal, vertical, and diagonal adjacent neighbours. We use a Torus topology in which the edges on the peripheral FPGAs are wrapped around in horizontal and vertical directions and are connected to FPGAs on the opposite side of the array. For example, FPGA 1 in is connected to FPGAs 3 and 7. In each FPGA, a certain number of pins are reserved for circuit I/O signals. Note that we use only horizontal and vertical wrap around and do not use wrap around in diagonal directions.

3.1.1 Placement and Routing Tools for the 8-way Mesh

Given a circuit, the experimental procedure for mapping it to a mesh architecture is given in Figure 2. In this section the specific placement and routing tools for the mesh will be briefly described. Given the circuit netlist and the FPGA logic and pin capacities, the circuit is partitioned into a minimum possible number of sub-circuits such that each sub-circuit fits in a single FPGA. The sub-circuits are then placed on the mesh array using a placement tool that is based on a force-directed placement algorithm described in [Shah91]. The objective here is to place closely connected sub-circuits on adjacent FPGAs in the mesh array. A detailed description of

the placement tool is given in [Khal97].

Given the placement of sub-circuits on the mesh and the netlist of interconnections between sub-circuits, the next step is inter-FPGA routing. The routing problem is complicated by the fact that FPGAs are used for both logic and routing. Each FPGA will have a number of I/O pins unused after all the pins needed for sources and sinks in that FPGA are accounted for, called *free pins*. An FPGA should have at least two free pins if it is to permit a route to pass through it.

We have developed a mesh routing tool called MROUTE that uses a heuristic tuned to the routing requirements of the mesh architecture. It first routes all those nets that do not use any free pins. It then routes all two-terminal nets using an algorithm that enumerates all possible shortest paths between source and target. A shortest path is chosen that attempts to minimize the congestion for the subsequent nets to be routed. Since the typical array is small (at most 6 X 8 in our case) and few nets connect FPGAs that are far apart, enumeration of all possible shortest paths is computationally feasible. For multi-terminal nets, a modified form of the single component growth algorithm is used [Kuh86]. The algorithm is adapted for mesh architectures to consider free pins and wire segments when routing multi-terminal nets. MROUTE gives consistently better results compared to an architecture independent multi-pass maze router for MFSs, FPSROUTE, that we had developed earlier.

3.2 Architectural Description and Routing for the Partial Crossbar

The partial crossbar architecture [Butt92] [Varg93] is used in logic emulators produced by Quickturn Design Systems [Quic96]. An example partial crossbar using four FPGAs and three FPIDs is shown in Figure 4. The pins in each FPGA are divided into N subsets, where N is the number of FPIDs in the architecture. All the pins belonging to the same subset in different FPGAs are connected to a single FPID. Note that any circuit I/Os will have to go through FPIDs to reach FPGA pins. For this purpose, a certain number of pins per FPID are reserved for circuit I/Os. The number of pins per subset (P_t) is a key architectural parameter that determines the number of FPIDs needed and the pin count of each FPID. The extremes of the partial crossbar architecture can be illustrated by considering a system with four FPGAs, and assuming 192 usable I/O pins per FPGA: a P_t value of 192 will require a single 768-pin FPID that acts as a full crossbar. A P_t value of 1 will require 192 4-pin FPIDs. Both of these cases are impractical. A good value of P_t should require low cost, low pin count FPIDs. For the above example, a P_t value of 12 will require 16 48-pin FPIDs. Taking into account the extra FPID pins required for circuit I/Os we will need to use 64 or 96-pin FPIDs that are commercially available [Icub97].

3.2.1 Routing Algorithm for the Partial Crossbar

Given a partitioned circuit, the placement problem is trivial in a partial crossbar because there is no locality inherent in the routing architecture. Therefore it does not matter which FPGA is used for a given sub-circuit. It is still important to use a routing algorithm that exploits architecture-specific features in order to obtain good results. Ideally, a routing algorithm for the partial crossbar should achieve 100% routability and ensure that each source-sink path across all inter-chip nets uses no more than two inter-chip hops. This will give the minimum possible net delay for each net and minimize the post-routing critical path delay.

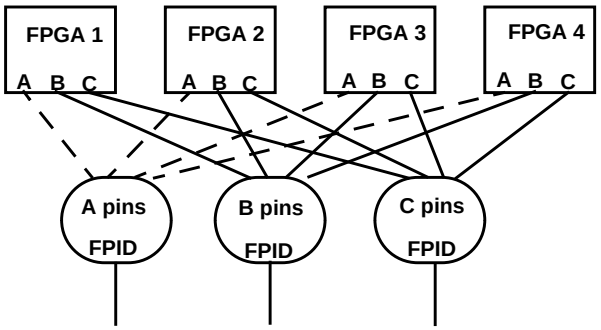

Figure 4 - The Partial Crossbar Architecture

We have developed a routing tool, called PCROUTE, for partial crossbar architecture that gives excellent results for all the circuits. Irrespective of the value of P_t, it achieves 100% routing completion and produces two-hop routing for all the nets in almost all circuits. For only two circuits, for the specific case of P_t= 4, it produced multi-hop routing paths for a negligible number of nets (1 out of 991 nets for the first circuit and 3 out of 645 nets for the second). In practical terms, this means it gives almost optimal results for all of our benchmark circuits. Note that the routing problem becomes more difficult for the partial crossbar as the value of P_t is reduced. This is due to reduced routing flexibility due to the low pin count FPIDs used in such cases.

The PCROUTE algorithm works as follows: for each net (irrespective of fanout), it evaluates paths through all available FPIDs. It uses a suitable cost function to choose an FPID that will guarantee balanced usage of FPIDs and will preserve the most options for two-hop routing of subsequent nets to be routed (congestion avoidance). Further details about PCROUTE are given in [Khal97]. We also show in [Khal97] that PCROUTE is equivalent in quality to other partial crossbar routers that have been proposed so far [Kim96] [Mak97a] [Lin97] [Slim94]. PCROUTE is better than [Mak97b] for speed because that algorithm splits each multi-terminal into a set of two-terminal nets and routes them independently, leading to multiple hops and even possible routing failures.

4 Experimental Results

In this section we determine the effect of varying the value of P_t on the routability and speed of the partial crossbar architecture and compare the partial crossbar and the 8-way mesh architectures.

4.1 Partial Crossbar Architecture: Analysis of P_t

Recall the definition of P_t, the number of pins per subset, given in Section 3.2. P_t is important because, depending on its size either very large FPIDs are needed or very many FPIDs are required. Here we explore the effect of P_t on the routability and speed of the partial crossbar architecture. We mapped the

Circuit	# FPGAs	Normalized post-routing critical path delay using PCROUTE, P_t = 4, 17, 47	Normalized post-routing critical path delay using FPSROUTE		
			P_t = 47	P_t = 17	P_t = 4
s35932	8	1.0	1.0	1.42	1.42
s38417	9	1.0	1.0	1.27	1.27
s38584	9	1.0	1.0	1.17	1.17
mips64	14	1.0	1.0	1.00	1.09
spla	18	1.0	1.38	1.46	1.62
cspla	18	1.0	1.24	1.24	1.36
mac64	6	1.0	1.0	1.0	1.0
sort8	12	1.0	1.09	1.14	1.22
fir16	10	1.0	1.0	1.03	1.03
gra	4	1.0	1.0	1.0	1.0
fpsdes	9	1.0	1.0	1.0	1.24
spsdes	8	1.0	1.0	1.0	1.10
ochip64	8	1.0	1.0	1.0	1.0
ralu32	9	1.0	1.0	1.04	**Routing failure**
iir16	6	1.0	1.0	1.0	1.00
Average		1.0	1.05	1.12	1.19

Table 3 - The Effect of P_t on the Delay of Partial Crossbar Architecture

fifteen benchmark circuits to the partial crossbar architecture using three different values of P_t (4, 17, 47). The values 4 and 47 are extreme cases (resulting in either many small FPIDS or few very large FPIDs) and the value of 17 is a reasonable choice as discussed in Section 3.2.

We used two routing algorithms to do these experiments: FPSROUTE, which employed a somewhat generic maze-routing algorithm, and PCROUTE, which used a algorithm that specifically addressed the nature of a partial crossbar. The first clear conclusion is that P_t has no significant impact on routability of partial crossbar, because all the circuits were routable by PCROUTE for the P_t values given above. The same was true for FPSROUTE as well except for routing failures in one circuit for the P_t = 4 case. An interesting point that follows from this conclusion is that we do not need links between FPIDs, as proposed in [Icub94], to improve the routability of the partial crossbar.

The effect of P_t on the speed of partial crossbar is shown in Table 3. The first column shows the circuit name. The second column gives the number of FPGAs needed to implement the circuit. The third column shows the normalized post-routing critical path delay obtained for the circuit using PCROUTE for three values of P_t (4, 17, 47). The critical path delay obtained by PCROUTE is set as 1. The columns 4-6 show the normalized post-routing critical path delay obtained using FPSROUTE for three values of P_t (4, 17, 47).

Observe that the PCROUTE algorithm, which is tuned for partial crossbars, gives the same delay value irrespective of the value of P_t. This shows that it is able to tackle the increased complexity of the routing task when we use very small values of P_t without any adverse effects on routability or speed.

We include the results for FPSROUTE to warn of the danger of using an inappropriate algorithm for the partial crossbar: here the effect of P_t on speed is quite significant, the delay increases as the value of P_t decreases. For P_t = 4, average increase in delay using FPSROUTE is 19% across all circuits, and up to 62% more. The partial crossbar is a very robust architecture that allows us to use a wide range of P_t values without any penalties on routability and speed

4.2 Comparison of 8-way Mesh and Partial Crossbar

In this section we compare the 8-way mesh architecture

Circuit	Number of FPGAs		% nets routed		Pin cost		Post-routing critical path delay (in nano seconds)	
	8-way mesh	Partial crossbar	8-way mesh	Partial crossbar	8-way mesh	Partial crossbar	8-way mesh	Partial crossbar
s35932	12 (3 X 4)	8	100	100	2304	3428	50.5	57.4
s38417	12 (3 X 4)	9	100	100	2304	3807	123.6	94.6
s38584	30 (5 X 6)	9	100	100	5760	3807	216.9	139.4
mips64	> 48 (6 X 8)	14	92	100	> 9216	5646	Rout. failure	461.9
spla	> 48 (6 X 8)	18	90	100	> 9216	7218	Rout. failure	196.3
cspla	> 40 (5 X 8)	18	85	100	> 7680	7218	Rout. failure	192.5
mac64	> 18 (3 X 6)	6	77	100	> 3456	2760	Rout. failure	622.9
sort8	> 28 (4 X 7)	12	80	100	> 5376	4944	Rout. failure	532.8
fir16	> 25 (5 X 5)	10	96	100	> 4800	4944	Rout. failure	238
gra	4 (2 X 2)	4	100	100	768	1912	60	70
fpsdes	> 18 (3 X 6)	9	88	100	> 3456	3807	Rout. failure	226.5
spsdes	> 15 (3 X 5)	8	84	100	> 2880	3428	Rout. failure	248.8
ochip64	8 (2 X 4)	8	100	100	1536	3428	46.7	63.2
ralu32	> 30 (5 X 6)	9	87	100	> 5760	3807	Rout. failure	316.8
iir16	> 15 (3 X 5)	6	89	100	> 2880	2760	Rout. failure	160.2
			Avg.: 91.2	Avg.: 100	Total: > 67392	Total: 62914		

Table 4 - Comparison of the 8-way Mesh and Partial Crossbar Architectures

with partial crossbar. Table 4 presents the results obtained after mapping fifteen benchmark circuits to the 8-way mesh and partial crossbar architectures. The first column shows the circuit name, the second column shows the number of FPGAs needed to implement the circuit, and the third column shows percentage of nets routed, in each architecture. The fourth and fifth columns show pin cost and post-routing critical path delay respectively, obtained for each architecture. All results for partial crossbar are for a P_t value of 17. This implies that the number of FPIDs used will be 11 but the size of FPID will depend on the number of FPGAs used. We could have used any suitable value of P_t, since it does not affect routability and speed. However, we choose a P_t value of 17 because it requires an FPID of a realistic size (< 400 pins) for the largest circuit that we mapped to the partial crossbar (18 FPGAs).

Notice that the partial crossbar is always routable for the first feasible partition of each circuit. Only five of the fifteen benchmark circuits were routable on the 8-way mesh even after mapping attempts with increased array sizes. The fact that so few circuits successfully routed indicates a basic flaw with the mesh architectures.

The mesh architectures failed for the majority of circuits due to a number of reasons. First, the locality available in inter-FPGA netlists for real circuits is not great enough for the nearest neighbor connections. Second, there are not enough free pins available for routing the non-local nets. To make matters worst, multiple hops needed to route many nets use up many precious free pins. It was initially surprising to find no success when the MFS was expanded in an attempt to obtain 100% routing completion. Clearly the larger MFS has more free pins. However, since the array was larger, this in turn leads to increase in average wire length and more inter-FPGA nets, partially nullifying the advantage of increased free pins.

Clearly, the partial crossbar architecture is superior to the 8-way mesh architecture. This is not surprising, considering that Quickturn [Quic96] initially used 8-way mesh in their first generation logic emulator called RPM [Walt91], but dropped it later and used partial crossbar [Butt92] for their next generation emulators.

The delay results show that for small array sizes, the 8-way mesh gives better speed than the partial crossbar. This is because some or all the nets on the critical paths may utilize

direct connections between FPGAs that are faster than connections that go via FPIDs. But the speed deteriorates as the array sizes get bigger.

For the same number of FPGAs, the partial crossbar always needs twice as many pins as an 8-way mesh. For four out of 15 circuits the 8-way mesh has less pin cost compared to the partial crossbar architecture. But the pin cost as well as the delay over all the circuits will be more for the 8-way mesh if we consider the number of FPGAs needed for the large array sizes that will be required to make some circuits routable.

5 Conclusions and Future Work

In this paper we evaluated and compared two commonly used MFS routing architectures using an experimental approach. To our knowledge, this is the first architecture study of board-level MFSs that considers post-routing critical path delay when evaluating the speed performance of different architectures. We have shown that the partial crossbar is superior to the 8-way mesh architecture in terms of pin cost, speed, and routability. The reason behind inferior results for the mesh architectures is that FPGAs are used for both logic and routing. This causes routability problems that cannot be solved even after increasing the size of the mesh.

The partial crossbar is a very robust architecture. The effect of varying a key architectural parameter (P_t) on the routability, speed, and cost is minor. It is important, however, to use an appropriate routing algorithm for the partial crossbar to obtain these results. We presented a routing algorithm for the partial crossbar (PCROUTE) that gives excellent routability and speed results for real benchmark circuits.

Mesh architectures should be avoided if the goal is to implement a wide variety of circuits on MFSs. We note that meshes and linear arrays have been used successfully in practice [Vuil96][Arno92], but only for implementing algorithms that require nearest neighbor type of connections when implemented. The disadvantage of partial crossbar is that it uses extra pins for some inter-FPGA nets that can be routed using direct connections between FPGAs. This also leads to a delay penalty. We have proposed an alternative architecture that uses both FPIDs and direct connections between FPGAs [Khal98] to give lower pin cost and delay compared to the partial crossbar.

Acknowledgments

The authors would like to thank Dave Galloway for his help with the partitioning tool and Jason Anderson for his help in synthesizing the benchmark circuits. This research was supported by the Information Technology Research Center (ITRC) of Ontario and MICRONET.

References

[Alte94] Altera Corporation, Reconfigurable Interconnect Peripheral Processor (RIPP10) Users Manual, Version 1.0, 1994.

[Apti96] Aptix Corporation, Product brief: The System Explorer MP4, 1996. Available on Aptix Web site: http://www.aptix.com.

[Arno92] J. M. Arnold, D. A. Buell, and E. G. Davis, “Splash 2,” Proceedings of 4th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 316-322, 1992.

[Brow92] S. Brown, R. Francis, J. Rose, and Z. Vranesic, Field Programmable Gate Arrays, Kluwer Academic Publishers, 1992.

[Butt92] M. Butts, J. Batcheller, and J. Varghese, “An Efficient Logic Emulation System,” Proceedings of IEEE International Conference on Computer Design, pp. 138-141, 1992.

[Cass93] S. Casselman, “Virtual Computing and The Virtual Computer,” Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, pp. 43-48, 1993.

[Chan93] P. K. Chan, M. D. F. Schlag, “Architectural Tradeoffs in Field-Programmable-Device-Based Computing Systems,” Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, pp. 152-161, 1993.

[Dray95] T. H. Drayer, W. E. King, J. G. Tront, and R. W. Conners, “MORRPH: A Modular and Reprogrammable Real-time Processing Hardware,” Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, pp. 11-19, 1995.

[Exem94] Exemplar Logic, VHDL Synthesis Reference Manual, 1994.

[Fidu82] C. M. Fiduccia, and R. M. Mattheyses, “A Linear-Time Heuristic for Improved Network Partitions”, Proc. of 19th ACM/IEEE Design Automation Conference, pp. 241-247, 1982.

[FCCM] Proceedings of IEEE Workshops/Symposia on FPGAs for Custom Computing Machines, 1992 to 1996.

[Gall94] D. Galloway, D. Karchmer, P. Chow, D. Lewis, and J. Rose, “The Transmogrifier: The University of Toronto Field-Programmable System”, CSRI Technical Report (CSRI-306), CSRI, University of Toronto, 1994.

[Gall95] D. Galloway, “The Transmogrifier C Hardware Description Language and Compiler for FPGAs,” Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines, pp. 136-144, 1995.

[Hauc94] S. Hauck, G. Boriello, C. Ebeling, “Mesh Routing Topologies for FPGA Arrays”, Proceedings of FPGA’94, 1994.

[Hutt96] M. Hutton, J.P. Grossman, J. Rose and D. Corneil, “Characterization and Parameterized Random Generation of Digital Circuits,” Proc.of the Design Automation Conference, pp. 94-99, 1996.

[Icub97] I-Cube, Inc., The IQX Family Data Sheet, May 1997. Available at: www.icube.com.

[Icub94] I-Cube, Inc., “Using FPID Devices in FPGA-based Prototyping,” Application note, Part number:D-22-002, February 1994.

[Joup87] Norman P. Jouppi, “Timing Analysis and Performance Improvement of MOS VLSI Designs,” IEEE Trans. on CAD, vol. CAD-6, no. 4, pp. 650-665, July 1987.

[Khal95] M. A. S. Khalid and J. Rose, “The Effect of Fixed I/O Pin Positioning on The Routability and Speed of FPGAs,” Proceedings of The Third Canadian Workshop on Field-Programmable Devices (FPD’95), pp. 92-104, 1995.

[Khal97] M. A. S. Khalid, Routing Architecture and Layout Synthesis for Multi-FPGA Systems, Ph.D. Thesis, University of Toronto, in progress.

[Khal98] M. A. S. Khalid and J. Rose, “A Hybrid Complete-Graph Partial-Crossbar Routing Architecture for Multi-FPGA Systems,” to be published in the Proc.

of 1998 Sixth ACM International Symposium on Field-Programmable Gate Arrays (FPGA'98).

[Kim96] C. Kim, H. Shin, "A Performance-Driven Logic Emulation System: FPGA Network Design and Performance-Driven Partitioning," IEEE Trans. on CAD, vol. 15, no. 5, pp. 560-568, May 1996.

[Knap96] D. W. Knapp, Behavioral Synthesis: Digital System Design Using the Synopsys Behavioral Compiler, Prentice Hall PTR, 1996.

[Kuh86] E. S. Kuh and M. Marek-Sadowska, "Global Routing," in Layout Design and Verification, Edited by T. Ohtsuki, North-Holland, pp. 169-198, 1986.

[Lewi97] D. M. Lewis, D. R. Galloway, M. Van Ierssel, J. Rose, and P. Chow, "The Transmogrifier-2: A 1 Million Gate Rapid Prototyping System," Proceedings of FPGA'97, pp. 53-61, 1997.

[Lin97] S. Lin, Y. Lin, and T. Hwang, "Net Assignment for the FPGA-Based Logic Emulation System in the Folded-Clos Network Structure," IEEE Trans. on CAD, vol. 16, no. 3, pp. 316-320, March 1997.

[Mak97a] Wai-Kei Mak, D. F. Wong, "On Optimal Board-Level Routing for FPGA-based Logic Emulation," IEEE Trans. on CAD, vol. 16, no. 3, pp. 282-289, March 1997.

[Mak97b] Wai-Kei Mak, D. F. Wong, "Board-Level Multi-Terminal Net Routing for FPGA-based Logic Emulation," ACM Trans. on Design Automation of Electronic Systems, vol. 2, no. 2, pp. 151-167, April 1997.

[Prep] Programmable Electronics Performance Corporation, HDL models for different circuits are available on their Web site: http//www.prep.org.

[Quic96] Quickturn Design Systems, Inc., System Realizer Data Sheet, 1996. Available on Quickturn Web site:http://www.quickturn.com.

[Shah91] K. Shahookar and P. Mazumder, "VLSI Cell Placement Techniques," ACM Computing Surveys, Vol. 23, No. 2, pp. 143-220, June 1991.

[Shih92] M. Shih, E. S. Kuh, "Performance-Driven System Partitioning on Multi-Chip Modules," Proc. of the Design Automation Conference, pp. 53-56, 1992.

[Slim94] M. Slimane-Kadi, D. Brasen, G. Saucier, "A Fast-FPGA Prototyping System That Uses Inexpensive High-Performance FPIC", Proceedings of FPGA'94, 1994.

[Syno97] Synopsys, Inc., Design Compiler(Version 3.4a), Behavioral Compiler (Version 3.4a), and Library Compiler (Version 3.4a), Reference Manuals. Documents available on-line.

[Van92] D. E. Van Den Bout, et al, "Anyboard: An FPGA-Based Reconfigurable System," IEEE Design and Test of Computers, pp. 21-30, June 1992.

[Varg93] J. Vargese, M. Butts, and Jon Batcheller, "An Efficient Logic Emulation System", IEEE Trans. on VLSI Systems, vol. 1, no. 2, pp. 171-174, June 1993.

[Vuil96] J. E. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, and P. Boucard, "Programmable Active Memories: Reconfigurable Systems Come of Age," IEEE Transactions on VLSI, Vol 4, No. 1, pp. 56-69, March 1996.

[Walt91] S. Walters, "Computer-aided Prototyping for ASIC-Based Systems," IEEE Design and Test, pp 4-10, June 1991.

[Xili97] Xilinx, Inc., Product Specification: XC4000E and XC4000X Series FPGAs, Version 1.2, June 16, 1997. Available on Xilinx Web site: www.xilinx.com.

[Yang91] S. Yang, Logic Synthesis and Optimization Benchmarks User Guide, Version 3.0, Microelectronics Center of North Carolina, January 1991.