

The Routability of Multiprocessor Network Topologies in FPGAs

Manuel Saldaña, Lesley Shannon and Paul Chow
Dept. of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada, M5S 3G4
{msaldana,lesley,pc}@eecg.toronto.edu

ABSTRACT

A fundamental difference between ASICs and FPGAs is that wires in ASICs are designed such that it matches the requirements of a particular design. Wire parameters such as: length, width, layout and the number of wires are designed to meet the performance requirements. Oppositely, in an FPGA, the area is fixed and the routing resources exist whether or not they are used, so the goal changes to meeting the performance requirements within the limits of the available resources.

In this paper, we investigate how this fundamental difference of resource usage affects the choice of network topology when building a Network-on-Chip for an FPGA. By exploring the routability of different multiprocessor network topologies on a single FPGA, we show that the underlying FPGA routing architecture does not benefit a particular topology. We also derive a cost metric that help us estimate the impact of the topology selection beyond 32 processors following a standard design flow and targeting commercial FPGAs.

Categories and Subject Descriptors

C.1.2 [Processor Architectures]: Multiple Data Stream Architectures (Multiprocessors); D.0 [Computer Systems Organization]: General

General Terms

Design

Keywords

Multiprocessor, FPGA, Network-on-Chip, Topology

1. INTRODUCTION

With the growing complexity of System-on-Chip (SoC) circuits, more sophisticated communication schemes are re-

quired to connect the increasing number and variety of intellectual property (IP) blocks. Approaches like AMBA [1], CoreConnect [2], WISHBONE [3] and SiliconBackplane [4] follow a shared bus scheme that works well for Master-Slave communication patterns where there are peripherals (slaves) that wait for data to be received or requested from a more complex processing IP (master). When there are several masters (e.g., processors) in the system, synchronization, message passing and I/O may saturate the bus, and contention will slow down data transfers.

The Network-on-Chip (NoC) [5, 6] provides a possible solution for this problem by creating a scalable interconnection scheme. The concept uses a set of buses connected to routers or switches that interchange packets, much in the same way as traditional computer networks or multiprocessor machines do. Consequently, NoC approaches have design parameters and properties similar to traditional networks. One of these parameters is the topology, which defines the interconnection pattern between the routers and switches.

Multiple topologies have been studied for NoCs on ASICs [7, 8]. A popular choice is the mesh [6, 9] because it provides structure, modularity, and better control over electrical characteristics. Moreover, a mesh topology is said to be orthogonal and therefore packet routing is simple. However, these advantages are clear for ASICs, but not necessarily for FPGAs [10]. The electrical characteristics of the FPGA are solved by the chip vendor, not by the user. As for structure and modularity it is perhaps intuitive to use a mesh topology in FPGAs since the reconfigurable fabric layout is in the form of a mesh. Nevertheless, the placement and routing of components on an FPGA will not typically result in a symmetric, well-organized structured layout that resembles a mesh. Furthermore, manually restricting the placement of components or routing of nets may lead to inefficient resource utilization for the logic that is not part of the network. Finally, there are other orthogonal topologies like hypercube or torus networks, or even tree topologies that also have a simple routing algorithm.

In this paper, we compare the routability of point-to-point network topologies on FPGAs by measuring the impact of each topology on a soft multiprocessor system implemented in commercial and modern FPGAs. We do this by measuring the logic utilization, logic distribution (area), maximum clock frequency, number of nets, and the place and route time for five different network topologies. We also derive a cost metric to try and extract trends for larger systems.

The rest of this paper is organized as follows. Section 2

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

describes the topologies implemented and a brief description of the computing node block. Section 3 describes the experimental framework used, and how the systems are generated. Section 4 presents the results obtained for the baseline system and Section 5 explores the chip area required for each topology. Section 6 shows the highest frequency that each system could achieve. Section 7 presents a metric we propose to evaluate the topologies and Section 8 provides some concluding remarks.

2. THE NETWORK-ON-CHIP

The objective of this work is to investigate how various multiprocessor network topologies utilize the resources of an FPGA, focusing mainly on the routing resources, i.e., wires. To do this, we create an infrastructure that allows us to investigate the impact of the topology selection on the overall design with respect to the FPGA resource utilization and system clock speed. The focus of this paper is not on the network interface itself, nor on the computing node, but on the interconnection patterns and how well they route in an FPGA fabric using a standard design flow.

2.1 Related Work

Brebner and Levi [10] discuss NoC implementations on FPGAs, but their focus is on the issues of using packet switching on a mesh topology in the FPGA and on implementing crossbar switches in the routing structure of the FPGA. Most NoC work assumes ASIC implementations and there are numerous studies including work on mesh topologies [6, 9] and fat trees [7]. Typically, studies on NoCs are done using register-transfer-level simulations [7] and simulation models [11].

In our work, we are not modeling NoCs to study bandwidth, latency, or architectures for packet routing. Instead, we create actual implementations by performing synthesis, mapping, placement and routing for real FPGAs using commercial tools. We try five different topologies and three different system sizes (8, 16 and 32 nodes) to examine how well the topologies can be mapped using an FPGA routing fabric.

Research has been done on synthesizing application-specific network topologies [12]. An approach, such as SIMMPL [13], is a mechanism for realizing a particular topology. In our work, we created a design flow and tools to automatically generate systems using a set of well known topologies.

Based on the philosophy of routing packets, not wires [9, 14], NoC architectures have been proposed as packet-switching networks, with the network interface itself being the focus of much of the research.

In this paper, we use a simple network interface, more similar to a network hub than a switch as it does not provide packet forwarding. Packets can only be sent to, and received from nearest-neighbor nodes. This makes the network interface extremely simple, but it is sufficient for our purposes as the focus of this work is on the routability of various topologies, not on the switching element architecture.

2.2 Network Topologies

Networks can be classified into two categories. Static networks consist of point-to-point, fixed connections between processors, and dynamic networks which have active elements, such as switches, that can change the connectivity

Table 1: Characteristics of the topologies studied

Topology	Diameter	Link Complexity	Degree	Regular	Bisection Width
ring	$\frac{n}{2}$	n	2	yes	2
star	2	$n - 1$	$1, n - 1$	no	1
mesh	$2(n^{1/2} - 1)$	$2(n - n^{1/2})$	2,3,4	no	$2\sqrt{n}$
hypercube	$\log_2 n$	$\frac{n \log_2 n}{2}$	$\log_2 n$	yes	$\frac{n}{2}$
fully connected	1	$\frac{n(n-1)}{2}$	$n - 1$	yes	$\frac{n^2}{4}$

pattern in the system according to a protocol. In an FPGA, the network can be dynamically reconfigured to adapt to communication patterns by utilizing the reconfigurability [15] of the FPGA.

In this paper, we focus on static message passing networks. The ring, star, mesh, hypercube and fully-connected topologies are selected as a representative sample, ranging from the simplest ring topology to the routing-intensive fully-connected system.

Network topologies can be characterized by a number of properties: node degree, diameter, link complexity, bisection width and regularity [16]. Node degree is the number of links from a node to its nearest neighbors. Diameter is the maximum distance between two nodes. Link complexity is the number of links the topology requires. A network is deemed to be regular when all the nodes have the same degree. Bisection width is the number of links that must be cut when the network is divided into two equal set of nodes. Table 1 shows a summary of these characteristics for each of the topologies used in this paper.

The characteristics of the network topology define the network interface of a node. For example, the four-dimensional hypercube is a regular topology, with all nodes having a degree of four. This means that this topology requires a single network interface type, each with four ports, i.e., four communication links. The network interface is used to communicate with other nodes in the network. The maximum distance (diameter) is four, which means that data going through the network may require redirection or routing at intermediate nodes and travel on up to four links. The link complexity is 32, which is the total number of point-to-point links that the overall system will have. In contrast, a 16-node mesh has a total of 24 links in the system, but it is not a regular topology, requiring three different versions of the network interface. Inner nodes require an interface with four ports, perimeter nodes require one with three ports and corner nodes use a two-port interface.

Figure 1 shows examples of the systems with different numbers of nodes and topologies that were implemented to carry out our experiments. Every topology can be seen as a graph that is made of edges (wires) and vertices (computing nodes). In our implementations, the links are 64 bits wide with 32 bits used for transmission and 32 bits used for reception, making it a full-duplex communication system. The links also include control lines used by the network interface.

2.3 Computing Node

The computing nodes in Figure 1 consist of a computing element and a network interface module. Figure 2 shows a computing node. The master computing node of the system

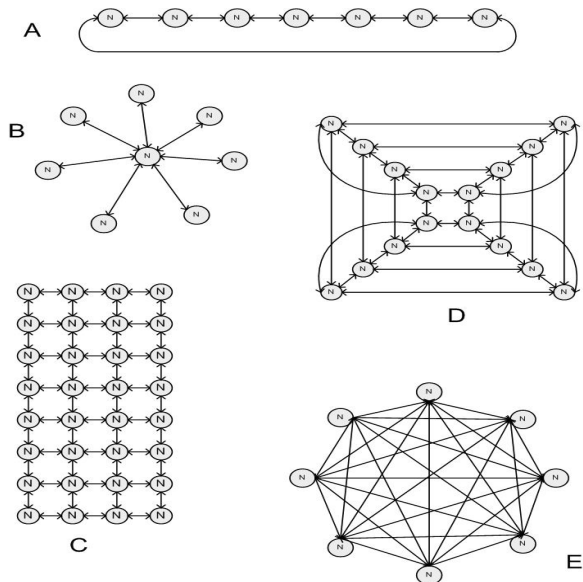


Figure 1: A) 8-node ring, B) 8-node star, C) 32-node mesh, D) 16-node hypercube, and E) 8-node fully-connected topology

is configured to communicate with the external world using a UART attached to the peripheral bus shown inside the dashed box of Figure 2. The rest of the nodes have no peripheral bus.

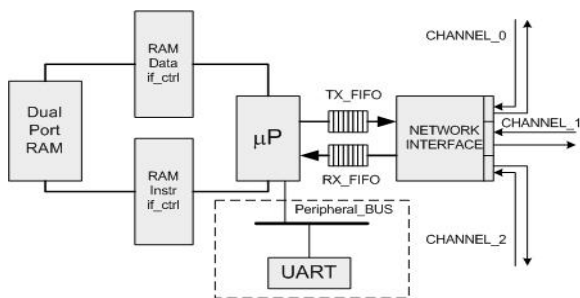


Figure 2: The computing node

We use a Harvard architecture soft core processor as the computing element so that data memory and program memory are accessed by independent memory buses. The communication between the computing element and the network interface is achieved by using two FIFOs: one for transmission and one for reception, 32 bit each.

The network interface module is an extremely simple block that has two sides: the network side, which is the interface to the network; and the processor side, which connects to the processor. On the network side, there are several links (channels) according to the degree of the node. There are two FIFO ports on the processor side. The FIFOs are used as message buffers for the processor.

The network interface is basically a hub that broadcasts the data to the neighbors on transmission, and it filters out the data from the neighbors on reception. It is effectively a FIFO multiplexer that is controlled by the destination field in the packet header. If the destination value matches

the processor's ID number, then the packet passes through the hub to the processor attached to the hub. Again, this interface is simple, but is enough for the purpose of this research, since we are interested in the connectivity pattern.

Implementing a single version of the network interface would not allow us to measure the difference in logic utilizations between the various topologies because the topologies requiring nodes of lesser degree should use less logic. It is likely that the optimizer in the synthesis tool would remove the unused ports, still allowing the study to be performed, but we chose to actually implement the different node degrees required to be certain that only the necessary logic was included.

The size of the remaining logic in the computing node is independent of the node degree i.e., the logic in the processors, the FIFOs, the memory controllers, the UART are independent of the topology selection.

3. IMPLEMENTATION PLATFORM

This section describes the platform and tools used to build the systems and conduct the experiments.

To build the net list, map the design, place it and route it, we use the Xilinx EDK tools version 7.1i [17] in combination with the Xilinx XST synthesis tool. To visualize the placement of the systems, we use the Xilinx FPGA Floorplanning tool. The network interface is developed in VHDL and simulated using ModelSim version 6.0b [18]. The routed nets are counted with the help of the Xilinx FPGA Editor. For section 6, we use the Xilinx Xplorer utility to try to meet the timing constraints. All the experiments are executed on an IBM workstation with a Pentium 4 processor running at 2.8 GHz with Hyperthreading enabled and 2 GB of memory.

Our multiprocessor systems use the Xilinx MicroBlaze soft-processor core [17] as the computing element. The computing element connects to the network interface through a Fast Simplex Link (FSL), a Xilinx core, which is a unidirectional, point-to-point, communication bus implemented as a FIFO.

We use a variety of Xilinx chips to implement the designs: the Virtex2 XC2V2000, and the Virtex4 XC4VLX25, XC4VLX40, XC4VLX60 and XC4VLX200. The LX version of the Virtex4 family only has Block RAM (BRAM) and DSP hard cores in addition to the FPGA fabric. They do not have PowerPC processors or Multi-gigabit Transceivers (MGTs). This provides a more homogeneous architecture that facilitates area comparisons.

The hard multiplier option for the MicroBlaze is disabled to minimize the impact of hard core blocks that may influence or limit the placement and routing. The BRAM are hard core blocks that also affect placement and routing, but they are essential for the MicroBlaze system to synthesize so they have not been eliminated.

A 32-node, fully-connected system requires 1056 links to be specified, and doing this manually is time consuming and error prone. Instead, we developed a set of tools that take a high-level description of the system that specifies the topology type, the number of nodes, the number of total links and the number of links per node, and they generate the files required by EDK.

The number of nodes is chosen based on the limitation of the hypercube to 2^d nodes, where d is the dimension. For $d = 3, 4$ and 5 we have 8, 16 and 32 nodes, respectively. These values are used for all topologies.

4. BASELINE SYSTEM

The main objective of this experiment is to measure the logic and routing resources required for each of the topologies. The timing constraints are chosen to be realistic, but not aggressive, so that the place and route times are not excessive. The 8 and 16-node systems are specified to run at 150 MHz and the 32-node systems are specified to run at 133 MHz to account for the slower speed grade of the XC4VLX200 chip that is used for those systems.

The logic resource usage is measured by counting the total number of LUTs required for a design and the number of LUTs related to only the interconnection network, i.e., those used to implement the network interface modules. The logic resources needed to implement the network are estimated by first synthesizing the network interface modules as stand-alone blocks to determine the number of LUTs required. These numbers are then used to estimate the usage of the entire network. For example, the 8-node star topology requires one 7-port network interface, which uses 345 LUTs, and seven 2-port network interfaces, which need 111 LUTs each. The total number of LUTs required by the network is $345 + (7 \times 111) = 1122$ LUTs. Note that this is only an estimate as the values reported by synthesizing the stand-alone block level may change at the system level due to optimizations that may occur. The register (flip flop) utilization is found by using the same method as used for finding the logic resource utilization.

The routing resource utilization is measured by counting the total number of nets in the design and the number of nets used to implement only the network links and network interfaces. The counting of nets is done by using the Xilinx FPGA Editor, which allows the user to filter out net names. The number of nets attributed to the network is found by counting the number of nets related to all the network interface modules in the design. This includes all nets that are used in the network interface module as well as the nets in the network topology itself. Including the nets in the network interface module is reasonable because more complex topologies use more complex network interfaces that also consume FPGA routing resources.

4.1 Results

Figure 3 shows a histogram of the number of LUTs needed to implement the complete systems, including the MicroBlaze, FSLs, memory interface controllers, switches, UART, and OPB bus. As expected, the system with the fully-connected network has the highest logic utilization, and as the system size increases, the difference with respect to the other topologies gets more pronounced because of the $O(n^2)$ growth in size. The difference is most significant with the 32-node system, which requires over twice the logic of the other systems. For the other topologies, the maximum difference in LUT usage amongst the topologies at the same node size ranges from about 5% in the 8-node systems to about 11% in the 32-node systems.

A more detailed view of the logic resources can be seen in Table 2. The *Logic Utiliz.* column is the total number of LUTs used for each design and these are the values shown in Figure 3. Since the ring has the simplest routing topology, it is used as the baseline for comparisons with the rest of the topologies.

Column *Logic Incr.* shows the increase in the number of LUTs for each topology relative to the ring topology. For

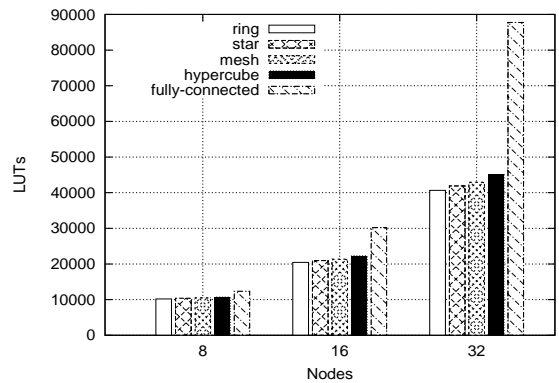


Figure 3: Logic utilization of systems

Table 2: Logic and register resources used by each system

Topology	Nodes	Logic Utiliz. (LUTs)	Logic Incr. (%)	Logic Ovrhd. (%)	Total Reg.	Reg. Incr. (%)	Reg. Ovrhd. (%)
ring	8	10197	0.0	8.7	2637	0.0	11.2
star	8	10393	1.9	10.8	2642	0.2	11.6
mesh	8	10470	2.7	10.7	2641	0.2	11.4
hypercube	8	10701	4.9	12.6	2645	0.3	11.5
fully con.	8	12376	21.4	22.3	2762	4.7	13.9
ring	16	20448	0.00	8.7	5186	0.0	11.4
star	16	20936	2.4	9.6	5190	0.1	11.8
mesh	16	21360	4.5	12.6	5202	0.3	11.7
hypercube	16	22272	8.9	16.2	5218	0.6	12.0
fully con.	16	30176	47.6	38.1	5490	5.9	16.3
ring	32	40648	0.00	8.7	10209	0.0	11.6
star	32	41880	3.0	9.0	10214	0.1	11.9
mesh	32	42936	5.6	13.6	10250	0.4	11.9
hypercube	32	45104	11.0	17.9	10306	0.9	12.4
fully con.	32	87760	115.9	57.8	11330	11.0	20.3

example, the fully-connected topology requires 21.4% more LUTs than the ring for the 8-node system. In contrast, the *Logic Ovrhd* column shows the number of LUTs used for the network interfaces as a fraction of the total LUTs required for the complete system. It is calculated as $(total\ number\ of\ LUTs\ for\ network\ interfaces)/(Total\ LUTs\ in\ the\ system)$. As expected, the ring topology has the lowest overhead for all node sizes and the fully-connected system overhead increases very quickly as the number of nodes increases.

Table 2 also shows the corresponding results for the register (flip flop) utilization of the various topologies. The trends mimic the logic utilization data, but the variation is smaller because the number of registers in the network interface module is small.

The routing resource usage of each system is presented in Table 3. The *Routing Utiliz.* column is the total number of nets used in the design. In general, the routing resource utilization follows a similar pattern to the logic resource utilization across the systems. The fully-connected system requires the most nets, as expected. It should also be noted that the 32-node, fully-connected topology design could be placed but not completely routed, leaving 56 unrouted nets.

Column *Routing Incr.* presents the difference in routing

resources relative to the ring topology. It can be seen that the greatest increase in routing for the ring, star, mesh and hypercube topologies occurs for the 32-node hypercube system with only a 10.6% increase relative to the 32-node ring system. This reflects the $O(n \log n)$ link complexity of the hypercube as compared to the $O(n)$ link complexity for the ring, star, and mesh topologies.

The *Routing Ovrhd* column is calculated as the total number of nets for all the network interfaces divided by the total number of nets in the entire system. A visual representation of how each network topology contributes to the global number of nets can be seen in Figure 4. From this figure it can be seen that the ring topology overhead is practically independent of the system size at about 6% of the total nets for the 8, 16 and 32-node systems. The star and mesh topologies increase slowly to a maximum of about 11% of the total nets for the 32-node system. The hypercube adds about 15% overhead to the global routing in the 32-node system. The fully-connected topology starts at 20% overhead for an 8-node system, and grows to around 55% of the total routing for the 32-node topology, which actually fails to completely route. The other topologies have much lower routing overhead and will likely be able to expand to 64-node systems, assuming large enough chips exist.

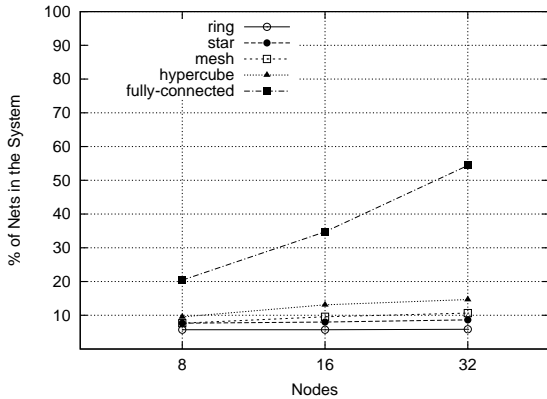


Figure 4: Topology impact on global routing

The place and route time data varies considerably because of how the place and route algorithms work and factors that impact the workstation performance. For the ring, star, mesh and hypercube topologies, the average times to place and route are approximately the same for a fixed number of nodes.

For the 8, 16 and 32-node systems the average times are 12 min., 30 min. and 4 hours 48 min., respectively. The fully-connected topology exhibits an exponentially growing time of 15 min. for the 8-node system, 12 hours for the 16-node system, and remained unroutable after 3 days for the 32-node system.

Table 3 also shows the clock frequency (*freq.*) achieved for each of the systems. Of the 8 and 16-node systems, only the fully-connected, 16-node system is not able to meet the 150 MHz requirement, achieving only 128 MHz. With the 32-node systems, the target is 133 MHz, but this is not achieved by the star and the fully-connected network. The star incurs congestion at the central node, which affects the timing, and the fully-connected system requires too many wires. Despite the fact that the placement and routing ef-

Table 3: Routing resources used by each system

Topology	Nodes	Routing Util. (nets)	Routing Increase (%)	Routing Ovrhd (%)	freq. (MHz)	Target Clock (MHz)
ring	8	10744	0.0	5.7	150	150
star	8	10956	2.0	7.6	151	150
mesh	8	11021	2.6	7.7	151	150
hyp.cube	8	11256	4.8	9.5	152	150
fully con.	8	13045	21.4	20.4	150	150
ring	16	21501	0.0	5.7	152	150
star	16	22013	2.4	8.0	150	150
mesh	16	22429	4.3	9.6	151	150
hyp.cube	16	23357	8.6	13.1	151	150
fully con.	16	31373	45.9	34.7	128	150
ring	32	42618	0.0	5.8	133	133
star	32	43888	3.0	8.6	100	133
mesh	32	44945	5.5	10.6	132	133
hyp.cube	32	47136	10.6	14.7	133	133
fully con.	32	90016	111.2	54.4	Fail	133

orts were set to *high*, no time was spent to try and push the tools to improve the results that did not meet the targets.

5. AREA REQUIREMENTS

For the previous experiments, LUT and flip flop counts are used as the reference metrics for logic resource utilization. However, for this experiment the number of slices is used to measure area usage. In the Xilinx architecture, each slice contains two LUTs. A design requires a certain number of LUTs and flip flops, and depending on how well the packing algorithm performs, the design will require more or less slices. Moreover, the place and route tools may not be able to utilize the two LUTs in every slice because of routing constraints and timing requirements. The number of slices better reflects the actual chip area required to implement the design. Also, the area constraints used by the Xilinx tools are specified in terms of slices.

For this experiment, the *Minimum Area Required* is defined as the smallest number of slices needed for the design to place and route successfully. It is determined by reducing, or compressing, the area used by the design until just before it fails to place and route and counting the number of slices in the compressed region at that point. This gives a measure of how efficiently the design can use the resources when the resources are close to being fully utilized, which models the effect of trying to implement a design on a chip that is close to full capacity.

The area compression is done using area constraints in the *User Constraints File*, i.e., the *.ucf* file. The constrained area is described by giving the coordinates of the bottom-left and top-right slice positions that define a rectangular area in the FPGA. The origin is fixed to $X0Y0$ and the second X coordinate is fixed at the maximum value allowed by the specific chip, i.e., the width of the FPGA fabric. The variable is the second Y coordinate. Decreasing the Y coordinate compresses the area available for the design and an iterative process of changing this coordinate is used to find the *Minimum Area Required*.

We define *Area Utilization* as the ratio of the number of slices actually used by the design to the total number of

Table 4: Area utilization for the 8-node systems using different chips

Topology	XC2V2000		XC4VLX25		XC4VLX40	
	Min. Area Req'd (slices)	Area Utiliz. (%)	Min. Area Req'd (slices)	Area Utiliz. (%)	Min. Area Req'd (slices)	Area Utiliz. (%)
ring	5376	92.5	9352	62.1	8208	70.7
star	5568	90.8	8736	67.5	8640	68.2
mesh	5376	94.4	9240	64.0	8280	73.5
hypercube	6336	81.9	8568	70.4	8280	72.9
fully con.	6528	92.5	9688	71.2	8208	84.1
Average	5836	90.4	9116	67.0	8323	73.9

slices in the area available to the design. The ideal area utilization is to use 100% of the slices available to the design.

In this experiment, the 8-node systems using the five topologies are compared using three different chips. This experiment also requires a time-intensive, iterative search to find the *Minimum Area Required* so the 16 and 32-node systems are not studied.

The fact that the MicroBlaze soft processor needs internal memory (BRAM) to store the code and data required a slight modification to our methodology. BRAMs are hard core macros that have fixed positions within the FPGA fabric, and they are organized as blocks of 18 Kbits each, distributed along different columns in the array. A problem may arise if the restricted area does not have enough BRAMs for the design. In that case, the map program will issue an error about the lack of resources or resources being over mapped. For that reason, the BRAM blocks are not constrained in our methodology. This implies that nets connected to the BRAMs may be outside of the restricted area. However, each MicroBlaze uses only 8 KB of BRAM, which is only four BRAM blocks. In all of the designs, the number of BRAMs outside the restricted area is small and the number of nets involved are not significant compared to the total number of nets in the design.

To ensure that the resulting maximum clock frequency is realistic, the timing requirement is set to 150 MHz, but a successful place and route in this experiment does not require that timing be met, only that it comes within about 20 MHz to ensure a respectable clock speed. The important requirement is that there are sufficient resources to complete a place and route. All of the Virtex 4 designs are able to achieve 150 MHz using *high effort*. The Virtex 2 device, being in a slower technology only has to meet 133 MHz as the timing constraint. The speeds achieved are 133 MHz for the ring, star and hypercube, 125 MHz for the mesh and 117 MHz for the fully interconnected system. These results are also achieved using *high effort*.

5.1 Results

Table 4 shows the minimum number of slices required *Min. Area Req'd* and the area utilization *Area Utiliz.* for the 8-node systems. Observe that a higher area utilization can be achieved in the Virtex 2 chip compared to the Virtex 4 chips and the XC4VLX40 achieves slightly higher utilization than the XC4VLX25.

Within the same chip, the fully-connected topology achieves equal or higher area utilization ratios than the rest of the topologies, even if the other topologies require fewer re-

sources than the fully-connected one. This occurs because the fully-connected has more logic to place in the empty spaces caused by MicroBlaze macros that have holes of unused slices in them due to the use of relational placement directives. These directives force certain parts of the logic into fixed relative positions, which results in unused slices within the bounds of the macro. In this case, the fully-connected topology can make more effective use of the empty spaces because it has more logic to place.

Table 4 also shows that the average *Min. Area Req'd* slices varies quite significantly across the different chips. A part of the explanation is due to an architectural change to the Configurable Logic Block (CLB) between the Virtex 2 and the Virtex 4 families. The effect of the change is that the implementation of the MicroBlaze in Virtex 4 uses 160 more LUTs than in a Virtex 2. This difference explains almost all of the differences in LUT requirements between the two FPGA families, which accounts for part of the difference in slice counts. The other important reason for the differences in area required arises because of the relative width of the MicroBlaze macro to the width of the FPGA fabric. For the same amount of logic, Virtex 2's fabric is wider than Virtex 4's fabric.

6. MAXIMUM FREQUENCY

The goal of this experiment is to determine the maximum frequency achievable by the different topologies. This is done by using the *Xilinx Explorer* utility, which is a Perl script that runs the map, and place and route tools up to six times. Each run uses a different combination of command line options that enable the tools to try various combinations of optimizations to meet the timing constraints. These options set the mapping effort level to high, place and route effort levels to high, try different seeds, enable timing-driven packing, global optimizations, register duplication and alternative algorithms for the mapping tool. The specified frequency is 180 MHz because it is the frequency at which the MicroBlaze synthesizes, being the slowest IP block in the system.

The fastest *Speed Grade* available for each chip is used. The XC4VLX200 is the largest part and it has a maximum speed grade of 11, which is not as fast as what is available for the other chips we use. The method for finding the maximum frequency is an iterative process and it takes a great amount of time to map, place and route all of the systems. Instead, the full set of 16-node systems in combination with one example from each of the 8 and 32-node systems are selected for this study. The fully-connected, 8-node network is chosen because it is the most challenging of the 8-node systems. The remaining 8-node networks should be easier than the corresponding 16-node networks. The 32-node ring system is selected because it is the easiest topology of the set of 32-node networks.

6.1 Results

Table 5 shows the results of executing the *Xilinx Explorer* utility. The maximum frequency achieved is shown in Column *Max Freq.* The 8-node, fully-connected system fails to meet the 180 MHz constraint. An examination of the longest critical paths shows that 88% of the delay is attributed to routing delay suggesting that routing is the main factor for not meeting timing.

Most of the 16-node systems meet the target frequency

Table 5: Results from running Xilinx *Xplorer*

Topology	Nodes	Max Freq. (MHz)	Speed Grade	Best Run	Total Runs
fully con.	8	170	12	2	6
ring	16	180	12	1	1
star	16	180	12	4	4
mesh	16	180	12	2	2
hypercube	16	180	12	2	2
fully con.	16	126	12	5	6
ring	32	123	11	5	6

of 180 MHz. Only the 16-node, fully-connected system is unable to meet the timing constraints. Examination of the timing report shows that for the longest critical paths, 72% of the delay is due to routing.

Interestingly, the 32-node ring is able to achieve 133 MHz in the previous experiments with about 50% of the critical path delay in the routing, but when the constraint is changed to 180 MHz only 123 MHz is achieved, with 72% of the critical path delay being in routing.

The *Best Run* is the iteration number in which the maximum frequency is obtained. The column *Total Runs* shows the total number of iterations for which the script invoked the map, place and route tools. The maximum value is 6, but it is less if the timing constraint is met in fewer iterations. In the case of the ring with 16 nodes, the topology is so simple that the maximum frequency is achieved by the first run. The star needed more runs indicating the tools have to work harder due to the congestion around the central node. Adding a few more nodes to the star will likely cause it to fail to meet the 180 MHz target. The fully-connected system requires six runs and still cannot meet the timing constraint, getting the best result in iteration number 5. A similar situation happens with the 32-node ring system.

The size of the MicroBlaze macros dictates the overall placement and will not change significantly for the different topologies. The fact that some topologies can meet timing and others fail means that placement of the nodes of the network is not the important factor for failing to meet timing. The only other factor would be the routing. In this experiment, it is observed that routing delay is the major component of the critical paths of the star and the fully-connected topologies, which is indicative of routing being the problem. From what we know about the star and fully-connected topologies, routing congestion becomes the problem for large node sizes, which supports the observation of routing delay being the major component of the critical paths.

7. A COST METRIC

We identify two types of factors that influence a topology selection: performance factors and implementation factors. The performance factors are: network latency and bandwidth. In an FPGA, the implementation factors are: area, logic and routing resources. In order to analyze possible interesting trends, we used the following cost metric (*CM*) to minimize:

$$CM = k_1 A_N + k_2 L_N + k_3 R_N + k_4 T_N + k_5 D_N - k_6 (Bw \times W \times freq)_N \quad (1)$$

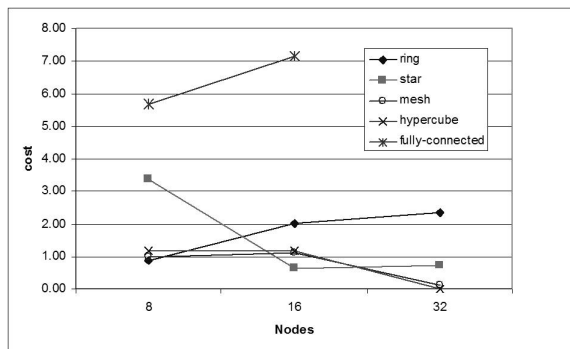


Figure 5: Cost metric trends

where A_N is the area in terms of number of slices used only by the network interfaces. L_N is the number of LUTs occupied by network devices. It is computed as $Logic Utiliz. \times Logic Ovrhd$ from table 2. Similarly, R_N is the number of nets required by each topology is computed as $Routing Util. \times Routing Ovrhd$ from table 3. The T_N is the place and route time for each system. $Diameter$ is the network diameter computed using the appropriate expression in Table 1. Recall that a small diameter indicates fewer links need to be traversed in the worst case, which means lower network latency.

In this paper we are not dealing with a particular protocol or particular network interface architecture; therefore, we define latency in terms of the diameter and we are not considering switch delay or initialization time delay to account for the total latency. Bw is the bisection width from table 1. W is the channel width (64 bits) and $freq$ is the system frequency as reported in table 3. Assuming a single clock cycle per data word transmission, then the product $BwWfreq$ represents bisection bandwidth which is defined as the bisection width times the bit rate. The subscript N in each of the attributes in equation 1 means that the data was normalized to prevent conflict with the different scale units. The constants $k_{1,2,..,6}$ are weights assigned according to the importance of each factor. The actual values of the constants depend on the application or restrictions in the design.

Figure 5 shows a plot of the cost metric for the studied topologies assuming that all the factors are equally important, i.e., $k_{1,2,..,6} = 1$.

We can see that the ring topology is good for 8 nodes because of its relatively low diameter and its easy implementation; however, the cost will increase consistently as the number of nodes grows because of a longer network latency and the bisection bandwidth does not scale well. The star network shows the opposite behavior. At 8 nodes the congestion surpasses the performance and its cost is high, but as we increase the number of nodes to 16, the diameter=1 keeps the latency constant and consequently the cost is low. Nevertheless, for 32 nodes the star will not improve because the congestion is such that higher frequencies can not be achieved reducing the bisection bandwidth and therefore the performance decreases. On the other hand, the Mesh and Hypercube have both similar costs and they are constantly reduced as the number of nodes increases. This trend shows that these topologies scale well in routing and logic while keeping a good performance. The fully-connected topol-

ogy is the most expensive topology under the assumption of equally important performance and implementation factors, because the routing, logic and area required undermine its performance. However, if the weights were set such that privilege the performance, then it would have a lower cost than some of the topologies despite its complexity (this case is not shown in figure 5). Since the fully-connected could not route, we do not have a metric for 32 nodes.

8. CONCLUSIONS

In ASICs, only the required routing is provided so it is important to make the correct tradeoffs between communication requirements for the application and the routing requirements of the interconnect topology. FPGAs have a rich routing fabric and the wires are there to use or not. This work has shown that the ring, star, mesh and hypercube topologies can all be routed on a modern FPGA. Even a fully-connected topology can work for small numbers of nodes, which suggests that for small networks, it is not as important to worry about routing considerations when picking the network topology for an NoC on an FPGA. As the number of nodes grows larger, the star and fully-connected topologies are the first to have difficulties because of routing congestion. Even with 16 nodes, it is still feasible to implement these topologies but the routing congestion makes it more difficult to achieve higher clock frequencies. At 32 nodes, the fully-connected topology fails to route, with 56 nets unrouted. This small number of unrouted nets suggests that the system routing requirements are at approximately the limits of the device capacity. To do better, the routing capacity of the device would have to be increased. A cost metric has been developed that estimates the impact of these topologies for even larger system sizes. It shows that the hypercube topology can be used to build even larger systems if the current FPGA array is simply expanded.

This work is focused on how well a modern FPGA routing fabric can support several selected network topologies. It would be useful to try other topologies. Also, another important concern for current designs is power, which is not considered in this work.

9. ACKNOWLEDGMENTS

Thanks to Jorge Carrillo and Trevor Brauer at Xilinx for their feedback and helpful comments. We acknowledge Xilinx and CMC Microsystems/SOCRN for providing the tools, development hardware and computing equipment. Thanks to CONACYT for the funding provided to Manuel Saldaña.

10. REFERENCES

- [1] ARM Corporation. "AMBA specification". [online] 1999. www.arm.com (Accessed: 2005).
- [2] IBM Corporation. "The Coreconnect Bus Architecture", [online] 1999. www.chips.ibm.com (Accessed: 2005).
- [3] OpenCores.org. "The WISHBONE System Architecture". [online] 2002. opencores.org/projects.cgi/web/wishbone (Accessed: 2005).
- [4] Sonics Inc. (online). www.sonicsinc.com/sonics/products/siliconbackplaneIII/ (Accessed: 2005).
- [5] G. de Micheli and L. Benini. Networks on chip: A new paradigm for systems on chip design. In *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, page 418, Washington, DC, USA, 2002. IEEE Computer Society.
- [6] S Kumar, A Jantsch, J Soinen, M Forsell, M Millberg, J Oberg, K Tiensyrja, and A Hemani. A network on chip architecture and design methodology. In *VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on*, pages 105–112, Pittsburgh, USA, 2002. IEEE Computer Society.
- [7] Adrijean Adriahtenainina, Herve Charlery, Alain Greiner, Laurent Mortiez, and Cesar Albenes Zeferino. Spin: A scalable, packet switched, on-chip micro-network. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, page 20070, Washington, DC, USA, 2003. IEEE Computer Society.
- [8] Partha Pratim Pande, Cristian Grecu, Michael Jones, André Ivanov, and Resve Saleh. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Trans. Comput.*, 54(8):1025–1040, 2005.
- [9] William J. Dally and Brian Towles. Route packets, not wires: on-chip interconnection networks. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 684–689, New York, NY, USA, 2001. ACM Press.
- [10] G. Brebner and D. Levi. Networking on chip with platform fpgas. In *Field-Programmable Technology (FPT), Proceedings. 2003 IEEE International Conference on*, pages 13–20, July 2003.
- [11] Tim Kogel, Malte Doerper, Andreas Wiefierink, Rainer Leupers, Gerd Ascheid, Heinrich Meyr, and Serge Goossens. A modular simulation framework for architectural exploration of on-chip interconnection networks. In *CODES+ISSS '03: Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 7–12, New York, NY, USA, 2003. ACM Press.
- [12] Davide Bertozzi and Antoine Jalabert. Noc synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Trans. Parallel Distrib. Syst.*, 16(2):113–129, 2005.
- [13] Lesley Shannon and Paul Chow. Simplifying the Integration of Processing Elements in Computing Systems using a Programmable Controller. In *IEEE Symposium on Field-Programmable Custom Computer Machines (FCCM'05)*, April 2005.
- [14] Charles L. Seitz. Let's route packets instead of wires. In *AUSCRYPT '90: Proceedings of the sixth MIT conference on Advanced research in VLSI*, pages 133–138, Cambridge, MA, USA, 1990. MIT Press.
- [15] T.A. Bartic, J.Y. Mignolet, T. Marescaux, D. Verkest, S Vernalde, and R. Lauwereins. Topology adaptive network-on-chip design and implementation. In *Computer and Digital Techniques, IEEE Proceedings*, pages 467–472. IEE Proceedings, July 2005.
- [16] J Duato and L Yalamanchili Ni. *Interconnection Networks, an Engineering Approach*. Computer Society Press, Los Alamitos, California, 1998.
- [17] Xilinx, Inc. <http://www.xilinx.com>.
- [18] ModelSim Home Page, [online] September 2005. <http://www.model.com/>.