A Parameterizable NoC Router for FPGAs

Mike Brugge
University of Windsor/Department of Electrical and Computer Engineering, Windsor, Ontario, Canada
Email: brugge@uwindsor.ca

Mohammed A. S. Khalid
University of Windsor/Department of Electrical and Computer Engineering, Windsor, Ontario, Canada
Email: mkhalid@uwindsor.ca

***Abstract*—The Network-on-Chip (NoC) approach for designing (System-on-Chip) SoCs is currently used for overcoming the scalability and efficiency problems of traditional on-chip interconnection schemes, such as shared buses and point-to-point links. NoC design draws on concepts from computer networks to interconnect Intellectual Property (IP) cores in a structured and scalable way, promoting design re-use. This paper presents the design and evaluation of a parameterizable NoC router for FPGAs. The importance of low area overhead for NoC components is pivotal in FPGAs, which have limited logic and routing resources. We obtain a low area router design by applying optimizations in switching fabric and dual purpose buffer/connection signals. We utilize a store and forward flow control with input and output buffering. We proffer a component library to increase re-use and allow tailoring of parameters for application specific NoCs of various sizes. The proposed router supports the mesh architecture which is well known for its scalability and simple XY routing algorithm. We present IP-core-to-router mapping strategies for multi-local port routers that enable ample opportunity to optimize the NoC for application specific data traffic. A set of experiments were conducted to explore the design space of the proposed NoC router using different values of key router parameters: channel width (flit size), arbitration scheme and IP-core-to-router mapping strategy. Area and latency results from the experiments are presented and analyzed. These results will be useful to designers who want to implement NoC on FPGAs.**

***Index Terms*—Network-on-Chip, NoC, System-on-chip, SoC, FPGA, VHDL, evaluation, router, arbiter, flit size, multi-local, mesh**

I. Introduction

The complexity of a system on silicon is comparable to other macro systems such as space shuttle or skyscrapers, when measured in terms of the number of basic elements intricately connected together, but at a micro level. Moore's law describes an important trend in the history of the integrated circuit (IC): the number of transistors that can be placed on an IC is increasing exponentially, doubling approximately every two years. This trend has continued for more than half a century. Increasing transistor density, higher operating frequencies, shorter time-to-market and reduced product life cycle, characterize today's semiconductor industry [8]. As semiconductor technology evolves, electronic industries continually push the envelope for greater functional and performance capabilities in new electronic systems. This is creating a continuing need for new design methodologies and design space exploration.

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing constraints. Embedded systems range from portable devices such as digital watches, cameras and MP3 players, to large stationary units like traffic lights and factory controllers. Complexity varies from low, with a single micro-controller chip, to very high with multiple intellectual property (IP) cores and peripherals. The exponential growth in chip density is opening the door for the implementation of even larger and more complex systems, where complete embedded systems can be built onto a single chip. This paradigm shift is known as System-on-Chip (SoC) and is becoming increasingly common and complex. SoCs may contain many hardware and/or software blocks, such as processors, DSPs, memories, peripheral controllers, gateways, and other custom logic blocks.

The on-chip interconnect architecture used in SoCs is a key factor that impacts the overall performance. Since the introduction of SoC concept, designers relied on a custom-designed ad-hoc mixture of buses and dedicated wires for on-chip interconnections. Dedicated wires are effective for systems with a small number of cores, but available routing resources are quickly used up as system complexity grows. They also provide poor reusability and flexibility. A shared bus is a set of wires common to multiple cores, which increases both reusability and scalability. This scheme works well for Master-Slave communication patterns, where peripherals (slaves) wait for data to be received or requested from a more complex IP core (master). However, when there are several masters in the system, contention creates a bottleneck which gets worse as complexity grows. Although using hierarchical bus models connected by bridges may reduce some of these constraints, it also complicates protocols while failing to fully eliminate the scalability problem. Design and verification times also grow with SoC complexity [13].

doi:10.4304/jcp.9.3.519-528

With the current trend in integration of more complex SoCs, there is a need for better communication infrastructure on chip that will solve the scalability problem by supporting multiple concurrent connections between IP cores, allow reuse of pre-tested IP cores to minimize design and verification times, all the while maintaining a low area-overhead. Many architectural templates have been proposed for hardware platforms for future SoCs to provide standardized communication. NoC has been introduced as a new interconnection paradigm able to integrate IP cores in a structured and scalable way [2][22]. This idea aims to allow system modules to communicate with each other over an on-chip network and has been gaining support world-wide. NoCs are based on the concepts adopted on the building of interconnection networks for parallel computers. Each router has a set of ports which are used to connect routers with its neighboring routers and with the IP cores of the system. This solution also promotes independent design of IP cores. There is a great need for research in hardware implementation of NoC-based systems to determine the feasibility of implementing various components, and also to accurately determine the design tradeoffs involved in NoC implementation.

ASICs are increasingly being replaced by FPGA for applications with low to medium volume, due to longer design cycles and high cost [14]. FPGA's have also continued to grow with the increase in chip density. Modern FPGA's have various hardware and/or software blocks embedded within them, such as DSP blocks, memory, and even processors. These blocks, along with customizable logic blocks, makes them the perfect candidate for NoC designs. A fundamental difference between ASICs and FPGAs is that wires in ASICs are designed such that they match the requirements of a particular design. Wire parameters such as length, width, layout and the number of wires can be varied to implement a desired circuit. Conversely, in an FPGA, area is fixed and routing resources exist whether or not they are used. The electrical characteristics of the FPGA are fixed by the chip vendor, not by the user [3]. Exploiting the advantages of NoC in FPGAs for implementing SoC designs is an active area of research where the goal becomes implementing a circuit within the limits of available resources. Hence, it is important to design a generic light-weight router, whose area can be traded off for performance in many different ways, to meet application requirements.

This paper presents the design and evaluation of a parameterizable NoC router for FPGAs. The importance of low area overhead for NoC components is crucial in FPGAs, which have fixed logic and routing resources. We achieve a low area router design through optimizations in switching fabric and dual purpose buffer/connection signals. We use a store and forward flow control with input and output buffering. We propose a component library to increase re-use and allow tailoring of parameters for application specific NoCs of various sizes. Our router supports the mesh architecture which is well known for its scalability and simple XY routing algorithm. We introduce IP-core-to-router mapping strategies for multi-local port routers that provide ample opportunity to optimize the NoC for application specific data traffic. A set of experiments were conducted to explore the design space of the proposed NoC router using different values of key router parameters: channel width (flit size), arbitration scheme and IP-core-to-router mapping strategy. Area and latency results from the experiments are presented and analyzed. These results will be useful to designers who want to implement NoC on FPGAs. This router was presented as a poster in a conference and an abstract only of this paper was previously published [23].

The rest of the paper is organized as follows: Section 2 presents an overview NoC and NoC router design. Section 3 provides a description of related work. An overview of the proposed router architecture is given in Section 4. In Section 5, we present experimental evaluation results for a stand-alone router used in a variety of mesh configurations, with different combinations of router parameter values. Section 6 concludes the paper with a summary and discussion of future work.

II. NOC BACKGROUND

A sample NoC-based system is shown in Figure 1. It shows a mesh topology used to interconnect sixteen IP cores. Each core is connected to the routing node (router) using a network adapter. Each router is connected to its nearest neighbor using a link.

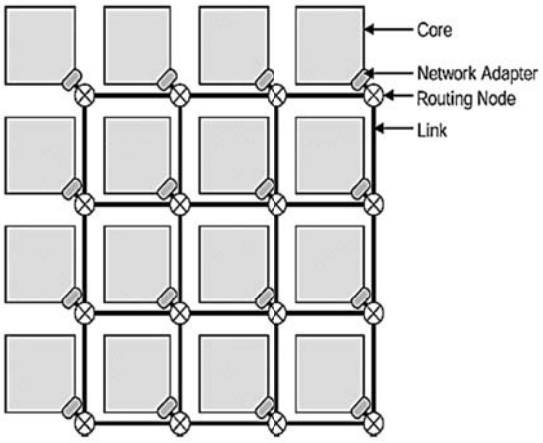

Figure 1. NoC Mesh Architecture

Network parameters are an important research topic among NoC designers. To further enhance performance, the parameters of the NoC should be chosen based on the specific application. Therefore, the goal in a general network design is to leave as much designer flexibility as possible. Not every network parameter can be created flexible and many of the parameters are dependent on each other. NoC implementation and evaluation can provide insight into how to select these parameters, although a better solution may be a flexible library of interchangeable components. We have chosen to create such a library using VHDL, and use an FPGA to provide

fast prototyping results. Due to timing, limitations had to be set on the design space. Network parameters can be broken into three groups [22]: Infrastructure, Communication Mechanism, and Mapping.

Infrastructure aims to determine the network architecture and includes topology, channel width, buffering and floor planning. These parameters are all application specific and should be left to the designer's discretion. Chanel width describes the size of the data passed between routers. It is important since it directly affects bandwidth but can lead to the side effects of increased area and power consumption. Our library allows for a parameterizable channel width. Topology refers to the way routers are connected in the network. It should be chosen to minimize area, while maximizing utilization without causing bottlenecks. Saldana et al. evaluate different topologies in terms of area and routing resources [3]. Ring and star achieve slightly better results, although both fail to provide solutions to the scalability problem. As the number of nodes increases, ring suffers large end to end delay and star suffers from a central bottleneck. Narasimhan et al. compare the performance of 2D torus to mesh, showing a slight edge for 2D torus [4]. They however, do not compare the extra routing resources needed or the increase area of each router due to a more complex routing algorithm. We restrict the topology to mesh, which is most common among FPGA networks, but allow for various implementation sizes up to an 8 x 8 network. With available FPGAs, it would be impractical to build anything larger due to area and routing resource constraints. Buffering defines the approach used to store messages while they cannot be scheduled. This has a serious impact on the area overhead of the network, however, it can also have a serious impact in reducing network latency. We use input and output buffering to prevent head-of-line blocking. This occurs when a packet or packets, experience blocking and cause the blocking of later packets which could otherwise be processed. The inclusion of an output buffer allows the blocked packet to move out of the input buffer, to unblock the later packets for processing. Buffer allocation should be based on traffic patterns. The authors of Hermes [8], design a generic router which has a parameterizable buffer depth. They also include insight through testing various buffer sizes for area and performance values. Floor planning involves the placement of network components on the chip, it is more important for ASIC when compared to FPGA implementation.

Communication mechanism deals with flow control, switching mode, switching mechanism, and routing algorithm. These parameters are usually set when designing the NoC platform. Flow Control deals with the allocation of channels and buffers to data as it travels from source to destination. The two extremes are packet switching and circuit switching. In circuit switching, there is a dedicated connection between the two modules in which raw data can be transmitted freely. This technique requires a setup time to build and tear down connections, and its channel reservation nature often leads to idle times and causes unreliable blocking. The only upside to this method is its ability to provide guaranteed bandwidth during connection times. This method does not scale as well. In packet switching, data is broken into packets which carry routing information. Packets can further be broken down into flow control units (flits). Modules can send packets at any time and there are often many different packets in flight at a given time. The routers must process and redirect each packet accordingly. The Switching mode defines how packets move through the network. The most important are store-and-forward (SAF), virtual cut-through (VCT), and wormhole (WH). In SAF, a switch cannot forward a packet until all its flits have been received. Therefore, latency is proportional to packet size. In WH, the first flit (header) determines the next hop and all remaining flits follow. Therefore, latency is proportional to flit size. This method combines packet switched and circuit switched ideas but also leads to channel reservation. It also requires a complex routing algorithm. VCT uses a combination of both ideas to provide latency based on flit size without idle times by guaranteeing buffering before setting up the connection. However, this method uses large number of buffers and very complex routing algorithms making it unsuitable for light-weight networks. We have chosen SAF for its light-weight algorithm and to prevent channel reservation. Future testing may extend flexibility to include WH as well. Switching refers to how connections are made inside a router. We use a partial crossbar scheme to save area. The routing algorithm determines the path the packet will take. We use XY routing for its simplicity and low area overhead. This scheme also prevents livelock. Routing schemes can also require congestion control and recovery mechanisms, which can lead to added area overhead. We allow this to be handled by the application layer.

Mapping determines how to integrate a given application to the NoC platform and includes scheduling and module mapping. Scheduling is a traditional computer science topic but most work neglects inter-processor communication. Arbitration schemes consider priority of packets in routers among the network and include static and dynamic. Dynamic arbitration makes a decision at run-time and is more flexible, however also requires a larger area. Our library provides a few different component to allow for area and performance trade-offs. Arbitration can also deal with preventing deadlock. Module mapping aims at selecting IP modules to different locations to minimize traffic. There parameters are application specific and are both tested later.

III. Related Work

Our Router has been designed and synthesized on an Altera Stratix II FPGA. Therefore, although there are a number of ASIC and custom IC implementations, we restrict our discussion of related work to FPGA implementations. This section is intended to provide a comprehensive review of the state of the art for NoC

implementation on FPGAs, although the authors do not make claims about its completeness.

The first working implementation of FPGAs was presented by Marescaux et al. [6]. It has many faults mainly large size, and a one dimensional architecture which fails to provide a high degree of scalability. They extend their work in [7], allowing a more flexible architecture, but still suffering large area. They use VCT flow control which is now [14] considered too area intensive for FPGA platforms because of complex routing logic without eliminating any buffer constraints.

Moraes et al, present Hermes [8], a router with parameterizable data width and buffer depth. They perform simulations on a 5 x 5 mesh to explore the parameter buffer depth. They conclude with the notion that increased buffer size reduced latency, but only to a saturation point. Their design uses centralized arbitration and routing units, which decreases area but stalls performance as routing requests are queued to be handled one at a time. Their design also suffers from a very low clock speed. They later extend their work to provide an automatic router generation and traffic analyzer [9].

A comparable router, RASoC [10], was presented by Zeferino et al. The main difference being they use a WH flow control. Performance differences are yet to be compared and may be considered for future work as a WH downfall is that it reserves channels which can cause blocking. However, WH also requires complex routing logic as well as extra bits in the datapath for framing. They also used Altera FPGA to synthesize their 5-port, 8-bit router which occupies 486 LE's and has a clock frequency of approximately 57MHz. This area is quite large for a router whose buffers are limited to 4 per port.

PNoc, proposed by Hilton et al in [13], gives us a router with circuit switched flow control. They test their router against bus based approaches to show improvements. However, routing complexity grows as the number of ports, or number of routers increase and therefore reduces scalability. It also suffers typical CS setup and teardown latencies and possible idle time which could block other needed communication.

Sethuraman et al. propose LiPaR in [14], which was a starting point of our design, but significant improvements were added by us. They use SAF, input and output buffering, and decentralized components. Optimizations are made in the crossbar matrix to reduce area through careful analysis of the XY routing algorithm. However, we extend these optimizations to the arbitration unit. They use a single 5x5 crossbar matrix for switching rather then 5 5x1 partial crossbars leading to a larger area. Their complex crossbar design results in a slower clock speed and increased area.

They later propose multi-local port routers (MLPR) in [15], which have the potential of improving area and performance metrics. However, the authors fail to provide any synthesis results to support their proposal. Another extension the authors propose is Optimap [16], an exhaustive CAD tool for mapping IP's and choosing network size.

Vestias et al. propose GNoC in [17], a generic router which supports a range of routing, switching and arbitration protocols. They create a tool for exploring the sharing of some decentralized components to reduce area that is based on the injection rate of ports. Unfortunately, they lock all protocols to certain values and do not explore them further. Their tool shows how they can save area when injection rates are low but does not test to see if performance is degraded.

MoCres, designed by Janarthanan et al. in [18], uses complex VCT flow control and attempts to reduce area by sacrificing area through centralizing components. They create multi-clock domain to enable high clock frequencies during transfers. Optimizations from XY routing in the crossbar matrix have been extended to the routing algorithm, and gave us the idea for a further arbitration unit extension. We have also used their idea of creating VHDL wrappers to simulate the stand-alone router or routing configurations to compare parameters.

Our paper attempts to zero in on all the best router characteristics from the above to make as many optimizations in area as possible while concentrating on system performance. We notice a lack of evaluation and comparison of network parameters on FPGAs and try to test accordingly. Most work has focused on dynamic arbitration schemes, mainly round robin (RRA), which may be too area consuming when implementing decentralized components. We see that the data width size is often set to 8-bit flits as many papers assume a size without analysis. Most importantly, we agree with the opportunity to optimize data traffic through use of MLPR. Our plan is to present area utilization and performance values for the above network parameters to help future designers make accurate decisions for their computing needs.

IV. Router Architecture

In this section we describe the architecture of the proposed parameterizable router for NoC implementation on FPGAs. The router has 4 ports, North, East, South, and West for communication with neighboring routers. There can also be anywhere from 1 to 4 Local ports for connecting to IP cores. FIFO buffers were created at the input and output ports for temporary storage. Communication between them is established by use of a two-way handshake of request/grant signals controlled by logic controllers. These signals are also used for configuration of the partial crossbar switch to reduce area overhead. The partial crossbar switch serves as the data connection for input to output port. We also use the empty/full status signals from the FIFO buffers as request/grant signals for inter/intra-router communication (Figure 2). Further details are provided in the following paragraphs.

Buffer size has been set to a depth of 8 bytes. This parameter has been previously explored in related work [8] and it was shown that a buffer size of 8 bytes gives reasonably good area and latency results. Flit size is parameterizable, with 8 bits being the smallest possible size. The first flit, known as the header, contains routing

coordinates used to identify which router and which local port the packet is destined for. With each port capable of having up to 4 local ports, the least significant 2 bits are used to identify between them. That leaves 6 bits for router identification, which allows for an 8 x 8 mesh. Our implementation does not include High Level Protocols (HLP), but could easily be implemented on an application level if necessary.

The block diagram of the proposed router is shown in Figure 3. Each port module is designed the same, and therefore, includes the requests for all output channels even though an input channel will never request its own output channel. Figure 4 shows the interaction among input and output channels within the router. Each input and output channel runs its own FSM logic and hence can request and set up concurrent connections. Below we will include details on the I/O channels and the crossbar switch designs.

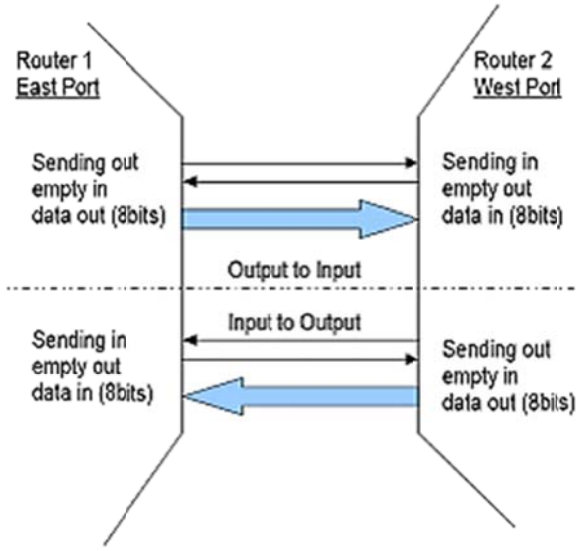

Figure 2. Intra-router Connections

A. Input Controller

All input channel modules are generic and include a buffer unit of depth 8 and a logic controller. The empty signal from the buffer enables outside data transfers. Once the whole packet has been transferred, the full signal from the buffer enables the input controller to decide its next hop from information in the header. It then sends a request signal to the appropriate port's output logic controller. The packet must wait for a grant from the output port before allowing the whole packet to be transferred, one flit at a time. Once the input buffer empties, the whole process can start over again.

B. Switching Mechanism

The crossbar switch is a set of demultiplexers having an interconnection allowing all possible connections between input and output channels. Three optimizations have been made in the crossbar switch. First, it uses a partial scheme, which includes one 5 by 1 unit for each output rather than one 5 by 5 unit for all outputs, for a 5 port router. Each output is connected to a different port. Next, there are no multiplexers in the design. The input data is connected to all partial crossbar units which will choose the appropriate data for the output. The fact that at a time, the output channel can only serve one input request is exploited here. The final optimizations are made in the partial units of the north and south. Though analysis of the XY routing algorithm, we can conclude that these units will never receive data from the east or west. This reduces the inputs of all of these units by two. All optimizations reduce the area without effecting performance of the router.

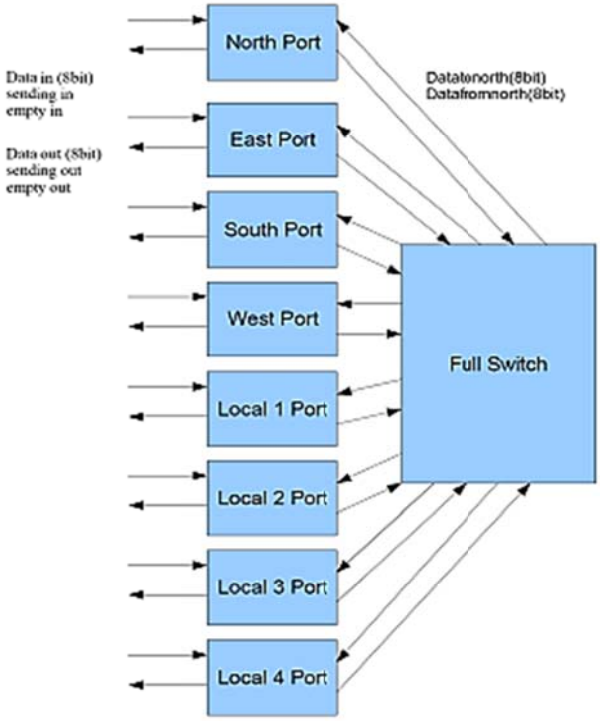

Figure 3. Block diagram of a 4-local port router

C. Output Controller

All output channel modules are generic and include a buffer unit of depth 8 and a logic controller. The empty signal from the buffer allows a grant to be made assuming there is a request from an input channel, while the full signal allows the output controller to send if output is ready. The output logic controller includes an arbitration unit which makes decisions on which port to grant access in the case when more than one port sends a request. Our router employs a variety of schemes including both static and dynamic. With an arbitration unit in each port, it is important to reduce logic area, while maintaining high performance. We will discuss these schemes in greater detail later. Once a grant has been made, it is also used to configure the partial crossbar matrix. The buffer must then empty before allowing arbitration to make another grant. During this time, other requests can be made which may affect which port receives the next grant. Arbitration may play an important role in preventing deadlock, a quality of service (QoS) that XY routing does not provide.

D. XY Routing

Each router contains a routing coordinate. This coordinate is relative to its position in the mesh. Once an

input buffer is full, the header can be compared to the coordinate. It first compares the 3 bits in the header to 3 bits in the coordinate to find its vertical displacement. If the header is greater, it is forwarded to the north, less, it is forwarded south, and equal, its vertical displacement is already correct. Next, the 3 header bits declaring is horizontal displacement are compared to the coordinate. The same process is used, until the packet arrives at the correct router. Once there, we use the last two header bits to determine if it should be forwarded to local port 1, 2, 3, or 4. Since the vertical displacement is taken care of first, we can conclude that if a packet arrives from the east or west, it can never be delivered to the north or south. Its vertical displacement is correct and it must either be forwarded horizontally, or to a local port. This leads to optimization and saves significant area as the crossbar matrix can often be the largest component of the router. XY routing is free of livelock and starvation.

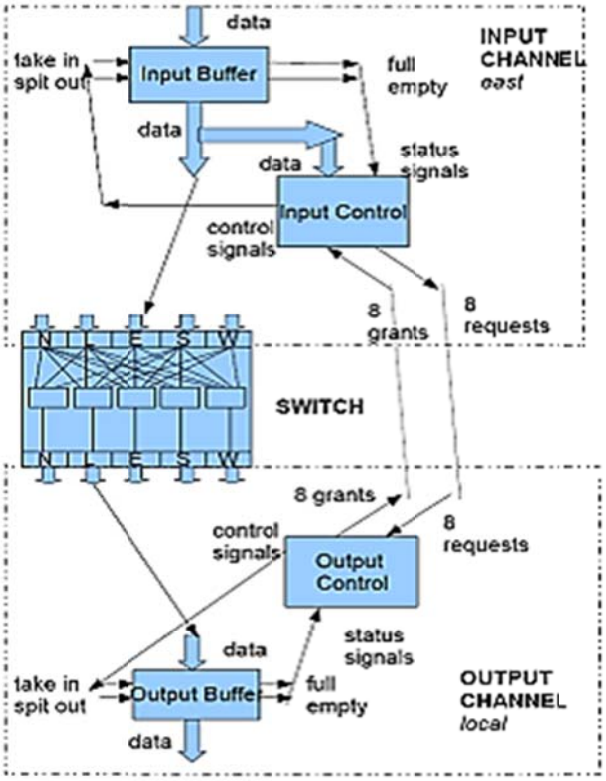

Figure 4. Interaction among I/O Channels

E. Arbitration

In static arbitration schemes, the priority of each port is chosen during design. First, we use a generic fixed scheme where priority is given to the north first, and degrades clockwise. Then we design a fixed scheme that varies for each port, and also includes optimizations in the north and south due to the XY routing algorithm. Static schemes cannot avoid deadlock. In dynamic arbitration schemes, the priority of each port is calculated during run-time by the unit. We include 3 counting schemes and a coin passing scheme. The counting schemes all have similar area results, but their performance depends on the application. The first scheme gives priority to the port that has been busiest (sending the most requests). The Next scheme gives priority to the port that has been waiting the longest. Here, the arbitration unit counts cycles after a request has been received for all ports. The last counting scheme gives priority to the port that sends the least packets (opposite to the first scheme). Finally, in coin passing scheme, one input port is assigned the coin. The port assigned with the coin, has priority, until it has been granted. Then the coin is passed to the next port, clockwise. If the port with the coin is not making a request, the unit grants the request of the port closes to it, again clockwise. This scheme is much like round robin used in many FPGA NoC router implementations.

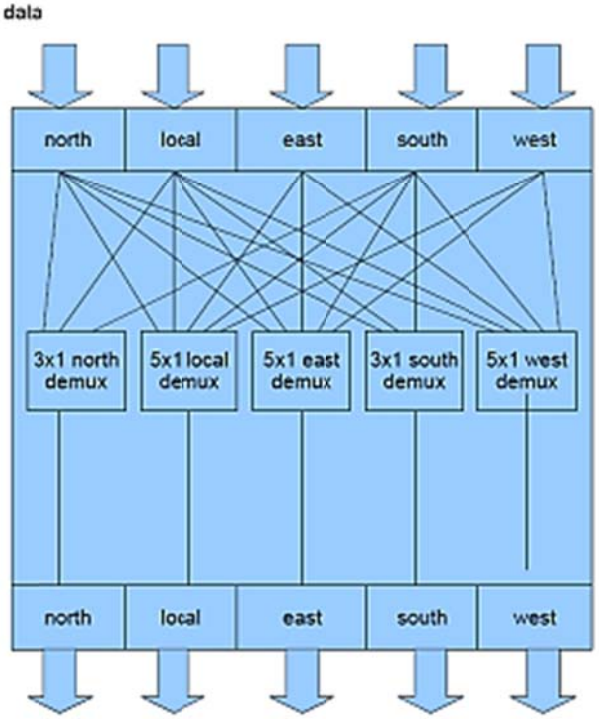

Figure 5. Proposed Switching Fabric for Router

V. Experimental Evaluation Results

In this section we first present the experimental framework for exploring NoC router parameters. We then present a summary and analysis of synthesis and simulation results. We use Altera Quartus to synthesize the NoC-based system to obtain area and clock frequency values [20]. We chose to target a popular Stratix II FPGA family, device EPIS40F1508C5. We use Mentor Graphics Modelsim [21], to model IP traffic and simulate activity. All router components and test bench wrappers have been implemented in VHDL. Components were originally tested for functionality in Quartus environment. The router coordinates are set before simulation. We vary values of the following three parameters: arbitration type, flit size, and IP-core-placement (IP-core-to-router mapping). Results of simulations focused on overall latency in terms of cycles. Using synthesis results, latency was later calculated in units of microseconds. Average throughput was also calculated (although not shown) using total number of packets sent.

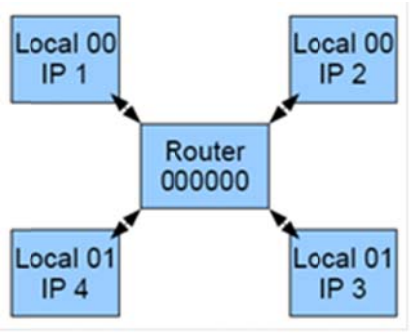

Figure 6. Single Router Configuration

A. *Experimental Evaluation Framework*

When setting up experiments for evaluating the proposed parameterizable router, we could not find commonly accepted techniques for router evaluation. This aspect of research is still work in progress with no commonly accepted benchmarks for NoC/router evaluation [22].Therefore, we created test scenarios to model NoC network traffic to allow mesh architecture and stand-alone router evaluation, and interface with our network protocols.

For exploring different arbitration schemes, we implemented a wrapper around our stand-alone router with different arbitration units embedded within. This wrapper focused on sending packets to the local node to create arbitration dilemmas, although packets were sent and received by all ports. In total 111 packets were sent out from various ports in groups from as small as 1 to as large as 10.

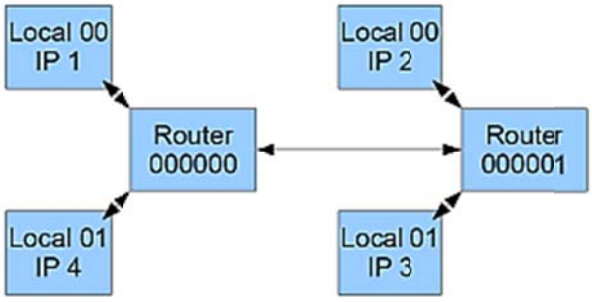

Figure 7. 1x2 Mesh Configuration

Next, for exploring effects of using different flit sizes, we implemented various wrappers around our stand-alone router with different datapath sizes (flit size). This wrapper was based on the traffic in the arbiter type test, but with larger packet sizes. In total 544 packets were sent out from various ports in groups from as small as 16 to as large as 64. As the flit size was increased, the test was modified to send less packets to keep the amount of data transferred the same. For example, if 544 8-bit packets were sent, only half that (272 packets) would be needed for 16-bit flits.

Finally, we implemented various wrappers around different configurations of mesh size, number of local ports and mapping. This wrapper was designed to model a 4 IP core application. The 3 basic configurations without mapping are shown in figures 6, 7, and 8. In total 201 packets were injected into the mesh through local ports. IP core 1 acted as the central processing node sending a total of 160 packets to IPs 2 and 3. IP cores 2 and 3 acted as custom logic blocks receiving 20 packets at a time and responding 5 packets to IP 4. IP core 4 acted as an output display of some sort, receiving the resulting 5 packets from IPs 2 and 3 each stage in the application. The application ended with IP core 4 sending a final packet to IP core 1.

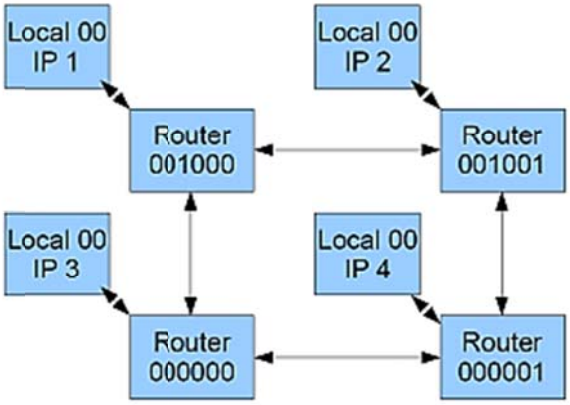

Figure 8. 2x2 Mesh Configuration

B. *Area Results*

The area results are shown in Figures 9, 10, and 11. Each parameter has two associated values, corresponding to synthesis optimizing for area (red) and synthesis optimizing for speed (blue). In most cases, when Quartus II is optimizing for speed, it tries to get rid of slow memory bits and LUTs as memory which can be very area expensive which can in turn slow down the design. It is important to note that this device has 384 M512's, 183 M4K's and 4 M1M rams, so any memory bits consumed by the routers buffering requirements should not constrain the design of the SoC. Our 5-port, 8-bit router consumes only 598 (1.45%) LEs, running at a frequency of over 100MHz, making it one of the most competitive FPGA implementations with all standard features. For arbitration schemes, static arbiters are much smaller, with the round robin like arbiter being the closest dynamic scheme, a couple hundred LE's larger. To no surprise, we observe increases in area as flit size increases, however they are not as significant as expected. In the configuration test, it can be seen that using fewer routers (although they have more local ports) reduces the overall network area.

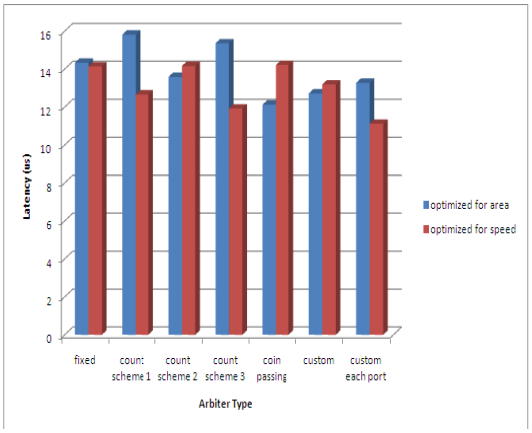

Figure 9. Effects of Arbiter Type on Area Utilization

C. *Effects of Arbitration Scheme on Latency*

Figure 12 shows the latency results for different arbitration schemes. The dynamic schemes outperformed the static fixed scheme, but not by much. However when each port has its fixed scheme custom designed, the gap between them closes. Overall the fixed scheme should be preferred because it gives good latency and area results. If the chance of deadlocks is high, then dynamic counting and coin passing schemes should be preferred even though they consume more area.

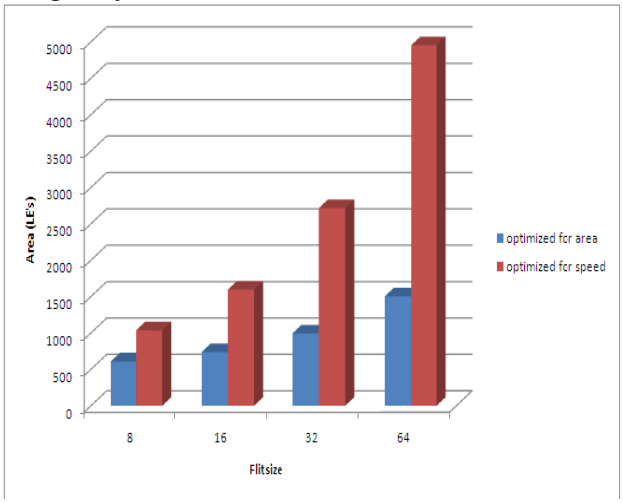

Figure 10. Effects of Flit Size on Area Utilization

D. *Effects of Flit Size on Latency*

Figure 13 shows latency results for different flit sizes. As flit size is increased, the latency decreases drastically. Note that when flit size is doubled, latency is reduced by approximately half. Current FPGAs restrict synthesis for larger flit sizes due to I/O pin restrictions, but it would be interesting to see how long this trend continues before a saturation point is reached. Based on these results we recommend using the largest flit possible given the area constraints for NoC implementation on FPGA.

E. *Effects of Configuration on Latency*

Figure 14 shows latency results for different configurations (IP-core-to-router mapping schemes). Although adding more local ports increases router congestion, it leads to decreased latency as packets did not have as many hops to travel. The most important result this paper presents is the importance of multi-router connections to a single IP core. This case explores connecting an IP core to more than 1 router, in this case connecting the main IP core (busiest) to both routers in a 1x2 mesh. This method shortens the number of hops for the main communication node, while decreasing router congestion. It was also important in this case, as IP core 1 was communicating with 2 nodes, both in separate direction. Therefore, IP core 1 was able to send packets to one core while waiting for a routing decision to be for connection to another core (1 X 2 map 2 extended in Figure 14).

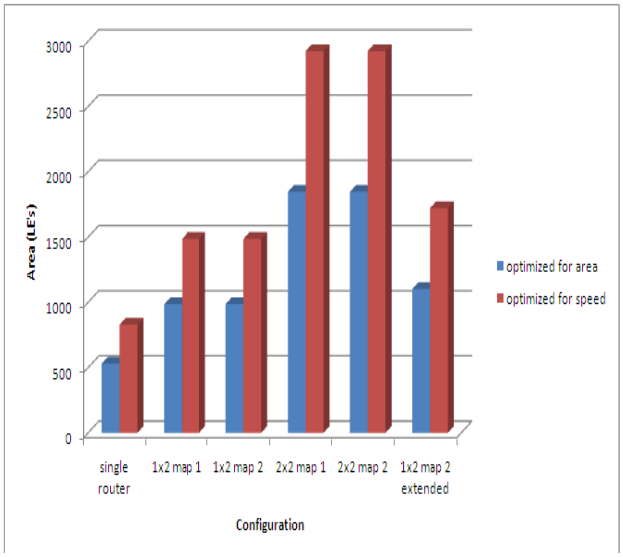

Figure 11. Effects of IP Core Placement on Area Utilization

V. Conclusion

This paper presented the design and evaluation results for a parameterizable NoC router for FPGAs. A NoC router platform was created with flexible parameters such as mesh size, number of local ports, channel size, and arbitration type. Our 5-port, 8-bit router consumes only 598 LE s running at a frequency of over 100MHz, making it one of the most competitive FPGA implementations with all standard features. Experimental evaluation results show that multiple local-port routers reduce overall network area and latency for our SAF router. The severity of latency and area trade-offs for different arbiter types, flit sizes and configurations have been presented. We expect that the evaluation results presented here will be useful to designers who want to implement NoC-based systems on FPGAs.

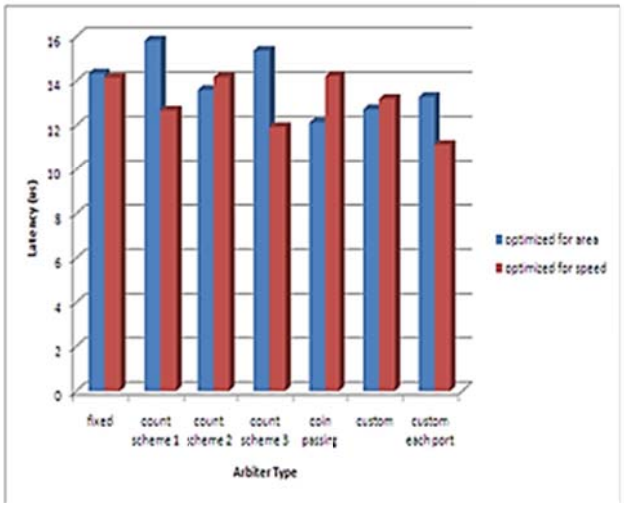

Figure 12. Effects of Arbiter Type on Latency

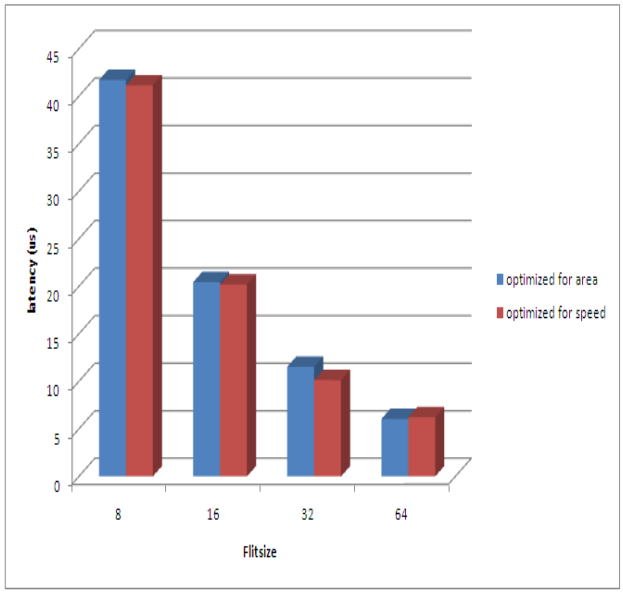

Figure 13. Effects of Flit Size on Latency

Follow up research can use the developed infrastructure to implement and evaluate NoC architectures with different communication mechanism parameters (switching mode, routing algorithm) to further decrease area and/or increase performance. Currently, another member of our research group is working on the design of a network interface to allow an IP core running Wishbone protocols to connect to our router which would allow router evaluation using real world applications.

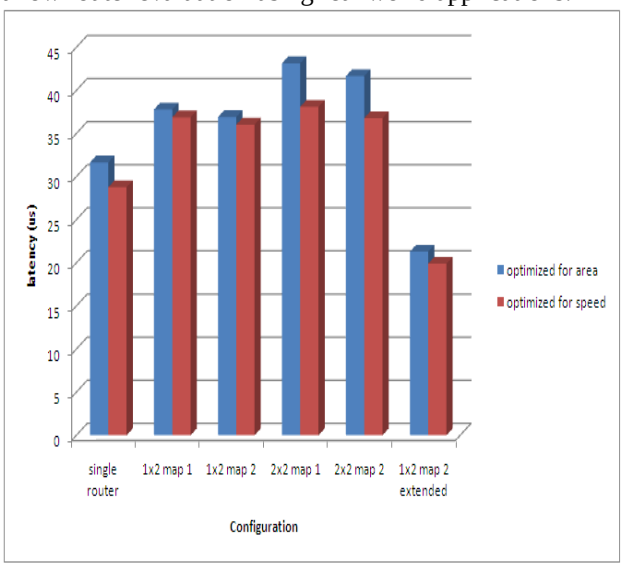

Figure 14. Effects of IP Core Placement on Latency

ACKNOWLEDGMENT

The authors would like to thank Thuan Le for his help and guidance.

REFERENCES

[1] Dally, William J., and Brian Towles. "Route packets, not wires: On-chip interconnection networks." In *Design Automation Conference, 2001. Proceedings*, pp. 684-689. IEEE, 2001.

[2] Ogras, Umit Y., R. Marcillescu, Hyung Gyu Lee, Puru Choudhary, Diana Marculescu, Michael Kaufman, and Peter Nelson. "Challenges and promising results in NoC prototyping using FPGAs." *Micro, IEEE* 27, no. 5 (2007): 86-95.

[3] Saldaña, Manuel, Lesley Shannon, and Paul Chow. "The routability of multiprocessor network topologies in FPGAs." In *Proceedings of the 2006 international workshop on System-level interconnect prediction*, pp. 49-56. ACM, 2006.

[4] Narasimhan, Ashok, O. Kumaravelu, and Ramalingam Sridhar. "An investigation of the impact of network parameters on performance of network-on-chips." In *Circuits and Systems, 2005. 48th Midwest Symposium on*, pp. 1617-1620. IEEE, 2005.

[5] Bjerregaard, Tobias, and Shankar Mahadevan. "A survey of research and practices of network-on-chip." *ACM Computing Surveys (CSUR)* 38, no. 1 (2006): 1.

[6] Marescaux, Théodore, Andrei Bartic, Dideriek Verkest, Serge Vernalde, and Rudy Lauwereins. "Interconnection networks enable fine-grain dynamic multi-tasking on FPGAs." In *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, pp. 795-805. Springer Berlin Heidelberg, 2002.

[7] Bartic, T. A., J-Y. Mignolet, Vincent Nollet, Theodore Marescaux, Diederik Verkest, Serge Vernalde, and Rudy Lauwereins. "Highly scalable network on chip for reconfigurable systems." In *System-on-Chip, 2003. Proceedings. International Symposium on*, pp. 79-82. IEEE, 2003.

[8] Moraes, Fernando, Ney Calazans, Aline Mello, Leandro Möller, and Luciano Ost. "HERMES: an infrastructure for low area overhead packet-switching networks on chip." *INTEGRATION, the VLSI journal* 38, no. 1 (2004): 69-93.

[9] L. Ost, A. Mello, J. Palma, F. Moraes, N. Calazans, "MAIA -a framework for networks on chip generation and verification," *Proc. of IEEE Design Automation Conference*, Jan. 2005, Page(s): 49-52.

[10] Zeferino, C.A.; Kreutz, M.E.; Susin, A.A., "RASoC: a router soft-core for networks-on-chip," *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol.3, no., pp.198,203 Vol.3, 16-20 Feb. 2004
doi: 10.1109/DATE.2004.1269230

[11] Zeferino, Cesar Albenes, and Altamiro Amadeu Susin. "SoCIN: a parametric and scalable network-on-chip." In *Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings. 16th Symposium on*, pp. 169-174. IEEE, 2003.

[12] Freitas, Henrique C., Dalton M. Colombo, Fernanda Lima Kastensmidt, and Philippe Olivier Alexandre Navaux. "Evaluating network-on-chip for homogeneous embedded multiprocessors in FPGAs." In *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, pp. 3776-3779. IEEE, 2007.

[13] Hilton, C.; Nelson, B., "PNoC: a flexible circuit-switched NoC for FPGA-based systems," *Computers and Digital Techniques, IEE Proceedings* - , vol.153, no.3, pp.181-188, 2 May 2006.

[14] Sethuraman, Balasubramanian, Prasun Bhattacharya, Jawad Khan, and Ranga Vemuri. "LiPaR: A light-weight parallel router for FPGA-based networks-on-chip." In *Proceedings of the 15th ACM Great Lakes symposium on VLSI*, pp. 452-457. ACM, 2005.

[15] Sethuraman, Balasubramanian. "Novel Methodologies for Performance & Power Efficient Reconfigurable Networks-

on-Chip." In *Field Programmable Logic and Applications, 2006. FPL'06. International Conference on*, pp. 1-2. IEEE, 2006.

[16] Sethuraman, B.; Vemuri, R., "optiMap: a tool for automated generation of NoC architectures using multi-port routers for FPGAs," *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol.1, no., pp., 6-10 March 2006.

[17] Véstias, Mário P., and Horácio C. Neto. "Area and performance optimization of a generic network-on-chip architecture." In *Proceedings of the 19th annual symposium on Integrated circuits and systems design*, pp. 68-73. ACM, 2006.

[18] Janarthanan, A.; Swaminathan, V.; Tomko, K.A., "MoCReS: an Area-Efficient Multi-Clock On-Chip Network for Reconfigurable Systems," *VLSI, 2007. ISVLSI '07. IEEE Computer Society Annual Symposium on*, vol., no., pp.455- 456, 9-11 March 2007.

[19] Wang, Ling, Jianye Hao, and Feixuan Wang. "Bus-Based and NoC Infrastructure Performance Emulation and Comparison." In *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*, pp. 855-858. IEEE, 2009.

[20] Altera Inc. http://www.altera.com.

[21] MentorGraphics Inc. http://mentorgraphics.com.

[22] Marculescu, Radu, Umit Y. Ogras, Li-Shiuan Peh, Natalie Enright Jerger, and Yatin Hoskote. "Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives." *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 28, no. 1 (2009): 3-21.

[23] Brugge, Mike and Khalid, Mohammed A. S. "Design and evaluation of a parameterizable NoC router for FPGAs (abstract only)," Proc. of 18th ACM/SIGDA International Symposium on FPGAs, Feb. 2010.