Performance and Resource Optimization of NoC Router Architecture for Master and Slave IP Cores[1]

Glenn Leary, Krishna Mehta, Karam S. Chatha
Arizona State University
Department of CSE, PO BOX 875406, Tempe, AZ 85287-5406
{gleary, kbmehta, kchatha}@asu.edu

ABSTRACT
System-on-Chip architectures incorporate several IP cores with well defined master and slave characteristics in terms of on-chip communication. The paper presents a parameterized NoC router architecture that can be optimized for performance and resource requirement by exploiting the master or slave behavior of the cores that are attached to it. We implemented the proposed router architecture for the IBM Coreconnect protocol and mapped it on the Xilinx Virtex series FPGA. We compared the FPGA based implementation against industry strength bus design that supports the IBM Coreconnect protocol, namely processor local bus (PLB). For similar resource requirements, our design demonstrated a 97.6% increase in throughput and 76.53% decrease in latency in comparison to the PLB. We also compared the proposed architecture with an existing NoC router design that is oblivious to master/slave IP cores. In the case of a router with all shared slaves our design resulted in 65.9% reduction in resources, 548% increase in throughput and 84.7% reduction in latency.

Categories and Subject Descriptors
B.4 [**Input/Output Data Communications]:** Interconnections

General Terms
Performance, Design

Keywords
Network-on-Chip, FPGA

1. INTRODUCTION
Network-on-Chip (NoC) has emerged as the pre-dominant technology to replace bus based architectures for on-chip communication in System-on-Chip (SoC) devices. There are several factors that have led to the advent of NoC. Increase in interconnect propagation delay relative to gate delay due to technology scaling implies that end to end on-chip communication requires several clock cycles. Thus, synchronous on-chip communication as assumed by several bus based architectures is no longer feasible. The emergence of Globally Asynchronous Locally Synchronous (GALS) design methodology based multi-processor SoC devices has also raised the need for global asynchronous communication. Further, the requirement for concurrent high performance communication cannot be easily addressed by traditional bus based architectures.

NoC addresses several of the on-chip communication challenges of nanoscale technologies by supporting asynchronous packet switching based communication. Long signal propagation delays are effectively pipelined by introducing multiple routers along the path. NoC supports high performance concurrent communication as the various routers operate in a decentralized manner.

NoC can be classified into two broad categories based on their topologies. Regular topologies such as mesh, torus or hypercube are suitable for processor architectures aimed at general purpose computing. Irregular or custom topologies are suitable for application specific SoC such as media-processors where the various cores demonstrate fairly well defined on-chip communication patterns. Irregular NoC have been demonstrated to be superior in router (resource) requirements and power consumption for application specific SoC in comparison to regular architectures [1].

The paper addresses the router design for an irregular topology NoC aimed at application specific SoC architectures. In the application specific SoC certain cores act as masters that initiate requests, and several other cores are slaves that respond to requests. Such well defined master/slave relationships offer the potential for optimizing the architecture of a router based on the cores attached to it. The optimizations include customization of the router ports for master or slave cores, and reduction in the number of arbiters and other controllers based on the exclusive or shared nature of the slave with which a master core interacts. We discuss router architectures for several configurations from single master/multiple slaves to multiple masters with both shared and exclusive slaves to a network of such routers.

We present experimental results that evaluate the performance and resource usage of the proposed router architecture by implementing the design for the IBM Coreconnect on-chip communication protocol and mapping it on the Xilinx Virtex II series FPGAs. We compared the proposed design with existing processor local bus (PLB) that implements the Coreconnect protocol and also against a NoC router architecture that is oblivious to master/slave cores.

[1]The research presented in this paper was supported in part by grants from the National Science Foundation (Career CCF-0546462 and CNS-0509540).

CODES+ISSS'07, September 30–October 3, 2007, Salzburg, Austria.

The remainder of this paper is organized as follows. In Section 2, we review previous work related to NoC. In Section 3, we present our novel NoC custom router architecture. We present the experimentation and results in Section 4. Finally, we summarize our contribution and discuss future work in Section 5.

2. PREVIOUS WORK

In recent past, several router architectures have been proposed for NoC. In the following paragraph, in the interest of space, we discuss only a few of the existing architectures. SPIN [2] was one of the seminal NoC designs that supported fat-tree topologies. Proteo [3] is a VSIA compliant NoC architecture that can be configured for ring, star and bus topologies. Xpipes [4] is a parameterized router design that can be utilized in irregular topologies. Nostrum [5] is a router design for mesh topologies that supports both best effort and guaranteed throughput traffic classes by reserving time slots. AEthereal [6] is also a mesh based NoC architecture that supports guaranteed throughput traffic by utilizing a centralized scheduler. QNoC [7] is another router architecture for planar mesh based topologies that also supports multiple levels of service classes for on-chip traffic. Wang et al. [8] and Banerjee et al. [9] presented power and power consumption models for NoC routers aimed at mesh topologies. Hu et al. [10] presented an analytical model for buffer optimization in application specific NoC. All the existing router architectures do not consider if a master or slave core is attached to the router. In contrast we present optimizations for a router aimed at irregular topologies that optimizes the resources in the router on the basis of the communication behavior of the core. An important side effect is that the operating frequency of the resulting design is also improved because of the reduction in critical paths within the router. The optimizations discussed as part of the paper are generic and can be applied to other existing router architectures.

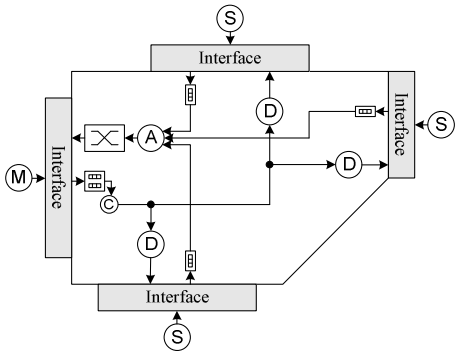

Figure 1. Single master, multiple slaves router design

Destination id	Source id	Payload (core protocol)

Figure 2. Format of NoC flit

3. ROUTER ARCHITECTURE

In the following sections we begin the discussion with a basic router supporting the communication between a single master and multiple slaves. We then extend the router architecture to support multiple masters with several slaves some of which are exclusive to certain masters while others are shared by a subset of masters. Finally, we consider the design of a router that includes connections to other routers with the masters and slaves being distributed across the network. In each of these architectures we support communication with split transactions. That is, each master can issue multiple read and write transactions to the slaves without waiting for the responses. We also discuss simplification of the routers to consider blocking read and write where a master waits for the response before issuing another request.

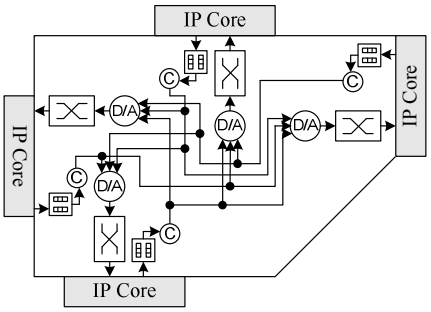

Figure 3. Router design oblivious to master/slave cores

3.1 Single Master and Multiple Slaves

The basic router architecture for a single master (M)/multiple slave (S) configuration is shown in Figure 1. The interface blocks that connect the masters and slaves to the routers perform the function of protocol conversion from the core to the network. For example, we implemented the interface for the IBM Coreconnect protocol to evaluate the performance of our design.

Each flit in the network has a control word and payload as shown in Figure 2. The control word includes the destination id and source id. The payload contains the word associated with the core native protocol. In our case we interfaced the NoC with the IBM Coreconnect protocol, and therefore the payload consists of Coreconnect control word and data. Therefore the width of the payload is dependent upon the native protocol of the cores. The NoC can easily support communication between cores supporting different protocols by inclusion of appropriate interface logic. The width of the destination and source identifiers is given by "$\log_2$ N" where N is the total number of cores in the network.

The master requests are fed to virtual channels or FIFOs. The master utilizes multiple virtual channels (VC) to avoid congestion due to a slow slave. The VCs are denoted as two FIFOs in Figure 1. The requests are selected from the VC through a round robin priority mechanism and fed to the decoders (D) associated with all the slaves. The decoders determine if the request is associated with the particular slave based on the destination id within the flit. The slave responses are buffered into a single VC. All the responses from the slave virtual channel are fed back to the master through a crossbar (denoted by the box with "X") which is controlled by the arbiter (A). The arbiter supports a round robin priority mechanism. The master and slave VCs, and the master arbiter are required as we support split transactions. Since multiple slaves can respond at the same clock cycle, the responses need to be buffered in the slave virtual channel and also arbitrated.

We can now compare the proposed design with a router shown in Figure 3 that is oblivious to master and slaves cores connected to

it. In comparison to such a router our design eliminates the arbiters and crossbars associated with the slaves. As the slaves only receive requests from a single master, the arbiters and crossbar can be removed. Further, the multiple VCs at each slave interface are also replaced by a single virtual channel.

Finally, if the protocol only supports blocking transactions we further optimize the resources in the routers. We can remove the VCs in the master and slaves as only one request is outstanding at a given time instance. The master arbiter can also be replaced by a simple controller that sets the crossbar based on the outstanding request. In other words as the master is aware of the expected response from a particular slave core it can itself set the crossbar and eliminate the need for arbitration.

3.2 Multiple Masters and Multiple Slaves

The design for the router with multiple masters and multiple slaves is shown in Figure 4. In the figure we distinguish between slaves that are shared (S_S) between the masters, and exclusive slaves (S_E) that are not shared between the masters. The router design for the exclusive slave is the same as the previous section. The master requests are buffered in virtual channels. Each request is selected from the virtual channel in a round robin manner and fed to only those slaves with whom the master can communicate. Similar to the master logic in the previous design, the input to the shared slaves must now be arbitrated. Further, the slave arbiters are preceded by a decoder to deduce if the incoming request is meant for the particular slave. In the figure the decoder/arbiter pairs are represented by circles with D/A within them. Also, the slave responses are now stored in multiple virtual channels (2 in the figure) to overcome congestion due to a busy or slow master. Finally, the master arbiters are also replaced by decoder/arbiter pairs similar to the slaves.

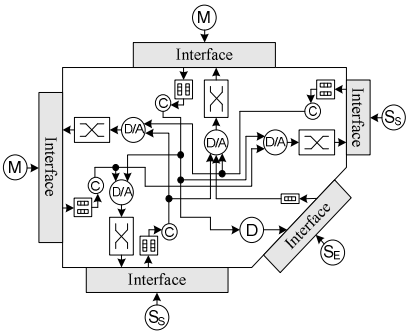

Figure 4. Multiple masters, multiple slaves router design

In comparison to the router design that is oblivious to master and slave cores (shown in Figure 3) the proposed design does not include the arbiters and crossbars for exclusive slaves. Similar to the previous design this is due to the fact that only a single master communicates with the exclusive slaves. The VCs at the exclusive slaves are also reduced to a single FIFO. Further, the arbiters at the shared slaves and masters only include inputs from the communicating masters and slaves, respectively. In other words a shared slave arbiter and crossbar only receive inputs from communicating masters and not all the cores on the router. Similarly, the master arbiter and crossbar only receive inputs from the slaves which whom the master communicates.

In the case of a protocol that only supports blocking transactions, we can eliminate the VCs from the masters and slaves, and the arbiters at the masters can be simplified as before.

3.3 Networked Masters and Slaves

The design for networked routers that are connected to multiple masters and slaves is shown in Figure 5. In comparison to the previous design, this router includes output and input ports for inter-router communication. The ports are symmetric with respect to each other. The design on the output port side consists of a decoder/arbiter pair and crossbar. Only those masters (or slave) are connected to the output port that performs communication with slave (or master) cores which are located along a path from the particular port. The output of the crossbar is fed into VCs located at the neighboring router. The VCs are dual clocked to support asynchronous communication. The write operation is synchronous with the clock of the neighboring router while the read operate is synchronous to the clock of the local router. The VCs of the input port are only connected to those cores that communicate with other cores located at the neighboring router. If a particular router is connected to multiple neighboring routers, then the decoder/arbiter pair of the output port can also receive inputs from the VCs associated with the input ports.

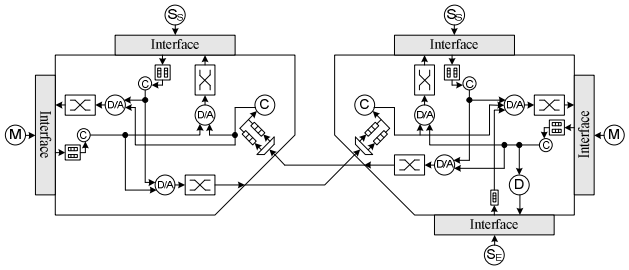

Figure 5. Networked router design

The output port for the router design that is oblivious to masters and slaves receives inputs from all cores and input ports of the router. Similarly, the input port is connected to all the cores and output ports of the router. In our design, in addition to all the optimizations specified in the previous two sections (3.1 and 3.2) we also optimize the number of connections between cores and the input/output ports associated with neighboring router communication.

The input and output port design remains unchanged in the case of blocking transactions. This is due to the fact that multiple masters can issue requests to neighboring slaves at the time instance. In other words, even though each master blocks on a outstanding request, there could be multiple requests being processed within the network. Consequently we require the decoder/arbiter and VC associated with the output and input ports, respectively.

3.4 NoC Design with Parameterized Blocks

We designed parameterized component blocks for decoders, arbiters, VCs and crossbars in VHDL. For example the decoder associated with a master/slave is parameterized on the number of inputs and the id of the respective core. The decoder associated with a output port is parameterized on a number of ids in addition to the number of inputs. Each id specifies the destination core that can be reached through the particular output port. The designer

specifies the topology of the network, the master and slave cores connected to the NoC, and path of communication routes. We utilize this information and the library of parameterized building blocks to generate the NoC. Therefore, the overall design flow can generate routers with variable number of input/output ports.

4. EXPERIMENTAL RESULTS

4.1 Comparison with PLB

In the first set of experiments, we compared our router architecture with the IBM Coreconnect PLB architecture [11]. We compared the resource usage by synthesizing the designs on the Xilinx Virtex II series FPGAs. The performance in terms of latency and acceptance rate was compared by utilizing synthetic traffic. The comparisons with the PLB were performed with blocking transfers for both the router as well as the PLB.

4.1.1 Resource Comparison

For the resource comparison we generated a design configuration consisting of two masters with four slaves. For our router we generated three different designs of this configuration: i) with all of the slaves shared, ii) with two shared slaves and two exclusive slaves, and iii) with all of the slaves exclusive. We then synthesized the designs using the Xilinx ISE synthesis tool to determine the resource requirements as well as the maximum operating frequency. The results are summarized in Table 1.

Table 1. Resource Comparison with PLB

Architecture Two Masters - Four Slaves	Frequency (MHz)	# Slices (13696)
NoC	--	--
All Shared Slaves	220	811
Two Shared Slaves	240	487
No Shared Slaves	349	172
PLB	--	--
PLB with DCR	169	430

As shown in Table 1, the required resources of our router are equivalent to PLB when two of the slaves connected to the router are exclusive to masters. (Table row "Two Shared Slaves.") Our router has a resource requirement of 487 slices and the PLB requires 430 slices. However, the maximum frequency of our router is 42% higher than the PLB with a value of 240 MHz compared to 169 MHz for PLB. If we eliminate the need for shared slaves (Table row "No Shared Slaves") our router shows a decrease in resources and increase in frequency. Here our router requires 172 slices, which represents a 60% decrease in resources over the PLB. Also, our router shows a 106.5% increase in frequency over the PLB with a value of 349 MHz compared to the PLB's 169 MHz. On the other hand, if we restrict our router to having all shared slaves (Table row "All Shared Slaves") our router requires 811 slices, which represents an increase of 87% in resources over the PLB. However, the frequency of our router remains higher with a value of 220 MHz compared with 169 MHz for the PLB. The increased resource requirement shown by our router was due to our router supporting split transactions. Therefore, the router had additional resources in terms of arbiters, decoders and VCs.

4.1.2 Performance Comparison

We compared the performance of our routers with PLB by injecting a synthetic stream of data traffic. The synthetic traffic was injected into the system at set injection rates in terms of the number of requests/responses per clock cycle per node. The traffic was generated with a uniform distribution using a random number generation. We then measured the acceptance rate in terms of requests/responses per clock cycle per node (Figure 6). We also measured the average latency in terms of clock cycles for the packets being injected into the system (Figure 7).

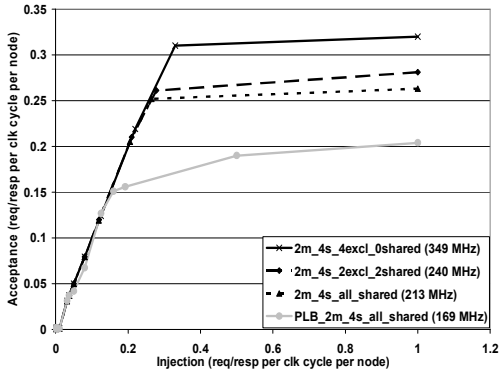

Figure 6. Acceptance Rate Comparison with PLB

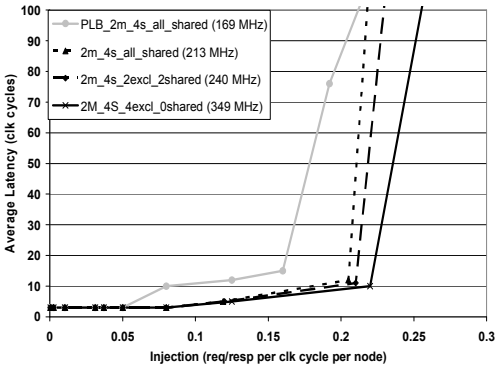

Figure 7. Latency Comparison with PLB

As shown in Figure 6, the acceptance rate of our router dramatically improves as the number of shared slaves is reduced. In the following discussion the throughput numbers in Gbps are presented by considering a data payload of 64 bits/flit and the frequency numbers shown in the figures. The PLB shows an acceptance rate of .151 (requests/responses per clock cycle per node, 1.635 Gbps) prior to congestion ("PLB_2M_4S_all_shared" curve in Figure 6). Our router with all shared slaves shows an acceptance rate of .205 (requests/responses per clock cycle per node, 2.79 Gbps) prior to congestion ("2M_4S_all_shared" curve in Figure 6), representing an 70.6% increase in performance over the PLB. If we reduce the number of shared slaves in our router to two and set the remaining two slaves as exclusive slaves ("2M_4S_2excl_2shared" curve in Figure 6), the acceptance rate increases to .21 (requests/responses per clock cycle per node, 3.23 Gbps), which represents a 97.6% increase over the PLB. Furthermore, if all of the slaves are exclusive ("2M_4S_4excl_0shared" curve in Figure 6), the acceptance rate again increases to a value of .219 (requests/responses per clock cycle per node, 4.89 Gbps), representing a performance increase of 199.1% over the PLB.

Figure 7 shows the effect that the reduction in the number of shared slaves has on the latency of the router. The PLB

configuration ("PLB_2M_4S_all_shared" curve in Figure 7) shows a latency of 15 clock cycles prior to congestion. The router with a configuration of all shared slaves ("2M_4S_all_shared" curve in Figure 7) shows a latency of 12 clock cycles, representing a 20% decrease in the latency. If the number of shared slaves is reduced to two with the remaining two slaves being exclusive ("2M_4S_2excl_2shared" curve in Figure 7) the latency drops to a value of 11 clock cycles, showing a 26.66% decrease in latency over the PLB. If all of the slaves are configured as exclusive ("2M_4S_4excl_0shared" curve in Figure 7) the latency drops to a value of 10 clock cycles, representing a decrease in latency of 33.33% over the PLB. However, the PLB is congesting at a much lower injection rate, .16 (requests/responses per clock cycle per node), compared with the routers, which are congesting at .25 (requests/responses per clock cycle per node). Also, if the differing frequencies are taken into account the decrease in latency in terms of nanoseconds (ns) is much more dramatic. With the PLB showing a latency of 88.65 ns while the router configurations demonstrate latencies of 23.45 ns for all shared slaves, 20.8 ns for 2 shared slaves, and 14.325 ns for all exclusive slaves. This represents decreases in latency of 73.54%, 76.53%, and 83.84% respectively.

4.2 Comparison with Existing Router

In our second set of experiments we compared our router architecture with a NoC router architecture which is oblivious to master/slave cores [9]. The resource usage comparison was performed by synthesizing the design. The performance comparison was performed with synthetic streams and also a JPEG benchmark.

4.2.1 Resource Comparison

We compared the resource usage for router designs with 2 masters and 3 slaves. For our router we also generated three configurations: i) two shared slaves with one exclusive slave, ii) with one shared slave and two exclusive slaves, and iii) with all exclusive slaves. We synthesized all of the configurations using the Xilinx ISE synthesis tool to determine the total number of slices required by each configuration along with the maximum operating frequency for each. The results are summarized in Table 2.

Table 2. Resource Comparison with Existing Router

Architecture Two Masters - Three Slaves	Frequency (MHz)	# Slices (13696)
NoC	--	--
All Shared Slaves	238	477
Two Shared Slaves	241	401
One Shared Slave	328	246
No Shared Slaves	337	234
Oblivious NoC	--	--
Five Ports	76	1176

As shown in Table 2 our router performs better in resource requirements as well as maximum frequency in every configuration over the pre-existing NoC router. The pre-existing router displayed a resource requirement of 1176 slices (Table row "Five Ports"). While our router with all shared slaves (Table row "All Shared Slaves"), showed a resource requirement of 477 slices, representing a 59% decrease in resource requirements over the pre-existing router. Our router also demonstrated a maximum frequency of 238 MHz, representing a 213% increase over the pre-existing router's maximum frequency of 76 MHz. As the number of shared slaves was reduced, our router demonstrated a further decrease in the required resources and an increase in the maximum frequency. With one shared slave (Table row "One Shared Slave"), the required resources reduces to 246 slices, representing an 89% reduction in resources compared with the pre-existing router. The frequency shown was 328 MHz, representing a 331.57% increase over the pre-existing router. This trend continues as the number of shared slaves is reduced to zero (Table row "No Shared Slaves"). With all of the slaves as exclusive our router shows a resource requirement of 234 slices and a frequency of 337 MHz. This represents a 90% decrease in resources and a 343% increase in frequency compared with the pre-existing router architecture.

4.2.2 Performance Comparison

We utilized a similar experimental set-up with synthetic data traffic as that utilized for the PLB comparison. However, we used split transactions instead of blocking transactions. We also performed the comparison using similar traffic patterns on equivalent configurations on both routers. The first configuration was 2 masters and 3 slaves with all of the slaves shared. The second configuration was 2 masters and 3 slaves with all of the slaves exclusive. The results for acceptance rate and latency are shown in Figures 8 and 9, respectively.

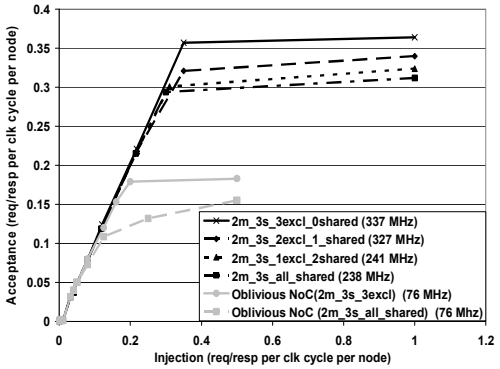

Figure 8. Performance Comparison with Existing NoC

As shown in Figure 8, our router performs better than the oblivious router in both configurations. The throughput numbers in Gbps are presented by considering a data payload of 64 bits/flit and the frequencies shown in the figures. For the 2 masters and 3 slaves with all of the slaves shared, the oblivious router shows an acceptance rate of .1084 (requests/responses per clock cycle per node, .504 Gbps) prior to congestion ("Oblivious NoC (2m_3s_all_shared)" curve in Figure 8). For the same configuration our router shows an acceptance rate of .215 (requests/responses per clock cycle per node, 3.28 Gbps) prior to congestion ("2m_3s_all_shared" curve in Figure 8). This represents an increase in performance of 548% over the oblivious router architecture. For the router configuration in which all of the slaves are exclusive the oblivious router shows an acceptance rate of .153 (requests/responses per clock cycle per node, .712 Gbps) prior to congestion ("Oblivious NoC (2m_3s_3excl)" curve in Figure 8). Our router architecture shows an acceptance rate of .221 (requests/responses per clock cycle per node, 4.78 Gbps) prior to congestion ("2m_3s_3excl_0shared" curve in Figure 8). This represents an increase in performance of 571%.

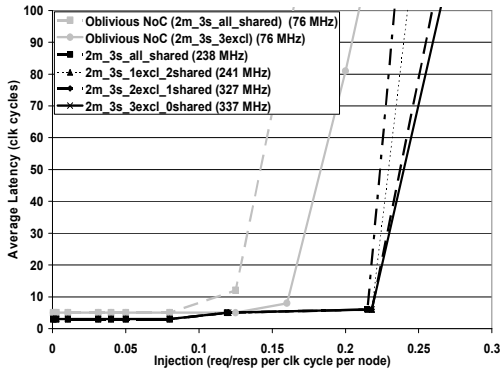

Figure 9. Latency Comparison with Existing NoC

Figure 9 shows the comparison between the routers in terms of latency. As shown in Figure 9, the latency of our router is lower than the pre-existing oblivious one. The pre-existing oblivious router with all slaves shared ("Oblivious NoC (2m_3s_all_shared)" curve in Figure 9) demonstrates a latency of 12 clock cycles prior to congestion. Our router demonstrates a latency of 6 clock cycles ("2m_3s_all_shared" curve in Figure 9). This represents a decrease in latency of 50%. However, if the differing frequencies are taken into account this decrease is much more dramatic. The oblivious router shows a latency of 165 ns while our router shows a latency of 25.2 ns. This represents a decrease in latency of 84.7%. For the configuration in which all of the slaves are exclusive the pre-existing oblivious router shows a latency of 8 clock cycles prior to congestion ("Oblivious NoC (2m_3s_3ecl)" curve in Figure 9). Our router shows a latency of 6 clock cycles ("2m_3s_3excl_0shared" curve in Figure 9). This represents a 25% decrease in latency. However, if the frequencies of the routers are taken into account our router performs much better. In this case the oblivious router has a latency of 110 ns and our router has a latency of 17.76 ns. This represents an 83.85% decrease in latency.

Table 3. JPEG Performance Comparison

	One Hop Latency (clk cycles)	Two Hop Latency (clk cycles)
Master/Slave Aware NoC	3 (16.17ns)	6 (32.39 ns)
Master Slave Oblivious NoC	8 (105.46 ns)	16 (210.04 ns)

4.2.3 Comparison for JPEG

We also compared the performance of the two designs by considering a JPEG benchmark from Opencores.org [12] that had 2 masters and 7 slaves (primarily memory banks). We considered a NoC design with 3 routers that were configured as follows: R1 (M1, S1, S2, S6, R2), R2 (M1, S2, S4, R1, R3) and R3 (S3, S5) where "R", "M" and "S" denote router, master and slave, respectively. The parenthesized list for each router denotes its connectivity. Table 3 summarizes the latency comparisons for traffic traces that go through one hop and two hops (maximum number of hops for our design). The comparisons are in nanoseconds due to the different operating frequencies of the two designs. The percentage reductions in one and two hop latencies of our router design over the existing design were 84.66% and 85.4%, respectively. The large reductions are primarily due to the higher operating frequency of our design. The resource requirements for our router design were 1187 slices as opposed to 3052 slices for the existing design (61.1% reduction).

5. CONCLUSION

The paper presented a parameterized NoC router architecture that exploits the master/slave communication behavior of the SoC cores to optimize the resources. Besides the reduction in the area requirements, the optimizations also lead to substantial increase in operating frequencies and consequently performance. We evaluated the router design by extensive experimentation with both an industrial strength bus design based on IBM Coreconnect protocol (PLB), and an existing router design that does not consider the optimizations. Our design demonstrated a 97.6% increase in throughput and 76.53% decrease in latency in comparison to the PLB while utilizing comparable resources (for 2 masters, 2 exclusive slaves and 2 shared slaves). In comparison to an existing NoC router architecture that is oblivious to master/slave cores connected, our design resulted in 65.9% reduction in resources. As well as a 548% increase in throughput and 84.7% decrease in latency for all shared slaves and 571% increase in throughput and 83.85% decrease in latency for all exclusive slaves. Finally for the JPEG design our design obtained a reduction of 85.03% and 61.1% in average latency and resource requirements, respectively. Future work will address automated generation of irregular NoC topologies that are sensitive to master/slave behavior of the cores, and support for differentiated traffic classes offering multiple Quality-of-Service levels.

6. REFERENCES

[1] A. Jalabert, S. Murali, L. Benini, and G. De Micheli, "XpipesCompiler: A tool for instantiating application specific networks on chip," in Proc. DATE, 2004.

[2] A. Andriahantenaina, A. Greiner, Micro-network for SoC: implementation of a 32-port SPIN network, in: DATE, Munich, Germany, March 2003.

[3] D. Siguenza-Tortosa, J. Nurmi, Proteo: a new approach to network-on-chip, in: Proceedings of IASTED International Conference on Communication Systems and Network, Malaga, Spain, 2002.

[4] M. Dall'Osso, G. Biccari, L. Giovanninni, D. Bertozzi, L. Benini, Xpipes: a latency insensitive parameterized network-on-chip architecture for multi-processor SoCs, in: Proceedings of ICCD, San Jose, CA, October 2003.

[5] M. Millberg, E. Nilsson, R. Thid, S. Kumar, A. Jantsch, The Nostrum backbone—a communication protocol stack for networks on chip, in: VLSI Design Conference, Mumbai, India, January 2004.

[6] J. Dielissen, A. Radulescu, K. Goossens, E. Rijpkema, Concepts and implementation of the Philips network-onchip, in: IP-Based SOC Design, November 2003.

[7] E. Bolotin, I. Cidon, R. Ginosar and A. Kolodny, "QNoC: QoS architecture and design process for Network on Chip", Special issue on Networks on Chip, The Journal of Systems Architecture, December 2003.

[8] H.-S. Wang, L.-S. Peh, S. Malik, Orion: a power-performance simulator for interconnection network, in: International Symposium on Microarchitecture, Istanbul, Turkey, November 2002.

[9] N. Banerjee, P. Vellanki, K.S. Chatha, A power and performance model for network-on-chip architectures, in: DATE, 2004.

[10] J. Hu, U. Ogras and R. Marculescu, System-Level Buffer Allocation for Application-Specific Networks-on-Chip Router Design, IEEE Transactions on CADICS, Vol. 25, No. 12, December 2006.

[11] Xilinx Embedded Development Kit, http://www.xilinx.com.

[12] http://www.opencores.org