DESIGN, IMPLEMENTATION AND EVALUATION OF A CONFIGURABLE NoC FOR AcENoCS FPGA ACCELERATED EMULATION PLATFORM

A Thesis

by

SWAPNIL SUBHASH LOTLIKAR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2010

Major Subject: Computer Engineering

DESIGN, IMPLEMENTATION AND EVALUATION OF A CONFIGURABLE NoC FOR AcENoCS FPGA ACCELERATED EMULATION PLATFORM

A Thesis

by

SWAPNIL SUBHASH LOTLIKAR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Paul V. Gratz
Committee Members,	A. L. Narasimha Reddy
	Eun Jung Kim
Head of Department,	Costas N. Georghiades

August 2010

Major Subject: Computer Engineering

ABSTRACT

Design, Implementation and Evaluation of a Configurable NoC for AcENoCs FPGA Accelerated Emulation Platform. (August 2010)

Swapnil Subhash Lotlikar, B.E., National Institute of Technology Karnataka, India

Chair of Advisory Committee: Dr. Paul V. Gratz

The heterogenous nature and the demand for extensive parallel processing in modern applications have resulted in widespread use of Multicore System-on-Chip (SoC) architectures. The emerging Network-on-Chip (NoC) architecture provides an energy-efficient and scalable communication solution for Multicore SoCs, serving as a powerful replacement for traditional bus-based solutions. The key to successful realization of such architectures is a flexible, fast and robust emulation platform for fast design space exploration. In this research, we present the design and evaluation of a highly configurable NoC used in AcENoCs (Accelerated Emulation platform for NoCs), a flexible and cycle accurate field programmable gate array (FPGA) emulation platform for validating NoC architectures. Along with the implementation details, we also discuss the various design optimizations and tradeoffs, and assess the performance improvements of AcENoCs over existing simulators and emulators.

We design a hardware library consisting of routers and links using verilog hardware description language (HDL). The router is parameterized and has a configurable number of physical ports, virtual channels (VCs) and pipeline depth. A packet switched NoC is constructed by connecting the routers in either 2D-Mesh or 2D-Torus topology. The NoC is integrated in the AcENoCs platform and prototyped on Xilinx Virtex-5 FPGA.

The NoC was evaluated under various synthetic and realistic workloads generated by AcENoCs' traffic generators implemented on the Xilinx MicroBlaze embedded

processor. In order to validate the NoC design, performance metrics like average latency and throughput were measured and compared against the results obtained using standard network simulators. FPGA implementation of the NoC using Xilinx tools indicated a 76% LUT utilization for a 5x5 2D-Mesh network. A VC allocator was found to be the single largest consumer of hardware resources within a router. The router design synthesized at a frequency of 135MHz, 124MHz and 109MHz for 3-port, 4-port and 5-port configurations, respectively. The operational frequency of the router in the AcENoCs environment was limited only by the software execution latency even though the hardware itself could be clocked at a much higher rate. An AcENoCs emulator showed speedup improvements of 10000-12000X over HDL simulators and 5-15X over software simulators, without sacrificing cycle accuracy.

To my Dad, Mom and my Sister

ACKNOWLEDGMENTS

I am very grateful to my advisor Dr. Paul V. Gratz, for giving me an opportunity to work under him. Without his constant guidance, encouragement and his valuable suggestions, the work presented in this thesis would have been virtually impossible. Thank You Dr. Gratz. My sincere thanks also go to Dr. A. L. Narasimha Reddy and Dr. Eun Jung Kim for their willingness to be on my thesis committee.

I would like to thank Mr. Vinayak Pai, my colleague and project partner on the AceNoCs project, for all the technical support during this research. I would also like to thank all the other CAMSIN research group members for their valuable suggestions during the research meetings.

I would also like to thank my parents for supporting me through difficult times and inculcating in me the sense of perfection. Whatever I am today is because of their concern for me. Thank you Mom and Dad.

Finally, I would like to thank all my friends who directly or indirectly helped me during my studies at Texas A&M and during the course of my research work.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

A. Motivation

VLSI scaling has resulted in miniaturization of integrated circuits and increased integration on a single chip. In order to meet computational demands, current System-on-Chip (SoC) designs incorporate a large number and a wide variety of heterogenous intellectual property (IP) cores on a single chip. These designs use buses or dedicated links as a means to establish communication between various cores. However, buses and dedicated links have several problems. They have scalability issues which limit their usage to interconnecting a small number of cores. Additionally, long global wires are prone to noise and have high power consumption. Technology scaling has caused the wire delay to become a dominant fraction of the clock cycle time as compared to the logic delay [1]. It is therefore essential to keep the wire length to a minimum. In order to overcome the limitations of these architectures, alternate means of on-chip communication have to be developed to effectively utilize the available communication bandwidth.

A design paradigm, Network-on-Chip (NoC), proposed by Dally and Towles, serves as an effective replacement for buses or dedicated links in a system with large number of processing cores [2]. The dimensions of the NoC decide the number of communicating IP cores that can be connected to it. NoC borrows concepts from the well established and scalable domain of computer networking. NoCs consist of a collection of network interfaces (NIs), routers, links and they follow a set of protocols

---

The journal model is *IEEE Transactions on Automatic Control.*

to establish communication between the cores. They are characterized by various parameters like router architecture, network topology, routing algorithms and flow control mechanisms. NoCs offer high communication bandwidth with low latency as compared to the traditional bus based or point-to-point communication architectures.

The shift in focus from computation-centric to communication-centric designs and the reduced time-to-market requirement has caused communication infrastructure validation to play a major role in the chip design flow. A fast exploration of the vast design space offered by NoC communication infrastructure is vital to arrive at an optimal configuration that meets the communication demands of a particular application. Typically, validation was accomplished using software or hardware description language (HDL) simulators. These simulations are time-consuming due to the inherent sequential nature of software simulators. There is a great demand for validation tools which are both fast as well as cycle accurate. Field programmable gate array (FPGA) based emulators can be used as an alternative to software simulators as they help to reduce the validation time without compromising cycle accuracy. They combine the flexibility of the software simulators together with the cycle accuracy of the HDL simulators. Design validation using such emulators permits the user to exploit the parallel nature of hardware and simulate the HDL at actual hardware speeds. Such an approach helps to detect design and architectural problems early in the design cycle and provides an early access of the target platform to the software developers, thus reducing the number of re-spins and expediting the design cycle.

The main goal of this thesis is to introduce AcENoCs (Accelerated Emulation platform for NoCs), an emulation platform for validating on-chip interconnection networks [3]. AcENoCs is built around a robust hardware software framework which makes efficient utilization of available FPGA resources. The NoC itself is modeled using the FPGA's hardware resources, and hence efficiently exploits the parallel nature

of hardware. The traffic generation and statistical analysis components are implemented in software, utilizing a soft IP processor, leveraging the greater state space resources available to software. The work presented in this thesis makes the following contributions:

1. A highly parameterized hardware library consisting of routers, links and the interconnection network.

2. Implementation and emulation flow for FPGA accelerated NoC emulation using a Xilinx Virtex-5 FPGA.

3. A complete, cycle accurate and a flexible FPGA accelerated emulation platform for validating on-chip interconnection networks.

The AcENoCs emulation platform has been jointly developed by a team of two researchers. It is not possible to present the hardware and software framework totally independent of each other. A combined framework is presented in this thesis with a focus on the hardware design.

B. Thesis Organization

The thesis is organized as follows: Chapter II provides a background on NoC. Chapter III describes the previous work in the area of NoC simulation and emulation. Chapter IV describes the AcENoCs framework and the features of its individual components. Chapter V describes in detail the design and implementation of the AcENoCs NoC hardware library components. Chapter VI describes the FPGA implementation and emulation flow for AcENoCs emulation platform. Chapter VII presents the experimental results of network validation, AcENoCs performance evaluation and hardware evaluation. Conclusions and future work are presented in Chapter VIII.

CHAPTER II

BACKGROUND

An integrated circuit typically consists of processing elements, storage elements, communication infrastructure and input/output (I/O) interfaces as its primary components [4]. With advancements in technology, the processing elements have become much faster. However, the performance of the communication infrastructure has not scaled at the same rate. Therefore, in modern technologies, the design of communication infrastructure has taken center-stage in the chip design cycle. NoC is an on-chip communication infrastructure that applies the principles of computer networks to efficiently establish on-chip communication and improve performance over existing bus-based communication architectures. The sections below discuss the bus-based and NoC communication architectures and the terminology associated with the NoC based solutions.

A. Bus and Point-to-Point Architectures

Point-to-Point links and buses have been the simplest and the most widely used means of on-chip communication in the past. A point-to-point scheme consists of dedicated channels between communicating nodes in a system. A bus based communication system, on the other hand, consists of communicating nodes connected to a single shared channel which acts as a broadcast medium [5]. The nodes which initiate a transaction on the shared channel are termed as bus masters and the nodes which respond to requests from masters are called as bus slaves. Messages transmitted by one node can be intercepted by all the other nodes connected to the shared medium. If the messages are addressed to a particular slave then that slave node will respond

by sending an acknowledgement message to the master. A bus arbiter controls access to the shared resource by granting access to only one of the several requesting masters.

There are several disadvantages associated with these kind of communication architectures. Both point-to-point as well as bus based communication schemes are not very scalable and cannot efficiently handle the communication requirements of modern SoC architectures. The performance of a bus degrades as the number of requestors connected to the bus increases. This can be attributed to the fact that the bandwidth of the communication channel is shared among all the bus requestors. This results in the serialization of the requests to the bus, thus increasing communication latencies. Also, the complexity and the delay of the arbiter increases as the number of requestors to the bus increases. Technology scaling has caused wire delay to become a dominant component of the overall clock cycle time [1]. Long wires in point-to-point links as well as buses result in increased delays and are susceptible to noise. Hence, on-chip communication using these schemes is becoming expensive in terms of both power as well as speed in the era of deep-submiron technologies (DSM).

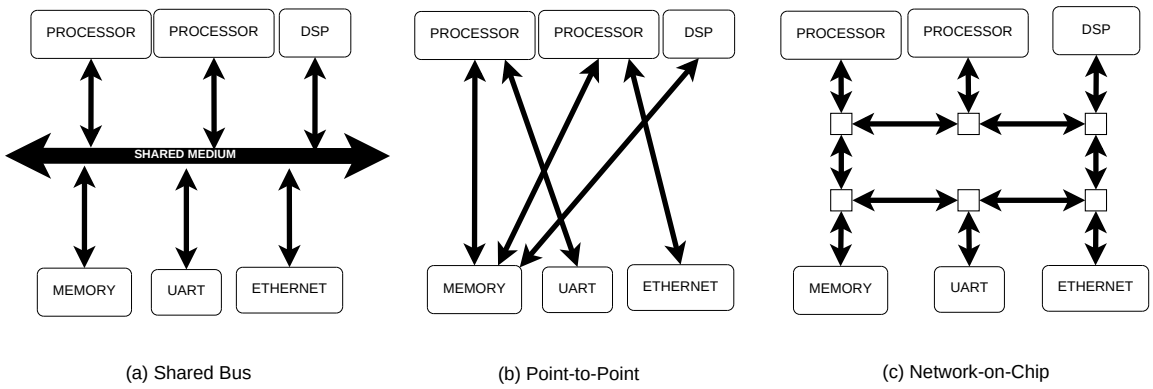

Figure 1. Bus, Point-to-Point and NoC Communication Architectures

B. Network-on-Chip (NoC)

The NoC is a highly scalable packet based communication architecture that overcomes the disadvantages of the bus based communication systems. Table I compares the bus based and NoC based on-chip communication schemes and has been adapted from Yoo et al. [6]. NoCs help to accomplish the transfer of maximum amount of information between communicating nodes within the least possible time. NoCs consist of routing elements connected by small point-to-point links forming a data routing network on chip. Unlike bus architectures, where the bus is occupied by one source node during the entire message transaction, the use of packet based communication in NoC allows for sharing of the links between various communicating nodes. This increases throughput and reduces communication latencies. NoC can be easily scaled by connecting additional routing elements to the existing network. The aggregate bandwidth of the network scales with increasing network size. NoCs support design reuse as the same routing element can be used to scale the NoC to higher dimensions. This reduces the time-to-market and validation costs. Thus, NoCs offer a highly efficient communication infrastructure for modern day SoC and Multicore architectures. Figure 1 shows some examples of bus, point-to-point and NoC based communication architectures.

C. NoC Architecture and Components

A NoC consists of routing nodes spread across an entire chip connected together by communication links [4]. A brief description of the various components of the NoC is provided below.

Processing Elements (PEs) are the computational elements of the chip. These can be general purpose processor cores, digital signal processing cores, arithmetic

Table I. Comparison of Bus and NoC Architectures

	BUS ARCHITECTURE		**NoC ARCHITECTURE**
-	A single master occupies the shared bus during entire transaction	+	Packet transactions share links in a multiplexed manner
-	Blocked transactions cause performance degradation	+	Multiple concurrent transactions possible thereby increasing throughput and lowering latency
-	Long bus wires prone to noise and error	+	Links between routing elements are short and hence less error prone
-	Long bus wires cause increased wire delays in DSM technology	+	Short point-to-point links keep wire delay to a minimum
-	Bus failure results in system failure	+	Multiple paths possible between two communicating elements improving fault tolerance
-	Shared bus is less scalable and gets slower as number of bus requestors increase	+	Aggregate network bandwidth scales with network size
-	All bus masters request a single arbiter. Arbiter complexity increases with number of requestors	+	Routing decisions are distributed across the network thus reducing complexity
+	Low area overhead	-	Additional area overhead due to addition of network routers
+	Design concepts are well understood. Low design complexity	-	NoC concept is relatively new. Design complexity is high

logic units, memory cores or any other specialized IP cores. They are the source and the sink for all the data in the on-chip communication network.

Network Interfaces (NIs) connect the processing elements to the main on-chip communication network. They decouple the computational elements from the communication infrastructure. NIs convert the messages generated by the PEs into packets and insert additional routing information based on the architecture of the under-

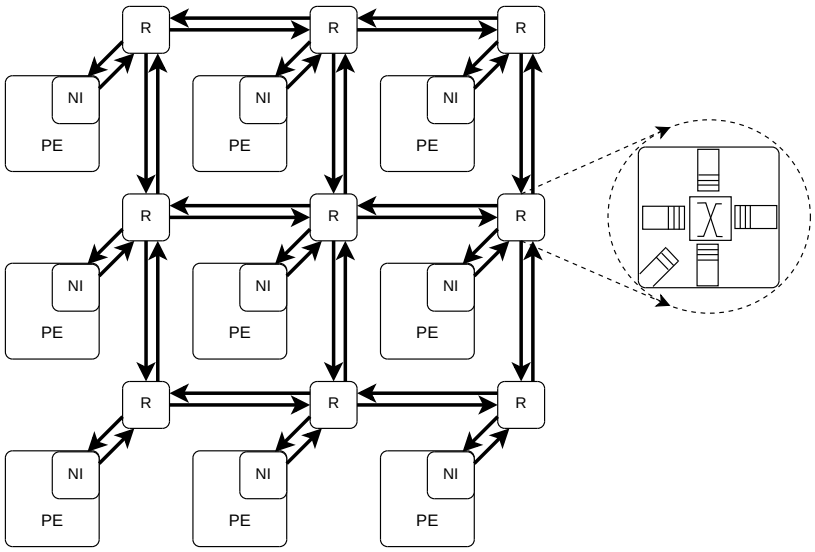

Figure 2. NoC Architecture

lying network. Packets are decomposed into smaller units called flow control units or flits which are transmitted over the network. Flits are further classified as head, body and tail flits. The head flit carries the routing information required to route the packet to its destination. The head flit allocates resources for the entire packet as it traverses from source to destination. The body and tail flits carry only the packet payload with no routing information and follow the head flit through the network. The tail flit de-allocates the resources which have been allocated to the packet by the head flit.

Routing Nodes are the heart of the communication network. They route the packets onto the appropriate link so that they can reach the intended destination. Routing protocols in conjunction with the routing information in the packet header are used to make routing decision at each routing node.

Channels or Links connect the routing nodes in an NoC. Two links are present between any two routers in the network, one each for data transmission in each

direction. Links provide the bandwidth required for data transmission. In addition to the data transmission links, additional links required for control may also be present.

Figure 2 depicts these components for a 3x3 NoC where the routing nodes connected as a grid.

D. NoC Topologies

Network topology refers to the static arrangement of routing nodes and links in an interconnection network. Selection of a good topology is essential to minimize the communication latency and maximize the bandwidth. The routing and the flow control schemes are heavily dependent on the type of the topology selected. Topologies can be classified into two categories : regular topologies and irregular topologies [5].

The most commonly used type of regular topology is the k-ary n-cube [7]. This topology consists of $N = k^n$ nodes arranged as a regular n-dimensional grid with k nodes in each dimension connected by a pair of communication links, one in each direction. Each of the nodes can act as an input or an output or a routing node. The most commonly used versions of the k-ary n-cube are the torus and mesh networks. Torus networks possess edge symmetry. All the nodes in each dimension form a ring. The edge symmetry of the torus network helps to improve the load balance across various communication channels. In a torus network all the nodes are of the same degree. Mesh networks are very similar to the torus, the only difference being the removal of the wrap around links along the edges. In a mesh network all the nodes in each dimension form a linear array. The removal of the edge symmetry in mesh networks can cause load imbalance for some of the traffic patterns. Nonetheless, the physical layout of these topologies is well matched to the chip's packaging constraints. They minimize the number of hops to the destination thus reducing the

communication latencies and improving the network throughput.

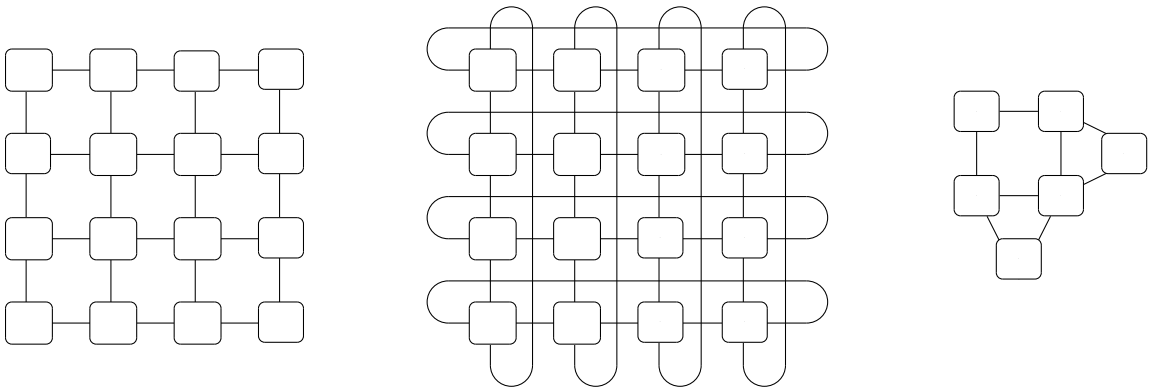

Figure 3. Mesh, Torus and Irregular Topologies

Irregular topologies are constructed by combining different regular topologies in hierarchical, hybrid or asymmetric fashion [4]. Figure 3 shows examples of all these topologies.

E. NoC Switching Techniques

Switching in an NoC is defined as the transport of data through the interconnection network. Two of the switching techniques employed in on-chip interconnection networks are circuit switching and packet switching. These techniques are briefly described below :

Circuit Switching involves the setup of a dedicated circuit from a source node to a destination node. One or more packets can be sent over the circuit. In this switching technique the packets carry only the payload and do not carry any routing information. Once all the packets have been transmitted the circuit can be disconnected. The advantage with this switching technique is that the circuit is setup prior

to packet transmission and provides guaranteed bandwidth along the circuit. The disadvantage with this switching technique is that the other nodes cannot use the path reserved for a circuit until it is freed. This results in increased delay for other nodes trying to send packets over the same path. Thus, circuit switching reduces the overall throughput of the interconnection network and increases the average latencies of communication [4], [5].

Packet Switching involves switching of the packets on a per-hop basis rather than setting up a dedicated route from the source to destination. In this switching technique the packet contains both routing information as well as payload. Since routing decisions are made at intermediate nodes in the interconnection network, there can exist multiple different paths from a source node to a destination node. The advantage with this approach is that since a particular path in the network is not reserved, the same path can be used by multiple packets. The disadvantage with this scheme is that since routing decisions are mode at every hop, it does not provide any guaranteed service along the packet transmission path [4], [5].

F. NoC Routing Algorithms

Switching is the transport of data, while routing is the intelligence behind it. Routing determines the path taken by a packet to travel from a source node to a destination node and is dependent on the network topology selected [5]. A good routing algorithm should meet two objectives. Firstly, a routing algorithm should balance the load across the network terminals even in the presence of non-uniform traffic conditions. Secondly, it should keep the lengths of the path from source node to destination node as small as possible. This section describes some of the routing techniques employed in on-chip interconnection networks.

Routing algorithms on the other hand can be classified as deterministic, oblivious or adaptive [5]. A brief description of these algorithms and examples is given below.

Deterministic Routing algorithms always choose the same path from a source node to a destination node even if multiple paths exist. The path chosen depends entirely upon the source and the destination coordinates. These algorithms are easy to implement and are deadlock free in nature. However, they do not balance the load evenly throughout the network. Common examples of these algorithms include the XY dimension order routing algorithm [8] and the source routing algorithms. In XY dimension order routing, a packet first travels along X-dimension. When the X-coordinate becomes equal to the destination X-coordinate, the packet travels along Y-dimension till the destination is reached. Thus, the distance traveled by the packet from source node to the destination node is equal to the manhattan distance between the two nodes. Figure 4 illustrates the XY dimension order routing algorithm for a packet traveling from source 01 to destination 32 in a 4x4 2D-Mesh network.

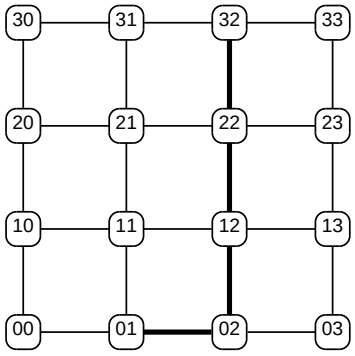

Figure 4. XY Dimension Order Routing for 2D-Mesh

Oblivious Routing algorithms are a superset of the deterministic routing algorithms. In these algorithms there is no fixed path from source to destination; however, the algorithm does not take into account the present state of the network. These al-

gorithms do a better job of balancing the load across the network as compared to the deterministic routing algorithms. A common example of this type of an algorithm is the Valiant et al.'s randomized routing algorithm which uniformly distributes traffic across all the paths in the network [9].

Adaptive Routing algorithms take into account the present state of the network to take a routing decision at each network node. The state of the network can be in the form of the status of a link or a node or the length of the queues in the routers and so on. Adaptive algorithms are more complex and difficult to implement.

G. NoC Flow Control Techniques

Flow control is a mechanism that governs how a packet moves along the network path, through the routing nodes, from its source to the destination. The routing nodes in the network consist of buffers which decouple the allocation of adjacent communication channels. Flow control governs the allocation of these buffers and channel bandwidth. Depending upon the granularity at which these resources are allocated the flow control mechanisms can be classified as being either packet-buffer flow control or flit-buffer flow control. In packet-buffer flow control both buffers and channel bandwidth are allocated in units of packets, whereas in flit-buffer flow control both buffers and channel bandwidth are allocated in units of flits [5], [10]. Different techniques are available to implement each of these flow control mechanisms. some of these techniques are described below.

1. Packet-Buffer Flow Control Techniques

In Store-and-Forward flow control, each routing node waits for a complete packet to be received and stored before forwarding it the next node. In this scheme, a routing

node waits for a packet sized buffer on the far side of the channel and an exclusive use of the channel bandwidth before forwarding the packet. Thus, if any of these resources are not available the packet will not be forwarded. This could result in wasted channel bandwidth. The major disadvantage of this approach is its very high latency as the entire packet has to be buffered before it can be forwarded.

Cut-Through flow control [11] overcomes the limitation of high latencies in store-and-forward flow control by forwarding the packet as soon as the head flit of the packet is received and the resource allocation is successful, without waiting for the entire packet to be received. However, the resource allocation for both buffers as well as channel bandwidth is still done at packet granularity. This approach gives lower latencies and higher channel utilization as compared to store-and-forward flow control.

Packet-buffer flow control techniques have some major disadvantages. These techniques make inefficient utilization of the buffer storage by allocating them at packet granularity. In these techniques, collision of two packets will result in increased latency as one of the packets has to wait until the other packet has been transmitted completely. The limitations of the packet-buffer flow control are overcome by flit-buffer flow control.

2. Flit-Buffer Flow Control Techniques

In Wormhole flow control [12], resource allocation for buffers and channel bandwidth is done at the granularity of flits. Once the head flit arrives at a node, it is immediately forwarded to the next hop when the resources are allocated to it without having to wait for the entire packet to be received at that node. The subsequent body and tail flits are forwarded as and when they arrive and grab the resources. In this technique, the latency at a routing node is just that of a flit and not the whole

packet. A particular packet may span across several routing nodes appearing like a worm. Hence the name wormhole flow control. The downside of this approach is that a stalling packet spanning multiple links will cause all those links to be blocked.

Virtual-Channel flow control [13], [14] overcomes the limitations of wormhole flow control. This technique associates a physical channel with several logical channels called virtual channels (VCs). All the VCs share the same physical channel. In this scheme, the flit has to acquire a VC resource in addition to a flit buffer and channel bandwidth. An arriving head flit is immediately forwarded to the next hop as and when these resources are acquired. The body and tail flits use the same VC allocated to the head flit and must acquire only a downstream flit buffer and channel bandwidth to proceed. The tail flit frees up the VC allocated to the packet. In this approach, if a packet on one of the VC stalls then the same physical link bandwidth can be used by a packet on another VC. This prevents the wastage of physical link bandwidth. Since the VCs are independent of each other, this approach helps to avoid resource dependent deadlocks described by Dally and Seitz [15].

H. Buffer Management Techniques

In flow control mechanisms which use flit buffers (packet in packet-buffer flow control), it is essential for an upstream routing node to know the availability of free flit buffers in the downstream nodes before sending out the flit on the link. This type of signalling between the upstream and downstream nodes is achieved through the means of buffer management techniques. Three flow control mechanisms which perform buffer management and commonly in use today are credit-based, on/off and ack/nack flow control [5].

In Credit-Based flow control, each upstream router keeps a count of the number

of buffers available in each VC fifo (first-in-first-out) in the downstream router. When an upstream router forwards a flit to a downstream router, the credit count for the appropriate downstream VC is decremented. When the downstream router forward the same flit, thus freeing its buffer, a credit is returned to the upstream router over the reverse channel. This causes an increase in the appropriate credit counter in the upstream router. If the credit count in an upstream router becomes zero then all the buffers in the downstream router corresponding to that counter are in use and a flit cannot be forwarded downstream.

In On/Off flow control, the upstream router only maintains a single bit which indicates whether it is allowed to forward data to the downstream router (on) or not (off). Downstream router only indicates changes in this bit to the upstream router. When the free buffer count in the downstream router falls below a specific threshold F_{off}, an off indication is provided to the upstream router. When the free buffer count rises above a threshold F_{on}, an on indication is provided to the upstream router.

With Ack/Nack flow control, the upstream router does not maintain any status of the buffer availability in downstream router. Upstream router forwards flits optimistically as and when they become available. If the downstream router has buffers available and accepts the flit, it sends an acknowledge (ack) indication to the upstream router. If no buffers are available to store the incoming flit, it sends a negative acknowledge (nack) indication to the upstream router. In that case, the upstream router has to retransmit the flit. Thus, flits have to be stored in the upstream router till an ack is received for the flit from the downstream router. This technique may result in flits being received out of order at a downstream node. Hence, the downstream node should be capable of re-ordering the received flits.

The NoC implemented for the AcENoCs emulation framework is a packet switched NoC and uses XY dimension order routing algorithm for 2D-Mesh and 2D-Torus

topologies. AcENoCs' NoC employs virtual-channel flow control with a credit-based buffer management technique.

This chapter briefly describes the concepts required to understand this thesis. With this background, the next chapter will introduce the framework of the AcENoCs FPGA emulation platform.

CHAPTER III

RELATED WORK

The need for fast exploration of the vast design space provided by NoCs in modern communication-centric designs has resulted in the development of several architectural simulators. These can be classified into two categories: software simulators and FPGA based emulators. Most of the software simulators model the NoC at a high level of abstraction and hence are not cycle accurate. On the other hand, FPGA based emulators are fast and maintain cycle accuracy.

A. NoCs for Software Simulators

NoC architecture validation can be performed at different levels of abstraction. The simulation speed decreases as we move towards more detailed simulations at lower levels of abstraction. In order to perform NoC architecture validation at the system level several SystemC based simulators have been proposed. Coppola et al. propose a NoC modeling and simulation framework based on a object oriented C++ library built on top of SystemC [16]. In this work, the authors model the NoC communication using simple transmission and reception application programming interfaces (APIs) called the MasterPort and SlavePort respectively. A message to be transmitted over the network is fragmented into several protocol data units (PDUs). A PDU represents the smallest data unit that can be transmitted over the network. This method uses specialized C++ classes to measure parameters like latency, throughput, packet loss ratio and so on.

Similarly, Kogel et al. propose another modular SystemC based NoC exploration framework to address system level design issues in the communication infrastruc-

ture [17]. Here, data exchange is modeled at a high level of abstraction in terms of packet transfers using transaction-level modeling (TLM) while still capturing the effect on performance. Network engines model traffic characteristics and NoC channel handles the processing of communication. Network statistics like resource utilization, latency and throughput are captured by evaluation modules connected to the network engines. This scheme achieves high simulation speeds but its accuracy is affected by the coarse granularity of packet based communication making it non-cycle accurate.

Goossens et al. present another SystemC simulator in which the NoC is simulated at the flit level and the IP-NI interface is modeled at the transaction level [18]. Here, the SystemC simulation is based on XML files for topology, mapping, NoC configuration and IP configuration. All IPs are modeled as traffic generators using SystemC. This approach is suitable for performance evaluation early in the design cycle when the HDL for IPs is not available. The measurement of throughput, latency and NI buffer statistics is done using the traffic generators itself. This approach is again not cycle accurate.

A common problem with all the system level validation approaches presented above is that they are all non-cycle accurate [16], [17], [18]. Prabhu et al. present a C++ based cycle accurate simulator for NoCs, Ocin_tsim [19]. Here, each component of the network is implemented as a separate C++ class in a modular and object-oriented design style. The traffic generators used in Ocin_tsim are capable of generating a wide variety of synthetic workloads as well as trace driven workloads. Although this simulator is cycle accurate, it is sequential in nature and its simulation speed degrades with increasing network size, making simulation time prohibitive to run real application traffic traces.

B. NoCs for Hardware Emulators

To overcome the limitations of the software simulators several FPGA based NoC emulators were proposed. Genko et al. propose an emulation platform using Xilinx Virtex-II Pro FPGA with an embedded PowerPC processor, emulating a network comprised of six routers [20], [21]. The hardware platform is comprised of traffic generators (TGs), traffic receptors (TRs), control module and the network to be emulated. The network to be emulated is generated using Xpipes compiler. Xpipes architecture uses static source routing and does not support dimension order routing. Xpipes router does not support configurable number of pipeline stages, a desirable feature in routers used for NoC architectural exploration [22]. Software running on the PowerPC processor handles the configuration of the platform and controls the entire emulation process. TG/TRs and controllers consume a fraction of the FPGA's hardware resources and this fraction is expected to grow with the size of the emulated network. Source queues are modeled in hardware and are statically allocated, resulting in inefficient memory utilization for non-uniform traffic.

Peng et al. present another FPGA based NoC emulation framework for a 4x4 2D-Mesh network [23]. In this framework, the authors use an FPGA without an embedded processor to efficiently utilize the available hardware resources. Instead, they use an external instruction set simulator (ISS) running on a host computer to control and configure the emulation system. Communication is established via a USB interface. Similar to Genko et al.'s work, the authors also implement traffic generators, traffic receivers, a controller, an analysis module and a USB controller in hardware, thus occupying additional hardware resources. Since the ISS runs on an external host computer, the communication latencies are high.

Another scheme to emulate large, parallel homogenous and heterogeneous NoCs

is presented by Wolkotte et al. [24]. Here, each router of a large NoC is simulated sequentially using a single router model synthesized on an FPGA. The router model used has 5 ports with 5 virtual channels (VCs) per port. Each VC buffer is 4 flit buffers deep. After the evaluation of each router, the state associated with that router and its links is stored in a memory and is retrieved whenever necessary. As this approach does not exploit the true parallel nature of hardware to speed up simulations, a speedup of only 80-300X compared to a SystemC simulator is achieved.

The concept of polymorphic on-chip networks is presented by Kim et al. in [25]. The polymorphic network is formed by interconnecting a configurable collection of network building blocks in an arbitrary network topology. The configuration of the polymorphic network can be done post-fabrication. This allows the flexibility to tune the network on a per-application basis, thereby improving performance across a range of applications as compared to a network of fixed configuration. The polymorphic networks support customization in terms of network topology, link width and buffer capacity. These features allow polymorphic networks to be used in on-chip interconnection network emulation platforms for architectural design space exploration. But, this flexibility in polymorphic networks comes at the cost of increased area overhead. This approach shows an average of 40% area overhead as compared to fixed networks.

To summarize, most of the system level NoC software simulators are non-cycle accurate or tend to become slow as we move towards lower levels of abstraction and networks of larger dimensions. FPGA emulators overcome limitations of software simulators and are both fast as well as cycle accurate. Most of the previous emulation platforms do not make efficient utilization of the FPGA resources, cannot handle non-uniform traffic or lack flexibility. AcENoCs overcomes the shortcomings of both software simulators and earlier FPGA emulation platforms in terms of speed, cycle accuracy and resource utilization.

CHAPTER IV

AcENoCs ARCHITECTURE

AcENoCs is a cycle accurate and a flexible FPGA accelerated emulation platform for validating on-chip interconnection networks. The AcENoCs emulation platform is instantiated on a Xilinx university program XUPV5 FPGA board. Figure 5 shows the AcENoCs emulation framework. The XUPV5 board houses the Virtex-5 (VLX110T) FPGA. It also contains other peripherals, including an RS-232 serial communication port, DDR2 RAM, System ACE compact flash interface and the JTAG programming interface.

AcENoCs is centered around a hardware/software framework that makes efficient utilization of the available FPGA resources. The FPGA's resources constitute the hardware framework and the MicroBlaze softcore processor with its associated peripherals constitute the software framework . The Microblaze processor is a five stage pipelined processor and operates at a maximum frequency of 125MHz. Some of the AcENOCs software operations are floating point intensive or require a lot of shift operations. These software operations are expensive in terms of the number of processor cycles. Therefore, add-ons like dedicated hardware floating point unit (FPU) and barrel shifter can be instantiated depending upon the performance requirements of the emulator.

An on-chip block RAM (BRAM) stores the program for the software running on the processor. Alternately, external DDR2 RAM can also be used for this purpose, but at the cost of reduced emulation speed due to increased memory access latencies. The hardware and the software framework interact with each other through the processor local bus (PLB). A brief explanation of each of the components of AcENoCs is also

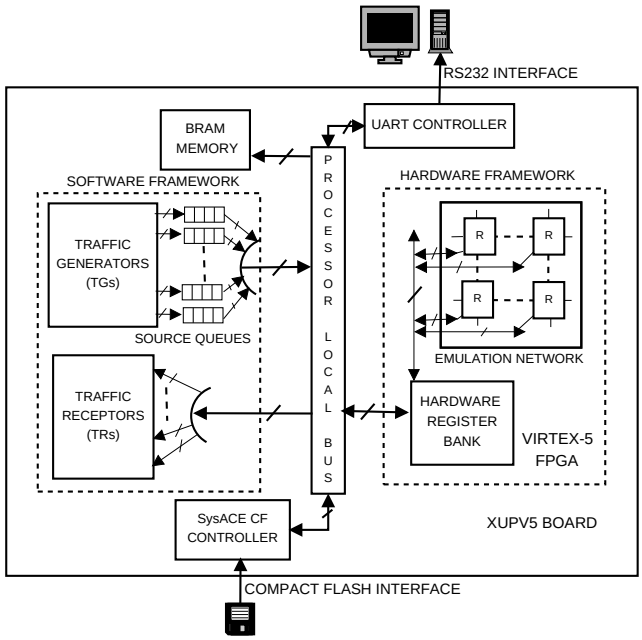

Figure 5. AcENoCs Emulation Framework

provided below.

A. Software Framework

AcENoCs software framework comprises of the traffic generators, traffic receptors, source queues and the clock generator.

Traffic Generators (TGs) are responsible for generating the packets to be injected into the network. There is one TG associated with each routing node in the emulation network. The generated packets are queued up in the source queues before getting injected into the network through a router's local port. The TGs are capable of generating both synthetic as well as trace driven workloads. The synthetic workloads supported include Uniform Random, Bit Complement, Bit Reversal, Matrix Transpose, Bit Shuffle, Bit Rotation and Hotspot [5], [26]. For trace driven workloads, the traces are read from an external SystemACE compact flash card. Each entry stored

on flash card is a packet descriptor and contains packet injection time, packet source address and the packet destination address. The TGs can be configured using an emulation configuration file.

Source Queues act as a temporary packet storage for each routing node and help to decouple the packet generation process from the state of the network. Software implementation allows dynamic memory allocation to the source queues, thus enabling them to efficiently handle non-uniform traffic conditions.

Traffic Receptor (TR) is associated with every routing node in the network. The TRs decode the information present in the packets received at the destination routing node and validate their destination. The TRs also play a role in the latency calculation of each received packet and report the average latency and throughput at the end of the emulation process.

Clock Generator is responsible for generating the clock to the emulated network called emulation clock. Each emulation clock cycle is comprised of several processor clock cycles, the number of actual processor cycles being dependent on the software execution latency. The provision of a software generated clock helps to achieve proper synchronization between software and the hardware events scheduled to occur in a particular emulation cycle. This kind of dynamic emulation clock control is much more efficient than static emulation clock control as it allows the hardware to run much faster depending on the software latency per emulation cycle. The period of the emulation cycle varies with the packet injection rate and also with the size of the network being emulated.

The ability to control emulation parameters like packet injection rate, packet sizes, traffic patterns, number of packets etc. through software adds a great amount of flexibility to the AcENoCs emulation platform. These parameters can be easily changed to explore several emulation configurations in a very short span of time.

B. Hardware Framework

The hardware framework of AcENoCs emulation platform is the on-chip interconnection network to be emulated. The NoC is constructed using a custom built and highly parameterized hardware library consisting of network routers and links. The routers can be interconnected using either a 2D-Mesh or a 2D-Torus topology. A routing scheme decides how a packet traverses the network to reach the intended destination. A flow control scheme decides the allocation of the network resources to the packet's flits as they traverse the network. The use of VC based flow control instantiates multiple buffers per physical channel instead of just one buffer. This helps to improve network throughput and avoid deadlock conditions. The buffers are implemented using the distributed RAM resources of the FPGA. A detailed design and implementation of the hardware library components for AcENoCs is provided in Chapter V.

C. Hardware-Software Interface

The interfacing between the hardware and software components on the AcENoCs embedded emulation platform happens through the PLB bus. AcENoCs has a register based interface for providing communication between the on-chip emulation network and the software running on the MicroBlaze processor. The registers are classified into five categories :

1. Clock Register : This register is used to provide software controlled clock to the emulated network.

2. Input Data and Data Valid Registers : These registers are used to inject flits generated by the TGs into the local data ports of the emulated network.

3. Output Data Registers : These registers are used to capture the flits ejected by the emulation network and transfer them to the software for analysis.

4. Input Status Registers : These registers maintain status information regarding the occupancy of the input VC fifos.

5. Output Status Registers : These registers contain indications of the output flit availability at a particular router.

All of these registers are connected to PLB and are software accessible.

D. Memory

On-Chip BRAM memory is used to store the programs running on the MicroBlaze processor to implement the functionality of the software framework. The BRAM also holds the data structures used for bookkeeping purposes. An external DDR2 RAM can also be used as an alternative to BRAM, but the memory access latency of DDR2 RAM is much higher than that of on-chip BRAM and hence impacts AcENoCs' performance.

E. Interfaces

The different interfaces present on the AcENoCs emulation platform are briefly explained below.

UART Serial Communication Interface on the XUPV5 FPGA board is used to display emulation configuration, debug messages during emulation process and statistics display at the end of emulation process.

SystemACE Compact Flash Interface on the XUPV5 board is used to read the trace driven workload data from an external compact flash memory. A maximum of

4-GB of trace data can be read from the external flash card.

JTAG Programming Interface is used to download the configuration file generated by the Xilinx toolset to the FPGA.

This chapter provides an overview of the AcENoCs emulation platform. The NoC hardware library components described in the next chapter are instantiated in this AcENoCs platform.

CHAPTER V

DESIGN AND IMPLEMENTATION OF AcENoCs HARDWARE FRAMEWORK

The AcENoCs hardware framework consists of NoC routers connected together using links. The routers can be connected using either a mesh or a torus topology. This chapter describes the design and implementation details of the NoC router, the network and the network wrapper used in the AcENoCs emulation platform. Various optimizations involved in the design of these components are also discussed together with the configurability added by each of them.

A. NoC Router Design

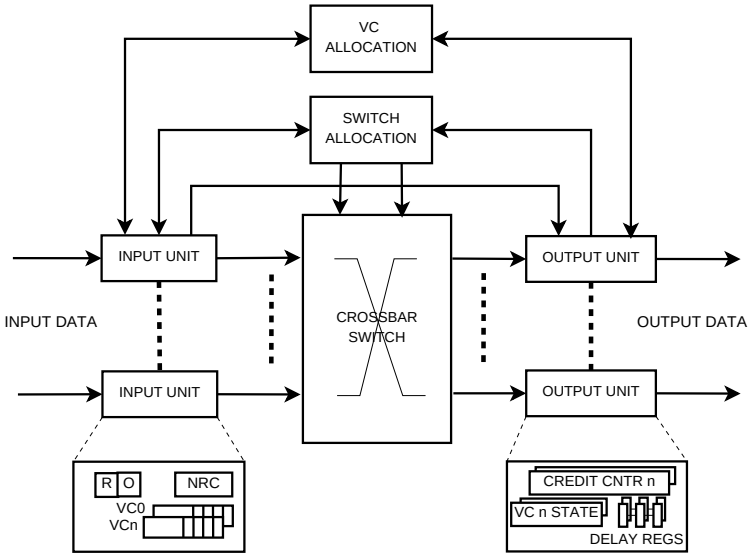

Figure 6. Router Architecture

A NoC router is the major building block of the AcENoCs hardware framework. The NoC router used in AcENoCs is comprised of an input unit, routing unit, virtual

channel (VC) allocation unit, switch allocation unit, crossbar unit and an output unit. Figure 6 illustrates the block diagram of the AcENoCs NoC router. The number of pipeline stages in the router is configurable and can vary from single stage pipeline to a five stage pipeline. The local port of each of the routers is used to connect to the software TGs and TRs in the emulation platform. A design of the various router components is discussed below.

1. Input Unit

The input unit receives flits from the neighboring routers in the interconnection network. The structure of the received flits is shown in Figure 7. The default flit width is 32-bits, the minimum required to carry the routing information present in the header flit.

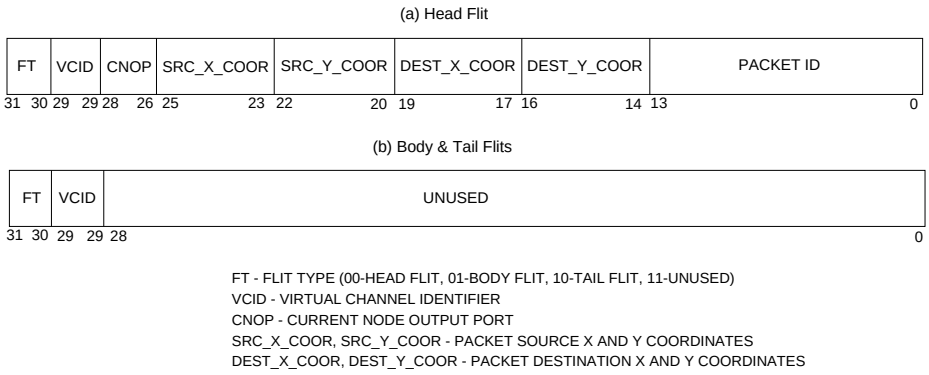

Figure 7. Flit Structure

One input unit is instantiated per input physical port present on the router. As shown in Figure 8, the input unit instantiates the VC fifo buffers (VC0-VCn) to buffer the received flits until they can be forwarded to the downstream router. The virtual channel identifier (VCID) present in the received flits is used to de-multiplex the flits

to the appropriate VC fifo. The number of VCs, the depth and width of each VC fifo can be configured as a parameter. Data is read out from the VC fifos as and when the fifo has data and the switch allocator and VC allocator are ready to accept new requests.

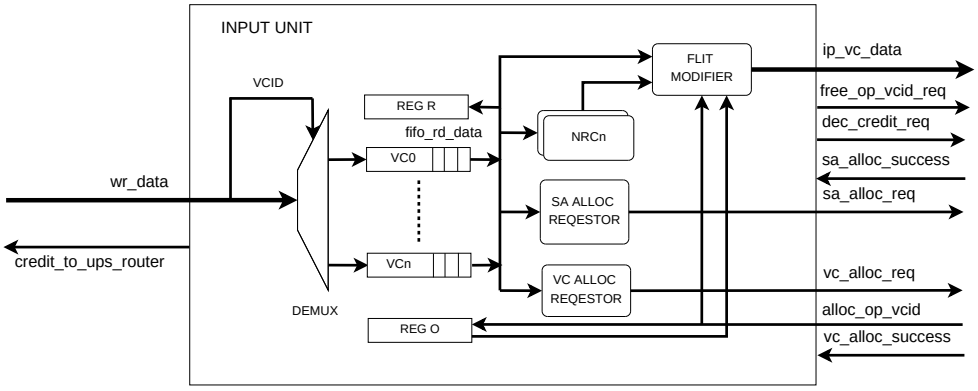

Figure 8. Input Unit Block Diagram

The output port for any packet at the current router is extracted from the current node output port (CNOP) field contained in the head flit of that packet. The extracted output port is stored in register "R" till all the flits of that packets have been transmitted by the current router. One register is present for each input VC instantiated in the input unit. The input unit also comprises of a next hop routing computation unit (NRC). This unit implements the look ahead routing technique [27]. The NRC unit at each router computes the output port for a packet at the downstream router. The output port is computed based on the coordinates of the downstream router and source/destination coordinates of the packet using a deadlock-free, XY dimension order deterministic routing algorithm. The coordinates of the downstream router can be computed by knowing the coordinates of the current router and the output port of the packet on the current router. In the XY dimension order routing

algorithm, a packet first travels along the X-direction and then along the Y-direction. The complete algorithm is provided in Algorithm 1 for 2D-Mesh topology . The computed output port is inserted in the outgoing header flit of the packet for use by the downstream router. The look ahead routing scheme enables the route computation to be done in parallel with switch and VC allocation. Alternate routing algorithms can be easily implemented by modifying the functionality implemented by NRC.

Algorithm 1 XY-Dimension Order Routing for 2D-Mesh

Input Coordinates of the downstream node(Xdownstream,Ydownstream)
Coordinates of the destination node(Xdest,Ydest)

Output Output port for downstream node(Output Port)

Procedure

if $(Xdest > Xdownstream)$ **then**
$OutputPort = PORT\_EAST$
else if $(Xdest < Xdownstream)$ **then**
$OutputPort = PORT\_WEST$
else if $(Ydest > Ydownstream)$ **then**
$OutputPort = PORT\_NORTH$
else if $(Ydest < Ydownstream)$ **then**
$OutputPort = PORT\_SOUTH$
else
$OutputPort = PORT\_LOCAL$
end if

The input unit generates requests to the VC allocator and the switch allocator to gain access to an output VC and the switching resource respectively. The output

VC assignment done by the VC allocator is stored in the register "O" instantiated per input VC. The value contained in this register will be used for directing the body and tail flits of the packet on the same output VC as the head flit or by the head flit itself in case of switch allocation failure. The VCID corresponding to the output VC assignment is inserted in the VCID field of each outgoing flit. The interface between the input unit and the VC and switch allocators is a request-acknowledge interface. A successful VC and switch allocation causes the flit to be transmitted on the link after traversing the switching resource.

Input unit plays an active role in the implementation of credit based flow control [5]. The input unit generates a request to the output unit to decrement the appropriate credit counter for every flit transmitted downstream on a particular output VC. It also sends a credit upstream, indicating that a flit buffer is freed in one of its VC fifo buffers. The input unit generates a request to free an output VC when the tail flit of a packet occupying an output VC is transmitted to the downstream router.

2. Virtual Channel (VC) Allocation Unit

The VC allocation unit allocates an output VC to a packet from amongst the several VCs available on the output port indicated by the routing unit. The VC allocation unit is triggered only for head flits and not for body and tail flits. Only those output VCs which are in IDLE state can be allocated to a packet by the VC allocator. The VC allocation is implemented in the form of two level arbitration. The first level of arbitration consists of one arbiter instance per input VC. This level of arbitration selects one output VC on an output port from among all the free output VCs on that particular output port. Thus the outcome of the first level of arbitration is a request to a single output VC from each of the input VCs on each input port. The second

Table II. Truth Table for Fixed Priority Arbiter

INPUTS				**OUTPUTS**			
R0	R1	R2	R3	G0	G1	G2	G3
1	X	X	X	1	0	0	0
0	1	X	X	0	1	0	0
0	0	1	X	0	0	1	0
0	0	0	1	0	0	0	1

level of arbitration consists of one arbiter instance per output VC. The winner of the first level of arbitration is routed to the appropriate second level arbiter. This arbiter selects one winner for an output VC from among the several requests from each of the input VCs. This kind of a virtual channel allocation scheme has been described by Mullins et al. [28].

The VC allocation implements both fixed as well as round-robin priority arbiters. The selection of arbitration scheme is done through a configuration. In a fixed priority arbiter, priority is assigned in a linear fashion to each of the request lines of the arbiter. Thus, if multiple request lines are asserted simultaneously at the input of a particular arbiter then the one with the highest priority will be serviced first. The fixed priority arbiter can be implemented using a priority encoder. The behavior of a four input fixed priority arbiter is indicated in Table II. R0, R1, R2, R3 represent the request lines and G0, G1, G2, G3 are the corresponding grant lines. The disadvantage with this scheme is that the lower priority requestors will starve if there are burst requests from a high priority requestor. The fixed priority scheme is not a fair arbitration scheme. Round-robin arbitration scheme overcomes this limitation of fixed priority arbitration. Round-robin arbitration operates on the principle that a request that

was just serviced will be given lower priority on the next round of arbitration [5]. All the pending requests will be serviced before the same request line gets selected again. Thus the round-robin arbiter can be considered as being similar to fixed priority but, with the priority rotating after each selection. The round-robin arbitration scheme exhibits strong fairness.

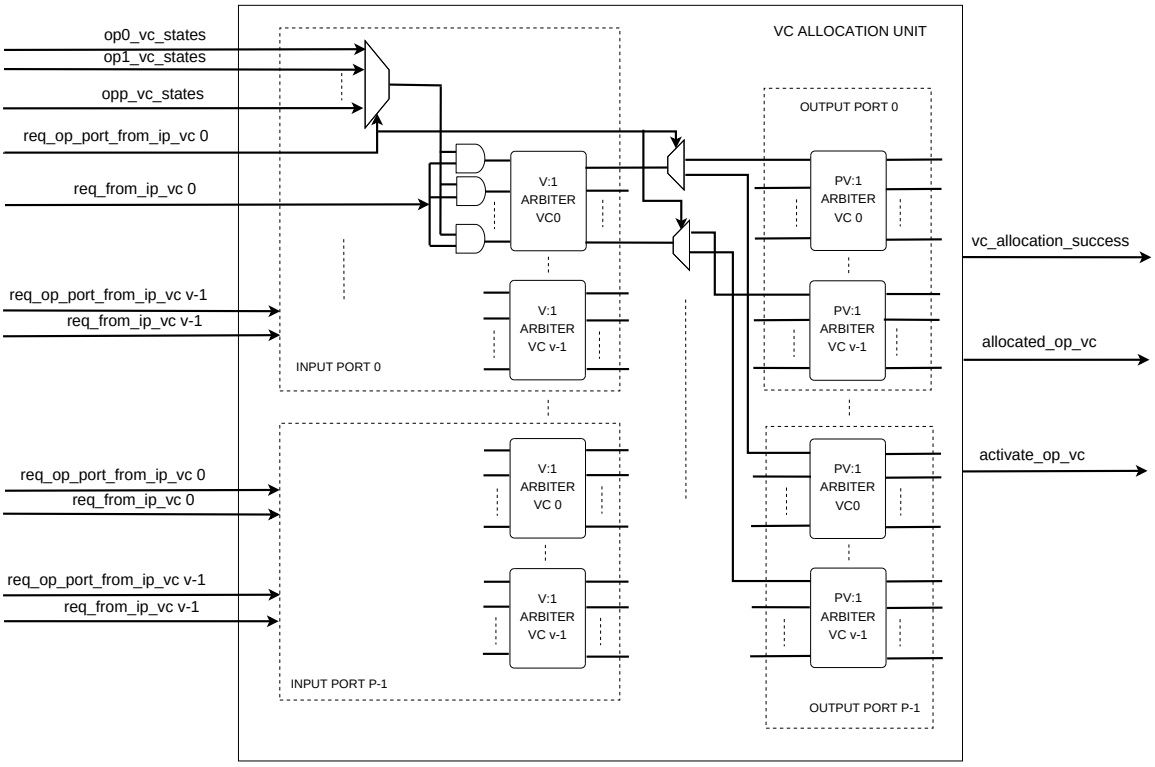

Figure 9. Virtual Channel Allocator Block Diagram

Figure 9 shows the implementation of VC allocator using two level of arbitration. The figure shows that, if there are P physical ports and V virtual channels per physical port then, V arbiters of the type V:1 will be instantiated per input physical port. There will be a total of PxV first level arbiters. One PV:1 arbiter will be instantiated per output virtual channel. There will be a total of PxV second level arbiters. Thus,

the complexity of VC allocator is very high and is strongly dependent on the number of physical ports on a router and the number of VCs per physical port. The key to lowering the complexity of the VC allocator is to reduce the number of physical ports on a router or split the high complexity arbiters into smaller low complexity arbiters.

3. Switch Allocation Unit

The switch allocation unit controls the access to the switching resource of the router i.e. the crossbar unit. Like the VC allocator, this can also be implemented using two levels of arbitration [28]. The first level of arbitration consists of one arbiter instance per input physical port. This arbiter decides which input VC on a physical port gains access to the crossbar. This reflects the sharing of a single crossbar physical port by all the input VCs at a particular input physical port. The second level of arbitration consists of one arbiter per output port. These arbiters decide which input port gains access to a particular output physical port. The switch allocator provides control signals to the crossbar unit to enable the appropriate data path in the crossbar. The switch allocator grants access to the crossbar if and only if credits are available on the downstream VC which has been allocated to the packet requesting the crossbar resource. The switch allocation is done in the same emulation cycle as the VC allocation, thus reducing the router pipeline latency. Unlike VC allocation, switch allocation is done for all the flits of a packet.

Like the VC allocator, the switch allocator also implements both round-robin and fixed priority based arbitration schemes, the selection of the arbitration scheme being done through a configuration. Figure 10 shows the implementation of the switch allocator using two levels of arbitration. The figure shows that if there are P physical ports and V VCs per physical port, then the first level of arbitration will have P arbiters of the type V:1. The second level of arbitration will have P arbiters of the

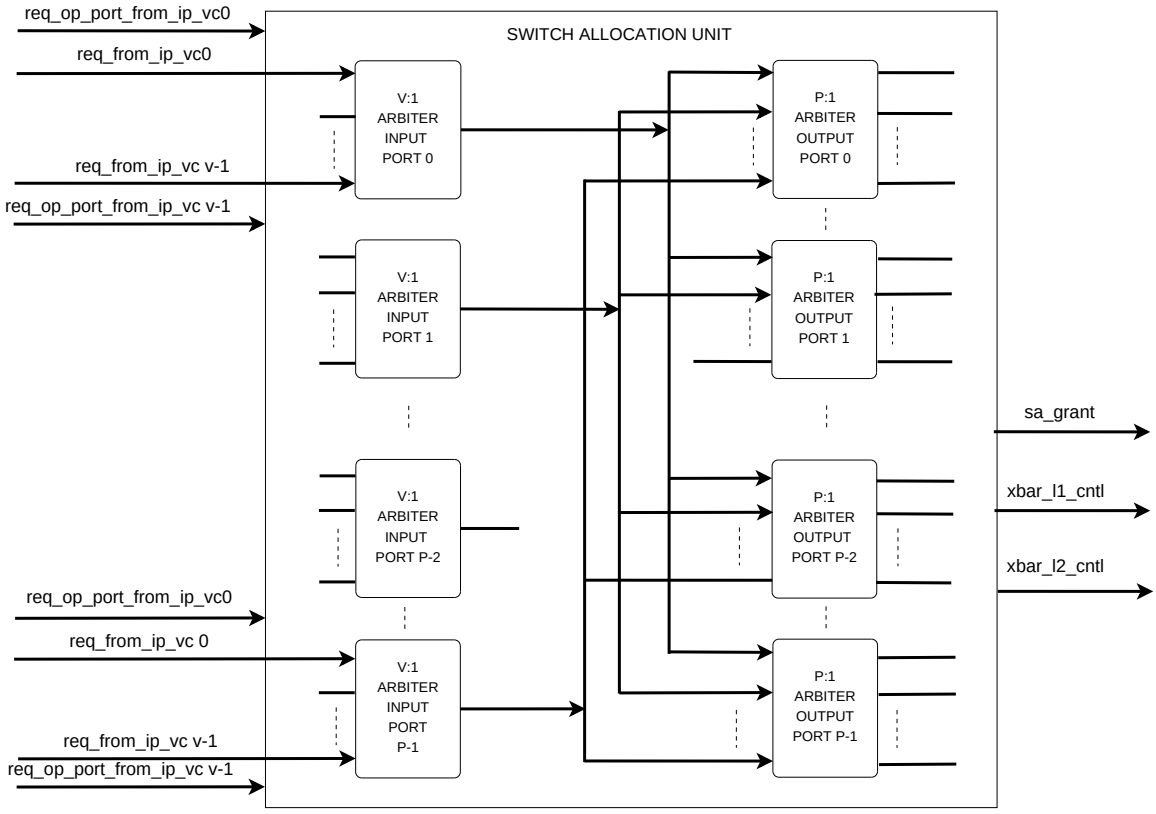

Figure 10. Switch Allocator Block Diagram

type P:1. The complexity of the switch allocator is much lower than that of a VC allocator.

4. Crossbar Unit

The crossbar unit is the switching resource of the router. The crossbar is fully connected and provides a data path from each input physical port to every output physical port. If a router has I input physical ports and O output physical ports then, the crossbar unit will have IxO physical ports. It is designed using two levels of multiplexers controlled by the switch allocation unit. The control signals to the first level of multiplexers are provided by the first level of arbiters of the switch allocator.

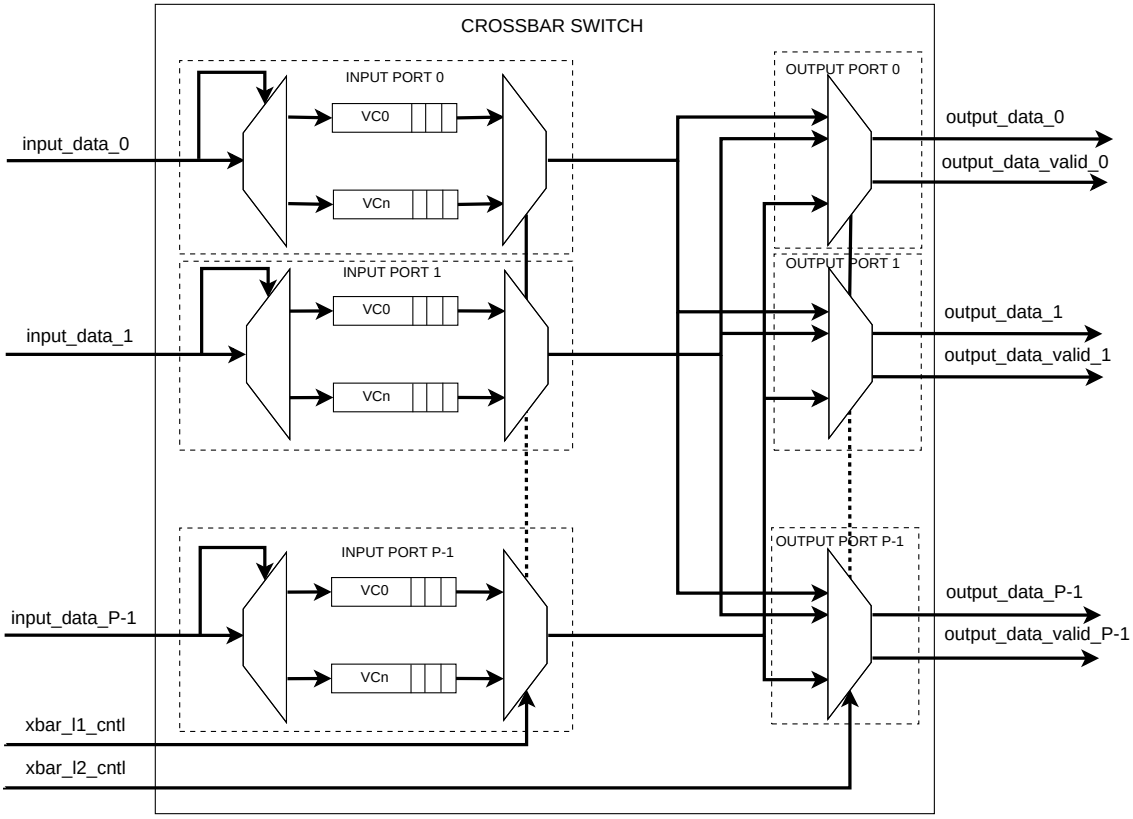

Figure 11. Crossbar Switch Block Diagram

Similarly, the control signals to the second level of multiplexers of the crossbar unit are provided by the second level of arbiters of the switch allocation unit. The input data to the crossbar unit comes directly from the VC fifos instantiated in the input unit. The crossbar unit supports multiple simultaneous active connections from its input to the output, thus increasing the throughput of the router. The crossbar unit drives its output data to the programmable delay unit along with the appropriate valid bits to qualify the data. An implementation of the crossbar unit using the two level multiplexer scheme is shown in Figure 11.

5. Output Unit

The output unit maintains the status of output VCs as either idle or active. The output VC is activated when it is allocated to a packet by the VC allocator. It is freed when the tail flit corresponding to that packet departs the router under consideration and a VC fifo buffer is allocated to it in the downstream router i.e. on the success of switch allocation to the tail flit of that packet. The output unit also keeps track of the credits available in the downstream routers VC fifo. The credits are decremented when a buffer is allocated to a particular flit in the downstream router and they are incremented when the same flit departs the downstream router. The departure of the flit from the downstream router is indicated to the upstream router by returning a credit indication over the reverse link. This is implemented using credit based flow control.

6. Programmable Delay Unit

This unit consists of a series of programmable delay registers which delay the output data and data valid from appearing on the link and is a unique feature of the AcENoCs hardware framework. These delay units can be used to emulate either a router pipeline stage or link delay. Thus, by varying the programmable delay, routers of different pipeline depth or even pipelined links can be emulated with ease.

B. Interconnection Network Design

The interconnection network is constructed by connecting NoC routers together in a particular topology. Two topologies supported by AcENoCs emulation framework are 2D-Mesh and 2D-Torus topologies. These topologies are explained in Chapter II. AcENoCs instantiates links to connect routers together for data flit traversal accord-

ing to the specifications of the network topology being emulated. The width of the link can be configured as a parameter. By default, link traversal is completed in a single cycle unless additional link delay is introduced using the programmable delay registers present in the NoC router. In addition to the links carrying the data flits, there are dedicated links available to implement credit based flow control for transporting credits from a downstream router to an upstream router.

A network wrapper instantiates this interconnection network together with a series of registers required to interface with the software framework of AcENoCs. These registers are connected to the PLB bus and can be accessed by the software running on the MicroBlaze processor. As discussed in the earlier chapters, five different categories of registers are defined namely, clock register, input data and data valid registers, output data registers, input status registers and output status registers. Clock register drives the emulation clock to the network. Input data and the data valid registers are connected to the local port of each of the routers in the network. These are used to drive the input flits into the network. The output data registers are also connected to the local port of each of the routers in the network. They capture the flits ejected from the network. Input status registers hold the fifo occupancy status of the input VC fifos on the local ports of each of the router in the network. These are checked by the software framework before driving data onto the local ports to prevent fifo overflow. Output status registers are used to indicate output flit availability on the local port of the routers in the network. These register values are used by the software as the qualifying signals for the the data contained in the output data registers. The network wrapper is combined with the software framework to form the complete AcENoCs emulation platform described in Chapter IV.

Table III provides a summary of the configurable features available in the AcENoCs hardware framework.

Table III. Summary of Configurable Features in AcENoCs Hardware Framework

FEATURE	**RANGE**
Router	
Number of router physical ports	> 1
Data width	≥ 32 bits
Number of virtual channels fifos	≥ 1
Virtual channel fifo depth	≥ 1 flit buffer
Virtual channel fifo width	≥ 32 bits
VC allocator arbitration scheme	Fixed-Priority, Round-Robin
Switch allocator arbitration scheme	Fixed-Priority, Round-Robin
Number of router pipeline stages	1-5
Links	
Link width	≥ 32 bits
Link delay	≥ 1

C. Discussion

Typical NoC routers are connected together by links of a given width. These links carry data flits from one router to another and their width directly affects the number of flits required to transmit a complete packet. In AcENoCs, the default link width of the instantiated routers is 32 bits, the minimum required to carry routing information in the header flit; however, different link widths can be easily emulated by changing the number of flits transmitted per packet. The number of flits per packet is given by :

$$\text{Number of Flits Per Packet} = \lceil \frac{Packet\ Size}{Flit\ Width} \rceil$$

Two options are available for implementing the VC fifos viz. dual port on-chip block RAM (BRAM) memory or the dual port distributed RAM memory using FPGA's lookup tables (LUTs). The Block RAM is available only in fixed block sizes and is typically used for storing large amount of data. The number of BRAM blocks available are not sufficient for implementing all the fifos of a 2D-Mesh network with dimensions greater than 3x3. The distributed RAM, on the other hand, is used for implementing smaller sized memories. Since the memory requirements for VC fifos is small, distributed RAM is used.

This chapter describes in detail the implementation of the NoC router, the network and its integration in the AcENoCs emulation framework. The next chapter provides details on the FPGA implementation flow for this hardware framework and the flow for NoC design space exploration using AcENoCs.

CHAPTER VI

AcENoCs FPGA IMPLEMENTATION AND EMULATION FLOW

A. AcENoCs FPGA Implementation Tool Flow

In this section, we present, step-by-step, the processes involved in the integration of the AcENoCs hardware and software components into a bitstream which can be downloaded onto an FPGA to form a robust and complete Hardware-Software NoC emulation framework. The inputs to each process and the outputs of each process in the tool flow are also discussed.

AcENoCs emulation framework is realized on an XUPV5 board containing a Virtex-5 FPGA. The FPGA implementation is done using the Xilinx Embedded Development Kit (EDK) tool chain. EDK facilitates the development of a complete standalone embedded processor system using Xilinx MicroBlaze soft core processor IP. The AcENoCs hardware and software library components are integrated into the embedded system using EDK. The entire FPGA implementation flow is indicated in the Figure 12. This has been adapted from the Xilinx embedded systems tools reference guide [29].

As a part of the system building process, all the necessary IP cores are provided as input to the tool in EDK format. At this point, we also specify the number of microblaze processors to be instantiated in the system and amount of on-chip BRAM memory to be allocated to each processor and the associated peripherals. Several user-defined registers are also instantiated for interaction between the hardware and software components. The interconnection of the various system components using the PLB system bus is also specified as a part of this process. This entire system building process is accomplished using the Xilinx Base System Builder (BSB) provided

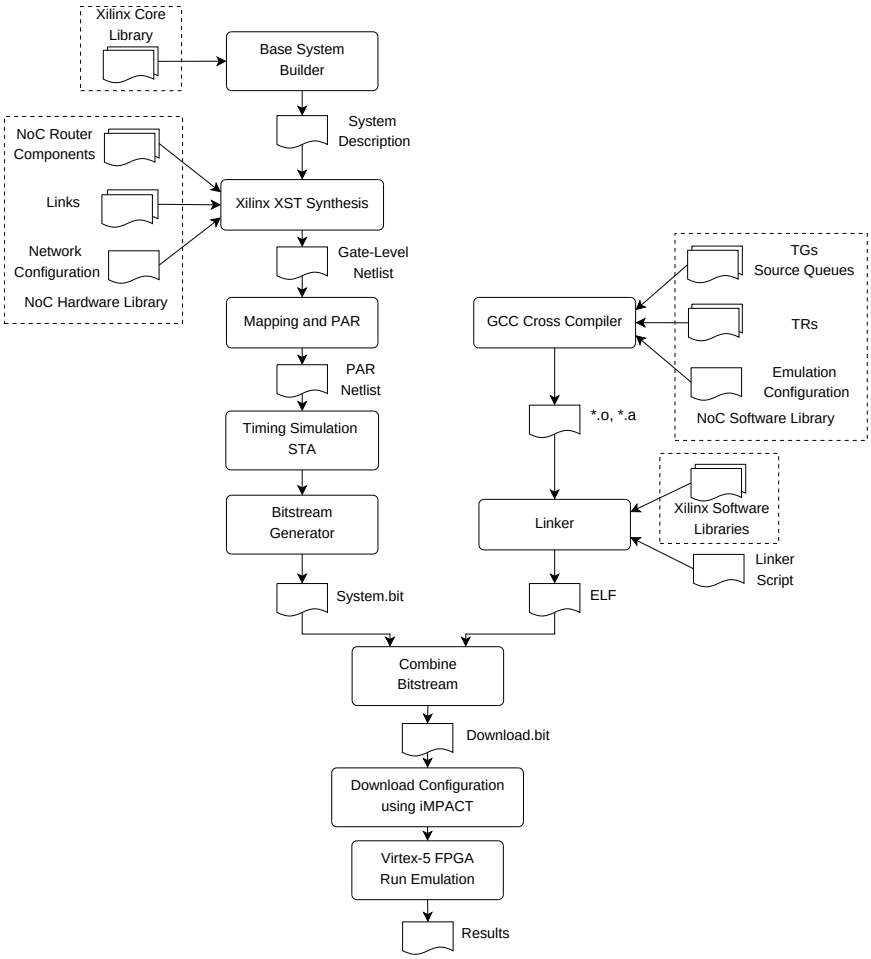

Figure 12. FPGA Implementation Flow

with the EDK tool chain.

Once the base system is built, the next step is the inclusion of NoC hardware library components in the base system. The NoC hardware library consists of NoC routers and its components, links and the network configuration file. The NoC hardware library components are described using verilog HDL and can be imported directly as one of the peripherals connected in the system, the interconnection of NoC components being specified through the network configuration file. The network configuration also enables the user to configure other network parameters like network

dimensions, number of VCs, depth of the VC fifos, router pipeline depth, arbitration schemes, flit width etc. At this point we have completely specified the hardware system and the interconnection among its components.

Once the NoC hardware library components are integrated into the base system, the entire system is synthesized using Xilinx XST synthesis tool. The synthesis results in the generation of gate level netlist of each system components in the form of NGC files. The generated netlist are then mapped to the FPGA technology and subjected to a place and route (PAR) process. The PAR process generates a placed and routed netlist which describes the placement of various components of the FPGA and the routing of the wires between these components. The PAR netlist is then subjected to a static timing analysis (STA) to verify that the netlist meets all the timing constraints and does not have any timing violations. A bitstream generator is then invoked to generate a BIT file which is the downloadable image of the hardware framework for the FPGA. The total time required for each of these steps is directly proportional to the size of the NoC being emulated.

The next step is to combine the NoC software library components along with the hardware image to generate a combined executable for the entire embedded system. The software library components consist of TGs along with the associated source queues, TRs and the user defined emulation configuration file. The emulation configuration file enables configuration of parameters such as flit injection rate, traffic pattern selection, packet size and the number of packets to be injected, random number generation scheme, size of the software source queues, etc. A linker script describing the mapping of the software program onto the code and data sections of the processor memory is generated. In addition to the code and data sections, the linker script also describes the amount of memory allocated to the statically allocated stack and dynamically allocated heap structures. The software is compiled and linked

with the Xilinx software libraries using the GCC cross-compiler provided by the EDK tool chain. This results in the creation of an ELF software image file. In the final step, this ELF file is merged with the BIT file produced in the previous step to form a final FPGA image download.bit. This BIT file is downloaded to the FPGA using the Xilinx iMPACT tool and the JTAG interface on the FPGA board. With this the AcENoCs platform is ready for the emulation process.

B. AcENoCs Emulation Flow

A user configures the AcENoCs emulator using two configuration files, the emulation configuration file for the software framework and the network configuration file for the hardware framework. The complete emulation process is governed by the parameters specified in the emulation configuration file. These two files are provided as inputs to the implementation flow which synthesizes the hardware framework, compiles and links the software framework and downloads the bitstream to the FPGA. The emulation process is triggered by the downloading of the BIT image onto the FPGA.

During each emulation cycle, the TGs generate a new packet based on the specified injection rate and the traffic pattern and inject it into the source queues. Depending upon the status of the VC fifos on the local port, a flit may also be injected from the source queues into the network. If a flit is ejected from the network, the TRs receive the flit, validate the data and update the bookkeeping structures based on the type of the flit received.

The emulation stops once the ejected packet count becomes equal to the injected packet count or an error is encountered. At the end of the emulation, statistics like number of emulation cycles, latency and throughput are displayed using the UART

serial communication interface. For NoC design space exploration, various parameters described in emulation configuration file and network configuration file could be varied and the entire emulation process repeated until the desired results are obtained. A change in only the emulation parameters requires a software recompilation, re-synthesis of the hardware framework is not necessary. This allows fast exploration of the vast design space offered by NoCs.

The next chapter describes the results of validating and evaluating the designed hardware framework and the complete emulation platform using the FPGA implementation and emulation flows.

CHAPTER VII

VALIDATION AND EVALUATION

In this chapter, we evaluate the performance of the AcENoCs emulator and validate its hardware framework by comparing the results to other standard network simulators. We present the results of network validation, AcENoCs performance evaluation and hardware evaluation under varying workloads, network sizes and packet sizes.

A. Evaluation Methodology

The AcENoCs baseline network configuration consists of a 5x5 2D-Mesh network with XY-dimension order routing, VC flow control with two VCs and eight flit buffers per VC. The NoC router is single stage pipelined. Except where otherwise noted, the AcENoCs baseline emulation network configuration was used for all results. Baseline results were obtained for the bit-complement traffic pattern for one million packets. Packets were configured to be 5 flits in size with flit size being fixed to 32 bits. The AcENoCs emulation flow described in Chapter VI was used to vary the emulation parameters. The interconnection network and the emulator performance are also compared against the Ocin_tsim software simulator [19] and the ABC network's verilog HDL simulator [30]. Any deviation from the baseline configuration has been described appropriately.

B. Network Validation

This section provides details on validation tests carried out for evaluating the designed 2D-Mesh NoC using AcENoCs. Two of the most commonly used metrics in NoC validation are network latency and throughput under a given traffic pattern and

injection rate. Latency is the number of cycles required for a packet to travel from a source node to a destination node and can vary across traffic patterns and injection rates. Throughput is the rate at which packets are ejected from the network. These metrics were measured for several traffic patterns, including bit-complement, bit-reversal, matrix transpose, shuffle, uniform random and were compared against the results obtained using other network simulators to validate their accuracy [19].

Figure 13 shows the variation of throughput with flit injection rate for some of the test workloads using AcENoCs and Ocin_tsim. Flit injection rate is the rate at which flits are injected into the network. Both, flit injection rate and throughput are shown as a percentage of the peak injection rate. AcENoCs results show that the throughput follows the injection rate very closely till 45% and 55% for bit-complement and uniform random traffic patterns respectively. Thereafter, the throughput levels off with further increase in injection rate due to network saturation. The leveling of the throughput graph occurs at a lower injection rate in Ocin_tsim as compared to AcENoCs. This is due to the the variation in the throttling characteristics of the two simulators. The total number of outstanding flits that could be present in the system at any instant of time is much lower in Ocin_tsim as compared to AcENoCs. Hence, the throttling of packets starts much earlier in Ocin_tsim than in AcENoCs. Thus, AcENoCs results are more accurate as compared to Ocin_tsim.

In the Figure 14, the variation of latency with flit injection rate is shown for bit-complement and uniform random traffic patterns for the same 5x5 2D-Mesh network. The graph indicates that there is only a small increase in latency until a particular injection rate after which it increases drastically even for minor changes in injection rate indicating network saturation. The drastic change in latency for bit-complement and uniform random traffic occurs around 40% and 50% respectively using AcENoCs. The differences in the latency values between AcENoCs and Ocin_tsim at lower injection

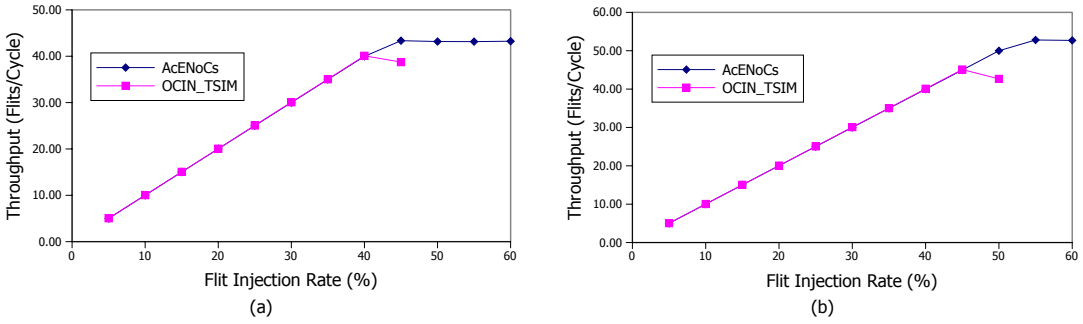

Figure 13. Throughput vs. Flit Injection Rate for (a) Bit-Complement and (b) Uniform Random Traffic

rates can be attributed to the slight differences in the packet generation (due to the random nature of the packet generation) and bookkeeping processes, and the router implementation for the two simulators. On the other hand, at higher injection rates, the latency values differ primarily due to the variation in throttling characteristics of the two simulators.

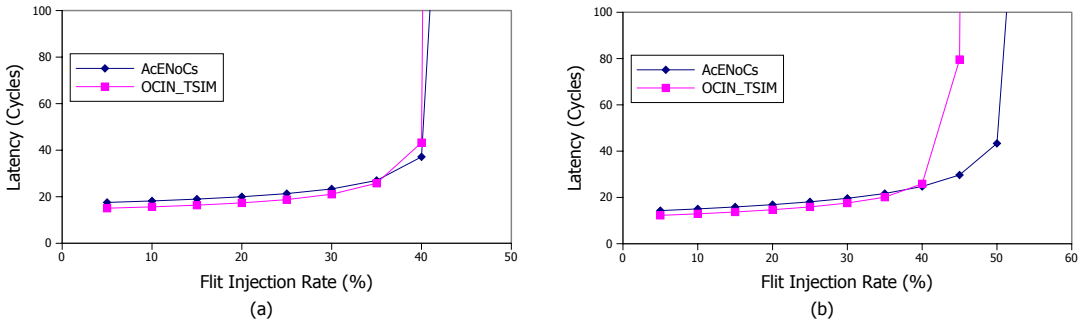

Figure 14. Average Latency vs. Flit Injection Rate for (a) Bit-Complement and (b) Uniform Random Traffic

To further validate the accuracy of the AcENoCs hardware framework, variation of latency with flit injection rate for varying network sizes and packet sizes was studied and the results obtained were compared against Ocin_tsim. As seen from Figure 15

both, AcENoCs and Ocin_tsim show an increase in the average packet communication latency with increase in network dimensions. Scaling of the network dimensions results in addition of nodes to the network. Some of the packets now have to travel over longer distances to reach the intended destination. This causes an increase in the average packet communication latency.

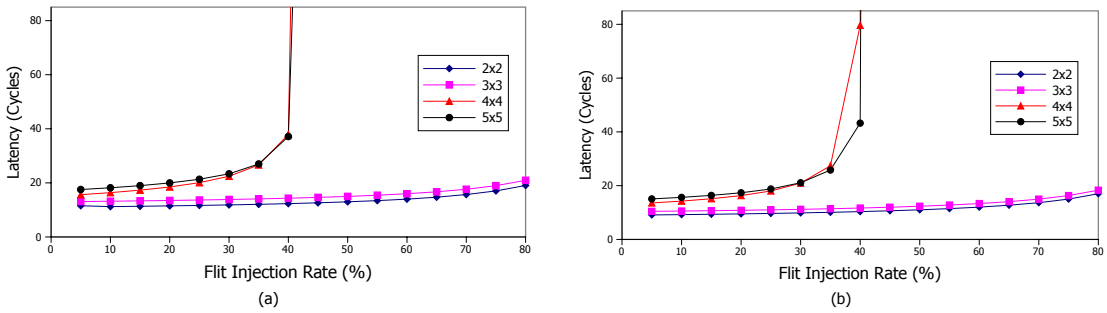

Figure 15. Average Latency vs. Flit Injection Rate across 2x2, 3x3, 4x4 and 5x5 2D-Mesh Networks for (a) AcENoCs and (b) Ocin_tsim

Both, AcENoCs and Ocin_tsim show an increase in the latency with increasing packet size. This is illustrated in Figure 16. As the packet size is increased, more cycles are required for the packet to traverse the network and be received at the destination. Also, larger packets cause higher congestion in the network as compared to smaller packets, for the same number of packets. Hence, network saturates at a lower injection rate for larger packets as compared to smaller packets.

All the above results show that both, AceNoCs and Ocin_tsim exhibit similar behavior for varying traffic patterns, network sizes and packet sizes. All the curves shown in this section are typical of 2D-Mesh networks.

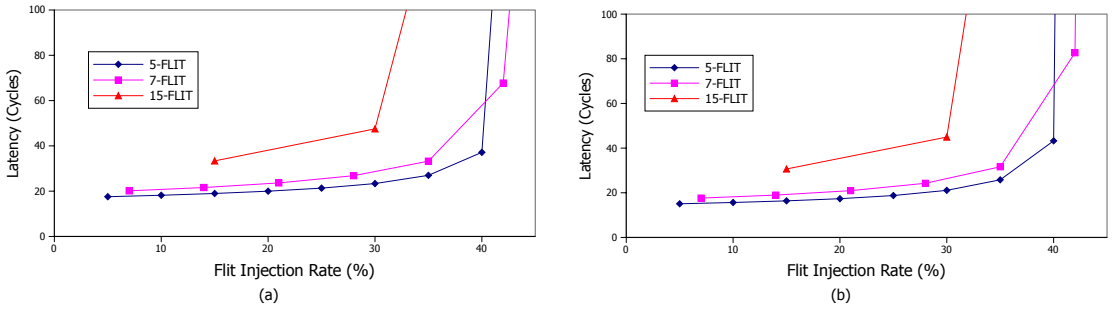

Figure 16. Average Latency vs. Flit Injection Rate for 5x5 2D-Mesh Networks under Varying Packet Sizes for (a) AcENoCs and (b) Ocin_tsim

C. Emulator Performance Evaluation

This section presents the results of AcENoCs emulator performance evaluation under varying packet and network sizes. Emulation cycles achieved per second was used as a measure to evaluate emulator performance. This parameter was calculated by measuring the amount of time in seconds it takes to run a fixed number of emulation cycles. The variation of the emulator performance with flit injection rate under varying network sizes is shown in the Figure 17. We observe that the emulator performance reduces with increasing network dimensions when holding the injection rate constant. This can be attributed to the fact that a larger network size with the same injection rate implies more processing in software in terms of traffic generations and traffic receptions and therefore more processor cycles are consumed for increased number of network nodes.

Figure 18 shows the variation of the emulator performance with flit injection rate for varying packet sizes in a 5x5 2D-Mesh network. We observe that the emulator performance improves with increase in the number of flits per packet. This can be attributed to the fact that that the packet generation rate varies inversely with the

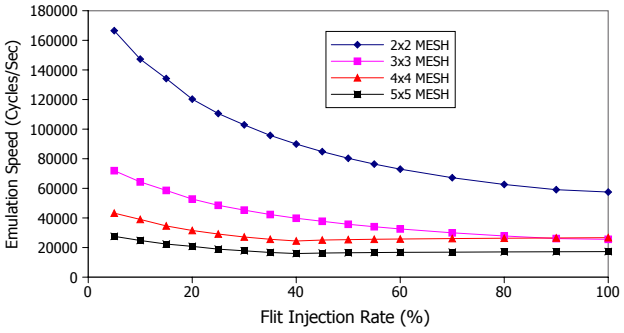

Figure 17. Emulation Speed vs. Flit Injection Rate under Varying Network Sizes

number of flits per packet. For larger packets, the TGs will have to generate fewer packets to maintain the same flit injection rate as compared to smaller packets. This decreases the number of processor cycles per emulation cycle, thus improving emulator performance. It is observed that the graph tends to flatten as the injection rate is increased beyond 45%, i.e. when the network approaches saturation. At this point, the source queues become full and the TGs initiate packet throttling.

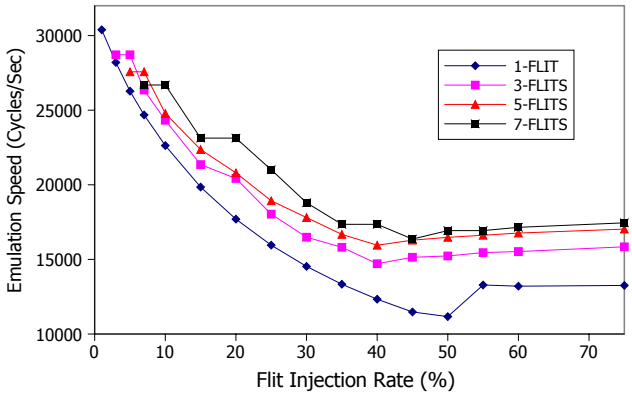

Figure 18. Emulation Speed vs. Flit Injection Rate for a 5x5 2D-Mesh Network under Varying Flit Sizes

AcENoCs' emulation speed was compared against Ocin_tsim, a software network

simulator [19], for synthetic and realistic workloads. The AcENoCs emulator was able to achieve a speedup in the range of 5-15X over Ocin_tsim for synthetic workloads. Traces generated by SPEC CPU 2000 [31] suite of benchmarks running on TRIPS [32] were used for evaluation of the AcENoCs baseline network under realistic workloads. For these workloads, AcENoCs showed a performance of approximately 17500 emulation cycles/sec, about 9X faster than Ocin_tsim. Ocin_tsim was run on an eight-core Intel Xeon processor, with each core operating at 3.2GHz. We also found that the AcENoCs emulator also shows a speedup of approximately 10000-12000X over HDL simulators under a similar set of workloads [30].

D. Hardware Evaluation

The network under test, consisting of the routers and links, was implemented on a Xilinx Virtex-5 VLX-110T FPGA. The synthesis was accomplished using the Xilinx XST synthesizer. The total resource consumption of the network on the FPGA varies with the dimensions of the network being implemented and the number of ports present on each router in the network. A study of the resource consumption of 3-Port, 4-Port and 5-port routers was done and the results obtained are indicated in Table IV for these router configurations. The table also indicates the maximum synthesizable frequency for each router configuration. It is clearly seen that the resource consumption of a single router increases significantly as we increase the number of ports on the router from 3 to 5.

The break-down of resource utilization by NoC router component is shown in Figure 19 for a 5-Port router. The chart indicates that the single largest consumer of hardware resources in a router is the VC allocator. As explained in Chapter V, the VC allocation unit is made up of two levels of arbitration. In order to reduce resource

Table IV. FPGA Resource Utilization for 3-Port, 4-Port and 5-Port Routers

ROUTER	LUTs	SLICE REGISTERS	LUTRAM	SYNTHESIS FREQUENCY
3-Port	997 (1.44%)	477 (0.69%)	24 (0.0013%)	134.644 MHz
4-Port	1691 (2.45%)	667 (0.96%)	32 (0.0018%)	124.353 MHz
5-Port	3040 (4.40%)	875 (1.27%)	40 (0.0022%)	109.064 MHz

consumption of this unit, we split a large high complexity arbiters into hierarchically connected smaller and low complexity arbiters. For example, in a 5-port router with two VCs, an arbiter with 10 request lines can be realized using two arbiters with 5 request lines each and one arbiter with 2 request lines. The arbiters with 5 request lines have much less resource utilization as compared to a single arbiter with 10 request lines. The size of the VC allocation unit grows with the number of router ports and the number of VCs supported per port. The second largest consumer of FPGA resources is the input unit. The input unit for all the five input ports combined consumes around 29% of each router's hardware resources, primarily due to the control logic for interfacing with other router components.

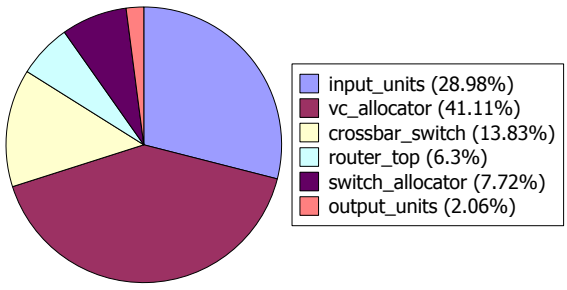

Figure 19. LUT Utilization for a 5-Port Router

A study of the router critical timing path was also done during router synthesis. Table V shows the critical path for a 5-port router. The table shows both, logic delay as well as wire delay components of the critical path. It is seen that the majority of the delay is due to the routing of wires between various components of the router. This behavior is typical of FPGA implementation tools. The input unit delay is high because it includes the Clock-to-Q delay component of the launching flip flop. VC allocator, switch allocator and crossbar switch, on the other hand, have a large delay due to their complex network of arbiters or multiplexers.

Table V. 5-Port Router Critical Path

UNIT NAME	**LOGIC DELAY**	**WIRE DELAY**	**TOTAL DELAY**
Input Unit	0.568ns	1.54ns	2.108ns
VC Allocator	0.430ns	2.987ns	3.417ns
Input Unit	0.086ns	0.380ns	0.466ns
Switch Allocator	0.258ns	1.455ns	1.713
Crossbar Switch	0.258ns	1.208ns	1.466ns
Total	1.6ns (17.5%)	7.57ns (82.5%)	9.17ns

In order to efficiently utilize the available FPGA resources and to fit a maximum dimension network on the FPGA, the unused ports on the router connected in the mesh network were removed. This reduces the complexity of the router components, especially the VC allocator. This optimization results in considerable FPGA resource savings. With this optimization, we were able to accommodate a 5x5 mesh network on the Virtex-5 FPGA.

The 5x5 mesh network represents the largest NoC which the Virtex-5 FPGA can accommodate due to its 76% LUT utilization. Around 10% of the LUT resources

Table VI. Percentage FPGA Resource Utilization under Varying Network Sizes

NETWORK	**LUTs**	**SLICE REGISTERS**	**LUTRAM**
2x2	4958 (7.17%)	2816 (4.07%)	96 (0.005%)
3x3	14637 (21.18%)	6372 (9.22%)	264 (1.47%)
4x4	29980 (43.37%)	11414 (16.51%)	512 (2.86%)
5x5	52520 (75.98%)	19569 (28.31%)	840 (4.69%)

are consumed by other components like MicroBlaze processor, RS-232 controller, clock generators, memory controllers and so on. Despite an apparent 14% remaining LUTs, larger network designs would not converge during the network synthesis step. Table VI shows the resource consumption for networks of different dimensions.

This chapter presents the results of all the experiments conducted using AcENoCs emulator and their comparison with other standard network simulators. The next chapter will present the concluding remarks and future improvements that can be incorporated in the AcENoCs emulator.

CHAPTER VIII

CONCLUSIONS AND FUTURE WORK

A. Conclusions

Advancements in VLSI technology have resulted in a tremendous increase in the computational power of processing elements and the number of processing cores integrated on a single chip. The inability of the communication infrastructure to keep pace with the increased computational power has caused a shift in focus from computation-centric design to a communication-centric design. Traditional bus based designs are not scalable and cannot efficiently handle the communication in large designs. Network-on-Chip is an effective replacement for buses in systems with large number of processing cores. An exploration of the vast design space provided by NoC in terms of router architecture, network topology, routing and flow control algorithms requires extremely fast and cycle accurate simulations to arrive at an optimum network architecture in a short time frame. Several software and HDL simulators have been developed to meet this objective, but they are either very slow or non-cycle accurate. FPGA based emulators reduce the validation time without compromising cycle accuracy and hence can be used as an alternative to software simulators.

This thesis presents AcENoCs, a novel FPGA based NoC emulator capable of fast and cycle accurate emulations. Specific focus is on the design and evaluation of the hardware framework for AcENoCs. A highly configurable library of NoC components like NoC router and links was designed using verilog HDL. Configurability is provided in terms of the number of router ports, number of virtual channels, depth and width of the virtual channel fifos, arbitration schemes and link widths. Additionally, the router could be configured to have variable number of pipeline stages ranging from

one to five stages. The design also supports link pipelining. The AcENoCs software framework adds additional flexibility by allowing easy control of parameters like the number of packets to be injected, packet size, traffic patterns and the flit injection rates. This efficient and well defined hardware-software framework makes AcENoCs an ideal platform for researchers wanting to validate their designs early in the design cycle.

Emulation tests performed to validate the accuracy of the AcENoCs hardware framework for 2D-Mesh network under several traffic patterns showed results similar to Ocin_tsim software simulator [19]. The latency and throughput plots, thus obtained, demonstrated characteristics typical of a standard 2D-Mesh network. With all the hardware optimizations, synthesis using Xilinx XST synthesizer indicated a 86% overall utilization of the FPGA resources for a 5x5 2D-Mesh network, the maximum that can be fitted on a Virtex-5 (VLX110T) FPGA. Finally, AcENoCs indicated speedups improvements of 10000-12000X over ABC network's HDL simulator [30] and 5-15X over Ocin_tsim software simulator running on a 3.2GHz processor.

B. Future Work

Currently, AceNoCs hardware framework supports emulation of 2D-Mesh and 2D-Torus networks with X-Y dimension order routing and credit based flow control. AcENoCs hardware framework can be extended to support additional network topologies including F-butterfly [33], Multidrop Express Channels (MECS) [34], as well as different routing schemes, and flow control schemes. To emulate networks larger than 5x5 on a Virtex-5 FPGA, our approach can be combined with the technique presented by Wolkotte et al. [24], with network of smaller dimensions being treated as a single block.

REFERENCES

[1] C. Duan, V. H. C. Calle, and S. P. Khatri, "Efficient on-chip crosstalk avoidance CODEC design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, no. 4, pp. 551–560, April 2009.

[2] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proceedings of the Design Automation Conference*, Las Vegas, NV, June 18-22 2001, pp. 684–689.

[3] S. Lotlikar, V. Pai, and P. Gratz, "AcENoCs: A flexible HW/SW platform for FPGA accelerated NoC emulation," in *Review IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, Scottsdale, AZ, October 2010.

[4] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys*, vol. 38, no. 1, 2006.

[5] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA: Morgan Kaufmann Publishers, 2004.

[6] H.-J. Yoo, K. Lee, and J. K. Kim, *Low-power NoC for High-performance SoC Design*. Boca Raton, FL: CRC Press, 2008.

[7] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 775–785, June 1990.

[8] H. Sullivan and T. R. Bashkow, "A large scale, homogeneous, fully distributed parallel machine," in *Proceedings of the 4th Annual International Symposium on Computer Architecture*, March 23-25 1977, pp. 105–117.

[9] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, Milwaukee, WI, May 11-13 1981, pp. 263-277.

[10] L.-S. Peh and W. J. Dally, "A delay model for router microarchitectures," *IEEE Micro*, vol. 21, no. 1, pp. 26-34, January/February 2001.

[11] P. Kermani and L. Kleinrock, "Virtual-cut through: A new computer communications switching technique," *Computer Networks*, vol. 3, no. 4, pp. 267–286, September 1979.

[12] W. J. Dally and C. L. Seitz, "The Torus routing chip," *Distributed Computing*, vol. 1, no. 4, pp. 187–196, December 1986.

[13] W. J. Dally, "Virtual-channel flow control," in *Proceedings of the International Symposium on Computer Architecture*, Seattle, WA, May 28-31 1990, pp. 60–68.

[14] W. J. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, March 1992.

[15] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547-553, May 1987.

[16] M. Coppola, S. Curaba, M. Grammatikakis, G. Maruccia, and F. Papariello, "OCCN: A network-on-chip modeling and simulation framework," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Paris, France, February 16-20 2004, pp. 174–179.

[17] T. Kogel, M. Doerper, A. Wieferink, R. Leupers, G. Ascheid, H. Meyr, and S. Goossens, "A modular simulation framework for architectural exploration of

on-chip interconnection networks," in *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, Newport Beach, CA, October 1-3 2003, pp. 7–12.

[18] K. Goossens, J. Dielissen, O. Gangwal, S. Pestana, A. Radulescu, and E. Rijpkema, "A design flow for application-specific networks on chip with guaranteed performance to accelerate SoC design and verification," in *Proceedings of the Design, Automation and Test in Europe*, Munich, Germany, March 7-11 2005, pp. 1182–1187.

[19] S. Prabhu, B. Grot, P. Gratz, and J. Hu, "Ocin_tsim - DVFS aware simulator for NoCs," in *Proceedings of the 1st Workshop on SoC Architecture, Accelerators and Workloads (SAW-1)*, Bangalore, India, January 10 2010.

[20] N. Genko, D. Atienza, G. De Micheli, J. Mendias, R. Hermida, and F. Catthoor, "A complete network-on-chip emulation framework," in *Proceedings of the Design, Automation and Test in Europe*, Munich, Germany, March 7-11 2005, pp. 246–251.

[21] N. Genko, D. Atienza, G. De Micheli, and L. Benini, "Feature - NoC emulation: A tool and design flow for MPSoC," *IEEE Circuits and Systems Magazine*, vol. 7, no. 4, pp. 42–51, 2007.

[22] F. Angiolini, "Interconnection systems for highly integrated computation devices," Ph.D. dissertation, University of Bologna, Bologna, Italy, 2007.

[23] P. Liu, C. Xiang, X. Wang, B. Xia, Y. Liu, W. Wang, and Q. Yao, "A NoC emulation/verification framework," in *Proceedings of the 6th International Conference on Information Technology: New Generations*, Las Vegas, NV, April 27-29 2009, pp. 859–864.

[24] P. Wolkotte, P. Holzenspies, and G. Smit, "Fast, accurate and detailed NoC simulations," in *Proceedings of the 1st International Symposium on Networks-on-Chip*, Princeton, NJ, May 7-9 2007, pp. 323–332.

[25] M. M. Kim, J. D. Davis, M. Oskin, and T. Austin "Polymorphic on-chip networks," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, Beijing, China, June 21-25 2008, pp. 101–112.

[26] K. Bolding, M. Fulgham, and L. Snyder, "The case for chaotic adaptive routing," *IEEE Transactions on Computers*, vol. 46, no. 12, pp. 1281–1292, December 1997.

[27] M. Galles, "Scalable pipelined interconnect for distributed endpoint routing: The SGI SPIDER chip," in *Proceedings of the Symposium on Hot Interconnects*, August 1996, pp. 141–146.

[28] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel routers for on-chip networks," in *Proceedings of the 31st Annual International Symposium on Computer Architecture*, Munich, Germany, June 19-23 2004, pp. 188–197.

[29] Xilinx Inc., *Embedded System Tools Reference Manual*, www.xilinx.com/support/documentation/sw_manuals/xilinx11/est_rm.pdf, Accessed September 2009.

[30] T. Jain, P. Gratz, A. Sprintson, and G. Choi, "Asynchronous bypass channel routers: Improving performance for DVFS and GALS NoCs," in *Proceedings of the 4th ACM/IEEE International Symposium on Networks-on-Chip*, Grenoble, France, May 3-6 2010, pp. 51-58.

[31] J. Henning, "SPEC CPU2000: Measuring CPU performance in the new millennium," *Computer*, vol. 33, no. 7, pp. 28–35, July 2000.

[32] P. Gratz, K. Sankaralingam, H. Hanson, P. Shivakumar, R. McDonald, S. Keckler, and D. Burger, "Implementation and evaluation of a dynamically routed processor operand network," in *Proceedings of the 1st International Symposium on Networks-on-Chip*, Princeton, NJ, May 7-9 2007, pp. 7–17.

[33] J. Kim, J. Balfour, and W. J. Dally, "Flattened butterfly topology for on-chip networks," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, Chicago, IL, December 1-5 2007, pp. 172–182.

[34] B. Grot, J. Hestness, S. Keckler, and O. Mutlu, "Express cube topologies for on-chip interconnects," in *Proceedings of the 15th IEEE International Symposium on High Performance Computer Architecture*, Raleigh, NC, February 14-18 2009, pp. 163–174.

VITA

Swapnil Subhash Lotlikar received his Bachelor of Engineering Degree in electronics and communication engineering from National Institute of Technology Karnataka, India in June 2005. He has three years of professional experience at Conexant Systems India Ltd., Pune where he worked as a Senior VLSI Design engineer. He joined the computer engineering program at Texas A&M University in August 2008 and received his Master of Science degree in August 2010. His research at Texas A&M University was focussed on Network-on-Chip(NoC) design and FPGA Accelerated NoC emulation. He can be reached at the following address:

c/o Dr. Paul V. Gratz
Computer Engineering Group,
Department of Electrical & Computer Engineering,
Texas A&M University,
College Station, TX 77843-3259 TAMU
Email: slotlikar@gmail.com

The typist for this thesis was Swapnil Subhash Lotlikar.