HW/SW CODESIGN AND DESIGN, EVALUATION OF SOFTWARE FRAMEWORK FOR AcENoCs : AN FPGA-ACCELERATED NoC EMULATION PLATFORM

A Thesis

by

VINAYAK PAI

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

December 2010

Major Subject: Computer Engineering

HW/SW CODESIGN AND DESIGN, EVALUATION OF SOFTWARE FRAMEWORK FOR AcENoCs : AN FPGA-ACCELERATED NoC EMULATION PLATFORM

A Thesis

by

VINAYAK PAI

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,	Gwan S. Choi
	Riccardo Bettati
Head of Department,	Costas N. Georghiades

December 2010

Major Subject: Computer Engineering

ABSTRACT

HW/SW Codesign and Design, Evaluation of Software Framework for AcENoCs : An FPGA-Accelerated NoC Emulation Platform. (December 2010)

Vinayak Pai, B.E., Visvesvaraya Technological University

Chair of Advisory Committee: Dr. Paul V. Gratz

Majority of the modern day compute intensive applications are heterogeneous in nature. To support their ever increasing computational requirements, present day System-on-Chip (SoC) architectures have adapted multicore style of modeling, thereby incorporating multiple, heterogeneous processing cores on a single chip. The emerging Network-On-Chip (NoC) interconnect paradigm provides a scalable and power-efficient solution for communication among multiple cores, serving as a powerful replacement for traditional bus based architectures. A fast, robust and flexible emulation platform is the key to successful realization and validation of such architectures within a very short span of time.

This research focuses on various aspects of Hardware/Software (HW/SW) codesign for AcENoCs (Accelerated Emulation Platform for NoCs), a Field Programmable Gate Array (FPGA) accelerated, configurable, cycle accurate platform for emulation and validation of NoC architectures. This work also details the design, implementation and evaluation of AcENoCs' software framework along with the various design optimizations carried out and tradeoffs considered in AcENoCs' HW/SW codesign for achieving an optimum balance between emulated network dimensions and emulation performance. AcENoCs emulation platform is realized on a Xilinx Virtex-5 FPGA. AcENoCs' hardware framework consists of the NoC built using configurable hardware library components, while the software framework consists of Traffic Generators (TGs) and their associated source queues, Traffic Receptors (TRs) along with

statistics analysis module and dynamically controlled emulation clock generator. The software framework is implemented using on-chip Xilinx MicroBlaze processor. This report also describes the interaction between various HW/SW events in an emulation cycle and assesses AcENoCs' performance speedup and tradeoffs over existing FPGA emulators and software simulators.

FPGA synthesis results showed that networks with dimensions upto 5x5 could be accommodated inside the device. Varying synthetic traffic workloads, generated by TGs, were used to evaluate the network. Real application based traces were also run on AcENoCs platform to evaluate the performance improvement achieved in comparison to software simulators. For improving the emulator performance, software profiling was carried out to identify and optimize the software components consuming highest number of processor cycles in an emulation cycle. Emulation testcases were run and latency values recorded for varying traffic patterns in order to evaluate AcENoCs platform. Experimental results showed emulation speedups in order of 10000-12000X over HDL (Hardware Description Language) simulators and 14-47X over software simulators, without sacrificing cycle accuracy.

To my Family and Friends for their love and encouragement

ACKNOWLEDGMENTS

Any record of work is incomplete without an expression of gratitude towards those who made it possible. A lot of people have either directly or indirectly contributed towards this thesis, and I owe a debt of gratitude to each and every one. This work may not have been possible without the strong support, constant guidance and remarkable patience of my advisor Dr. Paul V. Gratz. His approach to research and teaching and expert knowledge of the subject will continue to inspire me. I would also like to thank my committee members, Dr. Gwan Choi and Dr. Riccardo Bettati, for agreeing to be on my committee and for their valuable suggestions, availability and timely criticism. I would like to express my gratitude to Texas A&M University for giving me the opportunity to pursue my ambitions.

I also take this opportunity to thank Mr. Swapnil Lotlikar, my research partner on the AceNoCs project, for all the technical and non-technical support during research. I would also like to thank all CAMSIN research group members for their valuable suggestions.

No words are enough to express my gratitude to my family for their unconditional love and support. My parents Pratima and Ramanatha, my brother Vineeth and my grandparents have sacrificed quite a lot and provided me the best possible love and support. Whatever I am today is because of them.

Finally, special thanks to all my friends back home and at Texas A&M who have stood by me during my good and bad times.

TABLE OF CONTENTS

CHAPTER Page

I INTRODUCTION 1
A. Thesis Objective 3
B. Thesis Organization 7

II BACKGROUND ON NETWORK-ON-CHIP 9
A. Introduction to Network-on-Chip 9
B. NoC Architectural Overview 10
C. Network Topologies 11
D. Routing Algorithms 13
E. Flow Control 15
1. Packet-Buffer Flow Control Schemes 15
2. Flit-Buffer Flow Control Schemes 16
3. Buffer Management Schemes 17

III RELATED WORK 19
A. NoC Software Simulators 19
B. FPGA Based NoC Emulators 22

IV AcENoCs EMULATION PLATFORM FRAMEWORK 26
A. Overview of AcENoCs Emulation Platform 26
B. AcENoCs Hardware Framework 28
1. Network Router 31
2. Interconnection Links 33
C. AcENoCs Software Framework 33
D. Platform Interfaces 35

V AcENoCs SOFTWARE FRAMEWORK : DESIGN AND IMPLEMENTATION 36
A. Software Framework Components 36
1. Traffic Generators (TGs) 36
a. Synthetic Traffic Generator 38
b. Trace-based Traffic Generator 41
2. Source Queues/FIFOs 43

CHAPTER Page

3. Traffic Receptors (TRs) 44
4. Emulation Clock Generator 45
B. Design Decisions for Increased Emulator Performance . . . 45

VI AcENoCs HW/SW EMULATION FLOW 49

A. Hardware-Software Interface 49
B. HW/SW Flow during an Emulation Cycle 50

VII AcENoCs PLATFORM FPGA IMPLEMENTATION FLOW . . 54

A. Emulation Platform Configuration 54
B. AcENoCs FPGA Implementation Flow using Xilinx EDK Platform 56
C. Discussion 58

VIII AcENoCs PERFORMANCE EVALUATION AND VALIDATION RESULTS 59

A. Baseline Network Configuration 59
B. Hardware Framework Evaluation 60
C. Software Framework Evaluation 61
1. Software Profiling 61
2. Software Optimizations 64
D. Emulator Performance Evaluation 65
1. Evaluation under Synthetic Workloads 65
a. Comparison with other Simulators 68
2. Evaluation under Real Application based Workloads . 69
E. Emulation Testcase 69

IX CONCLUSIONS AND FUTURE WORK 71

A. Conclusions 71
B. Future Work 73

REFERENCES 74

APPENDIX A 80

VITA 82

LIST OF TABLES

TABLE		Page
I	Summary of NoC Features Supported by AcENoCs HW Framework .	30
II	Configurable Features in AcENoCs' Software Framework	55
III	Total FPGA Resource Consumption for Varying Network Dimensions	60

LIST OF FIGURES

FIGURE		Page
1	Generic 3x3 NoC System Design	10
2	Commonly Used NoC Topologies : a) 3x3 Mesh b) 3x3 Torus	12
3	X-Y Dimension Ordered Routing Example for 3x3 Mesh Network	14
4	AcENoCs HW/SW Emulation Framework	29
5	NoC Router Architecture	31
6	AcENoCs Flit Structure for 5x5 Network	37
7	Message Passing Mechanism between Dual Processors in Trace-based TG	42
8	AcENoCs HW/SW Emulation Flow	51
9	AcENoCs Emulation Platform Flow	56
10	Software Profiling Results (Average Processor Cycles per Emulation Cycle) before Software Optimizations	62
11	Software Profiling Results (Average Processor Cycles per Emulation Cycle) after Software Optimizations	63
12	Emulation Speed v/s Injection Rate for Varying Network Sizes for Bit-Complement and Transpose Traffic	66
13	Emulation Speed v/s Injection Rate for a 5x5 Mesh Network with Varying Flit Sizes for Bit-Complement and Transpose Traffic	67
14	AcENoCs v/s OCIN_TSIM Performance Comparison (in Cycles per Sec)	68
15	Average Latency v/s Injection Rate for Network Validation	69

CHAPTER I

INTRODUCTION

The shrinking feature sizes and associated technology scaling of the integrated circuits have brought about integration of large number of processing components on a single slice of silicon. With rapid advancements in VLSI technology, the computational requirements across the various application domains have increased significantly. To meet the high performance requirements of such heterogeneous applications with reasonable power consumption, the current day System-On-Chip (SoC) architectures incorporate multitude of general purpose and special processing intellectual property (IP) cores on a single chip.

Even though the computational capabilities of the processing cores have increased tremendously, their communicational capabilities have not scaled proportionately. Due to the ever increasing communication requirements between various on-chip processing cores and peripheral modules, design of a robust, power efficient and scalable communication infrastructure remains one of the key design challenges in implementation of multicore architectures. Traditionally, shared bus based communication schemes and point-to-point interconnects were used to achieve effective communication between the limited number of processing cores [1]. The bus based scheme was considered to be effective and cost-efficient for handling small scale communication requirements, while the dedicated interconnects aimed at achieving low latency communication. However, with scaling of silicon technology, these two widely used schemes have hit their limitations in terms of scalability, thereby limiting their usage to interconnecting smaller number of cores [2, 3]. In addition, long dedicated buses

---

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

and interconnects have become undesirable for use due to high power consumption, crosstalk and noise interference [4]. Longer wire length implies increased wire delay, which affects the speed of operation of a circuit. Failure of a bus or a point-to-point interconnect directly results in system failure, as no alternate paths are present from source to destination.

A new design paradigm, Network-On-Chip (NoC), proposed by Dally and Towles, serves as a effective and scalable alternative to bus based and point-to-point schemes for meeting communication requirements in a system with large number of processing cores [2]. NoCs offer low latency high bandwidth communication when compared to the bus based schemes. NoC borrows concepts from the well-established and scalable domain of computer networking. NoC primarily consists of network interfaces, routers and interconnecting links. Interconnecting links can be shared among several requestors and several links can operate simultaneoulsy, thereby exhibiting high degree of parallelism. NoCs have high fault tolerant mechanism since multiple paths exist from source to destination. Shorter links imply faster speed of operation and less noise interference. In recent years, various chip architects have turned to NoCs for providing reliable, cost-effective and energy-efficient means of communication in gigascale systems with multiple processing elements. For instance, the brainchild of Intel's terascale computing research program, called the Teraflops chip, contains 80 processing cores interconnected in a 2D mesh network configuration [5]. The chip operates at 5 GHz and is capable of delivering more than one trillion floating point operations per second. Tilera's Tile64 processor contains 64 processing tiles connected in a 8x8, 2D mesh configuration [6].

Due to the stiff competition prevailing amongst several SoC vendors, a great deal of emphasis has been placed on reduced time-to-market criterion. In addition, the challenge of dealing with ever increasing communication requirements between the

SoC components has scaled newer heights. Given this scenario, the validation of the intercommunication infrastructure between several IPs plays a significant role in the chip design cycle. Currently, a great deal of research activities are being conducted on several aspects of NoC design, particularly in the field of exploring various low latency router microarchitectures, optimized hop-count network topologies, multiple clock domain communication, intelligent adaptive routing algorithms, application mapping, etc. Hence, there is a vast design space offered by NoCs in the form of router microarchitecture, routing algorithms, network topologies and flow control schemes. In short, a fast exploration of the vast feature space offered by the NoCs along with design validation is vital to arrive at a optimum interconnect configuration required to meet the processing demands of an application.

A. Thesis Objective

Performing fast and accurate design validations at early stages in the design cycle provides the designer a critical insight into the possible design and architectural issues, thereby eliminating possible respins of the chip and contributing to significant reduction in overall design time. Typically, such early stage design space explorations and validations are performed using either the HDL (Hardware Descriptive Language) simulators or software simulators and consumes a significant portion of the overall chip design cycle. However, there are several performance tradeoffs involved with using such simulators. The performance of simulators can be evaluated using two benchmarking parameters, its ability to exhibit cycle accurate behavior and its speed in terms of cycles per second. HDL simulators are cycle accurate in nature but are extremely slow. Their simulation speeds are measured to be in the range between 3-5 cycles/sec, which is extremely slow for running real time applications. Some of

the software simulators [7, 8, 9, 10, 11] are not cycle accurate but fast, while other simulators [12, 13, 14, 15, 16] exhibit cycle accurate behavior but are invariably slow. Software based cycle accurate simulators suffer from low performance since modeling parallel nature of hardware using software techniques is an highly inefficient process. Hence, there is a great demand for validation tools which are both cycle accurate and fast.

With increasing size of applications and need for faster design validations, Field Programmable Gate Array (FPGA) based emulators have proven to be an effective replacements for HDL and software simulators due to their faster validation times combined with cycle accurate behaviour [17]. These emulators exhibit high performance efficiency by making use of actual FPGA hardware to model parallel hardware structures. The HDL is simulated at actual hardware speeds. Design validation using such emulators permit the researchers to explore, implement and validate new design ideas accurately within very short timeframe. It also serves as an ideal platform for researchers to weigh various design tradeoffs involving speed, area and power consumption. Since actual synthesized HDL is used for validation, such FPGA emulators permit accurate modeling of various architectural and design issues that may occur at actual chip operating frequency. FPGA based emulation platforms also contribute towards reducing the validation time required for hardware-software integration. The software developers can start testing the real software immediately even before the actual hardware is ready. This approach can be beneficial in detecting design and architectural issues early in the design cycle.

Several FPGA based NoC emulation schemes have been proposed in the past. Achieving greater emulation speeds has been the priority for most of the emulation schemes [18, 19], but at the cost of reduced emulated network sizes. On the other hand, there exists schemes which aim at emulating larger dimension networks while

sacrificing emulation performance [20]. There is a great demand for emulation schemes which strike an optimum balance between emulation performance and emulation network dimensions. The main contribution of this research is the Hardware/Software (HW/SW) codesign of a fast, configurable and cycle-accurate FPGA-accelerated NoC emulation platform, AcENoCs (**Ac**celerated **E**mulation Platform for **NoCs**), for validating and evaluating various aspects of on-chip interconnection networks [21]. This thesis report also details the design, implementation and evaluation of AcENoCs' software framework. This work discusses critical design decisions taken, various software code based optimizations performed on the basis of software profiling results and tradeoffs considered between emulation performance and emulated network sizes in AcENoCs' HW/SW codesign. This work also highlights the reconfigurable features available in the emulation platform together with the emulation flow between HW/SW framework and evaluates the AcENoCs platform on the basis of performance improvements and tradeoffs over existing NoC FPGA emulators and software simulators.

AcENoCs is realized on a Xilinx Virtex-5 FPGA and built using a HW/SW platform, making efficient utilization of the available FPGA's hardware resources. Achieving faster emulation performance together with realization of larger dimension NoC are the major design goals of AcENoCs. AcENoCs' balanced HW/SW framework helps it in achieving an optimum balance between emulation performance and network dimensions. AcENoCs' hardware framework is realized using FPGA's hardware resources and consists of the NoC built using configurable hardware library components (routers and links) and connected in a 2D Mesh/Torus configuration. Given limited FPGA resource space, AcENoCs can support larger dimension networks as compared to other FPGA emulators and can operate at speeds greater than software simulators. AcENoCs' high emulation performance can be attributed to its

capability to exploit the parallelism available in the hardware as opposed to the sequential nature of software simulators. AcENoCs software framework consisting of the Traffic Generators (TGs) and their associated source queues, Traffic Receptors (TRs) and statistics analysis models, and dynamic emulation clock generator is implemented on an on-chip soft IP processor. The software framework is designed for leveraging the greater state space resources available to software and for freeing additional FPGA resources that would have been consumed if the traffic models were implemented using FPGA's hardware. The software framework controls the emulation process, allows for easy reconfiguration of emulation parameters and defines a plug-and-play interface for realizing different router architectures and NoC topologies on the emulation platform.

The contributions made by this research work are as follows :

1. HW/SW codesign of a fast, cycle accurate and flexible FPGA-accelerated NoC emulation platform, called AcENoCs, for exploring the vast NoC design and feature space and validating on-chip networks.

2. Design of AcENoCs' software framework consisting of two different types of TGs capable of supporting synthetic and realistic workloads, software based dynamically allocated source queues/FIFOs (First In First Out), TRs and latency analysis modules and dynamically controlled emulation clock generator.

3. Support for configuring several NoC emulation parameters in software with minimal effort. Support for interfacing the software framework with different configurations of NoC topologies and router microarchitectures.

4. Integrated and well-defined emulation flow between various HW/SW events during an emulation cycle.

5. Several software framework based optimizations carried out on the basis of software profiling results, leading to improved emulator performance.

6. Evaluation of AcENoCs' performance for different dimension networks under varying flit sizes, flit injection rates and traffic workload conditions and subsequent comparisons/tradeoffs with other existing NoC emulators/simulators.

AcENoCs 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 software framework design.

B. Thesis Organization

This chapter presents the motivation and contribution made by this research and serves as an introduction to the content of the thesis. The rest of the thesis is organized as follows : Chapter II presents an introductory background about NoCs and presents a brief overview of basic concepts involving on-chip communication networks together with various NoC architectures, components and terminologies involved in the design of a simple interconnection network. Chapter III presents the related work in the field of NoC software simulators and FPGA based emulators. Chapter IV introduces the AcENoCs emulation framework and describes the various components of the emulation framework along with other system level details. Chapter V presents the AcENoCs' software framework, the main contribution of this research work, and discusses various design space exploration options provided by the software framework along with design decisions taken for improving emulator performance. Chapter VI presents the complete HW/SW emulation flow and describes the sequence of HW/SW interactions taking place in an emulation cycle. In Chapter VII, AcENoCs FPGA im-

plementation flow for running a complete emulation along with the reconfigurability options in hardware and software framework are discussed. Chapter VIII examines AcENoCs' performance evaluation methodology, emulation performance results obtained and its subsequent comparison with software and HDL simulators. Details regarding emulation testcase run in order to validate AcENoCs platform along with the results of software profiling are also presented. Chapter IX presents the summary of conclusions and future work.

CHAPTER II

BACKGROUND ON NETWORK-ON-CHIP

This chapter introduces the concept of Network-on-Chip (NoC) and presents the basic components and architectures involved in the design of a simple interconnection network. NoC serves as a scalable, low latency alternative to traditional bus based schemes for meeting communication requirements in an SoC environment. This chapter also presents the various concepts and terminologies associated with NoC communication scheme.

A. Introduction to Network-on-Chip

With rapid advancements in VLSI technology, the number of modules integrated on a single slice of silicon have multiplied significantly. Present day processor designs incorporate multiple processing cores to meet the high performance requirements with reasonable power dissipation. As a result, providing efficient, low latency communication between the various on-chip processing elements has become a primary concern for chip designers. A design paradigm called Network-on-Chip has been proposed as a solution for interconnection of chip multiprocessors and has emerged as a scalable alternative for bus based and point-to-point communication schemes [1]. NoCs overcome the limitations of bus based schemes by providing a scalable, low latency, high bandwidth interconnection medium. They can transfer maximum amount of data within least possible time, and well within cost and power constraints. The communication between the various nodes takes place in the form of packets. Packet based communication scheme provisions for sharing of link resources between various network nodes. NoC can be easily scaled to support communication requirements

for giga-scale systems with multiple processing cores. They also exhibit lesser wire delays, resulting in higher speed of circuit operation and have robust fault tolerance mechanism.

B. NoC Architectural Overview

A basic NoC design consists of several routing nodes interconnected via links. These routing nodes interface with the processing cores via Network Interface. Figure 1 illustrates the basic structure of a 3x3 mesh configuration NoC along with the various architectural components.

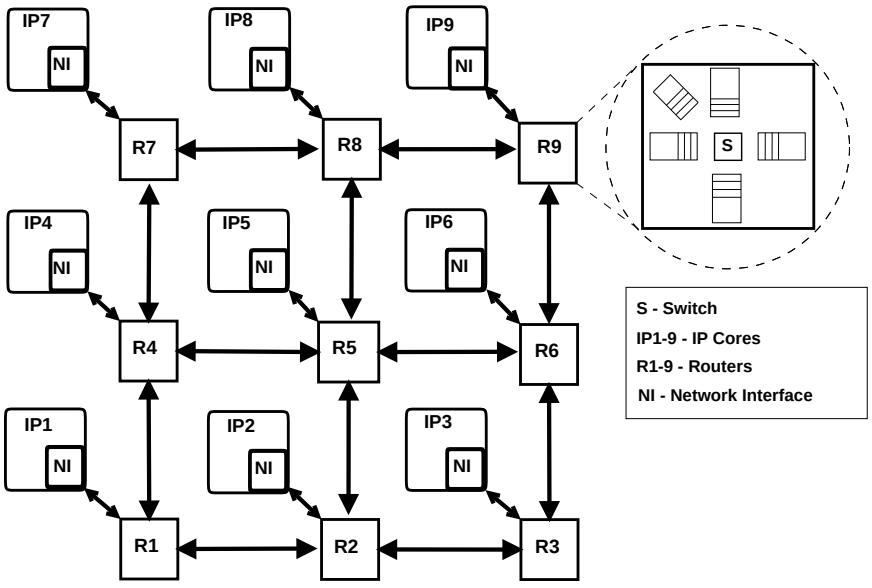


Fig. 1. Generic 3x3 NoC System Design

IP cores are the processing elements of the chip and are responsible for all the computations carried out. An IP core may represent a processor, a memory core or any peripheral device like UART. They are responsible for initiating communication in a system.

Network Interface (NI) connects the IP cores to the routing nodes on the interconnection network. NI assembles the data from the IP core into packets and injects them into the network by inserting additional routing information. Packets are further broken down into flow control units or flits, the basic units of data transmission. On the other end, NI also de-assemble the packets received from the network into data format recognized by the IP cores. Hence, NI helps in decoupling the processing cores from the network.

Routers are the main building blocks of an interconnection system. They are responsible for making the decisions involving the path to be taken for routing the packets to their respective destinations. The routing decisions are made based on the chosen routing scheme.

Links/Channels are used to interconnect the routers according to a specified network configuration and provide the raw communication bandwidth in the network. Flits are exchanged between network nodes through the data links. Dedicated control links may also be present to exchange flow control information between adjacent routers.

C. Network Topologies

A network topology defines the arrangement between the routing nodes and interconnecting links in a NoC. There are many different topologies available for interconnection, but the selection of an optimal topology depends on the required area cost, power consumption and system performance. Selection of an optimal topology is a major step in the design of an efficient NoC since the routing and flow control schemes are dependent on the nature of topology.

The most commonly used network topologies belong to the tori family, also

called k-ary, n-cubes [1]. These topologies contain nodes arranged in a n-dimensional grid, with each dimension housing k nodes. Each node is connected to its adjacent neighbor via a pair of links, one along each direction. Torus and mesh configurations are examples of k-ary, n-cube topologies. For instance, 3x3 2D mesh can be configured as 3-ary 2-mesh and 3x3 torus as 3-ary, 2-cube. Each of the nodes can be configured as a terminal node, acting as source and sink for data communication, or switching node, for forwarding the packets to the destination. Figure 2 depicts 3x3 2D torus and 2D mesh network topologies.

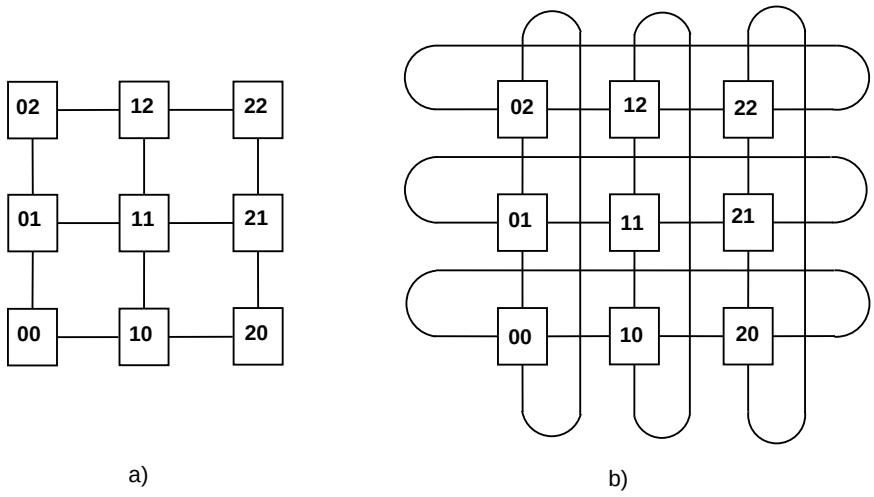


Fig. 2. Commonly Used NoC Topologies : a) 3x3 Mesh b) 3x3 Torus

Nodes connected in a torus configuration form a ring structure in each dimension, with each node connected directly to the adjacent nodes. The nodes along the network periphery are connected using a wrap-around link. As a result, torus networks display symmetric characteristics along the network edges, resulting in uniform distribution of load across the interconnecting channels.

Mesh networks are similar to torus, the only difference being the absence of

wrap-around links along the network edges. The nodes along each dimension form a linear array, with each node connected to its neighbor via direct point-to-point link. Mesh networks are asymmetric along the network edges and are susceptible to load imbalance problems, especially along the links connecting the central nodes.

D. Routing Algorithms

Routing schemes govern the path taken by a packet in order to traverse from its source to destination [1]. Routing decisions are made on a per-hop basis at each intermediate node till the packet reaches its destination. Hence, a packet traversing across the same source-destination pair may take multiple paths in the network based on the type of routing algorithm chosen. The type of routing algorithm chosen plays a significant role in defining the performance of packet switched networks, where routing decisions are made at each switching node.

A good routing algorithm should be capable of handling anomalies in the traffic and should be able to distribute the load evenly throughout the network. It should also aim at reducing the number of hops from source to destination, thereby reducing the message latency. It should be able to handle faults in the network without affecting the network throughput.

Commonly used routing algorithms fall under two categories : Deterministic routing and Adaptive routing [1].

In deterministic routing, packets traverse through the same pre-decided path for a given source-destination pair. These algorithms are commonly used since they are easy to implement and can be made deadlock free with minimal effort. However, since path diversity of the network is not exploited fully, they are incapable of balancing the load uniformly through the network. Source routing and X-Y dimension ordered

routing (X-Y DOR) algorithms fall under this category. In source routing, the source specifies the entire route information to the destination in the packet header. As the packet traverses through each intermediate node, the route information bits in the header are stripped off. In case of X-Y DOR, packets are routed first along the X-direction and then along the Y-direction till the destination is reached. X-Y DOR is commonly used due to its ease of implementation and capability to avoid deadlocks by restricting the formation of resource dependency cycles. Figure 3 shows X-Y DOR for a 3x3 mesh network for a packet traversing from node 00 to 22 and back.

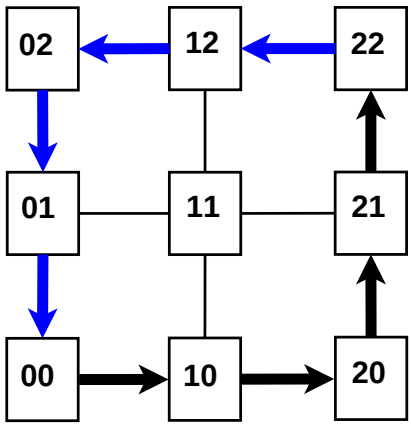


Fig. 3. X-Y Dimension Ordered Routing Example for 3x3 Mesh Network

In adaptive routing, the route taken by packets traversing between a source-destination pair may vary depending upon the current network state. Information regarding each router's buffer occupancies and interconnection link faults are taken into consideration while making the routing decisions at each intermediate node. Hence, these algorithms help in reducing congestion and have robust fault tolerance mechanism. However, they may be susceptible to deadlocks since they use only the local network state information while making routing decisions. They are highly

complex to implement.

E. Flow Control

Flow control is a mechanism that governs how the network resources are allocated to packets transmitted over the network [1]. Network's resources include channel bandwidth, buffer allocation and capacity and control state. A good flow control technique utilizes the network resources in an efficient manner, thereby achieving high channel bandwidth and low message latency.

Flow control schemes fall under two classes : Bufferless flow control and Buffered flow control [1]. Bufferless flow control drops or misroutes the packets if the network resources are not allocated immediately to the requesting packet. Buffered flow control provides temporary storage for blocked packets waiting for network resource allocation. Adding buffers in the network nodes decouples the allocation of adjacent channels, thereby preventing the blocked packets from getting dropped. Buffered flow control achieves better allocation and utilization of network resources and is the preferred flow control scheme in majority NoC designs. Allocation of network resources can be done at a packet level or flit level.

1. Packet-Buffer Flow Control Schemes

In this type of flow control scheme, network resources consisting of buffers and link bandwidth are allocated to packets.

In store-and-forward flow control, packets are buffered at each intermediate node till the entire packet is received. The packet is forwarded to the next node only if there is sufficient storage space available in the downstream node's buffer to store the entire transmitted packet. Also, the entire routing link's bandwidth must be

reserved exclusively for the packet to be forwarded. Since the entire packet needs to be buffered before it is forwarded ahead, this scheme suffers from high message latency overhead.

Cut-through flow control overcomes the high latency overhead in store-and-forward scheme. The packets are forwarded to its next hop destination as soon as their headers are received and necessary network resources are acquired. This eliminates the need to wait for the entire packet reception prior to packet forwarding, thereby resulting in high channel utilization and low latency.

However, packet-buffer scheme suffers shortcomings in the form of inefficient utilization of buffer storage space for packets. Secondly, since the entire link needs to be reserved till the packet is received in its entirety, other packets waiting to use the link may suffer from high contention latencies.

2. Flit-Buffer Flow Control Schemes

Here, the network resources are allocated on flit-by-flit basis.

Wormhole flow control is similar to cut-through, except that the network allocation happens at a finer granularity of flits. On arrival of a head flit at a particular node, it is immediately forwarded to its next hop destination once it acquires the network resources. Body and tail flits follow the head once they acquire the necessary resources. Hence, buffer space and link bandwidth required for forwarding only one flit needs to be allocated. Due to this reason, wormhole scheme utilizes the buffer space more efficiently. However, networks with wormhole scheme may suffer from throughput overhead since flits of a blocked packet may acquire several intermediate links, thereby rendering these links to be unusable for other waiting flits.

Virtual-channel (VC) flow control overcomes the packet blocking and idle channel bandwidth problem by allowing other flits to utilize the physical link even when a flit

using the same link has stalled. As a result, VC scheme achieves higher throughput compared to wormhole flow control. Each physical channel is divided into number of VCs. Similar to the wormhole scheme, an arriving head flit is forwarded as soon as it acquires a flit buffer in the downstream node, flit channel bandwidth and a VC in the physical link connecting the downstream node. Body and tail flits of a packet are allocated the same VC as its head flit but may not get instant access to the physical link since flits from other VCs may be contending for the same physical link.

3. Buffer Management Schemes

The main aim of buffered flow control schemes is to provide effective utilization of the network's resources without the packet being dropped. To ensure no packets are dropped, there needs to be some sort of communication between the buffers on the upstream and downstream node. This communication takes place through a dedicated control link between two adjacent nodes. The upstream node forwards flits only when there is sufficient space available in the downstream node's buffer. Commonly used buffer management schemes are credit-based, on/off and ack/nack [1].

In credit-based flow control, the upstream node keeps track of number of free buffer locations (called credits) available per VC in the downstream node. The credit counter is decremented every time a flit is forwarded to the downstream router. A credit is sent to back to the upstream router, via control link, once the downstream node forwards the flit. This causes the credit counter to be incremented. The upstream node stops forwarding flits when the credit count reaches zero count.

In on/off flow control, the downstream node sends a signal to the upstream node in the form of a control bit. This control bit status decides whether the upstream node is allowed to transmit the flit (on state) or not (off state). The downstream node sends an OFF status signal to the upstream node once the occupancy of buffers

in the downstream node crosses a high threshold value. Similarly, ON status is sent once the downstream buffer level falls below a certain low threshold value. On/off flow control scheme helps in reducing the amount of upstream signaling.

In ack/nack flow control, upstream node doesn't maintain any information regarding buffer availability status in the downstream node. The upstream node forwards the flits to the downstream node and waits for some sort of acknowledgment. If there is buffer space available in the downstream router, it responds back with a 'ack' signal to the upstream node. Negative acknowledgment (nack) is sent back in case the flit is dropped by the downstream node due to lack of buffering space. In such a scenario, the dropped flit is retransmitted. In any event, the upstream node must hold on to the flit until it receives an ack from its downstream counterpart. This scheme is rarely used due to its inefficient link bandwidth utilization.

CHAPTER III

RELATED WORK

Owing to the need for exploration of the vast design space offered by the NoCs and for identifying critical design and performance bottlenecks early into the chip design cycle, several researchers have proposed different kinds of NoC simulation frameworks and tools. The principle approach followed by these methodologies have focused on performing detailed and cycle accurate simulations or achieving reduced validation times (faster simulation speeds) or both. These methods can be broadly classified into two categories – Software Simulators and FPGA based Emulators.

A. NoC Software Simulators

NoC simulations using software simulators can be performed at different levels of abstraction. Operating at different levels of abstraction involves tradeoffs between simulation speed and accuracy. Simulation time increases as we move towards more cycle accurate and precise simulations. Several System C and HDL based simulation frameworks have been proposed for performing NoC simulations at system level.

Coppola et al. describe a NoC modeling and simulation framework based on C++ library built on top of System C [7]. The framework defines a communication API for performing design space exploration, validation and system level performance modeling of NoC components at various levels of abstraction. Kogel et al. propose a modular exploration framework, built using System C, for modeling and evaluating different types of on-chip network configurations like crossbar topology, shared bus and dedicated links [8]. This framework, built for system level exploration and performance modeling of on-chip communication networks, carries out data exchange

modeling at high levels of abstraction, thereby trading off full cycle accuracy for simulation speed. Another System C based NoC simulator, NNSE, proposed by Lu et al. and built for Nostrum NoC, allows the user to explore the NoC architectural space by configuring various NoC features like topology, flow control, routing schemes, etc. [9]. It also analyzes subsequent performance impact caused by varying the NoC configurations in terms of latency/throughput and performs extensive design validation through the use of synthetic and realistic traffic workloads. Pestana et. al present another System C based simulator which uses transaction level model for modeling the NoC components at higher levels of abstraction [11]. The user needs to specify XML files to model NoC architectures with varying topology, interconnection links and IP-to-network mapping. Since modeling is carried out at higher levels of abstraction, these System C based simulators compromise cycle accuracy for higher simulation speeds.

On the other hand, several researchers have prioritized full cycle accurate validations and have proposed simulators with full cycle accuracy but these simulators suffer from low speed performance. Siguenza et al. propose a VHDL based cycle accurate simulator for validating Proteo NoC [14]. This simulator allows reconfigurable and reusable IP blocks to be modeled into NoC architectures having ring, star and bus topologies and reports the latency and throughput parameters. Bertozzi et al. describes a cycle accurate simulator with NoC architecture components and links modeled in System C, used for verifying custom tailored NoC topologies [22]. OCIN_TSIM implements a C++ based cycle accurate simulator for modeling NoCs but its simulation speed makes it inefficient to execute real application trace workloads [12]. Many other cycle accurate simulators like Nirgam [23] and Worm_sim [24] have also been proposed.

In order to increase the simulation speed of cycle accurate simulators, various

approaches have been proposed. A configurable, mixed framework simulator built using System C and VHDL is presented by Goossens et al. [13]. The VHDL based model accounts for cycle accurate simulation but is time consuming, while the System C models are faster than their VHDL counterparts and simulate the NoC at flit-level and IP-to-network interface at transaction level. Since there are two types of simulation environments, the user can choose the appropriate environment based on the type and needs of the application to be validated. However, maintaining different abstraction models for desired tradeoff between accuracy and speed based on application domain is a difficult task owing to the need for precise synchronization between these models. A mixed SystemC/VHDL environment, called NoCGEN, used for modeling and evaluating various aspects of on-chip interconnection networks have been presented by Chan et al. [16]. It uses a configurable, modularized and synthesizable set of hardware library components to create an NoC architecture with different router configurations having varying data widths, routing algorithms and flow control schemes. System C is used to realize the traffic generators and traffic ejectors.

The inherent problem with these software based simulators lies in the sequential nature of execution. The simulation speed degrades significantly with increase in network dimensions and with a move towards lower levels of abstraction. This degradation in simulation speed makes them unsuitable for executing real application traffic workloads. Real application traces provide an accurate measure of the network performance. Even though these simulation tools providing accurate measure of the network performance metrics like latency, throughput and bisection bandwidth, they are incapable of providing accurate information related to total area occupied by the NoC architecture along with the impact of area on performance. To arrive at an optimum NoC configuration for a particular application, a researcher would want to consider all aspects of interconnect network design in terms of area, speed,

power consumption and subsequent effect on performance due to variations in these parameters.

B. FPGA Based NoC Emulators

In recent years, FPGA based emulation schemes have come into spotlight since they overcome the limitations imposed by the software simulators and allow for faster architectural space explorations and detailed, accurate design validations. Genko et al. propose a HW/SW emulation platform based on Xilinx Virtex-II Pro FPGA housing an embedded PowerPC processor and emulating a network of six routers [17, 18]. In order to reduce the overall simulation time, their scheme emulates real processing core behavior using traffic generator and receptor models. The hardware framework is comprised of network components forming the network to be emulated, traffic models and a control module. The configurable and synthesizable NoC components are generated using Xpipes compiler [25]. Two types of traffic generators, stochastic and trace based, are provided to support synthetic and trace workloads. The control module is responsible for synchronizing the various traffic components in the platform. The software running on the embedded PowerPC processor has the capability to program most of the NoC platform parameters, thereby controlling the entire emulation process. The source queues associated with each processing core is modeled in hardware and their sizes are statically decided. This results in inefficient utilization of the available FPGA provisioned memory in cases where the traffic conditions are non-uniform, ultimately resulting in premature throttling of packet generation process. The amount of FPGA hardware resources (look-up tables and flip flops) consumed by TG/TRs and control logic is expected to increase with increasing network dimension sizes. Since the framework uses hardware based traffic models, any

modifications/feature additions to these models requires a complete re-synthesis of the hardware. This can be a time consuming process especially for FPGA devices with resource consumption close to their capacity. Experimental runs show that the entire emulation process completes in a matter of few seconds as compared to hours taken by the simulation approach. Using such a framework allows exhaustive validation of the NoC by bombarding it with excessively large amount of packets within a short interval of time, something which is not feasible with software simulation approach.

Another flexible HW/SW NoC emulation platform for 4x4 mesh network, NoCOP, has been proposed by Liu et al. [19]. NoCOP's hardware framework is similar to the one proposed by Genko et al. In order to efficiently utilize the FPGA's hardware resources, the authors propose an FPGA based scheme without the on-chip processor. Instead, the emulation system is made configurable and controlled using external instruction set simulator (ISS) and USB communicator running on a host computer. USB based communication is used to establish connection between the external ISS and the traffic models on FPGA. The hardware framework consists of the network to be emulated, TGs, TRs, packet controller and analysis module and USB controller, all being implemented on FPGA hardware, thus occupying additional hardware resources. Since the ISS runs on external host computer, the communication latency will be high as compared to latencies incurred by an on-chip embedded soft processor.

Several other schemes have also been presented. Wolkotte et al. [20] present an FPGA based emulator capable of emulating large homogeneous and heterogeneous NoCs on a single FPGA. In the emulation framework proposed by Genko et al. [17], the size of the emulated NoC is limited by the amount of hardware resources available inside the FPGA. This places a firm restriction on further exploration of NoC design

space in terms of network dimensions. Scheme proposed by Wolkotte et al. overcomes this limitation by adapting a sequential style of modeling to emulate a large dimension NoC. Each router block is executed sequentially, one block per functional clock cycle. The state of each simulated router block along with its link states are stored in a large memory and retrieved in the subsequent functional cycles. The hardware platform consists of an FPGA board and an SoC board containing dual core ARM processors and an on-chip memory module for storing the router states. A memory interface is built inside the FPGA to gain access to the stored router states. The dual core ARM processors split the work of generating and consuming traffic amongst themselves. The advantage offered in terms of hardware area is visible when homogeneous NoCs are modeled. For homogeneous NoCs, all routers are identical in design and operate on the same clock domain. Hence, only one type of router architecture block is synthesized into the FPGA, resulting in minimal consumption of FPGA resources. Moreover, there is no restriction on the size of the NoC being emulated. However, since this scheme doesn't exploit the true parallel nature of hardware to speed up simulations, experimental comparisons with System C simulator shows a performance speed of only 80-300X.

There are other FPGA emulation platform tools like NoCem which explore the on-chip interconnection architecture in the context of multiprocessor communication [26]. This platform is capable of emulating memory system architecture and interprocessor communication architecture. It provides a realistic estimate of the communication latencies incurred for interprocessor communication as well as for processor-memory communication. Valle et al. present a highly scalable framework for modeling and evaluating complex, heterogeneous Multi-Processor SoCs (MPSoCs) in a fast and cycle accurate way [27]. The framework provides modeling and fast statistics extraction at three different levels : IP core, memory system and interconnection links.

Addition of IPs or memories have very minimal effect on emulation speed, thereby achieving a speedup in the range of 3X as compared to other cycle accurate MPSoC simulators. Among other commercially available FPGA based emulators, ZeBu-XL [28] uses multiple FPGA platforms to emulate a large dimension NoC. Inspite of being fast, these emulators are cost expensive and are not ideal for performing fast design space explorations due to lack of flexibility.

CHAPTER IV

AcENoCs EMULATION PLATFORM FRAMEWORK

In this chapter, we introduce AcENoCs emulation platform consisting of the hardware framework and the software framework. We also present system level details of the embedded platform on which AcENoCs is realized. The hardware framework consisting of configurable hardware library with NoC components will be presented in this chapter. The various components constituting the software framework will be introduced to the reader. More architectural and implementation aspects of the software framework will be presented in greater details in the next chapter.

A. Overview of AcENoCs Emulation Platform

In chapter III, we examined several NoC software based simulators and concluded that these exhibited a tradeoff between cycle accurate simulations and the simulation speed. We also concluded that there is a lack of simulation tools which perform exceedingly well in terms of speed and accuracy. We also reviewed various FPGA based NoC emulation tools that serve as an ideal replacement for software simulators due to their cycle accurate behavior and high emulation performance. Such FPGA emulation schemes serve as an ideal platform for researchers wanting to explore myriad design configurations and sort them accordingly based on speed, area and power consumption.

AcENoCs emulation platform was born out of a need to perform fast and accurate NoC feature space explorations and design validations and at the same time, emulate larger dimension network, given limited FPGA hardware resources. AcENoCs is a flexible, cycle accurate and FPGA-accelerated emulation platform built for running

fast and detailed NoC emulations [21]. It is designed using a HW/SW framework, can support larger dimension NoCs as compared to other FPGA based NoC emulators and can operate at speeds greater than software simulators. Additionally, AcENoCs' high emulation performance and utility can be attributed to HW/SW codesign optimizations performed based on hardware resource evaluation and software profiling results.

The major design goals of the AcENoCs emulation platform are listed below :

1. Supporting larger dimension NoC : AcENoCs can support larger dimension NoCs on FPGA when compared with other proposed FPGA emulation schemes [17, 19]. This can be attributed mainly to the implementation of the traffic models in software, which gets implemented using FPGA provisioned memory. This results in freeing of additional FPGA resources: Look Up Tables (LUTs) and Flip Flops (FFs), hence paving way for implementation of larger dimension NoCs on the hardware.

2. Fast Emulation Performance : AcENoCs is designed for running fast NoC emulations without sacrificing cycle accuracy. The reason for fast emulation performance can be attributed to the fact that the NoC is modeled using the FPGA's hardware resources, and hence efficiently exploits the parallel nature of hardware. It can generate more emulation cycles per second as compared to the software simulators. Our experimental results show a performance speed of 10000-12000X when compared to the HDL simulators and 14-47X when compared with cycle accurate software simulators like OCIN_TSIM [12]. AcENoCs can complete the entire emulation process in a matter of few seconds as compared to software schemes which require significant amount of time.

3. Reconfigurability and Flexibility : Since AcENoCs platform is built for fast NoC

design explorations during early phase of the chip design cycle, it is an essential requirement to provide flexibility and reconfigurability options to the end user. Several features of the hardware framework are made user configurable, thereby supporting exploration of wide range of router architectures and NoC topologies. The software framework allows for easy reconfiguration of emulation parameters and defines a plug-and-play interface for integrating different router architectures and NoC topologies in the emulation platform. The TGs can support synthetic traffic patterns as well as realistic trace workloads. Any feature additions to the traffic models can be made with very minimal effort in software and requires a mere recompilation.

AcENoCs is realized on a Xilinx XUPV5 FPGA board [29]. The XUPV5 board houses a Virtex-5 (VLX110T) FPGA device. AcENoCs is centered around a HW/SW framework with the NoC consisting of routers and links, implemented on FPGA hardware, constituting the HW framework and the software running on the MicroBlaze soft processor constituting the software framework. Among the peripherals found on the XUPV5 board, UART serial communication interface, 256MB DDR2 RAM, 1MB SRAM, System ACE compact flash interface and the JTAG programming interface also form a part of the framework. The hardware and the software framework interact via a register based interface through the Processor Local Bus (PLB). AcENoCs' HW/SW framework is illustrated in Figure 4.

B. AcENoCs Hardware Framework

AcENoCs hardware framework consists of the on-chip interconnection network to be emulated . The NoC is constructed using a custom designed hardware library of configurable NoC components : network routers and its interconnecting links.

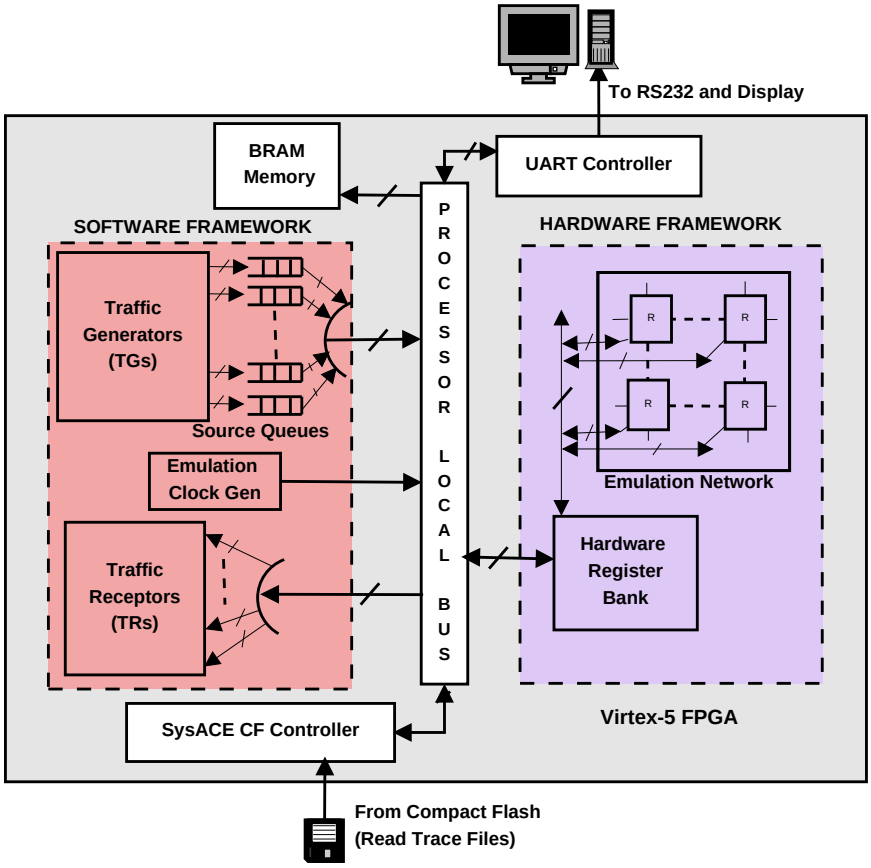


Fig. 4. AcENoCs HW/SW Emulation Framework

Routers can be interconnecting using links according to specified configuration to realize different network topologies. Several design parameters of the hardware framework components such as router and link parameters and network configurations can be configured by the user prior to emulation process, thereby adding flexibility to the emulation platform. In our current release, the AcENoCs emulation platform allows the routers to be connected in either 2D mesh or torus topologies.

The communication between the router nodes takes place in the form of packets. Packets are sub-divided into smaller transmission units called flits. Flits are the basic units of data transmission and storage. Flits are further classified into head, body

and tail flits. The head flit contains the the vital routing information required for a packet to traverse from source to destination. The body and tail flits contain no routing information and must follow the head flit through the network.

In the present release, AcENoCs supports X-Y dimension ordered routing (X-Y DOR). As stated earlier, allocation of network resources to a flit in the form of channels and storage buffers is decided by a flow control scheme. In AcENoCs, the flow control scheme is user configurable. The current release supports VC flow control using credit based buffer management. VC buffers help in avoiding network deadlocks by decoupling the request-reply message path and reserving separate, dedicated virtual channels for request and reply. They also help in increasing the throughput of the network by allowing the waiting packets to surpass the blocked packet, thereby making efficient use of the available channel bandwidth [30].

Table I shows the network features supported by the current release of AcENoCs for the NoC realized as part of its hardware framework.

Table I. Summary of NoC Features Supported by AcENoCs HW Framework

Network Feature	**Available Options**
Network Topology	2D Mesh and 2D Torus (upto 5x5)
Routing Scheme	X-Y Dimension Ordered Routing
Flow Control	Virtual-Channel Flow Control
Buffer Management	Credit-based Flow Control
Emulated Link Width	$\geq$ 32 bits

Described below are the components of the custom built hardware library that constitutes the hardware framework of AcENoCs emulation platform.

1. Network Router

Figure 5 illustrates the custom built router architecture implemented in AcENoCs. The router used in the AcENoCs hardware framework is a standard five-port, single stage pipelined router. The router can also have three or four ports depending on the topology requirement and the physical position of the router in the NoC. The router is modular in structure, coded using Verilog HDL and fully synthesizable. The router architecture is comprised of various blocks viz. input unit, route computation unit, VC allocation unit, switch allocation unit, crossbar switch and output unit. Route computation, VC allocation, switch allocation and crossbar traversal are all carried out in parallel in a single router stage and complete in one network cycle. Various routers can be interconnected via links in either 2D mesh or 2D torus configuration to realize an interconnection network.

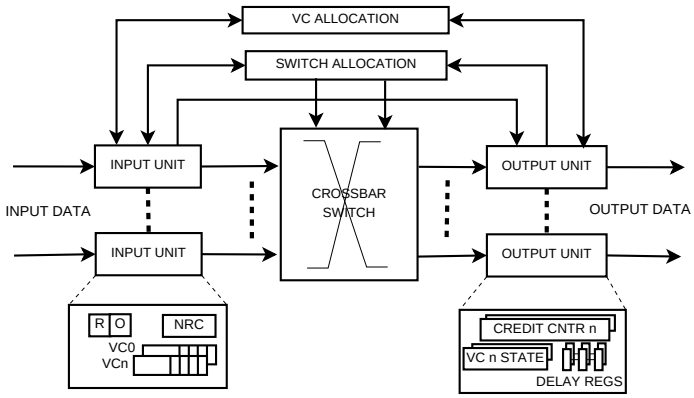


Fig. 5. NoC Router Architecture

Each router has five physical ports : North (N), South (S), East (E), West (W) and Local (L). The L port of each router is connected to the software based TGs/TRs models, through which traffic is injected into and ejected from the network.

Several router parameters like number of VCs per router physical port, depth of each VC buffer, packet data width, number of router and link pipeline stages and router arbitration schemes are made configurable.

A brief explanation of the router components is provided below :

1. Input Unit : One input unit is instantiated per input port of the router. The input unit receives flits from its neighboring routers and buffers them in VC FIFOs, based on VCID (Virtual Channel ID) field in the head flit, till the flits are forwarded to the downstream router. Each input unit also computes the output port for a packet at the downstream node using the next hop route computation (NRC) unit, based on the chosen routing scheme [31].

2. Virtual Channel Allocation Unit : The VC allocation unit is responsible for allocating an available output VC to a packet from amongst several VCs at the output port as indicated by the routing unit in the input unit. This unit consists of several arbiters and implements both round robin and fixed priority arbitration schemes.

3. Switch Allocation Unit : The switch allocation unit controls the crossbar switch access and decides which input port should gain access to the output port through the crossbar. This unit consists of several arbiters and implements both round robin and fixed priority arbitration schemes.

4. Crossbar Switch : It is the switching resource of the router and provides physical path from each input port to each output port. The crossbar unit provides multiple simultaneous connections from the input to the output. The access to the crossbar switch is controlled by the switch allocation unit.

5. Output Unit : One output unit is instantiated per output port of the router.

The output unit contains the credit counters required for establishing credit based flow control. It also maintains the availability status of each output virtual channel associated with that particular output unit.

6. Programmable Delay Unit : This unit consists of series of programmable delay registers that delay the valid output from appearing onto the physical link. These delay registers can be configured to emulate variable depth router pipeline stages and link traversal delays.

2. Interconnection Links

Routers are interconnected according to the specified network topology using the interconnection links. Data flits are exchanged between the routers using these links. Hence, links provide raw communication bandwidth in a network. Link traversal is completed in a single network cycle if no link delay registers are instantiated. Credits (tokens), required for credit based flow control, are also exchanged between the routers via another set of dedicated links. The data width for these interconnection link is 32 bits, minimum required to carry routing information in the header flit. Different link widths can be easily emulated by varying number of flits per packet.

C. AcENoCs Software Framework

AcENoCs software (SW) framework is realized on Xilinx MicroBlaze, an embedded soft processor core. As shown in Figure 4, the SW framework consists of Traffic Generators (TGs), Source Queues/FIFOs for each router node, Traffic Receptors (TRs) and Emulation Clock Generator. The SW framework controls the entire emulation process via the emulation configuration file. The SW framework is designed to free up additional FPGA resources (LUTs and FFs) that would have been consumed,

had the traffic models been implemented in HW. This provisions for realization of a larger dimension NoC on FPGA. Several emulation parameters such as total packets to be injected, flit injection rate, type of traffic pattern, etc. that control the emulation parameters can be configured using the software, making the platform extremely flexible. The MicroBlaze processor is a 5-stage, single issue pipelined processor and operates at a maximum clock frequency of 125 MHz. The software program running on the MicroBlaze is stored in FPGA based on-chip BRAM (Block RAM) memory resources. MicroBlaze accesses the FPGA BRAM using a dedicated Local Memory Bus interface (LMB). Alternately, higher capacity off-chip DDR2 RAM or SRAM can also be used for storage purposes, but this comes at the cost of reduced emulation speed due to increased access latencies. The SW framework is also responsible for generating the emulation clock dynamically and driving it to all the hardware network components.

A brief overview of the functionality of each component constituting AcENoCs SW framework is presented below.

1. Traffic Generators (TGs) : Each router is associated with a TG. TGs are responsible for building flits, generating packets and injecting them into the source queues associated with each hardware router. There are two different types of TGs supported in AcENoCs : Synthetic TGs for generating different types of synthetic traffic patterns and Trace-based TGs for supporting execution of real traces. TGs are also responsible for latency bookkeeping for each generated packet.

2. Source Queues : Each router node is associated with a source queue. A source queue operates in a First-In First-Out (FIFO) fashion. These structures provide temporary storage for flits prior to their injection into the network. These

queues are allocated and built dynamically at runtime during the emulation process. The buffered flits in these queues are injected into the network through PLB based registers.

3. Traffic Receptors (TRs) : Each router is also associated with a TR. TR is responsible for decoding the information in the packet received at the destination router and for validating each packet's destination. TRs are also responsible for calculating latency for each received packet. Passing of received packet data from destination router to TRs happens through PLB based register interface.

4. Emulation Clock Generator : The emulation clock driving the network is dynamically generated and controlled by the processor depending on the software operation latency. Such a scheme helps in achieving proper synchronization between the HW/SW events occurring in an emulation cycle.

D. Platform Interfaces

There are three major interfaces used by the AcENoCs emulation platform.

1. UART Serial Interface interfaces the XUPV5 board to the monitor for displaying various emulation statistics and debug messages to the end user.

2. SystemACE Compact Flash Interface is used by the MicroBlaze processor for reading the trace data stored in the external flash card into the FPGA.

3. JTAG Programming Interface is used to download the FPGA configuration file, generated by the Xilinx EDK platform, onto the FPGA.

CHAPTER V

AcENoCs SOFTWARE FRAMEWORK : DESIGN AND IMPLEMENTATION

In this chapter, the software environment built around the NoC in AcENoCs emulation platform is described in detail. We will discuss the design and implementation details of various software components that form an integral part of AcENoCs software framework. More details on the functionality of traffic models will be provided. Advantages gained by opting for a software based source queues and dynamically controlling emulation clock scheme will be presented. We will conclude this chapter by discussing some of the design decisions and optimizations undertaken to improve the overall performance of the emulator.

A. Software Framework Components

As discussed in Chapter IV, AcENoCs software framework mainly comprises of the software environment built around the NoC for validating the interconnection network extensively. AcENoCs software framework is coded entirely in C language. The main components of the software framework along with their design features and implementation details are provided below :

1. Traffic Generators (TGs)

Each NoC router on the hardware is associated with a software-based TG model connected through its local port (L). TGs act as the source for all data in the on-chip communication network. By opting for a software based scheme for implementing the TGs, additional hardware resources (LUTs and FFs), required for hardware based TGs, are freed and they can be effectively utilized to realize a larger dimension

network in hardware. TGs are responsible for building the head, body and tail flits, assembling them into packets and injecting the generated packets into their respective source queues flit-by-flit. TGs generate flits according to the chosen traffic pattern and the process is controlled by flit injection rate. The number of flits to be generated per packet is decided based upon the packet size and flit size entities specified by the user. Eventually, these flits are read out from the source queues and get injected into the network through the source router's local port. During the process of building a flit, TGs generate various flit fields such as flit type, packet destination address, source router's output port number for routing the packet to the next downstream node, source router's local port input virtual channel identifier (VCID) and packet id. Figure 6 shows the structure of the head, body and tail flits generated by the TGs.

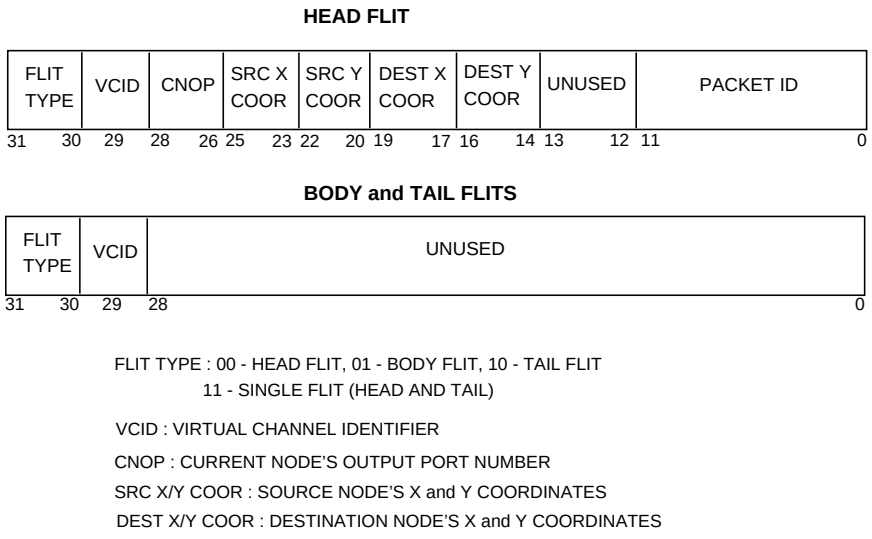


Fig. 6. AcENoCs Flit Structure for 5x5 Network

Apart from packet generation, TGs also play an active role in latency bookkeeping for each packet. Each TG stores the actual network cycle when a packet was

injected into its source queue in a separate data structure. This information is accessed by the TRs to calculate the packet latency when the packets reach their final destinations. Each TG instantiates a separate two-level data structure for storing the packet injection time of all the packets generated by that particular TG. At the first level, this data structure is indexed using the injected packet's source node id. To gain access to the second level of the structure, packet id field of the packet is used as an index qualifier. The final indexed location contains a 16-bit field for storing the packet injection time.

TGs initiate packet throttling process when the network experiences congestion conditions. During heavy congestion conditions, source queues associated with each router node become full and may overflow owing to the back-pressure due to congestion in the network. In such a scenario, TGs start packet throttling by delaying the generation and injection of packets into source queues.

There are two broad classes of traffic supported by the TGs : synthetic workloads and trace based realistic workloads. Synthetic workloads aim at abstract modeling of underlying network characteristics, when the network is subjected to real application traffic [1]. Examples of synthetic traffic patterns include uniform random, bit complement, etc. Realistic workloads represent real-time network traffic generated by a NoC system running full load applications. The AcENoCs emulation framework provides support for both these workloads. AcENoCs supports two flavors of TGs : Synthetic TG and Trace-based TG. The selection of TG to be instantiated in the emulation platform depends on the chosen traffic pattern type.

a. Synthetic Traffic Generator

AcENoCs' synthetic TGs are realized on a single MicroBlaze processor environment operating at 125 MHz. The synthetic TGs support seven commonly used synthetic

workloads for evaluating interconnection networks. All synthetic patterns, except Uniform Random, direct each source nodes traffic in the network to a unique predefined destination node. These seven patterns include :

1) Uniform Random : Each source node sends a packet to every other node in the network with equal probability. The destination node is generated using a uniform random process.

2) Bit Complement : The destination node coordinates are obtained by subtracting the source node X-Y coordinates from the maximum X-Y coordinates in the network.

3) Bit Reversal : The destination node coordinates are obtained by reversing the bit positions of source node coordinates.

4) Matrix Transpose : The destination node coordinates are obtained by directly taking the transpose of the source node coordinates.

5) Bit Shuffle : The destination node coordinates are obtained by performing a single shift rotate left operation on the source node coordinates.

6) Bit Rotation : A single shift rotate around right operation is carried out on source routers coordinates to determine destination coordinates.

7) Hotspot : Two kinds of traffic are generated in hotspot traffic pattern : hotspot traffic and background traffic [32]. During hotspot traffic generation, all source nodes in the network direct their traffic towards a specified set of one or more destination nodes. In the current release, we only support a single hotspot destination node. Once the hotspot traffic rate is achieved, TGs switch over to generating background traffic. The background traffic can be any synthetic pattern and can be configured by the user.

A uniform random injection process governs the packet generation in synthetic traffic mode. The probability that each node generates a packet in an emulation cycle

depends on the user specified flit injection rate and uniform random number generation. This involves generating an uniform random number for each node in every emulation cycle and comparing the number with the packet injection rate. Packet injection rate is derived from the user defined flit injection rate. A packet will be generated if the random number is lesser than the packet injection rate. AcENoCs uses an XOR-shift algorithm, proposed by Marsaglia, for generating uniform random numbers [33]. This algorithm was chosen due to its ability to generate highly accurate uniform random numbers within a specified range and most importantly, its consumption of much fewer processor cycles per number generation as compared to other techniques examined such as glibc's rand() function. The XOR-shift scheme uses XOR and shift operations for generating uniform random numbers within a specific range (consuming about 24 processor cycle per number generation) and completely avoids the division and modulus operations required by glibc's rand() function. During the software profiling stage, it was observed that a floating point divide operation in MicroBlaze consumes around 80-100 processor cycles, thereby incurring a huge impact on the emulator performance.

Even with the XOR shift algorithm, the random number generation still remains a bottleneck for achieving high emulator performance considering the fact that a random number needs to be generated for each node in every emulation cycle. To reduce this overhead, AcENoCs provide an option for pregenerating a prime number of uniform random numbers and storing them in a circular buffer statically prior to emulation. By choosing to generate prime number of uniform random numbers, a behavior similar to actual uniform random injection process for each TG can be emulated. The larger the prime number selected , greater will be accuracy of the uniform random process. This scheme reduces the random injection process to mere fetching of the stored array of numbers from memory, thereby boosting emulator performance.

This option can be handy when emulation performance assumes higher priority compared to the accuracy of the injection process. According to our experimental results, pregeneration scheme for random numbers was found to provide a emulation performance speedup ranging from 5-10% when compared to XOR shift scheme. By default, AcENoCs uses XOR shift algorithm for uniform random number generation. Pregeneration scheme can be enabled by the user in the emulation configuration file.

b. Trace-based Traffic Generator

AcENoCs also supports simulation of real application based trace files. Trace files are indicative of real time traffic generated by a network running full load applications. The sizes of the traces can run up to several gigabytes, thereby requiring some sort of external storage. In AcENoCs, the software running on the processor reads the trace data from compact flash (CF) card. A CF card slot is present on the XUPV5 Board and a CF interface needs to be instantiated inside the FPGA in order to read the traces. The processor reads and decodes each line of the trace data in order to inject a packet from a particular node. Each trace line contains information related to to-be-generated packet's source and destination node coordinates along with the packet injection cycle. Reading each trace line from the flash card, decoding the packet information and then generating packets in a sequential manner by a single processor incurs huge latencies and therefore contributes to significant degradation of the emulator performance. Considerable performance improvement can be achieved if the process of reading from flash card and generating packets can take place concurrently. AcENoCs achieves the parallelization of these two processes by implementing a dual processor environment, MicroBlaze1/MB1 and MicroBlaze2/MB2, with software on MB1 implementing a trace based TG to execute the trace and software on MB2 dedicated for reading the trace data from the flash card.

In the dual processor scheme, the trace file read operation, performed by MB2, is overlapped with the actual execution of the traces in MB1. Since MB2 prefetches the trace data and passes it to MB1, MB1 can directly start executing the traces without actually being delayed by any trace file I/O operations, as would be the case if the two processes were sequential in a single processor environment. The message passing from MB2 to MB1 happens through a shared storage structure : Shared Block RAM FIFO (SHBRAM) buffer. By overlapping the trace file I/O operations with the trace execution, AcENoCs is able to achieve significant speedups for emulating realistic workloads as compared to other software simulators. This dual processor environment synthesizes at a maximum frequency of 100 MHz.

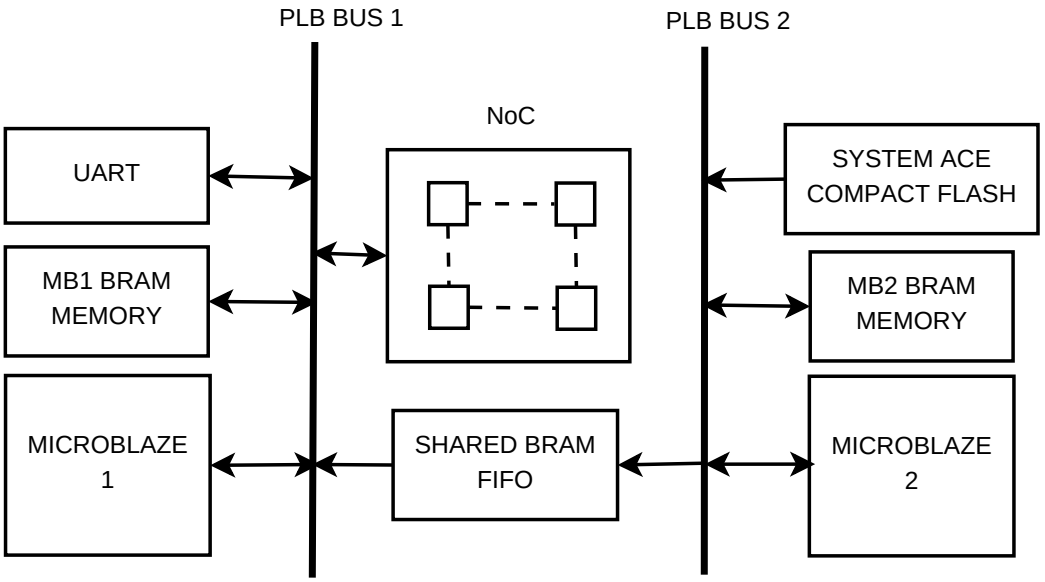


Fig. 7. Message Passing Mechanism between Dual Processors in Trace-based TG

The dual processor scheme for Trace based TGs is shown in Figure 7. The message passing structure, SHBRAM, is implemented as a shared object, which means only one processor is granted the access rights to SHBRAM at any instant of time.

In order to prevent any locking issues, each processor accesses the SHBRAM in an ordered sequential fashion. MB2 performs file read operation on the trace file and stores the trace data into the SHBRAM of size 32KB. At the start of each emulation cycle, MB1 reads the data from the SHBRAM and decodes the packet information. If the decoded packet needs to be injected in the current network cycle, MB1 stores the packet fields and then fetches the next data. The SHBRAM read process continues till MB1 encounters a packet whose intended injection cycle doesn't match the current network cycle. MB1 then builds the flits, as per the decoded flit fields, generates packets based on the stored source and destination node information and injects them into their respective node's source queue. During heavy congestion, MB1 initiates packet throttling by delaying further reads to the SHBRAM till enough space is available to hold atleast one more flit in the source queues. Throttling can also occur when the number of in-flight packets from each source becomes equal to the maximum number of in-flight packets allowed from a node at any instant (defined by number of packet id bits reserved in the header flit). This condition ensures correct calculation of packet latency values.

2. Source Queues/FIFOs

The source queues for each router node provide temporary storage for packets that have been generated prior to their injection into the network, and are implemented in software as singly-linked lists. Source queues decouple the traffic generation process at each node from the network state. This ensures that the generated packets are not lost and are injected into the network based on space availability in the source node's input VC FIFOs. Each source queue operates in a First In First Out fashion. The source queues for each terminal are dynamically allocated at runtime during the emulation process. The width of each location in the source queue is equivalent

to the emulated flit width. In every emulation cycle, packet generated by each TG is inserted into their respective source queues. The total number of outstanding flits that could be accomodated across all source queues is calculated at compile time based on the remaining software memory after the software program is loaded into memory. In every emulation cycle, a flit is read from the head of each non-empty queue and transmitted to the corresponding router through the input data and data valid registers. Once read out, the memory allocated to that flit is freed. Dynamic allocation of source queue depth avoids premature throttling of packets for non uniform traffic, as would have been the case if the source queues were implemented on hardware.

3. Traffic Receptors (TRs)

As shown in Figure 4, each router node is also associated with a TR. The TRs perform decoding of information in packets received at the destination router and validates its destination. TRs also plays a significant role in performing packet latency analysis by calculating the latency for each received packet. The latency of each received packet is calculated by accessing the packet injection time stored in the two level latency structure at the packet's source. The latency structure is indexed based on received packet's source node address and packet id. When a packet reaches its hardware based destination router, it is passed onto its software based associated TR through PLB based output data register and output status register. At the end of the entire emulation process, the average latency per packet is computed. Network throughput is also calculated by dividing the total number of flits received by total number of emulation cycles.

4. Emulation Clock Generator

In AcENoCs, the emulation clock driving the network routers on FPGA hardware is generated and controlled by the MicroBlaze processor (MB1 for Trace based TG). The emulation clock period is dynamically controlled depending on the software operation latency per emulation cycle. A single network emulation cycle is made up of several processor cycles. Generating and controlling the emulation clock using software helps in achieving proper synchronization between the different hardware and software events scheduled to occur in every emulation clock cycle. Dynamically controlling the emulation clock is more effective than the static clocking scheme based on worst case software operation latency since the hardware on the FPGA can be clocked at a much faster clock rate. Since the emulation clock rate directly depends on the latency of software operations scheduled to occur per emulation cycle, the clock rate decreases with increasing network sizes. This can be attributed to the fact that as network size increases, total software operation latency also increases due to increase in the packet generation and reception operations for the additional network nodes.

In AcENoCs, every emulation cycle defines an ordered sequence of hardware and software events. The emulation clock is allowed to proceed ahead to the next emulation cycle only when the software completes all assigned functions in a given emulation cycle (both traffic generation and reception). The emulation clock is transmitted to the hardware components on FPGA using PLB based clock register.

B. Design Decisions for Increased Emulator Performance

Two possible implementations of MicroBlaze are available in the Xilinx Embedded Development Kit (EDK) platform. : the area optimized 3–stage pipelined processor and the frequency optimized 5–stage pipelined version. Since the 3–stage version is

area optimized, it synthesizes at a much lesser frequency of 100MHz as compared to 125MHz in the 5–stage version. The software code for traffic generation and reception involves multiple looping constructs, if/else comparators and calls to various traffic functions, which maps to branches in the assembly code. Branches act as bottlenecks for achieving higher instructions per cycle (IPC) for a processor since they contribute to stalls due to control hazards. MicroBlaze implements a static branch prediction scheme with all branches treated as not taken. To reduce the branch latency overhead, automatic delay slots are inserted by the compiler after every branch instruction. Branch mispredictions incur 2 cycle penalty (1 cycle penalty with branch delay slot) for both versions of MicroBlaze. This implies lower branch misprediction penalties (in terms of cycle time) for the 5-stage pipeline, since it operates at a higher clock frequency. Reduced branch misprediction penalty coupled with the higher operating frequency of 5–stage pipeline gives higher performance as compared to the 3–stage pipeline architecture. Hence, AcENoCs implements the frequency optimized 5–stage MicroBlaze processor. In order to reduce the performance impact due to branch penalties, we tried to reduce the number of branch decisions in the assembled code. To achieve this, we carried out optimizations like loop unrolling, function inlining, reducing the number of function calls, etc. in the software to reduce the number of branches. Based on the expected workload, we also choose to instantiate a floating point unit (FPU) and a hardware barrel shifter as add-ons to the MicroBlaze. Significant performance improvement was observed with the addition of hardware barrel shifter since it completes any number of shifts in a shift operation, required for building flits, in exactly one processor cycle.

Software based implementation of the TGs and TRs helps in realizing a larger dimension network on the FPGA , as compared to the hardware based implementation schemes [18, 19]. By implementing the TGs and TRs in software, the FPGA

provisioned BRAM memory is used for storing the code associated with them. If the TGs and TRs were to be implemented in hardware, additional overhead in terms of FPGA logic resources (LUTs and FFs) would have incurred to implement these components. Considering increasing network sizes, the amount of FPGA hardware resources utilized for realizing TGs and TRs would have increased proportionately, leading to realization of a relatively smaller dimension NoC on the FPGA. Implementing the TGs and TRs in software allows efficient usage of FPGA resources by making room for additional routers and thereby, realizing a larger emulation network.

The emulation scheme described by Genko et al. employs a timestamp field in every packet for performing latency analysis [18]. The timestamp field is an overhead for maintaining statistics in hardware and limits the minimum emulated flit size. This results in loss of some flexibility in the emulator. Since AcENoCs maintains separate latency timekeeping structures for each TG to perform latency analysis, there is no need to transmit the packet injection timestamp field along with the packet. In AcENoCs, the minimum emulated flit size is 32 bits, the minimum required to carry route information in head flit.

In the previously proposed FPGA emulation schemes [17, 19], the source queues for each router were implemented in hardware and hence their depths had to be statically allocated prior to hardware synthesis. This is a serious disadvantage in schemes with hardware source queues considering non uniform traffic conditions. Eventually, the emulator may be forced to prematurely throttle the packet generation process under non-uniform traffic conditions. Throttling of the packets has implications on the packet injection rate since it drops below its specified value. This results in non accurate modeling of congestion behavior in the network. AcENoCs overcomes this premature throttling problem by implementing the source queues in software and allocating their depths dynamically. AcENoCs initiates packet throttling when source

queues overflow for injection rates higher than the network saturation rate. AcENoCs is able to sustain transitory injection rates which are higher than the network saturation injection rate for a longer time, thereby providing more accurate packet latency and network throughput analysis.

The decision to opt for a embedded MicroBlaze processor core for AcENoCs' software framework over the conventional external processor scheme is justified by the significant amount of emulation speedup achieved. This can be attributed to significant decrease in communication latencies between the hardware and the software components for an embedded processor core.

CHAPTER VI

AcENoCs HW/SW EMULATION FLOW

This chapter presents details on interactions between the hardware and software framework in AcENoCs emulation platform. It describes the PLB based register based interface which acts as the backplane for HW/SW interactions. This chapter also presents the entire HW/SW emulation flow and the various interactions happening between the HW/SW framework in an emulation cycle.

A. Hardware-Software Interface

AcENoCs emulation framework is realized on a Xilinx XUPV5 board, using the Xilinx Embedded Development Kit (EDK). This framework facilitates active interaction between the NoC hardware and software library components. AcENoCs' hardware framework and software framework interact with each other through the Processor Local Bus (PLB). We implement a 32 bit register based interface for providing communication between the emulation network and traffic models running on the MicroBlaze processor. This register bank is connected to the PLB bus and is made software accessible using a set of standard library functions defined by the Xilinx EDK tool. The register bank consists of six categories of registers :

1. Input Data Register : It is used to hold the flits that needs to be injected from the source queue into the local port VC FIFO of each source router. There exists one input data register per router on the network.

2. Input Data Valid Register : It is used as a data qualifier for the input data injected into local port of each router. A single valid bit is reserved in the register for each router.

3. Input Status Register : It indicates each router's local port input VC FIFOs full status. The number of status bits reserved for each router in the register depends on the the number of VC FIFOs configured per router port.

4. Clock Register : It is used to provide software generated clock to the emulated network.

5. Output Data Register : It holds the packets that are ejected from the network. This data is then read by the processor and passed to the respective TRs for destination validation and latency analysis. There exists one output data register per router.

6. Output Status Register : It indicates the availability of a packet at the destination router's output. The processor reads this register to check which routers have received a packet in the current emulation cycle and then reads the corresponding output data register. A single status bit is reserved in the register for each router.

B. HW/SW Flow during an Emulation Cycle

In this section, we present a detailed description of AcENOCs' complete HW/SW emulation flow during one complete network emulation cycle. Figure 8 depicts the sequence of HW/SW events (denoted as A,B, etc.) taking place in an emulation cycle.

Detailed simulation flow of these HW/SW events per emulation cycle are provided below :

A. On the positive edge of the emulation clock, the network components constituting the hardware framework are triggered. Flits are exchanged between the router nodes. A packet which has arrived at its intended destination router gets

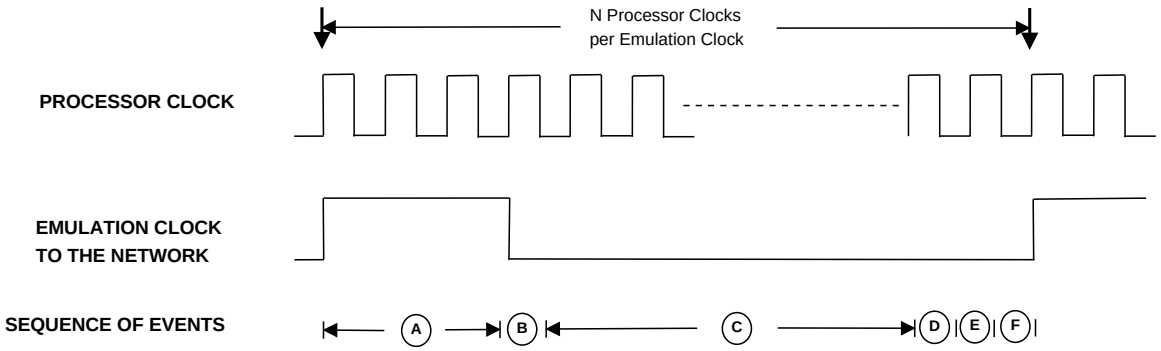


Fig. 8. AcENoCs HW/SW Emulation Flow

ejected through that routers' local port into its corresponding output data register. The availability of packets at the output router is indicated through the output status registers associated with each router. The necessary fields from the ejected packet such as its origin address, destination address and packet id are captured in the output data register.

B. The emulation clock is toggled and made low by writing 0 into the emulation clock register. Each router's local port FIFO full status is read from the corresponding bits in the input status register.

C. Traffic generation process is invoked sequentially for each TG node. A random injection process based on uniform random number generation and flit injection rate determines whether a TG gets to generate a packet during that emulation cycle. If a packet needs to be generated, the TG builds the packet and inserts the flits into its source queue. During congestion, no software memory may be available to allocate storage space for flits in the source queues. In such a situation, TGs initiate packet throttling and don't generate any packets until sufficient storage space becomes available. A flit is read from each non-empty

source queue and transmitted to corresponding router's local port VC FIFO by writing the flit data into the input data register and setting the data valid bit associated with that router. In case a router's local port FIFO full status is set, the flits are not read from the corresponding source queue.

D. Software reads the output status register to determine which among the output data registers contain valid received packet information. Based on this information, the appropriate output data register is read and the read packet data is forwarded to respective TR.

E. The TRs, on receiving valid packet data, decode the packet information and validate if the packets have arrived at the correct destination. Along with the packet decoding, the TRs also carry out latency analysis for each received packet based on the retrieved source id and packet id. Network throughput is also calculated.

F. The emulation clock is made high by writing 1 into the emulation clock register. This indicates the start of the next emulation cycle.

The emulation process is terminated after all the injected packets have been received at the destination or once an error is encountered. Error may occur when all the transmitted packets have not been received, when a packet has reached a wrong destination node or when the specified total outstanding flits in the source queues is more than the allocated software memory. Emulation may also be stopped when there is more than one packet in-flight, originating from the same source node, having the same packet id at any instant. This is due to the fact that such a condition may lead to incorrect calculation of packet latency value since the packet latency structure at each TG stores the packet injection time based on packet id field. If this condition is

not dealt with, initially transmitted packet's injection time gets overwritten with the latter packet's injection time. To avoid this latency miscalculation condition, a valid bit is associated with each entry in the TG latency bookkeeping structure. Before transmitting a packet, TG checks if the valid bit is reset. If it is not reset, an error is thrown. TGs set this bit when they inject the corresponding packet into the network. The TRs, on receiving a particular packet, reset the valid bit corresponding to the entry in the latency structure.

At the end of the emulation process, various emulation output statistics such as total number of packets injected/ejected, total number of emulation cycles, average latency and throughput and total emulation time are displayed to the end user.

CHAPTER VII

AcENoCs PLATFORM FPGA IMPLEMENTATION FLOW

This chapter focuses on the FPGA implementation flow of AcENoCs using the Xilinx Embedded Development Kit (EDK) platform. It also describes the various input files and libraries required at different stages of the implementation flow. This chapter also presents the user configurable options offered by the AcENoCs platform. We conclude this chapter with a discussion on advantages and flexibility offered by the AcENoCs' implementation flow.

A. Emulation Platform Configuration

As stated earlier, reconfigurability of the emulation platform was one of the major design goals of AcENoCs platform. AcENoCs emulation platform can be configured for different emulation runs using two configuration files :

1. Network Configuration File : This configuration file is used to configure AcENoCs' hardware framework parameters. The user can configure various network parameters like network dimensions, network topology, routing algorithm, flow control schemes and flit width. Various router parameters like number of router ports, number of VC FIFOs per router port, depth of each VC FIFO, router arbitration scheme and number of pipeline stages can also be configured via network configuration file. Any change in parameters specified in the network configuration file would require a complete hardware re-synthesis.

2. Emulation Configuration File : This configuration file supports configuration of AcENoCs' software framework parameters. The parameters specified in this configuration file control the entire emulation process. Various emulation pa-

rameters such as total number of packets to be injected, flit injection rate, traffic pattern, packet sizes and flit width can be configured via emulation configuration file. Software framework parameters such as uniform random number generation scheme, total number of outstanding flits in the source queues, enabling debug mode, selection of hotspot traffic node and hotspot packet injection rate can also be configured. Table II shows the reconfigurable options available in AcENoCs' emulation configuration file.

Table II. Configurable Features in AcENoCs' Software Framework

PARAMETERS	**AVAILABLE OPTIONS/VALUES**
Max. Number of Packets	1 – (2^{32}-1)
Packet Size	Fixed, Random
Fixed Packet Size	32 – 256 bits
Random Packet Size	Min – 32 bits, Max – 256 bits
Flit Size	32 – 256 bits
Type of Traffic Pattern	Uniform Random, Bit Complement, Bit Reverse, Shuffle, Bit Rotation, Transpose, Hotspot, Traces
Uniform Random Number Generation	XOR-shift method, glibc's rand(), Pregeneration Rand Method
Total No. of Outstanding Flits (across all queues)	6500 – 10250 (for 256KB SW BRAM Memory)
Flit Injection Rate	1% – 100%
Hotspot Node and Injection Rate	Any node, 1% – 100%
Heap Memory and Stack Memory Sizes	Depends on remaining SW memory after SW program is loaded into memory
Debug Mode	Available, not enabled (default)

B. AcENoCs FPGA Implementation Flow using Xilinx EDK Platform

AcENoCs emulation framework is realized using Xilinx EDK Platform. Xilinx EDK facilitates easy integration of NoC hardware library components with the NoC software library components, thereby creating a stand-alone embedded emulation platform on the FPGA for faster, cycle accurate NoC emulations. In this section, we deal with the FPGA implementation flow for realizing AcENoCs emulation platform using Xilinx EDK.

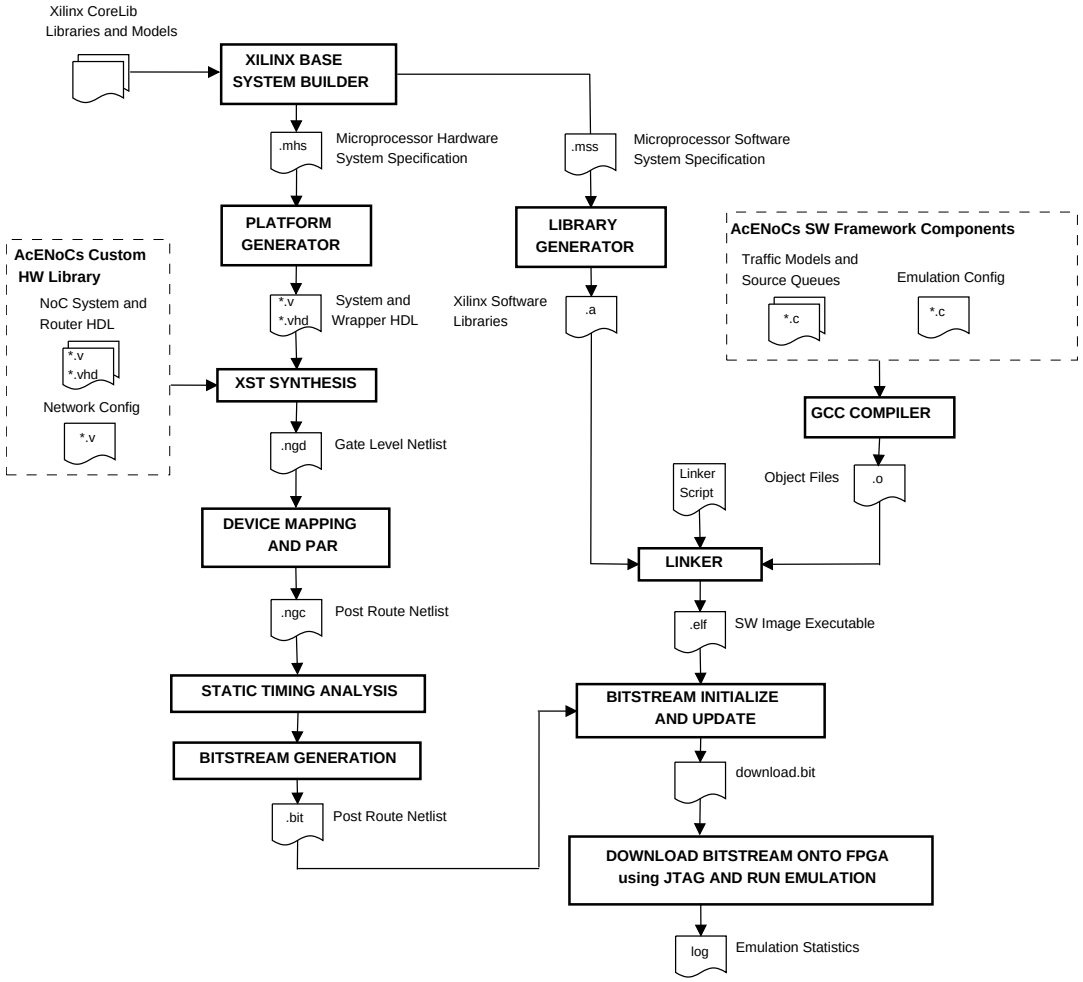


Fig. 9. AcENoCs Emulation Platform Flow

Standard Xilinx EDK synthesis flow [34], shown in Figure 9, was followed for generating the executable to be downloaded into the FPGA for starting the emulation process. The primary step in the EDK synthesis flow involves building a base system consisting of MicroBlaze processor and other necessary peripherals along with a standard bus interface such as PLB for interfacing the processor with different peripherals. This is accomplished using the Xilinx Base System Builder (BSB). The amount of FPGA BRAM memory to be associated with MicroBlaze can be specified during BSB process. NoC hardware library components consisting of network configuration file and the NoC to be emulated is then included into the base system. The base system also instantiates user-defined PLB interface registers for providing interaction between the HW/SW framework components on the FPGA.

Once the BSB is ready, synthesis of the entire hardware framework is carried out using the Xilinx XST Synthesis tool. The outcome of this step is the generation of gate level netlist for each component defined in the base system. These netlists are then given as inputs to the MAP and Place and Route (PAR) tool to map the design elements to device resources and place these synthesized cells efficiently on the device layout. This is followed by Static Timing Analysis (STA) step to verify that the synthesized design meets the desired timing constraints. The MAP, PAR and STA are part of Xilinx ISE design suite. Successful completion of these steps leads to the creation of an hardware platform based executable .bit image file. The total time for bitstream generation process varies according to the logic resource utilization of the FPGA resources. Time taken is higher for larger dimension networks since more FPGA logic resources are consumed.

Once .bit executable containing the FPGA hardware is generated, the NoC software library components consisting of emulation configuration file and the software framework components such as traffic models are then compiled using GCC compiler.

A linker script describing the memory mappings for the code and data sections of the software program in FPGA BRAM memory is generated. The BRAM heap and stack sizes are also specified in the linker script. The heap holds the static variables and the dynamic memory allocated for flits in the source queues. The outcome of this process is .elf, a software image executable. In the final step, the .bit and .elf bitstreams are initialized and combined into a single downloadable bitstream file, download.bit, which is downloaded onto the FPGA. This initiates the emulation process. Emulation results are displayed on the monitor using the serial UART interface.

C. Discussion

The ease with which emulation parameters can be changed for each emulation run is the standout feature in AcENoCs platform. By varying the various emulation parameters in the emulation configuration file and recompiling the software, the user can explore design aspects of NoC in a very short time span without hardware re-synthesis. To rerun emulation, the user just needs to re-program the FPGA with the new executable. Hence, AcENoCs provides flexibility very similar to that offered by software simulators. AcENoCs will serve as an ideal platform for researchers who wish to explore the NoC design space by exploiting the various features available in the emulator. Experimentation with new features such as addition of a new traffic pattern require changes in the software framework alone. AcENoCs' software framework defines a plug-and-play interface for integrating the traffic models with different router architectures and NoC topologies. Changes in the underlying network hardware can be achieved with minimal effort by modifying only the network configuration file. All these features combined with its fast and cycle accurate nature makes AcENoCs an ideal tool for quick, efficient and accurate NoC design space explorations.

CHAPTER VIII

AcENoCs PERFORMANCE EVALUATION AND VALIDATION RESULTS

Evaluation of AcENoCs performance is the main focus of this chapter. We examine the underlying performance evaluation methodology and evaluate AcENoCs' performance against OCIN_TSIM software simulator [12] and the ABC network's verilog HDL simulator [35]. We present the results of hardware evaluation, software profiling, emulator performance against simulated network sizes and workloads. Also presented are some details on emulation testcases run in order to validate AcENoCs platform.

A. Baseline Network Configuration

AcENoCs' baseline network configuration was used for obtaining all the results presented in this section. Any changes or deviation from the baseline configuration have been described explicitly. The baseline network configuration consists of a 5x5 dimension NoC configured using a 2D mesh topology. X-Y DOR is the chosen routing scheme and VC based flow control with credit based buffer management is the default flow control scheme. No programmable delay registers were instantiated to emulate router pipeline depth, hence the router was a single cycle router. Each router port is configured to contain two VC FIFOs with each FIFO deep enough to hold eight flits. Flit size was chosen to be 32 bits and each packet was made up of five flits (a head flit, three body flits and a tail flit). Bit-complement synthetic traffic pattern was used and the emulation was carried out for one million packets.

B. Hardware Framework Evaluation

In this section, we evaluate the FPGA area consumed by the NoC implemented on the hardware. We present the total FPGA resource utilization obtained by synthesizing varying dimension NoCs. The FPGA resource utilization also varies with the number of ports present on each router. A 5-port router consumed more logic resources than 4-port and 3-port router. We were able to accommodate a maximum 5x5 dimension by pruning out the unused ports on the routers at the periphery of the mesh network. The advantage gained in terms of fitting a larger dimension NoC in hardware as a result of moving TGs/TRs to software is evident from the fact that AcENoCs was able to fit a maximum size network of 3x3, when its hardware framework was instantiated on a Xilinx Virtex-II Pro FPGA device as compared to Genko's scheme which was able to fit a maximum size network of six routers, given the same device.

Table III illustrates the total FPGA resource utilization against varying network dimensions. As seen, FPGA LUTs consumption forms the bottleneck for implementing larger dimension NoCs on FPGA.

Table III. Total FPGA Resource Consumption for Varying Network Dimensions

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

C. Software Framework Evaluation

AcENoCs' network emulation clock is dynamically generated by software and consists of several processor cycles. The emulation clock controls the frequency at which the NoC operates and therefore defines the emulator's performance. Even though the hardware can be clocked at a much higher rate, the emulation clock frequency is limited by the software operation latency. High emulation performance can be achieved if the software operation latency is minimized. Performing profiling on software framework can help in identifying the software component that forms the bottleneck towards achieving high emulation performance. With this as our primary motive, MicroBlaze profiling was carried out for varying flit injection rates using the 32–bit processor cycle counters available in the Xilinx EDK platform. Increase in flit injection rate implies more packet generations and injections into the source queues by the TGs at each node and hence greater software operation latency in an emulation cycle. As a result, we expected to see increase in the number of processor cycles per emulation cycle with increasing injection rates. This resulted in degradation of emulator performance (emulation cycles per second) with increasing flit injection rates.

1. Software Profiling

Figure 10 shows the software profiling results with no software based optimiations applied. The graph indicates average processor cycles per emulation cycle for varying flit injection rates. All the values are averaged over 10,000 packet generations and receptions using the baseline network configuration. This profiling was carried out on the first stable version of the AcENoCs emulator. It can be seen from the graph that the emulator performance was well below the expected standards. It is clearly visible

that Packet Generation function along with Random Number Generation functions consume majority of the processor cycles per emulation clock. Hence, it was decided to optimize these two processes in order to achieve better emulation performance. Based on these conclusions, software code optimizations were carried out, targetting packet generation and random number generation processes, in order to improve the performance of the emulator.

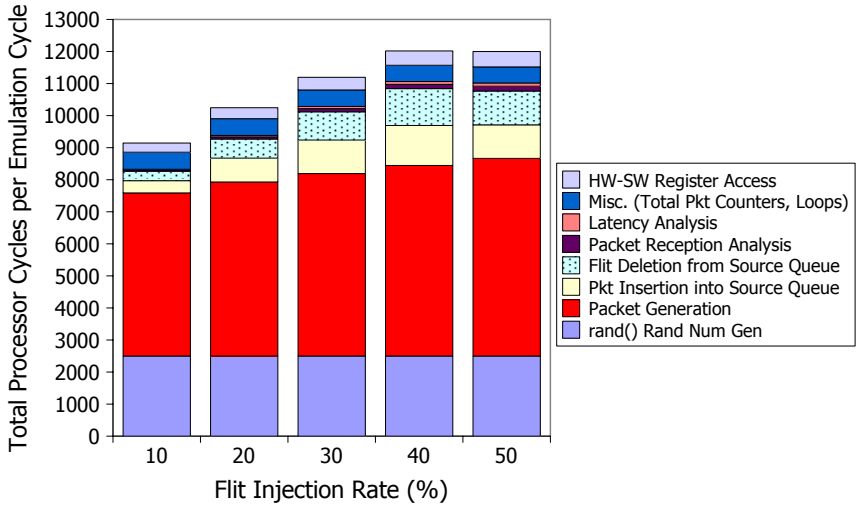


Fig. 10. Software Profiling Results (Average Processor Cycles per Emulation Cycle) before Software Optimizations

Figure 11 shows the latest profiling results for AcENoCs with software based code optimizations applied. The graph indicates the average processor cycles per emulation cycle with varying flit injection rates. Also illustrated is the breakdown of processor cycles consumed by each software component per emulation cycle. Note the improvement in emulator performance as a result of the software optimizations performed. These software based optimizations undertaken in order to improve emulator performance will be explained in detail in the next sub-section. Any reference

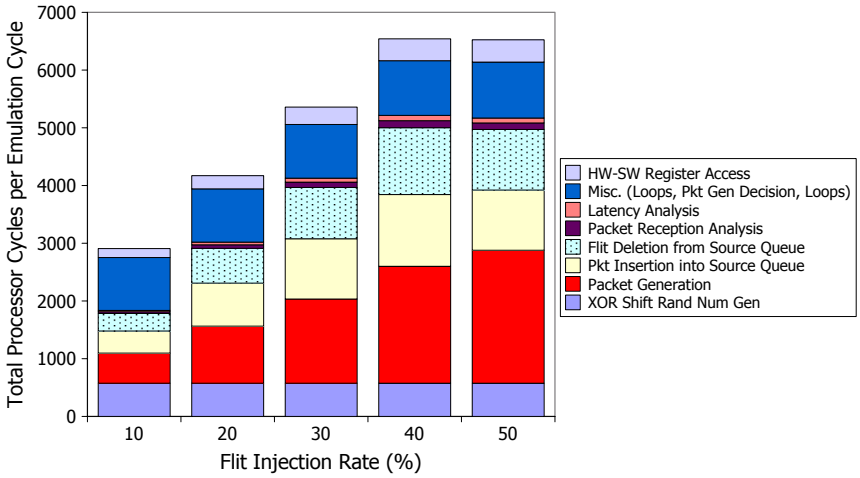


Fig. 11. Software Profiling Results (Average Processor Cycles per Emulation Cycle) after Software Optimizations

to the profiling results, henceforth, shall point to the latest profiling results shown in Figure 11.

As seen, packet generation process emerges to be the highest consumer of processor cycles in an emulation cycle, occupying about 30-35% of the processor's time in an emulation cycle. Uniform random number generation forms the single largest component of the packet generation process, consuming 20-50% of the total processor cycles for packet generations.

Among other significant contributors for packet generation include building the flit along with function calls across the traffic generation function. The packet insertion into the source queues and deletion from the source queues also consume a significant amount of the processor's time, totaling an average 35% of the total processor time in an emulation cycle. These functions' use of malloc() and free() function calls for allocating and deallocating memory respectively dominate their runtime. Traffic reception and latency analysis functions account for only 3% of the total processor cy-

cles. The processor spends about 6% of its time in an emulation cycle for interactions between the hardware components and software components through PLB register interface. Components like packet injection and ejection counters, loop constructs for calling packet generation functions per TG, per TR packet reception functions and terminating the emulation process, if–else comparators for per TG packet generation decision, etc. classified under Miscellaneous category account for approx. 15-20% of the total processor cycles per emulation cycle.

2. Software Optimizations

In order to improve emulator performance, several software based optimizations were carried out to reducing software operation latency. Compiler techniques like loop unrolling were applied on loop constructs to reduce the number of branching decisions to be made. Function inlining was also applied to reduce the branches in the assembled code but at the cost of increased code size. During profiling, it was found that building of flits involved a great deal of shift operations. Hence, hardware barrel shifter was integrated into the MicroBlaze's ALU for faster shift operations. It was also observed that function calls and return consumed considerable amount of processor cycles, and hence the number of functions called were reduced. Apart from these, the use of typecast operations for converting one datatype to another were also avoided.

Since random number generation constituted a sizable chunk of the total packet generation time, various schemes were used to find an optimal technique for generating them. The initial scheme for random numbers made use of glibc's rand() accompanied by a floating point divide operation for restricting the random numbers within a specific range. This technique had a negative impact on the emulator performance and increased the software latency (consuming approx. 90-100 cycles per random number generation). The chosen Marsaglia's XOR shift algorithm was

found to achieve 4X cycle improvement over the other schemes. We also evaluated a scheme which generated prime number of random numbers prior to start of emulation process and these number were stored and recycled throughout emulation. This technique reduced the number of random number generation cycles by almost two times for smaller injection rates as compared to XOR shift scheme. However, overhead involves loss of network injection rate accuracy and additional software memory for storing the numbers, having a direct impact on the number of outstanding flits stored across all source queues.

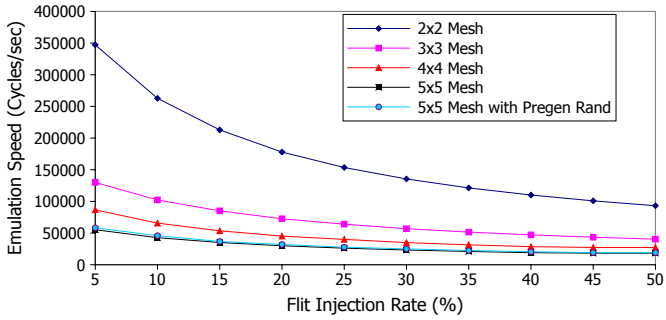
D. Emulator Performance Evaluation

This section deals with the evaluation of AcENoCs' performance for synthetic and real application based workloads under varying network dimensions and packet sizes. We compare AcENoCs' performance against OCIN_TSIM cycle accurate software simulator and ABC network's HDL simulator [35] under similar set of workloads. Emulation cycles per second is used as a metric to evaluate emulator performance.

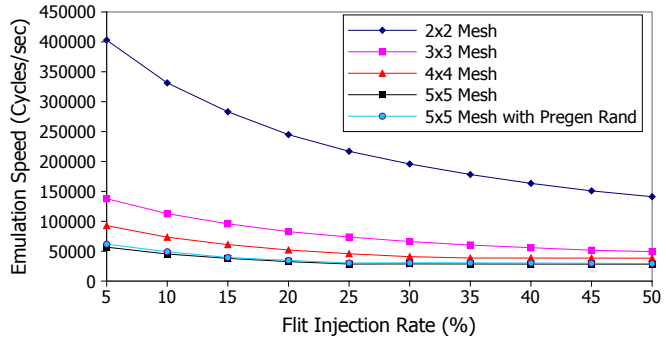
1. Evaluation under Synthetic Workloads

AcENoCs' performance was evaluated under varying network dimensions and packet sizes, when subjected to synthetic traffic patterns. Figure 12 illustrates AcENoCs' performance for varying flit injection rates under different network dimensions. Results are shown for bit complement and matrix transpose synthetic traffic patterns respectively.

In both cases, it can be seen that for a constant injection rate, the emulation speed decreases with increasing network dimensions. This is due to the fact that larger dimension network implies more number of network nodes that generate packets in



(a)



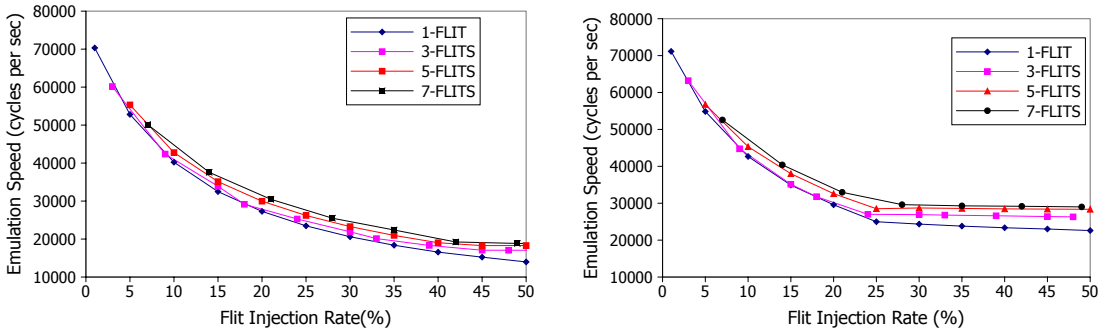
(b)

Fig. 12. Emulation Speed v/s Injection Rate for Varying Network Sizes for (a) Bit–Complement Traffic and (b) Transpose Traffic

an emulation cycle, leading to an increasing in the software operation latency per emulation cycle. Hence, more processor cycles are consumed in handling the traffic generation and reception for increased number of nodes. Emulation performance is higher for transpose traffic as compared to bit complement since nodes along one diagonal of the mesh in transpose traffic do not participate in traffic generation and reception process. However, this also implies that the time taken to complete the emulation process for transpose traffic is much higher as compared to bit complement pattern. It is also observed that for a given network, emulation speed decreases with increasing flit injection rates. Higher injection rate implies more number of

packet generations and hence more computations on the software, thereby reducing the emulation clock frequency. As seen from the plot, the emulator achieves a speedup of 5-10% when random number pregeneration technique is employed.

Next, we examine the effect of varying injection rates on the emulator performance for different flit sizes for a 5x5 mesh network under bit complement and transpose traffic patterns. Performance results are shown in Figure 13.



(a) (b)

Fig. 13. Emulation Speed v/s Injection Rate for a 5x5 Mesh Network with Varying Flit Sizes for (a) Bit-Complement Traffic and (b) Transpose Traffic

It can be observed that for a constant flit injection rate, AcENoCs' emulation speed increases as number of flits per packet is increased. This can be attributed to the fact that that as the number of flits per packet increases, the number of packets to be injected from each node into the network, to maintain the specified injection rate over a period of time, decreases. In other words, the traffic generation rate varies inversely with number of flits per packet. Less packet generation implies reduced software operation latency, and hence higher emulation performance. It can also be observed that the emulator performance tends to flatten for flit injection rates beyond 45% and 30% for bit complement and transpose traffic respectively. At this point, the

network under test is said to have reached its saturation point. The source queues tend to overflow and to avoid loosing a packet, throttling of packets is initiated.

a. Comparison with other Simulators

AcENoCs' emulation performance was compared against OCIN_TSIM software simulator running similar set of synthetic workloads. The results are shown in Figure 14. It needs to be noted that OCIN_TSIM was run on an 8–core Intel Xeon processor, with each core operating at 3.2 GHz. AcENoCs, in its available capability, runs on a much slower single core 125 MHz MicroBlaze processor.

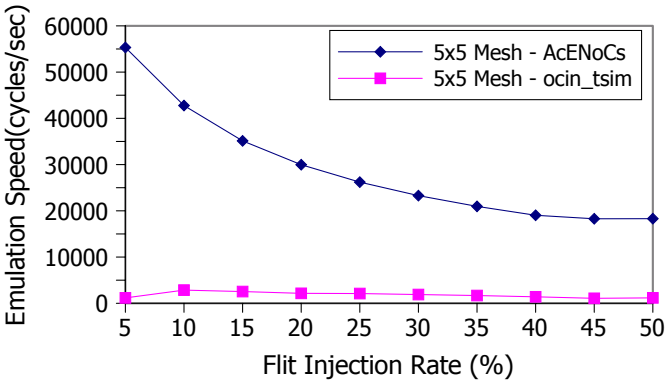


Fig. 14. AcENoCs v/s OCIN_TSIM Performance Comparison (in Cycles per Sec)

AcENoCs achieves a speedup in the range of 14-47X when compared to OCIN_TSIM, and about 10000-12000X when compared to HDL simulators like Modelsim, etc. in the non saturation operating region. AcENoCs can be made to achieve relatively better performance if the current MicroBlaze soft core processor configuration can be replaced by an embedded hard processor configuration, like PowerPC, which is capable of synthesizing at much higher frequencies.

2. Evaluation under Real Application based Workloads

AcENoCs platform is capable of executing real time traces generated by a NoC system running full load applications. For evaluating AcENoCs' performance under trace based workloads, we instantiated the dual processor based trace traffic TG environment for supporting real application traffic. AcENoCs' baseline network was evaluated with traces generated by SPEC CPU2000 [36] benchmark suite running on TRIPS OCN [37]. TRIPS OCN is a 5x5 2D mesh network, hence we could easily map the trace data onto our baseline network.

The emulator performance for trace based workloads was found to be approx. 17500 emulation cycles/sec, about 9X faster as compared to OCIN_TSIM. AcENoCs' lower performance for trace based traffic as compared to synthetic workloads can be attributed to the lower operational frequency of the dual processor environment.

E. Emulation Testcase

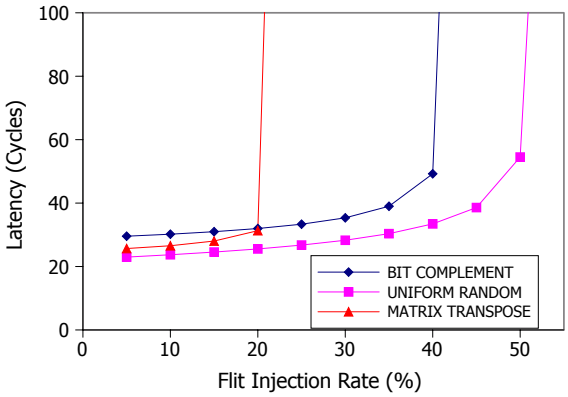


Fig. 15. Average Latency v/s Injection Rate for Network Validation

This section presents the tests that were carried out in order to verify the correct-

ness of AcENoCs emulator by observing the network parameter behavior. The network under consideration was a 5x5 2D mesh network. Average latency and throughput were measured for different synthetic traffic patterns.

Figure 15 shows the obtained average latency values against varying injection rates for different synthetic traffic patterns. The obtained curves exhibited characteristics typical of a standard 2D mesh network.

CHAPTER IX

CONCLUSIONS AND FUTURE WORK

A. Conclusions

Rapid advances in VLSI technology has led to integration of multiple heterogeneous processing cores on a single silicon slice. This has resulted in communication infrastructure becoming the bottleneck in the chip design cycle. Traditionally used buses and point-to-point links have become undesirable due to their scalability limitations, low performance, high power consumption and noise susceptibility. NoC serves as an ideal replacement for bus based schemes for handling communication requirements in a system with large number of cores.

In order to arrive at an optimal NoC architecture, fast and cycle accurate simulations must be carried out to explore the vast design space provided by on chip interconnection networks in terms of network topologies, router microarchitecture and routing algorithms. FPGA based NoC emulation schemes have been shown to overcome the shortcomings of software simulators and perform fast, cycle accurate simulations. There is a great demand for emulation schemes which offer an optimum balance between network dimensions and emulation performance. This thesis presents HW/SW codesign of AcENoCs, a novel FPGA based NoC emulator capable of fast and cycle accurate emulations, with a specific focus on the design and implementation of AcENoCs' software framework.

Two different flavors of TGs for supporting synthetic and realistic workloads, source queues/FIFOs, TRs and latency analysis modules and emulation clock generator were designed as part of AcENoCs' software architecture. Implementing traffic models in software made room for implementing a larger dimension network in hard-

ware. Controlling the emulation clock using software helped in achieving proper synchronization between the hardware and software events scheduled to occur in an emulation cycle. By dynamically allocating software source queues, specified injection rates were sustained for longer period of time, thereby modeling the congestion behavior of the network more accurately. Several emulation and software framework parameters were made user configurable through the emulation configuration file, thereby adding flexibility to the emulation platform. Support was provided for interfacing the software framework with various network configurations and router microarchitectures. Integrated and well defined HW/SW emulation flow was proposed for handling HW-SW interactions taking place in an emulation cycle. AcENoCs well-defined HW/SW framework provisions for efficient utilization of available FPGA resources and helps in achieving high emulation performance together with realization of larger dimension networks on hardware. All these features make AcENoCs an ideal platform for researchers wanting to validate their NoC designs early in the design cycle.

AcENoCs' performance was evaluated for varying network sizes and flit sizes against software simulators and HDL simulators. Since software operation latency dictated the performance of the emulator, software profiling was carried out in order to identify the software components that formed bottlenecks towards achieving high emulation performance. Based on the software profiling results, several software code based optimizations were performed, leading to improved emulator performance. Emulation tests were performed in order to validate AcENoCs platform and the plots obtained were characteristic of a standard 2D mesh networks. AcENoCs was able to achieve speedups of 10000-12000X compared to the HDL simulators and 14-47X compared to OCIN_TSIM software simulator running on a 3.2 GHz processor machine, without sacrificing cycle accuracy.

B. Future Work

With the emergence of complex SoC based design methodologies, more researchers are adapting Globally Asynchronous Locally Synchronous (GALS) based design styles for establishing intercommunication between modules operating on independent clock domains [38]. In order to explore the design space and evaluate various characteristics of such GALS based NoC designs, a robust and fast emulation platform is required. The current version of AcENoCs supports synchronous operation of router nodes, with all routers in the network operating on a common network clock. In addition to the currently implemented synchronous NoC emulation feature, AcENoCs' flexibility and versatility can be enhanced by incorporating support for emulating GALS based NoC designs, where each node in the network would operate at its own independent clock frequency.

The current AcENoCs architecture places a firm restriction on emulating larger dimension networks (greater than 5x5) due to limited FPGA hardware resources available. To overcome this limitation, AcENoCs' framework can be combined with the technique presented by Wolkotte et al. [20] by treating smaller dimension networks as a single block and executing each block in a sequential manner by storing the contents of routers and its associated links in memory.

REFERENCES

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

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

[3] L. Benini and G. De Micheli, "Networks on chips : A new SoC paradigm," *IEEE Transactions on Computers*, vol. 35, no. 1, pp. 70–78, Jan. 2002.

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

[5] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[6] Tilera Corporation, "Tile64 processor family," [Online]. Available: http://www.tilera.com/products/processors.php, Accessed Feb. 2009.

[7] M. Coppola, S. Curaba, M. Grammatikakis, G. Maruccia, and F. Papariello, "OCCN: A network-on-chip modeling and simulation framework," in *Proc. Design Automation and Test in Europe Conference*, Paris, France, Feb. 2004, pp. 174–179.

[8] 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 *Proc. 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, Newport Beach, CA, Oct. 2003, pp. 7–12.

[9] Z. Lu, R. Thid, M. Millberg, E. Nilsson, and A. Jantsch, "NNSE: Nostrum network-on-chip simulation environment," in *Proc. Swedish System-on-Chip Conference*, Stockholm, Sweden, Apr. 2005, pp. 1–4.

[10] F. Fazino, M. Palesi, and D. Patti, "Noxim: Network-on-chip simulator," [Online]. Available: http://sourceforge.net/projects/noxim, Accessed May 2010.

[11] S. Pestana, E. Rijpkema, A. Radulescu, K. Goossens, and O. Gangwal, "Cost-performance trade-offs in networks on chip: A simulation-based approach," in *Proc. Design Automation and Test in Europe Conference*, Paris, France, Feb. 2004, pp. 764–769.

[12] S. Prabhu, B. Grot, P. Gratz, and J. Hu, "Ocin_tsim - DVFS aware simulator for NoCs," in *Proc. 1st Workshop on SoC Architecture, Accelerators and Workloads (SAW-1)*, Bangalore, India, Jan. 2010, pp. 1–8.

[13] 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 *Proc. Design, Automation and Test in Europe*, Munich, Germany, Mar. 2005, pp. 1182–1187.

[14] D. A. Siguenza–Tortosa and J. Nurmi, "VHDL-based simulation environment for Proteo NoC," in *Proc. High-Level Design Validation and Test Workshop*, Cannes, France, Oct. 2002, pp. 1–6.

[15] L. Ost, A. Mello, J. Palma, F. Moraes, and N. Calazans, "MAIA - A framework

for networks on chip generation and verification," in *Proc. of IEEE Asia South Pacific Design Automation Conference*, Shanghai, China, Jan. 2005, pp. 49–52.

[16] J. Chan and S. Parameswaran, "NoCGEN: A template based reuse methodology for networks on chip architecture," in *Proc. 17th International Conference on VLSI Design*, Mumbai, India, Jan. 2004, pp. 717–720.

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

[18] 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.

[19] P. Liu, C. Xiang, X. Wang, B. Xia, Y. Liu, W. Wang, and Q. Yao, "A NoC emulation/verification framework," in *Proc. Sixth International Conference on Information Technology: New Generations*, Las Vegas, NV, Apr. 2009, pp. 859–864.

[20] P.Wolkotte, P. Holzenspies, and G. Smit, "Fast, accurate and detailed NoC simulations," in *Proc. First International Symposium on Networks-on-Chip*, Princeton, NJ, May 2007, pp. 323–332.

[21] S. Lotlikar, V. Pai, and P. Gratz, "AcENoCs: A flexible HW/SW platform for FPGA accelerated NoC emulation," in *Review 24th International Conference on VLSI Design*, Chennai, India, Jan. 2011.

[22] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, L. Benini, and G. De Micheli, "NoC synthesis flow for customized domain specific multiprocessor systems-on-

chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 2, pp. 113-129, Feb. 2005.

[23] NIRGAM, "A Simulator for NoC Interconnect Routing and Application Modeling," [Online]. Available: http://nirgam.ecs.soton.ac.uk, Accessed May 2010.

[24] Worm_sim, "A Cycle Accurate Simulator for Networks-On-Chip," [Online]. Available: http://www.ece.cmu.edu/~sld/software/worm_sim.php, Accessed May 2010.

[25] A. Jalabert, S. Murali, L. Benini, and G. De Micheli, "xpipesCompiler: A tool for instantiating application specific networks on chip," in *Proc. Design, Automation and Test in Europe Conference*, Paris, France, Feb. 2004, pp. 884–889.

[26] G. Schelle and D. Grunwald, "Onchip interconnect exploration for multicore processors utilizing FPGAs," in *Proc. 2nd Workshop on Architecture Research using FPGA Platforms*, Austin, Texas, Feb. 2006, pp. 1–4.

[27] P. Del Valle, D. Atienza, I. Magan, J. Flores, E. Perez, J. Mendias, L. Benini, and G. Micheli, "A complete multi-processor system-on-chip FPGA-based emulation framework," in *Proc. IFIP International Conference on Very Large Scale Integration*, Nice, France, Oct. 2006, pp. 140–145.

[28] Emulation and Verification Engineering, "ZeBu-XL System Emulator," [Online]. Available: http://university.eve-team.com/files/ZeBu-XL.pdf, Accessed Jun. 2010.

[29] Xilinx Inc., "Xilinx university program XUPV5-LX110T development system," [Online]. Available: http://www.xilinx.com/univ/xupv5-lx110t.htm, Accessed Dec. 2009.

[30] W. Dally, "Virtual-channel flow control," in *Proc. 17th Annual International Symposium on Computer Architecture*, Seattle, WA, May 1990, pp. 60–68.

[31] W. Dally and C. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, pp. 547–553, May 1987.

[32] K. Bolding, M. Fulgham, and L. Snyder, "The case for chaotic adaptive routing," *IEEE Transactions on Computers*, vol. 46, pp. 1281–1292, Dec. 1997.

[33] G. Marsaglia, "Xorshift RNGs," *Journal of Statistical Software*, vol. 8, pp. 16, 2003.

[34] Xilinx Inc., *Embedded System Tools Reference Manual* [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/est_rm.pdf, Accessed Dec. 2009.

[35] T. Jain, P. Gratz, A. Sprintson, and G. Choi, "Asynchronous bypass channel routers: Improving performance for DVFS and GALS NoCs," in *Proc. 4th ACM/IEEE International Symposium on Networks-On-Chip*, Grenoble, France, May 2010.

[36] J. Henning, "SPEC CPU2000: Measuring CPU performance in the new millennium," *Computer*, vol. 33, pp. 28–35, Jul. 2000.

[37] 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 *Proc. 1st ACM/IEEE International Symposium on Networks-on-Chip*, Princeton, NJ, May 2007, pp. 7–17.

[38] D. M. Chapiro, "Globally-asynchronous locally-synchronous systems," Ph.D. thesis, Stanford University, Stanford, CA, 1984.

APPENDIX A

TRAFFIC GENERATION FUNCTIONS

AcENoCs' Traffic Generator function is made up of several sub-functions. This information may be useful for future users of this platform, to integrate more additional features into the software framework. Briefly discussed below are sub-functions called by the TG function :

1. Main.c : This is the main function. For synthetic TGs, it calls the traffic generator function based on uniform random injection process. The uniform random injection process involves calling a uniform random number generator function, for each TG in every emulation cycle, and comparing it with the user specified flit injection rate to decide whether a packet needs to be generated by that particular TG. For trace based TGs, main function running on MB2 (MicroBlaze 2) reads the trace data from the compact flash card and stores it into a Shared Block RAM (SHBRAM) structure. Main function running on MB1 (MicroBlaze 1) reads the data from the SHBRAM and decides in which cycle the packet needs to be generated and injected into the network.

2. Traffic_Generate.c : This is the main traffic generator function for both kinds of TGs. It injects flits into the router's source queue and reads out the flits from each source queue for injection into network. It also initiates packet throttling when the network experiences congestion.

3. Generate_Pkts.c : Calls the "build flit" function. Supplies all the flit fields necessary for building the flits. Generates Packet Id field for each packet. Instantiates the two-level latency structure for storing the packet injection time.

For Synthetic TGs, it also stores the destination node address for each source node, as calculated according to the selected traffic pattern, along with the source router's output port number.

4. Build_Flits.c : Builds the head, body and tail flits and returns them to the caller function. The number of body flits vary based on number of flits per packet parameter.

5. Queue_FIFO.c : Contains the queue/FIFO data structure for storing the flits. The FIFO is realized using singly-linked list data structure. Contains functions required for inserting and deleting the flits from the source queue. Flits are added to the rear of the FIFO, as indicated by the rear pointer and are read out from the front of the FIFO, as indicated by the Front pointer.

6. Marsaglia_Rand.c : This function is called only for synthetic TGs. It contains code for uniform random number generation scheme based on Marsaglia's XOR shift algorithm [33].

7. config_dest.c : This function is called only for synthetic TGs. It generates the destination node address for packets generated by each TG, based on the type of selected synthetic traffic pattern.

8. current_node_op_port.c : Generates the source router's output port (N,S,E,W or L) for packets. This field is generated based on knowledge of the destination node coordinates and selected routing algorithm.

.

VITA

Vinayak Pai was born in Mysore, India. He received his Bachelor of Engineering degree in electronics and communications from Visvesvaraya Technological University, Belgaum, India in June 2005. For the next three years, he was employed with Wipro Technologies, Bangalore, working for their semiconductor and systems business unit as an Field Programmable Gate Array (FPGA) Design Engineer. He joined the graduate program in computer engineering at Texas A&M University (TAMU) in August 2008 and graduated with his M.S. in December 2010. At TAMU, his research was focused on Network-on-Chip (NoC) emulation and FPGA acceleration techniques. During the summer of 2009, he held an internship with Texas Instruments in Bangalore, India, working on integration verification of mixed signal circuits. He can be reached at the following address :

c/o Dr. Paul V. Gratz,
Computer Engineering Group
Department of Electrical and Computer Engineering
Texas A&M University
College Station, Texas - 77843-3259
Email : vinayak.pai123@gmail.com

The typist for this thesis was Vinayak Pai.