1-1-2007

Fuse-N: Framework for unified simulation environment for network-on-chip

Ashwini Raina
University of Nevada, Las Vegas

Follow this and additional works at: <https://digitalscholarship.unlv.edu/rtds>

Repository Citation

Raina, Ashwini, "Fuse-N: Framework for unified simulation environment for network-on-chip" (2007). *UNLV Retrospective Theses & Dissertations*. 2184.
<http://dx.doi.org/10.25669/0dnn-o18e>

This Thesis is protected by copyright and/or related rights. It has been brought to you by Digital Scholarship@UNLV with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.

FUSE-N: FRAMEWORK FOR UNIFIED SIMULATION ENVIRONMENT
FOR NETWORK-ON-CHIP

by

Ashwini Raina

Bachelor of Engineering
Sardar Patel College of Engineering
University of Mumbai
2004

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree in Engineering
Department of Electrical Engineering
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
August 2007

UMI Number: 1448416

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 1448416

Copyright 2007 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Thesis Approval
The Graduate College
University of Nevada, Las Vegas

July 27, 2007

The Thesis prepared by

Ashwini Raina

Entitled

fuse-N: Framework for Unified Simulation

Environment for Network-on-Chip

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering

M. V. V.
Examination Committee Chair

Allen S.
Dean of the Graduate College

J. P. L.
Examination Committee Member

E. Saberi
Examination Committee Member

L. P. Generali
Graduate College Faculty Representative

ABSTRACT

fuse-N: Framework for Unified Simulation Environment for Network-on-Chip

by

Ashwini Raina

Dr. Venkatesan Muthukumar, Examination Committee Chair
Associate Professor of Electrical and Computer Engineering
University of Nevada, Las Vegas

Steady advancements in semiconductor technology over the past few decades have marked incipience of Multi-Processor System-on-Chip (MPSoCs). Owing to the inability of traditional bus-based communication system to scale well with improving microchip technologies, researchers have proposed Network-on-Chip (NoC) as the on-chip communication model. Current uni-processor centric modeling methodology does not address the new design challenges introduced by MPSoCs, thus calling for efficient simulation frameworks capable of capturing the interplay between the application, the architecture, and the network. Addressing these new challenges requires a framework that assists the designer at different abstraction levels of system design.

This thesis concentrates on developing a framework for unified simulation environment for NoCs (fuse-N) which simplifies the design space exploration for NoCs by offering a comprehensive simulation support. The framework synthesizes the network infrastructure and the communication model and optimizes application mapping for design constraints. The proposed framework is a hardware-software co-design implementation using SystemC 2.1 and C++. Simulation results show the architectural,

network and resource allocation behavior and highlight the quantitative relationships between various design choices.

Also, a novel off-line non-preemptive static Traffic Aware Scheduling (TAS) policy is proposed for hard NoC platforms. The proposed scheduling policy maps the application onto the NoC architecture keeping track of the network traffic, which is generated with every resource and communication path allocation. TAS has been evaluated for various design metrics such as application completion time, resource utilization and task throughput. Simulation results show significant improvements over traditional approaches.

TABLE OF CONTENTS

ABSTRACT	iii
LIST OF FIGURES	vii
LIST OF TABLES	viii
CHAPTER 1 INTRODUCTION	1
1.1 Importance of Network-On-Chip.....	1
1.2 Design issues in NoC	3
1.3 Thesis Overview	5
1.3.1 fuse-N.....	5
1.3.2 Traffic Aware Scheduling policy in hard NoCs	6
1.4 Thesis Organization	6
CHAPTER 3 NETWORK ON CHIP – AN OVERVIEW	7
2.1 NoC by Example.....	7
2.2 Characteristics of NoC architecture.....	8
2.2.1 NoC Topology	8
2.2.2 Switching Technique	9
2.2.3 Routing Technique.....	10
2.2.4 NoC Simulation Environment Framework	11
CHAPTER 3 fuse-N: FRAMEWORK FOR UNIFIED SIMULATION ENVIRONMENT FOR NETWORK-ON-CHIP	15
3.1 Simulation Flow.....	15
3.1.1 Simulation parameter space	15
3.1.2 Application Characterization	18
3.1.3 Architecture Modeling	19
3.1.4 Network Infrastructure Modeling	19
3.1.5 Application Mapping Optimization	20
3.1.6 NoC Execution Platform.....	21
3.1.7 Simulation Result Space	21
3.2 fuse-N Implementation	22
CHAPTER 4 TAS: TRAFFIC AWARE SCHEDULING FOR HARD NOC	26
4.1 Background and Scope	26
4.2 Problem Statement.....	27
4.3 Algorithm Description	27
4.3.1 Longest-Path-Maximum-Delay Deadline Allocation.....	27

4.3.2 Scheduling and Optimization.....	29
CHAPTER 5 SIMULATIONS AND RESULTS	32
5.1 Benchmark description	32
5.2 Simulation Scenarios	33
5.3 Simulation results and explanation	34
CHAPTER 6 CONCLUSIONS AND FUTURE WORK.....	42
6.1 Conclusions.....	42
6.2 Future Work.....	43
REFERENCES	44
APPENDIX I SIMULATION RESULTS FOR EARLIEST DEADLINE FIRST BASED SIMULATED ANNEALING APPROACH.....	50
VITA.....	71

LIST OF FIGURES

Figure 1.	A relative evolution of wire and gate delays (source: 2003 ITRS, 2003).....	2
Figure 2.	Regular 4x4 tile-based NoC architecture.....	8
Figure 3.	Unified simulation methodology for NoC	12
Figure 4.	fuse-N framework simulation flow diagram.....	16
Figure 5.	Sample pre-configured arguments XML file.....	17
Figure 6.	(a) A simple Application Task Graph with directed edges (b) 3x3 2-D tile NoC architecture	18
Figure 7.	A sample 3x3 2-D tile network dot file	20
Figure 8.	Flow of data from source PE to destination PE, over layered communication architecture.....	22
Figure 9.	Block diagram representation of fuse-N framework implementation	24
Figure 10.	Pseudo code for Traffic Aware Scheduling policy	31
Figure 11.	Architecture Evaluations of the proposed NoC framework – Execution Time.	34
Figure 12.	Architecture Evaluations of the proposed NOC framework – Throughput.	35
Figure 13.	Architecture Evaluations of the proposed NOC framework – Utilization...	35
Figure 14.	Architectural Evaluation of a 3 x 3 and 4 x 4 TILE Topology	37
Figure 15.	Comparision of Scheduling Algorithm.....	39
Figure 16.	Evaluation of network parameters for various topologies	41

LIST OF TABLES

Table 1	Longest-Path-Maximum-Delay deadlines	28
Table 2	Characteristics of Standard Task Graph (STG)	32
Table 3	Architectural Evaluation of a 3 x 3 TILE Topology	36
Table 4	Architectural Evaluation of a 4 x 4 TILE Topology	36
Table 5	Comparison of Scheduling Algorithms	38
Table 6	Evaluation of Network Parameters	40

ACKNOWLEDGEMENTS

This thesis is dedicated to my parents and my sister, together who have crafted my life so beautifully. I am deeply indebted to Dr. Venki, my *guru*, for trusting my abilities and considering me for apprenticeship. He has significantly influenced my professional as well as personal life in more ways than one.

My research work has greatly benefited from brainstorming sessions with Shruti Patil, who I consider to be the *visiting PhD student* of lab B348. Positive attitudes such as Pavan Singaraju, Sourabh Mookerjea and Gopinath Balakrishnan have also been very contagious. Special thanks to Kunal Metkar for being my Google on C++ programming and design. I am also thankful to Naveen Chintalcheruvu, Vikram Mylaram and Shankar Neelakrishnan for making up a great lab B348. Imran Dalwai, Pradeep Nambisan and Venka Palaniappan have been a great source of stimulating discussions on life in general.

CHAPTER 1

INTRODUCTION

1.1 Importance of Network-On-Chip

Steady advancements in semiconductor technology over the past few decades have enabled the chip manufacturers to increase the amount of functionality on a single chip. This unprecedented growth puts forth many design issues such as: functionality, testability, wire delay, power management, signal integrity, packaging and management of physical limits that need to be considered [1]. To address these problems, researchers have proposed component-based design methodologies, capable of distributing the chip complexity, thus marking the incipience of Multiprocessor System-on-Chip (MPSoC). MPSoCs integrate together a large number of processing elements (PEs) and embedded memory connected over complex communication architectures. These PEs can be of the type a) general purpose and specialized processors such as digital signal processors (DSP) and VLIW cores, or b) embedded hardware such as FPGA or application specific intellectual property (IP) [2,3].

Recent technological advances in VLSI have greatly reduced the computation cost as compared to on-chip communication. The shared bus (single or multi-bus) communication architecture, currently used for MPSoCs, does not scale well with improving microchip technologies, thus posing the following design issues.

- Global wire lengths do not scale well with the shrinking transistor size, and as local processing cycle times decrease, the time spent on global communication, relative to the time spent on local processing increases drastically. As per the projections by international technology roadmap for semiconductors (ITRS) [4] relative delay for local wires, global wires and logic gates of the near future will show the following trend (Figure 1).

Figure 1. A relative evolution of wire and gate delays (source: 2003 ITRS, 2003)

- Global synchronization is becoming a dominant factor in MPSoC design as clock skew is claiming an ever larger relative part of the total cycle time.
- Over the past decade, design complexity has increased 50 times [5]. With increasing amount of PEs available for chip designers, the design efforts do not scale linearly with the system complexity.

Due to the above factors, the trend is towards sub-division of a complex single processor system into manageable reusable PEs and the differentiation of the local and

global communication. Many researchers have proposed the concept of scalable packet switching networks as an alternative approach to interconnect PEs in MPSoCs [1,6,7,8,9]. Implementing NoCs as the communication architecture for MPSoCs over the traditional bus based systems promises the following improvements:

- NoC based systems no longer function on a global clock and hence global synchronization is not an issue. An efficient design style called Globally Asynchronous Locally Synchronous (GALS) [10] promises reduction in power consumption in clock of high performance systems. The GALS design style can be very efficiently implemented using NoC. This further indicates that system can fully exploit the parallelism of computation.
- Such systems do not suffer from the deep submicron (DSM) effects, and are expected to exhibit predictable electrical properties.
- An NoC based approach brings great potential in the reuse of the communication network, since the switches/routers, the interconnects and the lower-level communication protocols can be designed, optimized and verified once and reused in a large number of products.
- With the complexity and number of the PEs, growing every year, it is essential for the underlying on-chip communication architecture to be scalable and robust. Unlike the bus architecture, NoC offers an elegant solution to this problem.

1.2 Design issues in NoC

An efficient NoC design methodology is based upon several key design choices, such as: network topology selection, good routing policy and efficient application to NoC

mapping. A formal categorization of the NoC design issues is given in [11] as is summarized below:

- Network infrastructure selection: Network design primarily consists of topology selection which plays an important role in minimizing network latency and power consumption, thus improving the overall throughput [12-21]. Another critical network design element is channel buffering which accounts for both latency and router area. Also, NoC are expected to connect multiple heterogeneous cores which pose unique challenges with respect to variable traffic loads (channel width problem) and irregular silicon area (floorplanning problem) which will force designers to delve into more unique and application specific network design.
- Communication model: The underlying network infrastructure is merely a backbone for communication and requires an efficient inter-node interaction model to capture the dynamism in the network and improve network performance [22-25]. The primary element of the NoC communication model is the routing policy. Two major concerns in selecting a particular routing strategy are implementation complexity and the overall performance. Another problem related to routing is the switching technique used inside the router for transferring packets along the switches.
- Application mapping optimization: Another aspect of efficient NoC design is the mapping and scheduling of both computation and communication over the network infrastructure, while optimizing certain design metrics as latency, throughput etc.

It can be seen from the above discussion, that an efficient application specific NoC design space is large and there exists a need for a unified simulation framework which can address all these above problems in a combined manner.

1.3 Thesis Overview

This thesis concentrates on developing a unified simulation framework for NoCs which simplifies the design space exploration for NoCs by offering a comprehensive simulation support. The aim of the thesis includes: 1) to design and implement a framework for unified simulation environment for NoC 2) to conceive an efficient traffic aware scheduling policy for off-line static scheduling for applications on hard NoCs.

1.3.1 fuse-N

To capture the trade-offs of the NoC design parameters in a unified manner, a simulation framework, designed using C++ and SystemC 2.1 is proposed, wherein the effects of all parameters could be simulated in-part or combined. The framework synthesizes the network infrastructure and the communication model and optimizes application mapping as per the design parameters. Quantitative relationships between various design choices can be accurately observed and revised design criteria can be applied and re-evaluated.

Specifically, the framework allows the designer to execute an application, in the form of application task graph, over a customizable NoC, with certain set of PEs, routing policy, router buffering, switching technique and the scheduling policy. The simulation framework generates simulation log files containing specific details such as PE utilization and throughput, router utilization and throughput, router buffer efficiency, application

execution time and network link utilization. The flexibility of the framework allows the designer to change and optimize the design as per the key design metrics.

1.3.2 Traffic Aware Scheduling policy in hard NoCs

A novel off-line non-preemptive static traffic aware scheduling policy is proposed for hard (both computation and communication components completely fixed) NoC platforms. The proposed scheduling policy maps the application task graph onto the underlying network architecture and schedules both computation and communication transactions of the tasks. While doing so, the scheduling algorithm keeps track of the network traffic it generates with every PE and communication path allocation, and takes it into consideration while scheduling the subsequent tasks. Results show an average 14.8% decrease in overall execution time, 18% improvement in utilization and around 24% increase in throughput, thus confirming our hypothesis

1.4 Thesis Organization

The remainder of this thesis is organized as follows: In Chapter 2, concepts and terminologies of NoCs are introduced. Chapter 3 discusses the propose fuse-N framework. Traffic Aware Scheduling policy is explained in Chapter 4. Simulations in proposed framework are given in Chapter 5. Finally, Chapter 6 contains a summary of the important results of this thesis.

CHAPTER 2

NETWORK ON CHIP – AN OVERVIEW

In this chapter, the basics of NoC are discussed. First a component based view is presented, that introduces the basic building blocks of a typical NoC. Delving further, system level architectural issues relevant to NoC-based designs are explained. Using the established foundations, a range of existing NoC frameworks will be discussed and the need for a novel unified approach towards NoC simulation is demonstrated.

2.1 NoC by Example

Figure 2 shows a regular 4x4 tile-based NoC, in which a set of tiles are connected in a 2D mesh topology. Each tile in the grid can be a general-purpose processor, a DSP, an embedded memory etc. In a simplified perspective, the NoC contains the following fundamental components:

- Processing Elements – Each core modules in the tile can be a general-purpose processor, a DSP, and embedded memory etc.
- Network adapters – These are the interfaces by which the PEs connect to the NoC, decoupling the computation (PEs) from communication (the network)
- Routers – These nodes route the data and control flits or phits (packets with smaller sizes) as per the routing strategy.

- Router-to-Router Links – These consist of the logical or physical channels connecting the router nodes

Figure 2. Regular 4x4 tile-based NoC architecture

2.2 Characteristics of NoC architecture

2.2.1 NoC Topology

An important aspect of NoC design is to determine a suitable topology for a particular application. NoC topologies are broadly classified as direct topologies, where each router is connected to a single PE, and indirect topologies where a set of PEs are connected to a router. Some examples of regular direct topologies are mesh based interconnect architectures like 2D mesh, tori, cube [27,28,29], honeycomb design [30], octagon structure [31] etc. Some tree-based indirect topologies such as binary tree, fat tree [32] and butterfly [33] have also been proven efficient for certain set of applications.

2.2.2 Switching Technique

Switching techniques have been a well researched area in traditional data networks for a long time. Other efficient switching techniques currently employed in computer networks are explained in detail in [34,35,36,37]. There are mainly four switching techniques which are considered promising for NoCs.

- Store-and-forward: Commonly known as packet switching, the entire packet is stored in the intermediate node buffer before forwarding it to a selected neighboring node based on the information within the packet header. The CLICHÉ [27] is an example of a store-and-forward NoC.
- Circuit switching: Circuit switching involves the establishment of a physical circuit between source and destination nodes and reserved until the transport of data is complete. During the transmission phase, all the packets belonging to that stream are transmitted over this reserved circuit of intermediate nodes.
- Wormhole: This technique combines the packet switching with the data streaming quality of circuit switching, thus achieving minimal packet latency. In wormhole switching [39], a packet is divided into flow control digits (flits) and then these flits are routed through the network one after another, in a pipelined fashion. At the end of stream transmission, the circuit connection is terminated by transmitting a special packet. The average waiting time in the router buffer queues is not of the whole packet but of the individual flit.
- Virtual cut-through (VCT): Virtual cut-through technique was proposed to minimize the latency issues with the store-and-forward technique [38]. In this switching technique, the forwarding router waits for a guarantee from the next

node in the path that it will accept the entire packet. This handshaking allows the forwarding router to transmit the intermediate flits as it receives them, thus reducing the network latency.

2.2.3 Routing Technique

Switching mechanism is only concerned with the transport of data, while a routing technique consists of the logic or intelligence behind that path of the data transport. Efficient routing schemes in parallel and distributed computing areas have been researched for a long time. In general, routing algorithms for NoC can be classified into two categories – deterministic routing and adaptive routing [22]. If the behavior or the routing algorithm is independent of the network conditions, then it belongs to the deterministic class of routing algorithms.

Dimension ordered routing or XY (or YX) routing is deterministic routing algorithm wherein a packet is first forwarded in the X dimension and then along the Y dimension, restricting the maximum number of allowed turns to one [40]. An extension to this algorithm has been proposed in [41,42], which imposes certain turn rules on the XY routing algorithm. Hot potato or deflection routing is another deterministic routing algorithm that forwards the packet towards the path with the lowest delay [43]. Every packet has preferred outputs along which it wants to leave the router, and when possible a packet is sent along one of these outputs.

In adaptive routing, the path that a packet chooses depends on the source and destination address as well on the dynamic traffic conditions. A contention aware hot potato routing scheme is proposed in [44]. A variation to the model developed in [41,42] is given in [45] where an odd-even adaptive routing algorithm for meshes is proposed.

Jingcao et al. [46], propose a routing technique which switches between deterministic and adaptive as per the network congestion. There has been an in-depth survey on some efficient routing algorithms [47,48]. Comparison of various routing algorithms over different topologies is discussed in [49,50,51].

2.2.4 NoC Simulation Environment Framework

The current uni-processor centric modeling methodology does not capture the new design challenges introduced by SoC with an increased number of PEs, running concurrent application programs [52]. Future SoCs are envisioned to integrate even greater amount of functionality, in terms of PEs, thus calling for efficient simulation frameworks understanding the interplay between the application, the PE architecture, and the NoC. Traditional simulation environments focus more towards the micro architectural details and analysis of the design. Such computation centric frameworks face limitations when the design at hand is built over communication-centric methodology.

Addressing these new challenges will require a framework to shift the level of abstraction up to the network level, which will enable the designer to better understand the trade-offs of different NoC design aspects such as topology, switching, routing, buffering, scheduling etc. The framework should be able to capture the desired system metrics in a unified way (Figure 3) and at the same time let the designer customize it according to the needs.

A number of EDA research groups are studying different aspects of NoC design, some of which are discussed below.

Figure 3. Unified simulation methodology for NoC

a) NetChip: xpipes, xpipescompiler and SUNMAP

NetChip is a NoC synthesis environment primarily composed of two tools namely SUNMAP [17] and the xpipescompiler [53]. SUNMAP tool is used to generate the network topology and the xpipescompiler is responsible for mapping cores on the network topology, at the same time optimizing it. This approach utilizes a traffic generator capable of creating communication patterns specific application domains. Due to lack of PE modeling support and over-simplified statistical communication traffic traces, this tool limits itself merely to a network topology selection and mapping utility.

b) NNSE: Nostrum NoC Simulation Environment

NNSE [54] is a SystemC based NoC simulation environment initially used for the Nostrum [55] NoC. Over revisions it has been equipped with a GUI and allows designers to (1) configure a network with respect to topology, flow control and routing algorithm etc.; (2) configure various regular and application specific traffic patterns; (3) evaluate the network with the traffic patterns in terms of latency and throughput. NNSE as a framework lacks a complete top to bottom approach which is essential for simulating

NoCs. Lack of a direct application simulation support and absence of exhaustive PE modeling capability restrict the scope of NNSE just to a network simulation environment.

c) ARTS Modeling Framework

ARTS [56] is a system-level framework to model networked multi-processor systems-on-chip (MPSoC) and evaluate the cross-layer causality between the application, the operating system (OS) and the platform architecture. It captures the impact of dynamic and unpredictable OS behavior on processor, memory and communication performance. ARTS features the ability to examine the impact of dynamic and unpredictable OS behavior on processor, memory and communication performance and thus can help the designer in better understanding the processor-memory or processor-communication correlations.

d) StepNP: A System-Level Exploration Platform for Network Processors

StepNP [57] is a System-Level Exploration Platform for Network Processing built in SystemC. It enables the creation of multi-processor architectures with models of interconnects (functional channels, NoCs), processors (simple RISC), memories and coprocessors. The network wrappers communicate with each other using SystemC Open Core Protocol (SOCP) [58]. Dependence of StepNP on SOCP alone as a communication protocol makes it inflexible in accommodating novel communication architecture, which are not directly supported by the protocol.

A lot of research teams have also focused on the usage of certain network simulators like OPNET [59,60], ns-2[61], OMNET [62], that offers the users with the abstraction of concurrent communication and flexible communication protocol definition. However, the

appropriateness of such simulators has been argued as they were not designed specifically to model both computation and communication [63].

CHAPTER 3

fuse-N: FRAMEWORK FOR UNIFIED SIMULATION ENVIRONMENT FOR NETWORK-ON-CHIP

In this chapter, the fuse-N framework and implementation are discussed. First, the overall simulation flow under fuse-N framework is explained using a component diagram. The different input/output formats related to the proposed framework are also introduced. Various framework aspects are presented along with the implementation details of the fuse-N framework. Finally, applications of the proposed framework are discussed.

3.1 Simulation Flow

fuse-N is a top-down approach of NoC design space exploration. Figure 4 shows the overall simulation flow/component diagram under the fuse-N framework.

3.1.1 Simulation parameter space

Simulation parameter space includes the initial arguments which define the simulation environment of fuse-N. Simulation parameters can be either provided through the fuse-N graphical user interface or through a pre-configured arguments file. Exhaustive listing of the initial arguments is given below:

Figure 4. fuse-N framework simulation flow diagram

- **Application Task Graph** – This parameter specifies the location of the application task graph file name as the argument. Application task graph file can be in the standard task graph (STG) [64] or Task Graph For Free (TGFF) [65] format.
- **Network Topology** – Defines the network topology of the NoC such as 2D Tile, 2D Torus etc.
- **Router Buffer Size** – Specifies the NoC router input buffer size. Buffer size value is specified in the form of number of packets the router input buffer can accommodate.
- **Switching Technique** – The switching techniques such as packet-switching, circuit-switching, wormhole switching, etc. are defined by this argument.
- **Routing Technique** – Defines the network routing policy (eg: XY routing, odd-even routing, etc.)

- Scheduling Policy – Task scheduling policies such as Earliest Deadline First (EDF), Least Slack Time (LST) and Traffic Aware Scheduling (TAS) are selected by this argument.
- Number of PEs – Specifies the total number of processing elements present in the design.
- Different Types of PEs – Total number of different type of PEs are present in the design is defined by this argument
- PE modeling parameters – Different PE specifications such as operation, execution time etc. are represented in the form of a tuple <"operation":num_of_PEs:"execution_time">. For example, a tuple <sum,sub,mul : 4 : 3,4,8 > specifies that there are 4 processors capable of performing a sum, sub and mul operation with execution times as 3 ns, 4 ns and 8 ns respectively.
- Pre-configured arguments file – It is also possible to provide the location of a preconfigured XML file to the framework. Figure 5 depicts a sample arguments XML file.

```

<?xml version="1.0" ?>
<ARG app_filePath="/home/TG/rand0000.stg" topology = "torus" sch_policy = "edf">
<TILE-0 id="0" pe_opr="opr1,opr2" inst_per_sec="4,9" r_buff_size="15" switchTech="vct" r_chWidth="4"/>
<TILE-0 id="1" pe_opr="opr3,opr5" inst_per_sec="6,10" r_buff_size="10" switchTech="vct" r_chWidth="4"/>
<TILE-0 id="2" pe_opr="opr1,opr4" inst_per_sec="4,8" r_buff_size="12" switchTech="vct" r_chWidth="4"/>
.
.
.
.
<TILE-N id="N" pe_opr="opr2,opr3" inst_per_sec="9,6" r_buff_size="20" switchTech="vct" r_chWidth="4"/>
</ARG>

```

Figure 5. Sample pre-configured arguments XML file

3.1.2 Application Characterization

Application characterization is a pre-processing component where certain characteristics of the Application Task Graph (ATG) are determined and set as per the underlying NoC architecture. An ATG is a generic directed graph $G(V, E)$ where the vertex represents a task and an edge or a link represents the communication latency (Figure 6(a)).

Figure 6. (a) A simple Application Task Graph with directed edges (b) 3x3 2-D tile NoC architecture

The communication latency for the tasks executing over NoC architecture is unknown, until they are mapped onto a respective processors. At this stage, the deadlines of various tasks as per the initial simulation arguments are calculated. Consider the ATG shown in Figure 6(a), for which we need to calculate the task deadlines for task t_1 , t_2 , t_3 and t_4 (t_0 and t_5 being dummy tasks) when executed over a 2D tile topology as shown in Figure 6(b).

Given the maximum input and output buffering in the routers, in the worst case a task on tile-1 might have to send a packet to a task on tile-9. In this case the packets sent from tile-0 to tile-9 can expect a maximum delay of h hops where h is $2 * (\text{sqrt}(\text{num_of_PEs}) - 1)$ (4 in our case). From this information along with the knowledge of execution times of the tasks, the deadlines for respective tasks can be calculated.

3.1.3 Architecture Modeling

Another important aspect of fuse-N simulation framework is the flexible architecture modeling component. As per the initial simulation arguments, the properties of the PEs present in the NoC are set. For example, a PE can be capable of multiple operations and each operation might have different execution time. Given the set of operations and their corresponding execution times, the PEs in the NoC can be modeled accordingly. With the increase in the complexity of the design, the modeling arguments to the architecture modeling component can be increased in order to suit the requirements of the design. Advanced initial arguments such as PE's power consumption, area, etc can be easily added and the framework can simulate and the overall effect is determined for design parameters. The modeled architecture configuration file is represented as an XML format for later reuse.

3.1.4 Network Infrastructure Modeling

Efficient network infrastructure plays a pivotal role in a NoC design. Network infrastructure modeling component is mainly concerned with the communication model definition and topology synthesis. Various characteristics of the router elements are described as per the initial simulation arguments. Communication model for router elements is modeled primarily based on the switching technique and the routing policy.

Network connectivity between different router elements is described by the topology argument. All neighbors for a particular router element are configured as per their topological location, thus restricting the degree of communication of a router element. The topological configuration is represented in a Dot [66] file format, which is a customizable directed graph layout methodology, shown in Figure 7.

```
graph G {
    label = "Architecture view\n";
    fontsize=20;
    size = "4,4";
    node [shape=box];
    {rank = same; "PE2"; "PE0"; "PE4"; }
    {rank = same; "PE3"; "PE7"; "PE1"; }
    {rank = same; "PE6"; "PE8"; "PE5"; }
    "PE2" -- "PE0";
    "PE2" -- "PE3";
    "PE0" -- "PE4";
    "PE0" -- "PE7";
    "PE4" -- "PE1";
    "PE3" -- "PE7";
    "PE3" -- "PE6";
    "PE7" -- "PE1";
    "PE7" -- "PE8";
    "PE1" -- "PE5";
    "PE6" -- "PE8";
    "PE8" -- "PE5";
}
```

Figure 7. A sample 3x3 2-D tile network dot file

3.1.5 Application Mapping Optimization

Once the architecture modeling and network infrastructure modeling is complete, the application tasks are ready to be mapped and scheduled on the target NoC architecture. Application mapping aims at mapping an application onto the NoC platform, while optimizing certain design parameters such as performance, power, etc. This component arbitrarily generates a mapping between the ATG and the target NoC architecture. The

tasks are scheduled on the respective PEs as per a scheduling policy such as EDF or LST and are submitted for execution to the NoC Execution Platform (discussed in Sub-section 3.1.6) component. Optimizations of mapping and scheduling are achieved by recursively revising the application mapping process, until all the design constraints are satisfied.

3.1.6 NoC Execution Platform

NoC execution platform is the actual hardware platform, modeled in SystemC v2.1 [26], which provides an interoperable modeling platform, enabling the development and exchange of very fast system-level C++ models. The PEs and the router elements are modeled based upon architecture and network modeling configuration outputs. The communication model of the NoC is modeled as a packet-switched network with the required network algorithms implemented for the data flow. The NoC execution platform component functions interactively, based on task mapping and scheduling information, repeatedly calculating the global simulation values along with the required heuristic values. These values are fed back in the application mapping optimization component and the solution is refined.

3.1.7 Simulation Result Space

This component is responsible for collecting the global simulation results and storing/displaying the results in a cognitive manner, bringing out the existent interplay between different design choices. Simulation result space component primarily deals with monitoring the simulation and writing simulation log files, plain data files and comparative data files.

3.2 fuse-N Implementation

Owing to the complexity of the on-chip communication design, researchers have proposed a layered design methodology to allow the designer to explore model behavior and communication at different level of abstractions [67]. Micro-network stack paradigm, as shown in Fig. 8, is an adaptation of the protocol stack [59], which offers abstraction levels to the NoC design flow.

Figure 8. Flow of data from source PE to destination PE, over layered communication architecture

The fuse-N implementation is based on the micro-network stack paradigm. The software layer consists of the system (i.e. the PEs) and the applications (i.e. the tasks/processes that execute on them). At this layer most of the network implementation details are hidden. Major design challenges of this layer are part of the SoC research. The architecture and control layer mainly deals with the network architecture and control protocols of the on-chip communication model. It is envisioned to perform the functions such as error detection and correction, packetized data transmission (customized by the

choice of switching and routing algorithms) and disassembly and assembly of messages. Finally, the physical layer consists of reliable and low power wiring challenges of the NoC architecture.

As a NoC has several clock domains, discrete event model is the most suitable model of execution. New hardware description languages such as SystemC and SystemVerilog [68] (a library of C++), make simulations at a broad range of abstraction levels readily available, and thus supports full range of abstractions needed in a modular NoC-based design. C++ language is chosen to model the initial simulation environment and SystemC v2.1 is used for describing the NoC hardware platform. The Graphical User Interface (GUI) for the fuse-N implementation is designed using C++/Tk [69], which is a complete C++ interface to the Tk GUI toolkit.

The GUI of the implementation interacts with the user to collect the initial simulation arguments. These arguments are stored by the GUI module in the *userInput* structure object. The main module next instantiates the *Tg_Operation* class and executes the *TgParser()* member function over a Standard Task Graph (STG) or a Task Graphs For Free (TGFF) Application Task Graph file. *Tg_Operation* object further instantiates the *Task* class, creates a data store of objects of this class and updates the object member variables as per the values generated from the *TgParser()* member function.

The main module next instantiates the *Topology* class and executes the *setTopology()* member function which operates on the simulation argument data and the task object data, and synthesizes the topology of the network. *Topology* class object stores the topology configuration into an object data store of *Arch* class type. In general, *Arch* class

objects represent every tile in the NoC and contain their configuration data of the embedded PE and the neighbor connectivity information.

Figure 9. Block diagram representation of fuse-N framework implementation

After completion of topology synthesis, the *Scheduler* class is instantiated and which executes the *setDeadline()* and *setSchedule()* member functions, thus mapping and scheduling the tasks as per a user entered scheduling algorithm. Both these member functions operate on the initial simulation arguments and the task and tile object data.

They further update the member variables of task objects by setting up the scheduled PE IDs and the schedule time.

A number of hardware library modules such as *Router*, *ProcElem* (PE) and *Packet* are developed using SystemC and are instantiated by the main program. The main program next starts the SystemC simulation. Hardware simulation is triggered by a *Dispatcher* module which evaluates the next release task as per the task object data and dispatch the task in the NoC execution platform. The simulation progresses for the stipulated amount of time and all the simulation details are captured into simulation log files. A simulation log parser is implemented in python language [70] which extracts the important simulation details such as number of tasks processed by every PE, number of packets sent by every router, buffer utilization over time, link utilization over time, etc and displays the information in a tabular format.

CHAPTER 4

TAS: TRAFFIC AWARE SCHEDULING FOR HARD NOC

In this chapter, a novel Traffic Aware Scheduling algorithm (TAS) is proposed for hard NoCs platforms. We focus on hard NoCs and reason out the need and scope of an efficient scheduling algorithm. The scheduling problem is formulated and a novel scheduling algorithm is proposed and explained in the form of TAS.

4.1 Background and Scope

Hard NoC platforms, Figure 2, have their architectures completely fixed. They don't offer real flexibility for architectural customization as both computation and communication components are pre-designed. In general, the computation sequence is depicted in the form of Application Task Graphs (ATG), which depicts control and data dependencies between the interacting computations or tasks. Inter-processor communication aware scheduling contains two main aspects: mapping processors to computation nodes, and scheduling communication on the links. The former is also known as the "mapping problem" and the latter as "communication scheduling" problem [11]. The problem of scheduling an ATG over multiple processors and minimizing a desired objective function is NP-complete in the strong sense, even if infinite number of processors are available [71].

Although scheduling algorithms has been a traditional research subject, most previous work has treated mapping and communication scheduling in a disjoint manner [72, 73]. Some proposed techniques like [74], have approached the scheduling problem in a combined way, but they still assume fixed IPC latency regardless of the distance between the interacting PEs. The unprecedented growth in MPSoC complexity and the envisioned scale of future NoCs, easily make such an assumption inaccurate.

4.2 Problem Statement

Given a target NoC execution platform and an ATG, a static, non-preemptive, viable schedule needs to be generated that minimizes the earliest completion time taking into consideration the network traffic dynamics. The above problem statement unfolds into following specification:

- For every task t_i in ATG, determine the PE, p_i on which it should be scheduled and the time slot T_i when it should be executed.
- Determine the exact time slot, for every communication event between a child task and its parent tasks.

4.3 Algorithm Description

4.3.1 Longest-Path-Maximum-Delay Deadline Allocation

For any task t_i , the mapping of the successor onto a PE is not known till the mapping stage. Thus, for deadline calculation it is important to estimate the communication cost over the longest path possible on the target NoC architecture. Moreover, the state of the routers i.e. the configuration of the input and output buffers, makes the interPE

communication dynamic for every successive mapping and scheduling. In order to account for the worst-case scenario, deadlines are allocated as per the maximum delay possible on any path, hence the name “longest-path-maximum-delay deadline allocation”. For example, Figure 6 depicts a target NoC architecture and a simple ATG to be executed on that architecture. Considering XY routing strategy, it is clearly evident that the longest path from any two PEs is h hops where

$$h = 2 \times (\sqrt{\text{num_of_PEs}} - 1)$$

Also, given the maximum size of input buffer size (b_{in}), output buffer size (b_{out}) and the router processing speed (R_{ps}), the maximum delay max_delay along the longest path can be easily calculated as:

$$\text{max_delay} = [(h + 1) \times b_{in} + h \times b_{out}] \times R_{ps}$$

For example, let the tasks t_1 , t_2 , t_3 and t_4 have execution times as 30, 70, 90 and 60 respectively. Also, assume that all the PEs are capable of processing t_1 , t_2 , t_3 and t_4 . The longest-path-maximum-delay deadlines for all the tasks are given in Table 1.

Table 1 Longest-Path-Maximum-Delay deadlines

Task ID	Deadline Calculation
t_1	$t_{1_dl} = t_{0_dl} + t_{0_exectime} + \text{max_delay} = 0 + 0 + 12 = \mathbf{12ns}$
t_2	$t_{2_dl} = t_{0_dl} + t_{0_exectime} + \text{max_delay} = 0 + 0 + 12 = \mathbf{12ns}$
t_3	$t_{3_dl} = \max(t_{1_dl} + t_{1_exectime}, t_{2_dl} + t_{2_exectime}) + \text{max_delay} = \max(30, 70) + 12 = \mathbf{92ns}$
t_4	$t_{4_dl} = t_{0_dl} + t_{0_exectime} + \text{max_delay} = 0 + 0 + 12 = \mathbf{12ns}$

In Table 1, t_{i_dl} represents the deadline of task t_i and $t_{i_execTime}$ represents the execution time of task t_i . Also assume $b_{in} = 20$, $b_{out} = 5$, $R_{ps} = 0.1\text{ns}$, $h = 4$, $t_{0_dl} = 0$ and $t_{0_exectime} = 0$.

4.3.2 Scheduling and Optimization

In order to apply the concept of simulated annealing [75] to the problem of scheduling, the algorithm begins with an initial solution, i.e. a feasible schedule, by mapping each task to a randomly chosen PE and repeatedly improving the solution through simulated annealing iterations. Each iteration is repeated L (temperature length) times, where

$$L = \frac{(\text{num_of_PEs})^2}{(\text{num_of_PE_types})}$$

num_of_PE_types represents the number of different type of PEs present in the architecture. At the end of L sub-iterations, the simulation temperature T is reduced by a factor of cooling rate (0.95). Following steps are carried out for each iteration:

- i) A list of ready tasks is generated by selecting tasks whose parent tasks have already scheduled. From the ready task list, a task t_{sch} is randomly selected.
- ii) Let $PST(i,k)$ and $PET(i,k)$ be the schedule time and *execution time* respectively of the parent task P_k , of task t_{sch} . Moreover, let maxCommDelay be the maximum delay cost on the longest path as defined Section 4.3.1. For this task a *release time window* (RTW) is generated, which is defined by the time interval:

$$[\max(PST(i,k)+PET(i,k))-\text{maxCommDelay}, \max(PST(i,k)+PET(i,k))+\text{maxCommDelay}]$$

The lower bound in this interval represents the earliest release time of t_{sch} in the network and the upper bound signifies the latest time by which it should be released.

- iii) From this RTW, a time instance is chosen at random and t_{sch} is scheduled for that time instance. This random selection is repeated over several sub-iterations for every task.
- iv) For every iteration an objective function is calculated which is defined as:

$$F_{obj} = \min[\text{earliest completion time} + \Sigma(\text{task_wait_time})]$$

With every iteration, TAS aims at minimizing the objective function.

- v) If current scheduling scheme does not violate any deadline, then it is a valid solution. A valid solution is accepted without any reservations. If the change in schedule time produces a schedule that violates the deadline of any single task, then the schedule is considered invalid, but may still be accepted. Given the *best_cost* (best simulation cost till current iteration), *curr_cost*(simulation cost of current iteration) and *sim_temp* (simulation temperature), the probability of accepting such a bad schedule is:

$$P = e^{\frac{(best_cost - curr_cost)}{sim_temp}}$$

If the schedule is not accepted, then the process is repeated until a schedule is obtained which is accepted by the iteration. The accepted solution becomes the initial solution the algorithm progresses into the next iteration.

- vi) After each iteration, T is cooled down by multiplying it with the cooling rate, taking the simulation to next cycle.

The process is continued till the T becomes 0. At this point, the best obtained schedule is chosen as the final solution. Figure 10 illustrates the pseudo code for TAS:

```

for each task in ATG do{
    task.setDeadline(); //set the longest-path-maximum-delay deadline
    task.mapToPE(); // initial solution
}
while simulation temp > cooling_threshold do{
    for temperature length{
        for task_sub_iteration_length{
            while each task is assigned a release time{
                genRTL(); // generates the Release Time Window
                t_sch = sel_rand(RTL); // rand task from RTL
                t_sch.genRTW(); //generate release time window
                t_sch.rel_time = sel_rand(RTW); //set rel time
            }
            sc_start(time); //init and run systemC simulation
            calcObjFunc(); //calculating objective function value
            probFunc(); // accepts or rejects a solution
        }
        for each task in ATG do{
            task.mapToPE(); // new mapping solution
        }
    }
    calc_simulation_temp(); // calculate new simulation temperature
}

```

Figure 10. Pseudo code for Traffic Aware Scheduling policy

CHAPTER 5

SIMULATIONS AND RESULTS

5.1 Benchmark description

For evaluation of the proposed framework and scheduling algorithm, a random set of benchmarks as well as a set of application specific benchmarks were chosen. Both types of these benchmark application graphs, also known as Standard Task Graph (STG) [77] have the following characteristics to choose from, as shown in Table 2.

Table 2 Characteristics of Standard Task Graph (STG)

Benchmark name	Num. of tasks	Num. of comm. transactions	Max. predecessors	Min. predecessors	Avg. predecessors	Max. PE execution time	Min. PE execution time	Avg. PE execution time	Parallelism
rand0000.stg	50	96/1225	6	0	1.92	10	1	5.24	4.763
rand0001.stg	50	22/1225	13	0	4.5	10	1	5.96	3.348
rand0002.stg	50	164/1225	9	0	3.28	10	1	4.84	3.408
rand0000.stg	100	570/4950	15	0	5.70	10	1	5.81	6.180
rand0001.stg	100	366/4950	11	0	3.66	10	1	5.33	6.58
rand0002.stg	100	908/4950	22	0	9.08	10	1	5.75	3.285
robot.stg	88	131	3	0	1.488	111	1	28.215	4.363
sparse.stg	96	67	6	0	0.697	34	0	20.166	15.868

5.2 Simulation Scenarios

To demonstrate the capabilities of the proposed fuse-N framework, we performed several experiments on random and application specific task graph sets and evaluated its effectiveness over following design abstractions:

- Architectural – Architecture based evaluation emphasize on the effect of different NoC architecture on the overall system performance. NoC architecture is modified by changing the *number of PEs*, *types of PEs*, *PE operation*, *PE execution time* and *topology*. The effect of such architecture changes is measured in terms of *earliest completion time* of the application, *average utilization* of PEs and *average throughput* of PEs.
- Scheduling – Evaluation of Scheduling tests the effectiveness of different scheduling algorithms over a fixed NOC architecture. Performance of as *earliest deadline first* (EDF) and *least slack first* (LST) scheduling algorithms is compared with the proposed TAS algorithm. Performance metrics for scheduling scenarios are same as that of architecture based scenarios.
- Network – This evaluation captures the effect of network design parameters such as *topology*, *router design* and *router input/output buffering*. Network evaluation metrics such as *average router throughput*, *average link throughput* and *average buffer utilization* are calculated under this scenario.

5.3 Simulation results and explanation

Architectural Evaluation:

For an architecture evaluation, we consider a 3x3 tile NoC with *EDF* scheduling and XY routing. Two types of PEs are present in the NOC – No. of PEs of Type-I and Type-II are five and four respectively. Figure 11 shows the average completion times of the two types of PEs for different application task graphs for architecture evaluation.

Figure 11. Architecture Evaluations of the proposed NoC framework – Execution Time.

Figure 12, illustrates the average throughput (per 100 ns) for different type of PEs in the NoC. It can be seen that architecture evaluation on rand-50 and sparse benchmarks results in disparity between the PE average throughput.

Figure 12. Architecture Evaluations of the proposed NOC framework – Throughput

The average utilization for different types of PEs is illustrated in Figure 13.

Figure 13. Architecture Evaluations of the proposed NOC framework – Utilization

The architectural evaluations are also performed based on the number of PEs and the topology. A 3 x 3 tile topology with 9 PEs is compared with 4 x 4 tile topology with 16 PEs. The evaluation metrics for architectural evaluation include: earliest completion time (ECT), processor utilization (U_p) and task (T_t) and processor throughput (T_p). Table 3 and Table 4 illustrate the results obtained from the 3 x 3 tile topology and the 4 x 4 tile topology respectively. Figure 14, illustrates the architectural evaluation comparison of the two topologies.

Table 3 Architectural Evaluation of a 3 x 3 TILE Topology

STG	No. of Tasks	ECT	Time Used	Utilization	Tt	Tp
random0000.stg	50	122.6	326	29.65	0.41	0.05
random0001.stg	50	232.4	365	17.58	0.22	0.02
random0002.stg	50	221.4	312	15.79	0.23	0.03
random10000.stg	100	404.5	730	19.93	0.25	0.03
random10001.stg	100	340.8	680	21.76	0.29	0.03
random10002.stg	100	502.3	712	16.03	0.2	0.03
robot.stg	88	2220.3	4552	19.72	0.045	0.004
sparse.stg	96	382.3	2030	46.77	0.26	0.022

Table 4 Architectural Evaluation of a 4 x 4 TILE Topology

STG	No. of Tasks	ECT	Time Used	Utilization	Tt	Tp
random0000.stg	50	120.6	326	16.9	0.42	0.03
random0001.stg	50	178.8	365	12.76	0.28	0.02
random0002.stg	50	179.3	312	10.88	0.28	0.02
random10000.stg	100	288.7	730	15.37	0.35	0.02
random10001.stg	100	260.3	680	16.33	0.38	0.02
random10002.stg	100	425.6	712	10.46	0.24	0.02
robot.stg	88	1832.5	4522	15.42	0.06	0.003
sparse.stg	96	289	2030	43.9	0.35	0.02

Figure 14. Architectural Evaluation of a 3 x 3 and 4 x 4 TILE Topology

Scheduling Evaluation:

The scheduling evaluation is performed by comparing the 1) EDF Scheduling Algorithm, 2) LST Scheduling Algorithm and 3) the proposed Traffic Aware Scheduling (TAS). The results are tabulated in Table 5 and Figure 15 graphically expresses the results. The scheduling algorithms are evaluated for a) earliest completion time (ECT), b) processor utilization (U_p), c) task throughput (T_t) and d) processor throughput (T_p). In all three algorithms, Simulated Annealing is used to heuristically determine the optimal schedule.

Table 5 Comparison of Scheduling Algorithms
EDF-SA Scheduling Algorithm:

STG	No. of Tasks	ECT	Tt	Time Used	Utilization	Tp
random0000.stg	50	88	0.57	326	41.16	0.063
random0001.stg	50	164.8	0.33	365	24.61	0.034
random0002.stg	50	114.9	0.44	312	30.17	0.048
random10000.stg	100	231.6	0.43	730	35.02	0.048
random10001.stg	100	256.7	0.39	680	29.43	0.043
random10002.stg	100	332.6	0.3	712	23.79	0.033
robot.stg	88	1662.5	0.053	4522	30.235	0.006
sparse.stg	96	384.9	0.25	2030	58.655	0.028

LST-SA Scheduling Algorithm:

STG	No. of Tasks	ECT	Tt	Time Used	Utilization	Tp
random0000.stg	50	93.6	0.53	326	36.7	0.059
random0001.stg	50	149.6	0.33	365	27.11	0.037
random0002.stg	50	111.6	0.45	312	31.06	0.05
random10000.stg	100	270.6	0.37	710	29.15	0.041
random10001.stg	100	200.6	0.5	680	37.67	0.055
random10002.stg	100	350.8	0.29	712	22.55	0.032
robot.stg	88	1577.5	0.056	4522	31.85	0.006
sparse.stg	96	404.65	0.235	2030	55.755	0.0265

TAS Algorithm:

STG	No. of Tasks	ECT	Tt	Time Used	Utilization	Tp
random0000.stg	50	80.3	0.62	324	44.83	0.069
random0001.stg	50	141.5	0.35	365	28.66	0.069
random0002.stg	50	114.7	0.44	312	30.22	0.048
random10000.stg	100	173.6	0.58	710	45.44	0.064
random10001.stg	100	166.8	0.6	680	45.3	0.067
random10002.stg	100	251.6	0.4	712	31.44	0.044
robot.stg	88	1288.6	0.068	4522	38.39	0.008
sparse.stg	96	380.6	0.25	2030	59.26	0.028

Figure 15. Comparison of Scheduling Algorithm

Network Evaluation:

The network evaluation is evaluated for the 3 x 3 and 4 x4 tile topology and the performance of NoC routers and channels are tabulated based on number of packets handled by the router, router throughput (Tr) and channel throughput (Tc). Also the maximum buffer usage during simulation is determined. Table 6 and Figure 16 illustrate the comparison of network parameters for different topologies.

Table 6 Evaluation of Network Parameters

3 x 3 Tile Topology:

STG	No. of Tasks	ECT	No. of pkts	Tr	Bmax	Tc
random0000.stg	50	122.6	446	0.4	8	0.15
random0001.stg	50	232.4	896	0.43	12	0.16
random0002.stg	50	221.4	718	0.36	10	0.13
random10000.stg	100	404.5	2296	0.63	18	0.23
random10001.stg	100	340.8	1468	0.48	12	0.17
random10002.stg	100	502.3	3406	0.75	23	0.29
robot.stg	88	2220.3	640	0.03	9	0.012
sparse.stg	96	382.3	404	0.1	4	0.038

4 x 4 Tile Topology:

STG	No. of Tasks	ECT	No. of pkts	Tr	Bmax	Tc
random0000.stg	50	120.6	576	0.3	8	0.09
random0001.stg	50	178.8	1278	0.45	12	0.14
random0002.stg	50	179.3	1086	0.38	10	0.12
random10000.stg	100	288.7	3096	0.67	18	0.21
random10001.stg	100	260.3	2044	0.49	12	0.17
random10002.stg	100	425.6	4676	0.69	23	0.24
robot.stg	88	1832.5	890	0	9	0.008
sparse.stg	96	289	634	0	4	0.041

Figure 16. Evaluation of network parameters for various topologies

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In order to simplify the design space exploration for NoC, a framework for unified simulation environment for NoC (fuse-N) was developed. The framework was based on a top-down multi-component design, where each component handled an important aspect of the NoC design. On completion of the framework component design, fuse-N was implemented in SystemC and C++ using the hardware-software co-design methodology. Also, a novel off-line non-preemptive static Traffic Aware Scheduling (TAS) policy is proposed for hard NoC platforms.

For evaluation purpose, a series of random as well as application specific benchmarks were run over the fuse-N implementation. Simulation results show the the architectural, network and resource allocation behavior and highlight the quantitative relationships between various design choices. TAS was also evaluated for various design metrics such as application completion time, resource utilization and task throughput. Simulation results show 14.8 % decrease in earliest completion time, 18% increase in utilization and around 24% gain in task throughput.

6.2 Future Work

As of now fuse-N implementation has a restricted set of topologies, routing algorithms, switching techniques and scheduling algorithms. It is essential to improve its capabilities which would further assist the designer at different abstraction levels of system design. Moreover, current implementation of fuse-N consists of behavioral PEs and it would be advisable to implement more complex PEs for better and accurate results.

REFERENCES

- [1] L. Benini and G. De Micheli, "Networks on Chips: a New SOC Paradigm", IEEE Computer, Jan. 2002, pp.70-78.
- [2] Marcello Coppola , Stephane Curaba , Miltos D. Grammatikakis , Riccardo Locatelli , Giuseppe Maruccia , Francesco Papariello, OCCN: a NoC modeling framework for design exploration, Journal of Systems Architecture: the EUROMICRO Journal, v.50 n.2-3, p.129-163, February 2004.
- [3] J. Hu, R. Marculescu; "Energy-Aware Mapping for Tile-based NOC Architectures Under Performance Constraints", Proceedings of ASP-Design Automation Conference, Jan. 2003, pp. 233-239.
- [4] International technology roadmap for semiconductors (ITRS) 2001. Tech. rep., International Technology Roadmap for Semiconductors.
- [5] The importance of sockets in SoC design. White paper downloadable from <http://www.ocpip.org>.
- [6] P. Guerrier and A. Greiner, "A Generic Architecture for on-Chip Packet-Switched Interconnections", DATE'2000, IEEE Press, 2000. pp.250-256.
- [7] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks", DAC'2001, ACM Press, 2001.pp.684-689.
- [8] Agarwal, A. 1999. The Oxygen project - Raw computation. Scientific American, 44-47.
- [9] Jantsch, A. and Tenhunen, H. 2003. Networks on Chip. Kluwer Academic Publishers.
- [10] A. Hemani , T. Meincke , S. Kumar , A. Postula , T. Olsson , P. Nilsson , J. Oberg , P. Ellervee , D. Lundqvist, Lowering power consumption in clock by using globally asynchronous locally synchronous design style, Proceedings of the 36th ACM/IEEE conference on Design automation, p.873-878, June 21-25, 1999, New Orleans, Louisiana, United States.

- [11] Umit Y. Ogras, Jingcao Hu, Radu Marculescu. "Key Research Problems in NoC Design: A Holistic Perspective". Proc. CODES+ISSS, Jersey City, NJ, Sept. 2005, 69-74, September, 2005.
- [12] J. Hu, R. Marculescu. Energy- and performance-aware mapping for regular NoC architectures. IEEE Trans. on CAD of Integrated Circuits and Systems, 24(4), April 2005.
- [13] S. Murali, G. De Micheli. Bandwidth-constrained mapping of cores onto NoC architectures. In Proc. DATE, 2004.
- [14] W. Hung, et. al. Thermal-aware IP virtualization and placement for Networks-on-Chip architecture. In Proc. ICCD, 2004.
- [15] G. Ascia et. al. Multi-objective mapping for mesh-based NoC architectures. In Proc. CODES, 2004.
- [16] A. Jalabert, et. al. xpipesCompiler: A tool for instantiating application specific Networks on Chip. In Proc. DATE, 2004.
- [17] S. Murali, G. De Micheli. SUNMAP: A tool for automatic topology selection and generation for NoCs. In Proc. DAC, 2004.
- [18] M. Kreutz, et. al. Communication architectures for System-On-Chip. Symposium on Integrated Circuits and Systems Design, 2001.
- [19] A. Pinto, et. al. Efficient synthesis of networks on chip. In Proc. ICCD, Oct. 2003.
- [20] K. Srinivasan, et. al. Linear programming based techniques for synthesis of Network-on-Chip architectures. In Proc. ICCD, 2004.
- [21] U. Y. Ogras, R. Marculescu. Energy- and performance- driven customized architecture synthesis using a decomposition approach. In Proc. DATE, 2005.
- [22] J. Duato, et. al. Interconnection Networks: An Engineering Approach. Morgan Kaufmann, 2002.
- [23] C. J. Glass, L. M. Ni. The turn model for adaptive routing. In Proc. ISCA, May 1992.
- [24] J. Hu, R. Marculescu. DyAD-Smart routing for Networks-on-Chip. In Proc. DAC, June 2004.
- [25] L. Shang et. al. PowerHerd: Dynamically satisfying peak power constraints in interconnection networks. In Proc. Intl. Symp. on Supercomputing, June 2003.

- [26] Open SystemC Initiative. SystemC. <http://systemc.org>, December 2004.
- [27] Kumar, S., Jantsch, A., Soininen, J.-P., Forsell, M., Millberg, M., Oberg, J., Tiensyrja, K., Hemani, A., "A network on chip architecture and design methodology," Proc. Symposium on VLSI, pp. 117–124, April 2002.
- [28] W. J. Dally, B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks", Proceedings of DAC 2001, pp. 683-689, Las Vegas, Nevada, USA, June 18-22, 2001.
- [29] Millberg, M., Nilsson, E., Thid, R., and Jantsch, A. 2004. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network-on-chip. In Proceedings of Design, Automation and Testing in Europe Conference (DATE). IEEE, 890–895.
- [30] A. Hemani, et al, "Network on a chip: an architecture for billion transistor era," Proc. of the IEEE NorChip Conf., Nov. 2000.
- [31] Karim, F., Nguyen, A., and Dey, S. 2002. An interconnect architecture for networking systems on chips. IEEE Micro 22, 36–45.
- [32] P. Guerrier, A. Greiner, "A generic architecture for on-chip packetswitched interconnections", Proceedings of Design, Automation and Test in Europe Conference and Exhibition 2000, pp. 250 -256.
- [33] Pande, P. P., Grecu, C., Ivanov, A., and Saleh, R. 2003. Design of a switch for network-on-chip applications. IEEE International Symposium on Circuits and Systems (ISCAS) 5, 217–220.
- [34] A. Tanenbaum, *Computer Networks*, Prentice Hall, fourth edition, 2002.
- [35] Forouzan, B. A., Coombs, C. A., and Fegan, S. C. 2000 *Data Communications and Networking 2nd Edition*. 2nd. McGraw-Hill Higher Education.
- [36] Kurose James, Ross Keith, "Computer Networking" Pearson Higher Education, 2002.
- [37] Bjerregaard, T. and Mahadevan, S. 2006. A survey of research and practices of Network-on-chip. *ACM Comput. Surv.* 38, 1 (Jun. 2006).
- [38] P. Kermani and L. Kleinrock. Virtual cut-through: a new computer communication switching technique. In *Computer Networks*, volume 3, pages 267{286, Sept. 1979.
- [39] W. J. Dally and C. L. Seitz. The torus routing chip. *Distributed Computing*, 1(3):187{196, 1986.

- [40] J. Duato, S. Yalamanchili, L. Ni, Interconnection Networks, an Engineering Approach, IEEE Computer Society Press, 1997.
- [41] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In 25 Years ISCA: Retrospectives and Reprint, pages 441 {450, 1998.
- [42] J.Wu; "A deterministic fault-tolerant and deadlock-free routing protocol in 2-D meshes based on odd-even turn model", Proceedings of the 16th international conference on Supercomputing, 2002, pp. 67-76.
- [43] J. T. Brassil, "Deflection routing in certain regular networks," Ph.D. dissertation, Univ. California at San Diego, 1991.
- [44] E. Nilsson; M. Millberg, J. Oberg, A. Jantsch, "Load Distribution with the Proximity Congestion Awareness in a Networks on Chip", Proceodings of Design Automation and Test in Europe, March 2003, pp. 1126-1127.
- [45] G.-M. Chiu. The odd-even turn model for adaptive routing. IEEE Tran. on Parallel and Distributed Systems, 11(7):729 {738, July 2000.
- [46] Jingcao Hu , Radu Marculescu, DyAD: smart routing for networks-on-chip, Proceedings of the 41st annual conference on Design automation, June 07-11, 2004, San Diego, CA, USA.
- [47] Terry Tao Ye , Luca Benini , Giovanni De Micheli, Packetization and routing analysis of on-chip multiprocessor networks, Journal of Systems Architecture: the EUROMICRO Journal, v.50 n.2-3, p.81-104, February 2004.
- [48] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. IEEE Tran. on Computers, 26:62 {76, Feb. 1993.
- [49] Harmanci,M., Escudero, N., Leblebici,Y., and Ienne, P. 2005. Quantitative modelling and comparison of communication schemes to guarantee quality-of-service in networks-on-chip. In International Symposium on Circuits and Systems (ISCAS). IEEE, 1782–1785.
- [50] De Mello, A. V., Ost, L. C., Moraes, F. G., and Calazans, N. L. V. 2004. Evaluation of routing algorithms on mesh based nocs. Tech. rep., Faculdade de Informatica PUCRS - Brazil. May.
- [51] Neeb, C., Thul, M., Wehn, N., 2005. Network-on-chip-centric approach to interleaving in high throughput channel decoders. In International Symposium on Circuits and Systems (ISCAS). IEEE, 1766–1769.
- [52] Xinping Zhu. "Software Tools for Modeling and Simulation of On-Chip Communication Architectures". PhD thesis, Princeton University, June, 2005.

- [53] A. Jalabert, S. Murali, L. Benini and G. De Micheli, xpipescompiler: A tool for instantiating application specific networks on chip, In the proceedings of 2004 Design Automation and Test in Europe Conference (DATE), 2004.
- [54] Zhonghai Lu, Rikard Thid, Mikael Millberg, Erland Nilsson, and Axel Jantsch. NNSE: Nostrum network-on-chip simulation environment. In Swedish System-on-Chip Conference (SSoCC'03), April 2005.
- [55] M. Millberg, E. Nilsson, R. Thid and A. Jantsch, Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip, In Proceedings of 2004 Design Automation and Test in Europe Conference (DATE), 2004.
- [56] Mahadevan, S., Storgaard. M., and Madsen, J. "ARTS: A System-Level Framework for Modeling MPSoC Components and Analysis of their Causality." 13th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), Atlanta USA. IEEE, Sept. 2005: 480-483.
- [57] P. Paulin, C. Pilkinton and E. Bensoudane, StepNP: A system-level exploration platform for network processors", IEEE Design and Test of Computers, 2002.
- [58] OCP-IP Association. OCP 2.0 specification. <http://www.ocpip.org>, December 4th, 2004.
- [59] X. Chang, Network simulation with OPNET, In proceedings of the 31st Winter Simulation Conference (WSC), pages 307-314, ACM Press, 1999.
- [60] OPNET Technologies Inc., OPNET Modeler, http://www.opnet.com/products/modeler/opnet_modeler.pdf, December 2004
- [61] K. Fall and K. Varadhan, editors. The ns Manual (formerly ns Notes and Documentation). The VINT Project, UC Berkeley, LBL, USC/ISI, Xerox PARC, 2000.
- [62] A. Varga, OMNET++ User Manual version 2.3. <http://www.omnetpp.org>, December 2004.
- [63] T. Lv, J. Xu, W. Wolf, I.B.Ozer, J. Henkel and S.T. Chandradhar, A methodology for architectural design of multimedia multiprocessor SoCs, IEEE Design & Test Computers, 22(1):18-25, 2005.
- [64] V. A. F. Almeida, I. M. M. Vasconcelos, J. N. C. Áraabe and D. A. Menascé, "Using Random Task Graphs to Investigate the Potential Benefits of Heterogeneity in Parallel Systems", Proc. Supercomputing '92, pp. 683-691 (1992).

- [65] Robert P. Dick , David L. Rhodes , Wayne Wolf, TGFF: task graphs for free, Proceedings of the 6th international workshop on Hardware/software codesign, p.97-101, March 15-18, 1998, Seattle, Washington, United States.
- [66] E.R. Gansner, E. Koutsofios, S.C. North, and K.P. Vo. A technique for drawing directed graphs. IEEE-TSE, March 1993. <http://hoagland.org/Dot.html>.
- [67] Luca Benini , Giovanni De Micheli, Powering networks on chips: energy-efficient and reliable interconnect design for SoCs, Proceedings of the 14th international symposium on Systems synthesis, September 30-October 03, 2001, Montréal, P.Q., Canada.
- [68] Fitzpatrick, T. 2004. System verilog for VHDL users. In Proceedings of Design, Automation and Testing in Europe Conference (DATE). IEEE Computer Society, 21334.
- [69] Maciej Soprezak, <http://cpptk.sourceforge.net>
- [70] Python language website, <http://www.python.org>
- [71] V. Sarkar, Partitioning and Scheduling Parallel Programs for Multiprocessors. Cambridge, MA: M.I.T. Press, 1989.
- [72] S. W. Bollinger and S. F. Midkiff, "Processor and link assignment in multicomputers using simulated annealing," in 1988 ICPP Proc., vol. I . Aug. 1988, pp. 1-7.
- [73] R. P. Bianchini Jr. and J. P. Shen, "Interprocessor traffic scheduling algorithm for multiple-processor networks," IEEE Trans. Comput., vol. C-36. no. 4. pp. 396-409, Apr. 1987.
- [74] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection constrained heterogeneous processor architectures. IEEE Tran. on Parallel and Distributed Systems, 4(2):175-187, Feb. 1993.
- [75] S. Kirkpatrick and C. D. Gelatt and M. P. Vecchi, Optimization by Simulated Annealing, Science, Vol 220, Number 4598, pages 671-680, 1983.
- [76] V. A. F. Almeida, I. M. M. Vasconcelos, J. N. C. Áraabe and D. A. Menascé, "Using Random Task Graphs to Investigate the Potential Benefits of Heterogeneity in Parallel Systems", Proc. Supercomputing '92, pp. 683-691 (1992).

APPENDIX I

SIMULATION RESULTS FOR EARLIEST DEADLINE FIRST BASED SIMULATED ANNEALING APPROACH

PARAMETERS		
NumOfPE		9
NumOfPETypes		2
Type Ratio		5:4
Topology		tile
Buffering		15
Scheduling policy		EDF
Routing		XY
Task Graph		rand0000.stg
Number of Tasks		50
Application Execution Time		88
Overall TP		0.568

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
0	A	4	21	12.743	0.0243
3	A	5	30	18.204	0.0303
5	A	6	39	23.665	0.0364
6	A	8	57	34.587	0.0485
8	A	6	44	26.699	0.0364
1	B	7	54	32.767	0.0425
2	B	3	24	14.563	0.0182
4	B	5	42	25.485	0.0303
7	B	6	54	32.767	0.0364
		50	365	24.609	0.0337

Type	TasksPerType	Time Used	Avg Util	Avg Tput	ECT
A	29	191	23.180	0.035	164.8
B	21	174	26.396	0.032	151.7

PARAMETERS		
NumOfPE		9
NumOfPETypes		2
Type Ratio		5:4
Topology		tile
Buffering		15
Scheduling policy		EDF
Routing		XY
Task Graph		rand0002.stg
Number of Tasks		50
Application Execution Time		114.9
Overall TP		0.435

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
1	A	6	39	33.943	0.0522
2	A	8	47	40.905	0.0696
4	A	3	22	19.147	0.0261
5	A	10	50	43.516	0.0870
6	A	1	4	3.481	0.0087
0	B	4	24	20.888	0.0348
3	B	6	42	36.554	0.0522
7	B	7	48	41.775	0.0609
8	B	5	36	31.332	0.0435
		50	312	30.171	0.0484

Type	TasksPerType	Time Used	Avg Util	Avg Tput	ECT
A	28	162	28.198	0.049	113.6
B	22	150	32.637	0.048	114.9

PARAMETERS		
NumOfPE		9
NumOfPETypes		2
Type Ratio		5:4
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		rand10000.stg
Number of Tasks		100
Application Execution Time		231.6
Overall TP		0.216

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
2	A	10	70	30.225	0.0432
5	A	10	90	38.860	0.0432
6	A	11	64	27.634	0.0475
7	A	10	70	30.225	0.0432
8	A	8	52	22.453	0.0345
0	B	14	96	41.451	0.0604
1	B	11	96	41.451	0.0475
3	B	14	96	41.451	0.0604
4	B	12	96	41.451	0.0518
		100	730	35.022	0.0480

Type	TasksPerType	Time Used	Avg Util	Avg Tput	ECT
A	49	346	29.879	0.042	384.7
B	51	384	41.451	0.055	404.5

PARAMETERS		
NumOfPE		9
NumOfPETypes		2
Type Ratio		5:4
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		rand10001.stg
Number of Tasks		100
Application Execution Time		256.7
Overall TP		0.195

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
0	A	19	121	47.137	0.0740
3	A	11	74	28.827	0.0429
4	A	12	83	32.333	0.0467
7	A	12	68	26.490	0.0467
8	A	12	88	34.281	0.0467
1	B	8	60	23.374	0.0312
2	B	10	78	30.386	0.0390
5	B	6	48	18.699	0.0234
6	B	10	60	23.374	0.0390
		100	680	29.433	0.0433

Type	TasksPerType	Time Used	Avg Util	Avg Tput	ECT
A	66	434	33.814	0.051	304.6
B	34	246	23.958	0.033	340.8

PARAMETERS		
NumOfPE		9
NumOfPETypes		2
Type Ratio		5:4
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		rand10002.stg
Number of Tasks		100
Application Execution Time		332.6
Overall TP		0.150

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
0	A	11	74	22.249	0.0331
2	A	11	79	23.752	0.0331
4	A	13	67	20.144	0.0391
5	A	12	68	20.445	0.0361
6	A	8	52	15.634	0.0241
1	B	12	114	34.275	0.0361
3	B	12	114	34.275	0.0361
7	B	10	66	19.844	0.0301
8	B	11	78	23.452	0.0331
		100	712	23.786	0.0334

Type	TasksPerType	Time Used	Avg Util	Avg Tput	ECT
A	55	340	20.445	0.033	489.4
B	45	372	27.962	0.034	502.3

PARAMETERS		
NumOfPE		9
NumOfPETypes		2
Type Ratio		5:4
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		robot.stg
Number of Tasks		88
Application Execution Time		1630.5
Overall TP		0.031

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
1	A	15	700	42.932	0.0092
3	A	12	580	35.572	0.0074
4	A	13	670	41.092	0.0080
7	A	12	580	35.572	0.0074
8	A	11	440	26.986	0.0067
0	B	9	540	33.119	0.0055
2	B	6	360	22.079	0.0037
5	B	4	292	17.909	0.0025
6	B	6	360	22.079	0.0037
		88	4522	30.815	0.0060

Type	TasksPerType	Time Used	Avg Util	Avg Tput	ECT
A	63	2970	36.431	0.008	2220.3
B	25	1552	23.796	0.004	2134.5

PARAMETERS		
NumOfPE		9
NumOfPETypes		2
Type Ratio		5:4
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		robot.stg
Number of Tasks		88
Application Execution Time		1694.5
Overall TP		0.030

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
0	A	14	660	38.950	0.0083
1	A	15	700	41.310	0.0089
4	A	15	750	44.261	0.0089
5	A	10	450	26.557	0.0059
6	A	9	410	24.196	0.0053
7	B	6	360	21.245	0.0035
3	B	7	420	24.786	0.0041
2	B	6	360	21.245	0.0035
8	B	6	412	24.314	0.0035
		88	4522	29.651	0.0058

Type	TasksPerType	Time Used	Avg Util	Avg Tput	ECT
A	63	2970	35.055	0.007	2328.5
B	25	1552	22.898	0.004	2286.3

PARAMETERS		
NumOfPE		9
NumOfPETypes		2
Type Ratio		5:4
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		sparse.stg
Number of Tasks		88
Application Execution Time		373
Overall TP		0.134

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
1	A	16	370	99.196	0.0429
3	A	11	230	61.662	0.0295
5	A	15	330	88.472	0.0402
6	A	14	335	89.812	0.0375
8	A	15	330	88.472	0.0402
0	B	7	105	28.150	0.0188
2	B	6	90	24.129	0.0161
4	B	5	115	30.831	0.0134
7	B	7	125	33.512	0.0188
		96	2030	60.471	0.0286

Type	TasksPerType	Time Used	Avg Util	Avg Tput
A	71	1595	85.523	0.038
B	25	435	29.155	0.017

PARAMETERS		
NumOfPE		9
NumOfPETypes		2
Type Ratio		5:4
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		sparse.stg
Number of Tasks		88
Application Execution Time		396.8
Overall TP		0.126

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
1	A	12	255	64.264	0.0302
5	A	16	355	89.466	0.0403
6	A	14	320	80.645	0.0353
7	A	13	310	78.125	0.0328
8	A	16	355	89.466	0.0403
0	B	10	150	37.802	0.0252
2	B	4	80	20.161	0.0101
3	B	5	75	18.901	0.0126
4	B	6	130	32.762	0.0151
		96	2030	56.844	0.0269

Type	TasksPerType	Time Used	Avg Util	Avg Tput
A	71	1595	80.393	0.036
B	25	435	27.407	0.016

PARAMETERS		
NumOfPE		16
NumOfPETypes		2
Type Ratio		9:7
Topology		tile
Buffering		15
Scheduling policy		EDF
Routing		XY
Task Graph		rand0000.stg
Number of Tasks		50
Application Execution Time		120.6
Overall TP		0.415

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
0	A	3	12	9.95	0.0249
2	A	5	30	24.88	0.0415
5	A	2	13	10.78	0.0166
6	A	4	31	25.70	0.0332
8	A	4	31	25.70	0.0332
10	A	3	17	14.10	0.0249
12	A	0	0	0.00	0.0000
13	A	4	26	21.56	0.0332
15	A	4	16	13.27	0.0332
1	B	4	24	19.90	0.0332
3	B	4	24	19.90	0.0332
4	B	0	0	0.00	0.0000
7	B	6	42	34.83	0.0498
9	B	4	30	24.88	0.0332
11	B	2	24	19.90	0.0166
14	B	1	6	4.98	0.0083
		50	326	16.895	0.026

Type	TasksPerType	Time Used	Avg Util	Avg Tput
A	29	176	16.215	0.027
B	21	150	17.768	0.025

PARAMETERS		
NumOfPE		16
NumOfPETypes		2
Type Ratio		9:7
Topology		tile
Buffering		15
Scheduling policy		EDF
Routing		XY
Task Graph		rand0001.stg
Number of Tasks		50
Application Execution Time		178.8
Overall TP		0.280

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
0	A	4	31	17.34	0.0224
1	A	2	18	10.07	0.0112
3	A	4	26	14.54	0.0224
4	A	3	22	12.30	0.0168
8	A	1	4	2.24	0.0056
10	A	5	40	22.37	0.0280
12	A	3	12	6.71	0.0168
13	A	6	34	19.02	0.0336
15	A	1	4	2.24	0.0056
2	B	1	6	3.36	0.0056
5	B	4	36	20.13	0.0224
6	B	3	18	10.07	0.0168
7	B	1	12	6.71	0.0056
9	B	3	30	16.78	0.0168
11	B	7	54	30.20	0.0391
14	B	2	18	10.07	0.0112
		50	365	12.759	0.017

Type	TasksPerType	Time Used	Avg Util	Avg Tput
A	29	191	11.869	0.018
B	21	174	13.902	0.017

PARAMETERS		
NumOfPE		16
NumOfPETypes		2
Type Ratio		9:7
Topology		tile
Buffering		15
Scheduling policy		EDF
Routing		XY
Task Graph		rand0002.stg
Number of Tasks		50
Application Execution Time		179.3
Overall TP		0.279

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
0	A	3	22	12.27	0.0167
1	A	5	25	13.94	0.0279
4	A	3	17	9.48	0.0167
5	A	1	9	5.02	0.0056
6	A	3	17	9.48	0.0167
9	A	6	34	18.96	0.0335
10	A	1	4	2.23	0.0056
11	A	2	13	7.25	0.0112
12	A	4	21	11.71	0.0223
2	B	2	12	6.69	0.0112
3	B	2	12	6.69	0.0112
7	B	3	30	16.73	0.0167
8	B	5	36	20.08	0.0279
13	B	4	24	13.39	0.0223
14	B	1	6	3.35	0.0056
15	B	5	30	16.73	0.0279
		50	312	10.876	0.017

Type	TasksPerType	Time Used	Avg Util	Avg Tput
A	28	162	10.039	0.017
B	22	150	11.951	0.018

PARAMETERS		
NumOfPE		16
NumOfPETypes		2
Type Ratio		9:7
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		rand10000.stg
Number of Tasks		100
Application Execution Time		288.7
Overall TP		0.346

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
2	A	7	48	16.63	0.0242
6	A	2	18	6.23	0.0069
7	A	10	55	19.05	0.0346
8	A	5	35	12.12	0.0173
9	A	6	39	13.51	0.0208
10	A	0	0	0.00	0.0000
11	A	9	61	21.13	0.0312
12	A	5	30	10.39	0.0173
14	A	5	40	13.86	0.0173
0	B	9	66	22.86	0.0312
1	B	10	90	31.17	0.0346
3	B	8	72	24.94	0.0277
4	B	8	54	18.70	0.0277
5	B	6	42	14.55	0.0208
13	B	6	36	12.47	0.0208
15	B	4	24	8.31	0.0139
		100	710	15.371	0.022

Type	TasksPerType	Time Used	Avg Util	Avg Tput
A	49	326	12.547	0.019
B	51	384	19.001	0.025

PARAMETERS		
NumOfPE		16
NumOfPETypes		2
Type Ratio		9:7
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		rand10001.stg
Number of Tasks		100
Application Execution Time		260.3
Overall TP		0.384

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
0	A	6	44	16.90	0.0231
5	A	11	74	28.43	0.0423
6	A	10	75	28.81	0.0384
7	A	4	31	11.91	0.0154
8	A	8	47	18.06	0.0307
9	A	5	30	11.53	0.0192
10	A	8	52	19.98	0.0307
12	A	4	26	9.99	0.0154
13	A	10	55	21.13	0.0384
1	B	6	54	20.75	0.0231
2	B	5	36	13.83	0.0192
3	B	2	12	4.61	0.0077
4	B	3	24	9.22	0.0115
11	B	6	42	16.14	0.0231
14	B	5	36	13.83	0.0192
15	B	7	42	16.14	0.0269
		100	680	16.327	0.024

Type	TasksPerType	Time Used	Avg Util	Avg Tput
A	66	434	18.526	0.028
B	34	246	13.501	0.019

PARAMETERS		
NumOfPE		16
NumOfPETypes		2
Type Ratio		9:7
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		rand10002.stg
Number of Tasks		100
Application Execution Time		425.6
Overall TP		0.235

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
0	A	6	34	7.99	0.0141
1	A	7	38	8.93	0.0164
4	A	6	39	9.16	0.0141
5	A	6	34	7.99	0.0141
7	A	6	34	7.99	0.0141
9	A	10	70	16.45	0.0235
10	A	5	40	9.40	0.0117
12	A	3	12	2.82	0.0070
15	A	6	39	9.16	0.0141
2	B	5	36	8.46	0.0117
3	B	7	54	12.69	0.0164
6	B	9	96	22.56	0.0211
8	B	3	24	5.64	0.0070
11	B	5	36	8.46	0.0117
13	B	6	54	12.69	0.0141
14	B	10	72	16.92	0.0235
		100	712	10.456	0.015

Type	TasksPerType	Time Used	Avg Util	Avg Tput
A	55	340	8.876	0.014
B	45	372	12.487	0.015

PARAMETERS		
NumOfPE		16
NumOfPETypes		2
Type Ratio		9:7
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		robot.stg
Number of Tasks		88
Application Execution Time		1832.5
Overall TP		0.055

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
0	A	9	410	22.37	0.0049
3	A	7	280	15.28	0.0038
4	A	7	330	18.01	0.0038
6	A	9	410	22.37	0.0049
7	A	9	510	27.83	0.0049
8	A	3	120	6.55	0.0016
9	A	8	420	22.92	0.0044
10	A	8	370	20.19	0.0044
15	A	3	120	6.55	0.0016
1	B	7	420	22.92	0.0038
2	B	5	300	16.37	0.0027
5	B	3	232	12.66	0.0016
11	B	2	120	6.55	0.0011
12	B	1	60	3.27	0.0005
13	B	2	120	6.55	0.0011
14	B	5	300	16.37	0.0027
		88	4522	15.423	0.003

Type	TasksPerType	Time Used	Avg Util	Avg Tput
A	63	2970	18.008	0.004
B	25	1552	12.099	0.002

PARAMETERS		
NumOfPE		16
NumOfPETypes		2
Type Ratio		9:7
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		robot.stg
Number of Tasks		88
Application Execution Time		2086.5
Overall TP		0.048

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
1	A	4	160	8.53	0.0021
3	A	6	340	18.12	0.0032
4	A	11	540	28.78	0.0059
6	A	3	120	6.39	0.0016
8	A	8	370	19.72	0.0043
11	A	6	240	12.79	0.0032
12	A	8	320	17.05	0.0043
14	A	12	580	30.91	0.0064
15	A	5	300	15.99	0.0027
0	B	2	120	6.39	0.0011
2	B	4	240	12.79	0.0021
5	B	4	240	12.79	0.0021
7	B	3	180	9.59	0.0016
9	B	4	240	12.79	0.0021
10	B	4	292	15.56	0.0021
13	B	4	240	12.79	0.0021
		88	4522	15.061	0.003

Type	TasksPerType	Time Used	Avg Util	Avg Tput
A	63	2970	15.816	0.003
B	25	1552	10.626	0.002

PARAMETERS		
NumOfPE		16
NumOfPETypes		2
Type Ratio		9:7
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		robot.stg
Number of Tasks		88
Application Execution Time		1876.5
Overall TP		0.053

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
1	A	4	160	8.53	0.0021
3	A	6	340	18.12	0.0032
4	A	11	540	28.78	0.0059
6	A	3	120	6.39	0.0016
8	A	8	370	19.72	0.0043
11	A	6	240	12.79	0.0032
12	A	8	320	17.05	0.0043
14	A	12	580	30.91	0.0064
15	A	5	300	15.99	0.0027
0	B	2	120	6.39	0.0011
2	B	4	240	12.79	0.0021
5	B	4	240	12.79	0.0021
7	B	3	180	9.59	0.0016
9	B	4	240	12.79	0.0021
10	B	4	292	15.56	0.0021
13	B	4	240	12.79	0.0021
		88	4522	15.061	0.003

Type	TasksPerType	Time Used	Avg Util	Avg Tput
A	63	2970	17.586	0.004
B	25	1552	11.815	0.002

PARAMETERS		
NumOfPE		16
NumOfPETypes		2
Type Ratio		9:7
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		sparse.stg
Number of Tasks		96
Application Execution Time		319.4
Overall TP		0.313

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
0	A	9	180	56.36	0.0282
3	A	6	120	37.57	0.0188
4	A	10	235	73.58	0.0313
5	A	7	175	54.79	0.0219
7	A	7	160	50.09	0.0219
8	A	6	150	46.96	0.0188
11	A	8	170	53.22	0.0250
12	A	5	110	34.44	0.0157
13	A	13	295	92.36	0.0407
1	B	5	75	23.48	0.0157
2	B	2	50	15.65	0.0063
6	B	2	30	9.39	0.0063
9	B	3	45	14.09	0.0094
10	B	3	65	20.35	0.0094
14	B	6	110	34.44	0.0188
15	B	4	60	18.79	0.0125
		96	2030	39.723	0.019

Type	TasksPerType	Time Used	Avg Util	Avg Tput
A	71	1595	55.486	0.025
B	25	435	19.456	0.011

PARAMETERS		
NumOfPE		16
NumOfPETypes		2
Type Ratio		9:7
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		sparse.stg
Number of Tasks		96
Application Execution Time		289
Overall TP		0.346

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
0	A	9	195	67.47	0.0311
1	A	8	170	58.82	0.0277
2	A	7	160	55.36	0.0242
5	A	8	170	58.82	0.0277
6	A	6	135	46.71	0.0208
9	A	5	110	38.06	0.0173
10	A	12	270	93.43	0.0415
11	A	7	160	55.36	0.0242
15	A	9	225	77.85	0.0311
3	B	2	50	17.30	0.0069
4	B	4	80	27.68	0.0138
7	B	2	50	17.30	0.0069
8	B	6	90	31.14	0.0208
12	B	3	45	15.57	0.0104
13	B	2	30	10.38	0.0069
14	B	6	90	31.14	0.0208
		96	2030	43.901	0.021

Type	TasksPerType	Time Used	Avg Util	Avg Tput
A	71	1595	61.323	0.027
B	25	435	21.503	0.012

PARAMETERS		
NumOfPE		16
NumOfPETypes		2
Type Ratio		9:7
Topology		tile
Buffering		20
Scheduling policy		EDF
Routing		XY
Task Graph		sparse.stg
Number of Tasks		96
Application Execution Time		348.4
Overall TP		0.287

Processor ID	Type	Num Of tasks	Time used	Utilization	Throughput
2	A	5	125	35.88	0.0144
4	A	13	295	84.67	0.0373
5	A	6	105	30.14	0.0172
6	A	3	75	21.53	0.0086
7	A	10	205	58.84	0.0287
10	A	7	160	45.92	0.0201
12	A	10	220	63.15	0.0287
14	A	11	260	74.63	0.0316
15	A	6	150	43.05	0.0172
0	B	6	110	31.57	0.0172
1	B	3	45	12.92	0.0086
3	B	2	30	8.61	0.0057
8	B	4	100	28.70	0.0115
9	B	3	45	12.92	0.0086
11	B	2	30	8.61	0.0057
13	B	5	75	21.53	0.0144
		96	2030	36.416	0.017

Type	TasksPerType	Time Used	Avg Util	Avg Tput
A	71	1595	50.867	0.023
B	25	435	17.837	0.010

VITA

Graduate College
University of Nevada, Las Vegas

Ashwini Raina

Home Address:

4213 Grove Circle Apt#1
Las Vegas, Nevada 89119

Degree:

Bachelor of Engineering, Information Technology, 2004
University of Mumbai

Special Honors and Awards:

Member, Phi Kappa Phi, Initiated in Spring 2007
Member, Tau Beta Pi, Nevada Chapter, Initiated in Spring 2007
Recipient of Ballys Technologies Scholarship, 2006-2007
Recipient of James F. Adams GPSA Scholarship, 2006-2007
Recipient of Sir Ratan Tata Trust Scholarship, 2002-2003
All India Talent Search Examination (AITSE) scholar, 1992

Publications:

“Cell-based Distributed Addressing Technique Using Clustered Backbone Approach”, IEEE Conference Selected in Fourth International Conference on Information Technology: New Generations, Las Vegas, 2007.

“HAUNT-24: Hierarchical, Application Confined Unique Naming Technique”, IEEE Conference Proceedings of Fifth International Conference on Intelligent Systems Design and Applications (ISDA 2005), 8-10 Sept. 2005, Poland.

Thesis Title: fuse-N: Framework for Unified Simulation Environment for Network-on-Chip

Thesis Examination Committee:

Chairperson, Dr. Venkatesan Muthukumar, Ph. D.
Committee Member, Dr. Emma Regentova, Ph. D.
Committee Member, Dr. Ibrahim Saberinia, Ph. D.
Graduate Faculty Representative, Dr. Laxmi Gewali, Ph. D.