Design of Platform for Exploring Application-Specific NoC Architecture

LIU, Zhouyi

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Philosophy

in

Electronic Engineering

The Chinese University of Hong Kong

March 2011

Abstracts

As a new approach to accommodate multiple processing elements (PEs) on a single chip, Network-on-Chip (NoC) brings enhanced performance and scalability, in comparison with conventional communication infrastructures. Since huge research margin exists in NoC research domain which ranges from topology, routing algorithm to flow control mechanism, many innovative NoC architectures with distinctive configurable features emerge recently that either emphasizes on short runtime, or concentrates on low power consumption or takes both into account. Among all those candidate NoCs, how to single out the most suitable one, satisfying strict performance requirement or resource constraint, for specific application becomes an compelling and urgent challenge. Systematic platform which is capable of searching optimal NoC for specific application is called for.

This thesis proposes an integrated environment aiming at optimal NoC exploration for specific application. The entire process consists of picking out a suitable architecture from available ones and exploring appropriate configuration of selected architecture. Graphical User Interface (GUI) is well designed as the service window of the whole package of user operation, including architecture selection, feature configuration, benchmark generation, performance evaluation as well as NoCs optimization and comparison. All operations proceed by observing decently developed rules to ensure integrity of architecture library and reproducibility of any evaluation.

Configurable NoC architectures integrated to platform (NoC Library) are low-latency & low-power oriented, respectively, providing broadened application space for various performance requirements / cost constraints. Associated features of NoC architecture are all extracted and displayed on GUI for convenient access. NoCs optimization concentrates on buffer distribution that influences chip area, power consumption as well as network performance. Comparison among NoCs reaches beyond plain numeric comparison alone. Validity check is introduced to ensure that the performance metrics to compare are obtained by identical standard.

摘要

单晶片网络，作为单晶片系统的发展趋势，在集成电路研究领域引起了广泛的兴趣。随着工艺的不断改进和电路设计能力的持续提高，将单元数目繁多的网络系统集成在单一晶片成为必然。网络中的各个处理和存储单元之间存在着大量的数据交换，以期实现复杂的功能或完成巨量的数值计算。设计和选择一个高效率的网络系统来协调和优化数据交换成为一个重要的议题。

近年来有许多单晶片网络设计相关的先进技术提出，以期增强片上网络的数据传输速度，降低晶片功耗和面积，由这些技术衍生出了不同的单晶片网络的设计。这些设计往往在某一项特定的参数上对通用网络有显著的改进，要么网络传输速度，要么网络功耗。然而这些参数的提高或者以牺牲其他性能指标为代价，或者仅仅适用某一特定类型的数据交换模式。没有哪一项设计可以全面解决所有问题。

单晶片网络设计的一个显著特点，是其设计的多样性和参数配置的复杂性。不同的网络设计具有各自的优势和局限性，往往是在特定的数据交换模式下发挥最大的功效。想要有效的比较两个网络设计，最好的方式就是把它们集成在一个统一的环境下，利用严格的控制流程来配置、评估和比较多个片上网络。如何配置选定的单晶片网络设计，让它在特定的数据交换环境下发挥最大功效，是一个同样重要的课题。

本文提出的是一个综合性的单晶片网络应用平台。它的功能是探寻已有片上网络的优点，为指定的数据交换模式，定位最合理的网络配置。综合性的平台可以保证所有片上网络的性能比较基于完全相同的条件，大大减少不同平台间比较造成的误差。与此同时，平台的优化功能可以自动调整用户定义的存储单元大小，提出更合理的配置建议。图形化的用户界面既便于用户操作，也保证系统文件不会被意外修改。

Contents

List of Figure

List of Table

Acknowledgement

When the first time stepped on this lovely campus, passion, relieve and thirsty all came up and I could barely hold my thought. It was not only about a dream that finally came true, but also the fancy of the next two years contribute to a great research topic. I saw nothing but sun shining, heard nothing but birds chirping.

Things were turning down half a year later. All at a once the whole year efforts turned to be naught and I hovered to have nowhere to settle down. I knew I had to start over, again, just had no idea from where. Things were not getting better after department transfer. Striving in cracking out source code, struggling in gloomy direction. Thesis of master degree was noting but a myth.

At this very moment, upon the completion of the thesis, I have my own heart filled with gratitude and appreciation. I am sincerely thankful for my supervisor, Professor Choy Chiu Sing, for his guidance as well as support. I would thank Zhang Min and Xin Ling, for their generous sharing and sharp insights. Thank Professor PUN Kong Pang, Professor LEUNG Ka Nang, Mr. YEUNG Wing Yee and all the ASIC Lab members who genuinely offer me help and forgive my immature all along. I can not make it without you all. I am so blessed to have the wonderful time with your accompany.

In the end, I would like to give my appreciation to my parents. Your love and concerns accompany me for all these years, which I could never ever pay off.

Chapter 1 Introduction

This chapter briefs the background of Network-on-Chip and similar platform designs presented by other literatures, introduces the overview of proposed platform by this thesis, and explains contribution of the platform design.

1.1 Network-on-Chip

With the development of silicon process technology and circuit design capacity, System-on-Chip (SoC) becomes the mainstream solution to challenge problems in the field of electronic devices. Typical SoC structure comprises of processors, controllers, memory arrays, and other possible components. The success of SoC relies on design of individual cores as well as the infrastructure interconnecting involved components.

Initially, SoC consists of a few components where communication among them are implemented by point-to-point connection. Given small number of communication paths, customizing interconnection between any pair of cores of data exchange while leaving others unconnected directly achieves tightened integrity with minimum design overhead. It occupies insignificant proportion of system design effort, with regard to the entire development duration.

Increasing complexity of chip design makes reuse of cores frequent, thus standardized core interface is widespread and share-medium architecture becomes the prevailing option in SoC domain. The backplane bus is the most common example of an on-chip, shared medium structure. The methodology of customizing connection of cores in SoC design was gradually abandoned in a general sense, and designers started to develop component interface complying with particular protocols. With well tested and verified bus structure and compatible components, it becomes no longer necessary to concern about details of interconnection. Design, verification and test of bus system are all pre-completed.

In spite of the advantage of low overhead and being convenient to use, shared medium infrastructure has limited scalability and poor power efficiency. Every data being broadcast indicates the data must reach every possible receiver at great energy cost. The impact is insignificant in the case of less than 10 cores. As future integrated systems predicted to contain tens or hundreds of components, bus based system would inevitably suffer from limited scalability and poor performance, as recognized in [1].

The plug-and-play fashion of shared-medium structure is appreciated. What is expected is an advanced communication infrastructure inheriting shared-medium's reliability, while retaining scalability and power efficiency. Communication centric design methodologies, e.g. Network-on-Chip (NoC), have been proposed in [2][3] where interconnection and communication among cores for a SoC will captivate the major portion of the design and test effort. As an approach to accommodate multiple components on a single chip, NoC satisfies all expectations and has huge research margin. Once dozens of cores involved, NoC is the promising one of the few options for now. Due to the vast range of NoC design, numerous NoC architectures are proposed with distinctive routing algorithm, flow control mechanism, arbitration strategy, etc. These configurable NoCs offer flexibility across plenty of products and reduce design-and-test effort as well as time-to-market [22].

1.2 Related Works

The NoC platform is an integrated environment that provides comprehensive support for on-chip network research. Most of them are simulation based, as proposed by literature [4][5][8][10][11]. Fundamental operation flow of the platform proceeds as configuration of network architecture, traffic pattern generation, simulation and performance evaluation. The platforms differ from one another in supported network topology, buffer distribution, traffic pattern and other configurations. Average network latency is the evaluation metric shared by all platforms, while some of them extend it further to include more. Several typical platform designs are listed as below with pros and cons elaborated, individually.

J.C. Villanueva, J. Flich and etc. from Universidad Politecnica de Valencia and Sun Microsystems [4] presents a full-system simulation framework by combining Virtutech Simics [6] with an extended Wisconsim GEMS toolset [7]. It is a complete software-level event-driven simulator supporting collective communication, wormhole & virtual cut through switching and virtual channels. Supported network topologies include two dimensional mesh and bidirectional ring. A subset of the SPLASH-2 benchmarks, including FMM, LU, Radix, Radiosity, FFT, Barnes and Raytrace are applied for performance evaluation. Due to the capacity of Simics, memory behaviors are fully emulated with cache misses being detected. Impacts of various topologies and flit sizes are analyzed and compared.

Table 1-1 Features of the Platform from Literature [4]

Pros	Cons
numerous benchmarks involved	software-level simulator
memory behaviors detectable	basic router functionality
configurable topology	
configurable flit size	

The system [5] proposed by B. Talwar and B. Amrutur from Indian Institute of Science is System-C based framework to explore the impact of various architectural parameters of the on-chip interconnection network on cost and performance. Its configurable parameters include topology (mesh, torus and folded torus), I/O ports number, buffer size, link bit-width and link pipeline degree. Available benchmarks are comprised of uniform random, hot-spot, matrix transpose and localized traffic patterns. Request-response traffic, or called data dependency is also taken into account. Latency and throughput are reported as performance metrics. Power consumption of link is revealed with that of network routers excluded. The authors also analyze the network parameter tradeoffs, such as NoC power/latency tradeoff, power-performance tradeoff with frequency scaling and power-performance tradeoff with voltage and frequency scaling.

Table 1-2 Features of the Platform from Literature [5]

Pros	Cons
data dependence traffic involved	basic router functionality
configurable topology	buffer allocation strategy ignored
configurable link	data size impact ignored
configurable buffer size	
frequency and voltage taken into account	

L. Papadopoulos, S. Mamagkakis and etc. from Democritus University Thrace, Greece extends Nostrum NoC Simulation Environment (NNSE) [9] by adding new features like interface for thread allocation, irregular topology and providing new evaluation metrics [8]. It is more suitable for implementing high level transformations on dynamic network and multimedia application.

Table 1-3 Features of the Platform from Literature [9]

Pros	Cons
can simulate fairly complex application	basic router functionality
	small configuration space

S. Chai, C. Wu etc. from University of Electronic Science and Technology of China implement general NoC router architecture in System-C [10], with support to mesh & torus topologies, configurable buffer depth and bit-width. Five synthetic traffic patterns (uniform, matrix transpose I, matrix transpose II, hot-spot and NN) are modeled for performance evaluation. It claims high-extensibility of design by code modification due to modularization modeling. Simulation results with different traffic patterns are compared and buffer optimization is also taken into account. The buffer optimization focuses on general buffer depth, by observing the impact of universal buffer depth on network performance. In spite of high extensibility claimed, it is not easy for the platform users to do it.

Table 1-4 Features of the Platform from Literature [10]

Pros	Cons
configurable topology	basic router functionality
configurable buffer depth	claimed extensibility is difficult to achieve
multiple traffic patterns	primitive buffer optimization

F. Fu, S. Sun and etc. from Harbin Institute of Technology, China builds an evaluation platform for NoC on multiple levels [11]. Transaction level modeling and RTL model are both implemented for speed simulation and cycle-accurate emulation, respectively. Synthetic benchmarks are applicable, including random uniform, random locality and fixed distribution traffic. Evaluation metrics include latency, throughput and link utilization.

Table 1-5 Features of the Platform from Literature [11]

Pros	Cons
configurable topology	basic router functionality
multiple interface protocol	small configuration space
multiple traffic patterns	buffer allocation strategy ignored

N. Banerjee, P. Vellanki and K.S. Chatha from Department of CSE, Arizona State University, Tempe, proposes a VHDL based cycle accurate RTL model for evaluating the latency, throughput, dynamic, and leakage power consumption of NoC based interconnection architectures [34]. The model is configurable in packet size, link length and width, virtual channel size as well as switch technique. The RTL model is synthesized to extract SPICE level netlist. Delay, dynamic and leakage power at 180nm are all available for comparison. The authors compare the performance and power consumption, respectively of the network for varying virtual channel number or switch technique.

Table 1-6 Features of the Platform from Literature [34]

Pros	Cons
VHDL based RTL model	basic router functionality
configurable buffer size	single traffic pattern
configurable link	primitive buffer optimization
selective switch technique	
power estimation possible	

To summarize, the platforms stated above all implement fundamental NoC functionality and are capable of running some traffic patterns to compare or report evaluated performance. Some platforms are supporting realistic cores, e.g. [5], where real applications and memory evaluation are possible. While most of them choose synthetic benchmark. Significant insufficiency of these platforms is concluded as below:

a) NoC architecture selected for platform development are fundamental, with limited configuration margin. Topology and buffer size are the common flexible features. Meanwhile, plenty of novel performance enhancement techniques, no matter low latency oriented or low power oriented, are not considered so far.
b) No optimization technique is proposed to obtain better configuration for specific application. Some literatures address on buffer parameter searching by adjustment of virtual channel number at all ports simultaneously. But it can not be justified as automatic optimization.
c) Accurate NoCs comparison across platforms is infeasible because neither traffic pattern generation nor measurement methodology is explicitly revealed and there is no mechanism to check validity of any comparison operation.

Platform proposed by this thesis addresses all those issues by introducing a systematic environment with enhanced NoC library, optimization technique and comparison mechanism. The NoC library includes multiple NoC architectures with advanced low-power and low-latency techniques. Buffer optimization tool is offered to obtain better configuration while comparison mechanism is introduced to the platform for result reproduction and valid comparison. Moreover, graphic user interface unifies the whole operation. All of them distinguish proposed platform from all the others stated before.

1.3 Platform Overview

Proposed platform by the thesis is a simulation-based, HDL-level integrated environment which applies synthetic benchmark for performance evaluation.

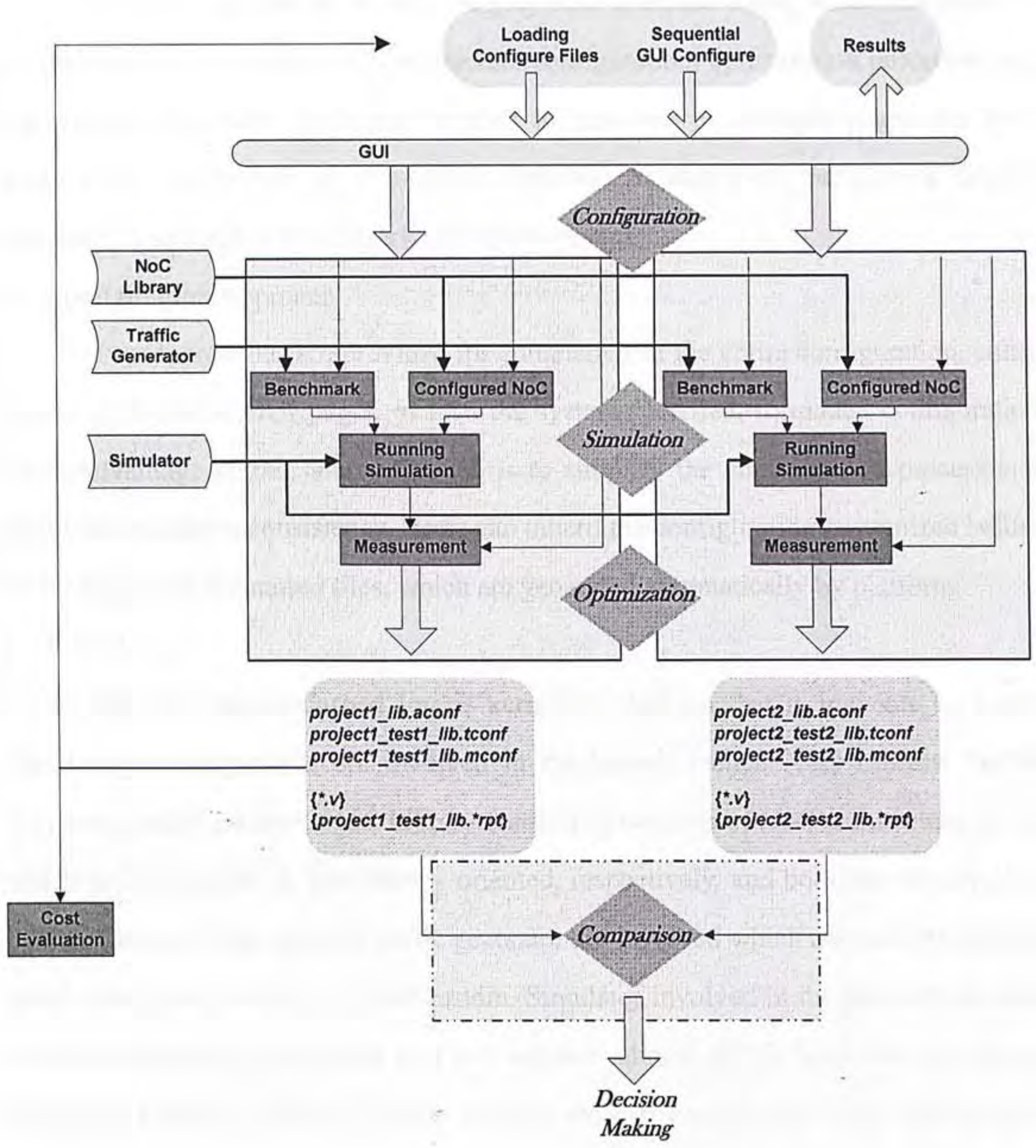
Simulation-based is the methodology chosen by all mentioned platforms above. There are a number of programming languages and tools to emulate behaviors of a complete network that are simpler in comparison with hardware description language, in terms of development time and effort. High level programming language is widely adopt because it is easier to model network behaviors and it is possible to run real applications and assess memory status. Nevertheless, HDL is still chosen to describe NoC in current platform. High level simulation is widespread so far partly due to the fact that NoC is rare to put into practice nowadays. However, it believes that NoC technique is deemed to implement on hardware extensively some day. Then software simulation can not deliver satisfied result any more. Further, many innovative NoC designs are realized in HDL. It is beneficial to have them in platform library.

Generally, the most accurate approach to evaluate NoC for specific application is to run the actual application on well-configured NoC. But it is not the best option in reality. Running real application requires completion of the entire NoC development flow, which comprises of NoC configuration, synthesizable code generation, hardware mapping, development of processing elements and integration of network components. That is tedious and time-consuming. If reconfiguration required, the whole flow has to re-traverse again. Apparently, running real applications in early development stage is an inappropriate development methodology. Therefore, synthetic benchmark is chosen to shorten development duration and the entire flow is simplified to a large extend.

Above all, this thesis aims at development of a systematic environment for NoC research domain. It covers wide range of NoC topics: fundamental network router architecture, low-power technique, low-latency technique, configurable buffer size, various traffic patterns, performance metrics (latency and throughput), cost metrics (area and power), resource efficiency (buffer and link utilization), buffer optimization and network comparison.

Figure 1-1 illuminates the platform structure. It composes of two major components, GUI and kernel, allowing two approaches of configuration and exporting sets of relevant files.

Figure 1-1 Illumination of Platform Structure



- GUI

What is viewable by users is the graphical interface alone, which composes of several sections, covering NoC architecture configuration, optimization procedure and comparison procedure. Each section may include one or multiple pages. As NoC architecture, low-power or low-latency, selected, related NoC parameters display followed by settings of benchmark and measurement.

- Configuration Approach

Two approaches are provided for completion of the entire configuration, either follow guidance of GUI pages, or load the system-specified, formatted configuration files. Advantage of the latter approach is to simplify the configuration process and maintain evaluation consistency. Users can inherit the configuration customized before by loading these formatted files, which are generated automatically by platform.

- Kernel

The part within dashed line is kernel of platform that is invisible by users. Three major components are involved in the kernel, named NoC Library, Traffic Generator, and Simulator. NoC Library consists of two sets of NoC architecture so far, which are low-power & low-latency oriented, respectively, and both are developed in System Verilog. Four types of traffic generator are involved which are random uniform, matrix transpose, locality and full custom. Simulator involved in the platform for now is Mentor Graphics QuestaSim 6.3f that supports almost all the hardware description languages (Verilog, VHDL, System Verilog, etc.). It ensures that NoC architectures developed by other languages can also be integrated into platform without linking to another simulator. A set of scripts in Perl called process links them all to make the platform operate properly.

- Export Files

As optimal NoC architecture and configuration explored, synthesizable code of each router can be generated for area, power estimation and further practical applications. In the mean time, performance reports *project_test_lib.*rpt* are also generated for future comparison. *project_lib.aconf*, *project_test_lib.tconf* and *project_test_lib.mconf* are files of NoC configuration, benchmark setting and measurement setting. All of them are produced for system purposes.

Proposed platform features in three major aspects. The first is configuration through GUI. With the friendly graphical interface, users can conveniently configure network architectures. This procedure can be completed without comprehension of design details of network architecture and without accessing HDL source code personally. It improves configuration efficiency, ensures architecture integrity and shortens development time. The second is buffer optimization technique. Proposed platform is capable of analyzing buffer utilization rate and offers suggestions for optimized buffer allocation. The third is NoCs comparison. Comparison among NoCs is prominent as so many innovative NoC designs emerge these days. To compare them consistently and efficiently is critical for selecting the final NoC for specific application. Particular schemes ensure that the comparison is meaningful and efficient.

All of those features ensure integrity of NoC library and shortens time for sifting out appropriate solution among multiple choices. If users' requirement can be satisfied by available NoC architecture with proper configuration, screening process is fast to complete. If not, the platform still assists users to explore configuration priority and possibilities.

1.4 Author's Contribution

Given the fact that so many innovative configurable NoC architectures emerge recently, it is necessary and urgent to have a platform to utilize these designs in a unified manner. It assists to identify pros and cons of each design and to single out the most suitable one for specific application or application domain.

The proposed platform achieves this objective by presenting a complete flow from NoC architecture configuration, to NoCs optimization and comparison. NoC architectures of proposed platform integrate advanced high-performance techniques that are unavailable in rival platforms. Moreover, accurate area and power estimation is feasible to obtain. The platform also offers suggestions for buffer optimization. The entire flow is strictly administered to satisfy system quality requirement, making sure that comparison proceeds validly. The platform is open to integrate more NoC architectures and extension of optimization tools is also feasible. Proposed platform design lays the foundation for systematic environment of NoC research domain in the long run.

The thesis is organized in six chapters. Chapter 2 elaborates router architecture of NoC library which are the most important platform component; Chapter 3 introduces benchmark and measurement setting; Platform structure, including file system, graphical interface and platform processes are described in Chapter 4; Chapter 5 addresses optimization technique as well as comparison mechanism, and Chapter 6 summarizes the thesis and presents the future work.

Chapter 2 NoC Library

NoC is such a communication structure that connects all the cores (processors, memories, controllers, etc.) and facilitates message transfer among them. The cores connected by NoC are called processing elements. A complete NoC infrastructure consists of three types of components, named network router, network interface and physical link. Network router, as backbone of NoC is the module guides the message from one network node to another. It determines the path of the message as well as tackles resource contention. Network interface is the module connecting network router to processing element. It is responsible for packetization and depacketization of the message, as well as signaling exchange for message acquisition and halt. Specific bus protocol must be complied by both processing element and network interfaces. Physical link connects neighboring or non-neighboring routers which transits the message from one spot to another. Simplest physical link is wire with or without register. Complex and high-efficiency physical links are also proposed to satisfy extraordinary quality requirements [18].

NoC library, as the most important platform component, comprises of two sets of configurable NoC architecture, low-power oriented architecture developed by M. Zhang and C.S. Choy [12] and low-latency oriented architecture developed by L. Xin and C.S. Choy [13], both are from Electronic Engineering Department, The Chinese University of Hong Kong. Each set of architecture is described by a number of System Verilog files which are called and copied by platform. Platform users can select either set of architecture, and configure it to apply to specific application.

2.1 Network Terminology

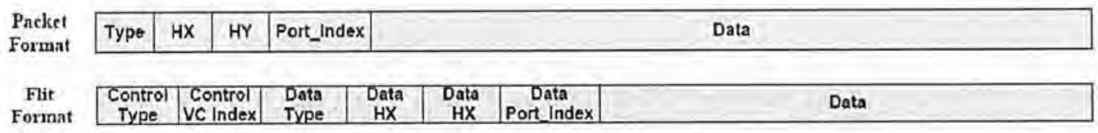
- **NoC Data Format**

To clarify the description of network communication, definitions of key terms are explained herein. *Data* and *message*, if not specified, are synonymous in this thesis, both are abstract description of the information generated by one processing element and required by another. The data unit within processing element could be in any shape, while the message transferring within network must comply with particular protocol for less complex hardware design overhead and communication efficiency. Thus besides of meaningful processing element payload, information on routing path, reserved buffer lane, message validity are all integrated into a message unit for network transfer. The encasement is called packetization, accomplished by network interface. Therefore what the network routers are dealing with is fixed message block, rather than arbitrary data shape.

When going through packetization step in network interface, data from processing element is sliced into pieces and packaged into fixed width of unit, called *packet*. It is the basic shape of network message, all bits of which share the same routing path within the network. A packet comprises with two parts, control signals and payload. Payload is the actual data segment for transfer. On contrary, control signal is added by network interface for routing computation. It includes packet status, destination address, output port index, etc. Width of control signal is determined by network size, maximum virtual channel number at single port and so on. While the width of payload can be arbitrary. Figure 2-1 shows the format of typical packet.

Packet is further sliced into smaller units, called *flit*, which is also the smallest unit in network data exchange. The number of flits comprising a packet is called packet length. As width and length of packet are determined by users, width of flit is fixed automatically since all flits are forced identical width for the sake of lower hardware complexity. Every packet consists of at least one *head flit* and one *tail flit*. Head flit comprises routing information while tail flit indicates transfer completion. If more than two flits are assigned for a packet, others rather than head or tail are named *body flits*. Since flit is just a transformed representation of identical message as packet, extra controlling information is also imbedded in flits, specificly within head flit alone. Head flit comprises the data contained in packet as well as flit type information (head, body or tail) and virtual channel information (which virtual channel at downstream router is reserved for current packet).

Figure 2-1 Typical NoC Data Format



Importance of data format is often understated in NoCs evaluation and comparison. Although network bandwidth is approximated by packet number, the maximum amount of data transferred by a link is one flit. If literatures reports evaluation results with data format undisclosed, comparison across NoC designs is meaningless. Impact of different packet lengths on network performance is seldom taken into account in NoCs comparison.

- **Topology**

Given the three types of network components and associated processing elements, the second step to build NoC is to organize all those modules in particular layout, called topology. Network topology can be simply categorized to regular and irregular. Irregular network is highly relevant with application and not commonly suitable for the purpose of reuse. Common regular topology includes two-dimensional mesh, torus, star, butterfly, etc. Two-dimensional mesh is the only topology covered in this thesis. The number of router port is configurable in both network architectures of NoC library, thus arbitrary two-dimensional topology is feasible to implement, as long as top HDL module and associated routing algorithm are revised accordingly.

- **Routing Algorithm**

Routing algorithm is used to determine the path that message follows from its source node to destination. It often relies on network topology. Routing algorithm falls into two types, determined or dynamic. Determined routing algorithm refers to fixed path once source and destination nodes are selected. On contrary, dynamic routing algorithm evaluates traffic congestion and allocates path dynamically. Determined routing algorithm is easy to implement and costs less chip resources. Dynamic routing algorithm might ease traffic congestion to some extent, yet resource occupation and computation time are its shortcomings in comparison with the former one.

- **Virtual Channel**

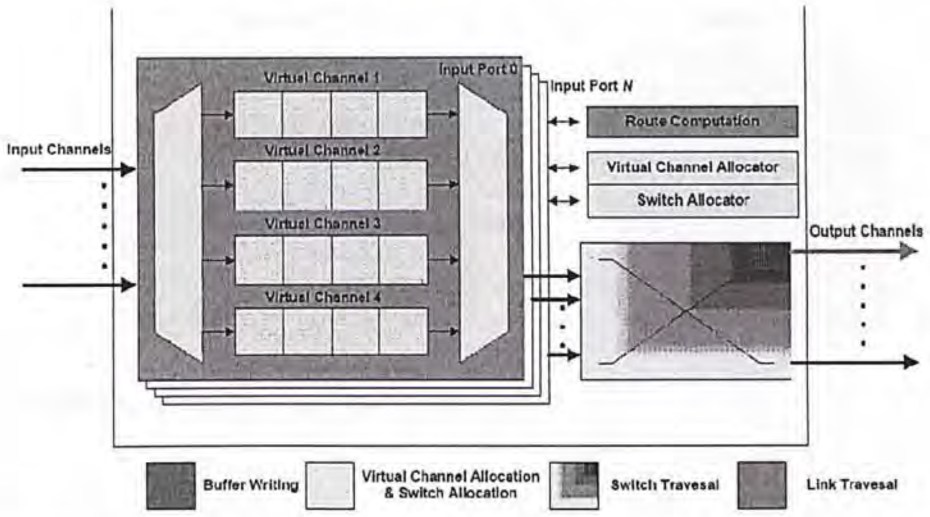
Virtual channel is a flow control mechanism proposed to increase network throughput while reduce average network latency [30]. It is obtained by dividing the buffer storage associated with each physical channel into several small queues that can hold one or more data unit. The virtual channels associated with one physical port are allocated independently.

2.2 Basic Structure

Function of network router is to receive packets and determine their succeeded network position, neighboring router buffer or local processing element. Architecture of basic network router is described as below (

Figure 2-2). Network router owns multiple input ports and output ports of identical number. Certain input ports are connected with associated output ports at each cycle, allowing packets at input ports to move forward. Three major components inside of router are buffer, controller and switch. Buffer is used to store packets in case downstream router is busy. Controller calculates succeeded path of arriving packets and switch provides the path from input ports to granted output ports. Suppose five ports (input and output) are labeled as *East*, *North*, *West*, *South* and *Local*. If *East* input port and *West* output port are connected by controller, then packets at *East* input port of current router move forward, leaving current router through *West* output port and ultimately arrive at *East* input port of downstream router.

Figure 2-2 Normal Network Router Architecture



Typical network router is of five pipeline stages, listed as buffer writing (BW), virtual channel allocation (VA), switch allocation (SA), switch traversal (ST) and link traversal (LT). Function of each stage is:

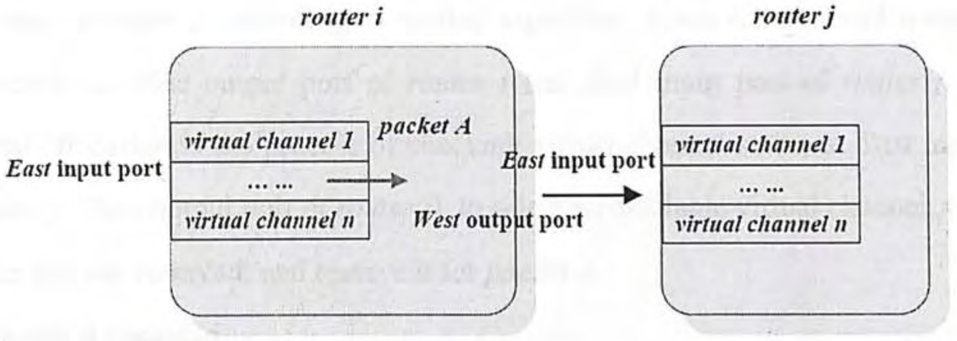
- **Buffer Writing**

Network router has configurable number of input ports as well as output ports, with the exact number depending on applied topology. For 2-D mesh topology, maximum number of input ports of the network router is five. As a packet arriving at one input port of the router, it is stored in buffers of the router temporarily in case of switch unavailable or downstream router being busy. This mechanism increases network throughput by letting idle links be utilized as much as possible.

- **Virtual Channel Allocation**

Since only stop at buffers during transferring, the packets stored in current router buffer has to be granted with buffer of downstream router before moving forward [16][20]. As virtual channel adopt as flow control mechanism, the packet has to succeed requesting of a non-full virtual channel. If buffers at downstream router are all fully filled, there is no way packets can keep transferring in spite that no contention occurs.

Figure 2-3 Virtual Channel Allocation



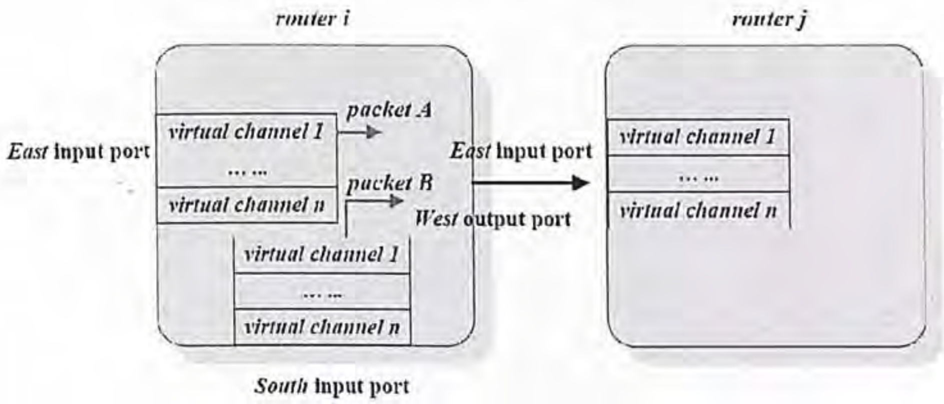
For example, in

Figure 2-3, *packet A* is stored at *virtual channel 1*, *East* input port of *router i*, and its next stop is *router j*, according to routing algorithm. Since *router i* and *router j* are connected via *West* output port of *router i* and *East* input port of *router j*, virtual channel allocation is the process of checking virtual channel status at *East* input port of *router j* (*West* output port of *router i*), to select an available virtual channel, which is neither full nor reserved, and reserve it for *packet A*.

- **Switch Allocation**

With virtual channel successfully reserved at downstream router, one packet still needs to compete switch with other packets requesting for identical output port of current router. The process of granting one output port to one input port requesting for it is called switch allocation.

Figure 2-4 Switch Allocation



In Figure 2-4, if *virtual channel 1* at *East* input port of *router j* is reserved for *packet A* (*virtual channel 1* at *East* input port of *router i*) and *virtual channel 2* at *East* input port of *router j* is reserved for *packet B* (*virtual channel 1* at *South* input port of *router i*), contention occurs as both packets requesting *West* output port of *router i* to reach *router j* simultaneously. Switch allocation is the mechanism to decide which input port will be granted the switch to connect to *West* output port of *router i*.

- **Switch Traversal**

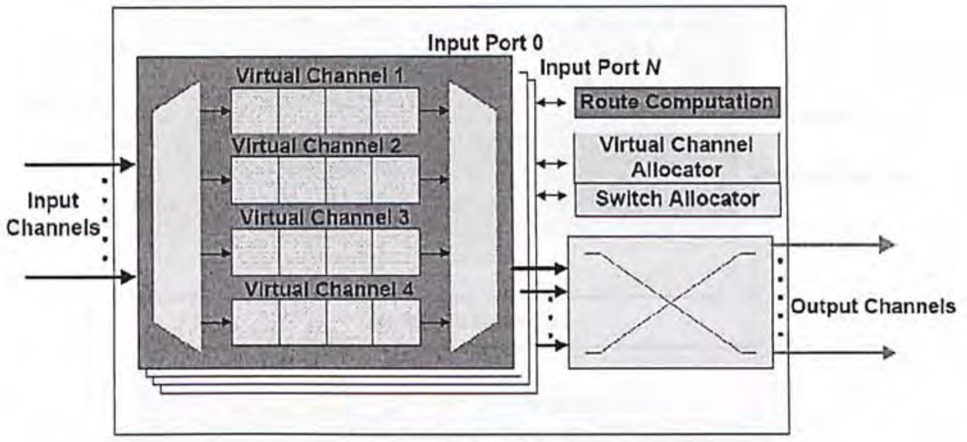
After packet having succeeded in both virtual channel allocation and switch allocation, associated input / output port.link is connected by switch and the granted packet takes one cycle to travel from input virtual channel to output port, called switch traversal.

- **Link Traversal**

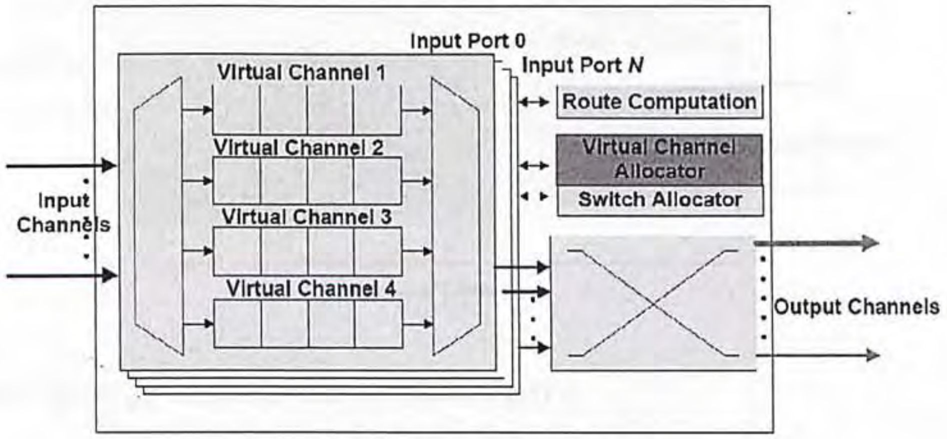
Neighboring routers are all connected with links, through which packets transfer from one node to another. Normally, one cycle is spent on travelling physical link from output port of current router to input port of downstream router. Actually latency depends on specified configuration.

Figure 2-5 illuminates the five pipeline stages of network router. The block in gray indicates that the blocks are activated at current pipeline stage.

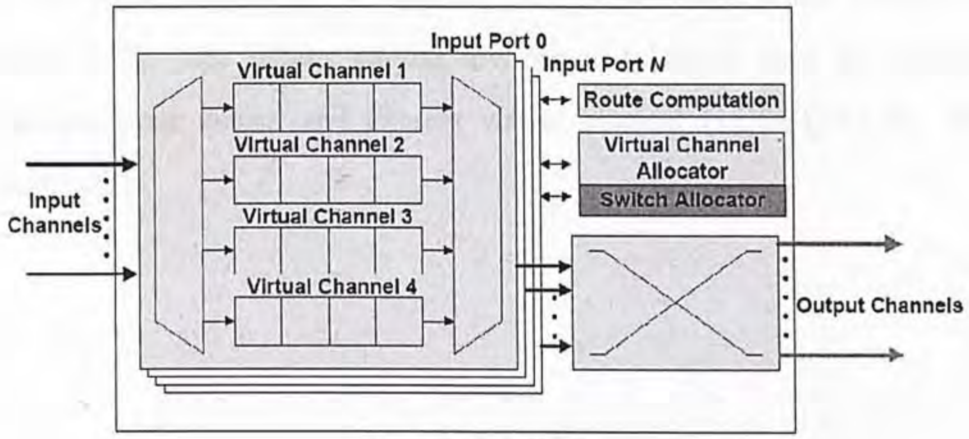
Figure 2-5 Five Stages of Normal Network Router



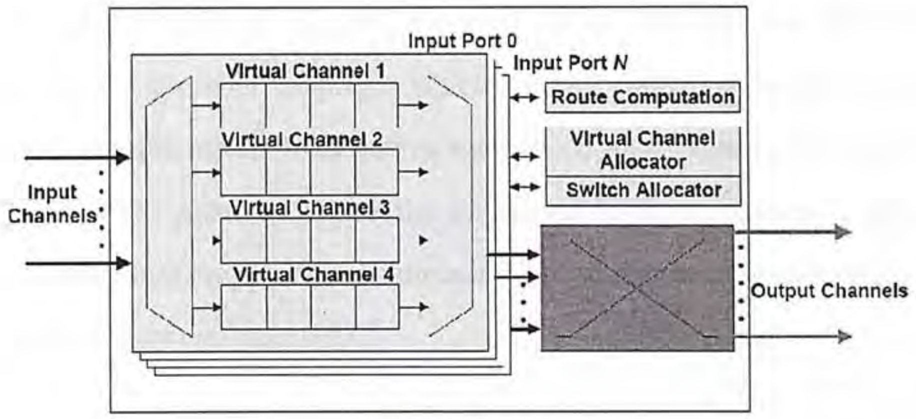
(a). Buffer Writing



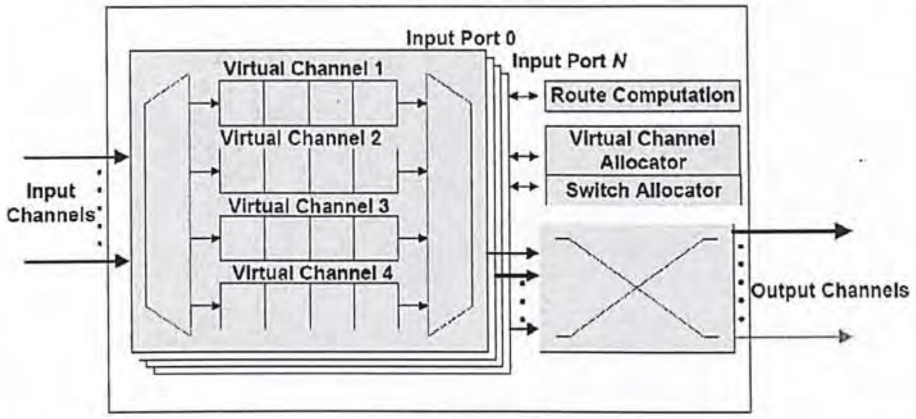
(b). Virtual Channel Allocation



(c). Switch Allocation



(d). Switch Traversal



(e). Link Traversal

2.3 Low-Power Oriented Architecture

The low-power oriented NoC architecture inherits fundamental features of basic network router and applies several low-power schemes, such as low-cost allocator design, clock gating and express virtual channel [15][24][25][26]. The summary is shown in

Table 2-1. First two low-power schemes in regard to cost reduction are impossible to be implemented in high level languages which emulate functionality alone. Express virtual channel insertion affects both latency and power, yet it is also never reported to be add into any NoC platform. Since the impact of EVC on latency is generally unpredictable while that on power is determined, current architecture supporting EVC is categorized as low-power oriented.

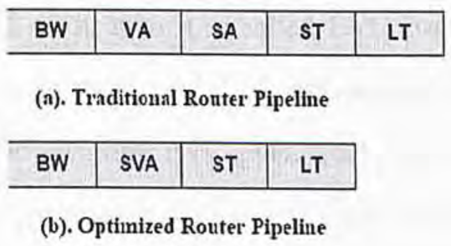
Table 2-1 Features of Low-Power NoC Architecture

Topology	2-D Mesh
Routing Algorithm	X-Y
Buffer	Size Configurable
Low-Power Schemes	
	Low-Cost Allocator
	Clock Gating
	Express Virtual Channel
Low-Latency Schemes	
	Low-Cost Allocator
	Express Virtual Channel

2.3.1 Low-Cost Allocator Design

Just as stated in last chapter, conventional network router is of five stages including virtual channel allocation and switch allocation. Low-power oriented architecture from NoC library merges two allocator into one, to reduce clock cycles from two to one. It scales silicon area and power consumption with little observed penalty. With substitution of generic allocator with a combined one, network router pipeline stage drops from five to four.

Figure 2-6 Pipeline Stage Comparison


BW
VA
SA
ST
LT
(a). Traditional Router Pipeline
BW
SVA
ST
LT
(b). Optimized Router Pipeline

2.3.2 Clock Gating

System clock contributes to significant power consumption to integrated circuits [27]. As clock frequency fixed, certain amount of power consumption is bound to generate by clock voltage reverse, no matter how extensively circuit components are in operation. Turning off system clock of specific components or sub-modules while they are idle is a feasible solution to optimize power consumption. Such technique named clock gating applies to most sub-modules within network router.

2.3.3 Express Virtual Channel Insertion

As a flow control mechanism, Express Virtual Channel (EVC) distinguishes certain paths within the network from others [31]. The privileged path is called EVC path that benefits in terms of latency and power. Low-power oriented architecture of NoC library supports express virtual channel that provides more flexibility to network configuration.

2.3.3.1 Notation

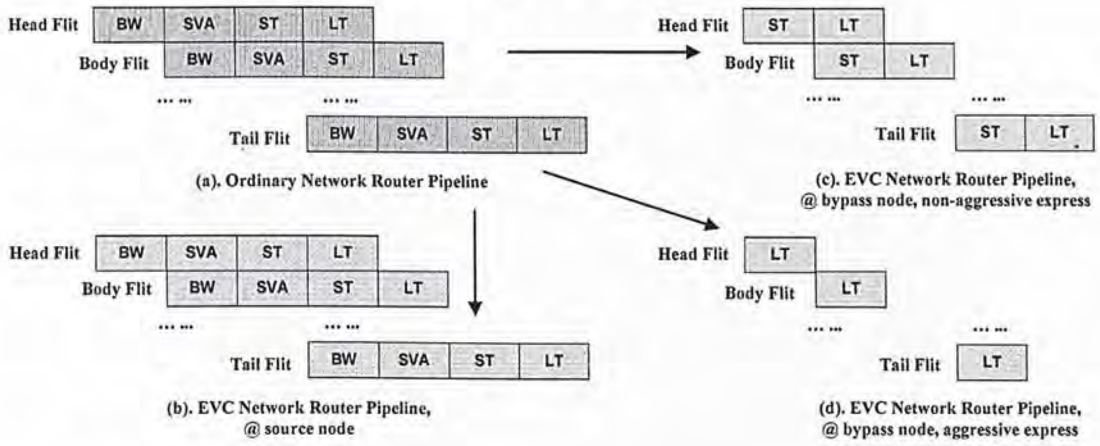
According to the definition of express virtual channel, EVC path starts at any node within the network and ends at another. The starting node is named *source* node, with ending node as *sink*. Routing algorithm calculates the routing path in between and all node(s) along the EVC path is(are) called *bypass* node(s). If, routing path of a packet overlaps with an EVC path, then it is called EVC packet, distinguished from other packets which never pass through any EVC path called NVC packets. Any EVC packet must associate with at least one EVC path. Once the EVC packet reaches its associated source node, network router would deal with it by distinctive mechanism rather than the conventional one.

2.3.3.2 Pipeline and Latency

Whenever EVC packet reaches associated source node, it proceeds router stages as normal until being granted to output port of source node and arriving at input port of the first bypass node. Then EVC packet skips BW (buffer write), SVA (switch and virtual channel allocation), ST (switch traversal), and directly go for LT (link traversal) stage. This is achieved by assigning EVC packets higher priority over NVC packets that contend resources with them.

Figure 2-7 lists pipeline stage of involved cases. When EVC path is inserted into network, pipeline stage at source node is identical as inserted before, only the pipeline stage at bypass node is updated. As two bypass schemes, non-aggressive and aggressive bypass, available, EVC flit could spend two cycles through bypass node (Figure 2-7(c)) or one cycle only (Figure 2-7(d)). In non-aggressive express router, switch traversal and link traversal stages are inherited; while in aggressive express router, only link traversal stage remains.

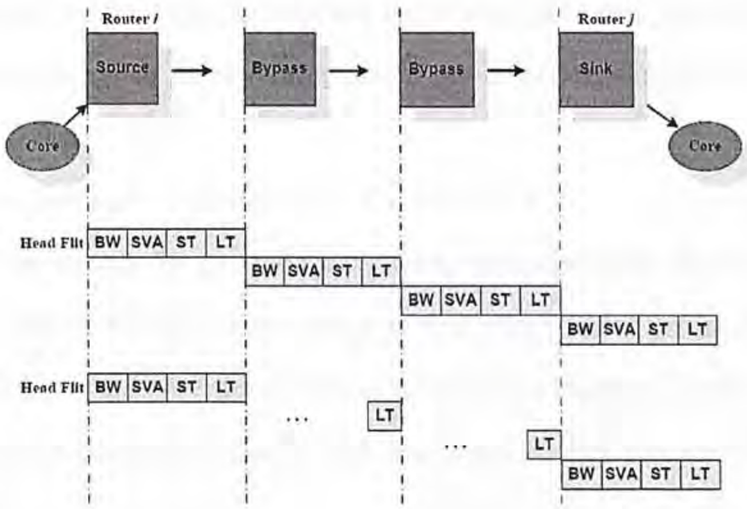
Figure 2-7 EVC Pipeline Comparison


Head Flit
BW
SVA
ST
LT
Body Flit
Tail Flit
(a). Ordinary Network Router Pipeline
(b). EVC Network Router Pipeline, @ source node
(c). EVC Network Router Pipeline, @ bypass node, non-aggressive express
(d). EVC Network Router Pipeline, @ bypass node, aggressive express

Just as shown in

Figure 2-8. A packet injects into network from node *i* with destination of node *j*. In conventional four stage pipelining network, the packet spends four clock cycles at each router, amounts to 16 cycles in total. If, with EVC path inserted, it spends 4 cycles at source node just like before, and then only 1 cycle at each bypass node and 4 cycles at sink node finally. Then the latency spent on the path adds up to 4+1+1+4 = 10 cycles, with six cycles cut on bypass routers. Thus, more routers are bypassed along EVC path, more latency improvement brought in.

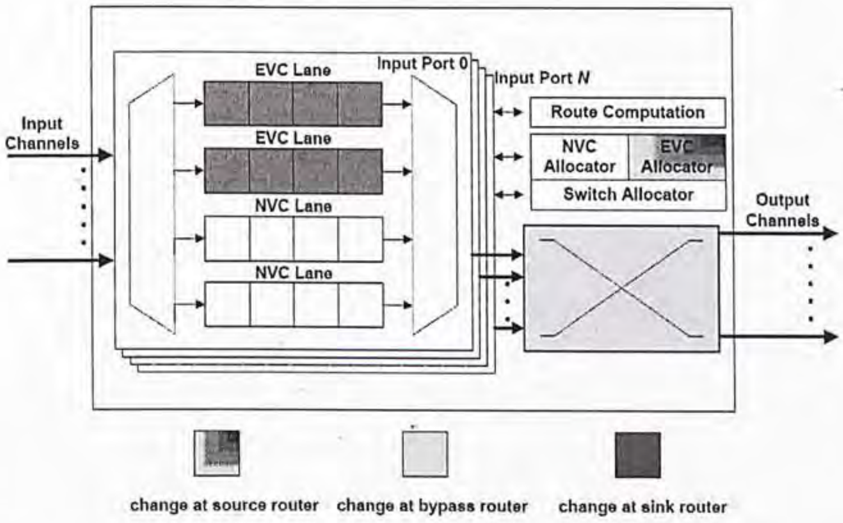
Figure 2-8 EVC Bypass Example



2.3.3.3 Power Consumption

Similar with latency, power consumed by skipped stages is saved to achieve less overall cost. Due to the insertion of EVC, routers of NoC can be categorized into four types, source, sink, bypass and normal, where normal is the one not involved in any EVC path. A router can be source and/or sink and/or bypass as long as no multiple EVC paths overlapped. Various types of router have different architectures. Figure 2-9 displays all possible network router structure. Specifically, EVC allocator is present in source router alone and EVC lane only in sink router, while switch in bypass router is tiny different from that of other types of router.

Figure 2-9 EVC Router Architecture Comparison



Thus in comparison with conventional network router, EVC insertion brings extra components and complicates arbitrate operation. However, the overhead is offset by improvement elsewhere. In general, relevant per-hop energy is given as

$$Enode = Erouter + Elink \qquad (1)$$

$$Erouter = Ewritebuffer + Ereadbuffer + Esvaarb + Exb \qquad (2)$$

where *Enode* is the energy for passing through one particular node, *Erouter* is the energy consumed on passing through router, *Elink* is the energy spent on passing link wire, *Ewritebuffer* and *Ereadbuffer* are the buffer write and read energy, *Esvaarb* is the virtual channel and switch arbitration energy, and *Exb* is the energy required to traverse the crossbar switch.

While EVC flit traveling, it skips router pipeline at bypass router without being buffered or evoking arbitrator. If non-aggressive bypass scheme adopt, switch traversal remains and the pipeline stage decreases to two. If aggressive bypass scheme adopt, switch traversal is further skipped as well then power consumption is significantly reduced to approach *Elink*. Equation (3,4) express energy constitution for bypass router when EVC flit going through.

$$Enode = Exb + Elink \qquad (3)$$

$$Enode = Econnect + Elink \qquad (4)$$

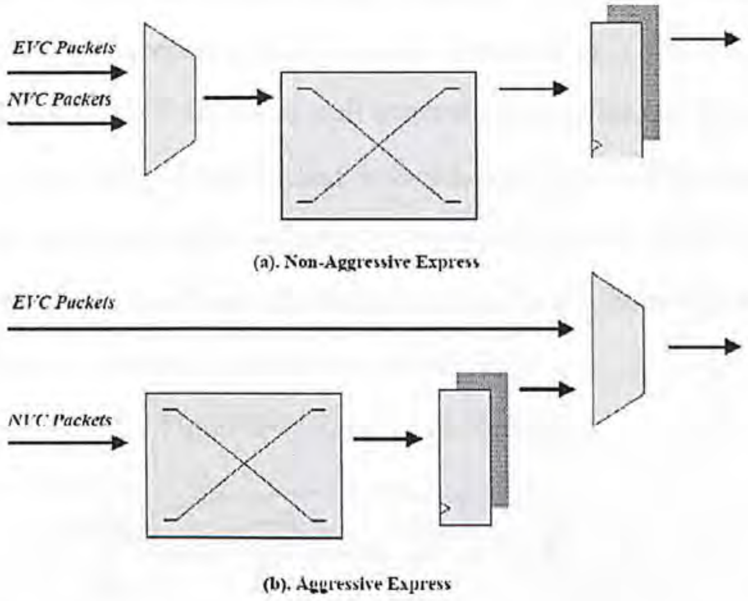
Econnect represents energy consumed by wire within network router when aggressive express applies. The actual energy of *Econnect* is tiny but specified here to distinguish from energy exhausted by passing switch itself. The bold line shown in

Figure 2-10 is the bypass wire for aggressive bypass scheme.

[illegible]

[illegible]

Figure 2-10 Comparison of Non-Aggressive and Aggressive Express



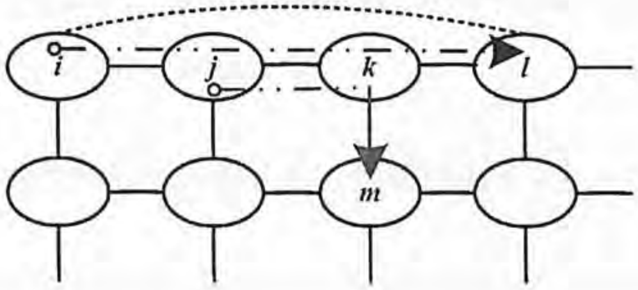
2.3.3.4 Deadlock Avoidance

EVC insertion has an underlying assumption that EVC packet must be in greater amount or is expected to reach destination as early as possible. Granting higher priority ensures enhancement to this expectation. Meanwhile underlying hazard could hurt network performance if no extra mechanism introduced. For example, if during a relative long period, EVC packets are continuously ejecting out from source node and NVC packets sharing the same path are injecting into network simultaneously. Then the NVC packets are blocked constantly until all EVC packets complete transferring. This is called deadlock which brings in substantial and unacceptable system delay.

Figure 2-11 indicates a case of two flows, sourcing from node *i* to node *l* and from node *j* to node *m* respectively, and EVC path is inserted from node *i* to node *l*. Assuming within a certain interval, node *i* and node *j* keep ejecting packets to their destinations. Since they both share the path from node *j* to node *k*, only one flow can be arbitrated to pass through west output port of node *j* at a single cycle. Due to the higher priority of EVC packets, the packets from node *j* to node *m* will be blocked indefinitely. Output flits from node *j* would be "*EEEE...*" where "*E*" represents EVC flits.

The scheme to eliminate deadlock is to set upper limit duration of continuous EVC flits transfer. Whenever source node ejecting EVC flits out a certain number of clock cycles in a row, succeeding EVC flits are forced to give way to the NVC flits. Continuous transfer of NVC flits is as well constrained to a fixed number of cycles for similar reason. When upper limit touched, it switches to transfer EVC flits instead. If it allows EVC flits to be transferred at most 4 in a row, and then 2 cycles are reserved for NVC flits, the deadlock avoidance parameter is set to be 4:2, output packets should be "*EEEENNEEEENN*..." where "*N*" represents NVC flits.

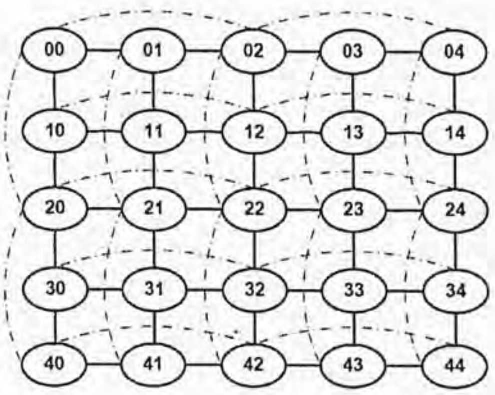
Figure 2-11 Dead Lock Example



2.3.3.5 EVC Insertion Strategy

The actual efficiency of express virtual channel largely rests with distribution of EVC paths and characteristics of applied traffics. In general, there are two approaches of EVC insertion. The first is uniform distribution, and the second is custom distribution. Figure 2-12 depicts uniform EVC insertion for 5×5 mesh network. The only insertion rule is EVC paths overlapping prohibited.

Figure 2-12 Uniform EVC Insertion for 5×5 Mesh Network



Uniform is just an abstract notation indicating that EVC paths are roughly distributed uniformly rather than a mandated principle to follow. For instance, given 4 ×4 network, designating (0, 0) -> (0, 2) as EVC path essentially has no difference from designating (0, 1) -> (0, 3). As no analysis flow proposed yet, custom distribution is also a general conception and manual analysis and tentative planning still works to some extent. [12] proposed a greedy EVC insertion algorithm for low-power objective. It models energy savings with regard to any possible EVC path inserted in the network and ranks them in descending order. Then iterative insertion process evokes to insert EVC path in sequence until insertion rule violation occurs. Although current insertion rule and insertion approach do not guarantee ideal EVC insertion outcome, evaluation experiments still shows considerable results.

2.4 Low-Latency Oriented Architecture

Low-latency architecture from NoC library shares the same network topology, routing algorithm, buffer configuration as well as allocator design with low-power architecture. Configurable parameters of low-latency architecture is less than that of low-power.

Table 2-2 Features of Low-Latency NoC Architecture

Topology	2-D Mesh
Routing Algorithm	X-Y
Buffer	Size Configurable
Pipeline Stage Number	4
Types of Router	1
Low-Latency Schemes	Look-ahead Bypass

2.4.1. Lookahead Bypass Scheme

In NoCs, the width of control signal is usually smaller than that of payload data. After obtained all the necessary control signals from its neighborhood, the router can complete all the allocations. Since the allocation computation is ahead of the switch traversal of payload data in the router pipeline, the control signal is required earlier. Similarly, suppose generation of control signal completes earlier, the control signal can be transferred to the next router in advance too. Once router completes allocation early enough, payload can bypass the input buffer and go across the router's switch directly. This is called lookahead bypass (LA-BP) scheme which is dynamically determined for each incoming flit.

2.4.2. Lookahead Bypass Router Architecture

The NoC architecture in [13] that uses lookahead bypass mechanism to speed up arbitration with small penalty along critical path is integrated into proposed platform as the low-latency architecture. Once lookahead bypass enabled, flit latency at single node reduces from four to two. Although it sounds like less improvement in comparison with express virtual channel that minimizes latency to one clock cycle utmost, it brings no adverse effect to other flits and results far significant enhancement in terms of network latency.

Figure 2-13 Lookahead Bypass Router Architecture

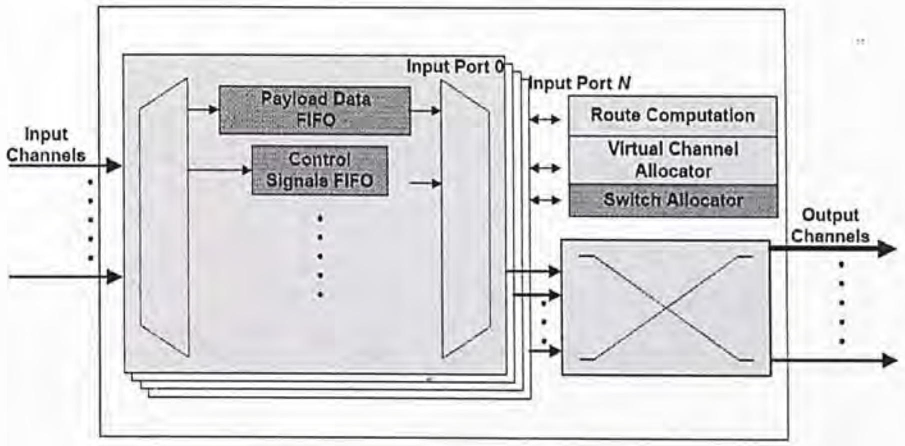


Figure 2-13 illuminates architecture of lookahead bypass router. The lookahead bypass router is capable of handling two types of flits: lookahead and normal ones, so that additional components are integrated into the normal router. A distinctive feature of lookahead bypass router is that routing information is separate from payload in router buffer. Therefore, input buffer is divided into two types. Payload FIFO stores flit payload while control signal FIFO stores control signal.

Switch allocator is another module redesigned for lookahead bypass. Since all flits need to send control signals to downstream router ahead of payload, the traversal of control signal is integrated into the pipeline stage of switch allocation. For the purpose of acceleration, two stage speculative control signal multiplexers are applied to switch allocator. If a flit wins the first stage arbiter, which arbitrates requests from the same input port, its control signal will be transferred by the first stage multiplexer instantaneously. Virtual channel allocator is similar with that of normal network router, while crossbar of lookahead router only delivers flit payload.

The essence of low-latency NoC architecture is to parallelize some arbitrate computation to reduce network router latency. Evaluation result shows distinctive improvement for various traffic characteristics. When injection rate is relatively low, low-latency network router brings in significant latency reduction; with heavy traffic, it as well shows enhancement to some extent.

Chapter 3 Benchmark and Measurement

The ultimate objective of NoC design is to run real applications on spread cores interconnected by network. If so, processors and cache structures are supposed to be taken into account for real circumstance evaluation. This entire process is so complex that plenty of efforts and time for functionality partition, processing elements development and network integration is demanded. Particularly, due to the complexity of real application, it is infeasible to track traffic or analyze capability of contention resolution. Therefore, it is more realistic to implement real application later on and to use synthetic benchmark for rough NoC evaluation at first. The synthetic traffic can capture the salient aspects of the application-driven workloads, and is easier to design and manipulate [21].

Synthetic benchmark is typically characterized and constructed along three dimensions: spatial distribution, temporal distribution and message size specification. Spatial distribution governs the spatial traffic pattern: who communicates with whom; temporal distribution describes the message generation probability over time while message size specification defines packet length as well as packet width [14].

Simple synthetic benchmark leads to easy measurement process and less time consumption. On the contrary, detailed consideration along all the three dimensions brings accuracy at the price of more efforts in performance analysis [29]. Proposed platform provides four types of synthetic benchmark to evaluate NoCs, including random uniform, matrix transpose, locality traffic and full custom. Each type of benchmark is implemented by corresponding traffic generator.

Traffic generator is the substitute of processing element to generate synthetic benchmark. Each generator associates with one processing element. It is controlled to eject data outwards. The data comprises no actual meaning but comply with particular statistics characteristic. Traffic generator is categorized by spatial distribution alone while temporal distribution of all traffics is manipulated by pseudo-random scheme. It generates traffic randomly in every single cycle but certain communication probability is satisfied if measurement interval is prolonged. The longer statistics interval, more accurate final communication probability is as pre-set value. Term definitions are listed as below.

Pi: communication probability of node *i*, to indicate the likelihood of sending messages to the network from node *i*. It is measured by number of packets ejected out every cycle.

***Dest*(*i*)**: an array of numbers, each element represents index of one node that receives messages generated from node *i*

***flow*(*i, j*)**:messages entering network from node *i* and leaving network at node *j*

***IR*(*i, j*)**: injection rate of *flow*(*i, j*), to indicate likelihood of occurrence of *flow*(*i, j*).

$P_i = \sum IR(i,j),\ \ j \in Dest(i)$

3.1 Benchmark Generation

3.1.1 Types of Traffic Patterns

Four traffic patterns are available in proposed NoC platform that are uniform random, matrix transpose, locality and full custom. Uniform random and matrix transpose are categorized as classic traffic patterns because they are fundamental ones for NoC evaluation. Uniform random is the most balanced traffic which emulates the best case of data communication among all nodes. It reflects the capability of target network but also hides network sins such as poor routing algorithm. Matrix transpose, on contrary, is extremely unbalanced and tends to get congested as communication probability increases. It predicts the worst capability of a network. Locality traffic pattern evolves from uniform random, but takes into account of the relation between transfer distance and communication probability. It reflects the fact that function components of heavy data exchange are always mapped localized. Finally, full custom traffic pattern allows any form of traffic spatial distribution. It is able to approximate any realistic application.

Table 3-1 Comparison Among Four Traffic Patterns

Traffic Pattern	Pros	Cons
Uniform Random	easy to implement and analyze;	best case, seldom occur in reality; network sins hidden;
Matrix Transpose	easy to implement and analyze;	worst case, seldom occur in reality; network sins magnified;
Locality	function components mapped localized;	treat nodes of identical distance equally;
Full Custom	emulate any traffic spatial distribution	difficult to implement and analyze

3.1.1.1 Classic Traffic Patterns

Certain types of traffic pattern are considered as first resort to evaluate a newly designed NoC, e.g. uniform and matrix transpose. They are both fundamental synthetic benchmarks to evaluate NoCs, which differ from each other along spatial dimension.

Uniform traffic pattern assumes that any node of the network has exact the same probability to send messages to any other nodes. Thus the entire traffic is strictly, uniformly distributed within the network. Uniform random traffic is quite benign because it naturally balances load across network channels, very close to the best-case workload.

Matrix transpose traffic pattern otherwise is one of the permutation type, which is seriously congested along certain paths. As network size increases, it deteriorates further. For instance, suppose X-Y routing algorithm adopt with injection rate of each node equals to 0.2 flit/cycle, then the total traffic across node (0,0) is 0.6 flit/cycle in 4×4 mesh network and 1.0 flit cycle in 6×6, which means the channel is saturated. Matrix transpose traffic pattern allows for extreme spatial distribution and magnifies the sins of network design.

Both traffic patterns are widely used due to easiness of implementation and reflection of NoC performance under extreme traffic characteristics. Allowing for both simultaneously delivers a relatively complete overview of NoC performance.

3.1.1.2 Locality Traffic Pattern

Classic traffic patterns either broadcast to every point of the network, or assign single destination for individual flow, overlooking the fact that function components of heavy data exchange are always mapped localized. A unified expression that is allowed to adjust the locality so as to analyze the network behavior under locality traffic is presented. By setting locality factor with distance *d* for each individual node, amount of traffic sending to nodes of certain distances is finely controlled.

According to the literature, locality traffic pattern has to follow two rules, first, the probabilities of a node sending message to the nodes of identical distance are exactly the same; second, this probability is determined by locality factor array *α(d)*. Given *α(d)*, communication distribution probability *DP* is calculated by Equation (5-6).

$$coef(d) = 1 + \frac{\alpha(d)}{d+1} \tag{5}$$

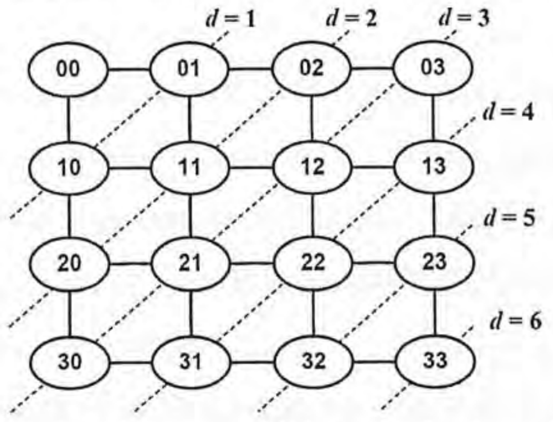
$$P_c = \frac{1}{\sum_{d=0}^{D} N(d) coef(d)} \tag{6}$$

$$DP(d) = coef(d) P_c \tag{7}$$

Where *coef(d)* is the distribution coefficient for a node with distance *d*; *N(d)* is the number of nodes of distance *d*; *D* is the maximum distance between the target node and any other node within the network; *Pc* is the common probability factor; *DP(d)* is the communication distribution probability.

For instance, according to Figure 3-1, if *α* fixed to constant 1, to determine the communication distribution probability of node (0,0), for its distance array [0, 1, 2, 3, 4, 5, 6], we obtain number of destination nodes with identical distance is [1, 2, 3, 4, 3, 2, 1]. By Equation (5), we obtain the coefficient array *coef(d)* of node (0,0) as [2, 1.5, 1.33, 1.25, 1.2, 1.166, 1.143]. By Equation (6), we calculate $P_c = 0.0474$. Thus *DP(d)* = [0.0948, 0.0711, 0.0630, 0.0593, 0.0569, 0.0553, 0.0542]. A specific case is *α* of all nodes being fixed to 0, then *coef(d)* is fixed to constant 1 for all distance, locality traffic pattern turns into random uniform traffic pattern.

Figure 3-1 Distance Graph for Locality Traffic Pattern



Since *DP(d)* is no less than zero, valid range of α depends on distance *d*, and it can be basically divided into four intervals (or points).

$\alpha = -(d+1)$ => *coef(d) = 0*

$0 > \alpha > -(d+1)$ => *1 > coef(d) > 0*

$\alpha = 0$ => *coef(d) = 1*

$\alpha > 0$ => *coef(d) > 1*

Communication probability of any flow is multiple of common probability factor. Thus selection of proper α value determines the approximate domain of final probability.

3.1.1.3 Full Custom

Full custom traffic pattern implies that any new characterized traffic pattern could be simulated by platform for more specific purposes. For instance, the generic traffic pattern which takes into account temporal burstiness [14] can be added to enhance platform capability of emulating real application of significant temporal distribution. Any imitation of real application by statistics can also integrated into proposed platform in this manner. Manual configuration through graphical interface flow by flow is required to import traffic pattern.

3.1.2 Traffic Generator

As synthetic benchmark applied to NoC exploration, the processing element which is supposed to be processor, memory, etc. in real application is replaced by other module named traffic generator which involves with no data processing but is capable of generating traffic of specified characteristics. In the meantime, network interface is simplified to FIFO [17].

The data produced by traffic generator is formatted as network requires then packetization module in conventional network interface becomes dispensable. FIFO, as a module within network interface, buffers the excessive messages when network is incapable of dealing with plenty of data timely. FIFO in our design is of very large volume to store thousands of packets, which is obviously not realistic. The reason of it is that data dependence is ignored in traffic generator case, and no stop signal would have ever been issued to pause traffic generator sending data out. Thus sufficient FIFO volume is necessary to prevent overflow or message drop as congestion being severe in some cases. If, which is rare, the FIFO is filled, it indicates that either faulty exists or current NoC design is incapable dealing with such amount of traffic. Structure of FIFO is simply the fundamental synchronous write and asynchronous read one. Since only simulation proceeds in platform operation, huge size of FIFO is not a concern. Normally, processing elements interact with network by receiving messages. Modules for receiving messages are omitted here because no further processing involved except that relevant packet information is written into text files for statistics.

The essence of traffic generator is to configure all flows sourcing from current node. Each flow is determined by three factors: source node, destination node and injection rate. Every node involved in sending data to network would mount a traffic generator and a FIFO. As source node of flow is default determined, traffic generator requires data from platform users to clarify destination node and injection rate of each flow. Specific design of each type of traffic generator is presented below.

- **Random Uniform**

The "Uniform" feature of random uniform traffic pattern is guaranteed by two rules: *Dest(i)* includes all the nodes of the network other than i , so if given mesh network $m\times n$, size of *Dest(i)* equals to $(m\times n-1)$; secondly, assign identical value to any $IR(i, j)$ or $IR(i, j)\equiv$Constant, $j\in Dest(i)$. Sum of injection rates of all flows amounts to communication probability of node i, $Pi = \Sigma IR(i, j)\equiv$Constant, $j\in Dest(i)$.

Since on hardware level, traffic generator associates with specific node and is irrelevant with that of any other node, proposed platform further extends it to allow communication probabilities of different nodes differ from one another. Therefore, communication probability of each node is read specifically and applied to configure associated traffic generator. Zero communication probability $Pi = 0$, namely no message sending to the network from current node, is permitted as well. Provided that, no traffic generator is connected to corresponding network router.

- **Matrix Transpose**

In comparison with random uniform, spatial distribution of matrix transpose pattern is concise as at most one unique flow exists for each node, with index of destination node obtained by unified expression. Nodes along diagonal line are excluded that they send and receive no message. Thus size of *Dest(i)* = 0 *or* 1. Matrix transpose pattern here is tailored to apply for non-square mesh topology as well. The nodes, whose destination node coordinate beyond network range, are excluded from data exchange. Given mesh network 6×5, then *node(5, x)*, $x = 0\ldots4$ are excluded from communication (suppose node index labeled from (0, 0)) which makes it virtually 5×5 network. Like random uniform, communication probabilities of valid nodes can be identical or distinctive, depending on desired traffic characteristics.

- **Locality**

In the case of locality traffic pattern, size of *Dest(i)* equals $(m\times n-1)$ as well if given network $m\times n$. These destination nodes are further grouped by distance from node i, *Dest(i,d)*. Upon selection of network size, group numbers and destination nodes associated with each group are calculated automatically by the platform. Users are supposed to fill locality factor array $\alpha(d)$ for each node, or unify them under the identical $\alpha(d)$. *coef*(*d*) and P_c are not directly used in traffic generator but combined to obtain total injection rate of group *IR*(*i, d, -*). The method is feasible because all flows within a group share the same injection rate. Picking up any node randomly among destination nodes of the group ensures exact individual *IR*(*i,d, j*), $IR(i, d, -) = \Sigma IR(i,d, j), j \in Dest(i, d)$.

- **Custom**

Full custom traffic pattern involves largest number of parameters to set. All the *Dest*(*i*) and *IR*(*i, j*) are configured manually one by one. This process is error-prone and there is no way to find them out automatically. Custom traffic generator is quite similar with locality as if *group* is not featured by distance. For instance, if node *i* from 4×4 mesh network is sending messages to all other nodes just like random uniform, but with all distinctive injection rates. Then we get size of *Dest*(*i*) equals 15, and *IR*(*i, j*), $j \in Dest(i)$ are all independent. It seems like slicing all flows sourcing from node *i* into 15 groups, each of which comprises with unique flow and is of particular injection rate.

3.2 Measurement Setting

After synthetic benchmark configured and applied to NoC with simulator evoked, simulation proceeds until all simulation data is collected. Then platform starts to conduct measurement.

3.2.1 Warming-up Period

Generally network is initialized with empty buffers and idle resources before packets injected in. Thus, packets that are injected early in the simulation confront less contention and traverse more swiftly. However, as buffers begin to fill up, later packets confront more contention with increased latency. The influence of initialization fades over time and at some point the simulation is justified to be warmed up.

Take warming-up period into account ensures that packets to be measured are all within steady state, otherwise systematic inaccuracy is introduced in. Although real application running on network also initializes with empty buffers, it always costs a long time before completion. In terms of synthetic traffic pattern, only statistic features are emulated with relatively short simulation interval to obtain evaluation outcome much faster. That is why warming-up period is critical for measurement.

In the domain of synthetic benchmark, standardized definition yet exists for warming-up period. All literatures choose to set warming-up period empirically and the exact setting is always unrevealed. Range of warming-up period varying from one literature to another makes designs from different authors incomparable.

Normally, warming-up period can be measured by clock cycle or packet number. The first approach is that, only packets injecting into network after certain number of system clock cycle are counted valid in measurement. Second approach is to start measurement after certain number of packets having completed transferring in network. Proposed platform supports both modes and records the setting for future comparison for reproduction.

3.2.2 Latency Definition

Network latency is a common metric for runtime estimation of NoC applied synthetic benchmark, but its definition is ambiguous. It can be measured per data word, header, packet, or transfer (with several packets as a block) while include or exclude the latency at FIFOs. In proposed platform, two types of latency are defined, called latency without source queue and latency with source queue, both with packet as data unit. By unifying and explicitly stated network latency, different NoCs can be compared under identical standard

$$Latency_L = Time_{out} - Time_{gen} \qquad (8)$$

$$Latency_S = Time_{out} - Time_{in} \qquad (9)$$

Where, $Time_{gen}$ indicates the time of packet generation, $Time_{in}$ is the time packet injecting into the network and $Time_{out}$ the time of ejecting out from network.

3.2.3 Throughput Definition

Network throughput is the metric to evaluate NoC bandwidth. By comparing throughput of messages injecting into and ejecting out from network, congested flows are distinguished out.

$$Throughput = Packet\_Num_{tot} / Time_{tot} \qquad (10)$$

Where $Time_{tot}$ represents the time interval for measurement, in the unit of clock cycle. $Packet\_Num_{tot}$ indicates the total number of packets received by the network (amount to all packets received by any node within the network).

3.2.4 Virtual Channel Utilization

Buffers at each port of router can be configured to identical size, or customized according to estimated utilization circumstances. As buffer in architectures of NoC library is divided into virtual channels, the utilization issue turns to how much virtual channel space is actually in use during simulation period. Specifically, how many virtual channels or how many flit-units of each virtual channel are frequently active. Buffer utilization gives helpful insights into optimal link capacity. It assists to distinguish which physical link is congested and which is of excessive buffers that can be removed for less overhead. Utilization rate, in terms of virtual channel number and virtual channel depth are monitored during the simulation.

Chapter 4 Platform Structure

The platform proposed is developed in Perl with NoC library described in System Verilog. It operates on Windows System due to Win32 GUI package [28] used for graphical interface. Two third-party tools are required for platform operation, ActivePerl and QuestaSim. ActivePerl 5.10.0 is now used to interpret Perl for GUI generation as well as processes operation. QuestaSim 6.3f is utilized as simulator to run System Verilog simulation. Proposed platform implements the basic functionality shared by all other platforms ranges from NoC configuration, traffic generation, simulation to performance evaluation, with unique functionality of buffer optimization and NoCs comparison.

The platform manipulates operations through graphical interface that exhibits notable superiority over rival platforms. Innovative network architectures emerge these days cover many configuration parameters. To comprehend meanings of it all and their impacts on network performance demands technique files and source code reading that is tedious and time-consuming. On the other hand, configuration process of other platforms is commonly realized by source code alteration that is hazardous to integrity of NoC library. Given the intention of efficient utilization of available NoCs rather than improvement of them, development of graphical interface that displays relevant parameters with other design details hidden is reasonable and sensible. It shortens configuration process and guarantees integrity of NoCs library. Through the guidance of instruction documents, quick configuration of an application-specific NoC architecture becomes feasible.

The choice of NoC configuration is always on the basis of rough estimation or empiricism that often misses the target. Few system feedback is provided in other platforms thus every parameter fine-adjustment is by guess and leads to unworthy trial. On the contrary, proposed platform provides optimization technique applied to users' initialized network architecture that helps to reach final desirable NoC configuration in a short time. It is a significant feature distinguishing proposed platform from others that never address on approaches of performance improvement or cost reduction.

Another unique feature of proposed platform is NoCs comparison mechanism. Typical comparison among NoCs turns to numeric value comparison of computed network latency or throughput without checking the preconditions. All literatures are reporting their designs of decent improvement over benchmarks while much critical information on NoCs is undisclosed leaving validity of comparison outcome in doubt. Proposed platform sets up an intact flow for NoCs comparison taking into account of consistency of traffic and measurement setting. That ensures the various NoCs for comparison are evaluated by equivalent standard.

This chapter organizes as follows. Firstly, it describes file trees of the entire platform, giving details of specific directories and files. It assists to sort out the whole backstage operations of the system and to comprehend the way of debug, whenever error occurs and of platform upgrade, by integrating additional optimization tools or NoC architectures. The second part elaborates graphical interface with an example follows, and the third is introduction of backstage operations.

4.1 File Tree

Two directories are critical to proposed platform. System installation directory which comprises all the scripts and files for platform build-up and output directory which saves files relevant with NoCs to evaluate. Suppose the platform is transplanted to a new system and the installation directory is $*platform_path*. Sub-directories and major files below are shown in Figure 4-1.

The operation by proposed platform is organized by *project* and *test*. Whenever a new a new NoC architecture configured by users, a new project sets up with user specified project name. When a new traffic pattern is chosen to evaluate the NoC architecture, a new test sets up also with a user specified test name. If the directory assigned for project is $*project_path*, then associate directory for specific project test is named as $*project_path/project_test_lib/*. Multiple test folders are allowed under the same

project directory. File tree of project test displays in

Figure 4-2.

Figure 4-1 Platform File Tree

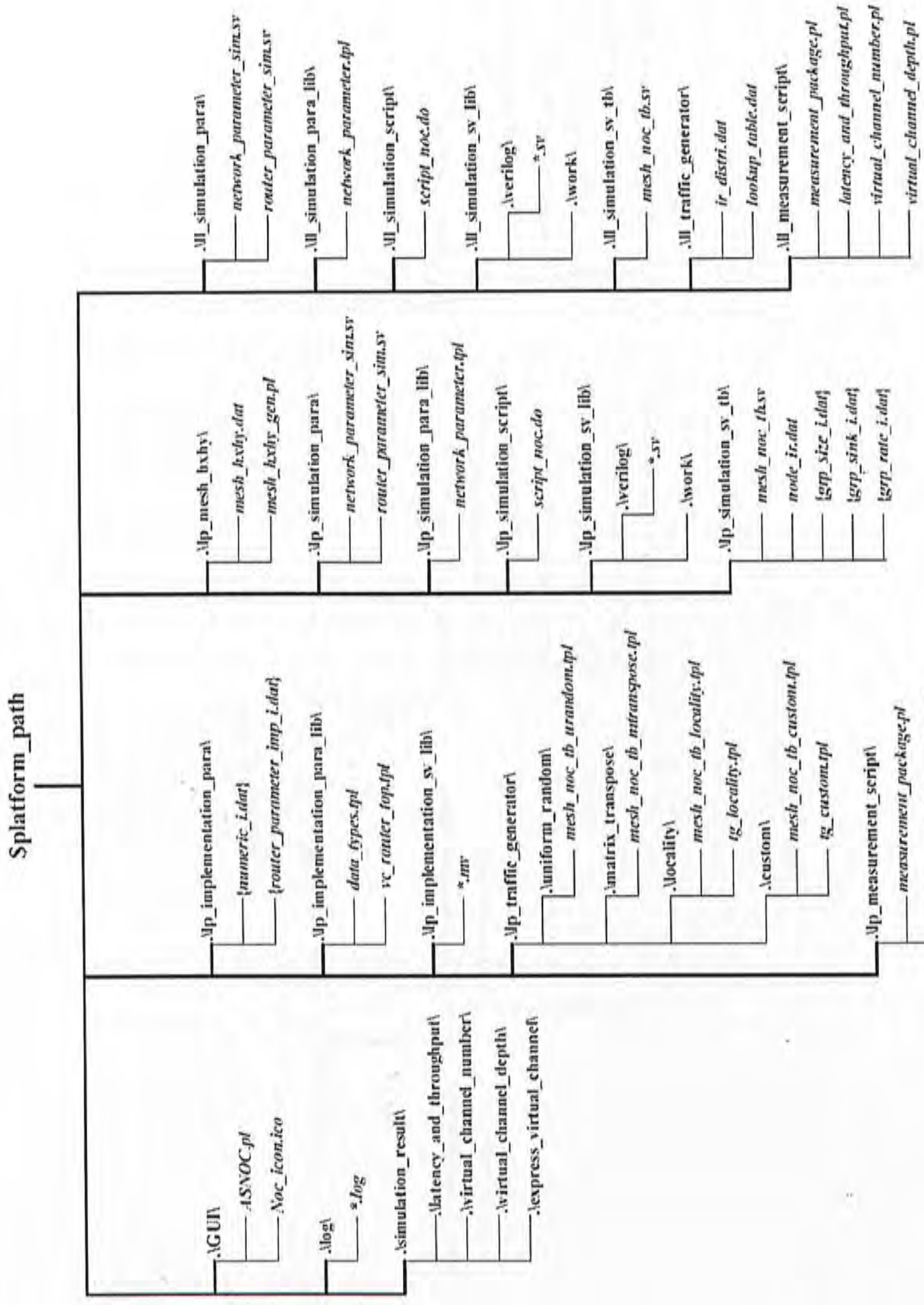


Figure 4-2 Project File Tree

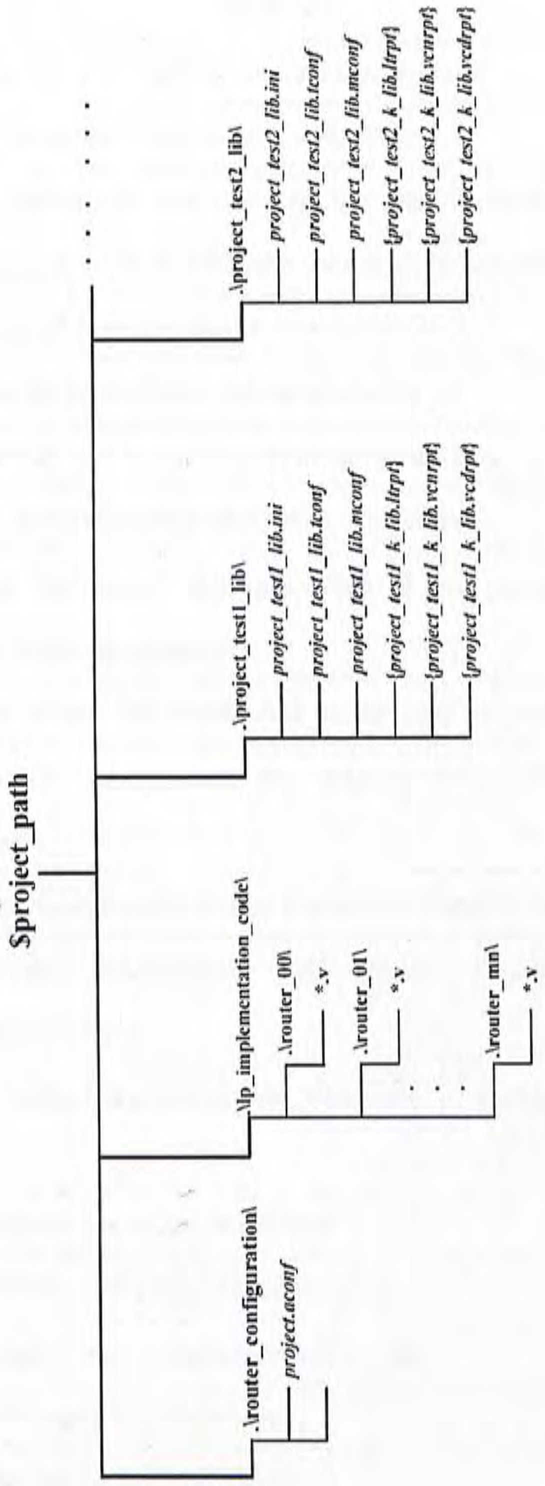


Table 4-1 Platform File Types

File Type	Description	Status
.pl	Perl script file, either for GUI generation or processes	c
.sv	System Verilog file, describes NoC architecture	c
.mv	System Verilog file with modified data type involved, used to generate single synthesizable router description for low power NoC	c
.v	Exported implementation code for low-power NoC	o
.do	Script file for QuestaSim to conduct simulation	v
.tpl	Template file, read by process to produce specific files	c
.log	Log file, records operation information of processes	v
.cmp	Compare file, record basic information of two *project* & *test* involved in system comparison	v
.ini	Initial file for *test*, link to associated *.mconf*, *.tconf* and *.rconf*	v
.aconf	Architectural configuration file, records NoC configuration information	o
.tconf	Traffic configuration file, records information of applied traffic	o
.mconf	Measurement configuration file, records information of measurement setting	o
.rconf	Report configuration file, records information of associated report files	o
.ltrpt	Report file for latency and throughput	o
.vcnrpt	Report file for virtual channel number utilization	o
.vcdrpt	Report file for virtual channel depth utilization	o
.crpt	Report file for NoCs comparison	o
.fm	Flow map file for specified traffics	v
.dat	Data file, generated by simulation and contains packets information in transfer	v

* **c:** fixed content; **v:** to be updated by system operation; **o:** output files for user recording or further utilization

File type involved in platform and related description are elaborated explicitly in Table 4-1. Several file types are relevant with associated tools while others are defined by proposed system. The definition of these file type distinguishes one from another. While the status of file type indicates if specific file is of fixed content, or is temporary files by system operations or the final output files. Certain types of file are recognized by specific system operations which will be elaborated later.

Files under *$platform_path* fall into three categorizes, system file, low-power NoC file and low latency NoC file. Whereas files under *$project_path* fall into architecture file and test file. Details are elaborated in following sub sections.

4.1.1 System Files

This part consists of files apply to either NoC architecture and will not be affected by more architectures to be integrated in the future.

- **GUI**

Single Perl script *ASNOC.pl* interprets the entire graphical user interface and almost all operations. Graphical user interface displays with all backstage operations activated by clicking *ASNOC.pl*. *noc_icon.ico* is a picture used for platform icon.

- **log**

Once system operations are evoked, corresponding log files are opened and written, no matter operations succeed or terminate due to some errors. All log files are saved under this directory with only the latest status recorded. These files are always neglected unless unexpected error occurs or debug is underway.

- **latency_and_throughput**

{*input_sq_time_i.dat*}, {*input_network_time_i.dat*}, {*output_network_time_i.dat*}, are written when each packet is generated, sent into the network and sent out from the network, measured in clock cycle, in terms of low-power NoC architecture. Similarly, {*packetin_i.dat*} and {*packetout_i.dat*} record contents of each packet injecting in and ejecting out at every node sequenced in time order. These two files are used to verify the correctness of network communication. Warning alarm issues if packet dropped or mis-delivered. *{sq_packet_num_i.dat}* keeps track of number of packets waiting in FIFO (network interface) every clock cycle. It assists analysis of congestion and traffic balancing. In terms of low-latency architecture, files like *{send_i.dat}*, *{recv_i.dat}* and *lt_result.dat* are generated and saved, with *lt_result.dat* records the computed average latency and throughput outcome.

- **virtual_channel_number**

  {vc_active_ij.dat} records number of virtual channel being active at each clock cycle. Since virtual channel status is updated every clock cycle, producing these files lengthens actual simulation duration significantly. It recommends to select target input port prudentially. *vc_utilization_number.dat* summarizes statistics of virtual channel number utilization.

- **virtual_channel_depth**

  {vc_credit_ij.dat} records number of virtual channel units being active at each clock cycle. Number of router input ports to be monitored for virtual channel depth is also to be selected prudentially. *vc_utilization_depth.dat.* also summarizes statistics of virtual channel depth utilization.

- **express_virtual_channel**

  This folder applies to low-power architecture only. *{tailarriving_time_ij.dat}* and *{tailleaving_time_ij.dat}* are used to check if every EVC packet bypasses the EVC path, where *_ij* indicates particular source node input port or sink node output port.

4.1.2 Low-Power NoC Related

Directories under this category are all related with projects using low-power NoC architecture. The *lp_* of directory name indicates low power. Further, name of the file under particular directory might comprise of *_i*, or *_ij*. If not specified, *_i* indicates the file is associated with a particular node whose index within the network is *i*. *_ij* indicates current file is associate with a particular port of particular node, where index of the node is *i* and index of the port is *j*. Files with *_i*, or *_ij* are always embraced in a bracket, suggesting it is a set of files, one file for each router in terms of _*i* and one file for each port in terms of _*ij*.

- **lp_mesh_hxhy**

    The top module of low power network architecture requires a data file, which is *mesh_hxhy.dat* in current system, to preserve relative coordination of any pair of nodes within the network. It embeds into packet to guide the transfer. As supportive network size of this platform restricted to maximum 10×10 (refer to section 4.2), the number of possible *mesh_hxhy.dat* files is 45 (suppose minimum network 2×2). However, none of these files are generated upon system installation. Instead, *mesh_hxhy_generate.pl* that is capable of generating desired content for any size of mesh topology network is put there which offers more flexibility for future extension. Every new NoC project creation leads to *mesh_hxhy.dat* overwritten.

- **lp_simulation_para**

    network_parameter_sim.sv and *router_parameter_sim.sv* are two parameter files to define NoC configuration and for NoC simulation. The first file contains information such as network size, packet length and selected sub-modules. While the second includes virtual channel number distribution, EVC path setting, etc.

- **lp_simulation_para_lib**

    File *network_parameter.tpl* saves the values of all fixed parameters for low power NoC architecture. Its contents are required to be copied and supplemented to each updated *network_parameter_sim.v*.

- **lp_simulation_script**

    script_noc.do in *Tcl* language is the script called by QuestaSim for simulation. Reference of QuestaSim provides details of its content. It also determines simulation interval and must be rewritten for new test bench.

- **lp_simulation_sv_lib**

    Low-power NoC description in System Verilog is under *./verilog/* directory and *./work/* is the work folder for QuestaSim simulation.

- **lp_simulation_sv_tb**

    mesh_noc_tb.sv is the final test bench customized through GUI which could be any type of the four traffic patterns. *node_ir.dat* contains communication probability of each network node *Pi*. {*grp_rate_i.dat*}, {*grp_sink_i.dat*} and {*grp_size_i.dat*} reflect the group information for each node in the case of locality. {*grp_rate_i.dat*} and {*grp_sink_i.dat*} are present for custom traffic pattern since group size is fixed to 1.

- **lp_traffic_generator**

Four sub-directories are under and each represents a type of traffic pattern. Under sub-directory *./uniform_random/* and *./matrix_transpose/* there are one file respectively, named *mesh_noc_tb_urandom.dat* and *mesh_noc_tb_mtranspose.dat*. Otherwise, under another two sub-directories, two files *mesh_noc_tb_locality.dat* & *tg_locality.dat* or *mesh_noc_tb_custom.dat* & *tg_custom.dat* exist. The difference is due to the fact that they have various value of *Group*(*i*).

- **lp_implementation_sv_lib**

The *.mv* files under this directory is also description of low-power NoC but are not completely in any HDL language. The *.sv* files under *./lp_simulation_sv_lib/* are modified by some unique system data type for parameter substitution. Thus, the platform is capable of exporting *System Verilog* description (no parameter involved).of individual network router.

- **lp_implementation_para**

Configuration parameters customized by users fall into two types, the network level parameter, such as network size, data format, and the router level parameter, such as virtual channel number, router type (if EVC enabled). Two data files about these parameters are exported and saved in this directory for each network router upon the completion of NoC configuration. Thus, two parameter files {*numeric_i.dat*} and {*router_parameter_imp_i.dat*} are utilized to configure network router *i*. They are used to substitute system data type declared in *.mv* files and to export the final implementation code for each network router.

- **lp_implementation_para_lib**

For the purpose of parameter substitution, all *.sv* files in NoC library are not completely in System Verilog, but with modification of customized character strings for subsequent replacement. *data_type.tpl* and *vc_router_top.tpl* are used for the substitution.

- **lp_measurement_script**

Saving all Perl scripts for NoC measurement. Each script implements one type of evaluation. Just as file name indicates, *latency_and_throughput.pl* calculates average latency and throughput of entire network as well as every single node, *virtual_channel_number.pl* and *virtual_channel_depth.pl* compile statistics of active number and unit respectively to obtain virtual channel utilization. All the data files required for measurement are from *./simulation_result/*. *measurement_package.pl* contains information of configured NoC architecture as well as measurement setting. All of it would be referred to by other *.pl* under current directory. This file is also rewritten whenever a new test created.

4.1.3 Low-Latency NoC Related

Similar with low-power NoC, the sub-directories for low-latency are initiated with *ll_*. The number of folders for *ll_* is smaller than *lp_* because that the two NoC architectures are developed separately with various interfaces. The platform retains their distinctive features without unifying them under the same standard. Therefore, *ll_* does not have counterpart for *_mesh_hxhy* or *_implementation_* ones. Further, due to different design of network interface, traffic generators also have various structures. Following will only describe the different parts.

- **ll_measurement_script**

Low-latency NoC architecture integrates some functionality of performance measurement into network interface design, thus the number of output files is less than that of low-power. Since statistics such as network latency is also computed upon the completion of simulation, some measurement scripts are not required any more. However, to comply with GUI interface, other scripts are needed to extract useful data from simulation output files and produce formatted files to read and display by GUI. The scripts are still named as *latency_and_throughput.pl*, *virtual_channel_number.pl* and *virtual_channel_depth.pl* but with different contents.

- **ll_traffic_generator**

Low-latency NoC generates any kind of traffic pattern by reading two relevant files named as *ir_distri.dat* and *lookup_table.dat*. Thus, it does not need the four types of traffic generator shown by low-power architecture. But it can still deliver all the possible traffics set by GUI. As configuration completes, system produces correct data files *ir_distri.dat* and *lookup_table.dat*. The operation is implemented in *ASNOC.pl* while current directory saves generated data files alone.

4.1.4 Project Related

$project_path is required to assign each ongoing NoC project. All relevant exported files are organized under this directory. Two tests are involved in the file tree shown in

Figure 4-2.

- **router_configuration**

.aconf file is saved under current directory which comprises of all parameters for NoC architecture. The system is capable of reading *.aconf* to generate all parameter files under *./lib_simulation_para/* and *./lib_implementation_para/*, where *lib* indicates *lp* or *ll*.

- **implementation_code**

This folder applies to low-power architecture only. The synthesizable code of individual network router generated from *.mv* of *./lp_implementation_sv_lib/* is saved here. Every router has its own sub-folder named as *./router_i/*, where *i* is the index of network router.

- **project_test_lib**

Both evaluation results and configuration files for NoCs comparison are located under current directory. It recommends not to move the position of any files, otherwise NoCs comparison might halt due to file missing. Details of these files will be described in section 5.3.

4.2 Processes

Graphical user interface is the only visible object as for platform users, who interact with the interface by input configuration, evoke simulation and view evaluation report. Meanwhile, many invisible operations are underway during the interaction. A number of files are created or overwritten to make the platform operate. All these operations are called *process* in our system and assigned distinctive identical number for reference. Each process, or several processes as a whole is called by actions on GUI such as button click. The process organization clarifies operations and is convenient to refer to.

Processes are categorized into five types: system level, low-power NoC simulation, low-latency NoC simulation, optimization, comparison and low-power NoC implementation. Initial number of process ID suggests the type of specific process.

System level, low-power NoC simulation and low-latency NoC simulation processes are listed by following tables. Other types of processes will be described respectively in particular sections of Chapter 5. The operation each process proceeds is straightforward for comprehension.

Table 4-2 Processes of System Level

Process ID	***Operation***
100	Evoke QuestaSim to Start Simulation
101	Check transfer success
102	Check EVC transfer success
103	Calculate EVC transfer efficiency
104	Generate *.aconf* file
105	Generate *.tconf* file
106	Generate *.mconf* file
107	Generate *.rconf* file
108	Read *.aconf* to produce *network_parameter_sim.sv* and *router_parameter_sim.sv* for the new project
109	Read *.tconf* to produce *node_ir.dat*, {*grp_size_i.dat*}, {*grp_sink_i.dat*}, {*grp_rate_i.dat*} or *ir_distri.dat* and *lookup_table.dat*, and *mesh_noc_tb.sv* for the new project
110	Read *.mconf* to update *measurement_package.pl* for the new project

Process 101, 102 and 103 will not be evoked by any GUI operation. Since traffic generator generates each packet with coordinates of its source node within payload, process 101 checks every packet injecting into network at its associated destination node to ensure they succeed transfer. Suppose *j* is index of a node chosen for check, packets are extracted from *packetout_j.dat* one by one. If source of *packet A* from *packetout_j.dat* is *i*, then system will search the identical payload from *packetin_i.dat*. Unless source of every ejected packet is successful located, transfer success is confirmed. Process 102 checks that if every EVC packet bypasses the EVC path in transfer and spends one clock cycle only at each bypass node. The time of each EVC packet arriving at source node and leaving from sink node are extracted from the files {*tail_leavingtime_ij.dat*} and {*tail_arrivingtime_ij.dat*}, then latency on EVC path is calculated and compared with designated EVC bypass latency. Thus process 101 and 102 are taken into account only when system integrity is at risk, while process

103 calculates EVC transfer efficiency as another evaluation metric. It is reserved for further extension.

Table 4-3 Processes involved of Low-Power NoC Simulation

Process ID	***Operation***
201	Read NoC configurations from GUI section 2, to generate *network_parameter_sim.sv* & *router_parameter_sim.sv* under *$platform_path/lp_simulation_para/*
202	Read network size from GUI section 2, to update *mesh_hxhy.dat*
203	Read synthetic traffic settings from GUI section 3, to generate *node_ir.dat*, {*grp_rate_i.dat*}, {*grp_size_i.dat*} and {*grp_sink_i.dat*} under *$platform_path/lp_simulation_sv_tb/* & *$project_path/project_test_lp/*
204	Read synthetic traffic settings from GUI section 3, read *mesh_noc_tb_*xxx*.dat* and *tg_*xxx*_pattern.dat*, to produce traffic generator for each network router, and then produce *mesh_noc_tb.sv* under *$platform/lp_simulation_sv_tb/* , Where xxx indicates any type of the four available traffic patterns
205	Read parameter of simulation interval from GUI section 3, to update *script_noc.do*
206	Read measurement parameter from GUI section 3, to update *measurement_package.pl*
207	Evaluate latency and throughput, to generate *project_test_lp.ltrpt* under *$project_path/project_testl_lp/*
208	Evaluate virtual channel number utilization, to generate *project_test_lp.vcnrpt* under *$project_path/project_test_lp/*
209	Evaluate virtual channel depth utilization, to generate *project_test_lp.vcdrpt* under *$project_path/project_test_lp/*
210	Read *project_test_lp.ltrpt*, to show on GUI section 4
211	Read *project_test_lp.vcnrpt*, to show on GUI section 4
212	Read *project_test_lp.vcdrpt*, to show on GUI section 4

Table 4-4 Processes involved of Low-Latency NoC Simulation

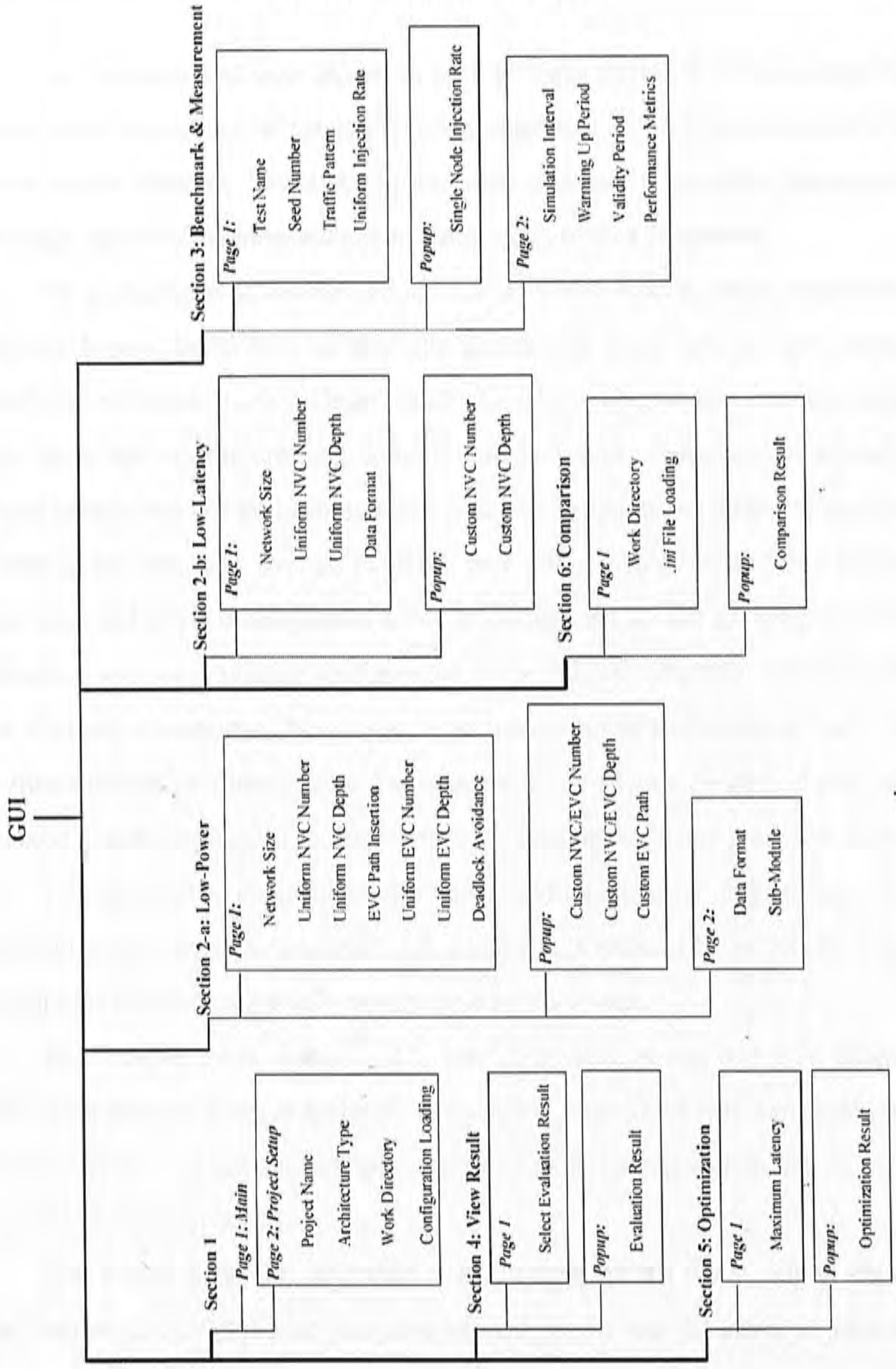
Process ID	***Operation***
301	Read NoC configurations from GUI section 2, to generate *network_parameter_sim.v* & *router_parameter_sim.v* under *$platform_path/ll_simulation_para/*
302	Read synthetic traffic settings from GUI section 3, to generate *injection_distribution.dat* and *lookup_table.dat* under *$platform_path/ll_simulation_sv_tb/* & *$project_path/project_test_ll/*
303	Read parameter of simulation interval from GUI section 3, to update *script_noc.do*
304	Read measurement parameter from GUI section 3, to update *measurement_package.pl*
305	Evaluate latency and throughput, to generate *project_test_ll.ltrpt* under *$project_path/project_testl_ll/*
306	Evaluate virtual channel number utilization, to generate *project_test_ll.vcnrpt* under *$project_path/project_test_ll/*
307	Evaluate virtual channel depth utilization, to generate *project_test_ll.vcdrpt* under *$project_path/project_test_ll/*
308	Read *project_test_ll.ltrpt*, to show on GUI section 4
309	Read *project_test_ll.vcnrpt*, to show on GUI section 4
310	Read *project_test_ll.vcdrpt*, to show on GUI section 4

Table 4-3 and Table 4-4 brief the processes involved in NoC simulation. Most processes of each category implement similar functionality. While they are assigned different process ID because different files are read or files are written into various folders. The number of low-latency related processes is two less than that of low-power because *mesh_hxhy.dat* is dispensable and generation of *mesh_noc_tb.sv* is simplified from two processes to one.

4.3 GUI Access

What users are virtually interacting with is the graphical user interface alone and the contents on the GUI is premier. The graphical interface is organized in *section* with single or multiple *pages* form a section. Structure of graphical page is illuminated by Figure 4-3. According to the figure, the entire GUI divides into 6 sections and each block indicates a page with relevant parameters or functionalities listed within. Contents of each section would be described briefly, then an example presents for deeper understanding.

Figure 4-3 Platform GUI Tree



4.3.1 Section 1: Project Setup

The first graphical page shown on upon platform started is the main page that displays basic information of the platform, e.g. supported network structure, available network router features, linked third-party tools and etc. It provides fundamental knowledge about the platform and guides quick configuration of network.

The second page is for NoC project set up. Given express virtual channel and lookahead bypass mechanism as principle features of low-power and low-latency network infrastructure in NoC library, users can select either architecture and assign project name and its work directory. Strictly, both mechanisms have positive affects on network latency and power, in comparison with ordinary router structure. When users are seeking for best NoC through platform, they might well consider NoCs of each library once and proceed comparison to obtain deeper insight. The advantageous NoC architecture with most suitable configuration is the ultimate objective. Assigned NoC name is forced to comprise library type, so its pattern is like user assigned name plus underline and then plus library type. For example, if users name the desired NoC with lookahead bypass mechanism as *Project*, then the final project name is *Project_LowL.*

Two approaches are available for NoCs configuration, by utilizing sequential graphical interface pages to configure each parameter in order or by loading formatted configuration files which are only recognizable by the system.

To configure NoC through GUI, just click *Next* button and then different choice of architecture leads to different configuration page. Once NoC configuration is officially started, it is not allowed to turn back to project setup section unless project completes or platform restarts.

The second approach otherwise requires three system configuration files as input: *project_lib.aconf*, *project_test_lib.tconf* and *project_test_lib.mconf* as indicated on

Figure 1-1. *project* is the name of specific NoC, *test* is the name of specific benchmark and measurement, and *lib* implies applied NoC architecture. *project_lib.aconf* comprises all data about NoC architecture. *project_test_lib.tconf* contains traffic characteristics while *project_test_lib.mconf* includes all necessary measurement settings. Loading *.aconf* is hazardous without checking the compatibility of selected architecture type with that of associate file.

4.3.2 Section 2-a: Low-Power Router Structure

This section consists of multiple pages and several pop out windows displaying all possible settings of low-power NoC architecture. Overall configurable parameters consist of network size, normal buffer allocation, EVCs insertion, data format and sub-module selection. Both different categories being shown on single page, or same category setting being distributed to multiple pages are allowed.

Network size constrains to 10×10 maximum so far. Partly, it is due to the fact that very large network interconnecting more than one hundred processing elements is rare in practice. Another reason is long response time for page generation. Maximum virtual channel number and virtual channel depth being restricted to eight is also for the consideration of real application. Simulation result shows that maximum value of eight is sufficient. More than eight virtual channels being all occupied simultaneously suggests the network is never capable of accommodating current traffics and requires redesign.

Four EVC paths utmost are allowed to set sourcing from the same router. It does not ensure the EVC insertion rule of non-overlapping and system does not check insertion rule violation during configuration. However, if insertion rule is violated, warning alarm issues when simulation is evoked. Deadlock avoidance parameter can be set to maximum nine. Sub-module selection involves multiple options for several secondary modules as well as pipeline stage of link between neighboring routers. Two pop out windows are provided to view configured buffer distribution and EVC path distribution. *network_parameter_sim.sv* and *router_parameter_sim.sv* are generated and saved in the directory *$platform_path/lp_simulation_para/* upon completion of this section.

4.3.3 Section 2-b: Low-Latency Router Structure

The lookahead bypass mechanism as the principle feature of low-latency NoC architecture applies to all network routers within the network. Thus router structure of low-latency architecture is identical for all nodes within the entire network. The GUI pages for low-latency NoC configuration receive parameters of network size, data format and buffer distribution. While the restrictions of network size, virtual channel number and depth at each port are identical with that of low-power architecture. All configurations of NoC architecture provided by section 1 & 2 are listed in following table.

Table 4-5 Configuration of GUI Section 1 & 2

GUI Section 1	**GUI Section 2**
Project Name	Network Size
Project Path	Normal Virtual Channel Number
Architecture Type	Normal Virtual Channel Depth
Architecture Configuration File (*if any*)	Secondary Module Selection
Test bench Configuration File (*if any*)	EVC Path Position (*if any*)
Measurement Configuration File (*if any*)	Express Virtual Channel Number (*if any*)
	Express Virtual Channel Depth (*if any*)
	Deadlock Avoidance Parameter (*if any*)

4.3.4 Section 3: Benchmark & Measurement

Benchmark pages are responsible for generating final test bench. At first, test name and seed number are supposed to be set by users. The name is used for the name of the generated files to distinguish it from others. A special case here is that users are allowed to set any integer number as seed values for randomness along temporal distribution of traffics. The actual value of number does not matter, only to keep consistency for NoCs comparison or simulation result reproduction. Be noted that seed values here can be set multiple, with one separated from another by space. The number of seed determines the number of succeeding simulation trials. It is recommended to be one or three for relatively shorter simulation duration.

As traffic pattern selected, two modes are provided for injection rate configuration, uniform and custom. The first mode applies to random uniform, matrix transpose and locality, set communication probability of all nodes to the identical value (for locality, that is to set value of locality factor array). While full custom can only select custom injection rate mode, with users input corresponding source node, destination node and injection rate manually flow by flow. Simulation interval (unit: clock cycle) is also to be configured here. Upon completion of this section, at most seven types of data files are produced: {*grp_rate_i.dat*}, {*grp_sink_i.dat*}, {*grp_size_i.dat*}, *node_ir.dat*, *ir_distri.dat*, *lookup_table.dat* and *mesh_noc_tb.v* under *./lib_simulation_sv_tb/* folder. Meanwhile, process proceeds to update the contents of *script_noc.do* with the value of simulation interval.

Performance metrics are also selected in current section. Different performance metrics require different files generated for further analysis in terms of low-power NoC. Thus the total number of files under *./simulation_result/* to be opened and written simultaneously during simulation is determined by configurations here. Given 4×4 mesh network, if all three metrics are enabled, number of active files amounts to 256, with large proportion of it written every clock cycle. For larger network to be configured, file number increases linearly. If EVC efficiency enabled, more files are involved. Due to this huge impact on actual duration of simulation, target virtual channels to assess utilization rate should be selected deliberately. Low-latency NoC would not produce so many files during simulation, while actual simulation duration still rests with evaluation choices. Otherwise, low-latency behaves differently as evaluation process integrated into simulation, then two types of file named {*recv_i.dat*} and {*send_i.dat*} respectively are generated during the process and saved under *./ simulation_result/* along with *over_result.dat*.

Other measurement settings include warming up and validity duration. They can be measured by either packet number or clock cycle. If clock cycle chosen, sum of the two parameters is not permitted to exceed total simulation interval. If packet number chosen, users should ensure that sufficient packets left for evaluation. System warning arouses by invalid measurement setting.

4.3.5 Section 4: View Result

Upon completion of all configurations, it arrives at start simulation page and is ready to evoke simulator. Once simulator evoked, graphical interface becomes idle and disabled until simulation completes. The total duration rests with network size, virtual channel distribution, benchmark & measurement settings and etc. After simulation successfully completes, select certain performance metric and click *View* button to check associated report displayed on popped out window. The performance reports are saved under *./simulation_result/* by default, while the copies are also saved under user defined *$project_path*.

Table 4-6 Configuration of GUI Section 3 & 4

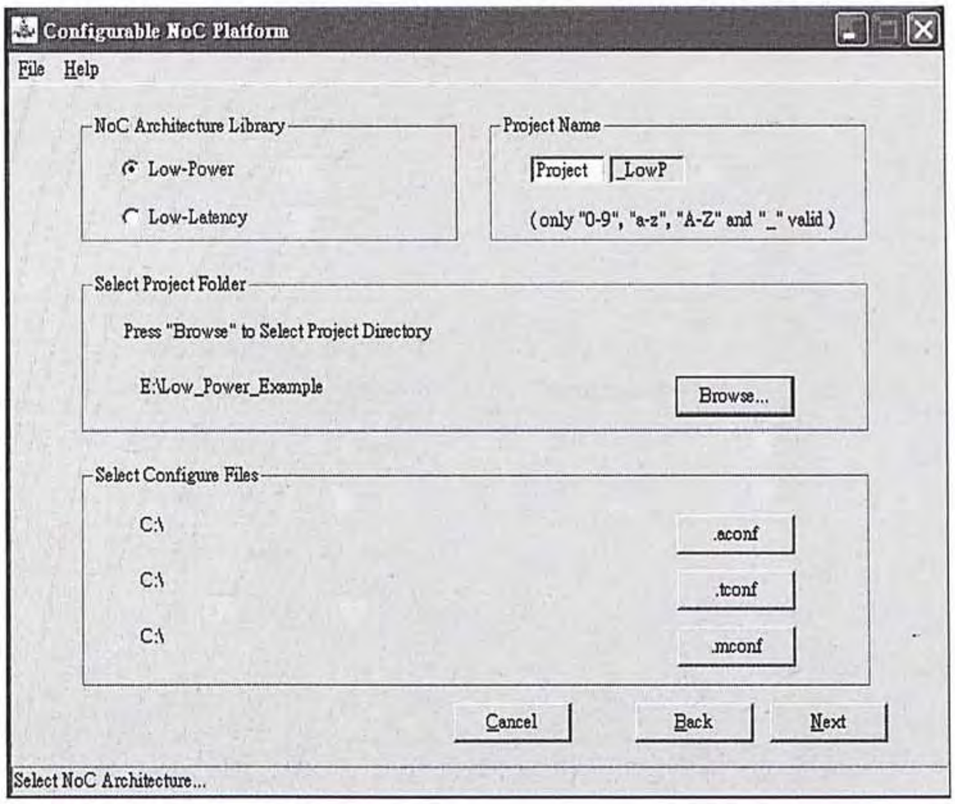
GUI Section 3	GUI Section 4
Test Name	View Latency and Throughput
Seed Number	View Virtual Channel Number Utilization
Traffic Pattern	View Virtual Channel Depth Utilization
Uniform Injection Rate (*if any*)	
Locality Factor (*if any*)	
Individual Flow Setting (*if any*)	
Simulation Interval	
Warming Up Period	
Validity Period	
Performance Metrics	

4.3.6 Low-Power NoC Example

A GUI configuration example of low-power NoC project is presented here for demonstration of NoC configuration flow.

After platform started, project name is assigned as *Project* and click *low-power oriented* button. *_LowP* would supplement to *Project* as the complete project name automatically. Work directory is *E:\Low_Power_Example*. Click *Next* button, then GUI section 2 of low-power oriented NoC page is shown on. The second configuration approach of loading formatted files is skipped here. Let the network be *4×4* mesh topology, normal virtual channel number and virtual channel depth both equal to *four*. Selection of both *Horizontal* and *Vertical* as insertion pattern indicates EVC paths are inserted along both dimensions and offset be *one* suggests only one bypass node exists along each EVC path. The insertion starts from node (0, 0) until exceed network range. EVC number be *two* and depth both to *four* implies the buffer size of express virtual channel. Deadlock avoidance parameter is assigned *three* to *one*. The *Full Custom* button is clicked if non-uniform buffer allocation or EVC path insertion apply.

Figure 4-4 NoC Configuration Example, Section 1


Configurable NoC Platform

File Help

NoC Architecture Library

Low-Power [✓]

Low-Latency

Project Name

Project _LowP

(only "0-9", "a-z", "A-Z" and "_" valid)

Select Project Folder

Press "Browse" to Select Project Directory

E:\Low_Power_Example

Browse...

Select Configure Files

C:\ .aconf

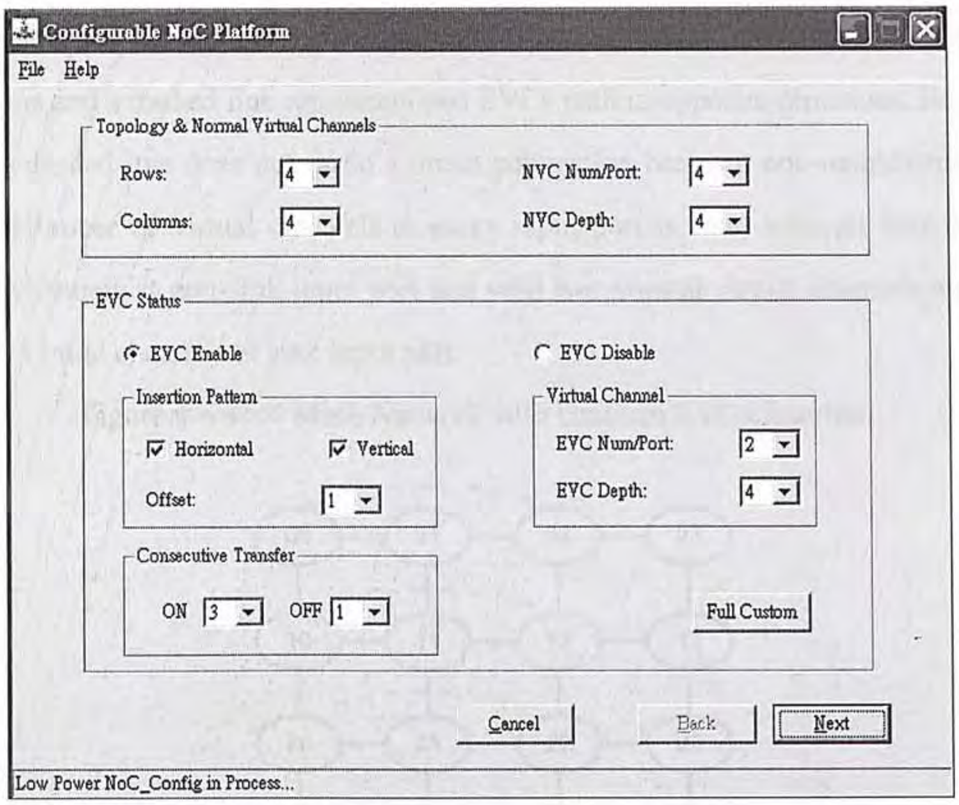
C:\ .tconf

C:\ .mconf

Cancel Back Next

Select NoC Architecture...

Figure 4-5 NoC Configuration Example, Section 2



Configurable NoC Platform

File Help

Topology & Normal Virtual Channels

Rows:	4	NVC Num/Port:	4
Columns:	4	NVC Depth:	4

EVC Status

[✓ EVC Enable] EVC Disable

Insertion Pattern

[✓ Horizontal] [✓ Vertical]

Offset: 1

Virtual Channel

EVC Num/Port: 2

EVC Depth: 4

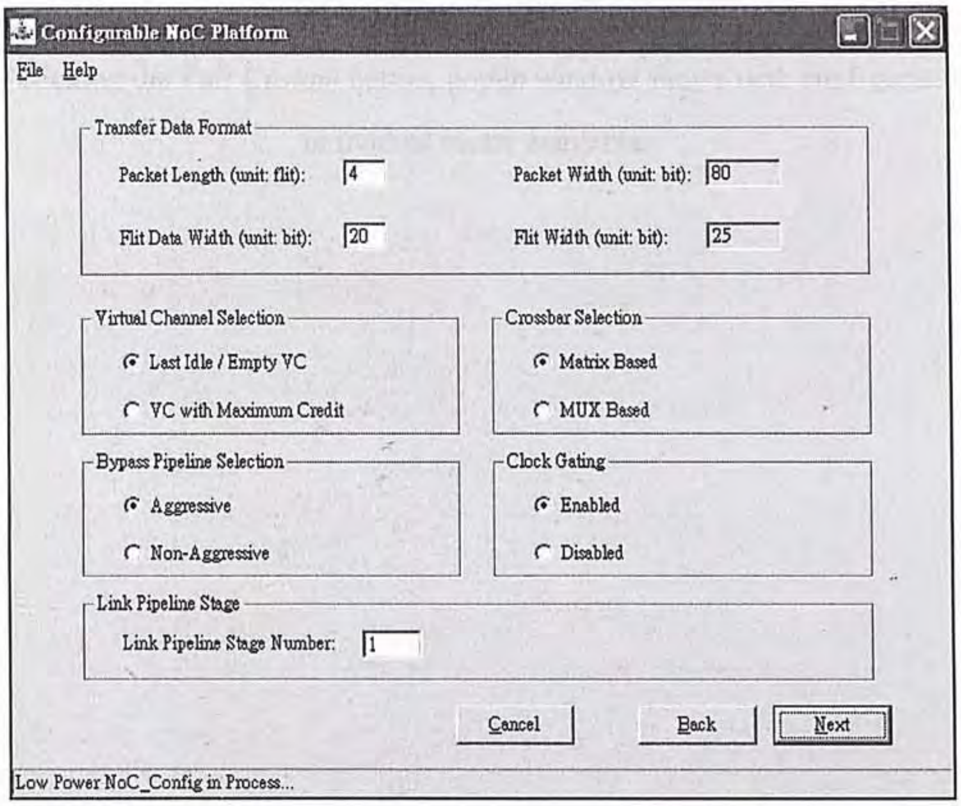
Consecutive Transfer

ON 3 OFF 1

Full Custom

Cancel Back Next

Low Power NoC_Config in Process...



Configurable NoC Platform

File Help

Transfer Data Format

Packet Length (unit: flit):	4	Packet Width (unit: bit):	80
Flit Data Width (unit: bit):	20	Flit Width (unit: bit):	25

Virtual Channel Selection

[✓ Last Idle / Empty VC] VC with Maximum Credit

Crossbar Selection

[✓ Matrix Based] MUX Based

Bypass Pipeline Selection

[✓ Aggressive] Non-Aggressive

Clock Gating

[✓ Enabled] Disabled

Link Pipeline Stage

Link Pipeline Stage Number: 1

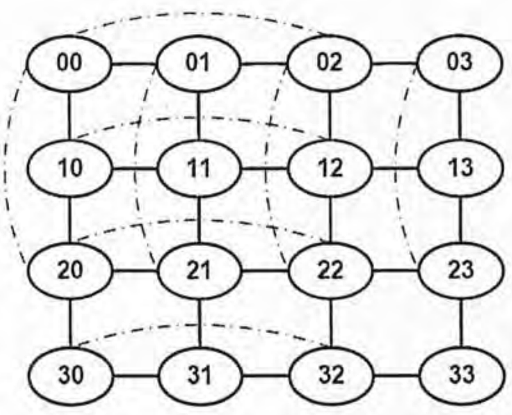
Cancel Back Next

Low Power NoC_Config in Process...

Upon completion of first page, it obtains 4×4 mesh topology network with

EVCs insertion shown as below. A solid line represents two links in opposite directions and a dashed line represents two EVCs path in opposite directions. Be noted that the dashed line does not mean a direct connection between non-neighboring two nodes. Number of virtual channels at every input port is four, with all four normal virtual channels at non-sink input port and with two normal virtual channels and two express virtual channels at sink input port.

Figure 4-6 4×4 Mesh Network with Uniform EVCs Insertion



By clicking the *Full Custom* button, popup window shows with configuration of individual router available.

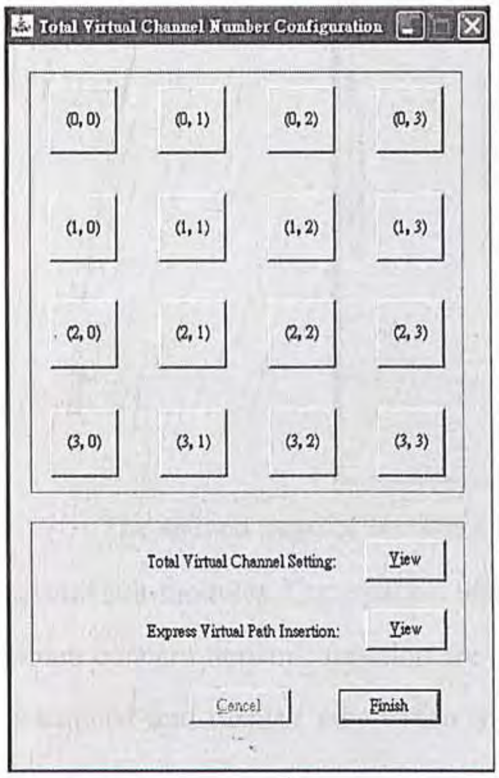
Figure 4-7 shows the popup window by clicking *Full Custom* and *(0, 0)* buttons. Since EVC path is inserted as Figure 4-6 shown, two EVC paths sourcing from node (0, 0) and end at node (0, 2) and (2, 0), respectively. In the meantime, node (0, 0) is the sink node of another two EVC path sourcing from node (0, 2) and (2, 0), two virtual channels allocated at west and south input ports, as shown in

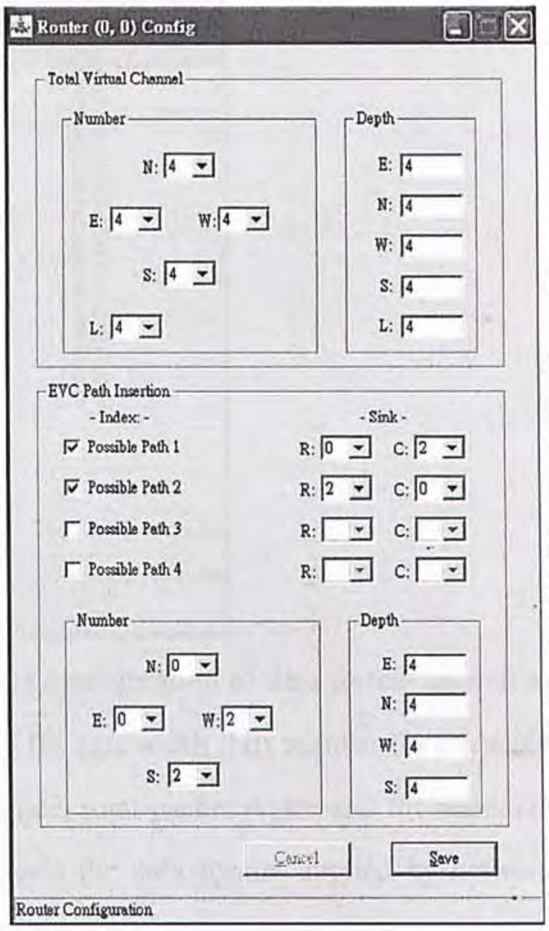
Figure 4-7. By clicking the two *View* button, overview of configured virtual channel and EVC path appears in

Figure 4-8 and

Figure 4-9.

Figure 4-7 Configuration GUI Page of Individual Network Router


Total Virtual Channel Number Configuration
(0, 0)
(0, 1)
(0, 2)
(0, 3)
(1, 0)
(1, 1)
(1, 2)
(1, 3)
(2, 0)
(2, 1)
(2, 2)
(2, 3)
(3, 0)
(3, 1)
(3, 2)
(3, 3)
Total Virtual Channel Setting: View
Express Virtual Path Insertion: View
Cancel
Finish



Total Virtual Channel

Number: N: 4, E: 4, W: 4, S: 4, L: 4

Depth: E: 4, N: 4, W: 4, S: 4, L: 4

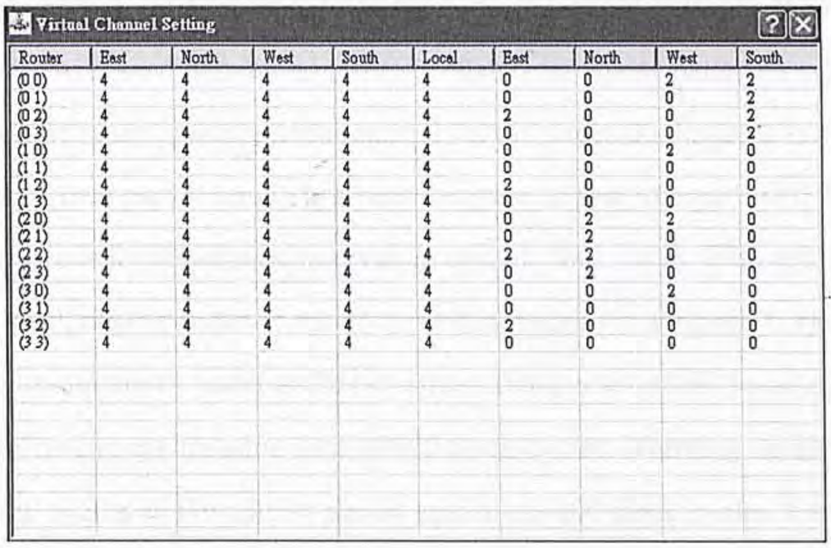
EVC Path Insertion

- Index: -	- Sink - R:	C:
[✓] Possible Path 1	0	2
[✓] Possible Path 2	2	0
[ ] Possible Path 3		
[ ] Possible Path 4		

Number: N: 0, E: 0, W: 2, S: 2

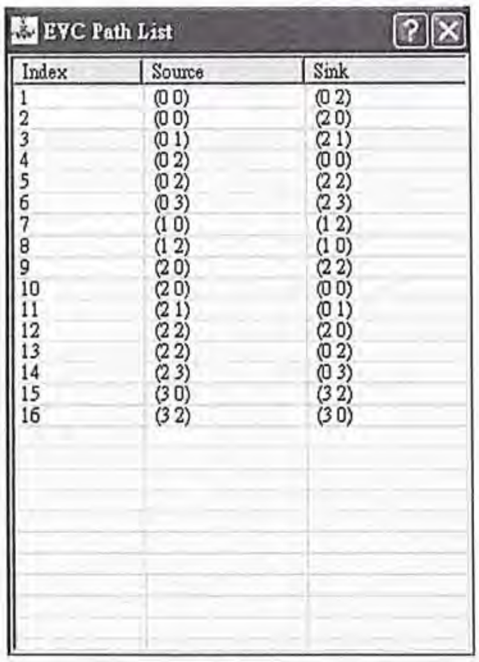
Depth: E: 4, N: 4, W: 4, S: 4

Figure 4-8 Total Virtual Channel Setting View



Router	East	North	West	South	Local	East	North	West	South
(0 0)	4	4	4	4	4	0	0	2	2
(0 1)	4	4	4	4	4	0	0	0	2
(0 2)	4	4	4	4	4	2	0	0	2
(0 3)	4	4	4	4	4	0	0	0	2
(1 0)	4	4	4	4	4	0	0	2	0
(1 1)	4	4	4	4	4	0	0	0	0
(1 2)	4	4	4	4	4	2	0	0	0
(1 3)	4	4	4	4	4	0	0	0	0
(2 0)	4	4	4	4	4	0	2	2	0
(2 1)	4	4	4	4	4	0	2	0	0
(2 2)	4	4	4	4	4	2	2	0	0
(2 3)	4	4	4	4	4	0	2	0	0
(3 0)	4	4	4	4	4	0	0	2	0
(3 1)	4	4	4	4	4	0	0	0	0
(3 2)	4	4	4	4	4	2	0	0	0
(3 3)	4	4	4	4	4	0	0	0	0

Figure 4-9 Express Virtual Path Insertion View


EVC Path List

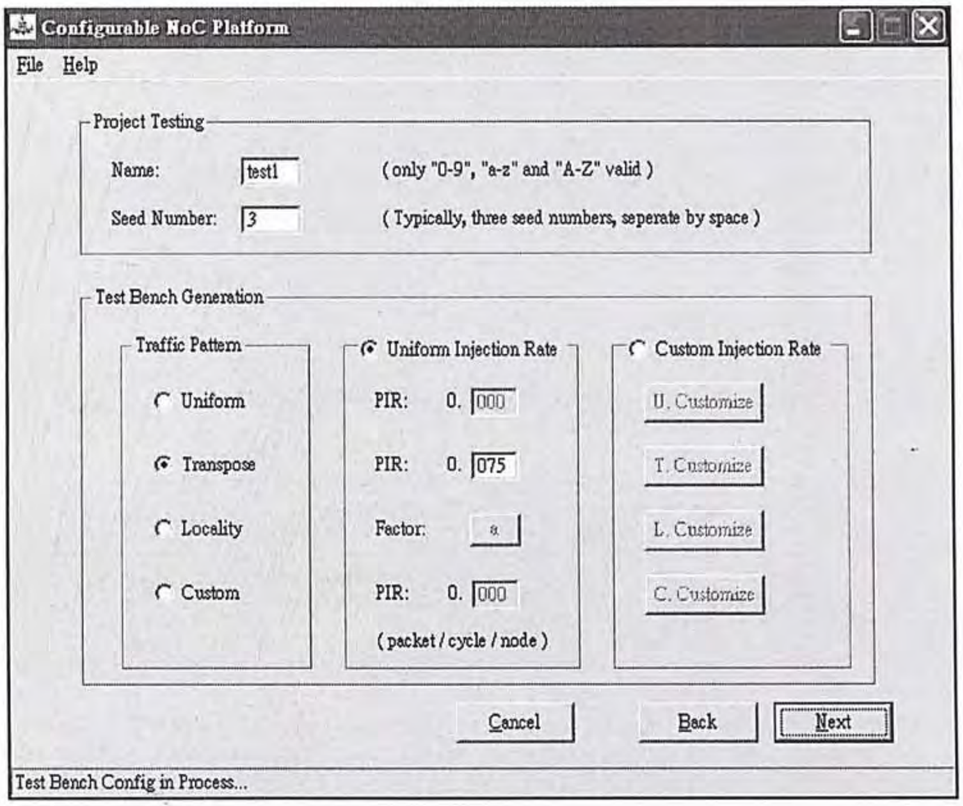
Index	Source	Sink
1	(0 0)	(0 2)
2	(0 0)	(2 0)
3	(0 1)	(2 1)
4	(0 2)	(0 0)
5	(0 2)	(2 2)
6	(0 3)	(2 3)
7	(1 0)	(1 2)
8	(1 2)	(1 0)
9	(2 0)	(2 2)
10	(2 0)	(0 0)
11	(2 1)	(0 1)
12	(2 2)	(2 0)
13	(2 2)	(0 2)
14	(2 3)	(0 3)
15	(3 0)	(3 2)
16	(3 2)	(3 0)

The second page of section 2 involves configuration of data format as well as several sub-modules. Once packet length and flit data width (bits number for payload), which concern network function are determined, total packet width and flit width are calculated and display automatically. It reveals the data format applied by network interface and the width of wires connecting two neighboring routers.

Architecture of the NoC has all been configured so far. Next step is to set the synthetic synthetic benchmark running on defined network. In current example, the benchmark is named *test1* with seed number equals to *3*. Matrix transpose pattern is selected with injection rate of every possible node (apart from the nodes on diagonal line) be 0.075 packet/cycle. Since packet length has been assigned to four, it equals to 0.3 flit/cycle. *Run Time* space is for simulation interval, in unit of clock cycle. With settings shown in Figure 4-10, twelve nodes of the 4×4 mesh network are assigned to send data to diagonal-counterpart nodes at the rate of 0.3 flits/cycle or 0.075 packets/cycle statistically. The entire emulation lasts 10000 clock cycles. Warming up period is set to 500 clock cycles while validity period equals to 9000 clock cycles. Latency & throughput, virtual channel number and virtual channel depth are all enabled for NoC evaluation.

Figure 4-11 informs that all configurations are successfully completed and it is set to provoke QuestaSim to start simulation.

Figure 4-10 NoC Configuration Example, Section 3



Project Testing

Name: testl (only "0-9", "a-z" and "A-Z" valid)

Seed Number: 3 (Typically, three seed numbers, seperate by space)

Test Bench Generation

Traffic Pattern: Uniform [✓ Transpose] Locality Custom

[✓ Uniform Injection Rate]

PIR: 0. 000

PIR: 0. 075

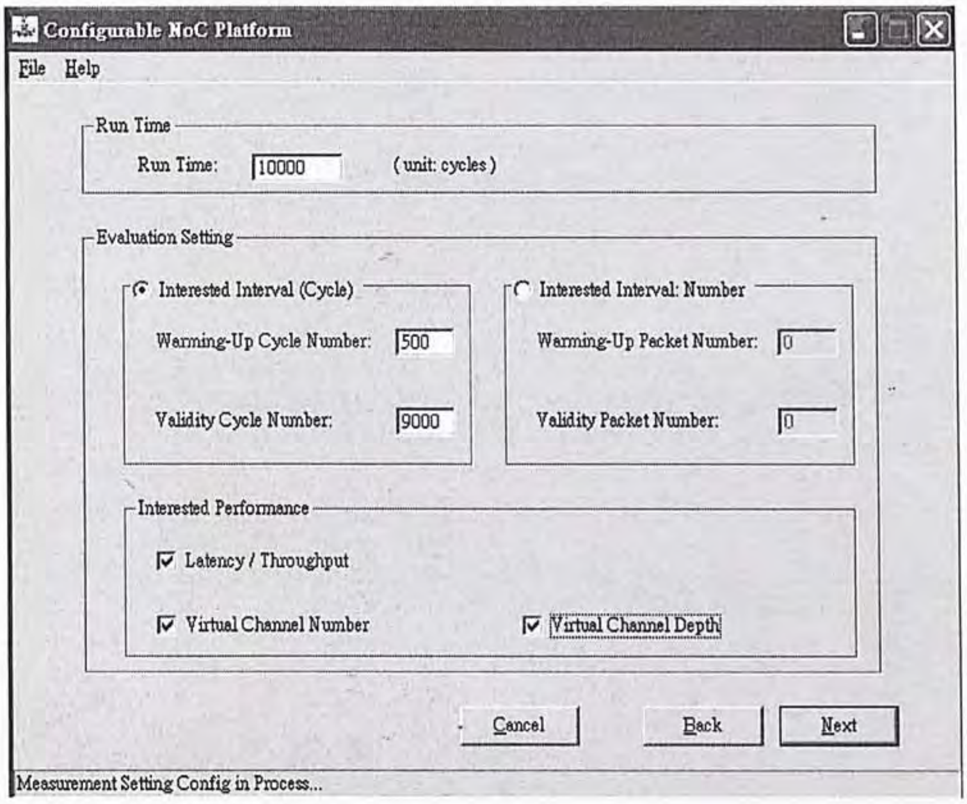
Factor: a

PIR: 0. 000

(packet / cycle / node)

Custom Injection Rate: U. Customize, T. Customize, L. Customize, C. Customize

Test Bench Config in Process...



Run Time

Run Time: 10000 (unit: cycles)

Evaluation Setting

[✓ Interested Interval (Cycle)]

Warming-Up Cycle Number: 500

Validity Cycle Number: 9000

Interested Interval: Number

Warming-Up Packet Number: 0

Validity Packet Number: 0

Interested Performance

[✓ Latency / Throughput]

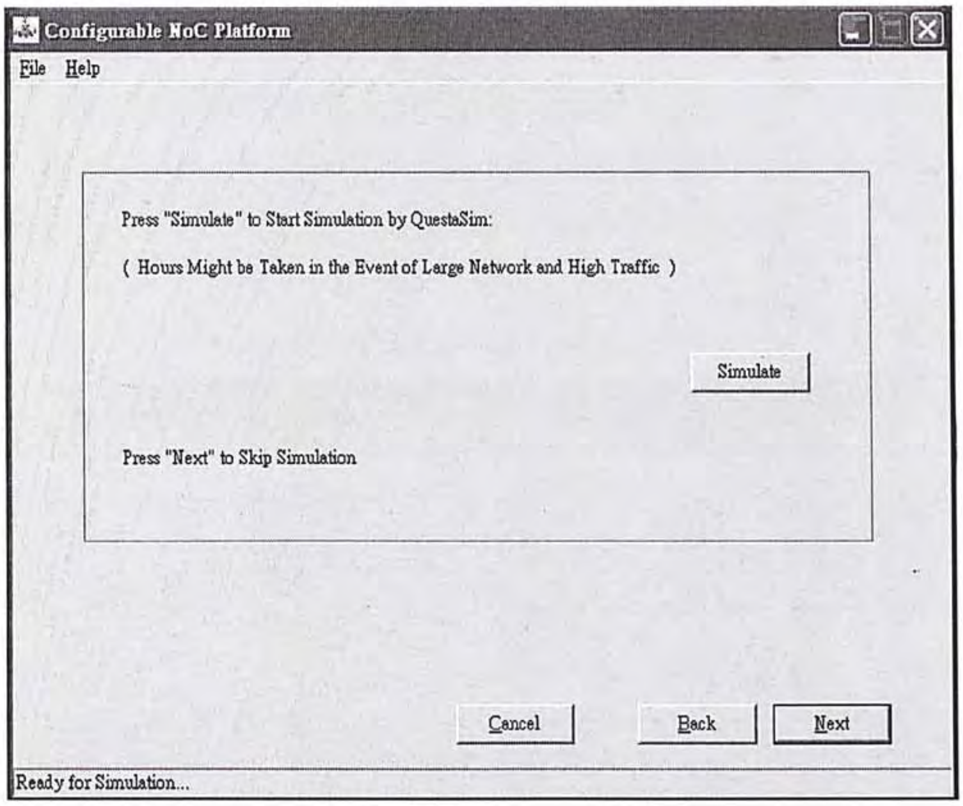
[✓ Virtual Channel Number]

[✓ Virtual Channel Depth]

Measurement Setting Config in Process...

Figure 4-11 [illegible] Configuration Example, Screen 4

Figure 4-11 NoC Configuration Example, Section 4

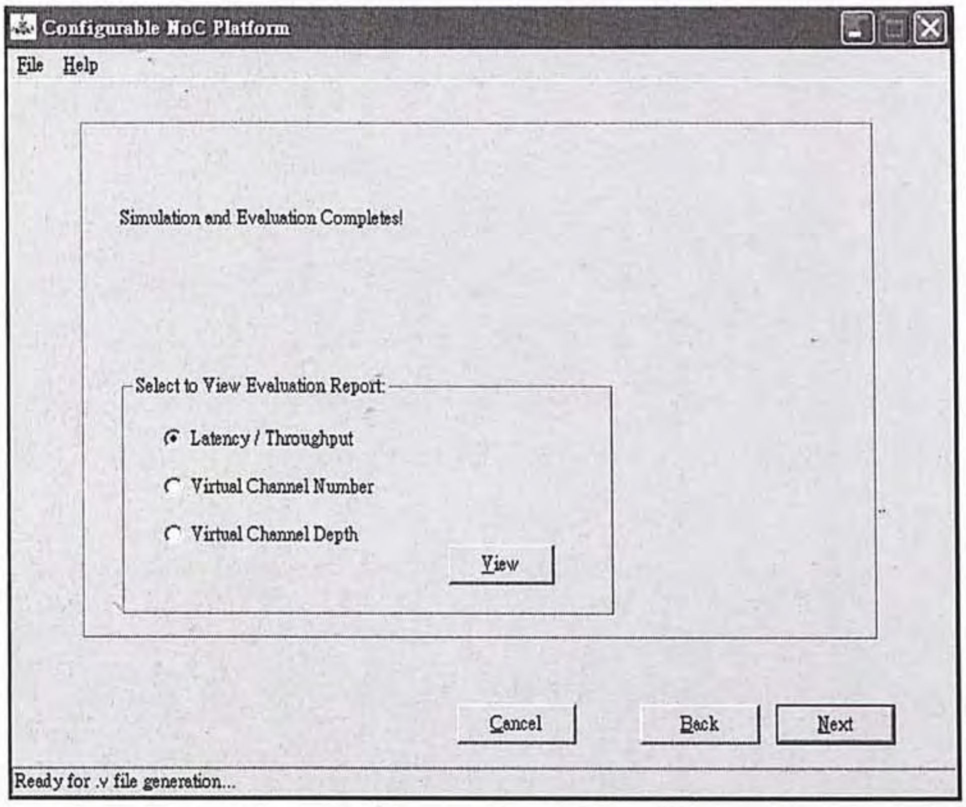


Press "Simulate" to Start Simulation by QuestaSim:

(Hours Might be Taken in the Event of Large Network and High Traffic)

Press "Next" to Skip Simulation

Ready for Simulation...



Simulation and Evaluation Completes!

Select to View Evaluation Report:

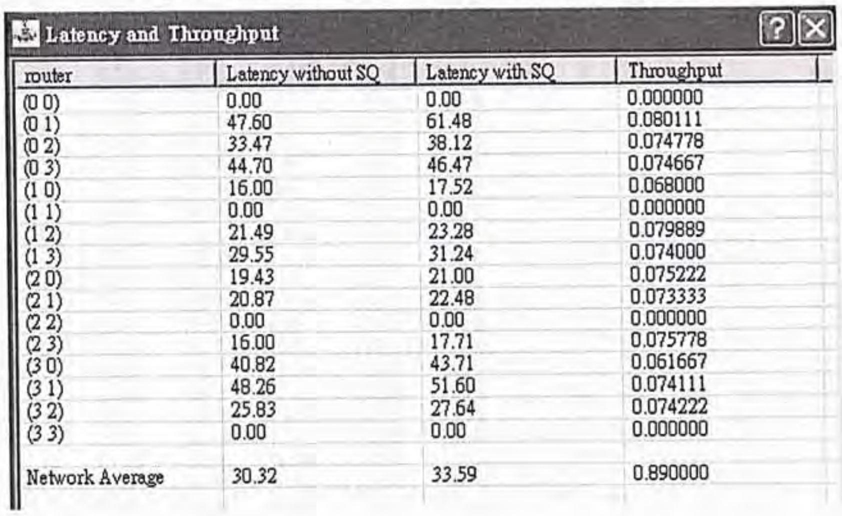
- [✓ Latency / Throughput]
- Virtual Channel Number
- Virtual Channel Depth

Ready for .v file generation...

When simulation ends, platform turns to section 4 page 2 of

Figure 4-11. Users can choose to check interested performance metrics, latency and throughput, virtual channel number or virtual channel depth, as enabled in GUI section 3. Outcomes show up by clicking associated buttons. Latency and throughput reports (Figure 4-12) summarizes latency without source queue, latency with source queue and throughput of packets sourcing from each network node. The averaged value of the entire network displays in the end.

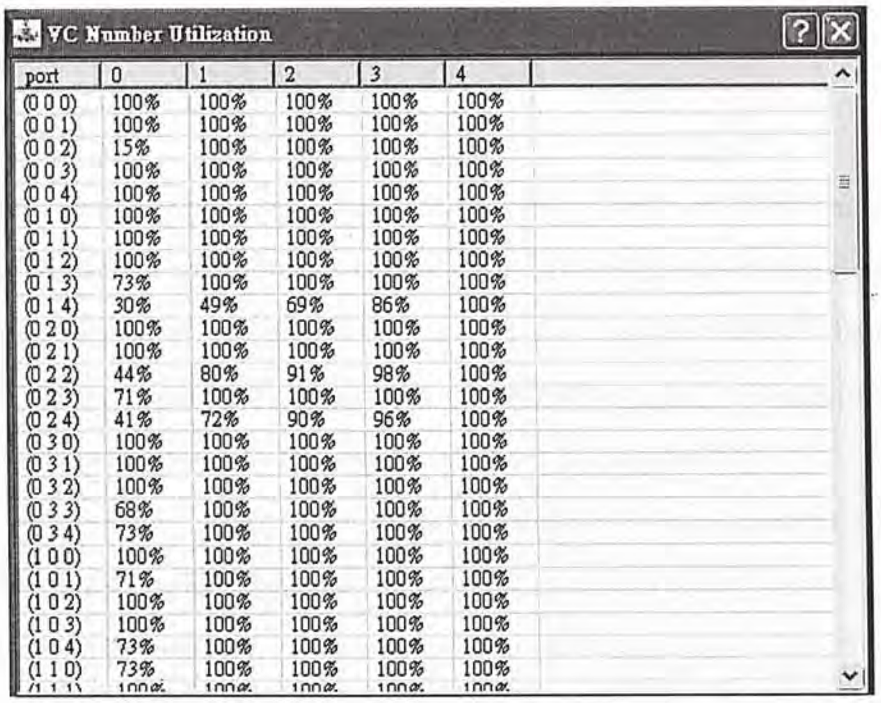
Figure 4-12 Latency and Throughput Report



router	Latency without SQ	Latency with SQ	Throughput
(0 0)	0.00	0.00	0.000000
(0 1)	47.60	61.48	0.080111
(0 2)	33.47	38.12	0.074778
(0 3)	44.70	46.47	0.074667
(1 0)	16.00	17.52	0.068000
(1 1)	0.00	0.00	0.000000
(1 2)	21.49	23.28	0.079889
(1 3)	29.55	31.24	0.074000
(2 0)	19.43	21.00	0.075222
(2 1)	20.87	22.48	0.073333
(2 2)	0.00	0.00	0.000000
(2 3)	16.00	17.71	0.075778
(3 0)	40.82	43.71	0.061667
(3 1)	48.26	51.60	0.074111
(3 2)	25.83	27.64	0.074222
(3 3)	0.00	0.00	0.000000
Network Average	30.32	33.59	0.890000

The column label of 錯誤! 書籤的自我參照不正確。 implies number of virtual channels, while value of cell below indicates the proportion that certain number of virtual channels are active simultaneously. Figure 4-14 otherwise lists the maximum number of units being occupied in terms of each virtual channel

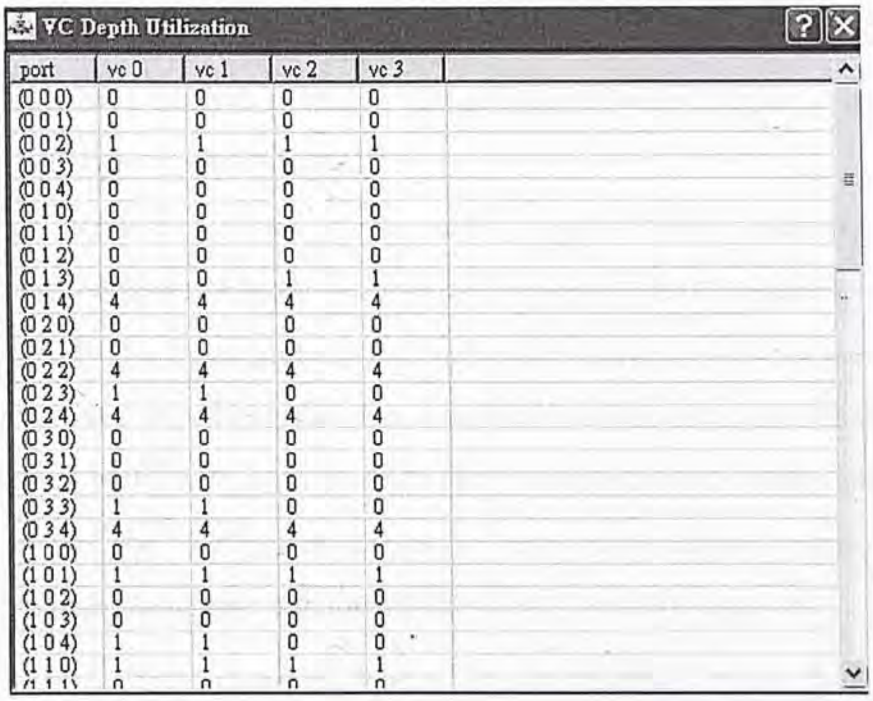
Figure 4-13 Virtual Channel Number Utilization Report



VC Number Utilization

port	0	1	2	3	4
(0 0 0)	100%	100%	100%	100%	100%
(0 0 1)	100%	100%	100%	100%	100%
(0 0 2)	15%	100%	100%	100%	100%
(0 0 3)	100%	100%	100%	100%	100%
(0 0 4)	100%	100%	100%	100%	100%
(0 1 0)	100%	100%	100%	100%	100%
(0 1 1)	100%	100%	100%	100%	100%
(0 1 2)	100%	100%	100%	100%	100%
(0 1 3)	73%	100%	100%	100%	100%
(0 1 4)	30%	49%	69%	86%	100%
(0 2 0)	100%	100%	100%	100%	100%
(0 2 1)	100%	100%	100%	100%	100%
(0 2 2)	44%	80%	91%	98%	100%
(0 2 3)	71%	100%	100%	100%	100%
(0 2 4)	41%	72%	90%	96%	100%
(0 3 0)	100%	100%	100%	100%	100%
(0 3 1)	100%	100%	100%	100%	100%
(0 3 2)	100%	100%	100%	100%	100%
(0 3 3)	68%	100%	100%	100%	100%
(0 3 4)	73%	100%	100%	100%	100%
(1 0 0)	100%	100%	100%	100%	100%
(1 0 1)	71%	100%	100%	100%	100%
(1 0 2)	100%	100%	100%	100%	100%
(1 0 3)	100%	100%	100%	100%	100%
(1 0 4)	73%	100%	100%	100%	100%
(1 1 0)	73%	100%	100%	100%	100%
(1 1 1)	100%	100%	100%	100%	100%

Figure 4-14 Virtual Channel Depth Utilization Report



VC Depth Utilization

port	vc 0	vc 1	vc 2	vc 3
(0 0 0)	0	0	0	0
(0 0 1)	0	0	0	0
(0 0 2)	1	1	1	1
(0 0 3)	0	0	0	0
(0 0 4)	0	0	0	0
(0 1 0)	0	0	0	0
(0 1 1)	0	0	0	0
(0 1 2)	0	0	0	0
(0 1 3)	0	0	1	1
(0 1 4)	4	4	4	4
(0 2 0)	0	0	0	0
(0 2 1)	0	0	0	0
(0 2 2)	4	4	4	4
(0 2 3)	1	1	0	0
(0 2 4)	4	4	4	4
(0 3 0)	0	0	0	0
(0 3 1)	0	0	0	0
(0 3 2)	0	0	0	0
(0 3 3)	1	1	0	0
(0 3 4)	4	4	4	4
(1 0 0)	0	0	0	0
(1 0 1)	1	1	1	1
(1 0 2)	0	0	0	0
(1 0 3)	0	0	0	0
(1 0 4)	1	1	0	0
(1 1 0)	1	1	1	1
(1 1 1)	0	0	0	0

Chapter 5 Optimization and Comparison

This chapter elaborates the flow of NoC buffer optimization and flow of NoCs comparison. Exportation of implementation code of low-power NoC is explained in the end.

5.1 Optimization Technique

Optimization technique of proposed platform focuses on high communication probability cases. NoC performance is steady when overall communication probability stays at a very low level, and then less likelihood of contention results in best network capability. Otherwise, if traffic approaches to saturation point, network performance becomes instable and some NoCs outperform others instantaneously.

As a re-usable infrastructure of communication system, network always has uniform distributed buffers. Suppose virtual channel adopt as flow control mechanism, it suggests identical number of virtual channels at each router input port which ensures various traffic patterns can be accommodated by identical network configuration. However, when mapping onto hardware for running real applications, buffer number at each input port must be customized individually. As buffer occupies significant portion of chip area, remove excessive virtual channels with acceptable latency and throughput penalty is worthwhile [23].

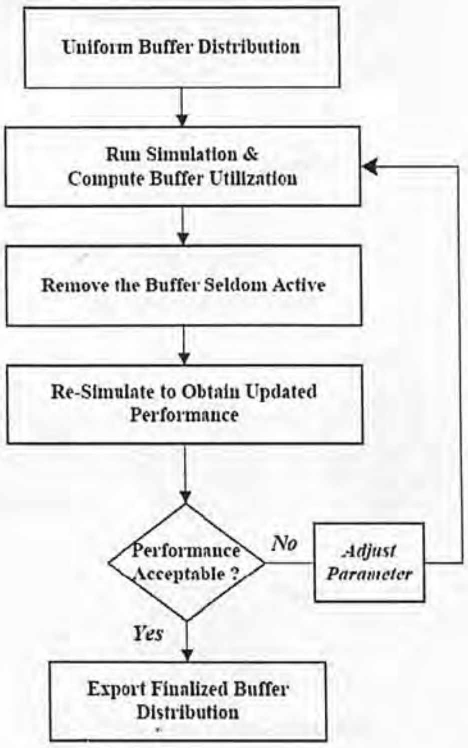
Some literatures [10][34] address on the impact of virtual channel number or virtual channel depth on network performance. However, all of them choose varying the buffer size at every port globally to observe the change of network performance, which indicates always uniform distributed buffer. However, in the case of unbalanced traffics, tailored buffer size at every single port is more effectual. Buffer optimization technique of proposed platform addresses the best virtual channel number at every single port in order to achieve minimum buffer size of the entire network.

5.1.1 Optimization Phase 1: Inactive Buffer Removal

Monitor buffer utilization during simulation interval provides straightforward implication for optimization: the forever idle buffers are eligible to be removed in no doubt, and the buffers that are left idle most of the time and removed with insignificant overhead could be reduced as well. The only challenge is to define the threshold of acceptable penalty.

Figure 5-1 illuminates the flow of buffer optimization. After buffer architecture initialized, simulation runs once with buffer utilization being computed. Then system removes the virtual channels that are never being utilized and the ones are seldom active (e.g. 1% chance of active only), and re-simulate to obtain updated performance. If performance of new NoC configuration is acceptable, virtual channel optimization is confirmed and system read the updated utilization report, further optimize virtual channel distribution and start new simulation again. It continues until performance unacceptable or all virtual channels are all frequently in use.

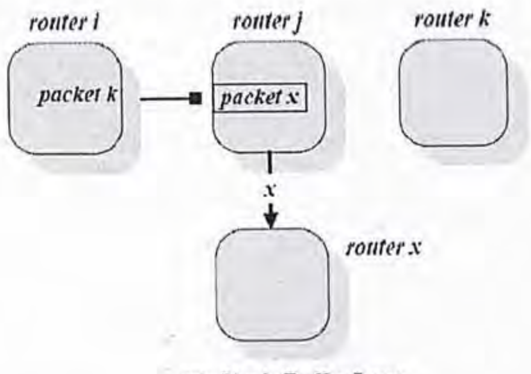
Figure 5-1 Plain Buffer Optimization Flow


Uniform Buffer Distribution
Run Simulation & Compute Buffer Utilization
Remove the Buffer Seldom Active
Re-Simulate to Obtain Updated Performance
Performance Acceptable ?
No
Adjust Parameter
Yes
Export Finalized Buffer Distribution

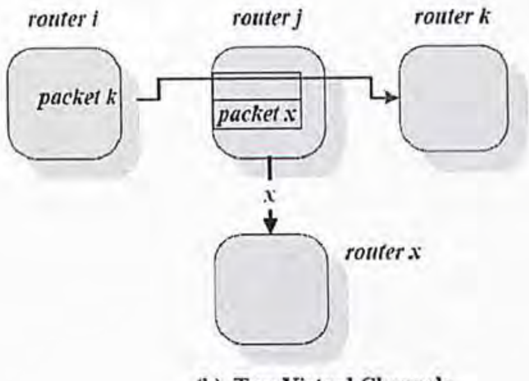
5.1.2 Optimization Phase 2: Infighting Analysis

Above optimization technique that terminates once all virtual channels being frequently in use ignores a critical feature of virtual channel as a flow control mechanism. Figure 5-2[a] depicts a network with single buffer lane allocated at each router input port. *packet x* (with *node x* as its destination) is stuck at *East* input port of *router j* because for some reason the link between *router j* and *router x* is unavailable. In the meantime, *node i* sustains sending packets to *node k* (*packet k*). Then *packet k* will be blocked at *router i* even though *router j* and *k* are both available. Figure 5-2[b] shows the same traffic characteristic with two virtual channels allocated to network router architecture. On this occasion, the *packet x* occupies *virtual channel 2* at *East* input port of *router j*, with *virtual channel 1* available. Then *packet x* being blocked has no impact on *packet k* as the connection is realized through the *virtual channel 1* at *router j*.

Figure 5-2 Virtual Channel Origination



(a). Single Buffer Lane



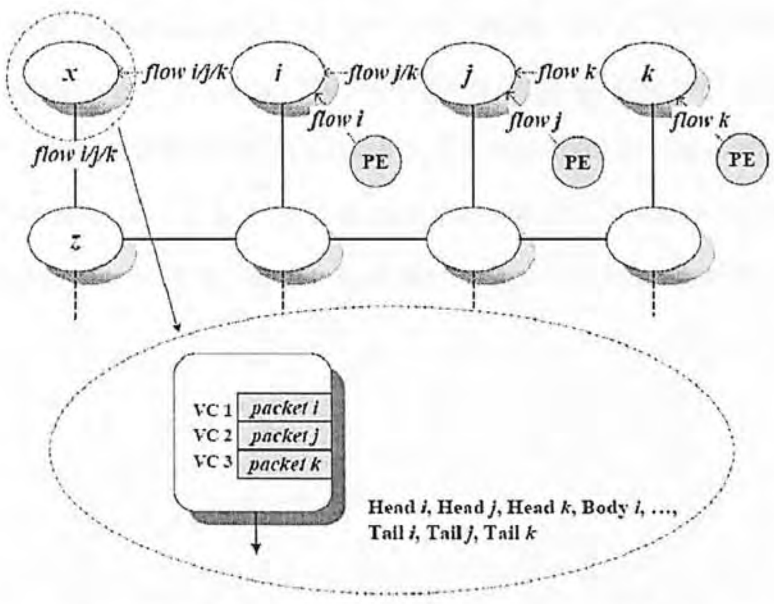
(b). Two Virtual Channels

However, all virtual channels being active observed from virtual channel utilization does not imply highest network efficiency [19].

Figure 5-3 presents a case of three flows sourcing from various nodes but reaching identical destination *node z*. The flows are competing for *West* input port as well as *South* output port of *router x*. Assuming the total workload of these three flows is relative high, it occurs that packets accumulating at virtual channels of *West* input port of *router x*. Suppose three packets currently occupying three virtual channels respectively, due to arbitration fairness of network router design, flits from various virtual channels will be granted to request and traverse switch crossbar alternatively. In consequence, it is observed that tail flits of three packets eject out from *South* output port of *router x* consecutively, which lengthens average latency of all packets. For example, if single buffer lane applies, sequence of flits leaving is *i*, *i*, *i*... *j*, *j*, *j*...*k*, *k*, *k*..., while if three virtual channels are allocated there, the sequence turns to *i*, *j*, *k*, *i*, *j*, *k*, *i*, *j*, *k*,...This is called *infighting* in proposed platform. Suppose packet comprises of four flits, then average latency of the three packets is eight for single buffer lane and eleven for multiple virtual channels. Infighting occurs at *router i* and *j* as well and it could impact overall network latency significantly.

Since allocation of multiple virtual channels at single input port could cause inefficiency in comparison with single buffer lane, infighting analysis examines spatial distribution of traffic and identifies the port where infighting is likely to occur. This is used as supplement for virtual channel optimization.

Figure 5-3 Three-Flow Infighting Example



Suppose uniformly distributed buffer is implemented on network 4×4 and random uniform traffic pattern applies to it. Intuitively, all links are assigned identical amount of traffic and should treat equally. Surprise, it is not the case.

Table 5-1 summarizes the flow distribution of uniform traffic for each router input port, assuming the total communication of any two nodes are 1. The first column is coordinate of network router, other column indicates single input port. The content of any cell is five numbers embraced in a bracket. For example, for the *East* input port of router (0 1), the value is (0 0 8 3 1). It means the total traffic moves from *East* input port to *West* output port is 8, to *South* output port is 3 and to local port is 1. It observes from

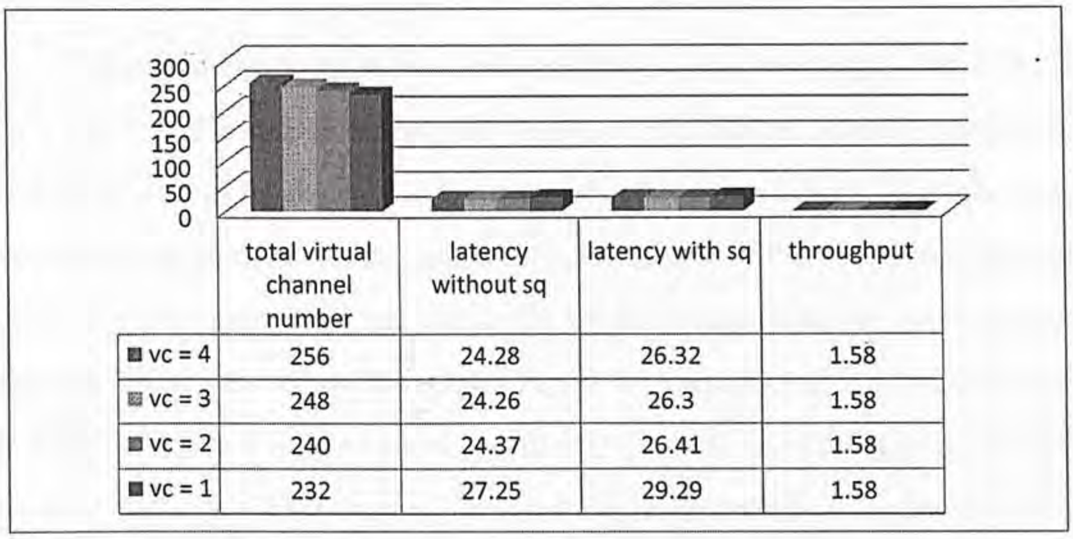
Table 5-1 that all 12 units of packet reaching *South* input port of (0 0), (0 1), (0 2) and (0 3) go to *Local* output port, and all 12 units of packet reaching *North* input port of (3 0), (3 1), (3 2) and (3 3) go to *Local* output port as well. Therefore, infighting is likely to occur at these ports.

Table 5-1 Flow Analysis on Uniform Traffic

Router	East	North	West	South	Local
(0 0)	(0 0 0 0 0)	(0 0 0 0 0)	(0 0 0 9 3)	(0 0 0 0 12)	(0 0 12 3 0)
(0 1)	(0 0 8 3 1)	(0 0 0 0 0)	(8 0 0 6 2)	(0 0 0 0 12)	(4 0 8 3 0)
(0 2)	(0 0 8 6 2)	(0 0 0 0 0)	(8 0 0 3 1)	(0 0 0 0 12)	(8 0 4 3 0)
(0 3)	(0 0 0 9 3)	(0 0 0 0 0)	(0 0 0 0 0)	(0 0 0 0 12)	(12 0 0 3 0)
...	...	...	...	...	...
(3 0)	(0 0 0 0 0)	(0 0 0 0 12)	(0 9 0 0 3)	(0 0 0 0 0)	(0 3 12 0 0)
(3 1)	(0 3 8 0 1)	(0 0 0 0 12)	(8 6 0 0 2)	(0 0 0 0 0)	(4 3 8 0 0)
(3 2)	(0 6 8 0 2)	(0 0 0 0 12)	(8 3 0 0 1)	(0 0 0 0 0)	(8 3 4 0 0)
(3 3)	(0 9 0 0 3)	(0 0 0 0 12)	(0 0 0 0 0)	(0 0 0 0 0)	(12 3 0 0 0)

Figure 5-4 compares four NoCs with tiny difference in buffer allocation while running identical uniform traffic pattern. The number of virtual channels at above identified input ports varies from four to one with average network latency and throughput obtained from simulation, respectively. The figure suggests that reduction of virtual channels at these ports affects little to network performance.

Figure 5-4 Infighting Affects on Uniform Traffic



	total virtual channel number	latency without sq	latency with sq	throughput
■ vc = 4	256	24.28	26.32	1.58
▨ vc = 3	248	24.26	26.3	1.58
■ vc = 2	240	24.37	26.41	1.58
■ vc = 1	232	27.25	29.29	1.58

5.1.3 Over-Optimization

According to Figure 5-4, if only one virtual channel is allocated to infighting input port, average packet latency is slightly longer than that of multiple virtual channel being allocated. The reason is that in the case of one virtual channel, the non-empty and non-busy virtual channel is not available every single cycle.

Figure 5-5 Wave Form of Continuous Requests to Single Virtual Channel

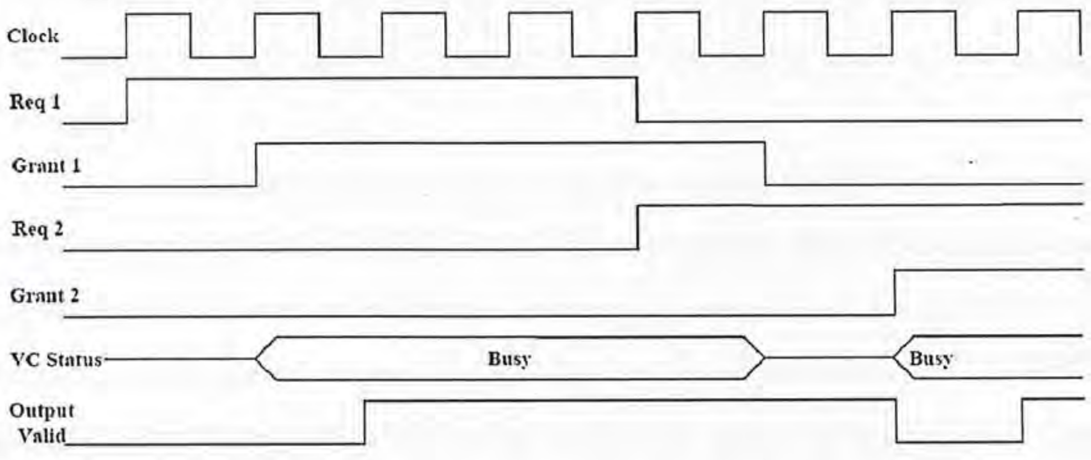


Figure 5-5 depicts the wave of two requests in a row to a single virtual channel. Req 1 and Req 2 could be the request signals of two packets stored in one virtual channel, or two virtual channels at the same or different input ports. Typical network router design requires one cycle latency from the release of the virtual channel to the availability of it again. Thus one void cycle appears at the requested output port that influences latency as well as throughput. Figure 4-13 reports that at some input ports, e.g. *West* input port of node (0, 0), or port (0 0 2), at most one virtual channel is active. However, this one virtual channel does not imply one identical virtual channel. It could be four virtual channels being requested and granted in rotation thus the circumstance in Figure 5-5 is evitable. Briefly, reduce virtual channel number to one at infighting input port could bring void requesting cycle to impact network performance negatively. It is demonstrated by following simulation result.

5.1.4 Optimization Example

Table 5-2 lists the NoC configuration as well as the traffic and measurement settings of optimization example. Low-power NoC architecture is selected without EVC insertion. All nodes are assigned identical communication probability. Warming up period is 500 clock cycles with validity period of 9000 cycles. Thus the measurement period is from 501 to 9500 clock cycle. Both transpose and uniform traffic patterns are taken into account with individual node injection rate equals to 0.075 packet per cycle and 0.050 packet per cycle. Thus there are four cases in total for evaluation.

Each evaluation consists of four simulation trials. The first one is by letting virtual channel distribution initialized with four at each input port, excluding the ones not involved in communication, such as *East* and *North* input port of node (0, 0). Thus the total virtual channel number is 256. Upon the completion of it with *.vcnrpt* file generated, the first optimization trial proceeds with parameter set to 100%. Then virtual channel distribution is updated and the second simulation starts. The second optimization proceeds with parameter set to 99%, and then start the third simulation. Infighting analysis proceeds after that followed by the fourth simulation.

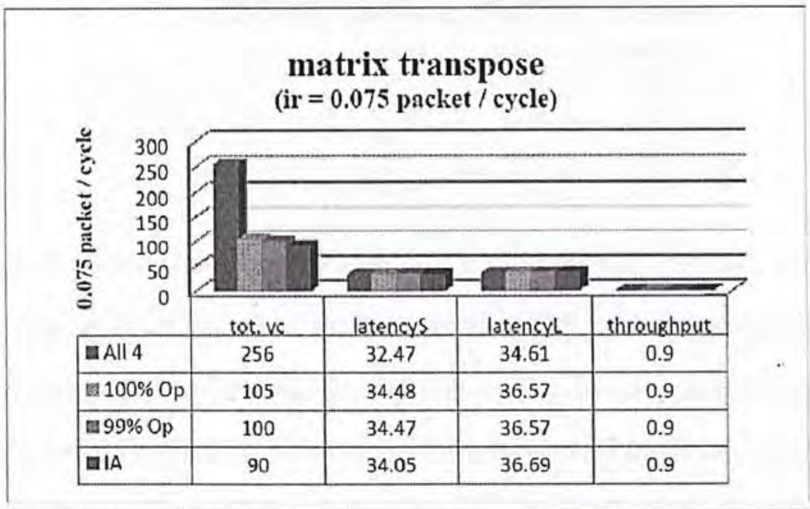
Table 5-2 NoC Architecture and Testing Configuration

Network Size	4x4
Total Virtual Channel Number	4
Normal Virtual Channel Depth	4
Packet Length	4
Flit Data Width	20
Virtual Channel Selection	Last Idle / Empty
Crossbar	Matrix Based
Bypass Pipeline	Aggressive
Clock Gating	Enabled
Link Pipeline Stage	1
Traffic Pattern	Uniform & Matrix Transpose
Injection Rate	0.075packet / cycle & 0.050packet / cycle
Simulation Interval	10000 cycles
Measurement Interval	501-9500 cycles

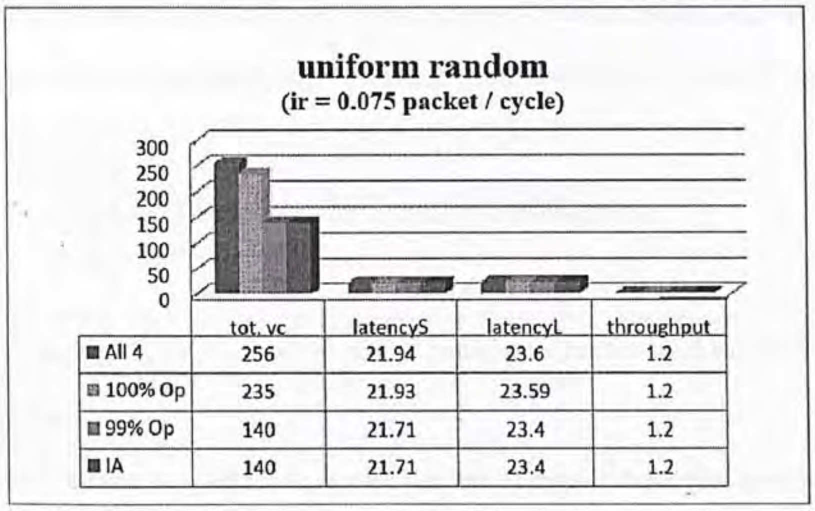
Each sub figure of Figure 5-6 summarizes the entire simulation result. Each column indicates total network virtual channel number, average latency without source queue, average latency with source queue and total throughput. Each row indicates four simulation trials, initialization, optmization with parameter 100%, optimization with parameter 99% and infighting analysis results.

The outcome shown in Figure 5-6 suggests that the optimization technique distinguishes redundant virtual channels effectively. Total virtual channel number decreases significantly with tiny performance penalty. Total active virtual channel number of uniform random is more than that of matrix transpose because of two reasons. First, total traffic of uniform random is more than traffic of matrix transpose; second, uniform random traffic is more balanced so that more ports are involved in data transfer. However, matrix transpose occupies more virtual channels in comparison with uniform random along certain paths, such as the one from (0, 3) to (3, 0) and from (2, 0) to (0, 2). That is why matrix transpose benefits more from infighting analysis.

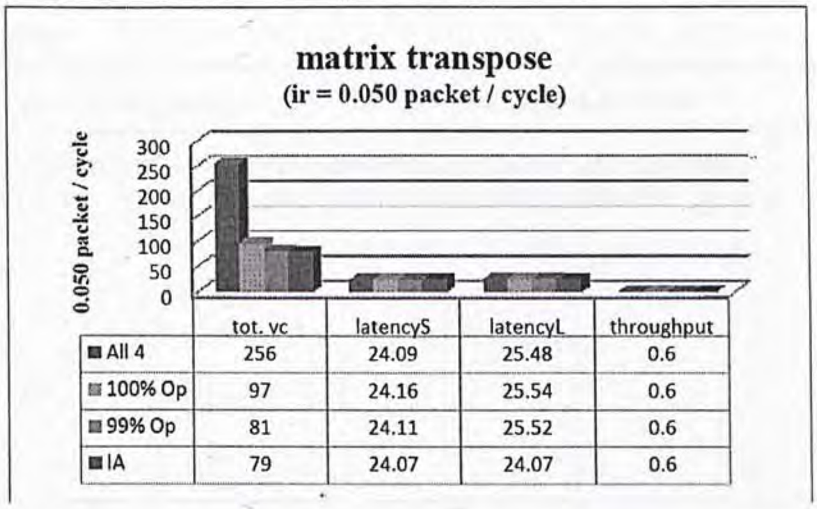
Figure 5-6 Optimization Example on matrix transpose and uniform



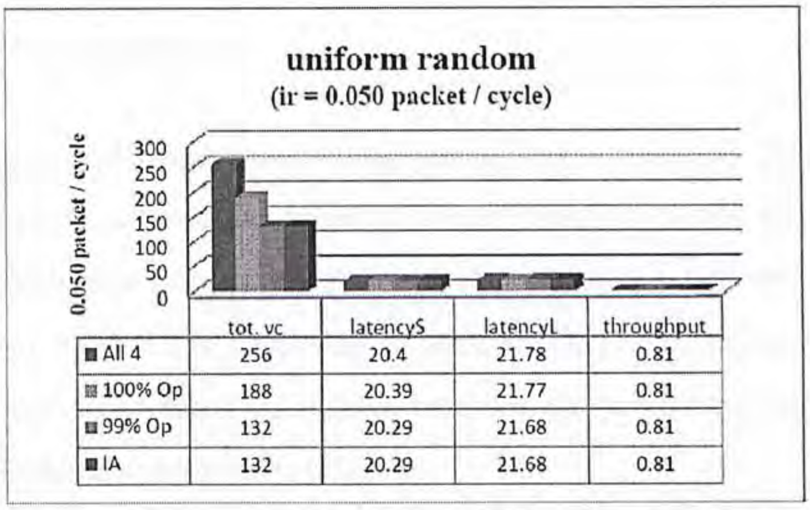
	tot. vc	latencyS	latencyL	throughput
■ All 4	256	32.47	34.61	0.9
▨ 100% Op	105	34.48	36.57	0.9
▨ 99% Op	100	34.47	36.57	0.9
■ IA	90	34.05	36.69	0.9



	tot. vc	latencyS	latencyL	throughput
■ All 4	256	21.94	23.6	1.2
▨ 100% Op	235	21.93	23.59	1.2
▨ 99% Op	140	21.71	23.4	1.2
■ IA	140	21.71	23.4	1.2



	tot. vc	latencyS	latencyL	throughput
■ All 4	256	24.09	25.48	0.6
▨ 100% Op	97	24.16	25.54	0.6
▨ 99% Op	81	24.11	25.52	0.6
■ IA	79	24.07	24.07	0.6



	tot. vc	latencyS	latencyL	throughput
■ All 4	256	20.4	21.78	0.81
▨ 100% Op	188	20.39	21.77	0.81
▨ 99% Op	132	20.29	21.68	0.81
■ IA	132	20.29	21.68	0.81

Infighting analysis shows noticeable impact on virtual channel number in terms of matrix transpose at injection rate equals to 0.075 packet per cycle. That is because of more centralized and congestion-prone traffic. While, infighting analysis shows neutral or positive impact on network performance to all the four cases.

Processes involved in platform optimization are listed in following table. Each optimize operation calls multiple times of all these processes while it shows on GUI as one click job.

Table 5-3 Processes involved of Optimization

Process ID	***Operation***
501	Read *project_test_lib.vcnrpt* to remove redundant virtual channel which are seldom active
502	Read *project_test_lib.vcdrpt* to minimize virtual channel depth which are seldom active
503	Update *router_parameter_sim.v* by optimized buffer distribution
504	Infighting analysis of traffic distribution and update buffer distribution

5.2 NoCs Comparison

Challenge of NoCs comparison cross systems lays on regularly undisclosed configuration details. The distinctive feature of proposed platform is that comparison no longer sticks with plain numeric values from evaluation metrics. Reported average network latency of two NoCs is not conclusive unless validity of comparison itself is confirmed by system in advance [32][33]. The proposed platform presents three rules to ensure the proceeding comparison is valid.

Rule 1: Two types of comparison are defined and distinguished from each other, specific comparison and general comparison. Specific comparison requires simulation once, while averaged outcome of multiple simulations is used for general comparison.

Rule 2: NoCs involved in either type of comparison must be applied with identical traffic pattern. The identity implies identical spatial distribution, temporal distribution and communication probability. Since pseudo-random mechanism applies to temporal distribution and the randomness is controlled by seed number in traffic generator, the NoCs in comparison must apply identical traffic generator along with exact the same seed number.

Rule 3: Measurement settings must be identical for projects to compare. It suggests identical simulation interval, warming up period, and validity period. All of it ensures evaluation results are obtained under exact the same traffics.

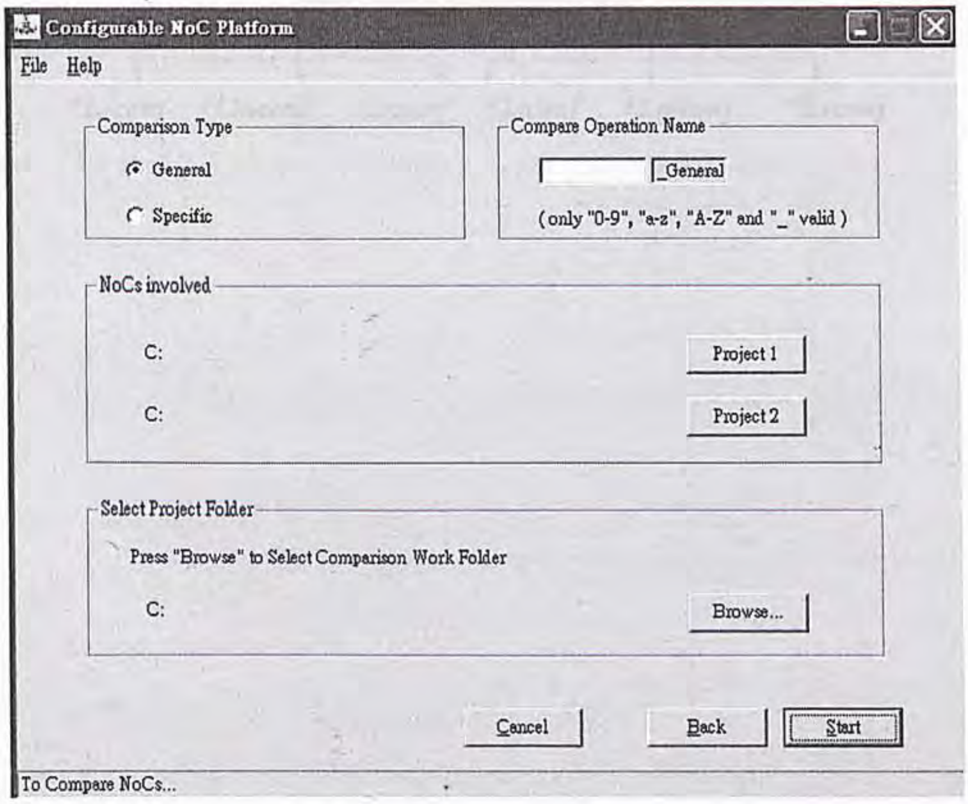
The feature of synthetic benchmark lays on its reflection of general case. In other words, when synthetic benchmark applied benign traffic temporal distribution is expected. However, it is not always the case. It is likely to occur that different choice of seed number for identical traffic generator brings congestion to one case yet nothing to another. Thus performance metrics to participate comparison must be the averaged result of multiple simulations.

Table 5-4 Processes involved of Comparison

Process ID	***Operation***
601	Read the two *.tconf* files to ensure validity
602	Read the two *.mconf* files to ensure validity
603	Read the two *.rconf* files to find report files, compare them to generate comparison outcome

Operation of NoCs comparison through GUI is simple. Users are requested to select comparison type, assign name for current operation, choose a folder to save relevant files and then load associated files of two projects: *project1_test1_lib1.ini* and *project2_test2_lib2.ini*. By clicking *Start* button, *.cmp* file is created under assigned work folder and then processes are evoked to proceed NoCs comparison. The two files *project1_test1_lib1.ini* and *project2_test2_lib2.ini* contains nothing but paths of their associated*.tconf*, *.mconf* and *.rconf* files with their contents states in Table 5-6, 5-7 and 5-8.

Figure 5-7 NoC Configuration Example, Section 6


Configurable NoC Platform

File Help

Comparison Type
(● General) ○ Specific

Compare Operation Name
[ ] _General
(only "0-9", "a-z", "A-Z" and "_" valid)

NoCs involved
C: Project 1
C: Project 2

Select Project Folder
Press "Browse" to Select Comparison Work Folder
C: Browse...

Cancel Back Start

To Compare NoCs...

Table 5-5 Contents of *.cmp* File

Description	**Associated Data or File Names**
Name of the Compare Operation	*Comparison0*
Project Name of the 1st NoC	*projec1*
Benchmark Name of the 1st NoC	*test1*
Architecture Library of the 1st NoC	*lib1*
Project Name of the 2nd NoC	*projec2*
Benchmark Name of the 2nd NoC	*test2*
Architecture Library of the 2nd NoC	*lib2*
Comparison Type	*General or Specific*
Comparison Folder	*User Assigned*

Figure 5-8 Configuration Tree for Comparison

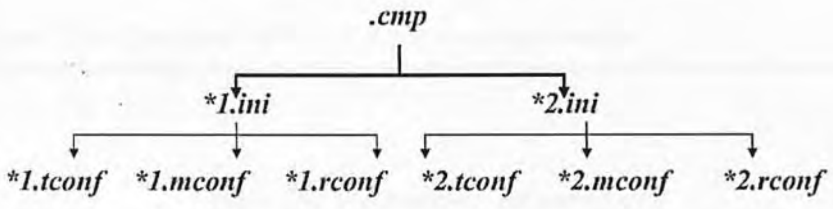


Table 5-6 Contents of *.tconf* File

Description	Associated Strings or File Names
Network Size	$m \times n$
Type of Traffic Pattern	*Uniform, or* *Transpose, or* *Locality, or* *Custom*
Communication Probability	*node_ir.dat*
Group Size of each Node	*{grp_size_i.dat}*
Group Sink of each Node	*{grp_sink_i.dat}*
Group Injection Rate of each Node	*{grp_rate_i.dat}*
Injection Distribution File	*ir_distr.dat*
Flow Distribution File	*lookup_table.dat*
Number of Evaluation Trials	*1 or larger than 3*
Seed Value of Each Evaluation Trial	*Any Integer Number*

Table 5-7 Function of *.mconf* File

Description	Associated Data or File Names
Type of Evaluation Metric	*Latency & Throughput, or* *Virtual Channel Number, or* *Virtual Channel Depth, or* *EVC Efficiency*
Measurement Scheme	*Clock Cycle or Packet*
Period of Warming-Up	*Any*
Period of Validity	*Any*
Specified Flow for Latency & Throughput Report	*Any*
Specified Port for Virtual Channel Number Report	*Any*
Specified Port for Virtual Channel Depth Report	*Any*

Table 5-8 Contents of *.rconf* File

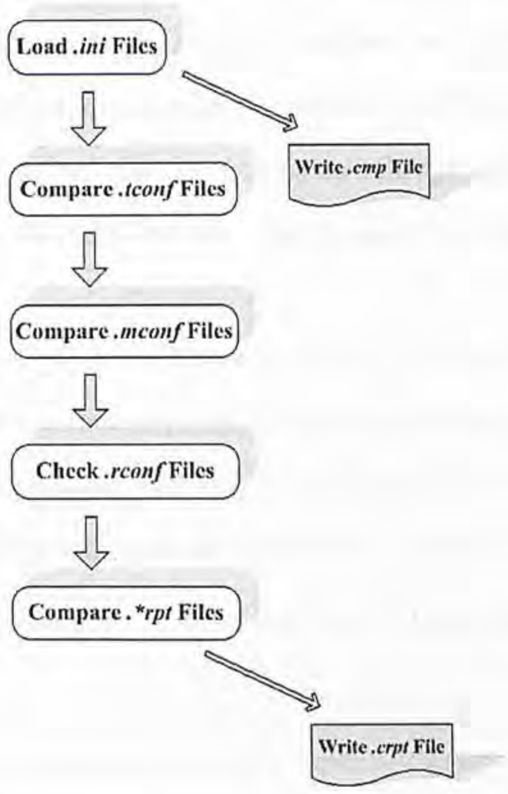
Description	Associated Data or File Names
Number of Evaluation Trials	*1 or larger than 3*
Seed Value of Each Evaluation Trial	*Any Integer Number*
Latency and Throughput Report	*{project1_test1 _lib1.ltrpt}*
Virtual Channel Number Utilization Report	*{project1_test1 _lib1.vcnrpt}*
Virtual Channel Depth Utilization Report	*{project1_test1 _lib1.vcdrpt}*

The contents of *.tconf* and *.mconf* must be identical for two NoCs involved in system comparison. It ensures that the project test is conducted under exactly the same standard. Since they are all generated by system automatically, format mismatch would not occur here. In terms of comparison of *.rconf*, the platform searches the files pointed to by *.rconf* to make sure every requested report file is available. Particular value of the seed number *k* should match each other. This whole operation is named comparison validity check. If validity check fails, operation terminates and warning alarm issues. Unless validity check completes successfully, evaluation reports listed in Table 5-8 are compared and the final comparison report named *comparison0.crpt* is generated under user assigned comparison folder. Figure 5-9 implies the flow of process 601, 602 and 603 and content of *.crpt* file is listed in Table 5-9.

Table 5-9 Contents of *.crpt* File

Description
Name of Compare Operation
Average Network Latency of 1st NoC
Average Network Latency of 2nd NoC
Total Buffer Size of st NoC
Total Buffer Size of nd NoC
Superior Application Choice

Figure 5-9 Flow of NoCs Comparison



5.3 Low-Power Implementation Code Export

A distinctive feature of proposed platform opposed to other platforms is being applicable for real application on hardware, rather than sticking with simulation alone. Many literatures present NoCs evaluation environment or architectural exploration framework, with real application running on. However, silicon area, power consumption and accurate running circumstances under contentions are impossible for estimation due to high-level simulator (developed in e.g. System-C) adopt to emulate router functionality. On the contrary, platform proposed by this thesis can export synthesizable HDL code of well configured NoC architecture. It applies to low-power architecture only because four types of network router are involved and each has its special structure. The network routers of low-latency architecture are all of identical structure thus export of it equals to exposure of entire source code.

As architectures to be integrated into NoC library are required to be described in HDL languages (specific language for each architecture varies), application-specific, synthesizable code can sure be generated. Although generation of configuration file along with original NoC source code is also feasible for synthesize, it believes that tailored code version without configurable features is preferred due to its simplicity. Therefore, every sets of source code of architecture design will be modified to some extent to fit in for this platform feature. That is also why process 401 to 403 are required.

If users select low-power architecture from NoC library to configure a NoC named *project* of size $m \times n$, then exported files for individual router are saved under *$project_path/lp_implementation_code/project_lp_router_i/* ($0 \leq i < m \times n$). Processes involved in implementation code generation are listed in Table 5-10.

Table 5-10 Processes involved of Low-Power NoC Implementation

Process ID	***Operation***
401	Read NoC configurations from GUI section 2, to generate {*numeric_i.dat*} & {*router_parameter_imp_i.dat*}
402	Read *data_types_template.dat*, {*numeric_i.dat*}, {*router_parameter_imp_i.dat*}, read all *.mv* files from *./lp_implementation_sv_lib/* to generate synthesizable individual network router under *$project_path/lp_implementation_sv/*
403	Read *vc_router_top_template.dat*, to generate synthesizable top network router code *mesh_noc_router.v* under *$project_path/lp_implementation_sv/*

- Process 401

Given a network router of *p* input ports, with *q* virtual channels at each port, to define buffer status of all virtual channels of a single network router, two signals are declared in System Verilog by following lines:

```
input  logic  [p-1:0][q-1:0]  fifo_empty;
input  logic  [p-1:0][q-1:0]  fifo_full;
```

Above is the common signal declaration in *.v* files of simulation library. Both *p* and *q* are network parameters that can be configured by *network_parameter_sim.v*. However, when exporting synthetic code for individual network router, numeric expression is expected rather than the parameterized.

```
input  logic  [4:0][3:0]  fifo_empty;
input  logic  [4:0][3:0]  fifo_full;
```

Preparation of multiple numeric versions of NoC library is infeasible due to so many parameters involved. Instead, all parameterized System Verilog data type are replaced by system defined data type. For instance, above declarations are substituted by below

```
input  logic  maxport_maxvc_2d_t  fifo_empty;
```

In the meantime, a line is added into the file *data_types_template.dat*,

```
maxport_maxvc_2d_t      [MAX_ROUTER_RADIX-1:0][TVCS_PERPC-1:0]
```

Likewise, other data types in *.v* files of simulation library are substituted as well and *data_types_template.dat* keeps expanding. The whole implementation NoC library is then obtained as well as finalized *data_types_template.dat* for parameter substitution.

Then process 401 proceeds as,

a) Read all parameter values from *network_parameter_imp.dat*
b) Read contents of *data_types_template.dat*, substitute all parameters by the values in *network_parameter_imp.dat*
c) Create files named {*router_parameter_imp_i.dat*} where *i* suggests index of network routers. Write in contents of *network_parameter_imp.dat*, and then supplement updated contents of *data_types_template.dat.*

- Process 403

This process aims for generation of *mesh_noc_router.v* which is the top module for the set of network routers to export. Since each network router has its own top router module and *.v* file, below contents have to be inserted into the file represent each node. This content saves in the file *vc_router_top.tpl.*

```
vc_router_top_iijj
vc_router_top_iijj_inst
   (.clk(clk),
    .rst_n(rst_n),
    .flits_in(flits_in[ii][jj]),
    .credits_in(credits_in[ii][jj]),
    .evcflags_in(evcflags_in[ii][jj]),
    .credits_out(credits_out[ii][jj]),
    .flits_out(flits_out[ii][jj]),
    .evcflags_out(evcflags_out[ii][jj]));
```

ii and *jj* would be replaced by actual row index or column index. Process 403 reads network size, e.g. $m \times n$, then inserts this module into duplicated *mesh_noc_router.v* file $m \times n$ times with *ii* and *jj* replaced by actual row index and column index of routers.

Finally, complete synthesizable NoC description is exported or cost estimation via other tools. This procedure is not integrated into platform itself yet. It is considered as a supplement procedure currently for users' decision-making.

Chapter 6 Summary and Future Work

6.1. Summary

Proposed platform provides a systematic environment for NoC architecture exploration. It covers network configuration, synthetic benchmark generation and NoC evaluation as what other platforms offer. Meanwhile, it provides advanced NoC library, graphical interface, buffer optimization technique, architecture comparison mechanism and cost estimation possibility. These features distinguish it from all other competitors.

The two network architectures integrated into platform are low-power and low-latency oriented respectively, with novel techniques proposed recent years for advanced network performance. It offers more flexibility than conventional network router structure to suit various applications. The traffics taken into account include classic traffic pattern such uniform, matrix transpose, as well as locality and full custom. Many real applications can be approximated by synthetic traffic through statistics, which emulates the communication with simplified models. The full custom traffic pattern makes it possible that any synthetic traffic can apply to configured NoC.

The graphical interface is a notable feature of proposed platform. It simplifies exploration procedure of NoC architecture significantly, having all network design details hidden from platform users. It is feasible because application-oriented users concern available configurations more than network implementation itself. It shortens development time and reduces hazard during NoC architecture exploration.

The optimization tool of proposed platform explores best buffer distribution for specific application. The buffer allocation matters because not only resources on chip is precious, but also buffer size at particular network node impacts on overall network performance. A certain amount of optimization time is worthwhile.

Given the reality of synthetic traffic widely used in the network-on-chip research domain, comparison among various NoCs becomes more cautious. If real applications apply, runtime will certainly be used to evaluate NoC performance. But with regard to synthetic traffics, data format, measurement interval, random temporal distribution are factored into final evaluation result. Inconsistency of any parameter could result in suspect comparison outcome. Thus comparison mechanism of proposed platform ensures NoCs comparison is proceeding meaningful, namely the involved NoCs performance are genuinely obtained from exact the same traffics.

6.2. Future Work

Although the proposed platform shows advantages over other similar platforms, it also has some topics for future expansion.

- More NoC architectures

Whether or not a platform can generate suitable NoC for specific application largely rests with the power of NoC library. If the library architectures comprise of numerous features targeting various traffic characteristic, then advanced performance is more likely to be achieved by platform generated NoC. Otherwise, even though all configurations of NoC architectures are finely explored and tailored, inefficient NoC will be ultimately exported. Automatic integration of new architecture is infeasible because corresponding graphical pages are required to develop any way. However, certain standard could be proposed, including format of top parameter files, format of output data file and etc. All of it minimizes integration effort and makes the use of current processes as many as possible.

- More optimization tools

Although buffer optimization tool which is rare observed in other platforms is embedded into proposed platform, more in depth analysis tools that are capable of searching the parameter value of better eventual NoC performance under specific application are deemed to be developed. Buffer optimization is taken into account as it fits to any architecture. While architecture specific optimization tools, such as the one to search suitable deadlock avoidance parameter value are expected. They explore the flexibility potential of existing NoC architectures to generate best application specific network.

- Transplant to Unix System

Current platform is developed under Windows system due to GUI package and third-party tools in use. However, all of it can also be implemented under Unix system. Then more tools are available for power and area estimation. Shorter response time and simulation is another concern. Since all platform processes are already in Perl, the transplantation requires only tiny modification of graphical user interface.

References

[1] P. Guerrier and A. Greiner. A Generic Architecture for on-chip Packet-Switched Interconnections. IN *Proceedings of the conference on Design, automation and test in Europe,* page 256. ACM, 2000

[2] L. Benini and G. De Micheli. Networks on chips: A New SoC Paradigm. *Computers*, 35(1):70-78, 2002.

[3] P. Wielage and K. Goossens. Networks on Silicon: Blessing or Nightmare? In *Digital System Design, 2002. Proceedings. Euromicro Symposium,* pages 196-200. IEEE 2002.

[4] J.C. Villanueva, J. Flich, J. Duato, H. Eberle, N. Gura, and W. Olesinski. A Performance Evaluation of 2d-mesh, ring, and cross bar Interconnects for Chip Multi-Processors. *Networks on Chip Architectures, 2009. NoCArc 2009. 2nd International Workshop,* pages 51-56, Dec. 2009.

[5] B. Tarwar and B. Amrutur. A System-C based Microarchitectural Exploration Framework for Latency, Power and Performance Trade-offs of On-Chip Interconnection Networks. *Network on Chip Architectures,* page 30, 2008.

[6] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallerg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. *Computer,* page 50-58, 2002.

[7] M.M.K. Martin, D.J. Sorin, B.M. Beckmann, M.R. Marty, M. Xu, A.R. Alameldeen, K.E. Moore, M.D. Hill, and D.A. Wood. Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset. *ACM SIGARCH*

Computer Architecture News, 33(4),:92-99, 2005.

[8] L. Papadopoulos, S. Mamagkakis, F. Catthoor, and D. Soudris. Application - Specific NoC Platform Design Based on System Level Optimization. *ISVLSI '07. IEEE Computer Sociaty Annual Symposium,* pages 311-316,2007.

[9] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The Nostrum Backbone – A Communication Protocol Stack for Networks on Chip. 2004.

[10] Song Chai, Chang Wu, Yubai Li, and Zhongming Yang. A NoC Simulation and Verification Platform based on System-C. *Computer Science and Software Engineering, 2008 International Conference,* vol. 3, page 423-426, 2008.

[11] F. Fu, S. Sun, J. Song, J. Wang, and M. Yu. A NoC Performance Evaluation Platform Supporting Designs at Multiple Levels of Abstraction. *Industry Electronics and Applications, 2009. 4th IEEE Conference,* page 425-429. IEEE, 2009.

[12] M. Zhang and C.S. Choy. Application-Specific Express Virtual Channels Insertion for Low-Power NoCs. *Proceedings: APSIPA ASC 2009: Asia-Pacific Signal and Information Processing Association, 2009 Annual Summit and Conference,* page 738-746, 2009.

[13] L. Xin and C. Choy. A low-latency NoC Router with Lookahead Bypass. *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium,* page 3981-3984. IEEE, 2010.

[14] J.H. Bahn and N. Bagherzadeh. A Generic Traffic Model for On-Chip Interconnection Networks. *Network on Chip Architectures,* page 22, 2008.

[15] A. Kumar, L. Peh, P. Kundu and N.K. Jha. Express Virtual Channels: Towards the Ideal Interconnection Fabric. *Proceedings of ISCA-34*, pages 150-161, 2007

[16] K.S. Shim, M.H. Cho and etc. Static Virtual Channel Allocation in Oblivious Routing. *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, pages 38-43, 2009

[17] T. Ono, M. Greenstreet, A Modular Synchronizing FIFO for NoCs, *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, pages 224-233, 2009

[18] Y.C. Lan, S.H. Lo, Y.C Lin, etc. BiNoC: A Bidirectional NoC Architecture with Dynamic Self-Reconfigurable Channel. *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, pages 266-275, 2009

[19] A. Banerjee and S.W. Moore. Flow-Aware Allocation for On-Chip Networks. *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, pages 183-192, 2009

[20] William J. Dally. Virtual Channel Flow Control. *Computer Architecture, 1990. Proceedings., 17th annual International Symposium on*, pages 60-69, 1990

[21] C. Grecu, A. Ivanov, etc. Towards Open Network-on-Chip Benchmarks. *Networks-on-Chip, 2007. NOCS 2007, First International Symposium on*, page 205, 2007

[22] T. Bjerregaard, S. Mahadevan. A Survey of Research and Practices of Network-on-Chip. *Journal ACM Computing Surveys (CSUR) Volumn 38 Issue 1*, 2006

[23] T.C. Huang, U.Y. Ogras and R. Marculescu. Virtual Channels Planning for Networks-on-Chip. *Quality Electronic Design, 2007. ISQED `07. 8th*

International Symposium on, page 879-884, 2007

[24] A. Banerjee, R. Mullins and S. Moore. A Power and Energy Exploration of Network-on-Chip Architectures. *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, pages 163-172, 2007

[25] K. Lee, S.J. Lee and H.J Yoo. Low-Power Network-on-Chip for High-Performance SoC Design. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, volume 14, Issue 2*, page 148-160, 2006

[26] J. Kim, C. Nicopoulos and D. Park. A Gracefully Degrading and Energy-Efficient Modular Router Architecture for On-Chip Networks. *Computer Architecture, 2006. ISCA `06. 33rd International Symposium on*, page 4-15, 2006

[27] M. Mueller, A. Wortmann, etc. The Impact of Clock Gating Schemes on the Power Dissipation of Synthesizable Register Files, *Circuits and Systems, 2004. ISCAS `04. Proceedings of the 2004 International Symposium on, volume 2*, page II-609-12, 2004

[28] http://perl-win32-gui.sourceforge.net/cgi-bin/index.cgi

[29] Z. Lu and A. Jantsch. Traffic Configuration for Evaluating Networks on Chips. *System-on-Chip for Real-Time Applications, 2005. Proceeding. Fifth International Workshop on*, page 535-540, 2005

[30] J. Duato. Improving the Efficiency of Virtual Channels with Time-Dependent Selection Functions. *Conference Proceedings on PARLE`92, volume 10, issue 1*, 1994

[31] T. Krishna, A. Kumar, etc. Express Virtual Channels with Capacitively Driven Global Links. *Micro, IEEE, volume 29, issue 4*, page 48-61, 2009

[32] E. Salminen, A. Kulmala and T.D. Hamalainen. On Network-on-Chip

Comparison. *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, pages 503-510, 2007

[33] W.J. Dally and B. Towles. Principles and Practices of Interconnection Networks. Morgan Kaufmann Publishers. 2007

[34] N. Banerjee, P. Vellanki and K.S. Chatha. A Power and Performance Model for Network-on-Chip Architectures. *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, pages 1250-1255, vol2, 2004