12-2010

Parameterizable network-on-chip emulation framework

Jaya Suseela
University of Nevada, Las Vegas

PARAMETERIZABLE NETWORK-ON-CHIP EMULATION FRAMEWORK

by

Jaya Suseela

Bachelor of Technology
Kerala University
2001

Master of Technology
Amrita Vishwa Vidya Peetham
2003

A thesis submitted in partial fulfillment
of the requirements for the

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

Graduate College
University of Nevada, Las Vegas
December 2010

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Jaya Suseela

entitled

Parameterizable Network-on-Chip Emulation Framework

be accepted in partial fulfillment of the requirements for the degree of

Master of Science in Engineering
Department of Electrical Engineering

Venkatesan Muthukumar, Committee Chair

Emma Regentova, Committee Member

Mei Yang, Committee Member

Ajoy Datta, Graduate Faculty Representative

Ronald Smith, Ph. D., Vice President for Research and Graduate Studies
and Dean of the Graduate College

December 2010

ABSTRACT

Parameterizable Network-on-Chip Emulation Framework

by

Jaya Suseela

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

Networks-on-Chip (NoCs) have been proposed as a promising solution to complex on-chip communication problems. But there is no public accessible HDL synthesizable NoC framework which connects industrial level cores and runs real applications on them. Moreover, many challenging research problems remain unsolved at all levels of design abstraction; design exploration of NoC architecture for applications, scheduling and mapping algorithms, evaluation of switching, topology or routing algorithm for efficient execution of application and optimizing communication cost, area, energy etc Solution to solve the above problem calls for the development of synthesizable, parameterizable NoC Framework that would evaluate and implement the above outstanding research problems and algorithms with minimum ease and flexibility.

The proposed NoC Framework has been used to specifically evaluate the following algorithms or variations in architecture: i) Evaluate Switching Algorithms compare latency, congestion, area and power of Wormhole (WH) and Store and Forward (SF) switching, ii) Efficient Router Architecture: Proposed an efficient Virtual Channel architecture with loopback for SF routing is introduced to improve throughput, latency and area, iii) Static routing algorithm: Proposed a simple and efficient routing algorithm

called “Mirror Routing” for Torus architectures. This helps in reducing congestion and the routing algorithm is also deadlock free, iv) Adaptive Routing Algorithm: Proposed and evaluated an adaptive routing algorithm for WK topology.

The simulation results show Wormhole Routing with better latency than Store and Forward. Area and Power usage is also relatively less for Wormhole Routing. Study on different traffic scenarios with different Virtual Channel architectures in Store and Forward routing shows considerable improvement in latency in Virtual Channel architecture with loopback. Also it is proved that the proposed Mirror Routing algorithm is able to handle a single congestion or fault in routing path. The latency increases with increase in size of Torus structure. The Adaptive routing algorithm proposed for WK Topology results in increase in latency but can be considered in scenarios where the receiver node at the congested link is comparatively slow or when the fault in link is permanent.

ACKNOWLEDGEMENTS

If with profound satisfaction I could present this research work in its present form, I owe a great deal to several people who showered unstinted support to me throughout. I need no second thought to place on record in resplendent letters the overwhelming patronage granted to me by these noble souls. In the array of names, sparkle an immaculate personality, Dr. Venkatesan Muthukumar, Associate Professor (ECE), UNLV, and Chairman of my Advisory Committee whose wide knowledge and logical way of thinking have been of great value for me. His understanding, encouraging and personal guidance have provided a good basis for the present thesis. In unequivocal terms I express my heartfelt gratitude to him. I was fortunate to have the guidance of three prominent professors, Dr. Mei Yang, Dr. Emma Regentova, and Dr. Ajoy Datta as the members of my research advisory committee. I have no reservation in acknowledging the valuable advises rendered by them. I had the privilege to enjoy the voluntary help from my lab-mates Ajay Mandava and Kranti Kumar. I thank them for their gestures of goodwill and friendliness.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

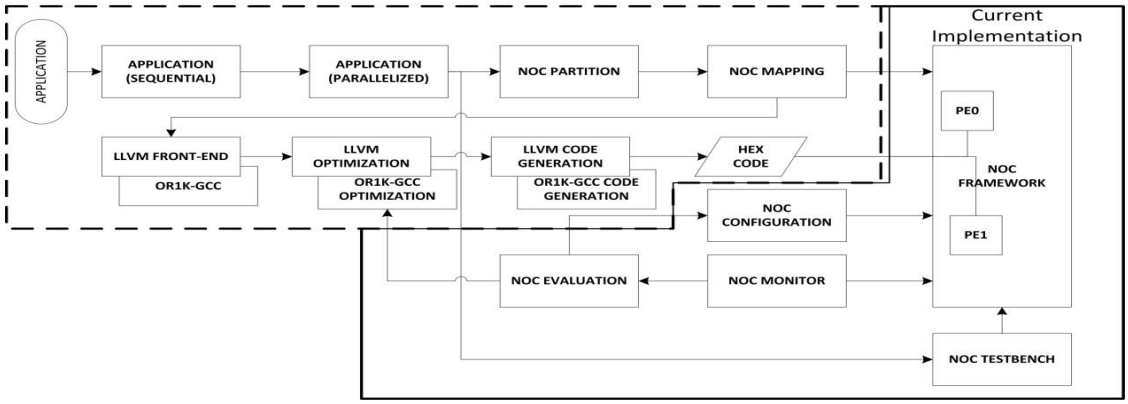
INTRODUCTION

Every chip design has four major aspects: Computation, memory, communication and Input/Output. With the scaling of microchip technologies, implementation of Systems on Chip (SoC) computation became cheaper, that lead to the introduction of multi-core and multi-processor paradigms, while the communication infrastructure encountered physical limitations such as delay and power due to long wires, parasitic capacitance, etc, especially in the Deep Submicron (DSM) process. As the number of processors on a SoC increased, bus arbitration and bandwidth became the bottleneck. Networks on Chips (NoC) has been proposed [6],[10],[11] as an attractive alternative to traditional dedicated wires to achieve high performance, modularity and to eradicate the bottlenecks in the communication infrastructure. The idea was widely accepted mainly due to the readily available networked communication abstraction models such as TCP/IP or OSI models.

To date, several prototype NoCs have been designed and analyzed in both industry and academia [3] but there exists no public accessible HDL NoC simulator or synthesizable framework which connects industrial level cores and run real world applications on them. Moreover, many challenging research problems remain to be solved at different design abstraction levels, from the physical link level through the network level, and all the way up to the system architecture and application software.

The main focus of this work is the design and implement of a parameterizable (flexible) synthesizable NoC framework, which can evaluate the tradeoffs between different architectural and algorithmic designs like: router architecture, switching

techniques, topology architecture and routing algorithms. The main components of the proposed NoC Framework includes: Processing Element (PE) [OpenRisc OR1K processor], TIMERs, UARTs, Instruction (I-MEM) and Data Memory (D-MEM), Core and Network Interfaces, Router, and the Channel. Conceptually the proposed NoC framework performs the following functions, i) Design Space Exploration (DSE) and ii) Evaluation of trade-offs between power, area, latency etc. for various design variations, while adhering to application requirements. The design of the NoC framework is divided into four procedural levels of abstraction (models), i) Application Model (which includes Traffic Generation and Monitoring), ii) NoC framework architecture model and its components iii) Communication Flow Model (that models communication between different NoC components and iv) Algorithm Models (which model switching and routing algorithms in the NoC architecture).



.

Figure 1. The Complete NoC Emulation Framework

The complete emulation framework of the proposed NoC framework is shown in Figure 1. The main contribution of this work is also shown in the Figure 1

The main contribution of work is the implementation of a parameterizable and synthesizable NoC framework in Verilog. The framework is used to: i) compare latency, congestion, area and power of Wormhole (WH) and Store and Forward (SF) switching, ii) Evaluate the proposed loopback modification in Virtual Channel (VC) architecture for SF routing that provides improved throughput, latency and area, iii) Evaluate the proposed routing algorithm called "Mirror Routing" for Torus architectures the reduces congestion and also deadlocks, iv) Evaluates an adaptive routing algorithm for WK topology.

The remainder of the thesis is organized as follows. Chapter 2 details the literature review done in the related field. Chapter 3, details the implementation following the abstraction levels discussed earlier. The concept of Virtual channels with loopback, its implementation details, the Mirror routing for torus structures and adaptive routing for WK topology are discussed in Chapter 4. Chapter 5, presents simulation and synthesis results, comparative study and performance analysis. Chapter 7 concludes the thesis.

CHAPTER 2

LITERATURE REVIEW

The NoC framework is divided into four procedural levels of abstraction, such as i) Application model ii) Architecture model and components iii) Communication Flow model and iv) Algorithmic model.

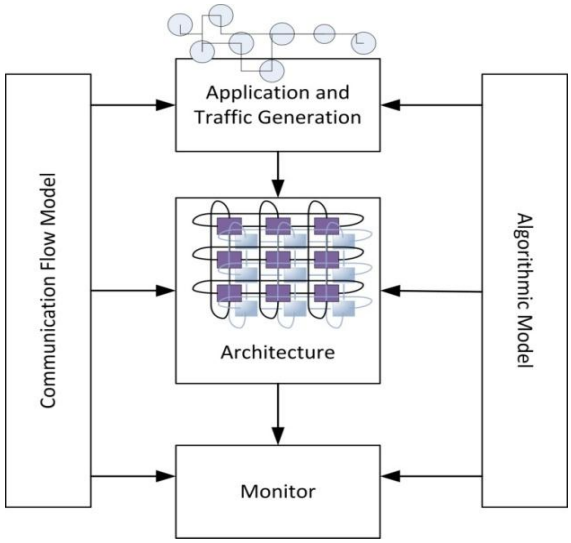


Figure 2. NoC Levels of Abstraction

The Application level encompasses three main components, i) Application Mapping and Scheduling ii) Traffic Controller, and iii) Monitor. The Architecture model includes, i) NoC Framework including, Processing Element (PE) such as OR1K processors, TIMERs, UARTs, Instruction (IMEM) and Data Memories (DMEM), Core and Network Interfaces, ii) Router and iii) Network Topology. The Channel Flow model defines the Control and Data flow in: i) System level, ii) Network Level and iii) Data link Level. The

System level defines the flow between Master to Slave, Slave to Master and between Masters. The control flow within the Router is defined in Network layer. The lowest level is the Data Link level. This level deals with encoding, decoding and synchronization of packets or flits. The Algorithmic model defines the various switching and routing algorithms used in data and control flow.

2.1 Application Model

Application Model deals with Traffic generation, Mapping and Scheduling of generated application tasks to NoC PEs and determining the performance in NoC using Traffic Monitors.

2.2.1 Application Mapping and Scheduling

Mapping is the assignment of applications tasks to NoC architecture and Scheduling is the ordering of tasks on the assigned cores. Most obvious objectives of mapping are i) Minimize the average hop distance, ii) to minimize the maximum energy consumption over the communication links of the NoC architecture iii) Balance work load and hence maximize timing performance. Figure 2, [11], shows an example of optimally mapping task graph onto 2-D mesh architecture.

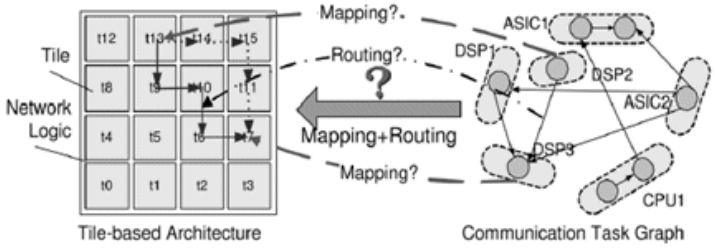


Figure 3. Tile-based Architecture and illustration of the mapping/routing problems.

Recently several application mapping and scheduling algorithms targeting at the NoC architecture have been presented [58], [61-64]. Without coordinated scheduling on both computation and communication workloads, speculative mapping algorithms may not generate effective runtime behavior. Algorithms proposed in [57], [59-60] handles both application mapping and scheduling process together. A novel off-line non-preemptive static Traffic aware scheduling (TAS) policy proposed in [60] determines inter process communication latency dynamically based on the application mapping and PE interaction.

2.1.2 Traffic Generation and Monitoring

The network performance highly depends on the characteristics of the data exchanged through the NoC. Therefore, traffic modeling is one of the most important aspects in NoC design to fully evaluate the architecture or communication flow. The traffic can be characterized by two aspects: i) Spatial or topological distribution, describing the source-destination communication and ii) temporal or traffic nature, fitting the traffic generation rate with a statistical distribution [34]. The most popular traffic generators uses hot spot, transpose and random patterns [43],[46]. All these previous Traffic Generators assumed that the traffic nature of the NoC is Poisson distribution. The Poisson distribution assumes that the number of packets generated during a fixed period of time occurs with a known average rate and independently of the time. However, the nature of many of the current and future pervasive applications running on NoC-based designs is expected to be heterogeneous [44], [45]. It possesses diverse statistical properties which include complex temporal correlation and non-Poisson distributions. The work of [44] shows that

the NoC traffic may exhibit Long Range Dependence (LRD). It means that the packet generation can be a time-dependent process that shows statistically significant correlations across large time scales. This property may directly affect the network resource allocation and management, with strong impact on queuing and nature of congestion [44].

The Traffic Generators (TGs) are used to emulate the behavior of the computation components of the SoC. They are designed to establish different traffic conditions. The NoC TGs can be classified into 3 categories [34]: i) application specific, by using the communication trace, a previously recorded communication of the NoC application [44], [45], [47], [48]; ii) parametrical, where the communication conditions are specified by the designer [47], [48]; and iii) stochastic, by using an statistical distribution [33]. Application specific generators require the specification of all the computation components of the NoC and the simulation of the application. It can be precise but it is restricted by the length of the used input traces, moreover, it cannot handle behaviors that are dependent on input data sets. Such limitations promote the use of the parametrical and stochastic traffic generators [43]. These generators allow the evaluation of the NoC in its early stage of development, and on its performance. In [49], stochastic processes are used for generating transaction sizes and transaction delay using several statistical laws (Poisson, Exponential, and Normal distribution).

Like Traffic Generators, Traffic Monitors (TM/M) are designed in NoCs to probe the internal traffic of NoC cores at run-time for accurate performance evaluation and refinement of application specific NoCs. The observations from TM can be used to

change buffer sizes or perform route modifications and hence reducing cost and latency [56]. In [32], hardware-based approach is used for performance evaluation of NoCs in FPGA. They use a synthesizable traffic generator (TG) to generate and inject packets into/from the NoC, while the Traffic Receptor (TR) module eject these packets from the NoC and collect data for performance evaluation.

2.2 Architecture Model

The Architecture Model defines the basic components of NoC. It defines the NoC Framework, the Router Architecture, including the Virtual Channel models and the Network Topology.

2.2.1 NoC Framework

The practical implementation and adoption of NoC design faces multiple issues. Moreover there is no public accessible HDL NoC simulator or synthesizable framework which connects industrial level cores and runs real applications on them. Orion [22] and LUNA [23] are two NoC simulators especially developed for power simulation of on-chip interconnection networks and they do not consider computational cores (PEs). FAST [21] is a functionally accurate NoC simulator limited to IBM's proprietary Cyclops-64 architecture. SICOSYS [28] is a general-purpose interconnection network simulator that captures essential details of low-level simulation. RSIM [27] simulates shared-memory multiprocessors and uniprocessors designed for high instruction level parallelism; it includes a multiprocessor coherence protocol and interconnect, and models contention at all resources. NoC simulators such as NNSE [24], Noxim [25] and NIRGAM[26] have flexibilities in configuring parameters of on-chip network and capabilities to obtain

performance metrics. But these simulators are based on SystemC and are not synthesizable. In [37], a framework for MPSoC NoC system modeling, simulation and evaluation, based on System C models is presented. Other frameworks are based on object-oriented languages, like the work presented in [38], where a C++ library is built on top of SystemC, or based on the Matlab simulation environment [39]. Other approaches generate NoC topologies getting the application graph representations or application descriptions as the starting point, using analytical and/or heuristic methods. Examples of these are SUNMAP [40], which are based on the Xpipes [41] NoC generator, which creates topologies modeled in SystemC, or in [42], where systems are specified in XML. These approaches are very suitable for system design early stages as they permit faster design space exploration. But performance analysis at RTL level is very important during design phase of NoC. In this work a parameterizable and synthesizable NoC framework is developed in Verilog. NoC designs are sensitive to many parameters such as router architecture, topology, buffer sizes, routing algorithms and flow control mechanisms.

2.2.2 Router Architecture

The Router Architecture largely depends on the switching technique and the network topology. A conventional NoC Router Architecture consists of: i) Input and Output Ports, ii) Input / Output Buffering and iii) Routing Unit. In general, NoC routers may have any number of input or output ports. However, the network topology ultimately determines the number of ports for each router. Most implementations [78-80] use five ports, four from four cardinal directions (North, South, East and West) and one from Processing

Element (PE). The NoC realization in [77] uses 4 types of routers that have 9, 6, 5 and 4 input/output ports.

Each port can have either input buffering or output buffering or both. A number of researchers reported various buffering strategies for contention resolution, to reduce area and design complexity. Buffers are basically FIFO which stores the packet as soon as it is available at its port. Input buffering normally has a lower complexity and consequently a lower cost of implementation [72].Also, the switch fabric and the memory at the inputs of an input-queued switch need to operate as fast as the line rate, whereas output buffering has to operate 4 or 5 times faster than the line rate.

Increasing buffer depth obviously can improve NoC performance. But increase in buffer depth results in increase in power consumption. According to the results obtained in [76], as the Input Buffer (IB) depth increased, performance of the network improved. But after the depth of 4 packets, performance improvement caused by IB depth's increase became smaller. So the optimum IB depth is determined as 4 packets [76]. However increase in Output Buffer (OB) depth made only little improvement to the network performance.

Depending on the route/destination information in the packets stored in Buffers, the Routing Unit routes them to the adjacent router. Each Router Unit integrates an arbiter module to resolve access conflict to the ports. Most of the arbiters use Round Robin (RR) [82-83] arbitral mechanism to respond to the requests from the ports. Chang Wu et. al. [81] discusses an arbitral mechanism based on lottery algorithm, where in the ports gain

tokens named ‘lottery’, based on the algorithm and the total number of lottery attained determines the communication priority for that port.

Virtual Channel

Two main resources compose interconnection network: buffers and channels/links. Typically, a single buffer is associated with each channel. However, a channel can be multiplexed in ‘n’ Virtual Channels (VC). VCs provide multiple buffers for each channel, increasing the resources allocation for each packet. The insertion of VCs also enables the use of special policies to allocate the physical channel bandwidth, allowing support to Quality of Service (QoS) [71]. Following summarizes the advantages resulting from effective use of VCs in NoCs.

i) Network Deadlock/Livelock: Since VCs provide more than one output path per channel there is a lesser probability of deadlock.
ii) Performance improvement: A packet or a flit at an input port of a router has to wait until that router is free. However VCs can provide another virtual path for the packets to be transmitted through that route, thereby improving the performance of the network. A higher buffer capacity and a larger number of VCs in the buffer will reduce network contention, thereby reducing latency.
iii) Reduced wire cost: VC provides an alternate path for data traffic, thus it uses the wires more effectively for data transmission.

Lot of research [65-70] has been done on the efficient implementation of VCs to result lower latency and power with optimum area. M.H. Ghadiry et. al. [36] compares several routing algorithm with several number of VCs and concludes a linear relation between

number of virtual channels and energy dissipation. Under-utilized VCs results in energy dissipation. Since the number of hops the packet has to travel to reach the destination is constant in minimal routing algorithms, energy dissipation is same for algorithms with same number of VCs. Increase in number of VCs improve performance but also cause to increase latency.

2.2.3. Network Topology

Network Topology refers to the way in which the Routers are connected in a NoC. There are two types of NoC: Regular and Irregular. This refers to whether the network topology is arranged in a regular network or an ad-hoc interconnection network. The ability of network to efficiently disseminate information depends largely on the topology. Application mapping and routing protocol are largely dependent on topology. Network latency, buffer size, power and area will become the bottle neck while deciding the layout. Linear, Mesh, Torus [2] are the most widely used homogenous topologies while Spidergon [30], [77] and WK recursive routing [26] is gaining more importance.

The WK-recursive networks [26] are a class of recursively scalable networks with many desirable properties. They offer a high degree of regularity, scalability and symmetry, which very well conform to a modular design and implementation of distributed systems involving a large number of computing elements. $WK(N_d,L)$ represents a WK topology with node degree N_d and expansion level L. For this family of topologies, starting from a $WK(N_d,1)$ and recursively arriving to an expansion level L, the following relations hold:

Total number of real nodes, $n = (N_d)^L$

Total number of links, $P = N_d *((N_d)^L -1)/2$

Diameter of WK (N_d, L), $D = 2^L -1$

M.H. Ghadiry et. al. [26], propose a VLSI implementation of the WK-recursive networks, and a develop a routing algorithm for the topology. The algorithm does not address the use of virtual channels. G. D. Vecchia et. al. [27], compare WK-Recursive with Mesh topology in case of power and latency and proved that the latency of WK recursive network for low traffic loads is superior to mesh topology. The power consumption in the WK-recursive is also less than that of the mesh network for low traffic loads while the power consumption in the two networks is almost equal for high traffic loads.

The power consumption of the NoC architecture is determined by both the physical links and routers. The power consumption of a physical link is dependent upon the length of the link, which in turn, is governed by the number of hopes the packet has to travel to reach its destination. Hence routing topologies plays a major role in NOC system design [9, 25].

A. Scherrer et. al. [24], compares power and latency for several topologies of NoC. Assuming a uniform traffic, this paper proves that for communication traffic of 1Gb/s the most power efficient topology is a fully connected structure and for medium traffic of around 25Gb/s, torus topology has the minimum power consumption and Mesh proves the most power efficient solution for high end traffics (>25 Gb/s).

Specifications of (2N-1)-by-(2N-1) 2D Mesh topology [55] are summarized in Table 1.

Table 1. 2D Mesh Topology Specifications

Total number of routers	$(2N-1)^2$
Total number of 5-port routers	$3N^2 -4N+1$
Total number of 9-port routers	N^2
Total number of short links	4(N-1)(2N-1)
Total number of long links	2N(N-1)

2.3 Algorithmic Model

Algorithmic Model details the Switching and Routing algorithms used in NoC Framework. Switching Technique defines the way in which the message/packet is forwarded in the network and Routing algorithm governs the path/route.

2.3.1. Switching Techniques

Switching in NoC is a networking technology that provides a temporary, but dedicated connection between two links/channels or nodes. Switching Techniques are broadly classified into circuit switching and packet switching based on network characteristics. Circuit switched networks reserve a physical path before transmitting the data packets, while packet switched networks transmit the packets without reserving the entire path. In packet switching, each packet finds its own route to destination based on the information it carries, such as the source and destination addresses. Packet switched networks are further classified as Store and Forward (SF), Wormhole (WH) and Virtual Cut Through (VCT) switching.

Store and Forward switching first stores incoming packets into the input buffer. Once a packet is received entirely, the router examines the header of the packet and queues the

packet in the Input Buffer. Each router has to wait until the entire packet is received. The main advantage of this approach is that they are applicable to large scale congested networks and NoC architecture with simple router logic.

Wormhole routing is a special case of circuit switching. Instead of storing a packet completely in a node and then transmitting it to the next node, wormhole routing operates by advancing the head of a packet directly from incoming to outgoing channels of the routing node. A packet is divided into a number of flits for transmission. The size of a flit depends on system parameters, in particular, the channel width. The header flit governs the route. As soon as a node receives the header flit of a message, it determines the next routing path and begins forwarding flits through that channel. As the header advances along the specified route, the remaining flits follow the same channels in a pipeline fashion. Because most flits contain no routing information, the flits in a message must remain in contiguous channels of the network and cannot be interleaved with the flits of other messages. When the header flit of a message is blocked, all of the flits of a message stop advancing and block the progress of any other message requiring the channels they occupy.

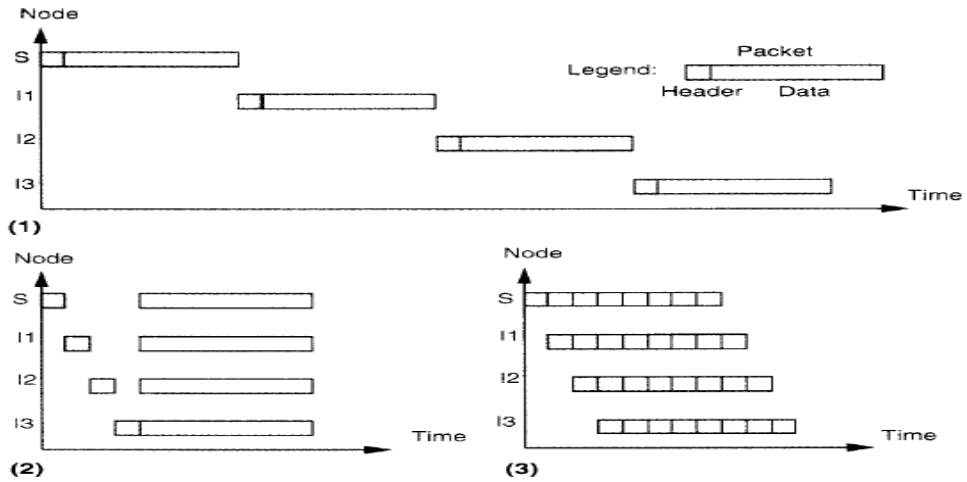


Figure 4. Comparison of different switching techniques: (1) Store and Forward Switching, (2) Circuit Switching (3) Wormhole Switching.

The major differences between these two techniques are on packet latency and size of packets. Small packets underutilize the network and increase segmentation and reassembly overhead, while large packets results in network congestion and buffer saturation [50]. Figure 4, borrowed from [8], illustrate the difference in latency between wormhole and store and forward switching.

2.3.2. Deadlock

The potential for deadlock arises in networks when cyclic buffer dependencies develop from the topology and routing algorithm of the network. Deadlock can occur in store and forward switching. Different scenarios of deadlock in store and forward switches are described in [15]. Freedom from deadlock and livelock [12] are crucial for NoCs, since deadlock/livelock detection and recovery mechanisms are expensive and they may lead to unpredictable delays.

Creating deadlock-free routers for arbitrary topologies without performing analysis ahead of time is shown in [18] by Toueg and Ullman. They present the Forward-Count Controller, which only accepts packets that have fewer hops than the free buffer slots in the controller. This technique requires a bounded path length but appears to be suitable for deterministic or adaptive routing.

Gunther et. al.[19], explains a technique using buffer ordering for preventing deadlock in store-and-forward (SAF) networks. The key idea is to have a monotonic ordering assigned to buffers that packets must follow. Gunther describes a descending buffer number intuitively as causing packets to continuously "drain" from the network. Gopal et. al [20], presents one of the more practical versions of this technique using graph coloring. From the network topology, each node is assigned a number that is different from its neighbors' numbers. When a packet is injected into the network, it is stored in buffer 0. Each time it is routed to a node with a lower graph number, its buffer number increases. Since the contiguous flits of a packet are always contained in the same or adjacent nodes of the network, wormhole routing is highly prone to deadlocks. The technique of Virtual Channels (VCs) allows deadlock-free routing to be performed in any tightly connected interconnection network. This technique involves splitting physical channels on cycles into multiple virtual channels and then restricting the routing so the dependence between the virtual channels is acyclic.

2.3.3 Routing Algorithms

Routing algorithms define the path taken by a packet between source and target switches. They must prevent deadlock, livelock and starvation [16-17] situations.

Routing algorithms can be classified based on (i) where the routing decisions are taken; (ii) how a path is defined and (iii) the path length.

Based on the node where routing decisions are taken, it is possible to classify the routing as Source Routing or Distributed Routing. In Source Routing, the whole path from source to destination is determined at the source node, while in Distributed Routing each switch receives a packet and defines the direction to send it. In Source Routing, the header of the packet has to carry all the routing information, increasing the packet size [9]. In Distributed Routing, the path can be chosen as a function of the network instantaneous traffic conditions. Distributed routing can also take into account faulty paths, resulting in fault tolerant algorithms.

Depending how a path is defined, routing can be classified as deterministic or adaptive routing. In deterministic routing, the path between every pair of source and destination is fixed. In adaptive routing, the path is a function of the network instantaneous traffic [4] and thereby increases the number of possible paths usable by packet to arrive to its destination. Depending on whether the algorithm can use all the possible paths between source and destination, adaptive algorithms are classified as partially adaptive or fully adaptive. Hence in presence of fault, fully adaptive algorithms have less packet drop than deterministic routing algorithms [36]. However, deadlock and live lock situations can happen in fully adaptive algorithms[8], which limit its usage. Adaptive routing algorithms has more complex routing logic and more power consumption while deterministic routing algorithms has simple routing logic and less power consumption[36].

Regarding the path length criterion, routing can be minimal or non-minimal [8,7]. Minimal routing algorithms guarantee shortest paths between source and target addresses. In non-minimal routing, the packet can follow any available path between source and target. Non-minimal routing offers great flexibility in terms of possible paths, but can lead to live lock situations and increase the latency to deliver the packet. Hence minimal routing helps not only in reducing the energy consumption of communication, but also to keep the network free from the live lock.

Implementation complexity and performance requirements are two major concerns in selecting the routing strategy. Compared to adaptive routing, deterministic routing requires less resource while guaranteeing an orderly packet arrival. On the other hand, adaptive routing provides better throughput and lower latency by allowing alternate paths based on the network congestion [14]. However, out-of-order message arrival remains an important problem associated with adaptive algorithms. Deterministic and partially adaptive algorithms based on the turn model [13], guarantee free deadlock and live lock operation, while fully adaptive strategies require extra precaution.

Deterministic minimal routing algorithm such as XY routing avoids deadlock [28]. Flits are first routed in the X direction, and then in the Y direction. West first, North Last (NL) and Negative first are different partially adaptive routing algorithm based on turn models. Turn model prevents deadlock by prohibiting turns. Mello, A. et. al. [50], prove that the deterministic XY routing algorithm outperforms partially adaptive North last, Negative first and West first algorithms for medium to large NoCs. A comparative study done by W.J.Dally [51], proves that full or partial adaptive algorithms not necessarily

benefit wormhole routing. Their simulation results indicates that the partially adaptive NL algorithm perform more badly than non-adaptive e-cube routing algorithm for all three traffic patterns (hotspot, uniform and local). M.H.Ghadiry et. al. [36], compare the node load caused by several routing algorithms in presence of a blocking or a fault in a node and concludes that the e-cube routing algorithm has better load balancing than other algorithms. Compressionless Routing prevents deadlock by using fine grained flow control and back pressure of wormhole routing.

For determining the most effective scheduling algorithms and suitable routing topologies we need a good idea of traffic in NoC. A flexible emulation environment is implemented on FPGA, in [5], which helps in exploring and monitoring the traffic, for different application specific topologies. Lot of research work has been done in this field [21-23]. P.Poplavko et. al. [35], consider dynamic application running on a multiprocessor NoC as a set of independent jobs and propose exact timing models that effectively model both computation and communication of a job.

CHAPTER 3

NOC FRAMEWORK

This chapter focuses on the architecture of NoC Framework and details every module from top level to the lowest Link level. Modules are described from the implementation perspective.

The proposed NoC Framework consist of five main modules; i) Processing Architecture, ii) Communication Infrastructure iii) Communication Paradigm iv) Monitor module and v) Traffic Generator module.

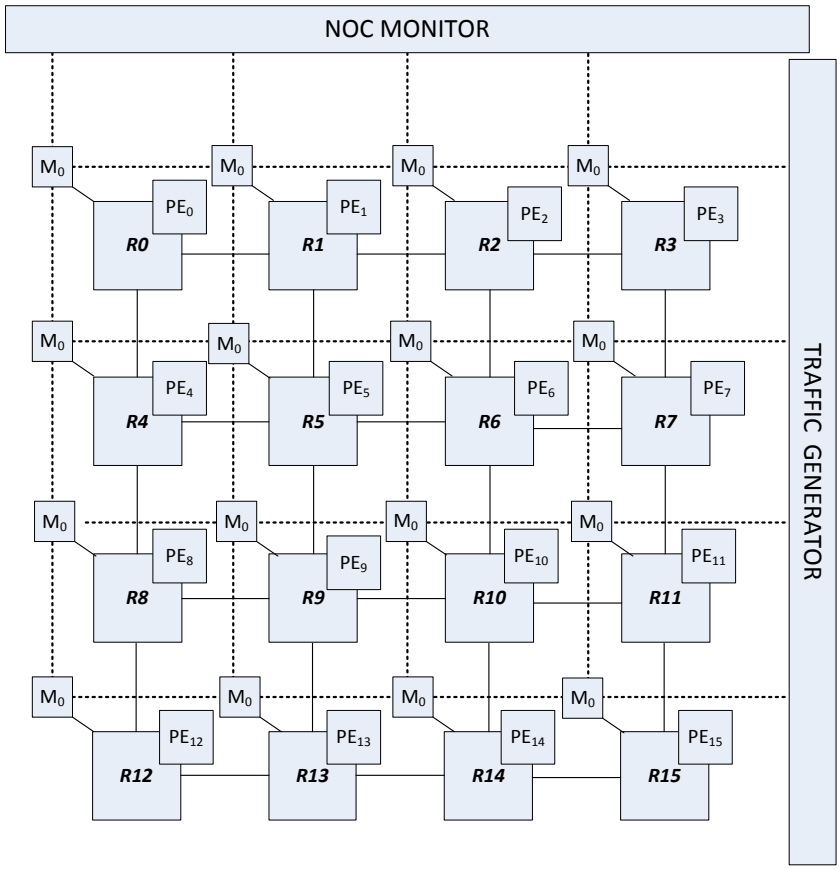


Figure 5. The NoC Framework

Figure 5 shows the proposed NoC Framework and the different modules of it.

The Processing Architecture module consists of Processing Element (PE), and Network Adapter module. The Communication Infrastructure consists of i) Network Topology and ii) Routing Node. The Communication Paradigm describes the Switching Techniques and Routing Algorithms employed in the NoC Communication Infrastructure. The Monitor module includes two modules; a Node Monitor, which monitors the activities in a Routing node and a NoC Monitor that monitors the communication in the Framework.

3.1 Processing Architecture

The NoC design consists of several Master/Slave Processing Elements (PEs) connected to the Communication Infrastructure through a Core/Network Interface. Figure 6 explains the components of the module.

The PEs can be a master PE or slave PE depending on whether it can initiate a message transfer or respond to a request. Only master PEs can initiate a message transfer. Slave PEs responds to the requests from master PE either by sending back the requested signals/data or by saving the received information. UART, TIMER, Instruction/Data Memory are considered as slave PEs while the master PEs used in the design are capable of performing Arithmetic and Logical Operations. Core/Network Interface receives signals from PEs and generates packets to be sent to the Communication Infrastructure. Hence, the main function of the Interface module is to transform the data to and from the format required by underlying Infrastructure.

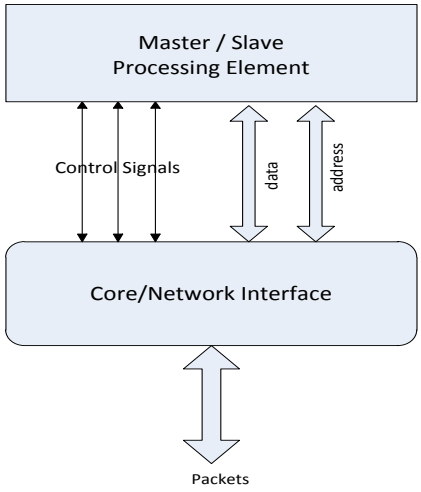


Figure 6. Processing Architecture

3.1.1 Processing Elements

The Framework uses several master and slave PEs. Slave PEs includes UART, TIMER, Single and Double Data Rate data memories (D-MEM), Instruction Memory (I-MEM) etc. The position of Slave PEs in topology, its operating frequency can be modified. Each master PE consist of one OpenRisc 1000 (OR1K) processor connected to an Instruction memory (IMEM) through a Wishbone bus (see Figure 7). The OpenRisc 1000 (OR1K) is a 32-bit Opensource processor. Its open and modular architecture allows a spectrum of chip and system implementations at a variety of price/performance points for a range of applications. Wishbone is open source hardware processor bus with many desired properties which are discussed in the following sessions.

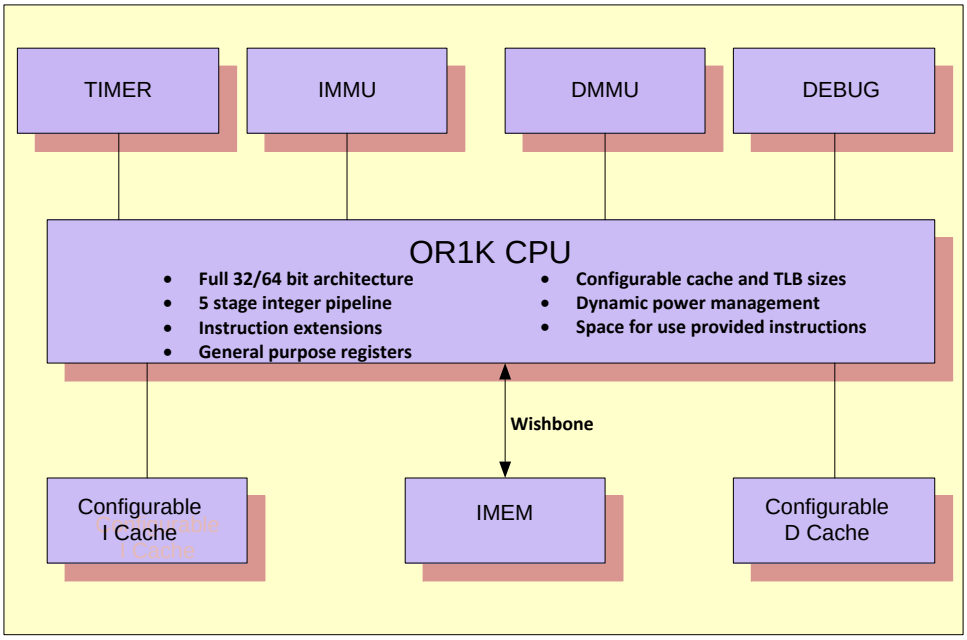


Figure 7. A Master PE

IMEM is a block ram with 8bit data bus and 32 bit address bus. The instructions to be executed by the core are loaded in IMEM.

OpenRISC 1000

The OpenRisc 1000 (OR1k) is a 32-bit load and store, Opensource, ARM9 based RISC embedded processor with 5 stage integer pipeline, virtual memory support (MMU) and basic DSP instruction support. The architecture defines several features that are quite useful for networking and embedded computer environments. Most notable features are: several instruction extensions, a configurable number of general-purpose registers, configurable cache and TLB sizes, dynamic power management support, and space for user-provided instructions. Moreover OpenRISC processor shows better performance per clock cycle than MicroBlaze in the Stanford benchmark and is therefore considered a

more efficient architecture than the MicroBlaze architecture [2]. MicroBlaze is significantly more efficient per area unit than OpenRisc, but it is highly optimized for Xilinx FPGAs. Following few sections explains certain special features of OR1K processor that paved way in choosing it as the primary master PE in the design.

Memory Management Unit

Memory Management Unit (MMU) is one of the most critical modules in OR1K that performs translation of virtual addresses to physical addresses (virtual memory management), memory protection, cache control, bus arbitration, etc. For user programs to execute in virtual address space, the memory management unit (MMU) must be enabled. OR1k MMU includes the following principal features:

1) Support for Effective Address (EA) of 32 bits and 64 bits
2) Support for implementation specific size of physical address spaces up to 35 address bits (32GByte)
3) Three different page sizes:
    - Level 0 pages (32GB (only with 64-bit EA)) translated with (Data/Instruction) D/I Area Translation Buffer (ATB)
    - Level 1 pages (16MB) translated with D/I Area Translation Buffer (ATB)
    - Level 2 pages (8KB) translated with D/I Translation Lookaside Buffer (TLB)
4) Address translation using one, two or three-level page tables
5) Powerful page based access protection with support for demand-paged virtual memory

6) Support for simultaneous multi-threading (SMT)

The MMU, together with the exception processing mechanism, provides the necessary support for the operating system to implement a paged virtual memory environment and for enforcing protection of designated memory areas.

Programmable Interrupt Controller

OR1K architecture has special support for fast exception processing also called fast context switch support. This allows very rapid interrupt processing. It is achieved with shadowing general-purpose and some special registers. Exceptions can occur while an exception handler routine is executing and multiple exceptions can become nested. Support for fast exceptions allows fast nesting of exceptions until all shadowed registers are used. All exceptions can be described as precise or imprecise and either synchronous or asynchronous. Synchronous exceptions are caused by instructions and asynchronous exceptions are caused by events external to the processor.

OR1K has 14 exceptions including; reset interrupt and 3 reserved exceptions. The Programmable Interrupt Controller has two special-purpose registers and 32 maskable interrupt inputs. If implementation requires permanent unmasked interrupt inputs, it can use interrupt inputs [1:0]. The interrupt controller mask register (PICMR) is a 32-bit special-purpose supervisor- level register accessible in supervisor mode.

Power Management

The OR1K architecture defines five architectural features for minimizing power consumption:

1) Slow down feature

2) Doze mode

3) Sleep mode

4) Suspend mode

5) Dynamic clock gating feature

The Slow down feature takes advantage of the low-power dividers in external clock generation circuitry to enable full functionality, but at a lower frequency so that power consumption is reduced. The slow down feature is software controlled. When software initiates the Doze mode, software processing on the core suspends. The clocks to the processor internal units are disabled except to the internal tick timer and programmable interrupt controller. However other on-chip blocks (outside of the processor block) can continue to function as normal.

In Sleep mode and Suspend mode, all processor internal units are disabled and clocks gated. Optionally, an implementation may choose to lower the operating voltage of the processor core. The processor should leave sleep mode and enter normal mode when a pending interrupt occurs.

If enabled, the Dynamic clock-gating feature automatically disables clock sub-trees to major processor internal units on a clock cycle basis. These blocks are usually the CPU, FPU/VU, IC, DC, IMMU and DMMU. This feature can be used in a combination with other power management features and low-power modes. The processor enters normal mode when it is reset.

Support for Custom Number of GPRs

Programs may be compiled with less than thirty-two registers. Unused registers are disabled (set as fixed registers) when compiling code. Such code is also executable on normal implementations with thirty-two registers but not vice versa. This feature is quite useful since users are expected to move from less powerful OpenRISC implementations with less than thirty-two registers to more powerful thirty-two register OpenRISC implementations.

The instruction set of OR1K is split into two instruction classes according to implementation importance, namely Class-I and Class-II. Class-I instruction set must always be implemented, where as instructions from Class-II are optional and an implementation may choose to use some or all instructions from this class based on requirements of the target application.

Wishbone

OR1K processor is connected to Instruction Memory (IMEM) through an open source hardware processor bus called Wishbone Bus. The Wishbone architecture is analogous to a microcomputer bus in that that they both: (a) offer a flexible integration solution that can be easily tailored to a specific application; (b) offer a variety of bus cycles and data path widths to solve various system

Problems [31]. Features of this technology include:

1) Simple, compact, logical IP core hardware interfaces that require very few logic gates.
2) Full set of popular data transfer bus protocols including:
    - READ/WRITE cycle

- BLOCK transfer cycle
- RMW cycle

3) Modular data bus widths and operand sizes up to 64-bits.
4) Supports both BIG ENDIAN and LITTLE ENDIAN data ordering.
5) Supports single clock data transfers.
6) Supports normal cycle termination, retry termination and termination due to error.
7) Modular address widths.
8) User-defined tags. These are useful for applying information to an address bus, a data bus or a bus cycle. They are especially helpful when modifying a bus cycle to identify information such as:
    - Data transfers
    - Parity or error correction bits
    - Interrupt vectors
    - Cache control operations
9) MASTER / SLAVE architecture for very flexible system designs.
10) Multiprocessing (multi-MASTER) capabilities. This allows for a wide variety of System-on-Chip configurations.

The Figure 8 shows the basic signals of Wishbone when connected as a Master and as a Slave. Brief description of each of those signals is given below.

- RST_I: The reset input RST_I forces the WISHBONE interface to restart. Furthermore, all internal self-starting state machines will be forced into an initial

state. This signal only resets the WISHBONE interface. It is not required to reset other parts of an IP core.

- CLK_I: The clock input CLK_I coordinates all activities for the internal logic within the WISHBONE interconnect. All WISHBONE output signals are registered at the rising edge of CLK_I. All WISHBONE input signals are stable before the rising edge of CLK_I.
- ADR_*: The address bus is used to pass a binary address.
- DAT_* : Data bus width used in the design is 32 bits.
- WE_*: The write enable output indicates whether the current local bus cycle is a READ or WRITE cycle. The signal is negated during READ cycles, and is asserted during WRITE cycles.

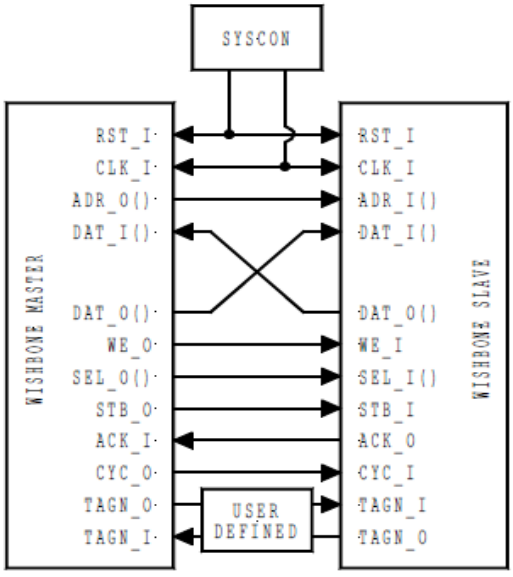


Figure 8. Wishbone Master and Slave Connections [31]

- STB_*: The strobe output indicates a valid data transfer cycle. The SLAVE asserts either the ACK_I, ERR_I or RTY_I signals in response to every assertion of the STB_O signal.
- ACK_*: The acknowledge input, when asserted, indicates the normal termination of a bus cycle.
- CYC_*:The cycle output, when asserted, indicates that a valid bus cycle is in progress. The signal is asserted for the duration of all bus cycles. For example, during a BLOCK transfer cycle there can be multiple data transfers. The CYC_O signal is asserted during the first data transfer, and remains asserted until the last data transfer.

The most frequently used data transfer in the design is READ and WRITE transfers. Figure 9 illustrates a read cycle, initiated by the master PE.

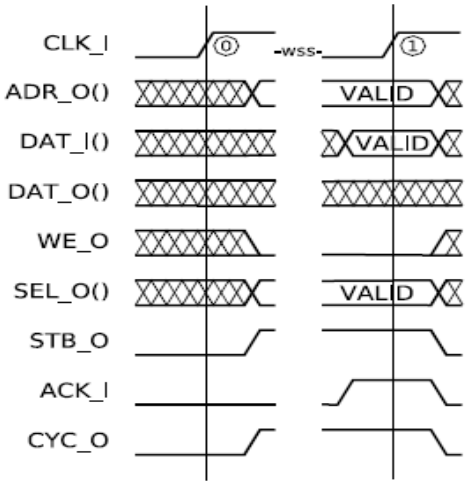


Figure 9. Single Wishbone Read Cycle [31]

A valid address is sent on ADR_O bus, setting signals CYC_O and STB_O to its active state and WE_O inactive, indicating a read. When the slave has data ready it responds by setting ACK_I, presenting the valid data on DAT_O. In the next clock cycle the master sets STB_O and CYC_O inactive, as does the slave with the ACK_O signal. Figure 10 illustrates the write cycle.

The write cycle handshaking protocol is initiated by the master, by setting STB_O and CYC_O active. Signal WE_O is set active to represent a write cycle with valid data on DAT_O and an address on ADR_O. The slave stores the data, and sends back acknowledge signal as ACK_I in the following clock cycle. Master then deactivates both STB_O and CYC_O and ACK_O is deactivated by slave.

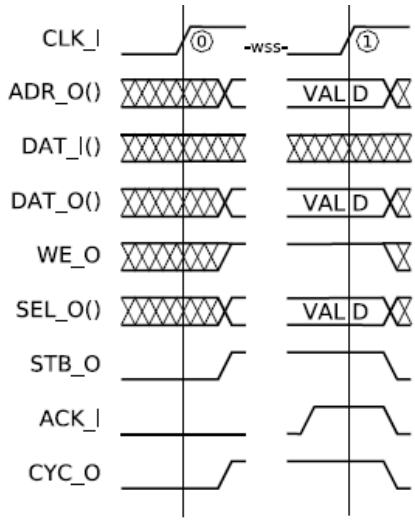


Figure 10. Single Wishbone Write cycle [31]

3.1.2. Network Adapters

Network Adapters (NA) implement the interface by which PEs connect to the NoC. Its main function is to generate and process packets. NA component implements a Core Interface (CI) at the core side and a Slave network interface (NI) at the slave side.

Core Interface

When the master PE wishes to make a request to the slave, the CI wraps the request into a packet containing the necessary data and the route to the slave and sends the packet. Hence, CI functions as a slave for the master PE. Figure 11, shows the implementation details of Core Interface Module.

CI is implemented as a Moore Machine which is triggered by the control signals generated from the PE and generates the packet. In the Store and Forward switching, every CI consists of a Routing Table (RT). RT is basically a lookup table which contains minimal routing path between every source node to every possible destination node in the NoC.

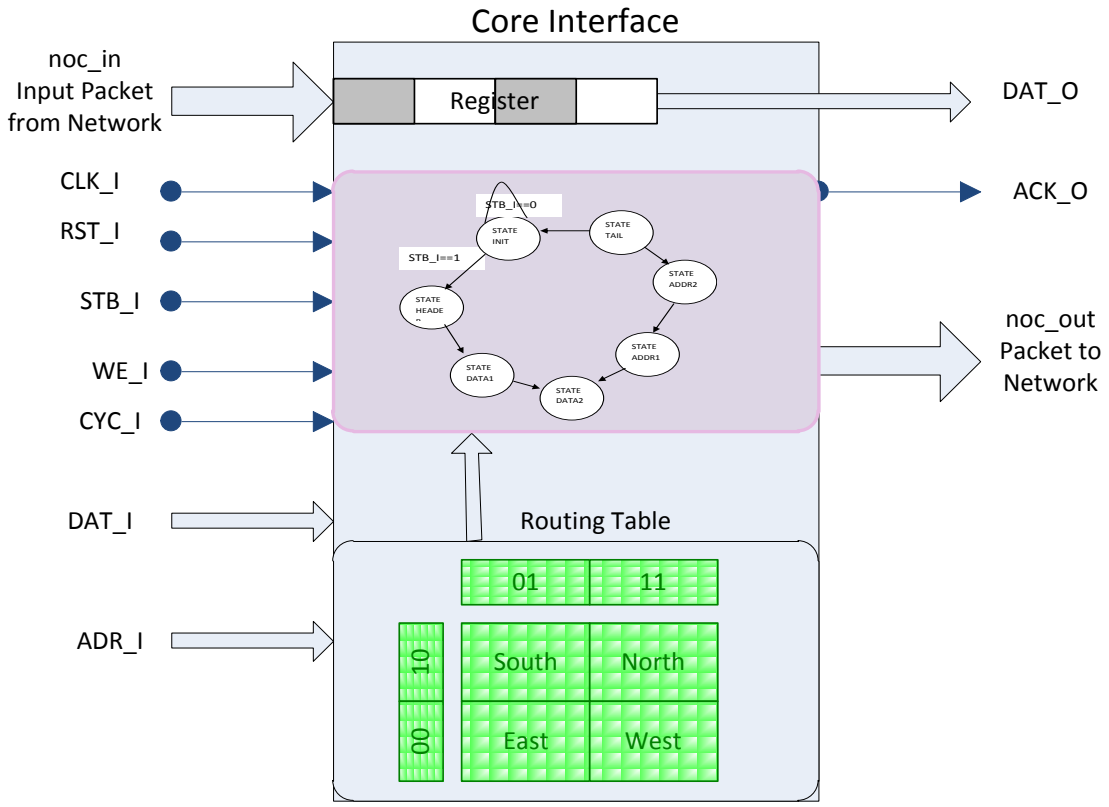


Figure 11. Core Interface Module

When the Master PE initiates a request/transfer, the CI FSM is triggered. In SF switching, CI reads in the destination address (ADR_I), looks up the RT and maps the address into routes to the intended destinations (i.e. host intelligent routing). This information is stored in the packet. In WormHole switching, CI does not use RT but uses the destination address and determines the route according to the routing algorithm.

The size and type of packet can be changed by modifying certain parameters in the top Verilog file. The entire message can be either generated as a single packet or the packets can be divided into flits before it is actually transmitted. The size and contents of the

packet can also be parameterized by modifying the corresponding parameters in a single configuration file. The packet format is shown in Figure 12.

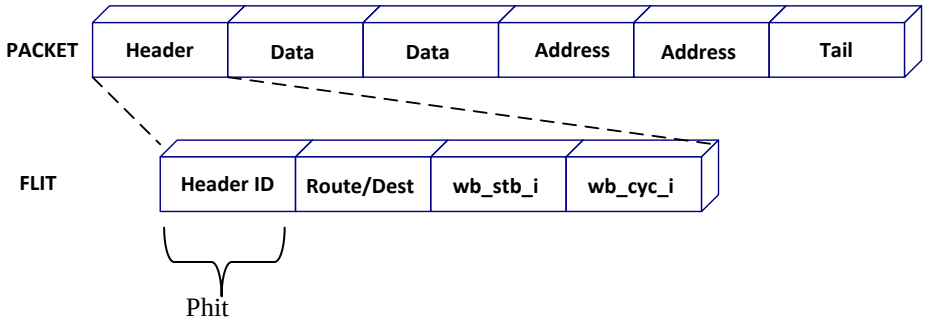


Figure 12. Packet

Each Flit, consists of 2 bit packet ID, to determine the type of packet.

- 00 –Header
- 01-Data
- 10-Address
- 11-Tail.

The data and address flits use three bits to determine the order or packet. Every packet has 2 bits of wishbone control signals namely STB_I and CYC_I. Each data or address packet includes 16 bits of data or address information.

In Store and Forward switching, the packet header contains the entire routing information in the Route/Dest phit. To reduce the length of packet, we use two bits to represent each direction; 00 for north, 01 for South; 10 for East and 11 for West. IP or end of route is represented by last two bits in the route in opposite direction. For example

a packet in the direction {East, East, North, North} will have the route as {10,10,00,00,01}. The last two bits represents the end of route will be in opposite direction to the final route. In Wormhole routing, Route/Dest phit will just have the destination address.

The tail bit contains four parity bits, one for each data and address flits. A parity bit is a bit that is added to ensure that the number of bits with the value one in a set of bits is even or odd. Parity bits are used as the simplest form of error detecting code. Even parity is used where, the parity bit is set to 1 if the number of ones in the corresponding data bits is odd. Even parity is a special case of a cyclic redundancy check (CRC), where the 1-bit CRC is generated by the polynomial x+1.

The State Machine used in CI module is shown in Figure 13. Each state represents the generation of single FLIT. CI has 2 modes of operation, ACK mode and non-ACK mode. In ACK mode, the CI after sending packet to the network waits for slave to send back a packet with acknowledge signal. This confirms the reception of packet. When the CI receives the packet whose ACK_I is active, it sends back ACK signal to Core PE indicating that the transfer is complete. This process even though is quite useful for safe data transmission, keeps the CI waiting until acknowledge is received. This can be avoided by disabling the ACK Mode.

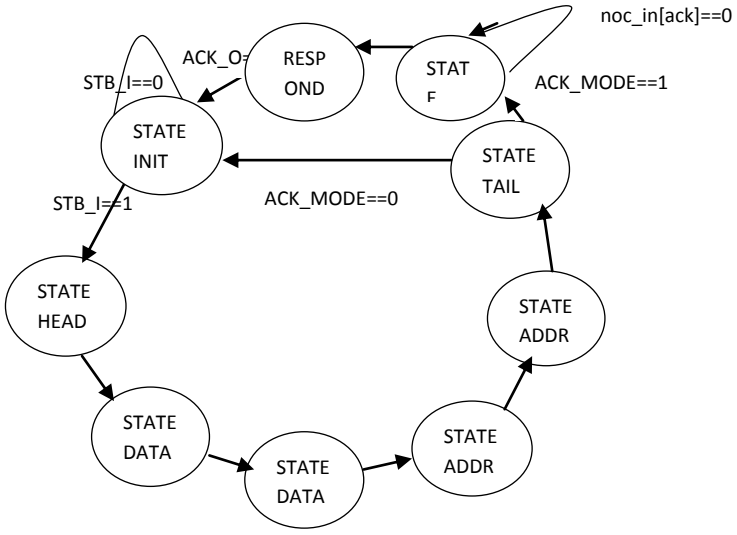


Figure 13. CI Flow Diagram

When CI acts as a receiver/destination node, it reads in the incoming packet (noc_in) to an input register (Register, Ref Figure 11), unpack the message and sends the data information (DAT_O) to the PE.

The slave Network Interface

The slave Network Adapter/Interface (NI) and CI are fundamentally the same but they differ in some aspects. The NI waits for the network, where the master network adapter waits on the IP core.

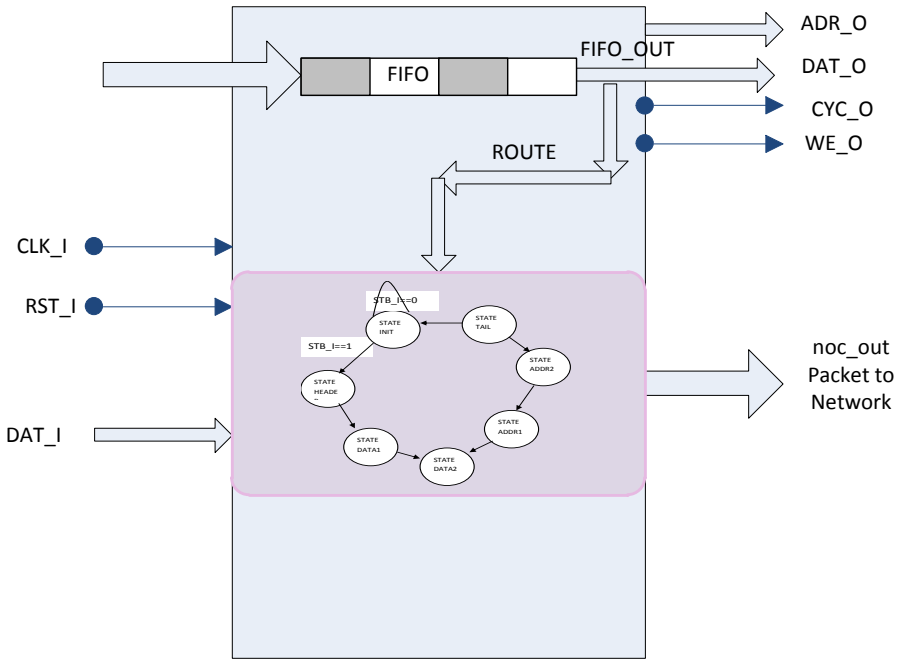


Figure 14. Slave Network Interface (NI)

The implementation details of Slave Network Interface module is shown in Figure 12. It consists of a FIFO to store the incoming packets and a FSM that generates control signals. It sends out data and control signals to Slave PE after unpacking the received packets (FIFO_OUT). When the Slave PE has to respond to the Master request, the FSM packs the message and sends it to interface.

In SF switching, as the packet travels from node to node through the network the route is updated with a return route. Hence the slave network adapter does not have to find the route back, and can follow same route followed by the incoming packet from Master. The route however is in the reverse order, so the order of the route has to be corrected. This further reduces the complexity of slave network adapter. In other words a slave network

adapter does not have any routing table. In Wormhole routing, the NI and CI has the same function. They just include the destination address to the header and the routing is determined by the Routing nodes.

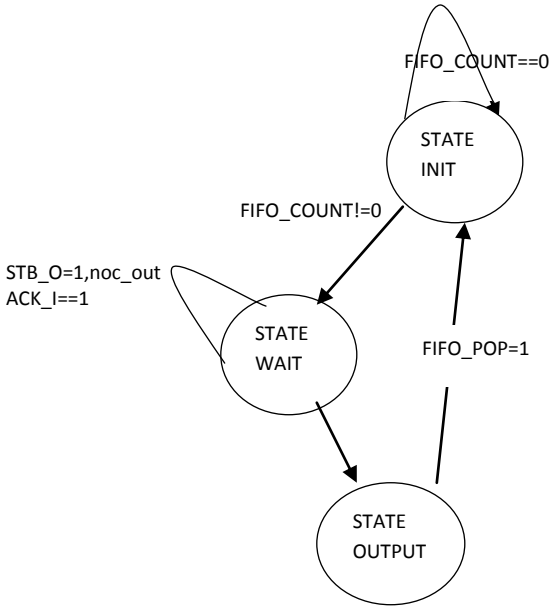


Figure 15. NI State Machine

The State Machine used in the implementation of NI is shown in Figure 15. The FSM is initiated when a packet is available in FIFO. It then enables strobe signals indicating a read to the address specified in the incoming packet. It then determines the route/address from the incoming packet. In ACK Mode, NI would wait until it receives a ACK_* signal from Master, indicating completion of transfer, whereas in normal mode, NI continues the FSM with the next packet in FIFO.

3.2 Communication Infrastructure

The Communication Infrastructure module transfers the message from Source Interface module to Destination Interface module. It consists of 2 components 1) Network Topology and 2) Routing Node. The way the Routing nodes are connected defines the Network Topology. Each Routing Node consists of 3 main components 1) Link Controller 2) Channels/Links 3) Router Arbiter. Following sections discusses each of these modules in detail.

3.3.1 Network Topology

The way the Routing Nodes are put together is called the Network Topology. The design can choose any topology from Torus, Mesh or WK-recursive topology. Figure 16 shows the Torus Topology. We concentrate on 3x3 torus and WK(4,2) topology for evaluation purpose.

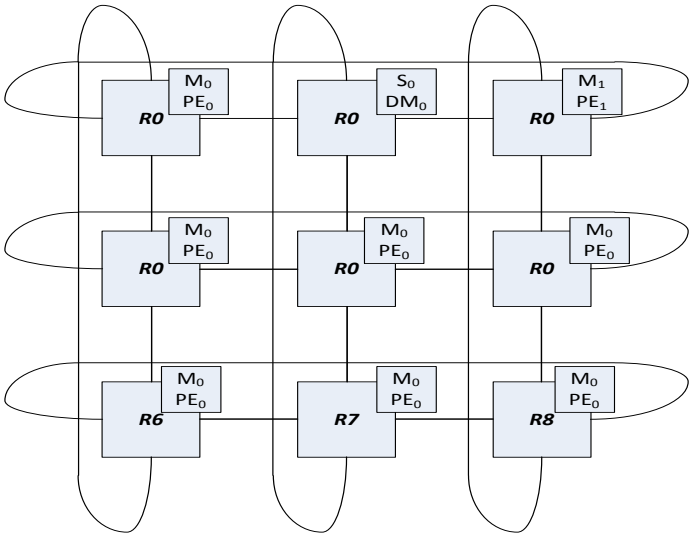


Figure 16. Torus Topology

3.3.2 Routing Node

Routing Nodes connects each PE in the desired topology. Routing logic consists of three components i) Link Controller ii) Channel/ Links and iii) Router Arbiter. Routing Nodes route the message/packets/flits according to chosen protocols. They implement the routing strategy. Each Router in Torus and Mesh topology has 5 input and 5 output ports. Four input port from the four cardinal directions (North, East, South and West) and one (ip_*) from the local Processing Element (PE). In WK topology, router nodes have 3 to 4 input/output ports depending on the position of router in network. The ports are named as North (noc_n*), South (noc_s*), Across (noc_a*), Over (noc_o*), East (noc_e*) and West (noc_w*). The ports to South-West or South-East are called Across port and those to North-West and North-East directions are named as Over port.

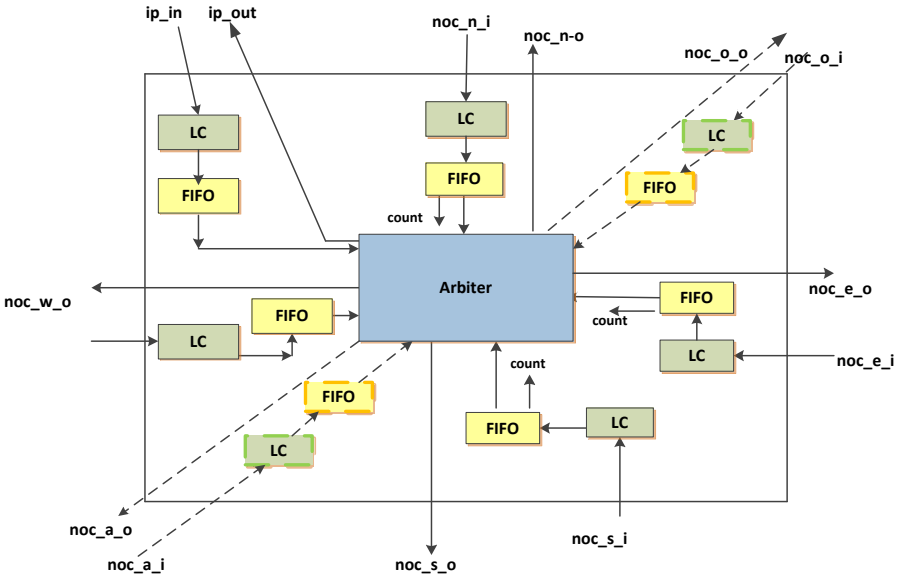


Figure 17. A Routing Node

Figure 17 shows the implementation details of a Routing Node. The dotted lines represent ports and logic for WK topology alone.

Link Controller

The design consists of a Link Controller (LC) module at every input port of the Routing node. LC provides an interface for the CI to the Topology at the IP Port. Most of the PEs will be working at a much smaller frequency compared to the clock frequency of Topology. The main function of LC is to match CI clock rate to that of the Topology. Synchronization registers are used to match clock rates between the slow PE and fast Topology.

For all other ports other than IP, the main job of LC inside each Routing Node is to verify and store the incoming packet to the corresponding Input Buffer. It is the job of LC to detect any packets in the physical link and push it to the input FIFO buffer as soon as it is available. LC does this by polling the link to check if there is a packet on the link. If either the strobe (STB_*) signal or acknowledge (ACK_*) signal (in ACK Mode) of the packet is active, it directs the packet to the Channel Buffer. The function of LC remains the same for all switching methods and all routing algorithm.

Channel/Link

Every link in the network is full duplex, i.e., two messages can simultaneously travel on the link in opposite directions. A link is available for communication if its associated channel is available to accept the packet. A READY and SEND signal is used to communicate between adjacent nodes. Whenever the channel is busy it informs this to every adjacent nodes by setting the READY signal low.

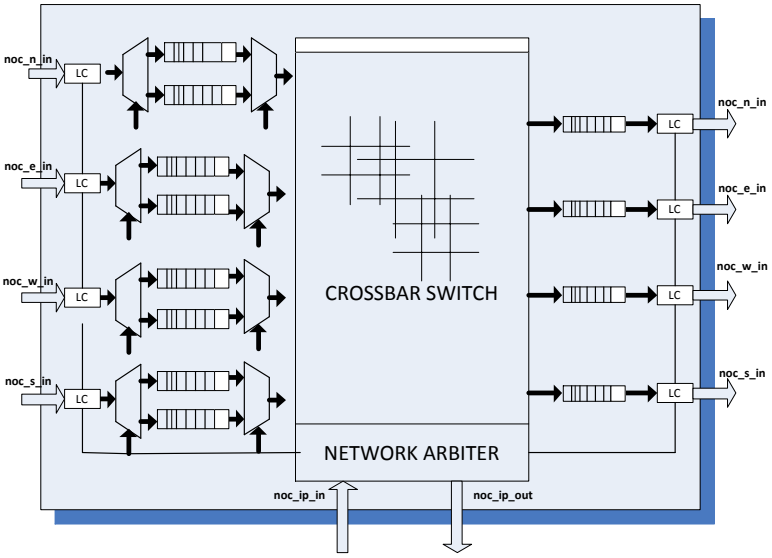


Figure 18. Network Router and Channels

Figure 18 shows the Network Router module with Virtual Channels and Link Controller. Every port has input buffering which means that the router stores the packet into a buffer as soon as it is available at its input port. Input buffering normally has a lower complexity and consequently a lower cost of implementation. Also, the switch fabric and the memory at the inputs of an input-queued switch need only run as fast as the line rate, whereas output buffering has to run 4 or 5 times as fast as the line rate. The width of each buffer can be modified to be equal to the packet length or flit length. The depth of each buffer can also be modified. In SF Switching the depth of buffer is 4 in packet routing and is equal to or more than 6 when the message is transmitted as flits. For Wormhole routing the normal buffer depth is 4.

To reduce congestion and avoid head on blocking in Wormhole routing, every physical channel is divided to Virtual Channels (VC). For WH routing, every input port

except the IP port has 4 virtual channels where as for SF switching every port has single FIFO. The IP port has a single FIFO in both cases.

The implementation of VC module is shown in Figure 19. The module consists of three main modules i) VC Identifier, ii) VC Buffers and iii) Switch Allocator. Each new incoming packet/flit is stored in the VC buffer determined by the VC identifier. VC identifier checks the buffer occupancy and directs the incoming packet/flit to the least occupied buffer. Access to a physical channel is now allocated on a cycle-by-cycle basis by the Switch Allocator module amongst waiting flits from any of the buffered packets which have been assigned a VC. The next packet to be sent to Arbiter is chosen in a daisy chain manner.

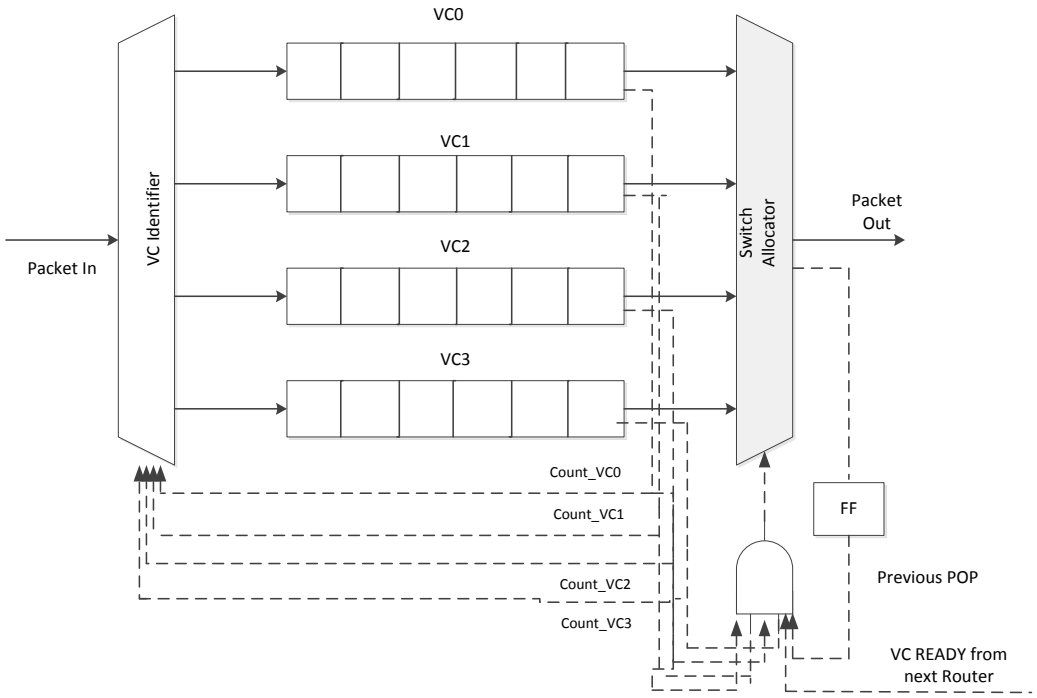


Figure 19. Implementation of Virtual Channel Module

Router Arbiter

The Router Arbiter is responsible for directing the incoming packet to the appropriate output Physical Channel/port (PC). Router Arbiter implements the Routing Algorithm.

In SF switching, the packet header contains the entire route from source to destination. This route corresponds to the minimal routing path loaded in Routing Table at Core Interface. For each packet stored in the Input buffer the Router Arbiter reads in the route, and determines the next route. It checks the READY signal from the next routing node and sends out the packet. When the READY signal is low, it means a congestion has occurred and the packet stays in the FIFO or find an alternative path according to the routing algorithm. The Arbiter can handle packets from all the five nodes simultaneously. Logic are added to Arbiter module to made it generic, even when the length of route is modified through parameters at the Top Level.

In WH Switching when the packet is send as flits, the arbiter reads in the header Route/Dest phit and determines the next route based on the routing algorithm. It also generates a CONNECT signal that establishes a circuit between the source and destination along the path made by header phit. The Arbiter disables this signal after routing the TAIL phit.

3.4 Communication Paradigm

Communication Paradigm explains the Routing policy and Switching Techniques used. The Framework can choose either i) Store and Forward (SF) and ii) Wormhole (WH) Switching for message forwarding. In SF switching, the message can be send either as packets or in form of flits. Each packet is 80 bits long and flit is 25bits long.

When message is transmitted as flits, each Routing node will wait until the entire message is reached before processing the HEADER. The end of message is determined by TAIL flit. In Wormhole routing, the message is transmitted as soon as the HEADER is available. The path is determined and is connected by the HEADER as it moves through the Network. The remaining flits follow the same path. The path is disconnected as a TAIL flit is detected.

For Torus and Mesh topology the design uses XY Algorithm for Routing. XY Routing algorithm is DeadLock free as it restricts few turns in Y to X directions. The design uses a new Minimal Routing Scheme for WK-recursive Topology, which is explained in detail in Chapter 6.

3.5. Traffic Generator

Traffic Generator (TG) module controls the traffic in the network. It can initiate a request or a start of transmission from top level. TG also determines the type of traffic along with the source and destination nodes. Different congestion scenarios and node failures can also be created through TG.

When the TG initiates a message transfer, it enables the initiation signal (send_*) to the corresponding PE, along with the destination node address and 32bit data to be send. The PE as soon as it receives the send_* signal, it overrides any access from processor and sends the data, address, wb_stb_* and wb_cyc_* signals to Core Network Interface.

The TG module generates three different kinds of traffic such as uniform, Sporadic and hot spot. In Uniform Traffic, packets are sent from a source node to a destination node

TG can also control the mode of channels in the sense it can generate a congested link or a faulty node scenario. TG does this by forcing the signals *_failed or by controlling the channel ready signals. To generate a scenario where in the slave, S1 is failed, it forces the signal s1_failed to 1 and to develop congestion in channel going east direction of node r1, TG forces the signal r1_east_ready to 0. This overrides the current situation and creates the indented scenario. This feature of TG is very helpful in evaluating the efficiency of routing algorithms. Figure 20 explains the scenario.

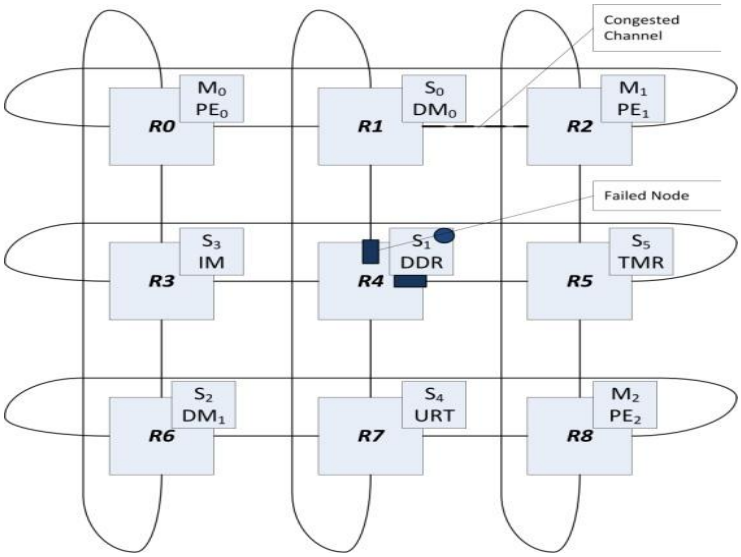


Figure 20. Torus structure with a Congested link and failed Node

3.6. Monitor

The design consists of a Transaction monitor module at each router and a monitor module – NoC Monitor at Network level (see Figure 3). Transaction monitor at each Router contains information about the buffer count in each VCs and sends this

information to top NoC monitor and traffic controller. It also keeps track of PE status. In ACK Mode the monitor waits for a definite amount of time until it receives back acknowledge packet from the destination node, and declares the node as faulty after that. Monitor also checks the READY signal which indicates the status of channel/link. When the FIFO is three of four percent full, it sends out a signal *_fifo_almost_full indicating near possibility of congestion in the route. NoC Monitor at top level keeps track of packet transactions and generate a verbose file indicating the initiation of a packet, time at which it reaches each node and time of arrival of packet at destination.

CHAPTER 4

NOC FRAMEWORK FLOW

This Chapter explains the control and data flow sequences followed in the transfer of messages between PEs. It first explains the sequence of operations followed when two PEs communicate with each other. Chapter also clearly explains how the underlying architecture modifies itself for different switching mechanism, Routing algorithms and modes of Operation (with and without Handshaking).

A packet flow can be initiated either by Traffic Generator or by the master PE. TG can be used to override/mask network conditions or signals from PE.

There are mainly three types of message transfers in a network. i) Master PE sending a message/data to slave PE ii) Master PE requesting data from slave PE and iii) Master to Master communication. Figure 21 shows the Framework and Routing Node with its associated components.

4.1 Master PE sending a message/data to slave PE

When the master PE wishes to send data to a slave PE, it send data, address and strobe signals to the Core Network Interface. As soon as CI receives a request (a high in strobe signal) it looks up the routing table compares the source and destination address and determines the route. In case of SF switching, CI generates the packet with the entire ‘route to follow’ in the header field along with data and control signals; whereas in WH routing, the packet header will only carry the destination address and the control signals.

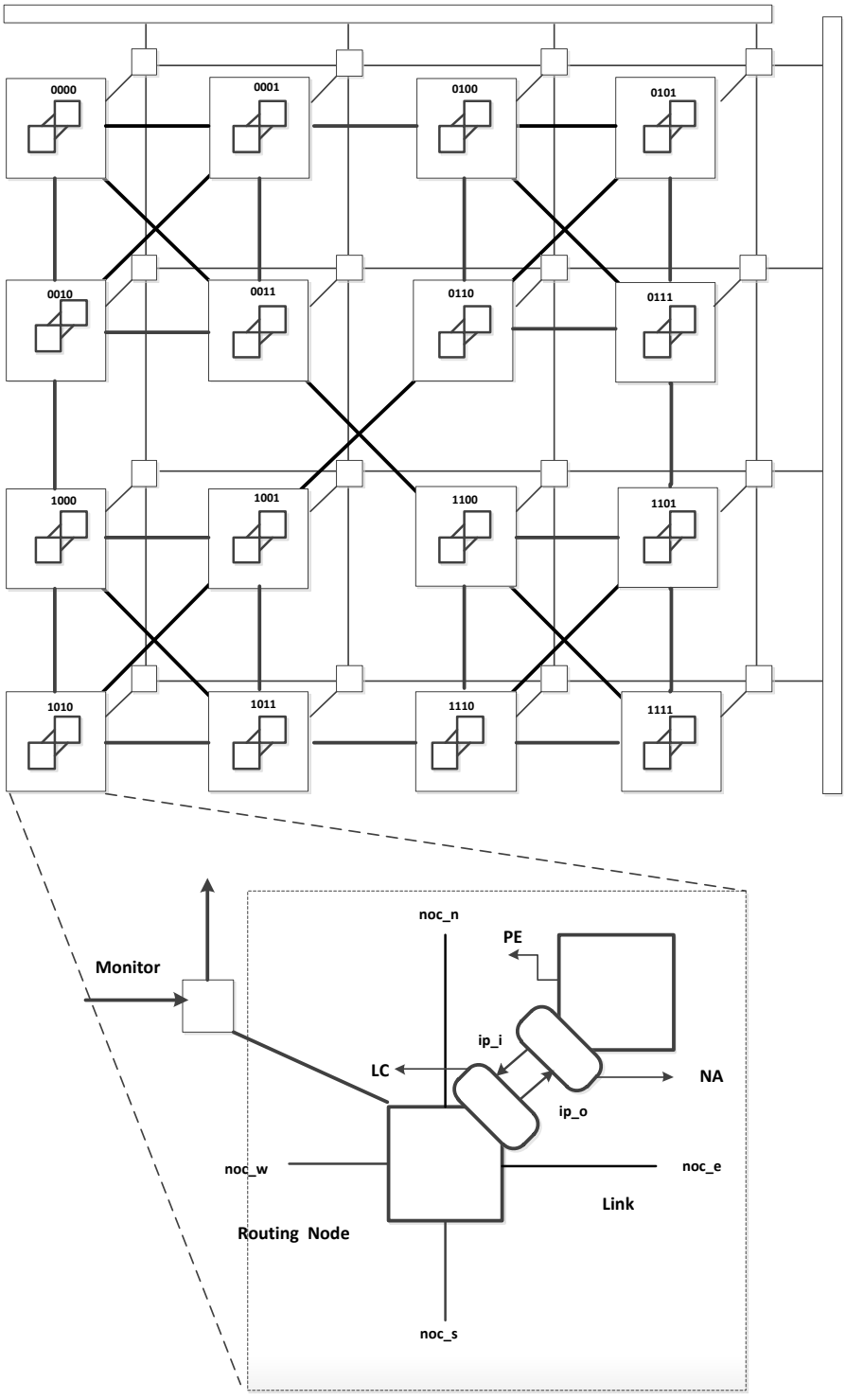


Figure 21. NoC Framework and Routing Node

The packet is then send to Routing Node through the IP input. The Link Controller (LC) polls the IP port of router and whenever it detects a valid packet it pushes the packet into the corresponding input buffer.

Router Arbiter module determines the next routing path for each packet at each input port and directs the packet to the corresponding output port. In the case of SF switching, arbiter waits until the entire packet is received at the input FIFO and then reads in the header flit to determine the next route. In WH switching, as soon as the header flit is received the arbiter reads in the header and determines the destination router address, from which it calculates the next route through an algorithm or LUT present in the arbiter. If the next router input buffer is full, then the packet waits until it is available. A READY signal from the next router is used to determine if the packet can be sent or not. Each router has 'READY' signals in all the port directions. READY signal is enabled or disabled based on the Input buffer occupancy. The packet hence moves forward through the routers until it reaches its destination node, where it gets directed to the IP port of router by router arbiter. The Link Controller module uses a FIFO to store the packets from faster network and sends it to the slower PE through the Network Interface module. This is needed for synchronizing two clock frequencies.

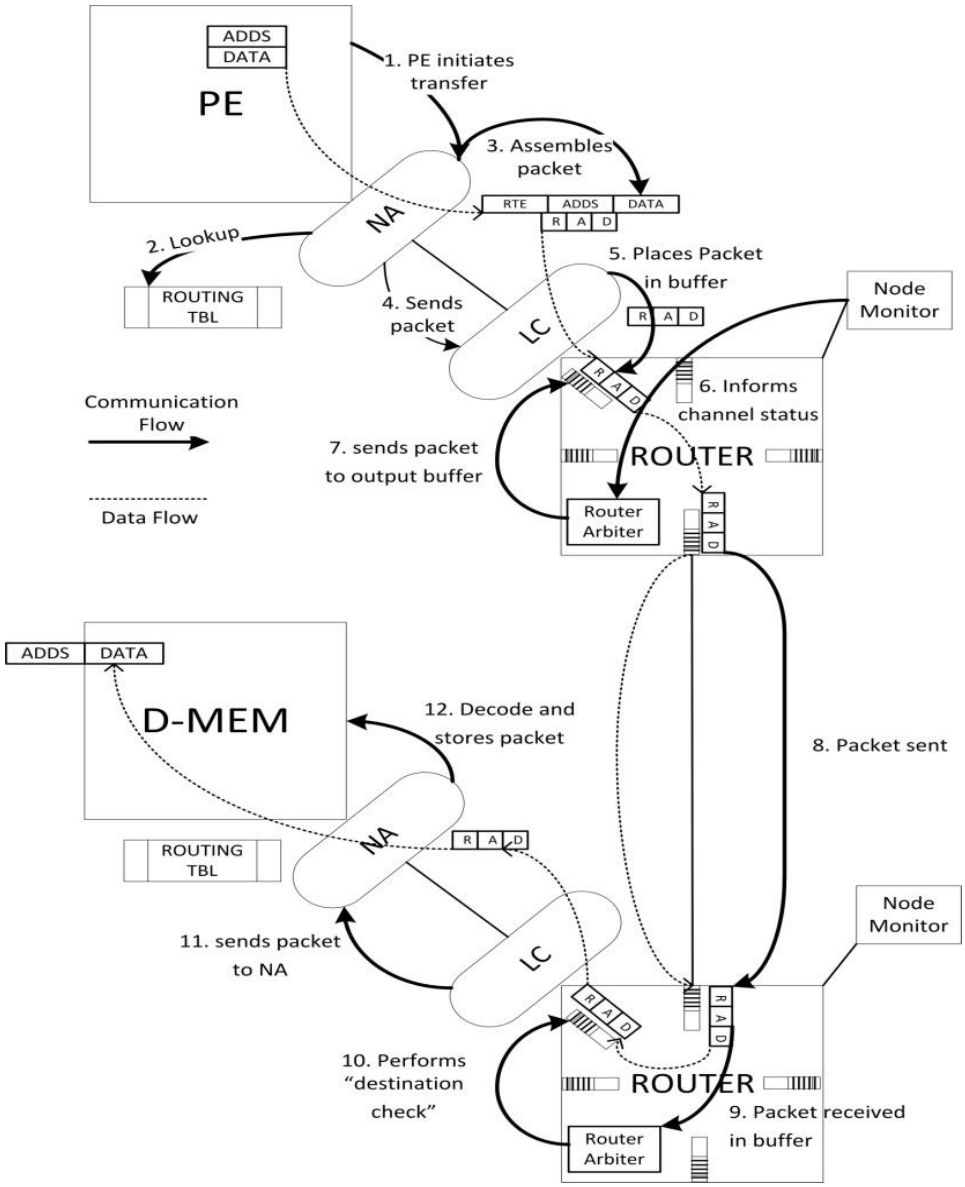


Figure 22. NoC Communication Flow

The design can be chose to work with or without a handshaking mode. In normal operation the master PE sending the packet will wait until an acknowledge signal is received from slave PE to which the packet is send. The slave NA does not have a routing table. It just looks at the arriving packet (from the master PE) and determines the router

by reversing the packet header. This helps reducing the area overhead caused by routing table and the logic needed to determine the route at every slave node. When the no acknowledge mode is selected, the master PE can send packets at successive clocks without waiting for the acknowledge signal. This mode is chosen to create different traffic scenarios during evaluation phase. Refer Figure 22.

4.2 Master PE Requesting Data from Slave PE

When a master requests a data from slave PE, It enables the strobe signals and the address signal from which it requires the data. The packet generated by CI is sent to slave PE through the slave PE in the same manner discussed above. The slave PE upon receiving the requested packet, reads the packet and determines the route from the header. The slave NI generates a new packet with the route and data requested and sends it back to the master node.

4.3 Master to Master Communication

Master to Master communication is same as master requesting the data, but instead of reversing the header, the route is determined from the routing table.

CHAPTER 5

OPTIMIZATIONS AND MODIFICATIONS

This Chapter elaborates the modifications considered during the design of NoC Framework. The major optimizations considered are i) Modification in Virtual Channel architecture for SF routing to improve throughput, latency and area ii) A novel routing algorithm called “Mirror Routing” for Torus architectures to avoid congested or faulty links/nodes and finally iii) An adaptive routing algorithm for WK topology.

5.1. Modifications to SF Routing - VC AND LOOPBACK VC

This section provides an overview of the architecture of a generic Virtual Channel (VC) router, we implemented for Wormhole switching. It also detail the implementation of loopback VC, a novel architecture implemented in our NoC system for Store and Forward Switching.

5.1.1 Overview of a Virtual Channel Router

Wormhole routing is prone to congestion and deadlocks. In 1987, Dally and Seitz introduced the idea of Virtual Channel to develop deadlock free routing algorithms for networks that use wormhole routing [52]. In his paper [51] he has described a theoretical model for networks using virtual channel flow control and has presented the results of his simulation studies. These results indicate a dramatic increase in network throughput due to the use of Virtual Channels.

Consider a scenario where in the packet at the top of FIFO has to be routed to east, but the east port is blocked due to congestion. This is head on blocking. To reduce congestion and avoid head on blocking every physical channel is divided into Virtual Channels

(VC). Every input port except the IP port has 4 virtual channels. The IP port has a single FIFO. However too many VCs could cause a packet to spread across many routers, which increases its latency in the network [53].

When a new packet reaches an input port of a router, an attempt is made to allocate an unused VC for the new packet. In cases where all the VCs are occupied, the VC identifier module checks the buffer count of each VC and directs the incoming packet to the lowest occupied FIFO. When more than one VC in a input port is occupied, the Switch Identifier module determines the next packet to be routed in a round robin manner. The packet is then routed to the output port based on the routing signals obtained from NoC Arbiter.

In our design, once a portion of a packet occupies a VC queue, other flits of same packet are not allowed to use only the same VC flit slots. In other words the VC Identifier looks for header and determines the VC. Once a VC is identified for a particular packet, the entire packet uses the same VC. This avoids wastage of VC resources if a packet gets blocked before reaching destination.

Assume there are six incoming packets P0, P1, P2, P3, P4, P5, P6 and P7 with designated output ports East, East, East, East, West, West and South respectively in current router. Consider the condition when East port is busy now. The Figure 23 shows the scenario where in P0, P1, P2, P3, occupying the head of VC0 to VC3 respectively cannot move forward hence blocking packets P5 and P6 to the West port. This is a head-on blocking scenario.

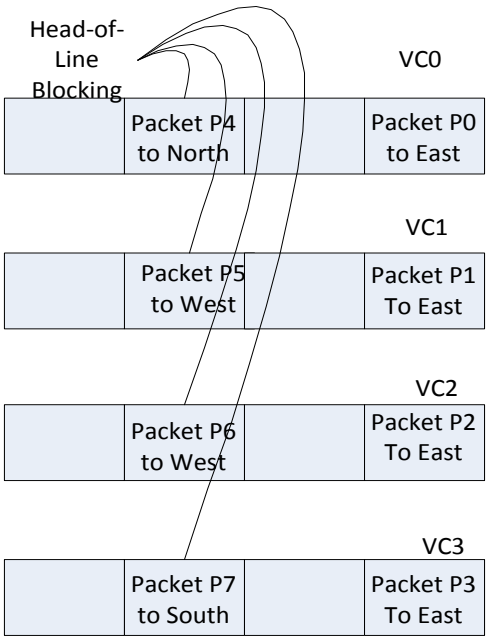


Figure 23. Head on Line Blocking

This can be avoided by allocating separate VCs for different ports.

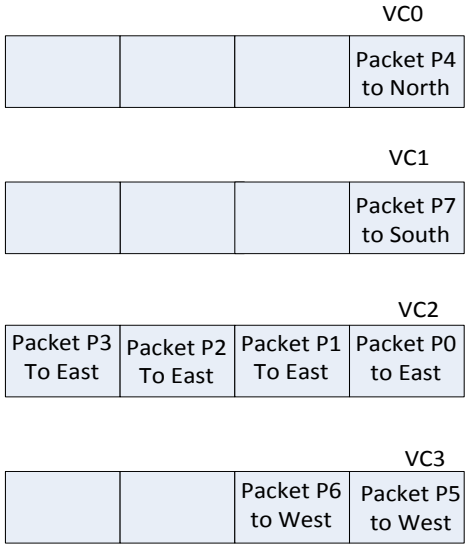


Figure 24. Packet allocation in modified Architecture

For example allocate all North going packets to VC0, South going packets to VC1, East going packets to VC2, and West going packets to VC3. The packet allocation is shown in Figure 24.

5.1.2 Virtual Channel with Loopback

The same issue can happen in SF routing where only 1 buffer is available at each input port. To avoid head-on-blocking in SF routing, we use loopbacked VCs. In presence of congestion, the packet from VC is routed back to the VC identifier logic to route it in a later point of time.

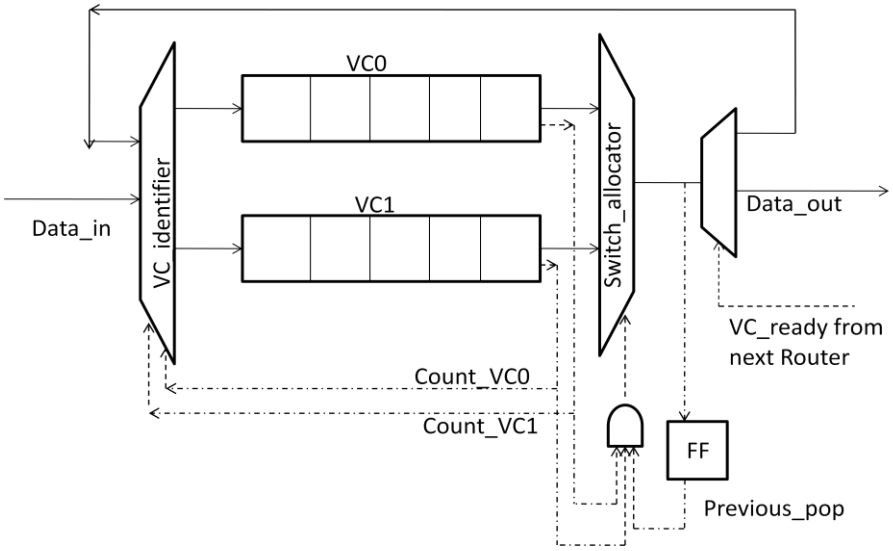


Figure 25. Loopback VC Top – Implementation details

The implementation consists of 2 Virtual Channels, VC0 and VC1 (See Figure 25). Whenever a packet reaches the VC identifier, it checks for the rate of occupancy in both Channels and allocate the packet to the least used. The switch allocator keeps track of

previous pop and sends the packet to arbiter in a round robin fashion. When the receiving channel is busy, indicating a congestion, then the packet routed from SA, is routed back to the VC identifier.

In cases where every outgoing channel is busy, the Switch Allocator will wait for VC_ready signal to toggle. When any of the channel becomes available, SA pops every available packet once. This is the worst case scenario. This helps in reducing power due to unnecessary loopback and also in routing packets whenever the resourse is available, increasing throughput.

5.2. Mirror Routing

This section explains a new adaptive routing algorithm for torus topology called Mirror routing. One of the main properties of torus structure is that Torus can exploit path diversity and hence many routing possibilities.

When the torus network is path diversified (or path multiplied, where several parallel lanes exist in each row or column), the address encapsulation mechanism can be used to take advantage of the path diversity while preserving the simplicity and obliviousness of dimension order routing. The encoding of the intermediate addresses can be done with the goal of balancing the load between parallel lanes, thus reducing the contention. [75]. This property facilitates adaptive routing, networks can be reconfigured around congested or faulty channels.

In SF Switching, if the adjacent link next to the source node is congested or failed, the NA in the source node chooses an alternative route, opposite in direction to the normal route. This is done by including an alternative route in Routing Table of every source

node for every possible failure or congestion. This results in increases in area of routing table a little but is not much compared to the improvement in the latency of received packets.

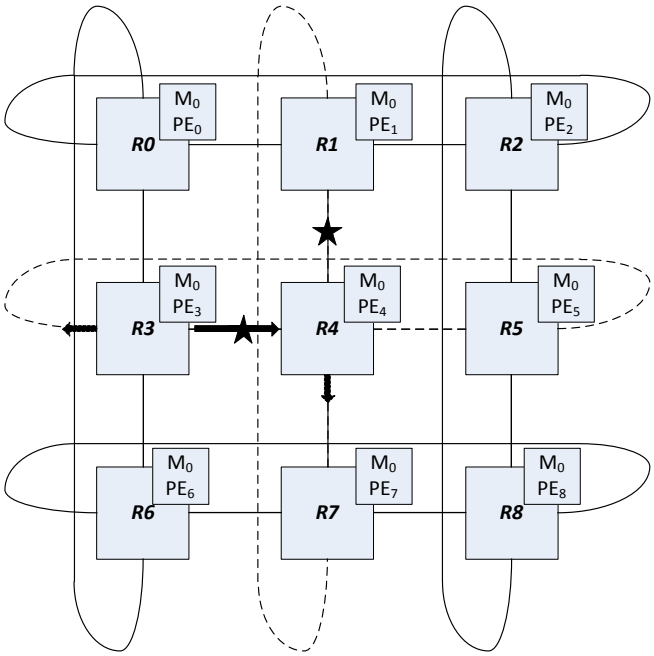


Figure 26. Mirror Routing

Consider the scenario where the PE, PE3 has to send packets to PE4 and link R3->R4 is congested. Mirror Routing will route the packet through R3 -> R5 -> R4, shown by dotted lines in Figure 26. Similarly if PE4 has to send packets to PE2 and link/channel R4-> R1 is congested, it will get redirected through R7.

3D Torus network also offers a high degree of path diversity and hence this algorithm is relevant in such topologies also.

5.3. Adaptive Routing for WK Topology

A WK-Recursive family of network topologies, denoted as $WK_{d,L}$, can be described by two parameters; the node degree d and the expansion level L.

In WK topology, each node is addressed using 4bits, as shown in Figure 27. In deterministic routing only the destination address is required to deliver a message.

In any $WK_{d,L}$ network, there are exactly t^{L-1} different $WK_{t,l}$ sub-networks that may be numbered 0 to t-1. Each WKt,l are connected by a complete graph and each link between two of them are referred to a flipping link. The flipping nodes are shown in dotted lines. Edge nodes shown by bold lines are called Extern nodes. All nodes except Extern nodes have 4 links/ports. Every Extern nodes have 3 ports.

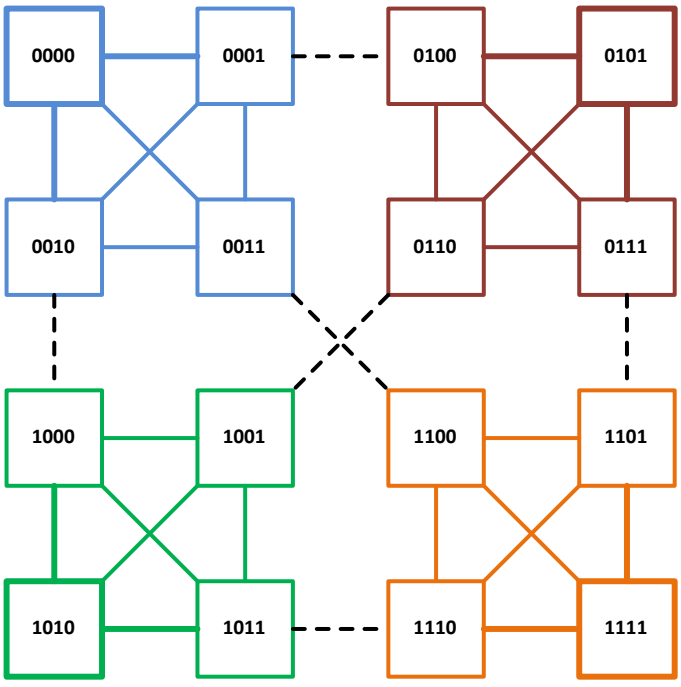


Figure 27. Node Addressing in WK Topology

The node degree is defined as the number of physical channels emanating from a node. The diameter is commonly used as a measure to compare the static network performance of a system. The diameter of a network is the maximum inter-node distance, i.e. the maximum number of links that must be traversed to send a message to the furthest node along a shortest path in the network. It gives a lower bound for the communication delay. The smaller the diameter of a network is, the less time it takes to send a message from one node to the node farthest away.

Table 2. Some Specifications of WK, Torus and Mesh Topology

Network	N.of Nodes	Node Degree	Diameter	N.of Links	Bisection Width
WK(d,L)	d^L	d	2^L-1	d *(d^L -1)/2	d
Torus	k^2	4	k-1	$2k^2$	2k
Mesh	k^2	2-4	2(k-1)	2k(k-1)	K

5.3.1 Minimal Routing Scheme

Every wormhole routing algorithm includes two important parts: i) Physical Channel Selection rule and ii) Virtual Channel Selection rule. Physical channel selection rule chooses the next physical channel to route the message and virtual channel selection rule chooses the proper virtual channel in selected physical channel by considering of deadlock avoidance conditions.

Physical Channel Selection Rule

Suppose S and T are the source node and destination node in $WK_{d,L}$, respectively. A routing path between them can be constructed as follows.

- Step 1. Compare the first two bits of the node addresses of S and T. If they are same, the destination node is in same sub-network. If they are different,
- Step 2. Determine the flipping edge. The flipping edge (X, Y) is the bridge between the two sub-networks.
- Step 3. Determine the routing path from S to X, and the routing path from X to T. The routing path from S to T is the concatenation of the routing path from S to X, the flipping edge (X, Y), and the routing path from Y to T.

Virtual channel selection rule

After selection of next physical channel according to the algorithm in previous section, suitable virtual channel must be selected in this physical channel by considering deadlock avoidance conditions. Each router node contains 4 channels at each output ports. Every packet arriving the port is routed to corresponding channels based on its next route. North/Over going packets to VC0, South/Across going packets to VC1, East going packets to VC2, and West going packets to VC3.

5.3.2 Adaptive Routing for WK-recursive

A new Adaptive Routing algorithm is introduced for WK routing, based on routing table in Table 3. The virtual channel selection rule for adaptive routing is same as that used in minimal routing. While selecting the physical channel, in step2, first determine the flipping edge (X, Y) between the two sub-networks. If the flipping edge is not

available or congested, choose the 2nd priority channel from the table. If that link is also unavailable choose 3rd priority link. For example, when a packet from a Over address has to reach a across destination address, it first choose the across path if available, and then chooses south link if not. If the south link is busy or not available, it then chooses the east or west link whichever is available at the point of time.

Table 3. Routing Table

Source	Destination/1st Priority	2nd Priority	3rd Priority
IP/Over/North/East/West	Across	South	East/West
South	Across	West	East
IP/Over/Across/North	South	East	Across
East/West	South	Across	West/East
IP/Over/Across/North	East	South	Across
South	East	Across	North

CHAPTER 6

RESULTS AND DISCUSSION

The NoC Flow starts with loading an application program in PE, evaluating the architecture for different configurations and algorithms and finally to observe the results. The Figure 28 shows the NoC Flow used in the design.

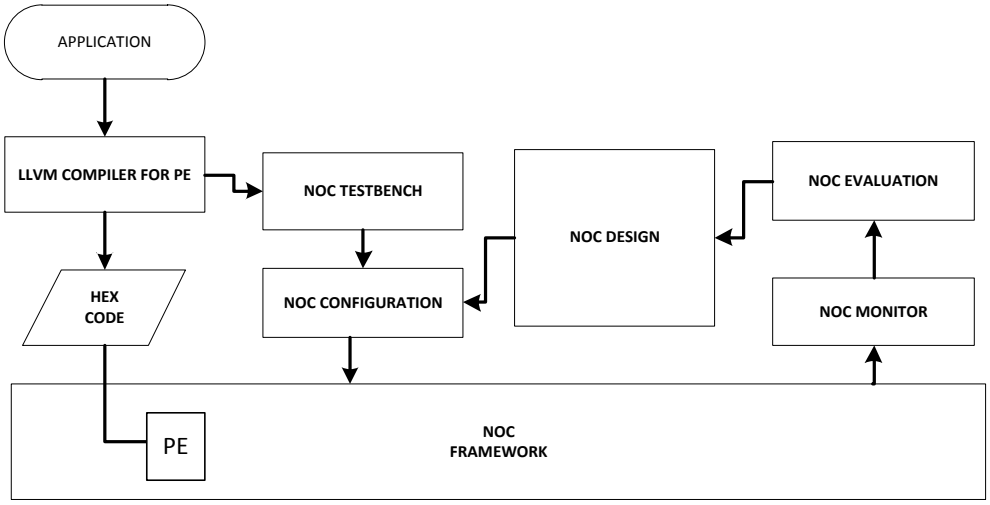


Figure 28. NoC Deisgn Flow

OR1k has a GNU toolchain including the GCC compiler and GNU debugger. The C Benchmark program is cross compiled using OR1K GCC and generated hexadecimal (opcode) code. The processor use on-chip RAM to execute a bootloader, in which the cache, stack and MMU are initiated and enabled. After initialization a jump is performed to the start of the program in external memory (IMEM).

A 32-bit special purpose supervisor-level register called Supervision Register (SR) defines the state of the processor. This register is read as soon as the processor restarts.

The 32bit unit present register (UPR) identifies the present units in the processor. It has a bit for each possible unit or functionality. The lower sixteen bits identify the presence of units defined in the OR1k architecture. The upper sixteen bits define the presence of custom units. CPU configuration register specifies the CPU capabilities and configuration.

.

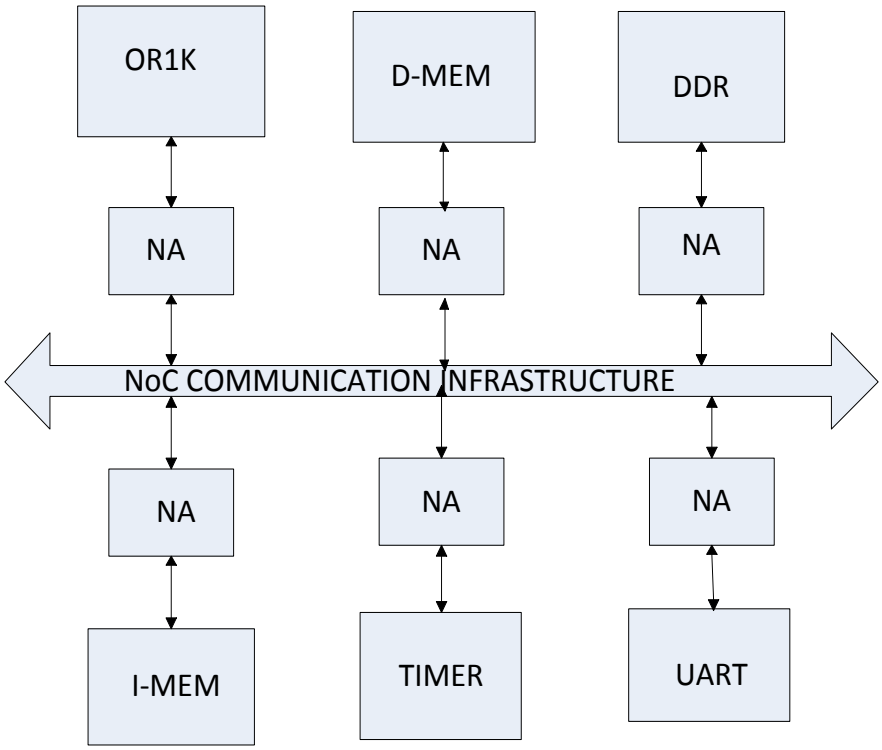


Figure 29. PEs connected to NoC Framework

The NoC components used for evaluation are shown in Figure 28. OR1k clock frequency is 250MHz. Wishbone clock frequency can be equal to core clock or half the frequency. The NoC network clock is 1GHz. Slave PEs include a TIMER, UART, and

normal Data Memory unit working at 125MHz and a DDR memory module running at twice the frequency (500MHz)

The hardware configuration is done by manually editing the verilog source code. Most of the configuration options can be changed by modifying the core configuration file.

6.1. SF vs WH

A detailed study was done on the latency for message transfers for WH and SF for various hops for uniform traffic. The design chose a buffer depth of four and four VCs at four ports of WH. The graph in Figure 30 proves that WH outperforms SF for uniform traffic.

Most of the NoC architectures are simulated with random traffic uniformly distributed over time and tiles. The average latency per packet is calculated and plotted versus the packet injection rate.

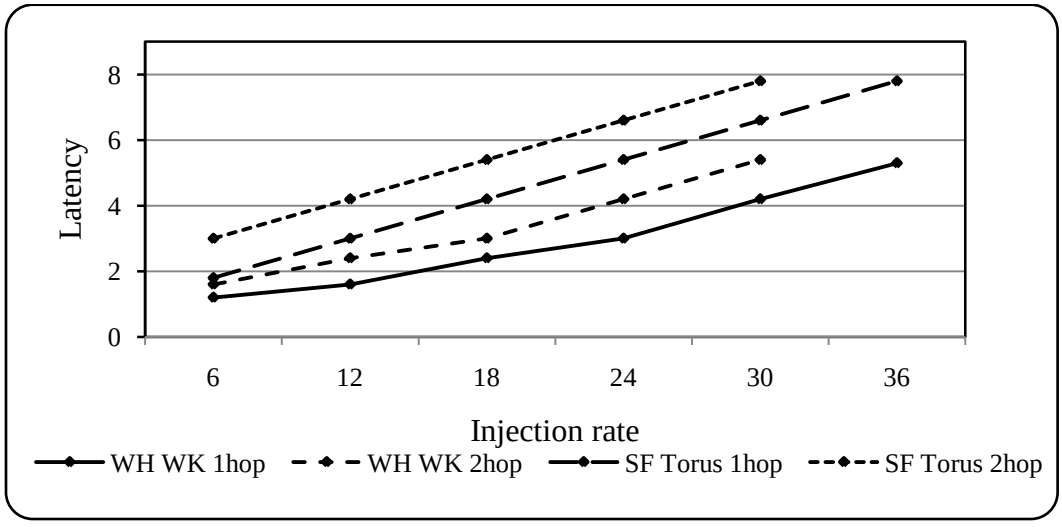


Figure 30. Injection Rate vs Latency for SF and WH under Uniform Traffic Conditions

WH routing also supports burst data transfer. A master sending multiple data to same destination can be packed into a single packet in WH, hence further reducing latency. Figure 31 shows the latencies for WH and SF, for hotspot traffic. The same will represent the congestion scenarios.

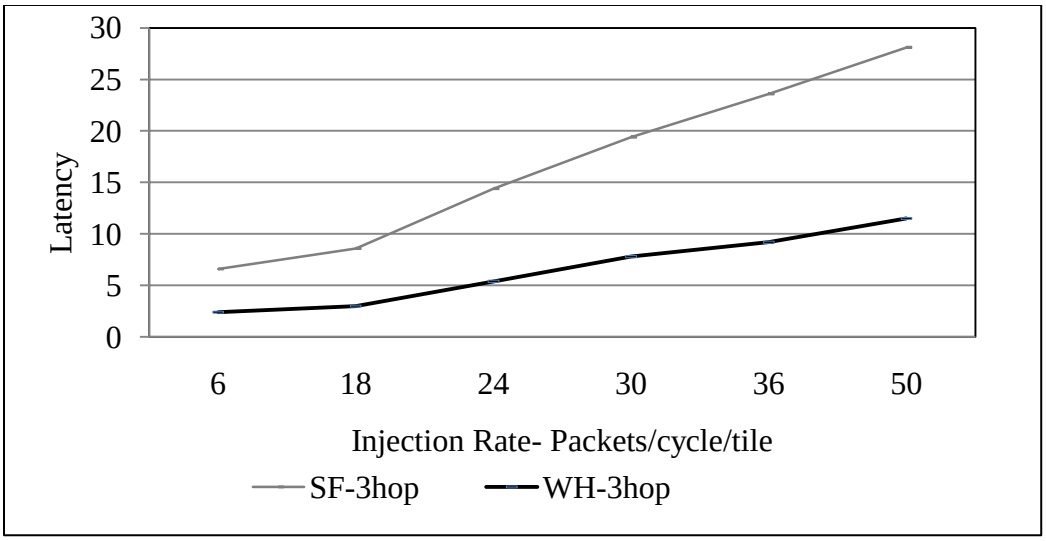


Figure 31. Injection Rate vs Latency for SF and WH in hotspot Traffic senario

6.2. SF Switching vs SF with LoopBack VCs

To evaluate the performance of SF with Loopback VCs, a hotspot scenario is developed. The traffic generator from a fast node M0 tends to send packets to a slow slave node S0 and another fast slave node S1. The packets saved in r1 due to slow PE, S0, results in congestion, in route r0r1. In normal SF routing with only a single buffer per Physical Channel, this results in head on blocking for packets routed to S1. The packets to S1 has to wait until all the packets to S0 is received.

In SF with 4VCs, packets to S0 will be stored in Virtual Channel VC3, where as packets to S1 will be stored in VC1. This avoids head on blocking. The packets to S1 will be routed as soon as the it arrives. The packet just needs to wait until its turn in (Round Robin) RR Switch Allocator.

The system configuration used to create the congestion scenario is shown in Figure 32. Master PE (M0) sending packets to a slow destination (S0) and to another faster node S2) creates congestion in route R0->R1.

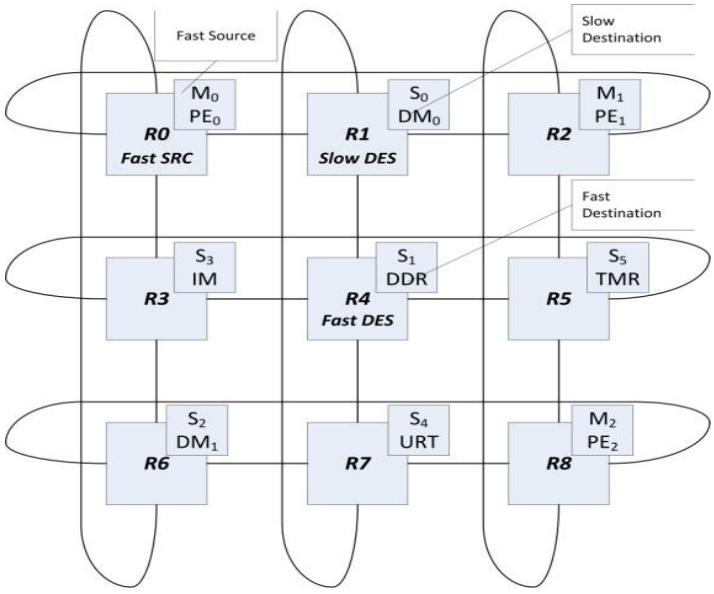


Figure 32. Torus Structure with Congestion Scenario

In SF with Loopback, there are only two VCs. When the packet to IP in r1 find a congested path, the packets are looped back until it finds a packet to a different route, ie to S1 and route them to south.

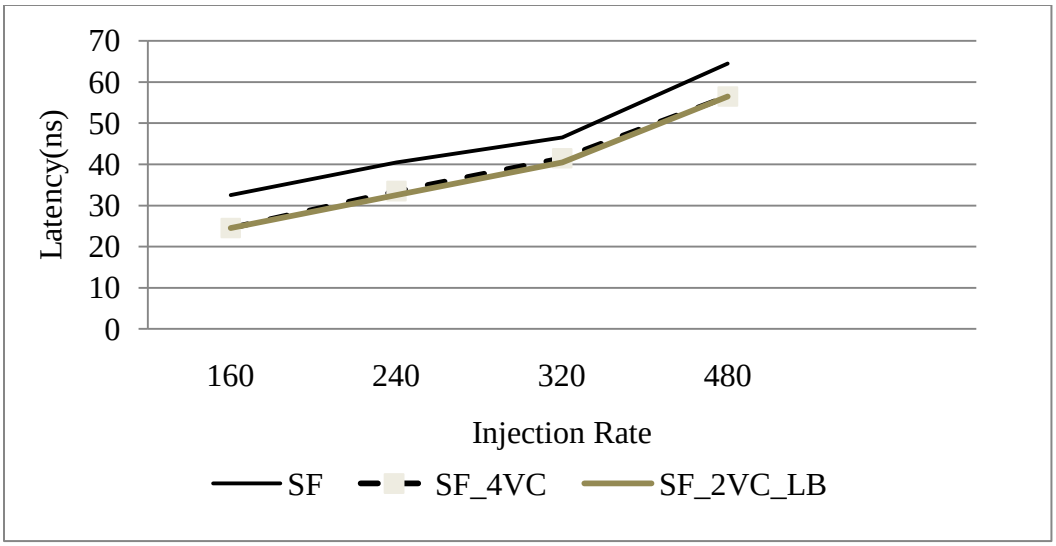


Figure 33. Injection Rate vs Latency for normal SF, SF with 4VC and SF with Loopback VC

Simulation Results (Trend graph) (Figure 33) shows a clear advantage of SF with 4VC and SF with loopback VC over normal SF. SF with 4VC follows closely SF with Loopback VCs.

6.3. Mirror Routing

To verify the adaptive Mirror Routing, we considered the scenario where the master M0 sends a message to Slave S0. In normal situations when all the links are free/not congested, Network Adapter at M0 would choose the path r0r1, which is the shortest feasible path. Now we send an information to the monitor that the link r0r1 is congested. In normal SF, the node r0 will wait until r0r1 is available whereas in SF with Mirror Routing, the NA at M0 chooses an alternative route for the message, ie through r0 -> r2 -> r1. A similar scenario is shown in Figure 20. Latency of packets using normal SF with

XY-routing and Mirror Routing for 3x3 and 4x4 torus are shown in graph below (see Figure 34).

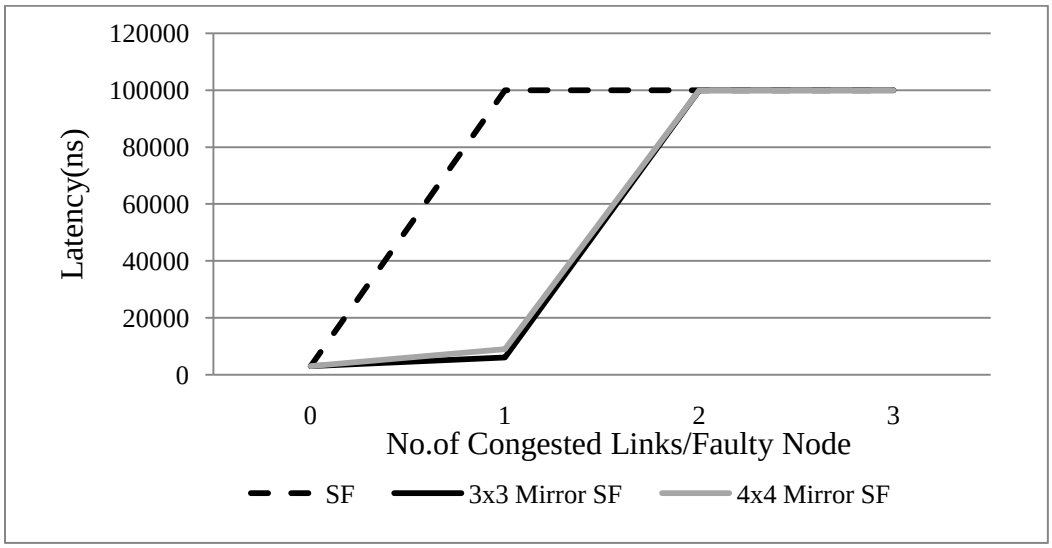


Figure 34. Congested/Faulty Link vs Latency

Mirror routing can afford single failure in the route.

6.4. Adaptive WormHole Routing for WK Topology

Adaptive Routing Algorithm for WK is verified for various hops and congestion scenarios. The Min and Max scenarios are designed and latency is calculated. In Min, the congested link is assumed to be the one which results in the minimum number of extra hops and hence minimum latency and for Max, the congested link results in a larger reroute. A worst case scenario where in the packets can choose a much larger route is included for reference.

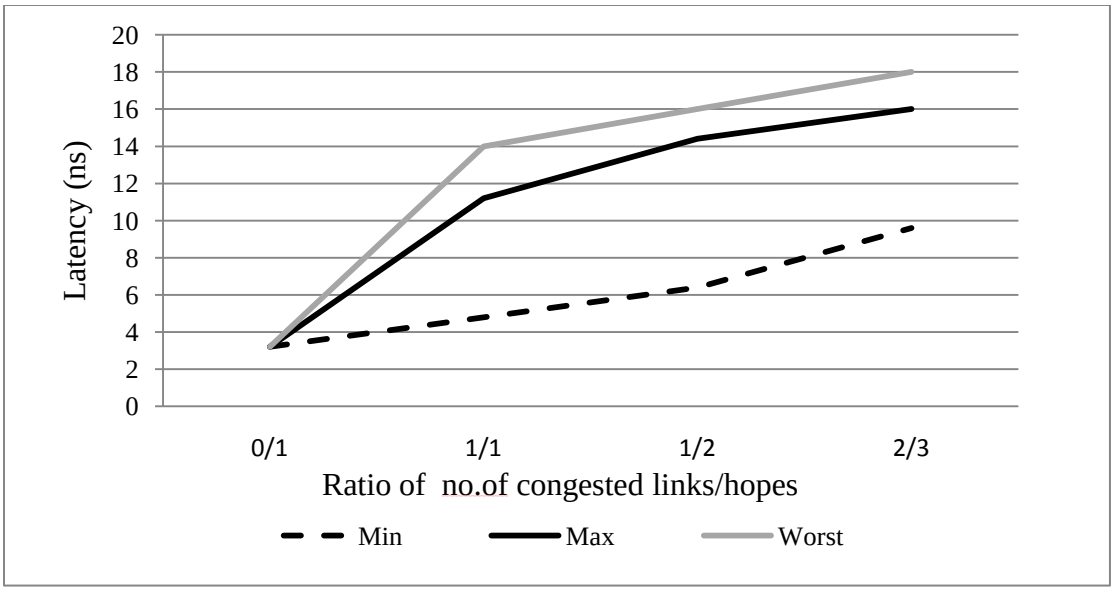


Figure 35. Latency vs Congestion hopes - Adaptive Routing

6.5 Synthesis Results

The design was synthesized for Virtex4 using Xilinx ISE. Table below gives the number of Slices and total on chip power for different modules in design for WH, SF, SF with 4 Virtual Channels and SF with loopback Virtual Channels.

Table 4. Synthesis Results

Block/Module	WH	SF	SF with 4VC	SF with Lp VC
Router (Channel width =22bits)	3330	933	3019	1334
Core Interface	32	40	40	40
Network Interface	34	34	34	34

The synthesis results proves that SF has lesser routing area compared to WH with VCs, by around 71%, whereas the SF with VCs has an area overhead of only 9.3% compared to WH switching router. SF switching with loopback VCs on other hand has considerable area improvement of around 59% compared to WH switching. The slice utilized for various PEs used in the design in Virtex4 is given in Table 5.

Table 5. No.of Occupied silices in Virtex4

Processing Element	No.of Occupied Slices in Virtex4
OR1K	2605
Timer	159
Uart	445

Xilinx Power Analyzer is used to calculate and compare the static and dynamic power of different modules in NoC. Table 6 shows the static power for different modules in different router configurations.

Table 6 Total Static on Chip Power from XPA

Block/Module	WH	SF	SF with 4VC	SF with Lp VC
Router	0.202	0.190	0.192	0.224
Core Interface	0.197	0.205	0.205	0.205
Network Interface	0.167	0.167	0.167	0.167

Table 7 shows the on-chip power for different router architectures, core and network interface modules for SF and WH switching modes.

Table 7 Total on Chip Power from XPA

Block/Module	WH	SF	SF with 4VC	SF with Lp VC
Router	0.202	0.190	0.192	0.224
Core Interface	0.197	0.205	0.205	0.205
Network Interface	0.167	0.167	0.167	0.167

The power analysis proves that the SF router architecture with loopback VCs utilizes 49% more power than SF and 33% more compared to router using WH switching technique with VCs.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Conclusions

A Synthesizable, Parametrizable, NoC Framework is designed and comparative studies where carried out. The NoC can choose 1) different Switching Policies – SF and WH 2) Different topologies – Torus, Mesh and WK-recursive 3) Choose or Modify the clock Frequency of Processing Elements 4) Very the packet format, size and type 5) Choose /Modify the Channel FIFO depth and width. The Traffic Generator can be used to generate different traffic scenarios, congestion scenarios and faulty node/links.

The NoC Framework can be used to study different routing algorithms with few changes. It can be used to study how well different Switching techniques are suitable for any particular application program. The Framework can be used to determine the optimal depth and size of buffers for any particular routing algorithm.

The newly introduced method of VC – loopback VC has proven better latency compared to normal SF or SF with 4 VCs. Mirror Routing algorithm proposed can handle single fault/ congestion in the routing path. The algorithm is simple and very easy to apply. The latency increases with increase in size of Torus structure. The Adaptive routing algorithm proposed for WK Topology results in increase in latency but can be considered in scenarios where the receiver node at the congested link is comparatively slow or when the fault in link is permanent.

7.2 Future Work

Network-on-chip is a very active research field with many practical applications in industry. The following topics are especially crucial for continued development and success of NoC paradigm: procedures and test cases for benchmarking, traffic characterization and modeling, design automation, latency and power minimization, fault-tolerance, QoS policies, prototyping, and network interface design. Future work includes, study and design of Asynchronous NoC Networks and compare their efficiency with present Synchronous design. Dynamically Reconfigurable NoC is also gaining importance these days.

REFERENCES

[1] OpenRisc 1000 architecture Manual from opencores.com

[2] “Interconnection Networks” (The Morgan Kaufmann Series in Computer Architecture and Design) by Jose Duato, SudhakarYalamanchili, Lionel NI.

[3] Young Jin Yoon, Nicola Concer, Michele Petracca, Luca Carloni “Virtual channels vs multiple physical networks: a comparative analysis” IEEE Proceedings of 47th Design Automotion Conference, Pages 162-165, 2010

[4] H. T. Salminen E., Kulmala A., “Survey on Network on chip Proposals” White Paper OCP-IP, March 2008.

[5] http://www.gaisler.com/doc/Evaluation_of_synthesizable_CPU_cores.pdf

[6] N. Genko, D. Atienza, G. De Micheli, “A Novel Approach for Network on Chip Emulation” Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium , May 2005, pp 2365 - 2368 Vol. 3

[7] C. Seitz, “Let’s Route Packets Instead of Wires”, Advanced Research in VLSI: Proceedings of the Sixth MIT Conference, pp. 133-138, 1990.

[8] Ni, L. M.; McKinley, P. K. A Survey of Wormhole Routing Techniques in Direct Networks. IEEE Computer Magazine, v.26(2), February, 1993, pp. 62-76.

[9] Liang, J.; Swaminathan, S.; Tessier, R. aSOC: “A Scalable, Single-Chip communications Architecture”, IEEE International Conference on Parallel Architectures and Compilation Techniques, Oct. 2000, pp. 37-46.

[10] Krishnan Srinivasan, Karam S. Chatha, and Goran Konjevod “Linear-Programming-Based Techniques for Synthesis of Network-on-Chip Architectures”, ICCD 2004. Proceedings. IEEE International Conference, Oct. 2004, pp 422 - 429

[11] W. J. Dally and B. Towles, “Route Packets, Not Wires: On-Chip Interconnection Networks”, ACM/IEEE Design Automation Conf., pp. 684-689, June 2001.

[12] S. Kumar, A. Jantsch, M. Millberg, J. Oberg, J. P. Soininen, M. Forsell, K. Tiensyrja and A. Hemani, “A Network on Chip Architecture and Design Methodology”, IEEE Symp. on VLSI, pp. 117-124, April 2002.

[13] J. Duato, et. al. “Interconnection Networks: An Engineering Approach”. Morgan Kaufmann, 2002.

[14] C. J. Glass, L. M. Ni. “The turn model for adaptive routing”. In Proc. ISCA, 1992.

[15] J. Hu, R. Marculescu. “DyAD-Smart routing for Networks-on-Chip”. In Proc. DAC, June 2004.

[16] S. Toueg and K. Steiglitz, “Some Complexity Results in the Design of Deadlock-Free Packet Switching Networks,” SIAM Journal on Computing, Vol. 10, No. 4, pp. 702-712, November 1981.

[17] Liang, J.; Swaminathan, S.; Tessier, R. aSOC: “A Scalable, Single-Chip communications Architecture. In: IEEE International Conference on Parallel Architectures and Compilation Techniques”, Oct. 2000, pp. 37-46.

[18] Ni, L. M.; McKinley, P. K. “A Survey of Wormhole Routing Techniques in Direct Networks”. IEEE Computer Magazine, v.26(2), February, 1993, pp. 62-76.

[19] S. Toueg and J. Ullman, Deadlock-Free Packet Switching Networks, Proc. 11th Annual ACM Symp. Theory of Computing, pp. 89-98, 1979.

[20] K. D. Gunther, Prevention of Deadlocks in Packet-Switched Data Transport Systems, IEEE Trans. Communications, Vol. COM-29, No. 4, pp. 512-524, April 1981.
[21] I. Gopal, Prevention of Store-and-Forward Deadlock in Computer Networks, IEEE Trans. Communications, Vol. COM-33, No. 12, pp. 1258-1264, December 1985.
[22] G. Varatkar and R. Marculescu , "On-chip traffic modeling and synthesis for MPEG-2 video applications" Very Large Scale Integration (VLSI) Systems, IEEE Transactions ,Jan. 2004, Vol 12, pp 108 – 119
[23] V. Soteriou, H-S . Wang, and L. Peh, "A statistical traffic model for on chip interconnection networks," Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium, Sept. 2006,pp 104 - 116
[24] A. Scherrer, A. Fraboulet, and T. Risset, "Automatic phase detection for stochastic on-chip traffic generation", Hardware/Software Codesign and System Synthesis, 2006. CODES+ISSS '06. Proceedings of the 4th International Conference, Oct. 2006, pp 88 - 93
[25] Y. Hu, H.Chen, Y.Zhu, A. A. Chien and C. Cheng, "Physical Synthesis of Energy-Efficient Network-on-Chip Through Topology Exploration and Wire Style Optimizations," Design (ICCD), pp.111-118, 2005.
[26] K. Srinivasan and K.S. Chatha , "A technique for low energy mapping and routing in network on chip architectures", Low Power Electronics and Design, 2005. ISLPED '05. Proceedings of the 2005 International Symposium, Aug. 2005, pp 387 - 392
[27] G. D. Vecchia and C. Sanges, "A recursively scalable network VLSI implementation," Future Generation Computer Systems, 4(3) 235-243, 1988.
[28] D.Rahmati, A.Kiasari, S.Hessabi, H.Sarbazi-Azad, "A Performance and Power Analysis of WK-Recursive and Mesh Networks for Network-on-Chips", Computer Design, 2006. ICCD 2006. International Conference on Oct. 2007, pp 142 - 147
[29] R.Wilhelm "Parallel and Distributed Programming", Advanced lecture, Summer term 2005.
[30] Mahmoud Moadeli, Ali Shahrabi, Wim Vanderbauwhede, Mohamed Ould-Khaoua "An Analytical Performance Model for the Spidergon NOC", Advanced Information Networking and Applications, 2007. AINA '07. 21st International Conference, May 2007, pp 1014 - 1021
[31] WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores Revision: B.3, Released: September 7, 2002
[32] N. Genko, D. Atienza, G. De Micheli, J. M. Mendias, R. Hermida, F. Catthoor, "A Complete Network-On-Chip Emulation Framework", Proc. Design, Automation and Test in Europe - DATE, pp. 246-251, 2005.
[33] A. Scherrer, A. Fraboulet, and T. Risset. Automatic phase detection for stochastic on-chip traffic generation. In Hardware/Software Codesign and System Synthesis, 2006. CODES+ISSS '06. Proceedings of the 4th International Conference, pages 88 {93, 22-25 2006.
[34] Johanna Sepulveda, Marrius Strum , Wand Jiang Chau, Ricardo Pires " The LRD traffic impact on the NoC based SoCs Pages 97-102.
[35] P. Poplavko, T. Basten, M. Bekooij, J. van Meerbergen, and B. Mesman, "Task-Level Timing Models for Guaranteed Performance in Multiprocessor Networks-on-

Chip," Proc. Int'l Conf. Compilers, Architectures and Synthesis for Embedded Systems (CASES), pp. 63-72, 2003.
[36] M.H. Ghadiry, M.Nadi, M.T. Manzuri-Shalmani, D.Rahmati, "Performance and Power analysis of Routing Algorithms on NOC" 2008
[37] S. Mahadevan, K. Virk, and J. Madsen, "Arts: A systemc-based framework for modelling multiprocessor systems-on-chip," Design Automation of Embedded Systems, 2006.
[38] M. Coppola, S. Curaba, M. D. Grammatikakis, R. Locatelli, G. Maruccia, and F. Papariello, "Occn: a noc modeling framework for design exploration," Journal of Systems Architecture, vol. 50, no. 2-3, pp. 129– 163, 2004.
[39] S. S. K. A. Mehran and Armin, "Smap: An intelligent mapping tool for network on chip," Signals, Circuits and Systems ISSCS 2007, pp. 1–4, 13-14 July 2007.
[40] S. Murali and G. D. Micheli, "Sunmap: a tool for automatic topology selection and generation for nocs," in DAC '04: Proceedings of the 41st annual conference on Design automation. New York, NY, USA: ACM, 2004, pp. 914–919.
[41] A. Jalabert, S. Murali, L. Benini, and G. D. Micheli, "xpipesCompiler: A tool for instantiating application specific Networks on Chip,", Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings, Feb. 2004 pp 884 - 889 Vol.2
[42] J. Joven, O. Font-Bach, D. Castells-Rufas, R. Martinez, L. Teres, and J. Carrabina, "xenoc - an experimental network-on-chip environment for parallel distributed computing on noc-based mpsoc architectures." In PDP. IEEE Computer Society, 2008, pp. 141–148.
[43] D. Wiklund, S. Sathe, and D. Liu. Network on chip simulations for benchmarking. In System-on-Chip for Real-Time Applications, 2004.Proceedings. 4th IEEE International Workshop on, pages 269 { 274, 19-21 2004.
[44] G. Varatkar and R. Marculescu. On-chip traffic modeling and synthesis for mpeg-2 video applications. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 12(1):108 119, jan. 2004.
[45] A. Scherrer, A. Fraboulet, and T. Risset. Long-range dependence and on-chip processor traffic. Microprocessors and Microsystems, 33(1):72 { 80, 2009. Selected Papers from ReCoSoC 2007 (Recon_gurable Communication-centric Systems-on-Chip).
[46] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. Qnoc: Qos architecture and design process for network on chip. J. Syst. Archit., 50(2-3):105{128, 2004.
[47] A. Feldmann, A. C. Gilbert, W. Willinger, and T. G.Kurtz. The changing nature of network tra_c: scalingphenomena. SIGCOMM Comput. Commun. Rev28(2):5{29, 1998.
[48] P. Abry and D. Veitch. Wavelet analysis of long-range-dependent tra_c. Information Theory, IEEE Transactions on, 44(1):2 {15, jan 1998.
[49] K. Lahiri, A. Raghunathan, and S. Dey. Evaluation of the traffic performance characteristics of system-on-chip communication architectures. In VLSI Design, 2001. Fourteenth International Conference on, pages 29 {35, 2001.
[50] Mello, A.; Copello Ost, L.; Gehm Moraes, O.; Laert, N.; Calazans, V. "Evaluation of Routing Algorithms on Mesh Based NOCs", Faculty of Informatics, Pontifícia Universidade Católica do Rio Grande do Sul, Brasil, Technical Report Series, No. 040, May 2004.

[51] W.J.Dally “Virutal Channel Flow Control” , IEEE Trans. On Parallel and Distributed Systems, Vol 3, No. 2, March 1992, pp 194.
[52] W.J.Dally and C.L.Seitz “Deadlock Free Message Routing in Multiprocessor Interconnection Networks” , IEEE Trans, on Computers, Vol C May 1987, pp 547-553
[53] W. J. Dally and B. Towles, “Principles and practices of interconnection networks,” Morgan Kaufmann, 2004.
[55] Mohsen Saneei, Ali Afzali-Kusha, Zainalabedin Navabi, “Low Latency Multi Leel Lesh Topology for NOCs” in 18th International Conference on Microelectronics (ICM) 2006 p36-39
[56] Kwanho Kim, Donghyun Kim, Kangmin Lee and Hoi Jun Yoo “Cost efficient Network-on-Chip Design Using Traffic Monitoring System” IEEE Design, Automation and Test in Europe,
[57] “Heng Yu, Yajun Ha, Bharadwaj Veeravalli “Communication-Aware Application Mapping and Scheduling for NoC Based MPSoCs”, Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on, June 2010, pp 3232 - 3235
[58] Guangyu Chen, Feihui Li, Mahmut Kandemir “Compiler Directed Application Mapping for NoC Based Chip Multiprocessors” Proceedings of the 2007 ACM SIGPLAN/SIGBED, pp 155 - 157
[59] Hsin-Chou Chi, Chia-Ming Wu, and Jun-Hui Lee, "Integrated Mapping and Scheduling for Circuit-Switched Network-on-Chip Architectures," IEEE International Symposium on Electronic Design, Test and Applications, Hong Kong, Jan. 2008.
[60] Ashwini Raina, V.Muthukumar “Traffic Aware Scheduling Algorithm for Network on Chip”, Information Technology: New Generations, 2009. ITNG '09. Sixth International Conference, April 2009, pp 877 - 882
[61] Jingcao Hu, Radu Marculescu “Communication and Task Scheduling of Application-Specific Neworks-on-Chip”, Computers and Digital Techniques, IEE Proceedings, Sept. 2005 Vol 152, Issue:5, pp 643 - 651
[62] G. Chen, F. Li, S.W. Son, and M. Kandemir, “Application mapping for chip multiprocessors,” IEEE Proc. DAC, pp. 620-625, June 2008.
[63] S. Murali and G. De Micheli, “Bandwidth-constrained mapping of cores onto NoC architectures,” Proc. DATE, pp. 896-901, Feb 2004.
[64] J, Hu and R. Marculescu, “Energy-aware communication and task scheduling for Network-on-Chip Architectures under Real-Time Constraints,” Proc DATE, pp. 234-239, Feb 2004.
[65] W. J. Dally, “Virtual-channel flow control”, IEEE Transactions on Parallel and Distributed systems, vol. 3, no. 2, pp. 194-205, March, 1992
[66] L.-S. Peh and W. J. Dally. A Delay Model and Speculative Architecture for Pipelined Routers. In International Symposium on High-Performance Computer Architecture, pages 255–266, Jan 2001.
[67] N. Kavaldjiev, G. J. M. Smit, and P. G. Jansen, “A virtual channel router for on-chip networks,” Proceedings of the IEEE International SOC Conference, 2004, pp 289 - 293
[68] Sung-Tze Wu, Chih-Hao Chao, I-Chyn Wey, An-Yen Wu “Dynamic Channel Flow Control of Network-on-Chip Systems for High Buffer” Paper appears in Signal Processing Systems, 2007 IEEE Workshop on 17-19 Oct 2007 pp 493-498

[69] Min Zhang, Chiu-Sing Choy "Low-Cost VC Allocator Design for Virtual Channel Wormhole Routers in Networks-on-Chip" Proceeding NOCS -08 Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip 2008
[70] Robert Mullins , Andrew West , Simon Moore, Low-Latency Virtual-Channel Routers for On-Chip Networks, Proceedings of the 31st annual international symposium on Computer architecture, p.188, June 19-23, 2004, München, Germany
[71] Aline Mello, Leonel Tedesco, Ney Calazans, Fernando Moraes "Virtual channels in networks on chip: implementation and evaluation on hermes NoC" Proceedings of the 18th annual symposium on Integrated circuits and system design 2005 pp 178 - 183
[72] Daewook Kim, Manho Kim and Gerald E. Sobelman "CDMA-Based Network-on-chip architecture", Circuits and Systems, 2004. Proceedings, 2004 IEEE Asia-Pacific Conference, Dec. 2004, pp 137 - 140 vol.1
[75] Assaf Shacham, Keran Bergman, Luca P. Carloni "On the design of a Photonic Network on Chip", Networks-on-Chip, 2007. NOCS 2007. First International Symposium, May 2007, pp 53 - 64
[76] Song Chai, Chang Wu, Yubai Li, Zhongming Yang "A NOC Simulation & Verification Platform based on SystemC" 2008 International Conference on Computer Science and Software Engineering, December 12~14,
[77]Abdelkrim Zitouni, Mounir Zid, Sami Badrouchi and Rached Tourki "A Generic and Extensible Spidergon NOC", World Academy of Science, Engineering and Technology, Jul 2007, pp 14-19
[78] Sami Badrouchi, Abdelkrim Zitouni, Kholdoun Torki and Rached Tourki "Asynchronous NOC Router Design" Journel of Computer Science 1(3) pp 429-436, 2005
[79] Thuan Le and Mohammed A. S. Khalid "NOC Prototyping on FPGAs: A Case Study Using an Image Processing Benchmark" This paper appears in: Electro/Information Technology, 2009. eit'09. IEEE International Conference on June 2009
[80] Dominique Borrione, Amr Helmy, Julien Schmaltz "Executable Formal Specification and Validation of NOC Communication Infrastructures" Proceedings of the 21st annual symposium on Integrated circuits and system design, 2008, pp 176-181
[81] Chang Wu, Yubai Li, Song Chai, Zhongming Yang "Lottery Router : A Customized Arbitral Priority NOC Router", Computer Science and Software Engineering, 2008 International Conference, Dec. 2008, pp 411 - 414
[82] Gao Xiaopeng, Zhang Zhe, Long Xiang " Round Robin Arbiters for Virtual Channel Router" Computational Engineering in Systems Applications, IMACS Multiconference on Oct 2006, pp 1610 - 1614
[83] Mario P. Vestias and Horacio . C. Neto "Area/Performance improvement of NOC Architectures" Reconfigurable Computing : Architectures and Applications 2006,Volune 3985.2006 pp 193-198
[84] Nikolaj Dalgaard Torring " Multiprocessor in a FPGA" Thesis prepared at Informatics Mathematical Modelling, Technical University of Denmark. June 2007

APPENDIX

VERILOG CODES

Arbiter Module (WH routing)

```
module r11_noc_arbiter(
clk_i , rst_i , pop_north , pop_south , pop_east , pop_west, pop_ip, pop_across,
pop_over, fifo_north, fifo_south, fifo_east, fifo_ip, fifo_west, fifo_across, fifo_over,
send_north, send_south, send_west, send_east, send_across, send_over, count_north,
count_south, count_west, count_east, count_across, count_over, count_ip ,
connect_ip_to_across, connect_ip_to_west, connect_ip_to_south,
connect_south_to_across, connect_south_to_west, connect_south_to_ip,
connect_west_to_across, connect_west_to_south, connect_west_to_ip,
connect_across_to_south, connect_across_to_west, connect_across_to_ip );

`include "noc_top_defines.v"
input clk_i ; input rst_i ; output pop_north ; output pop_south; output pop_east;
output pop_west; output pop_across; output pop_over;
input [fifo_Addr-1:0] count_north, count_south, count_west, count_east;
input [fifo_Addr-1:0] count_across, count_over, count_ip ;
output pop_ip ;
input send_north, send_south, send_west, send_east, send_across, send_over;


input [NOC_DATA_WIDTH-1:0] fifo_north , fifo_south , fifo_across ;
input [NOC_DATA_WIDTH-1:0] fifo_over , fifo_east, fifo_west ;
input [NOC_DATA_WIDTH-1:0] fifo_ip ;

output connect_ip_to_across, connect_ip_to_west,connect_ip_to_south
connect_south_to_across, connect_south_to_west, connect_south_to_ip,;
output connect_west_to_across, connect_west_to_south, connect_west_to_ip;

output connect_across_to_south, connect_across_to_west, connect_across_to_ip;
reg connect_ip_to_across, connect_ip_to_south, connect_ip_to_west,
connect_south_to_across, connect_south_to_west, connect_south_to_ip,
connect_west_to_across, connect_west_to_south, connect_west_to_ip;
reg connect_across_to_south; reg connect_across_to_west; reg connect_across_to_ip;
reg pop_ip_to_across; reg pop_ip_to_west; reg pop_ip_to_south;

reg pop_across_to_west, pop_across_to_ip , pop_across_to_south, pop_west_to_across,
pop_west_to_ip, pop_west_to_south, pop_south_to_across, pop_south_to_ip;
reg pop_south_to_west;
assign pop_ip = pop_ip_to_across | pop_ip_to_west | pop_ip_to_south;
```

```
assign pop_across = pop_across_to_west |pop_across_to_south | pop_across_to_ip;
assign pop_west = pop_west_to_ip | pop_west_to_across | pop_west_to_south;
assign pop_south = pop_south_to_ip | pop_south_to_across | pop_south_to_west;

always @(rst_i) begin
if(rst_i == 1'b0)
begin
connect_ip_to_across <=1'b0; connect_ip_to_west <=1'b0; connect_ip_to_south <=1'b0;
connect_across_to_west<=1'b0;                    connect_across_to_south<=1'b0;
connect_across_to_ip<=1'b0;  connect_west_to_across<=1'b0;
connect_west_to_south<=1'b0; connect_west_to_ip<=1'b0;
connect_south_to_across<=1'b0; connect_south_to_west<=1'b0;
connect_south_to_ip<=1'b0; pop_across_to_west <=1'b0; pop_across_to_south <=1'b0;
pop_across_to_ip <=1'b0; pop_ip_to_across <=1'b0; pop_ip_to_west <=1'b0;
pop_ip_to_south <=1'b0;  pop_west_to_across <=1'b0; pop_west_to_south <=1'b0;
pop_west_to_ip <=1'b0; pop_south_to_across <=1'b0; pop_south_to_west <=1'b0;
pop_south_to_ip <=1'b0;
end
end

always @(connect_ip_to_across or connect_ip_to_west or connect_ip_to_south or
connect_across_to_ip or connect_across_to_south or connect_across_to_west or
connect_south_to_ip or connect_south_to_across or connect_south_to_west or
connect_west_to_ip or connect_west_to_south or connect_west_to_across)
begin
pop_ip_to_across = connect_ip_to_across;
pop_ip_to_west =connect_ip_to_west;
pop_ip_to_south =connect_ip_to_south;

pop_across_to_ip = connect_across_to_ip;
pop_across_to_south = connect_across_to_south;
pop_across_to_west = connect_across_to_west;
pop_south_to_west = connect_south_to_west;
pop_south_to_across = connect_south_to_across;
pop_south_to_ip = connect_south_to_ip;
pop_west_to_ip = connect_west_to_ip;
pop_west_to_across = connect_west_to_across;
pop_west_to_south = connect_west_to_south;
end

always @(posedge clk_i)
begin
if(fifo_ip[NOC_DATA_WIDTH-1 : NOC_DATA_WIDTH-2]== 2'b00)//HEADER
begin
```

```
if((fifo_ip[6:5]== 2'b10 || fifo_ip[6:3]==4'b0110) && send_across == 1'b1)
connect_ip_to_across <= 1'b1;
else if((fifo_ip[6:5]== 2'b11 || fifo_ip[6:3] == 4'b0111) && send_south ==1'b1)
connect_ip_to_south <= 1'b1;
else if((fifo_ip[6:5]== 2'b00 || fifo_ip[6:3] == 4'b0100) && send_west == 1'b1)
connect_ip_to_west <= 1'b1;
end
else if (fifo_ip[NOC_DATA_WIDTH-1 : NOC_DATA_WIDTH-2]== 2'b11) //TAIL
begin
connect_ip_to_south <=1'b0;
connect_ip_to_across <=1'b0;
connect_ip_to_west <=1'b0;
end
end

always @(posedge clk_i)
begin
if(fifo_across[NOC_DATA_WIDTH-1 : NOC_DATA_WIDTH-2]== 2'b00)//HEADER
begin
if((fifo_across[6:5]== 2'b11 || fifo_across[6:3] == 4'b0111 )&& connect_ip_to_south
!=1'b1 && send_south ==1'b1)
connect_across_to_south <= 1'b1;
else if((fifo_across[6:5]== 2'b00 || fifo_across[6:3] == 4'b0100 )&& connect_ip_to_west
!=1'b1 && send_west ==1'b1)
connect_across_to_west <= 1'b1;
else if(fifo_across[6:3] == 4'b0101 )
connect_across_to_ip <= 1'b1;
end
else if (fifo_across[NOC_DATA_WIDTH-1 : NOC_DATA_WIDTH-2]== 2'b11) //TAIL
begin
connect_across_to_south <=1'b0;
connect_across_to_west <=1'b0;
connect_across_to_ip <=1'b0;
end
end

always @(posedge clk_i)
begin
if(fifo_west[NOC_DATA_WIDTH-1 : NOC_DATA_WIDTH-2]== 2'b00)//HEADER
begin
if((fifo_west[6:5]== 2'b11 || fifo_west[6:3] == 4'b0111) && connect_ip_to_south !=1'b1
&& connect_across_to_south !=1'b1 && send_south==1'b1)
connect_west_to_south <= 1'b1;
```

```
else if((fifo_west[6:5]== 2'b10 ||fifo_west[6:3]==4'b0110) && connect_ip_to_across !=1'b1 && send_across==1'b1)
connect_west_to_across <= 1'b1;
else if(fifo_west[6:3] == 4'b0101 && connect_across_to_ip !=1'b1)
connect_west_to_ip <= 1'b1;
end
else if (fifo_west[NOC_DATA_WIDTH-1 : NOC_DATA_WIDTH-2]== 2'b11) //TAIL
begin
connect_west_to_south <=1'b0;
connect_west_to_across <=1'b0;
connect_west_to_ip <=1'b0;
end
end


always @(posedge clk_i)
begin
if(fifo_south[NOC_DATA_WIDTH-1 : NOC_DATA_WIDTH-2]== 2'b00)//HEADER
begin
if((fifo_south[6:5]== 2'b00 || fifo_south[6:3] == 4'b0100) && connect_ip_to_west !=1'b1 && connect_across_to_west !=1'b1 && send_west==1'b1)
connect_south_to_west <= 1'b1;
if((fifo_south[6:5]== 2'b10 ||fifo_south[6:3]==4'b0110) && connect_west_to_across !=1'b1 && connect_ip_to_across !=1'b1 && send_across==1'b1)
connect_south_to_across <= 1'b1;
if(fifo_south[6:3] == 4'b0101 && connect_west_to_ip !=1'b1 && connect_across_to_ip !=1'b1)
connect_south_to_ip <= 1'b1;
end
else if (fifo_south[NOC_DATA_WIDTH-1 : NOC_DATA_WIDTH-2]== 2'b11) //TAIL
begin
connect_south_to_across <=1'b0;
connect_south_to_west <=1'b0;
connect_south_to_ip <=1'b0;
end
end

endmodule
```

Router Switch Module

```
assign across_o= (connect_ip_to_across & select_ip_i)?ip_i:(connect_over_to_across & select_over_i)?over_i: (connect_north_to_across & select_north_i)?north_i:
```

```
(connect_south_to_across & select_south_i)?south_i: (connect_east_to_across & select_east_i)?east_i: (connect_west_to_across & select_west_i)?west_i:25'b0;

assign over_o = (connect_ip_to_over & select_ip_i)?ip_i:(connect_across_to_over & select_across_i)?across_i:(connect_north_to_over & select_north_i)?north_i:(connect_south_to_over & select_south_i)?south_i: (connect_east_to_over & select_east_i)?east_i:(connect_west_to_over & select_west_i)?west_i:25'b0;


assign north_o = (connect_ip_to_north & select_ip_i)?ip_i:(connect_across_to_north & select_across_i)?across_i: (connect_over_to_north & select_over_i)?over_i:(connect_south_to_north & select_south_i)?south_i: (connect_east_to_north & select_east_i)?east_i:(connect_west_to_north & select_west_i)?west_i:25'b0;

assign south_o = (connect_ip_to_south & select_ip_i)?ip_i:(connect_across_to_south & select_across_i)?across_i: (connect_over_to_south & select_over_i)?over_i:(connect_north_to_south & select_north_i)?north_i: (connect_east_to_south & select_east_i)?east_i:(connect_west_to_south & select_west_i)?west_i:25'b0;

assign east_o = (connect_ip_to_east & select_ip_i)?ip_i:(connect_across_to_east & select_across_i)?across_i: (connect_over_to_east & select_over_i)?over_i:(connect_north_to_east & select_north_i)?north_i: (connect_south_to_east & select_south_i)?south_i:(connect_west_to_east & select_west_i)?west_i:25'b0;

assign west_o = (connect_ip_to_west & select_ip_i)?ip_i:(connect_across_to_west & select_across_i)?across_i: (connect_over_to_west & select_over_i)?over_i:(connect_north_to_west & select_north_i)?north_i: (connect_east_to_west & select_east_i)?east_i:(connect_south_to_west & select_south_i)?south_i:25'b0;


assign ip_o = (connect_west_to_ip & select_west_i)?west_i:(connect_across_to_ip & select_across_i)?across_i: (connect_over_to_ip & select_over_i)?over_i:(connect_north_to_ip & select_north_i)?north_i: (connect_east_to_ip & select_east_i)?east_i:(connect_south_to_ip & select_south_i)?south_i:25'b0;
```

VC FIFO Module

```
module VC1_fifo(
 clk ,  rst ,  data_i ,  data_o ,  push ,  pop,  count );
`include "noc_top_defines.v"

 input clk ;  input rst ;
 input [NOC_DATA_WIDTH-1:0]data_i ;
 output [NOC_DATA_WIDTH-1:0]data_o ;
 input push ;  input pop ;  output [fifo_Addr-1:0]count ;
  wire  [fifo_Addr-1:0]count ;  reg [NOC_DATA_WIDTH-1:0]mem[fifo_size-1:0];
 reg [fifo_vector:0]top ;  reg [fifo_vector:0]bottom ;  reg [fifo_vector:0]counter ;
 assign count = counter ;  assign data_o = mem[bottom];
 always @(posedge clk or negedge rst)  begin
 if (rst == 1'b0)
 begin
 top <= 0;
 bottom <= 0;
 counter <= 0; end
 else     begin
                    if(push == 1'b1)
                    begin
                    top <= top + one ;
                    mem[top] <= data_i ;
                    if(pop != 1'b1)
                    begin
                    counter <= counter + one;
                    end
                    end
                    if(pop == 1'b1)
                    begin
                    mem[bottom] <=80'hx;
                    bottom <=  bottom  + one ;
                    if( push != 1'b1)
                    begin
                    counter <= counter - one ;
                    end
                    end
          end
 end
 endmodule
```

VITA

Graduate College
University of Nevada, Las Vegas

Jaya Suseela

Degree:

Bachelor of Technology, 2001
Kerala University, India

Master of Technology in VLSI Design, 2005
Amrita Vishwa Vidya Peetham, India

Special Honors and Awards:

Member, Tau Beta Phi

Thesis Title: Parameterizable Network-on-Chip Emulation Framework

Thesis Examination Committee:

Chairperson, Dr. Venkatesan Muthukumar, Ph. D.
Committee Member, Dr. Emma Regentova, Ph. D.
Committee Member, Dr. Mei Yang, Ph. D.
Graduate Faculty Representative, Dr. Ajoy Datta, Ph. D.